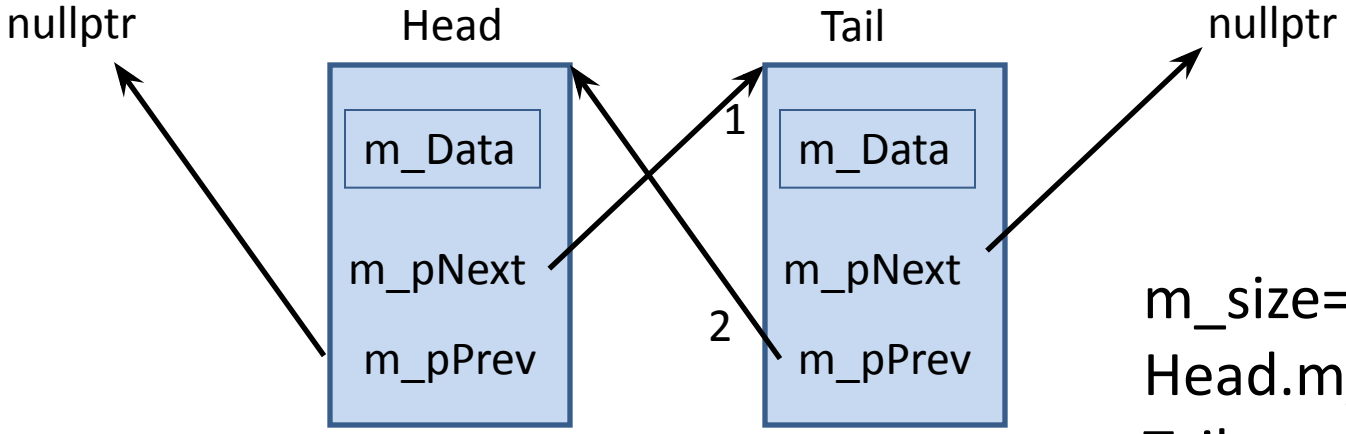
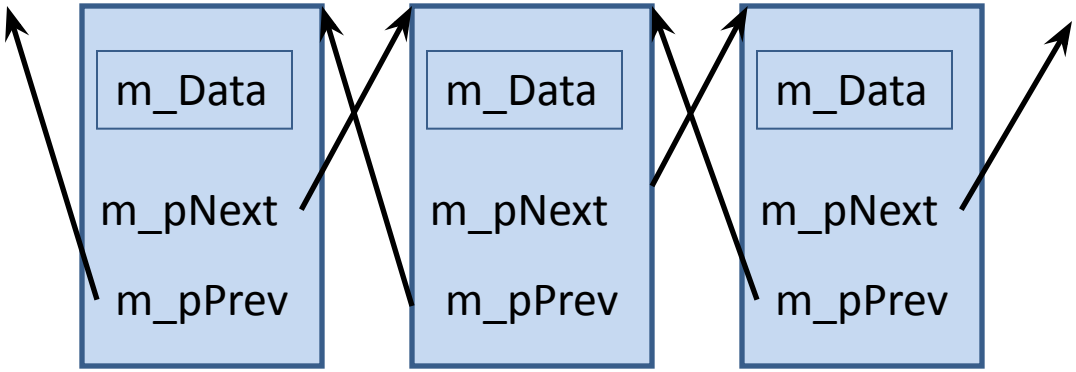
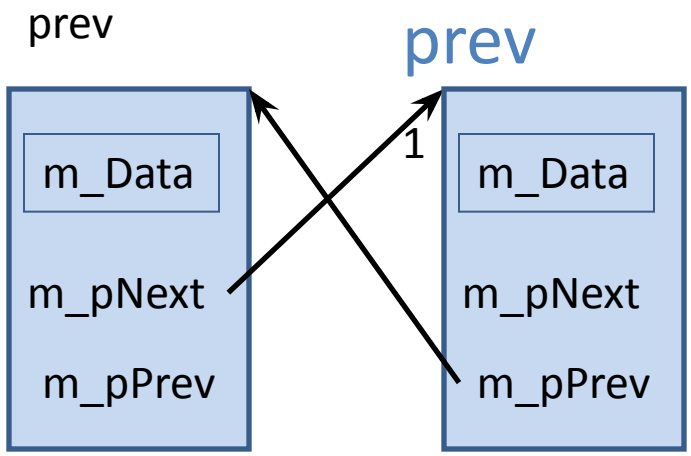
Двусвязный список



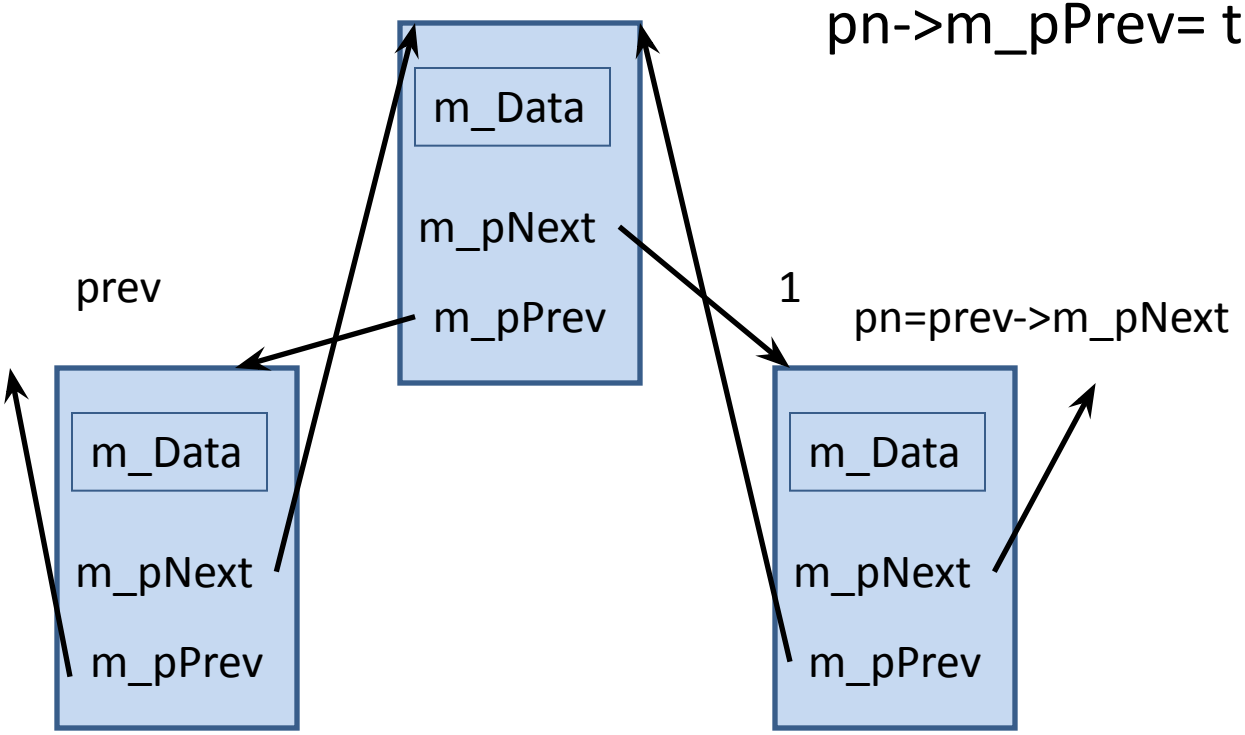
m_size=0;
Head.m_pNext=&Tail;
Tail.m_pPrev=&Head;



Добавление узла после prev

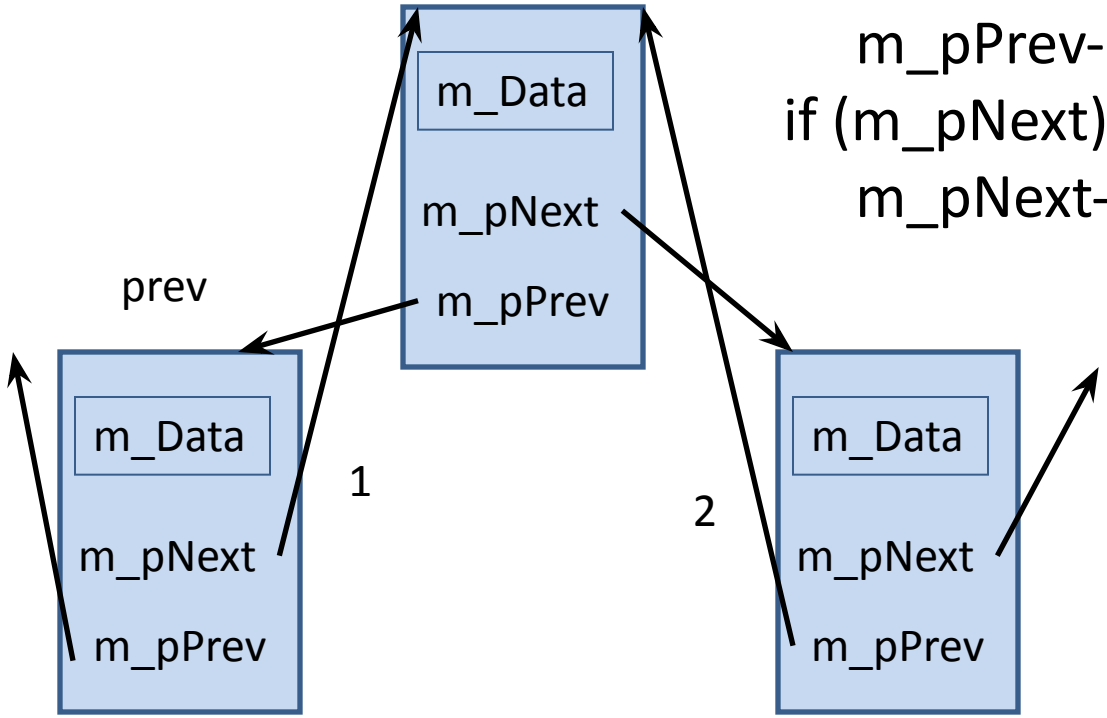


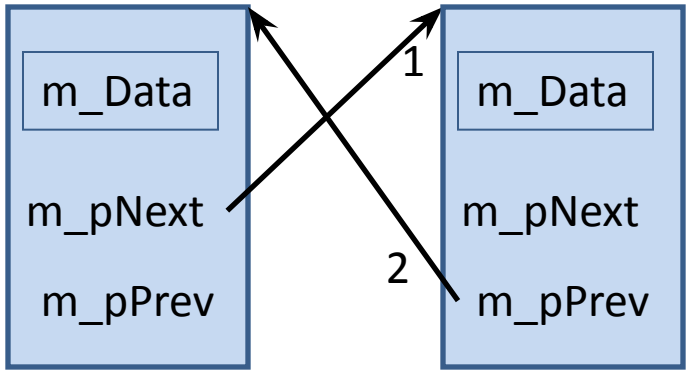

```
Node *pn;
pn= prev->m_pNext;
this->m_pPrev= prev;
this->m_pNext= pn;

prev->m_pNext= this;
pn->m_pPrev= this;
```



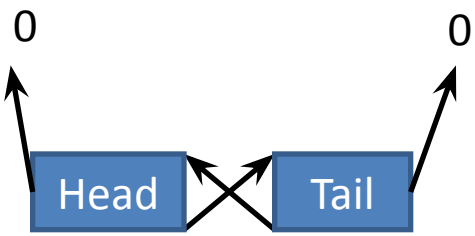
Исключение узла

```
if (m_pPrev)
    m_pPrev->m_pNext=this->m_pNext;
if (m_pNext)
    m_pNext->m_pPrev=this->m_pPrev;
```

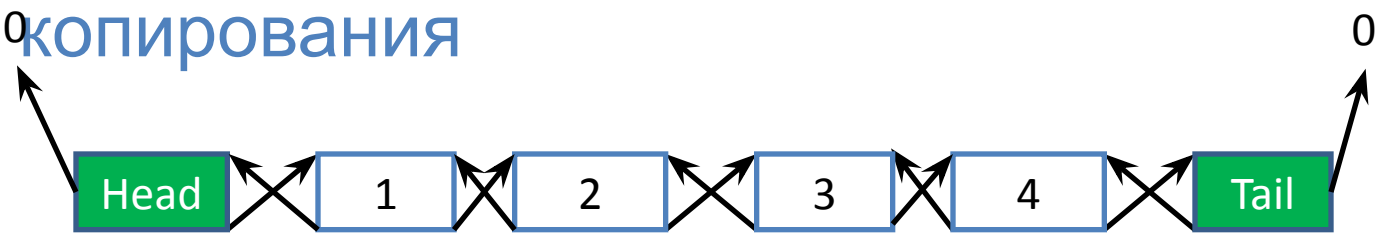


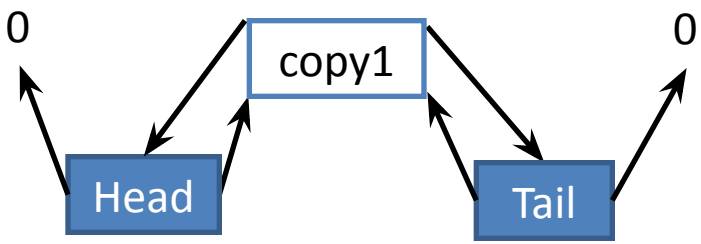


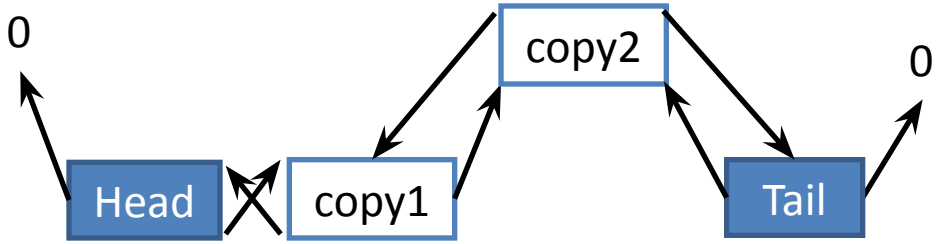
Конструктор копирования



size=0

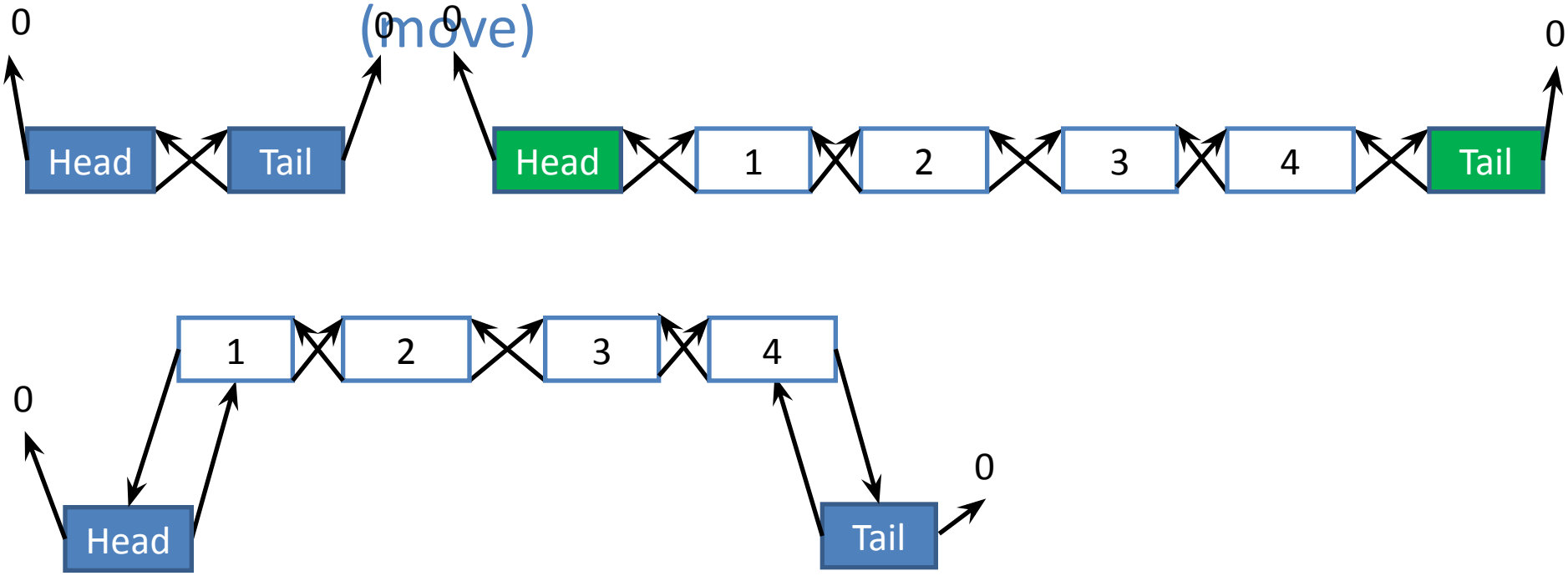






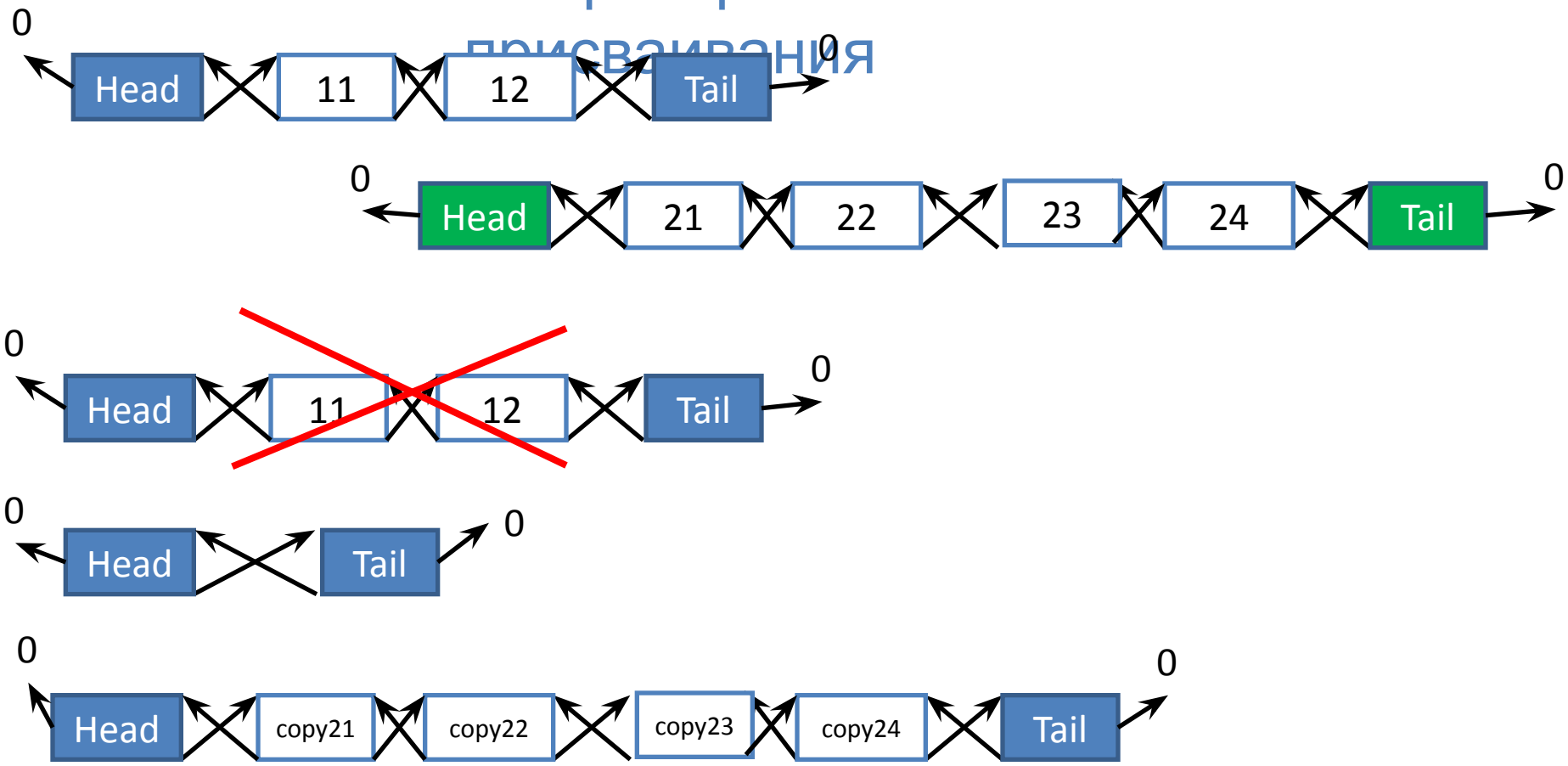

```
List::List (const List& other) //конструктор копирования
{
//1) связываем свои  Head  и  Tail;
//2) инициализируем поля класса
//3) считываем данные из другого списка, начиная с головы, а копируем  в конец
своего
}
```

Конструктор копирования (move)



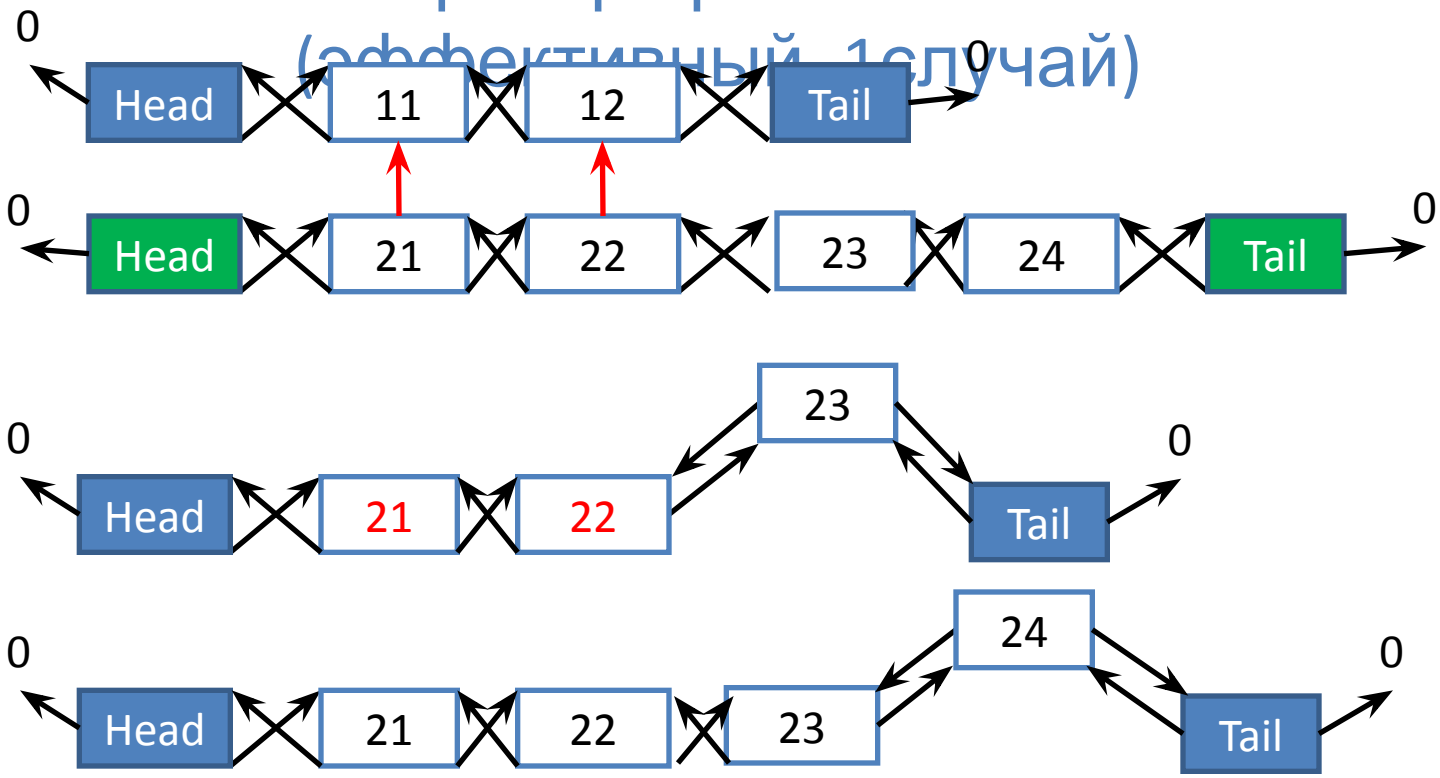

```
 //конструктор move- копирования
List::List (List&& other)
{
//1) забираем у "стражей" временного объекта весь список  //отобрали
//2) замыкаем стражей временного объекта друг на друга    // «обнулили»
}
```

Оператор присваивания



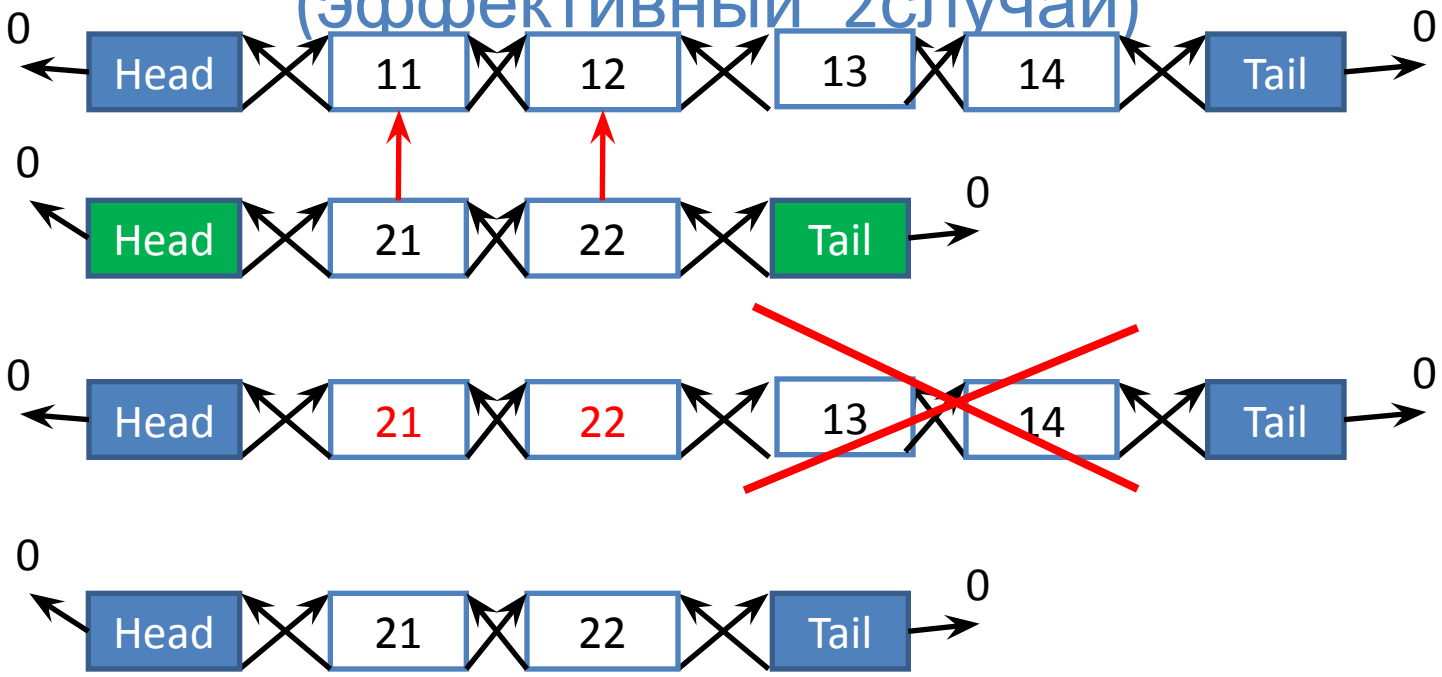

```
//оператор присваивания
List& List::operator=(const List& other)
{
//проверяем, что не себя копируем
//1) удаляем весь свой список (чистим)
//2) считываем данные из другого списка, начиная с головы, а копируем в конец
своего
}
```

Оператор присваивания (эффективный, 1случай)



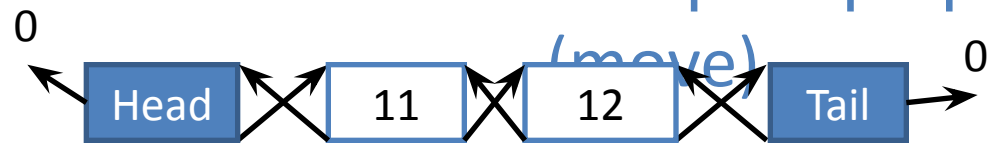

```
List& List::operator=(const List& other)  {
//проверяем, что не себя копируем
//1) определяем рабочие переменные для узлов, следующих после Head ( для каждого списка)
//2) определяем какой из списков имеет меньший размер: min_size
//3) копируем из other в «свой» список min_size элементов (в цикле)
//4) если свой список меньше другого, то добавляем недостающие данные в ХВОСТ  своего
//   иначе удаляем ненужные узлы  из своего списка
 }
```

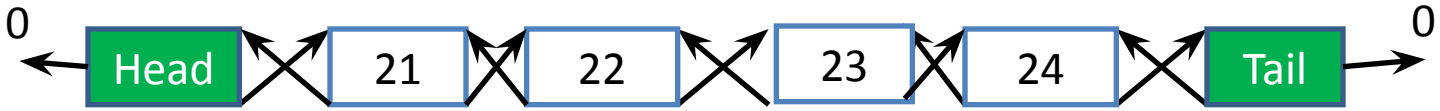
Оператор присваивания (эффективный 2случай)

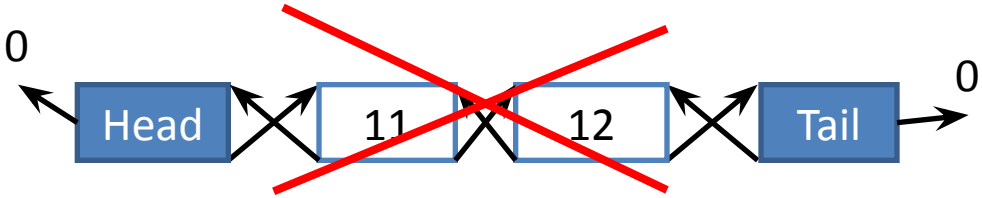


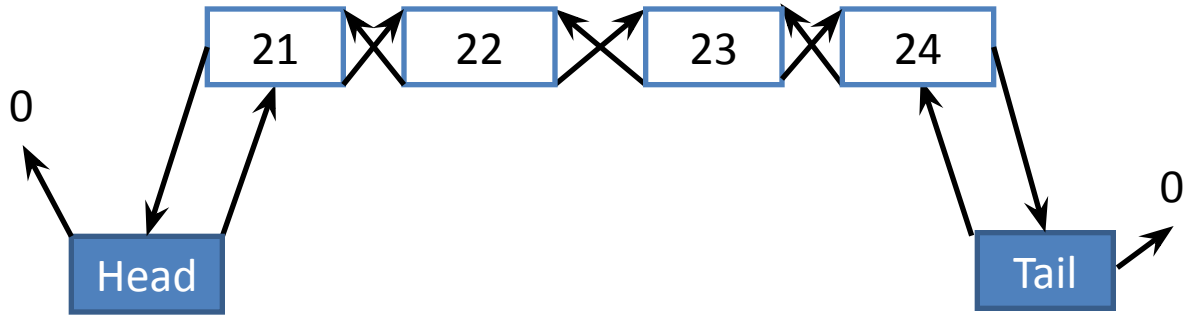

```
List& List::operator=(const List& other)  {
//проверяем, что не себя копируем
//1) определяем рабочие переменные для узлов, следующих после Head ( для каждого списка)
//2) определяем какой из списков имеет меньший размер: min_size
//3) копируем из other в «свой» список min_size элементов (в цикле)
//4) если свой список меньше другого, то  добавляем недостающие данные в ХВОСТ  своего
//   иначе удаляем ненужные узлы  из своего списка
 }
```

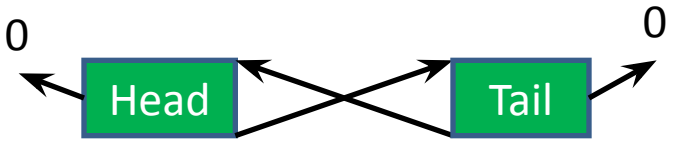
Оператор присваивания (move)












```
//оператор move-присваивания
List& List::operator=(List&& other)
{
//проверяем, что не себя копируем
//1) удаляем весь свой список (чистим)  // удалили
//2) забираем у "стражей" временного объекта весь список //отобрали
//3) замыкаем стражей временного объекта друг на друга     // «обнулили»
 }
```