Синхронизация потоков в Windows

Зачем нужна синхронизация

```
int count=0;
int data[10];

void StoreData(int element)
{
  data[count]=element;
  count++;
}
```

Зачем нужна синхронизация

Поток 1	Поток 2
StoreData(1):	StoreData(2):
Data[count]=element; count++;	Data[count]=element; count++;

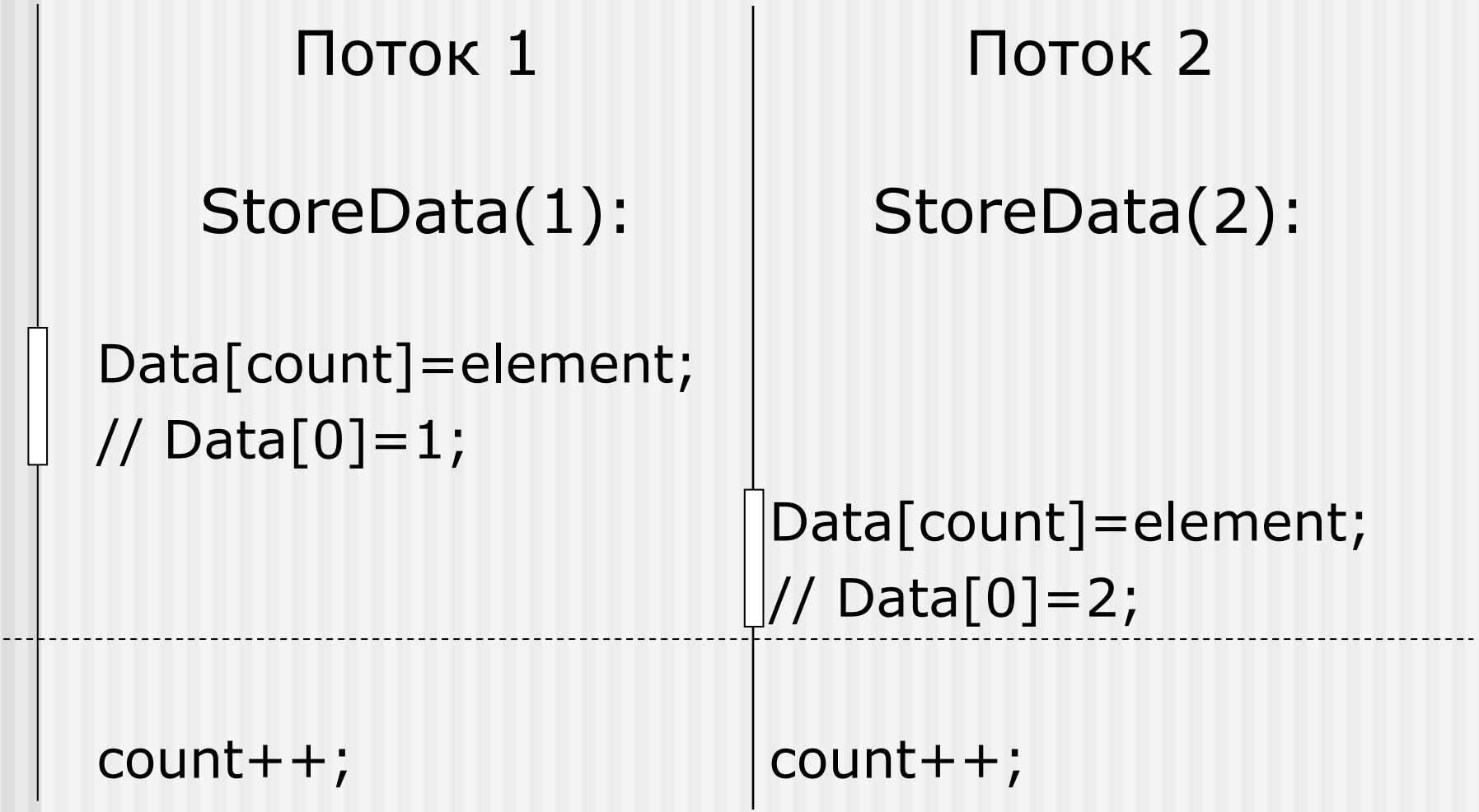
`Data[]={…}` `count=0`

Зачем нужна синхронизация

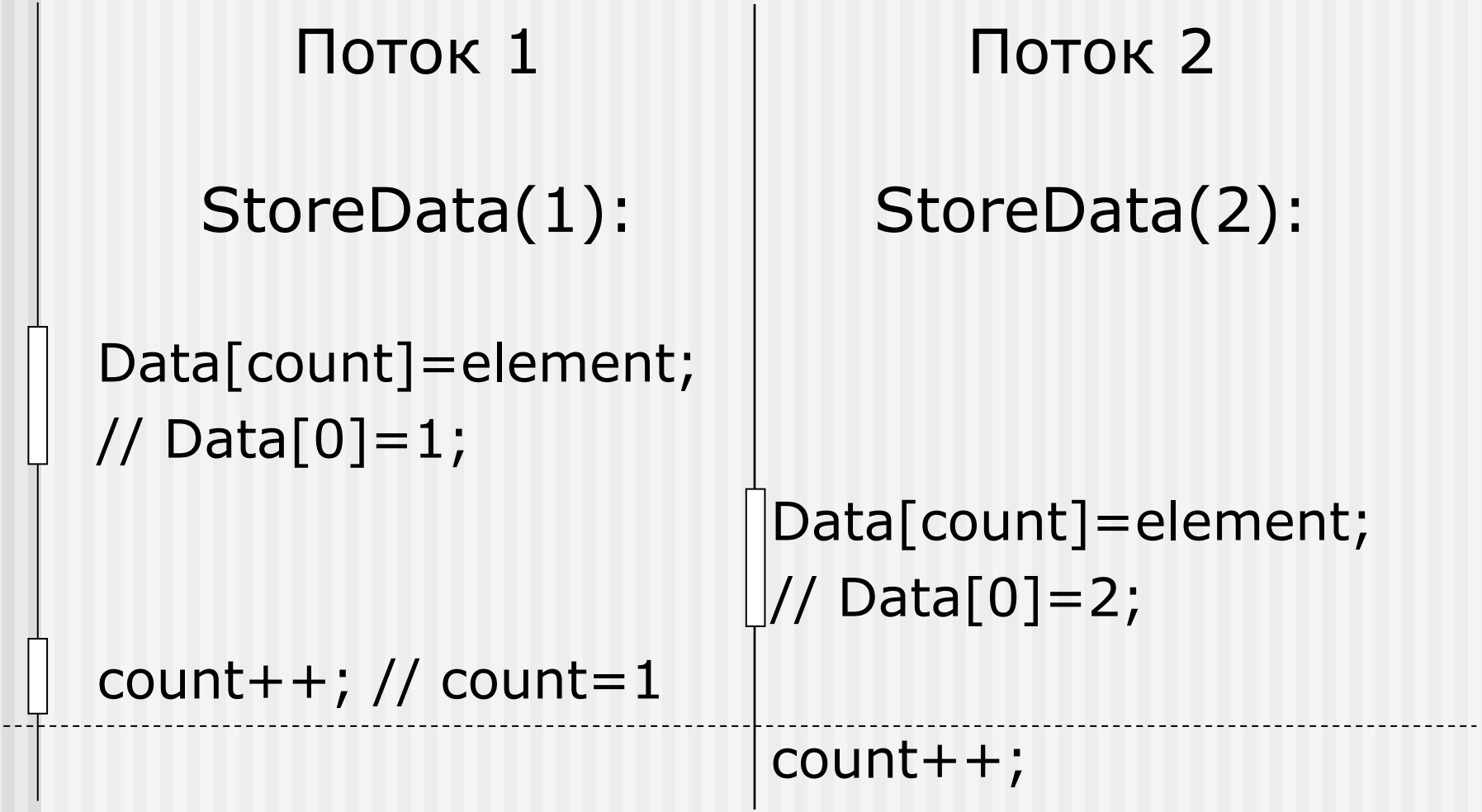
Поток 1	Поток 2
StoreData(1):	StoreData(2):
Data[count]=element; // Data[0]=1;	
	Data[count]=element;
count++;	count++;

`Data[]={1,…}` `count=0`

Зачем нужна синхронизация

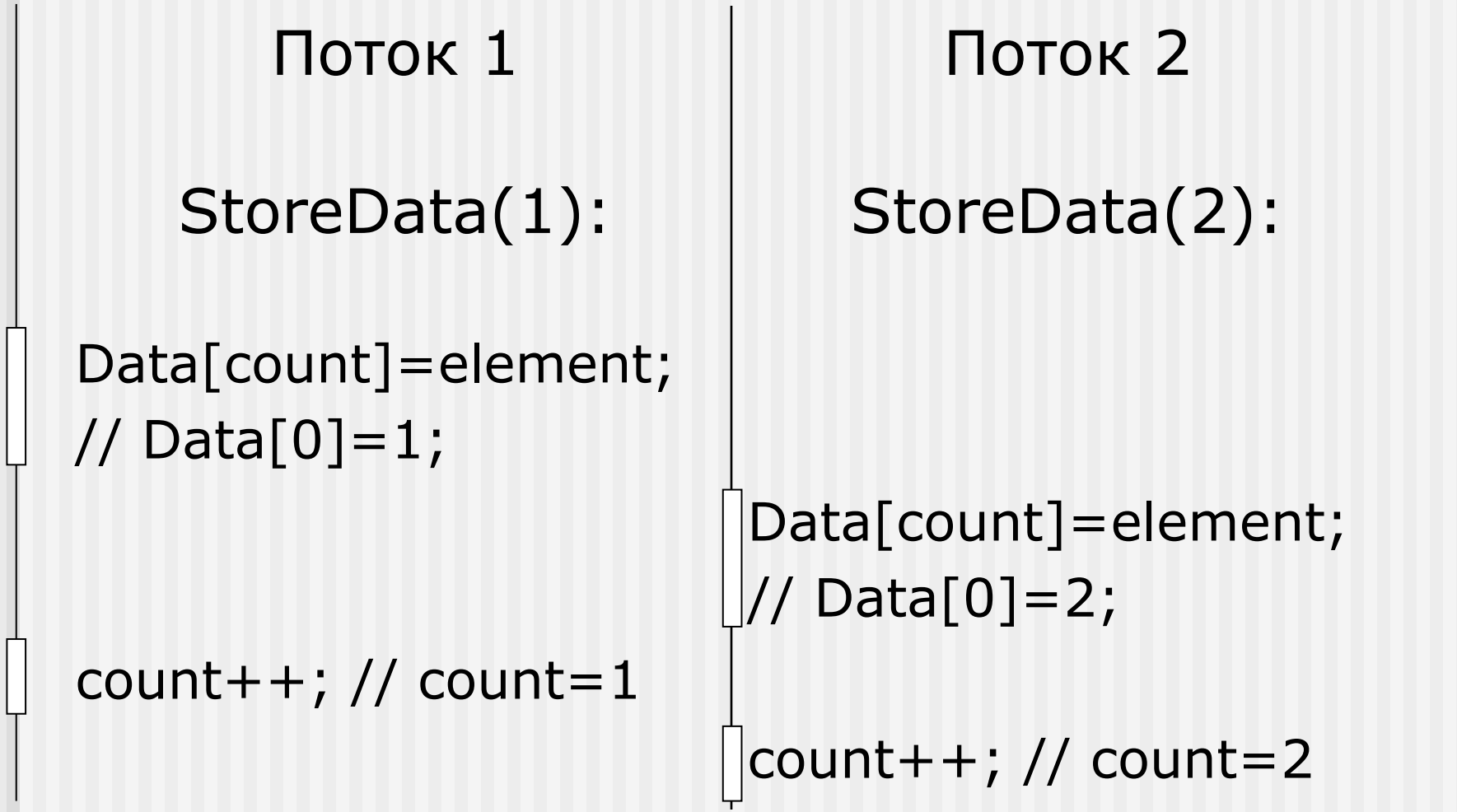


Зачем нужна синхронизация



`Data[]={2,…}` `count=1`

Зачем нужна синхронизация



`Data[]={2,?,…}  count=2`

Зачем нужна синхронизация

Поток 1

count++:

```
MOV AX,count
// ax=0

INC AX
//ax=1


MOV count,AX
// count=1
```

Поток 2

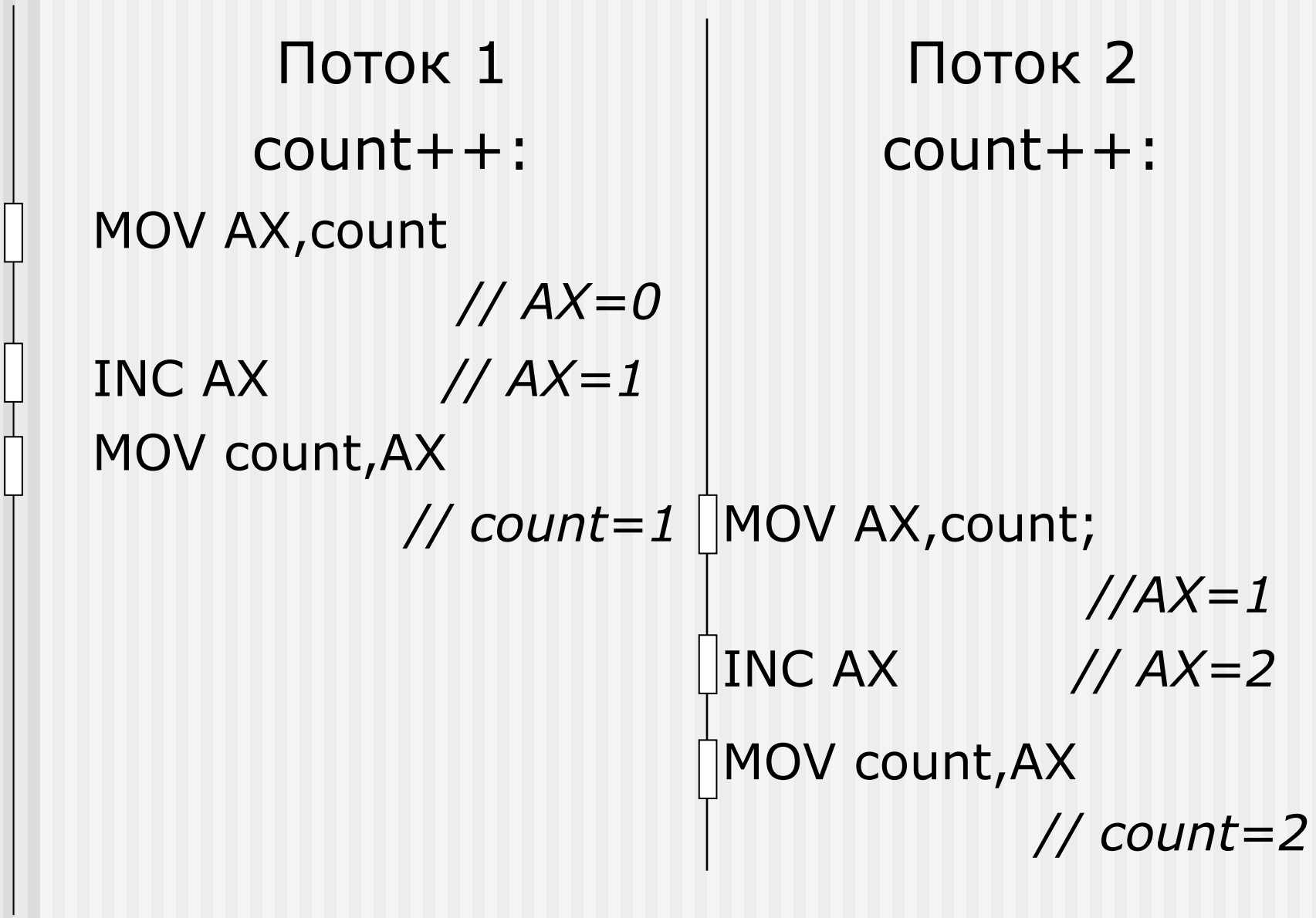
count++:

```
MOV AX,count;
// ax=0;

INC AX
//ax=1

MOV count,AX
// count=1
```

Зачем нужна синхронизация



Средства для синхронизации

- Interlocked функции
- Критические секции
- Объекты синхронизации и Wait-функции

Атомарные операции

- Выполняются за одну инструкцию над одним адресом в памяти:
  - читается старое значение;
  - вычисляется новое значение;
  - сохраняется новое значение.
- Аппаратное обеспечение гарантирует, что другие потоки не могут получить доступ к адресу памяти, с которым работает атомарная операция.
  - Другие потоки будут поставлены в очередь и будут ждать, пока не придёт их очередь выполнить операцию.
  - Атомарные операции над одним адресом памяти из разных потоков выполняются последовательно (возможно снижение производительности!).

Interlocked функции

Атомарное выполнение простейших операций:

- `LONG InterlockedDecrement(LPLONG Addend)`

```
Addend--;
return A
```

- `LONG Interloc`

```
LONG old=
Addend+=
return o
```

- `LONG Interloc`

```
Addend++
return A
```

```
LONG count=0;
int data[10];

void GoodStoreData(int element)
{
        LONG pos=InterlockedIncrement(&count);
        data[pos-1]=element;
}
```

```
LONG count=0;
int data[10];

void GoodStoreData3(int element)
{
        LONG pos=InterlockedExchangeAdd(&count,3);
        data[pos]=element;
        data[pos+1]=element;
        data[pos+2]=element;
}
```

Interlocked функции

Атомарное выполнение простейших операций:

- `LONG InterlockedExchange(LPLONG Value,LONG Target)`

```
LONG old=value;
value=target;
return value;
```

- `LONG InterlockedCompareExchange(LPLONG Destination,LONG Exchange,LONG Comperand)`

```
LONG old=Destination;
if(Destination==Comperand)
        Destination=Exchange;
return old;
```

Операция «Сравнение и замена»
(CAS, Compare And Swap)

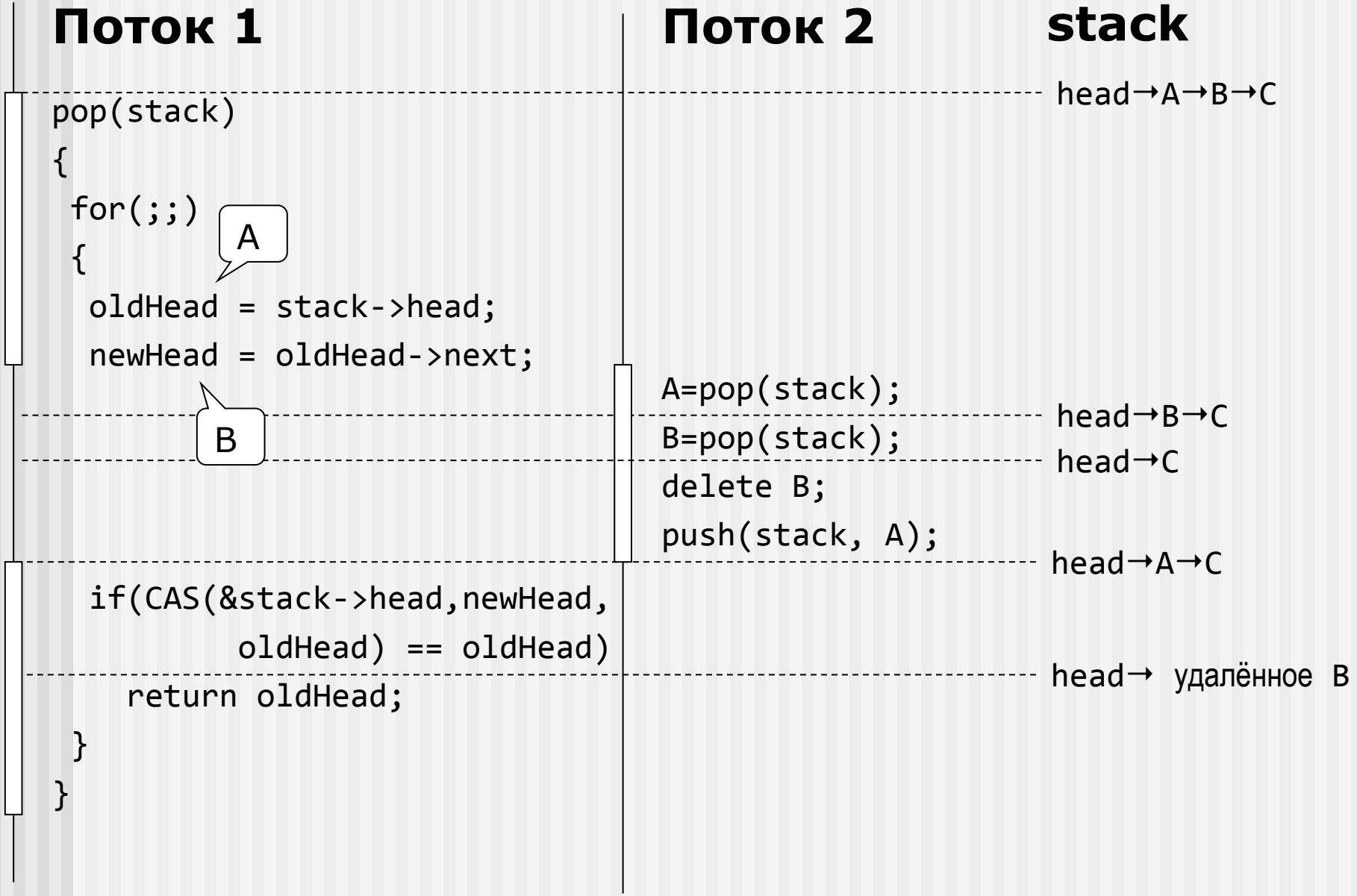
Почти потокобезопасная функция добавления элемента в очередь.

```
void LockFreeQueue::push(Node* newHead)
{
   for (;;)
   {
     // Копируем разделяемую переменную (m_Head) в локальную.
     Node* oldHead = m_Head;

     // Делаем изменения, которые пока не видны другим потокам.
     newHead->next = oldHead;

     // Пытаемся опубликовать наши изменения в разделяемую переменную.
     // Если разделяемая переменная не изменилась, CAS сработает
     // и функция завершает свою работу.
     // Иначе, повторяем всё с самого начала.
     if (InterlockedCompareExchange(&m_Head, newHead, oldHead) == oldHead)
         return;
   }
}
```

Проблема ABA



Решения проблемы ABA

- Тегирование данных
  - Используют младшие биты указателя как счётчик числа изменений.
  - До конца не спасает, так как этот счётчик может переполниться.
- Операция должна выполняться над промежуточными узлами, которые не являются пользовательскими данными и не должны повторяться.
  - Не надо забывать, что если что-то удалено delete, то одно из следующих new может вернуть этот же адрес.
- Использовать специальные значения (hazard pointer), чтобы сигнализировать о том, что структура находится в процессе изменения и требуется дополнительная синхронизация.
- Использовать более сложные инструкции процессора, если они имеются.