Лекция 5.
Связные списки

Даниил Михайлович Берлизов
Старший преподаватель Кафедры вычислительных систем СибГУТИ
E-mail: sillyhat34@gmail.com

Курс «Структуры и алгоритмы обработки данных»
Весенний семестр, 2021 г.

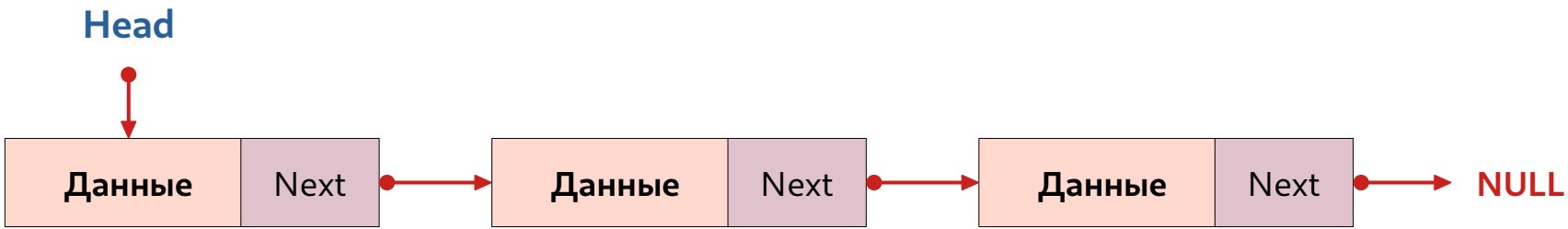
Связный список (linked list)

- **Связный список** (linked list) — это динамическая структура данных для хранения информации, в которой каждый элемент хранит указатели на один или несколько других элементов

Операция	Описание	Вычислительная сложность	Сложность по памяти
AddFront(*L*, *x*)	Добавляет элемент *x* в начало списка *L*	*O*(1)	*O*(1)
AddEnd(*L*, *x*)	Добавляет элемент *x* в конец списка *L*	*O*(*n*)	*O*(1)
Lookup(*L*, *x*)	Отыскивает элемент *x* в списке *L*	*O*(*n*)	*O*(1)
Size(*L*)	Возвращает количество элементов в списке *L*	*O*(1) или *O*(*n*)	*O*(1)

Односвязный список (singly linked list)

- Размер списка заранее неизвестен — элементы добавляются во время работы программы (динамически)
- Память под элементы выделяется динамически (функции: malloc, calloc, realloc, free)



Односвязный список (singly linked list)

```c
#include <stdio.h>
#include <stdlib.h>

struct listnode {
    char *key;              /* Ключ */
    int value;              /* Значение */
    struct listnode *next;  /* Указатель на следующий элемент */
};
```

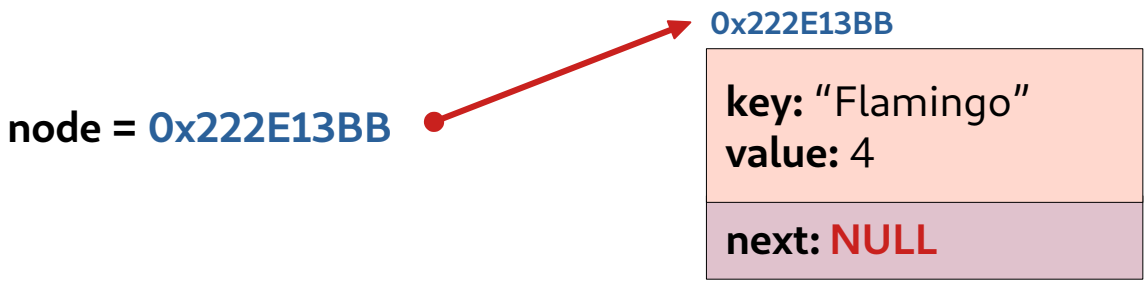
Создание элемента (выделение памяти)

```
struct listnode *list_createnode(char *key, int value)
{
    struct listnode *p;

    p = malloc(sizeof(*p));
    if (p != NULL) {
        p->key = key;
        p->value = value;
        p->next = NULL;
    }
    return p;
}
```

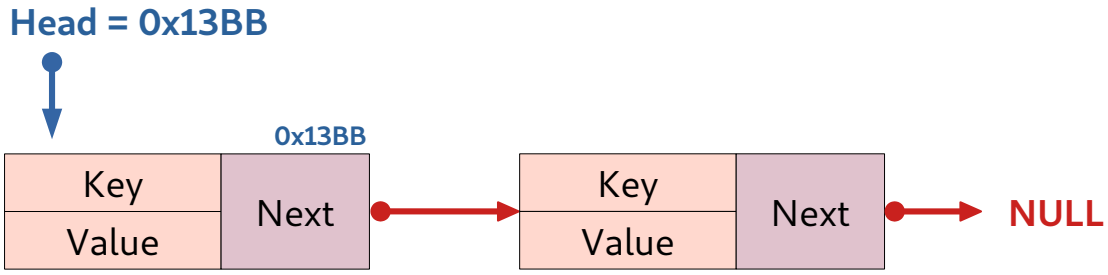
$T_{CreateNode} = O(1)$

Создание элемента (выделение памяти)

```
int main()
{
    struct listnode *node;

    node = list_createnode(“Flamingo”, 4); /* Список из одного элемента */
    return 0;
}
```

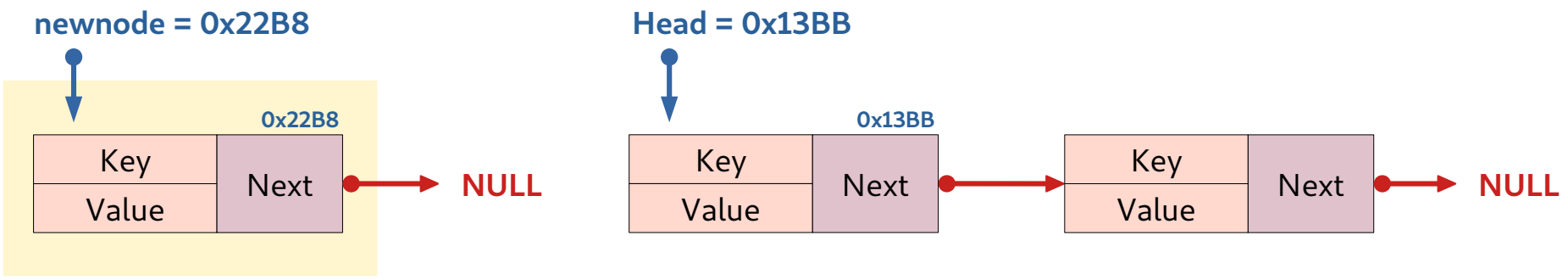


Добавление элемента в начало списка



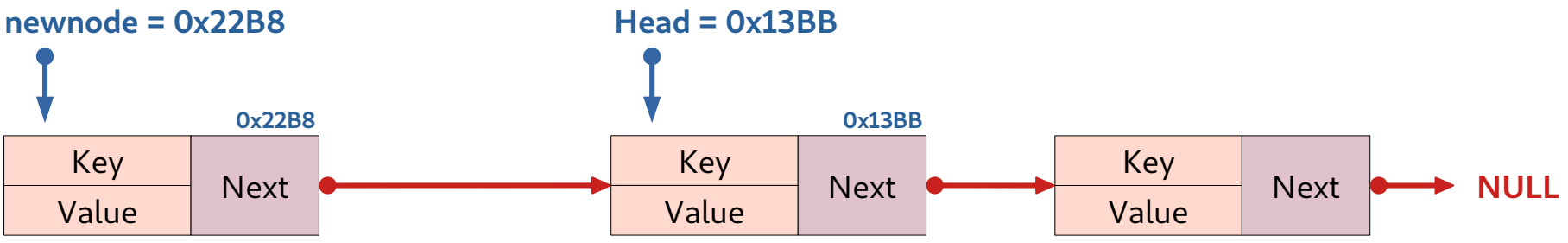
Добавление элемента в начало списка

1. Создаём новый узел **newnode** в памяти



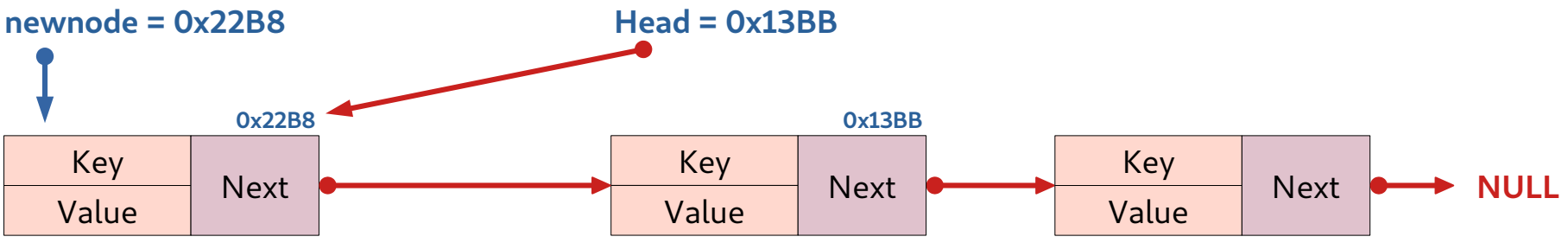
Добавление элемента в начало списка

2. Устанавливаем указатель **next** узла **newnode** на **head**



Добавление элемента в начало списка

3. Делаем головой списка узел **newnode**



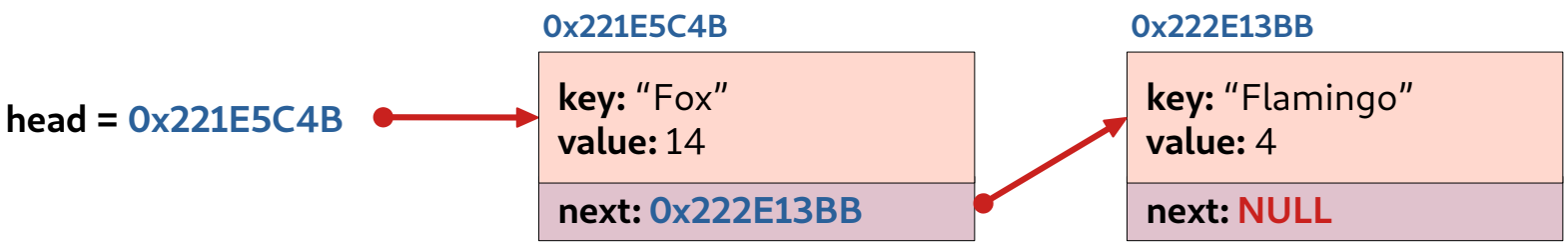
Добавление элемента в начало списка

```
struct listnode *list_addfront(struct listnode *list, char *key, int value)
{
    struct listnode *newnode;
    newnode = list_createnode(key, value);

    if (newnode != NULL) {
        newnode->next = list;
        return newnode;
    }
    return list;
}
```

$T_{AddFront} = O(1)$

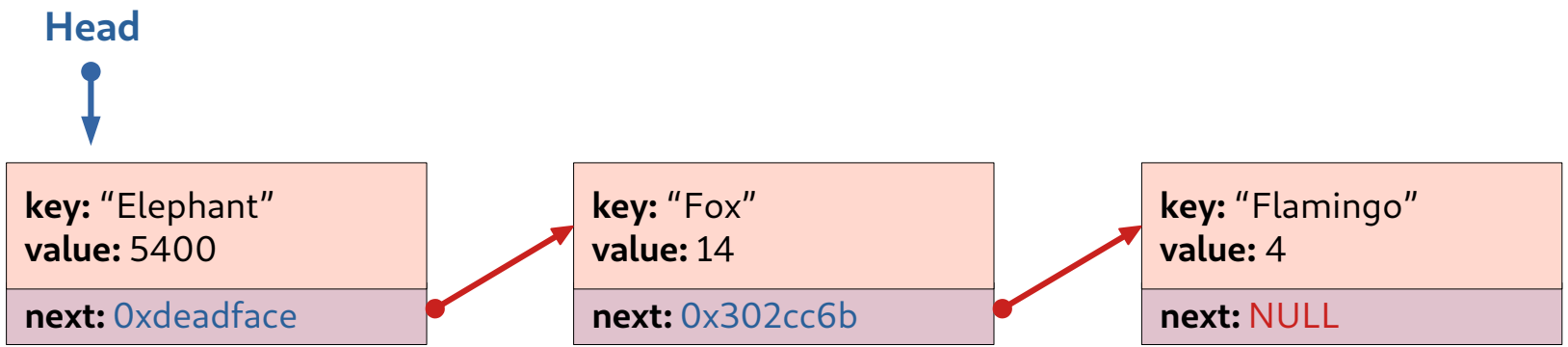
Добавление элемента в начало списка

```
int main()
{
    struct listnode *head;

    head = list_addfront(NULL, “Flamingo”, 4);
    head = list_addfront(head, “Fox”, 14);
    return 0;
}
```



Поиск элемента в списке (lookup)

- Начиная с головы списка, поочерёдно просматриваем узлы и сравниваем ключи
- В худшем случае требуется просмотреть все узлы, это требует ***O*(*n*)** операций



Поиск элемента в списке (lookup)

```
struct listnode *list_lookup(struct listnode *list, char *key)
{
    for ( ; list != NULL; list = list->next) {
        if (strcmp(list->key, key) == 0) {
            return list;
        }
    }
    return NULL;    /* Не нашли */
}
```

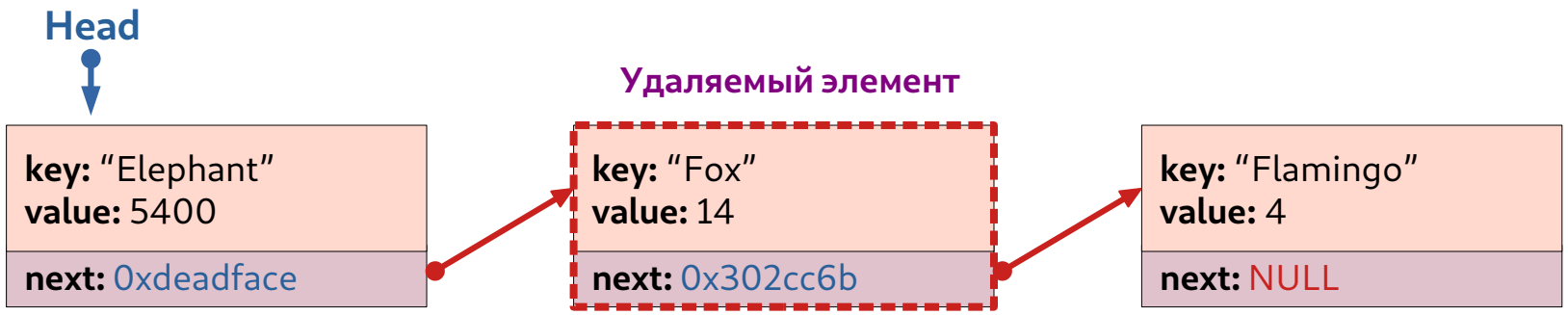
$T_{Lookup} = O(n)$

Поиск элемента в списке (lookup)

```
int main()
{
    struct listnode *head, *node;
    head = list_addfront(NULL, “Flamingo”, 4);
    head = list_addfront(head, “Fox”, 14);
    head = list_addfront(head, “Elephant”, 5400);

    node = list_lookup(head, “Fox”);
    if (node != NULL)
        printf(“Value: %d\n”, node->value);

    return 0;
}
```

Удаление элемента (delete)

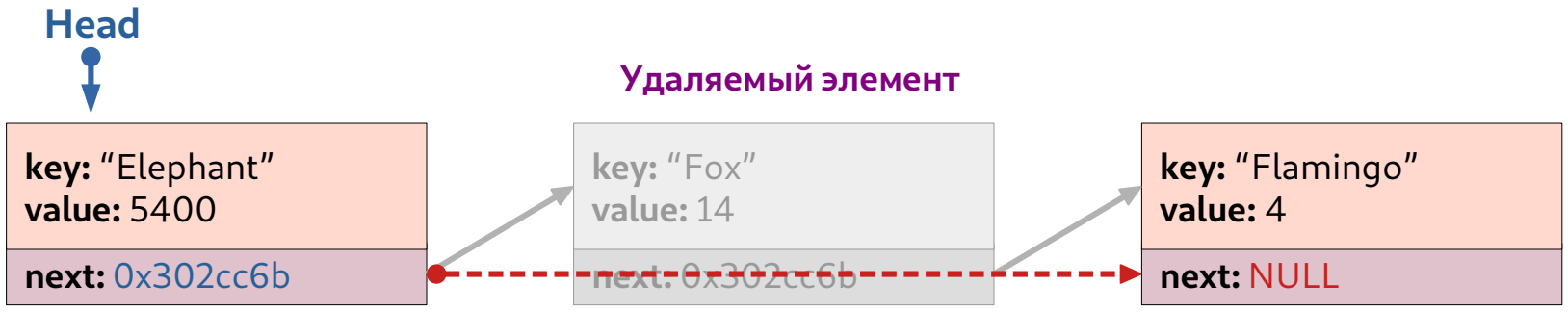
1. Находим элемент в списке — ***O*(*n*)**
2. Корректируем указатели — ***O*(1)**
3. Удаляем элемент из памяти — ***O*(1)**



- Удаляемый узел может быть в начале, середине (есть элементы справа и слева) или конце списка

Удаление элемента (delete)

1. Находим элемент в списке — ***O*(*n*)**
2. Корректируем указатели — ***O*(1)**
3. Удаляем элемент из памяти — ***O*(1)**



- Удаляемый узел может быть в начале, середине (есть элементы справа и слева) или конце списка

Удаление элемента (delete)

```
struct listnode *list_delete(struct listnode *list, char *key)
{
    struct listnode *p, *prev = NULL;
    for (p = list; p != NULL; p = p->next) {
        if (strcmp(p->key, key) == 0) {
            if (prev == NULL)
                list = p->next;          /* Удаляем голову */
            else
                prev->next = p->next;    /* Есть элемент слева */
            free(p);                     /* Освобождаем память */
            return list;                 /* Указатель на новую голову */
        }
        prev = p;                        /* Запоминаем предыдущий элемент */
    }
    return NULL;                         /* Не нашли */
}
```

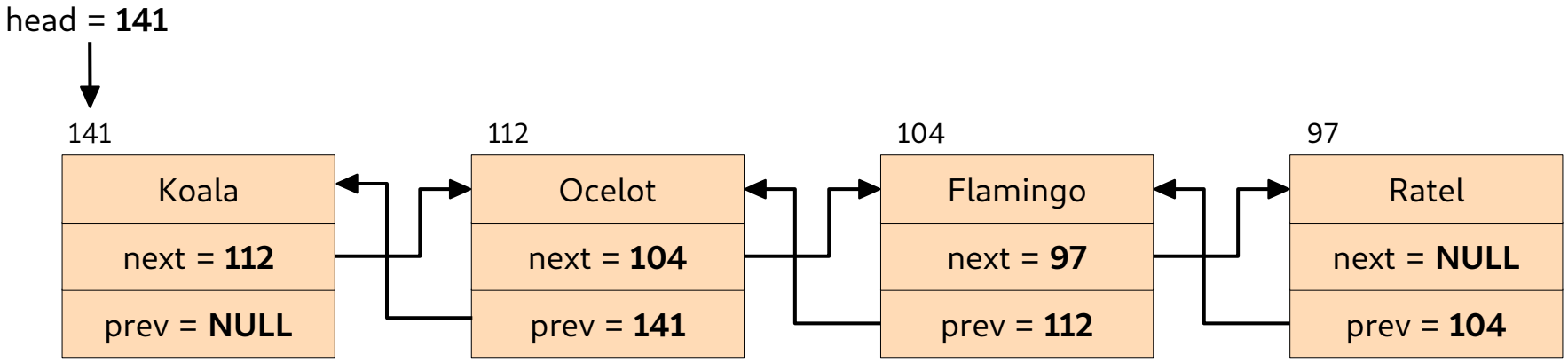
$T_{Delete} = O(n)$

Удаление элемента (delete)

```
int main()
{
    struct listnode *head, *node;
    head = list_addfront(NULL, “Flamingo”, 4);
    head = list_addfront(head, “Fox”, 14);
    head = list_addfront(head, “Elephant”, 5400);

    node = list_delete(head, “Fox”);
    if (node != NULL)
        head = node;    /* Указатель на новую голову */

    return 0;
}
```

Двусвязные списки

- Каждый узел двусвязного списка имеет три поля: *key*, *next* и *prev*
- Поле *key* — некоторый ключ, ассоциированный с узлом
- В поле *next* хранится адрес узла, следующего за текущим, в поле *prev* — предшествующего



Создание нового узла

```
function DoublyLinkedListCreateNode(key)
    node = AllocateMemory()    /* Выделяем память под узел */
    node.key = key
    node.next = NULL
    node.prev = NULL
    return node
end function
```

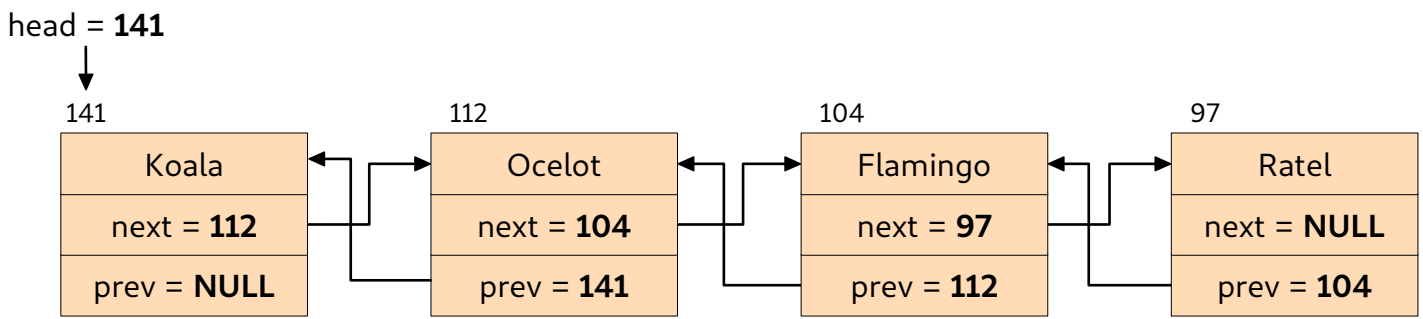
$T_{CreateNode} = O(1)$

Добавление узла в начало списка

```
function DoublyLinkedListAddFront(list, key)
    node = LinkedListCreateNode(key)
    node.next = list
    if list != NULL then
        list.prev = node
    end if
    return node
end function
```

$T_{AddFront} = O(1)$

- Функция создаёт в памяти новый узел с заданным значением поля *key*
- В поле *next* нового узла заносится адрес головы списка
- Если список не пуст, необходимо записать в указатель *prev* первого узла адрес нового элемента

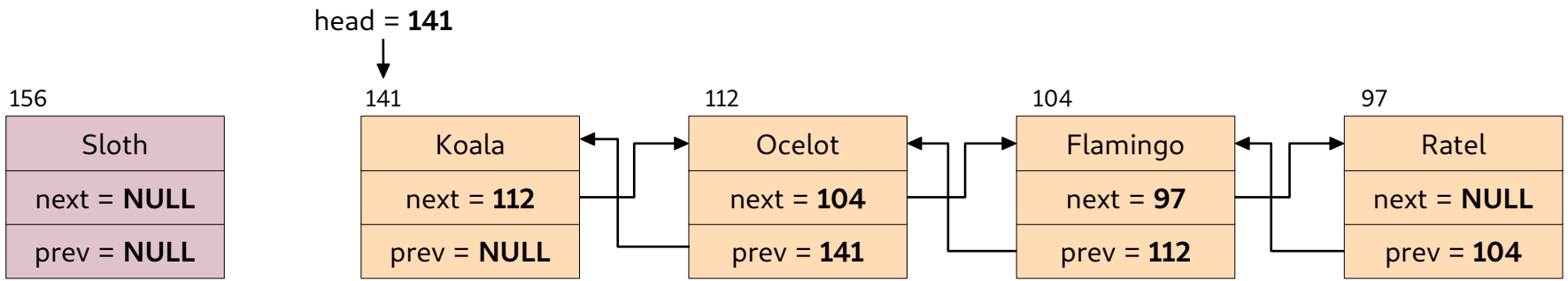


Добавление узла в начало списка

```
function DoublyLinkedListAddFront(list, key)
    node = LinkedListCreateNode(key)
    node.next = list
    if list != NULL then
        list.prev = node
    end if
    return node
end function
```

$T_{AddFront} = O(1)$

- Функция создаёт в памяти новый узел с заданным значением поля *key*
- В поле *next* нового узла заносится адрес головы списка
- Если список не пуст, необходимо записать в указатель *prev* первого узла адрес нового элемента

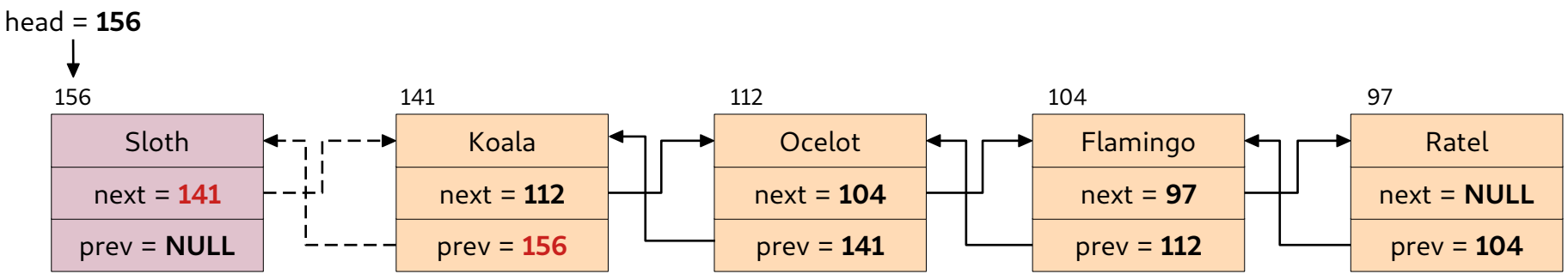


Добавление узла в начало списка

```
function DoublyLinkedListAddFront(list, key)
    node = LinkedListCreateNode(key)
    node.next = list
    if list != NULL then
        list.prev = node
    end if
    return node
end function
```

$T_{AddFront} = O(1)$

- Функция создаёт в памяти новый узел с заданным значением поля *key*
- В поле *next* нового узла заносится адрес головы списка
- Если список не пуст, необходимо записать в указатель *prev* первого узла адрес нового элемента

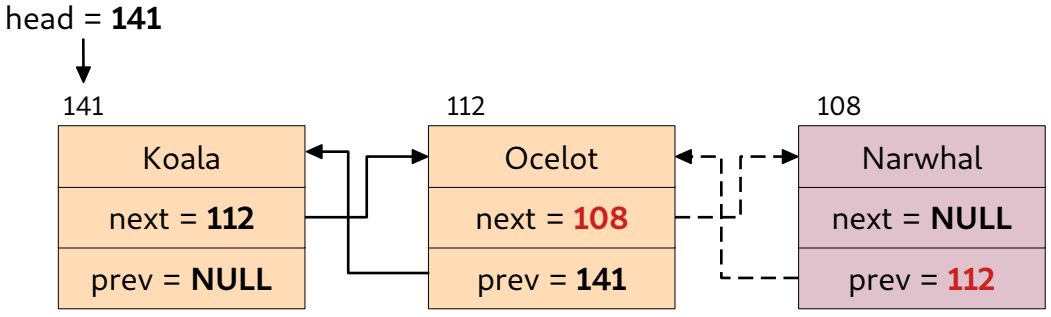


Добавление узла в конец списка

```
function DoublyLinkedListAddEnd(list, key)
    newnode = LinkedListCreateNode(key)
    if list = NULL then
        return newnode
    end if
    node = list
    while node.next != NULL do
        node = node.next
    end while
    node.next = newnode
    newnode.prev = node
    return list
end function
```

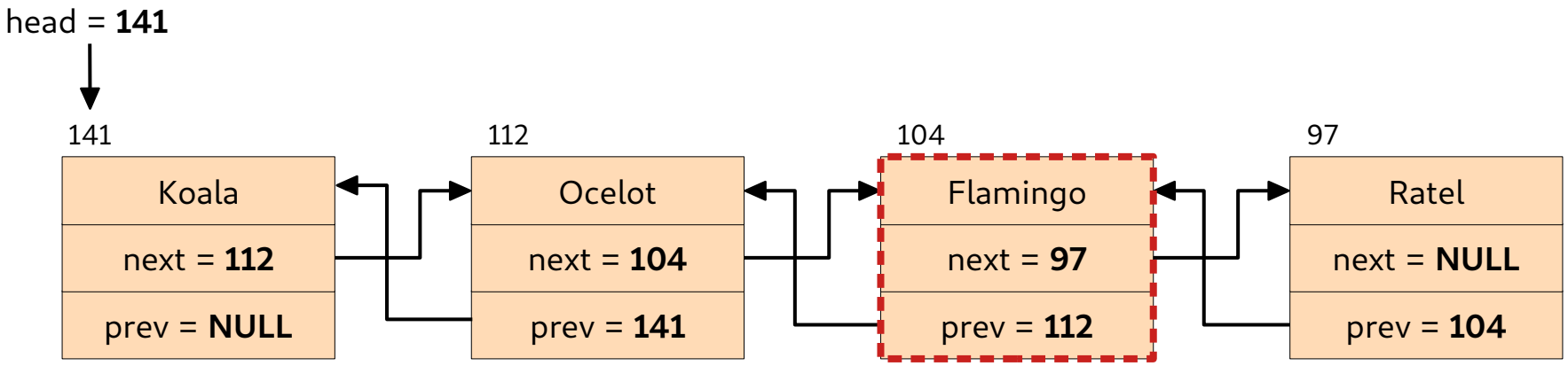
$T_{AddEnd} = O(n)$

- Выполняется проход по списку до последнего узла
- Указатель *next* последнего узла связывается с новым узлом



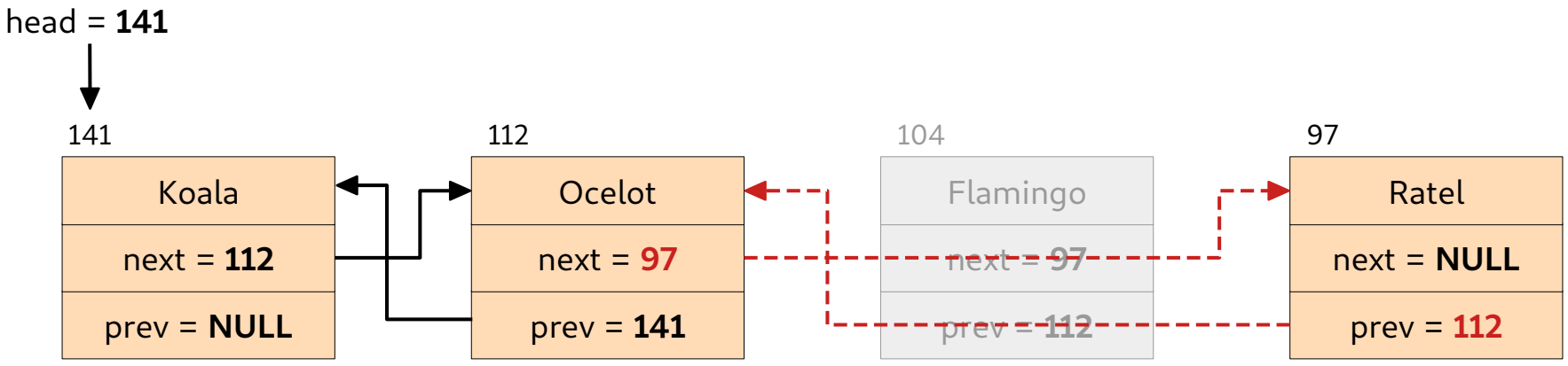
Удаление узла

- Удаляемый узел может находиться в начале, середине или конце двусвязного списка



Удаление узла

- Удаляемый узел может находиться в начале, середине или конце двусвязного списка



Удаление узла

```
function DoublyLinkedListDelete(list, key)
    node = list
    while node != NULL do
        if node.key = key then
            if node.prev = NULL then
                list = node.next
            else
                node.prev.next = node.next
            end if
            if node.next != NULL then
                node.next.prev = node.prev
            end if
            FreeMemory(node)
            return list
        end if
    end while
    return NULL
end function
```

$T_{Delete} = O(n)$

Домашнее чтение

- **[Kernighan]** Прочитать о реализации связных списков в «Практике программирования», С. 61—66
- **[DSABook]** **Глава 6. «Списки»**

ご清聴ありがとうございました!

Даниил Михайлович Берлизов
Старший преподаватель Кафедры вычислительных систем СибГУТИ
E-mail: sillyhat34@gmail.com

Курс «Структуры и алгоритмы обработки данных»
Весенний семестр, 2021 г.