

# Путеводитель по анализу памяти JVM

---

Владимир Плизга, Tibbo Systems

Joker, Петербург, октябрь 2024

- Я – Владимир Плизга
- Пишу на Java с 2011 г  
(финтех, IIoT)
- Люблю помогать людям  
(особенно разработчикам)





“Привет! Слушай тут чето ваш релиз не взлетел, OutOfMemory кидает. Давай вы щас быстро гляните, а то нам откатывать надо, пока бизнес не пришел...”

---

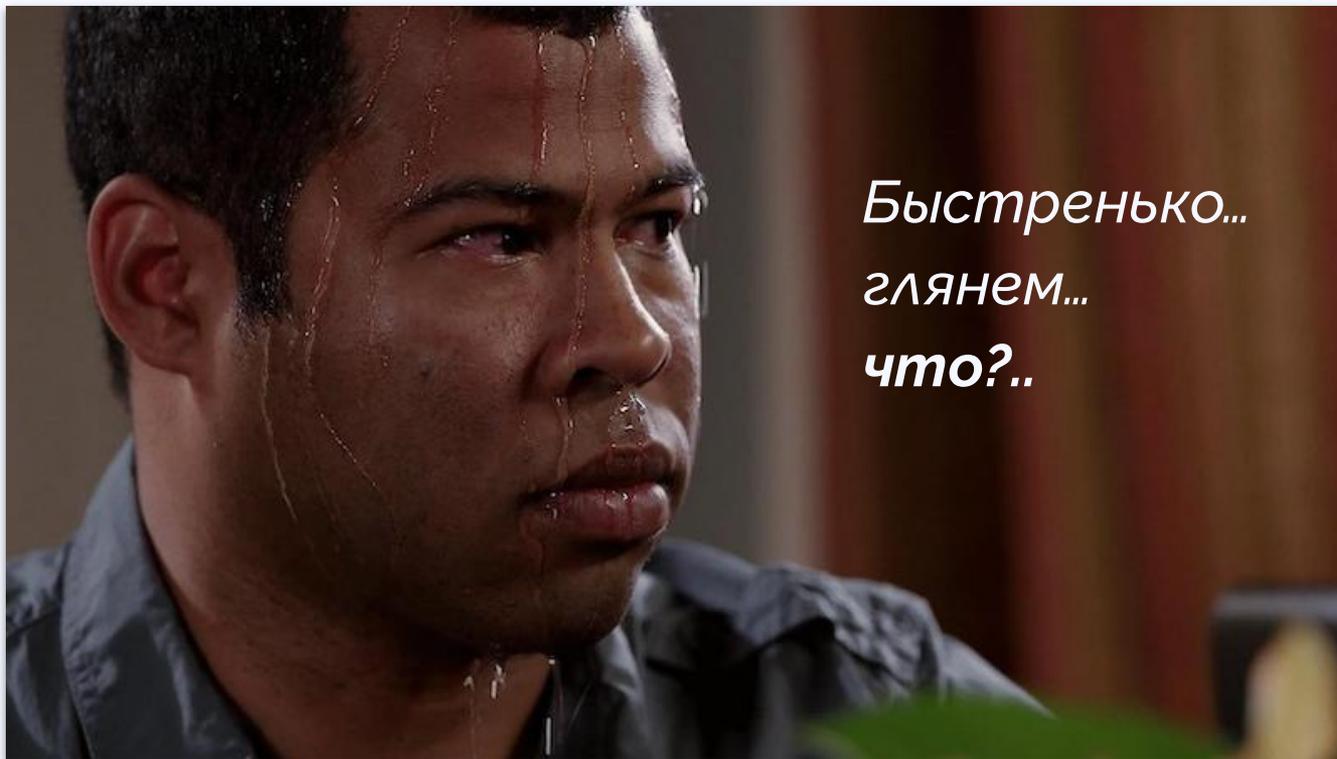
*Из рабочего чата*



“Привет! Слушай тут чето ваш релиз  
не взлетел, **OutOfMemory** кидает.  
Давай вы щас **быстро гляните**,  
а то нам **откатывать** надо,  
пока бизнес не пришел...”

---

*Из рабочего чата*



*Быстренько...  
глянем...  
что?..*

**Снять диагностику  
Как?**

**Посмотреть память  
Чем?**

**Нужны дампы  
Какие?**

**Найти утечку  
Где?**

**Хорошо бы спросить  
Кого?**

# План конспекта

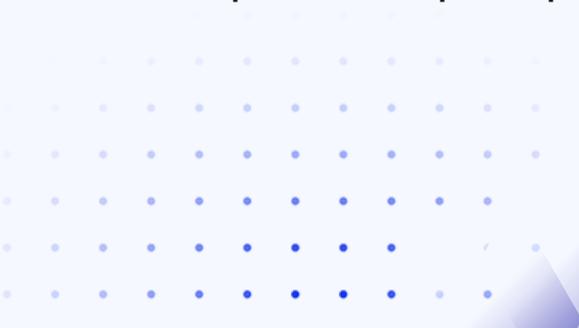




# Погружение

---

Через лабораторный кейс



# Подопытный кролик

## Spring Pet Clinic REST

- Демо-приложение на Spring Boot
- CRUD-операции на WebMVC
- Встроенная БД (hsqldb)
- OpenAPI (Swagger UI)
- ◆ вместо фронта на Angular



# Фича: дИИагностика

- Новый метод в REST API
- Принимает petId и строку симптомов
- Общается с нейросетью (якобы)
- Возвращает краткую сводку:
  - ◆ Код диагноза
  - ◆ Рекомендуемое лекарство
  - ◆ Дату следующего визита

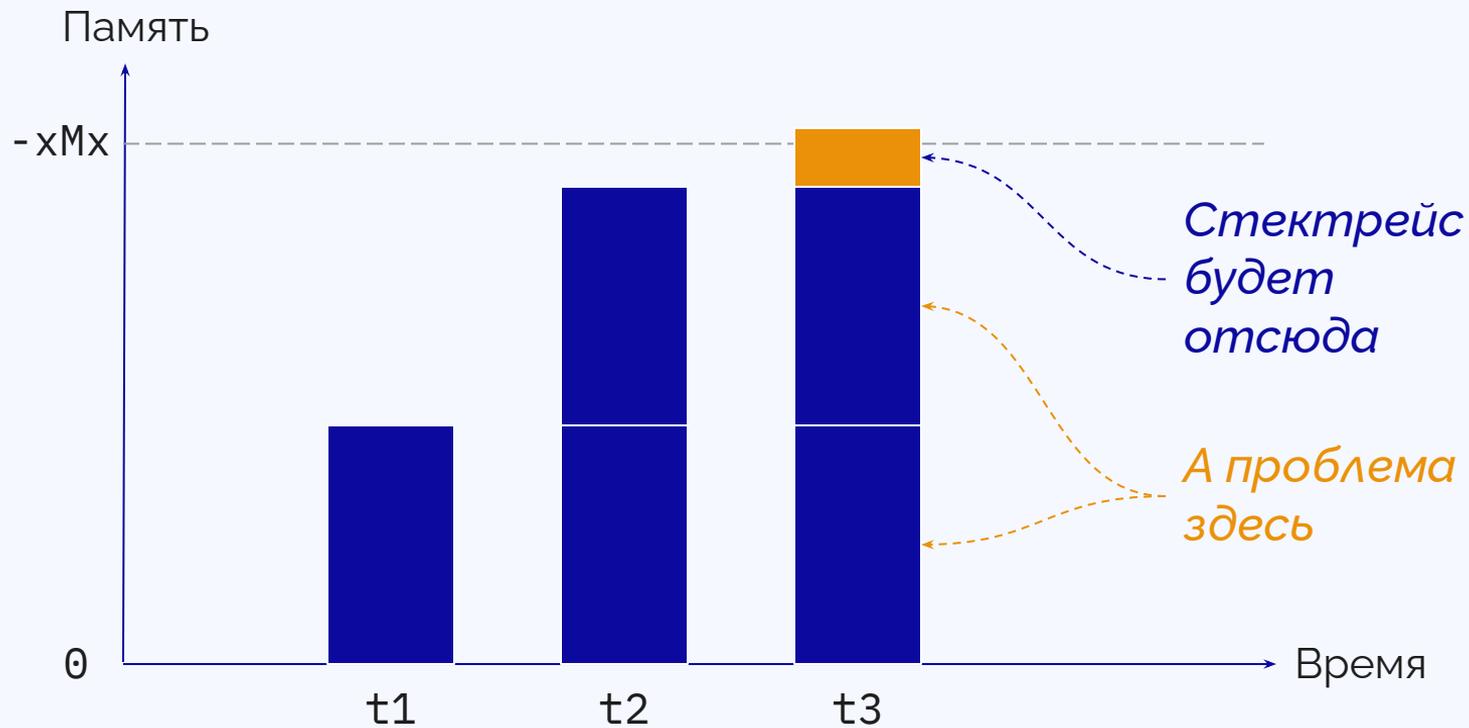
<https://github.com/Toparvion/spring-petclinic-rest>

# Гипотезы о причинах OOM

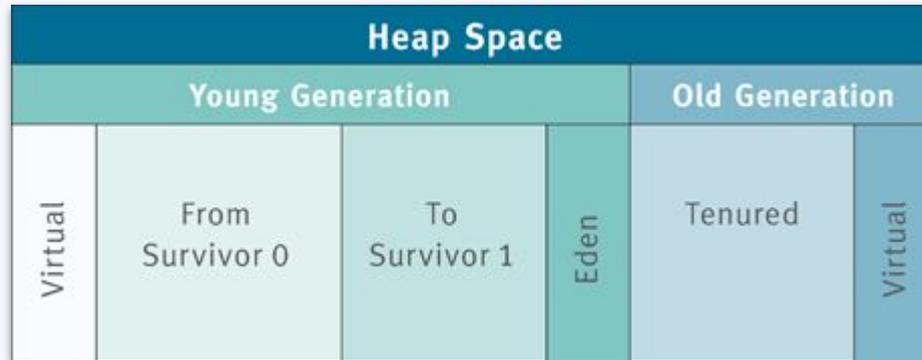
- ~~Криворукий программист~~
- Раздулся кэш summaries
- Распухли строки в Summary
- Выросло число объектов Summary
- *[прочее]*



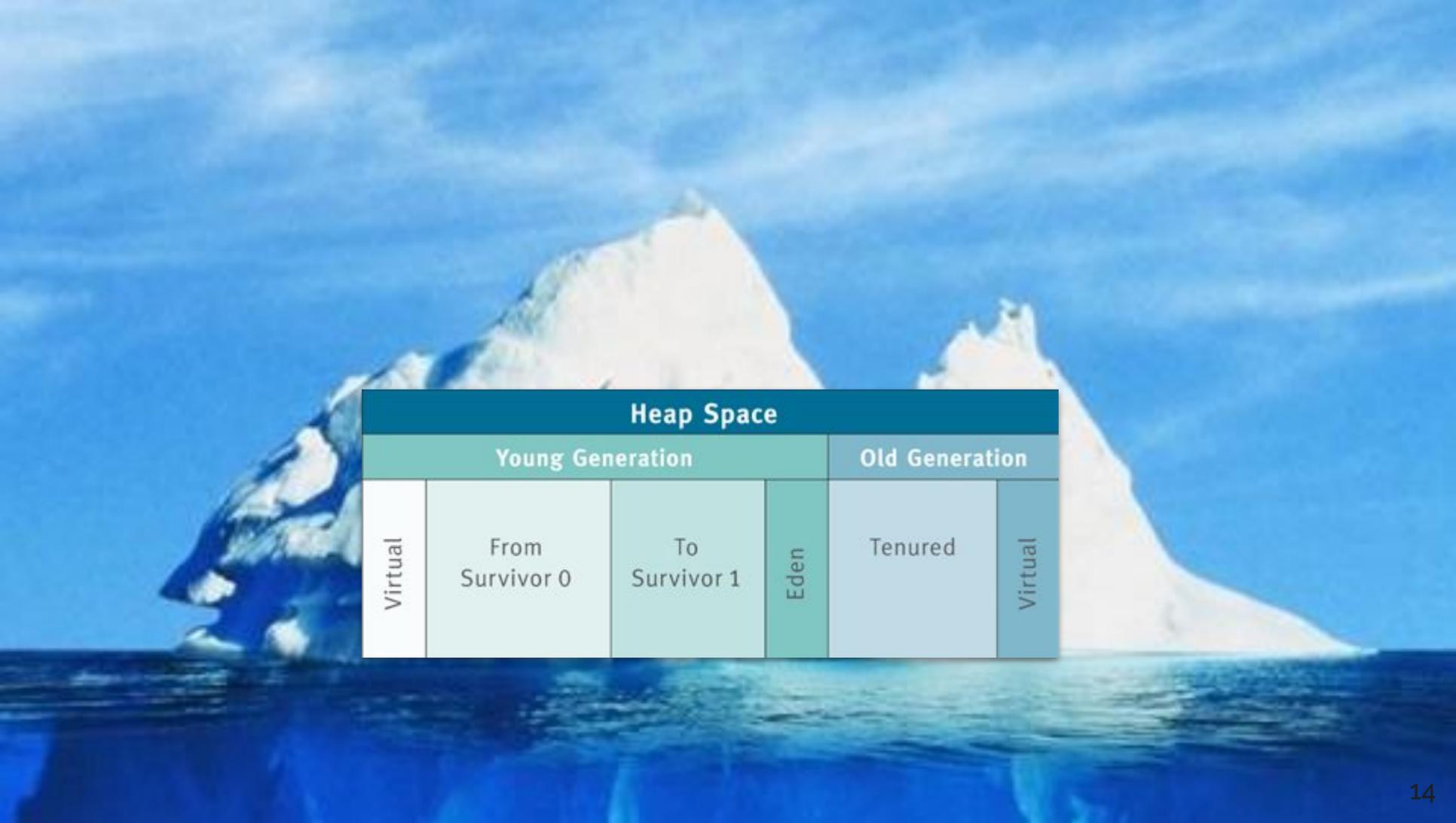
# Почему стектрейс OOM не важен



# Что будем смотреть

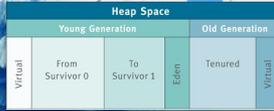


<https://www.linkedin.com/pulse/java-virtual-machine-changes-78-9-kunal-saxena>

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



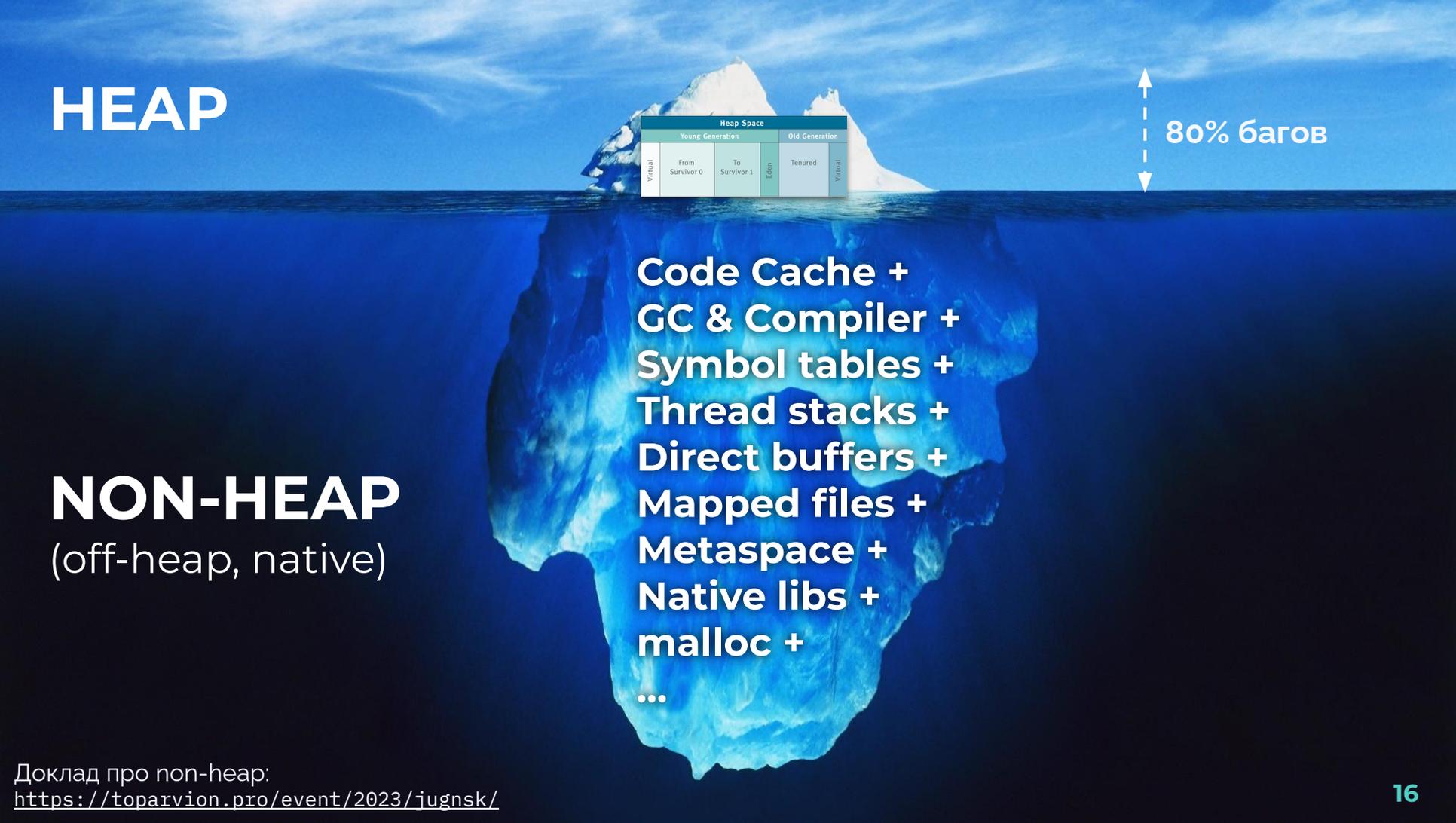
# NON-HEAP

(off-heap, native)

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

...

# HEAP



The image features a large iceberg floating in a blue ocean. The tip of the iceberg, which is above the water line, is labeled 'HEAP'. On this tip, there is a small table titled 'Heap Space' with columns for 'Young Generation' and 'Old Generation'. The 'Young Generation' is further divided into 'From Survivor 0', 'To Survivor 1', and 'Eden'. The 'Old Generation' is labeled 'Tenured'. The sides of the iceberg are labeled 'Virtual'. The much larger part of the iceberg is submerged below the water line and is labeled 'NON-HEAP (off-heap, native)'. This submerged part contains a list of components: 'Code Cache + GC & Compiler + Symbol tables + Thread stacks + Direct buffers + Mapped files + Metaspace + Native libs + malloc + ...'. To the right of the iceberg, a vertical dashed white arrow points upwards from the water level to the top of the iceberg, with the text '80% багов' (80% bugs) next to it, indicating that most bugs occur in the non-heap area.

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

80% багов

# NON-HEAP

(off-heap, native)

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

...



# Heap Dump

---

Основной источник информации для анализа

# Что есть heap dump?

- Снимок графа объектов в куче в какой-то момент
  - ◆ т.е. содержит классы приложения и библиотек
- Делается силами Java-машины
  - ◆ значит, она должна быть жива
- Сохраняется в двоичном формате HPROF
  - ◆ при работе на HotSpot® JVM



# Способы снятия дампа



## По запросу

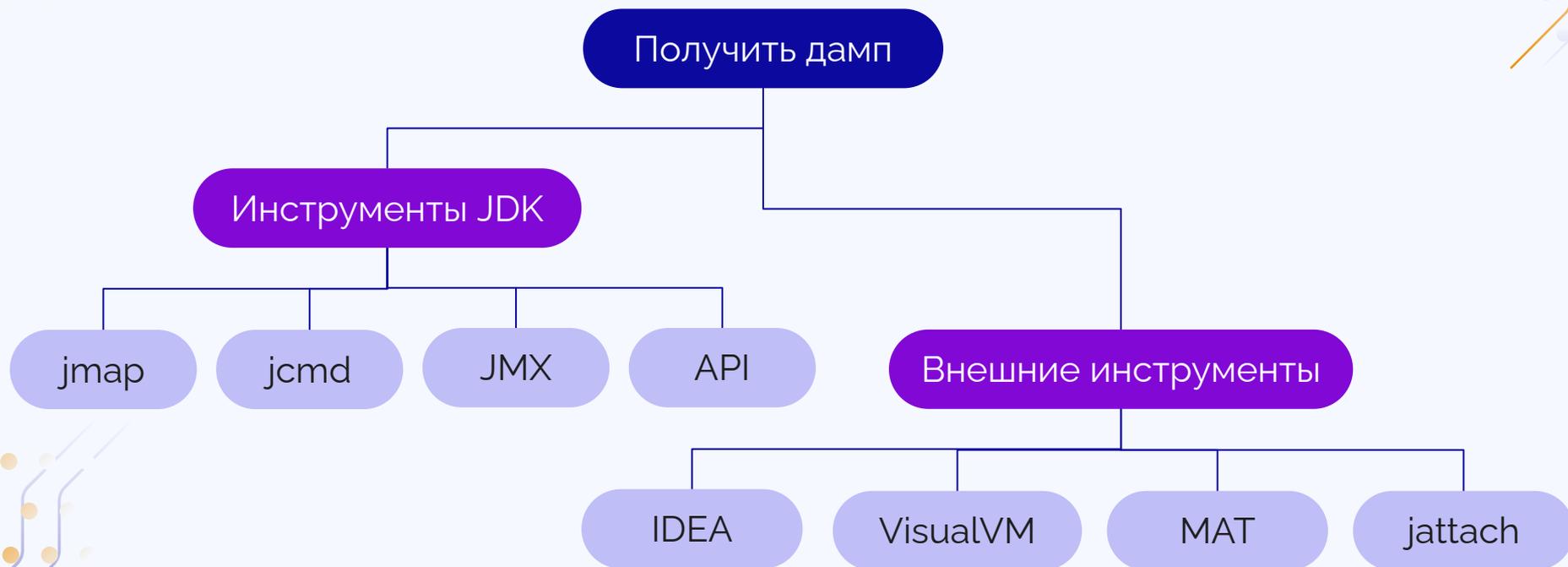
Когда есть проблема  
или её предчувствие



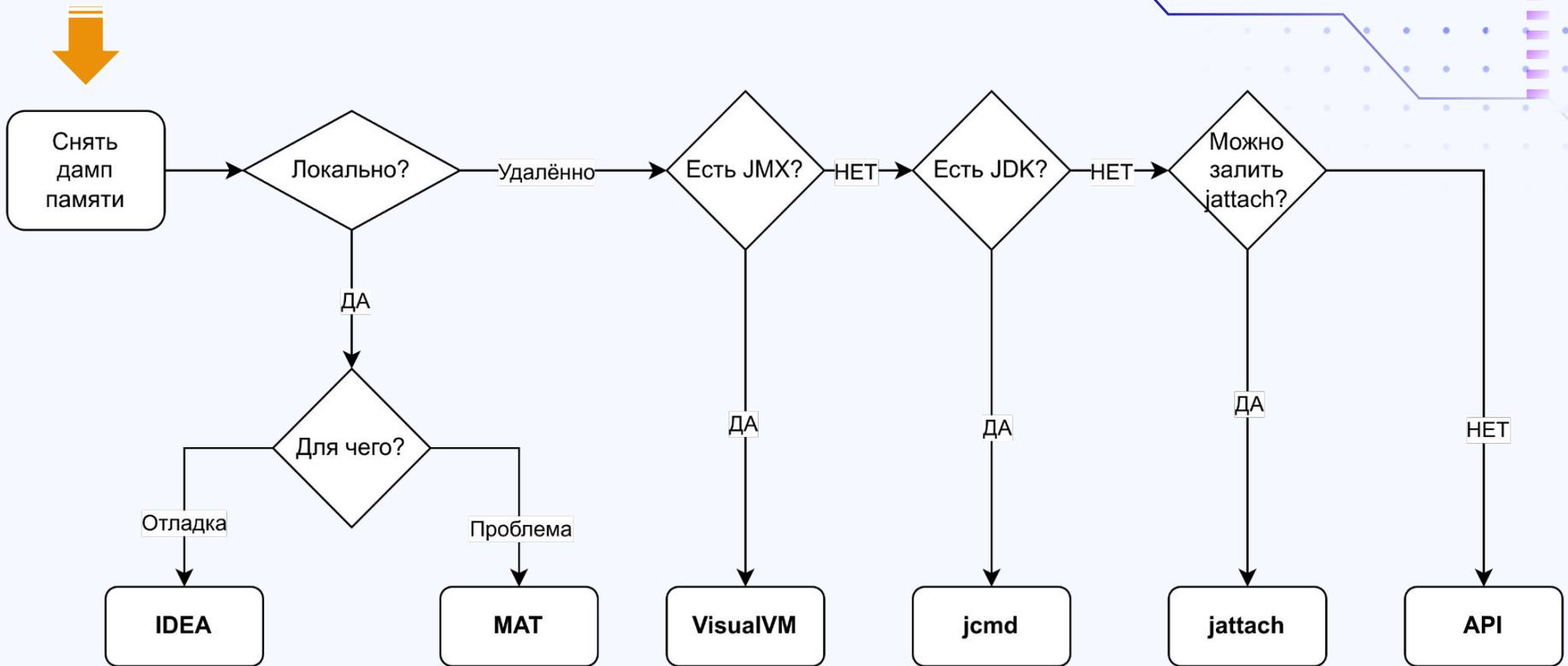
## Автоматически

Когда случается  
`OutOfMemoryError`

# Снятие дампов по запросу



[Список не исчерпывающий]



Примерный порядок выбора инструмента  
для снятия дампа по запросу

# Снятие дампов **автоматически**

- Через JVM-опцию `-XX:+HeapDumpOnOutOfMemoryError`
- По умолчанию сохраняет в `./java_pid<pid>.hprof`
  - ◆ но можно поменять: `-XX:HeapDumpPath=path`
  - ◆ `%p` в пути заменяется на PID процесса
- Срабатывает при любом OOM (они бывают разными)

# Предостережение 1

Снятие дампа может **надолго заморозить** приложение (до минут)

Можно *ускорить* за счёт:

- Отключения FullGC
- Включения сжатия
- Снятия core dump (см. далее)



## Предостережение 2

Вместе с дампом  
могут сохраниться

**конфиденциальные  
данные**

Можно его обфусцировать:

- Штукой от PayPal
- Экспортом из Eclipse MAT





# Терминология

---

Вокруг кучи и её дампа

# GC Roots

- **Отправные точки** для сборщика мусора (GC)
- Не удаляются сборщиком
- Не дают удалять свои **зависимые** объекты
  - ◆ Зависимость определяется:
    - Достижимостью (*см. далее*)
    - Типом ссылки: **hard/soft/weak/phantom**



# Разновидности GC Roots

## Используемые классы

И объявленные в них  
статические переменные

## Локальные переменные

В том числе  
аргументы методов

## Активные потоки

Не достигшие состояния  
TERMINATED

## Мониторы синхронизации

Пока они кем-то  
захвачены

другие



# Размеры объектов в дампе

*приблизительные...*





“Most of the HPROF-based tools  
**have problems** with deducing  
the actual instance footprint ...  
which can lead the analysis  
in the **wrong direction.**”

---

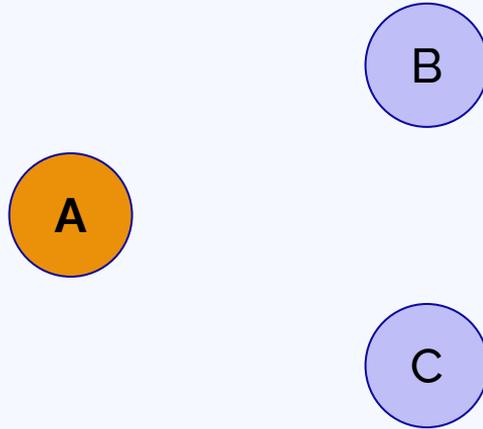
– <https://shipilev.net/blog/2014/heapdump-is-a-lie/>

# Размеры объектов в дампе

- **Shallow** size – собственный размер объекта
  - ◆ без учёта размера зависимых объектов
- **Retained** size – размер объекта и всех **ПОЛНОСТЬЮ** зависимых объектов
  - ◆ т.е. объём памяти, который освободится, если удалить этот объект вместе с **его** поддеревом

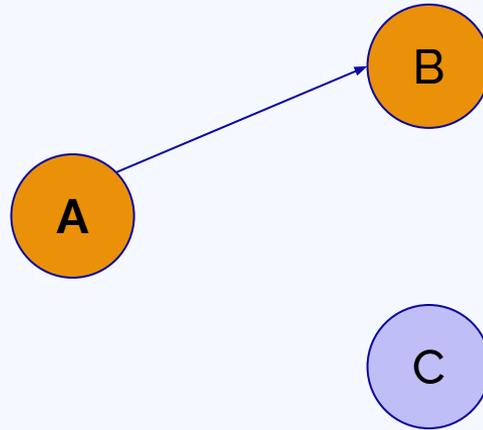


# Различия между `shallow` и `retained`



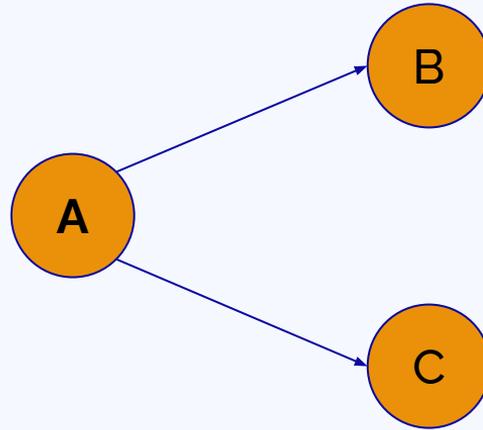
```
retained(A) = shallow(A)
```

# Различия между `shallow` и `retained`



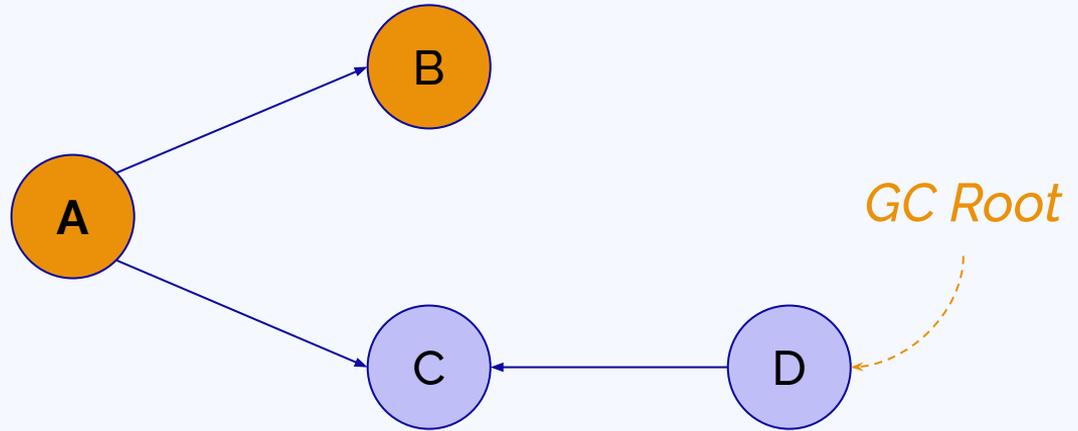
$$\begin{aligned} \text{retained}(A) &= \text{shallow}(A) + \text{shallow}(B) \\ &= \text{shallow}(A) + \text{retained}(B) \end{aligned}$$

# Различия между `shallow` и `retained`



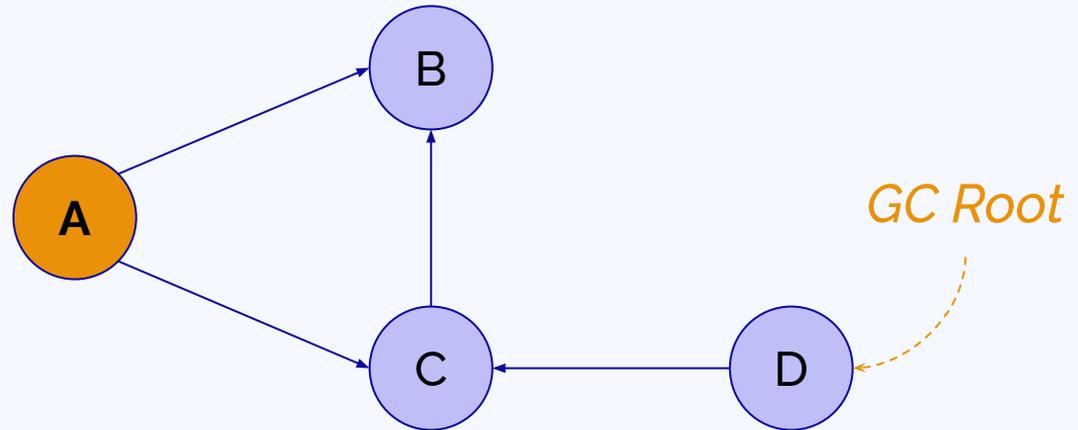
`retained(A) = shallow(A) + shallow(B) + shallow(C)`

# Различия между shallow и retained



$\text{retained}(A) = \text{shallow}(A) + \text{shallow}(B) + \text{shallow}(C)$

# Различия между `shallow` и `retained`



```
retained(A) = shallow(A) + shallow(B) + shallow(C)
```



**Небось опять  
о своих бабах  
думает**

**Почему в колонке  
retained сумма  
вообще не бьётся?**

# Размышления вслух

## Вопрос

Если простой сумме размеров полей нельзя верить, то как увидеть “чистый” состав объекта с т.з. retained size?

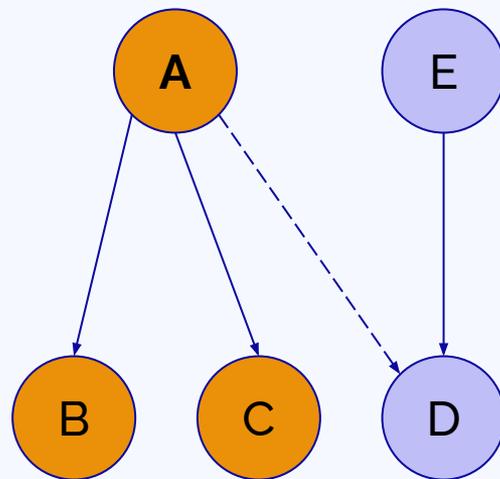
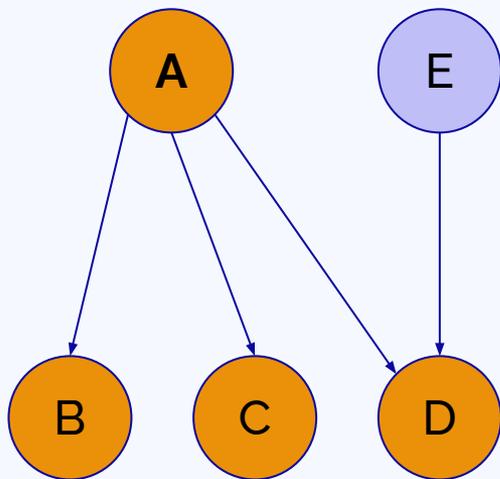
## Ответ

Нужно ввести такое **представление графа**, в котором для рассматриваемого объекта будут видны только те его поля, которые **вносят вклад** в retained size.

# Знакомтесь: Dominator Tree

- Разновидность трансформации графа объектов
- **Цель**: подсветить объекты, от которых больше всего зависят retained-размеры других
- **Упрощённо**: отфильтрованный граф (дерево), в котором видны только ссылки, входящие в retained size

# Object Graph **vs** Dominator Tree



В графе объектов есть **все поля** всех классов.

В дереве доминаторов нет полей, на объекты которых  
ссылается **кто-то кроме текущего** объекта.

# И зачем это всё?

## GC Roots

Чтобы **понимать**, на чём “держится” куча

## Retained size

Чтобы **оценивать** “влияние” объектов на размер кучи

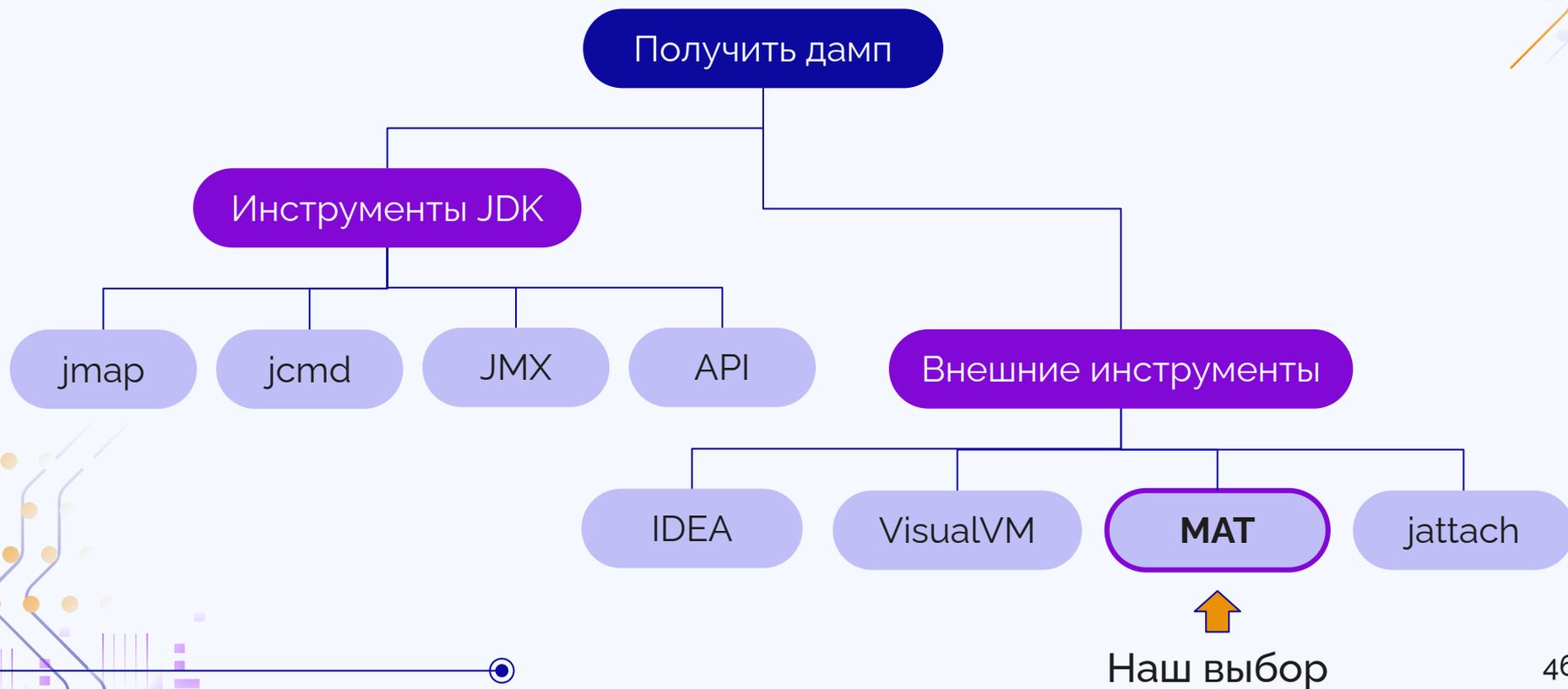
## Типы ссылок

Чтобы **различать** “жесткость” связей между объектами

## Dominator tree

Чтобы **видеть** “чистую” структуру зависимостей

# Инструменты для снятия дампов





# Eclipse MAT

---

Основной инструмент для анализа дампов памяти

# Eclipse MAT

- Полное имя: Eclipse **M**emory **A**nalyzing **T**ool
  - ◆ построен на Eclipse Equinox Platform
  - ◆ <https://eclipse.dev/mat/downloads.php>
- Поддерживает разные способы **СНЯТИЯ** дампов
- Для **анализа** требует предварительного парсинга
  - ◆ в помощь есть скрипт ParseHeapDump

## Acquire Heap Dump Dialog

Acquire a heap dump from a locally running Java process.

Choose a local process:

Description	PID	Heap Dump Provider
sun socket : com.intellij.idea.Main	16117	Attach API
sun socket : /home/vladimir/work/tools/mat//plu	46195	Attach API
sun socket : org.jetbrains.jps.cmdline.Launcher /s	94849	Attach API
sun socket : com.intellij.idea.Main	16117	Attach API using a helper JVM
sun socket : /home/vladimir/work/tools/mat//plu	46195	Attach API using a helper JVM
sun socket : org.jetbrains.jps.cmdline.Launcher /s	94849	Attach API using a helper JVM
com.intellij.idea.Main	16117	HPROF jmap dump provider
/home/vladimir/work/tools/mat//plugins/org.ecl	46195	HPROF jmap dump provider
org.jetbrains.jps.cmdline.Launcher /snap/intellij-i	94849	HPROF jmap dump provider

Refresh

Configure...

Начни с этого

“Механизмы”  
СНЯТИЯ

Снятие дампа локально через Eclipse MAT

# Навигация по графу объектов

Class Name	
 	<Regex>
▼ 	java.util.concurrent.ThreadPoolExecutor @ 0x52ae5cb10
▼  <b>workQueue</b>	java.util.concurrent.LinkedBlockingQueue @ 0x52ae5cb60

# Навигация по графу объектов

Class Name	
 	<Regex>
▼ 	java.util.concurrent.ThreadPoolExecutor @ 0x52ae5cb10
▼ 	<b>workQueue</b> java.util.concurrent.LinkedBlockingQueue @ 0x52ae5cb60
▼ 	<b>head</b> java.util.concurrent.LinkedBlockingQueue\$Node @ 0x9153af210
▼ 	<b>next</b> java.util.concurrent.LinkedBlockingQueue\$Node @ 0x9153af230

# Навигация по графу объектов

Class Name
  <Regex>
▼  java.util.concurrent.ThreadPoolExecutor @ 0x52ae5cb10
▼  <b>workQueue</b> java.util.concurrent.LinkedBlockingQueue @ 0x52ae5cb60
▼  <b>head</b> java.util.concurrent.LinkedBlockingQueue\$Node @ 0x9153af210
▼  <b>next</b> java.util.concurrent.LinkedBlockingQueue\$Node @ 0x9153af230
▼  <b>next</b> java.util.concurrent.LinkedBlockingQueue\$Node @ 0x9153af250
▼  <b>next</b> java.util.concurrent.LinkedBlockingQueue\$Node @ 0x9153af270
▼  <b>next</b> java.util.concurrent.LinkedBlockingQueue\$Node @ 0x9153af290
▼  <b>next</b> java.util.concurrent.LinkedBlockingQueue\$Node @ 0x9153af2b0

# Навигация по графу объектов

Class Name
<Regex>
java.util.concurrent.ThreadPoolExecutor @ 0x52ae5cb10
workQueue java.util.concurrent.LinkedBlockingQueue @ 0x52ae5cb60
head java.util.concurrent.LinkedBlockingQueue\$Node @ 0x9153af210
next java.util.concurrent.LinkedBlockingQueue\$Node @ 0x9153af230
next java.util.concurrent.LinkedBlockingQueue\$Node @ 0x9153af250
next java.util.concurrent.LinkedBlockingQueue\$Node @ 0x9153af270
next java.util.concurrent.LinkedBlockingQueue\$Node @ 0x9153af290
next java.util.concurrent.LinkedBlockingQueue\$Node @ 0x9153af2b0
next java.util.concurrent.LinkedBlockingQueue\$Node @ 0x9153af2d0
next java.util.concurrent.LinkedBlockingQueue\$Node @ 0x9153af2f0
next java.util.concurrent.LinkedBlockingQueue\$Node @ 0x9153afaa0
next java.util.concurrent.LinkedBlockingQueue\$Node @ 0x9153b0180
next java.util.concurrent.LinkedBlockingQueue\$Node @ 0x9153b01a0





# Основные разделы Eclipse MAT

## Overview

Схоже с Top Consumers

## Dominator Tree

Знакомый термин, да?

## Histogram

Просто и ясно

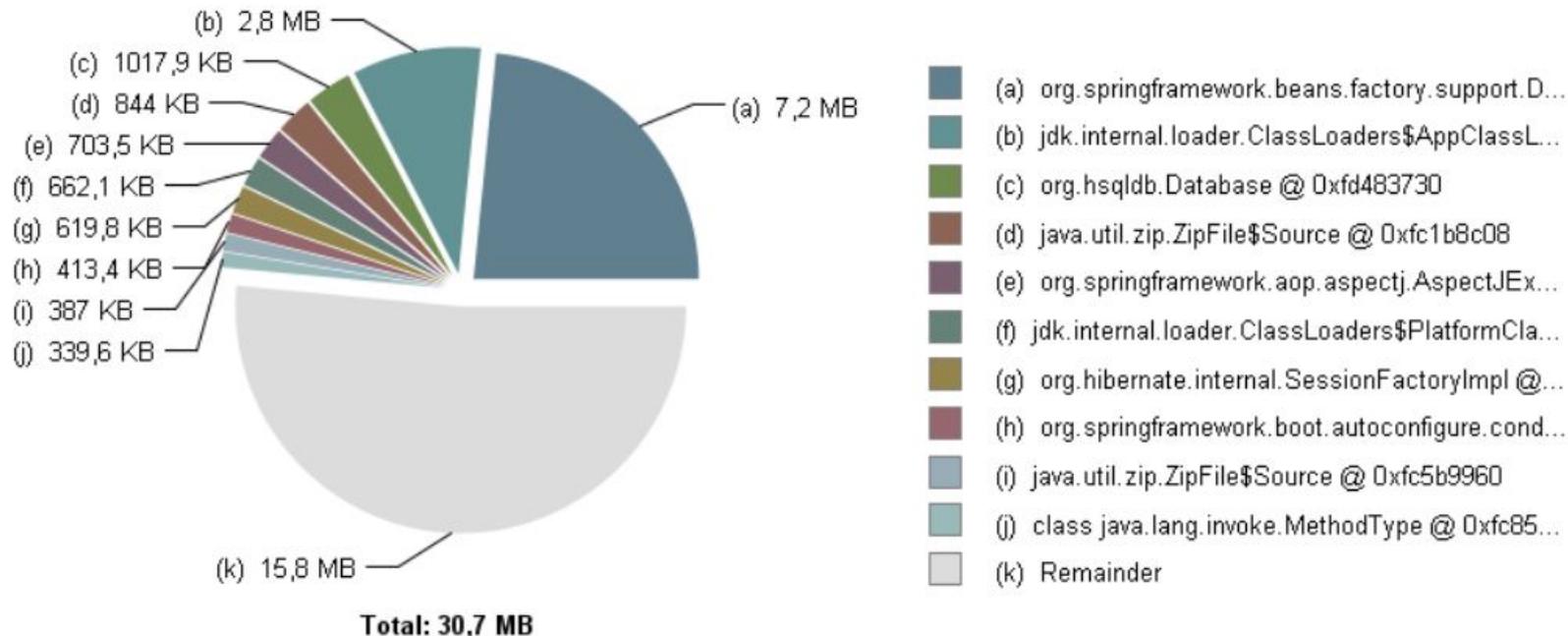
## Threads

Не только состояния



# Top Consumers

## ▼ Biggest Objects (Overview)



Top Consumers – разбивка объектов в куче по retained size

Class Name	Shallow Heap <Numeric>	Retained Heap <Numeric>	Percentage <Numeric>
> <Regex>			
> org.springframework.beans.factory.support.DefaultListableBeanFactory @ 0xfcc001e0	224	7 523 176	23,41 %
> jdk.internal.loader.ClassLoaders\$AppClassLoader @ 0xfc600468 <b>JNI Global</b>	96	2 971 528	9,24 %
> org.hsqldb.Database @ 0xfd483730	160	1 042 344	3,24 %
> java.util.zip.ZipFile\$Source @ 0xfc1b8c08	80	864 208	2,69 %
> org.springframework.aop.aspectj.AspectJExpressionPointcut @ 0xfd03da68	48	720 400	2,24 %
> jdk.internal.loader.ClassLoaders\$PlatformClassLoader @ 0xfc6004d0 <b>JNI Global</b>	96	677 952	2,11 %
> org.hibernate.internal.SessionFactoryImpl @ 0xfdaa2608	120	634 672	1,97 %
> org.springframework.boot.autoconfigure.condition.ConditionEvaluationReport @ 0xfccb47a0	32	423 344	1,32 %
> java.util.zip.ZipFile\$Source @ 0xfc5b9960	80	396 336	1,23 %
> class java.lang.invoke.MethodType @ 0xfc855aa8 <b>System Class</b>	48	347 776	1,08 %
> java.util.zip.ZipFile\$Source @ 0xfc2966b0	80	226 128	0,70 %
> class sun.util.locale.BaseLocale\$Cache @ 0xfc559d68 <b>System Class</b>	8	209 864	0,65 %

**GC Roots** (indicated by arrows pointing to the highlighted entries)

→ **Dominator Tree** – более общее представление Top Consumers

◆ Может иметь группировку по классам, пакетам и classloader'ам

Histogram ×

Class Name	Objects	Shallow Heap	Retained Heap
<code>*petclinic.[\w.]+</code> Можно фильтровать регулярками	<Numeric>	<Numeric>	<Numeric>
org.springframework.samples.petclinic.repository.jdbc.JdbcPet	13	624	>= 1 288
org.springframework.samples.petclinic.model.Owner	10	400	>= 456
org.springframework.samples.petclinic.model.PetType	6	144	>= 144
org.springframework.samples.petclinic.rest.controller.PetRestController	1	40	>= 88
org.springframework.samples.petclinic.service.ClinicServiceImpl	1	40	>= 40
org.springframework.samples.petclinic.service.perf.memory.PortfolioService	1	32	>= 2 496
org.springframework.samples.petclinic.rest.controller.OwnerRestController	1	32	>= 88
org.springframework.samples.petclinic.repository.jdbc.JdbcPetRepositoryImpl	1	32	>= 4 088
org.springframework.samples.petclinic.rest.controller.VetRestController	1	32	>= 88
org.springframework.samples.petclinic.rest.controller.VisitRestController	1	24	>= 56
org.springframework.samples.petclinic.rest.controller.UserRestController	1	24	>= 24

→ **Histogram** – общая разбивка объектов по классам

- ◆ Удобно для быстрой проверки наличия и числа объектов
- ◆ Может применяться к другим представлениям (Show as histogram)

Object / Stack Frame	Name	Shallow Heap	Retained Heap	Max. Locals' Retained Heap
<Regex>	.*logback-2.*	<Numeric>	<Numeric>	<Numeric>
java.lang.Thread @ 0xfc8a59c8	logback-2	104	216	
at jdk.internal.misc.Unsafe.park(Z)V (Unsafe.java(Native Method))				
at java.util.concurrent.locks.LockSupport.park()V (LockSupport.java:371)				
> at java.util.concurrent.locks.AbstractQueuedSynchronizer\$ConditionNode				32
> at java.util.concurrent.ForkJoinPool.unmanagedBlock(Ljava/util/concurrer				32
at java.util.concurrent.ForkJoinPool.managedBlock(Ljava/util/concurrent/f				
> at java.util.concurrent.locks.AbstractQueuedSynchronizer\$ConditionObjec				32
at java.util.concurrent.ScheduledThreadPoolExecutor\$DelayedWorkQueue				112
> <Java Local> java.util.concurrent.ScheduledThreadPoolExecutor\$Dela		32	112	
> <Java Local> java.util.concurrent.locks.ReentrantLock @ 0xfc8a5a78		16	16	
Σ Total: 2 entries				
> at java.util.concurrent.ScheduledThreadPoolExecutor\$DelayedWorkQueue				112
> at java.util.concurrent.ThreadPoolExecutor.getTask()Ljava/lang/Runnable;				408

Видно локальные переменные

→ **Threads** – сводка потоков как объектов

◆ Примерно как thread dump, только гораздо подробнее

# Круто, но...

- Каждый раздел удобен в **своих** случаях
- А если у меня **особый** случай?
- Что если об источнике проблемы **неизвестно ничего**, кроме факта существования?



Нужен способ делать **любые запросы** к дампу

# Кучно-реляционный маппинг

Heap Dump		Реляционная СУБД
Дамп	↔	Схема (база)
Класс	↔	Таблица
Экземпляр класса	↔	Строка таблицы
Поле класса	↔	Столбец таблицы



“Writing queries can be a bit of **an art**,  
with some **trial and error** required.”

---

– [https://wiki.eclipse.org/MemoryAnalyzer/OQL#Writing\\_Queries](https://wiki.eclipse.org/MemoryAnalyzer/OQL#Writing_Queries)

## 2 инструмента запросов в MAT

Фича \ Инструмент	OQL	Calcite SQL
Происхождение	<u>Встроен</u> в MAT	<u>Внешний</u> плагин
Движок	(n/a)	<u>Apache Calcite</u> (JDBC)
Поддержка JOIN	Нет*	<b>Есть</b>
Функции count/avg/sum/...	Нет	<b>Есть</b>
GROUP BY / ORDER BY	Нет*	<b>Есть</b>

\* Можно сделать косвенно



**сторонний  
плагин SQL**

**пользователь  
Eclipse MAT**

**встроенный  
движок OQL**

# Eclipse MAT Calcite SQL Plugin

Пример. Найти все дублирующиеся URL'ы в куче:

```
SELECT toString(u.file) AS file_str,  
       count(*) AS cnt,  
       sum(retainedSize(u.this)) AS sum_retained  
FROM java.net.URL u  
GROUP BY toString(u.file)  
HAVING count(*) > 1  
ORDER BY sum(retainedSize(u.this)) DESC
```

<https://github.com/vlsi/mat-calcite-plugin?tab=readme-ov-file#sample>

# Особенности Calcite SQL

Найти классы всех питомцев младше\* 2020-го года:

```
SELECT p.name AS "Name",  
       p.this['birthDate']['year'] AS "Year",  
       p.this['@className'] AS "Class"  
FROM instanceof.org.springframework.samples.petclinic.model.Pet p  
WHERE p.this['birthDate']['year'] > 2020
```

\* Поле `birthDate` имеет тип `LocalDate`

# Особенности Calcite SQL

Найти классы всех питомцев младше\* 2020-го года:

```
SELECT p.name AS "Name",  
       p.this['birthDate']['year'] AS "Year",  
       p.this['@className'] AS "Class"  
FROM instanceof.org.springframework.samples.petclinic.model.Pet p  
WHERE p.this['birthDate']['year'] > 2020
```

\* Поле `birthDate` имеет тип `LocalDate`

# Размышления вслух

## Вопрос

Если MAT Calcite SQL настолько похож на “настоящий” SQL, то нельзя ли доверить его генерацию нейросети?

## Ответ

Можно, при условии, что в состав промпта войдёт **описание схемы** БД (дампа) в виде набора DDL-выражений, а сама модель будет **знать диалект** MAT Calcite SQL.

# Фрагмент DDL дампа памяти

*-- table with entities of type 'Pet'*

```
CREATE TABLE "org.springframework.samples.petclinic.model.Pet" (  
  this INTEGER PRIMARY_KEY,          -- unique ID for each entity  
  birthDate INTEGER,                 -- can be joined with "...LocalDate" on 'this'  
  type INTEGER,                       -- can be joined with "...model.PetType" on 'this'  
  owner INTEGER,                      -- can be joined with "...model.Owner" on 'this'  
  visits INTEGER,  
  name VARCHAR(1024),  
  id INTEGER  
);
```

Да, ChatGPT читает комментарии.

а ты? o\_o

# Пример запроса через ChatGPT (1)

> *Who the Max cat belongs to?*

```
SELECT o.firstName, o.lastName
FROM "org.springframework.samples.petclinic.model.Owner" o
     JOIN "org.springframework.samples.petclinic.model.Pet" p
         ON o.this = p.owner
     JOIN "org.springframework.samples.petclinic.model.PetType" pt
         ON p.type = pt.this
WHERE toString(p.name) = 'Max' AND toString(pt.name) = 'cat'
```

# Пример запроса через ChatGPT (1)

> *Who the Max cat belongs to?*

```
firstName      |lastName
-----
Jean           |Coleman
```

Экспортированный результат из MAT

# Пример запроса через ChatGPT (2)

> *Who is the oldest animal in our clinic?*

```
SELECT p.name, toString(pt.name) AS type, MIN(bd."year") AS birthYear
FROM "org.springframework.samples.petclinic.model.Pet" p
JOIN "java.time.LocalDate" bd ON p.birthDate = bd.this
JOIN "org.springframework.samples.petclinic.model.PetType" pt
    ON p.type = pt.this
GROUP BY p.name, toString(pt.name)
ORDER BY MIN(bd."year") ASC
FETCH FIRST 1 ROWS ONLY
```

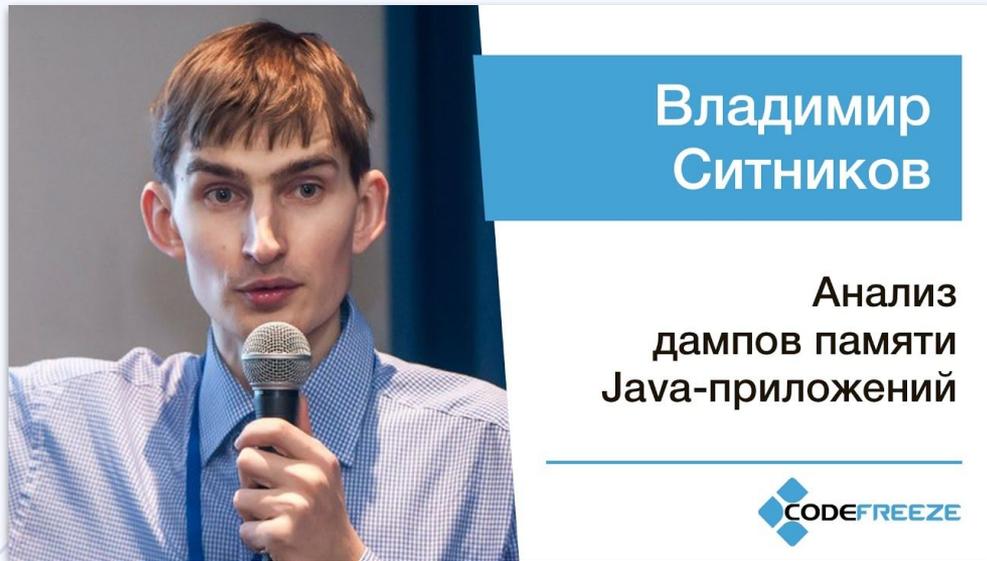
# Пример запроса через ChatGPT (2)

> *Who is the oldest animal in our clinic?*

name	type	birthYear
Mulligan	dog	2007

Экспортированный результат из MAT

# Где узнать больше



[https://youtu.be/t\\_WyfS9a7k?si=gMganDpBRYHKZaE4](https://youtu.be/t_WyfS9a7k?si=gMganDpBRYHKZaE4)



<https://github.com/toparvion/heap-talk>

# Попутное резюме по дампам

- **Снимать** с осторожностью
  - ◆ помня про замирание и секреты
- **Открывать** в Eclipse MAT
  - ◆ закладывая время на парсинг
- **Смотреть** на retained size
  - ◆ сортируя через dominator tree
- **Делегировать** LLM генерацию запросов
  - ◆ потягивая смузи трубочкой





# Альтернативы

---

Снятию дампа и его анализу

# Что не так с дампом?

- Снятие аффектит приложение
- Размер может быть большим
- Перед анализом нужен парсинг
- Некоторые действия не автоматизируемы

Плохая  
повторяемость



Далеко не всегда нужен снимок **всей кучи**

# Альтернатива 1: гистограмма

- Гистограмма классов может многое прояснить
- Для её получения не обязательно снимать дамп

```
$ jcmd <PID> GC.class_histogram
```

num	#instances	#bytes	class name (module)
1:	96283	10526160	[B
2:	93557	2993824	j.u.c.ConcurrentHashMap\$Node
3:	92015	2208360	java.lang.String
...			

## Альтернатива 2: аллокации (JFR)

- Бывает нужно понять, **как** были созданы объекты
- Для этого дампы не обязательны (а порой и бесполезны)
- Нужно захватывать **стекарейсы** аллокаций
- Можно применить Java Flight Recorder

# Альтернатива 2: аллокации (JFR)

Запись аллокаций нужно включить явно:

- Через настройки в Mission Control
- Через командную строку
- Через файл конфигурации \*.jfc

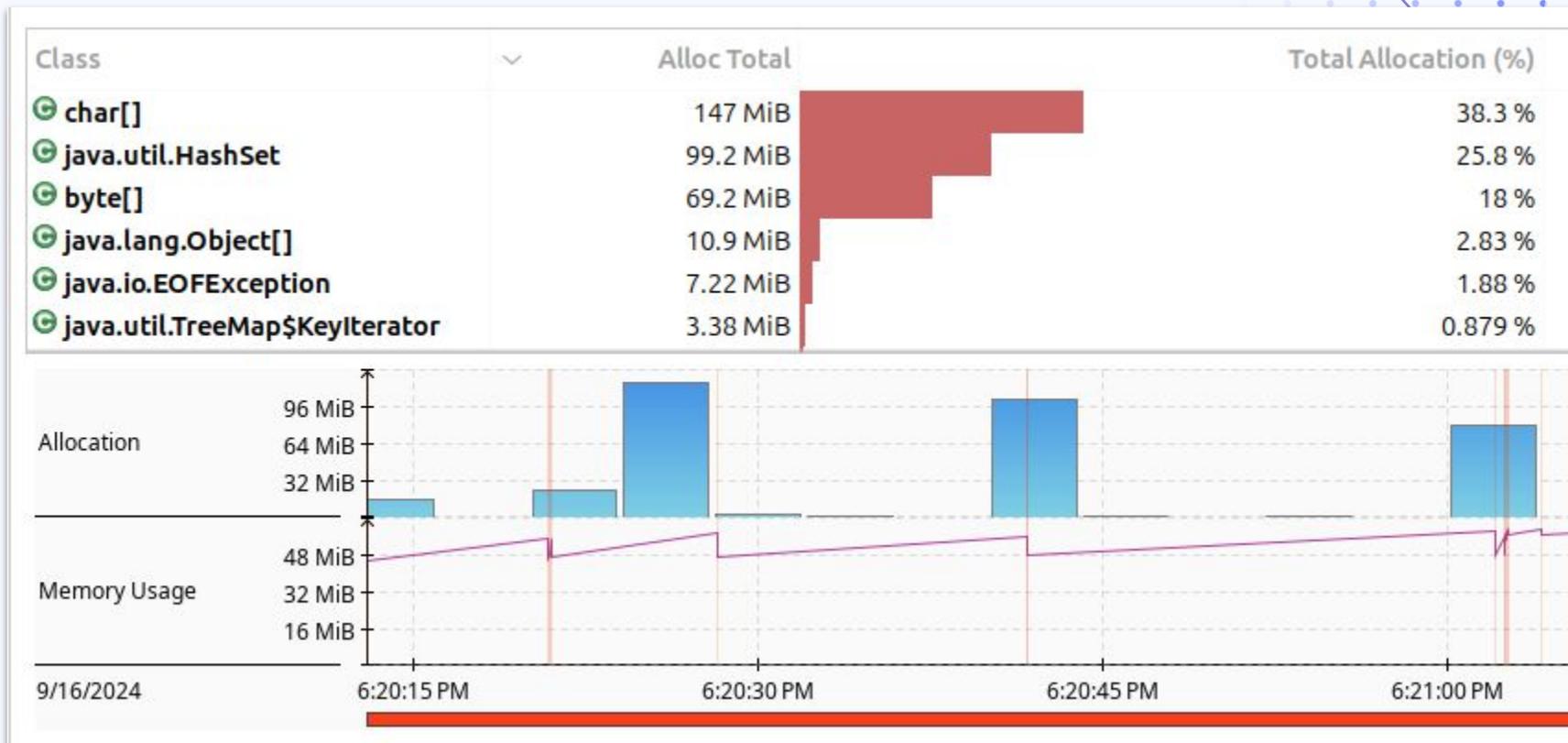


*Любым способом,  
результат одинаков*

Event Details for Settings for 'My Recording'  
Inspect and/or change the settings.

Filter: Allocation

- Java Application
  - Statistics
    - Thread Allocation Statistics
      - Enabled=true
      - Period=everyChunk
    - Allocation in new TLAB
      - Enabled=false
      - Stack Trace=true
    - Allocation outside TLAB
      - Enabled=false
      - Stack Trace=true
    - Object Allocation Sample**
      - Enabled=true
      - Event Emission Throttle=150/s
      - Stack Trace=true



Результат: раздел **Memory** в Mission Control

# Альтернатива 3: захват “на лету”

- Когда нужно увидеть значения переменных “в живую”
- Это как навтыкать `System.out.println()`
  - ◆ только в уже **запущенном** приложении
- Такое можно сделать с помощью btrace
  - ◆ а ещё jmint, но не на production

# Основы применения VTrace

1. Пишем т.н. `trace script` – класс-пробник
2. Подключаем его к JVM
  - a. Либо на лету
  - b. Либо при старте
3. Ловим результаты (в консоли или файле)
4. Повторяем, если надо

@BTrace

```
public class Tracer {  
    @OnMethod(clazz = "+org.rrd4j.core.RrdPrimitive",  
        method = "writeDouble", type = "void (int, double, int)")  
    public static void writeDouble(  
        @ProbeMethodName(fqn = true) String methodName,  
        @Self Object rrdDoubleArray,  
        int index, double value, int count) {  
        String pointer = str(Reflective.getLong("pointer", rrdDoubleArray));  
        println("Arguments: pointer=" + pointer +  
            ", index=" + index + ", value=" + value + ", count=" + count);  
        jstack();  
    }  
}
```

## @BTrace

```
public class Tracer {
```

```
    @OnMethod(clazz = "+org.rrd4j.core.RrdPrimitive",
              method = "writeDouble", type = "void (int, double, int)")
    public static void writeDouble(
        @ProbeMethodName(fqn = true) String methodName,
        @Self Object rrdDoubleArray,
        int index, double value, int count) {
        String pointer = str(Reflective.getLong("pointer", rrdDoubleArray));
        println("Arguments: pointer=" + pointer +
            ", index=" + index + ", value=" + value + ", count=" + count);
        jstack();
    }
}
```

```
@BTrace
```

```
public class Tracer {
```

```
    @OnMethod(clazz = "+org.rrd4j.core.RrdPrimitive",  
              method = "writeDouble", type = "void (int, double, int)")
```

```
    public static void writeDouble(  
        @ProbeMethodName(fqn = true) String methodName,  
        @Self Object rrdDoubleArray,  
        int index, double value, int count) {
```

```
        String pointer = str(Reflective.getLong("pointer", rrdDoubleArray));  
        println("Arguments: pointer=" + pointer +  
            ", index=" + index + ", value=" + value + ", count=" + count);  
        jstack();  
    }
```

```
}
```

```
}
```

```
@BTrace
```

```
public class Tracer {  
    @OnMethod(clazz = "+org.rrd4j.core.RrdPrimitive",  
             method = "writeDouble", type = "void (int, double, int)")  
    public static void writeDouble(  
        @ProbeMethodName(fqn = true) String methodName,  
        @Self Object rrdDoubleArray,  
        int index, double value, int count) {  
        String pointer = str(Reflective.getLong("pointer", rrdDoubleArray));  
        println("Arguments: pointer=" + pointer +  
            ", index=" + index + ", value=" + value + ", count=" + count);  
        jstack();  
    }  
}
```

```
@BTrace
```

```
public class Tracer {  
    @OnMethod(clazz = "+org.rrd4j.core.RrdPrimitive",  
        method = "writeDouble", type = "void (int, double, int)")  
    public static void writeDouble(  
        @ProbeMethodName(fqn = true) String methodName,  
        @Self Object rrdDoubleArray,  
        int index, double value, int count) {  
        String pointer = str(Reflective.getLong("pointer", rrdDoubleArray));  
        println("Arguments: pointer=" + pointer +  
            ", index=" + index + ", value=" + value + ", count=" + count);  
        jstack();  
    }  
}
```

```
@BTrace
```

```
public class Tracer {  
    @OnMethod(clazz = "+org.rrd4j.core.RrdPrimitive",  
        method = "writeDouble", type = "void (int, double, int)")  
    public static void writeDouble(  
        @ProbeMethodName(fqn = true) String methodName,  
        @Self Object rrdDoubleArray,  
        int index, double value, int count) {  
        String pointer = str(Reflective.getLong("pointer", rrdDoubleArray));  
        println("Arguments: pointer=" + pointer +  
            ", index=" + index + ", value=" + value + ", count=" + count);  
        jstack();  
    }  
}
```

# Альтернатива 4: dump, но не heap

- Heap dump можно получить из **core dump**:
  - ◆ “сырой” слепок адресного пространства процесса
  - ◆ делается силами ОС или внешнего инструмента (не JVM)
- Core dump может сниматься **по запросу** и **автоматически**

# Получение core dump по запросу

Может быть заметно быстрее, чем heap dump.

Однако:

- Требует отдельного инструмента ([gdb](#))
- Может отъесть много места на диске ([VIRT MEM](#))
- Нет гарантий успешной конвертации в HPROF
  - ◆ зато есть [пример](#) успеха

# Сводка альтернатив

→ `jcmd <PID> GC.class_histogram`

◆ когда надо проверить **состав классов**

→ `JFR / AsyncProfiler`

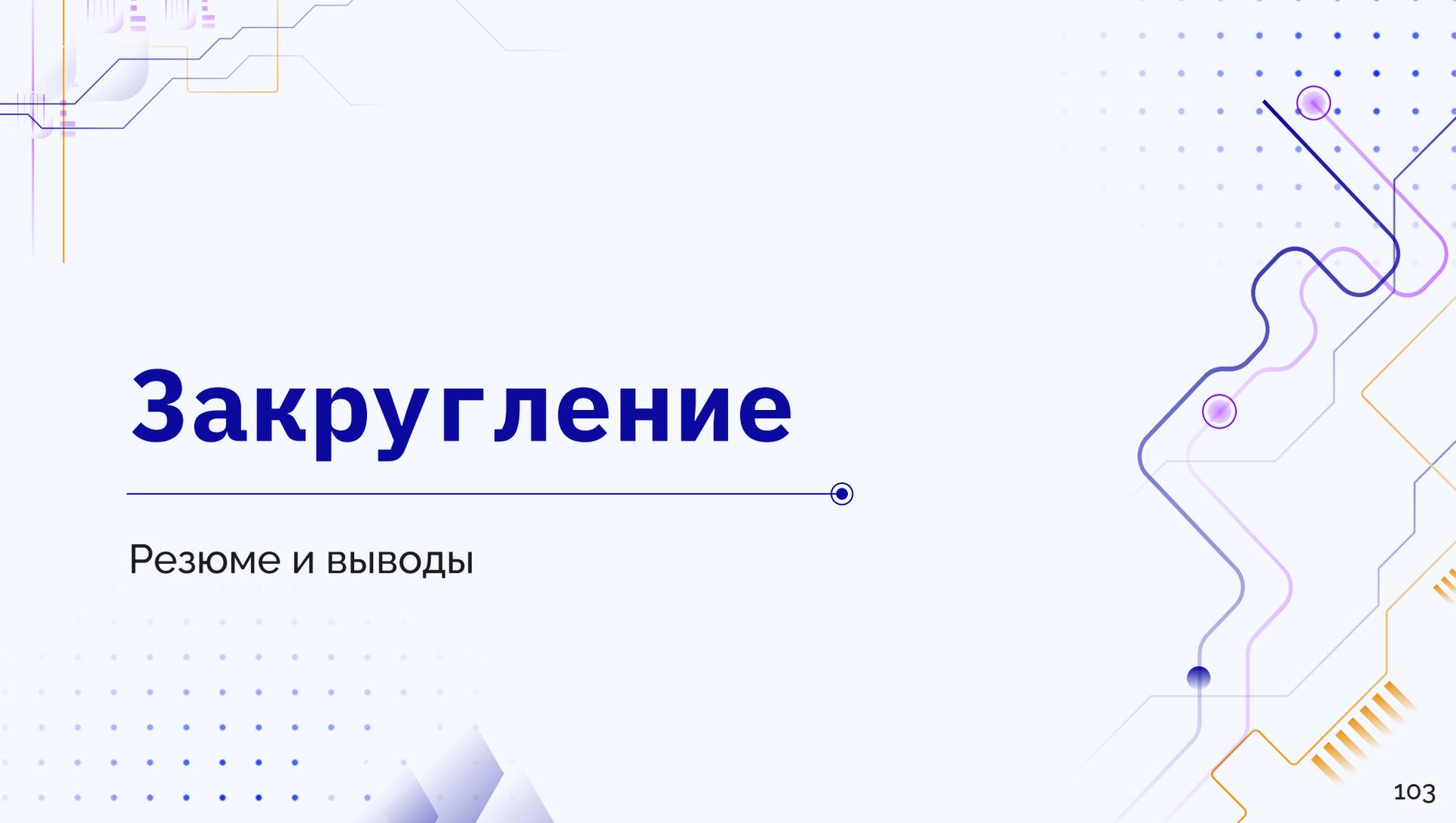
◆ когда надо выяснить точный **стекарейс** аллокации

→ `VTrace`

◆ когда нужно извлечь точные **значения** на горячую

→ `Core dump`

◆ когда надо снять **быстрее** или **после краха JVM**



# Закругление

---

Резюме и выводы

# Искать баги в дампах – это...



Как читать детектив,

# Искать баги в дампах – это...



# Takeaways

- Проблем с памятью не избежать, но можно **подготовиться**:
  - ◆ пропишите `HeapDumpOnOutOfMemoryError` **уже сегодня**
- Анализировать дампы лучше в Eclipse MAT
  - ◆ сначала **Top Consumers & Dominator Tree**, потом OQL/SQL
- Прежде чем снимать/открывать дамп, см. **альтернативы**
  - ◆ чтобы найти способ разобраться **быстрее**

# Полезные материалы

- Андрей Паньгин – Память Java-процесса по полочкам
  - ◆ [доклад] Для представления “общей картины”
- Владимир Ситников – Разбор сложных случаев OOM
  - ◆ [доклад] Для более глубокого погружения в тему
- Emily Chang - Monitor Java memory management
  - ◆ [статья] Для понимания метрик и поведения GC

# Полезные инструменты

- **MAT Calcite SQL Plugin** – движок SQL для Eclipse MAT
  - ◆ Проект на GitHub
- **HeapTalk** – генератор SQL для опроса heap-дампов
  - ◆ Описание на русском и проект на GitHub
- **BugClinic** – версия Spring PetClinic с описанными багами
  - ◆ Описание на русском и проект на GitHub

# Спасибо!

Можно задавать вопросы :)

Владимир Плизга  
@Toparvion



**CREDITS:** This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**



<https://toparvion.pro/event/2024/ioker/>

↑ слайды ↑