DATA STRUCTURES AND ALGORITHMS

Lecture 9: Variations in Linked List

Lecturer: Mohsin Abbas

National University of Modern Languages, Islamabad

LINKED LIST – ADVANTAGES

- Access any item as long as external link to first item maintained
- *Insert* new item *without shifting*
- *Delete* existing item without shifting
- Can expand/contract (flexible) as necessary

LINKED LIST – DISADVANTAGES

- Overhead of links
  - Used only internally, pure overhead
- If dynamic, then it must provide *Destructor*
- No longer have *direct access* to each element of the list
  - Many sorting algorithms need direct access
  - Binary search needs direct access
- Access of nth item now less efficient
  - Must go through first element, then second, and then third, etc.

LINKED LIST – DISADVANTAGES

- List-processing algorithms that require fast access to each element cannot be done as efficiently with linked lists
- Consider adding an element at the end of the list

Array	Linked List
`a[size++] = value`	

LINKED LIST – DISADVANTAGES

- List-processing algorithms that require fast access to each element cannot be done as efficiently with linked lists
- Consider adding an element at the end of the list

Array	Linked List
`a[size++] = value`	Get a new node; Set data part = value next part = *null_value*

LINKED LIST – DISADVANTAGES

- List-processing algorithms that require fast access to each element cannot be done as efficiently with linked lists
- Consider adding an element at the end of the list

Array	Linked List
`a[size++] = value`	Get a new node; Set data part = value next part = *null_value* If list is empty Set head to point to new node

LINKED LIST – DISADVANTAGES

- List-processing algorithms that require fast access to each element cannot be done as efficiently with linked lists
- Consider adding an element at the end of the list

Array	Linked List
`a[size++] = value`	Get a new node; Set data part = value next part = *null_value* If list is empty Set head to point to new node Else Traverse list to find last node Set next part of last node to point to new node

This is the inefficient part

SOME APPLICATIONS

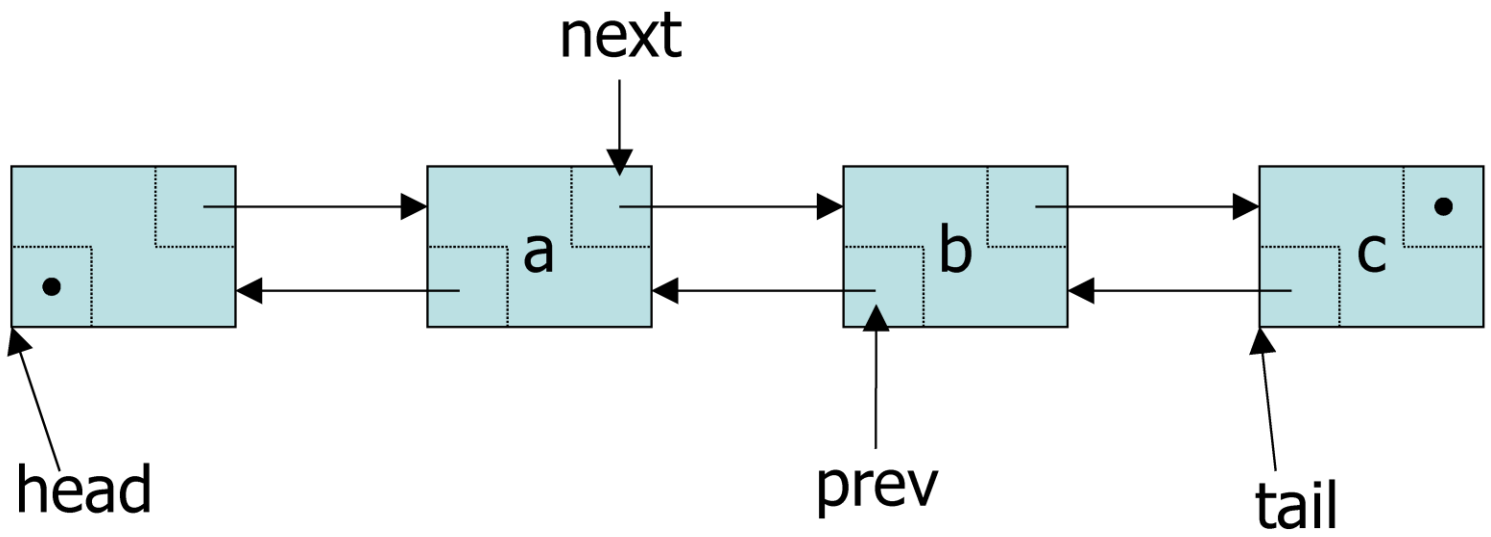
- Applications that maintain a Most Recently Used (MRU) list
  - For example, a linked list of file names
- Undo functionality in Photoshop or Word
  - A linked list of state
- A list in the GPS of the turns along your route

- Question to search:
  - Can we traverse the linked list in the reverse direction!

DOUBLY LINKED LIST

DOUBLY LINKED LIST

- Every node contains the address of the previous node except the first node
  - Both forward and backward traversal of the list is possible



NODE CLASS

- `DoubleListNode` class contains three data members
  - `data`: double-type data in this example
  - `next`: a pointer to the next node in the list
  - `prev`: a pointer to the previous node in the list

```
class DoubleListNode {
    public:
        double data;                // data
         DoubleListNode* next;      // pointer to next
         DoubleListNode* prev;      // pointer to previous
};
```

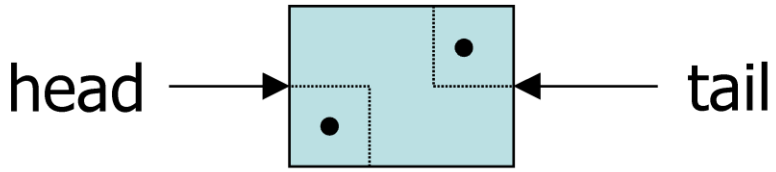
LIST CLASS

- List class contains two pointers
  - head: a pointer to the first node in the list
  - tail: a pointer to the last node in the list
  - Since the list is empty initially, head and tail are set to NULL

```
class List {
    public:
        List(void) { head = NULL; tail = NULL; }    // constructor
        ~List(void);                                // destructor

    .   .   .
    private:
        DoubleListNode* head;
        DoubleListNode* tail;
};
```

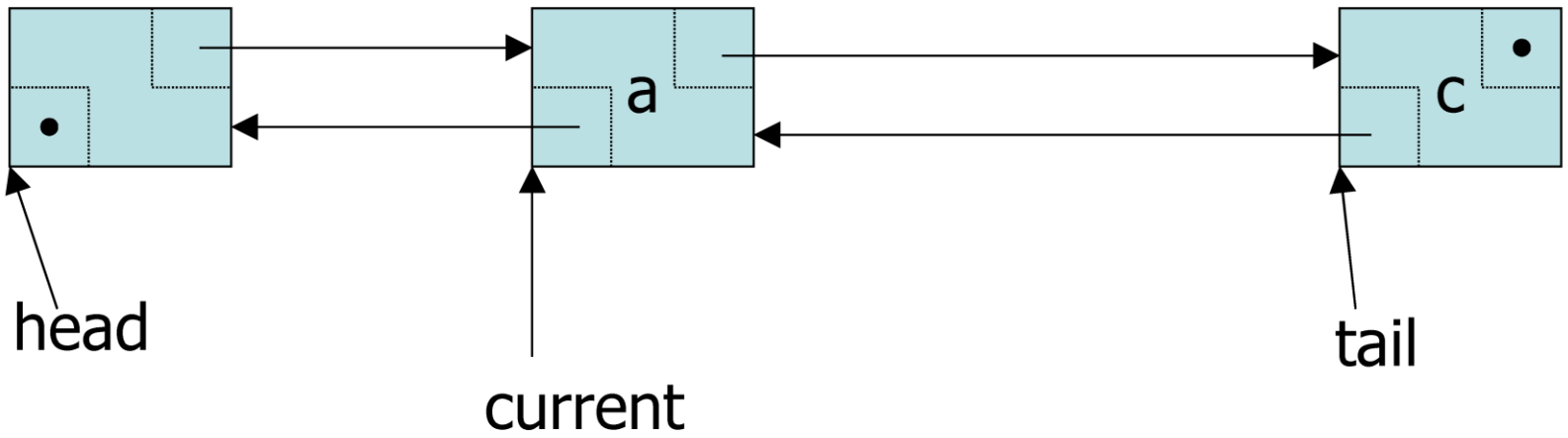
ADDING FIRST NODE




```
// Adding first node
head = new DoubleListNode;
head->next = null;
head->prev = null;
tail = head;
```

INSERTING A NODE IN DOUBLY LINKED LIST

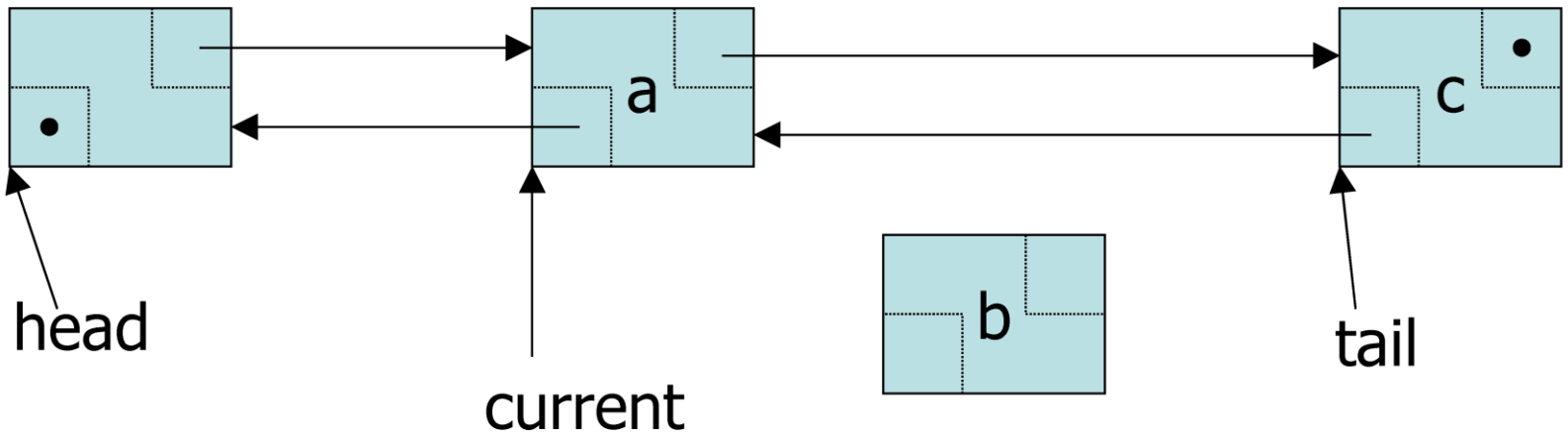
- To add a new item after the linked list node pointed by `current`




```
newNode = new DoublyLinkedListNode
newNode->prev = current;
newNode->next = current->next;
newNode->prev->next = newNode;
newNode->next->prev = newNode;
current = newNode;
```

INSERTING A NODE IN DOUBLY LINKED LIST

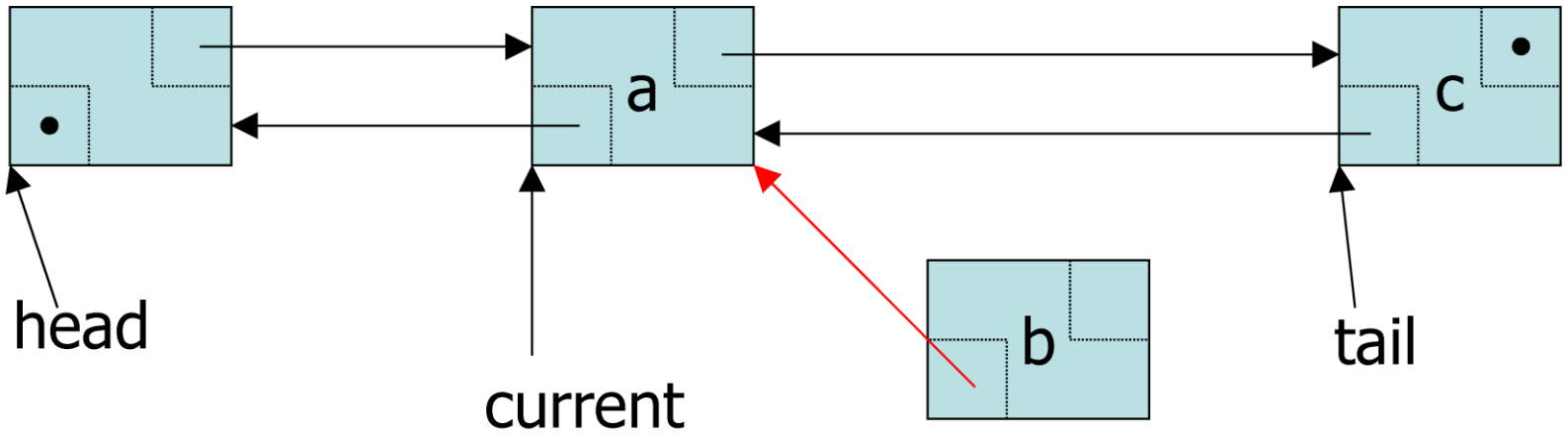
- To add a new item after the linked list node pointed by `current`




```
newNode = new DoublyLinkedListNode
newNode->prev = current;
newNode->next = current->next;
newNode->prev->next = newNode;
newNode->next->prev = newNode;
current = newNode;
```

INSERTING A NODE IN DOUBLY LINKED LIST

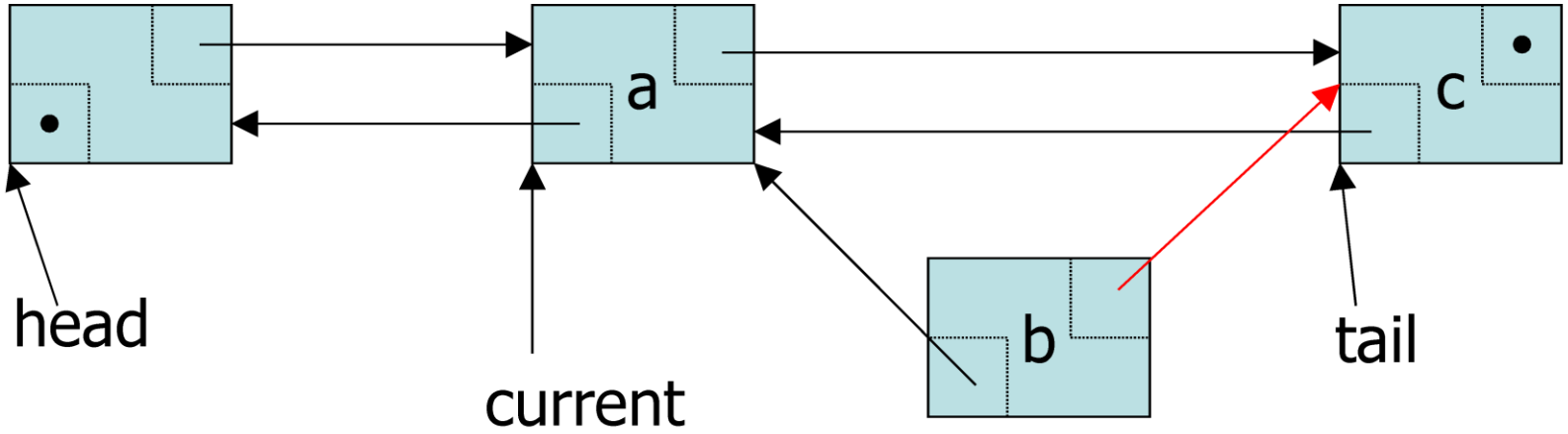
- To add a new item after the linked list node pointed by `current`




```
newNode = new DoublyLinkedListNode
newNode->prev = current;
newNode->next = current->next;
newNode->prev->next = newNode;
newNode->next->prev = newNode;
current = newNode;
```

INSERTING A NODE IN DOUBLY LINKED LIST

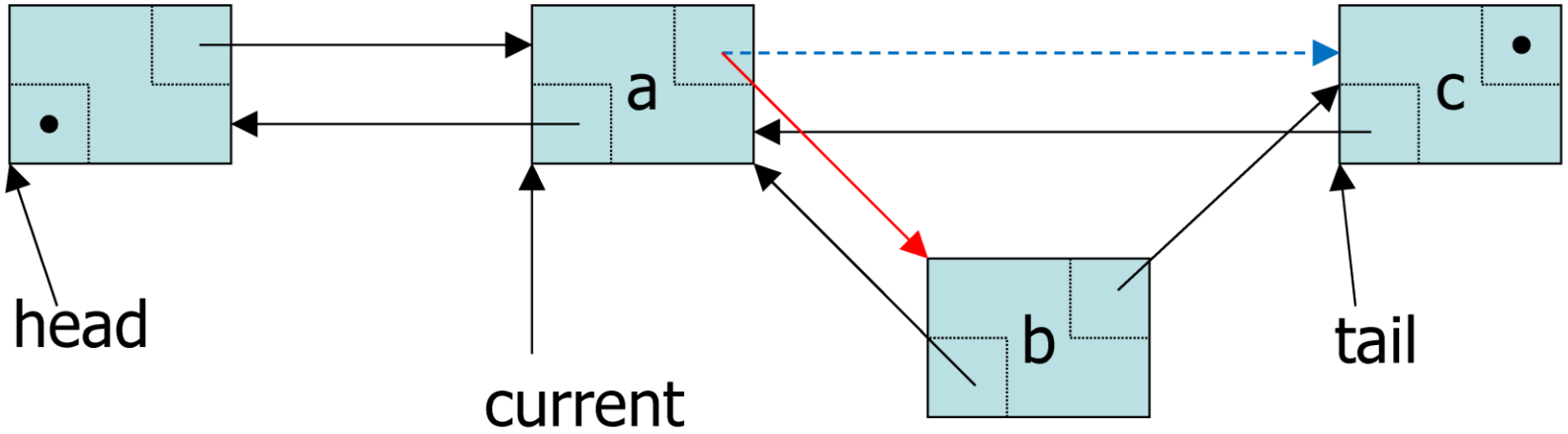
- To add a new item after the linked list node pointed by `current`




```
newNode = new DoublyLinkedListNode
newNode->prev = current;
newNode->next = current->next;
newNode->prev->next = newNode;
newNode->next->prev = newNode;
current = newNode;
```

INSERTING A NODE IN DOUBLY LINKED LIST

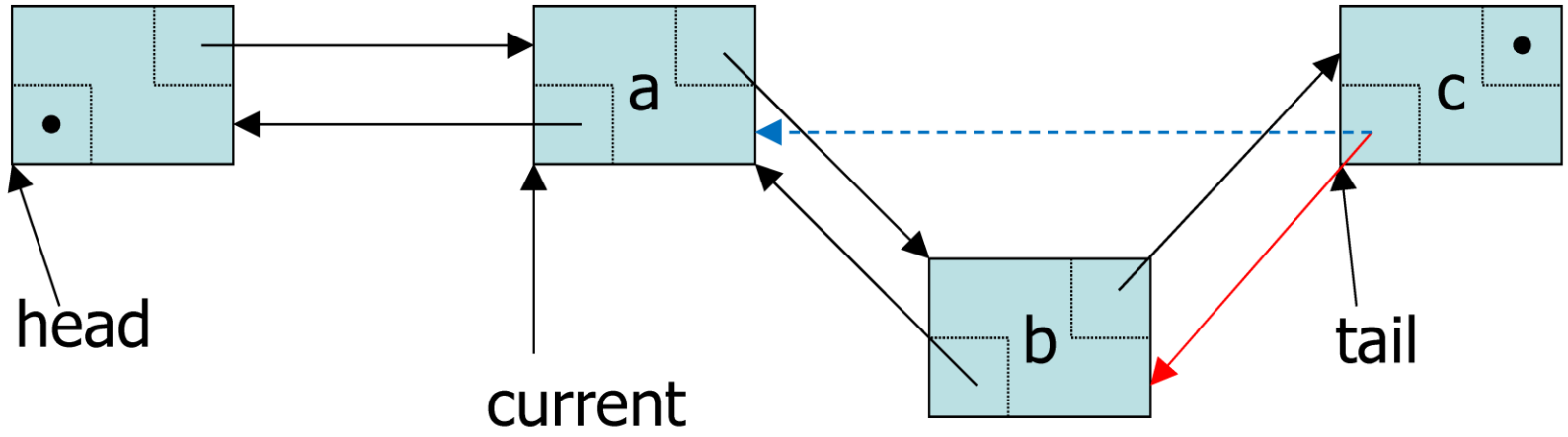
- To add a new item after the linked list node pointed by `current`




```
newNode = new DoublyLinkedListNode
newNode->prev = current;
newNode->next = current->next;
newNode->prev->next = newNode; //Current->next = newNode
newNode->next->prev = newNode;
current = newNode;
```

INSERTING A NODE IN DOUBLY LINKED LIST

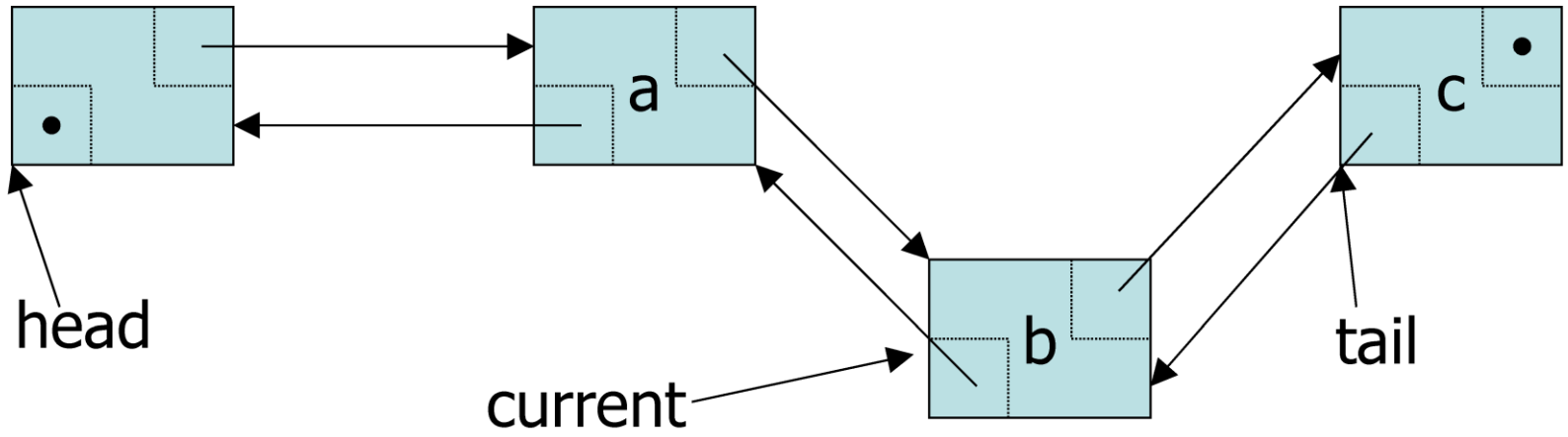
- To add a new item after the linked list node pointed by `current`




```
newNode = new DoublyLinkedListNode
newNode->prev = current;
newNode->next = current->next;
newNode->prev->next = newNode;
newNode->next->prev = newNode;
current = newNode;
```

INSERTING A NODE IN DOUBLY LINKED LIST

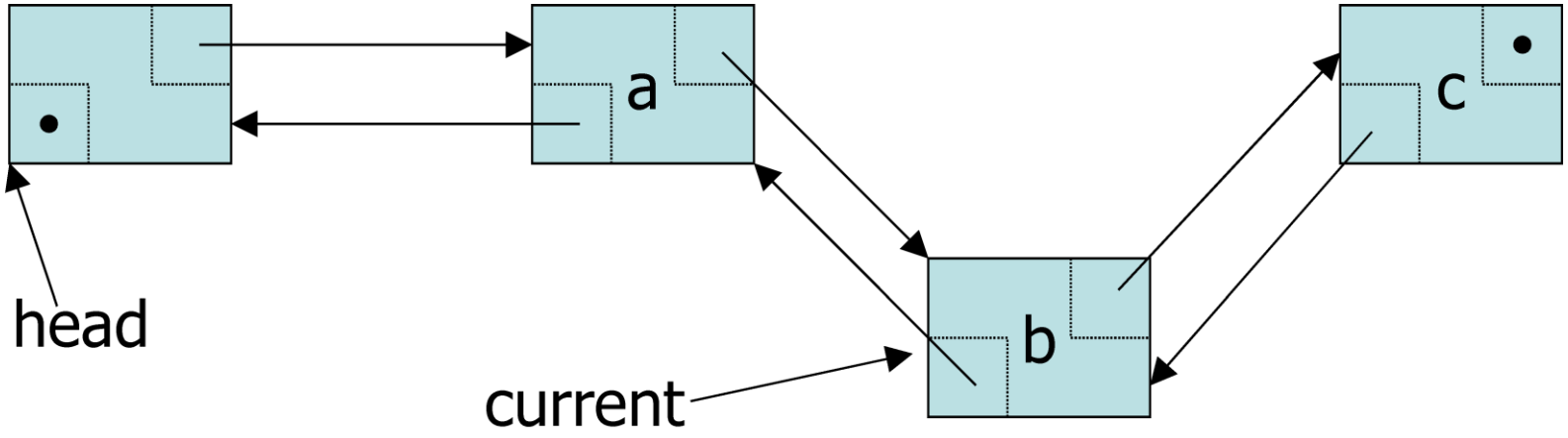
- To add a new item after the linked list node pointed by `current`




```
newNode = new DoublyLinkedListNode
newNode->prev = current;
newNode->next = current->next;
newNode->prev->next = newNode;
newNode->next->prev = newNode;
current = newNode;
```

DELETING A NODE IN DOUBLY LINKED LIST

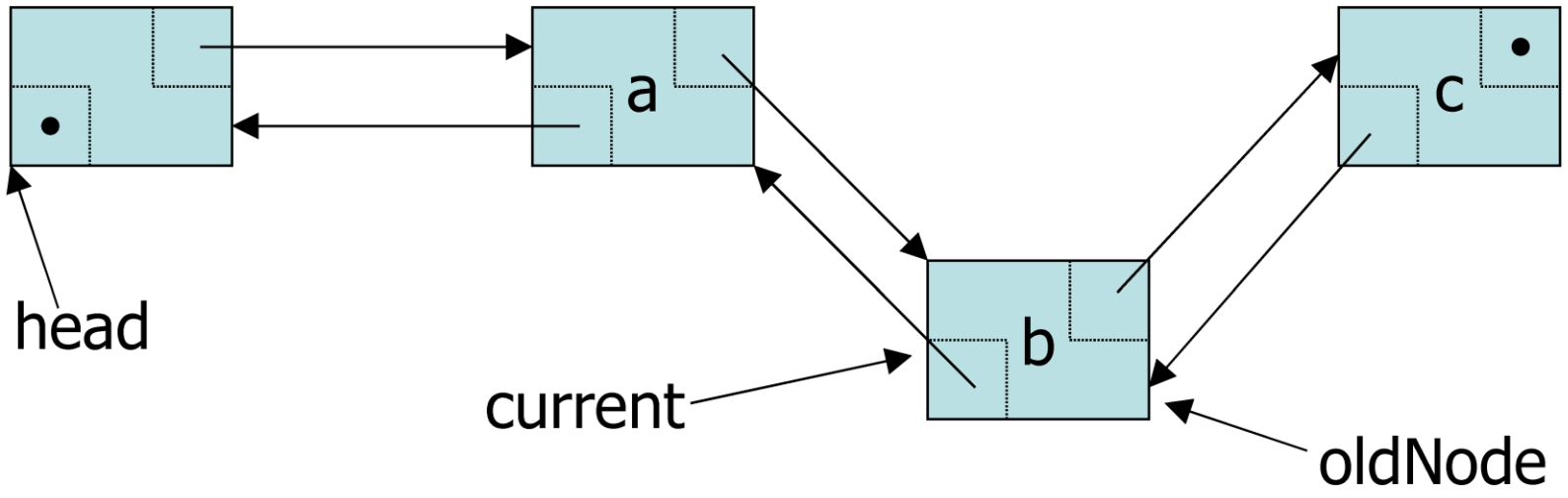
- Suppose `current` points to the node to be deleted from the list




```
oldNode = current;
oldNode->prev->next = oldNode->next;
oldNode->next->prev = oldNode->prev;
current = oldNode->prev;
delete oldNode;
```

DELETING A NODE IN DOUBLY LINKED LIST

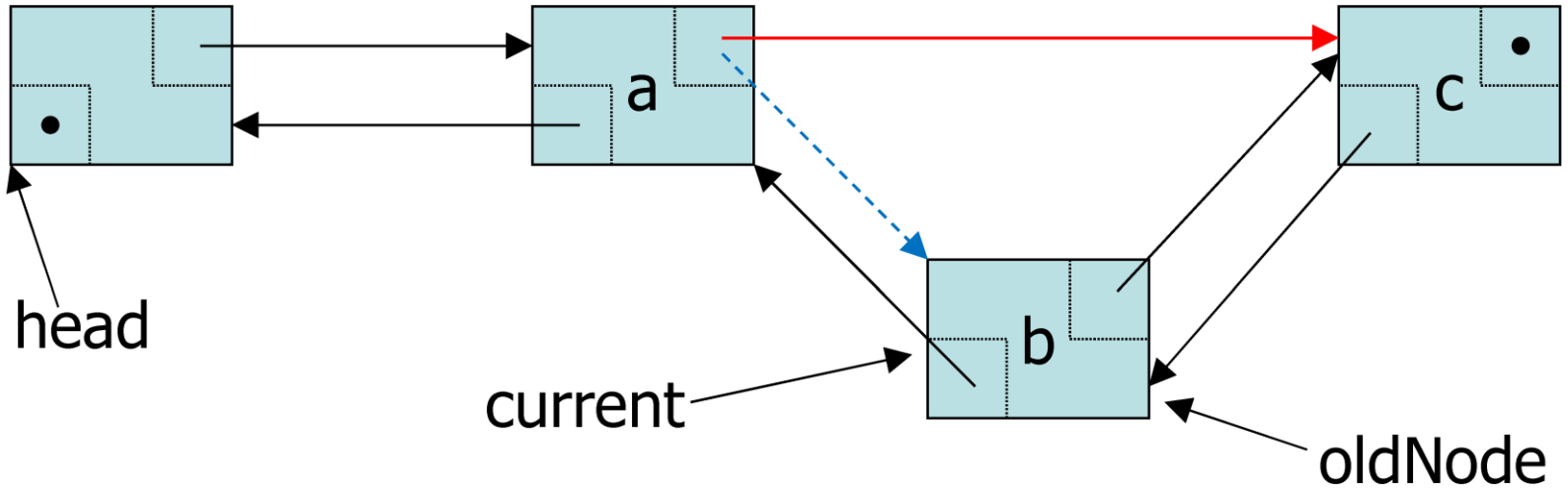
- Suppose `current` points to the node to be deleted from the list




```
oldNode = current;
oldNode->prev->next = oldNode->next;
oldNode->next->prev = oldNode->prev;
current = oldNode->prev;
delete oldNode;
```

DELETING A NODE IN DOUBLY LINKED LIST

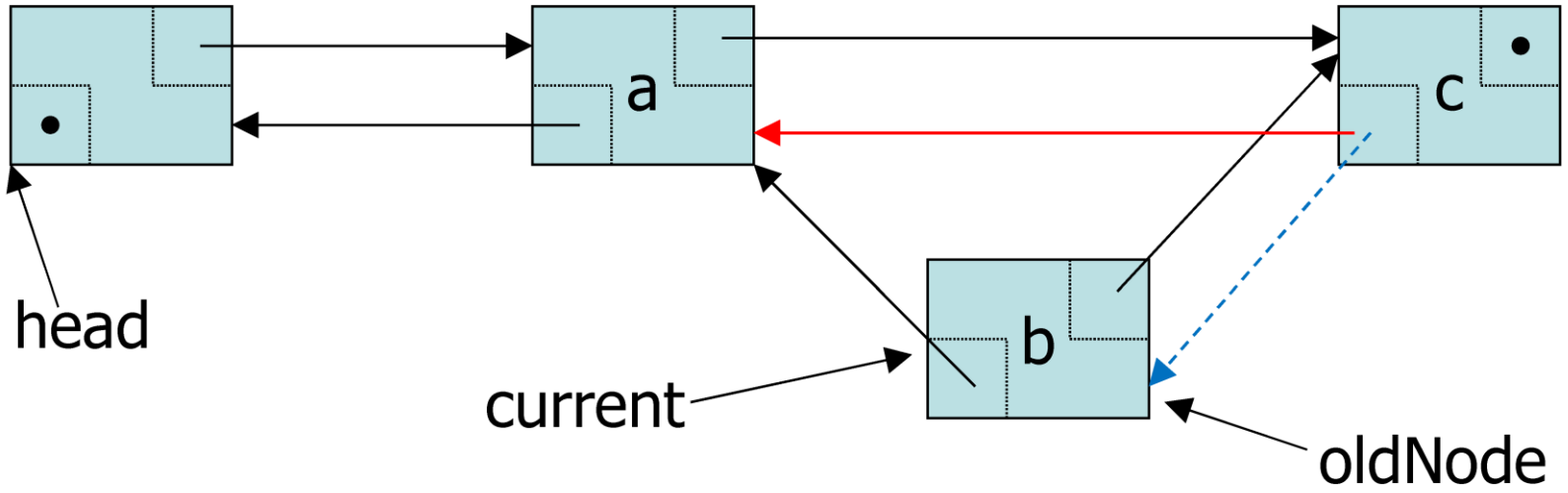
- Suppose `current` points to the node to be deleted from the list




```
oldNode = current;
oldNode->prev->next = oldNode->next;
oldNode->next->prev = oldNode->prev;
current = oldNode->prev;
delete oldNode;
```

DELETING A NODE IN DOUBLY LINKED LIST

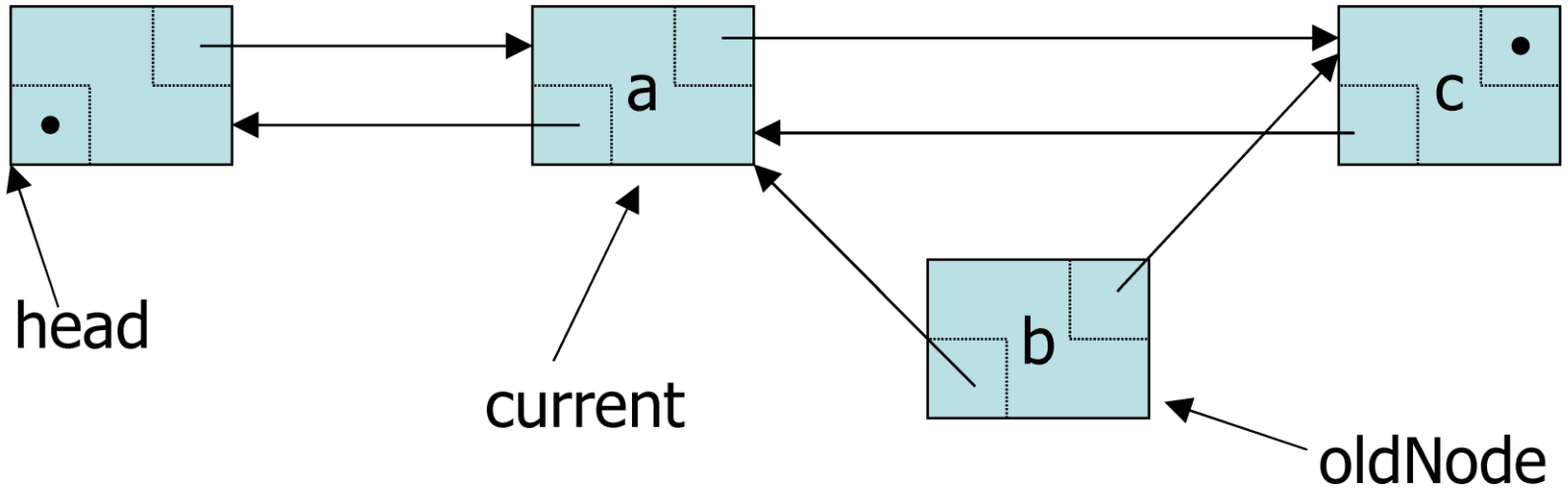
- Suppose `current` points to the node to be deleted from the list




```
oldNode = current;
oldNode->prev->next = oldNode->next;
oldNode->next->prev = oldNode->prev;
current = oldNode->prev;
delete oldNode;
```

DELETING A NODE IN DOUBLY LINKED LIST

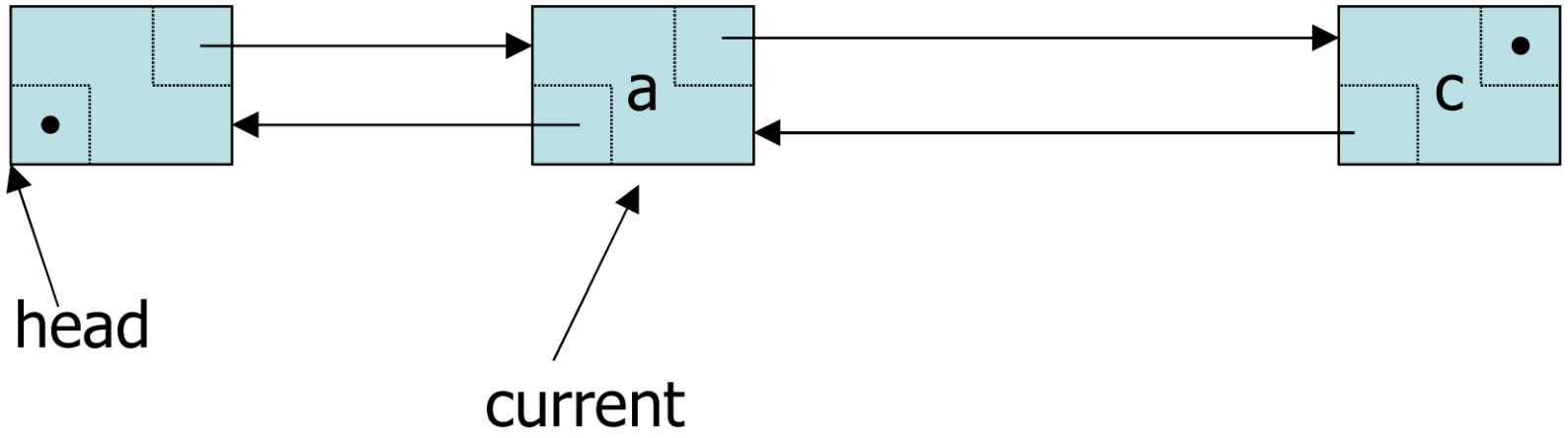
- Suppose `current` points to the node to be deleted from the list




```
oldNode = current;
oldNode->prev->next = oldNode->next;
oldNode->next->prev = oldNode->prev;
current = oldNode->prev;
delete oldNode;
```

DELETING A NODE IN DOUBLY LINKED LIST

- Suppose `current` points to the node to be deleted from the list

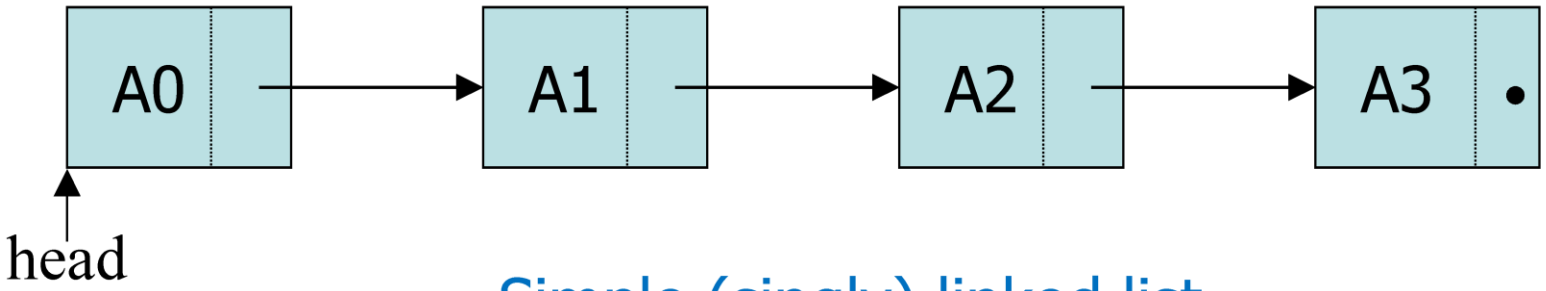



```
oldNode = current;
oldNode->prev->next = oldNode->next;
oldNode->next->prev = oldNode->prev;
current = oldNode->prev;
delete oldNode;
```

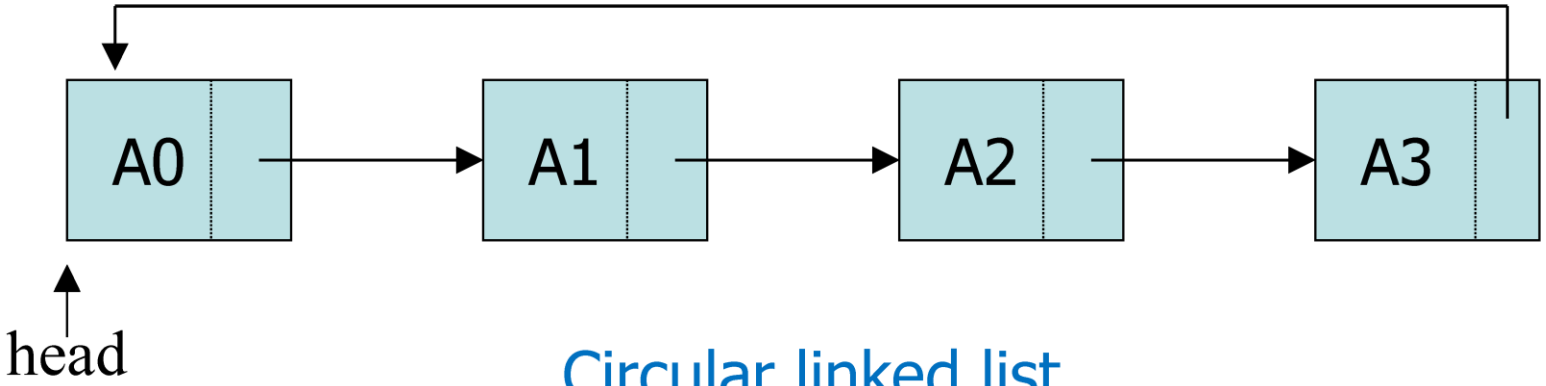
CIRCULAR LINKED LIST

CIRCULAR LINKED LIST

- A linked list in which the last node points to the first node



Simple (singly) linked list



Circular linked list

ADVANTAGES OF CIRCULAR LINKED LIST

- Whole list can be traversed by starting from any point
  - Any node can be starting point
  - What is the stopping condition?
- Fewer special cases to consider during implementation
  - All nodes have a node before and after it
- Used in the implementation of other data structures
  - Circular linked lists are used to create circular queues
  - Circular doubly linked lists are used for implementing Fibonacci heaps

DISADVANTAGES OF CIRCULAR LINKED LIST

- Finding end of list and loop control is harder
  - No NULL's to mark beginning and end

CONCLUSION

- In this lecture we have studied:
  - Variations in Linked List
  - Doubly Linked List
  - Circular Linked List

Question?