Netcracker
An NEC Company

Apache Cassandra
Потоки и память

О себе

- Дмитрий Константинов
- Системный архитектор и, по-прежнему, практикующий Java разработчик в компании Netcracker ☺

О себе

- Дмитрий Константинов
- Системный архитектор и, по-прежнему, практикующий Java разработчик в компании Netcracker ☺
- Активно работаю с различными OpenSource технологиями, такими как Apache Cassandra, Zookeeper, Kafka, Hazelcast
- Профессиональные интересы:
  - распределенные системы
  - производительность
  - отказоустойчивость

Наш опыт

- Несколько Cassandra кластеров в production
  - От 3 до 180 узлов в кластере
- Как open source версии, так и DSE
- Несколько datacenters
- Типы системы: как OLTP, так и offline batch
- ~ 5k - 200k запросов в секунду на кластер

О чем вообще говорим

- Какие проблемы и задачи, связанные с потоками и памятью, решались в Apache Cassandra

О чем вообще говорим

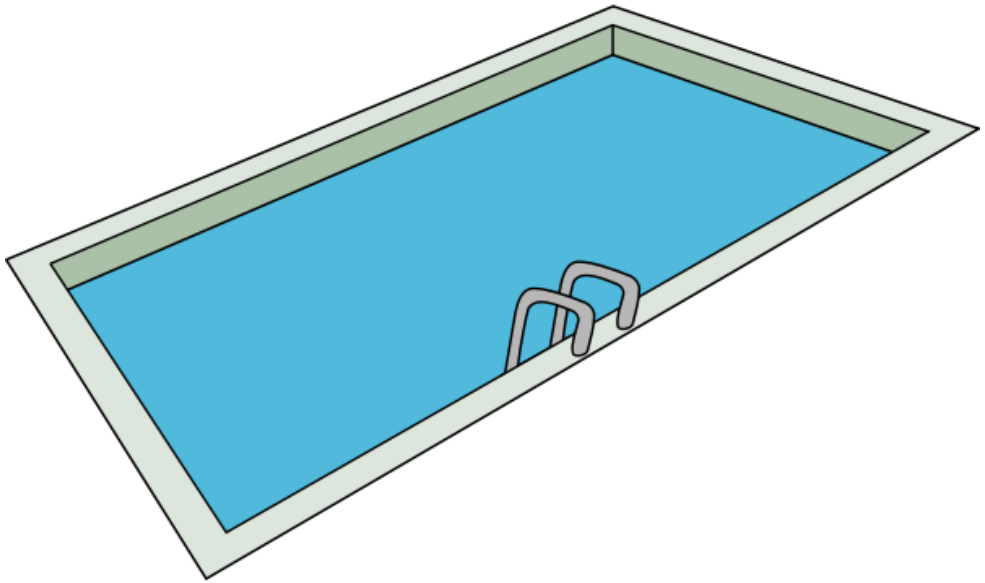
- Какие проблемы и задачи, связанные с потоками и памятью, решались в Apache Cassandra (решены не все ☺)

О чем вообще говорим

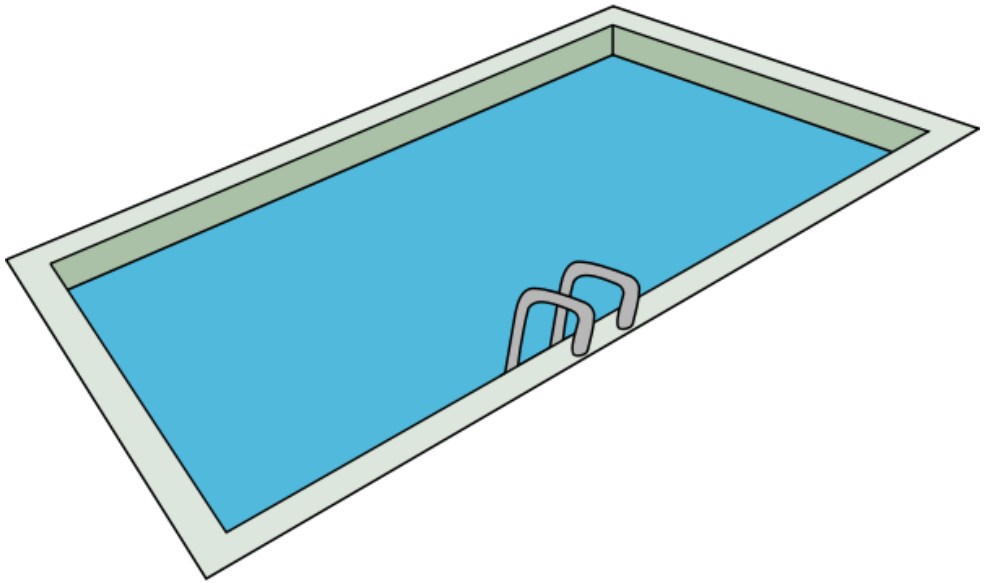
- Какие проблемы и задачи, связанные с потоками и памятью решались в Apache Cassandra (решены не все ☺)
- Какие приемы, утилиты, идеи могут быть использованы и в других приложениях

- Как отслеживать состояние потоков?

+



+



Thread pool

Потоки в Cassandra

- ScheduledThreadPoolExecutor
- ThreadPoolExecutor
- Netty event loop
- Shared pool executor

- Как понять их загрузку?

Thread pool monitoring – SJK

- https://github.com/aragozin/jvm-tools (Swiss Java Knife, SJK)

Алексей Рагозин

Thread pool monitoring – SJK

- https://github.com/aragozin/jvm-tools (Swiss Java Knife, SJK)
- SJK is integrated to Cassandra 4.0 (CASSANDRA-12197)
- ttop – cpu usage + allocation per thread

Thread pool monitoring – SJK

- https://github.com/aragozin/jvm-tools (Swiss Java Knife, SJK)
- SJK is integrated to Cassandra 4.0 (CASSANDRA-12197)
- ttop – cpu usage + allocation per thread

nodetool sjk ttop

```
process cpu=613.33%
  application cpu=608.06% (user=564.71% sys=43.35%)
  other: cpu=5.27%
  thread count: 132
  heap allocation rate 1830mb/s
[000903] user=96.45% sys= 1.64% alloc=  296mb/s - STREAM-IN-/10.4.140.127:7000
[000609] user=92.95% sys= 3.09% alloc=  318mb/s - STREAM-IN-/10.4.140.130:7000
[000490] user=92.56% sys= 3.33% alloc=  324mb/s - STREAM-IN-/10.4.140.128:7000
[000601] user=91.79% sys= 3.89% alloc=  317mb/s - STREAM-IN-/10.4.140.129:7000
[000852] user=88.29% sys= 4.16% alloc=  276mb/s - STREAM-IN-/10.4.140.131:7000
[000607] user=42.00% sys= 2.18% alloc=  131mb/s - CompactionExecutor:3
[000613] user=39.67% sys= 2.70% alloc=  136mb/s - CompactionExecutor:4
```

Thread pool monitoring – virtual tables

- Virtual tables – new monitoring capability, added in Cassandra 4.0
- CASSANDRA-14523 - Thread pool stats virtual table

Thread pool monitoring – virtual tables

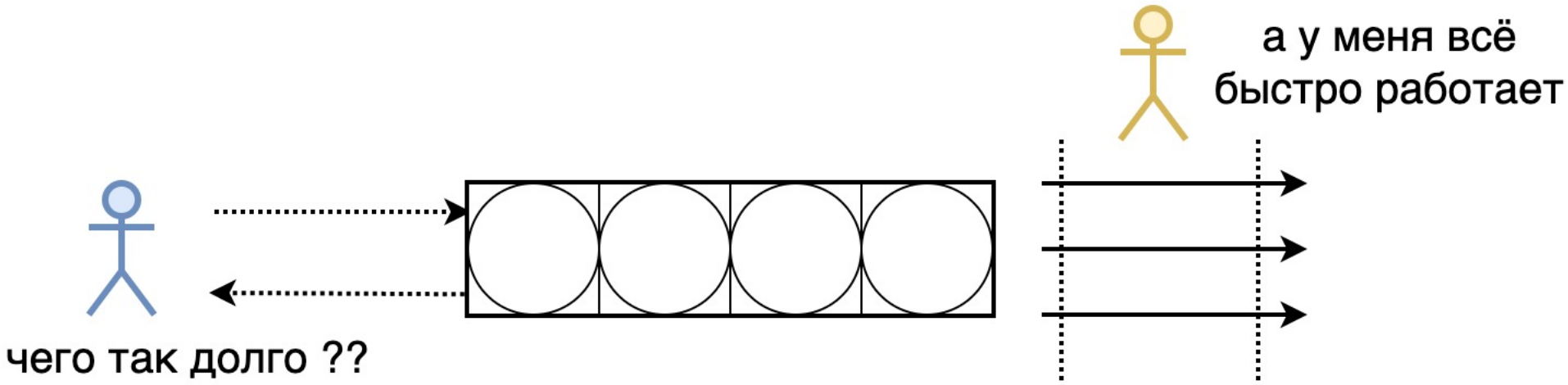
- Virtual tables – new monitoring capability, added in Cassandra 4.0
- CASSANDRA-14523 - Thread pool stats virtual table

```
cqlsh> select * from system_views.thread_pools ;
```

name	active_tasks	active_tasks_limit	blocked_tasks	blocked_tasks_all_time	completed_tasks	pending_tasks
CacheCleanupExecutor	0	1	0	0	0	0
CompactionExecutor	0	2	0	0	94	0
GossipStage	0	1	0	0	1143	0
HintsDispatcher	0	2	0	0	0	0
MemtableFlushWriter	0	2	0	0	25	0
MemtablePostFlush	0	1	0	0	71	0
MemtableReclaimMemory	0	1	0	0	25	0
MigrationStage	0	1	0	0	10	0
MutationStage	2	32	0	0	1753773	2
Native-Transport-Requests	7	128	0	0	584268	0
PendingRangeCalculator	0	1	0	0	6	0
PerDiskMemtableFlushWriter_0	0	2	0	0	25	0
ReadStage	0	32	0	0	110	0
RequestResponseStage	0	20	0	0	1168248	0
Sampler	0	1	0	0	0	0
SecondaryIndexManagement	0	1	0	0	0	0
ValidationExecutor	0	2	0	0	0	0
ViewBuildExecutor	0	1	0	0	0	0

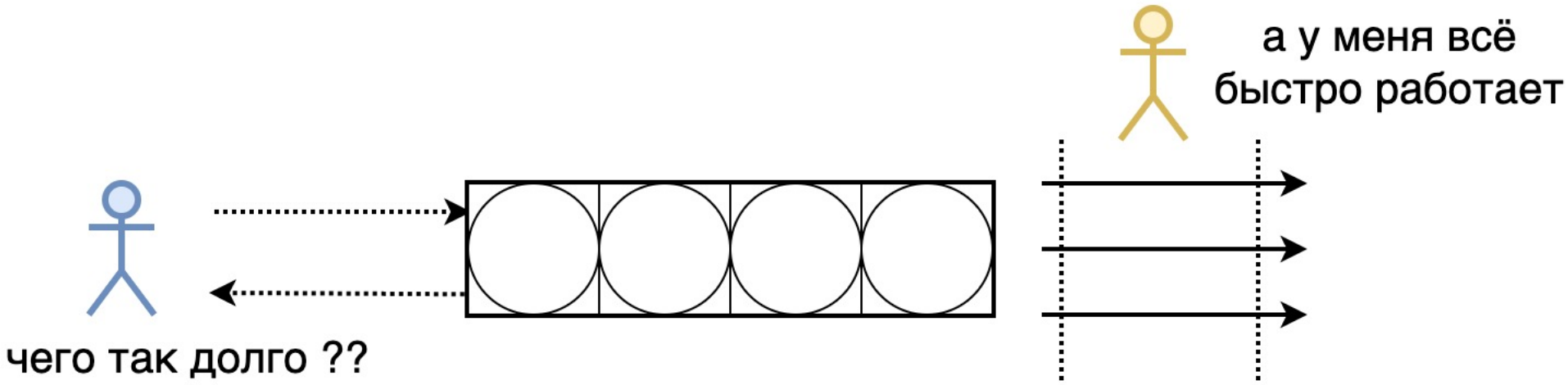
Queue time

- Операция с точки зрения внешнего мира операция выполняется долго, а время выполнения задачи в потоке – небольшое…



Queue time

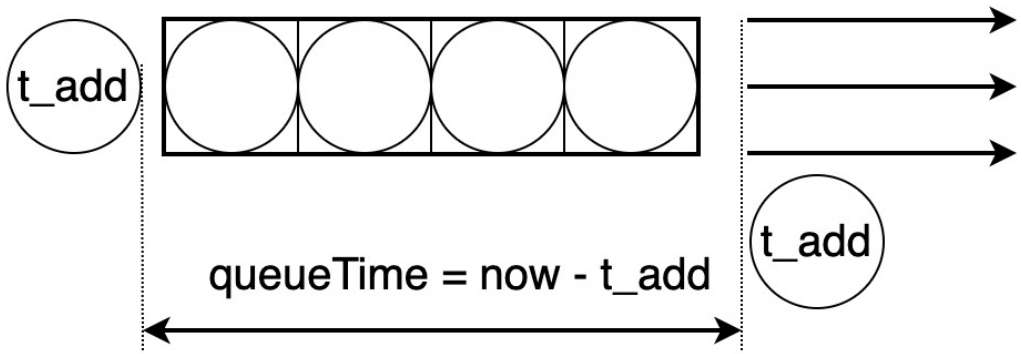
- Операция с точки зрения внешнего мира операция выполняется долго, а время выполнения задачи в потоке – небольшое…
- CASSANDRA-8398 (Expose time spent waiting in thread pool queue)



Queue time

```
Executor executor = Executors.newFixedThreadPool(10);

public void executeTask(final Runnable runnable) {
    long scheduledAtNs = getTime();
    executor.execute(() -> {
        long executionStartedAtNs = getTime();
        long queueTimeNs = executionStartedAtNs - scheduledAtNs;
        queueTime.update(queueTimeNs, TimeUnit.NANOSECONDS);
        runnable.run();
    });
}
```



- Выполнение периодических задач

ScheduledThreadPoolExecutor

Используется для различного рода периодически выполняемых задач, например:

- Trigger periodic Memtable flush
- Log slow operations
- Dynamic snitch update
- Dropped messages logging
- Trigger hints dispatch
- Unused connections monitoring

Типичная ловушка в scheduled executor

- ScheduledThreadPoolExecutor Javadoc : “If any execution of the task encounters an exception, subsequent executions are suppressed”

Типичная ловушка в scheduled executor

- ScheduledThreadPoolExecutor Javadoc : “If any execution of the task encounters an exception, subsequent executions are suppressed”

```
public void run(). {
    try {
        runnable.run();
    }
    catch (Throwable t) {
        JVMStabilityInspector.inspectThrowable(t);
        DebuggableThreadPoolExecutor.handleOrLog(t);
    }
}
```

- Как реализовать таймеры?

Таймеры

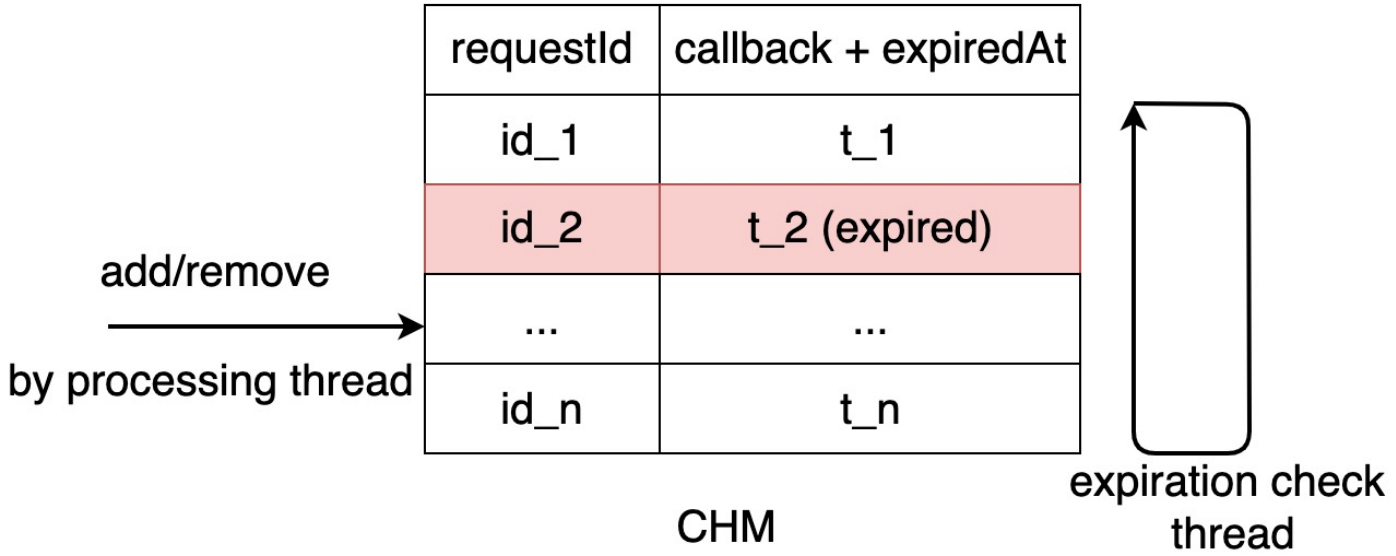
- При асинхронной обработке запросов нужно отслеживать timeout
- Таймауты одинаковые для класса запросов
- Таймер, операции:
  - Add(key) – в начале обработки запроса
  - Remove(key) – при успешной обработке запроса (нет срабатывания)
  - Trigger: вызов callback по таймауту

Таймеры – сканируемая CHM

- Таймауты для in-progress запросов, порядок и высокая точность срабатывания не требуются
- ConcurrentHashMap<Key, Expired>

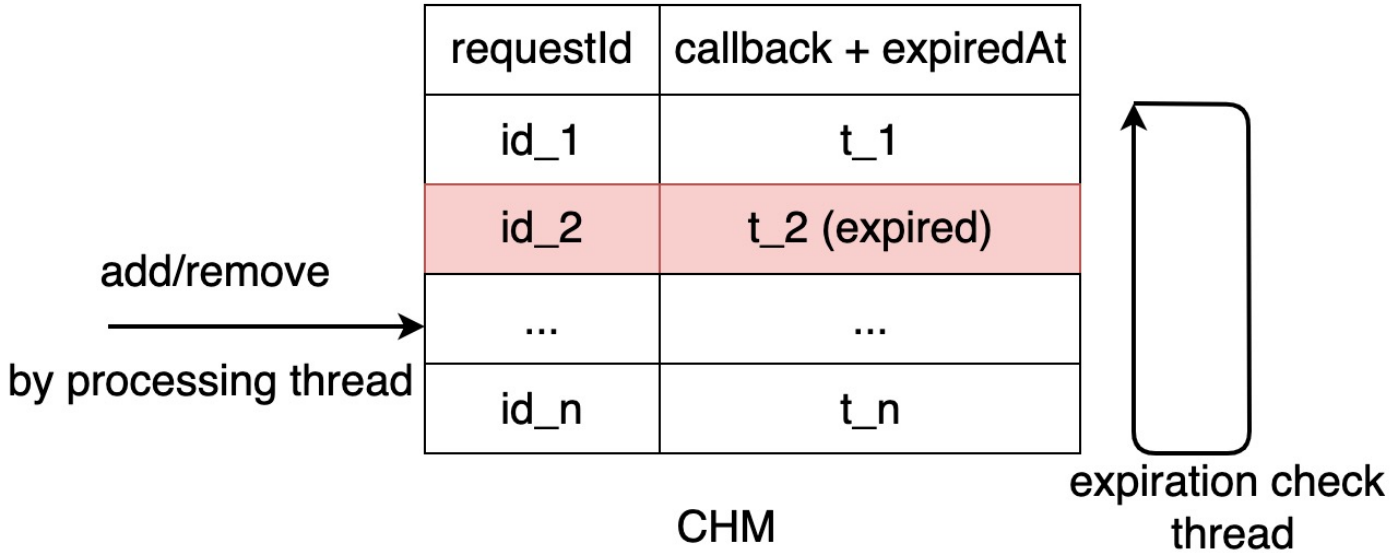
Таймеры – сканируемая CHM

- Таймауты для in-progress запросов, порядок и высокая точность срабатывания не требуются
- ConcurrentHashMap<Key, Expired>
- Expired = timeout + callback
- Периодически (min timeout / 2) поток “Callback-Map-Reaper” обходит все элементы Map, удаляет и вызывает callback для истекших запросов



Таймеры – сканируемая CHM

- Таймауты для in-progress запросов, порядок и высокая точность срабатывания не требуются
- ConcurrentHashMap<Key, Expired>
- Expired = timeout + callback
- Периодически (min timeout / 2) поток “Callback-Map-Reaper” обходит все элементы Map, удаляет и вызывает callback для истекших запросов
- Стоимость операций:
  - Add(key): O(1)
  - Remove(key): O(1)
  - Trigger: O(N)

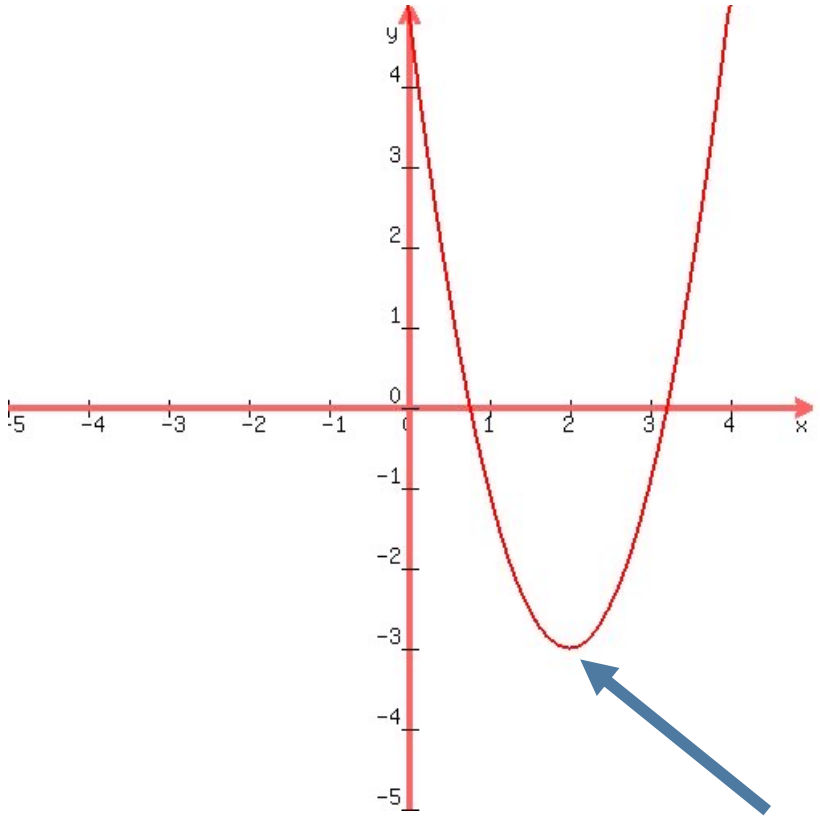


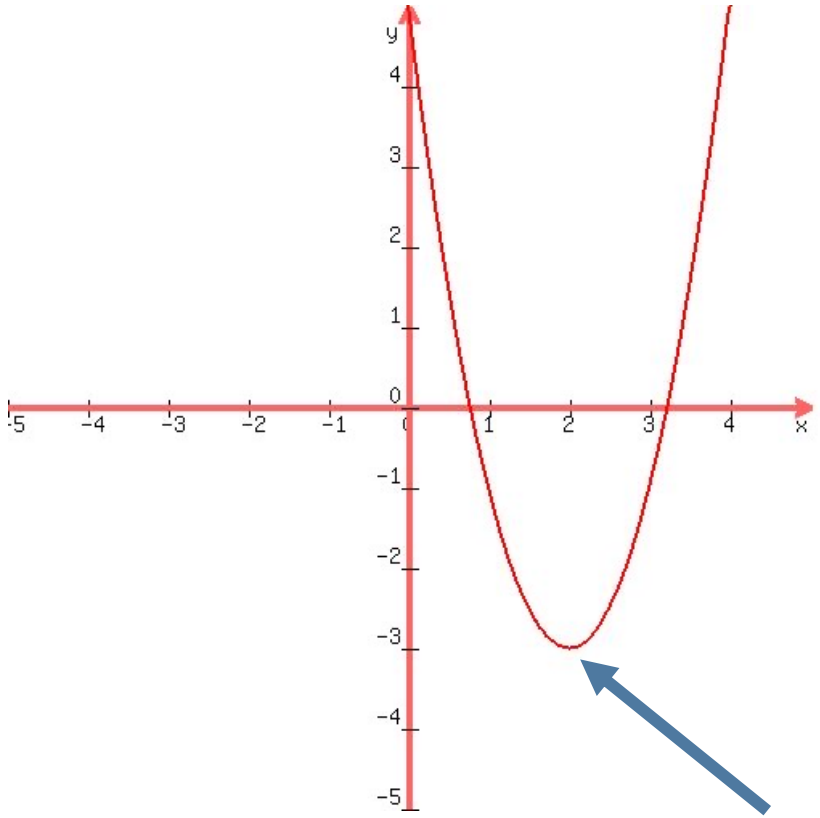
Таймеры

А если:

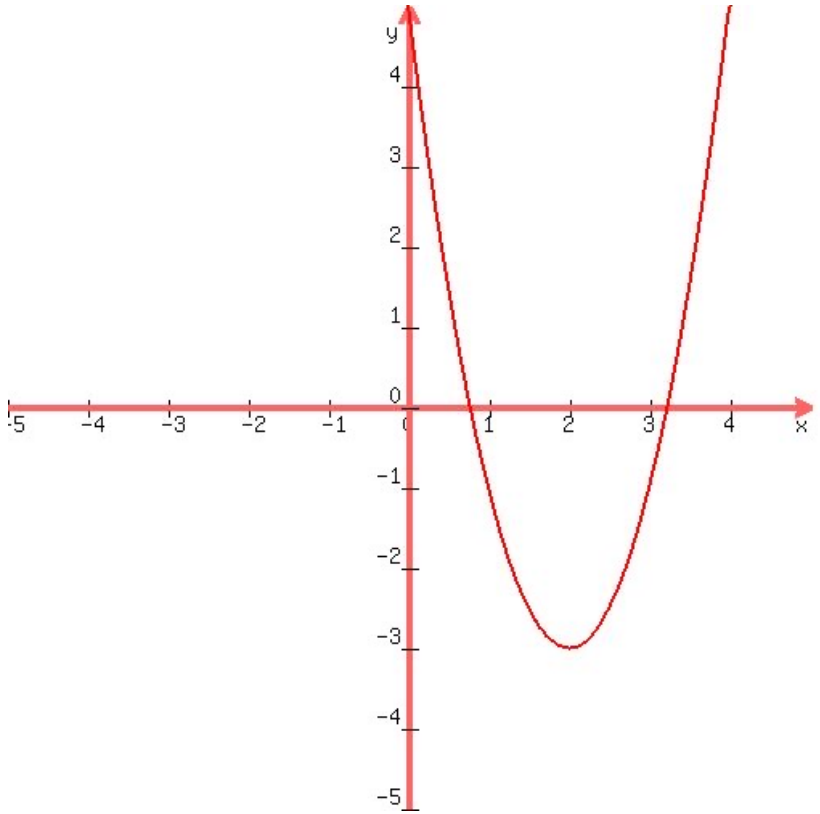
- задач много, и большинство надо выполнить
- хотим обрабатывать в правильном порядке

что делать?





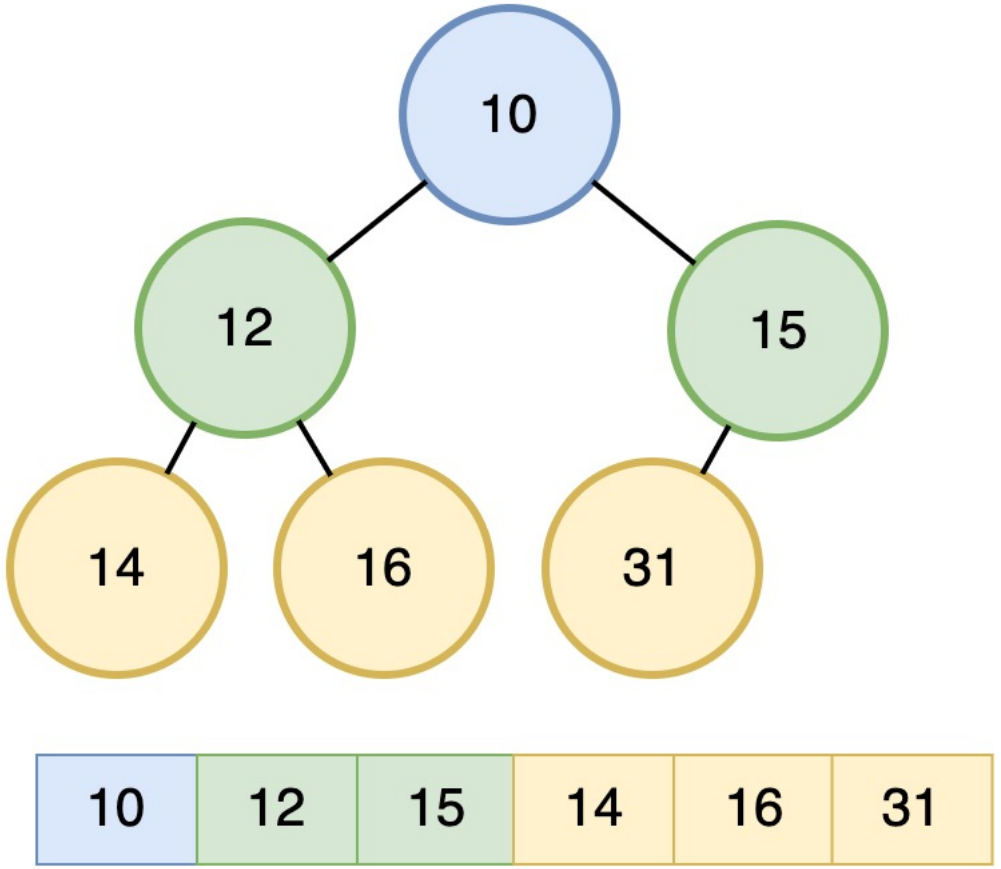
Min-??



Min-heap

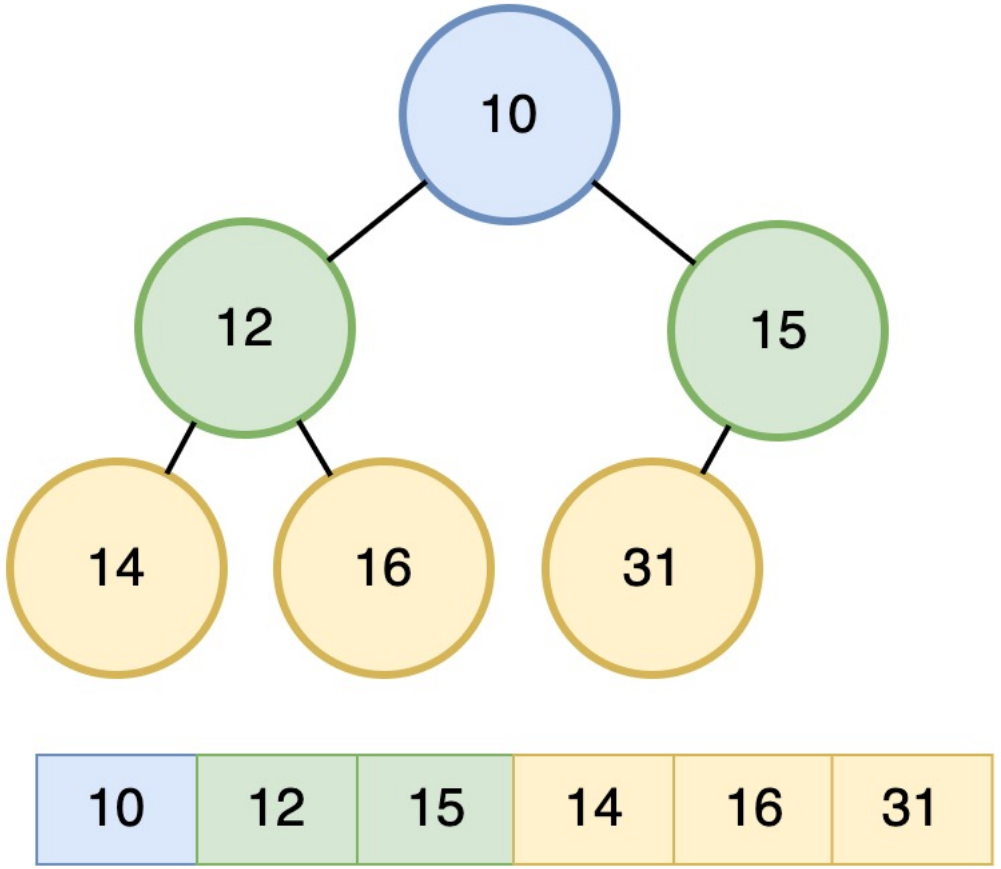
Таймеры - JDK Scheduled thread pool executor

- Бинарная min куча (ScheduledThreadPoolExecutor.DelayWorkQueue)



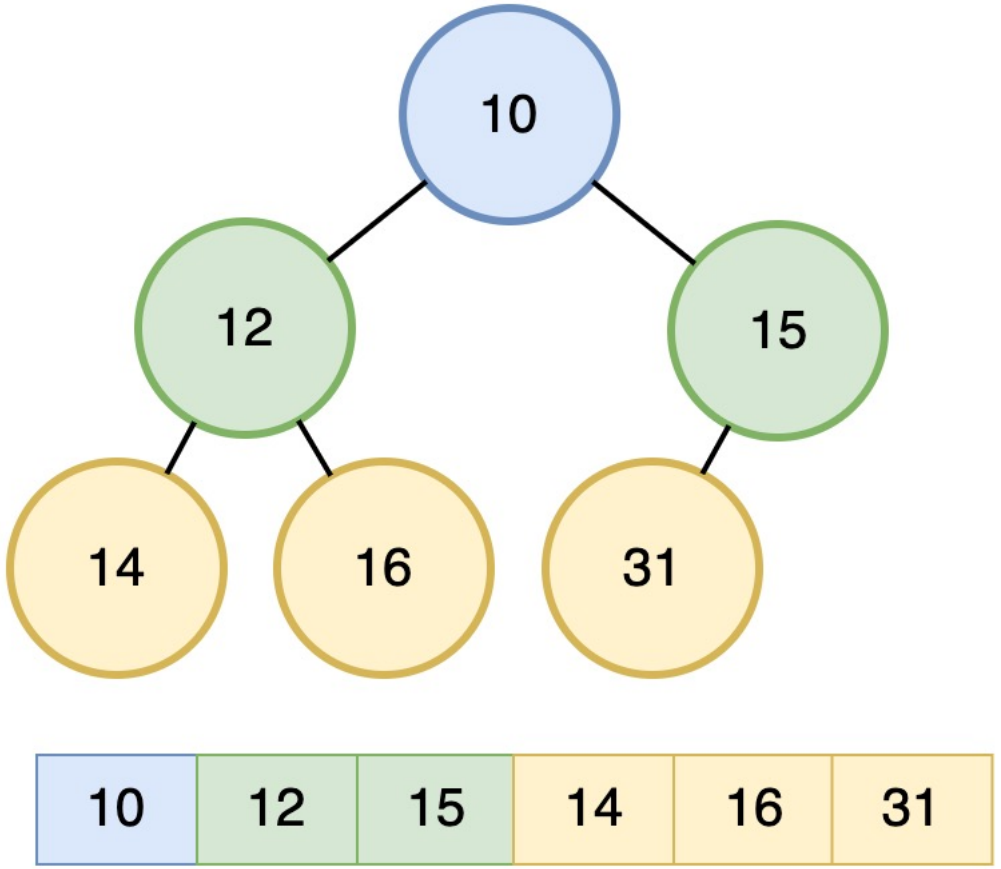
Таймеры - JDK Scheduled thread pool executor

- Бинарная min куча (ScheduledThreadPoolExecutor.DelayWorkQueue)
- Стоимость операций:
  - Add(key): O(log N)
  - Trigger: O(log N)



Таймеры - JDK Scheduled thread pool executor

- Бинарная min куча (ScheduledThreadPoolExecutor.DelayWorkQueue)
- Стоимость операций:
  - Add(key): O(log N)
  - Trigger: O(log N)
  - Remove(key): O(1) или O(log N)
- Про удаление:
  - 2 режима - setRemoveOnCancelPolicy
  - Отличается от PriorityQueue – ScheduledFuture содержит индекс в heap для быстрого поиска и удаления; удаление в PriorityQueue стоит O(n)



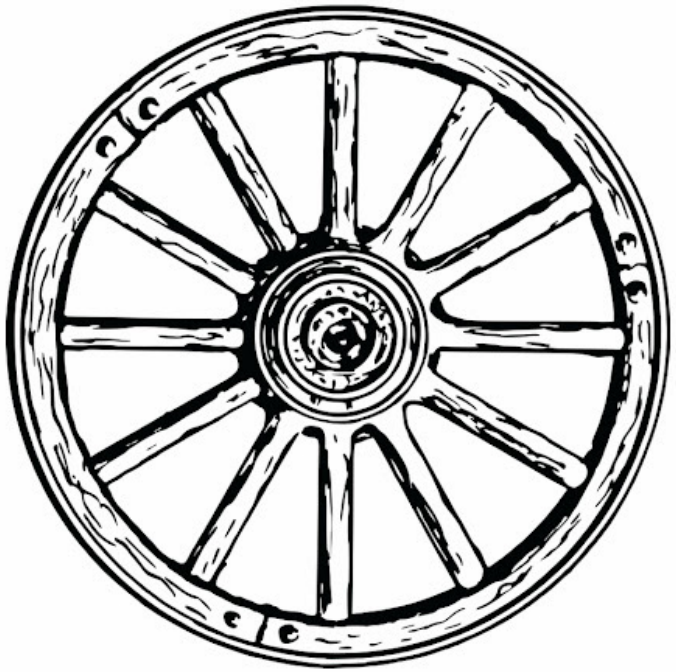
Таймеры

А если – нужно организовать тайм-ауты:

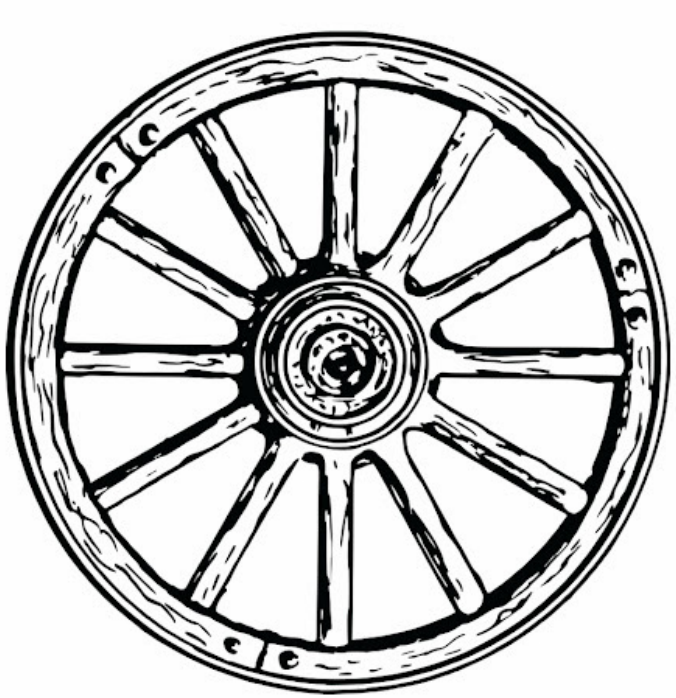
- задач много, и большинство отменяются
- высокая точность и порядок не важны

есть еще опции?

#

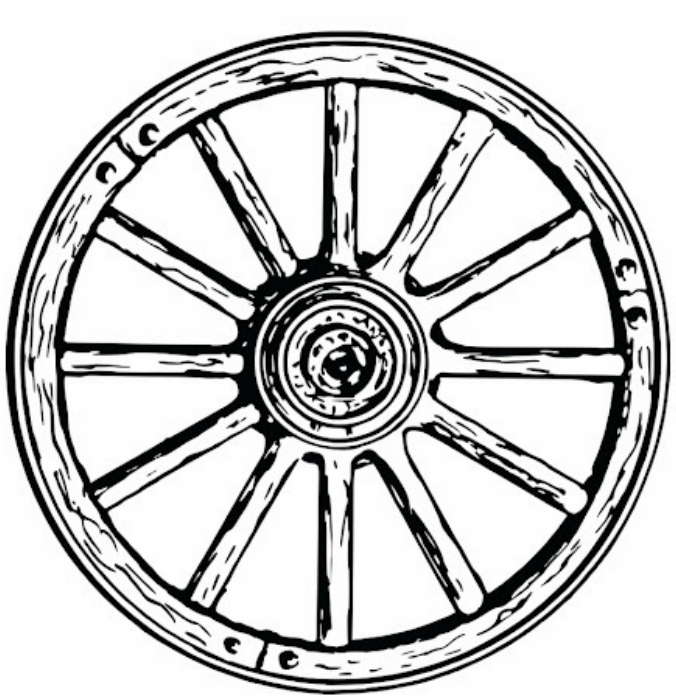


#



Hashed ?? ??

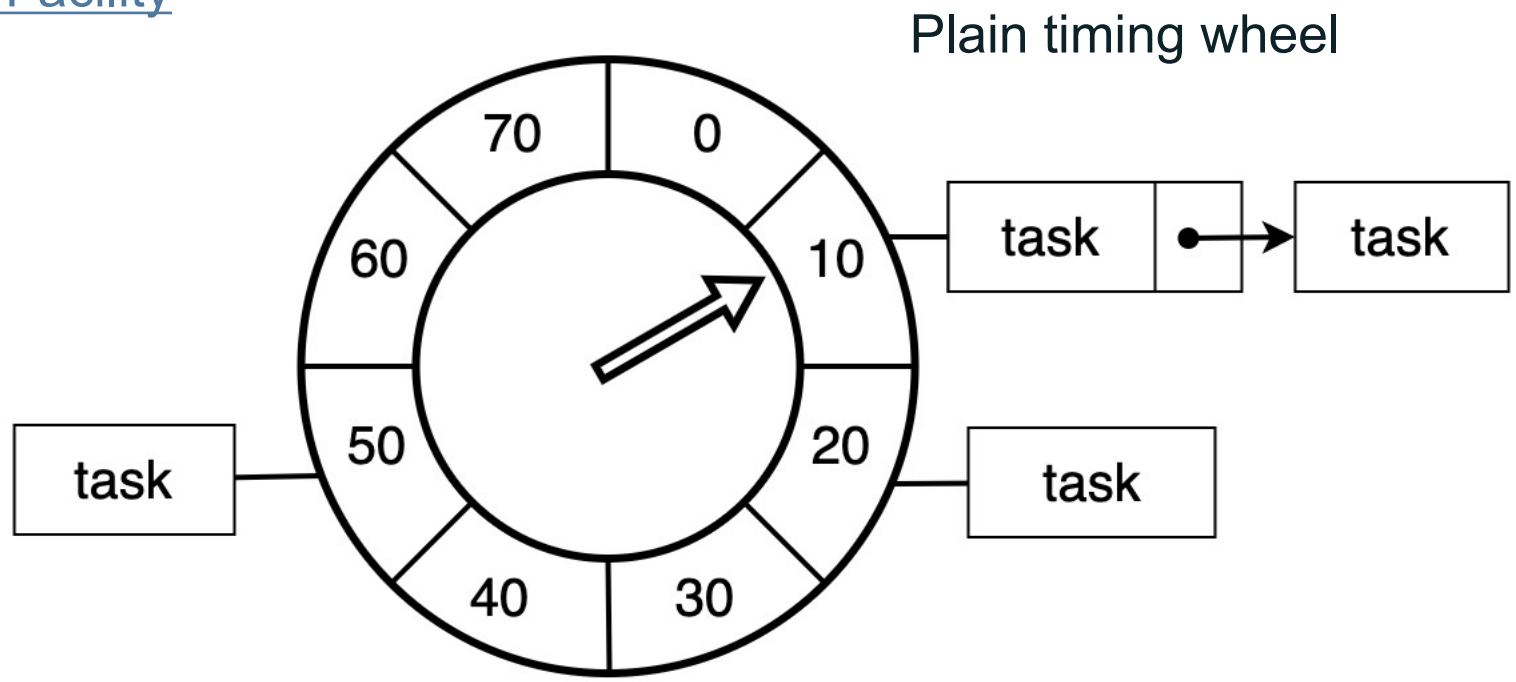
#



Hashed timing wheel

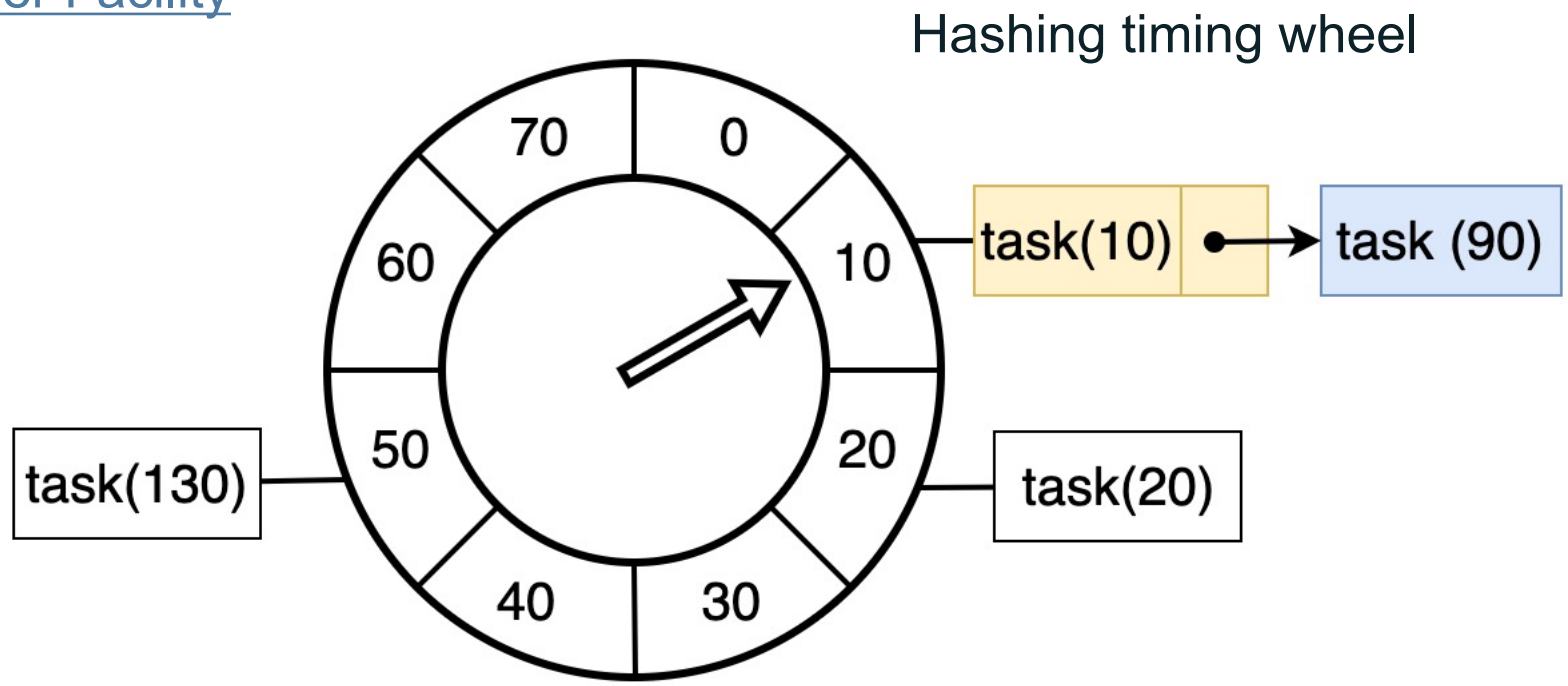
Таймеры - Hashed timing wheel

- 1987, Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility



Таймеры - Hashed timing wheel

- 1987, Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility



Таймеры - Hashed timing wheel

- 1987, Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility

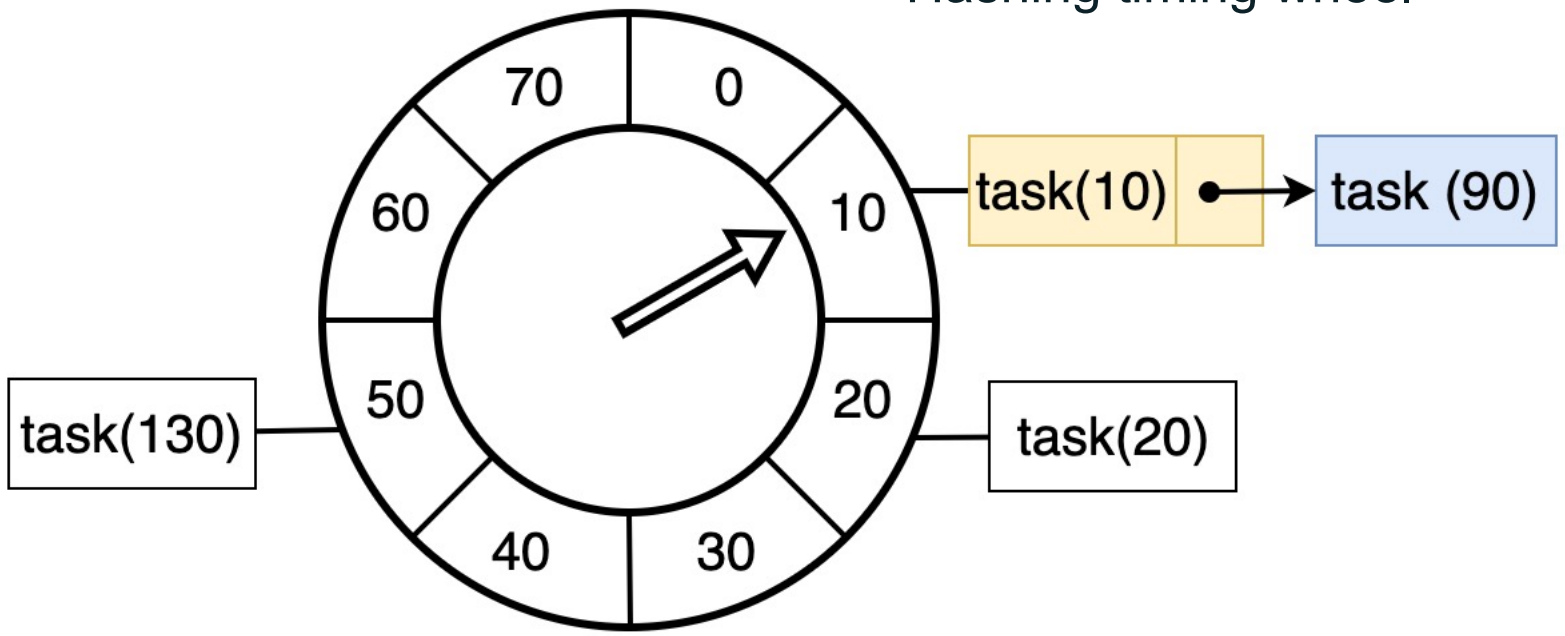
Стоимость операций:

Add(key): O(1)

Remove(key): O(1)

Trigger: O(1)

Hashing timing wheel



Таймеры - Hashed timing wheel

- 1987, Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility

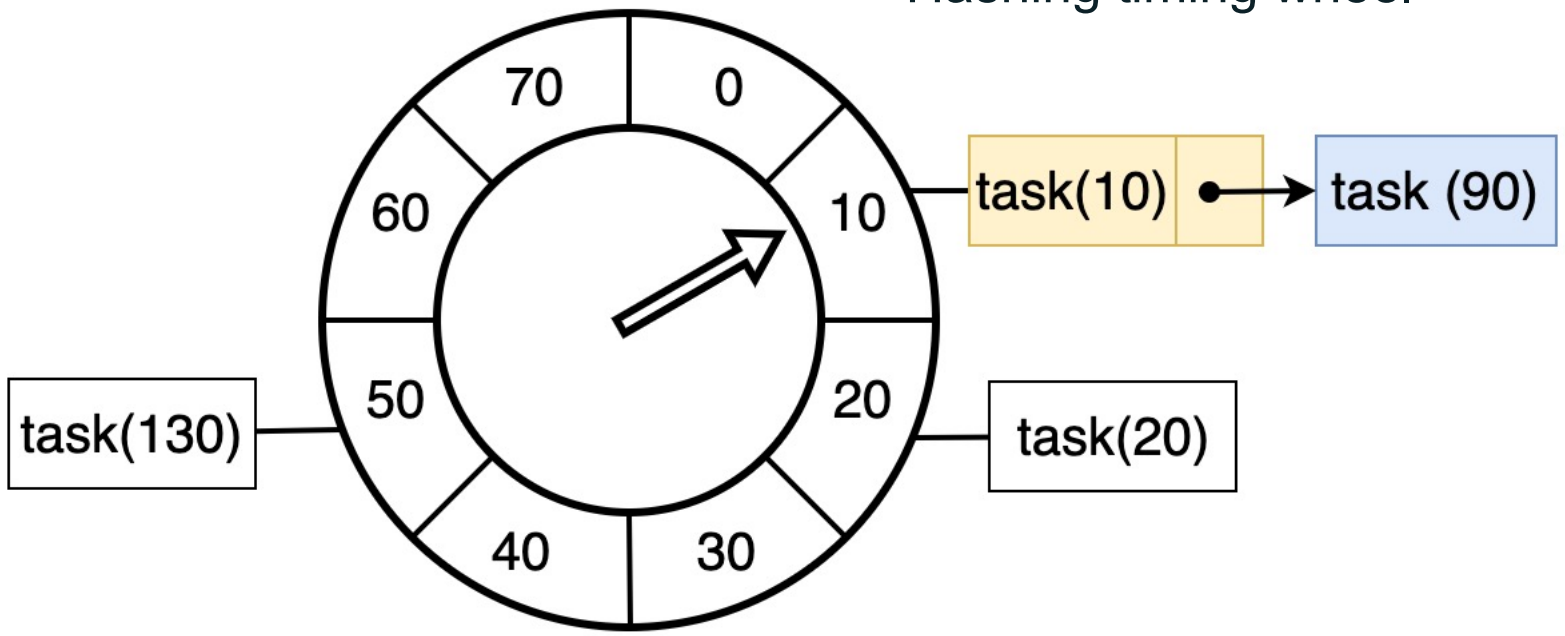
Стоимость операций:

Add(key): O(1)

Remove(key): O(1)

Trigger: O(1)

Hashing timing wheel

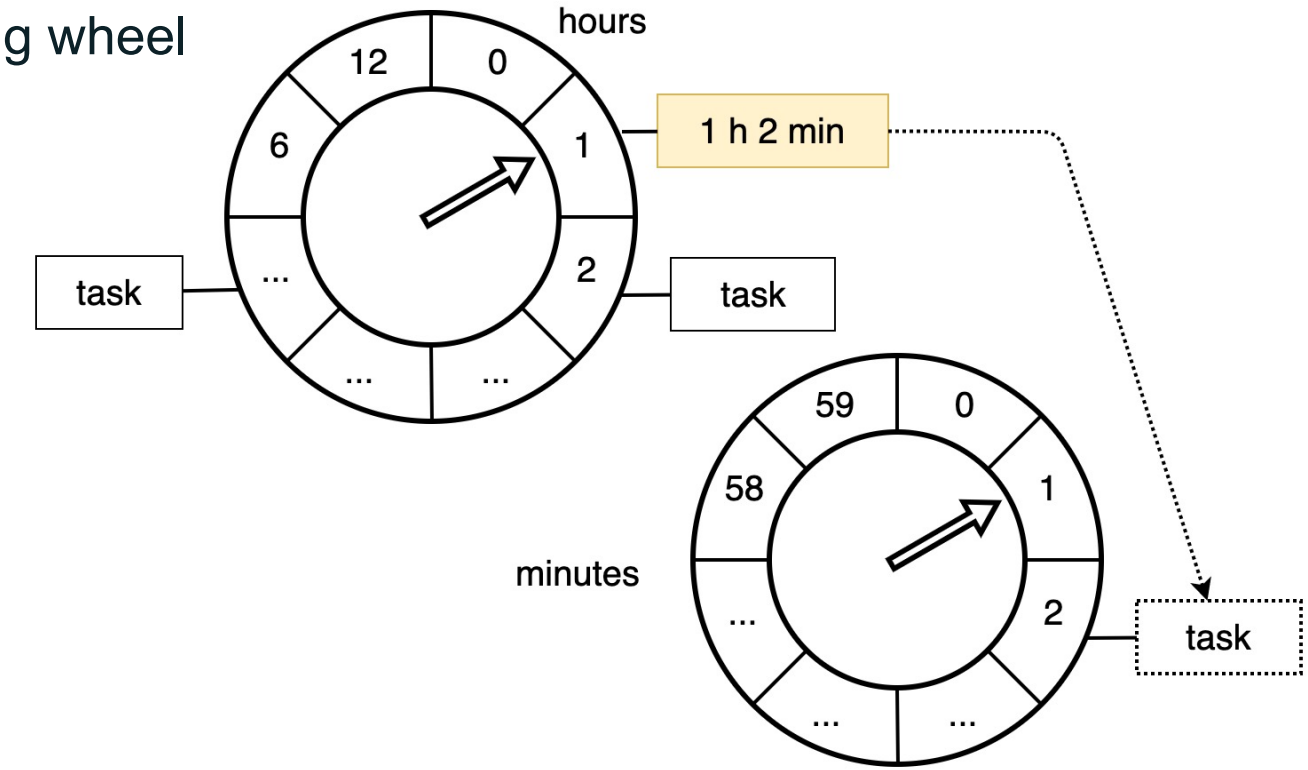


Используется в:

- Netty: io.netty.util.HashedWheelTimer
- Kafka: https://www.confluent.io/blog/apache-kafka-purgatory-hierarchical-timing-wheels/
- Linux kernel: https://lwn.net/Articles/646950/
- Caffeine cache library: com.github.benmanes.caffeine.cache.TimerWheel

Таймеры - Hashed timing wheel

Hierarchical timing wheel



Используется в:

- Netty: io.netty.util.HashedWheelTimer
- Kafka: https://www.confluent.io/blog/apache-kafka-purgatory-hierarchical-timing-wheels/
- Linux kernel: https://lwn.net/Articles/646950/
- Caffeine cache library: com.github.benmanes.caffeine.cache.TimerWheel

- Приоритизация операций по обработке запросов и фоновых задач

Rate limiting

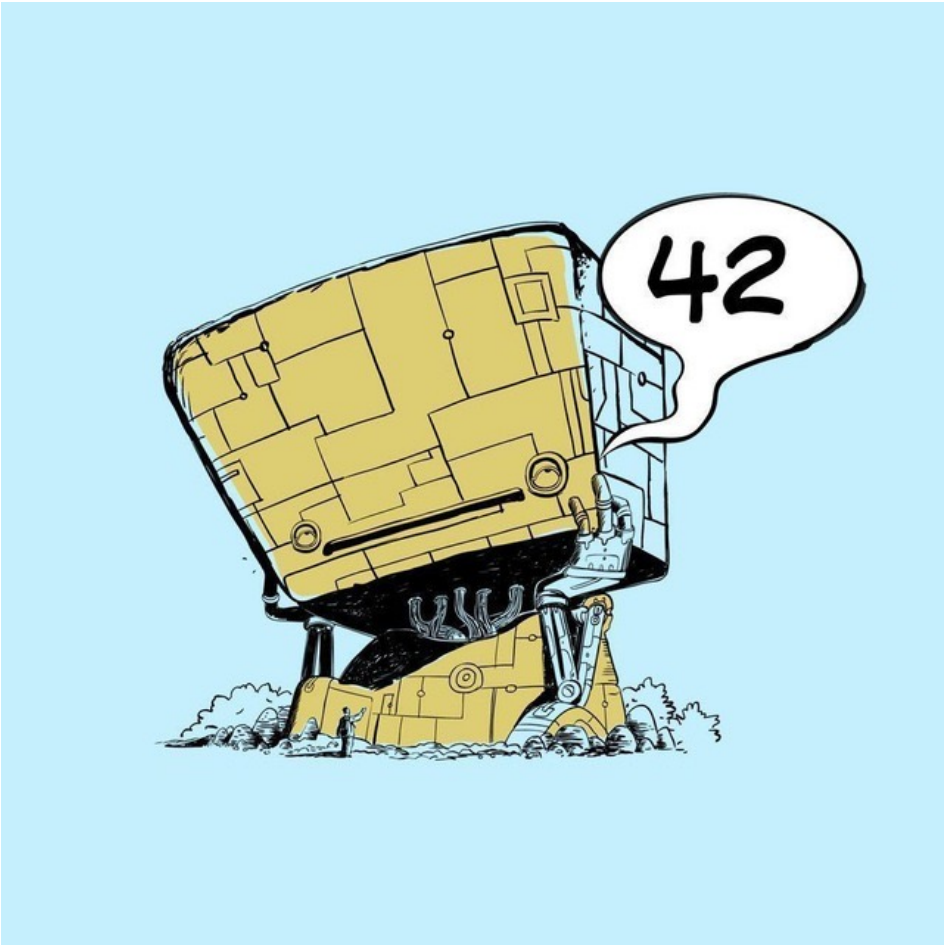
- Cassandra использует Guava RateLimiter для ограничения фоновых задач:
- Compaction (*compaction_throughput_mb_per_sec*)
- Hints dispatcher (*hinted_handoff_throttle_in_kb*)
- Batchlog recovery **(***batchlog_replay_throttle_in_kb*)
- Streaming (*stream_throughput_outbound_megabits_per_sec, inter_dc_stream_throughput_outbound_megabits_per_sec*)
- Еще варианты: bucket4j

Rate limiting

```
RateLimiter compactionRateLimiter = RateLimiter.create(Double.MAX_VALUE);

void init() {
    double throughput = getCompactionThroughputMbPerSec() * 1024.0 * 1024.0;
    compactionRateLimiter.setRate(throughput);
}

void process() {
    while(hasNext()) {
        int lengthRead = readAndProcessChunk();
        compactionRateLimiter.acquire(lengthRead);
    }
}
```



-XX:ThreadPriorityPolicy=42

- Cassandra sets MIN priority for compaction, hints executor and index summary manager threads
- XX:ThreadPriorityPolicy=0|1, 0 – default normal mode, 1 – aggressive mode

-

-XX:ThreadPriorityPolicy=42

- Cassandra sets MIN priority for compaction, hints executor and index summary manager threads
- XX:ThreadPriorityPolicy=0|1, 0 – default normal mode, 1 – aggressive mode
- XX:ThreadPriorityPolicy=1 - JDK 8 requires root: “Java HotSpot(TM) 64-Bit Server VM warning: -XX:ThreadPriorityPolicy requires root privilege on Linux”
- http://tech.stolsvik.com/2010/01/linux-java-thread-priorities-workaround.html

-XX:ThreadPriorityPolicy=42

- Cassandra sets MIN priority for compaction, hints executor and index summary manager threads
- XX:ThreadPriorityPolicy=0|1, 0 – default normal mode, 1 – aggressive mode
- XX:ThreadPriorityPolicy=1 - JDK 8 requires root: “Java HotSpot(TM) 64-Bit Server VM warning: -XX:ThreadPriorityPolicy requires root privilege on Linux”
- http://tech.stolsvik.com/2010/01/linux-java-thread-priorities-workaround.html
- Not working since java 9 ☹ - JEP 245: Validate JVM Command-Line Flag Arguments
- CASSANDRA-13107 - Remove -XX:ThreadPriorityPolicy=42 jvm flag (Java 11)

-XX:ThreadPriorityPolicy=42

- Cassandra sets MIN priority for compaction, hints executor and index summary manager threads
- XX:ThreadPriorityPolicy=0|1, 0 – default normal mode, 1 – aggressive mode
- XX:ThreadPriorityPolicy=1 - JDK 8 requires root: “Java HotSpot(TM) 64-Bit Server VM warning: -XX:ThreadPriorityPolicy requires root privilege on Linux”
- http://tech.stolsvik.com/2010/01/linux-java-thread-priorities-workaround.html
- Not working since java 9 ☹ - JEP 245: Validate JVM Command-Line Flag Arguments
- CASSANDRA-13107 - Remove -XX:ThreadPriorityPolicy=42 jvm flag (Java 11)
- JDK-8215962 - Support ThreadPriorityPolicy mode 1 for non-root users on linux/bsd (JDK 11.0.3)

-XX:ThreadPriorityPolicy=1 -> thread priorities set in Java are reflected by native operating system thread priorities

ioprio_set / ionice

- CASSANDRA-9946 - use ioprio_set on compaction threads by default instead of manually throttling), open ☹

I/O приоритеты в Linux

```
int ioprio_set(int which, int who, int ioprio);
```

↑

системный thread id

а у нас только java.lang.Thread ☹

42

Jpoint 2018, Андрей Паньгин, VMStructs: зачем приложению знать о внутренностях JVM, слайды: 48 - 58

Affinity

- https://issues.apache.org/jira/browse/CASSANDRA-1632 (эксперименты дали противоречивые результаты, дальше дело не пошло)

Affinity

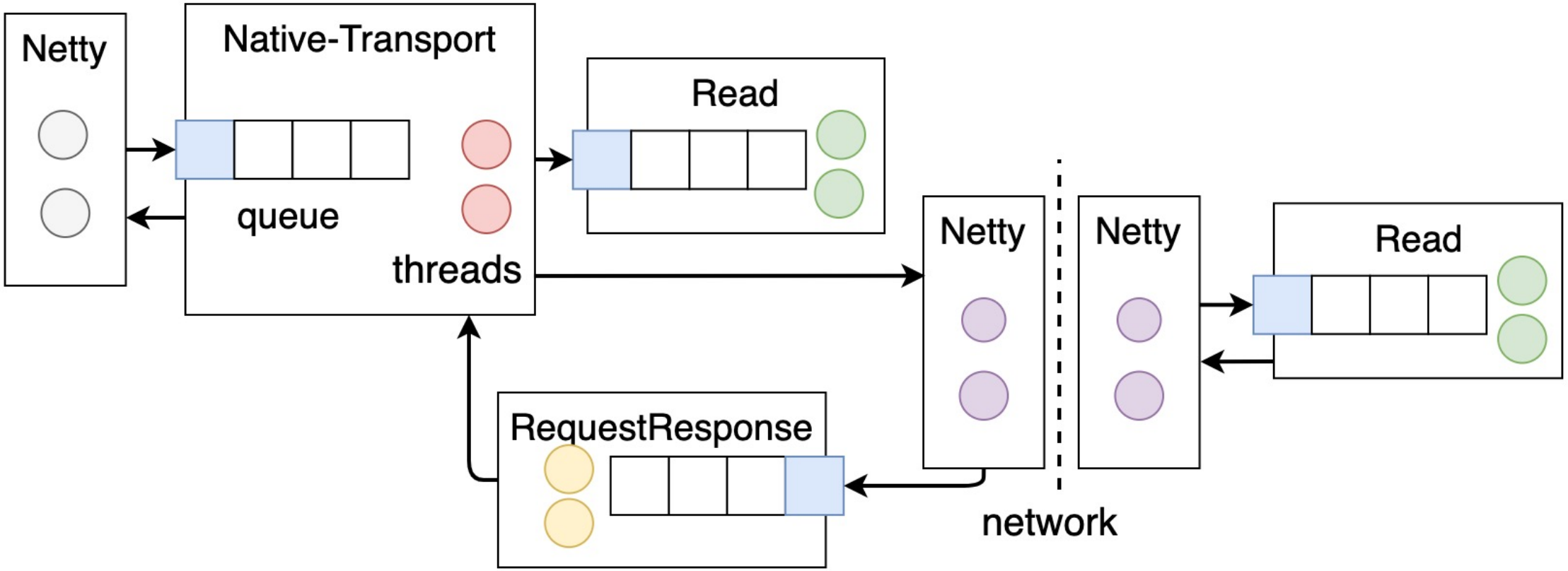
- https://issues.apache.org/jira/browse/CASSANDRA-1632 (эксперименты дали противоречивые результаты, дальше дело не пошло)
- https://github.com/OpenHFT/Java-Thread-Affinity

Peter Lawrey

- Накладные расходы при работе с потоками

Cassandra threading issues

- SEDA-like
- Too many threads
- Context switches
- Park/unpark cost

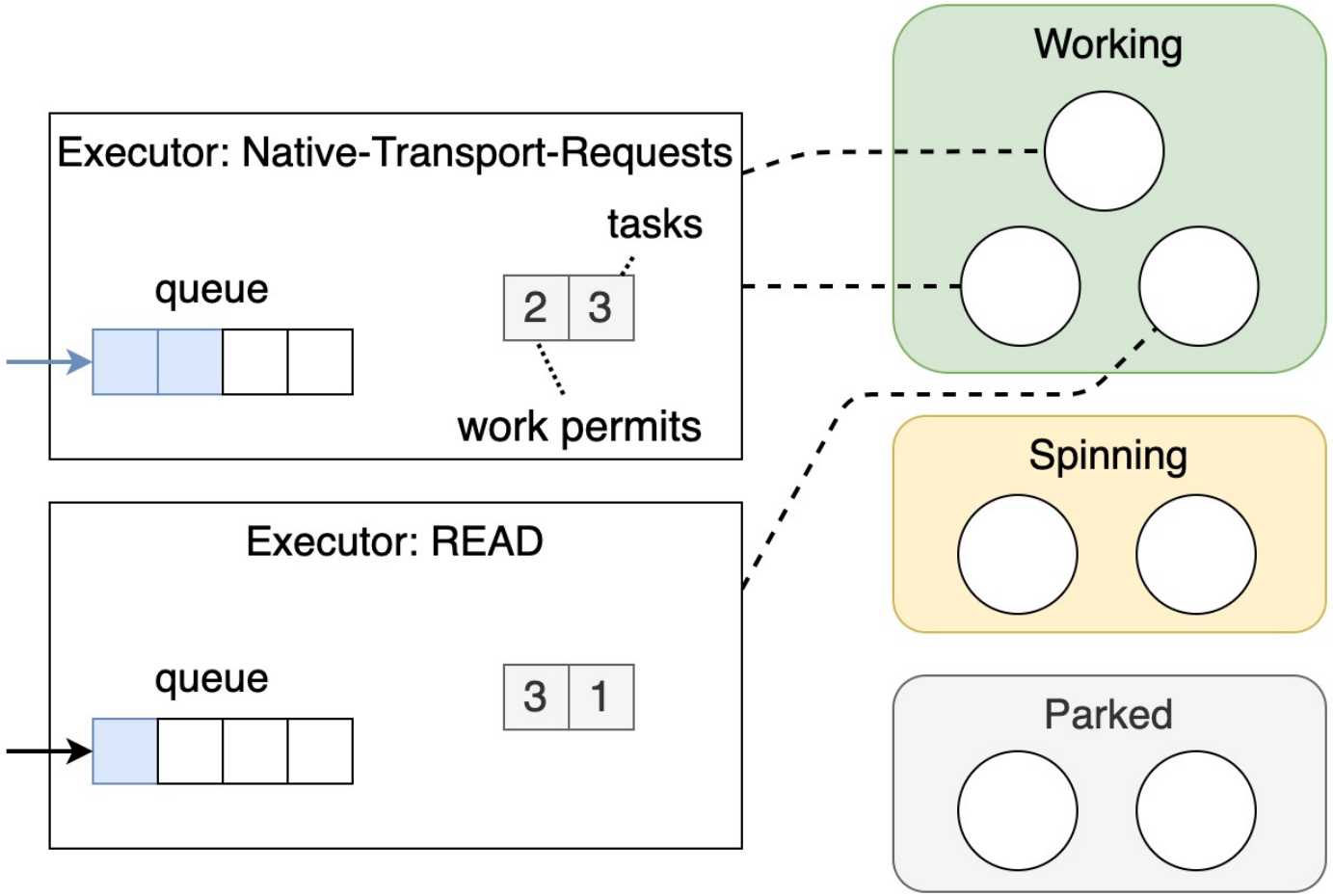


Cassandra threading issues

- CASSANDRA-4718 – “More-efficient ExecutorService for improved throughput”
- Discussed, checked, rejected:
  - Bulk take from queue
  - Disruptor
  - Jetty thread pool
  - ForkJoinPool
  - Akka

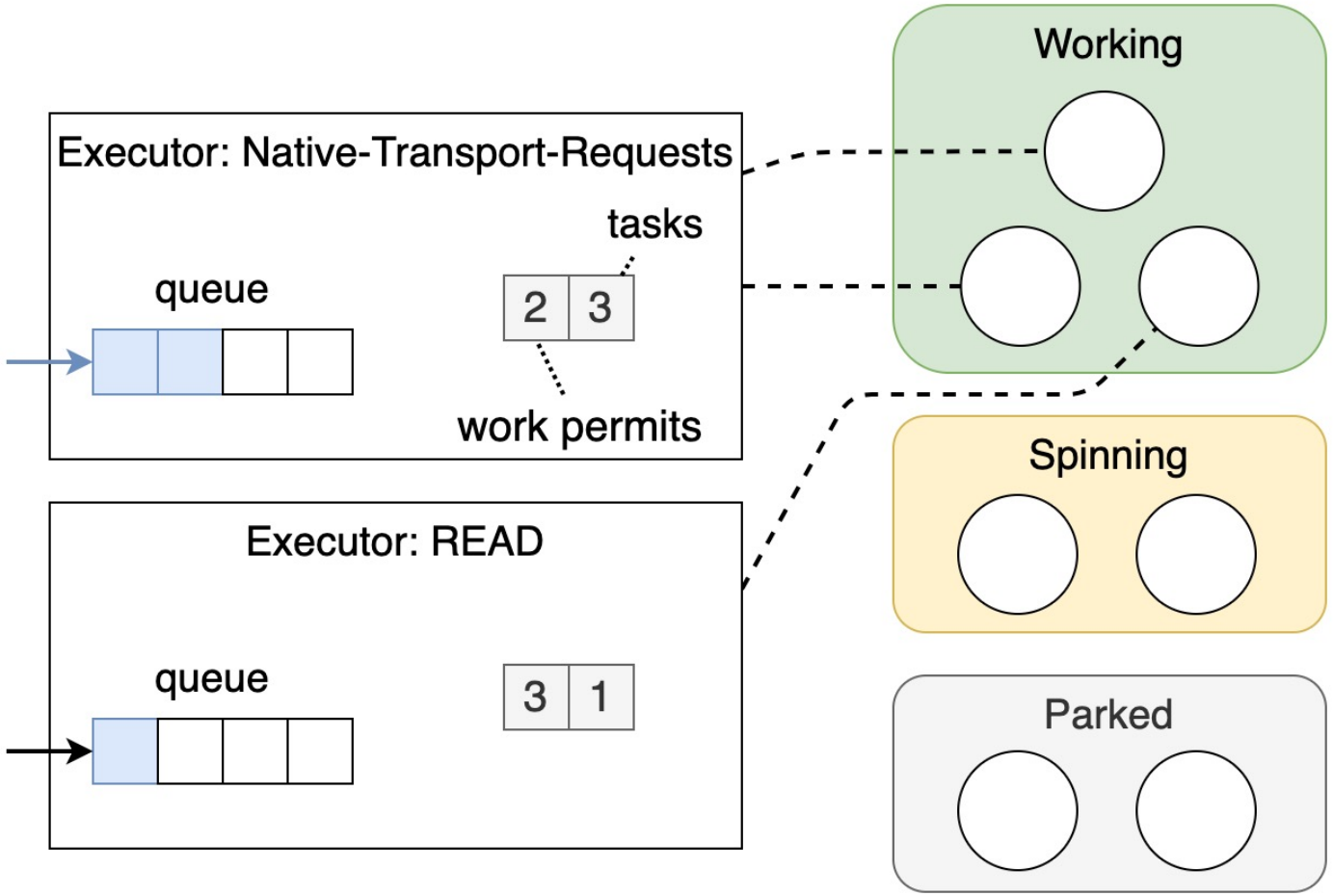
Shared pool executor

- SEDA-like
- Sharing threads between executors



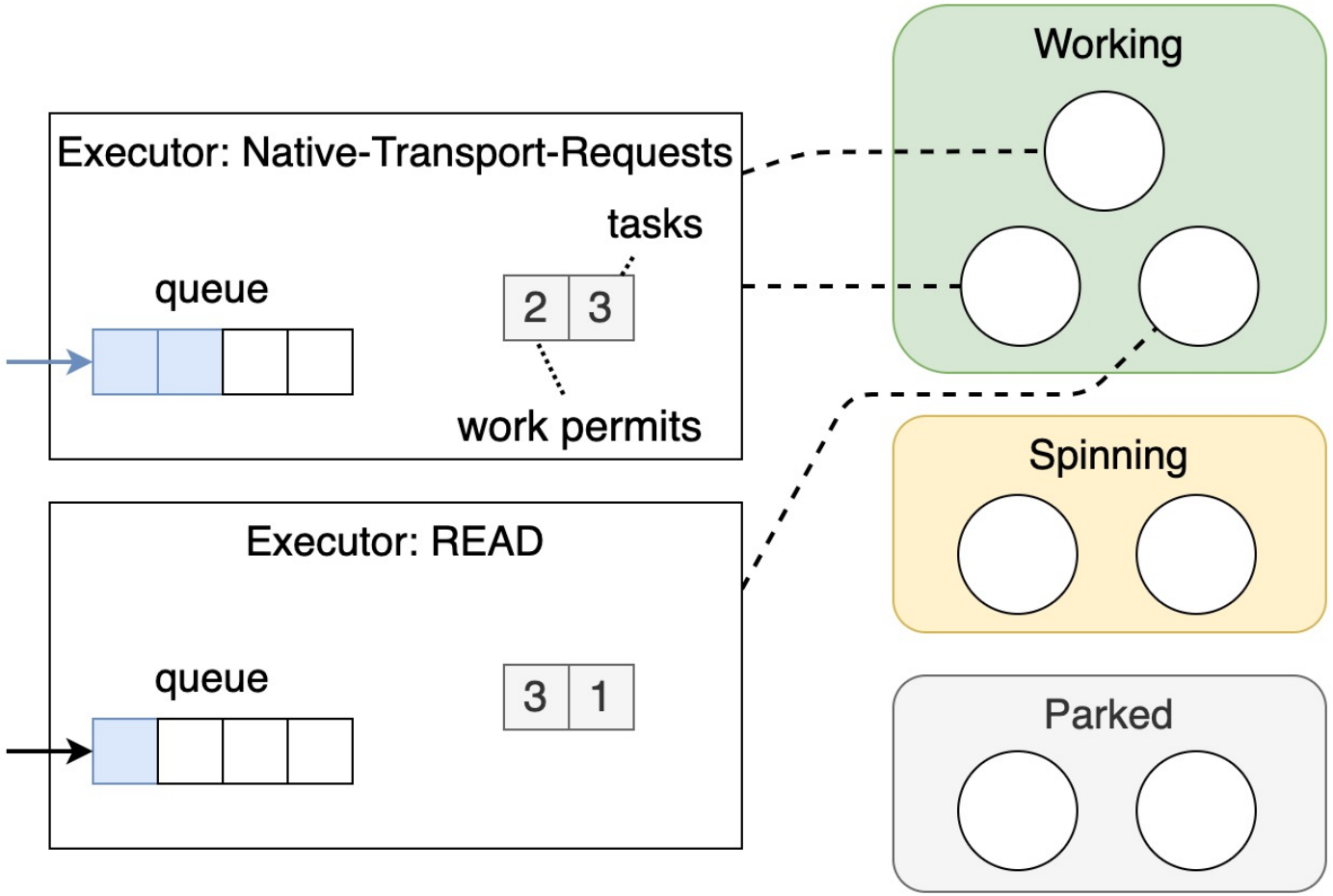
Shared pool executor

- SEDA-like
- Sharing threads between executors
- Avoid parking/unpark cost
- Limit working threads per stage using CAS



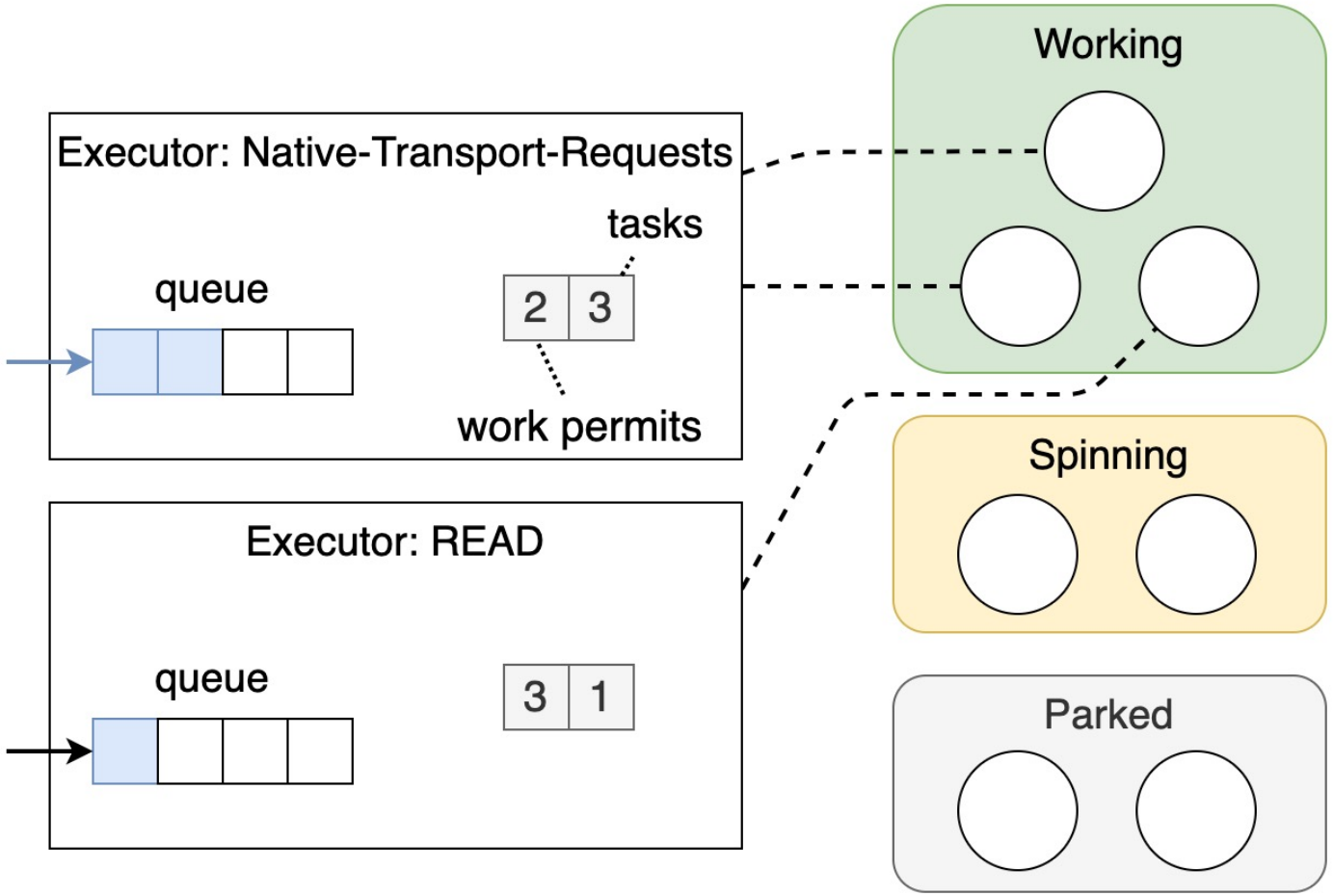
Shared pool executor

- SEDA-like
- Sharing threads between executors
- Avoid parking/unpark cost
- Limit working threads per stage using CAS
- Spinning time depends on number of spinning threads
- Task can be scheduled within the current thread (under concurrent limits)



Shared pool executor

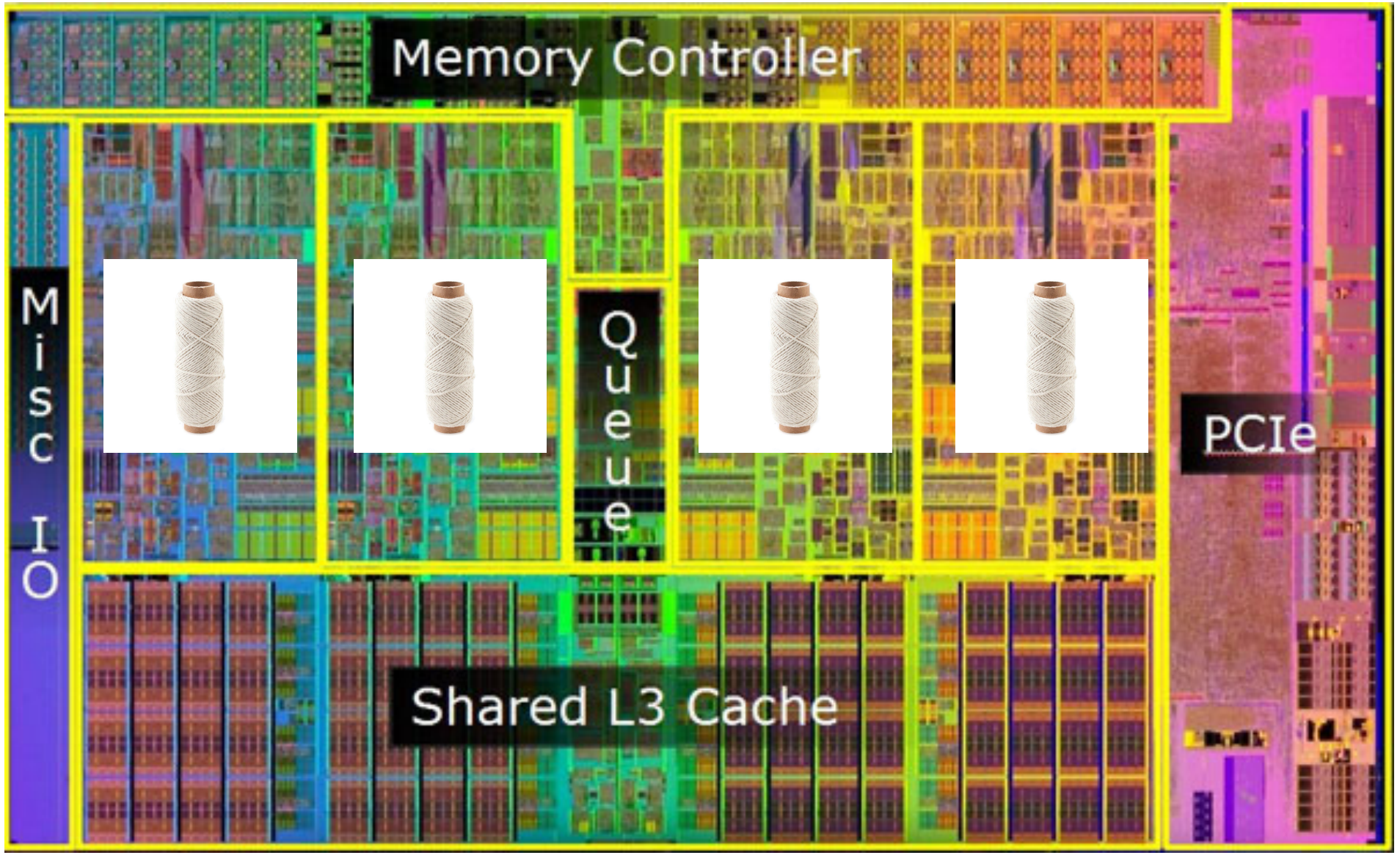
- SEDA-like
- Sharing threads between executors
- Avoid parking/unpark cost
- Limit working threads per stage using CAS
- Spinning time depends on number of spinning threads
- Task can be scheduled within the current thread (under concurrent limits)

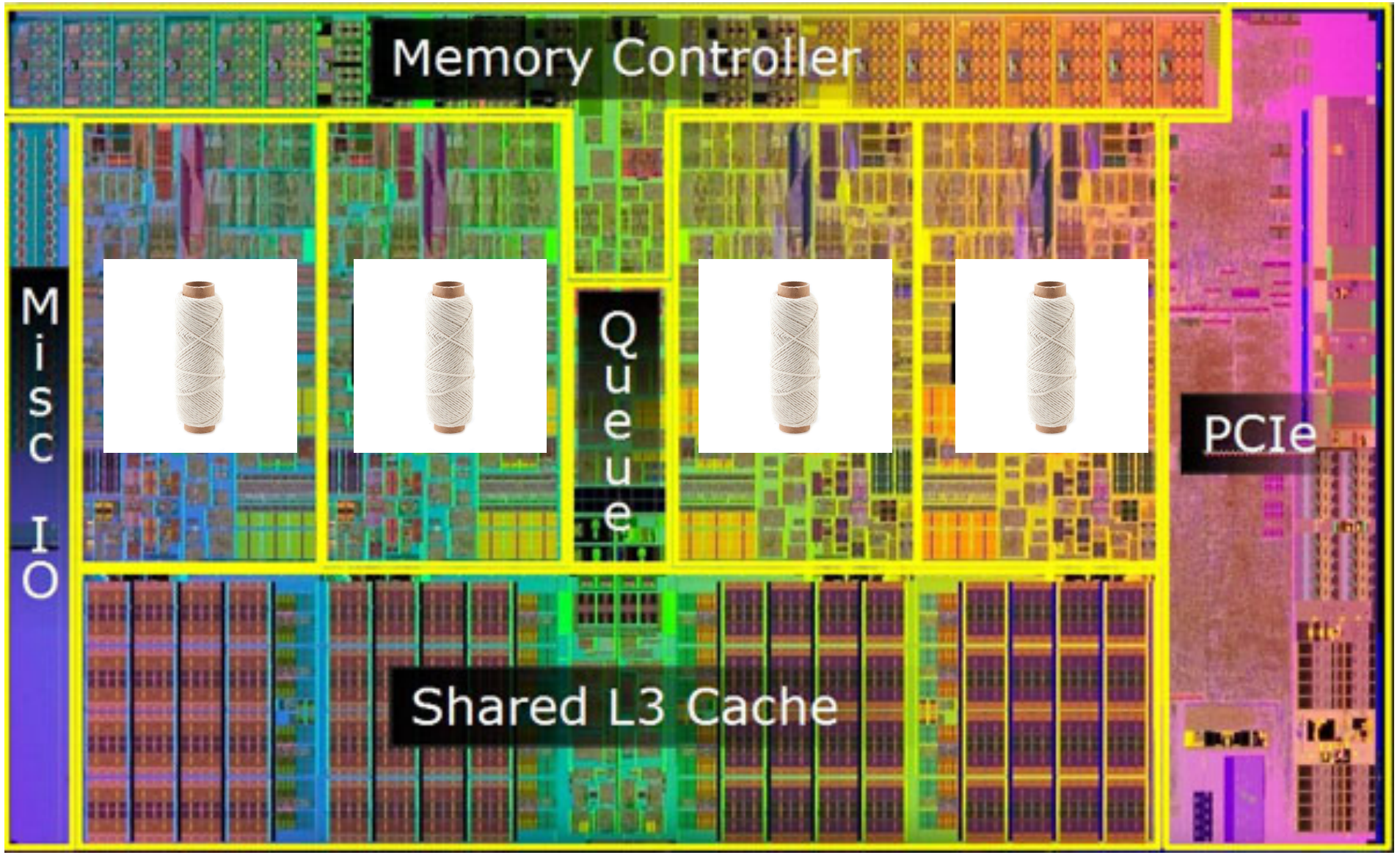


See: org.apache.cassandra.concurrent.SEPExecutor

Shared pool executor - issues

- CASSANDRA-16499 (single-threaded write workloads can spend ~70% time waiting on SEPExecutor)
- CASSANDRA-15022 (SEPExecutor local requests not triggered often)
- CASSANDRA-16186 (no limit for queue)





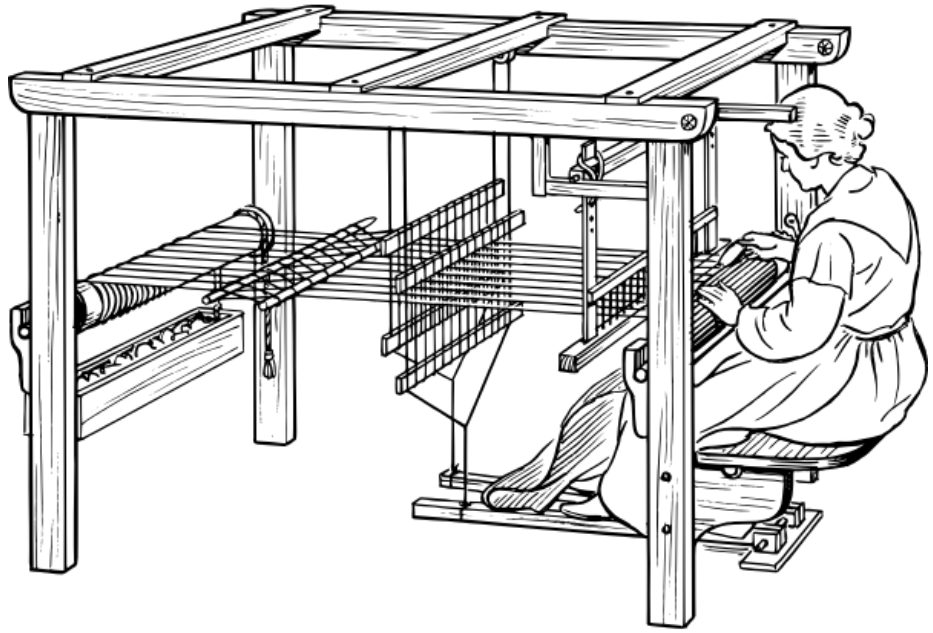
Thread per core

Thread per core (TPC)

- CASSANDRA-10989 / presentation with an analysis - Move away from SEDA to TPC, open ☹
- CASSANDRA-10528 - Proposal: Integrate RxJava
- DSE реализовал TPC (link1, link2), без шардирования данных по потокам
- Полностью реализовано в Scylla

Thread per core (TPC)

- CASSANDRA-10989 / presentation with an analysis - Move away from SEDA to TPC, open ☹
- CASSANDRA-10528 - Proposal: Integrate RxJava
- DSE реализовал TPC (link1, link2), без шардирования данных по потокам
- Полностью реализовано в Scylla
- А может Loom? (если дождемся ☺)

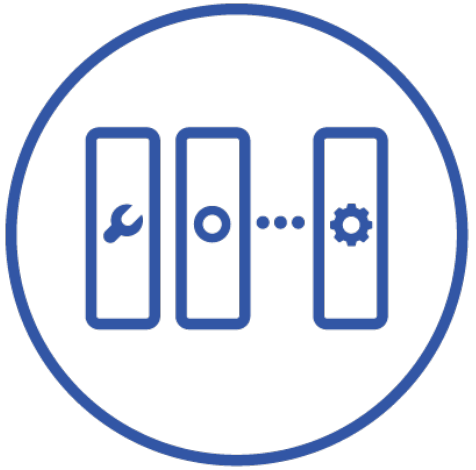


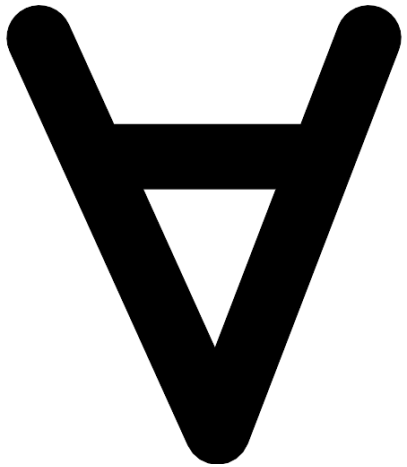
A Disruptor? Был…

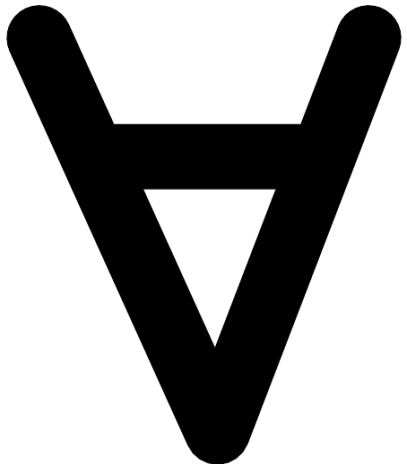
- CASSANDRA-5582 - Replace CustomHsHaServer with better optimized solution based on LMAX Disruptor (Thrift only, Cassandra 2.x)

A Disruptor? Был…

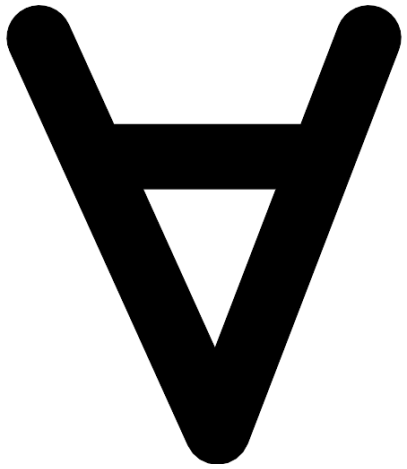
- CASSANDRA-5582 - Replace CustomHsHaServer with better optimized solution based on LMAX Disruptor (Thrift only, Cassandra 2.x)
- Other things are taken from High Frequency Trading (HFT) domain - Chronicle queue:
  - CASSANDRA-12151 - full query logging, docs
  - CASSANDRA-12151 - audit logs, docs
  - CASSANDRA-13460 - planned to use for diagnostic events, open







Thread + lock + ??



Thread + lock + all (math)
Thread local

ThreadLocal – для чего?

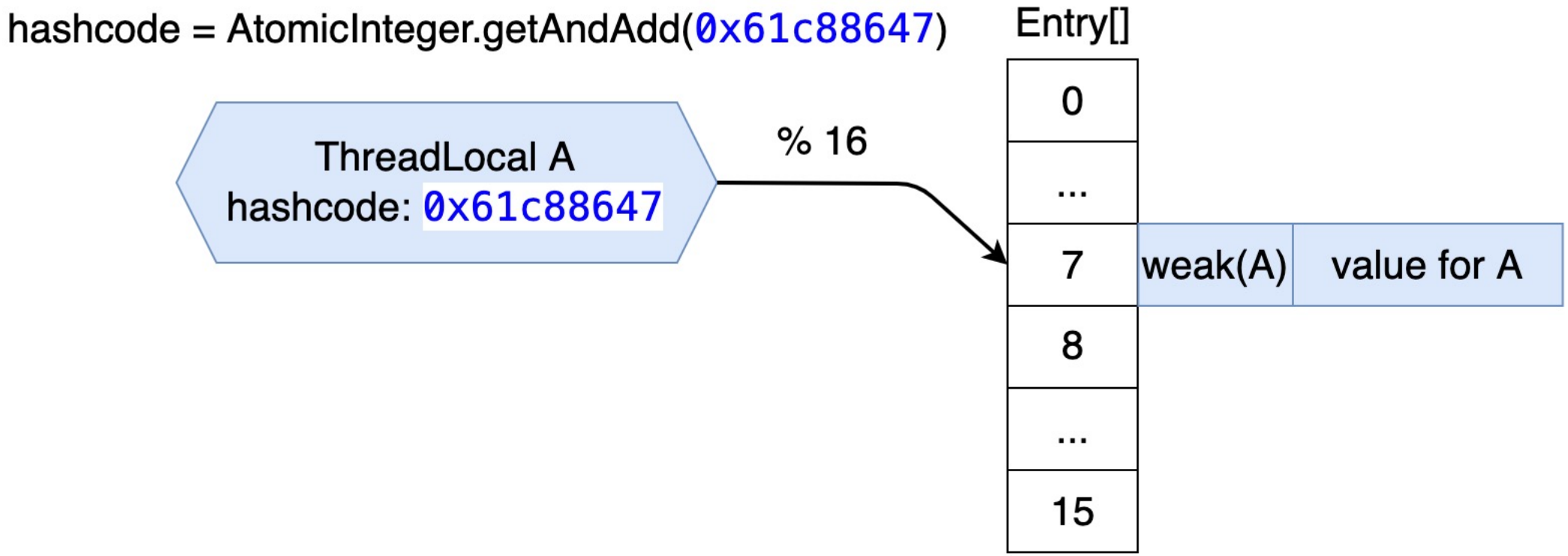
- Локальные буферы/пулы
- Контекст трассировки

JDK ThreadLocal

- Custom hash map as a field on Thread
- ThreadLocal is a key

JDK ThreadLocal

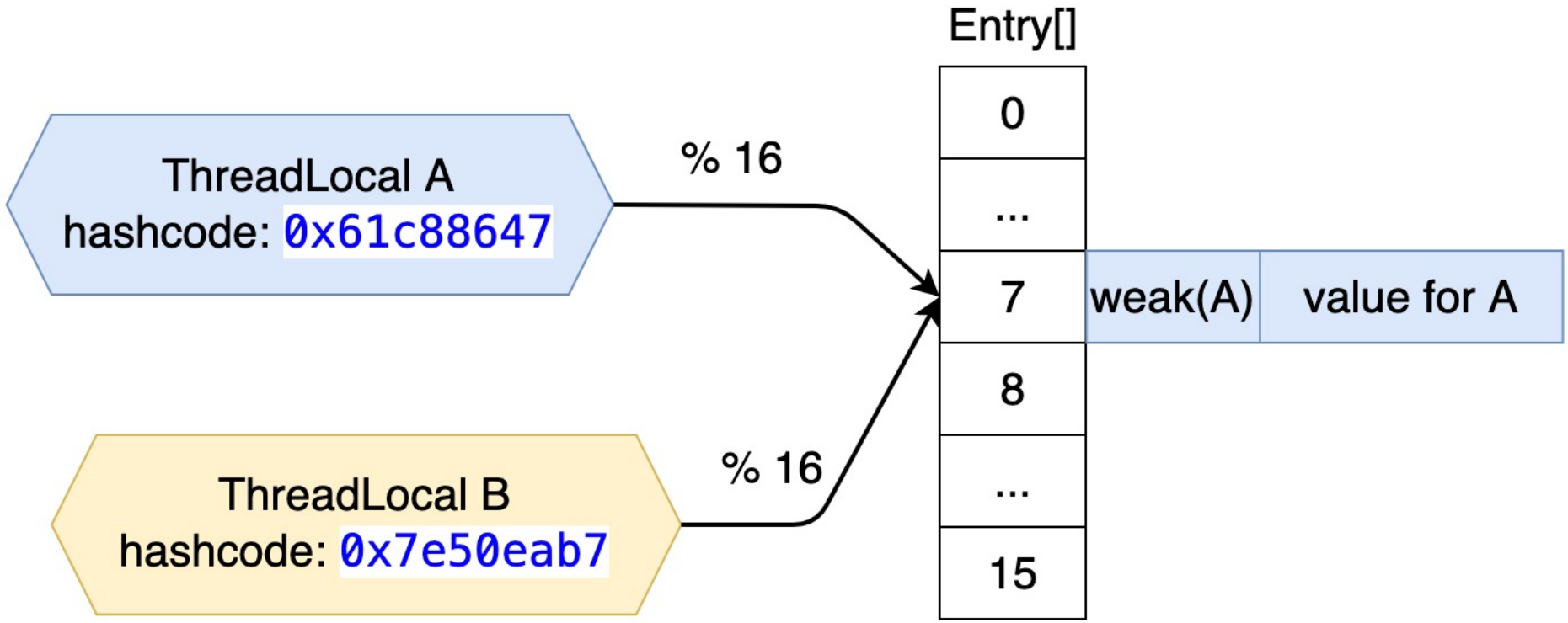
- Custom hash map as a field on Thread
- ThreadLocal is a key
- Open addressing



JDK ThreadLocal

- Custom hash map as a field on Thread
- ThreadLocal is a key
- Open addressing

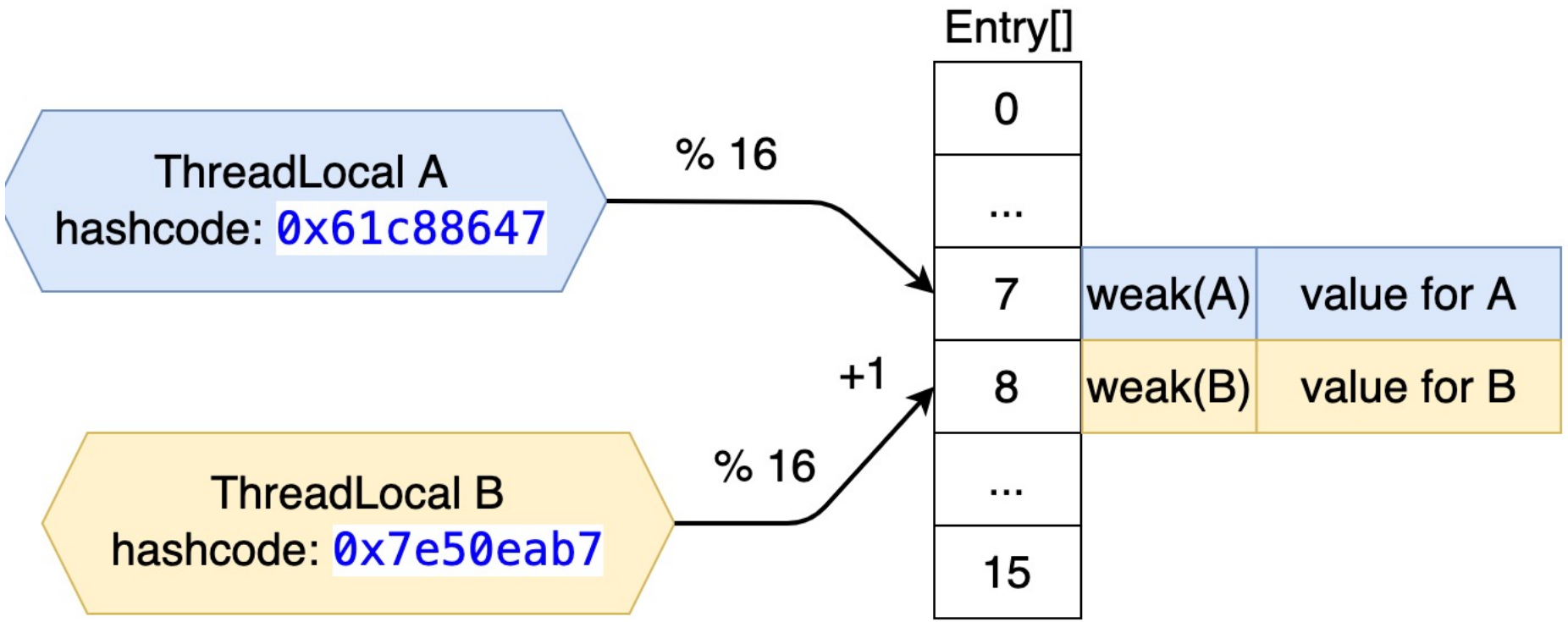
- Cached hashcode with avoiding of collisions for subsequent elements



JDK ThreadLocal

- Custom hash map as a field on Thread
- ThreadLocal is a key
- Open addressing

- Cached hashcode with avoiding of collisions for subsequent elements
- Line probing

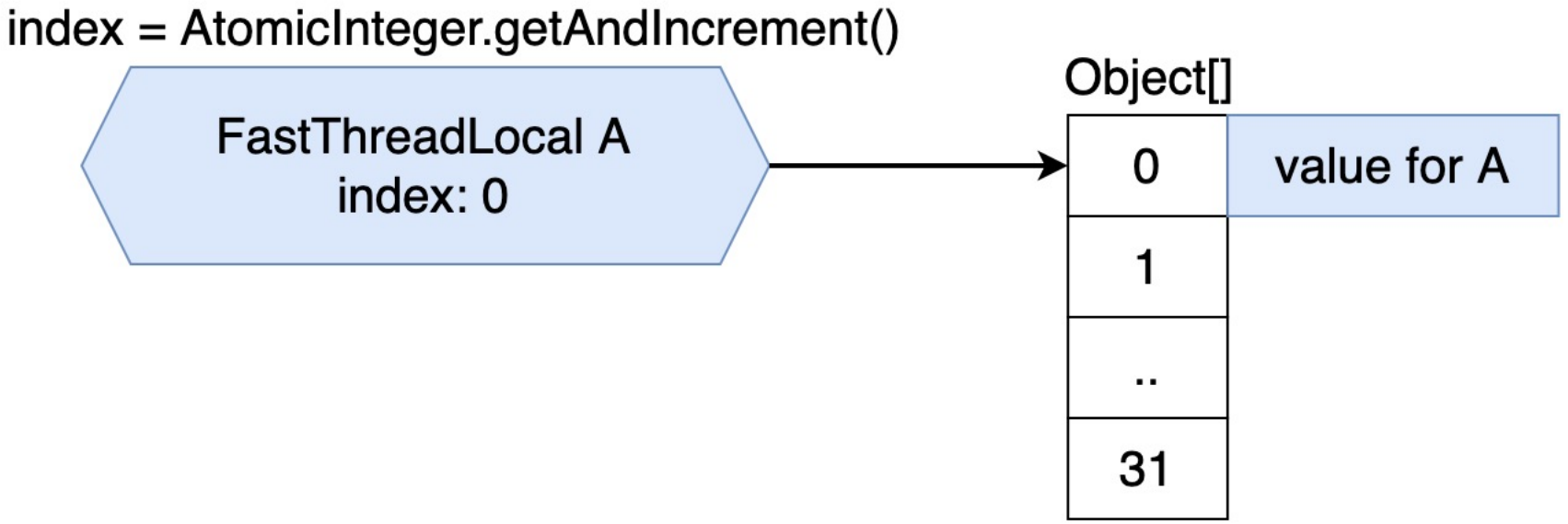


Netty ThreadLocal

- io.netty.util.concurrent.FastThreadLocal –custom thread with extra field
- Requires: io.netty.util.concurrent.FastThreadLocalThread
- Array with globally allocated index per thread local object
- getIfExists method is supported

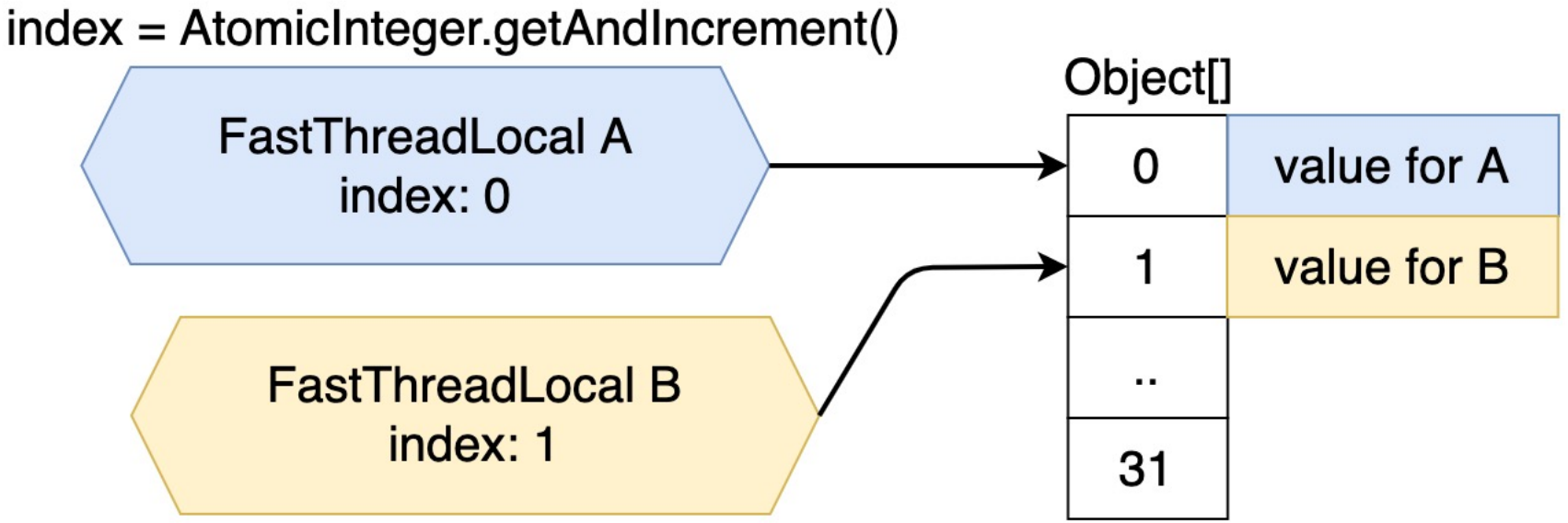
Netty ThreadLocal

- io.netty.util.concurrent.FastThreadLocal –custom thread with extra field
- Requires: io.netty.util.concurrent.FastThreadLocalThread
- Array with globally allocated index per thread local object
- getIfExists method is supported



Netty ThreadLocal

- io.netty.util.concurrent.FastThreadLocal –custom thread with extra field
- Requires: io.netty.util.concurrent.FastThreadLocalThread
- Array with globally allocated index per thread local object
- getIfExists method is supported



Netty ThreadLocal vs JDK thread local

- Netty thread local is not universal (static limited number of thread locals is expected)
- JMH: io.netty.microbench.concurrent.FastThreadLocalFastPathBenchmark
- 16 thread locals

```
# JMH version: 1.22
# VM version: JDK 11.0.5, OpenJDK 64-Bit Server VM, 11.0.5+10-b520.38

Benchmark              Mode  Cnt       Score       Error  Units
fastThreadLocal       thrpt   20  693606.993 ±  9417.349  ops/s
jdkThreadLocalGet     thrpt   20  489071.252 ± 10726.905  ops/s
```

- Обработка OutOfMemory ошибок

OOM handling

- Base way: OnOutOfMemoryError, HeapDumpOnOutOfMemoryError flags

OOM handling

- Base way: OnOutOfMemoryError, HeapDumpOnOutOfMemoryError flags - **does not cover thread creation failure** (JDK-8155004)
- https://github.com/airlift/jvmkill
  - Uses: JvmtiExport::post_resource_exhausted to detect OOM (heap + threads + metaspace)

OOM handling

- Base way: OnOutOfMemoryError, HeapDumpOnOutOfMemoryError flags - does not cover thread creation failure (JDK-8155004)
- https://github.com/airlift/jvmkill
  - Uses: JvmtiExport::post_resource_exhausted to detect OOM (heap + threads + metaspace)
- https://github.com/Netflix-Skunkworks/jvmquake
  - Kill when a JVM is unstable but not quite dead yet (aka "GC spirals of death")
  - Uses jvmti – GarbageCollectionStart and jvmti - GarbageCollectionFinish to track GC pauses

OOM handling

- Base way: OnOutOfMemoryError, HeapDumpOnOutOfMemoryError flags - does not cover thread creation failure (JDK-8155004)
- https://github.com/airlift/jvmkill
  - Uses: JvmtiExport::post_resource_exhausted to detect OOM (heap + threads + metaspace)
- https://github.com/Netflix-Skunkworks/jvmquake
  - Kill when a JVM is unstable but not quite dead yet (aka "GC spirals of death")
  - Uses jvmti – GarbageCollectionStart and jvmti - GarbageCollectionFinish to track GC pauses
- Direct buffer OOM cannot trigger JVM OOM error related options, e.g. OnOutOfMemoryError, HeapDumpOnOutOfMemoryError, etc. (see also: JDK-8257790 - ExitOnOutOfMemoryError doesn't cover OOMEs from Java libraries)
- *org.apache.cassandra.utils.JVMStabilityInspector#forceHeapSpaceOomMaybe*
  - CASSANDRA-15214 – analyze stack traces for OOM thrown from *java.nio.Bits.reserveMemory*
  - Intentionally produce a heap space OOM upon seeing a Direct buffer memory OOM

- Измерение потребления памяти

Measure memory usage

- Nodetool tablestats
- Jamm library
  - Java agent
  - Analyze object graph using reflection API (with skipping required parts)
  - Uses **java.lang.instrument.Instrumentation#getObjectSize**
  - Supports @Unmetered

Measure memory usage

- Nodetool tablestats
- Jamm library
  - Java agent
  - Analyze object graph using reflection API (with skipping required parts)
  - Uses **java.lang.instrument.Instrumentation#getObjectSize**
  - Supports @Unmetered
  - Замеряем размер структуры данных для строки-примера и используем как основу для реальных данных

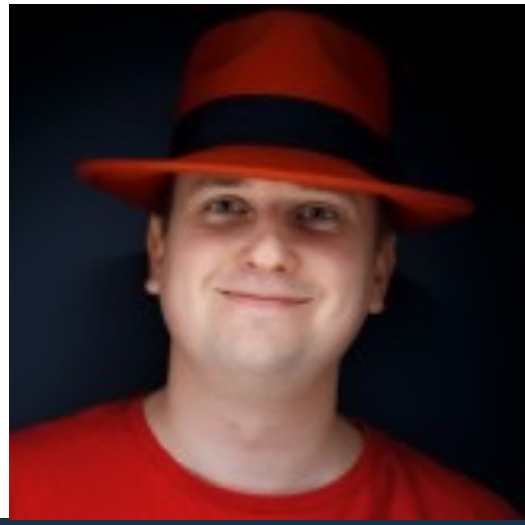
Measure memory usage

- Nodetool tablestats
- Jamm library
  - Java agent
  - Analyze object graph using reflection API (with skipping required parts)
  - Uses **java.lang.instrument.Instrumentation#getObjectSize**
  - Supports @Unmetered
  - Замеряем размер структуры данных для строки-примера и используем как основу для реальных данных
- jol
  - https://shipilev.net/jvm/objects-inside-out/
  - https://github.com/openjdk/jol
- https://openjdk.java.net/jeps/8249196 - JEP draft: Low-level Object layout introspection methods

Measure memory usage

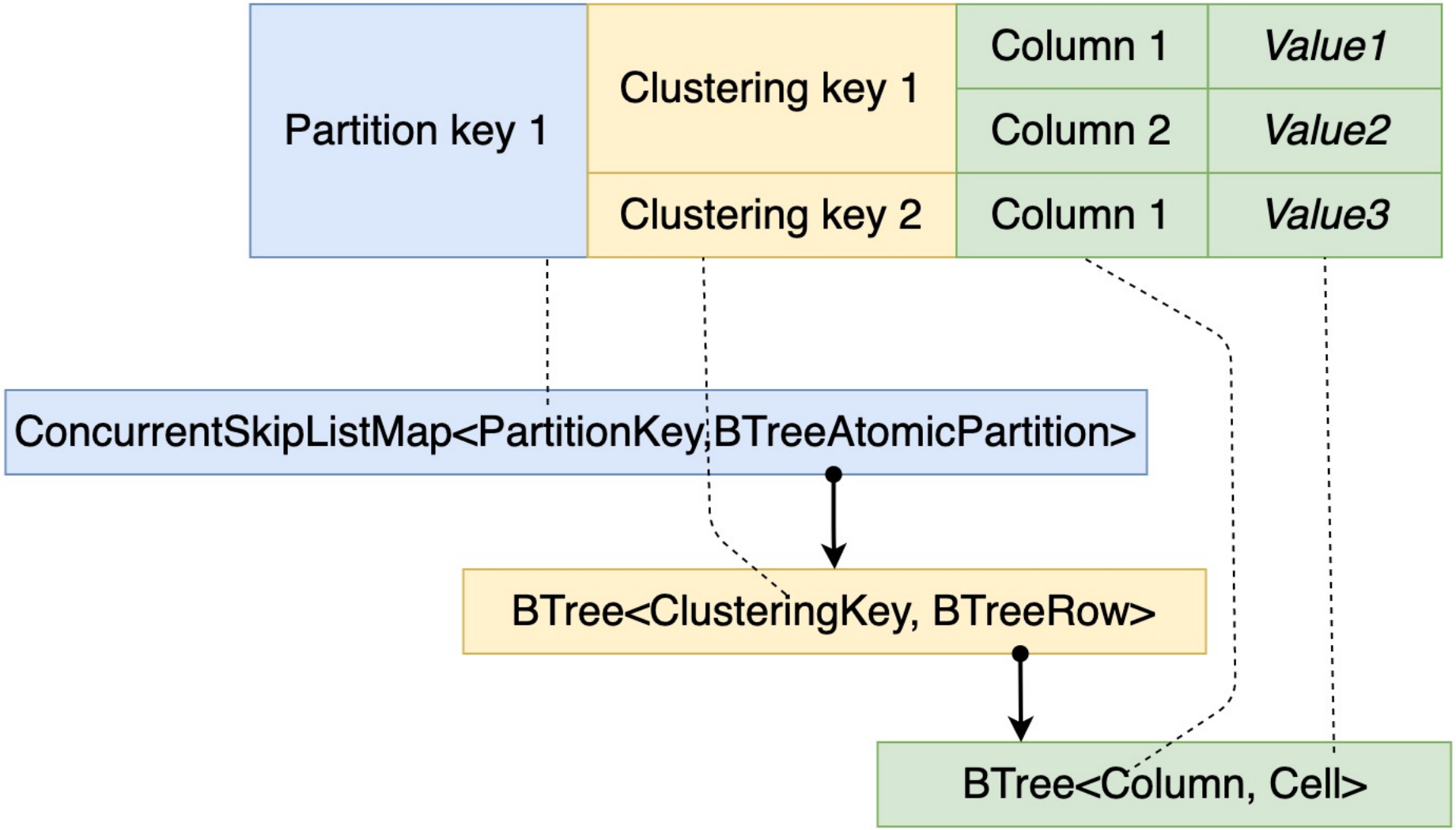
- Nodetool tablestats
- Jamm library
  - Java agent
  - Analyze object graph using reflection API (with skipping required parts)
  - Uses **java.lang.instrument.Instrumentation#getObjectSize**
  - Supports @Unmetered
  - Замеряем размер структуры данных для строки-примера и используем как основу для реальных данных
- jol
  - https://shipilev.net/jvm/objects-inside-out/
  - https://github.com/openjdk/jol
- https://openjdk.java.net/jeps/8249196

Joker 2021: Java-объекты наизнанку (сегодня, в 20:00 MSK)

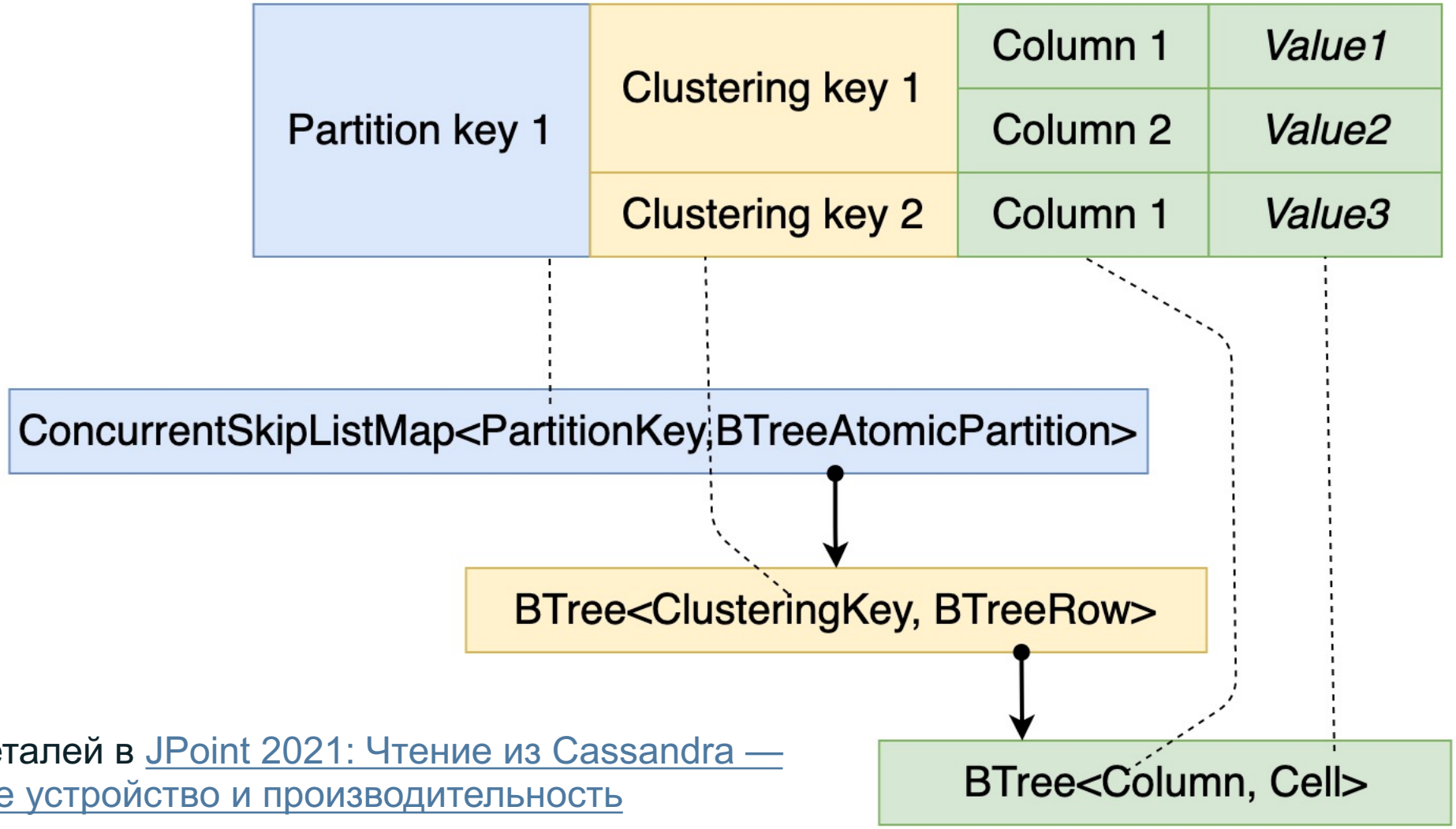


- Memory footprint

Что в Cassandra хипе? - Memtables



Что в Cassandra хипе? - Memtables



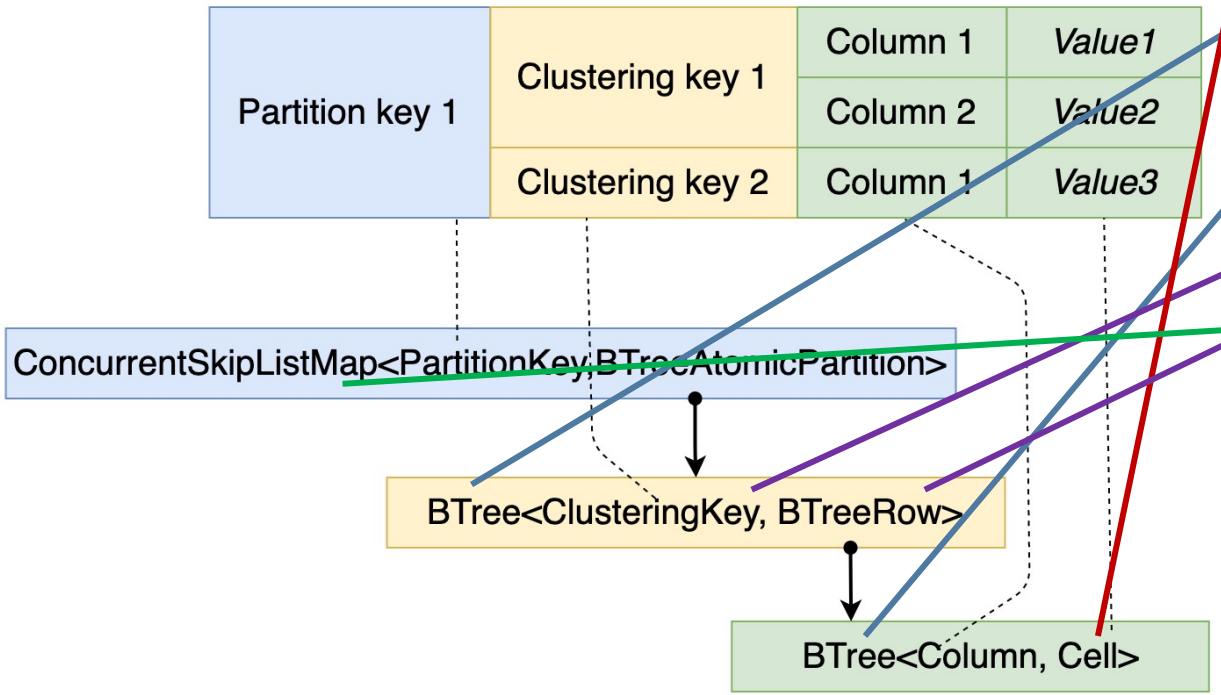
Больше деталей в JPoint 2021: Чтение из Cassandra — внутреннее устройство и производительность

Что в Cassandra хипе?

- native_objects mode

Memtables ->
Chunk|KeyCache ->

Что в Cassandra хипе?

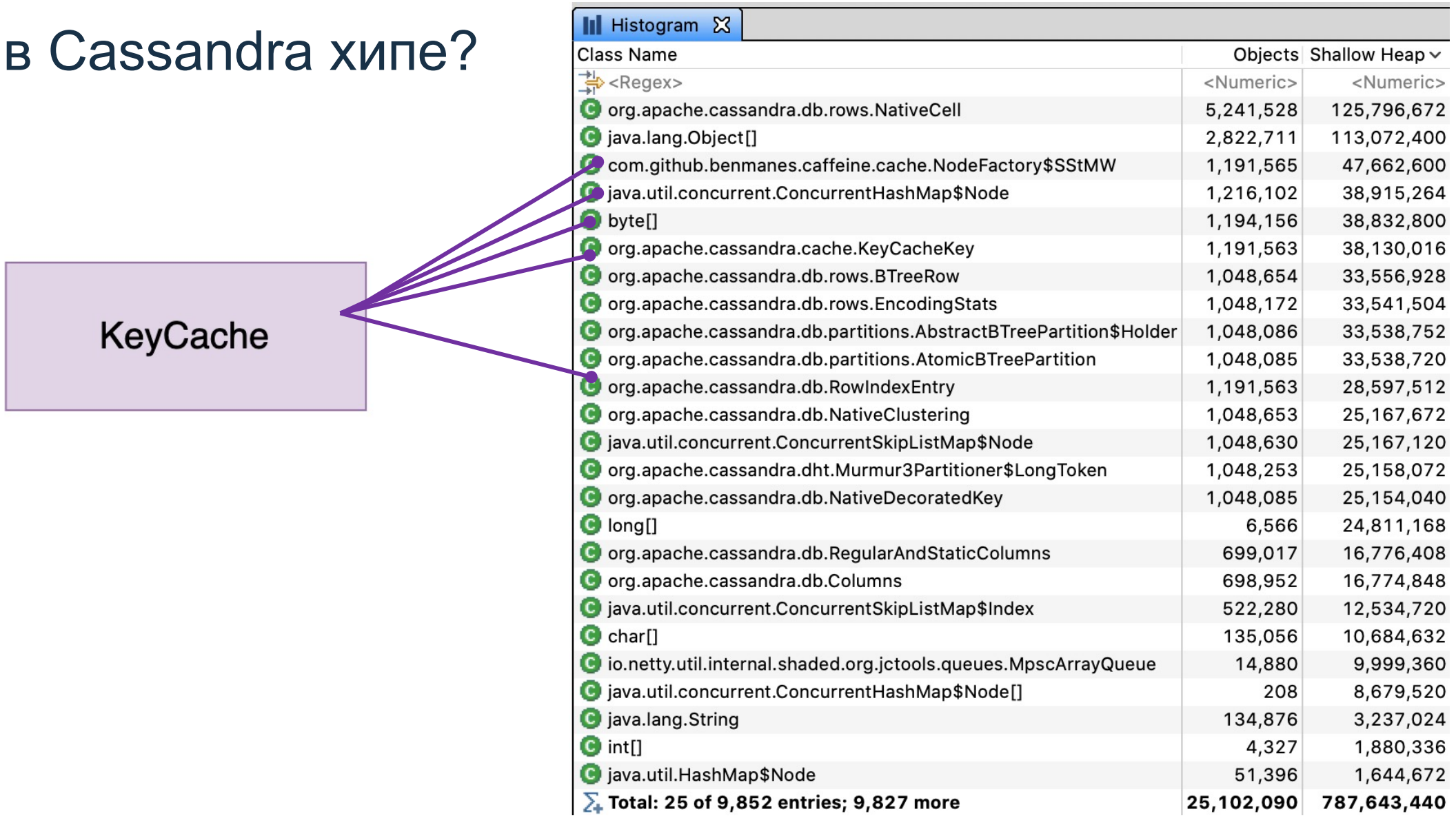


Histogram

Class Name	Objects	Shallow Heap
<Regex>	<Numeric>	<Numeric>
org.apache.cassandra.db.rows.NativeCell	5,241,528	125,796,672
java.lang.Object[]	2,822,711	113,072,400
com.github.benmanes.caffeine.cache.NodeFactory$SStMW	1,191,565	47,662,600
java.util.concurrent.ConcurrentHashMap$Node	1,216,102	38,915,264
byte[]	1,194,156	38,832,800
org.apache.cassandra.cache.KeyCacheKey	1,191,563	38,130,016
org.apache.cassandra.db.rows.BTreeRow	1,048,654	33,556,928
org.apache.cassandra.db.rows.EncodingStats	1,048,172	33,541,504
org.apache.cassandra.db.partitions.AbstractBTreePartition$Holder	1,048,086	33,538,752
org.apache.cassandra.db.partitions.AtomicBTreePartition	1,048,085	33,538,720
org.apache.cassandra.db.RowIndexEntry	1,191,563	28,597,512
org.apache.cassandra.db.NativeClustering	1,048,653	25,167,672
java.util.concurrent.ConcurrentSkipListMap$Node	1,048,630	25,167,120
org.apache.cassandra.dht.Murmur3Partitioner$LongToken	1,048,253	25,158,072
org.apache.cassandra.db.NativeDecoratedKey	1,048,085	25,154,040
long[]	6,566	24,811,168
org.apache.cassandra.db.RegularAndStaticColumns	699,017	16,776,408
org.apache.cassandra.db.Columns	698,952	16,774,848
java.util.concurrent.ConcurrentSkipListMap$Index	522,280	12,534,720
char[]	135,056	10,684,632
io.netty.util.internal.shaded.org.jctools.queues.MpscArrayQueue	14,880	9,999,360
java.util.concurrent.ConcurrentHashMap$Node[]	208	8,679,520
java.lang.String	134,876	3,237,024
int[]	4,327	1,880,336
java.util.HashMap$Node	51,396	1,644,672
Total: 25 of 9,852 entries; 9,827 more	**25,102,090**	**787,643,440**

Что в Cassandra хипе?

KeyCache



Histogram

Class Name	Objects	Shallow Heap
<Regex>	<Numeric>	<Numeric>
org.apache.cassandra.db.rows.NativeCell	5,241,528	125,796,672
java.lang.Object[]	2,822,711	113,072,400
com.github.benmanes.caffeine.cache.NodeFactory$SStMW	1,191,565	47,662,600
java.util.concurrent.ConcurrentHashMap$Node	1,216,102	38,915,264
byte[]	1,194,156	38,832,800
org.apache.cassandra.cache.KeyCacheKey	1,191,563	38,130,016
org.apache.cassandra.db.rows.BTreeRow	1,048,654	33,556,928
org.apache.cassandra.db.rows.EncodingStats	1,048,172	33,541,504
org.apache.cassandra.db.partitions.AbstractBTreePartition$Holder	1,048,086	33,538,752
org.apache.cassandra.db.partitions.AtomicBTreePartition	1,048,085	33,538,720
org.apache.cassandra.db.RowIndexEntry	1,191,563	28,597,512
org.apache.cassandra.db.NativeClustering	1,048,653	25,167,672
java.util.concurrent.ConcurrentSkipListMap$Node	1,048,630	25,167,120
org.apache.cassandra.dht.Murmur3Partitioner$LongToken	1,048,253	25,158,072
org.apache.cassandra.db.NativeDecoratedKey	1,048,085	25,154,040
long[]	6,566	24,811,168
org.apache.cassandra.db.RegularAndStaticColumns	699,017	16,776,408
org.apache.cassandra.db.Columns	698,952	16,774,848
java.util.concurrent.ConcurrentSkipListMap$Index	522,280	12,534,720
char[]	135,056	10,684,632
io.netty.util.internal.shaded.org.jctools.queues.MpscArrayQueue	14,880	9,999,360
java.util.concurrent.ConcurrentHashMap$Node[]	208	8,679,520
java.lang.String	134,876	3,237,024
int[]	4,327	1,880,336
java.util.HashMap$Node	51,396	1,644,672
Total: 25 of 9,852 entries; 9,827 more	**25,102,090**	**787,643,440**

Как оптимизировать использование памяти?

1. Уменьшить число объектов
   - На уровне кода
   - На уровне схемы – меньше колонок
2. Уменьшить накладные расходы для объектов

Memtable – to native

- CEP-11 Pluggable memtable implementations
- CASSANDRA-13474 Cassandra pluggable storage engine, in progress

Memtable – to native

- CEP-11 Pluggable memtable implementations
- CASSANDRA-13474 Cassandra pluggable storage engine, in progress
- Instagram implemented storage layer on top of RocksDB
  - https://instagram-engineering.com/open-sourcing-a-10x-reduction-in-apache-cassandra-tail-latency-d64f86b43589
  - https://github.com/ngcc/ngcc2019

Memtable – to native

- CEP-11 Pluggable memtable implementations
- CASSANDRA-13474 Cassandra pluggable storage engine, in progress
- Instagram implemented storage layer on top of RocksDB
  - https://instagram-engineering.com/open-sourcing-a-10x-reduction-in-apache-cassandra-tail-latency-d64f86b43589
  - https://github.com/ngcc/ngcc2019
- Interesting fact: Ignite 3 goes to the same direction: IGNITE-14747 - RocksDB research: configuration, lifecycle, basic integration

Lilliput project

- https://openjdk.java.net/projects/lilliput/ - “The goal of this project is to explore techniques to downsize Java object headers in the Hotspot JVM from 128 bits to 64 bits or less, reducing Java's memory footprint.”
- Joker 2021: Project Lilliput: Shrinking object headers in the Hotspot JVM

Roman Kennke

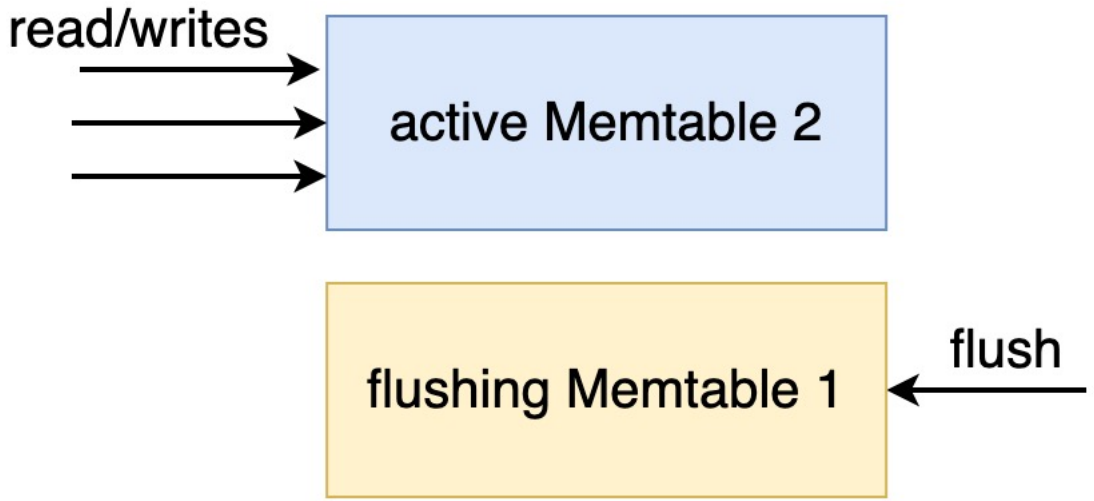
- Координация потоков: фоновые операции и обработка запросов

Concurrent switchover

- Memtable
  - Need to coordinate multiple threads involved into 3 concurrent activities:
    - Writes/reads
    - Flushers (create new memtable and flush old one)
    - Memory release

Concurrent switchover

- Memtable
  - Need to coordinate multiple threads involved into 3 concurrent activities:
    - Writes/reads
    - Flushers (create new memtable and flush old one)
    - Memory release



Concurrent switchover

- Memtable
  - Need to coordinate multiple threads involved into 3 concurrent activities:
    - Writes/reads
    - Flushers (create new memtable and flush old one)
    - Memory release
  - See: org.apache.cassandra.utils.concurrent.OpOrder

Concurrent switchover

- Memtable
  - Need to coordinate multiple threads involved into 3 concurrent activities:
    - Writes/reads
    - Flushers (create new memtable and flush old one)
    - Memory release
  - See: org.apache.cassandra.utils.concurrent.OpOrder

    http://stuff-gil-says.blogspot.com/2014/11/writerreaderphaser-story-about-new.html
    (the algorithm is implemented in HdrHistogram library used by Cassandra Java driver)

    http://concurrencyfreaks.blogspot.com/2013/12/left-right-classical-algorithm.html

- Проблемы для GC при работе с памятью

Bloom filter memory usage

- 1st version - single array: `long[] bits`

Bloom filter memory usage

- 1st version - single array: `long[] bits`
- 2nd version - multi-array: `long[][] bits`
  - CASSANDRA-2466 / CASSANDRA-3618
  - Reason: to avoid CMS full GCs caused by promotion failures for large contiguous chunks of heap

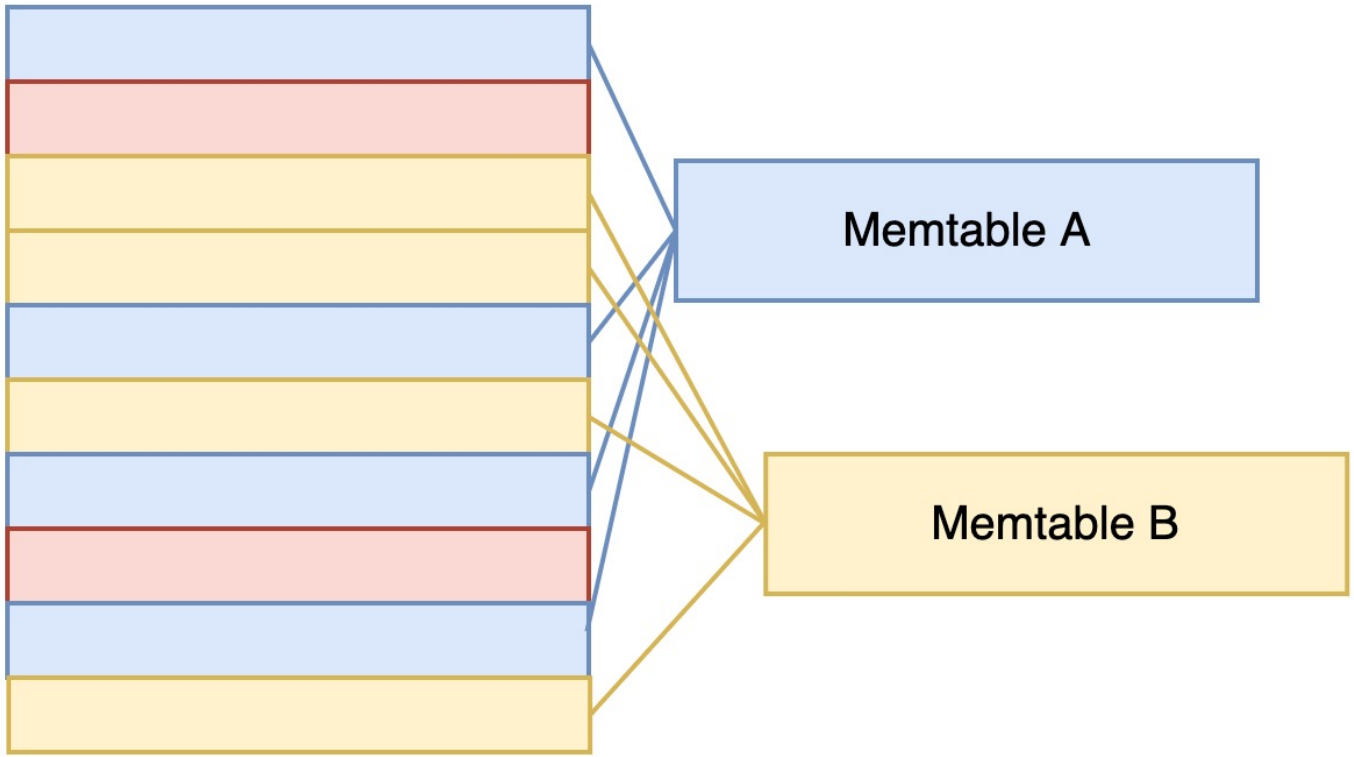
Bloom filter memory usage

- 1st version - single array: `long[] bits`
- 2nd version - multi-array: `long[][] bits`
  - CASSANDRA-2466 / CASSANDRA-3618
  - Reason: to avoid CMS full GCs caused by promotion failures for large contiguous chunks of heap
- 3rd version - offheap: CASSANDRA-4865

```
long peer = com.sun.jna.Native.malloc(size);
...
unsafe.getByte(peer + offset);
```

Memtables and heap fragmentation

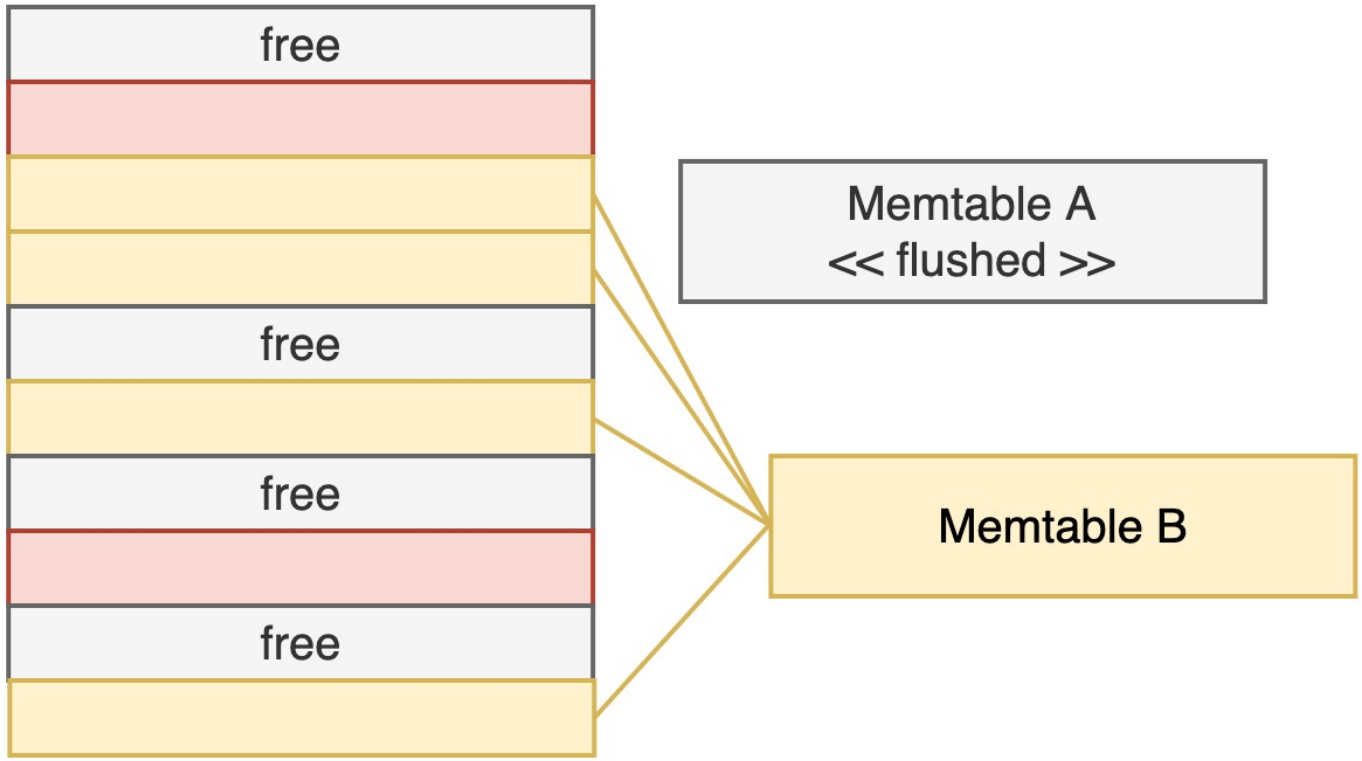
- Memtables accumulate data in memory during a long time (from GC point of view) and release it fully on flush -> fragmentation



Memtables and heap fragmentation

- Memtables accumulate data in memory during a long time (from GC point of view) and release it fully on flush -> fragmentation

Память фрагментирована

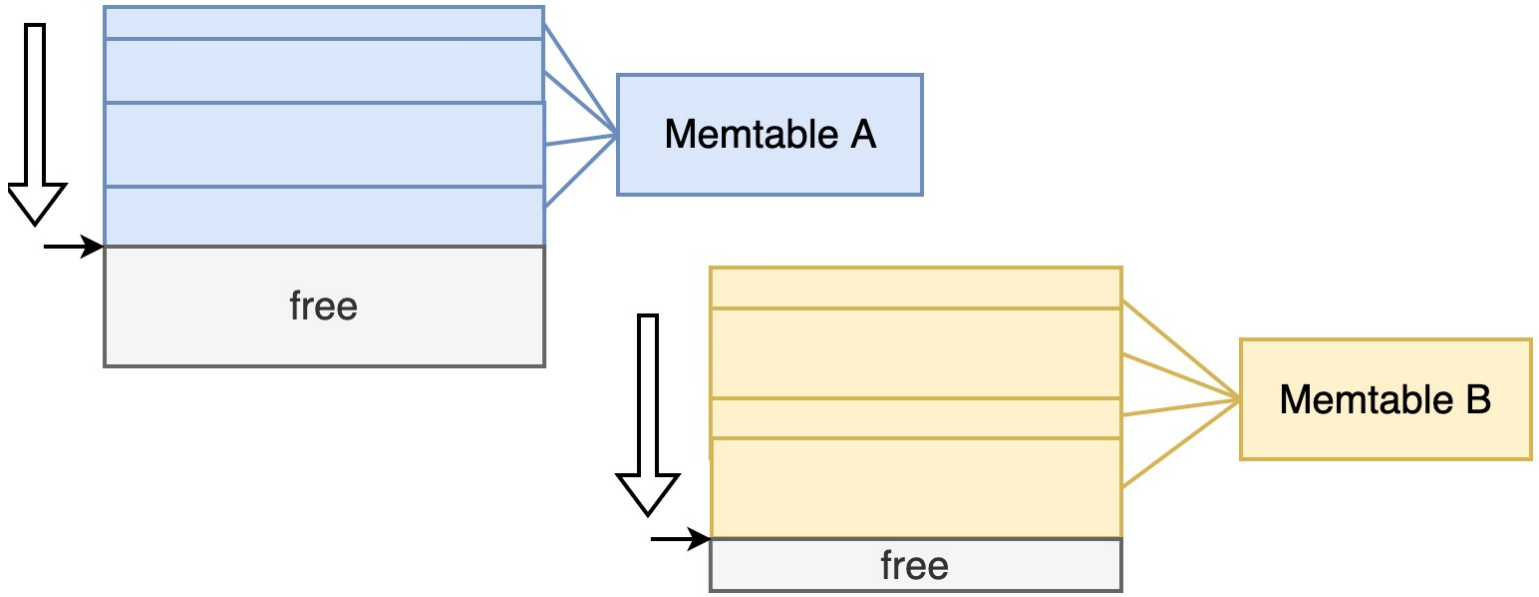


Slab allocator

- Борьба с heap fragmentation – “проблема швейцарского сыра”
  - HBASE: HBASE-3455 (Heap fragmentation in region server)
  - https://www.slideshare.net/cloudera/hbase-hug-presentation
- CASSANDRA-2252 – arena allocation for memtables

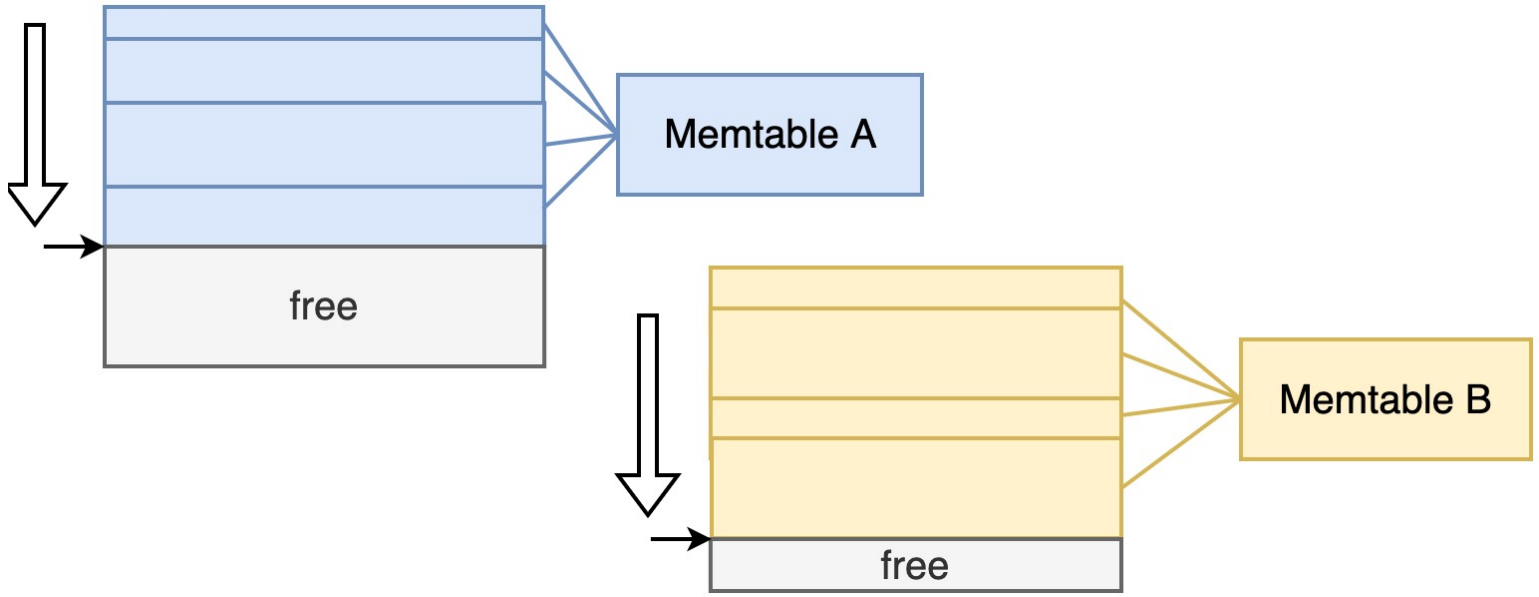
Slab allocator

- Борьба с heap fragmentation – “проблема швейцарского сыра”
  - HBASE: HBASE-3455 (Heap fragmentation in region server)
  - https://www.slideshare.net/cloudera/hbase-hug-presentation
- CASSANDRA-2252 – arena allocation for memtables



Slab allocator

- Борьба с heap fragmentation – “проблема швейцарского сыра”
  - HBASE: HBASE-3455 (Heap fragmentation in region server)
  - https://www.slideshare.net/cloudera/hbase-hug-presentation
- CASSANDRA-2252 – arena allocation for memtables



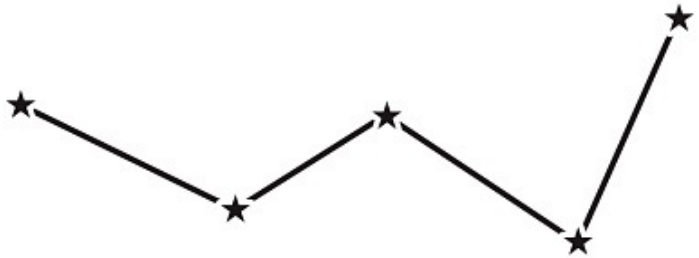
- Открытый вопрос: а насколько это полезно в G1/Shenandoah/ZGC?

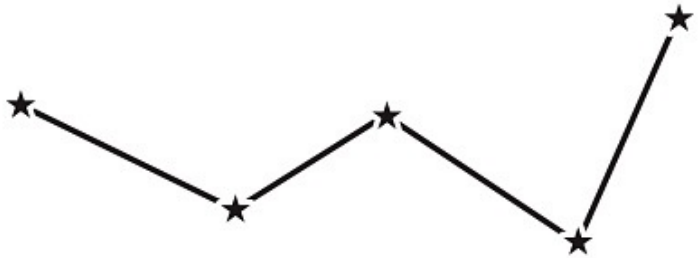
Эпилог

Итого

Итого

- 3rd party исходный код можно читать и для того, чтобы узнать что-то новое
- Не все проблемы решены, есть те, что ждут своих героев ☺





Кассиопея

Обозначение: Cas + sand + ra

Cassandra

Thank You

ThreadPoolExecutor в Cassandra

- **CacheCleanupExecutor** - *cleanup of keys no longer belonging to a node (triggered via nodetool)*
- **NettyStreaming-Outbound-<remote-node-address>** - *execute file transfer (streaming) tasks*
- **RepairJobTask** – *thread pool to execute tasks related to a repair job (inner sub-tasks)*
- **Repair#** - *run repair job itself*
- **Repair-Task** – *repair command executor*
- **HintsWriteExecutor -** *A single threaded executor that exclusively writes all the hints and otherwise manipulate the writers.*
- **SSTableBatchOpen -** *scan for sstables corresponding to a table and load them*
- **<Permissions|Roles|Credentials|JmxPermissions|NetworkPermissions>CacheRefresh** – *used to refresh cache entries in Caffeine*
- **CompactionExecutor** – *run compactions*
- **ValidationExecutor** – *run validations*
- **ViewBuildExecutor** - *build materialized views*
- **AntiEntropy** – *handle repair requests (receiver part)*
- **Migration -** *when node receives updated schema state from the schema migration coordinator node*
- **Misc** – *various internal operations*

ThreadPoolExecutor в Cassandra

- **MemtableFlushWriter** – flush Memtable content to disk as SSTable files
- **MemtablePostFlush**
- **MemtableReclaimMemory**
- **PerDiskMemtableFlushWriter_<index>**
- **CommitLogArchiver**
- **HintsDispatcher -** *executor for dispatching hints.*
- **SecondaryIndexManagement -** *executes tasks returned by Indexer#addIndexColumn which may require index(es) to be (re)built*
- **SASI-General**
- **SASI-Memtable**
- **Sampler** – *to add samples like topWritePartitionFrequency*
- **PendingRangeCalculator -** *Calculate pending ranges according to bootsrapping and leaving nodes.*
- **UserDefinedFunctions|UserDefinedScriptFunctions** – *UDF execution*
- **InternalResponse - ?**

ScheduledThreadPoolExecutor в Cassandra

- **ScheduledFastTasks -** *periodic fast (sub-microsecond) tasks*
  - *approximate time sampling*
- **ScheduledTasks -** *periodic short (sub-second) tasks, like*
  - *trigger periodic memtable flush*
  - *log slow operations*
  - *Dynamic snitch update*
  - *Dropped messages logging*
  - *Trigger hints dispatch*
  - *Unused connections monitoring*
  - *Pull unreceived schema versions*
  - *Check TLS certificates for hot reloading*
  - *Watch property file snitch changes*
- **NonPeriodicTasks -** *used for tasks that can have longer execution times, and usually are non periodic*
- **OptionalTasks -** *used for tasks that do not need to be waited for on shutdown/drain.*
- **GossipTasks** – *send gossip requests*
- **GossipStage – process gossip requests (receiver part?)**
- **IndexSummaryManager -** *distribute a fixed pool of memory for index summaries across a set of SSTables based on their recent read rates*
- **SnapshotCleanup –** *cleanup expired snapshots (since 4.1, https://issues.apache.org/jira/browse/CASSANDRA-16789)*
- **BatchlogTasks** – *trigger recovery of multi-partition batches*
- **Callback-Map-Reaper** – *timeouts for server-2-server requests*

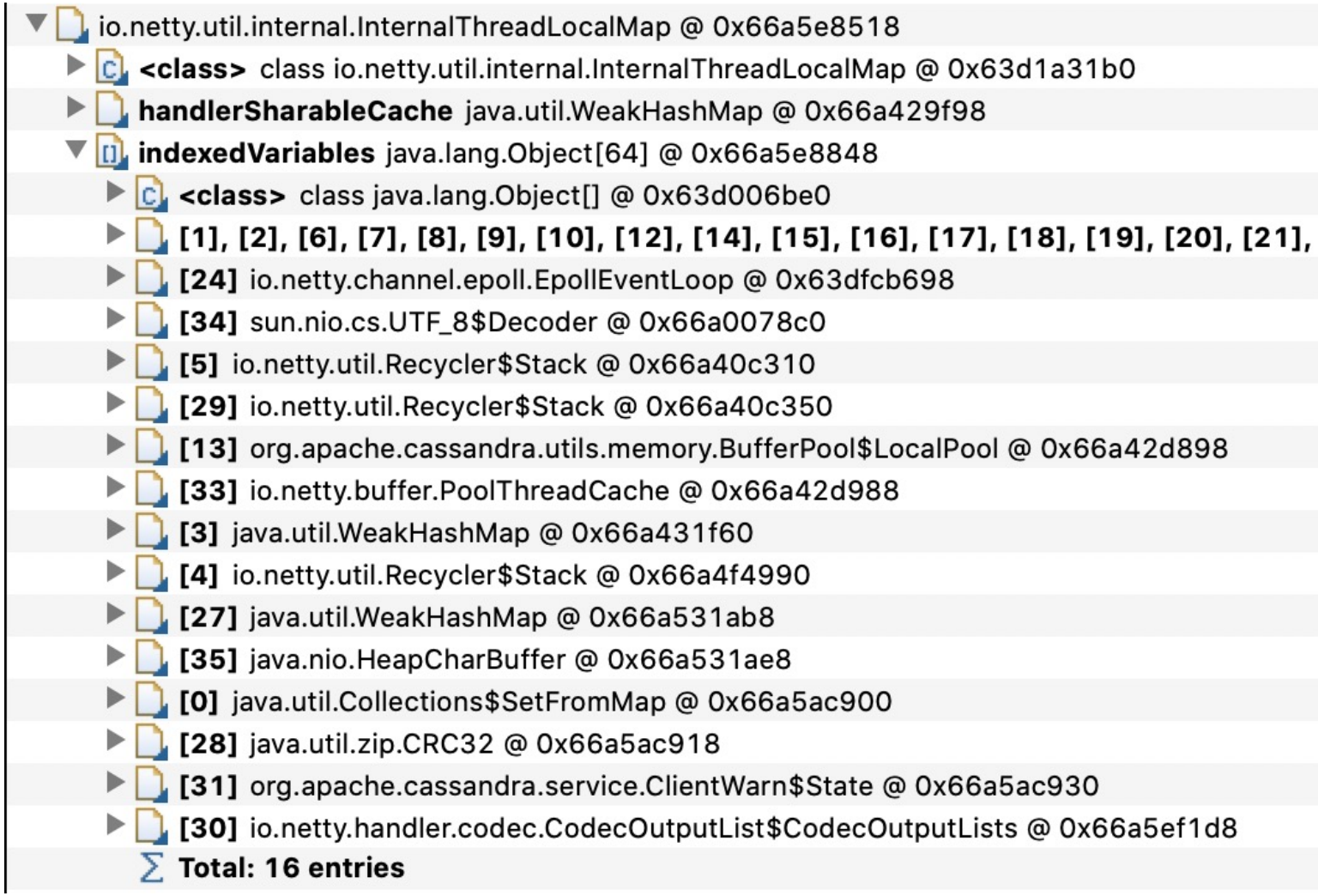
Thread pool monitoring - JMX

- Not provided for Netty event loop and ScheduledThreadPoolExecutor
- Configuration:
  - get/setCorePoolSize
  - get/setMaximumPoolSize
  - getMaxTasksQueued – queue size limit
- Metrics:
  - ActiveTasks – executed in a thread
  - PendingTasks – waiting in a queue
  - CompletedTasks
  - CurrentlyBlockedTasks – queue is full
  - TotalBlockedTasks

Timers – Future.get

- Future.get -> parkNano -> futexes (pthread_cond_timedwait) -> hrtimer
- https://hazelcast.com/blog/locksupport-parknanos-under-the-hood-and-the-curious-case-of-parking/
- https://www.kernel.org/doc/Documentation/timers/hrtimers.txt (red-black trees)

FastThreadLocal Usage in Cassandra



jvmquake: Why not GCHeapFreeLimit, GCTimeLimit

https://github.com/Netflix-Skunkworks/jvmquake

“There are also some options that are supposed to control GC overhead:

- GCHeapFreeLimit, GCTimeLimit and +UseGCOverheadLimit. These options are supposed to cause an OOM in the case where we are not collecting enough memory, or are spending too much time in GC. However in practice I've never been able to get these to work well, and GCOverheadLimit is afaik only supported in CMS.
- **TLDR**: In my experience these JVM flags are **hard to tune** and **only sometimes work** if they work at all, and often are limited to a subset of JVMs of collectors.”

:

Allocations

- CASSANDRA-16471 (org.apache.cassandra.io.util.DataOutputBuffer#scratchBuffer is around 50% of all memory allocations)
- CASSANDRA-15511 (Utilising BTree Improvements)
- CASSANDRA-15510 (BTree: Improve Building, Inserting and Transforming)
- CASSANDRA-15387 (Reduce compaction & local read path garbage)
- CASSANDRA-14654 (Reduce heap pressure during compactions)
- CASSANDRA-14261 (Compaction Profiling Improvements)
- CASSANDRA-13789 (Reduce memory copies and object creations when acting on ByteBufs)
- CASSANDRA-13760 (presize collections)

Netty allocator

- io.netty.buffer.PooledByteBufAllocatorMetric
  - numHeapArenas
  - numDirectArenas
  - numThreadLocalCaches
  - JMX metrics?
  - io.netty.buffer.PooledByteBufAllocator#dumpStats
- Return from another thread
- https://programmer.group/pool-area-of-netty-memory-pool.html
- https://www.fatalerrors.org/a/netty-memory-pool-pool-chunk-principle-is-explained-here.html
- https://github.com/netty/netty/pull/10267 (jemalloc 4)
  - https://people.freebsd.org/~jasone/jemalloc/bsdcan2006/jemalloc.pdf
  - https://www.facebook.com/notes/10158791475077200/
  - http://jemalloc.net/
  - https://www.youtube.com/watch?v=3IB52n4Tbuo

Malloc/jmalloc

- https://issues.apache.org/jira/browse/CASSANDRA-6126 (Set MALLOC_ARENA_MAX in cassandra-env.sh for new glibc per-thread allocator)
- https://issues.apache.org/jira/browse/CASSANDRA-7883 (Allow plugging JEMalloc for off-heap memtables)

GC

- https://thelastpickle.com/blog/2020/06/29/cassandra_4-0_garbage_collectors_performance_benchmarks.html
- https://www.scylladb.com/2020/10/06/c-scylla-in-battle-royale-against-javas-zgc-shenandoah-g1-cassandra/
- https://medium.com/outbrain-engineering/leveraging-shenandoah-to-cut-cassandras-tail-latency-d2a11725e4d8
- https://www.slideshare.net/ScyllaDB/comparing-apache-cassandra-40-30-and-scylladb
- https://rodrigo-bruno.github.io/mentoring/77998-Carlos-Goncalves_dissertacao.pdf
- https://openjdk.java.net/jeps/404