



# Скажите “Ой!”

JVM и OOM Killer



- Я – Владимир Плизгá
- 2011-2021: **ЦФТ** (Java)
  - бэкенд Интернет-банков
- 2021-  : **Tibbo Systems** (Java)
  - бэкенд IoT-платформы
- 2023-  : **StegoText.ru** (Java)
  - прикладная криптография



 [Toparvion](#)

 [toparvion.pro](#)

 [StegoTrend](#)



# Случай из практики

---

Projects /  AggreGate /  AGG-15393

Out Of Memory Killer stops the Java process several times a day



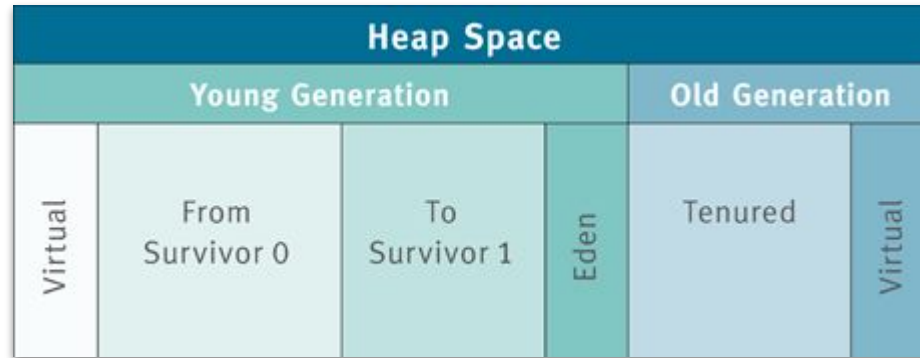
- Обработка  $\approx 20$  метрик с 10К устройств
- Liberica JDK 8, G1 GC, -Xmx16G
- Oracle Linux, 24 GB RAM, no swap

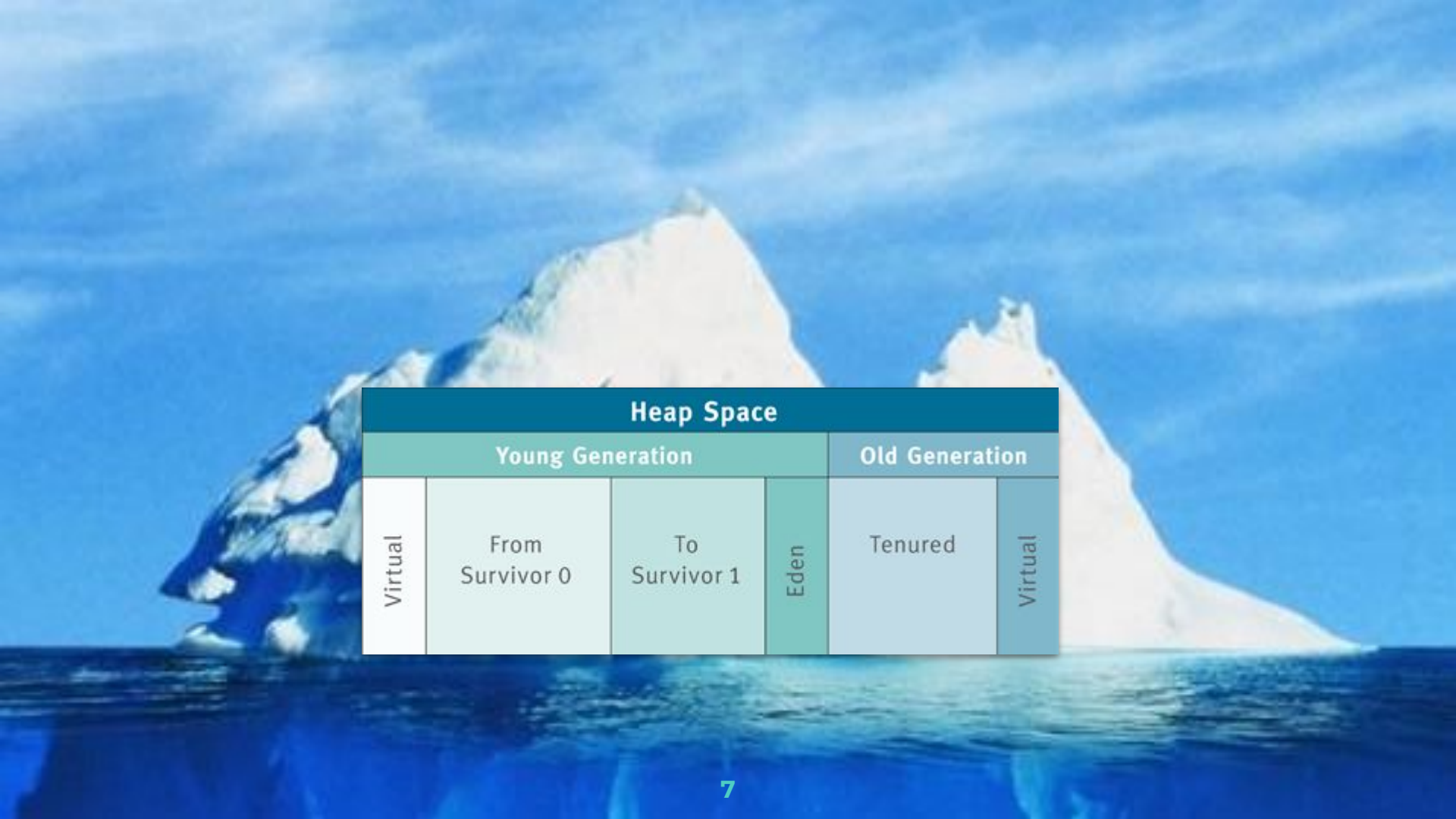
# Основные понятия

Чтобы говорить  
на одном языке



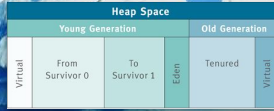
# java.lang.OutOfMemoryError: Java heap space



An iceberg floating in the ocean under a blue sky. The visible tip of the iceberg is small, while the much larger submerged part is hidden below the water line, illustrating the concept of virtual memory.

| Heap Space       |                 |               |      |                |         |
|------------------|-----------------|---------------|------|----------------|---------|
| Young Generation |                 |               |      | Old Generation |         |
| Virtual          | From Survivor 0 | To Survivor 1 | Eden | Tenured        | Virtual |

# HEAP



# NON-HEAP

(off-heap, native)

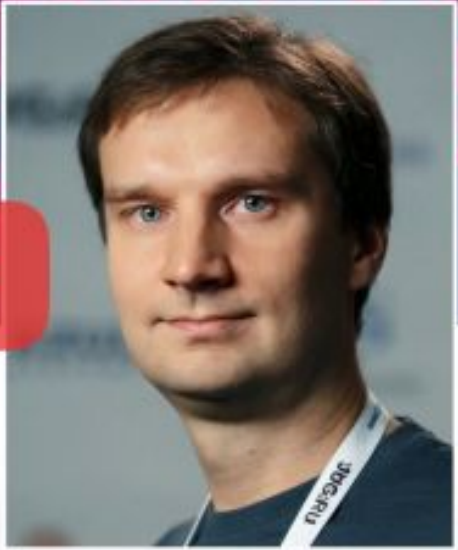

Code Cache +  
GC & Compiler +  
Symbol tables +  
Thread stacks +  
Direct buffers +  
Mapped files +  
Metaspace +  
Native libs +  
malloc +

...



Joker<?> 2018

**Андрей Паньгин**  
Одноклассники



Память Java-процесса  
по полочкам

---

<https://www.youtube.com/watch?v=kKigibHrV5I>

# Что бывает при нехватке non-heap?

---

В лучшем (редком) случае:

```
java.lang.OutOfMemoryError: Direct buffer memory
```

Но как правило: (логи ОС)

```
kernel: oom-kill:constraint=CONSTRAINT_NONE, ...
```

```
kernel: Out of memory: Killed process 3618718 (java)  
total-vm:28272408kB, anon-rss:22456024kB, ...
```

```
kernel: oom_reaper: reaped process 3618718 (java),  
now anon-rss:5949484kB, ...
```



# А чё так грубо?

Нельзя ли было просто выдать ошибку при попытке аллоцировать память?

# Memory Overcommitment

---

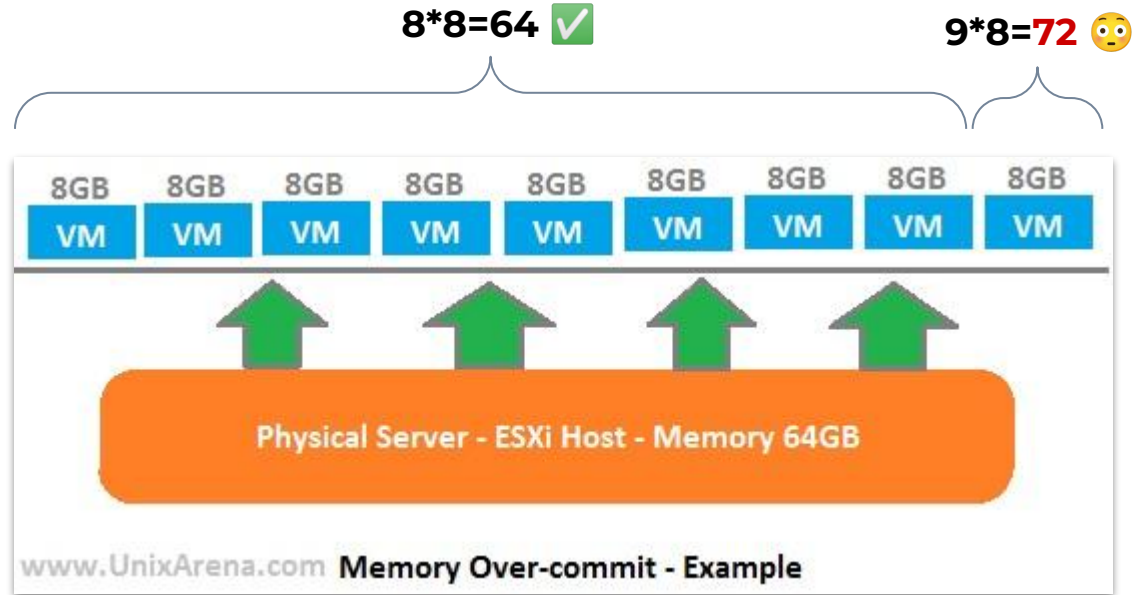
- Встроенная ~~спекуляционная~~ функция ОС
- Основана на предположении, что запрашиваемая память **не будет** использована полностью
- Позволяет запрашивать **больше** памяти, чем есть на самом деле

**С. Михалков**

*Жадный  
Вартан*



# Memory Overcommitment



# Заметка по терминологии

---

- “Memory over**commitment**” не относится к “**committed memory**” в JVM:
- в ОС *overcommitment* значит “понаобещала”
- в JVM *committed* значит “застолбила”

# Можно ли на это влиять?

---

- Да:

```
sysctl vm.overcommit_memory=<mode>
```

- 0 Heuristic overcommit handling (default)
- 1 Always overcommit
- 2 Don't overcommit



# Out Of Memory Killer

---

- “Предотвращатель” большой катастрофы
- Не имеет явного выключателя
- Поведение зависит от многих факторов\*
- Не должен считаться “штатным” случаем



# \* Memory Swapping (Paging)

---

- Вытеснение страниц памяти на диск
- Может **отложить** приход OOM Killer'a
- Может очень сильно **замедлять** работу
- Поэтому часто оказывается **выключенным**

# Что можно успеть при ООМ?

---

## В куче

- ▣ ~~try/catch~~
- ▣ -XX:+HeapDumpOnOutOfMemoryError
- ▣ -XX:OnOutOfMemoryError=cmd

## Вне кучи



© [tlum.ru](http://tlum.ru)

# Куда смотреть?

- В Linux: `top`, `pmap`, `cat /proc/<pid>/status`
- В JMX: `jconsole`, Mission Control, VisualVM
- В JVM: **Native Memory Tracking**



jconsole

# Native Memory Tracking

Главная  
зацепка в  
расследовании



# JVM NMT

---

- Встроенная функция JVM (HotSpot)
- Надо включить заранее:
  - `XX:NativeMemoryTracking=[off|summary|detail]`
- Имеет overhead:
  - -5-10% performance
  - +2 words/malloc

# Как получить результаты NMT

---

- ▣ Задать референтную точку: (опционально)

```
$ jcmd <pid> VM.native_memory baseline
```

- ▣ Запросить текущую статистику:

```
$ jcmd <pid> VM.native_memory summary[.diff]
```

- ▣ Если нет jcmd, можно через jattach:

```
$ jattach <pid> jcmd "VM.native_memory summary"
```

| Operations                 | Name         | Value     | Description  |
|----------------------------|--------------|-----------|--|
| ⚠ GC.heap_info             | baseline     | <Boolean> | request runtime to baseline current memory usage, so it can be compared against previous baseline. |
| ⚠ GC.run                   | detail       | <Boolean> | request runtime to report memory allocation >= 1K by each callsite.                                |
| ⚠ GC.run_finalization      | detail.diff  | <Boolean> | request runtime to report memory detail comparison against previous baseline.                      |
| ⚠ JFR.start                | scale        | <String>  | Memory usage in which scale, KB, MB or GB  |
| ⚠ Thread.print             | shutdown     | <Boolean> | deprecated.  |
| ⚠ VM.cds                   | statistics   | <Boolean> | print tracker statistics for tuning purpose.   |
| ⚠ VM.class_hierarchy       | summary      | <Boolean> | request runtime to report current memory summary, which includes memory usage.                     |
| ⚠ VM.classloaders          | summary.diff | <Boolean> | request runtime to report memory summary comparison against previous baseline.                     |
| ⚠ VM.metaspace             |              |           |  |
| ⚠ VM.native_memory         |              |           |  |
| ⚠ VM.print_touched_methods |              |           |  |
| ⚠ VM.stringtable           |              |           |  |
| ⚠ VM.symboltable           |              |           |  |
| ⚠ VM.systemdictionary      |              |           |  |
| 🚫 GC.class_histogram       |              |           |  |
| 🚫 JVMTI.data_dump          |              |           |  |

 VM.native\_memory

Mission Control — вариант jcmd для ленивых экономных

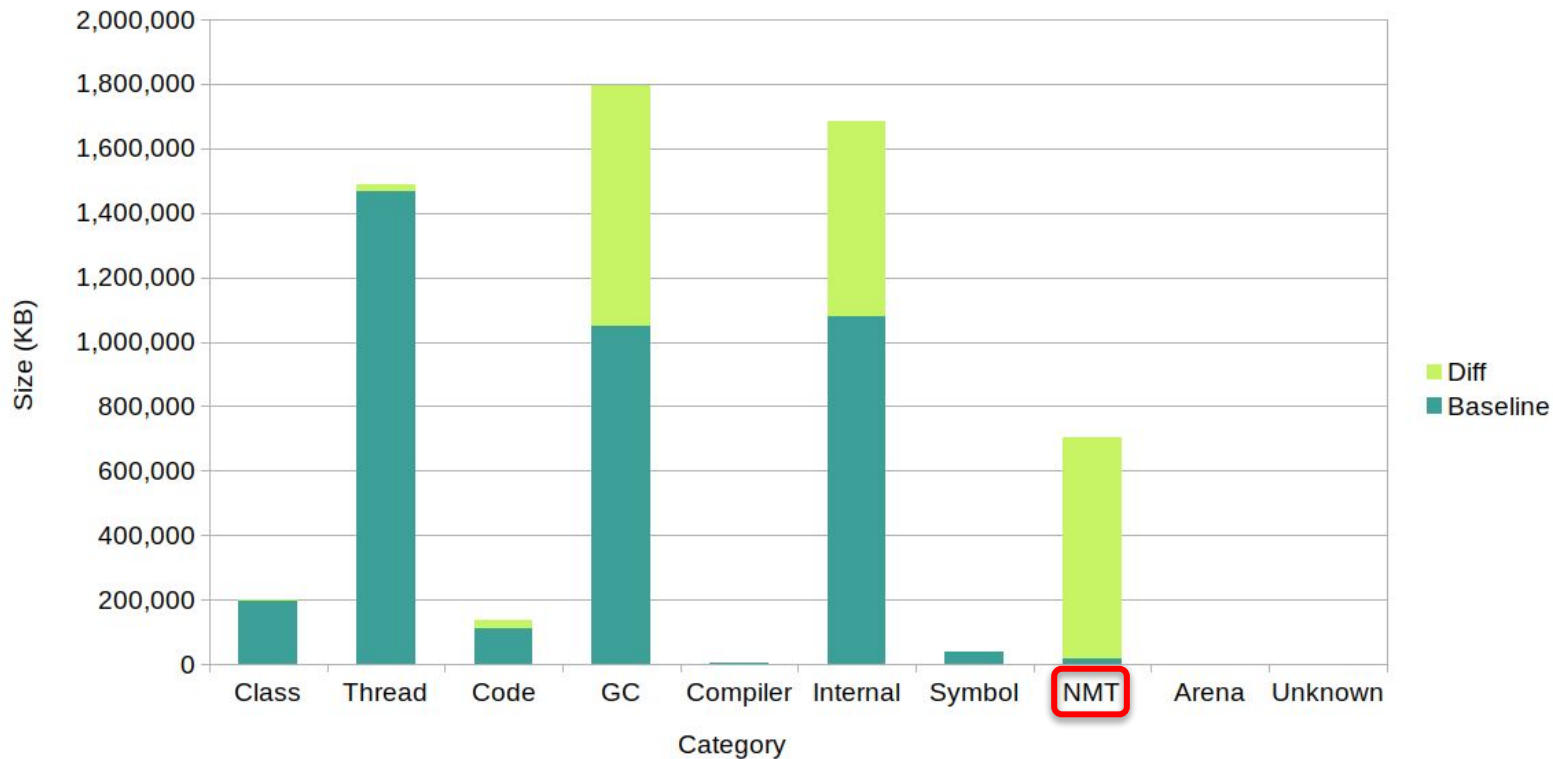


# Пример вывода

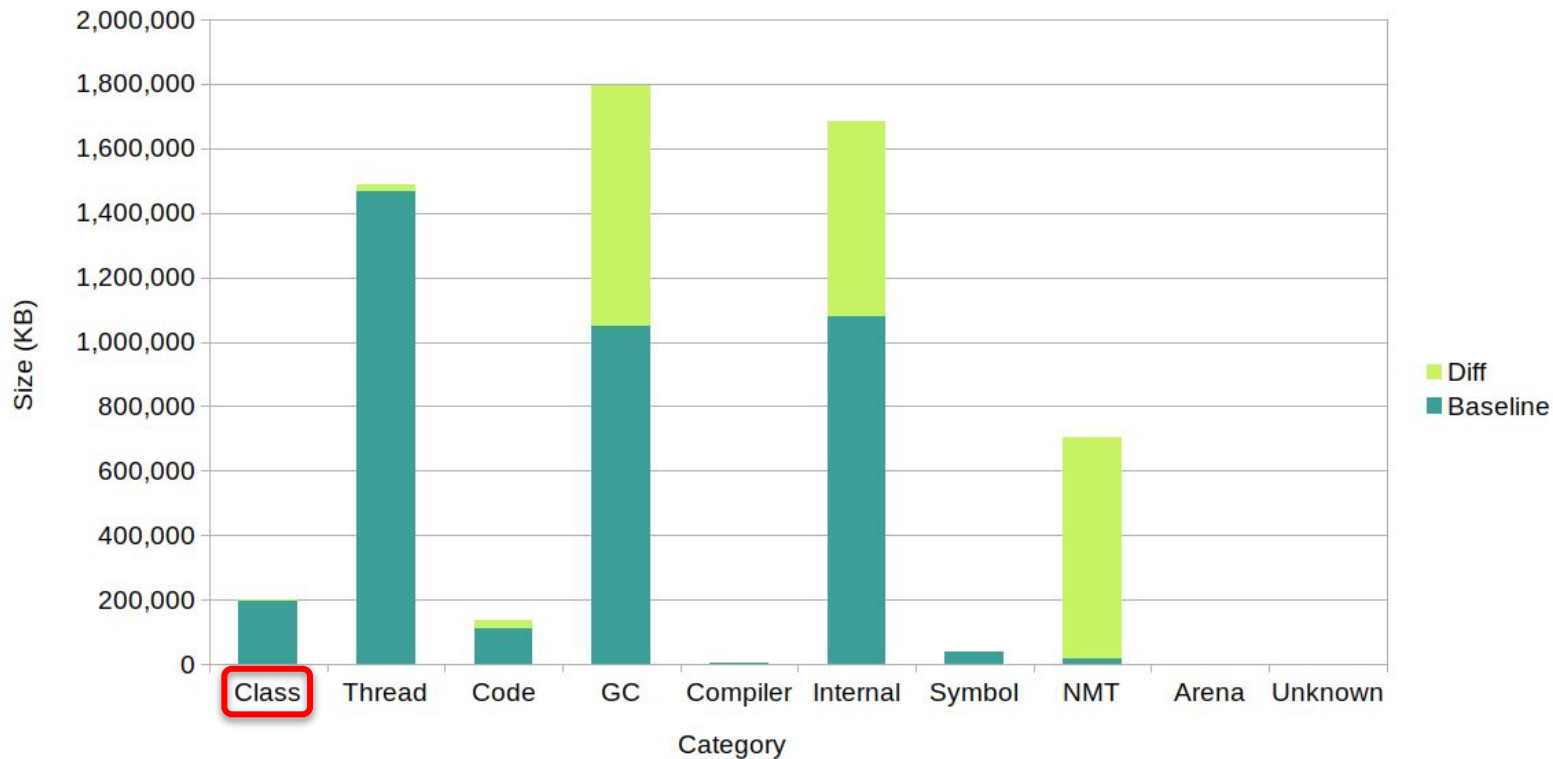
- Растёт почти с каждой версией JDK
- Категории описаны хоть и не полностью

```
Native Memory Tracking:
Total: reserved=22607111KB +1636088KB, committed=21415719KB +1656876KB
-
  Java Heap (reserved=15728640KB, committed=15728640KB)
    (mmap: reserved=15728640KB, committed=15728640KB)
-
  Class (reserved=1223832KB +2491KB, committed=200116KB +3259KB)
    (classes #30901 -280)
    (malloc=5272KB +443KB #65472 +5996)
    (mmap: reserved=1218560KB +2048KB, committed=194844KB +2816KB)
-
  Thread (reserved=1485308KB +18845KB, committed=1485308KB +18845KB)
    (thread #3734 +48)
    (stack: reserved=1468116KB +18624KB, committed=1468116KB +18624KB)
    (malloc=12822KB +165KB #22398 +288)
    (arena=4371KB +56 #7462 +96)
-
  Code (reserved=272270KB +3996KB, committed=135318KB +24016KB)
    (malloc=22670KB +3996KB #28082 +3766)
    (mmap: reserved=249600KB, committed=112648KB +20020KB)
-
  GC (reserved=1408163KB +315577KB, committed=1408163KB +315577KB)
    (malloc=791715KB +315577KB #525565 +248761)
    (mmap: reserved=616448KB, committed=616448KB)
-
  Compiler (reserved=5396KB +751KB, committed=5396KB +751KB)
    (malloc=5262KB +751KB #7728 +1060)
    (arena=135KB #7)
-
  Internal (reserved=1686946KB +603978KB, committed=1686942KB +603978KB)
    (malloc=1686910KB +603978KB #45439096 +44579107)
    (mmap: reserved=36KB, committed=32KB)
-
  Symbol (reserved=37270KB +35KB, committed=37270KB +35KB)
    (malloc=32564KB +35KB #362454 +460)
    (arena=4707KB #1)
-
  Native Memory Tracking (reserved=727516KB +701253KB, committed=727516KB +701253KB)
    (malloc=1341KB +539KB #16736 +6500)
    (tracking overhead=726175KB +700714KB)
-
  Arena Chunk (reserved=1050KB -10839KB, committed=1050KB -10839KB)
    (malloc=1050KB -10839KB)
-
  Unknown (reserved=30720KB, committed=0KB)
    (mmap: reserved=30720KB, committed=0KB)
```

# Пример вывода NMT



# Пример вывода NMT



# NMT: категория CLASS

- Метаданные классов (не экземпляров)
- Как правило, не растёт

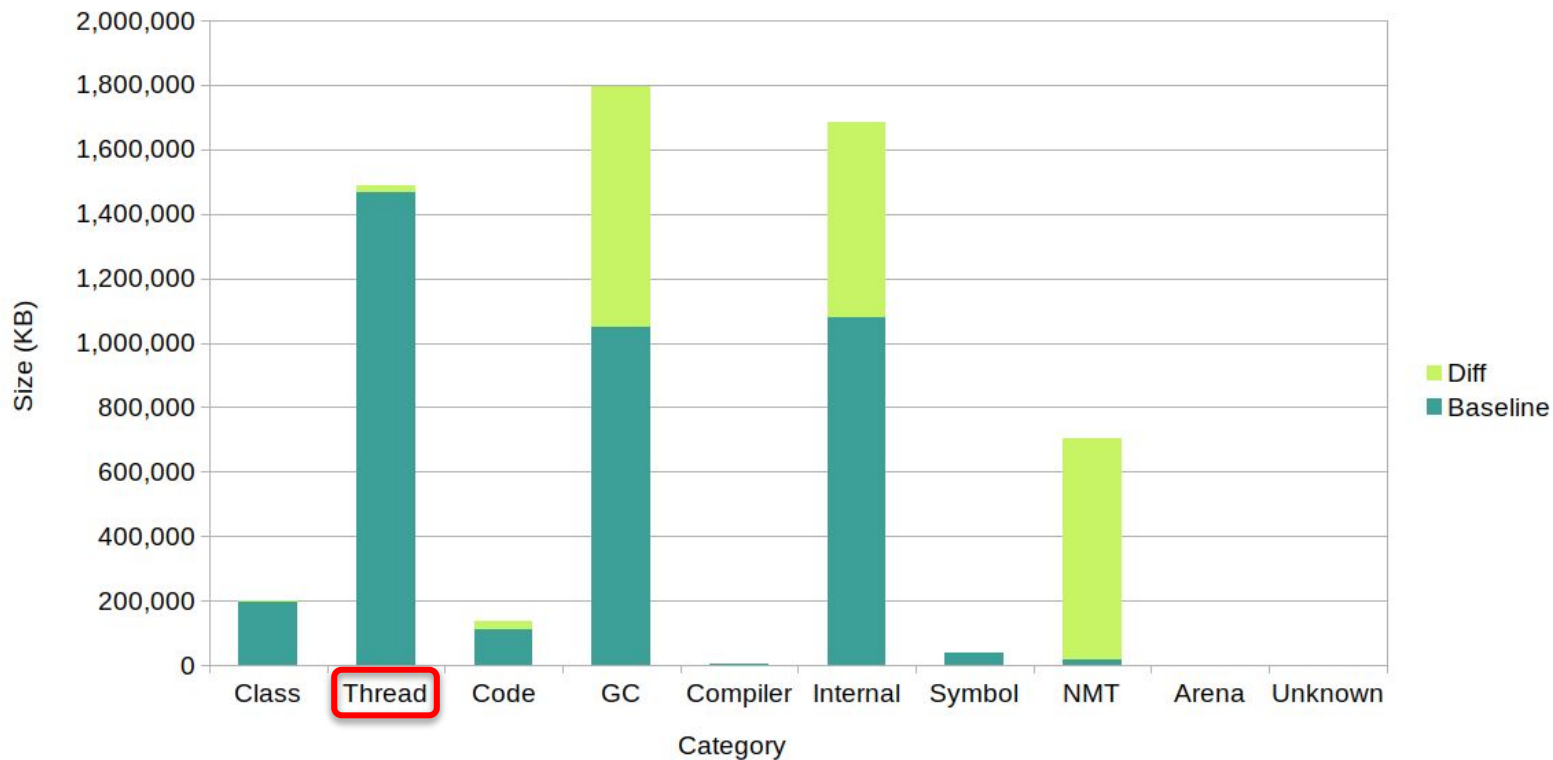


# NMT: категория CLASS

---

- Метаданные классов (не экземпляров)
- Как правило, не растет
  - но лучше понаблюдать
- По умолчанию не ограничена
  - но можно задать потолок:  
`-XX:MaxMetaspaceSize=256m`

# Пример вывода NMT



# NMT: категория **THREAD**

---

- Данные Java-потоков (стеки)
- Не имеет лимита через опции JVM
- Сильно зависит от числа потоков и от предельного размера стека:  
-Xss (по умолчанию 1M)

# Как рассчитать размер **THREAD**

---

- ▣ Определить число потоков по JMX или:  
`$ jcmd <pid> Thread.print | grep "tid=" | wc -l`
- ▣ Прикинуть размер стека тестом или:  
<https://github.com/apangin/jstackmem>
- ▣ Перемножить и взять с запасом



# Другие лимиты в **THREAD**

---

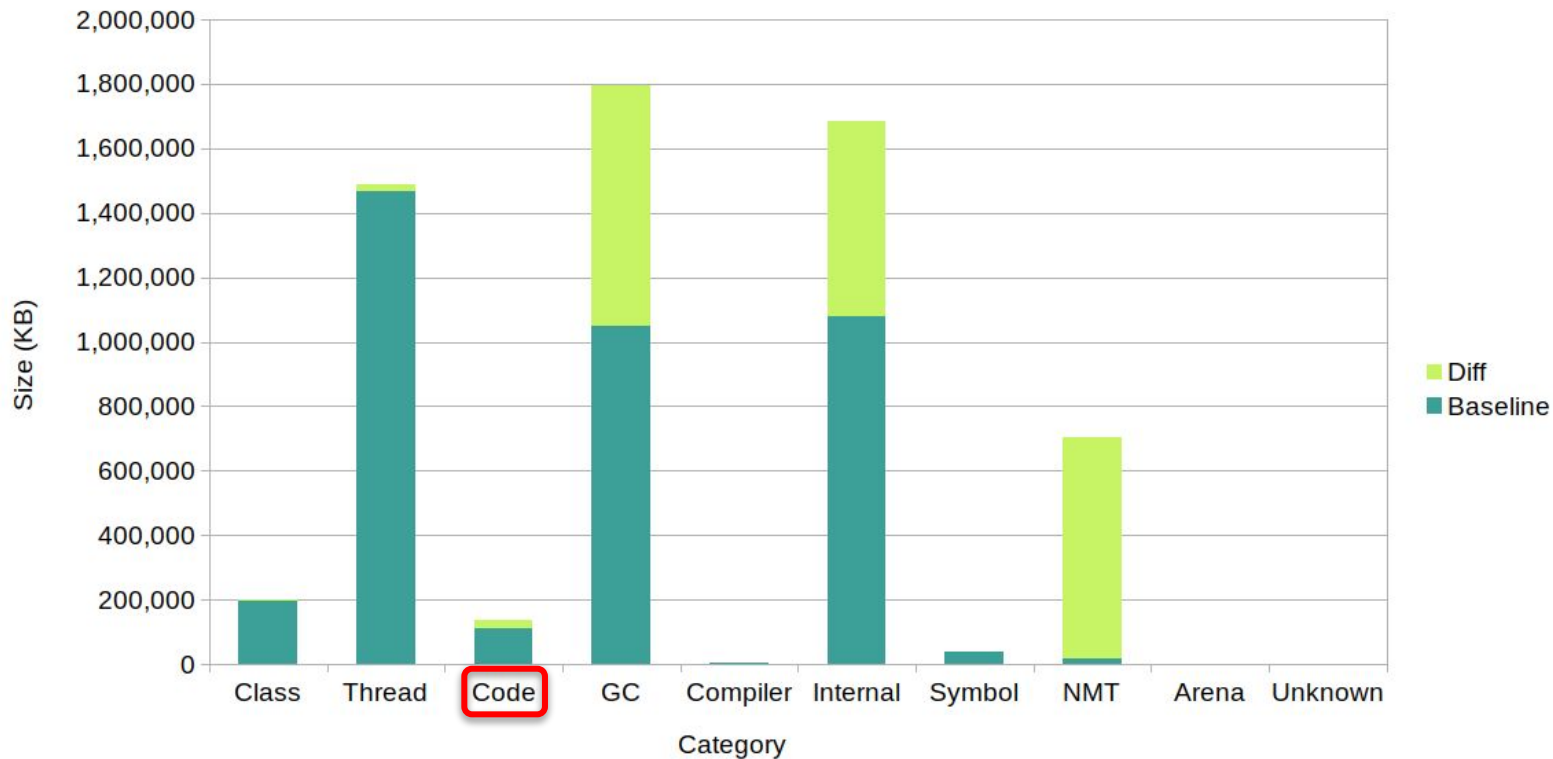
- Слишком много потоков в ОС:

```
java.lang.OutOfMemoryError:  
unable to create new native thread
```

- Превышен размер стека:

```
java.lang.StackOverflowError
```

# Пример вывода NMT

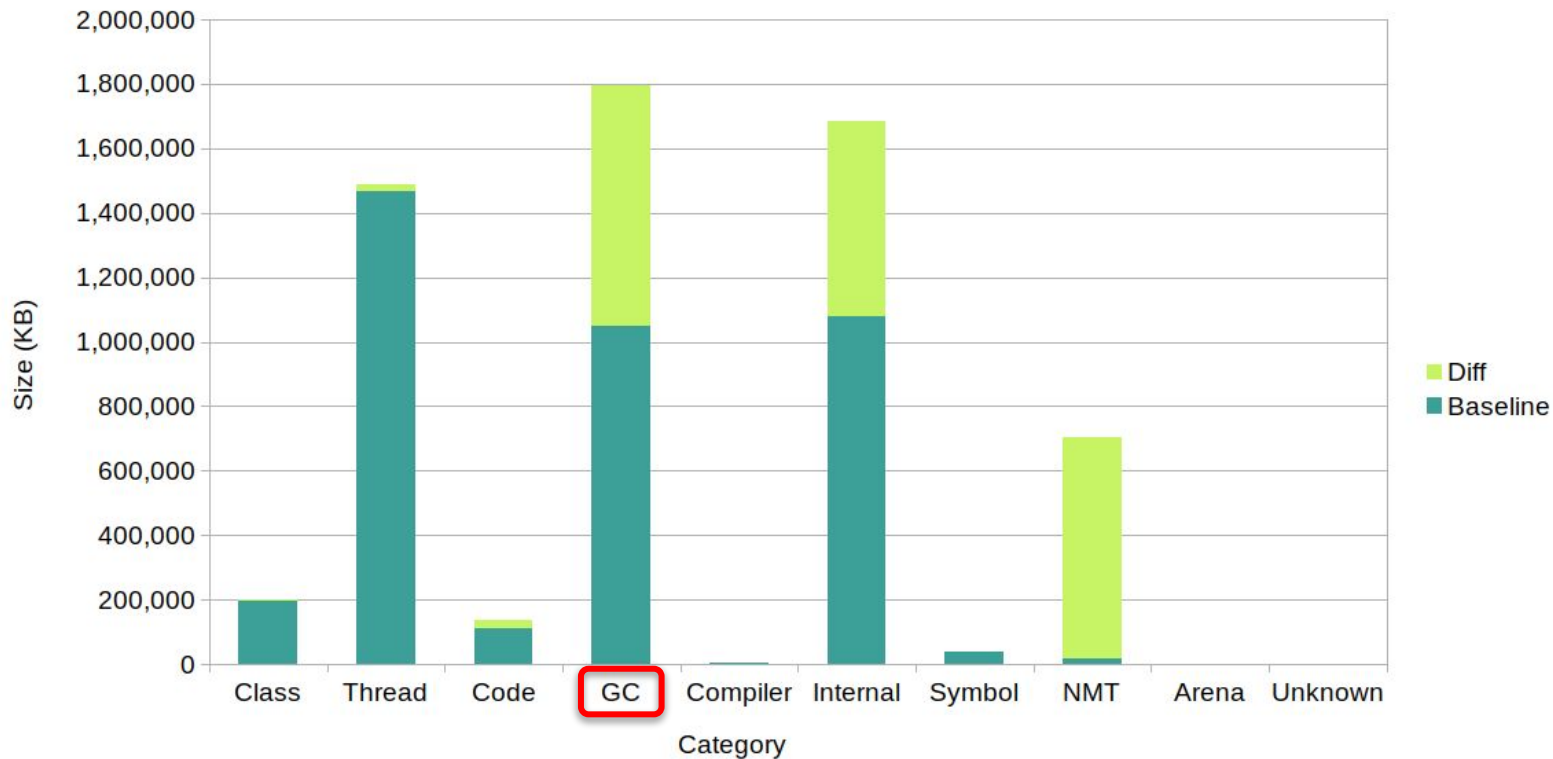


# NMT: категория CODE

---

- ▣ Хранилище сгенерированного кода
- ▣ Можно ограничить:  
`-XX:ReservedCodeCacheSize[=240M]`
- ▣ Подробнее:
  - ▣ Владимир Ситников — Java тормозит: CodeCache edition (JPoint 2019)

# Пример вывода NMT

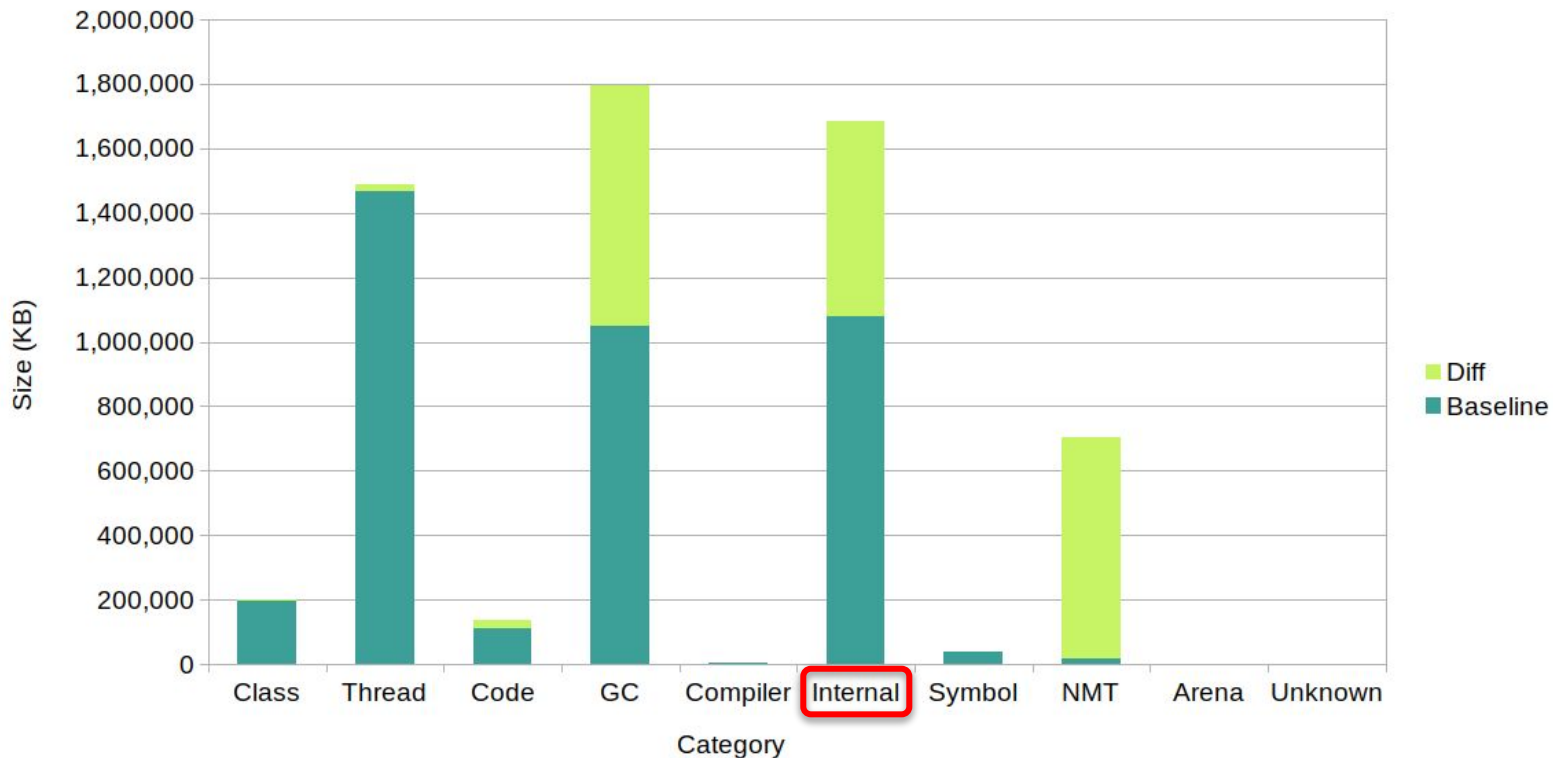


# NMT: категория GC

---

- Собственные данные сборщика мусора
- Не имеет ограничения
- Сильно зависит от самого сборщика:
  - <https://youtu.be/c755fFv1Rnk?t=678>

# Пример вывода NMT



# NMT: категория OTHER (INTERNAL)

---

- По смыслу означает “прочее”
  - поэтому в Java 11 разбита на две (+Other)
- Не имеет ограничения
  - кроме `-XX:MaxDirectMemorySize`
- Может содержать почти что угодно
  - поэтому нужны детали NMT

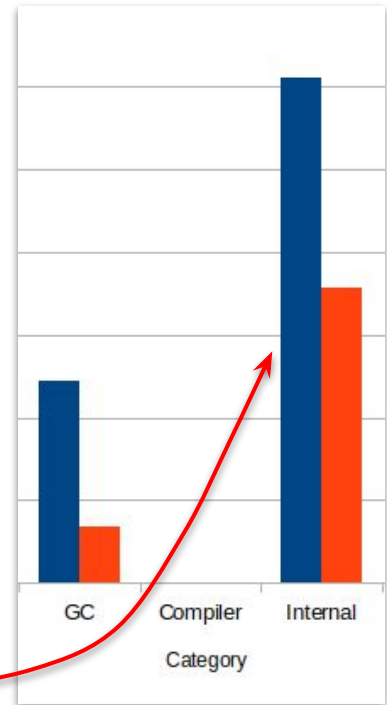
# NMT: категория OTHER (INTERNAL)

Когда категория **GC** недоговаривает:

```
[0x00007faeb7f9b5b5] BitMap::BitMap(unsigned long, bool)+0x1d5
[0x00007faeb82ca08f] OtherRegionsTable::add_reference(void*, int)+0x57f
[0x00007faeb82e0f40] InstanceClass::oop_oop_iterate_nv(oopDesc*,
  FilterOutOfRegionClosure*)+0xc0
[0x00007faeb82c3373]
HeapRegion::oops_on_card_seq_iterate_careful(MemRegion,
  FilterOutOfRegionClosure*, signed char*)+0x163
  (malloc=1154790KB type=Internal +846638KB
  #577395 +423319)
```

NMT details

≈70% 😬





# Пример деталей для INTERNAL

```
[0x00007fc009a30ab5] jni_GetByteArrayElements+0x165  
    (malloc=2263KB type=Internal +53KB  
                                           #579408 +13550)
```

# (спустя несколько дней)

```
[0x00007fc009a30ab5] jni_GetByteArrayElements+0x165  
    (malloc=282704KB type=Internal +280493KB  
                                           #72372172 +71806314)
```



# Зачем JNI в 2023 году в Java?

---

- Как послать управляемый `ping` из Java?
  - `InetAddress.isReachable(int)` — такое себе
- Готовые библиотеки не подошли
- Обёртка над rocksaw — выбранный путь

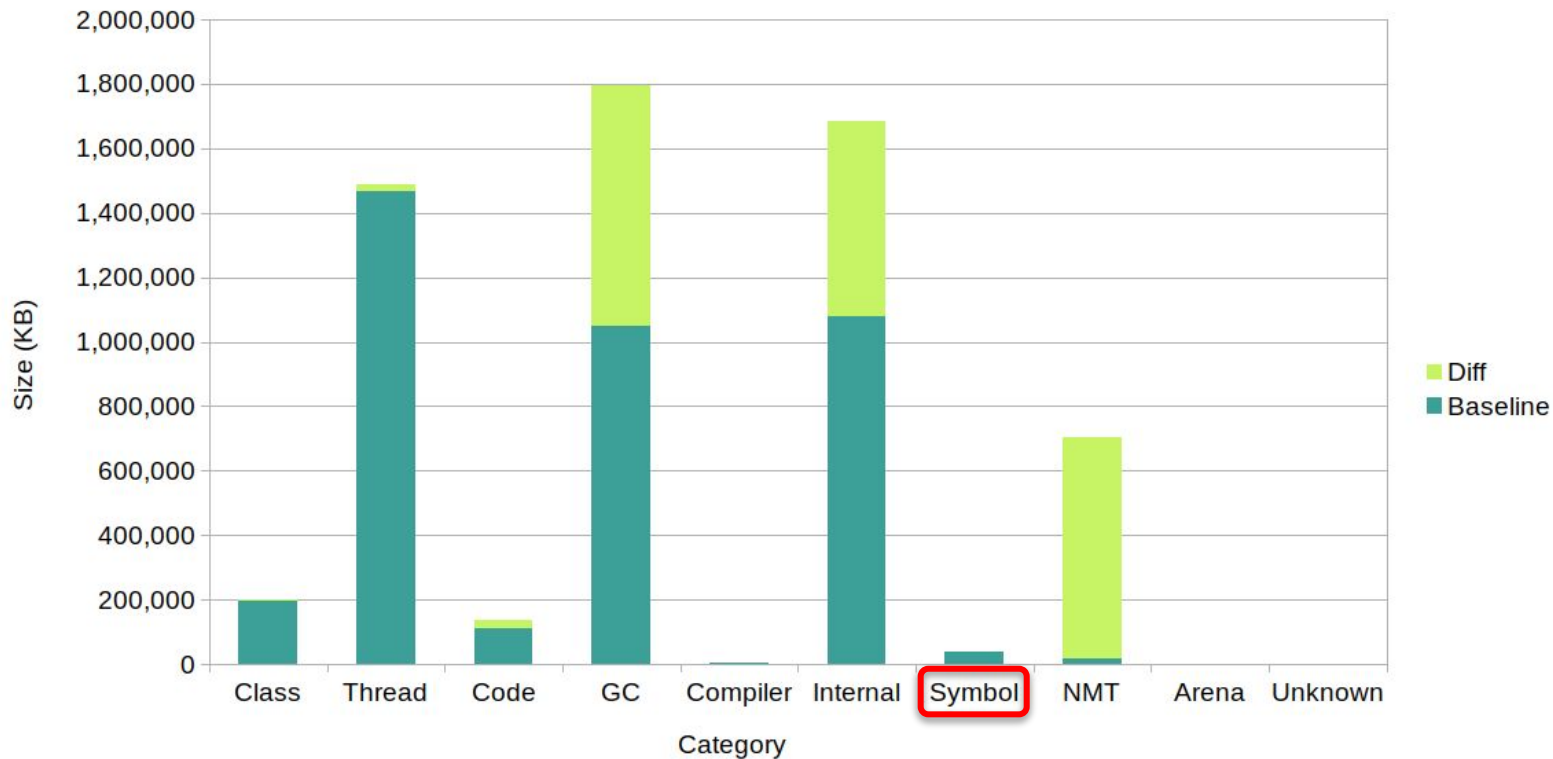
# И как это правильно ГОТОВИТЬ...

```
static void getAddr(JNIEnv *env, jbyteArray address,
                    char *addr)
{
    jbyte *buf = env->GetByteArrayElements(address, NULL);
    jint addrLen = env->GetArrayLength(address);

    int i;
    for(i = 0; i < addrLen; i++)
        // ...

    env->ReleaseByteArrayElements(address, buf, JNI_ABORT);
}
```

# Пример вывода NMT



# NMT: категория **SYMBOL**

---

- Две основные таблицы:
  - имена, сигнатуры, идентификаторы
  - ссылки на intern()'ed строки (не объекты)
- Можно посмотреть через jcmd:
  - `VM.stringtable [-verbose]`
  - `VM.symboltable [-verbose]`

# Пример фрагмента таблицы строк

---

```
$ jcmd `pgrep java` VM.stringtable -verbose
```

```
...
```

```
18: getDefaultSchemeId
```

```
80: Bundle-SymbolicName: org.apache.lucene.core
```

```
16: Asia/Novosibirsk
```

```
50: org.eclipse.update.internal.configurator.SiteEntry
```

```
11: → DYNAMIC
```

```
...
```

# Где ещё почитать о категориях

---

- [JVM Anatomy Quark #12: Native Memory Tracking](#)
- [Oracle Troubleshooting Guide – NMT](#)
- [Java using much more memory than heap size - StackOverflow](#)

# Прочие источники аллокаций







**NMT описывает  
всю нативную память**

**(нет)**

# Сравнение показаний JVM и ОС

---

```
$ jcmd `pgrep java` VM.native_memory summary | grep Total  
Total: reserved=23121490KB, committed=21934730KB
```

```
$ sudo cat /proc/`pgrep java`/status | grep RSS  
VmRSS: 22520924 kB
```

- ▣ Разница:  $22520924 - 21934730 = 586184$  KB
  - ▣ Где ещё **500+МБ** ? 🤔

# Как NMT учитывает ByteBuffers

---

- `ByteBuffer.allocateDirect()` ✓
  - учитывается в **Other** (до JDK 11 — **Internal**)
- `ByteBuffer.allocate()` ✓
  - учитывается в **Heap**
- `FileChannel.map()` ✗
  - ОПАНЬКИ

# Найти источник (через rmap)

```
$ rmap `pgrep java` | grep log
...
00007fd863000000 204800K rw-s- CommitLog-6-1690274673823.log
00007fd86f800000 204800K rw-s- CommitLog-6-1690274673822.log
...
00007fda7123a000      80K r--s- syslog.jar
...
```

```
commitlog_segment_size_in_mb: 200      # cassandra.yaml
```

# Промежуточный вывод

---

- Каждый mapped-файл отъедает нативную память по своему размеру
  - надо бороться с такими файлами



## mmap: will the mapped file be loaded into memory immediately?

Asked 9 years, 9 months ago Modified 3 years, 9 months ago Viewed 8k times

## 4 Answers

Sorted by  
Trending (recent votes count more)

No, yes, maybe. It depends.

42



Calling `mmap()` generally only means that to your application, the mapped file's contents are mapped to its address space as if the file was loaded there. Or, as if the file really existed in memory, as if they were one and the same (which includes changes being written back to disk, assuming you have write access).

No more, no less. It has no notion of loading something, nor does the application know what this means.

# Сколько на самом деле занято

---

```
$ pmap -X `pgrep java`
```

| Address      | Size   | Rss  | Mapping                       |
|--------------|--------|------|-------------------------------|
| 55d98e400000 | 4      | 4    | java                          |
| ...          |        |      |                               |
| 7fbf25e00000 | 204800 | 4    | CommitLog-6-1691993268689.Log |
| 7fbf32600000 | 2048   | 2048 |                               |
| 7fbf32800000 | 204800 | 1644 | CommitLog-6-1691993268688.Log |



# Что нужно знать об mapped buffers

---

- mapped-файл **не обязательно** занимает в памяти столько, сколько весит сам
- “Отработанные” части файла могут составлять заметную часть **buffer cache**
  - но это, как правило, не проблема
- Опасаться лучше не среднего, а **ПИКОВ**:



# Пример “частокола”, провоциро- вавшего OOM Killer



Плагин  
**Buffer Pools**  
для  
VisualVM

57



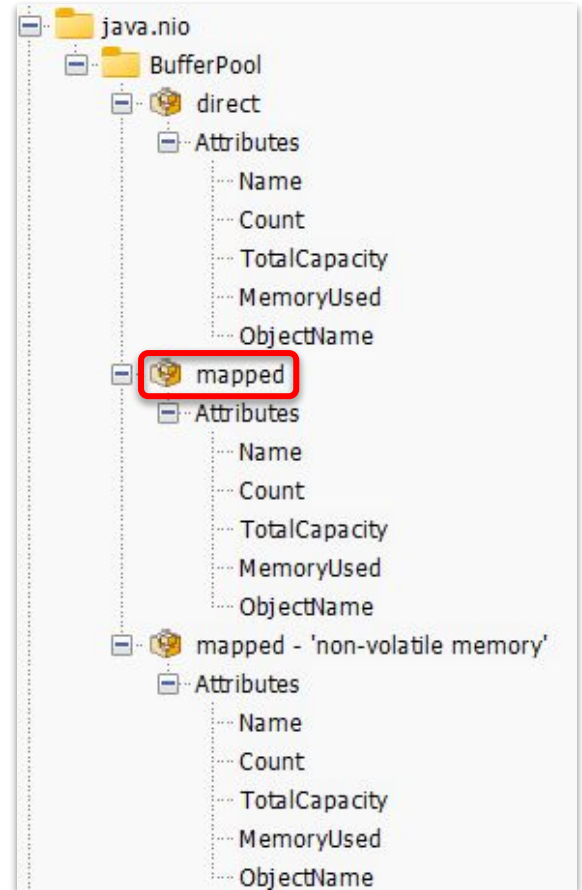
# Где ещё посмотреть

Для работы с JMX из  
консоли есть утилитка

Jmxterm:

[docs.cyclopsgroup.org/jmxterm](https://docs.cyclopsgroup.org/jmxterm)

**java.nio**  
JMX  
MBean



# Jmxterm в действии (сокращённо)

---

```
$ java -jar jmxterm-1.0.2-uber.jar
$>open 128163 ← PID
$>get -b java.nio:name=mapped,type=BufferPool *
Count = 2;
Name = mapped;
TotalCapacity = 419430400;
MemoryUsed = 419430400;
ObjectName = java.nio:type=BufferPool,name=mapped;
```

# Найти источник (через `async-profiler`)

---

1. Начать запись `mmap`-аллокаций:

```
$ sudo ./profiler.sh start -e mmap `pgrep java`
```

2. Узнать, как дела (если надо):

```
$ sudo ./profiler.sh status `pgrep java`
```

3. Сохранить текущий результат:

```
$ sudo ./profiler.sh dump -f mmap.html `pgrep java`
```



|   |  |   |
|---|--|---|
| .annobin_GI_mmap64.start  | .annobin_GI_mmap64.start   | .annobin_GI_mmap64.start  |
| sun/nio/ch/FileChannelImpl.map0   | sun/nio/ch/FileChannelImpl.map0                                      | sun/nio/ch/FileChannelImpl.map0                                     |
| sun/nio/ch/FileChannelImpl.map  | sun/nio/ch/FileChannelImpl.map                                       | sun/nio/ch/FileChannelImpl.map                                      |
| org/apache/lucene/store/MMapDirectory.map   | org/apache/lucene/store/MMapDirectory.map                            | org/apache/lucene/store/MMapDirectory.map                           |
| org/apache/lucene/store/MMapDirectory.openInput                                   | org/apache/lucene/store/MMapDirectory.openInput                      | org/apache/lucene/store/MMapDirectory.openInput                     |
| org/apache/lucene/store/FilterDirectory.openInput                                 | org/apache/lucene/store/Directory.openChe..                          | org/apache/lucene/store/Directory.openChe..                         |
| org/apache/lucene/store/FilterDirectory.openInput                                 | org/apache/lucene/store/Directory.openChe..                          | org/apache/lucene/store/Directory.openChe..                         |
| org/apache/lucene/store/FilterDirectory.openInput                                 | org/apache/lucene/store/Directory.openChe..                          | org/apache/lucene/store/Directory.openChe..                         |
| org/apache/lucene/store/FilterDirectory.openInput                                 | org/apache/lucene/store/Directory.openChe..                          | org/apache/lucene/store/Directory.openChe..                         |
| org/apache/lucene/codecs/lucene50/Lucene50CompoundFormat.write                    | org/apache/lucene/store/Directory.openChe..                          | org/apache/lucene/codecs/lucene50/Lucene50CompoundReader.r..        |
| org/apache/lucene/index/IndexWriter.createCompoundFile                            | org/apache/lucene/codecs/lucene50/Lucene5..                          | org/apache/lucene/codecs/lucene50/Lucene50CompoundReader.<init>     |
| org/apache/lucene/index/DocumentsWriterPerThread.sealFlushedSe..                  | org/apache/lucene/index/SegmentReader.<init>                         | org/apache/lucene/codecs/lucene50/Lucene50CompoundFormat.getCompo.. |
| org/apache/lucene/index/DocumentsWriterPerThread.flush                            | org/apache/lucene/index/ReadersAndUpdates.getReader                  | org/apache/lucene/index/SegmentCoreReaders.<init>                   |
| org/apache/lucene/index/DocumentsWriter.doFlush                                   | org/apache/lucene/index/BufferedUpdatesStream\$SegmentState.<init>   |   |
| org/apache/lucene/index/DocumentsWriter.flushAllThreads                           | org/apache/lucene/index/BufferedUpdatesStream.openSegmentStates      |   |
| org/apache/lucene/index/IndexWriter.prepareCommitInternal                         | org/apache/lucene/index/BufferedUpdatesStream.applyDeletesAndUpdates |   |
| org/apache/lucene/index/IndexWriter.commitInternal                                | org/apache/lucene/index/IndexWriter.applyAllDeletesAndUpdates        |   |
| org/apache/lucene/index/IndexWriter.commit  | org/apache/lucene/index/IndexWriter.maybeApplyDeletes                |   |
| com/tibbo/linkserver/plugin/context/search/ServerSearcher.commitNow               |  |   |
| com/tibbo/linkserver/plugin/context/search/ServerSearcher\$CommitTask.run         |  |   |
| java/util/concurrent/Executors\$RunnableAdapter.call                              |  |   |
| java/util/concurrent/FutureTask.run   |  |   |
| java/util/concurrent/ScheduledThreadPoolExecutor\$ScheduledFutureTask.access\$201 |  |   |
| java/util/concurrent/ScheduledThreadPoolExecutor\$ScheduledFutureTask.run         |  |   |
| java/util/concurrent/ThreadPoolExecutor.runWorker                                 |  |   |
| all   |  |   |

org/apache/lucene/index/IndexWriter.commit  
(717 samples, 22.32%)

Function: org/apache/lucene/index/IndexWriter.commit (717 samples, 22.32%)

Пример обнаружения пиков mmap-аллокаций в Apache Lucene



NMT doesn't give complete information for the memory used by the **class data sharing** (CDS) archive.

# (Application) Class Data Sharing

---

- Фича JVM для ускорения запуска и уменьшения потребления памяти

- Как проверить, включена ли:

```
$ jcmd `pgrep java` VM.info | grep CDS
```

```
CDS archive(s) not mapped
```

```
CDS: off
```

- До Java 12 по умолчанию выключена

# CDS для тех, кому за 12

---

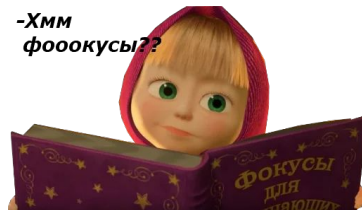
```
$ jcmd `pgrep java` VM.info | grep CDS
```

```
CDS archive(s) mapped at:
```

```
[0x0000000080000000-0x00000000800be100-0x00000000800be1000), size 12455936, SharedBaseAddress: 0x0000000080000000, ArchiveRelocationMode: 0.
```

```
CDS: on
```

-Хмм  
фооокусы??





# Выясняем источник (JDK 17)

---

```
$ pmap `pgrep java` | grep jsa
00000007ffb00000      472K rw--- classes.jsa
00000007ffc00000      520K rw--- classes.jsa
0000000800000000     4448K rw--- classes.jsa
0000000800458000     7716K r---- classes.jsa
```

```
$ du -h $JAVA_HOME/lib/server/classes.jsa
14M lib/server/classes.jsa
```

# Список классов вшит в JDK

---

```
$ cat $JAVA_HOME/lib/classlist | head -10
```

```
# NOTE: Do not modify this file.
```

```
...
```

```
java/lang/Object
```

```
java/io/Serializable
```

```
java/lang/Comparable
```

```
java/lang/CharSequence
```

```
java/lang/constant/Constable
```

Всего: **1401** класс  
(для JDK 17.0.7)

# Что нужно знать о CDS

---

- Немного **ускоряет** запуск JVM
- **Экономит** память, если в системе несколько процессов одной и той же JVM
- Со временем по памяти **не растёт**
- Можно отключить: `-Xshare:off`
- Подробнее — [в этой статье](#)



Если по NMT всё ОК, то  
OOM Killer не придёт

(нет)



The OOM killer selects  
**a task** to sacrifice  
for the sake of  
the overall system health.



# Java не всегда виновата

---

```
$ grep "invoked oom-killer" messages-20230702
kernel: dnf invoked oom-killer
kernel: new_free_mem_me invoked oom-killer
kernel: VM Thread invoked oom-killer
kernel: mem_nonheap_tes invoked oom-killer
kernel: InConsumerThrea invoked oom-killer
kernel: snmpd invoked oom-killer
kernel: MutationStage-8 invoked oom-killer
```

# Почему так?

```
1097     /**
1098     * out_of_memory - kill the "best" process when we run out of memory
1099     * @oc: pointer to struct oom_control
1100     *
1101     * If we run out of memory, we have the choice between either
1102     * killing a random task (bad), letting the system crash (worse)
1103     * OR try to be smart about which process to kill. Note that we
1104     * don't have to be perfect here, we just have to be good
1105     */
1106     bool out_of_memory(struct oom_control *oc)
1107     {
```

# Алгоритм выбора жертвы

```
226      /*
227      * The baseline for the badness score is the proportion of RAM that each
228      * task's rss, pagetable and swap space use.
229      */
230      points = get_mm_rss(p->mm) + get_mm_counter(p->mm, MM_SWAPENTS) +
231              mm_pgtables_bytes(p->mm) / PAGE_SIZE;
232      task_unlock(p);
233
234      /* Normalize to oom_score_adj units */
235      adj *= totalpages / 1000;
236      points += adj;
237
238      return points;
```



# Алгоритм выбора жертвы

---

1. Рассчитать “очки” для каждого процесса:  
`points = RSS + page_table + swap`
2. Нормализовать к 1000
3. Принять поправку на `oom_score_adj`
4. Взять процесс с максимальным `score`
5. **SIGKILL**

# И причём здесь Java?

---

```
$ ./oom-score-list.sh
PID      Score Process
988      0      irqbalance
99       0      writeback
...
992      0      chronyd
657      1      systemd-journal
269670   898    java
```



# Что можно сделать

---

- Узнать текущий oom\_score:

```
$ cat /proc/`pgrep java`/oom_score
```

- Поправить поправку (от -1000 до 1000):

```
$ sudo echo -100 > /proc/`pgrep java`/oom_score_adj
```

- Переключить поведение Killer'a:

```
$ sudo sysctl -w vm.oom_kill_allocating_task=1
```

# Java не всегда виновата

---

```
$ grep "invoked oom-killer" messages-20230702
kernel: dnf invoked oom-killer
kernel: new_free_mem_me invoked oom-killer
kernel: VM Thread invoked oom-killer
kernel: mem_nonheap_tes invoked oom-killer
kernel: InConsumerThrea invoked oom-killer
kernel: snmpd invoked oom-killer
kernel: MutationStage-8 invoked oom-killer
```

# Java не всегда виновата

---

```
$ grep "invoked oom-killer" messages-20230702
kernel: dnf invoked oom-killer
kernel: new_free_mem_me invoked oom-killer
kernel: VM Thread invoked oom-killer
kernel: mem_nonheap_tes invoked oom-killer
kernel: InConsumerThrea invoked oom-killer
kernel: snmpd invoked oom-killer
kernel: MutationStage-8 invoked oom-killer
```



# Если Killer'а всё-таки вызвала java

---

- В логе ядра будет виден поток, а не процесс
- Название потока обрезается на 15 символах
- PID потока не равен PID процесса java
- Стектрейс не отражает состояние потока
- Подробнее — [здесь](https://serverfault.com) (serverfault.com)

# А если поменять аллокатор памяти?

---

- Функция `malloc()` в \*nix ОС может иметь разные имплементации
- Их можно “подкладывать” на этапах:
  - компиляции
  - сборки/линковки
  - в runtime (подходит для JVM)

# Что не так с обычным malloc? (glibc)

```
$ pmap `pgrep java` | grep -B 1 " 65.*K .* [ anon ]"
00007f637c000000      132K rw---   [ anon ]
00007f637c021000  65404K ----- [ anon ]
00007f6380000000      132K rw---   [ anon ]
00007f6380021000  65404K ----- [ anon ]
...

```

} ≈100 штук

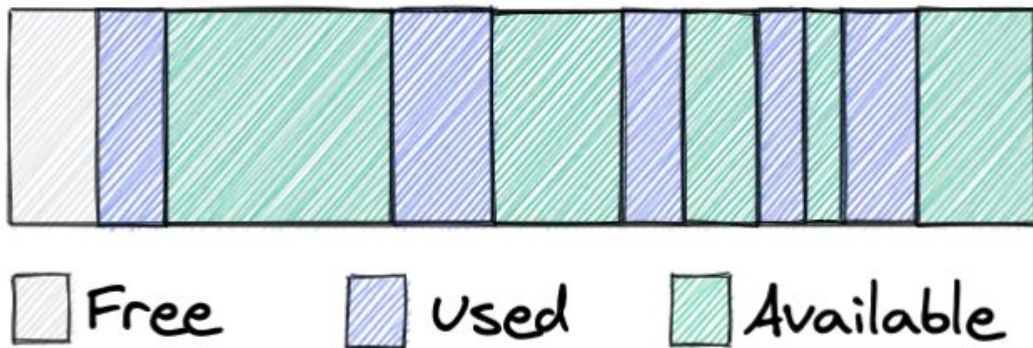
```
$ cat /proc/`pgrep java`/status | grep RssAnon
RssAnon:      1416720 kB
```



# Что не так с обычным malloc? (glibc)

---

- Может сильно фрагментировать память,
  - особенно в **многопоточных** приложениях



# Альтернатива — Jemalloc

---

- Открытый аллокатор родом из FreeBSD (2005)
- Заточен под **многопоточные** приложения
- Имеет встроенные средства **отладки**
- Применяется в Facebook, Firefox, Mac OS X, ...

# Как подключить Jemalloc к JVM

---

1. Скачать и собрать
2. Выставить переменную окружения:  
`export LD_PRELOAD=/path/to/libjemalloc.so`
3. Выставить переменную окружения:  
`export MALLOC_CONF=prof:true,...` (опционально)
4. (пере)Запустить JVM

# Jetty в действии

---

```
$ pmap `pgrep java` | grep -B 1 " 500K .* [ anon ]"  
00007fbefd280000      12K ----- [ anon ]  
00007fbefd283000    500K rw---- [ anon ]  
00007fbefd300000      12K ----- [ anon ]  
00007fbefd303000    500K rw---- [ anon ]
```

...

```
$ cat /proc/`pgrep java`/status | grep RssAnon  
RssAnon:    1296356 kB
```

# А что в контейнерах?

---

- Нужно учитывать лимит на cgroup:  
`memory.limit_in_bytes`
- Для Docker контейнеров есть опции  
`--oom-kill-disable`, `--memory-swap`, `--memory-swappiness`
- В JVM Buildpack NMT включен по умолчанию

# Как отключить NMT в JVM Buildpack

---

```
tasks.named<BootBuildImage>("bootBuildImage") {  
    environment.set(mapOf(  
        "BPE_DEFAULT_BPL_JAVA_NMT_ENABLED" to "false",  
    ))  
    // ...  
}
```

build.gradle.kts

# Java Buildpack Memory Calculator

---

- ▣ Вычисляет опции JVM для памяти
- ▣ Опирается на формулу:  
$$Xmx = \text{total memory} - (\text{headroom amount} + \text{direct memory} + \text{metaspace} + \text{reserved code cache} + (\text{thread stack} * \text{thread count}))$$
- ▣ Включает полезные умолчательные значения
- ▣ Позволяет себя тюнить

**Выводы**  
**Резюме**  
**Ссылки**

Ради чего  
всё это  
продолжалось





# Takeaways

---

## Предотвращение

- ❑ Делать запас памяти вне кучи (от 10%)
- ❑ Проверять сторонние либы на работу вне кучи

## Анализ

- ❑ NMT (jcmd)
- ❑ async-profiler
- ❑ pmap/top
- ❑ JMX/JFR

## Устранение

- ❑ Фиксировать утечки
- ❑ Сокращать ненужные кэши
- ❑ Использовать jemalloc

# Резюме

---

- Общего лимита нативной памяти **нет**
- Если пришёл OOM Killer — включай **NMT**
- Работу Killer'а надо не настраивать, а предотвращать



# Спасибо!

## Вопросы?

---

Владимир Плизгá

 [Toparvion](#)

 [toparvion.pro](#)

 [stegotext.ru](#)



<https://t.me/StegoTrend>

 [слайды](#) 



# Credits

---

Special thanks to all the people who made and released these awesome resources for free:

- ▣ Presentation template by SlidesCarnival
- ▣ Sticker packs by:
  - ▣ [https://t.me/addstickers/masha\\_mishka](https://t.me/addstickers/masha_mishka)
  - ▣ <https://t.me/addstickers/MASHAANDTHEBEARDIGE44>