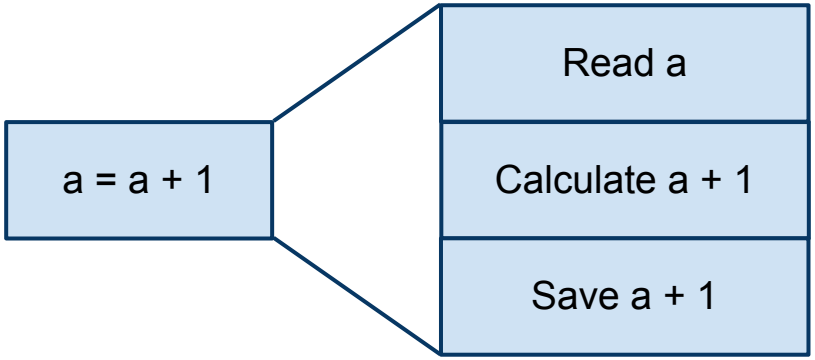
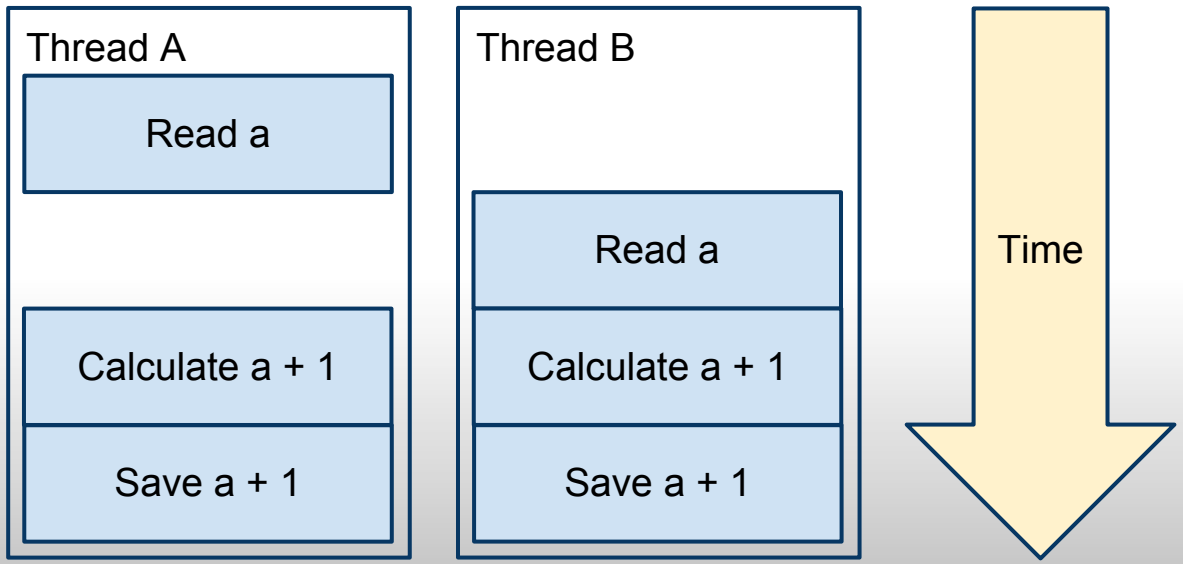
Lock-Free Algorithms

Present by PanPan

Multi-Thread Programming Problem

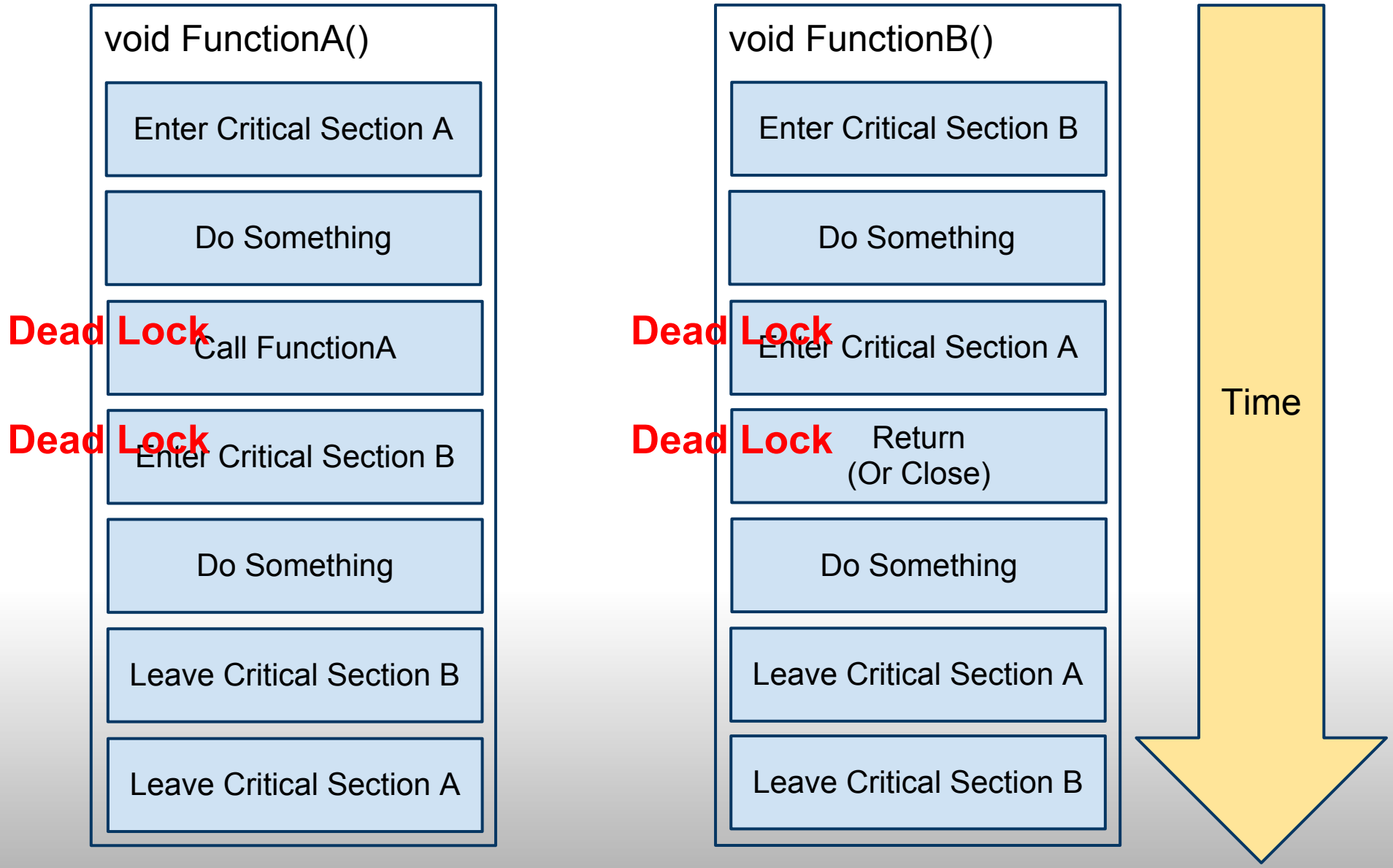




Multi-Thread Programming Solution

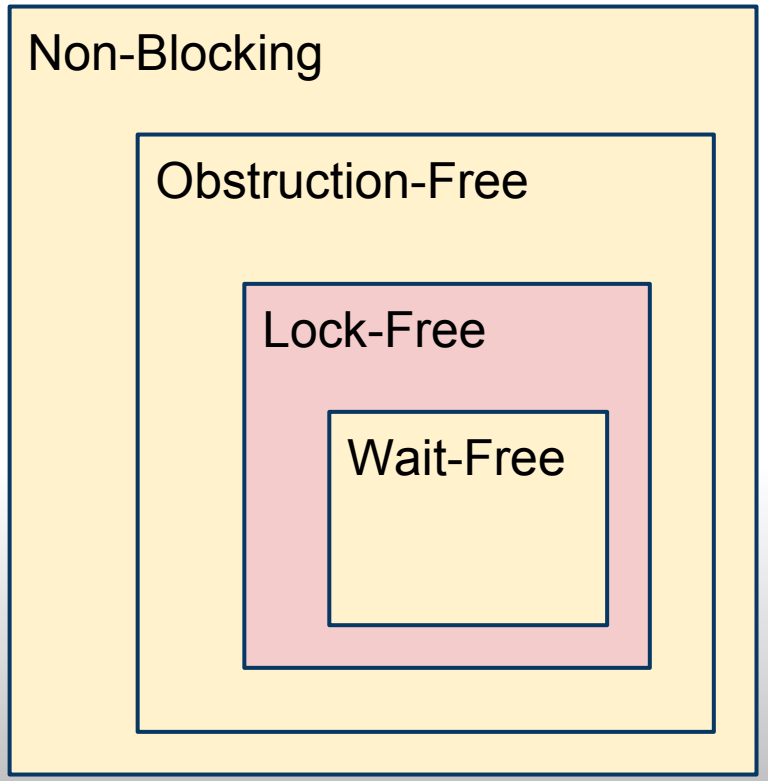
- Blocking
  - Mutex
  - Critical Section
- Non-Blocking
  - Wait-Free
  - Lock-Free
  - Obstruction-Free
- Local
  - No Shared Data

Dead Lock


void FunctionA()
Enter Critical Section A
Do Something
Dead Lock
Call FunctionA
Dead Lock
Enter Critical Section B
Do Something
Leave Critical Section B
Leave Critical Section A
void FunctionB()
Enter Critical Section B
Do Something
Dead Lock
Enter Critical Section A
Dead Lock
Return
(Or Close)
Do Something
Leave Critical Section A
Leave Critical Section B
Time

What is Lock-Free?

A lock-free data structure is one that doesn't use any mutex locks. The implication is that multiple threads can access the data structure concurrently without race conditions or data corruption, even though there are no locks — people would give you funny looks if you suggested that std::list was a lock-free data structure, even though it is unlikely that there are any locks used in the implementation. [1]



[1] http://www.justsoftwaresolutions.co.uk/threading/non_blocking_lock_free_and_wait_free.html

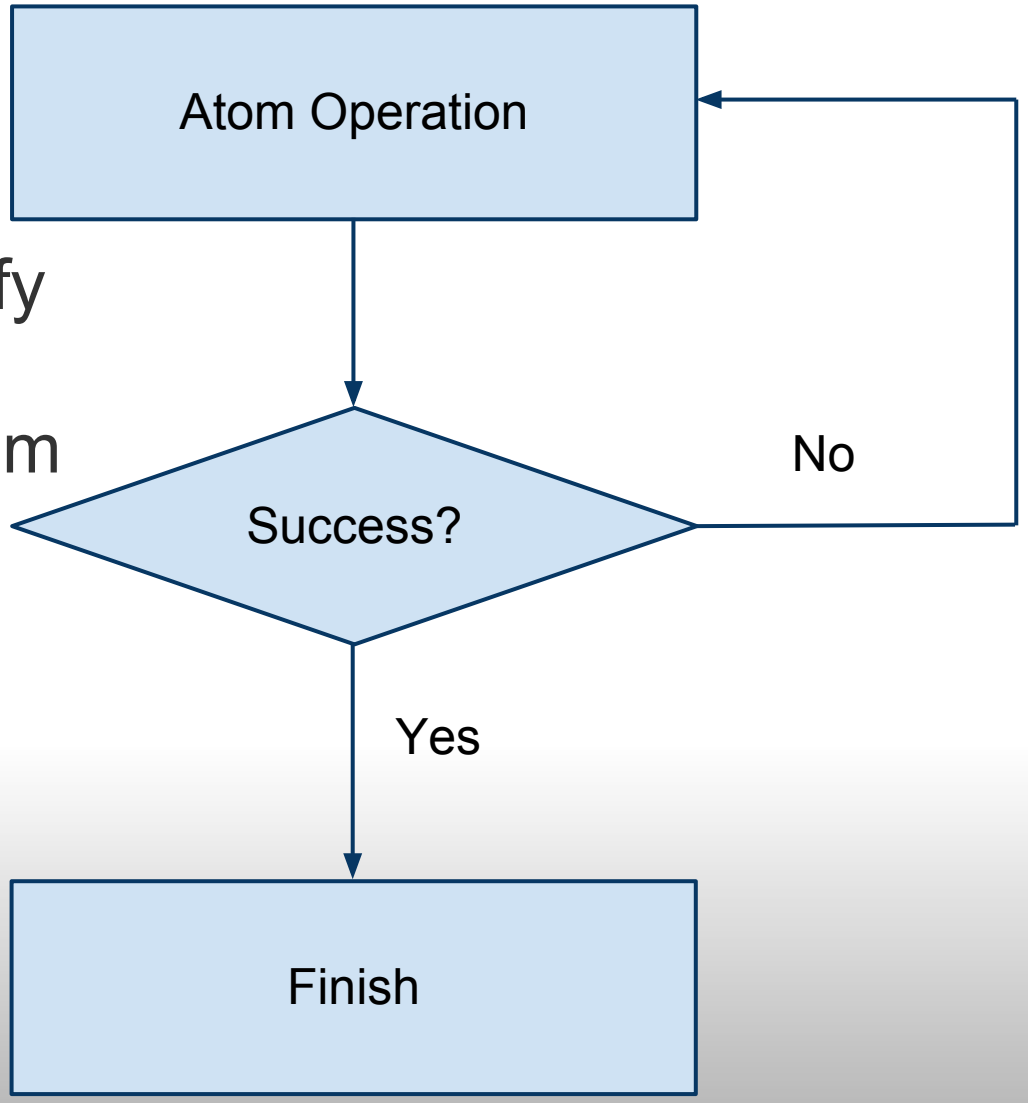
Advantages and Disadvantages

- No "Lock" (Also No Dead Lock)
- Better Performance

- Hard to Implement

Main Concept

- Use Atom Operation to Modify Critical Data
- Enter a Spin State When Atom Operation Fail



Compare and Swap

```
bool CAS(uint32* pointer, uint32 old_value, uint32 new_value)
{
    if (*pointer == old_value)
    {
        *pointer = new_value;
        return true;
    }
    return false;
}
```

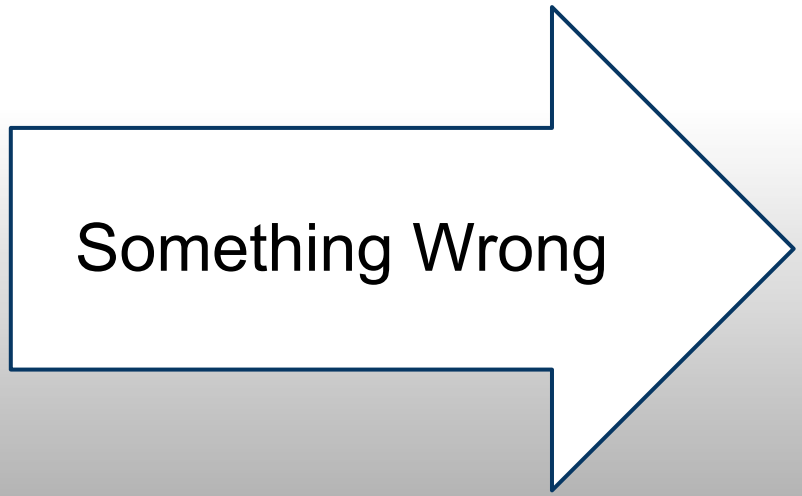
InterlockedCompareExchange - Windows API

Other Atom Operation

- CAS2 - Compare and Swap 2
- CASN - Compare and Swap N
- DCAS - Double Compare Swap
- MCAS - Multiword Compare and Swap
- LL/SC - Load Linked / Store Conditional

Example - Stack

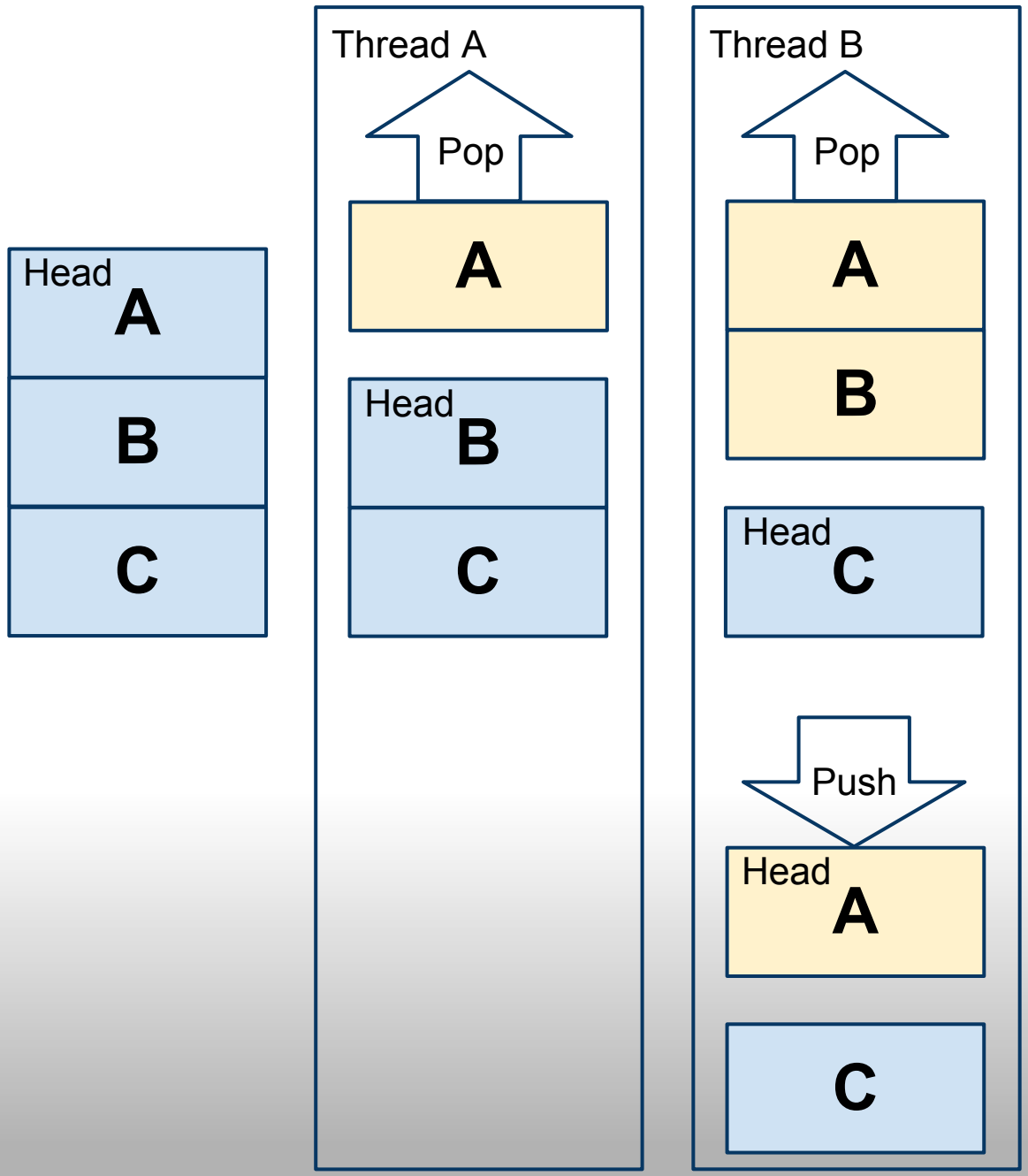
```
struct Node
{
    volatile Node* next;
};

class Stack
{
    volatile Node* head;
public:
    void Push(Node* node);
    void Pop();
    Stack() : head(0) {}
};
```




```
void Stack::Push(Node* node)
{
    while (true)
    {
        node->next = head;
        if (CAS(&head, node->next, node))
            break;
    }
}
```

```
void Stack::Pop()
{
    while (true)
    {
        node* old_head = head;
        if (head == 0)
            break;
        node* next_node = old_head->next;
        if (CAS(&head, old_head, next_node))
            break;
    }
}
```

ABA Problem



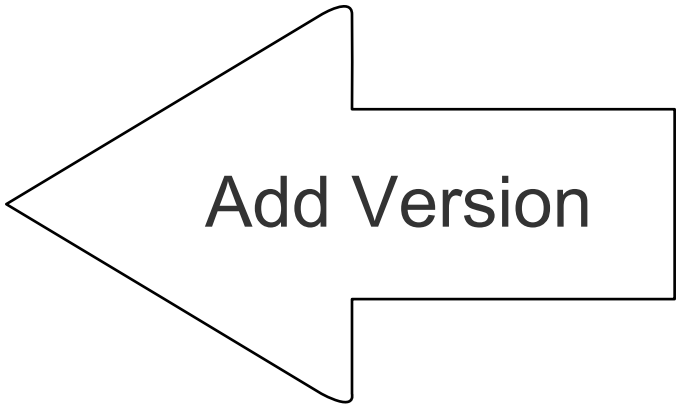

```
node* old_head = head;
if (head == 0)
    break;
node* next_node = old_head->next;
if (CAS(&head, old_head, next_node))
    break;
```

Solute ABA Problem

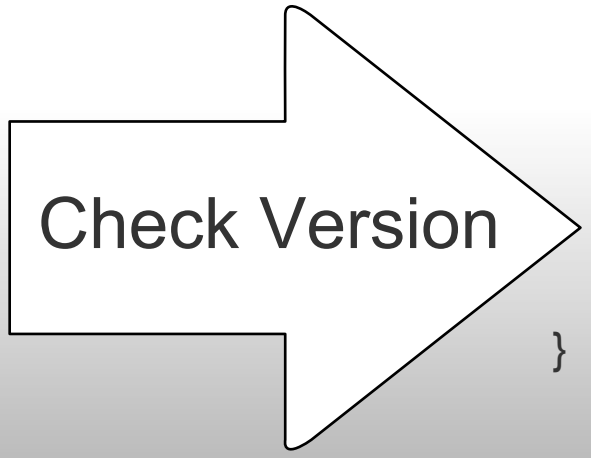
- LL/SC
  - It would return false when target have done a "write" action
- CAS2
  - Add a versioned value

Example - Stack(fix Pop)

```
class Stack
{
    volatile Node* head;
    volatile int version;
public:
    void Push(Node* node);
    void Pop();
    Stack() : head(0), version(0) {}
};
```




```
void Stack::Pop()
{
    while (true)
    {
        node* old_head = head;
        int old_version = version;
        if (head == 0)
            break;
        node* next_node = old_head->next;
        if (CAS2(&head, old_head, old_version, next_node, old_version + 1))
            break;
    }
}
```



Summary

- Lock-Free has Better Performance Than Blocking
- Lock-Free has no "Lock"
- Lock-Free is Hard To Implement
- Lock-Free's Spirit - Atom Operation
- Lock-Free's Trap - ABA Problem

Notes and References

- Toby Jones - Lock-Free Algorithms(Game Programming Gems 6)
- Jean-Francois Dube - Efficient and Scalable Multi-Core Programming(Game Programming Gems 8)
- Zhou Wei Ming - Multi-core Computing and Programming

Any Questions?

Thanks for your attention

By PanPan

2010-06-29