

Оптимизация производительности в играх



Кто влияет на перф игры



Программисты

Алгоритмы
Память
Архитектура
Оптимизация



Дизайнеры контента

Уровней
Квестов
NPC



Художники

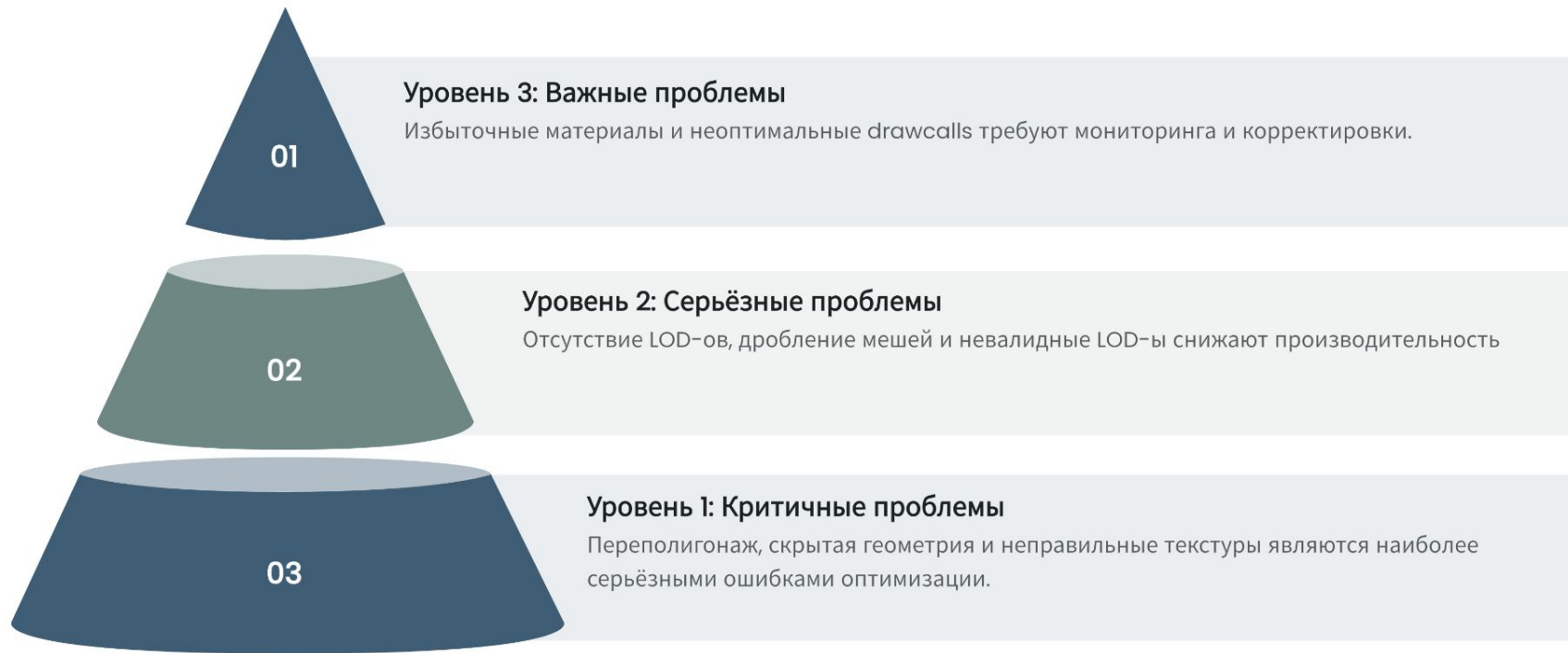
Моделей
Текстур
Анимаций
Визуальных эффектов



QA

Координация баланса всех
Контроль качества

Категории оптимизации



Переполгонаж

Основная проблема

Избыточное количество полигонов в моделях, превышающее необходимость для визуального качества и производительности системы.



Отсутствие LOD-систем

Нет или неправильная реализация LOD-систем для автоматического снижения качества моделей на значительном расстоянии от камеры.



Геометрическая плотность

Слишком высокая плотность геометрии на объектах, занимающих место на экране всего двадцать пикселей.



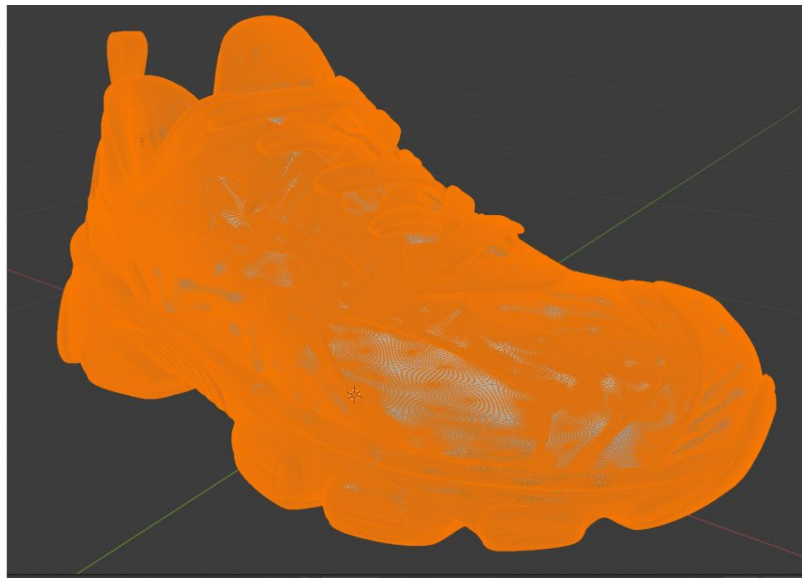
Скрытые части

Модель содержит миллион полигонов из которых видно максимум десять тысяч.

Переполлигонаж

Например вот такой моделью ботинка, и не только его очень легко накосячили PUBG на релиза.

Здесь 172 тысячи полигонов, выглядит классно, не спору



Скрытая геометрия



Cities Skylines 2.

Каждая модель жителя в городе имела скрытую модель зубов на 420 полигонов, из-за того, что полигонов было меньше 500, они не попадали под автоматический лодинг и постоянно рисовались в кадре

Дробление мешей



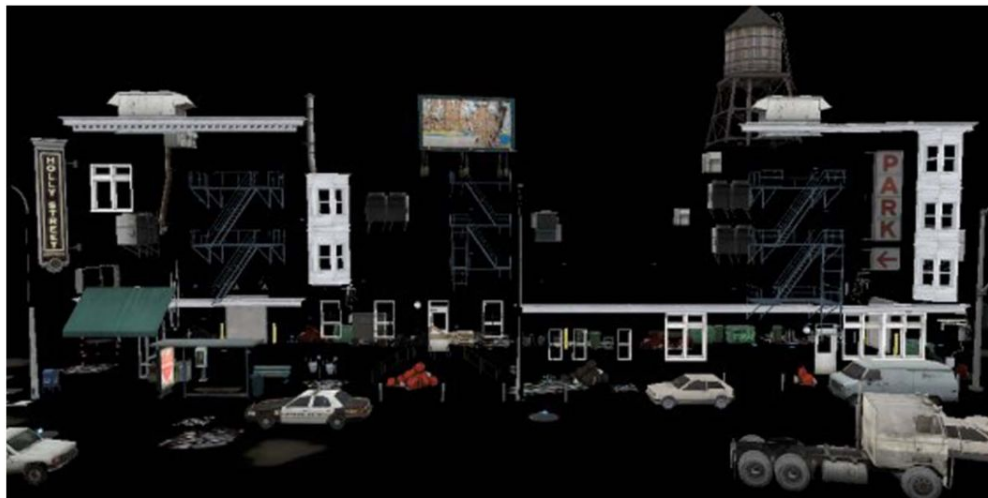
Left4dead

Вся сцена статичная и должна была
рисоваться одним вызовом отрисовки

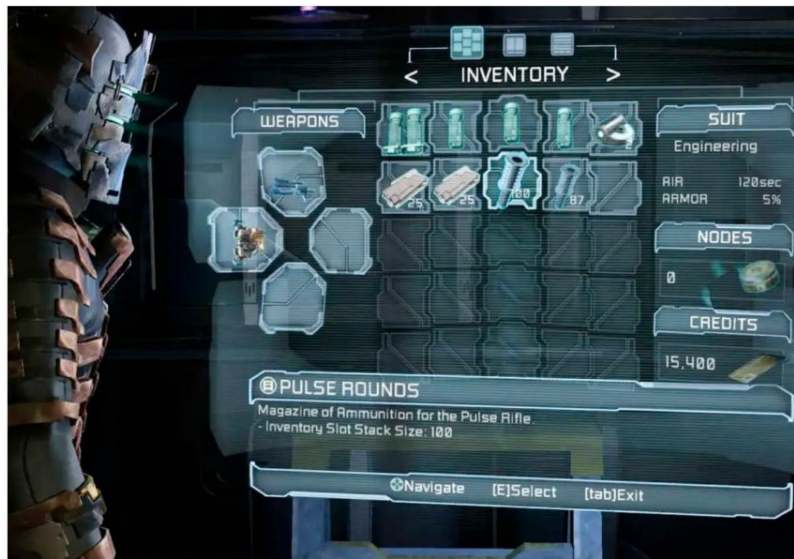
Дробление мешей

Но из-за ошибок при сборке текстур она попадали в разные ассеты, что приводило к отдельным вызовам для отрисовки каждого объекта.

Тут у нас каждый объект это несколько drawcall



Неподходящие текстуры



Dead Space

Текстуры для UI инвентаря были выполнены в 4к отдельными текстурами для каждого виджета, подсчитайте тут сколько у нас элементов и сколько надо было загрузить в память, просто чтобы отрисовать инвентарь

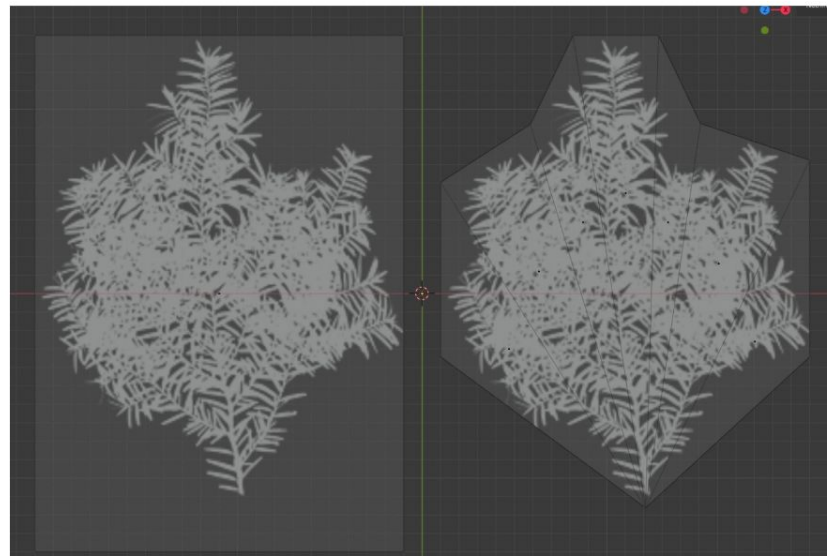
Схожие материалы



Assasins Creed: Origins

Текстура на двух разных деревьях, видимо делали разные команды, и таких косяков набирается очень много по всей игре.

Патч первого дня убирал 8Гб одинаковых материалов в ассетах



Ошибки геометрии и текстур

Скрытая геометрия

Вложенные меши, геометрия под землёй, внутри зданий и забытые проху-объекты.

Дробление мешей

Один видимый объект генерирует десятки drawcall-ов

Несоразмерные текстуры

4K-текстуры на мелких объектах

Материальная избыточность

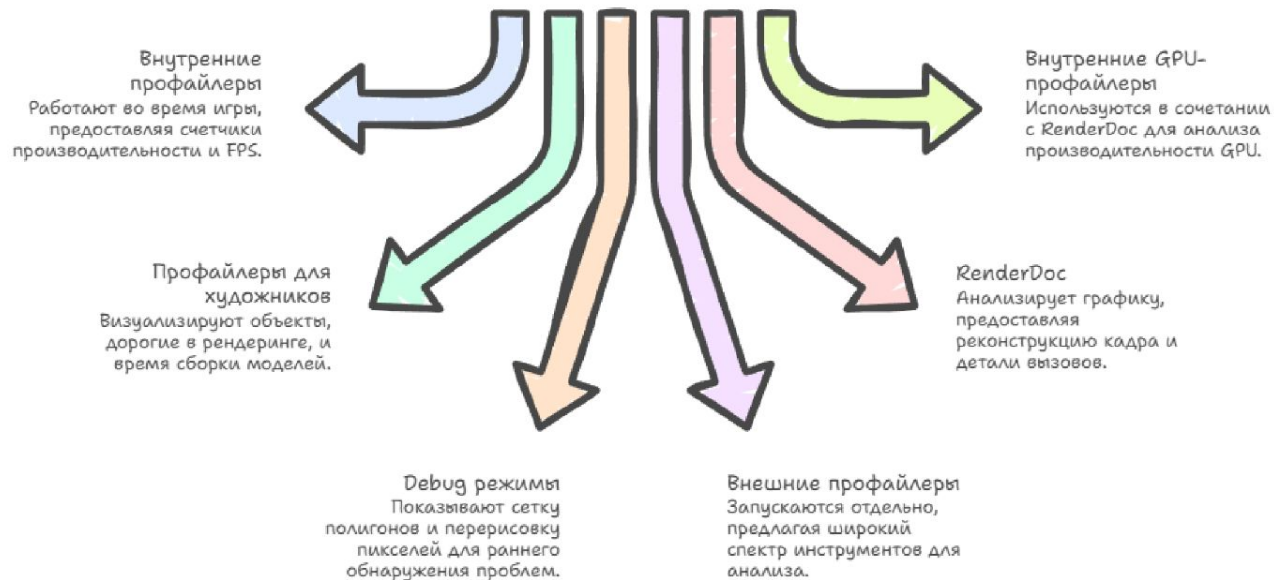
Уникальные материалы, но визуально неотличимы, процедурные материалы.



Косят все



Какой профайлер использовать для оптимизации игры?



Внешние профайлеры



Внешние профайлеры, которые запускаются отдельно от игры и подключаются к ней для сбора данных, и они всем хорошо знакомы (Tracy, Pix, Razor, VTune, Optick, Dr, RGP, IGPA, ASP, Systrace, Perfetto).

Tracy

Tracy — де-факто стандарт среди инди и АА разработчиков.

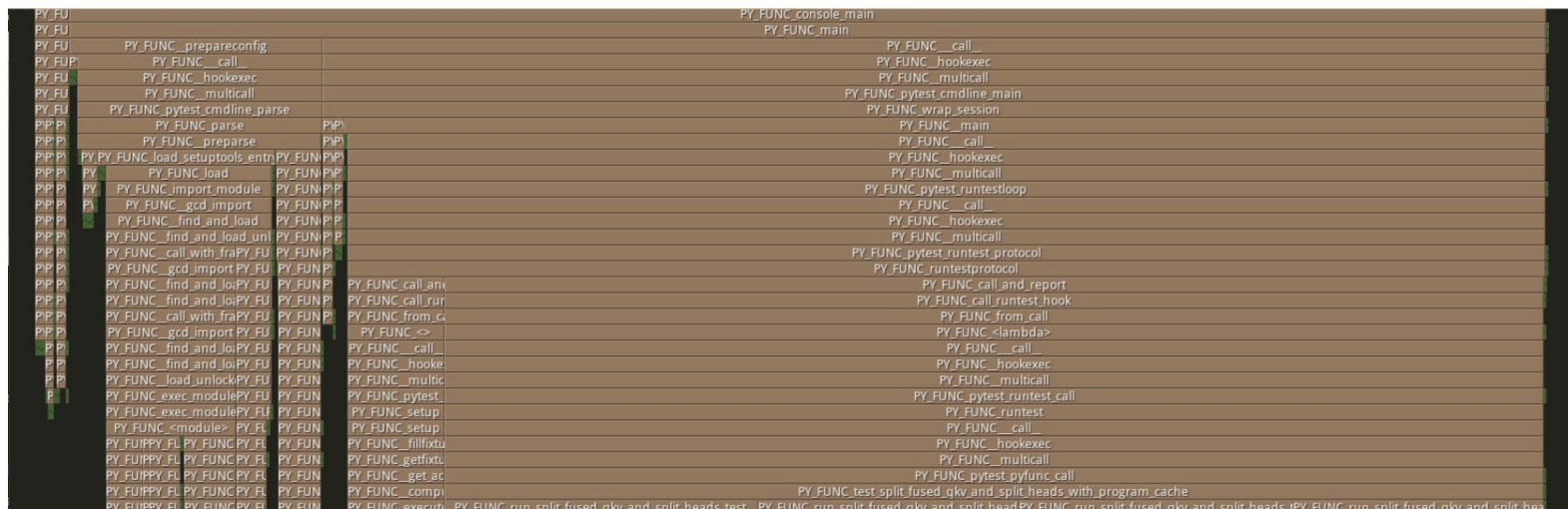
Открытый исходный код

Наносекундное разрешение

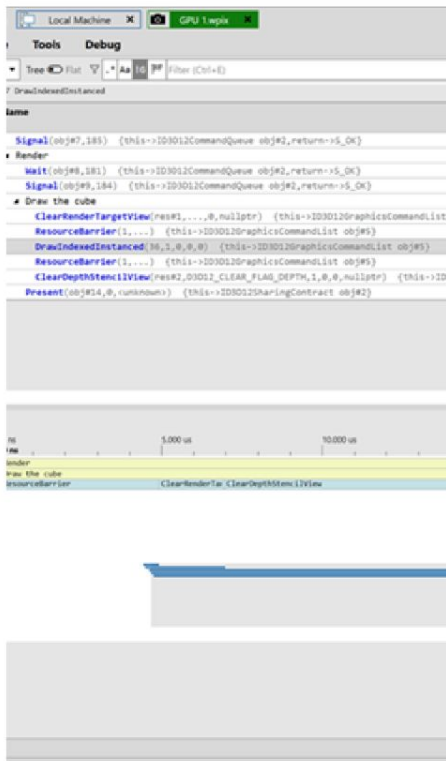
Поддержка CPU, GPU (OpenGL, Vulkan, D3D11/12, Metal)

Особенно хорош для поиска worst-case фреймов: находим самый медленный кадр и дебажим его.

Интегрирован во многие движки, включая Flax и Godot.



PIX



Главный инструмент для разработки под Xbox.

Живёт в нише GPU-профилирования:

захват одного фрейма

разбор draw calls

таймлайн GPU-очередей

инспекция ресурсов и шейдеров прямо в интерфейсе.

Незаменим при разработке под DirectX 12, и только PIX умеет показывать барьеры ресурсов, синхронизацию между очередями.

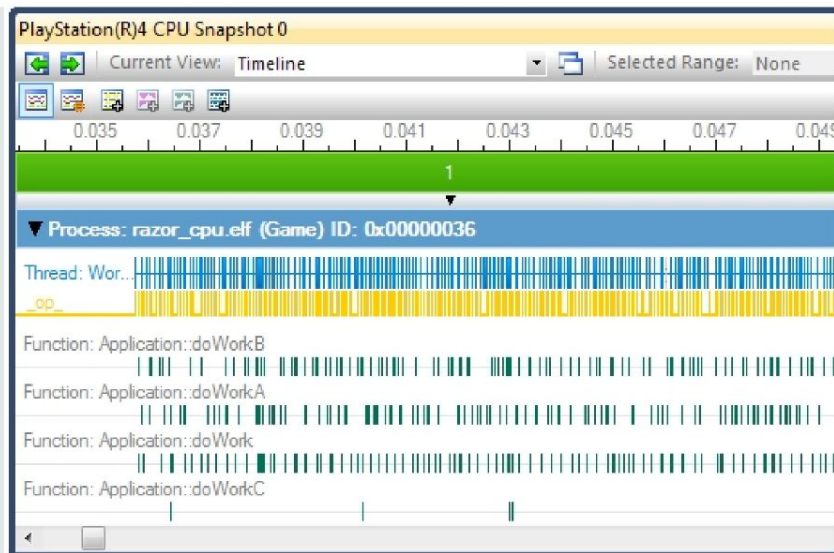
Из минусов — работает только на Windows и только с DX.

Razor

Внутренний инструмент Sony для разработки на PlayStation

Если вы работаете с PS4/PS5, то это ваш единственный полноценный вариант для GPU-профилирования на этой платформе.

Закрытый, доступен только зарегистрированным разработчикам. Тем, кто на консолях никогда не работал, стоит знать, что существует хотя бы теоретически.



VTune

Function Stack	CPU Time: Total	
	Effective Time	Spin Time
▼ Total	45.6%	54.4%
▼ RtlUserThreadStart	45.6%	54.4%
▼ BaseThreadInitThunk	45.6%	54.4%
▼ __scrt_common_main_seh	45.6%	54.4%
▼ WinMain	45.6%	54.4%
▼ GameMain	45.6%	54.4%
▼ GeApplicationImpl: MainLoop	45.1%	54.4%
▼ GeApplicationImpl: Update	39.2%	49.7%
▶ XYGameModule: Update	26.8%	4.7%
▶ QSEngineMgr: Update	8.7%	3.4%
▶ GeSceneModuleImpl: Update	2.2%	0.4%
▶ GeSceneModuleImpl: PostUpdate	0.5%	0.4%
▶ ReliefThread: WaitManMngSortDone	0.2%	0.0%
▼ QSEngineMgr: APEXUpdate	0.2%	39.2%
▼ QSAPEXManager: APEXUpdate	0.2%	39.2%
▼ QSApexScene: Simulate	0.2%	39.2%
▼ func@0x180020990	0.2%	39.2%
▶ func@0x1800f0d80	0.1%	0.0%
▶ func@0x1800f0ee0	0.0%	0.0%
▶ func@0x180070980	0.0%	0.0%
▼ physx: shdFind: Sync.wait	0.0%	39.1%
▶ WaitForSingleObject	0.0%	39.1%
▶ physx: shdFind: MutexImpl: unlock	0.0%	0.1%
▶ ReliefThread: StartWork	0.1%	0.0%

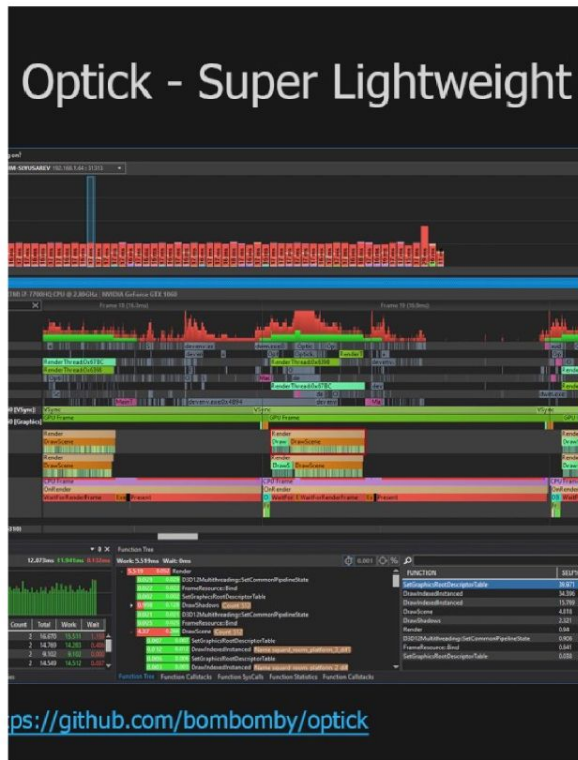
CPU-профилировщик от Intel, ориентированный в первую очередь на глубокий анализ производительности:

- многопоточность
- SIMD-эффективность
- кэш-промахи
- branch mispredictions

Для игр его используют там, где Trасу или Optick показали "тут медленно". Есть официальная интеграция с Unreal Engine. Из минусов – тяжёлый, громоздкий интерфейс, и очень высокий порог вхождения.

Optick

Optick - Super Lightweight



Лёгкий профайлер специально для игр, духовный наследник Brofiler.

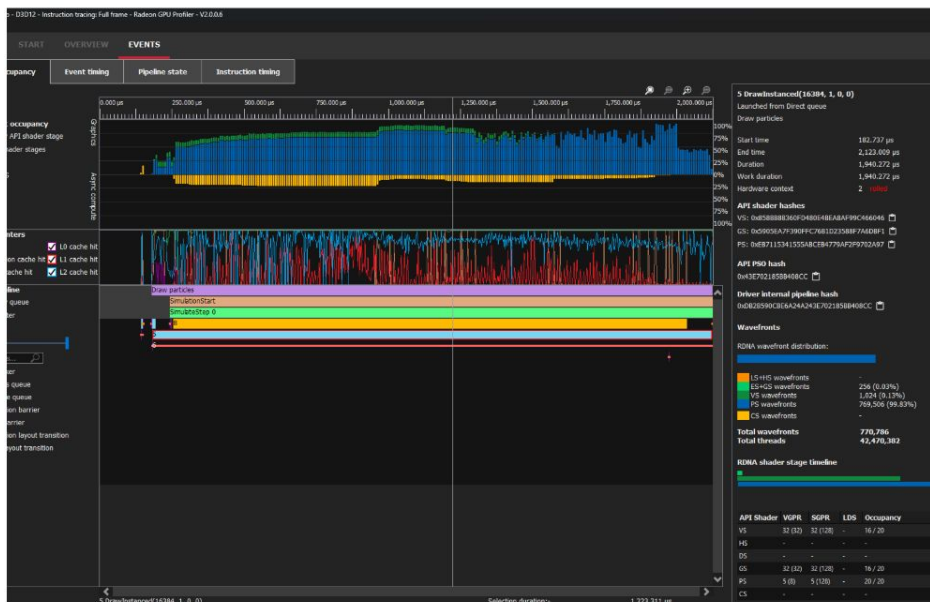
Позиционируется как минималистичная альтернатива Tracy: меньше возможностей, но быстрее интегрируется и меньше overhead на трассировку.

Поддерживает инструментацию, context switches и GPU-счётчики.

Встроен в движок Unreal Engine 4 как опциональный бэкэнд.

Последние три года не обновляется.

Radeon GPU Profiler

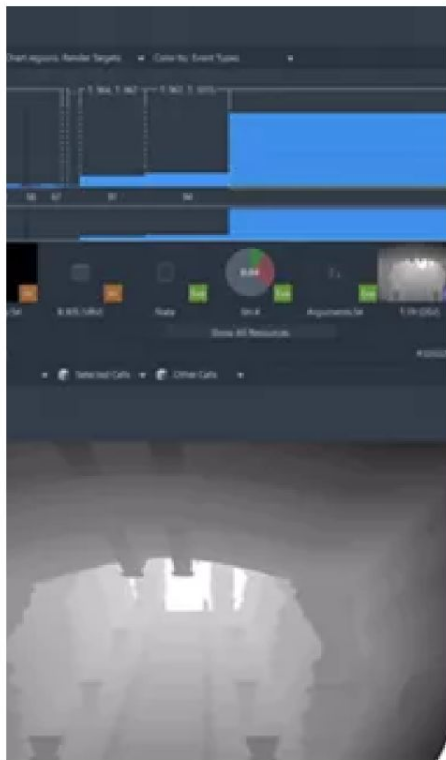


AMD-ответ на PIX, часть экосистемы GPUOpen.

- Показывает низкоуровневую картину работы GPU на картах Radeon:
- заполненность шейдерных процессоров
- сложность шейдеров
- барьеры
- синхронизацию
- и т.д.

Особенно ценен при профилировании Vulkan-приложений, где PIX недоступен.

Intel GPA



Набор инструментов от Intel для анализа графической нагрузки.

Ориентирован больше на художников и графических программистов:

- захват фрейма

- анализ draw calls

- просмотр текстур

- сложность шейдеров

- поддерживает D3D и Vulkan.

Есть плагин для Unreal Engine, который позволяет захватывать фреймы прямо из редактора.

Arm Streamline Performance Advisor



Часть пакета Arm Streamline, ориентированная на Android и мобильные платформы с Arm GPU.

Позволяет мониторить приложение и генерировать performance-отчёты.

Незаменим, если вы оптимизируете под мобильные видеокарты.

Systrace / Perfetto



Инструменты Google для профилирования Android-систем целиком.

Systrace — старший, Perfetto — его современная замена с более богатым интерфейсом и поддержкой длительных трейсов.

Показывают всю систему вокруг приложения.

Методы профилирования

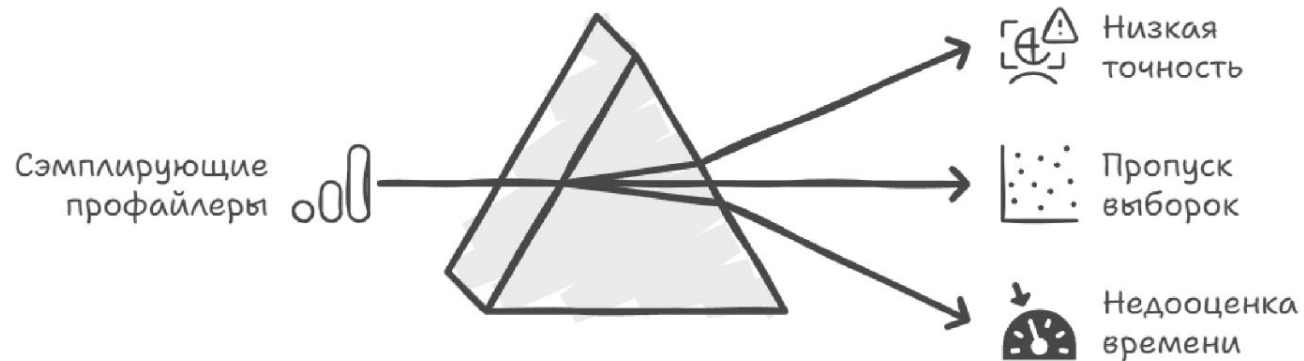


Сэмплирование



Данные о работе приложения собираются через регулярные промежутки времени (миллисекунду) профайлер останавливает программу и смотрит, какая функция сейчас выполняется, записывает стектрейс и после накопления достаточного количества сэмплов строит статистическую картину где программа провела больше всего времени.

Недостатки сэмплирования



Проблема быстрых функций

Время: 0 100 200 300 400 500 600 700 800 ns



[———— process_particles ————][other_work]
L 500 ns L 300ns L

Внутри [f][f][f][f][f]...[f][f][f] (5000 вызовов)

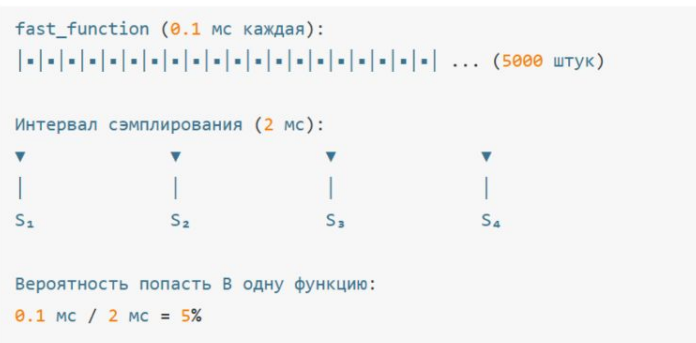
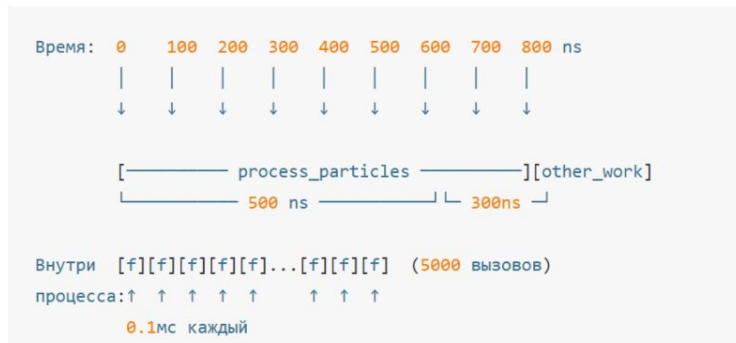
процесса: ↑ ↑ ↑ ↑ ↑ ↑ ↑

0.1мс каждый

Проблема быстрых функций

Занимают время, но не видны при сэмплировании

Можно пропустить реальные проблемы

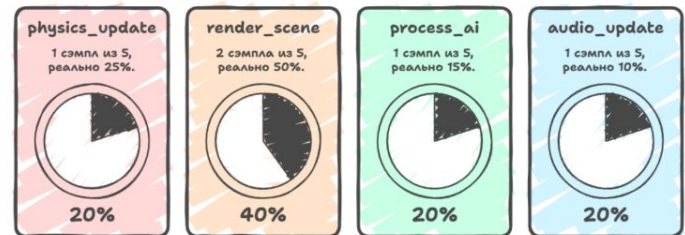
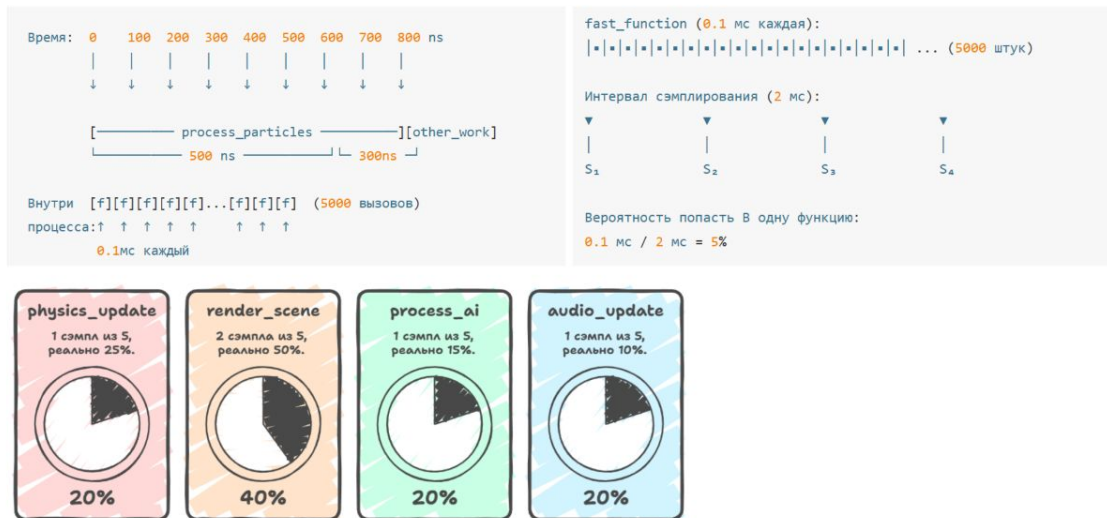


Проблема быстрых функций

Занимают время, но не видны при сэмплинговании

Можно пропустить реальные проблемы

Нашей fast_function нет в статистике



Инструментирование

Оптимизировать
код

Внести изменения
для улучшения
производительности.



4

метки

Вставить метки в
код для
измерения
времени.



1

Анализировать
данные

Проанализировать
записанные
данные для
выявления узких



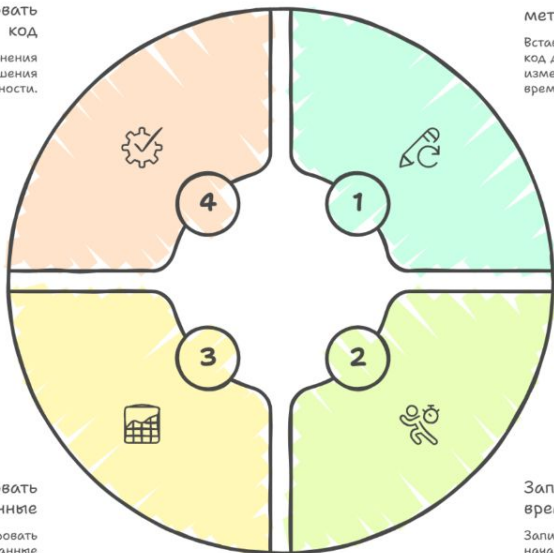
3

Записать
время

Записать время
начала и конца
выполнения.



2

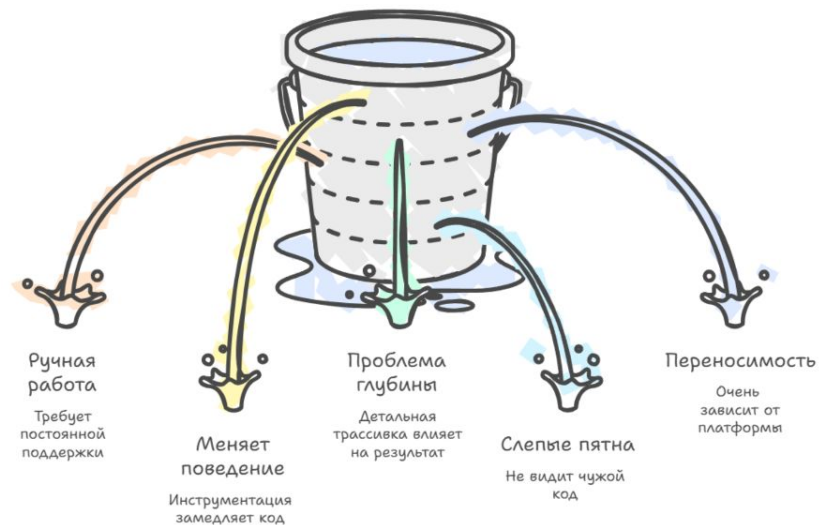


Надо вручную расставлять в коде специальные метки или макросы, чтобы записывать точное время начала и конца выполнения каждой важной функции или блока кода

Многопоточность + инструментирование

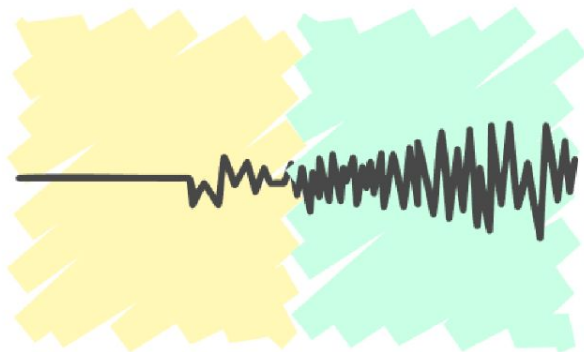


Недостатки инструментирования



Трассировка

Агрегированный ↔ Хронологический



Сэмплирование

Агрегирует
данные, теряет
хронологию

Трассировка

Сохраняет
хронологию
событий во
времени

Собирается поток событий во времени:

начало/конец задач

блокировки

ожидания

сообщения между потоками

очереди задачи

сохраняет хронологию

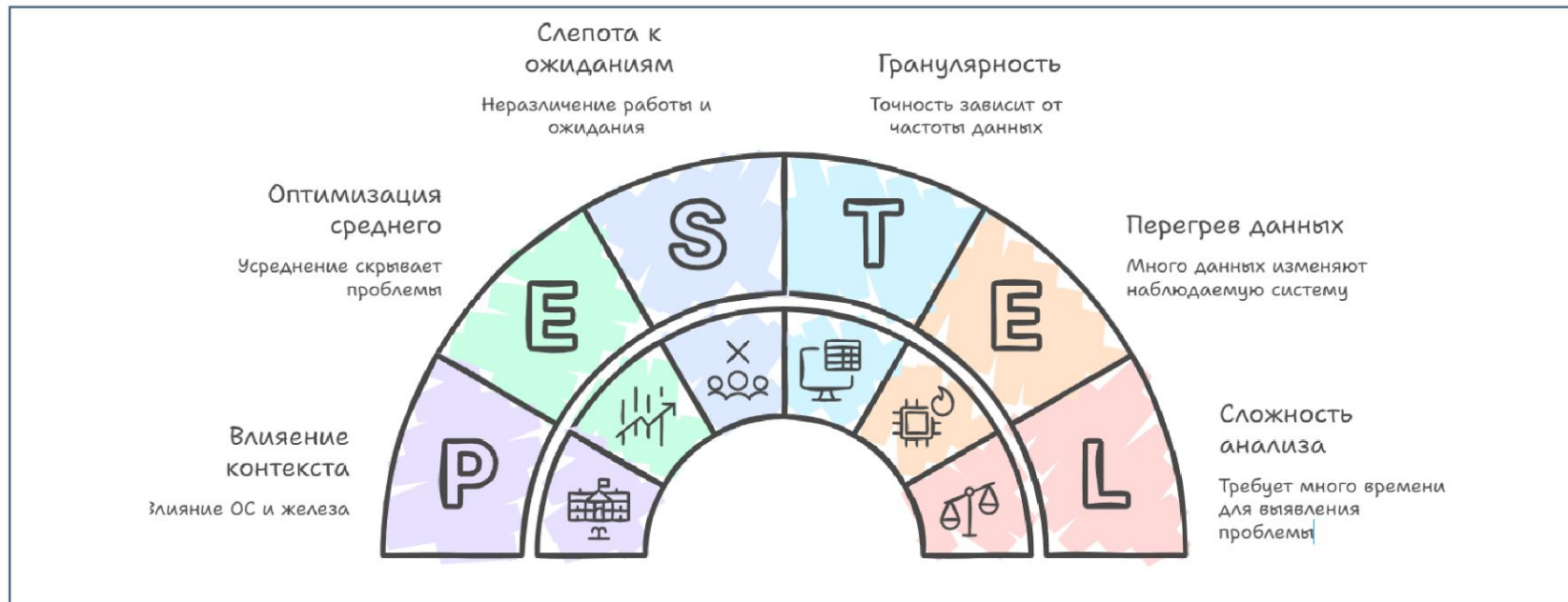
Хронология событий является ключевым моментом для

понимания поведения разных систем, task-графов и асинхронных пайплайнов.

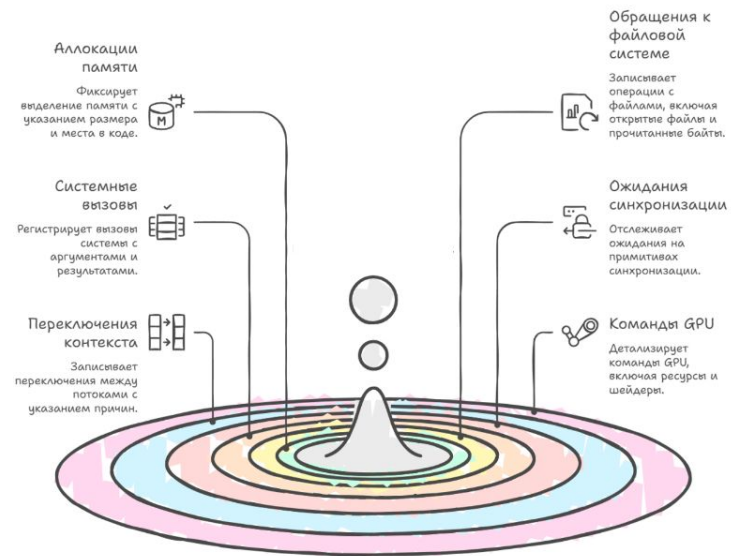
Недостатки трассировки



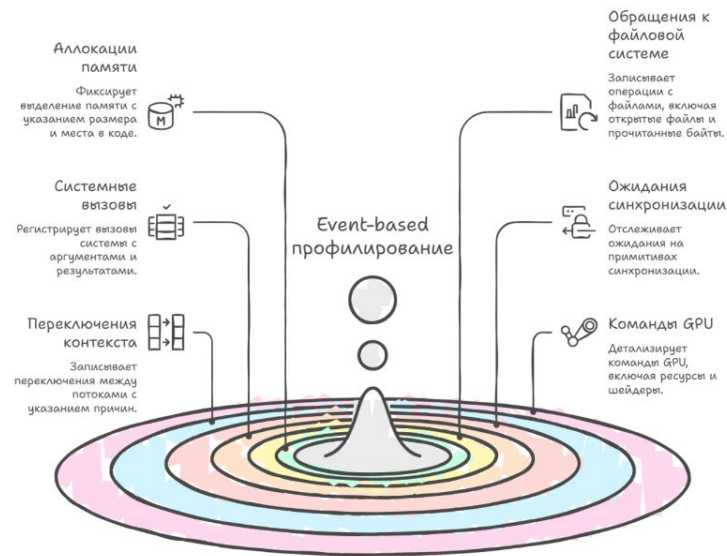
Недостатки классического профилирования



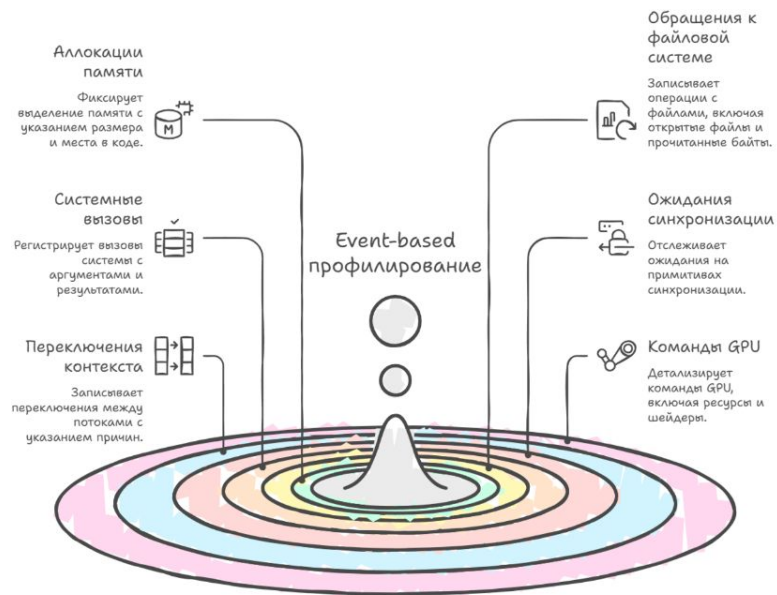
Логирование



Логирование → Event-based профилирование



Event-based профилирование



Старейший инструмент диагностики

Работает везде и всегда

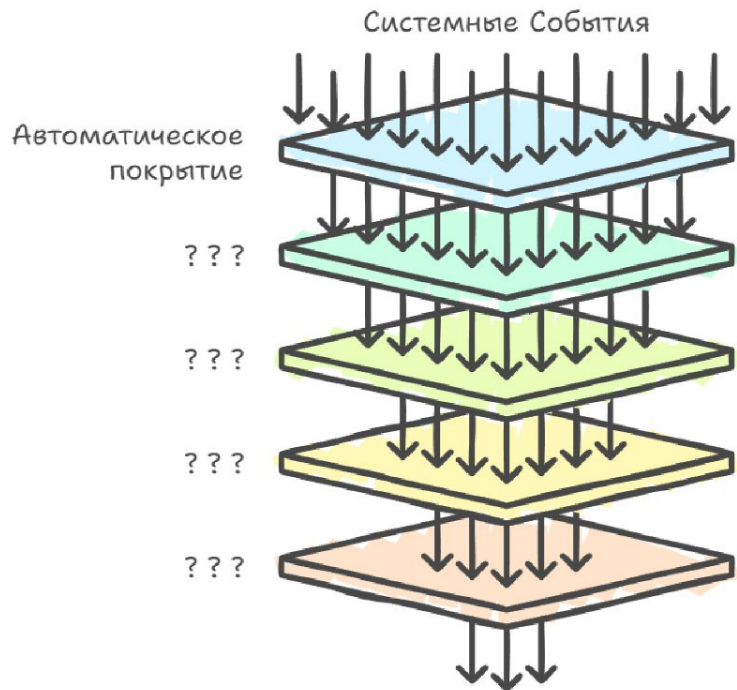
Фиксирует бизнес-логику

Минимальный порог входа

Асинхронность

Сохраняет контекст

Автоматическое покрытие



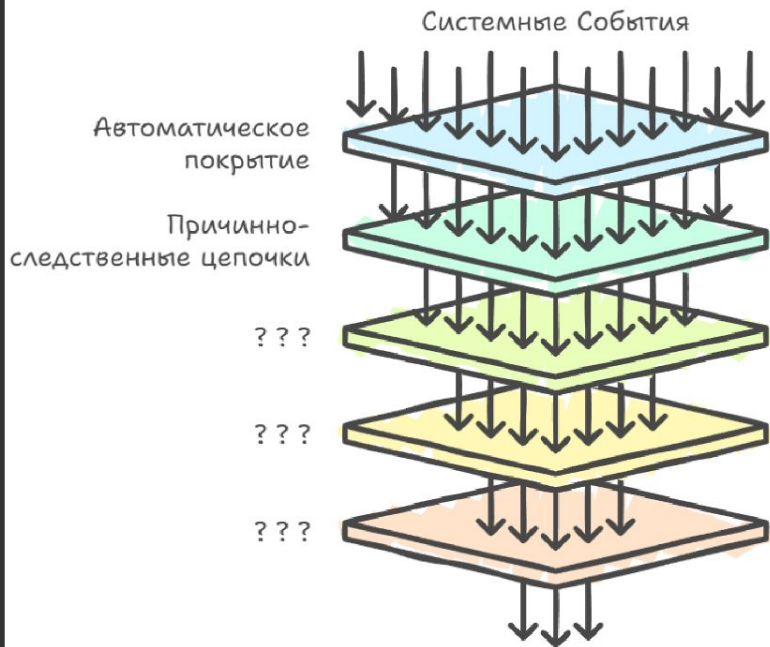
ETW на Windows, perf на Linux, Instruments на macOS
перехватывают системные события через хуки ОС и драйверов:

- аллокации
- системные вызовы
- context switches
- page faults
- прерывания

!!! без единой строки instrumentation в коде

Подключаемся к уже работающей программе и видим полную картину, включая сторонние библиотеки и middleware, которые никогда не были размечены.

Цепочки событий



Показывает не просто "функция X была медленной", а всю цепочку событий вокруг:

за 50 микросекунд до замедления произошёл context switch

поток был вытеснен планировщиком

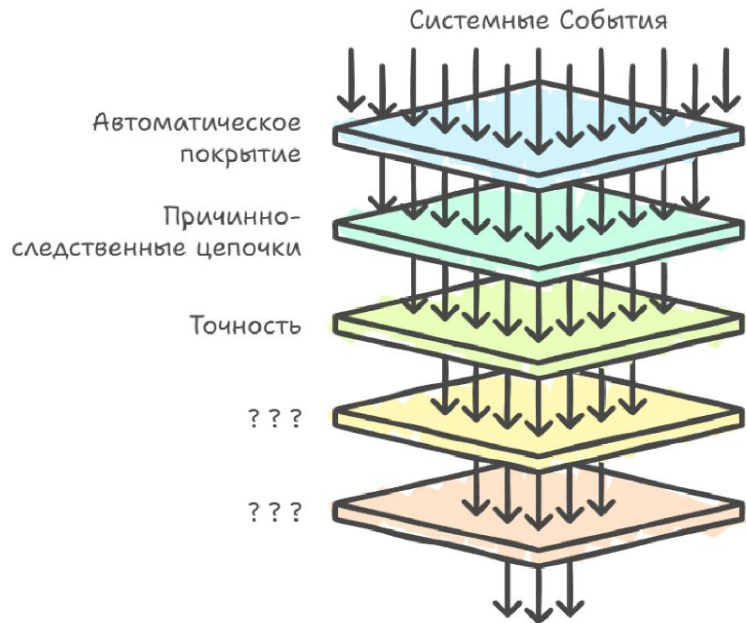
за это время другой поток освободил мьютекс

затем ваш поток возобновился

словил page fault при обращении к памяти

Это именно тот уровень "почему", который недостижим ни логированием, ни классическим профайлером.

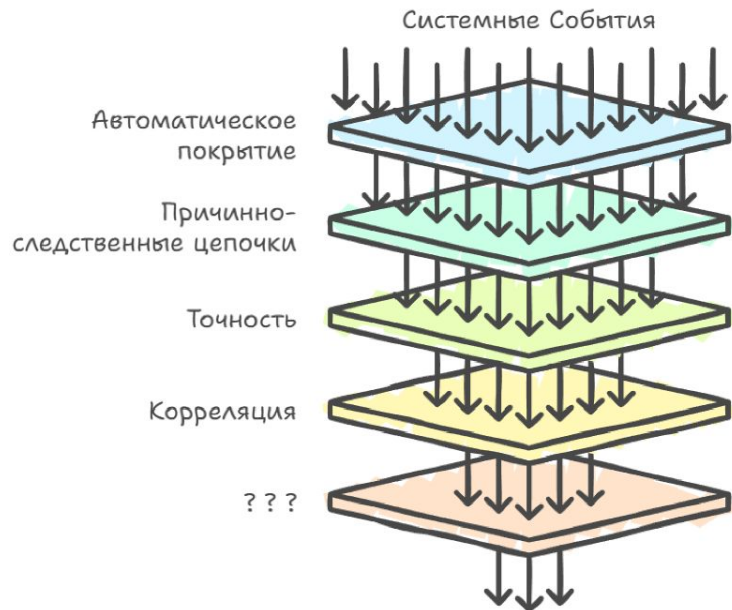
Точность событий



Системные события фиксируются на уровне ядра, где overhead минимален и контролируем или через аппаратные таймеры, которые не добавляют инструкций в код.

PMU (Performance Monitoring Unit) на современных процессорах считает cache misses, branch mispredictions и retiring instructions аппаратно, параллельно с выполнением программы, не замедляя её.

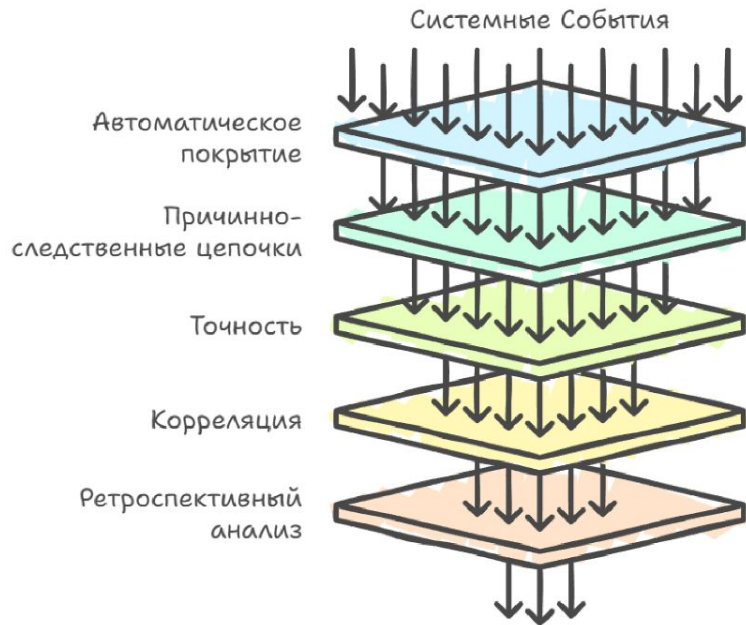
Корреляция слоев данных



Perfetto или WPA (позволяют одновременно видеть GPU timeline, CPU threads, частоты ядер, активность драйверов и сетевые события на одной временной оси.

Мы можем смотреть на все слои системы одновременно и видеть, что именно в этот момент процессор снизил частоту CPU с 3.2GHz до 1.8GHz и событие, никакое логирование в коде игры никогда этого не зафиксирует, потому что оно происходит вне игры.

Ретроспективный анализ

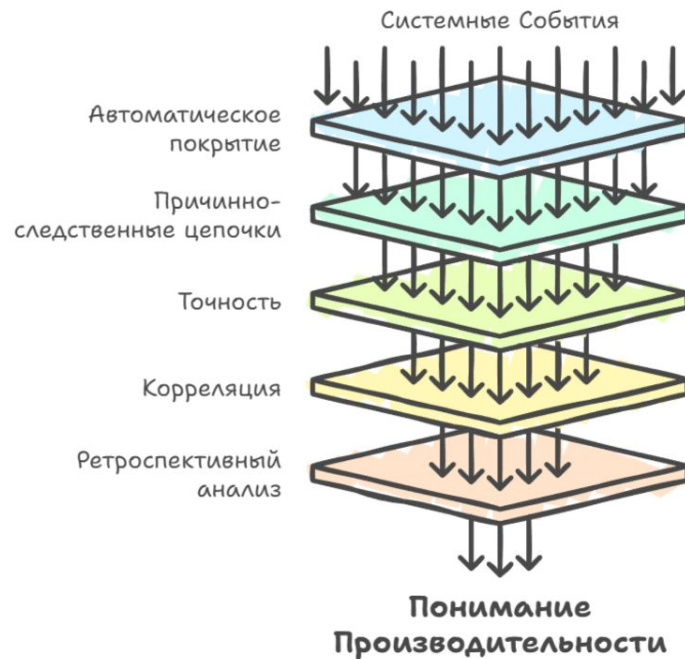


Логирование требует заранее знать, что именно вы хотите записать.

Event-based профилирование позволяет задать вопрос после захвата: "а покажи-ка мне все аллокации крупнее 1MB за последние 5 секунд" или "когда именно поток рендеринга ждал дольше 2ms и по какой причине" и получить ответ из уже собранных данных, не перезапуская сессию.

Понимание производительности

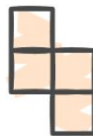
Отличие event-based подхода заключается в том, что он позволяет не столько ответить на вопрос «где мы тратим время», сколько на более фундаментальный вопрос «что вообще происходит в системе, когда и почему», что открывает несколько другой уровень понимания поведения программы



Статический анализ и профилирование

Работает на этапе компиляции или до неё, и позволяет проблему обнаружить до того, как она стала багом в продакшене или узким местом в профайлере.

Анализ
сложности
алгоритмов
Оценка поведения в
терминах большого O



Поиск неявных
ошибок
Анализ кода на
наличие ошибок

Статический анализ и профилирование



Найти потенциальный cache-unfriendly паттерн доступа к памяти в pull request дешевле на порядки, чем ловить его через PIX на финальном билде. Это фундаментальный сдвиг от реактивной диагностики к превентивной.

Поиск неявных ошибок

Анализ кода на наличие ошибок



Анализ сложности алгоритмов

Оценка поведения в терминах большого O



Подсчёт инстанций шаблонов

Измерение размера скомпилированного кода

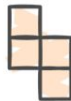


Статический анализ и профилирование

Анализ
векторизации
циклов
Оценка векторизации
циклов компилятором



Анализ
сложности
алгоритмов
Оценка поведения в
терминах большого O



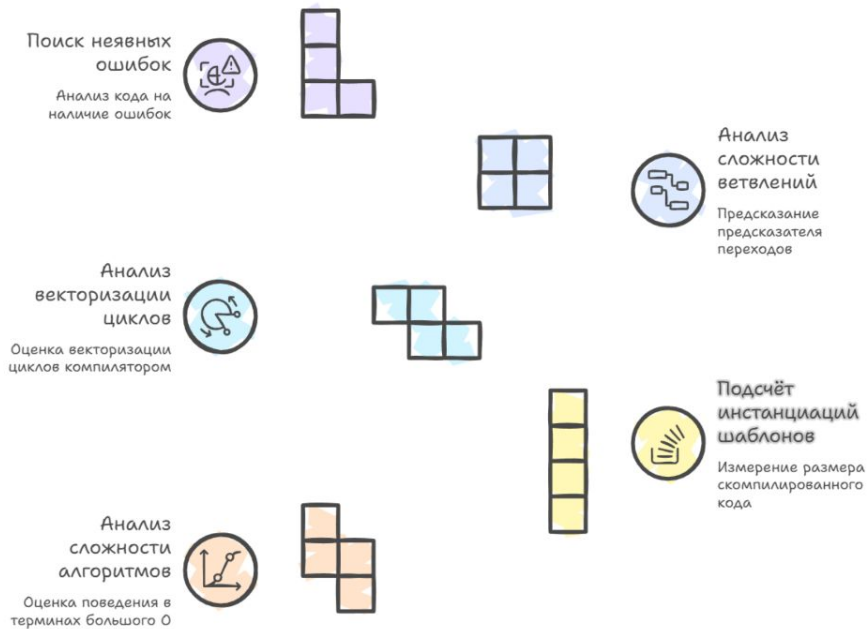
Поиск неявных
ошибок
Анализ кода на
наличие ошибок



Подсчёт
инстанциаций
шаблонов
Измерение размера
скомпилированного
кода

Статический анализ по определению не добавляет ничего в исполняемый код. Нет инструментации, нет observer effect, нет разницы между дебажной и релизной сборкой в части наблюдения.

Статический анализ и профилирование



Динамические инструменты видят только те пути выполнения, которые были реально пройдены во время сессии.

Статический анализ видит *все* ветви кода одновременно, включая редкие edge cases, error handling и платформозависимые пути компиляции.

Если интересно узнать больше

