Lock free Algorithm for Multi-core architecture

SDY Corporation
Hiromasa Kanda

Contents

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?

- Micro Parallelization **Multi-Process** ➡ **Multi-Thread**

Amdahl's law

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

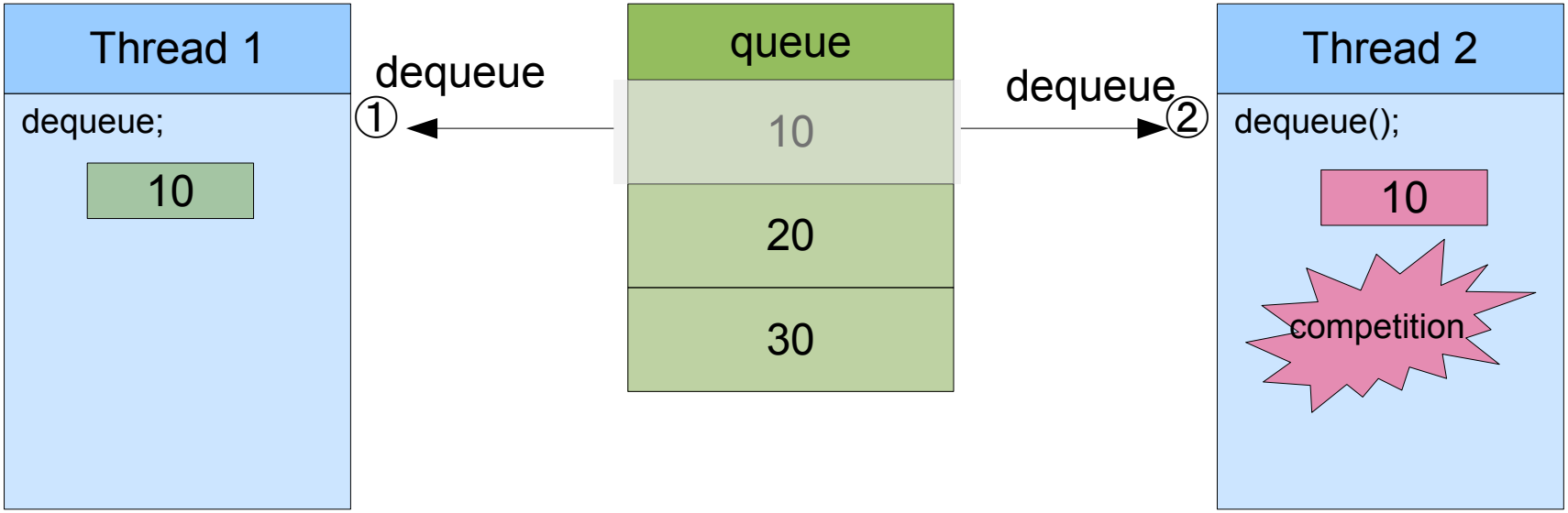
Problem of Multi-thread

- Shared resource synchronization

1.Introduction

Problem of Multi-Thread program 1/2

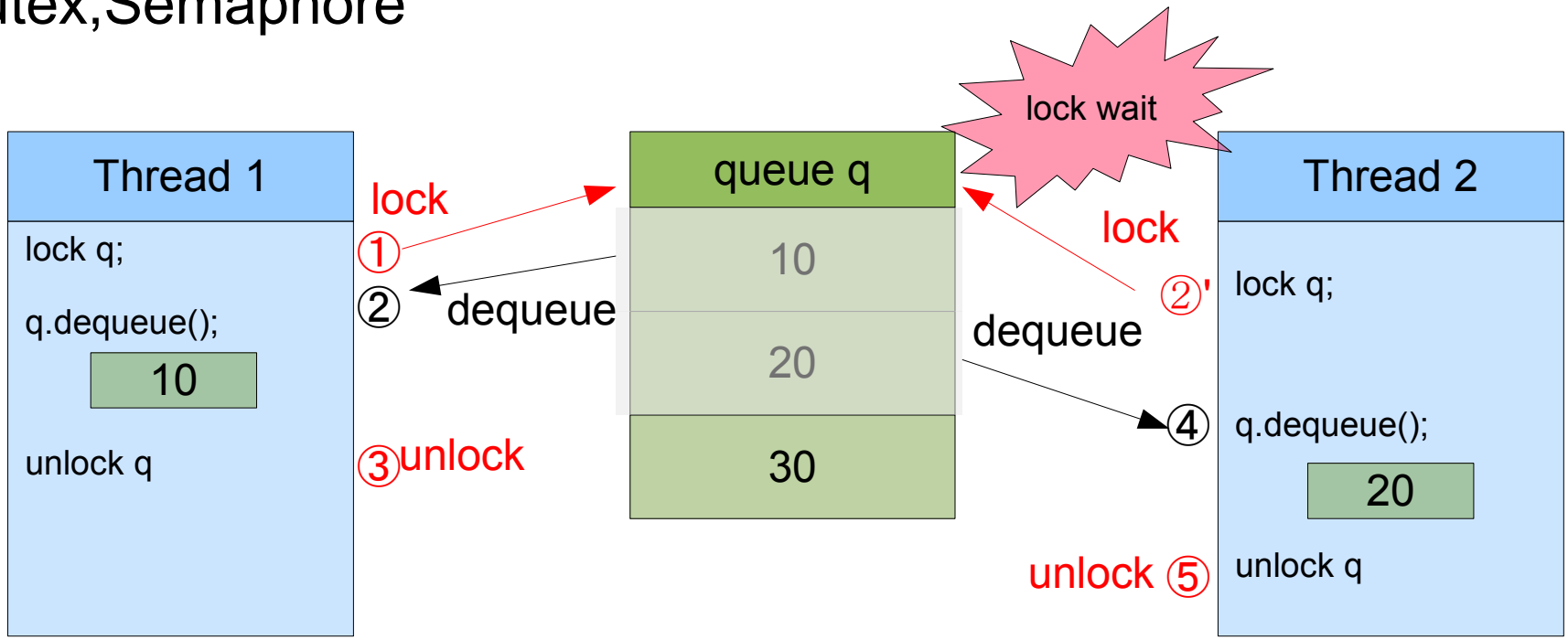
There was resources problem that share it when concurrent access multi-thread.



1.Introduction

Problem of Multi-Thread program 2/2

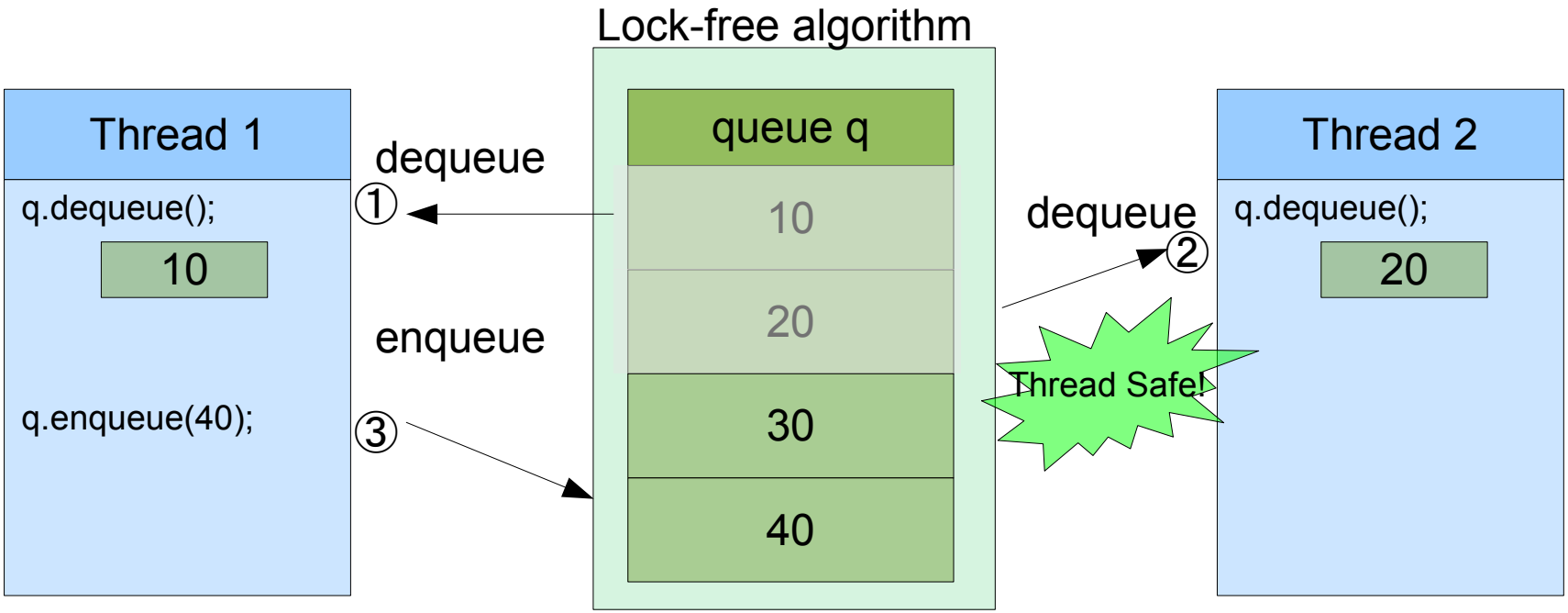
The traditional approach to multi-thread programming is using locks synchronize access to shared resources.
Mutex,Semaphore



1.Introduction

What's Lock free?

Lock-free is "non-blocking" algorithm that doesn't break value when access each thread at the same time.



1.Introduction

Lock-free algorithm performance

Lock-free algorithm performance is better than that of mutex lock algorithm, 100 times from 10 times.

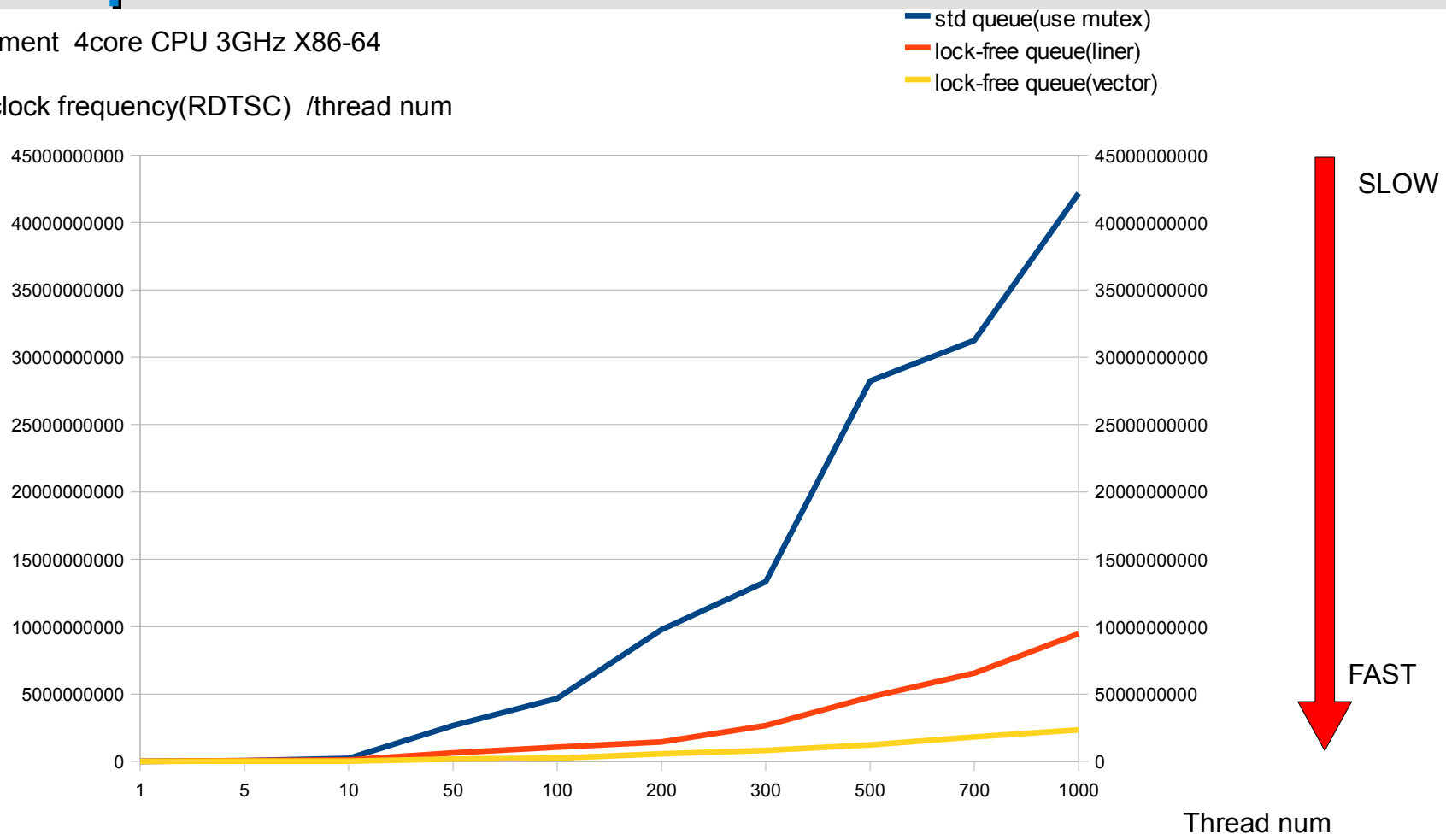
The difference extends further for more CPUs.

2.Lock free Algorithm

Queue performance

Measurement 4core CPU 3GHz X86-64

clock frequency(RDTSC) /thread num

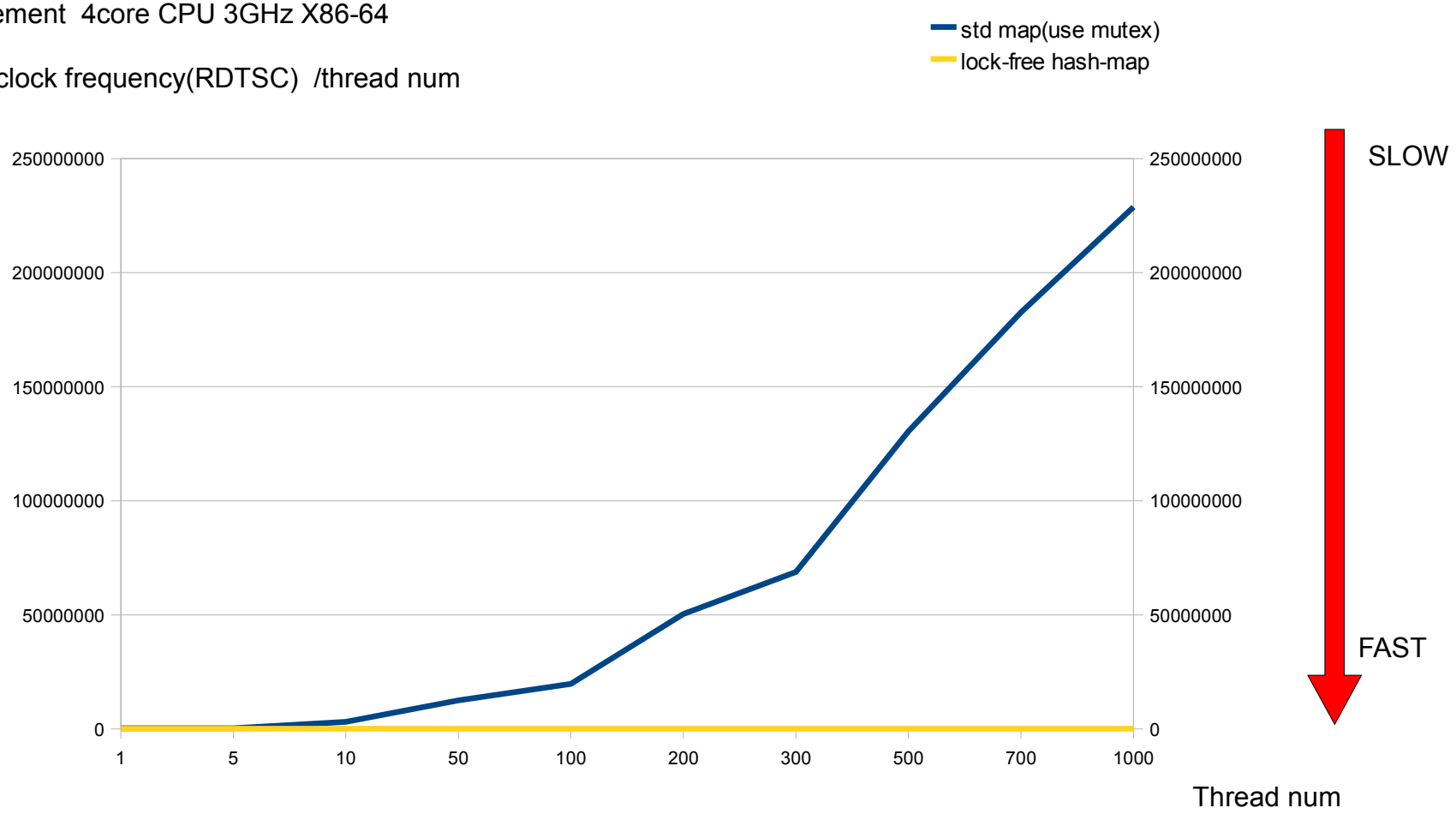


2.Lock free Algorithm

Lock-free hash-map performance

Measurement 4core CPU 3GHz X86-64

clock frequency(RDTSC) /thread num



2.Lock free Algorithm

Lock-free algorithm elements

Lock-free algorithm use 2 elements.

1) atomic operation.
Not use memory barrier. Blocked small memory address.
This new feature permits applications to use atomically update memory without using other synchronization primitives.
gcc4.1 support this operation on user space.

2) key-value transaction
Atomic operation is uses one key value.
Small memory address transaction composes lockfree algorithm data statement.

2.Lock free Algorithm

Lock-free Algorithm basic concept 1/3

Basic concept

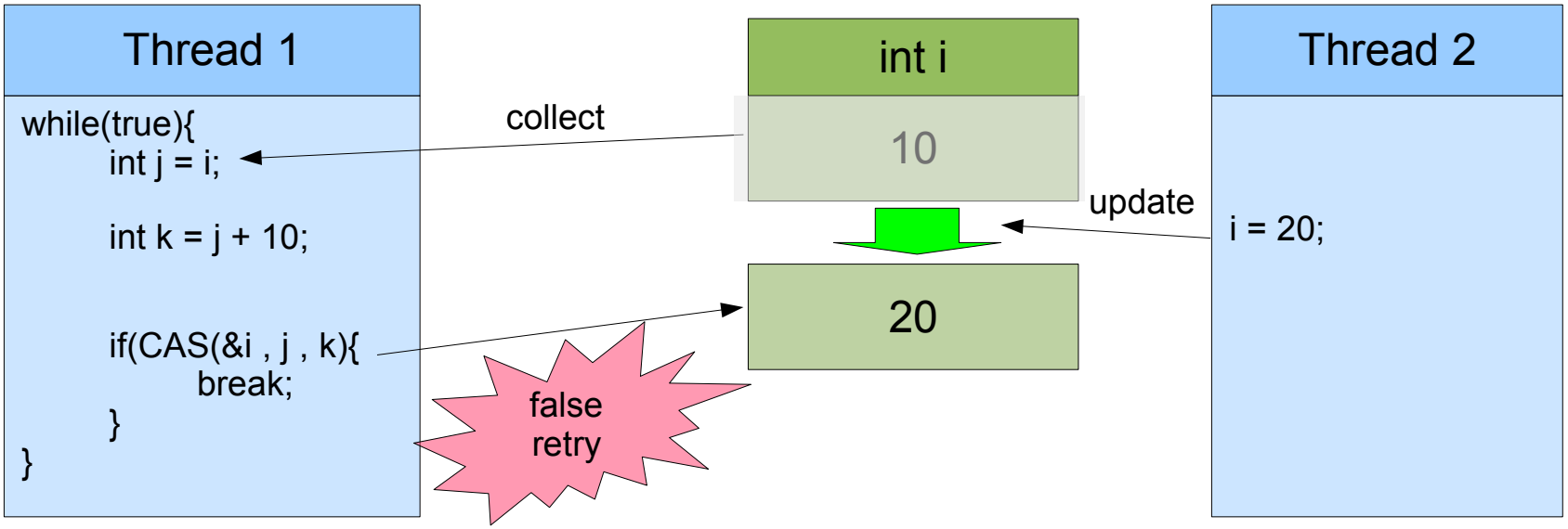
```
while(true){

    1.Data(pointer) collect

    2.Some operation

    3.Data(pointer) compare & update
        use CAS
    If (CAS = true) break;

}
```

Exclusive

Loop end case of CAS true

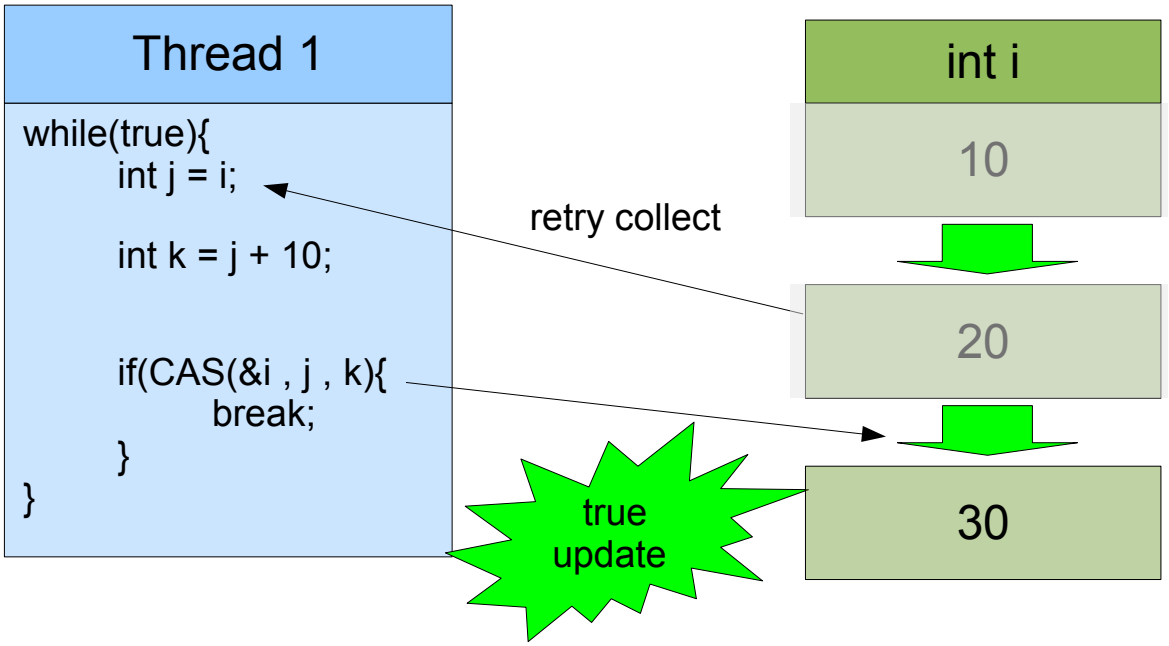
2.Lock free Algorithm

Lock-free Algorithm basic concept 2/3



2.Lock free Algorithm

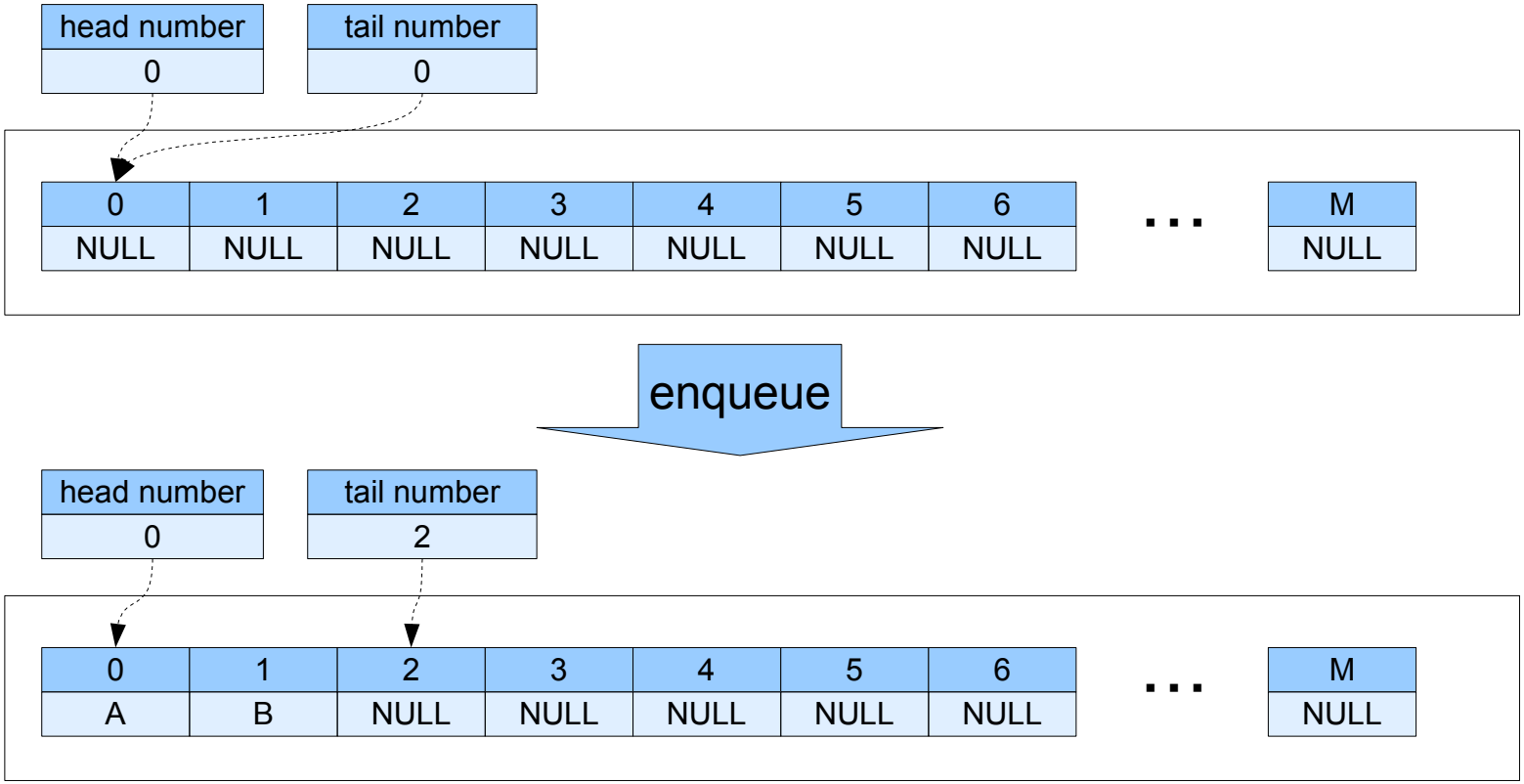
Lock-free Algorithm basic concept 3/3



2.Lock free Algorithm

Lock-free queue (using arrangement) 1/3

Vector lock-free queue (using arrangement)



2.Lock free Algorithm

Lock-free queue (using arrangement) 2/3

enqueue(value)

```
while(true){
        tail = tail_number;

        if ( CAS(&tail_number , tail , tail + 1 ) ) break;
}

node[tail].value = value;
```

Data collect (tail pointer)

Loop end case of tail pointer value is equal to tail

Set value

2.Lock free Algorithm

Lock-free queue (using arrangement) 3/3

dequeue(value)

```
while(true){
        head = head_number ;
        if( !(node[head].value) ){
                if( head_number == tail pointer ) return false;

        } else {
                if( CAS(&head_number , head, head + 1 ) ){

                        value = node[head].value;
                        Break;
                }
        }
}

node[head].value = NULL;

return true;
```

Data collect (head pointer)

Queue is empty?

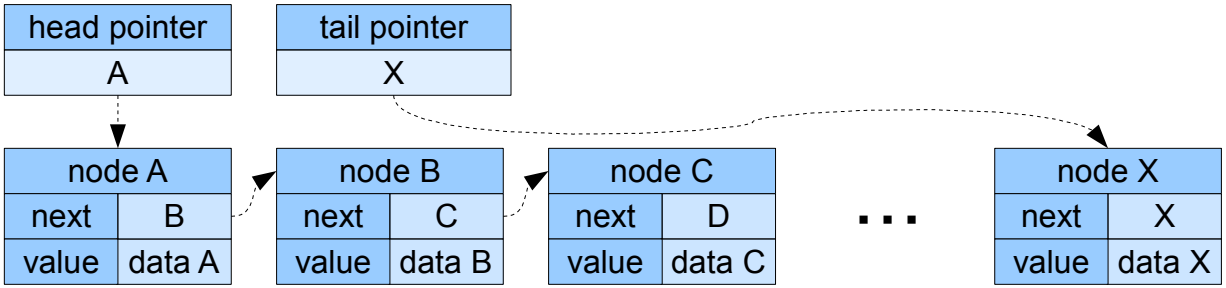
Loop end case of head pointer value is equal to head

Erase value

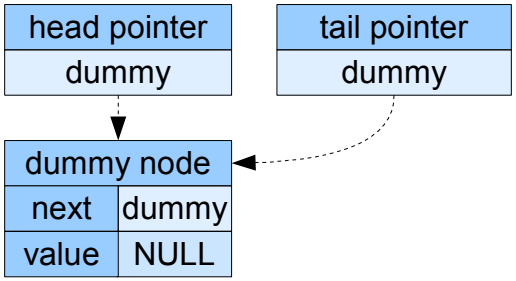
2.Lock free Algorithm

Lock-free queue (liner) 1/5

Liner Lock-free queue algorithm



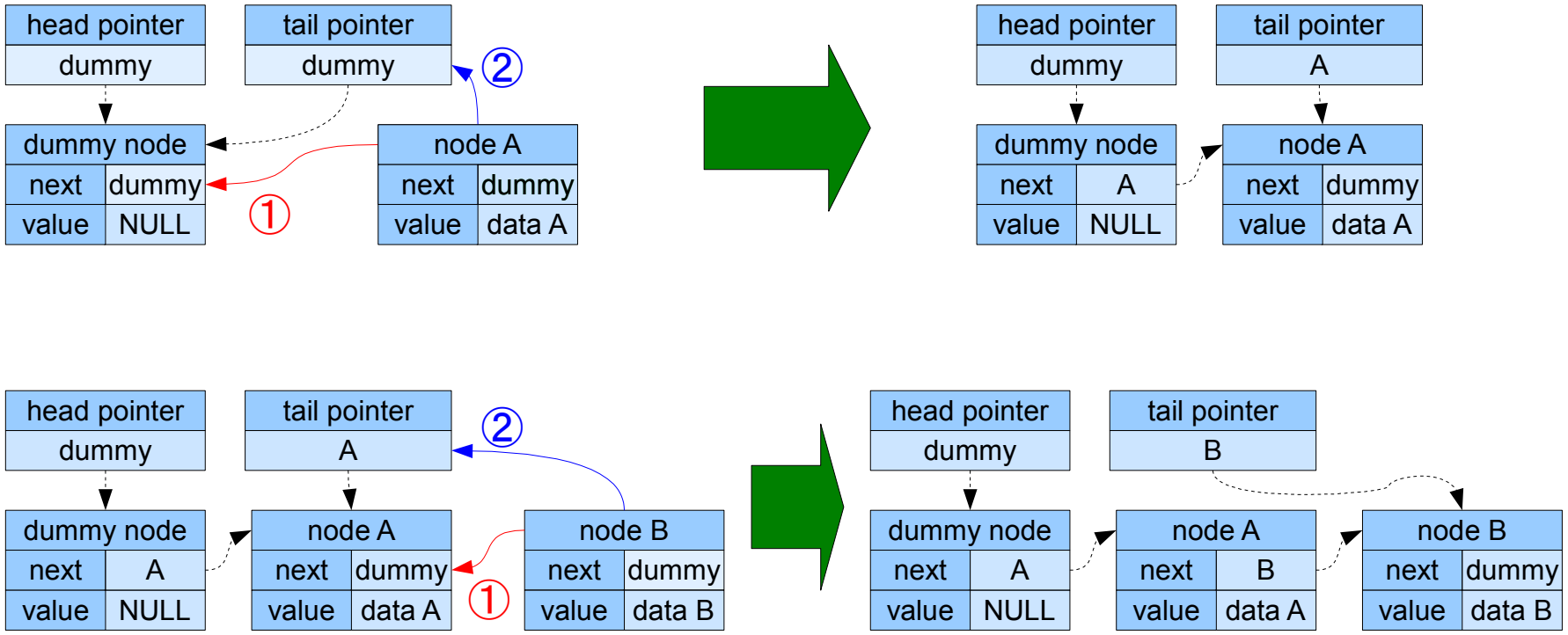
Initial state



2.Lock free Algorithm

Lock-free queue (liner) 2/5

enqueue



2.Lock free Algorithm

Lock-free queue (liner) 3/5

enqueue(value)

```
new_node = new node_type();
new_node->value = value;
new_node->next = dummy;

while( true ){

        tail = tail_pointer;


        if( CAS(&tail_pointer->next , dummy , new_node) ){

                TAS(&tail_pointer , new_node);
                break;

        }
}

return true;
```

Create node.

Data collect (tail pointer).

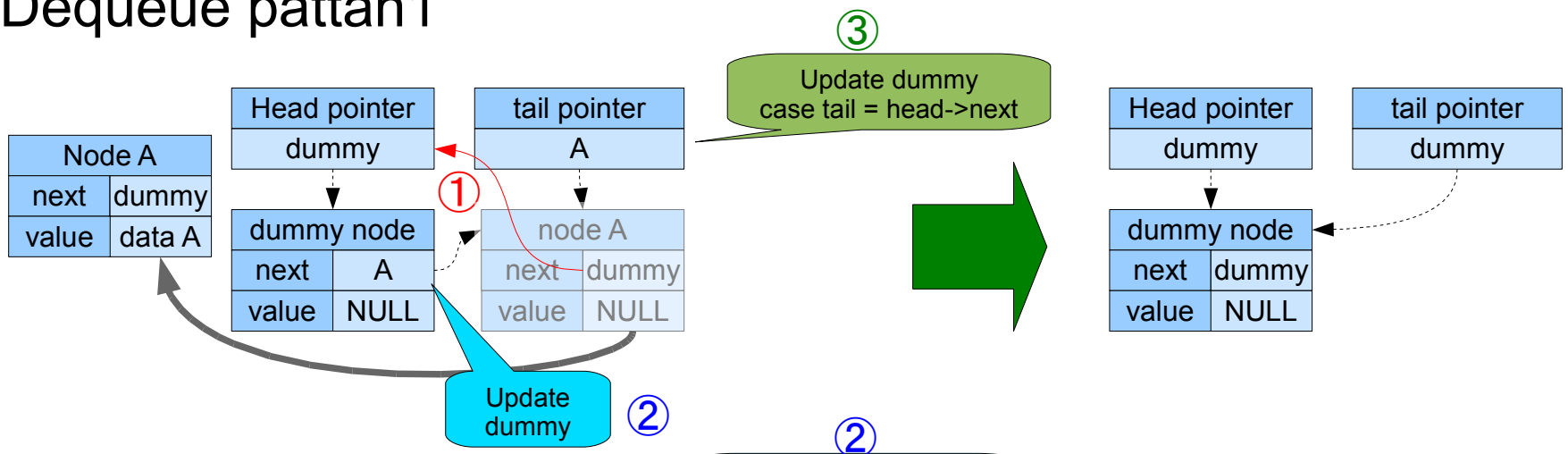
Loop end case of tail pointer's next value is equal to dummy.

Update tail pointer.

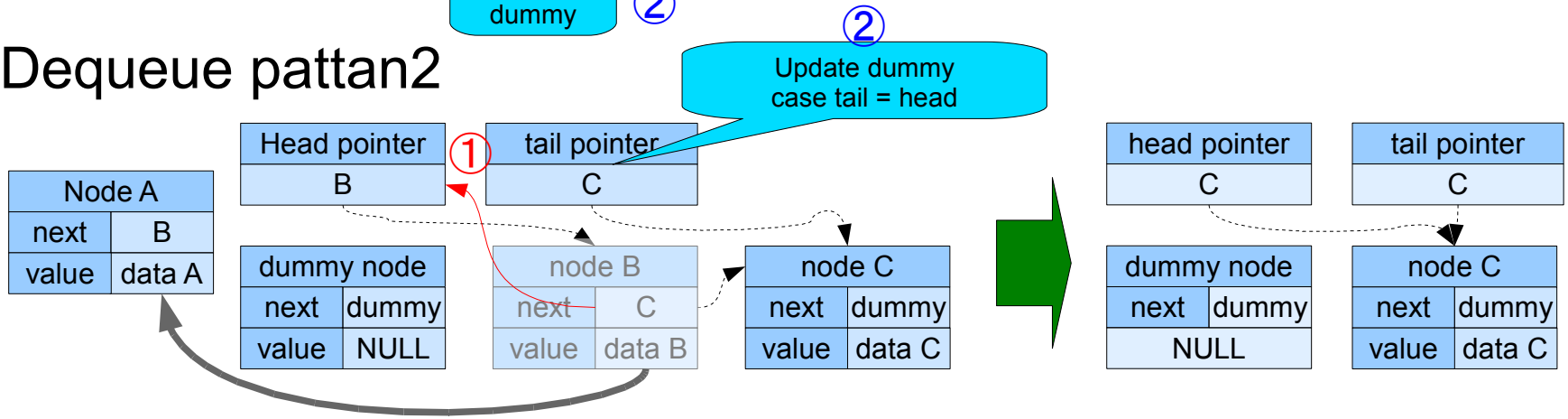
2.Lock free Algorithm

Lock-free queue (liner) 4/5

Dequeue pattan1



Dequeue pattan2



2.Lock free Algorithm

Lock-free queue (liner) 5/5

dequeue(value)

```
while( true ){
        head = head_pointer;
        next = head_pointer->next;
        if( head == dummy ){

                if( next == dummy ) return false;

                if( CAS(&head_pointer , head , next->next) ){
                        TAS(&dummy->next,dummy);
                        CAS(&tail_pointer , next , dummy );
                        value = next->value;
                        delete next;
                        break;
                }
        } else {
                if( CAS(&head_pointer , head , next) ){
                        CAS(&tail_pointer , head , dummy);
                        value = head->value;
                        delete head;
                        break;
                }
        }
}
return true;
```

Data collect (head pointer,next pointer).

Queue empty?

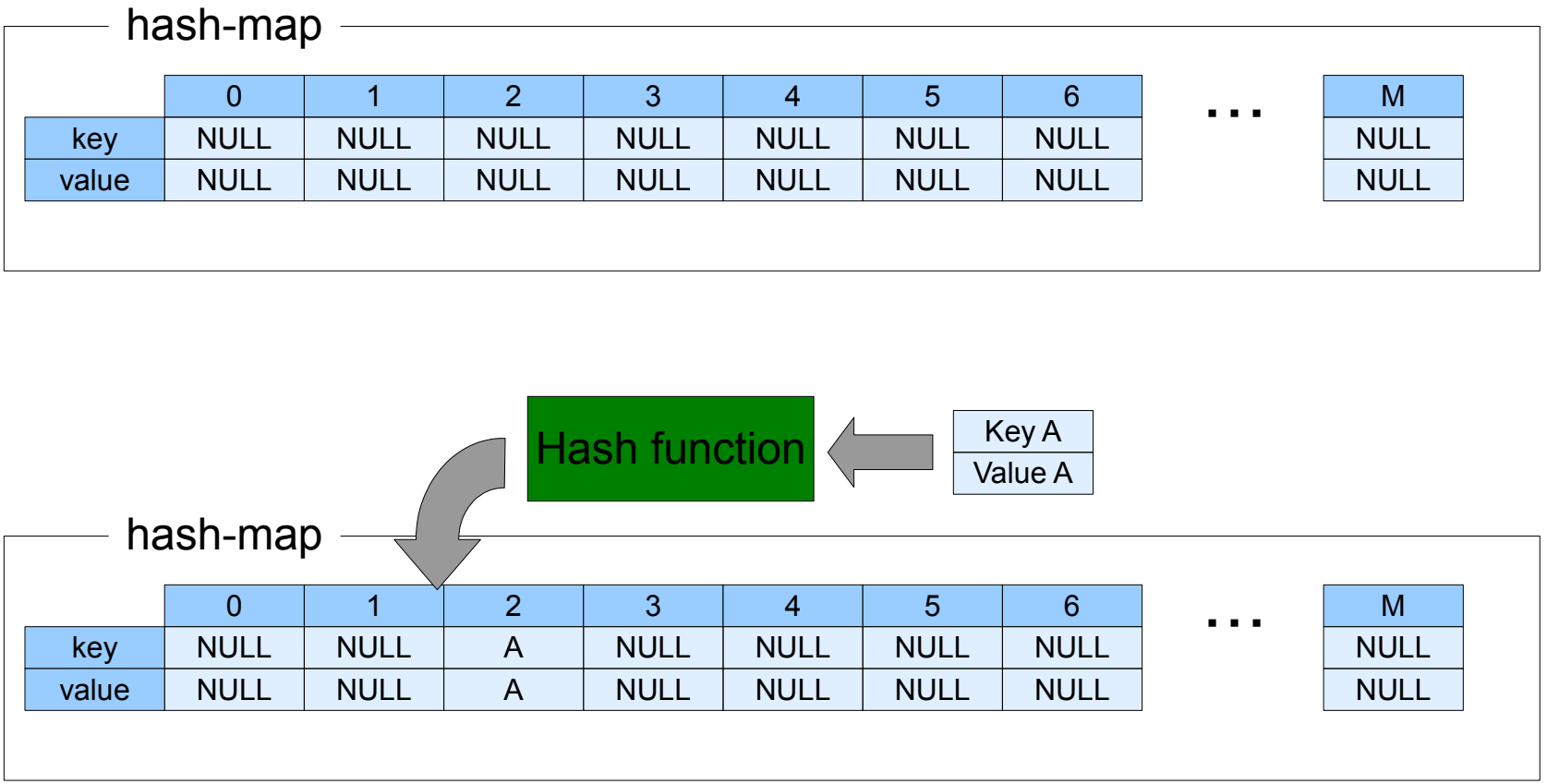
Case Pattan1
Loop end case of head pointer value is equal to collect head.

Case Pattan2
Loop end case of head pointer value is equal to collect head.

2.Lock free Algorithm

Lock-free hash-map 1/4

Lock-free hash-map algorithm



2.Lock free Algorithm

Lock-free hash-map 2/4

Insert(key , value)

```
hashvalue = get_hashvalue( key );
pre_key = NULL
do{
      for(;;){
            if( !hashmap[hashvalue].key ) break;

            if( hashmap[hashvalue].key == key ){
                  pre_key = hashmap[hashvalue].key;
                  break;
            }else{
                  hashmap[hashvalue].rehash = true;
                  hashvalue = get_rehashvalue( hashvalue );

} while( !CAS( &hashmap[hashvalue].key, pre_key, key ) );

TAS(&hashmap[hashvalue].value,value);
```

Get hash value

Break arrangement key is null

Break arrangement key equal to key

Retry re-hash

Loop end CAS true

Set value

2.Lock free Algorithm

Lock-free hash-map 3/4

return value find(key)

```
hashvalue = get_hashvalue( key );

for(;;){
        if( hashmap[hashvalue].key == key ){
                return hashmap[hashvalue].value;


        }else{
                if( !hashmap[hashvalue].rehash ) return NULL;


                hashvalue = get_rehashvalue( hashvalue );
        }
}
```

Get hash value

Return value case arrangement key is equal to key

Return null(false) Case key is not equal to key

Retry after get rehash value

2.Lock free Algorithm

Lock-free hash-map 4/4

erase(key)

```
hashvalue = get_hashvalue( key );
do{
      for(;;){
            if( hashmap[hashvalue].key == key ){
                  pre_key = hashmap[hashvalue].key;
                  break;
            }else{
                  if( !hashmap[hashvalue].rehash ) return false;

                  hashvalue = get_rehashvalue( hashvalue );
            }
      }
}while( !CAS( &hashmap[hashvalue].key, pre_key, NULL) );
TAS(&hashmap[hashvalue].value,NULL);
return true;
```

Get hash value

Data collect

Not find key

Retry after get re-hash value

Erase key and value

4.Summary

- Possible simple coding on userspace application
- Possible access faster than use mutex-lock
- I plan to create lock-free list.
  Present,considering some method that is find(),delete().. base on liner queue.

See source forge website:

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