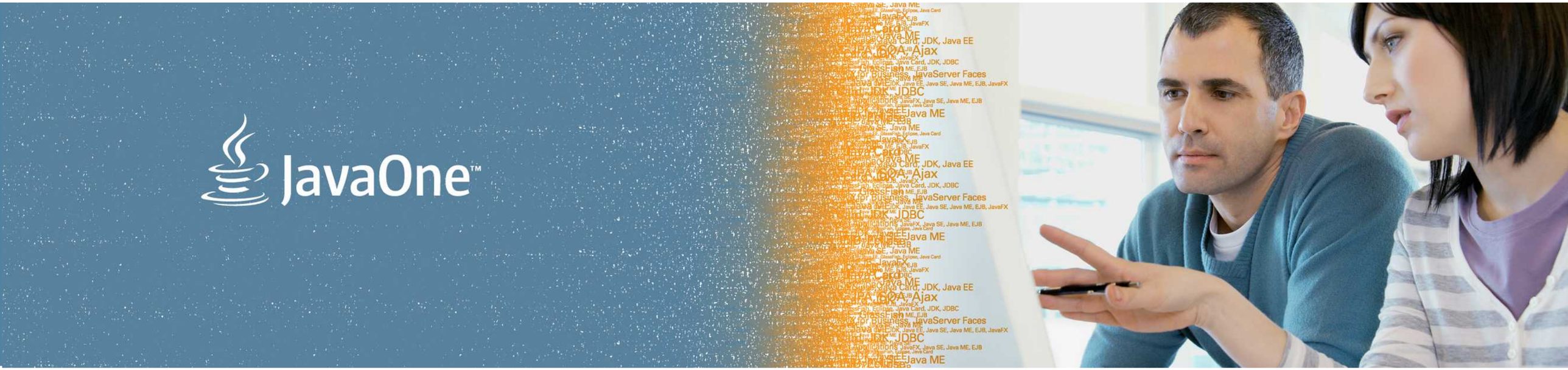

JavaOne™



ORACLE®

Java Platform Performance BoF

Sergey Kuksenko, Aleksey Shipilev

О чём у нас

- Отвечаем на вопросы по Java Performance
  - В основном, предварительно собранные в разных местах Рунета
  - Есть возможность задать вопрос прямо здесь
    - Пишем на бумажке, передаём вперёд ;)
- **Сначала вводные слова про:**
  - Performance Engineering
  - Startup
  - JIT
  - Concurrency and synchronization
- **Другие сессии:**
  - “Искусное тестирование производительности (Java)”, завтра, 11-45
  - “Диагностика и настройка GC”, сегодня, только что закончилось
  - “JDK7”, параллельно с нами

Performance Engineering

JavaOne™

ORACLE®

Performance Engineering

абстрактно и отлично об отличиях в абстракциях

- Computer Science → Software Engineering
  - Строим приложения по функциональным требованиям
  - В большой степени абстрактно, в “идеальном мире”
    - Теоретически неограниченная свобода – искусство!
    - Можно строить воздушные зАмки
  - Рассуждения при помощи формальных методов
- Software Performance Engineering
  - “Real world strikes back!”
  - Исследуем взаимодействие софта с железом на типичных данных
    - Производительность уже нельзя оценить
    - Производительность можно только *измерить*
  - Естественно-научные методы

Performance Engineering

первый шаг

- Классические ошибки первого шага
  - “я вижу, что метод *foo()* реализован неэффективно”
  - “по профилю видно, что метод *bar()* – самый горячий и занимает 5%”
  - “по-моему, у нас тормозит БД, и необходимо перейти с DB_X на DB_Y”
- Правильный первый шаг:
  - Необходимо выбрать метрику
    - ops/sec, transactions/sec
    - время исполнения
    - время отклика
  - Убедиться в корректности метрики
    - релевантна (учитывает реальный сценарий работы приложения)
    - повторяема

ЦЕЛЬ → улучшение метрики!

Performance Engineering

анализ узких мест (tips)

- **Низкая утилизация CPU**
  - Высокая дисковая, сетевая активность
  - Конфликт блокировок
  - Конфликт ресурсов ОС
  - Слабая параллелизация приложения
- **Высокая утилизация ядра ОС**
  - Частые блокировки
  - Частое обращение к ОС
- **Высокая утилизация CPU**
  - Неоптимальная архитектура приложения
  - Неправильное использование API
  - Неоптимизированные горячие методы
  - Неоптимальные настройки GC

Performance Engineering

инструменты для анализа системы

	Solaris	Linux	Windows	Что смотрим
Сеть	netstat, dtrace	netstat	perfmon	количество соединений, объем трафика
Диск	iostat, dtrace	iostat	perfmon	количество обращений к диску, задержка
Память	vmstat, prstat, dtrace	vmstat, top	perfmon	подкачка страниц, размер памяти
Процессы	ps, vmstat, mpstat, prstat, dtrace	ps, vmstat, top	perfmon	количество нитей, состояние нитей, переключения контекста
Ядро ОС	mpstat, lockstat, plockstat, dtrace, intrstat, vmstat	vmstat	perfmon	kernel time, блокировки, системные вызовы, прерывания ...

Performance Engineering

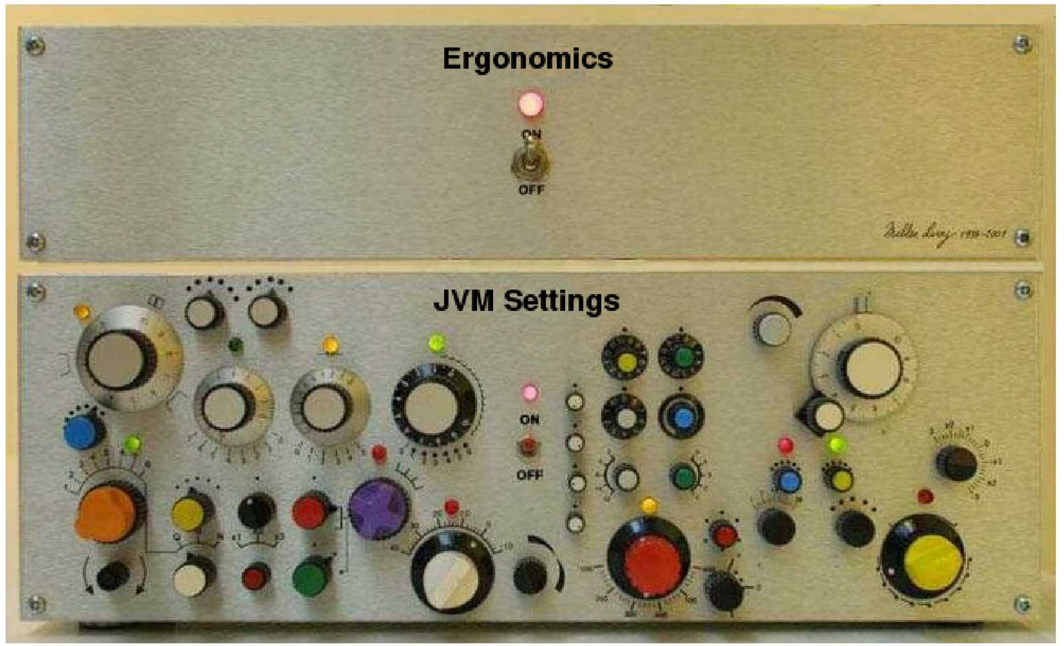
tools, tools, tools again, more tools

- VisualVM
  - http://visualvm.dev.java.net
- JRockit Mission Control
  - http://www.oracle.com/technetwork/middleware/jrockit/mission-control/index.html
- Sun Studio Analyzer
  - http://www.oracle.com/technetwork/server-storage/solarisstudio/overview/index.html
- DTrace
  - http://www.oracle.com/technetwork/systems/dtrace/dtrace/index.html
- Ещё могут быть полезны:
  - JProbe
  - OptimizeIt
  - YourKit

JVM tuning

настройка параметров JVM

- Что настраивать?
  - http://blogs.sun.com/watt/resource/jvm-options-list.html
  - http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html



JavaOne

ORACLE

JVM tuning

настройка параметров JVM

- **JVM сама подбирает оптимальные параметры своей работы**
  - Server vs. Client
  - Large pages (Solaris)
  - CompressedOops (64-bit VM)
- **Так что же настраивать?**
  - GC/Heap tuning
  - -XX:+UseNUMA (Solaris, Linux)
  - -XX+:UseLargePages (Linux, Windows)
    - http://www.oracle.com/technetwork/java/javase/tech/largememory-jsp-137182.html
- **Не забыть**
  - Использовать последнюю версию JDK

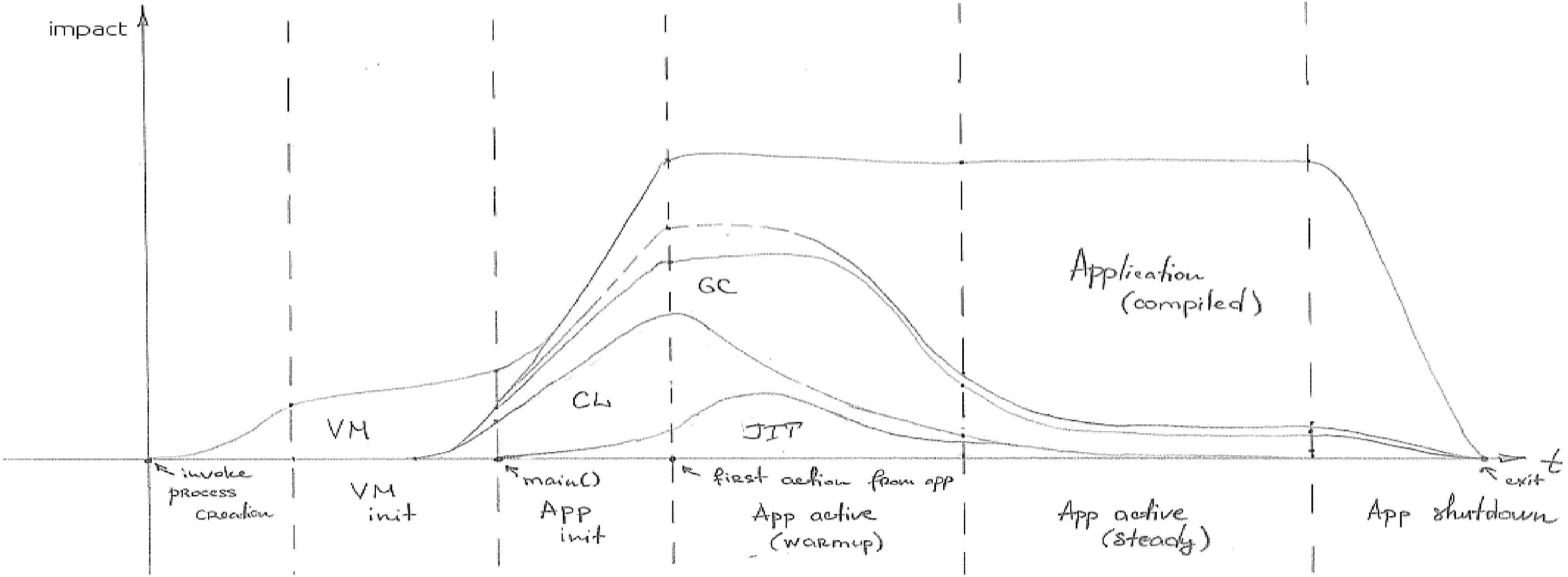
Startup

JavaOne™

ORACLE®

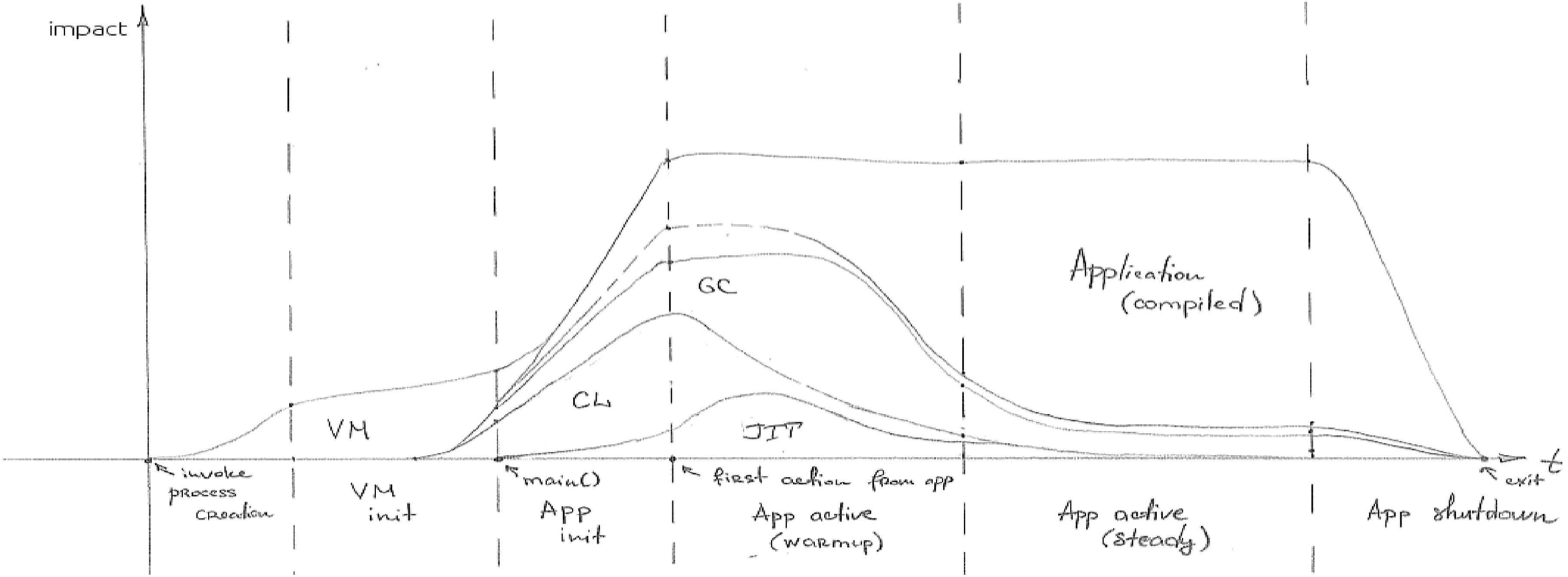
Startup

как измерять?



Startup

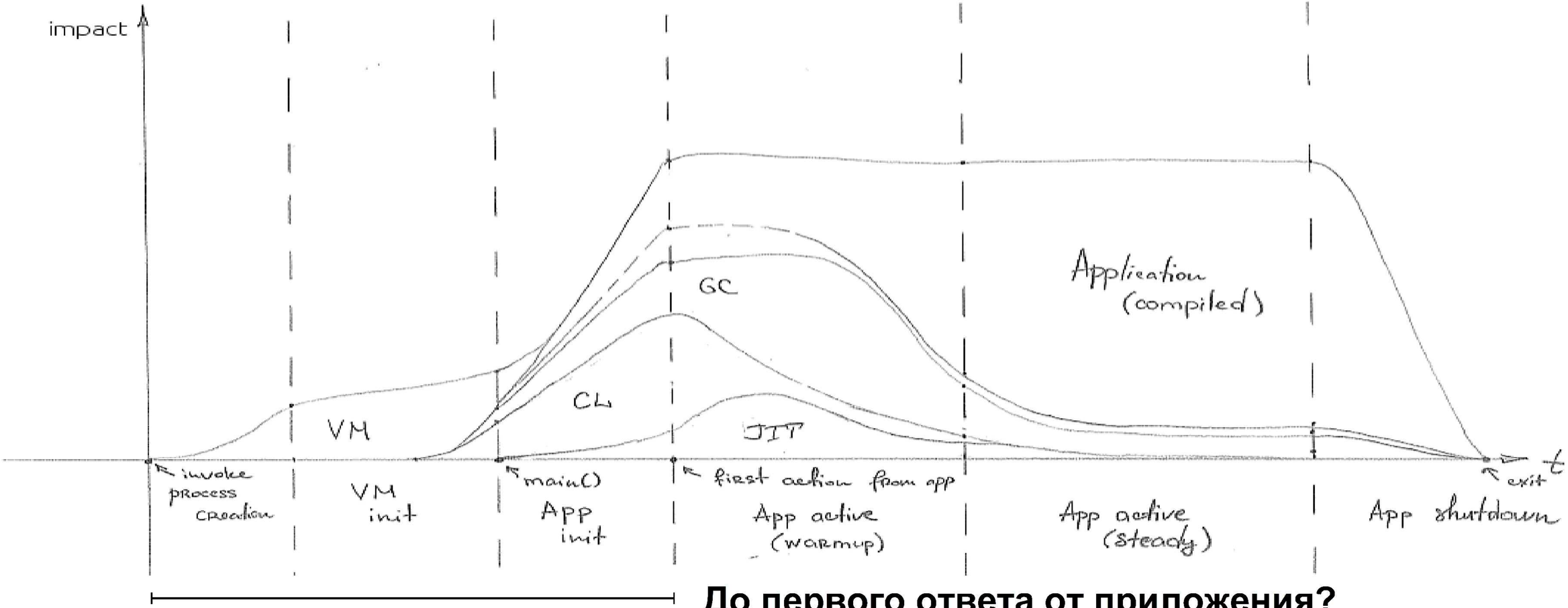
как измерять?



До входа в main()?

Startup

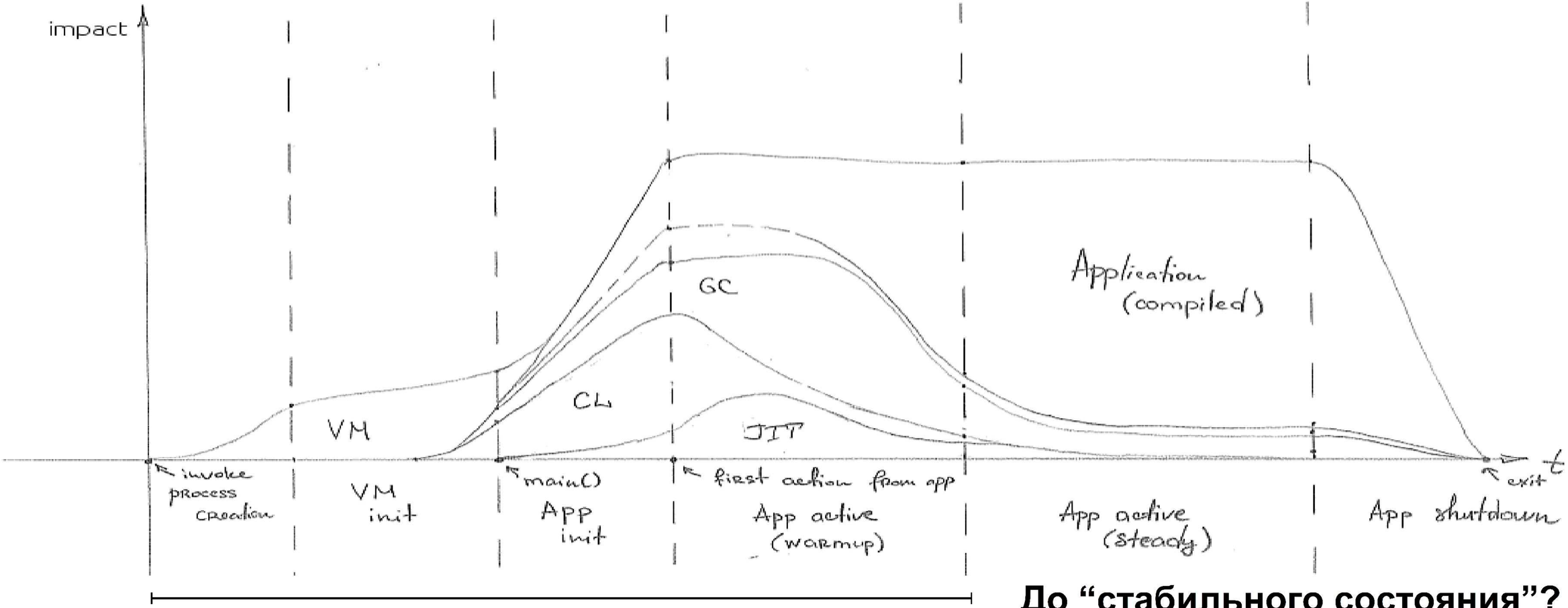
как измерять?



До первого ответа от приложения?

Startup

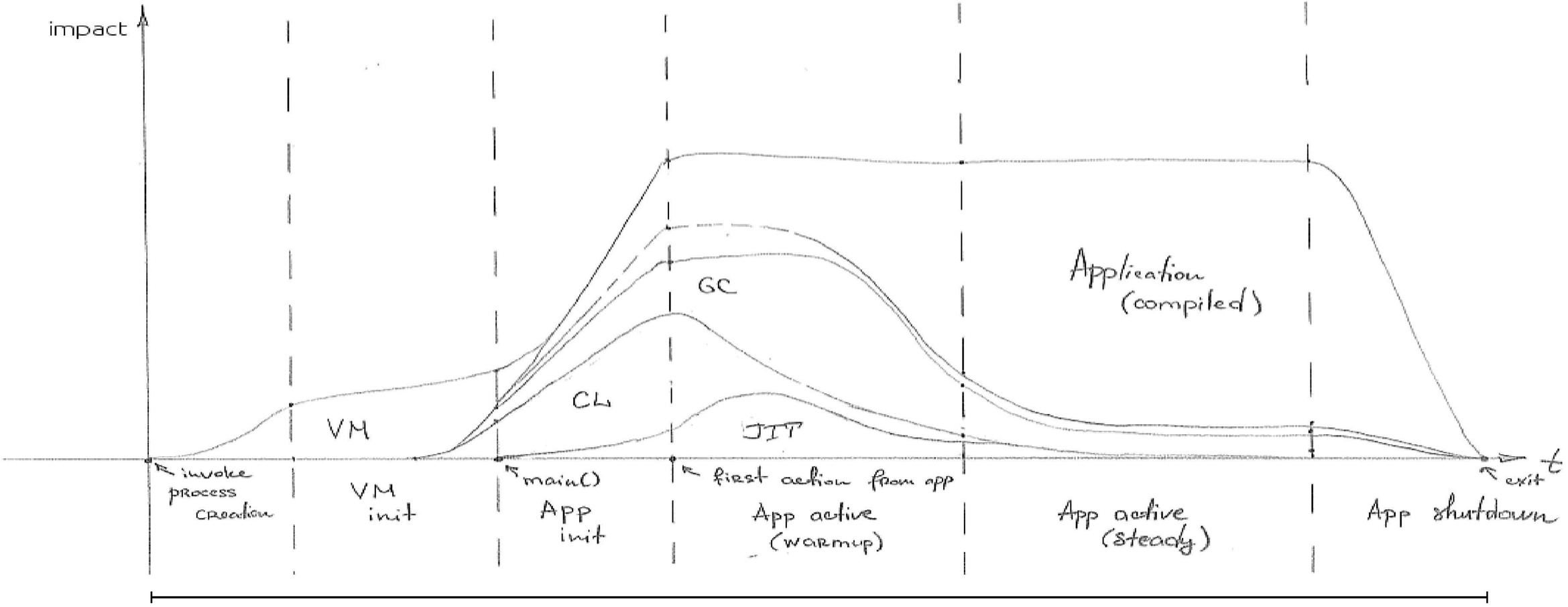
как измерять?



До “стабильного состояния”?

Startup

как измерять?



Время на пуск-остановку?

Startup

Eclipse

- **Типичная конфигурация:**
  - 2x2 Intel i5 2.6 Ghz, Ubuntu 10.10 i686, JDK 6u25
  - Eclipse JDT (Galileo)

- **Типичные метрики:**
  - 6.000 загруженных классов
  - 1.000 методов скомпилировано
  - 512 Mb зарезервированного пространства в куче
  - 25 Mb кучи использовано после стартапа

- **Известные “проблемы”:**
  - Загрузка и верификация классов
  - JIT-компиляция

Startup

Eclipse

- **Метрика: секунд на запуск-завершение**
  - Файловые кеши прогреты, практически нулевой дисковый I/O

	default	CDS	CDS + no-verify
абсолютное время	5.83 [5.74; 5.92]	4.81 [4.73; 4.84]	4.61 [4.56; 4.74]
загрузка классов	5.01 [4.91; 5.11]	3.39 [3.12; 3.66]	3.01 [2.94; 3.08]
компиляция	0.51 [0.43; 0.59]	0.51 [0.44; 0.58]	0.51 [0.42; 0.60]

Startup

длинные приложения

- **Важно только для коротких приложений**
  - Чем дольше работает приложение, тем меньше удельные затраты на загрузку классов и компиляцию
- **Пример: 8 часа работает IntelliJ IDEA 10.x:**
  - 26.600 классов загружено
  - 5315 методов скомпилировано
  - Загрузка классов:
    - Всего потрачено 202 с., ~0.7% общего времени
    - 10 мсек на класс
  - Компиляция:
    - Всего потрачено 112 с., ~0.03% общего времени
    - 20 мсек на метод

Concurrency

JavaOne™

ORACLE®

Concurrency

общие соображения

- **Не изобретайте велосипедов**
  - Лучше не слишком оптимальный, но надёжный код
    - Проще поставить ещё сервер и туда скалироваться
  - JRE никогда не соптимизирует самописные синхронизаторы
- **Пользуйтесь java.util.concurrent.***
  - Не отменяет требований к вменяемости разработчиков
  - Думаете, что нашли баг?
    - http://g.oswego.edu/dl/concurrency-interest/
- **Проверяйте ваши гипотезы**

JIT

JIT

факты

- **… есть**
- **… работает**
- **… работает хорошо**
- **… знает о железе всё:**
  - Количество и тип CPU
  - Поддерживаемые инструкции (SSEx, AVX, VIS)
  - Топологию памяти (в т.ч. размеры кэшей и их характеристики)
- **… знает о приложении много всего:**
  - Иерархию загруженных классов
  - Актуальную статистику создания объектов
  - Горячий код
  - Какие ветвления исполнялись
  - Какие значения использовались
  - Многое другое
- **… не боится использовать эти знания для компиляции**

JIT

оптимизации

compiler tactics
- delayed compilation
- tiered compilation
- on-stack replacement
- delayed reoptimization
- program dependence graph representation
- static single assignment representation

proof-based techniques
- exact type inference
- memory value inference
- memory value tracking
- constant folding
- reassociation
- operator strength reduction
- null check elimination
- type test strength reduction
- type test elimination
- algebraic simplification
- common subexpression elimination
- integer range typing

flow-sensitive rewrites
- conditional constant propagation
- dominating test detection
- flow-carried type narrowing
- dead code elimination

language-specific techniques
- class hierarchy analysis
- devirtualization
- symbolic constant propagation
- autobox elimination
- escape analysis
- lock elision
- lock fusion
- de-reflection

speculative (profile-based) techniques
- optimistic nullness assertions
- optimistic type assertions
- optimistic type strengthening
- optimistic array length strengthening
- untaken branch pruning
- optimistic N-morphic inlining
- branch frequency prediction
- call frequency prediction

memory and placement transformation
- expression hoisting
- expression sinking
- redundant store elimination
- adjacent store fusion
- card-mark elimination
- merge-point splitting

loop transformations
- loop unrolling
- loop peeling
- safepoint elimination
- iteration range splitting
- range check elimination
- loop vectorization

global code shaping
- inlining (graph integration)
- global code motion
- heat-based code layout
- switch balancing
- throw inlining

control flow graph transformation
- local code scheduling
- local code bundling
- delay slot filling
- graph-coloring register allocation
- linear scan register allocation
- live range splitting
- copy coalescing
- constant splitting
- copy removal
- address mode matching
- instruction peepholing
- DFA-based code generator

JIT

performance urban legends

final дает лучшую производительность

копируйте поля в локальные переменные!

Reflection – дорого

volatile запрещает JIT оптимизировать доступ к полю

вызов виртуального метода - дорого

избегайте get/set методов внутри самого класса

вручную вылизанный метод лучше аналога из classlib

immutable классы – плохо

native методы дорогие, System.arraycopy() нативный – значит ...

вручную выполненый inline – хорошо

Java медленна потому, что нельзя вручную выключить проверку выхода индекса за границы массива

создание объектов дорого – используйте Object pooling

JIT

как писать код

- **Используйте стандартные библиотеки**
  - Зачем писать собственный стандартный контейнер?
- **Используйте высокоуровневый API:**
  - java.util.*
  - java.util.concurrent.*
  - NIO, NIO.2
  - вообще библиотеки
- **Код должен правильным и понятным**
  - Сначала правильно
  - Потом алгоритмически “быстро”
  - Код не должен быть JIT-oriented
- **Правильно используйте возможности языка**
  - EPIC FAIL: штатная передача управления exception'ами
  - FAIL: Возврат “исключения” через return <код_ошибки>
  - FAIL: int вместо Enum или boolean

JIT

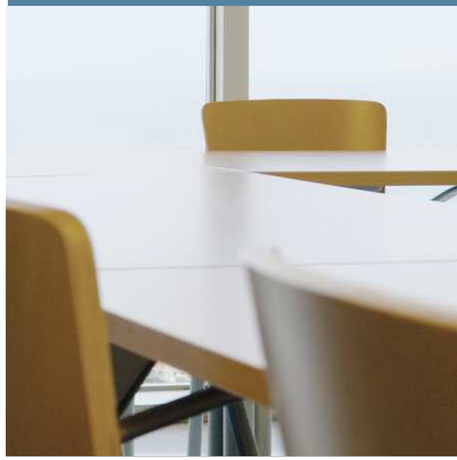
для любопытных

- Как получить ассемблерный код метода?
  - Обычным дебаггером ;)
  - JVMTI
  - -XX:+PrintAssembly
    - http://wikis.sun.com/display/HotSpotInternals/PrintAssembly


НЕСМОТРЯ НА ДЕТАЛЬНЫЙ АНАЛИЗ ТЕКУЩЕЙ СИТУАЦИИ, Я ТАК И НЕ СМОГ СОСТАВИТЬ ЧЁТКОЕ ПРЕДСТАВЛЕНИЕ ОБ ОБСУЖДАЕМОЙ ПРОБЛЕМЕ В СИЛУ ВОЗНИКШЕГО КОНГИТИВНОГО ДИССОНАНСА.

Q/A

Appendix

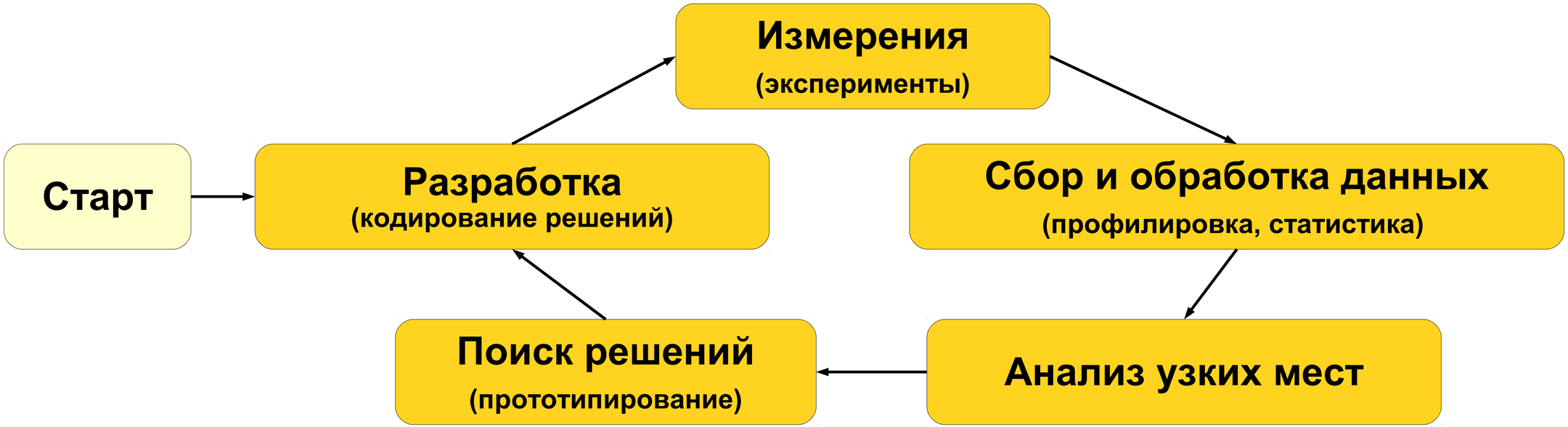


JavaOne™

ORACLE®

Performance Engineering

итеративный подход



Важно:

- Одно изменение за цикл!
- Документировать все изменения

JavaOne™

ORACLE®

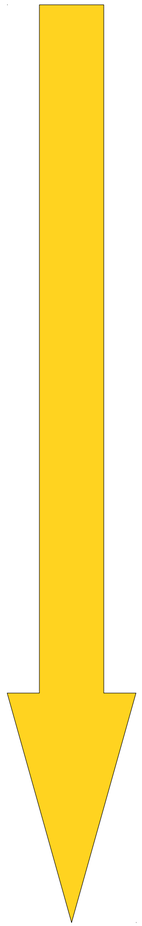
Performance Engineering

анализ узких мест

- **Что ограничивает скорость работы приложения?**
  - CPU
  - Ядро ОС
  - I/O (Сеть, Диск)

Performance Engineering

”нисходящий” метод поиска узких мест



- Уровень системы
  - Сеть
  - Диск
  - Database
  - Операционная система
  - Процессор/память
- Уровень приложения
  - Блокировки, синхронизация
  - Execution Threads
  - API
  - Алгоритмические проблемы
- Уровень JVM
  - Выбор JVM
  - Heap tuning
  - JVM tuning

Java 7, Java 8

что ожидать в области производительности

- **Java 7**
  - invokedynamic
  - NIO.2
  - Concurrency and Collection updates (Fork/Join)
  - XRender pipeline for Java2D (client)

- **Java 8 (или позже)**
  - Модульность
  - λ-выражения (замыкания)
  - Collection updates (filter, map, reduce)

Обратная совместимость

- Мешает ли улучшать производительность JVM?
  - **Да**, поэтому иногда расширяемся:
    - invokedynamic
    - Модульность
- Стоит ли расширяться по первому требованию?
  - **Нет**, развитие JVM/JIT, реализация новых методов оптимизации позволяет получить бОльший выигрыш
    - Вложенные классы
    - Reflection

Лучшая ОС для Java

угадайте, какая?

Solaris

- **Высокопроизводительный TCP/IP стек**
  - low-latency
  - up to 50% faster
- **DTrace**
  - мониторинг
- **NUMA**
  - MPO, Memory Placement Optimization
- **Large Pages**
  - Автоматическая аллокация
  - Разные размеры

Concurrency

элементная база

- **OS Threading**
  - мьютексы
    - mutex_lock()/mutex_unlock()
  - conditional waits
    - cond_wait()/cond_signal()
    - WaitForSingleObject
- **Compare-and-Swap (CAS)**
  - CAS(x1, x2, x3) = { if (x1 == x2) { x1 = x3 }; }
  - атомарная операция, поддерживаемая в “железе”: из нескольких одновременных CAS'ов успешно завершается только один
  - Миф: локальный CAS блокирует шину, и стоит больше на многопроцессорных системах
  - Факт: глобальный CAS требует трафика на шине

Concurrency

atomics

- **java.util.concurrent.Atomic***
  - обеспечивают атомарные операции над примитивами и указателями
  - альтернатива: synchronized {} или Lock'и
- **Трюк в использовании CAS'а:**
  - Изменение состояния атомика делается при помощи одного CAS'а
  - Чтение состояния не требует CAS'а

```
public final int incrementAndGet() {
    for (;;) {
        int current = get();
        int next = current + 1;
        if (compareAndSet(current, next))
            return next;
    }
}
```

```
mov     %ecx,%edx
mov     0x8(%ecx),%eax
lea     0x8(%ecx),%edi
mov     %eax,%ecx
inc     %ecx
lock cmpxchg %ecx,(%edi)
mov     $0x0,%ebx
jne     [ok]
mov     $0x1,%ebx
test    %ebx,%ebx
je      [ok]
```

Concurrency

volatile

- **Volatile определяет порядок чтения-записей в поле**
  - НЕ обеспечивает атомарности
  - Реализуется расстановкой барьеров
    - Какие из них вставятся в код, зависит от Hardware Memory Model
    - Эффект барьера зависит от HMM

```
PUSHL   EBP
SUB     ESP,8
MOV     EBX,[ECX + #12]
MEMBAR-acquire
MEMBAR-release
INC     EBX
MOV     [ECX + #12],EBX
MEMBAR-volatile
LOCK ADDL [ESP + #0], 0
ADD     ESP,8
POPL    EBP
TEST    PollPage,EAX
RET
```

```
push    %ebp
sub     $0x8,%esp
mov     0xc(%ecx),%ebx
inc     %ebx
mov     %ebx,0xc(%ecx)
lock addl $0x0,(%esp)
add     $0x8,%esp
pop     %ebp
test    %eax,0xb779c000
ret
```

Concurrency

intrinsic synchronization

- synchronized(object) { }, 4 состояния:
  - **Init**
  - **Biased**
    - Захватывается одним “владеющим” потоком, нет конфликтов
    - Захват владельцем: проверка на threadID
    - Захват не-владельцем: переход либо в Biased, либо в Thin
  - **Thin**
    - Захватывается несколькими потоками, но конфликтов нет
    - Захват: CAS
    - Конфликтный захват: переход в Fat
  - **Fat**
    - Захватывается несколькими потоками, конфликт на блокировке
    - Вызов примитива синхронизации из ОС

Concurrency

java.util.concurrent.Lock

- **Построены на базе j.u.c.AbstractQueueSynchronizer**
  - Использует CAS
  - Использует Unsafe.park()/unpark() → cond_wait()/cond_signal()/WaitForSingleObject()
- **ReentrantLock**
  - По семантике эквивалентен synchronized {}
  - Ставит потоки во внутреннюю очередь и делает park()
  - Non-Fair (default)
    - Не гарантирует отсутствие starvation, ибо barging FIFO (CAS)
    - Лучшая производительность
  - Fair
    - Гарантирует отсутствие starvation, FIFO
    - Честность в обмен на производительность

Concurrency

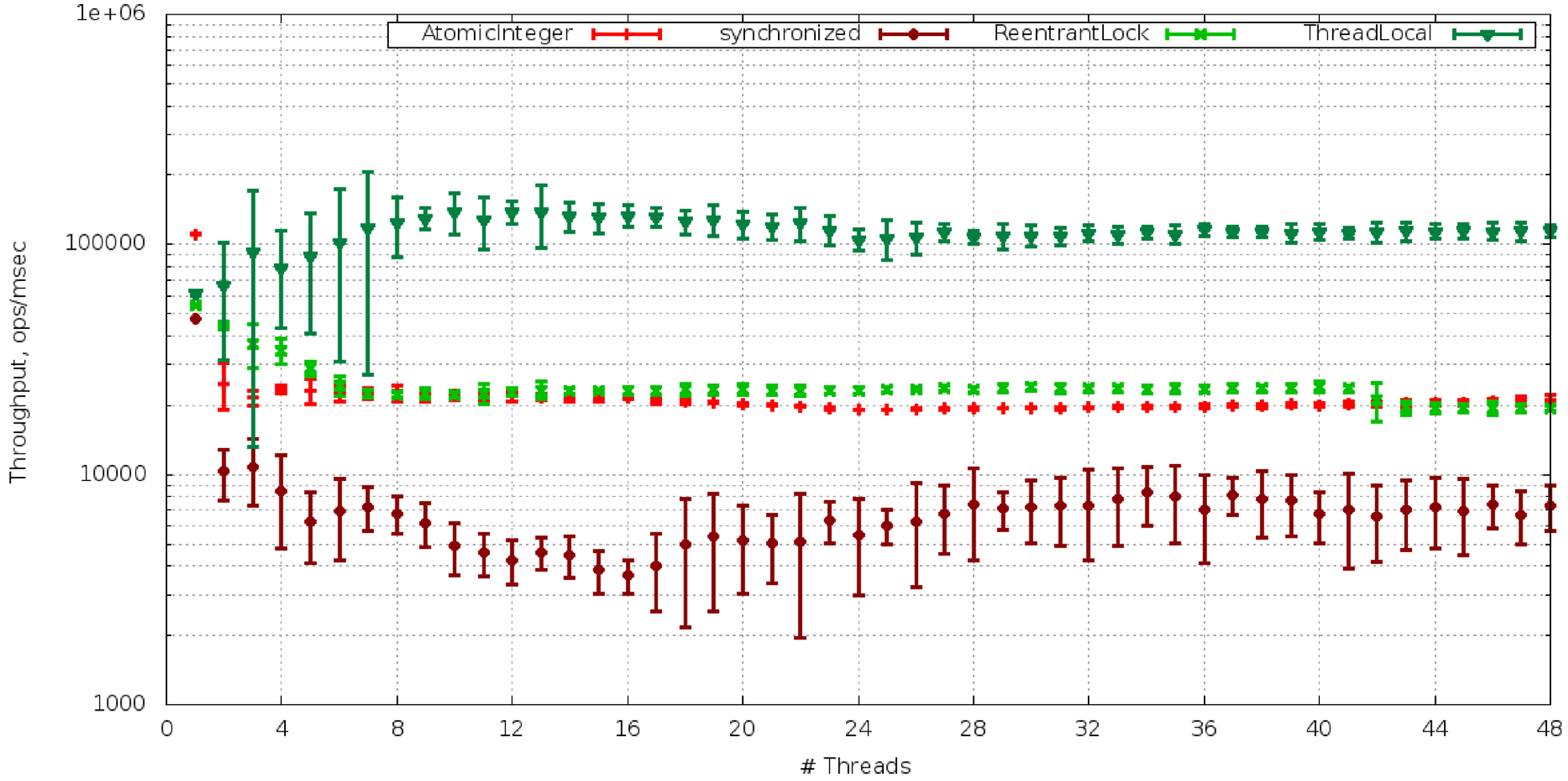
атомарный счётчик

```
private AtomicInteger atomic = new AtomicInteger();
private ReentrantLock lock = new ReentrantLock();
private final Object intrinsicLock = new Object();
private int primCounter = 0;

@GenerateMicroBenchmark
public void testAtomicInteger() {
    atomic.incrementAndGet();
}

@GenerateMicroBenchmark
public void testReentrantLock() {
    lock.lock();
    primCounter++;
    lock.unlock();
}

@GenerateMicroBenchmark
public void testIntrinsicLock() {
    synchronized (intrinsicLock) {
        primCounter++;
    }
}
```

ORACLE®

Concurrency

атомарный счётчик



Intel Xeon (Westmere-EP), 3.0 Ghz, 2x6x2 = 24 HW threads, SLES 11 x86_64, JDK 6u25

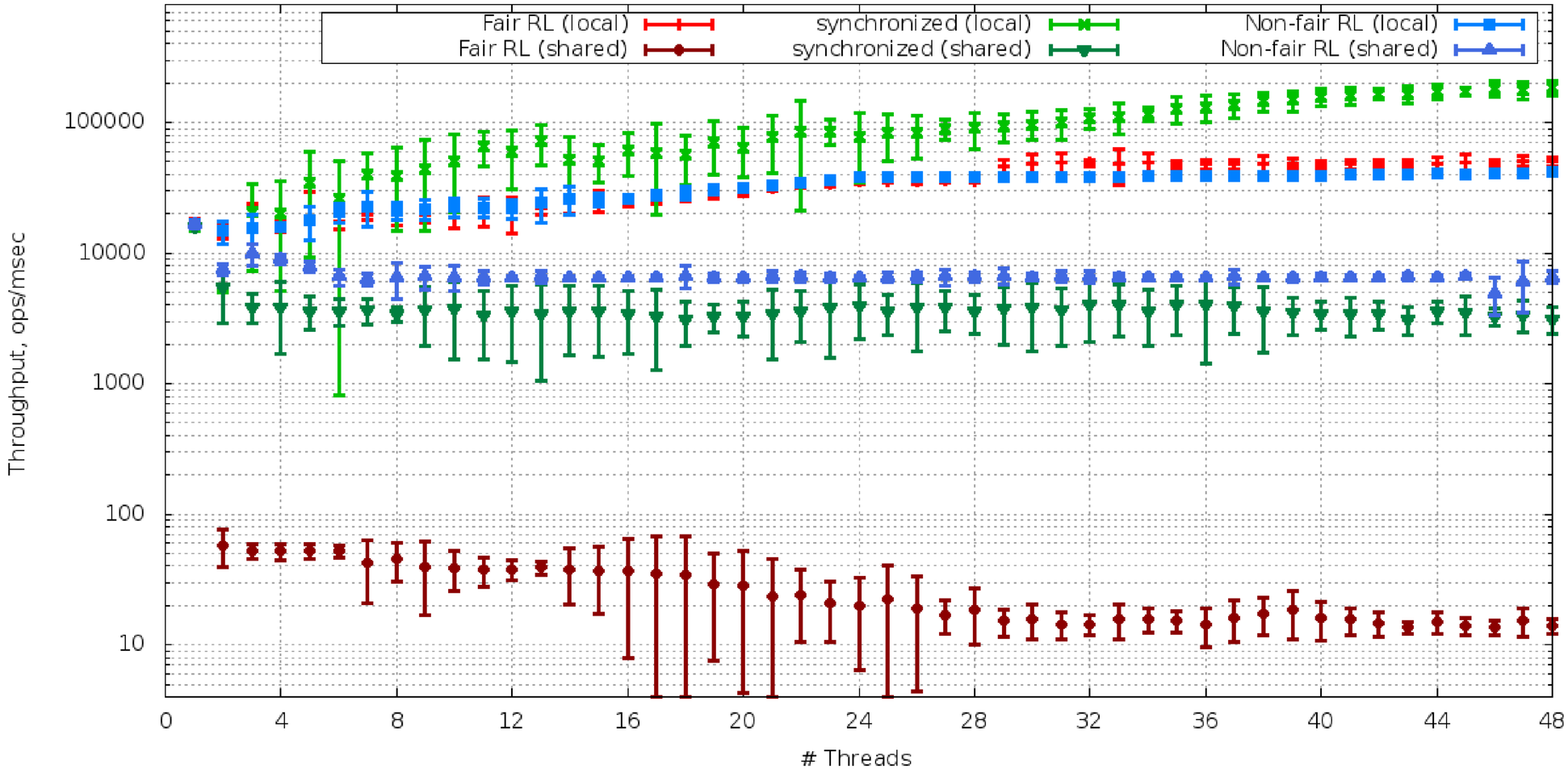
Concurrency

ReentrantLock vs. synchronized

- **Семантика одинакова**
  - Требования к видимости памяти
  - Рекурсивный
- **Плюсы j.u.c.RL**
  - Очередь потоков держится на стороне JVM
    - опционально, FIFO-политика при захвате-освобождении
    - позволяет быть “честным” на любой платформе
  - Barging FIFO policy
    - lock() может быть сразу удовлетворён, даже если в очереди есть потоки
    - сильно улучшает производительность при конфликте блокировок
  - Допускается несколько Condition
- **Минусы j.u.c.RL**
  - Нет scope'ов, требуется ручной unlock() через finally

Concurrency

производительность захвата



Intel Xeon (Westmere-EP), 3.0 Ghz, 2x6x2 = 24 HW threads, SLES 11 x86_64, JDK 6u25


JavaOne™

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.

The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.


JavaOne™