Advanced data structures

Variables

Lecturers: Mirko Jantschke, Pascal Scholz

4. Dezember 2018

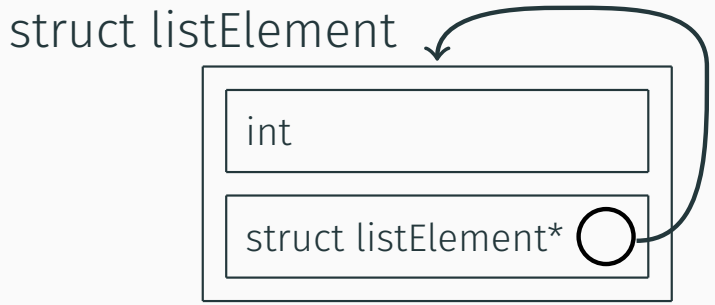
Contents

Lists and binary trees

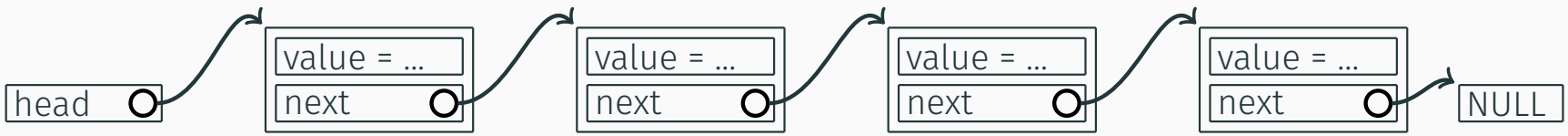
Yay, pointers

For better handling of datastructures, we define *structs* with additional pointers to themselves:

```
struct listElement {
    int value;
    struct listElement *next;
}
```

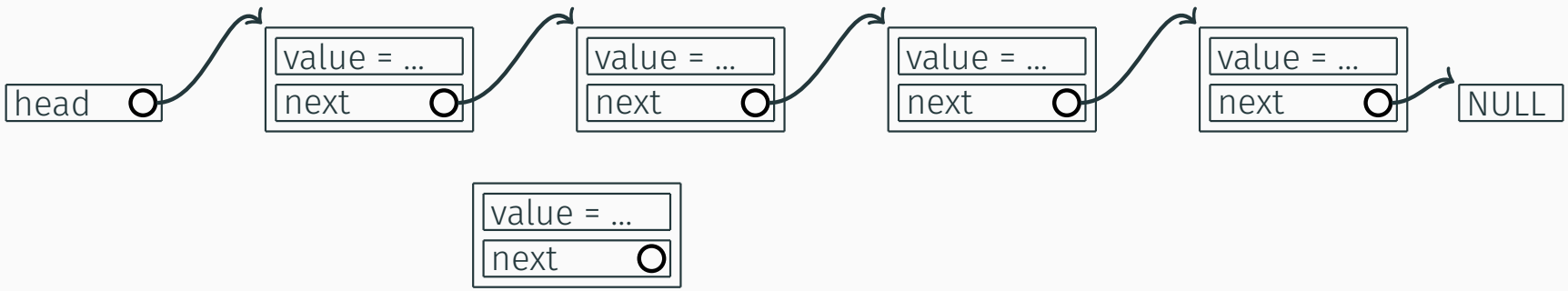


Single-linked lists



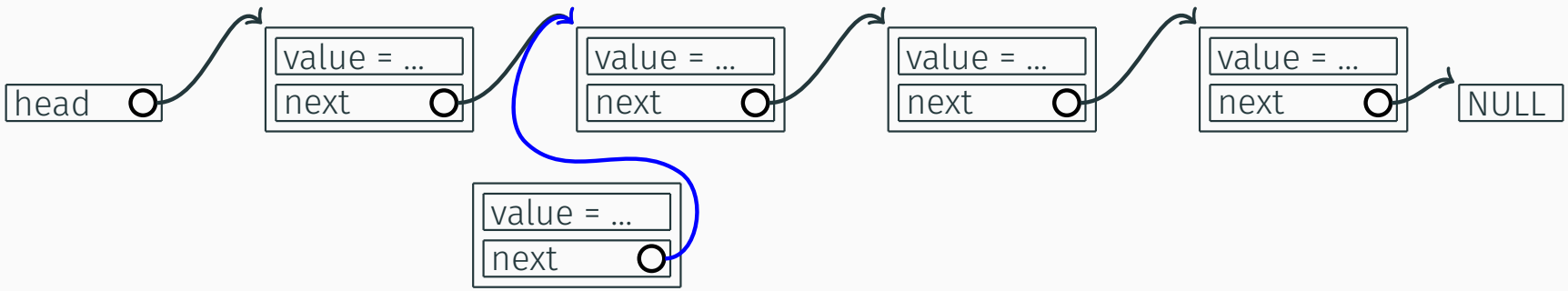
- Now we can build lists of this structures.

Single-linked lists



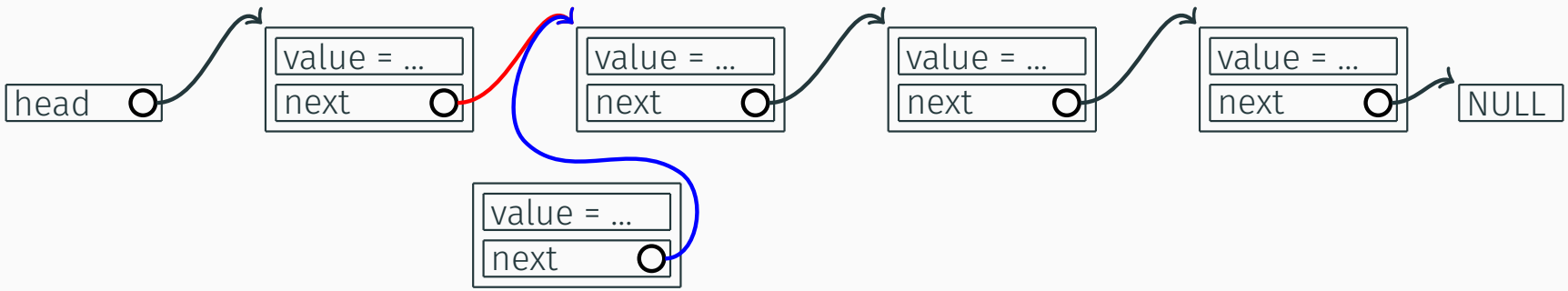
- Now we can build lists of this structures.
- If we want to add a value at a specific position...

Single-linked lists



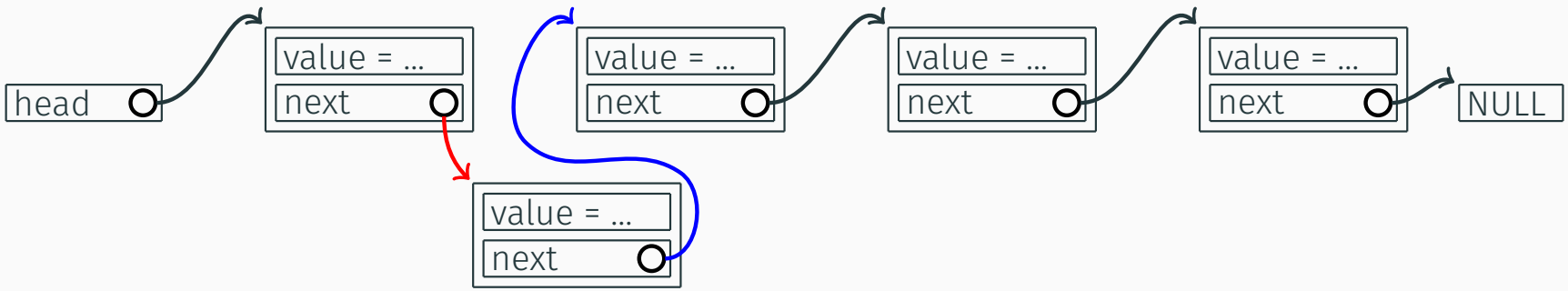
- Now we can build lists of this structures.
- If we want to add a value at a specific position...
- ... we just have to change the pointers.

Single-linked lists



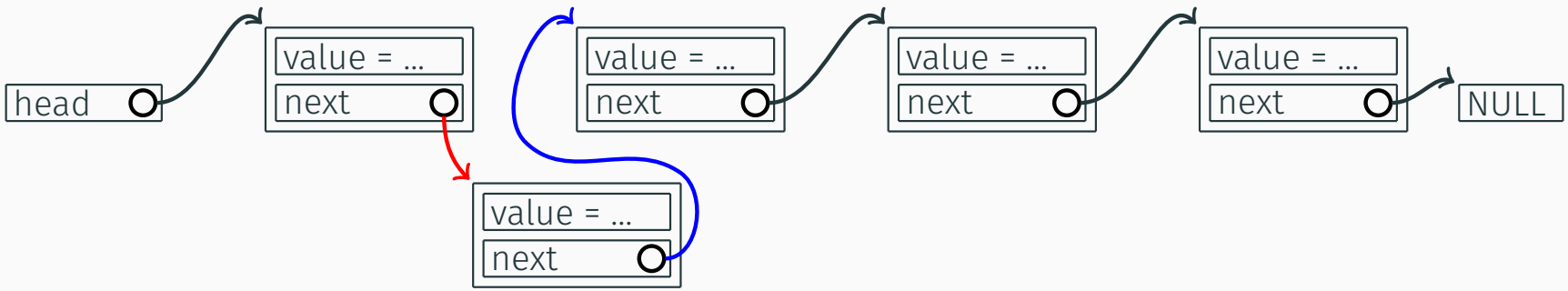
- Now we can build lists of this structures.
- If we want to add a value at a specific position...
- ... we just have to change the pointers.

Single-linked lists



- Now we can build lists of this structures.
- If we want to add a value at a specific position...
- ... we just have to change the pointers.

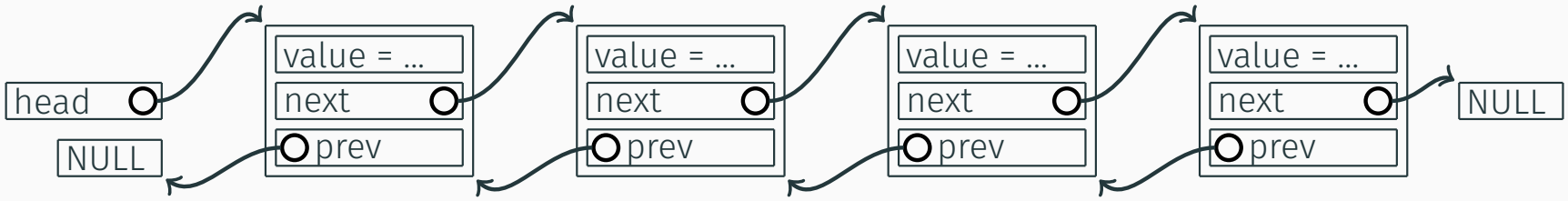
Single-linked lists



- Now we can build lists of this structures.
- If we want to add a value at a specific position...
- ... we just have to change the pointers.

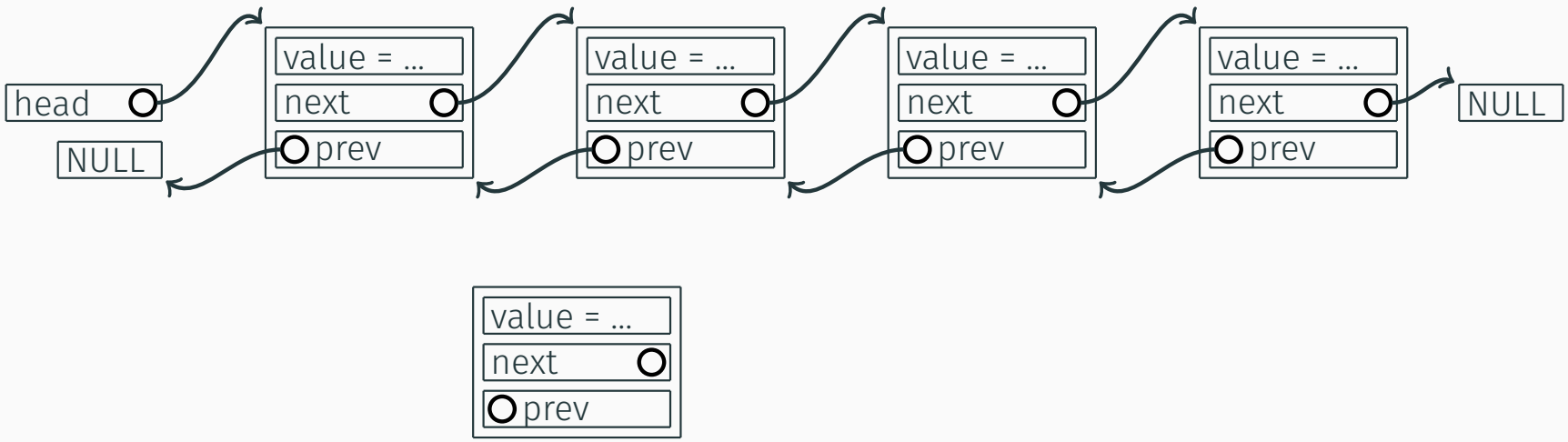
- Additionally we can easily iterate through this list.
- But only in one direction.

Suddenly: Double-linked lists



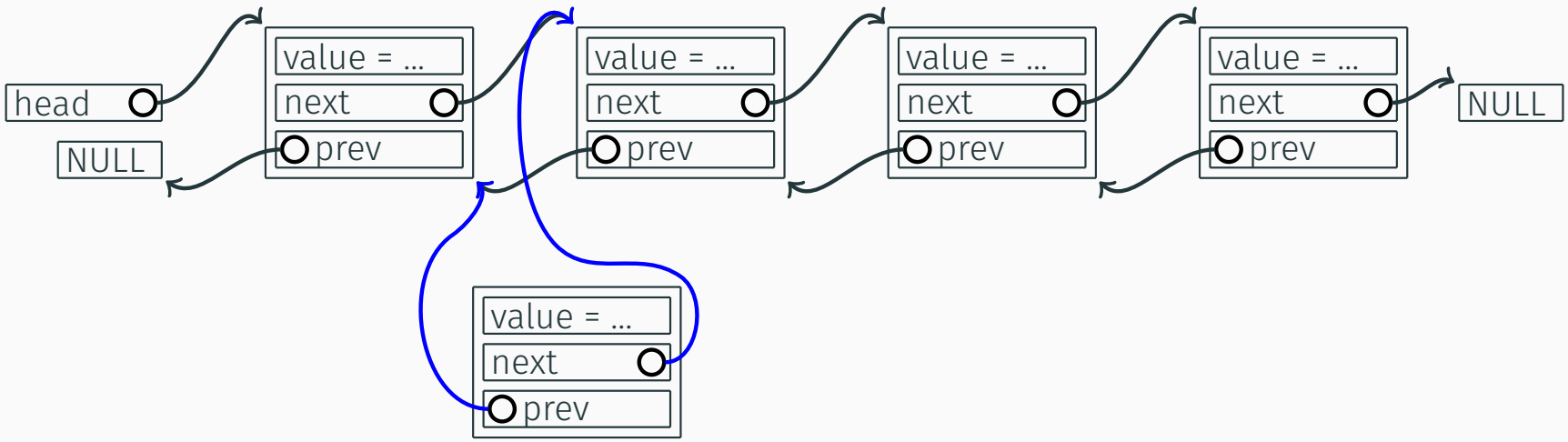
- With a pointer in each direction, we can iterate forwards and backwards.

Suddenly: Double-linked lists



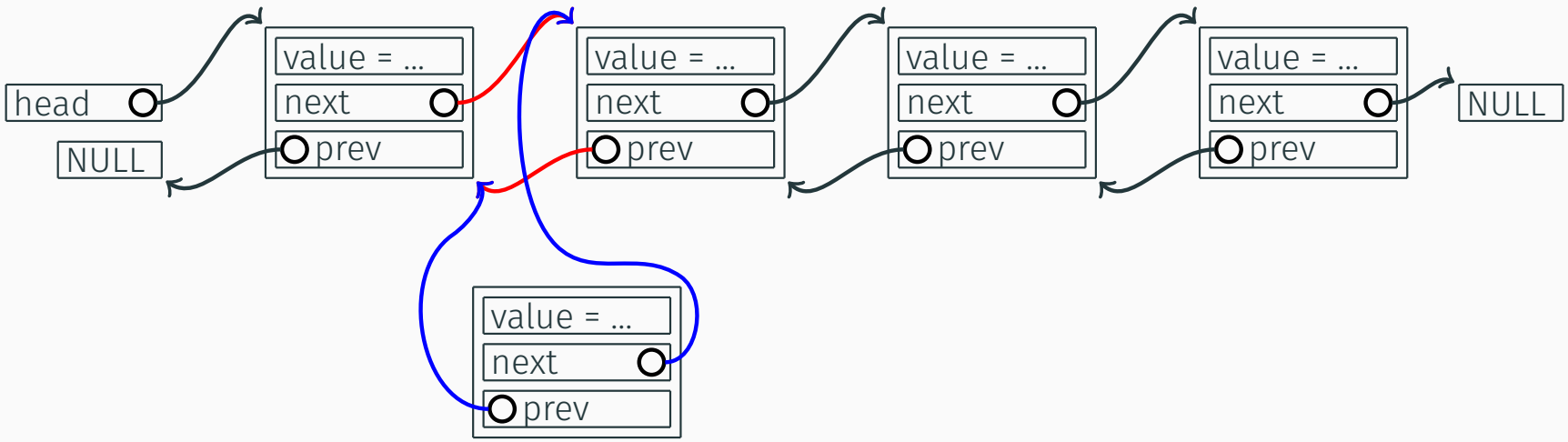
- With a pointer in each direction, we can iterate forwards and backwards.
- When inserting a value we now have to change a little more.

Suddenly: Double-linked lists



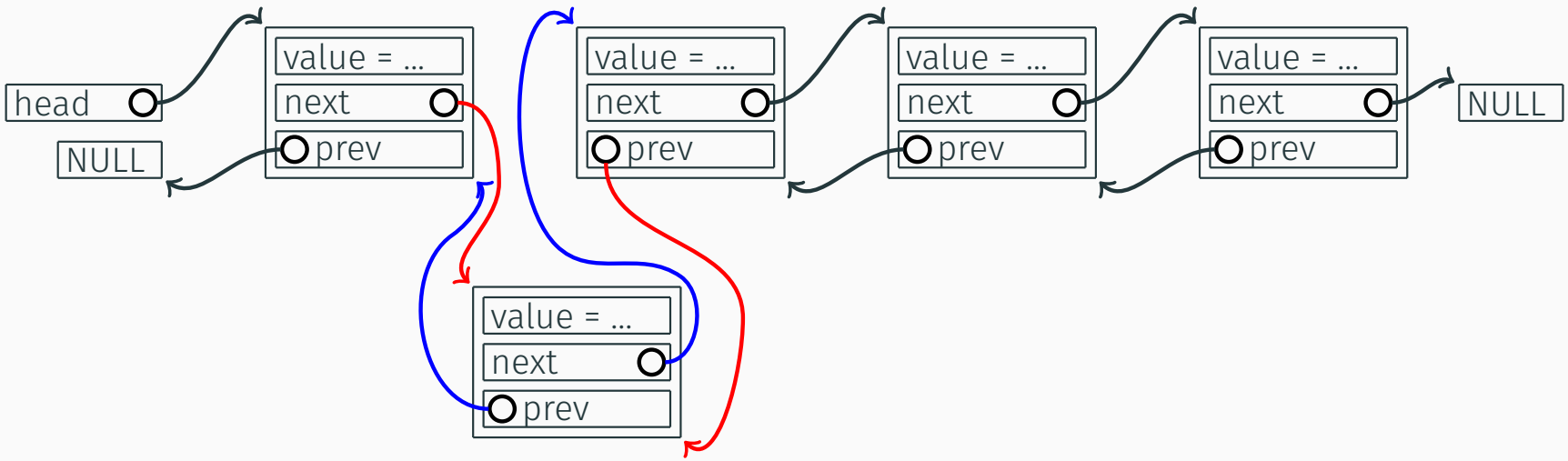
- With a pointer in each direction, we can iterate forwards and backwards.
- When inserting a value we now have to change a little more.

Suddenly: Double-linked lists



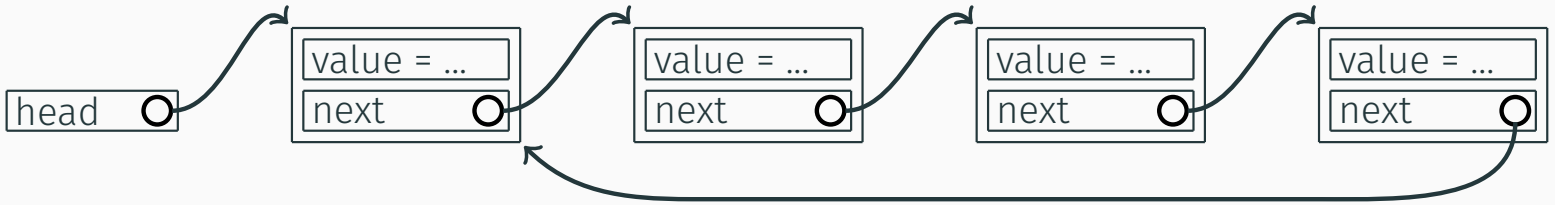
- With a pointer in each direction, we can iterate forwards and backwards.
- When inserting a value we now have to change a little more.

Suddenly: Double-linked lists

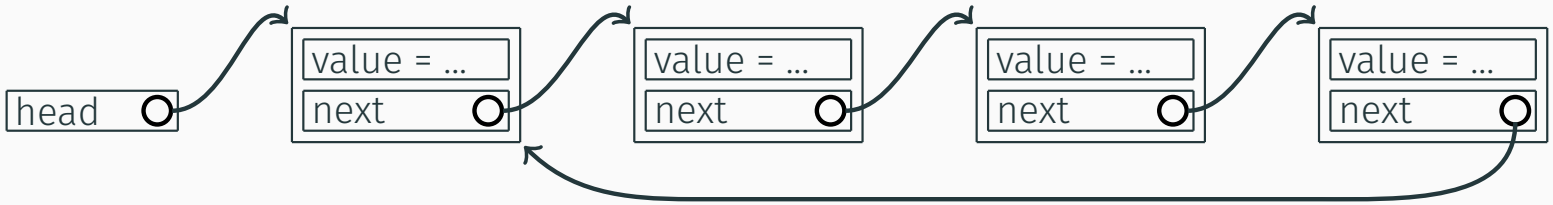


- With a pointer in each direction, we can iterate forwards and backwards.
- When inserting a value we now have to change a little more.

Cyclical lists

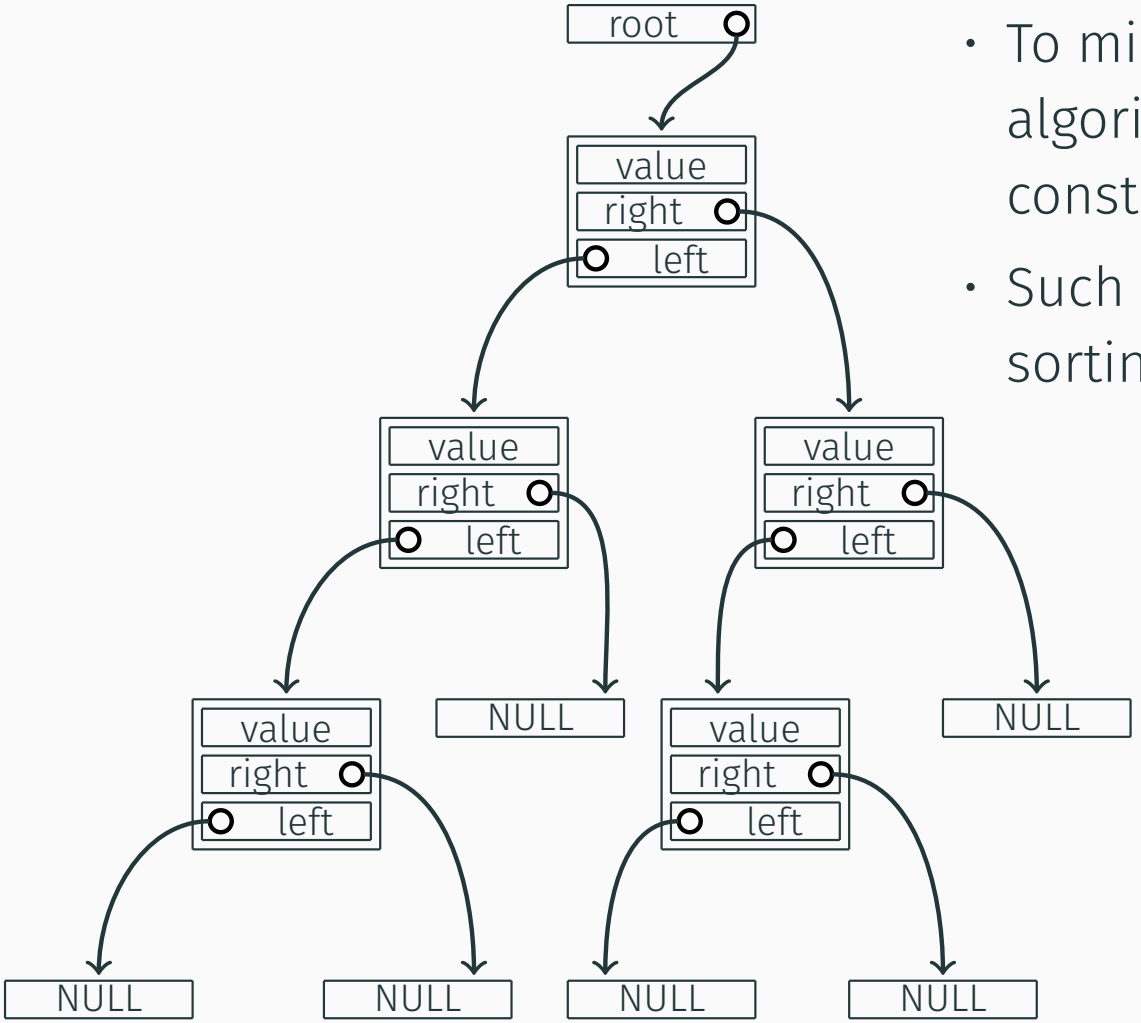


Cyclical lists



- You can also implement *double-linked cyclical lists*.

Binary trees



- To minimize the complexity of list algorithms, binary trees are constructed
- Such algorithms are e.g. searching, sorting, adding

The Dungeon

Back to the Dungeon

Again, we prepared a version of the ASCII-Dungeon for you in the repository (`https://github.com/fsr/c-slides`) in folder *materials/2_before/*.

- There are two monsters in the game, separated from each other.
- Edit the code to have a double chained list of monsters.
- You may want to add a few functions:
  - init_monster_list
  - free_monster_list
  - add_monster
  - print_monster_list

Pointers to functions

Higher order functions

It is possible to pass a function as an argument of another function. Doing so, you get a function that can call different other functions.

Since functions are placed in the text segment of you program, you simply pass the address of the function as a so called function pointer:

```
<return type> (*<function name>)(<parameter list>);
```

Example:

```
int (*op)(int a, int b) = add; /* Defined somewhere else */
int (*print)(const char *) = puts;  /* From stdlib */
```

Note that the & operator can be omitted.

Example

```c
#include <stdio.h>

int add(int a, int b) {
    return a + b;
}
int sub(int a, int b) {
    return a - b;
}
void printFunc(int (*f)(int, int), int a, int b) {
     printf("%d\n",f(a, b));
}
int main(void) {
    printFunc(add, 1, 2);
    printFunc(sub, 1, 2);
    return 0;
}
```

Mapping

Pointers to functions are often used for mapping. If you want to iterate through a list and call a function for every list item, you could do this with a new loop each time, but you also could write a function that takes a list and a function.

If only we had an example to try this out...
Oh wait... we do: The Dungeon.

- Write a mapping function that takes a list of monsters and a function.
- Edit the print_monster_list function to a print_entity function.
- Now you can print the monsters list by calling the mapping function passing the list and the print_entity function.