Непрерывное профилирование в Облаке на десятки тысяч хостов — от теории к практике

Егор Леванков
Группа Core разработки внутреннего Облака VK — One Cloud



Обо мне

Егор Леванков

- Работаю в группе Core разработки внутреннего Облака VK — One Cloud
- Занимаюсь разными вещами — в том числе профилированием



Контекст

Разные языки:
Java/Go/
C++/...

36K
серверов

100K
сервисов

750K
подов

Про что доклад

Зачем?



Про что доклад

- Зачем?
- Инструменты



Про что доклад



- Зачем?
- Инструменты
- Под капотом

Про что доклад



- Зачем?
- Инструменты
- Под капотом
- **Облачная специфика**

Про что доклад



- Зачем?
- Инструменты
- Под капотом
- Облачная специфика
- Сравнение

Про что доклад



- Зачем?
- Инструменты
- Под капотом
- Облачная специфика
- Сравнение
- Доработки

Про что доклад



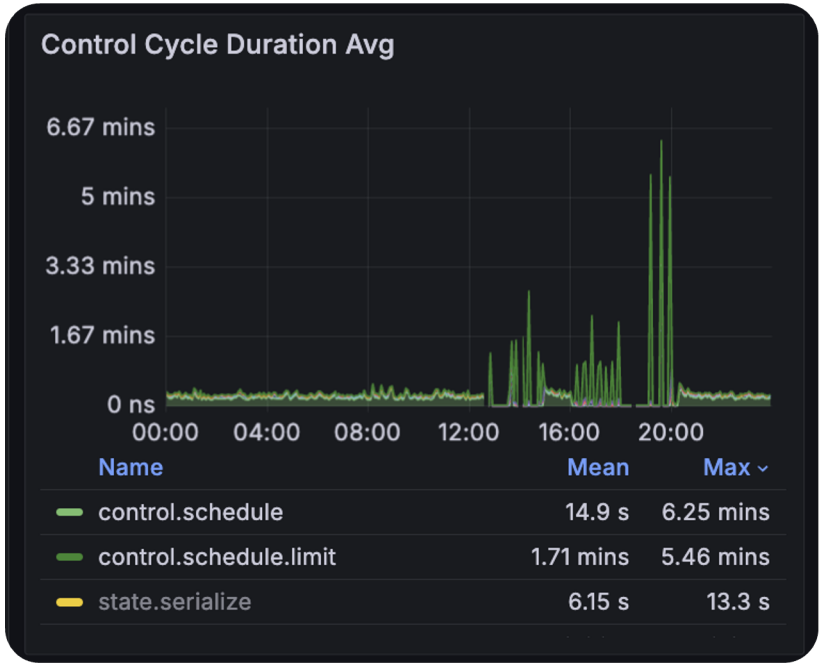
- Зачем?
- Инструменты
- Под капотом
- Облачная специфика
- Сравнение
- Доработки
- **Стоимость**

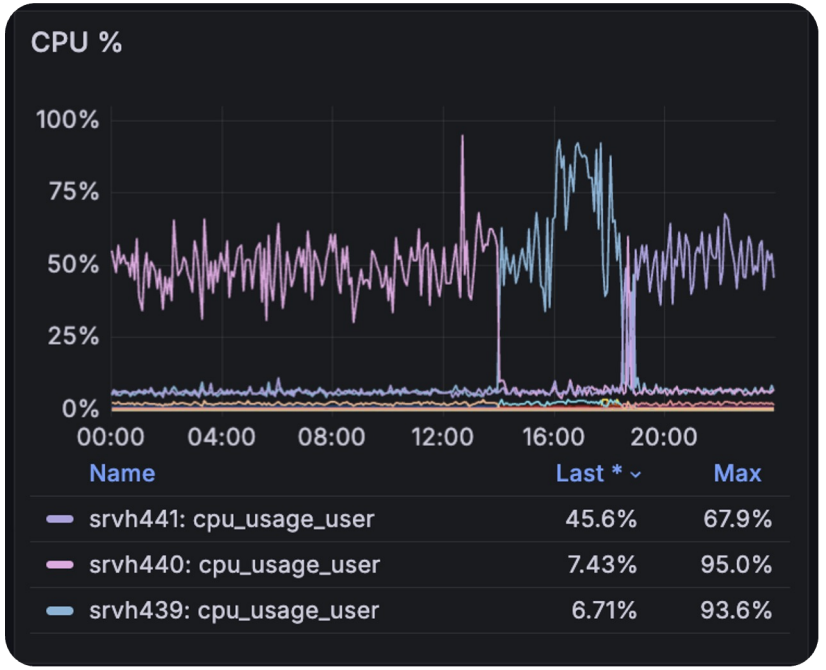
Поехали!
Начнём с примера

Пример-инцидент

Проблема

- «Мастер» — отвечает за размещение в облаке
- Стал очень медленно заселять, CPU в полке
- Reconciliation loop стал бежать >300 с вместо 15 с

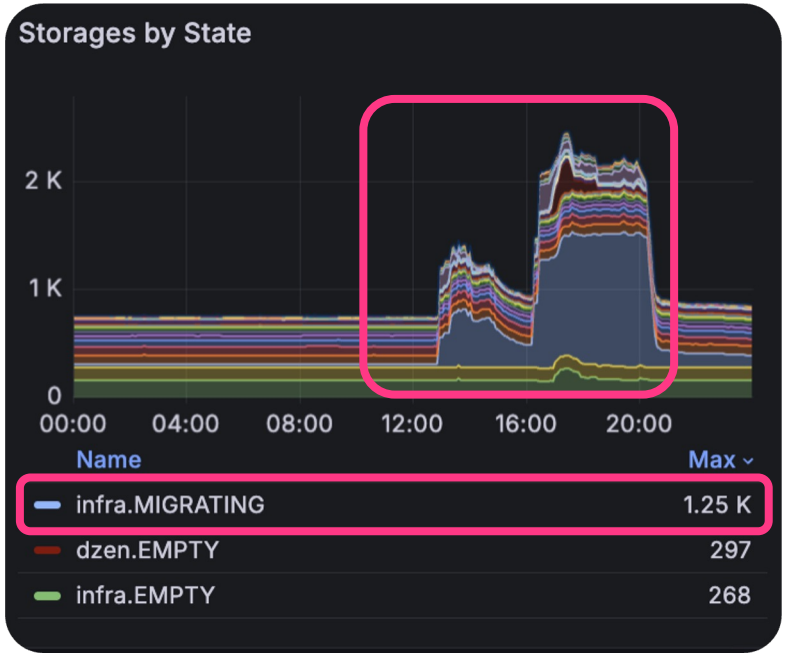


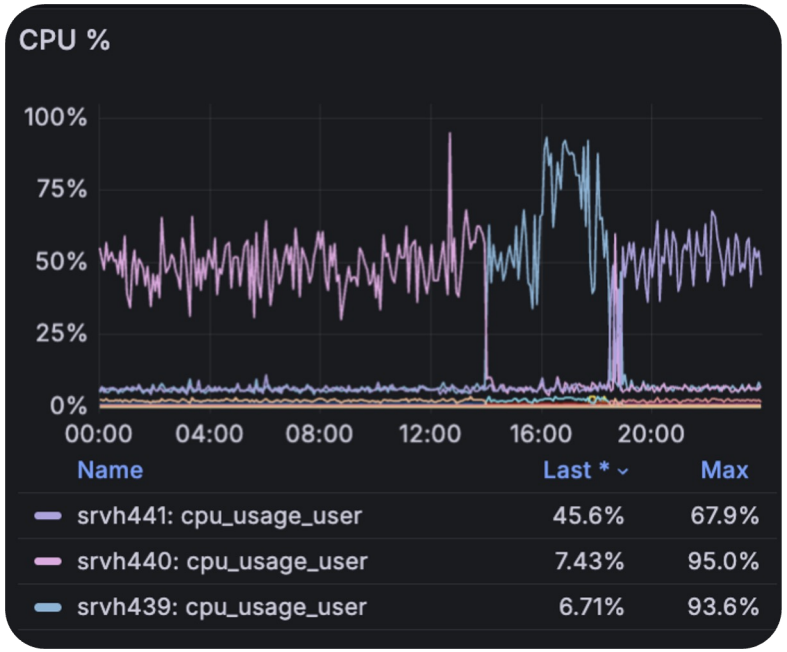


Пример-инцидент

Что видим

- Бьется с сильным ростом числа миграций, которые стриггерила автоматика
- Поотменяли миграции
- Все стало хорошо



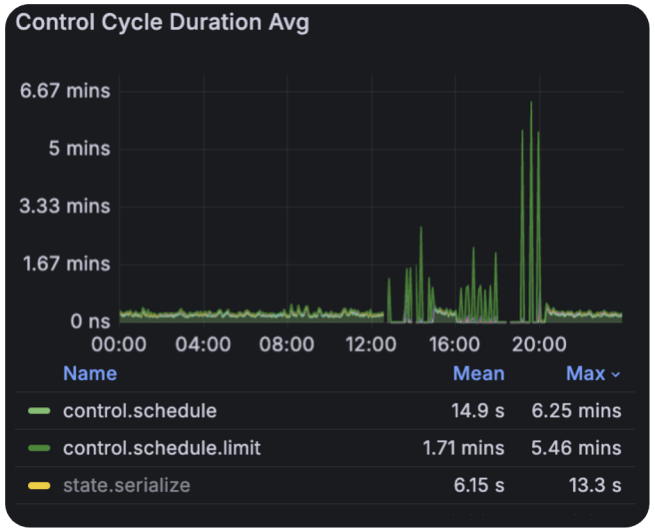


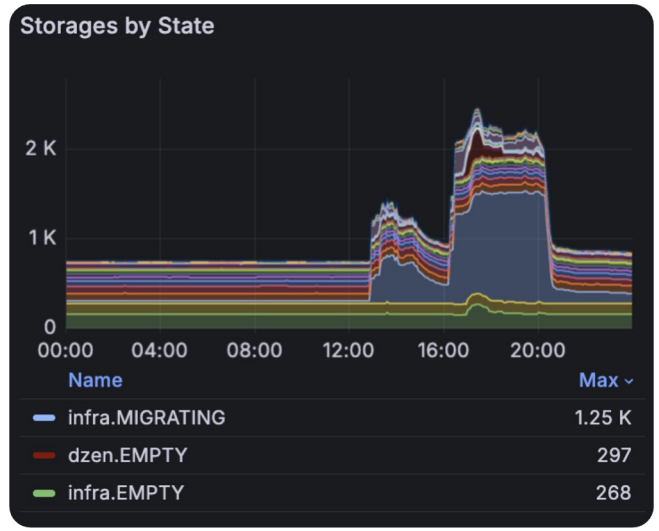
Что произошло?

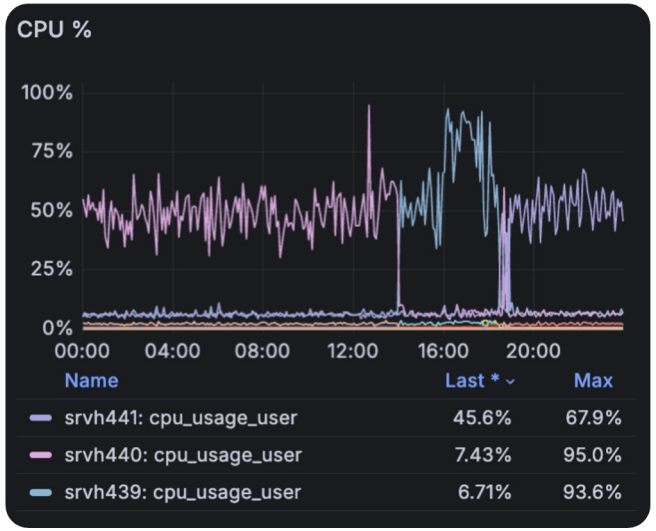
Давайте исследовать

Метрики

- Видим какие-то проблемы
- Не видим,
  где расходуется CPU
- Не видим root-cause







Давайте исследовать

Логи

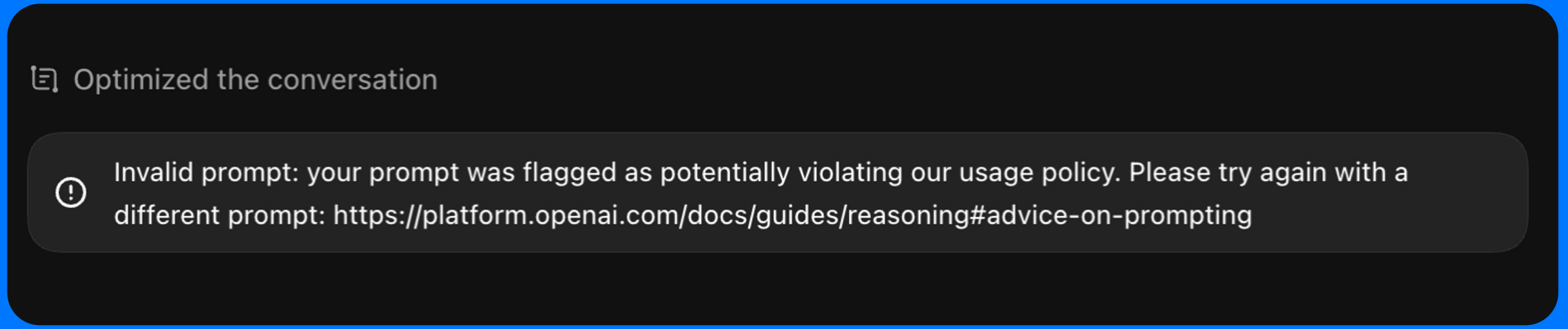
- Куча логов
- Читать долго
- Проблем сходу не видно
- Можно вчитаться,
  но не факт, что будут
  нужные логи

```
192.168.1.10 - - [08/Sep/2025:13:10:41 +0530] "PUT /api/v1/data HTTP/1.1" 302 12928 "http://
facebook.com" "Mozilla/5.0 (iPhone; CPU iPhone OS 17_0 like Mac OS X)"
203.0.113.45 - - [08/Sep/2025:14:02:34 +0530] "PUT /favicon.ico HTTP/1.1" 502 7507 "http://
example.com" "Mozilla/5.0 (iPhone; CPU iPhone OS 17_0 like Mac OS X)"
192.168.1.15 - - [08/Sep/2025:14:07:35 +0530] "PUT /api/v1/data HTTP/1.1" 304 11669 "http://
google.com/search?q=example" "PostmanRuntime/7.32.3"
[Mon Sep 08 13:04:18.000000 2025] [proxy:error] (111)Connection refused: AH00957: HTTP:
attempt to connect to 127.0.0.1:8080 (backend-server) failed
[Mon Sep 08 13:20:22.000000 2025] [auth_basic:error] AH01617: user admin: authentication
failure for "/wp-login.php": Password Mismatch
192.168.1.10 - - [08/Sep/2025:14:05:30 +0530] "DELETE /wp-login.php HTTP/1.1" 404 14390
"http://example.com" "Mozilla/5.0 (iPhone; CPU iPhone OS 17_0 like Mac OS X)"
203.0.113.45 - - [08/Sep/2025:13:21:52 +0530] "PUT /admin/upload.php HTTP/1.1" 404 2640
"http://google.com/search?q=example" "curl/8.0.1"
198.51.100.55 - - [08/Sep/2025:13:25:45 +0530] "GET /admin/upload.php HTTP/1.1" 201 9019
"http://example.com" "Googlebot/2.1 (+http://www.google.com/bot.html)"
198.51.100.55 - - [08/Sep/2025:13:07:10 +0530] "GET /about.html HTTP/1.1" 200 12192 "http://
example.com" "Mozilla/5.0 (iPhone; CPU iPhone OS 17_0 like Mac OS X)"
198.51.100.23 - - [08/Sep/2025:12:50:17 +0530] "GET /api/v1/data HTTP/1.1" 201 5706 "-"
"Mozilla/5.0 (Linux; Android 13)"
203.0.113.100 - - [08/Sep/2025:13:56:43 +0530] "GET / HTTP/1.1" 302 12118 "http://google.com/
search?q=example" "Mozilla/5.0 (iPhone; CPU iPhone OS 17_0 like Mac OS X)"
203.0.113.100 - - [08/Sep/2025:12:33:53 +0530] "DELETE /about.html HTTP/1.1" 500 2271 "http://
example.com/admin" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)"
198.51.100.77 - - [08/Sep/2025:13:54:16 +0530] "POST /blog/article?id=1234 HTTP/1.1" 200 342
"http://example.com/admin" "PostmanRuntime/7.32.3"
```

Давайте исследовать

Логи

Интересно: в момент инцидента LLM сказала:
«Я нарушаю правила» — и отказалась анализировать логи





Кадр: «Я, робот» (2004)

Давайте исследовать

Tracing

- У нас его нет :)
- Показывает картину высокоуровнево — **здесь бы не помог**



Давайте исследовать

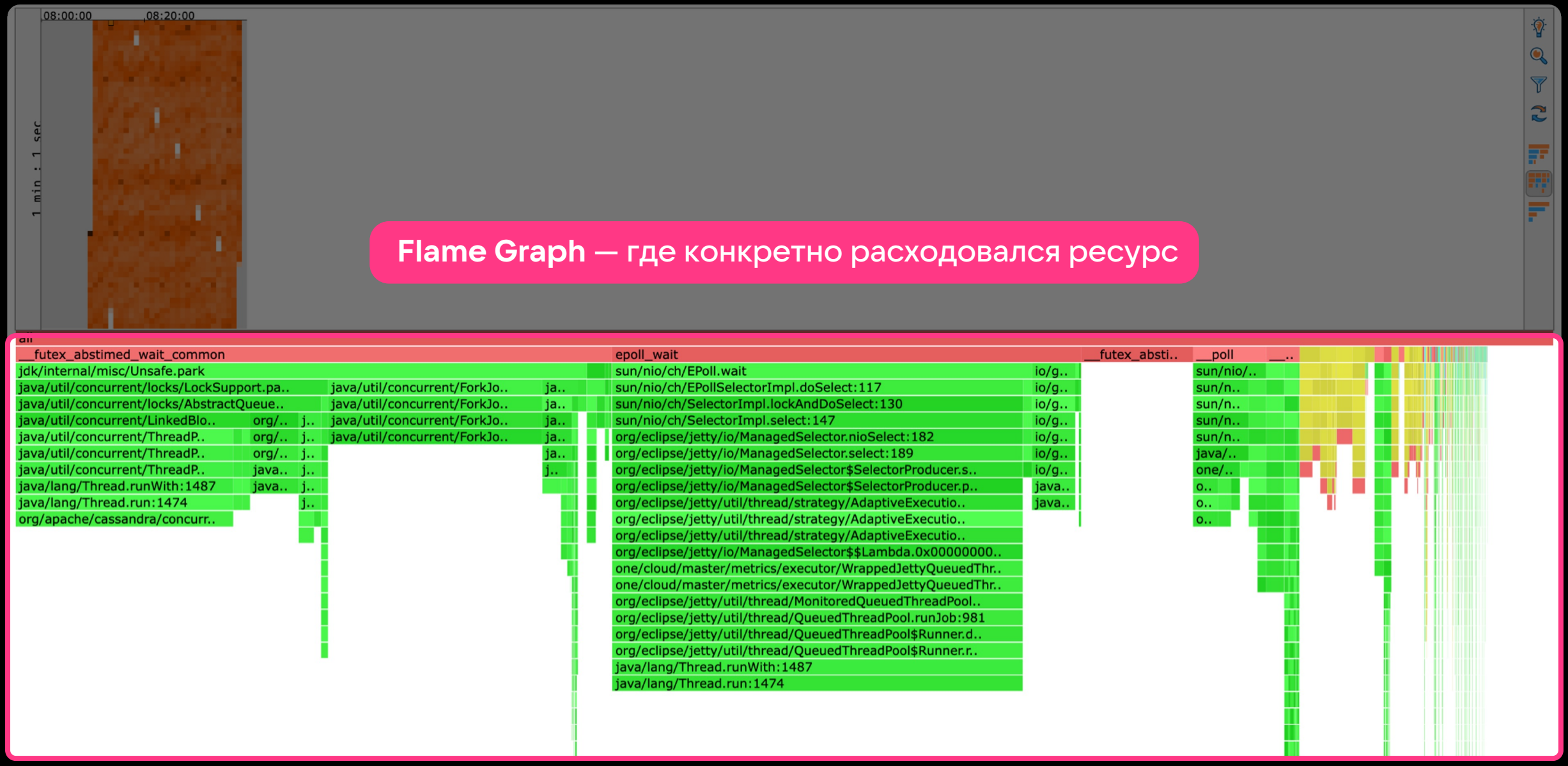
Так что нам надо?

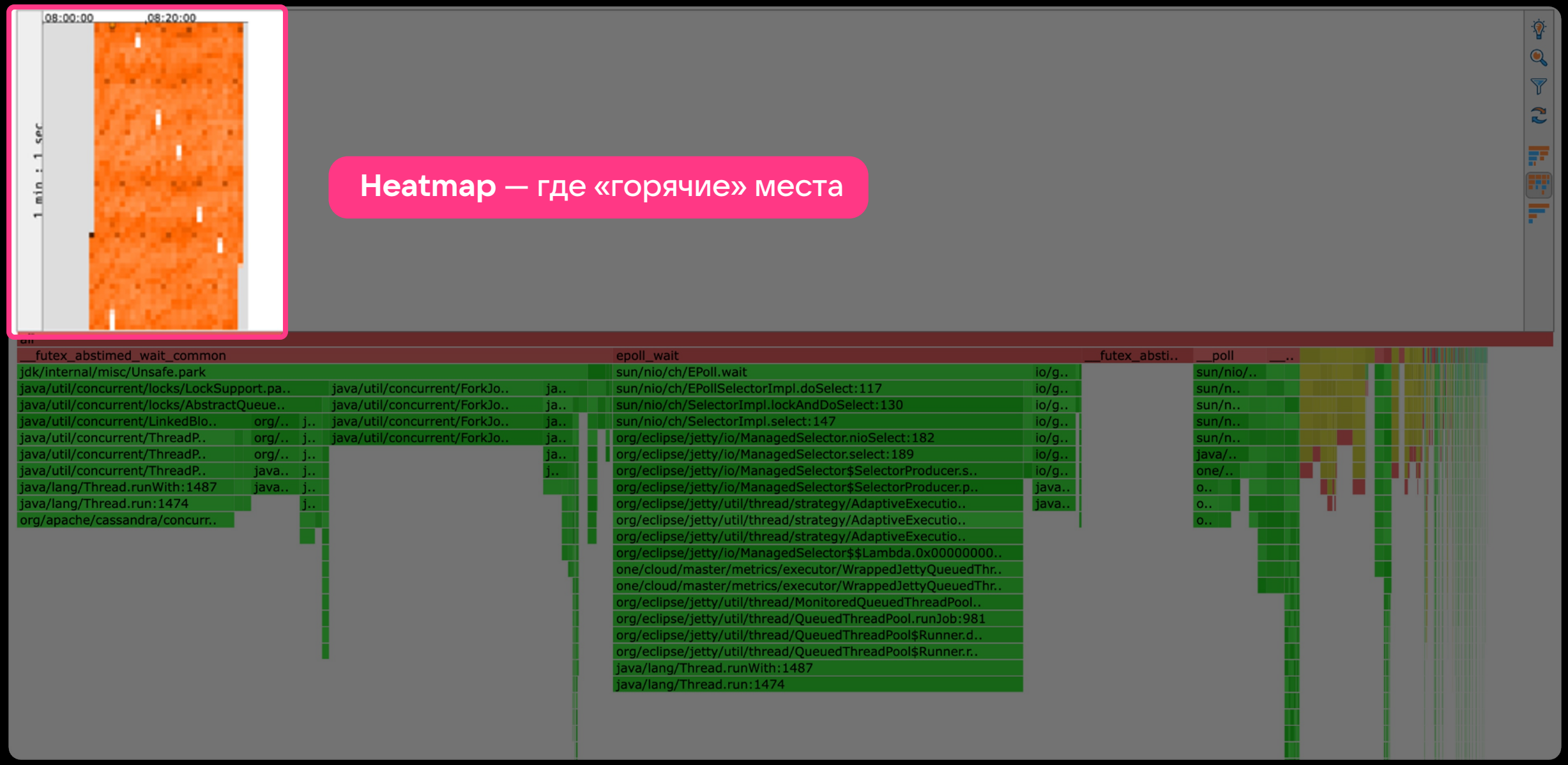
- **Исторические данные**
- **Охват всей системы**
  - Своё приложение
  - Native-библиотеки
  - И желательно kernel
- **Низкий оверхед**

Звучит как непрерывное профилирование!

Что хотим получить в результате?




08:00:00
08:20:00
1 min : 1 sec
Flame Graph — где конкретно расходовался ресурс
__futex_abstimed_wait_common
epoll_wait
__futex_absti..
__poll
jdk/internal/misc/Unsafe.park
java/util/concurrent/locks/LockSupport.pa..
java/util/concurrent/locks/AbstractQueue..
java/util/concurrent/LinkedBlo..
java/util/concurrent/ThreadP..
java/util/concurrent/ThreadP..
java/util/concurrent/ThreadP..
java/lang/Thread.runWith:1487
java/lang/Thread.run:1474
org/apache/cassandra/concurr..
java/util/concurrent/ForkJo..
sun/nio/ch/EPoll.wait
sun/nio/ch/EPollSelectorImpl.doSelect:117
sun/nio/ch/SelectorImpl.lockAndDoSelect:130
sun/nio/ch/SelectorImpl.select:147
org/eclipse/jetty/io/ManagedSelector.nioSelect:182
org/eclipse/jetty/io/ManagedSelector.select:189
org/eclipse/jetty/io/ManagedSelector$SelectorProducer.s..
org/eclipse/jetty/io/ManagedSelector$SelectorProducer.p..
org/eclipse/jetty/util/thread/strategy/AdaptiveExecutio..
org/eclipse/jetty/io/ManagedSelector$$Lambda.0x00000000..
one/cloud/master/metrics/executor/WrappedJettyQueuedThr..
org/eclipse/jetty/util/thread/MonitoredQueuedThreadPool..
org/eclipse/jetty/util/thread/QueuedThreadPool.runJob:981
org/eclipse/jetty/util/thread/QueuedThreadPool$Runner.d..
org/eclipse/jetty/util/thread/QueuedThreadPool$Runner.r..
java/lang/Thread.runWith:1487
java/lang/Thread.run:1474
sun/nio/..
sun/n..
java/..
one/..
o..



Heatmap — где «горячие» места

Непрерывное профилирование

Ограничения

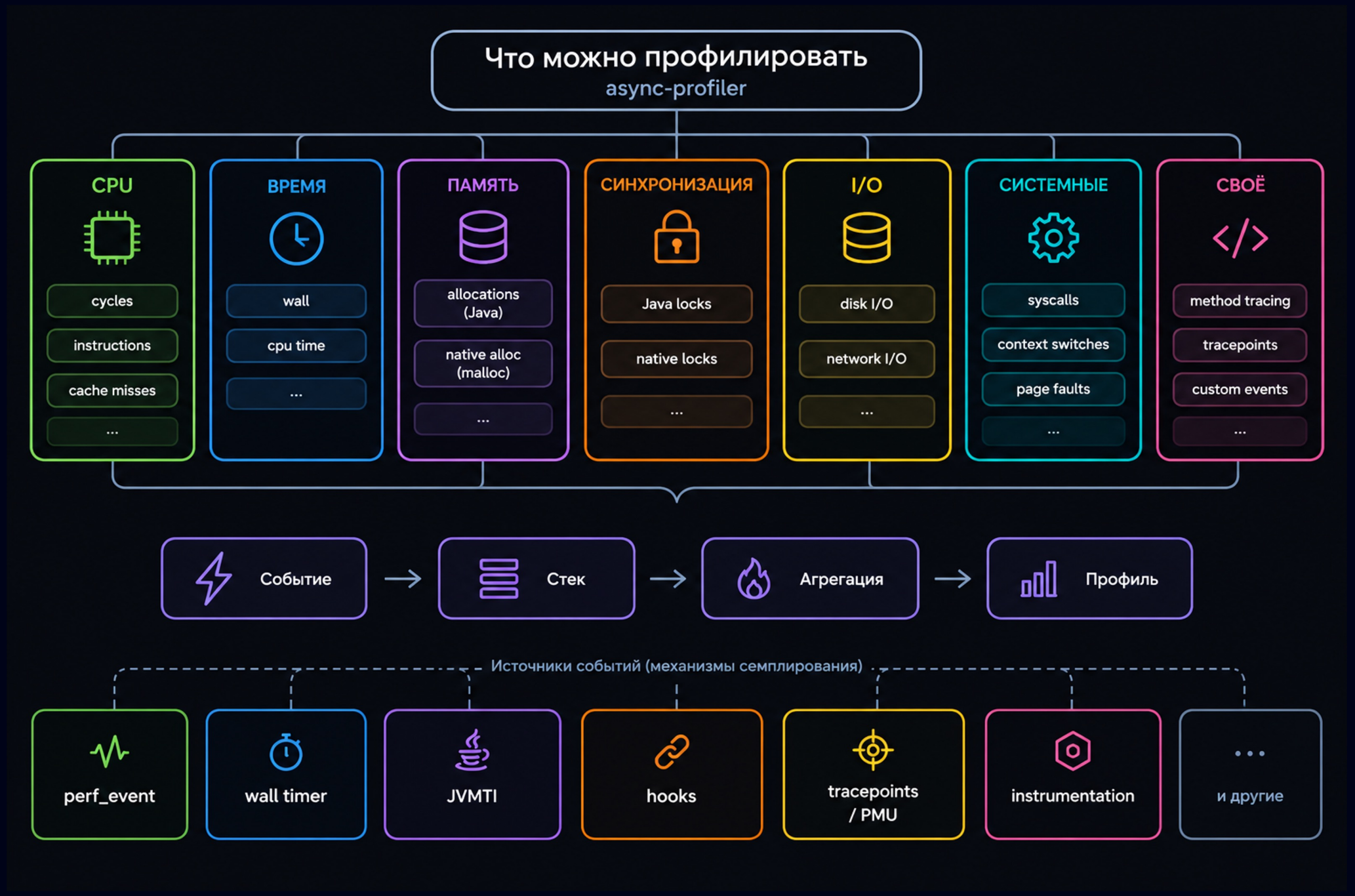
- Почти всегда это семплирование, а не полная трасса исполнения
- Короткое редкое событие может не попасть в выборку
- Частота семплирования влияет на точность, оверхед и объём хранения



Непрерывное профилирование



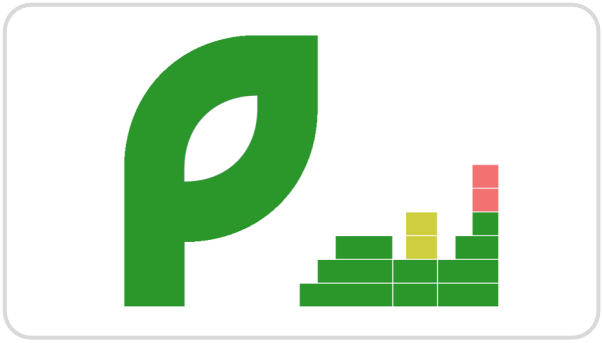
Какие ресурсы и события можно профилировать?


Что можно профилировать
async-profiler
CPU
cycles
instructions
cache misses
...
ВРЕМЯ
wall
cpu time
...
ПАМЯТЬ
allocations (Java)
native alloc (malloc)
...
СИНХРОНИЗАЦИЯ
Java locks
native locks
...
I/O
disk I/O
network I/O
...
СИСТЕМНЫЕ
syscalls
context switches
page faults
...
СВОЁ
method tracing
tracepoints
custom events
...
Событие
Стек
Агрегация
Профиль
Источники событий (механизмы семплирования)
perf_event
wall timer
JVMTI
hooks
tracepoints / PMU
instrumentation
и другие

Непрерывное профилирование

Чем мы пользуемся

Для JVM — надстройка над async-profiler (one-continuous-profiler)



Для натива (Go/C++/...) — Perforator



*для профилирования мастера Облака кастомный механизм на async-profiler-е

Историческая пауза

Историческая пауза

История async-profiler

- Написан Андреем Паньгиным
- Работал над HotSpot в Sun Microsystems
- Затем работал в Одноклассниках и писал One Cloud
- Здесь начал разработку async-profiler
- Сейчас продолжает его развитие в AWS

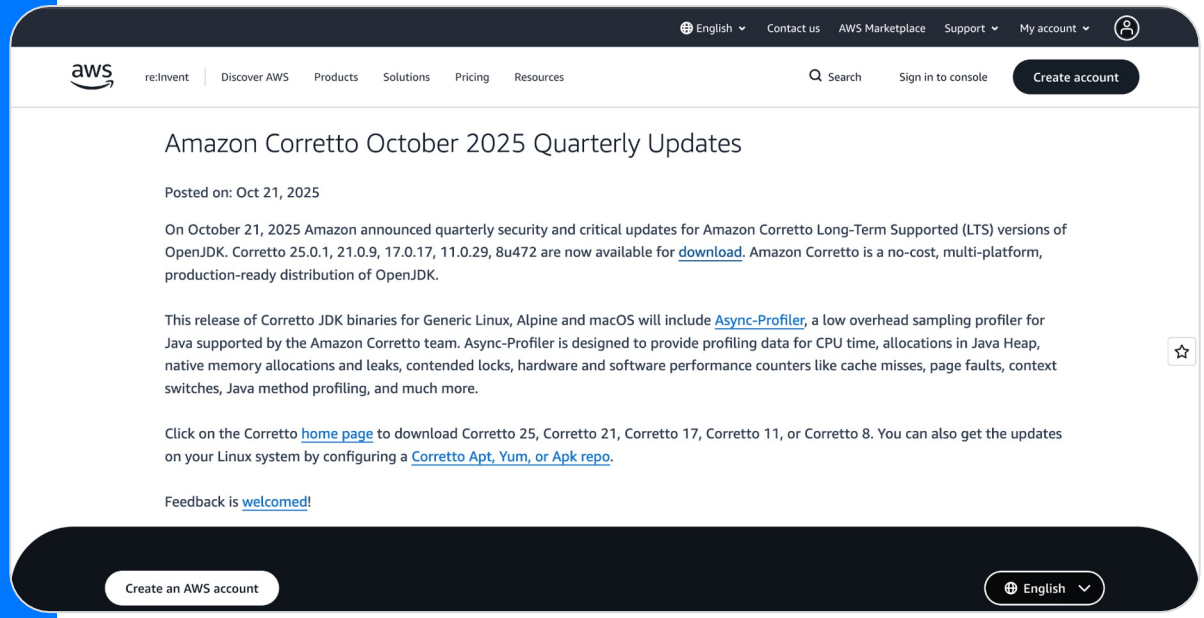


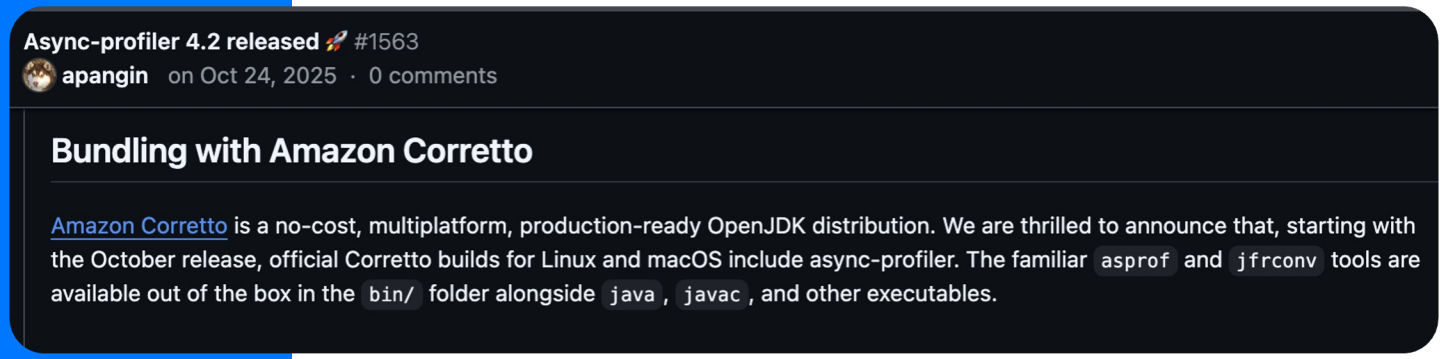
Андрей Паньгин

Историческая пауза

История async-profiler

- То есть он написан **буквально в нашей команде**!
- Стал **одним из самых известных Java-профилировщиков** и встроен в IntelliJ IDEA и Amazon Corretto JDK





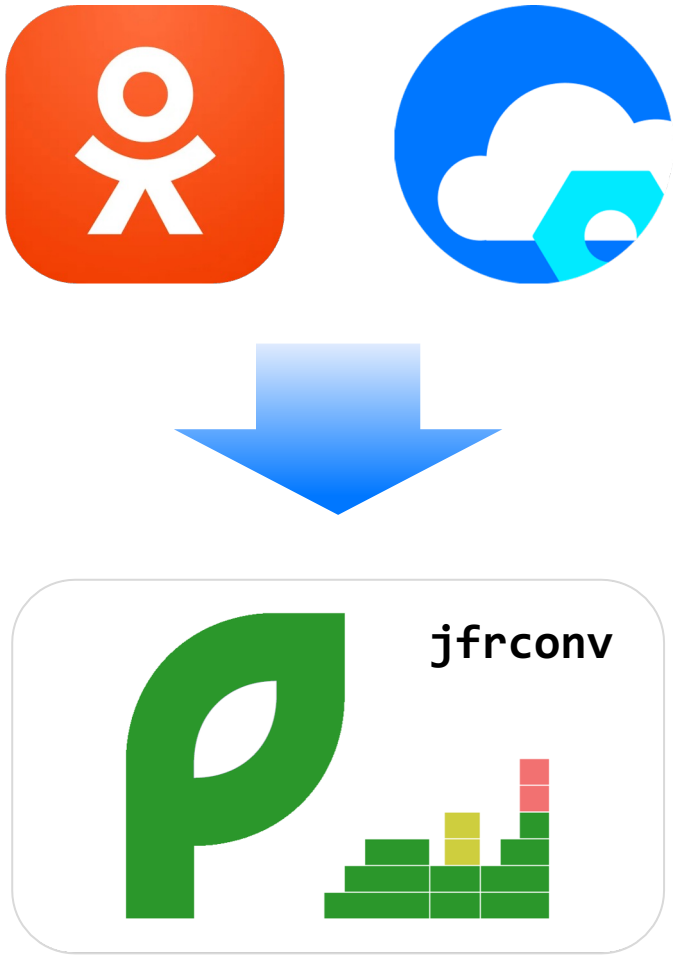
https://aws.amazon.com/about-aws/whats-new/2025/10/amazon-corretto-october-2025-quarterly-updates/

https://github.com/async-profiler/async-profiler/discussions/1563

Историческая пауза

Откуда ноги растут

- Изначально формирование HTML-отчёта было встроено во фронтенд нашей админки
- Потом перенесли в async-profiler jfrconv



Историческая пауза

Откуда ноги растут

EHOT / Cloud Core / one-cloud-web / Commits / **08b0f02a**

src/main/java/one/cloud/web/profiles/render/heatmap/MethodCache.java deleted 100644 → 0

```
-
-    public int index(long methodId, int location, byte type, boolean firstInStack) {
-        Method method;
-        if (methodId < nearCache.length) {
-            int mid = (int) methodId;
-            method = nearCache[mid];
-            if (method == null) {
-                method = createMethod(methodId, location, type, firstInStack);
-                nearCache[mid] = method;
-                return method.index = methodIndex.index(method);
-            }
-        } else {
-            // this should be extremely rare case
-            method = farMethods.get(methodId);
-            if (method == null) {
-                method = createMethod(methodId, location, type, firstInStack);
-                farMethods.put(methodId, method);
-                return method.index = methodIndex.index(method);
-            }
-        }
-
-        Method last = null;
-        Method prototype = null;
-        while (method != null) {
-            if (method.originalMethodId == methodId) {
-                if (method.location == location && method.type == type && method.start == firstInStack) {
-                    return method.index;
-                }
-                prototype = method;
-            }
-            last = method;
-            method = method.next;
-        }
-
```

github.com Heatmap converter (#944) · async-prof... Translate Ask Alice AI

Filter files...

- src
  - converter
    - Main.java
  - one
    - convert
      - FlameGraph.java
      - Index.java
      - JfrConverter.java
      - JfrToHeatmap.java
      - ResourceProcessor.java
    - heatmap
      - Heatmap.java
      - HtmlOut.java
      - HuffmanEncoder.java
      - LzNodeTree.java
      - Method.java
      - MethodCache.java
      - SampleList.java
      - StackStorage.java
      - SynonymTable.java
    - jfr
      - Dictionary.java
      - DictionaryInt.java
  - res
    - heatmap.html

Top Search within code

src/converter/one/heatmap/MethodCache.java

```
+    public int index(long methodId, int location, byte type, boolean firstInStack) {
+        Method method;
+        if (methodId < nearCache.length) {
+            int mid = (int) methodId;
+            method = nearCache[mid];
+            if (method == null) {
+                method = createMethod(methodId, location, type, firstInStack);
+                nearCache[mid] = method;
+                return method.index = methodIndex.index(method);
+            }
+        } else {
+            // this should be extremely rare case
+            method = farMethods.get(methodId);
+            if (method == null) {
+                method = createMethod(methodId, location, type, firstInStack);
+                farMethods.put(methodId, method);
+                return method.index = methodIndex.index(method);
+            }
+        }
+
+        Method last = null;
+        Method prototype = null;
+        while (method != null) {
+            if (method.originalMethodId == methodId) {
+                if (method.location == location && method.type == type && method.start == firstInStack) {
+                    return method.index;
+                }
+                prototype = method;
+            }
+            last = method;
+            method = method.next;
+        }
+
+        if (prototype != null) {
+            last.next = method = new Method(methodId, prototype.className, prototype.methodName, location, type, firstInStack);
+            return method.index = methodIndex.index(method);
+        }
```

Возвращаемся к сути

> Как же нам построить непрерывное профилирование в Облаке?

Здесь **2 ключевых части**:

непрерывное профилирование

в Облаке

Начнём с того, как профилировщики решают 1-ю задачу, чтобы реализовать 2-ю

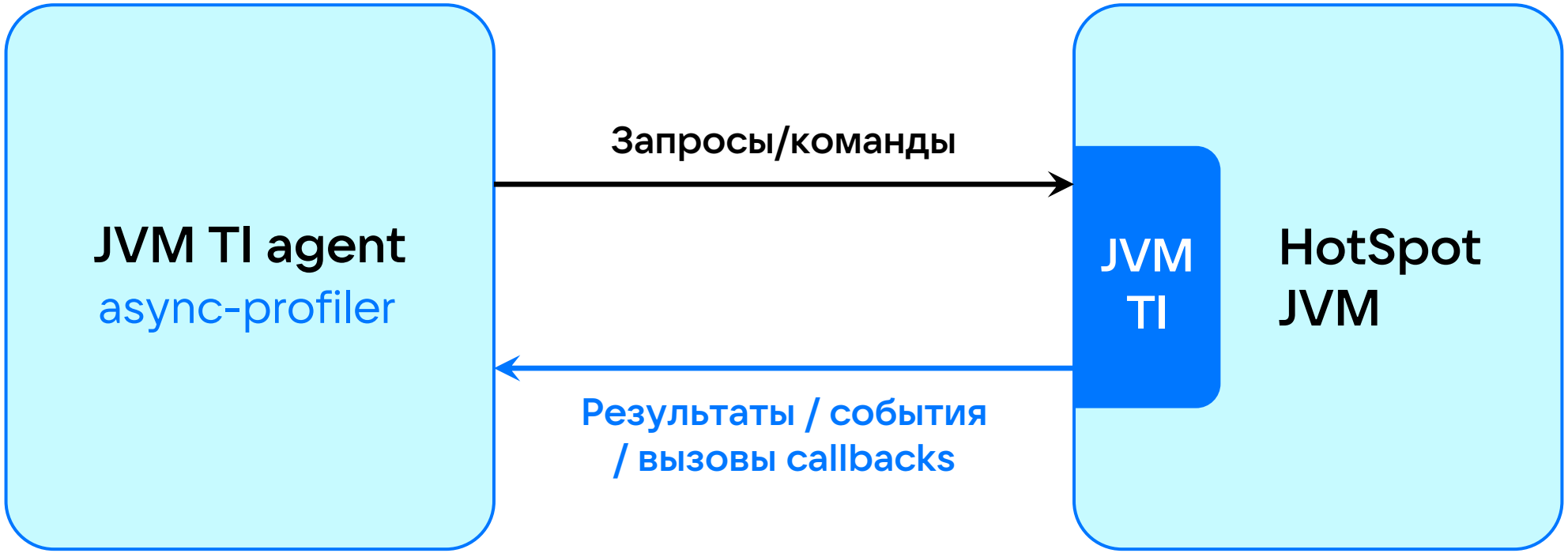
Как устроен async-profiler



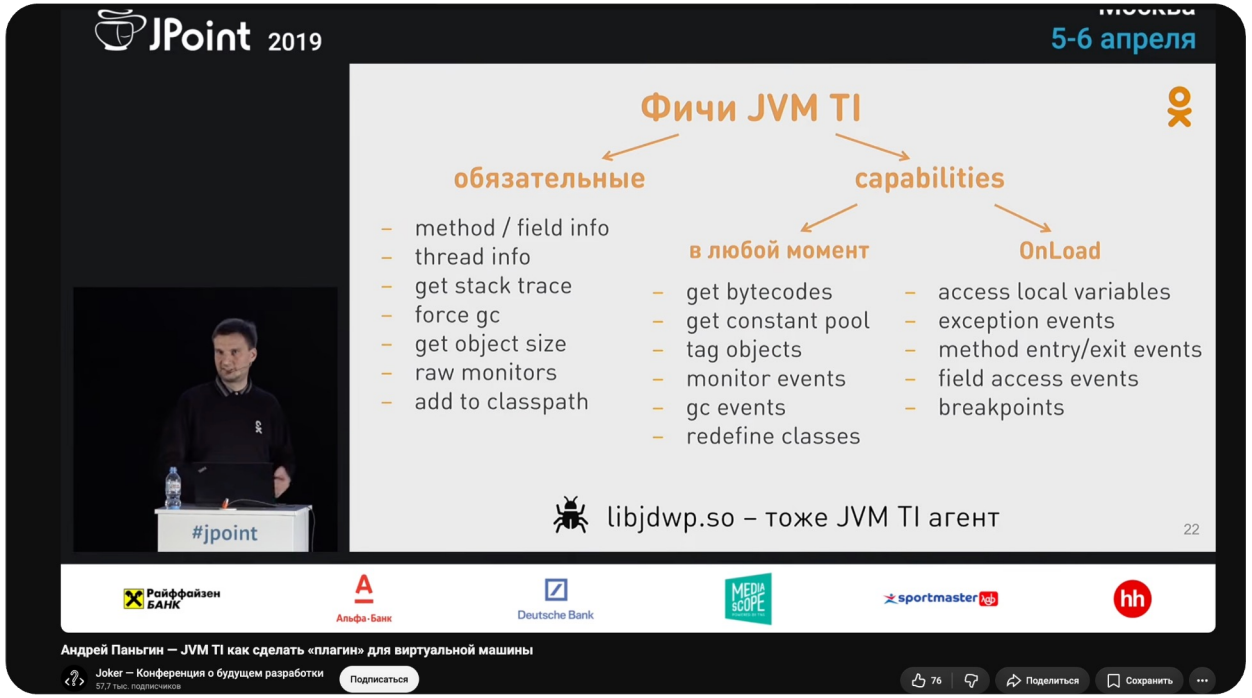
async-profiler

Что это по сути?

JVM TI — Java Virtual Machine Tool Interface



async-profiler



Доклад Андрея Паньгина
про написание JVM TI агентов

https://youtu.be/aiuKiE5-0g4?si=ur9l3uqJGQ2d6DZk

async-profiler

Как запускаем

Агент при старте

Загружаем в JVM
при её запуске

```
-agentpath:
libasyncProfiler.so
```

continuous-режим возьмем и вперед?

- Не все так просто из-за «облачной специфики»
- Давайте поймем, как оно работает внутри, чтобы понять, в чем проблема

async-profiler



Путь от CPU до Flame Graph-а на примере CPU cycles

async-profiler

Путь от CPU до Flame Graph-a

1

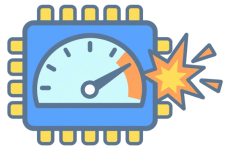


PMU обнаруживает overflow счётчика

async-profiler

Путь от CPU до Flame Graph-а

1



perf_events 2

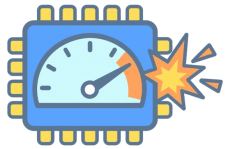


perf_events отправляет сигнал потоку

async-profiler

Путь от CPU до Flame Graph-a

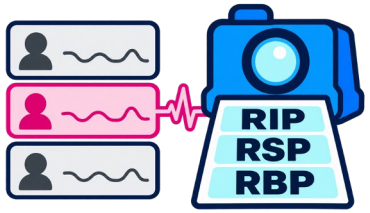
1



perf_events 2



Сигнал потоку 3

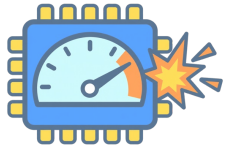


Обработчик сигнала получает регистры потока

async-profiler

Путь от CPU до Flame Graph-a

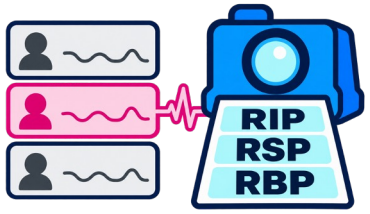
1



perf_events 2



Сигнал потоку 3

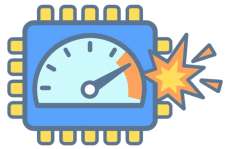


Код приложения на паузе: работает обработчик сигнала

async-profiler

Путь от CPU до Flame Graph-a

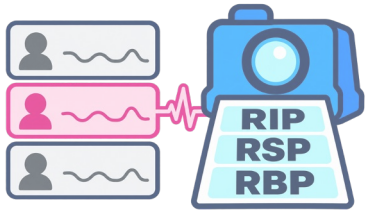
1



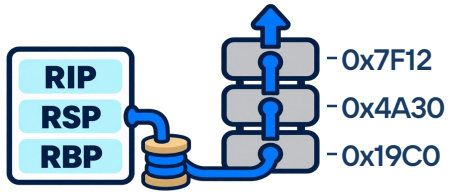
perf_events 2



Сигнал потоку 3



Stack Unwinding 4

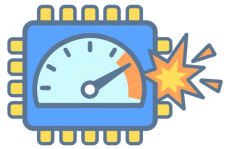


Stack Unwinding: восстанавливаем цепочку вызовов

async-profiler

Путь от CPU до Flame Graph-a

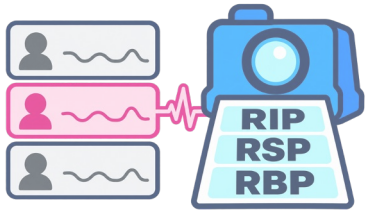
1



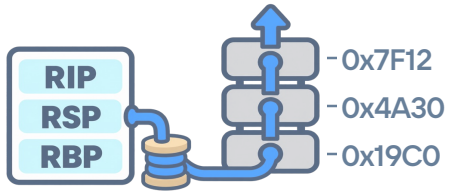
perf_events 2



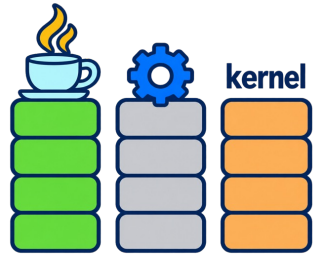
Сигнал потоку 3



Stack Unwinding 4



Java + native + kernel 5



Добавляем готовый kernel-стек к Java/native

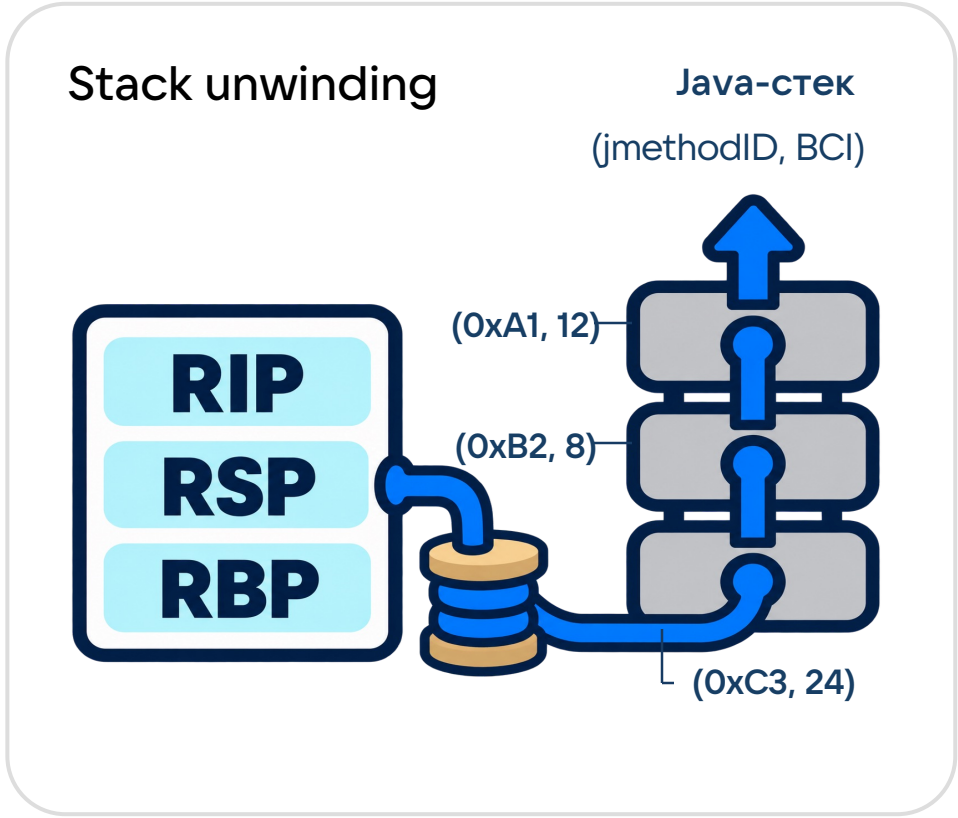
async-profiler

Stack Unwinding в Java

- Дано: регистры потока
- Хотим: кадры стека

Java: jmethodID + BCI

Может, JVM уже умеет?



async-profiler

GetStackTrace & Safepoint bias

До async-profiler: GetStackTrace из JVM TI



async-profiler

AsyncGetCallTrace — источник названия

Первый подход в async-profiler

+ Без ожидания safepoint

– Нестабильный API: мог вызвать краш JVM



async-profiler

Современный подход — VM Structs

Свой обход стека JVM
Без safepoint bias

Mixed-стек Java + native
в режиме vmx

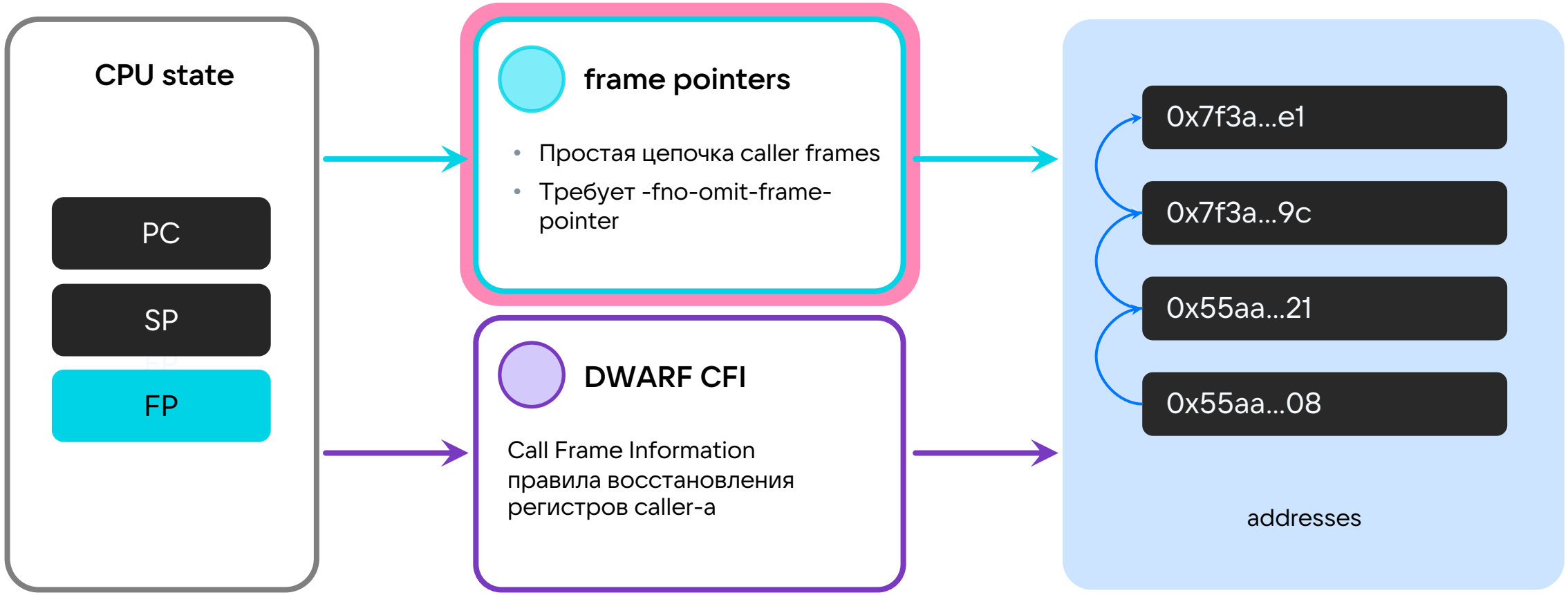




А как получаем натив & kernel?

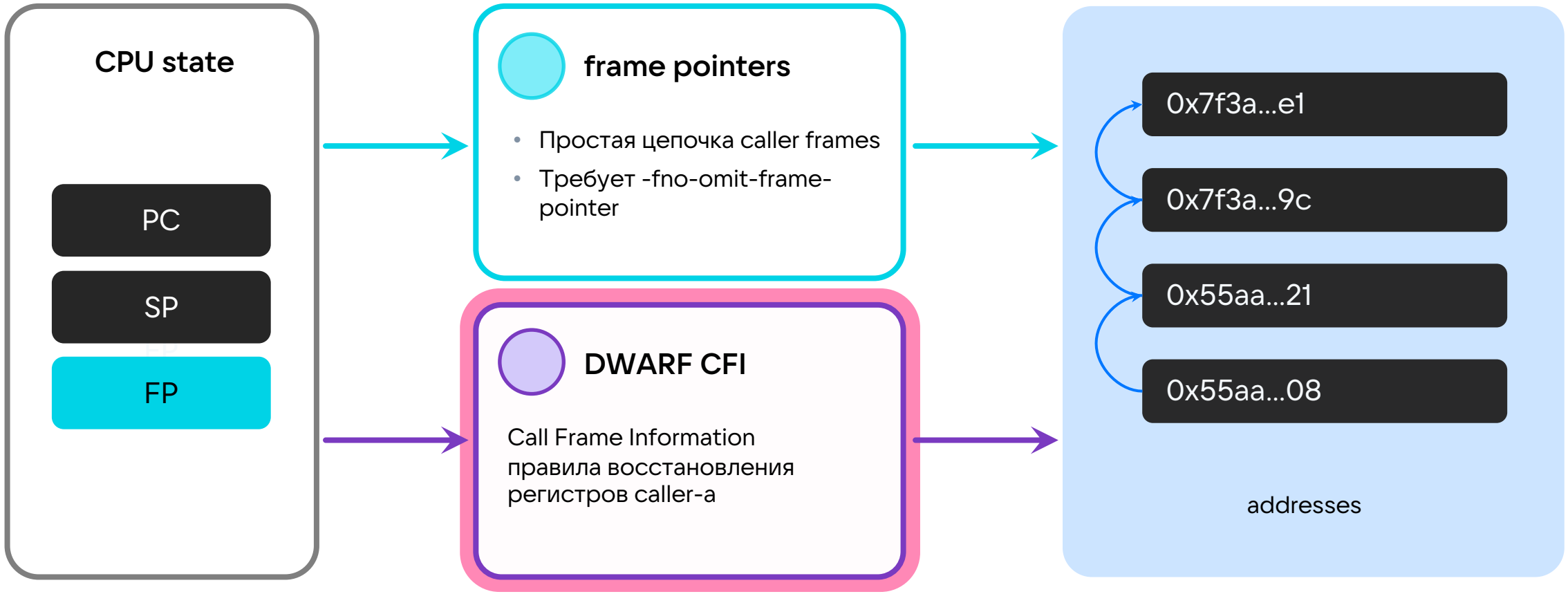
async-profiler

Stack Unwinding для натива



async-profiler

Stack Unwinding для натива



async-profiler

Stack Unwinding для kernel

Kernel-стек раскручивает Linux

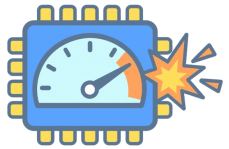


В обработчике сигнала kernel-кадры уже готовы

async-profiler

Путь от CPU до Flame Graph-a

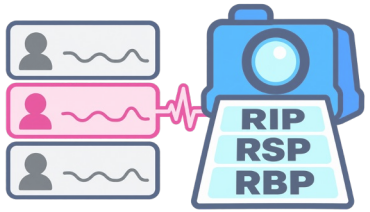
1



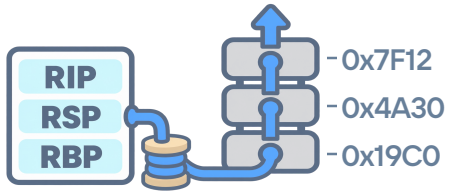
perf_events 2



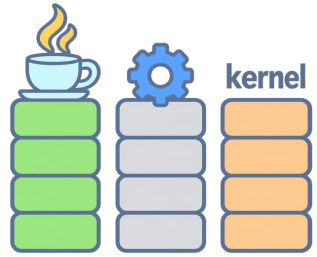
Сигнал потоку 3



Stack Unwinding 4

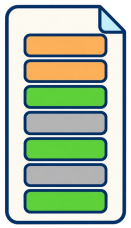


Java + native + kernel 5



Собираем кадры в семпл

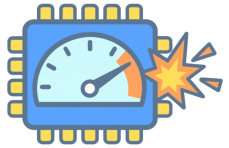
Семпл 6



async-profiler

Путь от CPU до Flame Graph-а

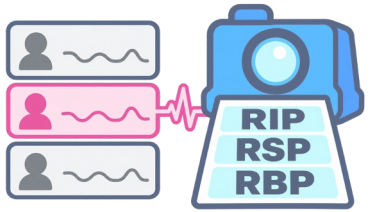
1



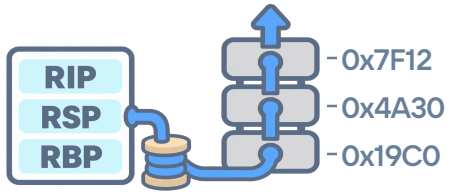
perf_events 2



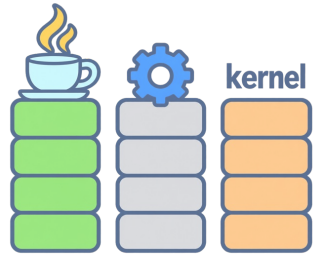
Сигнал потоку 3



Stack Unwinding 4

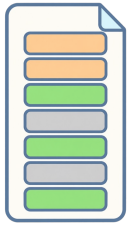


Java + native + kernel 5



Стек раскрутили. Теперь нужны имена

Сэмпл 6



Символизация: Java, native, kernel 7



async-profiler

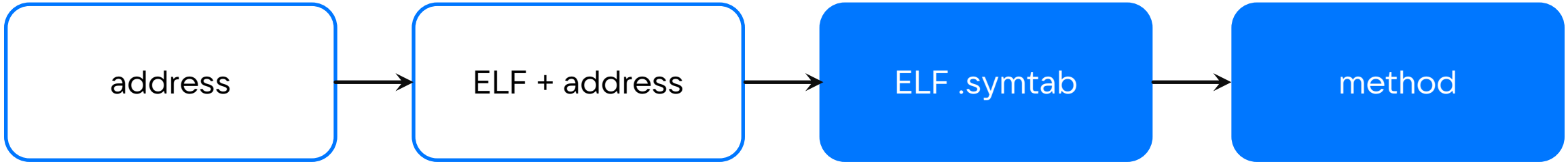
Путь от CPU до Flame Graph-a

Символизация Java



Метаданные JVM

Символизация native



Адрес внутри ELF

* В async-profiler native-кадры обычно без номеров строк и не учитывают inline-функции.
Другие профилировщики по DWARF .debug_info / .debug_line / GSYM резолвят

async-profiler

Путь от CPU до Flame Graph-a

Символизация kernel

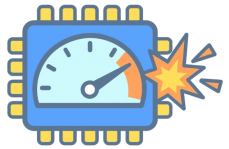



```
sudo cat /proc/kallsyms | head -5
0000000000000000 A __per_cpu_start
0000000000000000 A fixed_percpu_data
0000000000001000 A cpu_debug_store
0000000000002000 A irq_stack_backing_store
0000000000006000 A cpu_tss_rw
```

async-profiler

Путь от CPU до Flame Graph-а

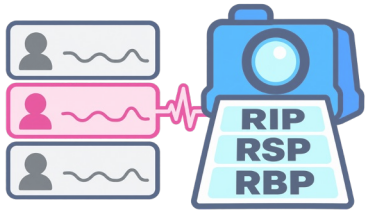
1



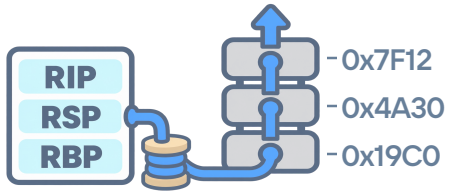
perf_events 2



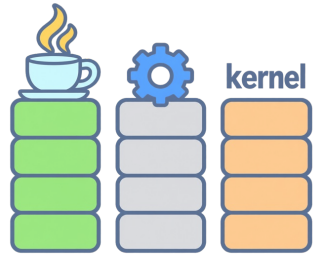
Сигнал потоку 3



Stack Unwinding 4

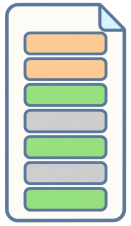


Java + native + kernel 5



Строим Flame Graph из JFR

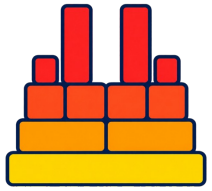
Семпл 6



Символизация: Java, native, kernel 7



JFR/Flame Graph 8



async-profiler

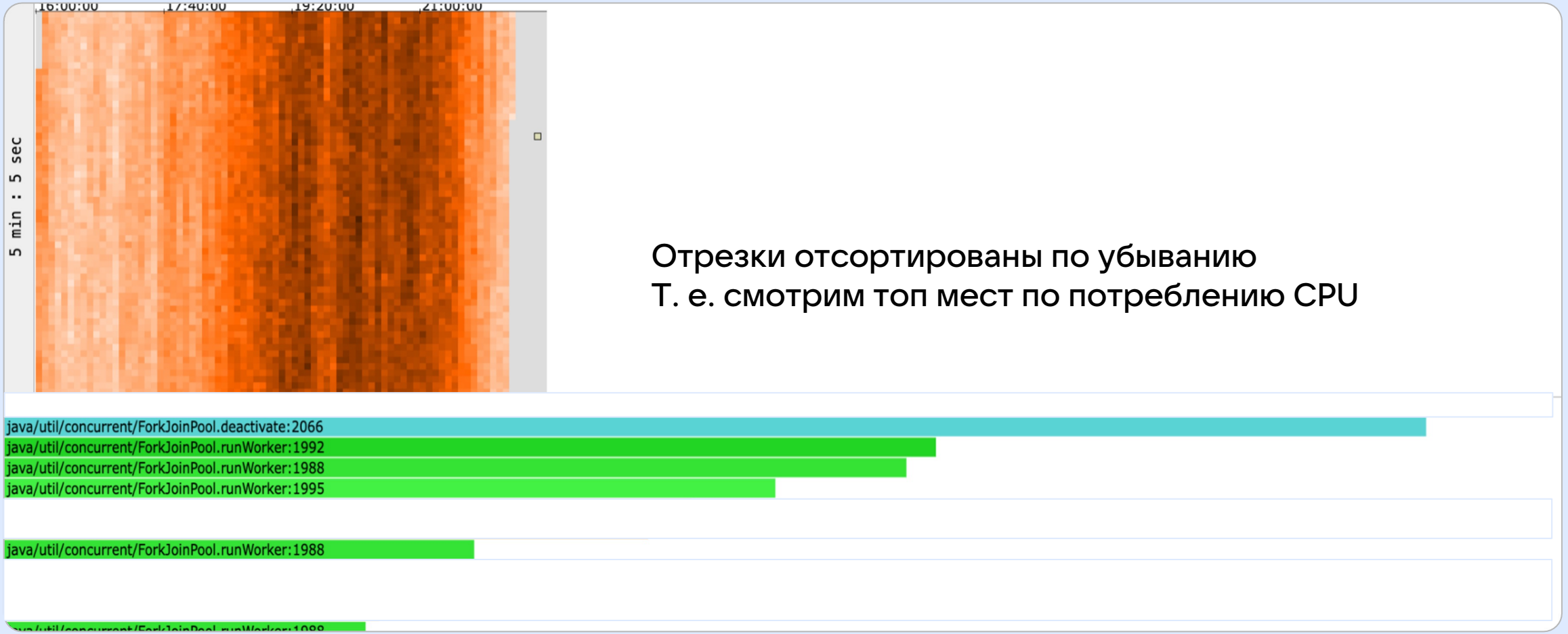
А натив & kernel?

async-profiler
видит натив и kernel

Это очень важное свойство async-profiler, т. к. не надо использовать другие инструменты

Облачные проблемы и решения

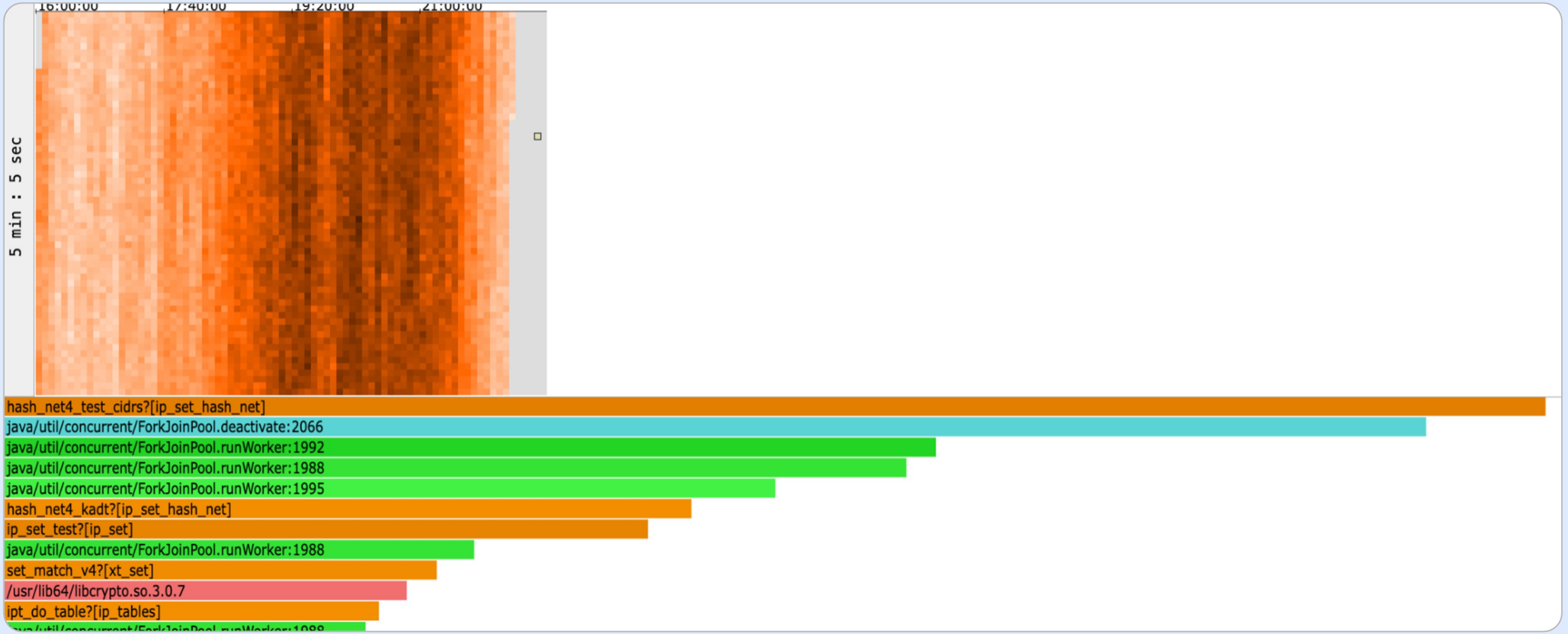
Видимость kernel помогла



Отрезки отсортированы по убыванию
Т. е. смотрим топ мест по потреблению CPU

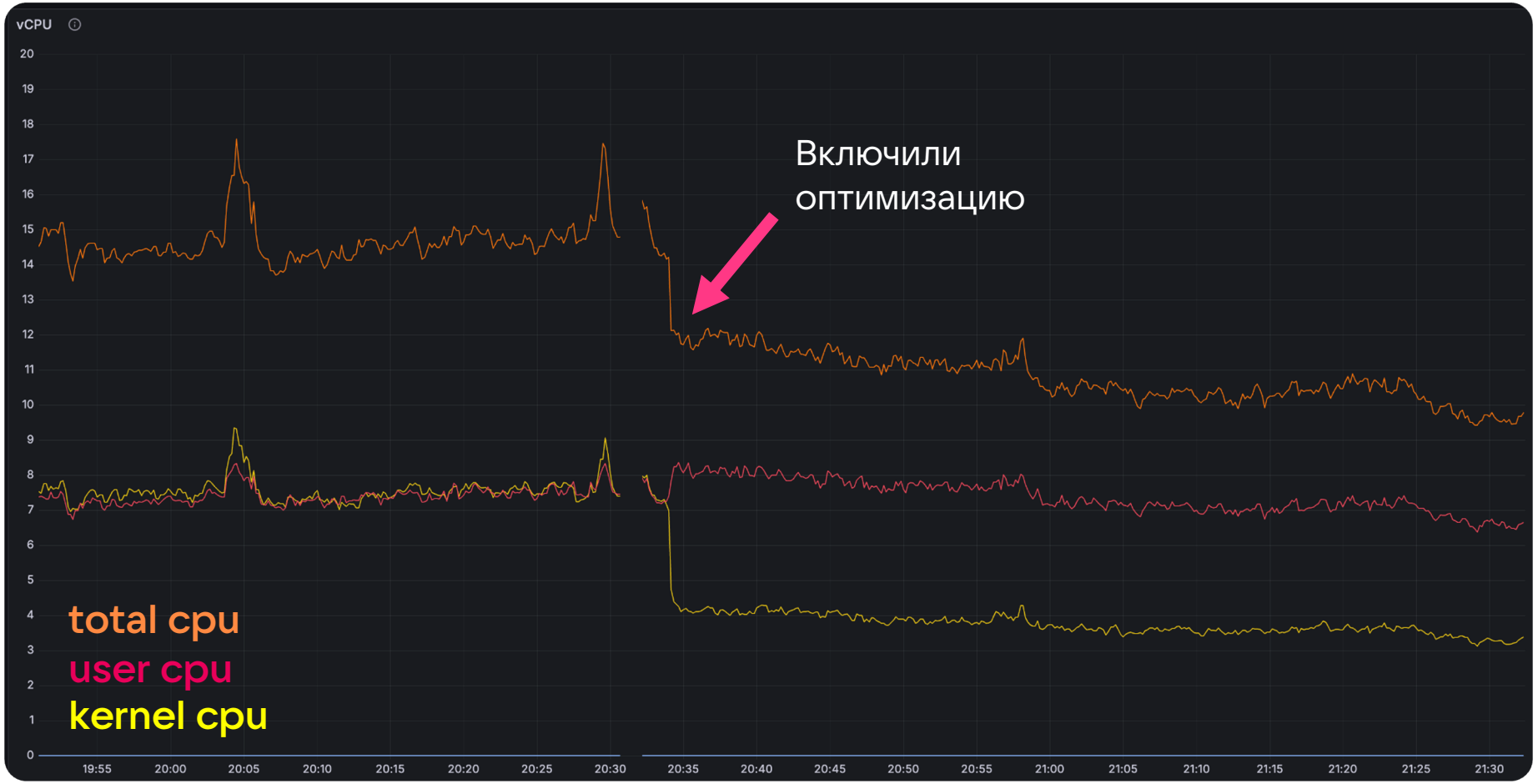
Облачные проблемы и решения

Видимость kernel помогла



Облачные проблемы и решения

Видимость kernel помогла



Облачные проблемы и решения

Но, что если проблема уже прошла?

Просто async-profiler нам уже не поможет

JVM или хост могли уже умереть

Непрерывное профилирование в Облаке

Облачные проблемы и решения

1 Контейнеров много, потоков ещё больше



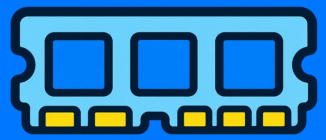
Хотим минимальный оверхед

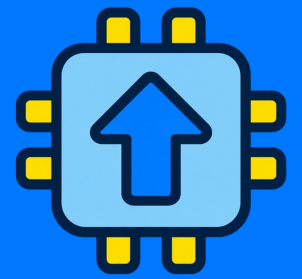
Облачные проблемы и решения

perf-event на каждый поток

- Отдельный FD
  на каждый поток
- Память на счётчики
  и буферы семплов
- Оверхед при
  переключении потоков





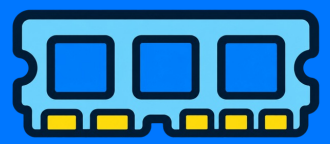


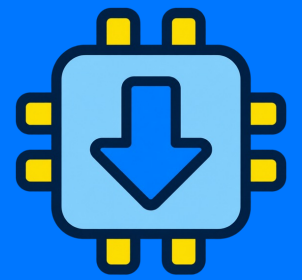
Облачные проблемы и решения

Каждому CPU по perf-счетчику

- Экономим файловые дескрипторы
- Экономим память для передачи стеков (по дефолту ее хватает на 64 потока на ядре)
- Снижаем оверхед на CPU



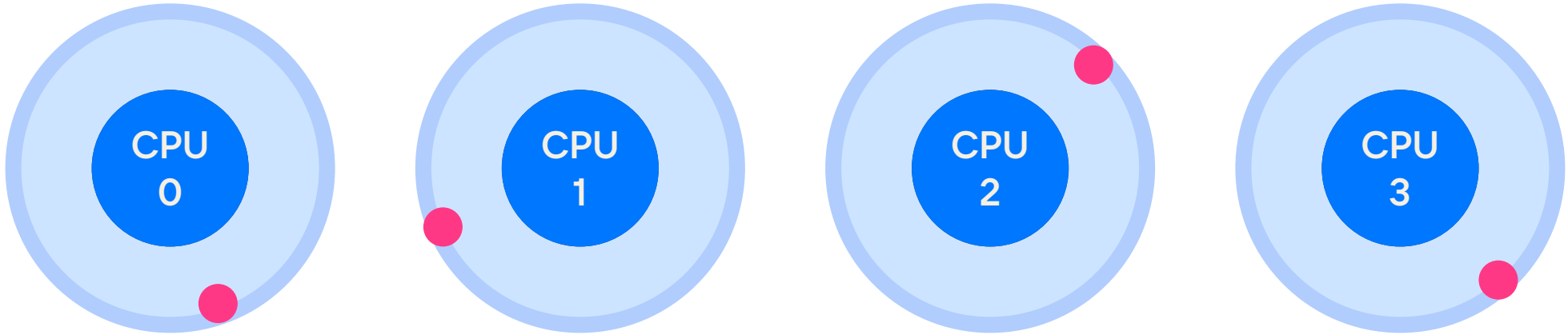




Облачные проблемы и решения

Каждому CPU по perf-счетчику

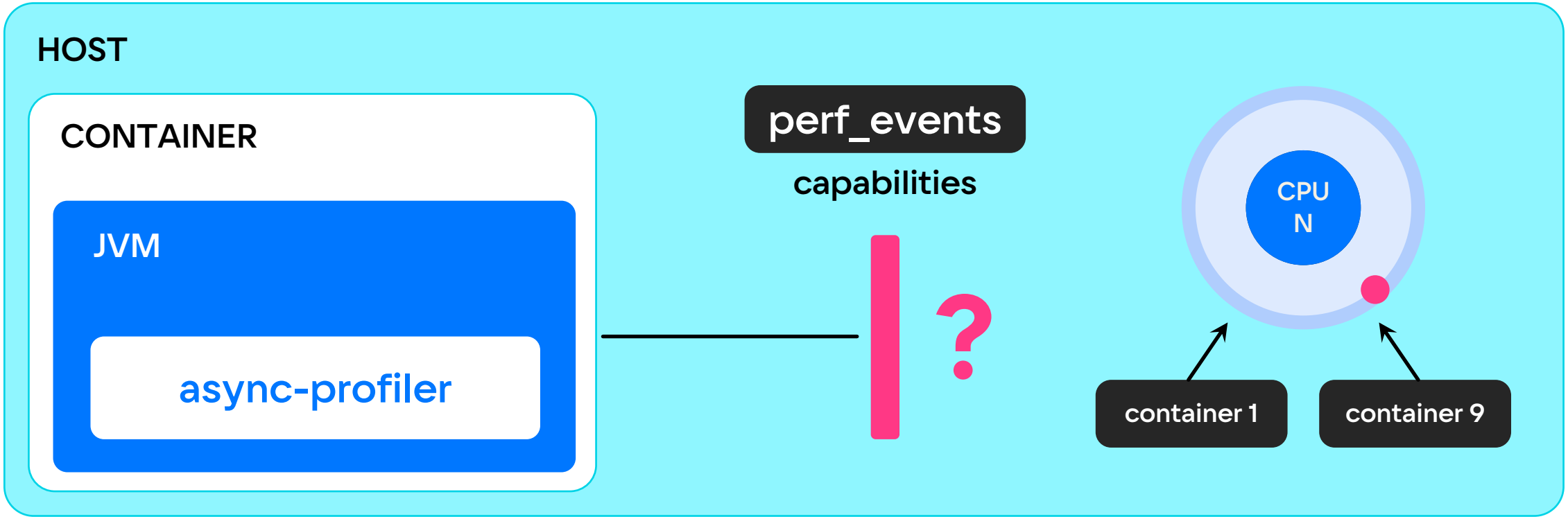
При overflow смотрим, какой поток сейчас исполняется на ядре



НО на CPU исполняются разные контейнеры хоста

Облачные проблемы и решения

2 Видим все исполняющиеся контейнеры на CPU → нужен широкий доступ



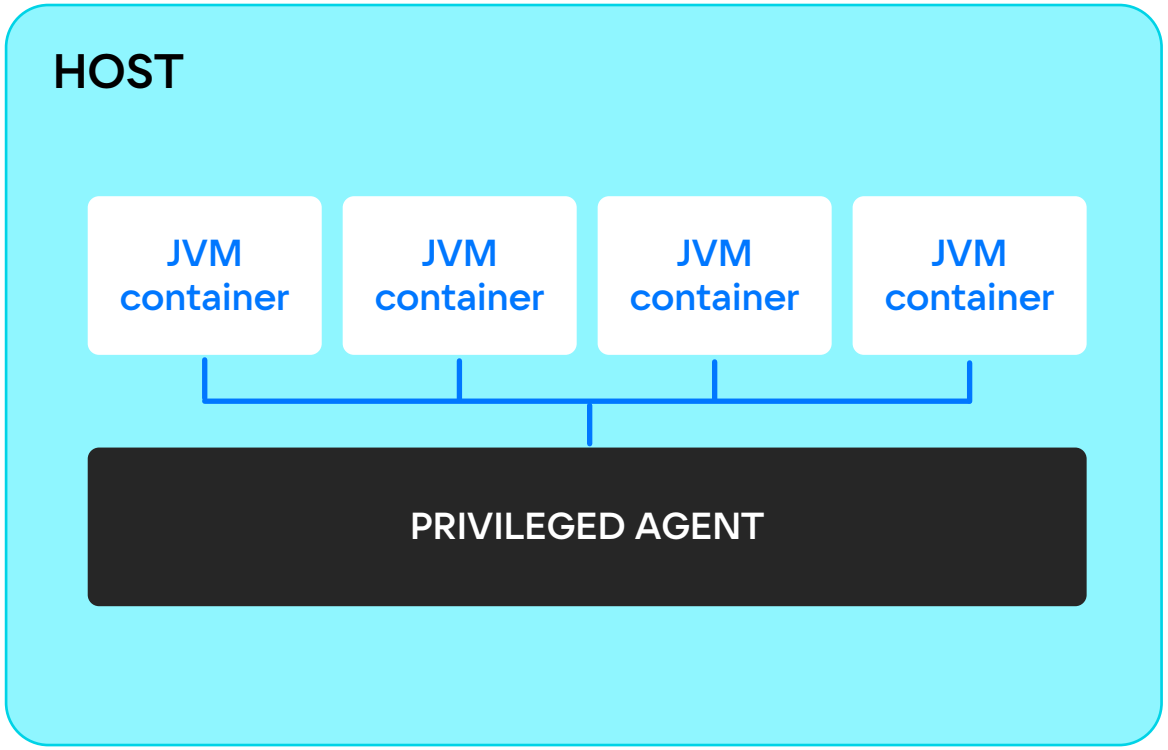
Для perf_events на CPU прав у контейнера не хватает

Облачные проблемы и решения

Привилегированный контейнер

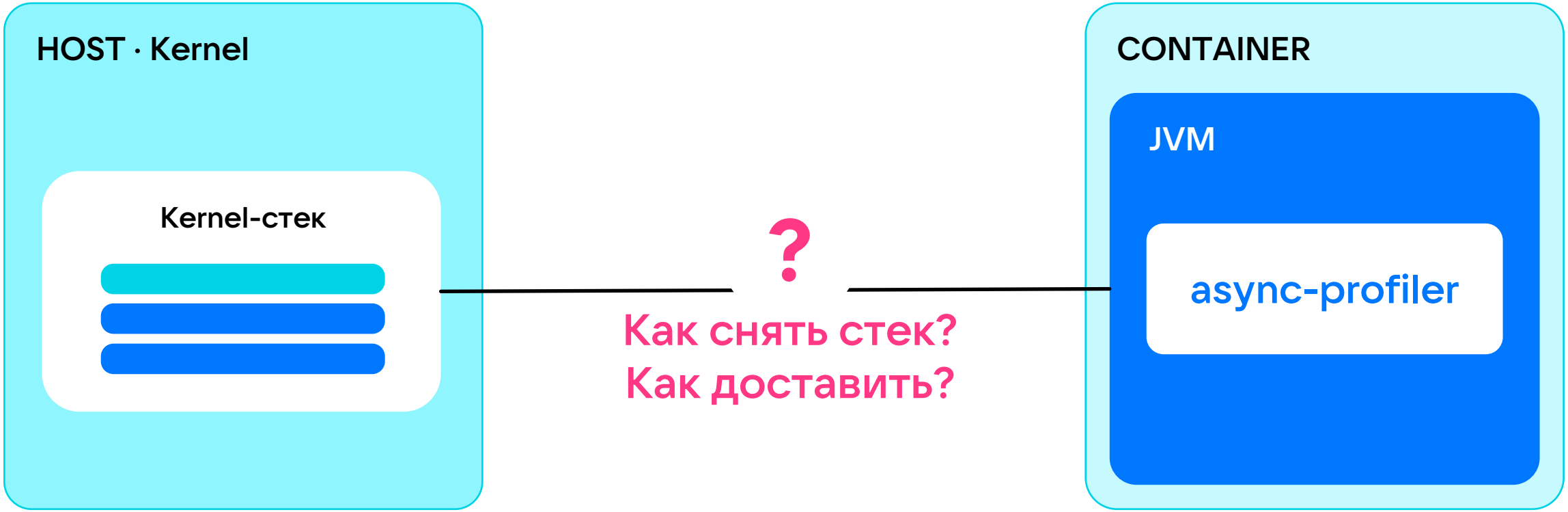
- Имеет расширенные права
- Видит весь хост
- Имеем доступ к perf_events на CPU
- Не зависит от жизни наблюдаемых контейнеров

НО теперь надо как-то async-profiler с ним подружить



Облачные проблемы и решения

3 С минимальным оверхедом получить kernel-стек на CPU
и обработать в async-profiler исполняющегося JVM-потока



Облачные проблемы и решения

eBPF

eBPF-программа (на Си) исполняется в ядре

Нет лишнего переключения между kernel и user space → экономим CPU

Триггерится при срабатывании perf_events

Много возможностей из коробки — eBPF helpers

- `bpf_get_stack`
- `bpf_send_signal_thread`

Передаём стек через eBPF map

Облачные проблемы и решения

4 Надо подружить **eBPF** и **async-profiler**

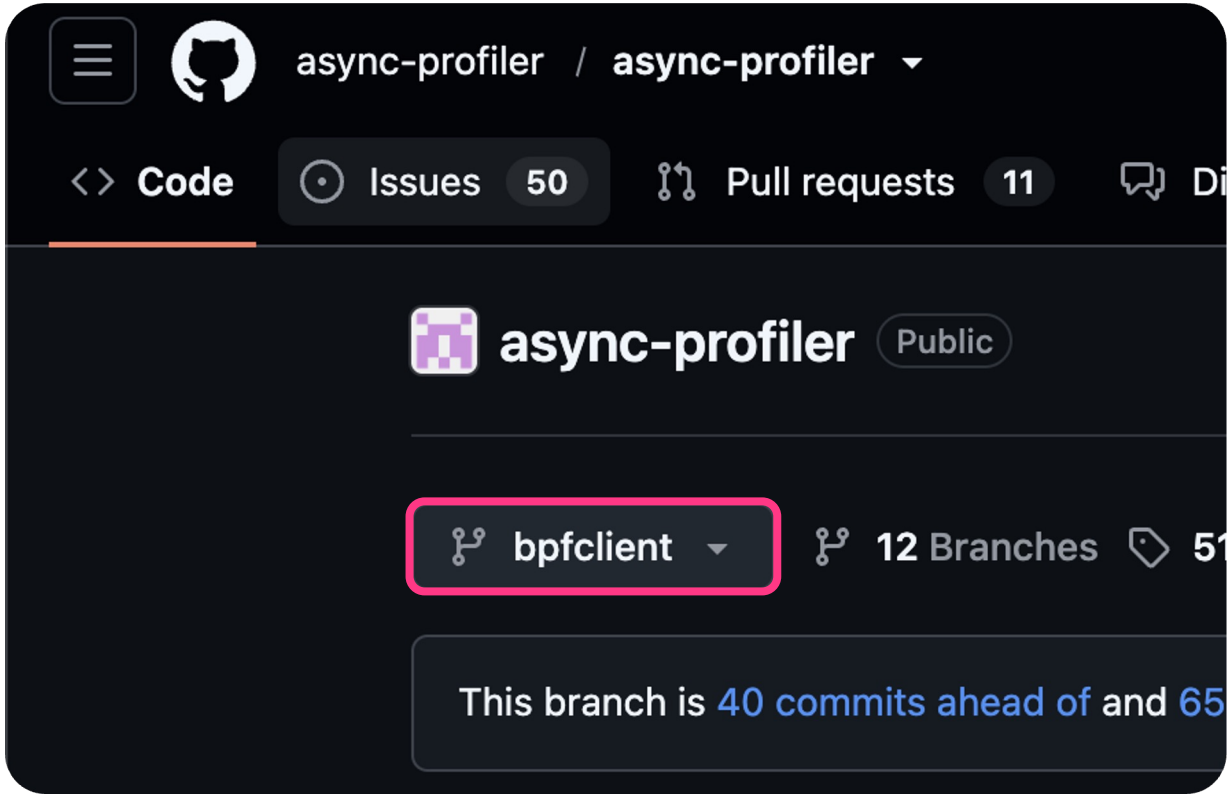


Обычный async-profiler не принимает готовые семплы eBPF

Облачные проблемы и решения

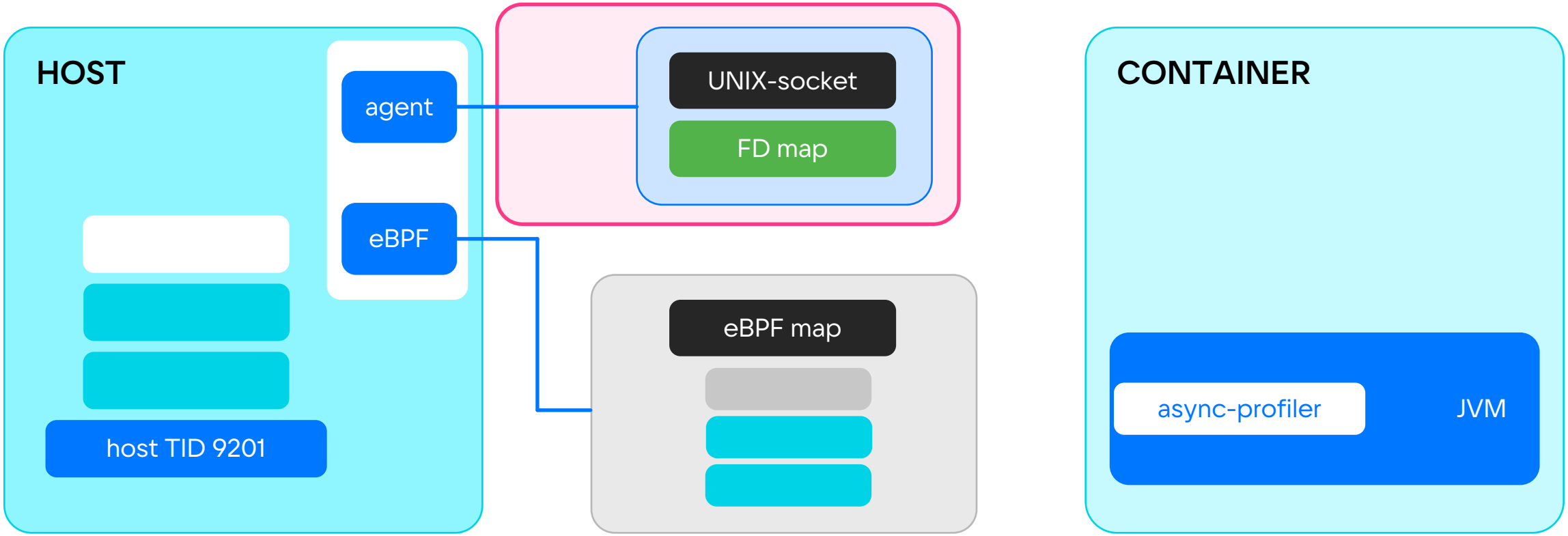
async-profiler с eBPF

- Патченный async-profiler
- kernel-стек получает из eBPF
- native и Java раскручивает сам



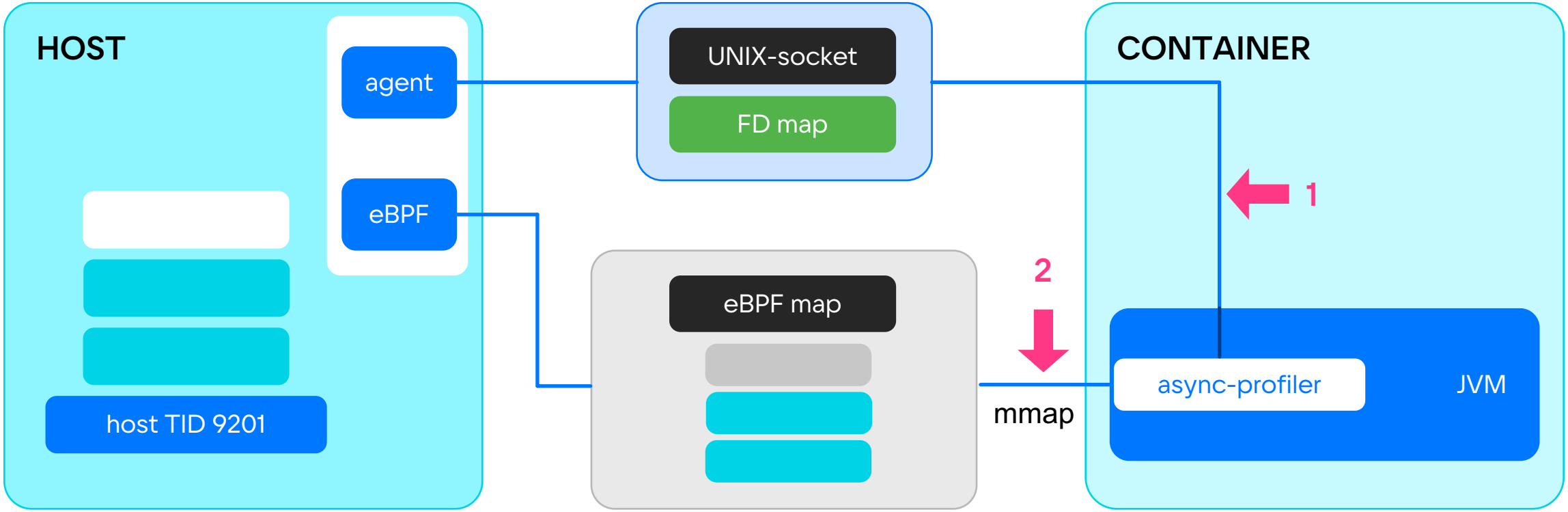
Облачные проблемы и решения

Передаём FD map при подключении



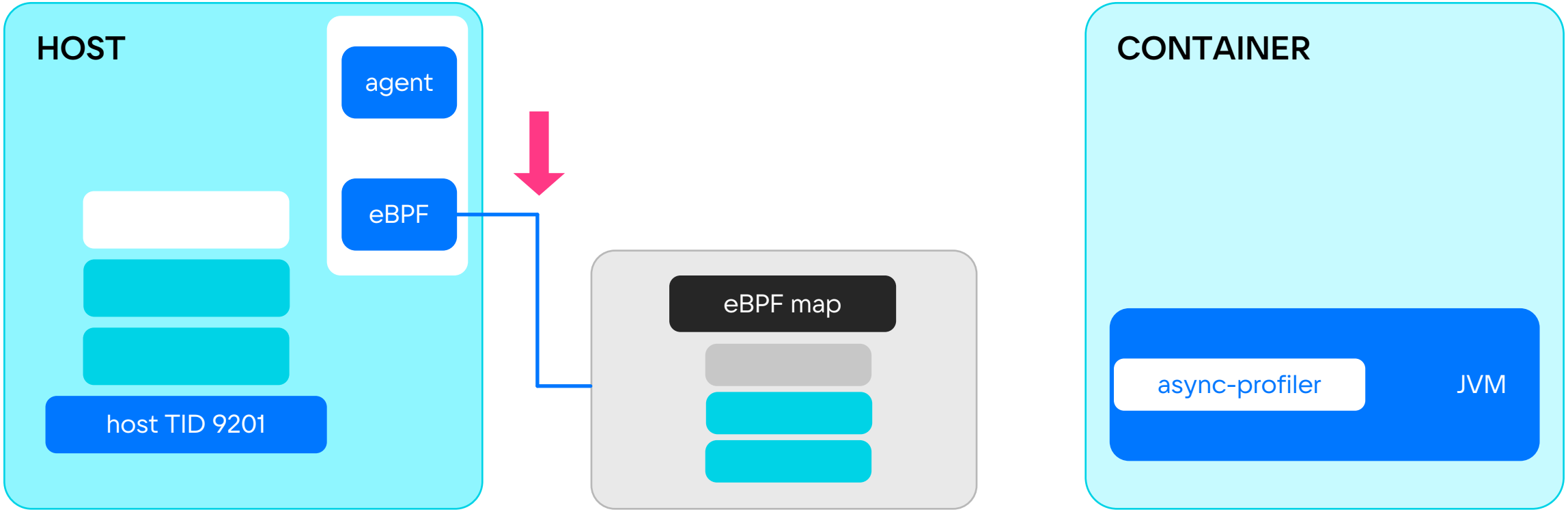
Облачные проблемы и решения

- Получаем дескриптор map
- Делаем mmap один раз



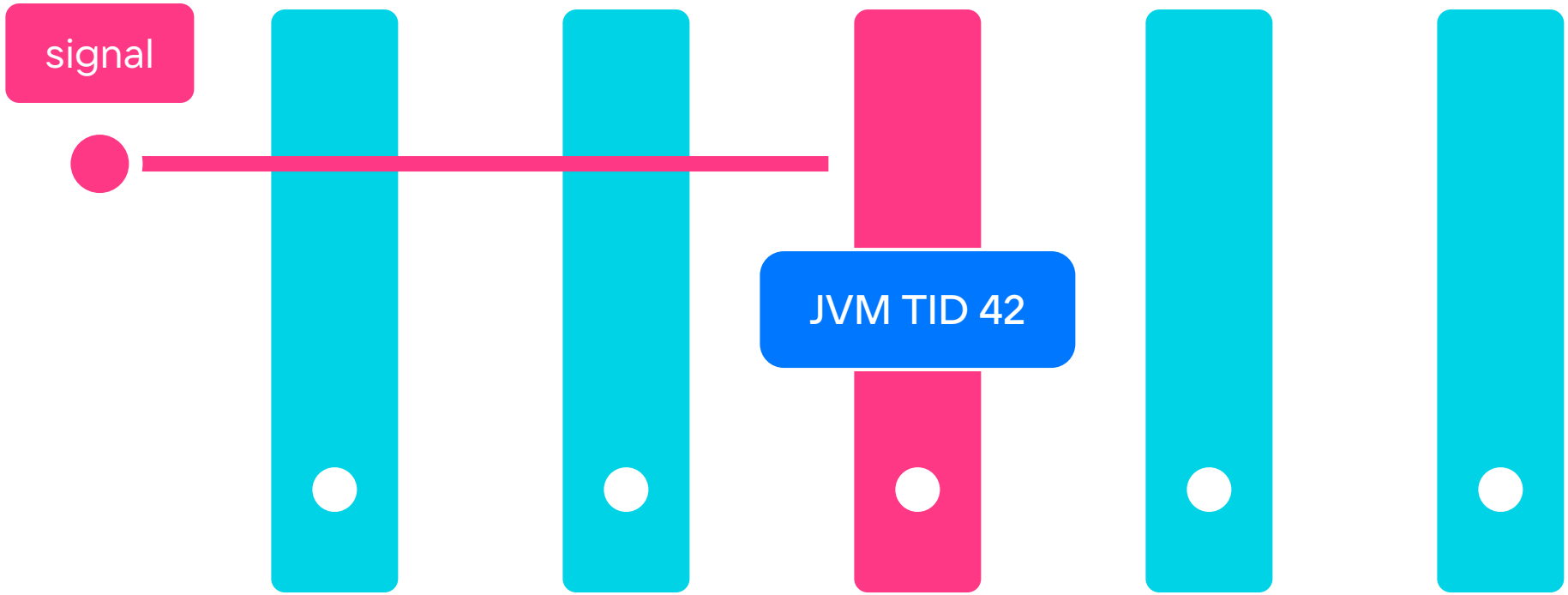
Облачные проблемы и решения

eBPF записывает данные семпла



Облачные проблемы и решения

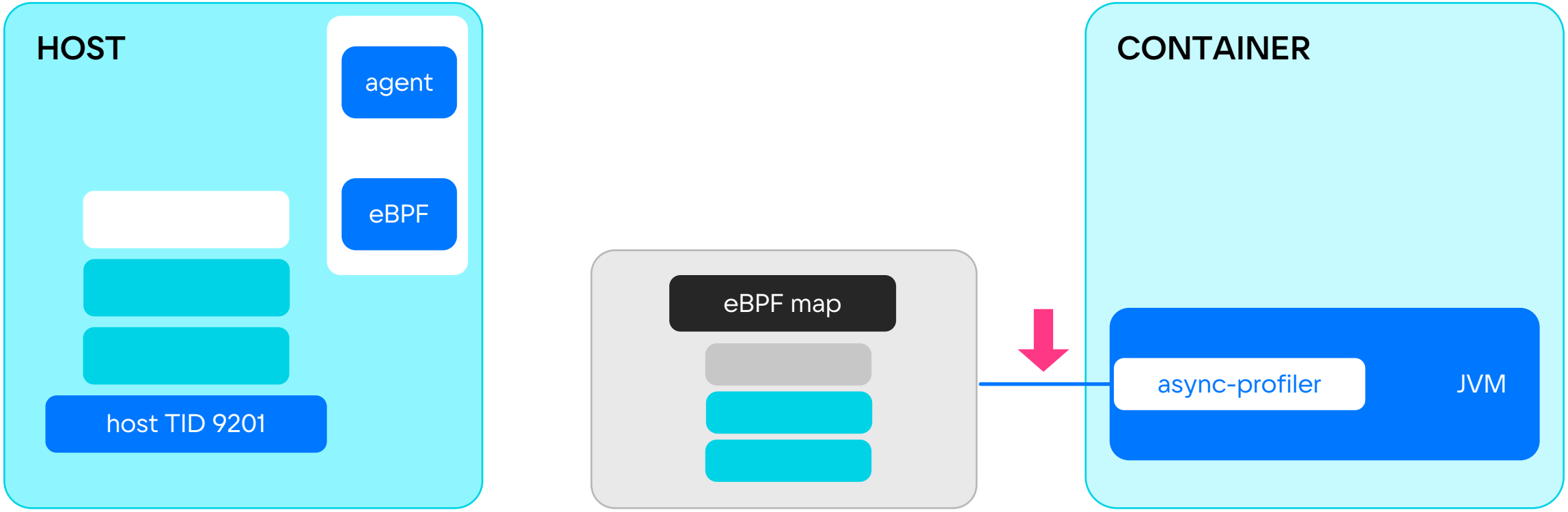
eBPF отправляет сигнал потоку



Поток выполняет обработчик сигнала async-profiler-а

Облачные проблемы и решения

На сигнале читаем ячейку map



Облачные проблемы и решения

Собираем семпл для JFR



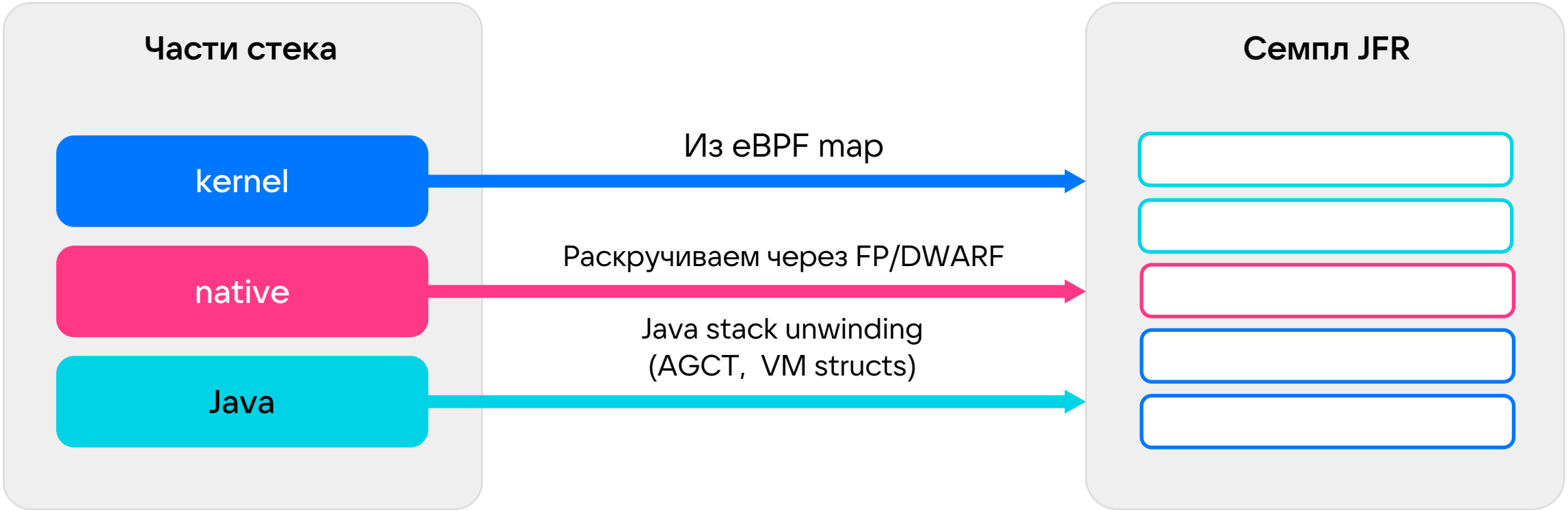
Облачные проблемы и решения

Собираем семпл для JFR



Облачные проблемы и решения

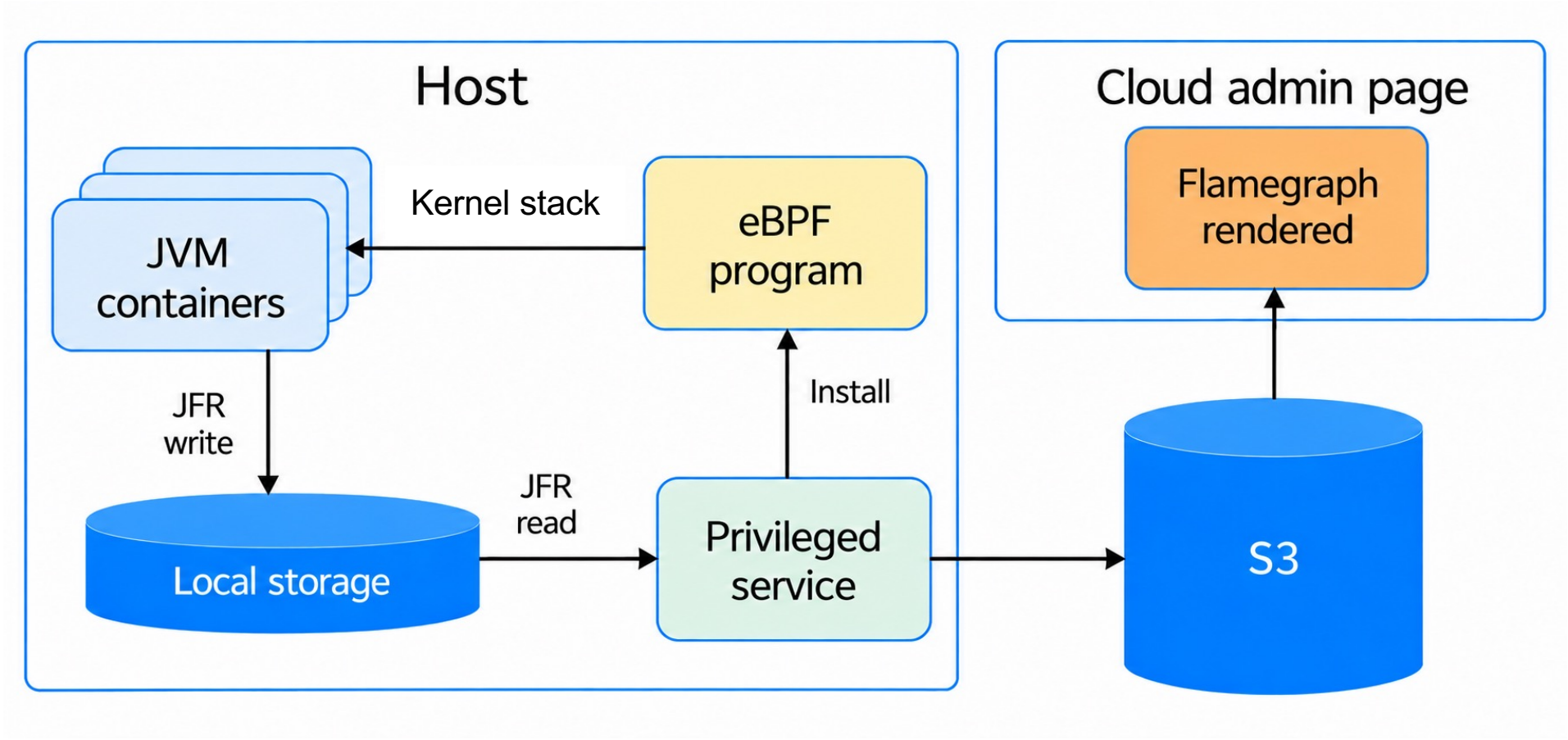
Собираем семпл для JFR



one-continuous-profiler

one-continuous-profiler

Архитектура one-continuous-profiler



one-continuous-profiler в Kubernetes

Примерная схема



Агент на каждом узле

Привилегированный DaemonSet

Подкладываем .so в контейнер

Доставка патченного async-profiler в JVM через ***hostPath***

Метаданные

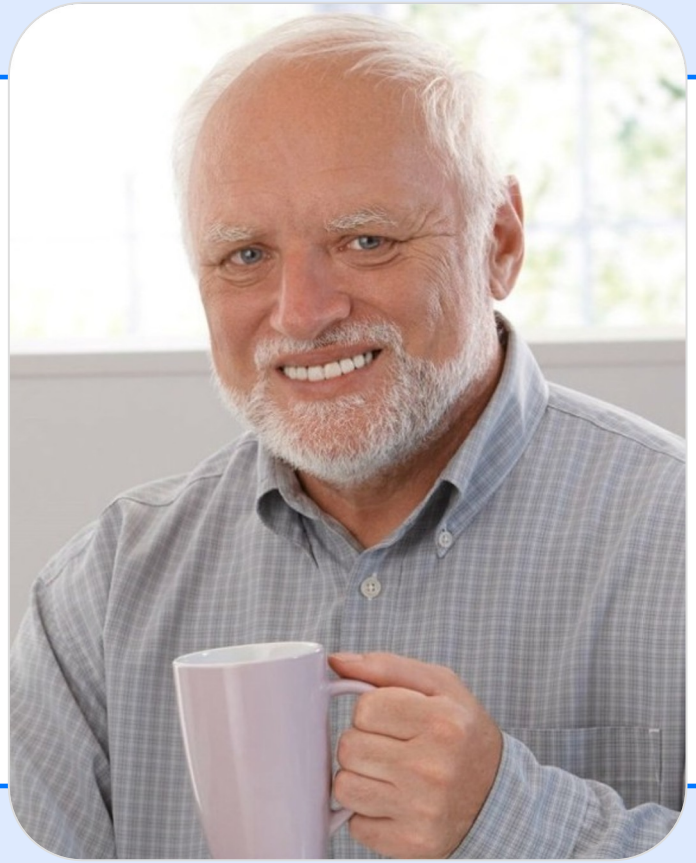
Связываем процесс с pod и сервисом через Kubernetes API

one-continuous-profiler в Kubernetes

А в Kubernetes заведется?

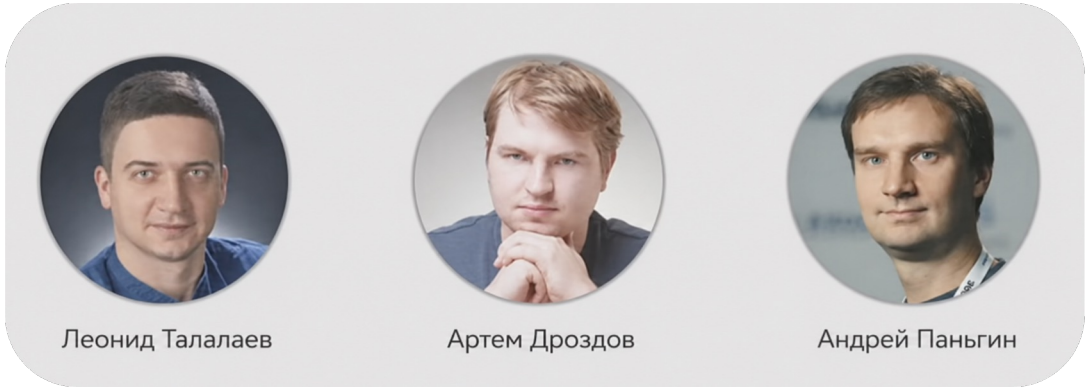
Но:

- схему не тестили
- one-continuous-profiler не open-source



one-continuous-profiler

Подробнее про one-continuous-profiler



Андрей Паньгин, Леонид Талалаев, Артем Дроздов

Рассказывают про непрерывное профилирование и формирование Flame Graph-ов

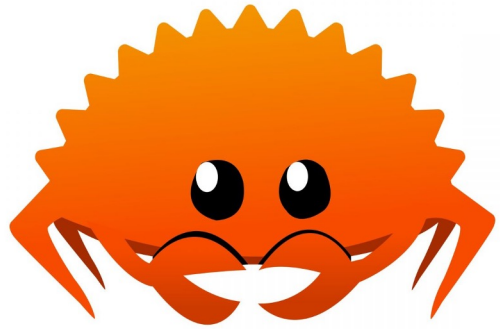
youtu.be/EcmR2BOeVZE

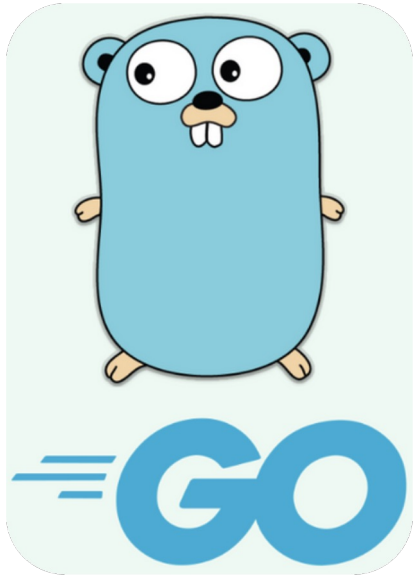
Native & Friends / Linux

Native & Friends / Linux

Другие языки

- Вспоминаем, что у нас не только JVM
- Go/C++/Rust
- Нужно их как-то непрерывно профилировать тоже








Perforator

Perforator is

- a production-ready
- open-source
- Continuous Profiling app
- that can collect CPU profiles from your production
- without affecting its performance
- inspired by Google-Wide Profiling



Perforator is

Минималистичность агента

Символизация откладывается на потом



Perforator is

Минималистичность агента

Символизация откладывается на потом

Масштабирование

Готовность жить в окружении 10к серверов и более



Perforator is

Минималистичность агента

Символизация откладывается на потом

Масштабирование

Готовность жить в окружении 10к серверов и более

Лицензия

Apache-2.0, отдельные части GPL-2.0



Native & Friends / Linux

Языки другие — проблемы те же

01

Контейнеров много, потоков еще больше

→ **01** perf_events на CPU

02

Видим все исполняющиеся контейнеры на CPU → нужен широкий доступ

→ **02** Привилегированный контейнер

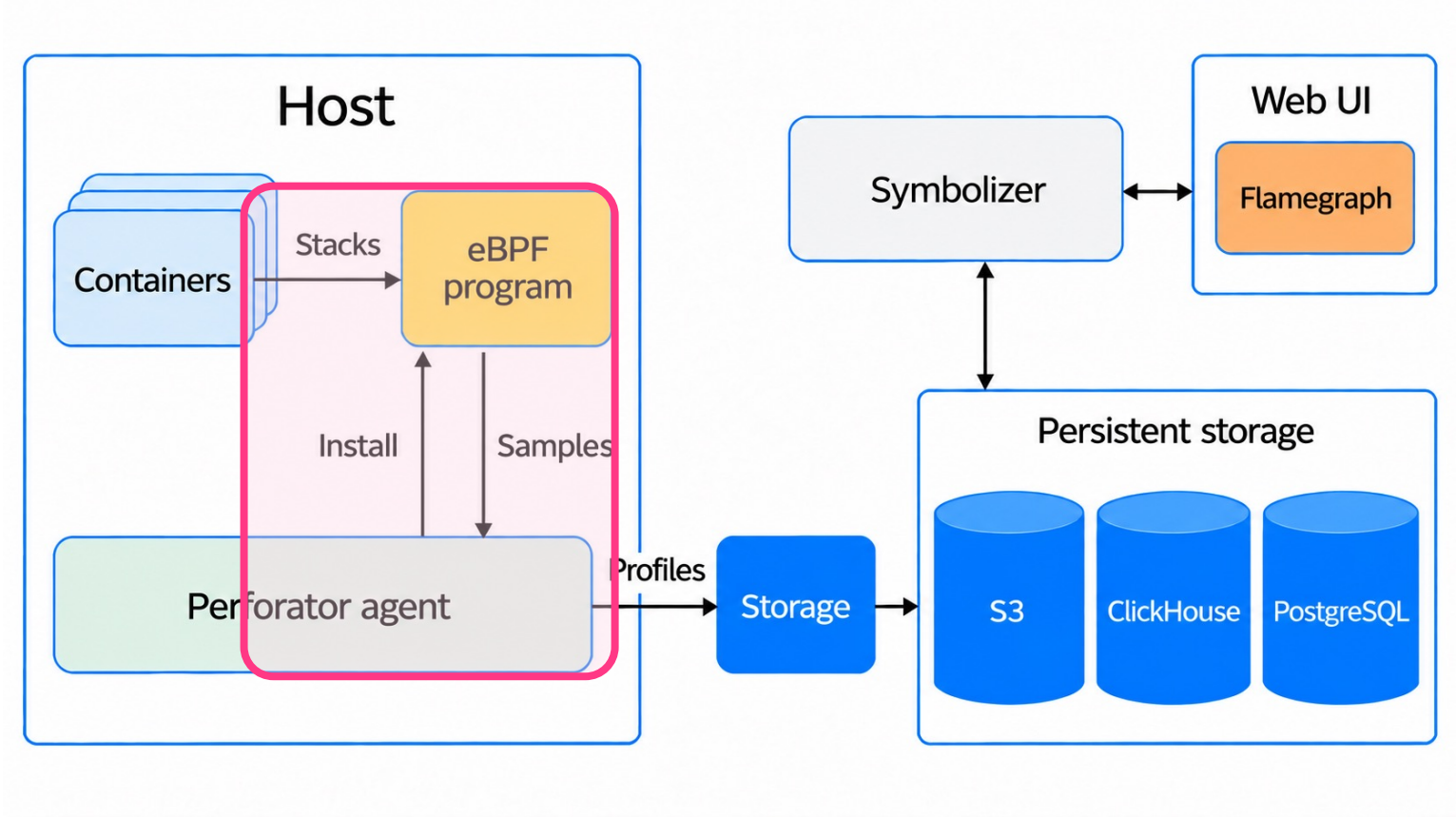
03

С минимальным оверхедом хотим получить стек

→ **03** eBPF

Perforator

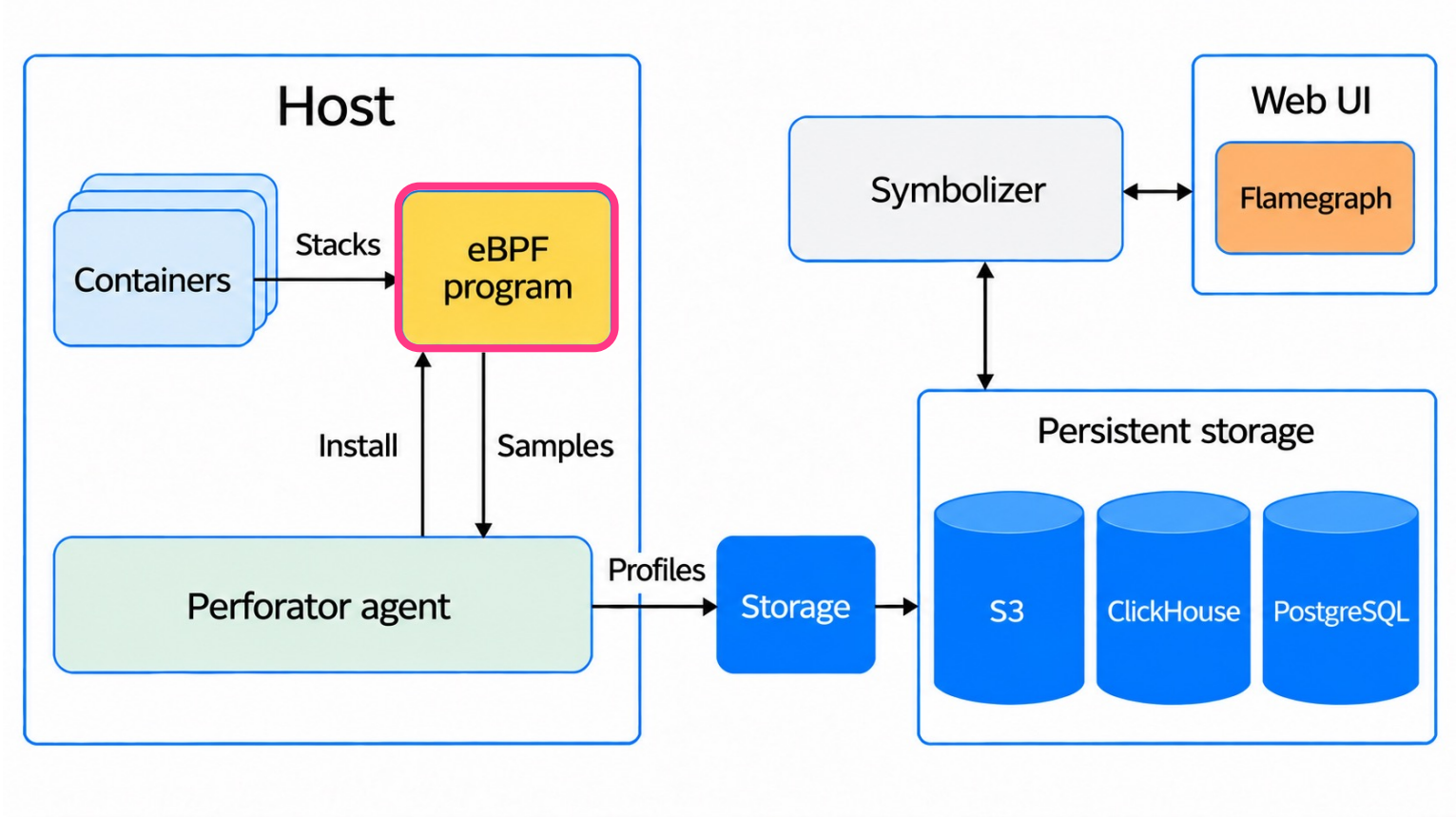
Устройство снятия семпла



eBPF-программа полностью восстанавливает стек и передает агенту уже готовый семпл

Perforator

Устройство снятия семпла

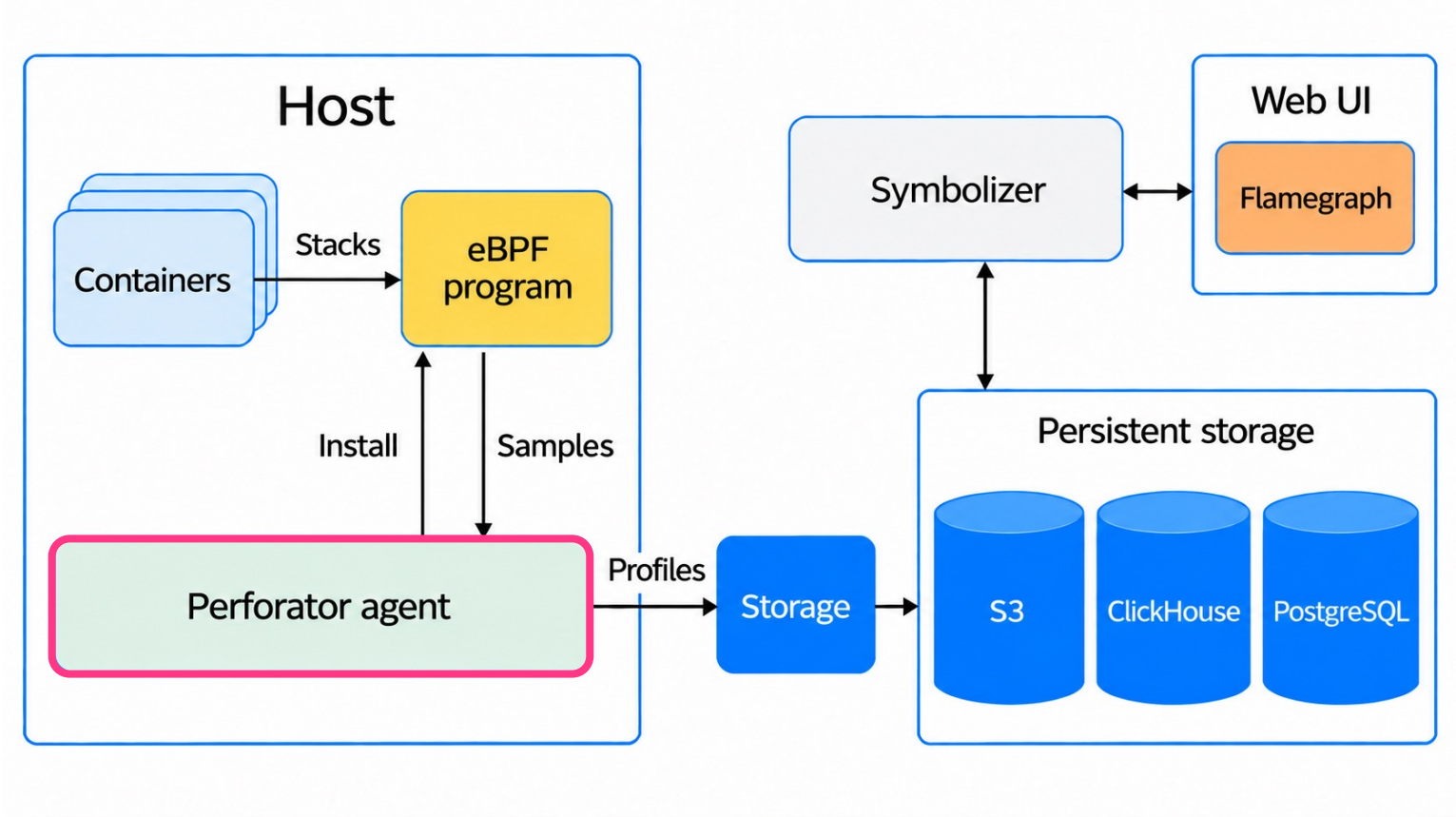


Раскрутка стека (stack unwinding) происходит с помощью FP или DWARF внутри eBPF!

Не внутри контейнера, как у one-continuous-profiler

Perforator

Устройство снятия семпла

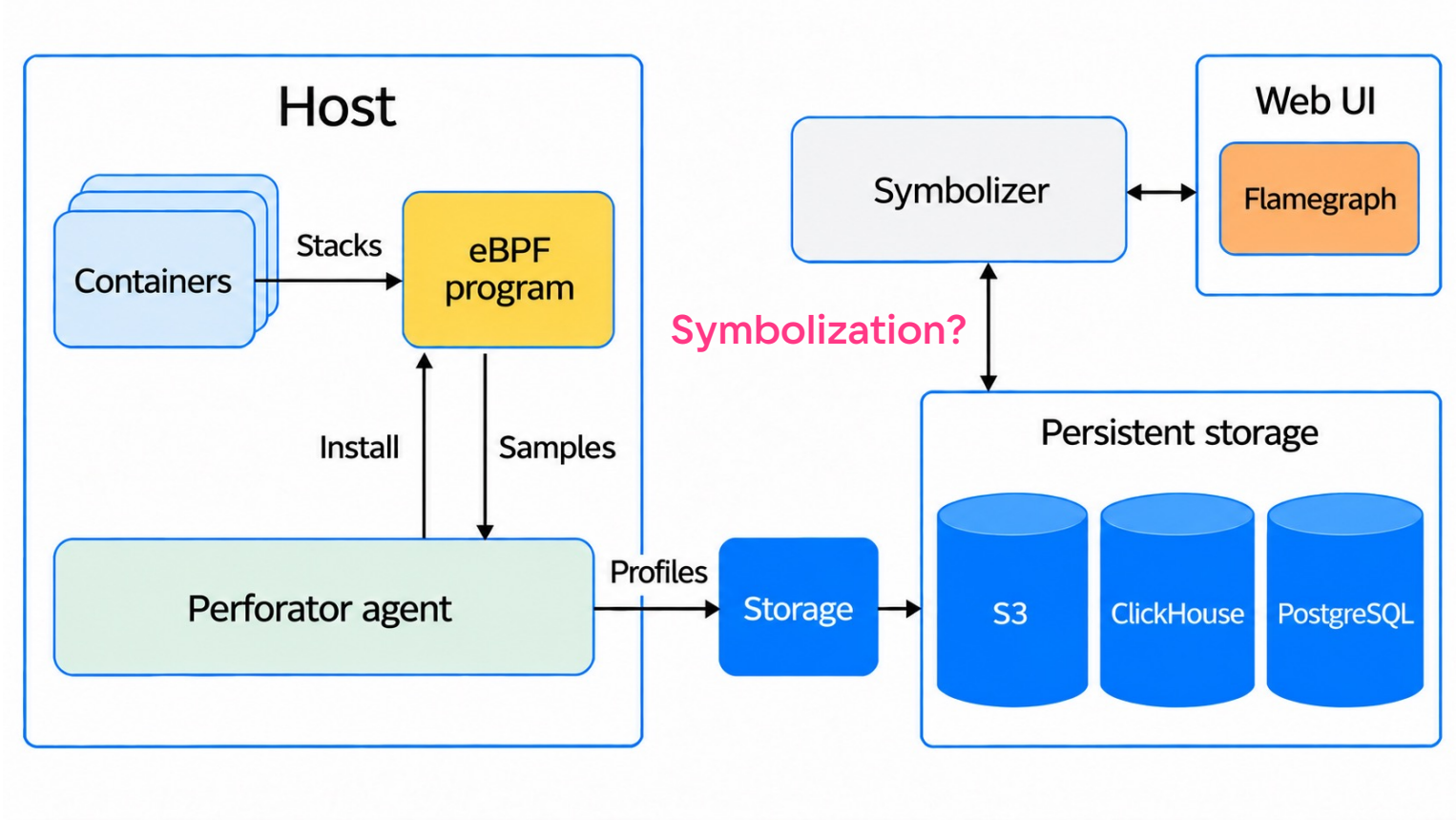


Агент:

- агрегирует семплы
- добавляет метаданные
- периодически формирует pprof

Perforator

Устройство снятия семпла

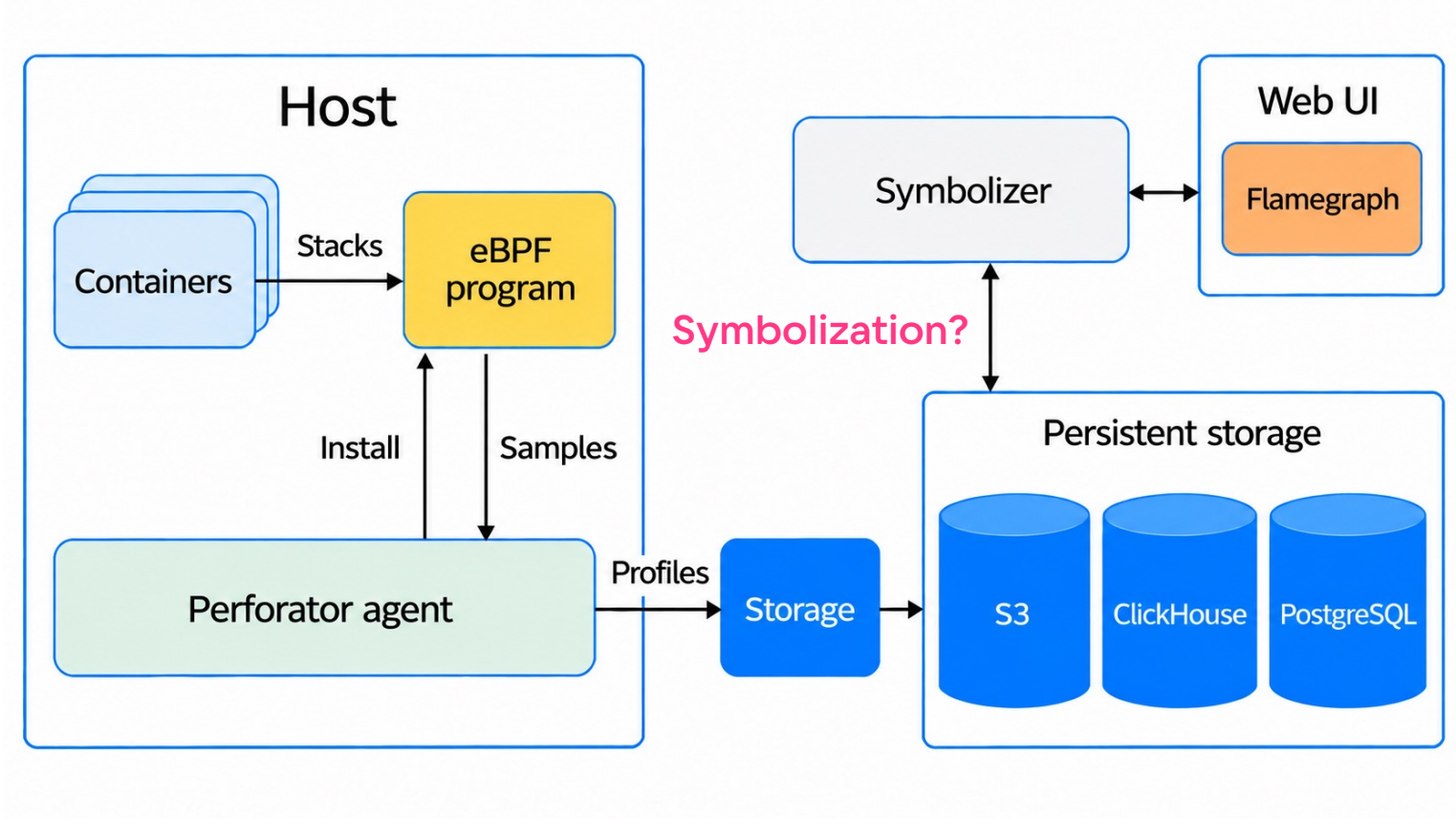


Где символизация?

- Это особенность Perforator: символизация происходит позже — при запросе профиля
- Почему? Это самая вычислительно тяжелая часть, которая теперь выполняется лениво on-demand

Perforator

Устройство снятия семпла



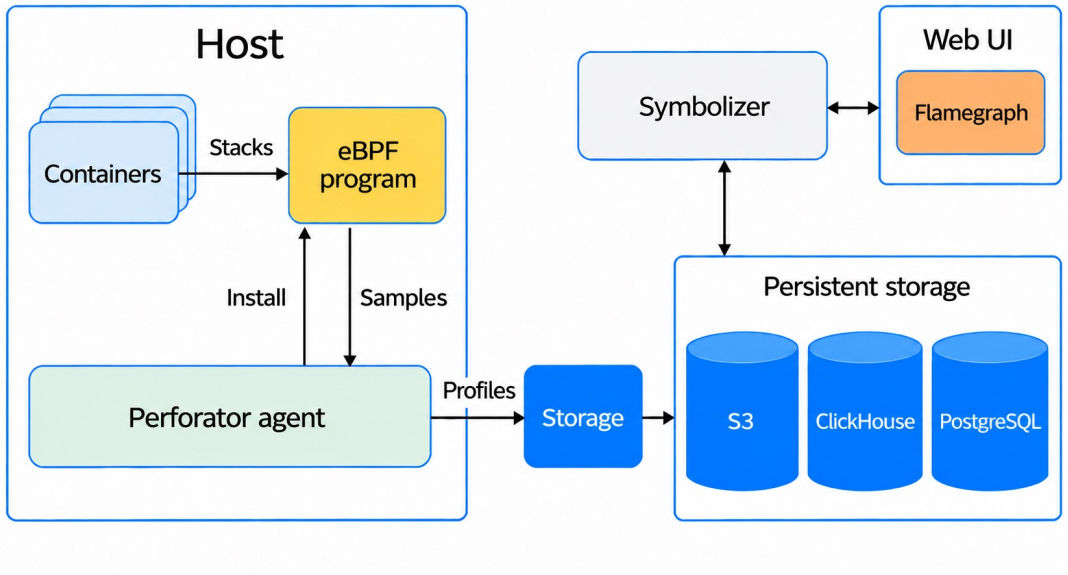
Но для символизации нужен бинарник (ELF)

- Да, поэтому все бинарники выгружаются в S3 с дедупликацией по build-id
- Кроме ELF формируется еще и GSYM — индекс, разработанный LLVM для ускоренной символизации

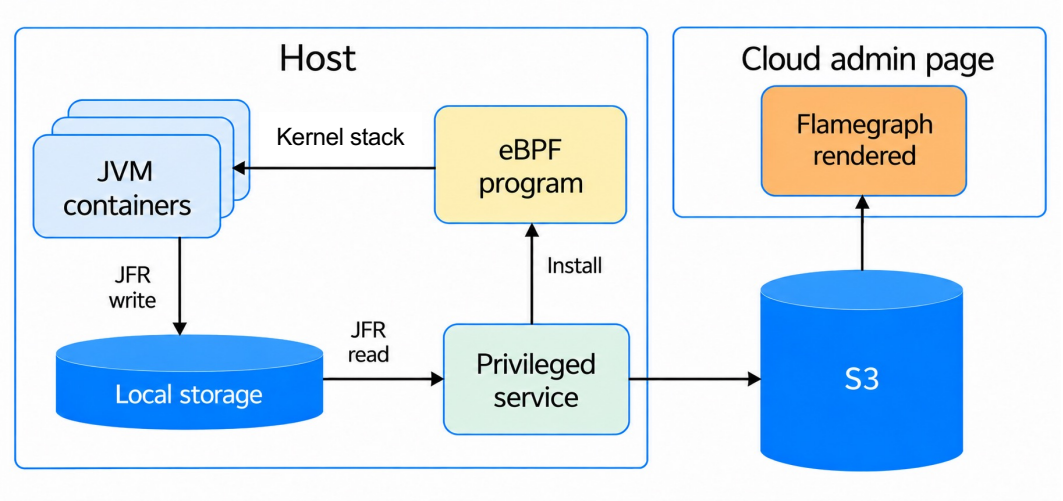
Perforator

Устройство снятия семпла

Perforator



one-continuous-profiler



Perforator

Устройство снятия семпла: все выглядит до боли знакомым

Perforator решает 2 задачи одновременно

Идейно Perforator = async-profiler + one-continuous-profiler

Есть свои особенности — вся раскрутка стека в eBPF, отложенная символизация

Решаемые задачи и подходы к решенияю одинаковые (снять семпл, раскрутить стек, сложить локально, выгрузить, символизировать)

Perforator

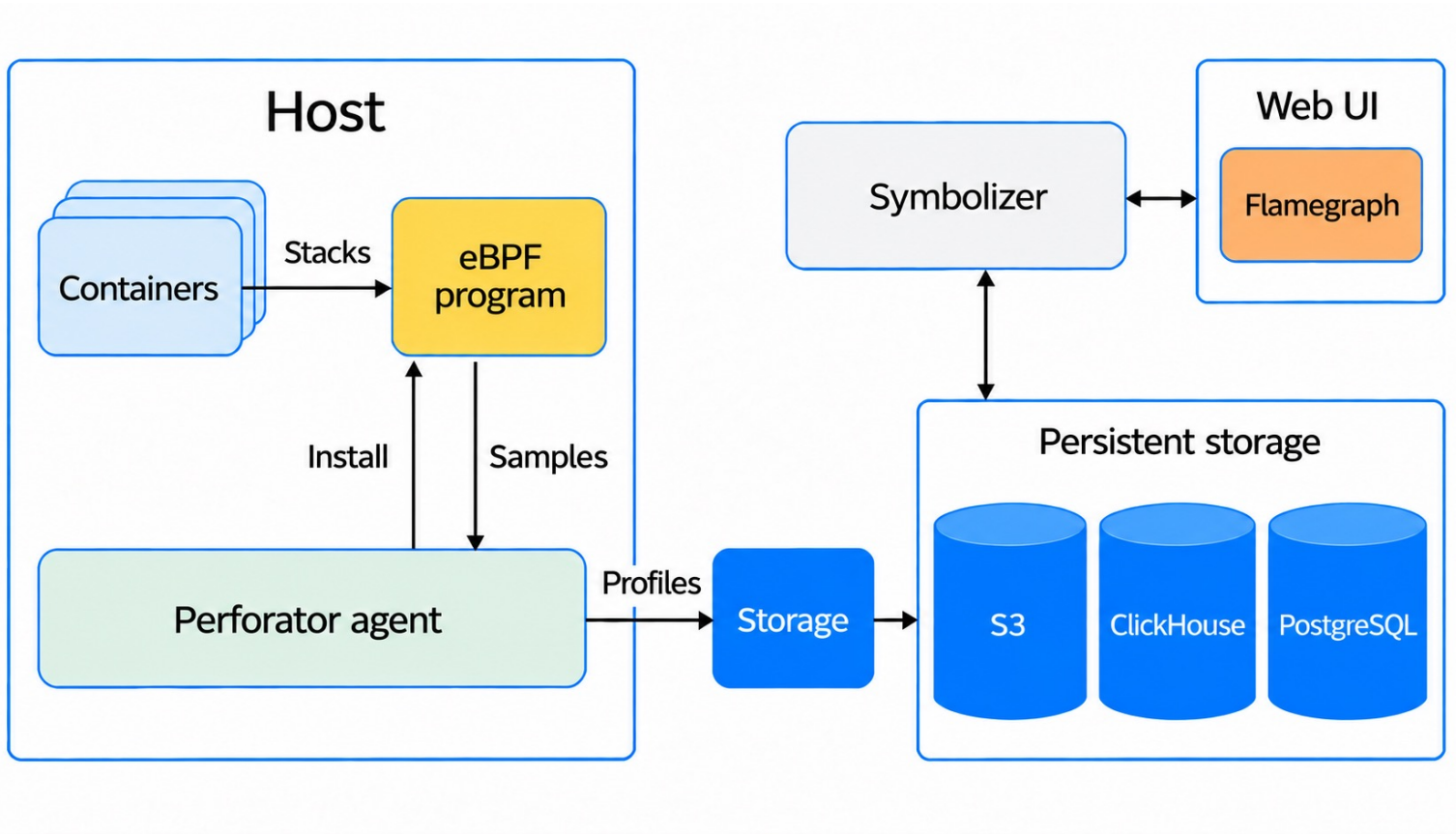
Архитектура

Сервисы

- Agent
- Storage
- Symbolizer
- Web

Хранилища

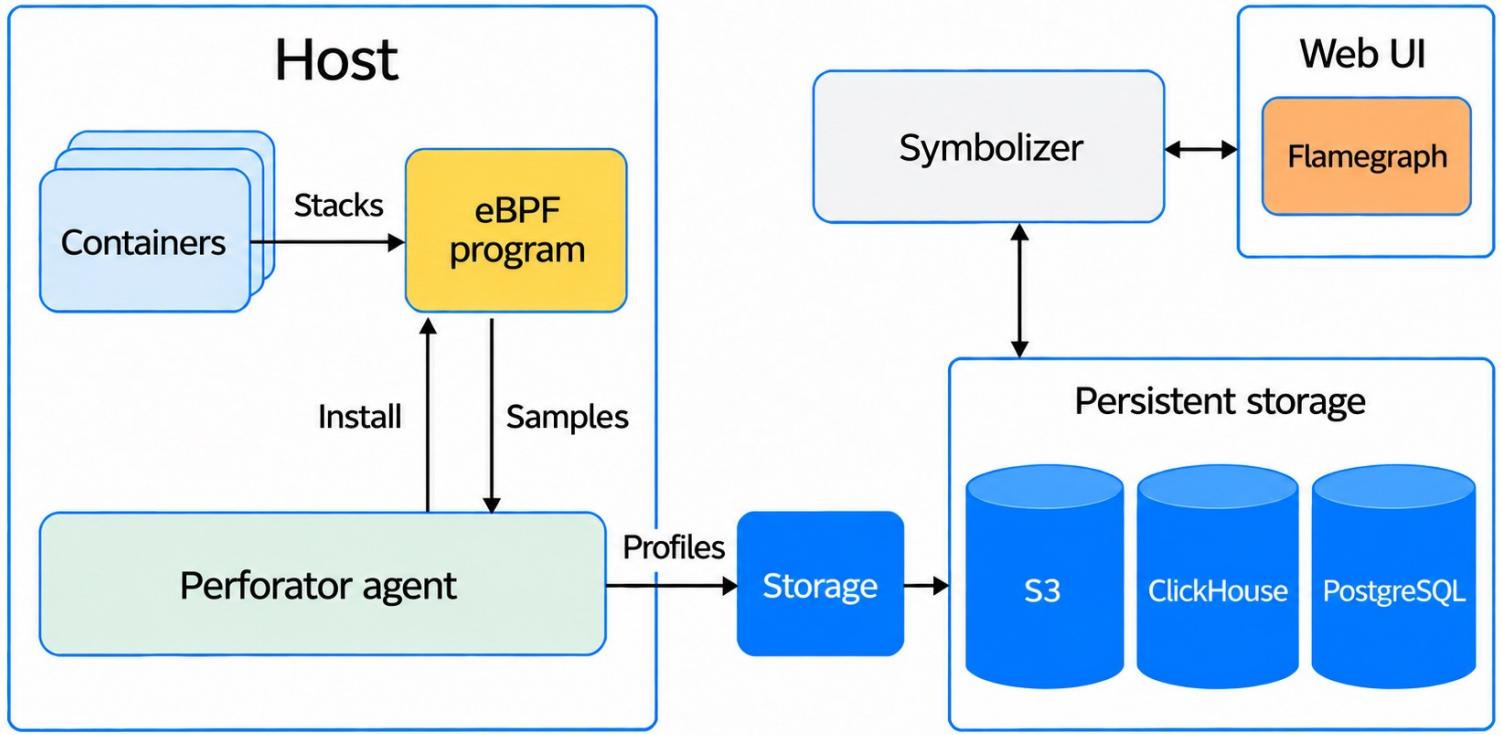
- Postgres
- ClickHouse
- S3



Perforator

Архитектура

Поднятие и эксплуатация такой инфраструктуры требуют времени и сил



Скачали, подняли и готово?

Адаптируем под себя

Поднимаем

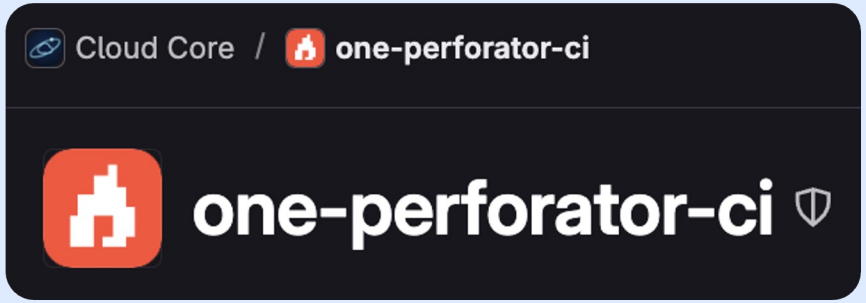
Проблема

Нужно научить подниматься в специфичном Облаке. Из коробки только Kubernetes

Итог

Свой CI/CD с учетом нашей специфики:

- свои Docker-образы
- специфичные systemd-юниты (доставка конфигураций, отгрузка логов, ...)
- свой формат манифестов



Адаптируем под себя

Cgroups

Проблема

Обнаружение контейнеров и их cgroups было завязано на Kubernetes



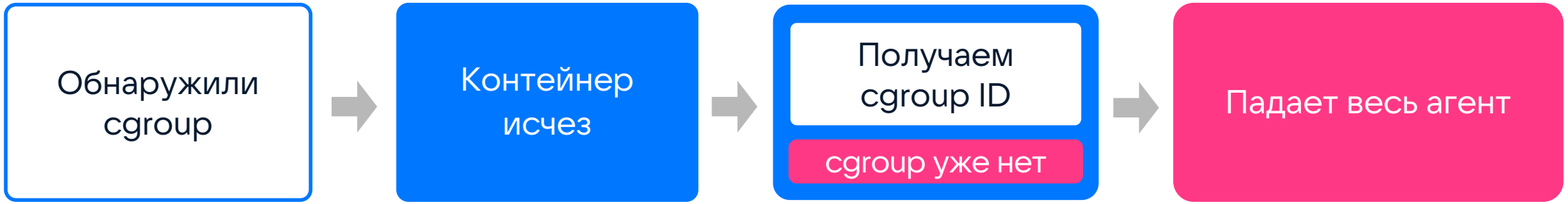
Итог

Добавили поддержку нашего формата cgroups

Адаптируем под себя

Cgroups

Проблема



Итог

Научили пропускать такие случаи

Адаптируем под себя

CPU

Проблема

Агент не видел
всех CPU на хосте

Итог

Завязали его на *possible*,
вместо *online*

```
1.cdb.abc-sync.hc.one-infra.ru: /# cat /sys/devices/system/cpu/online
0-1
1.cdb.abc-sync.hc.one-infra.ru: /# cat /sys/devices/system/cpu/possible
0-111
```

Адаптируем под себя

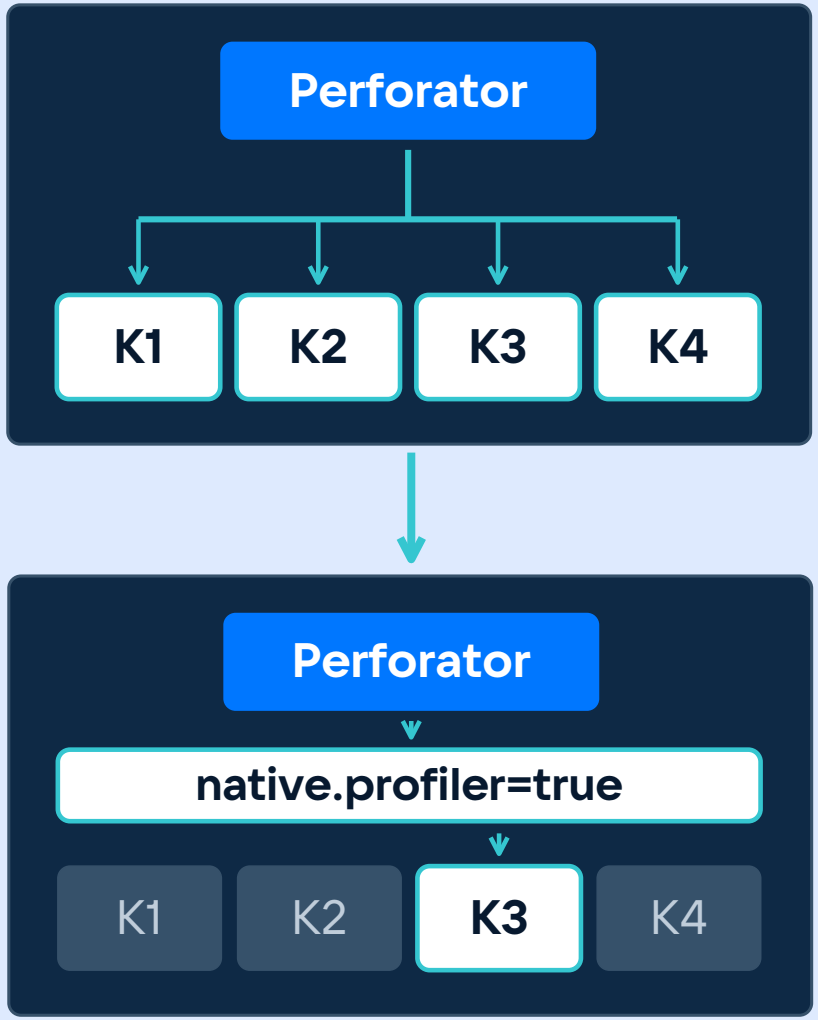
Профилирует всё, что видит

Проблема

Профилирует все обнаруженные контейнеры. Дорого

Итог

Сделали настройку, чтобы включать on demand



Адаптируем под себя

Ключи в S3

Проблема

Профили и бинари складывались в корень бакета, что приводило к плохой балансировке нагрузки в S3

Итог

Разбиваем UUID профилей и build ID бинарей на части

Было

```
019a1234-5678-7abc-8def-0123456789ab
```

Стало

```
019a/1234/5678/7abc/8def/0123/4567/89ab
```

Адаптируем под себя

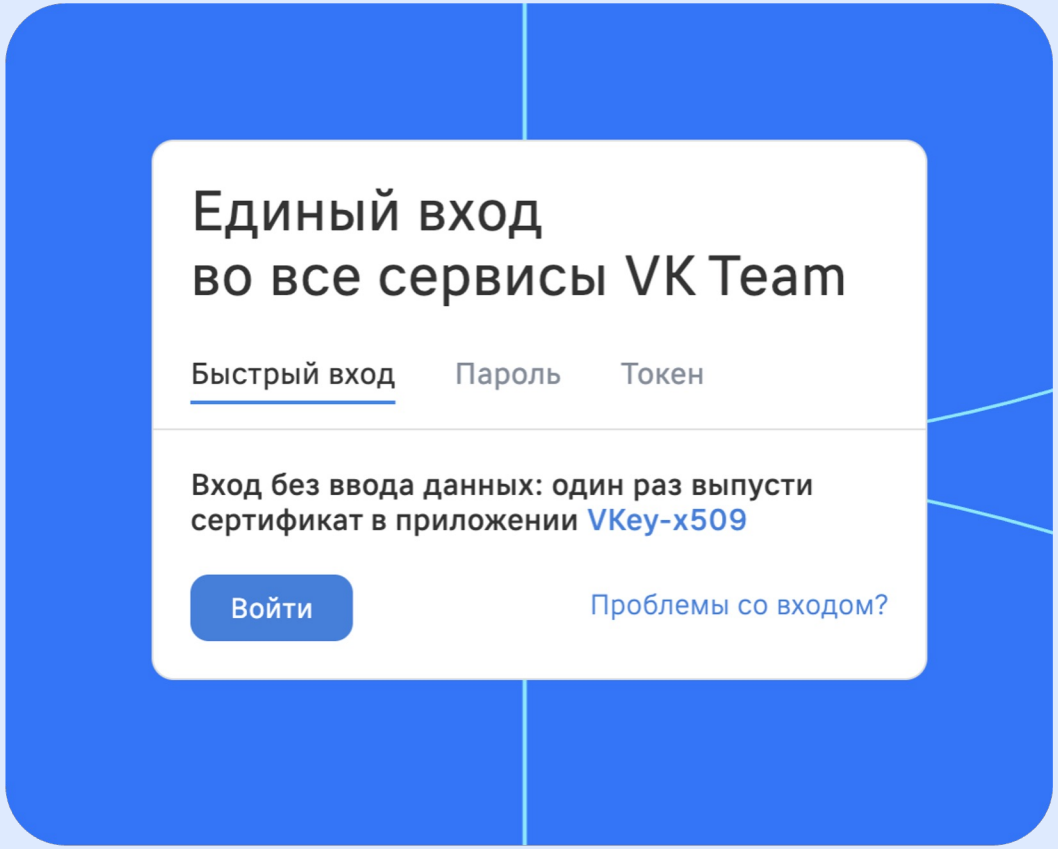
Авторизация

Проблема

Нужна была корпоративная авторизация

Итог

Добавили OIDC в сервис Web и пофиксили проблемы, возникшие из-за этого



Адаптируем под себя

Итого

Если Kubernetes —
все просто

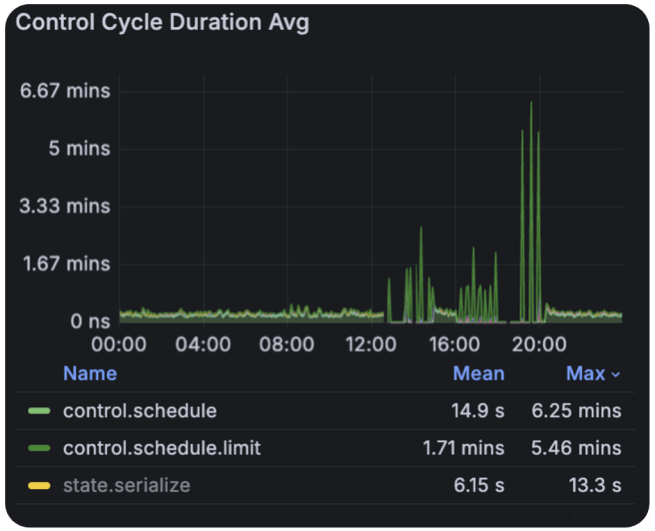
Если свое Облако —
придется покопаться

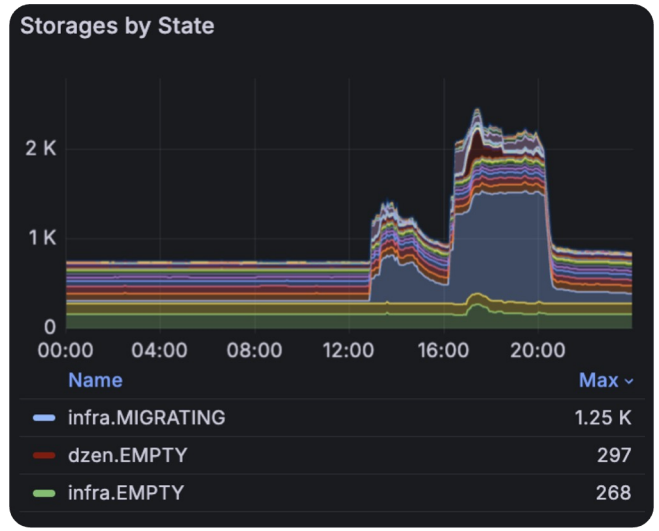
Давайте додебажим инцидент

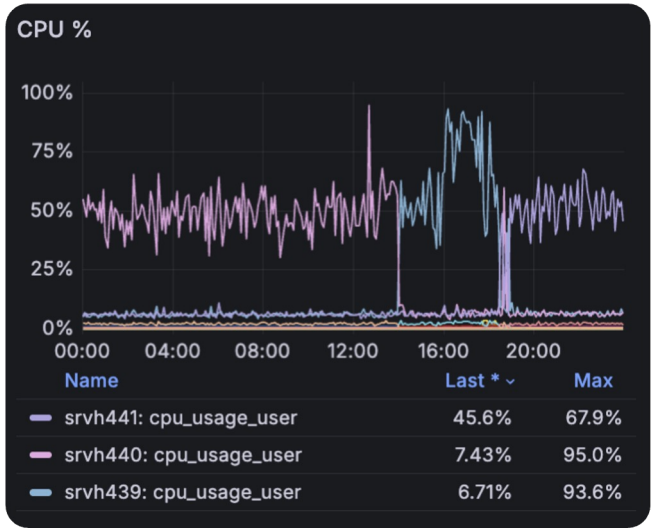
Дебажим

Что делаем

- Видим какие-то проблемы
- Не видим,
  где расходуется CPU
- Не видим root-cause
- В логах ничего
  подозрительного нет
- Проблему уже починили







Дебажим

Точка входа

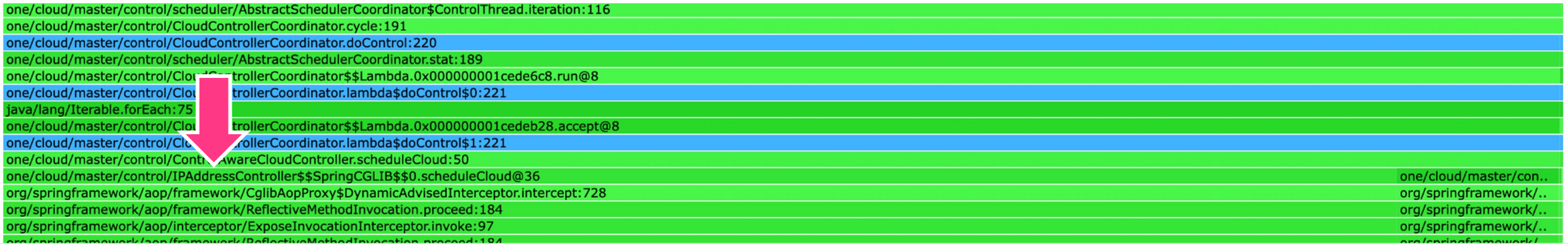
Находим точку входа. У нас это метод doControl



Дебажим

Горячий код

- Видим, что 90% тратится в IPAddressController



Дебажим

Выдача IP

- Все CPU тратим в IPAddressController.getNextUnassignedIps



Дебажим

Линейный поиск

Посмотрим на его реализацию

```
public List<IPAddressED> getNextUnassignedIps(ControlState ctx, IPPool pool, int count) {
    return pool.allIps(count, ip -> isIpNotChangedAndAssignable(ctx, ip, host: null))
            .map(ipAddressesService::getOrCreate)
            .toList();
}
```

- Находим линейный проход по всем IP в подсетях в пуле
- Поиск внутри подсети останавливается, когда находим N свободных IP
- Чем больше занятых IP, тем хуже работает

Дебажим

Причина нагрузки

```
public List<IPAddressED> getNextUnassignedIps(ControlState ctx, IPPool pool, int count) {
    return pool.allIps(count, ip -> isIpNotChangedAndAssignable(ctx, ip, host: null))
            .map(ipAddressesService::getOrCreate)
            .toList();
}
```

- Для миграции потребовалось запустить инфраструктурные сервисы, ответственные за нее. **Выдача IP для них заняла почти весь CPU.**

Дебажим

Быстрая выдача IP

Фикс в процессе

Задача

Динамический MEX

Решение

TreeSet + index

Выдача IP

Выдать минимум из TreeSet,
иначе next++

O(U) → O(log(K))

Возврат IP

Добавить IP в TreeSet

O(log(K))

Разбор инцидента



Проблему нашли, мы молодцы
Историческим профилям спасибо!

Как понять, что оно нам надо?

Когда применять

Когда вам нужно непрерывное профилирование

Появляются систематические проблемы с производительностью

- Долго работают запросы в endpoint /api/example

Когда применять

Когда вам нужно непрерывное профилирование

Появляются систематические проблемы с производительностью

- Долго работают запросы в endpoint /api/example

Не знаете, как воспроизводить

- Долго работает на продовых данных, локально и на тестинге не воспроизводится

Когда применять

Когда вам нужно непрерывное профилирование

Появляются систематические проблемы с производительностью

- Долго работают запросы в endpoint /api/example

Не знаете, как воспроизводить

- Долго работает на продовых данных, локально и на тестинге не воспроизводится

Не успеваете / не хотите на лету снимать профиль

- Инцидент починили, теперь надо разбираться
- Начинается только в 3 ночи, когда все спят

Когда применять

Когда вам нужно непрерывное профилирование

Появляются систематические проблемы с производительностью

- Долго работают запросы в endpoint /api/example

Не знаете, как воспроизводить

- Долго работает на продовых данных, локально и на тестинге не воспроизводится

Не успеваете / не хотите на лету снимать профиль

- Инцидент починили, теперь надо разбираться
- Начинается только в 3 ночи, когда все спят

Польза оправдывает затраты на инфраструктуру

Когда применять

Когда вам не нужно непрерывное профилирование

Когда применять

Когда вам не нужно непрерывное профилирование

Проблема единичная

- Долго работал запрос X, остальные OK
- Долго работает у юзера Y

Когда применять

Когда вам не нужно непрерывное профилирование

Проблема единичная

- Долго работал запрос X, остальные OK
- Долго работает у юзера Y

Ищете проблему уровнем выше, чем конкретный инстанс

- Долго работал запрос, который обрабатывался пятью микросервисами и двумя БД

Когда применять

Когда вам не нужно непрерывное профилирование

Проблема единичная

- Долго работал запрос X, остальные OK
- Долго работает у юзера Y

Ищете проблему уровнем выше, чем конкретный инстанс

- Долго работал запрос, который обрабатывался пятью микросервисами и двумя БД

Проблема постоянная — можно попрофилировать на лету

Итоги

Итоги

one-continuous-profiler

** Все цифры без учета RF=3*

33K

агентов

170

MiB RAM
на агента
(**5.7 TiB**
утилизирован
ной RAM)

146

TiB профилей

18

дней — TTL

Итоги

Perforator

** Все цифры без учета RF=3*

25K
агентов

400
MiB RAM на агента в среднем (**9.3 TiB** утилизированной RAM)

21
TiB профилей

210
GiB бинарей

9
GiB пользовательских результатов

16
GiB GSYM

21
день, TTL

Итоги

Финальные тезисы

Профилирование помогает быстро и точно разобраться с систематическими проблемами в производительности

```
public List<IPAddressED> getNextUnassignedIps(ControlState ctx, IPPool pool, int count) {
    return pool.allIps(count, ip -> isIpNotChangedAndAssignable(ctx, ip, host: null))
            .map(ipAddressesService::getOrCreate)
            .toList();
}
```

Итоги

Финальные тезисы

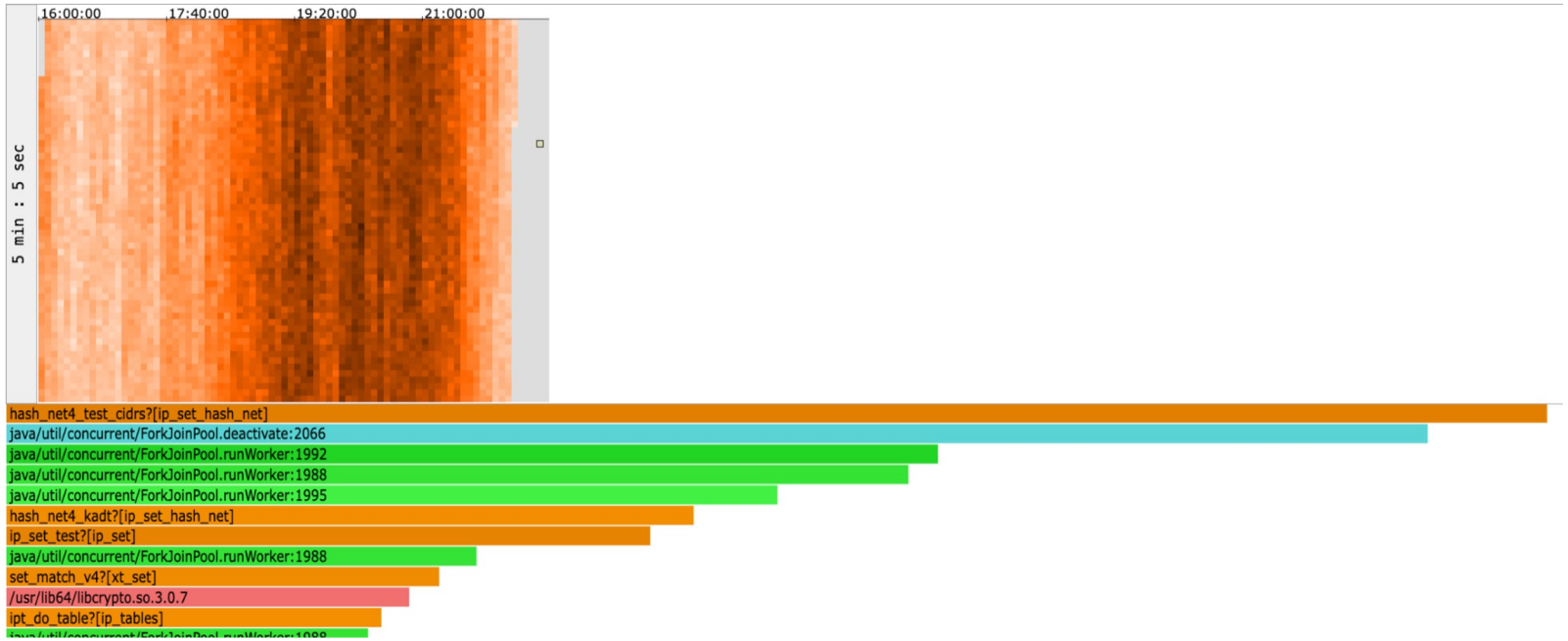
Непрерывное профилирование помогает расследовать, что произошло в прошлом, и отлавливать неожиданные проблемы

```
public List<IPAddressED> getNextUnassignedIps(ControlState ctx, IPPool pool, int count) {
    return pool.allIps(count, ip -> isIpNotChangedAndAssignable(ctx, ip, host: null))
            .map(ipAddressesService::getOrCreate)
            .toList();
}
```

Итоги

Финальные тезисы

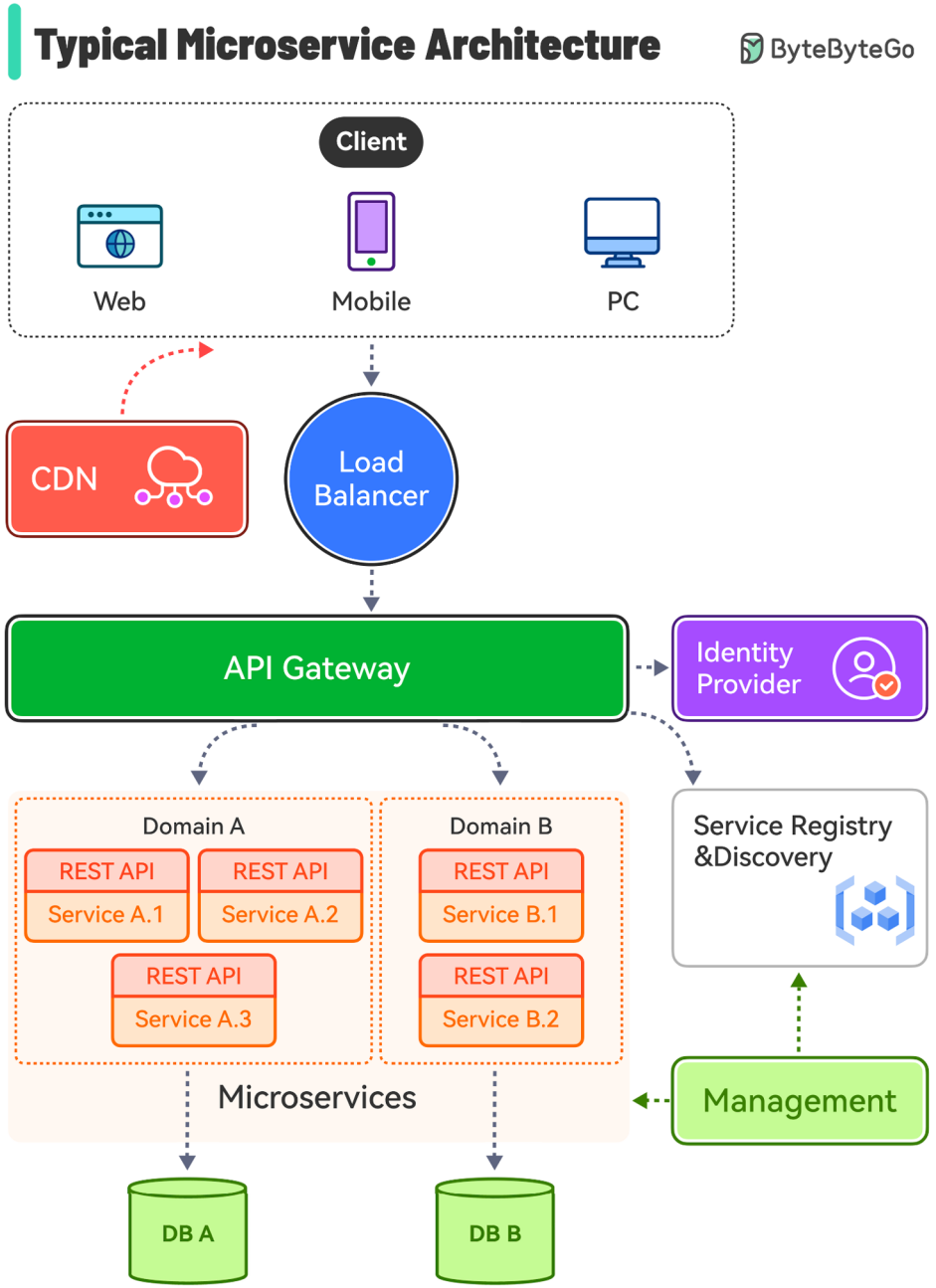
Иногда недостаточно видеть в профилях только код вашего сервиса



Итоги

Финальные тезисы

Непрерывное профилирование — не серебряная пуля



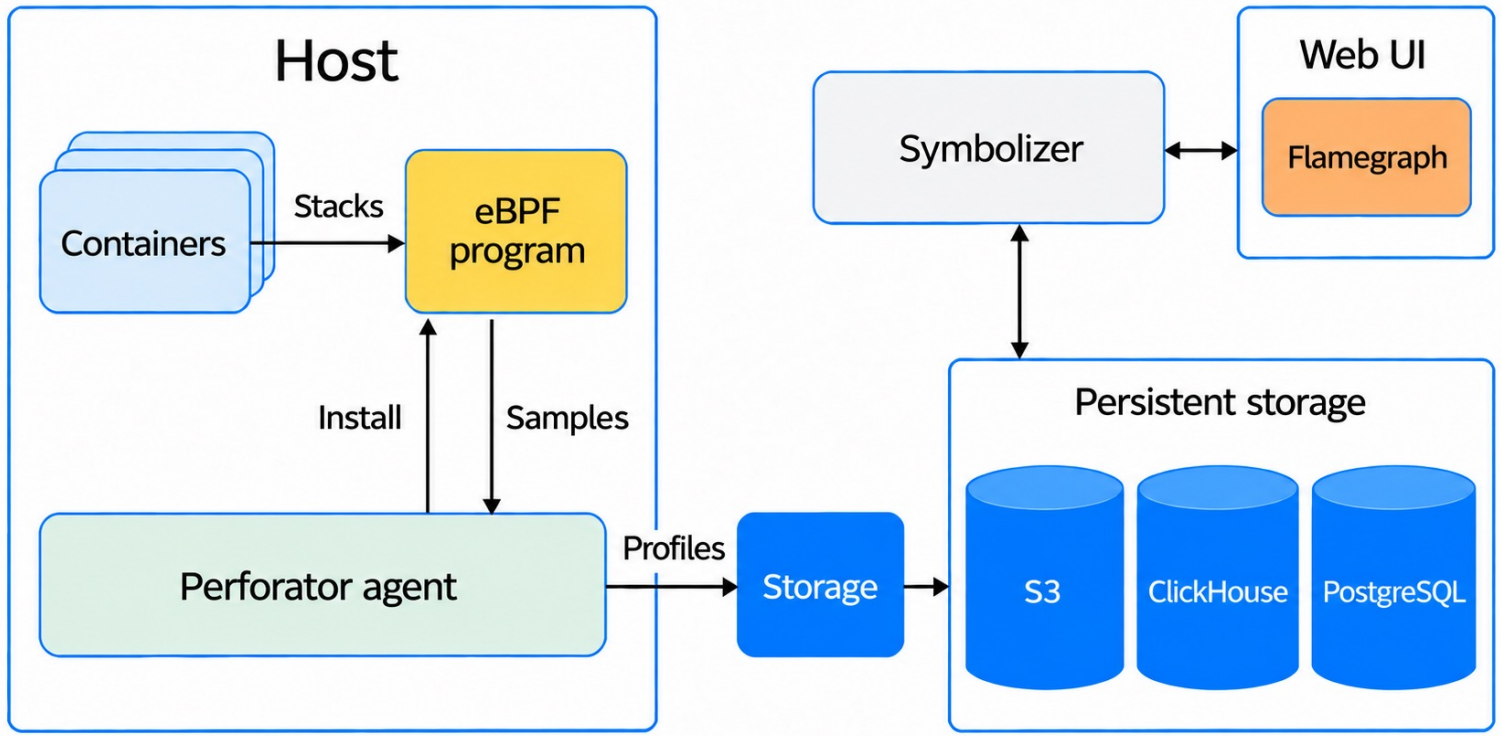
https://bytebytego.com/guides/what-does-a-typical-microservice-architecture-look-like/

Итоги

Финальные тезисы

Непрерывное профилирование — не бесплатно

Платим как железом, так и человеко-часами на развертывание, адаптацию и поддержку

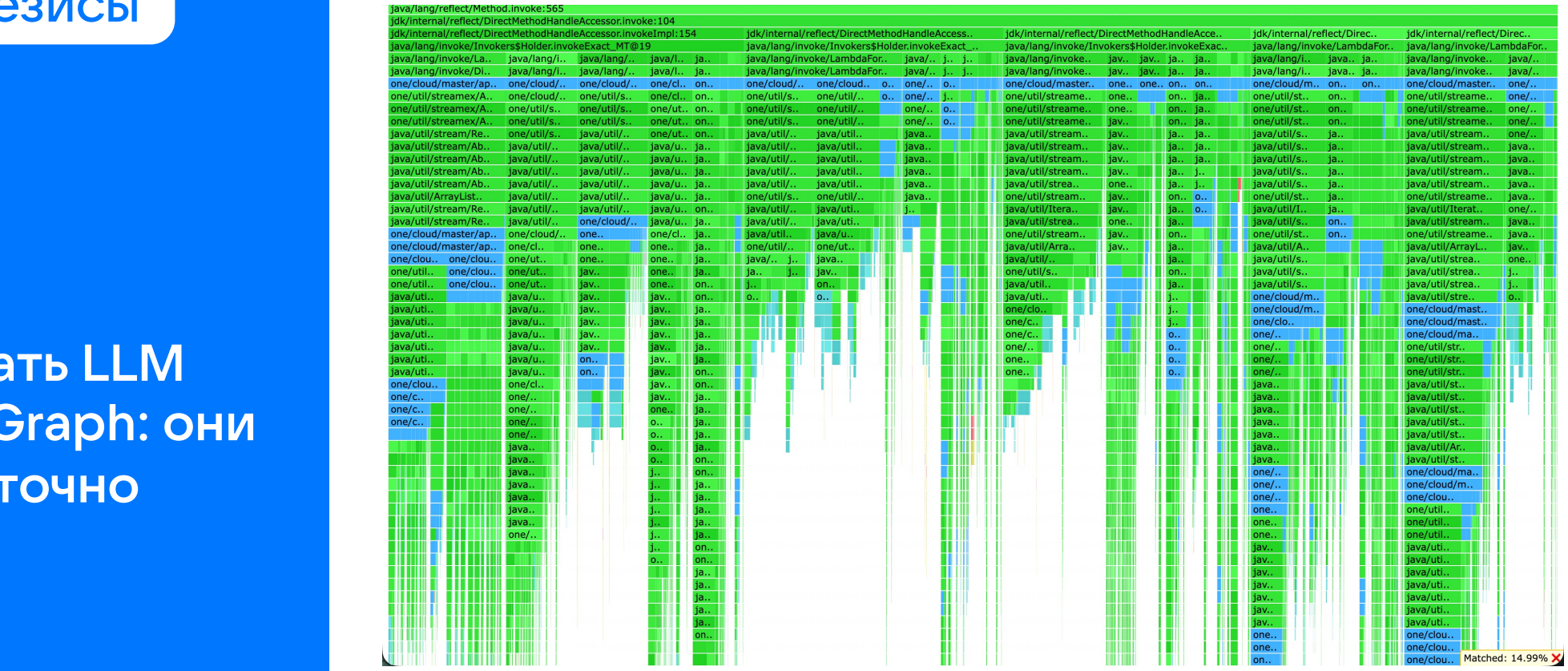


Итоги

Финальные тезисы

Ах да, AI/LLM!

Удобно отдавать LLM анализ Flame Graph: они бывают достаточно громоздкие



Спасибо за внимание!

Егор Леванков
Группа Core разработки внутреннего Облака VK — One Cloud

