



ETU "LETI"
SAINT PETERSBURG ELECTROTECHNICAL UNIVERSITY

Средства повышения масштабируемости параллельных программ на основе потокобезопасных структур данных с ослабленной семантикой выполнения операций

Студент гр. 4307:

Табаков А. В.

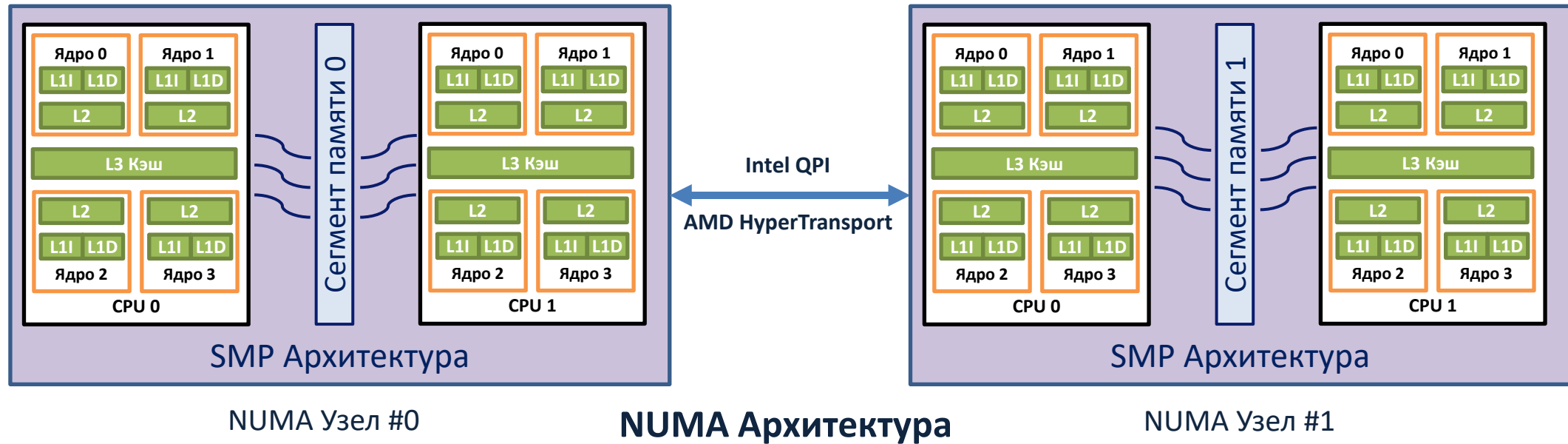
Руководитель: к. т. н., доцент

Пазников А. А.

СОДЕРЖАНИЕ

- Обеспечение синхронизации в параллельных программах
- Ослабленные структуры данных
- Оптимизация алгоритмов вставки и удаления структуры Multiqueues
- Алгоритм балансировки структуры Multiqueues
- Экспериментальное исследование оптимизированных алгоритмов
- Построение ослабленных структур данных на основе циклических списков
- Экспериментальное исследование ослабленной циклической очереди

СИНХРОНИЗАЦИИ В ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ



Одна из ключевых задач параллельного программирования – **масштабируемая синхронизация** потоков при обращении к разделяемым областям памяти

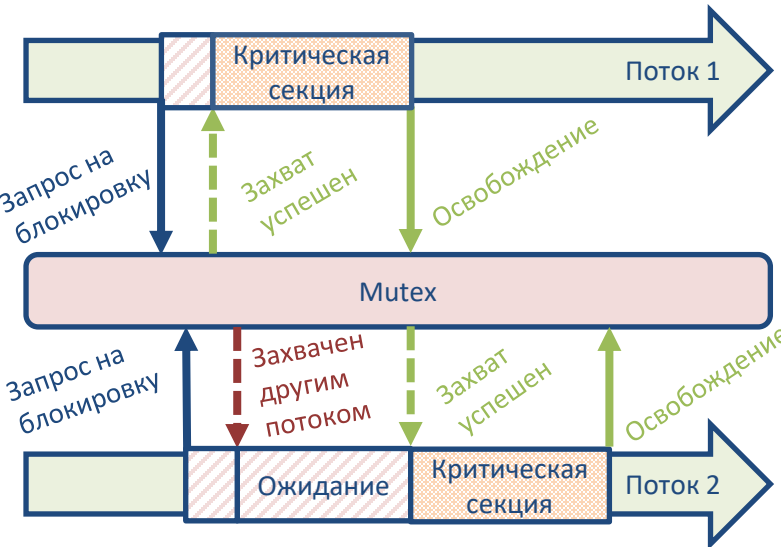
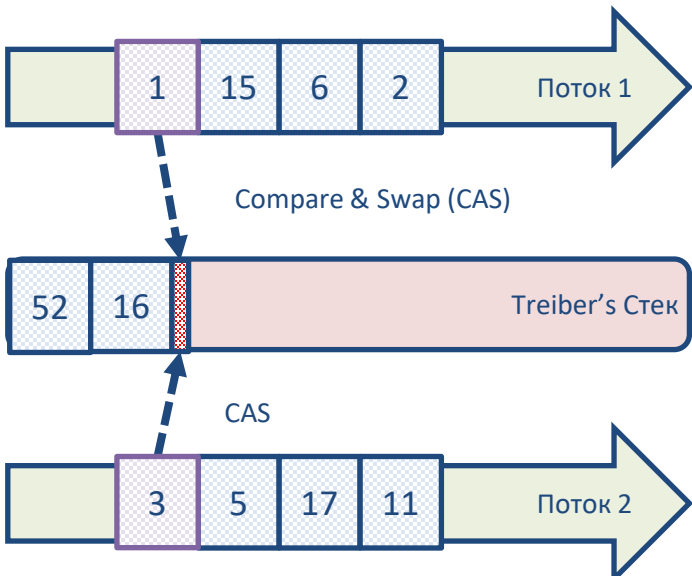
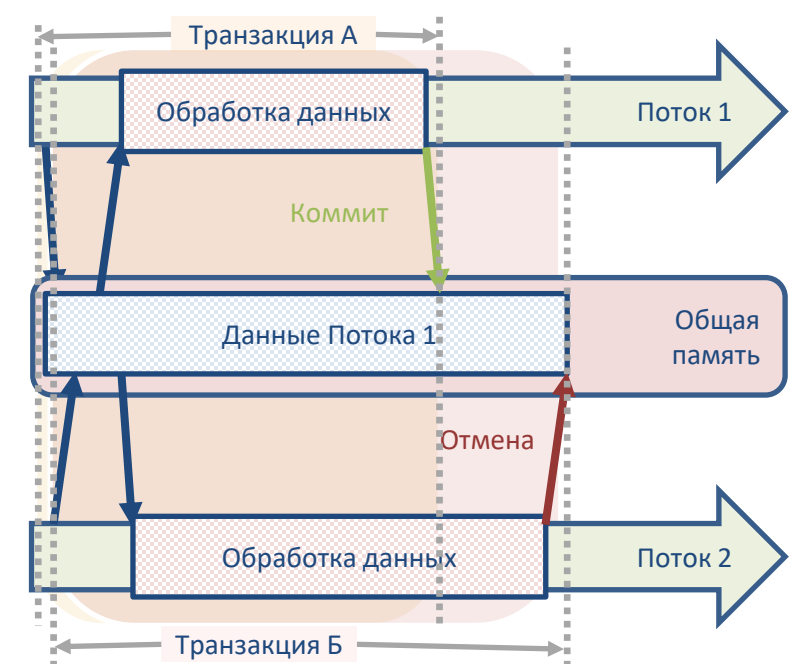
Аспекты безопасности (safeness)

- Обеспечение потокобезопасности (thread-safety)
- Исключение тупиковых ситуаций (deadlocks, livelocks)
- Соответствие критериям корректности: линеаризуемость (linearizability)

Аспекты масштабируемости (scalability)

- Состязание потоков (contention, oversubscription)
- Узкие места (bottlenecks, hotspots)
- Эффективность использования кэш-памяти (invalidations, cache misses)
- Голодание потоков (thread starvation), проблема справедливого доступа к ресурсам (fairness)

МЕТОДЫ СИНХРОНИЗАЦИИ

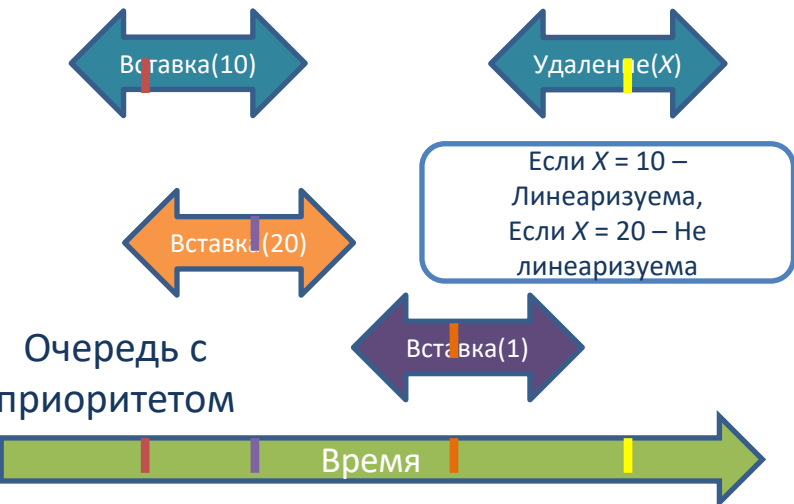
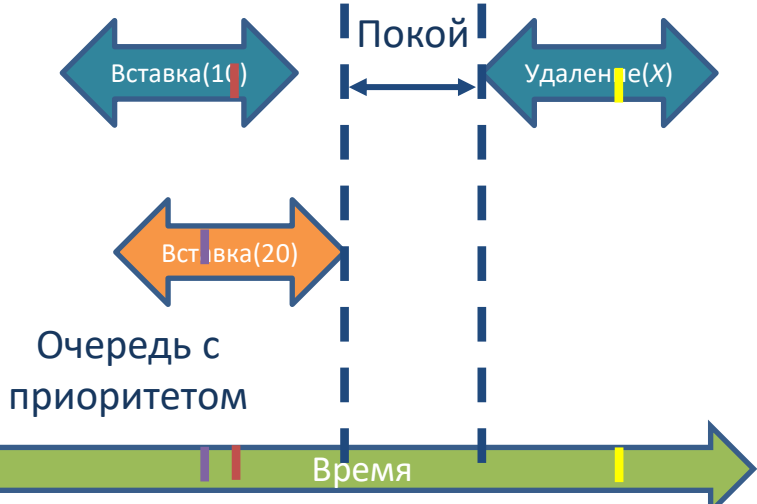
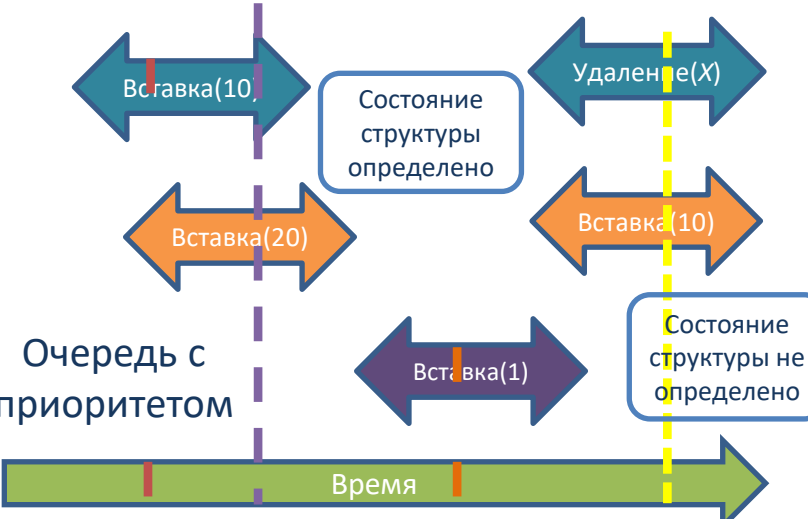
Блокировки (locks, mutexes, spinlocks)	Неблокируемые структуры и алгоритмы (non-blocking)	Транзакционная память (transactional memory)
<p>Блокировки организуют последовательный доступ к общему ресурсу. Доступ на одновременную работу с данным разделяемым ресурсом предоставляется только одному потоку¹.</p> 	<p>Параллельное выполнение программы гарантирует непрерывный прогресс². Существует три класса гарантии прогресса: wait-free (без остановок), lock-free (без блокировок), obstruction-free (без препятствий).</p> 	<p>Транзакционная память выделяет группы инструкций в атомарные транзакции – конечные последовательности операций транзакционного чтения/записи памяти³.</p> 

¹ Herlihy M., Shavit N. The art of multiprocessor programming. – Morgan Kaufmann, 2011. – С. 219-225.

² Goetz B. et al. Java concurrency in practice. – Pearson Education, 2006.

³ Shavit N., Touitou D. Software transactional memory //Distributed Computing. – 1997. – Т. 10. – №. 2. – С. 99-116.

КРИТЕРИИ КОРРЕКТНОСТИ ВЫПОЛНЕНИЯ ПАРАЛЛЕЛЬНЫХ ОПЕРАЦИЙ

<p>Линеаризуемость (Linearizability)</p>	<p>Согласованность покоя (quiescent consistency)</p>	<p>Квази-линеаризуемость (quasi-linearizability)</p>
<p>Данный критерий устанавливает, что параллельное выполнение операций должно быть эквивалентно некоторому корректному последовательному выполнению данных операций.⁴</p> 	<p>Согласованность покоя предполагает, что порядок выполнения непересекающихся операций должен соответствовать их вызову в настоящем времени, в то же время, порядок выполнения пересекающихся операций может быть изменён.⁵</p> 	<p>Во время выполнения некоторых операций могут произойти несколько событий, одновременно изменяющие структуру данных таким образом, что после выполнения одной из операций состояние структуры данных не определено.⁶</p> 

⁴ Herlihy M., Shavit N. The art of multiprocessor programming. – Morgan Kaufmann, 2011. – С. 219-225.

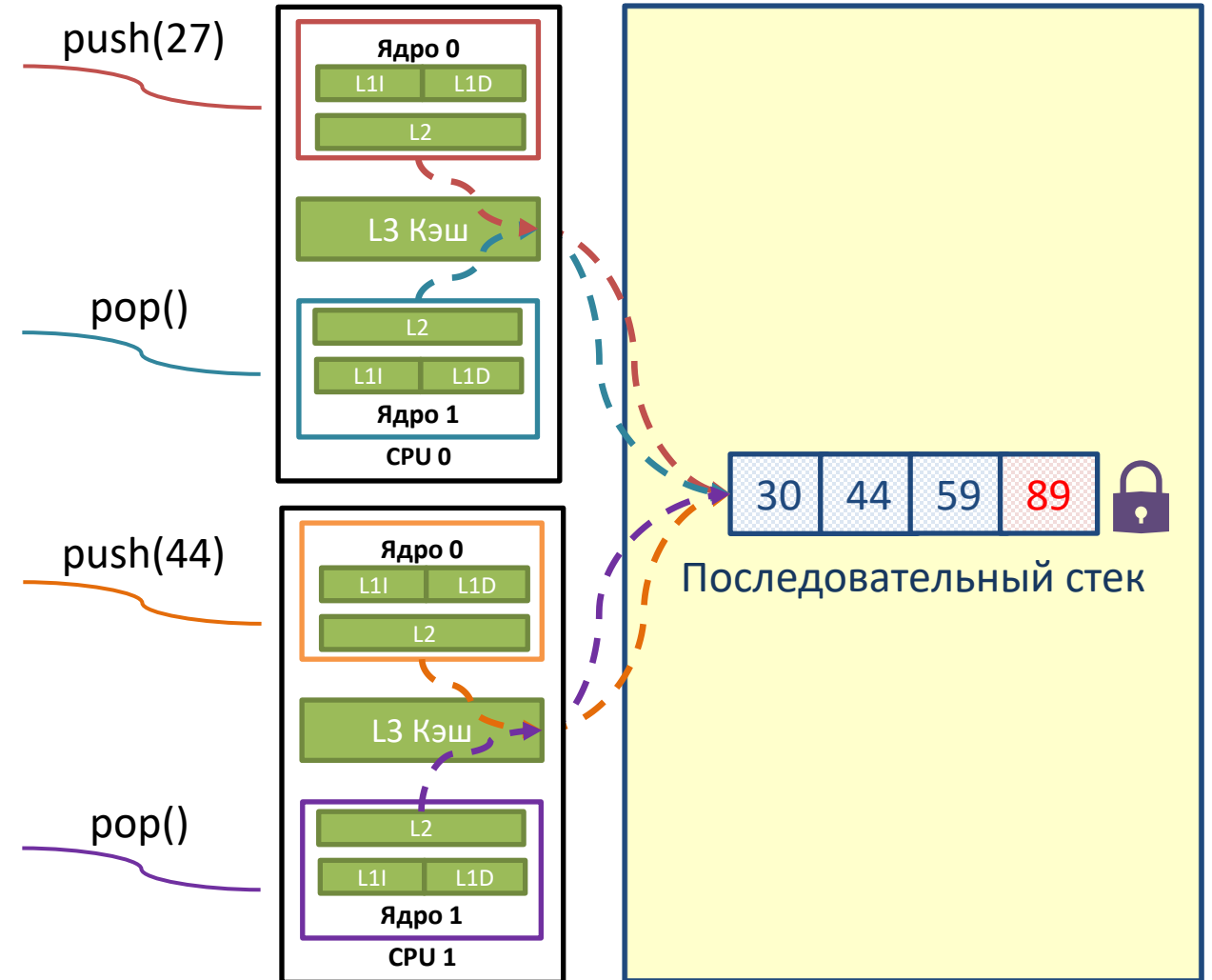
⁵ Derrick J. et al. Quiescent consistency: Defining and verifying relaxed linearizability //International Symposium. – Springer, 2014. – С. 200-214.

⁶ Afek Y., Korland G., Yanovsky E. Quasi-linearizability: Relaxed consistency for improved concurrency – Springer, 2010. – С. 395-410.

РАЗДЕЛЯЕМЫЙ СТЕК СТРОГОЙ СЕМАНТИКИ

Существуют реализации **потокобезопасного стека строгой семантики (strong concurrent stack)**, как на базе блокировок, так и без использования блокировок. Их **недостатки**:

- Потоки имеют единую точку выполнения операций (bottleneck, hotspot), высокий уровень состязательности (contention)
- Невозможно выполнить действия параллельно несколькими потоками
- Голодание потоков (thread starvation)



СОДЕРЖАНИЕ

- Обеспечение синхронизации в параллельных программах
- Ослабленные структуры данных
- Оптимизация алгоритмов вставки и удаления структуры Multiqueues
- Алгоритм балансировки структуры Multiqueues
- Экспериментальное исследование оптимизированных алгоритмов
- Построение ослабленных структур данных на основе циклических списков
- Экспериментальное исследование ослабленной циклической очереди

ОСЛАБЛЕННЫЙ НА k ЭЛЕМЕНТОВ СТЕК⁷

k-relaxed stack

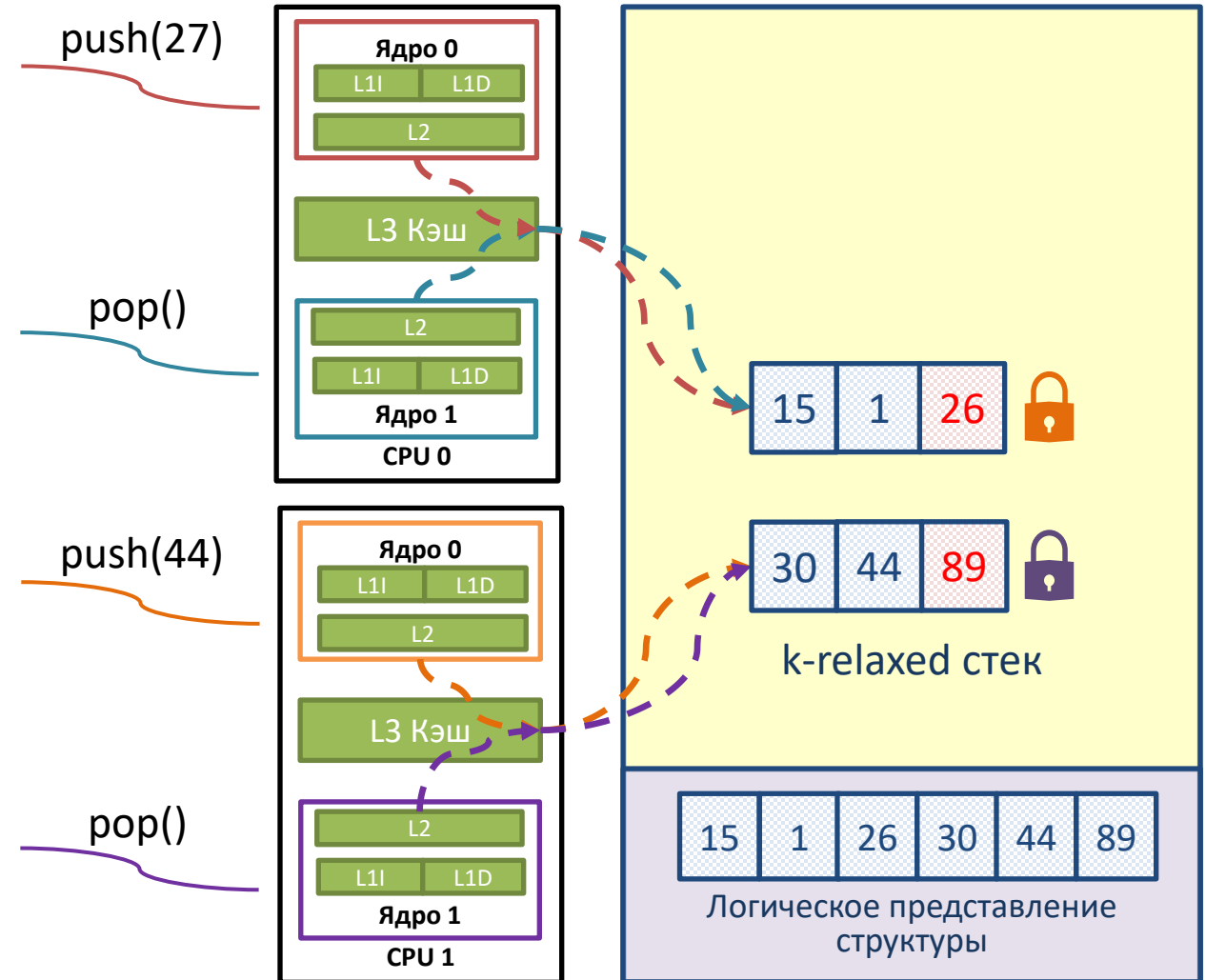
Преимущества ослабленных структур:

- Потоки меньше состязаются за ресурс
- Высокая масштабируемость
- Высокая пропускная способность (низкая латентность)

Недостатки ослабленных структур:

- Операции не линейризуемы
- Результат операции находится в определённой области значений (структура данных может не соответствовать строгой спецификации)

В практике параллельного программирования **данными недостатками можно** пренебречь для повышение масштабируемости.

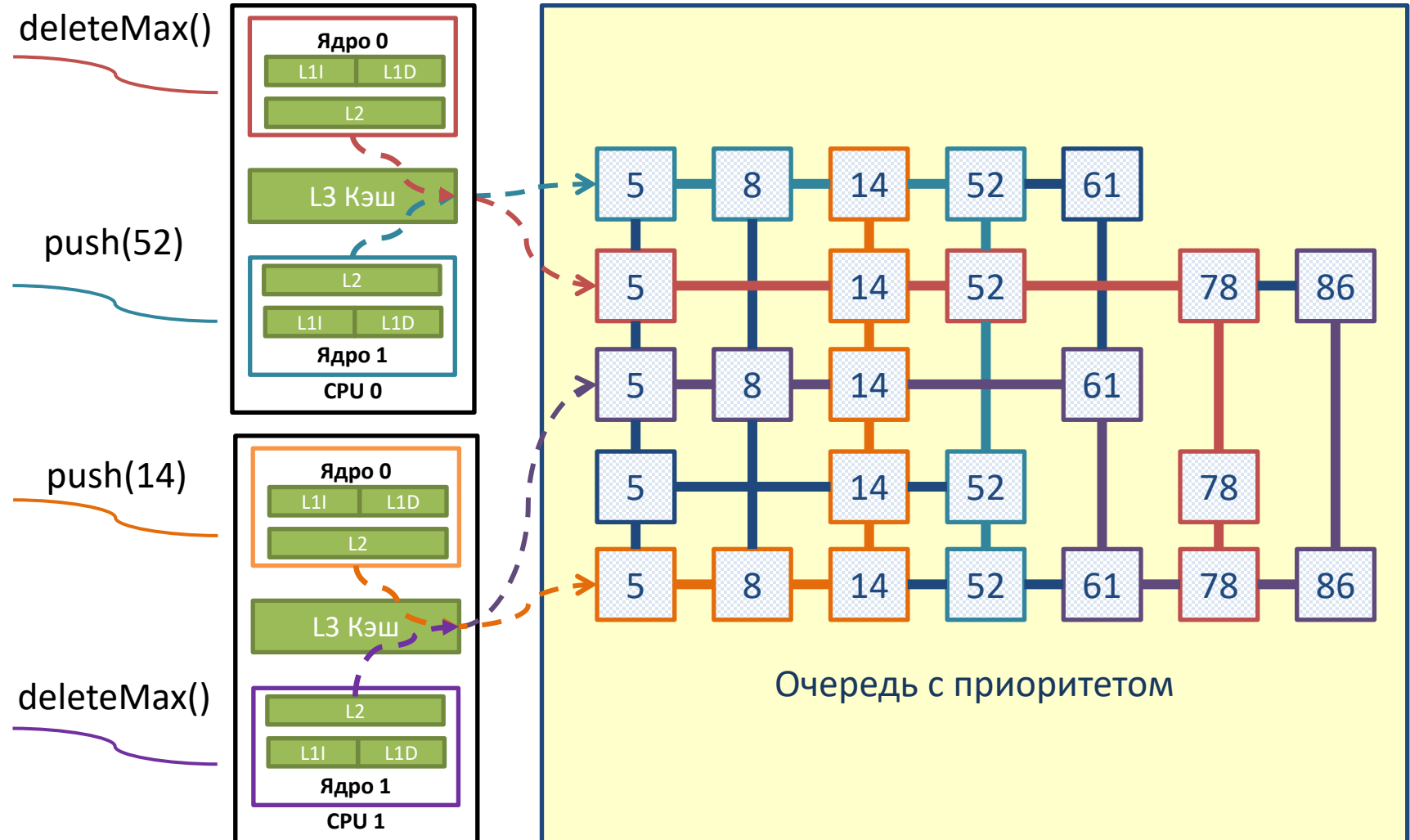


⁷ Talmage E., Welch J. L. Improving average performance by relaxing distributed data structures // Distributed Computing. – Springer, Berlin, 2014. – С. 421-438

ОСЛАБЛЕННАЯ ОЧЕРЕДЬ С ПРИОРИТЕТОМ НА ОСНОВЕ СПИСКА С ПРОПУСКАМИ⁸

SprayList

- Узлы верхних уровней создаются с некоторой случайной вероятностью
- Потоки имеют доступ ко всему множеству точек исполнения операции над структурой

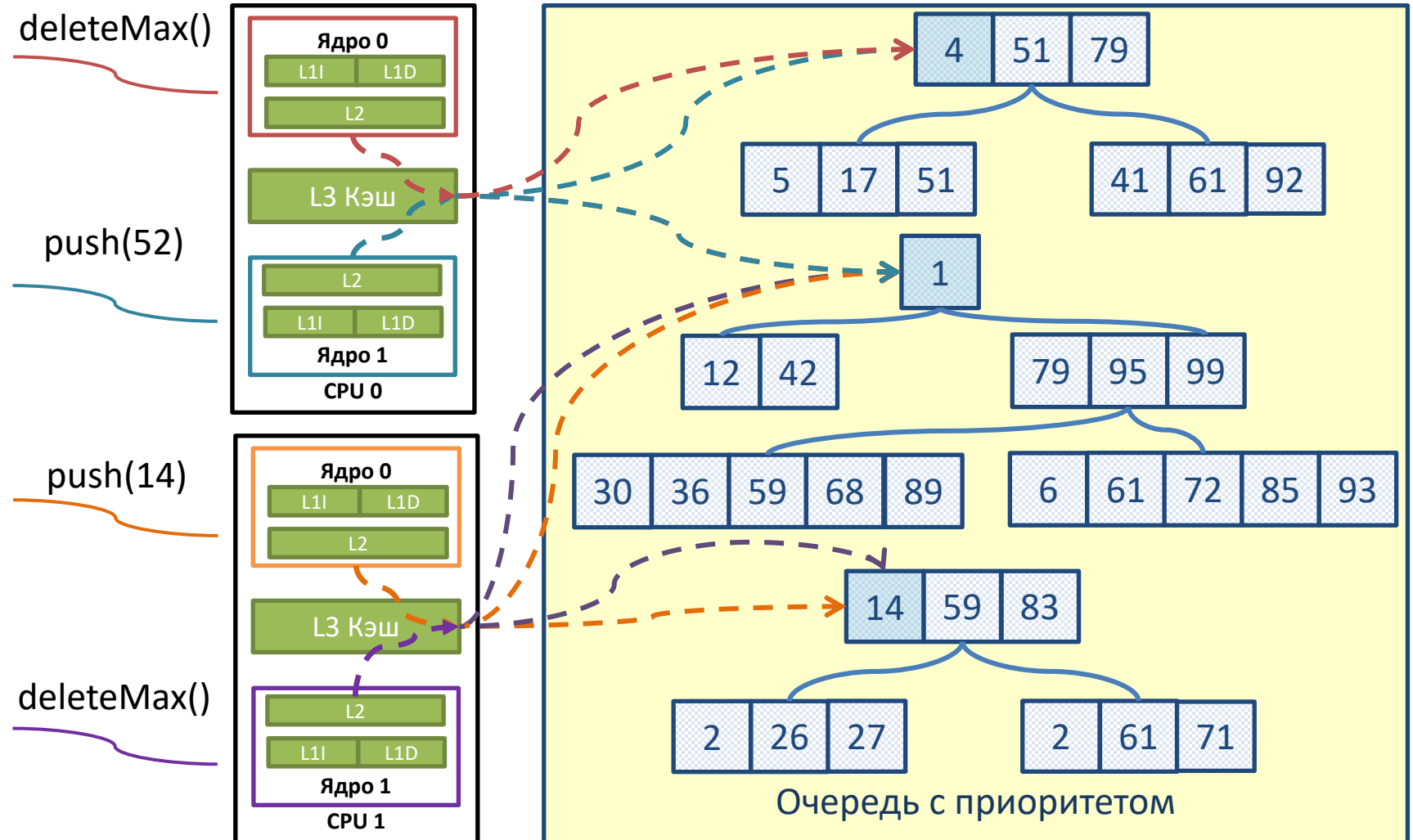


⁸ Alistarh D. et al. The spraylist: A scalable relaxed priority queue // Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. – 2015. – С. 11-20.

ОСЛАБЛЕННАЯ ОЧЕРЕДЬ С ПРИОРИТЕТОМ НА ОСНОВЕ ДЕРЕВА СО СЛИЯНИЕМ⁹

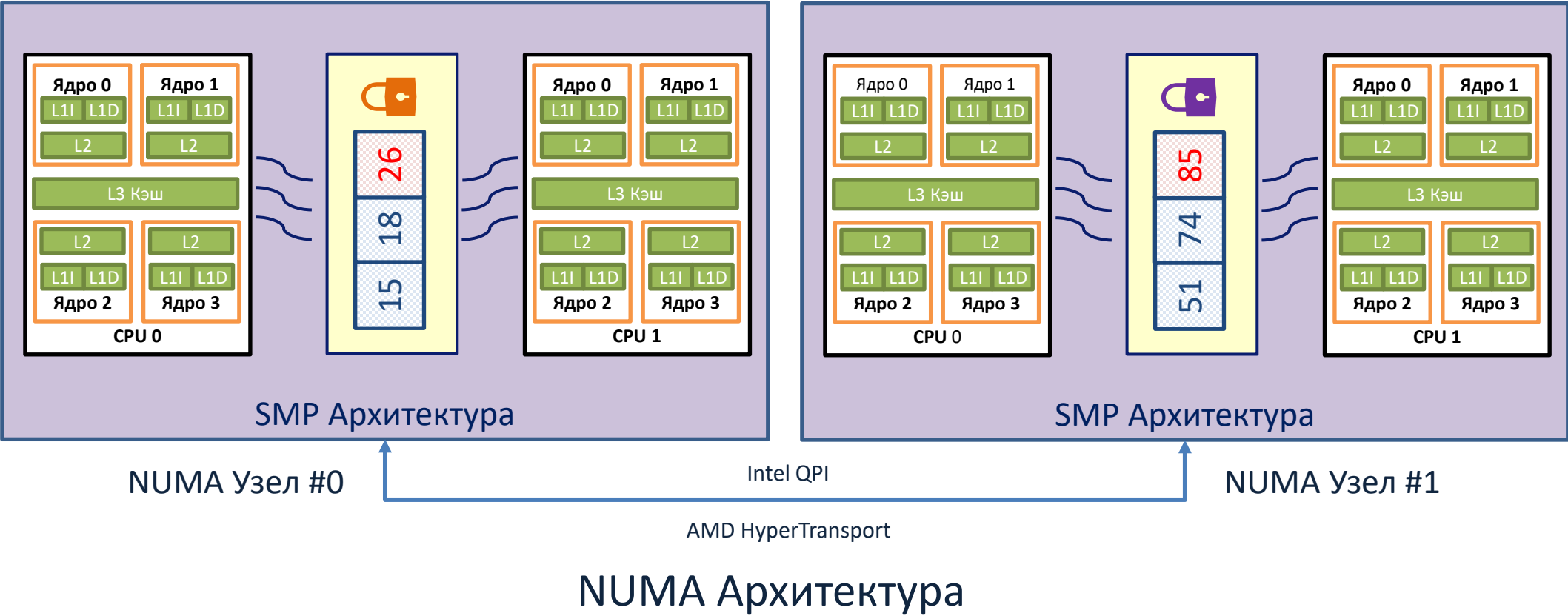
k-LSM

- Поток могут обращаться как к локальным деревьям, так и к общему
- Общее дерево создаётся путём слияния локальных деревьев



⁹ Wimmer M. et al. The lock-free k-LSM relaxed priority queue //ACM SIGPLAN Notices. – 2015. – T. 50. – №. 8. – С. 277-278.

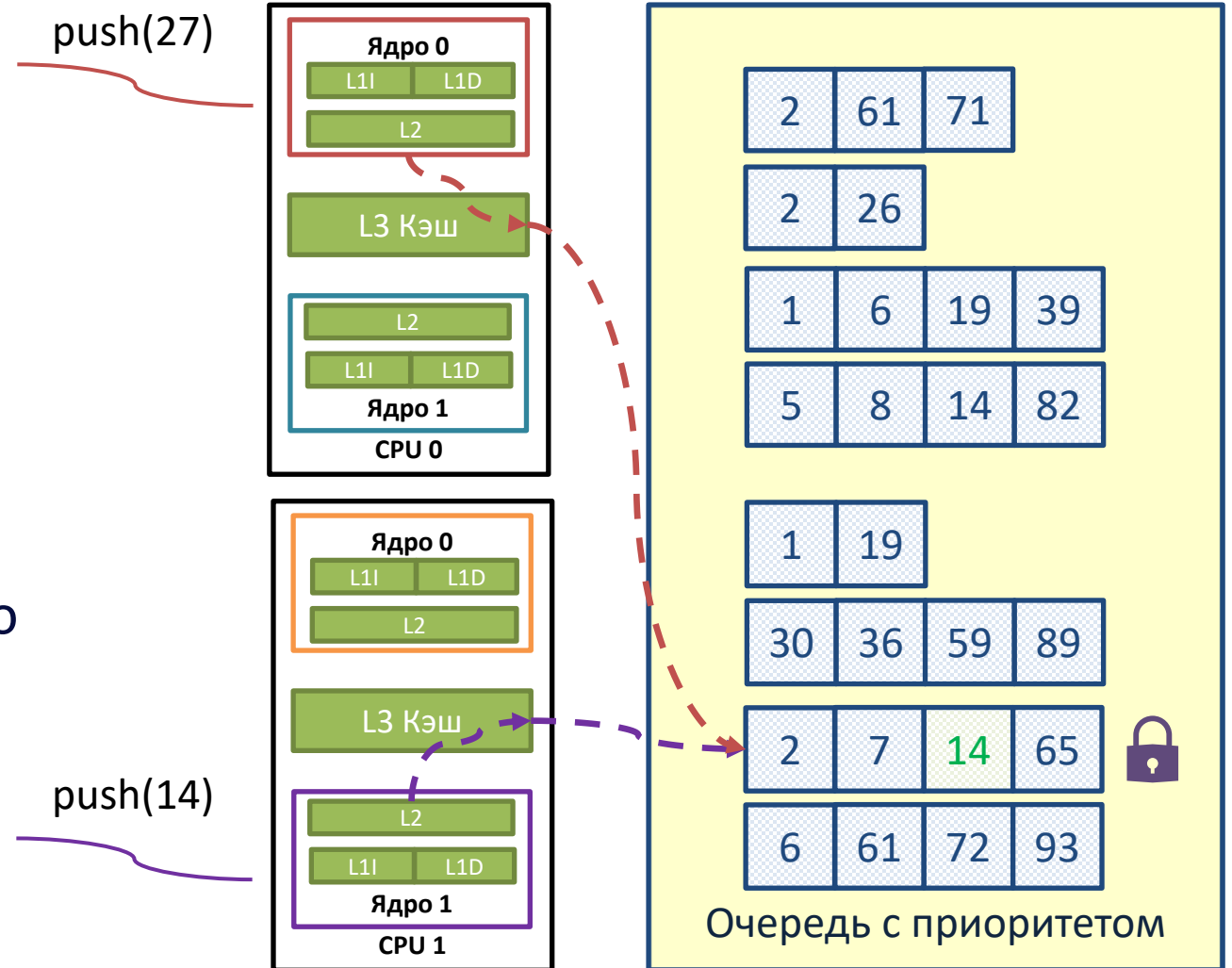
ОСЛАБЛЕННЫЕ СТРУКТУРЫ ДАННЫХ В МНОГОПРОЦЕССОРНЫХ СИСТЕМАХ



ОСЛАБЛЕННАЯ ОЧЕРЕДЬ С ПРИОРИТЕТОМ

Операция вставки

1. Поток случайным образом выбирает одну очередь из множества¹⁰
2. Попытка заблокировать мьютекс очереди
3. В случае успешной блокировки выполняется операция над очередью
4. В ином случае алгоритм повторяется сначала

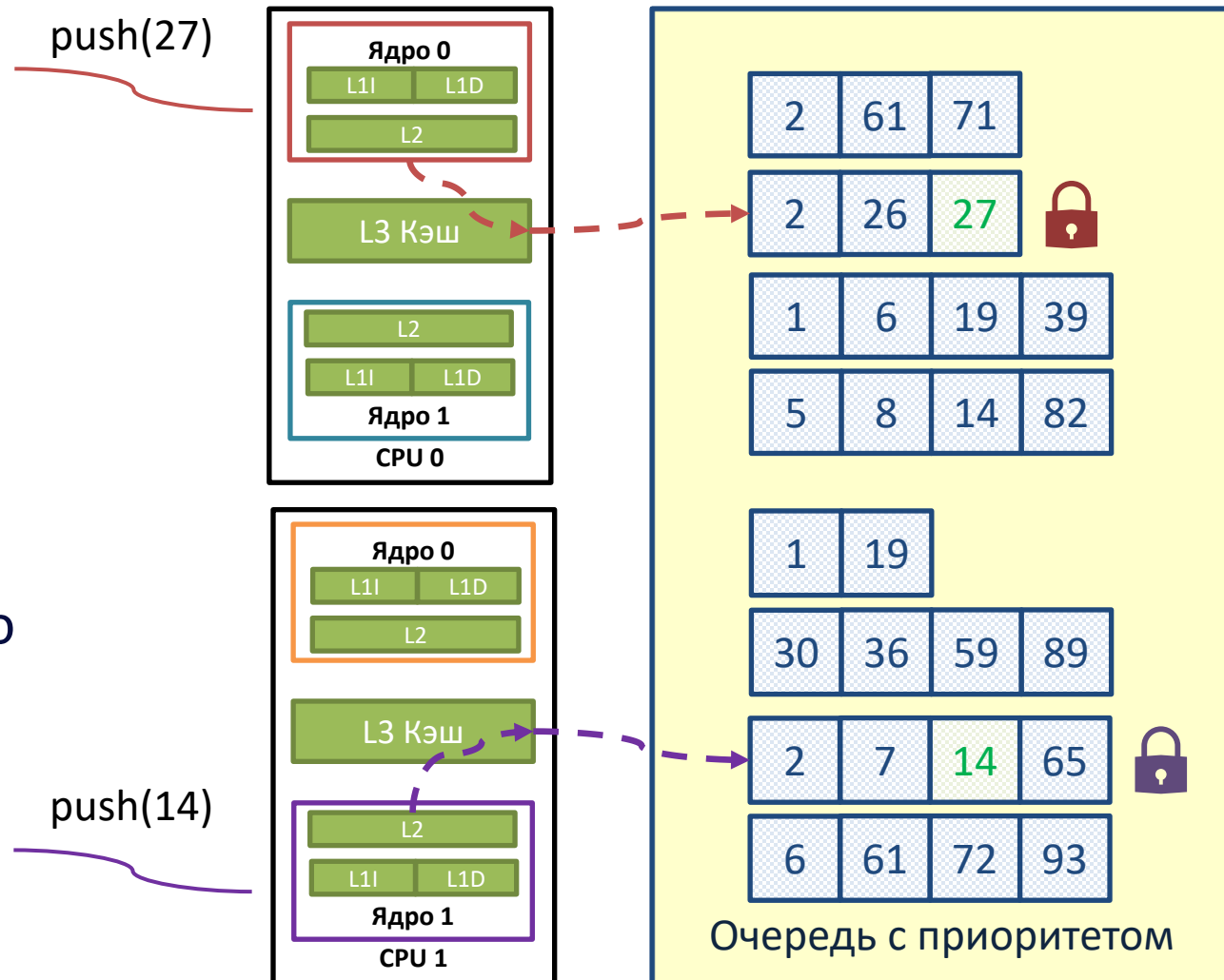


¹⁰Rihani H., Sanders P., Dementiev R. Multiqueues: Simpler, faster, and better relaxed concurrent priority queues //arXiv preprint arXiv:1411.1209. – 2014.

ОСЛАБЛЕННАЯ ОЧЕРЕДЬ С ПРИОРИТЕТОМ

Операция вставки

1. Поток случайным образом выбирает одну очередь из множества¹⁰
2. Попытка заблокировать мьютекс очереди
3. В случае успешной блокировки выполняется операция над очередью
4. В ином случае алгоритм повторяется сначала



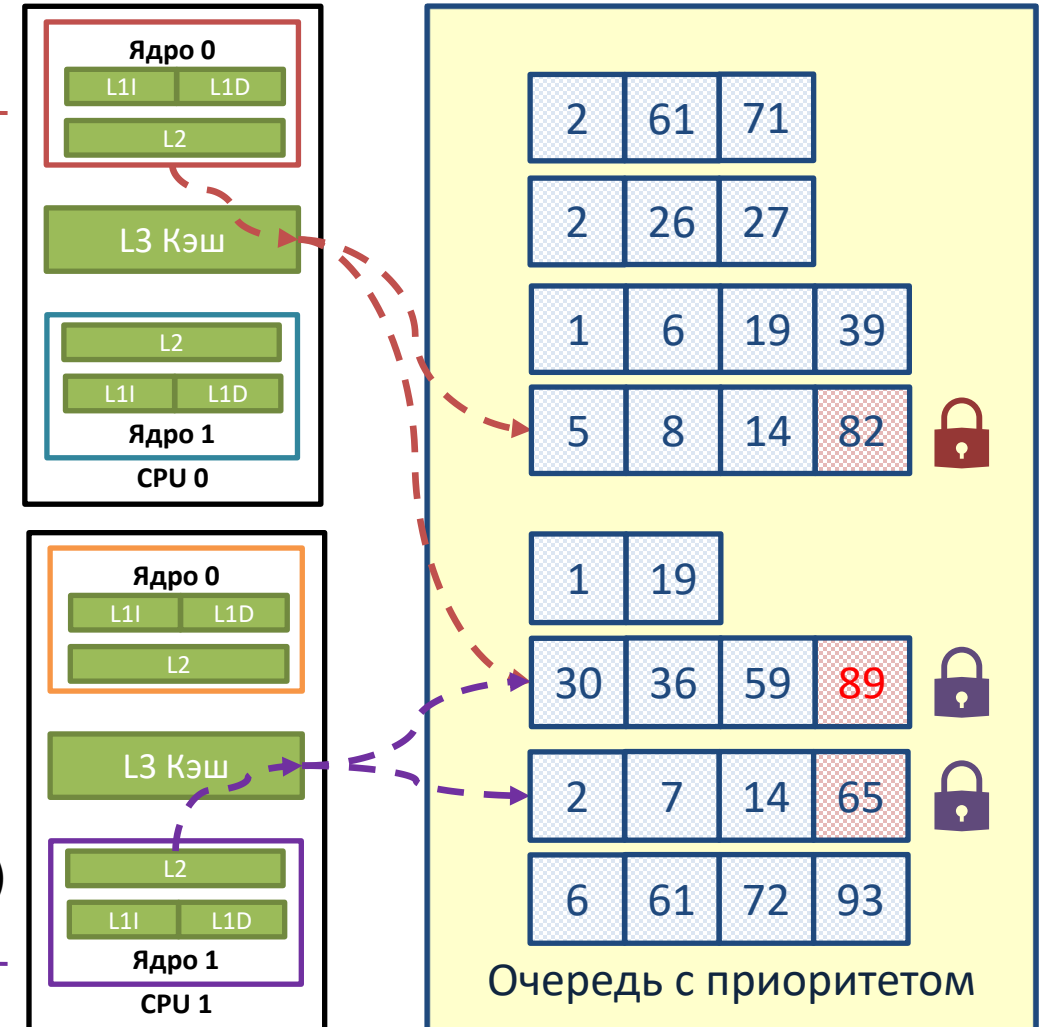
¹⁰Rihani H., Sanders P., Dementiev R. Multiqueues: Simpler, faster, and better relaxed concurrent priority queues //arXiv preprint arXiv:1411.1209. – 2014.

ОСЛАБЛЕННАЯ ОЧЕРЕДЬ С ПРИОРИТЕТОМ

Операция удаления максимального

1. Поток случайным образом выбирает 2 очереди из множества¹⁰
2. Попытка заблокировать их мьютексы, при неудачной попытке захвата мьютекса одной из очередей, выбирается другая структура
3. После захвата мьютексов сравниваются максимальные элементы выбранных очередей и удаляется максимальный `deleteMax()`

`deleteMax()`



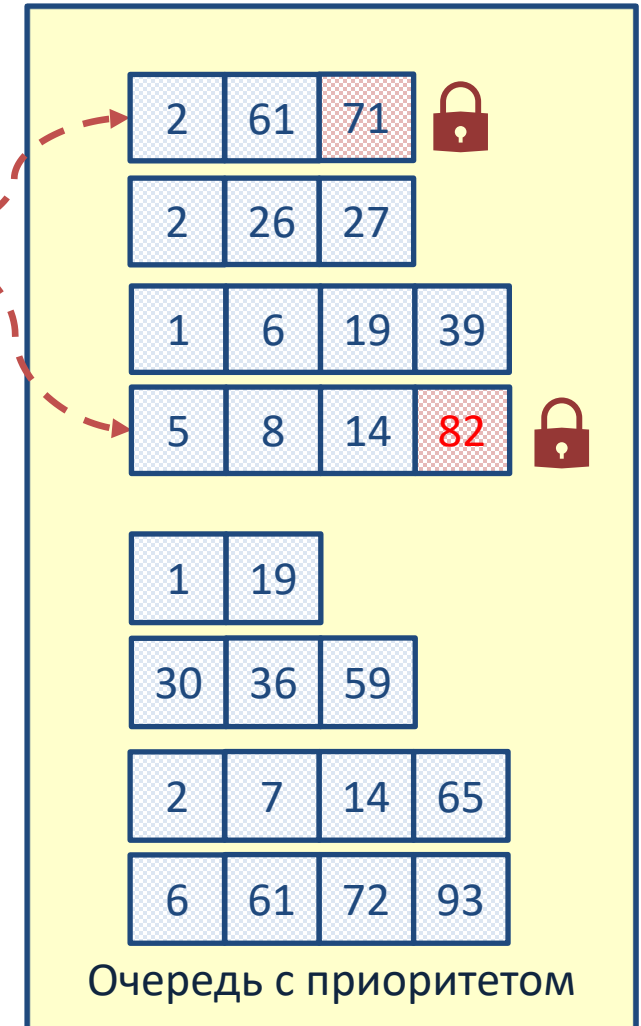
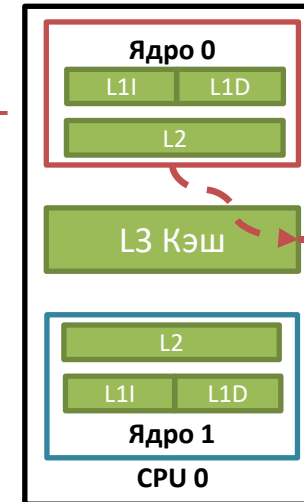
¹⁰Rihani H., Sanders P., Dementiev R. Multiqueues: Simpler, faster, and better relaxed concurrent priority queues //arXiv preprint arXiv:1411.1209. – 2014.

ОСЛАБЛЕННАЯ ОЧЕРЕДЬ С ПРИОРИТЕТОМ

Операция удаления максимального

1. Поток случайным образом выбирает 2 очереди из множества¹⁰
2. Попытка заблокировать их мьютексы, при неудачной попытке захвата мьютекса одной из очередей, выбирается другая структура
3. После захвата мьютексов сравниваются максимальные элементы выбранных очередей и удаляется максимальный `deleteMax()`

`deleteMax()`



¹⁰Rihani H., Sanders P., Dementiev R. Multiqueues: Simpler, faster, and better relaxed concurrent priority queues //arXiv preprint arXiv:1411.1209. – 2014.

СОДЕРЖАНИЕ

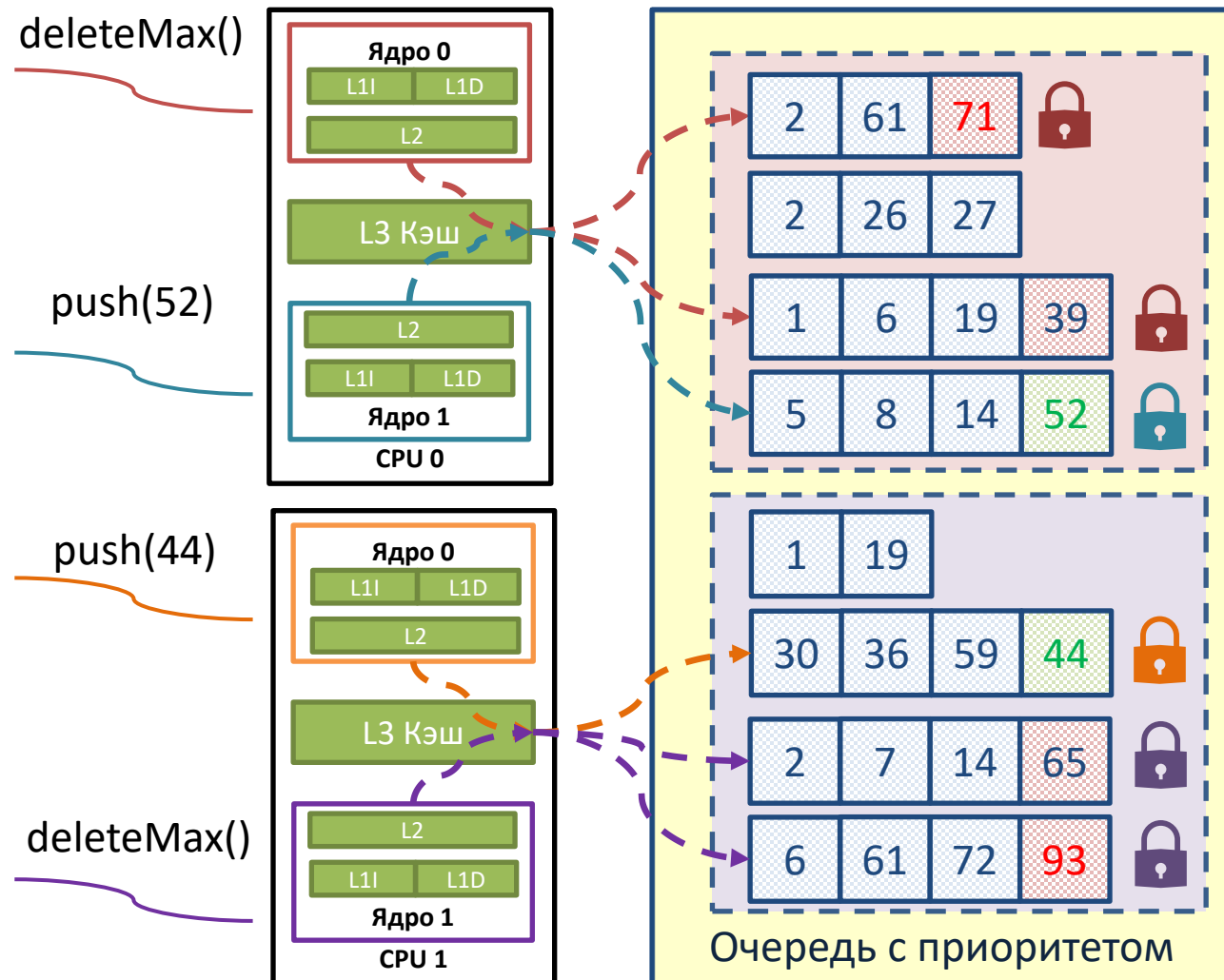
- Обеспечение синхронизации в параллельных программах
- Ослабленные структуры данных
- **Оптимизация алгоритмов вставки и удаления структуры Multiqueues**
- Алгоритм балансировки структуры Multiqueues
- Экспериментальное исследование оптимизированных алгоритмов
- Построение ослабленных структур данных на основе циклических списков
- Экспериментальное исследование ослабленной циклической очереди

ОПТИМИЗАЦИЯ ОПЕРАЦИИ ВСТАВКИ И УДАЛЕНИЯ

ORTHALFINSERT и ORTHALFDELETE

Локализация области поиска незаблокированной очереди среди множества очередей

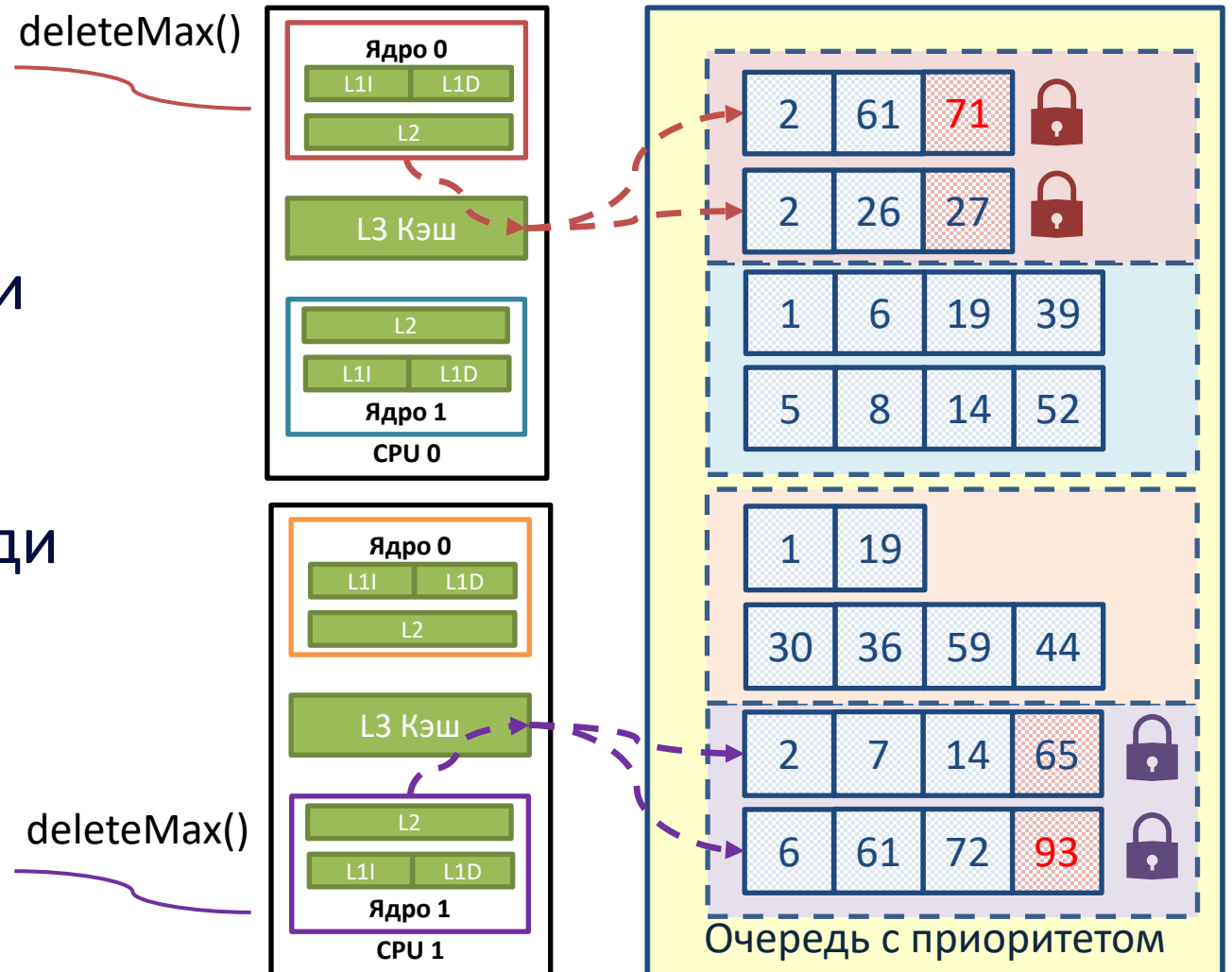
Первая половина потоков выполняет поиск среди первой половины множества очередей, вторая – из второй



ОПТИМИЗАЦИЯ ОПЕРАЦИИ УДАЛЕНИЯ ОПТЕХАСТDELETE

Локализация области поиска
незаблокированной очереди среди
«привязанных» очередей

Первый поиск осуществляется среди
локального множества очередей,
если он неудачен, то выполняется
поиск среди половины очередей



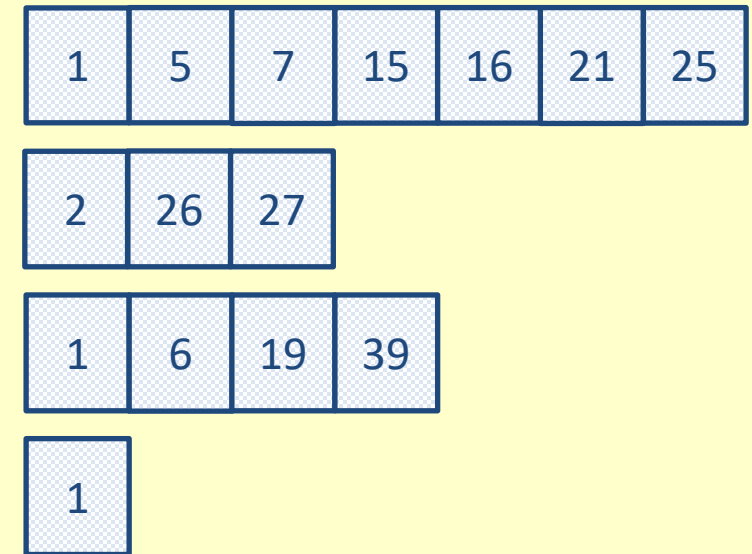
Содержание

- Обеспечение синхронизации в параллельных программах
- Ослабленные структуры данных
- Оптимизация алгоритмов вставки и удаления структуры Multiqueues
- Алгоритм балансировки структуры Multiqueues
- Экспериментальное исследование оптимизированных алгоритмов
- Построение ослабленных структур данных на основе циклических списков
- Экспериментальное исследование ослабленной циклической очереди

АЛГОРИТМ БАЛАНСИРОВКИ

Балансировка необходима во избежание появления пустых очередей.
Алгоритм запускается после каждой 10^6 операции над структурой

```
1  q1 = FindLargestQueue();
2  q2 = FindShortestQueue();
3  if q1.size() > AvgSizeOfAllQueues()* $\alpha$  then
4    Lock(q1);
5    q2IsLocked = LockWithTimeout(q2);
6    if q2IsLocked then
7      sizeToTransfer = q1.size()* $\omega$ 
8      TransferElements(q1, q2, sizeToTransfer)
9      Unlock(q2);
10   end
11   Unlock(q1);
12 end
```

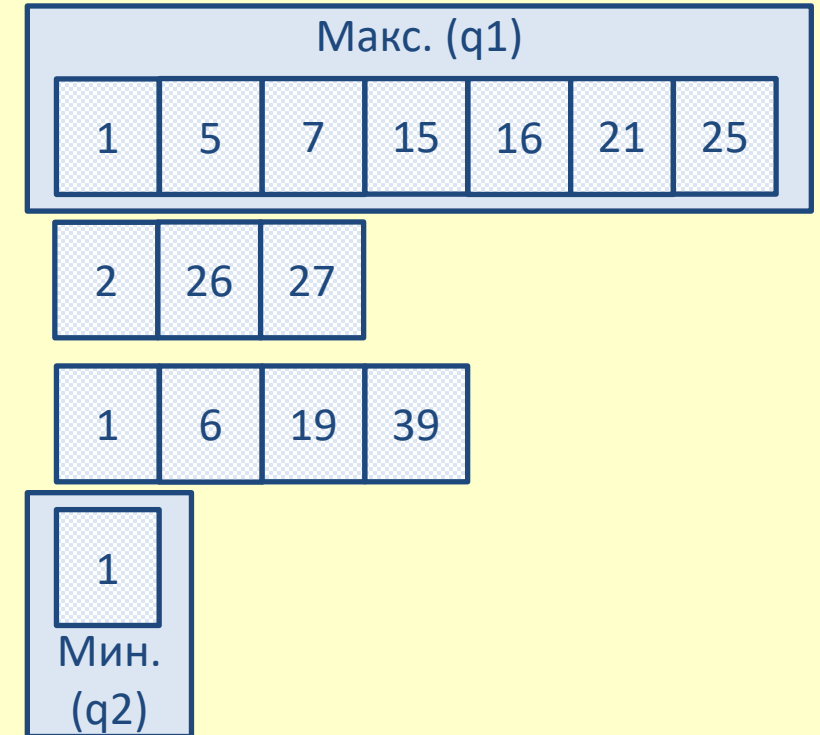


Очередь с приоритетом с ослабленной семантикой выполнения операций

АЛГОРИТМ БАЛАНСИРОВКИ

Поиск самой большой и самой маленькой, по численности элементов, очередей

```
1  q1 = FindLargestQueue();
2  q2 = FindShortestQueue();
3  if q1.size() > AvgSizeOfAllQueues()*c then
4      Lock(q1);
5      q2IsLocked = LockWithTimeout(q2);
6      if q2IsLocked then
7          sizeToTransfer = q1.size()*s
8          TransferElements(q1, q2, sizeToTransfer)
9          Unlock(q2);
10     end
11     Unlock(q1);
12 end
```

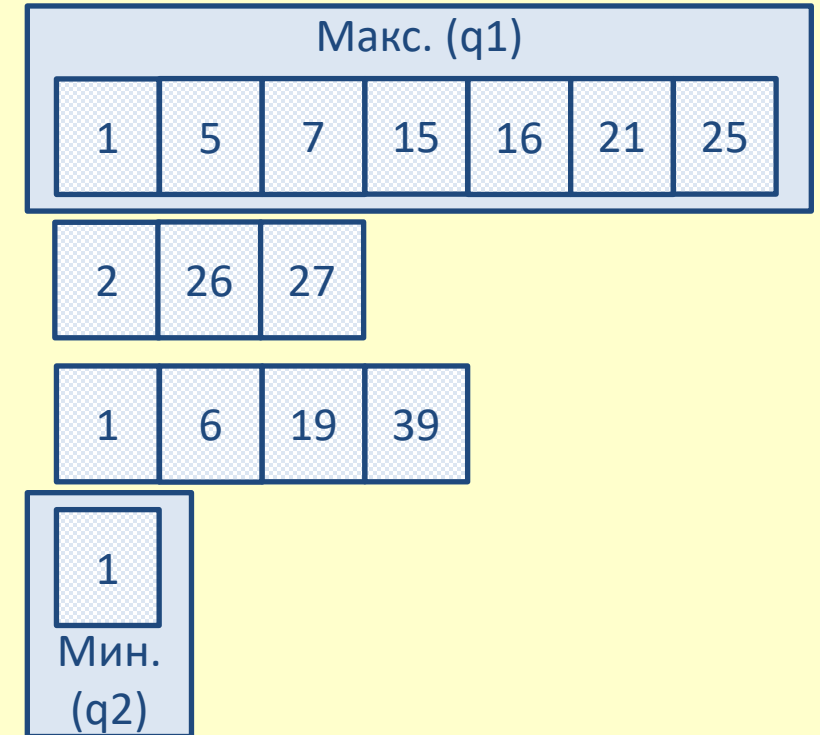


Очередь с приоритетом с ослабленной семантикой выполнения операций

АЛГОРИТМ БАЛАНСИРОВКИ

Если размер большой очереди больше чем $c\%$ среднего размера всех очередей, то выполняется балансировка

```
1  q1 = FindLargestQueue();
2  q2 = FindShortestQueue();
3  if q1.size() > AvgSizeOfAllQueues()*c then
4      Lock(q1);
5      q2IsLocked = LockWithTimeout(q2);
6      if q2IsLocked then
7          sizeToTransfer = q1.size()*s
8          TransferElements(q1, q2, sizeToTransfer)
9          Unlock(q2);
10     end
11     Unlock(q1);
12 end
```

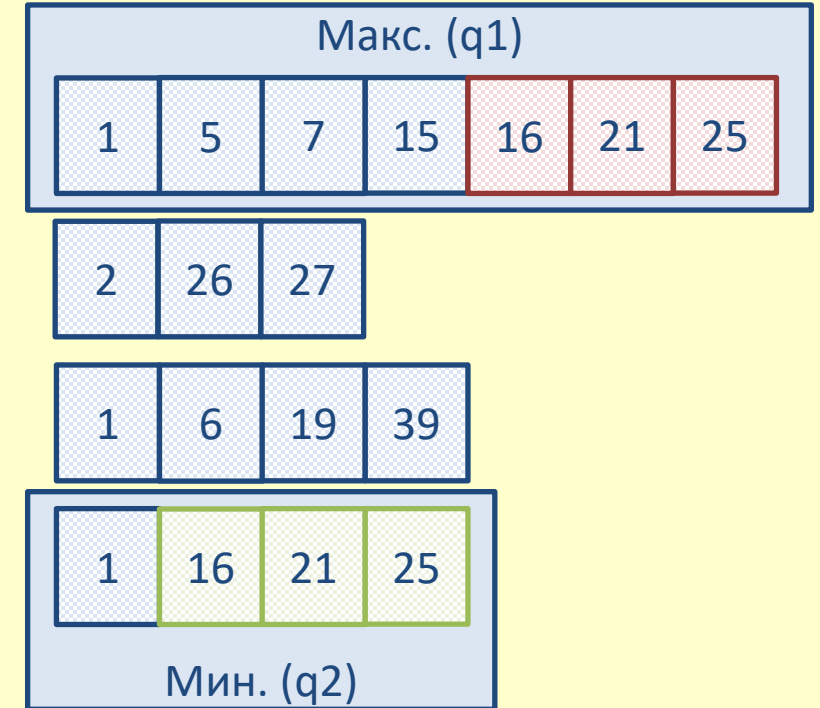


Очередь с приоритетом с ослабленной семантикой выполнения операций

АЛГОРИТМ БАЛАНСИРОВКИ

Перемещается $s\%$ элементов из самой большой в самую маленькую очередь

```
1  q1 = FindLargestQueue();
2  q2 = FindShortestQueue();
3  if q1.size() > AvgSizeOfAllQueues()*c then
4      Lock(q1);
5      q2IsLocked = LockWithTimeout(q2);
6      if q2IsLocked then
7          sizeToTransfer = q1.size()*s
8          TransferElements(q1, q2, sizeToTransfer)
9          Unlock(q2);
10     end
11     Unlock(q1);
12 end
```



Очередь с приоритетом с ослабленной семантикой выполнения операций

СОДЕРЖАНИЕ

- Обеспечение синхронизации в параллельных программах
- Ослабленные структуры данных
- Оптимизация алгоритмов вставки и удаления структуры Multiqueues
- Алгоритм балансировки структуры Multiqueues
- Экспериментальное исследование оптимизированных алгоритмов
- Построение ослабленных структур данных на основе циклических списков
- Экспериментальное исследование ослабленной циклической очереди

ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ

Спецификация вычислительного узла:

- 2 Intel Xeon X5670
- 6 Ядре 2.93 GHz per CPU
- 12M L3 Кэш
- 16 Gb RAM

Обозначения:

- $b = N / t$ – *Пропускная способность*
- N – Количество операций
- t – Время выполнения (секунд)
- p – Количество потоков
- k – Количество очередей на один поток

Эксперимент 1. Пропускная способность в зависимости от потоков:

- Количество потоков от 1 до 32 (p)
- 10^6 Операций вставки
- $0,5 \cdot 10^6$ Операций удаления
- $2 \cdot 10^6$ Случайных операций

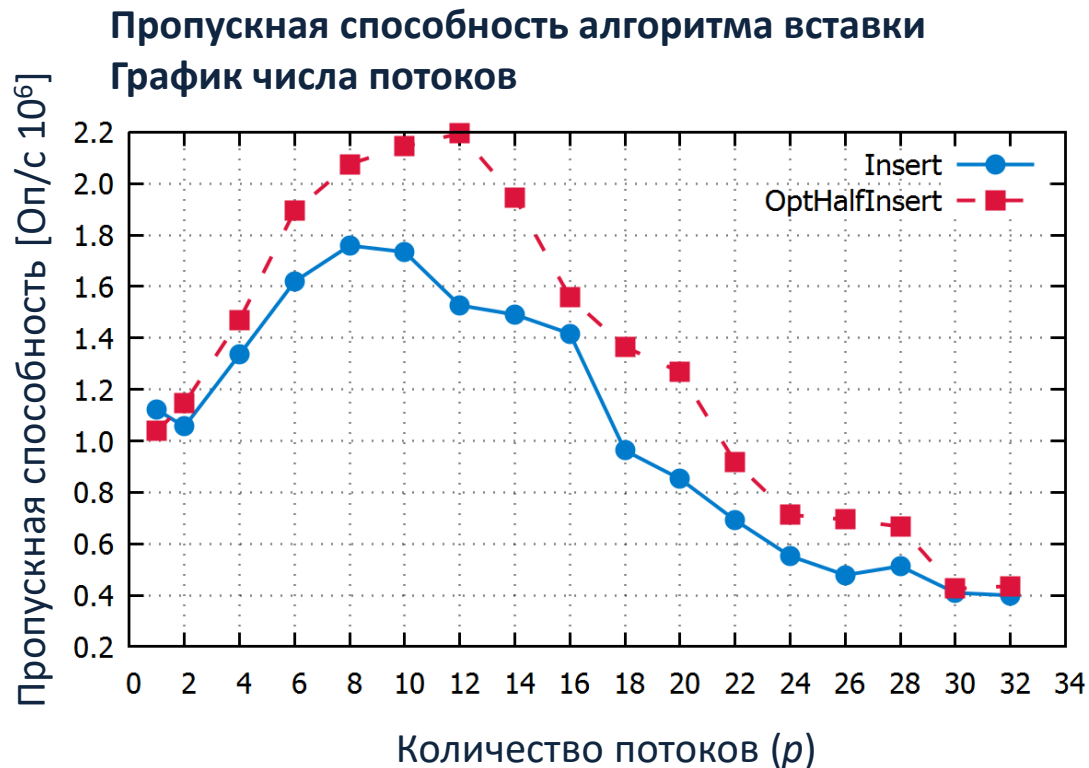
Эксперимент 2. Пропускная способность в зависимости от количества очередей на один поток:

- Количество очередей от 2 до 10 (k)
- 10^6 Операций вставки
- $0,5 \cdot 10^6$ Операций удаления

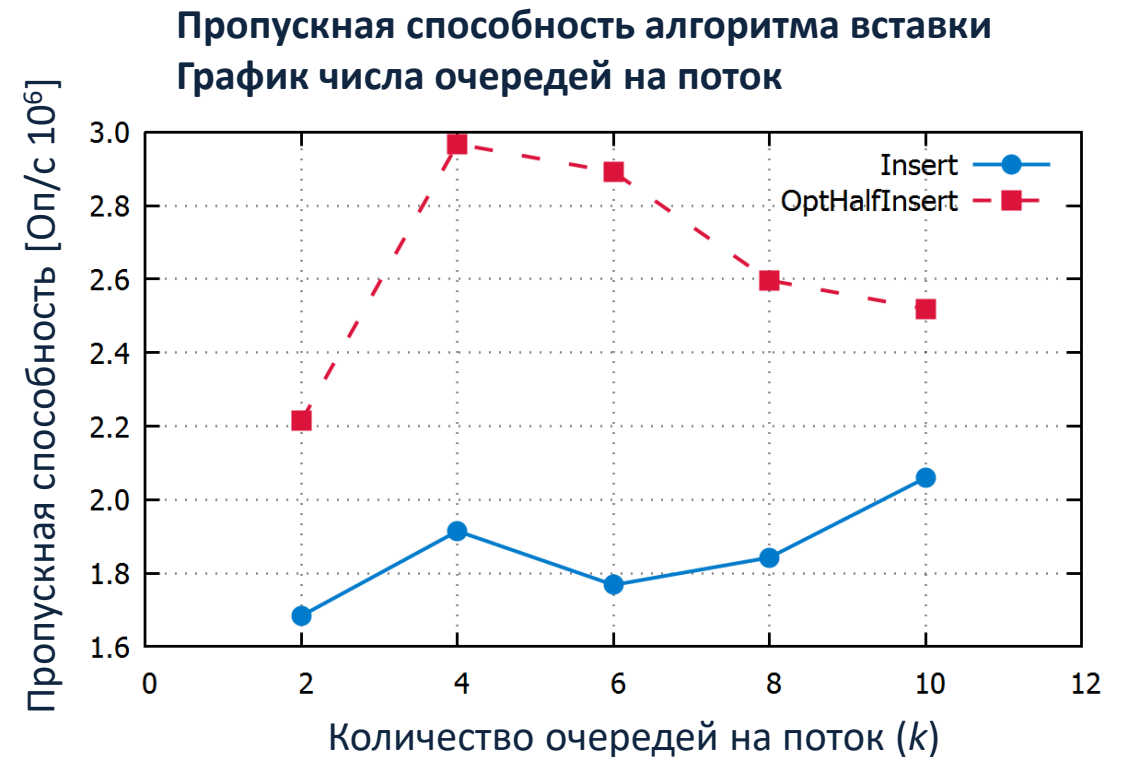
ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ

ОПЕРАЦИЯ ВСТАВКИ

Количество очередей на поток $k = 2$
Количество операций $N = 10^6$



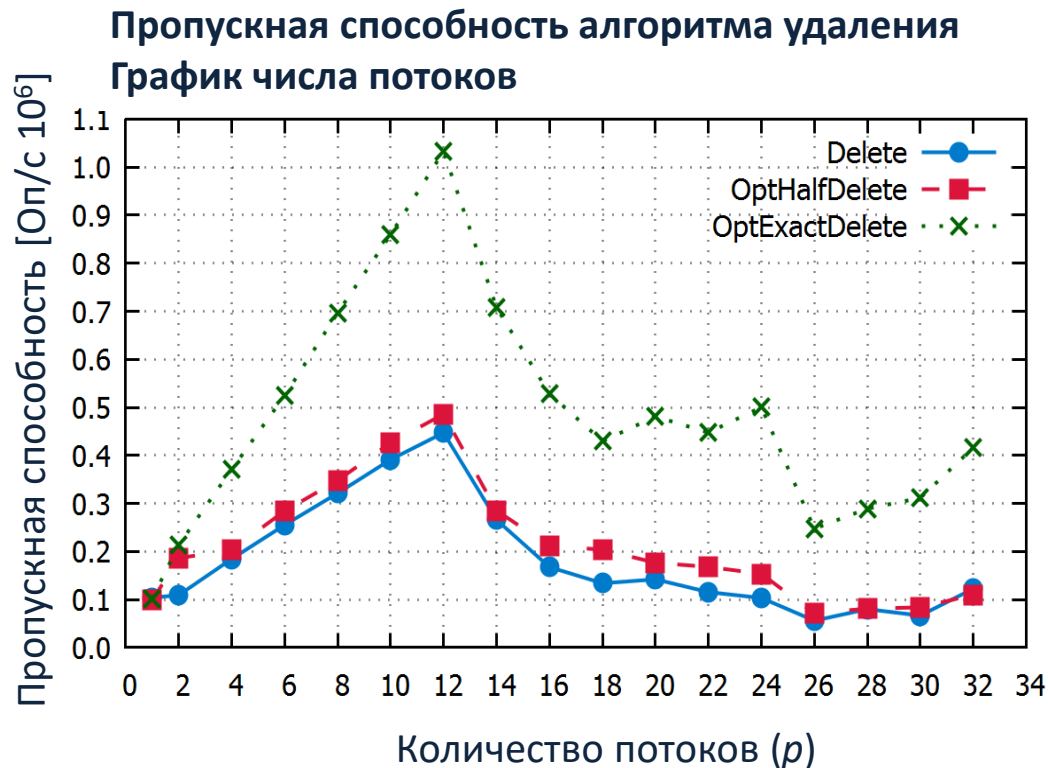
Количество потоков $p = 12$
Количество операций $N = 10^6$



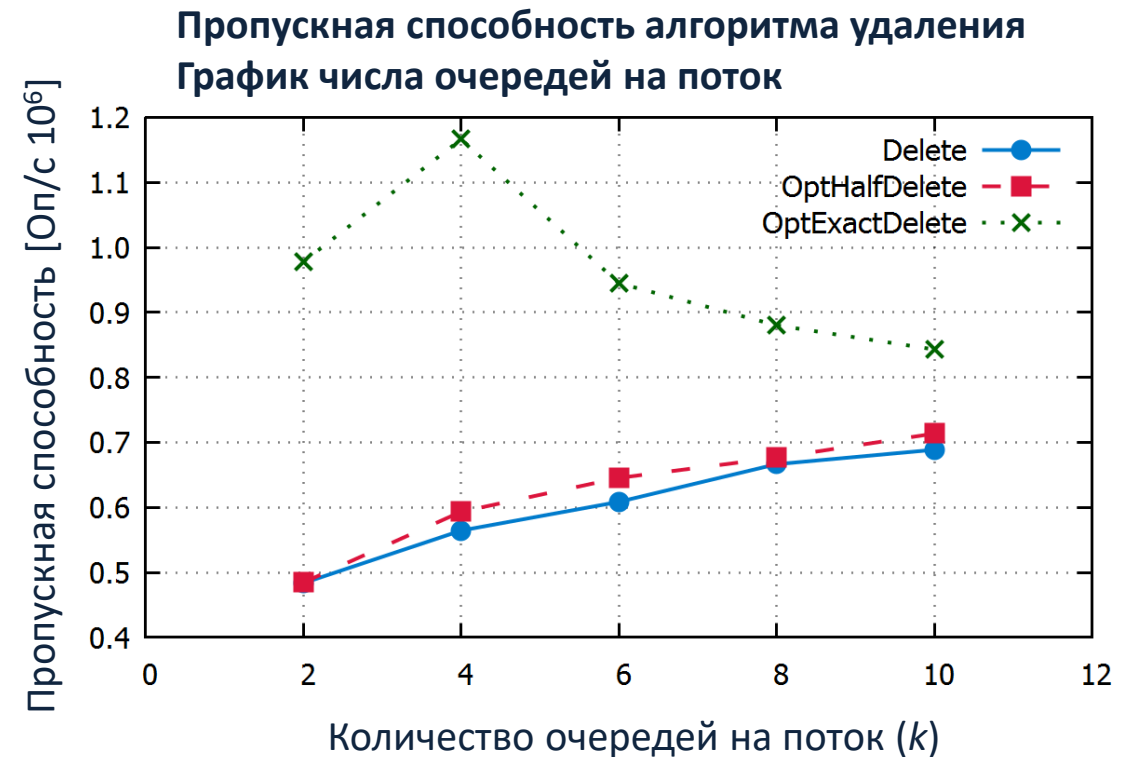
ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ

УДАЛЕНИЕ МАКСИМАЛЬНОГО ЭЛЕМЕНТА

Количество очередей на поток $k = 2$
Количество операций $N = 0,5 \cdot 10^6$



Количество потоков $p = 12$
Количество операций $N = 0,5 \cdot 10^6$

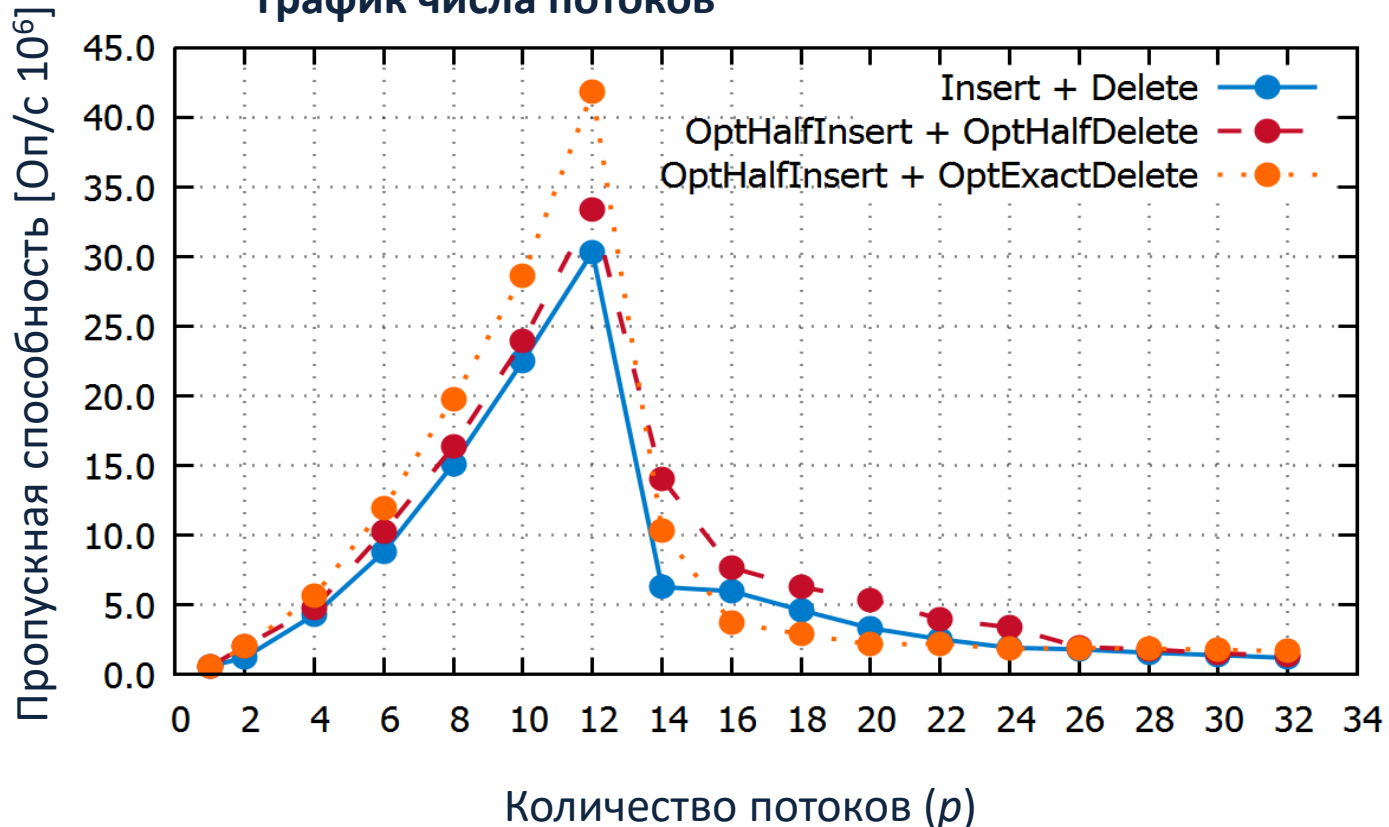


ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ СЛУЧАЙНЫЕ ОПЕРАЦИИ ВСТАВКИ И УДАЛЕНИЯ

Количество очередей на поток $k = 2$

Количество операций $N = 2 \cdot 10^6$

Пропускная способность случайных операций
График числа потоков



СОДЕРЖАНИЕ

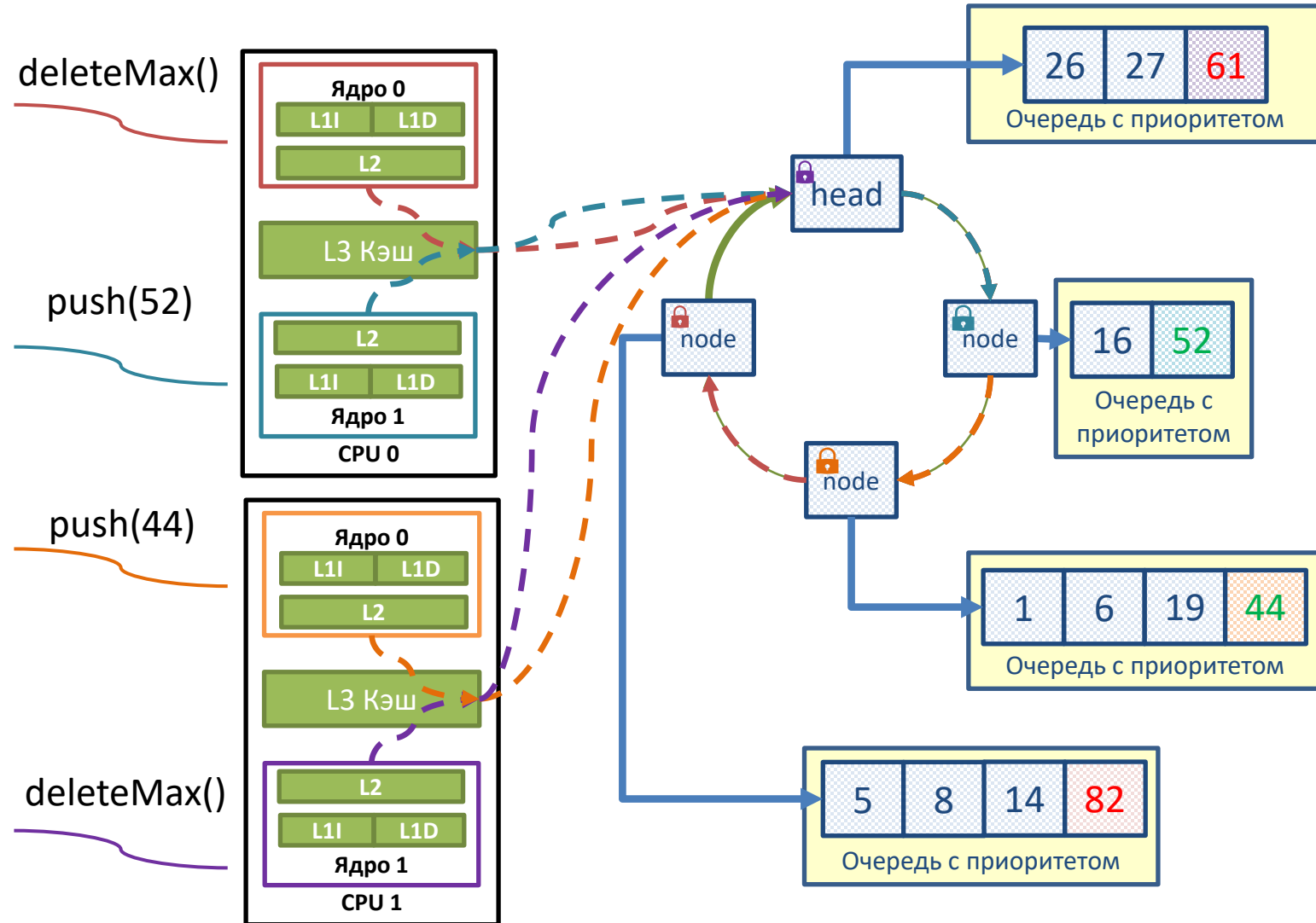
- Обеспечение синхронизации в параллельных программах
- Ослабленные структуры данных
- Оптимизация алгоритмов вставки и удаления структуры Multiqueues
- Алгоритм балансировки структуры Multiqueues
- Экспериментальное исследование оптимизированных алгоритмов
- Построение ослабленных структур данных на основе циклических списков
- Экспериментальное исследование ослабленной циклической очереди

ЦИКЛИЧЕСКАЯ ОСЛАБЛЕННАЯ ОЧЕРЕДЬ С ПРИОРИТЕТОМ

Circular Relaxed Concurrent Priority Queue (CPQ)

Каждый узел связного циклического списка содержит указатель на собственную последовательную структуру

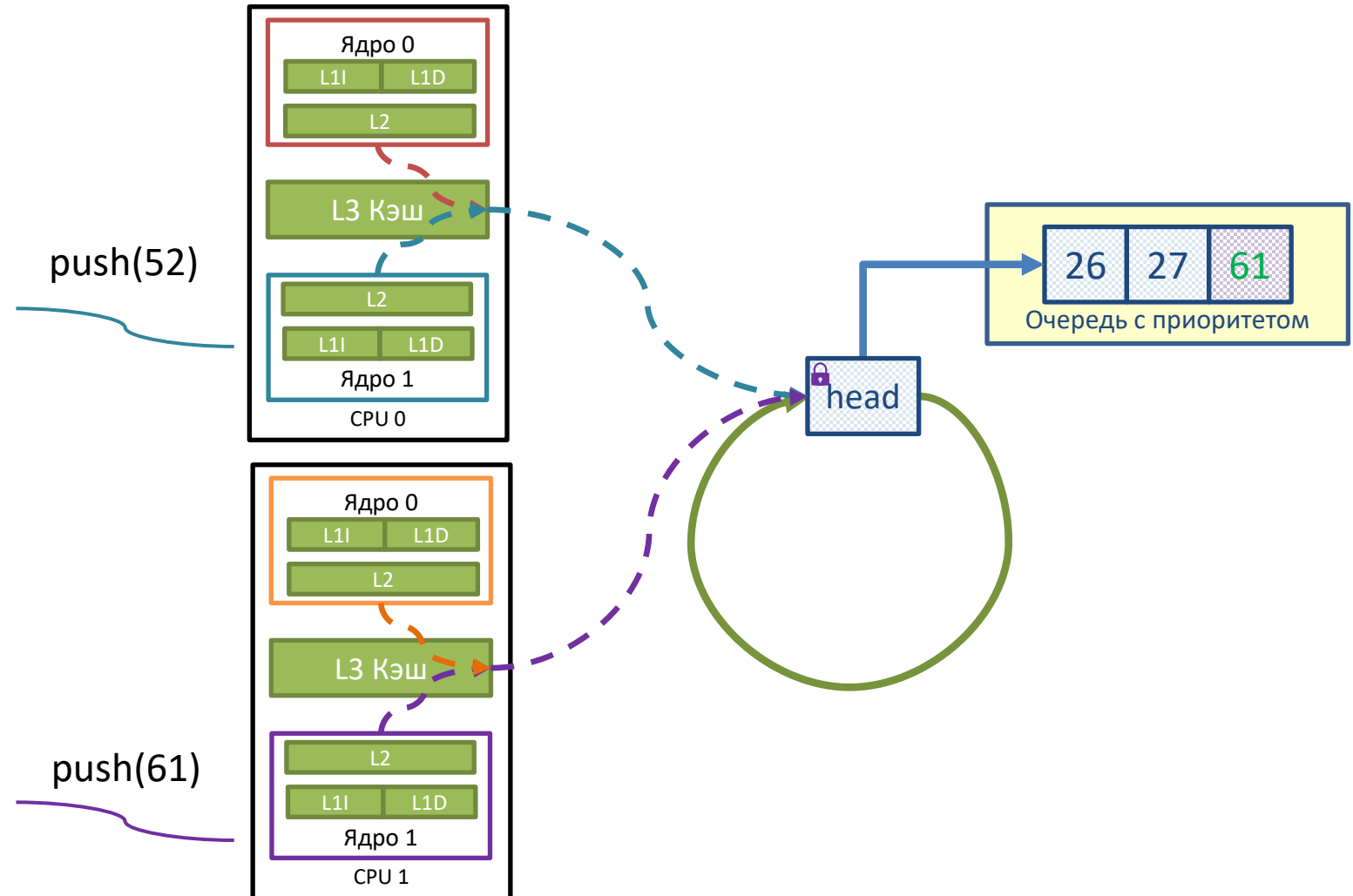
Блокировка запрещает запись в последовательную структуру, но разрешает чтение



CPQ - АЛГОРИТМ ВСТАВКИ

Изначально циклический список имеет только один узел указывающий на структуру данных

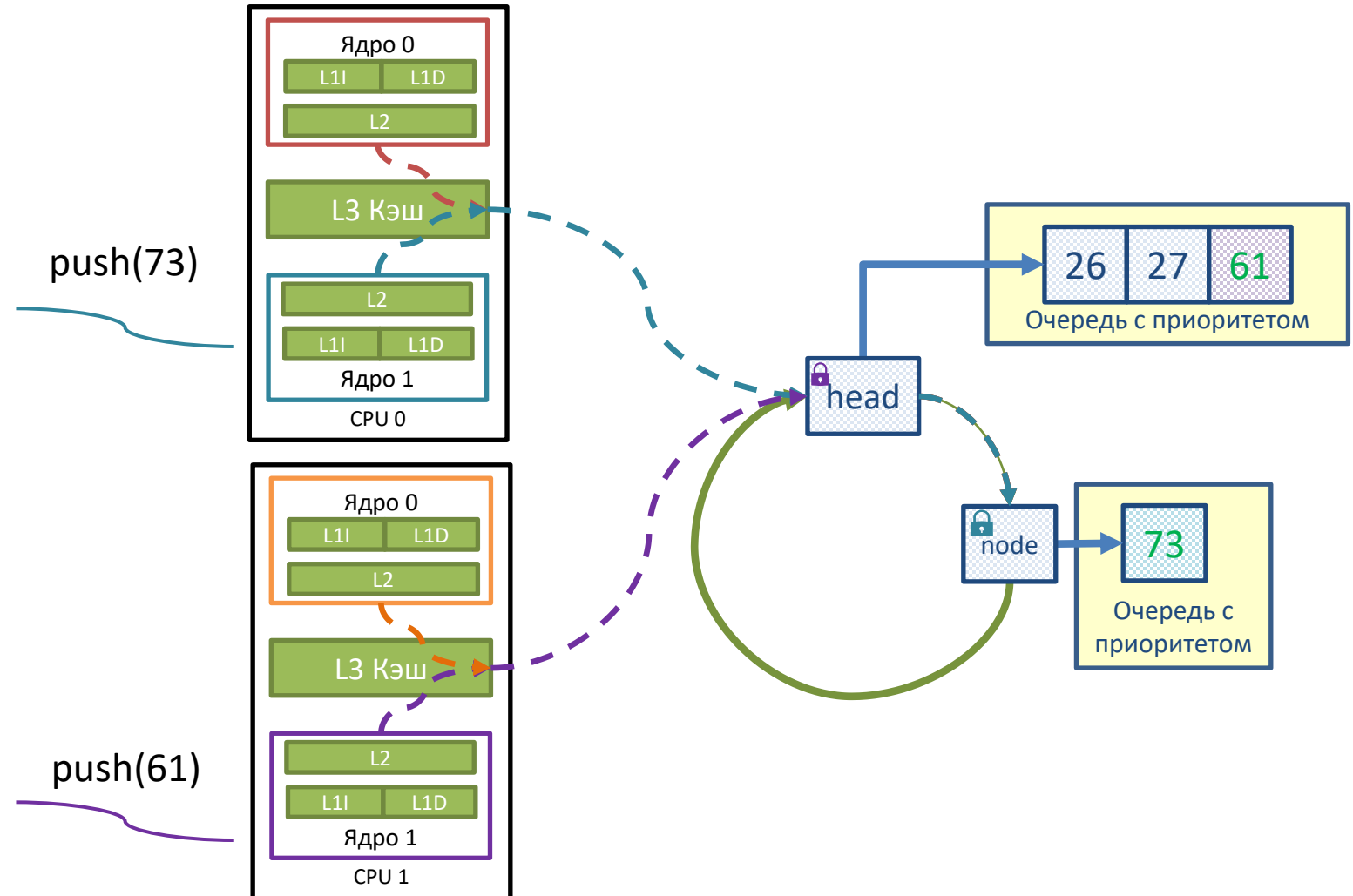
```
1  head = CPQHead();
2  node = head;
3  do
4    if TryLock(node) == true then
5      PushElement(node, el);
6      Unlock(node);
7      return;
8  end
9  node = NextNode(node);
10 while node != head;
11 node = CreateNewNode(head);
12 Lock(node);
13 PushElement(node, el);
14 Unlock(node);
```



CPQ - АЛГОРИТМ ВСТАВКИ

Если двум потокам требуется одновременный доступ на запись в данную структуру, то один из потоков заблокирует доступный узел, в то же время другой, создаст новый

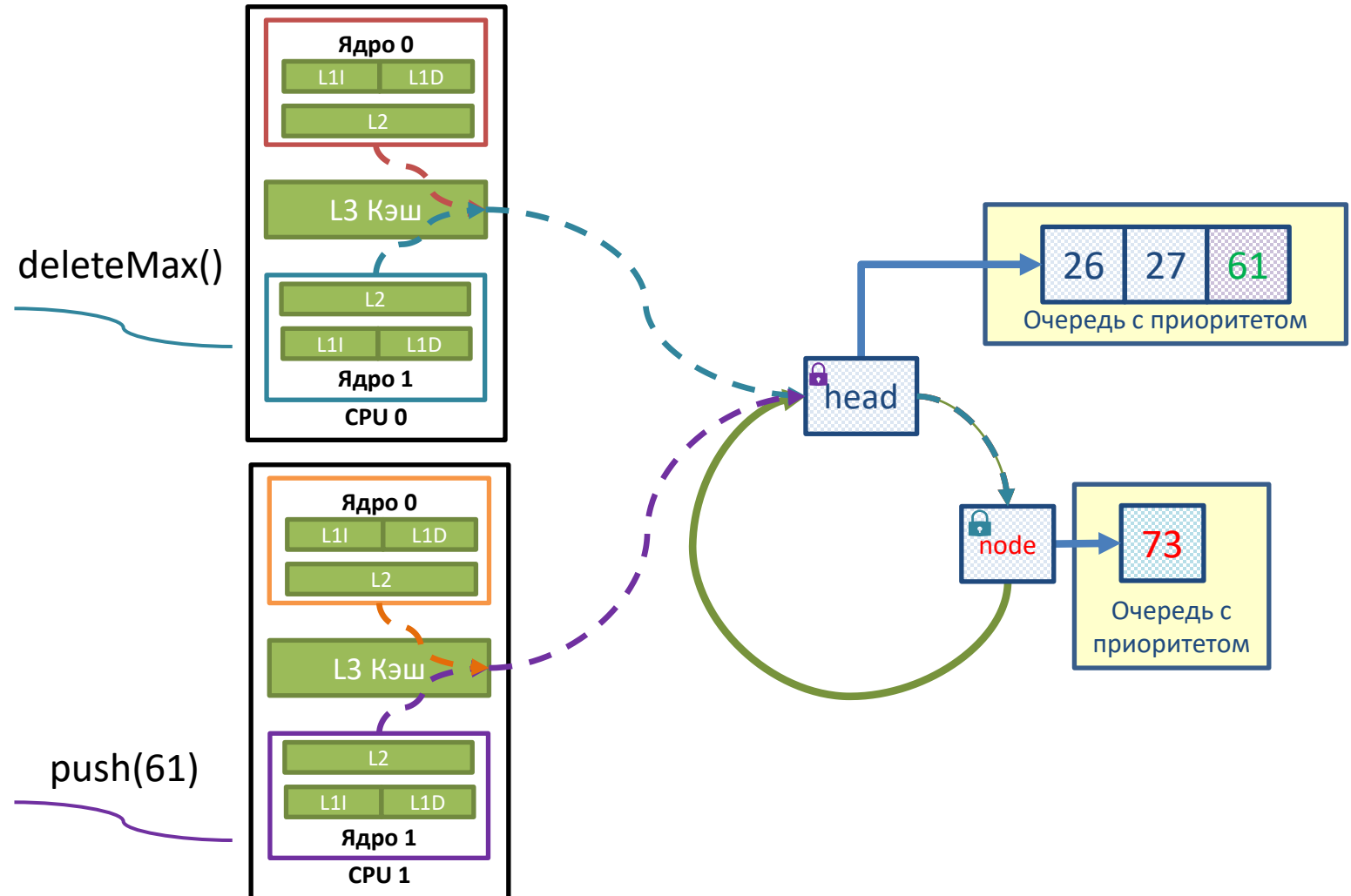
```
1  head = CPQHead();
2  node = head;
3  do
4    if TryLock(node) == true then
5      PushElement(node, el);
6      Unlock(node);
7      return;
8  end
9  node = NextNode(node);
10 while node != head;
11 node = CreateNewNode(head);
12 Lock(node);
13 PushElement(node, el);
14 Unlock(node);
```



CPQ - АЛГОРИТМ УДАЛЕНИЯ МАКСИМАЛЬНОГО ЭЛЕМЕНТА

Во время операции удаления, происходит чтение максимальных элементов всех очередей. Удаление происходит из очереди с максимальным элементом

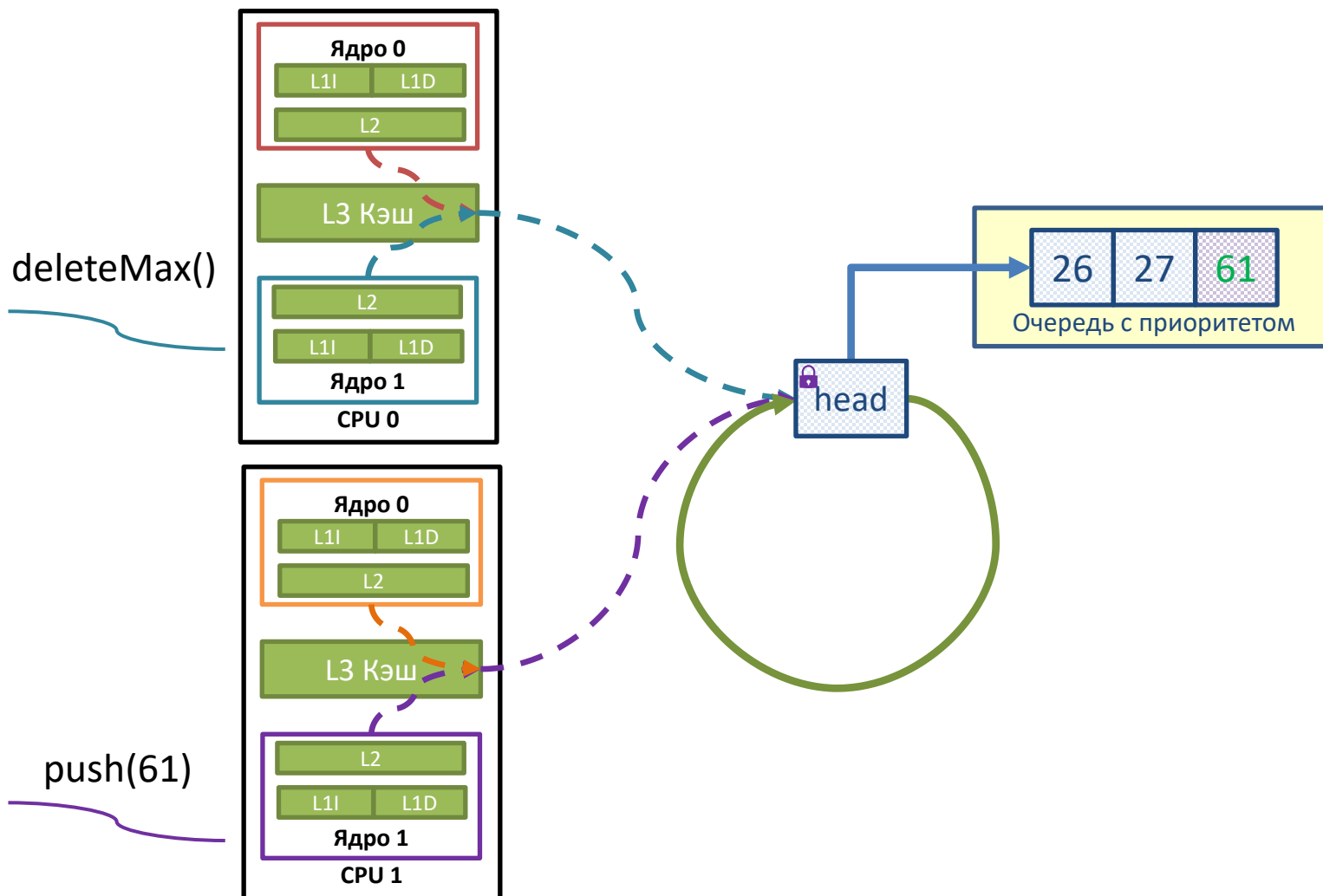
```
1  head = CPQHead();
2  node = NextNode(node);
3  priorValue = GetTopValue(node);
4  priorNode = node;
5  do
6    value = GetTopValue(node);
7    if value > priorValue then
8      priorValue = value
9      priorNode = node;
10 end
11 node = NextNode(node);
12 while node != head;
13 Lock(priorNode);
14 DeleteMax(priorNode);
15 Unlock(priorNode);
```



CPQ - АЛГОРИТМ УДАЛЕНИЯ МАКСИМАЛЬНОГО ЭЛЕМЕНТА

Если последовательная структура становится пустой, узел удаляется из списка

```
1  head = CPQHead();
2  node = NextNode(node);
3  priorValue = GetTopValue(node);
4  priorNode = node;
5  do
6    value = GetTopValue(node);
7    if value > priorValue then
8      priorValue = value
9      priorNode = node;
10  end
11  node = NextNode(node);
12  while node != head;
13  Lock(priorNode);
14  DeleteMax(priorNode);
15  Unlock(priorNode);
```



СОДЕРЖАНИЕ

- Обеспечение синхронизации в параллельных программах
- Ослабленные структуры данных
- Оптимизация алгоритмов вставки и удаления структуры Multiqueues
- Алгоритм балансировки структуры Multiqueues
- Экспериментальное исследование оптимизированных алгоритмов
- Построение ослабленных структур данных на основе циклических списков
- Экспериментальное исследование ослабленной циклической очереди

ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ

Спецификация вычислительного узла:

- AMD 3800X
- 8 Ядер 4.35 GHz
- 2 x DDR4 3200 MHz 16 Gb RAM

Обозначения :

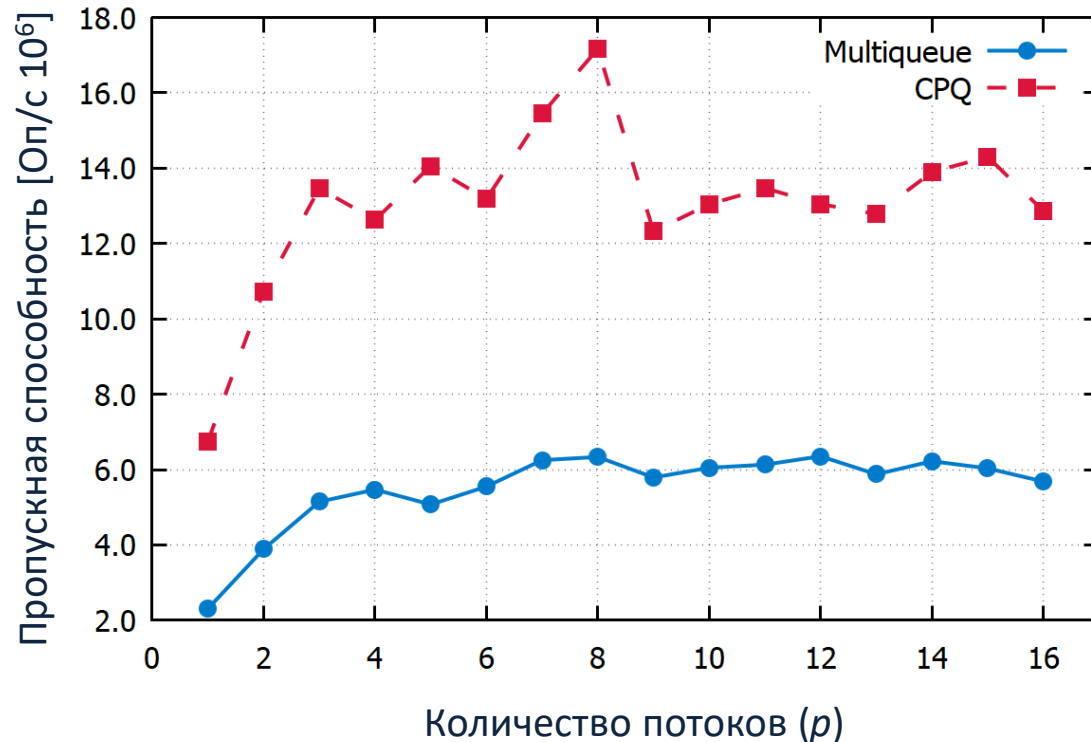
- $b = N / t$ – *Пропускная способность*
- N – Количество операций
- t – Время выполнения (секунд)
- p – Количество потоков

Для сравнительного анализа использовалась структура данных Multiqueues с оптимизированными алгоритмами OptHalfInsert и OptExactDelete.

ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ ОПЕРАЦИИ ВСТАВКИ И УДАЛЕНИЯ

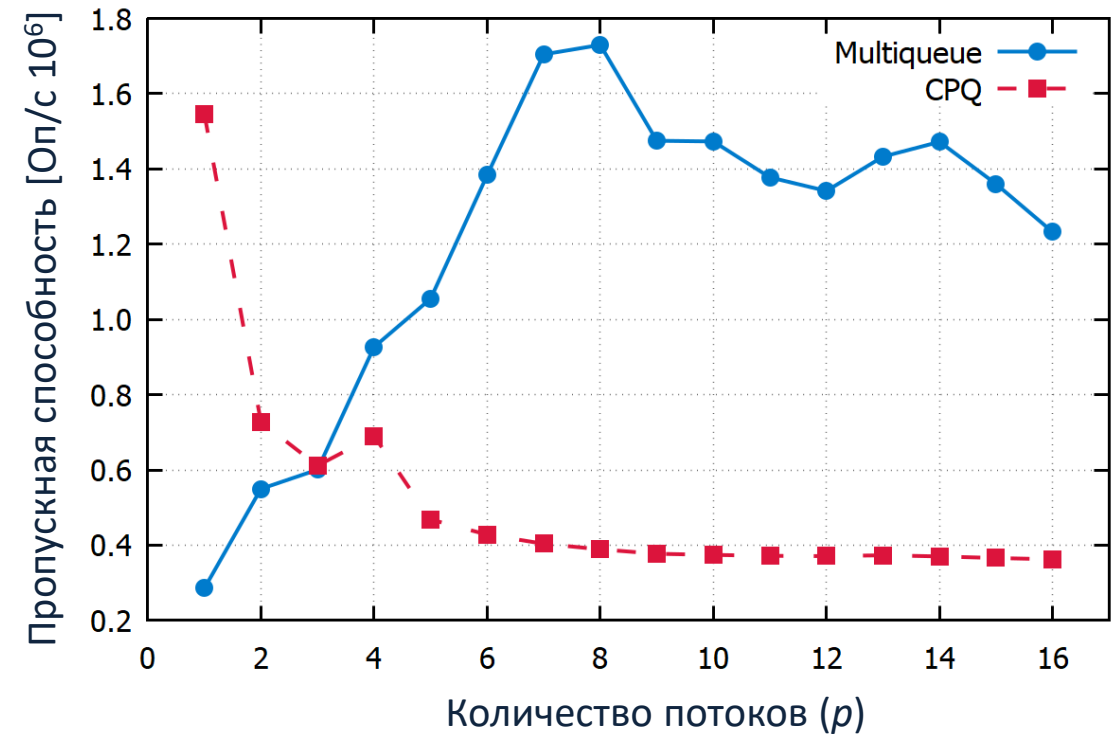
Количество операций $N = 10^7$

Пропускная способность алгоритма вставки
График числа потоков



Количество операций $N = 5 \cdot 10^6$

Пропускная способность алгоритма удаления
График числа очередей на поток



ЗАКЛЮЧЕНИЕ

- **Mutltiqueues:**
 - Реализованы оптимизированные алгоритмы для ослабленной структуры данных Multiqueues, увеличивающие пропускную способность операций вставки и удаления максимального (минимального) элемента в 1.2 и 1.6 раз, соответственно
 - В задачах, в которых преобладает операция удаления максимального элемента, предлагается использовать данную структуру, поскольку она показывает лучшую масштабируемость данной операции
- **Ослабленные структуры данных на основе циклических списков:**
 - Предложен подход создания ослабленных структур данных с использованием связанного циклического списка на основе данного подхода разработана ослабленная структура данных CPQ
 - В задачах с преобладающим числом операции вставка, стоит рассматривать структуру CPQ, кроме того, данная структура демонстрирует гораздо большую точность при выполнении операции удаления максимального элемента
- Представленные структуры могут быть использованы в многопоточных системах, при решении задач, где точность результата операции может находиться в заданной области

Реализация Multiqueues		Реализация Circular Relaxed Concurrent Priority Queue	
C++	Kotlin	C++	Kotlin
Github.com /Komdosh/Multiqueues	Github.com /Komdosh/KotMultiqueues	Github.com /Komdosh/CircularPriorityQueue	Github.com /Komdosh/RelaxedCycleDS

ОСНОВНЫЕ ПУБЛИКАЦИИ ПО ТЕМЕ ДИССЕРТАЦИИ

Индексируемые Scopus и Web of Science:

1. **A. V. Tabakov**, A. A. Paznikov, Optimization of Data Locality in Relaxed Concurrent Priority Queues // Proc. of CEUR Workshop, 2020. – pp. 419-431.
2. **A. V. Tabakov**, A. A. Paznikov, Using relaxed concurrent data structures for contention minimization in multithreaded MPI programs // 2019 J. Phys.: Conf. Ser. 1399 033037. doi: 10.1088/1742-6596/1399/3/033037
3. Goncharenko E. A., Paznikov A. A., **Tabakov A. V.** Evaluating the performance of atomic operations on modern multicore systems //Journal of Physics: Conference Series. – IOP Publishing, 2019. – T. 1399. – №. 3. – C. 033107.
4. **A. V. Tabakov**, A. A. Paznikov, Modelling of Parallel Threads Synchronization in Hybrid MPI + Threads Programs // XXII International Conference on Soft Computing and Measurements (SCM)), St. Petersburg, Russia, 2019, pp. 197-199. doi: 10.1109/SCM.2019.8903806
5. **A. V. Tabakov**, A. A. Paznikov, Algorithms for Optimization of Relaxed Concurrent Priority Queues in Multicore Systems // Proc. of the 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus), 2019. – pp. 360-365. 10.1109/ElConRus.2019.8657105

Индексируемые ВАК:

1. **Табакوف А. В.**, Пазников А. А. Алгоритмы оптимизации потокобезопасных очередей с приоритетом на основе ослабленной семантики выполнения операций // Известия СПбГЭТУ «ЛЭТИ». – 2018. – № 10. – С. 42-49
2. Пазников А. А., **Табакوف А. В.**, Куприянов М. С. Алгоритм выбора локальных окрестностей децентрализованных диспетчеров в пространственно-распределенных вычислительных системах //Известия СПбГЭТУ ЛЭТИ. – 2020. – №. 10. – С. 54-62.

Публикации в журналах и конференциях:

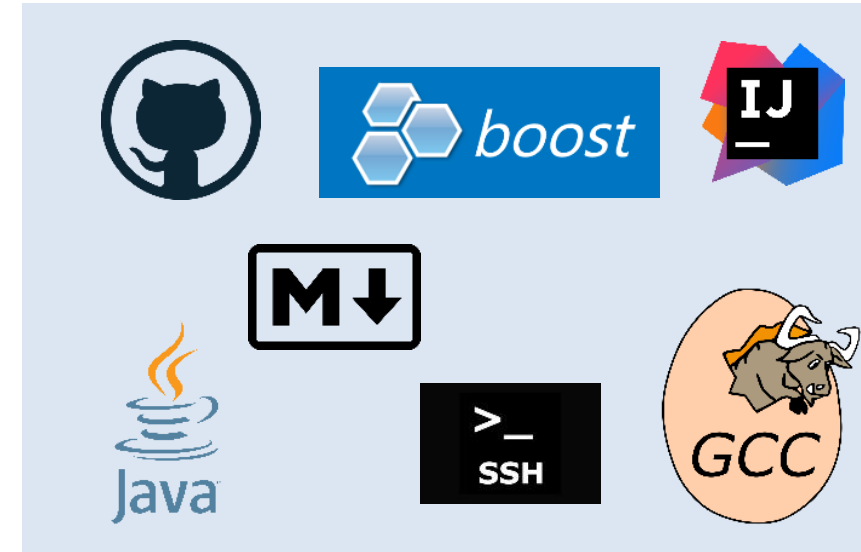
1. **Табакوف А. В.**, Пазников А. А. Моделирование синхронизации параллельных потоков при выполнении гибридных MPI+threads программ // Материалы XXI IEEE международной конференции по мягким вычислениям (SCM), 2019. – С. 293-295
2. **Табакوف А. В.**, Веретенников Л. М., Потокобезопасные структуры данных с ослабленной семантикой выполнения операций // Материалы VI международной конференции “Наука настоящего и будущего”, СПбГЭТУ “ЛЭТИ”, Россия, 2018, С. 105-107

СВИДЕТЕЛЬСТВО О РЕГИСТРАЦИИ ПРОГРАММЫ



БЭКАП СЛАЙДЫ

ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ



АКТУАЛЬНОСТЬ

Обработка больших массивов данных
Существенное ускорение вычислений за счёт
незначительной потери точности

Совершенствование blockchain алгоритмов

Оптимизация распределённой структуры
данных выдачи части блока для вычислений

Популярность серверных решений

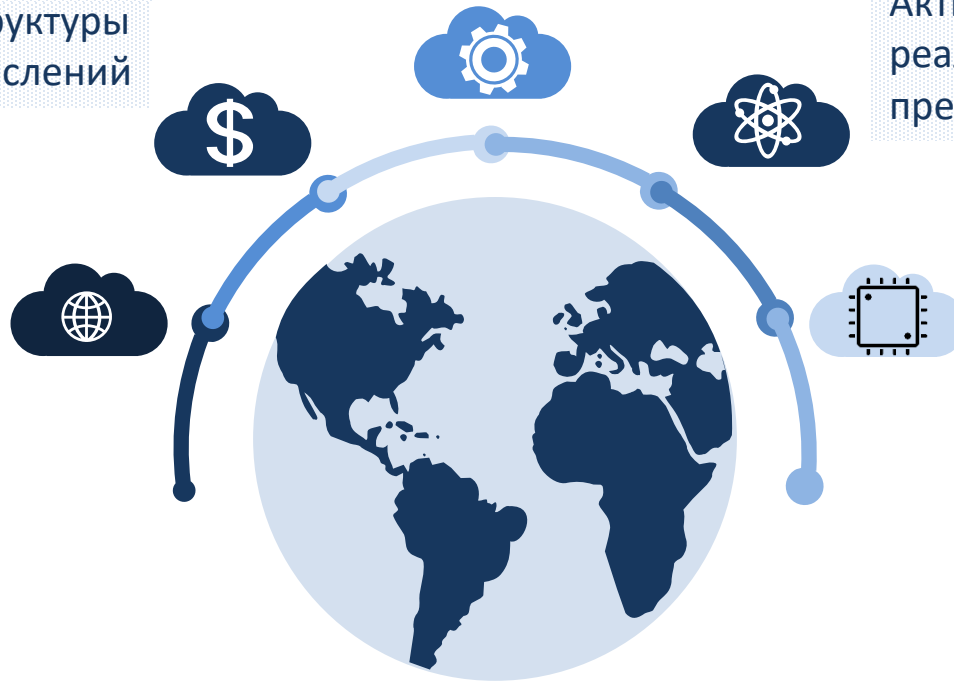
Уменьшение времени доступа к
распределённым данным,
хранимым в памяти

Распространение нейронных сетей

Активация синапсов может быть
реализована в виде обращения к
предложенным структурам данных

Развитие многоядерных архитектур

Потокобезопасные структуры данных
хранятся в общей памяти (RAM)



СТРУКТУРЫ ДАННЫХ С ОСЛАБЛЕННОЙ СЕМАНТИКОЙ ВЫПОЛНЕНИЯ ОПЕРАЦИЙ

Данный подход основан на компромиссе между масштабируемостью (производительностью) и корректностью семантики выполнения операций.

Ослабление семантики выполнения операций позволяет увеличить возможности структуры к масштабируемости.

Преимущества ослабленных структур

Потоки меньше состязаются за ресурс

Масштабируемость

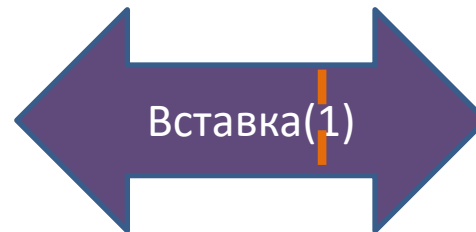
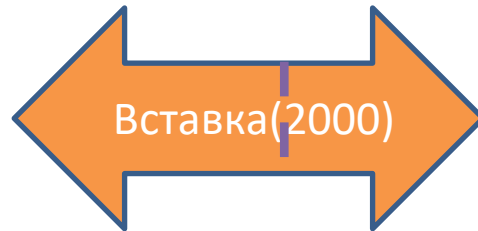
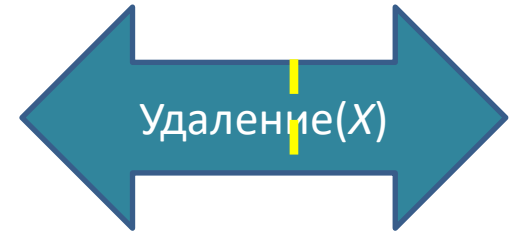
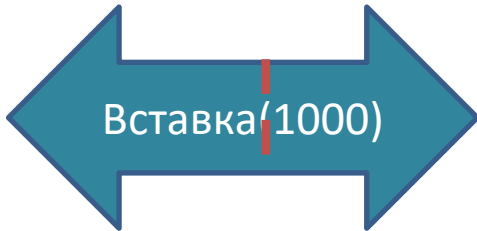
Высокая пропускная способность

Недостатки ослабленных структур

Операции не линеаризуемы

Результат операции непредсказуем

ЛИНЕАРИЗУЕМОСТЬ

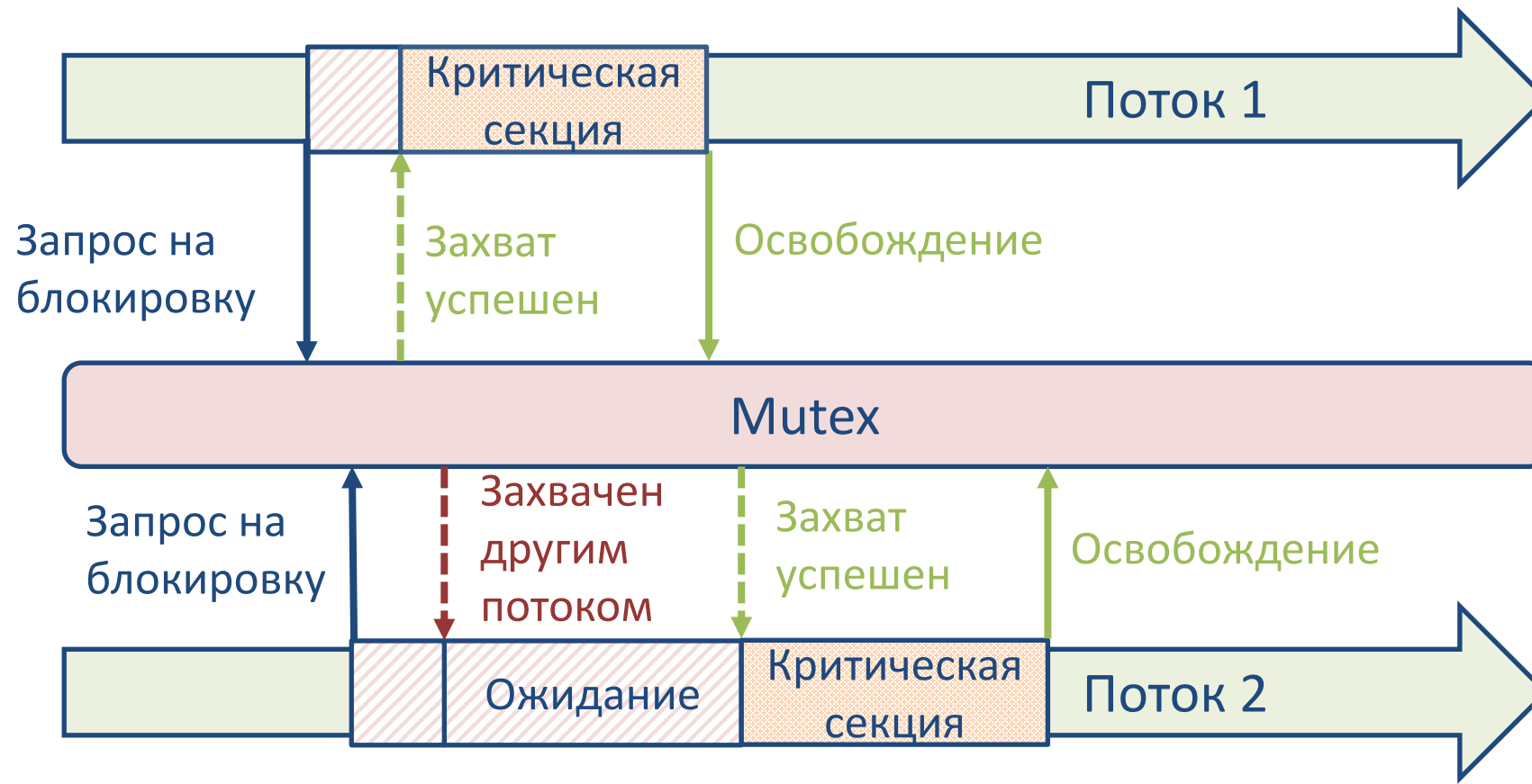


Если $X = 1000$ – Линеаризуема,
Если $X = 2000$ – Не линеаризуема



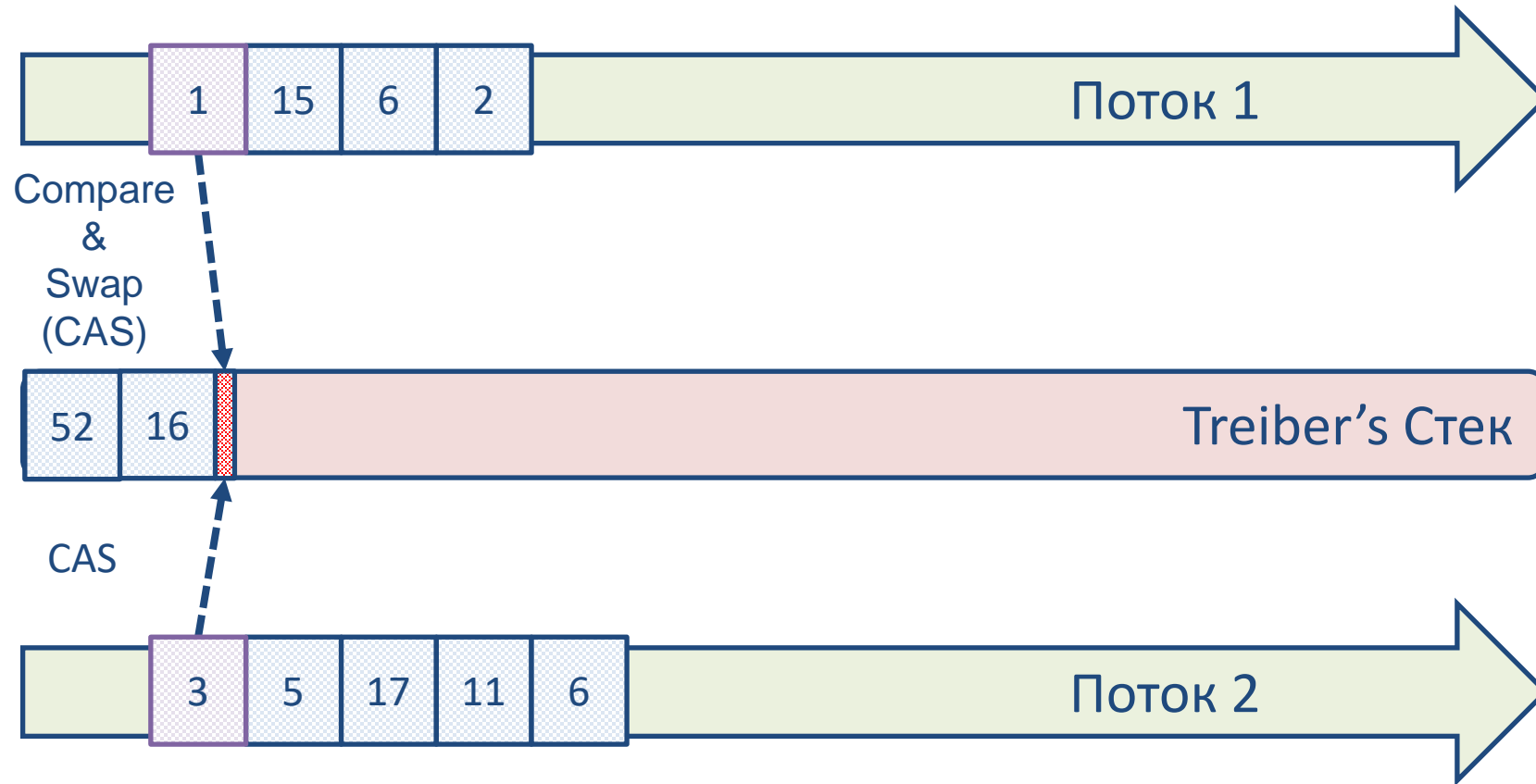
МЕТОДЫ СИНХРОНИЗАЦИИ

Блокировки (Locks) – механизм синхронизации для ограничения одновременного доступа к ресурсу



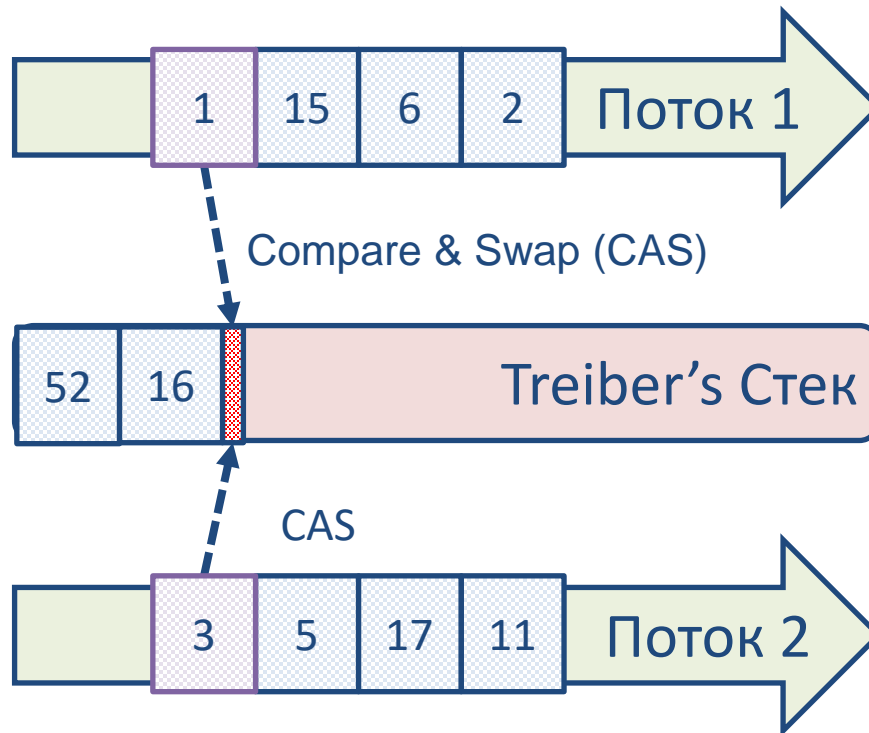
МЕТОДЫ СИНХРОНИЗАЦИИ

Неблокируемые структуры – Последовательность операций
линеаризуема, каждая операция гарантирует прогресс для всей системы



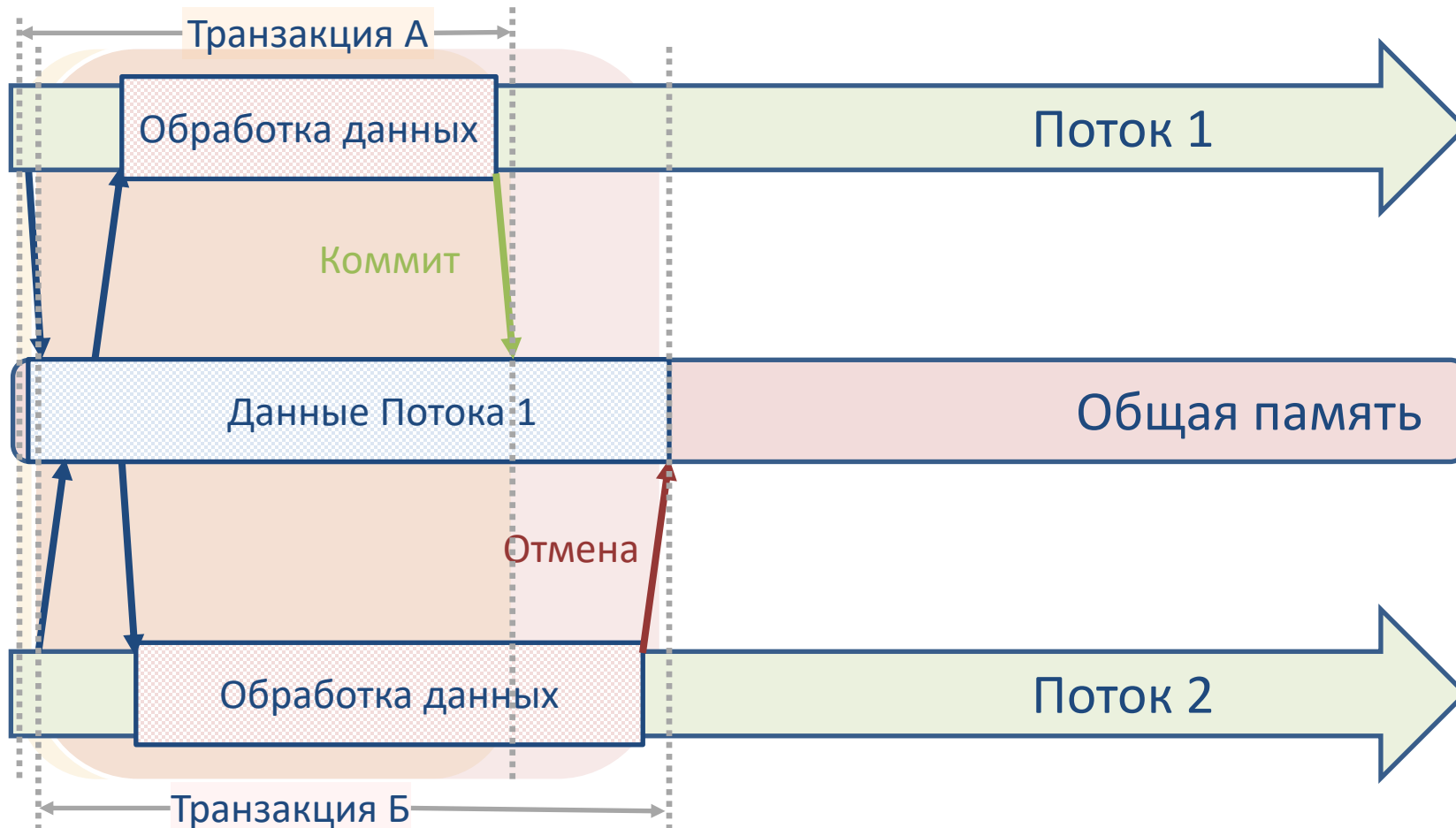
МЕТОДЫ СИНХРОНИЗАЦИИ

Неблокируемые структуры – Последовательность операций
линеаризуема, каждая операция гарантирует прогресс для всей системы



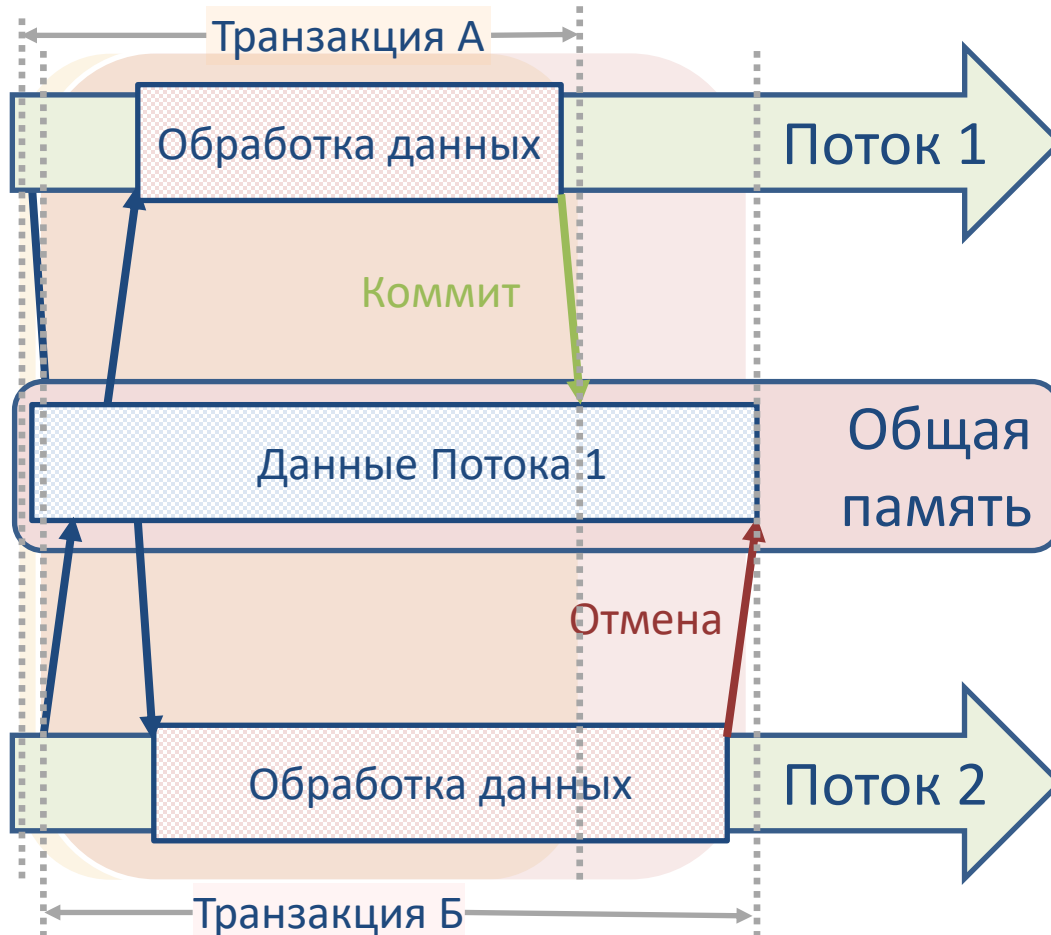
МЕТОДЫ СИНХРОНИЗАЦИИ

Транзакционная память – Объединение группы операций чтения и записи в единую неделимую операцию



МЕТОДЫ СИНХРОНИЗАЦИИ

Транзакционная память – Объединение группы операций чтения и записи в единую неделимую операцию

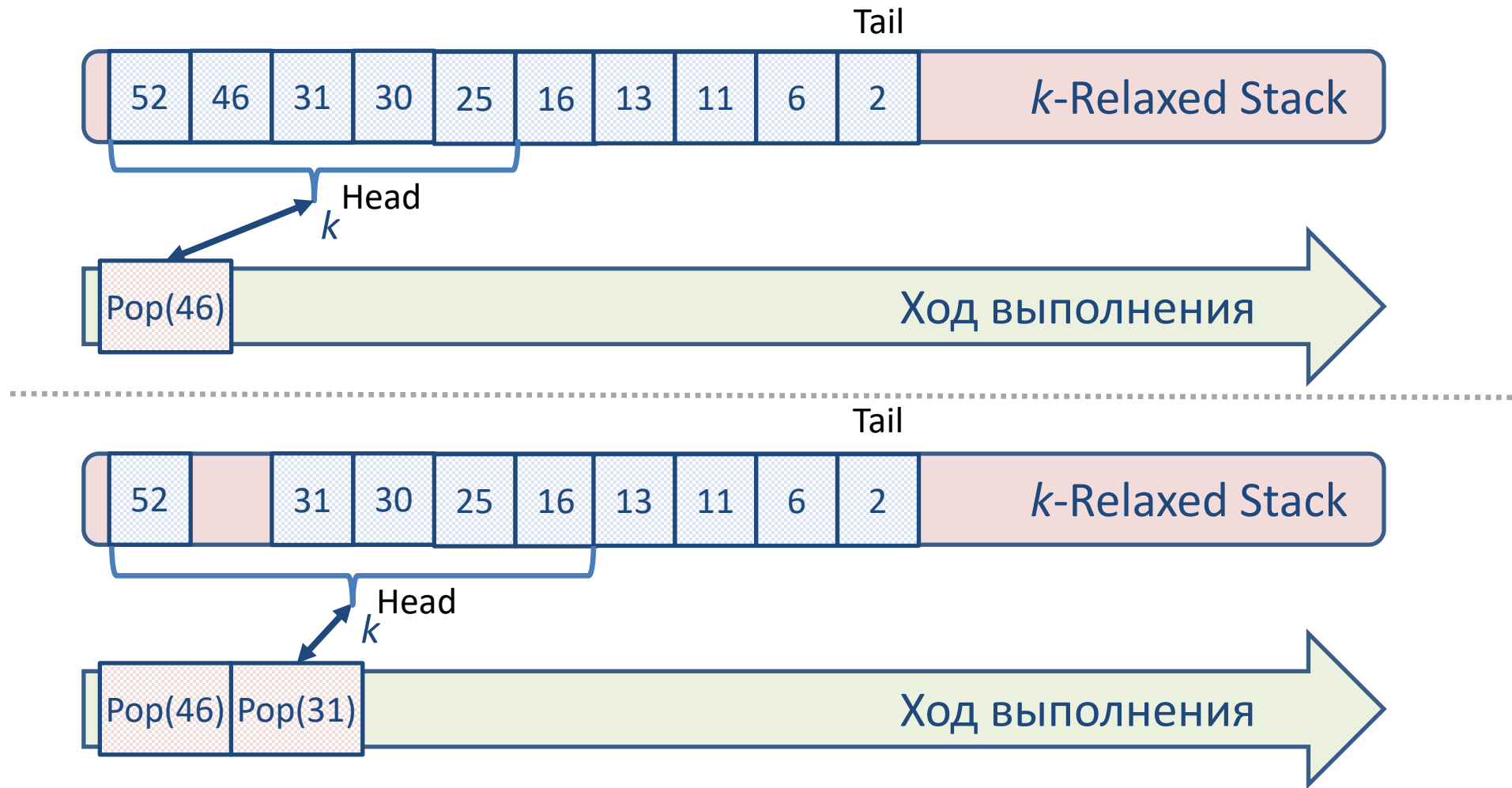


ПОСЛЕДОВАТЕЛЬНЫЙ СТЕК



ОСЛАБЛЕННЫЙ НА k ЭЛЕМЕНТОВ СТЕК

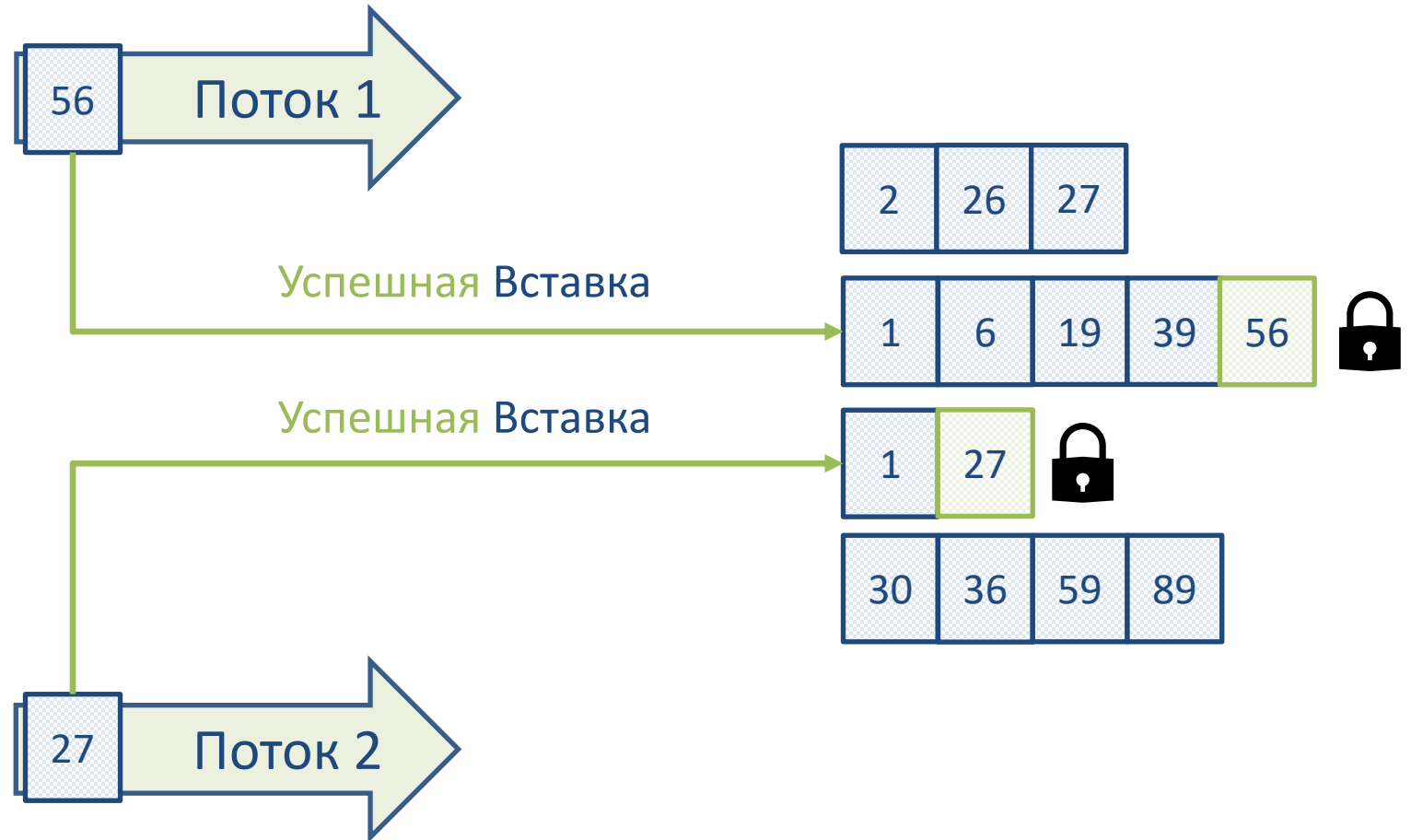
k -RELAXED STACK



ОСЛАБЛЕННАЯ ОЧЕРЕДЬ С ПРИОРИТЕТОМ

Операция вставки

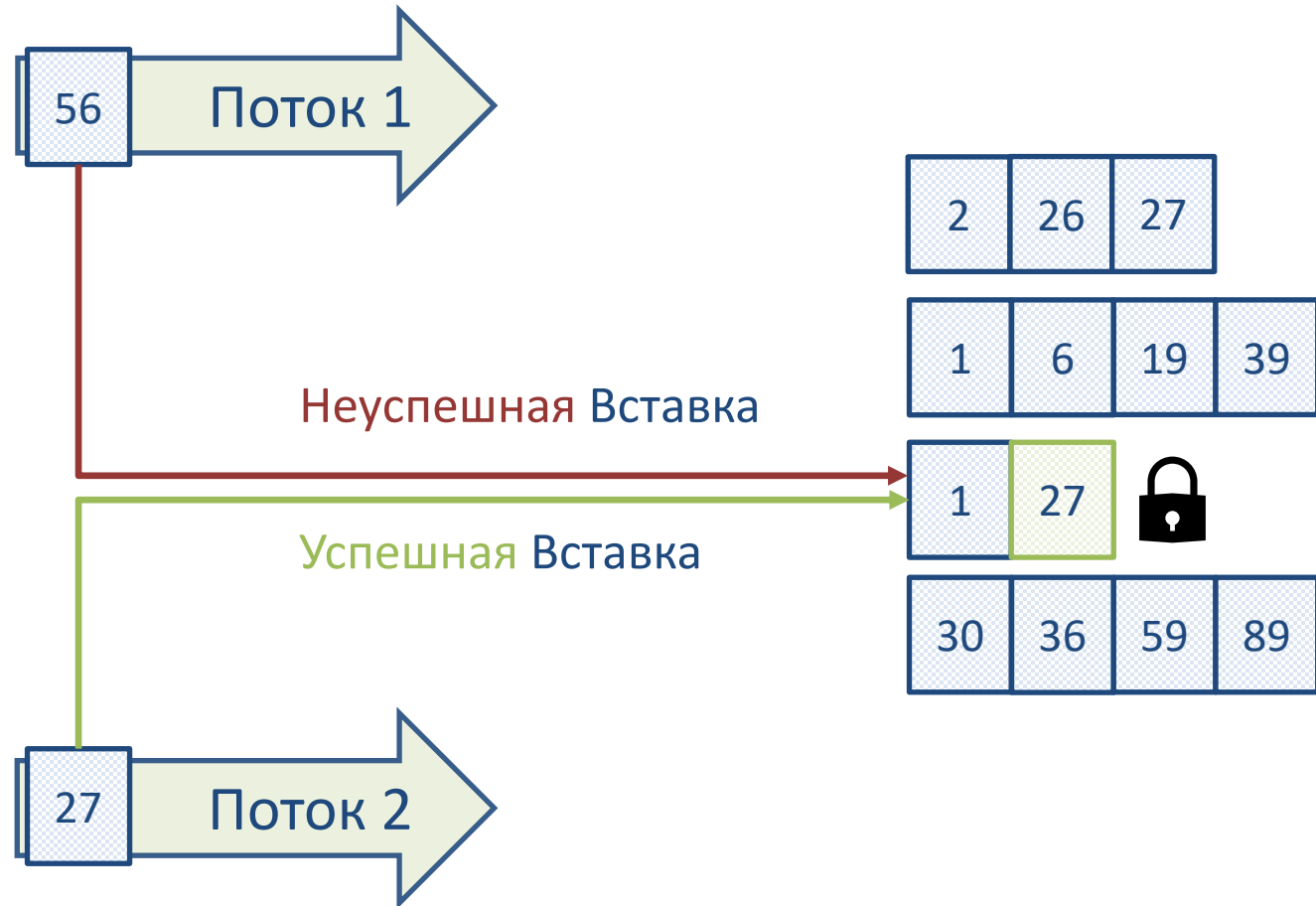
- Поток случайным образом выбирает одну очередь и множества
- Попытка заблокировать очередь
- После блокировки происходит вставка элемента в эту очередь



ОСЛАБЛЕННАЯ ОЧЕРЕДЬ С ПРИОРИТЕТОМ

Операция вставки

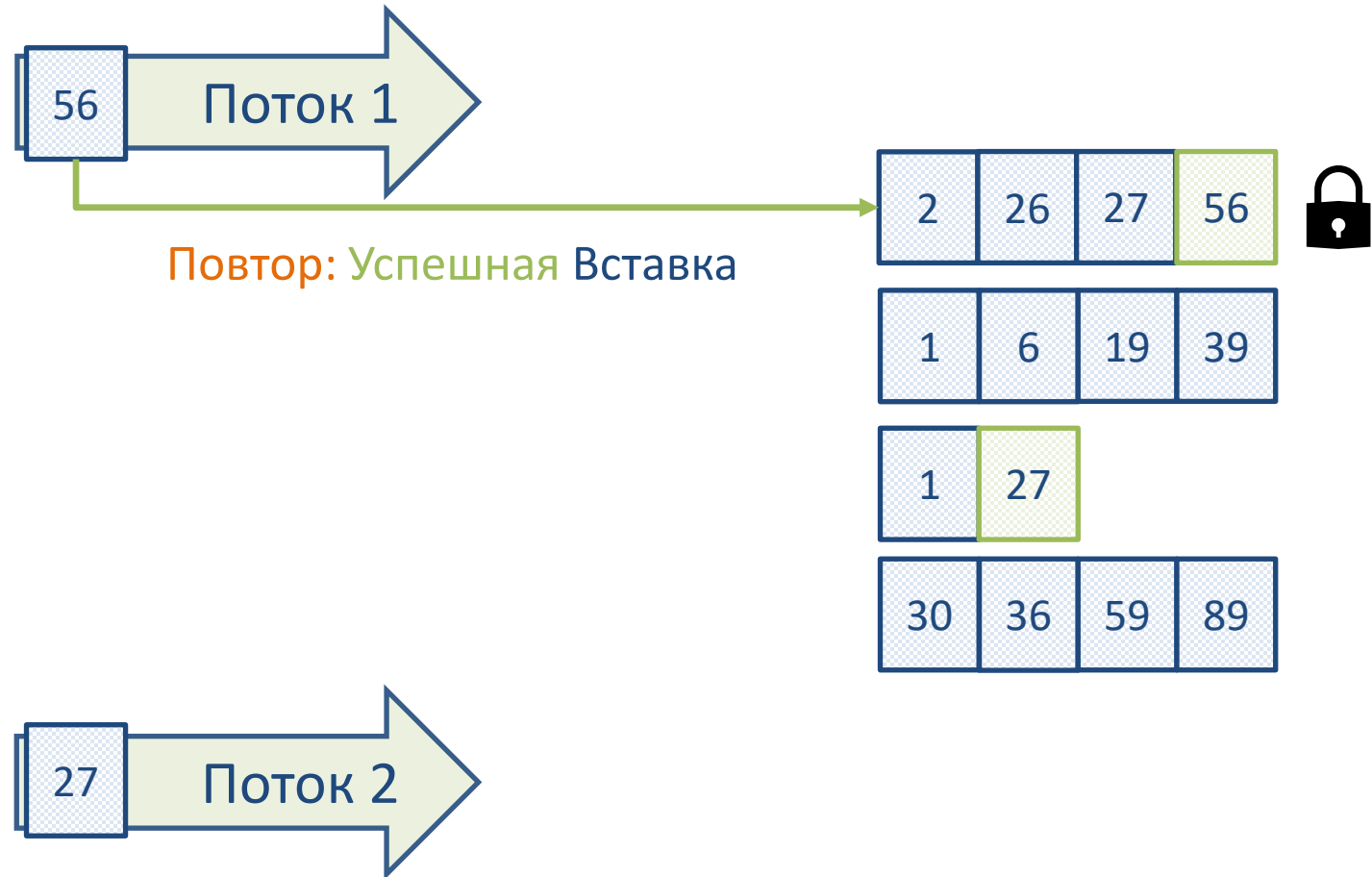
Если поток в данный момент не может заблокировать выбранную очередь, он выбирает другую очередь



ОСЛАБЛЕННАЯ ОЧЕРЕДЬ С ПРИОРИТЕТОМ

Операция вставки

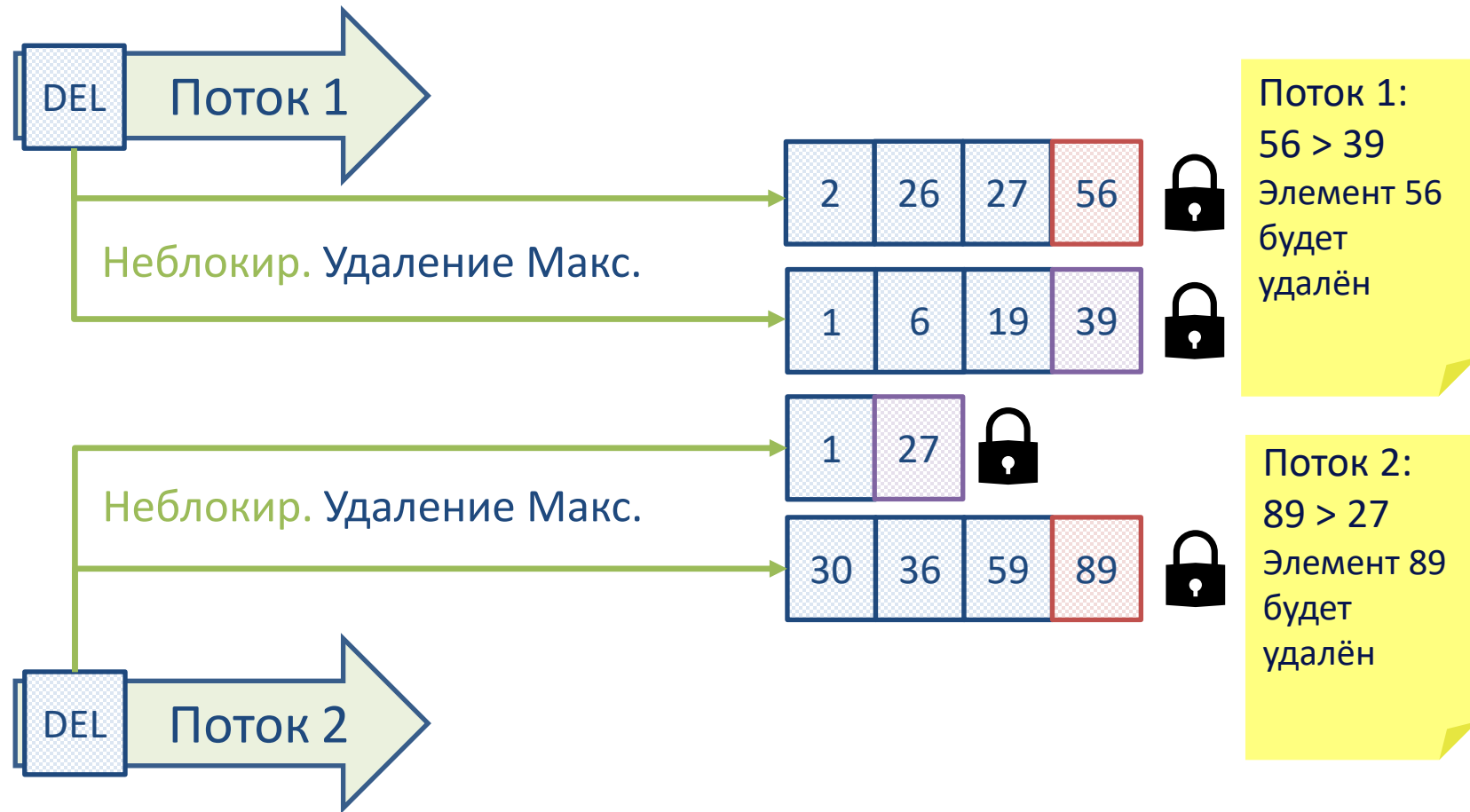
Если поток в данный момент не может заблокировать выбранную очередь, он выбирает другую очередь



ОСЛАБЛЕННАЯ ОЧЕРЕДЬ С ПРИОРИТЕТОМ

Удаление максимального элемента

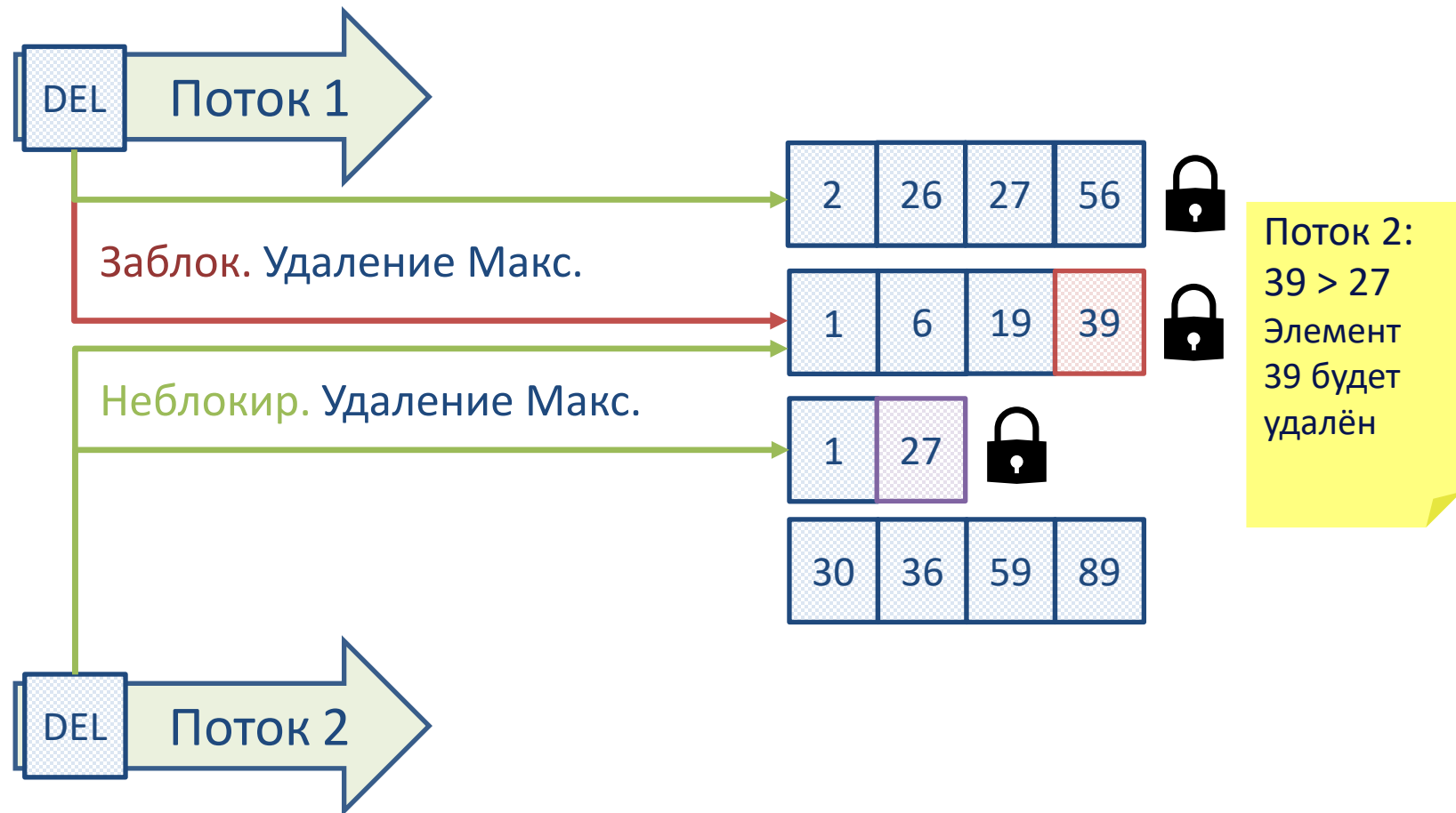
- Поток случайным образом выбирает 2 очереди
- Попытка заблокировать их
- Сравнивает их максимальные элементы
- Удаляет максимальный элемент



ОСЛАБЛЕННАЯ ОЧЕРЕДЬ С ПРИОРИТЕТОМ

Удаление максимального элемента

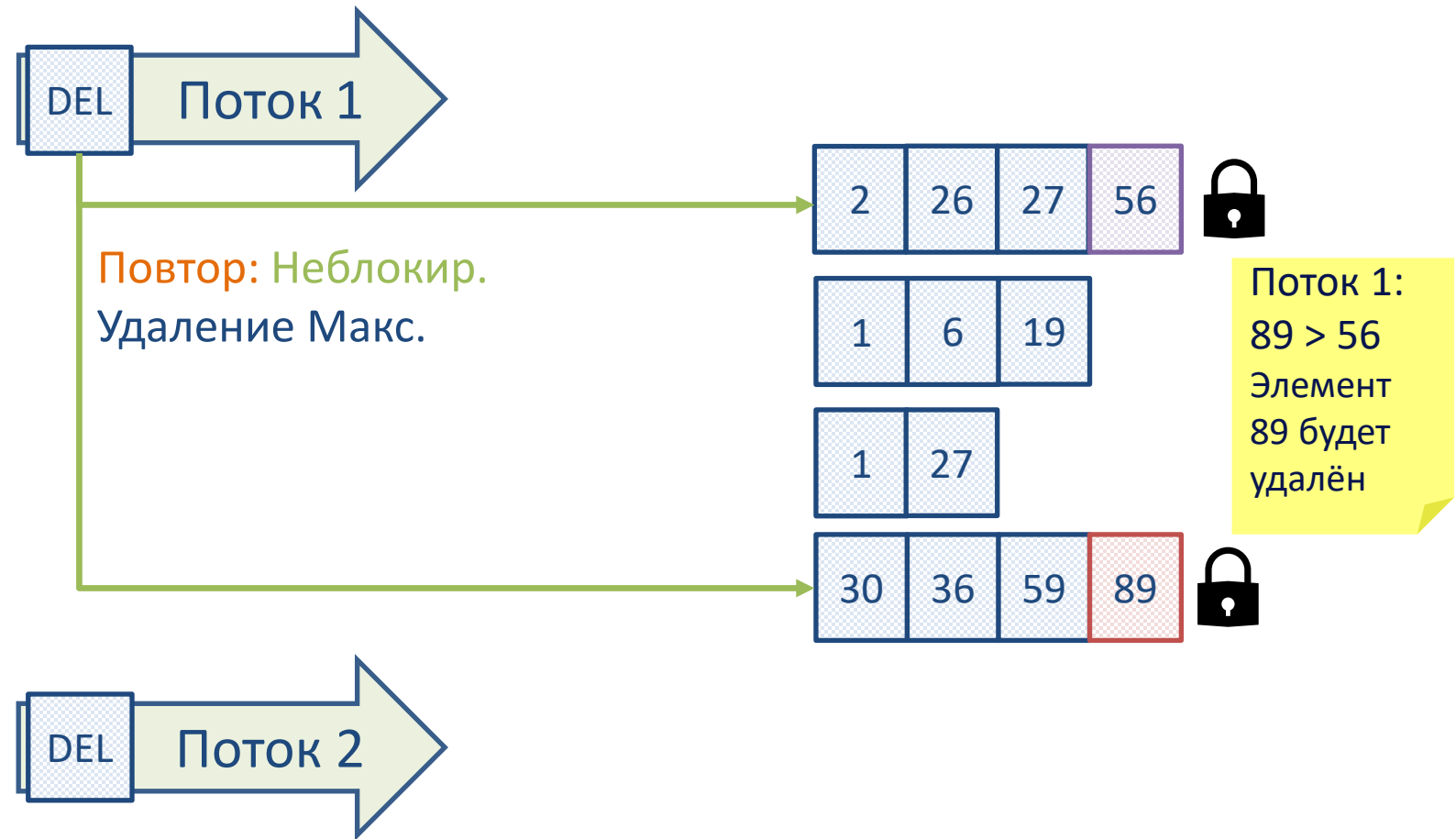
Если поток не может заблокировать одну из выбранных очередей, он выполняет поиск другой подходящей очереди



ОСЛАБЛЕННАЯ ОЧЕРЕДЬ С ПРИОРИТЕТОМ

Удаление максимального элемента

Если поток не может заблокировать одну из выбранных очередей, он выполняет поиск другой подходящей очереди



ОСЛАБЛЕННАЯ ОЧЕРЕДЬ С ПРИОРИТЕТОМ¹⁰

Модель

Ослабленная очередь с приоритетом – является композицией последовательных очередей с приоритетом, защищённые от одновременного доступа блокировками

Нотация:

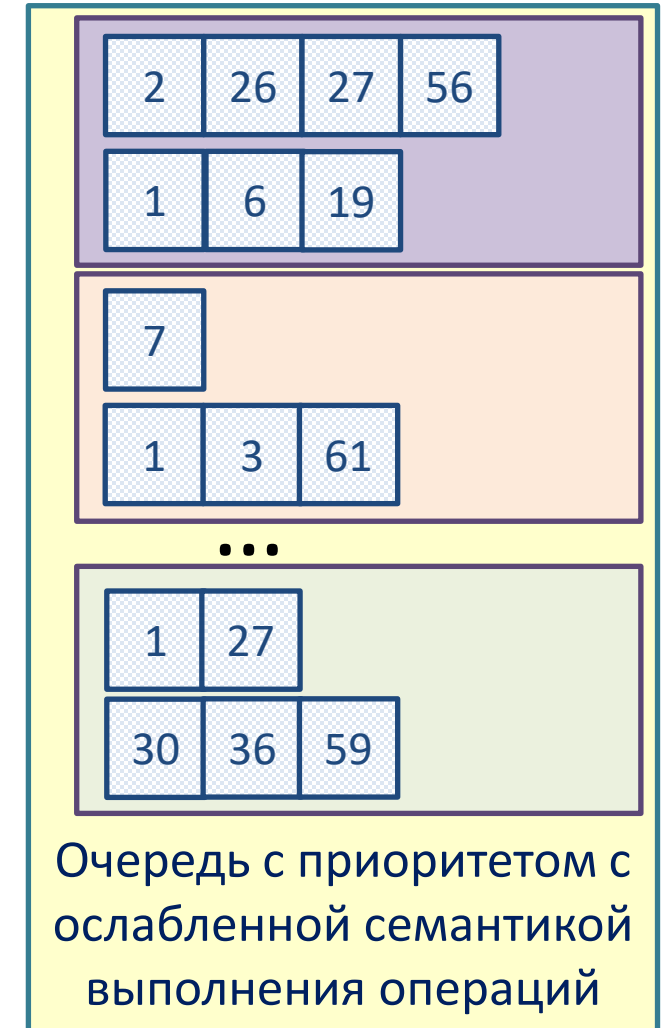
- p – количество потоков
- k – количество очередей на один поток

Поток 1

Поток 2

...

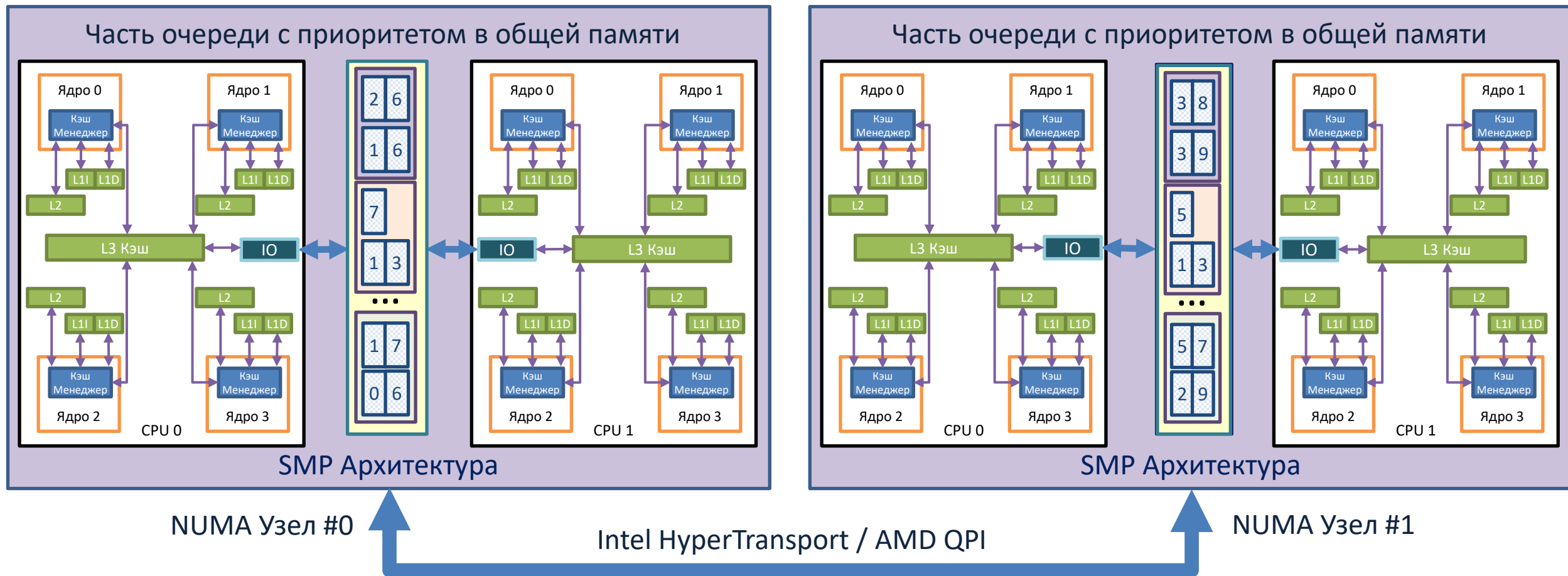
Поток P



¹⁰ Rihani H., Sanders P., Dementiev R. Multiqueues: Simpler, faster, and better relaxed concurrent priority queues //arXiv preprint arXiv:1411.1209. – 2014.

ОСЛАБЛЕННАЯ ОЧЕРЕДЬ С ПРИОРИТЕТОМ

Модель



Часть ослабленной очереди с приоритетом может распределёно находится в общей памяти разных NUMA узлов.

ОПТИМИЗАЦИЯ ОПЕРАЦИИ ВСТАВКИ

ORTHALFINSERT

```
1  do
2    if  $p_i \in \{0, 1 \dots p/2\}$  then
3       $q = \text{RANDQUEUE}(0, kp/2); // q \in \{0, 1 \dots kp\}$ 
4    else
5       $q = \text{RandQueue}(kp/2+1, kp); // q \in \{kp/2+1 \dots kp\}$ 
6    end
7  while TryLock( $q$ ) == false;
8  Insert( $q$ ,  $el$ );
9  Unlock( $q$ );
```

Нотация:

- p – количество потоков
- p_i – *id* текущего потока
- k – количество очередей на один поток
- $q1, q2$ – очереди с приоритетом

ОПТИМИЗАЦИЯ ОПЕРАЦИИ УДАЛЕНИЯ

ORTHALFDELETE

```
1  do
2    if  $p_i \in \{0, 1 \dots p/2\}$  then
3       $(q1, q2) = \text{Rand2Queue}(0, kp/2)$ ;  $//(q1, q2) \in \{0, 1 \dots kp\}$ 
4    else
5       $(q1, q2) = \text{Rand2Queue}(kp/2+1, kp)$ ;  $//(q1, q2) \in \{kp/2+1 \dots kp\}$ 
6    end
7     $q = \text{GetMaxElementQueue}(q1, q2)$ ;
8    while TryLock( $q$ ) == false;
9    DeleteMax( $q$ );
10   Unlock( $q$ );
```

Нотация:

- p – количество потоков
- p_i – *id* текущего потока
- k – количество очередей на один поток
- $q1, q2$ – очереди с приоритетом

ОПТИМИЗАЦИЯ ОПЕРАЦИИ УДАЛЕНИЯ ОРТЕХАСТDELETE

```
1  firstIteration = true;
2  do
3    if firstIteration then
4       $(q1, q2) = \text{Rand2Queue}(p_i, p_i+k); \text{ // } (q1, q2) \in \{p_i, p_i+1 \dots p_i+k\}$ 
5    else
6       $(q1, q2) = \text{Rand2Queue}(0, kp); \text{ // } (q1, q2) \in \{0, 1 \dots kp\}$ 
7    end
8    firstIteration = false;
9     $q = \text{GetMaxElementQueue}(q1, q2);$ 
10 while  $\text{TryLock}(q) == \text{false};$ 
11  $\text{DeleteMax}(q);$ 
12  $\text{Unlock}(q);$ 
```

Нотация:

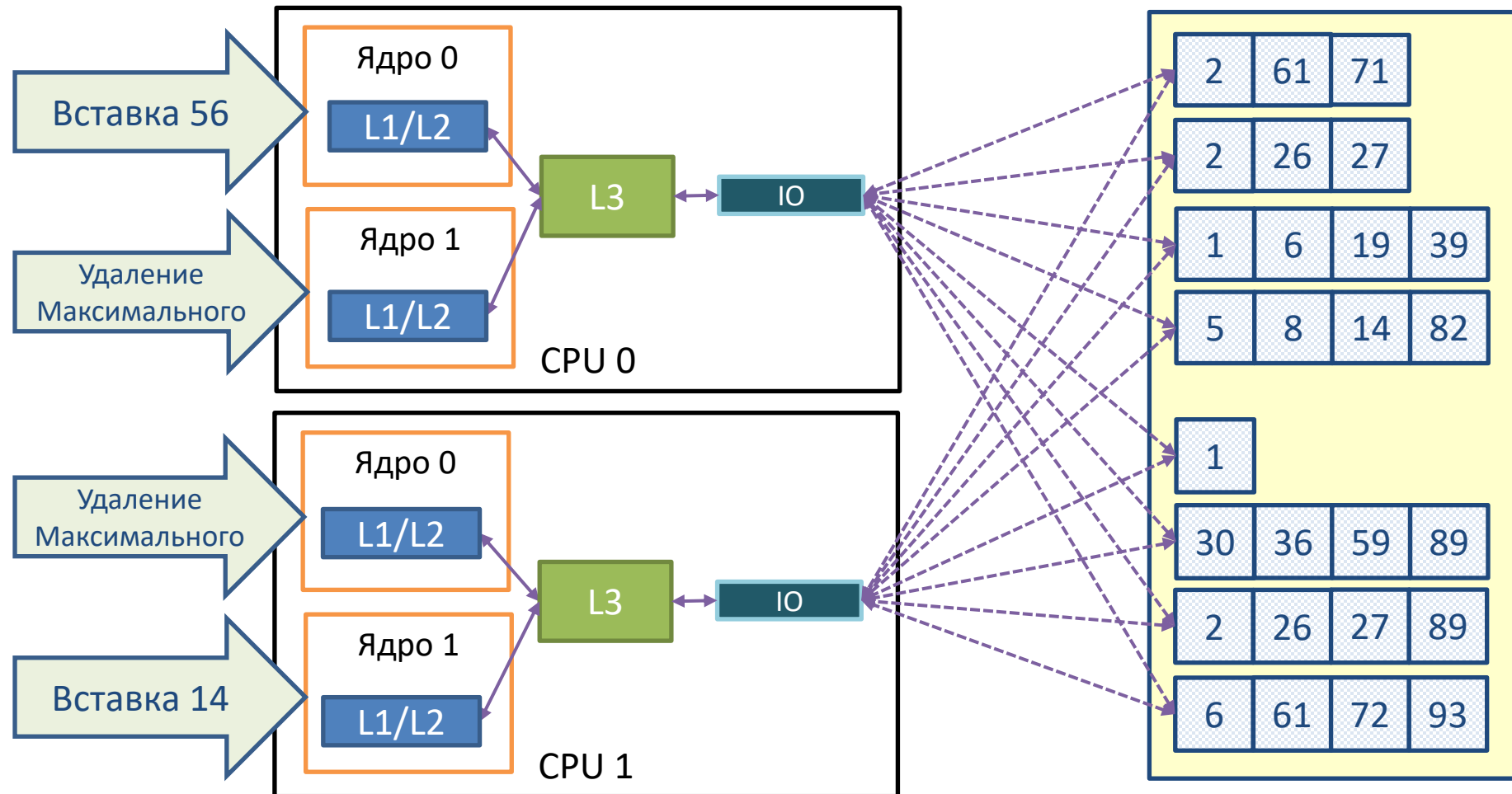
- p – количество потоков
- p_i – *id* текущего потока
- k – количество очередей на один поток
- $q1, q2$ – очереди с приоритетом

ОСЛАБЛЕННАЯ ОЧЕРЕДЬ С ПРИОРИТЕТОМ

Метод выбора очереди

- Поток выбирает случайную очередь¹

- Далее выполняет операцию с ней

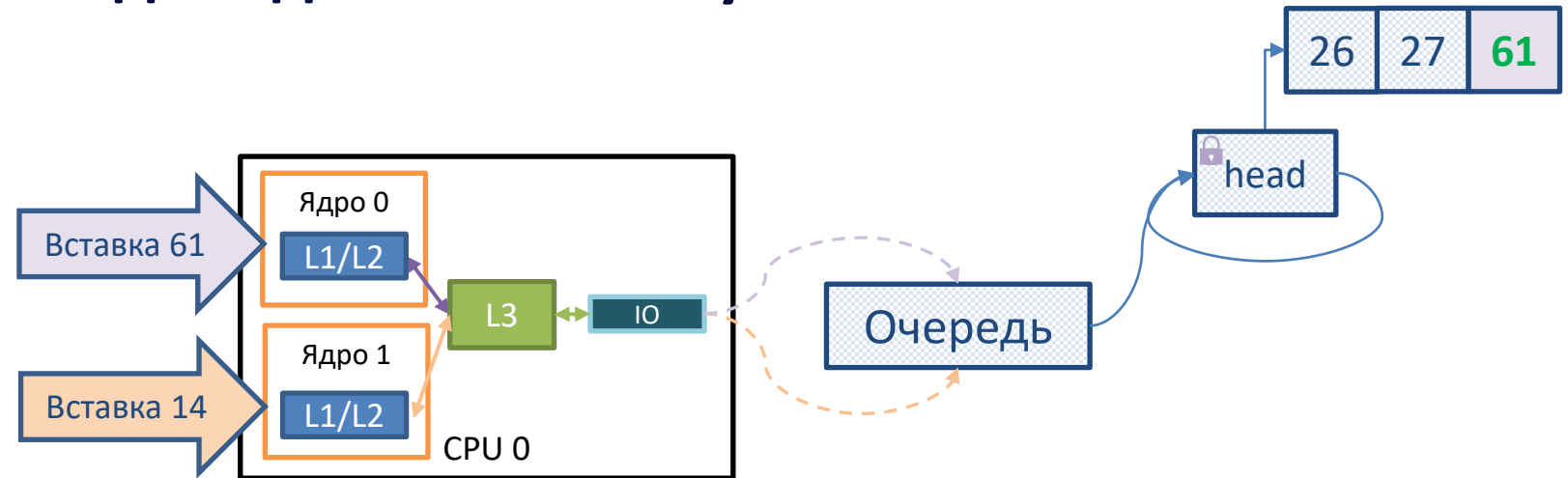


¹Rihani H., Sanders P., Dementiev R. Multiqueues: Simpler, faster, and better relaxed concurrent priority queues //arXiv preprint arXiv:1411.1209. – 2014.

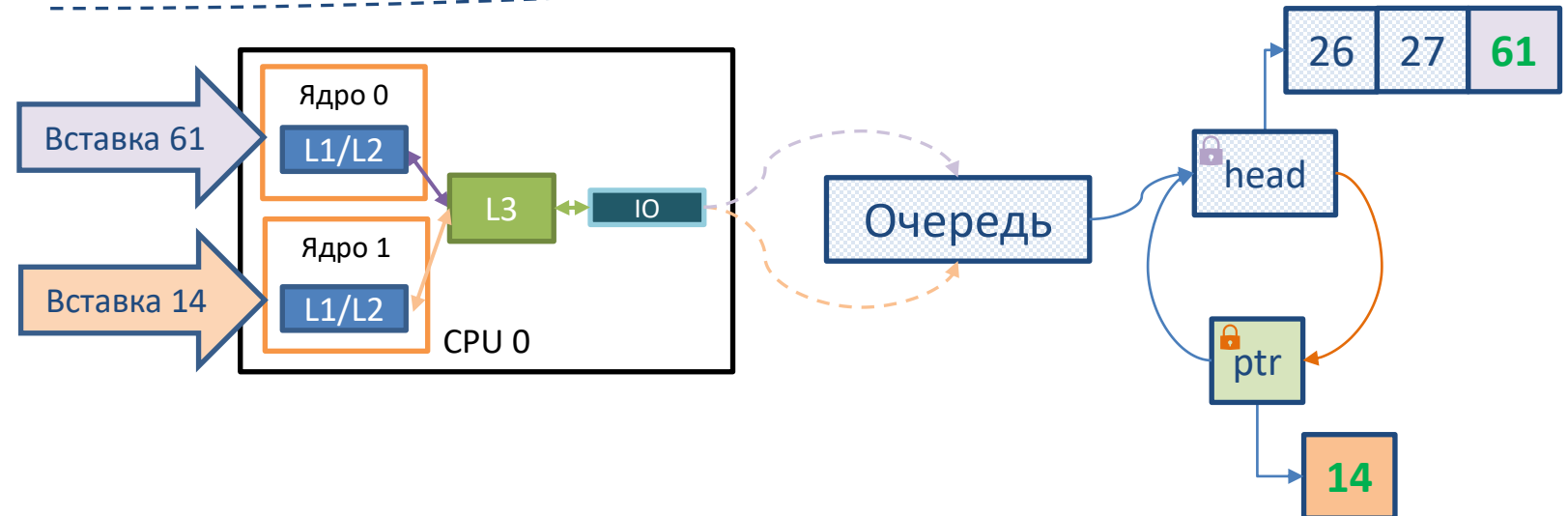
ЦИКЛИЧЕСКАЯ ОСЛАБЛЕННАЯ ОЧЕРЕДЬ С ПРИОРИТЕТОМ (CRQ)

Метод создания нового узла

Изначально циклический список имеет только один узел указывающий на структуру данных



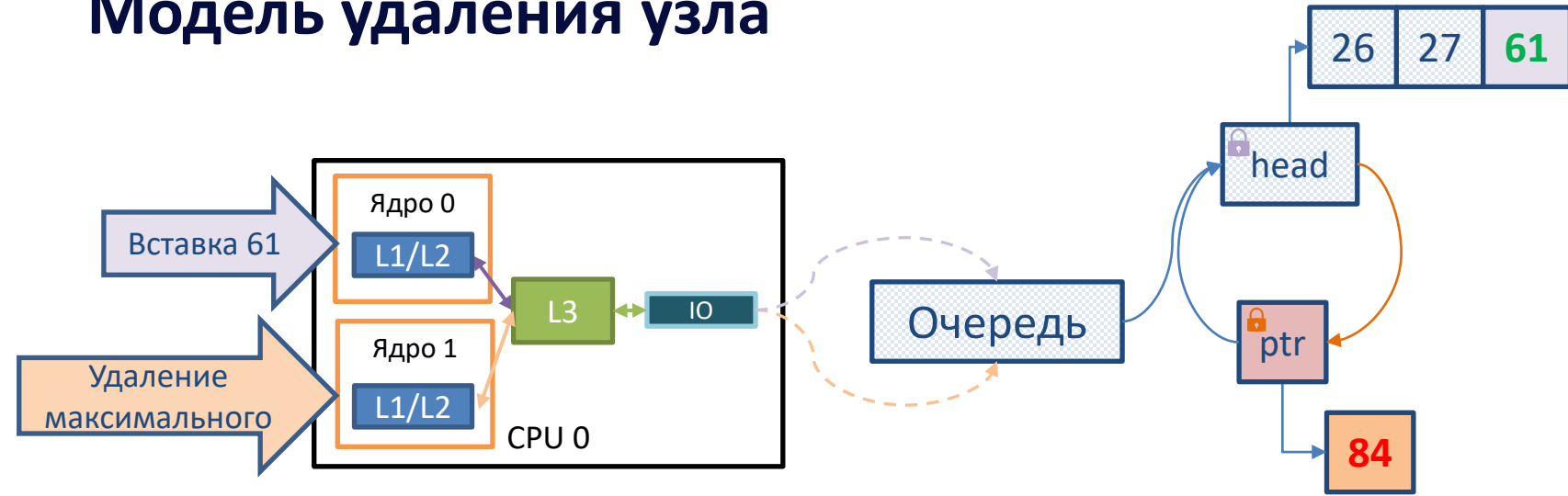
Если двум потокам требуется одновременный доступ на запись в данную структуру, то один из потоков заблокирует доступный узел, в то же время, другой не найдёт доступный ему узел и создаст новый



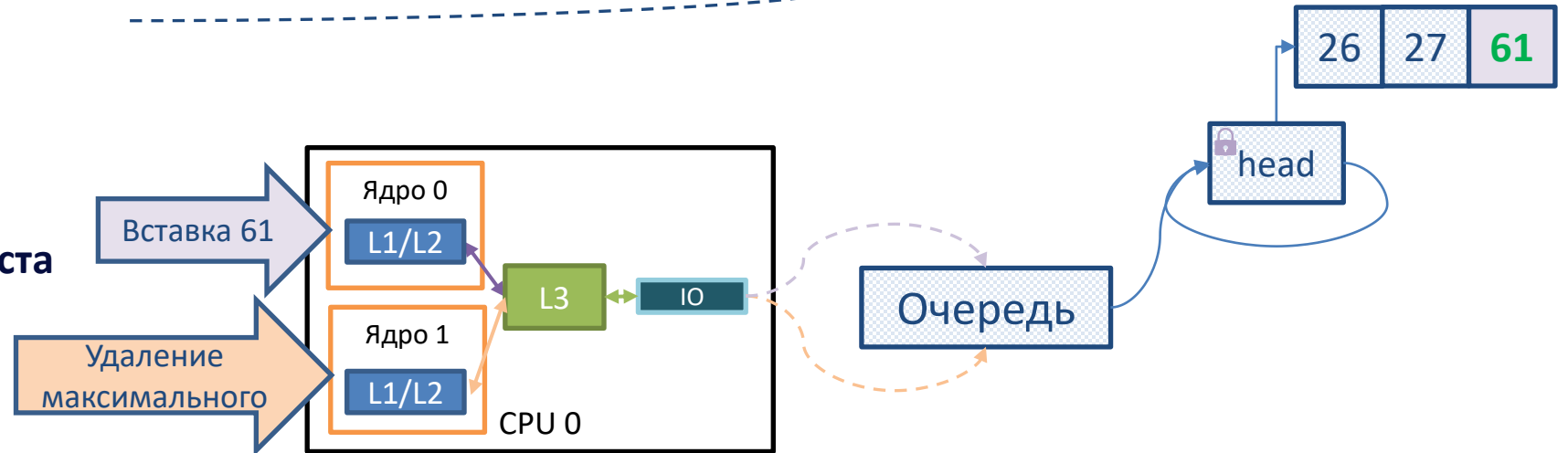
ЦИКЛИЧЕСКАЯ ОСЛАБЛЕННАЯ ОЧЕРЕДЬ С ПРИОРИТЕТОМ (CRQ)

Модель удаления узла

Когда структура данных в узле становится пустой - узел удаляется

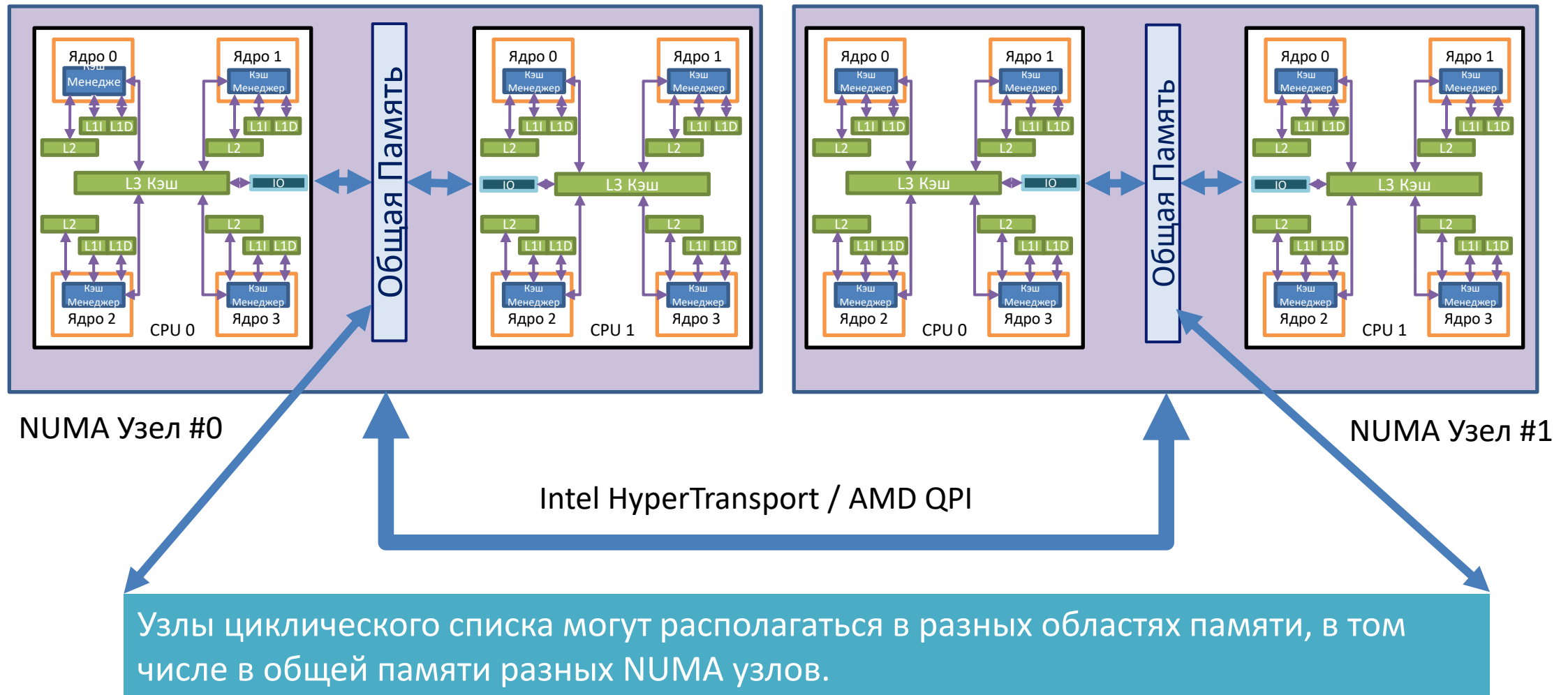


Данная операция удаления необходима для увеличения производительности поиска подходящих узлов и экономии места

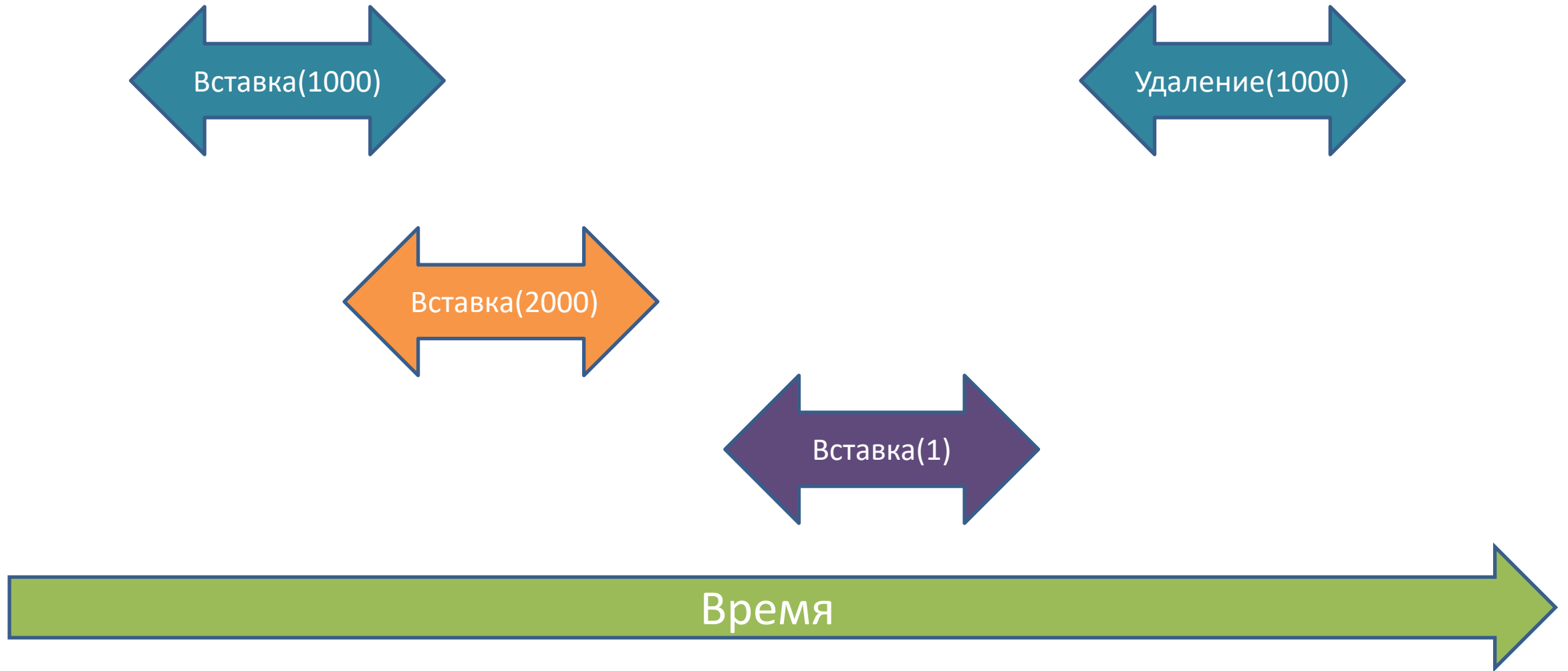


ЦИКЛИЧЕСКАЯ ОСЛАБЛЕННАЯ ОЧЕРЕДЬ С ПРИОРИТЕТОМ (CRQ)

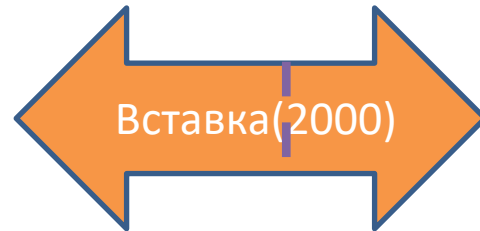
Модель



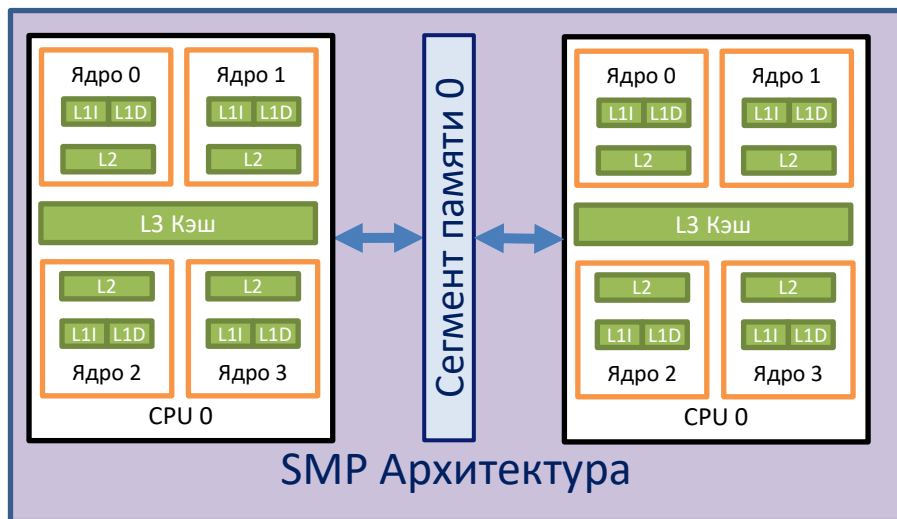
ЛИНЕАРИЗУЕМОСТЬ



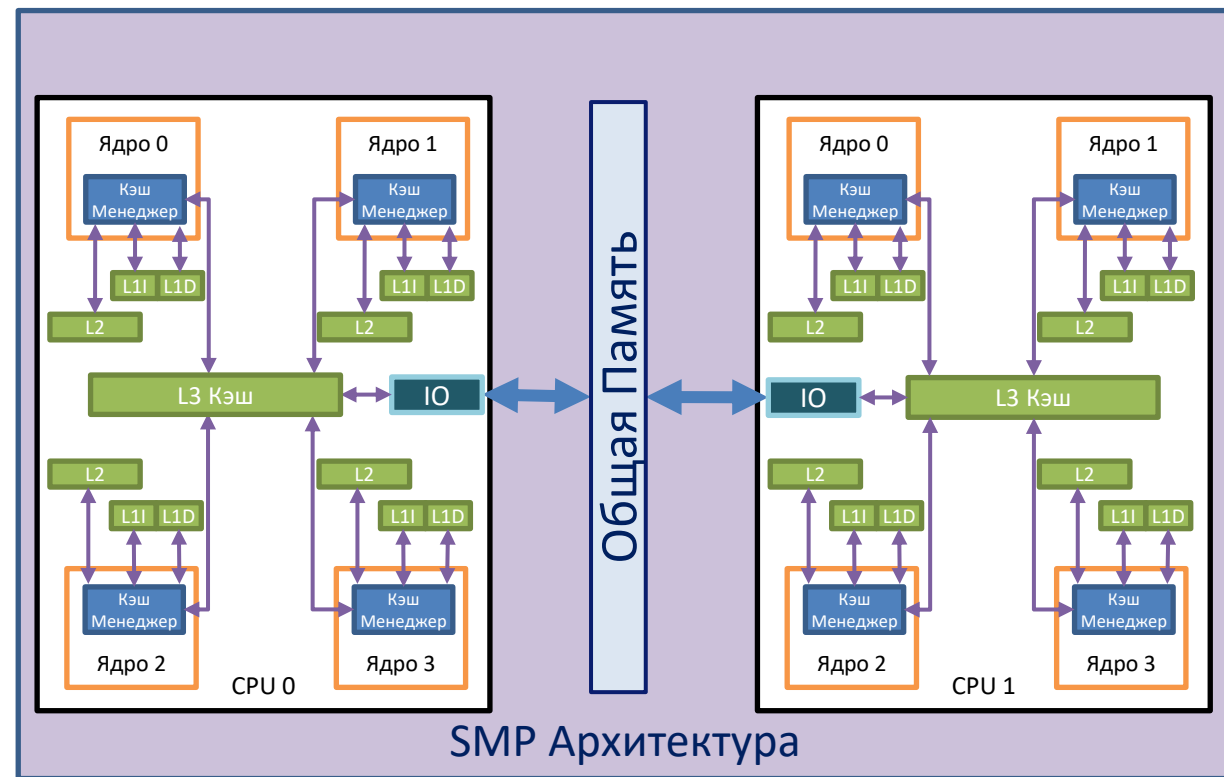
ЛИНЕАРИЗУЕМОСТЬ



МНОГОЯДЕРНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ



NUMA Узел #0



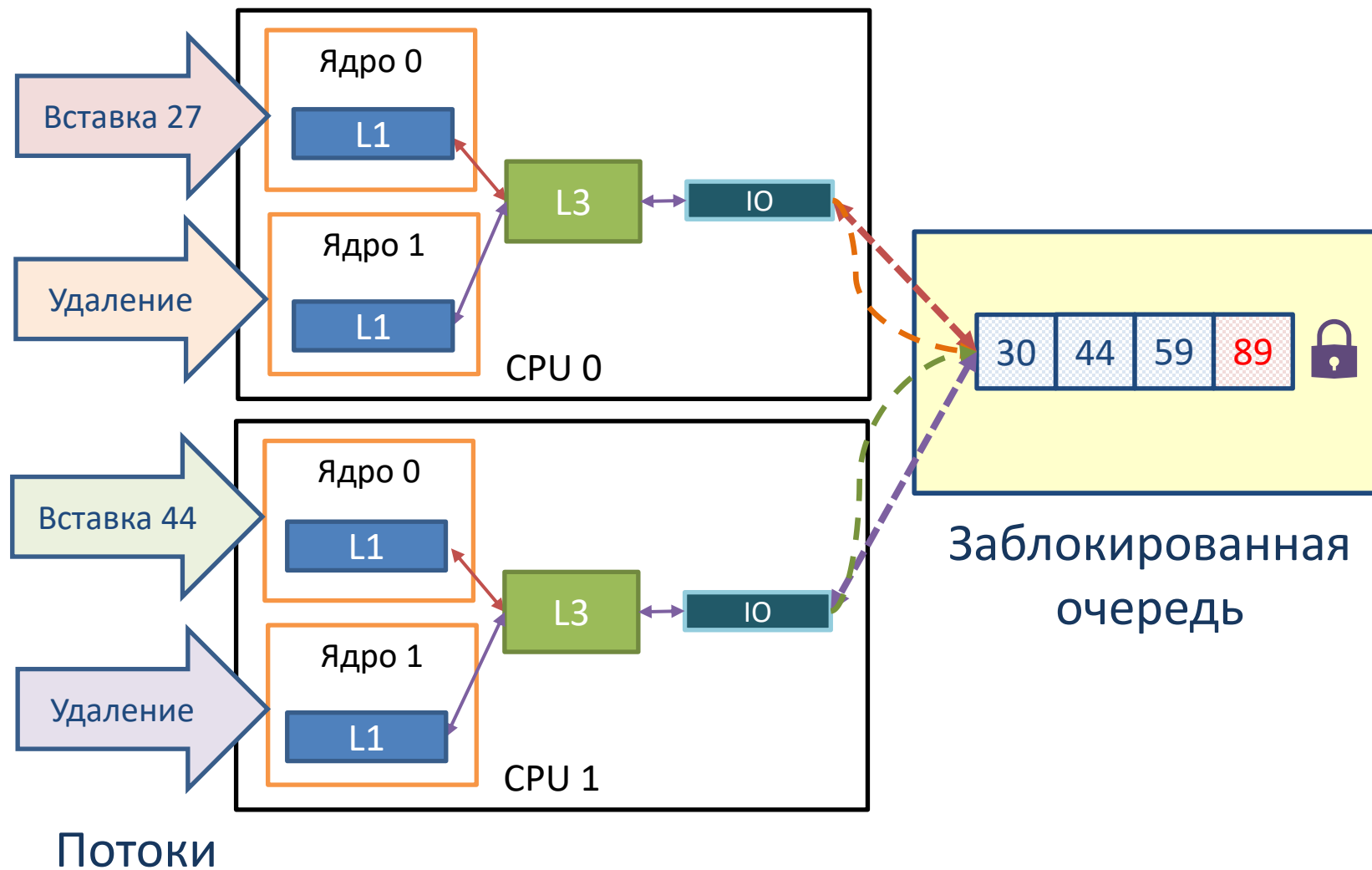
NUMA Узел #1

Intel QPI / AMD HyperTransport

NUMA Архитектура

ПОСЛЕДОВАТЕЛЬНЫЙ СТЕК

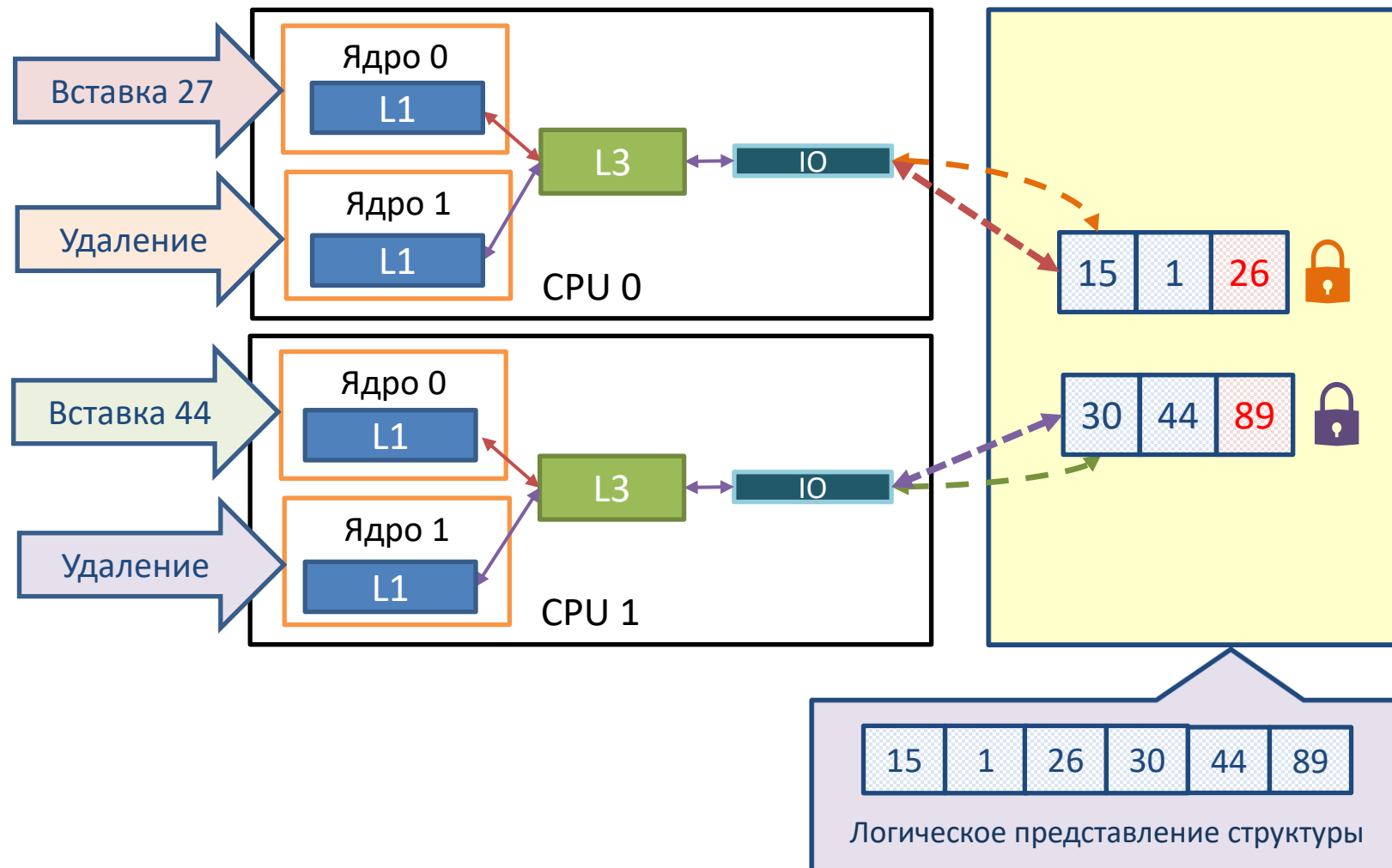
- Потоки имеют единую точку выполнения операций
- Невозможно выполнить действия параллельно



ОСЛАБЛЕННЫЙ НА k ЭЛЕМЕНТОВ СТЕК⁷

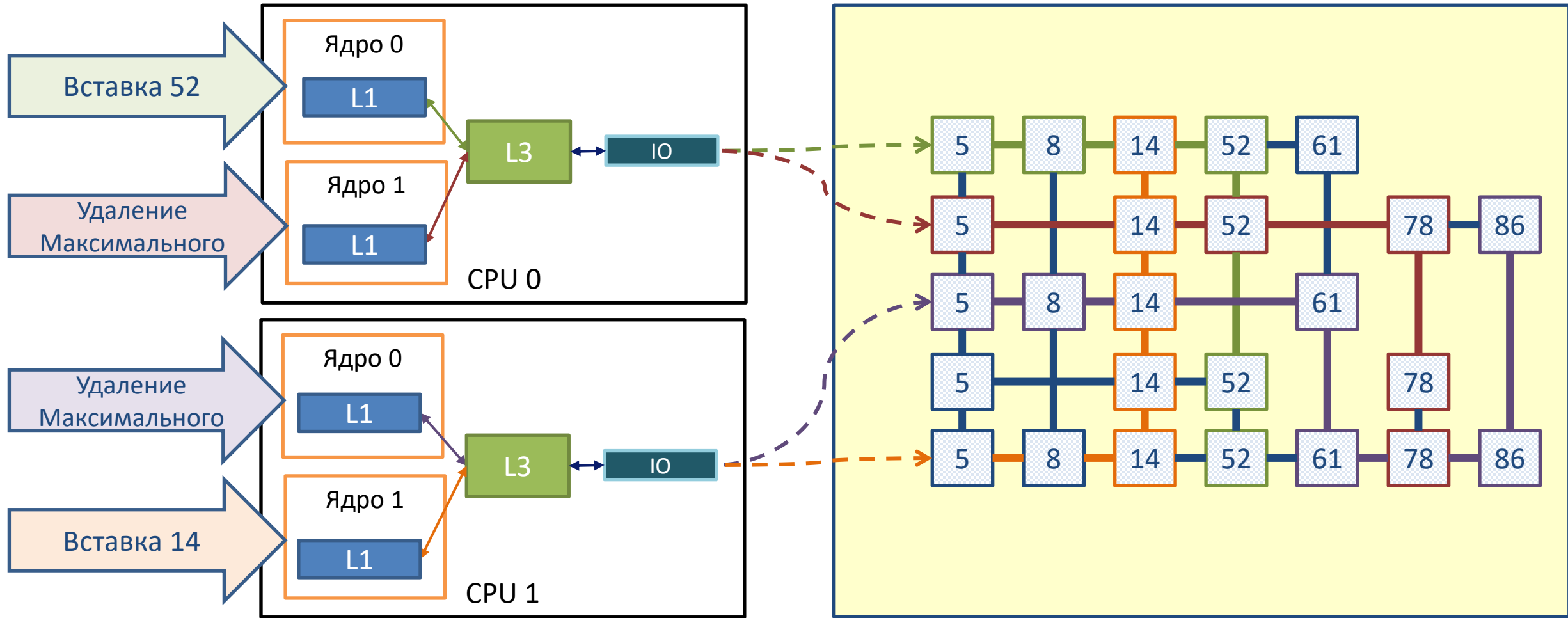
k-RELAXED STACK

- Потоки имеют множество точек выполнения операций
- Логически структура представляет единое целое
- Результат выполнения непредсказуем



⁷ Talmage E., Welch J. L. Improving average performance by relaxing distributed data structures //International Symposium on Distributed Computing. – Springer, Berlin, Heidelberg, 2014. – С. 421-438

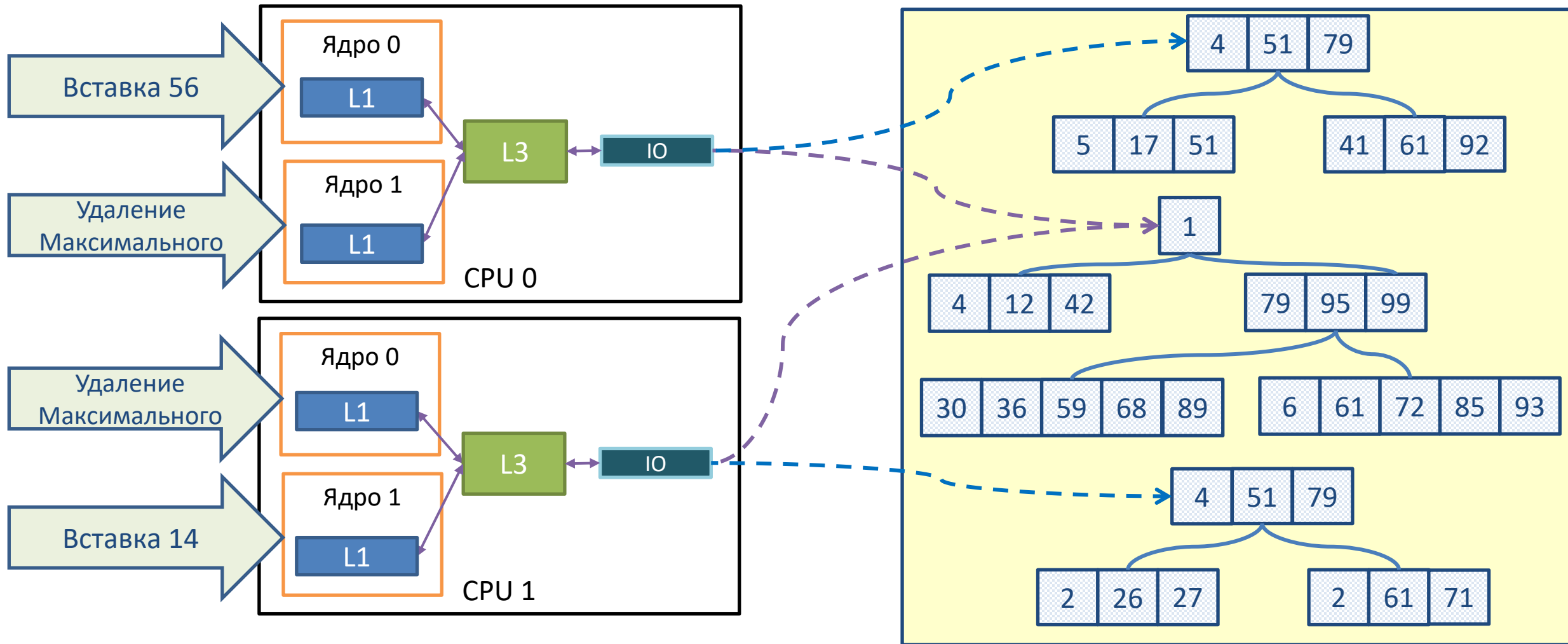
К⁸ SprayList



⁸ Alistarh D. et al. The spraylist: A scalable relaxed priority queue // Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. – 2015. – С. 11-20.

Журнально-структурированное дерево со слиянием⁹

k-LSM



⁹ Wimmer M. et al. The lock-free k-LSM relaxed priority queue //ACM SIGPLAN Notices. – 2015. – T. 50. – №. 8. – C. 277-278.

ЗАКЛЮЧЕНИЕ

- Реализованы оптимизированные алгоритмы вставки и удаления максимального (минимального) элемента для ослабленной структуры данных Multiqueues
- Предложен подход создания ослабленных структур данных с использованием узлов циклического списка
- Разработана структура данных на основе предложенного подхода

Реализация Multiqueues		Реализация Circular Relaxed Concurrent Priority Queue	
C++	Kotlin	C++	Kotlin
Github.com /Komdosh/Multiqueues	Github.com /Komdosh/KotMultiqueues	Github.com /Komdosh/CircularPriorityQueue	Github.com /Komdosh/RelaxedCycleDS
