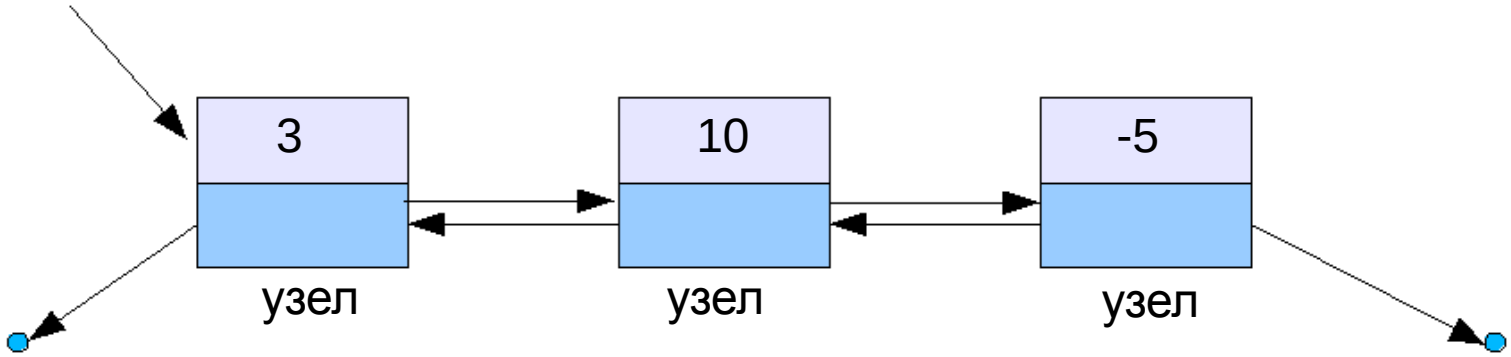
Двусвязный список

18 июля 2017 г.

Двусвязный список

- совокупность узлов, состоящих из двух частей: значения и информации о предыдущем и о следующем элементах списка
- только последовательный доступ к элементам
- можно передвигаться в обе стороны



Особенности

- элементы списка могут храниться не последовательно, а в разных участках памяти
- для связи текущего элемента со следующим хранится указатель на следующий элемент
- для связи текущего элемента с предыдущим хранится указатель на предыдущий элемент
- обход списка в любом направлении
- позволяет эффективно вставлять элементы в произвольную позицию списка и удалять элементы из произвольной позиции

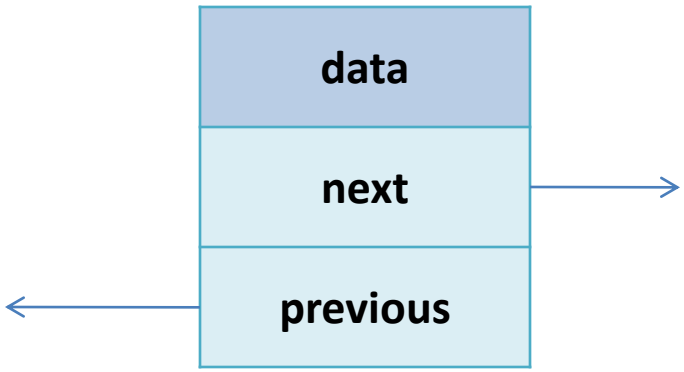
Основные операции

- вставить элемент в список (insert)
- удалить элемент из списка (erase)
- добавить элемент в конец списка (push_back)
- удалить элемент из конца списка (pop_back)
- добавить элемент в начало списка (push_front)
- удалить элемент из начала списка (pop_front)
- узнать количество элементов (size)
- проверить на пустоту (empty)

Реализация: узлы

- узел представляет собой структуру
- узел содержит поле с данными и указатели:
  - на следующий узел
  - на предыдущий узел

```
template <typename T>
struct node {
        T data;
        node* next;
        node* previous;
};
```



Объявление класса

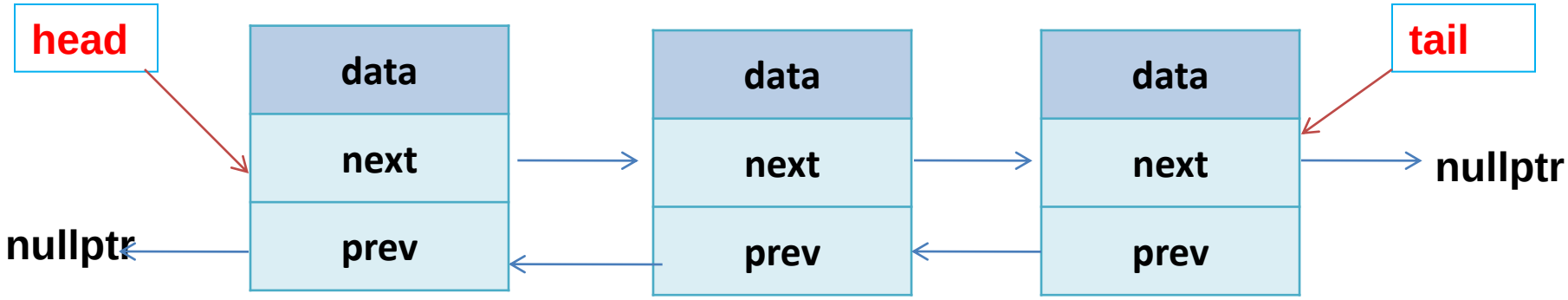
- создается класс для реализации списка
- в классе дополнительно создаются поля-указатели на голову и хвост списка (head и tail)

```
template <typename T>
struct node { … };

template <typename T>
class MyList {
        node* head;
        node* tail;
};
```

Голова и хвост списка

- голова всегда указывает на первый элемент в списке
- хвост всегда указывает на последний элемент в списке
- если в списке нет ни одного элемента, значения головы и хвоста устанавливаются в NULL (или nullptr)



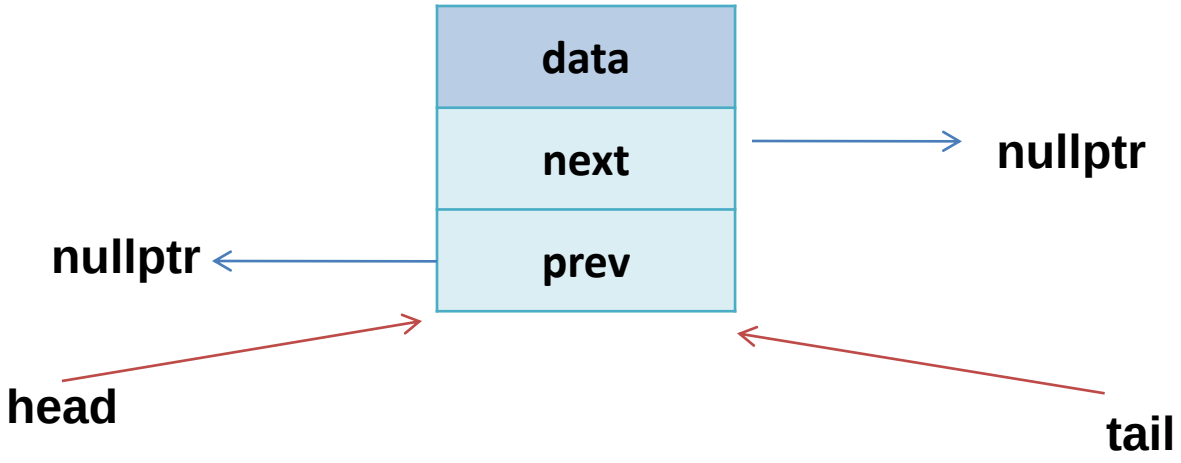
Инициализация

- в начале работы список пуст
- указатели на хвост и на голову ни на что не указывают

```
template <typename T>
MyList <T>::MyList() :
    head(nullptr),
    tail(nullptr)
{
}
```

Создание первого узла

- выделяется память под новый узел
- значение в новом узле устанавливается равным переданному значению
- оба указателя в узле устанавливаются в nullptr
- head и tail указывают на этот новый элемент

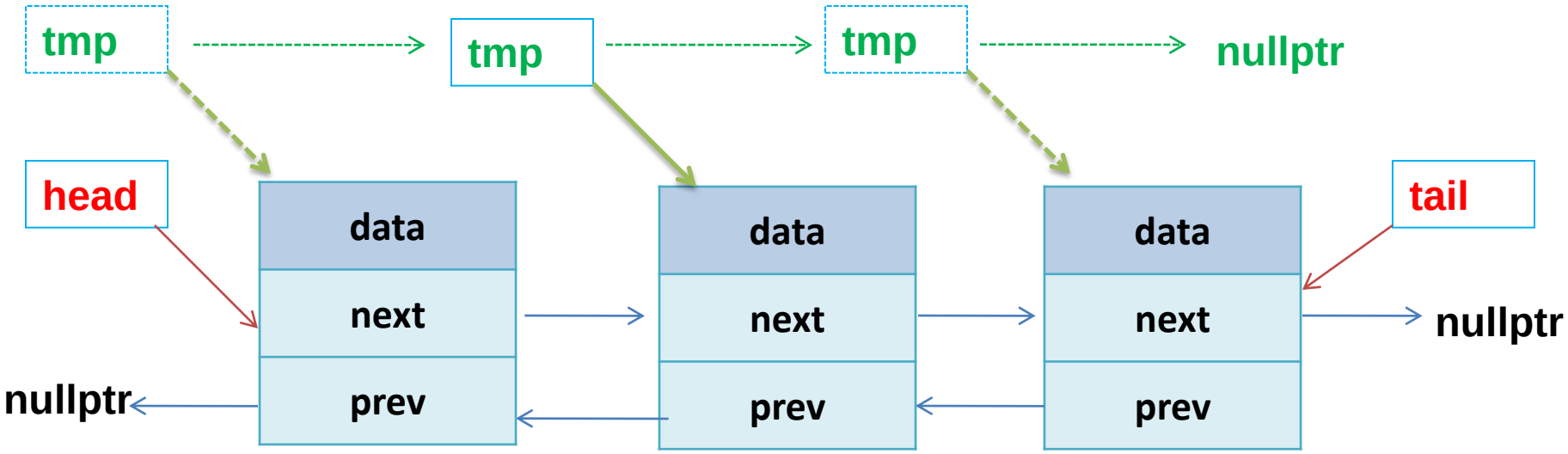


Обход списка

- обходить список можно с начала или с конца
- обход организуется в цикле или рекурсивно
- создается указатель node* tmp, значение которого устанавливается в head (указатель на начало списка)
- данные можно получить через tmp->data
- значение указателя tmp меняется на tmp->next (переход к следующему узлу)
- цикл работает до тех пор, пока указатель tmp не равен NULL (или nullptr)

Обход списка (пример)

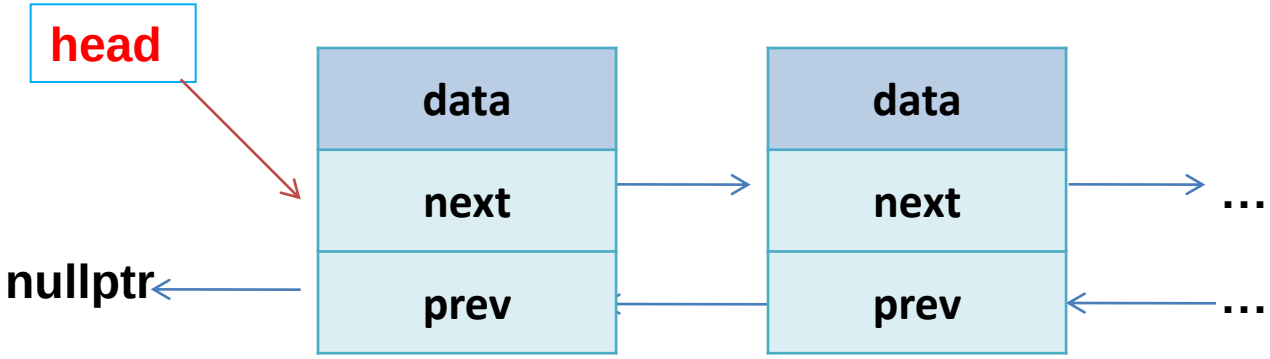
```
node<T>* tmp = head;
while (tmp != nullptr) {
    std::cout << tmp->data << std::endl;
    tmp = tmp->next;
}
```

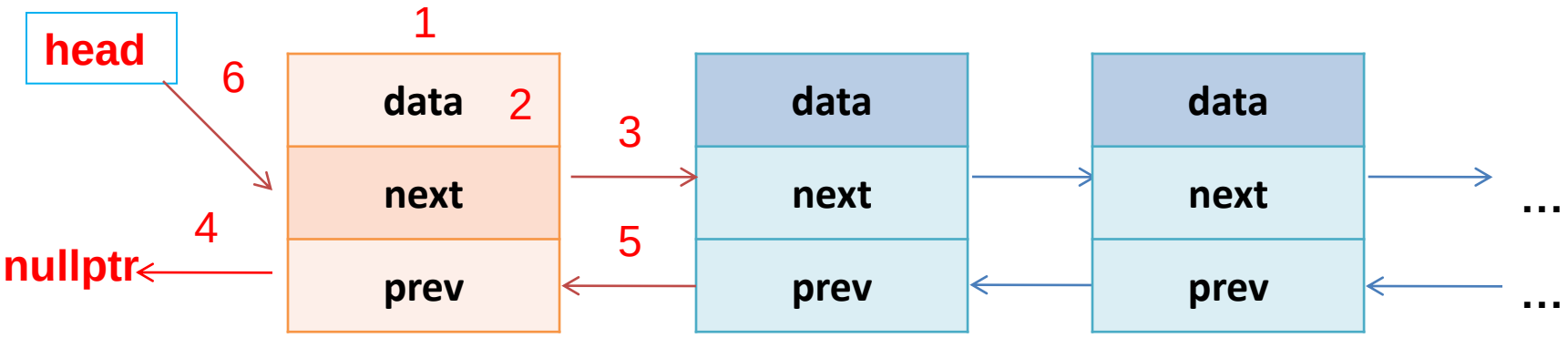


Вставка узла в начало (push_front)

1) выделяется память под новый узел
2) значение в новом узле устанавливается равным переданному значению
3) next в новом узле указывает на элемент, который прежде был первым
4) previous в новом узле устанавливается в nullptr
5) previous в узле, который был первым, указывает на новый узел
6) head указывает на новый элемент

Вставка узла в начало (схема)

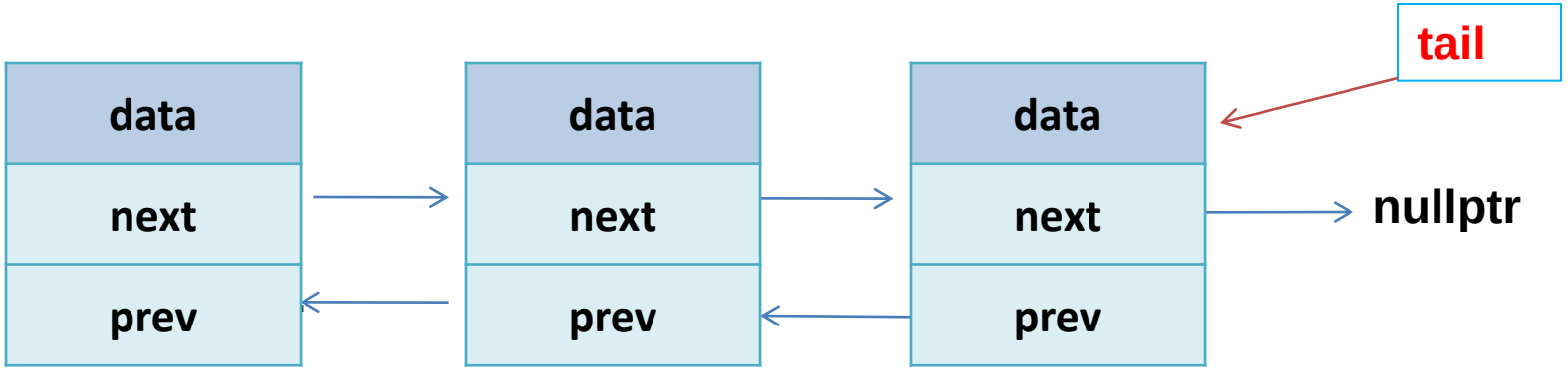


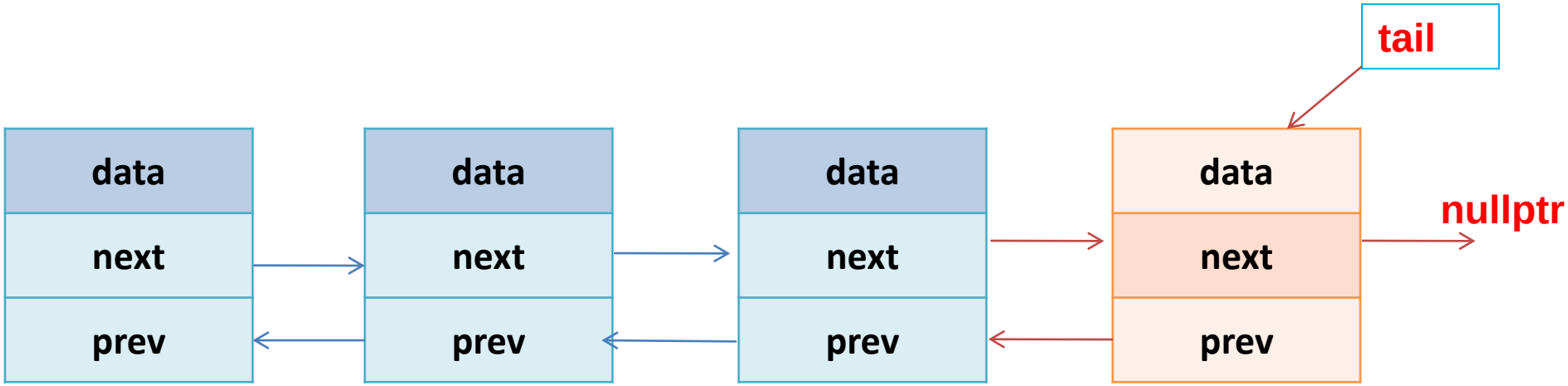


Вставка узла в конец (push_back)

- выделяется память под новый узел
- значение в новом узле устанавливается равным переданному значению
- next в новом узле указывает на nullptr
- previous в новом узле указывает на элемент, который был последним
- next в узле, который был последним, указывает на новый элемент
- tail указывает на новый элемент

Вставка узла в конец (схема)

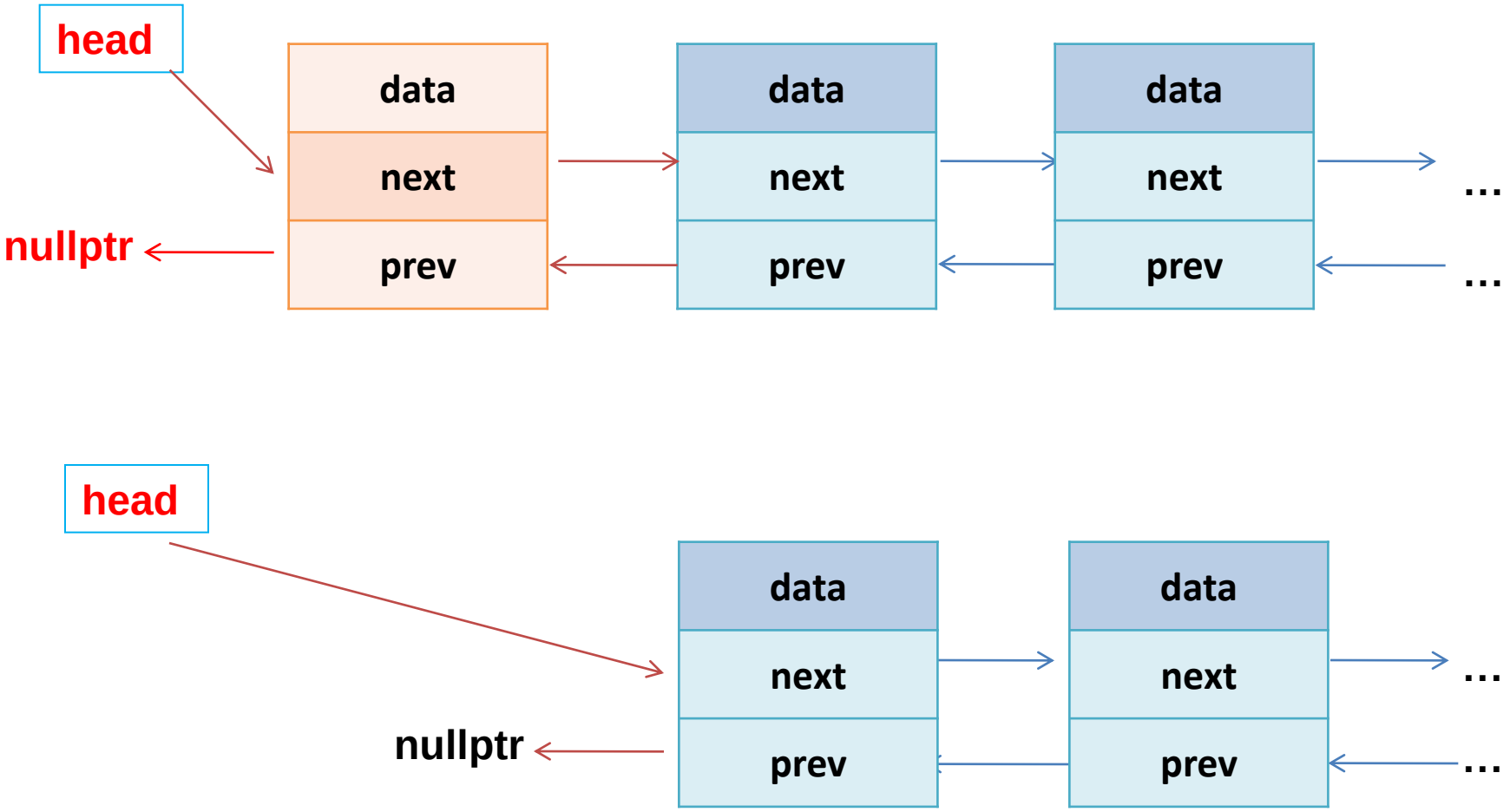




Удаление узла из начала (pop_front)

- требуется проверка, есть ли узлы в списке и не единственный ли узел нужно удалить
- previous в узле, который следовал за первым, устанавливается в nullptr
- head указывает на элемент, который следовал за первым
- освобождается память, которую занимал первый узел

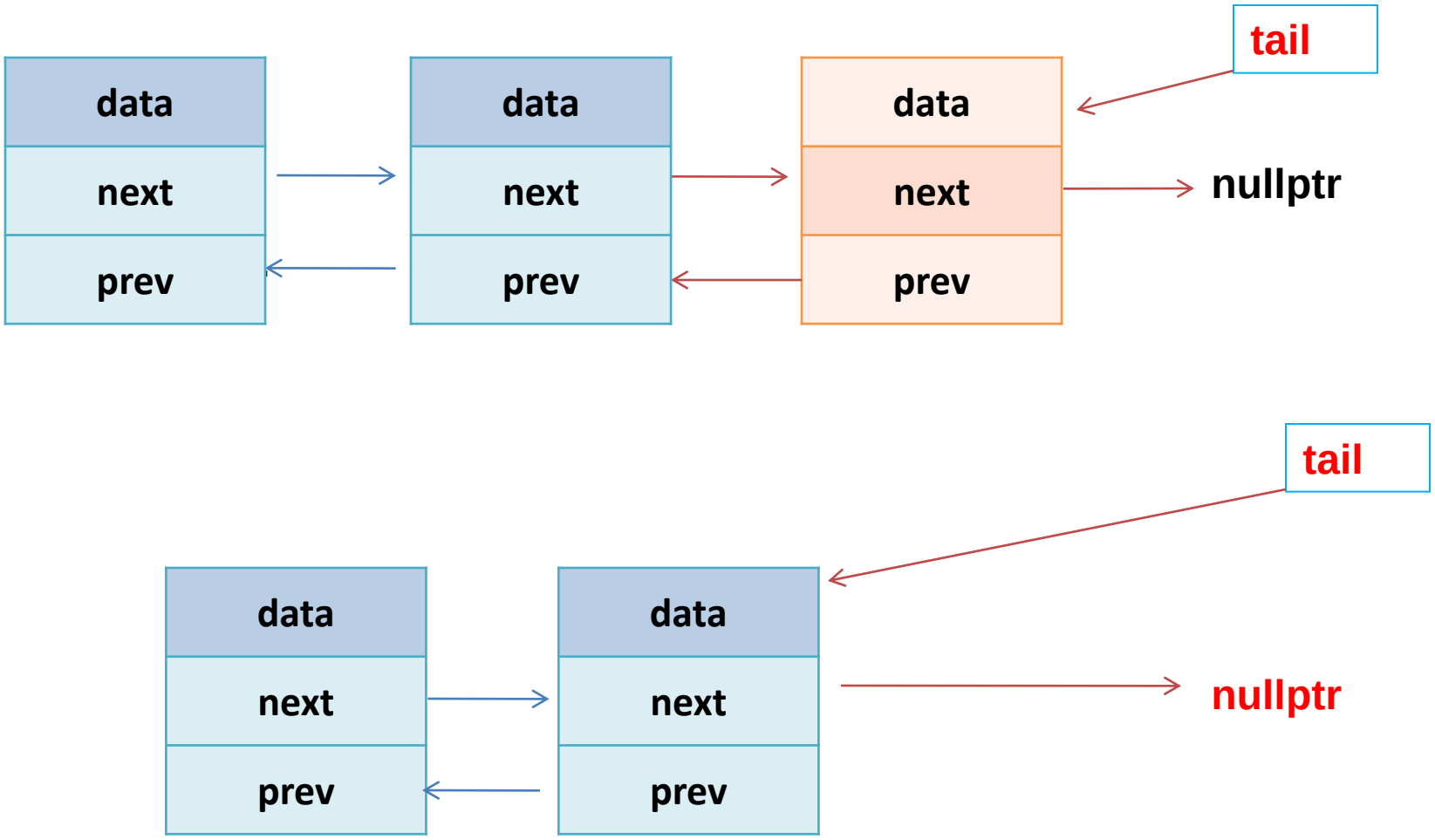
Удаление узла из начала (схема)



Удаление узла из конца (pop_back)

- требуется проверка, есть ли узлы в списке и не единственный ли узел нужно удалить
- tail указывает на элемент, который был предпоследним
- next в предпоследнем узле устанавливается в nullptr
- освобождается память, которую занимал последний узел

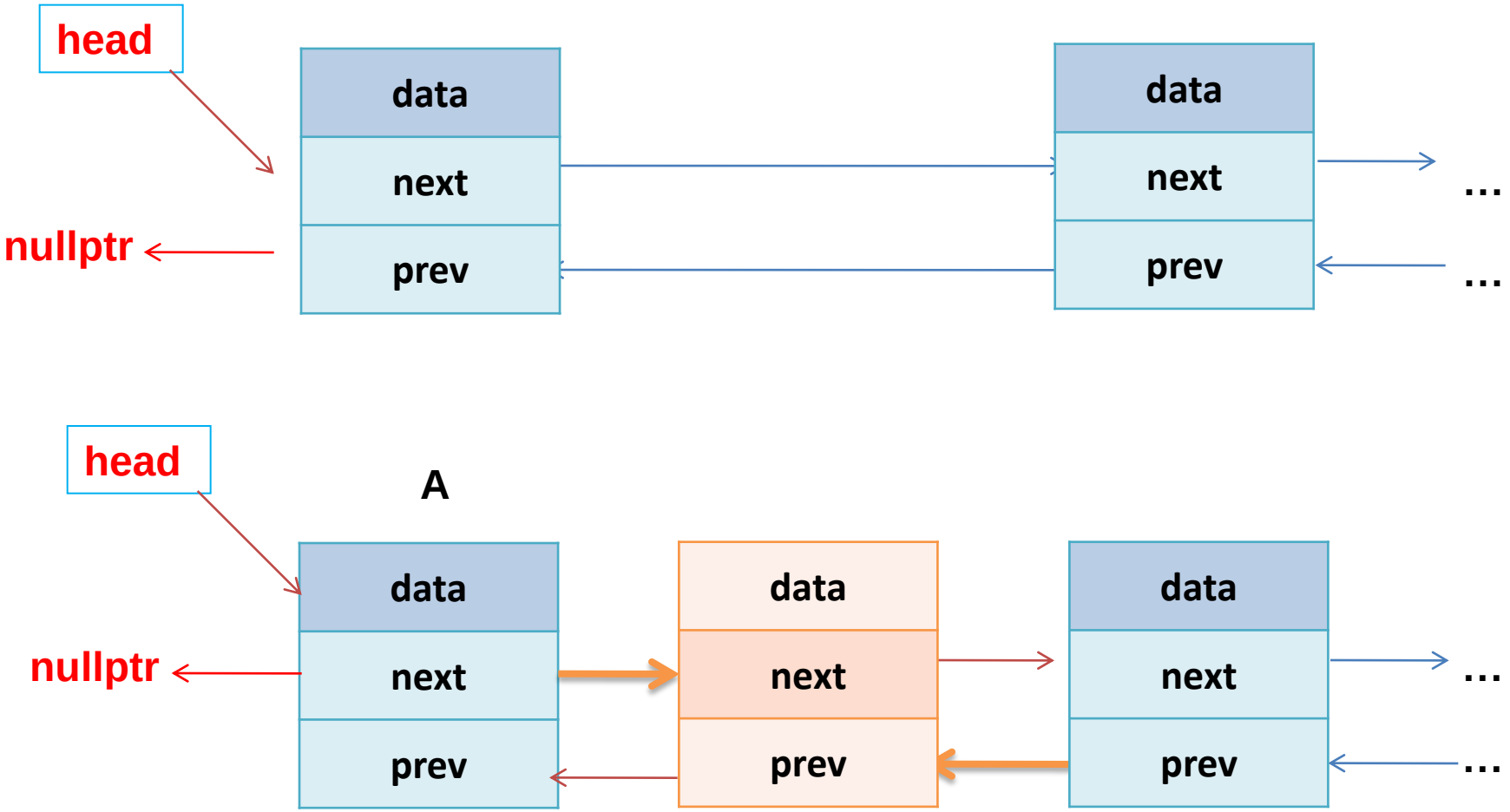
Удаление узла из конца (схема)



Вставка узла в середину (после существующего узла A)

- создается новый узел, значение в котором устанавливается равным переданному значению
- находится узел A, после которого нужно вставить новый узел
- previous в новом узле указывает на узел A
- next в новом узле указывает на узел, на который ранее указывал next из узла A
- previous в узле, следовавшем за A, указывает на новый узел
- next в узле A указывает на новый узел

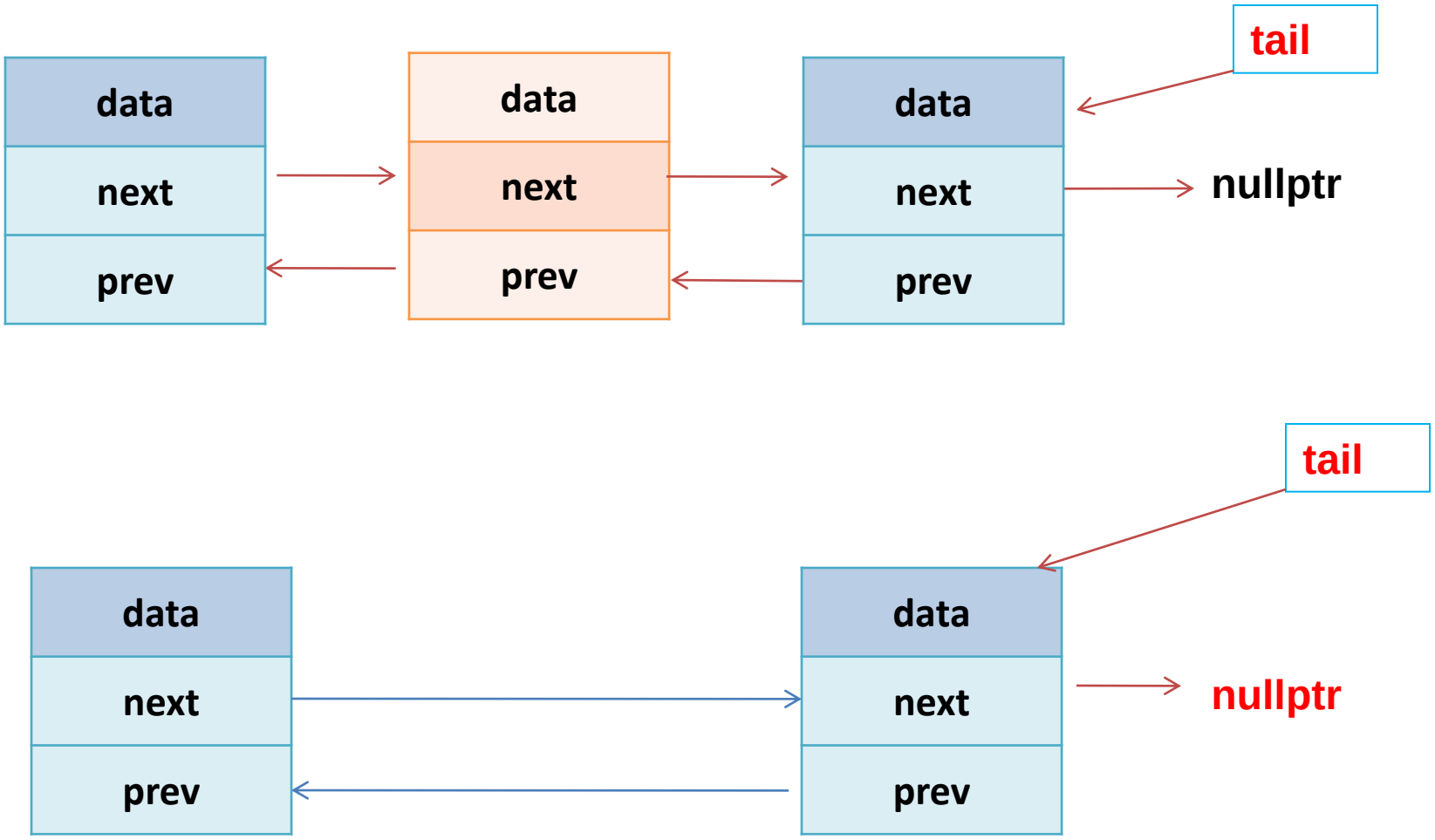
Вставка узла (схема)



Удаление узла из середины (заданного узла A)

- находится узел A, который нужно удалить
- next в предыдущем узле начинает указывать на узел за удаляемым узлом
- previous в узле за удаляемым узлом A начинает указывать на узел перед A
- освобождается память, которую занимал узел A

Удаление узла из середины (схема)



Преимущества и недостатки списков

+

- размер списка не ограничен размером свободного последовательного участка памяти
- эффективное динамическое добавление и удаление элементов

–

- на поля-указатели расходуется дополнительная память
- осуществляется только последовательный доступ к элементам
- => сложность прямого доступа к элементу

Вопросы?