LECTURE-5

Linear Data Structures

Linked Lists

CS202: Data Structures (Fall 2025)

Dr Maryam Abdulghafur, Momina Khan

Department of Computer Science, SBASSE

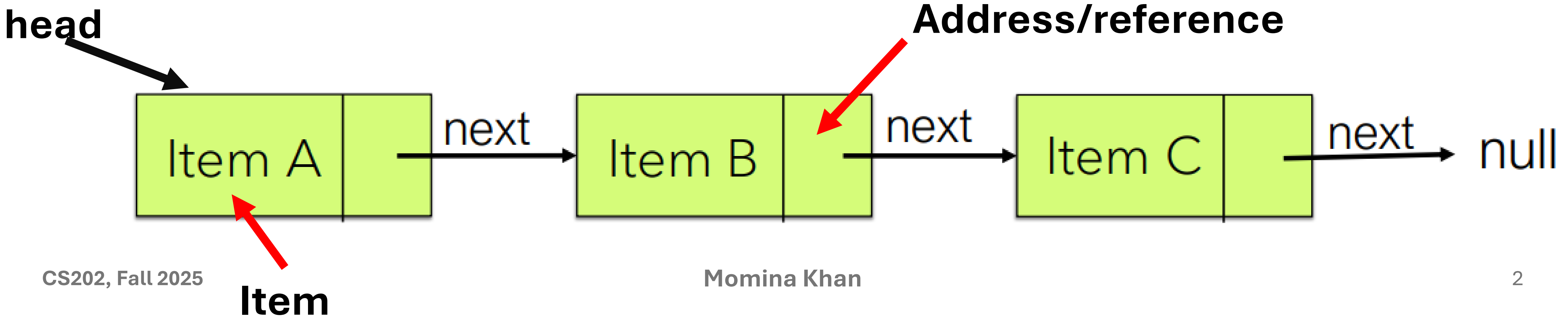
Quick recap

Can we stretch an array? NO

- Dynamic Array (A hack around rigid sized array)
- Growing an array means
  - making a bigger
  - Copying contents
  - Discarding old smaller one

Costly!

- Linked List



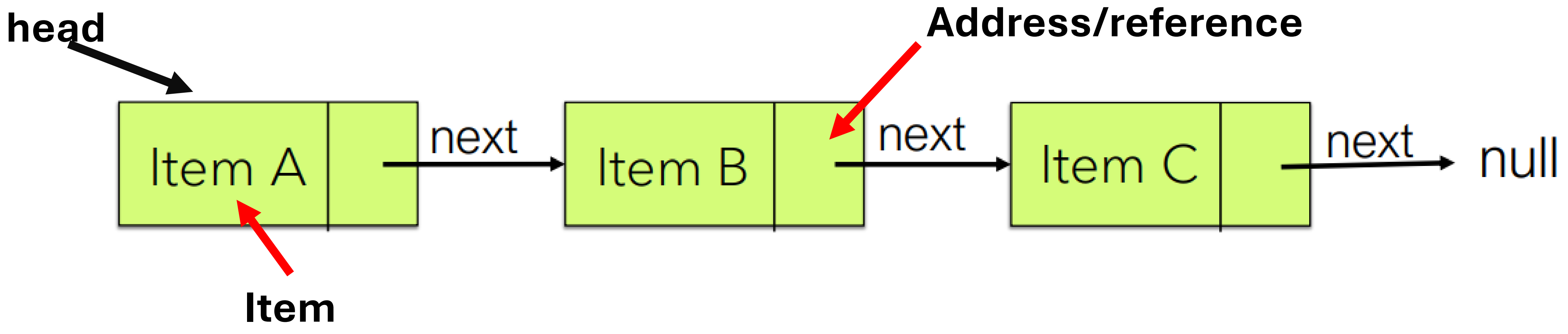
Agenda

- Linked Lists
- How useful are Singly Linked Lists?
- How do Singly LLs compare with Doubly LLs?
- Circular Lists

List (Linked List)

Terms to remember:

- Node,
- link(next pointer),
- <key, value> pair



Array vs Linked List

Array	Linked List
Contiguous Storage	Non-Contiguous
Fixed Size	Grows with insertions (shrinks too!)
Indexed access	Sequential access (follow the links)
Memory allocation can be compile time and **runtime**	Memory allocation is **runtime**

List (Linked List)

- Pros:
  - Size flexible; grows on demand
  - No need to allocate extra memory
  - Easier to add elements to the middle
- Cons:
  - Sequential Access (Has to traverse to the end)
  - Cache unfriendliness
  - Memory overhead (extra pointers!)

Caveat!

List ADT

```
template <typename T>
class List {
Public:
        virtual void insert( const T& element) = 0;

        virtual void delete( const T& element) = 0;

        virtual T find(const T& element ) = 0;

        virtual T get( int index) = 0;

        virtual bool isEmpty() = 0;

        virtual bool size() = 0;
```

You can enhance this interface with more functions like:

- **insertAtHead()**
- **deleteAtTail()**

List (Linked List)

```
//You have access to the head pointer.
void List<T>::insertAtHead(T object) {

// code comes here!



}
```

Always draw a diagram!

Corner cases: List empty, deleting a last node etc.

List (Unsorted Linked List)

find(k)	O(.)
insertAtStart(int k)	O(.)
insertAtLoc(int k, int loc)	O(.)
insertAtEnd(int k)	O(.)
deleteFromStart()	O(.)
deleteFromLoc(int loc)	O(.)

Find the worst-case time complexity in terms of O(.)

List (Unsorted Linked List)

find(k)	O(N)
insertAtStart(int k)	O(1)
insertAtLoc(int k, int loc)	O(N)
insertAtEnd(int k)	O(N)
deleteFromStart()	O(1)
deleteFromLoc(int loc)	O(N)

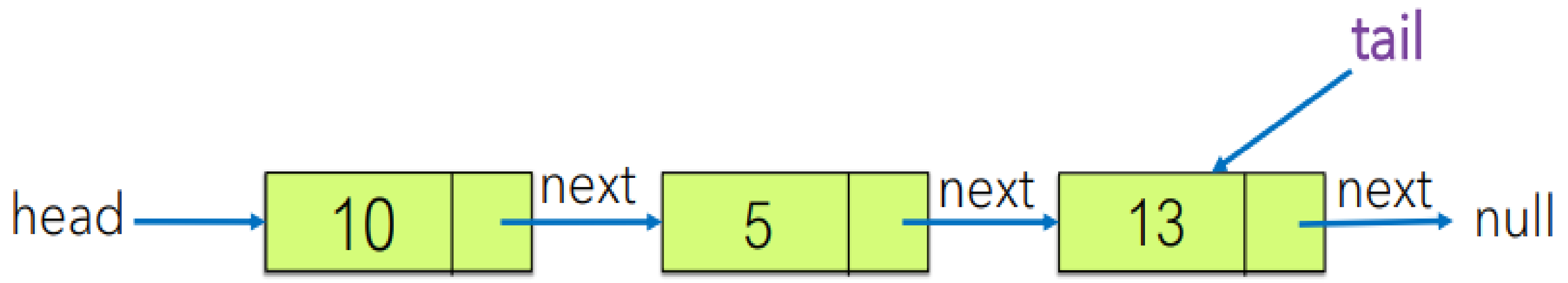
Most operations are linear since they warrant a list traversal!

List (Linked List)

Can we improve our insertAtEnd() time to something better than linear??

Introduce a second pointer to the end of the list called a ...

tail pointer



What would insertAtEnd(data) function look like with our new design?

List (Linked List)

```
//You have access to the head pointer and tail pointer.
void List<T>::insertAtEnd(T object) {

// code comes here!



}
```

Always draw a diagram!

Corner cases: List empty, deleting a last node etc.

List (sorted Linked List)

Which of the following describes the time complexity of the insert(k), delete(k), deleteEnd() and find(k) in a sorted singly linked list?

find(k)	O(.)
insert(k)	O(.)
delete(k)	O(.)
deleteEnd()	O(.)

Use the best possible algorithm (e.g., insertion at the beginning vs. end) to find the time complexity in terms of O.

List (sorted Linked List)

Which of the following describes the time complexity of the insert(k), delete(k), deleteEnd() and find(k) in a sorted singly linked list?

find(k)	O(N)
insert(k)	O(N)
delete(k)	O(N)
deleteEnd()	O(N)

Use the best possible algorithm (e.g., insertion at the beginning vs. end) to find the time complexity in terms of O.

Singly Linked List – Some Observations

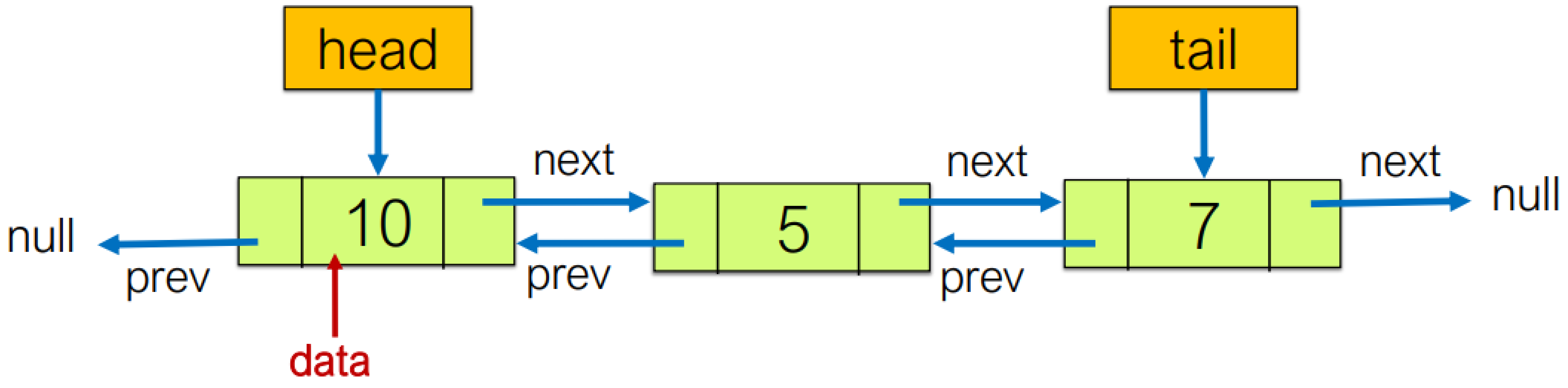
What is the time complexity (best case scenario)

- for inserting an element to the end? **O(1)**
- for inserting an element at the second-last position? **O(N)**
- for deleting an element form the end? **O(1)**
- for deleting an element from the second-last position? **O(N)**

Running Time is dominated by linear uni-directional traversal!

Can we do any better?

Doubly Linked List



Can you spot the differences between this and a singly linked list?

Doubly Linked List

- **How would the node structure change?**
- **What benefits are we deriving from this change?**
- **What remains the same? ☺**

List (Unsorted Doubly Linked List)

find(k)	**O(.)**
insertAtStart(int k)	O(.)
insertAtLoc(int k, int loc)	O(.)
insertAtEnd(int k)	O(.)
deleteFromStart()	O(.)
deleteFromLoc(int loc)	O(.)

Find the worst-case time complexity in terms of O(.)

List (Unsorted Doubly Linked List)

find(k)	O(N)
insertAtStart(int k)	O(1)
insertAtLoc(int k, int loc)	O(N)
insertAtEnd(int k)	O(1)
deleteFromStart()	O(1)
deleteFromLoc(int loc)	O(N)

Some operations are still linear since they warrant a list traversal!

List (Sorted Doubly Linked List)

Which of the following describes the time complexity of the insert(k), delete(k), deleteEnd() and find(k) in a sorted doubly linked list?

find(k)	O(.)
insert(k)	O(.)
delete(k)	O(.)
deleteEnd()	O(.)

?

Use the best possible algorithm (e.g., insertion at the beginning vs. end) to find the time complexity in terms of O.

List (Sorted Doubly Linked List)

Which of the following describes the time complexity of the insert(k), delete(k), deleteEnd() and find(k) in a sorted doubly linked list?

find(k)	O(N)
insert(k)	O(N)
delete(k)	O(N)
deleteEnd()	O(1)

Some operations are still linear since they warrant a list traversal!

Doubly Linked List – Some Observations

- What is the time complexity (Big O) for

– inserting an element to the end?	O(1)
– inserting an element at the second-last position?	O(1)
– deleting an element form the end?	O(1)
– deleting an element from the second-last position?	O(1)
– Searching an element?	O(N)
– Inserting/deleting an element from kth location?	O(N)

Worth noting that the last is O(N) since it involves searching first!