LUXOFT
ENGINEERING BUSINESS PERFORMANCE

Исследование возможностей и ограничений JVM по оптимизации памяти: история одного проекта

Александр Макаров

Luxoft

План доклада

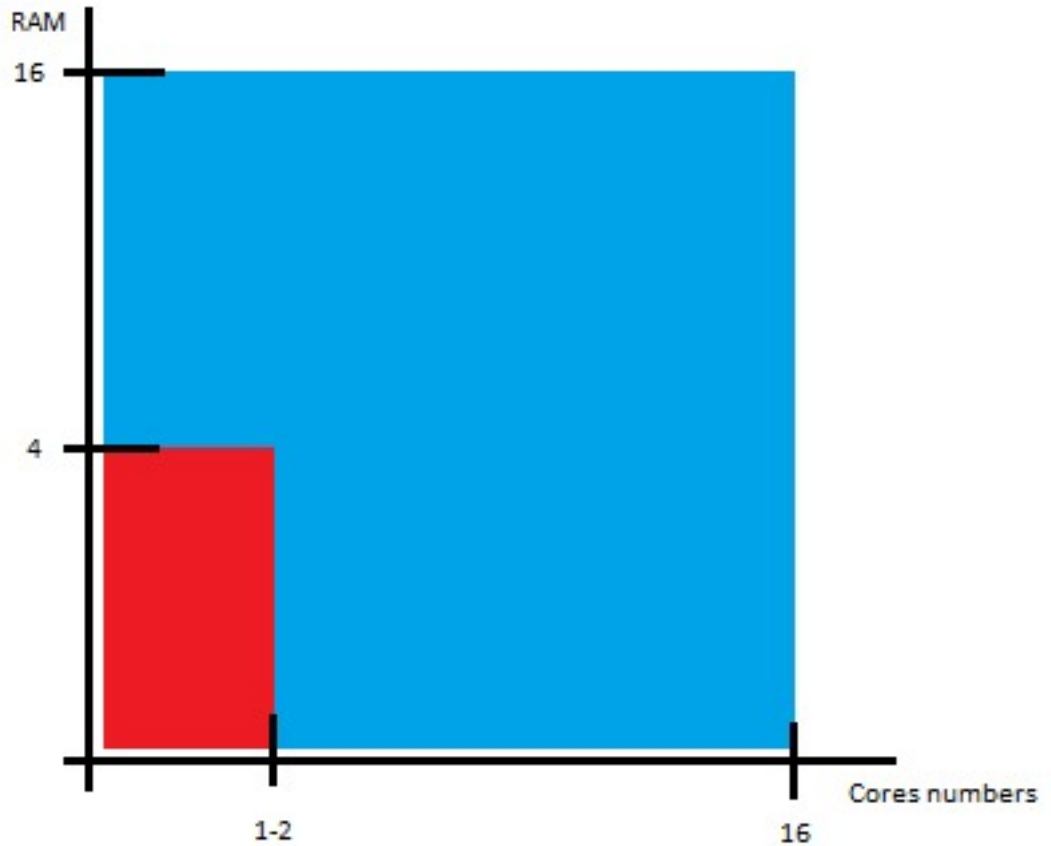
- **История оптимизации проекта**
  - Первоначальные требования и их изменения
  - История “ужатия” проекта до требований заказчика
- **Исследование возможностей JVM**
  - Тестовое приложение, режимы тестирования
  - Быстрая оптимизация работы приложения
  - Типы ссылок и стратегии сборки мусора, зависимость от количества объектов

Требования

- **Value-at-Risk Calculator**
  - Много простых вычислений
  - Разветвленное двоичное дерево
  - Операции работы со строками
- **Первичные требования: 16 Gb, 16 ядер, Red Hat Linux, Java 1.6, 5-15 минут работы**
- **Жестко заданная платформа: HotSpot, Java, Cassandra, etc**
- **Крайне важно уложиться в требования по “железу”**

Изменение требований

- Обусловлено запусом системы на гриде
- Изменение требований: 4 Gb, 1 ядро, 1 час работы
- Характеристики нодов жестко заданы



Первые шаги

- **Работа с виртуальной машиной**
  - Тюнинг: -Xmx, -Xms, -server, -XX:NewSize, -XX:NewRatio, -XX:SurvivorRatio, -XX:UseBiasedLocking, -XX:+UseStringCache, etc
  - Стратегии сборки мусора
- **Работа с кэшами**
  - Удаление ненужных
  - Более гибкая чистка существующих

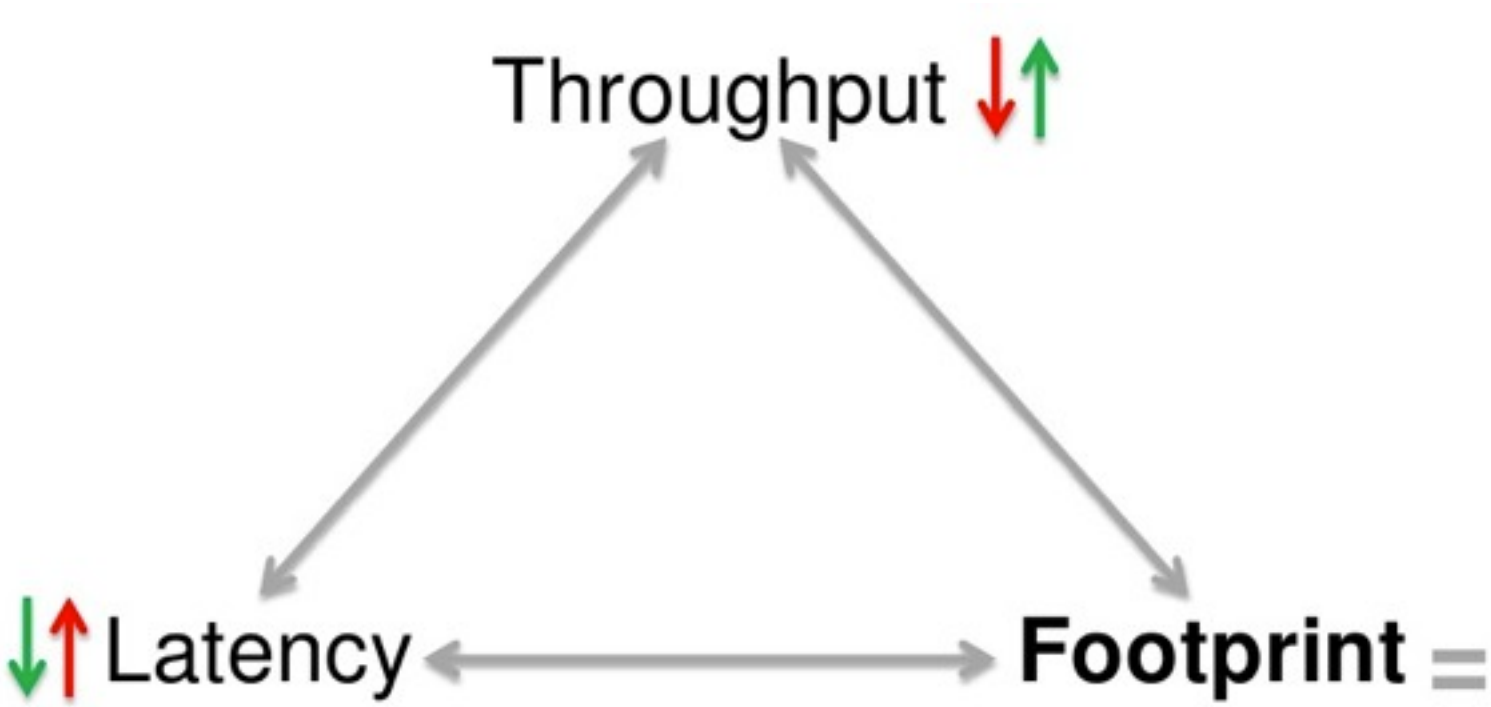
Тестовое приложение

- “Чистое” java приложение
- Используем все типы ссылок: Strong, Soft, Weak, Phantom
- Ограничения на работу: 1GB, 2 ядра
- Два типа объектов: большие (2 мб) и маленькие (16 кб)
- Тестирование на различных типах VM: HotSpot, Jrockit, OpenJVM
- Инструменты замера: VisualVM, MXBean

Критерии оценки

Три характеристики:

- Throughput – объем ресурсов, затрачиваемый на GC
- Latency – на какое время прерывается приложение
- Footprint – объем используемой памяти



Сборщики мусора

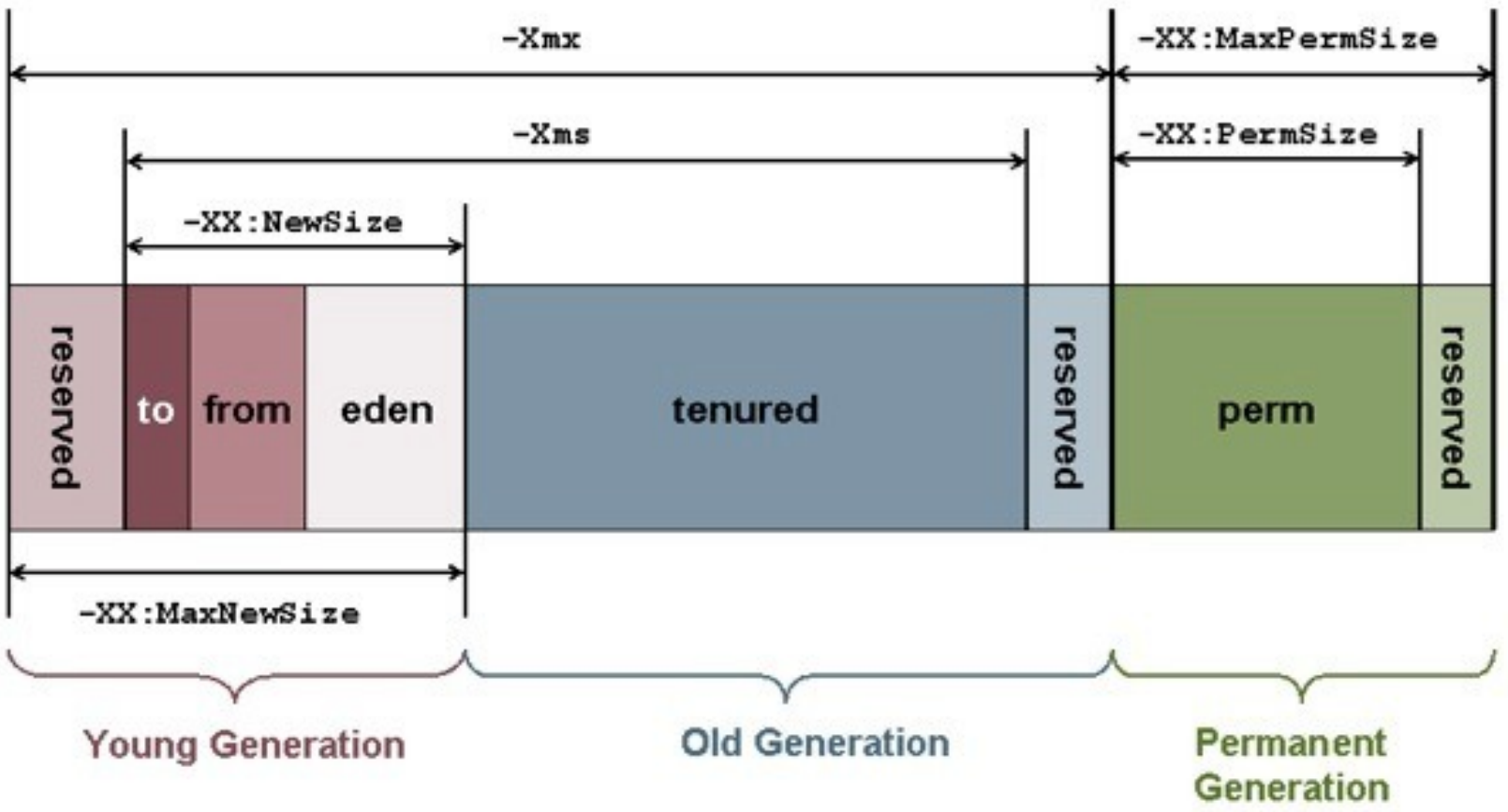
- Serial (-XX:+UseSerialGC)
- Parallel (-XX:+UseParallelGC)
- Parallel compacting (-XX:+UseParalleOldGC)
- Concurrent Mark-Sweep (-XX:+UseConcMarkSweepGC)
- G1 collector (-XX:+UseG1GC)

Оптимален для нас - Concurrent Mark-Sweep

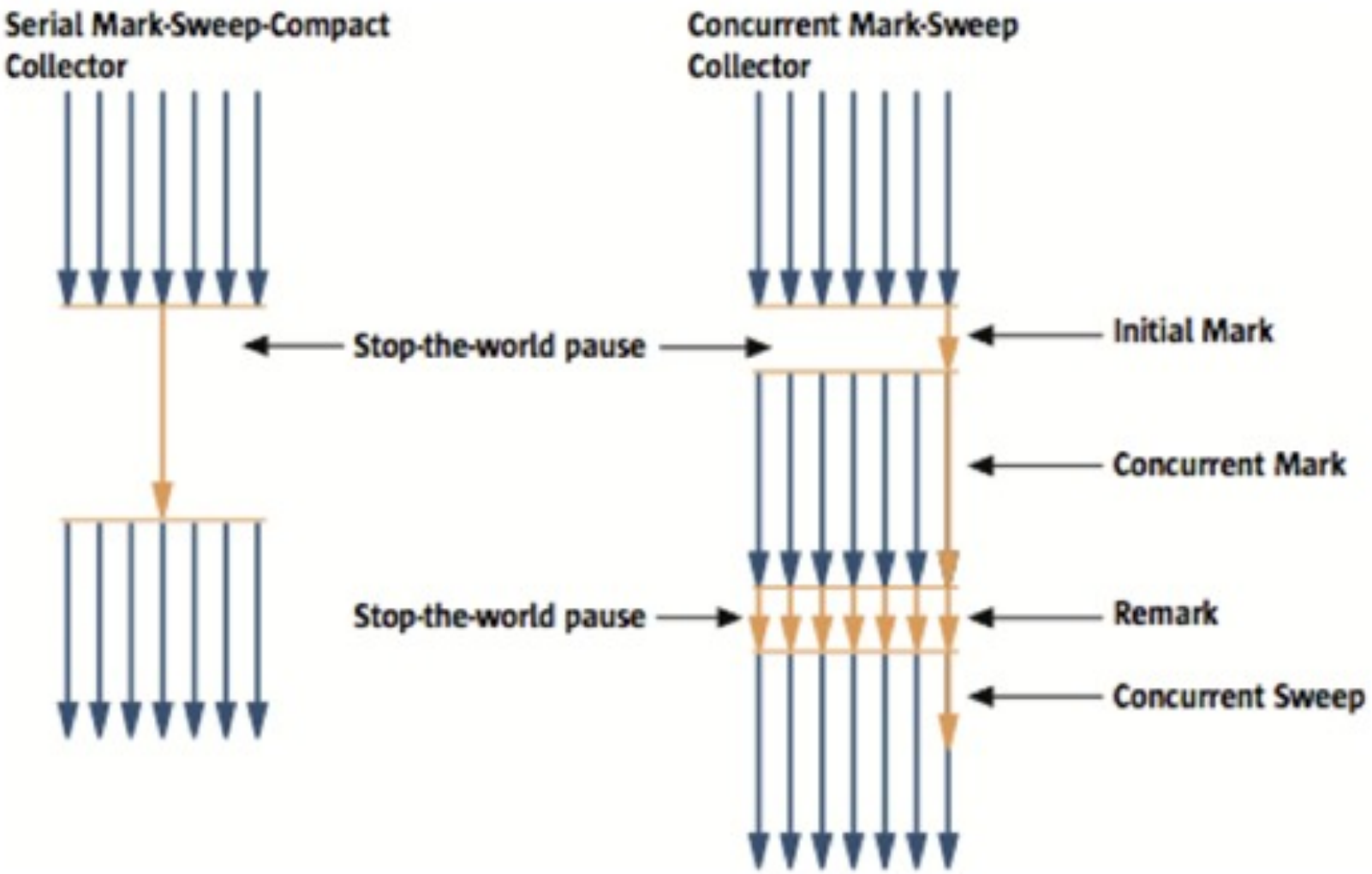
Анатомия Heap

Слабая гипотеза о поколениях:

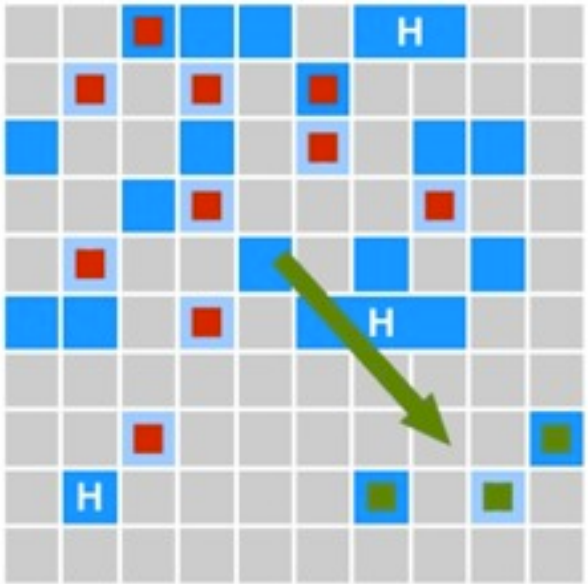
- Большинство объектов временные
- Старые объекты редко ссылаются на молодые

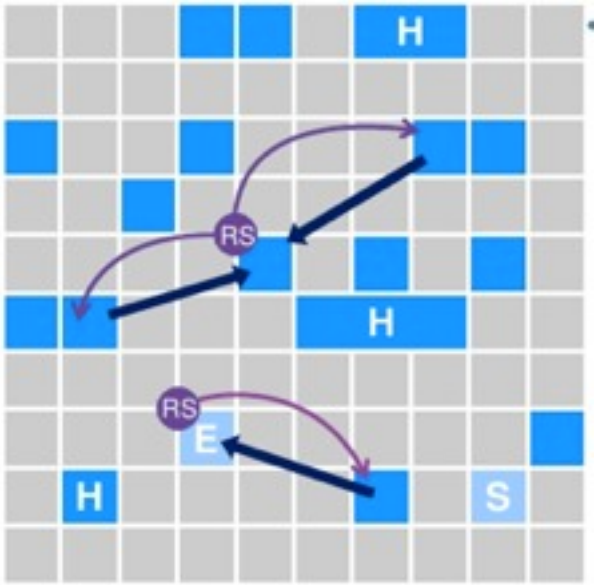


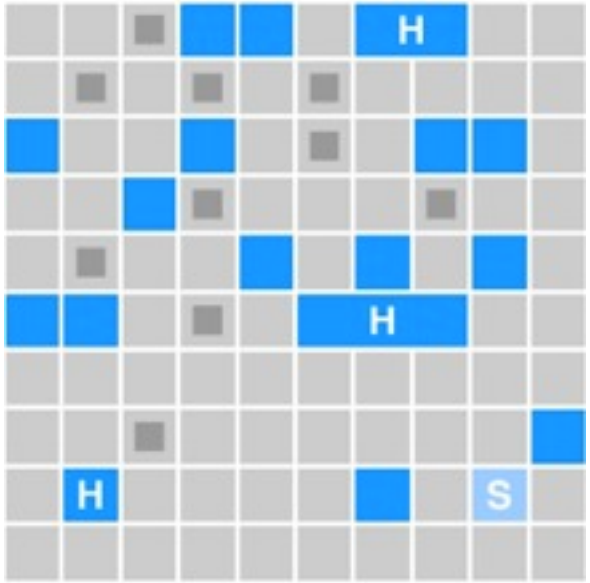
Concurrent Mark-Sweep



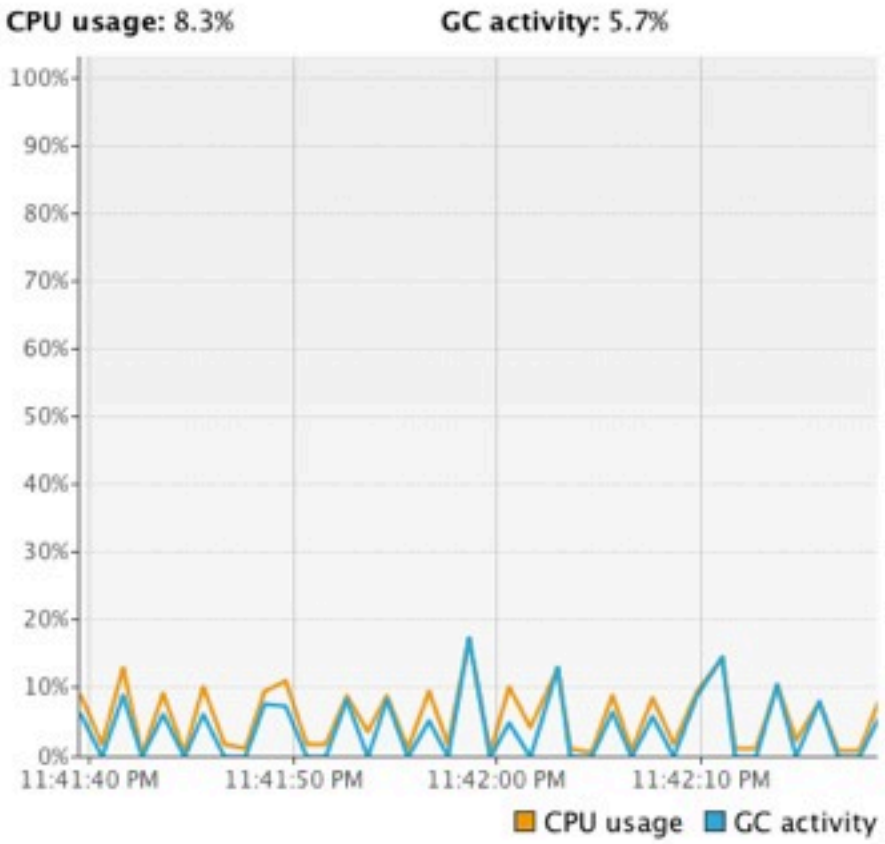
Garbage First

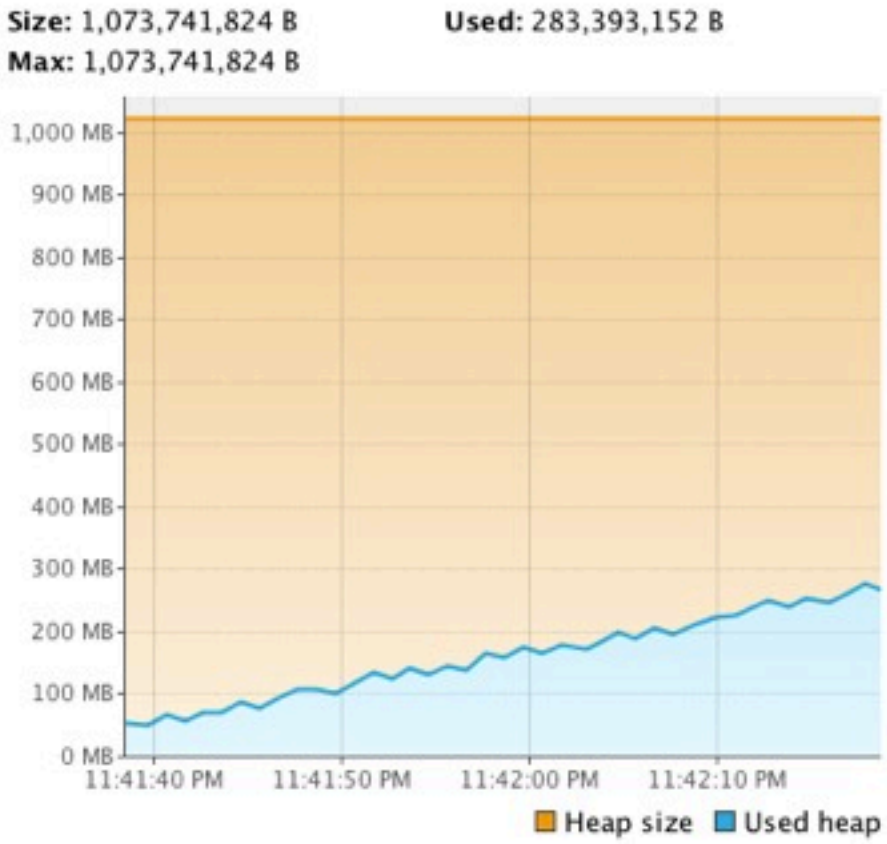






System.gc()


CPU usage: 8.3%
GC activity: 5.7%
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
11:41:40 PM
11:41:50 PM
11:42:00 PM
11:42:10 PM
CPU usage
GC activity


Size: 1,073,741,824 B
Used: 283,393,152 B
Max: 1,073,741,824 B
1,000 MB
900 MB
800 MB
700 MB
600 MB
500 MB
400 MB
300 MB
200 MB
100 MB
0 MB
11:41:40 PM
11:41:50 PM
11:42:00 PM
11:42:10 PM
Heap size
Used heap

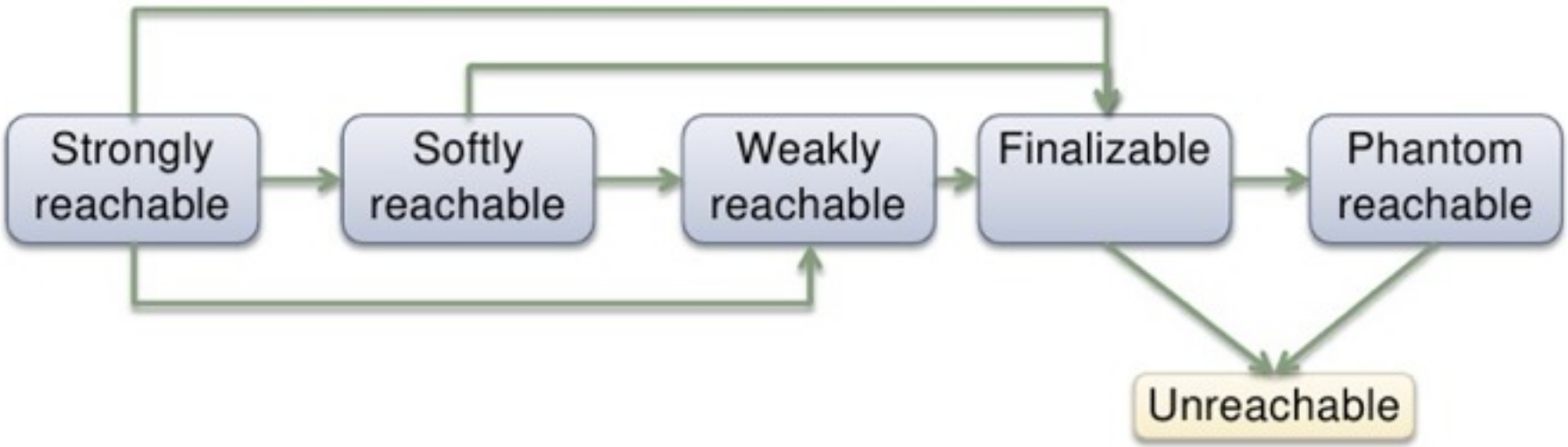
Типы ссылок и работа с ними

- Strong/Hard
- Soft
- Weak
- Phantom

Strong и Phantom сразу отошли на второй план

Soft и Weak наиболее оптимальны для работы с объектами

Типы ссылок и работа с ними

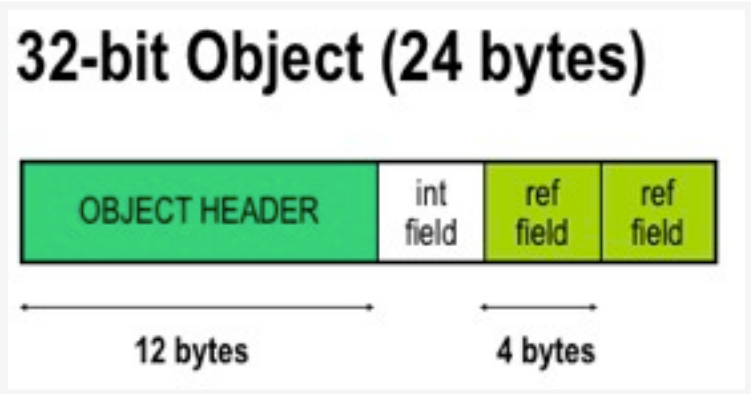


PhantomReference: example

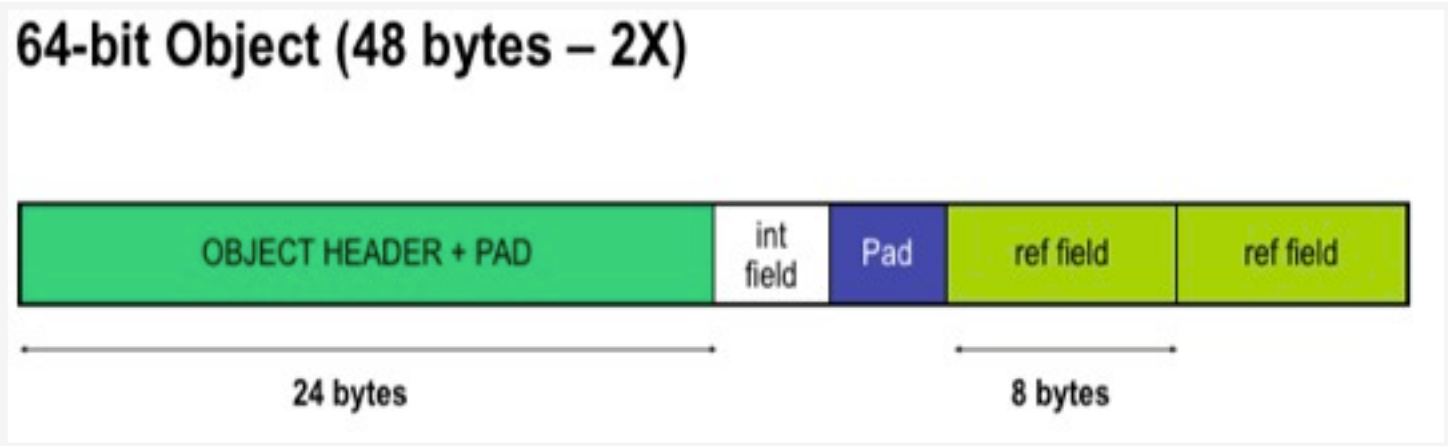

```
...

private ReferenceQueue<? super T> referenceQueue;

public PhantomReferencePollingThreadTest(ReferenceQueue<? super T>
referenceQueue{
   this.referenceQueue = referenceQueue;

   @Override
   public void run() {
        Reference<?> ref = null;
        while ( (ref = referenceQueue.poll()) == null ){
        // Waiting
        }

        if (ref instanceof PhantomReferenceTest<?>){
                ( ( PhantomReferenceTest<T> ) ref ).cleanup();
        }
    }
}
...
```

“Сжатые” ссылки

- Распаковка “сжатых” указателей дешевле
- Работают для кучи размером не более 32Gb



8 байт



16 байт

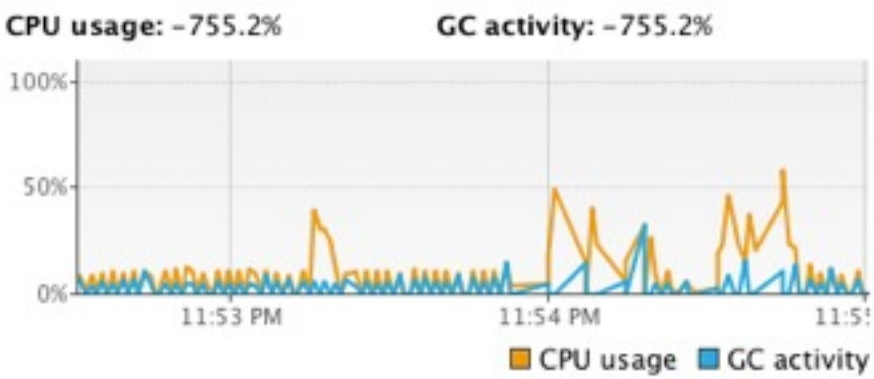
Инстументы анализа

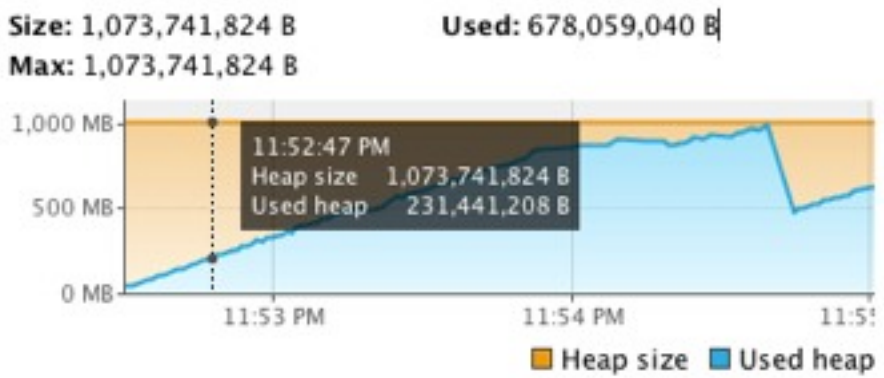
- **Средства JVM**
  - -XX:+PrintGCDetails, -XX:+PrintGC, -XX:PrintReferenceGC
- **MXBean**
  - Работает удаленно
  - Не всегда удобно обрабатывать информацию
- **VisualVM**
  - Промахи по размеру объектов
  - Наглядное представление

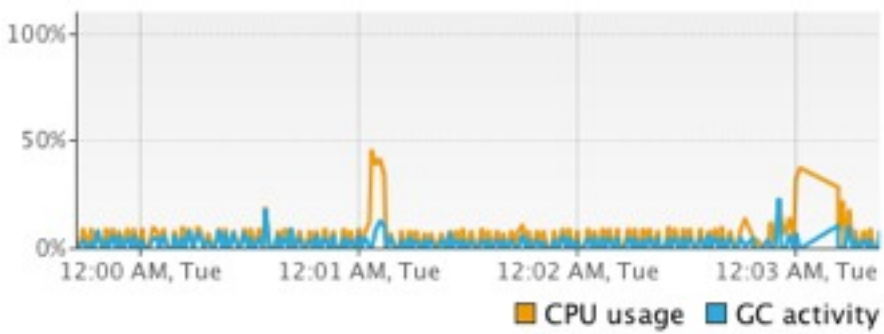
Уменьшение набора объектов

- Замена объектов на ссылки
- Уменьшение количества строковых данных
- Уменьшение количества создоваемых объектов: с 3М до 600К

Исходные результаты работы







Быстрая оптимизация

- **Использование следующих ключей:**
  - -server
  - -XX:+UseCompressedStrings, -XX:+UseStringCache, -XX:+OptimizeStringConcat
- **Результат:**
  - Big Objects (BO): особых изменений нет
  - Small Objects (SO): задержка на stop-the-world фазу упала примерно в 2 раза

Стратегии сборки мусора

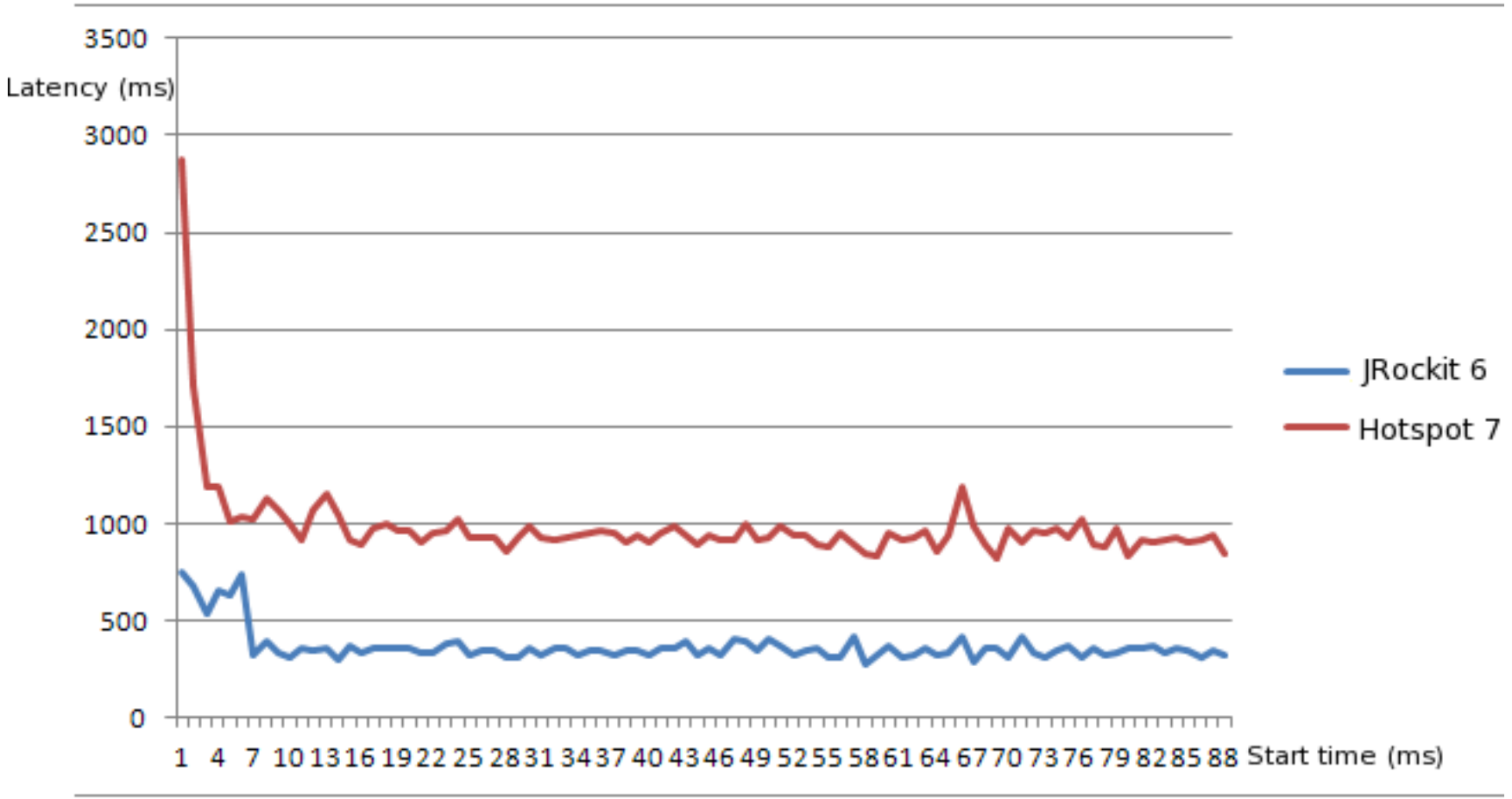
	Serial	P	PC	CMS	G1
Young/ Old	2.1	2	1.9	2.5	2.3
Latency	12	15	78	185	112
S1/S2	0.8	0.8	0.6	0.5	-

JRockit JVM

Особенности:

- Нет разбиения Young на поколения
- Динамические режимы
  - throughput
  - pausetime
  - deterministic

JRockit vs HotSpot



OpenJVM vs HotSpot

Использование следующих тэгов:

- Существенных особенностей обнаружено не было
- Несущественные преимущества G1
- Оптмизирована под Unix системы (Red Hat сообщество – активный участник процесса)

Учитываем контекст задачи

Как специфика задачи может помочь при выборе параметров?

- **Работа со строками:**
  - -XX:+UseCompressedStrings, -XX:+UseStringCache, -XX:+OptimizeStringConcat
- **Поведение:**
  - -XX:<any_gc>, -XX:+UseBiasedLocking, -XX:DDisableExplicitGC, -XX:+AggresiveOpts
- **Поведение GC:**
  - -XX:MaxNew/PermSize, -XX:<any>Ratio, -XX:<generations_size>

Выбор VM: 32 vs 64

Имеет ли смысл выбирать x64 JVM?

- Размеры указателей увеличиваются
- Сжатые указатели
- x64 работают быстрее, с аналогичными указателями по размеру

Small Objects: Поведение системы

- **Никаких аномалий замечено не было**
- **Серьезный прирост по производительности:** G1 и CMS
- **Вспомогательные тэги:**
  - XX:<any_strings>, -server
  - XX:AggresiveOpts

Big Objects: Поведение системы

- CMS тратитм много времени на фазу “упаковки”
- G1 в 1.6 был экспериментален и не рассматривался нами
- Работа с размерами поколений сильно увеличила производительность CMS
- P, PC не могут нормально работать в такой системе, тк OldGeneration перегружен

Итоги и результаты

- Используйте ключи оптимизации:
  - -server, -XX:<strings>
- Выбор GC
  - G1, если доступен
  - Concurrent Mark-Sweep или Parallel Compacting
- Отслеживание количества создаваемых объектов
- На тонкую настройку уходит много времени, но результаты могут быть впечатляющими!
- Правильная работа с кэшами: внедряйте инструменты мониторинга на ранних этапах
- Узнавайте требования как можно раньше! :)

Спасибо за внимание!

Александр Макаров

E-mail: amsilf@gmail.com

Skype: makarovalexader1