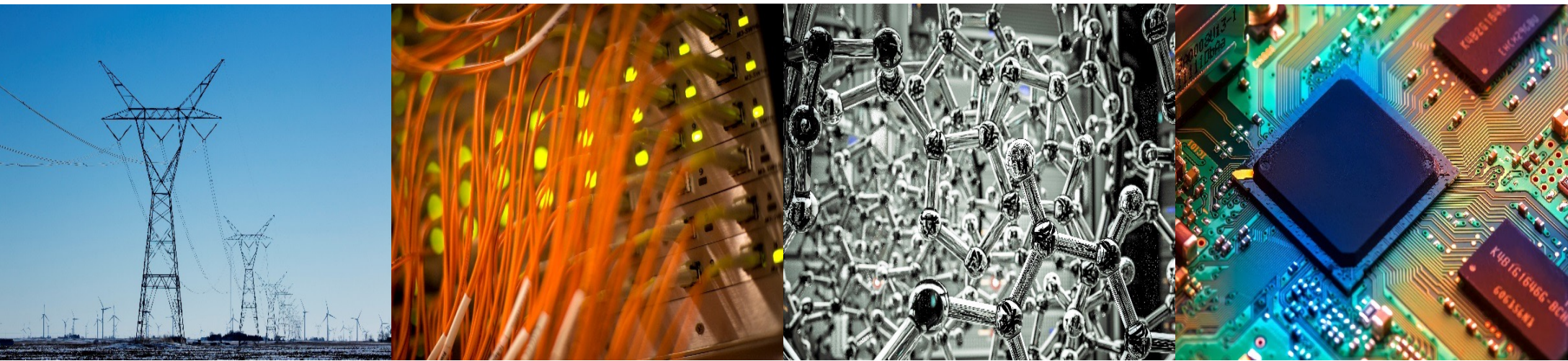


ECE 220 Computer Systems & Programming

Lecture 17 – Linked Lists

March 26, 2026



- **MT2 is next Thursday, conflict request deadline is this Sunday**

I ILLINOIS

Electrical & Computer Engineering

GRAINGER COLLEGE OF ENGINEERING

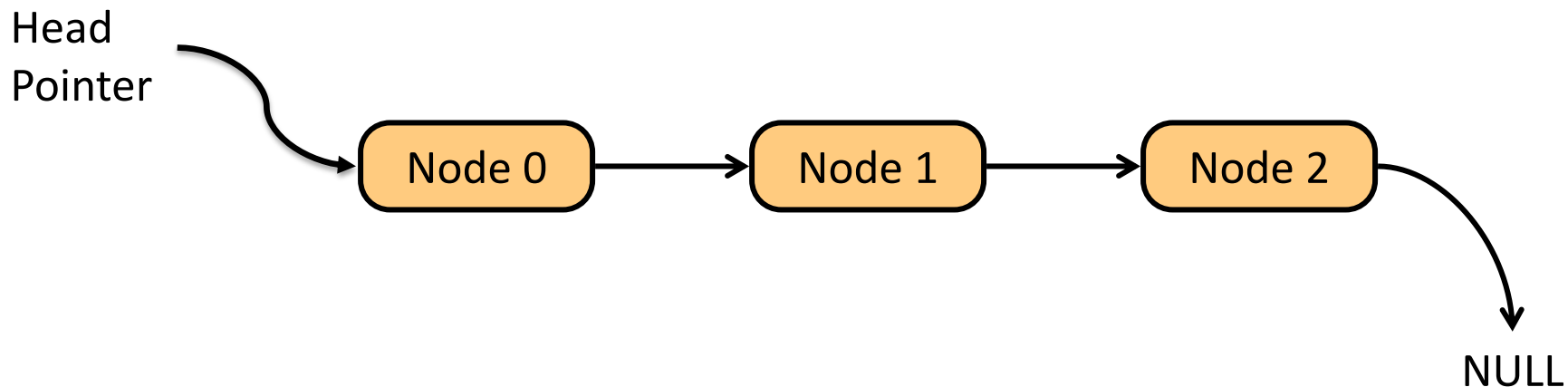
Lecture 16 Recap

- “Static” vs. Dynamic Memory Allocation: mechanism, location, lifetime, size
- malloc, free, calloc, realloc
- Memory Leak vs. Segmentation Fault

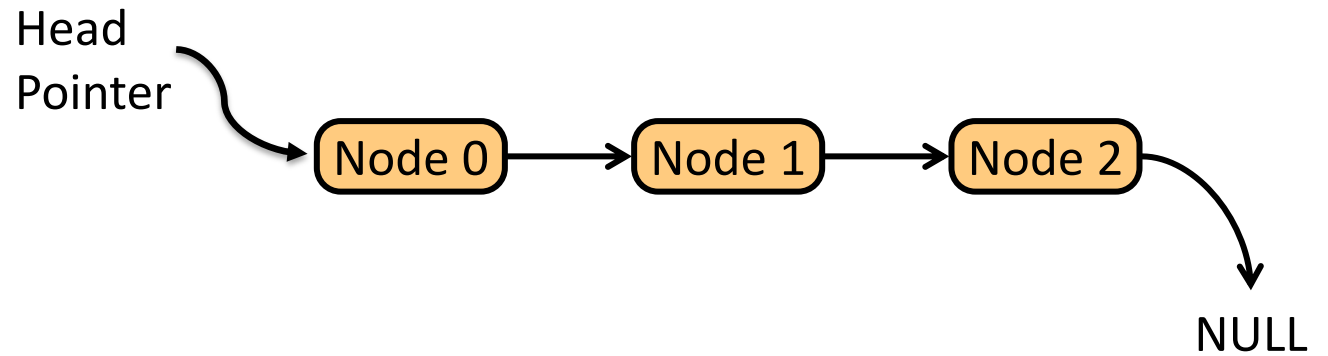
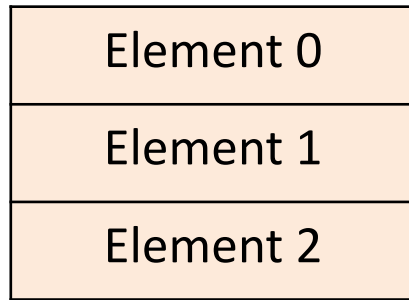
The Linked List Data Structure

A **linked list** is an ordered collection of **nodes**, each of which contains some **data**, connected using **pointers**.

- Each node points to the next node in the list.
- The first node in the list is called the _____
- The last node in the list is called the _____



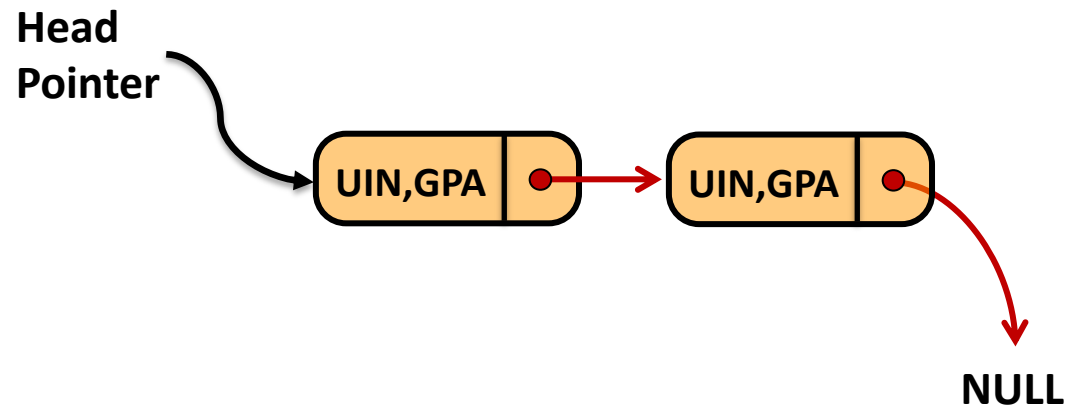
Array vs. Linked List



	Array	Linked List
Memory Allocation		
Memory Structure		
Memory Overhead		
Order of Access		
Insertion/Deletion		

Example: A List of Student Records

```
typedef struct studentStruct Node;  
struct studentStruct {  
    int UIN;  
    float GPA;  
    Node *next;  
};
```



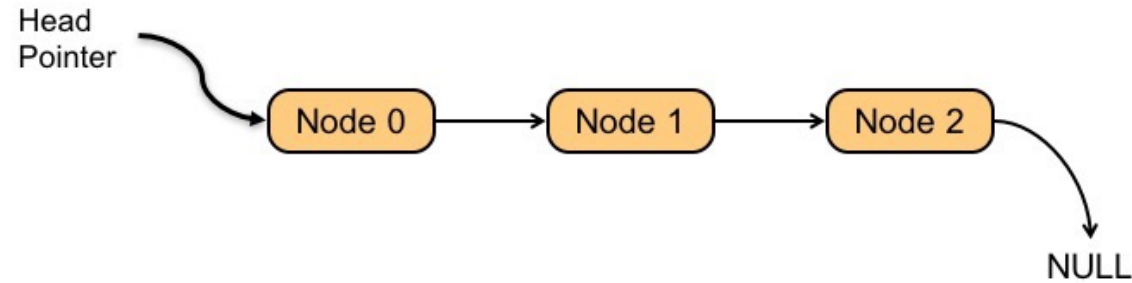
We have a list of 200 student records (nodes) **sorted by UIN**

1. Traverse the list to find a student record by UIN
2. Add a new student record to the sorted list at the correct location
3. Delete a student record from the list

Traverse a sorted list to find a student record by UIN

```
/* If matching UIN is found, print "record found" and return a pointer  
to this node, otherwise print "record not found" and return NULL */
```

```
Node *find_node(Node *head, int S_UIN){
```



```
/* Algorithm */
```

```
/* base case 1:
```

```
list is empty/reach the tail OR
```

```
current node's UIN is past the range (record not found) */
```

```
/* base case 2:
```

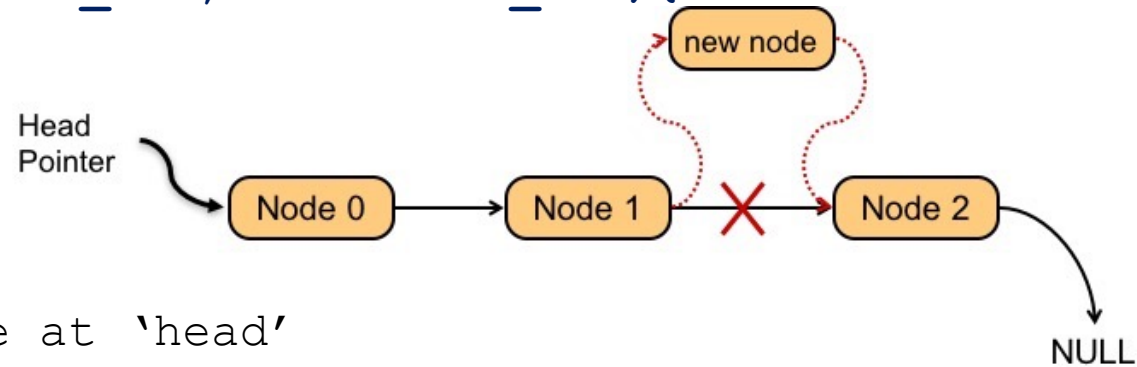
```
current node's UIN matches S_UIN */
```

```
/* recursive case: traverse the remaining list */
```

```
}
```

Add a new student record to a sorted list

```
/* add a new node to a sorted list at the correct location */  
void add_node(Node **list, int new_UIN, float new_GPA){
```



```
/* Algorithm
```

```
base case 1: insert new node at 'head'
```

```
base case 2: UIN already exists
```

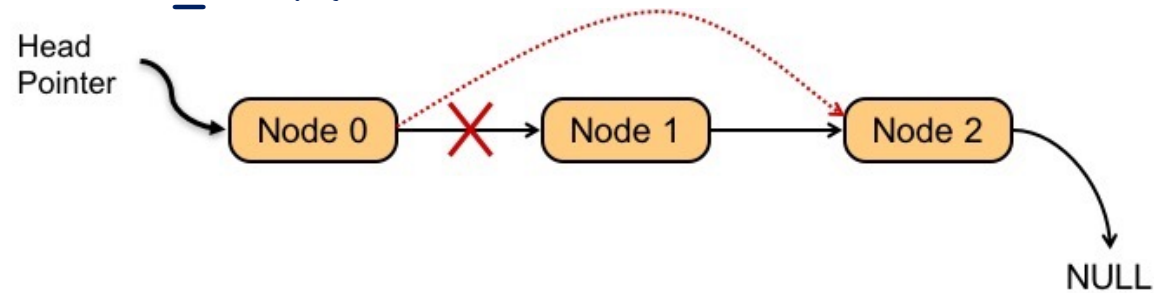
```
recursive case: traverse the remaining list */
```

```
}
```

7

Delete an existing student record from a sorted list

```
/* remove a node from a sorted list */  
void remove_node(Node **list, int old_UIN) {
```



```
/* Algorithm
```

```
base case 1:
```

```
empty list OR node with matching UIN not in list (record not found)
```

```
base case 2:
```

```
found node with matching UIN, redirect pointers and remove node
```

```
recursive case: traverse the remaining list */
```

```
}
```