CSC 148: Introduction to Computer Science

Week 5

Linked list insertion and deletion

It's all about the links.

University of Toronto Mississauga,

Department of Mathematical and Computational Sciences

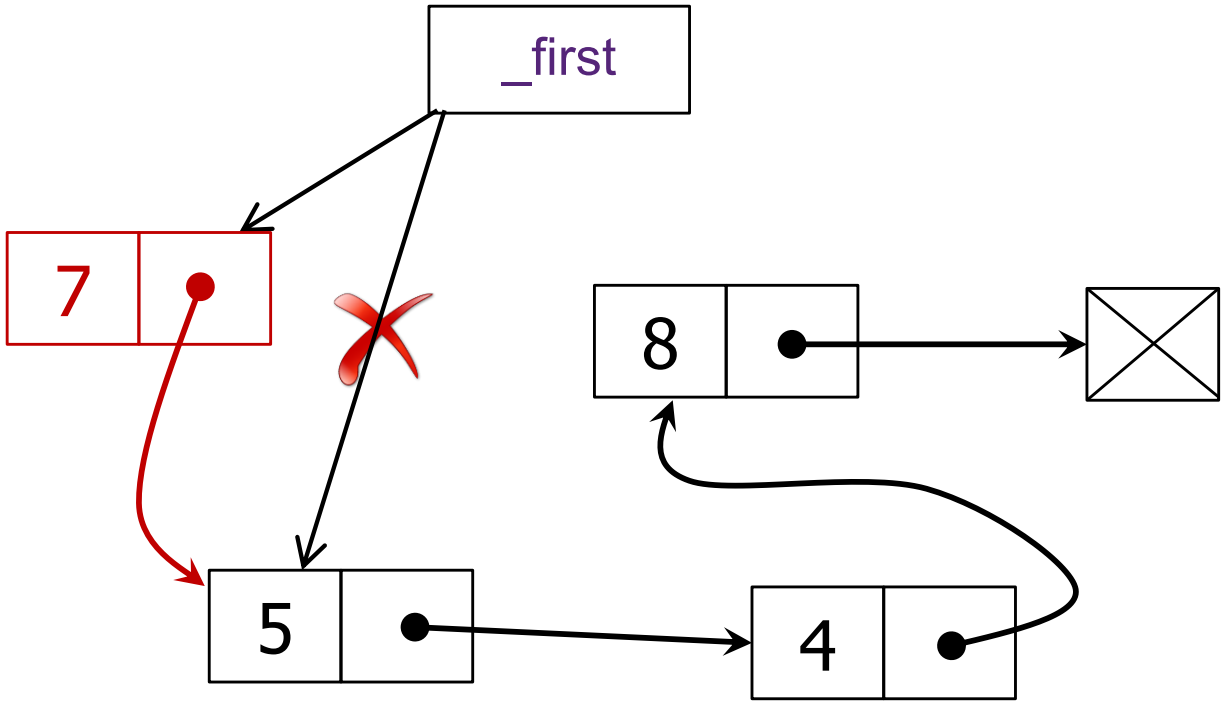
Insert

- We might want to implement all sorts of insert variations depending on what operations we want the linked list to support, e.g.
  - Prepend
  - Append
  - Insert at a given index

- Use diagrams to visualize such operations!

Prepend (insert at the front)

- Easy: simply adjust the _first reference



append

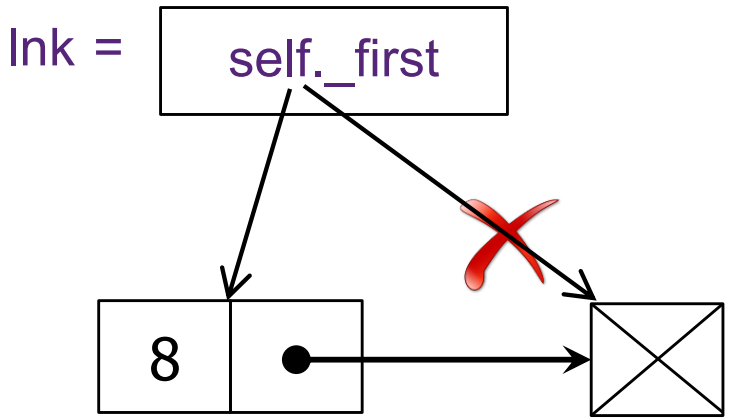
- We'll need to change...
  - some node inside the list
    - possibly the last node
  - possibly _first .. why?

- Always draw diagrams!

append

- First node being appended
  - Sort of similar to prepend in this particular corner case...

List is initially empty. Appending a new node.

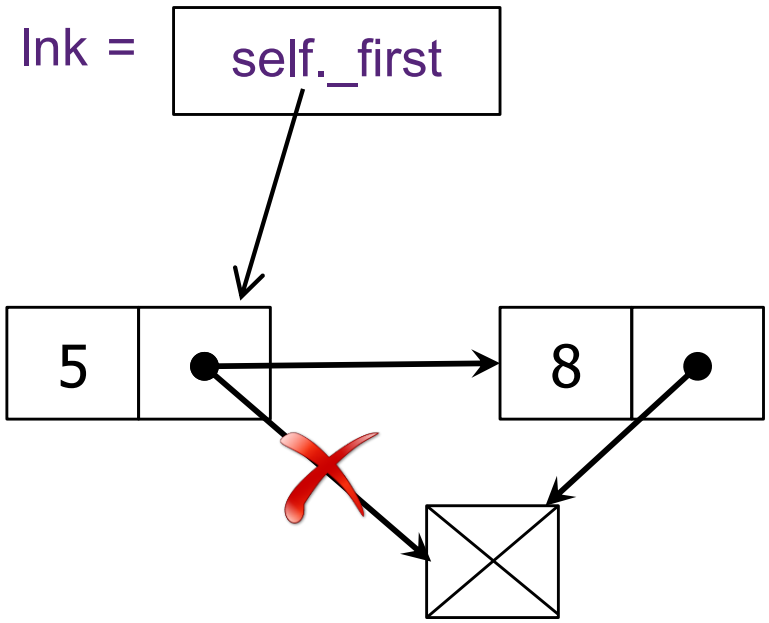


- Always draw diagrams!

append

- First node being appended

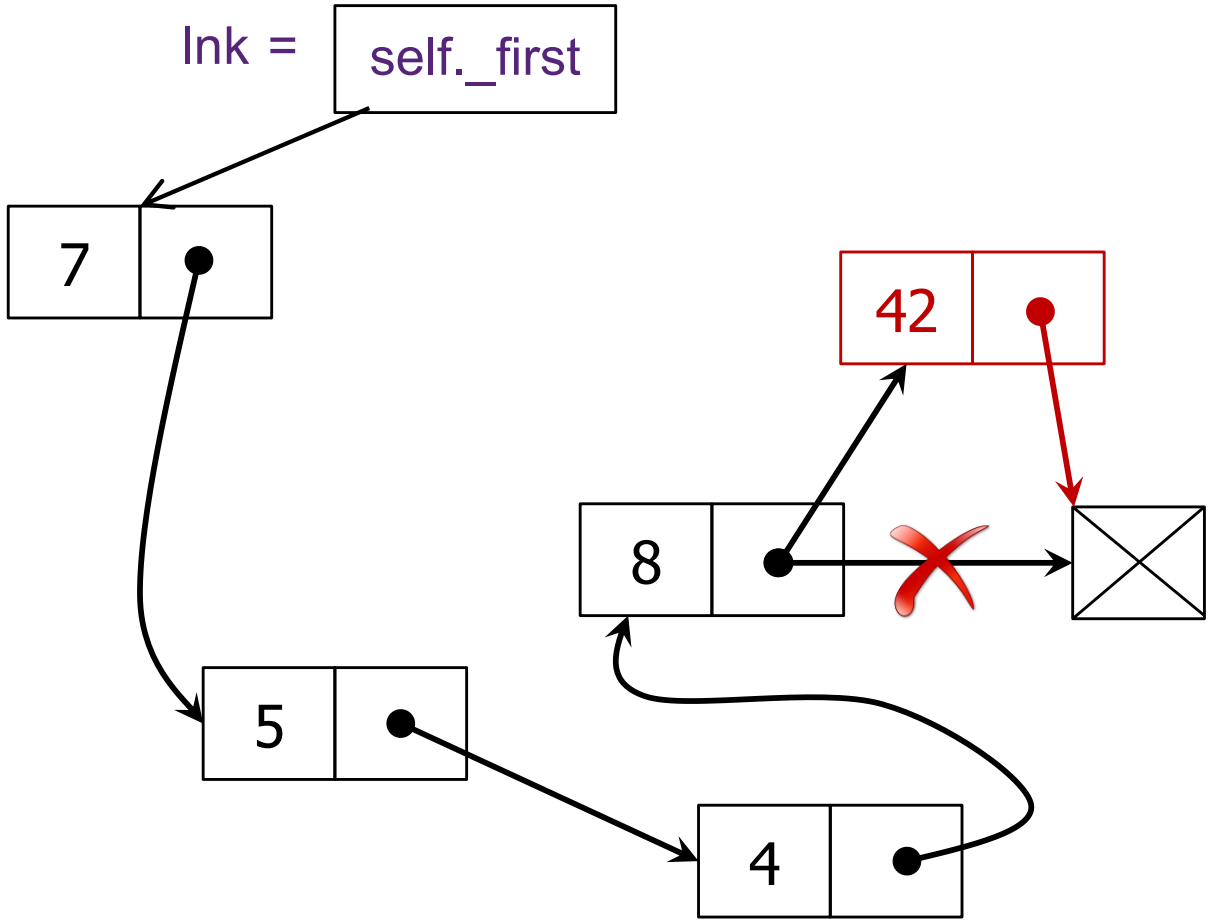
List has one element (5). Appending a new node (8).



- Always draw diagrams!

append

- Several nodes in the list:



- Always draw diagrams!

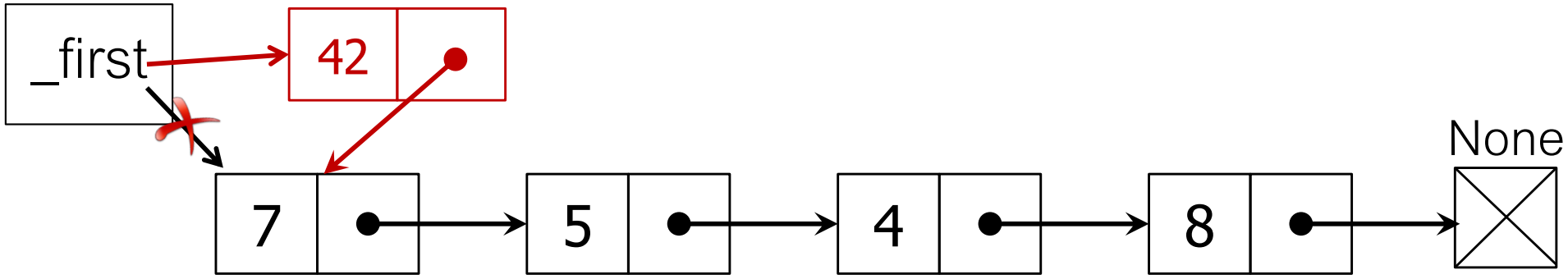
Insert a node (at an index in the list)

- Worksheet ...

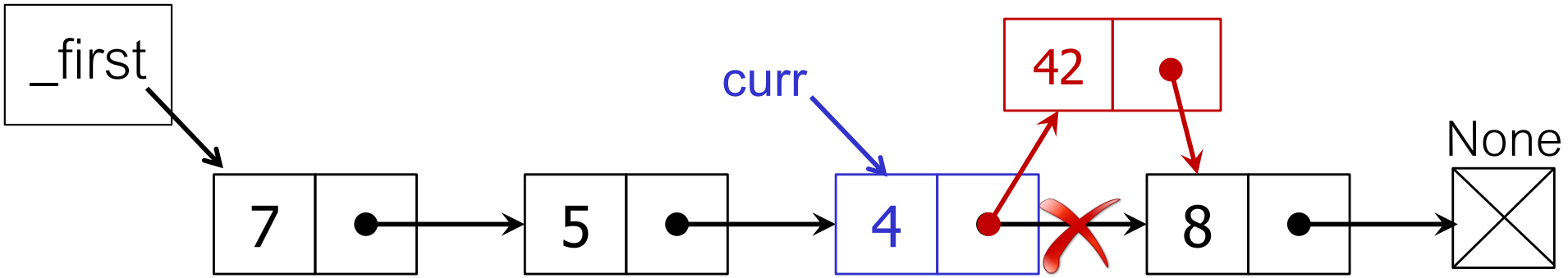
```
def insert(self, index: int, item: Any) -> None:
    """Insert the given item at the given index.

    Raise IndexError if index > len(self) or index < 0.
    Note that adding to the end of the list is okay.
    """
```

Recap

1. Figure out when we need to modify self._first vs. a _Node in the list.



2. When index > 0, iterate to the (index-1)th node and update links.



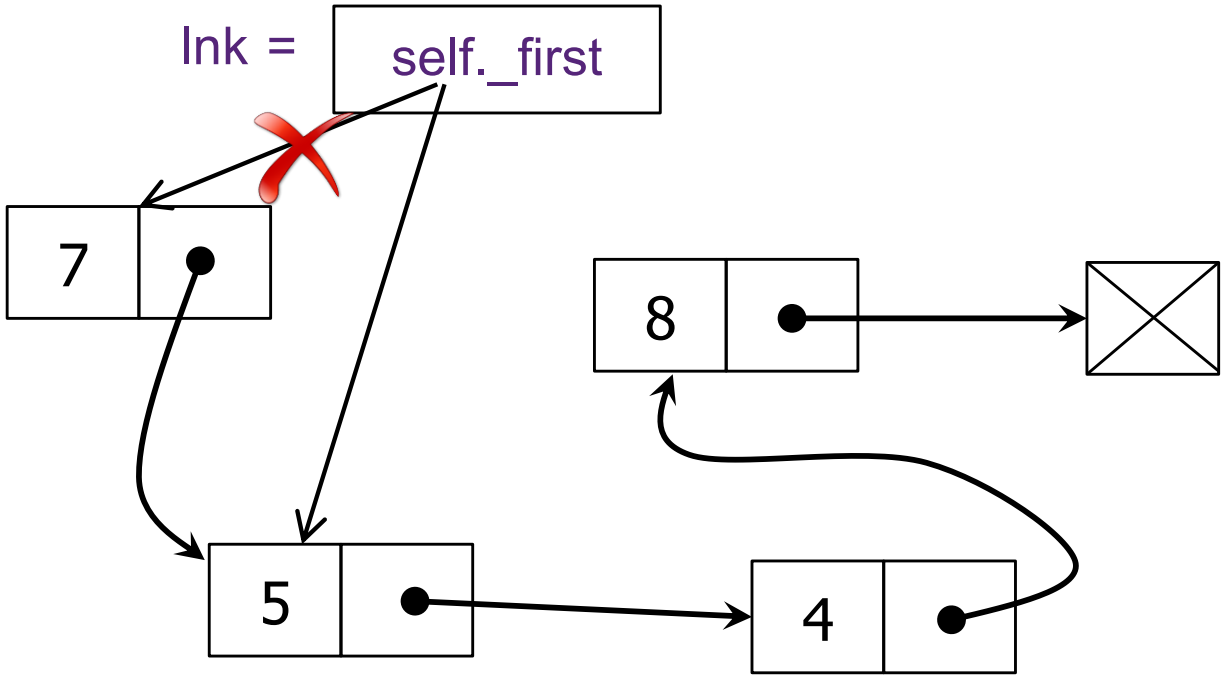
Delete / pop

- We might want to implement all sorts of delete variations depending on what operations we want the linked list to support, e.g.
  - Delete from the front
  - Delete from the back
  - Delete (pop) from any index in the list

- Ultimately, the "pop from an index" version covers all cases
  - But, let's visualize such operations first ...

delete_from_front

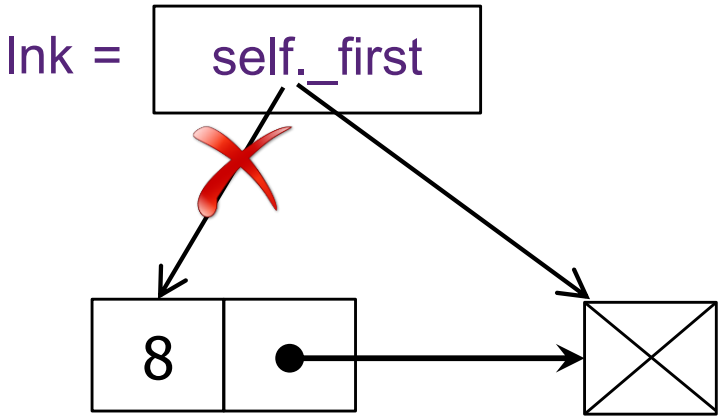
- Easy: make _first reference the second node (garbage collection takes care of former first node automatically)
  - No need to walk the list ...



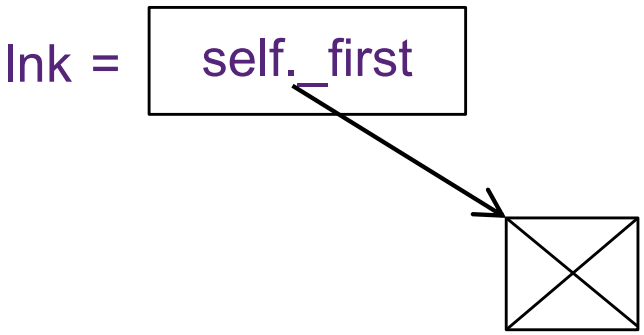
delete_from_front

- Easy: make _first reference the second node (garbage collection takes care of former first node automatically)
  - Consider corner cases though ...

What if only 1 node?

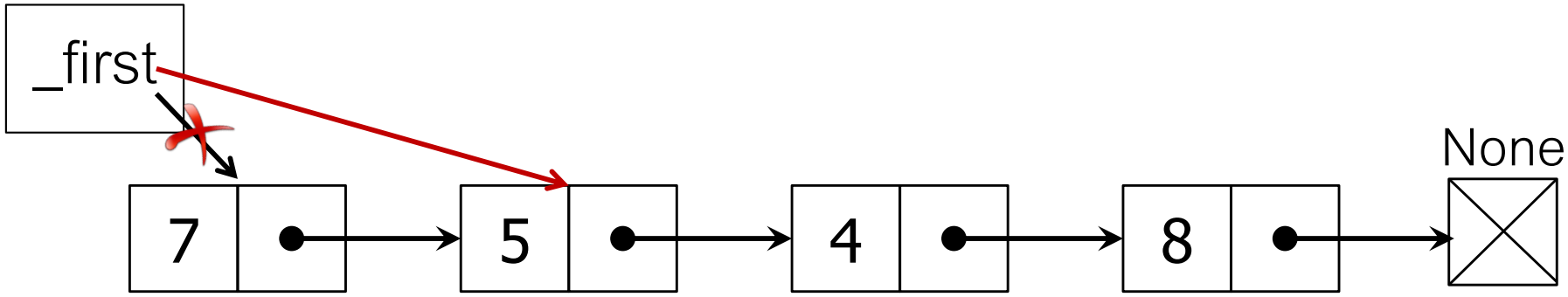


What if list is empty?

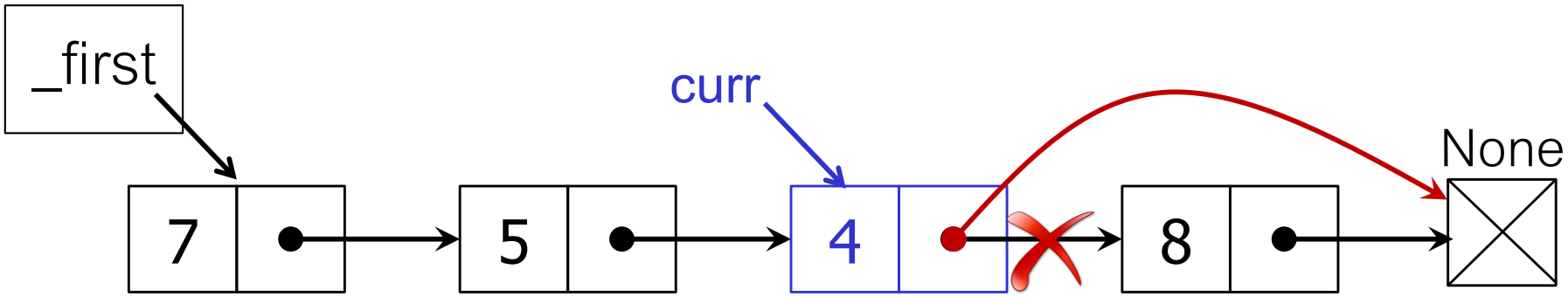


Same Key Ideas!

1. Figure out when we need to modify `self._first` vs. a `_Node` in the list.



2. When `index` > 0, iterate to the `(index-1)`th node and update links.



The “Problem of Previous”

- Strategy #1: iterate to the node *before* the desired position.

```
i = 0
curr = self._first
while not (curr is None or i == index - 1):
    curr = curr.next
    i += 1
```

The “Problem of Previous”

- Strategy #2: track the previous node explicitly

```
i = 0
prev = None
curr = self._first
while not (curr is None or i == index):
    prev, curr = curr, curr.next
    i += 1
```