ПУТЕВОДИТЕЛЬ ПО ПРОФИЛИРОВАНИЮ ПРИЛОЖЕНИЙ НА JVM

Владимир Плизга
Tibbo Systems
JPoint 2025

ПРОФИЛЬ

Профилирование 😎

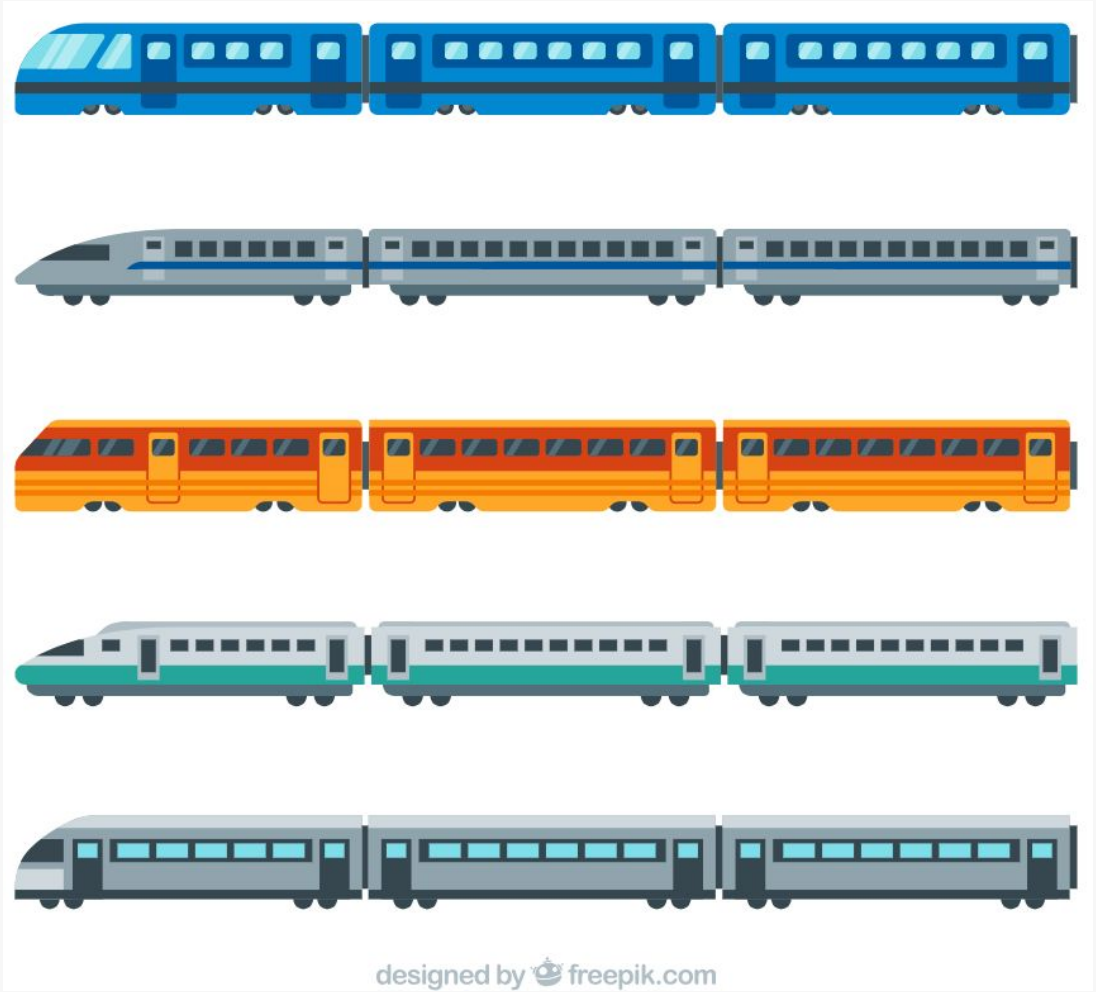
ПРОФИЛЬ

Профилирование 😎

ФАС

~~**Фасирование**~~ 😕

ПРОФИЛЬ



ФАС

Изображение от pngtree.com

Профилирование 😎

~~**Фасирование**~~ 😕

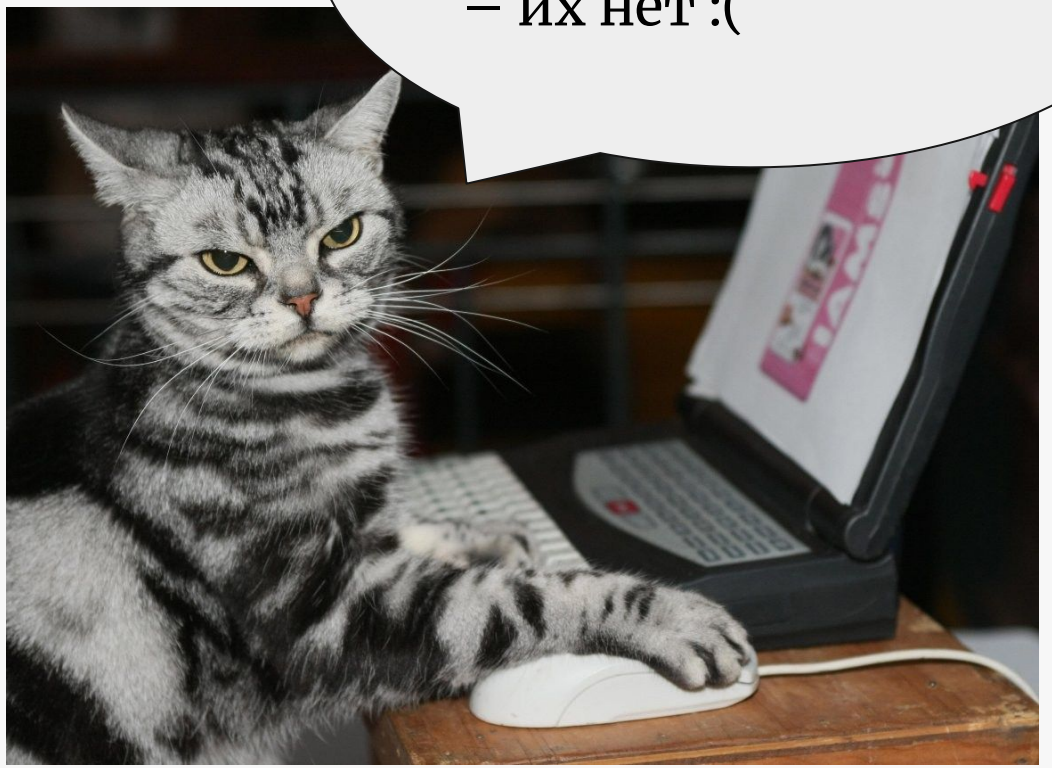
ПУТЕВОДИТЕЛЬ ПО ПРОФИЛИРОВАНИЮ ПРИЛОЖЕНИЙ НА JVM

Владимир Плизга
Tibbo Systems
JPoint 2025

ПРИВЕТ!

- Я – Владимир Плизга́
- Пишу на Java с 2011 г.
  - ФинТех (интернет-банк)
  - Промышленный IoT
- Люблю помогать людям (особенно разработчикам)

КОНТЕКСТ



КОНТЕКСТ

“Логи и метрики не помогают!”

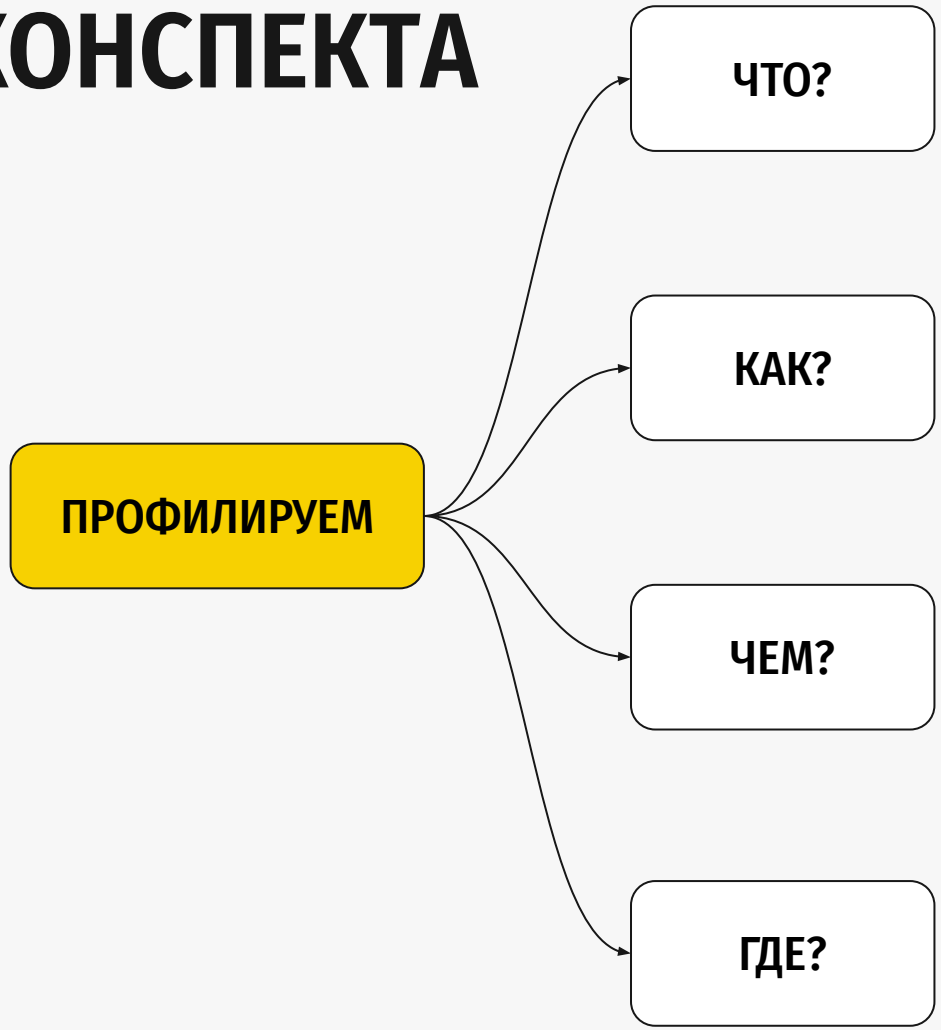
“Хоть на работу не ходи!”

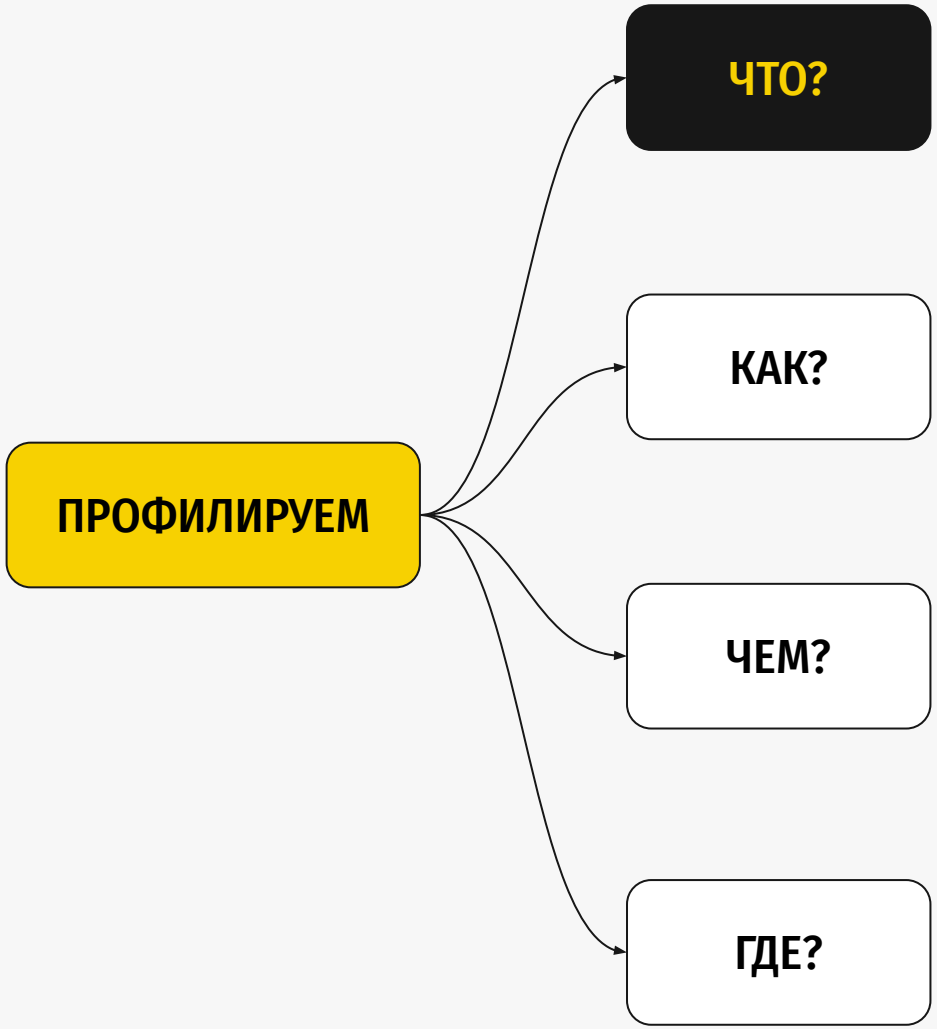
ЧТО ДЕЛАТЬ?

- Выявлять **проблемные** участки в коде
- Доставать их **стектрейсы**
- Оценивать их **продолжительность**

PROFILING TO THE RESCUE

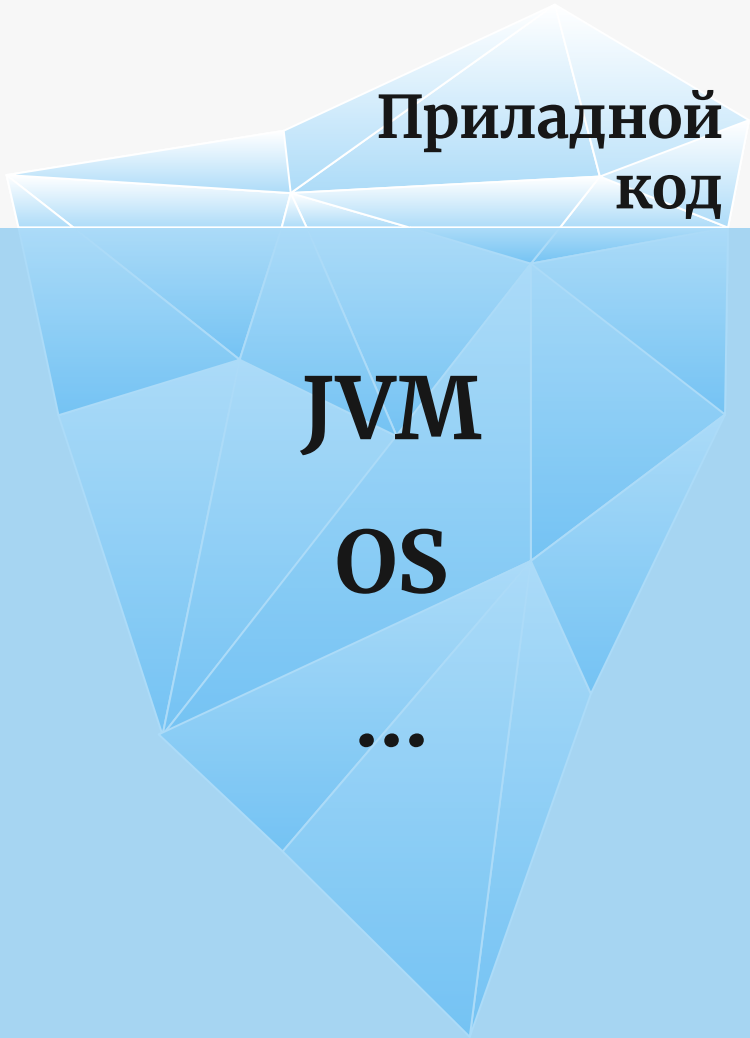
ПЛАН КОНСПЕКТА





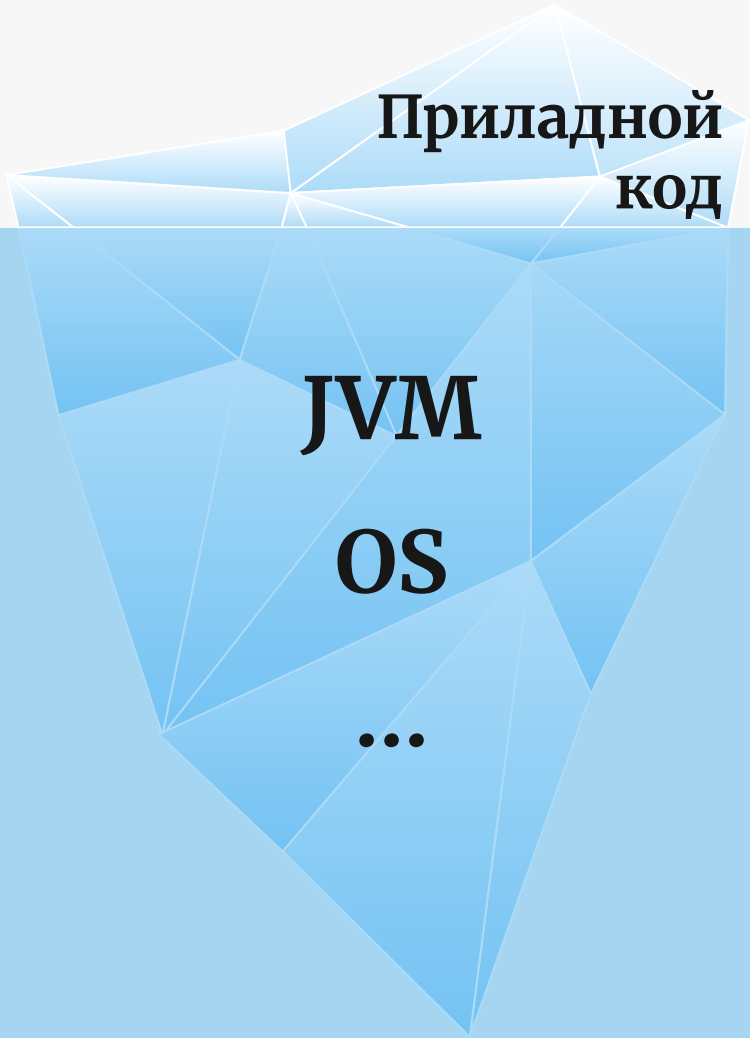
- Вглубь
- Вширь

ЧТО МЫ ПРОФИЛИРУЕМ?



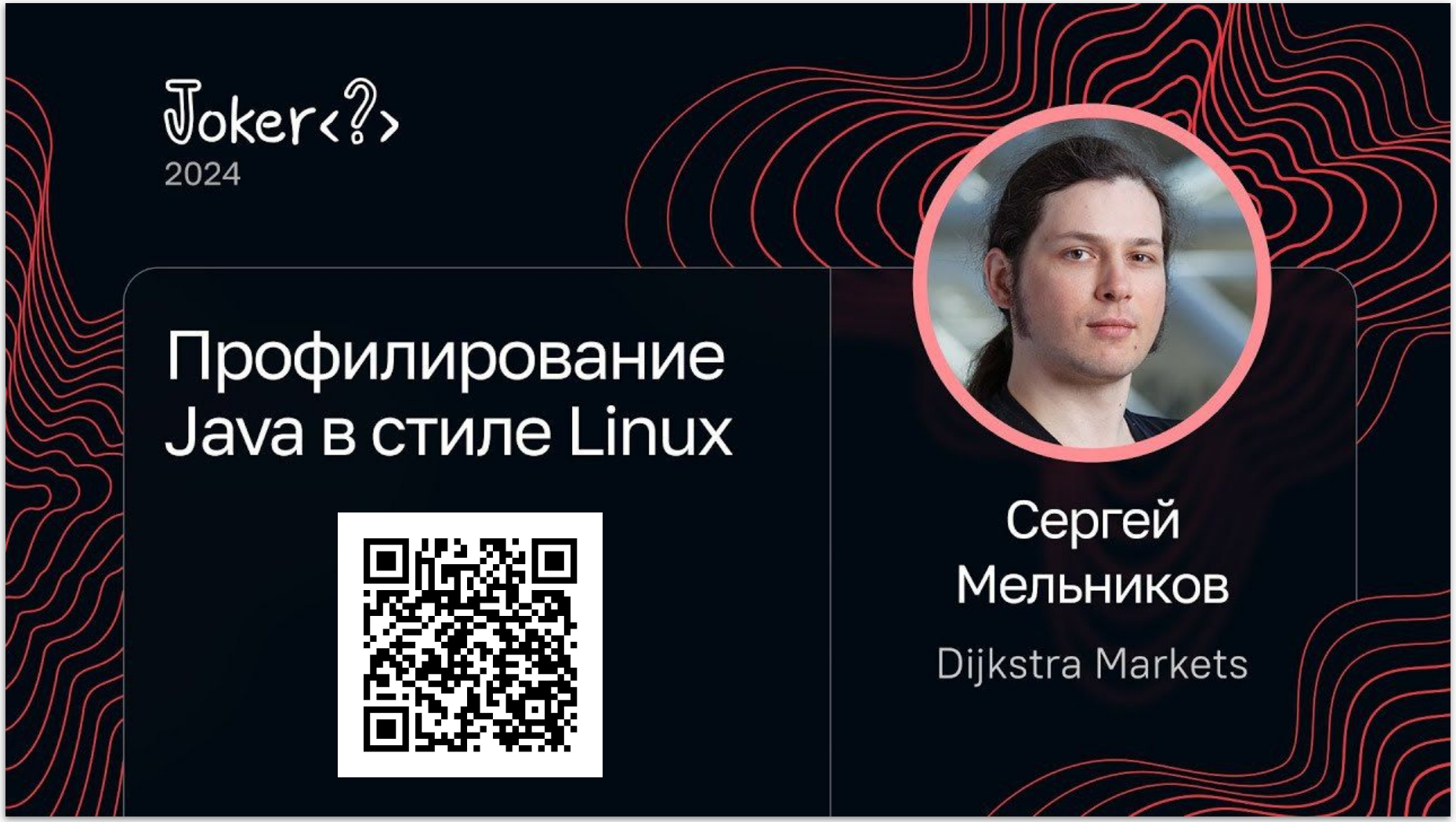
ЧТО МЫ ПРОФИЛИРУЕМ?

Наш фокус

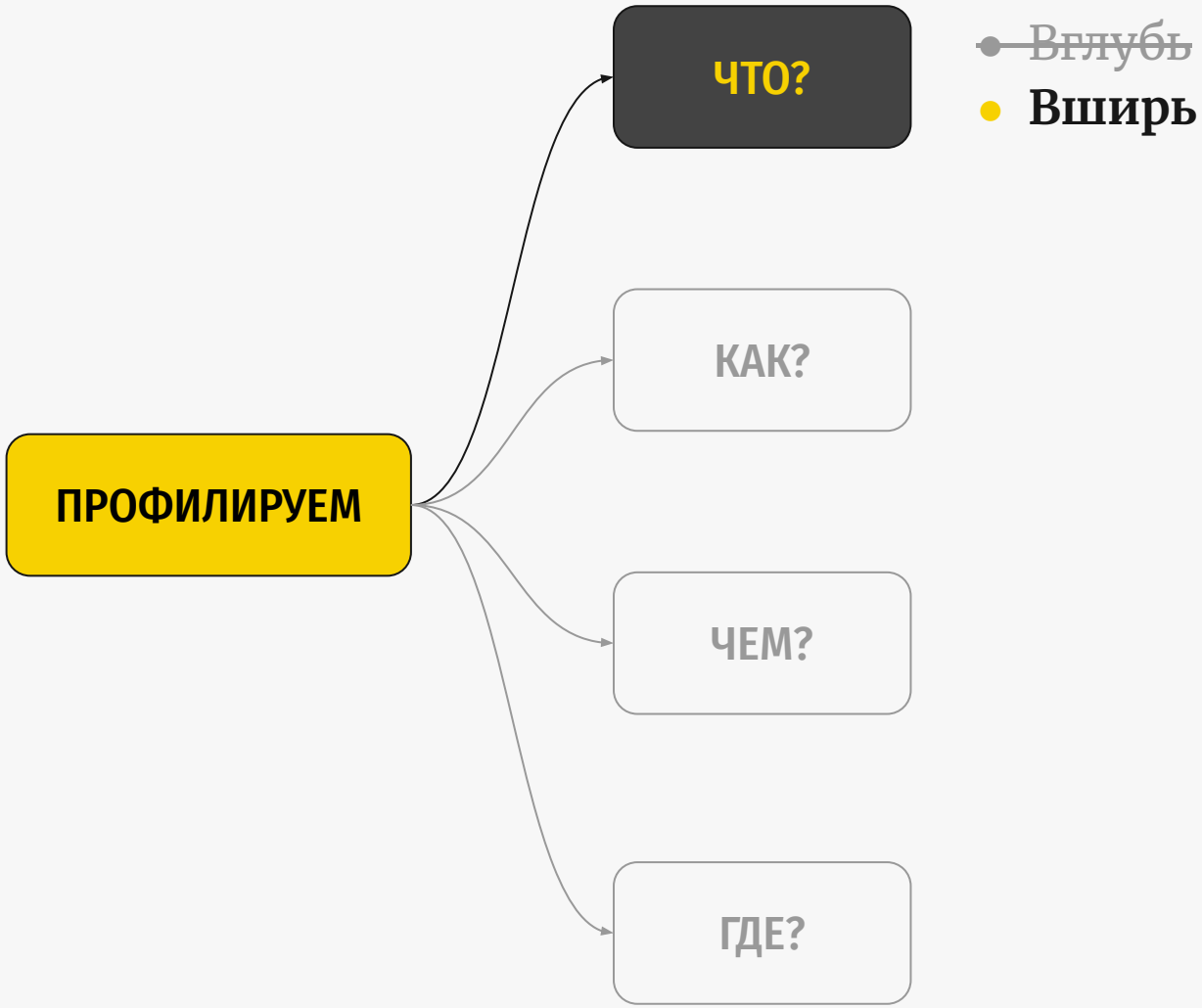


Цепочка следствий

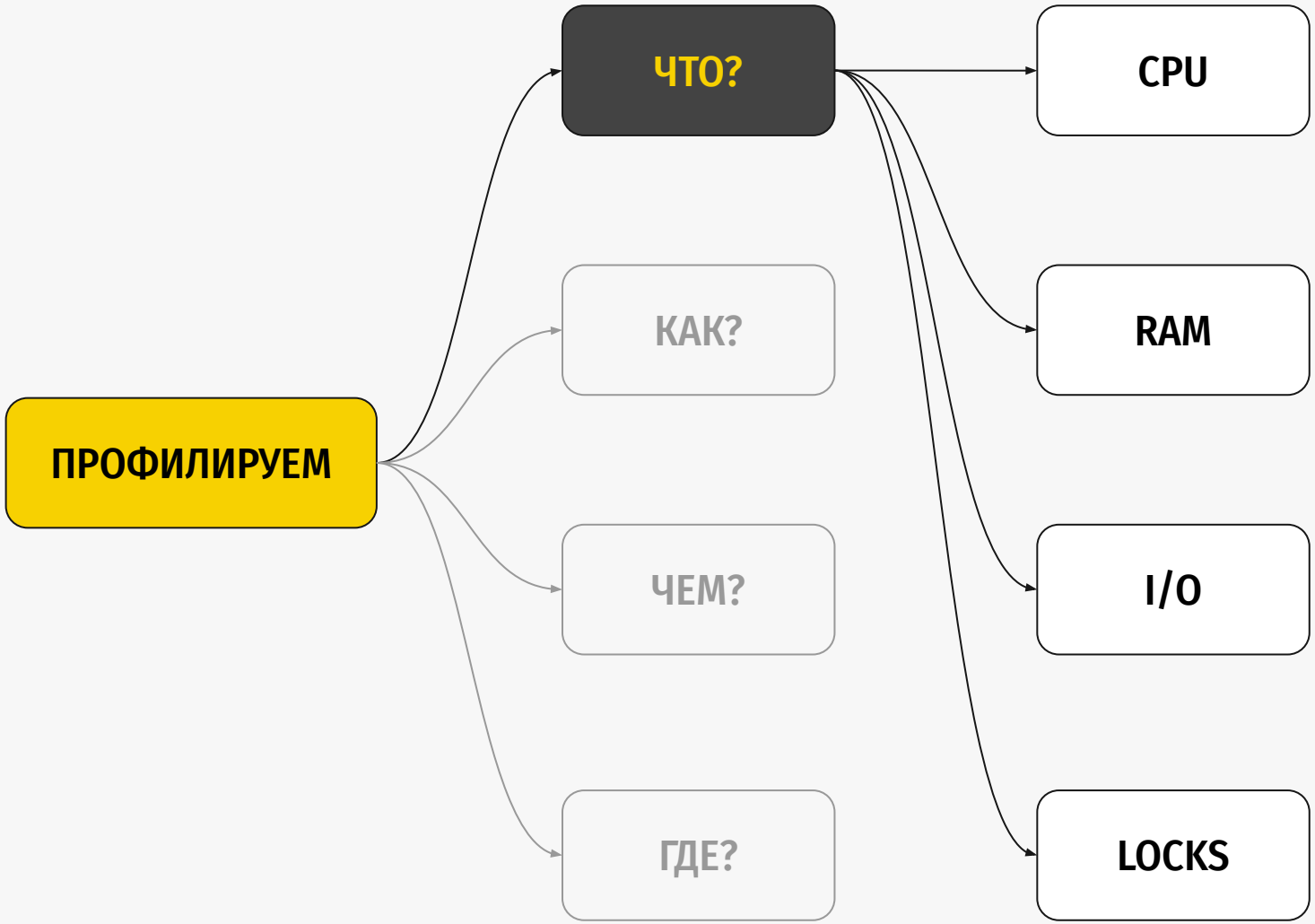
ЕСЛИ ХОЧЕТСЯ ЗАНЫРНУТЬ

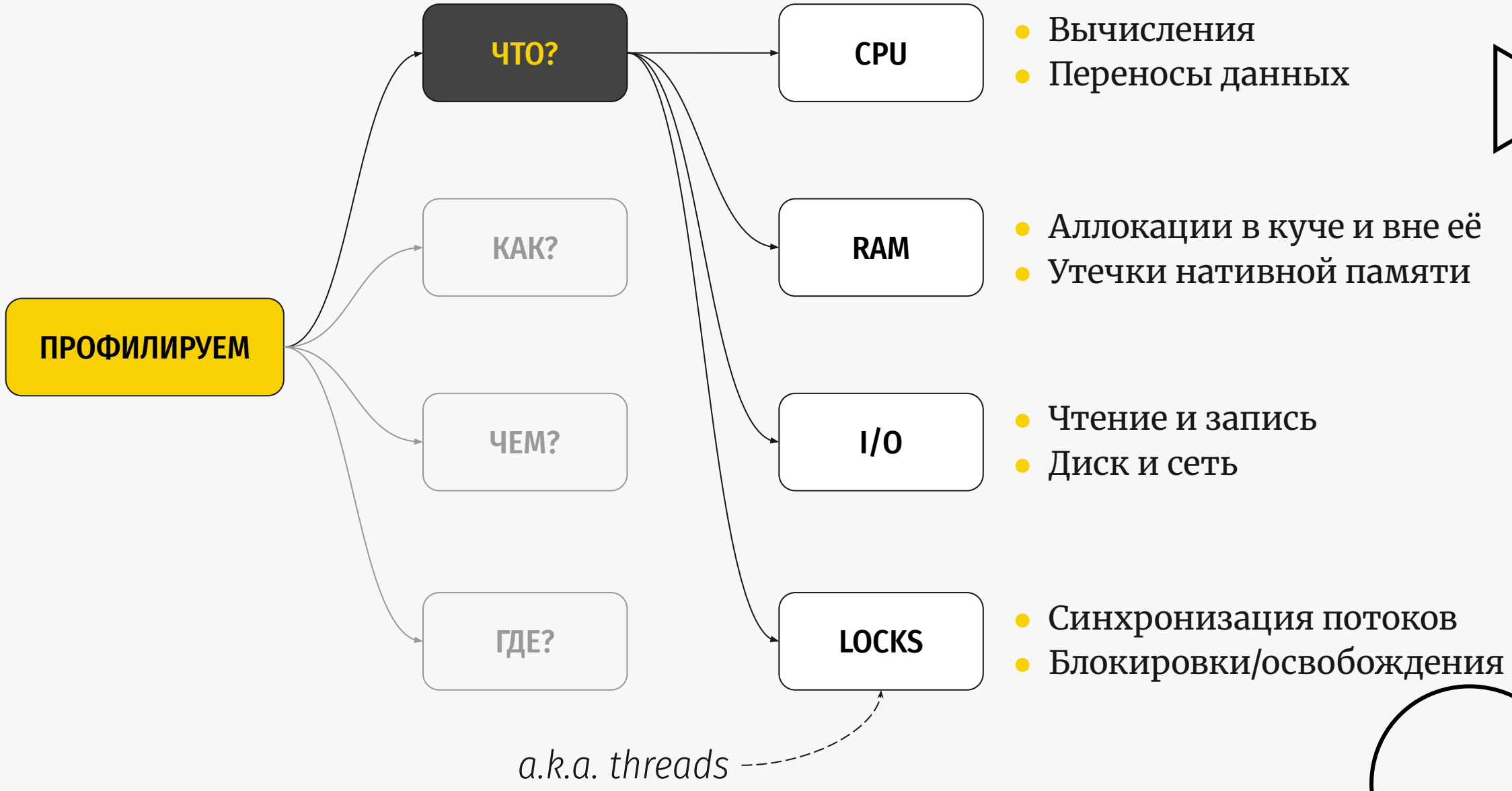


https://www.youtube.com/watch?v=EZA5zcfFA4w



- ~~Вглубь~~
- **Вширь**



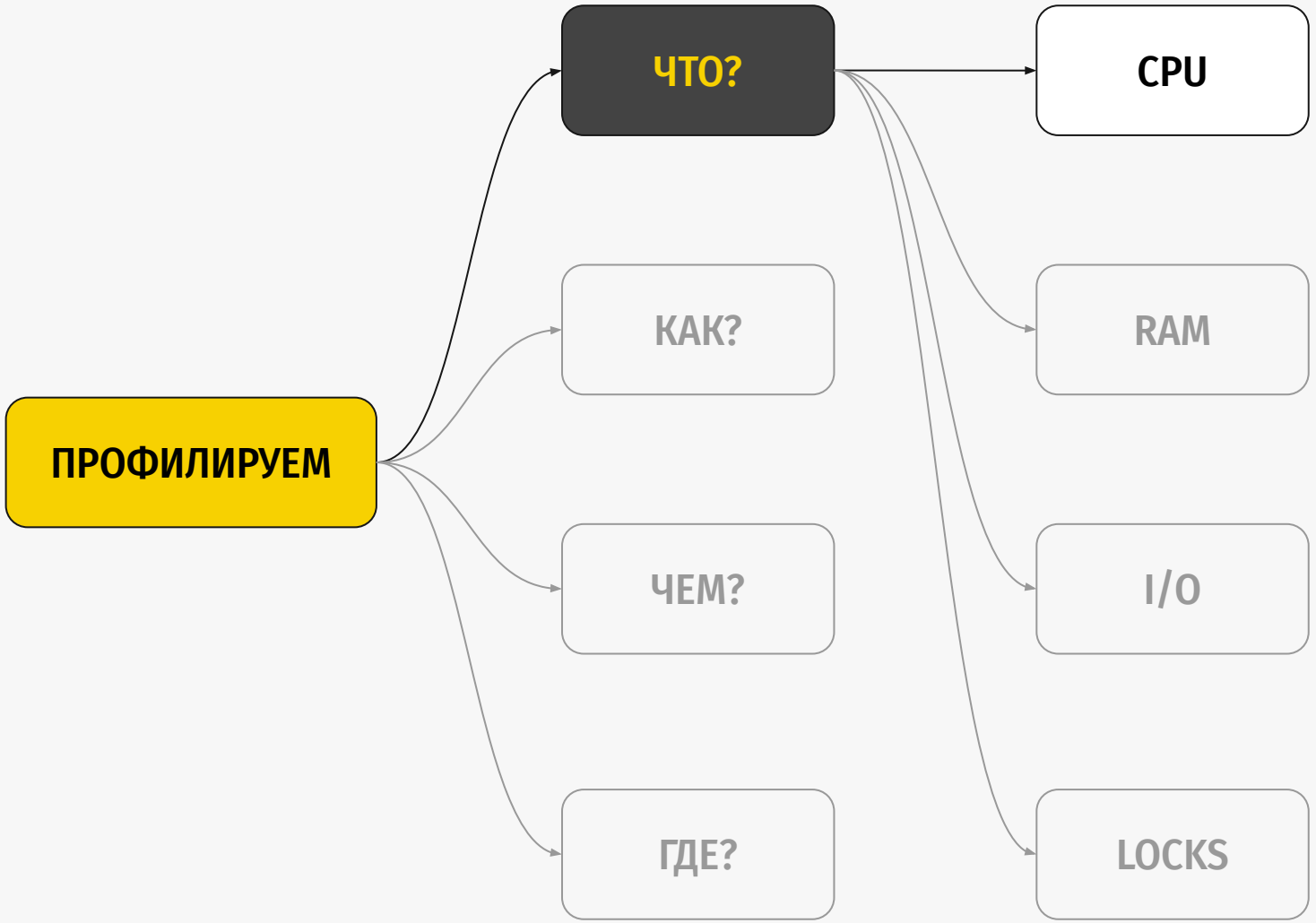


- Вычисления
- Переносы данных

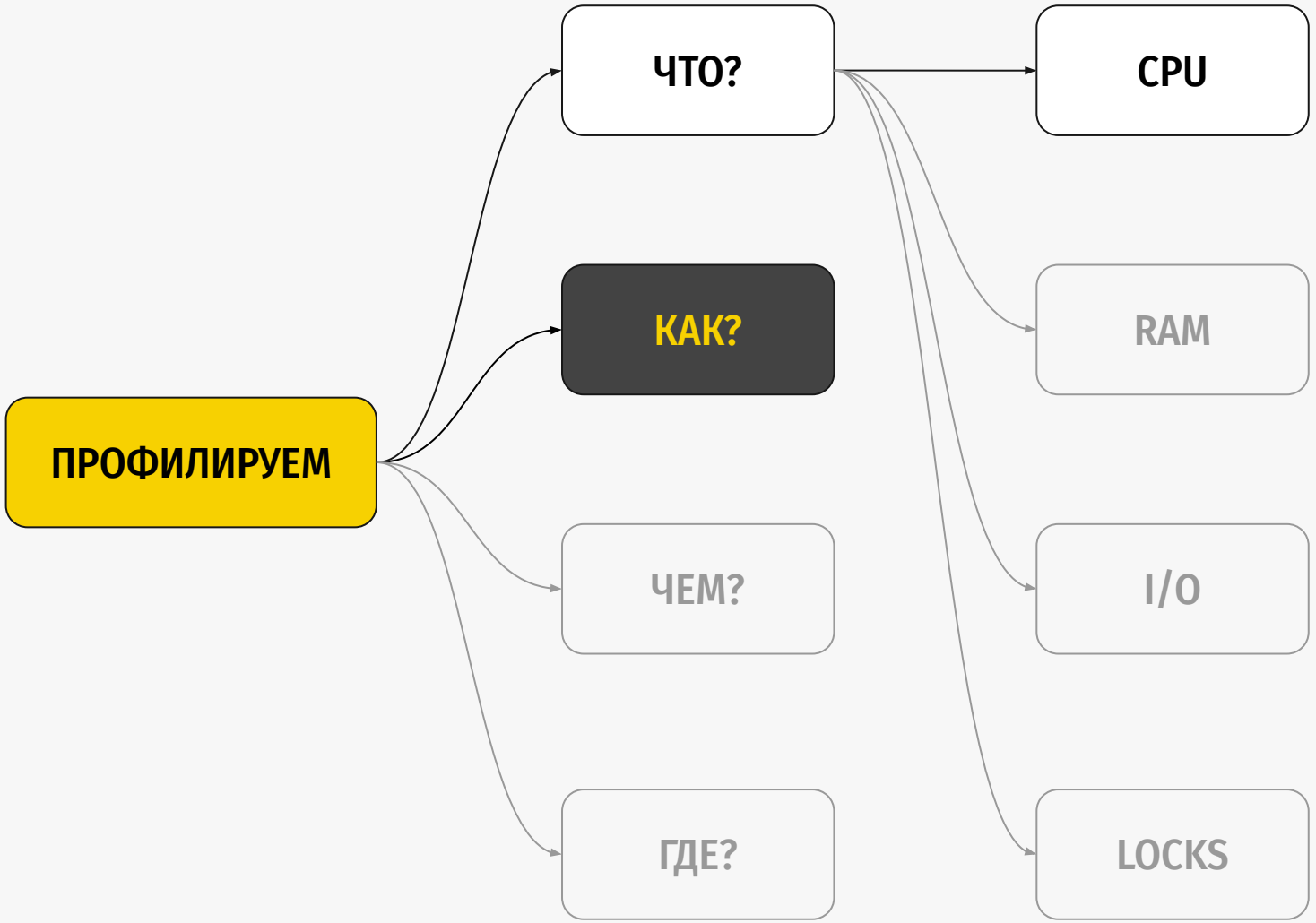
- Аллокации в куче и вне её
- Утечки нативной памяти

- Чтение и запись
- Диск и сеть

- Синхронизация потоков
- Блокировки/освобождения



- Вычисления
- Переносы данных



- Вычисления
- Переносы данных

КАК ПРОФИЛИРОВАТЬ CPU

Подход 1. Sampling

1. Снимаем дампы потоков с какой-то частотой
2. Склеиваем их
3. Чем чаще встречается метод, тем больше он занимает CPU

SAMPLING

Плюсы

- ✓ Легко реализовать
- ✓ Просто использовать
- ✓ Управляемый overhead

SAMPLING

Плюсы

- ✓ Легко реализовать
- ✓ Просто использовать
- ✓ Управляемый overhead

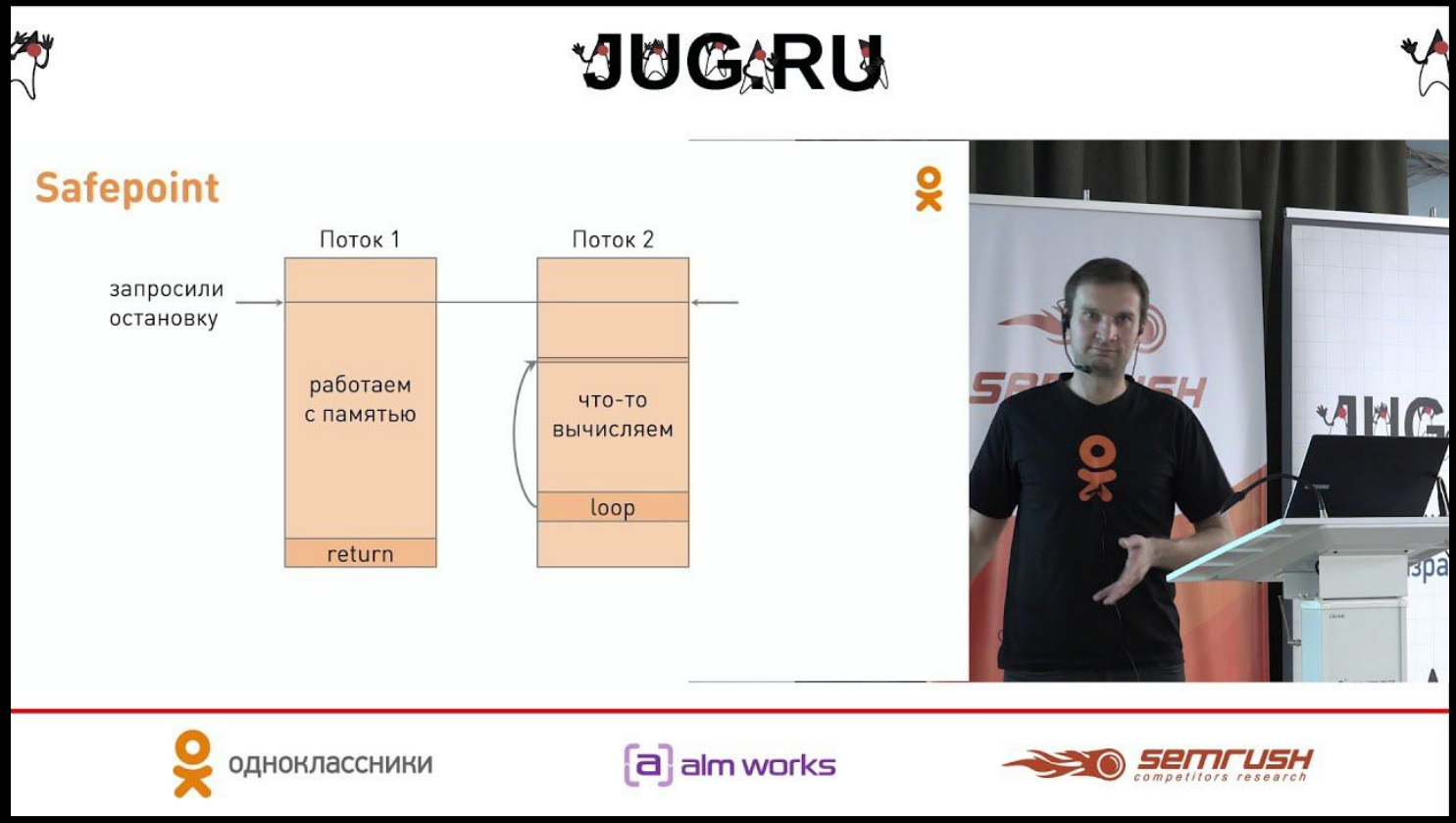
Минусы

- x Не всегда точен
- x Подвержен safepoint bias*

* SAFEPOINT BIAS (НА ПАЛЬЦАХ)

- Поток JVM нельзя остановить в любом месте
- Но можно в т.н. safepoint’ах
  - например, на выходе из метода и повторении циклов
- Оптимизации в JVM могут влиять на исполнение:
  - методы становятся inline
  - циклы превращаются в последовательности
- **Результат**: safepoint’ы прореживаются, sampling теряет в точности

ГДЕ УЗНАТЬ БОЛЬШЕ



Андрей Паньгин — Искусство Java профилирования

https://www.youtube.com/watch?v=QiGrTvsCZmA

SAMPLING

Плюсы

- ✓ Легко реализовать
- ✓ Просто использовать
- ✓ Управляемый overhead

Минусы

- x Не всегда точен
- x Подвержен safepoint bias*

SAMPLING

Плюсы

- ✓ Легко реализовать
- ✓ Просто использовать
- ✓ Управляемый overhead

Минусы

- x Не всегда точен
- x Подвержен safepoint bias*
- x Не умеет считать вызовы**

** ПОДСЧЕТ ЧИСЛА ВЫЗОВОВ

- Пример тормозящей цепочки методов:
  - `findUsersByIds → fetchDataFromDB → executeSQL`
- Возможные причины:
  - метод `executeSQL()` долго работает с большим `IN`
  - метод `findUsersByIds()` делает запросы по каждому `ID`
- Sampling укажет **только** на `executeSQL()`
- Различить можно по числу вызовов

КАК ПРОФИЛИРОВАТЬ CPU

Подход 2. Tracing a.k.a. instrumentation

1. Внедряем в байт-код метки вызова методов
2. Строим стектрейсы по ним
3. Получаем точные временные рамки работы методов

TRACING

Плюсы

- ✓ Высокая точность
- ✓ Подсчет числа вызовов
- ✓ Путь к high-level метрикам

TRACING

Плюсы

- ✓ Высокая точность
- ✓ Подсчет числа вызовов
- ✓ Путь к high-level метрикам

Минусы

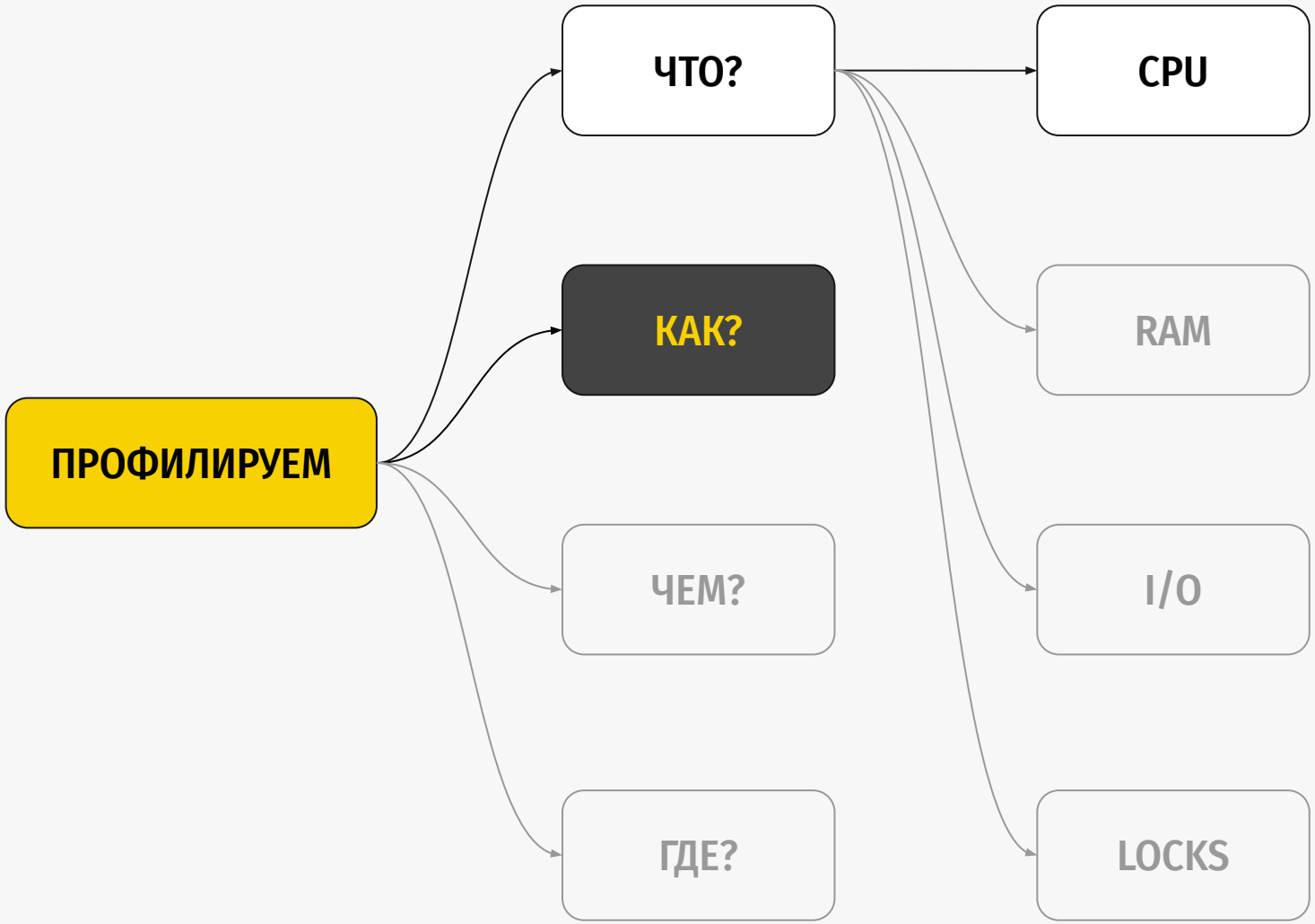
- x Лютый overhead
- x Сложность настройки
- x Может влиять на профиль*

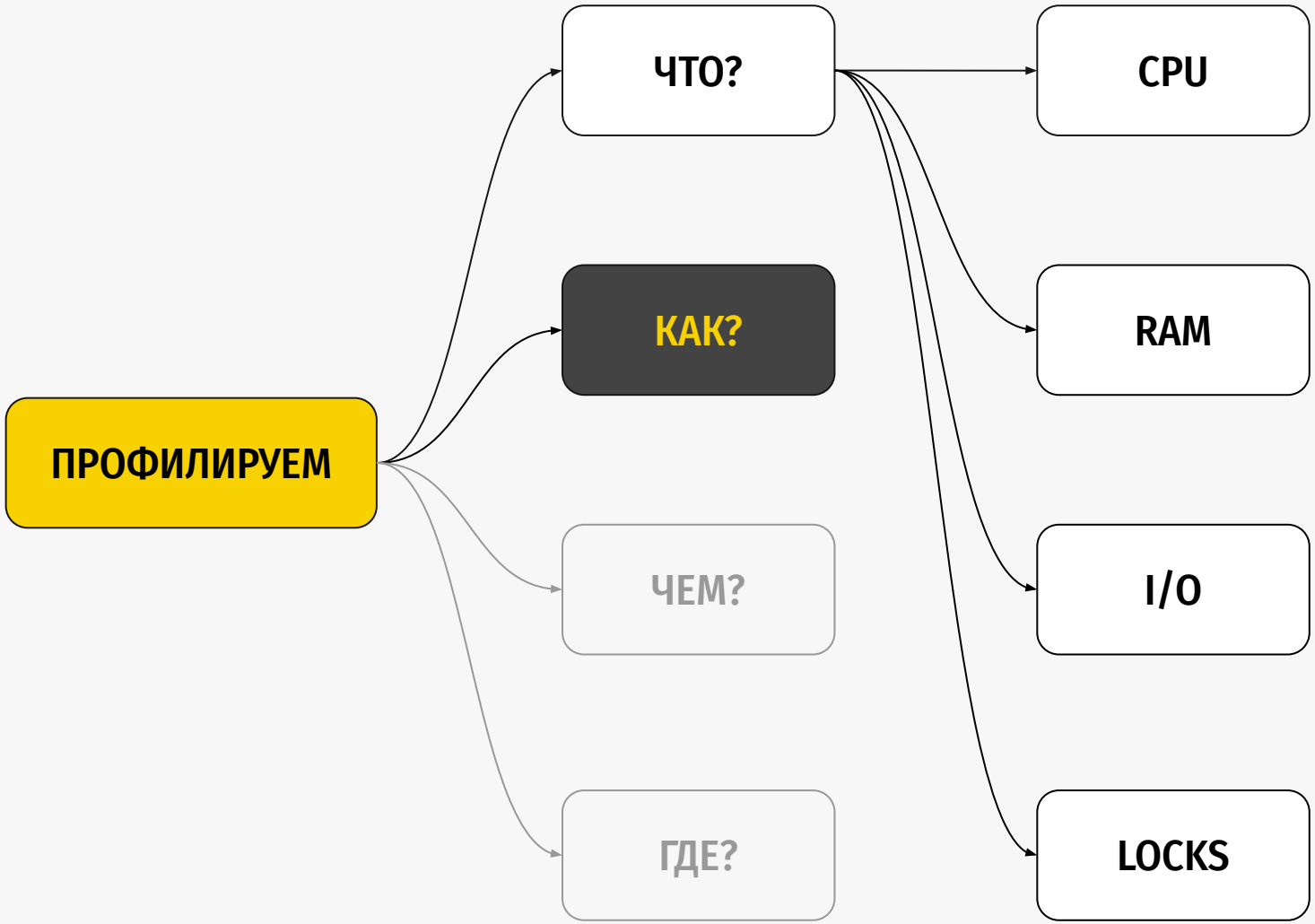
* (ДЕ)ОПТИМИЗАЦИЯ (НА ПАЛЬЦАХ)

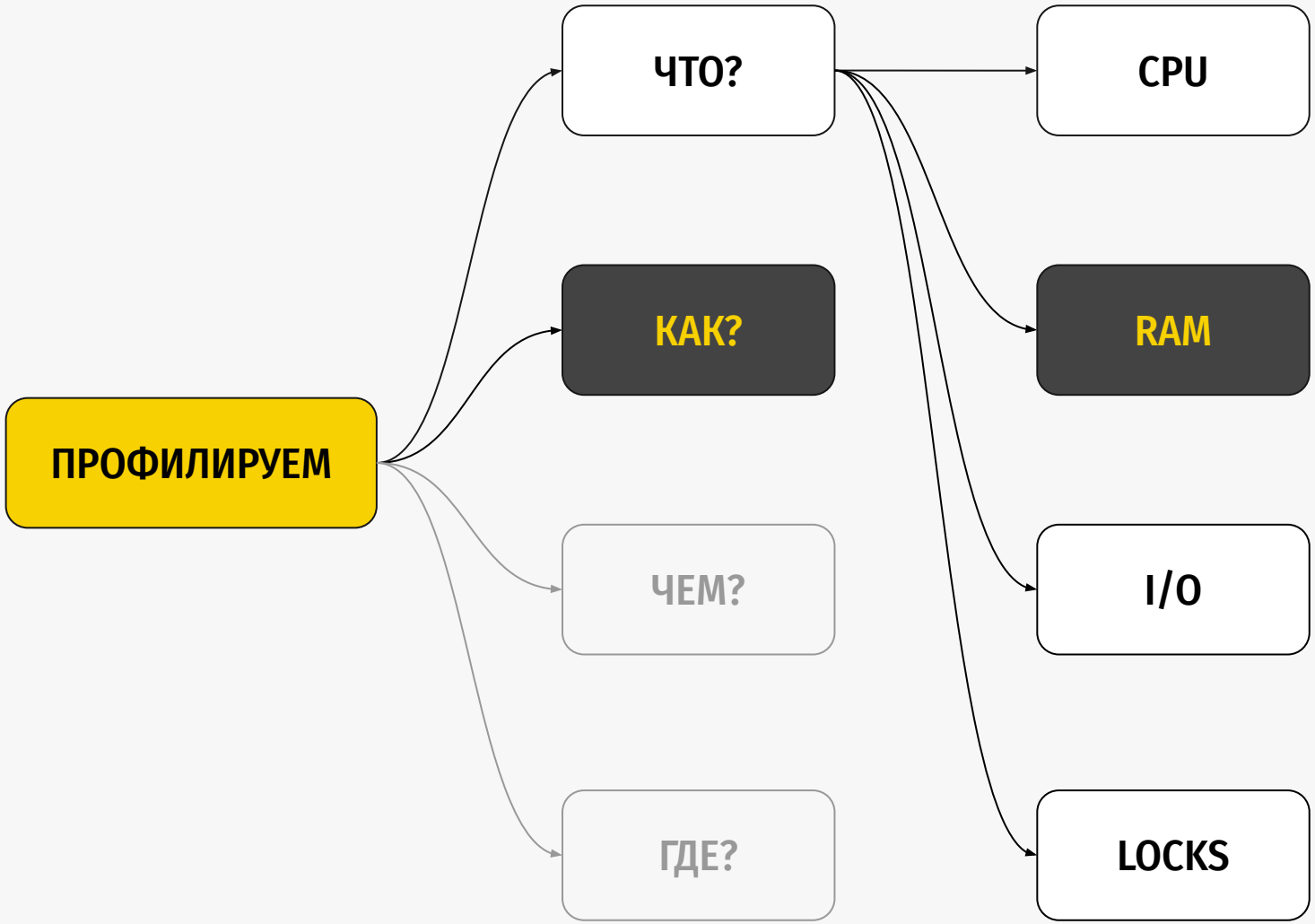
- Наиболее “горячие” методы оптимизируются самой JVM
- Их исполняемый код может не соответствовать исходному
- Включение трейсинга меняет исходный класс
- Это приводит к:
  - (не)применению других оптимизаций
  - деоптимизации ранее сгенерированного кода
- **Результат**: профиль меняется, трейсинг может врать

ГДЕ УЗНАТЬ БОЛЬШЕ

https://www.youtube.com/watch?v=9valLOxgDbI







АЛЛОКАЦИЯ ПАМЯТИ В КУЧЕ (НА ПАЛЬЦАХ)

- Новые объекты создаются в поточно-локальных буферах
  - **TLAB** - Thread-Local Allocation Buffer
- Если объект не влазит в текущий TLAB, создастся новый
- Если объект огромный, он может быть создан вне TLAB
  - это требует синхронизации, поэтому называется **slow path**
- На оба этих случая в JVM есть события, на них можно подписаться
- **Подробнее**: Как JVM аллоцирует объекты? (Хабр)

КАК ПРОФИЛИРОВАТЬ ДРУГИЕ РЕСУРСЫ

Подход 3. Event-based

1. Полагаемся на события обращения к ресурсам внутри JVM
2. Собираем стектрейсы с них
3. Получаем точную картину по ресурсам

EVENTS

Плюсы

- ✓ Высокая точность
- ✓ Умеренный overhead
- ✓ Нативная поддержка

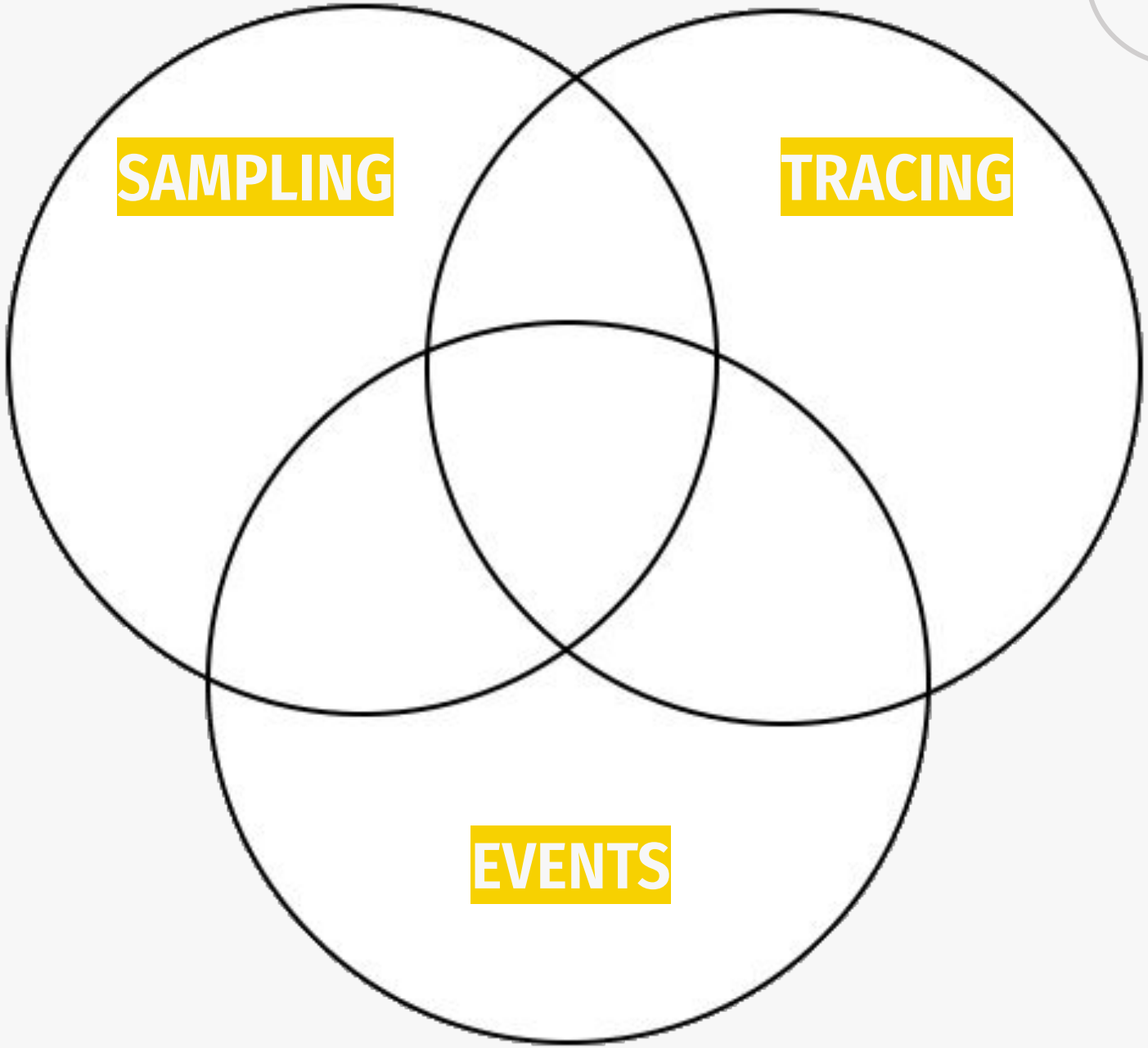
Минусы

- x Слабая связь с бизнес-логикой приложения
- x Ограниченная применимость

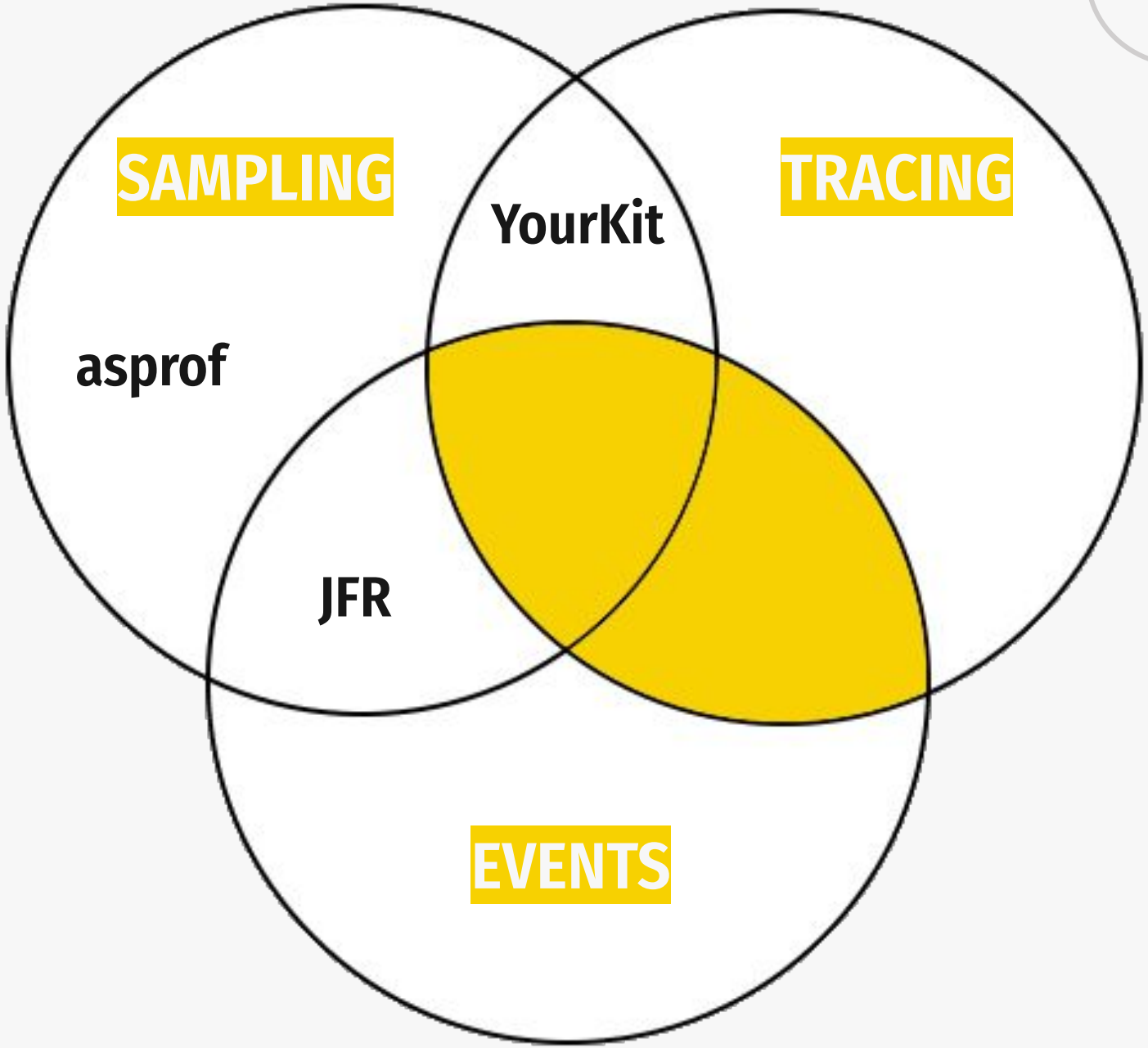
ПОПУЛЯРНЫЕ ПРЕДСТАВИТЕЛИ

- Sampling
  - Async-profiler a.k.a. asprof
- Tracing
  - YourKit Java Profiler a.k.a. YourKit
- Events
  - JDK Flight Recorder a.k.a. JFR

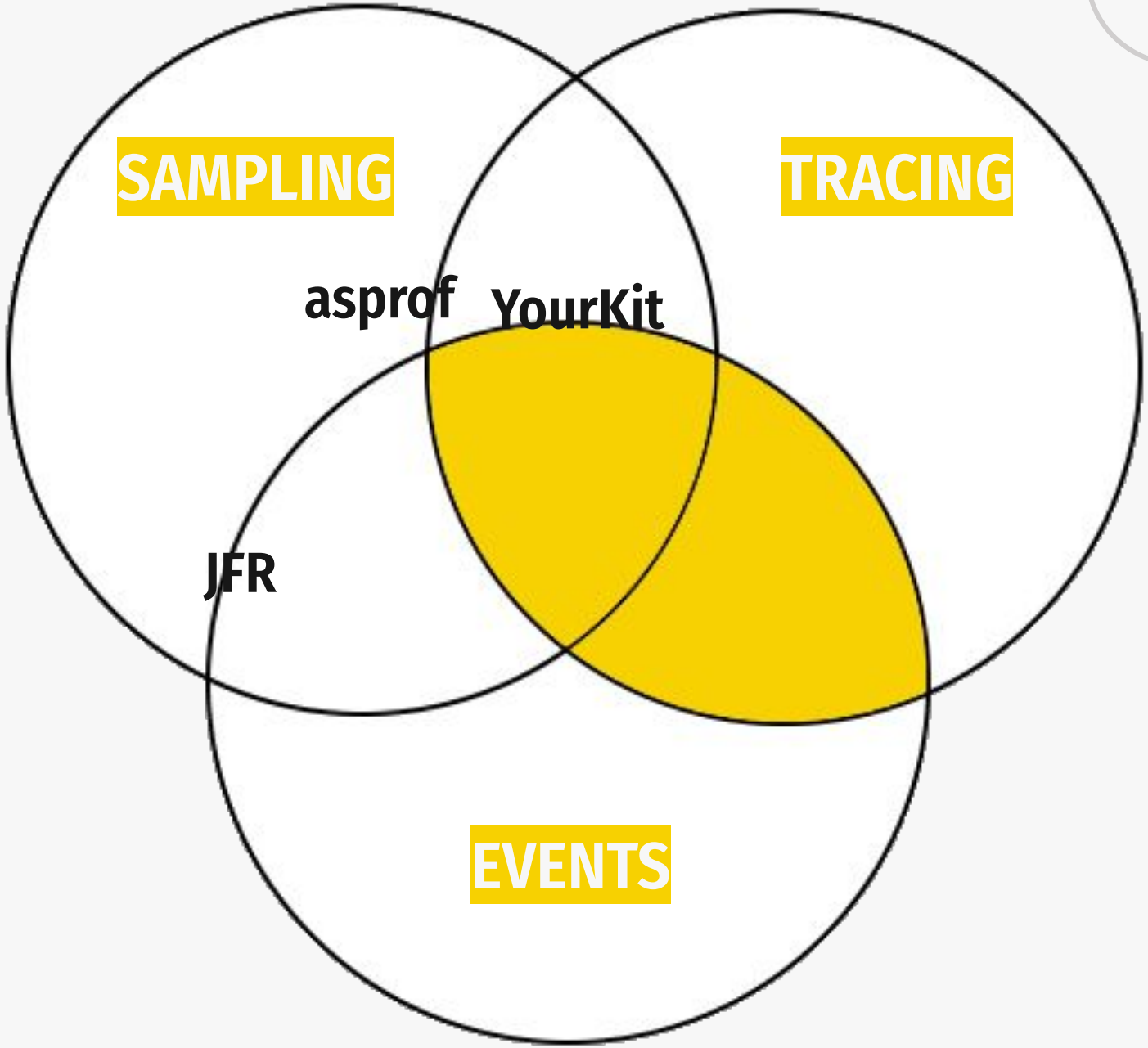
А ЕСЛИ ТОЧНЕЕ

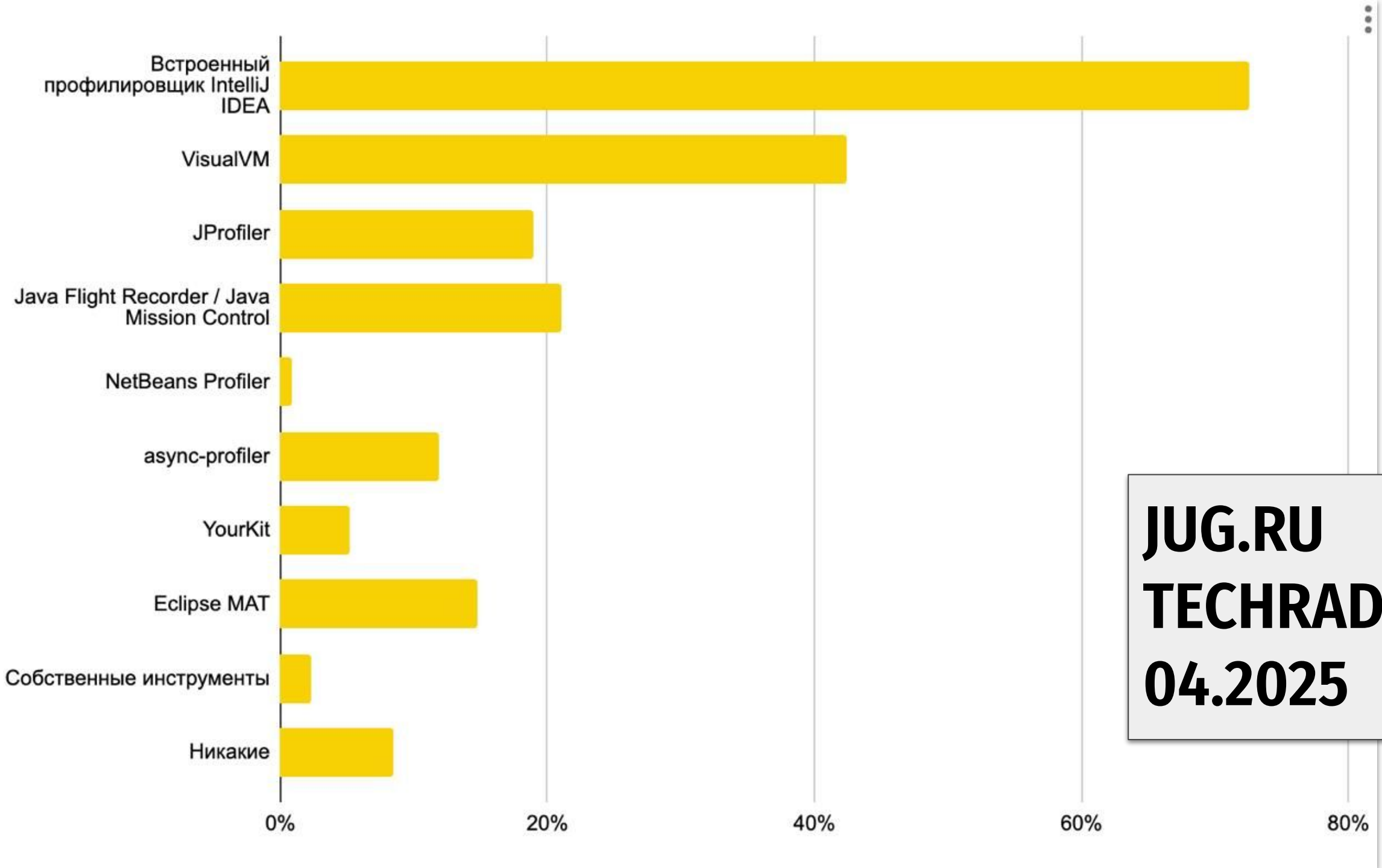


А ЕСЛИ ТОЧНЕЕ



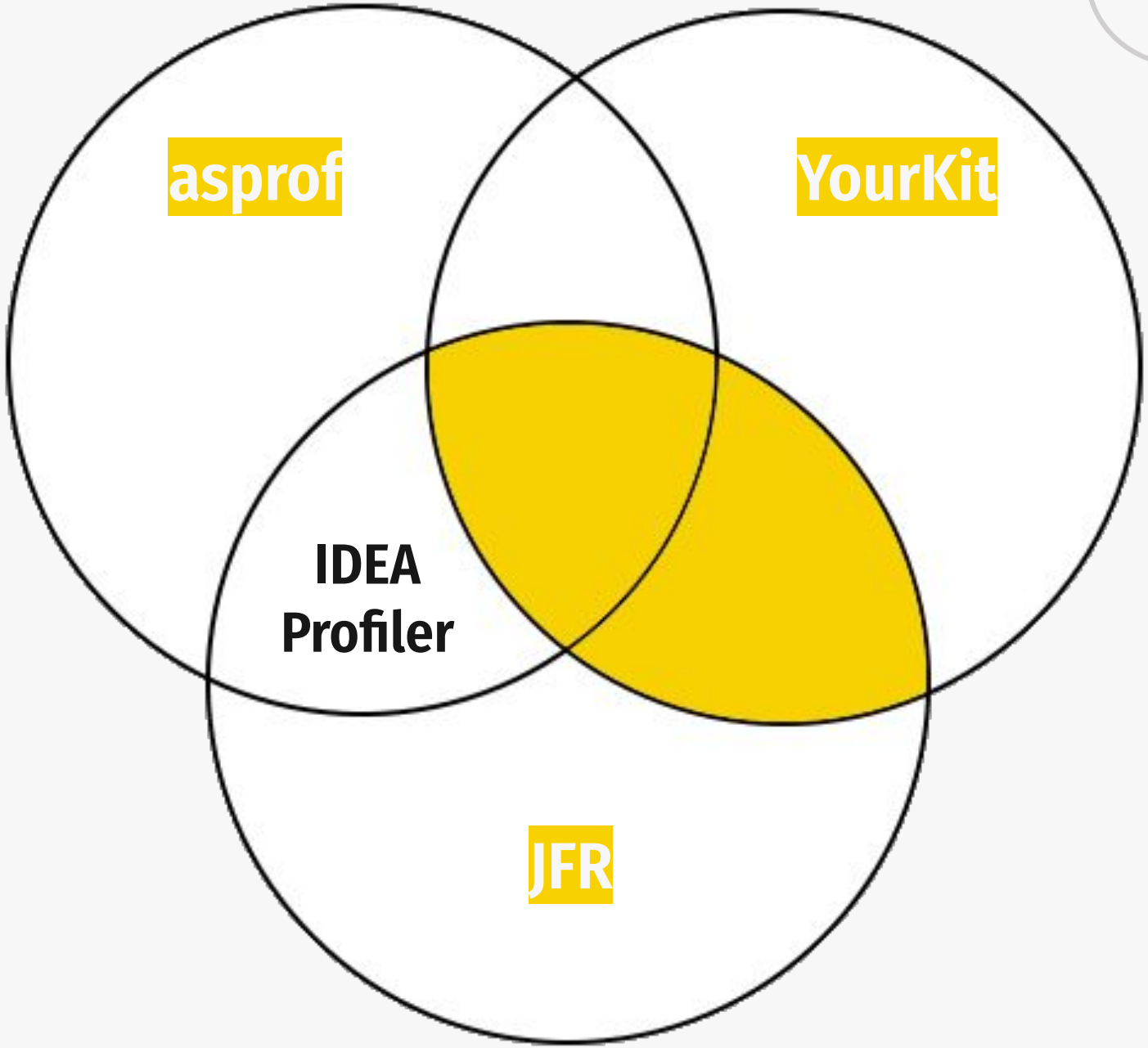
А ЕСЛИ ЕЩЕ ТОЧНЕЕ



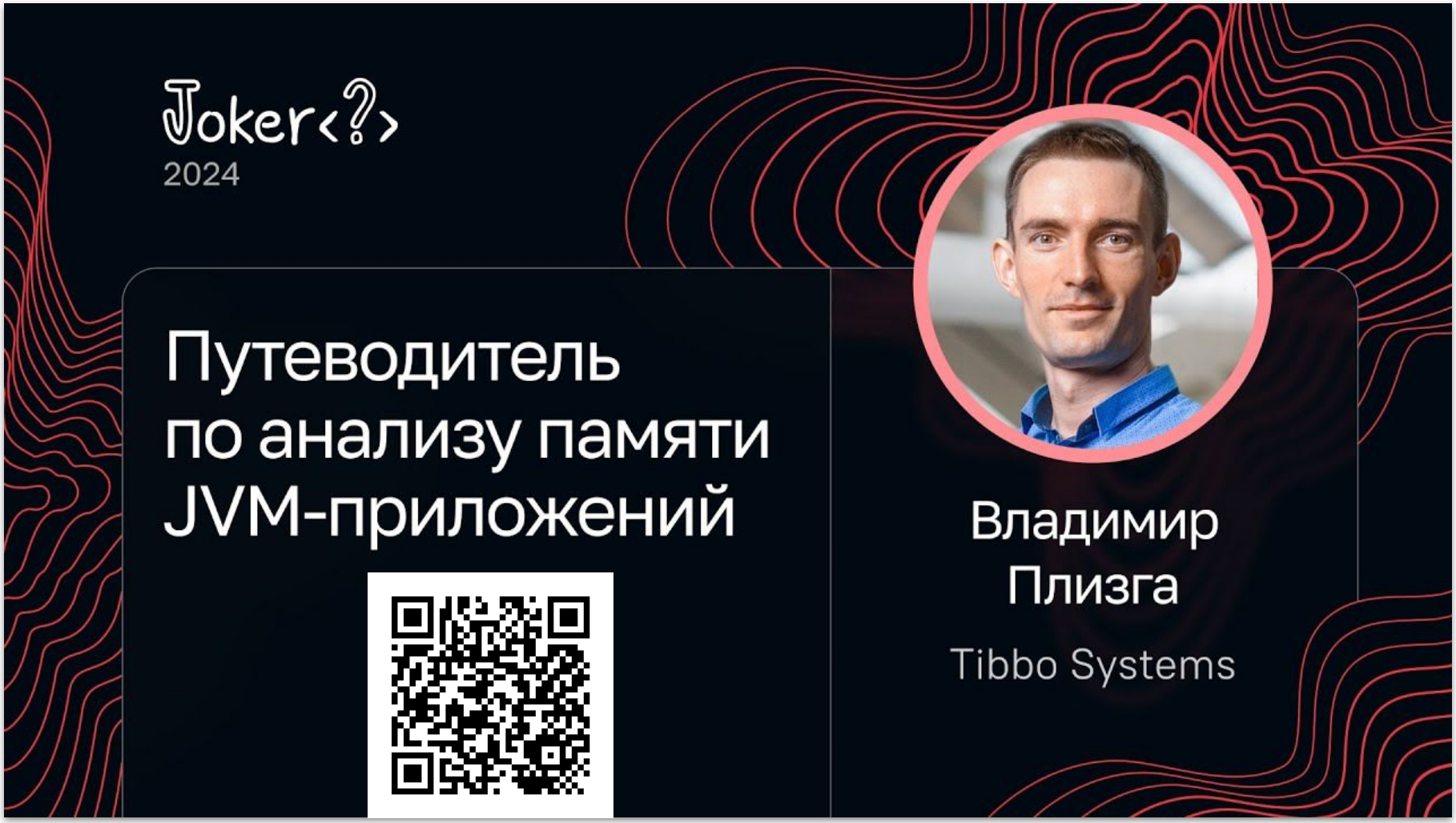


JUG.RU TECHRADAR 04.2025

А ГДЕ ЖЕ ПРОФАЙЛЕР ИЗ IDEA?



О ПРОФИЛИРОВАНИИ ПАМЯТИ



https://www.youtube.com/watch?v=fPns2O-cnYQ

А ГДЕ ЖЕ ПРОФАЙЛЕР <XXX>?

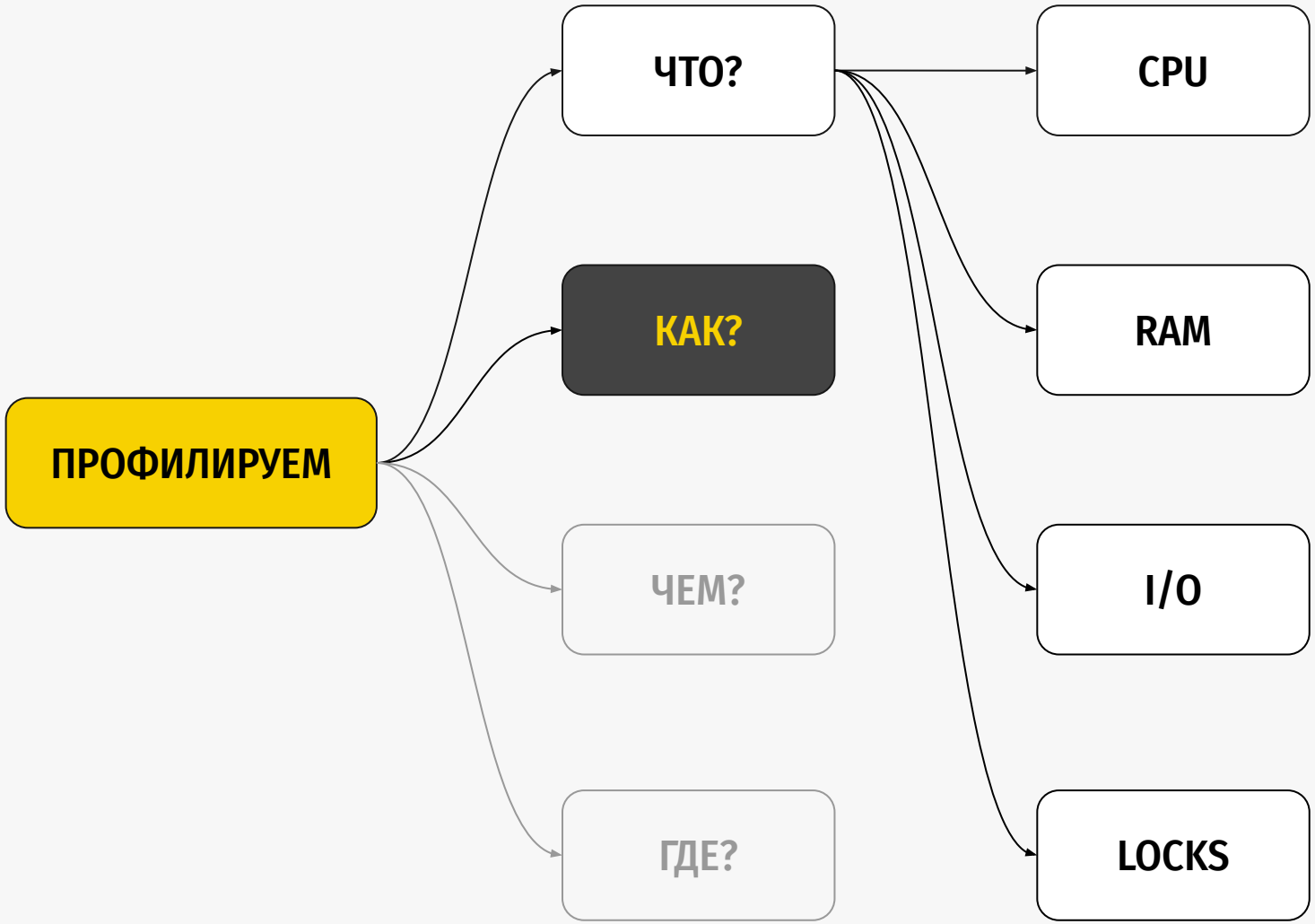
- VisualVM
- JProfiler
- Digma.AI
- Alibaba Arthas
- NetBeans Profiler
- ...

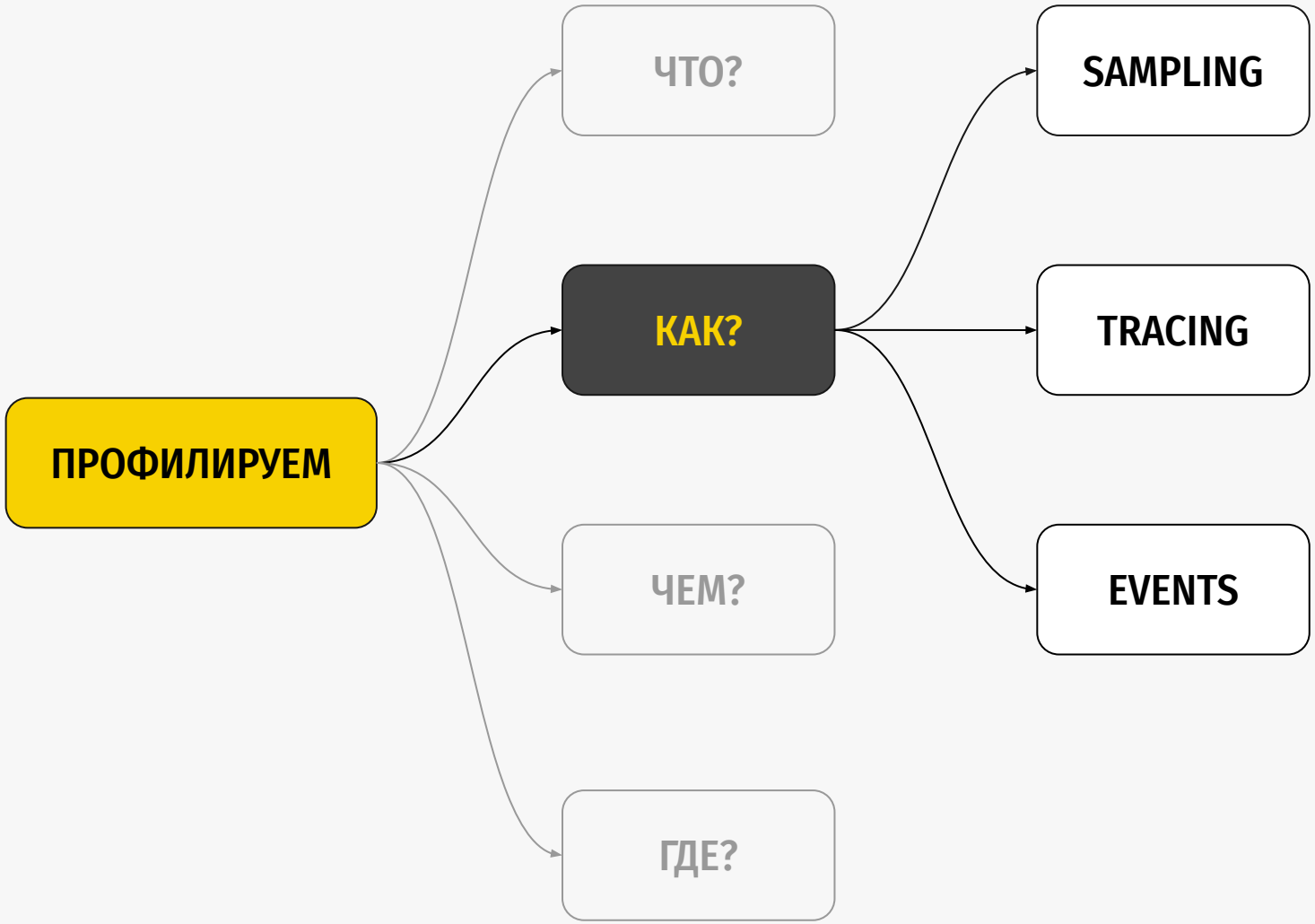
- OTEL Agent
- DataDog
- NewRelic
- Micrometer
- ...

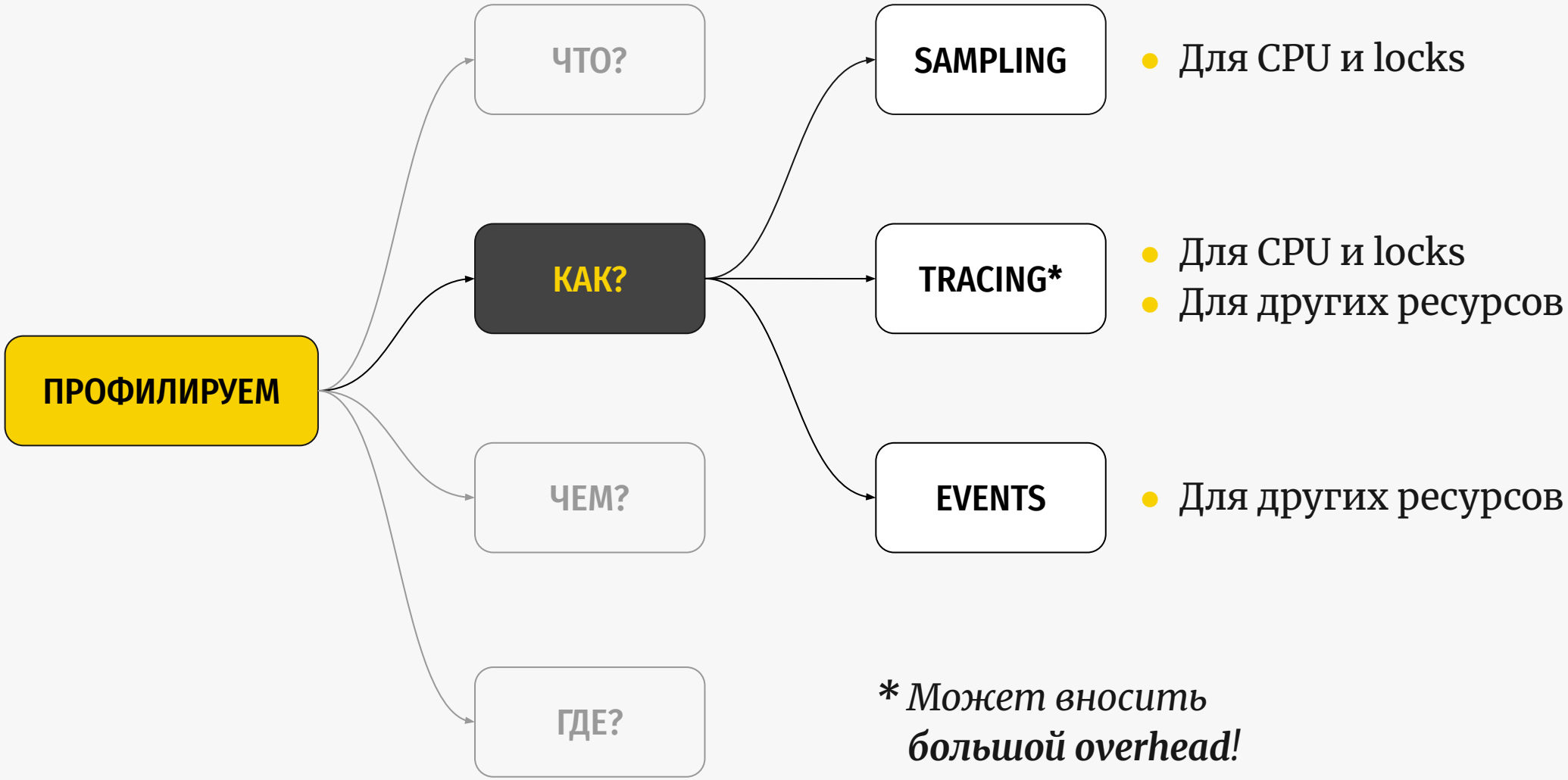
Вне фокуса доклада

ПОЧЕМУ НЕТ ЕДИНОГО СУПЕР ПРОФАЙЛЕРА

https://www.youtube.com/watch?v=7IkHIqPeFjY







* *Может вносить* ***большой overhead!***

ОБЩИЙ ПОДХОД

1. Подключить профайлер к приложению
2. Запустить профилирование
3. Выполнить проблемное действие ← *если знаешь какое*
4. Остановить профилирование
5. Сохранить результаты

ВАРИАНТЫ РЕЗУЛЬТАТОВ ПРОФИЛИРОВАНИЯ

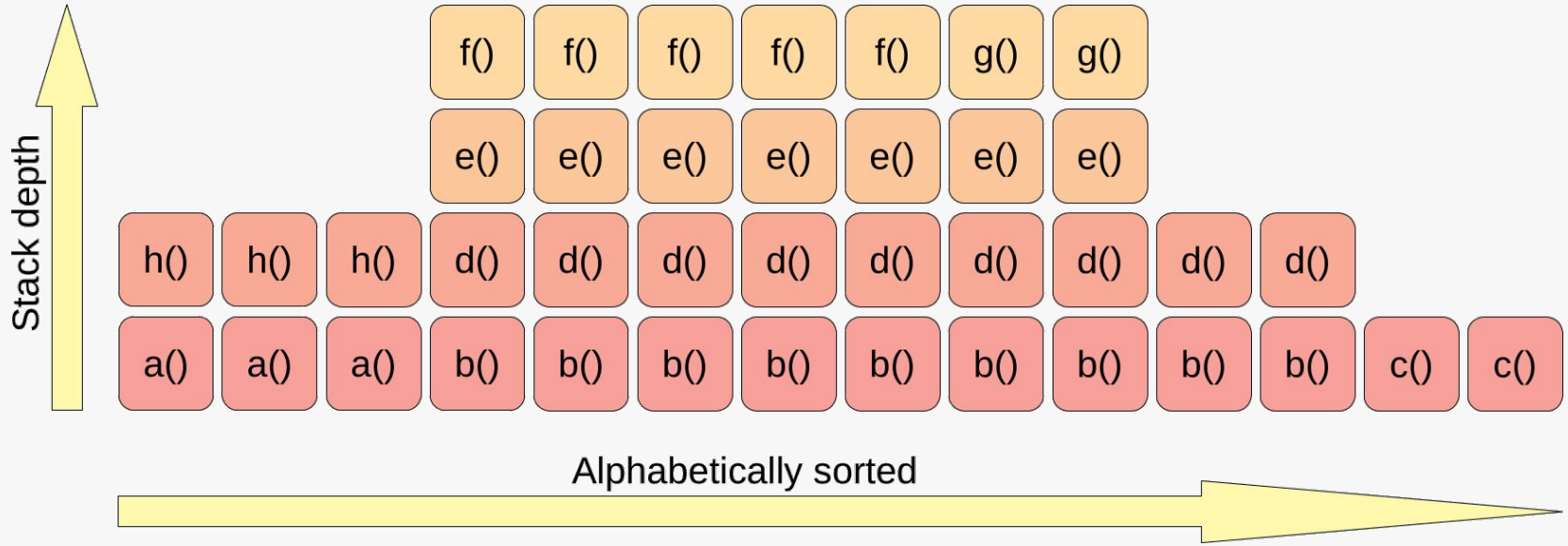
- **Обычный текст** (в т. ч. **CSV**)
- **Записи JFR**
  - Бинарные “логи” со встроенным сжатием
  - Поддерживают разнотипные события в одном файле
- **Flame Graphs** 🔥
  - Интерактивная визуализация множества стектрейсов
  - Как правило, в SVG

CPU profile

Produced by async-profiler

КАК СТРОИТСЯ FLAME GRAPH

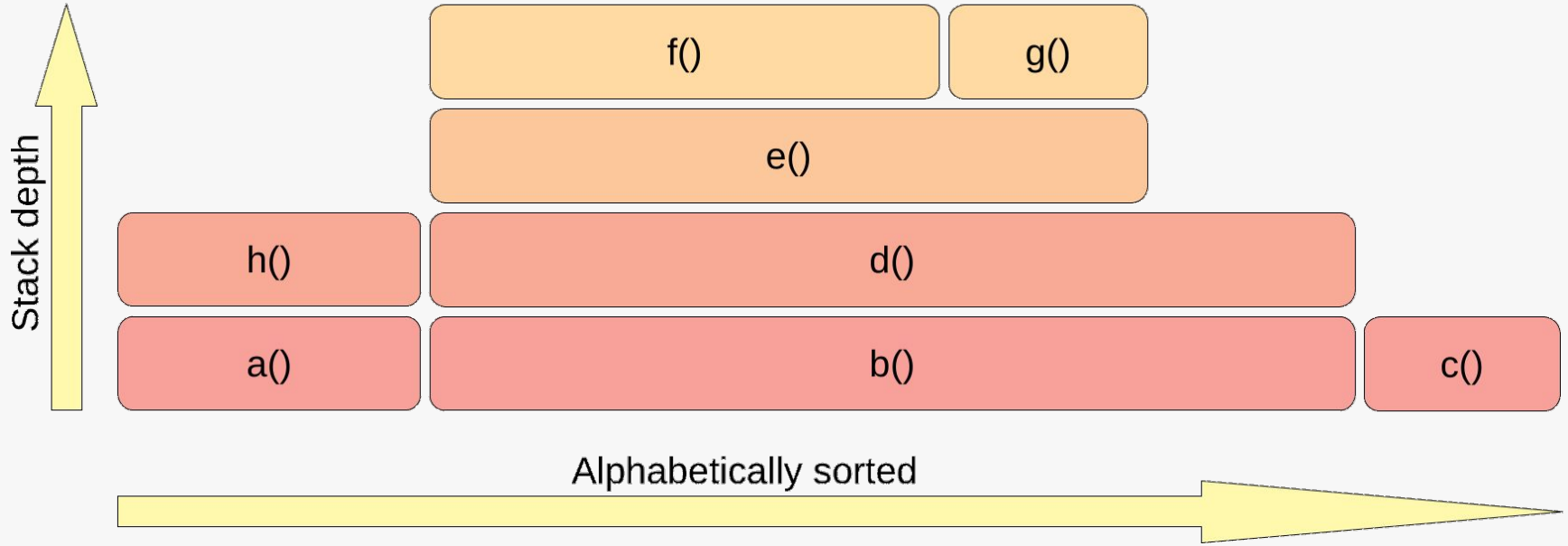
1. Выписать стектрейсы в алфавитном порядке нижнего этажа



https://krzysztofslusarski.github.io/2022/12/12/async-manual.html#flames

КАК СТРОИТСЯ FLAME GRAPH

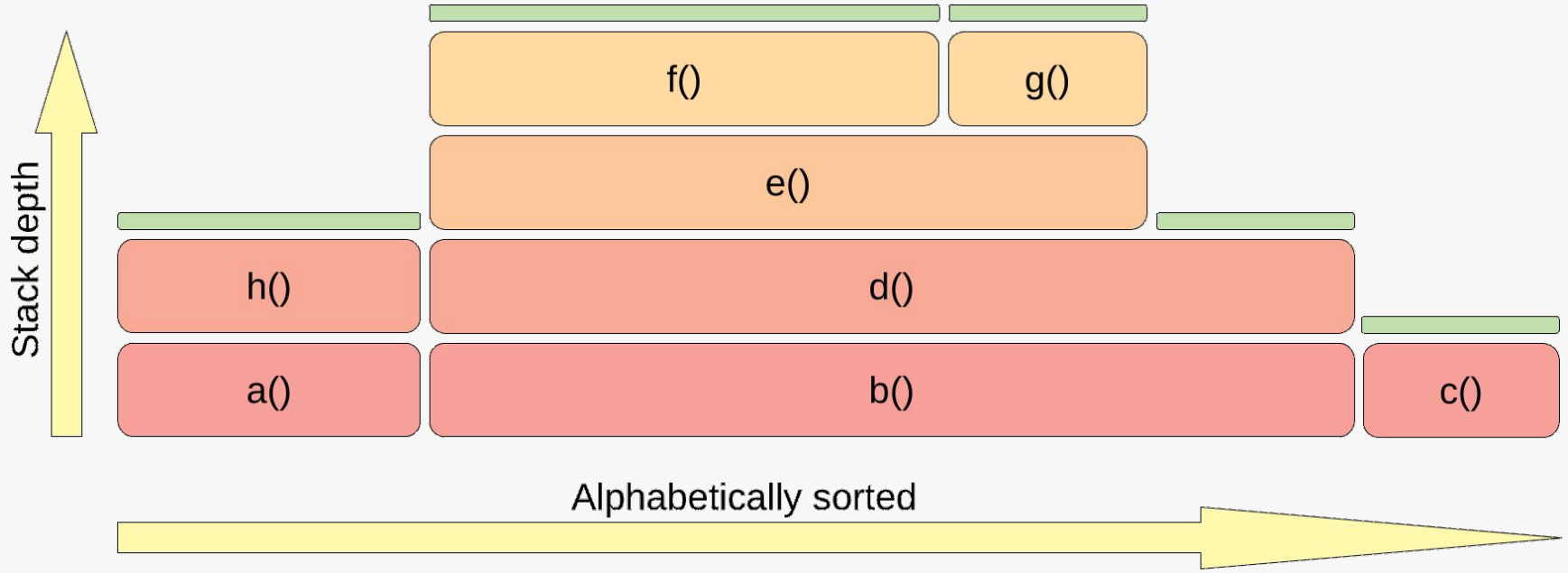
2. Объединить одинаковые соседние этажи



https://krzysztofslusarski.github.io/2022/12/12/async-manual.html#flames

КАК СТРОИТСЯ FLAME GRAPH

3. Обозначить верхний этаж по измеряемому ресурсу



https://krzysztofslusarski.github.io/2022/12/12/async-manual.html#flames

ПРИМЕРЫ ИЗМЕРЯЕМЫХ РЕСУРСОВ

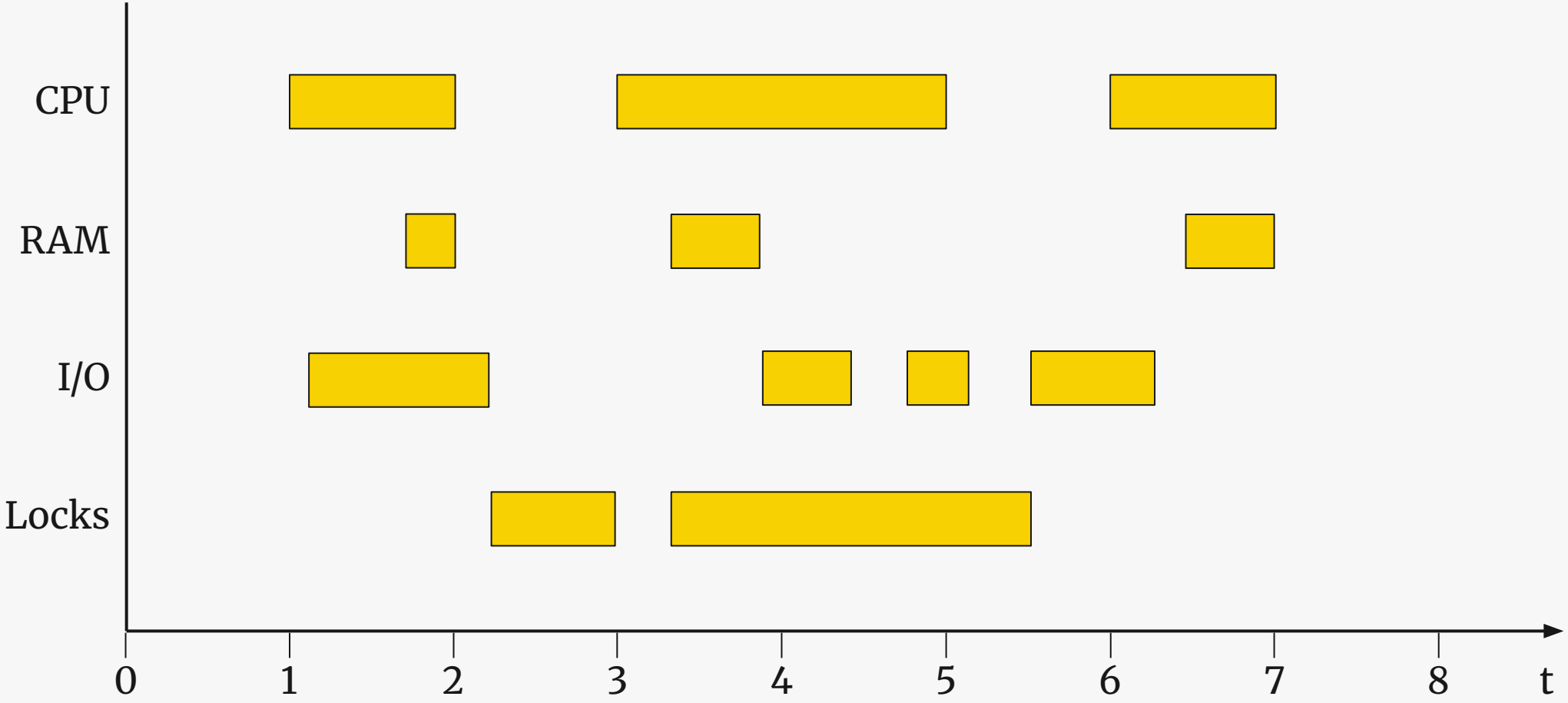
- **Размеры/количество** аллоцированных объектов в куче
- **Объем** переданных/записанных данных
- Время, проведенное на процессоре
- Время, прошедшее за наблюдение

} *Чё? o_O*

CPU TIME VS WALL CLOCK TIME

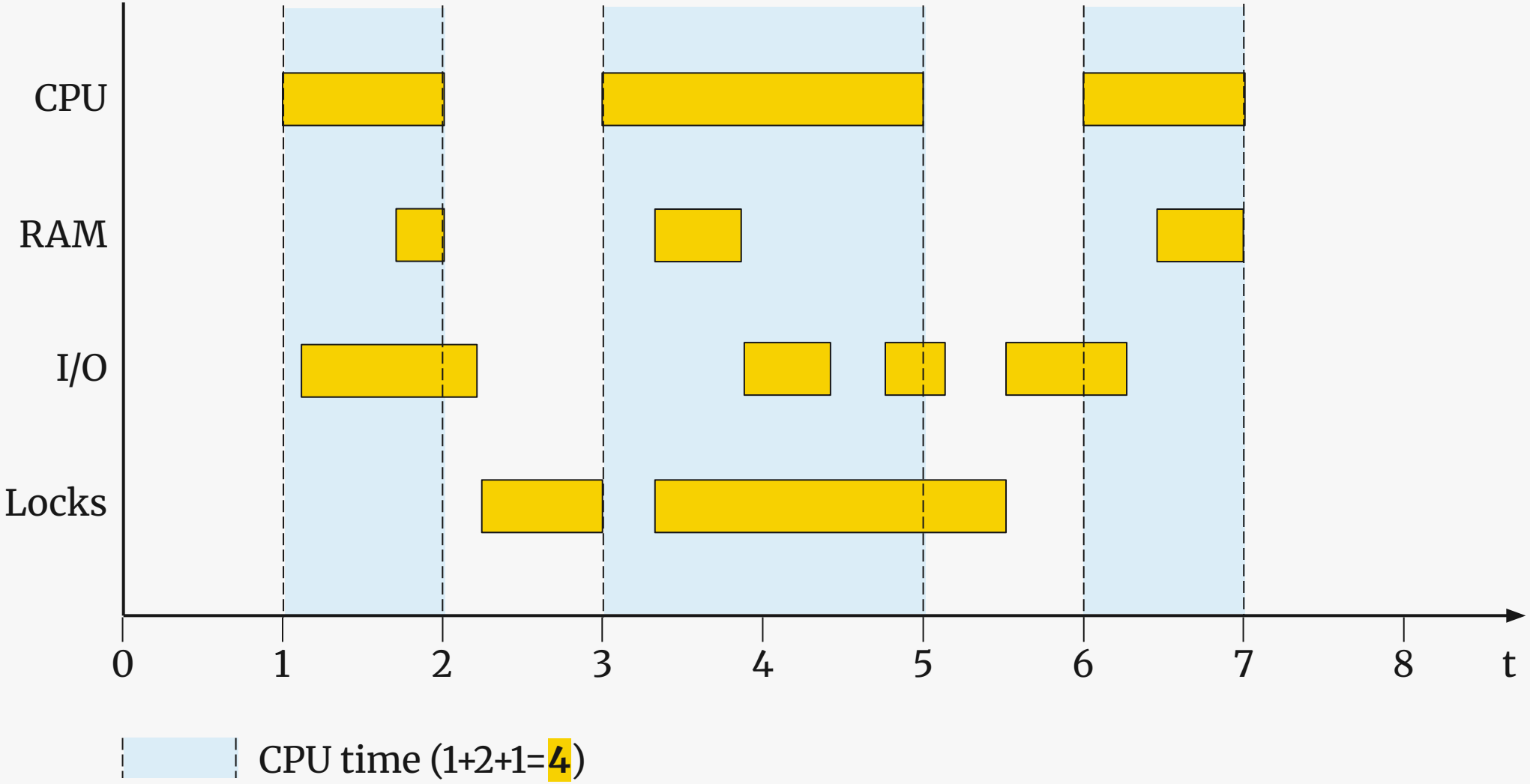
- CPU time – время, проведенное потоком на процессоре
- Wall clock time – общее время потока
  - Включая ожидание (в т. ч. I/O)
  - Включая блокировки
  - Включая сон

ПРОФИЛЬ ОДНОГО ВЫЗОВА МЕТОДА

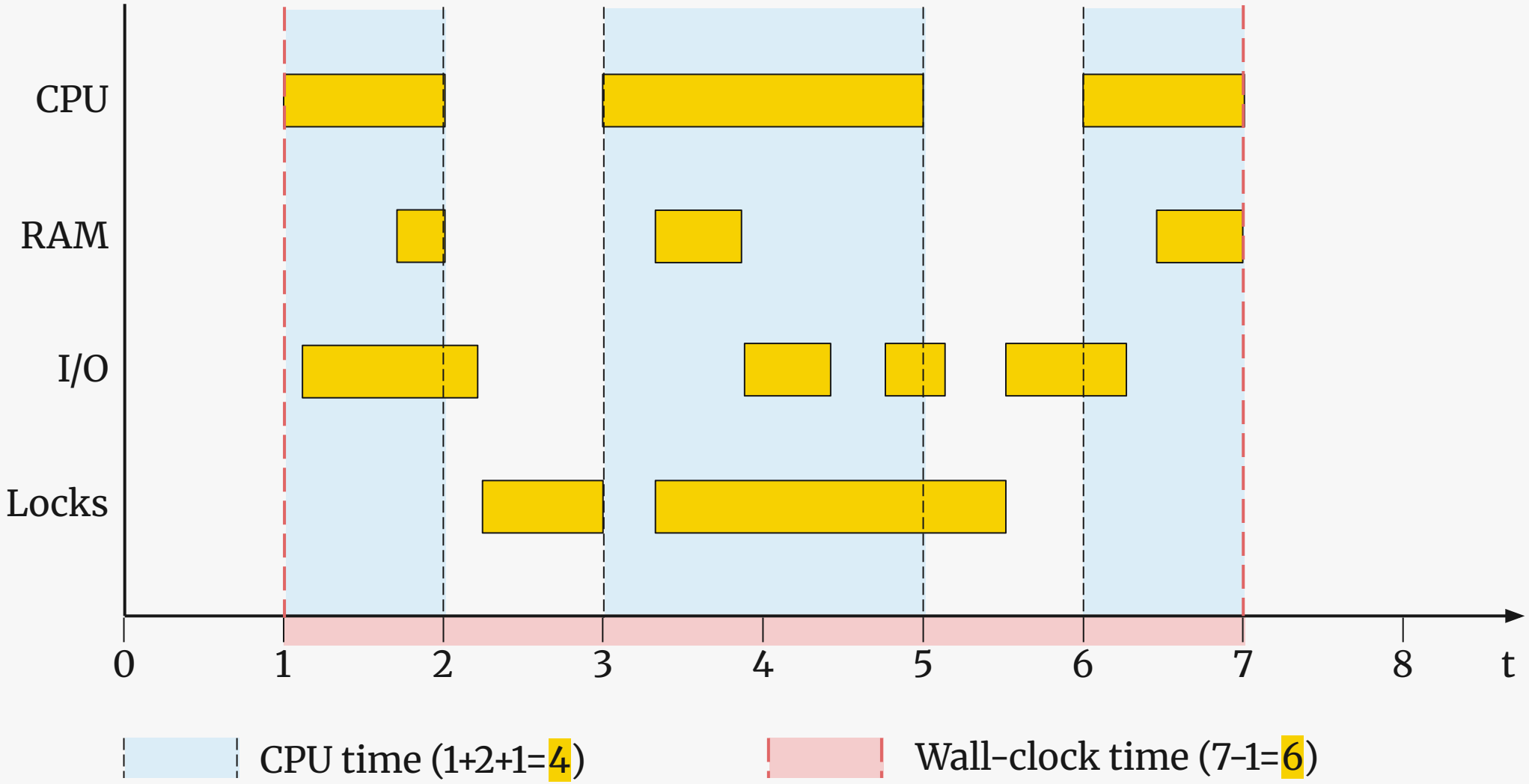


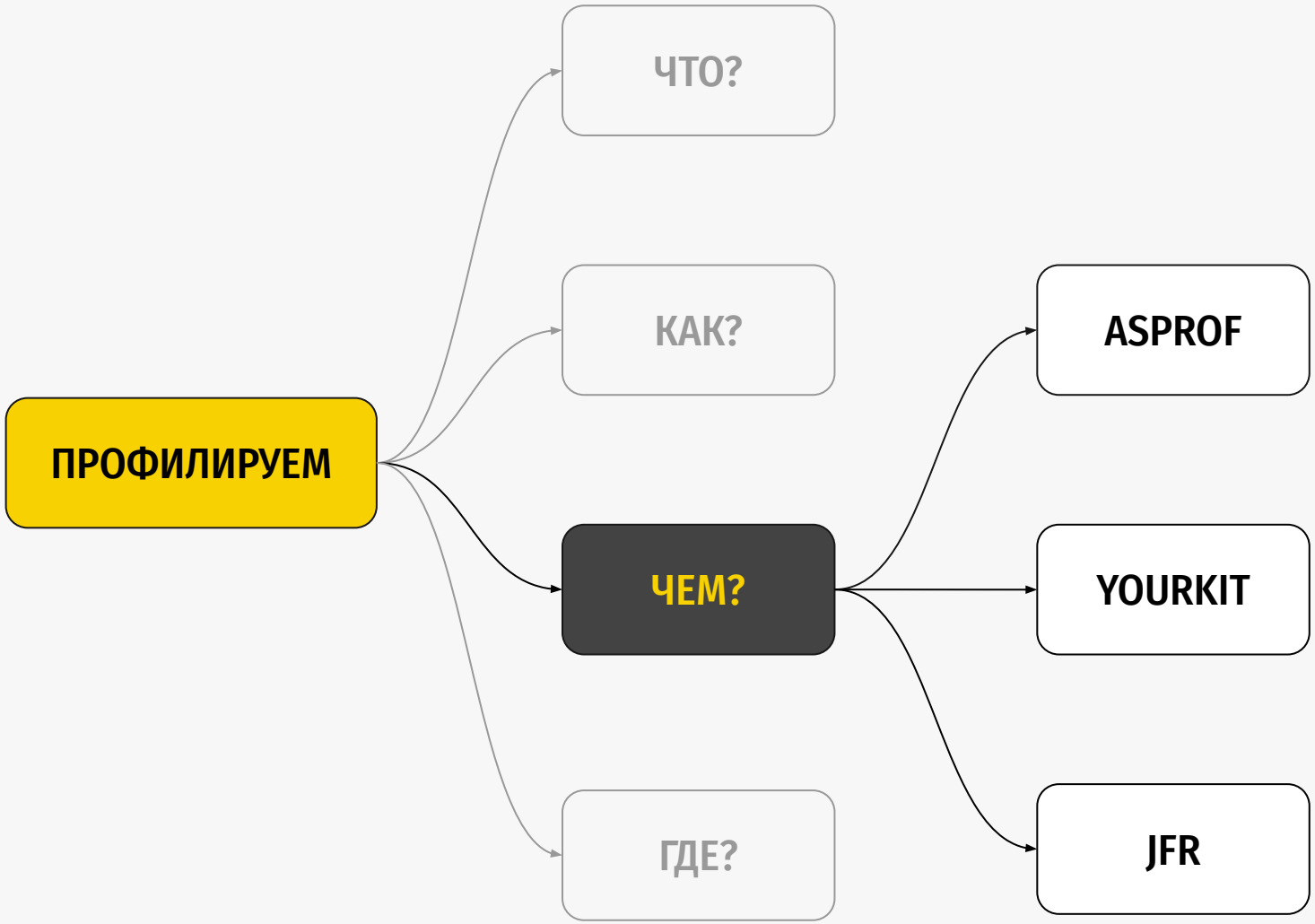
Периоды владения ресурсами за 1 вызов метода

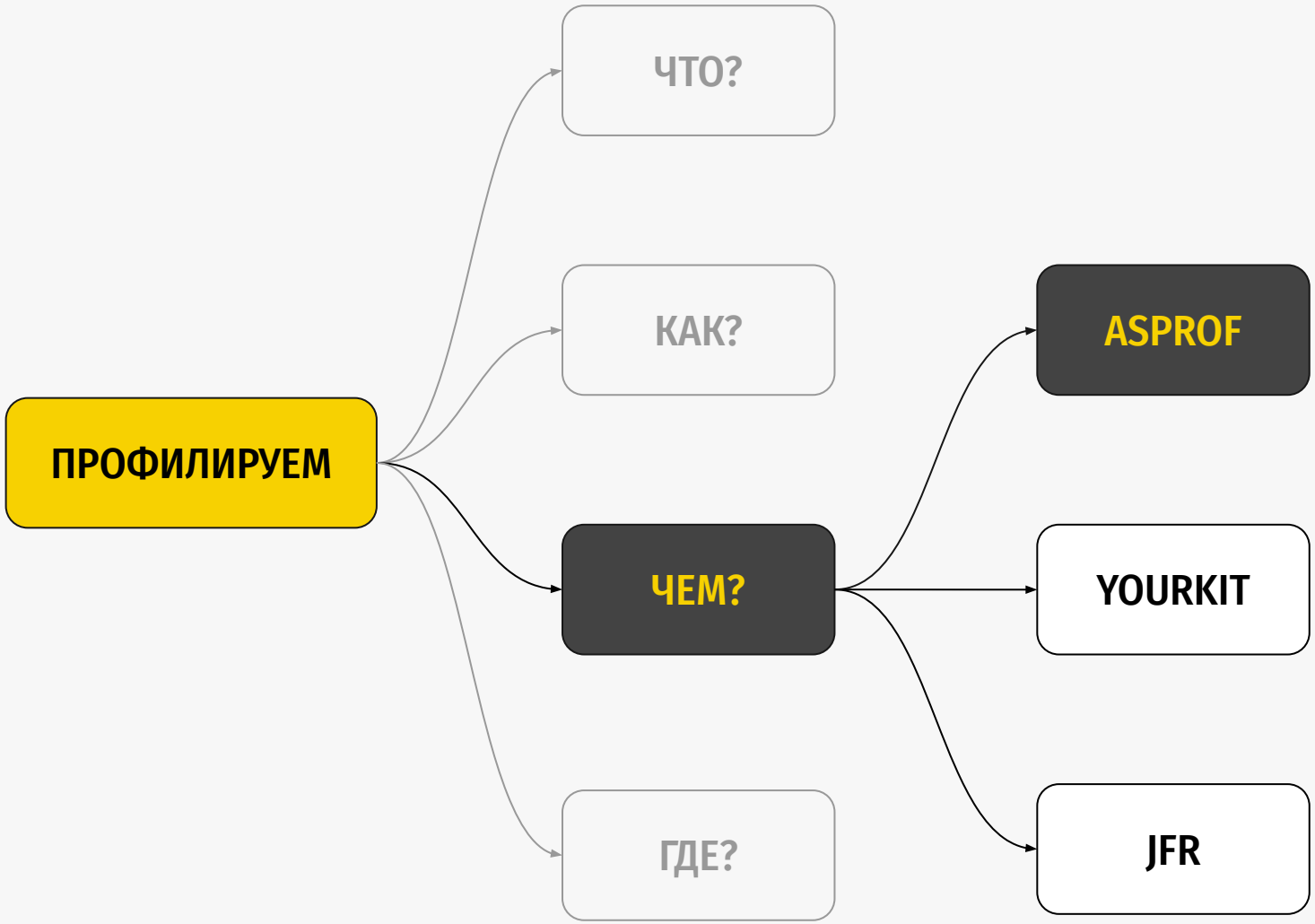
CPU TIME



CPU TIME VS WALL CLOCK TIME







ASYNC-PROFILER: ОБЩЕЕ

- **Доступность**: open source
- **Разработчик**: Андрей Паньгин и контрибьюторы
- **Тип**: sampling (в основном)
- **Интерфейсы**:
  - CLI
  - Java API
  - GUI (IDEA Ultimate)

AsyncGetCallTrace (AGCT) is a non-standard extension of HotSpot JVM ...

async-profiler ... got its name after this function.

async-profiler GitHub

ASYNC-PROFILER: ПОДКЛЮЧЕНИЕ (CLI)

- **При старте JVM:**
  - `java -agentpath:/path/to/asprof.so=start,event=cpu …`
- **На лету:**
  - `asprof -d 30 -e cpu -f profile.html <pid>`

  ⚠️ Для этого режима JVM лучше запускать с опциями:

  `-XX:+UnlockDiagnosticVMOptions -XX:+DebugNonSafepoints`

ASYNC-PROFILER: ОСОБЕННОСТИ РАБОТЫ

- Async-profiler собирает callstack'и вне safepoint'ов
- Ему нужно сопоставлять их с исходным кодом
- Он берет данные у JVM
- Если она запущена без `+DebugNonSafepoints`, их может не быть
- **Результат: профайлер теряет в точности**
  - Например, не видит простые inlined-методы
- Подробнее: Why JVM modern profilers are still safepoint biased?

ASPROF: ЗАПУСК С JMH

jpoint.ru

ДОКЛАД Performance JVM Tooling

04.04 / 19:30 — 20:15 UTC+7 (Asia/Novosibirsk) ЗАЛ 1

JMH: вводный курс по микробенчмаркам

Доклад для тех, кто хочет начать использовать JMH для написания микробенчмарков или углубить свои знания этого инструмента: вы познакомитесь с базовыми принципами создания микробенчмарков и основными возможностями JMH.

RU

Спикеры

Григорий Кошелев
Контур

Биография Страница спикера →

ASYNC-PROFILER: CPU TIME

- **Включается** опцией `-e cpu`
- Почти не подвержен safepoint bias
- Частота сэмплирования: `-i 10ms`
- Включает нативные фреймы в стектрейсы

ПАЛИТРА FLAME GRAPH В ASPROF

Kernel	__x64_sys_futex
	do_syscall_64
	entry_SYSCALL_64_after_hwframe
Native	__lll_lock_wake
	pthread_mutex_unlock
C++	SharedRuntime::handle_ic_miss_helper_internal
	SharedRuntime::handle_ic_miss_helper
	SharedRuntime::handle_wrong_method_ic_miss
Native	ic_miss_stub
C1 compiled	java/util/ComparableTimSort.countRunAndMakeAscending
Interpreted	java/util/ComparableTimSort.sort
	java/util/Arrays.sort
Java inlined	java/util/Arrays.sort
Java compiled	java/util/ArrayList.sort
Match found	java/util/Collections.sort
	nonapi/io/github/classgraph/utils/CollectionUtils.sortIfNotEmpty

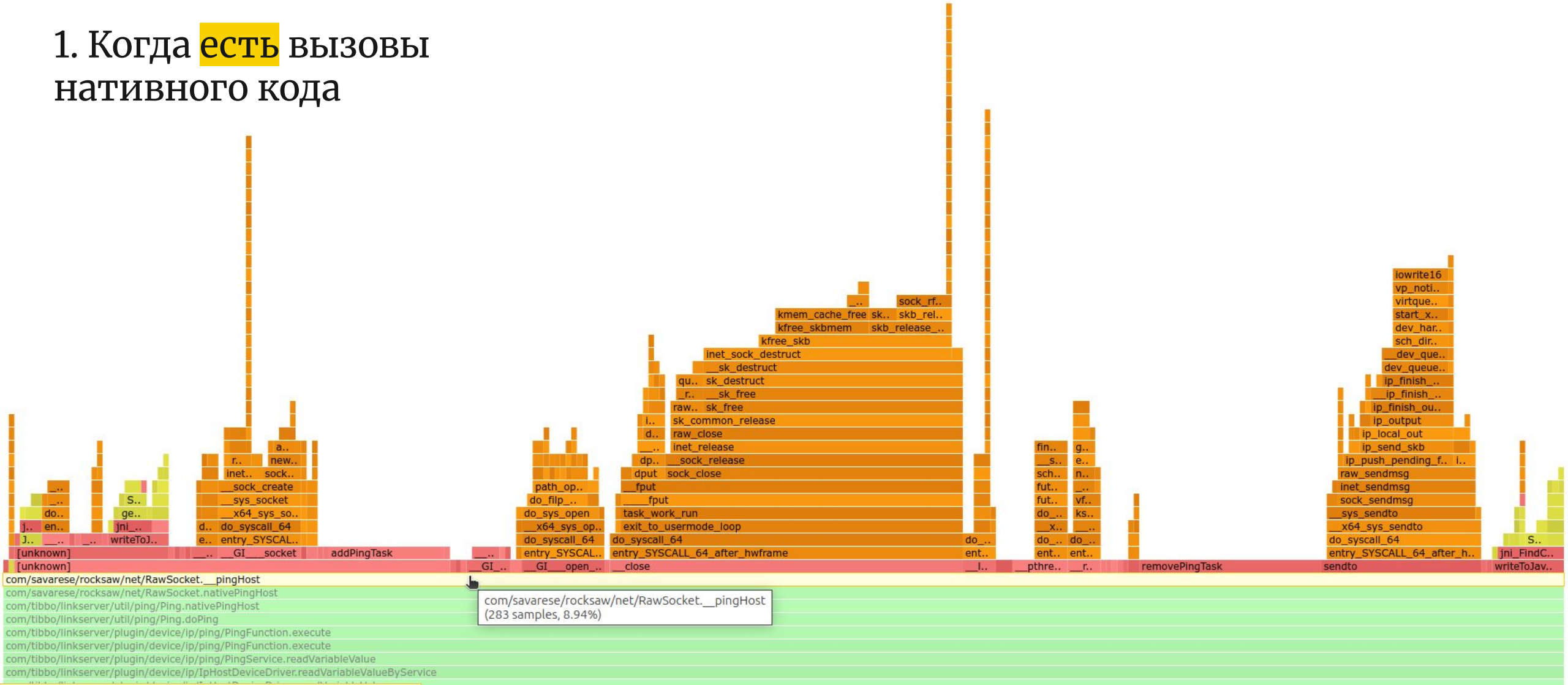
https://github.com/async-profiler/async-profiler/blob/master/docs/FlamegraphInterpretation.md

КОГДА НУЖНЫ НАТИВНЫЕ ФРЕЙМЫ

- Когда есть нативный код

КОГДА НУЖНЫ НАТИВНЫЕ ФРЕЙМЫ

1. Когда есть вызовы нативного кода

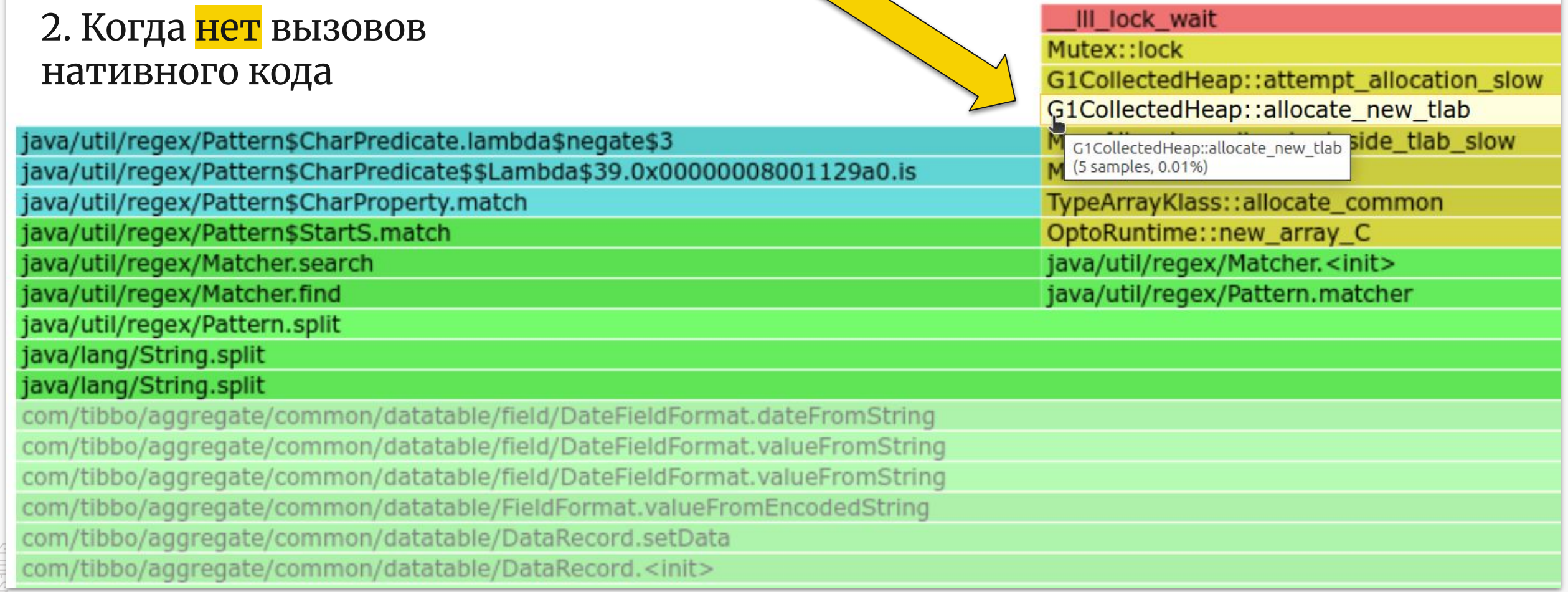


КОГДА НУЖНЫ НАТИВНЫЕ ФРЕЙМЫ

- Когда есть нативный код
- Когда нет нативного кода
  - но оказывается, он есть

КОГДА НУЖНЫ НАТИВНЫЕ ФРЕЙМЫ

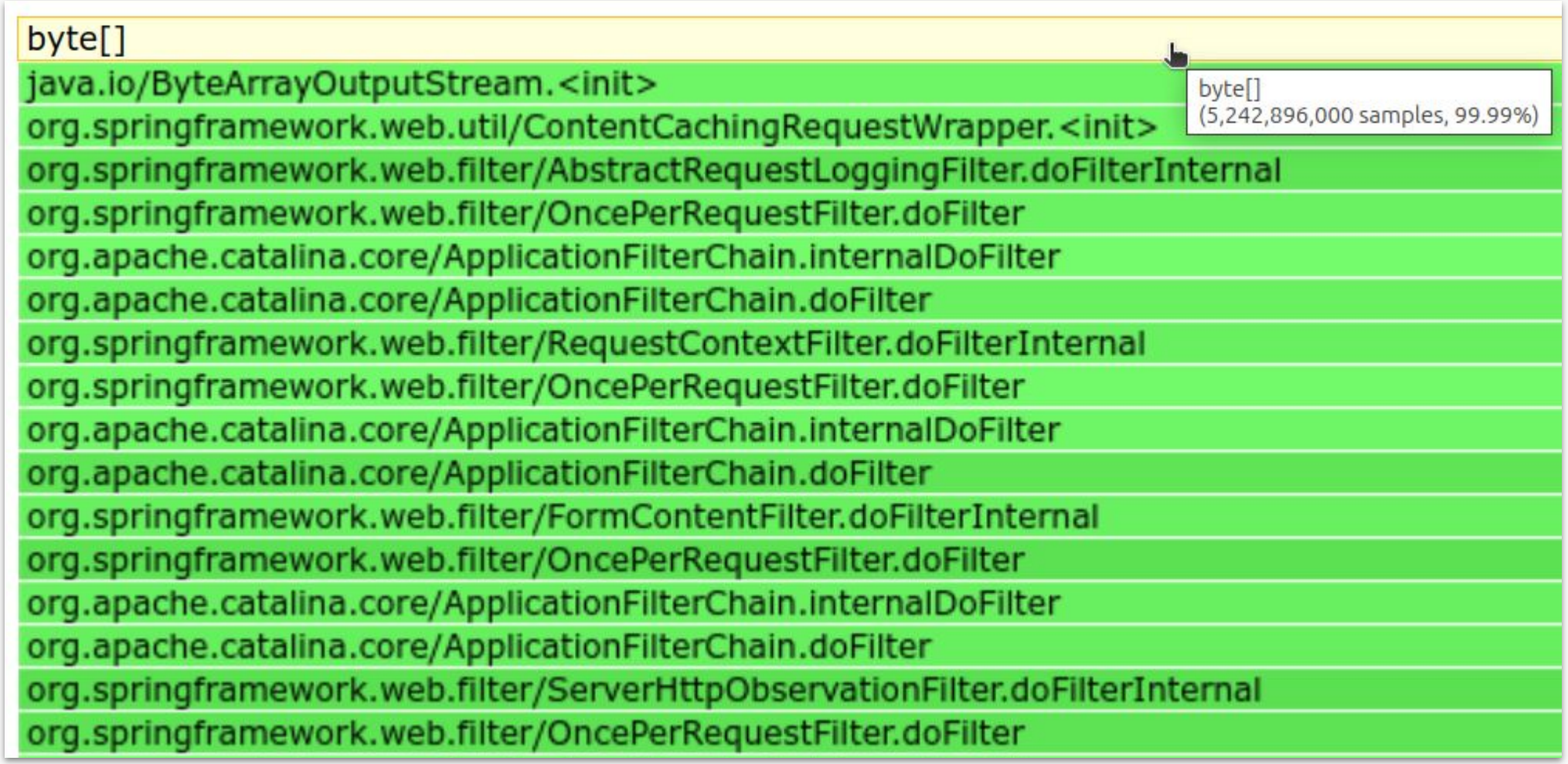
2. Когда нет вызовов нативного кода



ASYNC-PROFILER: ALLOCATION

- Включается опцией `-e alloc`
- Захватывает аллокации памяти **в куче**
- Не все, а выше порога: `--alloc 500k`
- Можно использовать с флагом `--live`
  - сохранит только те объекты, которые не были удалены к концу сеанса профилирования (для обнаружения утечек)

ПРИМЕР КРУПНОЙ АЛЛОКАЦИИ



https://krzysztofslusarski.github.io/assets/async-demos/alloc.html

КОГДА НУЖНЫ НАТИВНЫЕ ФРЕЙМЫ

- Когда есть нативный код
- Когда нет нативного кода
  - но оказывается, он есть

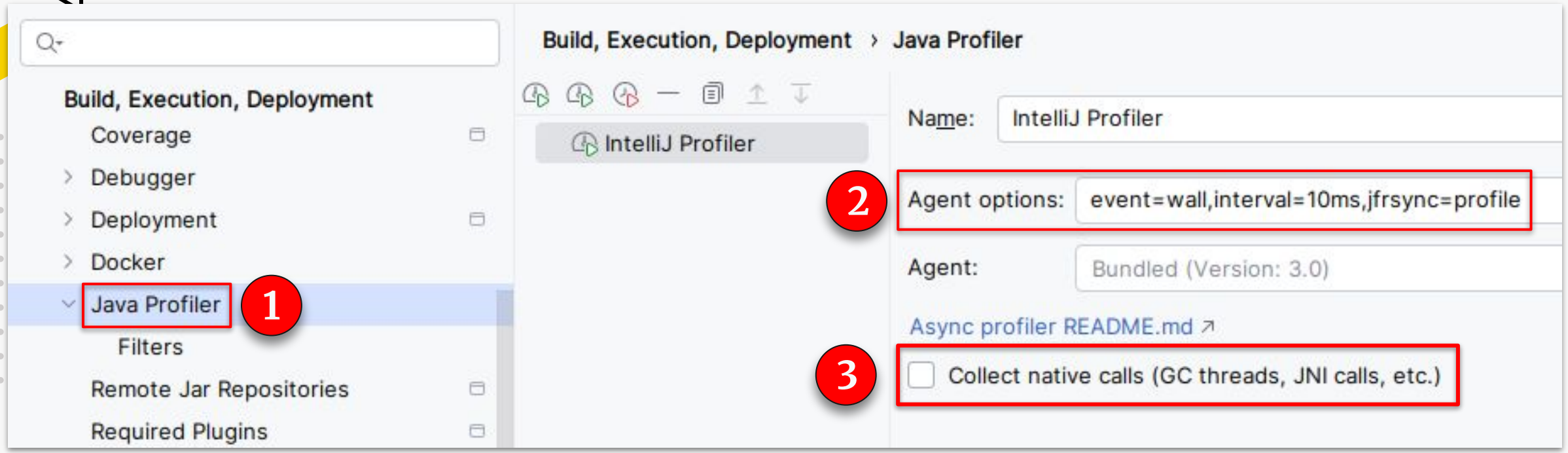
КОГДА НУЖНЫ НАТИВНЫЕ ФРЕЙМЫ

- Когда есть нативный код
- Когда нет нативного кода
  - но оказывается, он есть
- Когда метод ждет на уровне ОС
  - например, `java.net.PlainSocketImpl#accept0`

КОГДА НУЖНЫ НАТИВНЫЕ ФРЕЙМЫ

- Когда есть нативный код
- Когда нет нативного кода
  - но оказывается, он есть
- Когда метод ждет на уровне ОС
  - например, `java.net.PlainSocketImpl#accept0`
- Когда нужно увидеть активность GC
  - например, `GangWorkers`

ASYNC-PROFILER: ПОДКЛЮЧЕНИЕ (IDEA)

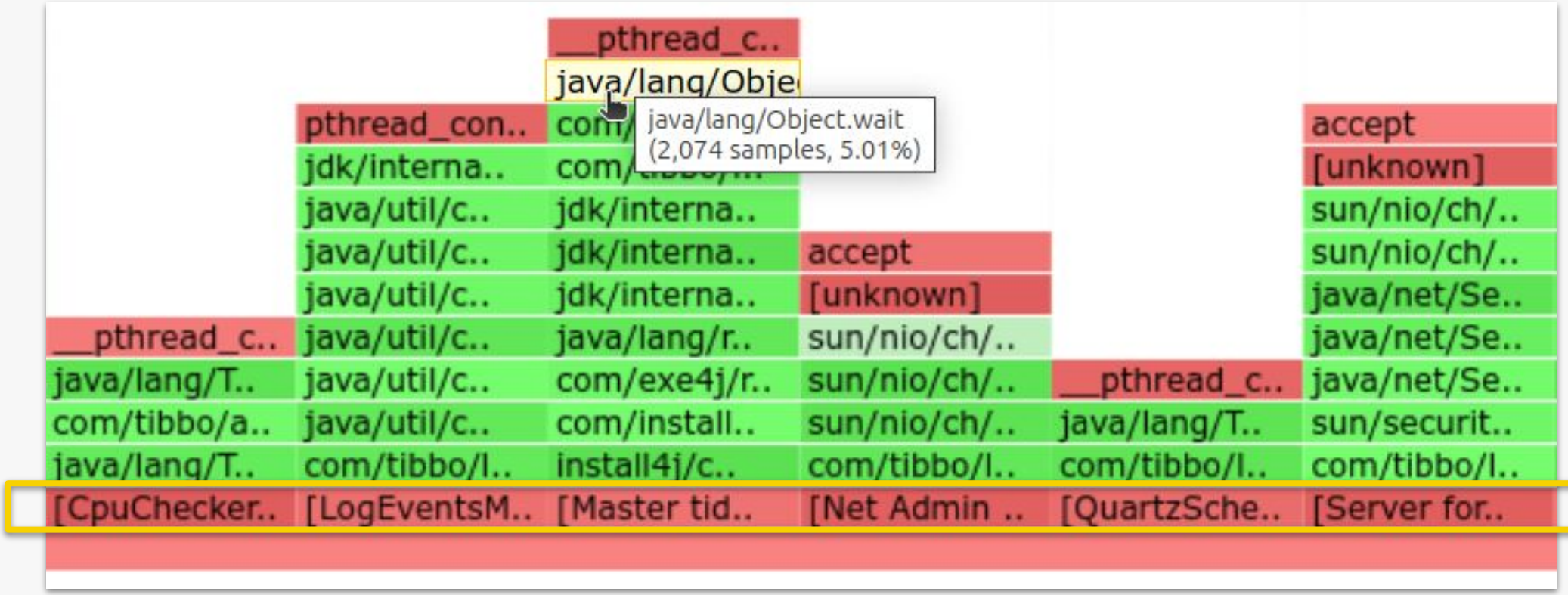


КАКОЙ РЕЖИМ ЛУЧШЕ, ЕСЛИ НЕПОНЯТНО, КУДА СМОТРЕТЬ?

ASYNC-PROFILER: WALL-CLOCK

- Включается опцией `-e wall`
- Захватывает потоки в любом состоянии
- Подходит для выяснения общих затрат времени
- Частота сэмплирования: `-i 10ms`
- Лучше использовать с флагом `-t` (threads)

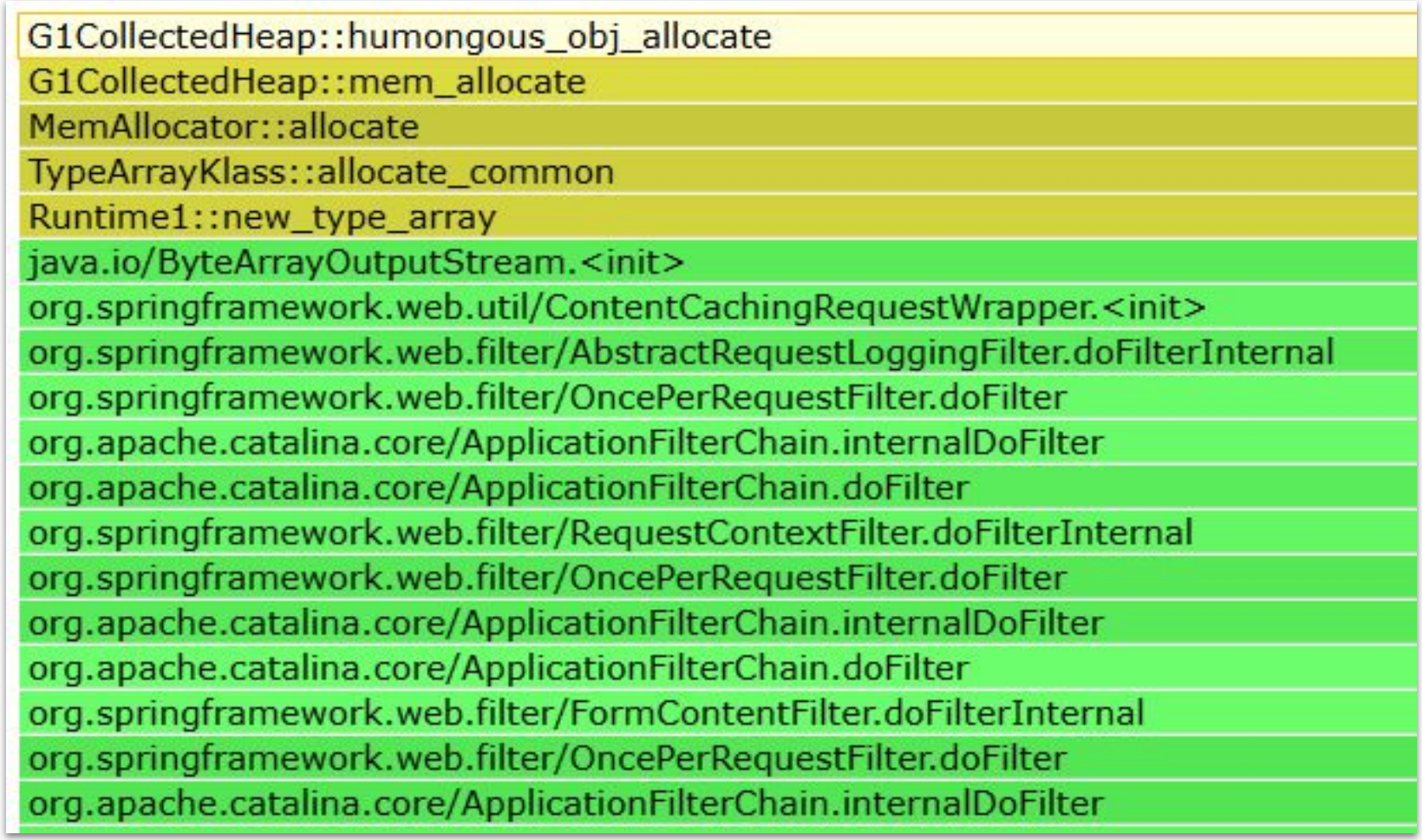
ПРИМЕР РАЗДЕЛЕНИЯ НА ПОТОКИ



ASYNC-PROFILER: ДРУГИЕ РЕЖИМЫ

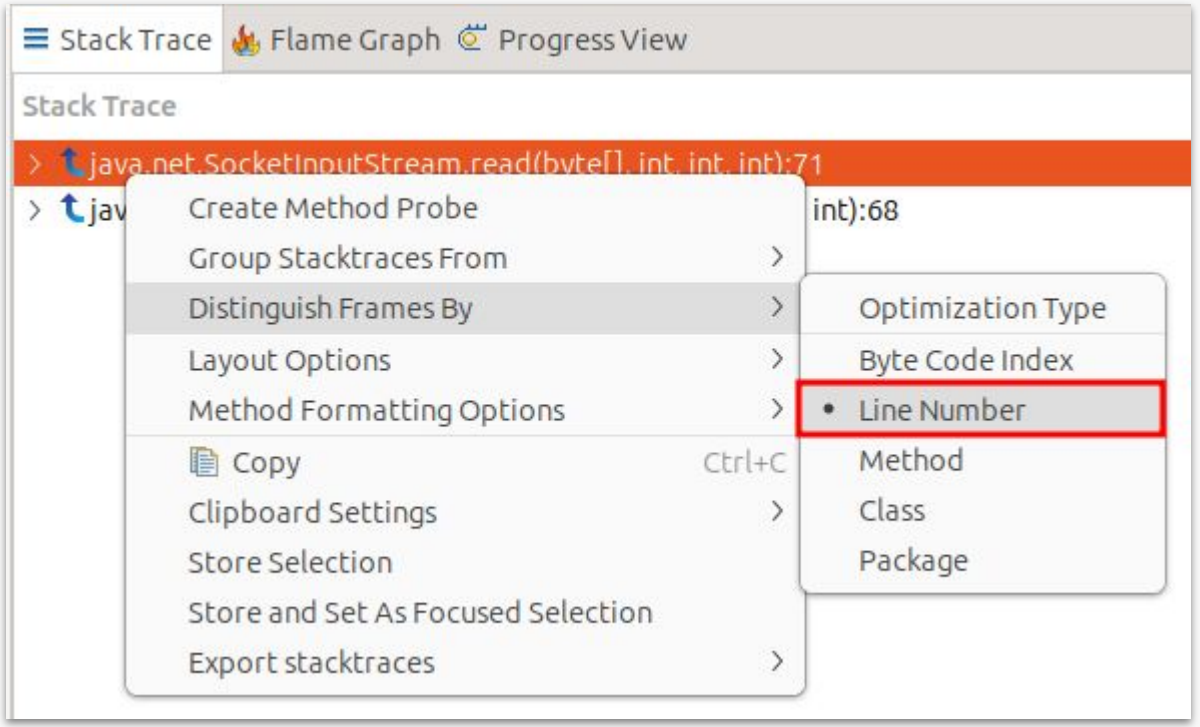
- Аллокации нативной памяти (`-e nativemem`)
- Блокировки (`-e lock`)
- Отдельные Java-методы (`-e ClassName.methodName`)
- И не Java тоже:
  - `Java_java_lang_ClassLoader_defineClass1`
  - `Java_java_lang_Throwable_fillInStackTrace`
  - `JVM_StartThread`
  - `G1CollectedHeap::humongous_obj_allocate`
- *Прочие*

ОТЛОВ ОГРОМНЫХ АЛЛОКАЦИЙ (G1 GC)



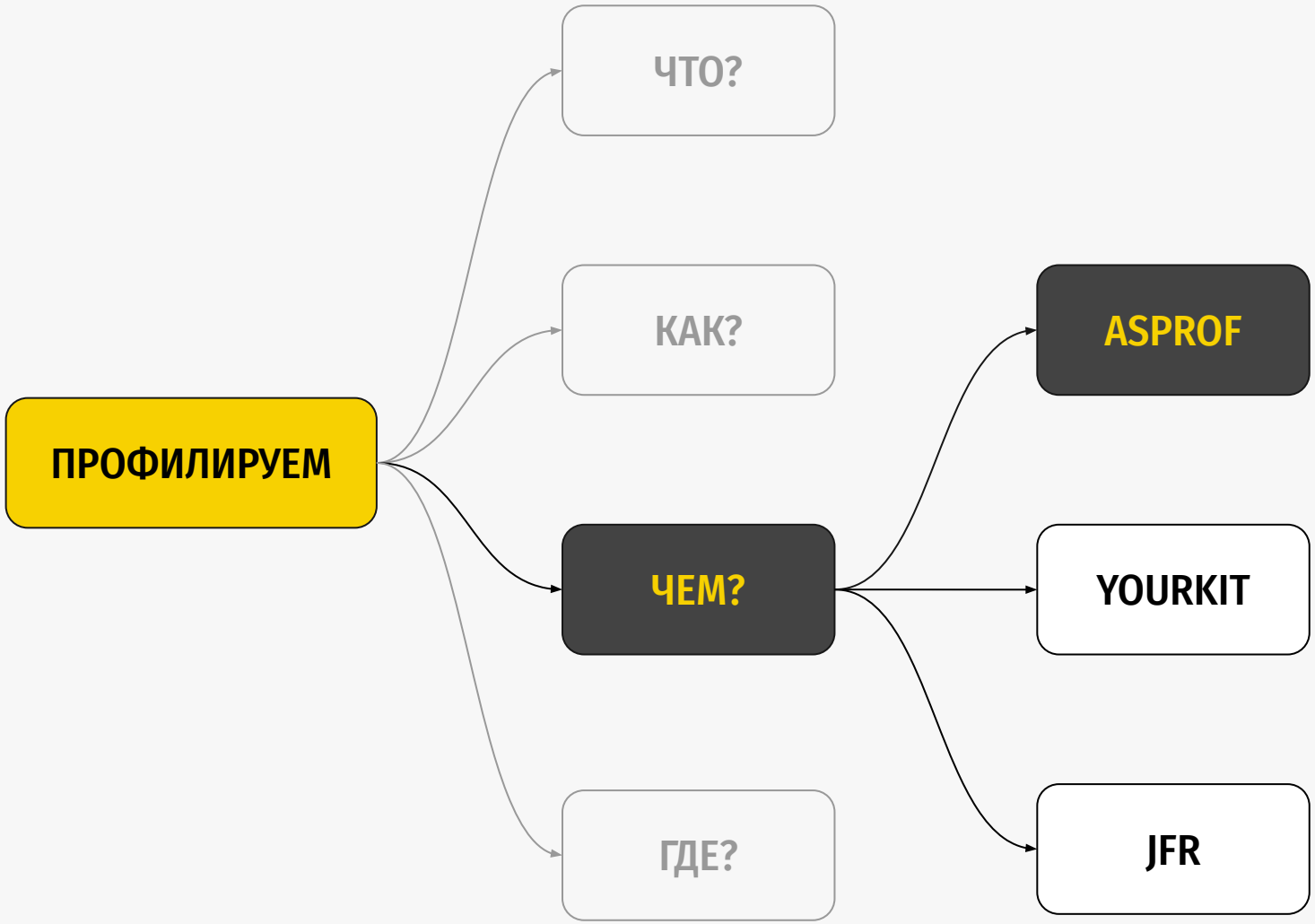
ASYNC-PROFILER: ГДЕ НОМЕРА СТРОК?

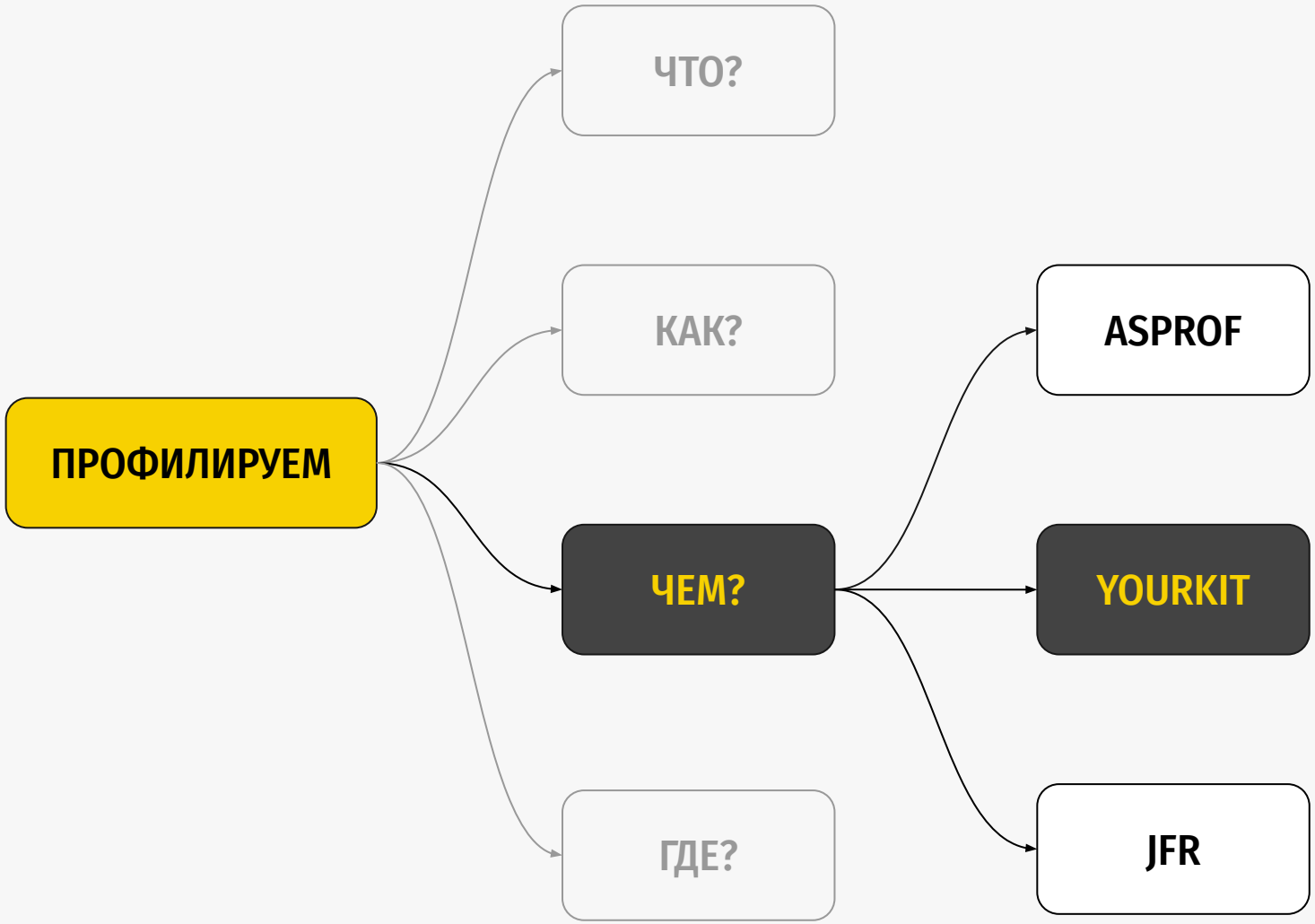
Формат экспорта	Номера строк
Plain text	✖
Деревья вызовов (HTML)	✖
Flame Graphs (SVG)	✖
JFR-записи (бинарный)	✅



ASYNC-PROFILER: ЧЕГО НЕ ХВАТАЕТ

- Как получить количества вызовов методов?
- Как профилировать ввод-вывод?
- Как подключаться удаленно?
- Как мониторить “на лету”?





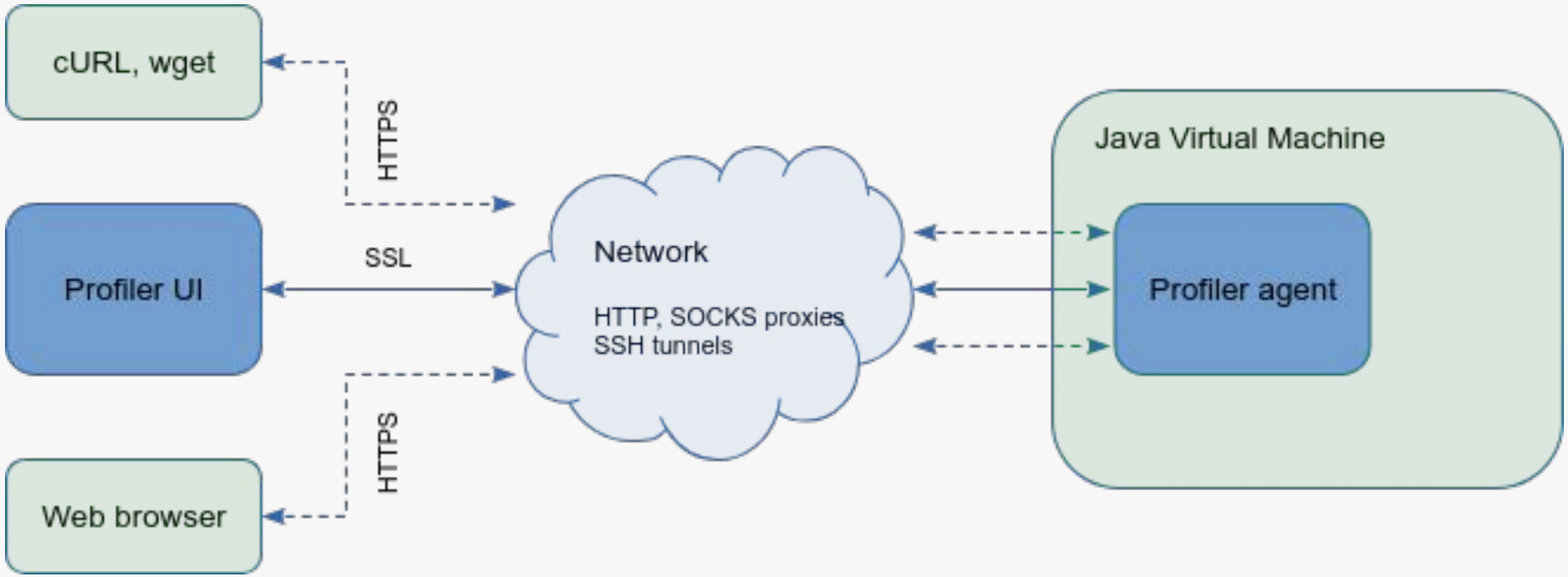
YOURKIT: ОБЩЕЕ

- **Доступность**: платные лицензии
- **Разработчик**: YourKit GmbH
- **Тип**: sampling & tracing
- **Интерфейсы**:
  - GUI
  - CLI
  - HTTP API
  - Java API

The [open source] license is granted to developers of non-commercial Open Source projects, with an established and active community.

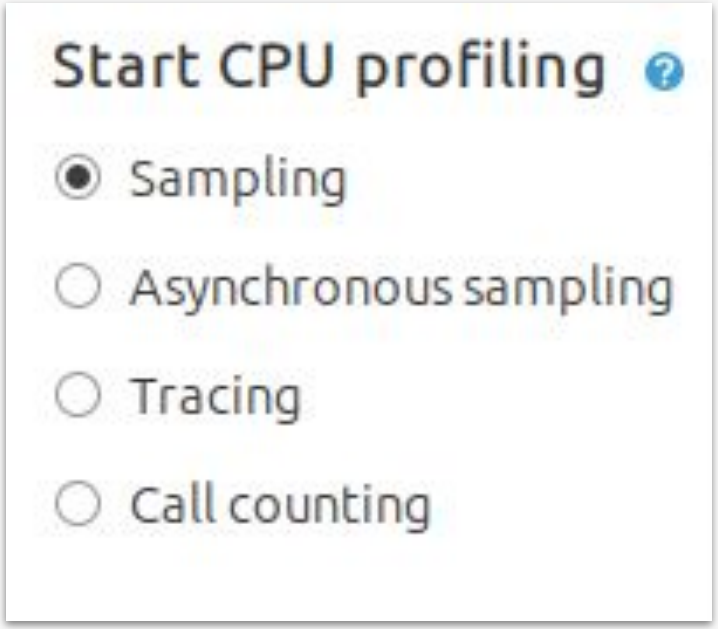
yourkit.com

YOURKIT: АРХИТЕКТУРА



https://www.yourkit.com/docs/java-profiler/latest/help/architecture.jsp

YOURKIT: РЕЖИМЫ ДЛЯ CPU



← Распадается еще на два

YOURKIT: CPU TRACING

- Захватывает моменты входа и выхода в/из методов
- Сохраняет точные длительности и количества вызовов
- Может применяться с **CPU time** и **wall-clock time**
- По умолчанию – адаптивный

YOURKIT: CPU ADAPTIVE TRACING

- Исключает короткие, но часто вызываемые методы
- Основывается на статистике по ходу профилирования
- Отражается в результатах:

DemoApp.java:87 **DemoApp.drawDemo**(Graphics2D)	30,041	6 %	493
DemoApp.java:72 sun.java2d.**SunGraphics2D.clip**(Shape)	28,336	6 %	493
SunGraphics2D.java:2059 sun.java2d.**SunGraphics2D.intersectShapes**(Shape, Shape, boolean,	28,221	6 %	493
SunGraphics2D.java:463 sun.java2d.**SunGraphics2D.intersectRectShape**(Rectangle2D, Sha	28,221	6 %	493
SunGraphics2D.java:509 sun.java2d.**SunGraphics2D.intersectByArea**(Shape, Shape, bc	28,218	6 %	350
SunGraphics2D.java:542 java.awt.geom.**Area.<init>**(Shape)	26,271	5 %	350
Area.java:126 java.awt.geom.**Area.pathToCurves**(PathIterator)	26,270	5 %	350
Area.java:195 <...> sun.awt.geom.**AreaOp.calculate**(Vector, Vector)	26,209	5 %	350
Area.java:169 <...> sun.awt.geom.**Curve.insertQuad**(Vector, double, double, c	≥ 26	0 %	≥ 35,272
Area.java:176 <...> sun.awt.geom.**Curve.insertCubic**(Vector, double, double, do	7	0 %	5,600

https://www.yourkit.com/docs/java-profiler/latest/help/tracing_settings.jsp

ТАК ГДЕ ЖЕ НАГЛЯДНЫЙ ПРОФИТ TRACING'А?

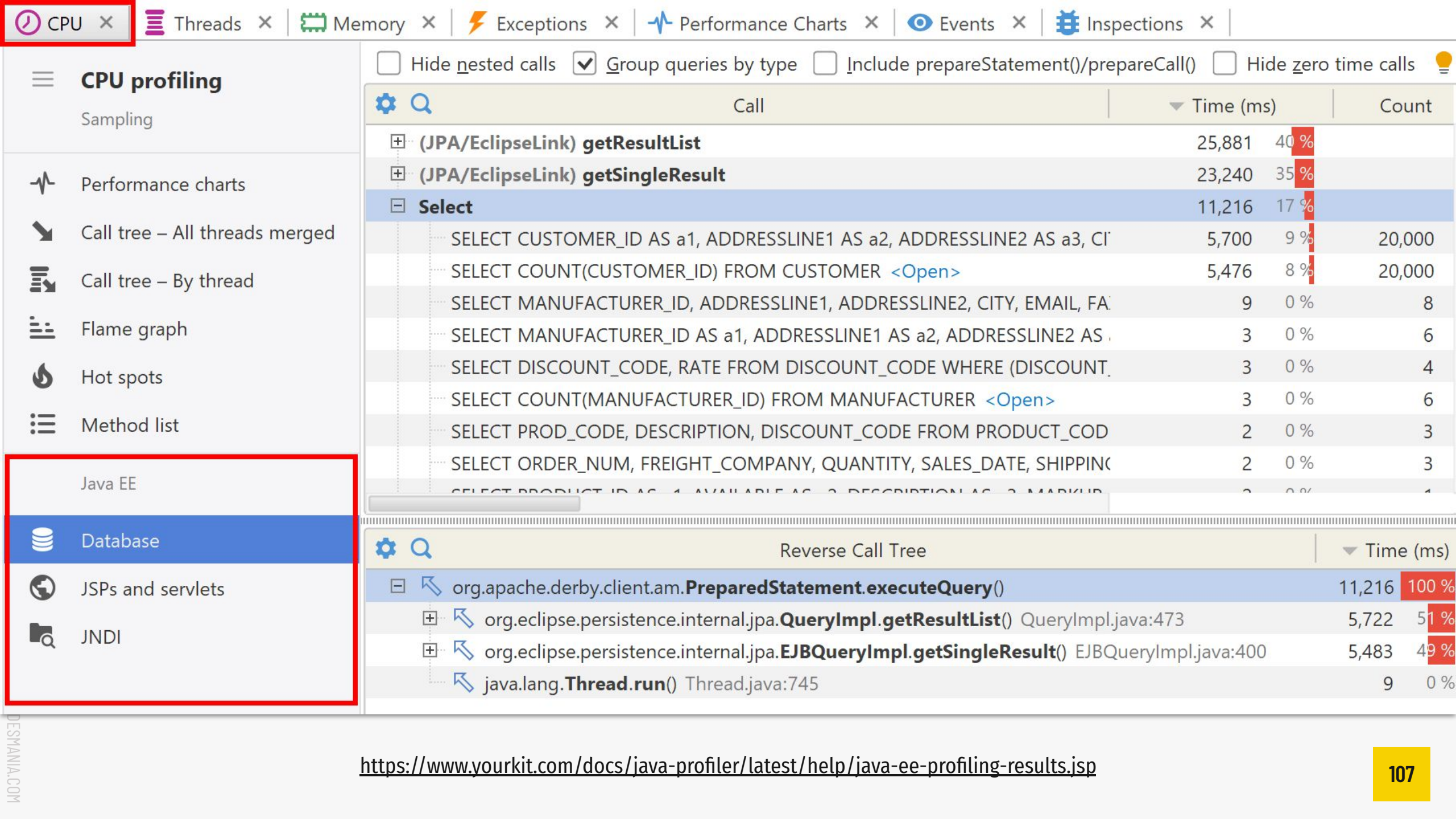
SAMPLING

Call Tree	Time (ms)		Samples
<All threads>	40,643	100 %	147,212
com.tibbo.linkserver.**Server.main**(String[])	30,979	76 %	2,843
<...> java.lang.**Thread.run**()	6,745	17 %	111,367
com.tibbo.linkserver.**Server.<clinit>**()	2,752	7 %	163
<...> sun.launcher.**LauncherHelper.checkAndLoadMain**(boolean, int, String)	69	0 %	6
<...> com.intellij.rt.execution.application.**AppMainV2$1.run**()	33	0 %	3,012
com.tibbo.aggregate.common.context.**EventDispatcher.run**()	22	0 %	1,411
<...> com.sun.management.internal.**GarbageCollectorExtImpl.createGCNotification**(long, String, String,	17	0 %	7
<...> com.sun.management.**GcInfo.<init>**(GcInfoBuilder, long, long, long, MemoryUsage[], MemoryUsage[	9	0 %	3

TRACING

Call Tree	Time (ms)		Avg. Time (ms)	Count
<All threads>	4,736,405	100 %		
java.lang.**Thread.run**()	4,146,780	88 %	37,358	111
Thread.java:840 <...> java.util.concurrent.**ThreadPoolExecutor$Worker.run**()	3,078,869	65 %	32,753	94
Thread.java:840 org.apache.tomcat.util.threads.**TaskThread$WrappingRunna**	775,699	16 %	64,641	12
TaskThread.java:63 org.apache.tomcat.util.threads.**ThreadPoolExecutor$V**	632,256	13 %	63,225	10
ThreadPoolExecutor.java:659 org.apache.tomcat.util.threads.**ThreadPc**	632,256	13 %	63,225	10
ThreadPoolExecutor.java:1176 <...> org.apache.tomcat.util.threads.**T**	623,010	13 %	1,384	450
ThreadPoolExecutor.java:1191 org.apache.tomcat.util.net.**SocketProce**	9,238	0 %	20	450
ThreadPoolExecutor.java:1192 <...> org.apache.tomcat.util.threads.**Thr**	2	0 %	< 0.1	450

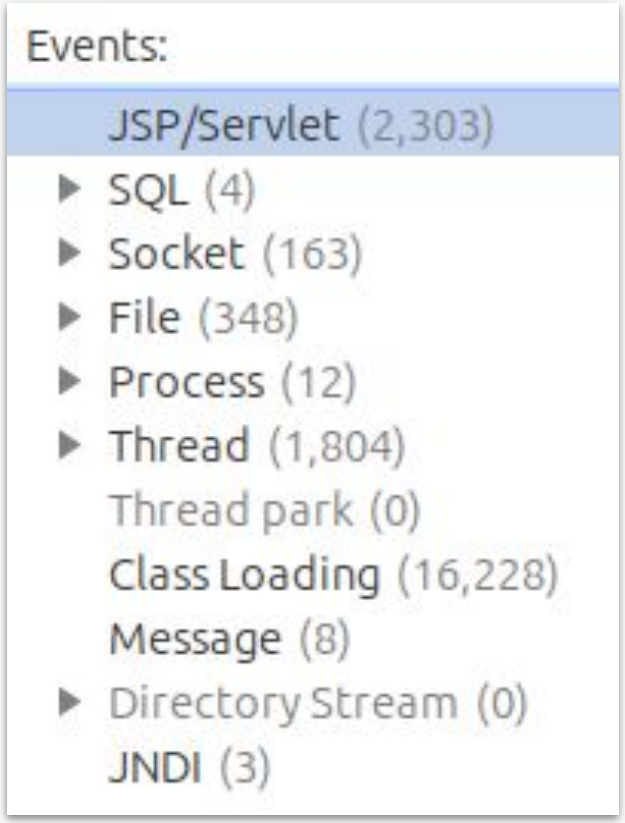
ЧТО ТАМ ПРО ПУТЬ К HIGH-LEVEL ПРОБАМ?

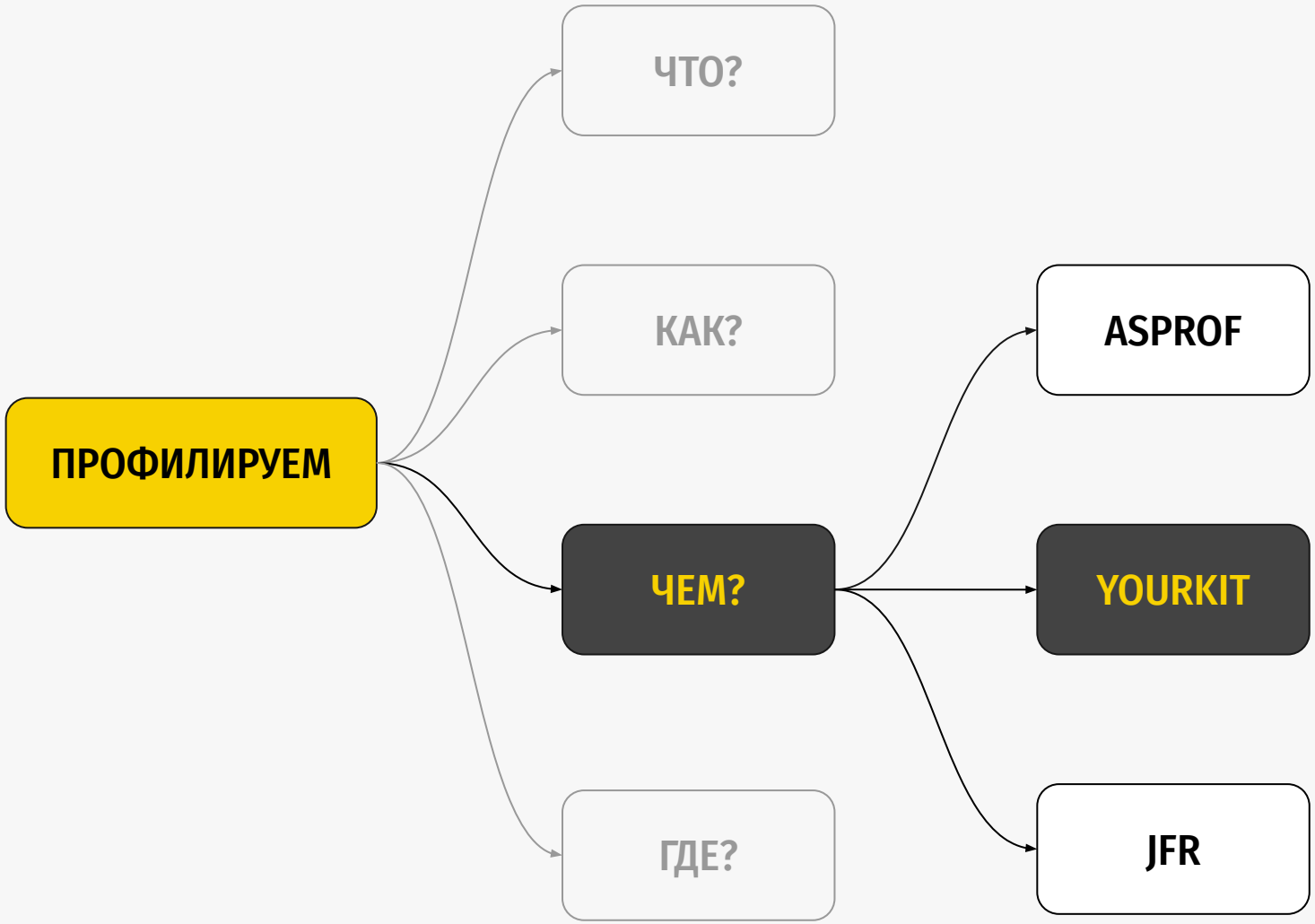


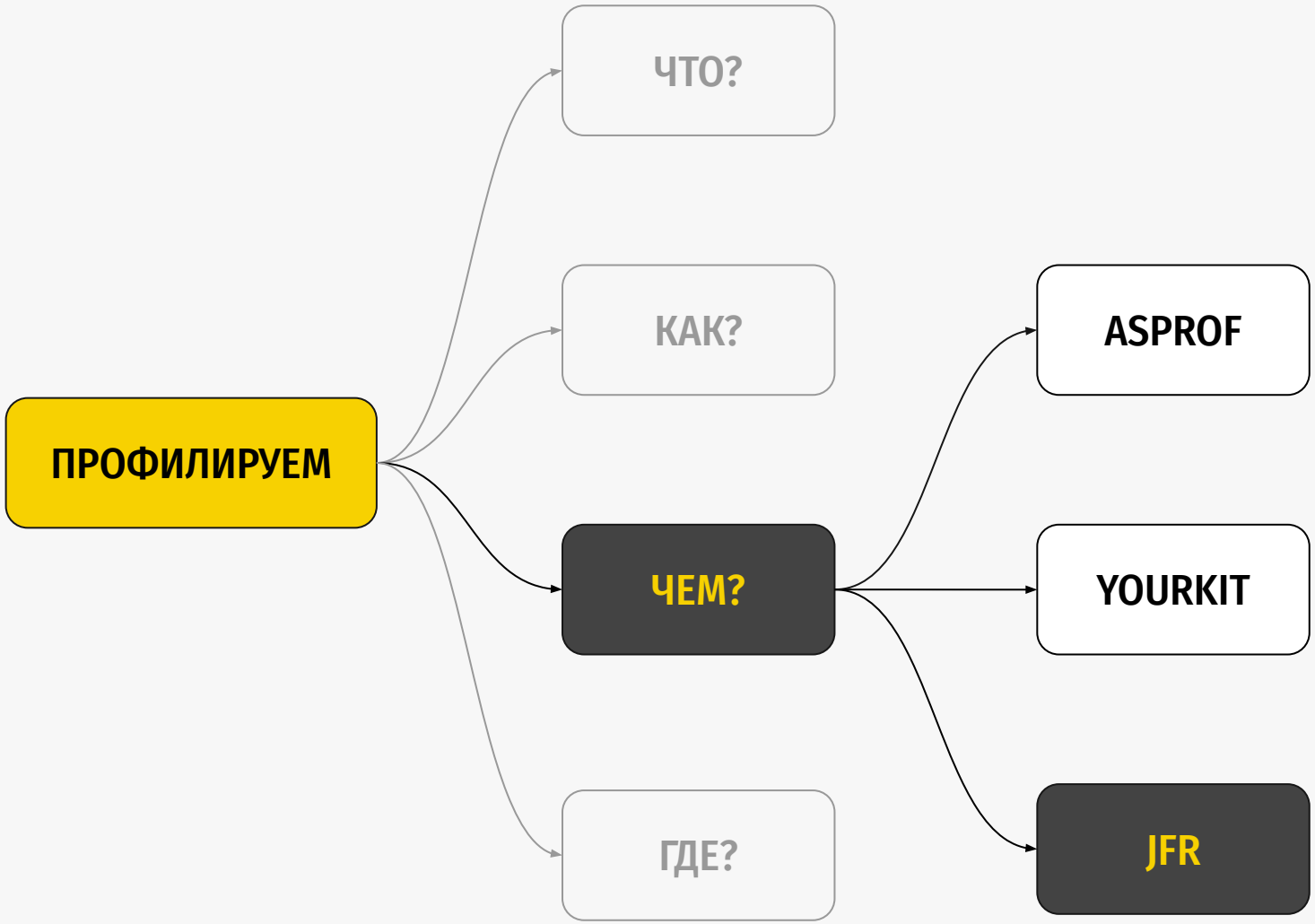
https://www.yourkit.com/docs/java-profiler/latest/help/java-ee-profiling-results.jsp

YOURKIT: ДРУГИЕ РЕЖИМЫ

- CPU time
- Wall clock time
- Аллокации памяти
- Исключения
- Блокировки
- События







JFR: ОБЩЕЕ

- **Распространение**: в составе HotSpot® JVM
- **Разработчик**: Sun/Oracle
- **Тип**: events + sampling
- **Интерфейсы**:
  - CLI
  - Java API
  - GUI: Java Mission Control

The JDK Flight Recorder was designed to minimize the Observer Effect in the profiled system, and is meant to be always on in production systems.

Wikipedia: JDK Flight Recorder

JFR: CPU SAMPLING PROFILING

- У JFR свой механизм получения стектрейсов
- Особенности:
  - Не поддерживает Wall-clock time
  - Выводит только Java-код, не нативный
  - Пропускает некоторые методы, например, `System.arraycopy`
  - Сильно зависит от `-XX:+DebugNonSafepoints`
- Подробнее: Alexey Ragozin – Lies, darn lies and sampling bias

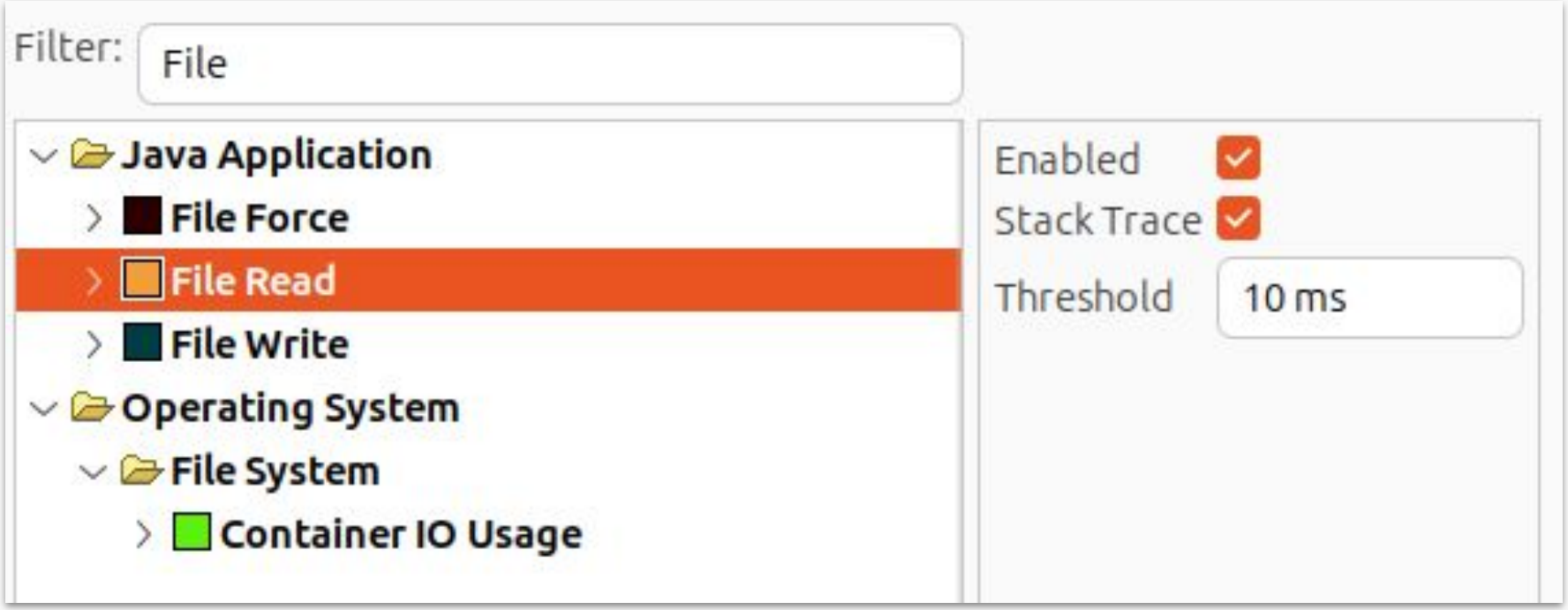
JFR: EVENTS

- На версию JDK 21 в HotSpot поддерживается 500+ событий
- Бывают с длительностью и без
- Сохраняются в бинарные файлы *.jfr
- Есть предустановленные наборы:
  - **default** – легковесный, малоинформативный
  - **profile** – увесистый, подробный (профилирующий)
- По умолчанию все отключены

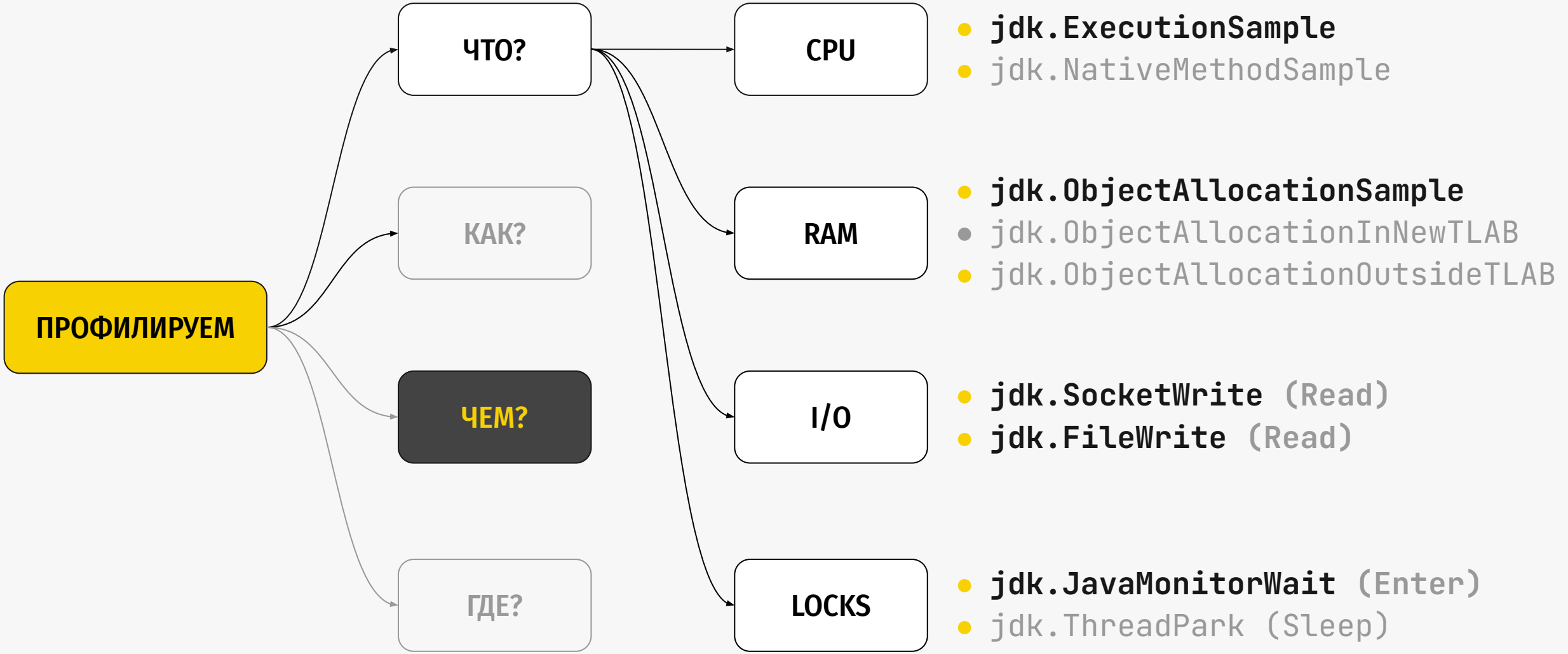
JFR: ПОДКЛЮЧЕНИЕ

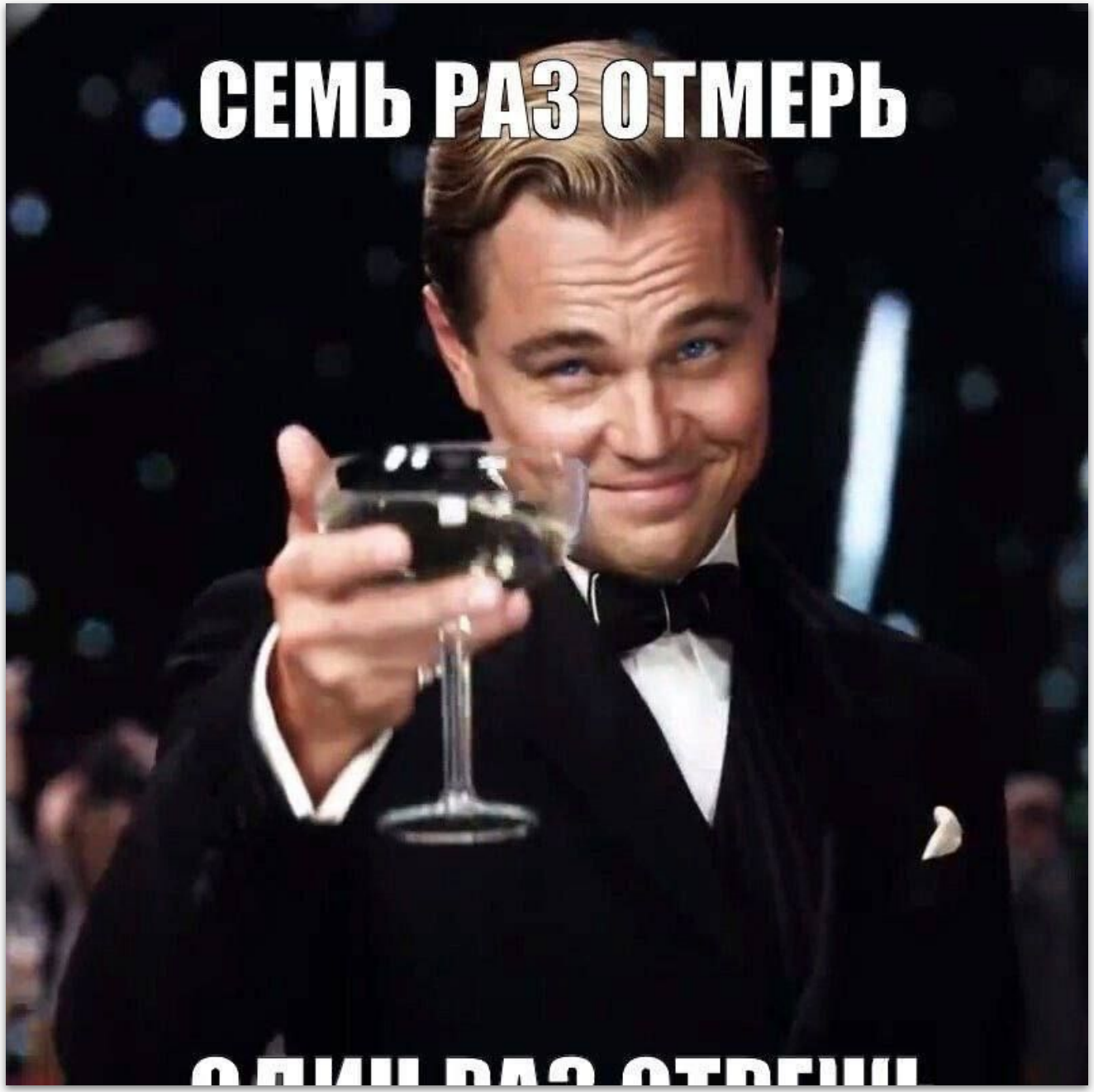
- **При старте:**
  - `java -XX:StartFlightRecording=filename=record.jfr …`
- **На лету:**
  - `jcmd <pid> JFR.start filename=record.jfr`
  - Либо через GUI (например, Java Mission Control)
  - Либо удаленно по JMX:
    - `com.sun.management.DiagnosticCommand.jfrStart`

JFR: ВЫБОР СОБЫТИЙ В JMC

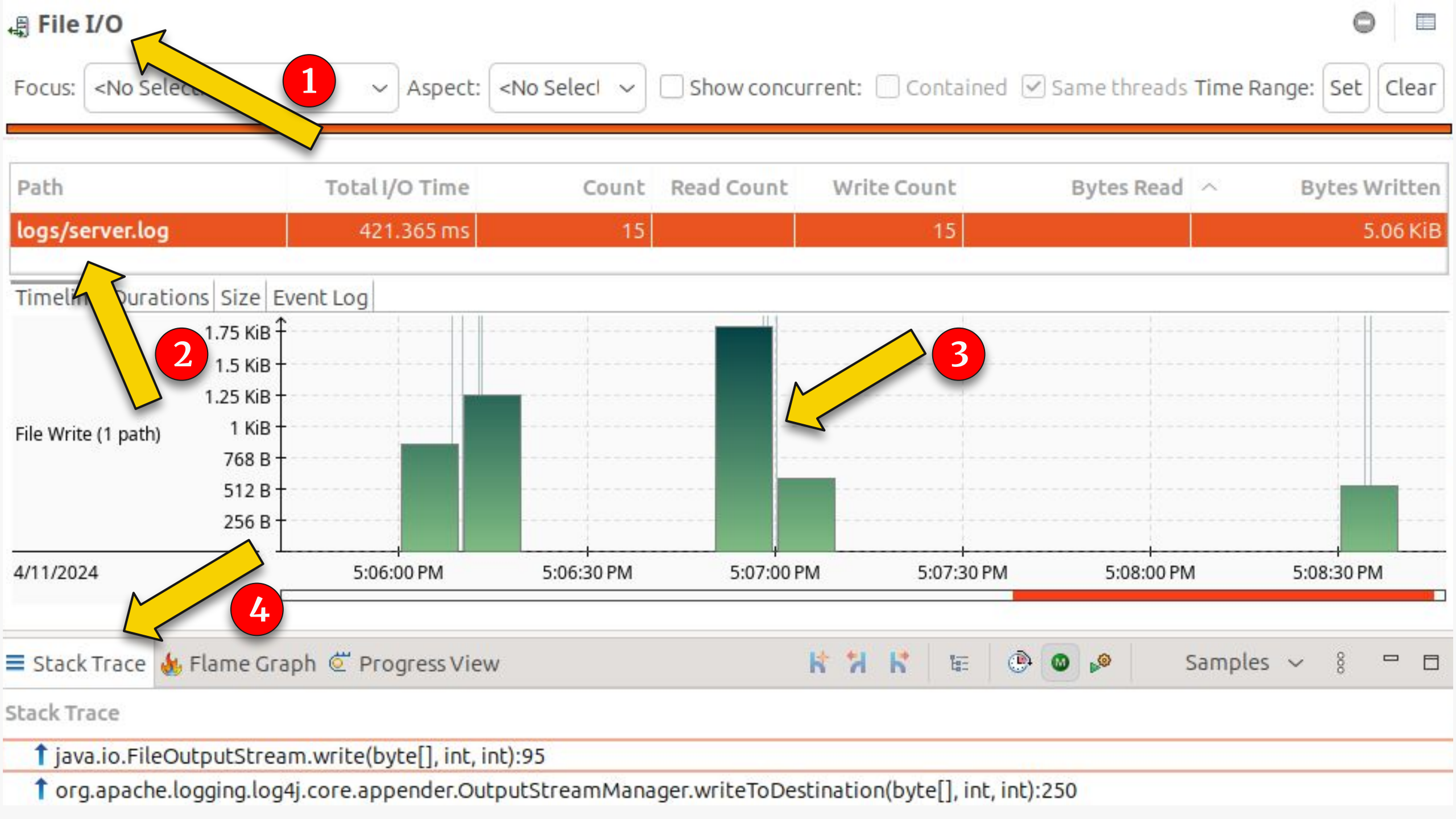


JFR СОБЫТИЯ ДЛЯ РЕСУРСОВ



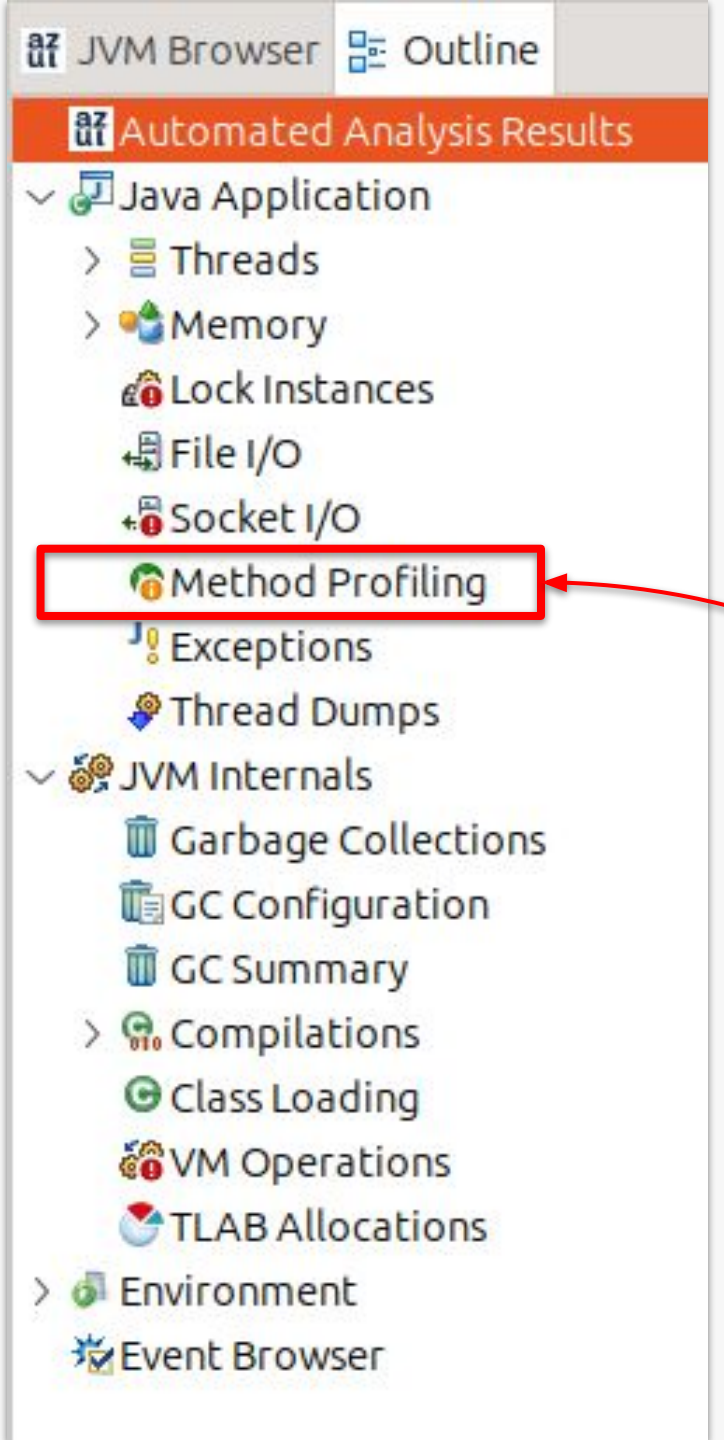


КАК УВИДЕТЬ СТЕКТРЕЙС ОБРАЩЕНИЯ К РЕСУРСУ В JMC?


File I/O
Focus:
<No Select
Aspect:
<No Select
Show concurrent:
Contained
Same threads
Time Range:
Set
Clear
Path
Total I/O Time
Count
Read Count
Write Count
Bytes Read
Bytes Written
logs/server.log
421.365 ms
15
15
5.06 KiB
Timeline
Durations
Size
Event Log
File Write (1 path)
1.75 KiB
1.5 KiB
1.25 KiB
1 KiB
768 B
512 B
256 B
4/11/2024
5:06:00 PM
5:06:30 PM
5:07:00 PM
5:07:30 PM
5:08:00 PM
5:08:30 PM
1
2
3
4
Stack Trace
Flame Graph
Progress View
Samples
Stack Trace
java.io.FileOutputStream.write(byte[], int, int):95
org.apache.logging.log4j.core.appender.OutputStreamManager.writeToDestination(byte[], int, int):250

ДЕРЕВО РЕЗУЛЬТАТОВ В JMC

- Содержит записанные события
- Если чего-то нет, см. **Event Browser**
- Почти везде wall-clock time



Результаты профилирования sampling'ом (CPU time)

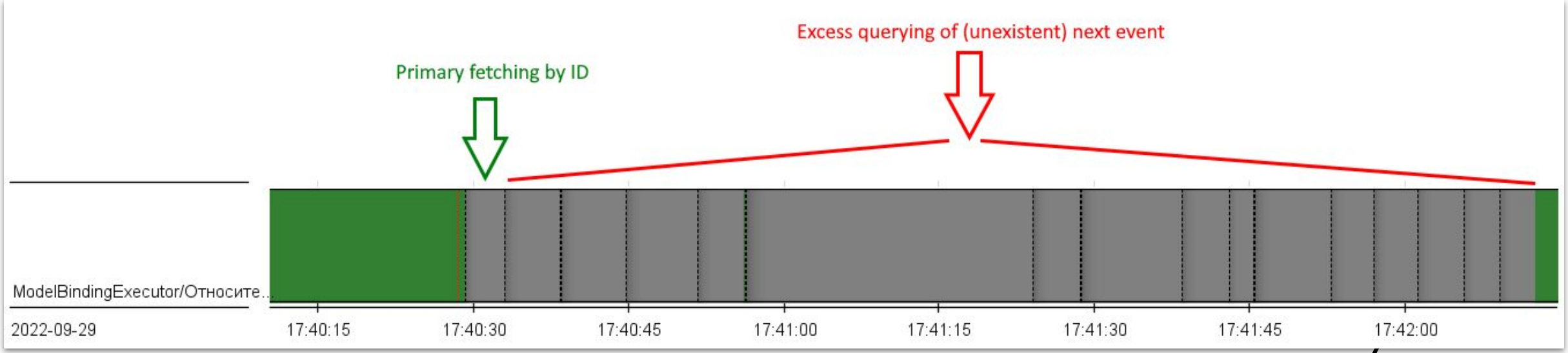
ℹ️ Подробности:
Alexey Ragozin – Hunting down code hotspots with JDK Flight Recorder

WALL-CLOCK: ПРИМЕР ИЗ ЖИЗНИ

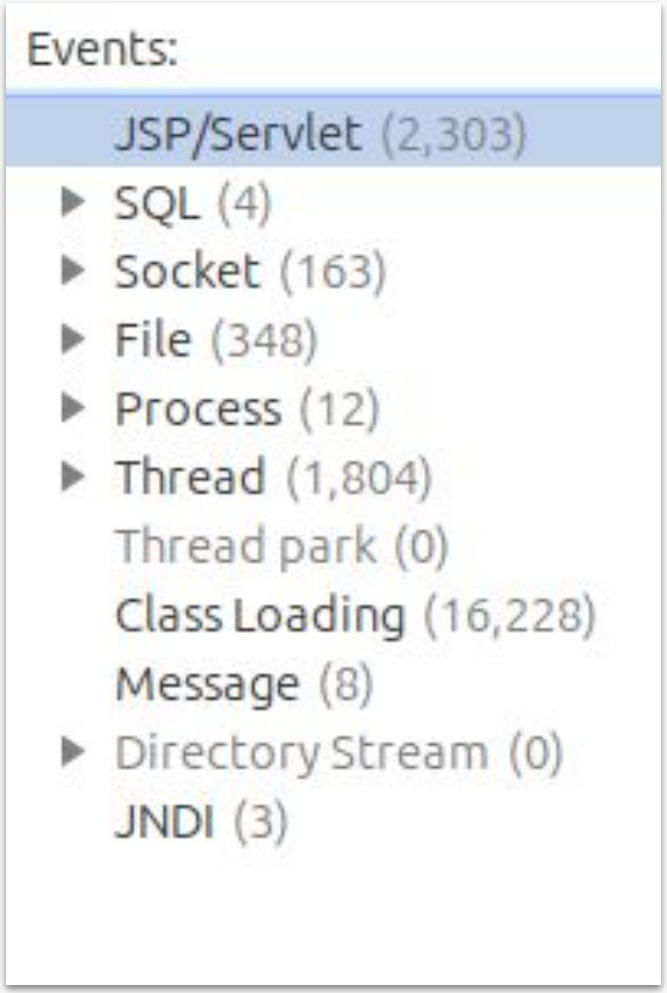
Projects / [redacted] / Add parent / [redacted]

Eliminate excess load on Cassandra upon fetching events by ID

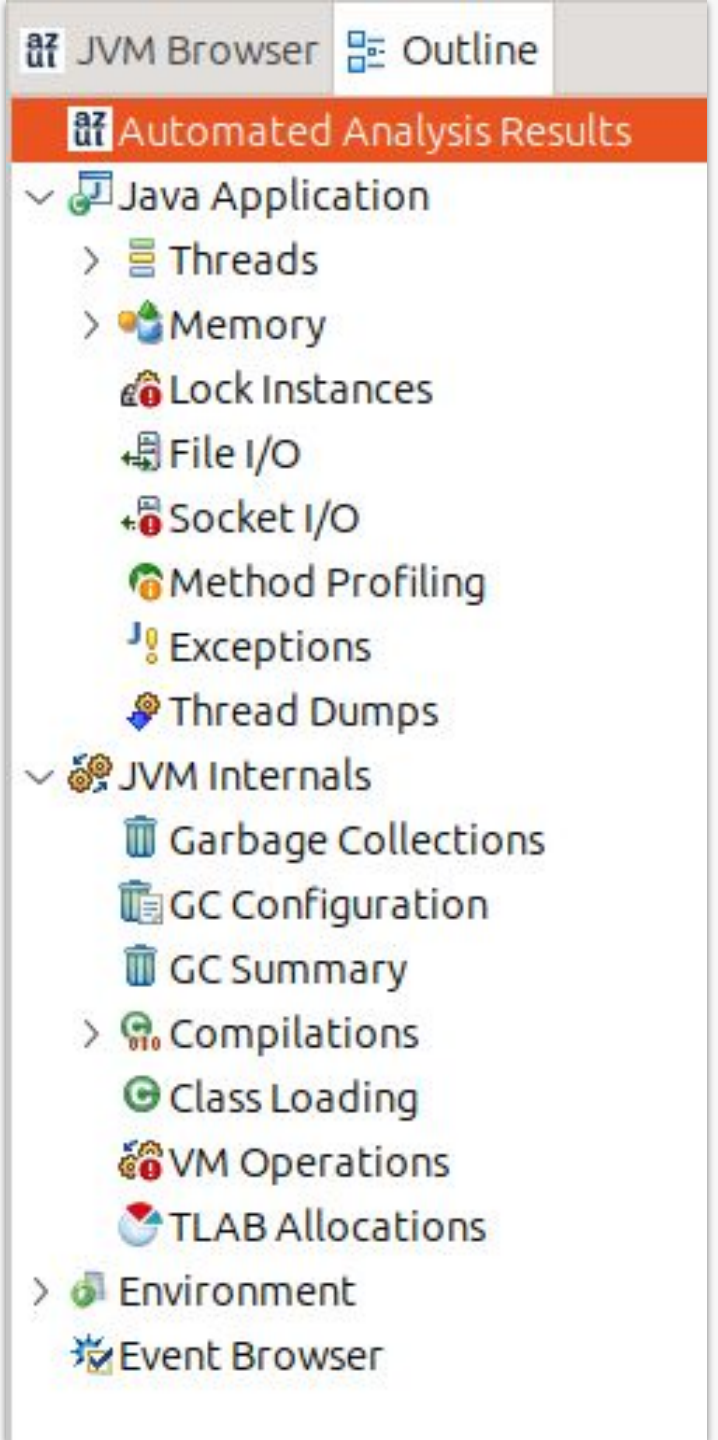
```
iterator.hasNext() ? iterator.next() : null
```

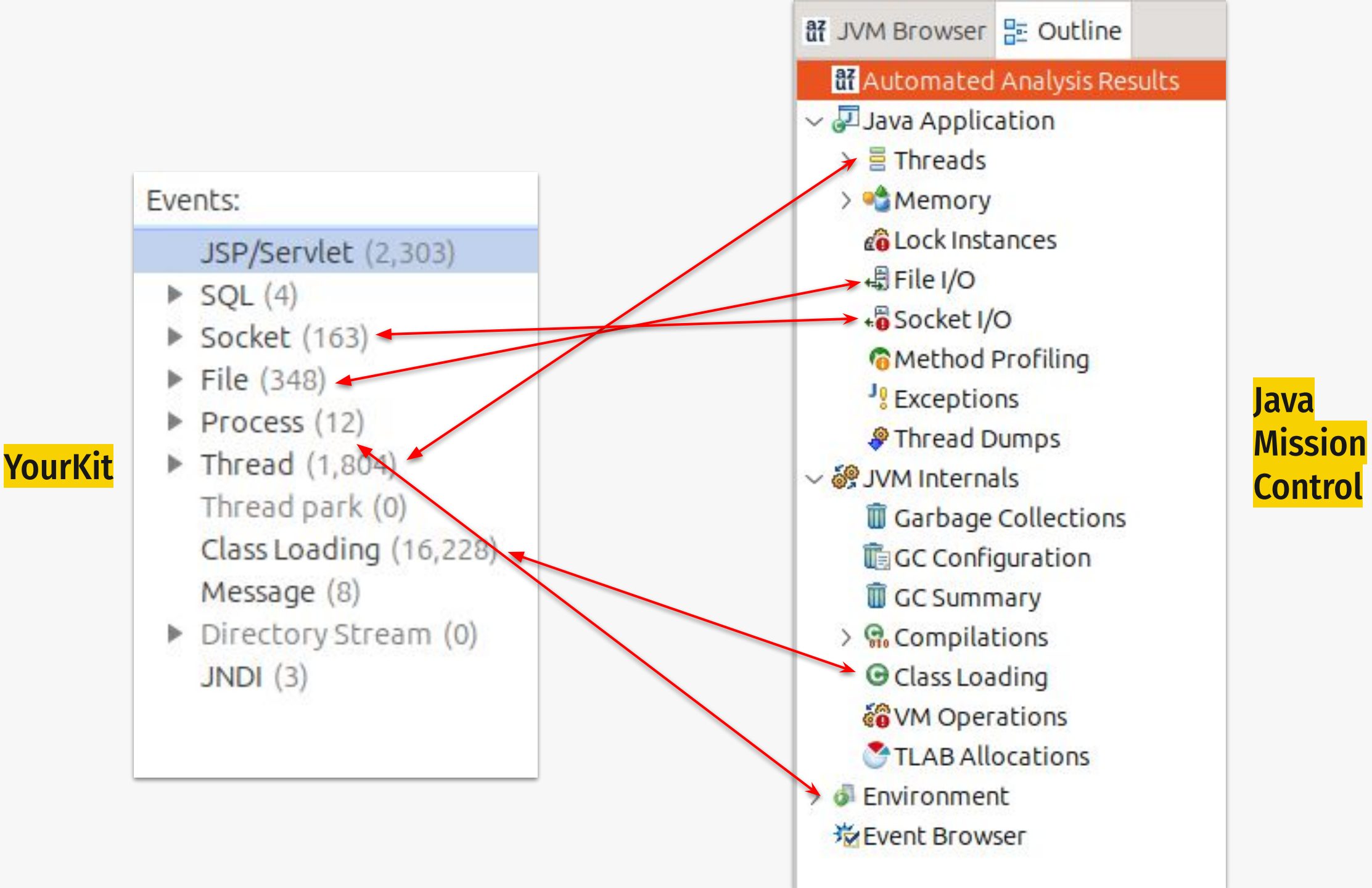


YourKit



Java Mission Control



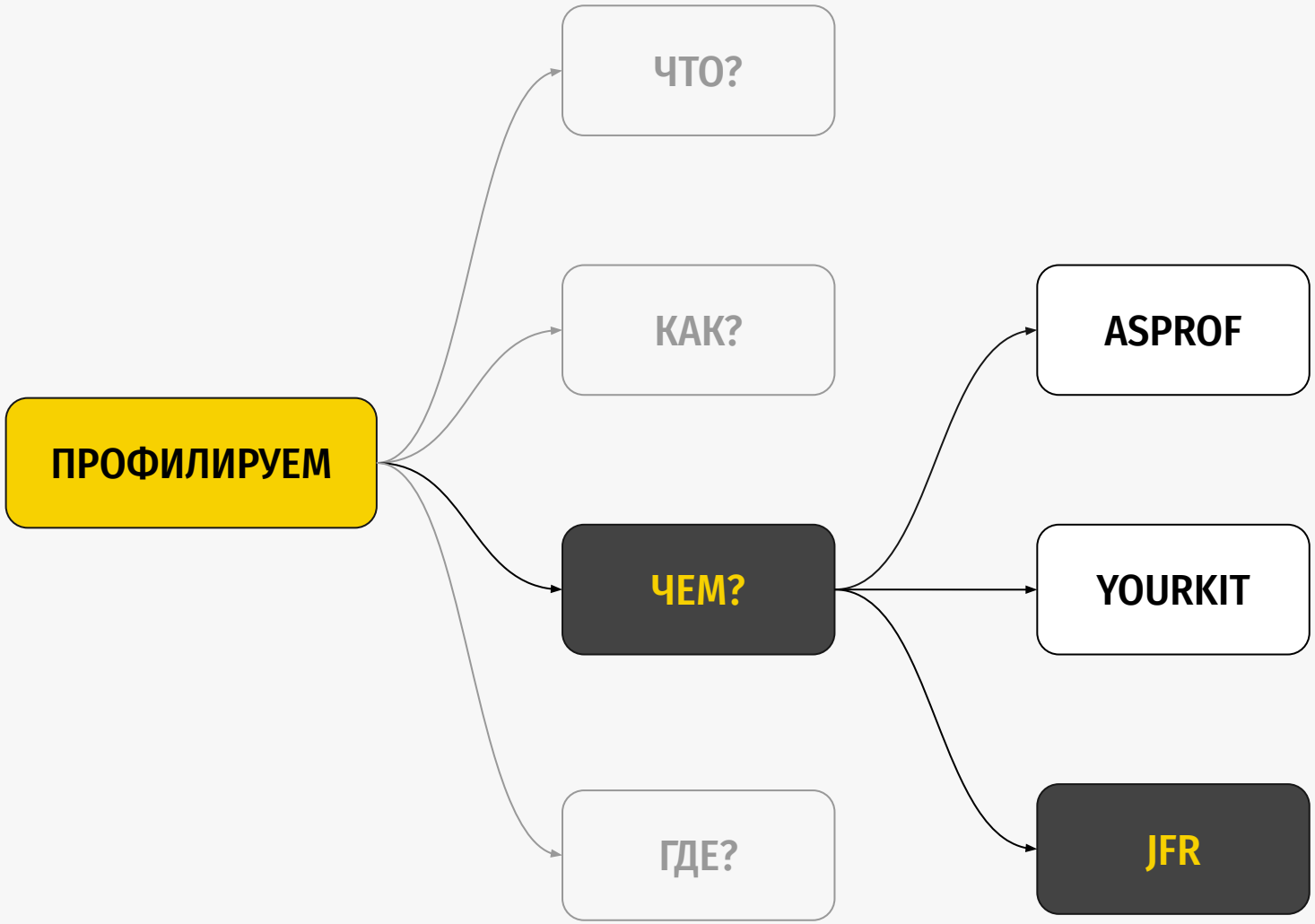


JFR: КАК ДОБАВИТЬ СВОЕ СОБЫТИЕ

1. Объявить в коде как Java класс
2. Выпускать в нужных местах
3. Искать в Event Browser’е в JMC

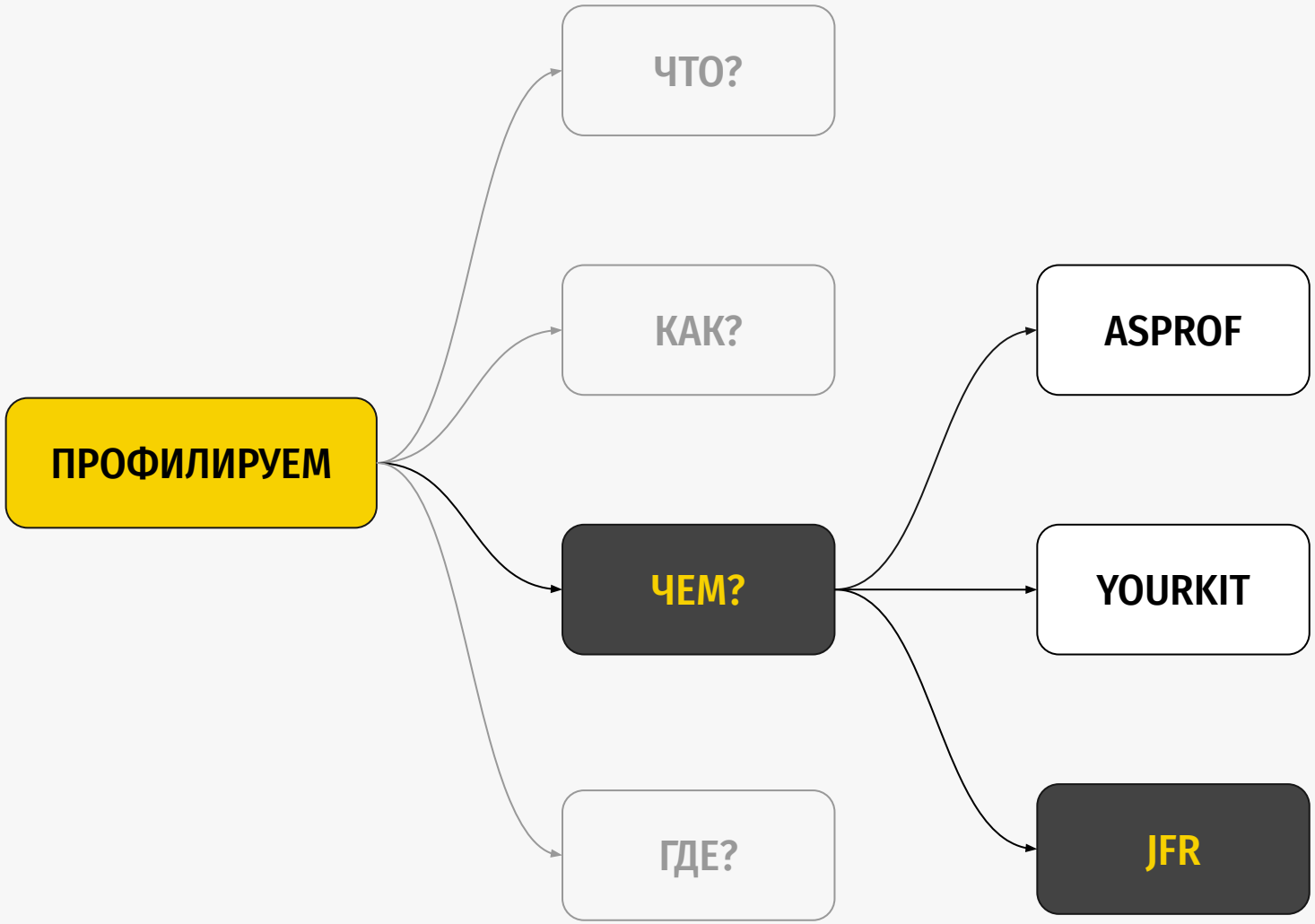
Примеры и объяснения в статье:

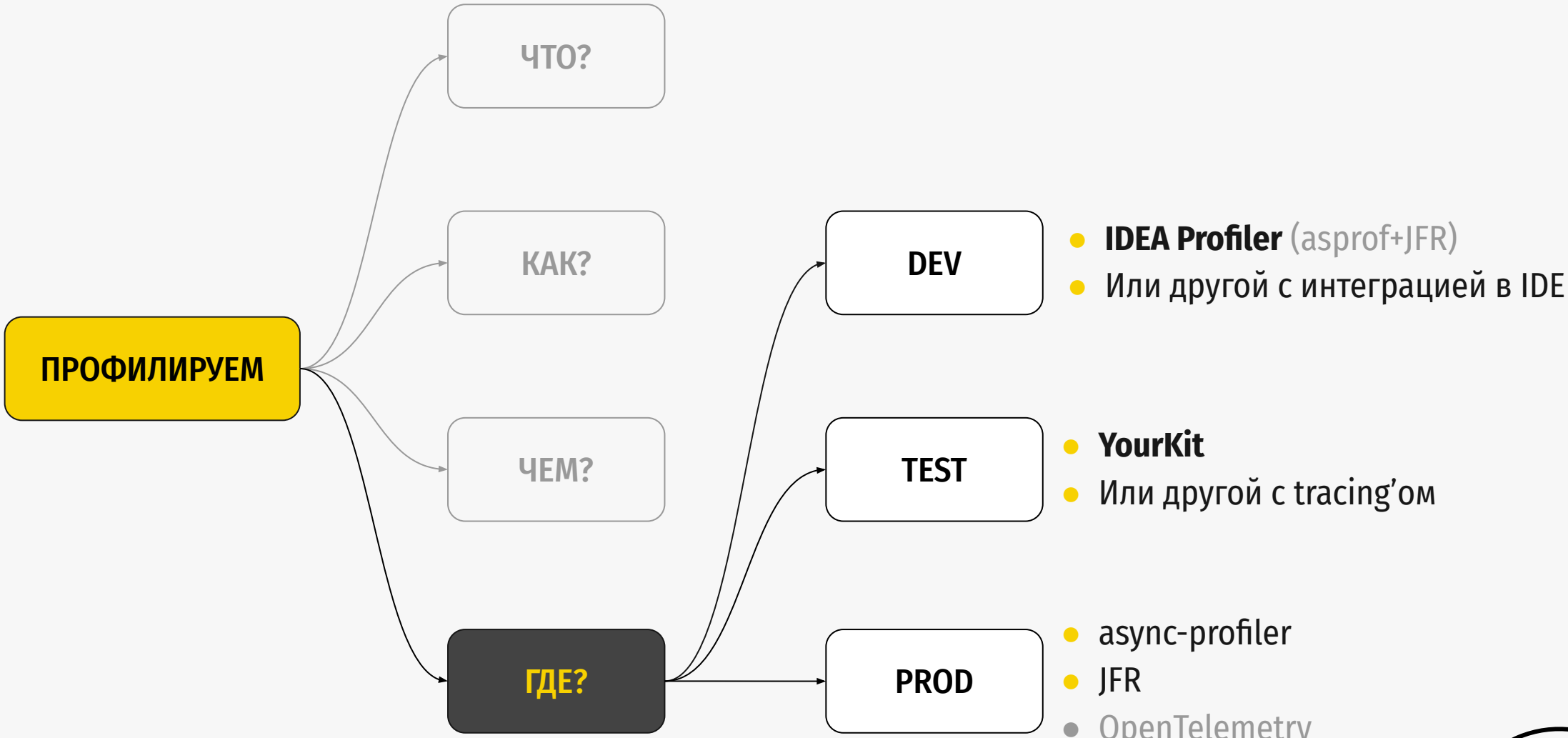
Monitoring REST APIs with Custom JDK Flight Recorder Events



СВОДКА

Фича \ Инструмент	**ASPROF**	**YOURKIT**	**JFR**
Sampling/**T**racing/**E**vents	S+T	S+T	E+S
Собственный GUI	✖	✅	✔
Open source	✅	✖	✅
Удаленное профилирование	✖	✅	✅
Стектрейсы с native-частью	✅	✖	✖
Поддержка high-level метрик	✖	✅	✖





CONTINUOUS PROFILING @ PRODUCTION

- **async-profiler**:
  - `asprof --loop 1h -f profile-%t.jfr <pid>`
- **JFR**:
  - `jcmd <pid> JFR.start filename=long.jfr \`
    `maxage=24h maxsize=2G settings=profile.jfc`
- **Удаленно (в кластере)**:
  - YourKit Connection Broker / JProfiler Perfino
  - OTEL-based tools

ПОМНИТЬ ПРИ ЗАПУСКЕ НА PRODUCTION

- **async-profiler**:
  - При работе в контейнере может требовать дополнительных приседаний
  - Если пользователь не root, могут понадобиться флажки для ядра Linux
- **JFR**:
  - Складывает чанки записей в репозиторий (`/tmp`)
  - Стектрейсы по умолчанию – 64 этажа (`stackdepth`)


arm food
ПОПРОБУЙ МАМИНЫХ ЛЮЛЕЙ!*
*Люля кебаб от 99 рублей

ВЫВОДЫ

СОВЕРШЕННОГО ПРОФАЙЛЕРА ПОКА НЕТ

И не нужен...

Зелёная зона: подитог

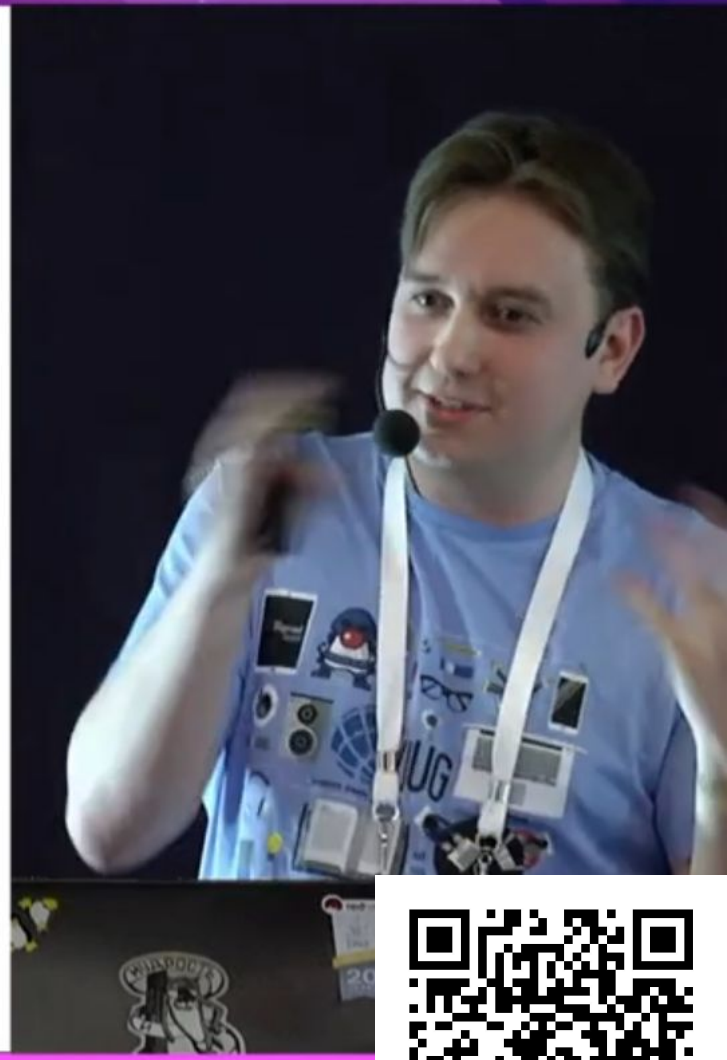
Профилирование –
необходимая часть ежедневной разработки

Наблюдения:

- >95% проблем находится на первых же заходах
- >90% проблем тривиально разрешимы
- Чёткие инструкции по запуску профилировки **сильно** помогают: отлично, если есть однострочник, или однокнопочник, или APM
- Возьмёте девелопера за руку, и с ним один раз попрофилируете – это **уверенно** купирует боязнь базовой перформансной работы[1]

[1] «Нет-нет, не надо закрывать это окно, оно боится тебя больше, чем ты его»

redhat



https://www.youtube.com/watch?v=CgRJUqO-dMQ

НА ЗАМЕТКУ

- Освойте какой-нибудь профайлер сегодня (пока не пригорело)
  - Не знаете, какой выбрать – берите async-profiler
- Начинайте с sampling'а и в режиме wall-clock time
  - Сомневаетесь? Повышайте частоту/длительность
- Профилируйте заранее: dev, test, stage
  - Не всё то bottleneck, что только на production

НЕ УПОМЯНУТЫЕ ССЫЛКИ

Статьи

- Why (Most) Sampling Java Profilers Are F*king Terrible
- Предрелизная оценка производительности на YourKit
- Couldn’t We Just Use AsyncGetCallTrace In A Separate Thread?

Доклады

- Safepoint – и пусть весь мир подождёт
- Профайлер в каждый дом

СПАСИБО!

Вопросы?

Владимир Плизгá
Tibbo Systems

@stopshelf
toparvion.pro

Слайды