Lock free Algorithm for Multi-core architecture

SDY Corporation
Hiromasa Kanda

Contents

1.Introduction

- Background needed Multi-Thread
- Problem of Multi-Thread program
- What's Lock free?

2.Lock free Algorithm

- Atomic operation
- Lock-free queue
- Lock-free hash-map

3.Performance

4.Summary

1.Introduction

Background needed Multi-Thread

Multi-core and SMT(HT)

- Limited CMOS scaling
- Manage memory access and CPU clock

What is needed in application?

- Parallelization → **Multi-thread**

Amdahl's law

- The speedup of a program using multiple processors in parallel computing is limited

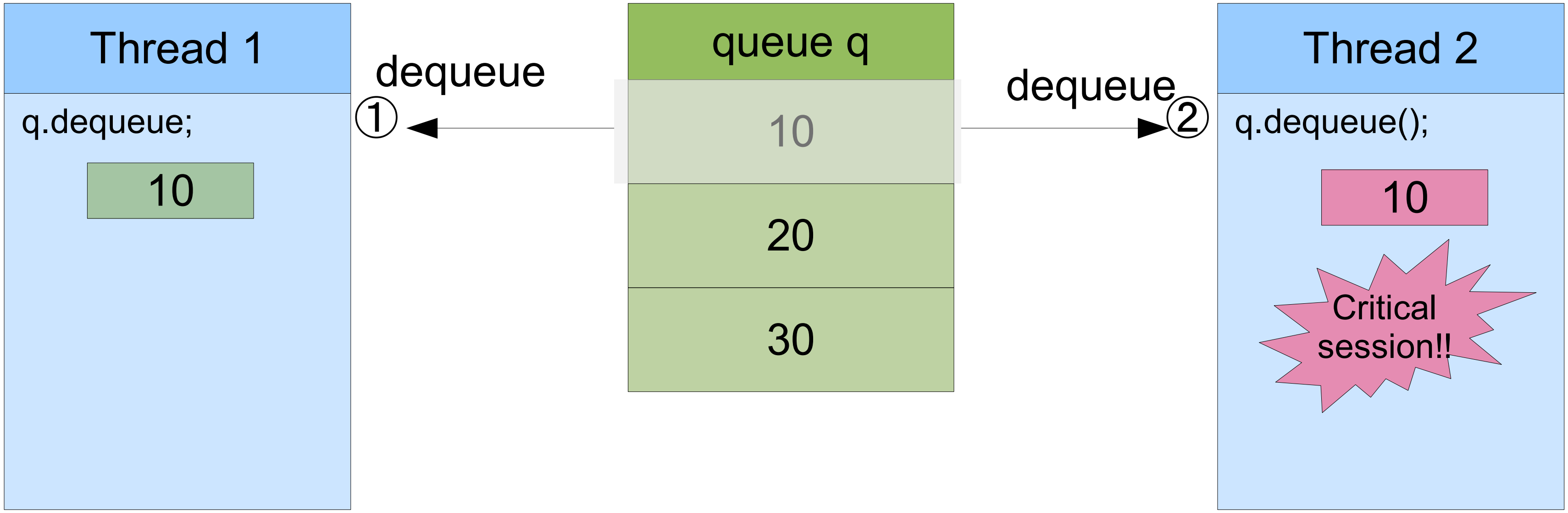
Problem of Multi-thread

- Scheduling
- Shared resource

1.Introduction

Problem of Multi-Thread program 1/2

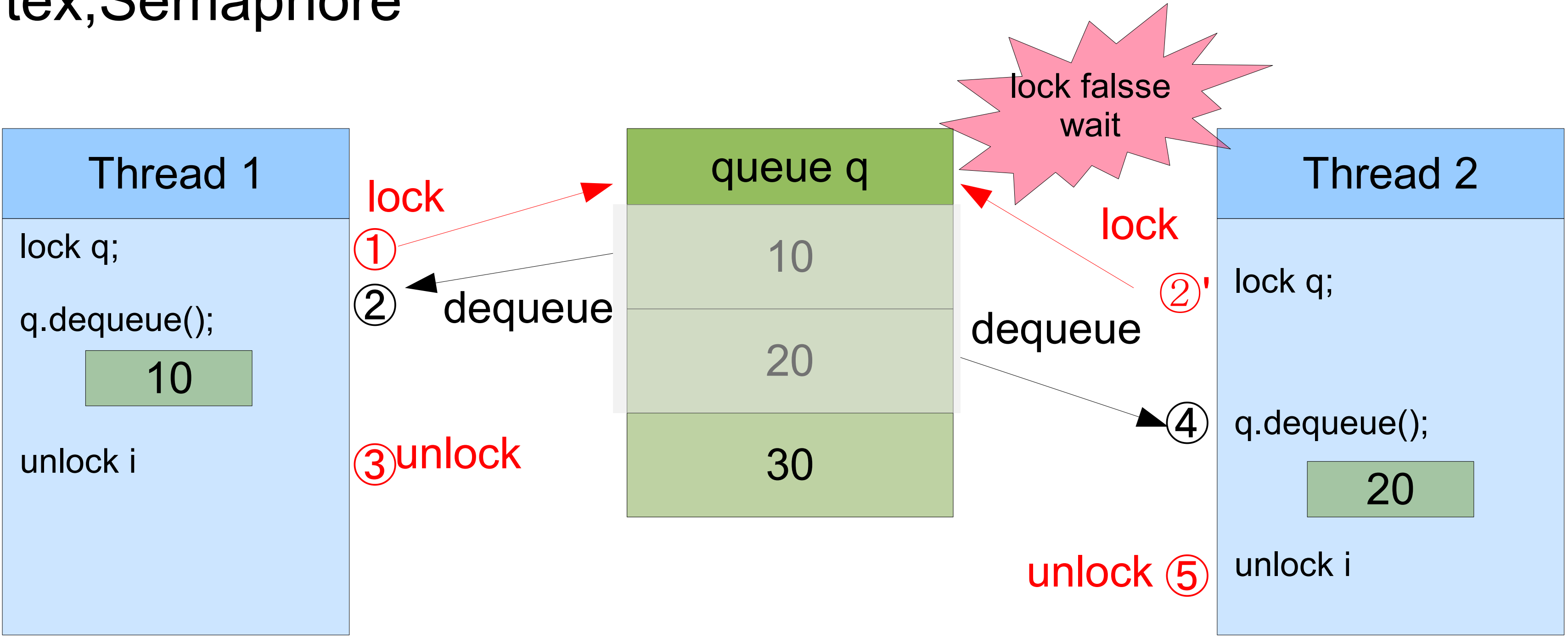
There was resources problem that share it when concurrent access multi-thread.



1.Introduction

Problem of Multi-Thread program 2/2

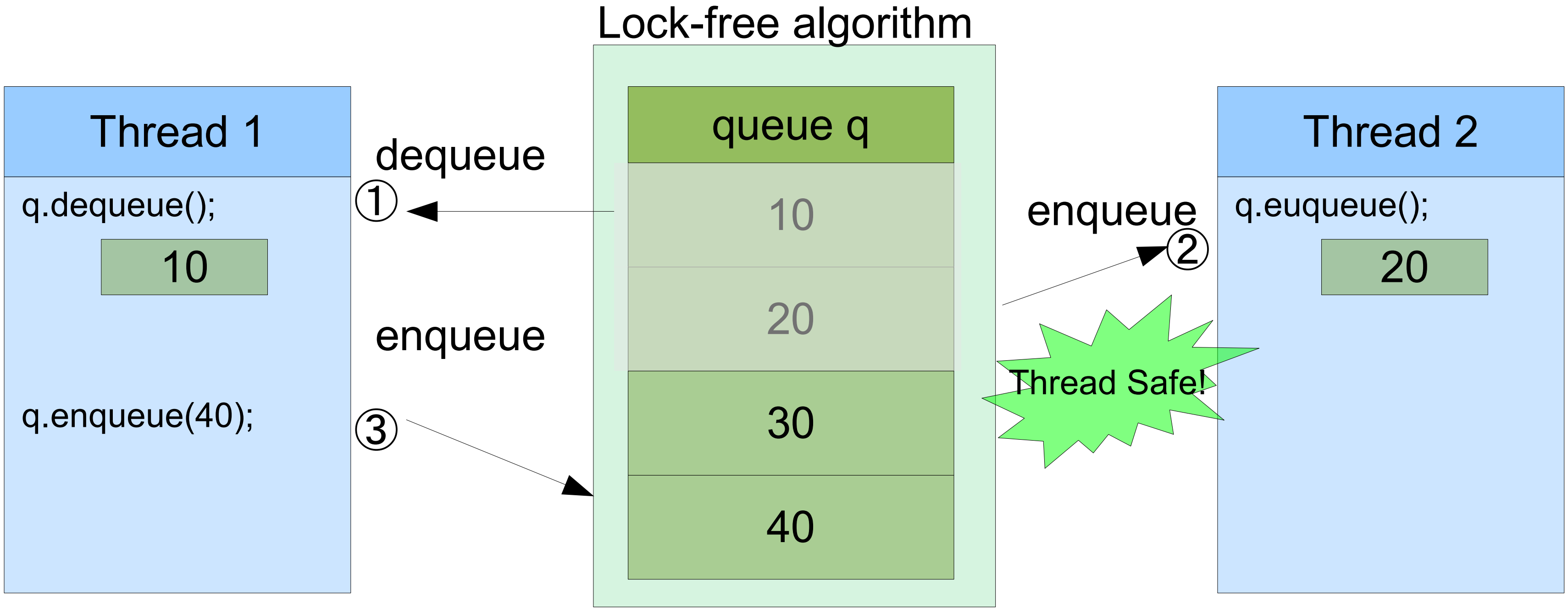
The traditional approach to multi-thread programming is using locks synchronize access to shared resources.
Mutex,Semaphore



1.Introduction

What's Lock free?

Lock-free is "non-blocking" algorithm that is not broken value when access each thread at the same time.



2.Lock free Algorithm

Atomic operation

"atomic operation" is built-ins
 that is no memory operand will be moved across the operation,either forward or backward.

・Test-and-Set operation TAS

```
__sync_test_and_set(&p , a)
```

・Fetch-and-Add(Sub) operation

```
__sync_fetch_and_add(&p , i )
```

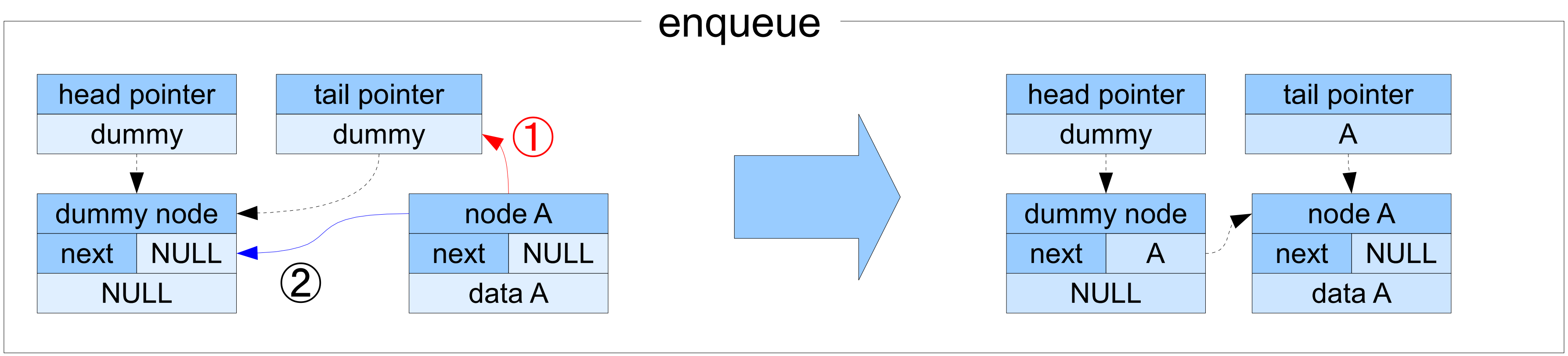
・Compare-and-swap operation CAS

```
__sync_compare_and_swap(&p , a , b )
```

2.Lock free Algorithm

Lock-free queue 1/10

Lock-free queue algorithm(Java Concurrent queue)



enqueue

head pointer
dummy
dummy node
next NULL
NULL

tail pointer
dummy

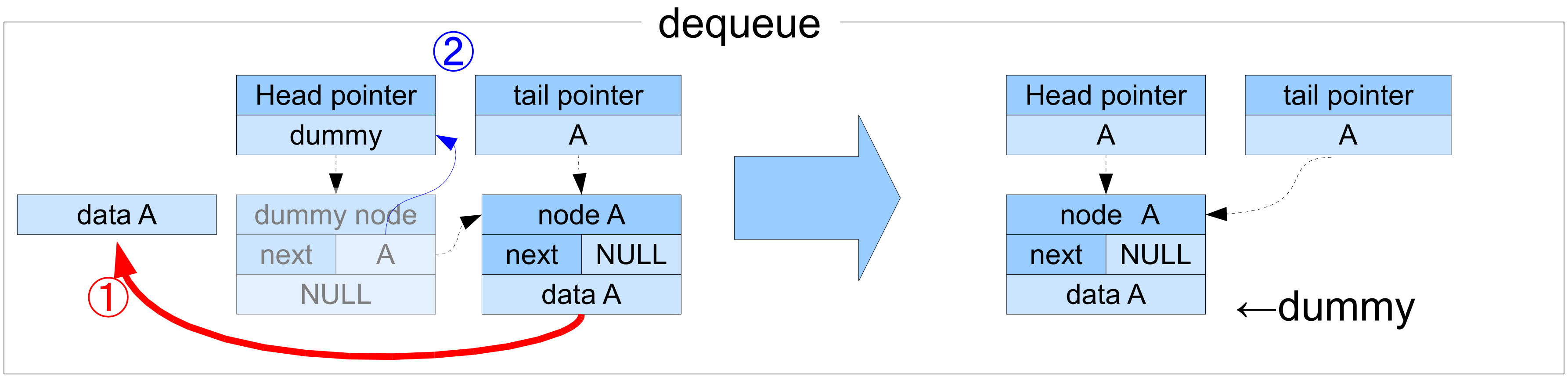
①

node A
next NULL
data A

②

head pointer
dummy
dummy node
next A
NULL

tail pointer
A
node A
next NULL
data A



dequeue

data A

①

Head pointer
dummy
dummy node
next A
NULL

②

tail pointer
A
node A
next NULL
data A

Head pointer
A

tail pointer
A

node A
next NULL
data A

←dummy

2.Lock free Algorithm

Lock-free queue 2/10

Lock-free queue algorithm(Java Concurrent queue)
enqueue(value)

```
E01: node = new node ;
E02: node–>value = value ;
E03: node–>next.ptr = NULL ;
E04: While(true) {
E05:    tail = TailPointer;
E06:    next = tail.ptr–>next;
E07:    if ( tail == TailPointer ) {
E08:        if (next.ptr == NULL ){
E09:            if ( CAS(&tail.ptr–>next, next, node) ) {
E10:                break ;
E11:            }
E12:        } else {
E13:            CAS(&TailPointer, tail, next.ptr) ;
E14:        }
E15:    }
E16: }
E17: CAS(&TailPointer, tail, node) ;
```

2.Lock free Algorithm

Lock-free queue 3/10

Lock-free queue using Java algorithm
dequeue(value)

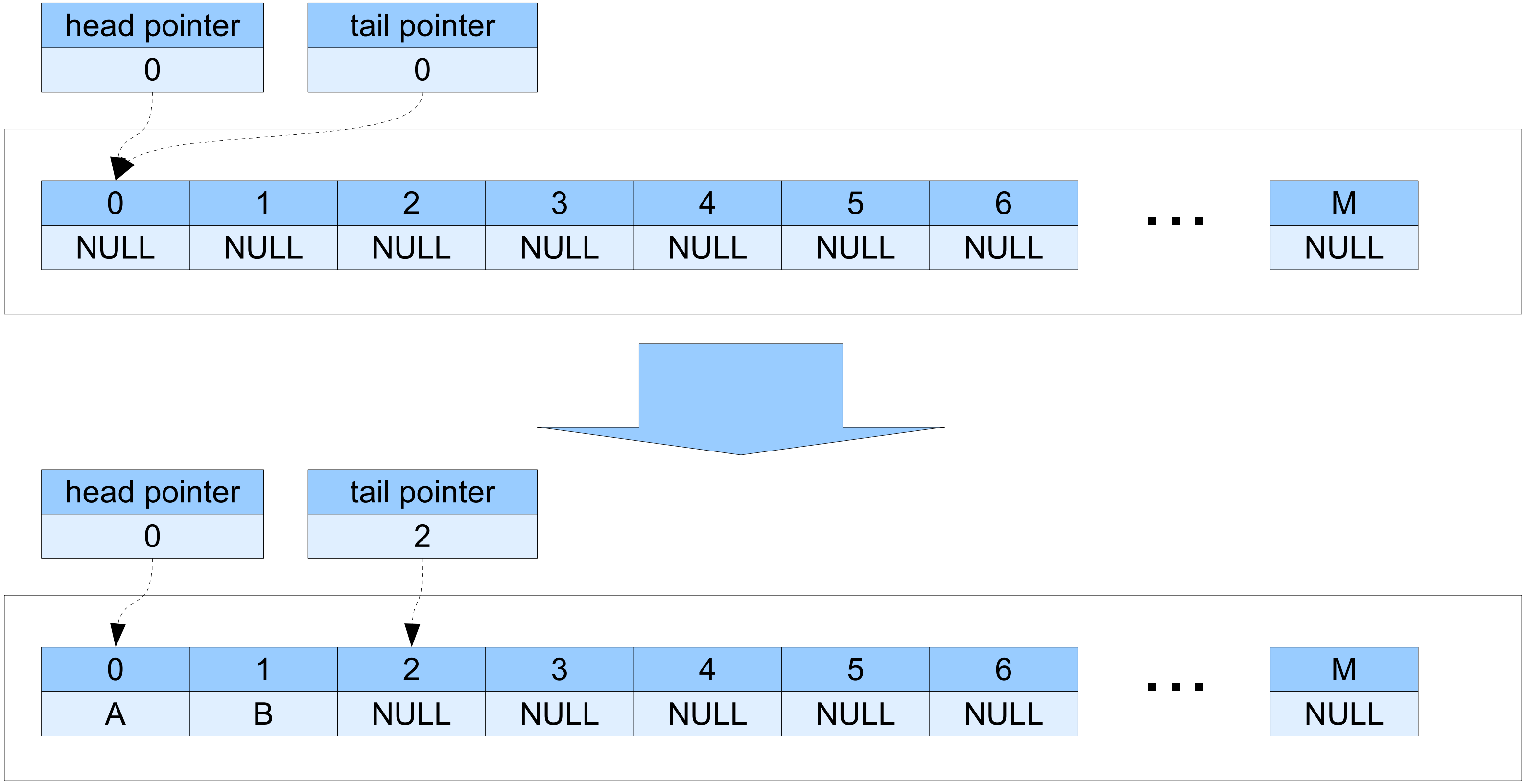
```
while(true){
    head = HeadPointer ;
    tail = TailPointer ;
    next = head–>next ;
    if ( head == HeadPointer ) {
        if ( head.ptr == tail.ptr ) {
            if ( next.ptr == NULL ) {
                return FALSE ;
            }
            CAS(&TailPointer, tail, next.ptr) ;
        } else {
            value = next.ptr–>value ;
            if ( CAS(&HeadPointer, head, next) ) {
                break ;
            }
        }
    }
}
delete head ;
return true ;
```

Critical Session

2.Lock free Algorithm

Lock-free queue 4/10

Non liner lock-free queue (using arrangement)



head pointer	tail pointer
0	0

0	1	2	3	4	5	6	…	M
NULL	NULL	NULL	NULL	NULL	NULL	NULL		NULL

head pointer	tail pointer
0	2

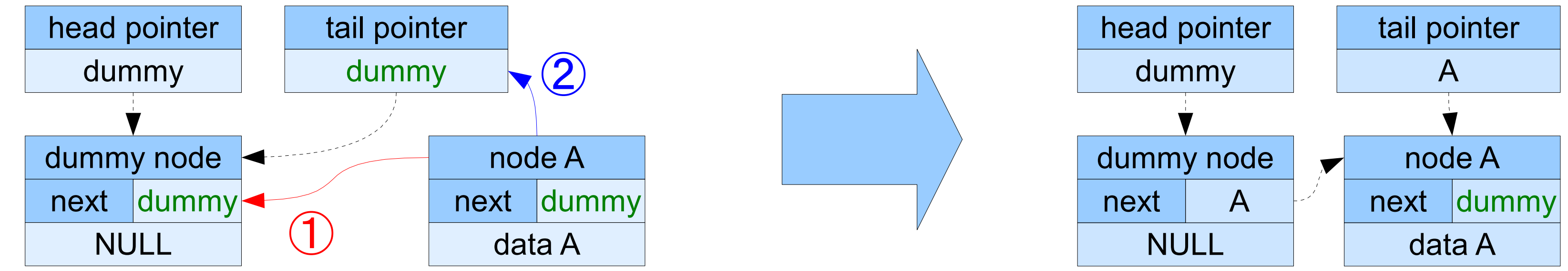
0	1	2	3	4	5	6	…	M
A	B	NULL	NULL	NULL	NULL	NULL		NULL

2.Lock free Algorithm

Lock-free queue 5/10

Lock-free queue new algorithm

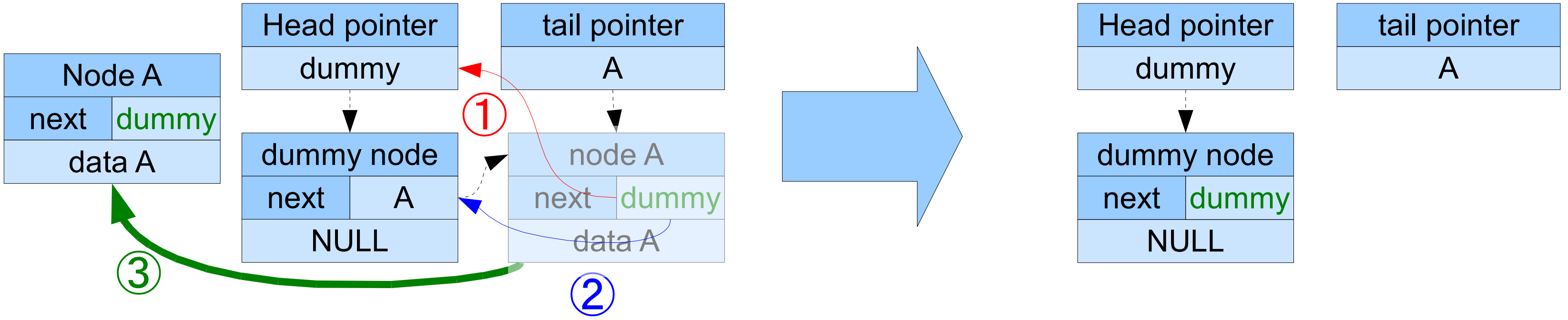
enqueue pattan1



head pointer
dummy
tail pointer
dummy
②
dummy node
next dummy
NULL
①
node A
next dummy
data A

head pointer
dummy
tail pointer
A
dummy node
next A
NULL
node A
next dummy
data A

dequeue pattan1



Node A
next dummy
data A
Head pointer
dummy
tail pointer
A
①
dummy node
next A
NULL
node A
next dummy
data A
②
③

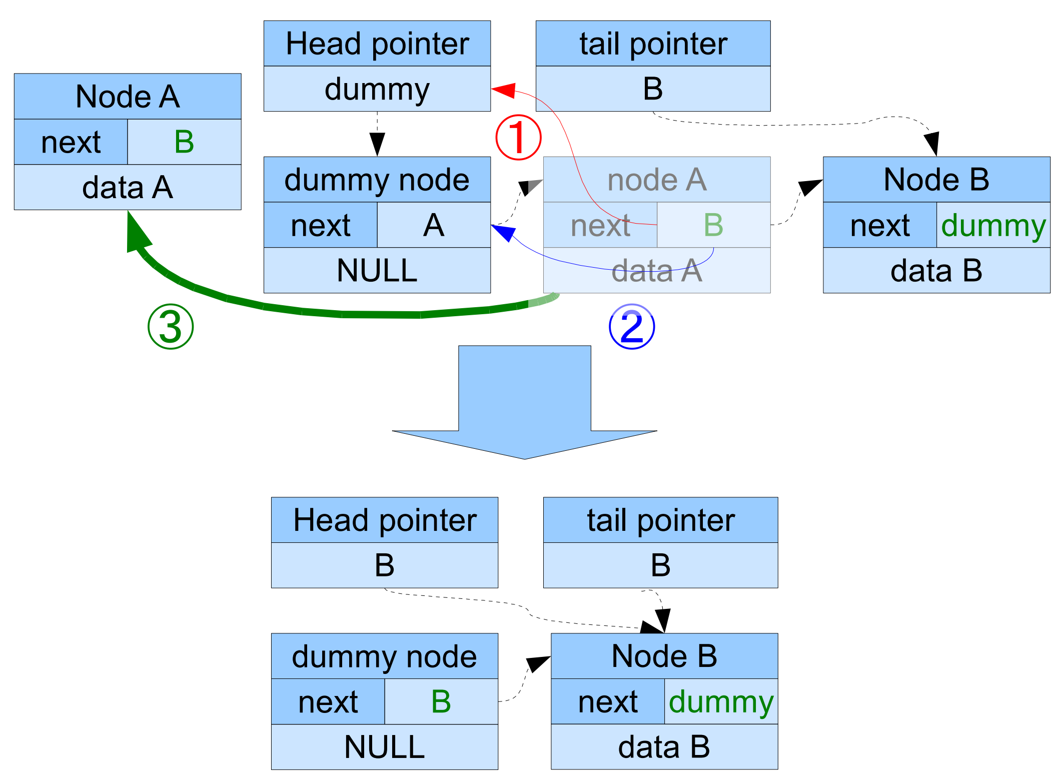
Head pointer
dummy
tail pointer
A
dummy node
next dummy
NULL

2.Lock free Algorithm

Lock-free queue 6/10

Lock-free queue new algorithm

dequeue pattan2

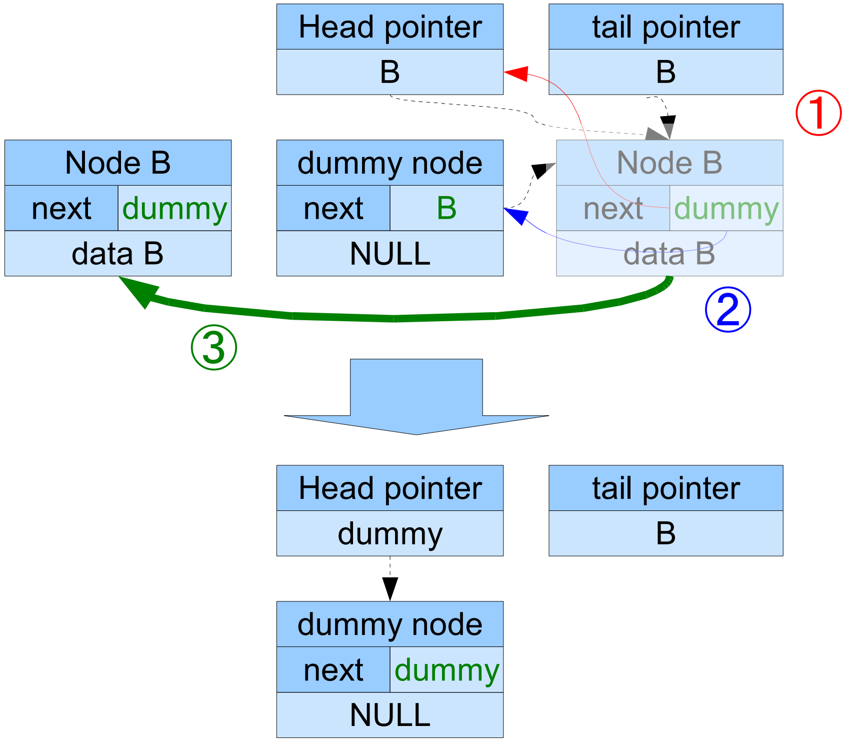


2.Lock free Algorithm

Lock-free queue 7/10

Lock-free queue new algorithm

dequeue pattan3 (case of dequeue2)



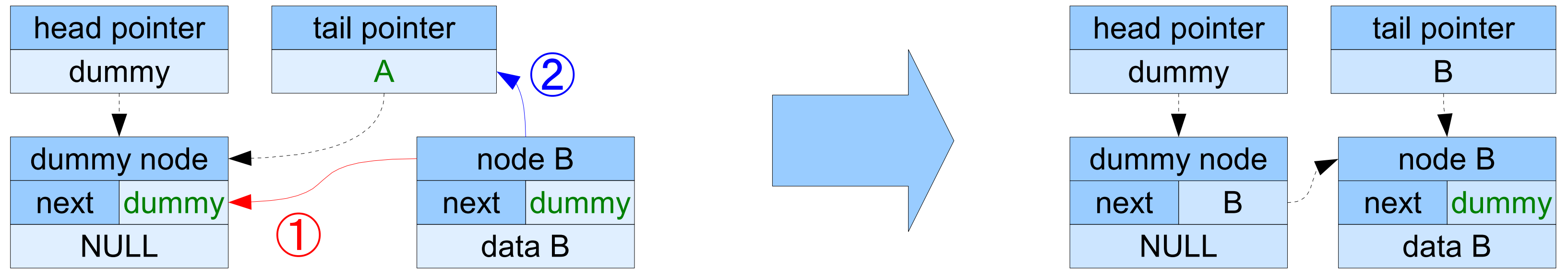
Same result dequeue pattan1

2.Lock free Algorithm

Lock-free queue 8/10

Lock-free queue new algorithm

enqueue pattan2 (case of dequeue1)

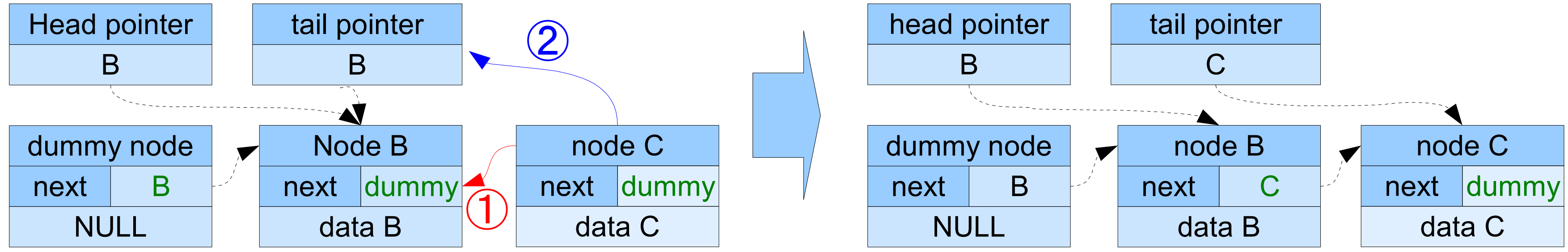


head pointer
dummy
tail pointer
A
②
dummy node
next
dummy
NULL
①
node B
next
dummy
data B

head pointer
dummy
tail pointer
B
dummy node
next
B
NULL
node B
next
dummy
data B

Same result dequeue pattan1

enqueue pattan3 (case of dequeue2)



Head pointer
B
tail pointer
B
②
dummy node
next
B
NULL
Node B
next
dummy
data B
①
node C
next
dummy
data C

head pointer
B
tail pointer
C
dummy node
next
B
NULL
node B
next
C
data B
node C
next
dummy
data C

Same result dequeue pattan2

2.Lock free Algorithm

Lock-free queue 9/10

Lock-free queue using new algorithm
enqueue(value)

```
E01: node = new node ;
E02: node–>value = value ;
E03: node–>next.ptr = dummy ;
E04: while( true ) {
E05:     tail = TailPointer;
E06:     if ( CAS(&tail–>next,dummy,node) ) {
E07:             CAS(&tail,TailPointer,node) ;
E08:             break ;
E09:     } else {
E10:             next =  tail->next;
E11:             if ( next != dummy ) {
E12:                     CAS(&TailPointer, tail ,next ) ;
E13:             }
E14:     }
E15: }
```

2.Lock free Algorithm

Lock-free queue 10/10

Lock-free queue using new algorithm
dequeue(value)

```
while ( true ) {
    head = HeadPointer ;
    next = HeadPointer –>next ;
    if ( head == HeadPointer ) {
        if ( head == dummy ) {
            if ( next == dummy )  return false ;
            if ( CAS(HeadPointer, head, next–>next) ){
                TAS(&dummy–>next,dummy) ;
                value = next–>value ;
                delete next ;
                return true ;
            }
        } else {
            if ( CAS(HeadPointer, head, next–>next) ) {
                CAS(TailPointer, head , dummy) ;
                value = head->value ;
                delete head ;
                return true;
            }
        }
    }
}
```

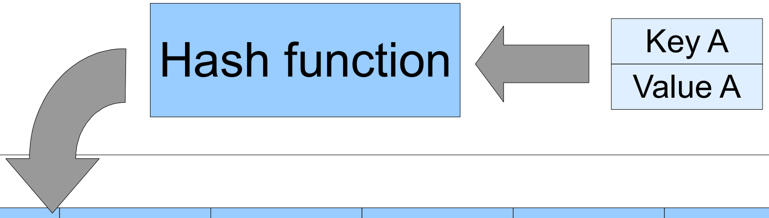
2.Lock free Algorithm

Lock-free hash-map

Lock-free hash-map algorithm

hash-map

	0	1	2	3	4	5	6	…	M
key	NULL	NULL	NULL	NULL	NULL	NULL	NULL		NULL
value	NULL	NULL	NULL	NULL	NULL	NULL	NULL		NULL

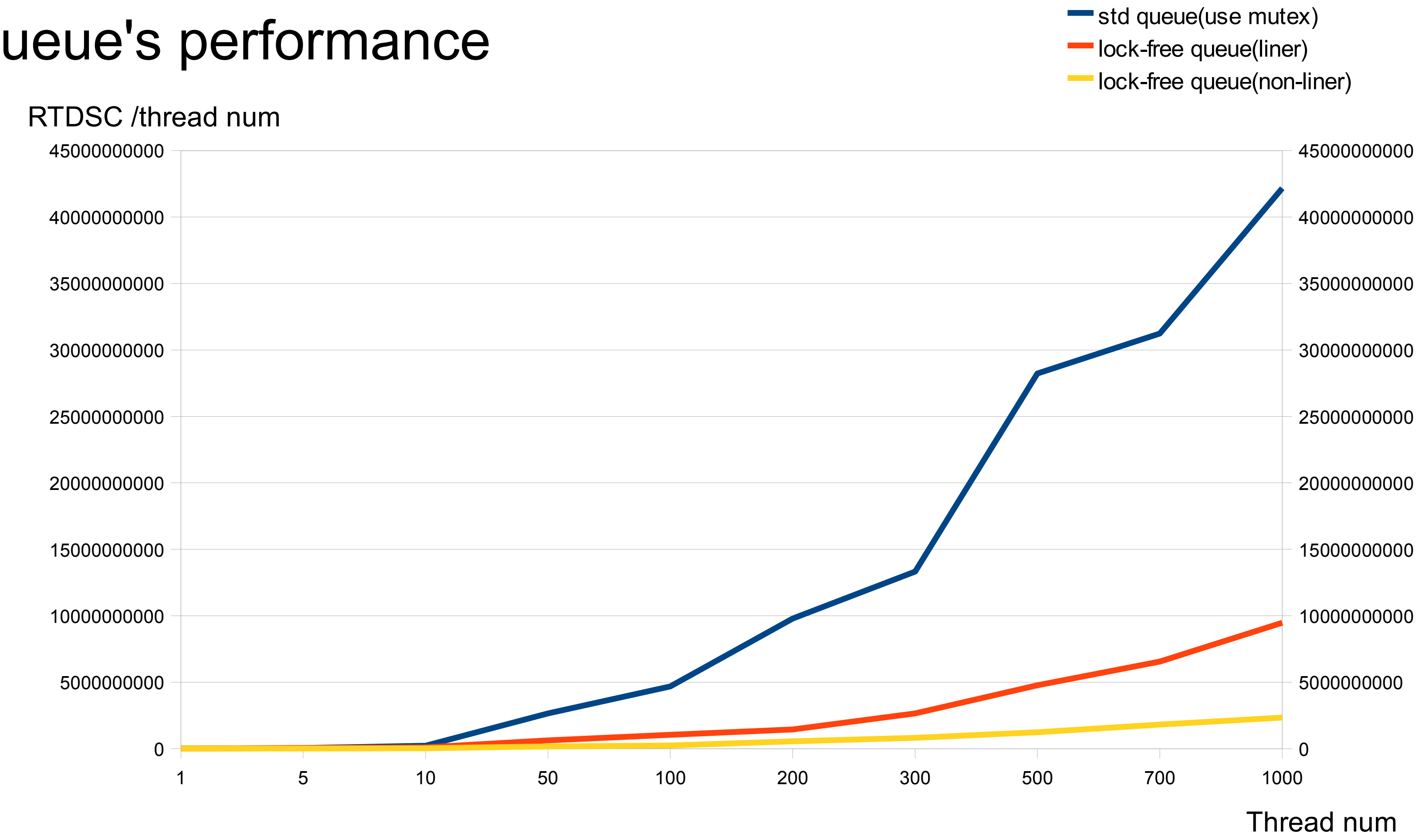


hash-map

	0	1	2	3	4	5	6	…	M
key	NULL	NULL	A	NULL	NULL	NULL	NULL		NULL
value	NULL	NULL	A	NULL	NULL	NULL	NULL		NULL

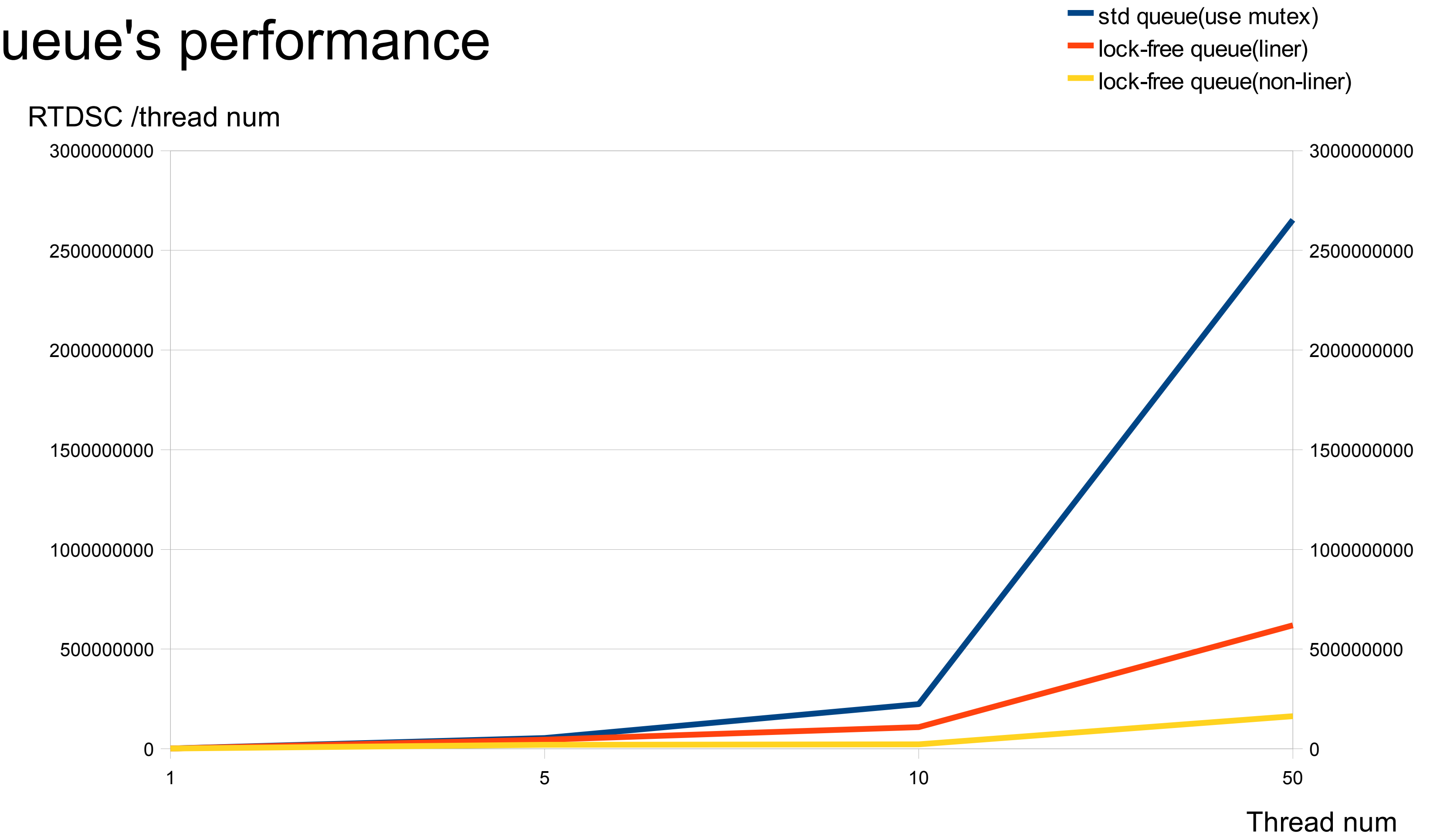
3.Performance

queue's performance



3.Performance

queue's performance



4.Summary

- Possible simple coding on application

- Possible access faster than use mutex

- I will plans to create “list” and “liner hash-map”. Present,considering some method that is find(),delete().. base on liner queue.

See source forge website:

http://sourceforge.jp/projects/c-lockfree/