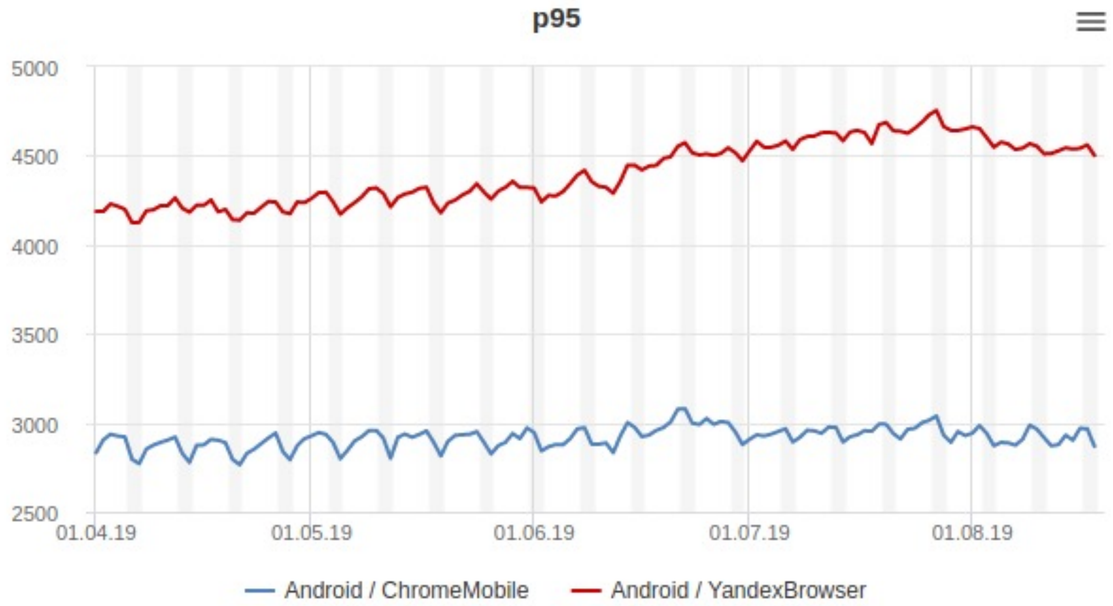
Яндекс

Как мы ускоряли создание процессов в Android

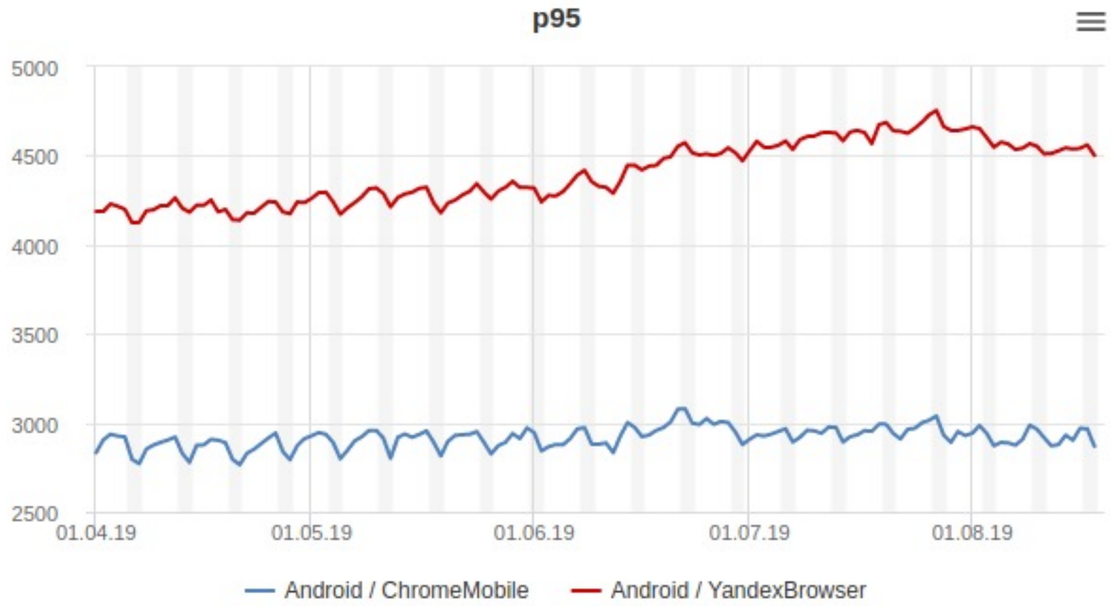
Вадим Петров, Александр Семашко

YandexBrowser отстает от Chrome (Android)



Отстаем от Chrome в 1.5 раза по скорости открытия страниц. В чьем коде причина?

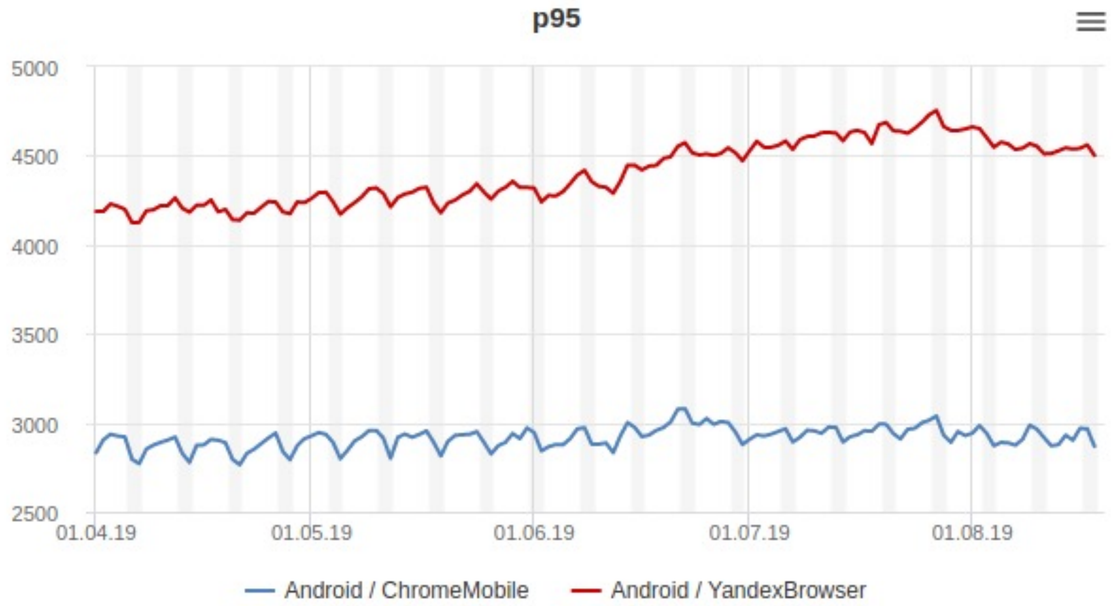
YandexBrowser отстает от Chrome (Android)



Отстаем от Chrome в 1.5 раза по скорости открытия страниц. В чьем коде причина?

- Яндекс.Браузер

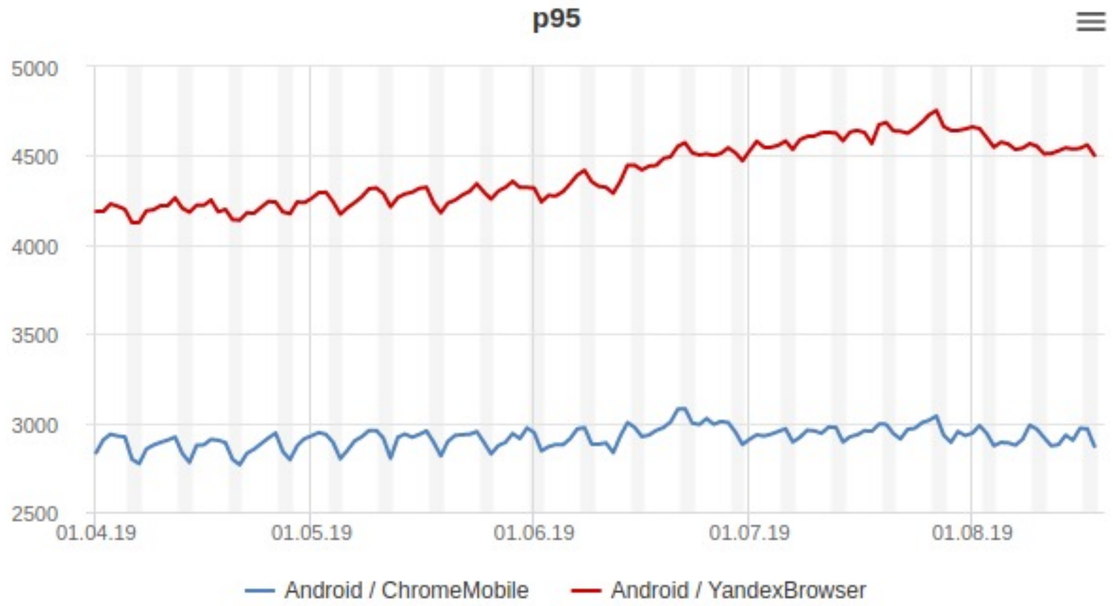
YandexBrowser отстает от Chrome (Android)



Отстаем от Chrome в 1.5 раза по скорости открытия страниц. В чьем коде причина?

- Яндекс.Браузер
- Google Chrome

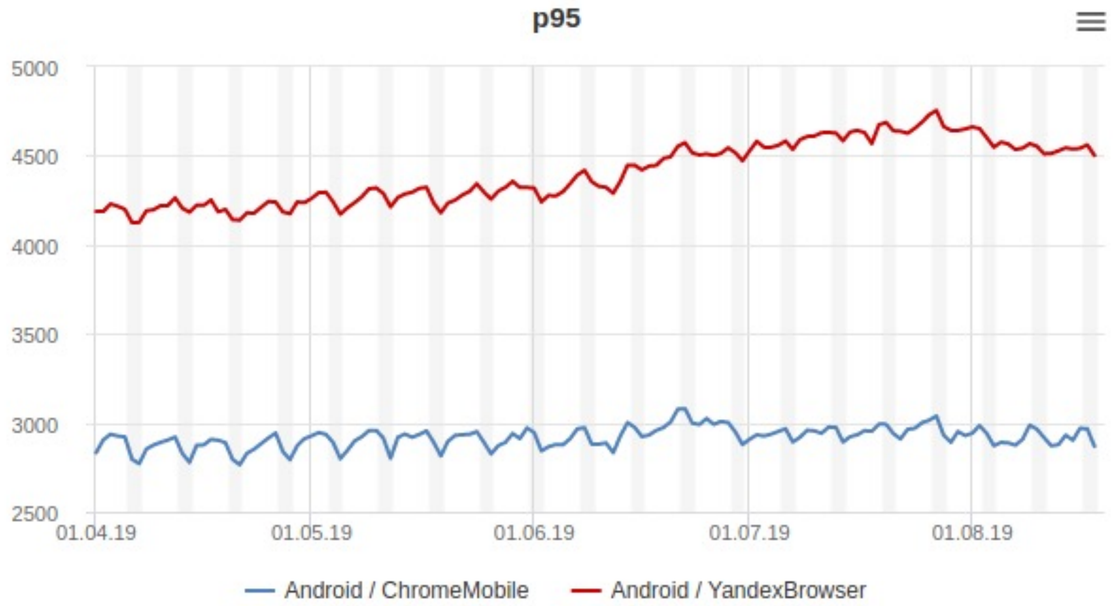
YandexBrowser отстает от Chrome (Android)



Отстаем от Chrome в 1.5 раза по скорости открытия страниц. В чьем коде причина?

- Яндекс.Браузер
- Google Chrome
- Chromium

YandexBrowser отстает от Chrome (Android)

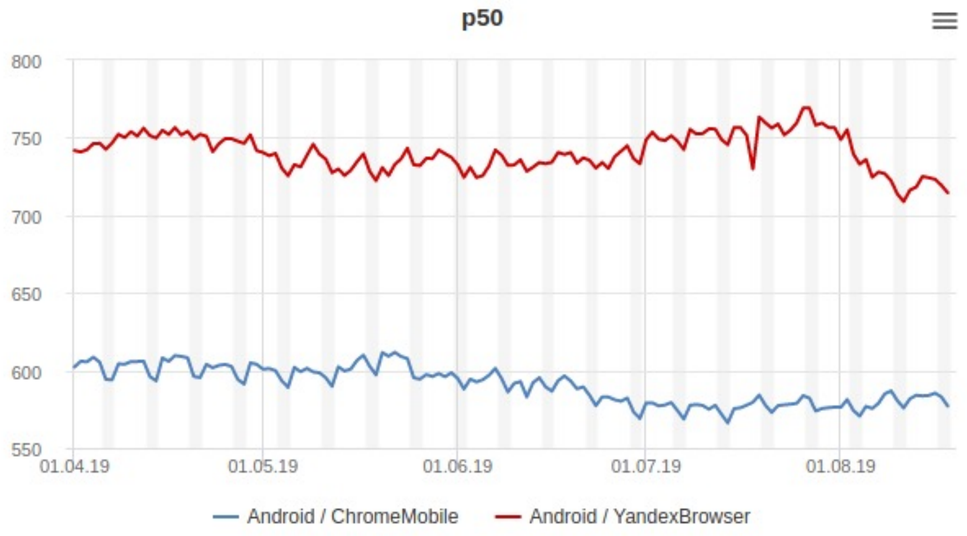


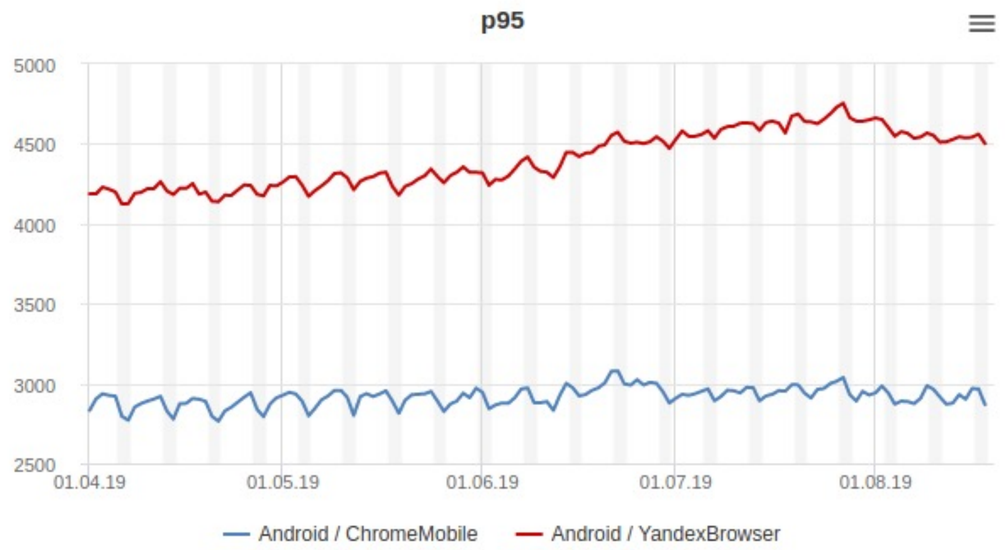
Отстаем от Chrome в 1.5 раза по скорости открытия страниц. В чьем коде причина?

- Яндекс.Браузер
- Google Chrome
- Chromium
- Android

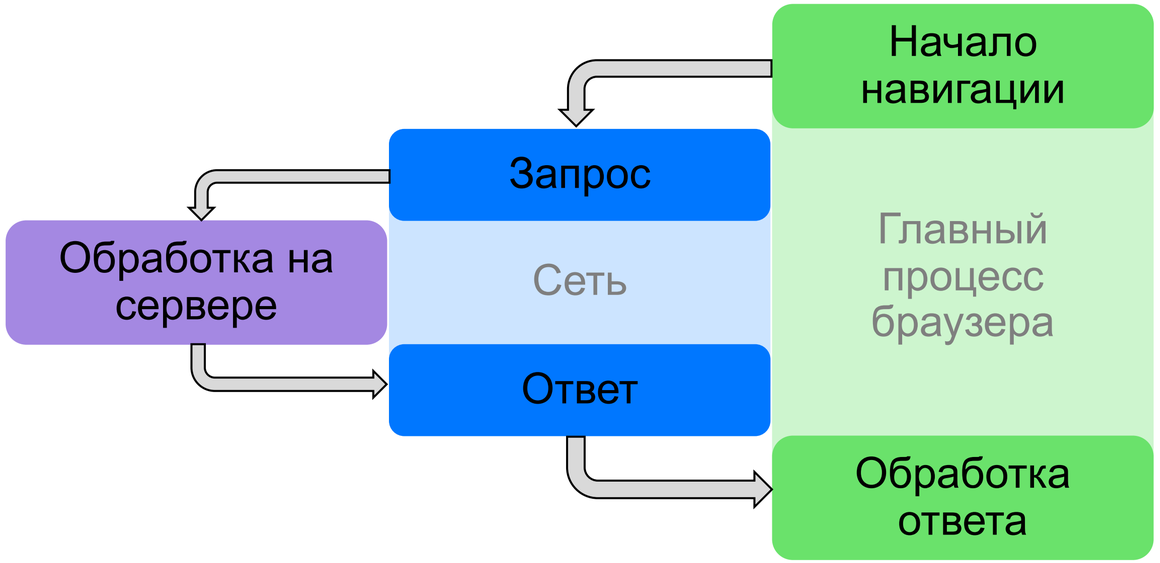
YandexBrowser vs Chrome (Android)

SERP (Search Engine Results Page) Яндекса собирает данные о скорости своей загрузки с помощью Navigation Timing, Paint Timing и других API.
Обнаружили, что на Android в Яндекс.Браузере он грузится медленнее, чем в Chrome.

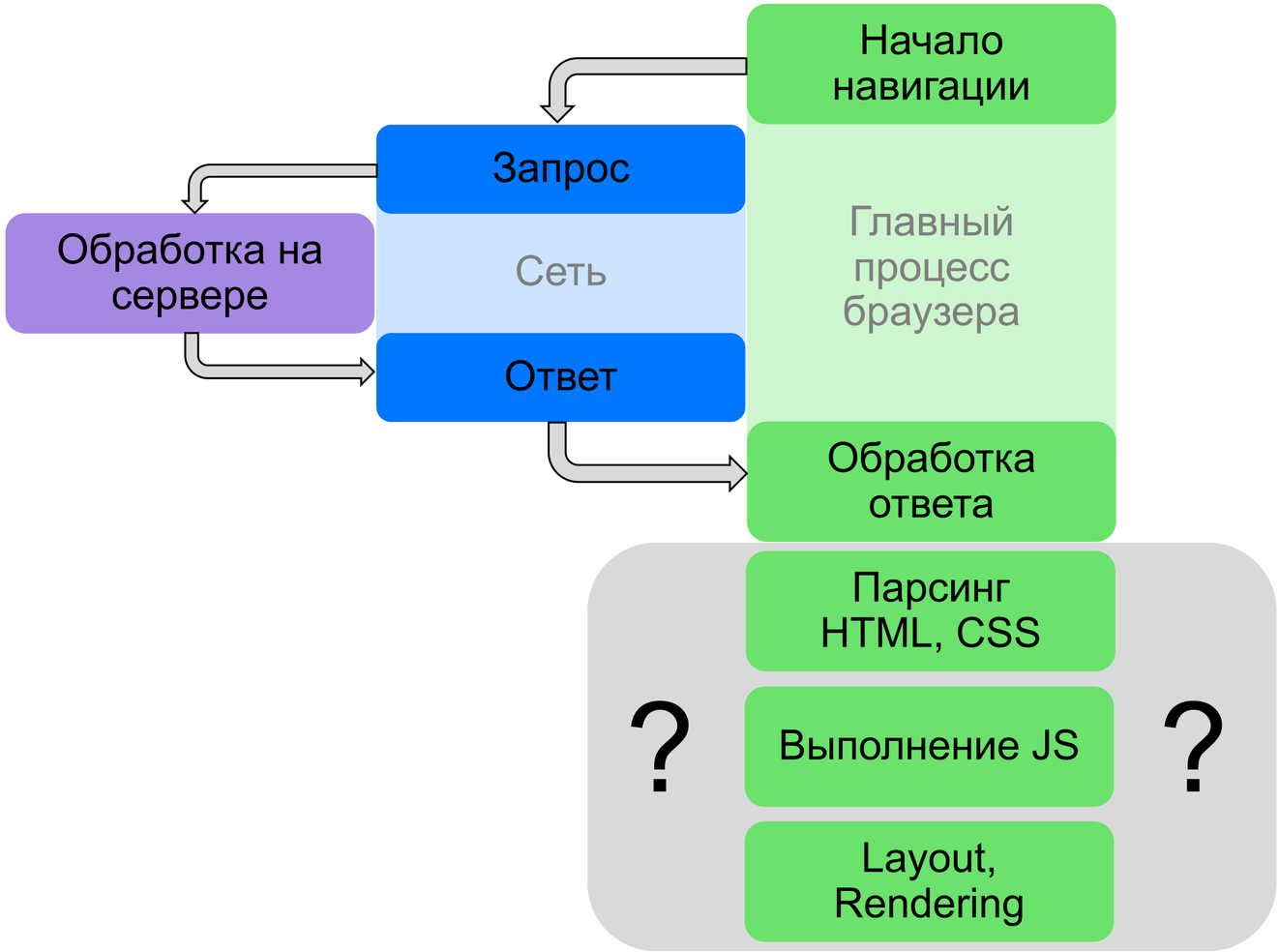




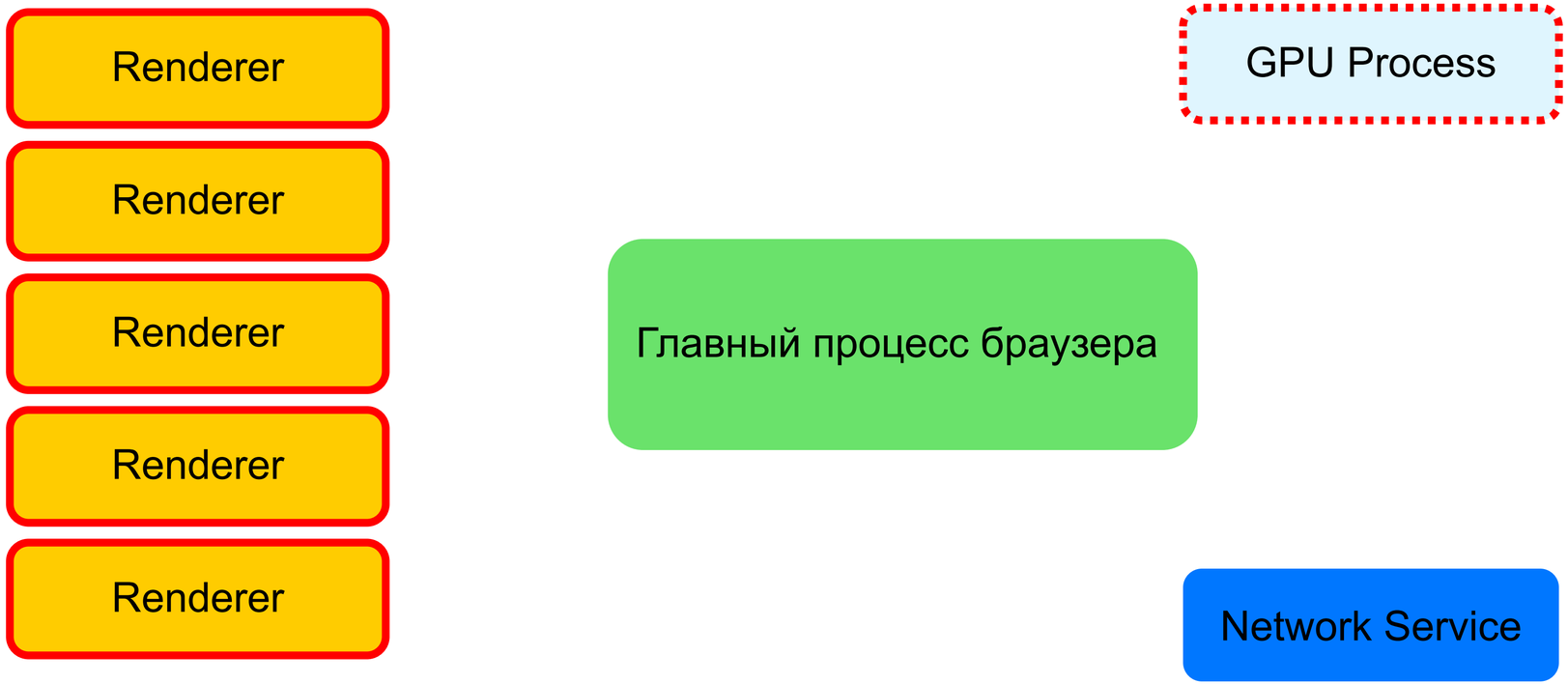
Скорость открытия страниц



Скорость открытия страниц



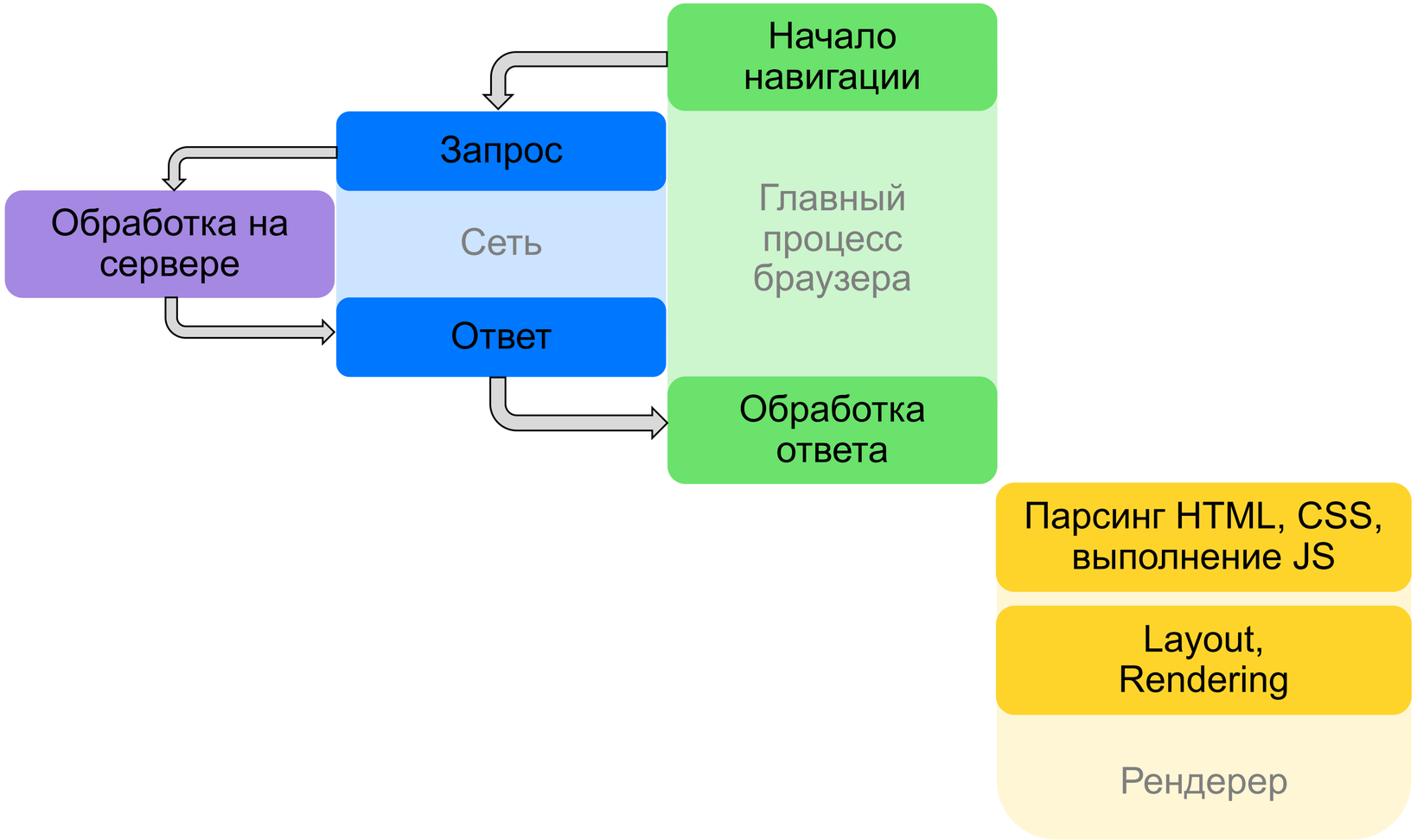
Важный факт про браузер: процессы



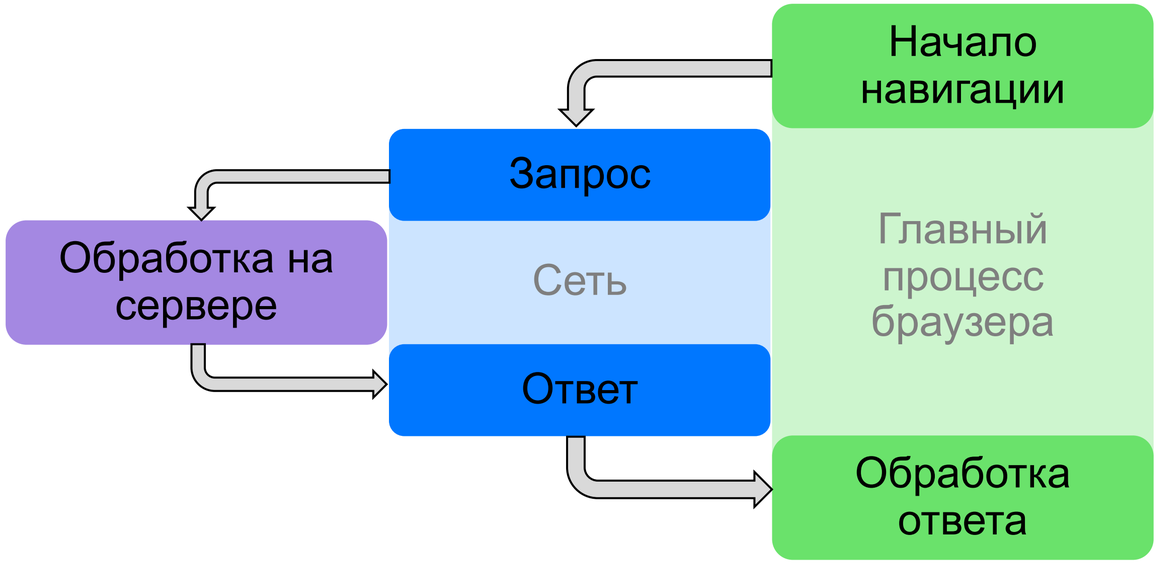
Важный факт про браузер: процессы

Новые рендереры запускаются,
когда возникает необходимость

Скорость открытия страниц



Скорость открытия страниц



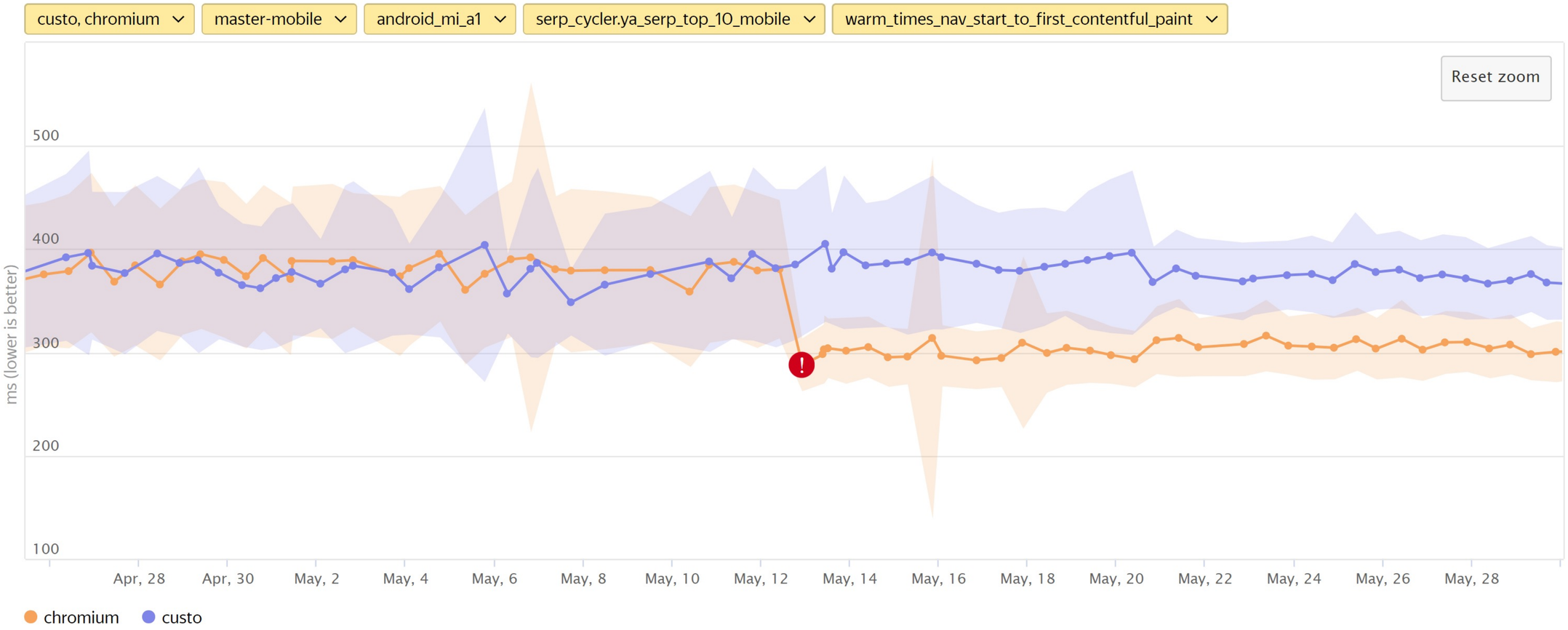
Выяснили, что отставание "начинается" на этапе обработки HTML

Что делать с этой проблемой ?

Давайте вместе пройдем путь решения этой проблемы.

Перф-тесты

Не показывают отставания



Аналитика

- Типы навигаций и их скорости

Аналитика

- Типы навигаций и их скорости
- Старт браузера

Аналитика

- Типы навигаций и их скорости
- Старт браузера
- Влияние девайсов
- Влияние сети

Аналитика

- Типы навигаций и их скорости
- Старт браузера
- Влияние девайсов
- Влияние сети
- Роботы

Аналитика

- Типы навигаций и их скорости
- Старт браузера
- Влияние девайсов
- Влияние сети
- Роботы
- Расширения

https://www.debugbear.com/blog/chrome-extension-performance-2021

Аналитика

- Типы навигаций и их скорости
- Старт браузера
- Влияние девайсов
- Влияние сети
- Роботы
- Расширения
- Поняли, что p95 у нас сильно хуже, тогда как на p50 мы близки

https://www.debugbear.com/blog/chrome-extension-performance-2021

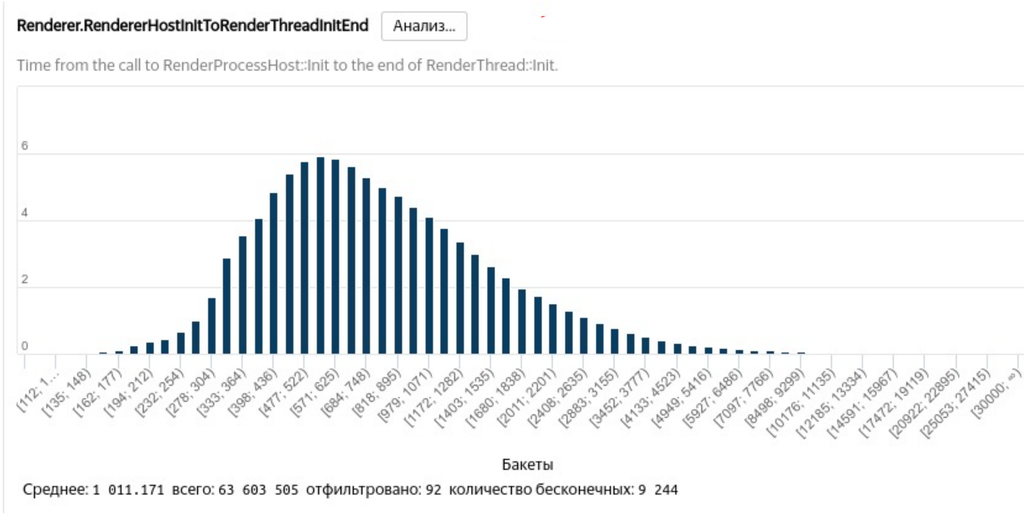
Предсоздание процессов

Для навигации иногда можно взять уже существующий рендерер.

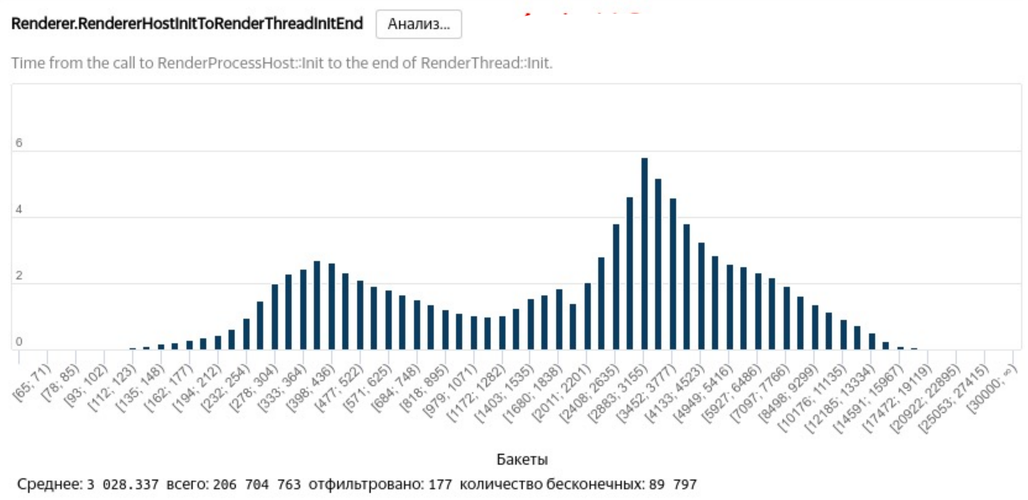
Трудности:
- OOM-killer
- Всегда остаются сценарии, в которых готового рендерера нет

Собрали метрики и поняли, что это не решает проблему полностью.

Про создание процессов

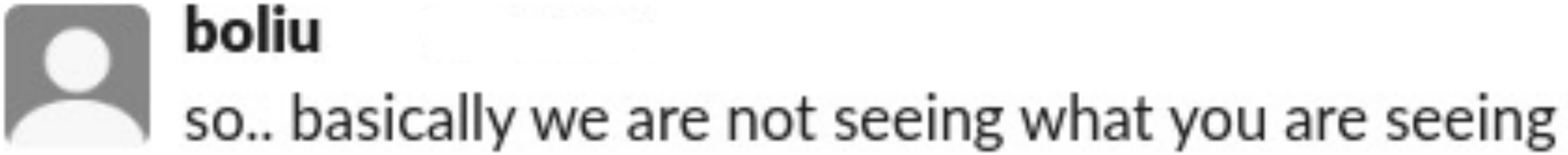


Android 6 и младше



Android 7 и старше

Общение с гуглом

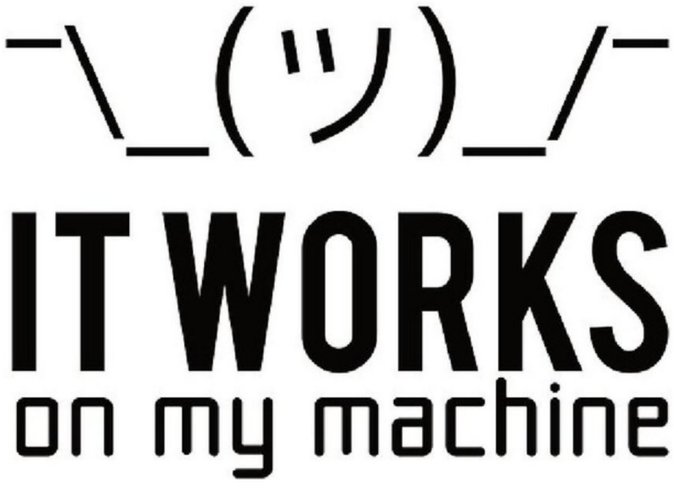


Ручное тестирование

- Сильные девайсы
- Только что поставленный браузер
- Все то же, что и в перф-тестах

Ручное тестирование

- Сильные девайсы
- Только что поставленный браузер
- Все то же, что и в перф-тестах



Honor 7A

- Особенно медленный девайс

Honor 7A

- Особенно медленный девайс
- Пожил с ним несколько дней

Honor 7A

- Особенно медленный девайс
- Пожил с ним несколько дней
- Написал скрипт, который открывал много сайтов и снимал метрики

Honor 7A

› Chrome

Chrome 80.0.3987.117:

```
Histograms
Stats accumulated from browser startup to previous page load; reload to get stats as of this page load.

Refresh
Histogram: MPArch.ChildProcessLaunchSubsequent recorded 26 samples, mean = 864.4 (flags = 0x41)
0     ...
10    ------------------------O                                                  (1 = 3.8%) {0.0%}
12    ...
449   ----------O                                                               (1 = 3.8%) {3.8%}
533   ------------------------------O                                           (3 = 11.5%) {7.7%}
633   ------------------------------O                                           (3 = 11.5%) {19.2%}
752   ------------------------------------------------------------------O       (6 = 23.1%) {30.8%}
894   ----------------------------------------------------------------------------O (7 = 26.9%) {53.8%}
1062  ------------------------------O                                           (3 = 11.5%) {80.8%}
1262  ----------O                                                               (1 = 3.8%) {92.3%}
1500  ----------O                                                               (1 = 3.8%) {96.2%}
1782  ...
```

860ms

Honor 7A

- Chrome

Chrome 80.0.3987.117:

```
Histograms
Stats accumulated from browser startup to previous page load; reload to get stats as of this page load.

Refresh
Histogram: MPArch.ChildProcessLaunchSubsequent recorded 26 samples, mean = 864.4 (flags = 0x41)
0     ...
10    ------------------------O                                    (1 = 3.8%) {0.0%}
12    ...
449   ----------O                                                  (1 = 3.8%) {3.8%}
533   ------------------------------O                             (3 = 11.5%) {7.7%}
633   ------------------------------O                             (3 = 11.5%) {19.2%}
752   ------------------------------------------------------------O (6 = 23.1%) {30.8%}
894   ------------------------------------------------------------------------------O (7 = 26.9%) {53.8%}
1062  ------------------------------O                             (3 = 11.5%) {80.8%}
1262  ----------O                                                 (1 = 3.8%) {92.3%}
1500  ----------O                                                 (1 = 3.8%) {96.2%}
1782  ...
```

860ms

- Yandex Browser

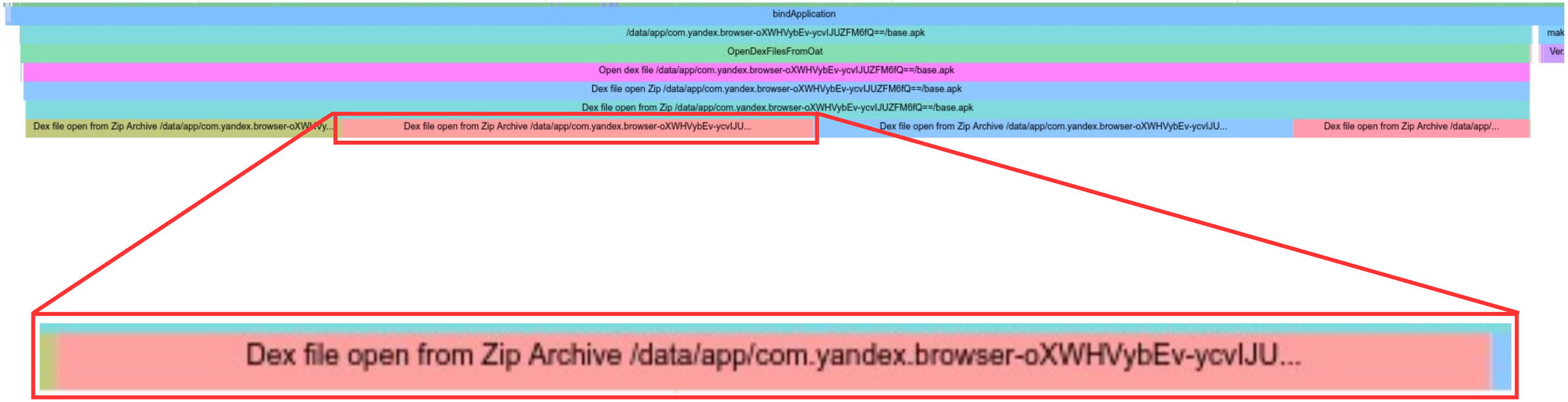
Yandex.Browser 20.2.3 от 21го февраля 2020:

```
Stats accumulated from browser startup to previous page load; reload to get stats as of this page load.

Refresh
Histogram: MPArch.ChildProcessLaunchSubsequent recorded 25 samples, mean = 7334.1 (flags = 0x1)
0      ...
1262   -------O                                                   (1 = 4.0%) {0.0%}
1500   ...
5017   -------O                                                   (1 = 4.0%) {4.0%}
5961   ------------------------------------------------------------O (9 = 36.0%) {8.0%}
7083   ------------------------------------------------------------------------------O (10 = 40.0%) {44.0%}
8416   --------------O                                            (2 = 8.0%) {84.0%}
10000  --------------O                                            (2 = 8.0%) {92.0%}
```

7300ms

Трейсы



Каждый рендерер во время запуска читает .dex файлы из APK.

Трейсы

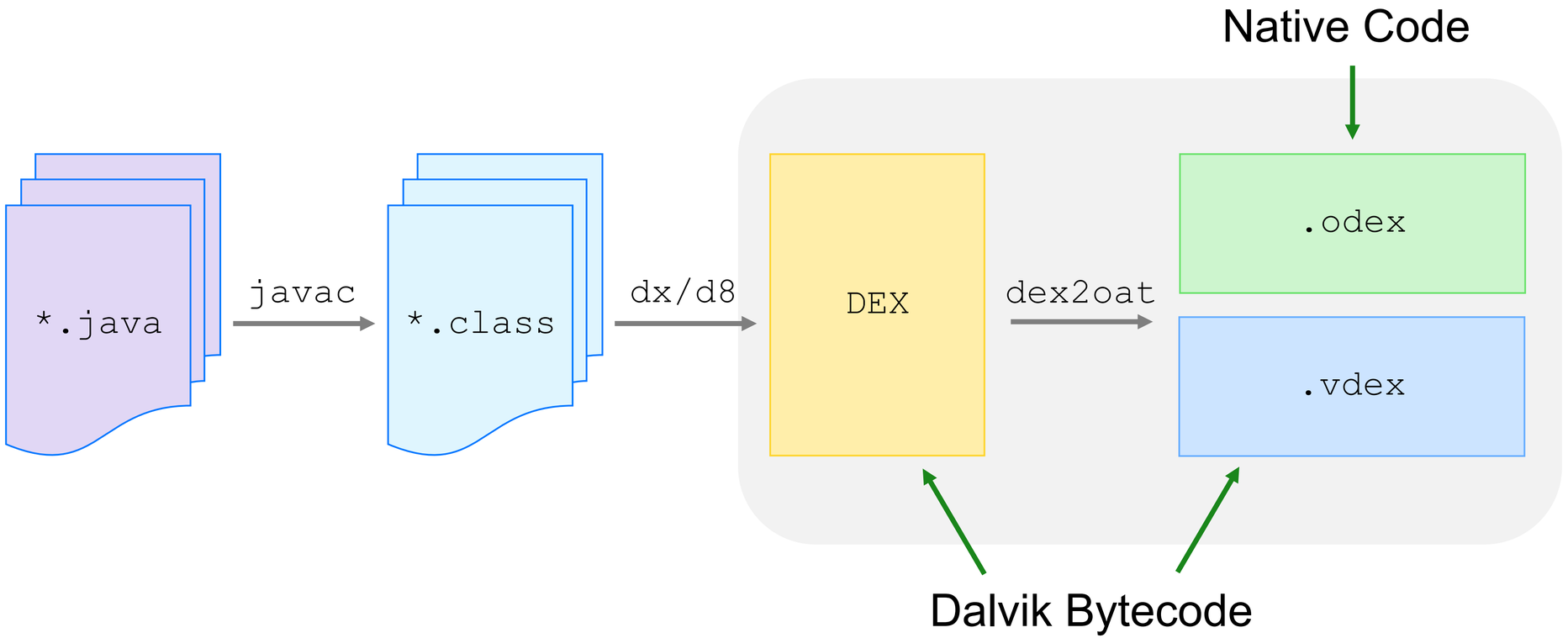
```
native: #04 pc 00000000001a3b1c  /system/lib64/libart.so (art::DexFileVerifier::Verify()+52)
native: #05 pc 00000000001a3984  /system/lib64/libart.so (art::DexFileVerifier::Verify(art::DexFile const*, unsigned char const*, unsigned long, char c
native: #06 pc 0000000000193210  /system/lib64/libart.so (art::DexFile::OpenOneDexFileFromZip(art::ZipArchive const&, char const*, std::__1::basic_stri
native: #07 pc 0000000000192b54  /system/lib64/libart.so (art::DexFile::OpenAllDexFilesFromZip(art::ZipArchive const&, std::__1::basic_string<char, std
native: #08 pc 000000000019268c  /system/lib64/libart.so (art::DexFile::OpenZip(int, std::__1::basic_string<char, std::__1::char_traits<char>, std::__1
native: #09 pc 000000000019236c  /system/lib64/libart.so (art::DexFile::Open(char const*, std::__1::basic_string<char, std::__1::char_traits<char>, std
native: #10 pc 000000000040b4e8  /system/lib64/libart.so (art::OatFileManager::OpenDexFilesFromOat(char const*, _jobject*, _jobjectArray*, art::OatFile
native: #11 pc 00000000003cbb1c  /system/lib64/libart.so (art::DexFile_openDexFileNative(_JNIEnv*, _jclass*, _jstring*, _jstring*, int, _jobject*, _jobj
native: #12 pc 00000000001ecec8  /system/framework/arm64/boot-core-libart.oat (Java_dalvik_system_DexFile_openDexFileNative__Ljava_lang_String_2Ljava_l
at dalvik.system.DexFile.openDexFileNative(Native method)
at dalvik.system.DexFile.openDexFile(DexFile.java:353)
at dalvik.system.DexFile.<init>(DexFile.java:100)
at dalvik.system.DexFile.<init>(DexFile.java:74)
at dalvik.system.DexPathList.loadDexFile(DexPathList.java:374)
at dalvik.system.DexPathList.makeDexElements(DexPathList.java:337)
at dalvik.system.DexPathList.<init>(DexPathList.java:157)
at dalvik.system.BaseDexClassLoader.<init>(BaseDexClassLoader.java:65)
at dalvik.system.PathClassLoader.<init>(PathClassLoader.java:64)
at com.android.internal.os.ClassLoaderFactory.createClassLoader(ClassLoaderFactory.java:73)
```

И каждый рендерер выполняет верификацию .dex файлов.

Компиляция приложений

При установке приложения ART выполняет "компиляцию" инструментом dex2oat. В результате *обычно* создаются файлы:

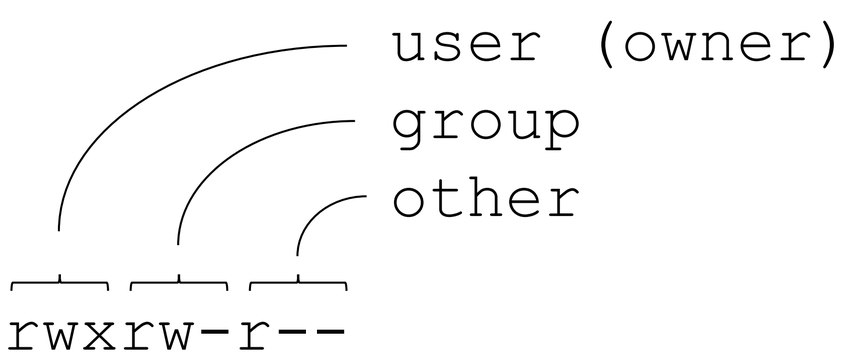
- vdex - распакованный dex + метаданные о верификации
- odex - скомпилированный нативный код.



Права на odex/vdex файлы

Обнаружили, что когда рендереры запускаются медленно, у файлов odex и vdex отсутствует бит на доступ "read other":

```
-rw-r----- 1 system all_a86 2801664 2020-03-04 20:46 base.art
-rw-r----- 1 system all_a86 13223700 2020-03-04 19:46 base.odex
-rw-r----- 1 system all_a86 36417660 2020-03-04 20:46 base.vdex
```




```
111110100      110100000
rwxrw-r--      rw-r-----
 7  6  4        6  4  0
```

Права на odex/vdex файлы

Обнаружили, что когда рендереры запускаются медленно, у файлов odex и vdex отсутствует бит на доступ "read other":

```
-rw-r----- 1 system all_a86 2801664 2020-03-04 20:46 base.art
-rw-r----- 1 system all_a86 13223700 2020-03-04 19:46 base.odex
-rw-r----- 1 system all_a86 36417660 2020-03-04 20:46 base.vdex
```

Под рутом можно сделать `chmod 644` этим файлам, т.е. дать доступ на чтение всем пользователям, `rw-r--r--`.

И тогда рендереры стартуют быстро.

А иногда на том же Honor 7A рендереры "сами по себе" запускаются быстро, при этом права у этих файлов также 644.

Кто все эти пользователи?
Изоляция приложений и процессов в Android

Application Sandbox основана на том, что у каждого приложения свой user и group. По умолчанию у приложения нет доступа к другим приложениям, системным файлам и процессам, и т.д.

Кроме того, рендереры в браузере - это изолированные сервисы (`android:isolatedProcess`).

- Отдельный пул пользователей, из-под которых запускаются процессы.
- Дополнительные настройки SELinux, которые применяются на старте процесса.

Дебаг Android

- Собрали LineageOS с доп. логами и отладочным кодом.
- Нашли в исходниках AOSP, что права на файлы odex/vdex зависят от режима компиляции. Доступ "для всех" пропадает при использовании profile-guided optimization (PGO), который впервые появился в Android 7.
- Узнали, что можно выполнить компиляцию со своими параметрами для исправления прав.

Режимы компиляции в ART

- ahead-of-time (AOT)
- just-in-time (JIT)
- profile-guided optimization (PGO) **- начиная с Android 7.0 (N)**

Режимы dex2oat, описанные в документации (Android O):

- *verify*: только верификация DEX
- *quicken*: верификация DEX и оптимизация *некоторых* инструкций
- *speed*: верификация DEX и AOT-компиляция всего кода
- *speed-profile*: верификация DEX и AOT-компиляция с использованием результатов профилирования

Компиляция "вручную"

Компиляцию можно запустить, выполнив на устройстве простую команду с помощью adb shell или Runtime.getRuntime().exec():
`cmd package compile -m speed` – дефолтный режим
`cmd package compile -m speed-profile` – с использованием PGO, выше скорость.

Также можно флагом `-r shared` сказать, что код нужен и другим пакетам, это запрещает использовать профили.
Он используется, например, для WebView.

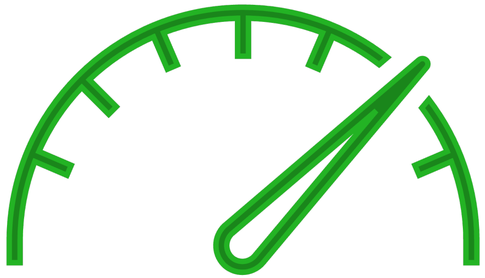
Итого: почему у нас все плохо (было)

Мы изолируем рендереры
=> они запускаются под "бесправными" user-ами, не имеющими никакого отношения к user-у, соответствующему нашему приложению (user A).

Итого: почему у нас все плохо (было)

Мы изолируем рендереры
=> они запускаются под "бесправными" user-ами, не имеющими никакого отношения к user-у, соответствующему нашему приложению (user A).

Система компилирует наше приложение с использованием PGO



Итого: почему у нас все плохо (было)

Мы изолируем рендереры
=> они запускаются под "бесправными" user-ами, не имеющими никакого отношения к user-у, соответствующему нашему приложению (user A).

Система компилирует наше приложение с использованием PGO
=> после этого только user A и его группа могут прочитать odex/vdex.

Итого: почему у нас все плохо (было)

Мы изолируем рендереры
=> они запускаются под "бесправными" user-ами, не имеющими никакого отношения к user-у, соответствующему нашему приложению (user A).

Система компилирует наше приложение с использованием PGO
=> после этого только user A и его группа могут прочитать odex/vdex.

В итоге каждый рендерер на старте вынужден заново распаковывать .dex файлы из APK, читать их и выполнять полную верификацию.

Итого: почему у нас все плохо (было)

Мы изолируем рендереры
=> они запускаются под "бесправными" user-ами, не имеющими никакого отношения к user-у, соответствующему нашему приложению (user A).

Система компилирует наше приложение с использованием PGO
=> после этого только user A и его группа могут прочитать odex/vdex.

В итоге каждый рендерер на старте вынужден заново распаковывать .dex файлы из APK, читать их и выполнять полную верификацию.

Это долго: на некоторых телефонах это занимает больше 5 секунд!

Почему у Chrome все хорошо

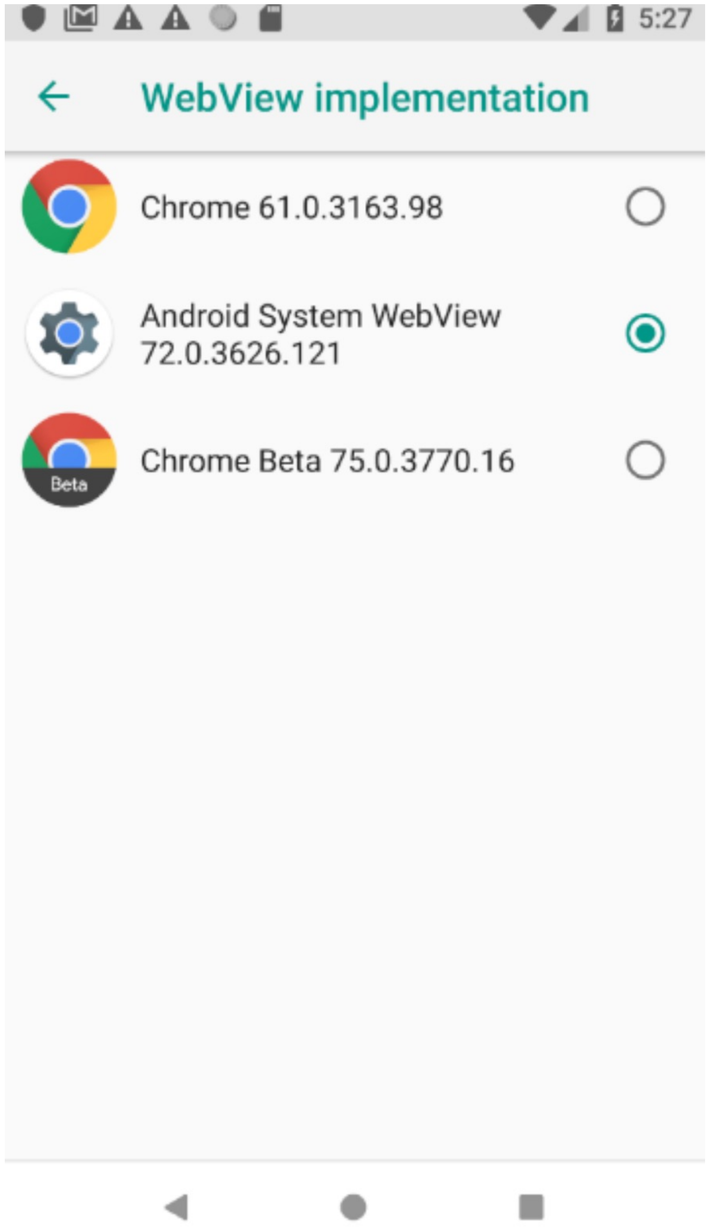
- Chrome тесно связан с Android
- Chrome является провайдером WebView
- В Chrome сильно меньше java-кода

Типы Chrome

- **Chrome**: отдельно Хром, отдельно WebView (Android 5 и 6)

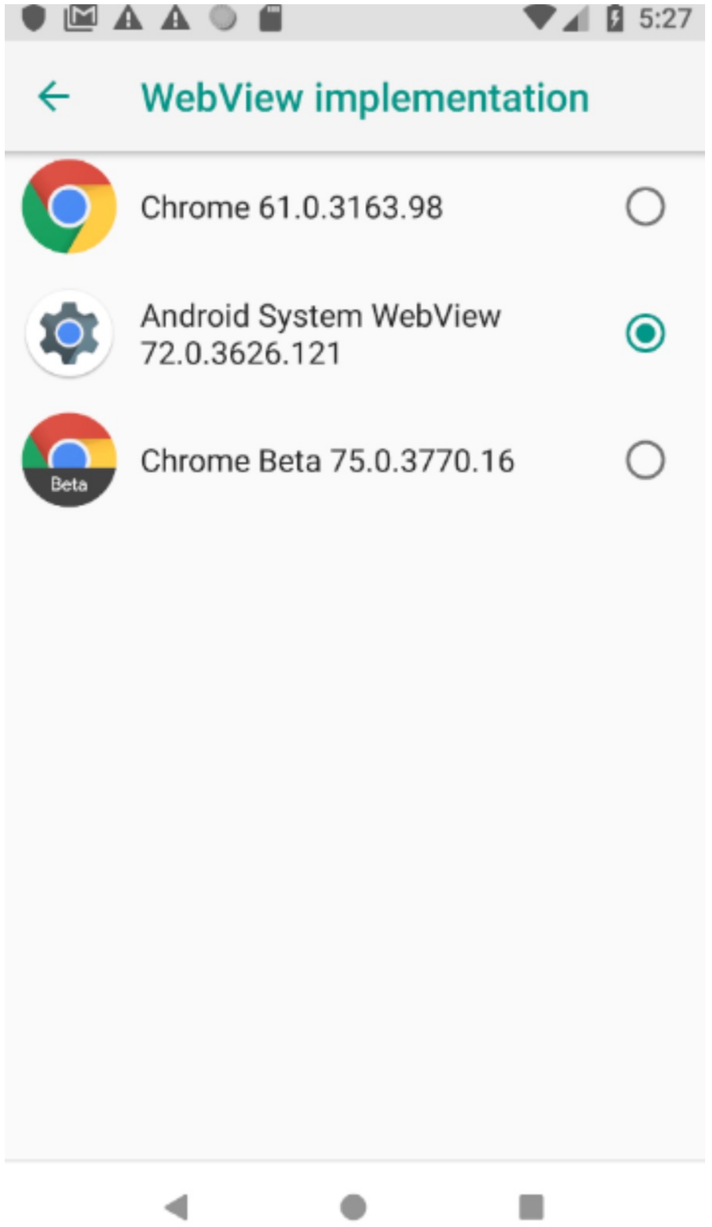
Типы Chrome

- **Chrome**: отдельно Хром, отдельно WebView (Android 5 и 6)
- **Monochrome**: Chrome и WebView в одном APK (Android 7, 8, 9)



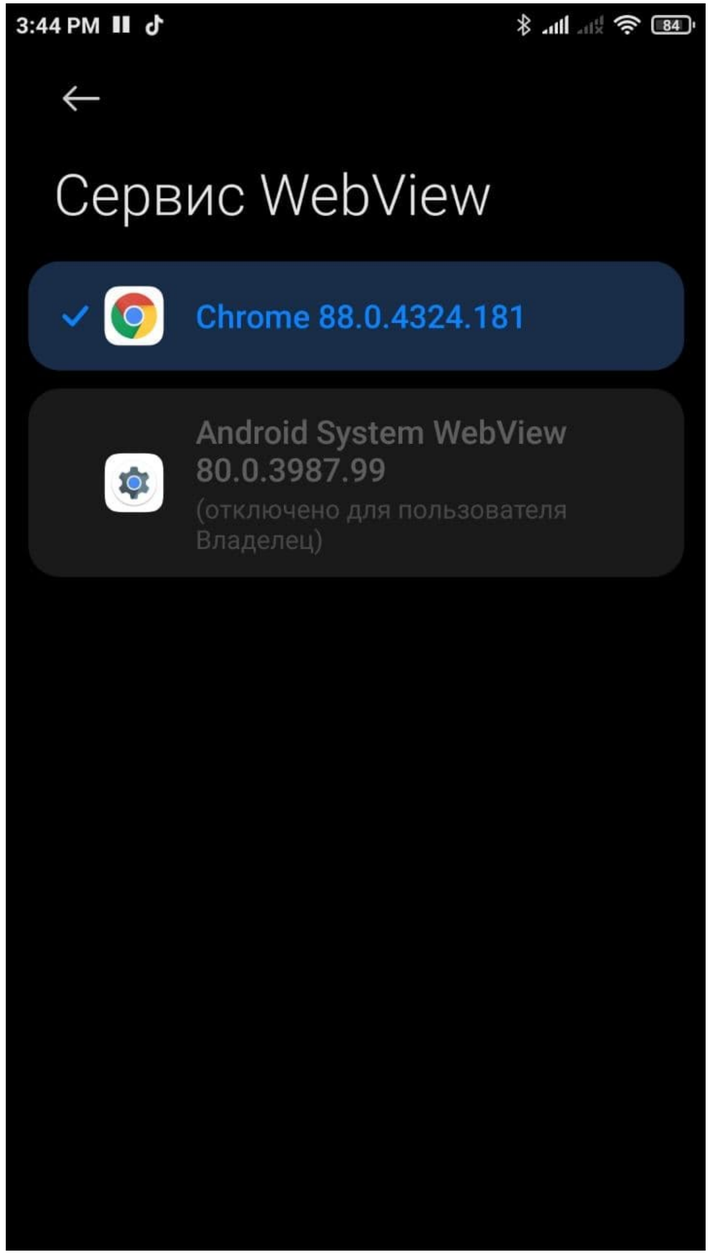
Типы Chrome

- **Chrome**: отдельно Хром, отдельно WebView (Android 5 и 6)
- **Monochrome**: Chrome и WebView в одном APK (Android 7, 8, 9)
- **Trichrome**: Chrome отдельно, WebView отдельно, но общий нативный код в отдельном APK (Android 10+)



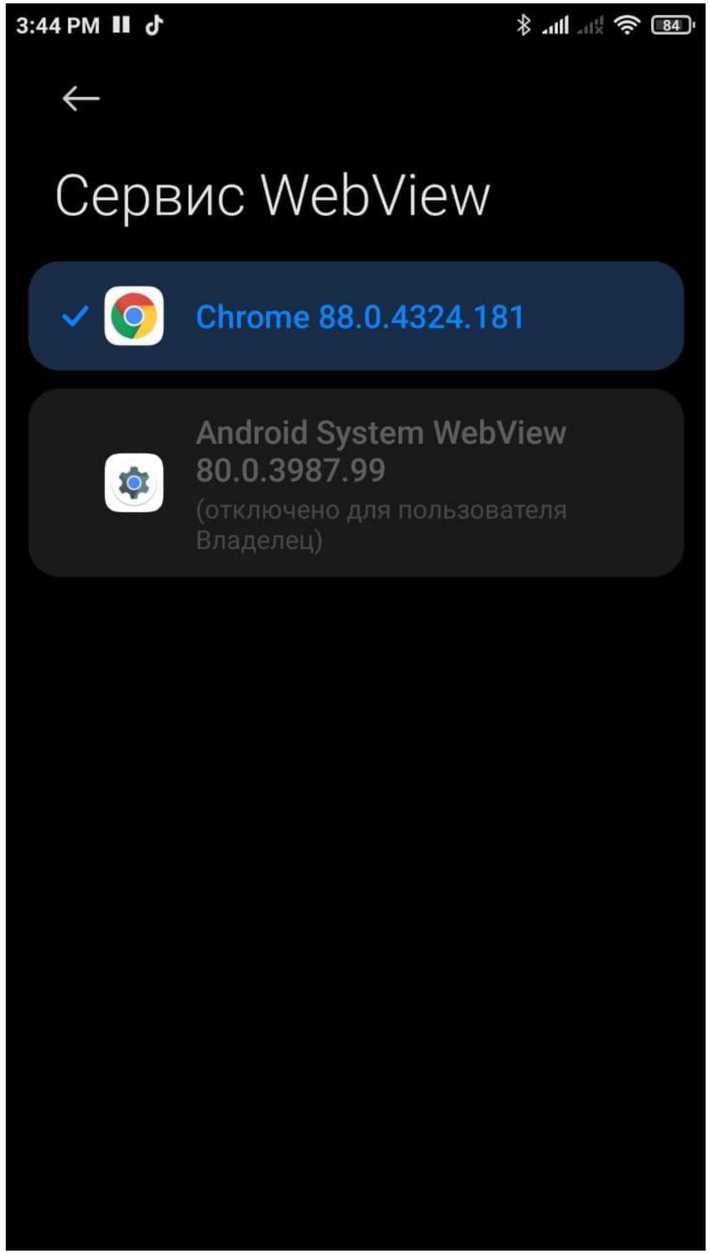
Дистрибуция у Chrome

- Можем ли мы наш браузер сделать провайдером WebView?



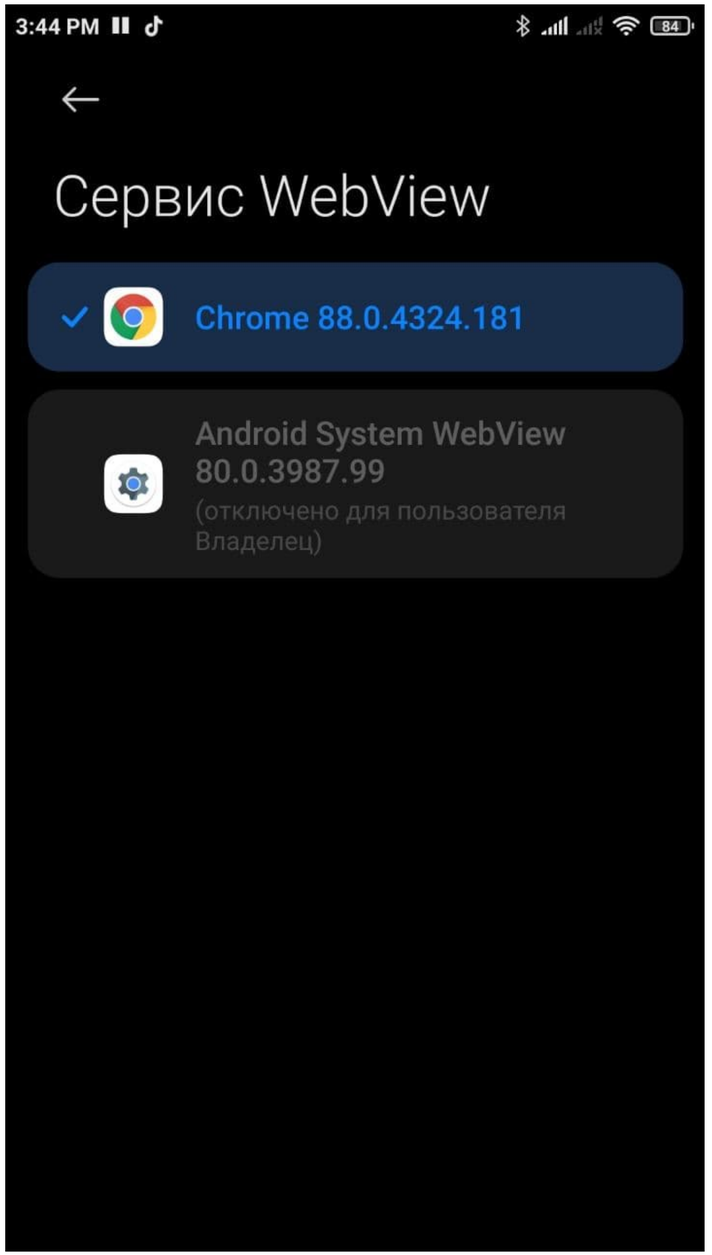
Дистрибуция у Chrome

- Можем ли мы наш браузер сделать провайдером WebView?
- **Нет**



Дистрибуция у Chrome

- Можем ли мы наш браузер сделать провайдером WebView?
- **Нет**
- Список допустимых пакетов захардкожен в read-only файловой системе.
- Наливается на заводе. На реальном девайсе видим:
  Chrome (Stable, Beta, Dev, Canary, Debug), Google WebView.



Другие браузеры под Android

Chrome: подвержен проблеме в редких случаях, когда не является системным приложением

Opera: не использует изоляцию, проблемы нет

Edge: подвержен проблеме

Samsung browser: подвержен проблеме

Firefox: один процесс, падает сайт - падает браузер

Решение для Android 10+

App Zygote

Zygote - процесс, который порождает (fork)
процессы приложений, благодаря чему они все используют одну и ту же копию ART и некоторых системных классов и ресурсов.

App Zygote: приложение реализует интерфейс ZygotePreload, с помощью которого загружает данные, которые нужны дочерним процессам.

С App Zygote тоже проблемы

Через 5 секунд после завершения последнего процесса, который форкнулся от App Zygote, она умирает ¯_(ツ)_/¯

Чтобы самим управлять временем жизни своей zygote, сделали процесс-холдер.

Решение для старых Android 7 – 9 (перекомпиляция)

Т.к. world-readable доступ теряется только при компиляции с profile-guided optimization, можно перекомпилировать без PGO:

```
cmd package compile -m speed
```

(или задав режим `-r shared`)

Влияет не только на рендереры:

- Ухудшение скорости старта браузера, производительности главного процесса
- Увеличивает энергопотребление

Решение для старых Android 7 – 9 (деизоляция)

- изоляцию для 7-9 можно отключить

Результаты

- p95 ускорили на 40%
- Среднее ускорили на 18%

Основные метрики Поиска					
★ Время отрисовки первого сниппета от первого байта	773.2179	631.8461	▼	-141.3718	-18.28%
Основные метрики ранжирования					
★ Позапросный профицит всей выдачи v6.1.b	0.216	0.218	▲	0.001951	0.90%
★ SAT v3	2.0168	2.0089	▼	-0.007958	-0.39%
★ Доля сессий с Session Time > 30 (SAT v3)	28.1996	28.0449	▼	-0.1547	-0.55%
★ Сверхдлинные, делённые на Session Time (SAT v3)	0.1259	0.1277	▲	0.001757	1.40%
★ Доля кликнутых по сверхдлинным	45.093	45.2431	▲	0.15	0.33%
★ Доля запросов со сверхдлинными кликами по top3 визуальных	37.9708	38.1145	▲	0.1437	0.38%
★ Доля запросов со сверхдлинными кликами по top6 визуальных	43.6203	43.7684	▲	0.1481	0.34%
Метрики для контроля					
☆ Доля запросов без page open	3.6857	3.3181	▼	-0.3676	-9.97%
☆ Доля запросов без page loaded	9.0297	8.5651	▼	-0.4646	-5.15%
☆ Общее количество page loaded в сессии	85 133	87 637	▲	2 504	2.94%

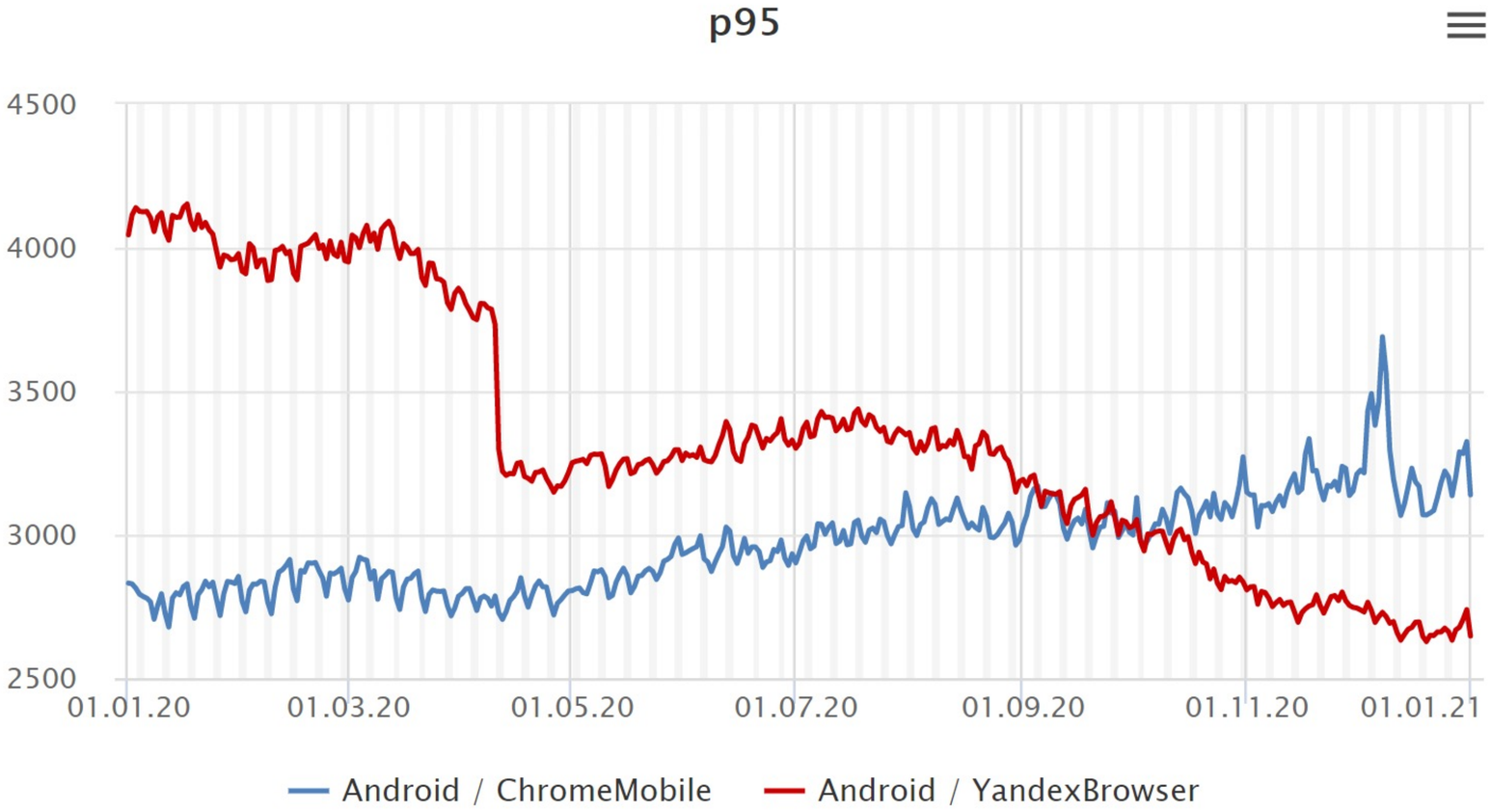
Результаты

- Навигейты (открытия страничек) **+1,02%**
- Поисковая доля **+0.04%**

– [Предсказание] Возвращаемость						
Количество активных дней на пользователя	8.3981	8.4006	0.005285	▲	0.002519	0.03%
Возвращаемость на 9 день	0.502	0.5032	0.001412	▲	0.001157	0.23%
Возвращаемость на 10 день	0.498	0.4993	0.000139	▲	0.001296	0.26%
Возвращаемость на 12 день	0.4961	0.4976	0.000054	▲	0.001508	0.30%
Возвращаемость на 13 день	0.4999	0.501	0.002122	▲	0.001099	0.22%
Возвращаемость на 14 день	0.502	0.5028	0.006048	▲	0.000787	0.16%
Возвращаемость на 15 день	0.5011	0.5017	0.0359	▲	0.000591	0.12%
+ Навигейты прокрашено: 2 / 5						
– [Предсказание] Навигейты						
Кол-во навигейтов на пользователя	210.541	212.6898	4.7e-12	▲	2.1488	1.02%

Результаты

- Быстрее Chrome!



Бонус: проблема у Гугла

Недавно в Google озаботились той же (и еще одной похожей) проблемой: https://crrev.com/c/2602657. Решение? Перекомпиляция!

```
@WorkerThread
@VisibleForTesting
static void fixDexIfNecessary(Runtime runtime) {
    ApplicationInfo appInfo = ContextUtils.getApplicationContext().getApplicationInfo();
    @DexFixerReason
    int reason = needsDexCompile(appInfo);
    if (reason > DexFixerReason.NOT_NEEDED) {
        Log.w(TAG, "Triggering dex compile. Reason=%d", reason);
        try {
            String cmd = "cmd package compile -r shared ";
            if (reason == DexFixerReason.NOT_READABLE && BuildConfig.ISOLATED_SPLITS_ENABLED) {
                // Isolated processes need only access the base split.
                String apkBaseName = new File(appInfo.sourceDir).getName();
                cmd += String.format("--split %s ", apkBaseName);
            }
            cmd += ContextUtils.getApplicationContext().getPackageName();
            runtime.exec(cmd);
        } catch (IOException e) {
            reason = DexFixerReason.FAILED_TO_RUN;
        }
    }
    RecordHistogram.recordEnumeratedHistogram("Android.DexFixer", reason, DexFixerReason.COUNT);
}
```

Выводы

- Технические проблемы сильно влияют на продукт в целом
- Ошибки допускают все
- Важно докапываться до основной причины проблемы
- Можно дебажить даже Android

Яндекс

Спасибо

Вадим Петров

Руководитель службы производительности приложений

lof84@yandex-team.ru

Александр Семашко

Ведущий разработчик

ahest@yandex-team.ru