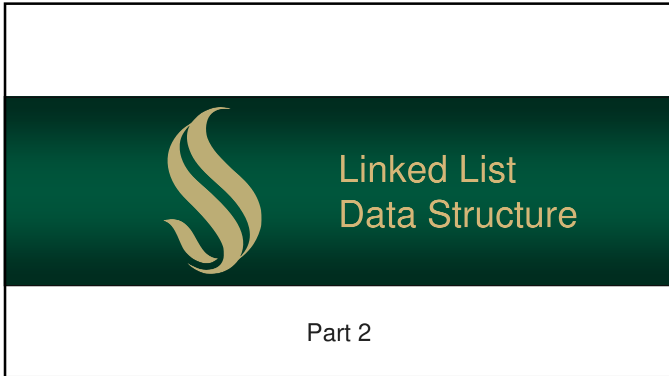
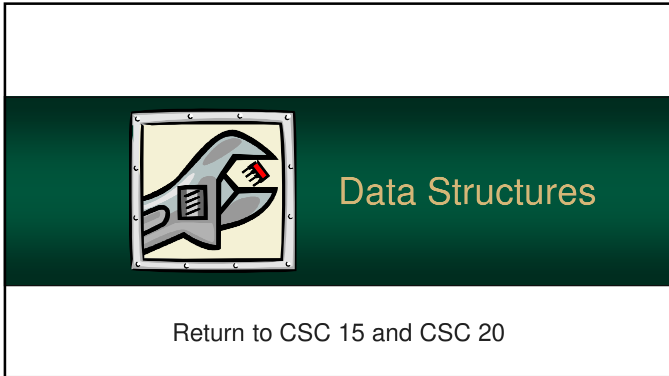
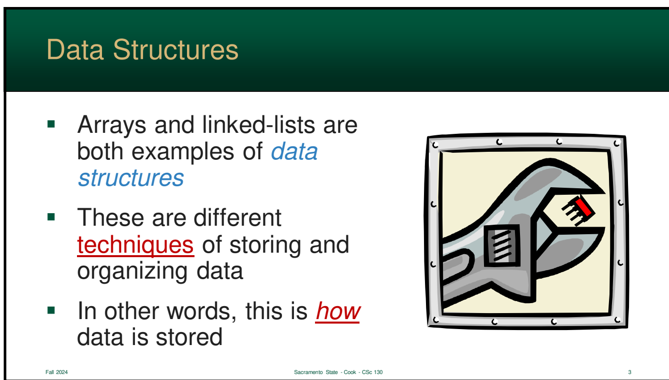


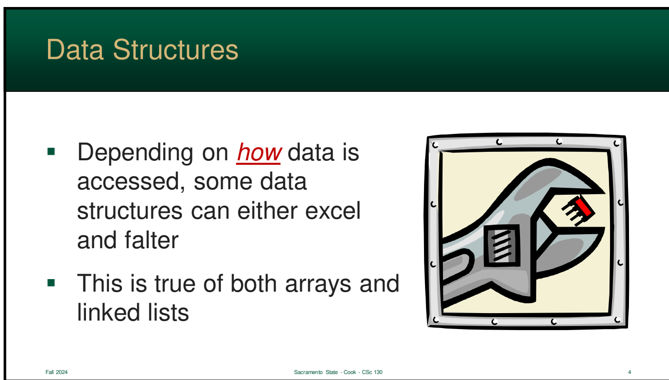
1



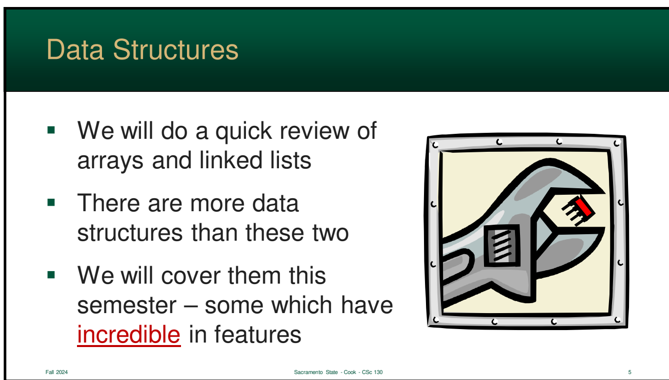
2



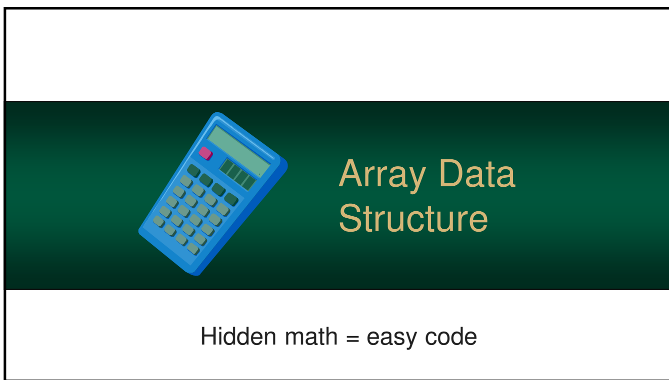
3



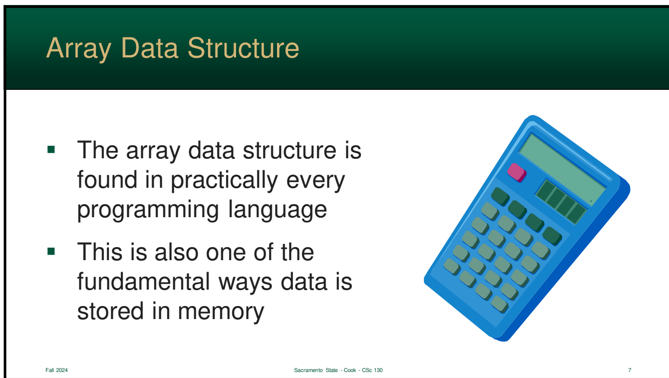
4



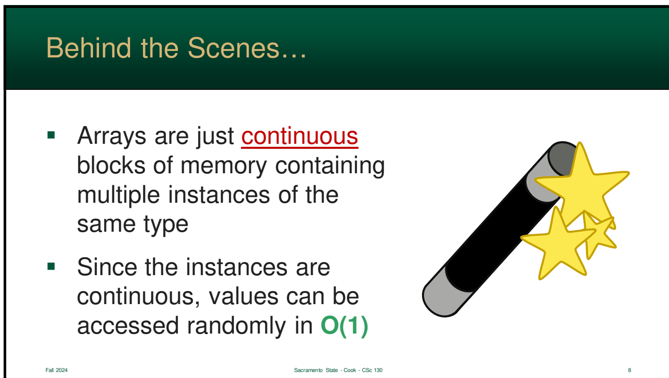
5



6



7



8

Array Math Example: 64-bit int

- Let's assume the array starts at address 2000
- Each array element will take 8 bytes (for 64-bit integers)
- Array elements are stored continuous

Address	Value
2000	446576696E20436F
2008	6F6B000000000000
2016	53616372616D656E
2024	746F205374617465
2032	4353419645000000

9

Array Math Example: 64-bit int

- `array[0]` is 2000
- `array[1]` is 2008
- `array[2]` is 2016
- `array[3]` is 2024
- `array[4]` is 2032
- etc…

Address	Value
2000	446576696E20436F
2008	6F6B000000000000
2016	53616372616D656E
2024	746F205374617465
2032	4353419645000000

10

Behind the Scenes…

- So, when an array element is read, internally, a mathematical equation is used
- It uses the start array, the array index, and the size of each element

```
start + (index × element_size)
```

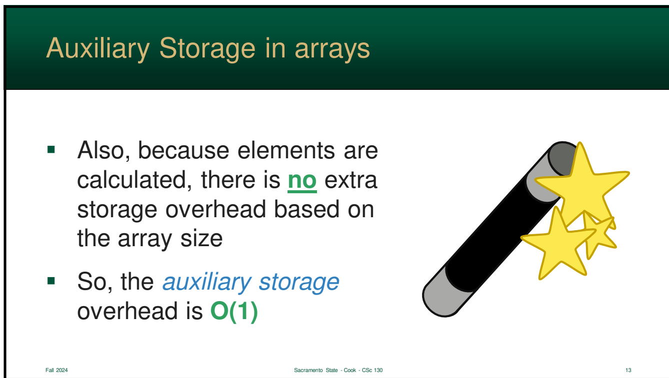

11

Behind the Scenes…

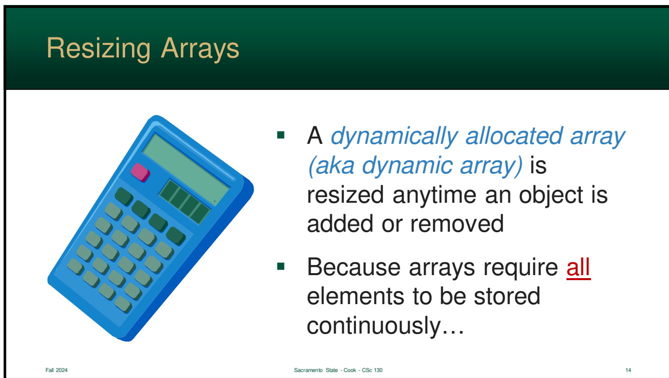
- *This is why the C Programming Language uses zero as the first array element*
- If zero is used with this formula, it gets the start of the array

```
start + (index × element_size)
```

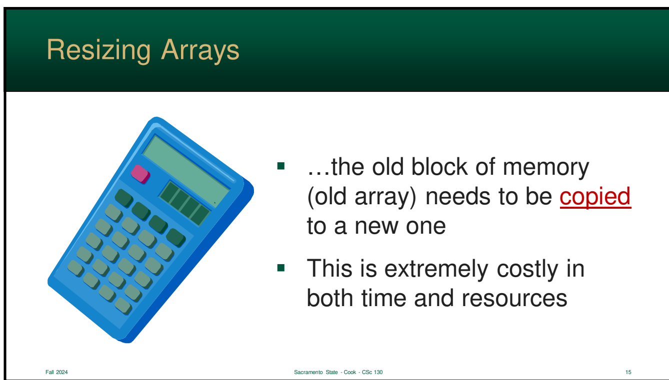

12



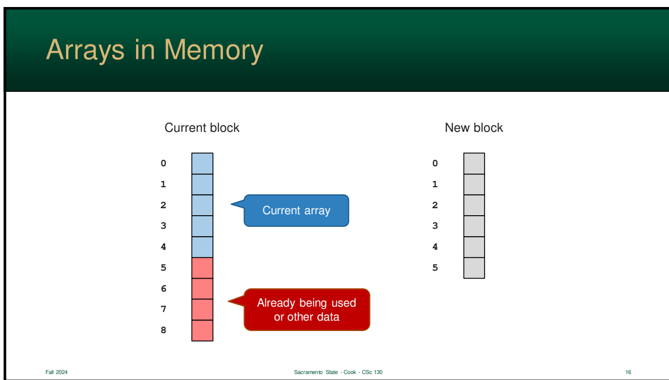
13



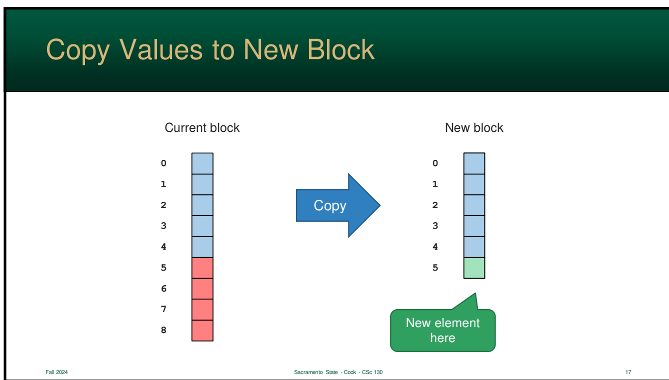
14



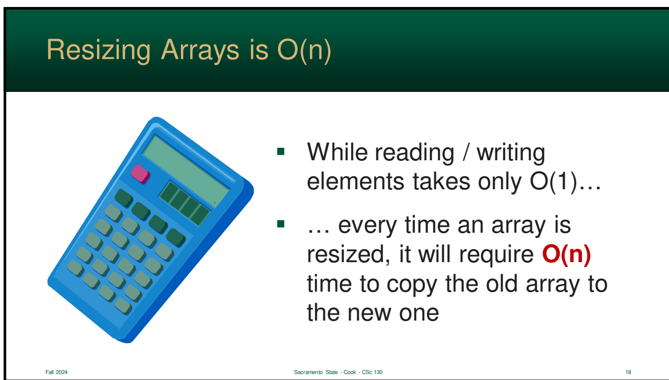
15



16



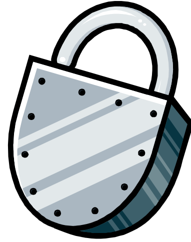
17



18

Fixed-Sized Arrays

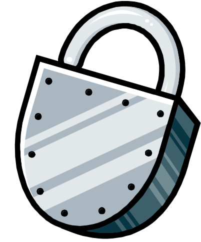
- Arrays can also have a fixed sized called a *capacity*
- The array is never resized and often only partially filled
- Also known as:
  - *fixed array*
  - *partially filled array*



19

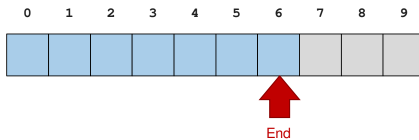
Fixed-Sized Arrays

- An "end" index is maintained
- This type of array overcomes the O(n) nature of dynamic arrays
- But a cost – it has a limit that cannot be exceeded



20

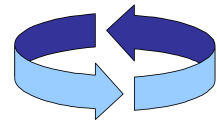
Fixed-Size Array



21

Fixed-Size Wrapping Around

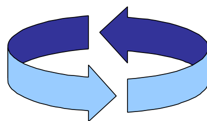
- Sometimes, you might need an array that wraps
- These are useful if both the first and last items can be removed
- … or older items can be discarded if space is needed



22

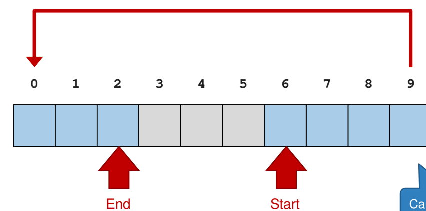
Fixed-Size Wrapping Around

- In addition to a "end" index, a "start" index is maintained
- Once the end of the array is reached, the array "wraps" to index 0
- … and continues until end is reached



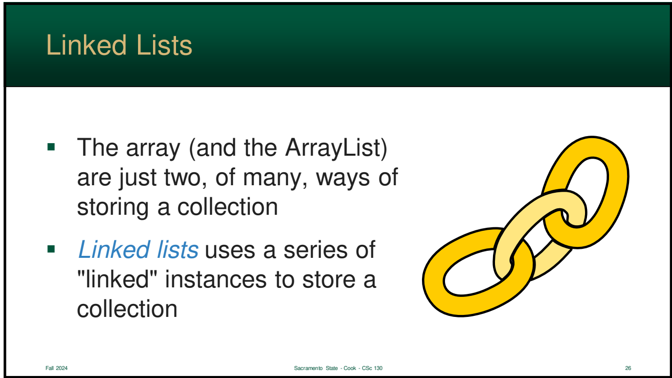
23

Fixed-Size Array



24

25



26

How Does It Work?

- Since a variable can contain either an instance reference or null, we can something quite clever
- An instance can contain a reference to another instance – *of the same class*
- This creates a chain of connected instances.
- It ends when the "link" is null

27

How Does That Work?

```
class Book
{
    public String name;
    public Book sequel;
}
```

A reference to another Book instance

28

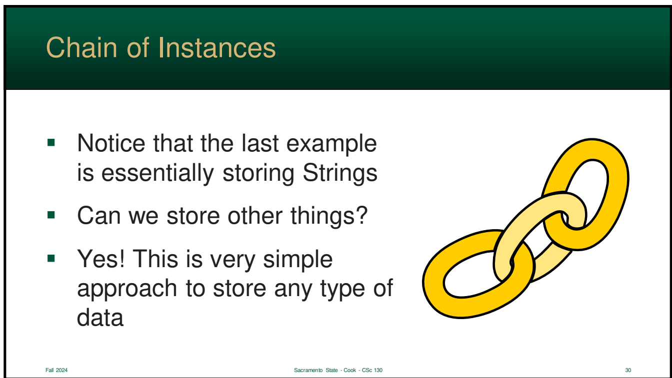
Chain of Books

name	sequel
The Fellowship of the Ring	→

name	sequel
The Two Towers	→

name	sequel
Return of the King	null

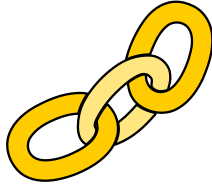
29



30

Chain of Instances

- Each link in our chain, that stores a piece of information, is called a *Node*
- The definition of a Node is extremely simple: data and a link to the next node



31

Generic Node Class

```
public class Node
{
    public Object data;
    public Node next;
}
```

We can make this another type

32

Creating a List (not well, though)

```
Node list = new Node();

list.data = "rat";
list.next = new Node();

list.next.data = "owl";
list.next.next = new Node();

list.next.next.data = "cat";
list.next.next.next = null;
```


33

Constructors Will Help

```
public Node(Object initData, Node initNext)
{
    this.data = initData;
    this.next = initNext;
}

public Node(Object initData)
{
    this.data = initData;
    this.next = null;
}
```


34

Constructors Do Help

- Constructors do help
- Though it is still a tad hard to read

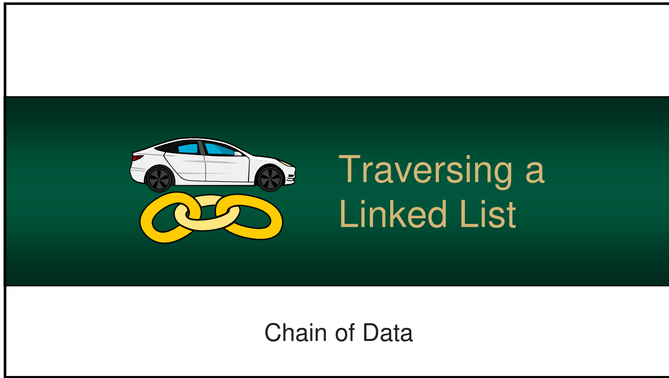
```
Node list = new Node("Rat", new Node("Owl", new Node("Cat")));
```


35

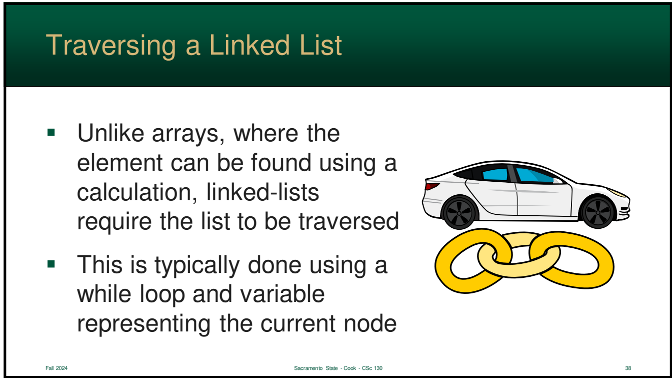
We Could Also Do This…

```
Node rat = new Node("Rat");
Node owl = new Node("Owl");
Node cat = new Node("Cat");

rat.next = owl;
owl.next = cat;
```


36



37



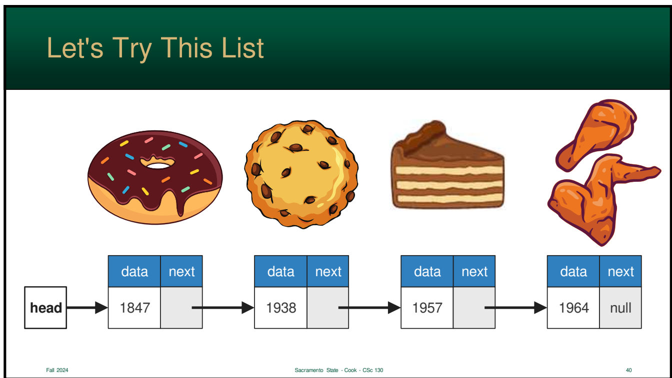
38

Node for Integers

```
public class Node
{
    public int data;
    public Node next;
}
```

Let's use int's for now

39

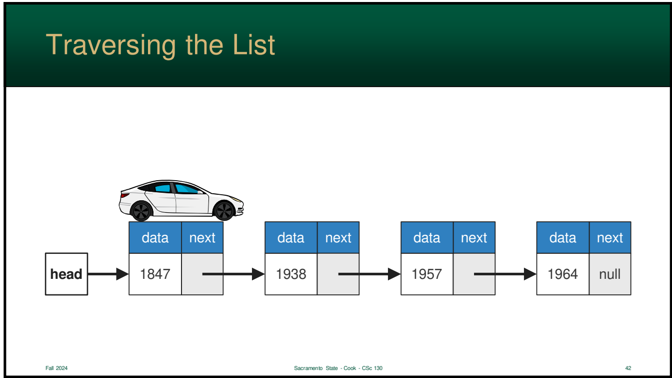


40

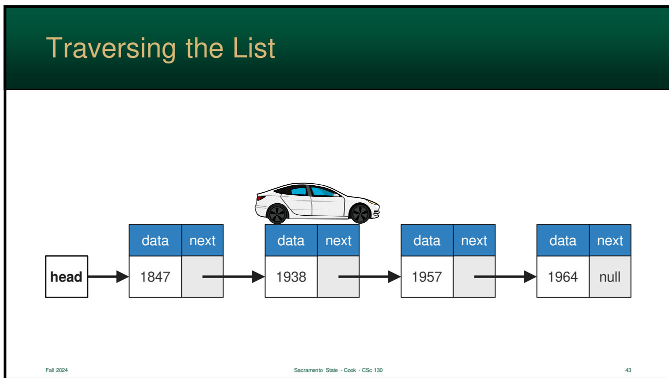
While Loop – Follow the Links

```
current = head;

while (current != null)
{
    System.out.println(current.data);
    current = current.next;   //Go to next node
}
```

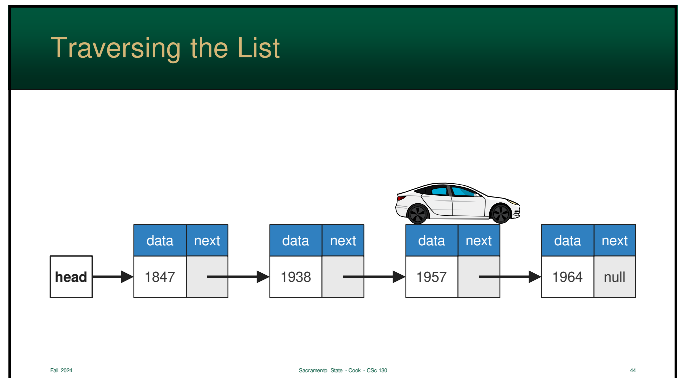

41



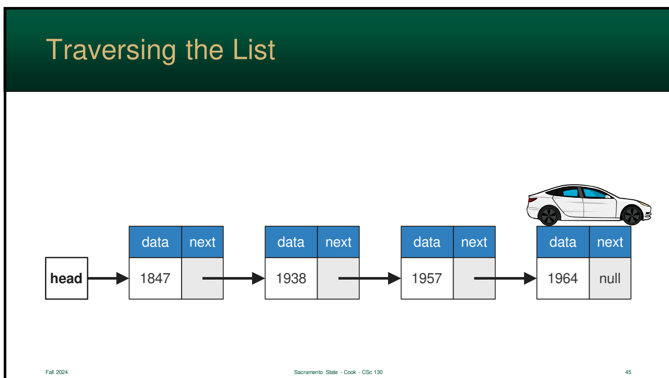
42



43



44



45

Adding to Linked Lists

Chain of Data

46

Adding to Linked Lists

- Adding to Linked Lists is easy to do, but must done with considerable care
- The links (references) need to be updated in a specific order
- … or a link will be lost

Fall 2024 Sacramento State - Cook - CSc 130 47

47

Adding to Linked Lists

- The first item in a linked list is referred to as the Head (alternatively *Front*)
- The last item, in which the next field is null, is called the *Tail*

Fall 2024 Sacramento State - Cook - CSc 130 48

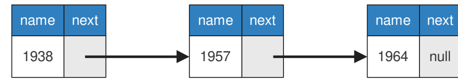
48

Adding to Linked Lists

- In this section, we will add a new node to the front, middle, and end of a linked list
- Most of these actions require just two steps

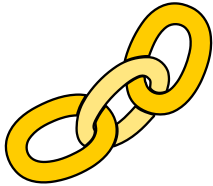
49

Let's Assume We Have This List



50

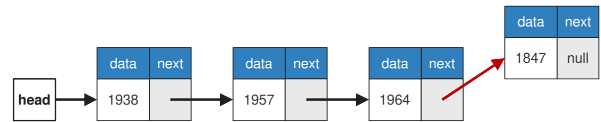
Adding to the Tail of a List



1. Link the tail node (who's next field is null) to the newly added node
2. If a reference to the tail is being maintained, it is linked to the newly added node

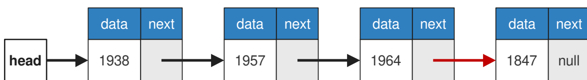
51

Add Tail: 1. Link Tail to the New Node



52

Add Tail: Resulting List



53

Adding to the Tail of a List

```
// add is the new node
// tail is the last node in the list

tail.next = add;
```

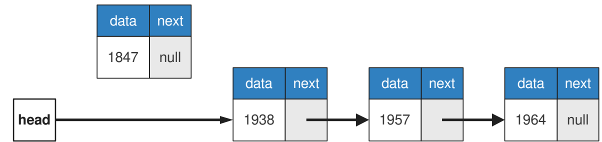

54

Adding to the Head of a List

1. The newly added node is linked to the head of the list
2. The head is then linked to the newly added node

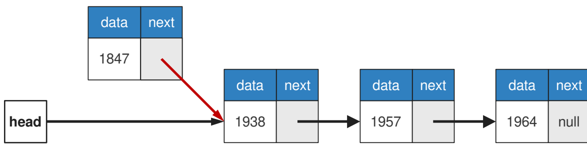
55

Adding to the Head of a List



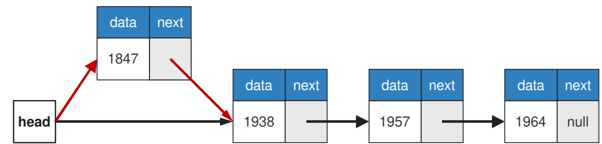
56

Add Head: 1. Link Node to Head's Reference



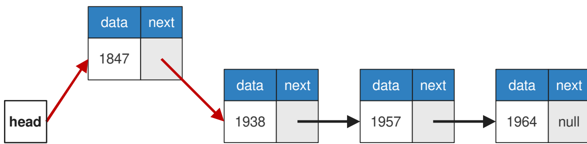
57

Add Head: 2. Set Head Reference to the Node



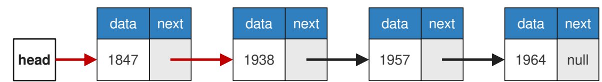
58

Add Head: 2. Set Head Reference to the Node



59

Add Head: Resulting List



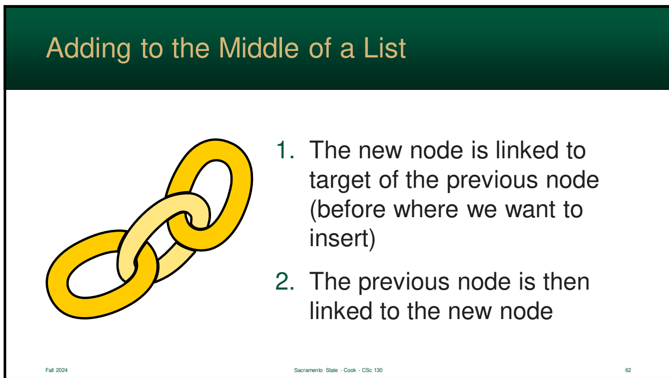
60

Adding to the Head of a List

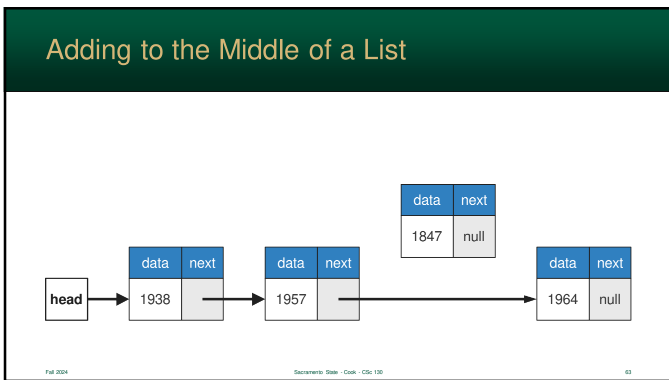
```
// add is the new node
// head is the first node in the list

add.next = head;
head = add;
```

Fall 2024 Sacramento State - Cook - CSc 130 61

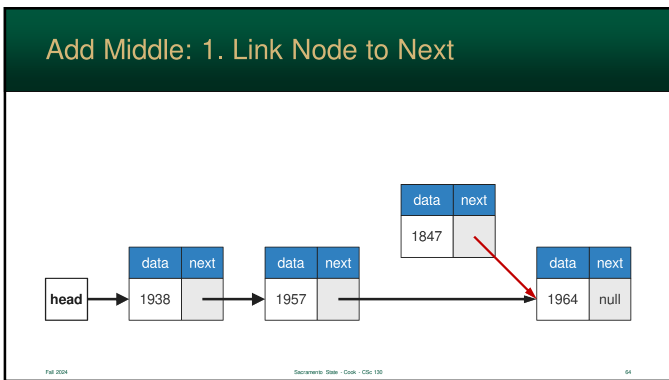
61



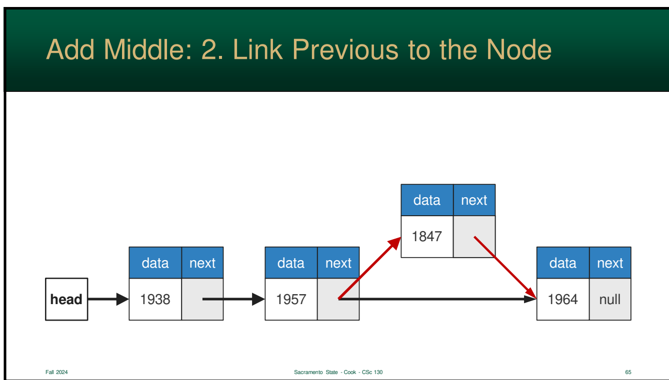
62



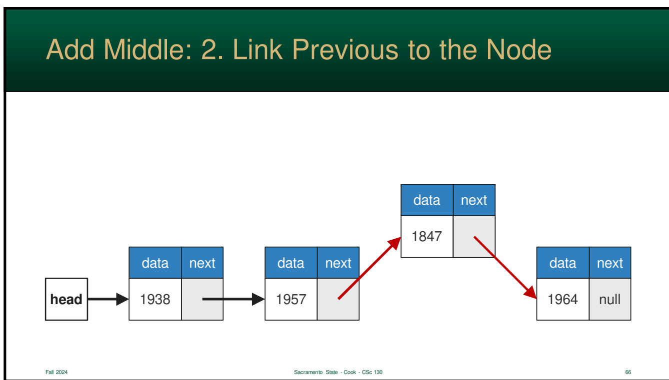
63



64

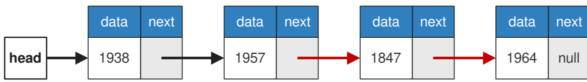


65



66

Add Middle: Resulting List



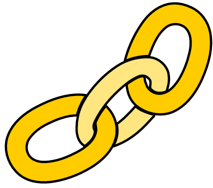
67

Adding to the Middle of a List

```
// add is the new node
// prev is the node before where
// add is to be inserted

add.next = prev.next;
prev.next = add;
```


68

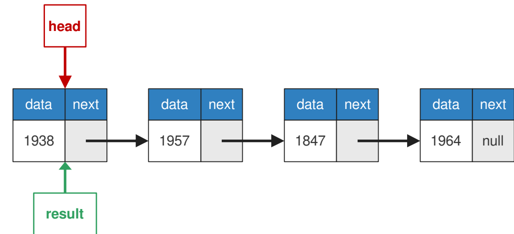
Removing The Head Node



1. Save Link to the Old Head
2. Update the Head Reference to the Head's next link
3. Remove the link from the old head to the new head

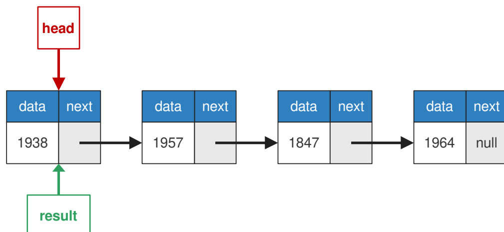
69

Remove Head: 1. Save Link to the Old Head



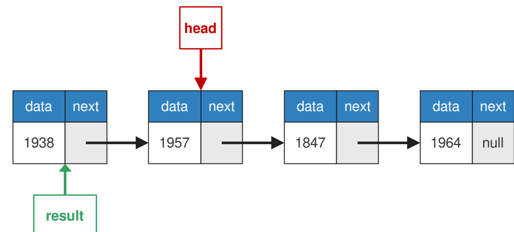
70

Remove Head: 2. Update the Head Reference



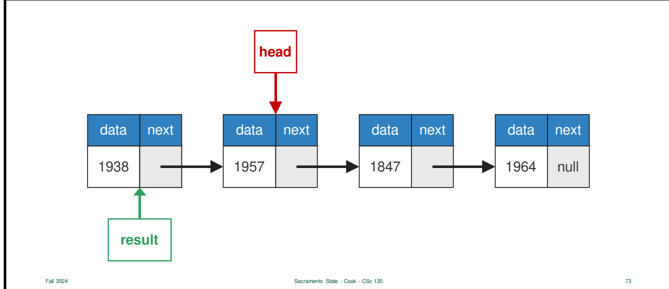
71

Remove Head: 2. Update the Head Reference



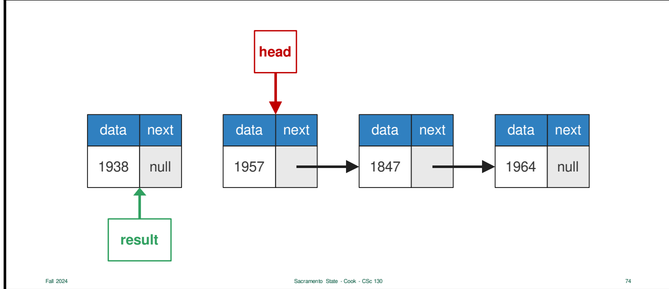
72

Remove Head: 3. Remove the Link



73

Remove Head: Complete



74

Remove Head

```
// Save a reference to the head
result = head;

//Set head to the head's next link
head = head.next;

//Remove link between old head and new head
result.next = null;
```

Exactly the same as a singly linked list

75

Maintaining a Tail Node

One should keep track the caboose

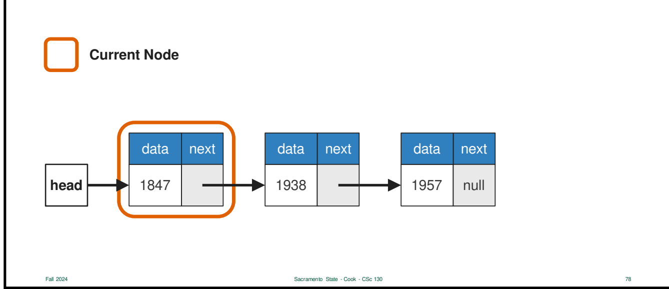
76

Head and Tail Nodes

- Linked lists maintain a link to the head node
- Often, in well-written linked lists, a link to the tail node is also maintained
- It is far more efficient

77

Adding to the End



78

Adding to the End

Current Node

head → [data: 1847 | next] → [data: 1938 | next] → [data: 1957 | next: null]

79

Adding to the End

Current Node

Found the last node in the list. next == null

head → [data: 1847 | next] → [data: 1938 | next] → [data: 1957 | next: null]

80

Adding to the End

Current Node

The new node is added

head → [data: 1847 | next] → [data: 1938 | next] → [data: 1957 | next] → [data: 1964 | next: null]

81

While Loop – Follow the Links

```
current = head;
while (current.next != null)
{
    current = current.next;  //Go to next node
}

// Current is now the tail. Link tail to new node
current.next = add;
```


82

So, that took awhile…

- Notice that, to get the tail now, we had to write loop to traverse all the nodes
- If we *knew* where the tail was beforehand, we wouldn't need a loop

83

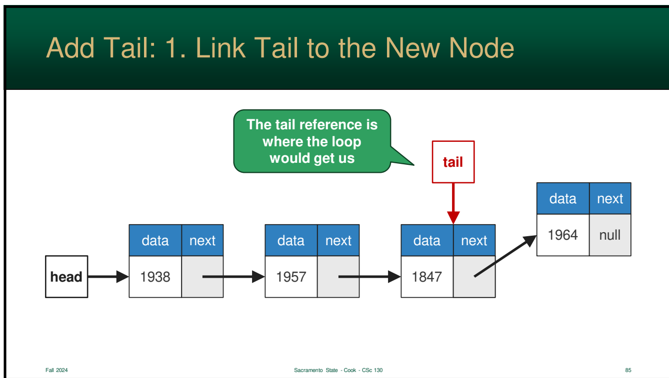
Adding to the End (with a tail)

The tail reference is where the loop would get us

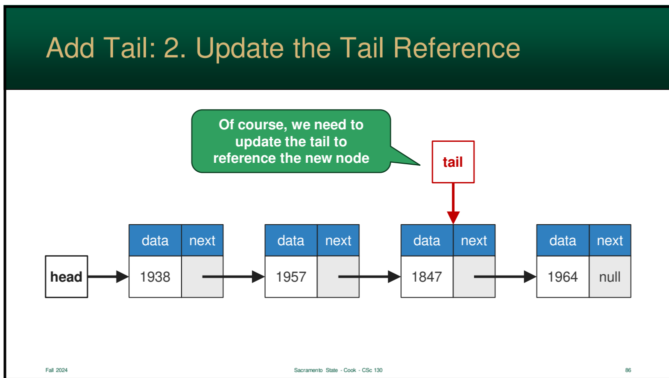
tail

head → [data: 1938 | next] → [data: 1957 | next] → [data: 1847 | next: null]

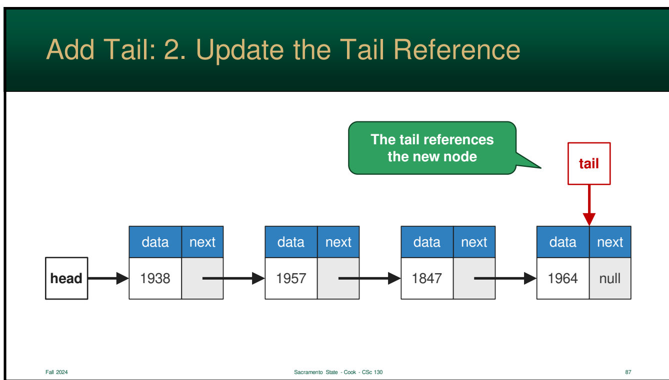
84



85



86



87

Adding to the end – with a Tail Node

```
tail.next = add;
tail = add;
```

Fall 2024 Sacramento State - Cook - CSc 130 88

88

Finding Second-to-Last

- As we noticed with the Singly-Linked list, finding the last item (to add at the end) required a loop
- … or was immediate if we maintained a tail node reference

Fall 2024 Sacramento State - Cook - CSc 130 89

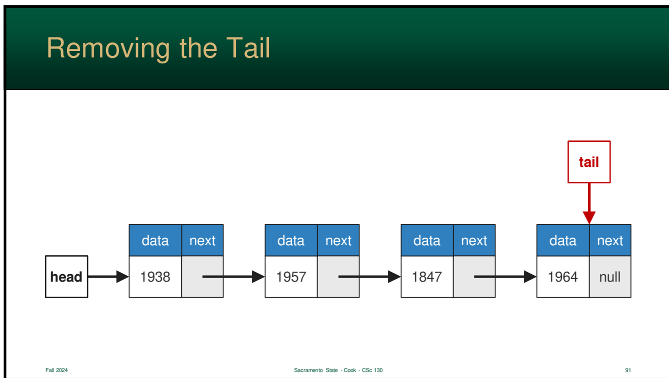
89

Finding Second-to-Last

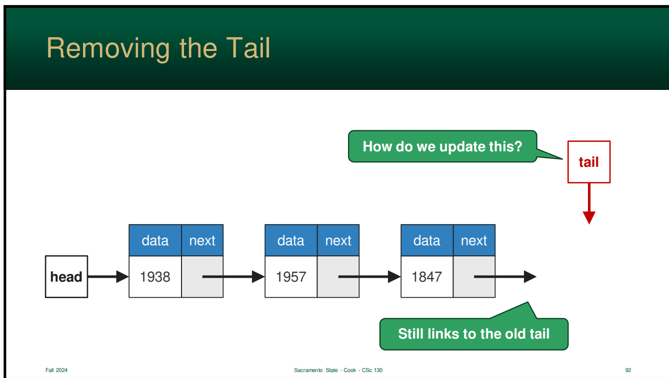
- But, how do we remove the last item?
- We can find the last item immediately, but that would make second-to-last the new tail

Fall 2024 Sacramento State - Cook - CSc 130 90

90



91



92

Remove Last –Linked List

```
current = head;
while (current != null)
{
    if (current.next == last)
    {
        nextToLast = current;
    }
    current = current.next;    //Go to next node
}

// Now remove last
result = last;
nextToLast.next = null;       //Remove the old list
last = nextToLast;
```

Fall 2024 Sacramento State - Cook - CSc 130 93

93

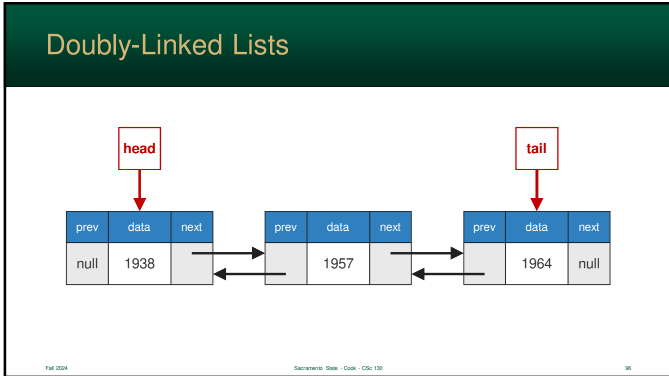
94

Doubly-Linked Lists

- Another variation of a linked list is the *doubly-linked list*
- As the name implies, there are two sets of links – one that points to the next node and one that points to the previous

Fall 2024 Sacramento State - Cook - CSc 130 95

95



96

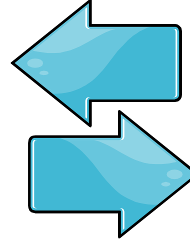
Doubly-Linked List Node

```java
public class Node
{
    public Object data;
    public Node prev;
    public Node next;
}
```

Sometimes called `last`

97

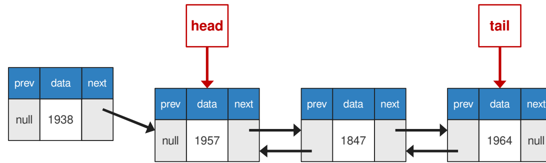
Doubly Linked List: Add to the Head



1. Link New Node to the Head
2. Link Head Back to the New Node
3. Update the Head Reference to new node

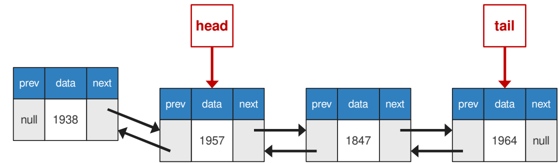
98

Add Head: 1. Link New Node to the Head



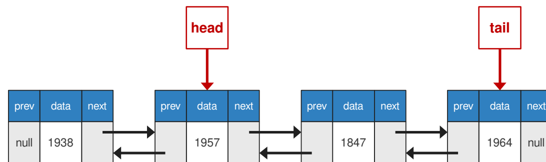
99

Add Head: 2. Link Head Back to the New Node



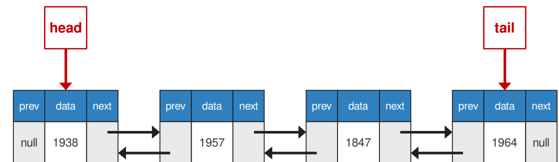
100

Add Head: 3. Update the Head Reference



101

Add Head: 3. Update the Head Reference



102

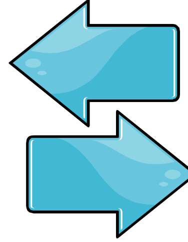
Add Head – Doubly Linked List

```
// Link the new node to old head
add.next = head;

// Link the old head back to the new node
head.prev = add;

//Set head to the new node
head = add;
```

Also may be wise to check if the head == null

103

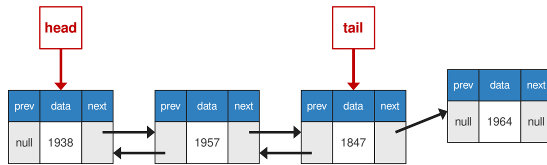
Doubly Linked List: Add to the Tail



1. Link Tail to the New Node
2. Link the New Node to the Old Tail
3. Update the Tail Reference to the New Node

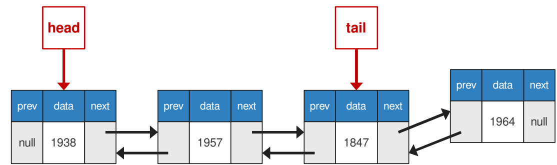
104

Add Tail: 1. Link Tail to New Node



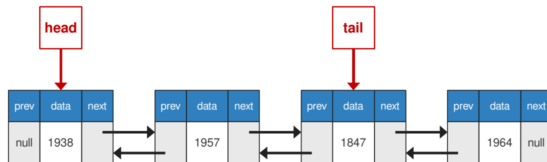
105

Add Tail: 2. Link the New Node to the Old Tail



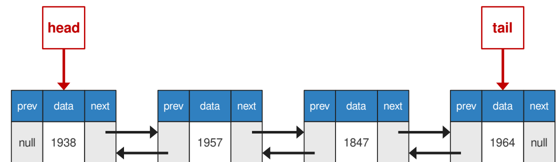
106

Add Tail: 3. Update the Tail Reference



107

Add Tail: 3. Update the Tail Reference



108

Add Tail – Doubly Linked List

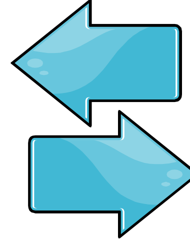
```
// Link the old tail to the new node
tail.next = add;

// Link new node back to the old tail
add.prev = tail;

//Set tail to the new node
tail = add;
```

Also may be wise to check if the tail == null

Fall 2024 Sacramento State - Cook - CSc 130 109

109

Doubly Linked List: Remove Head

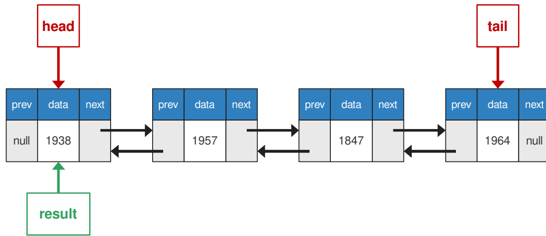


1. Save Link to the Old Head
2. Update the Head Reference to the Head's next reference
3. Remove links between Old Head and New Head

Fall 2024 Sacramento State - Cook - CSc 130 110

110

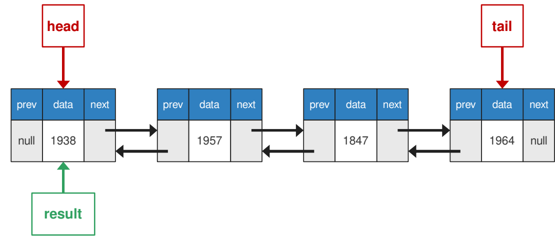
Remove Head: 1. Save Link to the Old Head



Fall 2024 Sacramento State - Cook - CSc 130 111

111

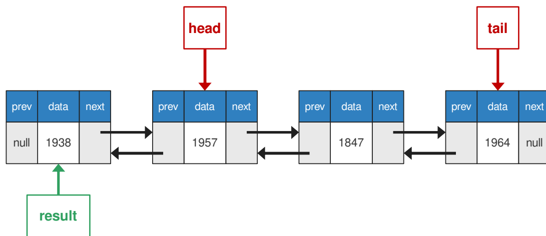
Remove Head: 2. Update the Head Reference



Fall 2024 Sacramento State - Cook - CSc 130 112

112

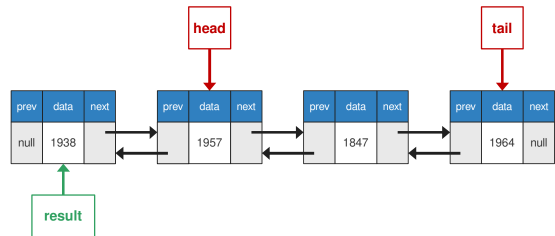
Remove Head: 2. Update the Head Reference



Fall 2024 Sacramento State - Cook - CSc 130 113

113

Remove Head: 3. Remove Links



Fall 2024 Sacramento State - Cook - CSc 130 114

114

Remove Head: Complete

head

tail

prev	data	next
null	1938	null

prev	data	next
null	1957	

prev	data	next
	1847	

prev	data	next
	1964	null

result

Fall 2024 Sacramento State - Cook - CSc 130 115

115

Remove Head – Doubly Linked List

```
// Save a reference to the tail
result = head;

//Set head to the head's next link
head = head.next;

//Remove links between old head and new head
head.prev = null;
result.next = null;
```

Exactly the same as a singly linked list

Fall 2024 Sacramento State - Cook - CSc 130 116

116

Doubly Linked List: Remove Tail

1. Save Link to the Old Tail
2. Update the Tail Reference to the previous reference of the current Tail
3. Remove links between the New Tail and the Old Tail

Fall 2024 Sacramento State - Cook - CSc 130 117

117

Remove Tail: 1. Save Link to the Old Tail

head

tail

prev	data	next
null	1938	

prev	data	next
	1957	

prev	data	next
	1847	

prev	data	next
	1964	null

result

Fall 2024 Sacramento State - Cook - CSc 130 118

118

Remove Tail: 2. Update the Tail Reference

head

tail

prev	data	next
null	1938	

prev	data	next
	1957	

prev	data	next
	1847	

prev	data	next
	1964	null

result

Fall 2024 Sacramento State - Cook - CSc 130 119

119

Remove Tail: 2. Update the Tail Reference

head

tail

prev	data	next
null	1938	

prev	data	next
	1957	

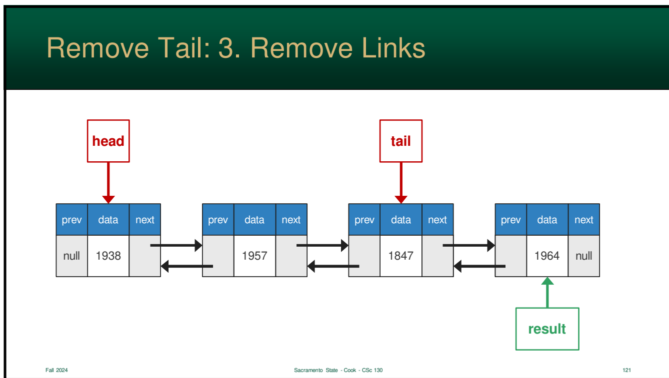
prev	data	next
	1847	

prev	data	next
	1964	null

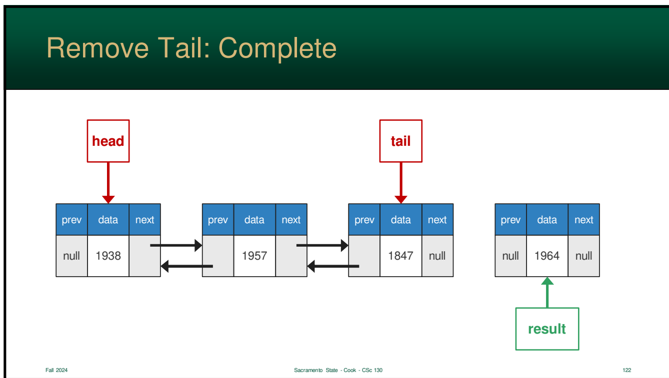
result

Fall 2024 Sacramento State - Cook - CSc 130 120

120



121



122

Remove Tail – Doubly Linked List

```
// Save a reference to the tail
result = tail;

//Set tail to the previous of the old tail
tail = tail.prev;

//Remove links between old tail and new tail
tail.next = null;
result.prev = null;
```

Fall 2024 Sacramento State - Cook - CSc 130 123

123

Singly-Linked List Class

Creating a train of nodes

124

Linked List Class

- Maintaining both a head and tail node can be a tad difficult
- So, we can place them into a LinkedList class
- Then we can write methods to add to the end (the tail) and the front (the head)

Fall 2024 Sacramento State - Cook - CSc 130 125

125

Linked List Class

```
class LinkedList
{
    public Node head;
    public Node tail;
}
```

Fall 2024 Sacramento State - Cook - CSc 130 126

126

Linked List Class

```
public void AddTail(Node node)
{
    if (head == null)      //Add first node
    {
        head = node;       //Link both
        tail = node;
    }
    else
    {
        tail.next = node;    //Link old tail to the new node
        tail = node;         //Now the new node is the tail
    }
}
```


127

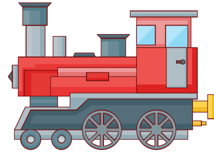
Linked List Class

```
public void AddHead(Node node)
{
    if (head == null)      //Add first node
    {
        head = node;       //Link both
        tail = node;
    }
    else
    {
        node.next = head;    //Link new node to the current head
        head = node;         //Now the new node is the head
    }
}
```


128

Linked List Class

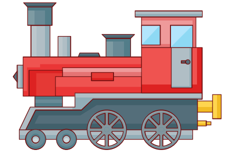
- Now that we have compensated for the head/tail being null, we can also a method to remove the head
- But there are more cases that need to be considered



129

Linked List Class

1. There are 2+ nodes (the head and tail are different)
2. There is only one node (the head and tail are the same)
3. There are no nodes



130

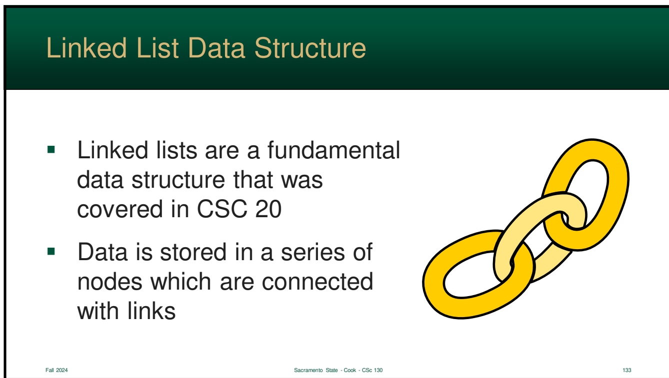
```
public Node RemoveHead()
{
    Node result;

    if (head == null)
    {
        result = null;          //We could also throw an error
    }
    else if (head == tail)      //Just one node. Set both head/tail to null
    {
        result = head;          //...or tail - it doesn't matter here. Note, we are saving the reference in 'result'.
        head = null;            //Deference both
        tail = null;
    }
    else                        //2 or more nodes.
    {
        result = head;
        head = head.next;       //Link new node to the current head
    }

    return result;
}
```


131

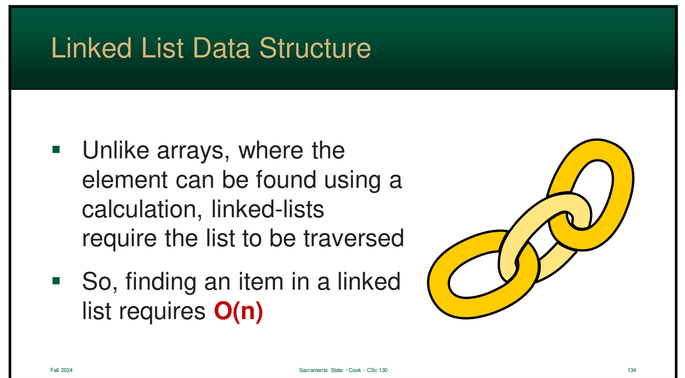
Linked List Big-O

How Good Is This?

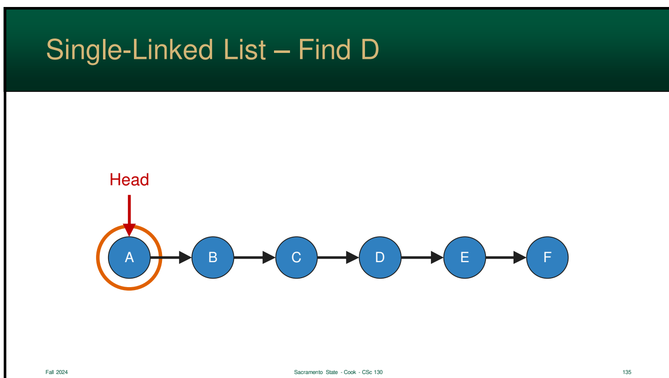
132



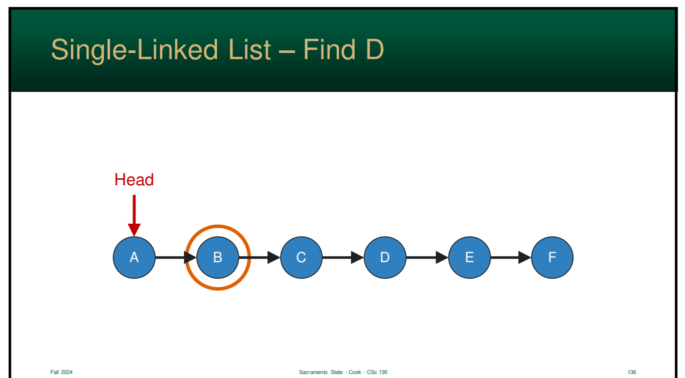
133



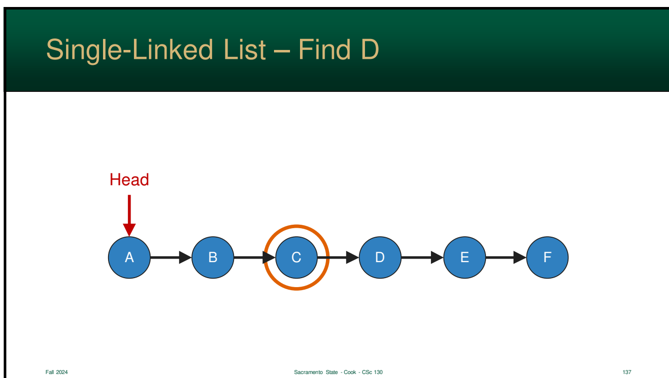
134



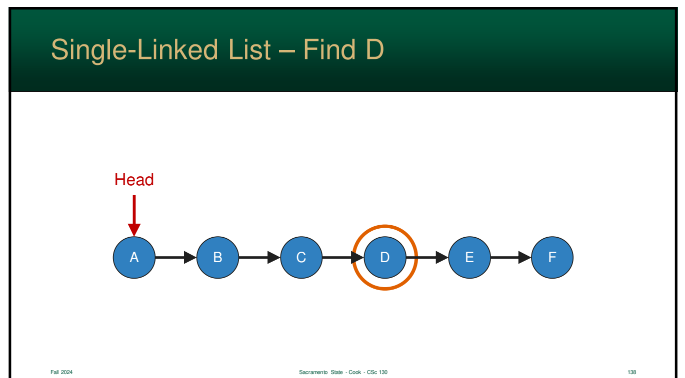
135



136



137



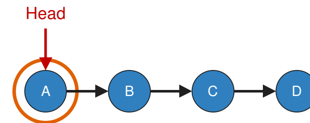
138

Head and Tail Nodes

- Linked lists maintain a link to the head node
- Often, in well-written linked lists, a link to the tail node is also maintained
- Why? It has a huge impact on time complexity

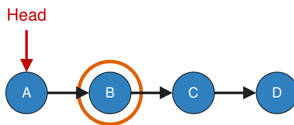
139

Append Value – No Tail Node



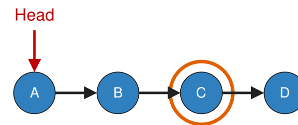
140

Append Value – No Tail Node



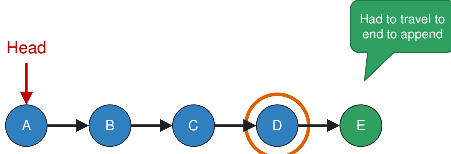
141

Append Value – No Tail Node



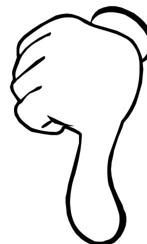
142

Append Value – No Tail Node



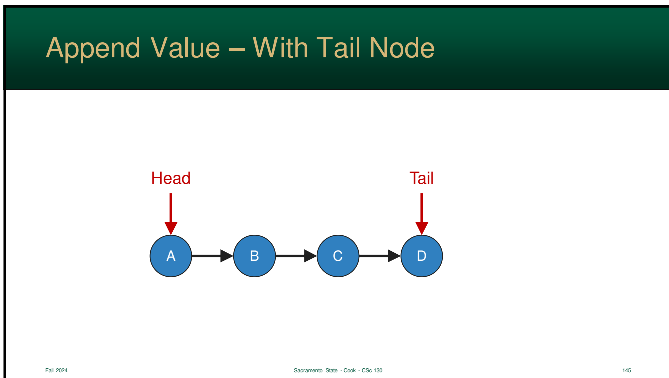
143

Head and Tail Nodes

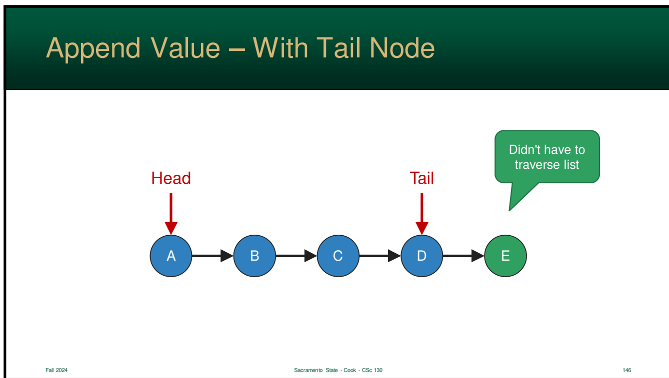


- Without a tail node, the entire list must be traversed to find the end
- This will require **O(n)**
- Adding a tail node, will decrease it to **O(1)**

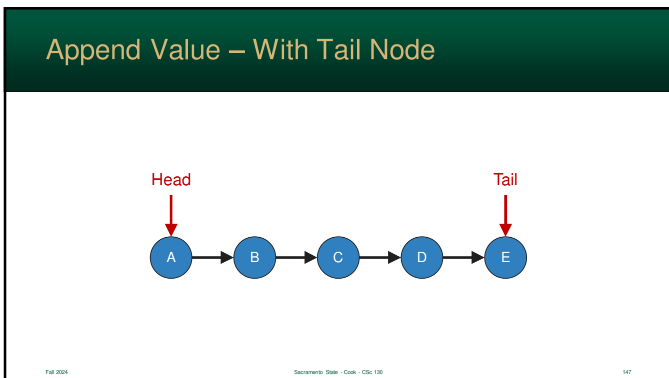
144



145



146



147

Use a Tail Node!

- Unless you are only appending nodes at the head of a linked list, maintain a tail node
- For **all** the examples used in these slides… assume the linked list has a tail node

148

Auxiliary Storage in Linked Lists

- Unlike arrays, linked lists must store the "next" links between nodes
- So, the *auxiliary storage* overhead is O(n)
  - …which is usually the size of an address
  - 64-bit system → 8 bytes

149

Big-O: Test Your Might…

```
LinkedList list;

for(i = 0; i < list.Count; i++)
{
   total += list.Find(i);
}
```

O(n)

O(n)

$O(n^2)$

150

Iterators

- To avoid accidental $O(n^2)$, major programming languages support *iterator objects*
- They store information about the current state (e.g. a node) when data is being are sequentially read

Fall 2024 Sacramento State - Cook - CSc 130 151

151

Iterators

- Iterators maintain O(n) for sequentially accessing all the list's elements
- This is the purpose of the For-Each Statement
- Notation varies greatly between languages (when they are supported)

Fall 2024 Sacramento State - Cook - CSc 130 152

152

Dynamic Array vs. Linked List

Operation	Dynamic Array	Linked List
Find (to read or write)	O(1)	**O(n)**
Insert (arbitrary)	**O(n)**	**O(n)**
Add first/last	**O(n)**	O(1)
Remove first/last	**O(n)**	O(1)
Auxiliary storage	O(1)	**O(n)**

Fall 2024 Sacramento State - Cook - CSc 130 153

153