ANNOUNCEMENTS
today is **No Laptop Monday**!

Week08a

WARMUP
"a chain is only as strong as its weakest link"
- what does this expression mean?
- what is a chain?
- what is a link?
- is this true for real-world metal chains? why or why not?

TODAY linked lists

linked lists

record lecture

(p)review: list interface

list interface

```
– // Get the element with this index.
– ElementType get(int index);

– // Add (append) an element to the back of the list.
  void add();

– // Add (insert) an element into the list so it has this index
  void add(int index, ElementType element);

– // Remove (delete) the element in the list at this index.
  void remove(int index);

– // Get the number of elements currently in the list.
  int size();
```

list interface (cont.)

```
– // NOTE: Many other functions could be included
  //       in this interface.
  – void sort(); // Sort the list.
  – void reverse(); // Reverse the list.

  – List<ElementType> sorted(); // Get sorted copy of the list.
  – List<ElementType> reversed(); // Get reversed copy of list.

  – // Get index of first element with this value.
    int find(ElementType element);

  – ...
```

a few weeks ago,
we implemented the list interface
using an array

the *array list*

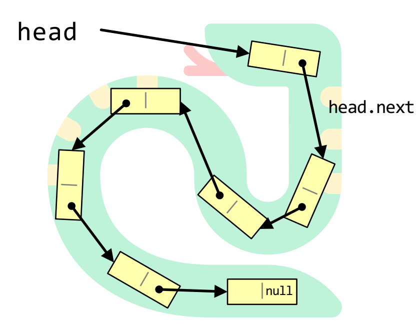
this week, we will implement
the list interface using nodes with
"links" (references) to other nodes

this will be called a **linked list**

note

today we will be discussing the
simplest possible linked list

(LinkedList literally just has a
reference to Node head.)



some other implementations are possible.
some will be faster than this one.

for linked lists, do NOT memorize big O runtimes out of context

why are we doing this?

A: it will be cool to see two very different implementations of the same interface ⛄

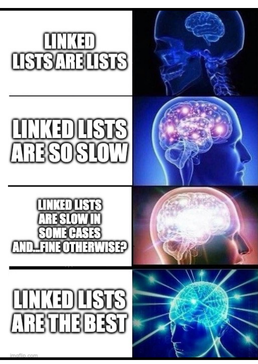
B: linked lists will prepare us for trees and graphs 🌳

C: linked lists are incredibly FUNdaMENTAL 🥳

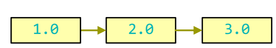
(for us, as fundamental as arrays)

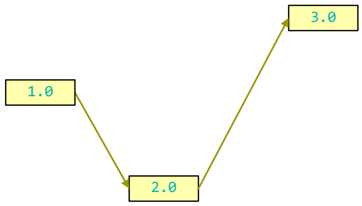
D: linked lists are actually big O better (than array lists) in very specific cases

E: linked lists are actually really, really important
(especially in the C programming language)



linked list





linked list

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

```
class Node {
  Value value;
  Node next;

  Node(Value value) {
    this.value = value;
  }
}
```

```
LinkedList list = new LinkedList();

list.head = new Node(1.0);
```

1.0

```
list.head.next = new Node(2.0);
```

1.0 → 2.0

```
list.head.next.next = new Node(3.0);
```

1.0 → 2.0 → 3.0

linked list

```
class LinkedList {
  Node head;

  void add(Value value) {
    ...
  }
}
```

```
class Node {
  Value value;
  Node next;

  Node(Value value) {
```

```
LinkedList list = new LinkedList();

list.add(1.0);
```

1.0

```
list.add(2.0);
```

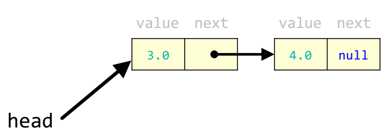
1.0 → 2.0

```
list.add(3.0);
```

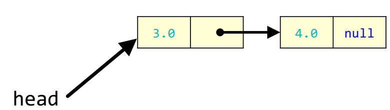
1.0 → 2.0 → 3.0

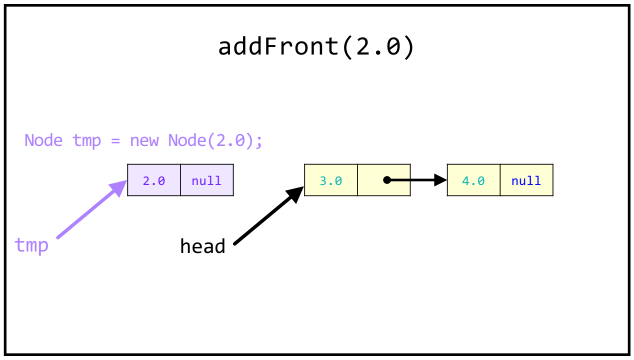
example: `addFront(value)`

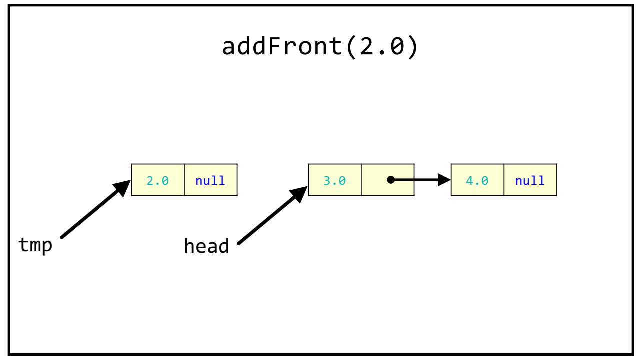
`addFront(2.0)`

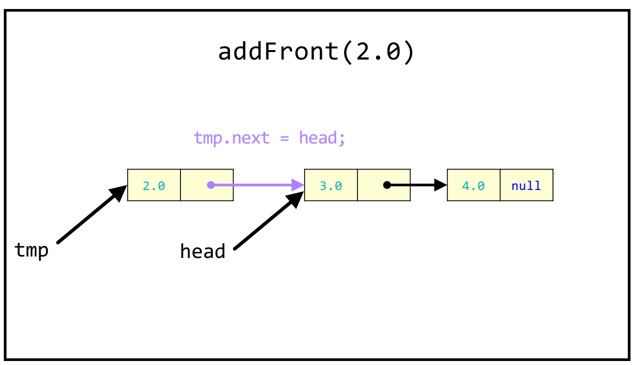


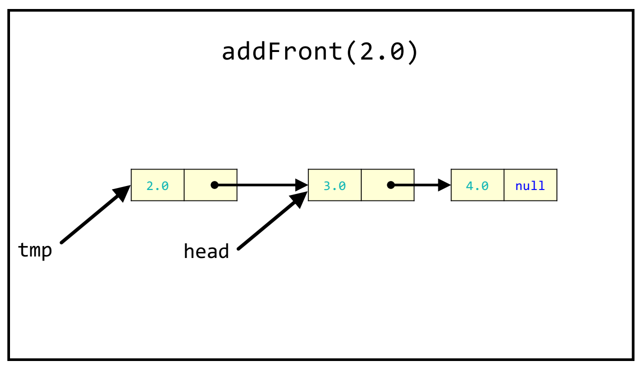
`addFront(2.0)`

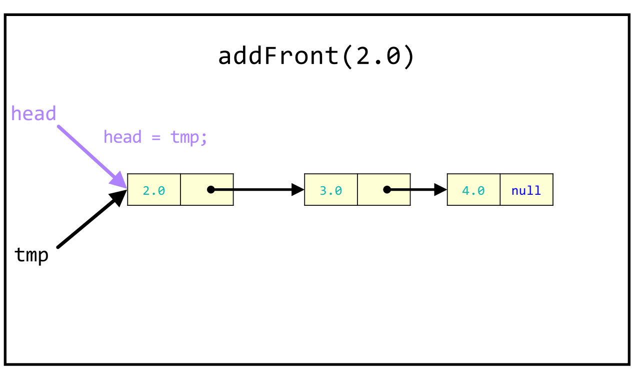


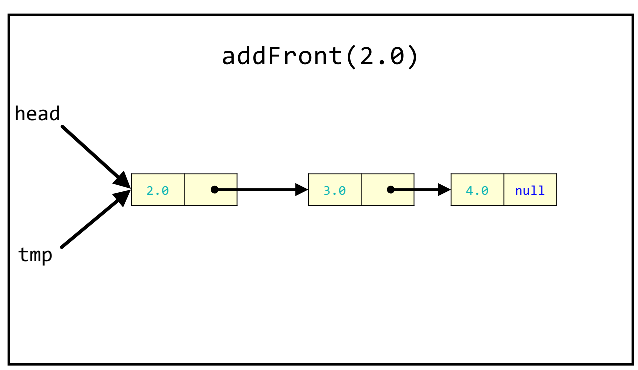

addFront(2.0)
Node tmp = new Node(2.0);
2.0
null
3.0
4.0
null
tmp
head

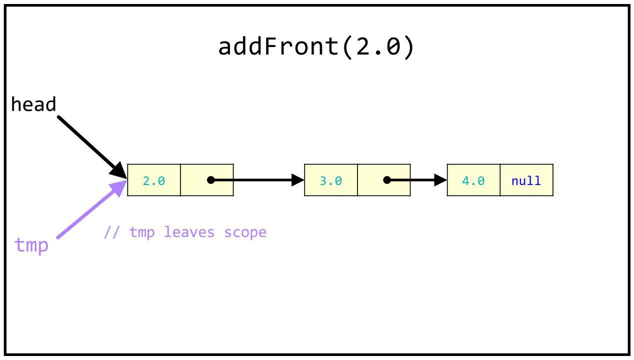

addFront(2.0)
2.0
null
3.0
4.0
null
tmp
head

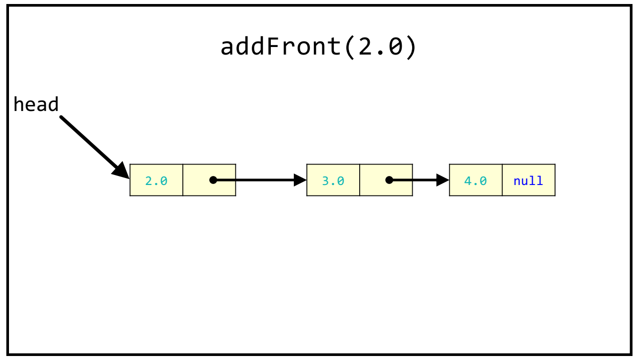

addFront(2.0)
tmp.next = head;
2.0
3.0
4.0
null
tmp
head


addFront(2.0)
2.0
3.0
4.0
null
tmp
head

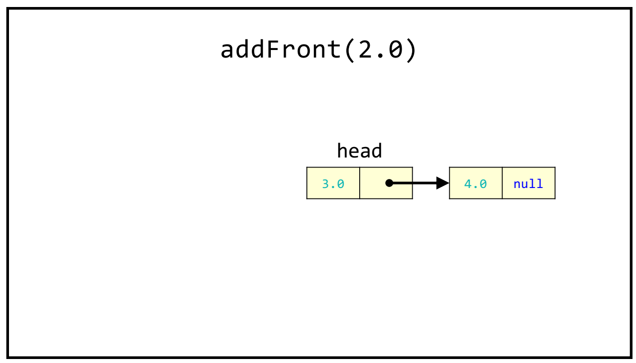

addFront(2.0)
head
head = tmp;
2.0
3.0
4.0
null
tmp

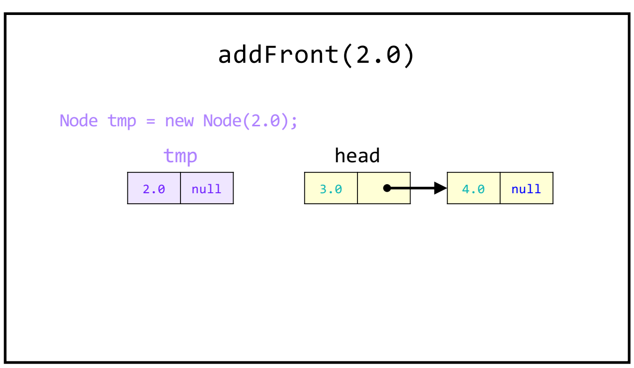

addFront(2.0)
head
2.0
3.0
4.0
null
tmp

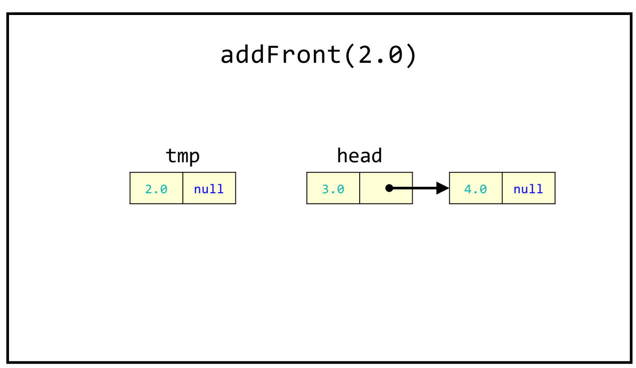




same thing but with
labels instead of arrows
for head and `tmp`

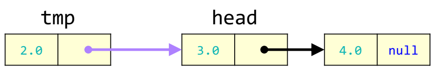




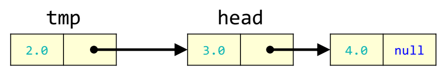


addFront(2.0)

tmp.next = head;

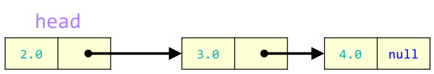


addFront(2.0)

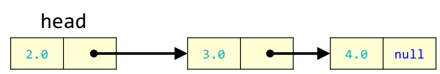


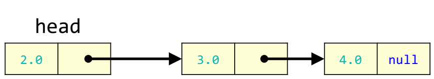
addFront(2.0)

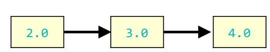
head = tmp;



addFront(2.0)



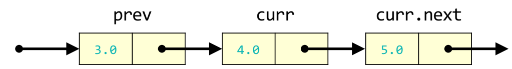




(even) more abstract diagram

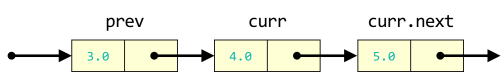
example: `remove`

remove

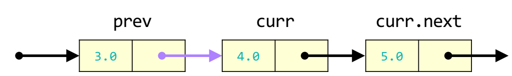



```
Node prev = null; // the previous node
Node curr = head; // the current node
while (...) {
    ...
    prev = curr;
    curr = curr.next;
}
```

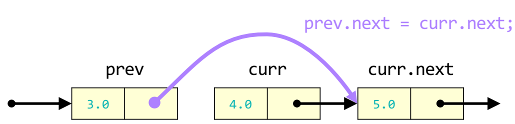
remove



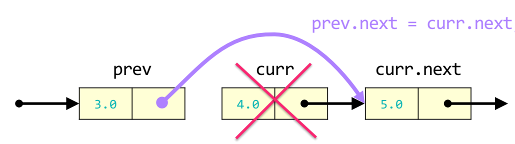
remove



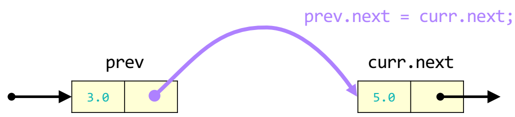
remove



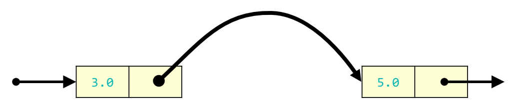
remove



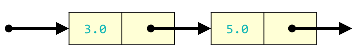
remove



remove

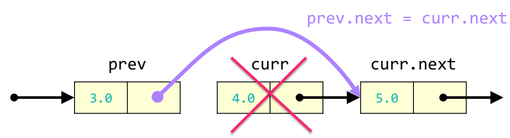


remove



where did the middle node go?

- way back when...we created it using `new`
- then we just like..."stopped referring to it"
- is it gone now? 😨



🚛 it has been garbage collected

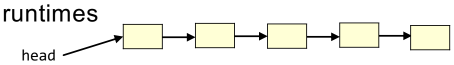
[board discussion of
"no directed path from stack to the node"]

big O runtimes

what is the big O runtime of size()?
[pointing activity]

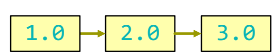
O(n) 😢

singly-linked list
worst case runtimes

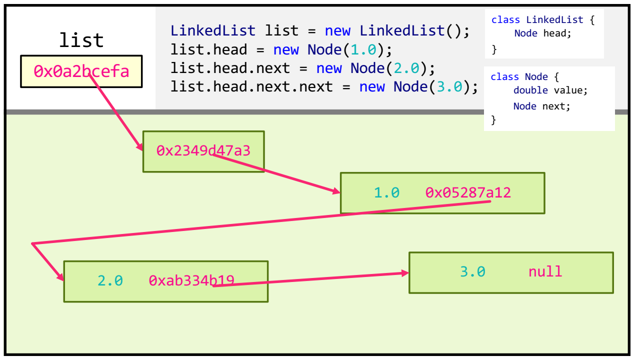


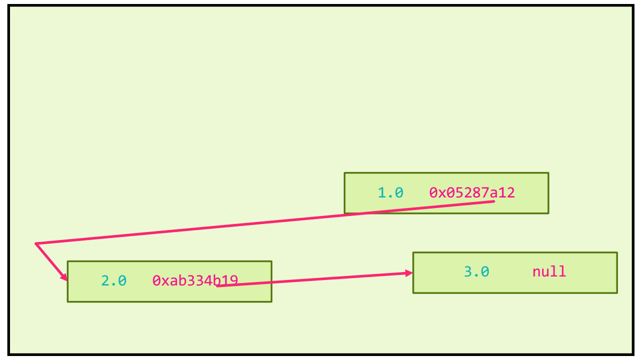
- `list.add(index, value)`
  - `// O(n)`
- `list.removeByIndex(index)`
  - `// O(n)`
- `list.size()`
  - `// O(n)`
- `list.addFront() // list.add(0, value)`
  - `// O(1)`
- `list.removeFront() // list.removeByIndex(0)`
  - `// O(1)`
- `list.addBack()`
  - `// O(n)`
- `list.removeBack()`
  - `// O(n)`

beyond big O runtime



what does this actually look like in memory?





what does this *mean*?

cons? 🙁

pros? 😳
(how is this very different than an array list?)

https://x.com/_kzr/status/1672497446705037312_