





ORACLE®

Диагностирование проблем и настройка GC в HotSpot JVM

Владимир Иванов

Hotspot JVM Quality Engineering, Oracle Corporation

vladimir.x.ivanov@oracle.com

Содержание

- **Кратко о сборке мусора**
- Мониторинг GC
- Настройка GC
 - Объем используемой памяти
 - Молодое поколение
 - Паузы
 - CMS
- G1
- Выводы



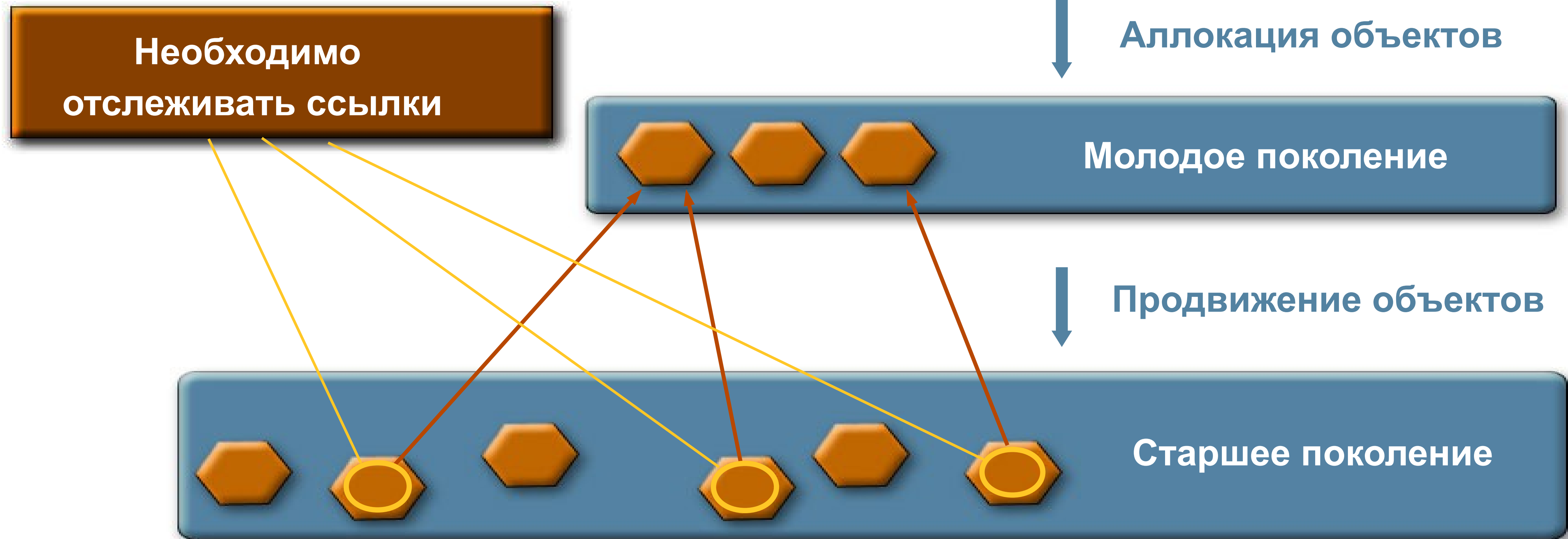
Сборка мусора (GC)

- Находит и освобождает место, занимаемое ненужными объектами
- Автоматическая и безопасная
- Проще, если граф объектов “заморожен”
 - Stop-the-world (STW) паузы
- Возможны различные подходы
 - с дефрагментацией или без
 - Алгоритмы: copying, mark-sweep, mark-compact, ...
 - Аллокация: linear, free lists, ...

Сборка мусора с поколениями (I)

- Слабая гипотеза о поколениях
 - Большинство объектов временные
 - Старые объекты редко ссылаются на молодые
- Молодые и старые объекты содержатся отдельно
 - В пространствах называемых “поколения” (generations)
 - Возможны разные алгоритмы для молодого и старого поколения
 - Молодое поколение можно собирать отдельно от старого

Сборка мусора с поколениями (II)



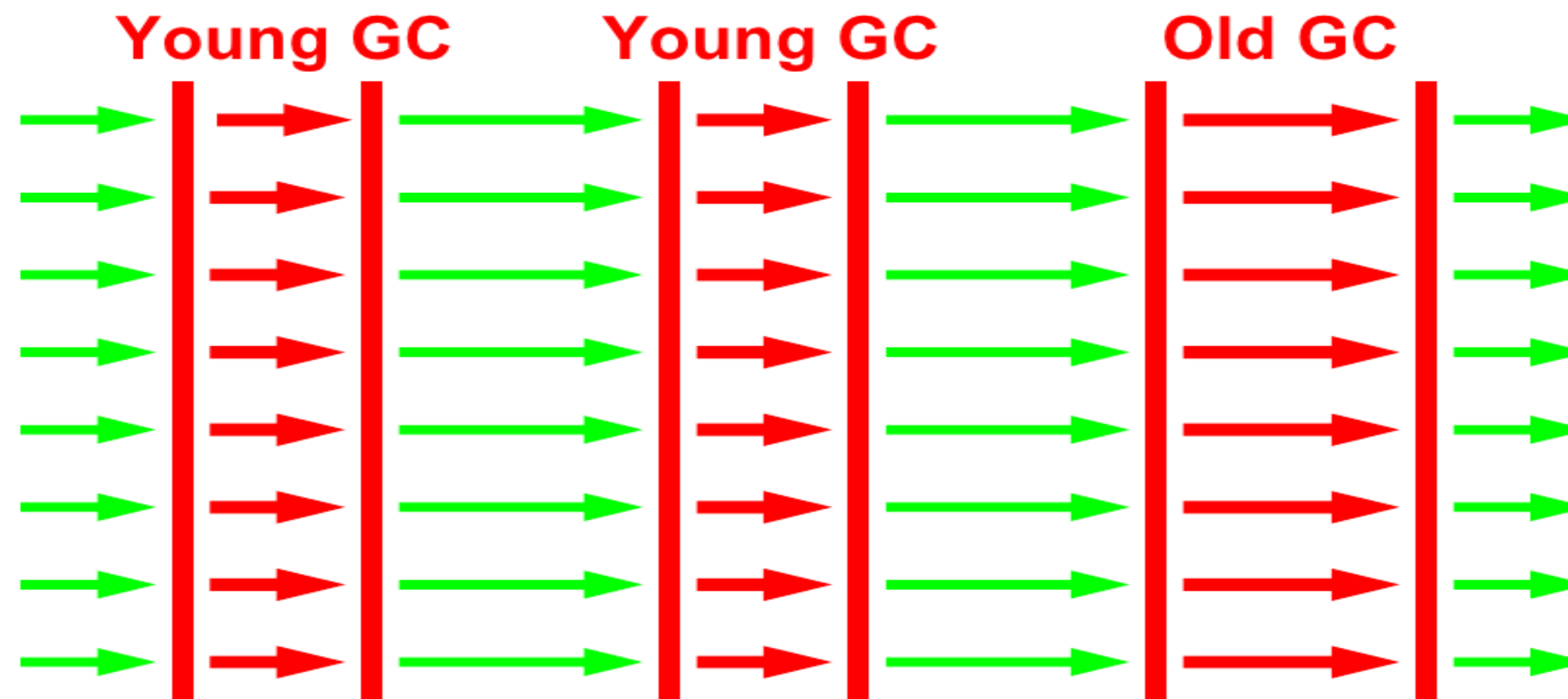
GC в Hotspot

- SerialGC
 - последовательная сборка молодого и старого поколений
- ParallelGC
 - максимальный throughput
 - параллельная сборка молодого и старого поколений
- CMS
 - предсказуемость
 - по возможности, сборка мусора в фоновом режиме
- G1
 - предсказуемость
 - слабо подвержен фрагментации

GC в Hotspot: о чем будем говорить

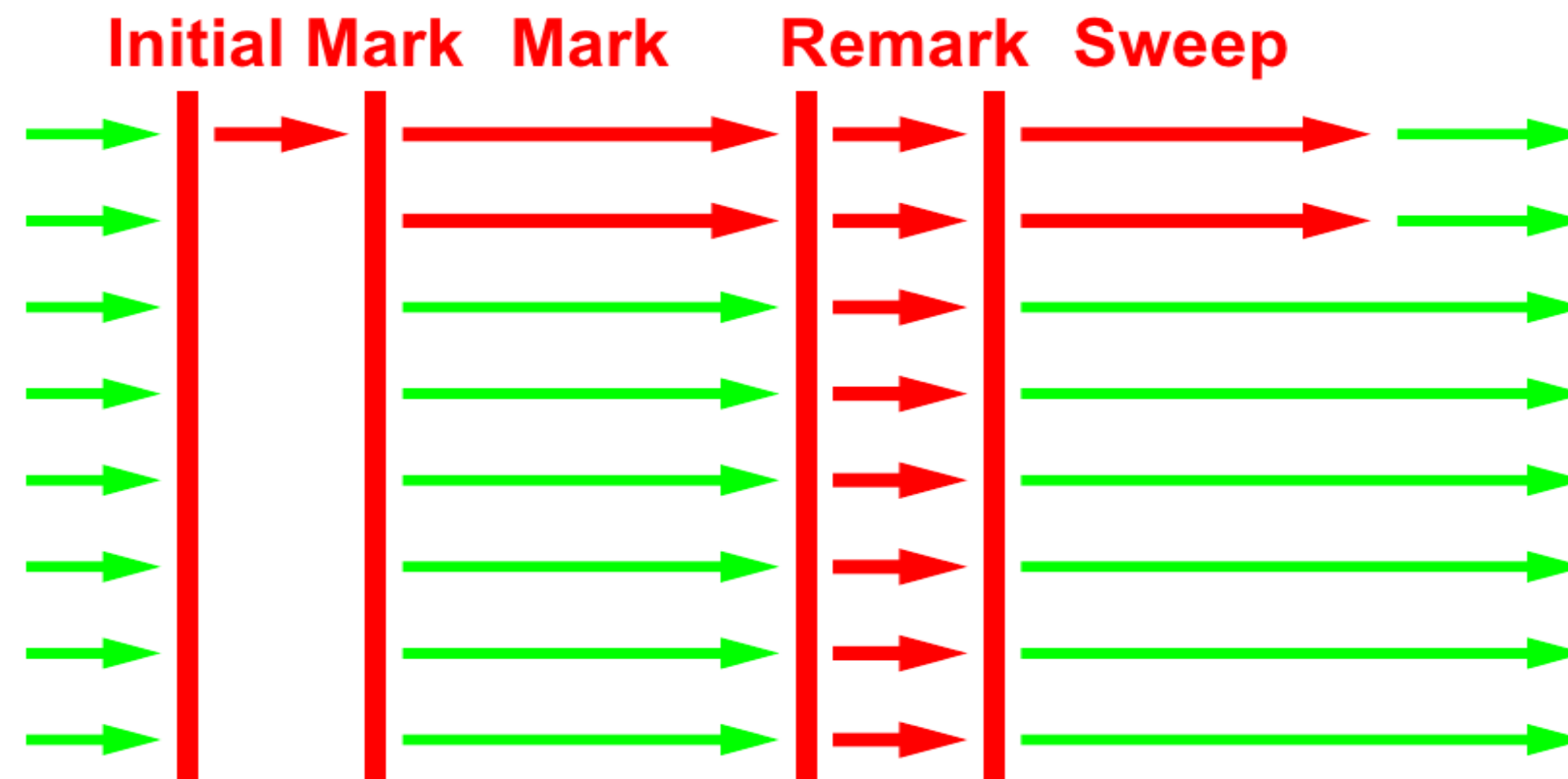
- **SerialGC**
 - последовательная сборка молодого и старого поколений
- **ParallelGC**
 - максимальный throughput
 - параллельная сборка молодого и старого поколений
- **CMS**
 - предсказуемость
 - по возможности, сборка мусора в фоновом режиме
- **G1**
 - предсказуемость
 - слабо подвержен фрагментации

ParallelGC: описание



- Молодое поколение: параллельный копирующий GC
- Старшее поколение: параллельный Mark-Compact GC
 - Аллокация: линейная
- `-XX:+UseParallelGC -XX:+UseParallelOldGC`

Concurrent Mark Sweep: описание



- Молодое поколение: параллельный копирующий GC
- Старшее поколение: фоновый Mark-Sweep GC
 - Аллокация: free-листы
 - Компактификация только при FullGC
- -XX:+UseConcMarkSweepGC

Содержание

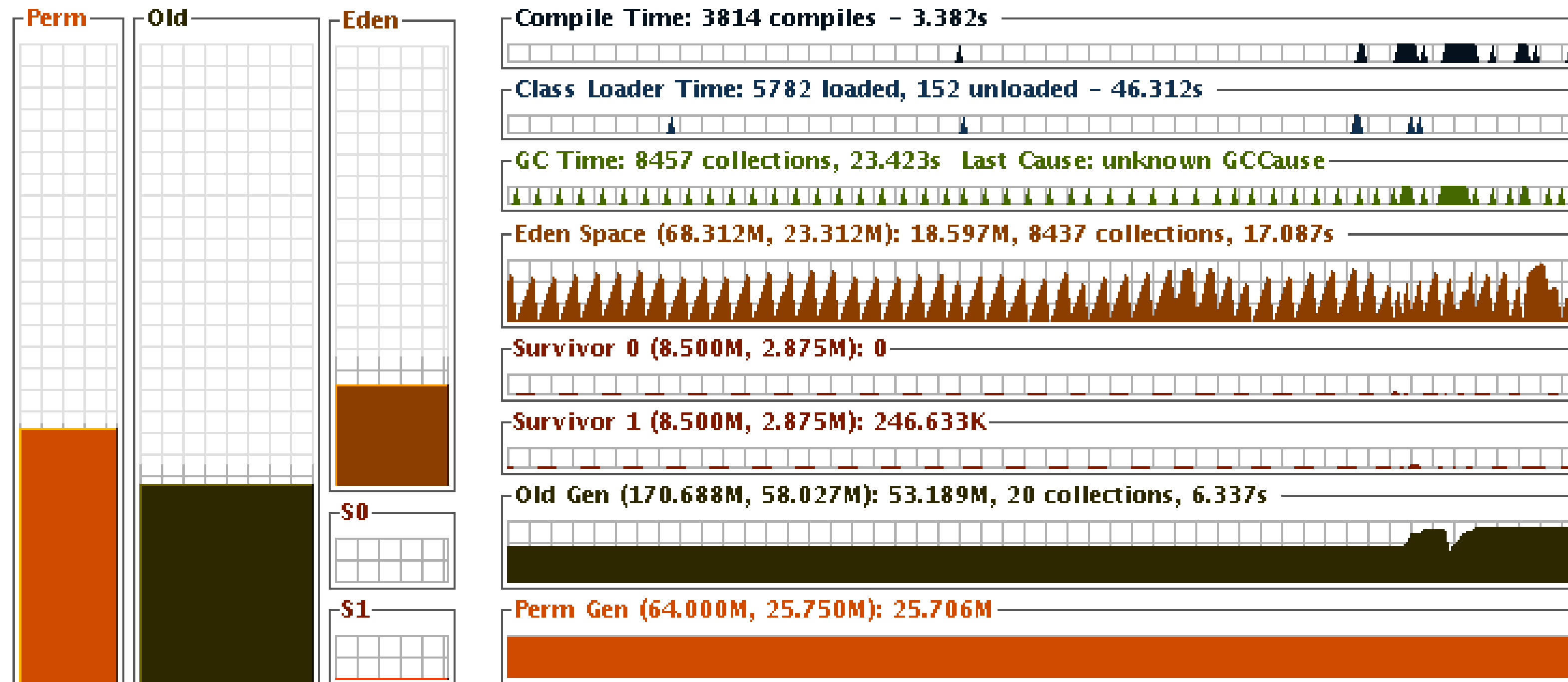
- Кратко о сборке мусора
- **Мониторинг GC**
- **Настройка GC**
 - Объем используемой памяти
 - Молодое поколение
 - Паузы
 - CMS
- G1
- Выводы



Диагностический вывод GC

- Параметры VM
 - XX:+PrintGCDetails -XX:+PrintGCTimeStamps -Xloggc:<file>
 - XX:+PrintGCDateStamps
 - XX:+PrintHeapAtGC
- Минимальные накладные расходы
- Анализ диагностического вывода GC
 - **PrintGCStats**
 - **GChisto**: <http://gchisto.dev.java.net/>

VisualVM / VisualGC



Histogram x

Parameters

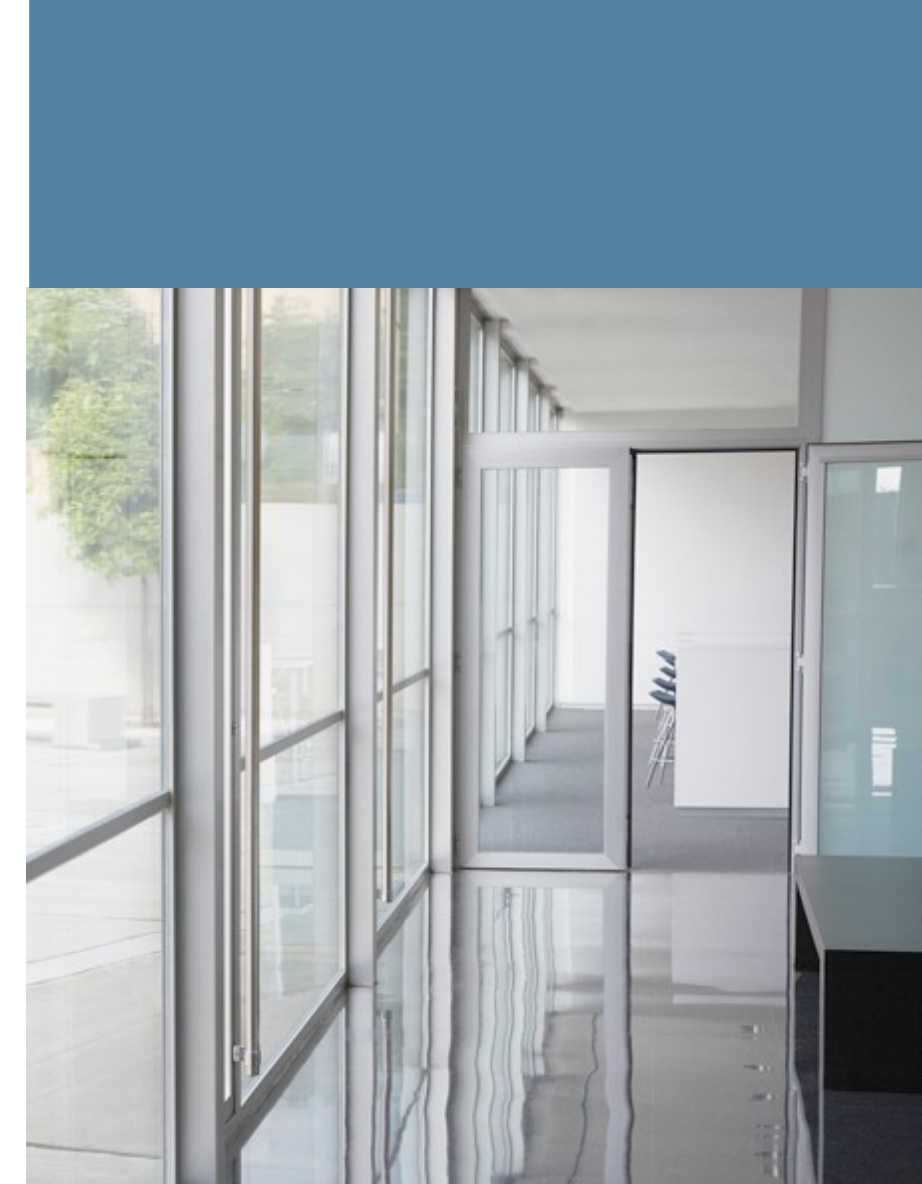
Tenuring Threshold: 15 Max Tenuring Threshold: 15 Desired Survivor Size: 1507328 Current Survivor Size: 3014656

Histogram

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Содержание

- Кратко о сборке мусора
- Мониторинг GC
- **Настройка GC**
 - Объем используемой памяти
 - Молодое поколение
 - Паузы
 - CMS
- G1
- Выводы



Идеальный сборщик мусора

- Не прерывает работу приложения
- Использует ровно столько оперативной памяти, сколько нужно приложению
- Не требует дополнительных вычислительных ресурсов

Производительность GC

- 3 характеристики

Throughput

Объем вычислительных ресурсов, затрачиваемых на GC

Предсказуемость

На какое время прерывается работа приложения

Footprint

Объем используемой памяти

Принципы настройки GC

- Чем больше доступно памяти GC, тем лучше
- Оптимизировать по 2 параметрам из 3
 - Throughput
 - Длительность пауз
 - Объем используемой памяти

Содержание

- Кратко о сборке мусора
- Мониторинг GC
- Настройка GC
 - **Объем используемой памяти**
 - Молодое поколение
 - Паузы
 - CMS
- G1
- Выводы



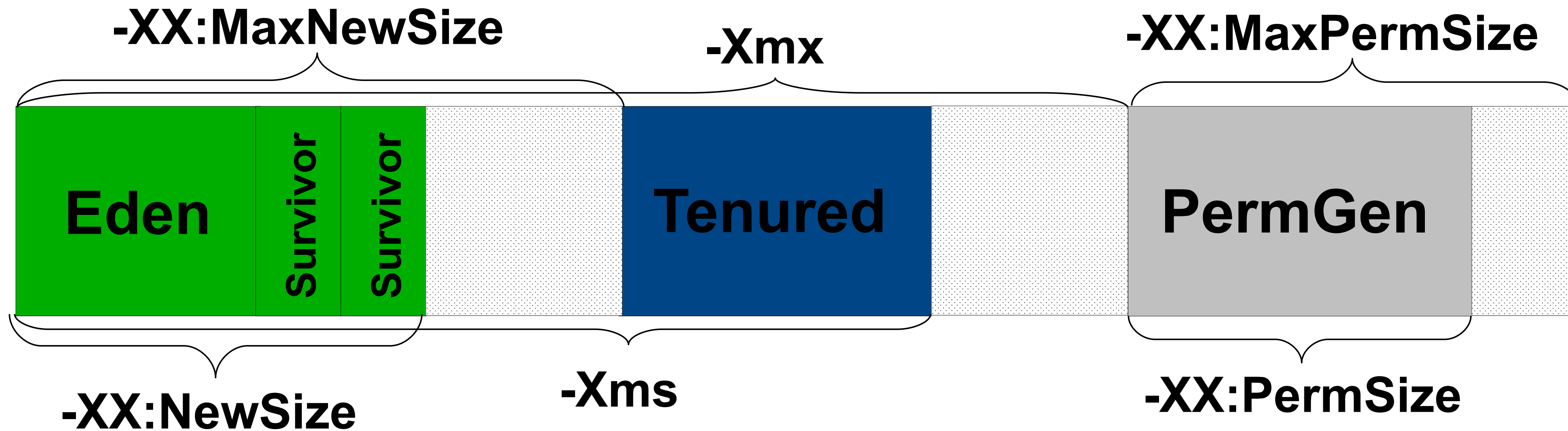
Критерии выбора размера хипа

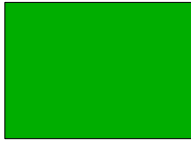


- Чем больше памяти, тем лучше для GC
 - Как для молодого, так и для старшего поколения
 - Более редкие сборки
 - Ниже затраты на сборку мусора
- Доступный объем памяти ограничен
 - Физическая память
 - 32-битный режим
 - Наличие других приложений в системе

Размеры поколений

- Размер **молодого** поколения
 - Определяет частоту малых сборок (**Minor GC**)
 - Влияет на то, сколько объектов умирает, не покидая молодое поколение
- Размер **старшего** поколения
 - Определяет частоту полных сборок (**Full GC**)
 - Не меньше суммарного размера долгоживущих объектов, используемых приложением в стабильном состоянии

Управление размерами поколений



-  Молодое поколение (Young Gen)
-  Старшее поколение (Old Gen)
-  Permanent Generation (Perm Gen)

Размер хипа: основные цели

- Объем используемой памяти не должен превышать объем физической памяти
- Максимизировать объем памяти, освобождаемый при малых и полных GC
- Относится ко **всем** GC

Размер хипа: с чего лучше начать?

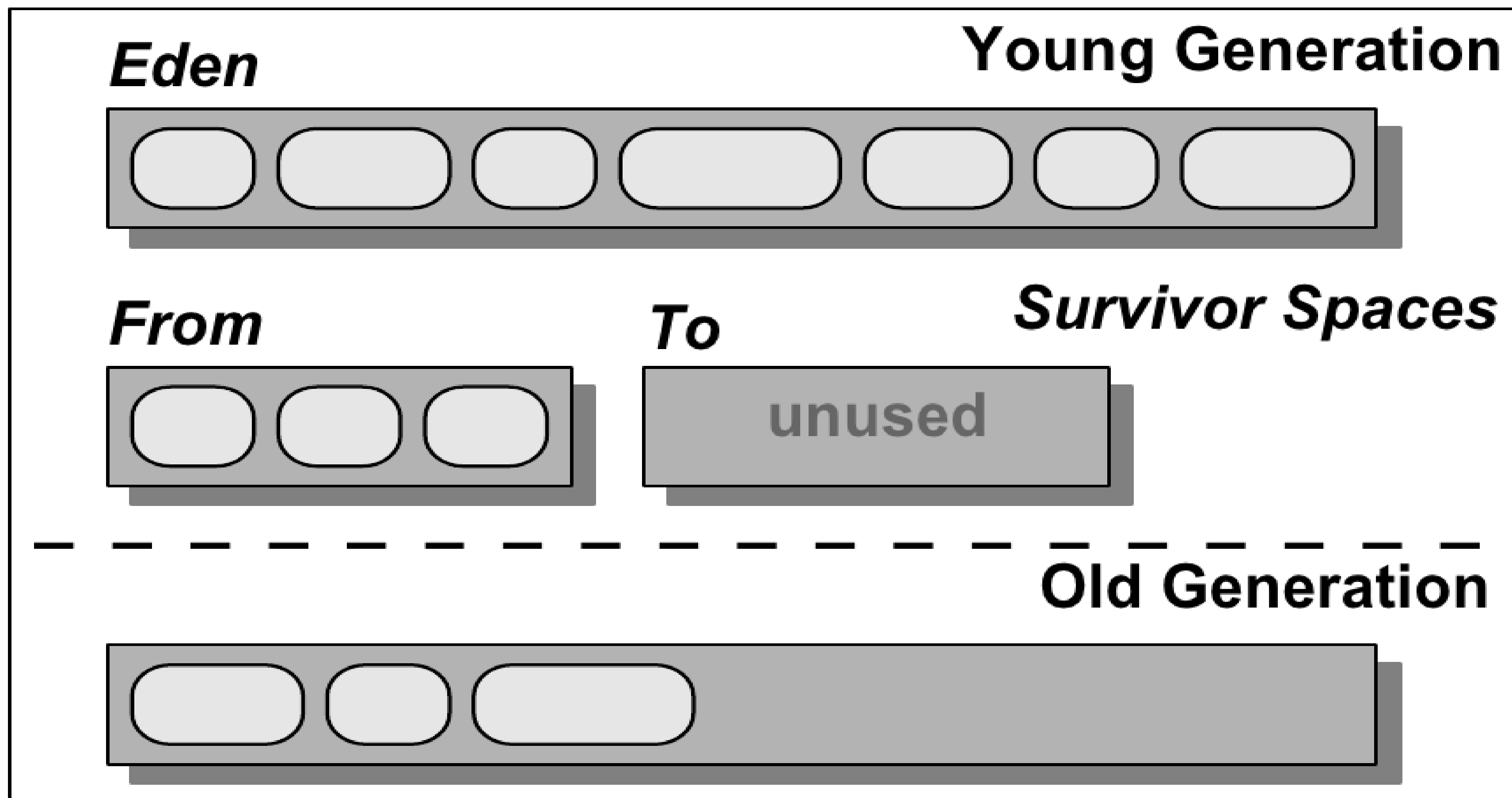
- **LDS = Live Data Size**
 - размер множества достижимых объектов
 - для приложения в стабильном состоянии
- **-Xms / -Xmx = 3-4x LDS**
- **-XX:PermSize = 1.2-1.5x** от макс. размера PermGen
- **Размеры поколений**
 - Молодое поколение: **-Xmn = 1-1.5x LDS**
 - Старшее поколение: **-Xmx = 2-3x LDS + -Xmn**
- **Пример:**
 - Если **LDS == 512m**, тогда **-Xmn768m -Xms2g -Xmx2g**

Содержание

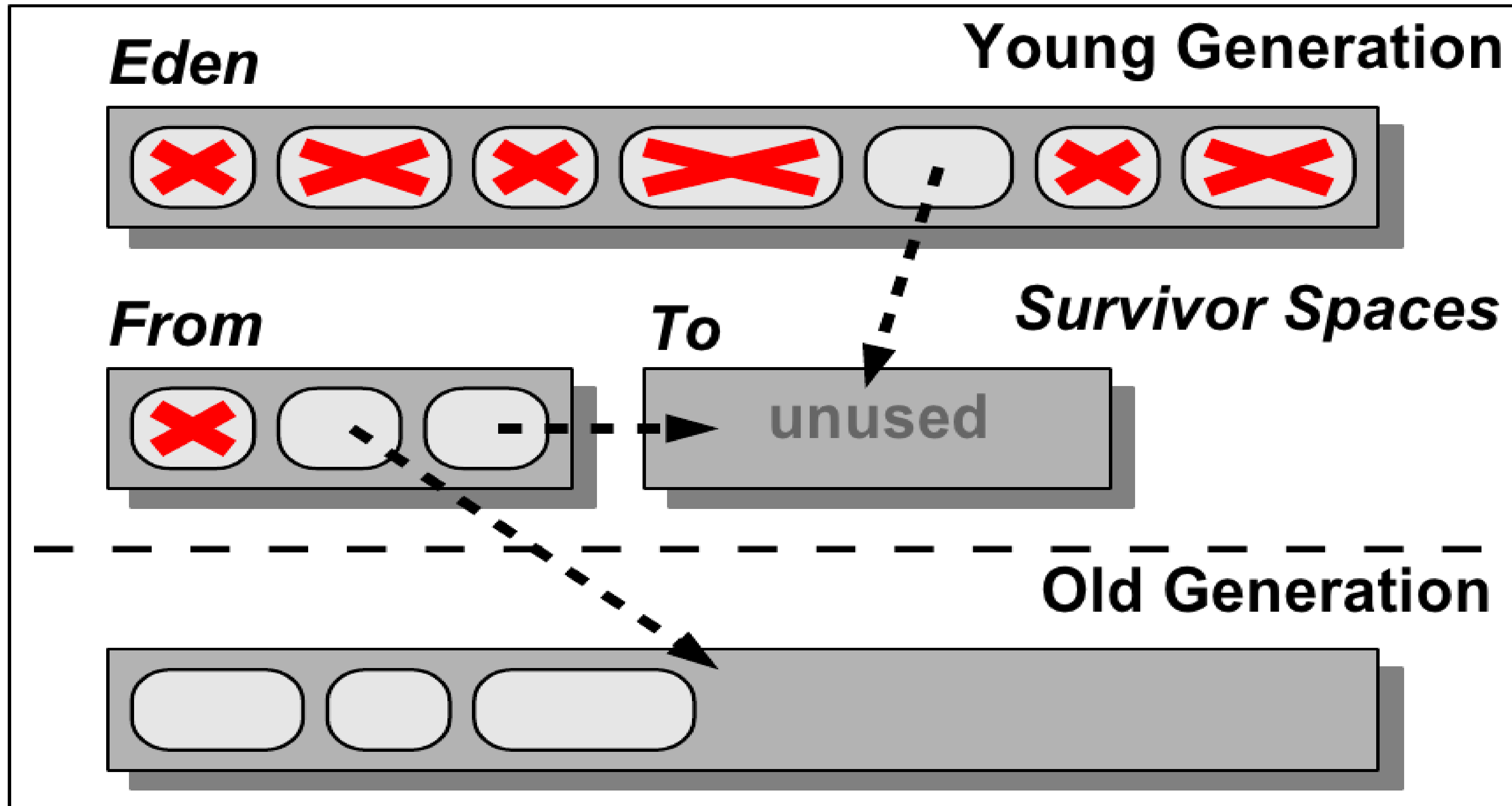
- Кратко о сборке мусора
- Мониторинг GC
- Настройка GC
 - Объем используемой памяти
 - **Молодое поколение**
 - Паузы
 - CMS
- G1
- Выводы



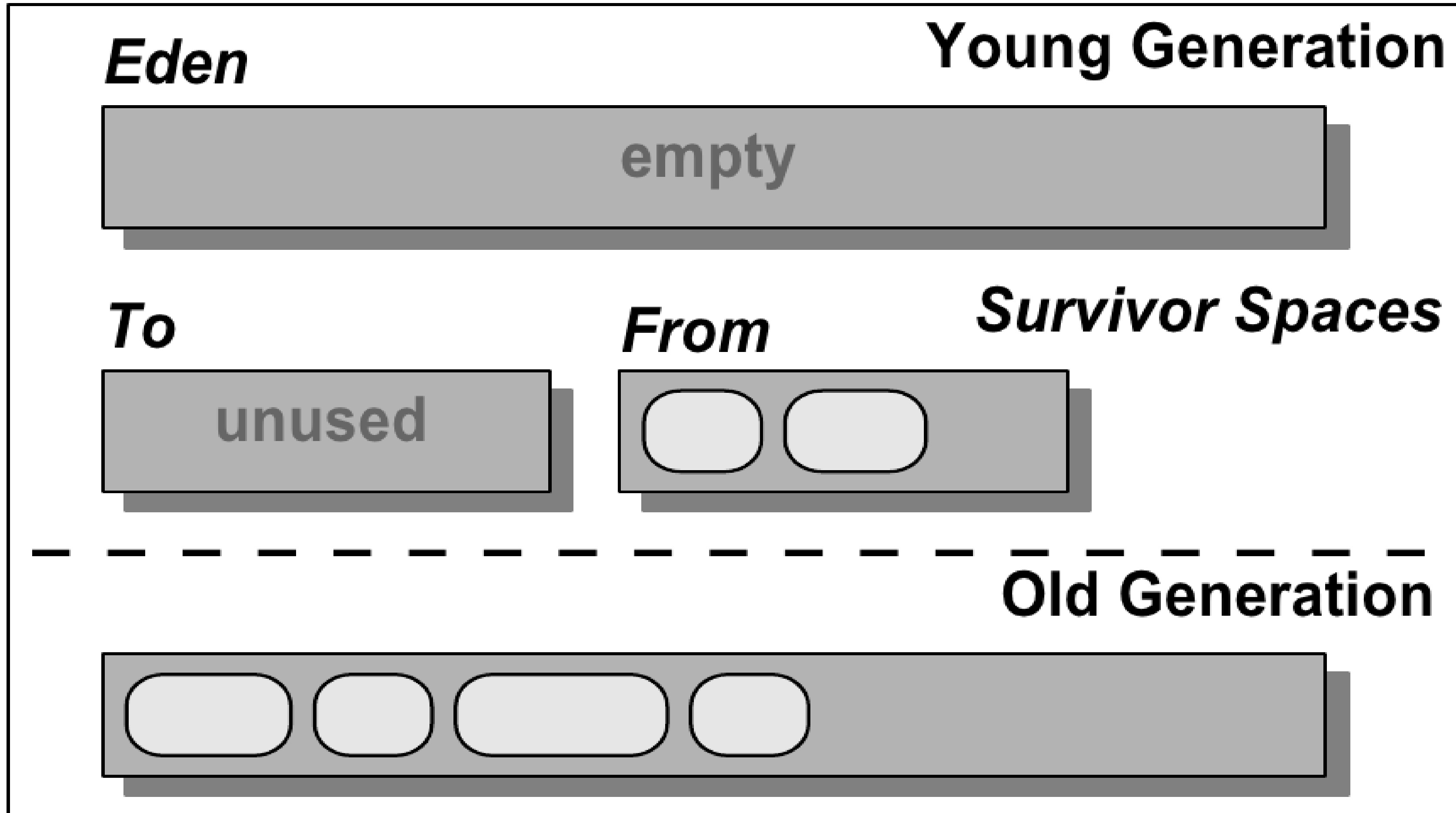
Структура молодого поколения



Молодое поколение: сборка мусора (I)

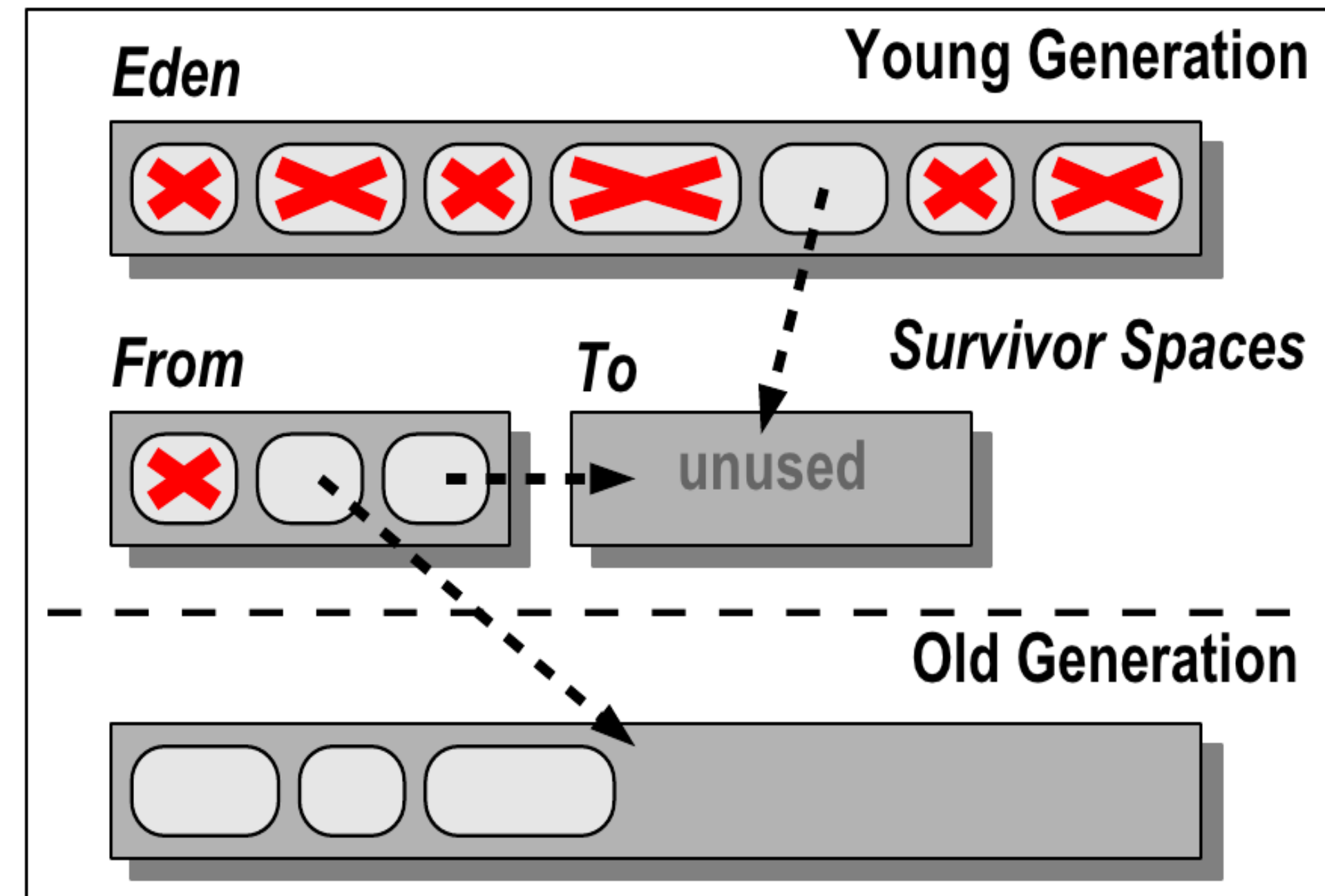


Молодое поколение: сборка мусора (II)

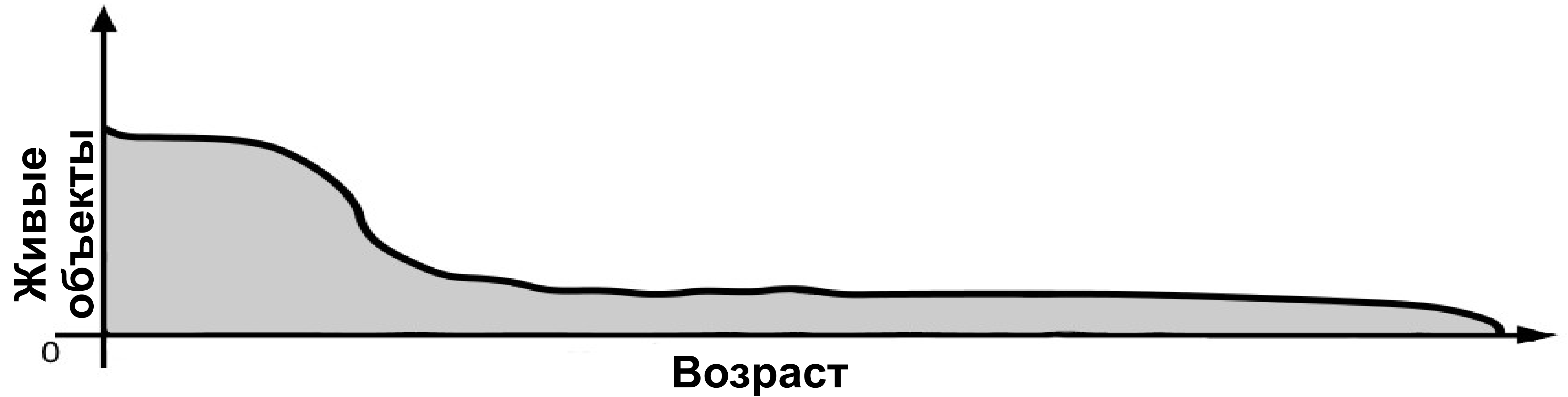


Перемещение в старшее поколение

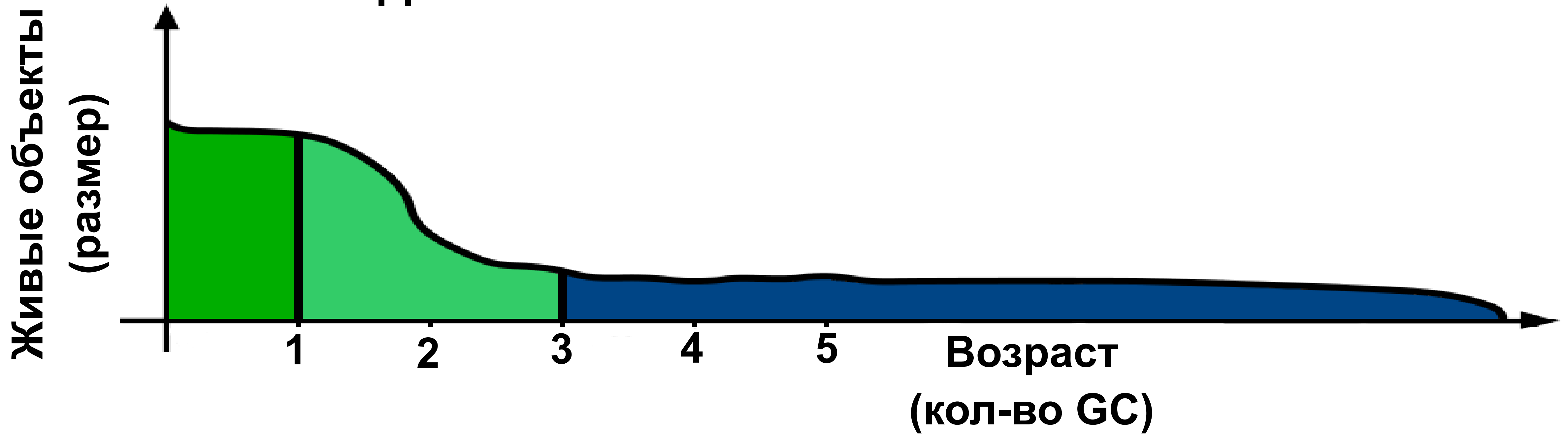
- Временные объекты должны умирать в молодом поколении
- Долгоживущие объекты должны попадать в старшее поколение как можно раньше


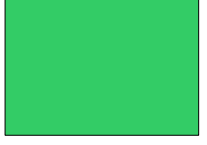



Время жизни объектов



Время жизни объектов: Как должно быть



-  Регион Eden (Young Gen)
-  Регион Survivor (Young Gen)
-  Старшее поколение (Old / Tenured Gen)

Время жизни объектов

- `-XX:+PrintTenuringDistribution`

Desired survivor size 3342336 bytes, new threshold 6 (max 6)

- age 1: 2483440 bytes, 2483440 total
- age 2: 501240 bytes, 2984680 total
- age 3: 50016 bytes, 3034696 total
- age 4: 49088 bytes, 3083784 total
- age 5: 48616 bytes, 3132400 total
- age 6: 50128 bytes, 3182528 total

Время жизни объектов

- `-XX:+PrintTenuringDistribution`

Desired survivor size 3342336 bytes, new threshold 6 (max 6)

- age 1: 2483440 bytes, 2483440 total
- age 2: 501240 bytes, 2984680 total
- age 3: 50016 bytes, 3034696 total
- age 4: 49088 bytes, 3083784 total
- age 5: 48616 bytes, 3132400 total
- age 6: 50128 bytes, 3182528 total

- Возраст (age) – количество пережитых малых GC

Время жизни объектов

- `-XX:+PrintTenuringDistribution`

Desired survivor size 3342336 bytes, new threshold 6 (max 6)

- age 1: 2483440 bytes, 2483440 total
- age 2: 501240 bytes, 2984680 total
- age 3: 50016 bytes, 3034696 total
- age 4: 49088 bytes, 3083784 total
- age 5: 48616 bytes, 3132400 total
- age 6: 50128 bytes, 3182528 total

- Суммарный размер объектов определенного возраста

Время жизни объектов

- `-XX:+PrintTenuringDistribution`

Desired survivor size 3342336 bytes, new threshold 6 (max 6)

- age 1: 2483440 bytes, 2483440 total
- age 2: 501240 bytes, 2984680 total
- age 3: 50016 bytes, 3034696 total
- age 4: 49088 bytes, 3083784 total
- age 5: 48616 bytes, 3132400 total
- age 6: 50128 bytes, 3182528 total

- Суммарный размер объектов, не старше определенного возраста

Время жизни объектов

- `-XX:+PrintTenuringDistribution`

Desired survivor size 3342336 bytes, new threshold 6 (max 6)

- age 1: 2483440 bytes, 2483440 total
- age 2: 501240 bytes, 2984680 total
- age 3: 50016 bytes, 3034696 total
- age 4: 49088 bytes, 3083784 total
- age 5: 48616 bytes, 3132400 total
- age 6: 50128 bytes, **3182528** total

- Общий размер объектов в Survivor пространстве

Время жизни объектов

- `-XX:+PrintTenuringDistribution`

Desired survivor size 3342336 bytes, new threshold 6 (max 6)

- age 1: 2483440 bytes, 2483440 total
- age 2: 501240 bytes, 2984680 total
- age 3: 50016 bytes, 3034696 total
- age 4: 49088 bytes, 3083784 total
- age 5: 48616 bytes, 3132400 total
- age 6: 50128 bytes, 3182528 total

- Эффективный размер Survivor пространства
== (Размер Survivor) x TargetSurvivorRatio

- `-XX:TargetSurvivorRatio=<percent> ([1..100])`

Время жизни объектов

- `-XX:+PrintTenuringDistribution`

Desired survivor size 3342336 bytes, new threshold 6 (max 6)

- age 1: 2483440 bytes, 2483440 total
- age 2: 501240 bytes, 2984680 total
- age 3: 50016 bytes, 3034696 total
- age 4: 49088 bytes, 3083784 total
- age 5: 48616 bytes, 3132400 total
- age 6: 50128 bytes, 3182528 total

- Текущий пороговый возраст копирования в старшее поколение

Время жизни объектов

- **-XX:+PrintTenuringDistribution**

Desired survivor size 3342336 bytes, new threshold 6 (max 6)

- age 1: 2483440 bytes, 2483440 total
- age 2: 501240 bytes, 2984680 total
- age 3: 50016 bytes, 3034696 total
- age 4: 49088 bytes, 3083784 total
- age 5: 48616 bytes, 3132400 total
- age 6: 50128 bytes, 3182528 total

- **Максимальный пороговый возраст копирования**
-XX:MaxTenuringThreshold (MTT)

Время жизни объектов: Как должно быть

Desired survivor size 6684672 bytes, new threshold 8 (max 8)

- age 1: 2315488 bytes, 2315488 total
- age 2: 19528 bytes, 2335016 total
- age 3: 96 bytes, 2335112 total
- age 4: 32 bytes, 2335144 total

Время жизни объектов: Как бывает (I)

Desired survivor size 3342336 bytes, new threshold 1 (max 6)
- age 1: 3956928 bytes, 3956928 total

Время жизни объектов: Как бывает (I)

Desired survivor size 3342336 bytes, new threshold 1 (max 6)
- age 1: 3956928 bytes, 3956928 total

Вывод: размер Survivor **слишком мал**.

Решение: увеличить размер Survivor и/или Eden.

-XX:MaxNewSize / -Xmn

-XX:SurvivorRatio=<ratio>

соотношение между Eden и Survivor

Время жизни объектов: Как бывает (II)

Desired survivor size 3342336 bytes, new threshold 6 (max 6)

- age 1: 2483440 bytes, 2483440 total
- age 2: 501240 bytes, 2984680 total
- age 3: 50016 bytes, 3034696 total
- age 4: 49088 bytes, 3083784 total
- age 5: 48616 bytes, 3132400 total
- age 6: 50128 bytes, 3182528 total

Время жизни объектов: Как бывает (II)

Desired survivor size 3342336 bytes, new threshold 6 (max 6)

- age 1: 2483440 bytes, 2483440 total
- age 2: 501240 bytes, 2984680 total
- age 3: 50016 bytes, 3034696 total
- age 4: 49088 bytes, 3083784 total
- age 5: 48616 bytes, 3132400 total
- age 6: 50128 bytes, 3182528 total

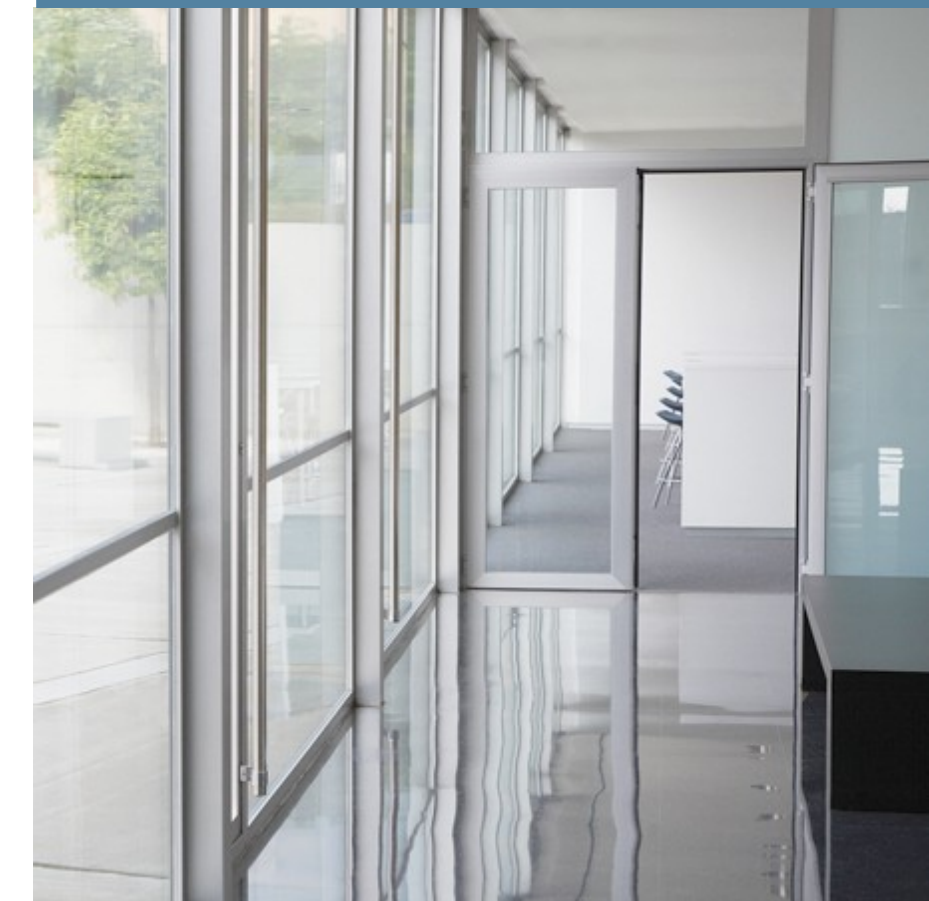
Вывод: настройки не оптимальны.

Решение: 2 варианта:

- Увеличить МТТ: 6 → [7...]
- Установить МТТ = 2

Содержание

- Кратко о сборке мусора
- Мониторинг GC
- Настройка GC
 - Объем используемой памяти
 - Молодое поколение
 - Паузы
 - CMS
- G1
- Выводы



Паузы: Молодое поколение

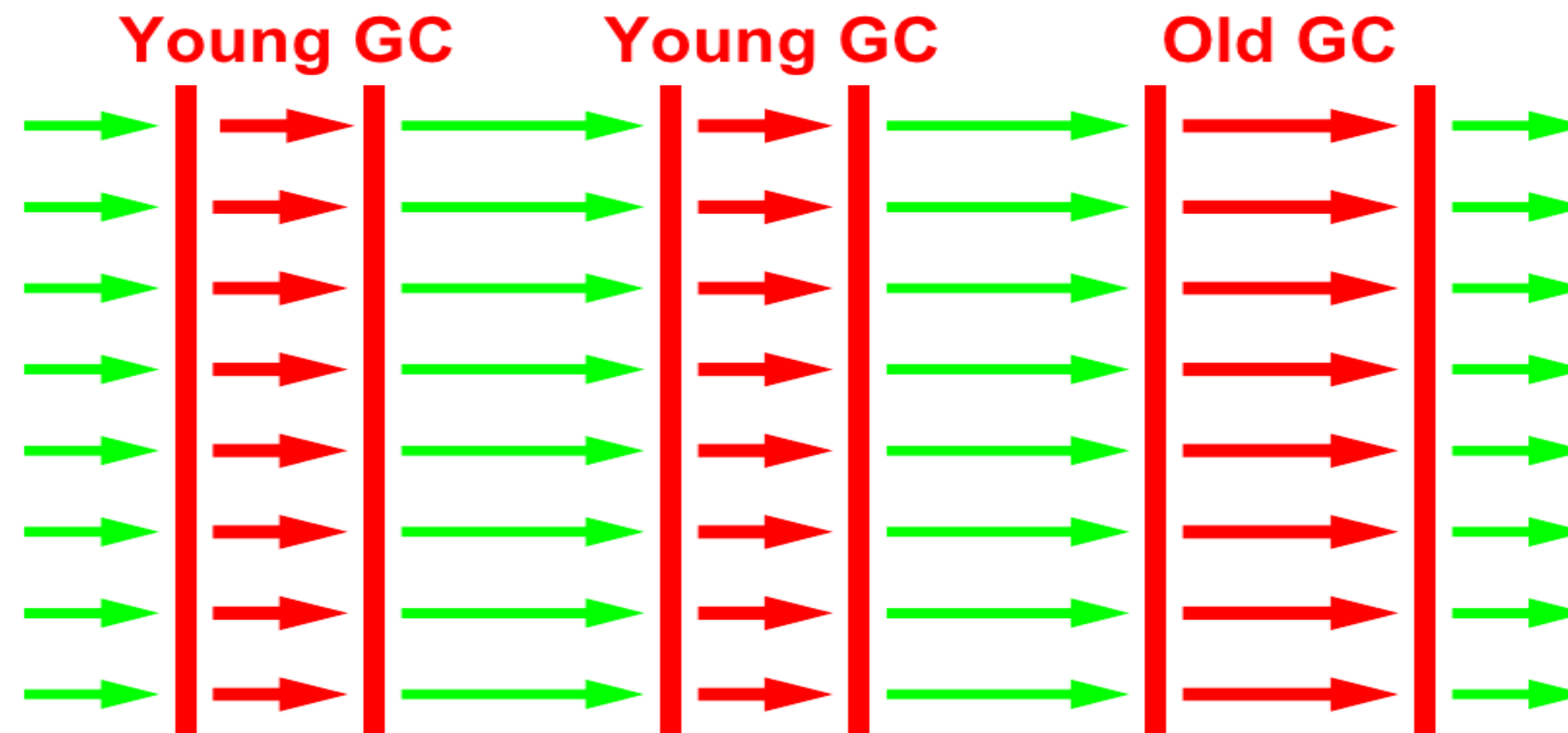
- Продолжительность малых сборок
 - Часто является причиной задержек, вызванных GC
 - Важна как продолжительность, так и частота
- Малые сборки
 - продолжительны → ↓ размер молодого поколения
 - часты → ↑ размер молодого поколения

Паузы: Старшее поколение (I)

- ParallelOldGC vs CMS
 - ParallelOldGC может удовлетворять достаточно жестким ограничениям на длительность пауз
- **NB!**

прежде чем использовать CMS убедитесь, что ParallelOldGC не подходит

Паузы: Старшее поколение (II)



- `-XX:+UseParallelOldGC`
- `-XX:ParallelGCThreads=<num>`

Паузы: Старшее поколение (III)

- Частые полные сборки → ↑ размер старшего поколения
 - постарайтесь зафиксировать размер молодого поколения
- Продолжительные полные сборки → ParallelOldGC не подходит
 - изменение размера старшего поколения не поможет
 - Совет:
 - использовать CMS
 - исключить полные stop-the-world (STW) сборки (FullGC)

Паузы: Полные сборки

Как бывает

- Происходят **только** полные сборки
- Причина:
 - **дисбаланс** размеров молодого и старшего поколений
 - после полной сборки в старшем поколении нет свободного места
- Решение:
 - ↑ размер старшего поколения
 - оптимально ли настроено молодое поколение?

Паузы: выбор GC для старшего поколения (I)

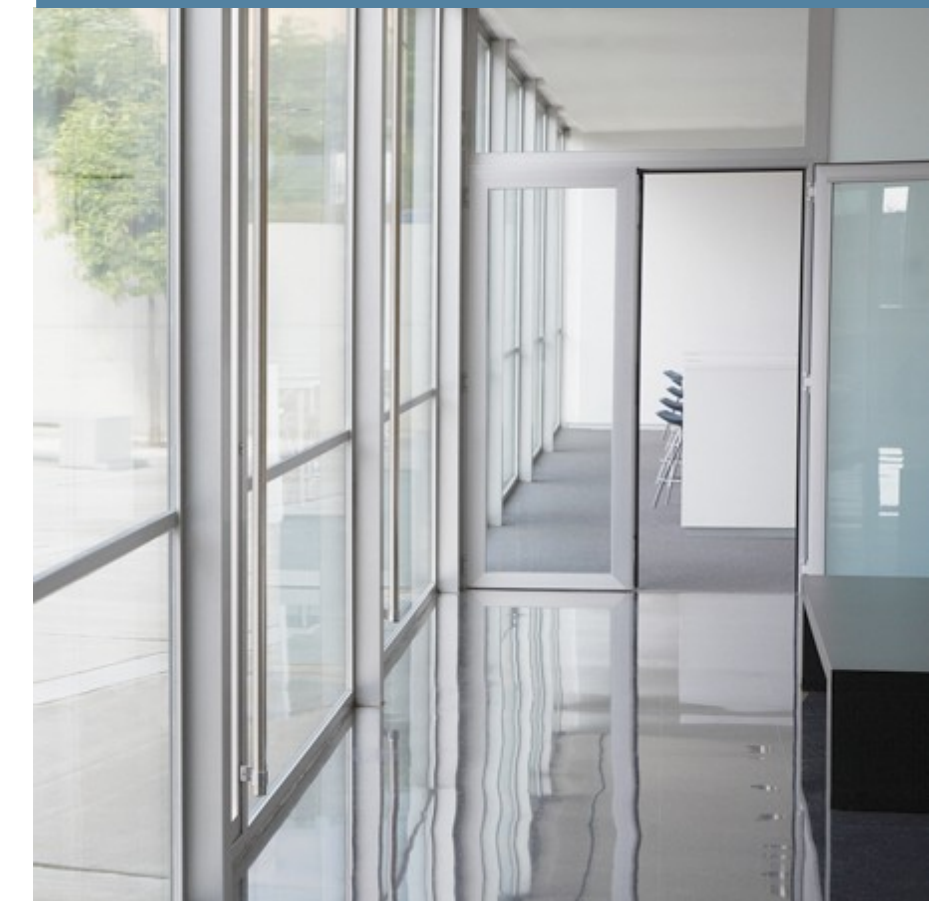
- Продолжительность и частота малых и полных сборок устраивает
→ ParallelOldGC
- Продолжительность малых сборок устраивает
Полные сборки слишком продолжительны/часты
→ CMS
- Продолжительность малых сборок недопустима
→ требуется модификация приложения

Паузы: выбор GC для старшего поколения (II)

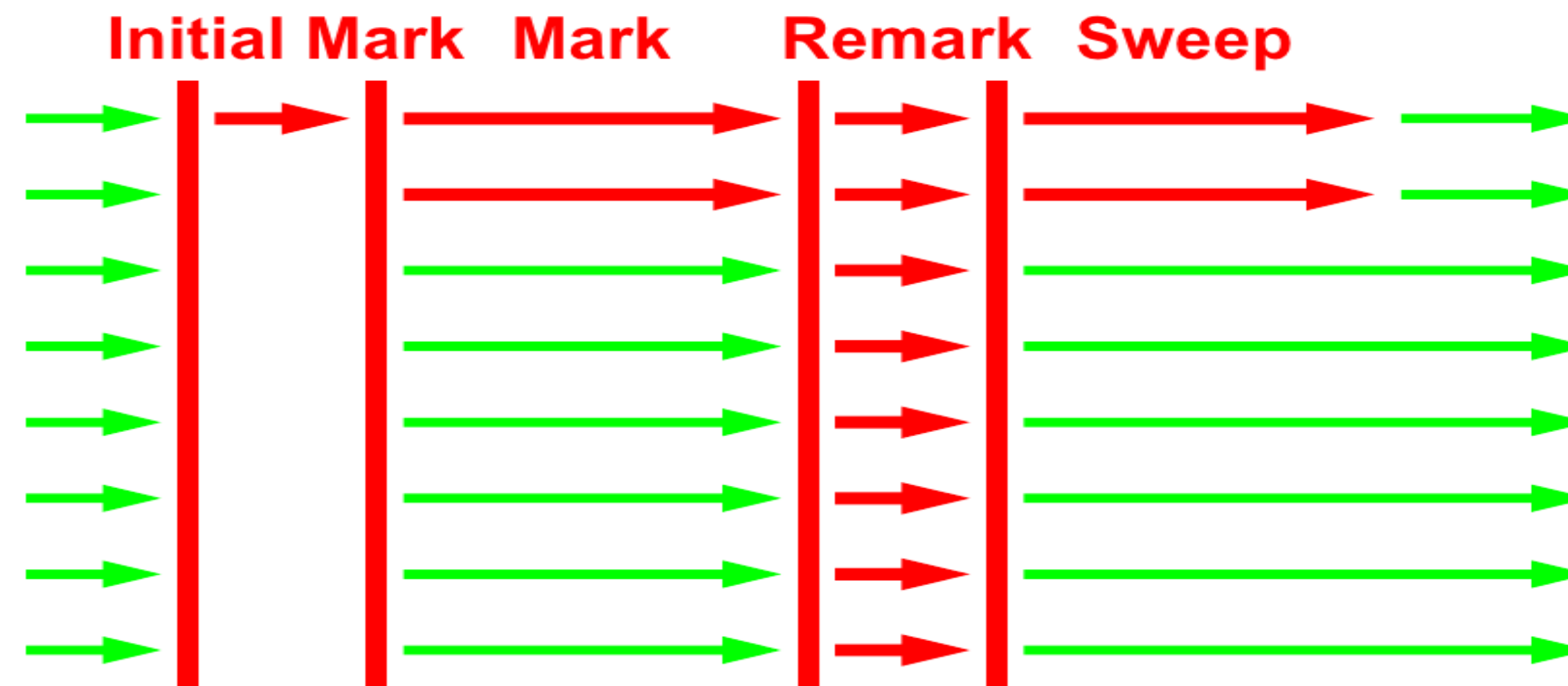
- Продолжительность малых сборок может конфликтовать с требованиями приложения
- Что стоит попробовать:
 - уменьшить количество объектов со средней продолжительностью жизни
 - Если возможно, запускать приложение в нескольких JVM с меньшим размером хипа

Содержание

- Кратко о сборке мусора
- Мониторинг GC
- Настройка GC
 - Объем используемой памяти
 - Молодое поколение
 - Паузы
 - **CMS**
- G1
- Выводы



CMS: Concurrent Mark Sweep

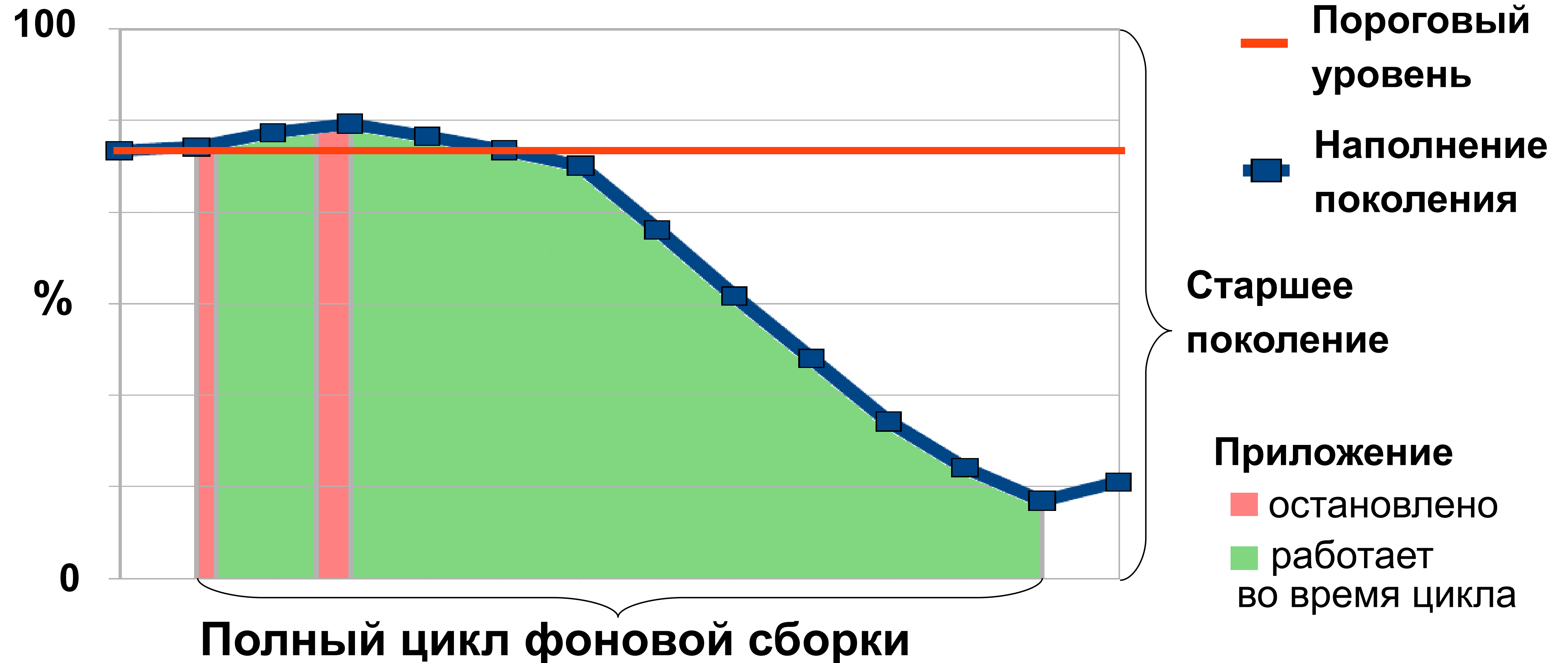


- 3 режима GC
 - GC в молодом поколении (**Minor GC**)
 - Полная сборка (**Major GC**)
 - **Фоновая сборка** (2 коротких STW паузы)
 - С остановкой приложения (**Full GC**)

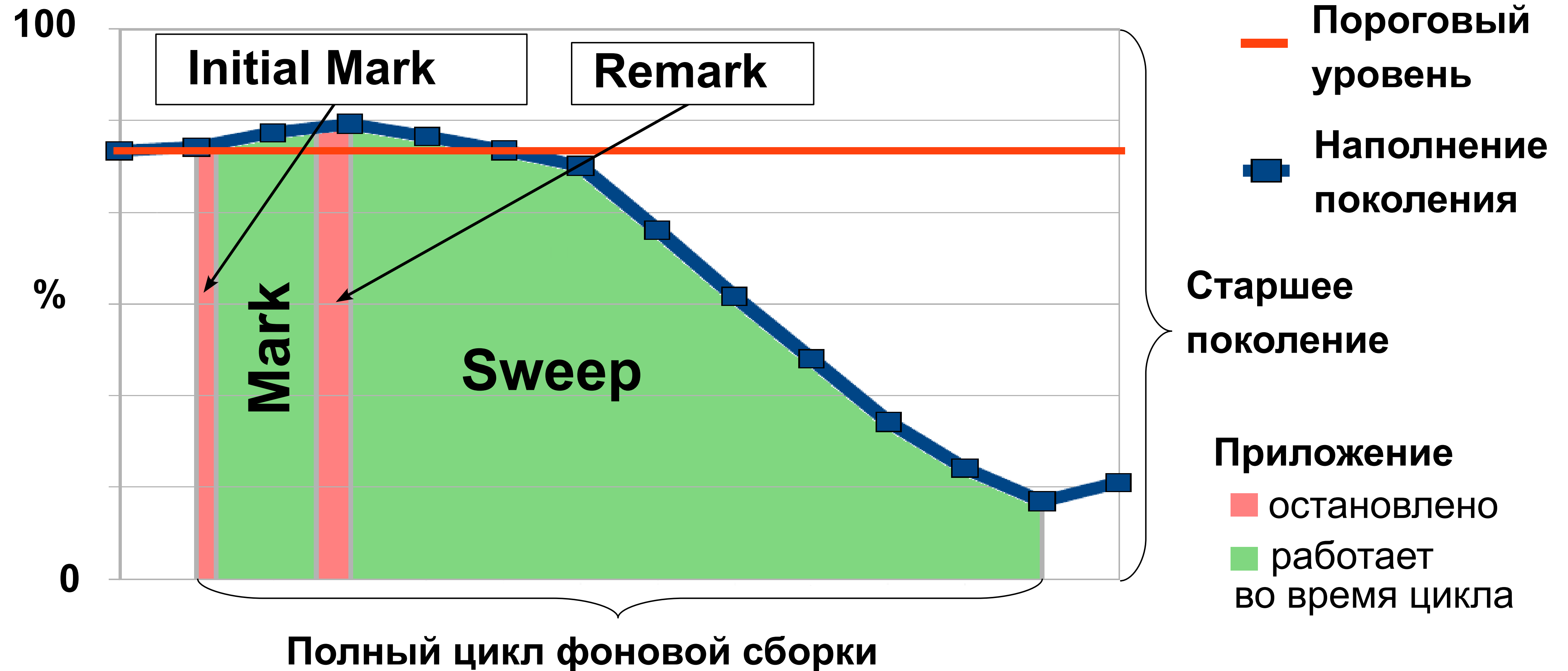
CMS: как стартовать фоновую сборку

- по пороговому наполнению поколения
 - XX:CMSInitiatingOccupancyFraction=<percent>
 - XX:CMSInitiatingPermOccupancyFraction=<percent>
- System.gc()
 - XX:+ExplicitGCInvokesConcurrent
 - XX:+ExplicitGCInvokesConcurrentAndUnloadClasses

CMS: Фоновая сборка Как должно быть



CMS: Фоновая сборка Как должно быть



CMS: Сценарии работы

Как должно быть

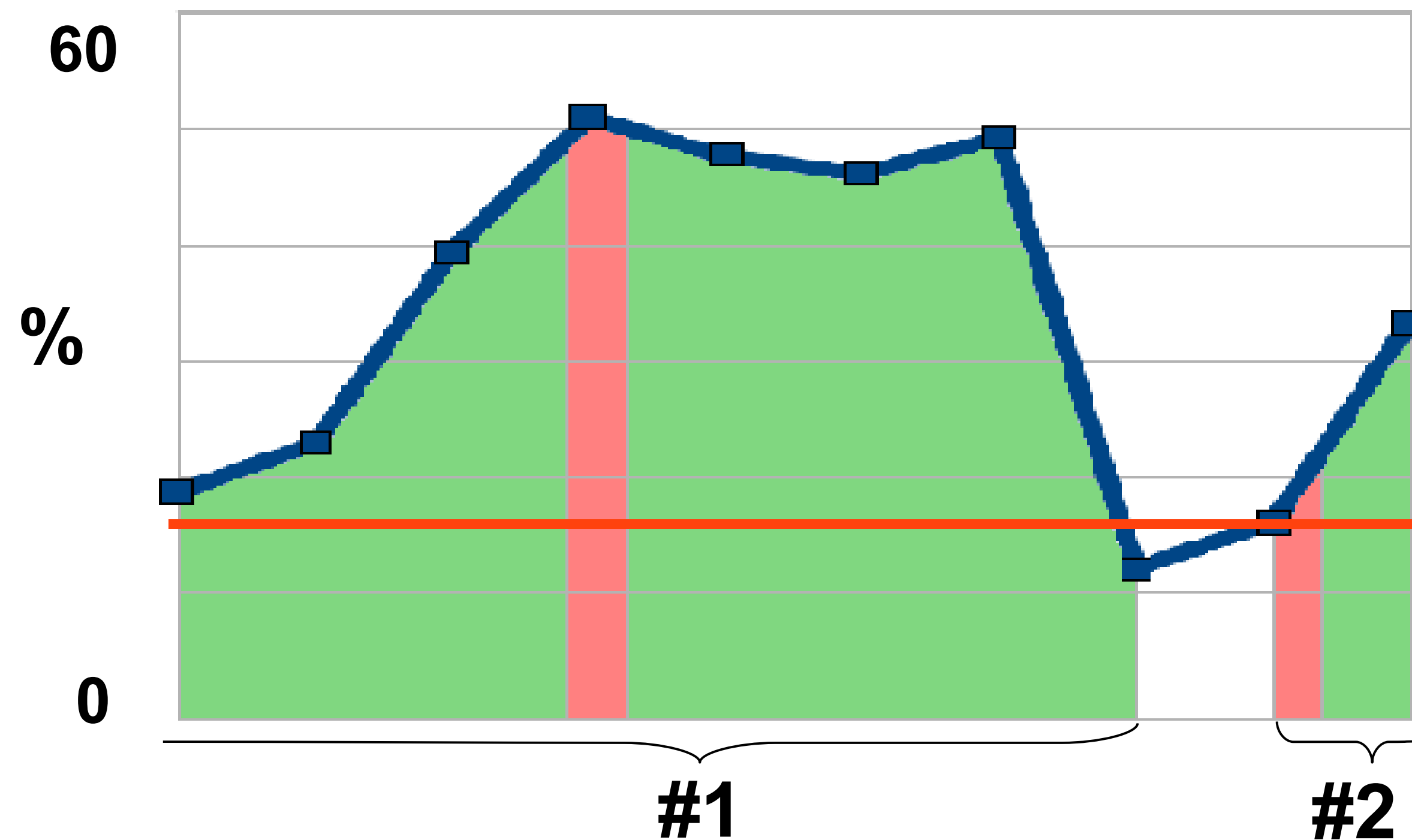
- Скорость наполнения старшего поколения **сильно варьируется**
 - Старшее поколение может заполняться очень быстро
 - Частые GC
 - Цикл фоновой сборки должен стартовать заблаговременно
- В старшее поколение попадает **мало объектов**
 - Старшее поколение заполняется медленно и монотонно
 - GC редки
 - Цикл фоновой сборки безопасно начинать достаточно поздно

CMS: Сценарии работы

Как бывает (I)

- Цикл начинается слишком рано

- Частые циклы фоновой сборки
- Избыточные накладные расходы

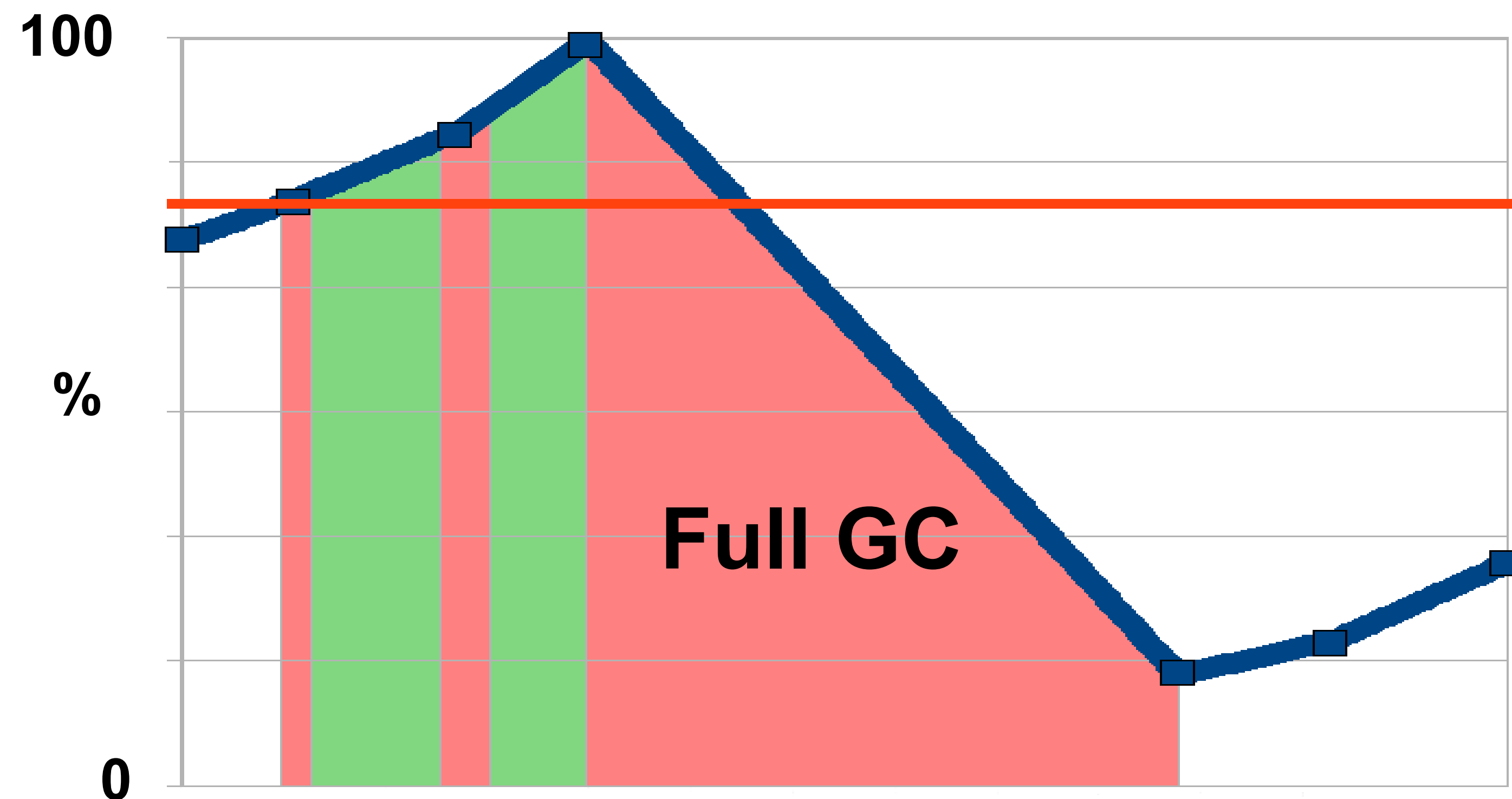


- Пороговый уровень
 - Наполнение поколения
- Приложение
- остановлено
 - работает во время цикла

CMS: Сценарии работы

Как бывает (II)

- Цикл начинается слишком **поздно**
 - Высока вероятность полной сборки (**Full GC**)



— Пороговый уровень

■ Наполнение поколения

Приложение

- остановлено
- работает во время цикла

CMS: Настройка

- Минимизируйте перемещение объектов в старшее поколение
 - Аллокация памяти в старшем поколении дорога (free lists)
 - Фрагментация
- `-XX:ParallelCMSThreads=<num>`
 - Количество потоков, участвующих в фоновом GC
 - Компромисс между:
 - длительностью цикла фоновой сборки
 - overhead фоновой сборки

CMS: Фрагментация старшего поколения

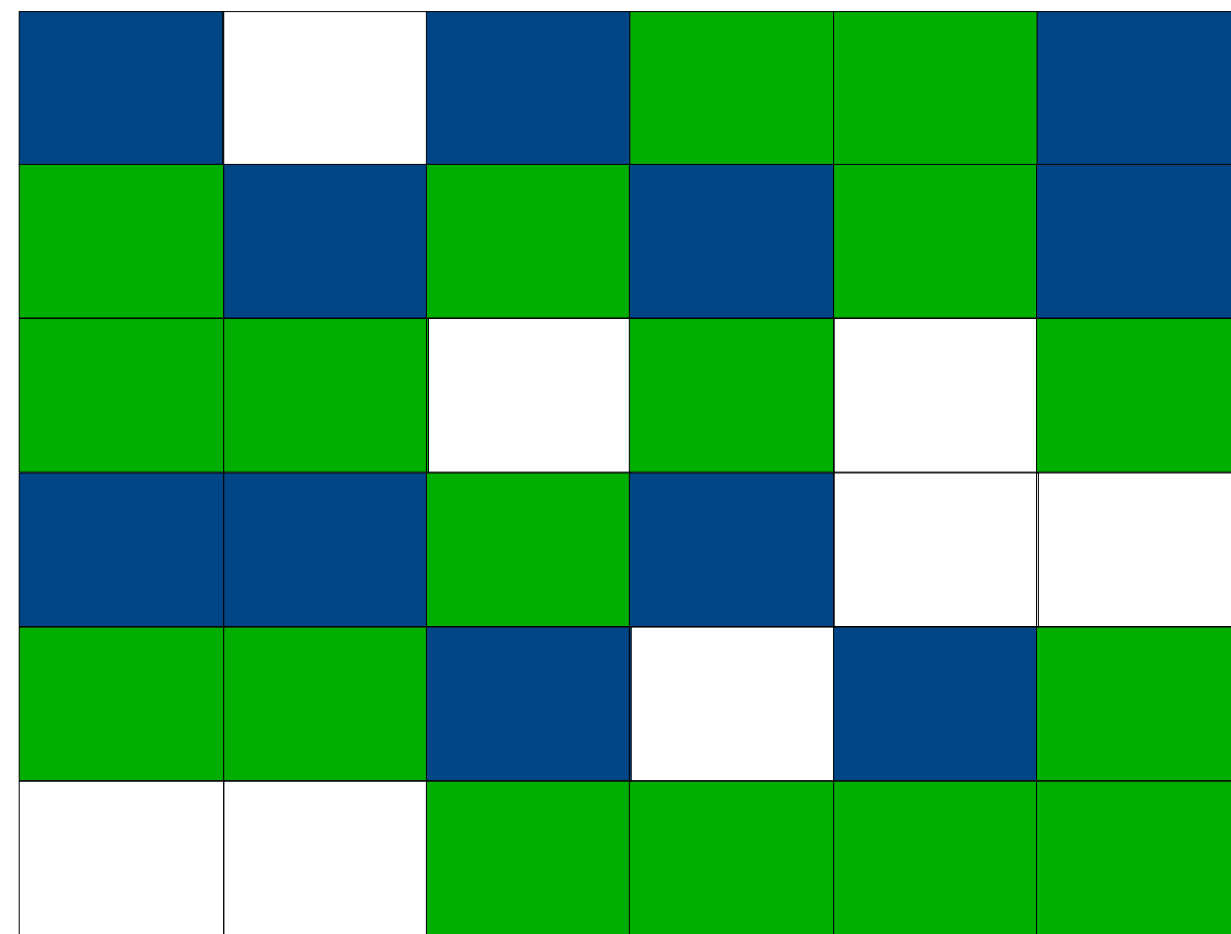
- Неизбежна
- Компактификация только при **Full GC**
- Можно минимизировать
 - Как можно меньше копировать объектов в старшее поколение
 - Модифицировать приложение
 - Избегайте создания больших объектов разных размеров

Содержание

- Кратко о сборке мусора
- Мониторинг GC
- Настройка GC
 - Объем используемой памяти
 - Молодое поколение
 - Паузы
 - CMS
- **G1**
- Выводы



G1: Описание



 Регион из молодого поколения

 Регион из старшего поколения

 Пустой регион

- Хип разбит на регионы
- Сборка мусора – **эвакуация** регионов
- Основан на поколениях
 - Сборка молодого поколения
Эвакуация **всех** молодых регионов
 - Сборка старшего поколения
Эвакуация регионов с **наибольшим количеством мусора**

G1: Использование

- `-XX:+UseG1GC`
- Задаваемые цели на длительность и частоту пауз
 - `-XX:MaxGCPauseMillis=<num>`
 - `-XX:GCPauseIntervalMillis=<num>`

Содержание

- Кратко о сборке мусора
- Мониторинг GC
- Настройка GC
 - Объем используемой памяти
 - Молодое поколение
 - Паузы
 - CMS
- G1
- **Выводы**



Подводим итоги

- Объем используемой памяти
 - Конфигурация хипа
- Длительность пауз
 - Конфигурация молодого поколения
 - Контроль времени жизни объектов
 - Выбор сборщика для старшего поколения
- ParallelOldGC
- CMS
 - Настройка фонового режим
- G1

Необычные конфигурации

- Размер молодого поколения = **80%** хипа
- Размер старшего поколение = **1.2x** от размера долгоживущих живых объектов
- Начало фоновой сборки при **95%** занятости старшего поколения

“If you aren't assessing, you are guessing.”

Measure, don't guess!

Q & A



