



ORACLE®

О чём молчит профайлер

Presenter: Александр Отенко
CMTS, Performance Engineering
Oracle

Содержание

- С чего начать
- Куда делось CPU?
 - Не то же самое, что и “Куда делось время?”
- Куда делась память?
- Почему CPU никуда не делось?
 - Диагностировать «неиспользование» ресурса труднее, чем использование – строка кода, которая «не использует» ресурс, отсутствует

Профиль – это свидетель, а не прокурор

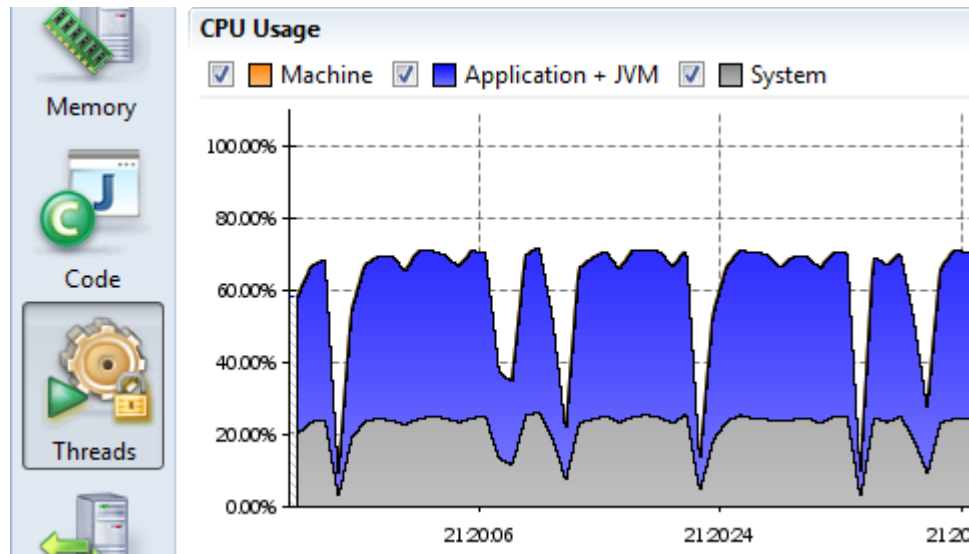
- Нужно доказать состоятельность показаний
 - Все ли улики сходятся?
 - Насколько надёжные выводы?
- Собрать важные улики
- Отбросить не важные
- Компетентность не купишь
- Нужна некая «теория происходящего»

Бюджет CPU

- Ваш профайлер так умеет?
- Thread.run = 7.5 sec (100%)
 - Из них JDBC Query = 150ms (2%)
- Следовательно, 98% времени отклика – это эффективность Java кода
 - Правда? Счастливчик! Должно масштабироваться
 - (Профайлер промолчал, что это разные миллисекунды)

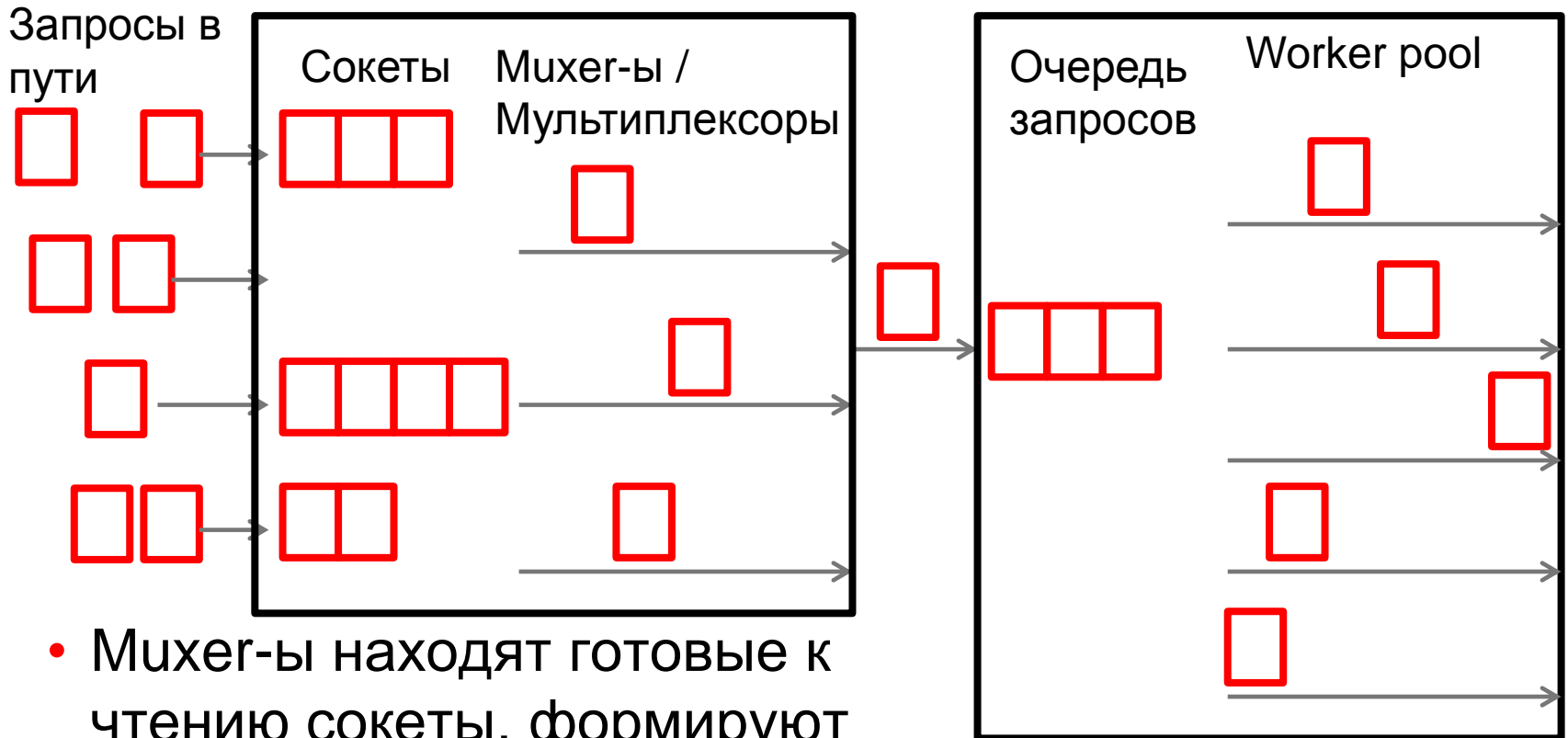
Куда делось CPU?

- А вот так ваш профайлер умеет?



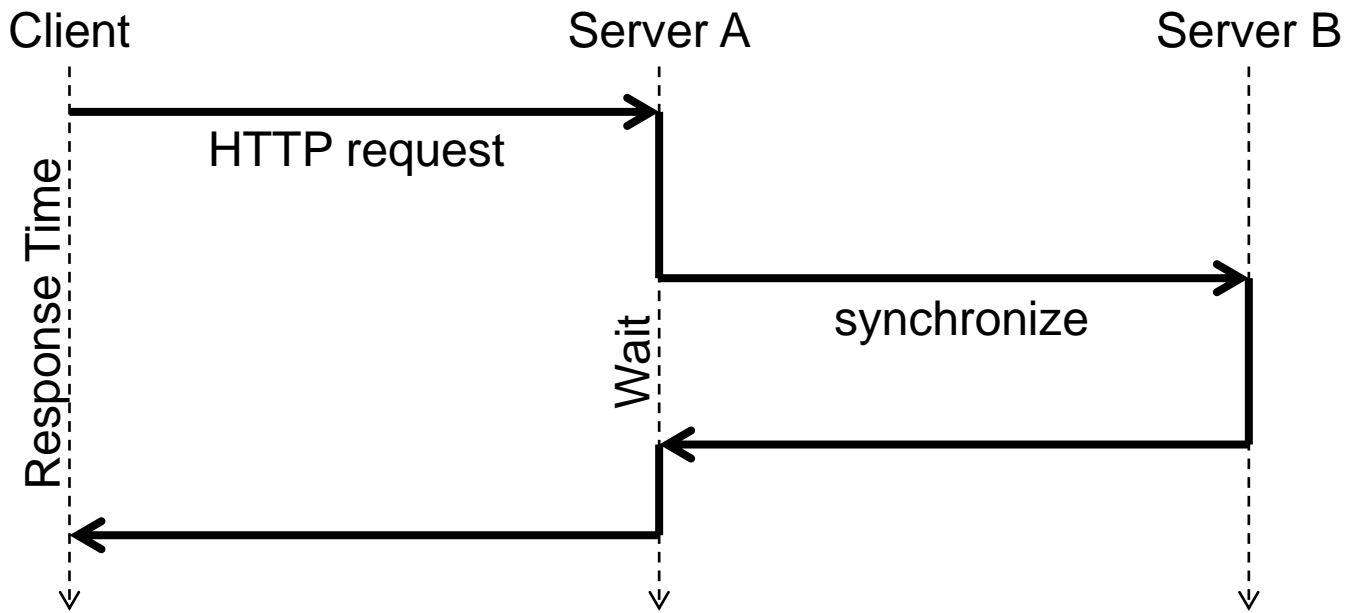
- Кто знает, что за серенькое «остальное CPU»?
 - А как оно будет выглядеть в профиле?
 - А как выглядит в профиле это «беленькое» CPU?

Асинхронная обработка запросов



- Миксер-ы находят готовые к чтению сокеты, формируют запрос, передают в Worker pool
- Worker-ы обрабатывают запросы (и отправляют ответ)

Бюджет CPU



- Расследуем wait?
 - Никак нет! (об этом позже)
- Гипотеза: одно соединение – один поток
 - Улики есть?

Бюджет CPU

Method Details

Predecessor Successor

Trace	Sample Count
weblogic.socket.NIOSocketMuxer.processSockets()	20832
weblogic.socket.SocketMuxer.readReadySocket(MuxableSocket, SocketInfo, long)	12578
weblogic.socket.SocketMuxer.readReadySocketOnce(MuxableSocket, SocketInfo)	12563
weblogic.socket.BaseAbstractMuxableSocket.dispatch()	5881
weblogic.servlet.internal.MuxableSocketHTTP.dispatch()	5610
jrockit.vm.Locks.monitorEnter(Object)	7200
sun.nio.ch.SelectorImpl.select()	745
weblogic.socket.NIOSocketMuxer.fillSelectedList()	181
jrockit.vm.Locks.monitorExit(Object)	72

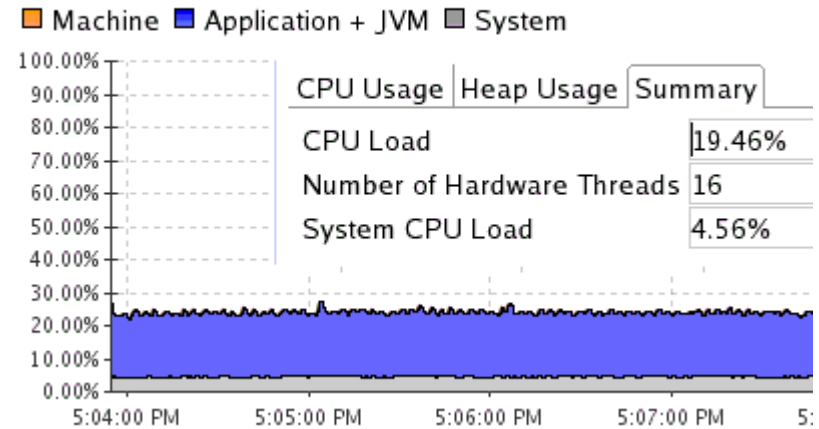
Overview Hot Methods Exceptions Code Generation

- *Честный* профайлер соберёт samples пропорционально использованию CPU
- *На этом основании* заключаем, что два вида dispatch делят работу Muxer примерно пополам

Улики

Thread	Percentage
ExecuteThread: '0' for queue: 'webloc	9.06
ExecuteThread: '1' for queue: 'webloc	9.66
ExecuteThread: '2' for queue: 'webloc	10.23
ExecuteThread: '3' for queue: 'webloc	10.68

CPU Usage

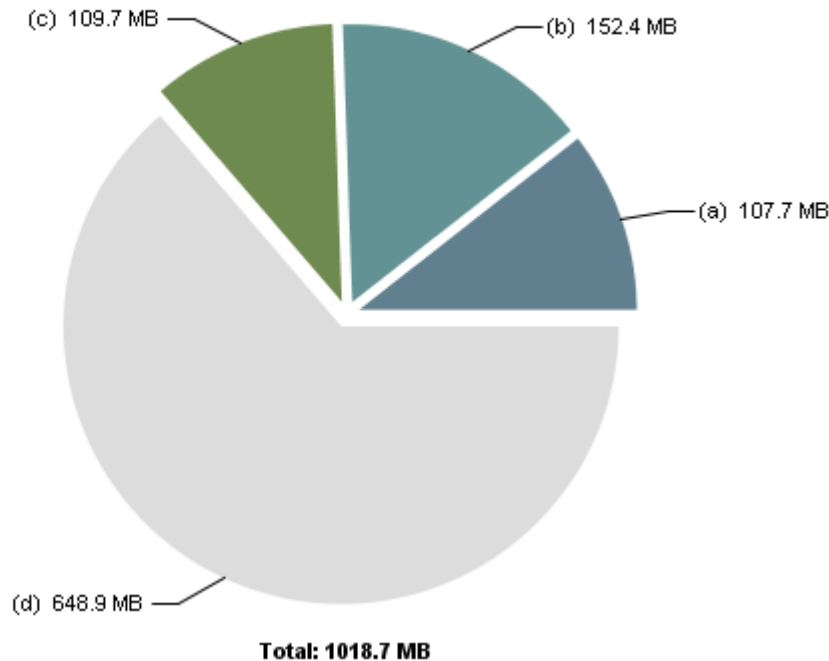


- Миксер съели 40% user CPU
- User CPU занимает 19.46% от 16 ядер
 - из которых 40% – это Миксер
- System CPU занимает 4.56% от 16 ядер
 - В основном, I/O из Миксер-а, которое не попало в CPU samples
- Вполне возможно Миксер тратит
 - $0.5 * (40% * 19.46% + 4.56%) * 16 = 0.98$ ядра CPU
 - на соединение synchronization

Куда делась память?

- Ваш профайлер, должно быть, такое умеет?

▼ Overview

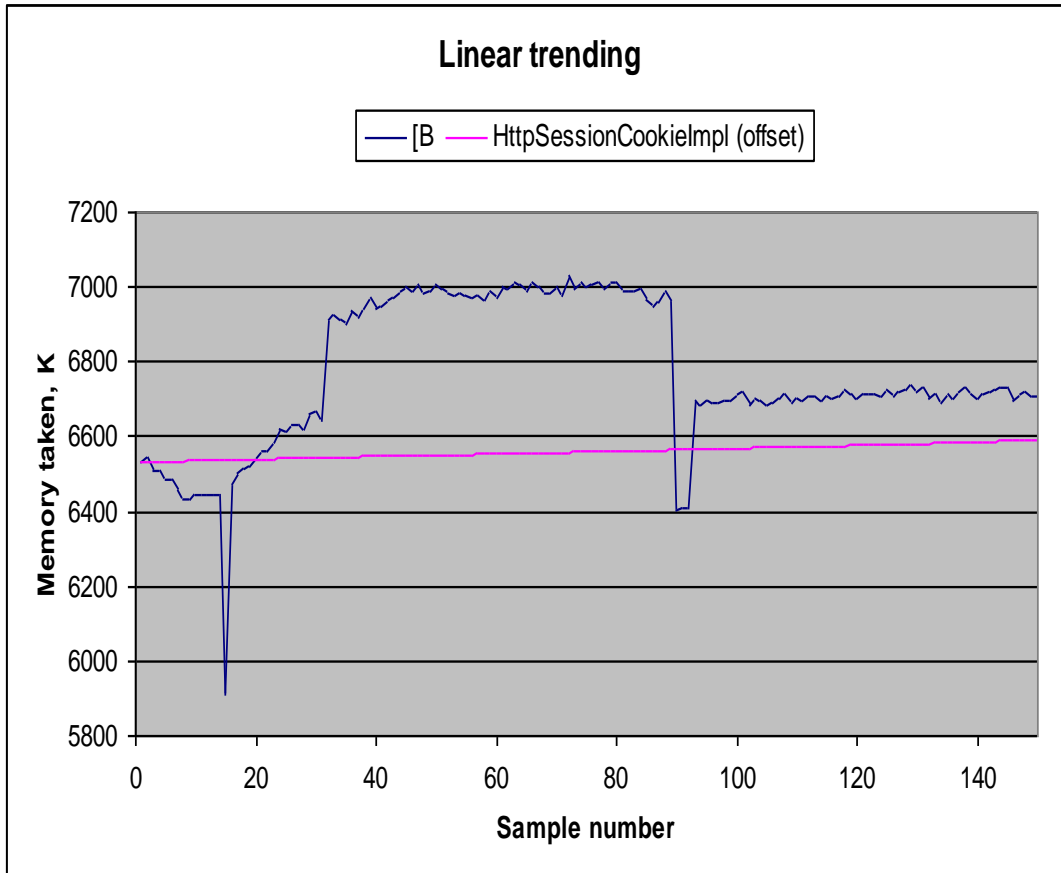


- a) Replication Manager
- b) Replicated Session Data
- c) Prepared Statements
- d) Remainder

Проблема концептуальная

- Проблема не в реализации
- Один heap dump указывает только *размер* объектов
- Но где размер *поменялся*?
 - (И почему этот профайлер считает, что он поменялся именно там?)

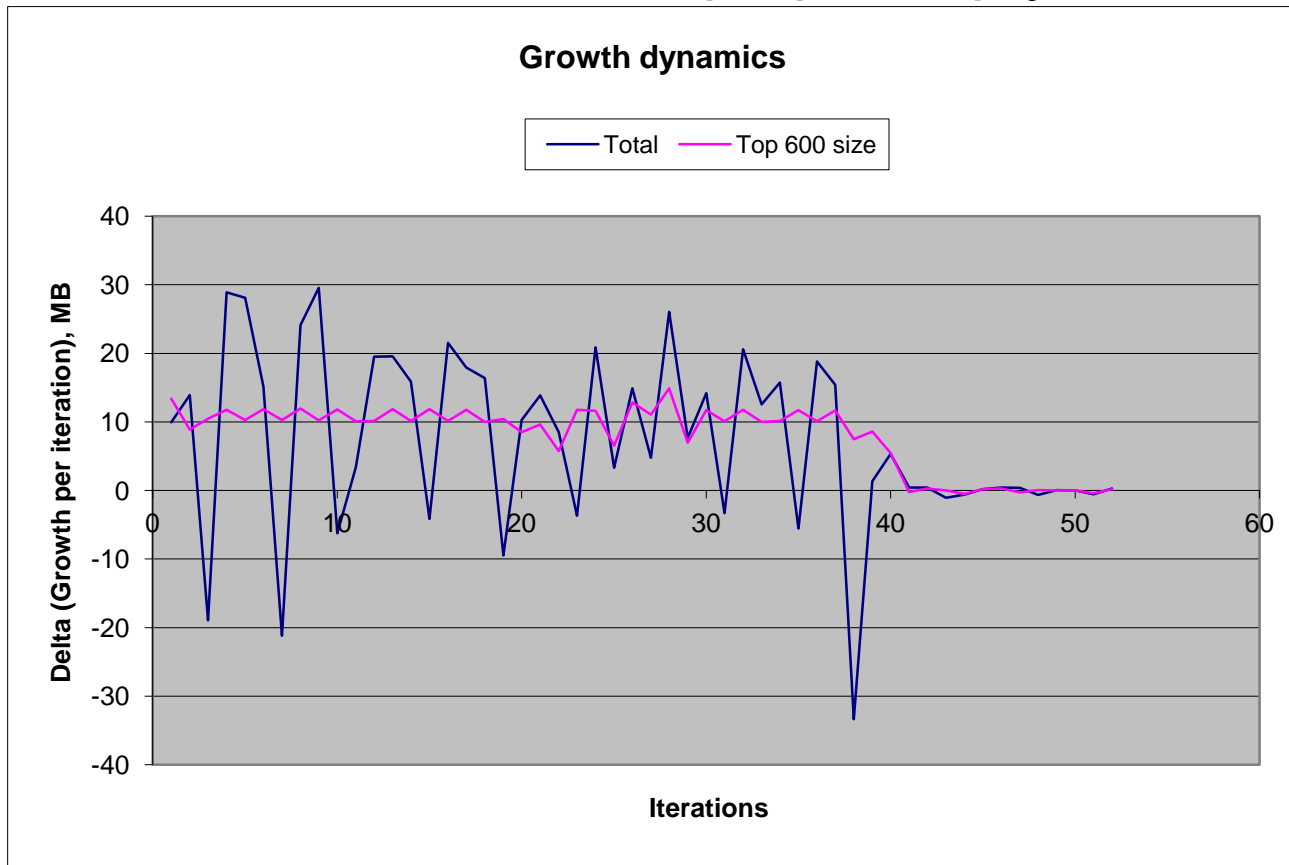
Два дампа?



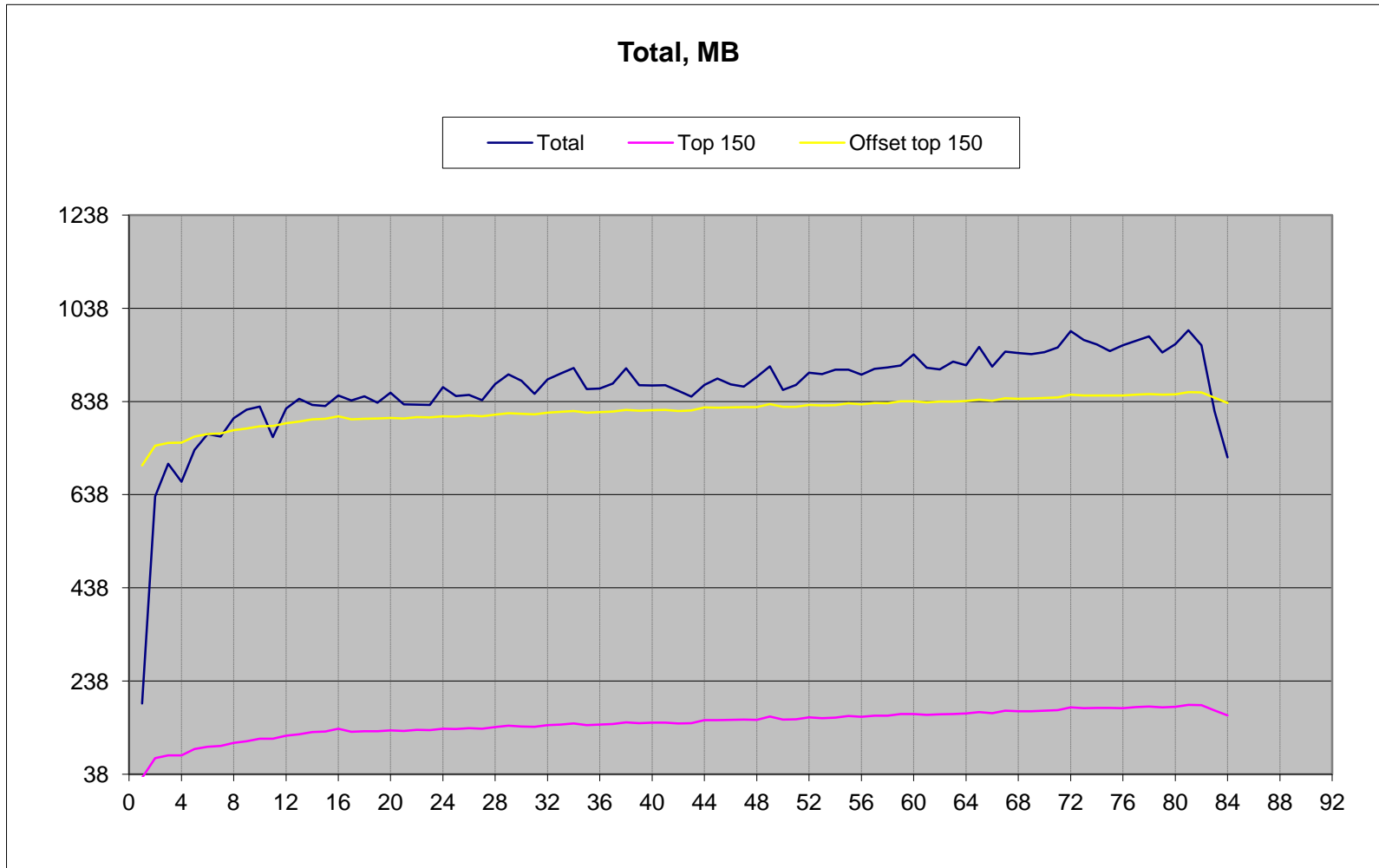
- Два дампа укажут на разницу в размере
- Но всё равно должно немножко повезти
 - Растёт ли [B на самом деле?
 - Что, если получить дампы в неподходящее время?

Гипотеза о тенденции

- Ага, а вот такое ваш профайлер умеет?



Что нам на самом деле нужно



Память

- Memory leak analyzer в JRMС это умеет
 - (хотя не умеет строить граф)
- Ищите классы, уникальные для приложения
 - Какой смысл смотреть на String? HashMap\$Entry?
- Они могут быть крошечные, но иметь тенденцию, совпадающую с ростом потребления памяти
 - Mem Leak вынюхает – были случаи по несколько КБ за итерацию
 - Лучше всего – маленькие и несколько
 - Мало экземпляров = меньше кода просматривать

Почему CPU никуда не делось?

- А вот такое ваш профайлер умеет?
- Отклик = CPU% * CPU count / Throughput + Latency

CPU utilization	Measured TPS	Average RT (us)	Projected TPS (User#/ART)	Time on CPU per request (us)
65.11%	123875	1056.7	151415	126








- «Measured TPS» против «Projected TPS» – погрешность теста (в данном случае очень даже ого-го!)
- Запрос почти 10x времени «не на CPU» против «на CPU» (и это мы ещё профайлер не открывали)

Latency

- А ваш профайлер такое умеет?

Event Types

Filter Column Show Only Operative Set

Event Type	Duration	Count
 Java Wait	1 h 59 m 49 s 483.807 ms	27,936
 Socket Read	25 m 03 s 199.690 ms	19,650
 Java Sleep	6 m 30 s 74.579 ms	13
 Thread Parked	1 m 03 s 927.323 ms	1,870
 Java Blocked	9 s 227.428 ms	126
 Socket Write	5 s 956.906 ms	57
 Object Allocation in New TLA	4 s 263.636 ms	91

Overview Log Graph Threads Traces Histogram

Классификация Latency

- А вот так?

Histogram

Group By:

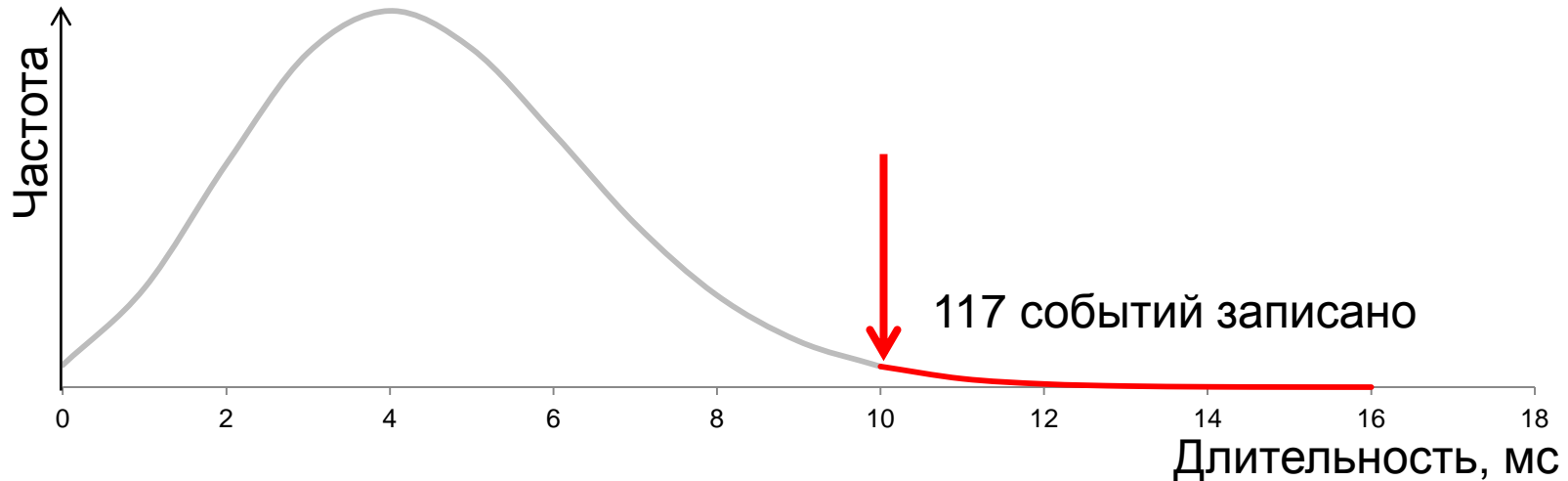
Group By:	Total	Average	Count
⊖ java.lang.String	8 s 736.184 ms	74.668 ms	117
⊖ weblogic.servlet.internal.session.ReplicatedSessionData	453.186 ms	56.648 ms	8
⊖ weblogic.rjvm.MsgAbbrevJVMConnection\$WritingState	38.058 ms	38.058 ms	1

Traces

Trace	Sample Count	Total Duration
-------	--------------	----------------

- Всего 117 захватов lock за 5 минут? Серьёзно?
- Средняя latency порядка десятков ms?

Запись Latency



- Профайлер записывает только latencies, превышающие некоторый порог
 - Но никому об этом не сказал
 - Точнее, может сказать, да спросить нужно
- Имеет ли смысл оптимизация до меньшего?
 - А 117 имело смысл?

Запись Latency – как проверить

- Thread count * Recording duration = on-CPU-time + off-CPU-time
- Latency = Thread count * Recording duration – CPU% * CPU count * Recording duration

Thread count	Recording duration	CPU %	CPU count	Recorded Latency	Missing Latency
47	6 min	19.64%	24	153 min	>100 min

- Таким образом, любая проблема может стать проблемой номер один
 - Понизить порог и перезаписать

Latency

Event Types

Filter Column Show Only Operative Set

Event Type	Duration	Count
Java Blocked	6 h 54 m 09 s 396.217 ms	1,564,919
Java Wait	2 h 30 m 30 s 823.902 ms	662,498
Socket Read	1 h 08 m 16 s 98.837 ms	416,007
Java Sleep	6 m 01 s 115.489 ms	12
Thread Parked	2 m 27 s 266.903 ms	2,823
Socket Write	57 s 276.333 ms	1,414
Object Allocation in New TLA	46 s 627.783 ms	761

Overview Log Graph Threads Traces Histogram


Thread count	Recording duration	CPU %	CPU count	Recorded Latency	Missing Latency
200	400 sec	27.02%	24	38592 sec	38814 sec

- Понижение порога – не панацея
 - Но хоть диагностика более репрезентативная
- Например, вот здесь отсутствующие latencies меньше 1 ms

Знай задачу





Histogram







Group By: Show Only C

Group By:	Total	Average
 java.util.LinkedList	4 m 57 s 696.862	7.148 ms

← //

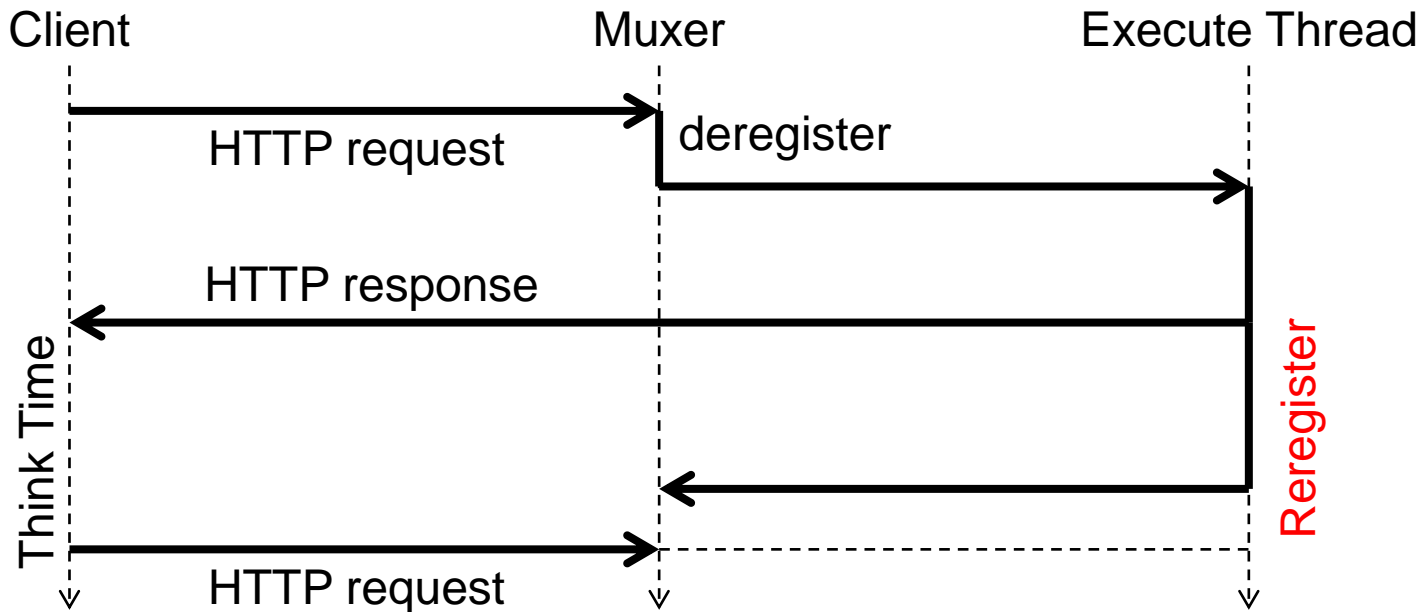
Traces

Trace	Sample Count	To
▷  sun.nio.ch.EPollArrayWrapper.setInterest(SelChImpl, int)	41,395	4 m 55 s
▷  sun.nio.ch.EPollArrayWrapper.updateRegistrations()	240	1 s
▷  sun.nio.ch.EPollArrayWrapper.add(SelChImpl)	10	
▷  sun.nio.ch.EPollArrayWrapper.release(SelChImpl)	3	

 Overview  Log  Graph  Threads  Traces  Histogram

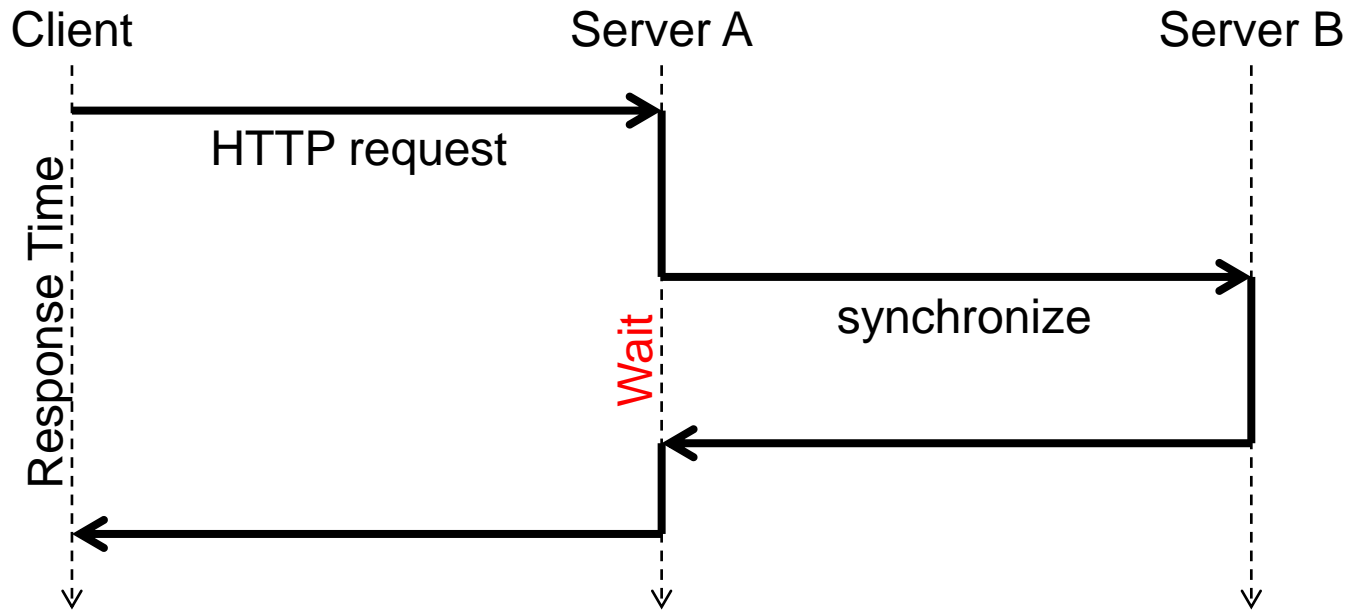
- Горячий Lock! Чинить немедленно!
 - Да ну?

Latency



- На самом деле reregistering interest может занять всё Think Time, и это всё равно никак не скажется на производительности

Latency



- Таким же образом и эффективность wait не имеет значения в этом случае



Q & A

Latency

- А вот такое ваш профайлер умеет?:
- Отклик = CPU% * CPU count / Throughput + Latency

	Request A	Request B	Request C	Request D
Mix	90%	5%	2.5%	2.5%
Response Time (ms)	0.987	1.596	1.714	1.83
Weighted RT	0.8883	0.0798	0.04285	0.04575

CPU utilization	Measured TPS	Average RT (sum weighted)	Projected TPS (User#/ART)	Time on CPU per request (ms)
65.11%	123875	1.0567	151415	0.126

- Measured против Projected TPS – погрешность теста (в данном случае очень даже ого-го!)
- Запрос почти 10x времени не на CPU против на CPU (и это мы ещё профайлер не открывали)