Lock-Free Queue

목차

- 시작하며…
- Lock-free
- Lock-free Queue
  - Push(T elem)
  - Pop( )
  - ABA 문제
- 성능 측정
- 마치며…
- 더 자세한 코드를 보고 싶으시다면...

시작하며...

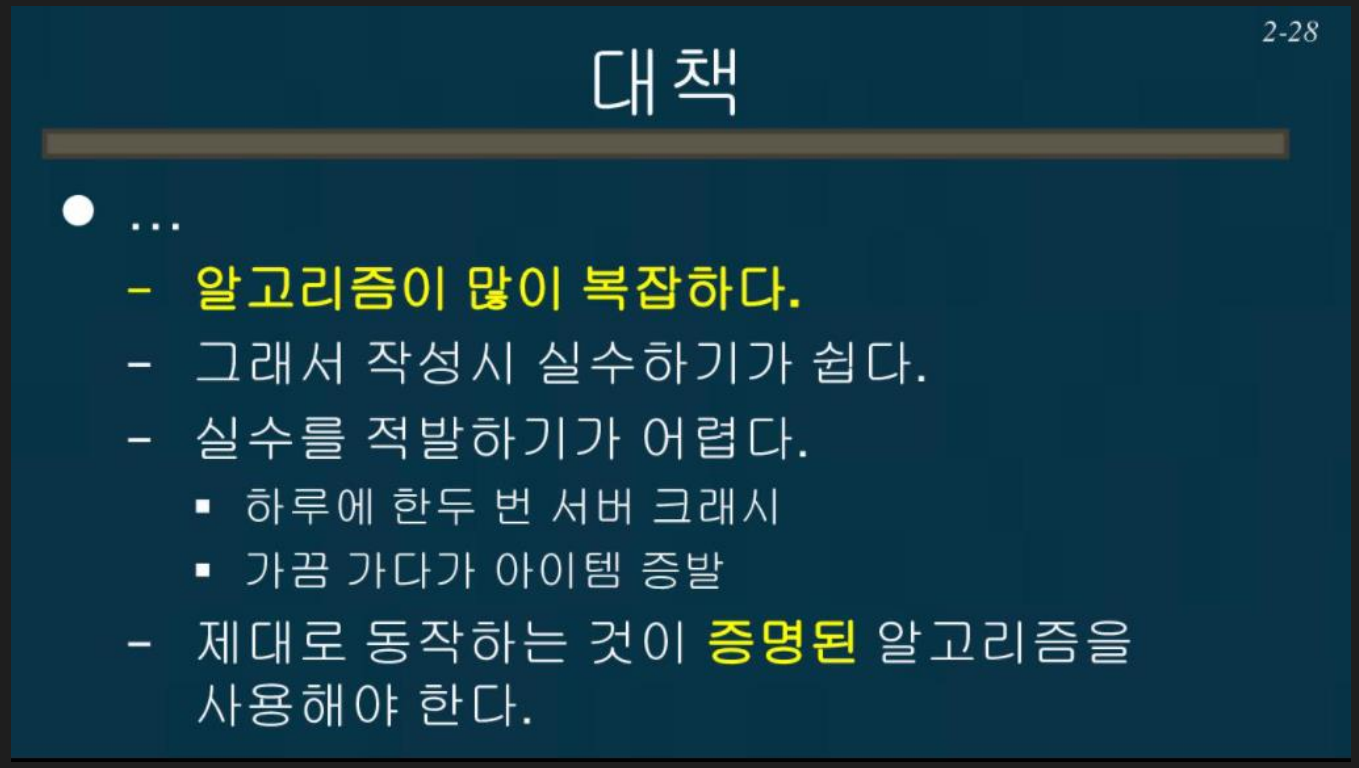
이 ppt는 개인 프로젝트에서 Task Queue로 사용하기 위해서 구현해 본 Lock-free Queue에 대해서 다룹니다.

Lock-free Queue의 코드가 왜 이렇게 짜여졌는지 알아보고 최종적으로 성능 체크도 해보도록 하겠습니다.

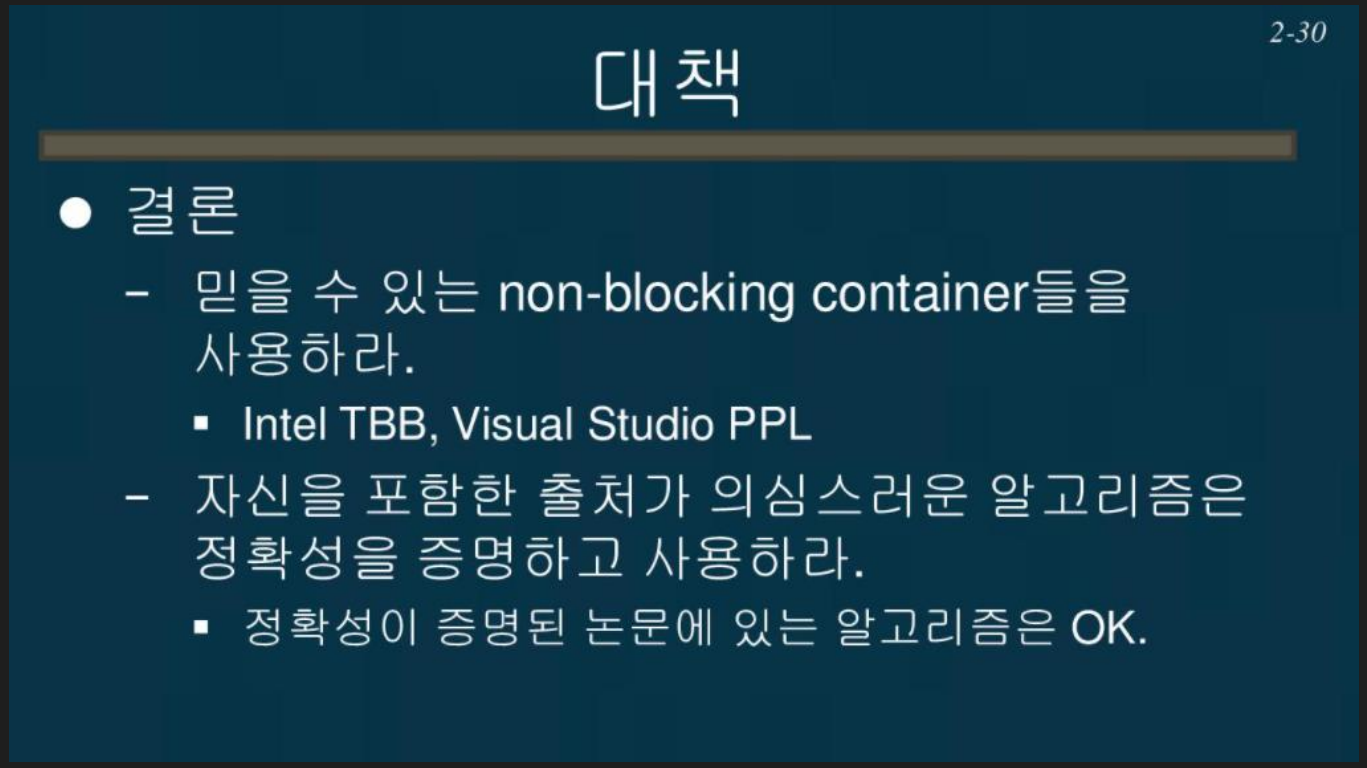
들어가기 전에 다음 사항을 참고 바랍니다.

시작하며...

- Ndc2014 시즌 2 : 멀티쓰레드 프로그래밍이 왜 이리 힘드나요? (Lock-free 에서 Transactional Memory까지) 에서 발췌



시작하며...



여기서 보여드리는 코드도 오류가 있을 수 있으니 신중히 보시길 권합니다.

Lock-free

- Lock-free는 Mutex와 같은 동기화 요소를 사용하지 않고 공유 데이터에 안전하게 접근하는 것을 뜻합니다.

- Lock-free는 Compare And Swap 이라고 하는 CAS 연산자를 통해 이뤄집니다.

```
1 bool CAS(destination, expected, desired)
2 {
3     if (destination == expected)
4     {
5         destination = desired;
6         return true;
7     }
8     else
9     {
10        return false;
11    }
12 }
```

Lock-free

- CAS는 변경 대상, 예상 값, 변경 값 을 인자로 받아서 변경 대상이 예상 값과 같으면 변경 값으로 값을 변경하고 예상 값과 다르다면 값을 변경하지 않는 연산입니다.

- 그리고 이러한 연산은 원자적으로 이뤄집니다.

```
1     const auto _Prev = static_cast<_Uint8_t>(_INTRIN_SEQ_CST(_InterlockedCompareExchange64)
  ((volatile long long *)_Tgt,
2 002F8C97  mov         eax,dword ptr [_Old_exp]
3 002F8C9A  mov         edx,dword ptr [ebp-4]
4 002F8C9D  mov         esi,dword ptr [_Tgt]
5 002F8CA0  mov         ecx,dword ptr [ebp+14h]
6 002F8CA3  mov         ebx,dword ptr [_Value]
7 002F8CA6  lock cmpxchg8b qword ptr [esi]
8 002F8CAA  mov         dword ptr [_Prev],eax
9 002F8CAD  mov         dword ptr [ebp-0Ch],edx
10        static_cast<long long>(_Value), static_cast<long long>(_Old_exp)));
```

lock prefix: 뒤의 명령어(cmpxchg8b)가 상호 배타적으로 수행되도록 함

Lock-free

- Lock-free의 기본적인 구현 방법은 이 CAS연산자를 이용해서 변경대상을 성공적으로 변경할 때 까지 반복하는 것입니다.

```
while(true)
{
    int expected = destination;
    if (CAS(destination, expected, desired))
    {
        // destination의 값을 바꾸는데 성공했다. 탈출
        break;
    }
    // 다른 스레드에서 destination의 값을 먼저 바꿨다. 재시도
}
```

Lock-free Queue

- 그럼 Lock-free Queue는 어떻게 구현할 수 있을까요.
  여기서 구현한 Queue는 2가지 메서드를 제공합니다.

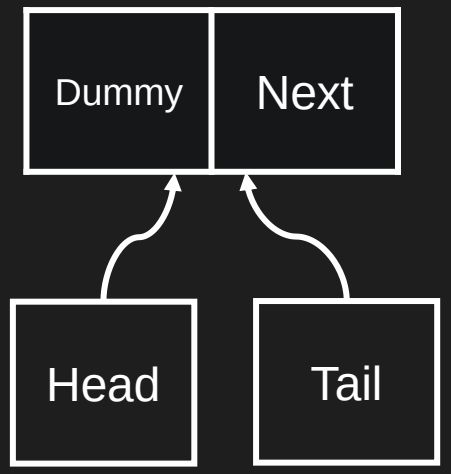
1. Push(T elem) : Queue의 끝에 원소를 삽입합니다.
2. T Pop( ) : Queue의 처음에서 원소를 빼냅니다.

- 이제 각 메서드가 어떻게 구현 됐는지 하나씩 보도록 하겠습니다.

Push(T elem)

- 실제 코드를 보기 전에 그림을 통해 Push 메서드의 동작을 알아보겠습니다.

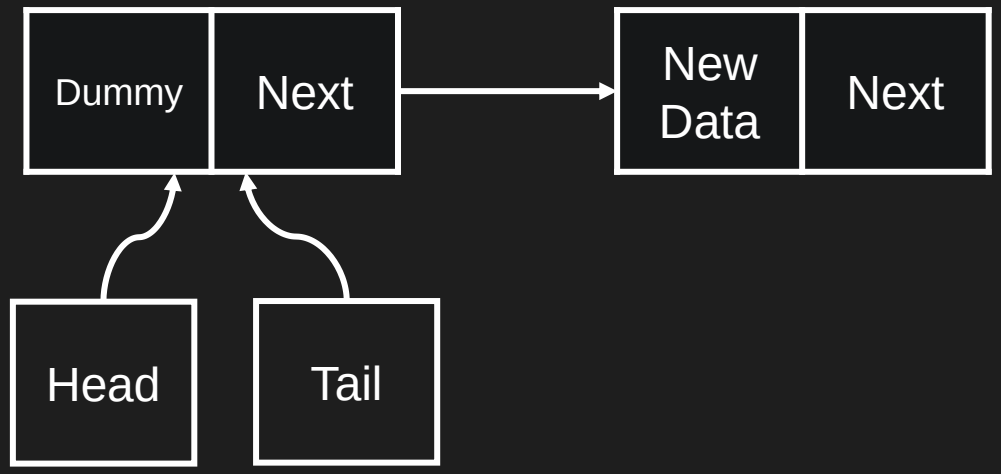
- 이 Queue는 생성되면 다음과 같이 데이터가 없는 더미 노드를 Head와 Tail이 가리키고 있는 초기 상태를 가집니다.



Push(T elem)

- 여기에 새로운 데이터를 Tail에 추가한다면…

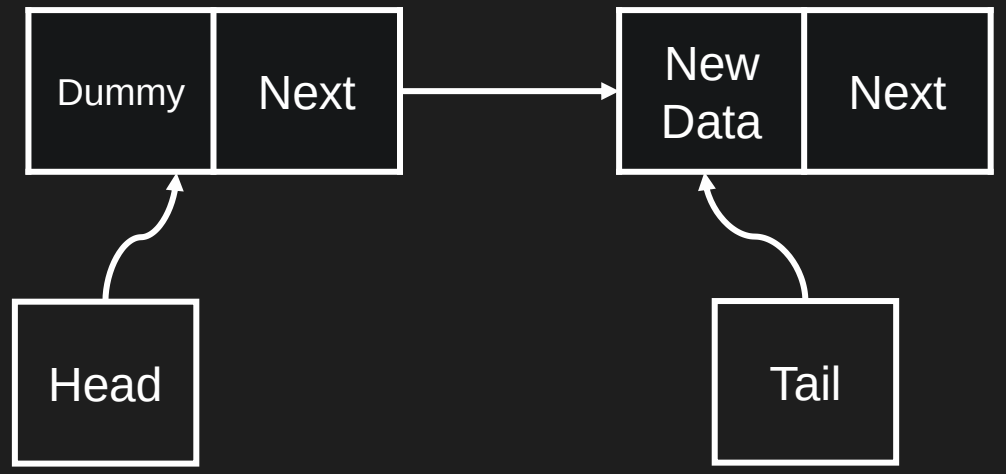
1. Tail이 가리키는 더미 노드의 Next가 새로운 데이터 노드를 가리키게 합니다.



Push(T elem)

2. 이제 Tail이 새로운 노드를 가리키도록 이동합니다.

메서드가 해야 할 일은 매우 간단합니다. 그럼 코드를 볼까요.



Push(T elem)

```
1 void Push( T* data )
2 {
3     unsigned int newData = LockFreeLinkPolicy::AllocLockFreeLink( );
4     LockFreeLinkPolicy::IndexToLink( newData )->m_data = data;
5
6     while ( true )
7     {
8         StampIndex last = m_tail;
9         IndexedLockFreeLink* lastLink = LockFreeLinkPolicy::IndexToLink( last.GetIndex( ) );
10        StampIndex next = lastLink->m_nextStampIndex;
11        StampIndex lastForTest = m_tail;
12        if ( last == lastForTest )
13        {
14            unsigned int nextIndex = next.GetIndex( );
15            if ( next.GetIndex( ) == 0 )
16            {
17                StampIndex newNext;
18                newNext.SetStamp( next.GetStamp( ) + 1 );
19                newNext.SetIndex( newData );
20
21                if ( lastLink->m_nextStampIndex.CompareExchange( next, newNext ) )
22                {
23                    StampIndex newTail;
24                    newTail.SetStamp( last.GetStamp( ) + 1 );
25                    newTail.SetIndex( newData );
26
27                    m_tail.CompareExchange( last, newTail );
28                    return;
29                }
30            }
31            else
32            {
33                StampIndex newTail;
34                newTail.SetStamp( last.GetStamp( ) + 1 );
35                newTail.SetIndex( next.GetIndex( ) );
36
37                m_tail.CompareExchange( last, newTail );
38            }
39        }
40    }
41 }
```

- Push 메서드의 전체 코드입니다.
- lock-free 알고리즘에서 발생할 수 있는 문제를 해결하기 위한 추가 코드로 인해 앞에서 알아본 Push 메서드의 동작에 비해 코드가 복잡해 졌습니다.
- 우선은 Push 메서드의 동작에 집중하여 코드를 살펴보겠습니다.

Push(T elem)

```
1  void Push( T* data )
2  {
1      unsigned int newData = LockFreeLinkPolicy::AllocLockFreeLink( );
       LockFreeLinkPolicy::IndexToLink( newData )->m_data = data;
5
6      while ( true )
7      {
2          StampIndex last = m_tail;
           IndexedLockFreeLink* lastLink = LockFreeLinkPolicy::IndexToLink( last.GetIndex( ) );
10         StampIndex next = lastLink->m_nextStampIndex;
11         StampIndex lastForTest = m_tail;
12         if ( last == lastForTest )
13         {
14             unsigned int nextIndex = next.GetIndex( );
15             if ( next.GetIndex( ) == 0 )
16             {
17                 StampIndex newNext;
18                 newNext.SetStamp( next.GetStamp( ) + 1 );
19                 newNext.SetIndex( newData );
20
21                 if ( lastLink->m_nextStampIndex.CompareExchange( next, newNext ) )
22                 {
23                     StampIndex newTail;
24                     newTail.SetStamp( last.GetStamp( ) + 1 );
25                     newTail.SetIndex( newData );
26
27                     m_tail.CompareExchange( last, newTail );
28                     return;
29                 }
30             }
31             else
32             {
33                 StampIndex newTail;
34                 newTail.SetStamp( last.GetStamp( ) + 1 );
35                 newTail.SetIndex( next.GetIndex( ) );
36
37                 m_tail.CompareExchange( last, newTail );
38             }
39         }
40     }
41 }
```

1. 새로운 데이터를 위한 노드를 할당합니다.

2. 이제 삽입 시도가 시작됩니다. 우선 Tail이 가리키고 있는 노드(last)와 해당 노드의 다음 노드(next)를 지역 변수에 저장해 놓습니다.

Push(T elem)

```
1  void Push( T* data )
2  {
3      unsigned int newData = LockFreeLinkPolicy::AllocLockFreeLink( );
4      LockFreeLinkPolicy::IndexToLink( newData )->m_data = data;
5
6      while ( true )
7      {
8          StampIndex last = m_tail;
9          IndexedLockFreeLink* lastLink = LockFreeLinkPolicy::IndexToLink( last.GetIndex( ) );
10         StampIndex next = lastLink->m_nextStampIndex;
11         StampIndex lastForTest = m_tail;
12 ③       if ( last == lastForTest )
13         {
14 ④           unsigned int nextIndex = next.GetIndex( );
15             if ( next.GetIndex( ) == 0 )
16             {
17                 StampIndex newNext;
18                 newNext.SetStamp( next.GetStamp( ) + 1 );
19                 newNext.SetIndex( newData );
20
21                 if ( lastLink->m_nextStampIndex.CompareExchange( next, newNext ) )
22                 {
23                     StampIndex newTail;
24                     newTail.SetStamp( last.GetStamp( ) + 1 );
25                     newTail.SetIndex( newData );
26
27                     m_tail.CompareExchange( last, newTail );
28                     return;
29                 }
30             }
31             else
32             {
33                 StampIndex newTail;
34                 newTail.SetStamp( last.GetStamp( ) + 1 );
35                 newTail.SetIndex( next.GetIndex( ) );
36
37                 m_tail.CompareExchange( last, newTail );
38             }
39         }
40     }
41 }
```

3. Tail이 변하지 않았는지 검사합니다.
Tail이 변했다면 이전에 읽어 놓은 next도 정상적인 값이 아니기 때문에 재시도 해야 합니다.

4. next의 nullptr여부에 따라 따른 동작을 하게 됩니다.
우선 nullptr 이 아닐 때를 보겠습니다.

Push(T elem)

```
1  void Push( T* data )
2  {
3      unsigned int newData = LockFreeLinkPolicy::AllocLockFreeLink( );
4      LockFreeLinkPolicy::IndexToLink( newData )->m_data = data;
5
6      while ( true )
7      {
8          StampIndex last = m_tail;
9          IndexedLockFreeLink* lastLink = LockFreeLinkPolicy::IndexToLink( last.GetIndex( ) );
10         StampIndex next = lastLink->m_nextStampIndex;
11         StampIndex lastForTest = m_tail;
12         if ( last == lastForTest )
13         {
14             unsigned int nextIndex = next.GetIndex( );
15             if ( next.GetIndex( ) == 0 )
16             {
17                 StampIndex newNext;
18                 newNext.SetStamp( next.GetStamp( ) + 1 );
19                 newNext.SetIndex( newData );
20
21                 if ( lastLink->m_nextStampIndex.CompareExchange( next, newNext ) )
22                 {
23                     StampIndex newTail;
24                     newTail.SetStamp( last.GetStamp( ) + 1 );
25                     newTail.SetIndex( newData );
26
27                     m_tail.CompareExchange( last, newTail );
28                     return;
29                 }
30             }
31 ⑤           else
32             {
33                 StampIndex newTail;
34                 newTail.SetStamp( last.GetStamp( ) + 1 );
35                 newTail.SetIndex( next.GetIndex( ) );
36
37                 m_tail.CompareExchange( last, newTail );
38             }
39         }
40     }
41 }
```

변경 대상 / 예상 값 / 변경 값

= m_tail의 값이 last와 같으면 m_tail을 newTail로 갱신

5. next가 nullptr이 아니라는 의미는 다른 스레드가 Tail이 가리키는 노드의 next는 변경했지만 Tail을 아직 변경하지 못한 경우입니다.

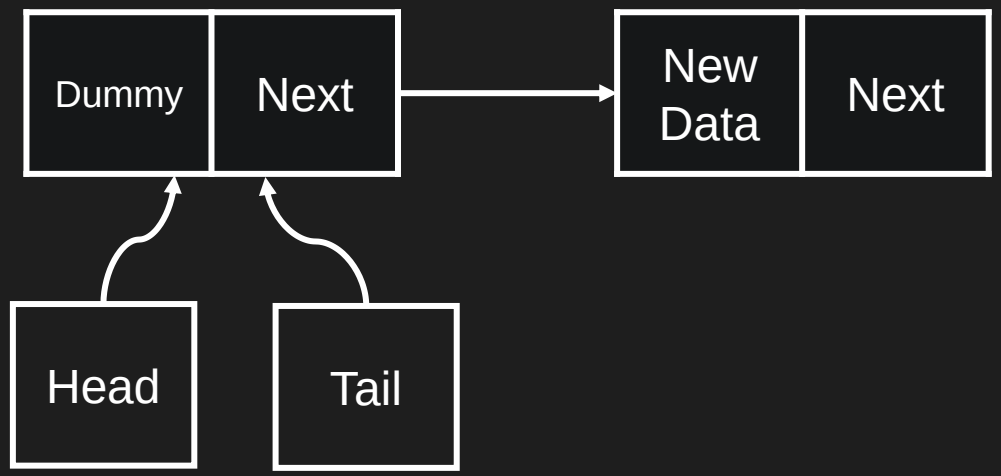
이 경우에는 능동적으로 tail을 next로 변경해 줍니다.

왜 이렇게 해야 하는지 그림으로 살펴보겠습니다.

Push(T elem)

m_tail.CompareExchange(last, newTail)이 실행되기 전 상태는 아래 그림과 같습니다.

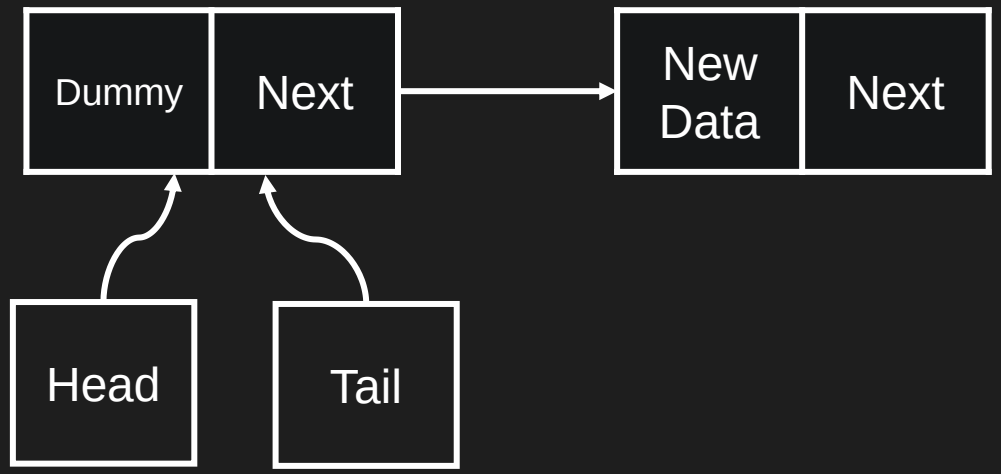
Next에 새로운 노드를 연결했지만 Tail은 갱신되지 않은 상태입니다. 이 상태에서 스케줄러에 의해 실행이 다른 스레드로 넘어 갔다고 생각해봅시다.



Push(T elem)

- 다른 스레드에서도 새로운 데이터를 삽입하고자 합니다. Queue는 삽입 순서가 보장되야 합니다. 먼저 삽입된 데이터 보다 앞에 삽입할 수 없습니다.

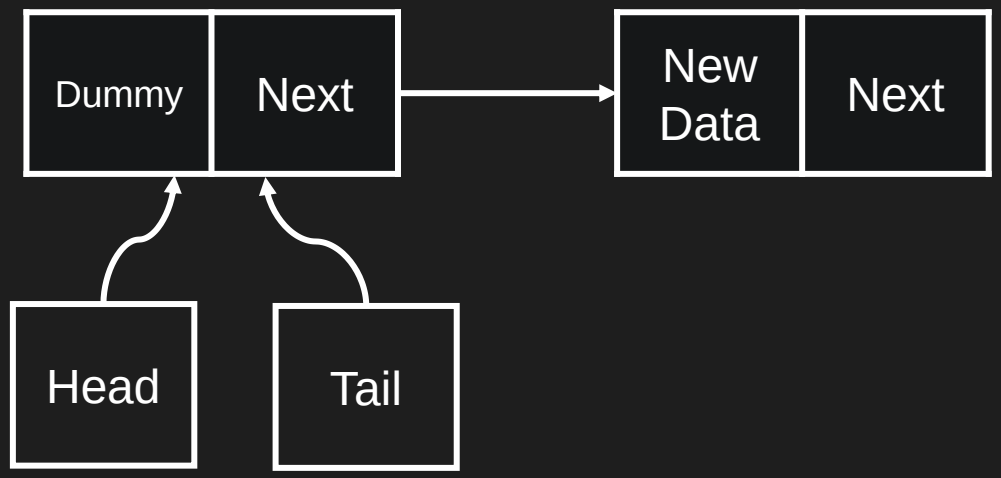
- 현재 Queue의 Tail이 가리키는 노드의 Next가 nullptr이 아니기 때문에 tail의 위치에 값을 삽입할 수 없습니다.



Push(T elem)

- 여기서 2가지 선택을 할 수 있습니다.

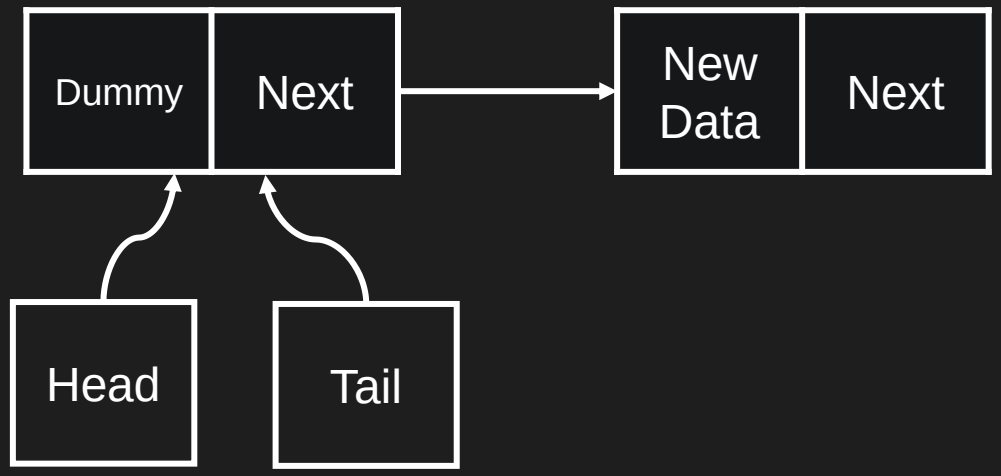
1. 원래 스레드가 Tail을 이동시킬 때 까지 기다린다.
2. 자신이 직접 Tail을 옮긴다.



Push(T elem)

- 원래 스레드가 Tail을 옮기기를 기다린다면 이제 다른 모든 스레드는 원래 스레드의 작업을 기다리는 블로킹 상태가 됩니다.

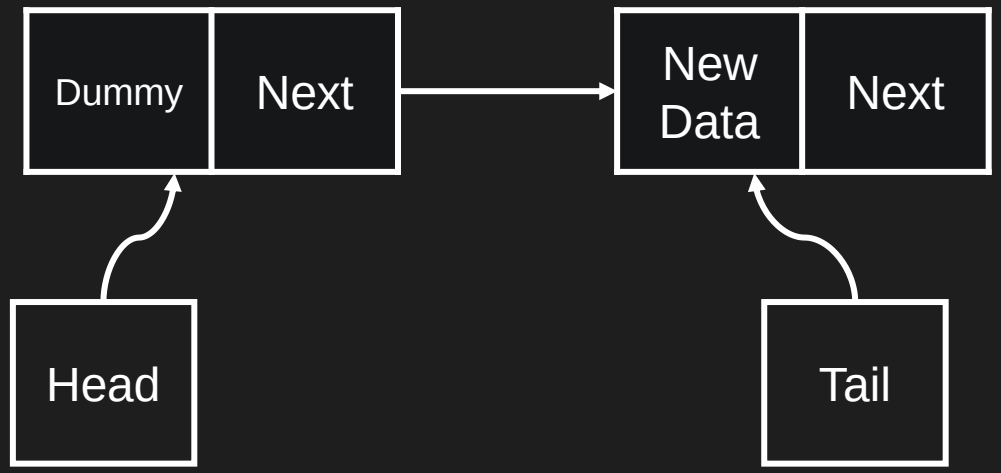
- 이후 발생하는 모든 Push연산은 원래 스레드가 다시 실행 되기 전까지 모두 실패하게 됩니다. 이런 상황을 Convoying이라고 합니다.



Push(T elem)

- 이런 블로킹 알고리즘은 이제 더 이상 Lock-free라고 할 수 없습니다.

- Lock-free는 다른 스레드의 상태에 상관 없이 실행되야 합니다. 그래서 능동적으로 Tail을 옮깁니다. 본래의 스레드는 로컬에 저장중인 last가 Tail 과 달라졌기 때문에 CAS가 실패하게 되므로 문제 없이 동작합니다.



Push(T elem)

```
1  void Push( T* data )
2  {
3      unsigned int newData = LockFreeLinkPolicy::AllocLockFreeLink( );
4      LockFreeLinkPolicy::IndexToLink( newData )->m_data = data;
5
6      while ( true )
7      {
8          StampIndex last = m_tail;
9          IndexedLockFreeLink* lastLink = LockFreeLinkPolicy::IndexToLink( last.GetIndex( ) );
10         StampIndex next = lastLink->m_nextStampIndex;
11         StampIndex lastForTest = m_tail;
12         if ( last == lastForTest )
13         {
14             unsigned int nextIndex = next.GetIndex( );
15             if ( next.GetIndex( ) == 0 )
16             {
17                 StampIndex newNext;
18                 newNext.SetStamp( next.GetStamp( ) + 1 );
19                 newNext.SetIndex( newData );
20
21 ⑥              if ( lastLink->m_nextStampIndex.CompareExchange( next, newNext )
22                 {
23                     StampIndex newTail;
24                     newTail.SetStamp( last.GetStamp( ) + 1 );
25                     newTail.SetIndex( newData );
26
27                     m_tail.CompareExchange( last, newTail );
28                     return;
29                 }
30             }
31             else
32             {
33                 StampIndex newTail;
34                 newTail.SetStamp( last.GetStamp( ) + 1 );
35                 newTail.SetIndex( next.GetIndex( ) );
36
37                 m_tail.CompareExchange( last, newTail );
38             }
39         }
40     }
41 }
```

다시 돌아와서 next 가 nullptr일 때를 살펴보겠습니다.

6. next가 nullptr 이라면 Tail이 Queue의 마지막을 가리키고 있으므로 Tail이 가리키는 노드의 next가 새로운 노드를 가리키도록 변경합니다.

Push(T elem)

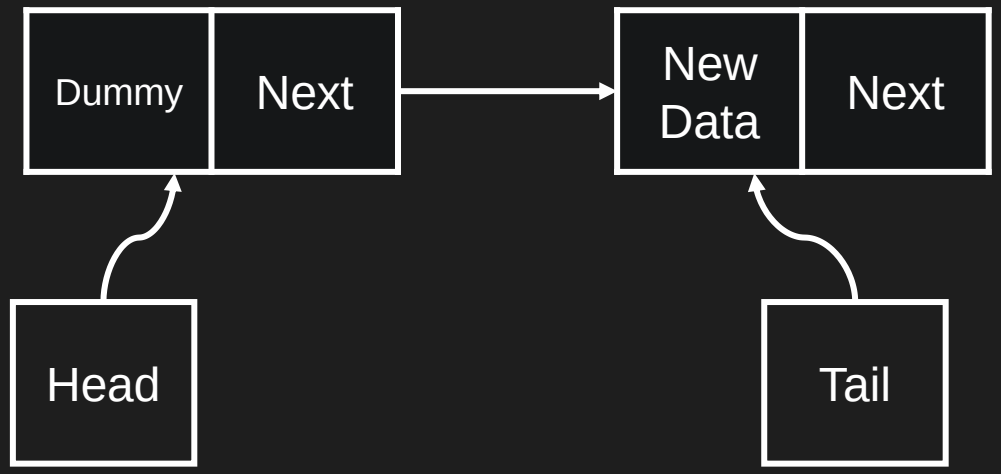
```
1 void Push( T* data )
2 {
3     unsigned int newData = LockFreeLinkPolicy::AllocLockFreeLink( );
4     LockFreeLinkPolicy::IndexToLink( newData )->m_data = data;
5
6     while ( true )
7     {
8         StampIndex last = m_tail;
9         IndexedLockFreeLink* lastLink = LockFreeLinkPolicy::IndexToLink( last.GetIndex( ) );
10        StampIndex next = lastLink->m_nextStampIndex;
11        StampIndex lastForTest = m_tail;
12        if ( last == lastForTest )
13        {
14            unsigned int nextIndex = next.GetIndex( );
15            if ( next.GetIndex( ) == 0 )
16            {
17                StampIndex newNext;
18                newNext.SetStamp( next.GetStamp( ) + 1 );
19                newNext.SetIndex( newData );
20
21                if ( lastLink->m_nextStampIndex.CompareExchange( next, newNext ) )
22                {
23                    StampIndex newTail;
24                    newTail.SetStamp( last.GetStamp( ) + 1 );
25                    newTail.SetIndex( newData );
26
27                ⑦  m_tail.CompareExchange( last, newTail );
28                    return;
29                }
30            }
31            else
32            {
33                StampIndex newTail;
34                newTail.SetStamp( last.GetStamp( ) + 1 );
35                newTail.SetIndex( next.GetIndex( ) );
36
37                m_tail.CompareExchange( last, newTail );
38            }
39        }
40    }
41 }
```

7. next가 변경되었다면 삽입이 성공한 것입니다. 마지막으로 tail을 옮겨주고 루프에서 나옵니다.

Pop( )

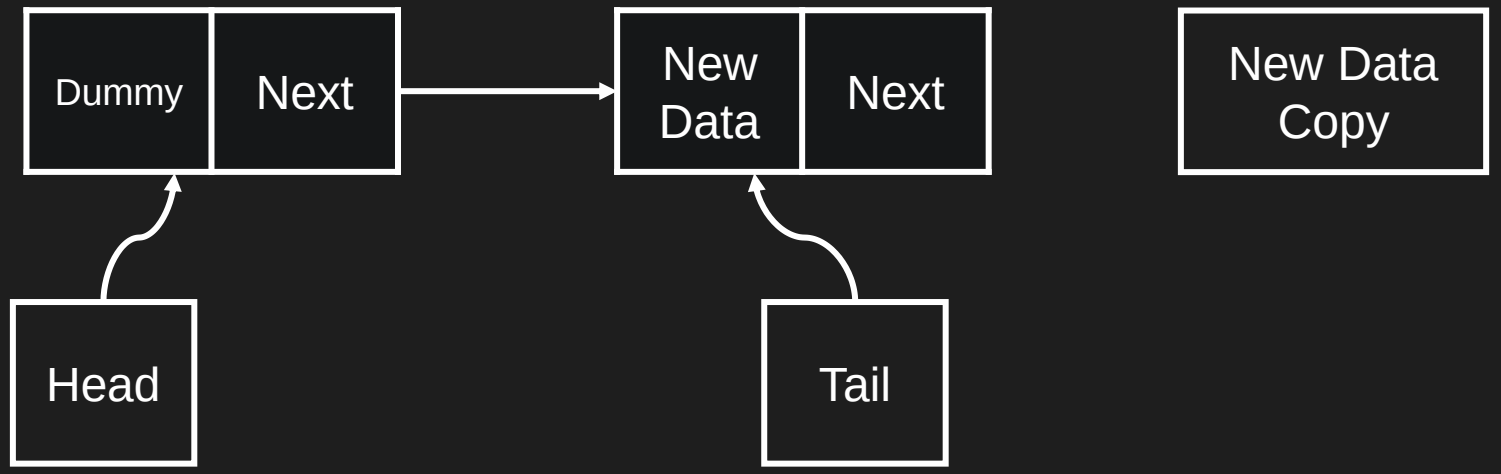
- 이제는 Pop 메서드 입니다. Pop 메서드도 마찬가지로 그림을 통해 동작을 먼저 알아보겠습니다.

- Queue의 상태는 다음과 같습니다.



Pop( )

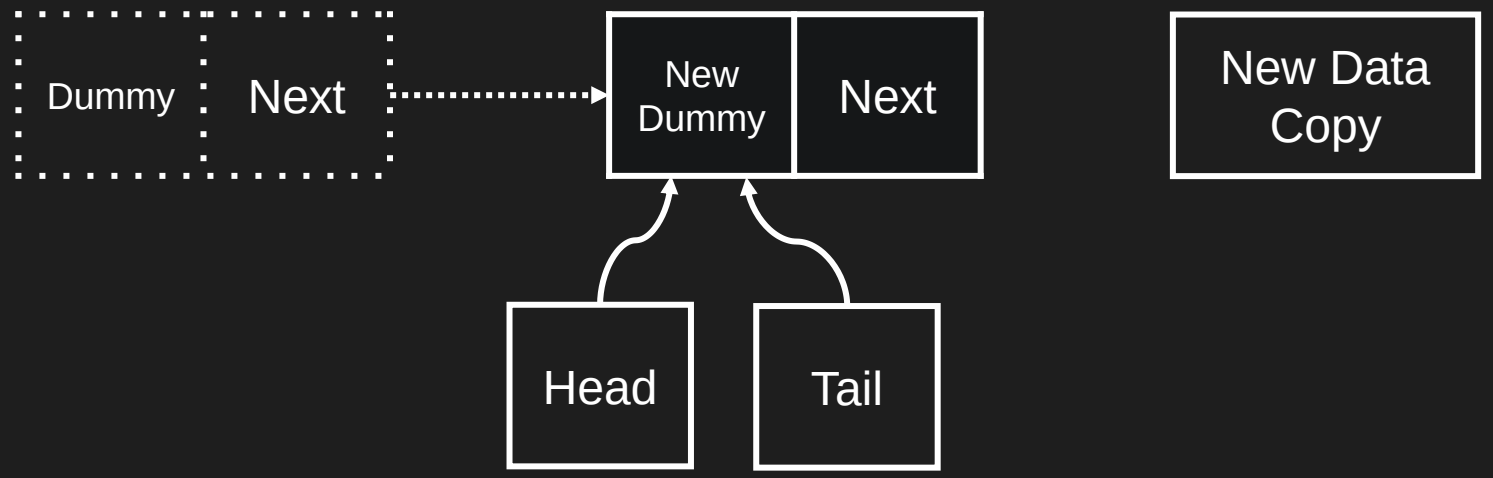
- Pop은 Push에 비해서 간단합니다. 지금 Queue는 Head와 Tail이 동일한 노드를 가리키고 있지 않기 때문에 비어 있지 않습니다. 따라서 Head의 Next가 가리키는 노드의 값을 별도의 변수에 저장해 놓습니다.



Pop( )

- 이제 Head가 Next를 가리키도록 변경합니다. 이제 이 노드는 새로운 더미 노드가 되었습니다.

- 기존 더미 노드는 삭제합니다. 그리고 보관해 놓은 값을 반환하면 됩니다.



Pop( )

```
1  T* Pop( )
2  {
3      while ( true )
4      {
5  ①       StampIndex first = m_head;
6          StampIndex last = m_tail;
7          IndexedLockFreeLink* firstLink = LockFreeLinkPolicy::IndexToLink( first.GetIndex( ) );
8          StampIndex next = firstLink->m_nextStampIndex;
9          StampIndex firstForTest = m_head;
10         if ( first == firstForTest )
11         {
12             if ( first.GetIndex( ) == last.GetIndex( ) )
13             {
14                 if ( next.GetIndex() == 0 )
15                 {
16                     return nullptr;
17                 }
18
19                 StampIndex newTail;
20                 newTail.SetStamp( last.GetStamp( ) + 1 );
21                 newTail.SetIndex( next.GetIndex( ) );
22
23                 m_tail.CompareExchange( last, newTail );
24             }
25             else
26             {
27                 IndexedLockFreeLink* nextLink = LockFreeLinkPolicy::IndexToLink( next.GetIndex( ) );
28                 auto data = reinterpret_cast<T*>( nextLink->m_data );
29
30                 StampIndex newHead;
31                 newHead.SetStamp( first.GetStamp( ) + 1 );
32                 newHead.SetIndex( next.GetIndex( ) );
33
34                 if ( m_head.CompareExchange( first, newHead ) )
35                 {
36                     LockFreeLinkPolicy::DeallocLockFreeLink( first.GetIndex( ) );
37                     return data;
38                 }
39             }
40         }
41     }
42 }
```

- Pop 메서드의 전체 코드입니다.

1. Head 가 가리키고 있는 노드 (first) Tail이 가리키고 있는 노드 (last) Head의 다음 노드(next)를 지역 변수에 저장해 놓습니다.

Pop( )

```
1  T* Pop( )
2  {
3      while ( true )
4      {
5          StampIndex first = m_head;
6          StampIndex last = m_tail;
7          IndexedLockFreeLink* firstLink = LockFreeLinkPolicy::IndexToLink( first.GetIndex( ) );
8          StampIndex next = firstLink->m_nextStampIndex;
9          StampIndex firstForTest = m_head;
10 ②       if ( first == firstForTest )
11         {
12 ③           if ( first.GetIndex( ) == last.GetIndex( ) )
13             {
14                 if ( next.GetIndex() == 0 )
15                 {
16                     return nullptr;
17                 }
18
19                 StampIndex newTail;
20                 newTail.SetStamp( last.GetStamp( ) + 1 );
21                 newTail.SetIndex( next.GetIndex( ) );
22
23                 m_tail.CompareExchange( last, newTail );
24             }
25             else
26             {
27                 IndexedLockFreeLink* nextLink = LockFreeLinkPolicy::IndexToLink( next.GetIndex( ) );
28                 auto data = reinterpret_cast<T*>( nextLink->m_data );
29
30                 StampIndex newHead;
31                 newHead.SetStamp( first.GetStamp( ) + 1 );
32                 newHead.SetIndex( next.GetIndex( ) );
33
34                 if ( m_head.CompareExchange( first, newHead ) )
35                 {
36                     LockFreeLinkPolicy::DeallocLockFreeLink( first.GetIndex( ) );
37                     return data;
38                 }
39             }
40         }
41     }
42 }
```

2. 다른 스레드에 의해서 Head가 변경 되었는지 검사하고 변경됐다면 재시도 합니다.

3. Head와 Tail이 동일한 노드를 가리키고 있는지 확인합니다. 만약 동일한 노드를 가리키고 있다면 Queue는 비어 있는 상태입니다. 우선 Queue가 비었을 때 처리를 살펴보겠습니다.

Pop( )

```
1 T* Pop( )
2 {
3     while ( true )
4     {
5         StampIndex first = m_head;
6         StampIndex last = m_tail;
7         IndexedLockFreeLink* firstLink = LockFreeLinkPolicy::IndexToLink( first.GetIndex( ) );
8         StampIndex next = firstLink->m_nextStampIndex;
9         StampIndex firstForTest = m_head;
10        if ( first == firstForTest )
11        {
12            if ( first.GetIndex( ) == last.GetIndex( ) )
13            {
14  ④             if ( next.GetIndex() == 0 )
15                {
16                    return nullptr;
17                }
18
19                StampIndex newTail;
20                newTail.SetStamp( last.GetStamp( ) + 1 );
21                newTail.SetIndex( next.GetIndex( ) );
22
23                m_tail.CompareExchange( last, newTail );
24            }
25            else
26            {
27                IndexedLockFreeLink* nextLink = LockFreeLinkPolicy::IndexToLink( next.GetIndex( ) );
28                auto data = reinterpret_cast<T*>( nextLink->m_data );
29
30                StampIndex newHead;
31                newHead.SetStamp( first.GetStamp( ) + 1 );
32                newHead.SetIndex( next.GetIndex( ) );
33
34                if ( m_head.CompareExchange( first, newHead ) )
35                {
36                    LockFreeLinkPolicy::DeallocLockFreeLink( first.GetIndex( ) );
37                    return data;
38                }
39            }
40        }
41    }
42 }
```

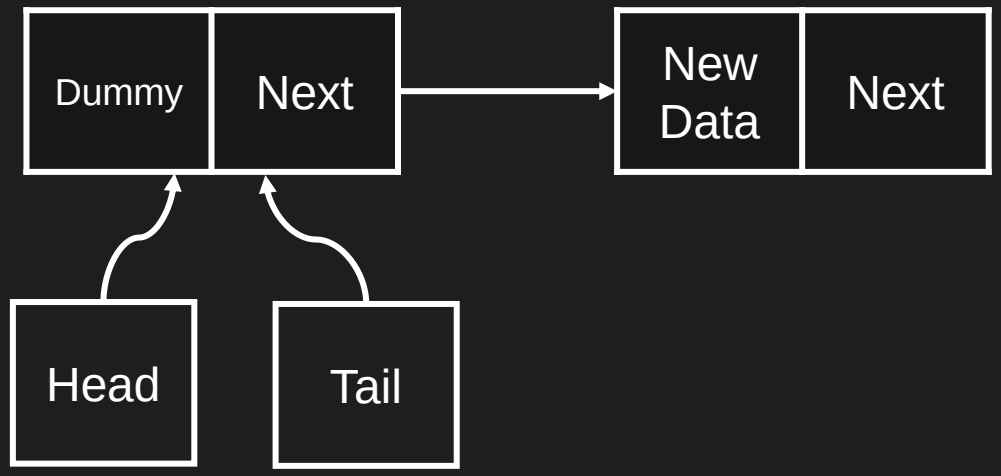
4. next가 nullptr이라면 Queue는 정말로 비어 있습니다. 비어 있는 Queue에 대한 Pop연산 처리를 해줍니다. 이 코드는 nullptr를 반환하였습니다.

그런데 next가 nullptr이 아닌 경우는 Queue가 어떤 상태일까요.

Pop( )

- Push 메서드에서 보았던 상태입니다. 새로운 노드가 삽입 됐지만 Tail이 새로운 노드를 가리키기 전에 다른 스레드에 CPU 자원을 뺏겼습니다.

- 이 때 해야 할 작업은 정해져 있습니다. 스스로 Tail을 뒤로 이동시킵니다.




```
1 StampIndex newTail;
2 newTail.SetStamp( last.GetStamp( ) + 1 );
3 newTail.SetIndex( next.GetIndex( ) );
4
5 m_tail.CompareExchange( last, newTail );
```

Pop( )

```
1 T* Pop( )
2 {
3     while ( true )
4     {
5         StampIndex first = m_head;
6         StampIndex last = m_tail;
7         IndexedLockFreeLink* firstLink = LockFreeLinkPolicy::IndexToLink( first.GetIndex( ) );
8         StampIndex next = firstLink->m_nextStampIndex;
9         StampIndex firstForTest = m_head;
10        if ( first == firstForTest )
11        {
12            if ( first.GetIndex( ) == last.GetIndex( ) )
13            {
14                if ( next.GetIndex() == 0 )
15                {
16                    return nullptr;
17                }
18
19                StampIndex newTail;
20                newTail.SetStamp( last.GetStamp( ) + 1 );
21                newTail.SetIndex( next.GetIndex( ) );
22
23                m_tail.CompareExchange( last, newTail );
24            }
25            else
26
27 ⑤             IndexedLockFreeLink* nextLink = LockFreeLinkPolicy::IndexToLink( next.GetIndex( ) );
28                auto data = reinterpret_cast<T*>( nextLink->m_data );
29
30                StampIndex newHead;
31                newHead.SetStamp( first.GetStamp( ) + 1 );
32                newHead.SetIndex( next.GetIndex( ) );
33
34 ⑥             if ( m_head.CompareExchange( first, newHead ) )
35                {
36                    LockFreeLinkPolicy::DeallocLockFreeLink( first.GetIndex( ) );
37                    return data;
38                }
39            }
40        }
41    }
42 }
```

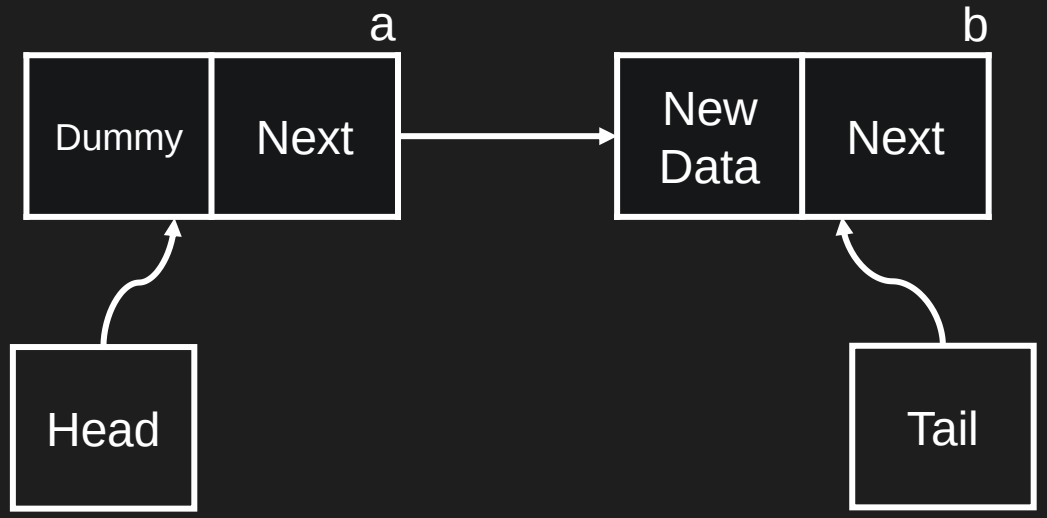
5. Queue가 비어 있지 않다면 next의 데이터를 지역변수에 복사해 놓습니다.

6. Head를 next로 이동시키고 기존의 Head의 노드를 삭제한 다음 데이터를 반환하면 끝입니다.

ABA 문제

- Push 메서드를 살펴보면서 Lock-free 알고리즘에서 발생할 수 있는 어떤 문제에 대해 언급하였는데 여기서 한번 살펴보도록 하겠습니다.

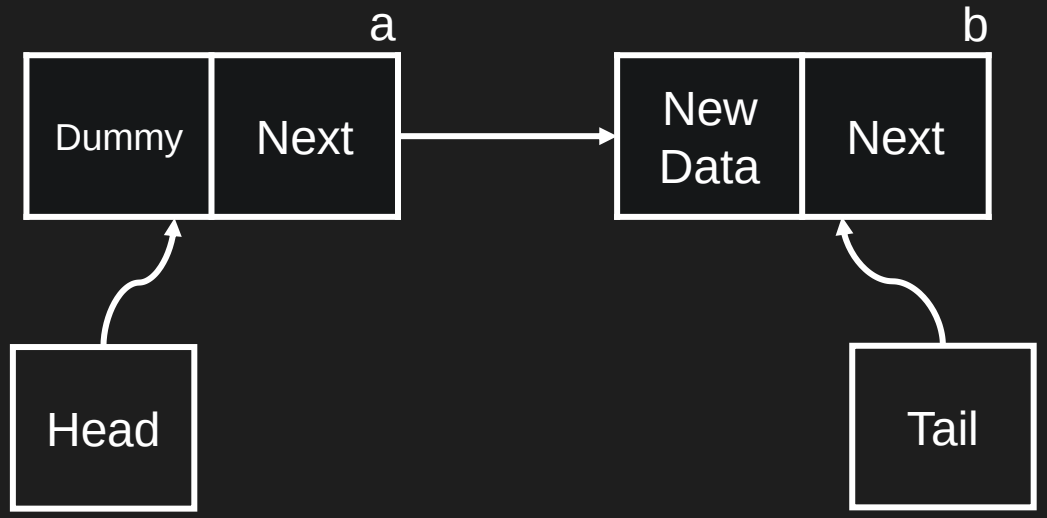
- 다음과 같은 Queue를 생각해 보겠습니다. 각 노드의 주소를 a, b라 하겠습니다.



ABA 문제

- 스레드1 에서 Pop메소드를 실행하려고 하는데 Head를 전진시키기 전에 다른 스레드로 CPU 선점권을 뺏겼다고 가정하겠습니다

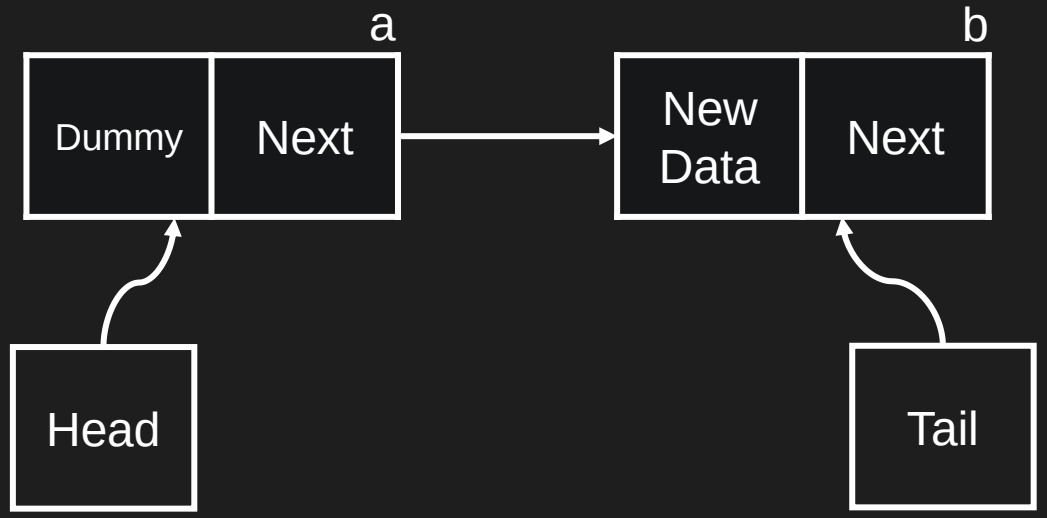
- 이 경우 스레드1의 로컬 변수에 Head의 주소 a와 Next의 주소 b를 저장해 놓은 상태입니다.



ABA 문제

- 이제 다른 스레드들이 Push와 Pop을 실행합니다.

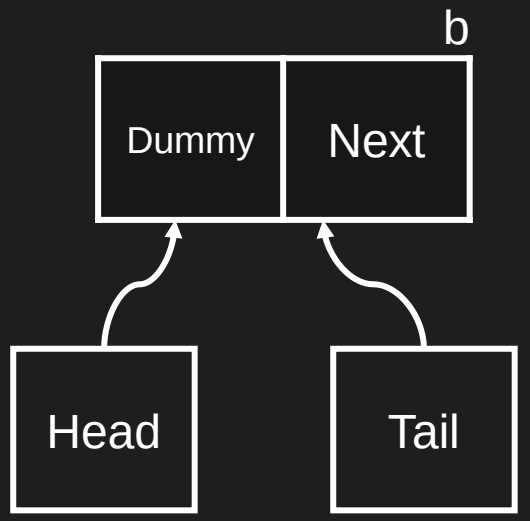
- 여러 차례의 메모리 반납과 할당이 이뤄지는데 문제는 이 메모리가 재활용 된다는 것입니다. 다음과 같은 순서로 메소드가 실행되는 경우를 봅시다.



ABA 문제

- Pop() -> Push() -> Pop()

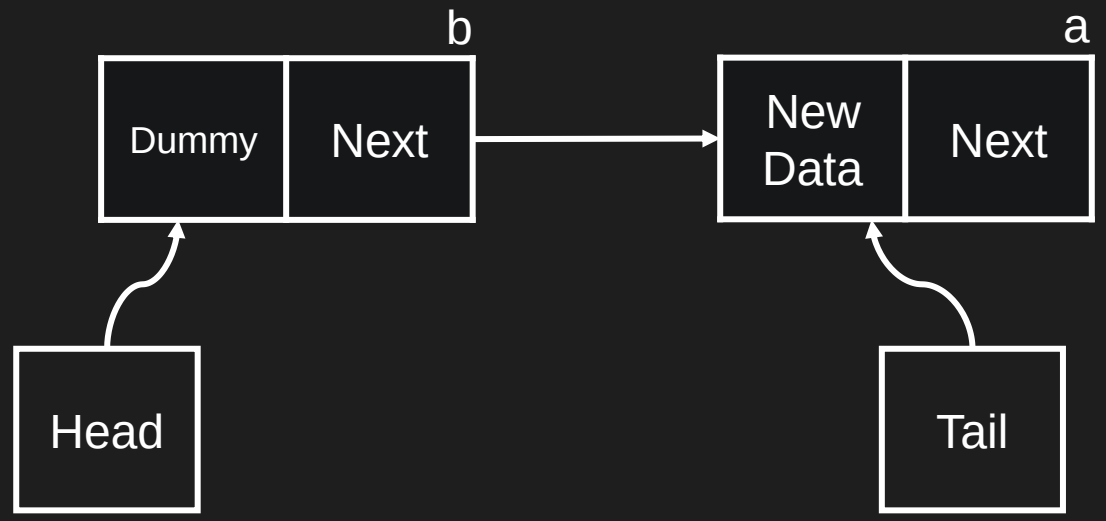
- 먼저 Pop() 메소드를 통해 주소 a의 메모리 영역을 해제합니다.



ABA 문제

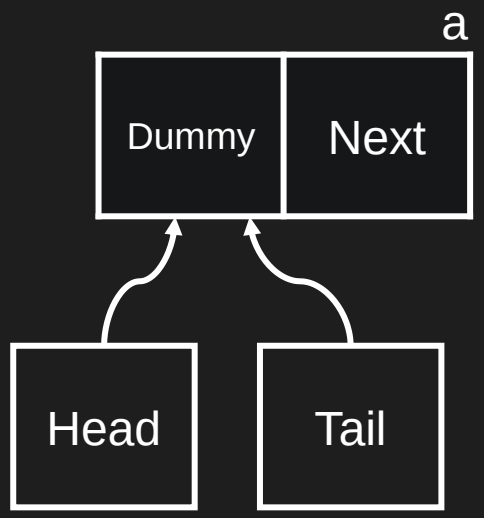
- Pop() -> Push() -> Pop()

- 이제 다시 Push가 실행되는데 메모리가 재활용되어 이전의 a 주소가 다시 나왔다고 생각해봅시다.



ABA 문제

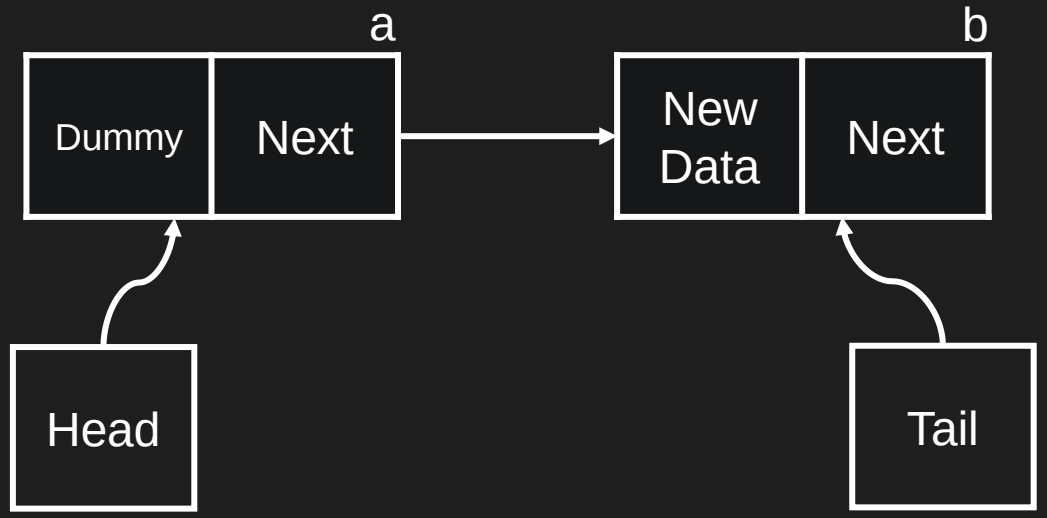
- Pop() -> Push() -> Pop()
- 그리고 또 다시 Pop이 실행되면 최종적인 Queue는 아래와 같이 됩니다.



ABA 문제

- 이제 길고 긴 시간이 지나 스레드1이 다시 실행 되었다고 생각해봅시다.

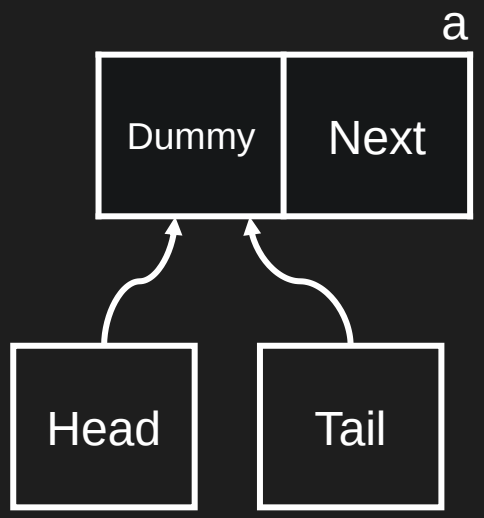
- 스레드1이 생각하고 있는 Queue의 모양은 다음과 같습니다. a 주소 노드의 다음으로 b 주소의 노드가 연결되어 있죠. 하지만 실제 Queue의 상태는…



ABA 문제

- a 주소의 노드 뒤에 아무것도 없습니다.

- 문제는 이 때 스레드1의 CAS(Head, a, b)가 성공합니다. 그리고 Head는 스레드1의 로컬 Next 변수에 저장한 b를 가리키게 됩니다. 게다가 b는 해제된 메모리 영역입니다.



ABA 문제

- 메모리의 재활용으로 인해서 CAS연산의 안전장치(현재 값과 예상 값이 같을 때만 값을 변경)가 무효화 됐습니다.
- 이런 문제를 ABA 문제라고 합니다.
- ABA 문제는 모든 CAS와 new, delete를 사용할 경우 모든 Lock-free 알고리즘에서 발생할 수 있습니다.
- 그럼 이걸 어떻게 회피해야 할까? 여기서는 2가지 방법을 소개하겠습니다.

ABA 문제

- 첫번째 방법은 메모리가 곧바로 재활용이 되지 않도록 하는 것 입니다.

- 이전 예제에서 스레드1은 Head를 로컬 변수에 저장하고 있었습니다. 만약 이 Head 주소의 메모리가 레퍼런스 카운트에 의해서 관리되고 있다면 다른 스레드에서 Pop을 하더라도 실제로 메모리가 반납되지 않아서 재활용되지 않습니다.

- 그래서 GC가 사용되는 환경(C#, Python, etc…)에서는 ABA 문제는 발생하지 않습니다.

ABA 문제

- 두번째 방법은 주소를 확장하는 것입니다.
  Tag 혹은 Stamp라 불리는 추가적인 비트를 더해서 매 CAS시도 시 이 값을 다르게 설정합니다.

```
1 StampIndex newTail;
2 newTail.SetStamp( last.GetStamp( ) + 1 );
3 newTail.SetIndex( next.GetIndex( ) );
4
5 m_tail.CompareExchange( last, newTail );
```

- 메모리가 재활용 된다고 해도 추가된 비트가 매번 다르기 때문에 ABA문제를 회피 할 수 있습니다.

ABA 문제

- StampIndex 클래스는 ABA 문제를 해결하기 위해 4byte의 Stamp와 4byte의 메모리 풀에 대한 Index를 하나로 묶어 사용하고 있습니다.

```
1 struct alignas(8) StampIndex
2 {
3 public:
4     void Set( unsigned int index, unsigned int stamp )
5     {
6         m_stampIndex = ( static_cast<unsigned long long>( stamp ) << IndexBits ) | index;
7     }
8
9     // 일부 함수 생략
10 private:
11     static constexpr unsigned int IndexBits = std::numeric_limits<unsigned int>::digits;
12     unsigned long long m_stampIndex = 0;
13 };
14
```

성능 측정

- 구현도 했으니 Lock 버전의 Queue와 성능을 비교해 보겠습니다.
  Lock 버전 Queue는 다음과 같이 구현하였습니다.

```cpp
template <typename T>
class LockQueue
{
public:
    void Push( T data )
    {
        std::lock_guard lk( m_lock );
        m_queue.push( data );
    }

    std::optional<T> Pop( )
    {
        std::lock_guard lk( m_lock );
        if ( m_queue.empty( ) == false )
        {
            T data = m_queue.front( );
            m_queue.pop( );
            return data;
        }

        return {};
    }

private:
    std::mutex m_lock;
    std::queue<T> m_queue;
};
```

성능 측정

- 테스트 코드는 다음과 같습니다.
- 10000번 이하까지는 Push만 하다가 이후에는 임의로 Push 혹은 Pop을 호출합니다.
- LoopCount = 64000000, ThreadCount = 8

```
1 auto job1 = [&lQueue, LoopCount, ThreadCount]( )
2 {
3     for ( int i = 1; i <= LoopCount / ThreadCount; ++i )
4     {
5         if ( rand( ) % 2 == 0 || i < ( 10000 / ThreadCount ) )
6         {
7             lQueue.Push( i );
8         }
9         else
10        {
11            lQueue.Pop( );
12        }
13    }
14 };
```

성능 측정

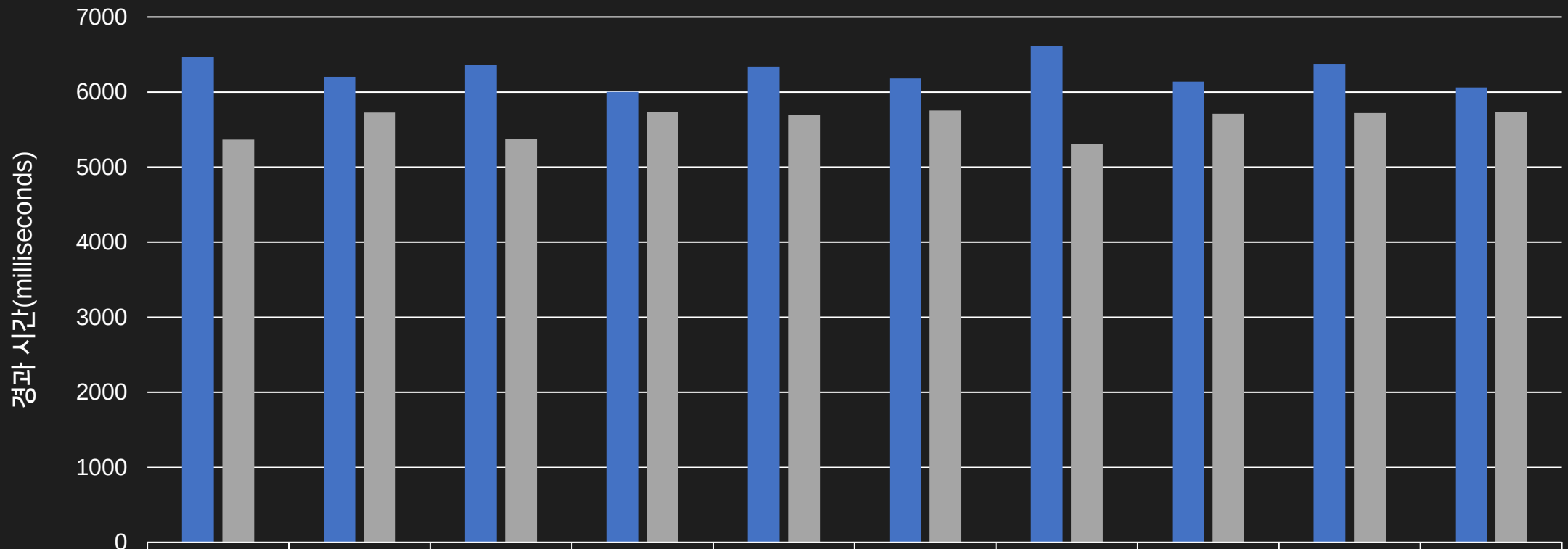
- 성능 측정에 사용한 PC 사양입니다.

CPU : Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz 2.30 GHz
(2코어 4스레드)

RAM : 8 GB

Visual Studio 2017 64 bit 빌드

성능 측정



	1 회차	2 회차	3 회차	4 회차	5 회차	6 회차	7 회차	8 회차	9 회차	10 회차
Lock	6473	6204	6361	6001	6340	6182	6610	6139	6376	6062
Lock-Free	5369	5728	5374	5735	5694	5755	5310	5711	5720	5731

성능 측정

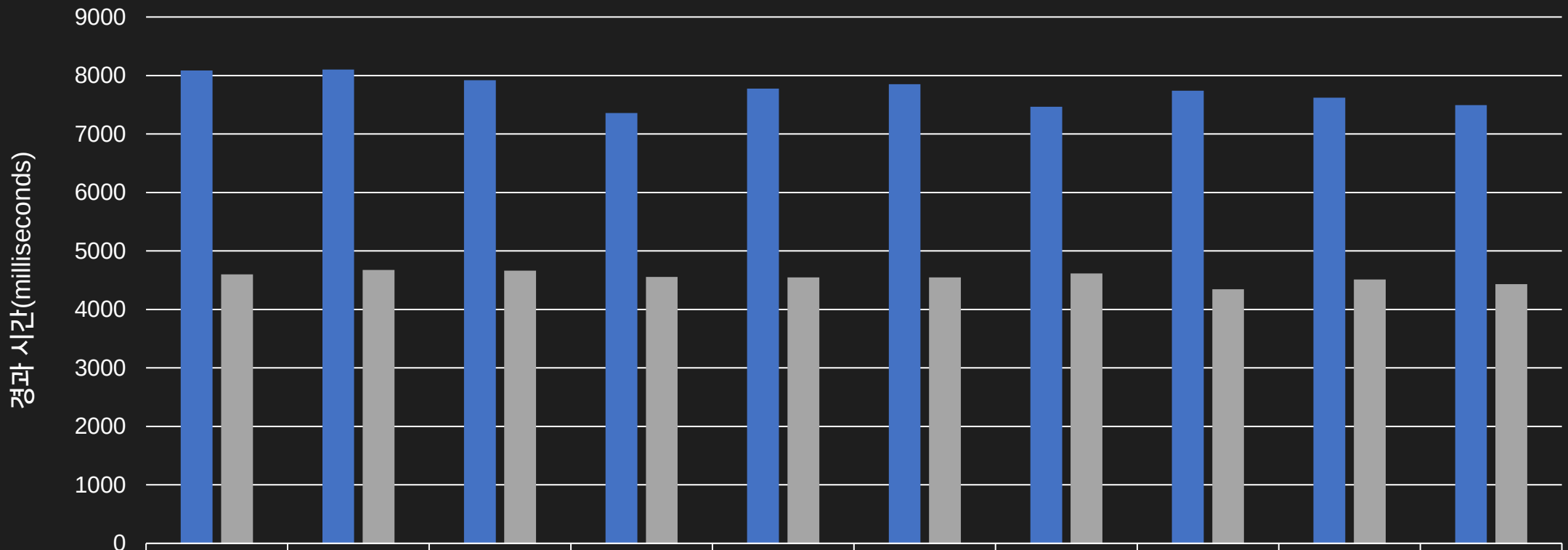
- 다른 PC에서 측정한 자료도 보시겠습니다.

CPU : Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz 3.60 GHz
(4코어 8스레드)

RAM : 16 GB

Visual Studio 2017 64 bit 빌드

성능 측정



	1 회차	2 회차	3 회차	4 회차	5 회차	6 회차	7 회차	8 회차	9 회차	10 회차
Lock	8086	8104	7920	7361	7776	7853	7466	7740	7621	7496
Lock-Free	4599	4676	4665	4555	4547	4547	4617	4344	4512	4431

성능 측정

- Lock-Free Queue가 Lock 버전의 Queue 보다 빠른 것을 확인할 수 있습니다. 그런데 잠시 이것을 보시도록 하겠습니다.

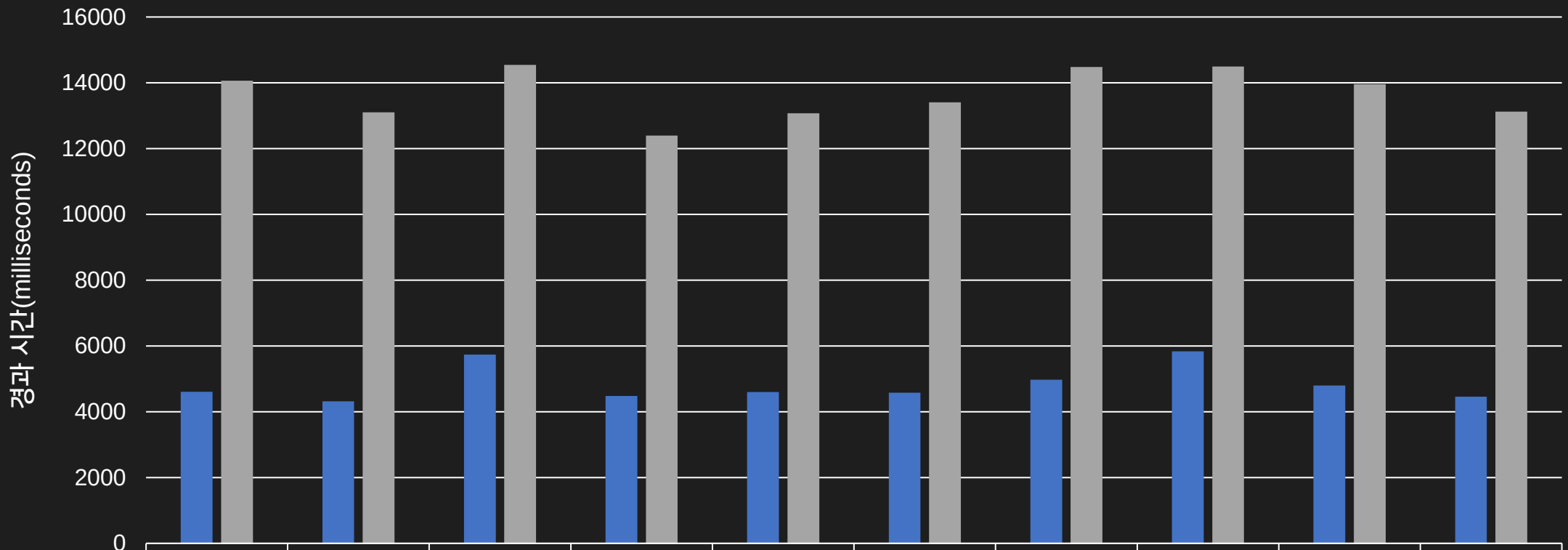
- 동일한 코드의 32 bit 빌드 버전입니다.

CPU : Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz 3.60 GHz
(4코어 8스레드)

RAM : 16 GB

Visual Studio 2017 32 bit 빌드

성능 측정

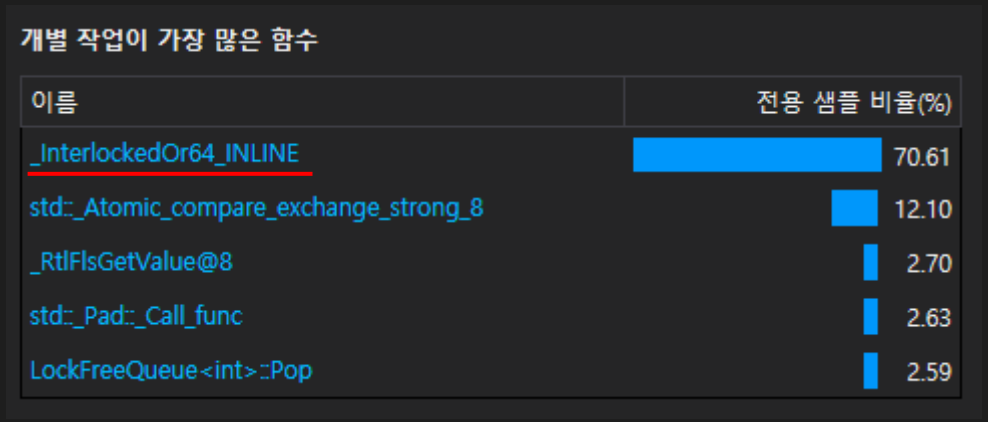


	1 회차	2 회차	3 회차	4 회차	5 회차	6 회차	7 회차	8 회차	9 회차	10 회차
Lock	4607	4317	5736	4479	4598	4582	4975	5835	4799	4462
Lock-Free	14068	13103	14546	12394	13079	13408	14483	14495	13958	13128

성능 측정

- 64bit와는 달리 lock-free의 성능이 급감하였습니다.

- 어디가 문제 있는지 프로파일링 해보면…



성능 측정

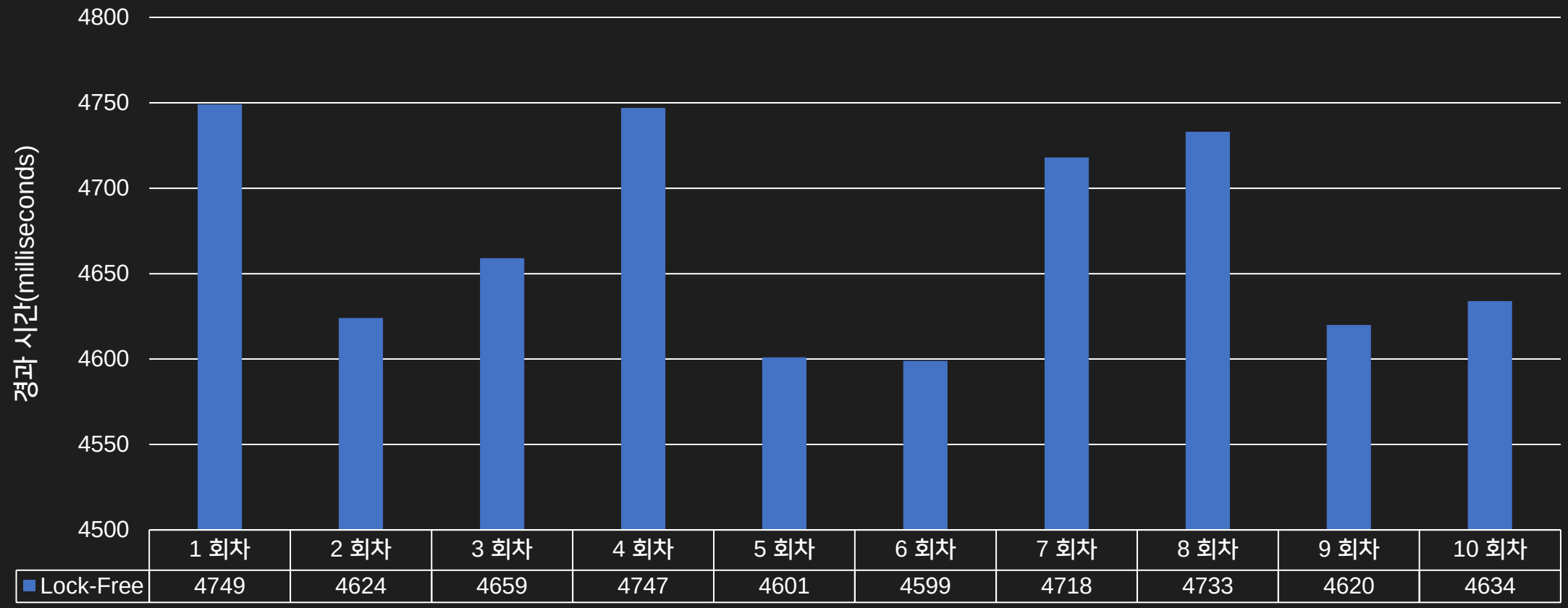
- _InterlockedOr64_INLINE 이라는 함수가 대부분의 시간을 차지하고 있습니다.

- 이 함수는 std::atomic_load 에서 호출되는 함수입니다. 즉 다음과 같은 코드가 성능을 잡아 먹는 원인이 됩니다.

```
1 StampIndex last = m_tail;
2
3 // StampIndex의 대입연산자는 다음과 같이 오버로딩돼 있습니다.
4 StampIndex& operator=( const StampIndex& other )
5 {
6     if ( this != &other )
7     {
8         m_stampIndex = std::atomic_load( reinterpret_cast<const std::atomic_llong*>( &other ) );
9     }
10
11     return *this;
12 }
```

성능 측정

- std::atomic_load 를 하지 않는다면 성능은 다음과 같이 상향 됩니다.



	1 회차	2 회차	3 회차	4 회차	5 회차	6 회차	7 회차	8 회차	9 회차	10 회차
Lock-Free	4749	4624	4659	4747	4601	4599	4718	4733	4620	4634

성능 측정

- 그런데 32bit 프로그램에서 8byte 자료형을 읽고 쓸 때는 다음과 같이 2번에 걸쳐서 메모리에 접근합니다.

```
 1 int main()
 2 {
 3 00D81002  in          al,dx
 4 00D81003  sub         esp,10h
 5     volatile long long a = 1;
 6 00D81006  mov         dword ptr [ebp-10h],1
 7 00D8100D  mov         dword ptr [ebp-0Ch],0
 8     volatile long long b = 2;
 9 00D81014  mov         dword ptr [ebp-8],2
10 00D8101B  mov         dword ptr [ebp-4],0
11
12     a = b;
13 00D81022  mov         eax,dword ptr [b]
14 00D81025  mov         dword ptr [a],eax
15 00D81028  mov         eax,dword ptr [ebp-4]
16 00D8102B  mov         dword ptr [ebp-0Ch],eax
17 }
```

- 만약 14번 째 줄 실행 후 다른 스레드가 해당 변수에 값을 쓰게 된다면 8byte의 절반은 업데이트된 값이 저장됩니다.

성능 측정

- 그러므로 std::atomic_load 를 제거할 수 는 없습니다. 따라서 32bit 에서 성능을 끌어 올리기 위해서는 다른 방법을 생각해봐야 할 것 같습니다.

- StampIndex를 16bit Stamp, 16bit 메모리 풀 인덱스로 구성하는 방법도 있겠습니다. 이렇게 구성하면 ABA 문제가 발생하려면 점유권을 빼앗긴 동안 65,536번 Push와 Pop이 발생해야 합니다.

- 아니면 Stamp를 사용하지 않고 레퍼런스 카운터를 통해 참조하고 있다면 메모리를 재활용하지 못하게 만드는 방법도 있겠습니다.

마치며...

- 일단 제가 준비한 내용은 여기까지 입니다.

- 그래프를 자세히 보셨던 분이라면 64bit에서 lock 버전 Queue의 성능이 떨어진 것에 눈치 채셨을 텐데 주제에서 벗어난 내용이라 다루지는 않았지만 std::mutex의 구현 차이로 인해 성능의 차이가 있더군요. 정말 성능이 중요한 상황에서는 고려할 만한 상황인 것 같습니다.

더 자세한 코드를 보고 싶으시다면...

- https://github.com/xtozero/LockFreeDataStructure/tree/master/LockFreeDataStructure/Source

참고자료

- UE4 LockFreeList.h LockFreeList.cpp
- Boost lockfree/queue.hpp
- https://preshing.com/20130618/atomic-vs-non-atomic-operations/