

Многопоточное Программирование: Железо и спин-локи

Роман Елизаров, JetBrains, elizarov@gmail.com

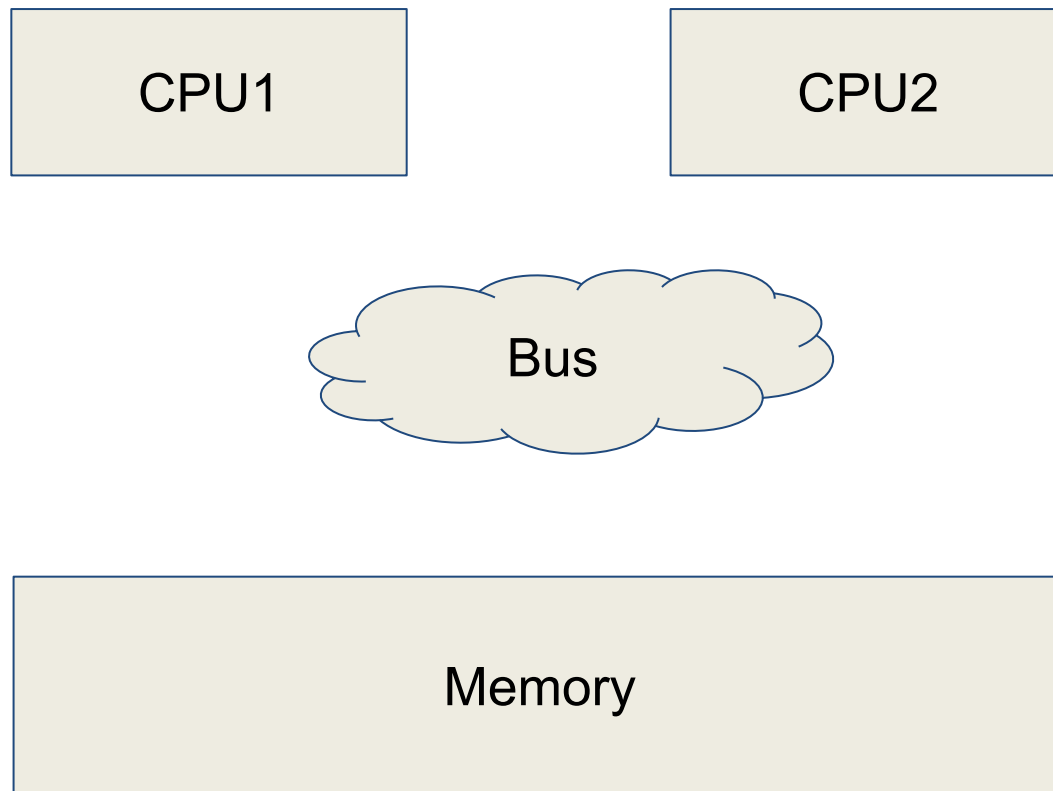
Никита Коваль, JetBrains, ndkoval@ya.ru

ИТМО 2019

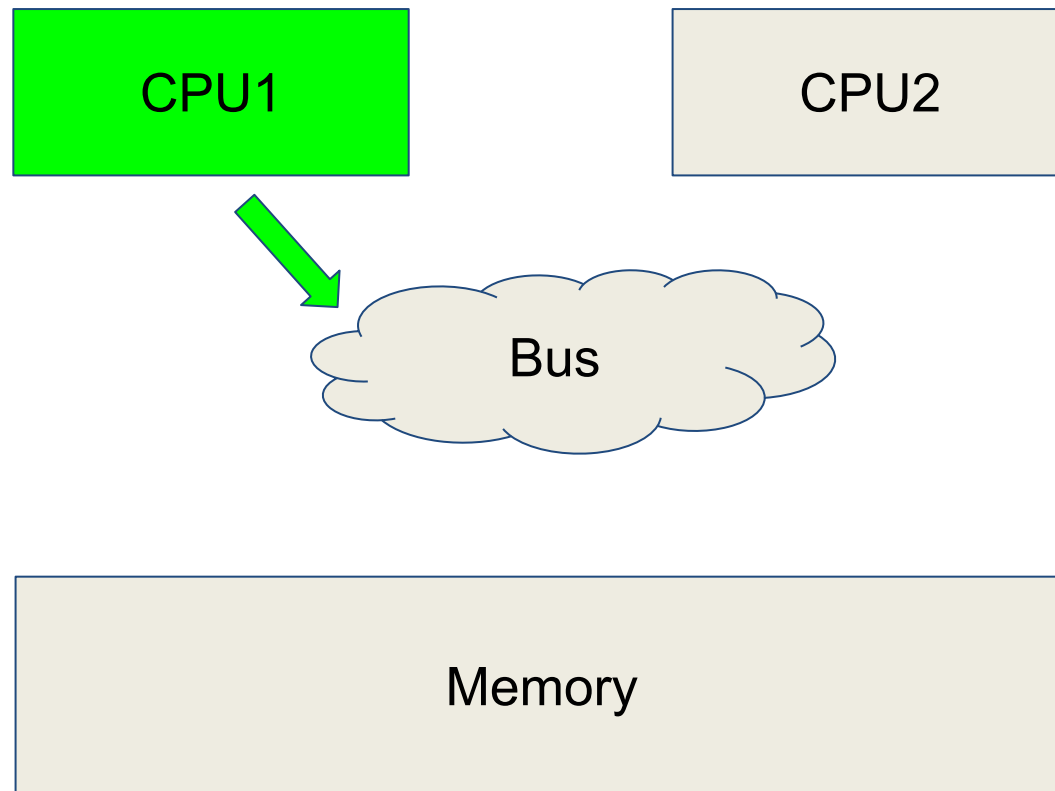


ITMO UNIVERSITY

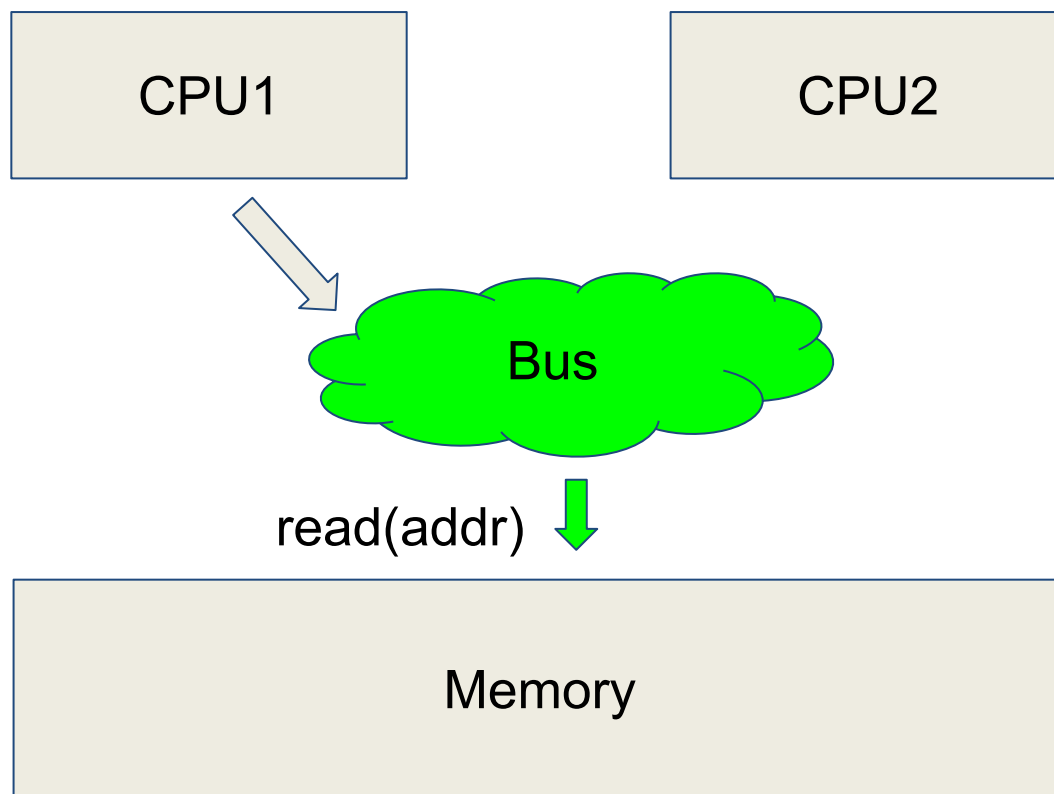
Работа с общей память



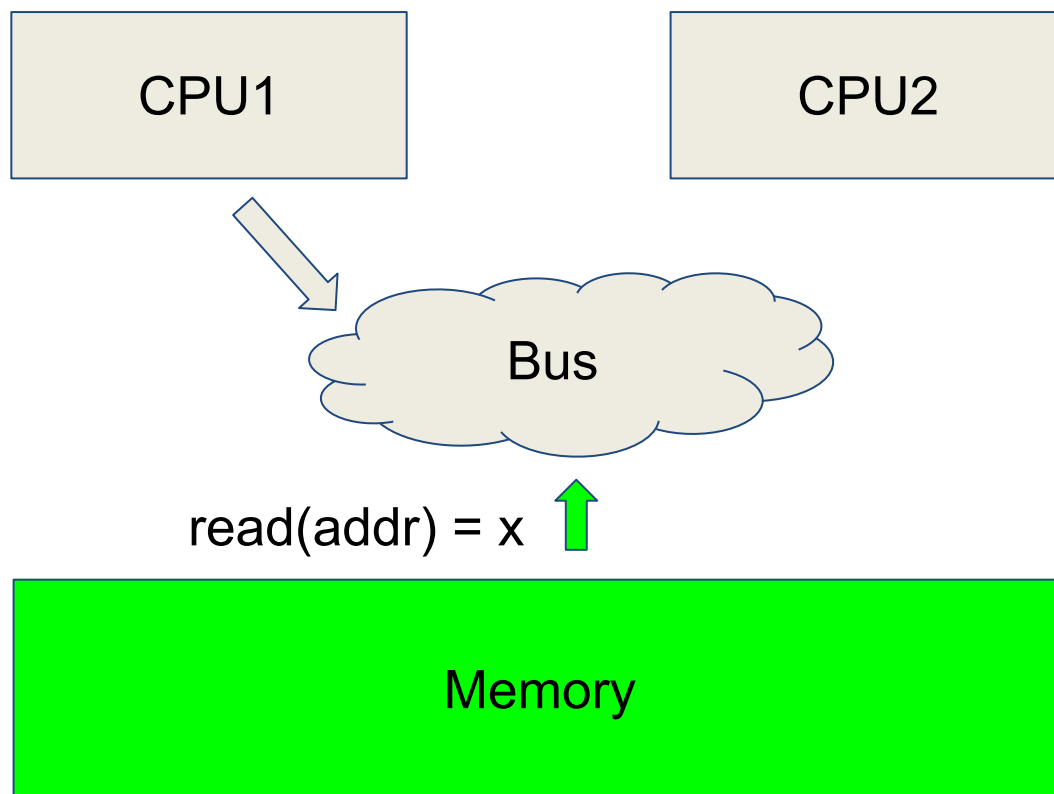
Работа с общей память



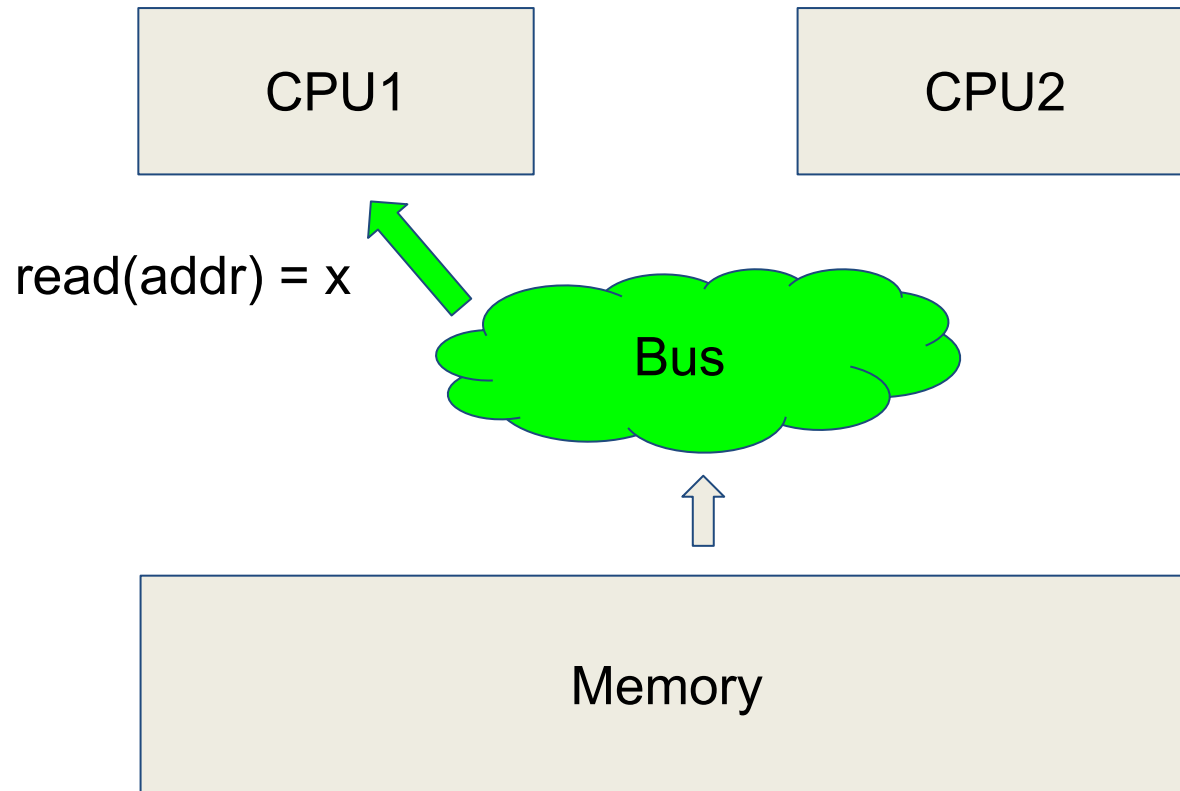
Работа с общей память



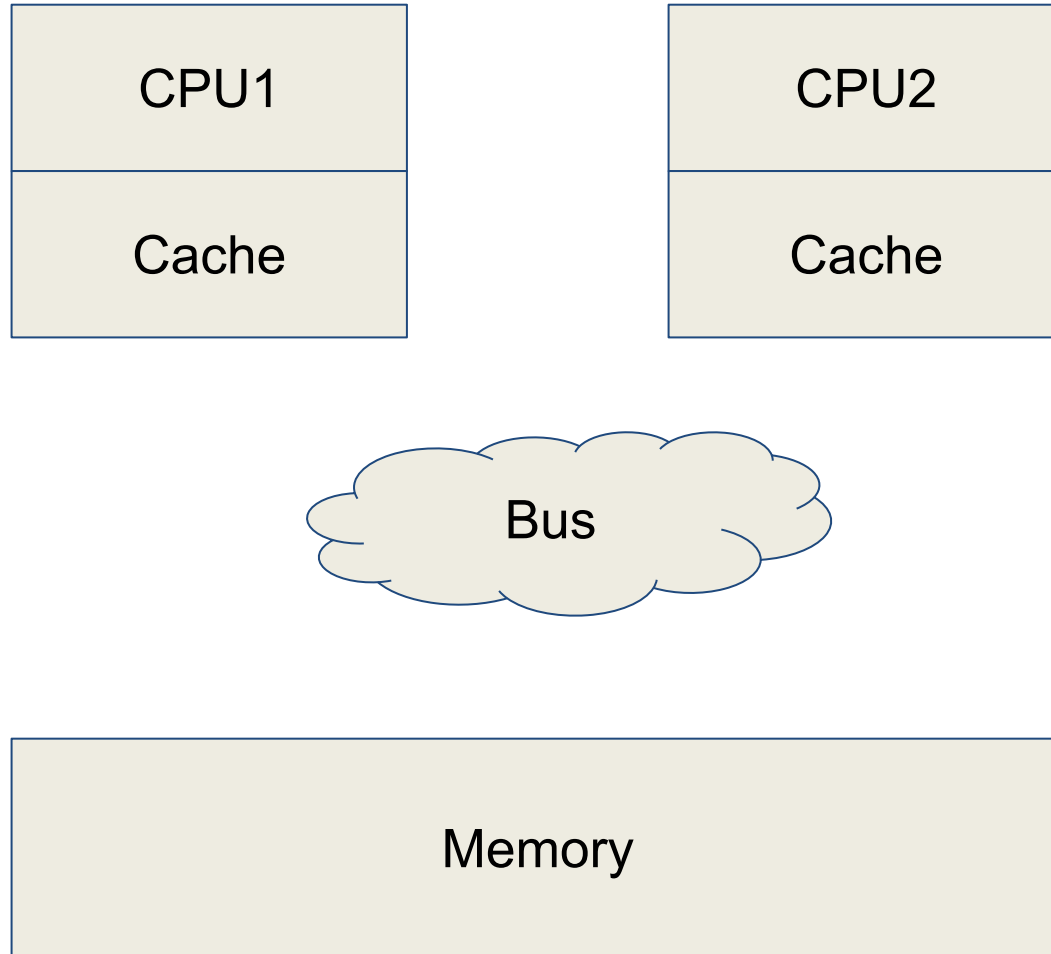
Работа с общей память



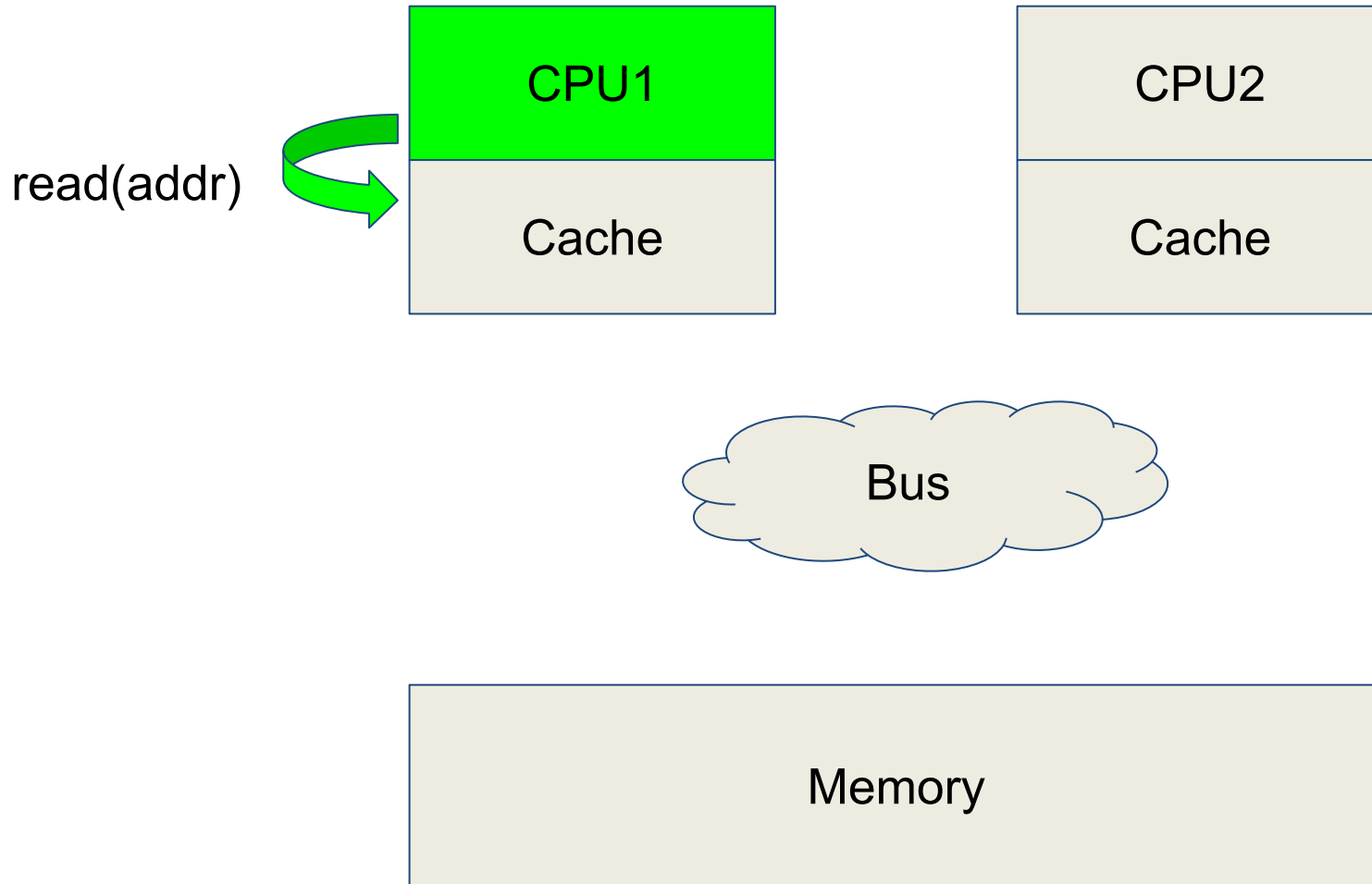
Работа с общей память



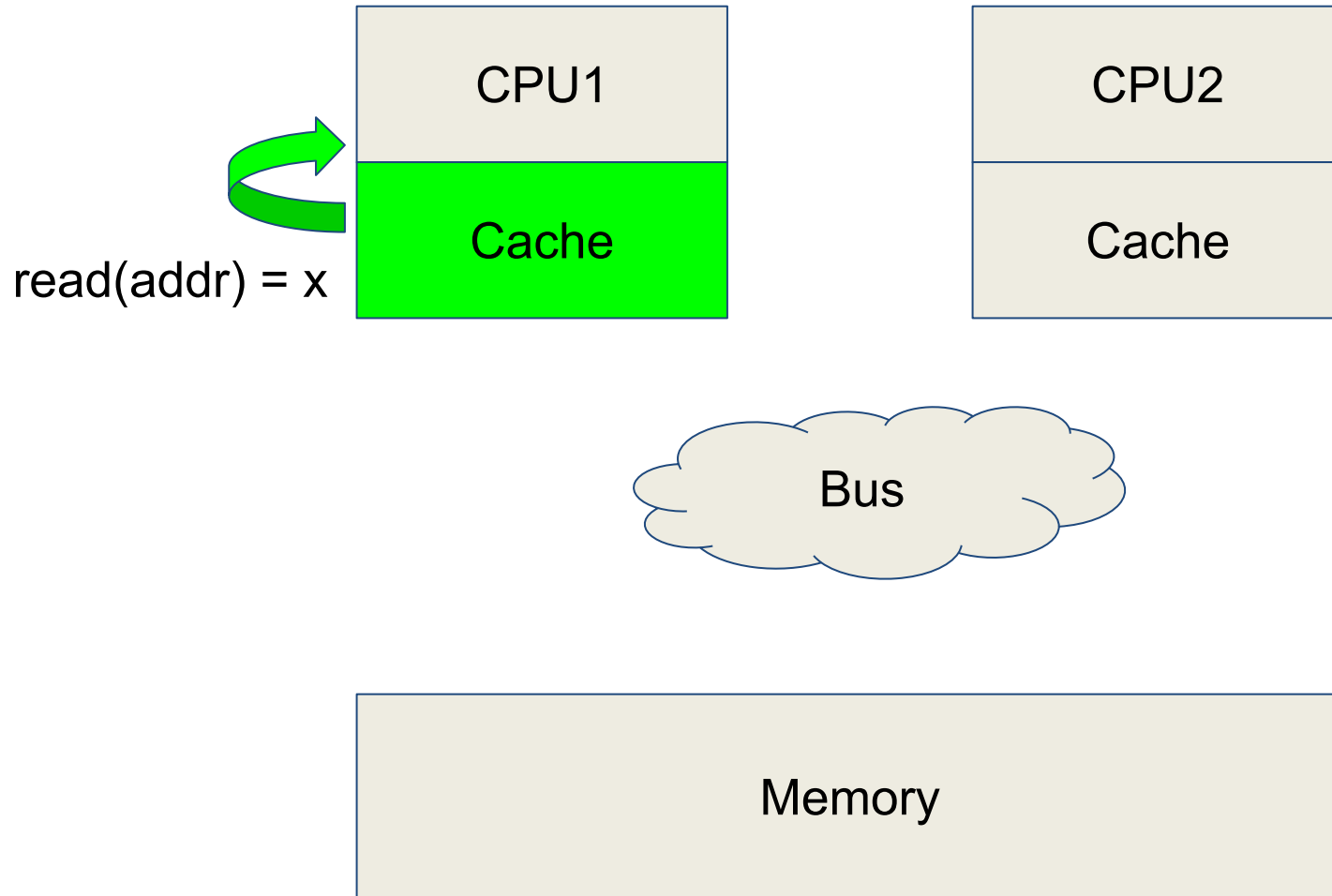
Кэш



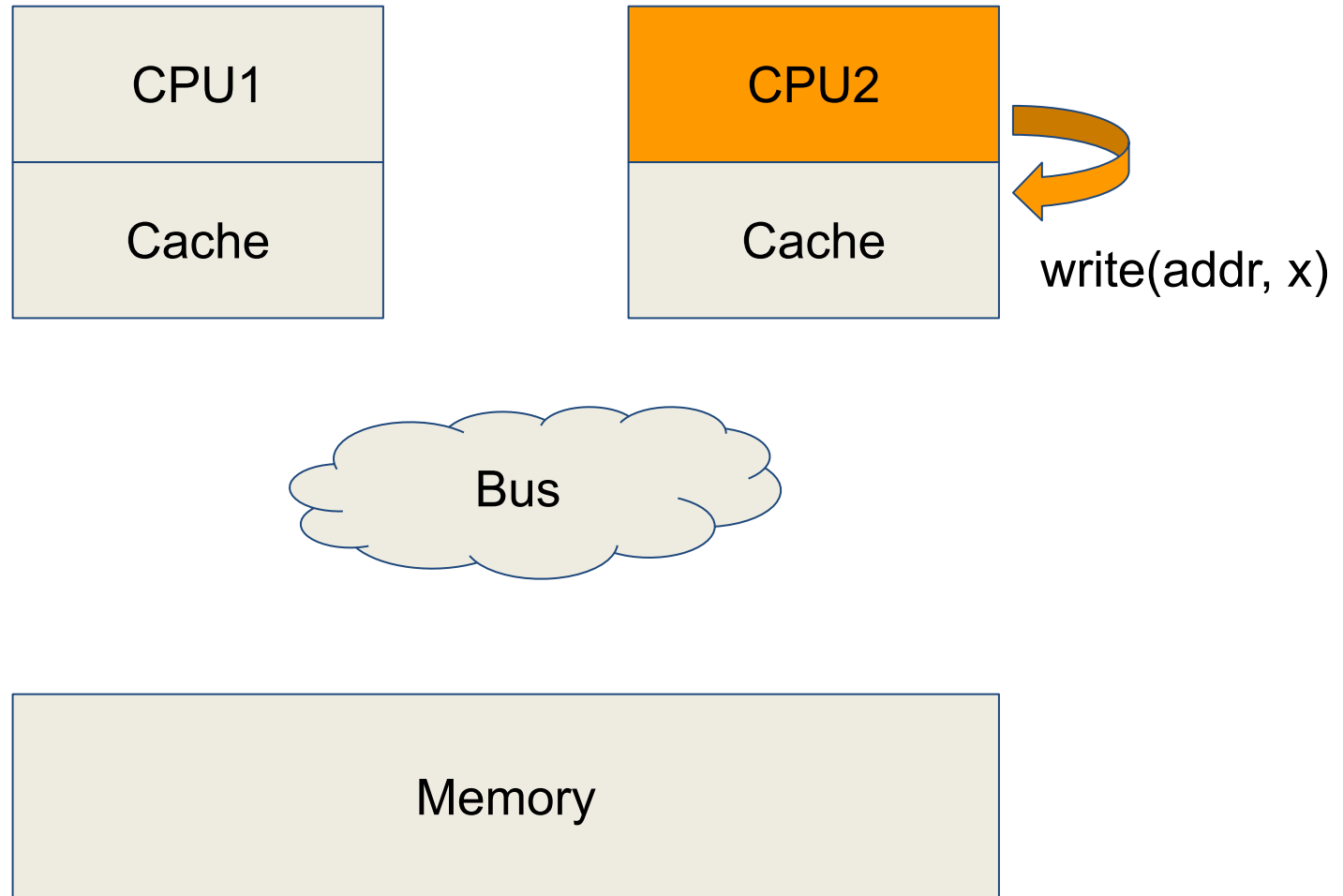
Кэш



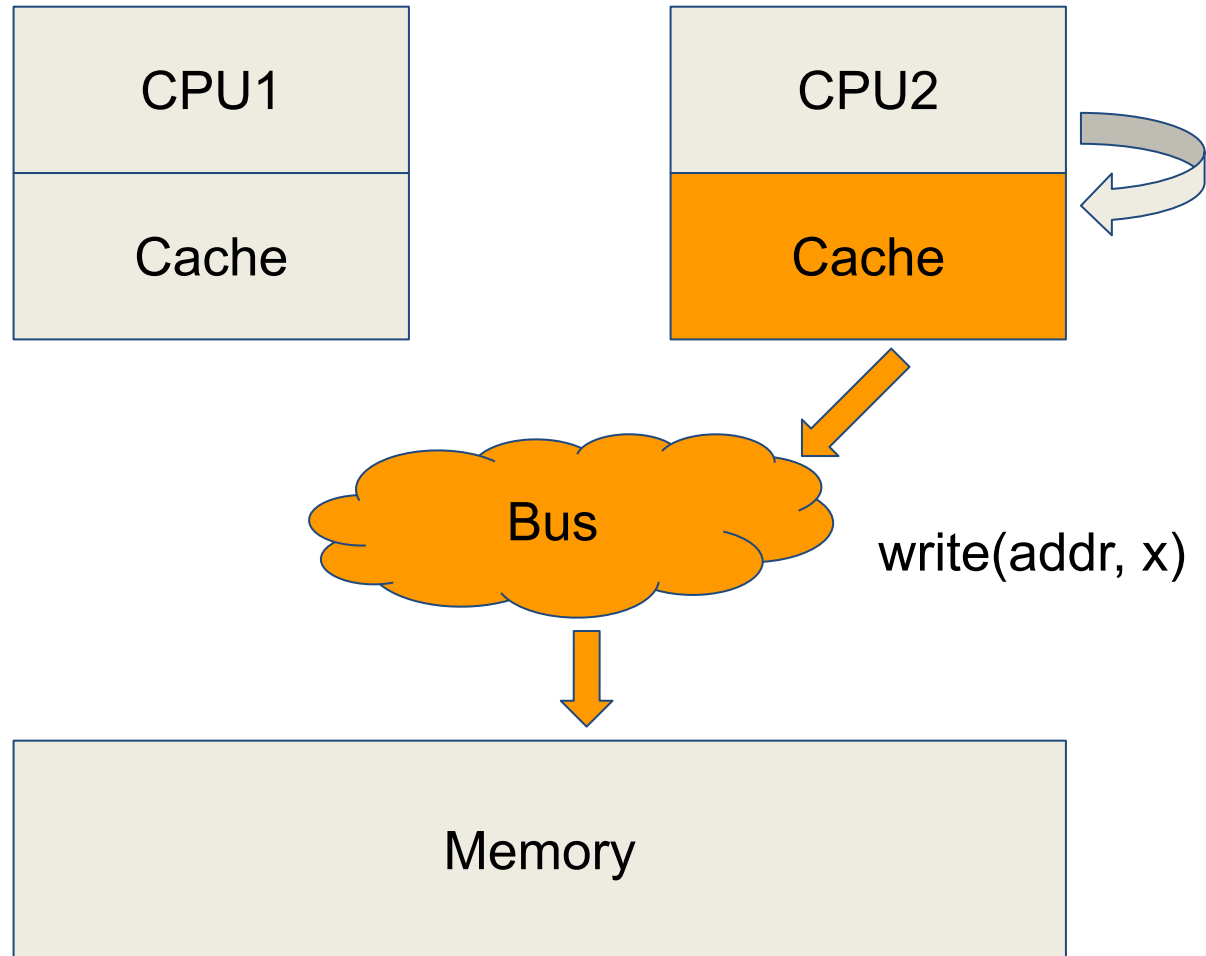
Кэш



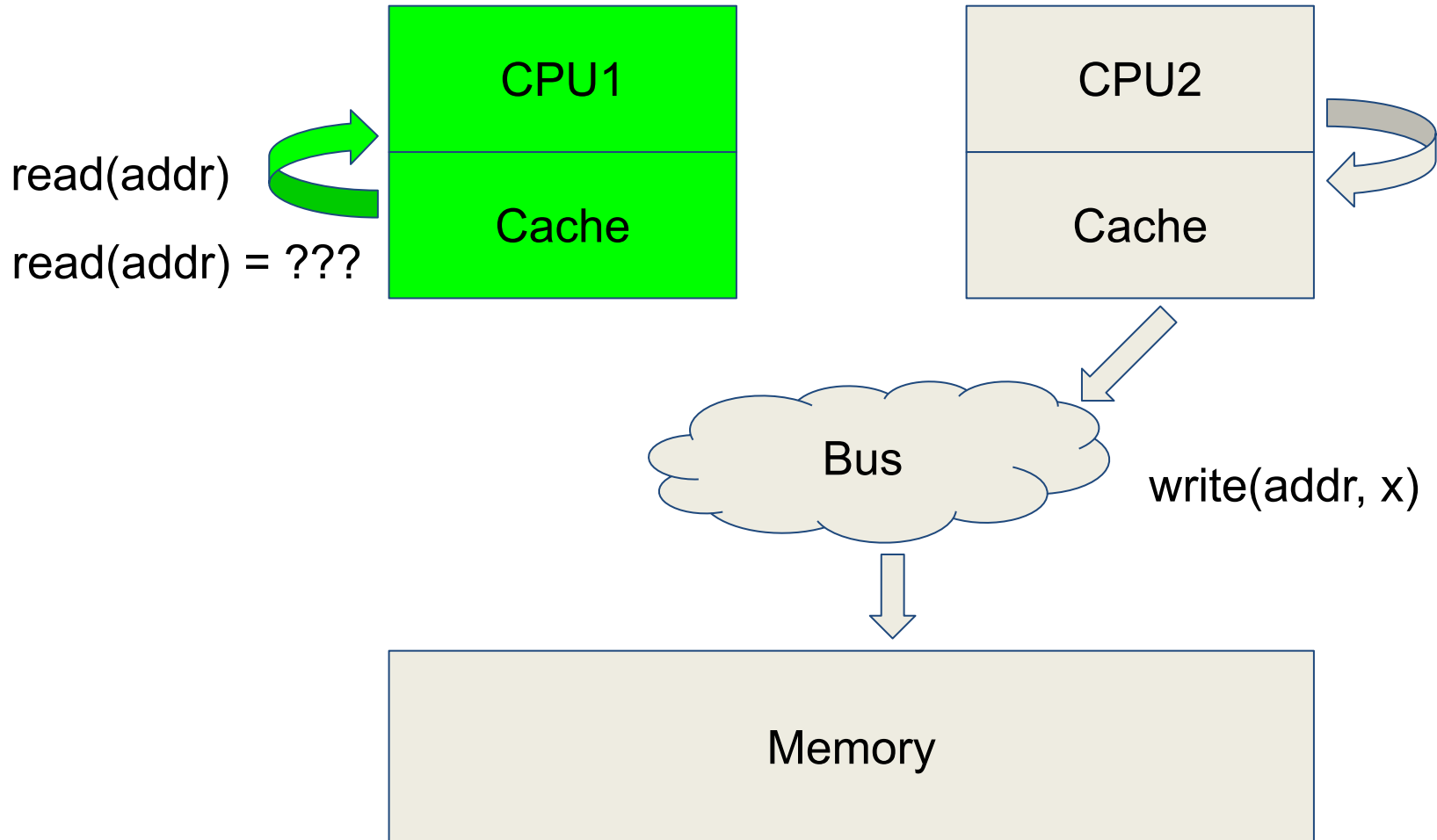
Когерентность кэша



Когерентность кэша



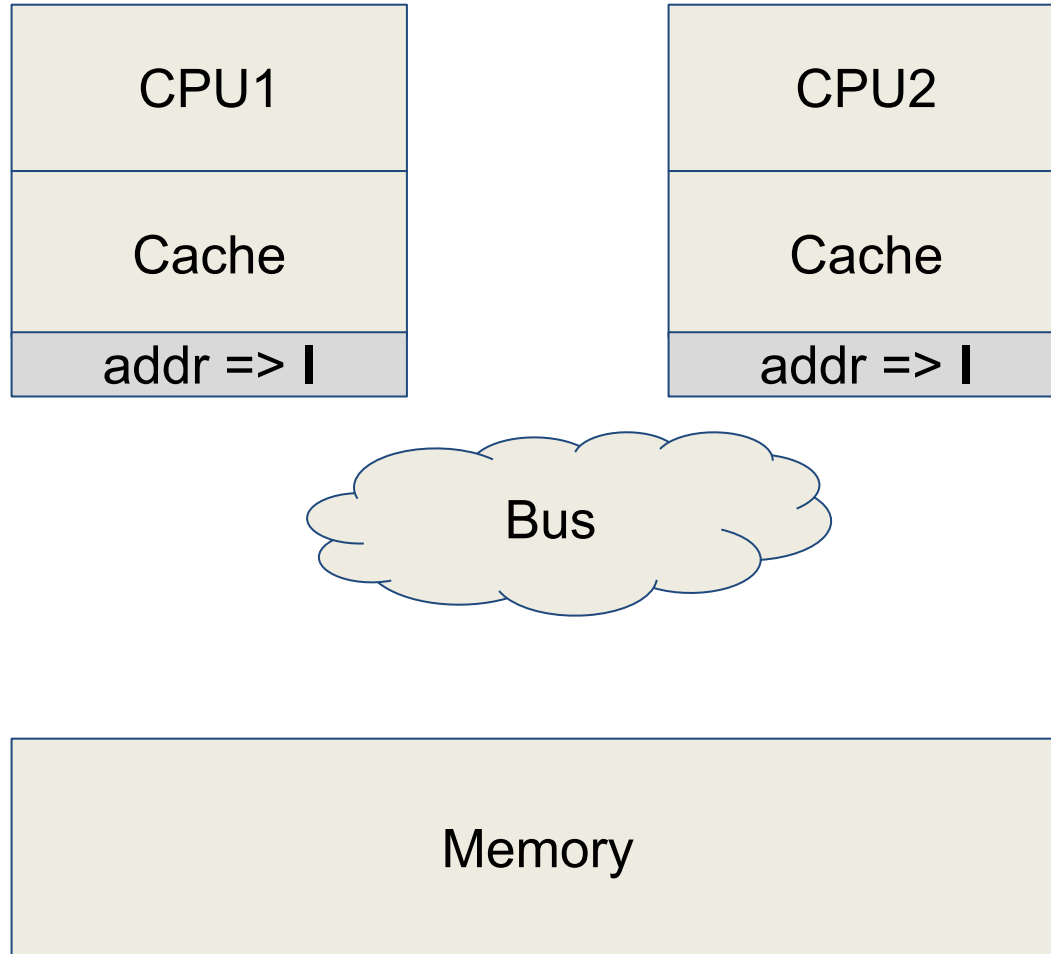
Когерентность кэша



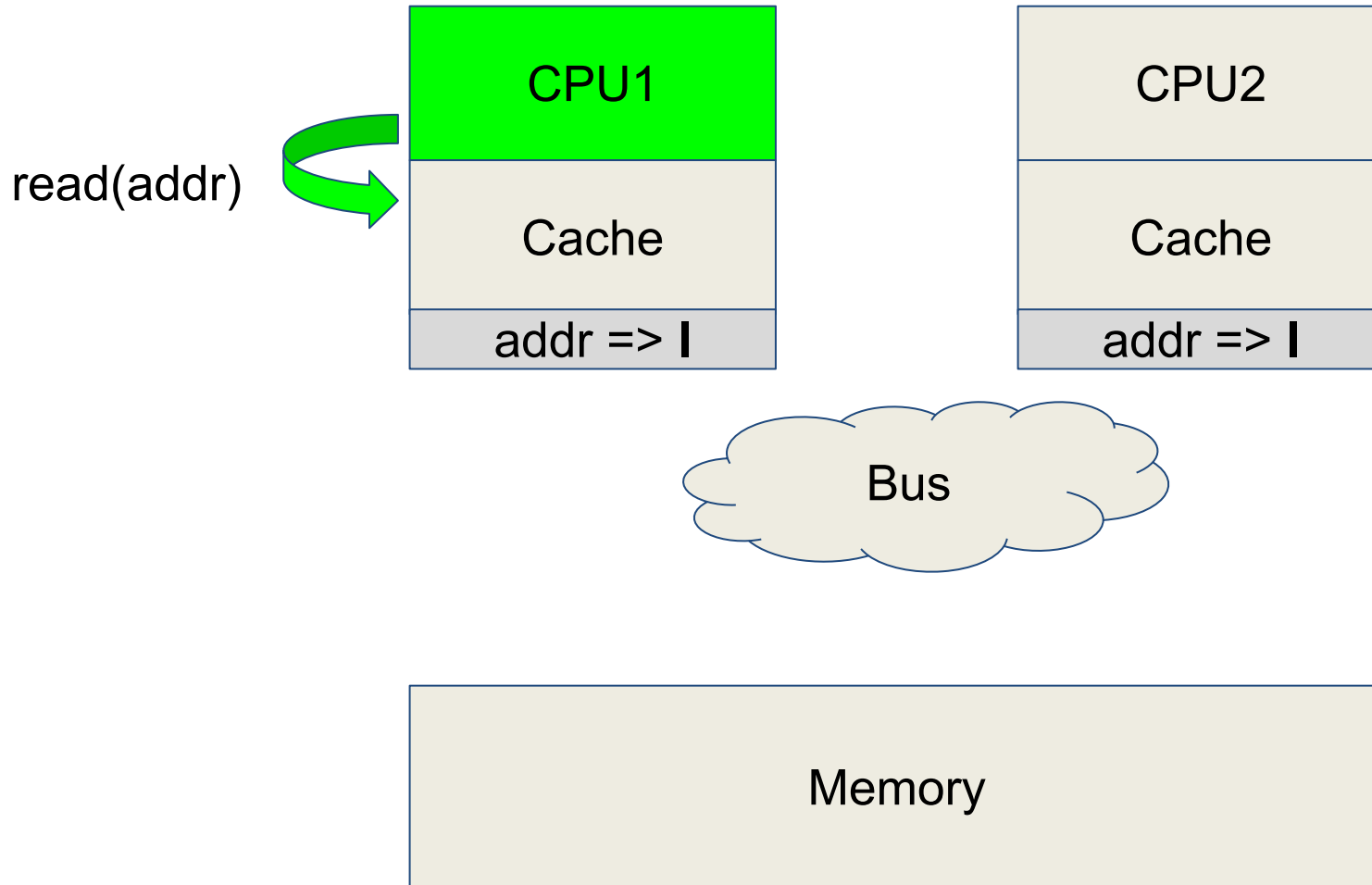
Протокол MESI

M	Modified
E	Exclusive
S	Shared
I	Invalid

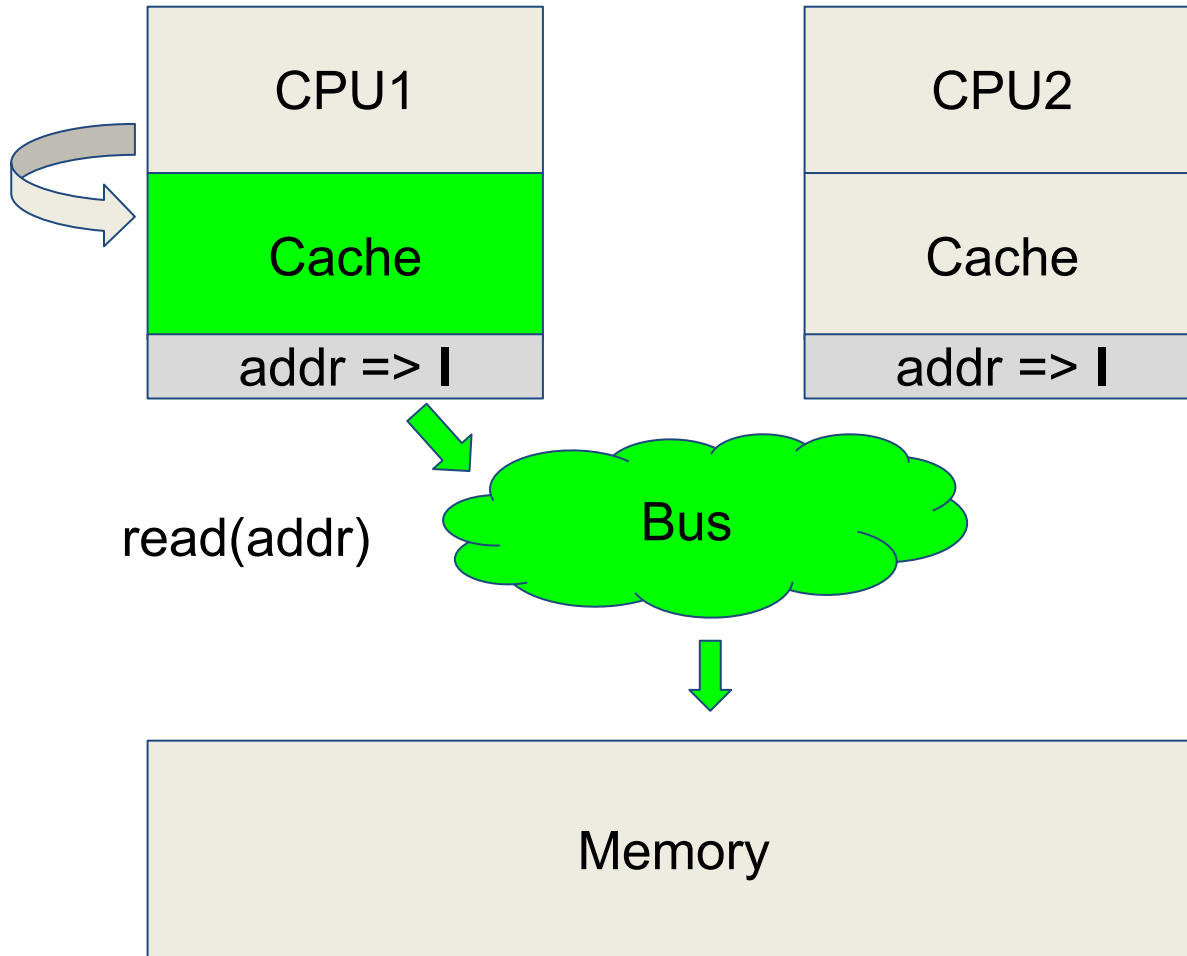
MESI наглядно



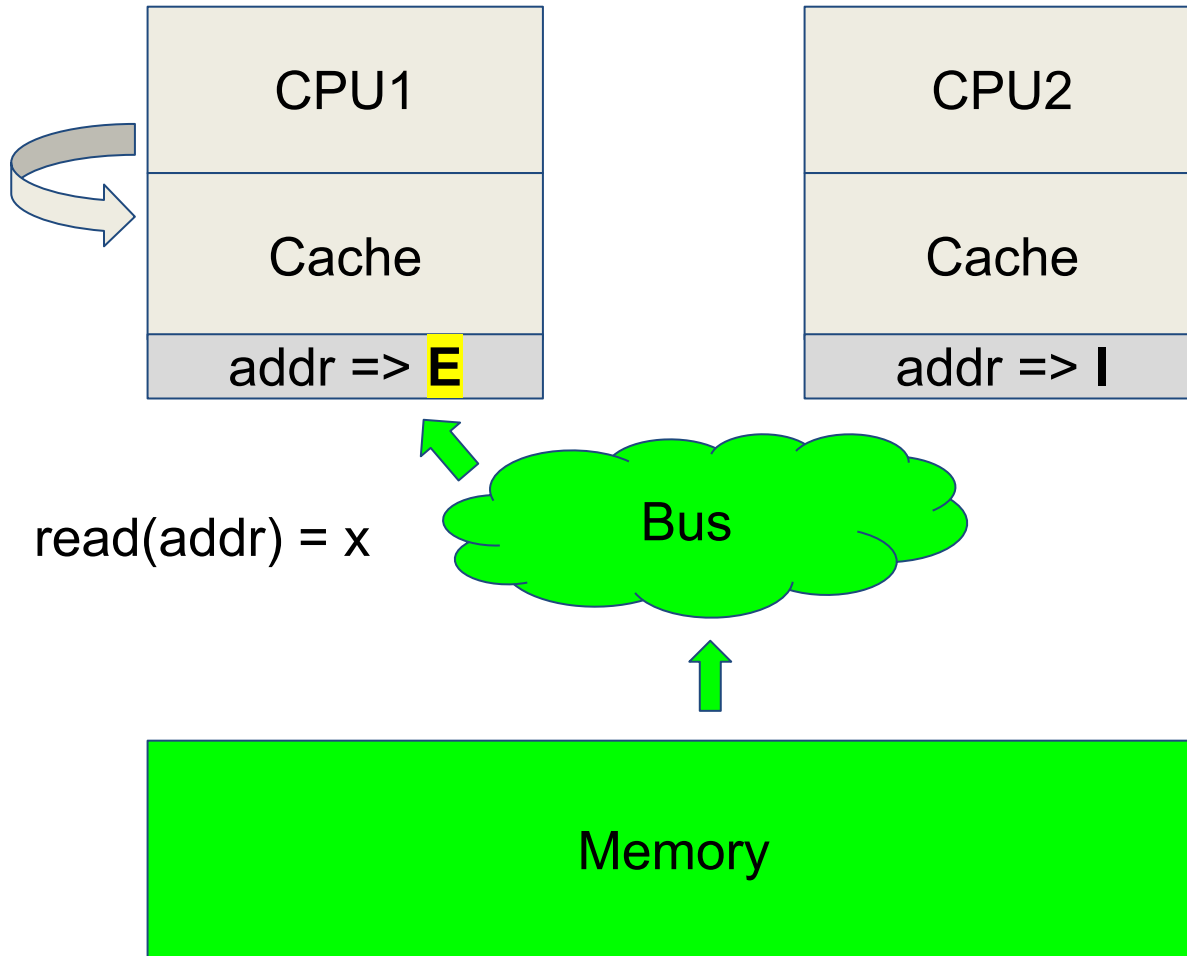
MESI наглядно



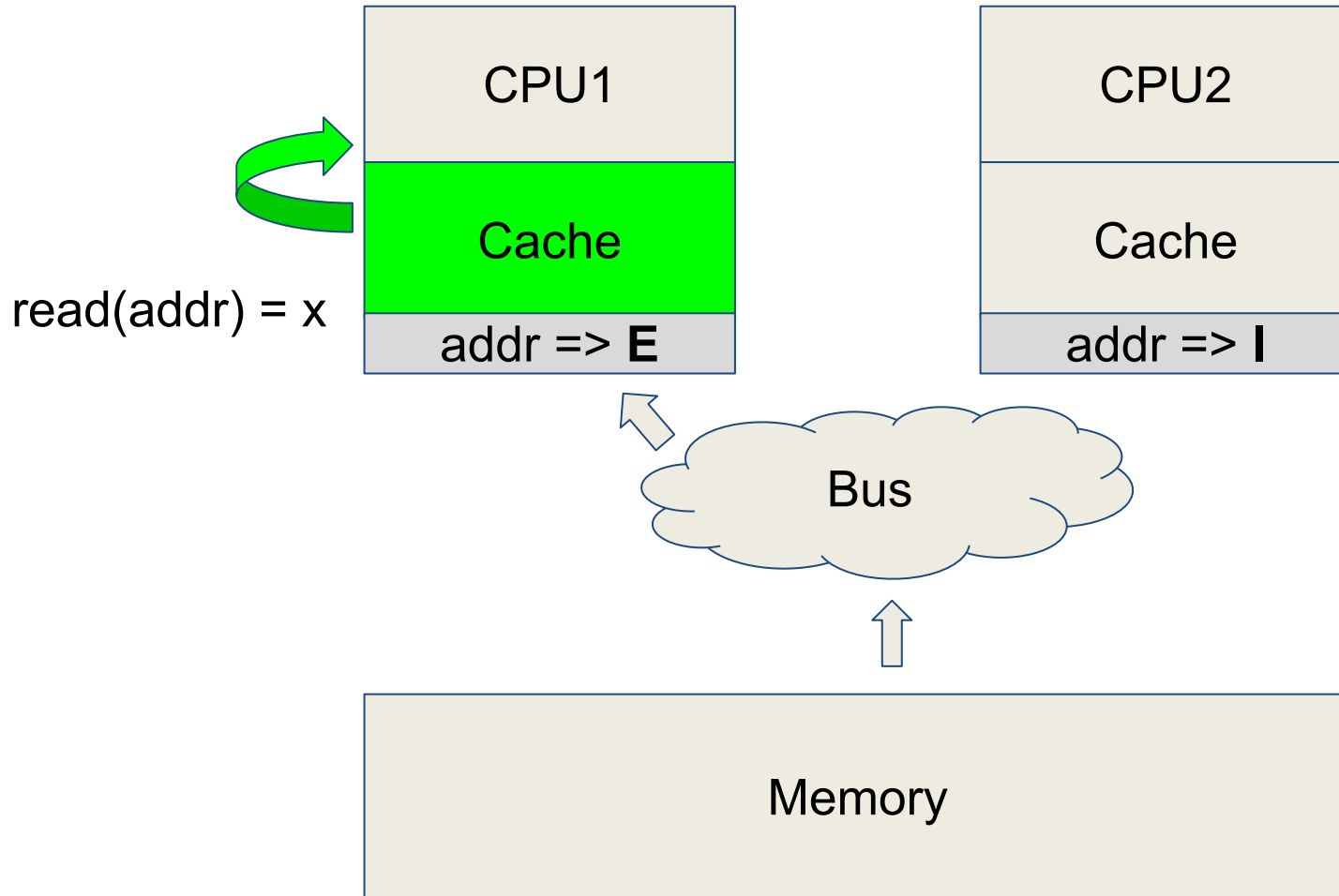
MESI наглядно



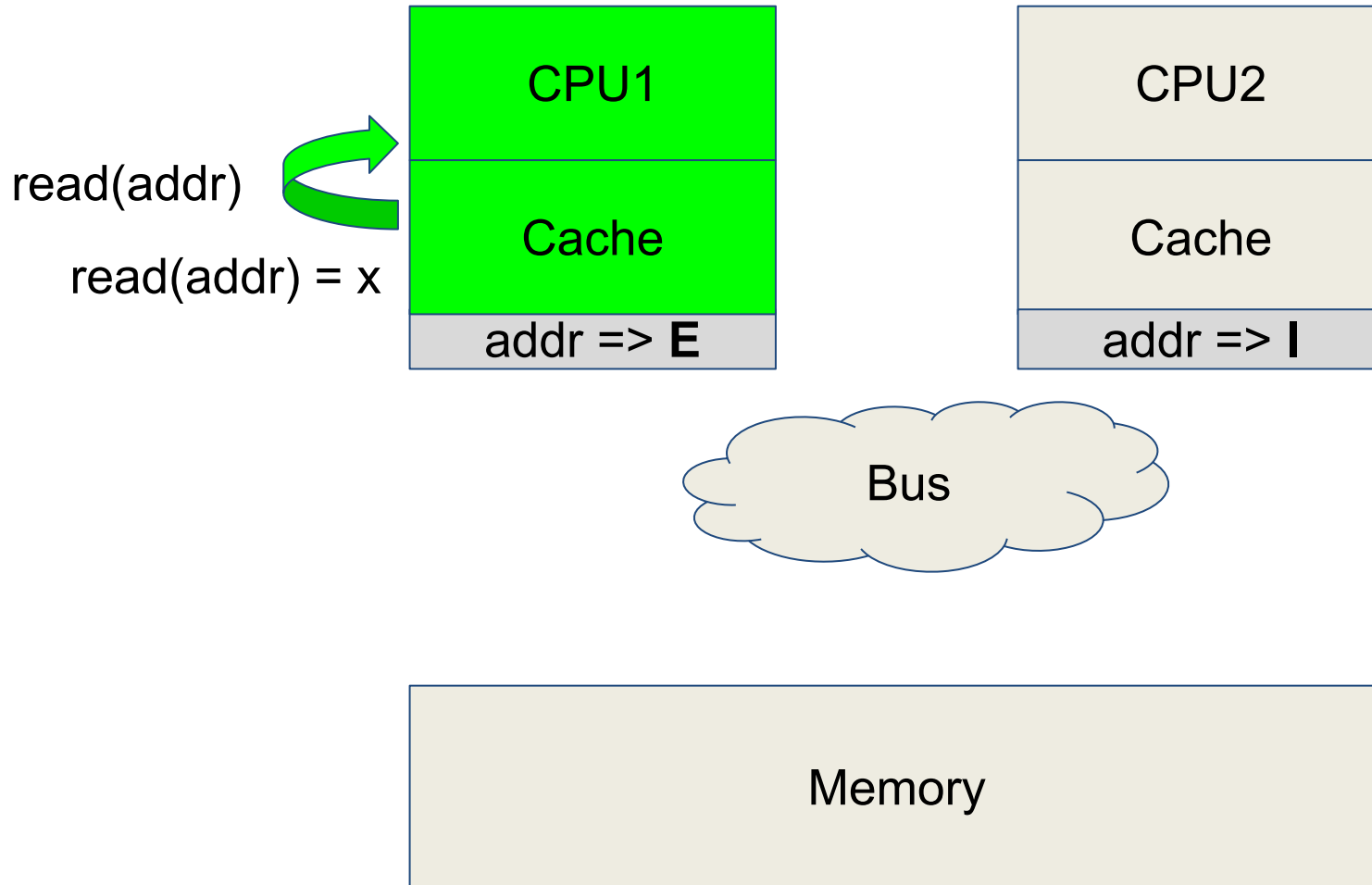
MESI наглядно



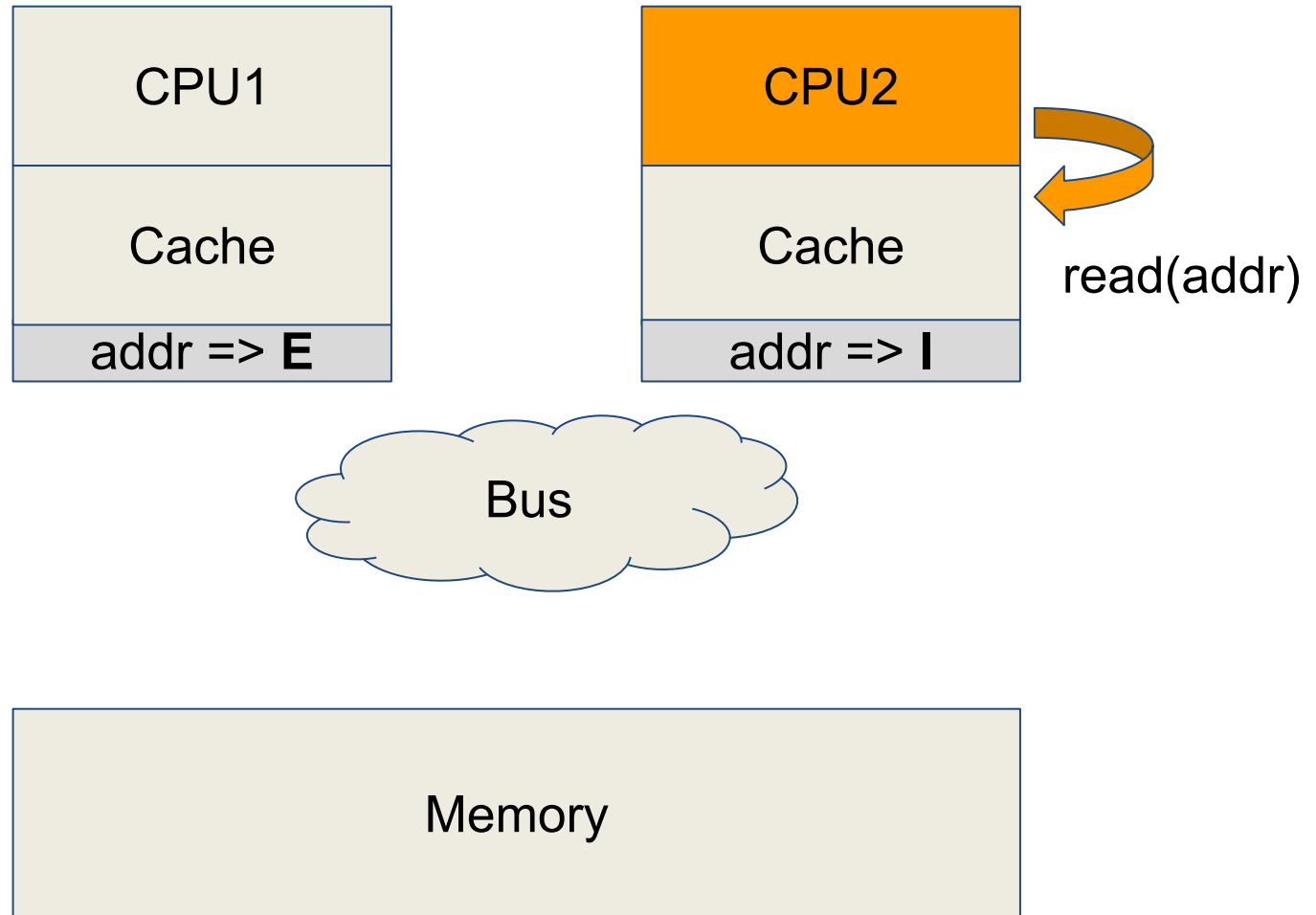
MESI наглядно



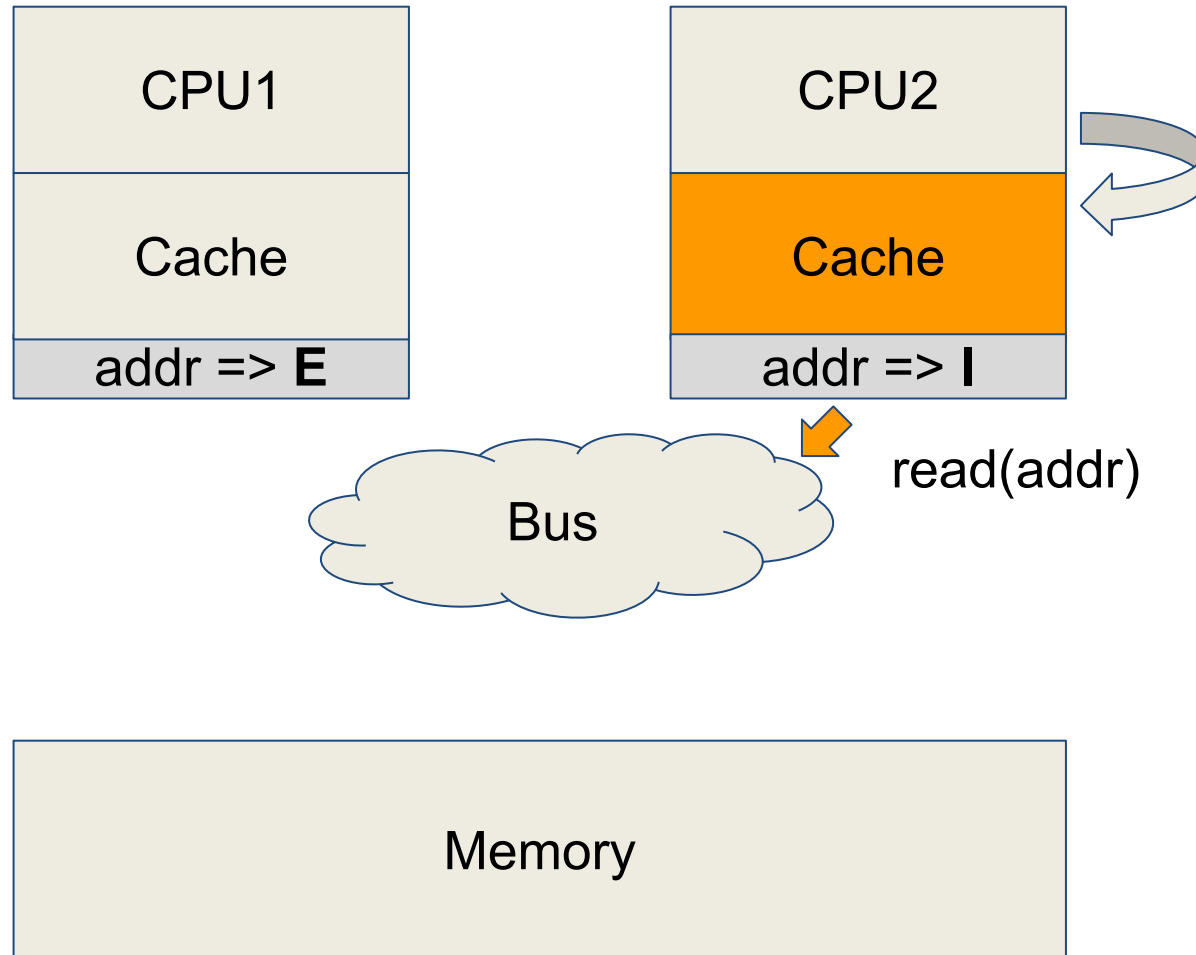
MESI наглядно



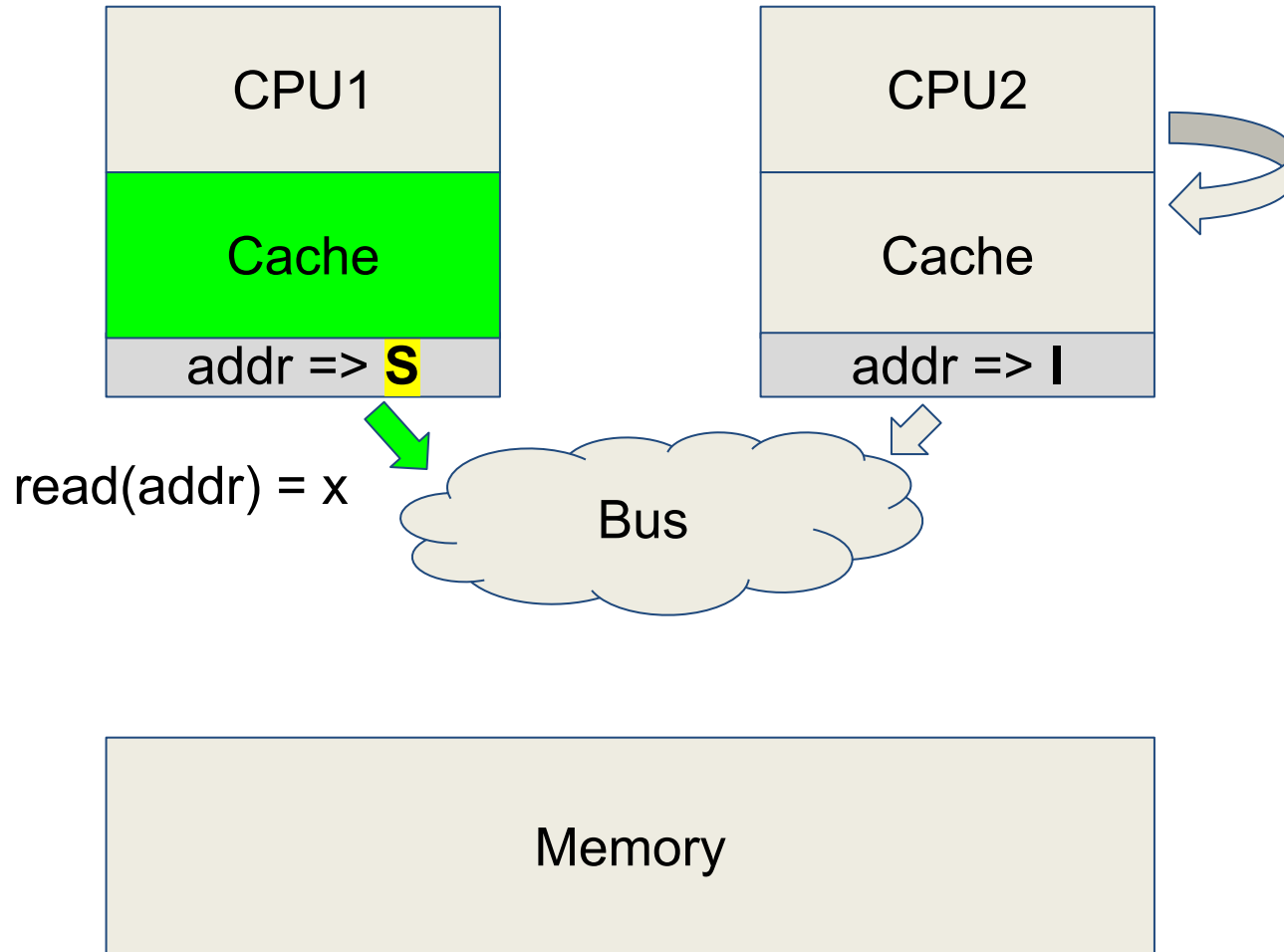
MESI наглядно



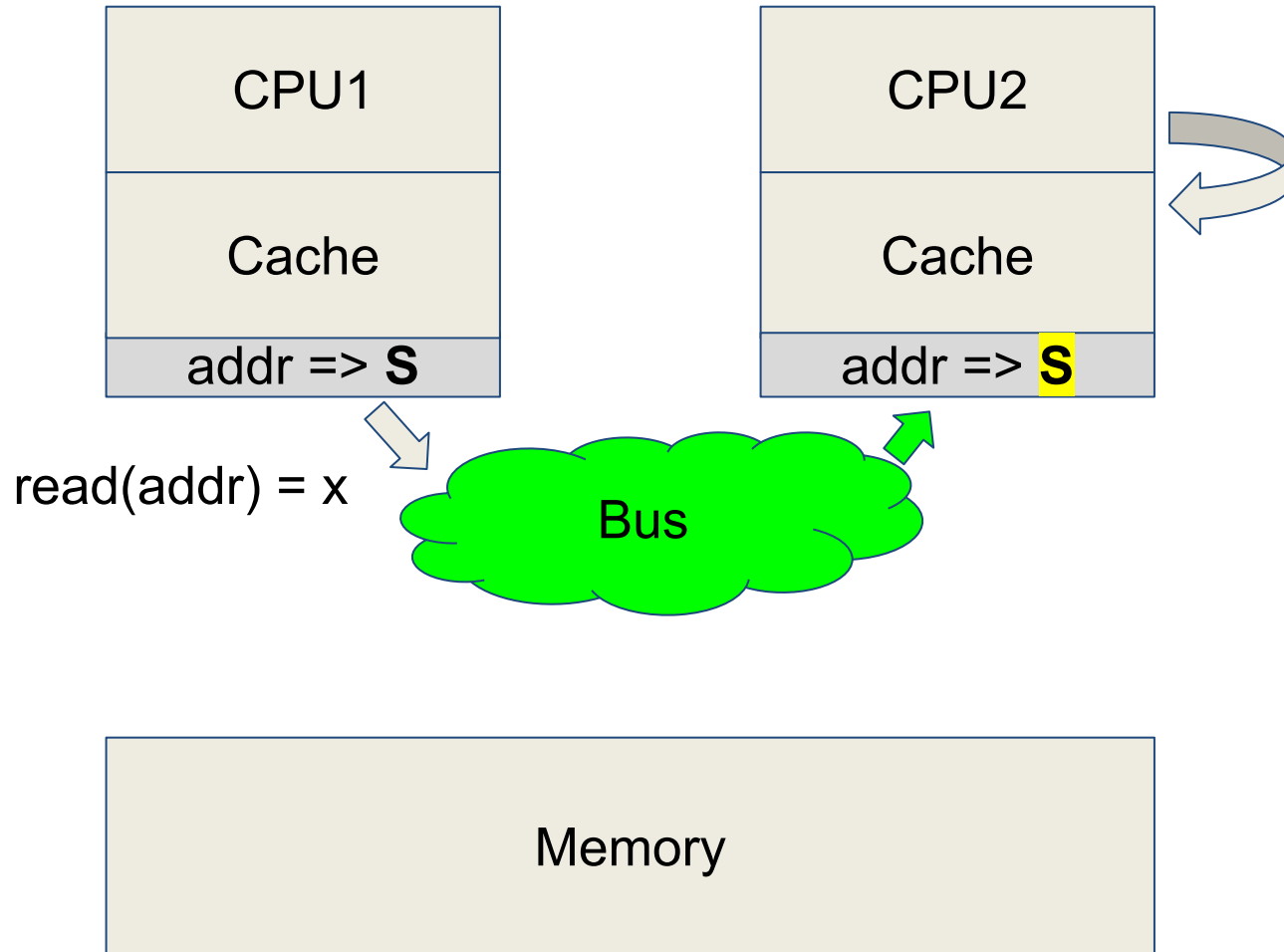
MESI наглядно



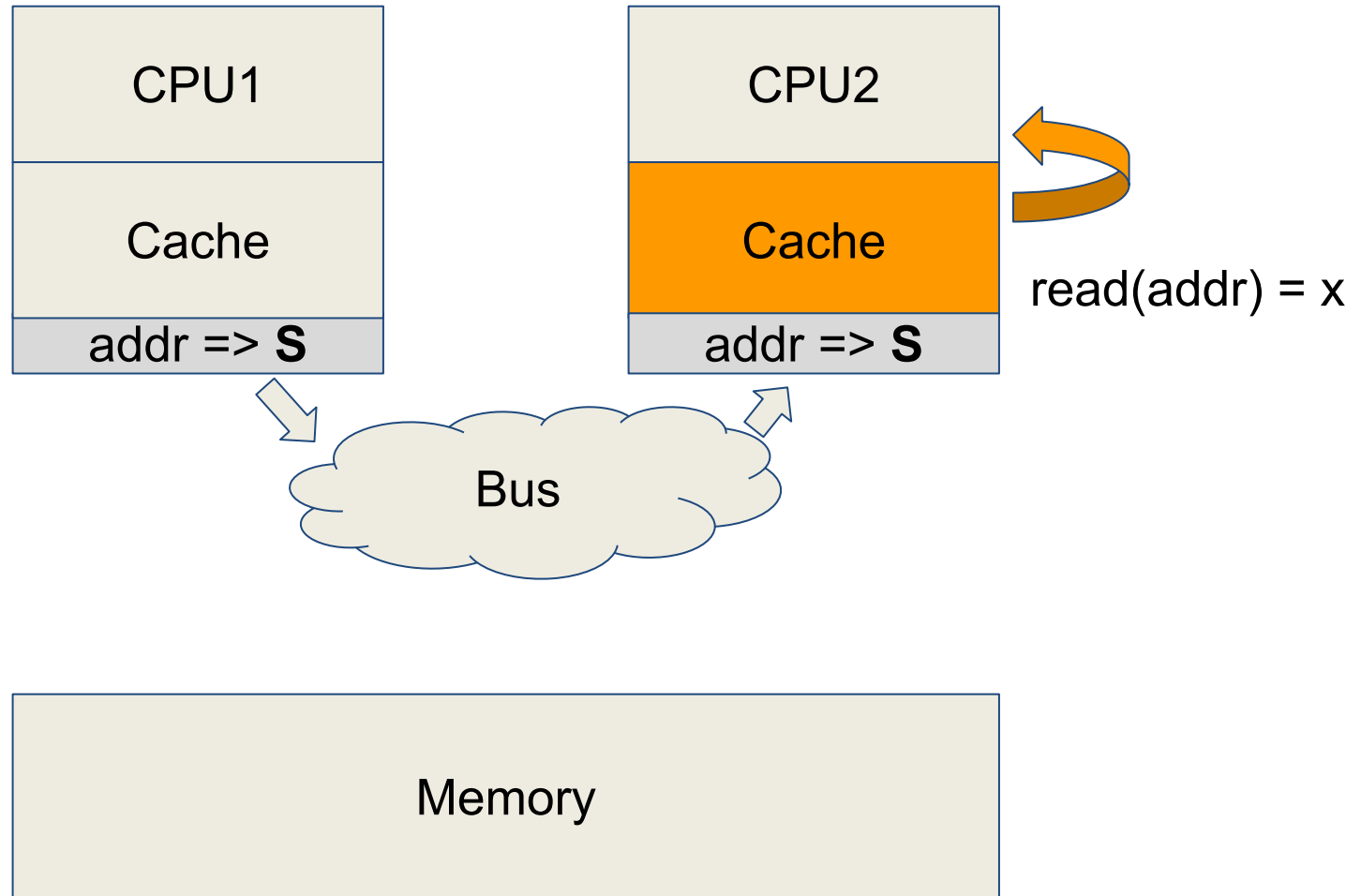
MESI наглядно



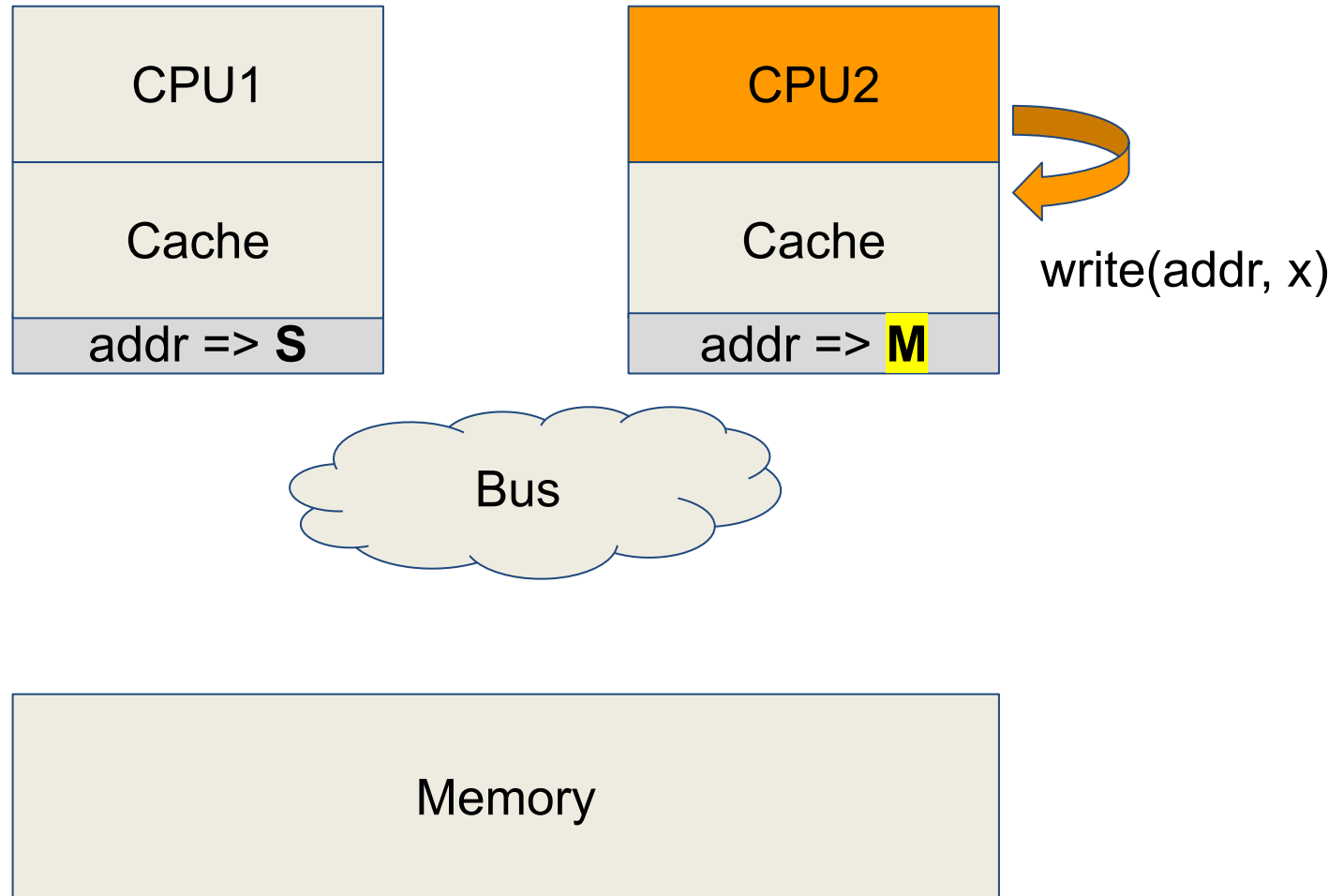
MESI наглядно



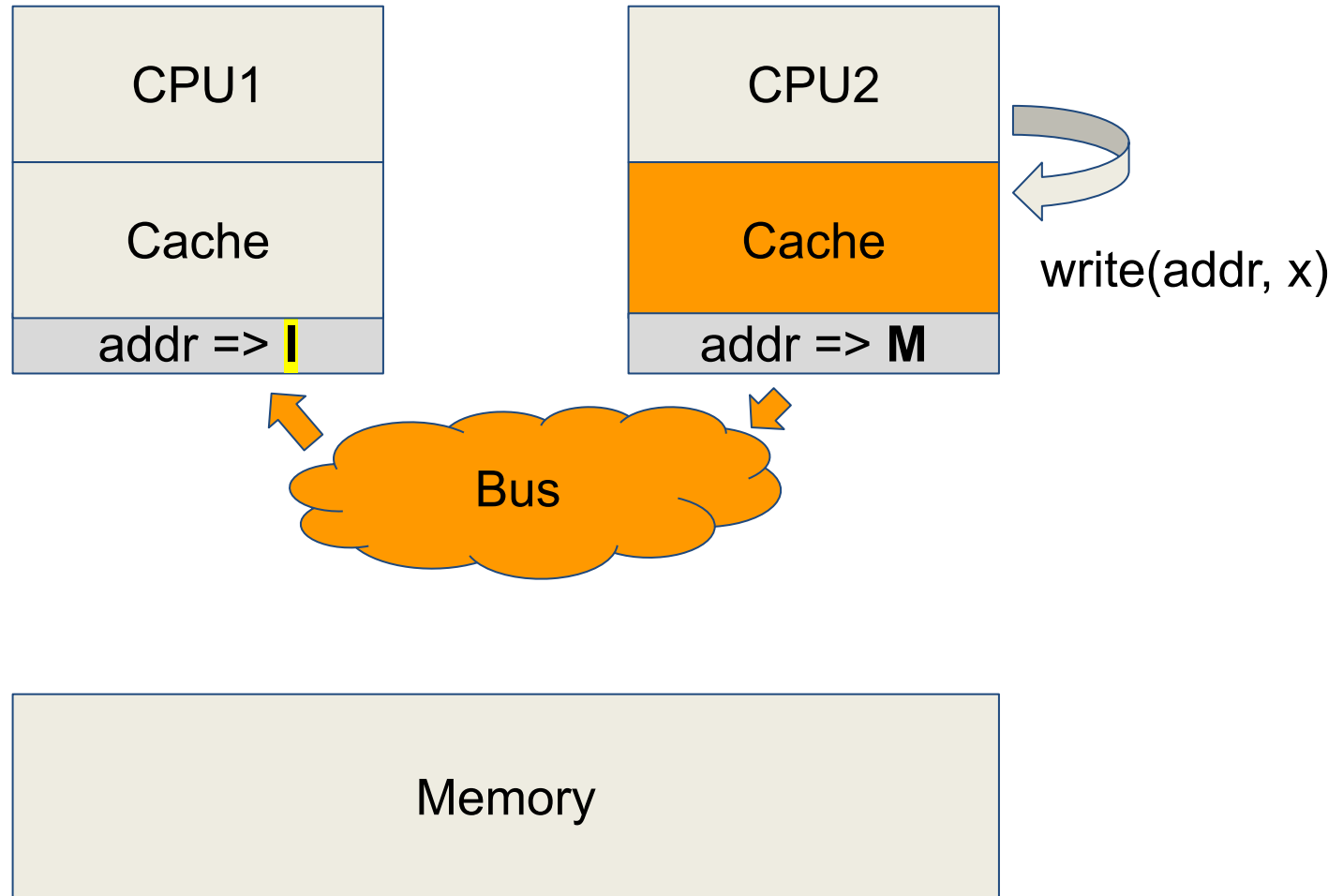
MESI наглядно



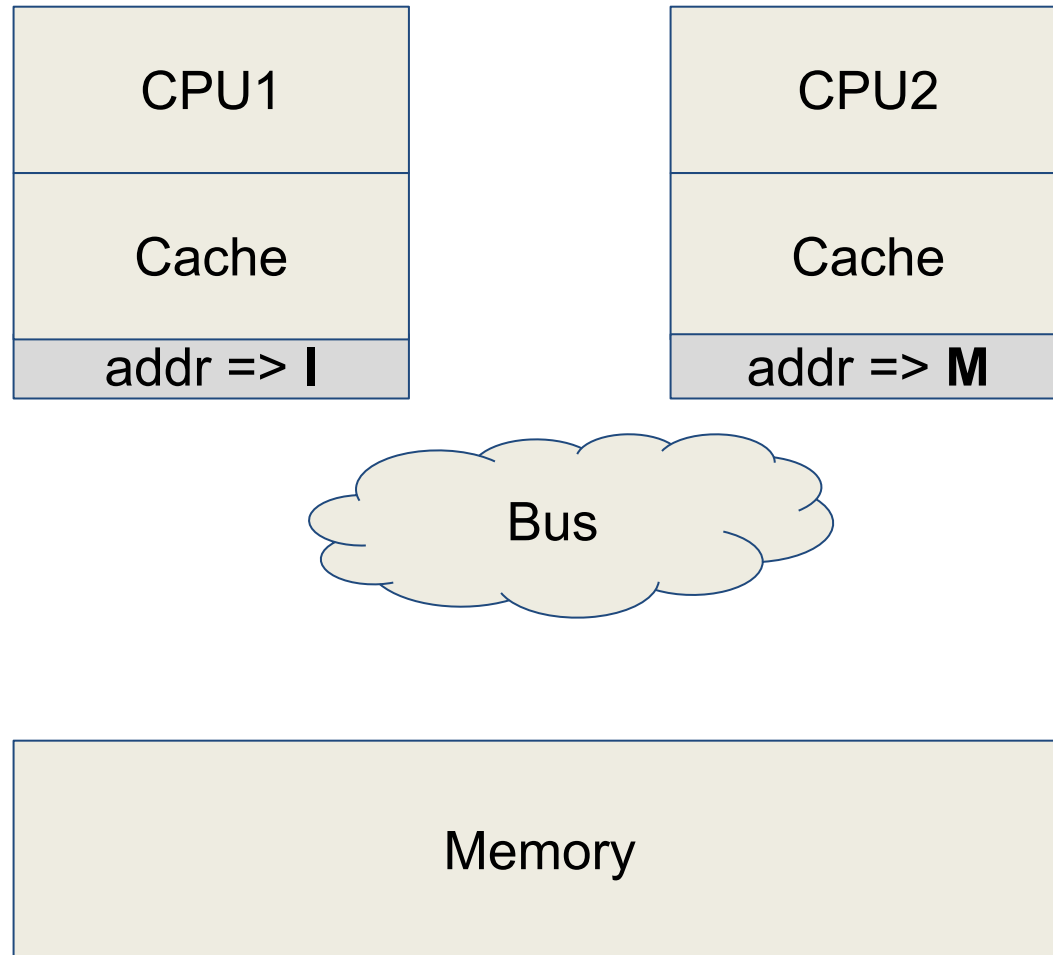
MESI наглядно



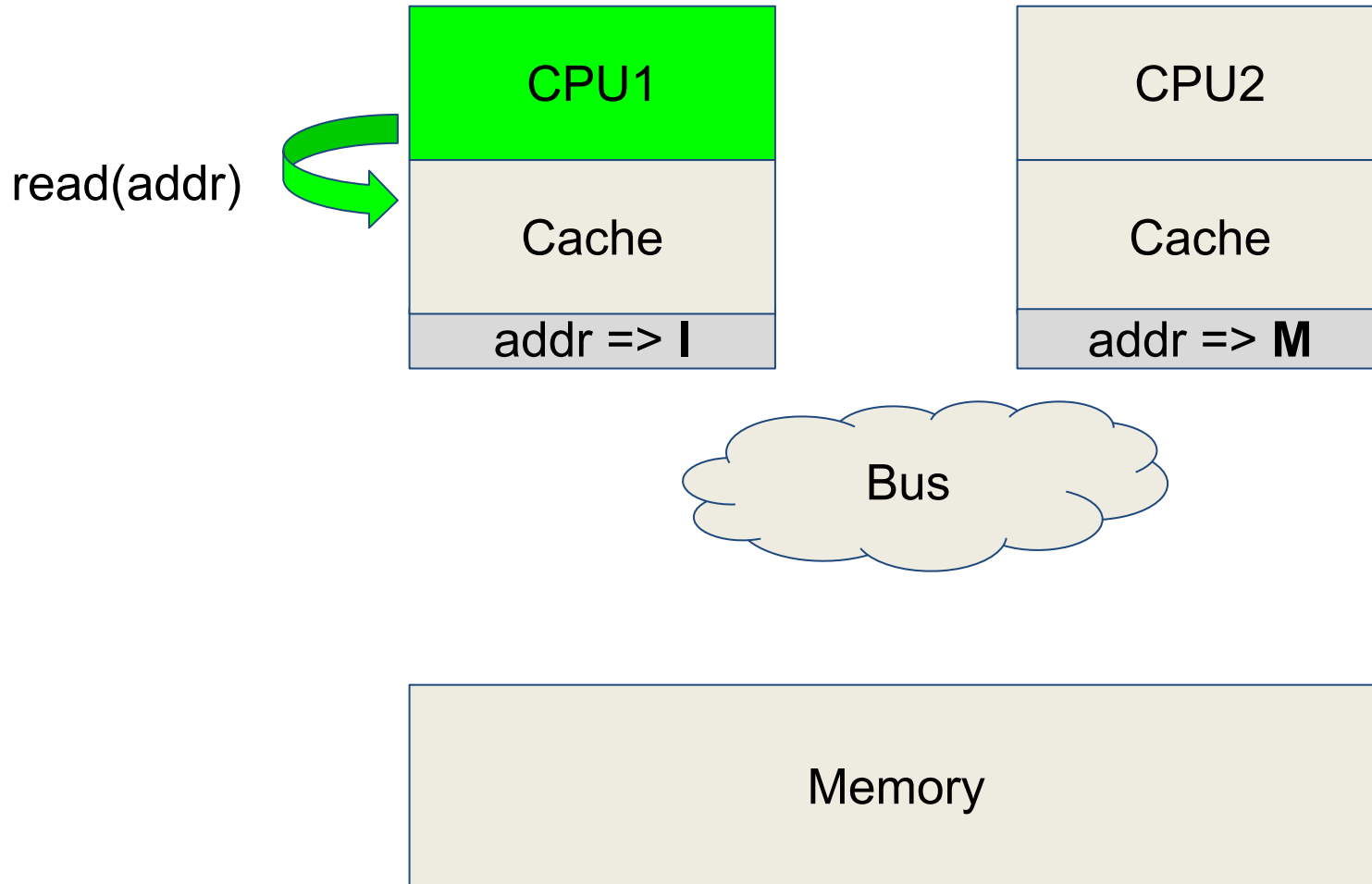
MESI наглядно



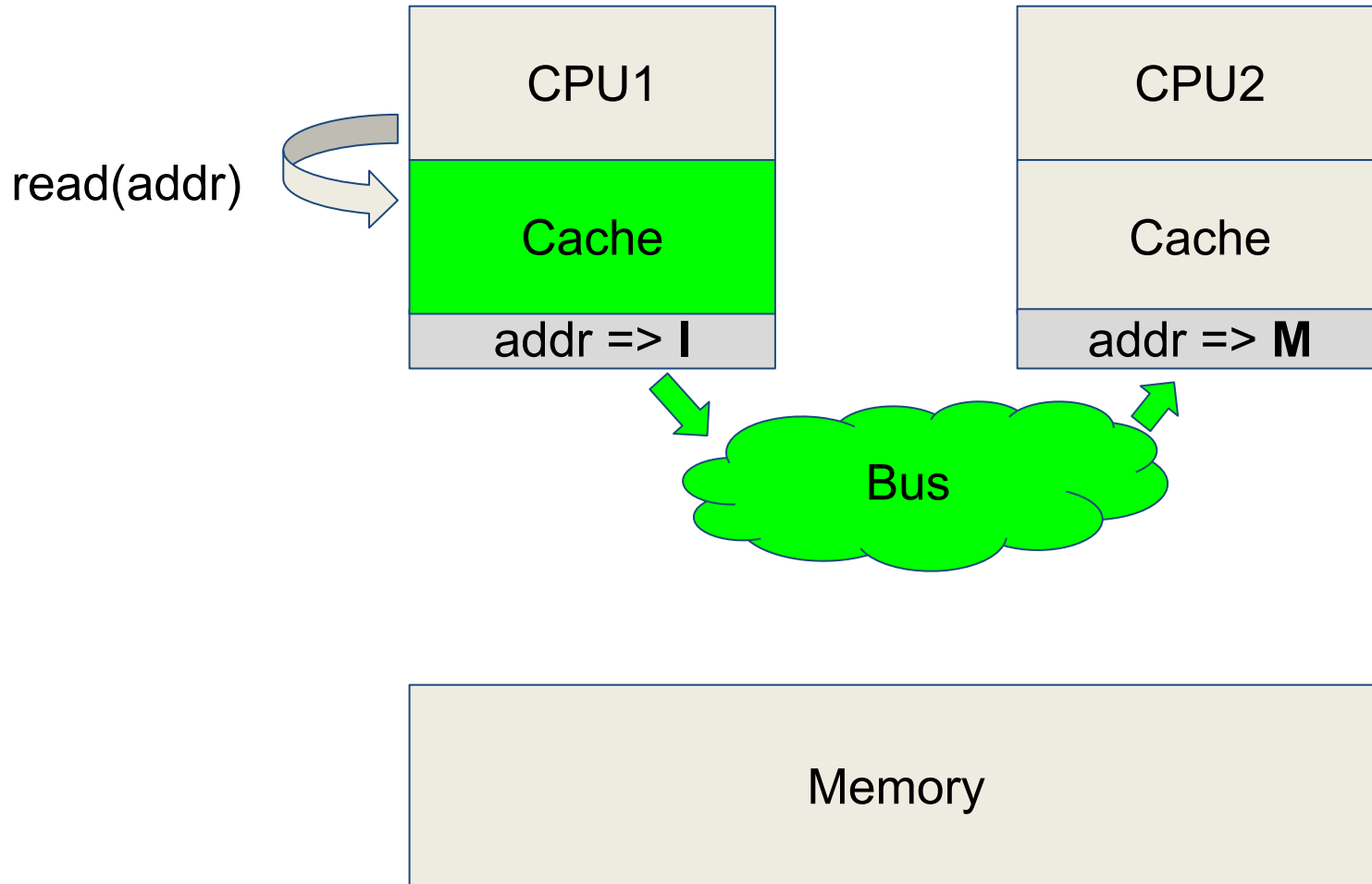
MESI наглядно



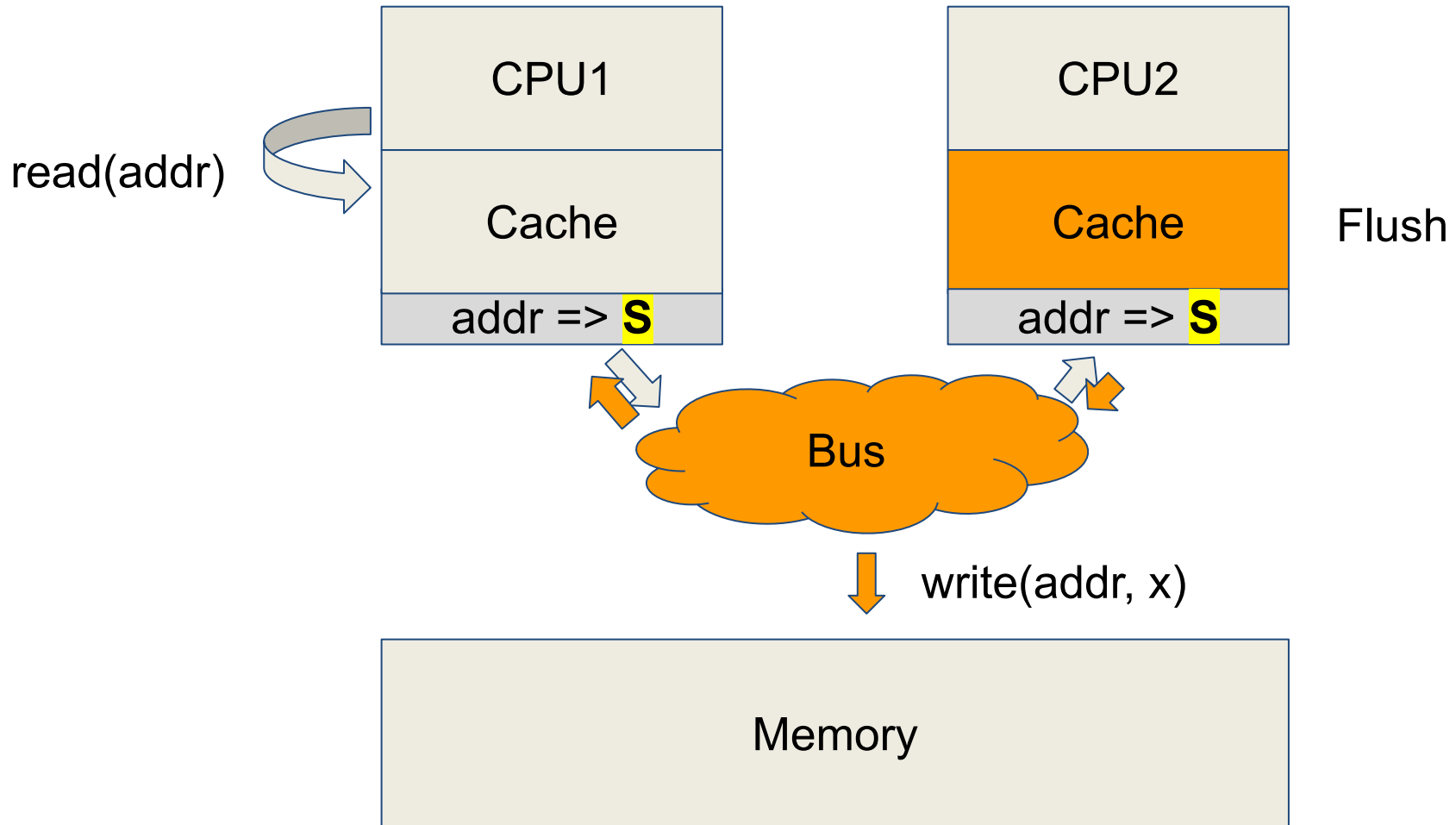
MESI наглядно



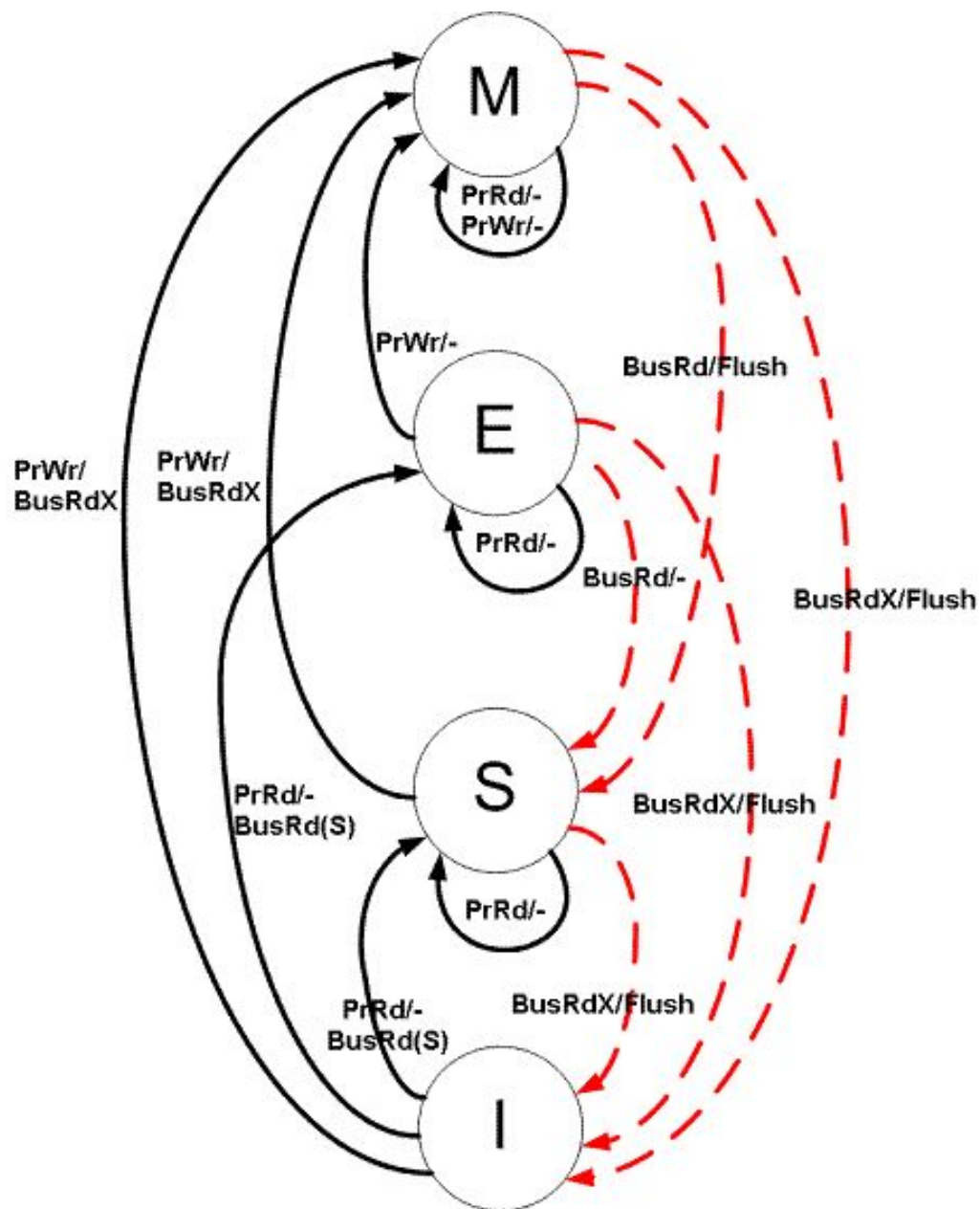
MESI наглядно



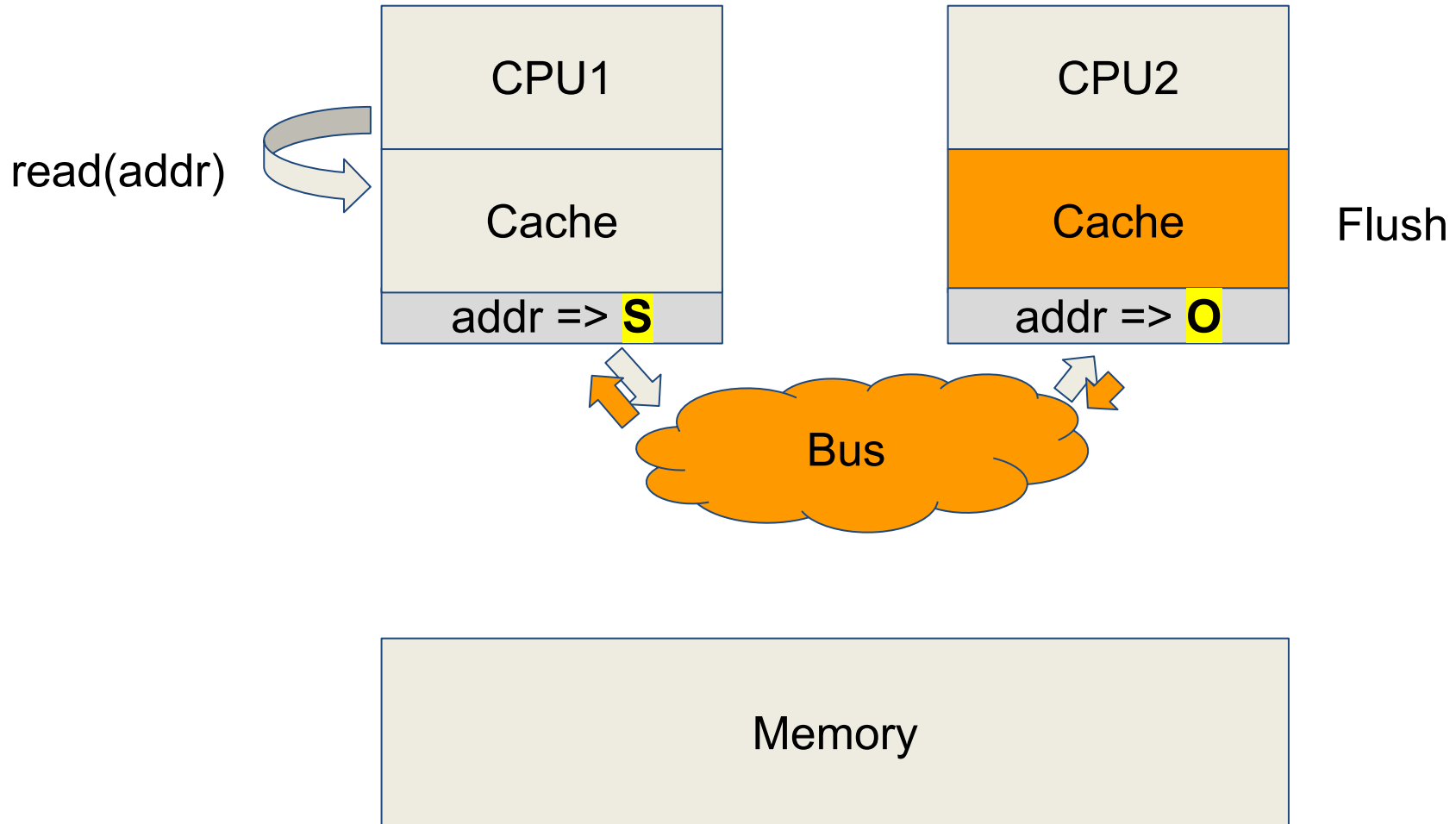
MESI наглядно



Весь MESI



MESI -> MOSI (O == Owned)



Больше ада

- **MESIF** = MESI + Forward (Intel)
- **MOESI** = MESI + Owned (AMD)

Test-And-Set (aka get-and-set)

// Последовательная спецификация

class TASRegister:

shared r

def getAndSet(x): **atomically do**

 old = r

 r = x

 return old

def read():

 return r

def write(x):

 r = x

Test-And-Set lock

```
class TASLock:  
    boolean locked  
  
    def lock():  
        while locked.getAndSet(true):  
            pass  
  
    def unlock():  
        locked = false
```

Test-And-Set lock

```
class TASLock:  
    boolean locked
```

```
    def lock():  
        while locked.getAndSet(true):  
            pass
```

```
    def unlock():  
        locked = false
```

**Сброс всех кешей
(Exclusive!)**

Test-And-Test-And-Set lock

```
class TTSLock:
    boolean locked

    def lock():
        loop:
            while locked: pass
            if !locked.getAndSet(true):
                break

    def unlock():
        locked = false
```

Test-And-Test-And-Set lock

```
class TTSLock:  
    boolean locked
```

```
    def lock():  
        loop:  
            while locked: pass  
            if !locked.getAndSet(true):  
                break
```

```
    def unlock():  
        locked = false
```

Крутится без шума
на шине (Shared)

Test-And-Test-And-Set lock

```
class TTSLock:
    boolean locked

    def lock():
        loop:
            while locked: pass
            if !locked.getAndSet(true):
                break

    def unlock():
        locked = false
```

Крутится без шума
на шине (Shared)

Invalidation storm

Test-And-Test-And-Set lock

```
class TTSLock:  
    boolean locked  
  
    def lock():  
        loop:  
            while locked: pass  
            if !locked.getAndSet(true):  
                break  
  
    def unlock():  
        locked = false
```

Все ломается сюда

Invalidation storm

Backoff

```
class TTSTimeoutLock:
```

```
    boolean locked
```

```
    def lock():
```

```
        loop:
```

```
            while locked: pass
```

```
            if !locked.getAndSet(true):
```

```
                break
```

```
                delay()
```

```
    def unlock():
```

```
        locked = false
```

**Подождать при
неудаче**

Backoff

```
class TTSTimeoutLock:
```

```
    boolean locked
```

```
    def lock():
```

```
        loop:
```

```
            while locked: pass
```

```
            if !locked.getAndSet(true):
```

```
                break
```

```
                delay()
```

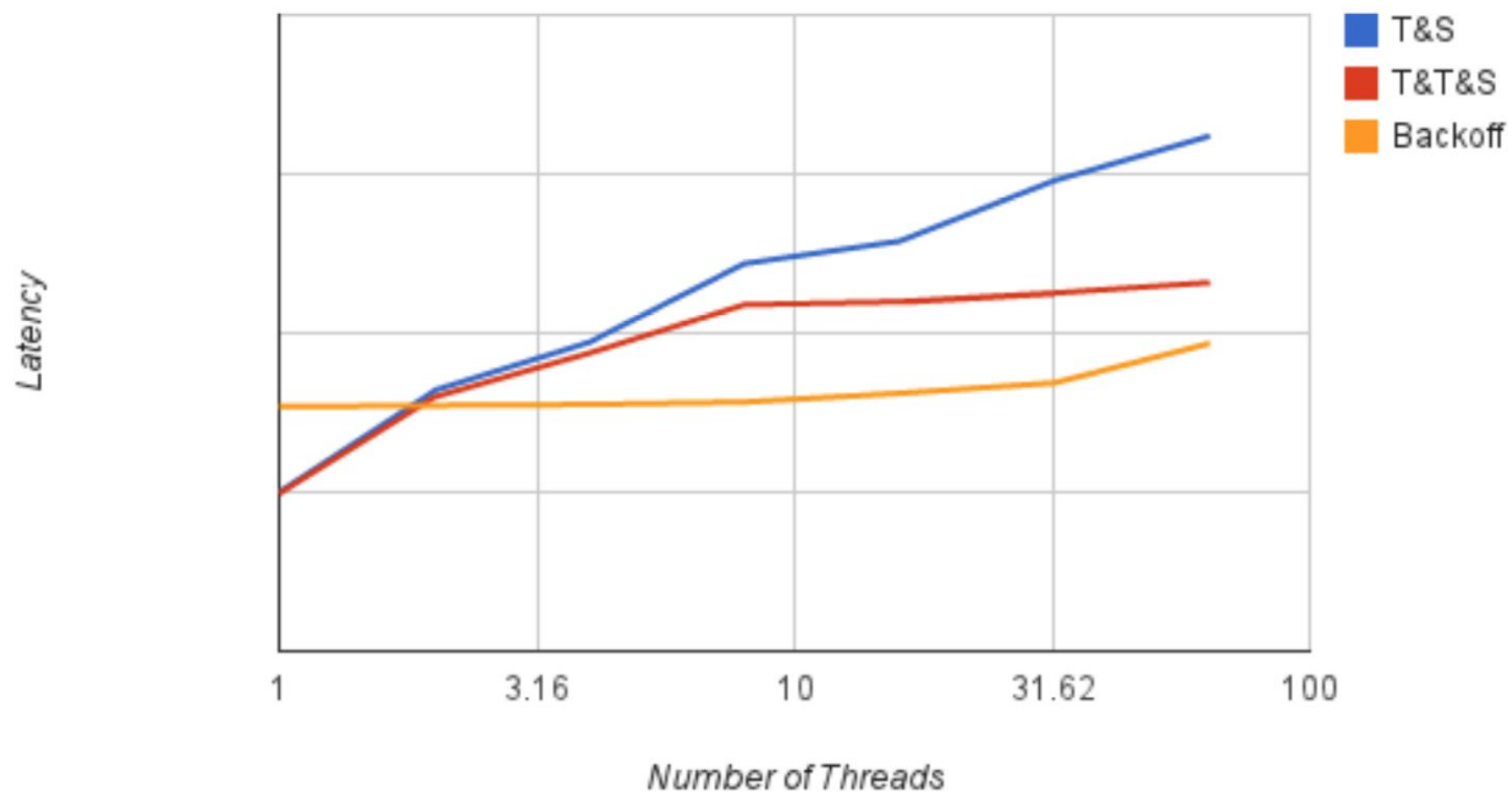
```
    def unlock():
```

```
        locked = false
```

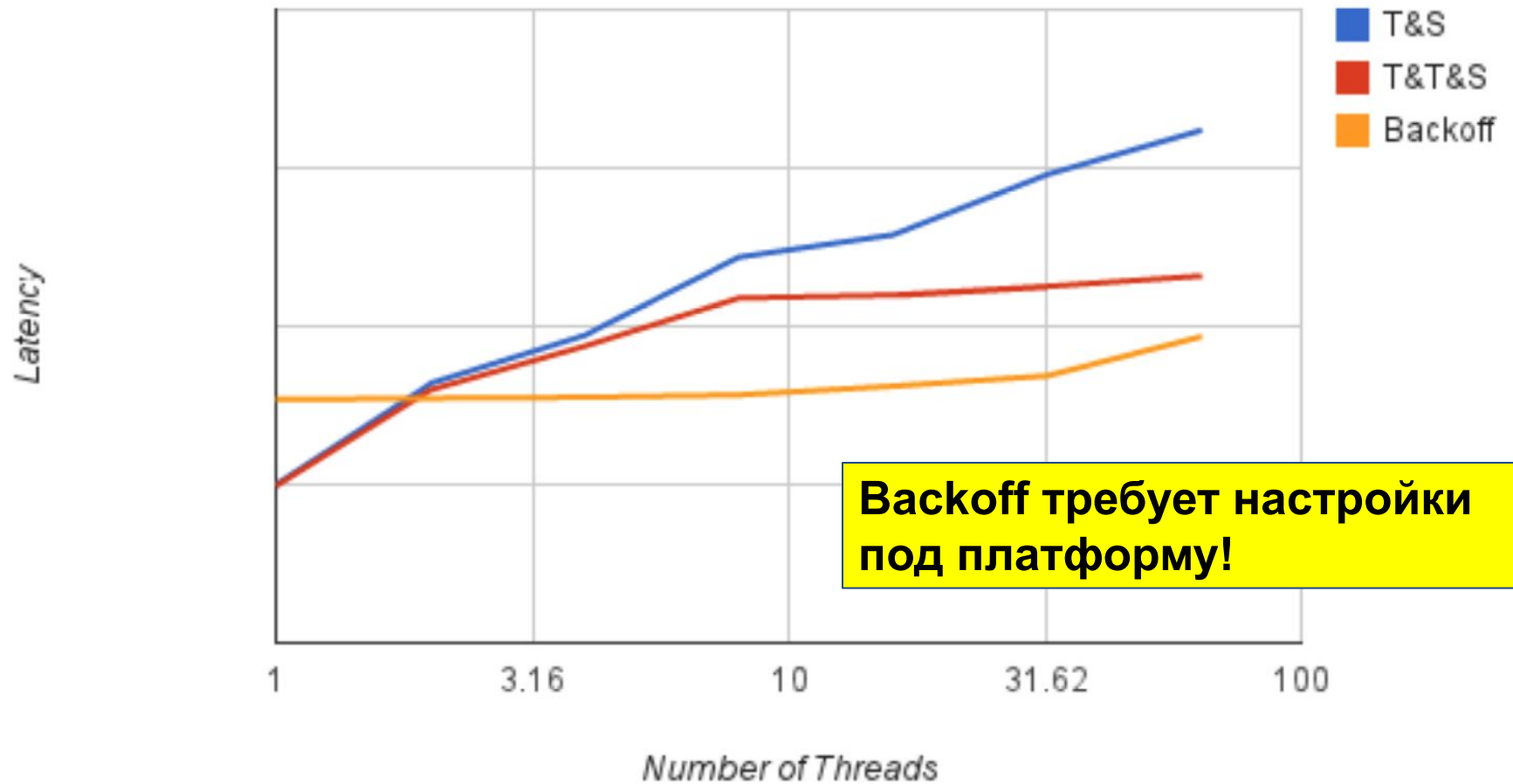
**Подождать при
неудаче**

1. Случайное время
2. Экспоненциально увеличивать

Lock Scalability - Latency



Lock Scalability - Latency



CLH Lock

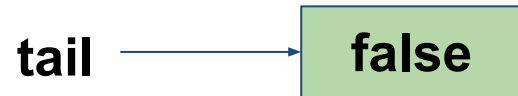
- Travis **C**raig, Anders **L**andin, Erik **H**agersten
- Устраним лишние инвалидации
- Храним очередь ждущих потоков
- First-Come First-Served

CLH Lock

```
class QNode:  
    boolean locked // shared, atomic
```

```
class CLHLock:  
    tail = QNode() // shared, atomic
```

CLH Lock: Начало: Не занято



Не занято

CLH Lock: Поток 1

Thread 1

Поток хочет lock

tail



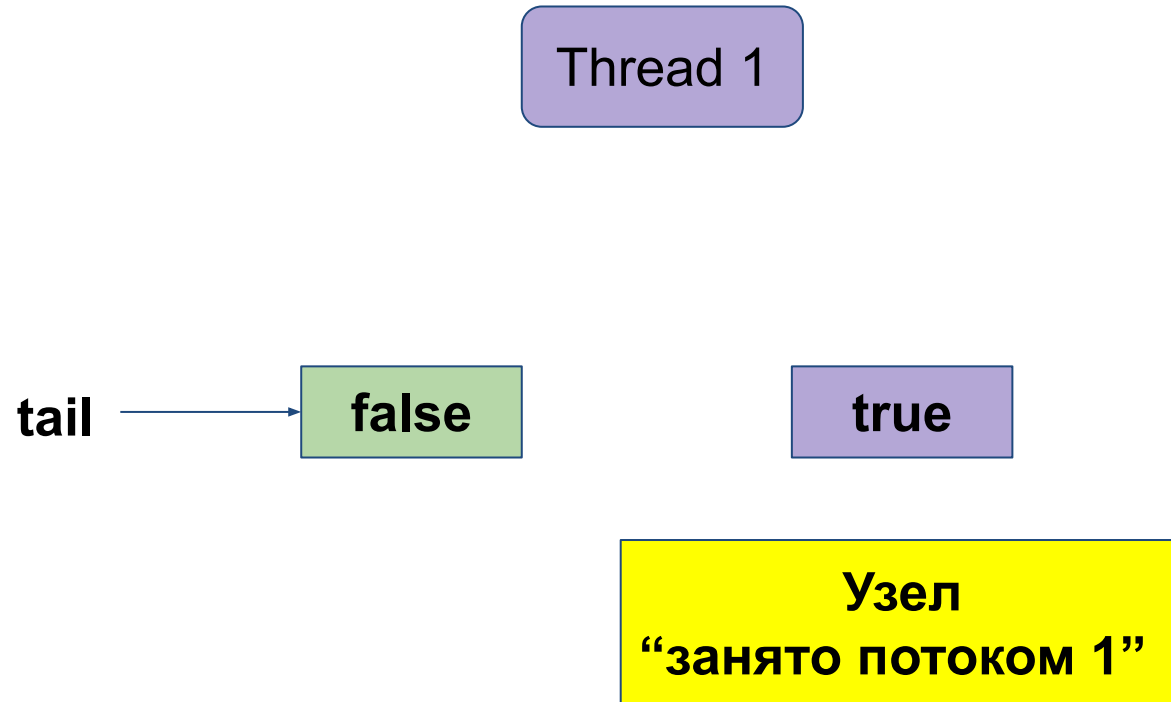
false

CLH Lock

```
class QNode:  
    boolean locked // shared, atomic
```

```
class CLHLock:  
    tail = QNode() // shared, atomic  
    treadlocal my = QNode()
```

CLH Lock: Узел 1

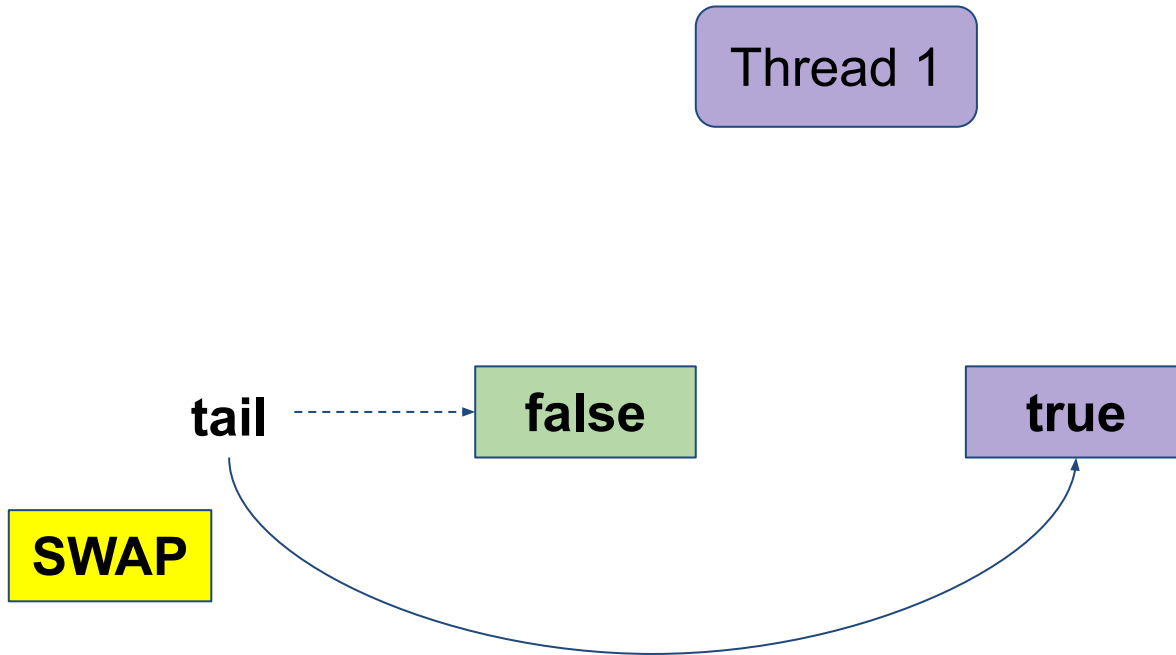


CLH Lock

```
class QNode:  
    boolean locked // shared, atomic
```

```
class CLHLock:  
    tail = QNode() // shared, atomic  
    treadlocal my = QNode()  
  
    def lock():  
        my.locked = true  
        pred = tail.getAndSet(my)  
        while pred.locked: pass
```

CLH Lock: Захват блокировки

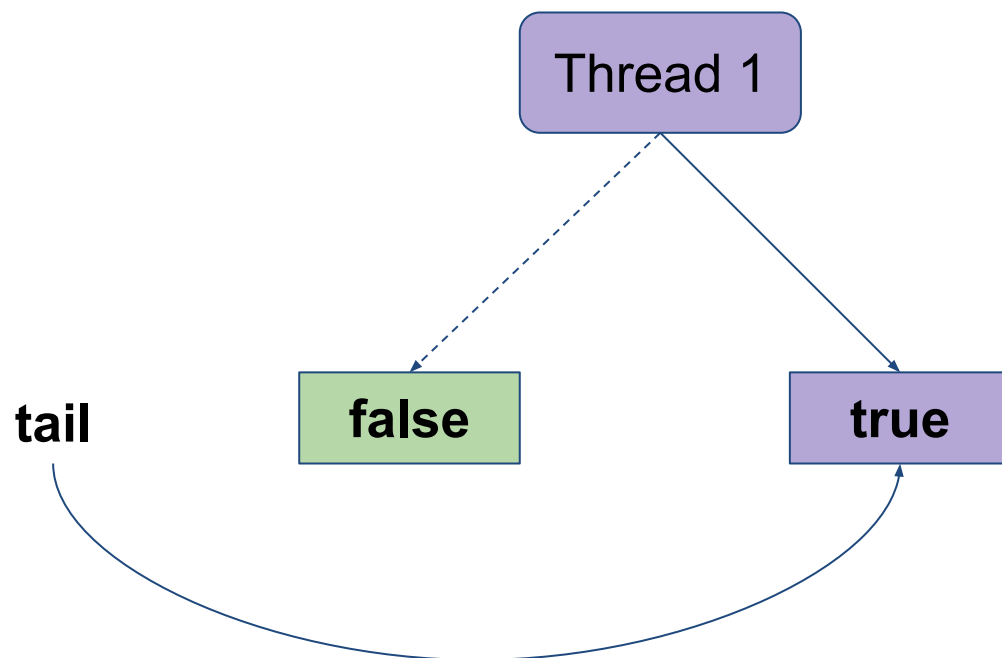


CLH Lock

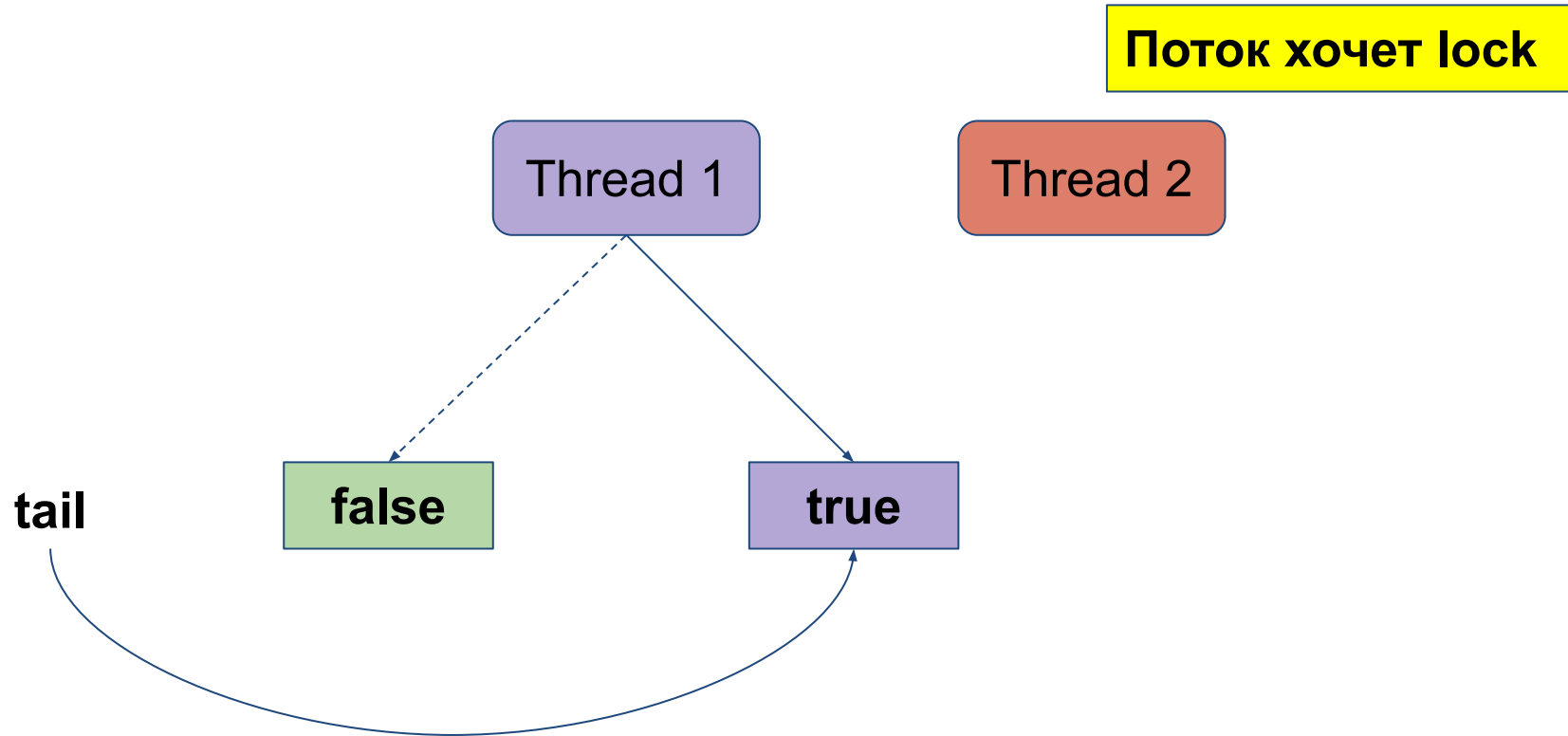
```
class QNode:  
    boolean locked // shared, atomic
```

```
class CLHLock:  
    tail = QNode() // shared, atomic  
    treadlocal my = QNode()  
  
    def lock():  
        my.locked = true  
        pred = tail.getAndSet(my)  
        while pred.locked: pass
```

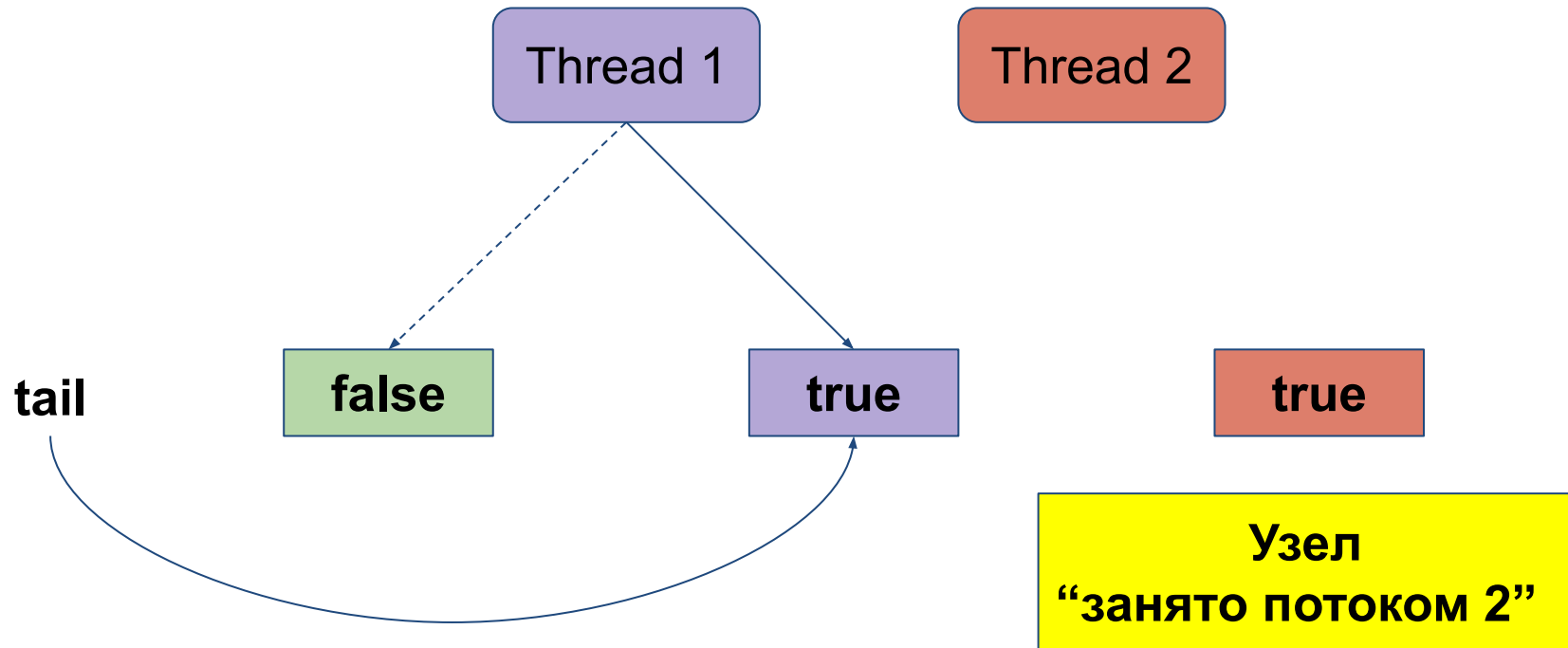
CLH Lock: Занято 1-м потоком



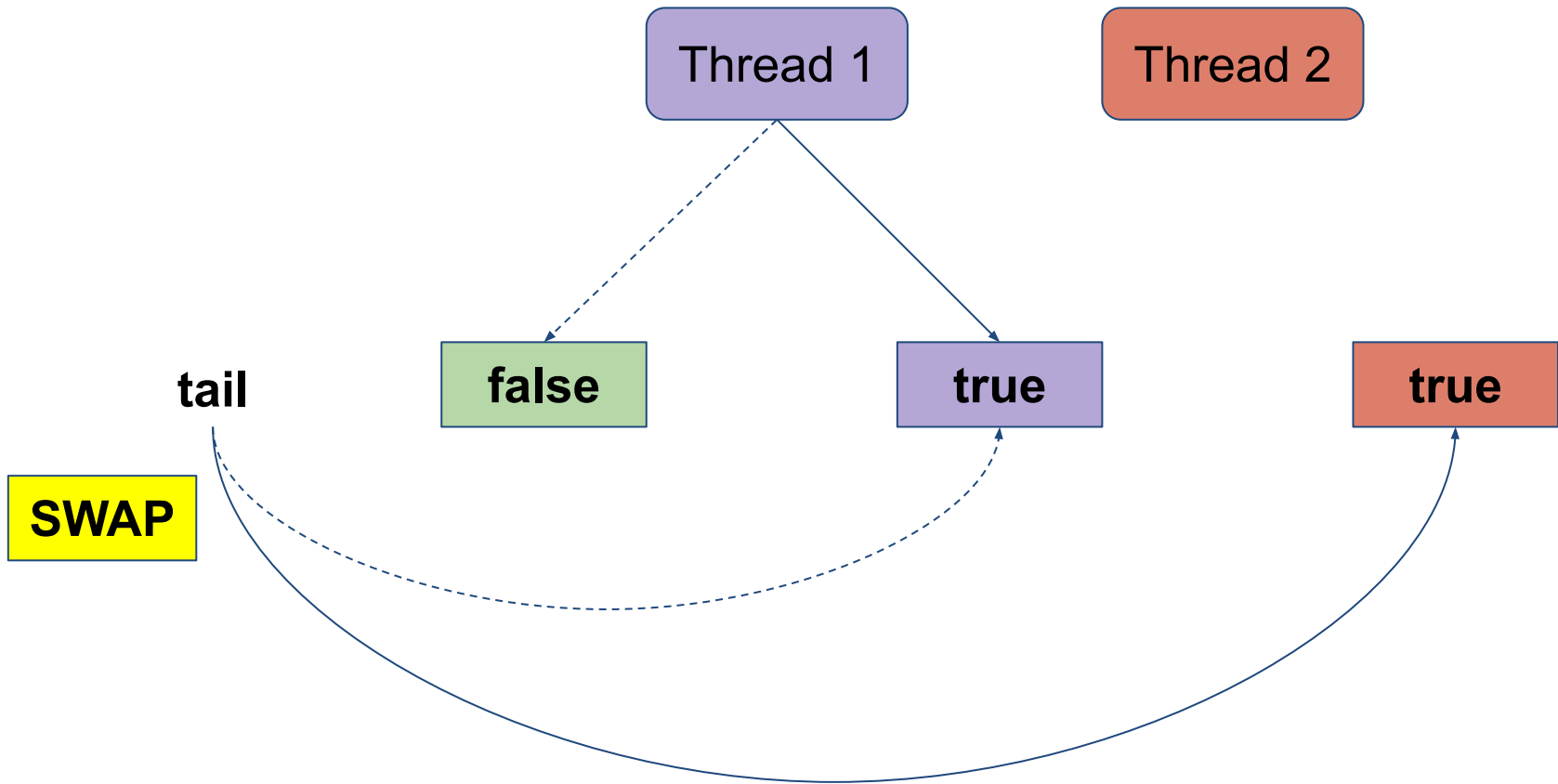
CLH Lock: Поток 2



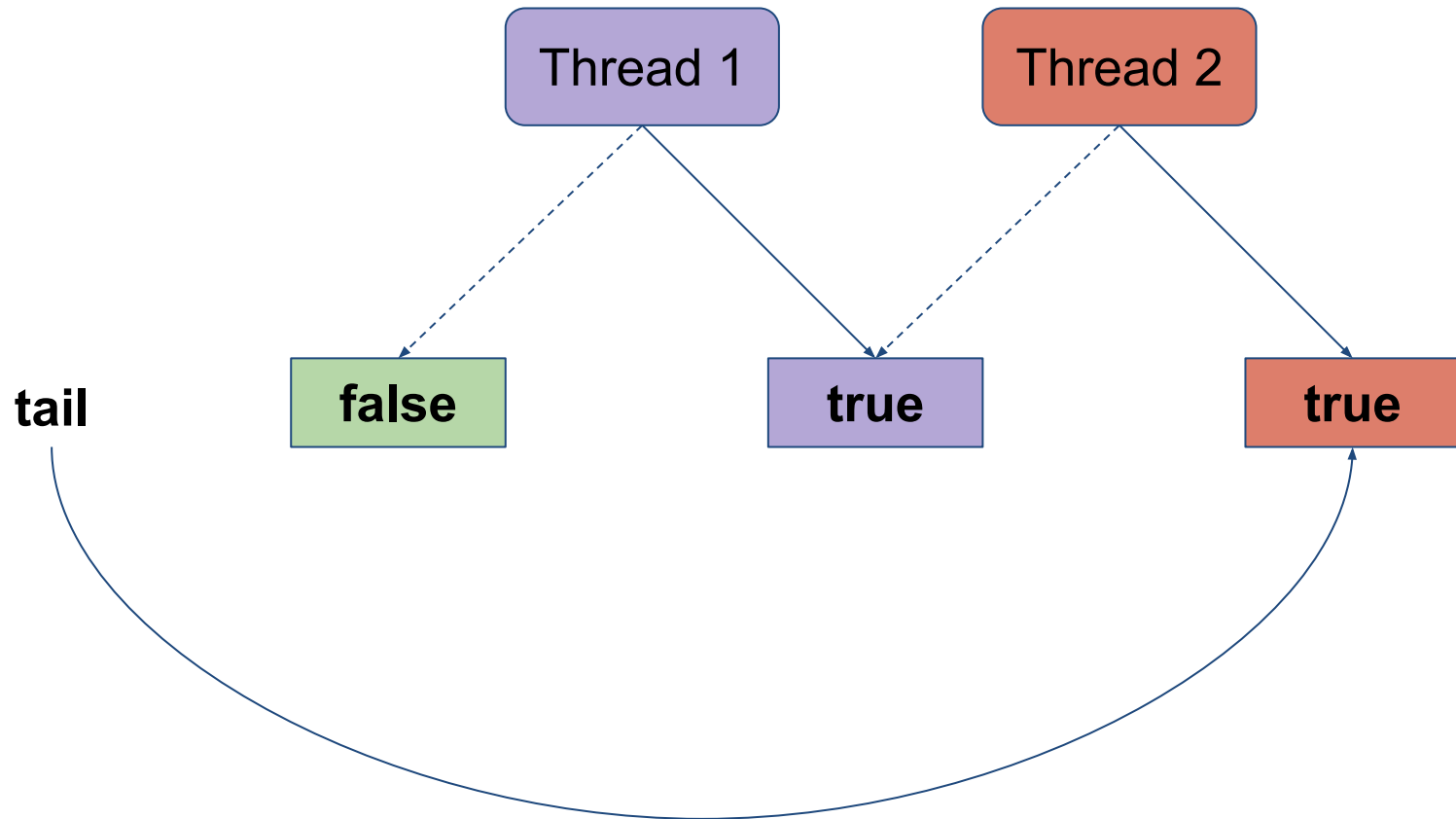
CLH Lock: Узел 2



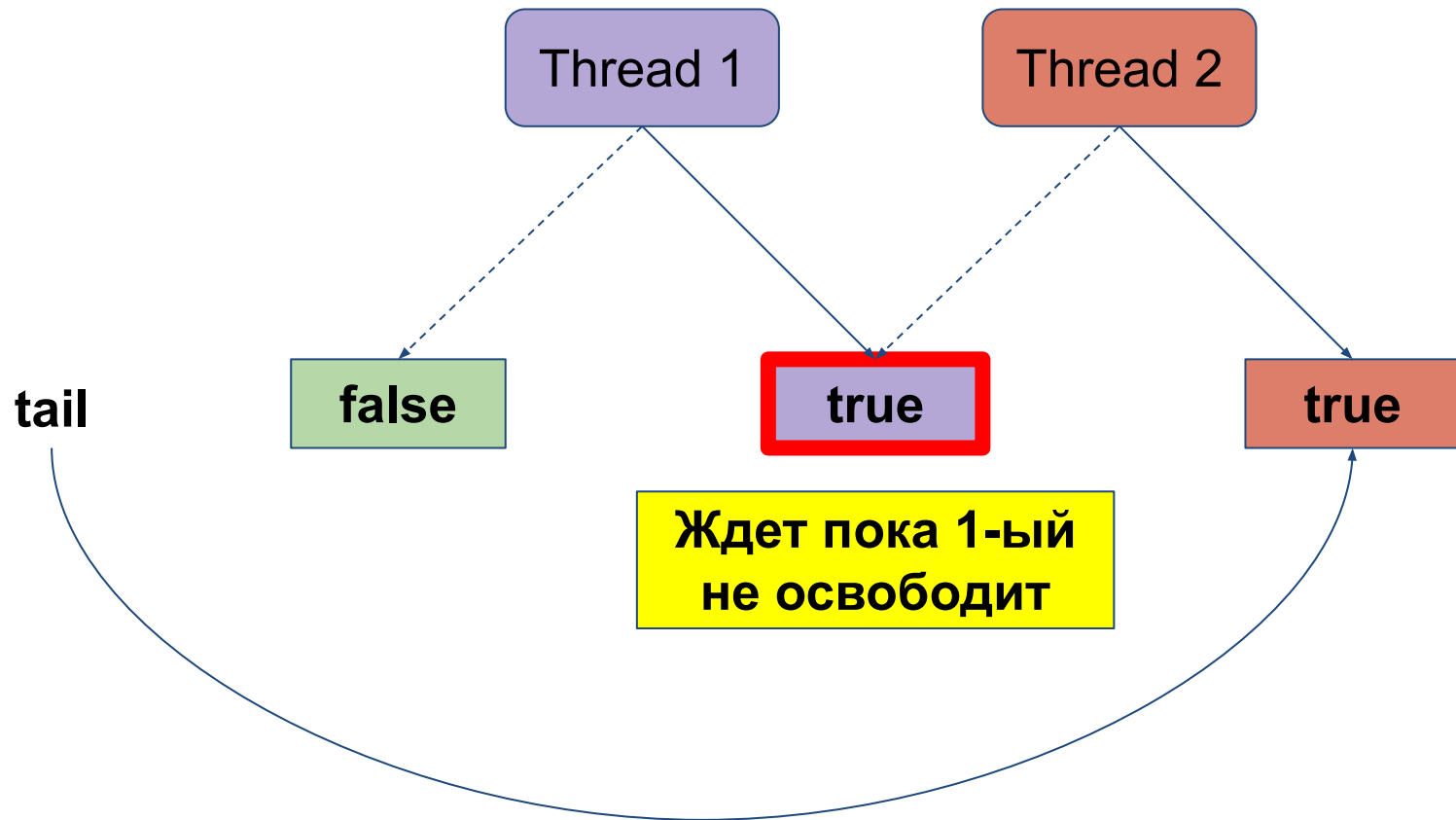
CLH Lock: Добавление в очередь



CLH Lock: Поток 2 в очереди



CLH Lock: Поток 2 ждет

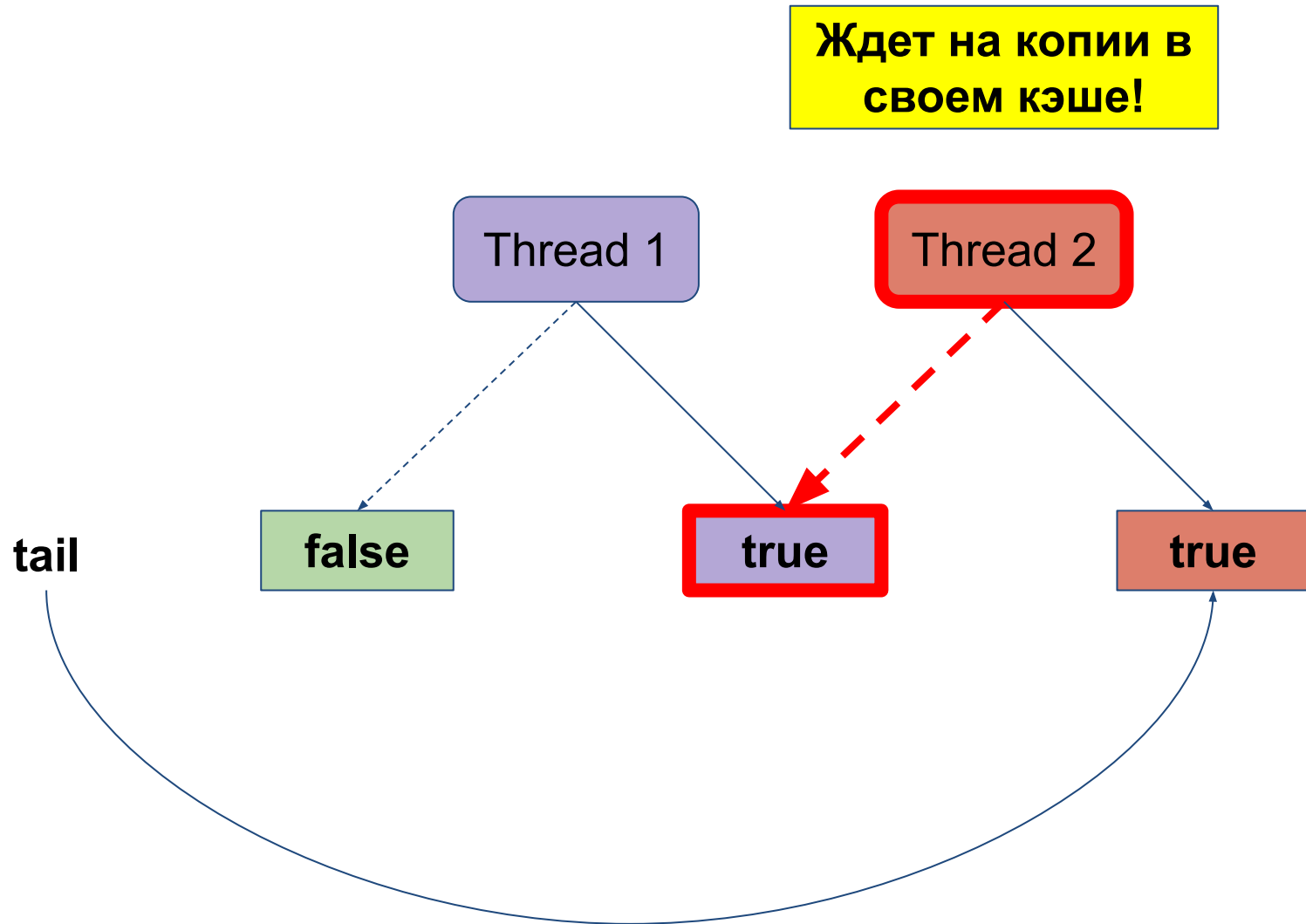


CLH Lock

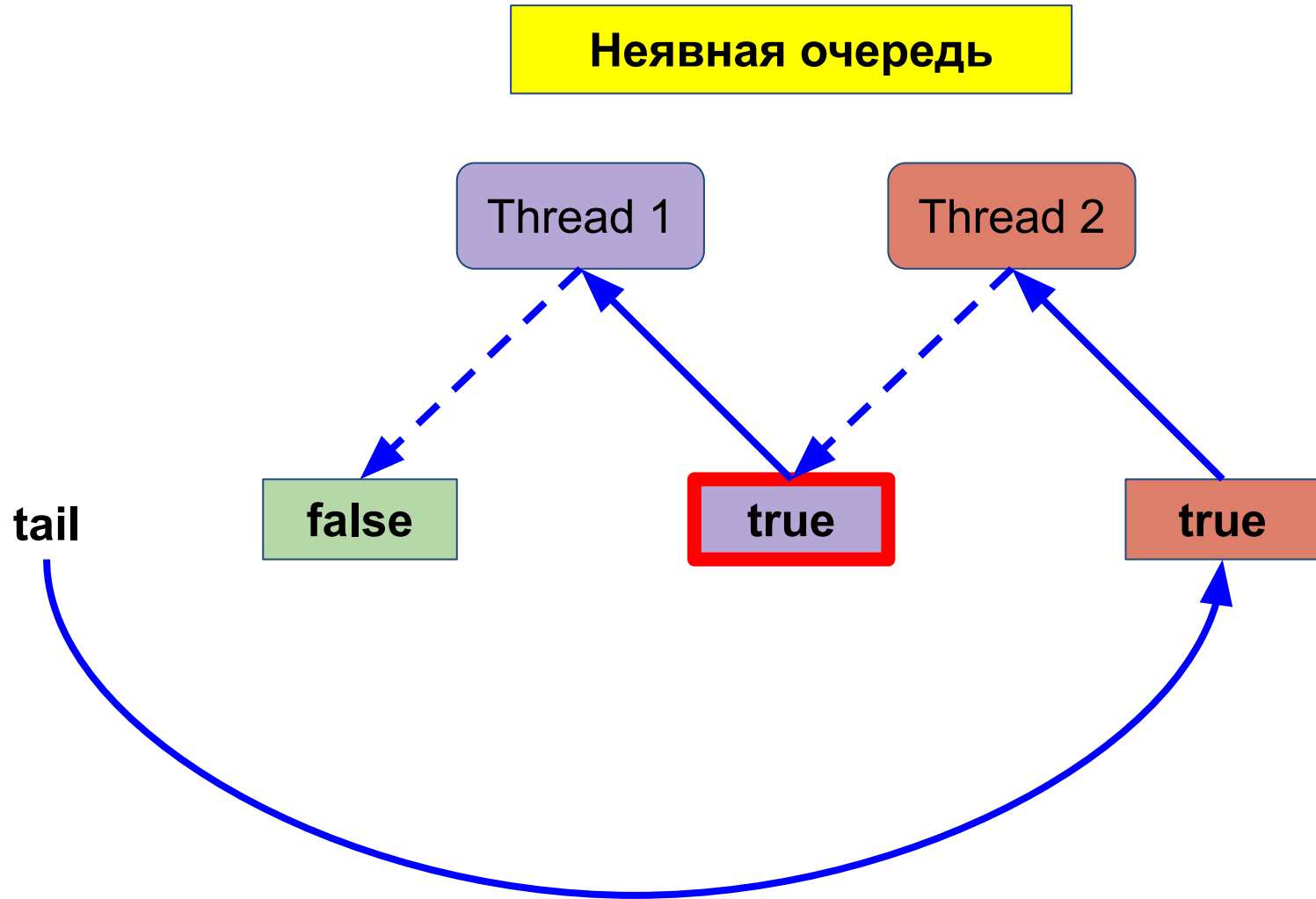
```
class QNode:  
    boolean locked // shared, atomic
```

```
class CLHLock:  
    tail = QNode() // shared, atomic  
    treadlocal my = QNode()  
  
    def lock():  
        my.locked = true  
        pred = tail.getAndSet(my)  
        while pred.locked: pass
```

CLH Lock: Поток 2 ждет



CLH Lock: Где очередь?



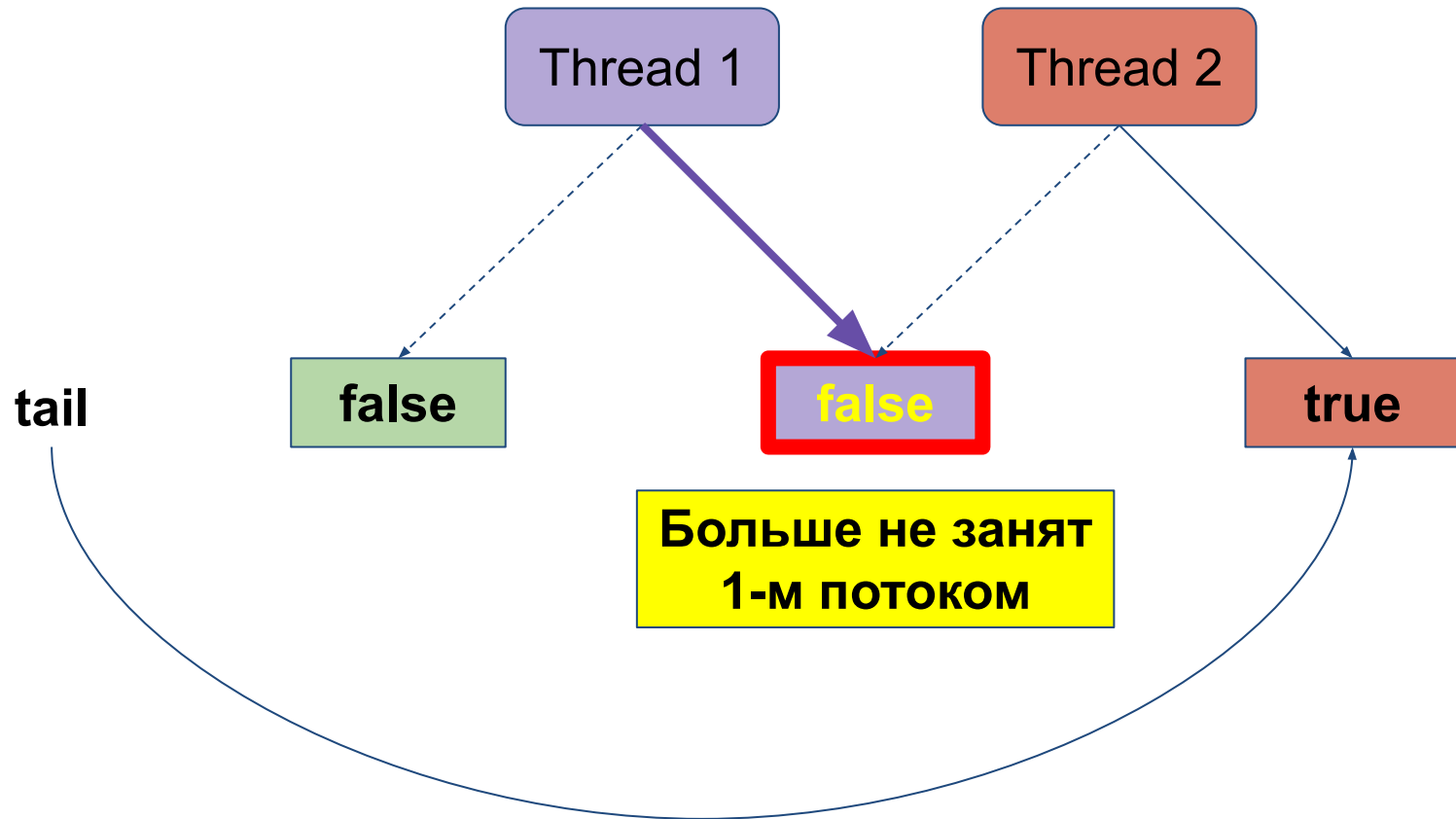
CLH Lock: unlock

```
class QNode:  
    boolean locked // shared, atomic
```

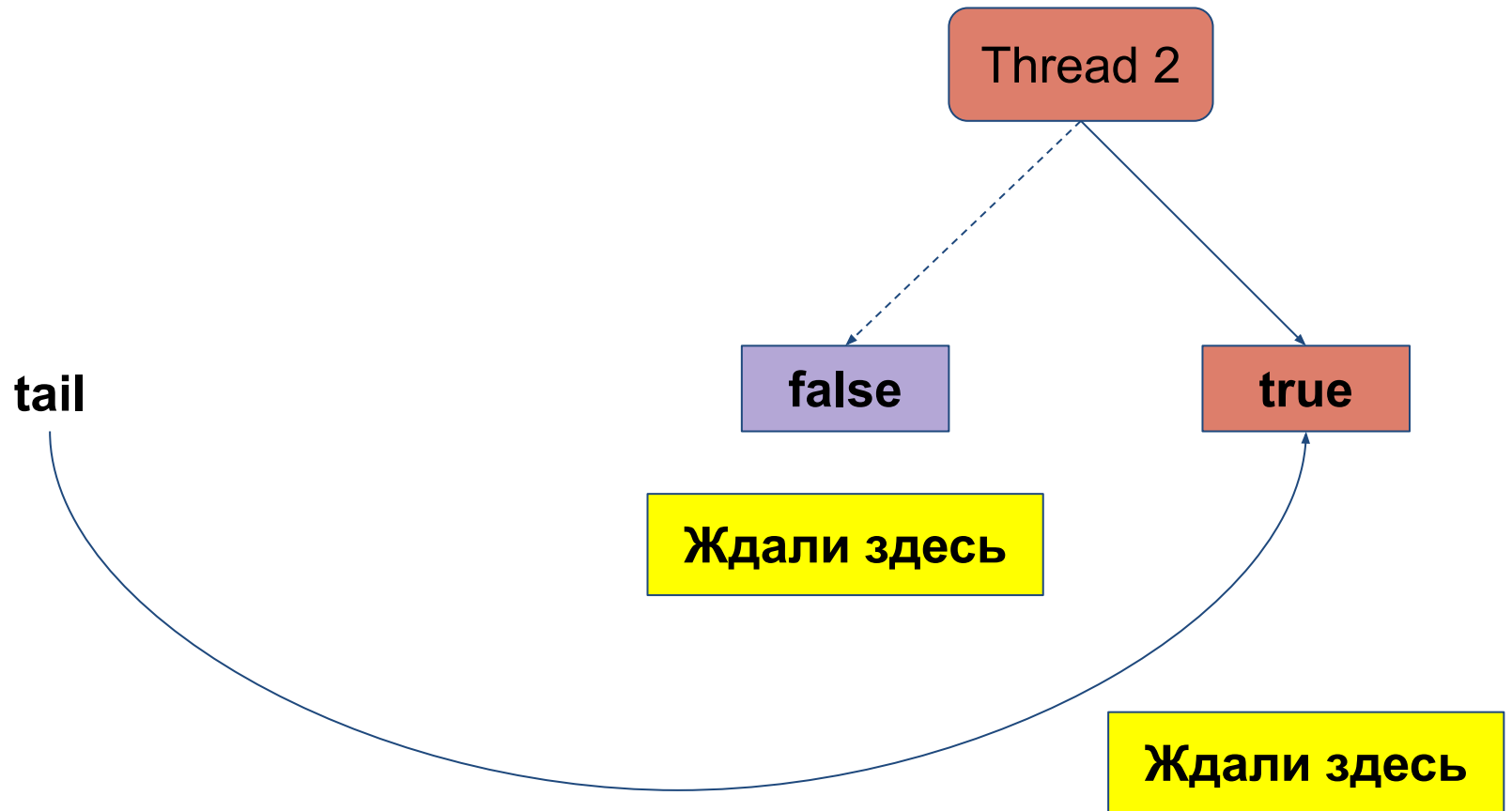
```
class CLHLock:  
    tail = QNode() // shared, atomic  
    treadlocal my = QNode()  
  
    def lock():  
        my.locked = true  
        pred = tail.getAndSet(my)  
        while pred.locked: pass  
  
    def unlock():  
        my.locked = false  
        my = pred
```

No invalidation storm!

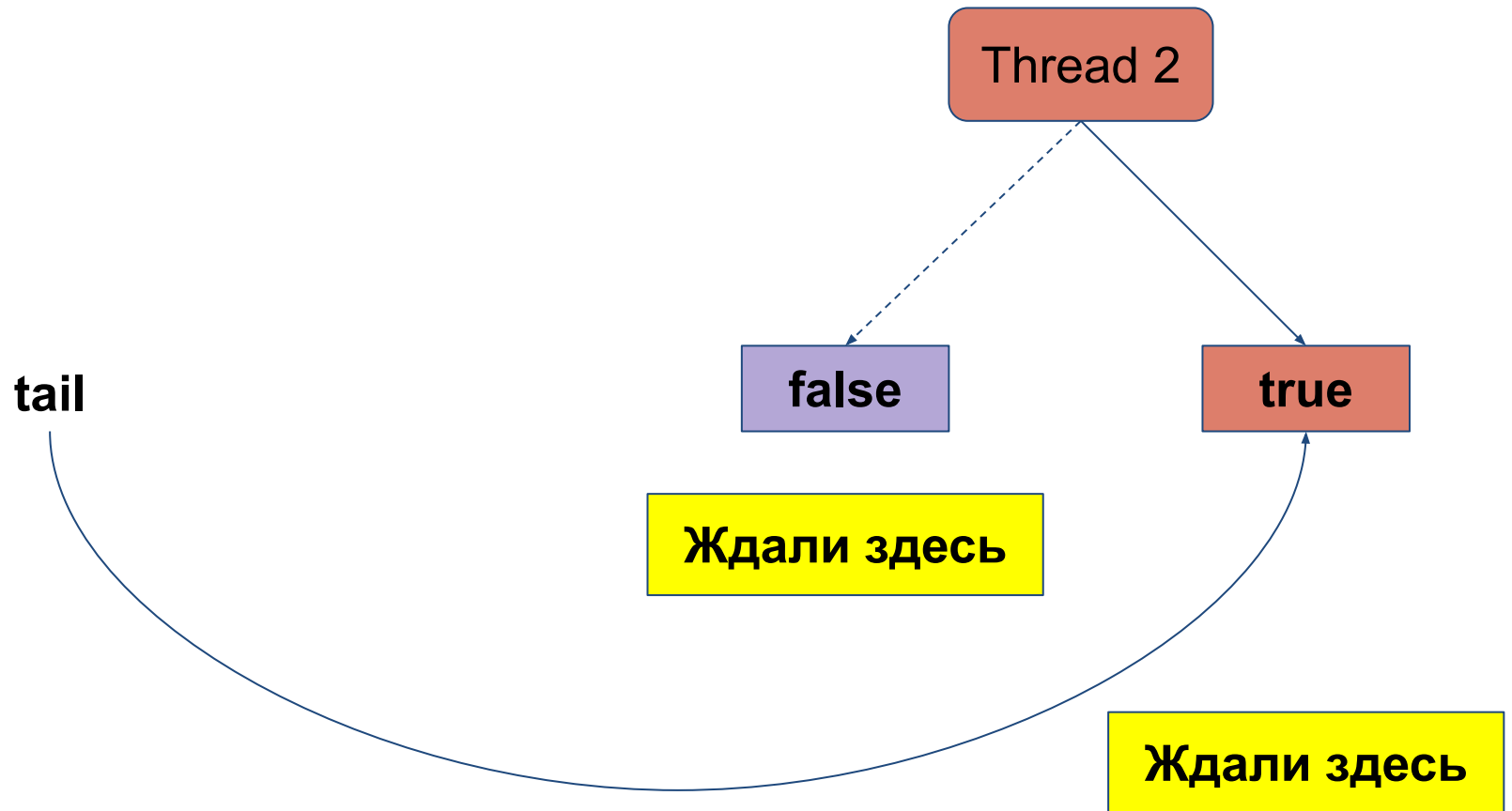
CLH Lock: Поток 1 освобождает



CLH Lock: Занято 2-м потоком



CLH Lock: Занято 2-м потоком



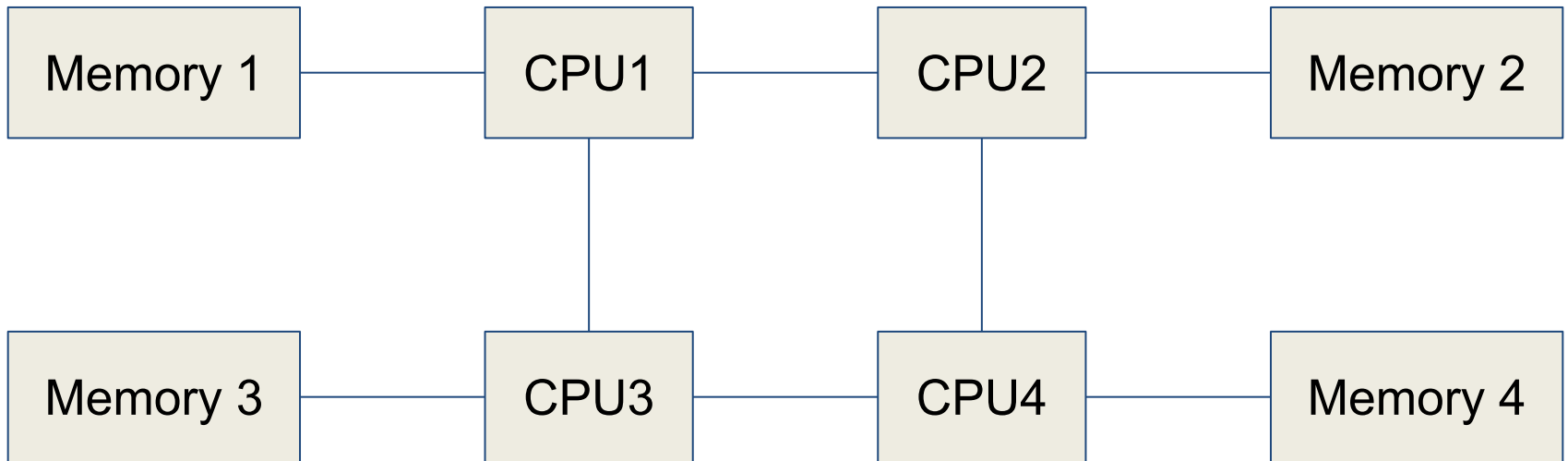
CLH Lock: Красивый memory reuse

```
class QNode:  
    boolean locked // shared, atomic
```

```
class CLHLock:  
    tail = QNode() // shared, atomic  
    treadlocal my = QNode()  
    threadlocal pred  
  
    def lock():  
        my.locked = true  
        pred = tail.getAndSet(my)  
        while pred.locked: pass  
  
    def unlock():  
        my.locked = false  
        my = pred
```

NUMA

Non-Uniform Memory Access



CLH Lock

- **Хорошо**

- Освобождение блокировки влияет только на один поток
- Занимает мало памяти

- **Плохо**

- Ждет на “чужой памяти”, а она может быть “далеко” (NUMA!)

MCS Lock

- John **M**ellor-**C**rummey and Michael **S**cott
 - Algorithms for scalable synchronization on shared-memory multiprocessors, 1991
- Ждем на своей памяти
- First-Come First-Served

MCS Lock

```
class QNode:  
    boolean locked // shared, atomic  
    QNode next = null
```

```
class CLHLock:  
    tail = null // shared, atomic
```

MCS Lock: Начало: Не занято



Не занято

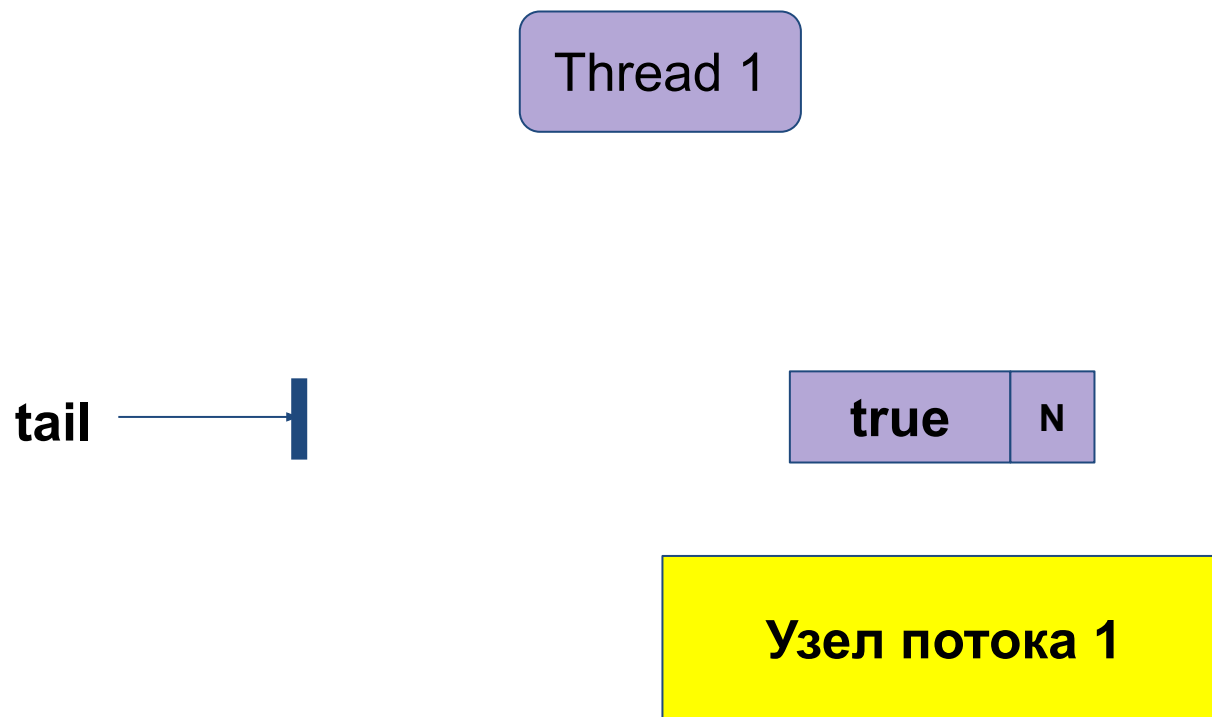
MCS Lock: Поток 1

Thread 1

Поток хочет lock

tail —→

MCS Lock: Узел 1



MCS Lock

```
class QNode:
```

```
    boolean locked // shared, atomic
```

```
    QNode next = null
```

```
class CLHLock:
```

```
    tail = QNode() // shared, atomic
```

```
    treadlocal my = null
```

```
    def lock():
```

```
        my = QNode() // new alloc!
```

```
        my.locked = true
```

```
        pred = tail.getAndSet(my)
```

```
        if pred != null:
```

```
            pred.next = my
```

```
            while my.locked: pass
```

MCS Lock: Захват

Thread 1

tail



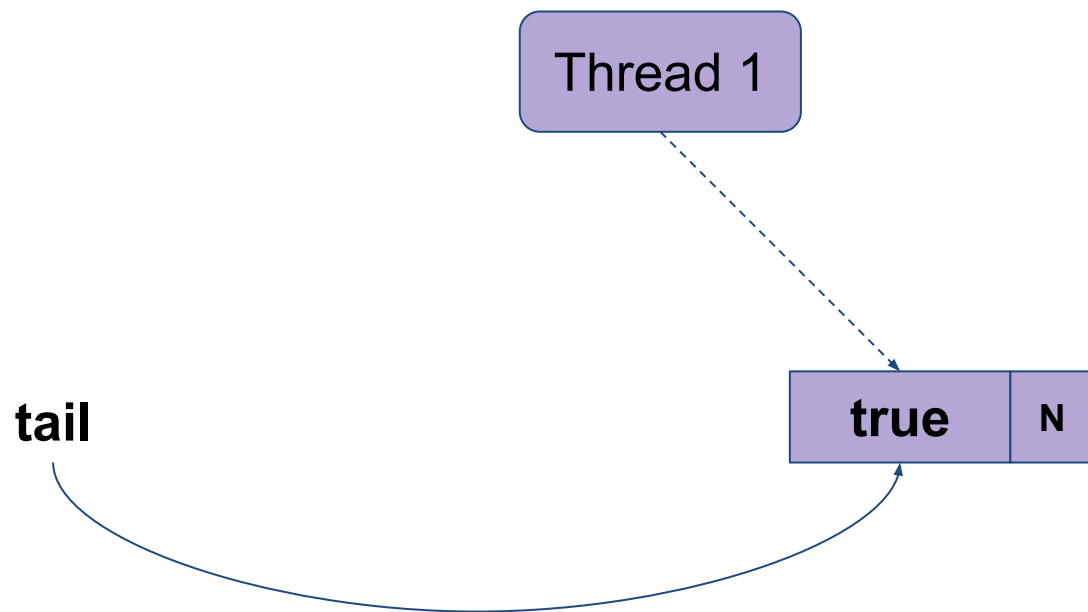
true

N

SWAP



MCS Lock: ЯВНЫЙ СПИСОК



MCS Lock

```
class QNode:
```

```
    boolean locked // shared, atomic
```

```
    QNode next = null
```

```
class CLHLock:
```

```
    tail = QNode() // shared, atomic
```

```
    treadlocal my = null
```

```
    def lock():
```

```
        my = QNode() // new alloc!
```

```
        my.locked = true
```

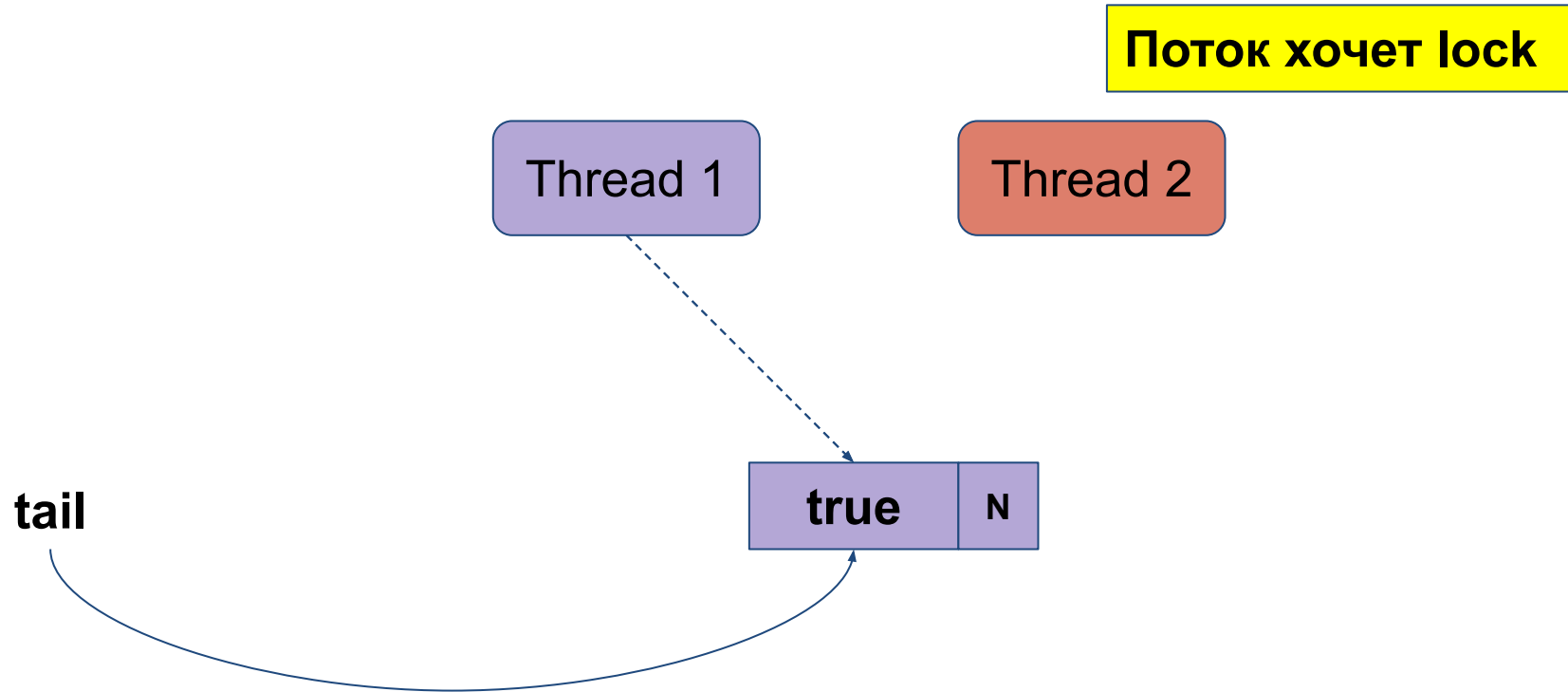
```
        pred = tail.getAndSet(my)
```

```
        if pred != null:
```

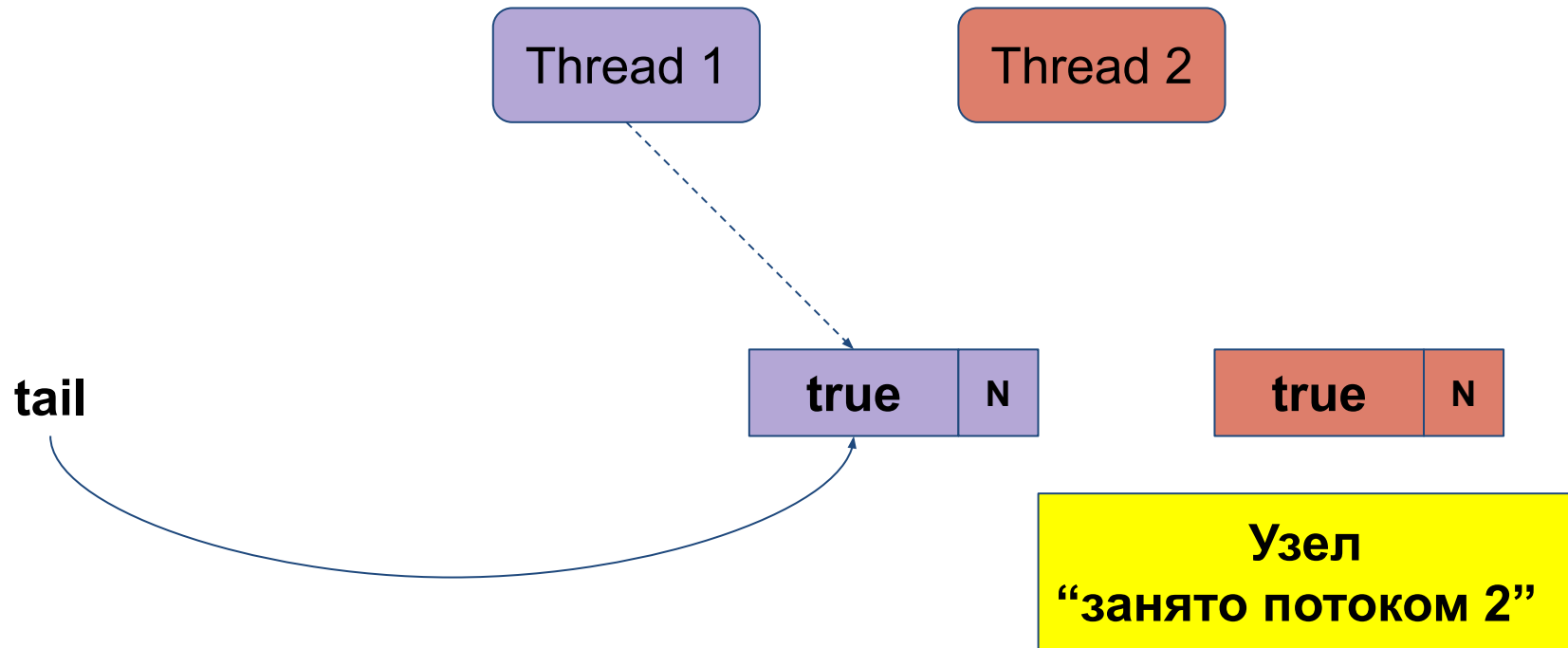
```
            pred.next = my
```

```
            while my.locked: pass
```

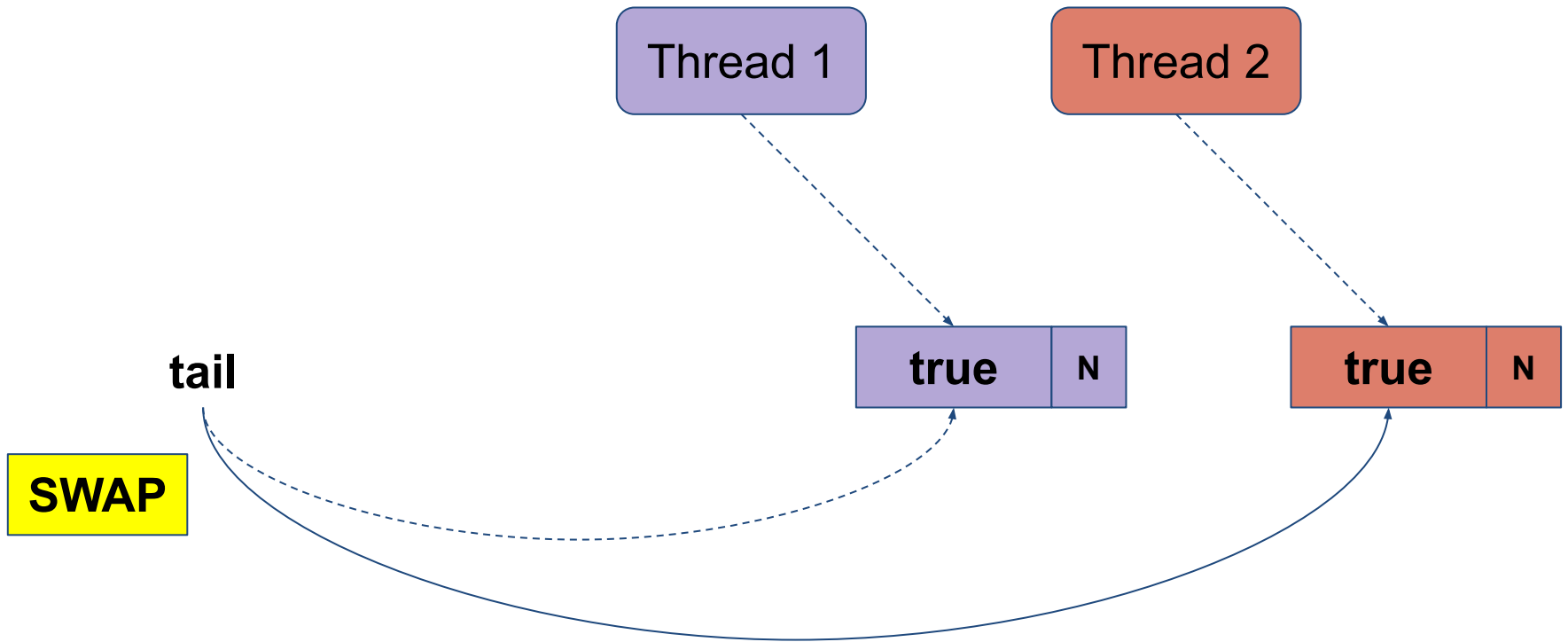
MCS Lock: Поток 2



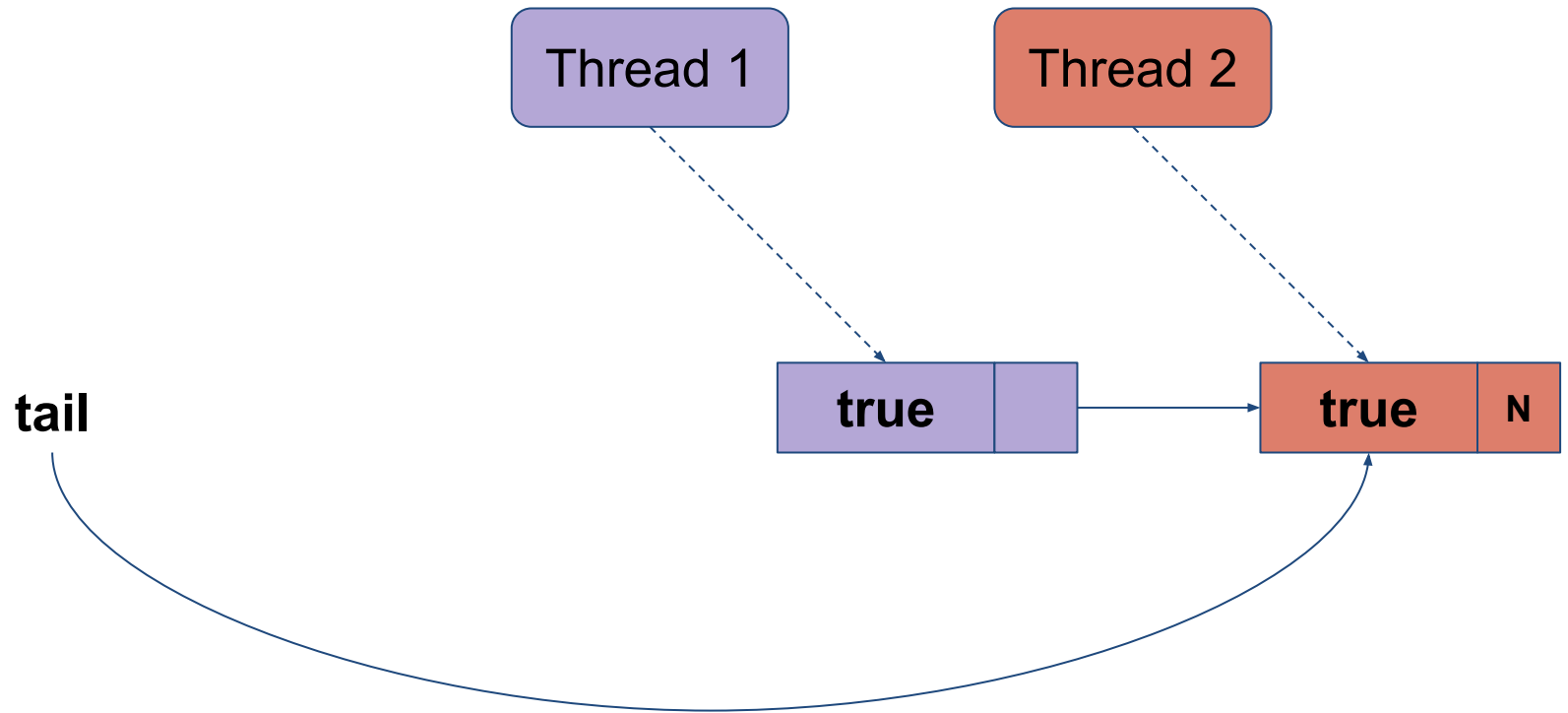
MCS Lock: Узел 2



MCS Lock: Добавление в очередь



MCS Lock: Создали список

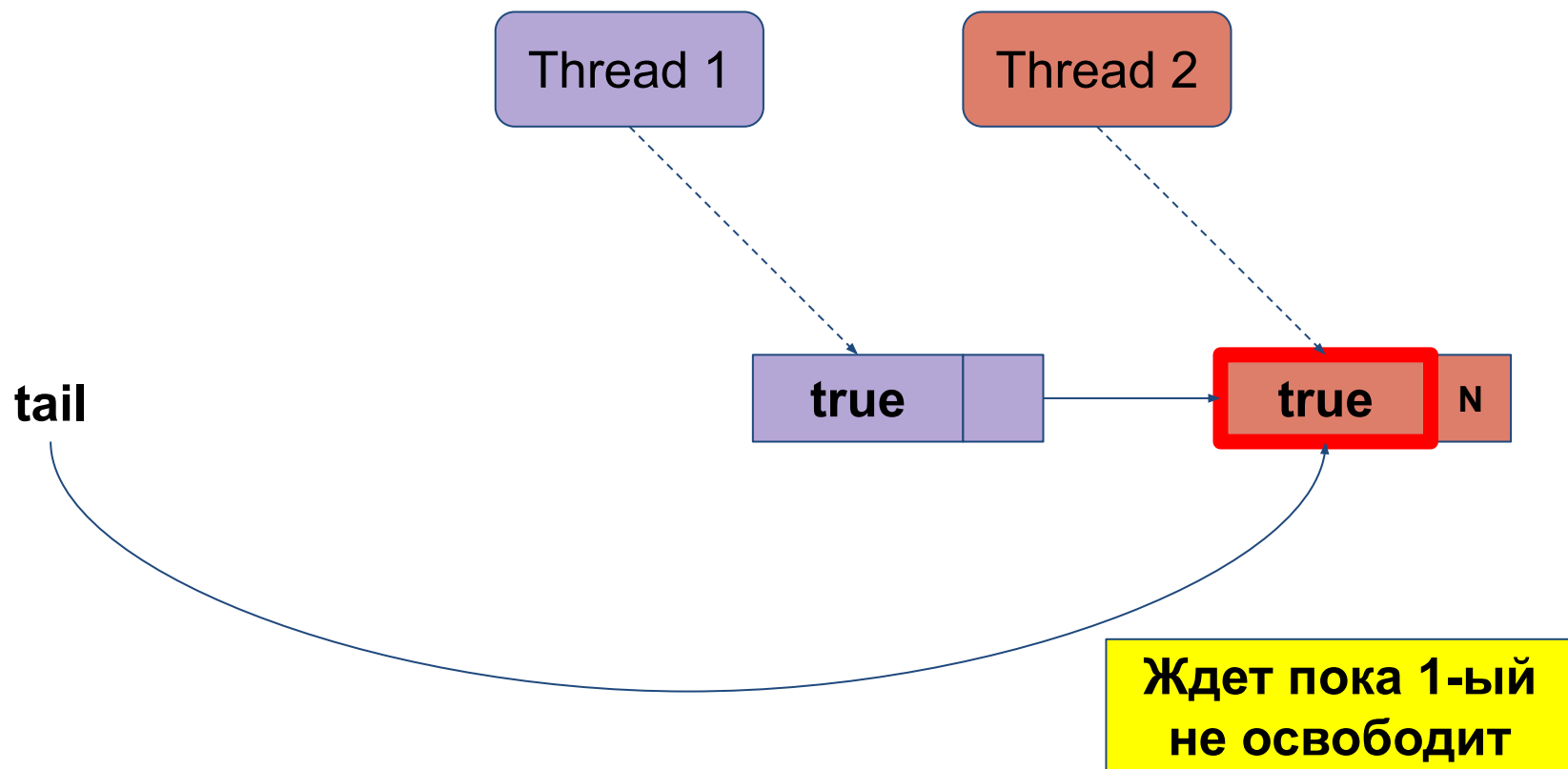


MCS Lock

```
class QNode:  
    boolean locked // shared, atomic  
    QNode next = null
```

```
class CLHLock:  
    tail = QNode() // shared, atomic  
    treadlocal my = null  
  
    def lock():  
        my = QNode()  
        pred = tail.getAndSet(my)  
        if pred != null:  
            pred.next = my  
        while my.locked: pass
```

MCS Lock: Ждем на своем объекте



MCS Lock

```
class QNode:
```

```
    boolean locked // shared, atomic
```

```
    QNode next = null
```

```
class CLHLock:
```

```
    tail = QNode() // shared, atomic
```

```
    treadlocal my = null
```

```
    def lock():
```

```
        my = QNode()
```

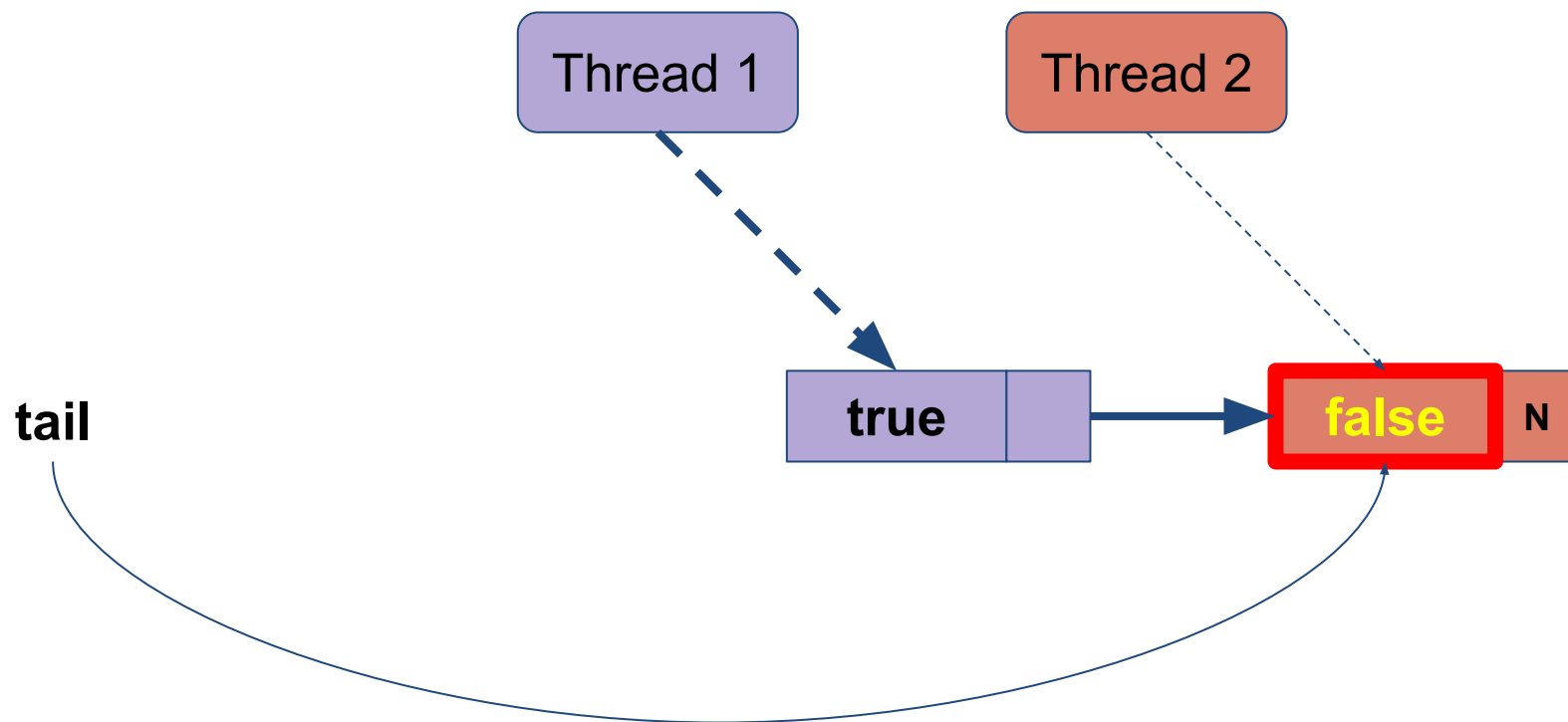
```
        pred = tail.getAndSet(my)
```

```
        if pred != null:
```

```
            pred.next = my
```

```
            while my.locked: pass
```

MCS Lock: Поток 1 освобождает

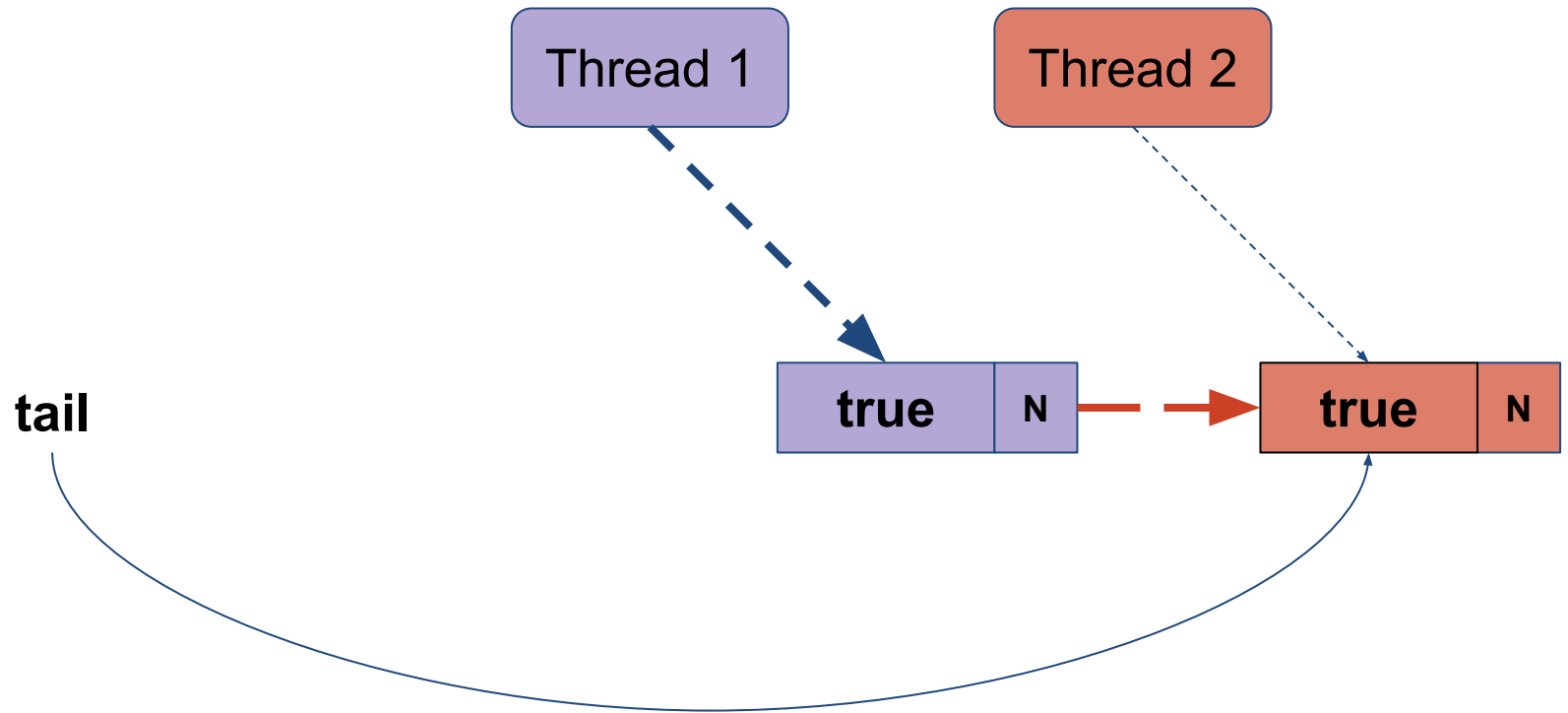


MCS Lock

```
class QNode:  
    boolean locked // shared, atomic  
    QNode next = null
```

```
class CLHLock:  
    tail = QNode() // shared, atomic  
    treadlocal my = null  
  
    def unlock():  
        if my.next == null:  
            if tail.CAS(my, null): return  
            else:  
                while my.next == null: pass  
                my.next.locked = false
```

MCS Lock: А что если ссылки еще нет?

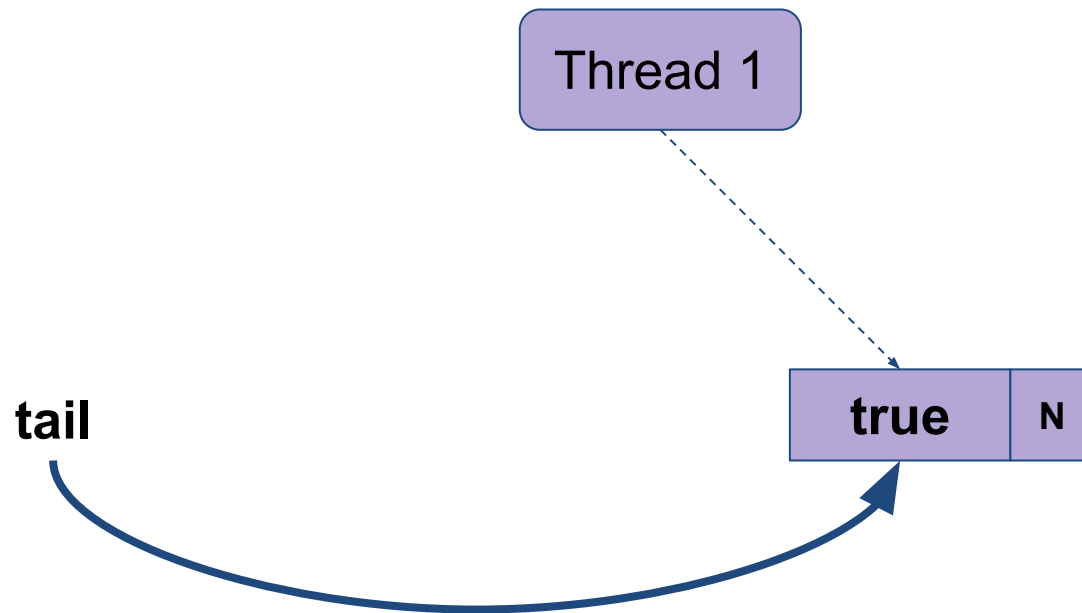


MCS Lock

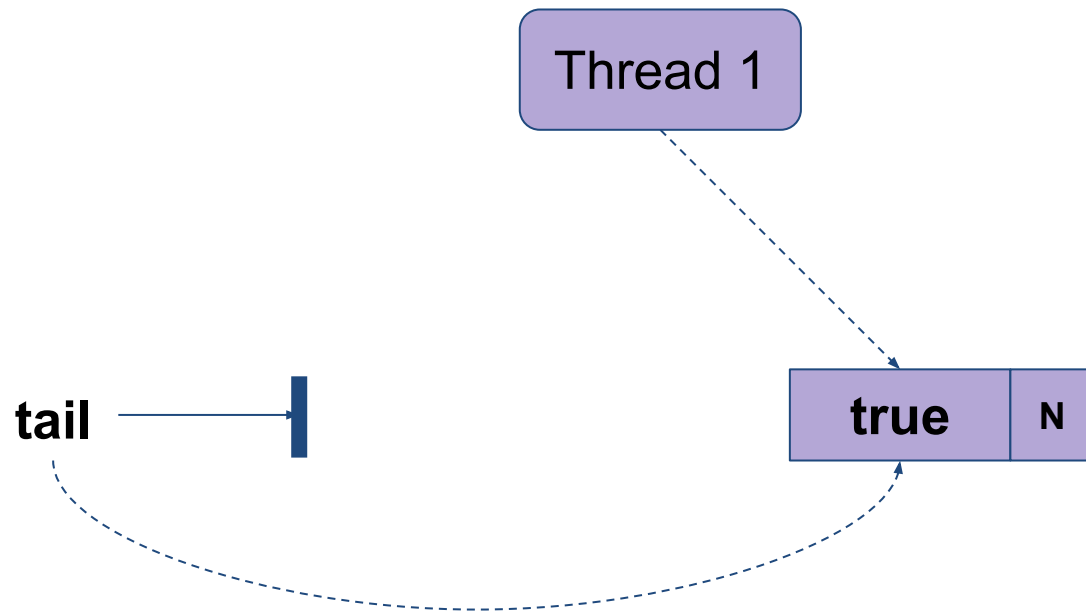
```
class QNode:  
    boolean locked // shared, atomic  
    QNode next = null
```

```
class CLHLock:  
    tail = QNode() // shared, atomic  
    treadlocal my = null  
  
    def unlock():  
        if my.next == null:  
            if tail.CAS(my, null): return  
            else:  
                while my.next == null: pass  
            my.next.locked = false
```

MCS Lock: Случай А: Других в очереди нет



MCS Lock: Случай А: Других в очереди нет

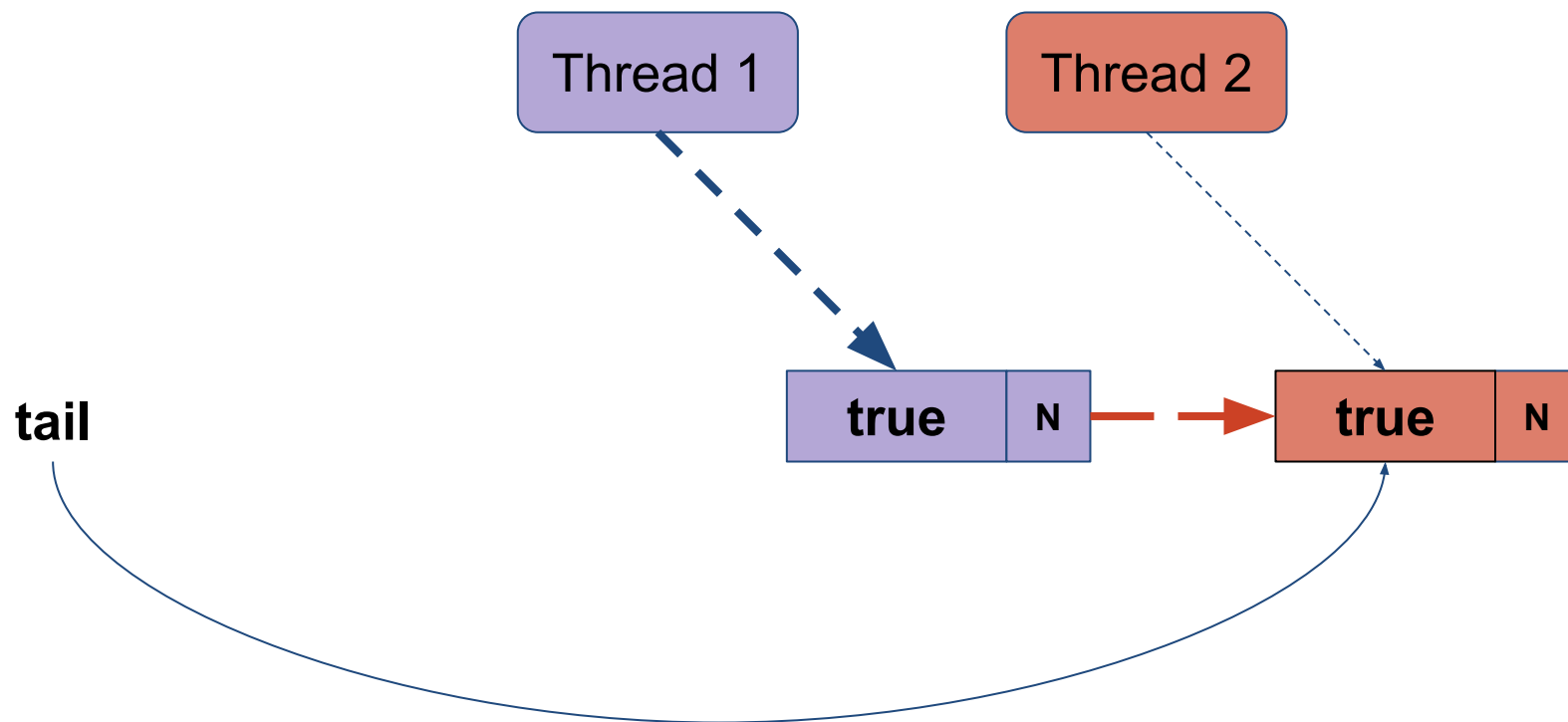


MCS Lock

```
class QNode:  
    boolean locked // shared, atomic  
    QNode next = null
```

```
class CLHLock:  
    tail = QNode() // shared, atomic  
    treadlocal my = null  
  
    def unlock():  
        if my.next == null:  
            if tail.CAS(my, null): return  
            else:  
                while my.next == null: pass  
            my.next.locked = false
```

MCS Lock: Случай Б: Другой не успел



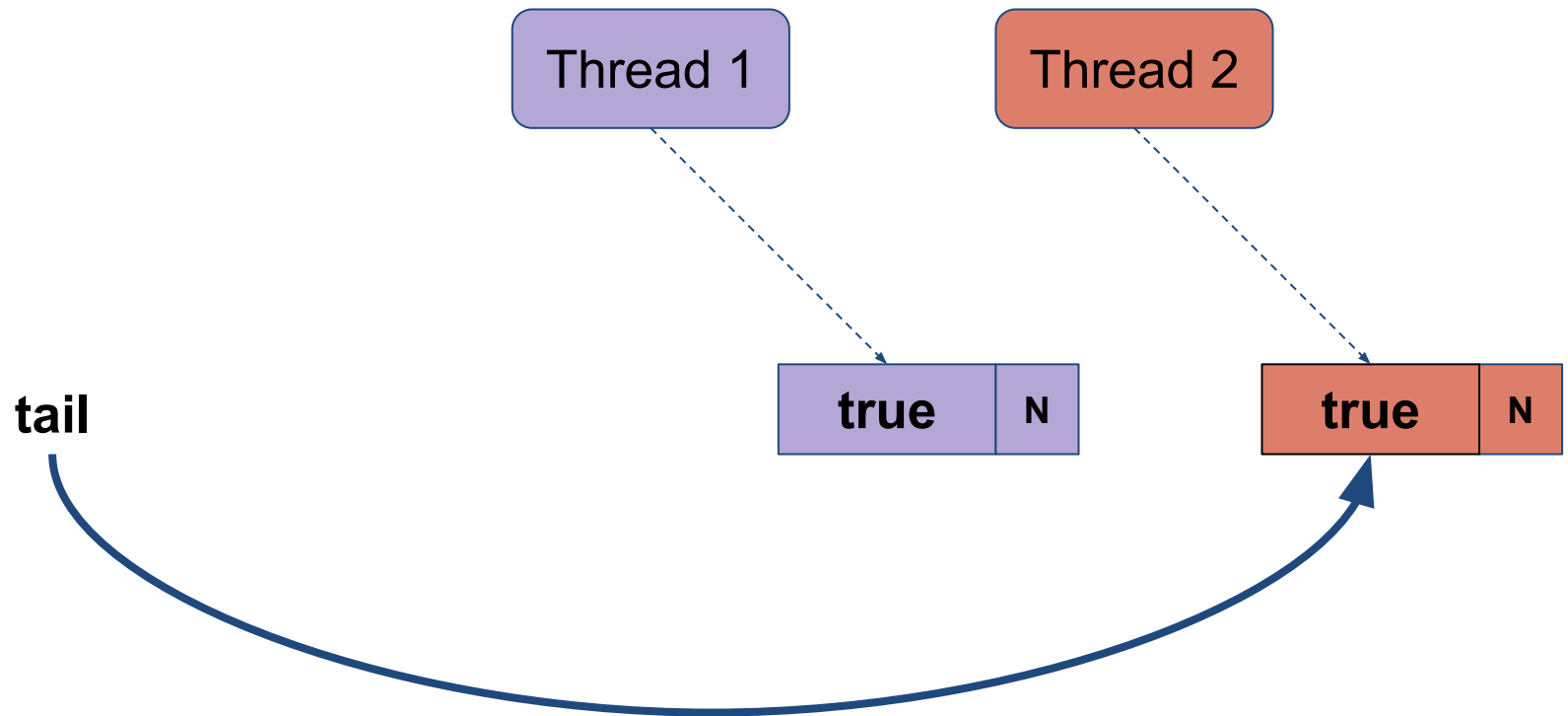
MCS Lock: Пишем сначала tail, потом next

```
class QNode:  
    boolean locked // shared, atomic  
    QNode next = null
```

```
class CLHLock:  
    tail = QNode() // shared, atomic  
    treadlocal my = null  
  
    def lock():  
        my = QNode()  
        pred = tail.getAndSet(my)  
        if pred != null:  
            pred.next = my  
            while my.locked: pass
```

Могли застрять между

MCS Lock: Случай Б: Другой не успел

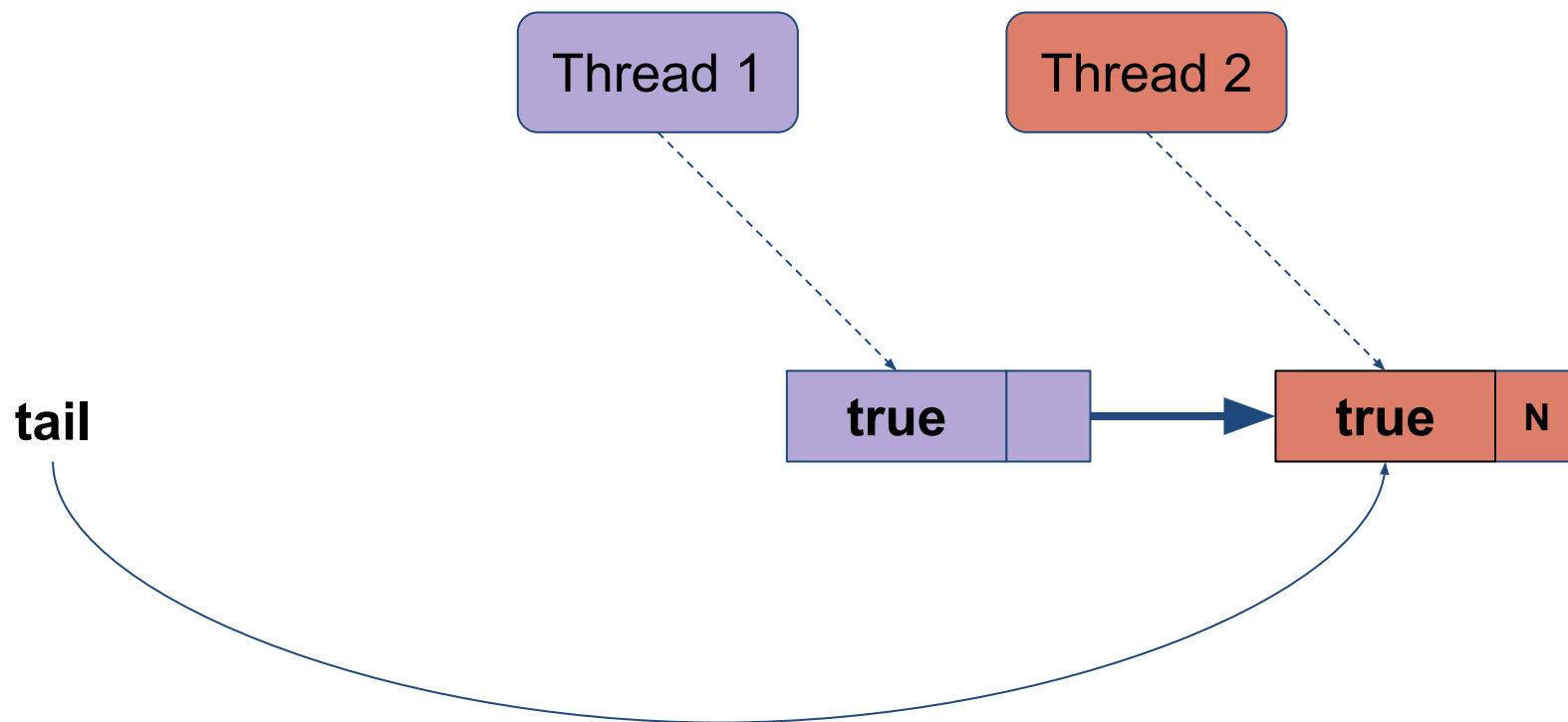


MCS Lock: Дождемся пока появится!

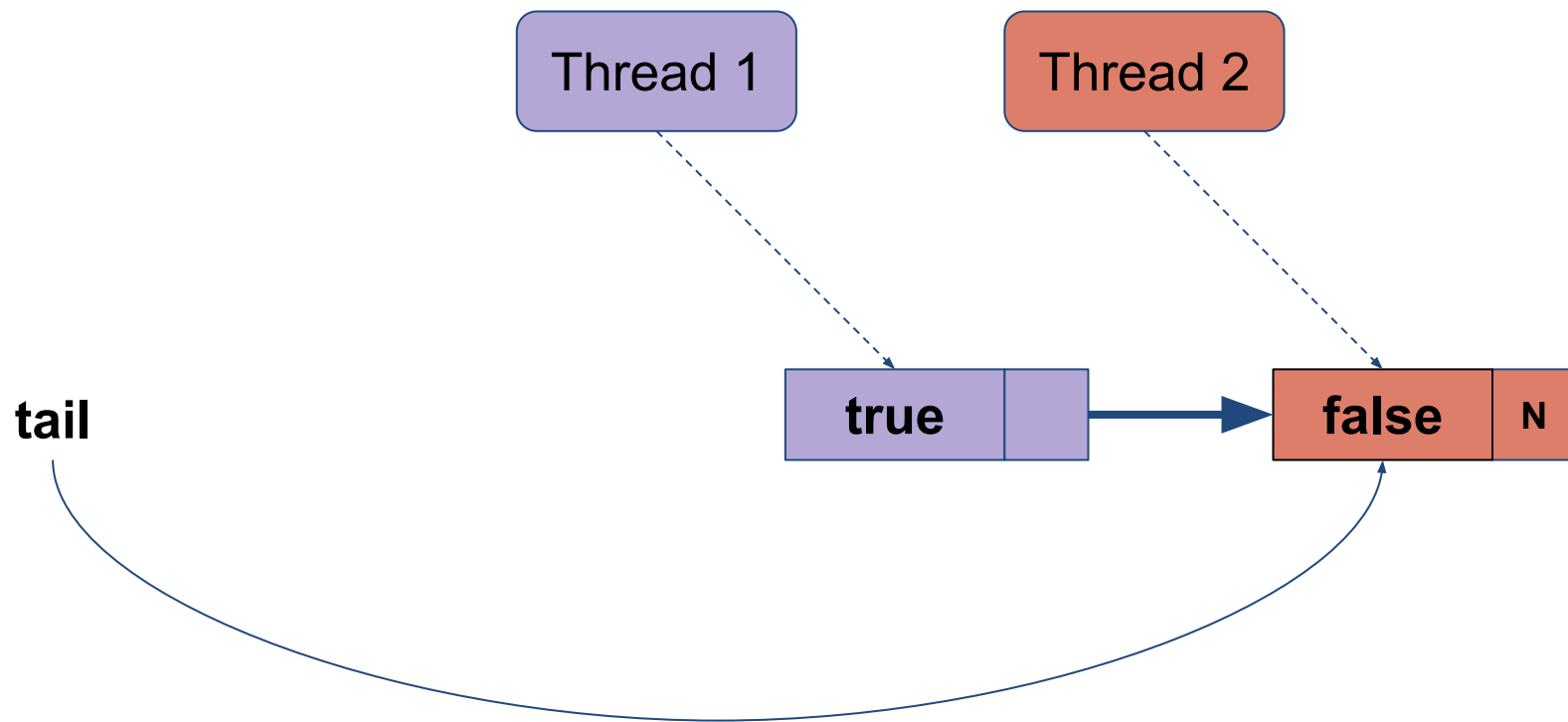
```
class QNode:  
    boolean locked // shared, atomic  
    QNode next = null
```

```
class CLHLock:  
    tail = QNode() // shared, atomic  
    treadlocal my = null  
  
    def unlock():  
        if my.next == null:  
            if tail.CAS(my, null): return  
            else:  
                while my.next == null: pass  
        my.next.locked = false
```


MCS Lock: Случай Б: Другой не успел



MCS Lock: Освободим лок



MCS Lock: Освободим лок

```
class QNode:
```

```
    boolean locked // shared, atomic
```

```
    QNode next = null
```

```
class CLHLock:
```

```
    tail = QNode() // shared, atomic
```

```
    treadlocal my = null
```

```
    def unlock():
```

```
        if my.next == null:
```

```
            if tail.CAS(my, null): return
```

```
            else:
```

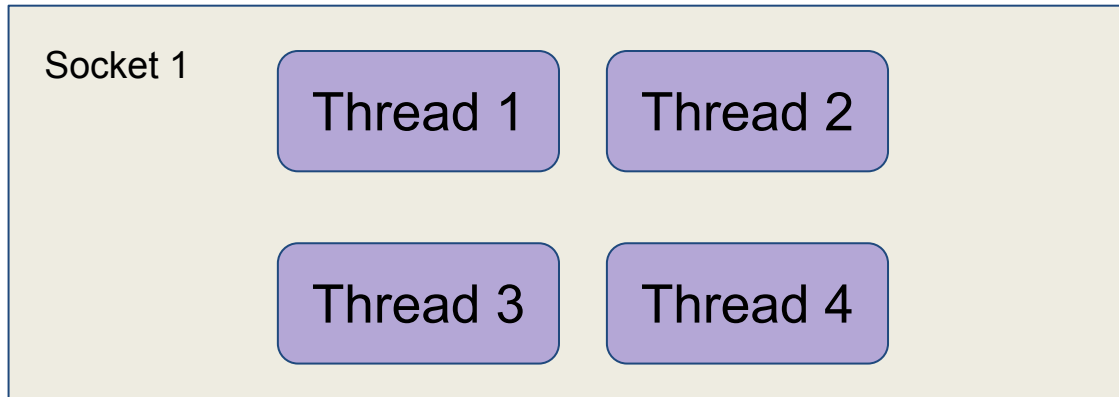
```
                while my.next == null: pass
```

```
                my.next.locked = false
```

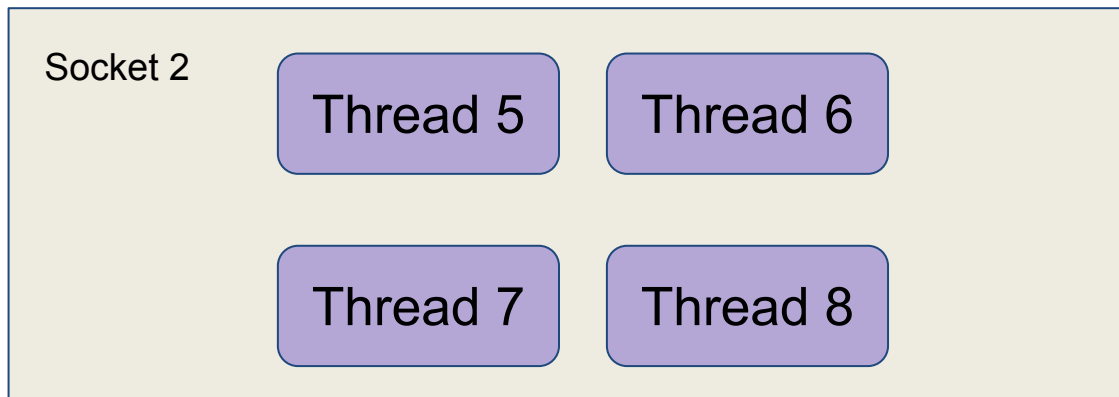
MCS Lock

- **Хорошо**
 - Ждет на “своей” памяти
- **Плохо**
 - tail изменяется всеми потоками -- будет большая конкуренция
 - Любой FCFS лок будет от этого страдать!

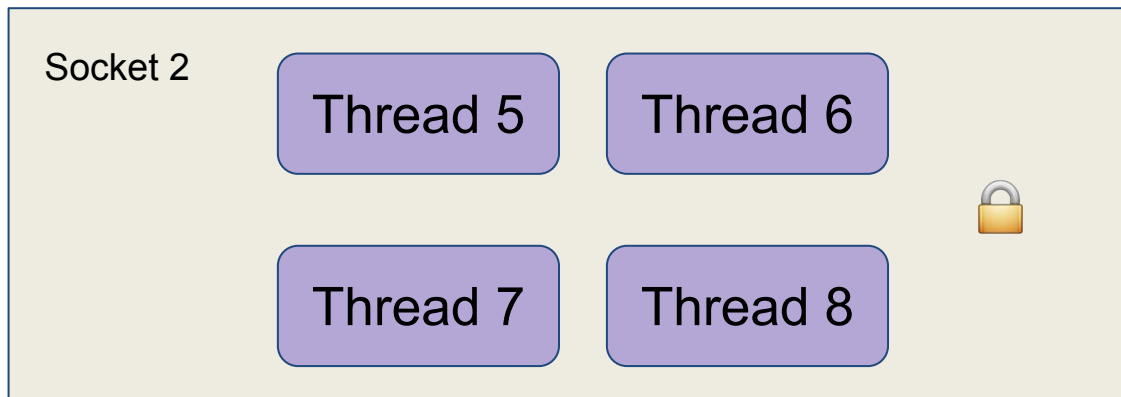
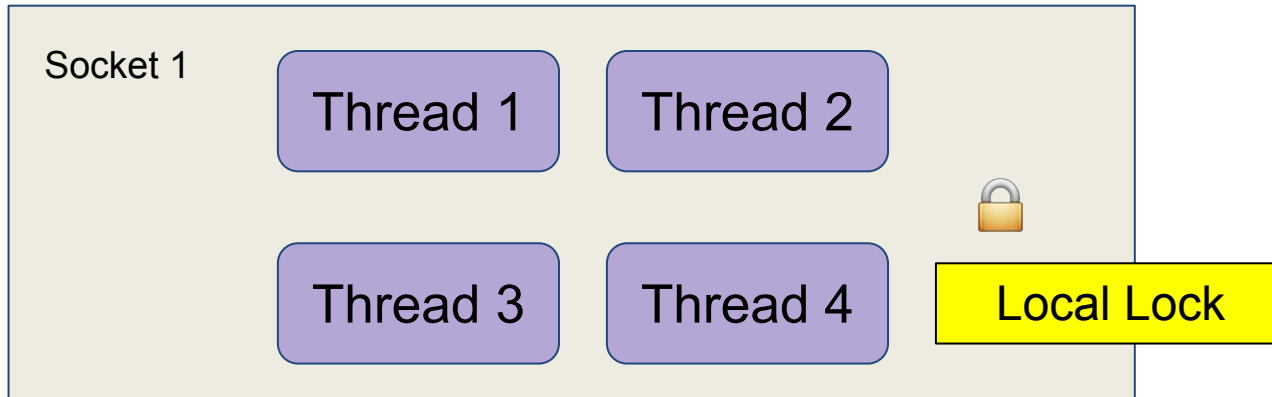
NUMA: Нечестность окупается!



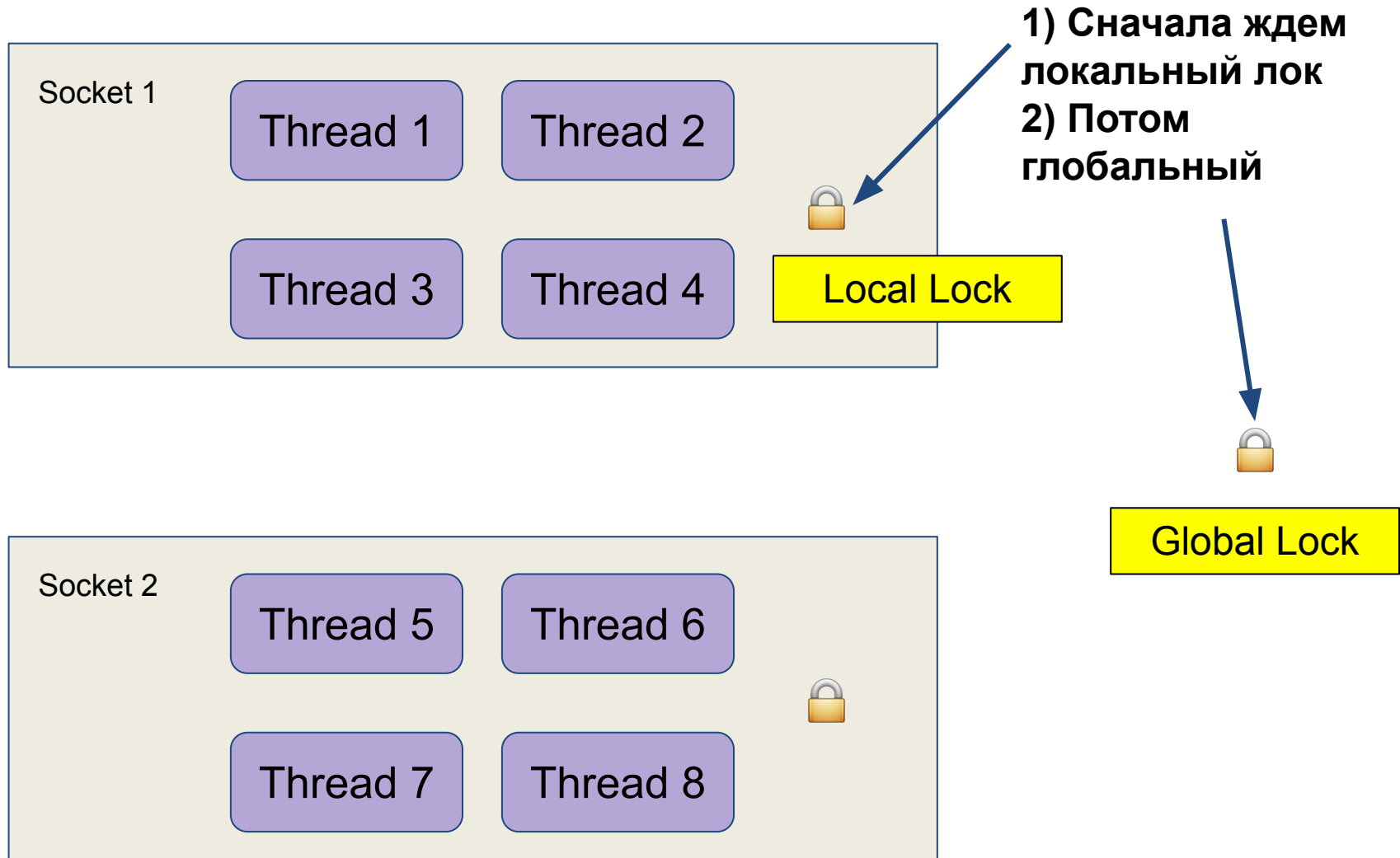
Медленный доступ к памяти между узлами в NUMA



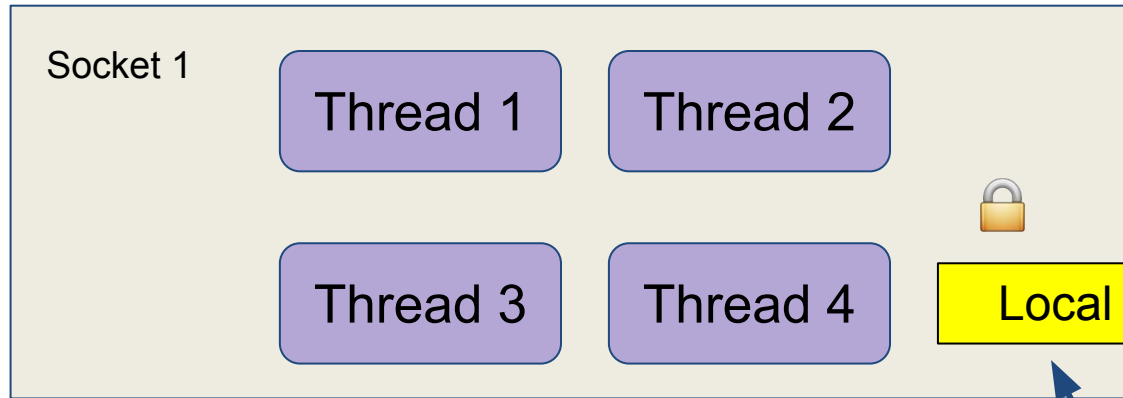
NUMA: Lock Cohorting



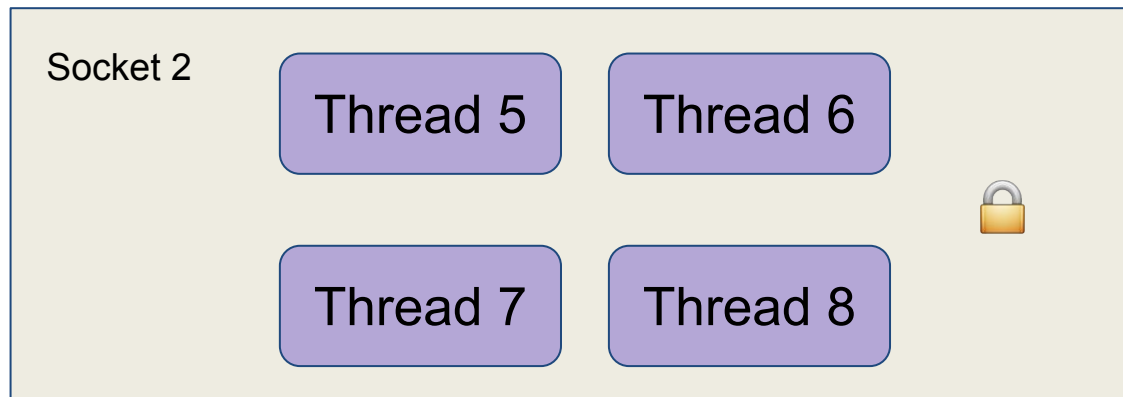
NUMA: Lock Cohorting



NUMA: Lock Cohorting

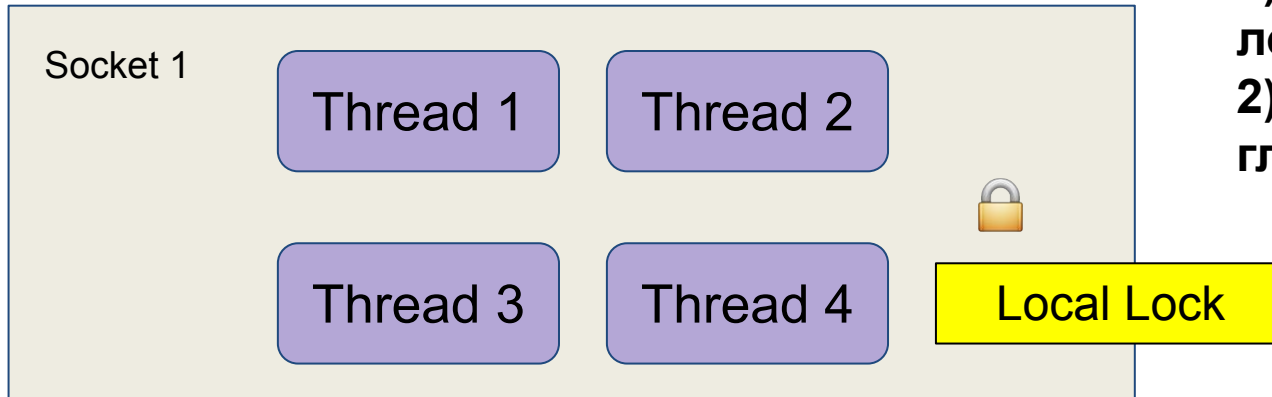


- 1) Сначала ждем локальный лок
- 2) Потом глобальный

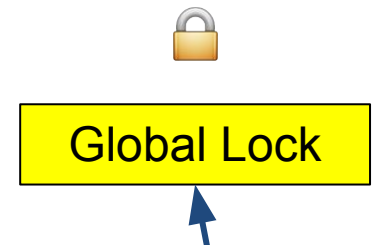
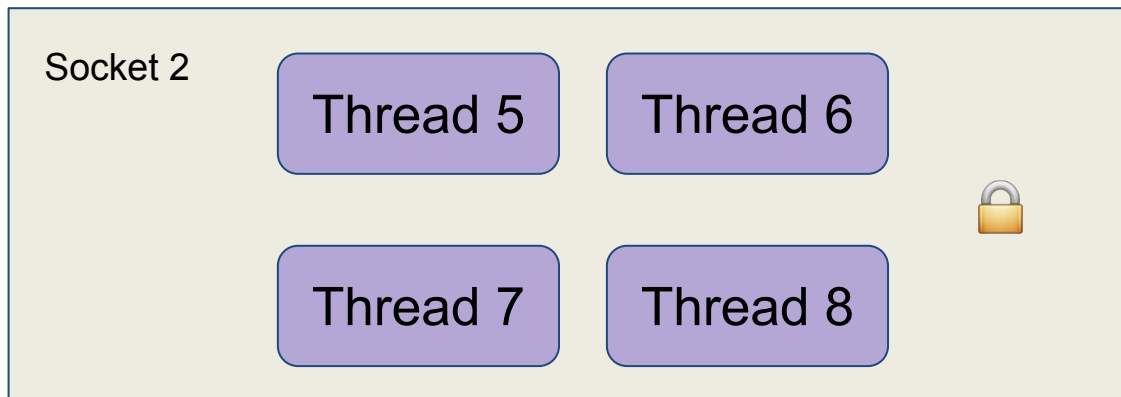


- 3) Сначала освободим локальный лок
- 4) Если есть еще один ждущий этот локальный лок, разбудим его

NUMA: Lock Cohorting



- 1) Сначала ждем локальный лок
- 2) Потом глобальный



- 3) Сначала освободим локальный лок
- 4) Если нет еще ждущего, то освободим глобальный лок

NUMA: Какие локи могут участвовать в когорте?

- **Global Lock: Thread-Oblivious**
 - Брать и освобождать могут любые потоки
 - Например TTS (Backoff) Lock
- **Local Lock: Cohort Detection**
 - Должен уметь понимать есть ли другие локи, которые его ждут
 - Например MCS Lock