Многопоточное Программирование: Flat-combining and friends

Виталий Аксёнов, ИТМО, aksenov@itmo.ru
Роман Елизаров, JetBrains, elizarov@gmail.com
Никита Коваль, JetBrains, ndkoval@ya.ru

ИТМО 2019

Motivation

- Представим себе структуру данных, которая охраняется одним локом. Например, стек или очередь. (Да хоть что угодно!)
- Представим себе высокую нагрузку на структуру данных.
- Каждый раз брать блокировку - дорого, там появляется contention.

Flat-combining. Idea.

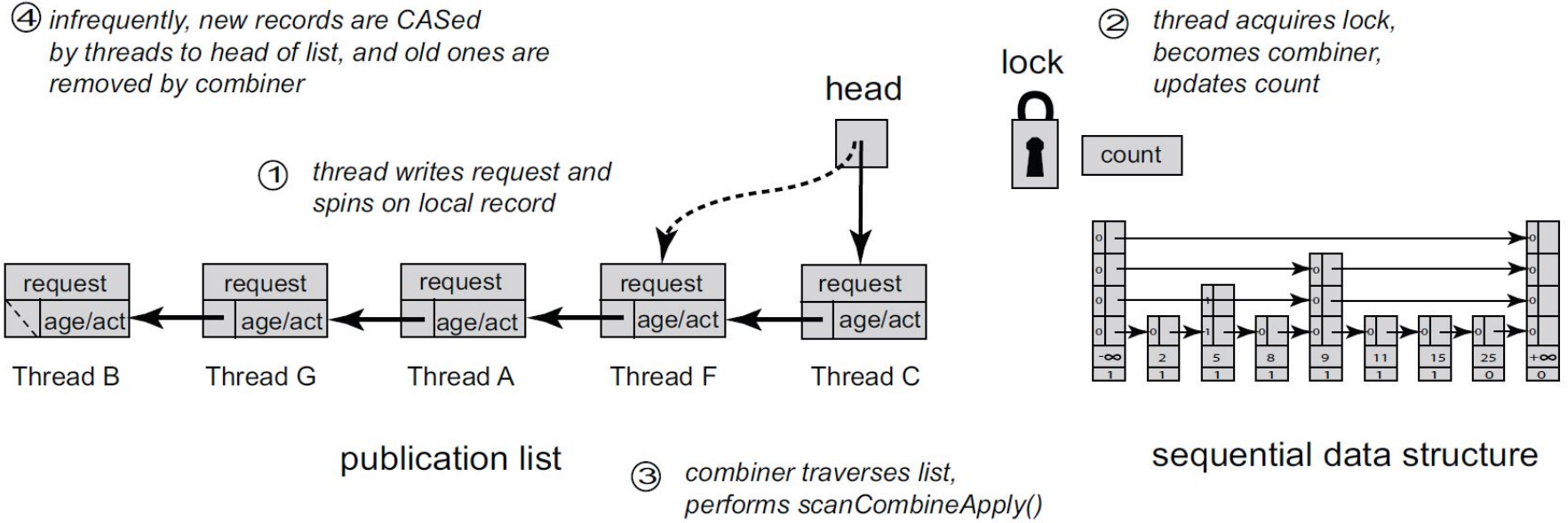
[Hendler et al., Flat Combining and the Synchronization-Parallelism Tradeoff, 2010]

Блокировку брать часто невыгодно. Давайте будем класть запрос на операцию в очередь, но не всегда его будем удалять.

Запросы получается должны быть активными и неактивными. Неактивные надо иногда удалять, а зачем они нам?

Как передавать лок? А можно и не передавать. Кто-то берёт лок, а потом проходит по всей очереди и выполняет запросы-операции, помечая их неактивными.

Flat-combining. Picture.



Flat-combining. Step 1.

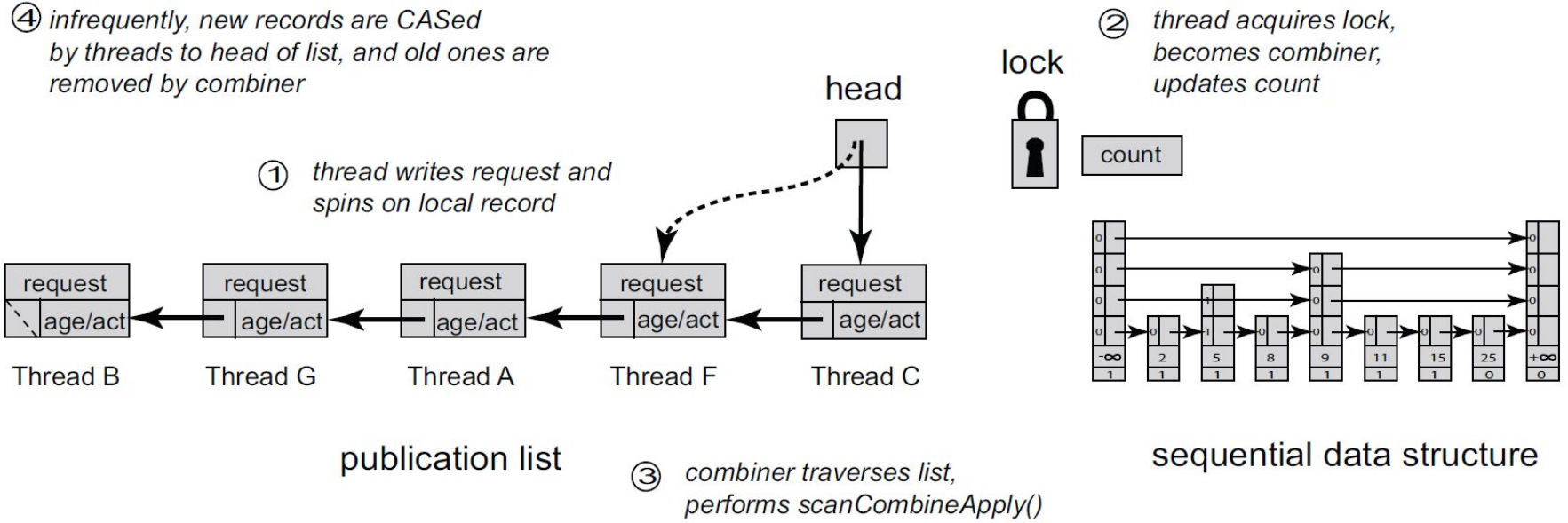
Берём `thread local` объект Request. Если его нет, то создаём и помещаем в список.

```
Request {
      Operation op
      Response res
      int age
      Request next
      boolean active
}
```

Помещаем в `op` операцию, которую хотим применить.

Если `active` помечено, то продолжаем в Шаг 2, иначе, нам нужно в Шаг 4 (проверить, что лежит в очереди, и пометить `active`).

Flat-combining. Picture.



Flat-combining. Step 2.

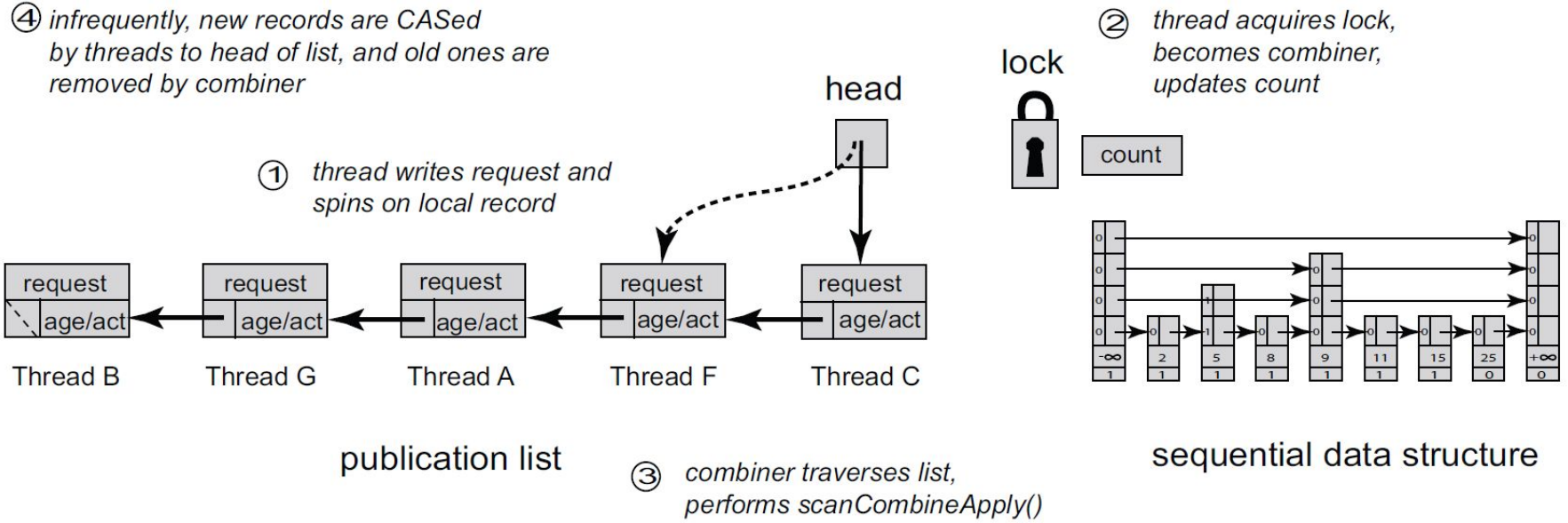
Проверяем глобальную блокировку.

Если она взята, то мы спинимся на `res` поле.

Иногда, снова проверяем взят ли лок и `active` поле нашей структуры Request. Если не `active`, то переходим в Шаг 4.

Если мы успешно взяли лок, то мы становимся `combiner`.

Flat-combining. Picture.

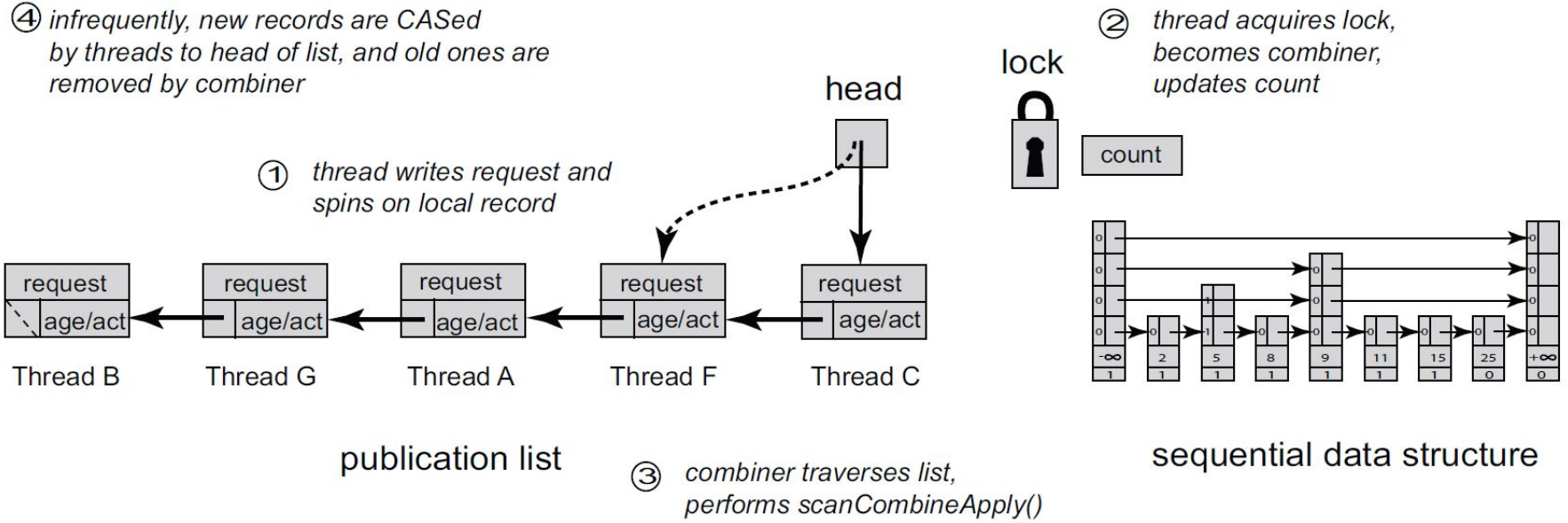


Flat-combining. Step 3.

Если мы combiner:

1. Увеличиваем счётчик `count`.
2. Проходим по всему списку. Выполняем каждый `active` запрос, выставляем его `age` в `count` и помечаем его как не `active`.
3. Если `count` свидетельствует о `cleanup` фазе, то мы проходим по списку ещё раз и удаляем из списка все не `active` запросы, у которых `age` намного меньше, чем `count`.
4. Отпустить блокировку.

Flat-combining. Picture.



Flat-combining. Step 4.

Если структура Request помечена как не active, мы делаем её active.

Далее проверяем, лежит ли она в списке. Если нет, то добавляем в конец.

Flat-combining. Thoughts.

- Можно чуть-чуть ускорить время работы.
  a. Вместо того, чтобы всем процессам бороться за блокировку, когда она освободилась. Combiner может выбрать следующего combiner из активных процессов.
  b. Также, при большой нагрузке, имеет смысл менять combiner очень часто: например, использовать тот же самый combiner, скажем, 5 раз.
- Также бесплатно получили ускорение от cache-friendliness. У нас операции выполняет один и тот же процесс (combiner), поэтому меньше cache-invalidation. (А в случае стека можно не всегда обращаться к структуре!)
- А иногда даже можно ускорить асимптотически. Например, приоритетную очередь на skip-list можно сделать не за `O(k log n)`, а быстрее. А некоторые структуры данных даже невозможно сделать конкуррентными… Например, pairing heap.
- А самое приятное - всё автоматически линеаризуется!

Flat-combining. Thoughts.

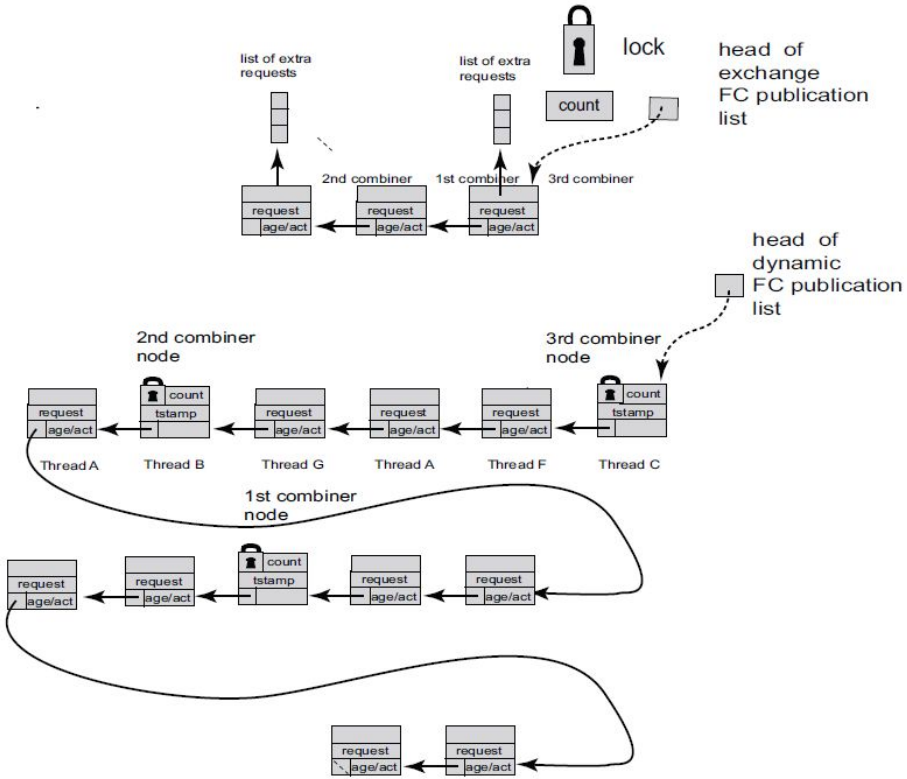
Остаётся одна напрягающая вещь. Процессы, которые спинятся на res, на самом деле ничего полезного не делают. А могли бы… Мы теряем вычислительные мощности просто так. Что делать?

- Иерархический Flat-combining [Hendler et al., Scalable Flat-combining Based Synchronous Queues, 2010].
- Parallel combining. [Aksenov et al., Parallel Combining …, 2018]

Hierarchical Flat-combining

- Задача: сделать стек или unfair rendez-vous queue.
- Преимущество этих структур данных в том, что можно запросы ставить в пары любым способом.
- Делаем двухуровневый flat-combining:
  - На первом уровне есть один большой список запросов. Этот список разбивается на подсписки, у каждого из которых есть combiner. Каждый combiner проходит по своему подсписку и выполняет запросы. У него осталась пачка запросов одного типа, и он их закидывает на следующий уровень.
  - На втором уровне обычный flat-combining. Теперь один запрос - это мета-запрос. Combiner-ы первого уровня борются за то, чтобы стать combiner второго уровня.

Hierarchical Flat-combining. Picture.



Parallel Combining

- Flat-combining - это такая своя версия блокировки. Какие ещё блокировки бывают? Конечно, RW-локи, которые позволяют операциям на read выполняться параллельно.
- Обычно структуры данных имеют два вида операций: read and write.
- Combiner может дать отмашку процессам с операциями типа read, чтобы они все выполнили их параллельно.
- А потом самому пройтись и выполнить операции типа write.
- Получается такая версия RW-лока, но, оказывается, работает быстрее.

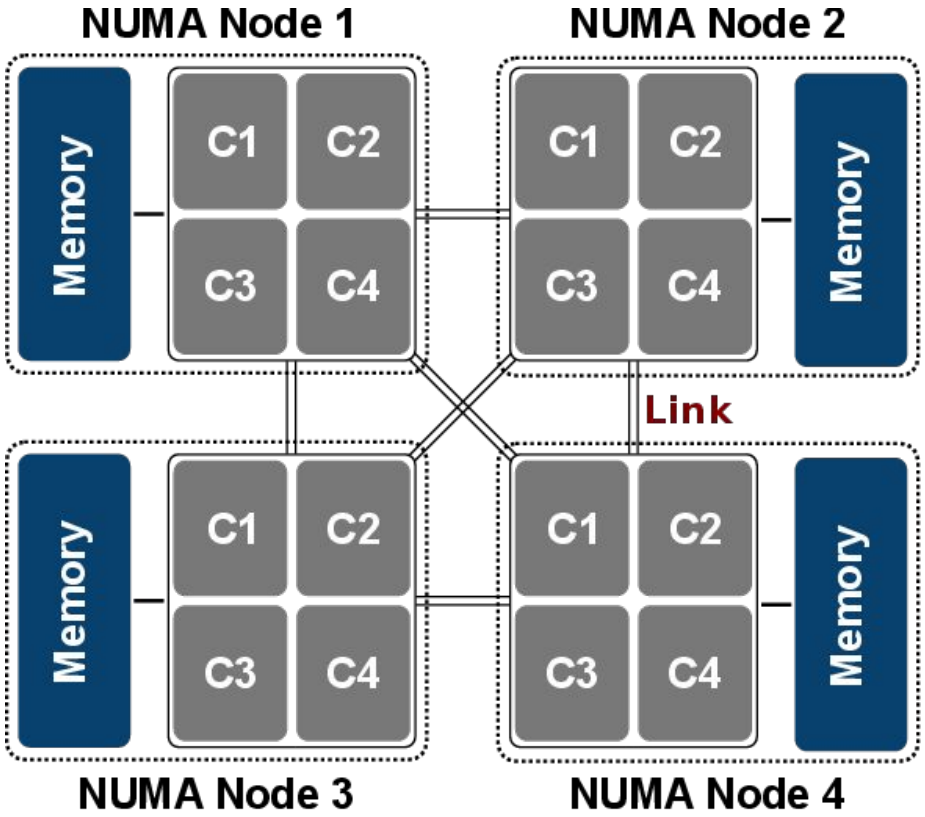
Parallel Combining

- Но это не основная идея Parallel Combining, а побочный спецэффект.
- Combiner собирает запросы от процессов, их агрегирует, а потом применяет.
- Неожиданно, но есть такие структуры данных, которые применяют сгруппированные запросы параллельно.
- Это parallel batched структуры данных.
- Написаны они на fork-join семантике, и обычно проще реализуемы, чем конкуррентные структуры, потому что запросы синхронизированы.

Flat-combining. Thoughts.

- Flat-combining работает не очень хорошо, как на самом деле кажется.
- Сравнения показывают, что Java-йный ReentrantLock (он же CLH Lock) работает лучше, чем самописный flat-combining. (Может у меня руки не оттуда? :-))
- Например, binary heap, ограждённый ReentrantLock, вообще летает. Может для priority queue и не нужно ничего выдумывать?
- Но вот самописный RW flat-combining уделывает ReadWriteLock без всяких сомнений, благодаря честной параллелизации.
- Где ещё используется flat-combining? Да, в блокировках, конечно! (Кто бы сомневался, да?)
- Рассмотрим необычные локи, а иерархические NUMA-friendly локи.

NUMA


NUMA Node 1
NUMA Node 2
Memory
C1
C2
C3
C4
C1
C2
C3
C4
Memory
Link
Memory
C1
C2
C3
C4
C1
C2
C3
C4
Memory
NUMA Node 3
NUMA Node 4

Hierarchical CLH Lock

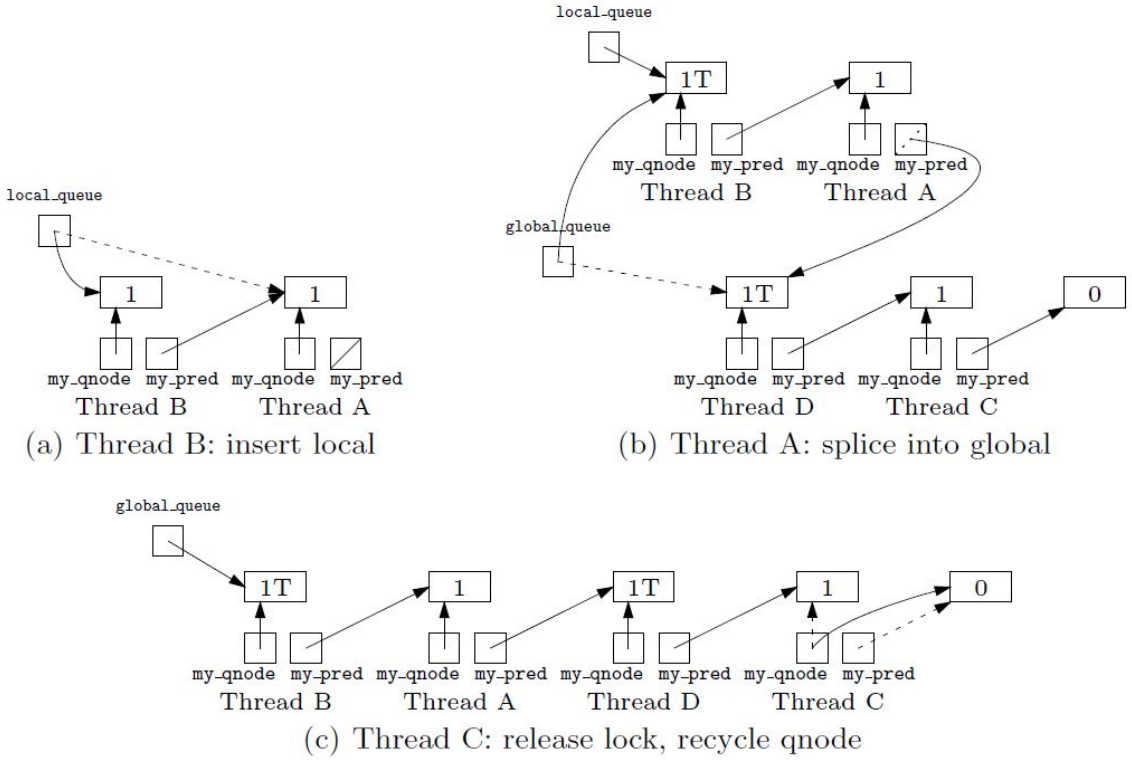
[Luchangco et al., A Hierarchical CLH Queue Lock, 2006]

- Есть локальные и глобальная CLH очереди.
- Если процесс становится первым в локальной очереди, он ждёт combining time, и потом вставляет всю свою очередь в глобальную.
- Если стал первым в глобальной очереди, то взял лок.

Hierarchical CLH Lock. Pseudocode.

```
qnode* acquire_HCLH_lock(local_q* lq, global_q* gq, qnode* my_qnode)
{
  // Splice my_qnode into local queue.
  do {
    my_pred = *lq;
  } while (!CAS(lq, my_pred, my_qnode));
  if (my_pred != NULL) {
    bool i_own_lock = wait_for_grant_or_cluster_master(my_pred);
    if (i_own_lock) {
      // I have the lock. Return qnode just released by previous owner.
      return my_pred;
    }
  }

  // At this point, I'm cluster master.  Give others time to show up.
  combining_delay();

  // Splice local queue into global queue.
  do {
    my_pred = *gq;
    local_tail = *lq;
  } while (!CAS(gq, my_pred, local_tail));

  // Inform successor that it is new master.
  local_tail->tail_when_spliced = true;

  // Wait for predecessor to release lock.
  while (my_pred->successor_must_wait);

  // I have the lock. Return qnode just released by previous owner.
  return my_pred;
}
```

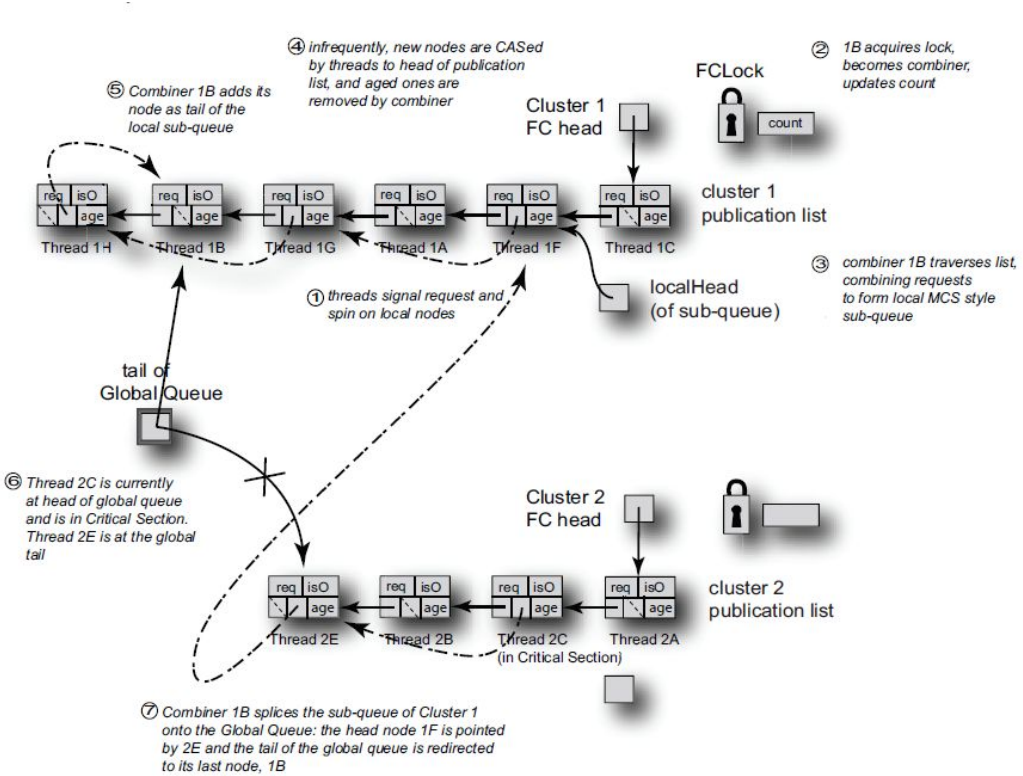
Hierarchical CLH Lock. Picture.



HCLH and Flat-combining NUMA Locks

- CAS на tail в CLH Lock приводит к большому contention.
- [Dice et al., Flat-combining NUMA Locks, 2011]
- Давайте локальную очередь сделаем на Flat-combining.
- Глобальная очередь у нас будет MCS.
  - Получается спин на своей ноде.
- Алгоритм получается чуть попроще.
- Работает в 2 раза лучше чем HCLH под большим contention

Flat-combining NUMA Lock. Picture.



What about NUMA Algorithms in General?

- Обычный способ сделать NUMA алгоритм:
  - Сделать копию структуры данных на каждой ноде.
  - Синхронизировать структуры каким-то образом.
- [Calciu et al., Black-box Concurrent Data Structures for NUMA Architectures, 2017]
  - Синхронизируем запросы с нод в общем логе.
  - Внутри каждой ноды делаем flat-combining.

Black-box DS for NUMA