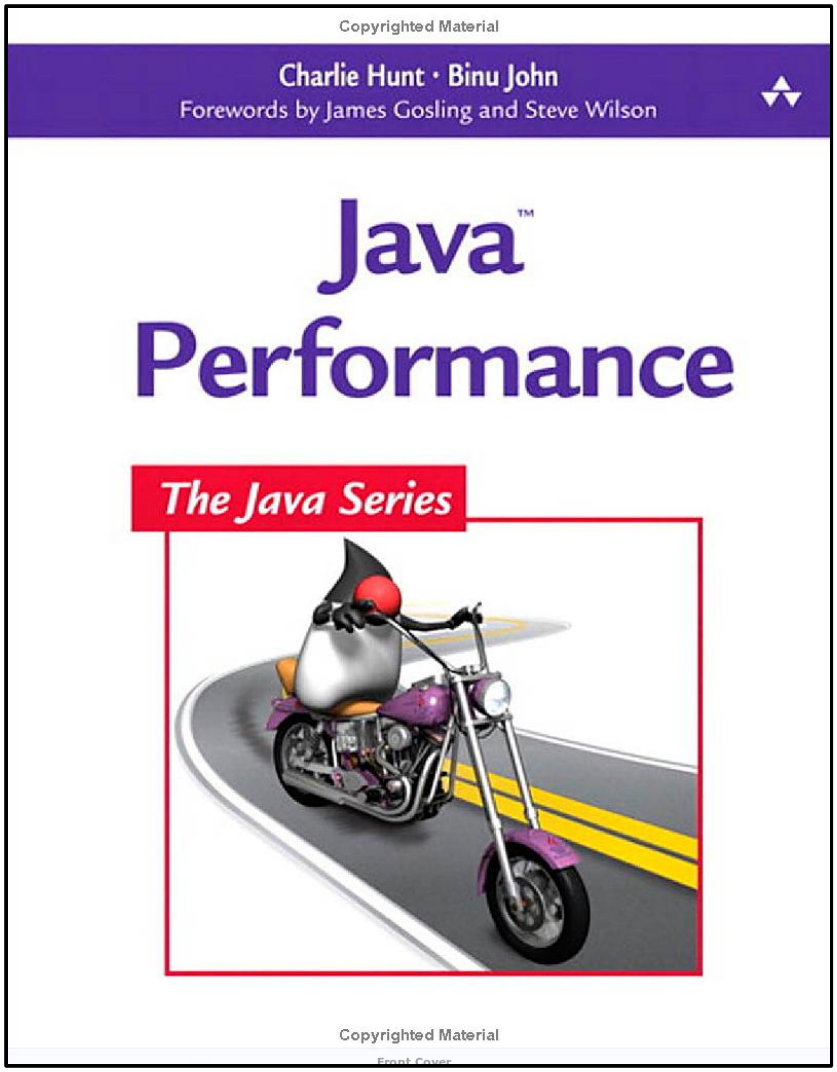
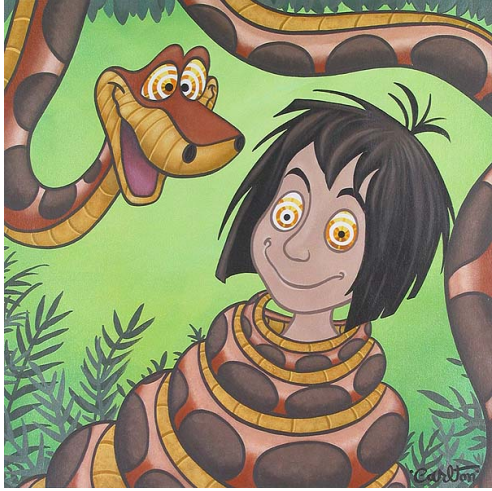
ORACLE®

Intro: Performance Engineering

Sergey Kuksenko, Aleksey Shipilev


Copyrighted Material
Charlie Hunt · Binu John
Forewords by James Gosling and Steve Wilson
Java™
Performance
The Java Series
Copyrighted Material



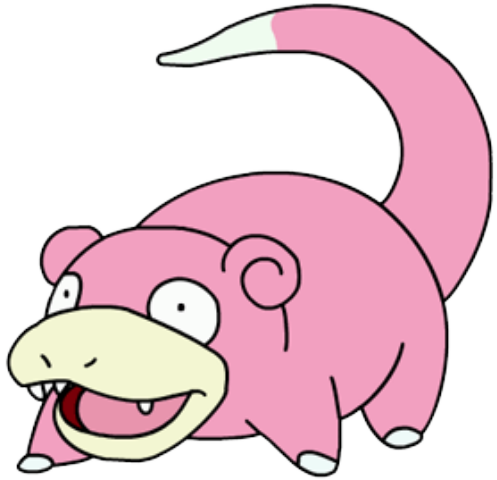
Performance Engineering

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
    - Основываемся на эмпирических данных



У меня всё медленно!
Ваш первый шаг?

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”

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:**
  - **Выбрать метрику**
    - ops/sec, transactions/sec
    - время исполнения
    - время отклика
  - **Убедиться в корректности метрики**
    - релевантна (учитывает реальный сценарий работы приложения)
    - повторяема

Цель – улучшение метрики!

Метрики

Метрики

Throughput, bandwidth

- **Throughput, Bandwidth**
  - Количество работы, выполненное системой за единицу времени
  - Принимает разные формы:
    - MB/sec
    - ops/sec, transactions/sec
    - FPS (frames per second, frags per second)
    - MIPS
    - FLOPS

Метрики

Latency, Response Time

- **Время...**
  - ...работы
    - Execution time: общее время исполнения
  - ...отклика
    - Latency: время отдельной операции
    - Response time: задержка между стимулом и реакцией
  - ...запуска
    - Startup time: время до начала работы
    - Time to performance: время до начала *хорошей* работы
  - etc.

Метрики

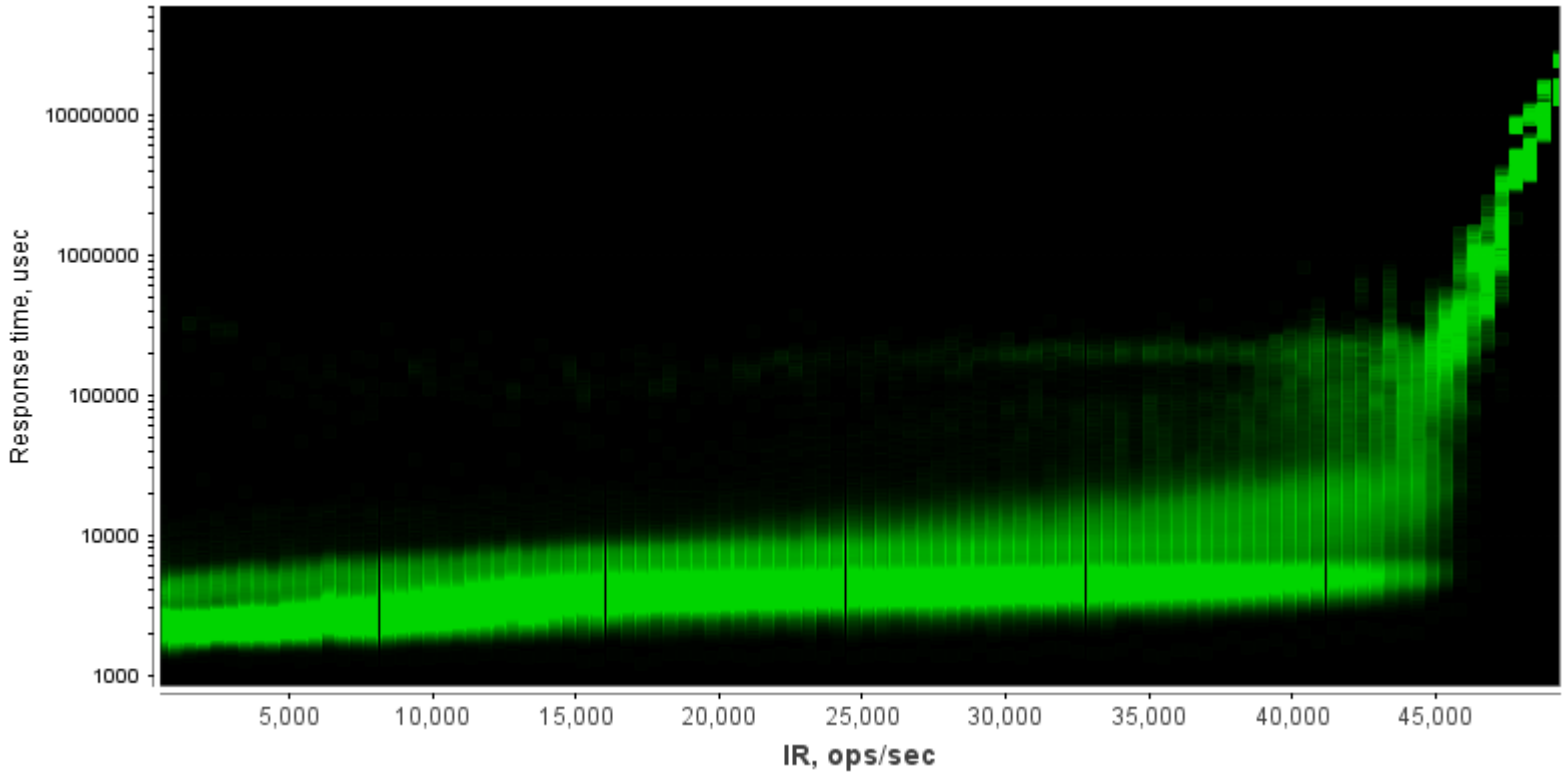
Throughput vs. Response Time

- **Производительность – композитная метрика:**
  - throughput **vs** RT → throughput **&** RT
- **Оказывается, что проще улучшить throughput:**
  - DDR2 PC2-3200: 3200 Mb/sec, CL 4ns
  - DDR2 PC2-6400: 6400 Mb/sec, CL 5ns
- **В большинстве систем, RT тесно связан с throughput**
  - Из пункта А в пункт Б один автобус может перевезти сотню человек
    - Достаточно много автобусов перевезут миллион человек
    - Несколько больших автопоездов перевезут миллион человек
    - ...но в обоих случаях будут дикие очереди на погрузку и выгрузку :)

Метрики

Throughput vs. Response Time

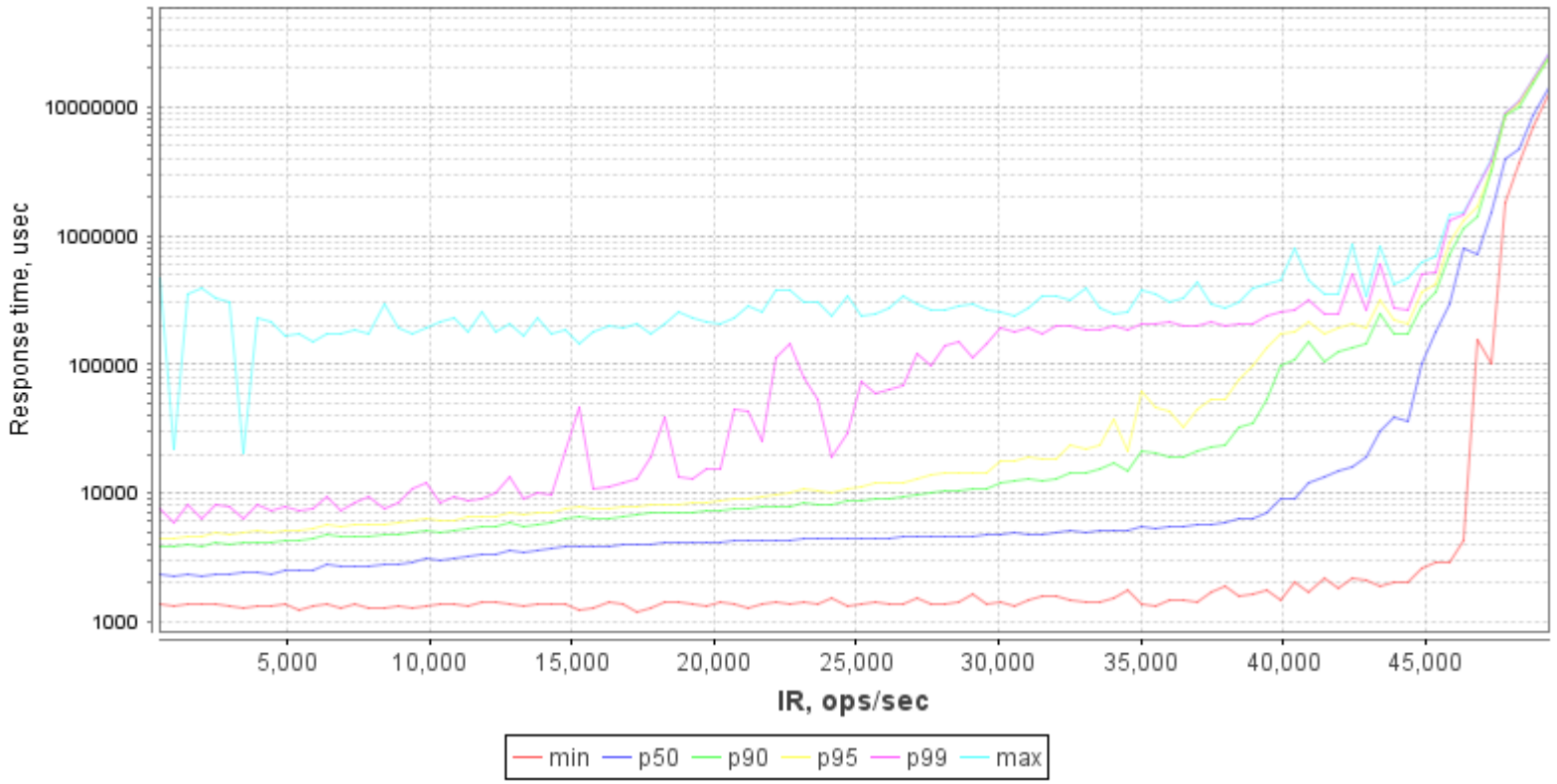
- **Обычно RT быстро растёт с throughput**



Метрики

Throughput vs. Response Time

- **Обычно RT быстро растёт с throughput**



Метрики

Прочие

- **Прочие метрики:**
  - потребление ресурсов
    - память, сеть, etc.
  - отказоустойчивость
    - MTBF, MTTR
  - потребляемая и отводимая мощность
    - performance per watt
    - TDP
  - etc.

Теория

Теория

Утилизация

- **Утилизация – на сколько ресурс занят?**

$$Utilization = \frac{ResourceBusyTime}{TotalTime}$$

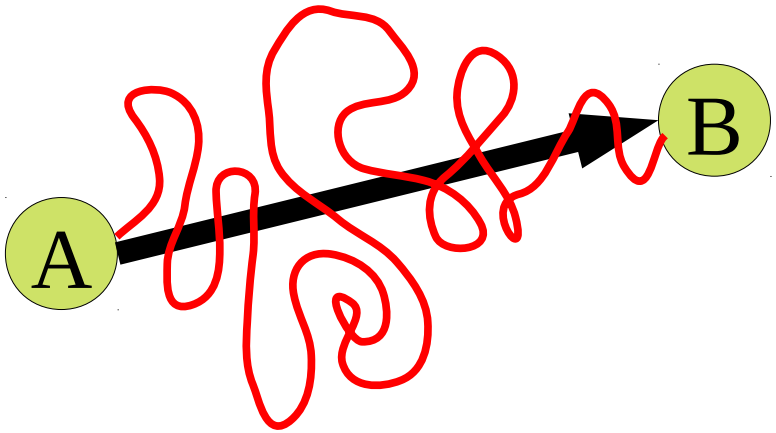
- **Idle – на сколько ресурс свободен?**

$$IdleTime = 1 - Utilization$$

Теория

Эффективность

- **Эффективность (efficiency)**
  - Оценка КПД для времени: часть общего времени, потраченная на выполнение **полезной** работы
  - Субъективно, невозможно строго вычислить
    - Высокая утилизация CPU не означает хорошую эффективность



Теория

SpeedUp

- **“A в N раз быстрее B” означает**

$$SpeedUp = \frac{time(B)}{time(A)} = \frac{throughput(A)}{throughput(B)}$$

Теория

%Boost

- **“A на n% быстрее B” означает:**

$$SpeedUp = 1 + \frac{n}{100\%}$$

$$Boost\% = (SpeedUp - 1) * 100\%$$

$$Boost\% = \frac{time(B) - time(A)}{time(A)}$$

$$Boost\% = \frac{throughput(A) - throughput(B)}{throughput(B)}$$

Теория

Закон Амдала

- **Допустим, есть две независимые части:**
  - Часть **A** занимает 70% времени, ускорябельна в 2 раза
  - Часть **B** занимает 30% времени, ускорябельна в 6 раз
  - Где профит больше?

Теория

Закон Амдала

- **Допустим, есть две независимые части:**
  - Часть **A** занимает 70% времени, ускорябельна в 2 раза
  - Часть **B** занимает 30% времени, ускорябельна в 6 раз
  - Где профит больше?

Ускорим **B** в 6 раз:

Ускорим **A** в 2 раза:

Теория

Закон Амдала

- **Закон Амдала**

$$Part(A) = \frac{A}{A + B}$$

$$SpeedUp = \frac{1}{(1 - Part(A)) + \frac{Part(A)}{SpeedUp(A)}}$$

Теория

Закон Амдала

- **Допустим, есть две независимые части:**
  - Часть **A** занимает 70% времени, ускорябельна в 2 раза
  - Часть **B** занимает 30% времени, ускорябельна в 6 раз
  - Где профит больше?

Ускорим **B** в 6 раз:

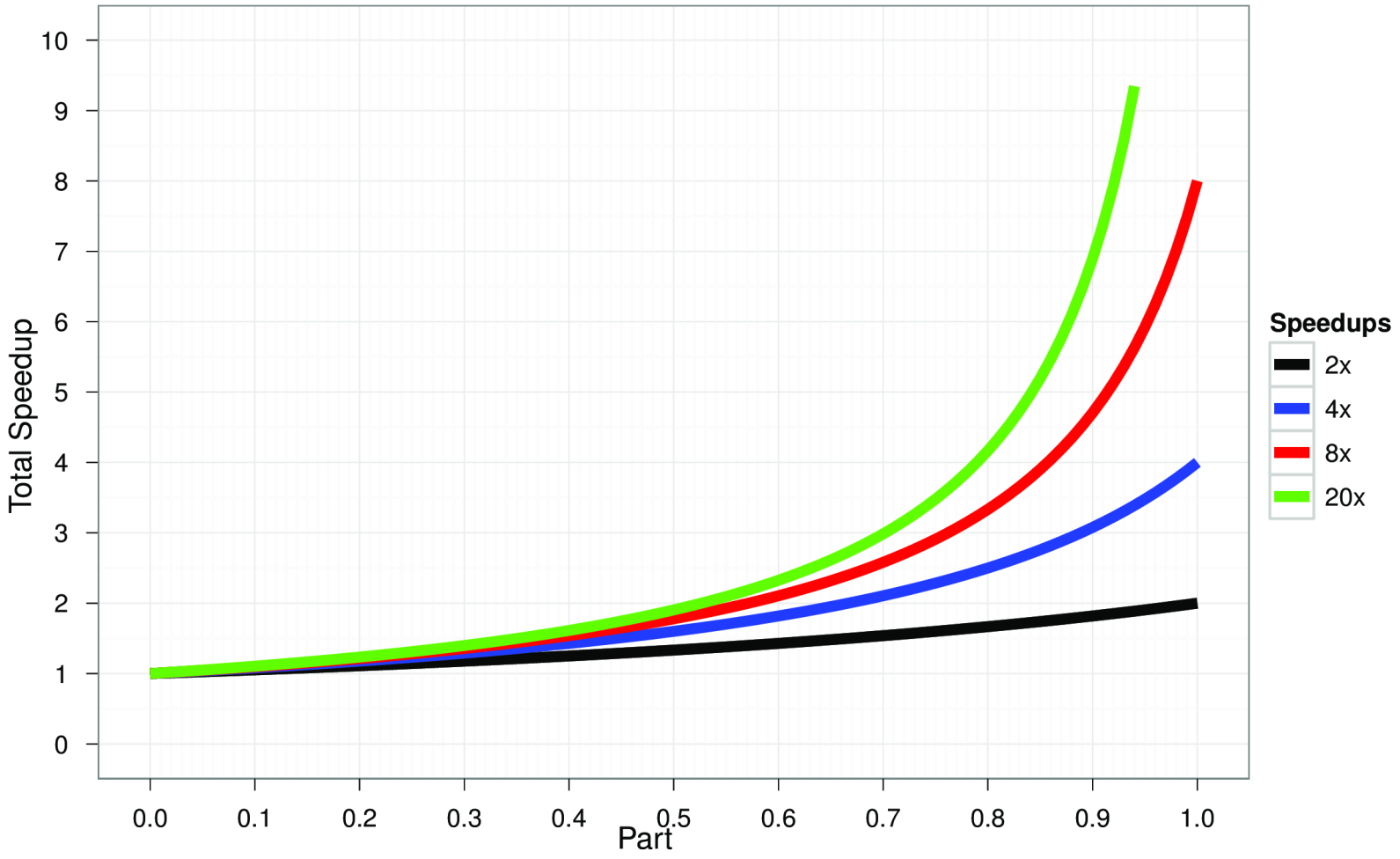
+33%

Ускорим **A** в 2 раза:

+53%

Теория

Закон Амдала



Теория

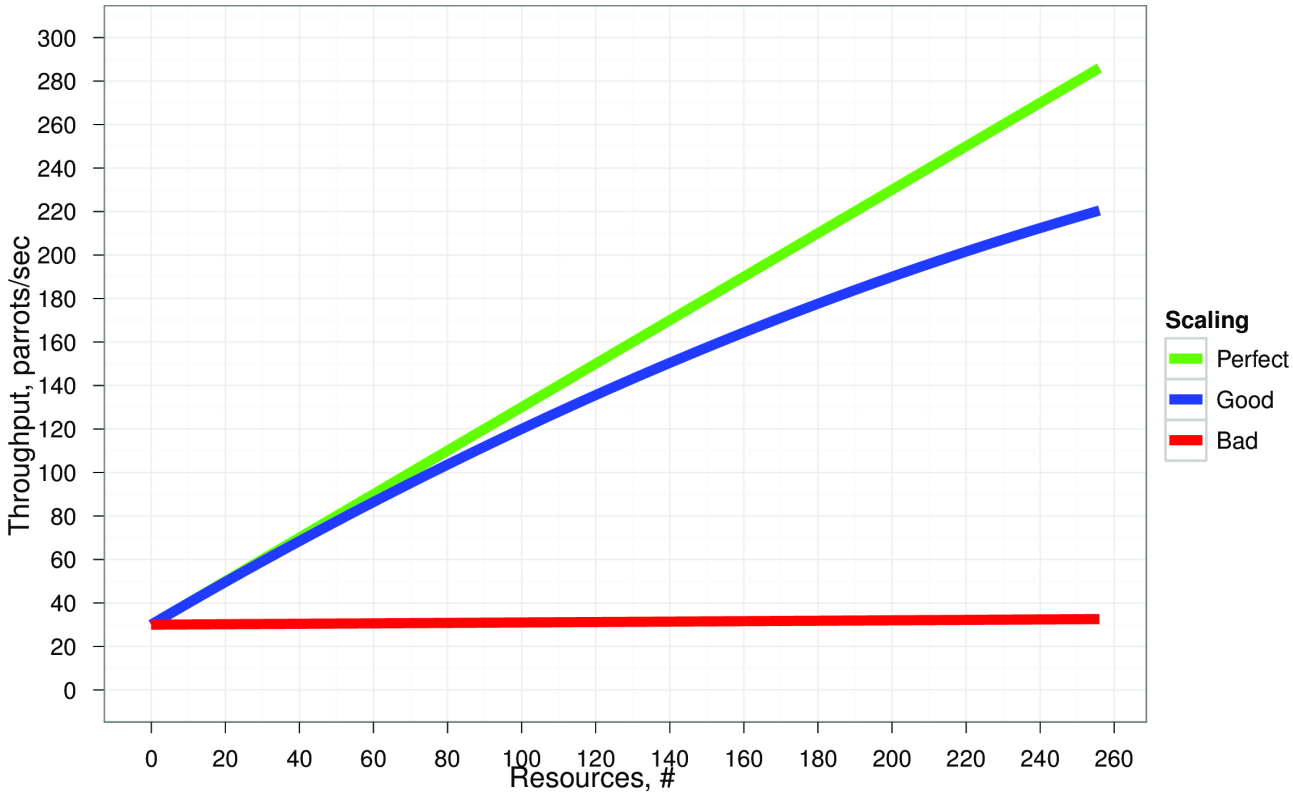
Scalability

- **Масштабируемость (scalability)**
  - Способность системы увеличивать производительность при добавлении ресурсов
- **Ресурсы:**
  - CPU ~ processor scaling
  - тактовая частота ~ frequency scaling
  - память ~ memory scaling
  - etc.

Теория

Scalability (вариант 1)

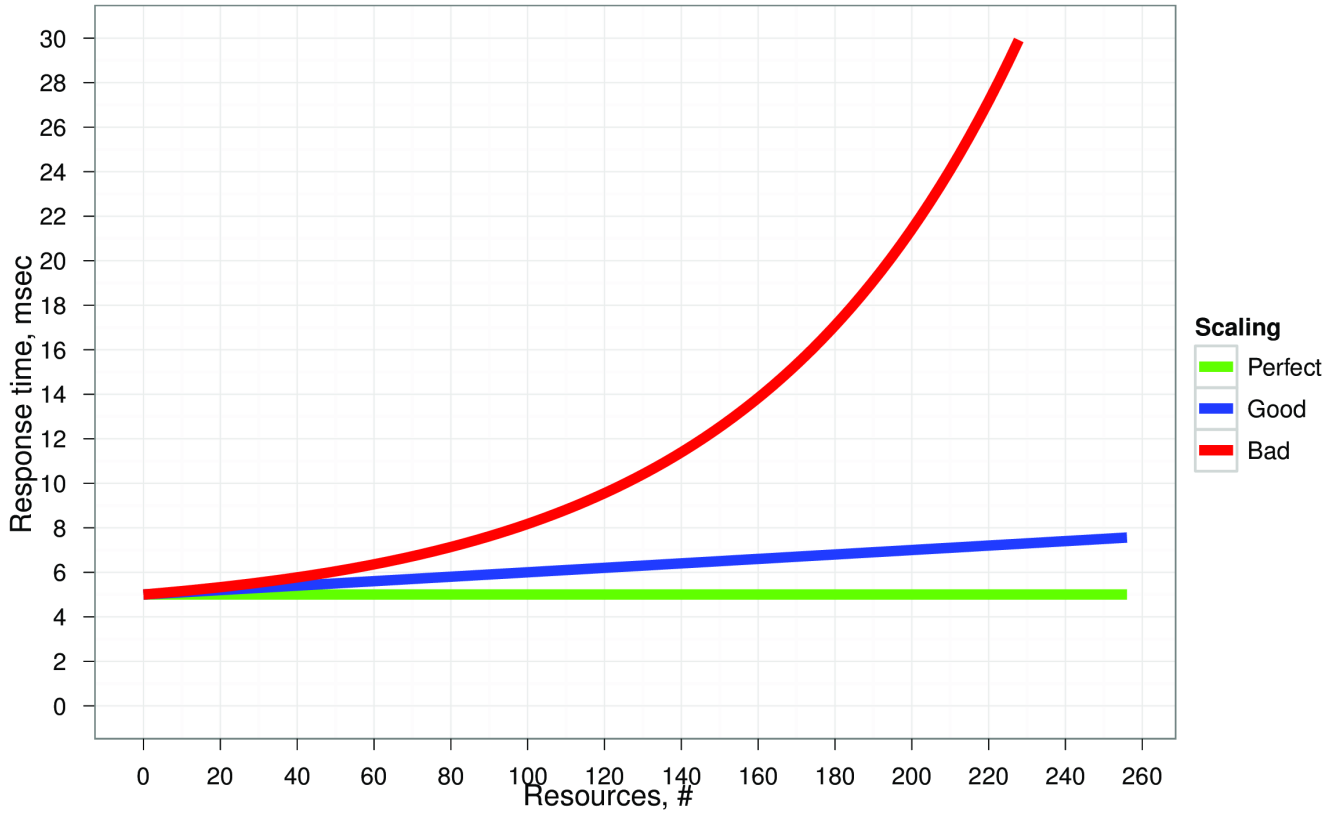
Добавляем ресурсы → работаем быстрее



Теория

Scalability (вариант 2)

Добавляем ресурсы и нагрузку → работаем так же



Методология

Классические ошибки второго шага

Узнай себя!

- **«*Я вижу*, что метод foo() реализован плохо, перепишем и посмотрим, что изменится»**

Классические ошибки второго шага

Узнай себя!

- **«*Я вижу*, что метод foo() реализован плохо, перепишем и посмотрим, что изменится»**
  - …а метод не используется вообще.
  - …или используется, но занимает пару миллисекунд
  - …или используется, но *дело не в этом*.
- **Не самый плохой вариант, если таких методов мало, и изменения очень быстрые**

Классические ошибки второго шага

Узнай себя!

- **«*По профилю видно*, что метод bar() – самый горячий и занимает целых 5%, надо его убить»**

Классические ошибки второго шага

Узнай себя!

- **«*По профилю видно*, что метод bar() – самый горячий и занимает целых 5%, надо его убить»**
  - …а оказывается, что это 5% от всего приложения, которое занимает 6.25% CPU 16-процессорной системы
  - …или это метод, помогающий всем остальным быть быстрее
  - …или он действительно проблемный, но *дело не в этом*.

Классические ошибки второго шага

Узнай себя!

- ***«Полносистемная профилировка показывает,*** **что у нас тормозит база данных, и нужно срочно перейти с СУБД$_X$ на СУБД$_Y$»**

Классические ошибки второго шага

Узнай себя!

- ***«Полносистемная профилировка показывает,*** **что у нас тормозит база данных, и нужно срочно перейти с СУБД$_X$ на СУБД$_Y$»**
  - …а вдруг оказывается, что диски слабоваты
  - …или оказывается, что злые админы зашейпили сеть
  - …или просто БД уже давно никто не «пылесосил»

- **Без шуток, мы наблюдали спонтанные переезды с X на Y**
  - ...и обратно, немного погодя. А потом опять с X на Y.

Как ускорить приложение?

- К.О. сообщает:
  - “Нужно что-то где-то как-то изменить!”
- Что?
- Где?
- Как?

Как ускорить приложение?

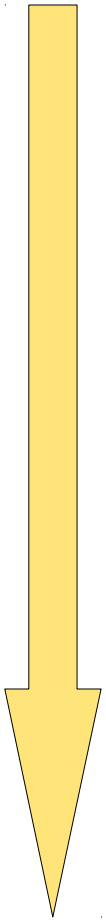
- К.О. сообщает:
  - “Нужно что-то где-то как-то изменить!”
- Что мешает работать быстрее?
- Где это находится?
- Как это исправить?

Как ускорить приложение?

- **К.О. сообщает:**
  - **“Нужно что-то где-то как-то изменить!”**
- **Что мешает работать быстрее?**
  - Используем голову и monitoring tools
- **Где это находится?**
  - Используем голову и profiling tools
- **Как это исправить?**
  - Используем голову и прямые руки

Performance Engineering

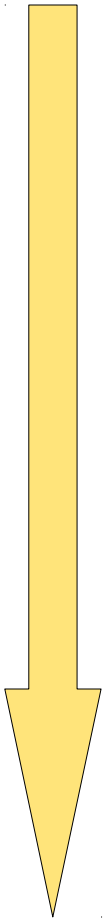
Классический ”нисходящий” метод поиска узких мест



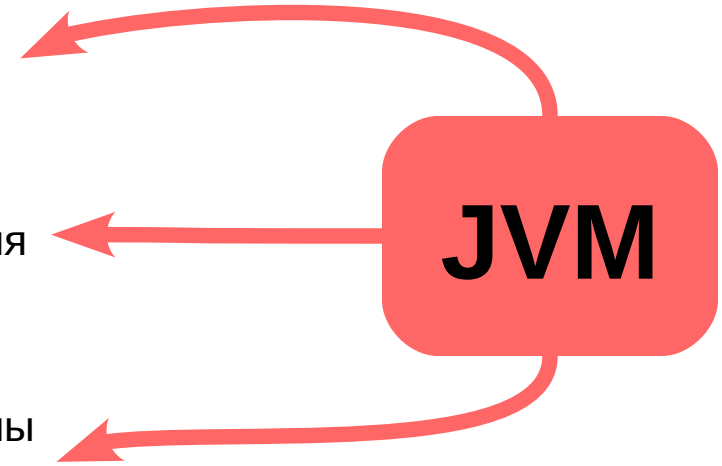
- **Уровень системы**
  - Сеть
  - Диск
  - Операционная система
  - Процессор/память
- **Уровень приложения**
  - Блокировки, синхронизация
  - Execution Threads
  - API
  - Алгоритмические проблемы
- **Микроархитектурный уровень**
  - Code/data alignment
  - Cache optimizations
  - Processor stalls
  - Branch prediction

Performance Engineering

”нисходящий” метод поиска узких мест

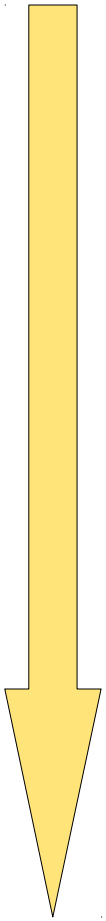


- **Уровень системы**
  - Сеть
  - Диск
  - Операционная система
  - Процессор/память
- **Уровень приложения**
  - Блокировки, синхронизация
  - Execution Threads
  - API
  - Алгоритмические проблемы
- **Микроархитектурный уровень**
  - Code/data alignment
  - Cache optimizations
  - Processor stalls
  - Branch prediction



Performance Engineering

”нисходящий” метод поиска узких мест (Java World)



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

Performance Engineering

инструменты для анализа системы (monitoring tools)

	Solaris	Linux	Windows	Что смотрим
Сеть	netstat, dtrace, nicstat	netstat, nicstat	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
- NetBeans Profiler
  - http://www.netbeans.org
- DTrace
  - http://www.oracle.com/technetwork/systems/dtrace/dtrace/index.html
- Ещё могут быть полезны:
  - JProbe
  - OptimizeIt
  - YourKit

Tips and Tricks

Системный уровень

- **I/O**
  - Сеть
  - Диск
- **Memory paging (swapping)**
- **CPU**

Системный уровень

CPU

- **CPU, на что обратить внимание**
  - CPU utilization
    - User time
    - System (kernel) time
    - Idle time
  - Context switching
    - Voluntary context switching
    - Involuntary context switching
  - CPU scheduler run queue length

Системный уровень

CPU

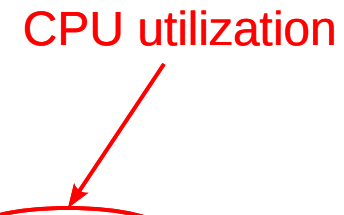
vmstat

```
procs -----------memory----------   ---swap-- -----io---- -system-- ----cpu----
 r   b   swpd   free   buff  cache    si   so    bi    bo   in   cs us sy id wa
256  0      0 2945568 115784 556012    0    0     0     0 1396 1879 99   1  0  0
256  0      0 2946176 115792 556012    0    0     0    40 1170 1608 100  0  0  0
256  0      0 2945888 115792 556012    0    0     0     0 1310 1818 100  0  0  0
256  0      0 2945600 115792 556012    0    0     0     0 1189 1609 100  1  0  0
256  0      0 2946176 115792 556012    0    0     0     0 1261 1755 100  1  0  0
256  0      0 2945888 115792 556012    0    0     0     0 1149 1562 100  0  0  0
12   0      0 2945616 115800 556012    0    0     0    20 1445 2952 97   0  3  0
177  0      0 2946312 115800 556012    0    0     0     0 1358 2150 96   1  3  0
215  0      0 2945988 115800 556012    0    0     0     0 1374 2497 100  0  0  0
180  0      0 2946108 115800 556012    0    0     0     0 1141 1604 99   1  0  0
127  0      0 2946876 115800 556012    0    0    40     0 1271 1810 100  0  0  0
90   0      0 2947164 115808 556052    0    0     0    24 1283 1786 100  1  0  0
```

Системный уровень

CPU

vmstat

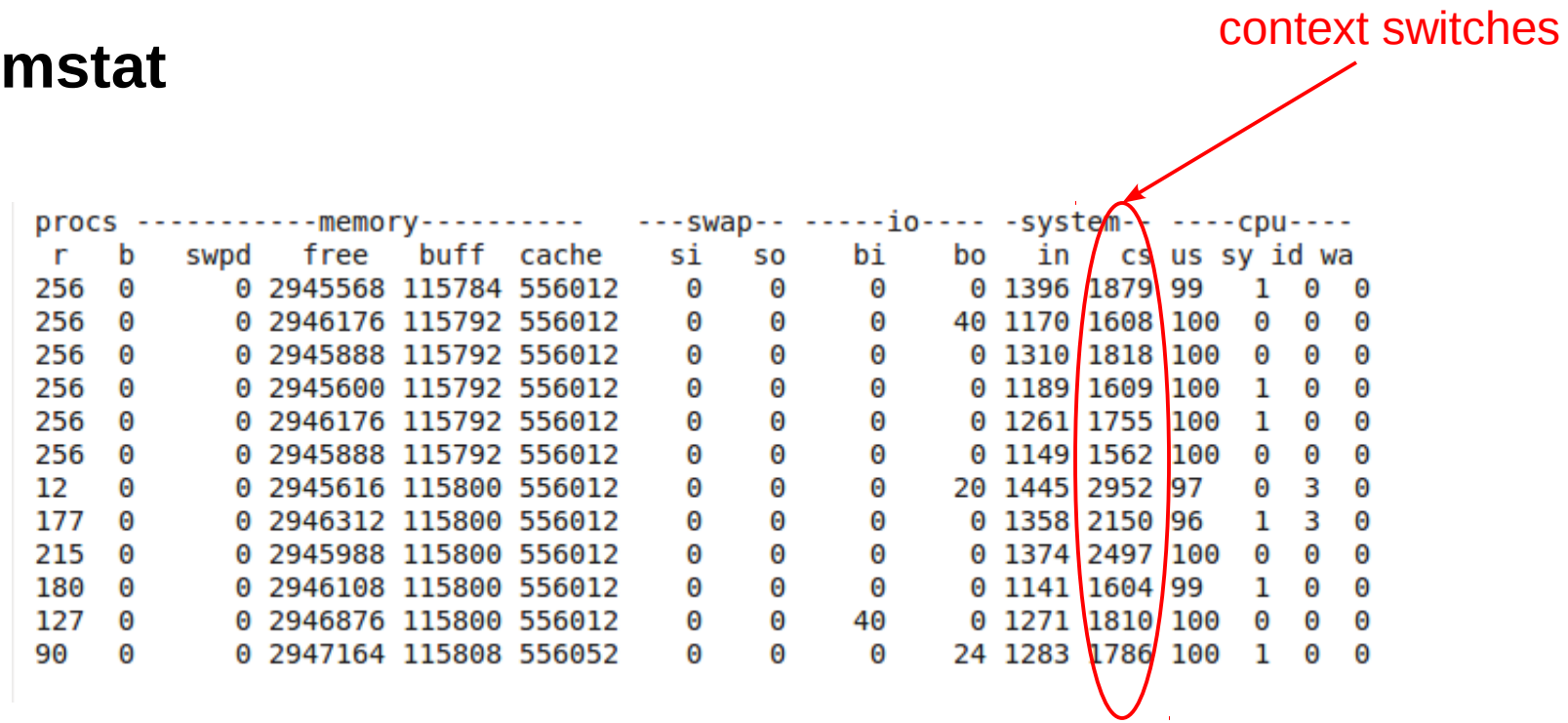



```
procs -----------memory----------   ---swap-- -----io---- -system-- ----cpu----
 r   b   swpd   free   buff  cache    si   so    bi    bo   in   cs us sy id wa
256  0      0 2945568 115784 556012    0    0     0     0 1396 1879 99   1  0  0
256  0      0 2946176 115792 556012    0    0     0    40 1170 1608 100  0  0  0
256  0      0 2945888 115792 556012    0    0     0     0 1310 1818 100  0  0  0
256  0      0 2945600 115792 556012    0    0     0     0 1189 1609 100  1  0  0
256  0      0 2946176 115792 556012    0    0     0     0 1261 1755 100  1  0  0
256  0      0 2945888 115792 556012    0    0     0     0 1149 1562 100  0  0  0
12   0      0 2945616 115800 556012    0    0     0    20 1445 2952 97   0  3  0
177  0      0 2946312 115800 556012    0    0     0     0 1358 2150 96   1  3  0
215  0      0 2945988 115800 556012    0    0     0     0 1374 2497 100  0  0  0
180  0      0 2946108 115800 556012    0    0     0     0 1141 1604 99   1  0  0
127  0      0 2946876 115800 556012    0    0    40     0 1271 1810 100  0  0  0
90   0      0 2947164 115808 556052    0    0     0    24 1283 1786 100  1  0  0
```

Системный уровень

CPU

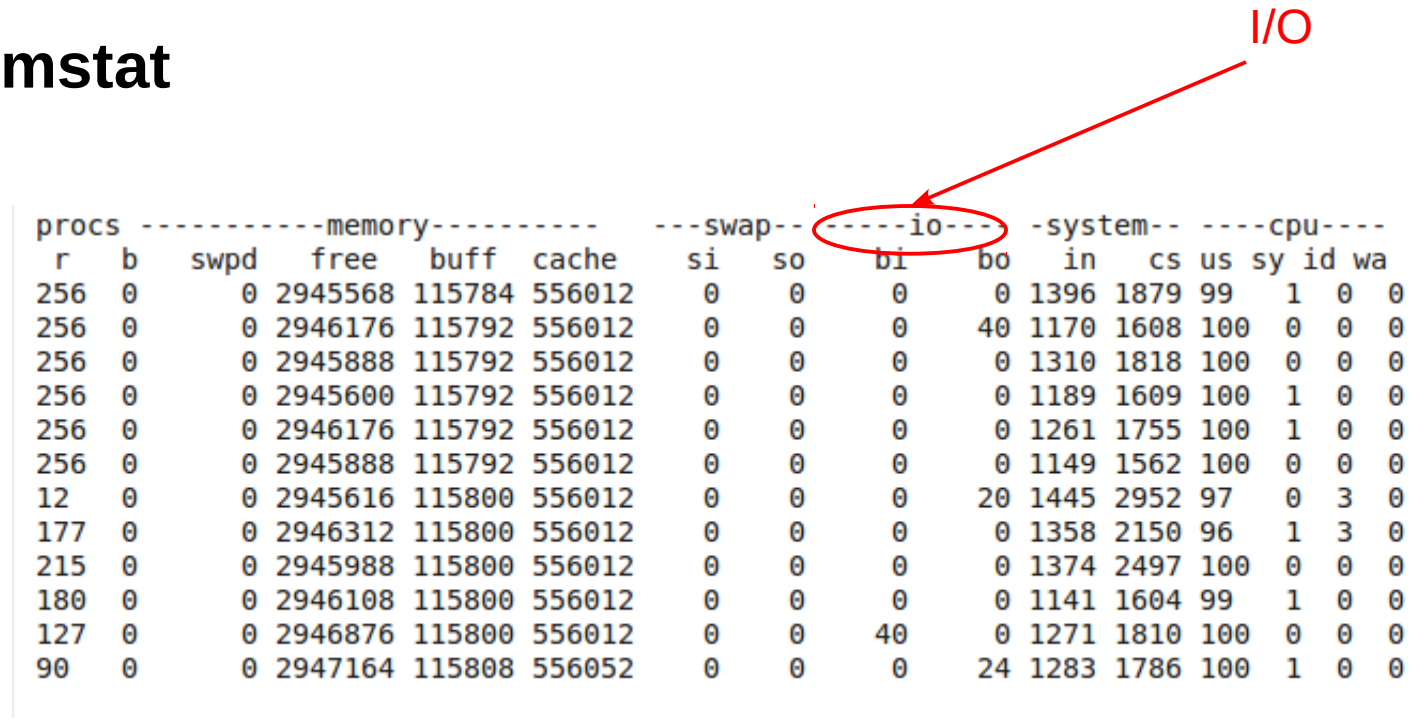
vmstat



Системный уровень

CPU

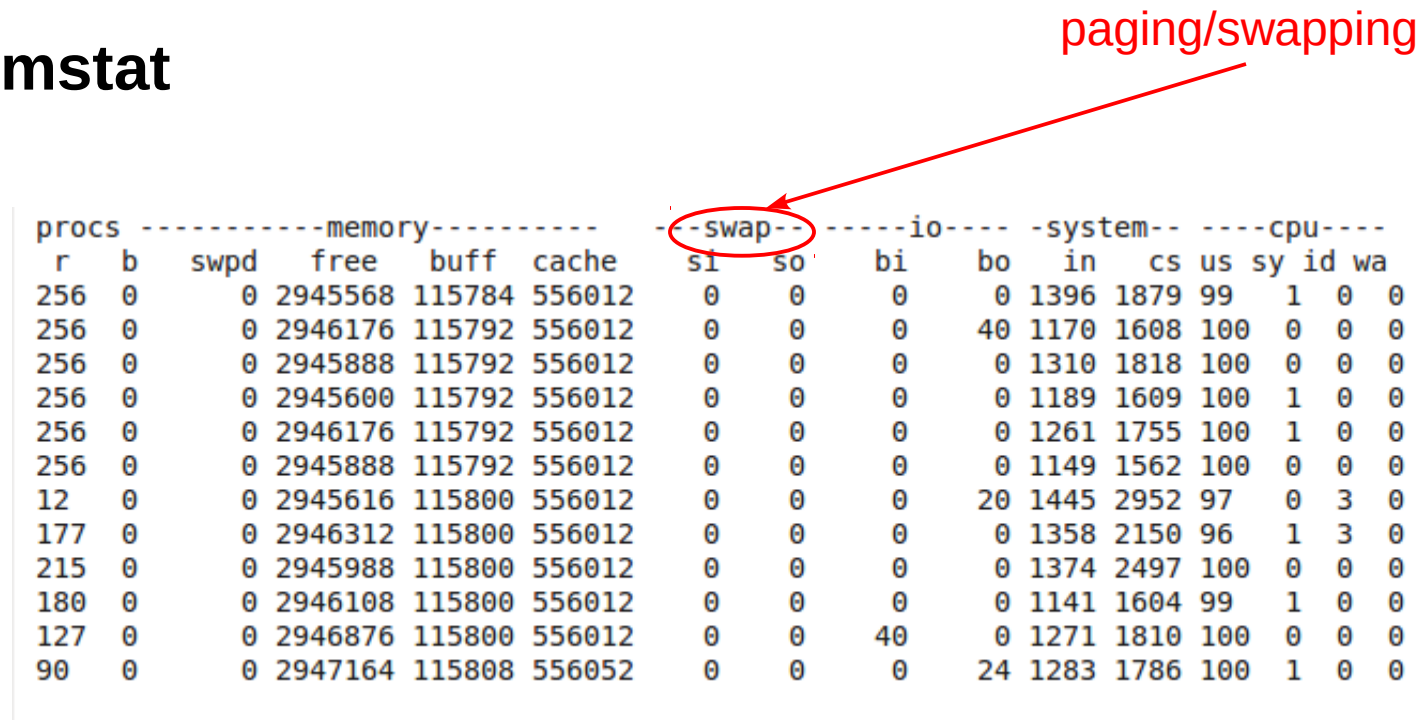
vmstat



Системный уровень

CPU

vmstat

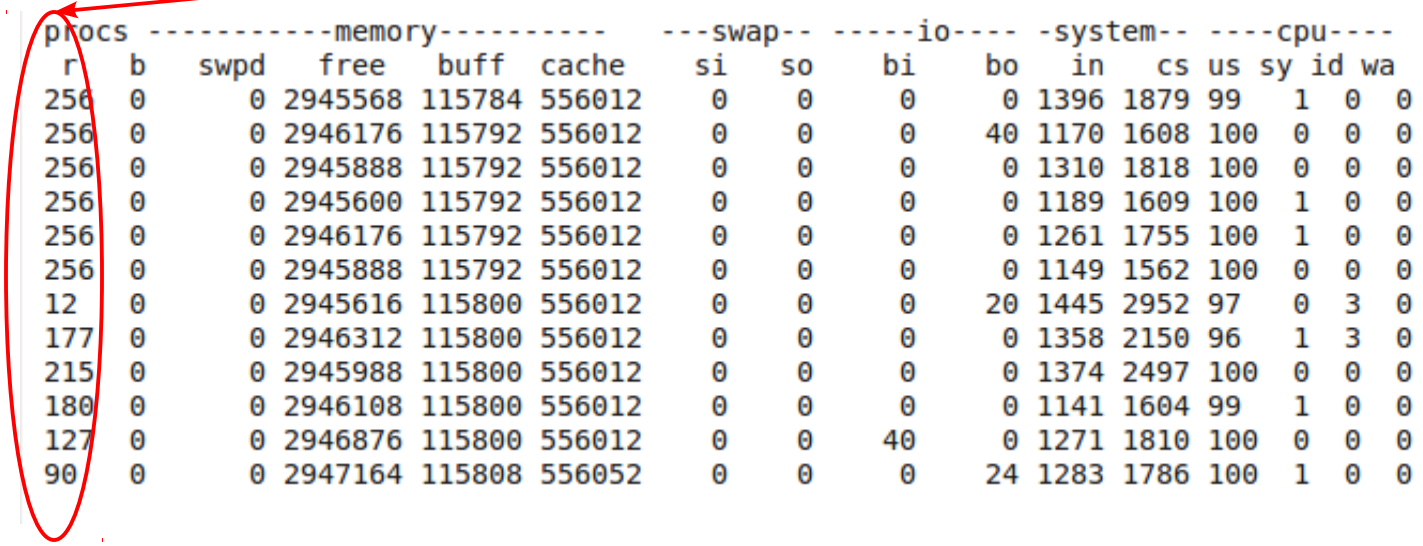


Системный уровень

CPU

vmstat

CPU scheduler run queue length



```
procs -----------memory----------   ---swap-- -----io---- -system-- ----cpu----
 r    b   swpd   free   buff  cache    si   so    bi    bo   in   cs us sy id wa
256   0      0 2945568 115784 556012    0    0     0     0 1396 1879 99   1  0  0
256   0      0 2946176 115792 556012    0    0     0    40 1170 1608 100  0  0  0
256   0      0 2945888 115792 556012    0    0     0     0 1310 1818 100  0  0  0
256   0      0 2945600 115792 556012    0    0     0     0 1189 1609 100  1  0  0
256   0      0 2946176 115792 556012    0    0     0     0 1261 1755 100  1  0  0
256   0      0 2945888 115792 556012    0    0     0     0 1149 1562 100  0  0  0
12    0      0 2945616 115800 556012    0    0     0    20 1445 2952 97   0  3  0
177   0      0 2946312 115800 556012    0    0     0     0 1358 2150 96   1  3  0
215   0      0 2945988 115800 556012    0    0     0     0 1374 2497 100  0  0  0
180   0      0 2946108 115800 556012    0    0     0     0 1141 1604 99   1  0  0
127   0      0 2946876 115800 556012    0    0    40     0 1271 1810 100  0  0  0
90    0      0 2947164 115808 556052    0    0     0    24 1283 1786 100  1  0  0
```

Системный уровень

CPU (симптомы)

- Высокая утилизация ядра ОС (system/kernel time)
  - I/O
  - конфликт ресурсов ОС
  - конфликт блокировок

Системный уровень

CPU (симптомы)

- Низкая утилизация CPU (high idle time)
  - I/O (+ high kernel time)
  - конфликт ресурсов ОС (+ high kernel time)
  - конфликт блокировок (+ high kernel time)
  - слабая параллелизация приложения

Системный уровень

CPU (симптомы)

- Высокая утилизация CPU (user time)
  - Неоптимальная архитектура приложения
  - Неправильное использование API
  - Неоптимальные настройки JVM (GC)
  - Неоптимизованный код приложения
  - (на этом месте можно переходить на нижний уровень)

Системный уровень

CPU (симптомы)

- **Voluntary context switching**
  - I/O
  - Блокировки (lock contention)
- **Involuntary context switching**
- **CPU scheduler run queue length**
  - Рекомендуемое значение не больше 4*<кол-во CPU>
  - Что делать если больше?
    - Увеличить количество CPU
    - Уменьшить количество тредов
    - Копать глубже и оптимизировать каждый тред

Уровень JVM

выбор JVM

- **Oracle (Sun) HotSpot**
  - Client VM
  - Server VM
- **Oracle (BEA) JRockit**
- **[эту JVM нам нельзя упоминать]**
- **[и эту тоже]**
- **[и эту]**

Уровень JVM

выбор GC

- **Oracle (Sun) HotSpot**
  - -XX:+UseSerialGC (по-умолчанию)
  - -XX:+ParallelGC
  - -XX:+ParallelOldGC
  - -XX:+ConcMarkSweepGC
  - -XX:+UseG1GC
- **Oracle (BEA) Jrockit**
  - -Xgc:{single,gen}{con,par}{con,par}

Уровень JVM

выбор размера кучи

- **Слишком мало?**
  - GC негде развернуться, видно по долгим и/или частым сборкам
  - PermGen!
- **Слишком много?**
  - Значит, у кого-то отобрали:
    - Кэши ФС
    - Стеки потоков
    - Direct-буфера
    - Нативные буфера

Уровень JVM

тюнинг

- **Подавляющее большинство общеизвестных настроек выставлено по умолчанию**
- **Остальной тюнинг симптоматический:**
  - Понимаем, что не работает
  - Понимаем/спрашиваем, как можно сделать быстрее
  - Печатаем -XX:+PrintFlagsFinal и находим нужный флаг
  - Гоняем функциональные тесты
  - Гоняем перформансные тесты
  - ???
  - PROFIT!

Уровень приложения

блокировки и синхронизация

- **Конфликты на блокировках видны в профиле!**
  - Даже банальный jstack может обозначить проблему
  - А если взять сотню jstack'ов подряд...
- **Всегда видна только самая дикая блокировка**
  - “Convoying”: остальные прячутся за ней
- **Лучше много маленьких, чем одна большая**
  - “lock striping”
- **Лучше lock-free, чем маленькая блокировка**
  - Большая предсказуемость
  - Ещё лучше отсвечивается в профиле!

Уровень приложения

API и алгоритмические проблемы

- **Алгоритмическая сложность**
  - Не зря на собеседованиях спрашивают про сложность, ой не зря
  - Компиляторы никогда не будут делать алгоритмические преобразования чужого кода
  - Ваш супероптимизированный bogosort внутри останется bogosort'ом
- **Пользуйтесь правильным API**
  - Вам точно нужен List, а не Collection? Точно List, а не Set? Точно SortedSet, а не Set?
  - Отходите от умолчаний, только когда очень нужно
  - Exceptions
- **Думаем заранее о concurrency**
  - Immutability
  - Бьём себя по рукам за synchronized {}
  - Бьём себя по рукам за Collections.synchronized...{}


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

Q/A

Backup

Теория

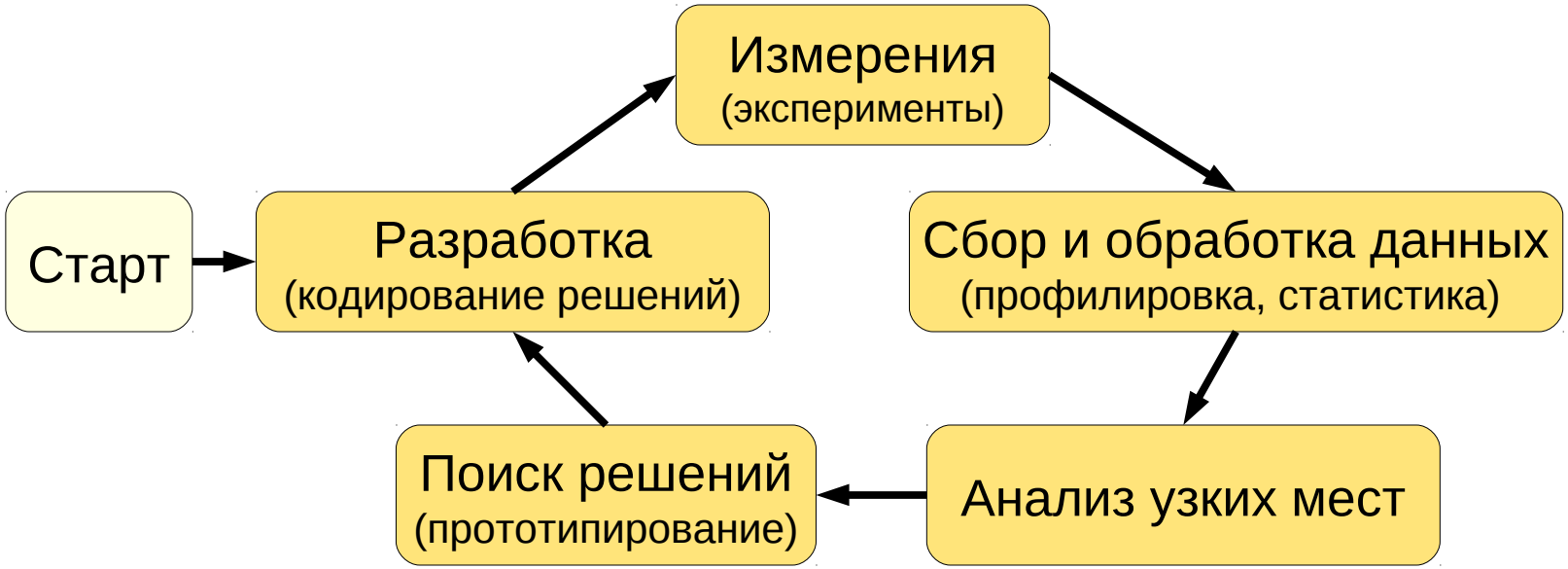
Когда закон Амдала может не работать

Composability

- **Предположим, есть функциональные блоки A и B**
  - Есть ли разница при последовательном (...) и параллельном (||) исполнении?
- **Общая функциональность:**
  - Functionality(A ... B) = Functionality(A || B)
  - “Black Box”: поведение одинаково
- **Общая производительность:**
  - **Performance(A ... B) ? Performance(A || B)**
  - **Про это сказать толком ничего нельзя**
    - A и B соревнуются за аппаратные ресурсы
    - Возможно, > (эффективный размер кеша меньше)
    - Возможно, < (два потока на HT машине)
    - Возможно, = (нет конфликтов)

Performance Engineering

итеративный подход



Важно:

- Одно изменение за цикл!
- Документировать все изменения

JIT

для любопытных

Как получить ассемблерный код метода?

- Обычным дебаггером ;)
- JVMTI
- -XX:+PrintAssembly
  - http://wikis.sun.com/display/HotSpotInternals/PrintAssembly


ORACLE®