

ТИНЬКОФФ

Сколько памяти нужно для сборки?

Андроид разработчикам о JVM

Юрий Анисимов



Сборка замедляется постепенно

Замедлить сборку может, как слишком малое, так и излишне большое выделение памяти.



Использование оперативной памяти



Как оценить количество выделенной памяти?



Поможет ли увеличение памяти ускорить сборку?



Как определить приемлемое количество памяти?



Сколько памяти потребляет Java процесс?

Что будет в докладе

Основы сборки мусора

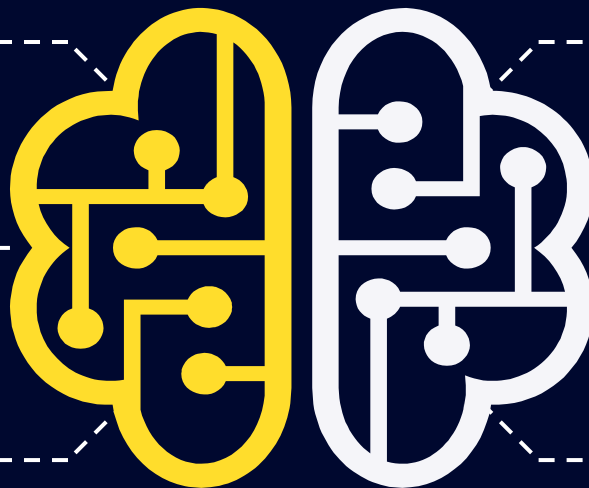
Процессы сборки

Garbage-First GC

Профилирование

Parallel GC

ОСНОВЫ ТЮНИНГА



В докладе не будет советов в стиле «делайте так...»

«No Silver Bullet»

Frederick P. Brooks, Jr.



Юрий Анисимов

Разработчик Инфра.



y.anisimov@tinkoff.ru

Мобильный банк



Случайное приложение



- ▼ ~95k строк кода
- ▼ 100+ модулей
- ▼ полностью на Kotlin
- ▼ Hilt, Ktor, Compose

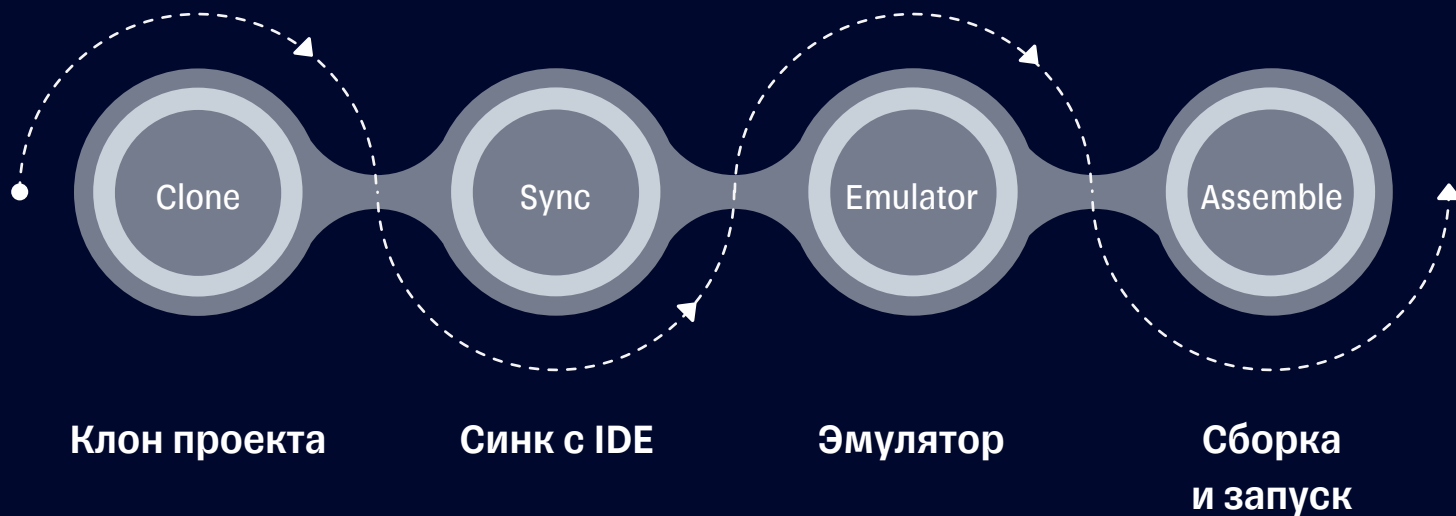
Настройки Java Virtual Machine (JVM)

gradle.properties

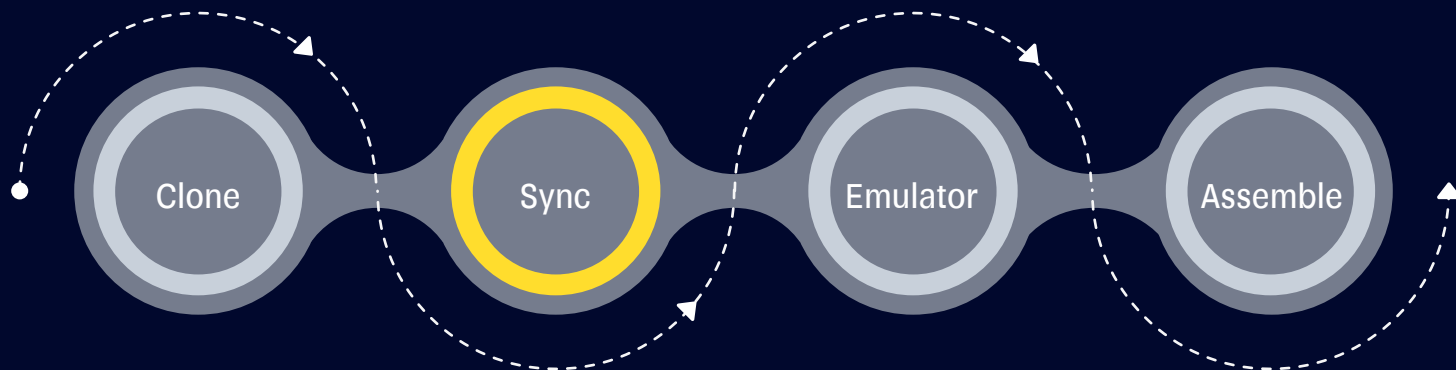
```
org.gradle.jvmargs=-Xmx6g -Dfile.encoding=UTF-8  
...
```

Heap space — область для динамического выделения памяти под Java объекты и классы JRE.

Первый запуск приложения новым разработчиком



Первый запуск приложения новым разработчиком



Клон проекта

Синк с IDE

Эмулятор

Сборка
и запуск

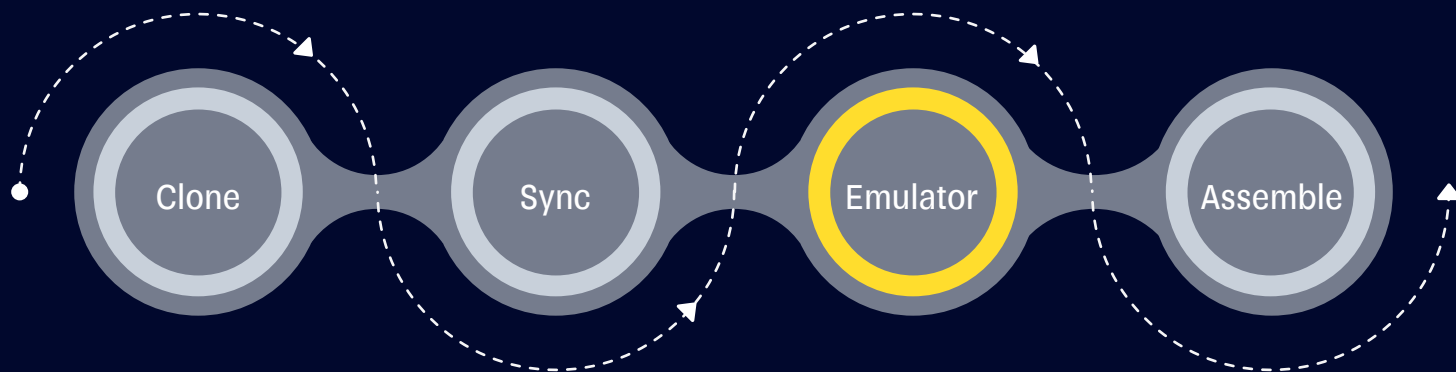
i Android Studio performance could be improved

Increasing the maximum heap size from 1280MB to 4096MB could make the IDE perform better, based on the available memory and your project size.

Actions ▾ Don't show again



Первый запуск приложения новым разработчиком



Клон проекта

Синк с IDE

Эмулятор

**Сборка
и запуск**

- Pixel 4
- Android 13
- Настройки
по умолчанию

Результаты



- ✓ Откуда взялся второй Java процесс?
- ✓ Нужно ли такое количество RAM?
- ✓ Как быть, если доступно всего 16 Гб?

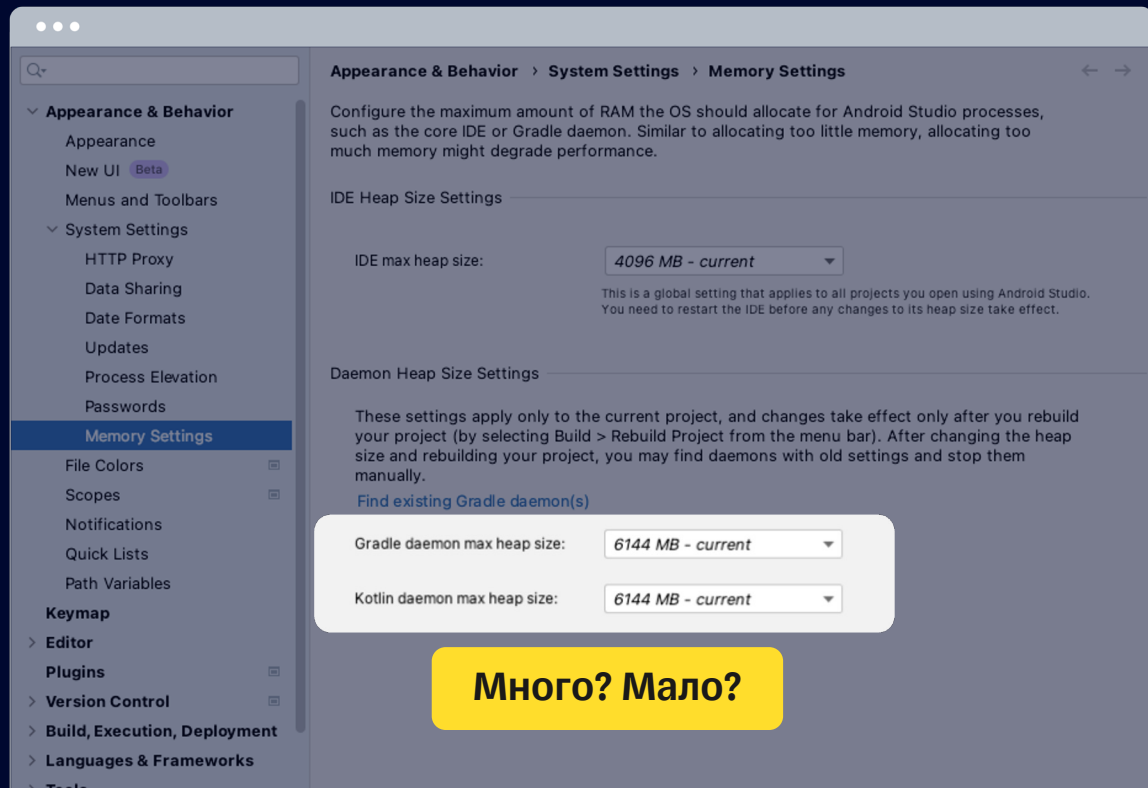
Список Java процессов

```
% jps
```


Список Java процессов

```
% jps
29384
39779 GradleDaemon
40011 KotlinCompileDaemon
40056 Jps
```

Настройки памяти



Build Performance Analyzer

The screenshot displays the Build Performance Analyzer interface. At the top, there are navigation options: "Search Everywhere Double", "Go to File", "Recent Files", and "Navigation Bar". Below this, the "Build Analyzer" tab is active, showing an "Overview" section. The main content area is divided into three columns:

- Build finished on 09.10.2023, 19:47**
Total build duration was 96.3s
Includes:
Build configuration: 3.3s - [Optimize this](#)
Critical path tasks execution: 92.6s
- Common views into this build**
[Tasks impacting build duration](#)
[Plugins with tasks impacting build duration](#)
[All warnings](#)
- Gradle Daemon Memory Utilization**
2.6% (2.5s) of your build's time was dedicated to garbage collection during this build.
You can change the Gradle daemon heap size on the memory settings page.
[Edit memory settings](#)
The default garbage collector was used in this build running with JDK 17.
Note that the default GC was changed starting with JDK 9. This could impact your build performance by as much as 10%.
Recommendation: [Fine tune your JVM](#).
[Don't show this again.](#)

The bottom status bar includes icons for Git, Profiler, Logcat, App Quality Insights, Build, TODO, Problems, Terminal, Services, App Inspection, and Layout Inspector. On the right side, there are vertical toolbars for Notifications, Grade, Device Explorer, and Running Devices.

Build Performance Analyzer

The screenshot shows the Build Performance Analyzer tool interface. The main content area is titled "Overview" and displays the following information:

- Build finished on 09.10.2023, 19:47**
Total build duration was 96.3s
- Includes:**
Build configuration: 3.3s - [Optimize this](#)
Critical path tasks execution: 92.6s
- Common views into this build**
 - [Tasks impacting build duration](#)
 - [Plugins with tasks impacting build duration](#)
 - [All warnings](#)
- Gradle Daemon Memory Utilization**
 - 2.6% (2.5s) of your build's time was dedicated to garbage collection during this build. You can change the Gradle daemon heap size on the memory settings page. [Edit memory settings](#)
 - The default garbage collector was used in this build running with JDK 17. Note that the default GC was changed starting with JDK 9. This could impact your build performance by as much as 10%. **Recommendation: [Fine tune your JVM](#).** [Don't show this again.](#)

The interface also features a search bar with "Search Everywhere Double" and navigation options like "Go to File", "Recent Files", and "Navigation Bar". The bottom status bar includes icons for Git, Profiler, Logcat, App Quality Insights, Build, TODO, Problems, Terminal, Services, App Inspection, and Layout Inspector.

Тонкая настройка JVM

The screenshot shows a web browser displaying the Android Developers page titled "Increase the JVM heap size". The page content includes:

- Section Header:** "Increase the JVM heap size" with a link icon.
- Text:** "If you observe slow builds, and in particular the garbage collection takes more than 15% of the build time in your Build Analyzer results, then you should increase the Java Virtual Machine (JVM) heap size. In the `gradle.properties` file, set the limit to 4, 6, or 8 gigabytes as shown in the following example:"
- Code Block:**

```
org.gradle.jvmargs=-Xmx6g
```
- Text:** "Then test for build speed improvement. The easiest way to determine the optimal heap size is to increase the limit by a small amount and then test for sufficient build speed improvement."
- Note:** A blue note box states: "★ **Note:** If you change the default memory limit, you also have to set the `-XX:MaxMetaspaceSize=1g` argument due to [Gradle issue #19750](#)." with an external link icon.
- Text:** "If you also use the JVM parallel garbage collector, then the entire line should look like:"
- Code Block:**

```
org.gradle.jvmargs=-Xmx6g -XX:+HeapDumpOnOutOfMemoryError -Dfile.encoding=UTF-8 -XX:+UseParallelGC
```

The left sidebar contains a navigation menu with categories like "What's new in Android build", "Configure your build", "Optimize your build" (with "Overview" selected), "Troubleshoot build performance", "Profile your build", "Gradle tips and recipes", "Manage manifest files", "Shrink your app", "D8 and R8 compiler version compatibility", "Enable multidex", "Extend your build", and "Publish your library". The right sidebar lists various optimization tips such as "Avoid compiling unnecessary resources", "Experiment with putting the Gradle Plugin Portal last", "Use static build config values with your debug build", "Use static dependency versions", "Create library modules", "Create tasks for custom build logic", "Convert images to WebP", "Disable PNG crunching", "Experiment with the JVM parallel garbage collector", and "Increase the JVM heap size".

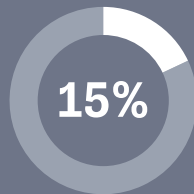
Build Performance Analyzer

The screenshot shows the Build Performance Analyzer tool interface. The main content area displays the following information:

- Build:** Sync × Build Output × Build Analyzer ×
- Overview** (selected tab)
- Build finished on 09.10.2023, 19:47**
Total build duration was 96.3s
- Includes:**
Build configuration: 3.3s - [Optimize this](#)
Critical path tasks execution: 92.6s
- Common views into this build**
 - [Tasks impacting build duration](#)
 - [Plugins with tasks impacting build duration](#)
 - [All warnings](#)
- Gradle Daemon Memory Utilization**
 - 2.6% (2.5s)** of your build's time was dedicated to garbage collection during this build. You can change the Gradle daemon heap size on the memory settings page.
 - [Edit memory settings](#)
 - The default garbage collector was used in this build running with JDK 17. Note that the default GC was changed starting with JDK 9. This could impact your build performance by as much as 10%. **Recommendation:** [Fine tune your JVM](#).
 - [Don't show this again.](#)

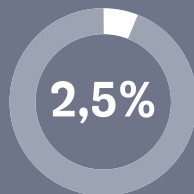
The interface also features a search bar with the text "Search Everywhere Double", and a bottom status bar with icons for Git, Profiler, Logcat, App Quality Insights, Build, TODO, Problems, Terminal, Services, App Inspection, and Layout Inspector. On the right side, there are vertical toolbars for Notifications, Grade, Device Explorer, and Running Devices.

До какого размера уменьшать?



Сколько будет нормально?

Судя по документации, 15% это — крайне плохо.



Откуда взялось 2,5%?

При том, что потребление памяти не достигло лимита.

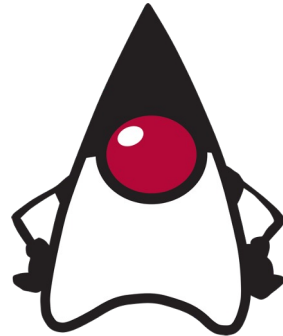
Java Platform, Standard Edition

HotSpot Virtual Machine Garbage Collection Tuning Guide



Release 17
F41050-02
April 2022

Release 17



Управление памятью



- ✓ Поиск «мертвых» объектов
- ✓ Освобождение неиспользуемых блоков

Управление памятью



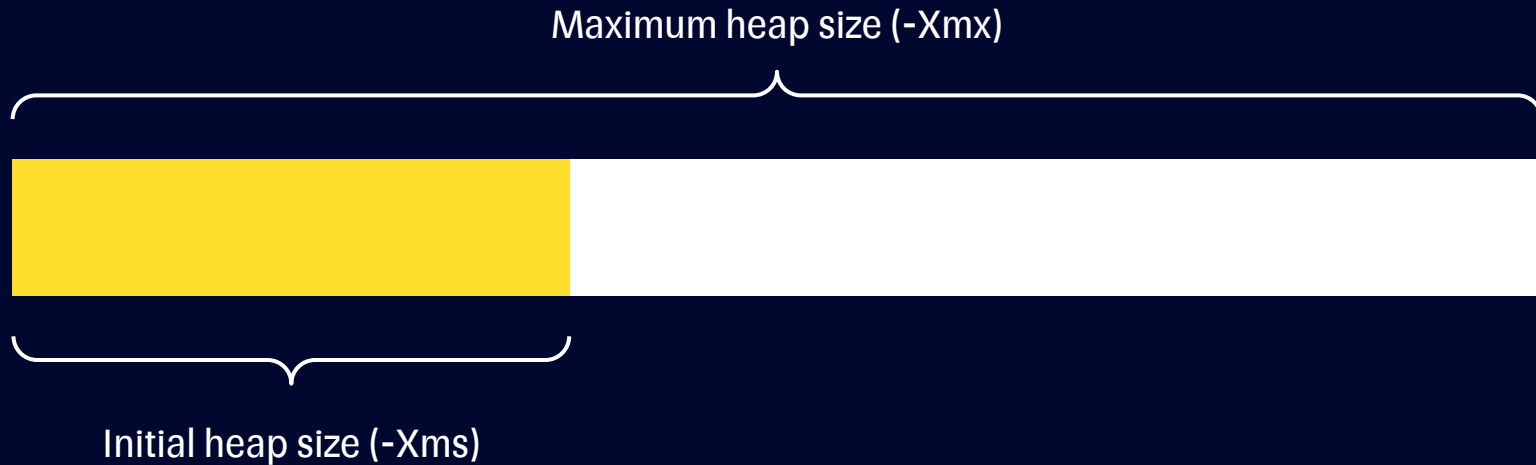
- ✓ Поиск «мертвых» объектов
- ✓ Освобождение неиспользуемых блоков
- ✓ Получение памяти из ОС
- ✓ Выделение памяти приложению

Размер Java Heap

Maximum heap size (-Xmx)



Размер Java Heap



Начальный размер для 16 Гб RAM?

Maximum heap size (-Xmx)



Initial heap size?

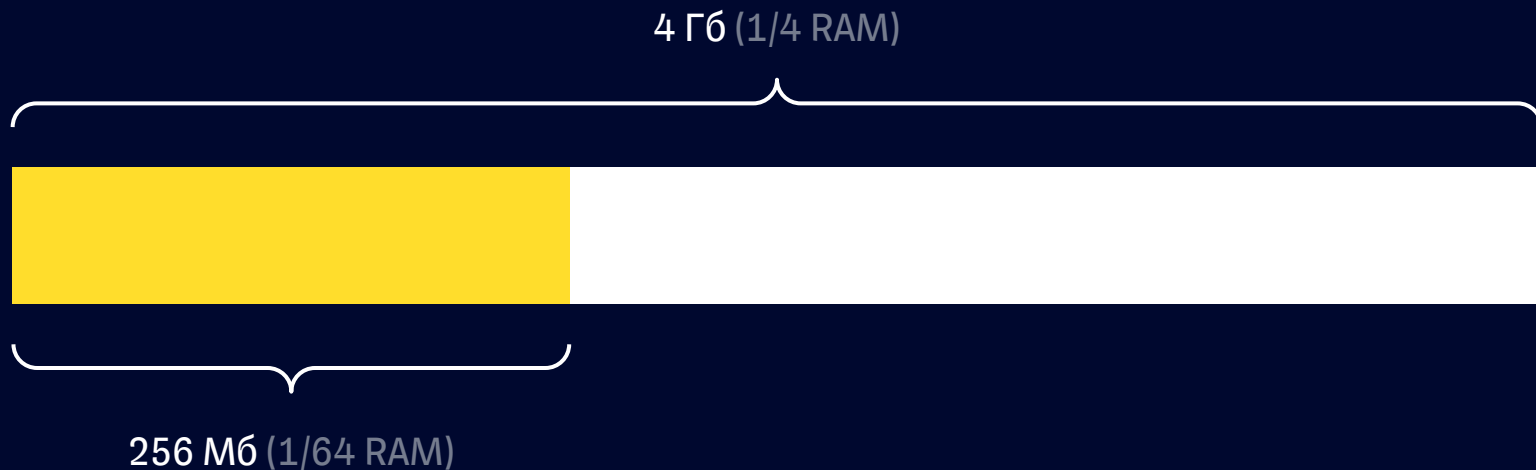


256 Mб



512 Mб

Начальный размер для 16 Гб RAM?



Эргономика

Процесс улучшения перфоманса приложения за счет эвристик JVM и GC. Например основанных на поведении приложения



Эргономика

Процесс улучшения перфоманса приложения за счет эвристик JVM и GC. Например основанных на поведении приложения



Throughput Goal

Время GC относительно всего времени работы

`-XX:GCTimeRatio=<nnn>`

Эргономика

Процесс улучшения перфоманса приложения за счет эвристик JVM и GC. Например основанных на поведении приложения



Throughput Goal

Время GC относительно всего времени работы

`-XX:GCTimeRatio=<nnn>`



Maximum Pause-Time Goal

Максимальное время остановки приложения

`-XX:MaxGCPauseMillis=<nnn>`

Эргономика

Процесс улучшения перфоманса приложения за счет эвристик JVM и GC. Например основанных на поведении приложения



Throughput Goal

Время GC относительно всего времени работы

-XX:GCTimeRatio=<nnn>



Maximum Pause-Time Goal

Максимальное время остановки приложения

-XX:MaxGCPauseMillis=<nnn>



Footprint (heap size)

Занимаемая область памяти

-Xms, -Xmx

Эргономика

Процесс улучшения перфоманса приложения за счет эвристик JVM и GC. Например основанных на поведении приложения



Throughput Goal

-XX:GCTimeRatio=<nnn>



Maximum Pause-Time Goal

-XX:MaxGCPauseMillis=<nnn>



Footprint (heap size)

-Xms, -Xmx

Варианты реализации GC



**Serial
Collector**



**Parallel
Collector**



Garbage-First



The Z GC

Варианты реализации GC



Serial Collector

`-XX:+UseSerialGC`

Для однопроцессорных
машин с малым
потреблением памяти
(100 Мб)



Parallel Collector



Garbage-First



The Z GC

Варианты реализации GC



Serial Collector

`-XX:+UseSerialGC`

Для однопроцессорных машин с малым потреблением памяти (100 Мб)



Parallel Collector

`-XX:+UseParallelGC`

Высокая производительность.
Длинные паузы для очистки мусора.



Garbage-First



The Z GC

Варианты реализации GC



Serial Collector

`-XX:+UseSerialGC`

Для однопроцессорных машин с малым потреблением памяти (100 Мб)



Parallel Collector

`-XX:+UseParallelGC`

Высокая производительность.
Длинные паузы для очистки мусора.



Garbage-First

`-XX:+UseG1GC`

Баланс между короткими паузами и высокой производительностью.



The Z GC

Варианты реализации GC



Serial Collector

`-XX:+UseSerialGC`

Для однопроцессорных машин с малым потреблением памяти (100 Мб)

Parallel Collector

`-XX:+UseParallelGC`

Высокая производительность.
Длинные паузы для очистки мусора.

Garbage-First

`-XX:+UseG1GC`

Баланс между короткими паузами и высокой производительностью.

The Z GC

`-XX:+UseZGC`

Минимальные паузы для любого размера хипа за счет потери в производительности.

Варианты реализации GC



Serial Collector

`-XX:+UseSerialGC`

Для однопроцессорных машин с малым потреблением памяти (100 Мб)



Parallel Collector

`-XX:+UseParallelGC`

Высокая производительность.
Длинные паузы для очистки мусора.



Garbage-First

`-XX:+UseG1GC`

Баланс между короткими паузами и высокой производительностью.



The Z GC

`-XX:+UseZGC`

Минимальные паузы для любого размера хипа за счет потери в производительности.

Выбор по умолчанию в Java 17?



Serial
Collector



Parallel
Collector



Garbage-First



The Z GC

```
org.gradle.jvmargs=-Xmx6g -Dfile.encoding=UTF-8
```

```
...
```

Выбор по умолчанию в Java 17?



Serial
Collector



Parallel
Collector

Основной до Java 8



Garbage-First

Начиная с Java 9



The Z GC

Релиз в Java 15

```
org.gradle.jvmargs=-Xmx6g -Dfile.encoding=UTF-8
```

```
...
```

Варианты реализации GC



**Serial
Collector**



**Parallel
Collector**



Garbage-First



The Z GC



Generational Garbage Collection

Варианты реализации GC



Serial
Collector



Parallel
Collector



Garbage-First



The Z GC

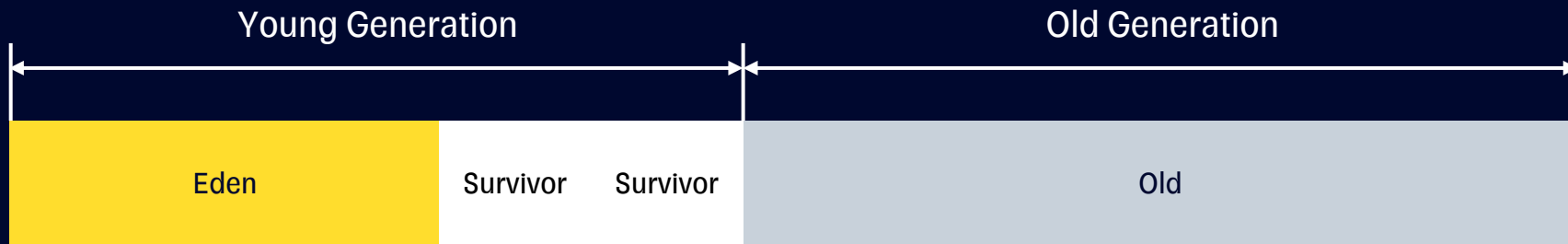


Generational Garbage Collection

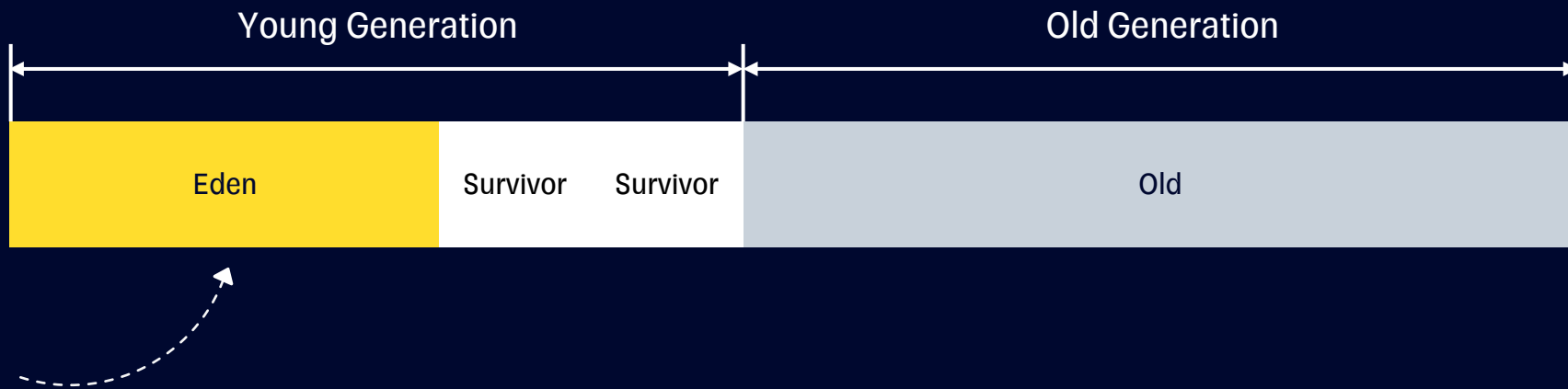
Большинство объектов «умирают» вскоре после создания.



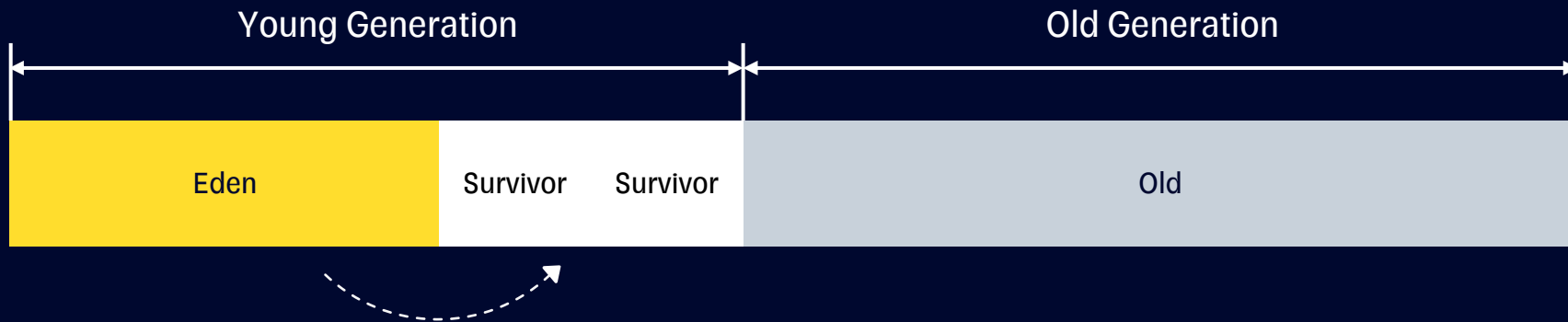
Generational collection



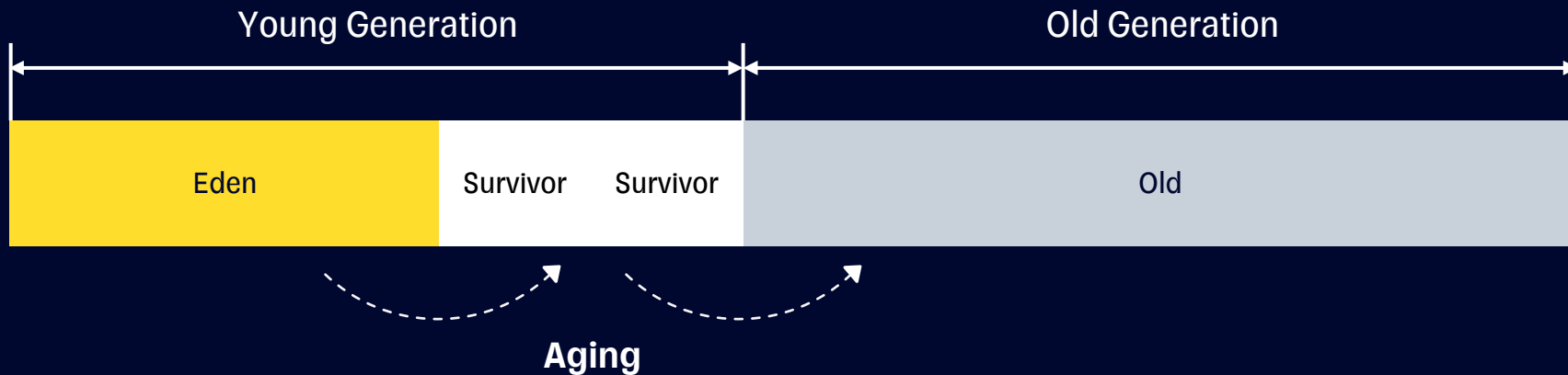
Generational collection



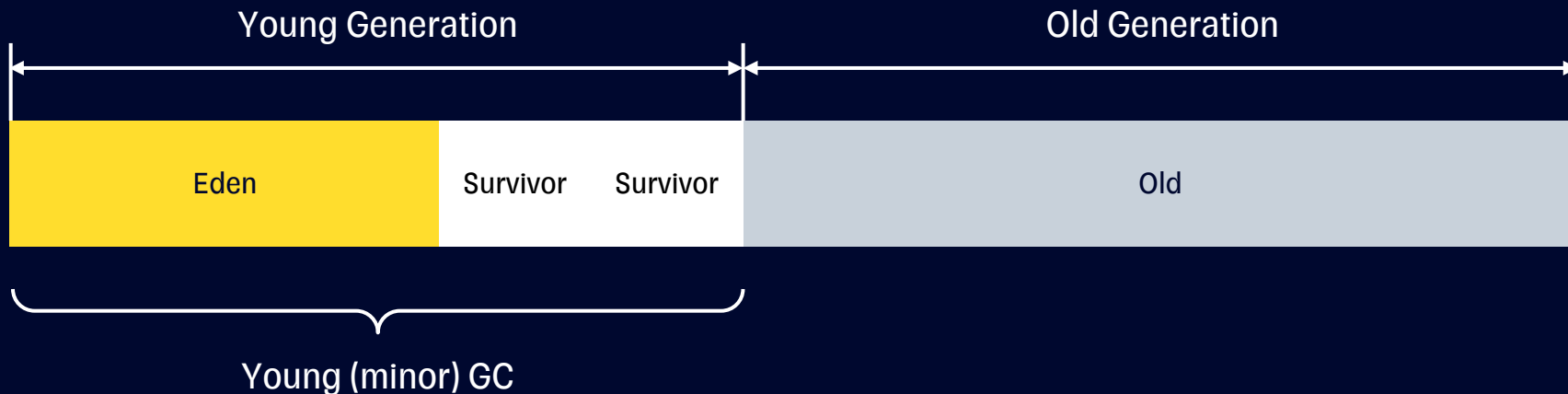
Generational collection



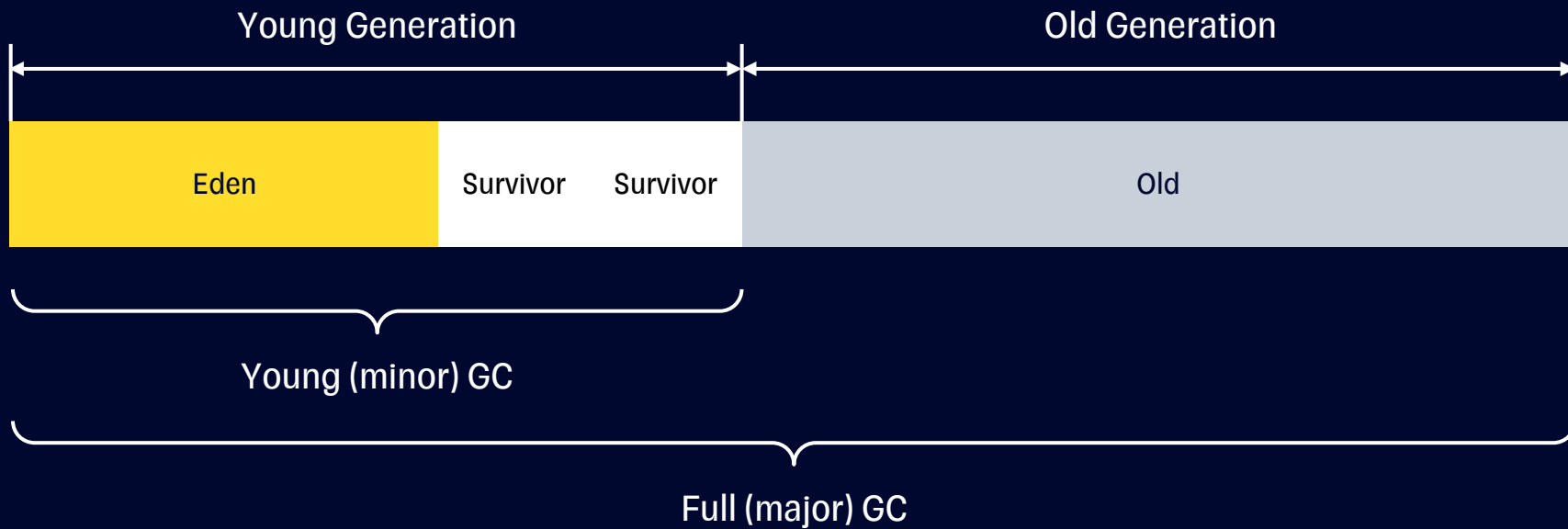
Generational collection



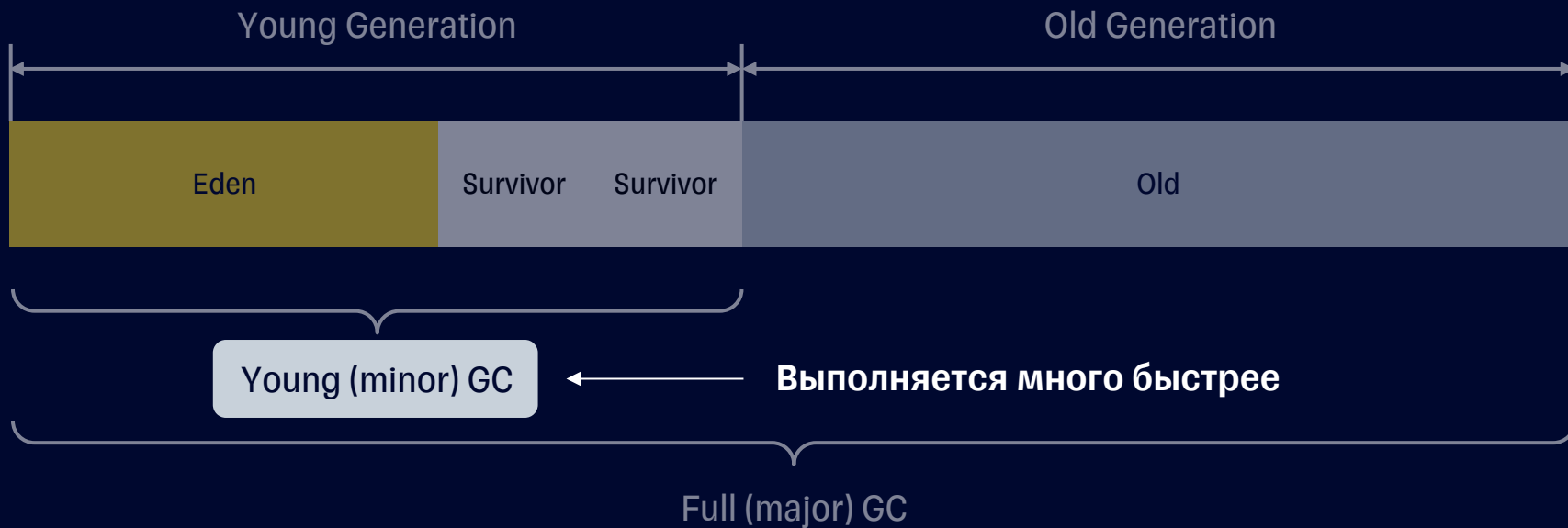
Generational collection



Generational collection



Generational collection



До какого размера уменьшать?



- Откуда взялись 2,5%?
- Сколько будет нормально?

Статистика Young generation

```
% jstat -gcnew <PID>
```

Опции jstat:

- -gcnew
- -gcold
- -gcutil
- ...

Статистика Young generation

```
% jstat -gcnew <PID>
```

S0C	S1C	S0U	S1U	TT	MTT	DSS	EC	EU	YGC	YGCT
0.0	61440.0	0.0	58942.6	15	15	141312.0	2441216.0	262144.0	103	2.510

Опции jstat:

- -gcnew
- -gcold
- -gcutil
- ...

Статистика Young generation

```
% jstat -gcnew <PID>
```

S0C	S1C	S0U	S1U	TT	MTT	DSS	EC	EU	YGC	YGCT
0.0	61440.0	0.0	58942.6	15	15	141312.0	2441216.0	262144.0	103	2.510



Размер Eden space: 2+ Гб

Статистика Young generation

```
% jstat -gcnew <PID>
```

```
S0C  S1C      S0U  S1U      TT  MTT  DSS      EC      EU  
0.0  61440.0  0.0  58942.6  15  15   141312.0  2441216.0  262144.0
```

```
YGC  YGCT  
103  2.510
```

- Размер Eden space: 2+ Гб
- **Количество срабатываний Young GC: 100+**
- **Время работы Young GC: 2.50 сек.**

До какого размера уменьшать?



- ✓ Откуда взялись 2,5%?
- Сколько будет нормально?

Эргономика

Garbage-First GC



Maximum Pause-Time Goal

-XX:MaxGCPauseMillis=200



Throughput Goal

-XX:GCTimeRatio=12

Эргономика

Garbage-First GC



Maximum Pause-Time Goal

-XX:MaxGCPauseMillis=200



Throughput Goal

-XX:GCTimeRatio=12

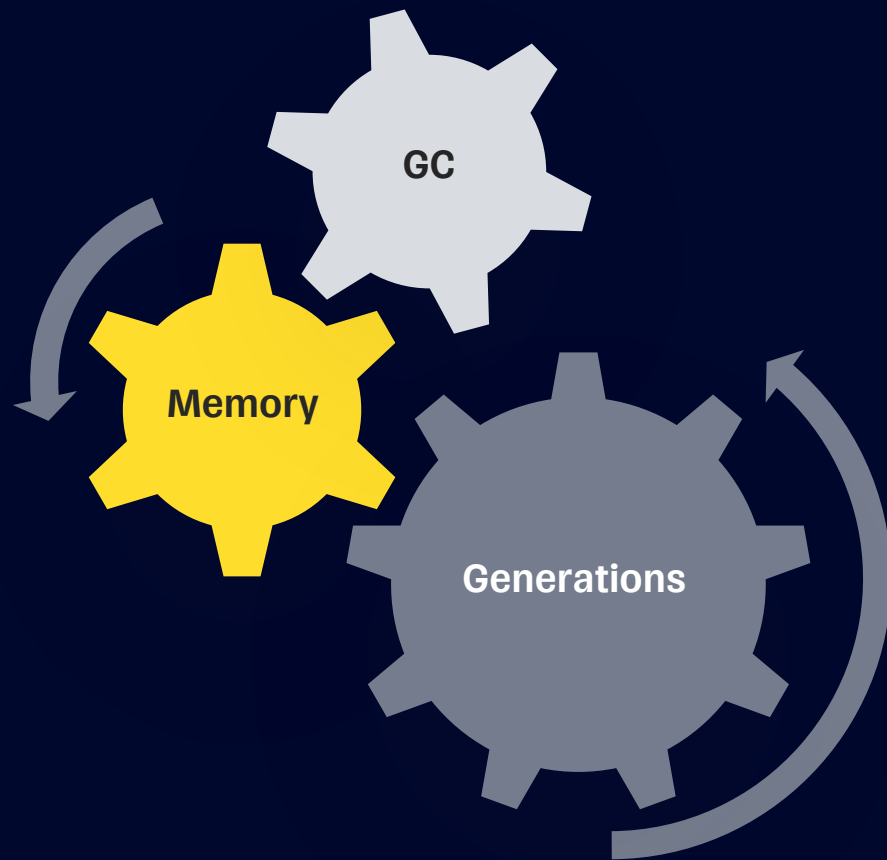
$$\frac{1}{\text{GCTimeRatio} + 1} = \frac{1}{12 + 1} \sim 8\%$$

**«Для Garbage-First
приемлемое время сборки
мусора будет в пределах 8%
(из практики — 5%)»**

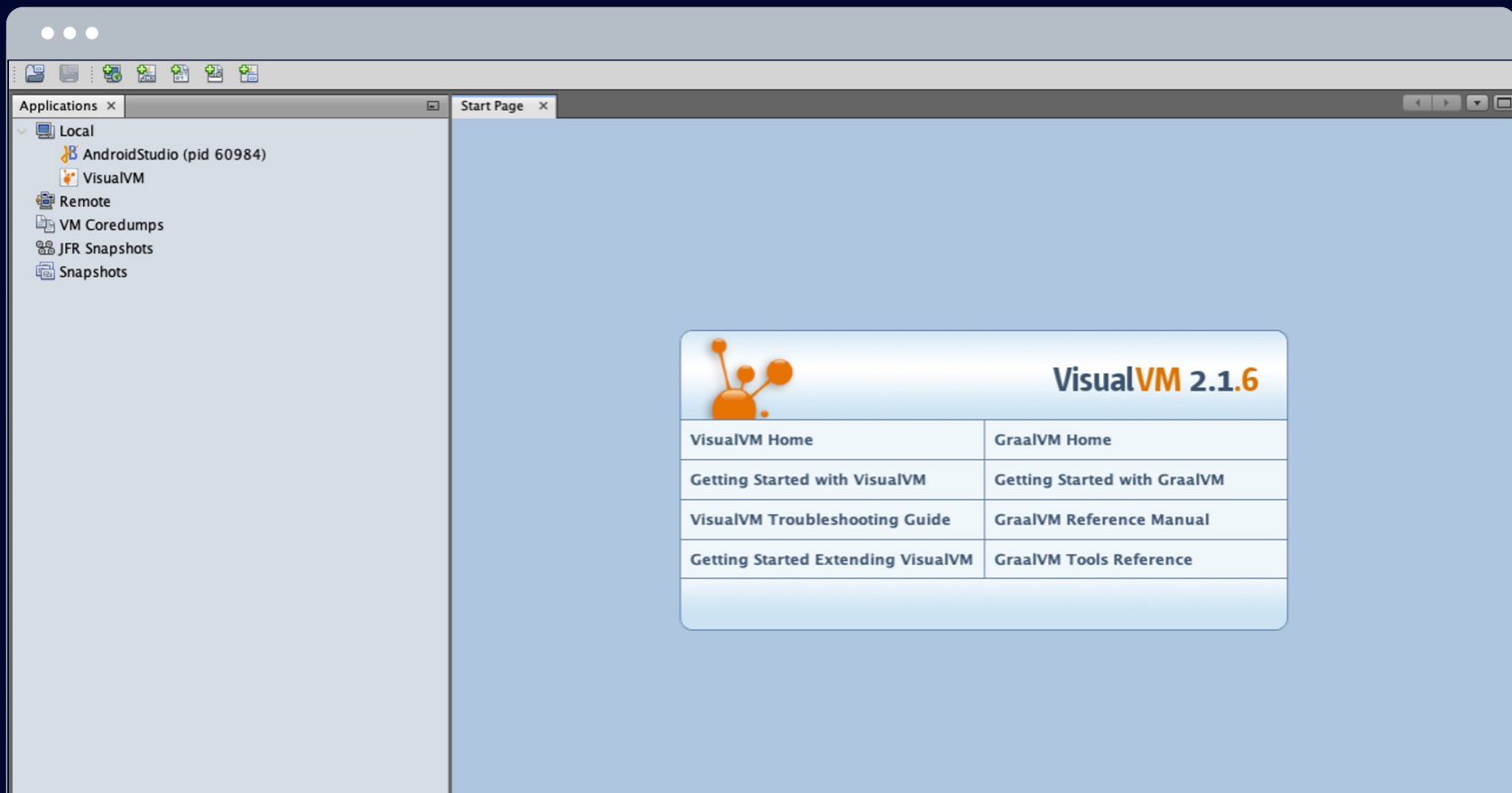


Профилирование

- ✓ Активность GC
- ✓ Выделение и освобождение памяти
- ✓ Использование областей Java heap




VisualVM с плагином Visual GC



Applications x Start Page x

- Local
 - AndroidStudio (pid 60984)
 - VisualVM
- Remote
- VM CoreDumps
- JFR Snapshots
- Snapshots

 **VisualVM 2.1.6**

VisualVM Home	GraalVM Home
Getting Started with VisualVM	Getting Started with GraalVM
VisualVM Troubleshooting Guide	GraalVM Reference Manual
Getting Started Extending VisualVM	GraalVM Tools Reference

Сценарий профилирования

```
% ./gradlew --stop  
% ./gradlew :app:assembleDebug --rerun-tasks --no-daemon
```

- ✓ Сборка отладочной версии
- ✓ Крайний случай с запуском всех тасок
- ✓ Показатели GC в рамках одного сеанса

Gradle демон: Monitor

The screenshot displays the Gradle Daemon Monitor interface for a daemon with PID 44313. The interface is divided into several sections:

- Overview:** Shows the daemon's uptime as 1 min 36 sec. It includes checkboxes for monitoring CPU, Memory, Classes, and Threads, all of which are checked. There are buttons for "Perform GC" and "Heap Dump".
- CPU:** A line graph showing CPU usage over time. The current usage is 90.1%. The y-axis ranges from 0% to 100%.
- GC activity:** A line graph showing garbage collection activity over time. The current activity is 0.5%. The y-axis ranges from 0% to 100%.
- Heap:** A stacked area chart showing heap size and used heap over time. The total size is 5,473,566,752 B and the used heap is 4,414,504,960 B. The maximum heap size is 6,442,450,968 B. The y-axis ranges from 0 GB to 5 GB.
- Classes:** A table showing the number of loaded and unloaded classes.

Total loaded:	79,259	Shared loaded:	0
Total unloaded:	44,288	Shared unloaded:	0
- Threads:** A table showing the number of live and started threads.

Live:	300	Daemon:	128
Live peak:	318	Total started:	461

Gradle демон: Monitor

The screenshot shows the IntelliJ IDEA interface with the Gradle Daemon Monitor open. The monitor displays the following information:

- Gradle Daemon (pid 44313)**
- Monitor** (checked): CPU, Memory, Classes, Threads
- Uptime:** 1 min 36 sec
- Buttons:** Perform GC, Heap Dump
- CPU usage:** 90.1%
- GC activity:** 0.5%
- Heap:** Size: 5,473,566,752 B; Max: 6,442,450,968 B
- Used heap:** 4,414,504,960 B
- Classes:** Total loaded: 79,259; Shared loaded: 0; Total unloaded: 44,288; Shared unloaded: 0
- Threads:** Live: 300; Live peak: 318; Daemon: 128; Total started: 461

Gradle демон: Monitor

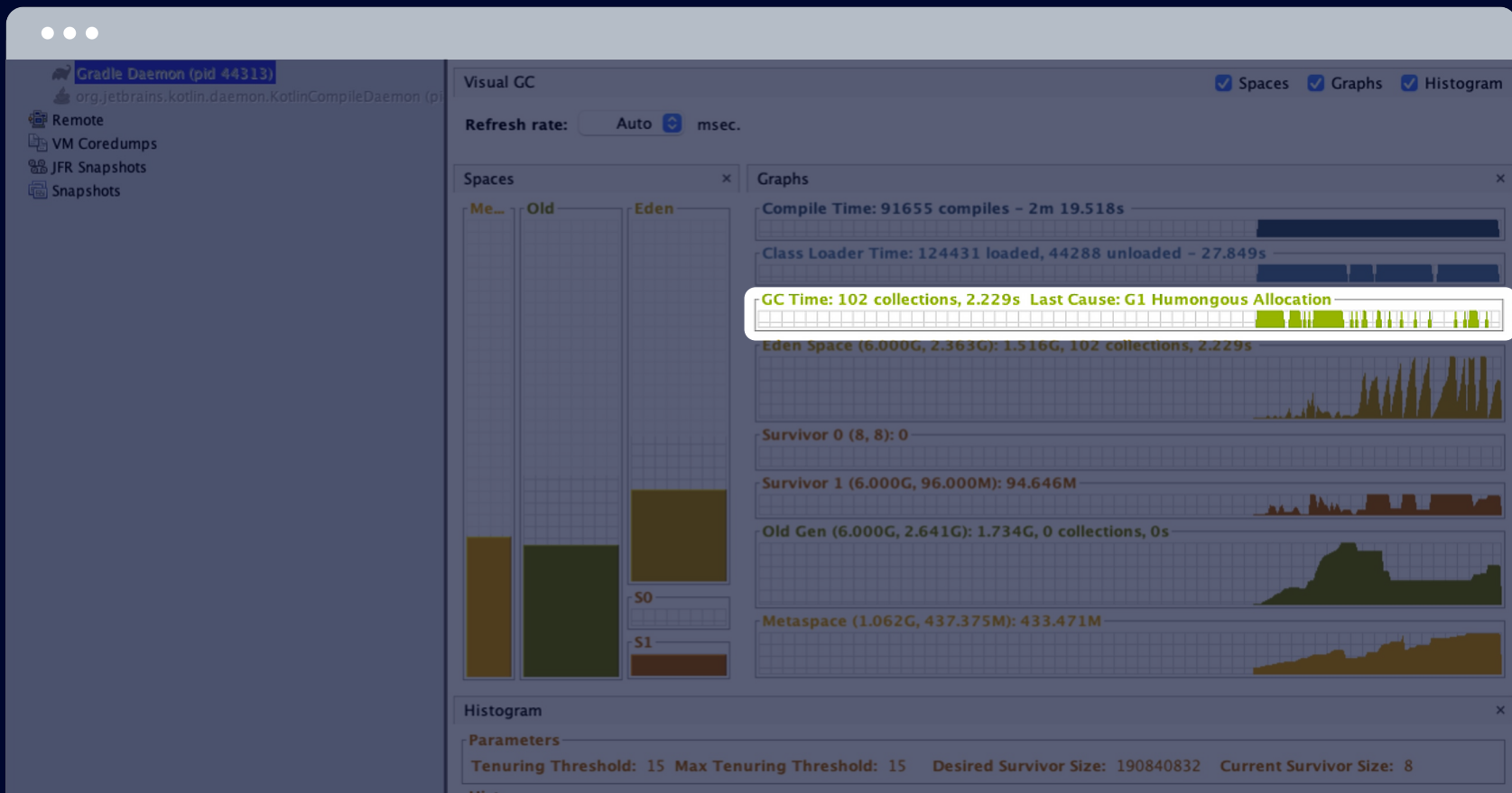


Gradle демон: Visual GC

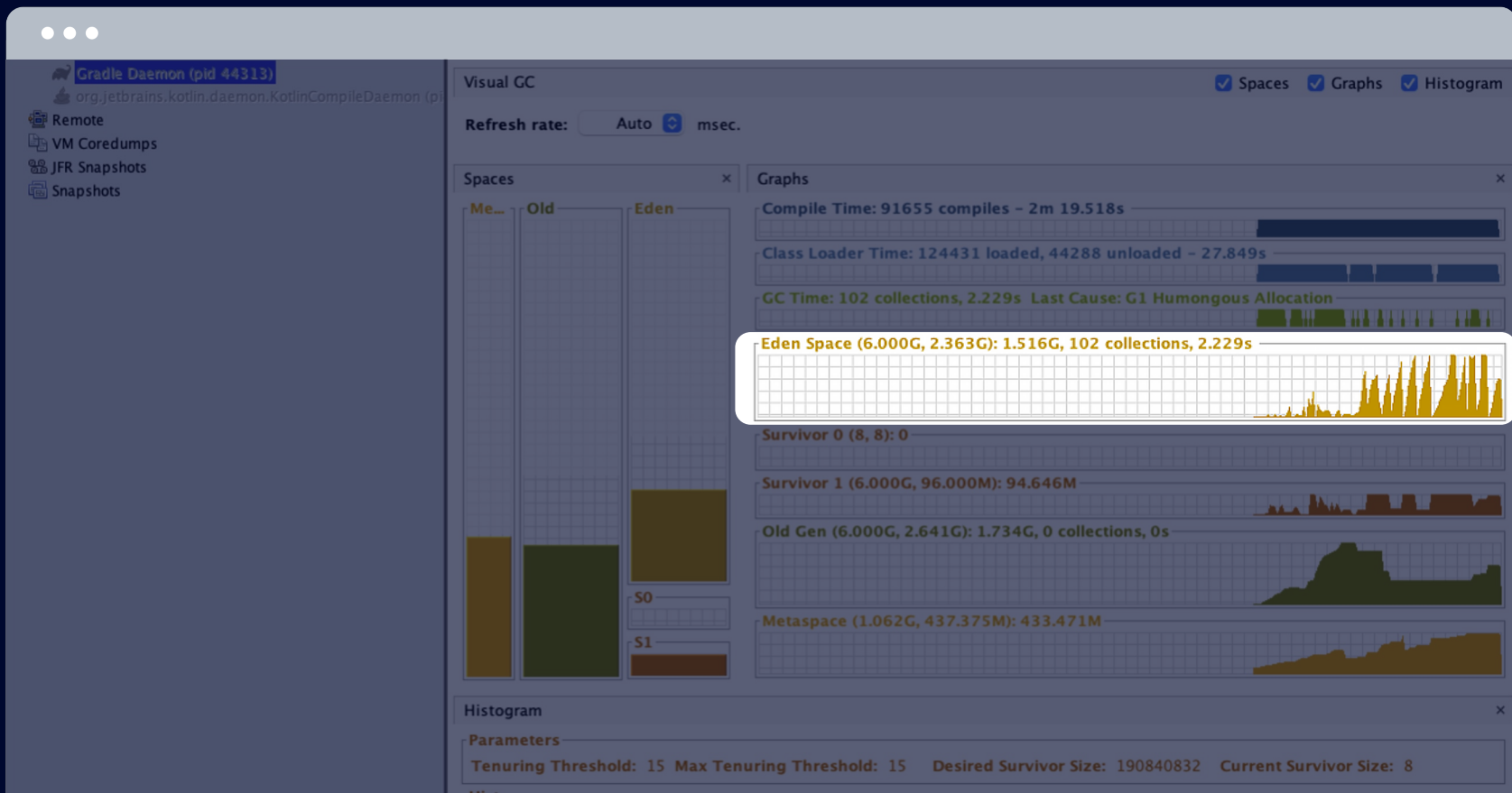
The screenshot displays the Visual GC tool interface for a Gradle daemon (pid 44313). The interface is divided into several sections:

- Left Panel:** A sidebar with navigation options: Remote, VM Coredumps, JFR Snapshots, and Snapshots.
- Visual GC Header:** Includes a title bar, checkboxes for Spaces, Graphs, and Histogram, and a Refresh rate dropdown set to Auto msec.
- Spaces:** A bar chart showing memory usage for Me..., Old, and Eden spaces. Below the chart are labels for S0 and S1.
- Graphs:** A series of horizontal bar and area charts showing performance metrics:
 - Compile Time: 91655 compiles - 2m 19.518s
 - Class Loader Time: 124431 loaded, 44288 unloaded - 27.849s
 - GC Time: 102 collections, 2.229s Last Cause: G1 Humongous Allocation
 - Eden Space (6.000G, 2.363G): 1.516G, 102 collections, 2.229s
 - Survivor 0 (8, 8): 0
 - Survivor 1 (6.000G, 96.000M): 94.646M
 - Old Gen (6.000G, 2.641G): 1.734G, 0 collections, 0s
 - Metaspace (1.062G, 437.375M): 433.471M
- Histogram:** A section for parameters:
 - Parameters
 - Tenuring Threshold: 15 Max Tenuring Threshold: 15 Desired Survivor Size: 190840832 Current Survivor Size: 8

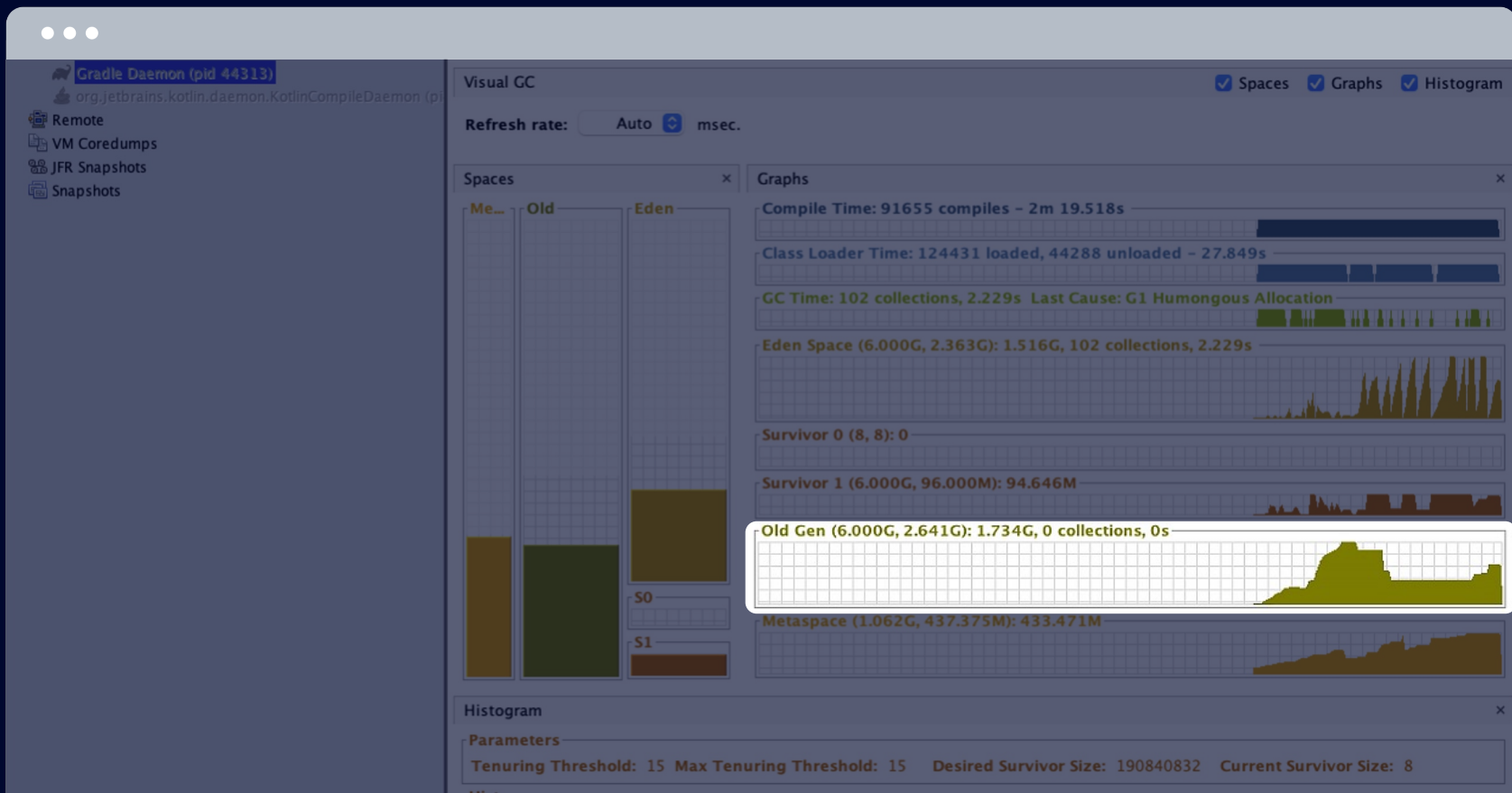
Gradle демон: Visual GC



Gradle демон: Visual GC



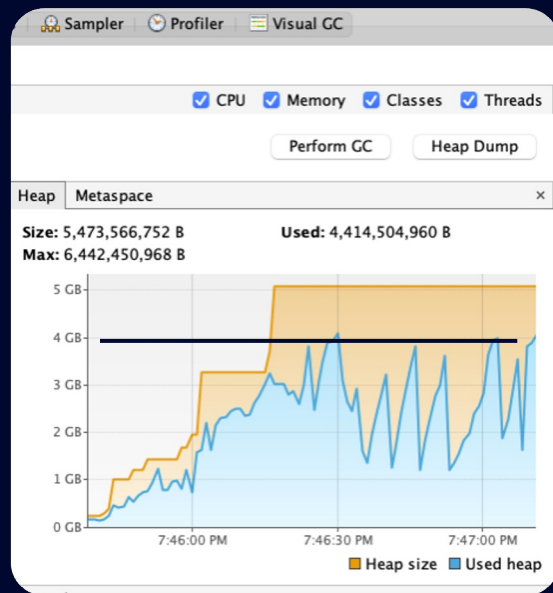
Gradle демон: Visual GC



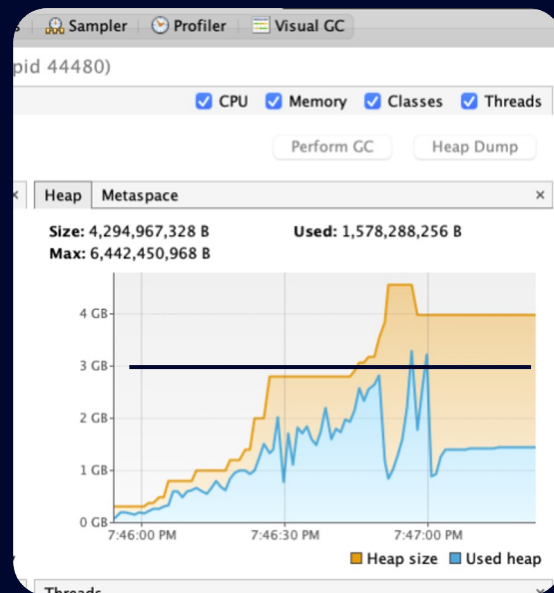
Kotlin демон: Monitor



Ограничение Java heap



Gradle daemon
-Xmx4g



Kotlin daemon
-Xmx3g

Опции JVM для Kotlin демона

gradle.properties

```
org.gradle.jvmargs=-Xmx4g -Dfile.encoding=UTF-8  
kotlin.daemon.jvmargs=-Xmx3g
```

Способы задания опций:

- Gradle daemon arguments
- kotlin.daemon.jvm.options
- kotlin.daemon.jvmargs
- kotlin extension
- Specific task definition



Влияние на перфоманс



Размер Java Heap

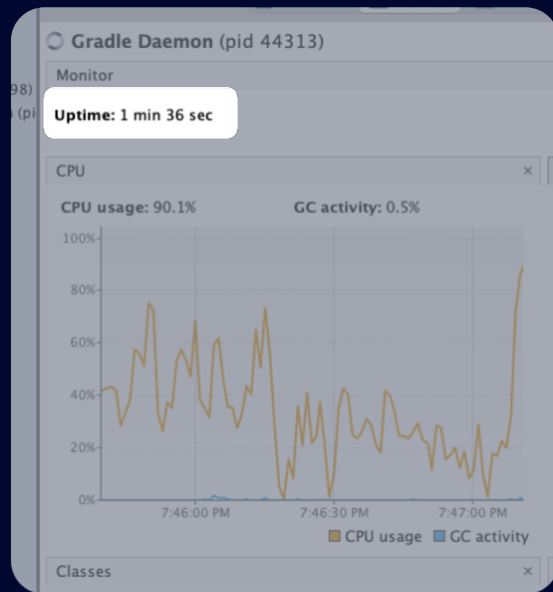
Производительность обратно пропорциональна количеству доступной памяти.



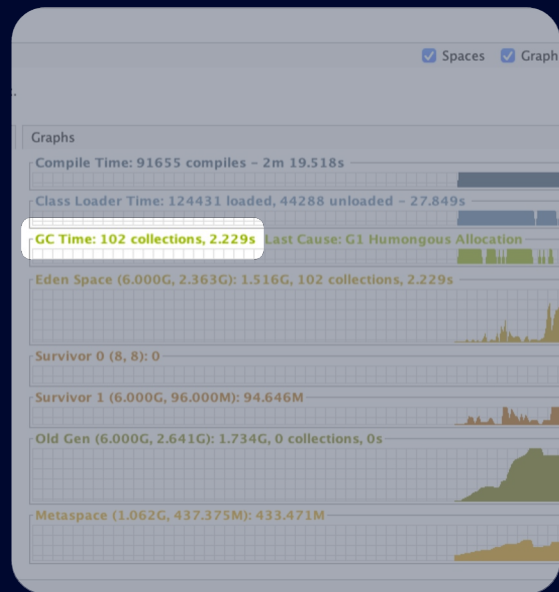
Размер Young Generation

Больше размер, реже срабатывает Young GC.
При этом чаще срабатывает Full GC.

Ключевые параметры

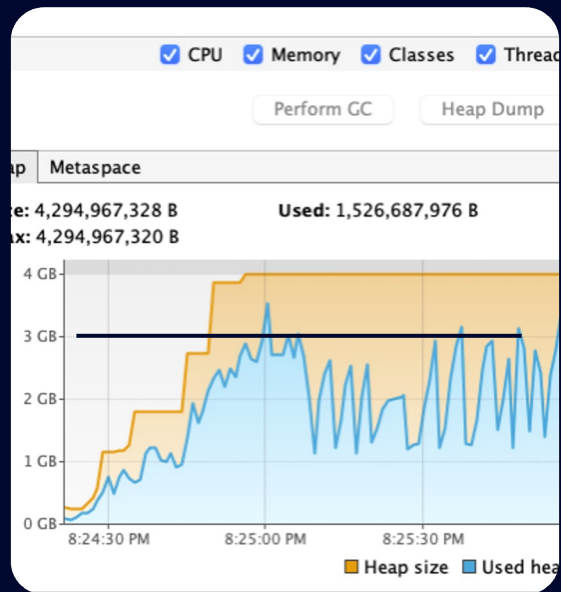


Время работы
процесса

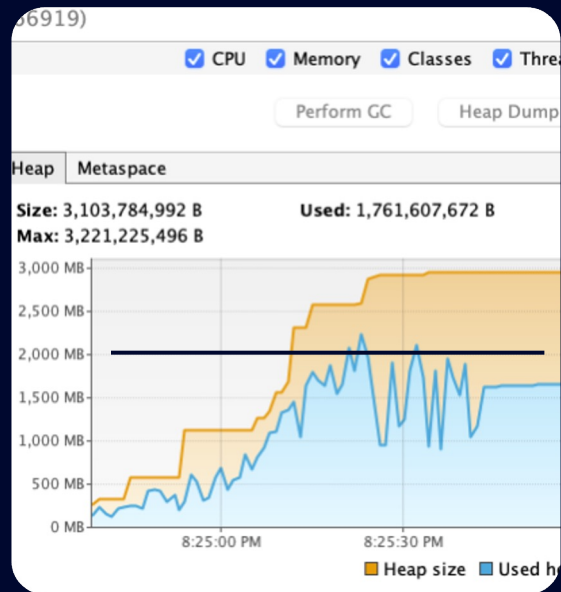


Время сборки
мусора

Ограничение Java heap



Gradle daemon
-Xmx3g



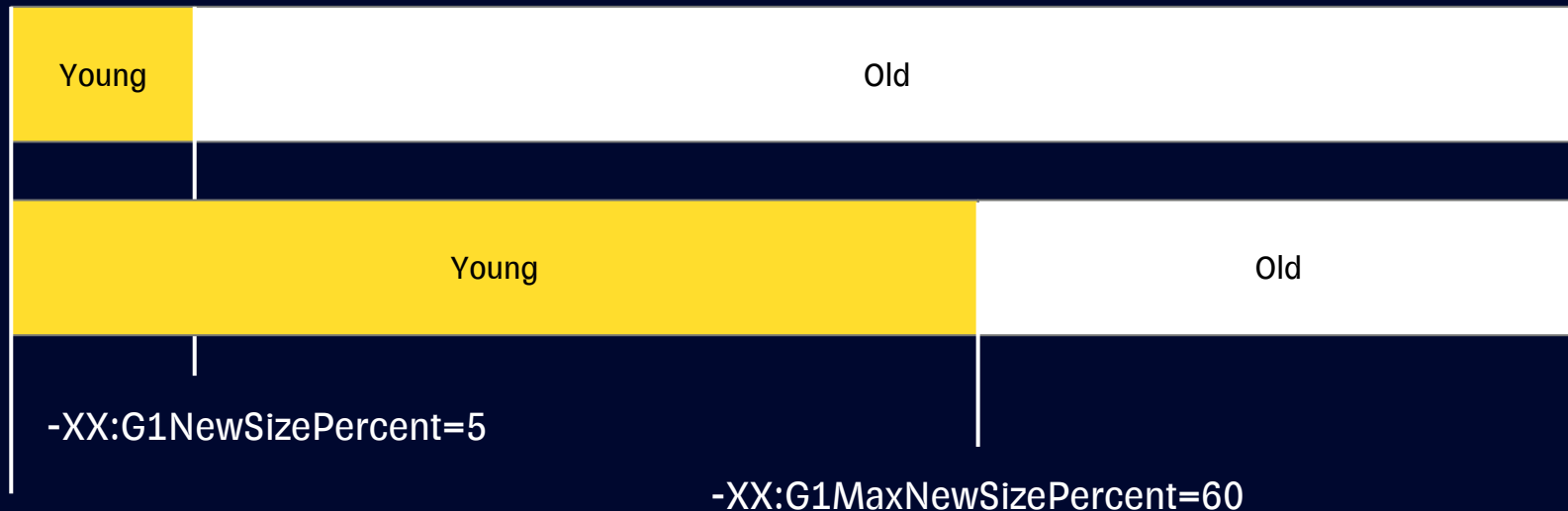
Kotlin daemon
-Xmx2g

Предел уменьшения Java heap

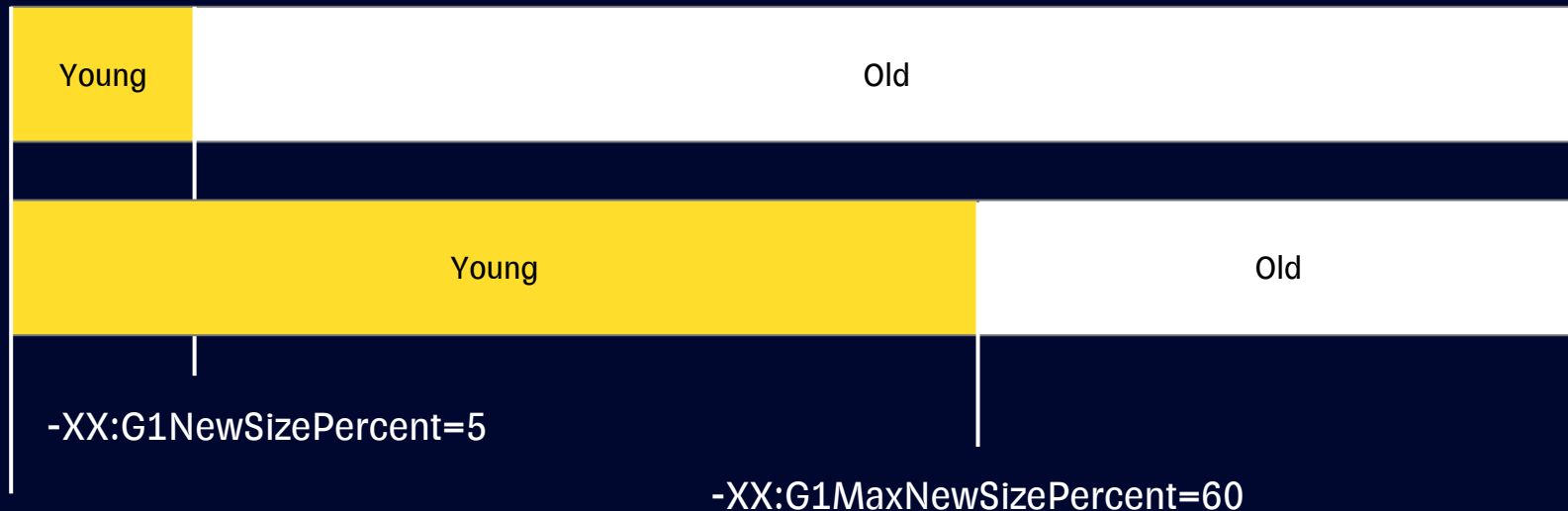


Все «живые» объекты + 10-20%

Garbage-First: размер Young Generation

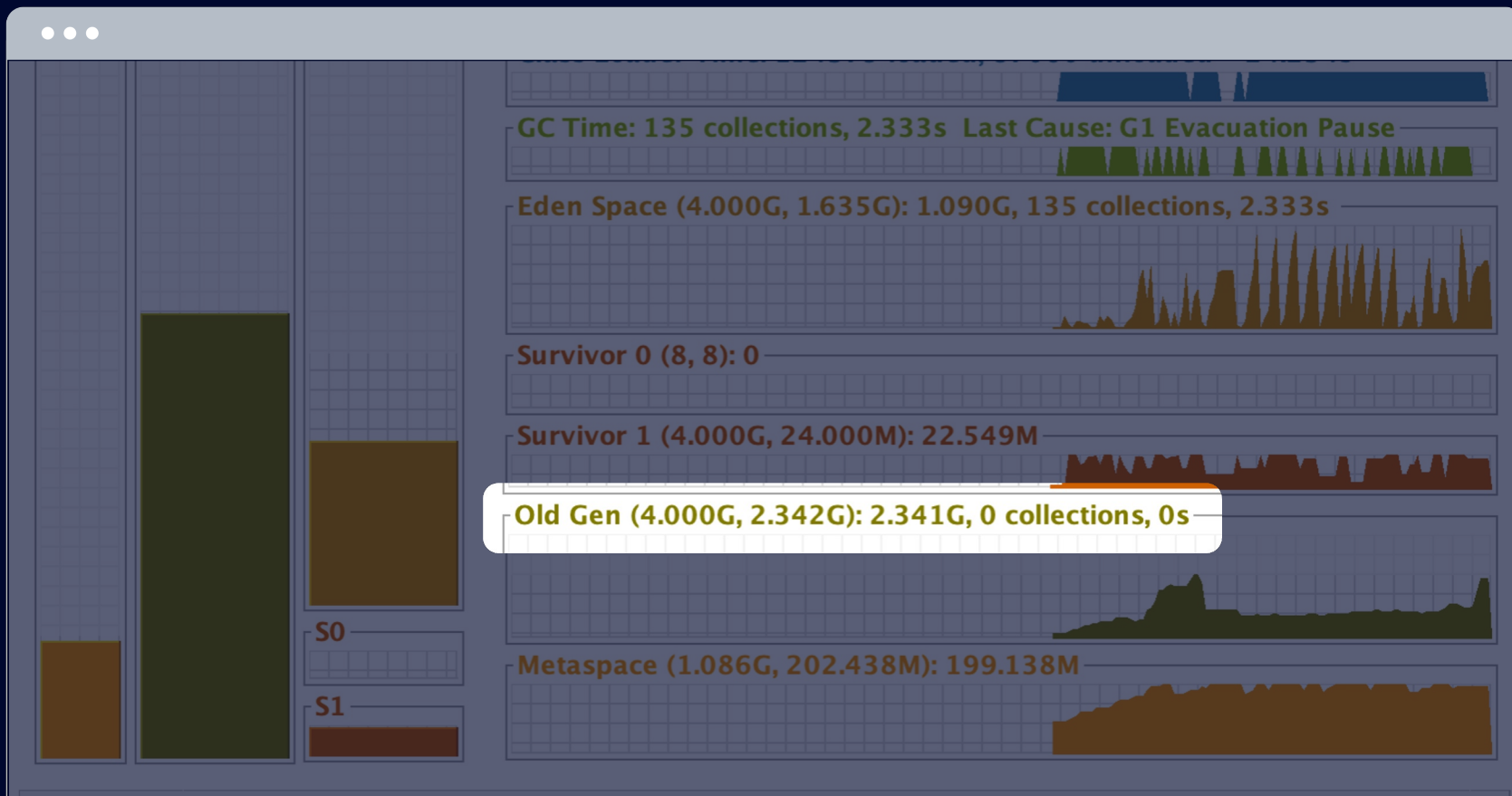


Garbage-First: размер Young Generation



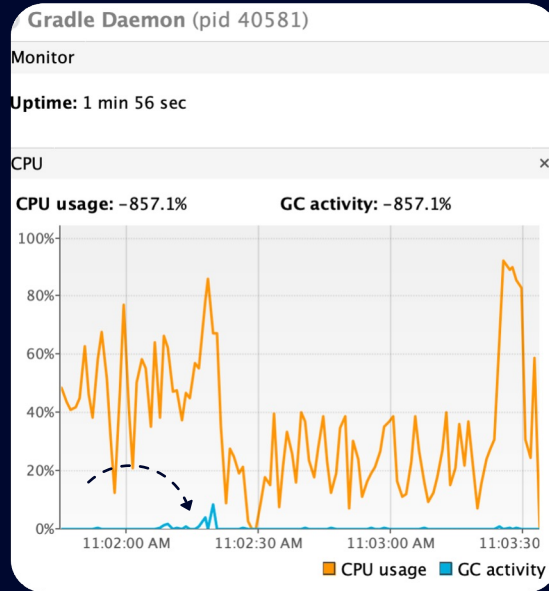
Heap size = <Old Gen used> + 5%

Visual GC: Gradle daemon

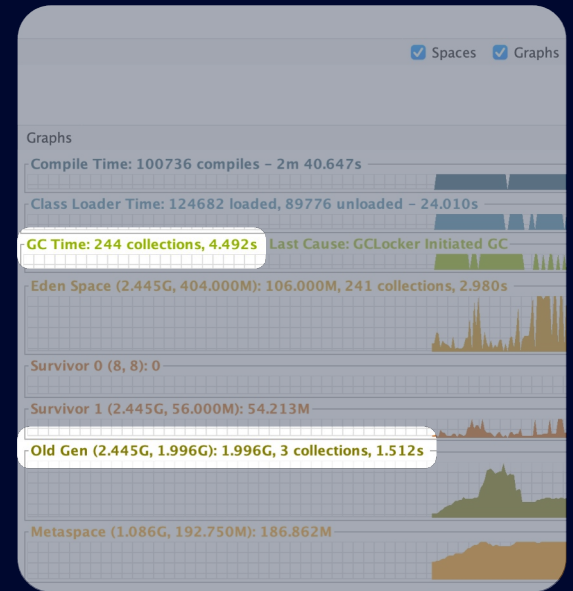


Gradle

-Xmx2500m

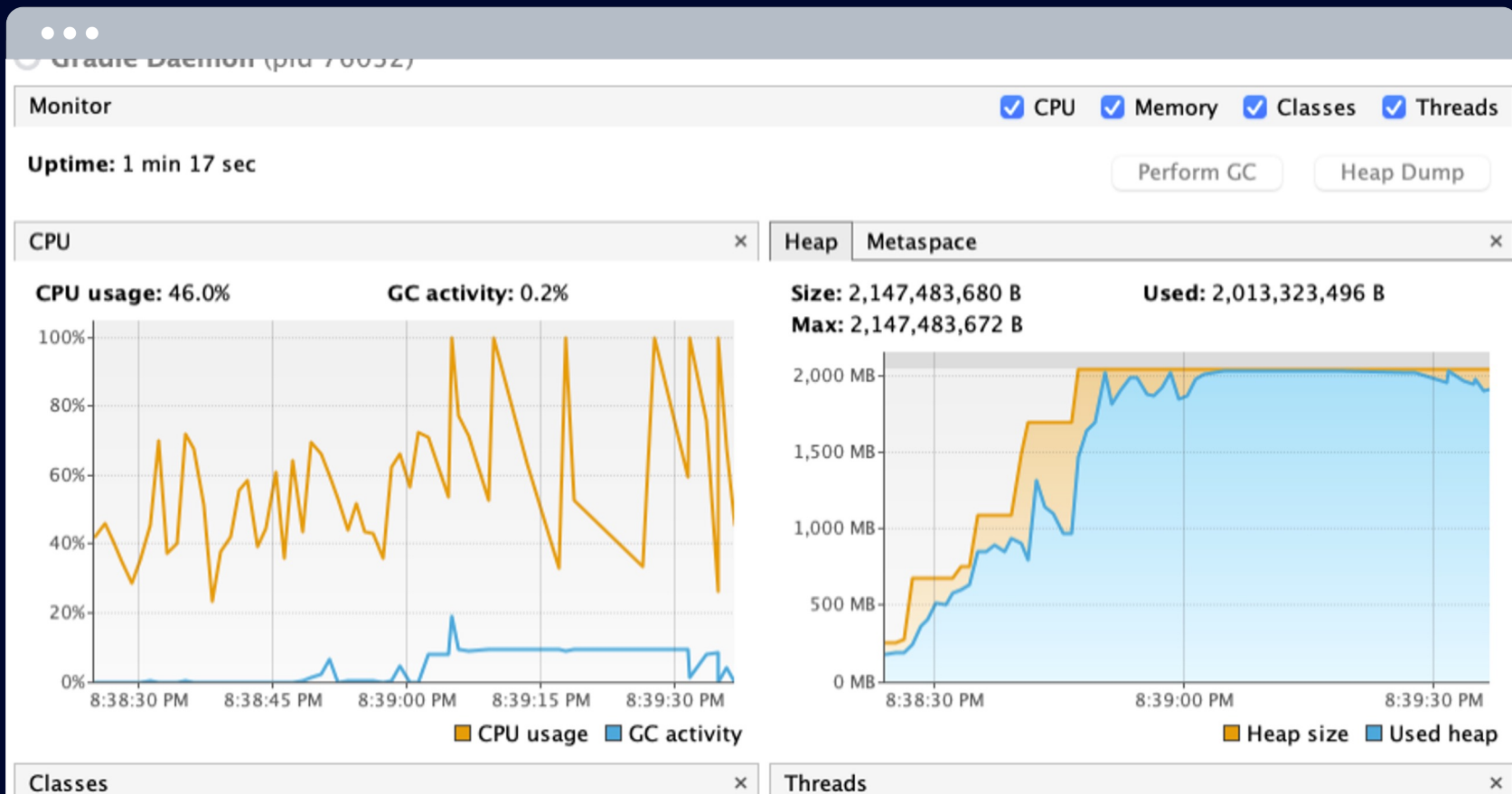


Monitor

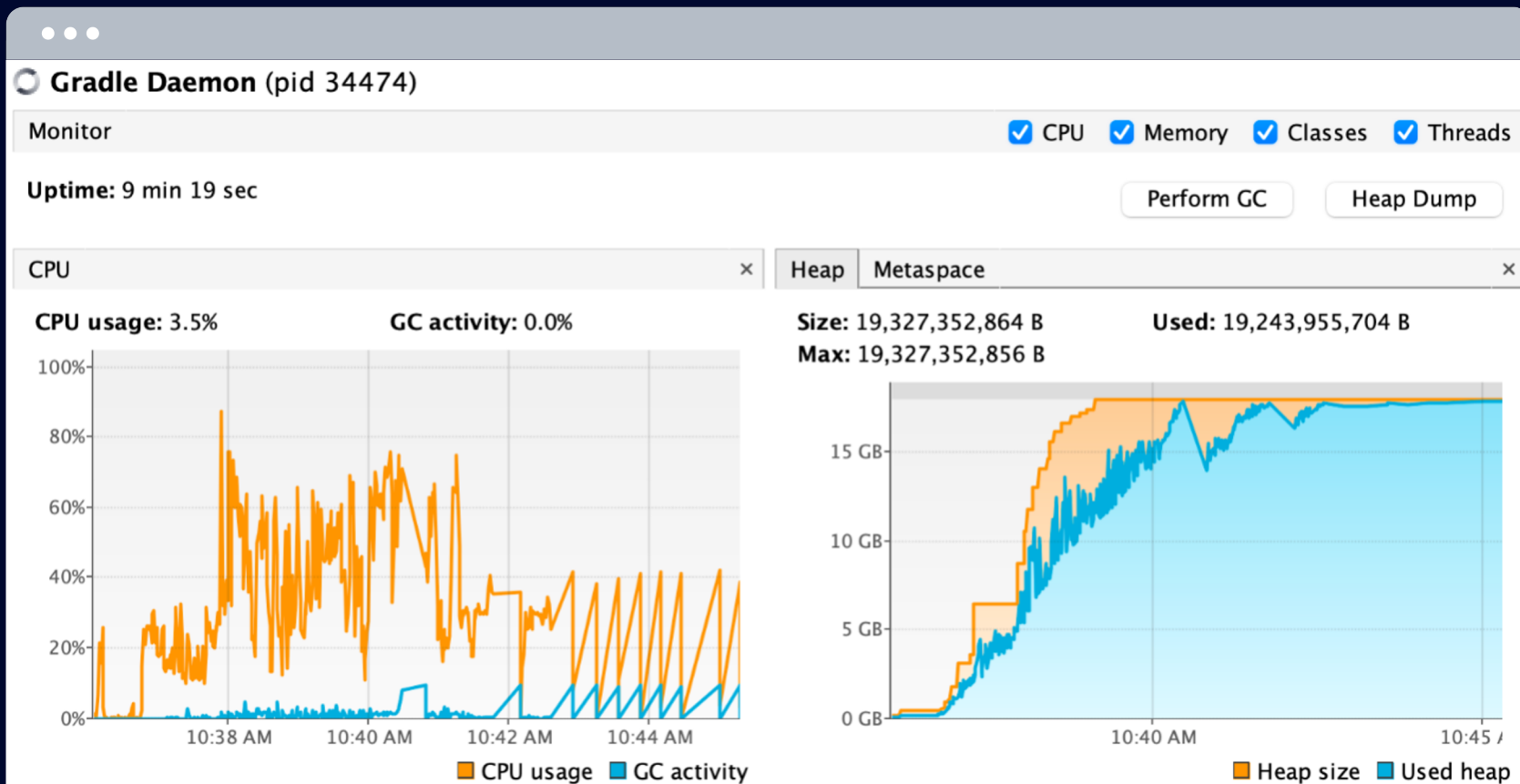


Visual GC

Ошибка: OutOfMemoryError



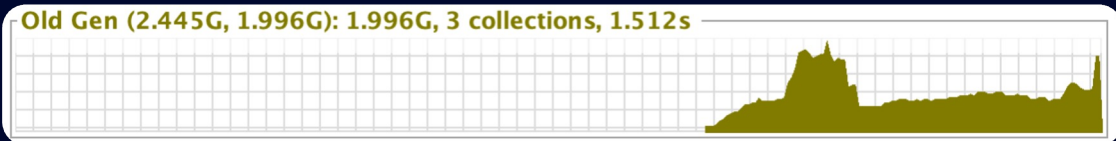
Время стремится в бесконечность



Всплески на графиках

Узкое место —
задачи, которые
затрагивают всю
кодovou базу

Old Gen (2.445G, 1.996G): 1.996G, 3 collections, 1.512s



:app:processDebugResources

:app:dexBuilderDebug

:app:mergeExtDexDebug

:app:packageDebug

Зависят от размера кодовой базы

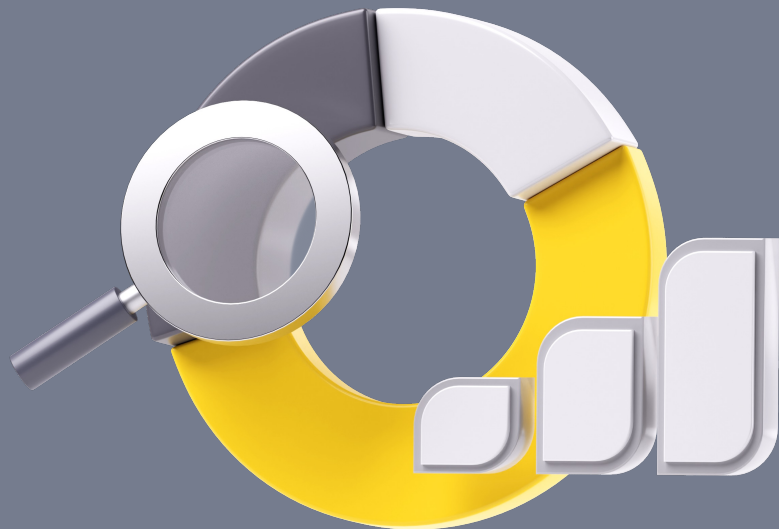
- ✓ Синхронизация с IDE
- ✓ Сборка релизной версии
- ✓ Статический анализ (Lint)

Тестирование результатов

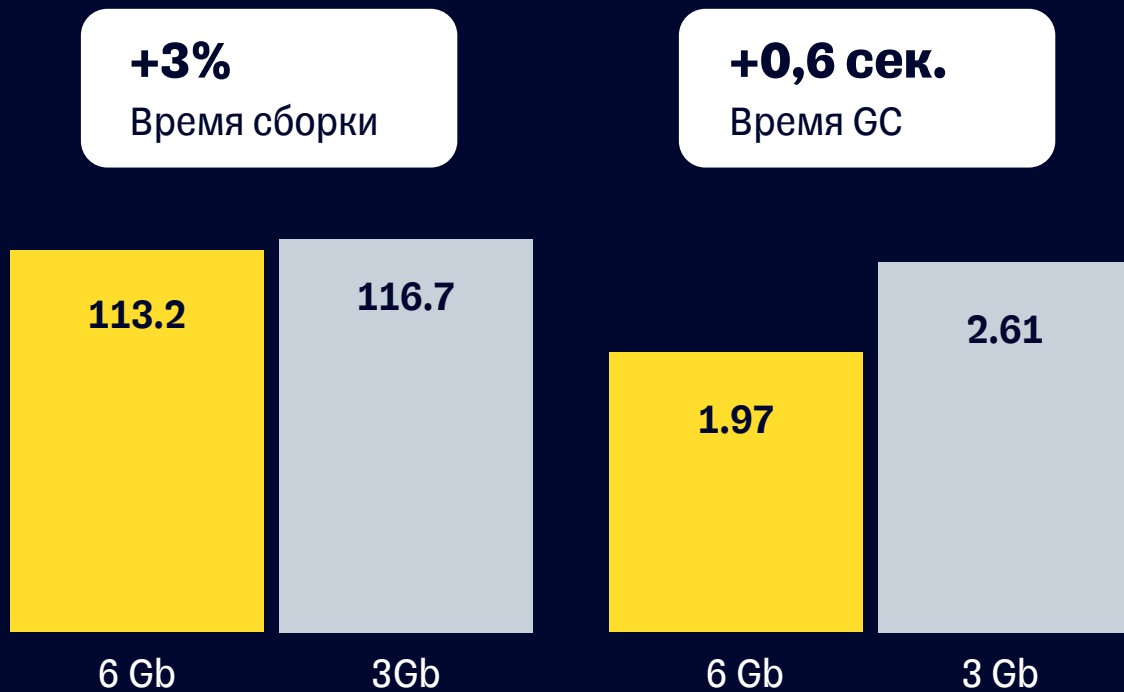
```
org.gradle.jvmargs=-Xmx3g -Dfile.encoding=UTF-8  
kotlin.daemon.jvmargs=-Xmx1g  
...
```

Gradle Profiler

- 2 прогревочных запуска
- 10 замеров Apple
- M1 Max
- 10 воркеров



G1 GC: изменение Heap size



Влияние размера памяти на время сборки

Gradle

+100%

Увеличение размера
Java heap для Gradle
демона

Gradle + Kotlin

+200%

Увеличение размера
Java heap с учетом
тюнинга Kotlin демона

Время сборки

-3%

Сокращение времени
сборки

**«Профит от увеличения
памяти имеет предел.
С определенного момента
перестает влиять на время
сборки.»**

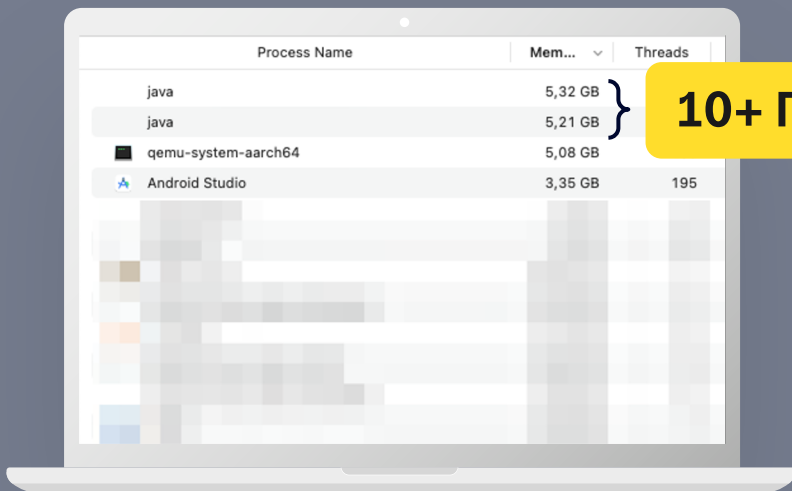


Использование RAM

```
org.gradle.jvmargs=-Xmx6g
```



```
org.gradle.jvmargs=-Xmx3g  
kotlin.daemon.jvmargs=-Xmx1g
```



3+1 ? 4



ТОП процессов (по памяти)

Process Name	Mem...	Threads	Ports	PID	User
qemu-system-aarch64	5,09 GB	100	450	30046	
java	3,74 GB	152	391	36863	
Android Studio	3,35 GB	183	633	29384	
java	1,71 GB	117	312	37087	

```
org.gradle.jvmargs=-Xmx3g -Dfile.encoding=UTF-8  
kotlin.daemon.jvmargs=-Xmx1g  
...
```

JVM Runtime



- ✓ Java Heap
- ✓ Class Loading
- ✓ JIT Compilation
- ✓ Threads

JVM Runtime



Java Heap



Heap memory



Class Loading



JIT Compilation



Non-Heap memory



Threads

Non-heap memory



- ✓ **Metaspace**
- ✓ **Code Cache**
- ✓ **Thread stack**
- ✓ **Garbage collection**

Native Memory Tracking

```
org.gradle.jvmargs=-Xmx3g ... -XX:NativeMemoryTracking=summary  
kotlin.daemon.jvmargs=-Xmx1g -XX:NativeMemoryTracking=summary
```

Enabling NMT causes a 5% - 10%
performance overhead



Native Memory Tracking

```
org.gradle.jvmargs=-Xmx3g ... -XX:NativeMemoryTracking=summary  
kotlin.daemon.jvmargs=-Xmx1g -XX:NativeMemoryTracking=summary
```

Enabling NMT causes a 5% - 10%
performance overhead

```
jcmd <PID> VM.native_memory summary
```



NMT Gradle daemon

Total: reserved=5402797KB, committed=4156941KB

- Java Heap (reserved=3145728KB, committed=3145728KB)
- Class (reserved=5976KB, committed=5976KB)
 - (Metadata:)
 - (reserved=303104KB, committed=174272KB)
 - (Class space:)
 - (reserved=1048576KB, committed=28864KB)
- Thread (reserved=324289KB, committed=324289KB)
- Code (reserved=259561KB, committed=162249KB)
- GC (reserved=205770KB, committed=205770KB)
- Symbol (reserved=34808KB, committed=34808KB)

**«Для сборки нужно больше
памяти, чем сумма Java heap
для Gradle и Kotlin
процессов.»**



Build Performance Analyzer

The screenshot shows the Build Performance Analyzer tool interface. At the top, there are three window control buttons. Below them is a search bar with the text "Search Everywhere Double ⌘". Below the search bar are three menu items: "Go to File ⌘⇧O", "Recent Files ⌘E", and "Navigation Bar ⌘↑".

The main content area is divided into three sections:

- Overview** (selected in the dropdown menu):
 - Build finished on 09.10.2023, 19:47**
Total build duration was 96.3s
 - Includes:**
Build configuration: 3.3s - [Optimize this](#)
Critical path tasks execution: 92.6s
- Common views into this build**
 - [Tasks impacting build duration](#)
 - [Plugins with tasks impacting build duration](#)
 - [All warnings](#)
- Gradle Daemon Memory Utilization**
 - 2.6% (2.5s) of your build's time was dedicated to garbage collection during this build.
You can change the Gradle daemon heap size on the memory settings page.
 - [Edit memory settings](#)

A tooltip is displayed over the "Edit memory settings" button, containing the following text:

The default garbage collector was used in this build running with JDK 17.
Note that the default GC was changed starting with JDK 9. This could impact your build performance by as much as 10%.
Recommendation: [Fine tune your JVM](#) .
[Don't show this again.](#)

The bottom status bar contains the following icons and text: Git, Profiler, Logcat, App Quality Insights, Build, TODO, Problems, Terminal, Services, App Inspection, and Layout Inspector.

Build Performance Analyzer

The screenshot displays the Build Performance Analyzer interface. At the top, there are three window control buttons. Below them is a search bar with the text "Search Everywhere Double ⌘". Below the search bar are three menu items: "Go to File ⌘⇧O", "Recent Files ⌘E", and "Navigation Bar ⌘↑".

The main content area is divided into three columns:

- Left Column:** Shows build completion information: "Build finished on 09.10.2023, 19:47" and "Total build duration was 96.3s". Below this, it lists "Includes: Build configuration: 3.3s - Optimize this" and "Critical path tasks execution: 92.6s".
- Middle Column:** Titled "Common views into this build", it lists "Tasks impacting build duration", "Plugins with tasks impacting build duration", and "All warnings".
- Right Column:** Titled "Gradle Daemon Memory Utilization", it states "2.6% (2.5s) of your build's time was dedicated to garbage collection during this build." and "You can change the Gradle daemon heap size on the memory settings page." Below this is a button "Edit memory settings".

A white tooltip box is overlaid on the right column, containing the text: "The default garbage collector was used in this build running with JDK 17. Note that the default GC was changed starting with JDK 9. This could impact your build performance by as much as 10%. **Recommendation:** Fine tune your JVM . Don't show this again."

A large yellow box at the bottom of the right column contains the text: "Parallel Collector (Throughput collector)".

The bottom of the interface shows a sidebar with "Structure", "Build Variants", and "Bookmarks". The bottom status bar includes icons for "Git", "Profiler", "Logcat", "App Quality Insights", "Build", "TODO", "Problems", "Terminal", "Services", "App Inspection", and "Layout Inspector". On the right side, there are vertical icons for "Notifications", "Gradle", "Device Explorer", and "Running Devices".

Эргономика

Parallel Collector



Maximum Pause-Time Goal

-XX:MaxGCPauseMillis=<MAX_ULONG>



Throughput Goal

-XX:GCTimeRatio=99

$$\frac{1}{\text{GCTimeRatio} + 1} = \frac{1}{99 + 1} = 1\%$$

Parallel GC

```
org.gradle.jvmargs=-Xmx3g -Dfile.encoding=UTF-8 -XX:+UseParallelGC  
kotlin.daemon.jvmargs=-Xmx1g -XX:+UseParallelGC
```

```
...
```


Parallel GC

```
org.gradle.jvmargs=-Xmx3g -Dfile.encoding=UTF-8 -XX:+UseParallelGC  
kotlin.daemon.jvmargs=-Xmx1g -XX:+UseParallelGC  
...
```

Как изменится время сборки?



Сократится

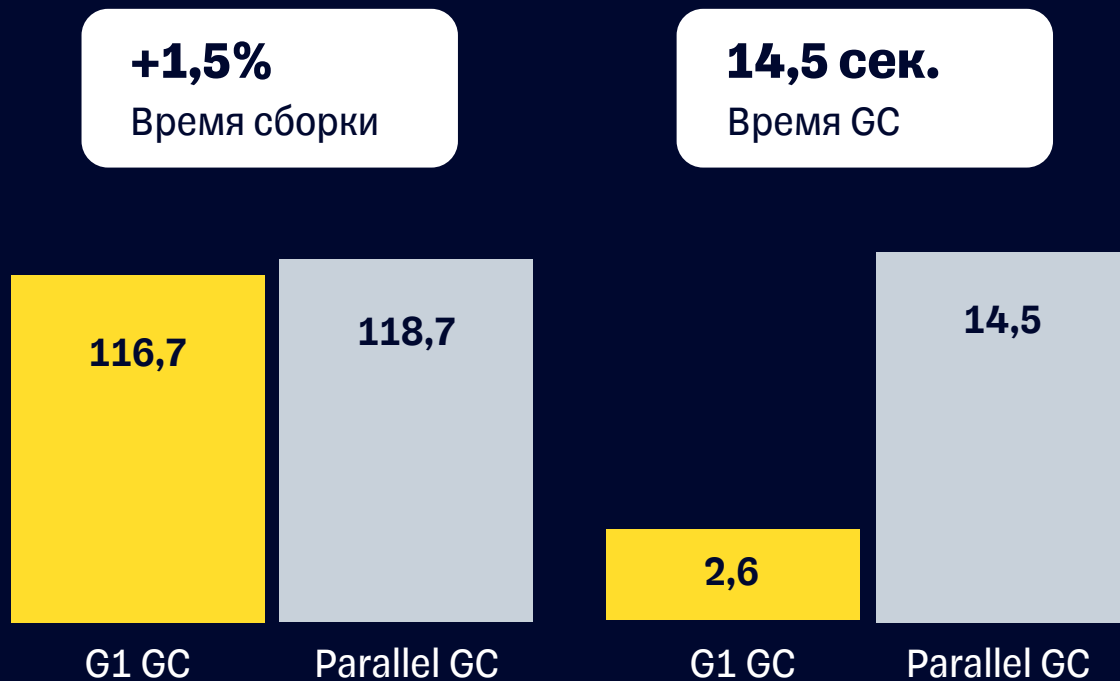


Увеличится

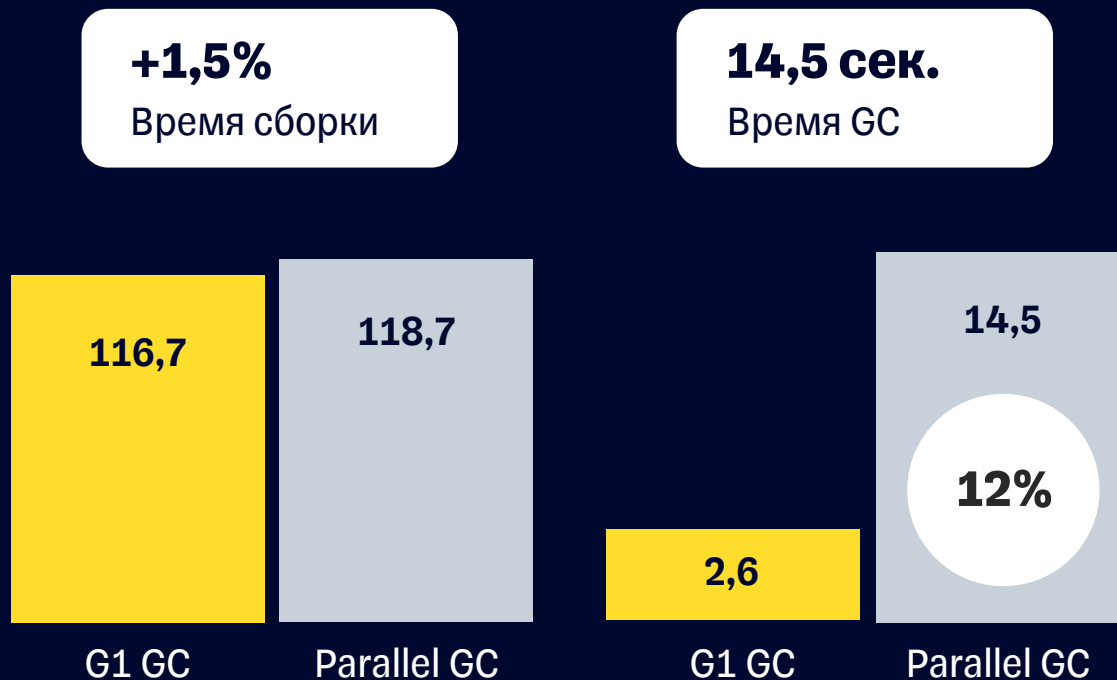


Сохранится

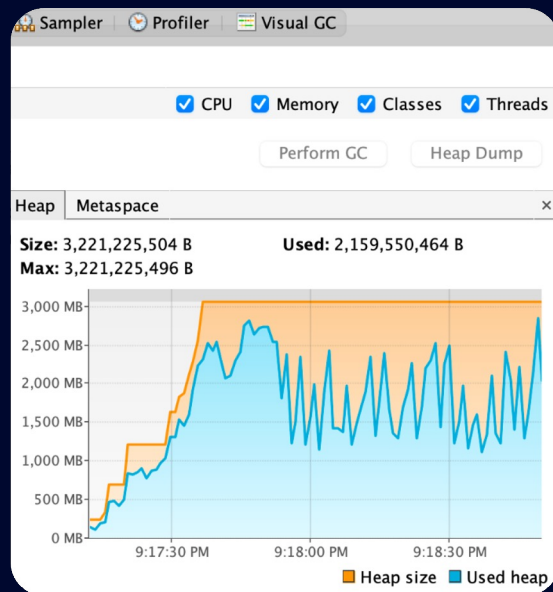
G1 GC ПРОТИВ Parallel GC



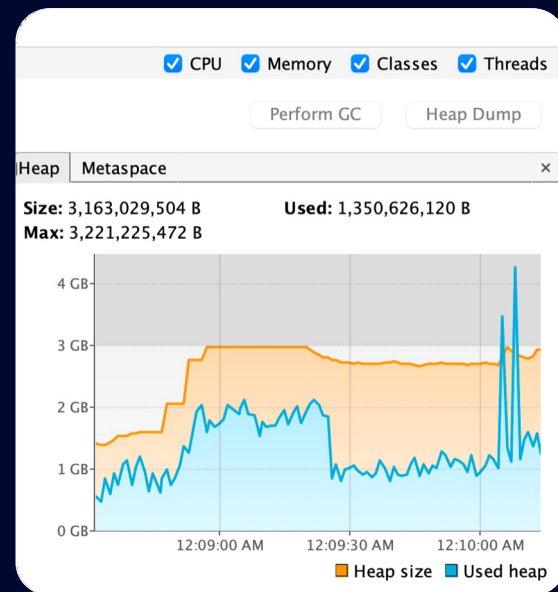
G1 GC ПРОТИВ Parallel GC



Размер Java heap Gradle демон



`-XX:+UseG1GC`



`-XX:+UseParallelGC`

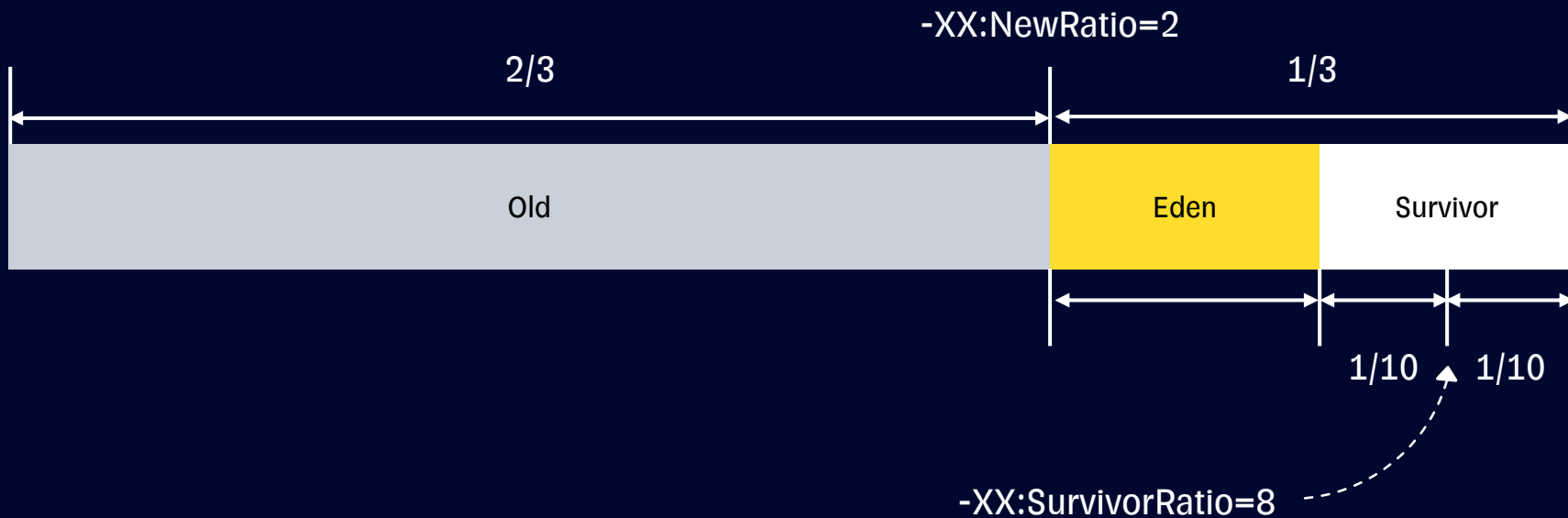
Parallel GC

Расположение блоков памяти



Parallel GC

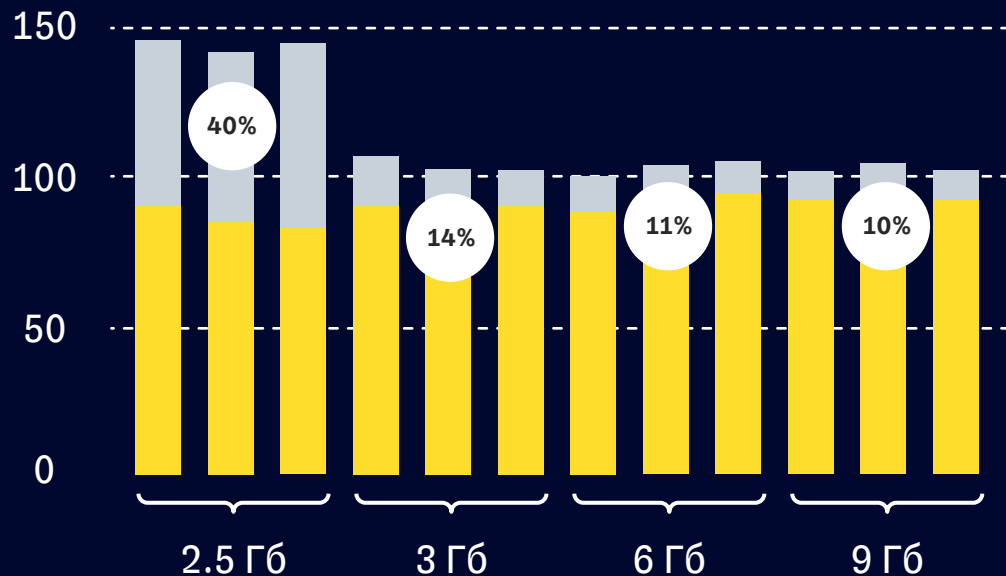
Расположение блоков памяти



Parallel GC: изменение Heap size

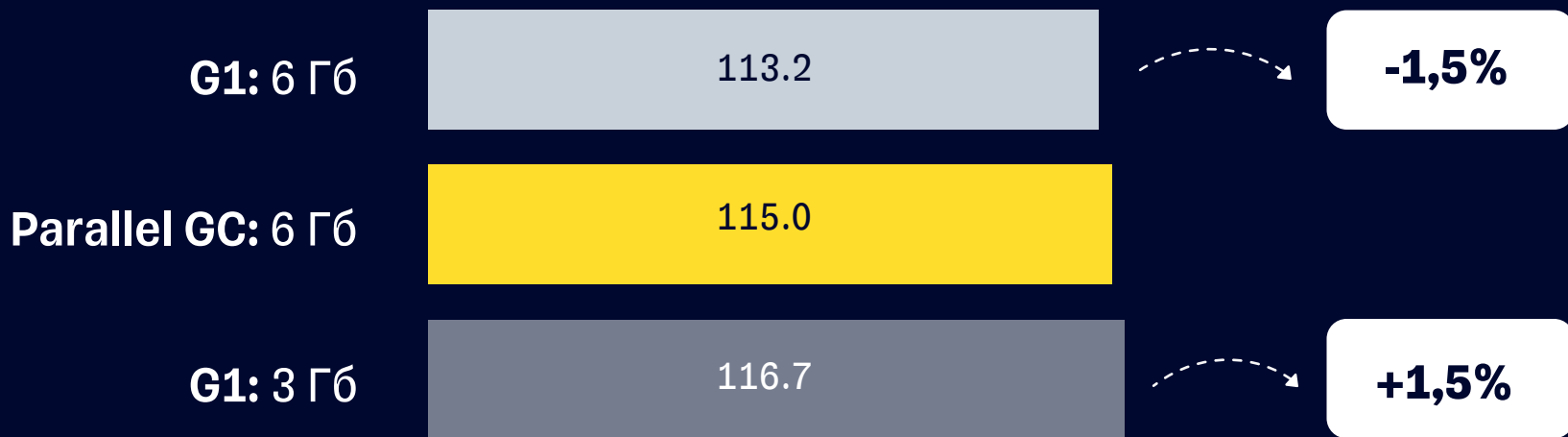


Время сборки vs. размер Java heap



- Прямая зависимость между временем сборки и GC
- Минимальное время сборки мусора — 10%

Parallel GC против Garbage-First



**«Любые советы по улучшению
производительности стоит
проверять на своем
окружении.»**





Заключение

Заключение



На время сборки
влияет размер Java
heap



Профит
от увеличения памяти
имеет предел



Выигрыш можно
оценить по времени
работы GC



Java процессу нужна
память для своих
задач



Проверяйте
советы из статей
по перфомансу

Заключение



На время сборки
влияет размер Java
heap



Профит
от увеличения памяти
имеет предел



Выигрыш можно
оценить по времени
работы GC



Java процессу нужна
память для своих
задач



Проверяйте
советы из статей
по перфомансу

Заключение



На время сборки
влияет размер Java
heap



Профит
от увеличения памяти
имеет предел



Выигрыш можно
оценить по времени
работы GC



Java процессу нужна
память для своих
задач



Проверяйте
советы из статей
по перфомансу

Заключение



На время сборки
влияет размер Java
heap



Профит
от увеличения памяти
имеет предел



Выигрыш можно
оценить по времени
работы GC



Java процессу нужна
память для своих
задач



Проверяйте
советы из статей
по перфомансу

Заключение



На время сборки
влияет размер Java
heap



Профит
от увеличения памяти
имеет предел



Выигрыш можно
оценить по времени
работы GC



Java процессу нужна
память для своих
задач



Проверяйте
советы из статей
по перфомансу

Устаревшая информация (Java 8)

The screenshot shows the Android Developers website page titled "Increase the JVM heap size". The page content includes a navigation menu on the left, a main article area, and a right-hand sidebar. A prominent white callout box contains a note about a Gradle issue. The main article text explains how to increase the JVM heap size to improve build speed. A code block shows the configuration for the JVM heap size. The sidebar lists various optimization tips.

developer.android.com/build/optimize-your-build#increase-the-jvm-heap-size

Developers Essentials Design & Plan Develop Google Play Search English Android Studio Sign In

Filter

What's new in Android build

Configure your build

Optimize your build

Overview

Troubleshoot build performance

Profile your build

Gradle tips and recipes

Manage manifest files

Shrink your app

D8 and R8 compiler version compatibility

Enable multidex

Extend your build

Publish your library

To measure build speed with different configurations, see [Profile your build](#).

★ **Note:** Be aware of Gradle [issue #19750](#): setting the `org.gradle.jvmargs` property can lead to "Daemon disappeared" failures. Until the bug is fixed you must explicitly set JVM argument defaults. The key JVM arguments that you must explicitly set are the `-XX:MaxMetaspaceSize=256m` and `-XX:+HeapDumpOnOutOfMemoryError` arguments; for the full list of arguments, see the [java command documentation](#).

Increase the JVM heap size

If you observe slow builds, and in particular the garbage collection takes more than 15% of the build time in your [Build Analyzer](#) results, then you should increase the Java Virtual Machine (JVM) heap size. In the `gradle.properties` file, set the limit to 4, 6, or 8 gigabytes as shown in the following example:

```
org.gradle.jvmargs=-Xmx6g
```

Then test for build speed improvement. The easiest way to determine the optimal heap size is to increase the limit by a small amount and then test for sufficient build speed improvement.

★ **Note:** If you change the default memory limit, you also have to set the `-XX:MaxMetaspaceSize=1g` argument due to [Gradle issue #19750](#).

On this page

- Optimize your build configuration
 - Keep your tools up to date
 - Use KSP instead of kapt
 - Avoid compiling unnecessary resources
 - Experiment with putting the Gradle Plugin Portal last
 - Use static build config values with your debug build
 - Use static dependency versions
- Create library modules
- Create tasks for custom build logic
- Convert images to WebP
- Disable PNG crunching
- [Experiment with the JVM parallel garbage collector](#)
- Increase the JVM heap size
- Use non-transitive R classes
- Use non-constant R classes

Полезные ССЫЛКИ



- [JDK 17 Documentation](#)
- [Garbage Collection Tuning Guide](#)
- [Java Virtual Machine Guide](#)
- [JDK Tool Specifications](#)
- [Troubleshooting Guide](#)

Памяти много не бывает



Google Chrome



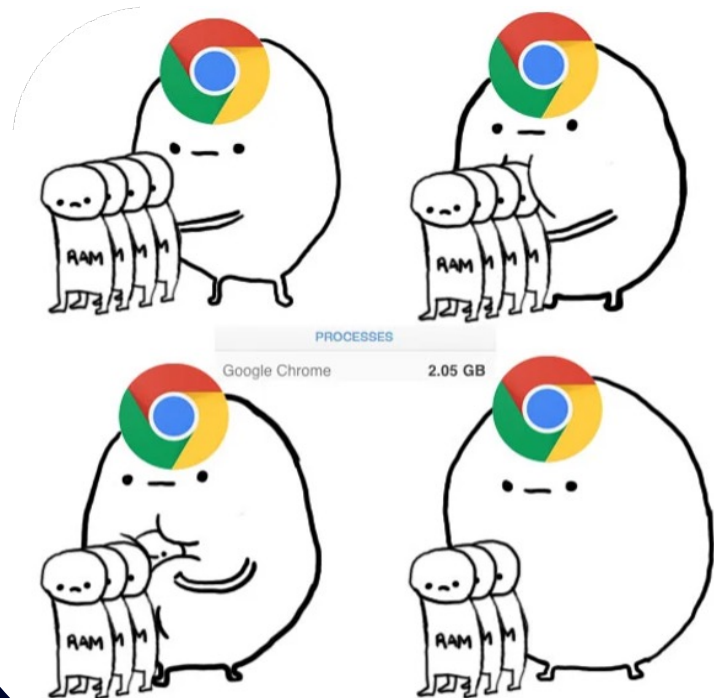
Android Studio



Android Emulator



Gradle



ТИНЬКОФФ

Слайды



Юрий Анисимов | Разработчик Инфра.

@ y.anisimov@tinkoff.ru

ТИНЬКОФФ