

ПУТЕВОДИТЕЛЬ ПО ПРОФИЛИРОВАНИЮ ПРИЛОЖЕНИЙ НА JVM

Владимир Плизга

Tibbo Systems

JPoint 2025

ПРОФИЛЬ



Профилирование 😎



ПРОФИЛЬ



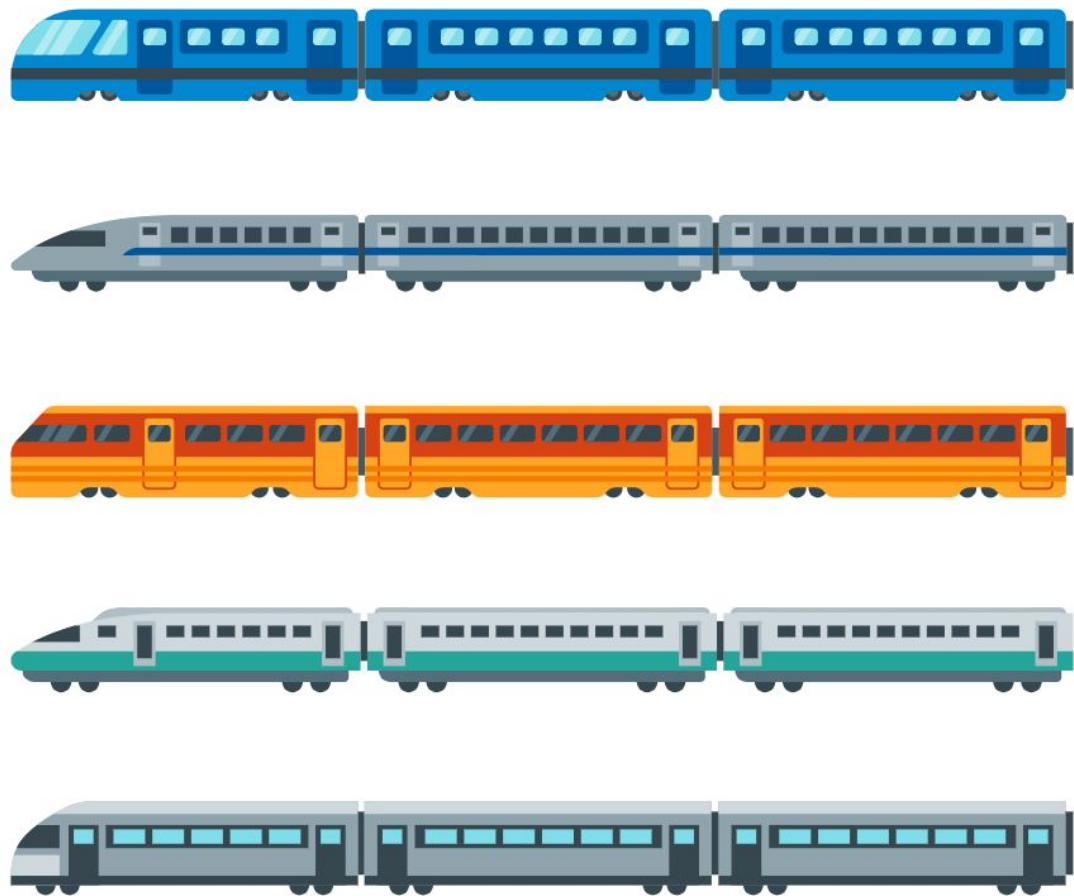
Профилирование 😎

ФАС



Фасирование 😞

ПРОФИЛЬ



designed by freepik.com

Профилирование 😎

ФАС



Изображение от pngtree.com

Фасирование 😞

ПУТЕВОДИТЕЛЬ ПО ПРОФИЛИРОВАНИЮ ПРИЛОЖЕНИЙ НА JVM

Владимир Плизга

Tibbo Systems

JPoint 2025

ПРИВЕТ!

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



КОНТЕКСТ

“Когда происходят
ошибки, у нас есть
стектрейсы”



“А когда падает
производительность
– их нет :(”



КОНТЕКСТ

“Логи и метрики не помогают!”



“Хоть на работу не ходи!”

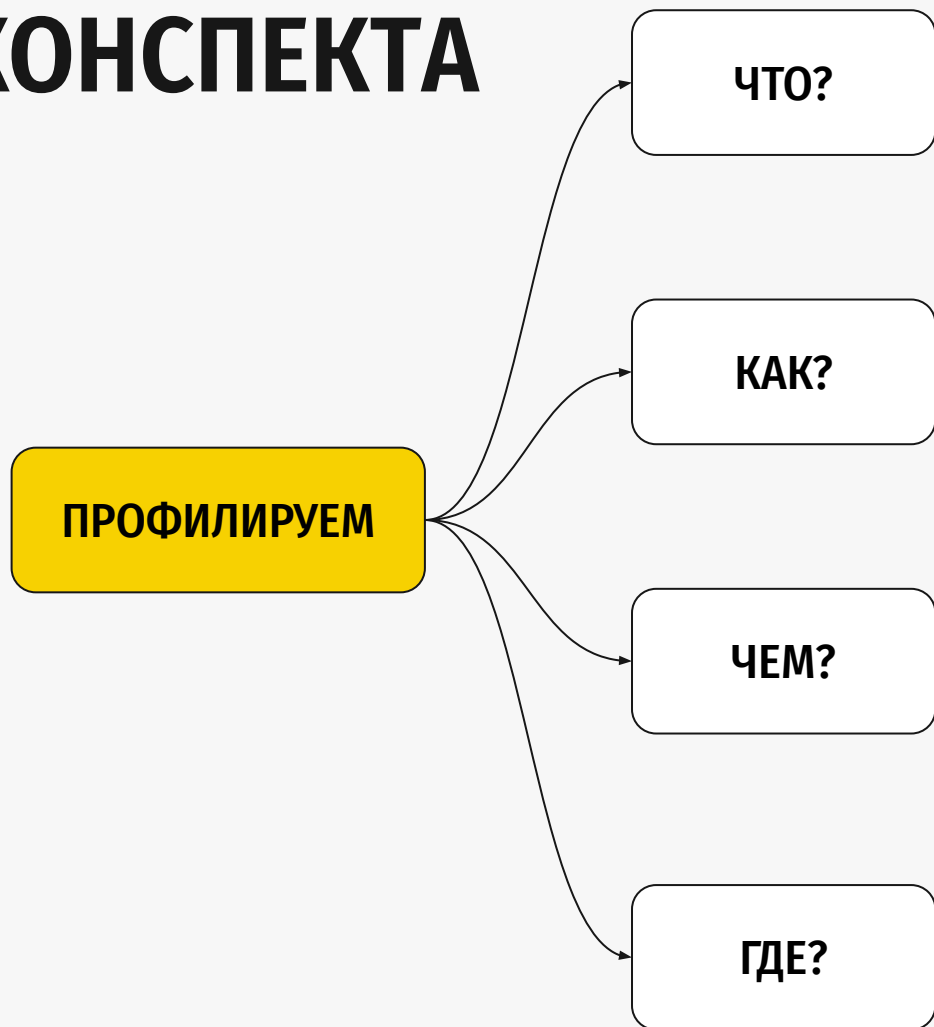


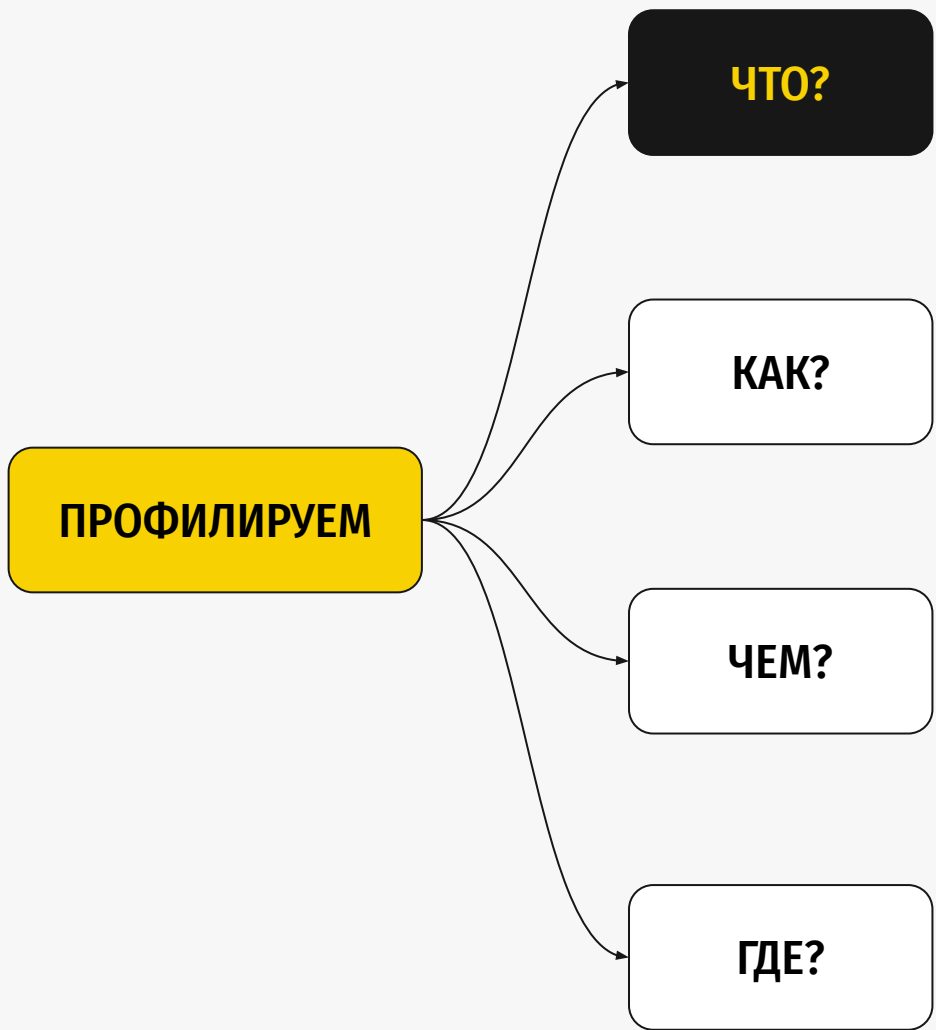
ЧТО ДЕЛАТЬ?

- 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

Прикладной
код

JVM

OS



...

Цепочка
следствий

ЕСЛИ ХОЧЕТСЯ ЗАНЫРНУТЬ

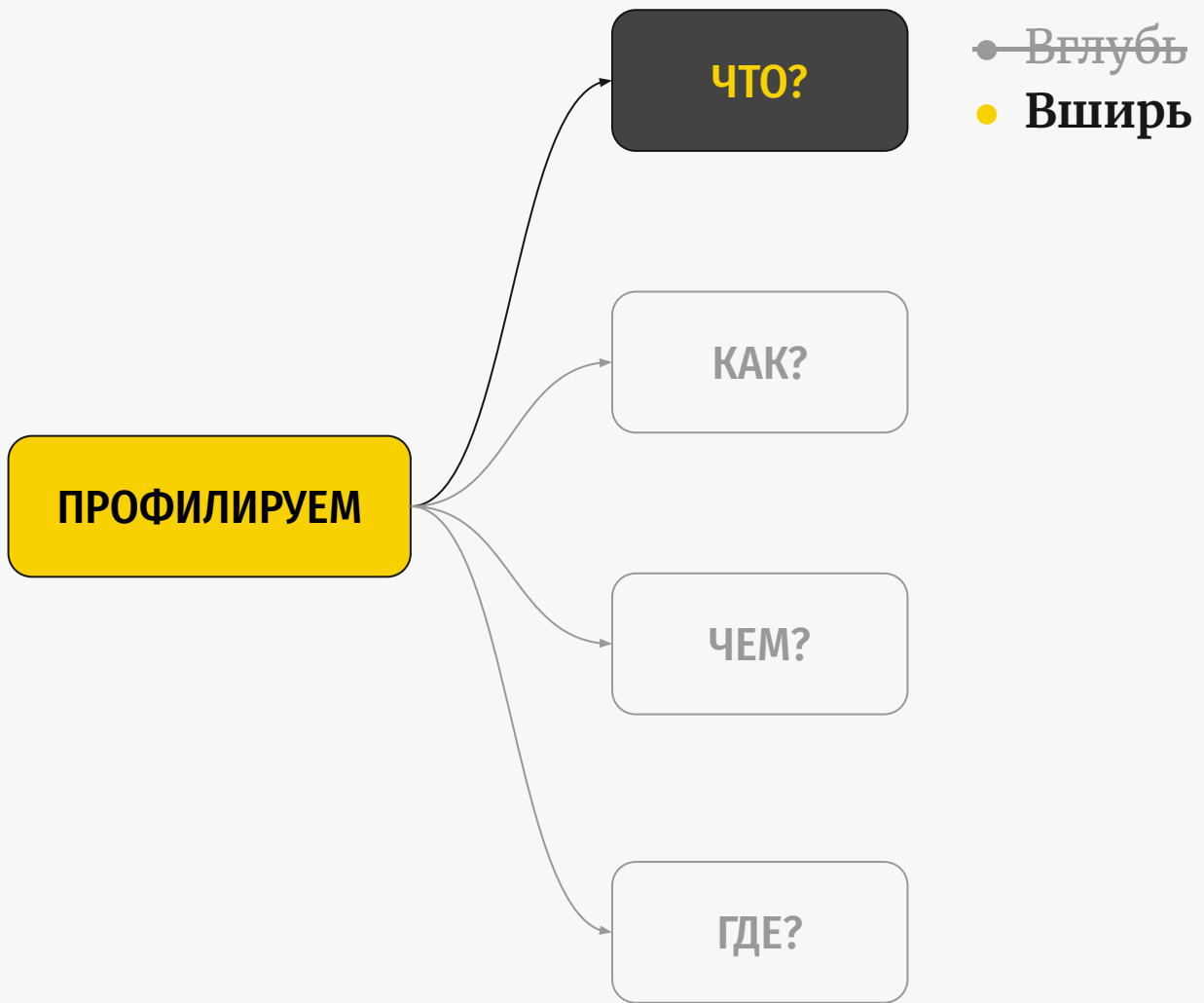
Joker<?>
2024

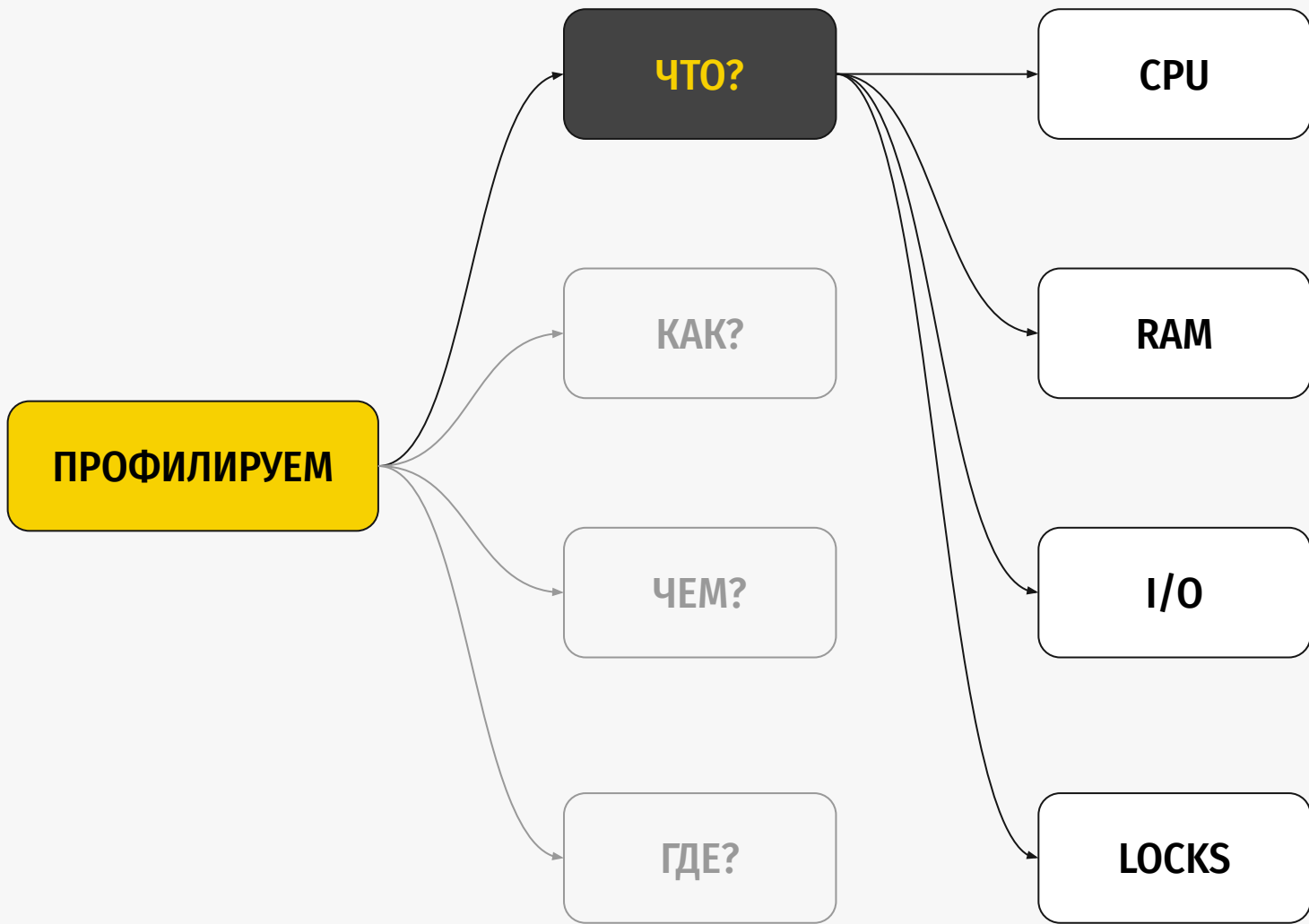
Профилирование
Java в стиле Linux

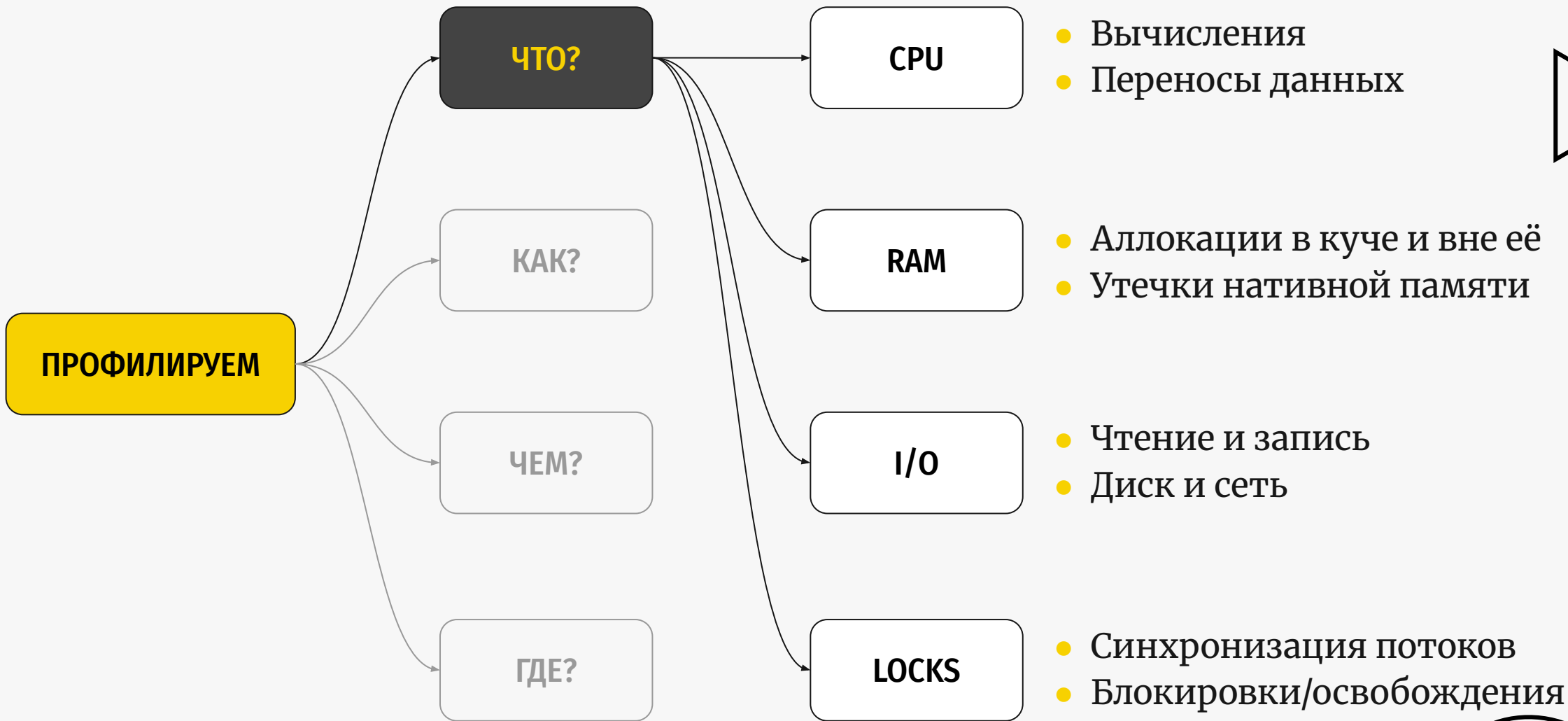


Сергей
Мельников
Dijkstra Markets

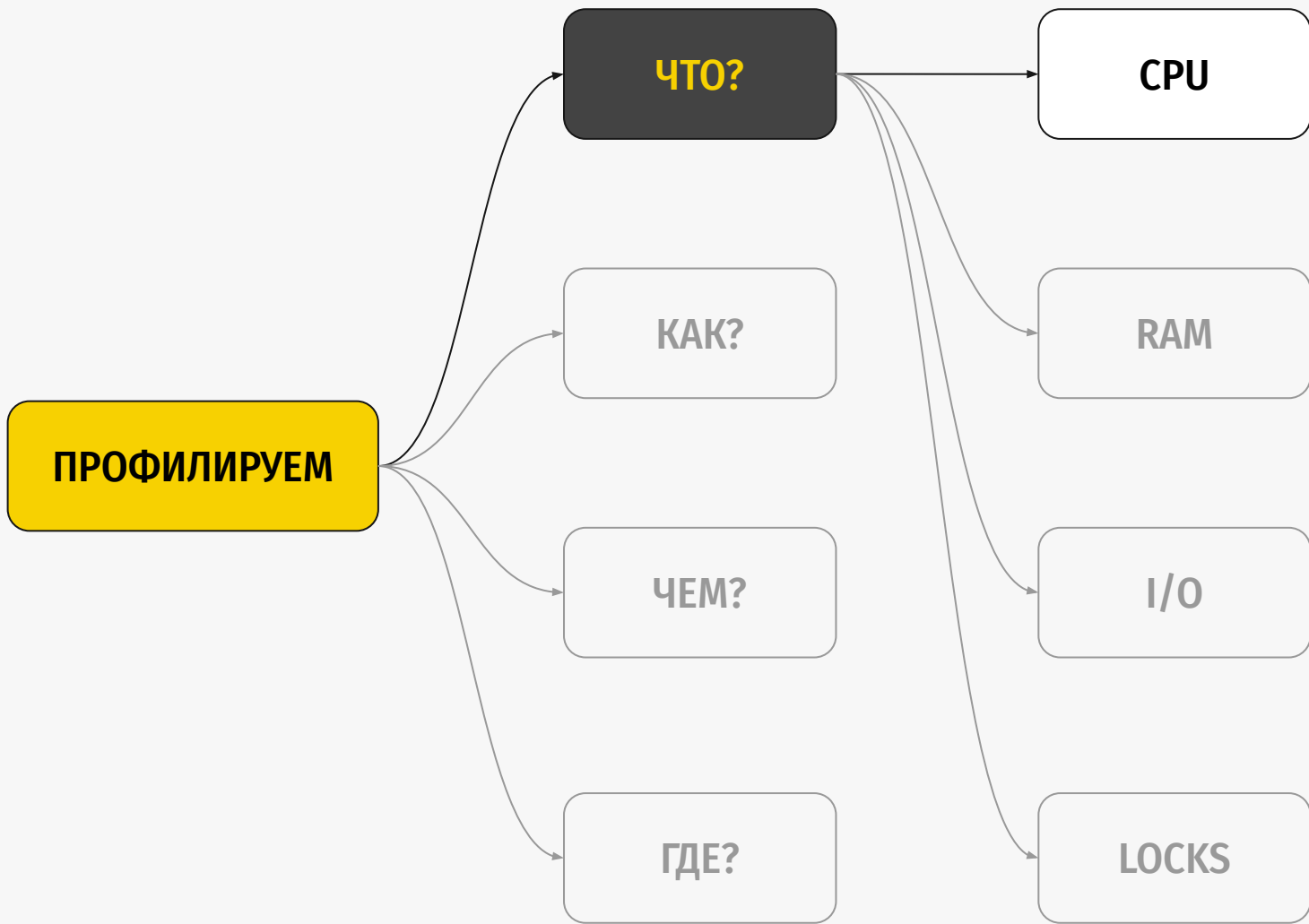
<https://www.youtube.com/watch?v=EZA5zcfFA4w>



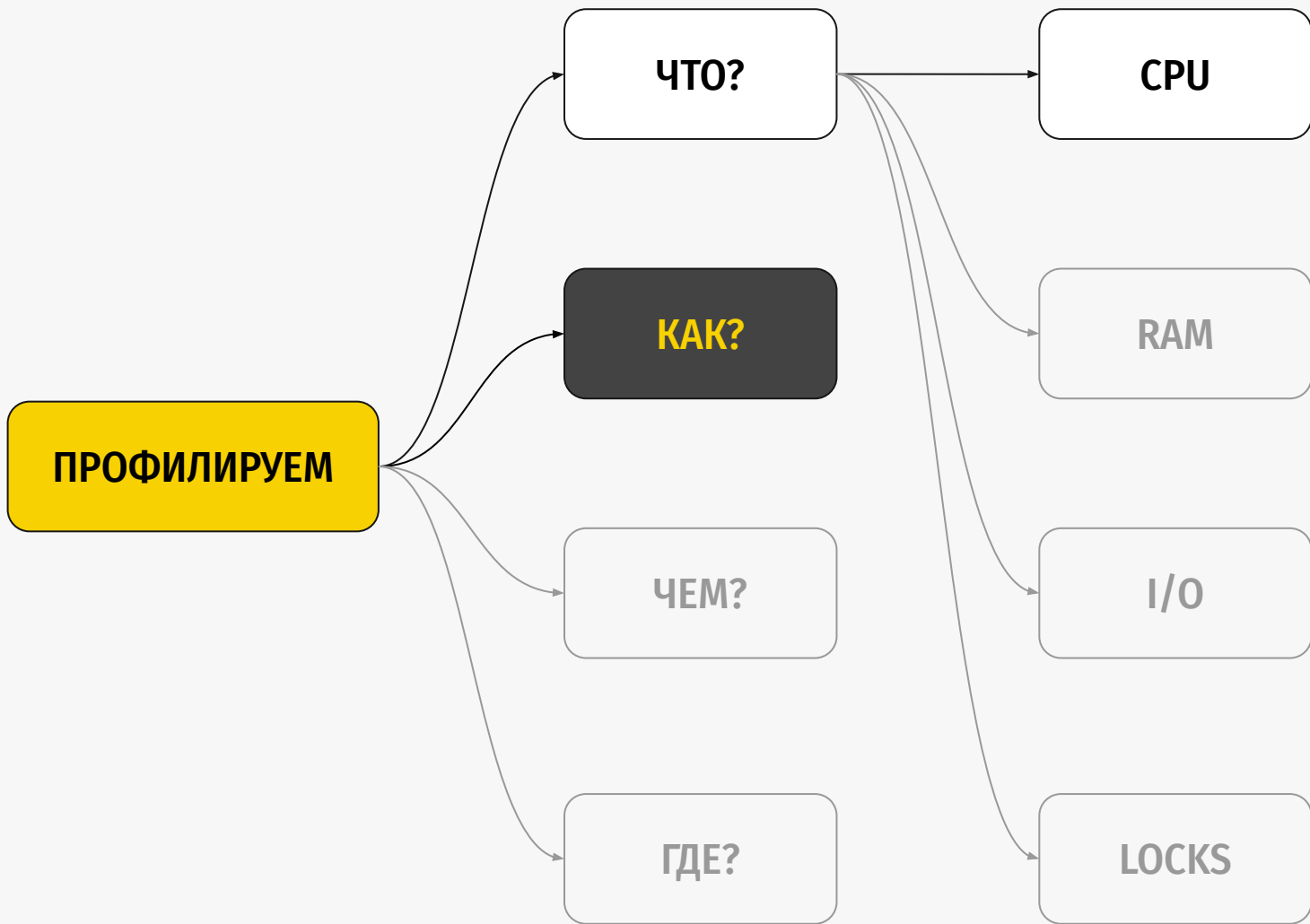




a.k.a. threads



- 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

2009

НЕ ОТХОДИ
ОТ СТАНКА
ДО ПОЛНОЙ ОСТАНОВКИ
СЕРДЦА

SAMPLING

Плюсы

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

Минусы

- ✗ Не всегда точен
- ✗ Подвержен saferpoint bias*

* SAFERPOINT BIAS (НА ПАЛЬЦАХ)

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

ГДЕ УЗНАТЬ БОЛЬШЕ

JUGARU

Safepoint

запросили остановку

Поток 1

работаем с памятью

return

Поток 2

что-то вычисляем

loop

оdnokлассники alm works SEMRUSH competitors research

Андрей Паньгин — Искусство Java профилирования

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



SAMPLING

Плюсы

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

Минусы

- ✗ Не всегда точен
- ✗ Подвержен safepoint bias*

SAMPLING

Плюсы

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

Минусы

- ✗ Не всегда точен
- ✗ Подвержен safepoint bias*
- ✗ 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

- ✗ Лютый overhead
- ✗ Сложность настройки
- ✗ 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



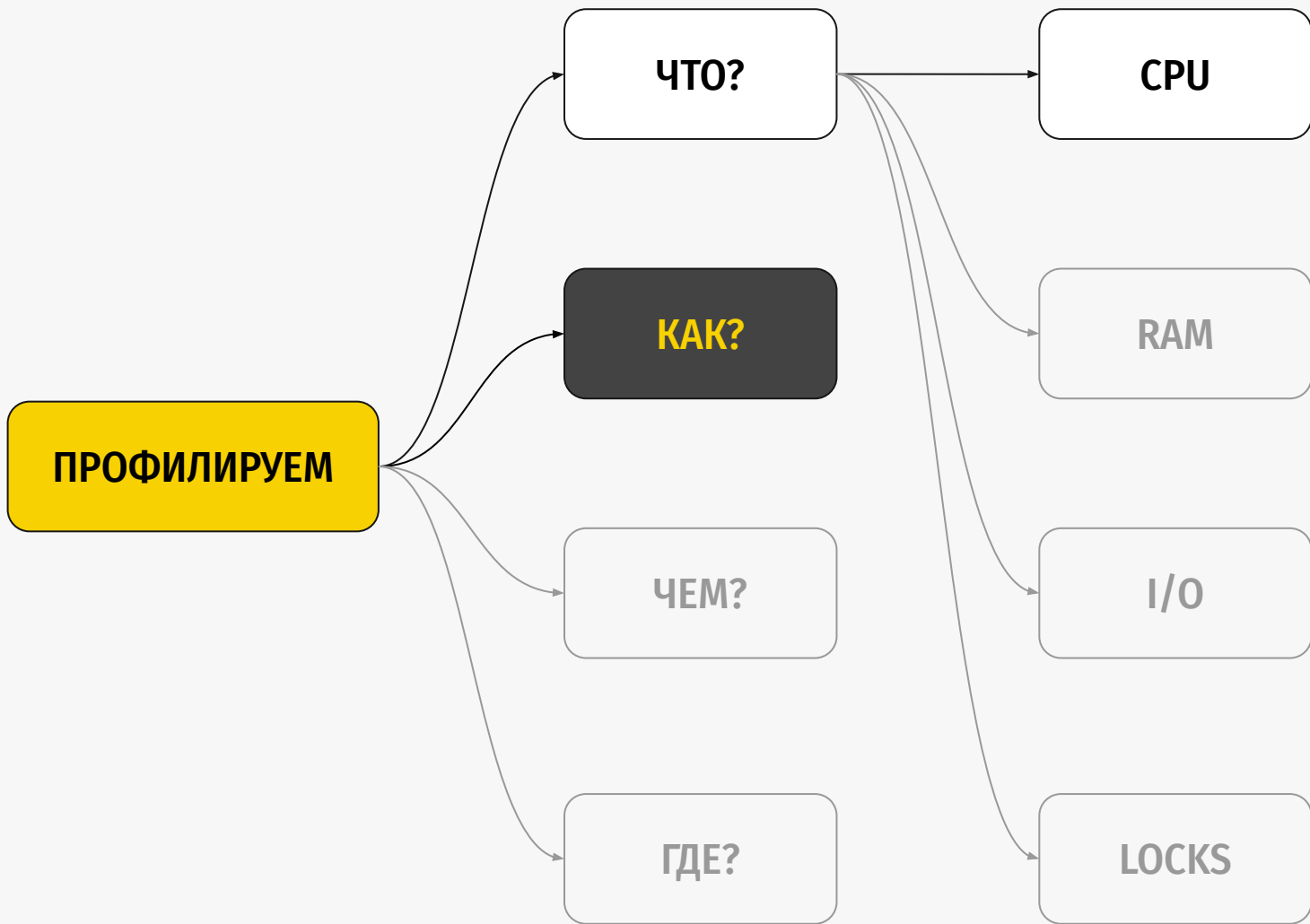
Иван Крылов

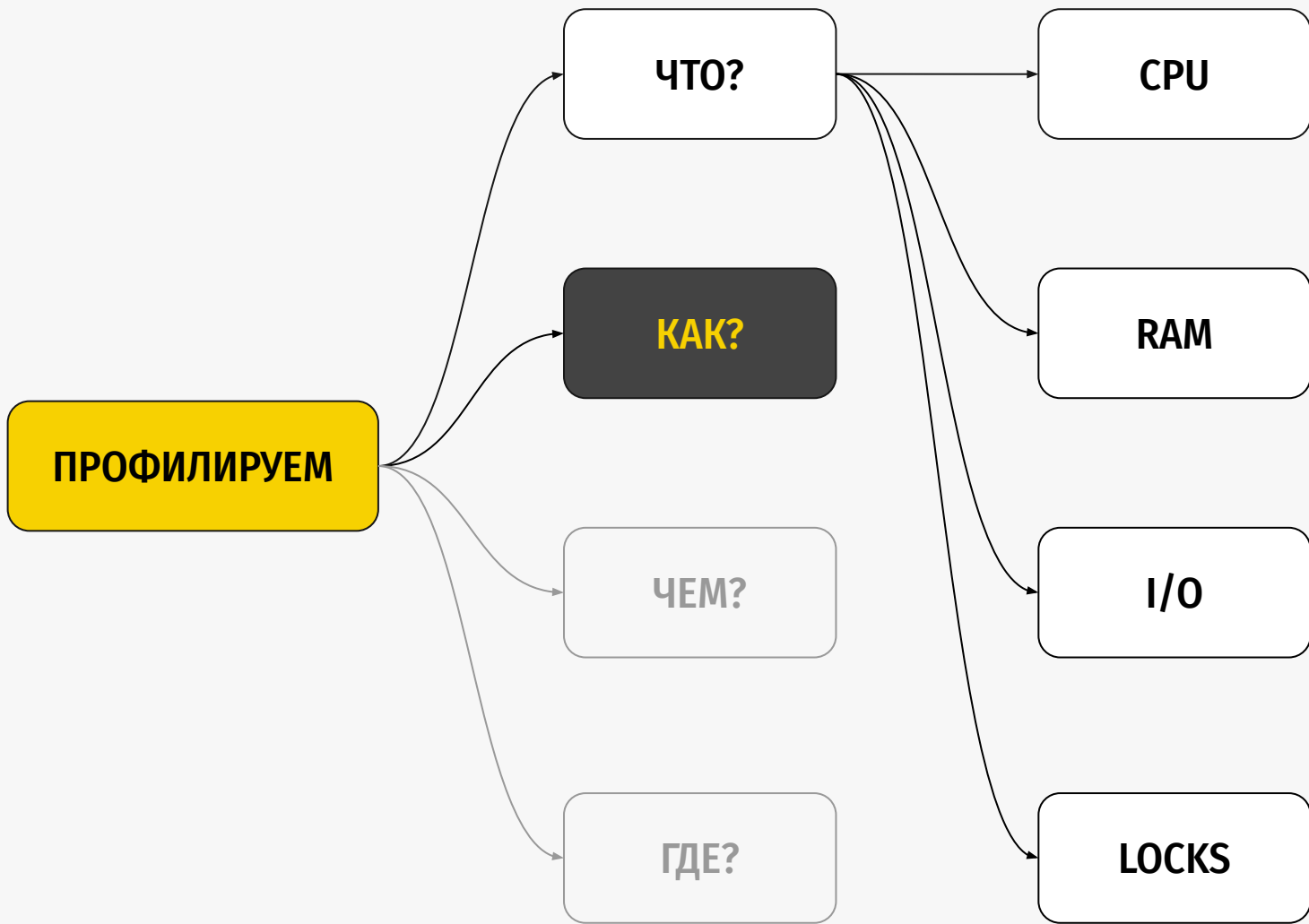
Azul Systems

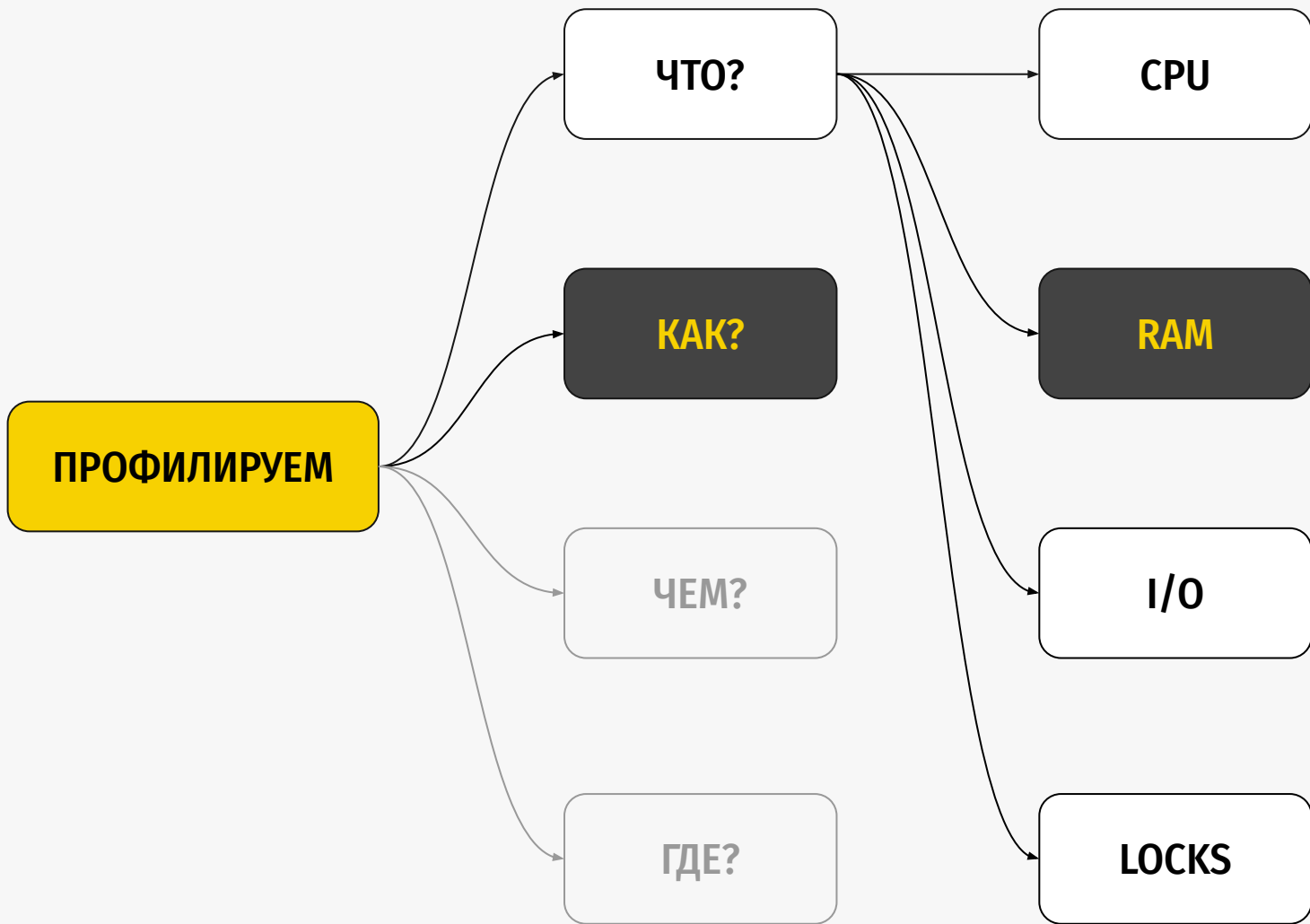
Жизненный цикл
JIT кода



<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, создается 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

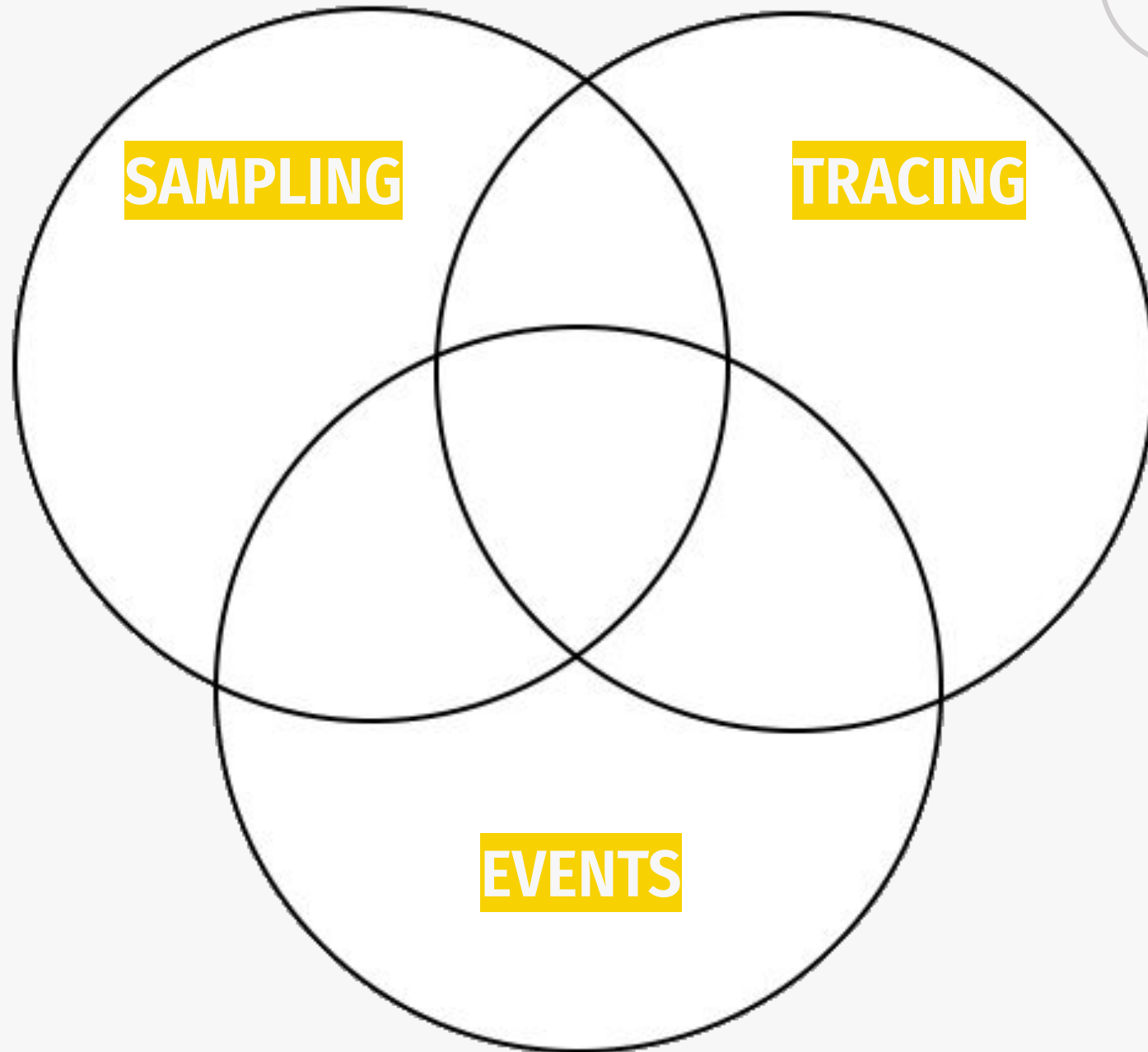
Минусы

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

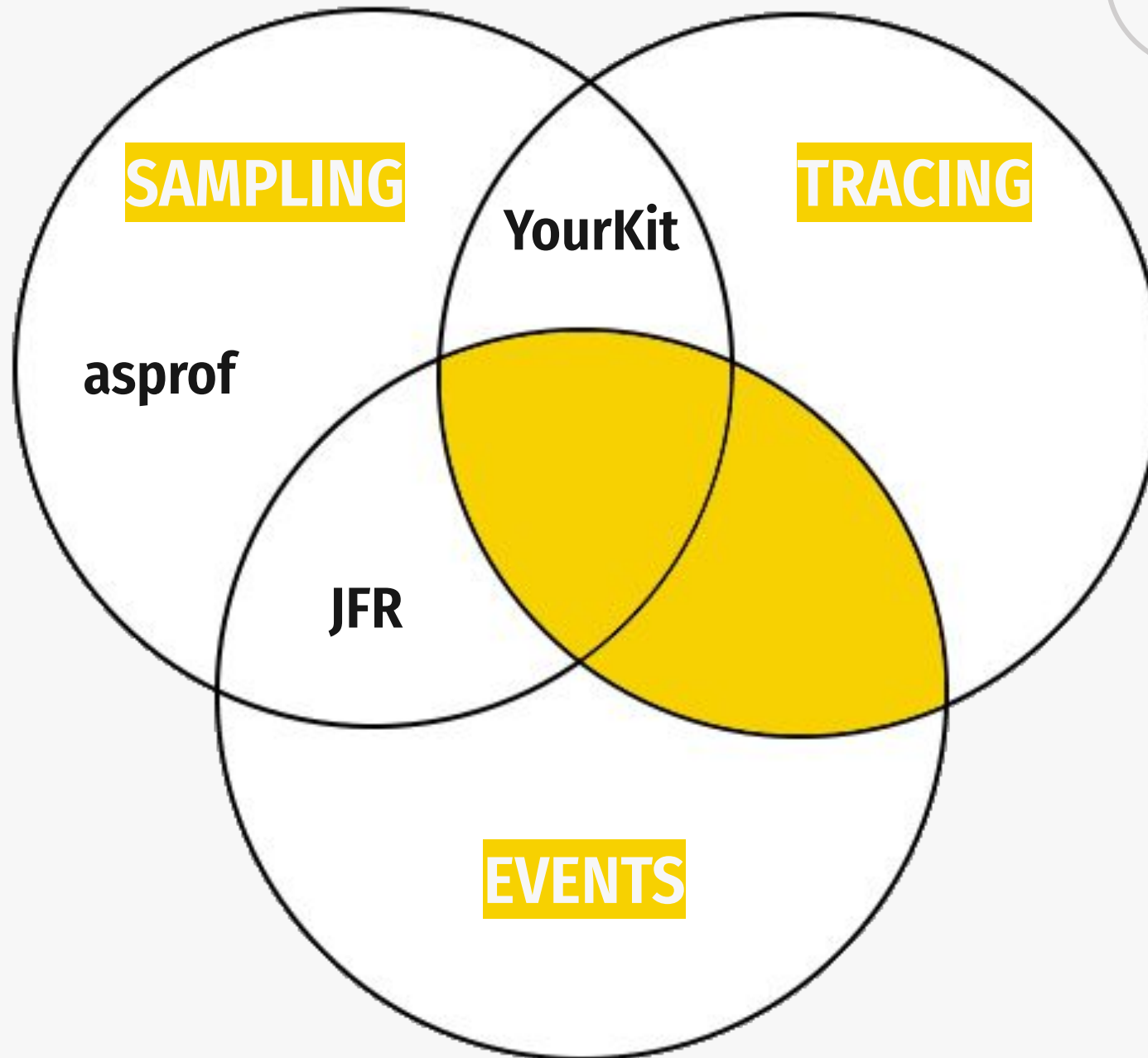
ПОПУЛЯРНЫЕ ПРЕДСТАВИТЕЛИ

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

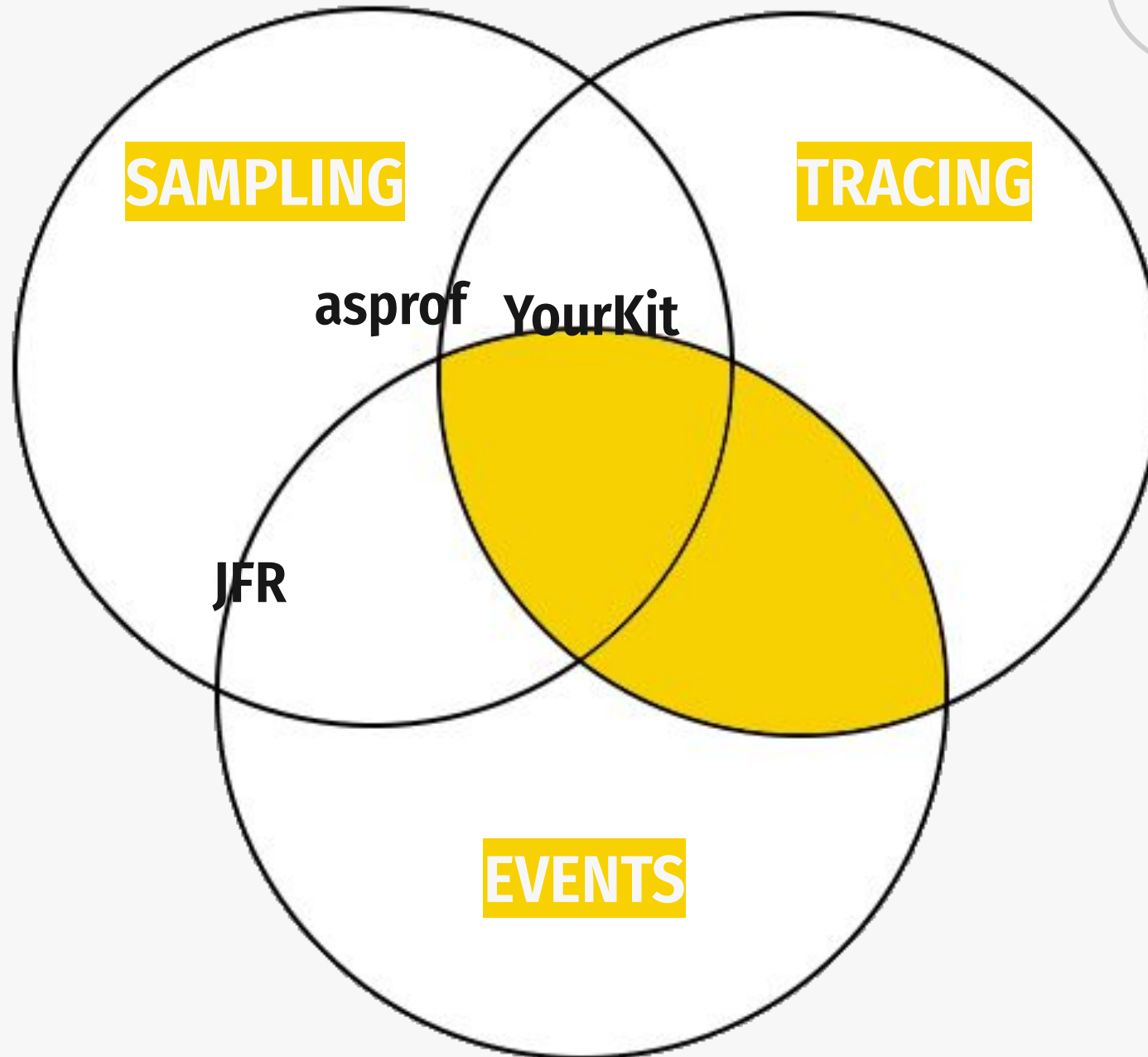
А ЕСЛИ ТОЧНЕЕ

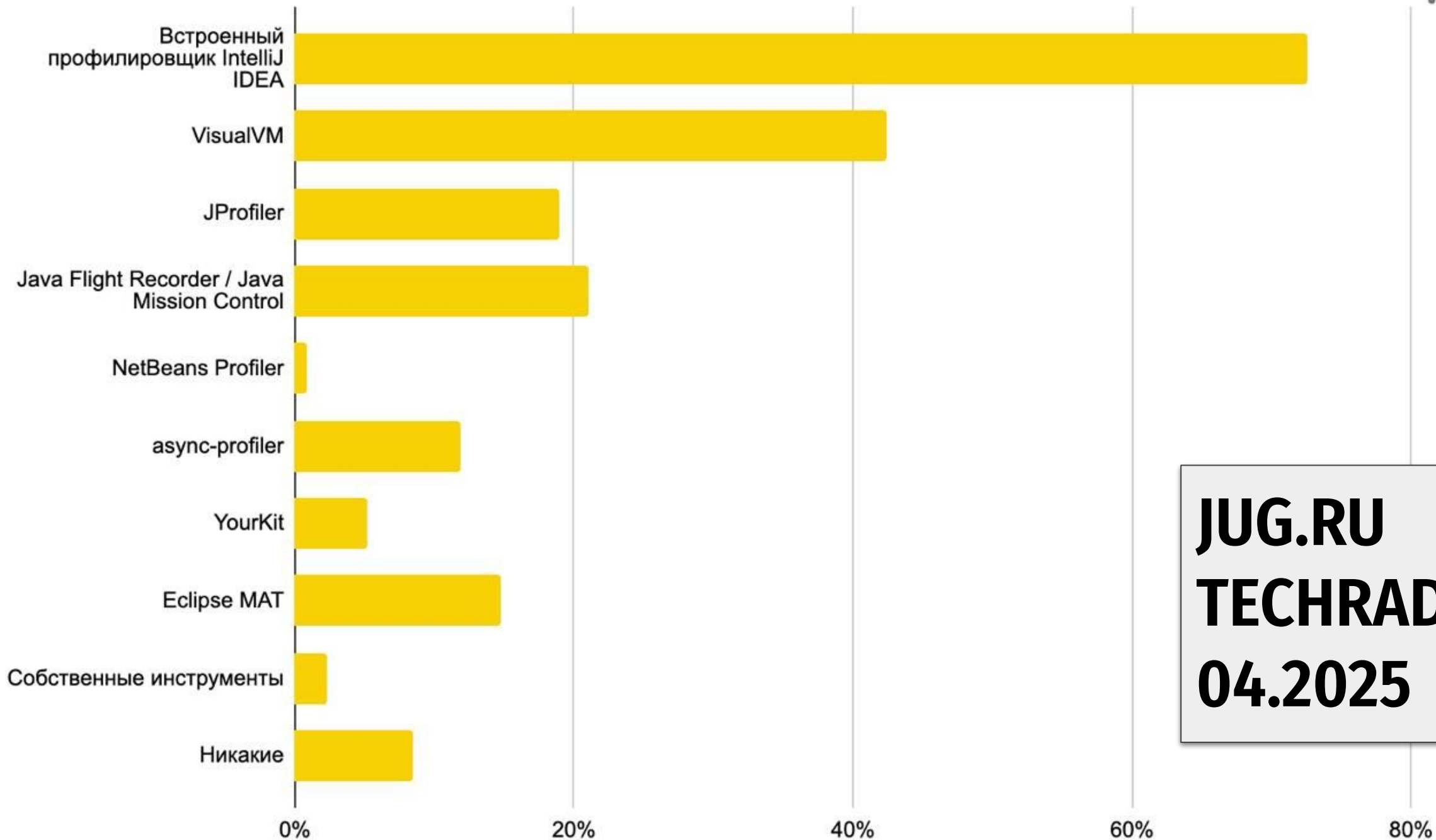


А ЕСЛИ ТОЧНЕЕ



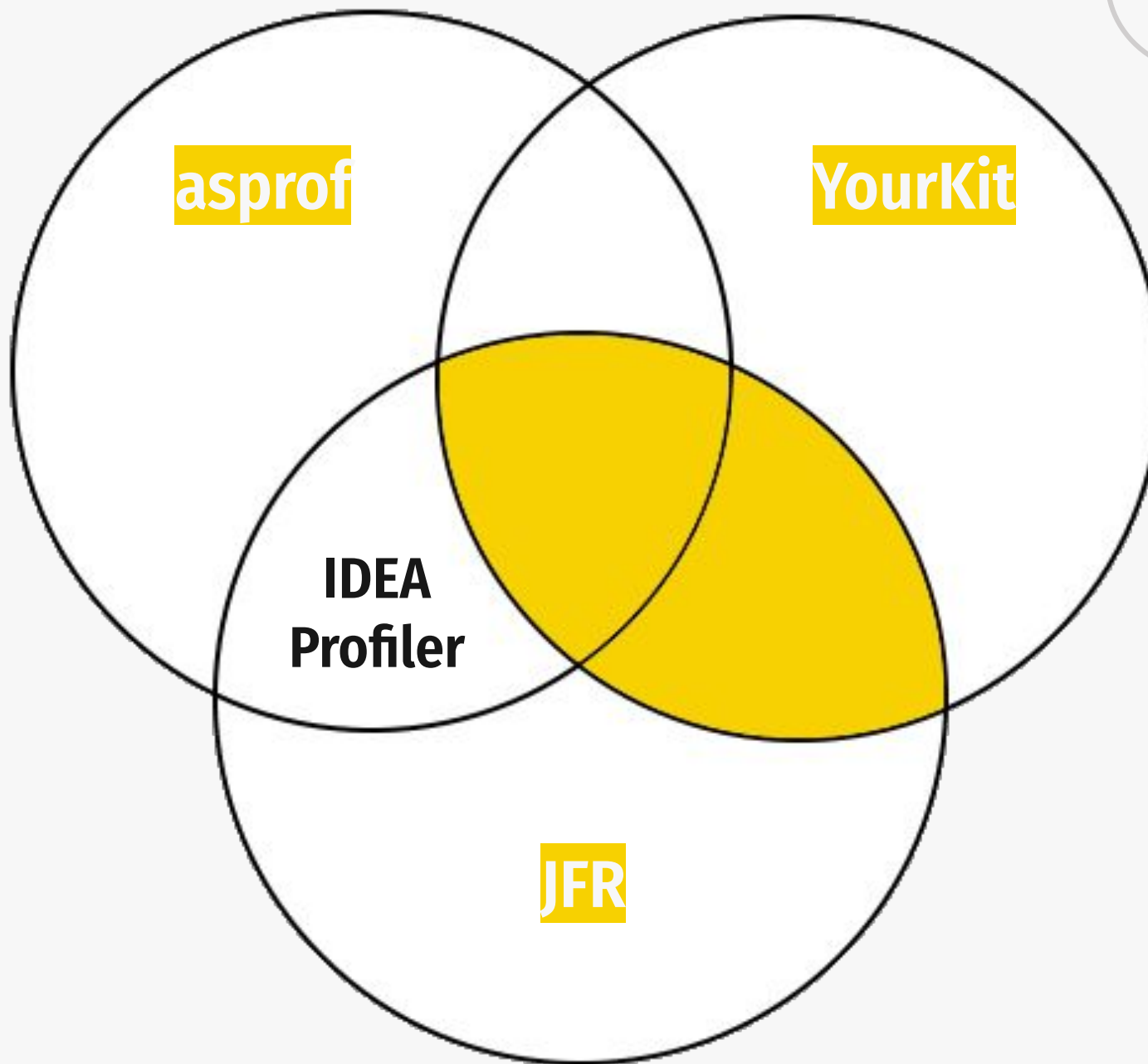
А ЕСЛИ ЕЩЕ ТОЧНЕЕ



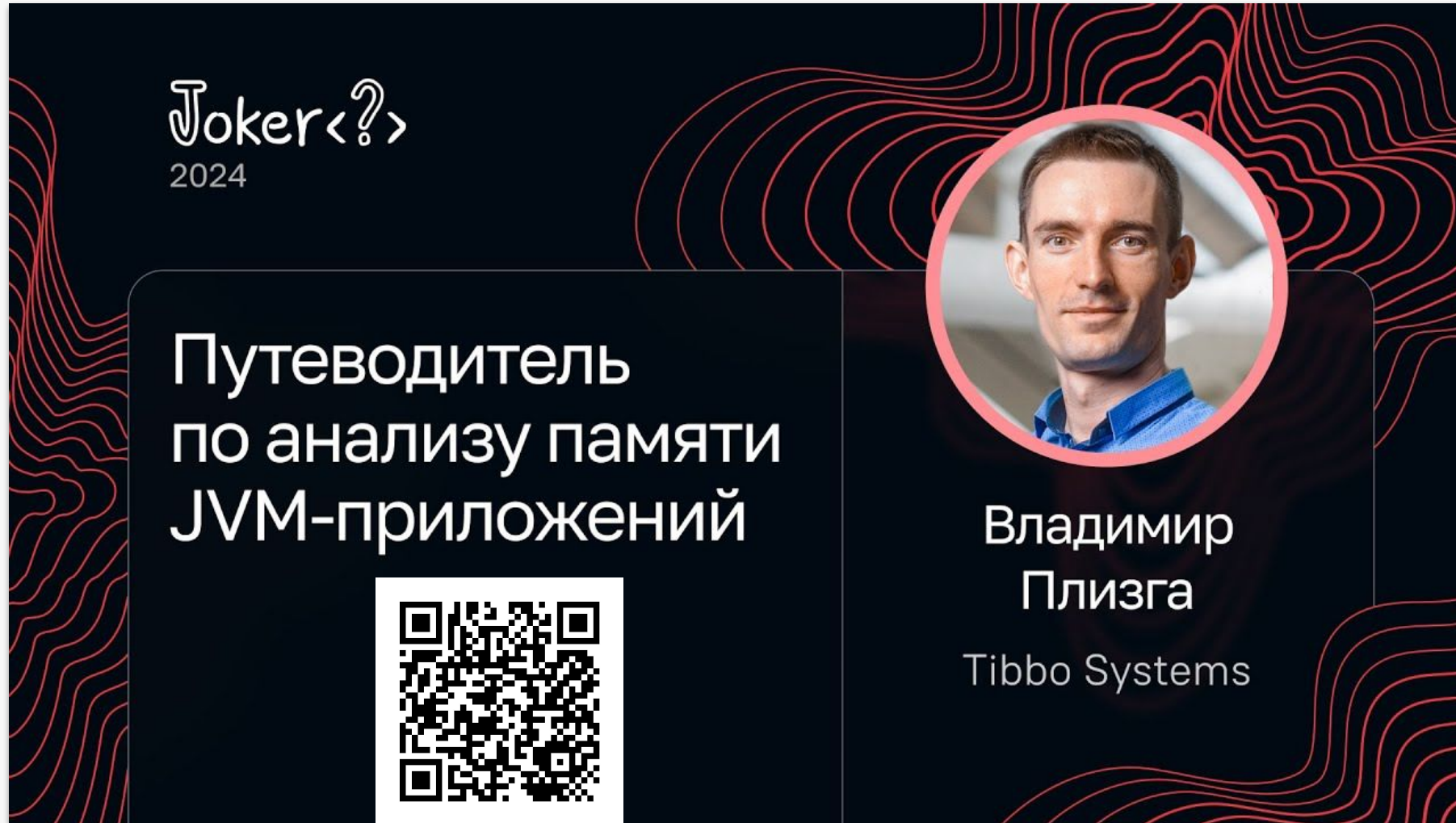


JUG.RU
TECHRADAR
04.2025

А ГДЕ ЖЕ ПРОФАЙЛЕР ИЗ IDEA?




О ПРОФИЛИРОВАНИИ ПАМЯТИ



Joker<?>
2024

Путеводитель
по анализу памяти
JVM-приложений



Владимир
Плизга
Tibbo Systems

<https://www.youtube.com/watch?v=fPns20-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

Joker<?> 2017

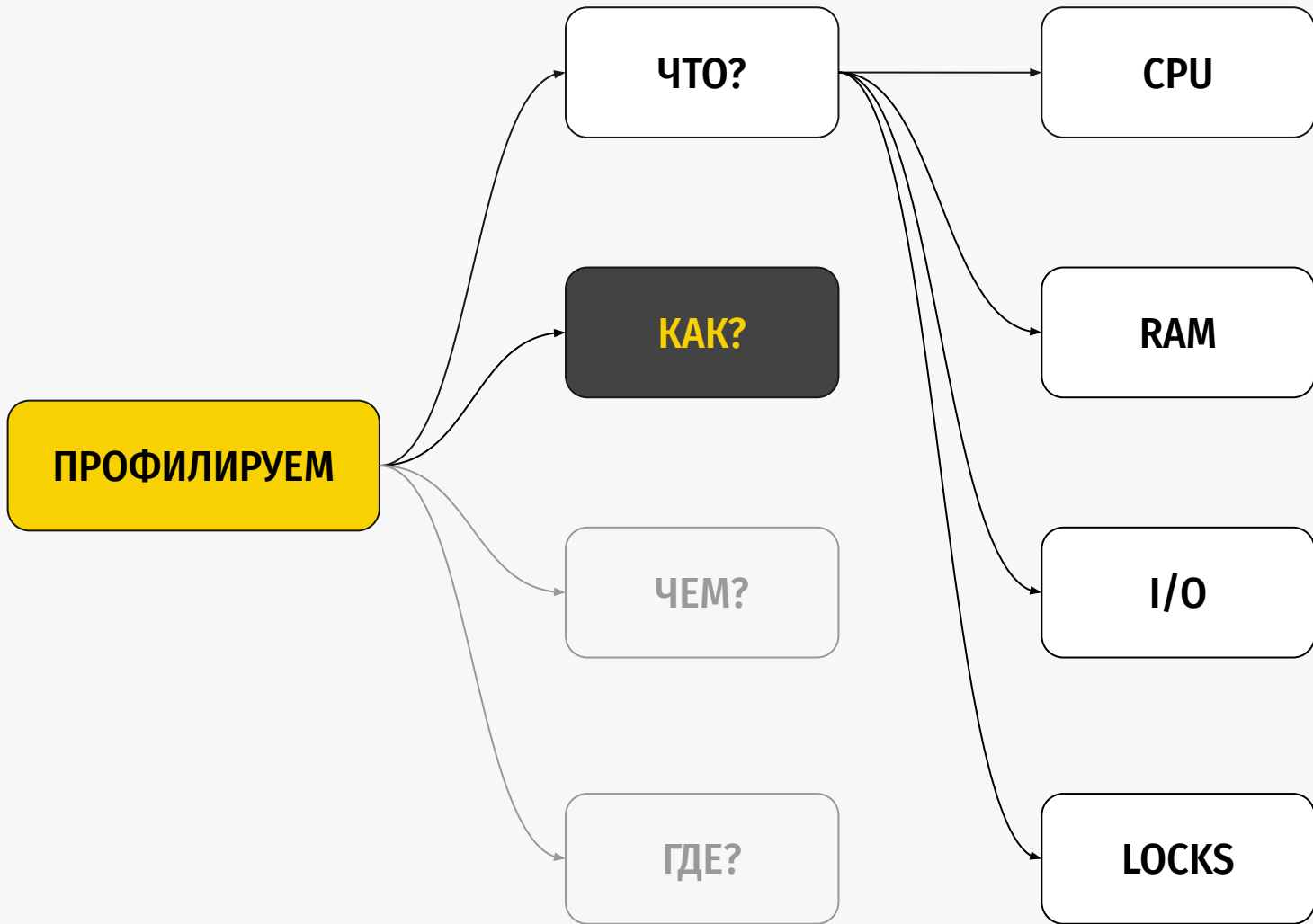
Nitsan Wakart

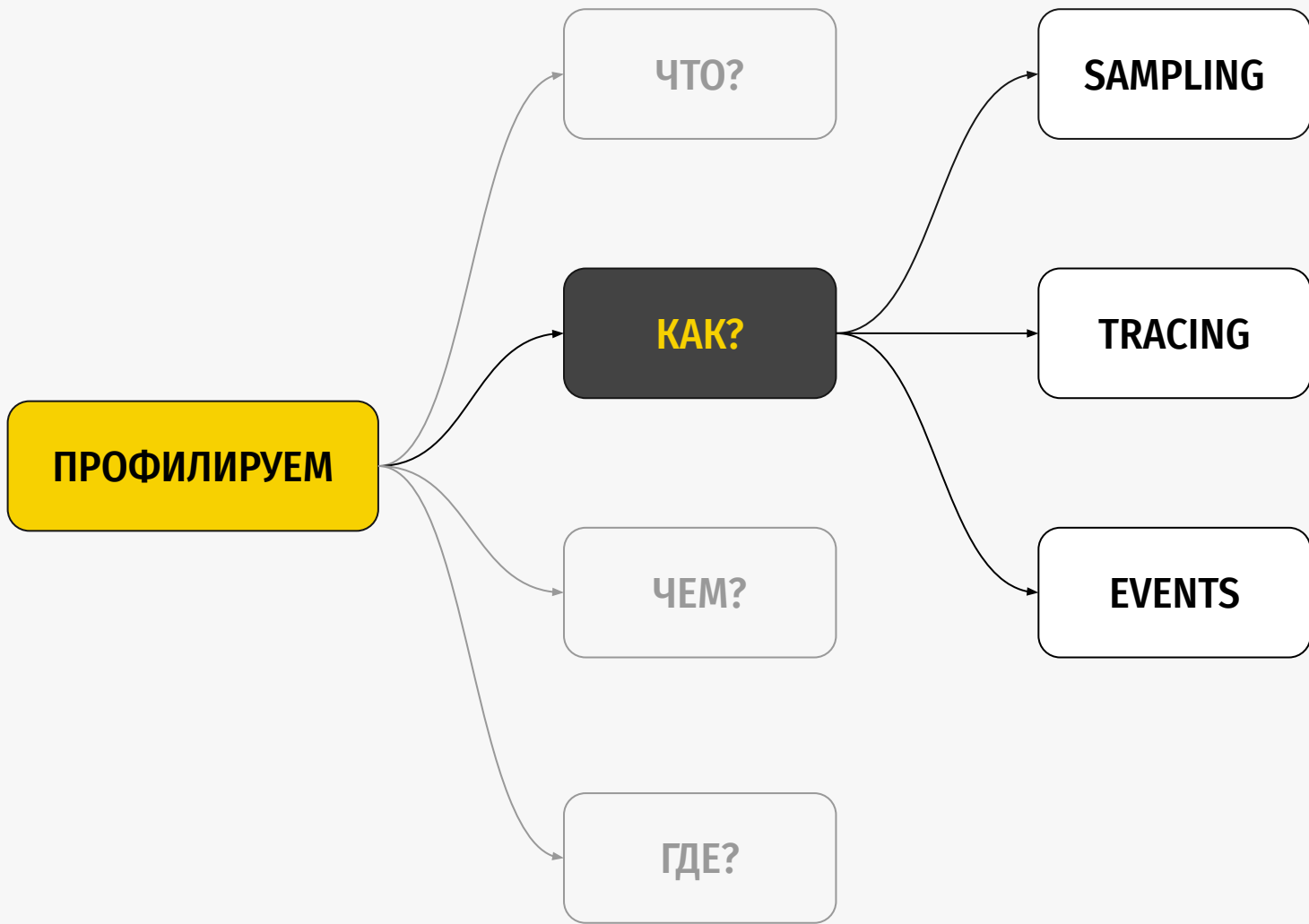
TTNR Labs

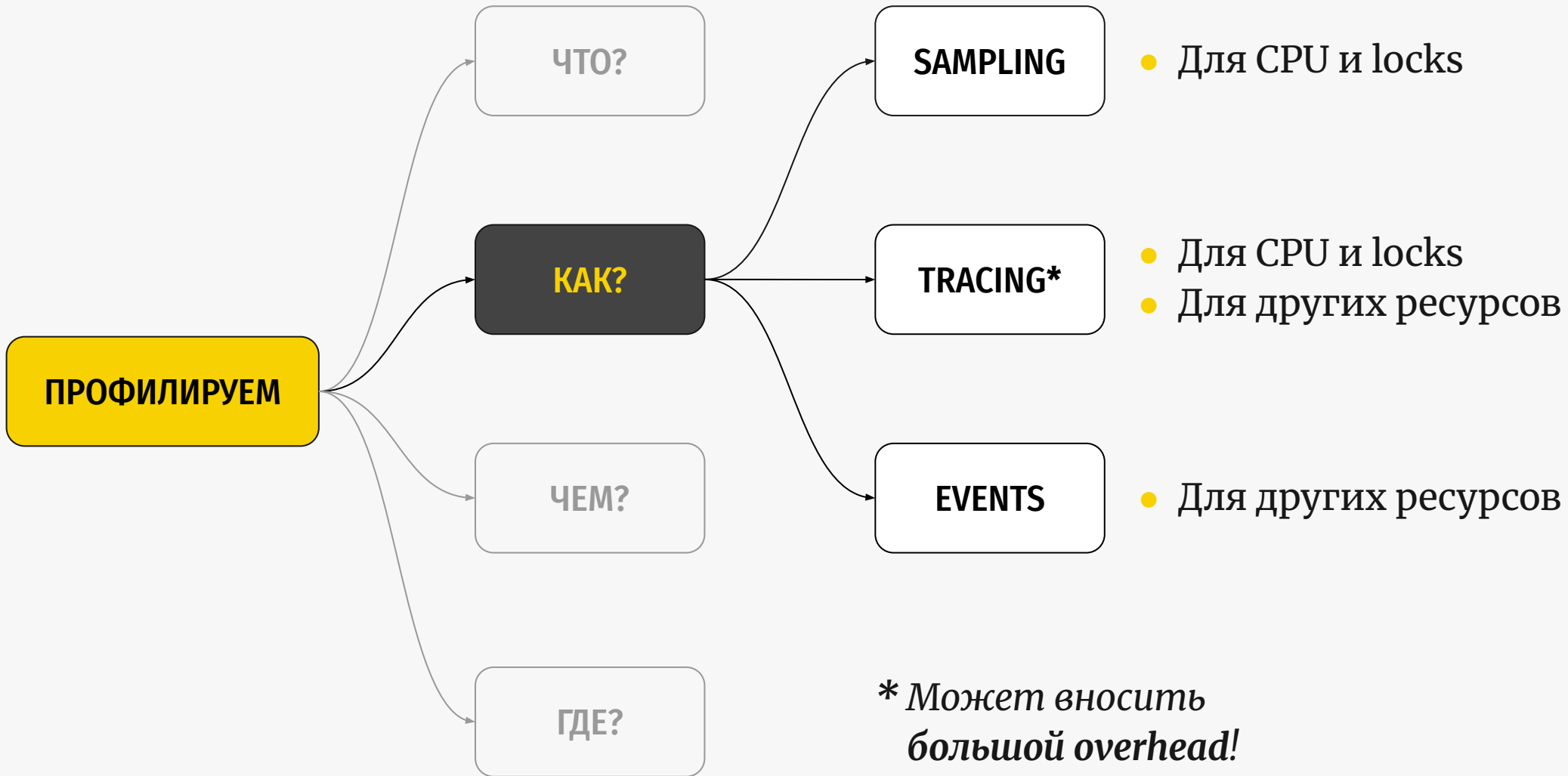
Profilers are lying
hobbitises



<https://www.youtube.com/watch?v=7lkHlqPeFjY>









ОБЩИЙ ПОДХОД

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

← если знаешь какое

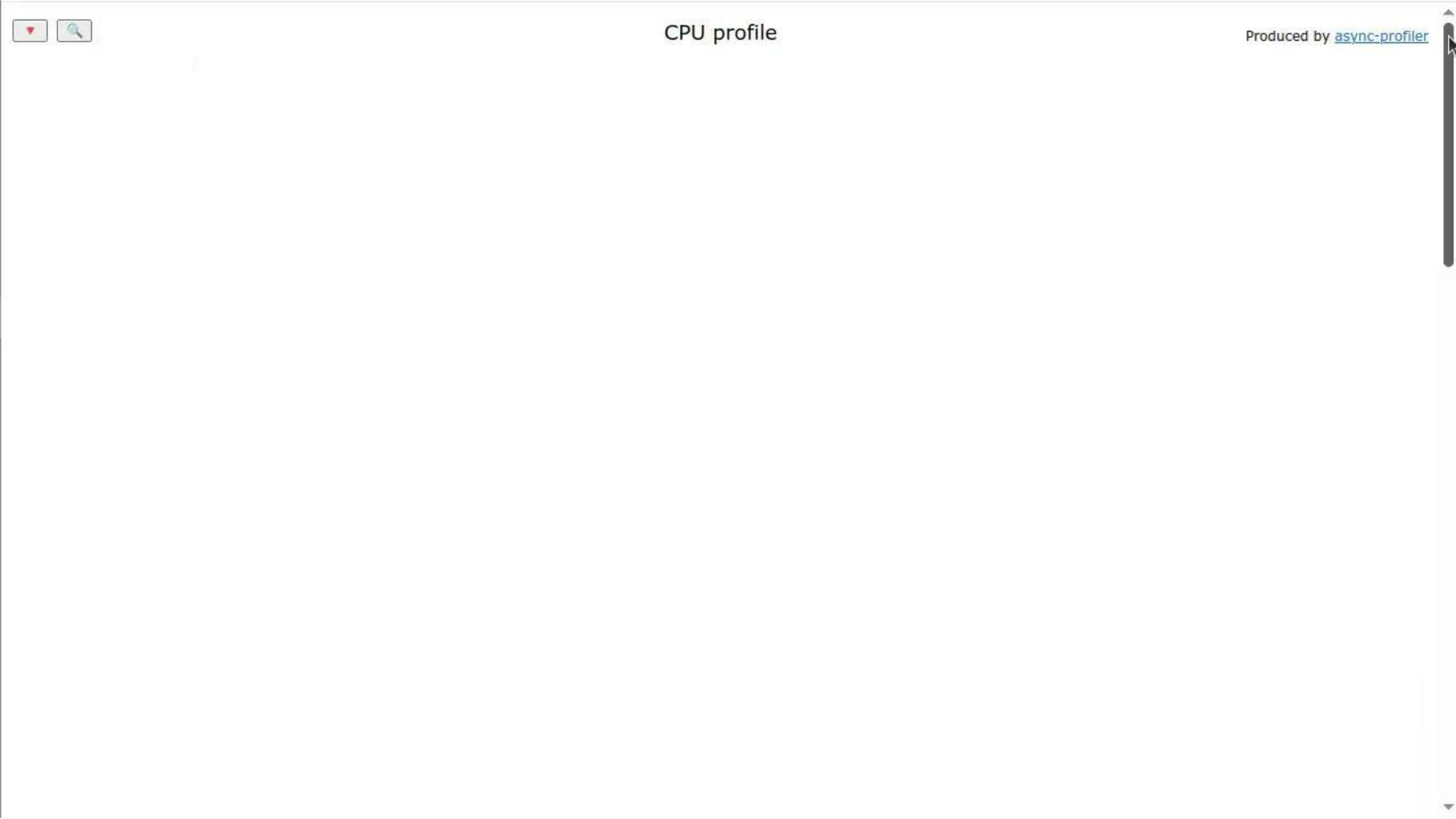


ВАРИАНТЫ РЕЗУЛЬТАТОВ ПРОФИЛИРОВАНИЯ

- 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

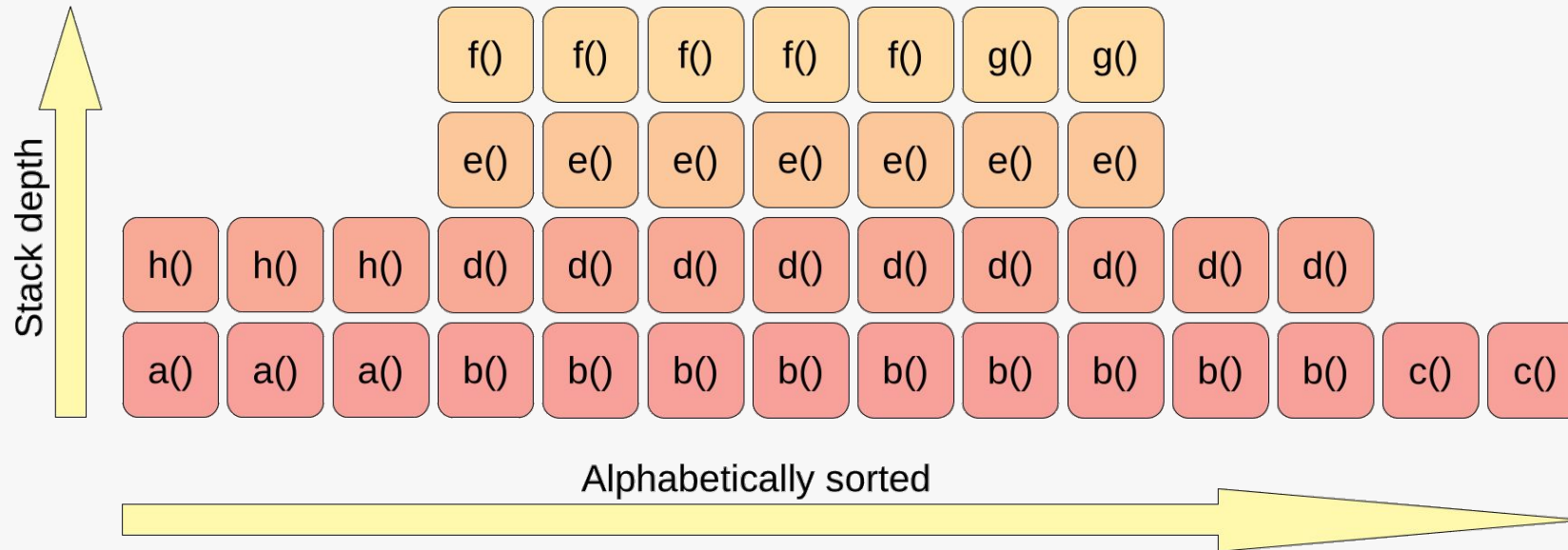
Produced by [async-profiler](#)





КАК СТРОИТСЯ FLAME GRAPH

1. Выписать стектрейсы в алфавитном порядке нижнего этажа

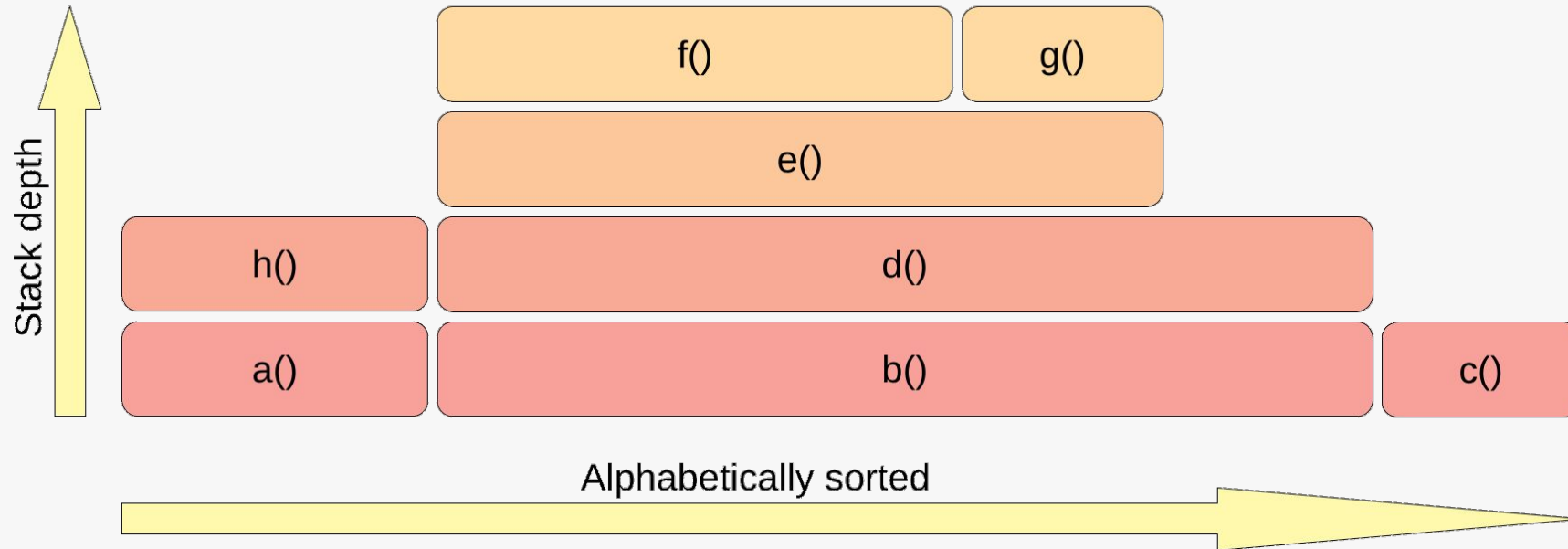


<https://krzysztofslusarski.github.io/2022/12/12/async-manual.html#flames>



КАК СТРОИТСЯ FLAME GRAPH

2. Объединить одинаковые соседние этажи

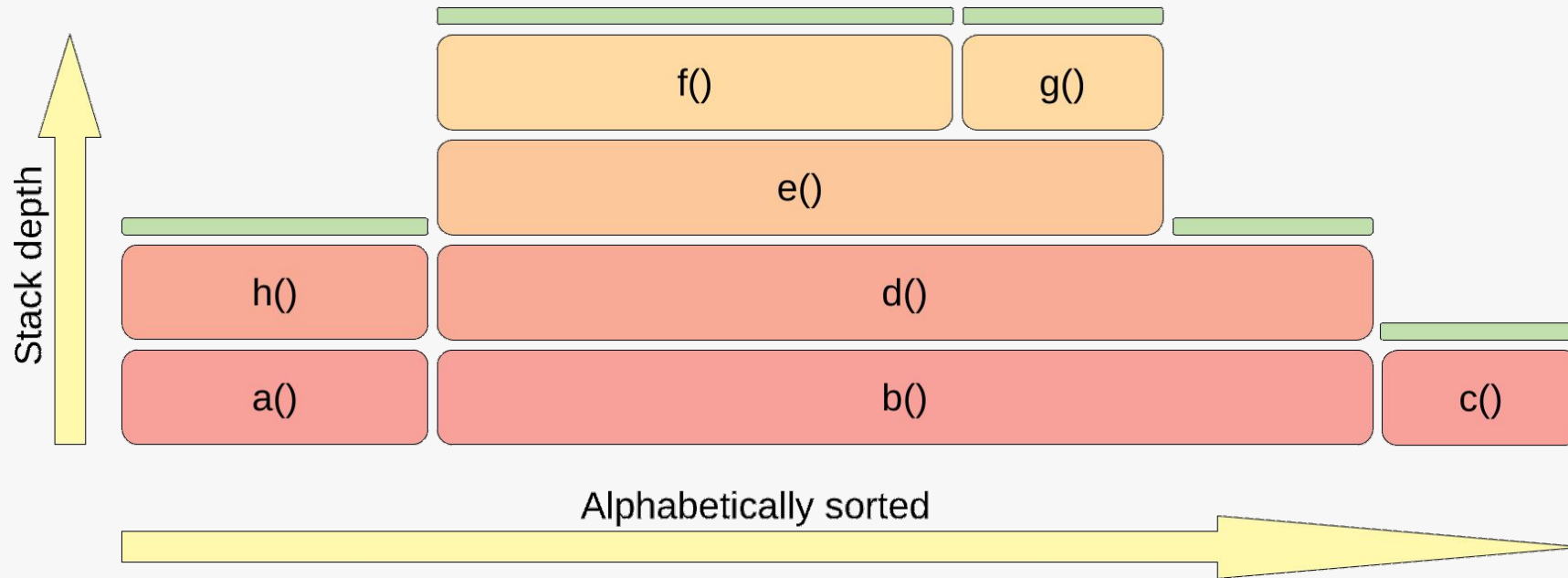


<https://krzysztofslusarski.github.io/2022/12/12/async-manual.html#flames>



КАК СТРОИТСЯ FLAME GRAPH

3. Обозначить верхний этаж по **измеряемому ресурсу**



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

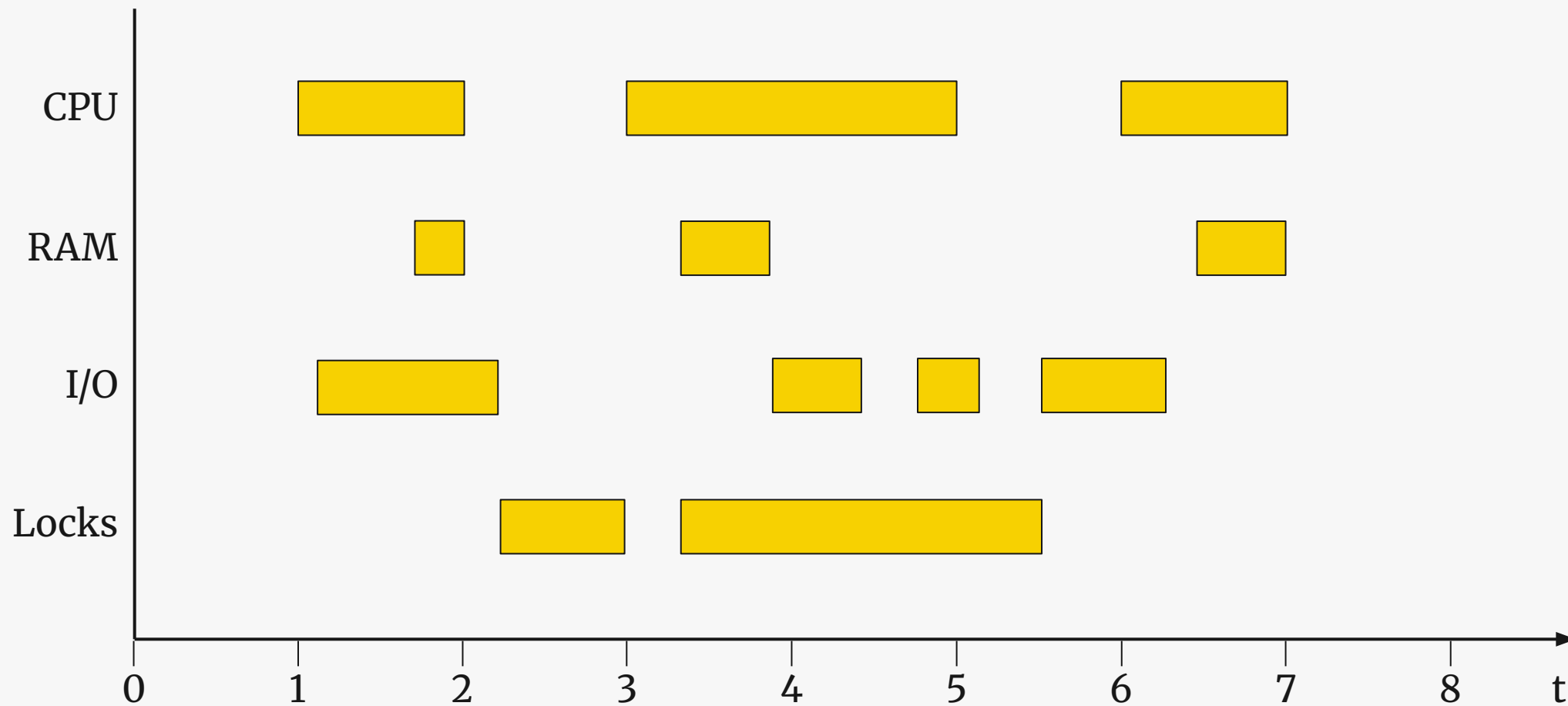
ПРИМЕРЫ ИЗМЕРЯЕМЫХ РЕСУРСОВ

- 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

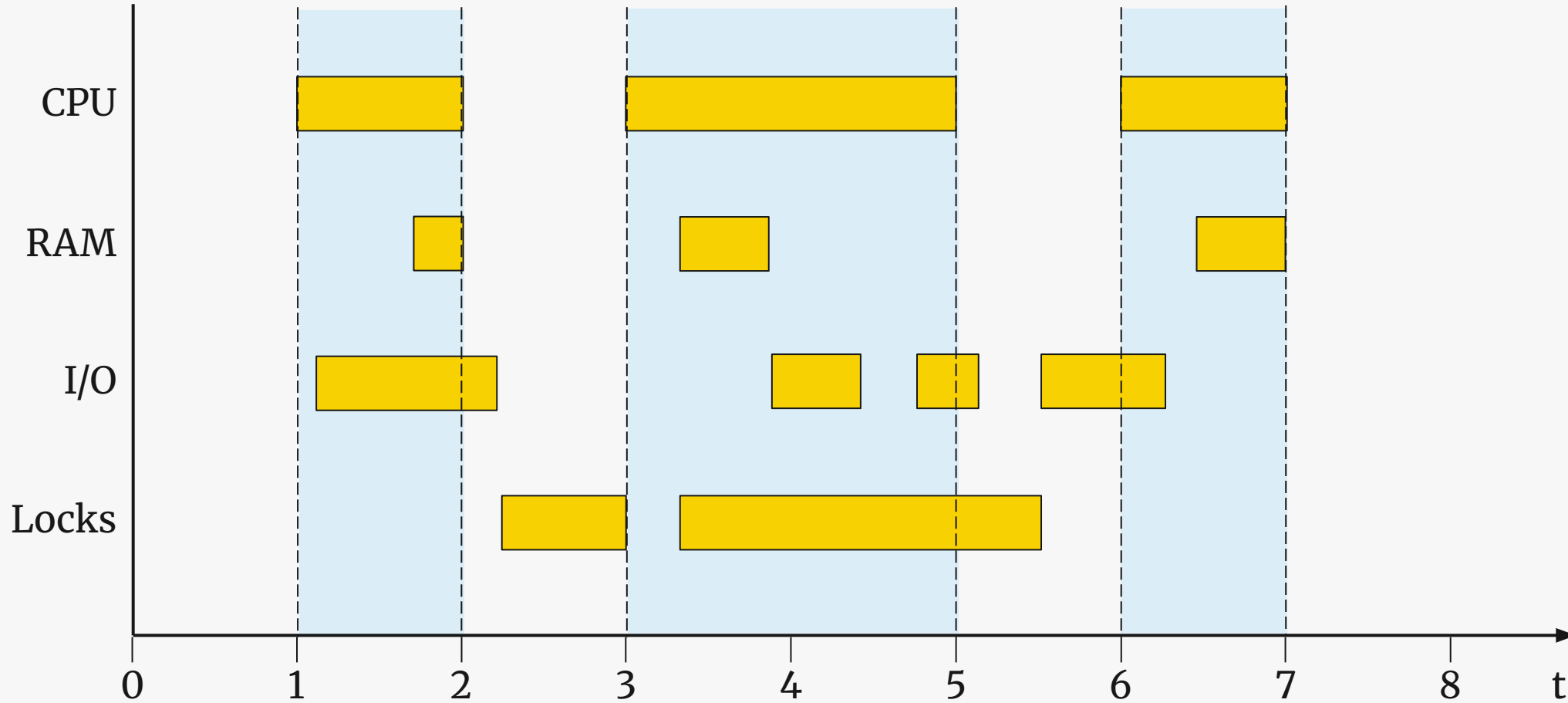
- CPU time – время, проведенное потоком на процессоре
- Wall clock time – общее время потока
 - Включая ожидание (в т. ч. I/O)
 - Включая блокировки
 - Включая сон

ПРОФИЛЬ ОДНОГО ВЫЗОВА МЕТОДА



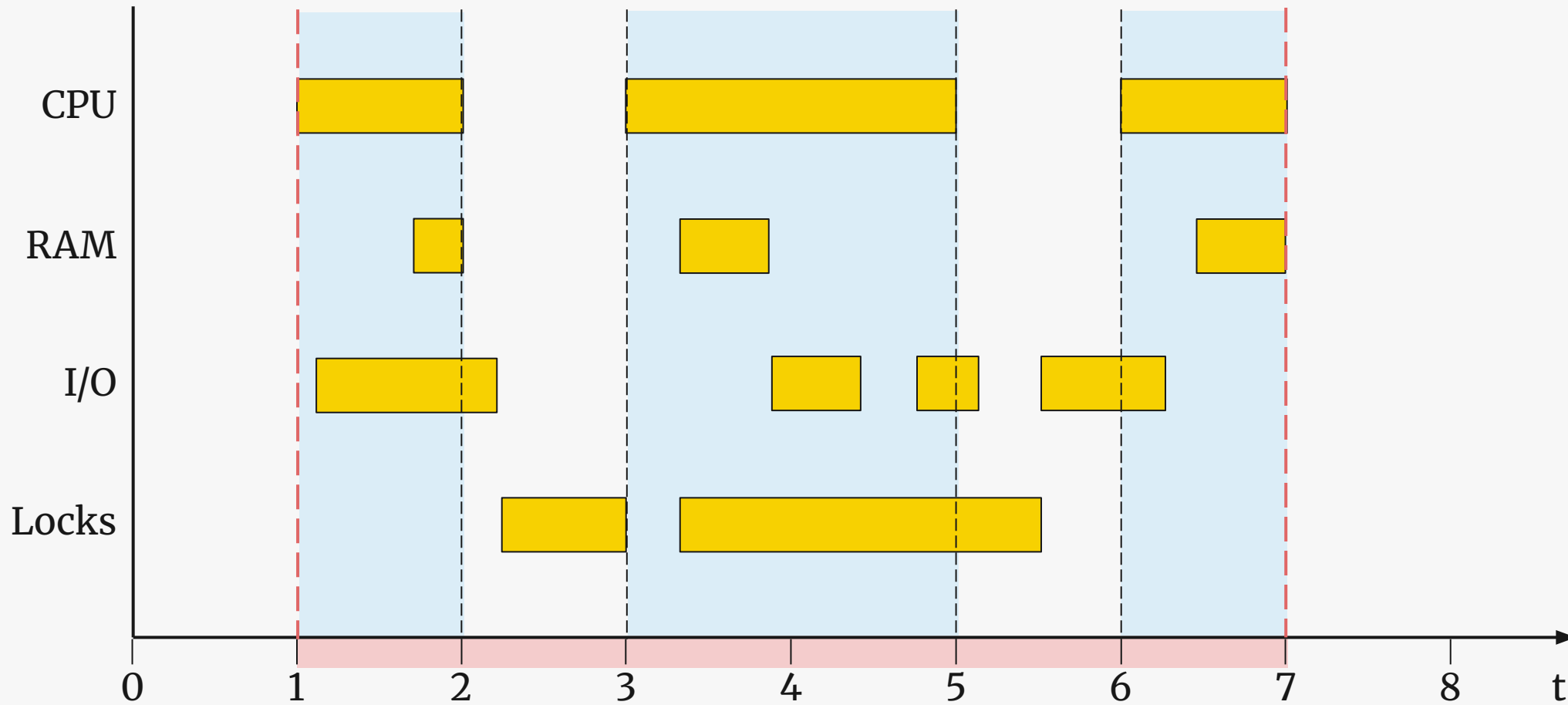
 Периоды владения ресурсами за 1 вызов метода

CPU TIME



— CPU time (1+2+1=4)

CPU TIME VS WALL CLOCK TIME

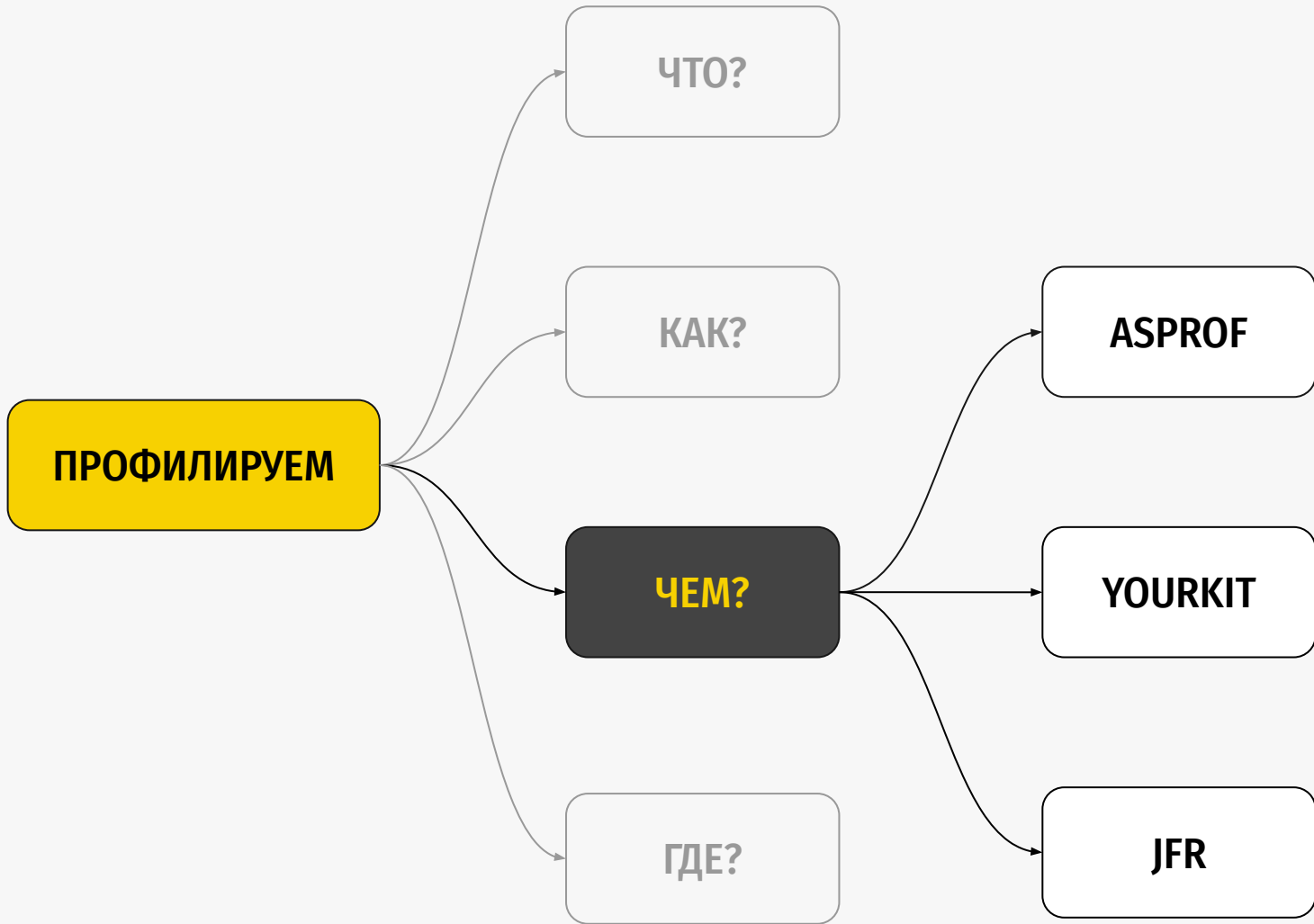


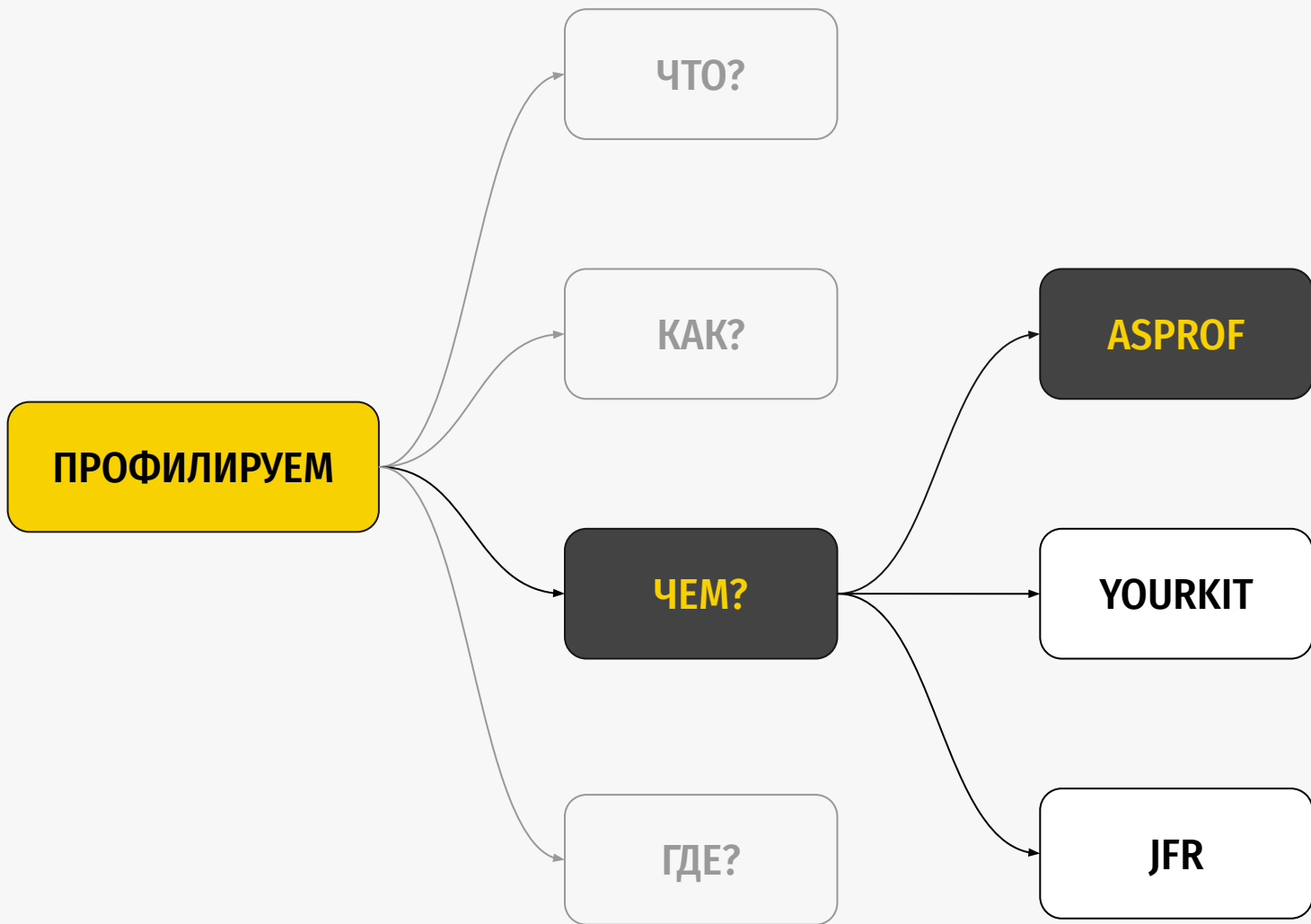
CPU time (1+2+1=4)

Wall-clock time (7-1=6)









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 Кошелёв
Контур

▾ Биография

Страница спикера →

ASync-PROFILER: CPU TIME

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

ПАЛИТРА FLAME GRAPH В ASPROF

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

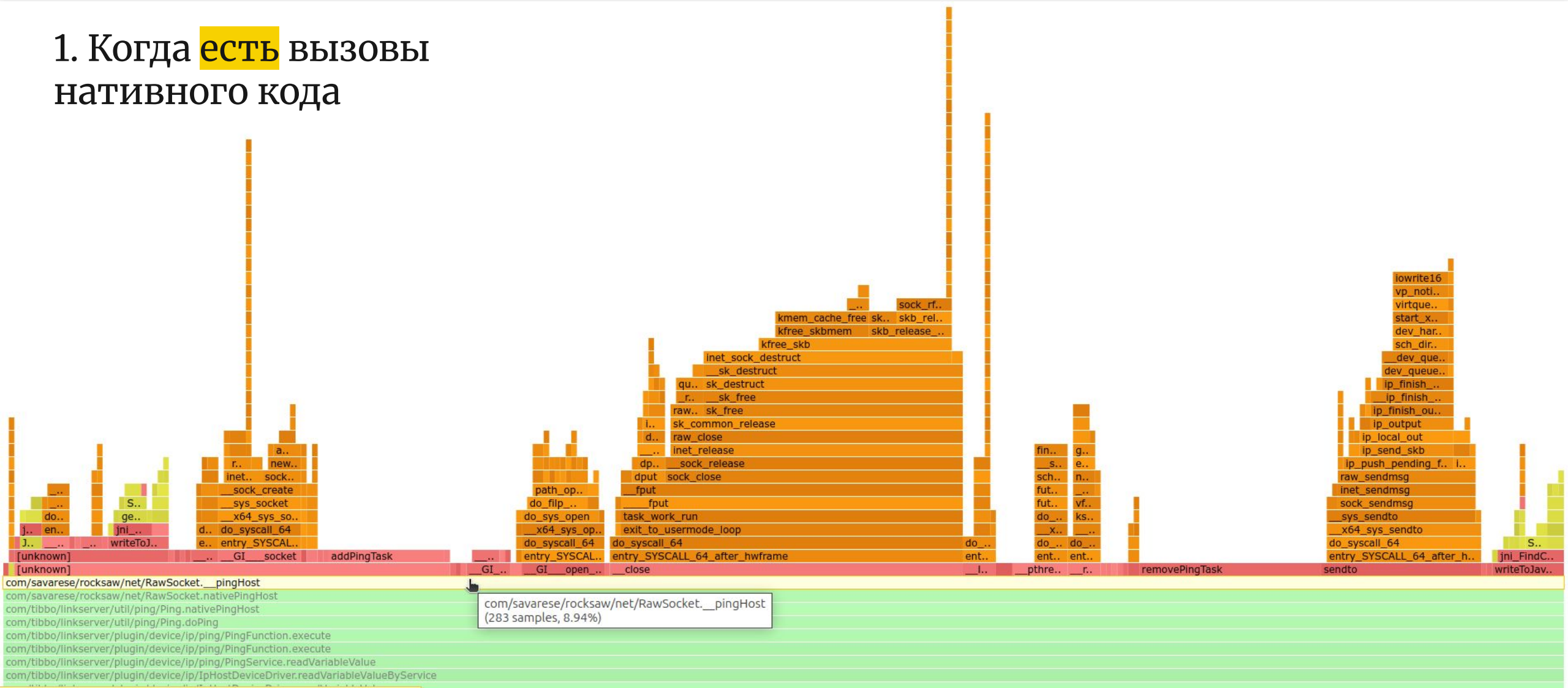
<https://github.com/async-profiler/async-profiler/blob/master/docs/FlamegraphInterpretation.md>

КОГДА НУЖНЫ НАТИВНЫЕ ФРЕЙМЫ

- Когда **есть** нативный код

КОГДА НУЖНЫ НАТИВНЫЕ ФРЕЙМЫ

1. Когда **есть** вызовы
нативного кода

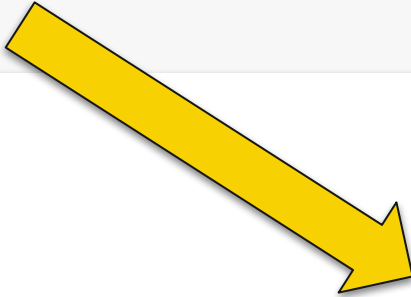


КОГДА НУЖНЫ НАТИВНЫЕ ФРЕЙМЫ

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

КОГДА НУЖНЫ НАТИВНЫЕ ФРЕЙМЫ

2. Когда **нет** вызовов
нативного кода



java/util/regex/Pattern\$CharPredicate.lambda\$negate\$3	Mutex::lock
java/util/regex/Pattern\$CharPredicate\$\$Lambda\$39.0x00000008001129a0.is	G1CollectedHeap::attempt_allocation_slow
java/util/regex/Pattern\$CharProperty.match	G1CollectedHeap::allocate_new_tlab
java/util/regex/Pattern\$StartS.match	Mutex::lock
java/util/regex/Matcher.search	Mutex::lock
java/util/regex/Matcher.find	G1CollectedHeap::allocate_new_tlab (5 samples, 0.01%)
java/util/regex/Pattern.split	Mutex::lock
java/lang/String.split	TypeArrayKlass::allocate_common
java/lang/String.split	OptoRuntime::new_array_C
com/tibbo/aggregate/common/datatable/field/DateFieldFormat.dateFromString	java/util/regex/Matcher.<init>
com/tibbo/aggregate/common/datatable/field/DateFieldFormat.valueFromString	java/util/regex/Pattern.matcher
com/tibbo/aggregate/common/datatable/field/DateFieldFormat.valueFromString	
com/tibbo/aggregate/common/datatable/FieldFormat.valueFromEncodedString	
com/tibbo/aggregate/common/datatable/DataRecord.setData	
com/tibbo/aggregate/common/datatable/DataRecord.<init>	

ASYNС-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

```
byte[]
java.io/ByteArrayOutputStream.<init>
org.springframework.web.util/ContentCachingRequestWrapper.<init>
org.springframework.web.filter/AbstractRequestLoggingFilter.doFilterInternal
org.springframework.web.filter/OncePerRequestFilter.doFilter
org.apache.catalina.core/ApplicationFilterChain.internalDoFilter
org.apache.catalina.core/ApplicationFilterChain.doFilter
org.springframework.web.filter/RequestContextFilter.doFilterInternal
org.springframework.web.filter/OncePerRequestFilter.doFilter
org.apache.catalina.core/ApplicationFilterChain.internalDoFilter
org.apache.catalina.core/ApplicationFilterChain.doFilter
org.springframework.web.filter/FormContentFilter.doFilterInternal
org.springframework.web.filter/OncePerRequestFilter.doFilter
org.apache.catalina.core/ApplicationFilterChain.internalDoFilter
org.apache.catalina.core/ApplicationFilterChain.doFilter
org.springframework.web.filter/ServerHttpObservationFilter.doFilterInternal
org.springframework.web.filter/OncePerRequestFilter.doFilter
```

byte[]
(5,242,896,000 samples, 99.99%)

<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)

Build, Execution, Deployment > Java Profiler

Build, Execution, Deployment

- Coverage
- > Debugger
- > Deployment
- > Docker
- Java Profiler** 1
- Filters
- Remote Jar Repositories
- Required Plugins

IntelliJ Profiler 2

Name: IntelliJ Profiler

Agent options: event=wall,interval=10ms,jfrsync=profile 3

Agent: Bundled (Version: 3.0)

[Async profiler README.md](#)

Collect native calls (GC threads, JNI calls, etc.) 3

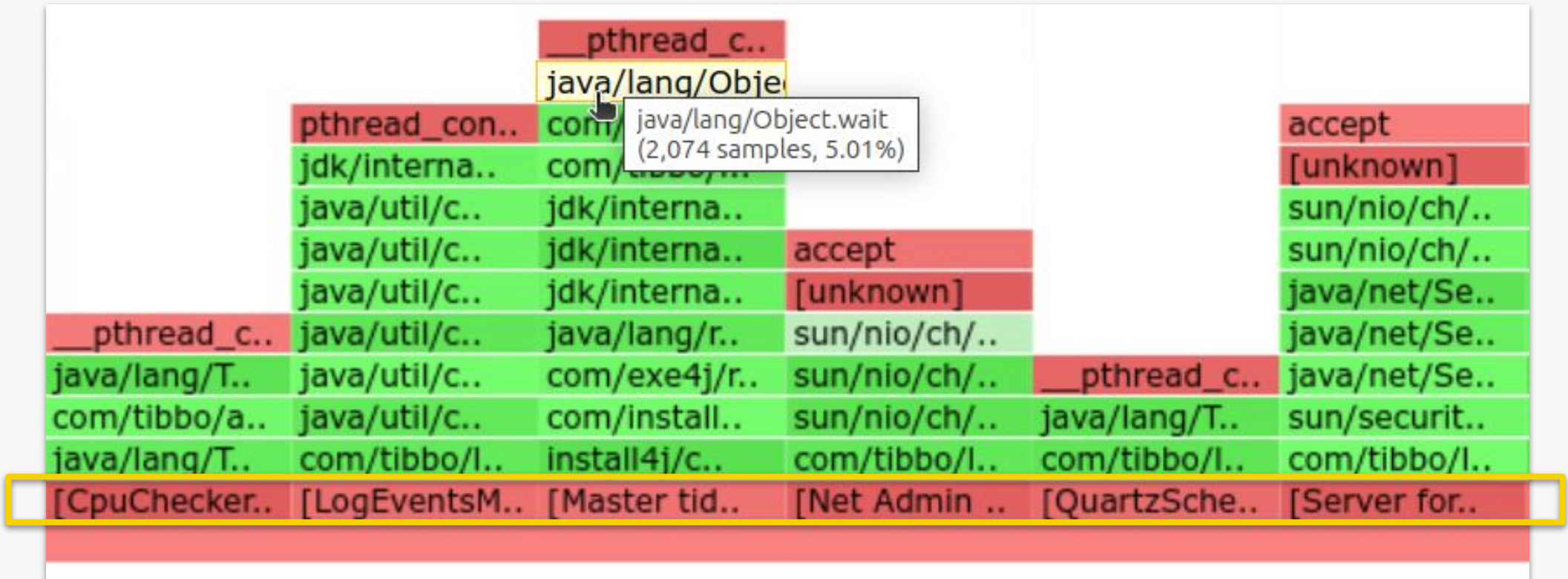
**КАКОЙ РЕЖИМ ЛУЧШЕ,
ЕСЛИ НЕПОНЯТНО,
КУДА СМОТРЕТЬ?**

ASYNCPROFILER: 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

Имена
ПОТОКОВ
→



java/lang/Object.wait
(2,074 samples, 5.01%)

ASYNС-PROFILER: ДРУГИЕ РЕЖИМЫ

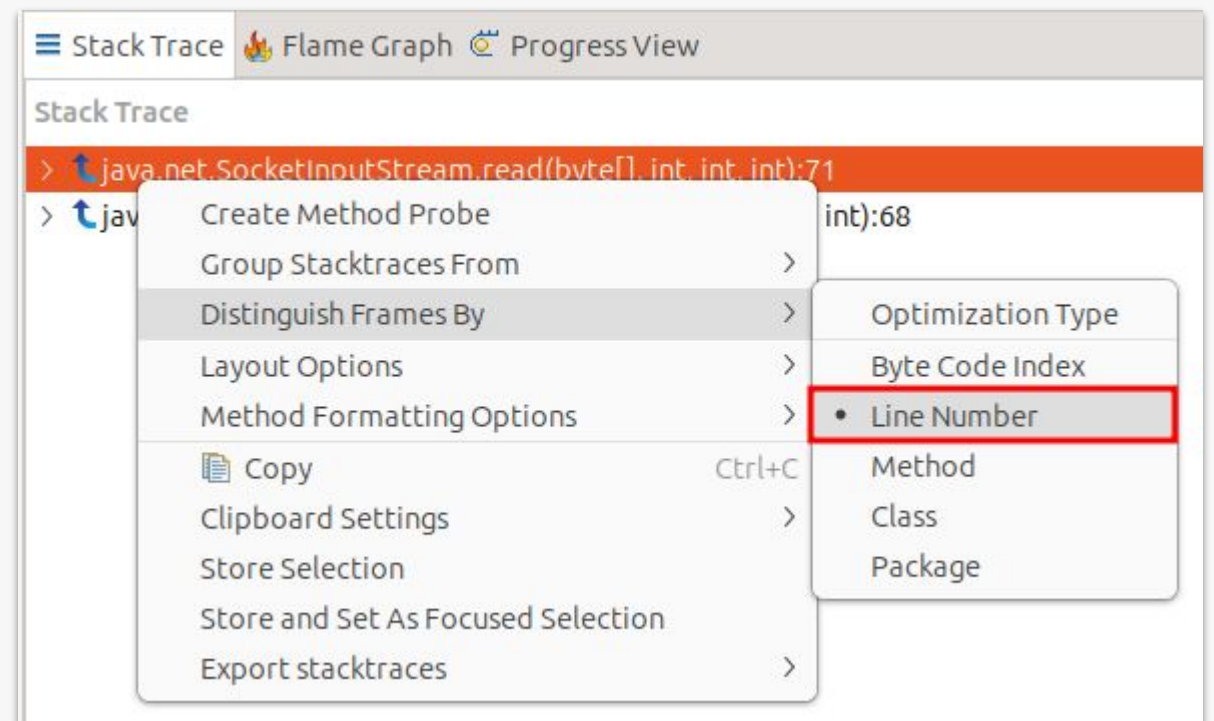
- Аллокации нативной памяти (`-e nativemem`)
- Блокировки (`-e lock`)
- Отдельные Java-методы (`-e ClassName.methodName`)
- И не Java тоже:
 - `Java_java_lang_ClassLoader_defineClass1`
 - `Java_java_lang_Throwable_fillInStackTrace`
 - `JVM_StartThread`
 - `G1CollectedHeap::humongous_obj_allocate`
- Прочие

ОТЛОВ ОГРОМНЫХ АЛЛОКАЦИЙ (G1 GC)

```
G1CollectedHeap::humongous_obj_allocate  
G1CollectedHeap::mem_allocate  
MemAllocator::allocate  
TypeArrayKlass::allocate_common  
Runtime1::new_type_array  
java.io/ByteArrayOutputStream.<init>  
org.springframework.web.util/ContentCachingRequestWrapper.<init>  
org.springframework.web.filter/AbstractRequestLoggingFilter.doFilterInternal  
org.springframework.web.filter/OncePerRequestFilter.doFilter  
org.apache.catalina.core/ApplicationFilterChain.internalDoFilter  
org.apache.catalina.core/ApplicationFilterChain.doFilter  
org.springframework.web.filter/RequestContextFilter.doFilterInternal  
org.springframework.web.filter/OncePerRequestFilter.doFilter  
org.apache.catalina.core/ApplicationFilterChain.internalDoFilter  
org.apache.catalina.core/ApplicationFilterChain.doFilter  
org.springframework.web.filter/FormContentFilter.doFilterInternal  
org.springframework.web.filter/OncePerRequestFilter.doFilter  
org.apache.catalina.core/ApplicationFilterChain.internalDoFilter
```

ASYNC-PROFILER: ГДЕ НОМЕРА СТРОК?

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





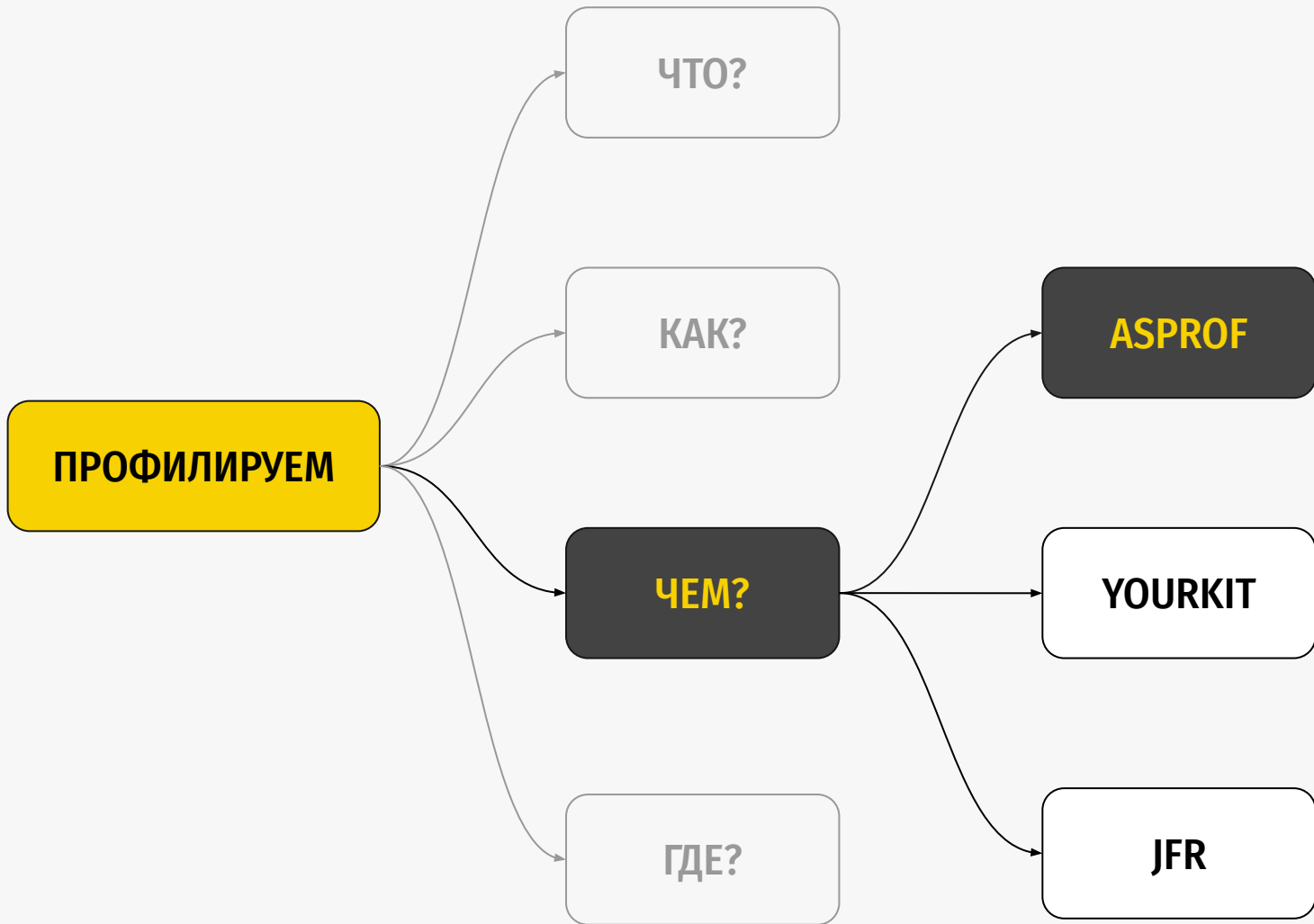
КОВАЛЬСКИ!
НУ ЧТО ТАМ?

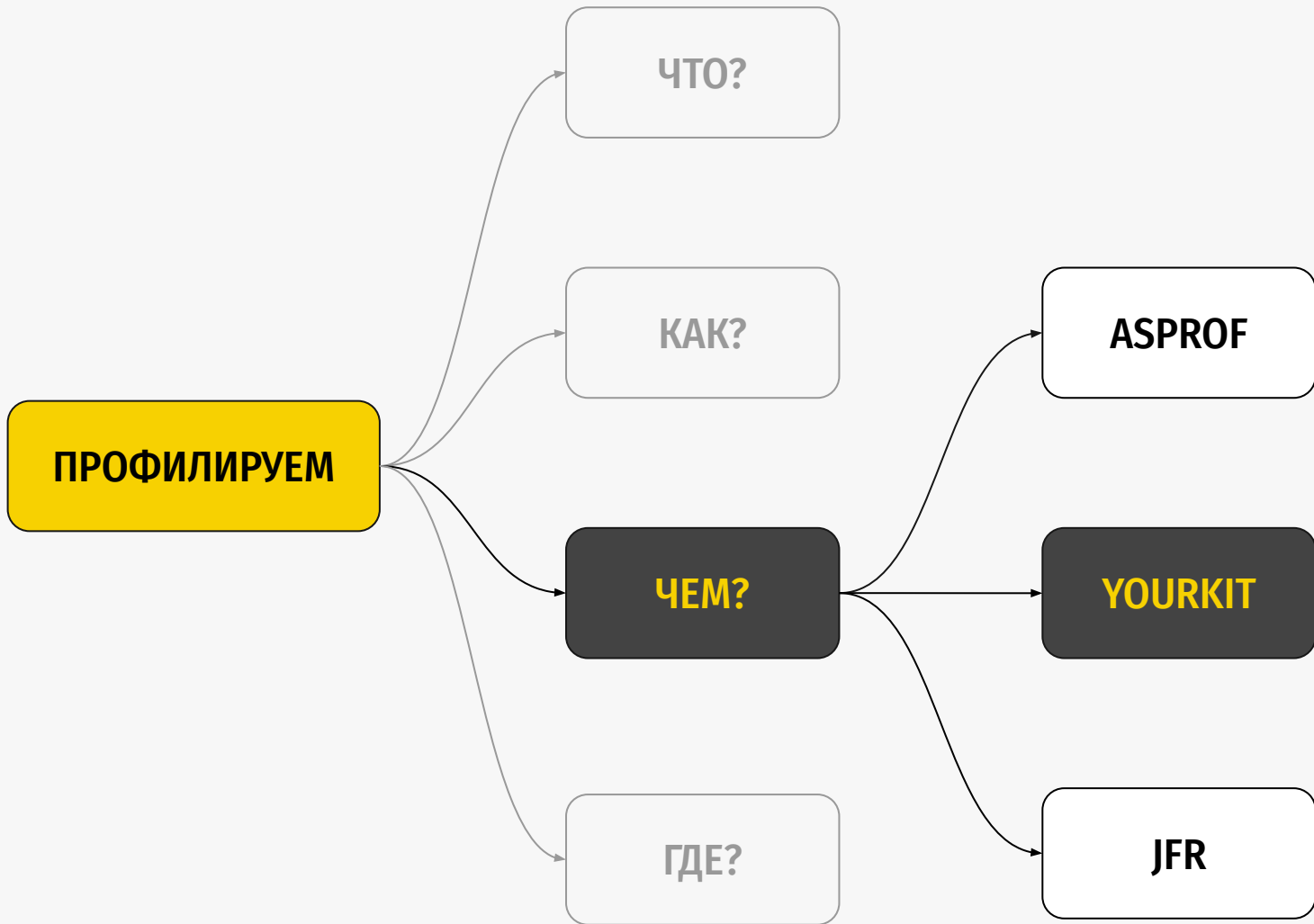


ПО-МОЕМУ ЭТО
НЕ МАДАГАСКАР...

ASYNС-PROFILER: ЧЕГО НЕ ХВАТАЕТ

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





YOURKIT: ОБЩЕЕ

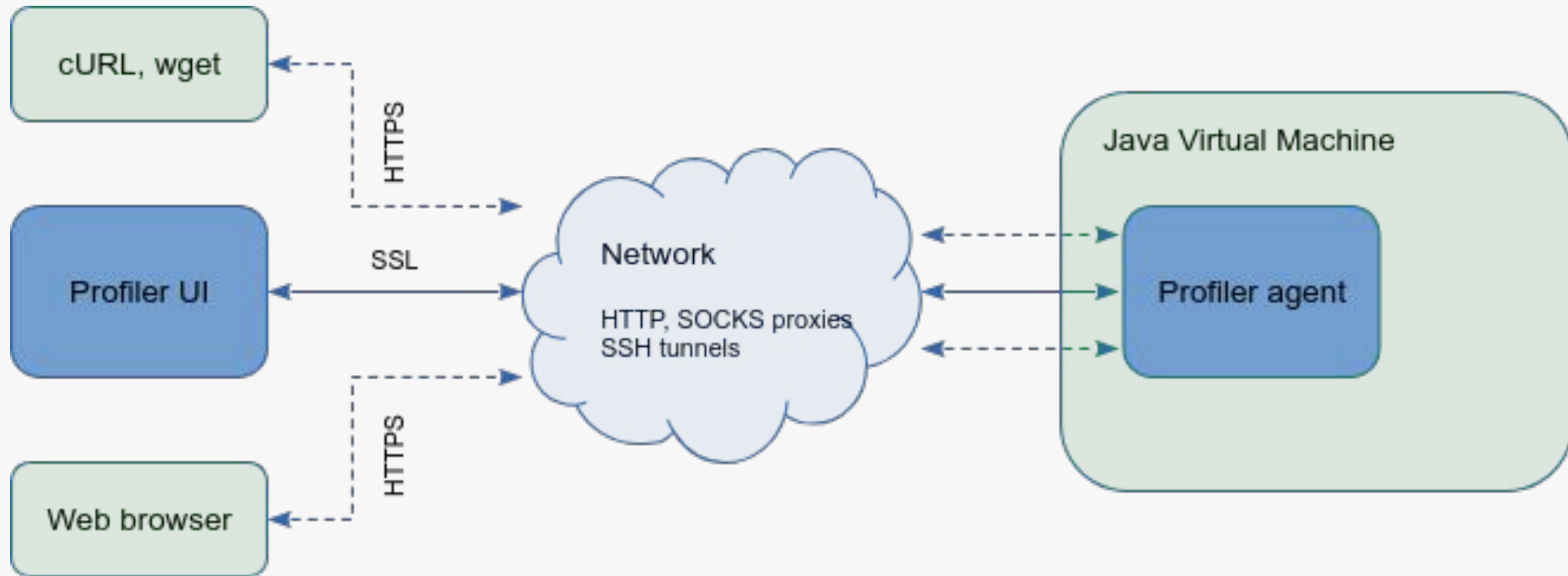


- Доступность: платные лицензии
- Разработчик: YourKit GmbH
- Тип: sampling & tracing
- Интерфейсы:
 - GUI
 - CLI
 - HTTP API
 - Java API

**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**



Start CPU profiling 

- Sampling
- Asynchronous sampling
- Tracing
- Call counting

← 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, boolean)	28,221	6 %	493
SunGraphics2D.java:463	sun.java2d.SunGraphics2D.intersectRectShape(Rectangle2D, Shape)	28,221	6 %	493
SunGraphics2D.java:509	sun.java2d.SunGraphics2D.intersectByArea(Shape, Shape, boolean)	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, double, double)	≥ 26	0 %	≥ 35,272
Area.java:176	<...> sun.awt.geom.Curve.insertCubic(Vector, double, double, double, double, double, double)	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, s	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?

CPU profiling
Sampling

- Performance charts
- Call tree – All threads merged
- Call tree – By thread
- Flame graph
- Hot spots
- Method list
- Java EE
- Database**
- JSPs and servlets
- JNDI

Hide nested calls
 Group queries by type
 Include prepareStatement()/prepareCall()
 Hide zero time calls

Call	Time (ms)	Count
(JPA/EclipseLink) getResultList	25,881 40 %	
(JPA/EclipseLink) getSingleResult	23,240 35 %	
Select	11,216 17 %	
SELECT CUSTOMER_ID AS a1, ADDRESSLINE1 AS a2, ADDRESSLINE2 AS a3, CI	5,700 9 %	20,000
SELECT COUNT(CUSTOMER_ID) FROM CUSTOMER <Open>	5,476 8 %	20,000
SELECT MANUFACTURER_ID, ADDRESSLINE1, ADDRESSLINE2, CITY, EMAIL, FA	9 0 %	8
SELECT MANUFACTURER_ID AS a1, ADDRESSLINE1 AS a2, ADDRESSLINE2 AS	3 0 %	6
SELECT DISCOUNT_CODE, RATE FROM DISCOUNT_CODE WHERE (DISCOUNT	3 0 %	4
SELECT COUNT(MANUFACTURER_ID) FROM MANUFACTURER <Open>	3 0 %	6
SELECT PROD_CODE, DESCRIPTION, DISCOUNT_CODE FROM PRODUCT_COD	2 0 %	3
SELECT ORDER_NUM, FREIGHT_COMPANY, QUANTITY, SALES_DATE, SHIPPING	2 0 %	3
SELECT PRODUCT_ID AS 1, AVAILABLE AS 2, DESCRIPTION AS 3, MARKUP	2 0 %	4

Reverse Call Tree

Call	Time (ms)
org.apache.derby.client.am.PreparedStatement.executeQuery()	11,216 100 %
org.eclipse.persistence.internal.jpa.QueryImpl.getResultList() QueryImpl.java:473	5,722 51 %
org.eclipse.persistence.internal.jpa.EJBQueryImpl.getSingleResult() EJBQueryImpl.java:400	5,483 49 %
java.lang.Thread.run() Thread.java:745	9 0 %

YOURKIT: ДРУГИЕ РЕЖИМЫ

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

Events:

JSP/Servlet (2,303)

▶ SQL (4)

▶ Socket (163)

▶ File (348)

▶ Process (12)

▶ Thread (1,804)

Thread park (0)

Class Loading (16,228)

Message (8)

▶ Directory Stream (0)

JNDI (3)

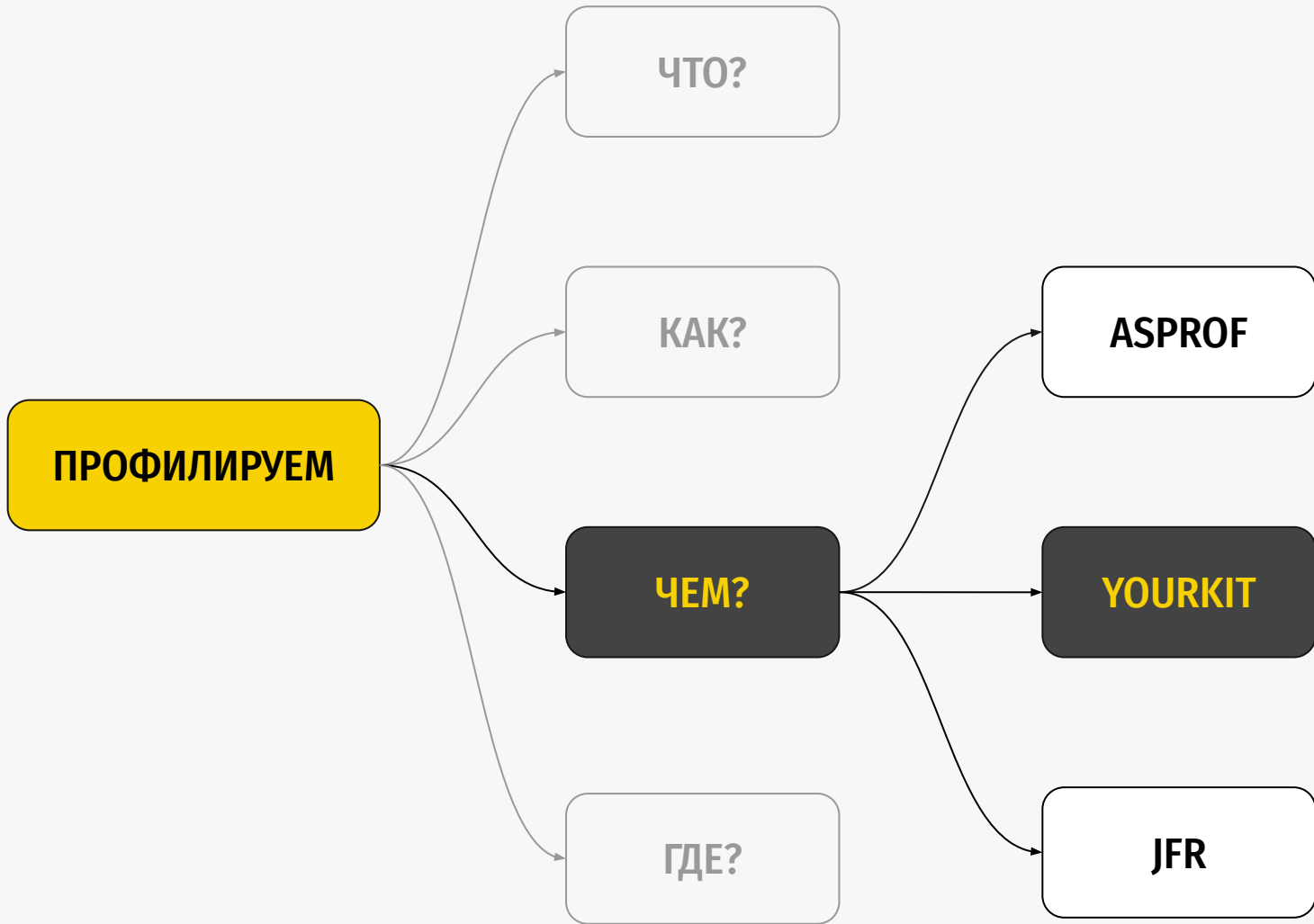


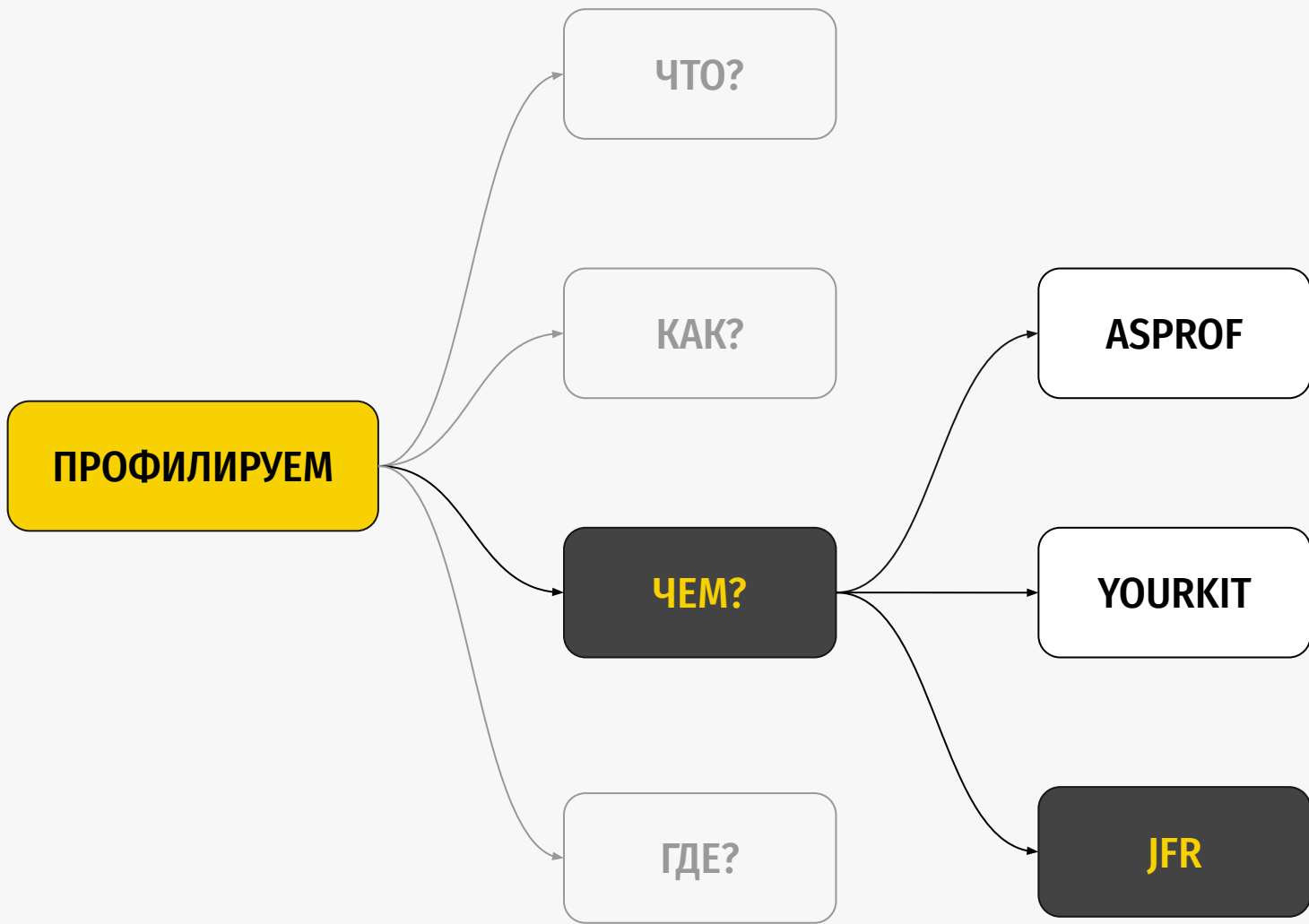


Н.В. ДЕМИДОВ

**ИСКУССТВО
ЖИТЬ
НА**

1500р.






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



Filter:

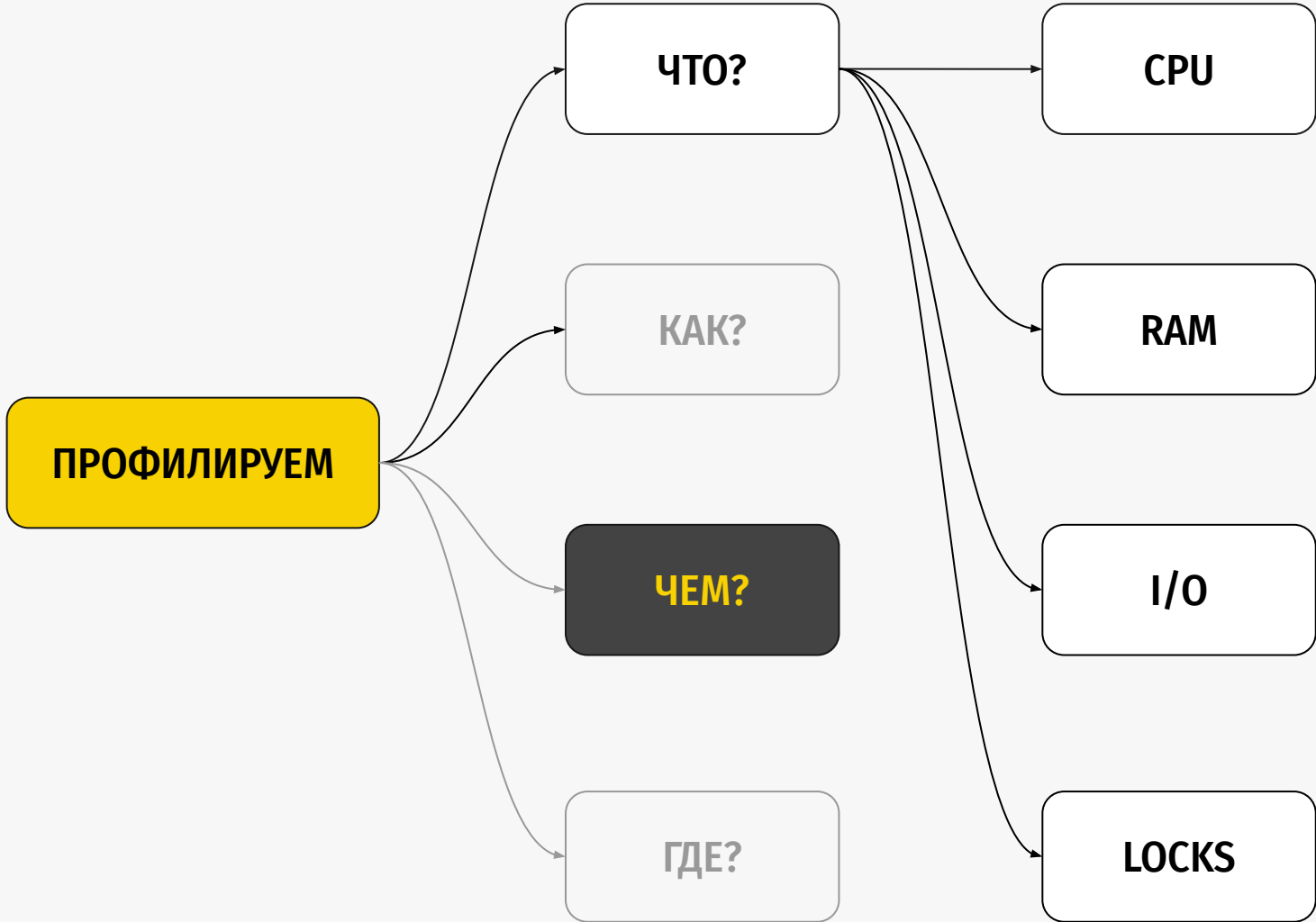
- Java Application
 - File Force
 - File Read**
 - File Write
- Operating System
 - File System
 - Container IO Usage

Enabled

Stack Trace

Threshold

JFR СОБЫТИЯ ДЛЯ РЕСУРСОВ



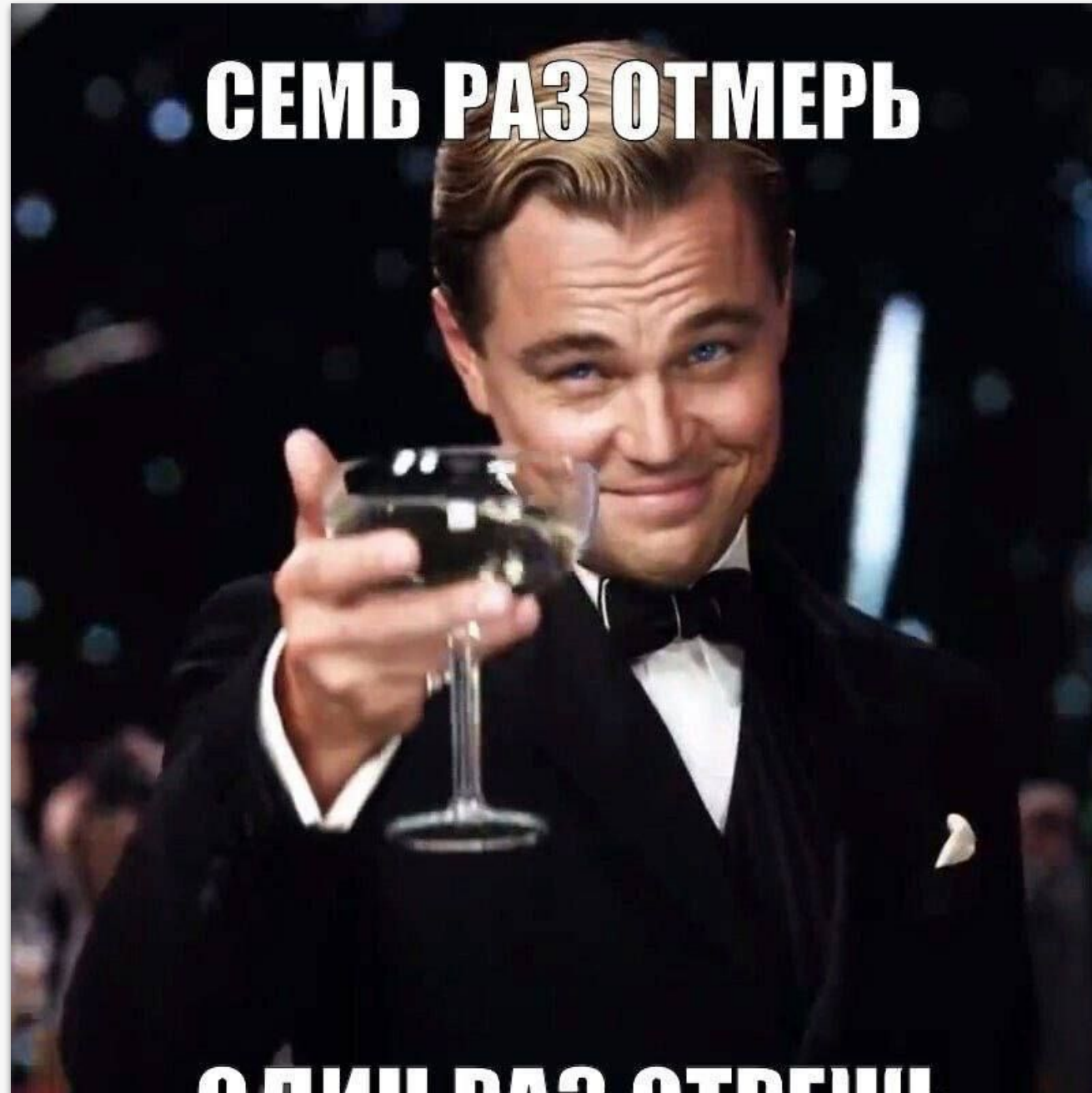
- `jdk.ExecutionSample`
- `jdk.NativeMethodSample`

- `jdk.ObjectAllocationSample`
- `jdk.ObjectAllocationInNewTLAB`
- `jdk.ObjectAllocationOutsideTLAB`


- `jdk.SocketWrite (Read)`
- `jdk.FileWrite (Read)`

- `jdk.JavaMonitorWait (Enter)`
- `jdk.ThreadPark (Sleep)`

СЕМЬ РАЗ ОТМЕРЬ



ОДИН РАЗ ОТРЕШИ



КАК УВИДЕТЬ СТЕКТРЕЙС ОБРАЩЕНИЯ К РЕСУРСУ В JMS?

File I/O

Focus: <No Selection> **1** Aspect: <No Selection> 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)

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

2 **3** **4**

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



i Подробности:
[Alexey Ragozin – Hunting down
code hotspots with
JDK Flight Recorder](#)

JVM Browser Outline

Automated Analysis Results

- Java Application
 - Threads
 - Memory
 - Lock Instances
 - File I/O
 - Socket I/O
 - Method Profiling**
 - Exceptions
 - Thread Dumps
- JVM Internals
 - Garbage Collections
 - GC Configuration
 - GC Summary
 - Compilations
 - Class Loading
 - VM Operations
 - TLAB Allocations
- Environment
 - Event Browser

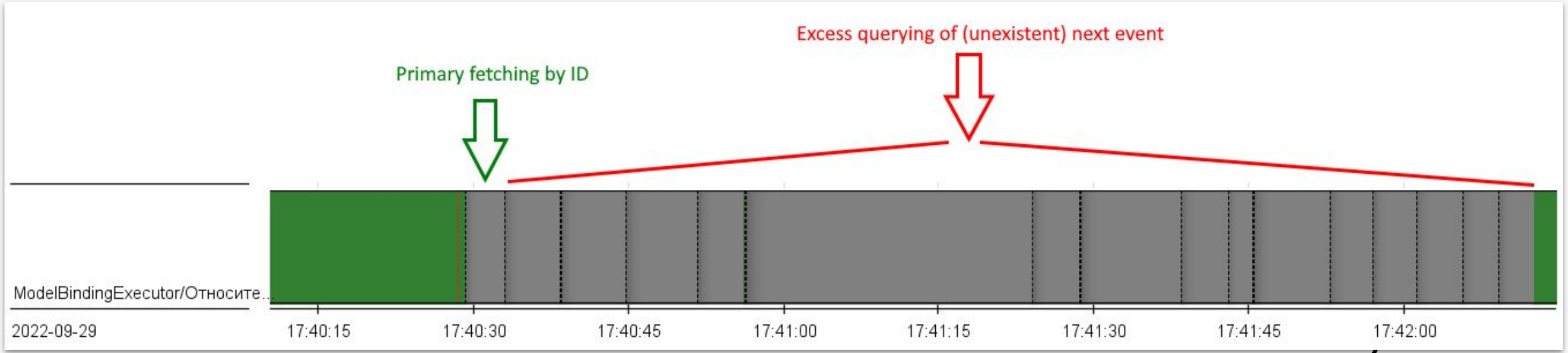
Результаты
профилирования
sampling'ом
(CPU time)

WALL-CLOCK: ПРИМЕР ИЗ ЖИЗНИ

Projects /  AggreGate /  Add parent /  AGG-14311

Eliminate excess load on Cassandra upon fetching events by ID

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



YourKit

Events:

JSP/Servlet (2,303)
▶ SQL (4)
▶ Socket (163)
▶ File (348)
▶ Process (12)
▶ Thread (1,804)
Thread park (0)
Class Loading (16,228)
Message (8)
▶ Directory Stream (0)
JNDI (3)

Automated Analysis Results

- Java Application
 - Threads
 - Memory
 - Lock Instances
 - File I/O
 - Socket I/O
 - Method Profiling
 - Exceptions
 - Thread Dumps
- JVM Internals
 - Garbage Collections
 - GC Configuration
 - GC Summary
 - Compilations
 - Class Loading
 - VM Operations
 - TLAB Allocations
- Environment
 - Event Browser

Java
Mission
Control

YourKit

Events:

JSP/Servlet (2,303)
▶ SQL (4)
▶ Socket (163)
▶ File (348)
▶ Process (12)
▶ Thread (1,804)
Thread park (0)
Class Loading (16,228)
Message (8)
▶ Directory Stream (0)
JNDI (3)

Automated Analysis Results

- Java Application
 - Threads
 - Memory
 - Lock Instances
 - File I/O
 - Socket I/O
 - Method Profiling
 - Exceptions
 - Thread Dumps
- JVM Internals
 - Garbage Collections
 - GC Configuration
 - GC Summary
 - Compilations
 - Class Loading
 - VM Operations
 - TLAB Allocations
 - Environment
 - Event Browser

Java Mission Control

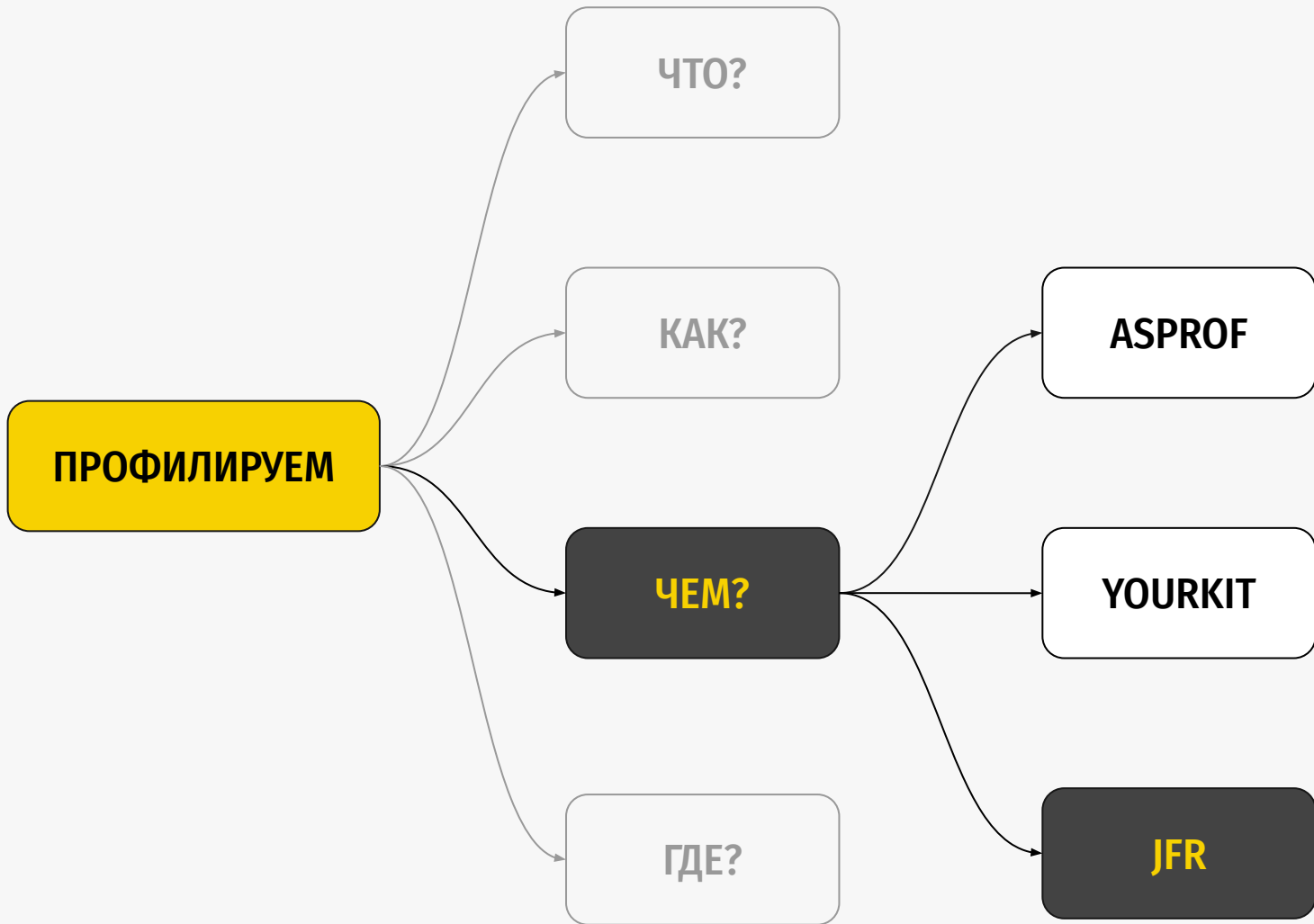
JFR: КАК ДОБАВИТЬ СВОЕ СОБЫТИЕ

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



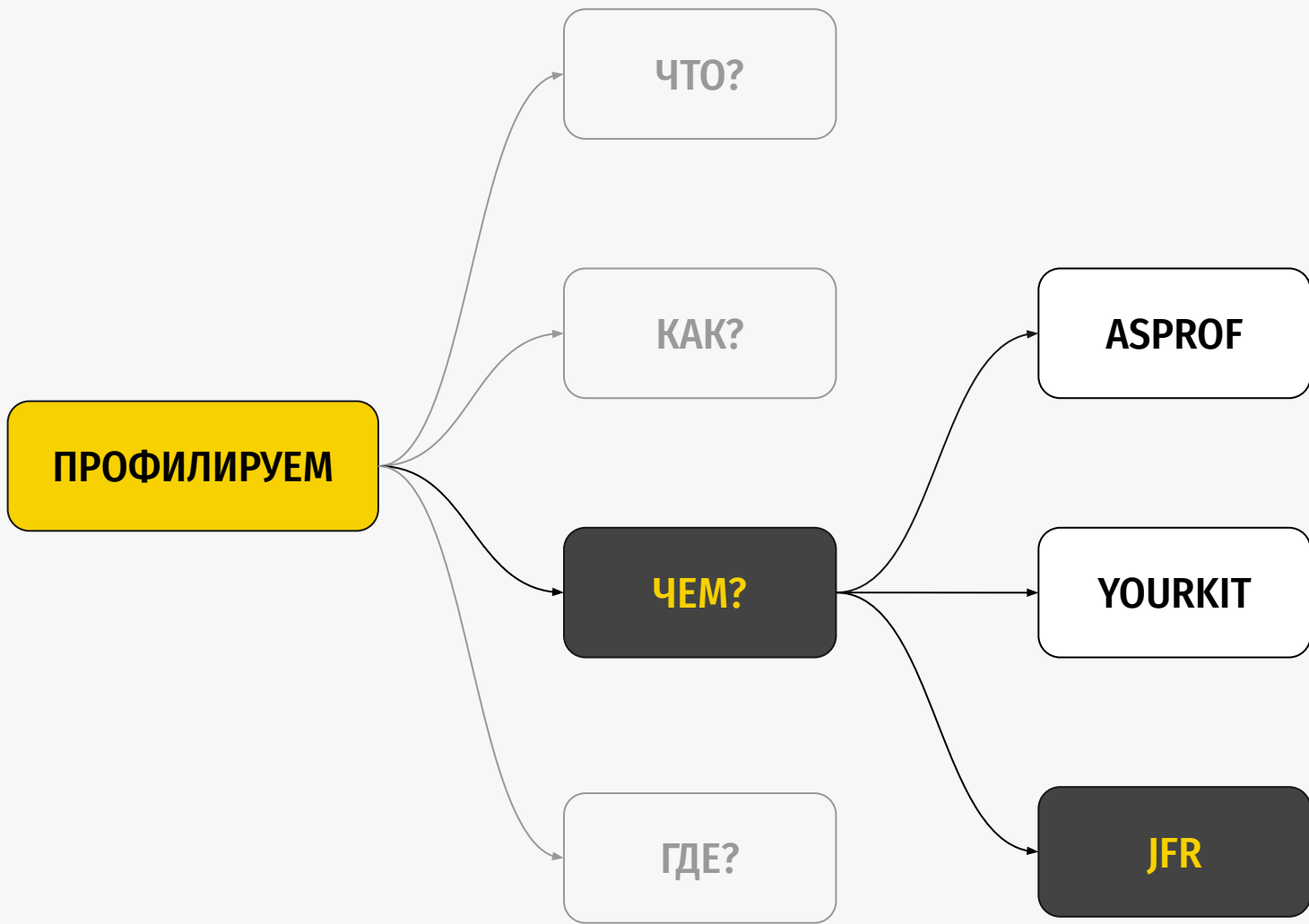
Примеры и объяснения в статье:

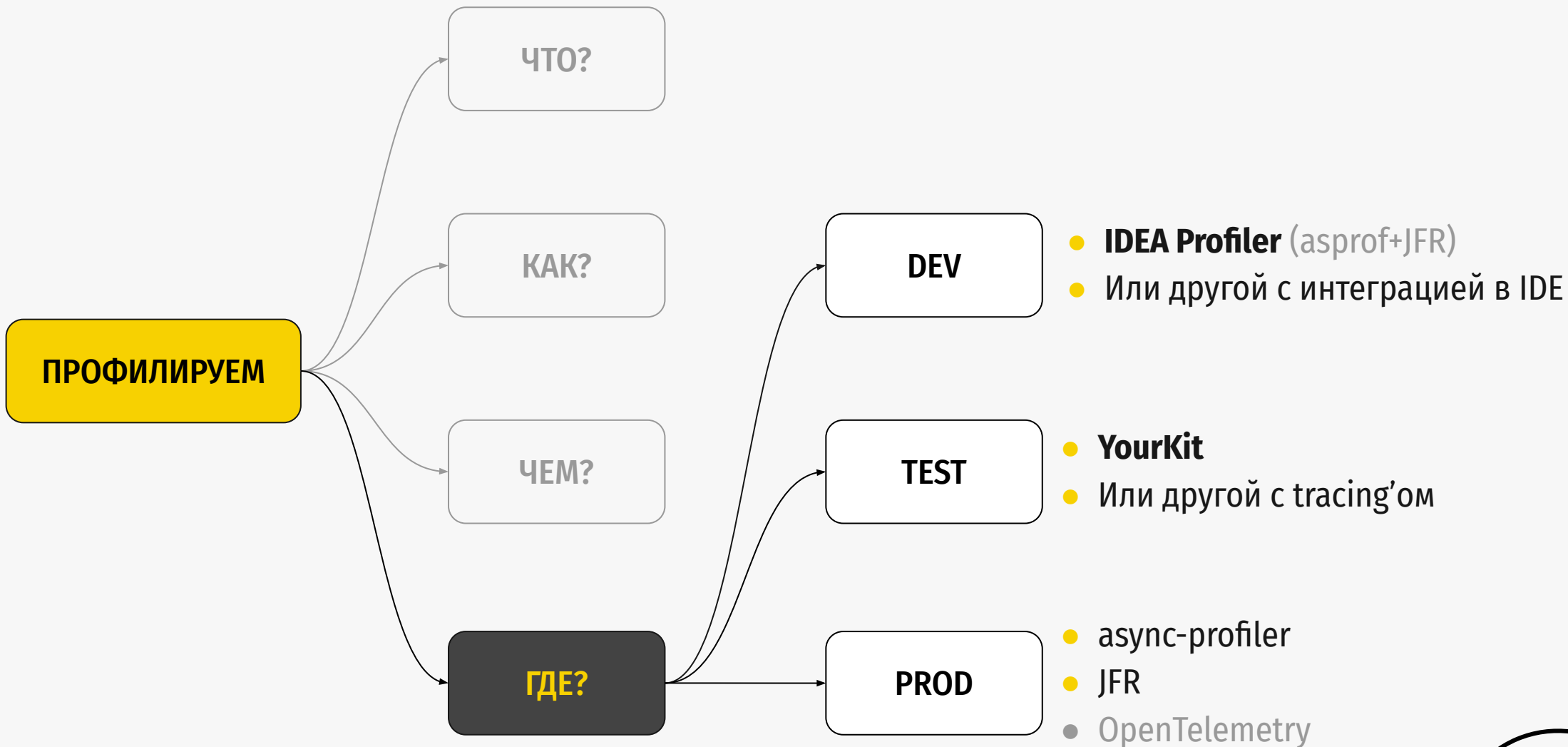
[Monitoring REST APIs with Custom JDK Flight Recorder Events](#)



СВОДКА

Фича \ Инструмент	ASPROF	YOURKIT	JFR
Sampling/Tracing/Events	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)

БАРАНИНА ПЛОВ ХАЧАПАУРИ

с БАРАНИНОЙ
с ИНДЕЙКОЙ
с СЫРОМ



arm X food


ПОПРОБУЙ МАМИНЫХ
ЛЮЛЕЙ!*

*Люля кебаб от 99 рублей





ВЫВОДЫ



СОВЕРШЕННОГО ПРОФАЙЛЕРА ПОКА **НЕТ**

И не нужен...

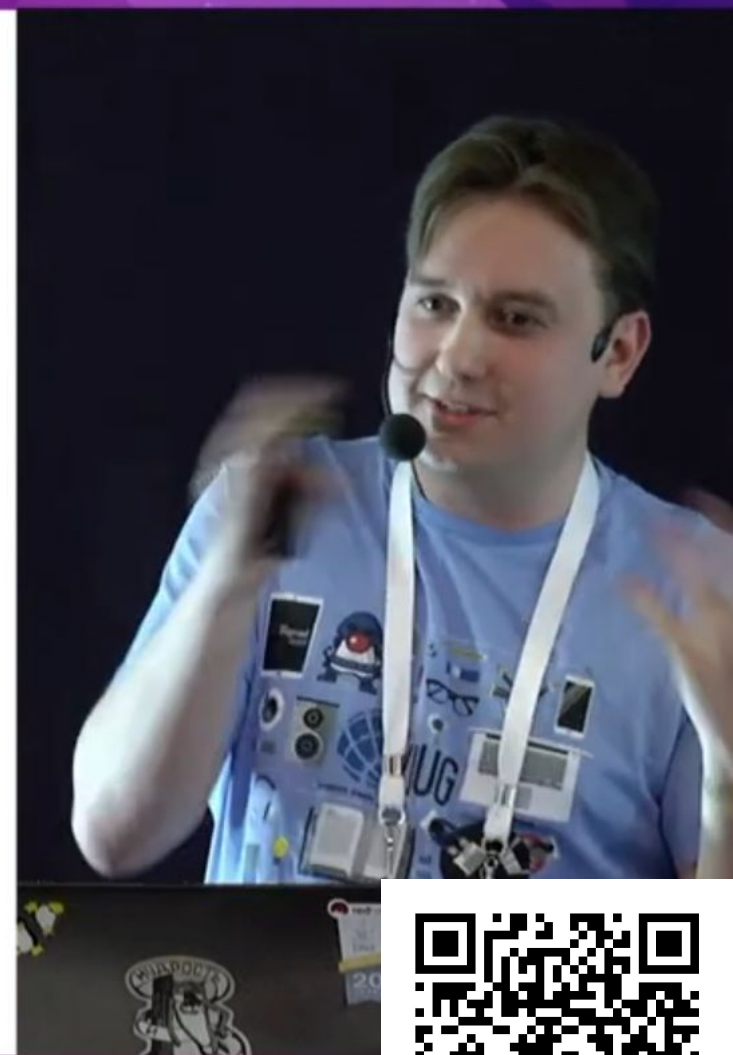
Зелёная зона: подитог

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 работы¹

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



НА ЗАМЕТКУ

- Освойте какой-нибудь профайлер сегодня (пока не пригорело)
 - Не знаете, какой выбрать – берите async-profiler
- Начиная 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

- [Saferpoint – и пусть весь мир подождёт](#)
- [Профайлер в каждый дом](#)

СПАСИБО!

Вопросы?

Владимир Плизга
Tibbo Systems



[@stopshelf](#)



[toparvion.pro](#)



[Слайды](#)