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

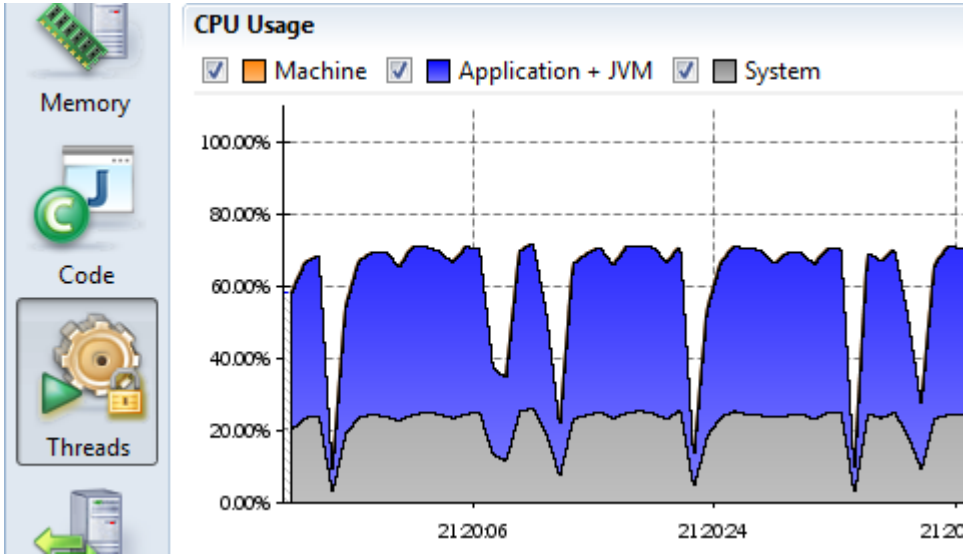
- Нужно доказать состоятельность показаний
  - Все ли улики сходятся?
  - Насколько надёжные выводы?
- Собрать важные улики
- Отбросить не важные
- Компетентность не купишь
- Нужна некая «теория происходящего»

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)

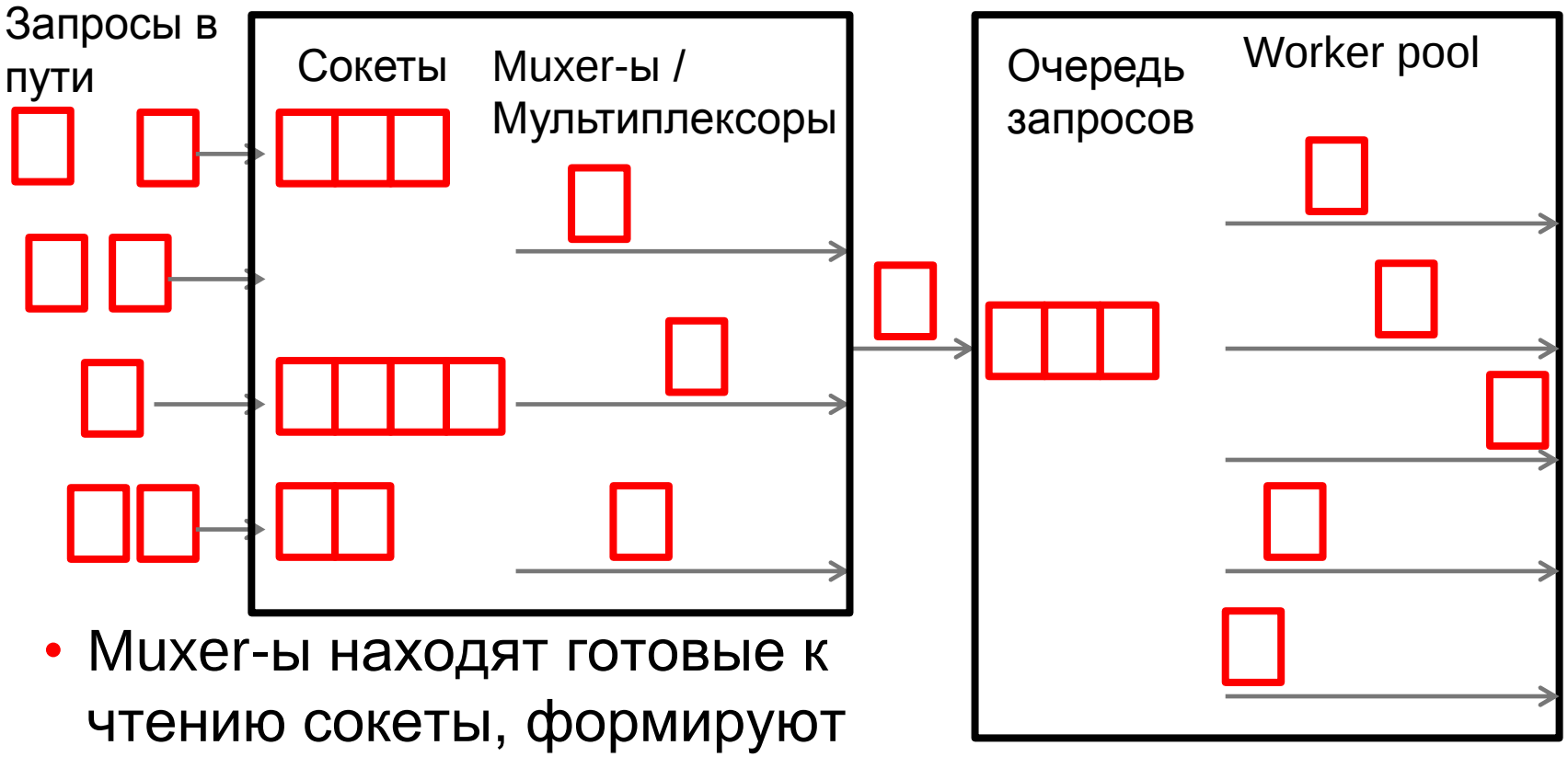
Куда делось CPU?

- А вот так ваш профайлер умеет?



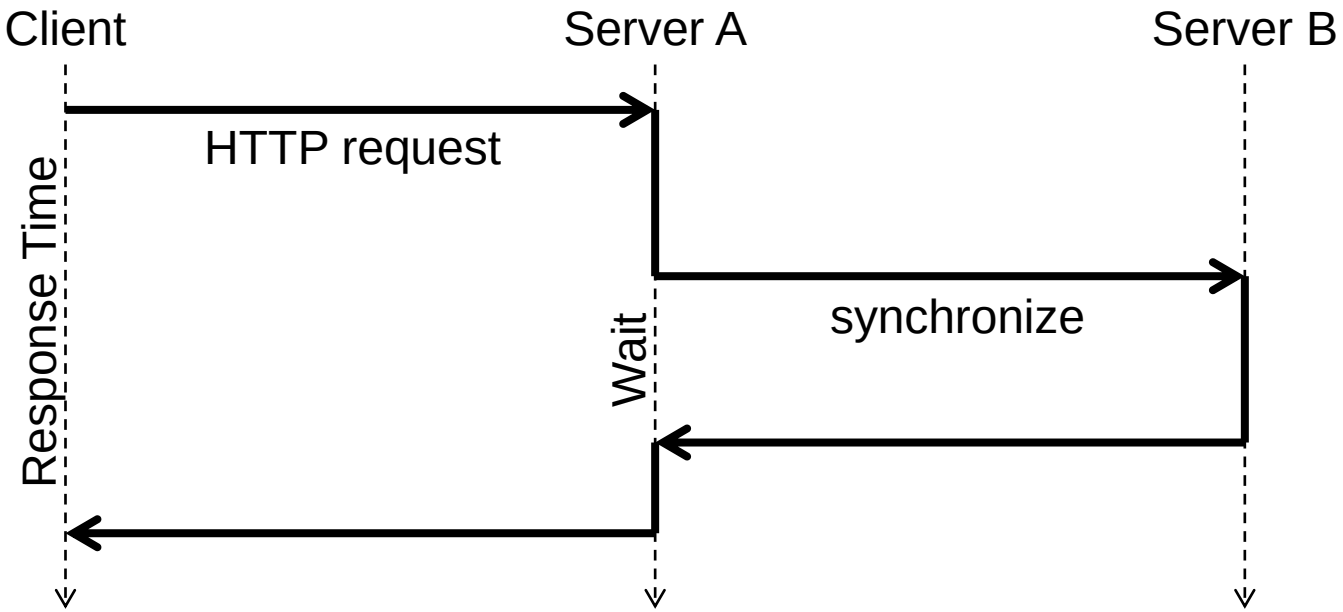
- Кто знает, что за серенькое «остальное CPU»?
  - А как оно будет выглядеть в профиле?
  - А как выглядит в профиле это «*беленькое*» CPU?

Асинхронная обработка запросов



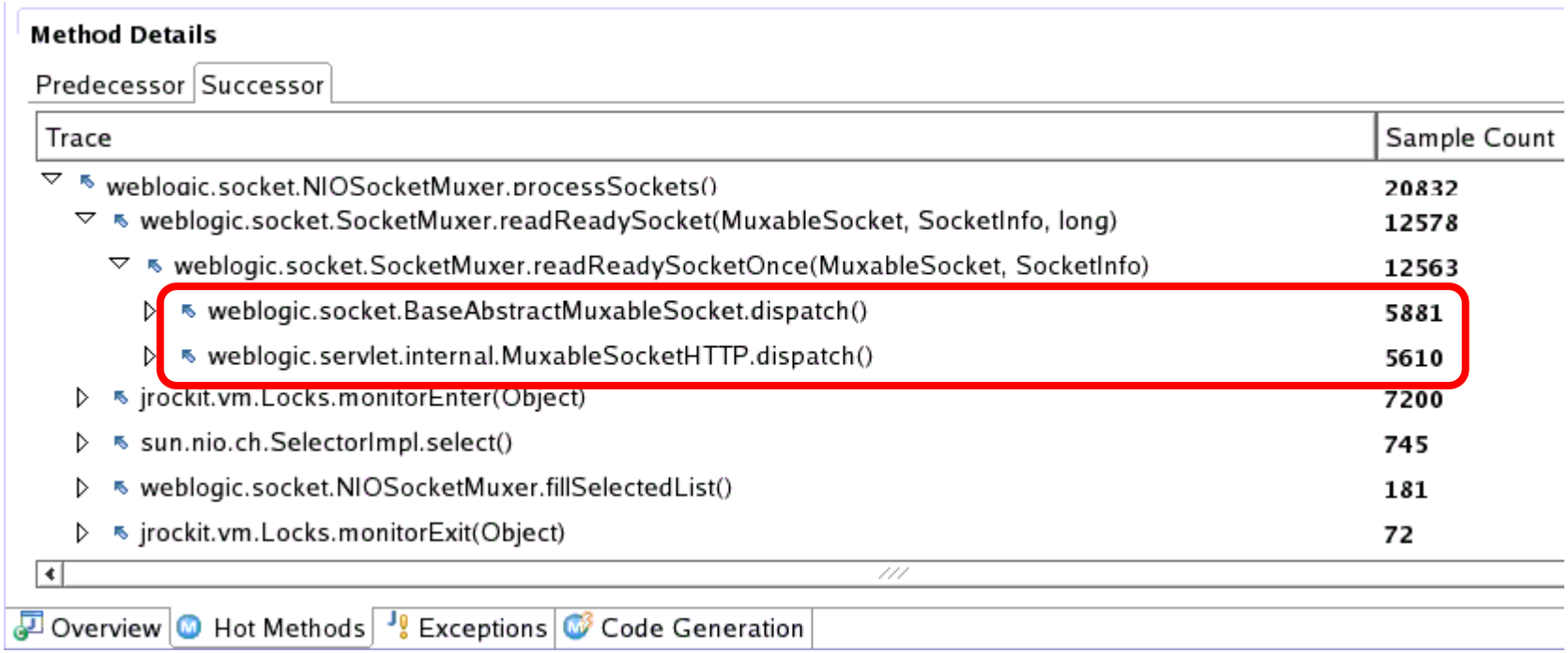
- Muxer-ы находят готовые к чтению сокеты, формируют запрос, передают в Worker pool
- Worker-ы обрабатывают запросы (и отправляют ответ)

Бюджет CPU



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

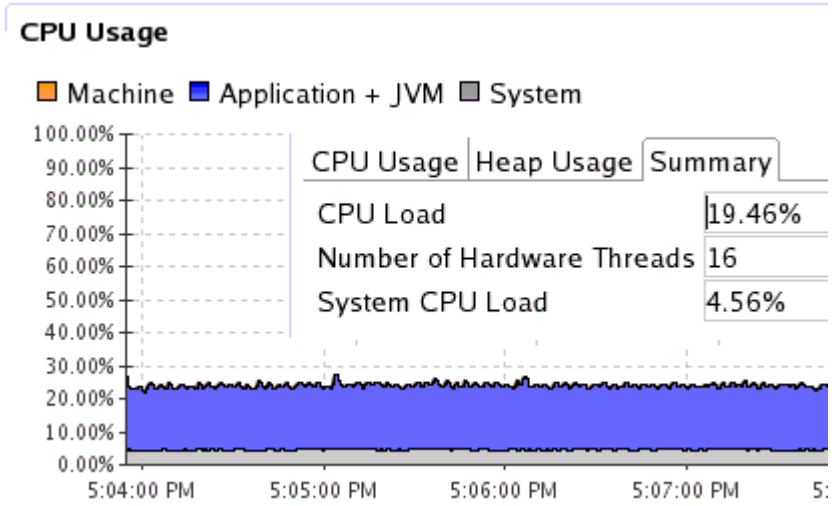
Бюджет CPU



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

Улики

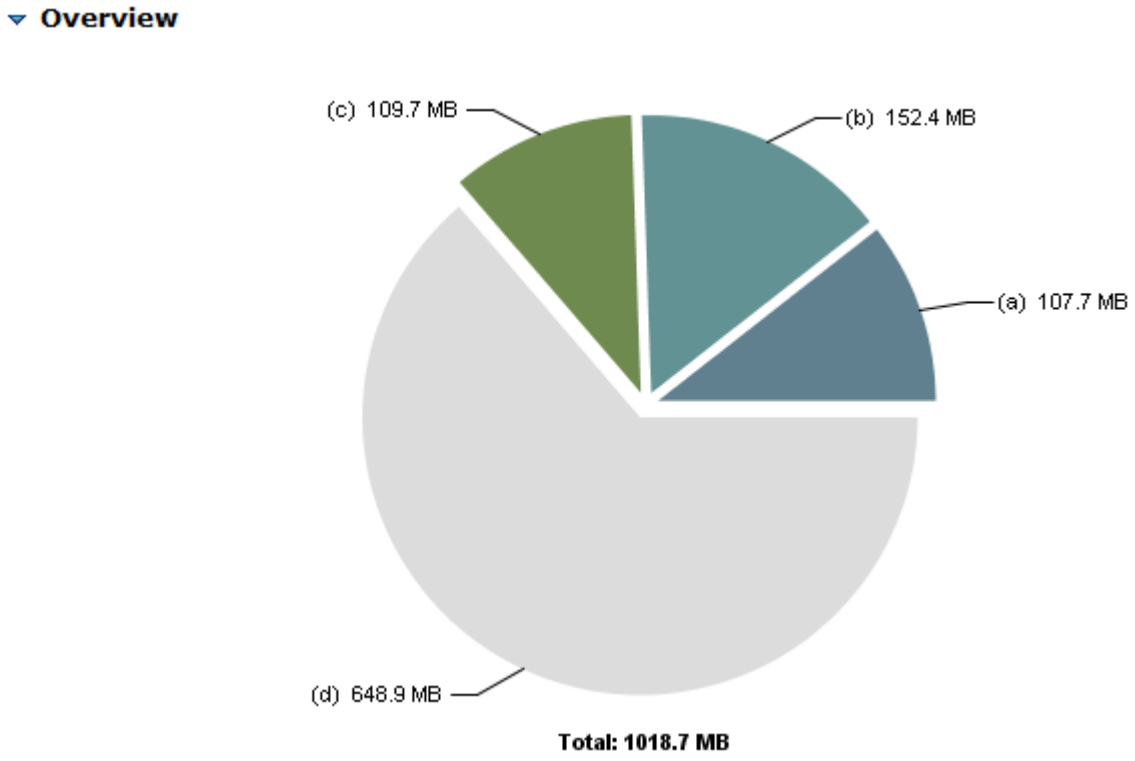
Thread	Percentag
ExecuteThread: '0' for queue: 'weblc	9.06
ExecuteThread: '1' for queue: 'weblc	9.66
ExecuteThread: '2' for queue: 'weblc	10.23
ExecuteThread: '3' for queue: 'weblc	10.68



- Muxer съели 40% user CPU
- User CPU занимает 19.46% от 16 ядер
  - из которых 40% – это Muxer
- System CPU занимает 4.56% от 16 ядер
  - В основном, I/O из Muxer-а, которое не попало в CPU samples
- Вполне возможно Muxer 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?



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

Проблема концептуальная

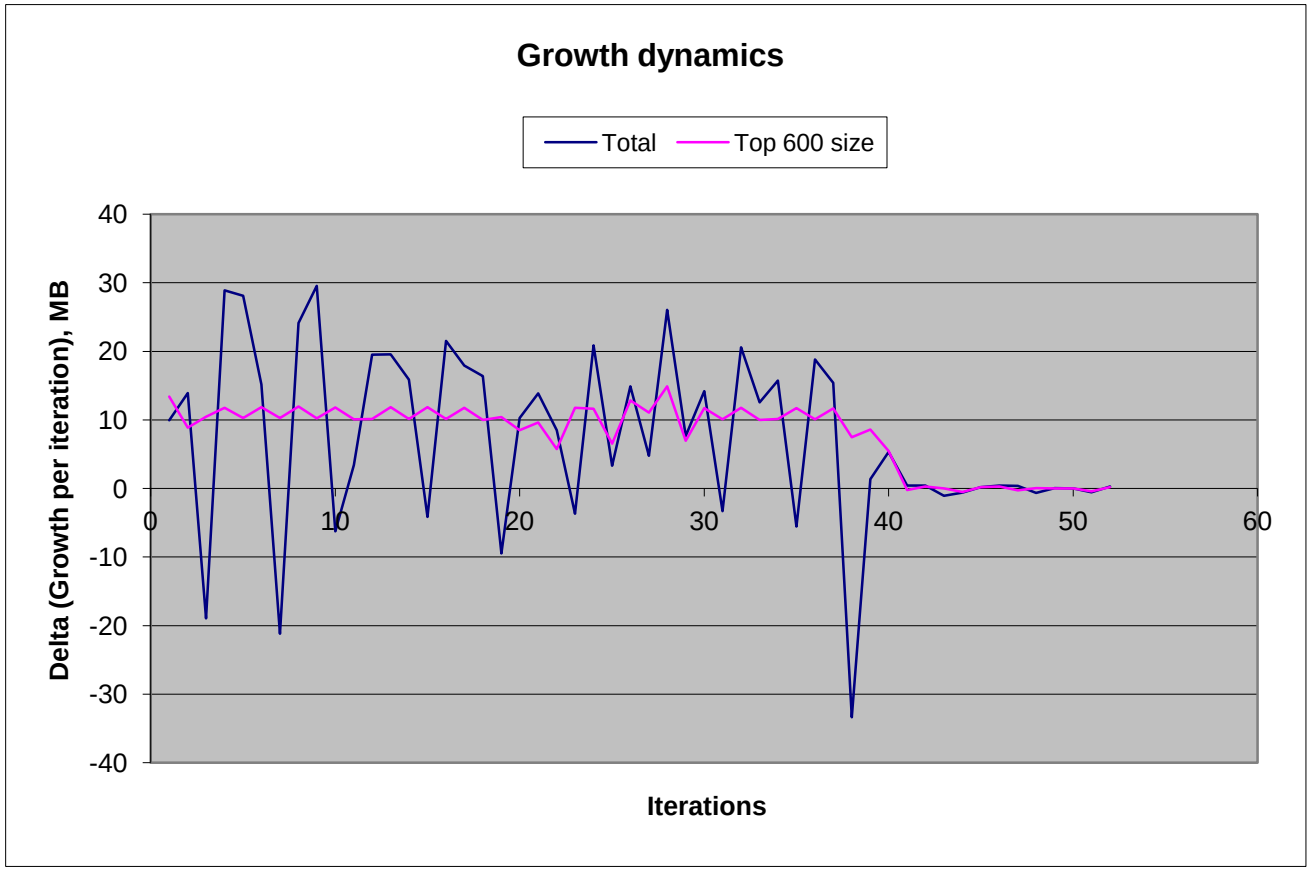
- Проблема не в реализации
- Один heap dump указывает только *размер* объектов
- Но где размер *поменялся*?
  - (И почему этот профайлер считает, что он поменялся именно там?)

Два дампа?

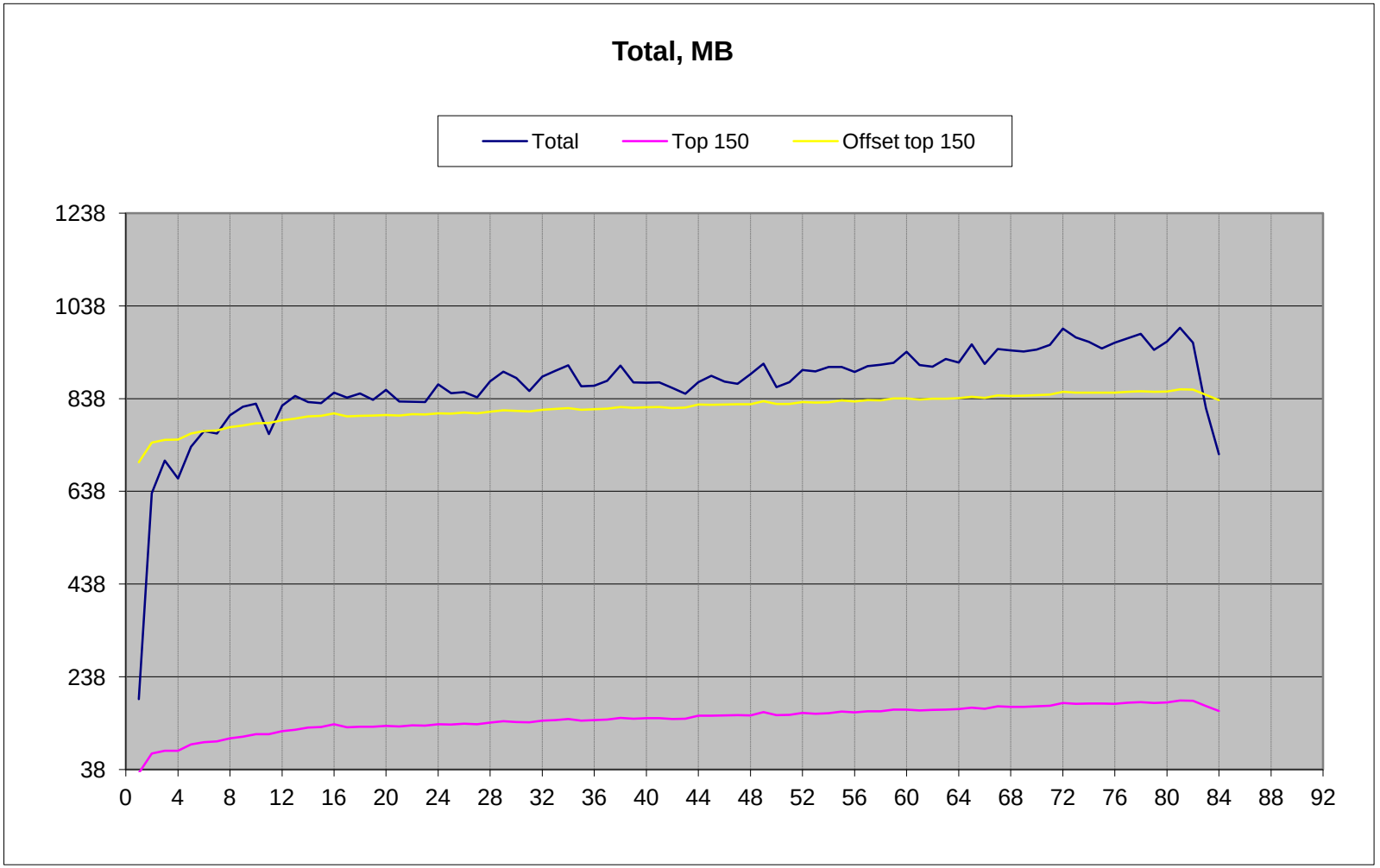
- Два дампа укажут на разницу в размере
- Но всё равно должно немножко повезти
  - Растёт ли [B *на самом деле*?
  - Что, если получить дампы в неподходящее время?

Гипотеза о тенденции

- Ага, а вот такое ваш профайлер умеет?



Что нам на самом деле нужно



Память

- Memory leak analyzer в JRMC 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 инстансов = меньше кода просматривать

Почему CPU никуда не делось?

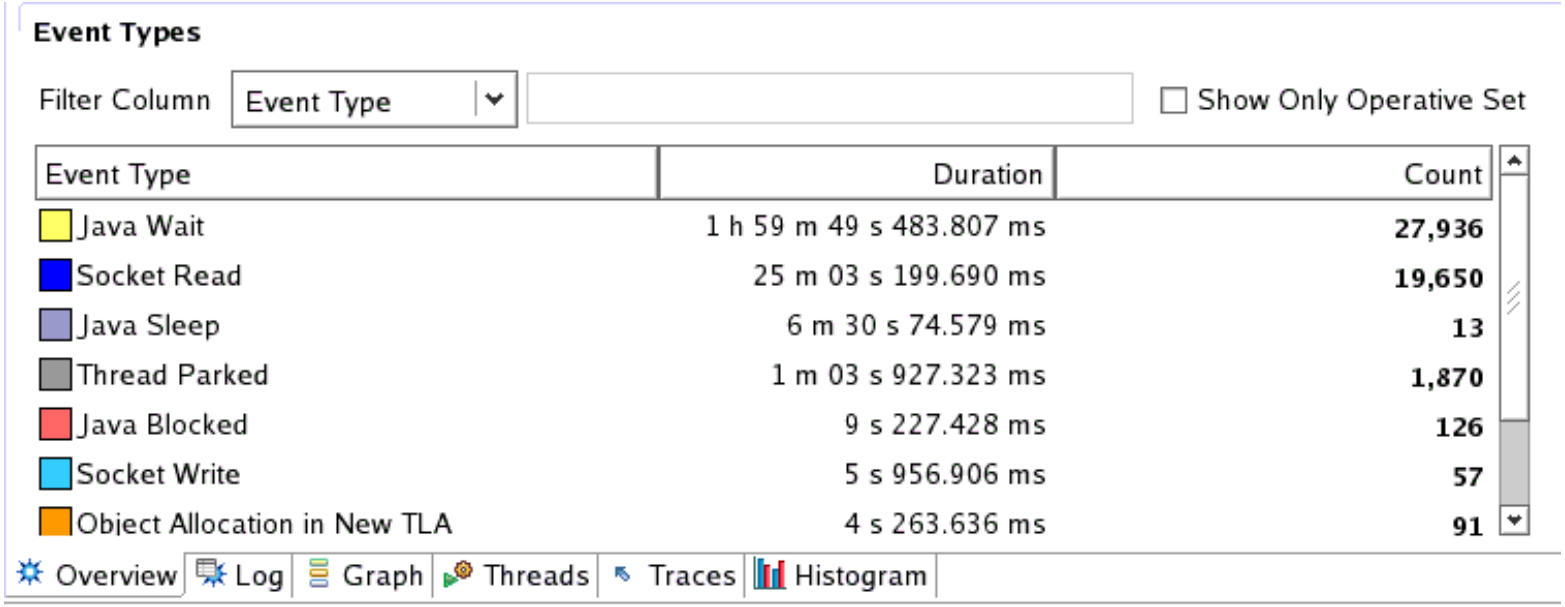
- А вот такое ваш профайлер умеет?
- Отклик = CPU% * CPU count / Throughput + Latency

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)

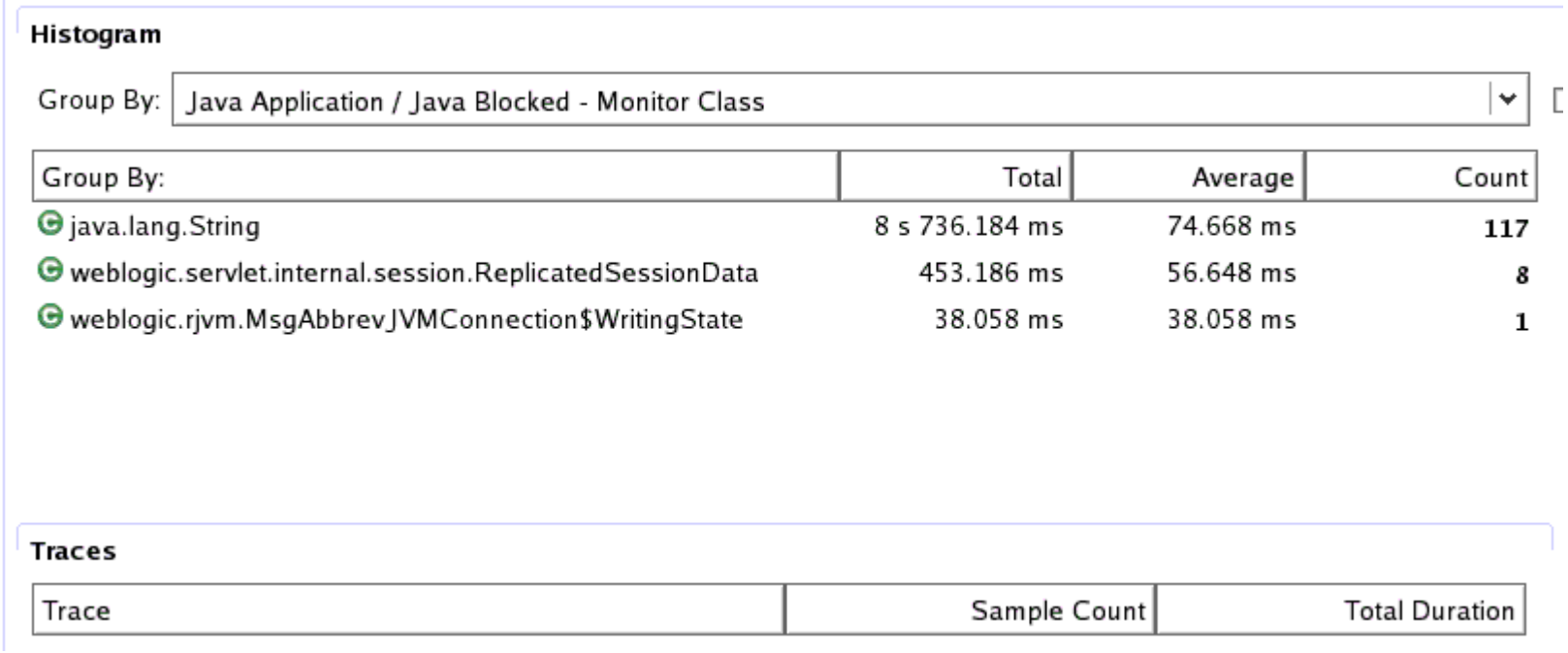
Latency

- А ваш профайлер такое умеет?



Классификация Latency

- А вот так?



Histogram

Group By: Java Application / Java Blocked - Monitor Class

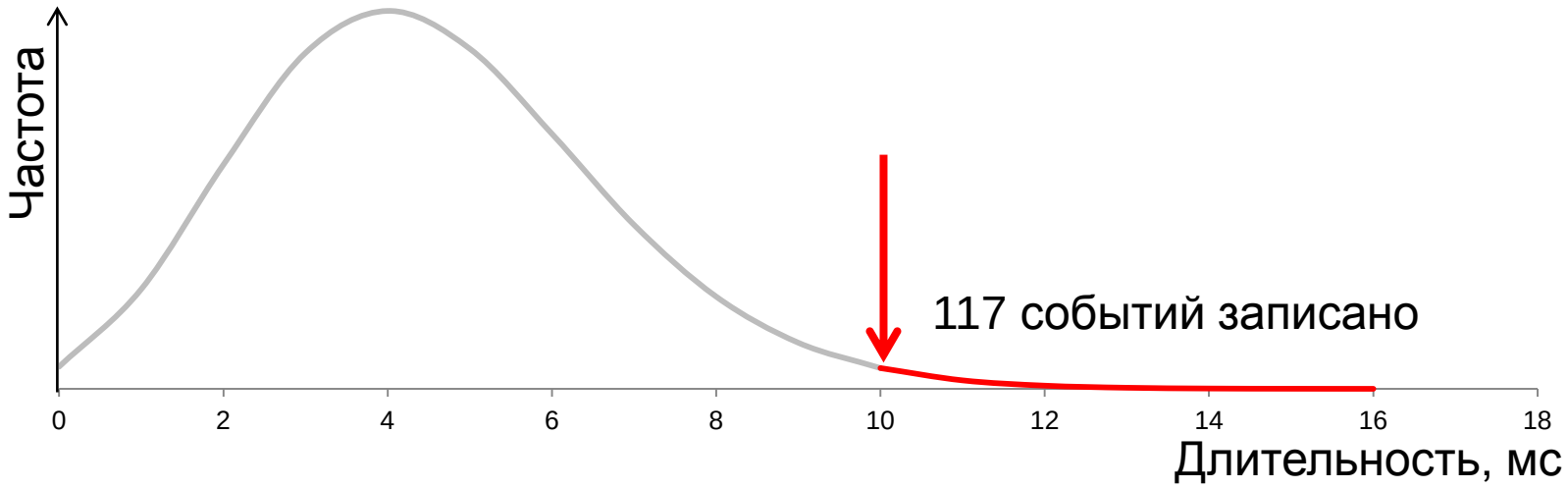
Group By:	Total	Average	Count
java.lang.String	8 s 736.184 ms	74.668 ms	117
weblogic.servlet.internal.session.ReplicatedSessionData	453.186 ms	56.648 ms	8
weblogic.rjvm.MsgAbbrevJVMConnection$WritingState	38.058 ms	38.058 ms	1

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?

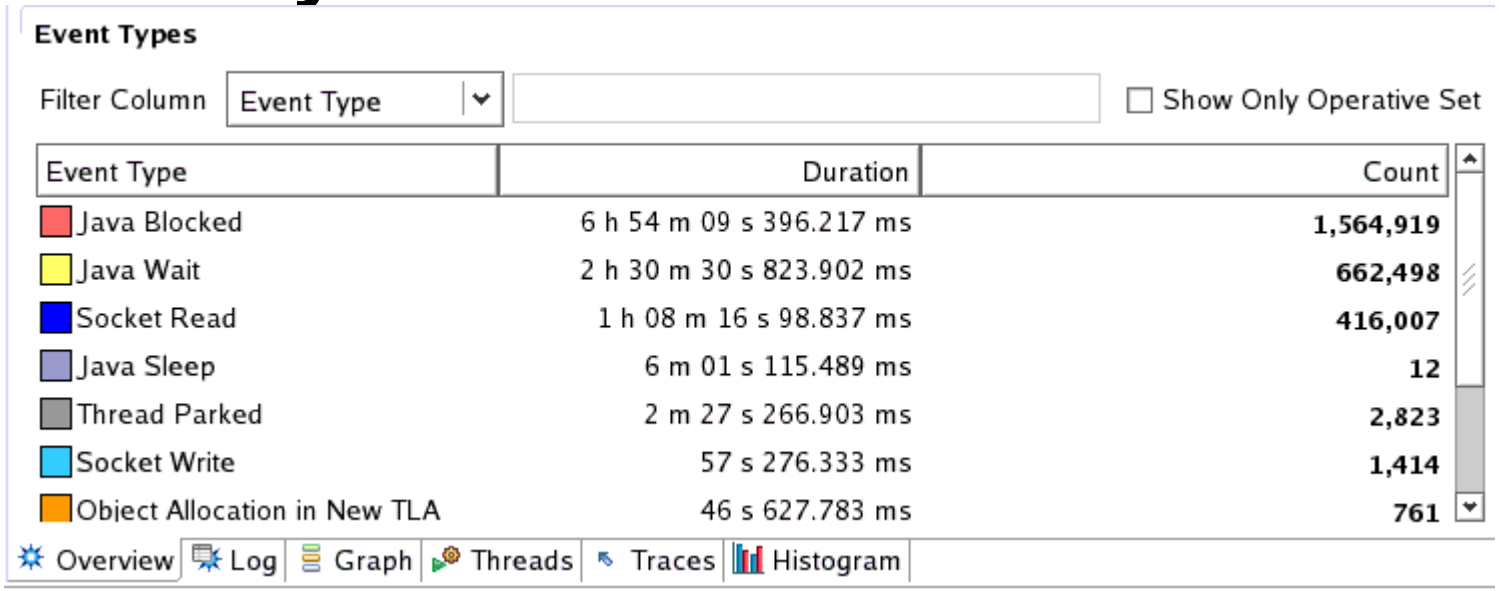
Запись Latency – как проверить

- 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

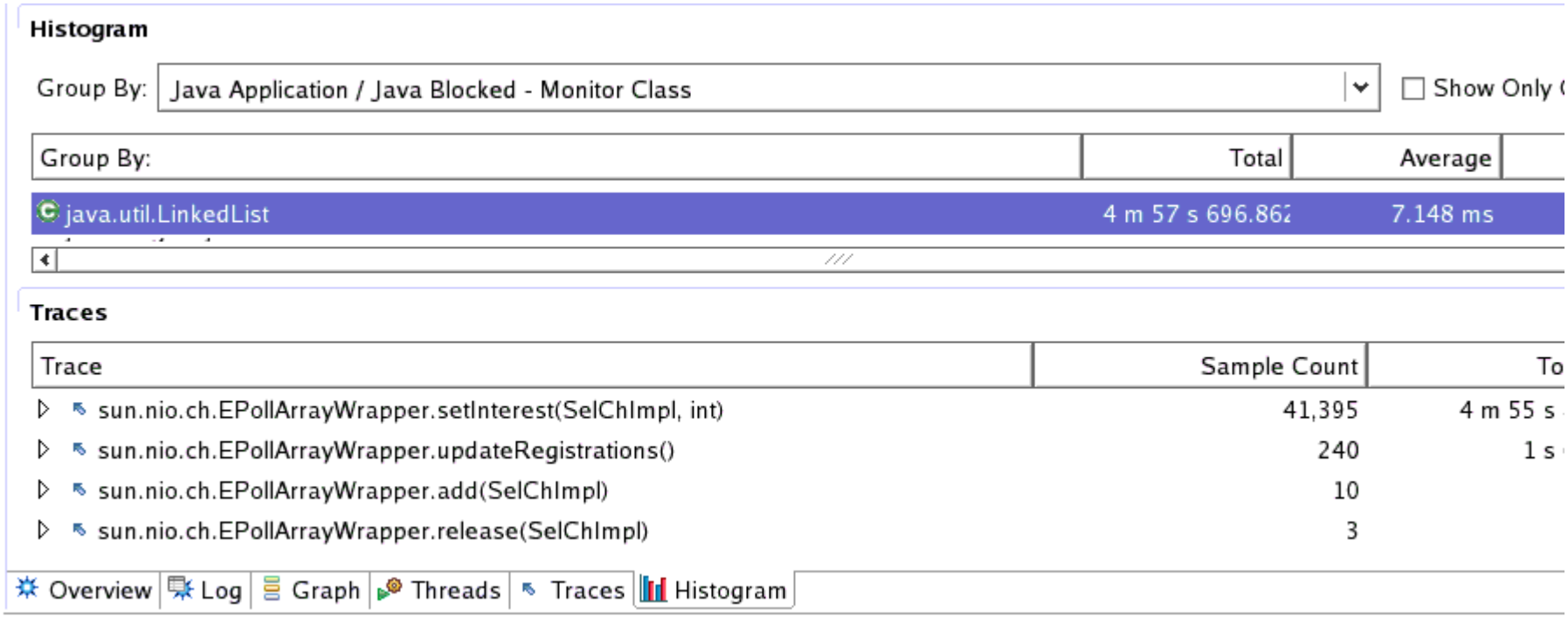
Latency



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

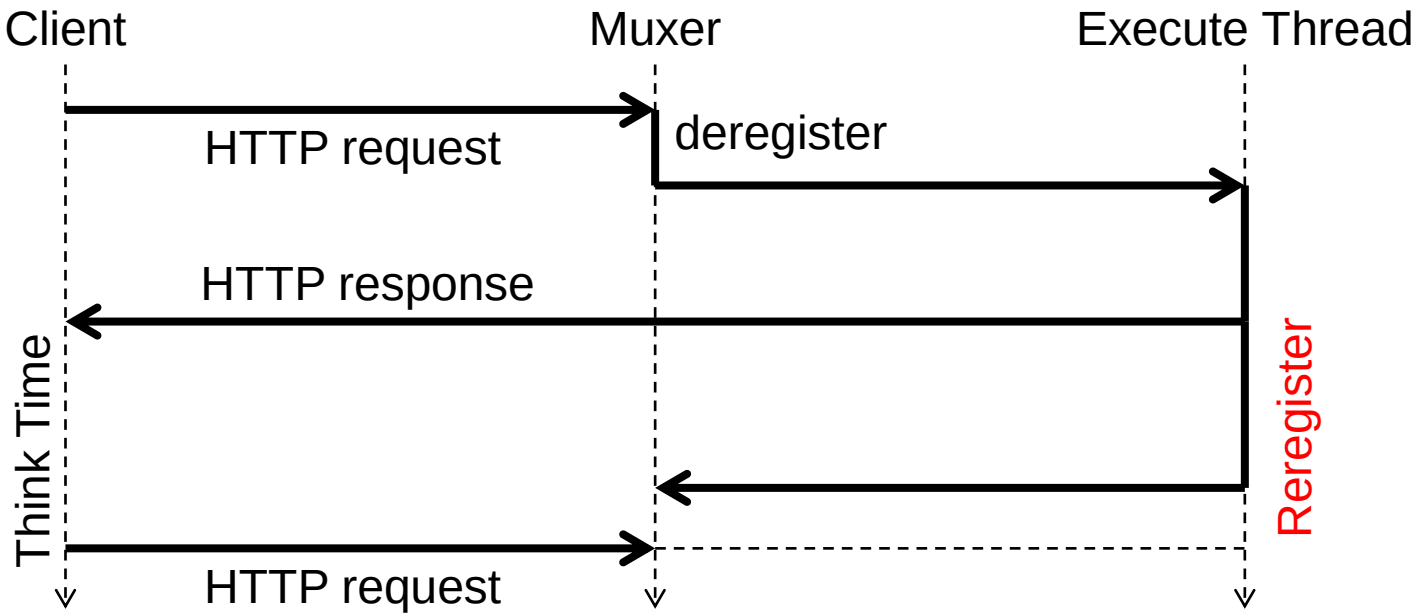
- Понижение порога – не панацея
  - Но хоть диагностика более репрезентативная
- Например, вот здесь отсутствующие latencies меньше 1 ms

Знай задачу



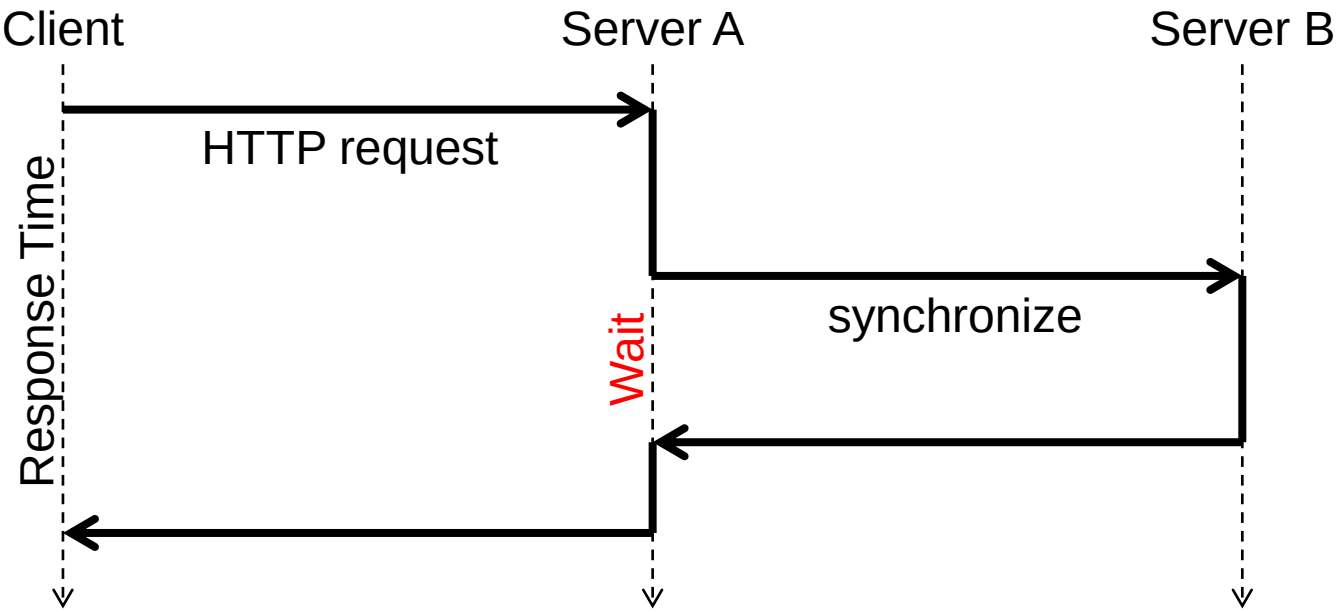
- Горячий Lock! Чинить немедленно!
  - Да ну?

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 10х времени не на CPU против на CPU (и это мы ещё профайлер не открывали)