Linked Lists

Inserting elements

using structs and pointers

Prior knowledge/Previous Lectures

- C Basics: Includes, prints, compiling and executing
- Standard data types, variable declaration and initialisation
- Loops & Conditions: if, for, and while
- Pointers and memory allocation
- Custom data types: struct
- Data structures: array

Learning Outcome

After this lecture, you should know:

- How to create list elements
- How to concatenate elements into a list
- How to insert elements at the end/the beginning
- How to insert elements in between other elements

What are Lists?

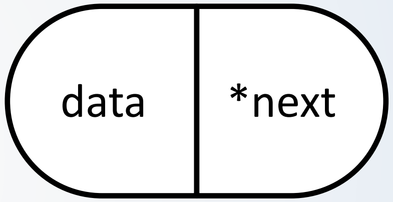
- *Recap:*
  - All variables are allocated blocks of memory
  - Arrays
    - One large block of memory
    - Not dynamic!
      - Breaks for more than 100 elements
      - Waste of memory for less than 100 elements

```c
int my_array[100];
int *my_array = malloc(100*sizeof(int));
```

- Lists are dynamic
  - Every node is allocated memory when necessary
    - Random memory assignment!
  - Every list element links to the next *(singly linked list)*

List elements

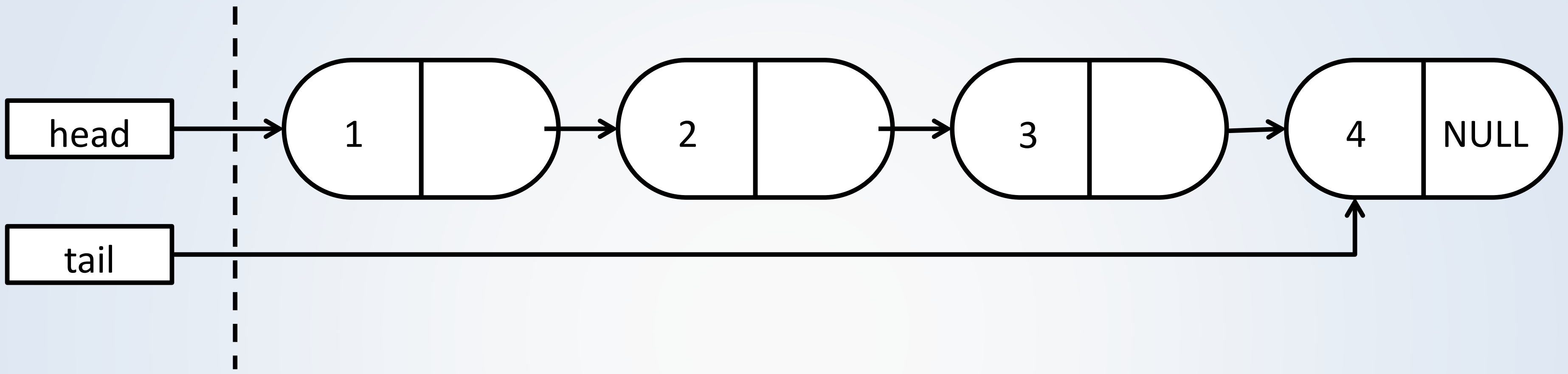
- Elements in a list are called *node*
  - Nodes contain the data to store and a pointer to the next node.
  - How can we create a node?
    - A: **Struct**
- Pointers to specific list elements
  - Head: start of the list
  - Tail: end of the list
- No other pointers to nodes!



```
struct node {
   int data;
   struct node *next;
};
```

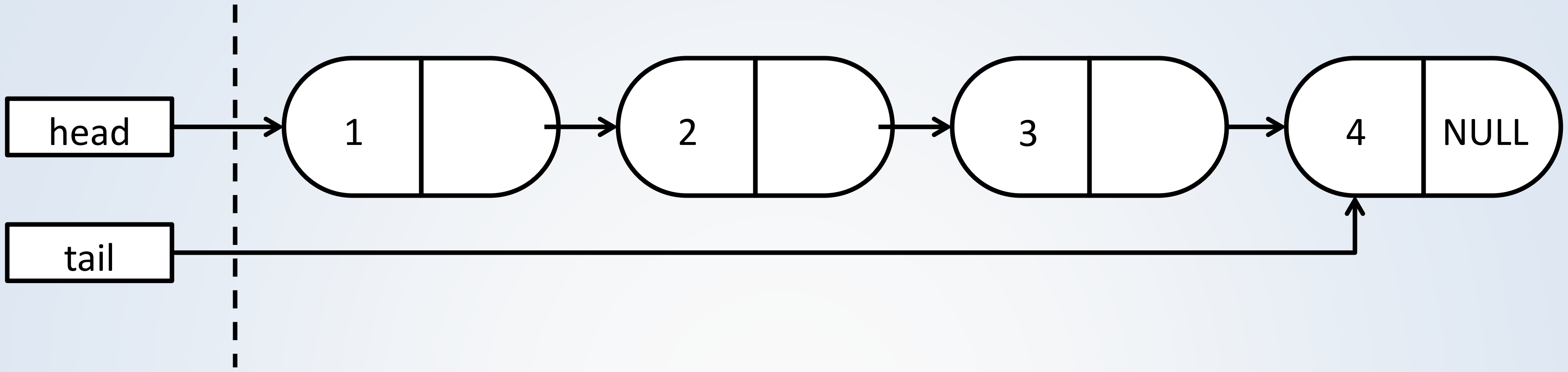
```
struct node *head, *tail;
```

Singly linked list



- Every element points to the next in the list
- *head* points to the start
- *tail* points to the end
  - Only for speed-up of insertion

Insertion rules

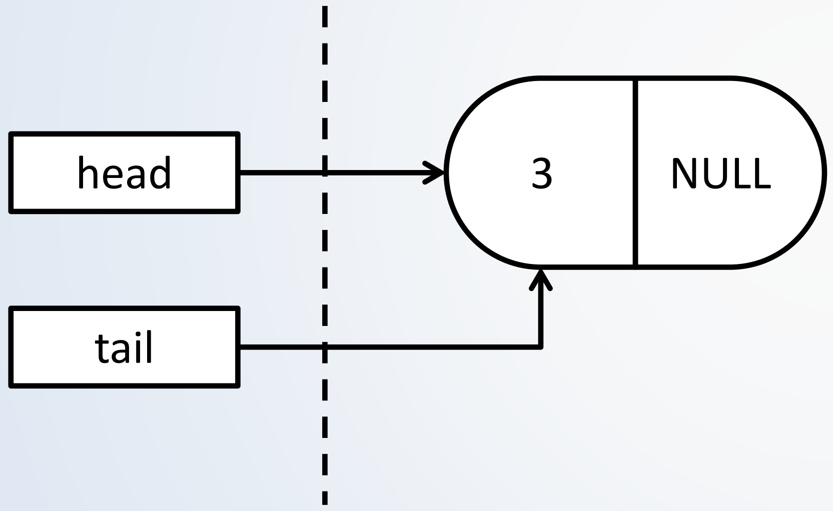


Group exercise!

1. Create new element
2. Change the *next* pointer in the **new element** to point to the element after the point of insertion
3. Change the *next* pointer of the **element before** the point of insertion to point to the new element
4. Update *head* and *tail* if needed

Creating a list

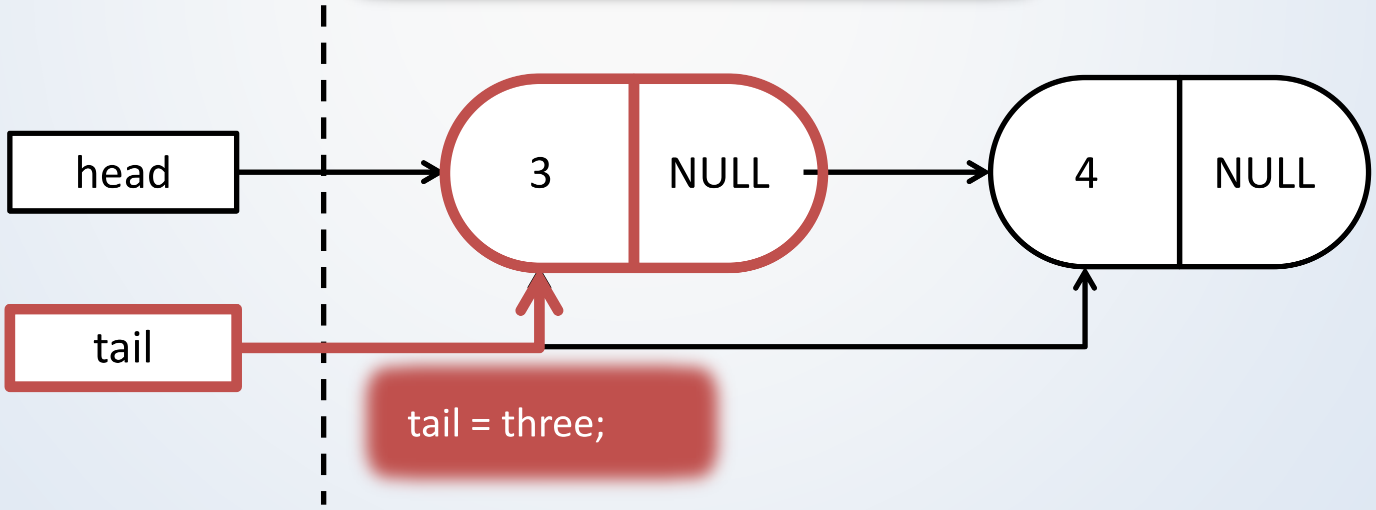
One element List



```
struct node *head, *tail;

struct node *three;
three = malloc(sizeof(struct node));
three->data = 3;
three->next = NULL;

head = three;

tail = three;
```

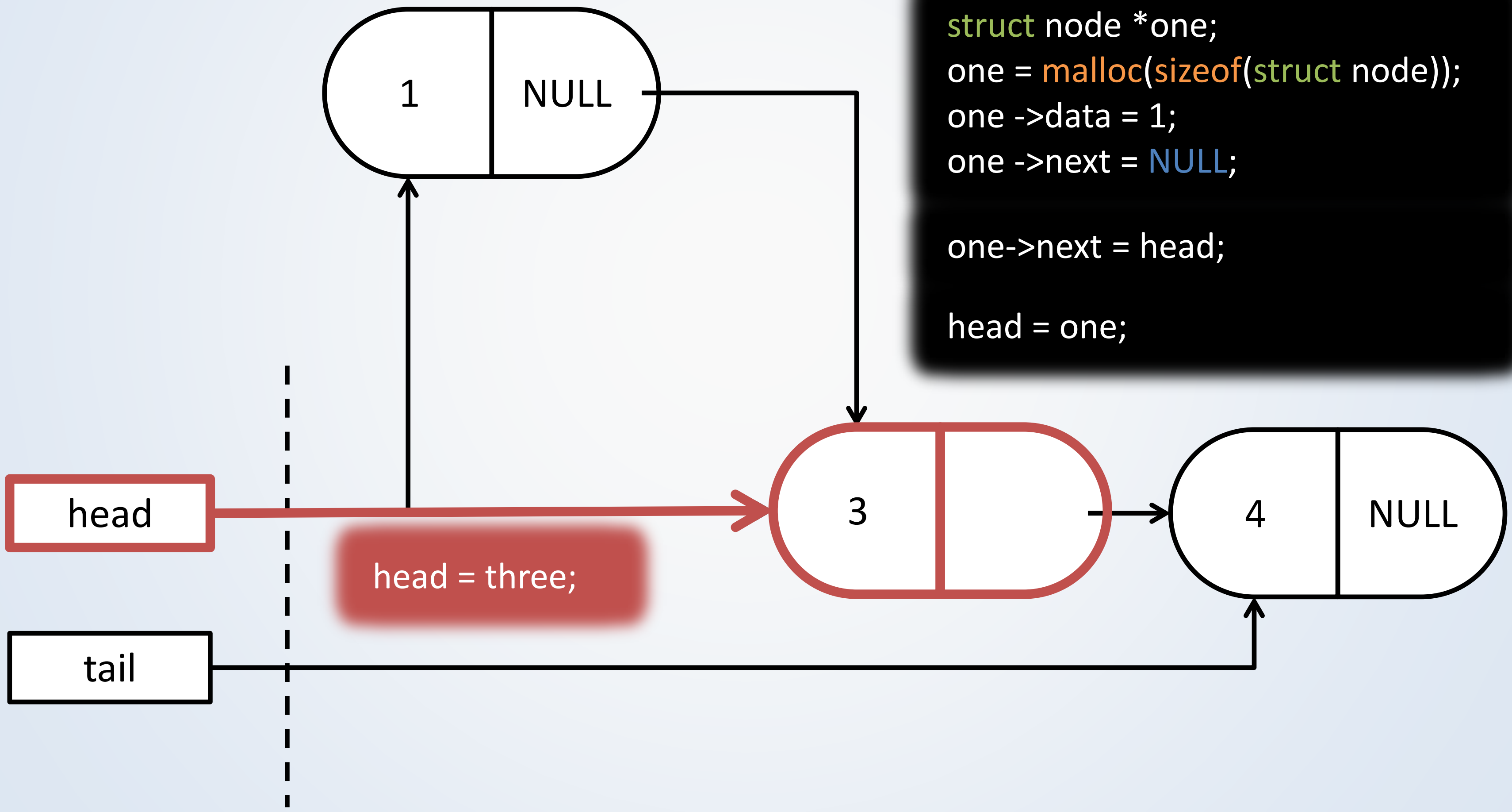
Inserting an element after tail

```
struct node *four;
four = malloc(sizeof(struct node));
four->data = 4;
four->next = NULL;

tail->next = four;

tail = four;
```



```
tail = three;
```

Inserting an element before head



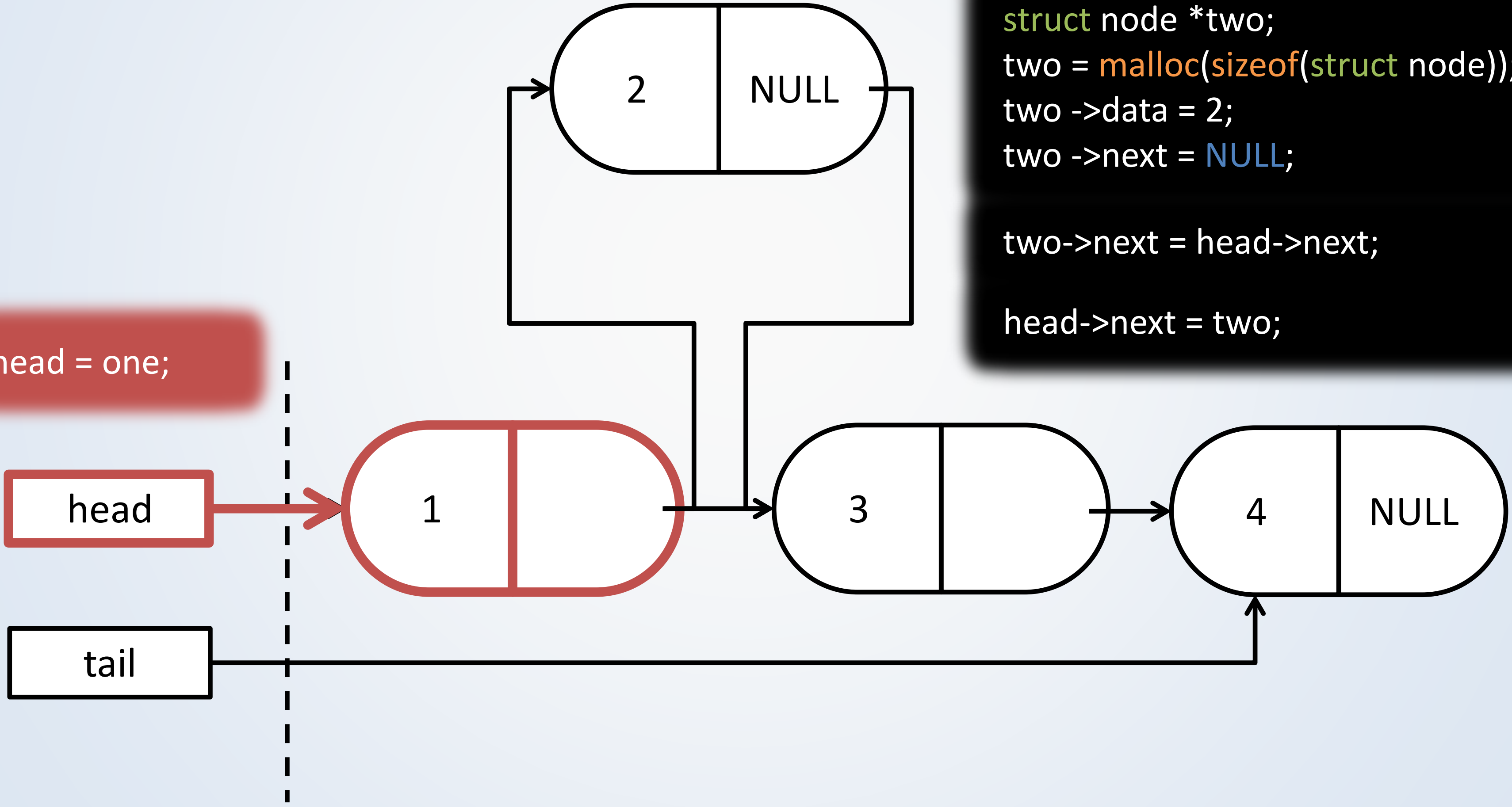
```
struct node *one;
one = malloc(sizeof(struct node));
one ->data = 1;
one ->next = NULL;

one->next = head;

head = one;
```

```
head = three;
```

Inserting an element after head

```
struct node *two;
two = malloc(sizeof(struct node));
two ->data = 2;
two ->next = NULL;

two->next = head->next;

head->next = two;
```

head = one;



Summary

- Nodes have to have a data field and a pointer to the next element
- *head* and *tail* are the only known pointers
- Insertion rules:
    1. Create new element
    2. Change the *next* pointer in the **new element** to point to the element after the point of insertion
    3. Change the *next* pointer of the **element before** the point of insertion to point to the new element
    4. Update *head* and *tail* if needed

Order matters!

To be continued

Next sessions:

- List Iterators and other list types
- Functions using pointers and return values
- Smart list insertion using iterators and functions
- Sorting lists and arrays

Thank you!

Recommended reading

- http://cslibrary.stanford.edu/103/LinkedListBasics.pdf

Alternative programming languages

- Java: https://www.cs.cmu.edu/~adamchik/15-121/lectures/Linked%20Lists/linked%20lists.html
- Python: http://www.openbookproject.net/thinkcs/python/english2e/ch18.html
- C#: http://www.functionx.com/csharp1/examples/linkedlist.htm