

Indian Institute of Technology Madras
ZANZIBAR CAMPUS

Z5007 · PROGRAMMING AND DATA STRUCTURES

Linked Lists: Nodes, References and Pointer Surgery

Lecture 13 · Thursday 29 October 2026 · Canning, Broder & Lafore Ch 5

Dr. Innocent Nyalala | MTech Semester 1 | IIT Madras Zanzibar

Today's Plan

IIT Madras Zanzibar

1. (0–5 min) Why another structure? The cost of inserting at the front of an array
2. (5–13 min) Nodes and references: what a variable really holds in Python
3. (13–18 min) Walking the chain: traversal and search
4. (18–31 min) Changing the chain: insert and delete, every pointer drawn
5. (31–37 min) Two pointers: the middle in one pass; reversing a list
6. (37–50 min) Costs, live demo, practice questions, your questions

Learning Outcomes

IIT Madras Zanzibar

By the end of this session, you will be able to:

Explain	why inserting at the front of an array is O(n) and at the front of a linked list O(1)
Draw	a linked list as nodes and arrows, and say what head, tail, next and None mean
Trace	a traversal with cur = cur.next and say when it stops
Order	the pointer changes for insert and delete so that no node is lost
Compare	array and linked list costs, and choose the right one for a task

Key Terms for Today

IIT Madras Zanzibar

Term	Meaning	Example
Node	a small object holding a value and a reference to the next node	[12 | •]
Reference (pointer)	a value that says where an object is; following it reaches the object	n1.next
None	Python's 'no object'; used as the end-of-list marker	last.next = None
head	reference to the first node; the only way into the list	head → 12
tail	reference to the last node; makes adding at the end fast	tail → 33
Traversal	visiting every node by following next from head	cur = cur.next
Singly linked	each node knows only the NEXT node, not the previous one	today
Contiguous	stored side by side in memory, like an array	Python list

Linked Lists

→ **Why Another Structure?**

Nodes and References

Walking the Chain

Changing the Chain

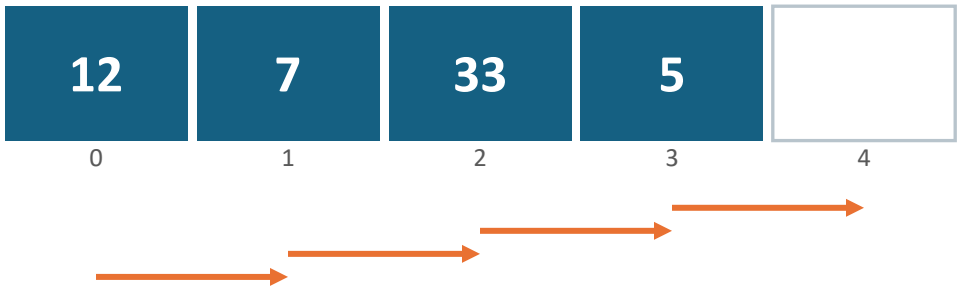
Two-Pointer Techniques

Costs, Demo and Practice

The Problem: Inserting at the Front of an Array

IIT Madras Zanzibar

Insert 99 at index 0:



every item moves one box right: 4 moves

After the insert:



n items → n moves → O(n) per insert at the front

? **Think:** a list of 1,000,000 items. How many moves does ONE insert at the front cost? And one delete from the front?

Definition: Contiguous storage: items in neighbouring memory boxes. Fast to index (a[i] is O(1)), slow to insert or delete near the front.

The Idea: Each Item Remembers Where the Next One Is

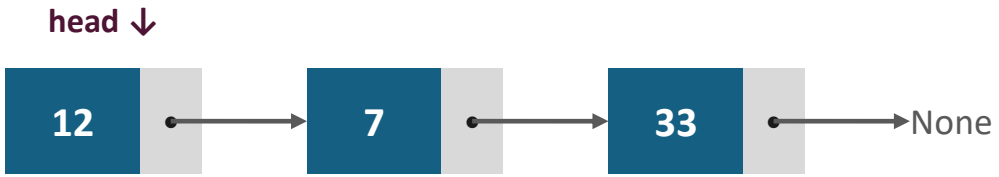
IIT Madras Zanzibar

In memory: scattered boxes

0x18	33	next: None
0x40	12	next: 0x9C
0x9C	7	next: 0x18

head = 0x40

How we draw it: follow the arrows



1. Nodes can be anywhere in memory.
2. Each node holds a value and the address of the next node.
3. None marks the end. head is the way in.

Treasure hunt: each clue says where the next clue is. To add a clue at the start, write one clue pointing at the old first clue.

Linked Lists

Why Another Structure?

→ **Nodes and References**

Walking the Chain

Changing the Chain

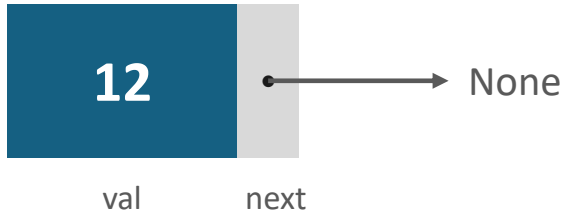
Two-Pointer Techniques

Costs, Demo and Practice

The Node: a Value and a Reference

```
class Node:
    def __init__(self, val, next=None):
        self.val = val      # the data
        self.next = next    # next node or None
```

n = Node(12)



1. val holds the data: a number, a string, a student record, anything.
2. next holds a reference to another Node, or None at the end of the list.
3. A new node is not linked to anything: its next is None until we set it.

? **Think:** n = Node(12). What is n.next? What is n.next.val?

Definition: Reference: a value that tells Python where an object is. Following it (n.next) reaches that object.

What a Variable Holds: an Arrow, Not a Box

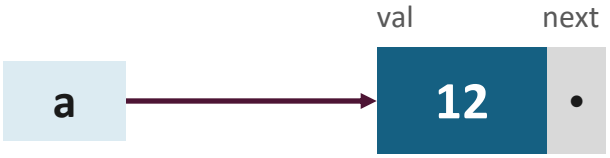
IIT Madras Zanzibar

```
a = Node(12)
b = a
b.val = 99
```

1. A variable holds an arrow to an object.
2. Assignment copies the arrow.
3. Two names can reach one object.

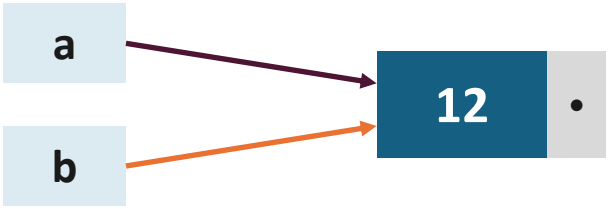
1. a = Node(12)

Makes one node holding 12. a stores an arrow to it.



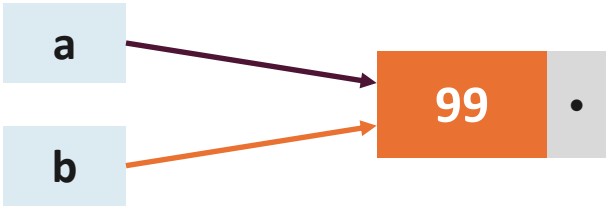
2. b = a

Copies the arrow, not the node. Still only one node.



3. b.val = 99

Follows b's arrow and changes 12 to 99 inside that node.



? **Think:** after these three lines, what is a.val? Why?

Building a Chain by Hand, One Line at a Time

```
n1, n2, n3 = Node(12), Node(7), Node(33)
n1.next = n2
n2.next = n3
head = n1
```

1. Each next = … draws one arrow.
2. The last node keeps next = None.
3. head = n1 gives the list its entrance.

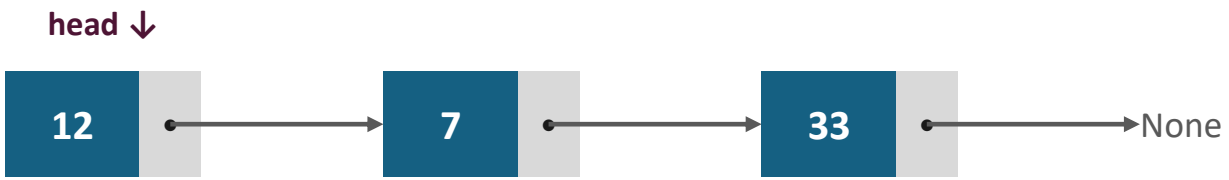
Line 1: three separate nodes



Lines 2 and 3: link them



Line 4: head is the way in



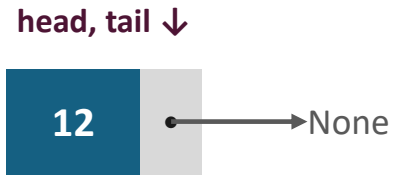
? **Think:** if I now write head = n2, what happens to the node holding 12?

The List Object: head, tail and a Counter

IIT Madras Zanzibar

Empty `head = None,  tail = None,  n = 0`

One node



`n = 1`

Many nodes



`n = 3`

Rule: every operation must work for the empty list, a one-node list, and a long list.

? **Think:** in the one-node list, is head is tail True or False?

Linked Lists

Why Another Structure?

Nodes and References

→ **Walking the Chain**

Changing the Chain

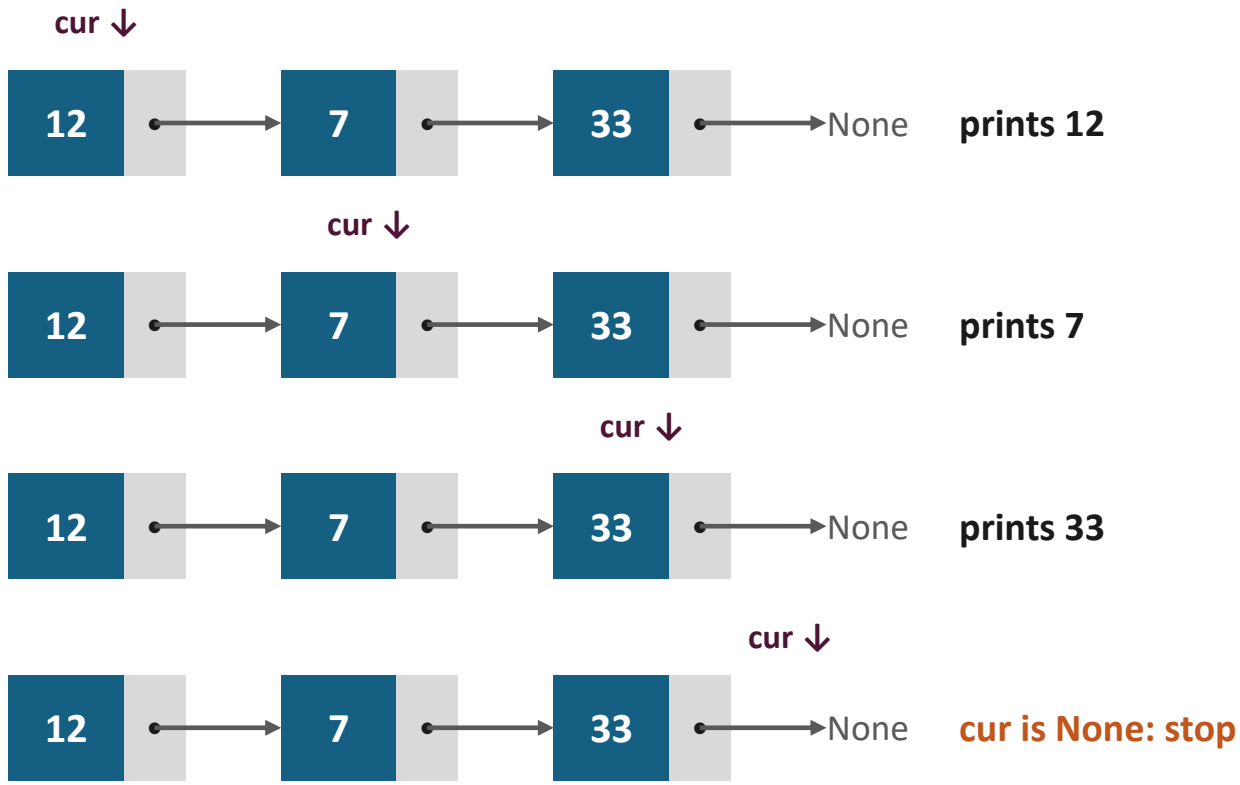
Two-Pointer Techniques

Costs, Demo and Practice

Traversal: Follow the Arrows Until None

```
cur = head
while cur is not None:
    print(cur.val)
    cur = cur.next
```

1. cur starts at head.
2. Visit the node, then cur = cur.next.
3. Stop when cur is None.
4. head itself never moves.



Think: how many times does the loop body run for n nodes? What is the cost?

Predict, Then Reveal: an Off-by-One Bug

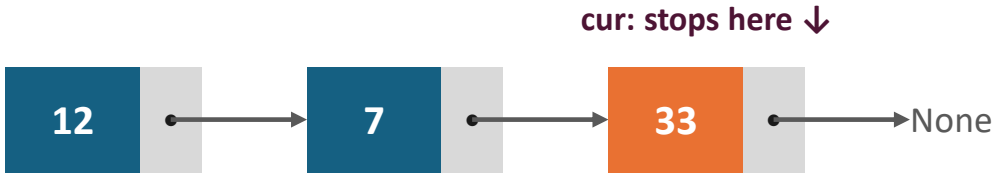
IIT Madras Zanzibar

```
cur = head
while cur.next is not None:    # changed
    print(cur.val)
    cur = cur.next
```

List: 12 → 7 → 33 → None

Predict: what is printed?

Prints 12, 7. The last node is never visited.



1. Empty list: cur is None, and None.next raises AttributeError.
2. Use cur is not None to visit all; cur.next is not None to stop AT the last node.

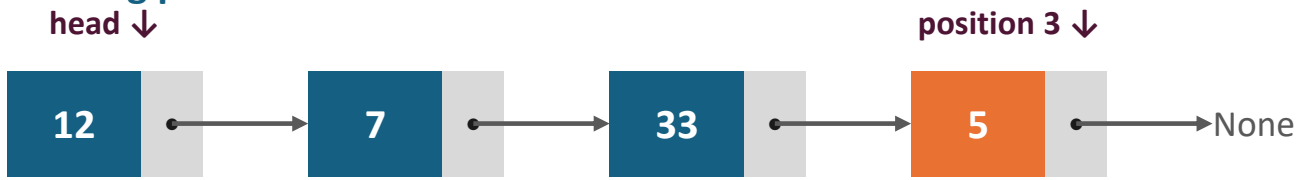
? **Think:** what does the changed loop do on an empty list?

Search and Access by Position: Why O(n)

IIT Madras Zanzibar

Task	Array	Linked list	Why
get the i-th item	O(1)	O(i), worst O(n)	must follow i arrows from head
search for a value	O(n)	O(n)	may have to visit every node
binary search (sorted)	O(log n)	no gain	jumping to the middle is O(n)

Reaching position 3: three arrows



? **Think:** can binary search help on a sorted linked list? Why or why not?

Linked Lists

Why Another Structure?

Nodes and References

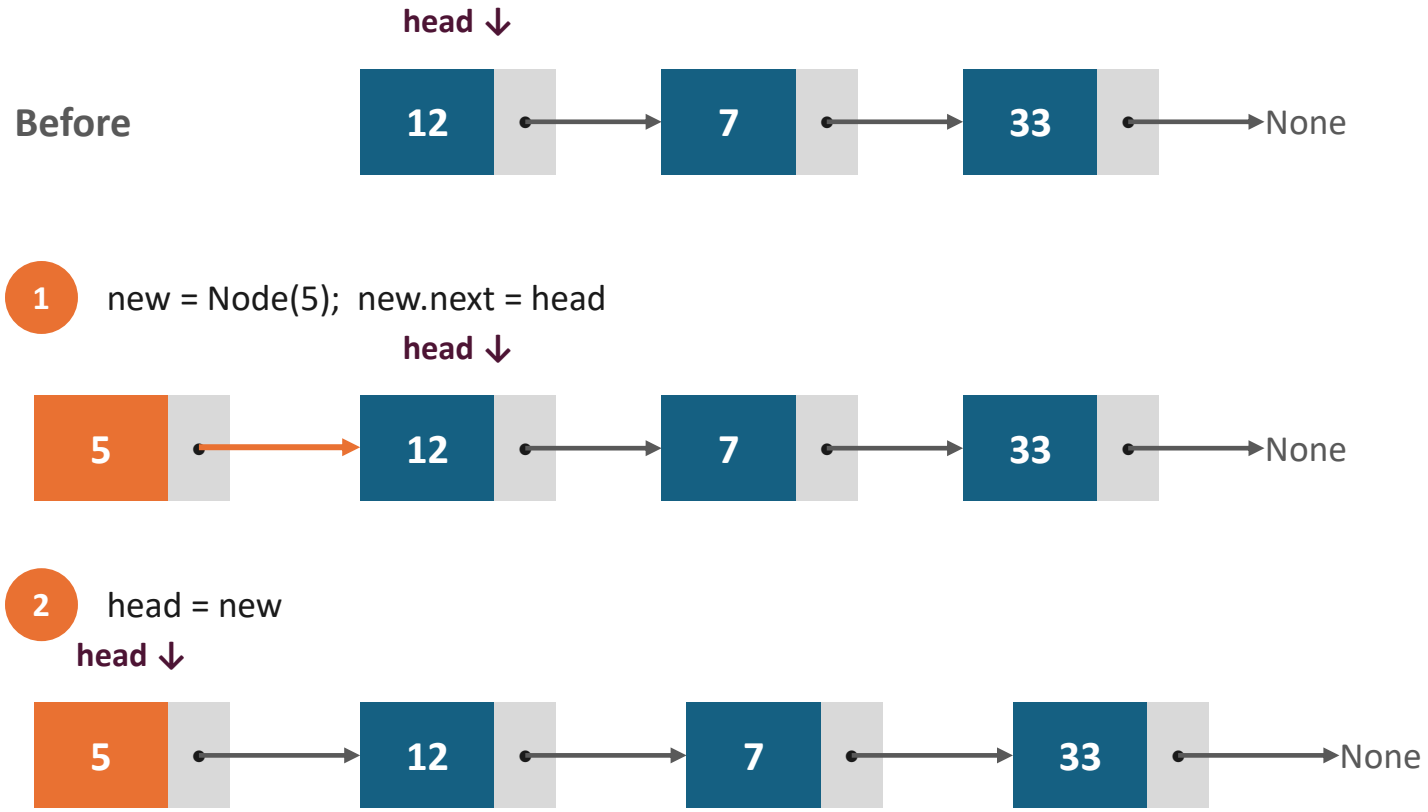
Walking the Chain

→ **Changing the Chain**

Two-Pointer Techniques

Costs, Demo and Practice

Insert at the Front (push_front): Two Steps, O(1)



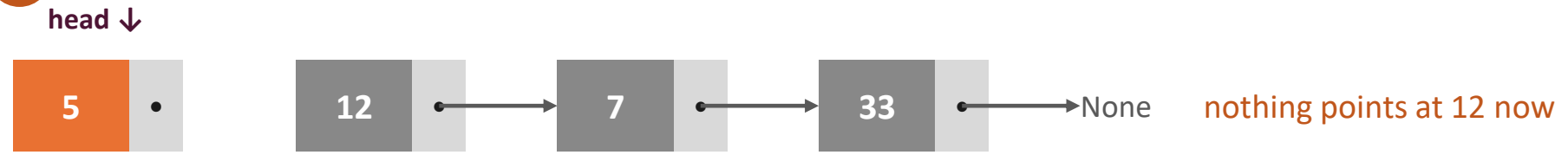
1. Step 1 links the new node to the old front.
2. Step 2 moves head.
3. Empty list: also set tail = new.
4. Cost: 2 steps, O(1).

? **Think:** what goes wrong if I do step 2 before step 1?

What If I Swap the Two Steps?

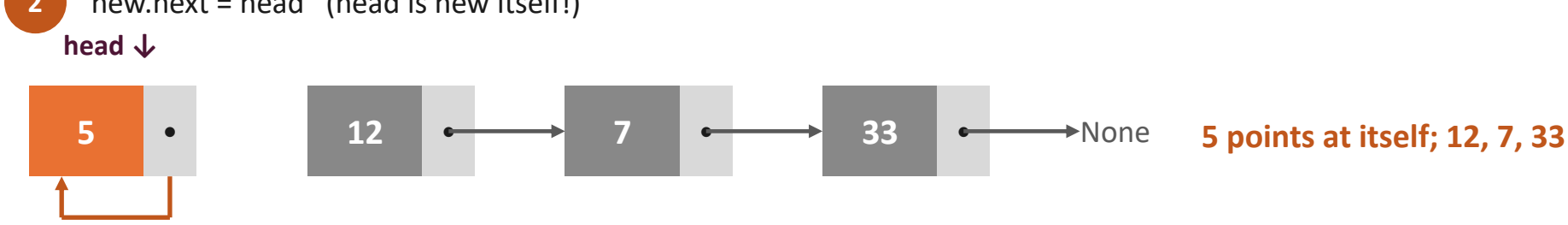


1 head = new (wrong: done first)



nothing points at 12 now

2 new.next = head (head is new itself!)

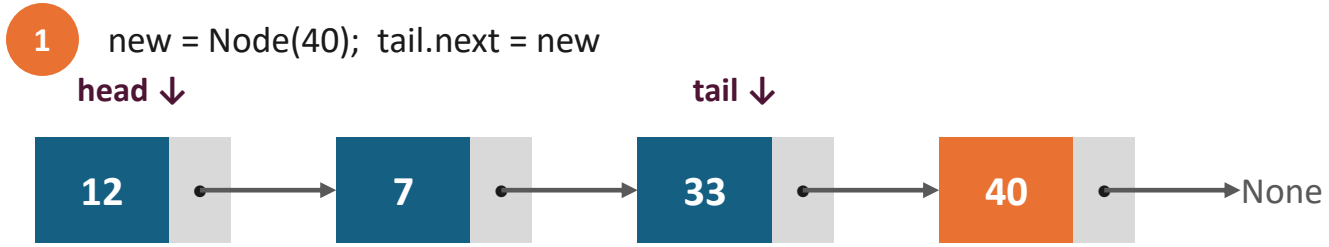


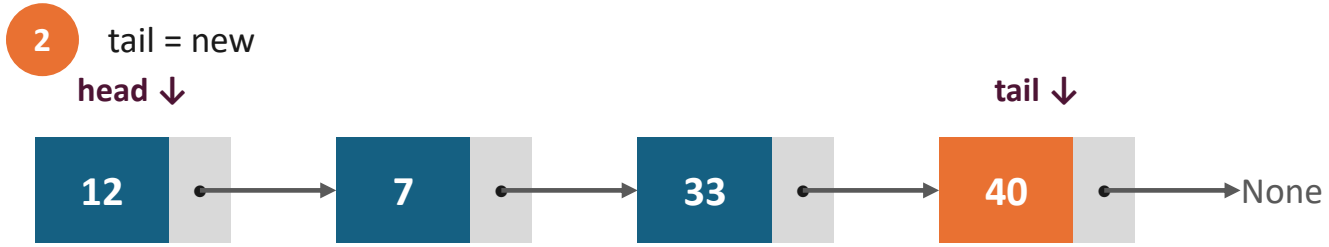
5 points at itself; 12, 7, 33 are lost

> **Rule:** connect the new node first, then move the old references.

Insert at the End (push_back): Why We Keep tail







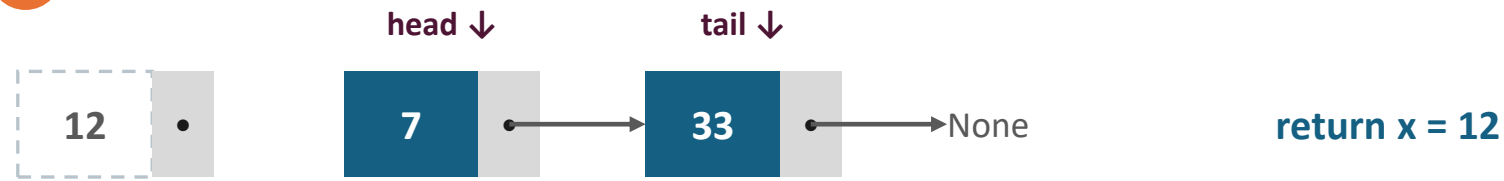
1. Step 1: old last node → new.
2. Step 2: move tail.
3. Empty list: head = tail = new.
4. With tail: O(1). Without: O(n).

? **Think:** what does push_back cost if the list keeps only head?

Remove from the Front (pop_front): One Step

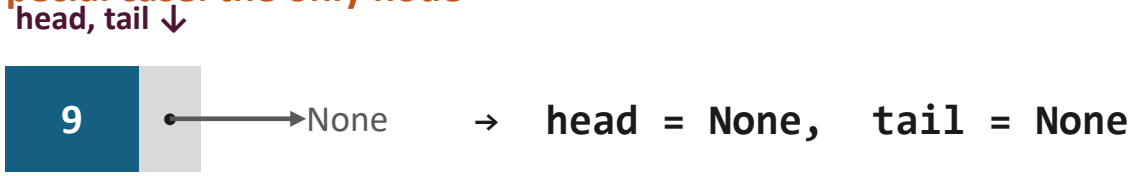


1 x = head.val; head = head.next



return x = 12

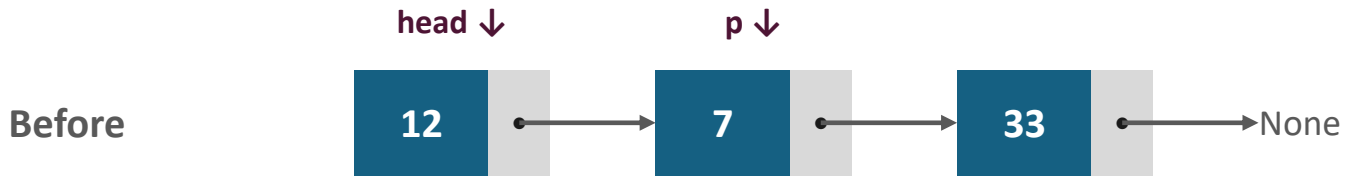
Special case: the only node



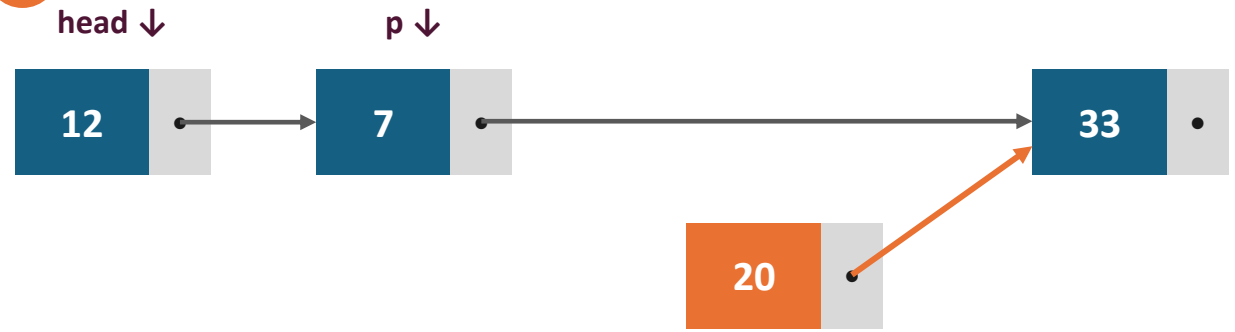
1. Empty list: raise IndexError.
2. List becomes empty: set tail = None too.

? **Think:** one-node list: what are head and tail after pop_front?

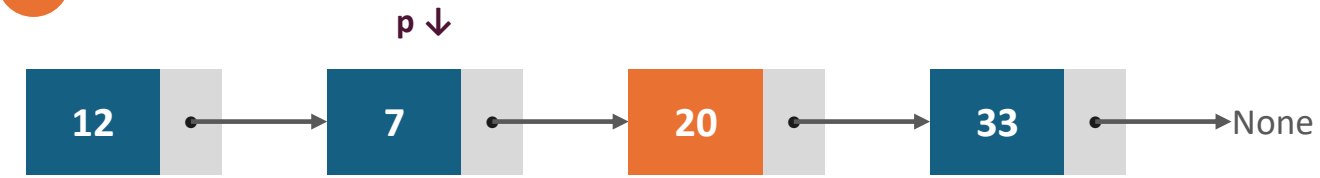
Insert After a Node You Already Hold



1 new = Node(20); new.next = p.next



2 p.next = new



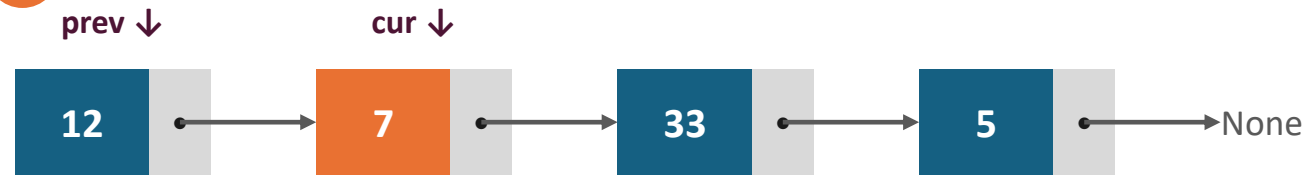
1. Step 1: new → node after p.
2. Step 2: p → new.
3. Cost: O(1) once you hold p.

? **Think:** what extra step is needed if p is the tail?

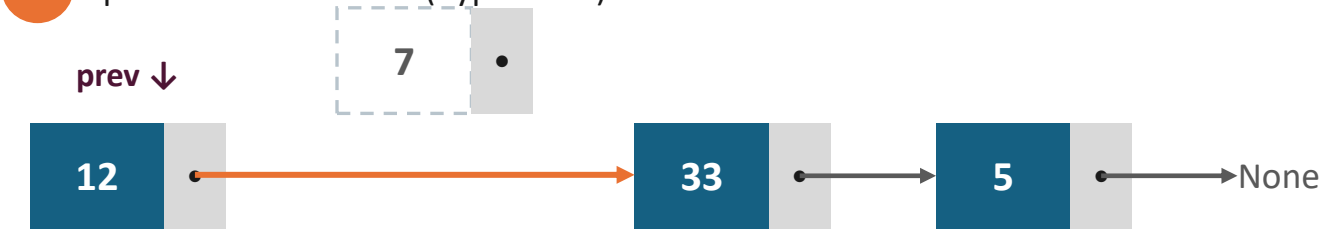
Delete a Node in the Middle: You Need the One Before It



1 walk: prev, cur = cur, cur.next until cur.val == 7



2 prev.next = cur.next (bypass cur)



1. cur = node to delete; prev = node before it.
2. First node: head = cur.next.
3. Last node: tail = prev.
4. Finding it is O(n); the bypass is O(1).

? **Think:** why is deleting the LAST node O(n) even with a tail reference?

The Special-Case Checklist

IIT Madras Zanzibar

Operation	Empty list	One node	First node	Last node
push_front	set tail too	—	—	—
push_back	set head too	—	—	—
pop_front	raise IndexError	set tail = None	—	—
remove(x)	return False	head and tail	move head	move tail to prev

How to test: for every method, try the empty list, a one-node list, the first node, the last node, and a long list.

? **Think:** which special case appears in every row of the table?

Linked Lists

Why Another Structure?

Nodes and References

Walking the Chain

Changing the Chain

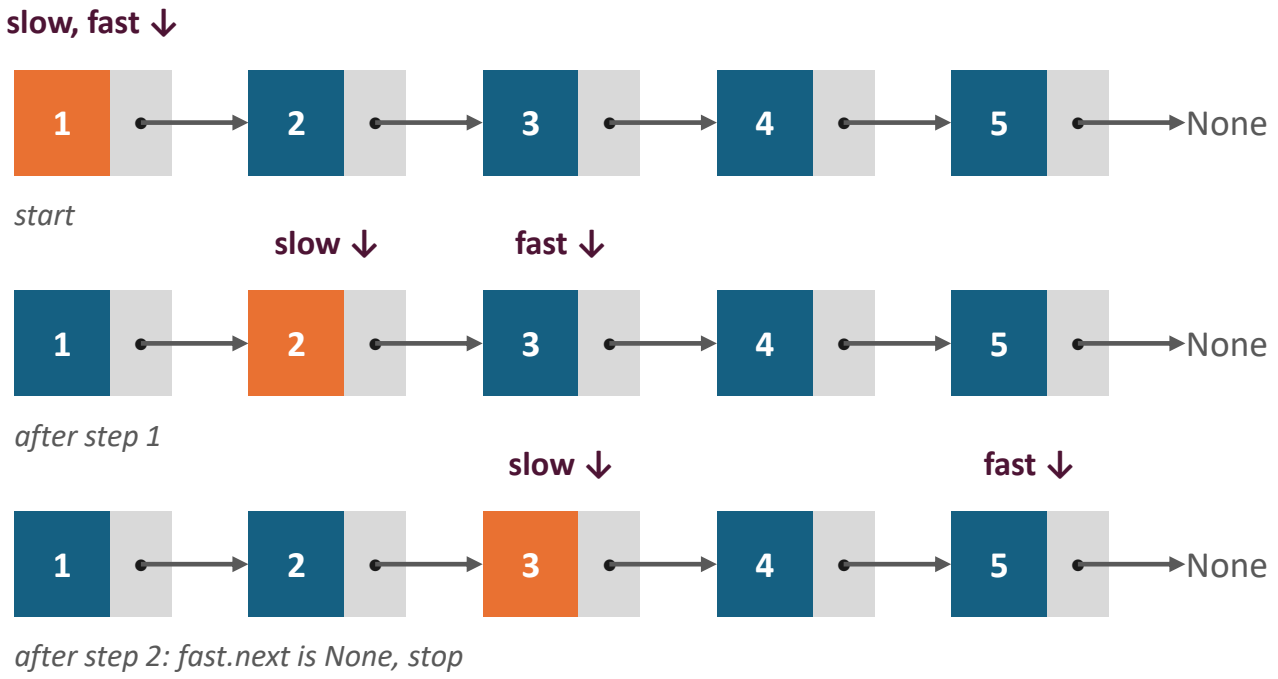
→ **Two-Pointer Techniques**

Costs, Demo and Practice

The Middle in One Pass: slow and fast

```
slow = fast = head
while fast is not None and fast.next is not None:
    slow = slow.next
    fast = fast.next.next
```

1. fast covers twice the distance of slow.
2. When fast reaches the end, slow is halfway.
3. One pass, O(1) extra memory.



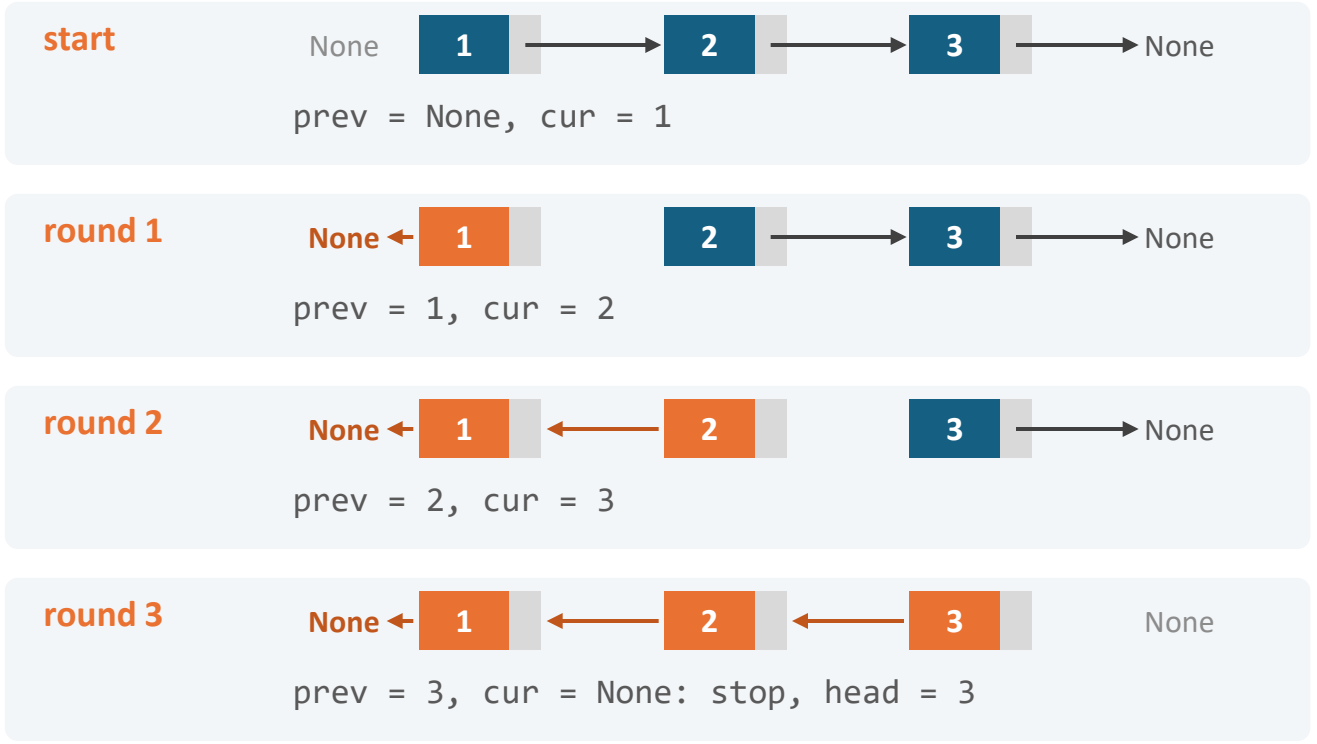
Think: with 6 nodes (1 to 6), where does slow stop?

Reversing a List in Place: Three References

IIT Madras Zanzibar

```
prev, cur = None, head
while cur is not None:
    nxt = cur.next      # save
    cur.next = prev     # turn
    prev, cur = cur, nxt # step
head = prev
```

1. Save, turn, step: three moves per node.
2. O(n) time, O(1) extra memory.
3. The new head is prev.



? **Think:** why must nxt be saved BEFORE cur.next = prev?

Linked Lists

Why Another Structure?

Nodes and References

Walking the Chain

Changing the Chain

Two-Pointer Techniques

→ **Costs, Demo and Practice**

Array or Linked List? The Cost Table

IIT Madras Zanzibar

Operation	Python list (array)	Singly linked list (head + tail)
get i-th item	O(1)	O(n)
insert / delete at front	O(n)	O(1)
insert at end	O(1) amortized	O(1)
delete at end	O(1)	O(n): needs the node before tail
insert after a node you hold	O(n) shift	O(1)
search	O(n)	O(n)
extra memory per item	none (some spare capacity)	one reference (next)

? **Think:** print jobs are added at the back and removed at the front. Array or linked list? Why?

Where Linked Lists Are Used

IIT Madras Zanzibar

1 **Queues**

yesterday's LinkedQueue: no capacity limit, O(1) at both ends

2 **Playlists, browser history**

next / previous: the doubly linked list (Friday)

3 **Hash tables (Week 8)**

keys that collide in one slot are kept in a short chain

4 **collections.deque**

Python's deque is a linked list of blocks: O(1) at both ends

In practice: you rarely write one at work, but you use structures built on one every day.

Live Demo

IIT Madras Zanzibar

```
cd ~/PDS/Class_Demos/Week_04
python3 W04_Thu_Oct29_linked_list.py          # build, find, remove, pop, middle
python3 W04_Thu_Oct29_linked_list.py reverse  # reverse in place, round by round
python3 W04_Thu_Oct29_linked_list.py race     # insert at the front: list vs linked list
```

What to watch

1. Compare each printed chain with the slide pictures.
2. In the race, n doubles: the list's time grows about 4×, the linked list's about 2×.
3. That is $O(n^2)$ total against $O(n)$ total for n inserts at the front.

In-Class Exercise (6 minutes, in pairs)

IIT Madras Zanzibar

IN-CLASS TASK: draw first, then answer

1. The list is head → 3 → 8 → 5 → None. Write, in order, the pointer changes for push_front(1). Then for push_back(9).

2. With cur = head, what does this loop print for the list above? while cur is not None: print(cur.val); cur = cur.next.next

3. A one-node list holds 4. After pop_front(), give head and tail. What bug happens if tail is not updated?

4. You hold p (the node 8). Insert 6 right after it. Give the two steps in order. What is the cost?

5. Run the middle-finding loop (slow / fast) on 3 → 8 → 5 → 9. Where does slow stop?

Solution

IIT Madras Zanzibar

1 **1: push_front(1) then push_back(9)**

new.next = head; head = new → 1, 3, 8, 5. tail.next = new; tail = new → 3, 8, 5, 9.

2 **2: prints 3, 5, then crashes**

cur.next.next skips a node; at 5, cur.next is None and None.next raises AttributeError.

3 **3: head = None, tail = None**

if tail still points at the old node, push_back links after a node outside the list: the item vanishes.

4 **4: new.next = p.next; p.next = new**

6 points at 5 first, then 8 points at 6. Two steps: O(1).

5 **5: slow stops at 5**

(3,3) → (8,5) → (5,None). With 4 nodes slow ends at the second middle node.

Common Mistakes (Check These This Afternoon)

IIT Madras Zanzibar

1 **Moving head while walking**

use cur = head and move cur; head must stay at the front

2 **Wrong order of steps**

connect the new node first, then move head, tail or p.next

3 **Forgetting tail**

update tail when the list empties and when the last node changes

4 **Not testing the empty list**

every method: empty, one node, first, last, long

Summary and Exam Checkpoints

IIT Madras Zanzibar

Node and reference

a node holds val and next; a variable holds an arrow to an object

Traversal

cur = head; while cur is not None: …; cur = cur.next. O(n)

Pointer surgery

front and after-a-node inserts are O(1); connect first, then redirect

Costs

array: fast by position. Linked list: fast at the front and in the middle

Next

Today, 2:00 pm: Lab 3: you implement the circular queue and the linked list

Fri 30 Oct: Doubly linked lists, sentinels, circular lists, and an LRU cache

References and Further Reading

[1] Canning, J., Broder, A. & Lafore, R. (2022). Data Structures & Algorithms in Python. Addison-Wesley. Chapter 5 (Linked Lists).

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2022). Introduction to Algorithms, 4th ed. MIT Press. Section 10.2 (Linked lists).

[3] Goodrich, M. T., Tamassia, R. & Goldwasser, M. H. (2013). Data Structures and Algorithms in Python. Wiley. Chapter 7 (Linked Lists).

[4] Python documentation: Data model (objects, values and types: references); collections.deque.