Ходячие объекты-мертвецы

или

GC всегда прав

Иван Углянский
Excelsior LLC

EXCELSIOR

Иван Углянский

JVM инженер в проекте Excelsior JET

- разрабатываю рантайм

JUGNsk co-lead

iugliansky@excelsior-usa.com

@dbg_nsk

EXCELSIOR

Excelsior JET JVM

Java compatible

- Лицензиаты Oracle
- JCK каждую ночь

Excelsior JET JVM

Java compatible

- Лицензиаты Oracle
- JCK каждую ночь

Честный AOT

- Межпроцедурные оптимизации
- JIT для подгружаемых классов
- Интерпретатора нет

EXCELSIOR

Excelsior JET JVM

Java compatible

- Лицензиаты Oracle
- JCK каждую ночь

Честный AOT

- Межпроцедурные оптимизации
- JIT для подгружаемых классов
- Интерпретатора нет

Автоматическое управление памятью (Сборка мусора)

EXCELSIOR

Сборка мусора

EXCELSIOR

А что в спеке?

The Java Virtual Machine Specification, (§2.5.3):

> *Heap storage for objects is reclaimed by an automatic storage management system (known as a garbage collector); objects are never explicitly deallocated. The Java Virtual Machine assumes no particular type of automatic storage management system, and the storage management technique may be chosen according to the implementor's system requirements.*

EXCELSIOR

Сборка мусора

Ожидание:

- JVM собирает мусор, как хочет (в рамках корректности)

EXCELSIOR

Сборка мусора

Ожидание:

- JVM собирает мусор, как хочет (в рамках корректности)
- Разработчик не задумывается об управлении памятью

Сборка мусора

Реальность:

```
java.lang.NullPointerException
at java.util.ResourceBundle$Control.newBundle(ResourceBundle.java:2640)
at java.util.ResourceBundle.loadBundle(ResourceBundle.java:1501)
at java.util.ResourceBundle.findBundle(ResourceBundle.java:1465)
at java.util.ResourceBundle.findBundle(ResourceBundle.java:1419)
at java.util.ResourceBundle.getBundleImpl(ResourceBundle.java:1361)
... 3 more
```

Сборка мусора

Реальность:

```
java.lang.NullPointerException
at
at
at
at
at
..
```

```
Exception in thread "1" java.lang.OutOfMemoryError: Java heap space
        at StressTimer$Task.run(StressTimer.java:10)
        at java.util.TimerThread.mainLoop(Unknown Source)
        at java.util.TimerThread.run(Unknown Source)
Exception in thread "2" java.lang.OutOfMemoryError: Java heap space
        at StressTimer$Task.run(StressTimer.java:10)
        at java.util.TimerThread.mainLoop(Unknown Source)
        ...
```

EXCELSIOR

Сборка мусора

Реальность:

```
java.lang.NullPointerException
at
at
at
at
at
..
```

```
Exception in thread "1" java.lang.OutOfMemoryError: Java heap space

Exceptio
```

```
Java HotSpot(TM) 64-Bit Server VM warning: INFO:
os::commit_memory(0x00000000e0000000, 257949696, 0) failed;
error='Not enough space' (errno=12)
#
# There is insufficient memory for the Java Runtime Environment
to continue.
# Native memory allocation (mmap) failed to map 257949696
bytes for committing reserved memory.
# An error report file with more information is saved as:
```

EXCELSIOR

Когда конкретно GC придет за объектом?

EXCELSIOR

В этом докладе

- Про политики GC относительно времени жизни объектов на примерах
- Как это влияет на исполнение?
- Как ~~не получить OOM~~ это учитывать в коде?

EXCELSIOR

Разминка

EXCELSIOR

Разминка

```
public static void main(String[] args) {
    final int size = 512*1024*1024 / (Integer.SIZE / 8);
    int[] arr1 = new int[size];
    System.out.println("arr1 (" + arr1 + ") allocated");
    System.gc();
    int[] arr2 = new int[size];
    System.out.println("arr2 (" + arr2 + ") allocated");
}
```

512mb

EXCELSIOR

Разминка

```
public static void main(String[] args) {
    final int size = 512*1024*1024 / (Integer.SIZE / 8);
    int[] arr1 = new int[size];
    System.out.println("arr1 (" + arr1 + ") allocated");
    System.gc();
    int[] arr2 = new int[size];
    System.out.println("arr2 (" + arr2 + ") allocated");
}
```

512mb

EXCELSIOR

Разминка

```
public static void main(String[] args) {
    final int size = 512*1024*1024 / (Integer.SIZE / 8);
    int[] arr1 = new int[size];                                512mb
    System.out.println("arr1 (" + arr1 + ") allocated");
    System.gc();
    int[] arr2 = new int[size];                                512mb
    System.out.println("arr2 (" + arr2 + ") allocated");
}
```

EXCELSIOR

Разминка

```
public static void main(String[] args) {
    final int size = 512*1024*1024 / (Integer.SIZE / 8);
    int[] arr1 = new int[size];                                512mb
    System.out.println("arr1 (" + arr1 + ") allocated");
    System.gc();
    int[] arr2 = new int[size];                                512mb
    System.out.println("arr2 (" + arr2 + ") allocated");
}
```

```
java -Xmx768m Test
```

EXCELSIOR

Разминка

```
public static void main(String[] args) {
    final int size = 512*1024*1024 / (Integer.SIZE / 8);
    int[] arr1 = new int[size];                                  512mb
    System.out.println("arr1 (" + arr1 + ") allocated");
    System.gc();
    int[] arr2 = new int[size];                                  512mb
    System.out.println("arr2 (" + arr2 + ") allocated");
}
```

```
java -Xmx768m Test
```

→

```
arr1 ([I@4aa8f0b4) allocated
Exception in thread "main"
java.lang.OutOfMemoryError: ...
```

EXCELSIOR

Разминка

```
public static void main(String[] args) {
    final int size = 512*1024*1024 / (Integer.SIZE / 8);
    int[] arr1 = new int[size];                                512mb
    System.out.println("arr1 (" + arr1 + ") allocated");
    System.gc();
    int[] arr2 = new int[size];                                512mb
    System.out.println("arr2 (" + arr2 + ") allocated");
}
```

```
java -Xcomp -Xmx768m Test
```

EXCELSIOR

Разминка

```
public static void main(String[] args) {
    final int size = 512*1024*1024 / (Integer.SIZE / 8);
    int[] arr1 = new int[size];                                  512mb
    System.out.println("arr1 (" + arr1 + ") allocated");
    System.gc();
    int[] arr2 = new int[size];                                  512mb
    System.out.println("arr2 (" + arr2 + ") allocated");
}
```

```
java -Xcomp -Xmx768m Test
```

→

```
arr1 ([I@4aa8f0b4) allocated
arr2 ([I@9e89d68) allocated
```

EXCELSIOR

Разминка

```
public static void main(String[] args) {
    final int size = 512*1024*1024 / (Integer.SIZE / 8);
    int[] arr1 = new int[size];
    System.out.println("arr1 (" + arr1 + ") allocated");
    System.gc();
    int[] arr2 = new int[size];
    System.out.println("arr2 (" + arr2 + ") allocated");
}
```

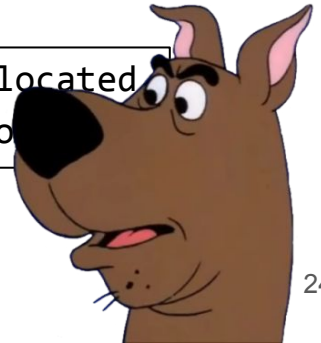
512mb

512mb

```
java -Xcomp -Xmx768m Test
```

→

```
arr1 ([I@4aa8f0b4) allocated
arr2 ([I@9e89d68) allo
```



EXCELSIOR

Разминка

```
public static void main(String[] args) {
    final int size = 512*1024*1024 / (Integer.SIZE / 8);
    int[] arr1 = new int[size];
    System.out.println("arr1 (" + arr1 + ") allocated");
    System.gc();
    int[] arr2 = new int[size];
    System.out.println("arr2 (" + arr2 + ") allocated");
}
```

Имеем право собирать первый массив?

EXCELSIOR

Разминка

```
public static void main(String[] args) {
    final int size = 512*1024*1024 / (Integer.SIZE / 8);
    int[] arr1 = new int[size];
    System.out.println("arr1 (" + arr1 + ") allocated");
    System.gc();
    int[] arr2 = new int[size];
    System.out.println("arr2 (" + arr2 + ") allocated");
}
```

Область видимости переменной arr1

EXCELSIOR

Разминка

```
public static void main(String[] args) {
    final int size = 512*1024*1024 / (Integer.SIZE / 8);
    int[] arr1 = new int[size];
    System.out.println("arr1 (" + arr1 + ") allocated");
    System.gc();
    int[] arr2 = new int[size];
    System.out.println("arr2 (" + arr2 + ") allocated");
}
```

Область видимости переменной arr1

Область **жизни** переменной arr1

EXCELSIOR

Разминка

```
public static void main(String[] args) {
    final int size = 512*1024*1024 / (Integer.SIZE / 8);
    int[] arr1 = new int[size];
    System.out.println("arr1 (" + arr1 + ") allocated");
    System.gc();
    int[] arr2 = new int[size];
    System.out.println("arr2 (" + arr2 + ") allocated");
}
```

Здесь GC имеет право собрать первый массив

EXCELSIOR

Разминка

```
public static void main(String[] args) {
    final int size = 512*1024*1024 / (Integer.SIZE / 8);
    int[] arr1 = new int[size];                                  512mb
    System.out.println("arr1 (" + arr1 + ") allocated");
    System.gc();
    int[] arr2 = new int[size];                                  512mb
    System.out.println("arr2 (" + arr2 + ") allocated");
}
```

```
java -Xmx768m Test
```

→

```
arr1 ([I@4aa8f0b4) allocated
Exception in thread "main"
java.lang.OutOfMemoryError: ...
```

EXCELSIOR

Разминка

```
public static void main(String[] args) {
    final int size = 512*1024*1024 / (Integer.SIZE / 8);
    int[] arr1 = new int[size];
    System.out.println("arr1 (" + arr1 + ") allocated");
    System.gc();
    int[] arr2 = new int[size];
    System.out.println("arr2 (" + arr2 + ") allocated");
}
```

Интерпретатор не вычисляет время жизни!

EXCELSIOR

Разминка

```
public static void main(String[] args) {
    final int size = 512*1024*1024 / (Integer.SIZE / 8);
    int[] arr1 = new int[size];
    System.out.println("arr1 (" + arr1 + ") allocated");
    System.gc();
    int[] arr2 = new int[size];
    System.out.println("arr2 (" + arr2 + ") allocated");
}
```

Интерпретатор не вычисляет время жизни!

(предполагает худший случай)

EXCELSIOR

История #1. Объекты-призраки

История #1. Объекты-призраки

Слабые ссылки:

EXCELSIOR

История #1. Объекты-призраки

Слабые ссылки:

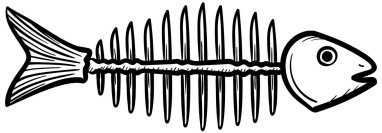
1. java.lang.ref.*
2. Не останавливают GC от сборки референта

EXCELSIOR

История #1. Объекты-призраки

Слабые ссылки:

1. java.lang.ref.*
2. Не останавливают GC от сборки референта
3. «Протухают»

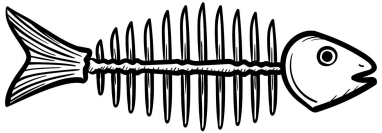


EXCELSIOR

История #1. Объекты-призраки

Слабые ссылки:

1. java.lang.ref.*
2. Не останавливают GC от сборки референта
3. «Протухают»



EXCELSIOR

Пример

```
public static void main(String[] args) {
    Object obj = new Object();
    WeakReference ref = new WeakReference<>(obj);
    System.gc();
    System.out.println(ref.get().hashCode());
}
```

EXCELSIOR

Пример

```
public static void main(String[] args) {
    Object obj = new Object();
    WeakReference ref = new WeakReference<>(obj);
    System.gc();
    System.out.println(ref.get().hashCode());
}
```

EXCELSIOR

Пример

```
public static void main(String[] args) {
    Object obj = new Object();
    WeakReference ref = new WeakReference<>(obj);
    System.gc();
    System.out.println(ref.get().hashCode());
}
```

Пример

```
public static void main(String[] args) {
    Object obj = new Object();
    WeakReference ref = new WeakReference<>(obj);
    System.gc();
    System.out.println(ref.get().hashCode());
}
```

java Test → *1252585652*

Пример

```
public static void main(String[] args) {
    Object obj = new Object();
    WeakReference ref = new WeakReference<>(obj);
    System.gc();
    System.out.println(ref.get().hashCode());
}
```

```
java Test
```

→

```
1252585652
```

```
java -Xcomp Test
```

→

```
Exception in thread "main"
java.lang.NullPointerException
```

EXCELSIOR

Пример

```
public static void main(String[] args) {
    Object obj = new Object();
    WeakReference ref = new WeakReference<>(obj);
    System.gc();
    System.out.println(ref.get().hashCode());
}
```

java Test → *1252585652*

java -Xcomp Test →

```
Exception in thread "main"
java.lang.NullPointerException
```

EXCELSIOR

Объекты-призраки

Но ведь это искусственный пример?

Объекты-призраки

Но ведь это искусственный пример?

```
java.lang.NullPointerException
at java.util.ResourceBundle$Control.newBundle(ResourceBundle.java:2640)
at java.util.ResourceBundle.loadBundle(ResourceBundle.java:1501)
at java.util.ResourceBundle.findBundle(ResourceBundle.java:1465)
at java.util.ResourceBundle.findBundle(ResourceBundle.java:1419)
at java.util.ResourceBundle.getBundleImpl(ResourceBundle.java:1361)
... 3 more
```

Реальный пример

```
private static ResourceBundle getBundleImpl(...,
                                            ClassLoader loader,
                                            ...) {
    ...
    CacheKey cacheKey = new CacheKey(baseName, locale, loader);
    ...
    // обращение к loader через cacheKey
    ...
    return bundle;
}
```

EXCELSIOR

Реальный пример

```
private static ResourceBundle getBundleImpl(...,
                                            ClassLoader loader,
                                            ...) {
    ...
    CacheKey cacheKey = new CacheKey(baseName, locale, loader);
    ...
    // обращение к loader через cacheKey
    ...
    return bundle;
}
```

Слабая ссылка

Последнее использование

Реальный пример

- JDK 1.8.0_181
- java.util.ResourceBundle.getBundle(...)
- Для проявления: -Xcomp + итерации
- JDK-8209184

EXCELSIOR

Что с этим делать?

Что с этим делать?

- Перед использованием проверять, что ссылка не протухла

Что с этим делать?

- Перед использованием проверять, что ссылка не протухла
- Продлить жизнь объекта!

Что с этим делать?

```
public static void main(String[] args) {
    Object obj = new Object();
    WeakReference ref = new WeakReference<>(obj);
    System.gc();
    System.out.println(ref.get().hashCode());
}
```

EXCELSIOR

Что с этим делать?

```
public static void main(String[] args) {
    Object obj = new Object();
    WeakReference ref = new WeakReference<>(obj);
    System.gc();
    System.out.println(ref.get().hashCode());
    // use obj ???
}
```

EXCELSIOR

Что с этим делать?

```
public static void main(String[] args) {
    Object obj = new Object();
    WeakReference ref = new WeakReference<>(obj);
    System.gc();
    System.out.println(ref.get().hashCode());
    // use obj ???
}
```

Но какое именно использование добавить?

EXCELSIOR

Что с этим делать?

```
public static void main(String[] args) {
    Object obj = new Object();
    WeakReference ref = new WeakReference<>(obj);
    System.gc();
    System.out.println(ref.get().hashCode());
    // use obj ???
}
```

Но какое именно использование добавить?

Как убедить компилятор его не выкидывать?

EXCELSIOR

Использование

? Передать в метод

? Записать в поле

EXCELSIOR

Использование

? Передать в метод

? Записать в поле

✓ `Reference.reachabilityFence(obj)`

Since Java 9

EXCELSIOR

Лечение

```
public static void main(String[] args) {
    Object obj = new Object();
    WeakReference ref = new WeakReference<>(obj);
    System.gc();
    System.out.println(ref.get().hashCode());
    // use obj ???
}
```

EXCELSIOR

Лечение

```
public static void main(String[] args) {
    Object obj = new Object();
    WeakReference ref = new WeakReference<>(obj);
    System.gc();
    System.out.println(ref.get().hashCode());
    Reference.reachabilityFence(obj);
}
```

EXCELSIOR

Лечение

```
private static ResourceBundle getBundleImpl(...) {
    ...
    CacheKey cacheKey = new CacheKey(baseName, locale, module,
                                     callerModule);
    ...
    // keep callerModule and module reachable for as long
    // as we are operating with WeakReference(s) to them
    // (in CacheKey)
    ...
    Reference.reachabilityFence(callerModule);
    Reference.reachabilityFence(module);
    return bundle;
}
```

Объекты-призраки

Подумаешь, один пример!

Объекты-призраки

JDK-8212178

Объекты-призраки

com.sun.xml.internal.stream.util.ThreadLocalBufferAllocator:

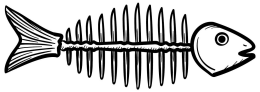
```
public static BufferAllocator getBufferAllocator() {
    SoftReference<BufferAllocator> bAllocatorRef = tlba.get();
    if (bAllocatorRef == null || bAllocatorRef.get() == null) {
        bAllocatorRef = new SoftReference(new BufferAllocator());
        tlba.set(bAllocatorRef);
    }

    return bAllocatorRef.get();
}
```

EXCELSIOR

Объекты-призраки

com.sun.xml.internal.stream.util.ThreadLocalBufferAllocator:

```
public static BufferAllocator getBufferAllocator() {
    SoftReference<BufferAllocator> bAllocatorRef = tlba.get();
    if (bAllocatorRef == null || bAllocatorRef.get() == null) {
        bAllocatorRef = new SoftReference(new BufferAllocator());
        tlba.set(bAllocatorRef);
    }

    return bAllocatorRef.get();
}
```



Выводы

- GC имеет право собрать объект после последнего использования
- Компилятор может выкидывать использования при оптимизации
- Reference.reachabilityFence для продления жизни объекта

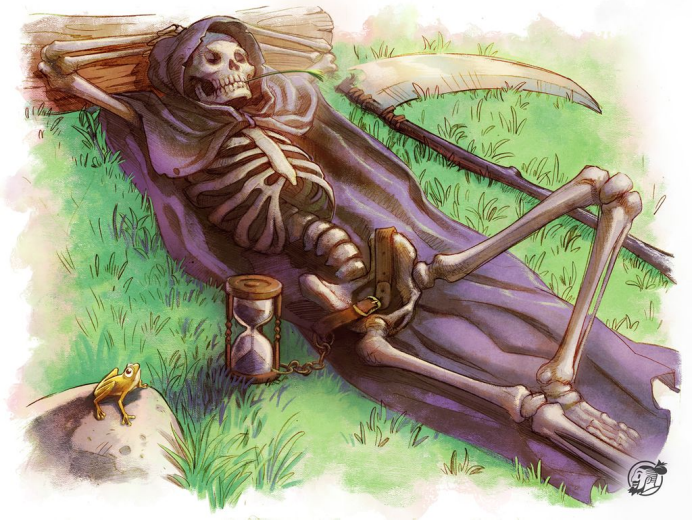
EXCELSIOR

Ходячие объекты-мертвецы

EXCELSIOR

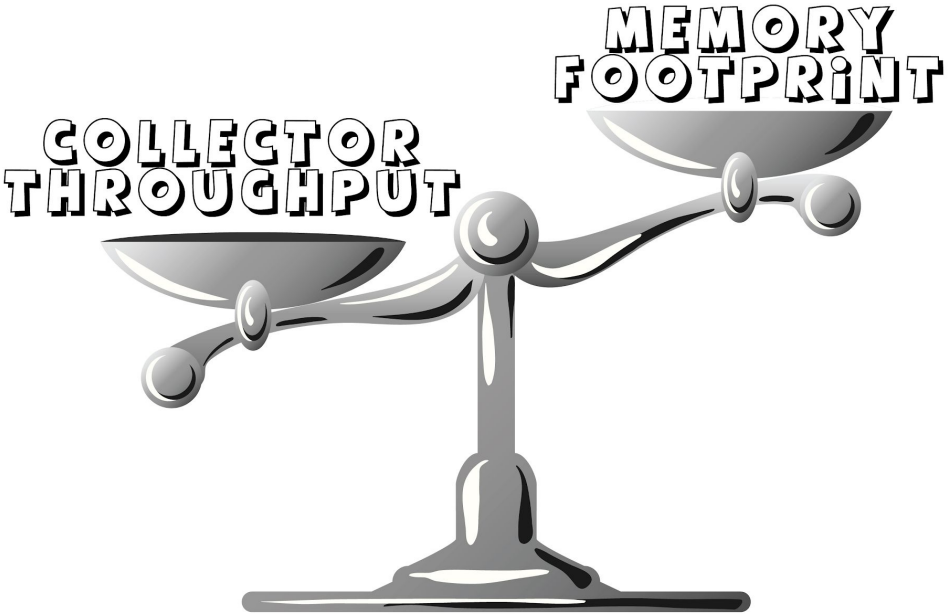
Ходячие объекты-мертвецы

Обратная проблема: объекты мертвы, но GC за ними приходить не торопится.



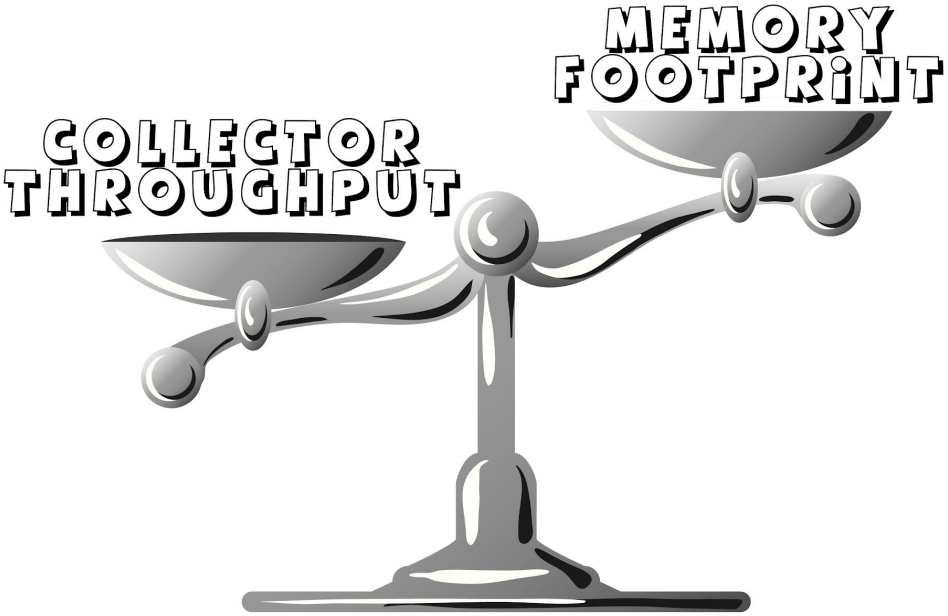
EXCELSIOR

Ходячие объекты-мертвецы



EXCELSIOR

Ходячие объекты-мертвецы



Memory Drag - память, которую GC «протаскивает» до следующей* сборки (характеристика алгоритма GC)

EXCELSIOR

Ходячие объекты-мертвецы

Большой **Memory Drag** ⇒

EXCELSIOR

Ходячие объекты-мертвецы

Большой **Memory Drag** ⇒

Много зомби-объектов ⇒

Ходячие объекты-мертвецы

Большой **Memory Drag** ⇒
Много зомби-объектов ⇒

Результат:

- Неожиданные OOM
- Проблемы с java.lang.ref

EXCELSIOR

История #2. F-reachables

История #2. F-reachables

`Object.finalize()` JavaDoc in JDK 9+:

> *The finalization mechanism is inherently problematic.*

EXCELSIOR

История #2. F-reachables

Object.finalize() JavaDoc in JDK 9+:

> *The finalization mechanism is inherently problematic. Finalization can lead to performance issues, deadlocks, and hangs.*

EXCELSIOR

История #2. F-reachables

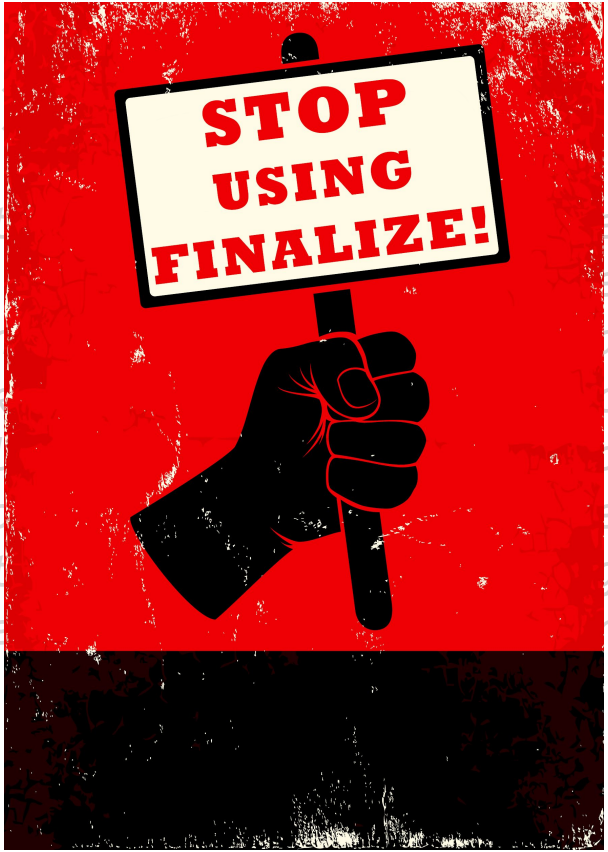
Object.finalize() JavaDoc in JDK 9+:

> *The finalization mechanism is inherently problematic. Finalization can lead to performance issues, deadlocks, and hangs. Errors in finalizers can lead to resource leaks; there is no way to cancel finalization if it is no longer necessary; and no ordering is specified among calls to finalize methods of different objects. Furthermore, there are no guarantees regarding the timing of finalization.*

EXCELSIOR

История #2. F-reachables

```
Object.finalize(

    The finalizat                    ently problematic.
    Finalization                     e issues,
    deadlocks, an                    alizers can lead to
    resource leak                    cancel finalization
    if it is no l                    o ordering is
    specified amo                    ethods of different
    objects. Furt                    guarantees
    regarding the                    n.
```



EXCELSIOR

История #2. F-reachables

- JDK 8 — 89 реализаций finalize()

EXCELSIOR

История #2. F-reachables

- JDK 8 — 89 реализаций finalize()
- JDK 9 — 87 реализаций (Object.finalize() is deprecated since 9)

EXCELSIOR

История #2. F-reachables

- JDK 8 — 89 реализаций finalize()
- JDK 9 — 87 реализаций (Object.finalize() is deprecated since 9)
- JDK 11 — 84 реализации

EXCELSIOR

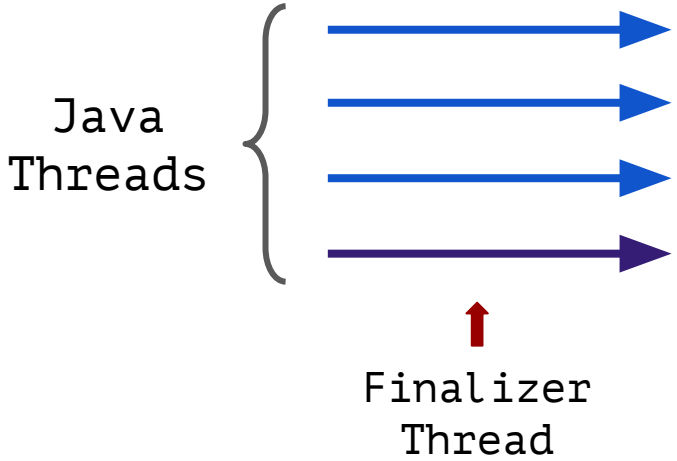
История #2. F-reachables

Объекты с нетривиальным finalize():

- живут дольше

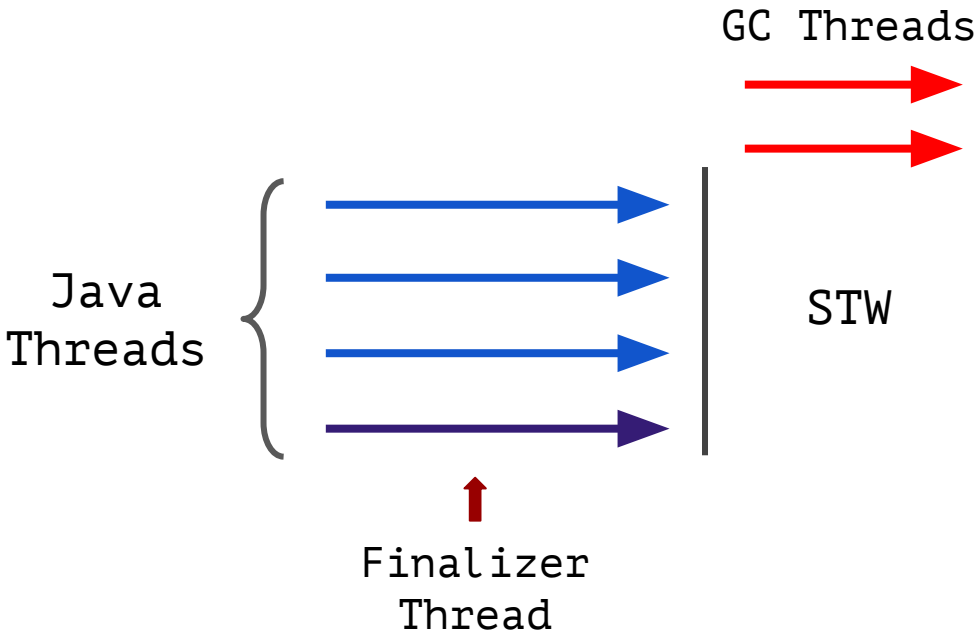
EXCELSIOR

История #2. F-reachables



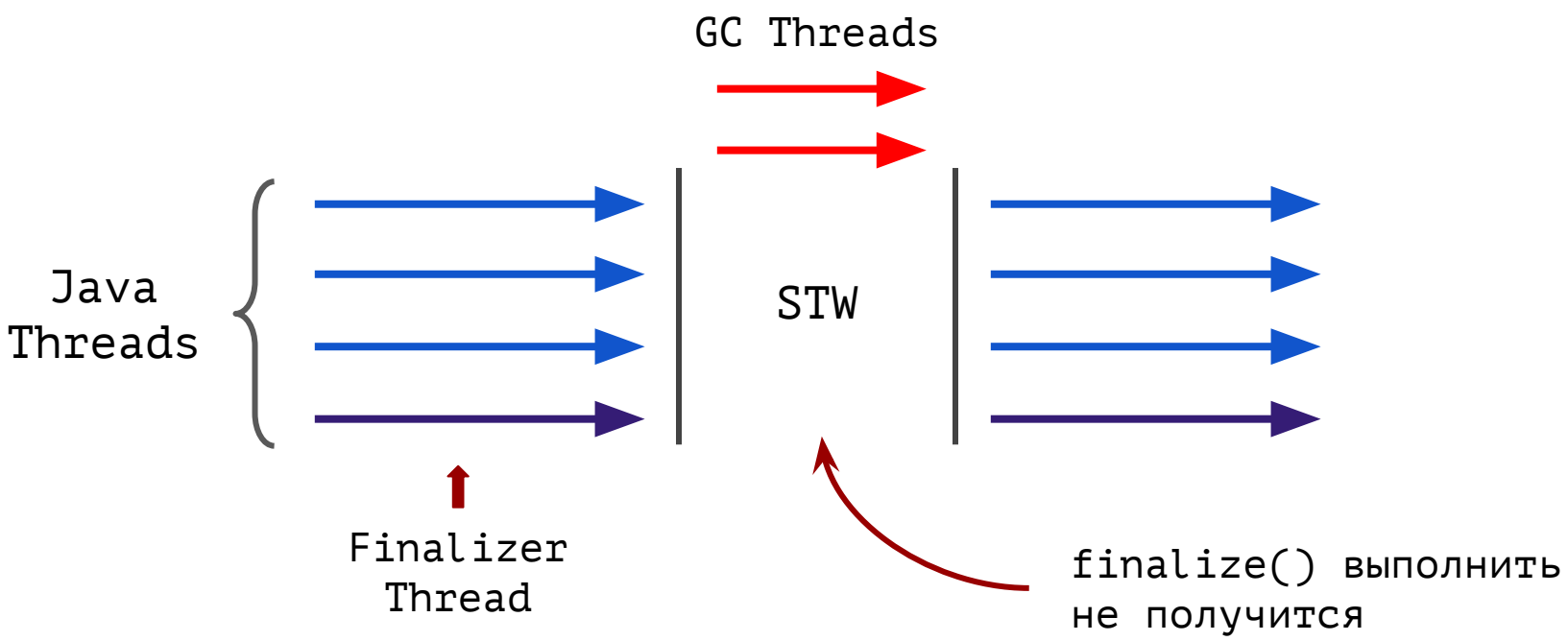
EXCELSIOR

История #2. F-reachables



EXCELSIOR

История #2. F-reachables



EXCELSIOR

История #2. F-reachables

Объекты с нетривиальным finalize():

- живут дольше: для STW как минимум (!) на один цикл GC

EXCELSIOR

История #2. F-reachables

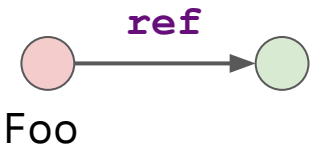
```
class Foo {
    Object ref;
    Foo(Object r) {
        ref = r;
    }
    void finalize() {
        ref.call();
    }
}
```

EXCELSIOR

История #2. F-reachables

```
class Foo {
    Object ref;
    Foo(Object r) {
        ref = r;
    }
    void finalize() {
        ref.call();
    }
}
```

ref используется из finalize, значит он тоже выживет



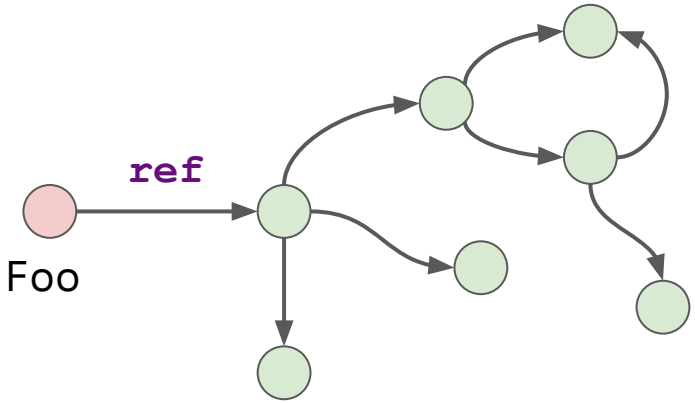
EXCELSIOR

История #2. F-reachables

```
class Foo {
    Object ref;
    Foo(Object r) {
        ref = r;
    }
    void finalize() {
        ref.call();
    }
}
```

`ref` используется из finalize, значит он тоже выживет

как и **все** достижимое из `ref`



EXCELSIOR

История #2. F-reachables

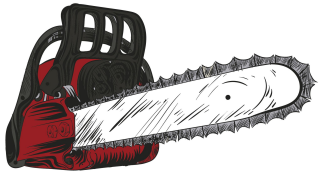
Объекты с нетривиальным finalize():

- живут дольше: для STW как минимум (!) на один цикл GC
- дополнительно удерживают все достижимые из них объекты (f-reachables)

EXCELSIOR

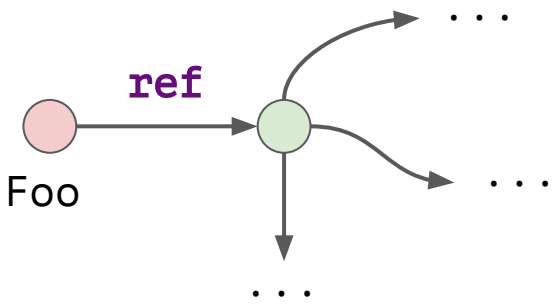
JVM vs F-reachables

Как JVM может бороться с F-reachables?



JVM vs F-reachables

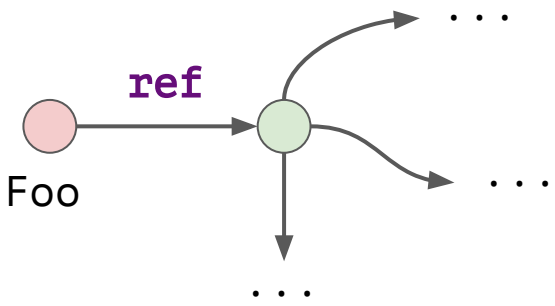
```
class Foo {
    Object ref;
    Foo(Object r) {
        ref = r;
    }
    void finalize() {
        ref.call();
    }
}
```



EXCELSIOR

JVM vs F-reachables

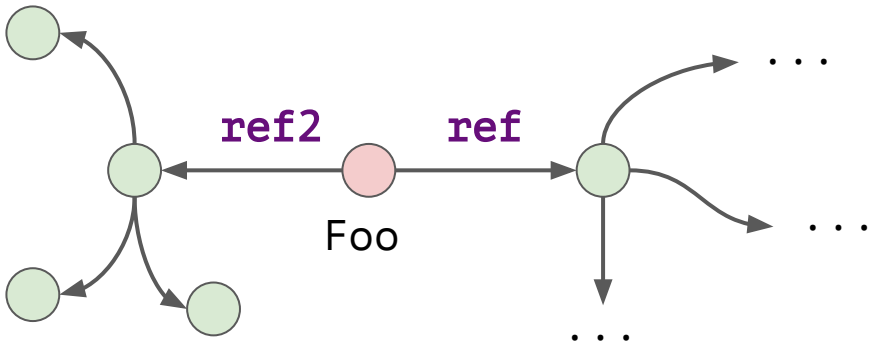
```
class Foo {
    Object ref;
    Object ref2;
    Foo(Object r) {
        ref = r;
    }
    void finalize() {
        ref.call();
    }
}
```



EXCELSIOR

JVM vs F-reachables

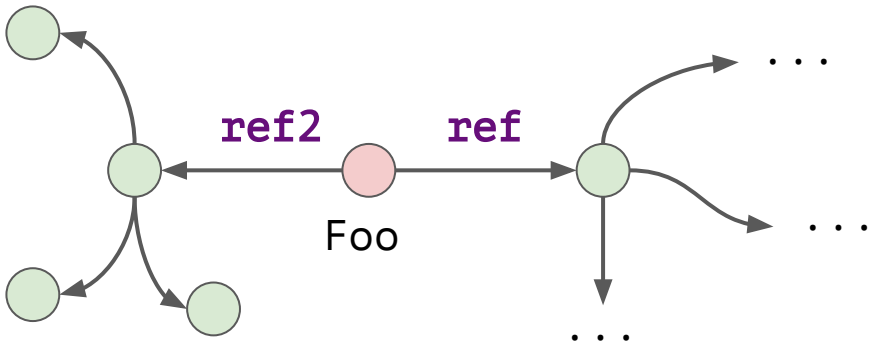
```
class Foo {
    Object ref;
    Object ref2;
    Foo(Object r) {
        ref = r;
    }
    void finalize() {
        ref.call();
    }
}
```



EXCELSIOR

JVM vs F-reachables

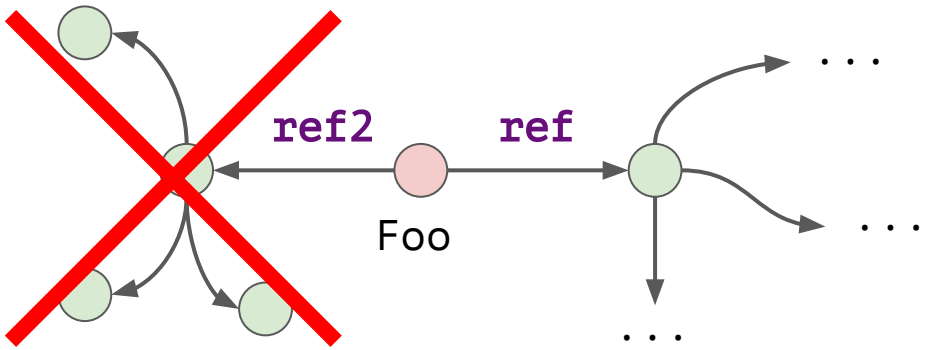
```
class Foo {
    Object ref;
    Object ref2;
    Foo(Object r) {
        ref = r;
    }
    void finalize() {
        ref.call();
    }
}
```



но **ref2** не используется в finalize!

JVM vs F-reachables

```
class Foo {
    Object ref;
    Object ref2;
    Foo(Object r) {
        ref = r;
    }
    void finalize() {
        ref.call();
    }
}
```



но **ref2** не используется в finalize!

можно сократить f-reachables (так делается в Excelsior JET)

Developer vs F-reachables

Как обнаружить проблему с F-reachables?

EXCELSIOR

Developer vs F-reachables

Как обнаружить проблему с F-reachables?

```
jmap -finalizerinfo <PID>
```

EXCELSIOR

Developer vs F-reachables

Как обнаружить проблему с F-reachables?

jmap -finalizerinfo <PID>

```
Unreachable instances waiting for finalization
#instances  class name
-----------------------
    147532  StressTimer$FinalizeBallast
       222  java.util.Timer$1
       221  StressTimer$Task
```

EXCELSIOR

Developer vs F-reachables

Как обнаружить проблему с F-reachables?

```
jmap -finalizerinfo <PID>
```

HotSpot-specific

EXCELSIOR

Developer vs F-reachables

Как обнаружить проблему с F-reachables?

jmap -finalizerinfo <PID>

HotSpot-specific

-Djet.gc.finalizer.stat

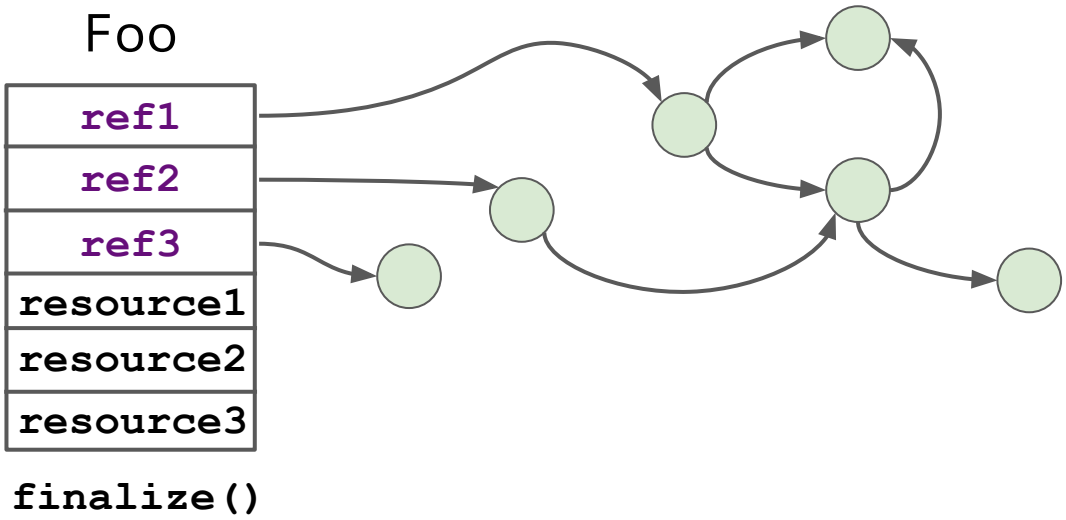
EXCELSIOR

Developer vs F-reachables

Как Вы можете бороться с F-reachables?

Developer vs F-reachables

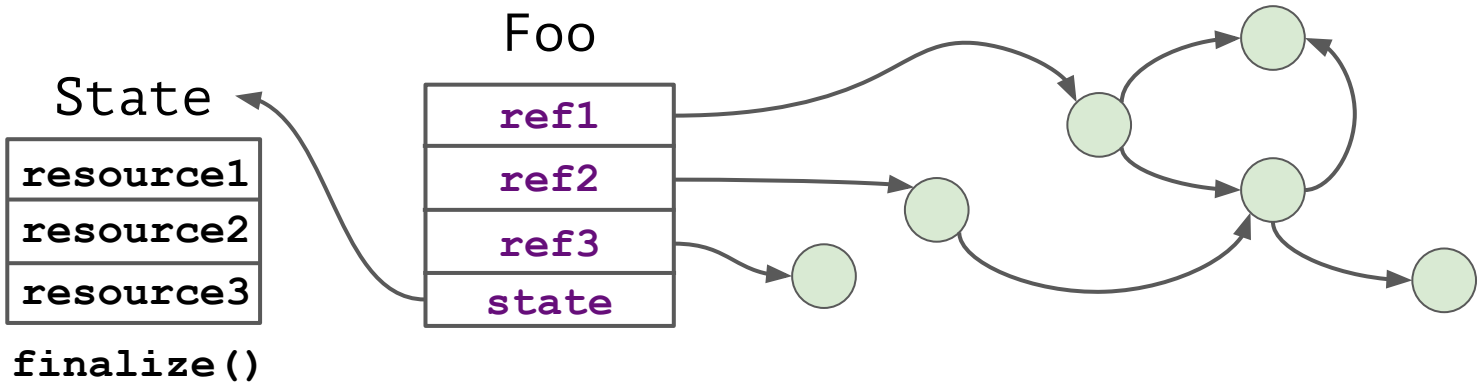
Как Вы можете бороться с F-reachables?



EXCELSIOR

Developer vs F-reachables

Как Вы можете бороться с F-reachables?



EXCELSIOR

Developer vs F-reachables

Как Вы можете бороться с F-reachables?

EXCELSIOR

Developer vs F-reachables

Если не finalize(), то кто?

Developer vs F-reachables

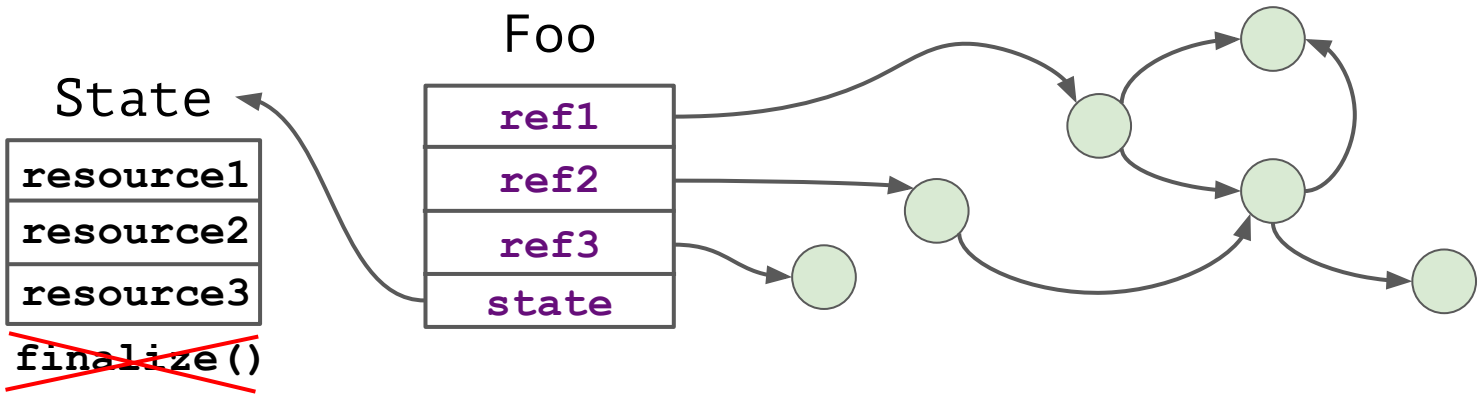
Если не finalize(), то кто?

java.lang.ref.Cleaner

(ранее sun.misc.Cleaner)

EXCELSIOR

Developer vs F-reachables




```
static class State implements Runnable { ... }
static final Cleaner cleaner = Cleaner.create();
...
var foo = new Foo();
cleaner.register(foo, foo.getState());
```

EXCELSIOR

Developer vs F-reachables

Если не finalize(), то кто?

java.lang.ref.Cleaner

(ранее sun.misc.Cleaner)

Поможет с Memory Drag только в Java 9+!

JDK-8071507

EXCELSIOR

Выводы

- finalize() увеличивает Memory Drag
- JVM могут облегчить симптомы (но делают это редко)
- java.lang.ref.Cleaner since Java 9

EXCELSIOR

История #3. Непотизм GC

Непотизм (кумовство) – вид фаворитизма, предоставляющий привилегии родственникам или друзьям независимо от их профессиональных качеств.

EXCELSIOR

Пример

```
public class DummyList<E> {

    private Node<E> first;
    private Node<E> last;

    private class Node<E> {
        E value;
        Node<E> next;
        public Node(E value, Node<E> next) { ... }
    }
}
```

EXCELSIOR

Пример

```
public void addLast(Node n) {
    final var l = last;
    last = n;
    if (l == null) {
        first = n;
    } else {
        l.next = n;
    }
}
```

EXCELSIOR

Пример

```java
public void addLast(Node n) {
    final var l = last;
    last = n;
    if (l == null) {
        first = n;
    } else {
        l.next = n;
    }
}
```

```java
public void removeFirst() {
    if (first != null) {
        var next = first.next;
        first = next;
        if (next == null) {
            last = null;
        }
    }
}
```

EXCELSIOR

Пример

```
public void addLast
    final var l = l
    last = n;
    if (l == null)
        first = n;
    } else {
        l.next = n;
    }
}
```


```
removeFirst() {
t != null) {
next = first.next;
t = next;
next == null) {
last = null;
```

EXCELSIOR

История #3. Непотизм

Сборка мусора по поколениям:

- Молодое/старое поколения
- Minor/Major GC

EXCELSIOR

История #3. Непотизм

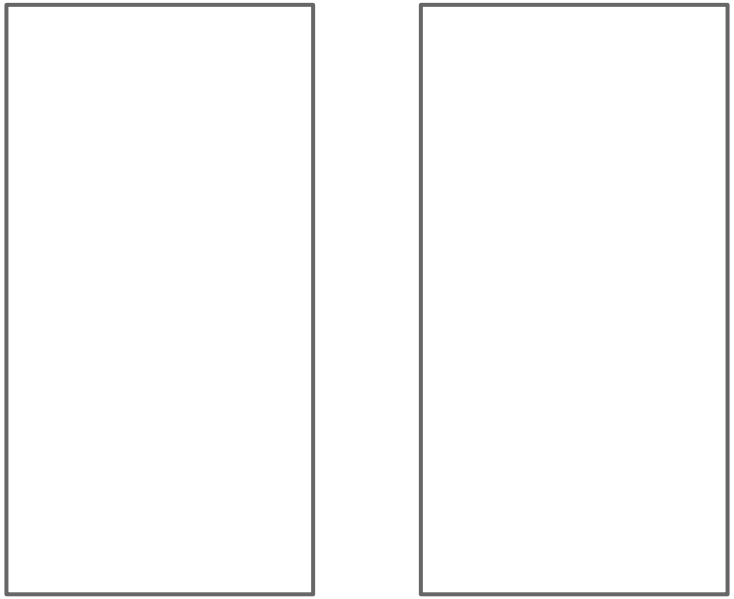
Сборка мусора по поколениям:

- Молодое/старое поколения
- Minor/Major GC
- Все достижимое из старого поколения должно переживать Minor GC

EXCELSIOR

История #3. Непотизм

Old Generation

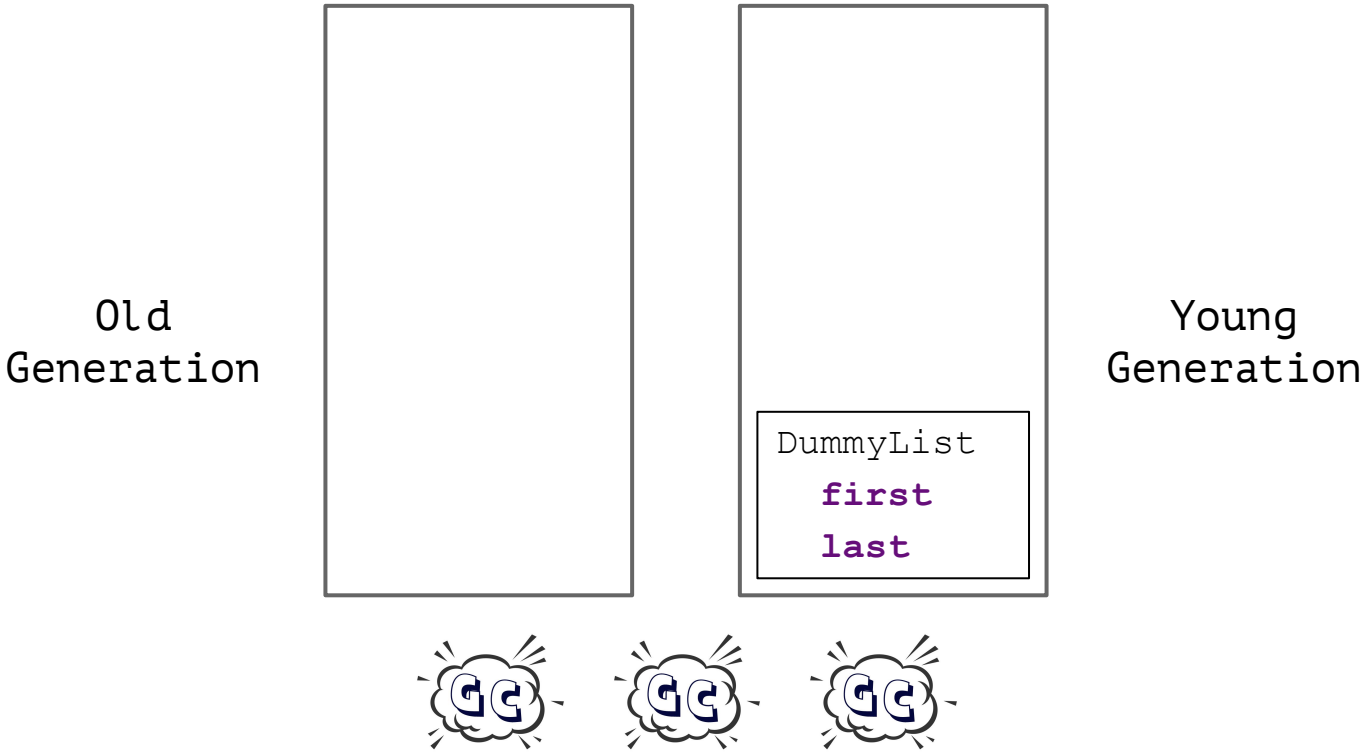


Young Generation

История #3. Непотизм


```
new DummyList()
```

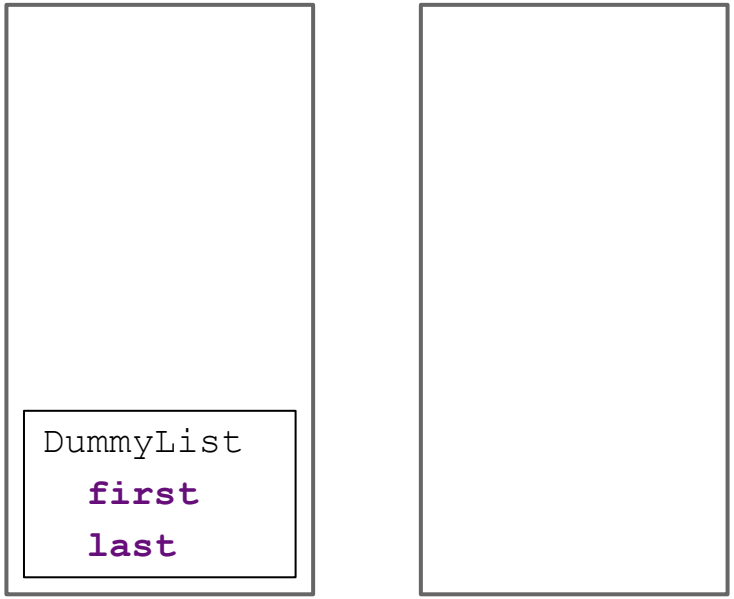
История #3. Непотизм



EXCELSIOR

История #3. Непотизм

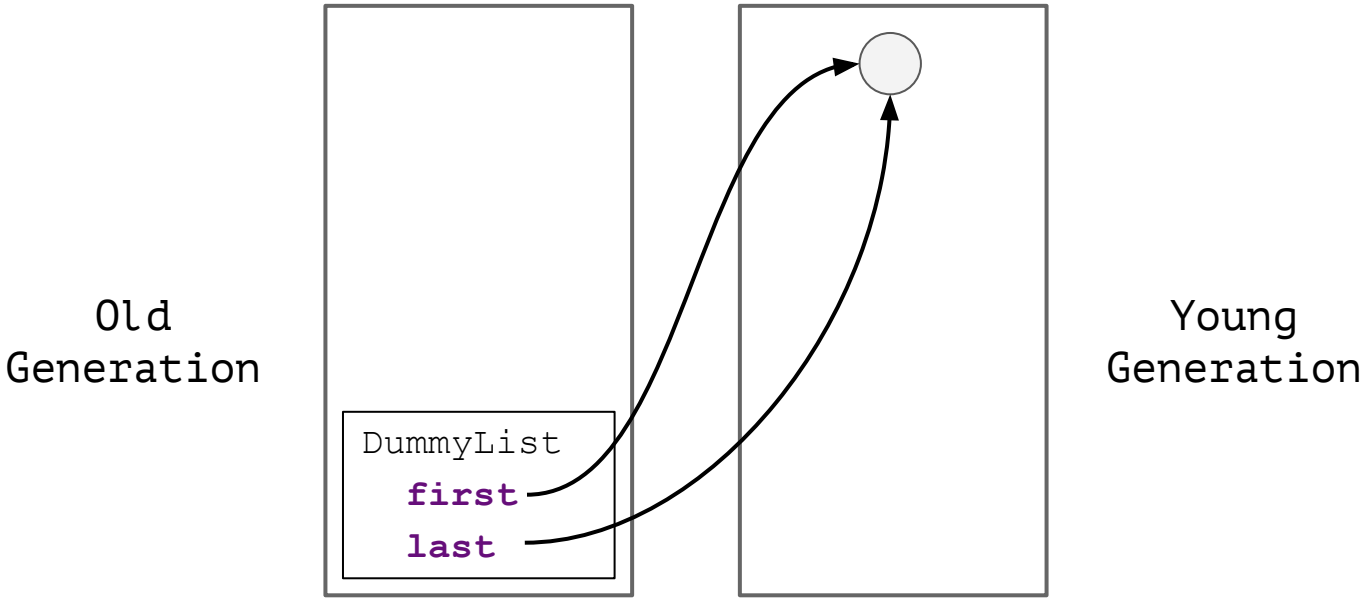
Old Generation



Young Generation

EXCELSIOR

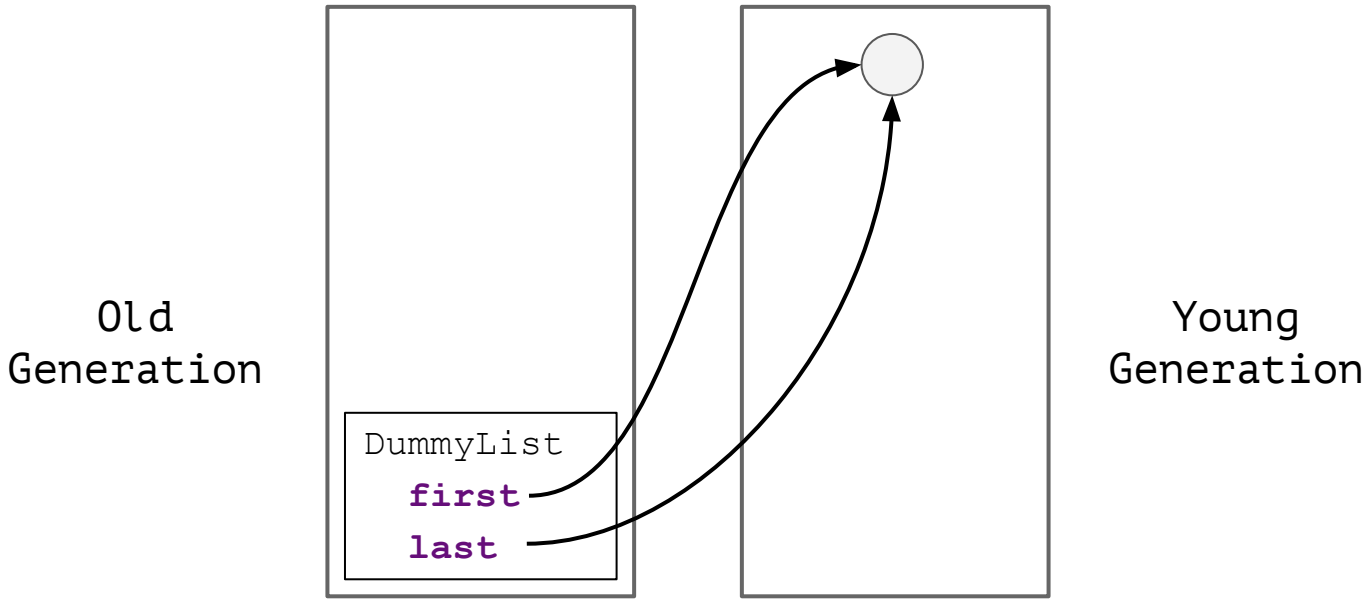
История #3. Непотизм



DummyList.addLast(...)

EXCELSIOR

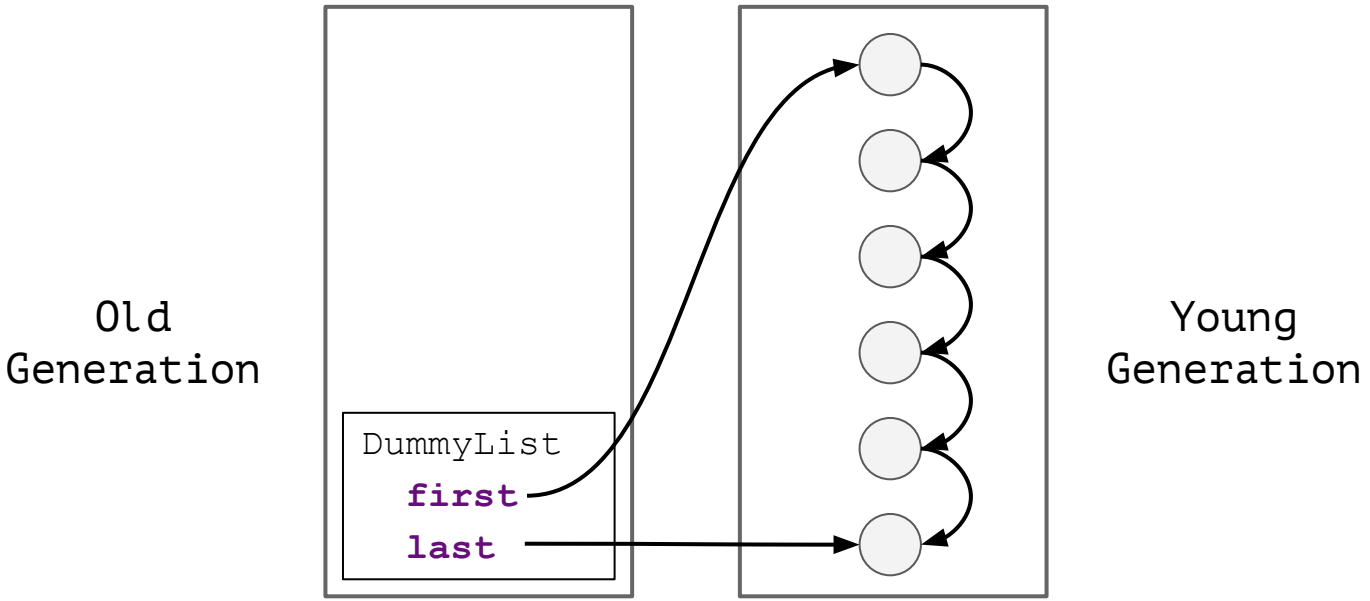
История #3. Непотизм



DummyList.addLast(...)

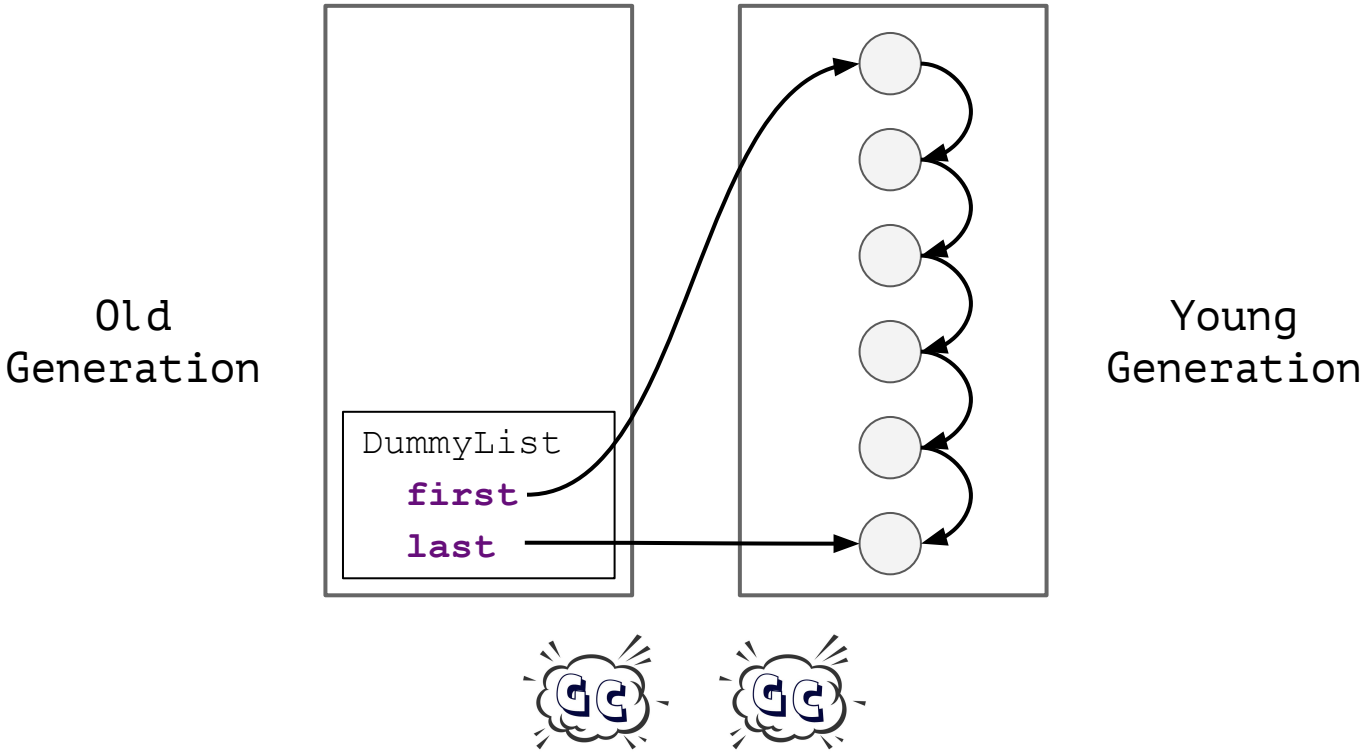
EXCELSIOR

История #3. Непотизм



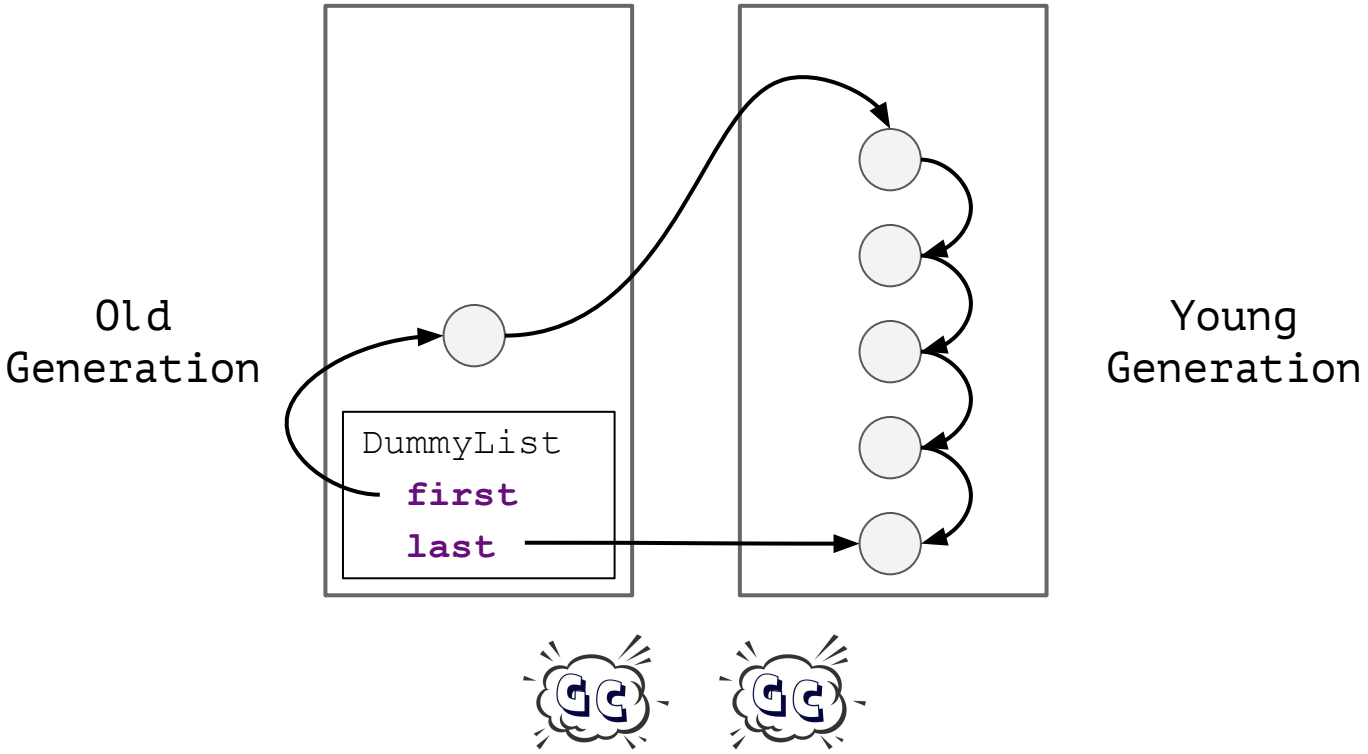
DummyList.addLast(...)

История #3. Непотизм



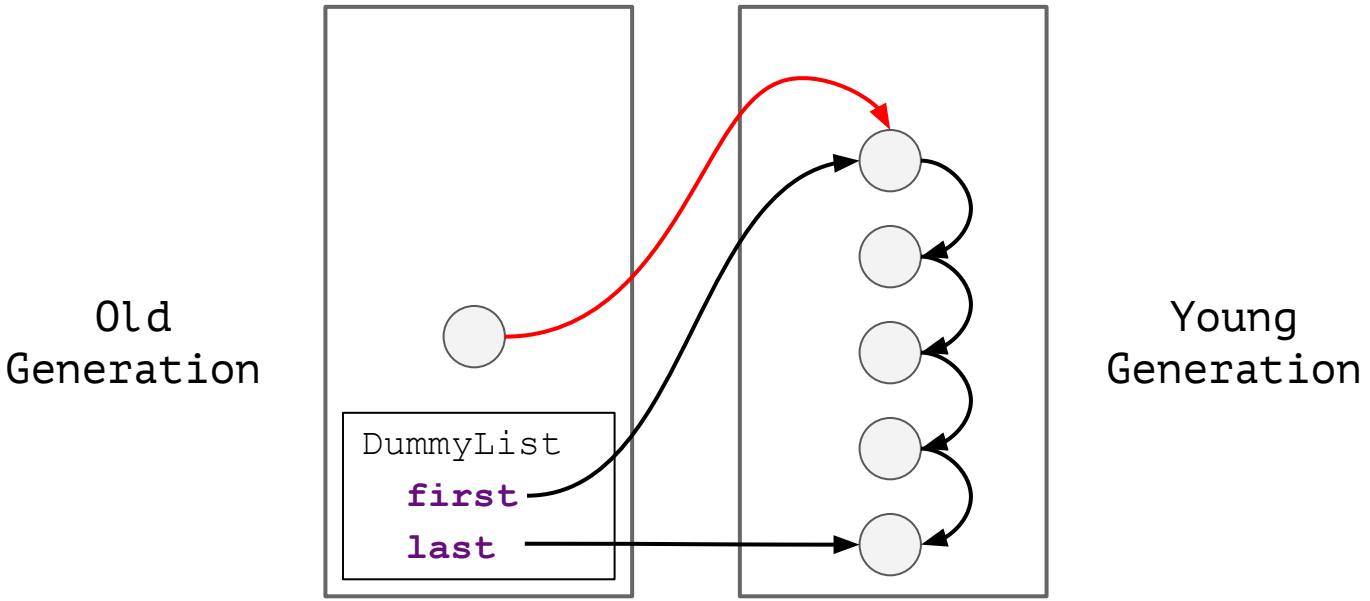
EXCELSIOR

История #3. Непотизм



EXCELSIOR

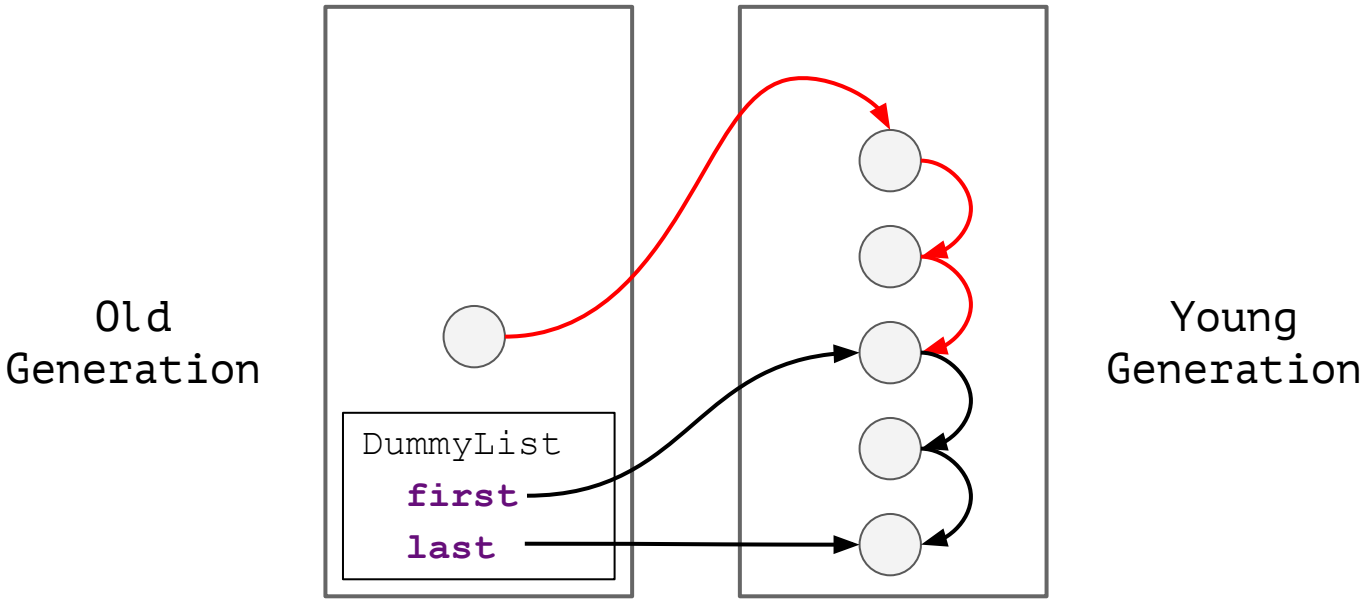
История #3. Непотизм




```
DummyList.removeFirst(...)
```

EXCELSIOR

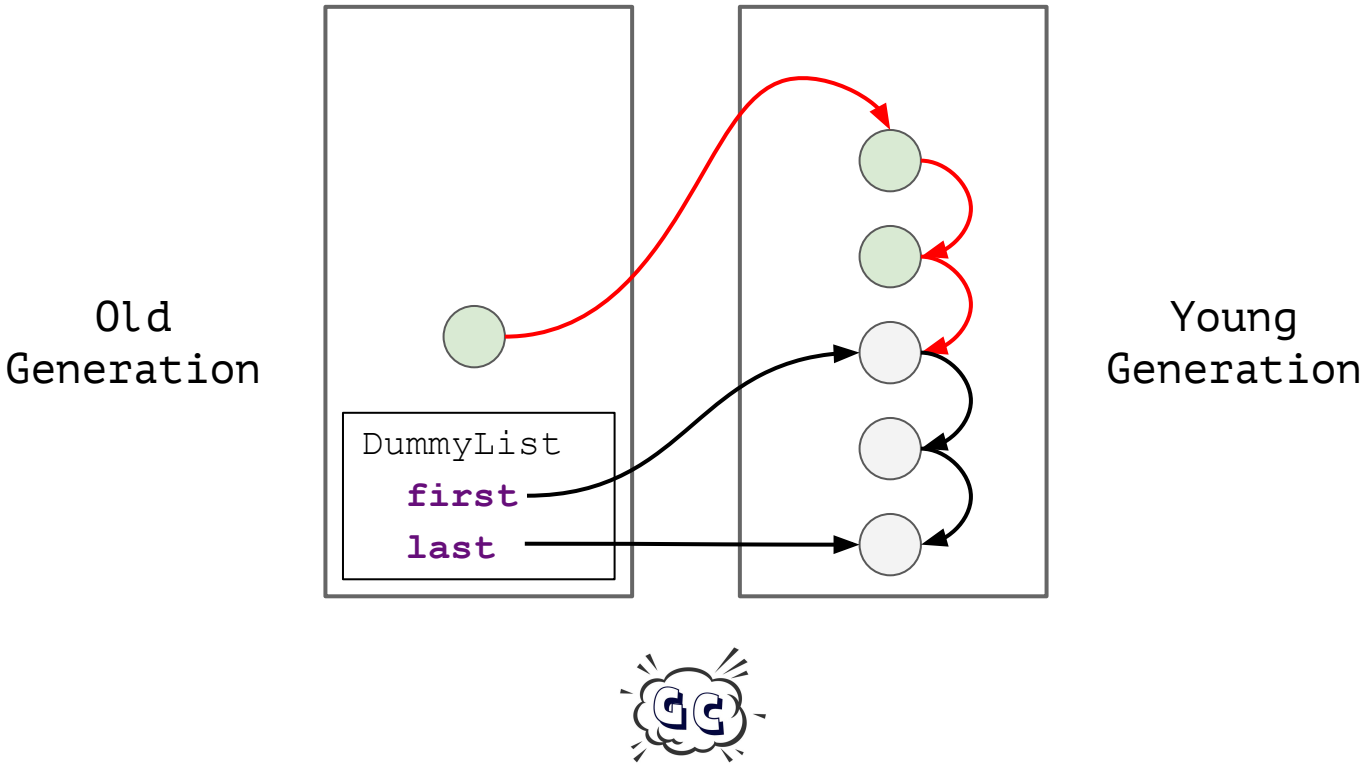
История #3. Непотизм




```
DummyList.removeFirst(...)
```

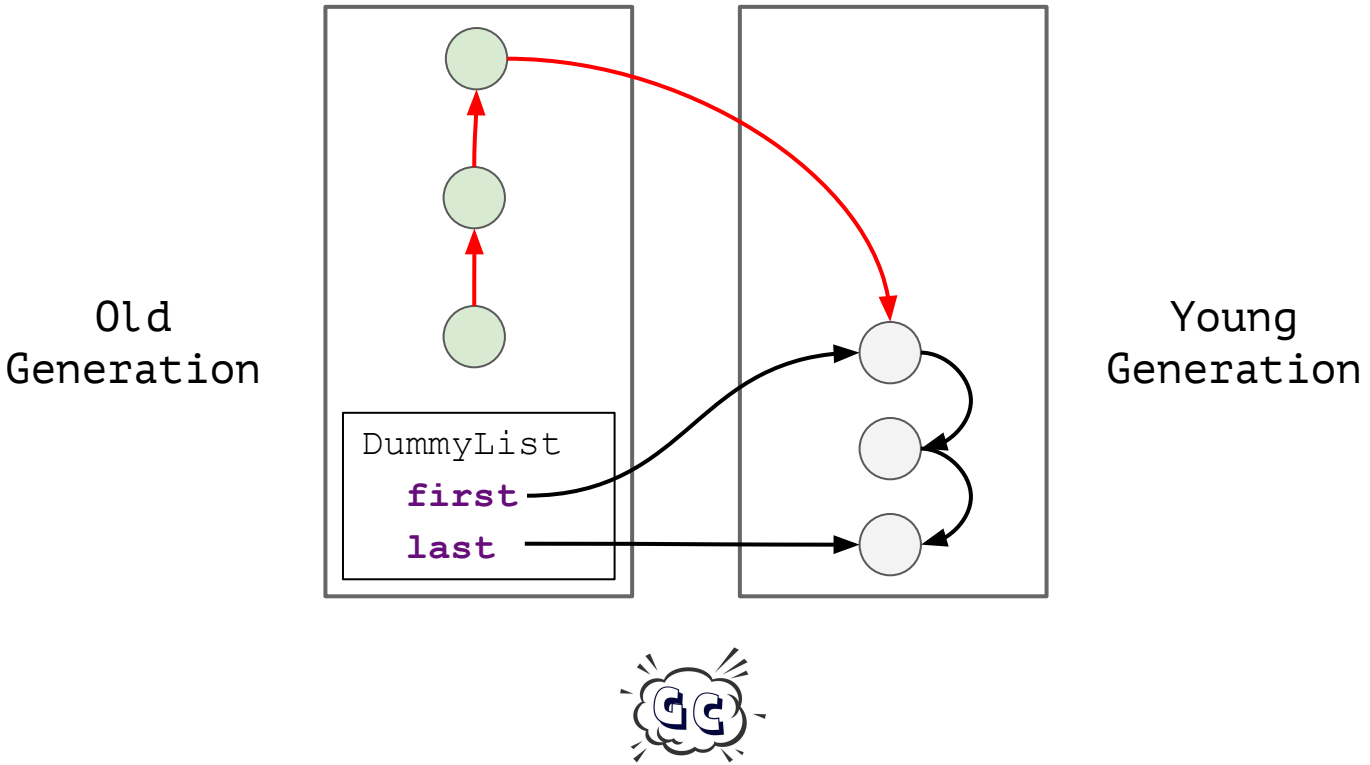
EXCELSIOR

История #3. Непотизм



EXCELSIOR

История #3. Непотизм



EXCELSIOR

История #3. Непотизм

Сборка мусора по поколениям сознательно увеличивает Memory Drag.

EXCELSIOR

История #3. Непотизм

Сборка мусора по поколениям сознательно увеличивает Memory Drag.

Старое поколение - источник ходячих мертвецов

EXCELSIOR

История #3. Непотизм

Но ведь это искусственный пример?

История #3. Непотизм

Но ведь это искусственный пример?

Tony Printezis о борьбе с непотизмом LinkedHashMap в TwitterJDK:

https://youtu.be/M9o1LVfGp2A?t=1391

EXCELSIOR

Как обнаружить непотизм?

Как обнаружить непотизм?

По косвенным признакам:

1. Частые и долгие Full GC (проверить по -XX:+PrintGCDetails)

2. Много подозрительных живых объектов (смотреть jmap -dump <PID>)

EXCELSIOR

Что делать?

Что делать?

- Ждать Full GC

EXCELSIOR

Что делать?

- Ждать Full GC
- Занулять ссылки из объектов при удалении

EXCELSIOR

Что делать?

java.util.LinkedList:

```
/**
 * Unlinks non-null first node f.
 */
private E unlinkFirst(Node<E> f) {
    final E element = f.item;
    final Node<E> next = f.next;
    f.item = null;
    f.next = null; // help GC
    ...
}
```

EXCELSIOR

Что делать?

java.util.LinkedList:

```
/**
 * Unlinks non-null first node f.
 */
private E unlinkFirst(Node<E> f) {
    final E element = f.item;
    final Node<E> next = f.next;
    f.item = null;
    f.next = null; // help GC
    ...
}
```

EXCELSIOR

Что делать?

java.util.LinkedHashMap:

```
void afterNodeRemoval(Node<K,V> e) {
    // unlink
    LinkedHashMap.Entry<K,V> p =
        (LinkedHashMap.Entry<K,V>)e;
    b = p.before, a = p.after;
    p.before = p.after = null;
    ...
}
```

EXCELSIOR

Что делать?

- Ждать Full GC
- Занулять ссылки из объектов при удалении

Что делать?

- Ждать Full GC
- Занулять ссылки из объектов при удалении
- Увеличить размер молодого поколения

  -Xmn

  -XX:NewRatio

  -XX:NewSize

EXCELSIOR

Ходячие объекты-мертвецы

А может ли GC вообще не прийти за мертвым объектом?

EXCELSIOR

Ходячие объекты-мертвецы

EXCELSIOR

История #4. Консерватизм

Точный GC

```
public static void main(String[] args) {
    final int size = 512*1024*1024 / (Integer.SIZE / 8);
    int[] arr1 = new int[size];
    System.out.println("arr1 (" + arr1 + ") allocated");
    System.gc();
    int[] arr2 = new int[size];
    System.out.println("arr2 (" + arr2 + ") allocated");
}
```

EXCELSIOR

Точный GC

```
public static void main(String[] args) {
    final int size = 512*1024*1024 / (Integer.SIZE / 8);
    int[] arr1 = new int[size];
    System.out.println("arr1 (" + arr1 + ") allocated");
    System.gc();
    int[] arr2 = new int[size];
    System.out.println("arr2 (" + arr2 + ") allocated");
}
```

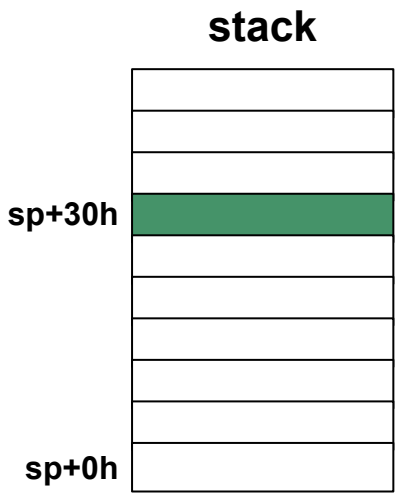
Как GC понимает, какие локалы живы?

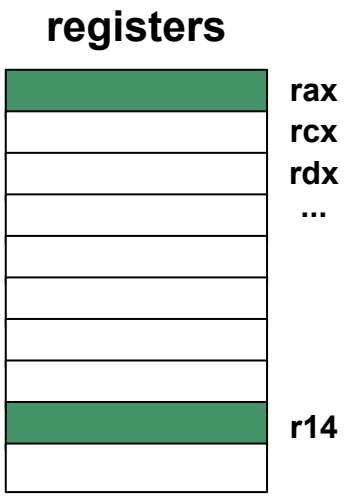
EXCELSIOR

Точный GC

EXCELSIOR

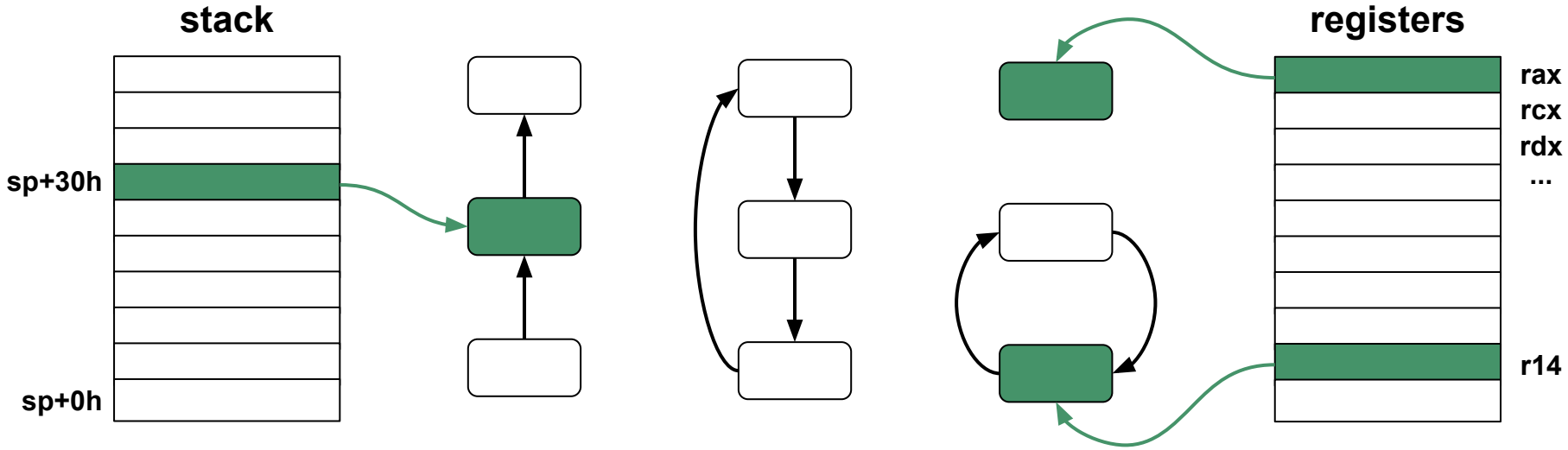
Точный GC





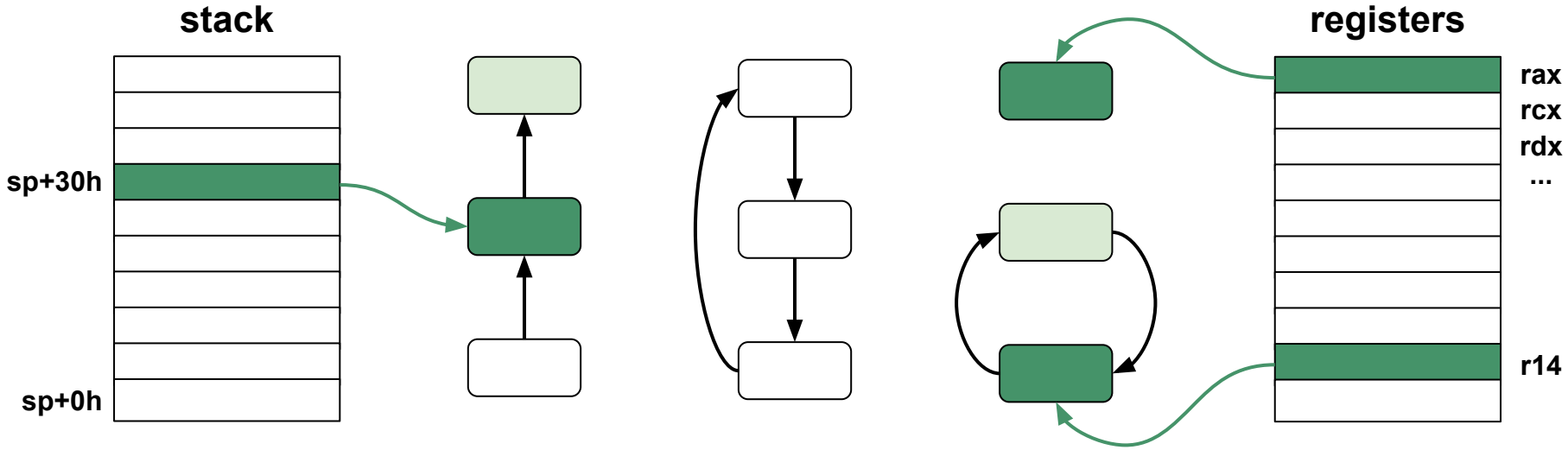
EXCELSIOR

Точный GC



EXCELSIOR

Точный GC



Точный GC

- Поддержка от компилятора

Точный GC

- Поддержка от компилятора
- Генерация карт стека и регистров

EXCELSIOR

Точный GC

- Поддержка от компилятора
- Генерация карт стека и регистров
- Карты только в safe-points!

EXCELSIOR

Консервативный GC

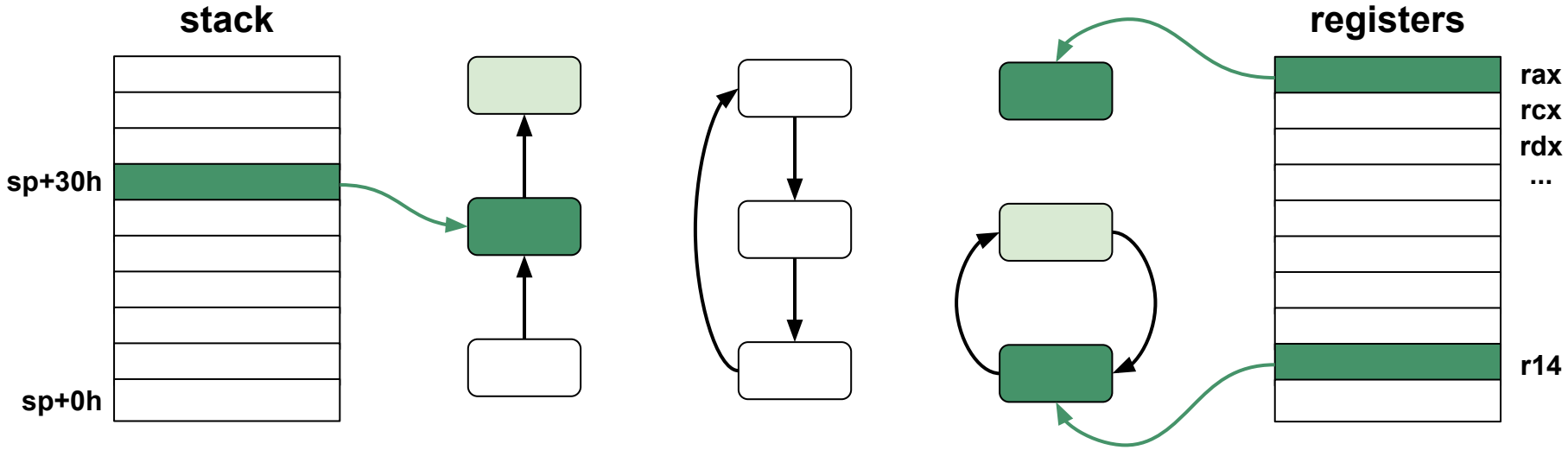
А нельзя ли без поддержки от компилятора?

Консервативный GC

- Предполагаем худшее: каждое значение может быть указателем
- Отсеиваем лишнее в рантайме

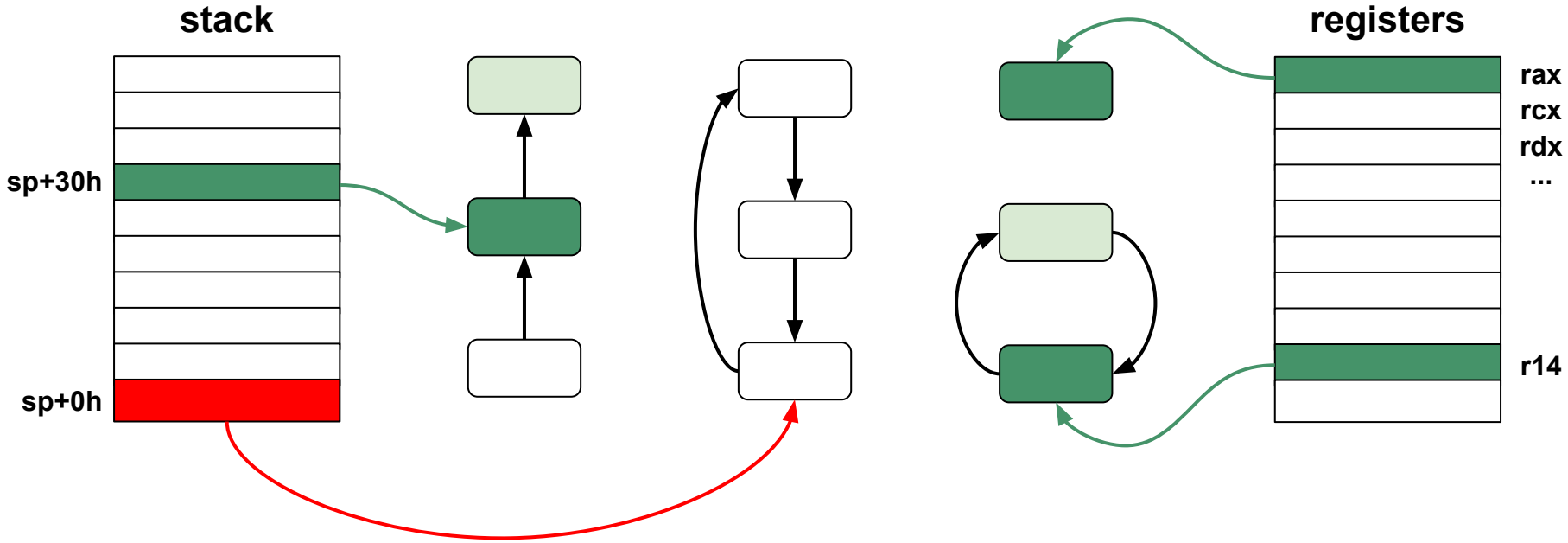
EXCELSIOR

Консервативный GC



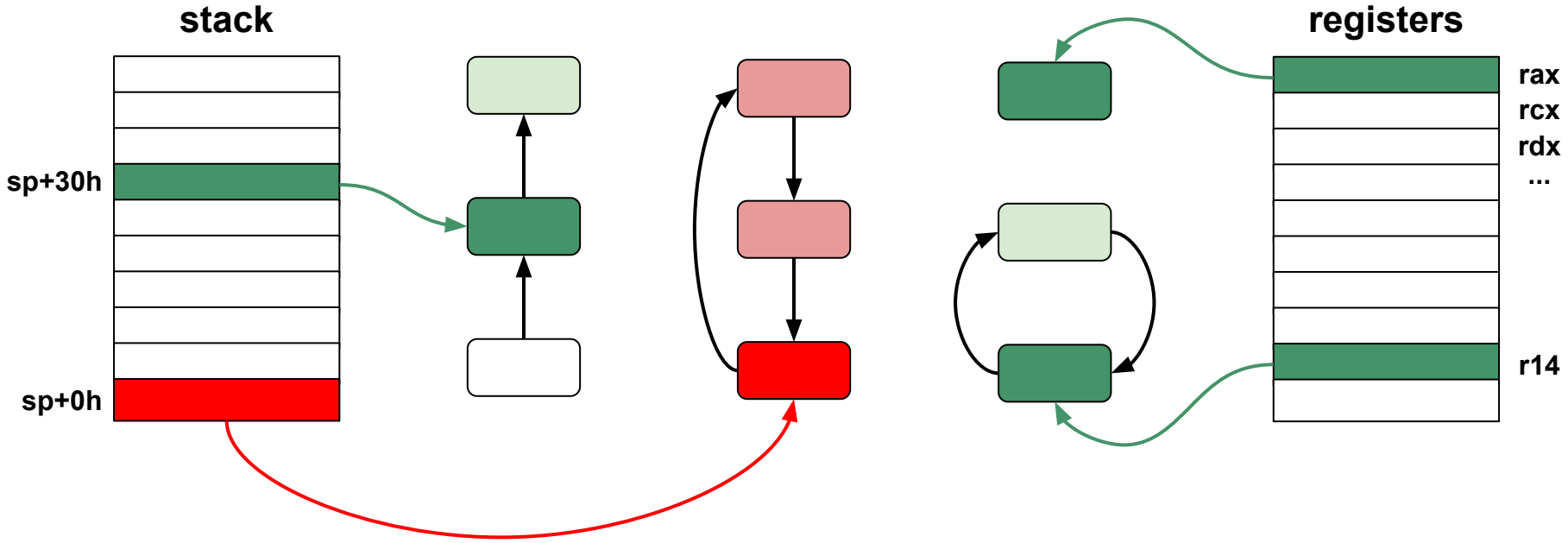
EXCELSIOR

Консервативный GC



EXCELSIOR

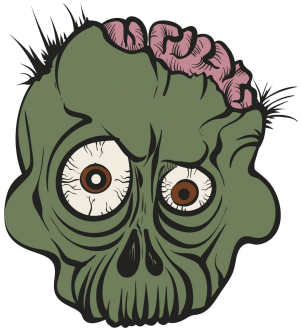
Консервативный GC



EXCELSIOR

Консервативный GC

- Предполагаем худшее: каждое значение может быть указателем
- Отсеиваем лишнее в рантайме
- Иногда ошибаемся ⇒ увеличиваем Memory Drag на величину ошибки



EXCELSIOR

Консервативный GC

Зачем такое нужно?

Консервативный GC

Зачем такое нужно?

- Нет карт ⇒ они не занимают место
- Нет карт ⇒ нет затрат на генерацию

EXCELSIOR

Консервативный GC

Зачем такое нужно?

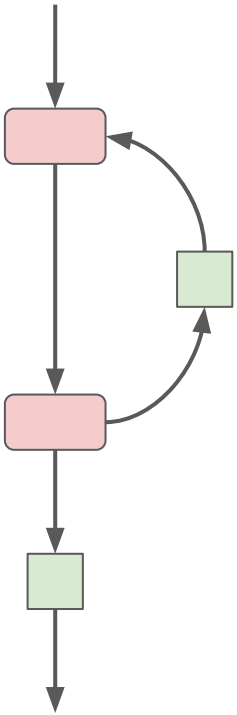
- Нет карт ⇒ они не занимают место
- Нет карт ⇒ нет затрат на генерацию
- Нет карт ⇒ можно выкинуть safe-points

EXCELSIOR

Safe-points

1 - 2 машинные инструкции

Обратные дуги циклов и эпилоги



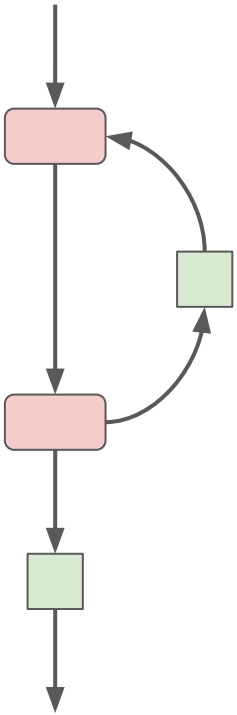
EXCELSIOR

Safe-points

1 - 2 машинные инструкции

Обратные дуги циклов и эпилоги

Ухудшают производительность ⇒
Агрессивно оптимизируются



EXCELSIOR

Мир без safe-points

EXCELSIOR

Мир без safe-points

- ✓ Производительность растет
- ✓ Размер кода уменьшается

EXCELSIOR

Мир без safe-points

✓ Производительность растет

✓ Размер кода уменьшается

? Многое работает через safe-points, поэтому потребует переработки

EXCELSIOR

Консервативный GC

А так вообще делают?

Консервативный GC

А так вообще делают?

Теория:

Boehm GC, 1988, conservative

Immix, 2008, mostly-precise

EXCELSIOR

Консервативный GC

А так вообще делают?

Теория:

Boehm GC, 1988, conservative

Immix, 2008, mostly-precise

Практика:

Консервативный GC

А так вообще делают?

Теория:

Boehm GC, 1988, conservative

Immix, 2008, mostly-precise

Практика:

MobiVM

Консервативный GC

А так вообще делают?

Теория:

Boehm GC, 1988, conservative

Immix, 2008, mostly-precise

Практика:

MobiVM

scala-native

EXCELSIOR

Консервативный GC

А так вообще делают?

Теория:

Boehm GC, 1988, conservative

Immix, 2008, mostly-precise

Практика:


MobiVM


scala-native


Excelsior JET
(с 1999 по 2015)

EXCELSIOR

Неограниченный Memory Drag

Консервативная ошибка:

- обычно продлевает жизнь объекта на один или несколько циклов GC (пока ложный корень на стеке)

EXCELSIOR

Неограниченный Memory Drag

Консервативная ошибка:

- обычно продлевает жизнь объекта на один или несколько циклов GC (пока ложный корень на стеке)
- иногда продлевает навсегда

EXCELSIOR

Неограниченный Memory Drag

```
public class Executor extends Thread {

    public abstract class Task implements Runnable { }

    final TaskQueue queue = new TaskQueue();

    . . .

}
```

EXCELSIOR

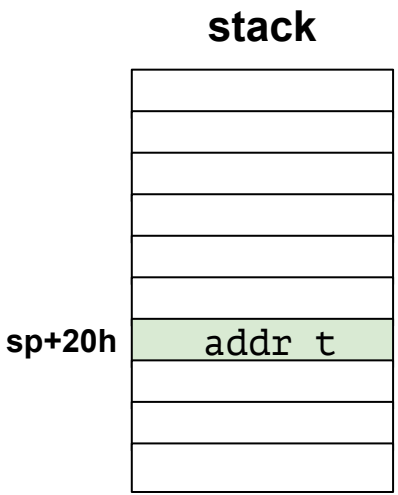
Неограниченный Memory Drag

```
public void mainLoop() {
    while (true) {
        synchronized (queue) {
            while (queue.isEmpty()) {
                queue.wait();
            }

            Task t = queue.get();
            t.run();
        }
    }
}
```

EXCELSIOR

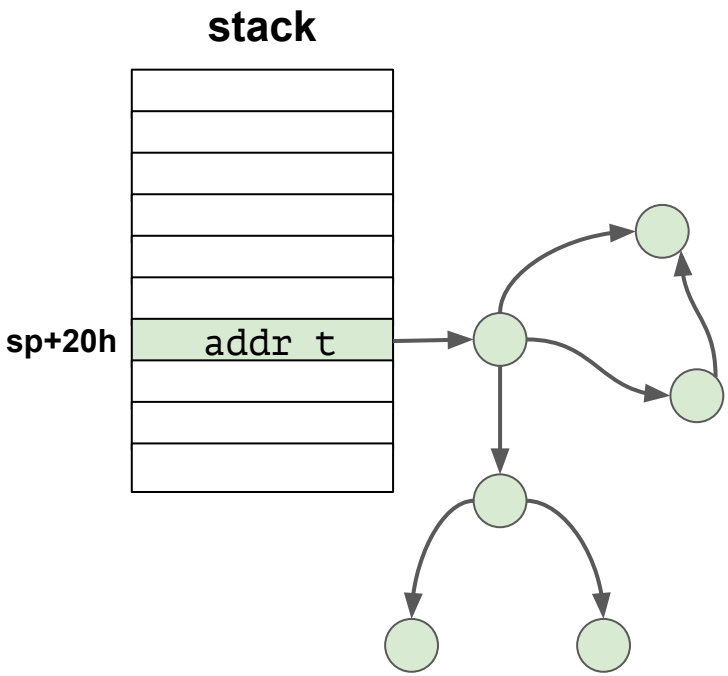
Неограниченный Memory Drag

```
public void mainLoop() {
    while (true) {
        synchronized (queue) {
            while (queue.isEmpty()) {
                queue.wait();
            }

            Task t = queue.get();
            t.run();
        }
    }
}
```



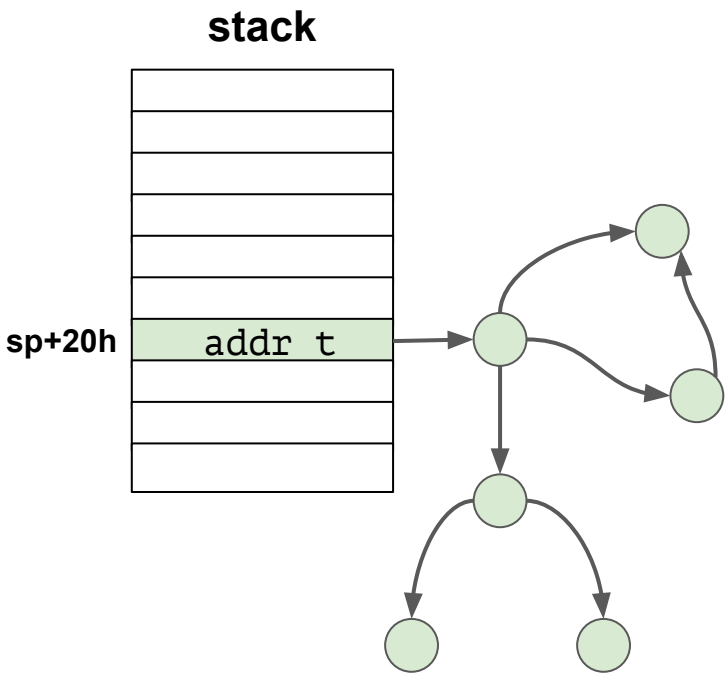
Неограниченный Memory Drag

```
public void mainLoop() {
    while (true) {
        synchronized (queue) {
            while (queue.isEmpty()) {
                queue.wait();
            }

            Task t = queue.get();
            t.run();
        }
    }
}
```



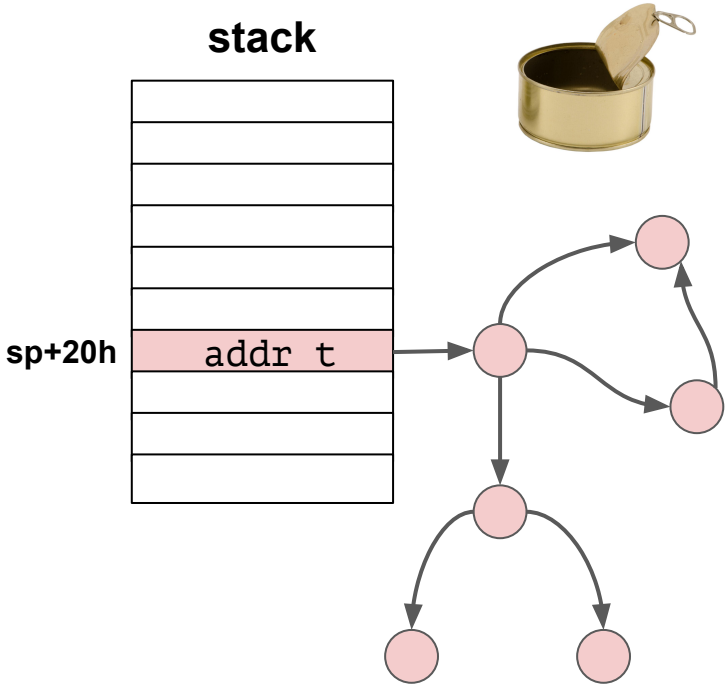
Неограниченный Memory Drag

```
public void mainLoop() {
    while (true) {
        synchronized (queue) {
            while (queue.isEmpty()) {
                queue.wait();
            }

            Task t = queue.get();
            t.run();
        }
    }
}
```



Неограниченный Memory Drag

```
public void mainLoop() {
    while (true) {
        synchronized (queue) {
            while (queue.isEmpty()) {
                queue.wait();
            }

            Task t = queue.get();
            t.run();
        }
    }
}
```



Неограниченный Memory Drag

Может это искусственный пример?

EXCELSIOR

Неограниченный Memory Drag

Может это искусственный пример?

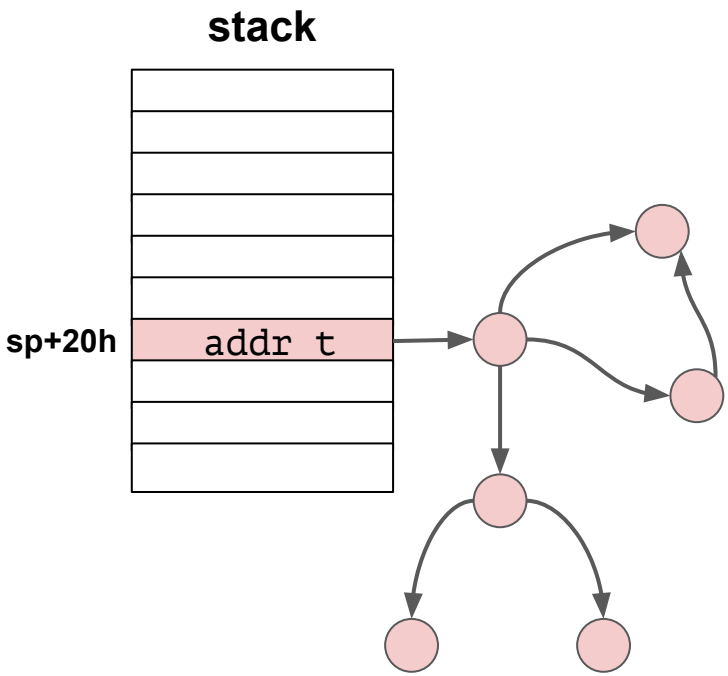
Нет, это класс `java.util.Timer` из JDK

И там все еще хуже

EXCELSIOR

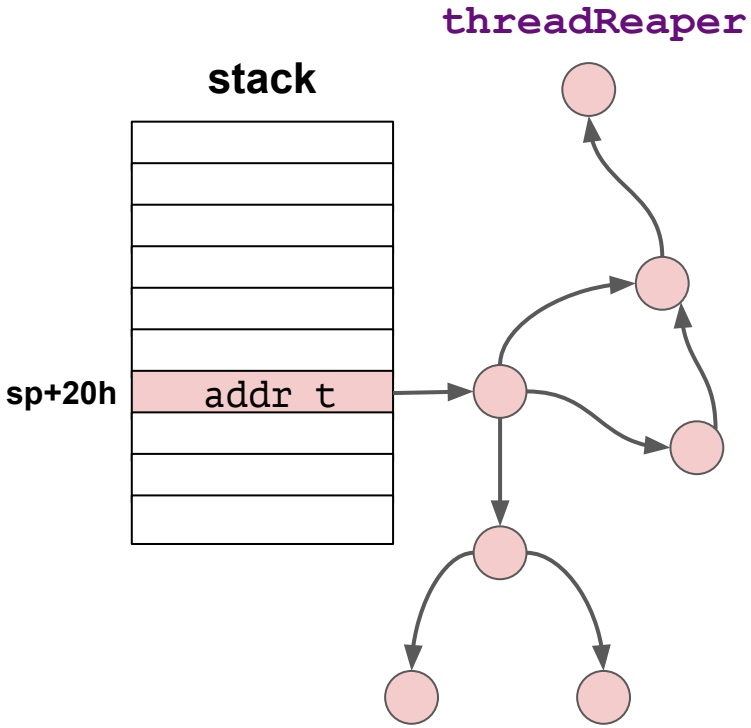
Неограниченный Memory Drag

```
public void mainLoop() {
    while (true) {
        synchronized (queue) {
            while (queue.isEmpty()) {
                queue.wait();
            }

            Task t = queue.get();
            t.run();
        }
    }
}
```



Неограниченный Memory Drag

```
public void mainLoop() {
    while (true) {
        synchronized (queue) {
            while (queue.isEmpty()) {
                queue.wait();
            }

            Task t = queue.get();
            t.run();
        }
    }
}
```



Неограниченный Memory Drag

Как Вы можете
бороться с консервой?

EXCELSIOR

Неограниченный Memory Drag

Как Вы можете
бороться с консервой?

EXCELSIOR

Неограниченный Memory Drag

Как Вы можете
бороться с консервой?

EXCELSIOR

Неограниченный Memory Drag

java.util.Timer javadoc:

After the last live reference to a Timer object goes away and all outstanding tasks have completed execution, the timer's task execution thread terminates gracefully (and becomes subject to garbage collection). However, this can take arbitrarily long to occur. By default, the task execution thread does not run as a daemon thread, so it is capable of keeping an application from terminating. If a caller wants to terminate a timer's task execution thread rapidly, the caller should invoke the timer's ***cancel*** *method.*

EXCELSIOR

Выводы

Консервативный GC - большая сила, но и большая ответственность

EXCELSIOR

Выводы

Консервативный GC - большая сила, но и большая ответственность

Консерва не только в GC (смотри первый пример)

EXCELSIOR

Заключение

Заключение

Не нужно пытаться предугадать время жизни объекта и рассчитывать на это в коде!

Заключение

Не нужно пытаться предугадать время жизни объекта и рассчитывать на это в коде!

Если вы все-таки на это решились:

Используйте правильные ~~костыли~~ решения (reachabilityFence, Cleaner, Timer.cancel)

EXCELSIOR

Q & A

iugliansky@excelsior-usa.com

@dbg_nsk

www.excelsiorjet.com/blog

EXCELSIOR