VK Звонки: десктопный клиент для видеозвонков без лимита на число участников — с нуля за 365 дней

Павел Мацула, тимлид команды разработки нативных клиентов

О чём я расскажу?

1

День 1.
Постановка
задачи

О чём я расскажу?

1

День 1.
Постановка
задачи

2

День 30.
Proof-of-concept.
Консольный
клиент

О чём я расскажу?

1

День 1.
Постановка задачи

2

День 30.
Proof-of-concept.
Консольный клиент

3

День 100.
Minimum Viable Prototype.
Qt-клиент

О чём я расскажу?

1. День 1. Постановка задачи
2. День 30. Proof-of-concept. Консольный клиент
3. День 100. Minimum Viable Prototype. Qt-клиент
4. День 200. Nice-to-have features

О чём я расскажу?

1. День 1. Постановка задачи
2. День 30. Proof-of-concept. Консольный клиент
3. День 100. Minimum Viable Prototype. Qt-клиент
4. День 200. Nice-to-have features
5. День 300. Bottlenecks. Tips & tricks

День 1. Постановка задачи

Лето 2020

РАСЦВЕТ ВИДЕОСВЯЗИ

Наступили времена, когда весь мир ушёл на самоизоляцию и работу в удалённом режиме.

Единственная надежда — качественная видеосвязь, чтобы объединить их всех.

Что у нас уже было?

По платформам:

- Видеозвонки на портале vk.com в большинстве десктопных (Windows, macOS, Linux) браузеров

Что у нас уже было?

По платформам:

- Видеозвонки на портале vk.com в большинстве десктопных (Windows, macOS, Linux) браузеров
- Видеозвонки в приложении VK на мобильных устройствах Android и iOS

Что у нас уже было?

По платформам:

- Видеозвонки на портале vk.com в большинстве десктопных (Windows, macOS, Linux) браузеров
- Видеозвонки в приложении VK на мобильных устройствах Android и iOS
- Бэкенд для p2p- и групповых видеозвонков

Что у нас уже было?

По платформам:

- Видеозвонки на портале vk.com в большинстве десктопных (Windows, macOS, Linux) браузеров
- Видеозвонки в приложении VK на мобильных устройствах Android и iOS
- Бэкенд для p2p- и групповых видеозвонков
- Инфраструктура для разработки, в т.ч. на C++

Что у нас уже было?

По платформам:

- Видеозвонки на портале vk.com в большинстве десктопных (Windows, macOS, Linux) браузеров
- Видеозвонки в приложении VK на мобильных устройствах Android и iOS
- Бэкенд для p2p- и групповых видеозвонков
- Инфраструктура для разработки, в т.ч. на C++

Действительно, на десктопах уже можно было совершать видеозвонки — через браузеры.

Зачем нам что-то ещё?

Достоинства браузерной версии:

- Браузер — **главная точка входа в соцсеть** для пользователей на десктопах
- Браузер **есть на всех десктопах**, и больше ничего не надо устанавливать
- Вкладка браузера представляет собой «песочницу», сам браузер ~~не~~ **редко крэшится**

Зачем нам что-то ещё?

Достоинства браузерной версии:

- Браузер — **главная точка входа в соцсеть** для пользователей на десктопах
- Браузер **есть на всех десктопах**, и больше ничего не надо устанавливать
- Вкладка браузера представляет собой «песочницу», сам браузер ~~не~~ **редко крэшится**

Недостатки браузерной версии:

- **Производительность** (количество видеостримов)
- Сложно или невозможно реализовать захват экрана в 4K-разрешении, ML-обработку изображения и звука
- Сложно или невозможно реализовать режим «рации» по нажатому пробелу, звук входящего вызова в другой вкладке и т.д. (вкладка чаще всего должна быть в фокусе и кликнута)
- Браузеры **есть не на всех платформах** (Колонка)

Зачем нам что-то ещё?

Достоинства браузерной версии:

- Браузер — **главная точка входа в соцсеть** для пользователей на десктопах
- Браузер **есть на всех десктопах**, и больше ничего не надо устанавливать
- Вкладка браузера представляет собой «песочницу», сам браузер ~~не~~ **редко крэшится**

Недостатки браузерной версии:

- **Производительность** (количество видеостримов)
- Сложно или невозможно реализовать захват экрана в 4K-разрешении, ML-обработку изображения и звука
- Сложно или невозможно реализовать режим «рации» по нажатому пробелу, звук входящего вызова в другой вкладке и т.д. (вкладка чаще всего должна быть в фокусе и кликнута)
- Браузеры **есть не на всех платформах** (Колонка)

Electron — хорошая штука, но не решит все проблемы.

Звонки в браузере — подробнее

VK Звонки: выходя за лимиты браузера /
Вадим Горбачев

https://live.jugru.org/video?v=MTAwMTE3iiM3NzI5ijA

И всё-таки, зачем?

Мы поставили перед собой настоящий челлендж:

- поддержать групповые звонки на неограниченное число участников: бесконечность — не предел!
- при этом добиться максимальной производительности и качества — никаких компромиссов;

И всё-таки, зачем?

Мы поставили перед собой настоящий челлендж:

- поддержать групповые звонки на неограниченное число участников: бесконечность — не предел!
- при этом добиться максимальной производительности и качества — никаких компромиссов;
- в том числе для всех обязательных для современной видеосвязи функций:
    - демонстрация экрана в 4K;
    - шумоподавление на стороне клиента;
    - улучшение черт лица и виртуальный фон;
    - не говоря о другой продуктовой логике;

И всё-таки, зачем?

Мы поставили перед собой настоящий челлендж:

- поддержать групповые звонки на неограниченное число участников: бесконечность — не предел!
- при этом добиться максимальной производительности и качества — никаких компромиссов;
- в том числе для всех обязательных для современной видеосвязи функций:
    - демонстрация экрана в 4K;
    - шумоподавление на стороне клиента;
    - улучшение черт лица и виртуальный фон;
    - не говоря о другой продуктовой логике;
- обеспечить работу на Windows, macOS и Linux;
    - а в перспективе — предоставить нативный SDK для любых платформ;

И всё-таки, зачем?

Мы поставили перед собой настоящий челлендж:

- поддержать групповые звонки на неограниченное число участников: бесконечность — не предел!
- при этом добиться максимальной производительности и качества — никаких компромиссов;
- в том числе для всех обязательных для современной видеосвязи функций:
    - демонстрация экрана в 4K;
    - шумоподавление на стороне клиента;
    - улучшение черт лица и виртуальный фон;
    - не говоря о другой продуктовой логике;
- обеспечить работу на Windows, macOS и Linux;
    - а в перспективе — предоставить нативный SDK для любых платформ;
- и сделать это в условиях пандемии в кратчайшие сроки.

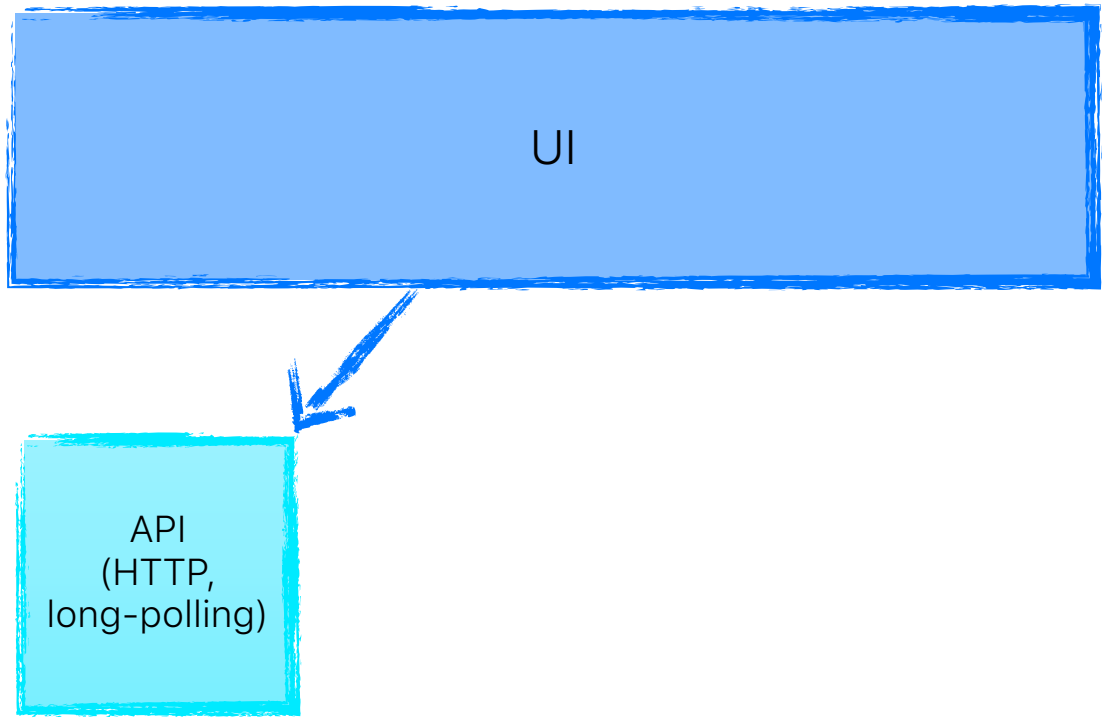
Поехали!

(*) Осторожно, будет некоторое количество C++

День 30.
Proof-of-concept.
Консольный клиент

Как установить звонок?

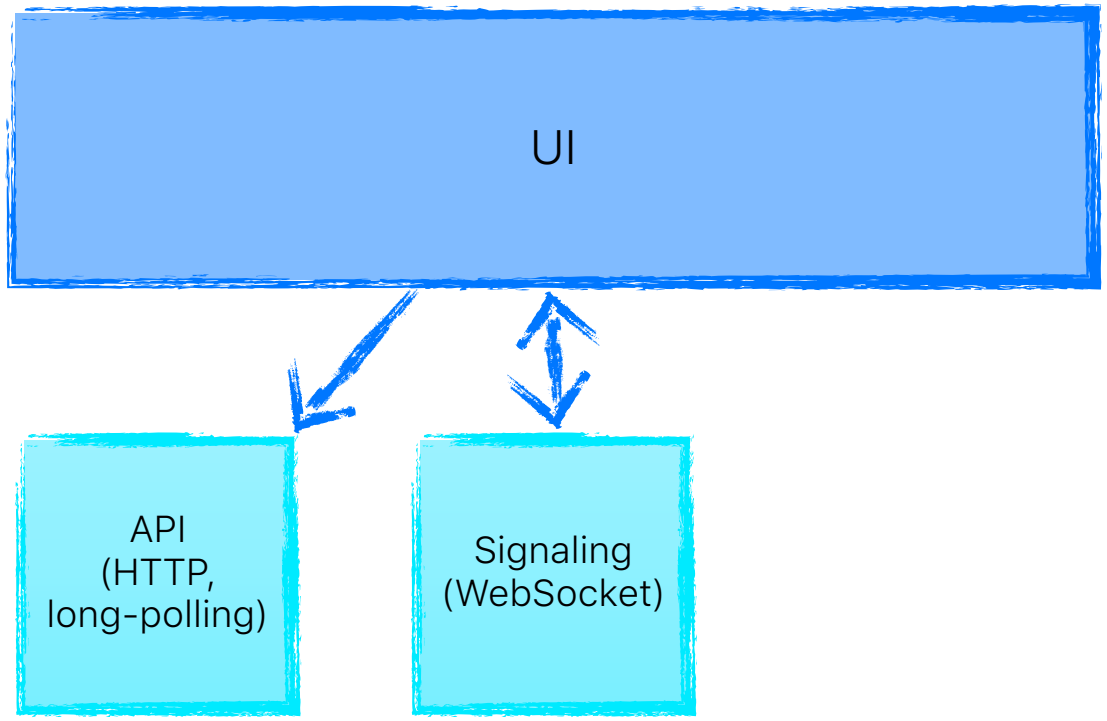
API — для логина в VK ID и получения от соцсети общей информации.



Как установить звонок?

API — для логина в VK ID и получения от соцсети общей информации.

Signaling — для обмена сообщениями о событиях в активном звонке, в т.ч. для служебной информации.

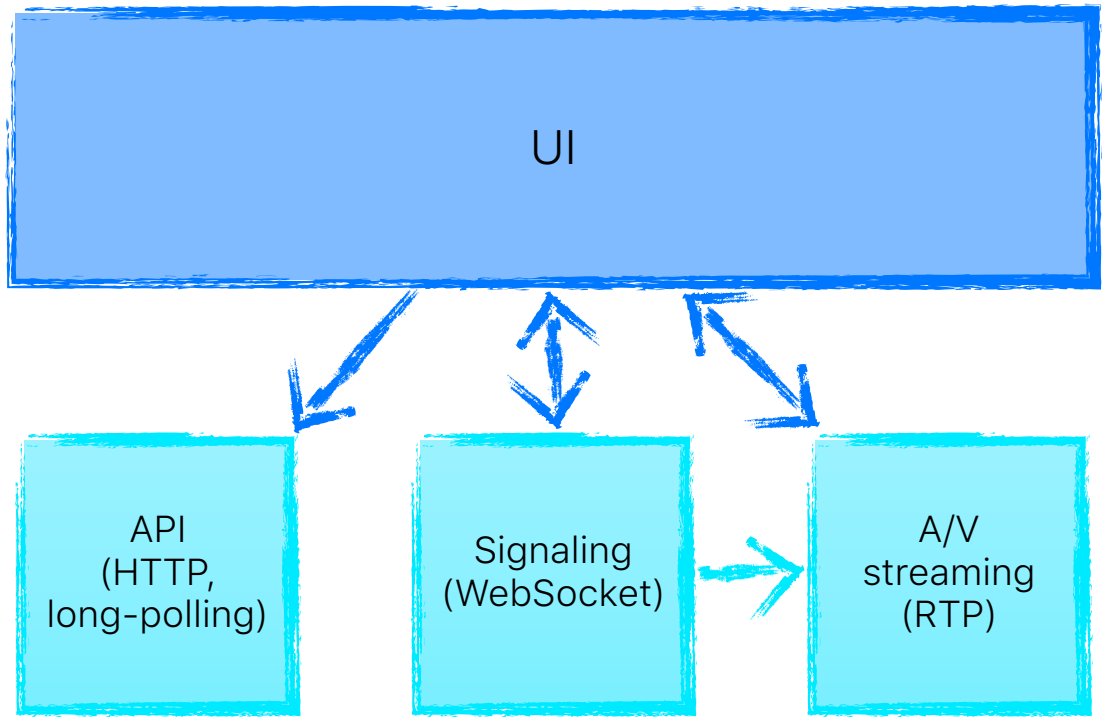


Как установить звонок?

API — для логина в VK ID и получения от соцсети общей информации.

Signaling — для обмена сообщениями о событиях в активном звонке, в т.ч. для служебной информации.

A/V streaming — для обмена аудио-видео потоками.



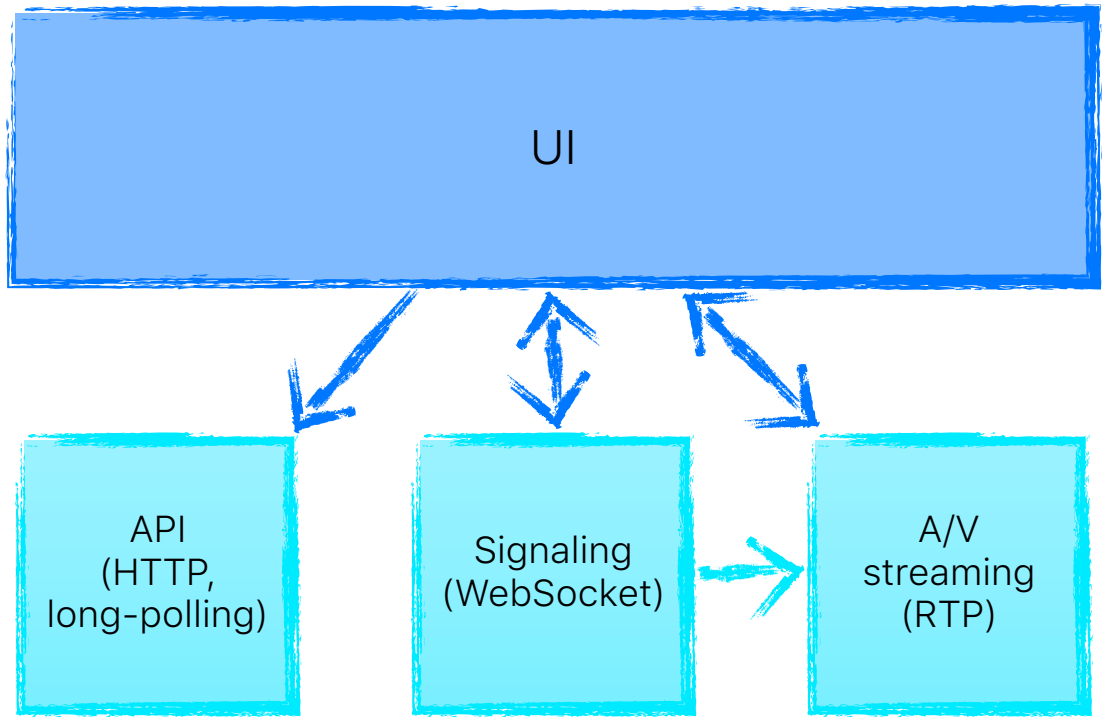
Как установить звонок?

API — для логина в VK ID и получения от соцсети общей информации.

Signaling — для обмена сообщениями о событиях в активном звонке, в т.ч. для служебной информации.

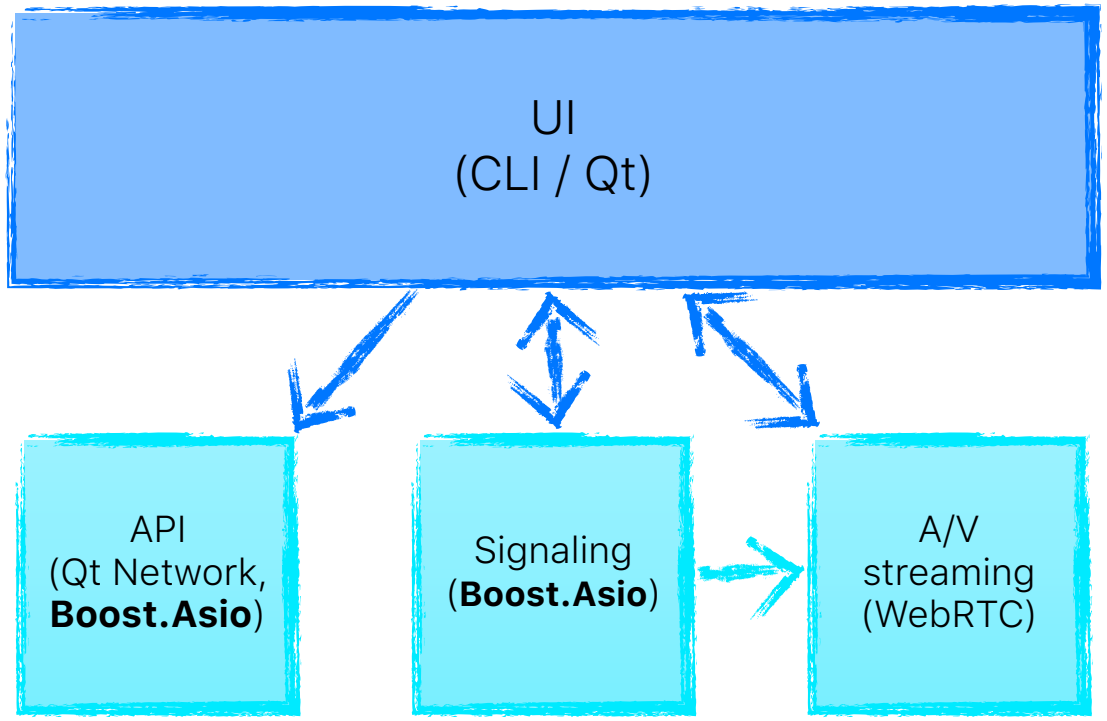
A/V streaming — для обмена аудио-видео потоками.

Достаточно поддержать то, что предоставляет бэкенд.



Из чего это сделать?

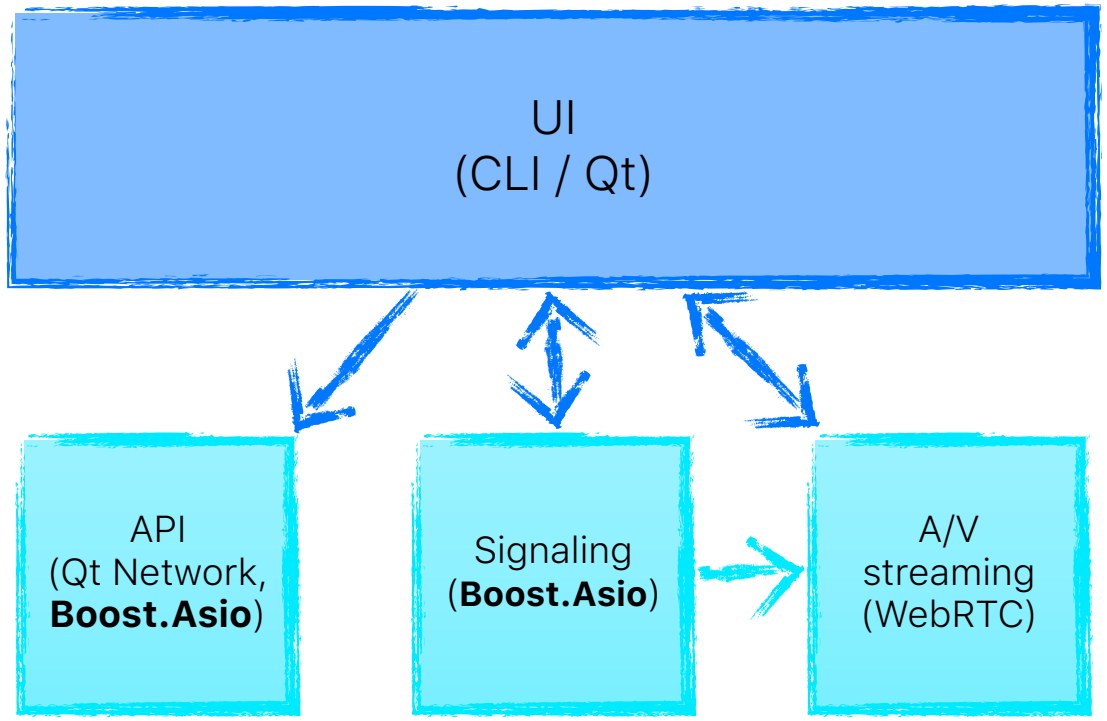
Почему Boost.Asio?



Из чего это сделать?

Почему Boost.Asio?

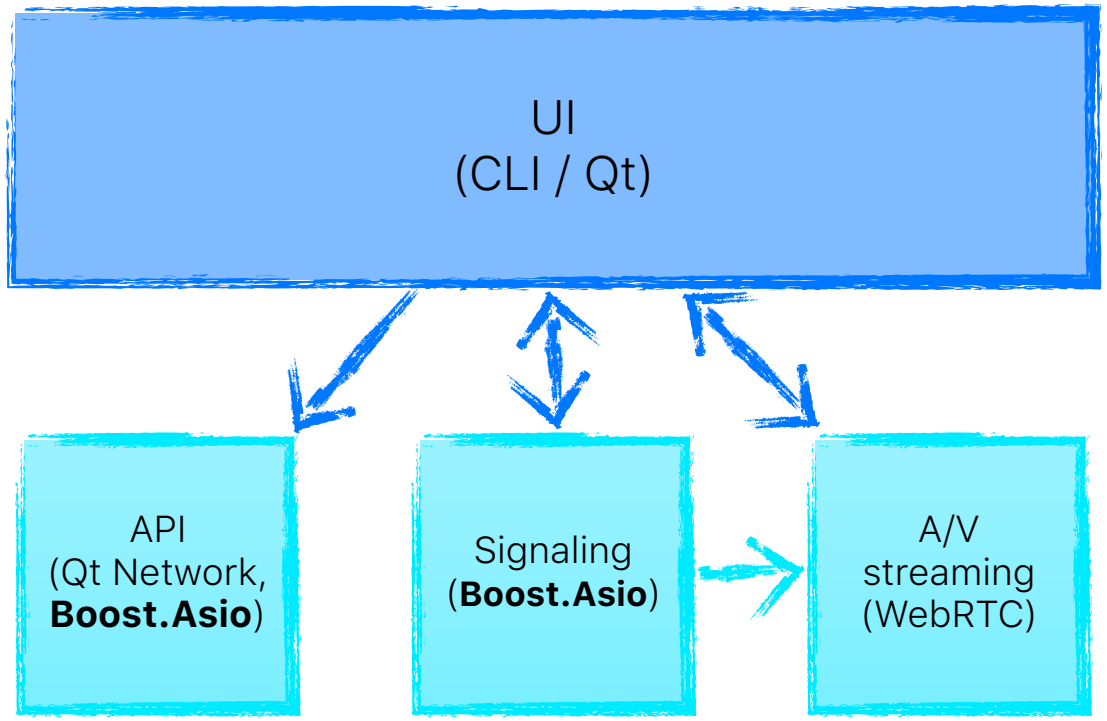
- Качество кода проверено временем (с 2003 г.)



Из чего это сделать?

Почему Boost.Asio?

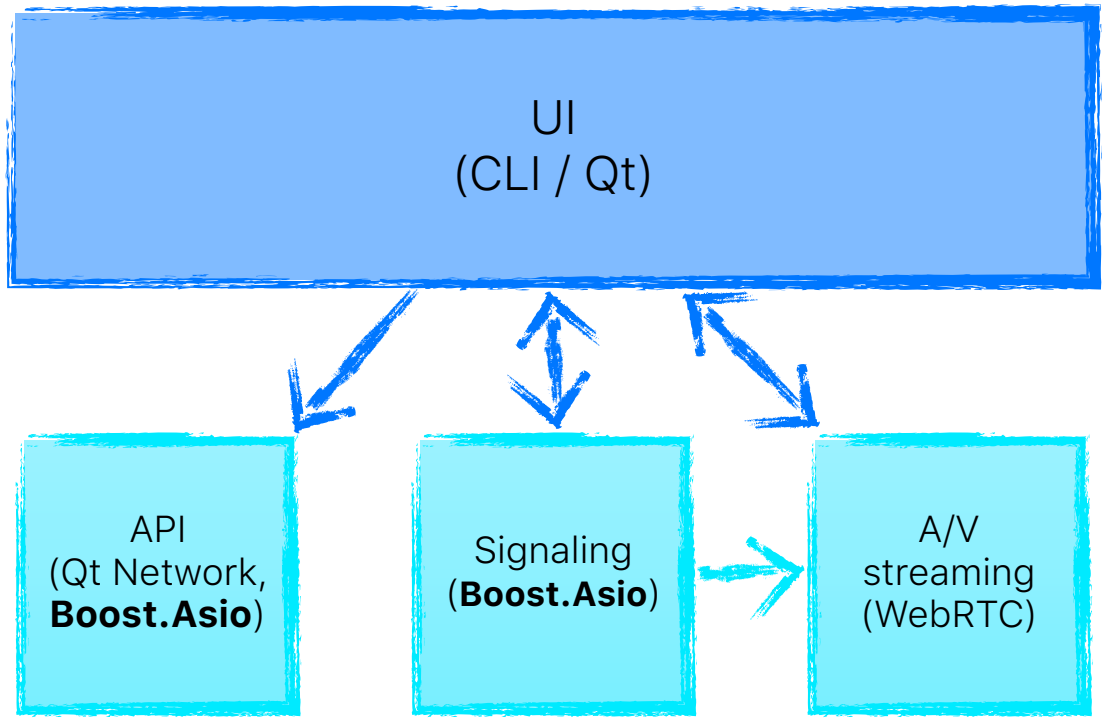
- Качество кода проверено временем (с 2003 г.)
- Хорошая документация



Из чего это сделать?

Почему Boost.Asio?

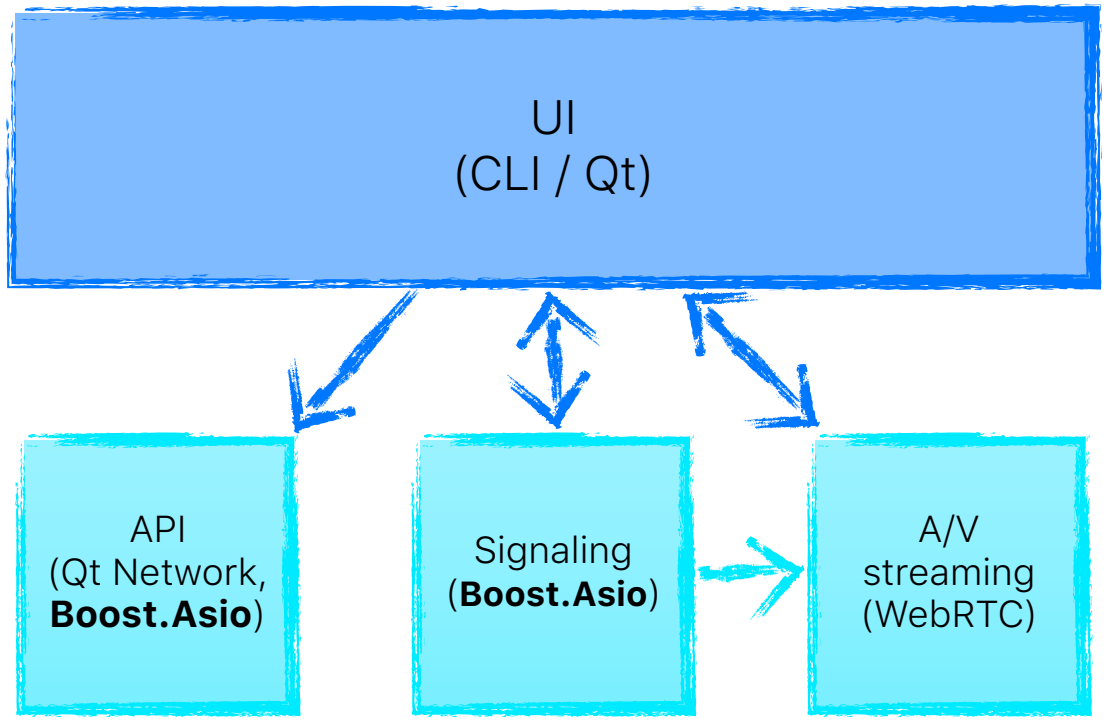
- Качество кода проверено временем (с 2003 г.)
- Хорошая документация
- С появлением Boost.Beast (в 2016 г.) писать HTTP- и WebSocket-клиенты стало намного проще



Из чего это сделать?

Почему Boost.Asio?

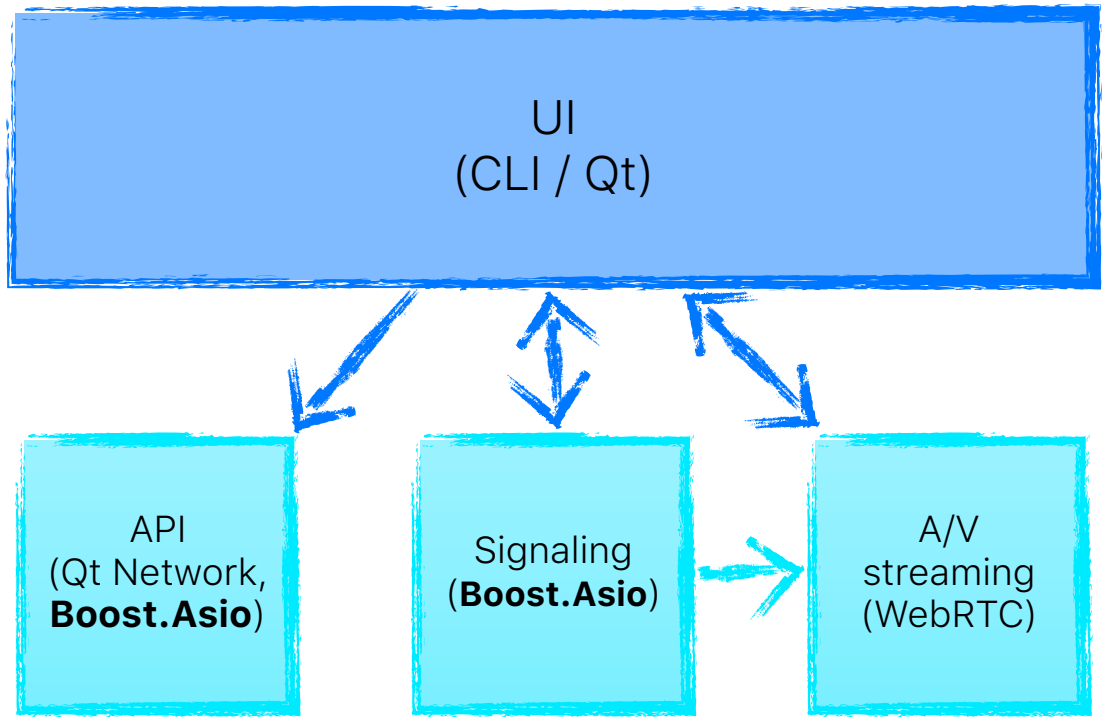
- Качество кода проверено временем (с 2003 г.)
- Хорошая документация
- С появлением Boost.Beast (в 2016 г.) писать HTTP- и WebSocket-клиенты стало намного проще
- Есть такие фичи, как permessage_deflate



Из чего это сделать?

Почему Boost.Asio?

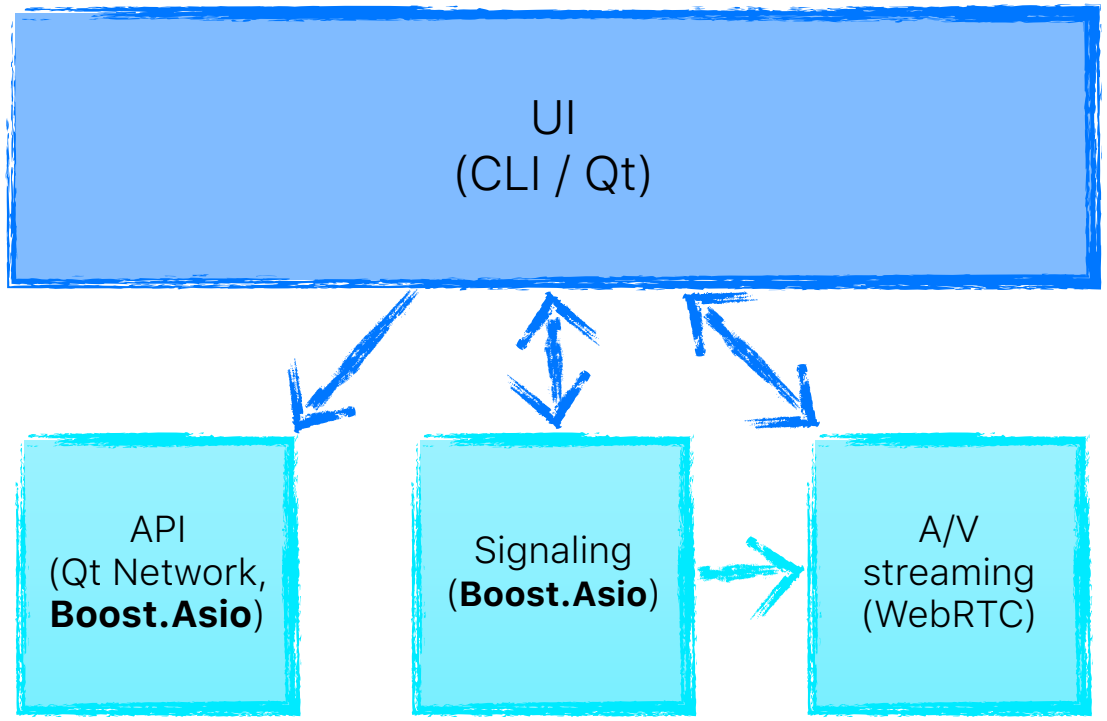
- Качество кода проверено временем (с 2003 г.)
- Хорошая документация
- С появлением Boost.Beast (в 2016 г.) писать HTTP- и WebSocket-клиенты стало намного проще
- Есть такие фичи, как permessage_deflate
- Есть корутины, не надо было ждать C++20



Из чего это сделать?

Почему Boost.Asio?

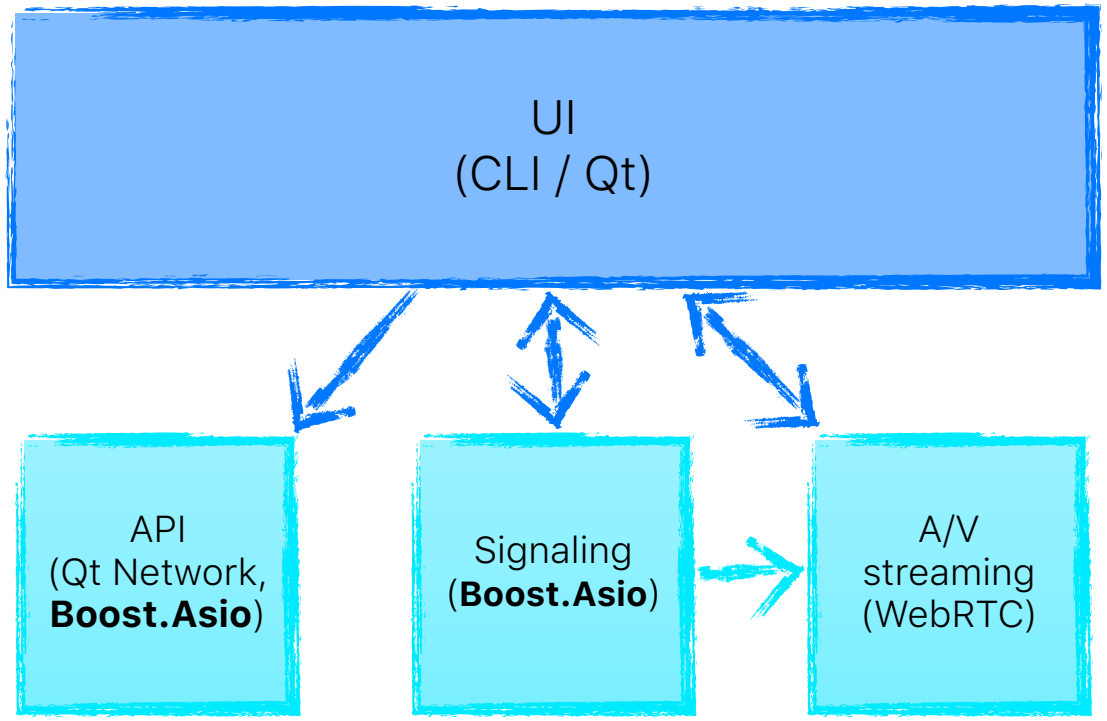
- Качество кода проверено временем (с 2003 г.)
- Хорошая документация
- С появлением Boost.Beast (в 2016 г.) писать HTTP- и WebSocket-клиенты стало намного проще
- Есть такие фичи, как permessage_deflate
- Есть корутины, не надо было ждать C++20
- ~~Войдет в C++20~~ не вошёл



Из чего это сделать?

Почему Boost.Asio?

- Качество кода проверено временем (с 2003 г.)
- Хорошая документация
- С появлением Boost.Beast (в 2016 г.) писать HTTP- и WebSocket-клиенты стало намного проще
- Есть такие фичи, как permessage_deflate
- Есть корутины, не надо было ждать C++20
- ~~Войдет в C++20~~ не вошёл



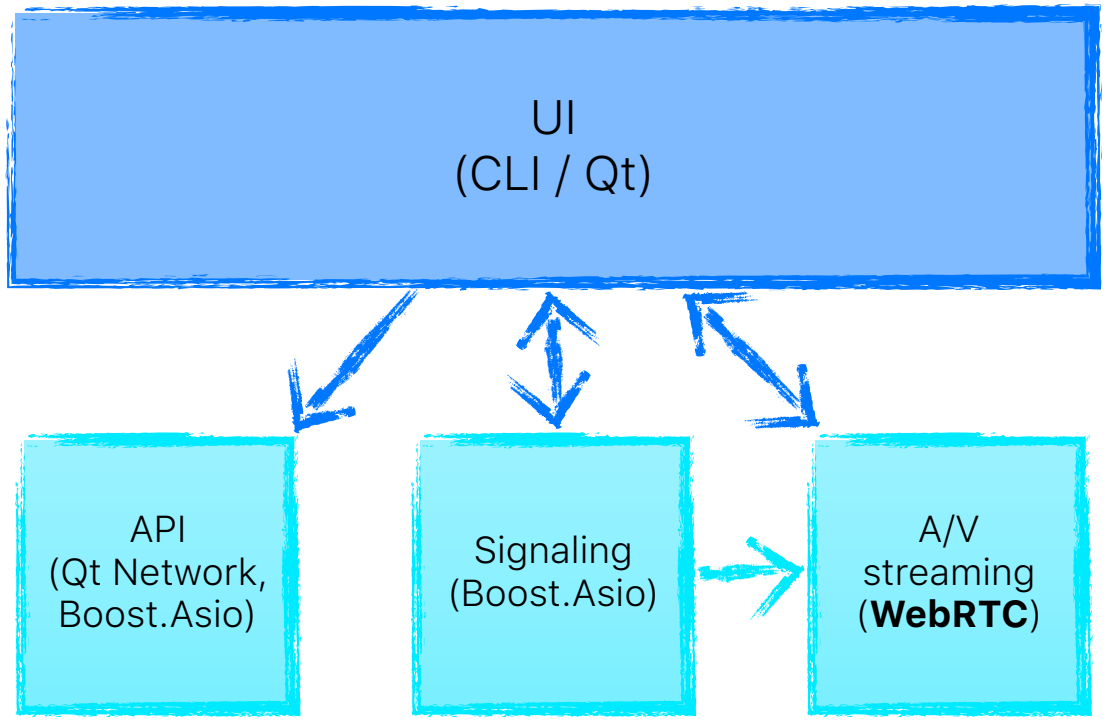
Почему не %libraryname%? Смотрели на бенчмарки

Минутка вдохновения: корутины в Boost.Beast

```
boost::future<HTTPClient::CallParams>
HTTPClient::testJoinByLink(const std::string& joinLink,
                           const std::string& name) {
    return postToQueue([this, joinLink, name](
        boost::asio::yield_context yield) {

        doOpenHttps(yield);

        auto res1 = doSendHttps<LoginRequest>(
            {name}, yield);

        const auto& token = res1.token;

        auto res2 = doSendHttps<GetParamsRequest>(
            {token}, yield);

        std::vector<api::IceServer> iceServers = {
            res2.stun_server,
            res2.turn_server
        };

        auto res3 = doSendHttps<JoinRequest>(
            {token, joinLink}, yield);

        doCloseHttps(yield);

        return HTTPClient::CallParams{
            std::move(iceServers),
            res3.signaling_url
        };
    });
}
```

Из чего это сделать?

Почему WebRTC?

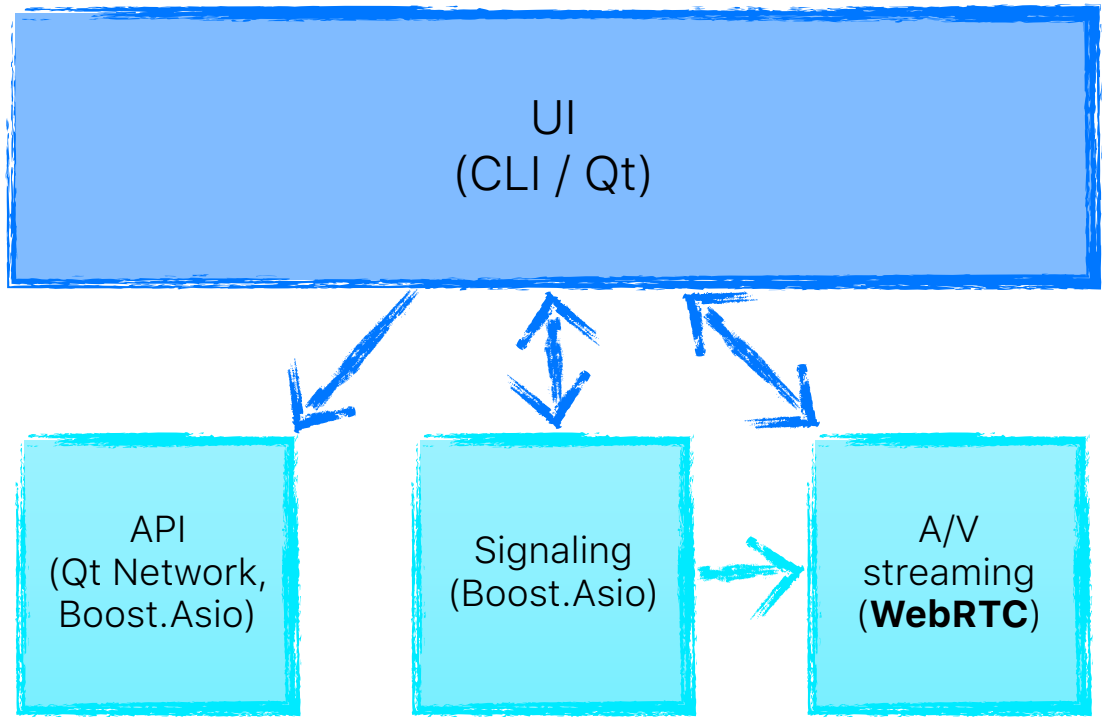


Из чего это сделать?

Почему WebRTC?

Без вариантов:

- Уже используем в браузерах и мобилках
- Стандарт де-факто для браузера



Что там с нативным WebRTC?

«Технологии, лежащие в основе WebRTC, реализованы как открытый веб-стандарт и доступны в виде обычных API-интерфейсов JavaScript во всех основных браузерах»

Что там с нативным WebRTC?

«Технологии, лежащие в основе WebRTC, реализованы как открытый веб-стандарт и доступны в виде обычных API-интерфейсов JavaScript во всех основных браузерах»

- Хотите попробовать? К вашим услугам:
  - простой Get Started гайд на JavaScript
  - подробная документация от Mozilla

Что там с нативным WebRTC?

«Технологии, лежащие в основе WebRTC, реализованы как открытый веб-стандарт и доступны в виде обычных API-интерфейсов JavaScript во всех основных браузерах»

- Хотите попробовать? К вашим услугам:
  - простой Get Started гайд на JavaScript
  - подробная документация от Mozilla

«Для собственных клиентов, таких как приложения для Android и iOS, доступна библиотека, обеспечивающая те же функции»

Что там с нативным WebRTC?

«Технологии, лежащие в основе WebRTC, реализованы как открытый веб-стандарт и доступны в виде обычных API-интерфейсов JavaScript во всех основных браузерах»

- Хотите попробовать? К вашим услугам:
  - простой Get Started гайд на JavaScript
  - подробная документация от Mozilla

«Для собственных клиентов, таких как приложения для Android и iOS, доступна библиотека, обеспечивающая те же функции»

- Хотите попробовать?
  - сначала соберите нативную библиотеку из исходников

Что там с нативным WebRTC?

«Технологии, лежащие в основе WebRTC, реализованы как открытый веб-стандарт и доступны в виде обычных API-интерфейсов JavaScript во всех основных браузерах»

- Хотите попробовать? К вашим услугам:
  - простой Get Started гайд на JavaScript
  - подробная документация от Mozilla

«Для собственных клиентов, таких как приложения для Android и iOS, доступна библиотека, обеспечивающая те же функции»

- Хотите попробовать?
  - сначала соберите нативную библиотеку из исходников
  - потом разберитесь с примером "peerconnection", общим для WinAPI и X.org (благодаря #ifdef)

Что там с нативным WebRTC?

«Технологии, лежащие в основе WebRTC, реализованы как открытый веб-стандарт и доступны в виде обычных API-интерфейсов JavaScript во всех основных браузерах»

- Хотите попробовать? К вашим услугам:
  - простой Get Started гайд на JavaScript
  - подробная документация от Mozilla

«Для собственных клиентов, таких как приложения для Android и iOS, доступна библиотека, обеспечивающая те же функции»

- Хотите попробовать?
  - сначала соберите нативную библиотеку из исходников
  - потом разберитесь с примером "peerconnection", общим для WinAPI и X.org (благодаря #ifdef)
  - хотя документация есть в некоторых заголовочных файлах, будьте готовы читать код

Что там с нативным WebRTC?

«Технологии, лежащие в основе WebRTC, реализованы как открытый веб-стандарт и доступны в виде обычных API-интерфейсов JavaScript во всех основных браузерах»

- Хотите попробовать? К вашим услугам:
  - простой Get Started гайд на JavaScript
  - подробная документация от Mozilla

«Для собственных клиентов, таких как приложения для Android и iOS, доступна библиотека, обеспечивающая те же функции»

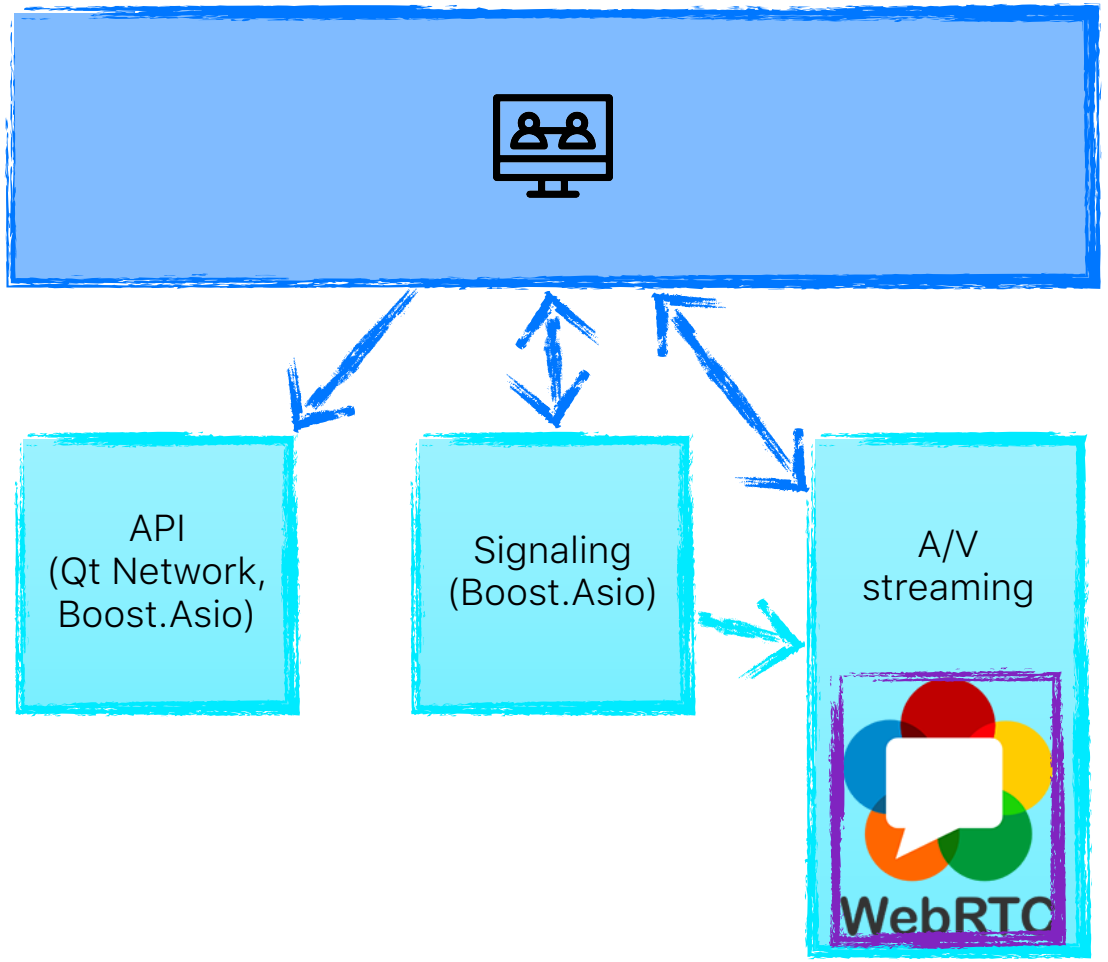
- Хотите попробовать?
  - сначала соберите нативную библиотеку из исходников
  - потом разберитесь с примером "peerconnection", общим для WinAPI и X.org (благодаря #ifdef)
  - хотя документация есть в некоторых заголовочных файлах, будьте готовы читать код
  - ...или искать ответы в Google-группе discuss-webrtc

Что там с WebRTC?

Выводы после поверхностного изучения примера:

- Под все платформы — Clang
- Под Windows — также MSVC (*)
- Под Linux — также GCC (*)

(*) в обоих случаях надо патчить скрипты



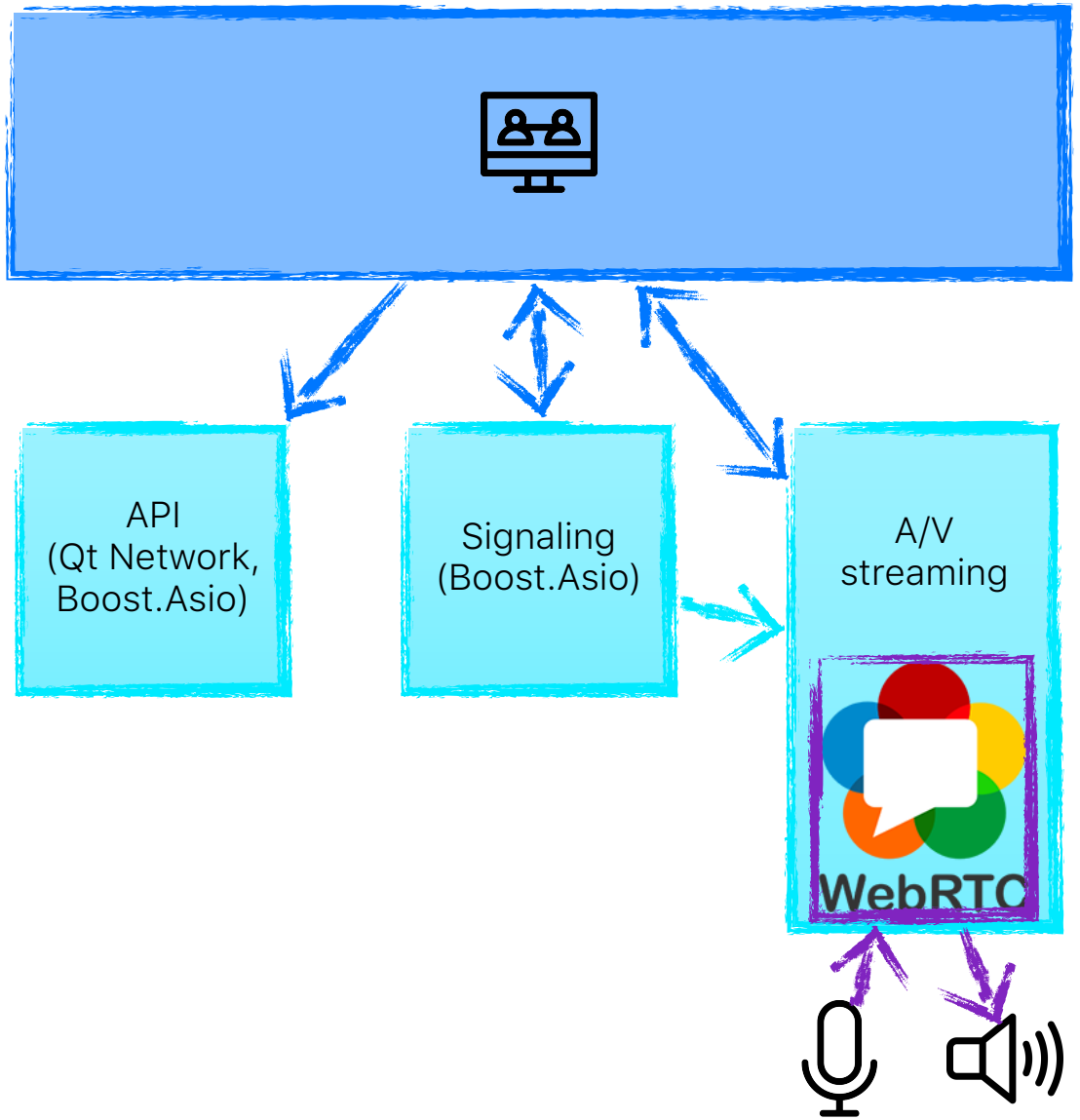
Что там с WebRTC?

Выводы после поверхностного изучения примера:

- Под все платформы — Clang
- Под Windows — также MSVC (*)
- Под Linux — также GCC (*)

  (*) в обоих случаях надо патчить скрипты

- Захват микрофона и воспроизведение звука работают из коробки для всех платформ — ура!



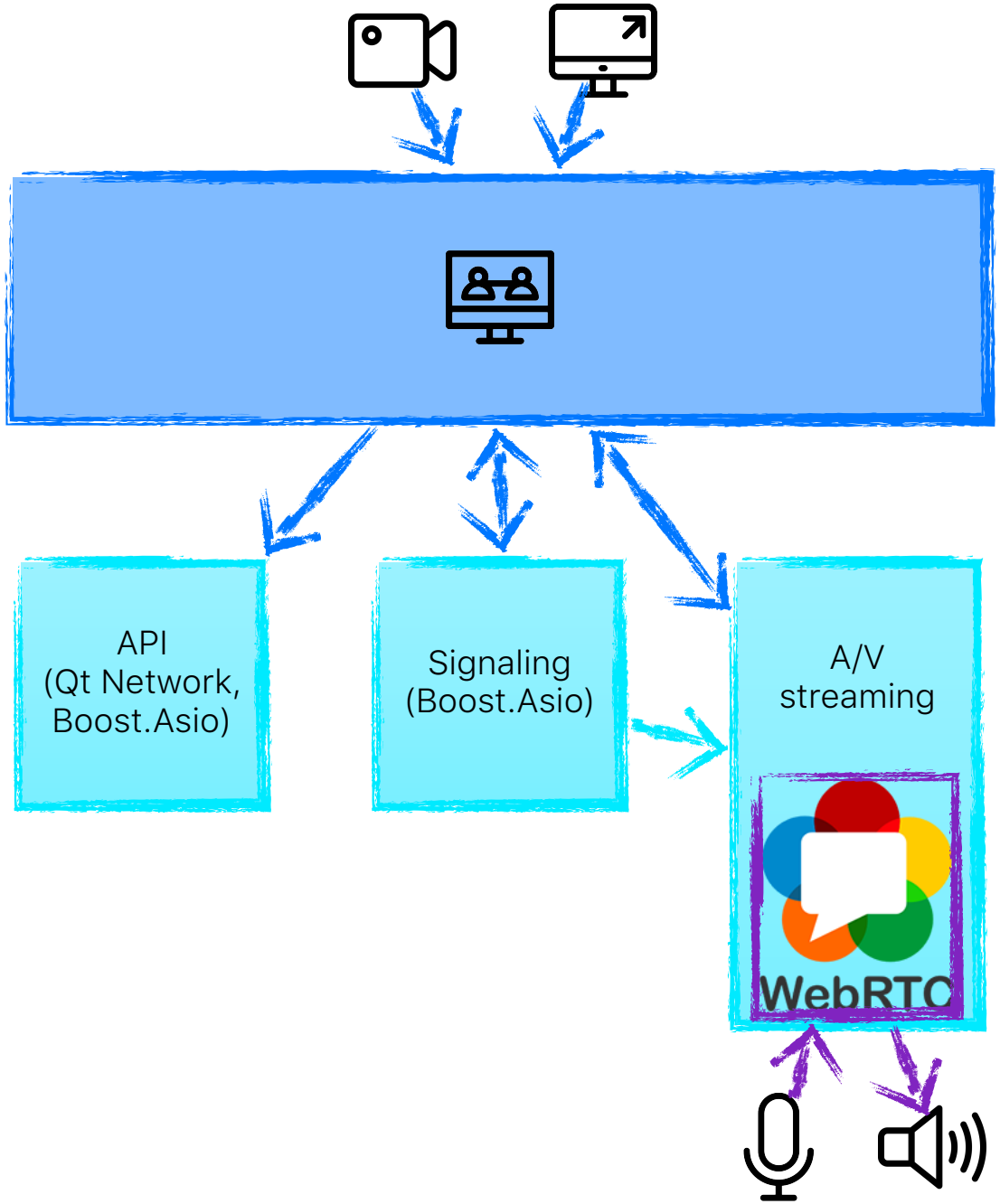
Что там с WebRTC?

Выводы после поверхностного изучения примера:

- Под все платформы — Clang
- Под Windows — также MSVC (*)
- Под Linux — также GCC (*)

  (*) в обоих случаях надо патчить скрипты

- Захват микрофона и воспроизведение звука работают из коробки для всех платформ — ура!
- Захват камеры не работает на macOS — увы, придется искать другое кросс-платформенное решение.



Что не так с нативным WebRTC?

Сборка нативной библиотеки WebRTC из исходников — не самое приятное занятие:

Что не так с нативным WebRTC?

Сборка нативной библиотеки WebRTC из исходников — не самое приятное занятие:

- Для чекаута исходников нужен depot_tools
  - Исходники занимают 13G, качаются часами, сборка длится десятки минут
  - Для WebRTC может не подойти depot_tools
  - Для depot_tools может не подойти Python

Что не так с нативным WebRTC?

Сборка нативной библиотеки WebRTC из исходников — не самое приятное занятие:

- Для чекаута исходников нужен depot_tools
  - Исходники занимают 13G, качаются часами, сборка длится десятки минут
  - Для WebRTC может не подойти depot_tools
  - Для depot_tools может не подойти Python
- Для управления сборкой используется GN
  - Мало распространена за пределами проектов Google

Что не так с нативным WebRTC?

Сборка нативной библиотеки WebRTC из исходников — не самое приятное занятие:

- Для чекаута исходников нужен depot_tools
  - Исходники занимают 13G, качаются часами, сборка длится десятки минут
  - Для WebRTC может не подойти depot_tools
  - Для depot_tools может не подойти Python
- Для управления сборкой используется GN
  - Мало распространена за пределами проектов Google

Всё это вызывает сложности при попытке встроить сборку WebRTC в собственный проект.

Что не так с нативным WebRTC?

Сборка нативной библиотеки WebRTC из исходников — не самое приятное занятие:

- Для чекаута исходников нужен depot_tools
  - Исходники занимают 13G, качаются часами, сборка длится десятки минут
  - Для WebRTC может не подойти depot_tools
  - Для depot_tools может не подойти Python
- Для управления сборкой используется GN
  - Мало распространена за пределами проектов Google

Всё это вызывает сложности при попытке встроить сборку WebRTC в собственный проект.

- На гитхабе есть набор cmake-скриптов, но он заброшен в 2017г. и не совместим со свежим WebRTC
- Их автор занимается предоставлением сборок WebRTC и свежих скриптов на коммерческой основе

Что не так с нативным WebRTC?

Основной класс для работы с аудио- и видеопотоками — webrtc::PeerConnection.

Для создания объекта нужно 3 потока:

- signaling; worker; network

Что не так с нативным WebRTC?

Основной класс для работы с аудио- и видеопотоками — webrtc::PeerConnection.

Для создания объекта нужно 3 потока:

- signaling; worker; network

Класс rtc::Thread — основной механизм для защиты данных в многопоточной среде.

Что не так с нативным WebRTC?

Основной класс для работы с аудио- и видеопотоками — webrtc::PeerConnection.

Для создания объекта нужно 3 потока:

- signaling; worker; network

Класс rtc::Thread — основной механизм для защиты данных в многопоточной среде.

Но:

- webrtc::PeerConnection может породить собственные треды
    - конкретно — в аудио- и видеообработчиках

Что не так с нативным WebRTC?

Основной класс для работы с аудио- и видеопотоками — webrtc::PeerConnection.

Для создания объекта нужно 3 потока:

- signaling; worker; network

Класс rtc::Thread — основной механизм для защиты данных в многопоточной среде.

Но:

- webrtc::PeerConnection может породить собственные треды
  - конкретно — в аудио- и видеообработчиках
- нет интерфейса для управления ими

Что не так с нативным WebRTC?

Основной класс для работы с аудио- и видеопотоками — webrtc::PeerConnection.

Для создания объекта нужно 3 потока:

- signaling; worker; network

Класс rtc::Thread — основной механизм для защиты данных в многопоточной среде.

Но:

- webrtc::PeerConnection может породить собственные треды
  - конкретно — в аудио- и видеообработчиках
- нет интерфейса для управления ими
- и они не останавливаются в деструкторе

Что не так с нативным WebRTC?

Основной класс для работы с аудио- и видеопотоками — webrtc::PeerConnection.

Для создания объекта нужно 3 потока:

- signaling; worker; network

Класс rtc::Thread — основной механизм для защиты данных в многопоточной среде.

Но:

- webrtc::PeerConnection может породить собственные треды
  - конкретно — в аудио- и видеообработчиках
- нет интерфейса для управления ими
- и они не останавливаются в деструкторе

Поэтому все равно надо следить за временем жизни и защитой данных.

Минутка вдохновения: статический анализ в Clang

```
class SessionMock :
    public std::enable_shared_from_this<SessionMock>
{
private:
    void doSetOffer(const std::string& offer)
        RTC_EXCLUSIVE_LOCKS_REQUIRED(signalingThread);

    void doCreateAnswer()
        RTC_EXCLUSIVE_LOCKS_REQUIRED(signalingThread);

    void doSomething();

private:
    rtc::Thread* signalingThread;

    rtc::scoped_refptr<webrtc::Observer> offerObserver
        RTC_GUARDED_BY(signalingThread);

    rtc::scoped_refptr<webrtc::Observer> answerObserver
        RTC_GUARDED_BY(signalingThread);
};

void SessionMock::doSomething()
{
   RTC_DCHECK_RUN_ON(signalingThread);
   session->doCreateAnswer();
}

// README: Thread-Safety Analysis
```

Что не так с нативным WebRTC?

Стайлгайд Chromium рекомендует пользоваться 3 типами умных указателей:

- std::unique_ptr<>; rtc::scoped_refptr<>; WeakPtr<>

Что не так с нативным WebRTC?

Стайлгайд Chromium рекомендует пользоваться 3 типами умных указателей:

- std::unique_ptr<>; rtc::scoped_refptr<>; WeakPtr<>

Публичный интерфейс WebRTC до сих пор во многих случаях использует **сырые** указатели.

Что не так с нативным WebRTC?

Стайлгайд Chromium рекомендует пользоваться 3 типами умных указателей:

- std::unique_ptr<>; rtc::scoped_refptr<>; WeakPtr<>

Публичный интерфейс WebRTC до сих пор во многих случаях использует **сырые** указатели.

В примере демонстрируется **единый god-object**, который наследует от нескольких интерфейсов.

Что не так с нативным WebRTC?

Стайлгайд Chromium рекомендует пользоваться 3 типами умных указателей:

- std::unique_ptr<>; rtc::scoped_refptr<>; WeakPtr<>

Публичный интерфейс WebRTC до сих пор во многих случаях использует **сырые** указатели.

В примере демонстрируется **единый god-object**, который наследует от нескольких интерфейсов.

В реальной жизни оказывается невозможным унаследоваться от всех необходимых интерфейсов:

- либо несовместимы сами интерфейсы,
- либо отличаются типы указателей в тех методах, куда их надо передавать.

Что не так с нативным WebRTC?

Стайлгайд Chromium рекомендует пользоваться 3 типами умных указателей:

- std::unique_ptr<>; rtc::scoped_refptr<>; WeakPtr<>

Публичный интерфейс WebRTC до сих пор во многих случаях использует **сырые** указатели.

В примере демонстрируется **единый god-object**, который наследует от нескольких интерфейсов.

В реальной жизни оказывается невозможным унаследоваться от всех необходимых интерфейсов:

- либо несовместимы сами интерфейсы,
- либо отличаются типы указателей в тех методах, куда их надо передавать.

Мы выбрали такой путь: спрятать множество прокси-объектов за общим фасадом.

Минутка вдохновения: асинхронные цепочки вызовов в объектах WebRTC

```
void SessionMock::doSetOffer(const std::string& offer)
{
    webrtc::SdpParseError error;
    auto sdp = webrtc::CreateSessionDescription(
        webrtc::SdpType::kOffer, offer, &error);

    struct Observer
        : public SetRemoteDescriptionObserverInterface
    {
        Observer(std::weak_ptr<SessionMock> owner);

        void OnSetRemoteDescriptionComplete(
            webrtc::RTCError error) override
        {
            auto session = this->owner.lock();
            if (!session)
                return;

            RTC_DCHECK_RUN_ON(ref->signalingThread);

            if (error.ok())
                session->doCreateAnswer(key);
        }
    };

    if (!this->offerObserver)
        this->offerObserver =
            new Observer{weak_from_this()};

    peer->SetRemoteDescription(
        std::move(sdp), this->offerObserver);
}
```

Что ещё не так с нативным WebRTC?

- Сам WebRTC на текущий момент обязательно должен компилироваться с C++14.

Что ещё не так с нативным WebRTC?

- Сам WebRTC на текущий момент обязательно должен компилироваться с C++14.
- Ничего не должно мешать использовать его заголовки в нашем проекте с C++17/20.

Что ещё не так с нативным WebRTC?

- Сам WebRTC на текущий момент обязательно должен компилироваться с C++14.
- Ничего не должно мешать использовать его заголовки в нашем проекте с C++17/20.
- Но WebRTC внутри себя очень активно использует библиотеку Abseil.

Что ещё не так с нативным WebRTC?

- Сам WebRTC на текущий момент обязательно должен компилироваться с C++14.
- Ничего не должно мешать использовать его заголовки в нашем проекте с C++17/20.
- Но WebRTC внутри себя очень активно использует библиотеку Abseil.
- А Abseil, по замыслу создателей, должен был бороться с несоответствием стандартов, а не...

А что не так с Abseil?

```
////////////////////////////////// absl/base/config.h

#if defined(_MSC_VER) && _MSC_VER >= 1910 &&         \
    ((defined(_MSVC_LANG) && _MSVC_LANG > 201402) || \
     (defined(__cplusplus) && __cplusplus > 201402))

#define ABSL_USES_STD_OPTIONAL 1

#endif

////////////////////////////////// absl/types/optional.h

#include "absl/base/config.h"

#ifdef ABSL_USES_STD_OPTIONAL

#include <optional>

namespace absl {
ABSL_NAMESPACE_BEGIN
using std::bad_optional_access;
using std::optional;
using std::make_optional;
using std::nullopt_t;
using std::nullopt;
ABSL_NAMESPACE_END
}  // namespace absl

#else  // ABSL_USES_STD_OPTIONAL

#include "absl/types/internal/optional.h"

#endif  // ABSL_USES_STD_OPTIONAL
```

ABI-несовместимость!

Минутка просветления: фикс для конфигурации Abseil

```
//////////////////////////////////// abseil_config_fix.h

#pragma once

#include <absl/base/config.h>

#undef ABSL_USES_STD_ANY
#undef ABSL_USES_STD_OPTIONAL
#undef ABSL_USES_STD_VARIANT
#undef ABSL_USES_STD_STRING_VIEW

#include <absl/types/any.h>
#include <absl/types/optional.h>
#include <absl/types/variant.h>
#include <absl/strings/string_view.h>
```

А как же JSON?

Сообщения API и сигналинга — в JSON.

Обычно работа с ними сводится к куче boilerplate-кода.

А как же JSON?

Сообщения API и сигналинга — в JSON.

Обычно работа с ними сводится к куче boilerplate-кода.

Мы с помощью одного-единственного макроса:

- Генерим описание структуры с полями нужных типов
- Генерим сериализатор и десериализатор JSON
- Рефлексия — через Boost.Hana
- Под капотом — rapidJSON

А как же JSON?

Сообщения API и сигналинга — в JSON.

Обычно работа с ними сводится к куче boilerplate-кода.

Мы с помощью одного-единственного макроса:

- Генерим описание структуры с полями нужных типов
- Генерим сериализатор и десериализатор JSON
- Рефлексия — через Boost.Hana
- Под капотом — rapidJSON

В отличие от известных решений, полностью DRY.

А как же JSON?

Сообщения API и сигналинга — в JSON.

Обычно работа с ними сводится к куче boilerplate-кода.

Мы с помощью одного-единственного макроса:

- Генерим описание структуры с полями нужных типов
- Генерим сериализатор и десериализатор JSON
- Рефлексия — через Boost.Hana
- Под капотом — rapidJSON

В отличие от известных решений, полностью DRY.

Почему не protobuf?

- Усложнится сборка
- Не будет контроля над сгенеренным кодом
- Потеряем в производительности

Минутка вдохновения: работа с JSON

```
struct TestStruct {
    DEFINE_MESSAGE_MEMBERS(
        TestStruct, void,  // no base class
        (int32_t, i),
        (std::string, s),
        (std::optional<int32_t>, opt_i),
        (std::optional<std::string>, opt_s)
    );
};

TEST(TestStructSuite, TestFromJSON) {
    std::string input =
R"({
    "i": 1,
    "s": "two"
})";

    auto result = TestStruct::fromJSON(input);
    EXPECT_EQ(1, result.i);
    EXPECT_EQ("two", result.s);
    EXPECT_EQ(std::nullopt, result.opt_i);
    EXPECT_EQ(std::nullopt, result.opt_s);
}
```

День 30. Proof-of-concept — итоги

- Нативный WebRTC — тот еще фрукт

День 30. Proof-of-concept — итоги

- Нативный WebRTC — тот еще фрукт
- Сделать аудиозвонки можно быстро

День 30. Proof-of-concept — итоги

- Нативный WebRTC — тот еще фрукт
- Сделать аудиозвонки можно быстро
- Сделать видеозвонки мы тоже сможем!

День 30. Proof-of-concept — итоги

- Нативный WebRTC — тот еще фрукт
- Сделать аудиозвонки можно быстро
- Сделать видеозвонки мы тоже сможем!
- Заложена хорошая кодовая база

День 100. Minimum Viable Prototype. Qt-клиент

День 100. Minimum Viable Prototype. Qt-клиент

1. UI-фреймворк

День 100. Minimum Viable Prototype. Qt-клиент

1. UI-фреймворк
2. Управление зависимостями

День 100. Minimum Viable Prototype. Qt-клиент

1. UI-фреймворк
2. Управление зависимостями
3. Захват камеры

День 100. Minimum Viable Prototype. Qt-клиент

1. UI-фреймворк
2. Управление зависимостями
3. Захват камеры
4. Захват экрана

День 100. Minimum Viable Prototype. Qt-клиент

1. UI-фреймворк
2. Управление зависимостями
3. Захват камеры
4. Захват экрана
5. Подавление шума

День 100. Minimum Viable Prototype. Qt-клиент

1. UI-фреймворк
2. Управление зависимостями
3. Захват камеры
4. Захват экрана
5. Подавление шума
6. Автотесты

День 100. Minimum Viable Prototype. Qt-клиент

1. UI-фреймворк
2. Управление зависимостями
3. Захват камеры
4. Захват экрана
5. Подавление шума
6. Автотесты
7. Автообновление

UI-фреймворк

Qt: QWidgets vs QML

Почему Qt?

- Наиболее активно развивающийся и самый популярный
- Лучшая документация, что только можно представить
- У нас уже была экспертиза

Qt: QWidgets vs QML

Почему Qt?

- Наиболее активно развивающийся и самый популярный
- Лучшая документация, что только можно представить
- У нас уже была экспертиза

QWidgets — это набор C++ классов для построения UI

QML — это JavaScript-подобная обвязка вокруг Qt-классов

Qt: QWidgets vs QML

Почему Qt?

- Наиболее активно развивающийся и самый популярный
- Лучшая документация, что только можно представить
- У нас уже была экспертиза

QWidgets — это набор C++ классов для построения UI

QML — это JavaScript-подобная обвязка вокруг Qt-классов

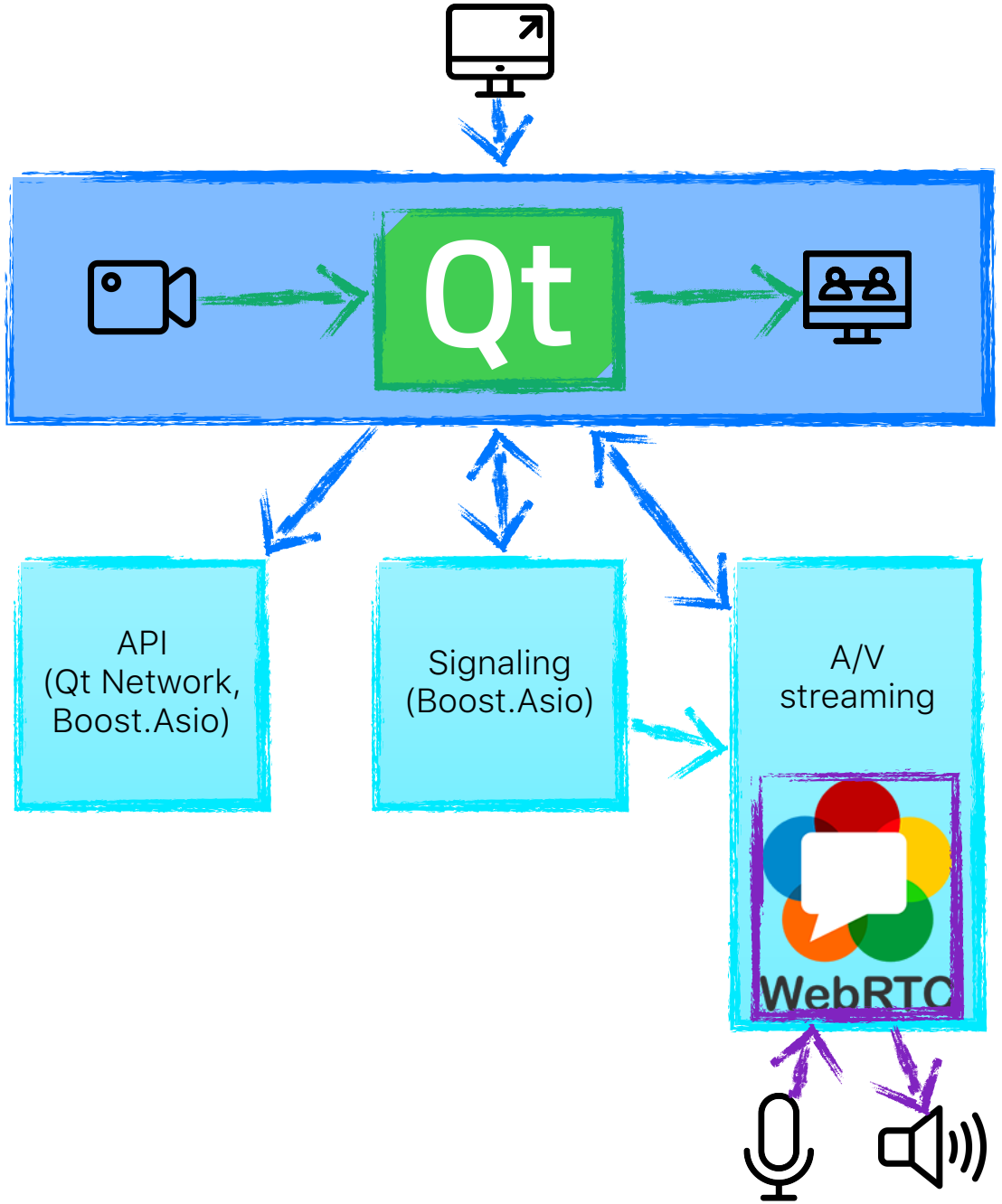
Почему QWidgets?

- У нас уже была экспертиза, ~~и вообще мы плюсовики~~
- Нам нужно будет делать много кастомных виджетов
- Подключение их через QML будет тормозить разработку
- Ходят слухи, что QML работает менее эффективно

Захват камеры

Захват камеры

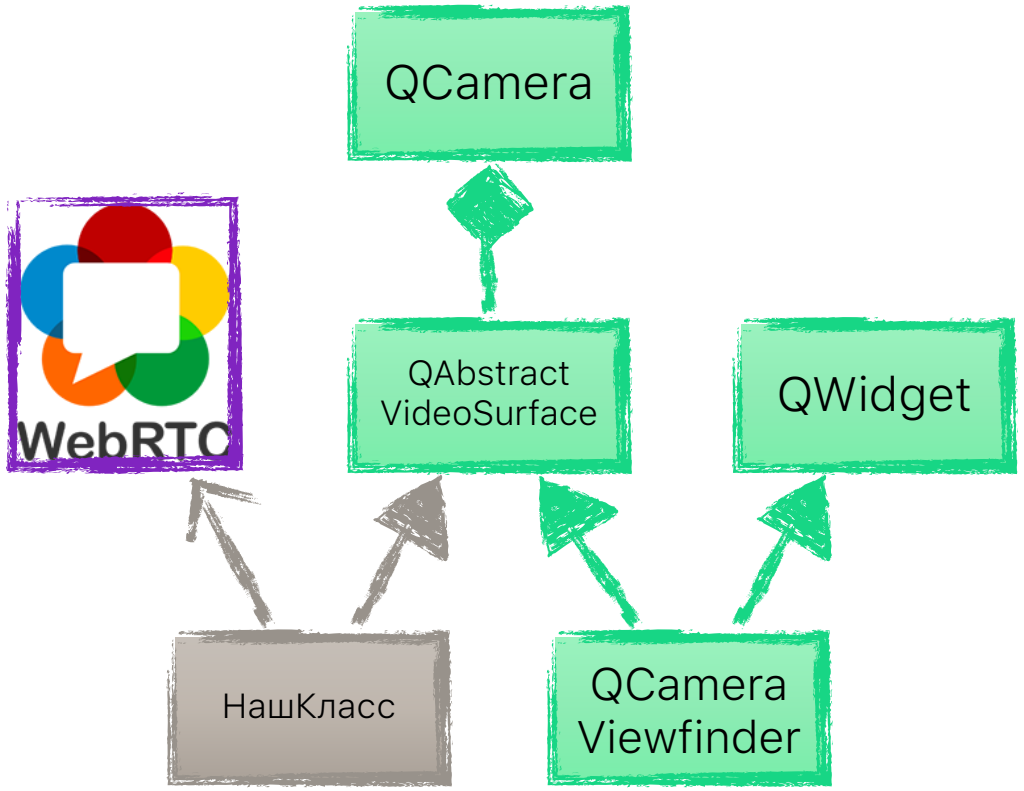
Почему Qt Multimedia? Она поддерживает камеры из коробки! Попробуем... It works!



Захват камеры

Выводы после изучения примеров и первоначальной реализации:

- QCamera и QCameraViewfinder поддерживают все распространенные модели камер
- Если наследоваться от QAbstractVideoSurface, то можно получать кадры в виде QVideoFrame
- Но метаинформацию очень трудно обобщить на все кейсы («звонок другу из Австралии»)



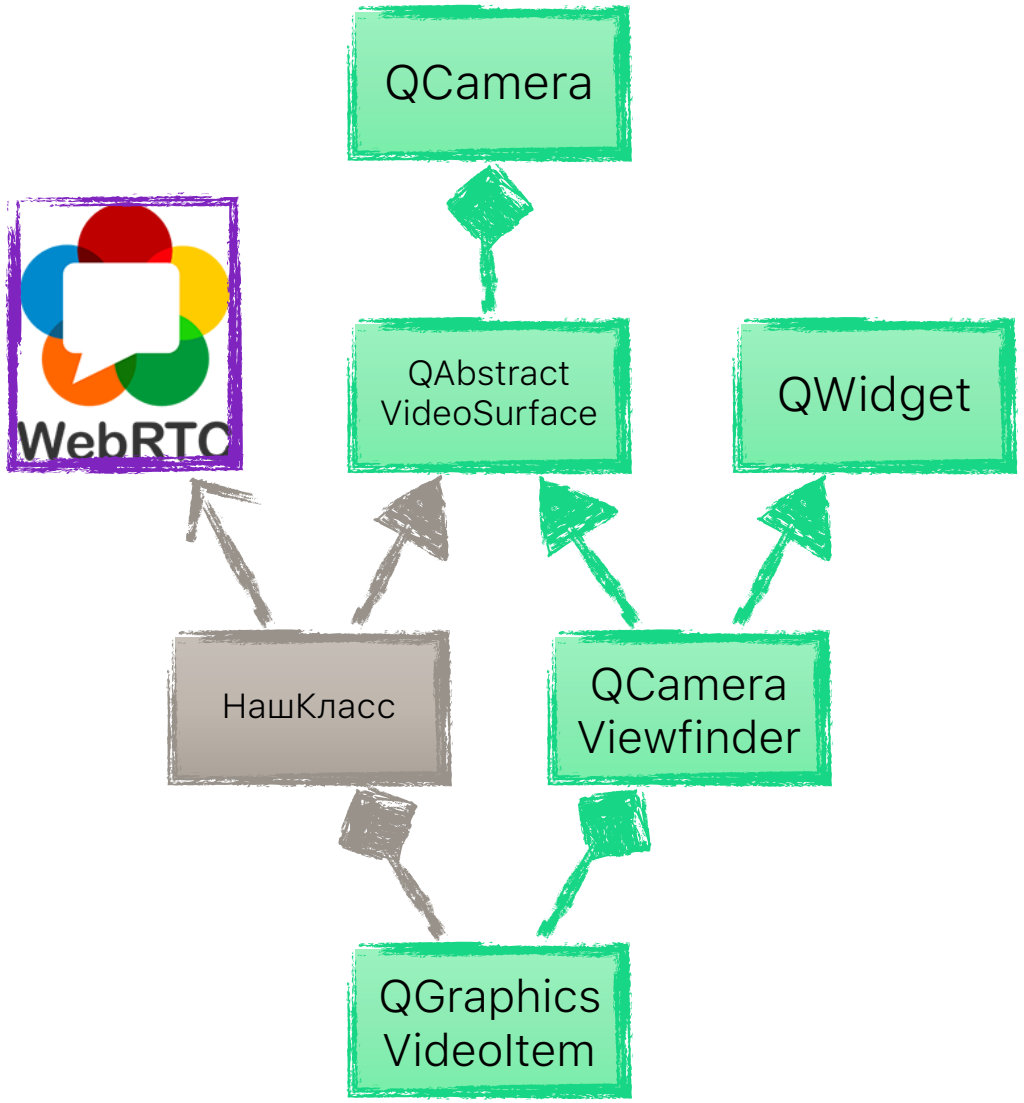
Захват камеры

Выводы после изучения примеров и первоначальной реализации:

- QCamera и QCameraViewfinder поддерживают все распространенные модели камер
- Если наследоваться от QAbstractVideoSurface, то можно получать кадры в виде QVideoFrame
- Но метаинформацию очень трудно обобщить на все кейсы («звонок другу из Австралии»)

Воспользуемся внутренностями QCameraViewfinder:

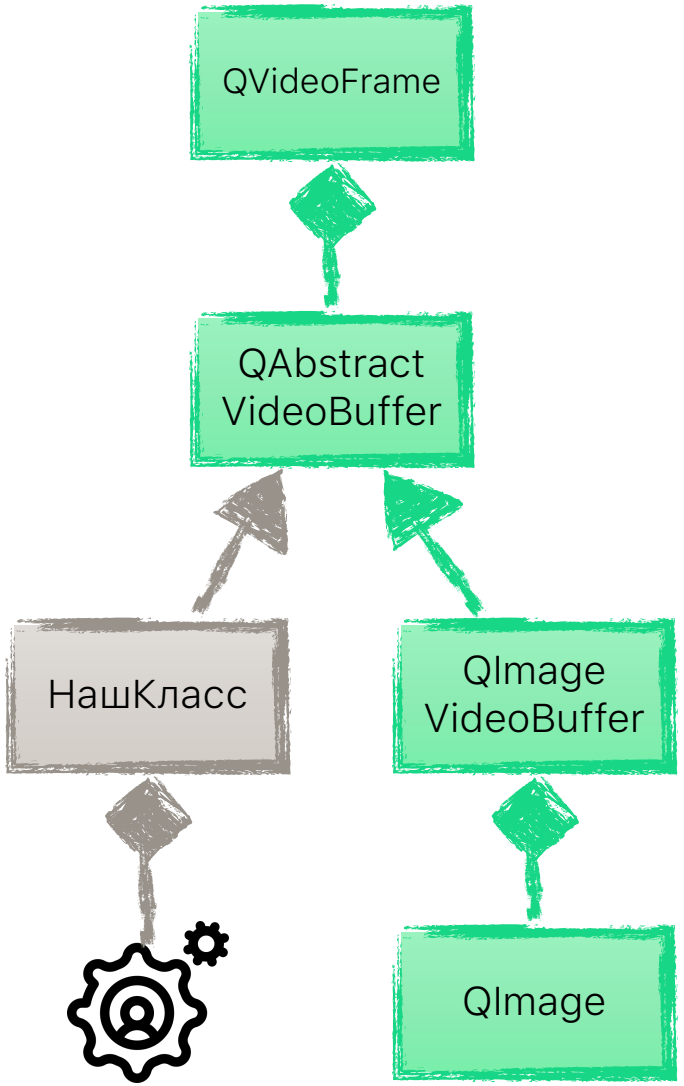
- QVideoFrame придется «рисовать» через QGraphicsVideoItem::paint()
- Зато таким образом его можно сразу сконвертировать из YUV в RGB



Захват экрана

Захват экрана

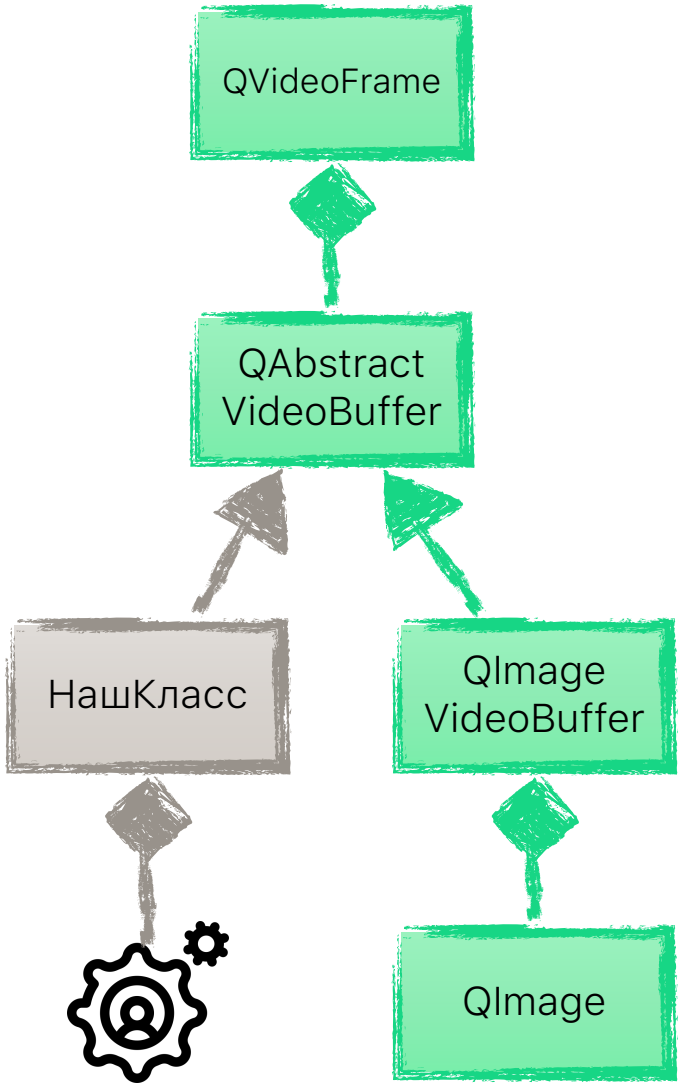
Почему Qt Multimedia? Поддержки из коробки нет, но QVideoFrame оказался удобной абстракцией



Захват экрана

Почему Qt Multimedia? Поддержки из коробки нет, но QVideoFrame оказался удобной абстракцией

Почему не средствами WebRTC? Архитектурно неудобно.

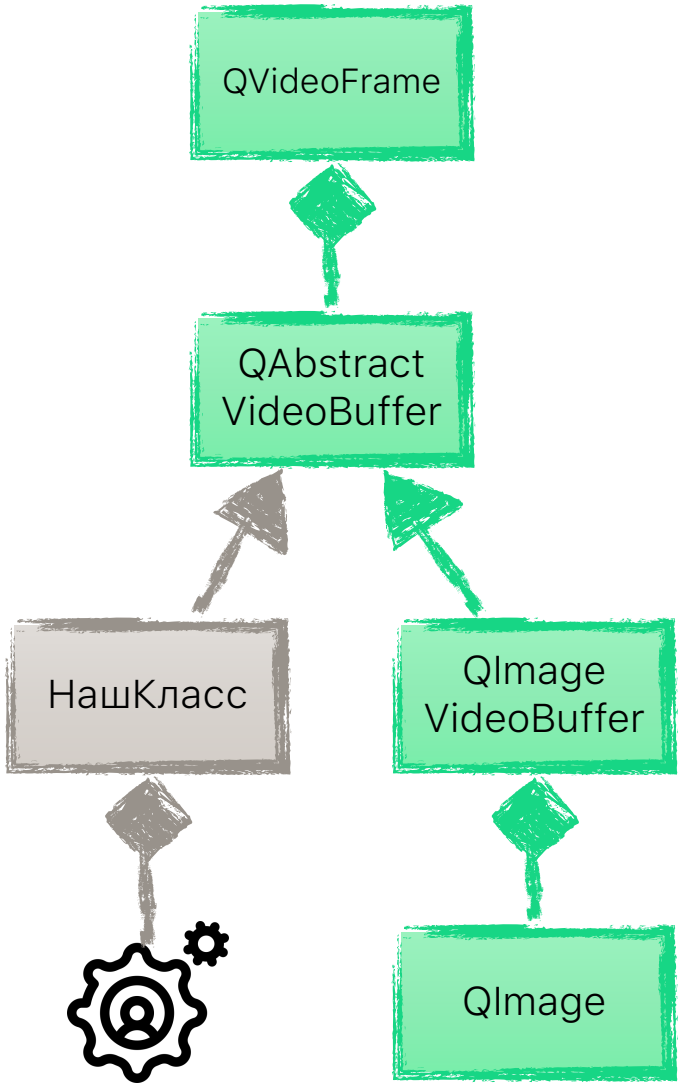


Захват экрана

Почему Qt Multimedia? Поддержки из коробки нет, но QVideoFrame оказался удобной абстракцией

Почему не средствами WebRTC? Архитектурно неудобно.

Почему не [screen_capture_lite]? Было поздно, ну и, возможно, мы добились большей эффективности.



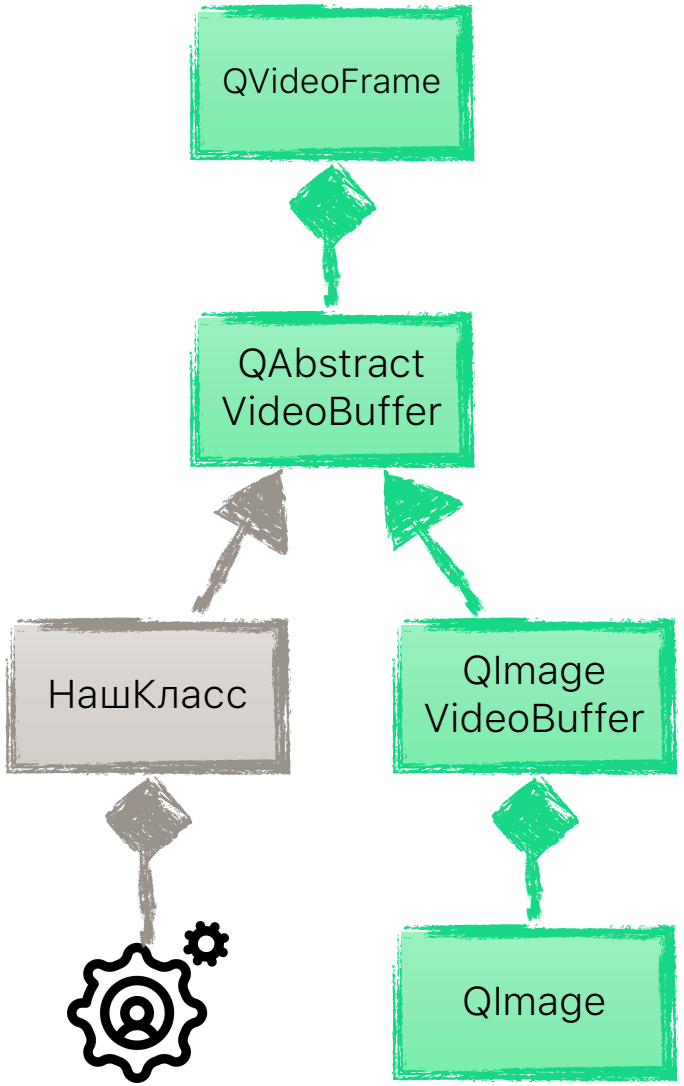
Захват экрана

Почему Qt Multimedia? Поддержки из коробки нет, но QVideoFrame оказался удобной абстракцией

Почему не средствами WebRTC? Архитектурно неудобно.

Почему не screen_capture_lite? Было поздно, ну и, возможно, мы добились большей эффективности.

Реализовали всё руками через API операционных систем.



Захват экрана

Почему Qt Multimedia? Поддержки из коробки нет, но QVideoFrame оказался удобной абстракцией

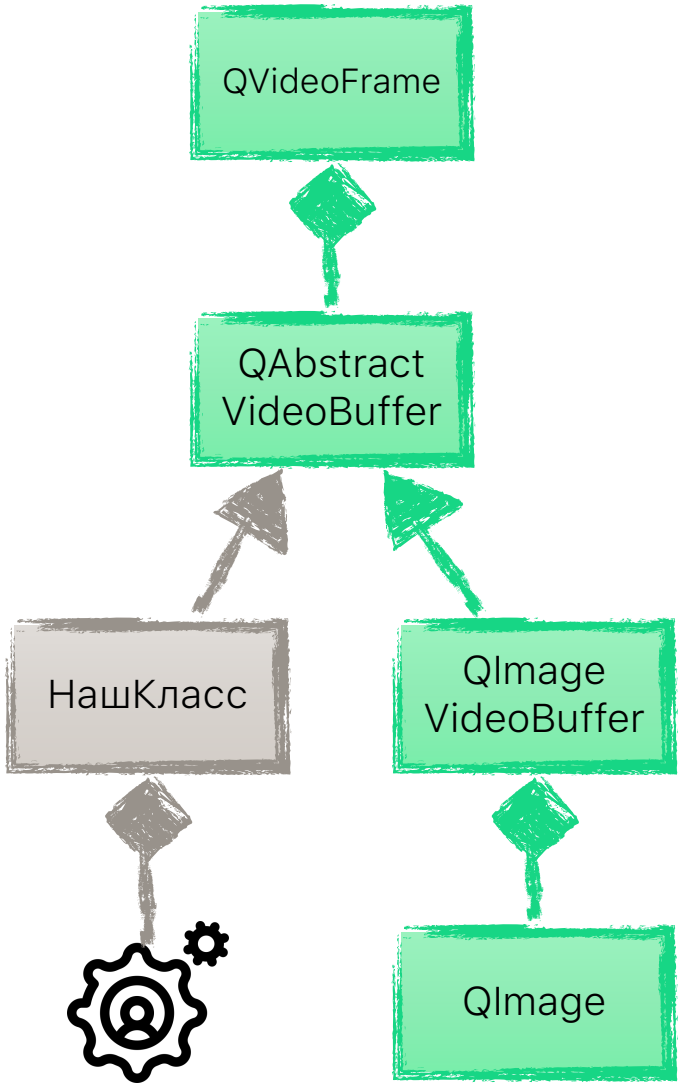
Почему не средствами WebRTC? Архитектурно неудобно.

Почему не screen_capture_lite? Было поздно, ну и, возможно, мы добились большей эффективности.

Реализовали всё руками через API операционных систем.

Выводы по использованию QAbstractVideoBuffer:

- Идея в том, чтобы отобразить произвольные данные на адресное пространство процесса и читать из него (аналогично mmap(2) для файлов)
- Хорошо работает для:
  - CGImages в macOS через CFDataGetBytePtr()
  - PixelBufferObject в OpenGL через glMapBuffer()



Подавление шума

Подавление шума — подробнее

Нейросети против пылесоса, или Как мы убрали лишний шум в звонках ВКонтакте / Александр Тоболь

https://habr.com/ru/company/vk/blog/572950/

Управление зависимостями

Управление зависимостями

CMake — стандарт де-факто для управления сборкой проекта на C++

Управление зависимостями

CMake — стандарт де-факто для управления сборкой проекта на C++

Conan — а это что-то новенькое! Пакетный менеджер зависимостей для C++

(*) это как npm для NodeJS или pip для Python

Управление зависимостями

CMake — стандарт де-факто для управления сборкой проекта на C++

Conan — а это что-то новенькое! Пакетный менеджер зависимостей для C++

(*) это как npm для NodeJS или pip для Python

В частности, сделали собственный рецепт для сборки WebRTC

conan — подробнее

Пакетный менеджер зависимостей conan — революция в подключении библиотек на C++. Часть 1 / Александр Шарамет

https://vk.com/video-147415323_456239621

Автотесты

Автотесты — подробнее

Как мы автотестировали десктопное приложение
«Звонки ВКонтакте» / Михаил Шваркунов

https://vk.com/video-147415323_456239590

День 100. Minimum Viable Prototype — итоги

День 100. Minimum Viable Prototype — итоги

- Qt действительно подходит для многих задач

День 100. Minimum Viable Prototype — итоги

- Qt действительно подходит для многих задач
- Можем сами пользоваться своим клиентом на стендапах

День 200. Nice-to-have features

День 200. Nice-to-have features

1. Виртуальный фон

День 200. Nice-to-have features

1. Виртуальный фон
2. Улучшение черт лица

День 200. Nice-to-have features

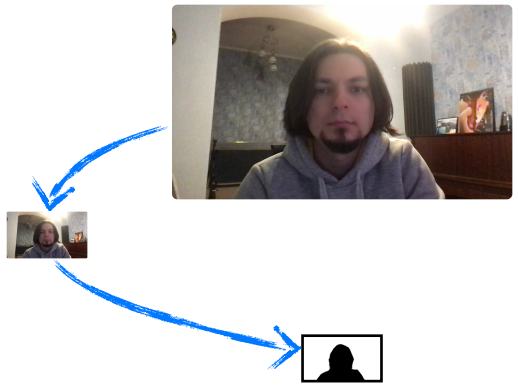
1. Виртуальный фон
2. Улучшение черт лица
3. Сбор крэш-дампов

День 200. Nice-to-have features

1. Виртуальный фон
2. Улучшение черт лица
3. Сбор крэш-дампов
4. Продуктовая логика

Виртуальный фон

Виртуальный фон



Сегментирование выполняется нейросетью

- инференс на Windows и Linux — ONNX Runtime, на macOS — в TensorFlow Lite.
- на входе — уменьшенный кадр
- на выходе — альфа-канал

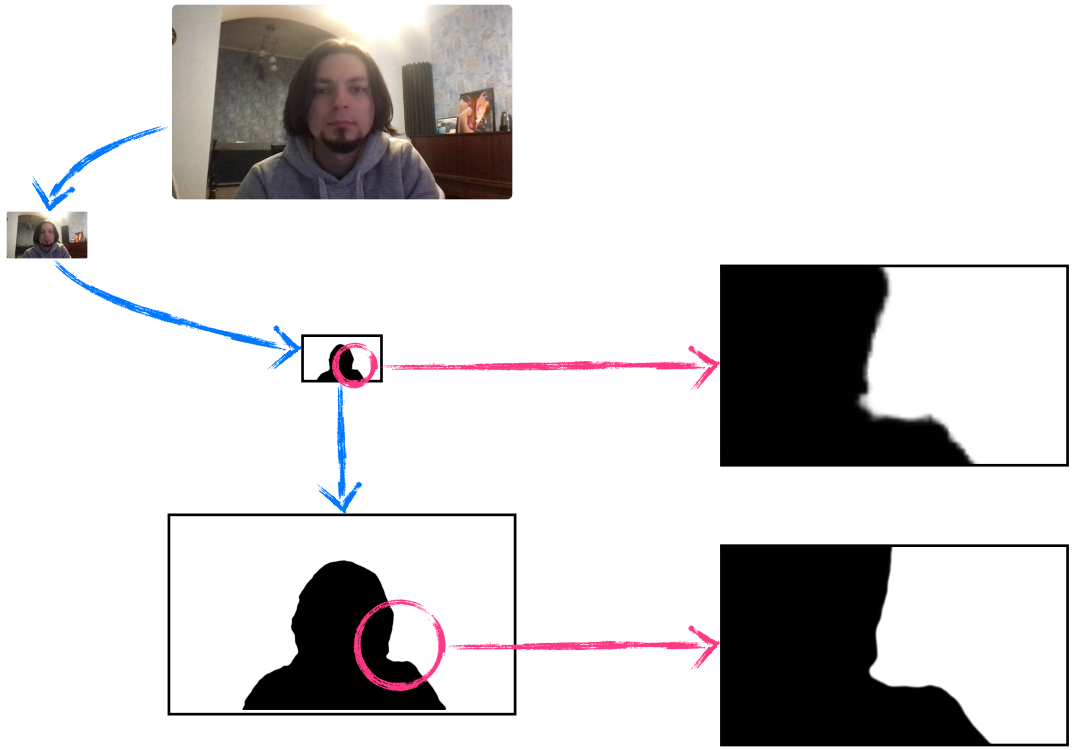
Виртуальный фон

Сегментирование выполняется нейросетью

- инференс на Windows и Linux — ONNX Runtime, на macOS — в TensorFlow Lite.
- на входе — уменьшенный кадр
- на выходе — альфа-канал

Много дорогих операций — нужен OpenGL:

- Сглаживание краёв — kawase-blur и сигмоида для альфа-канала



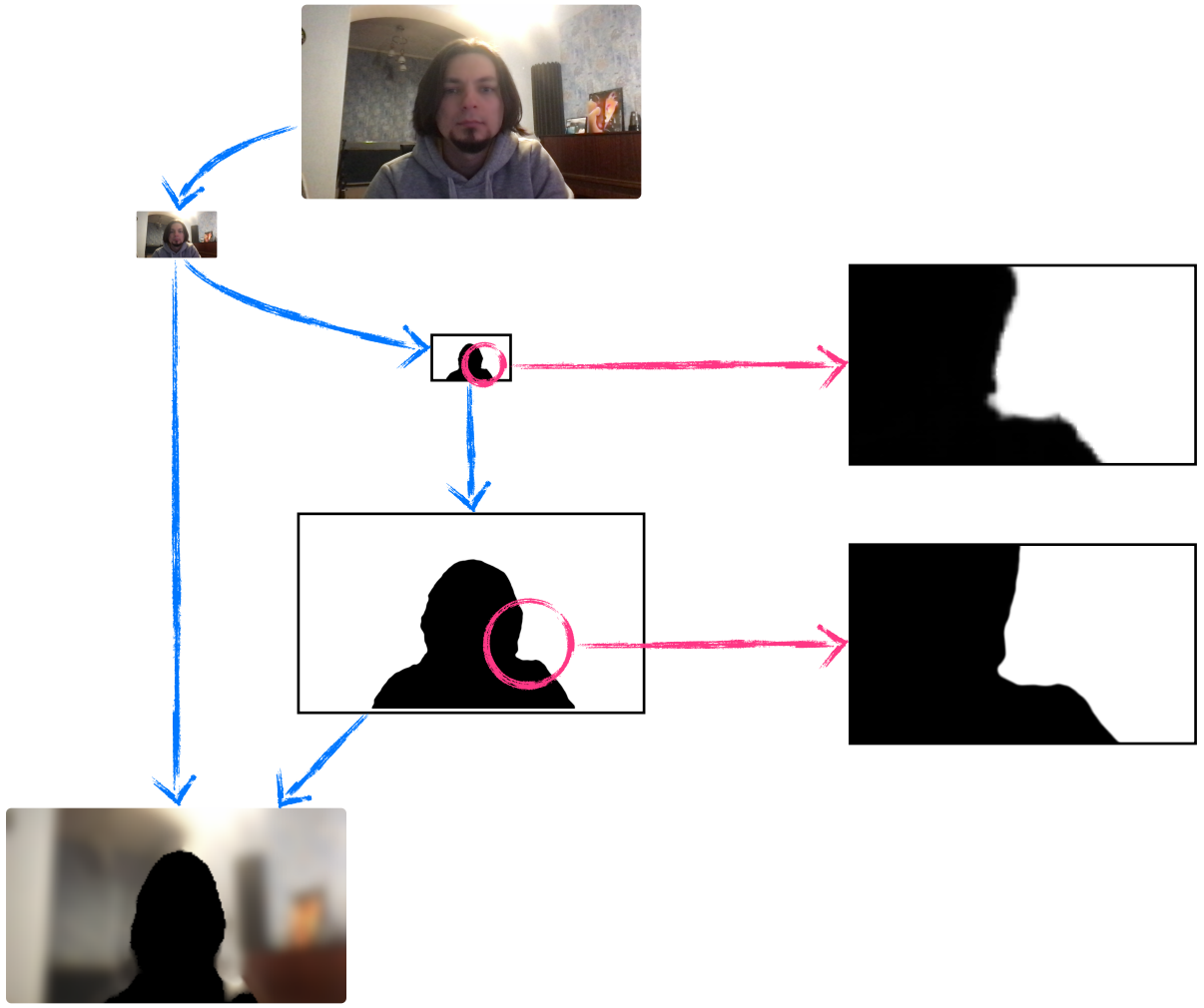
Виртуальный фон

Сегментирование выполняется нейросетью

- инференс на Windows и Linux — ONNX Runtime, на macOS — в TensorFlow Lite.
- на входе — уменьшенный кадр
- на выходе — альфа-канал

Много дорогих операций — нужен OpenGL:

- Сглаживание краёв — kawase-blur и сигмоида для альфа-канала
- Размытие фона — kawase-blur
  - для уменьшенного кадра — качество то же
  - с учётом альфа-канала — эффект гало



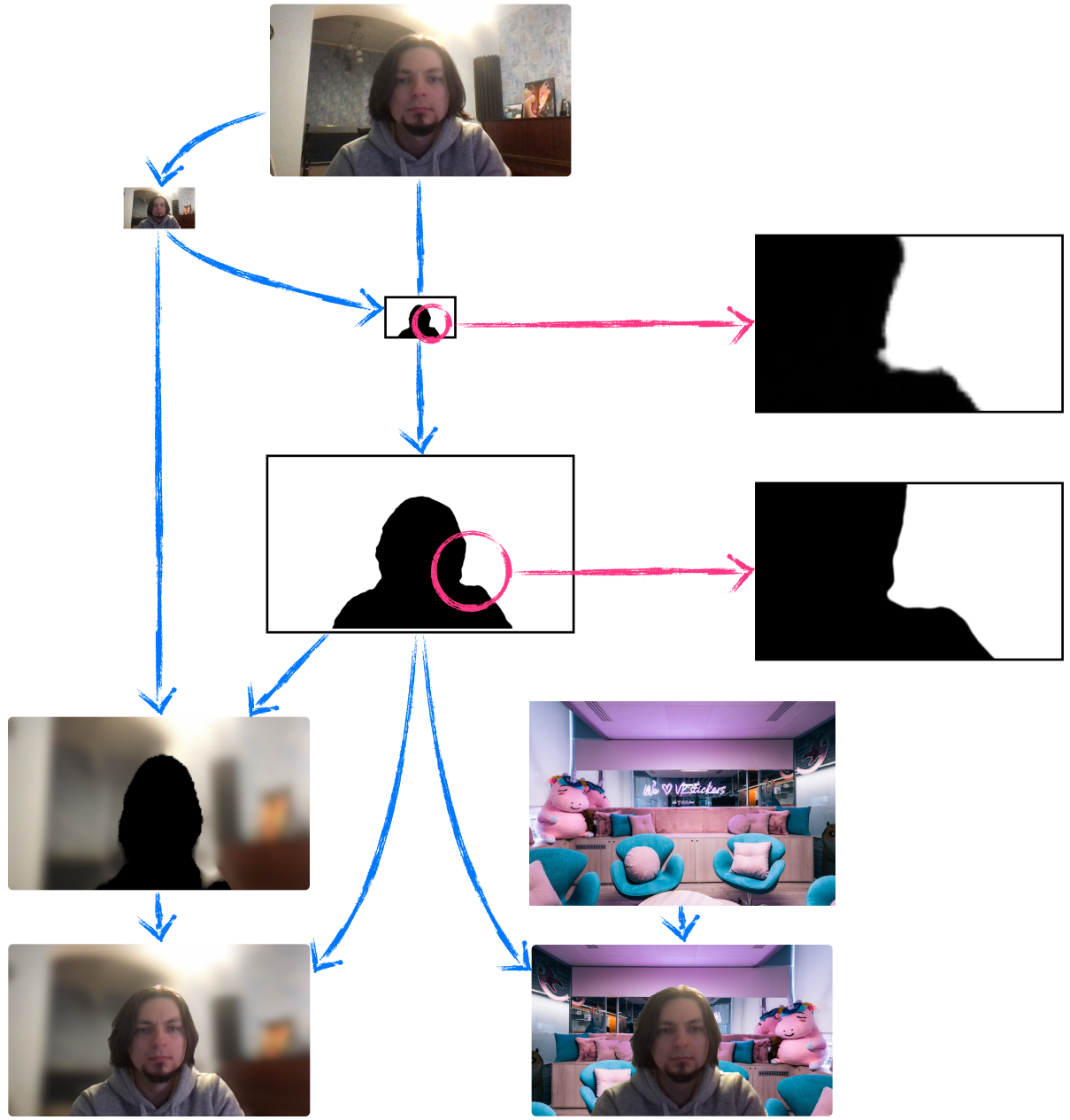
Виртуальный фон

Сегментирование выполняется нейросетью

- инференс на Windows и Linux — ONNX Runtime, на macOS — в TensorFlow Lite.
- на входе — уменьшенный кадр
- на выходе — альфа-канал

Много дорогих операций — нужен OpenGL:

- Сглаживание краёв — kawase-blur и сигмоида для альфа-канала
- Размытие фона — kawase-blur
  - для уменьшенного кадра — качество то же
  - с учётом альфа-канала — эффект гало
- Масштабирование и композирование



Улучшение черт лица

Сбор крэш-дампов

День 100. Nice-to-have feature — итоги

День 100. Nice-to-have feature — итоги

- Можно было бы запускать в продакшен

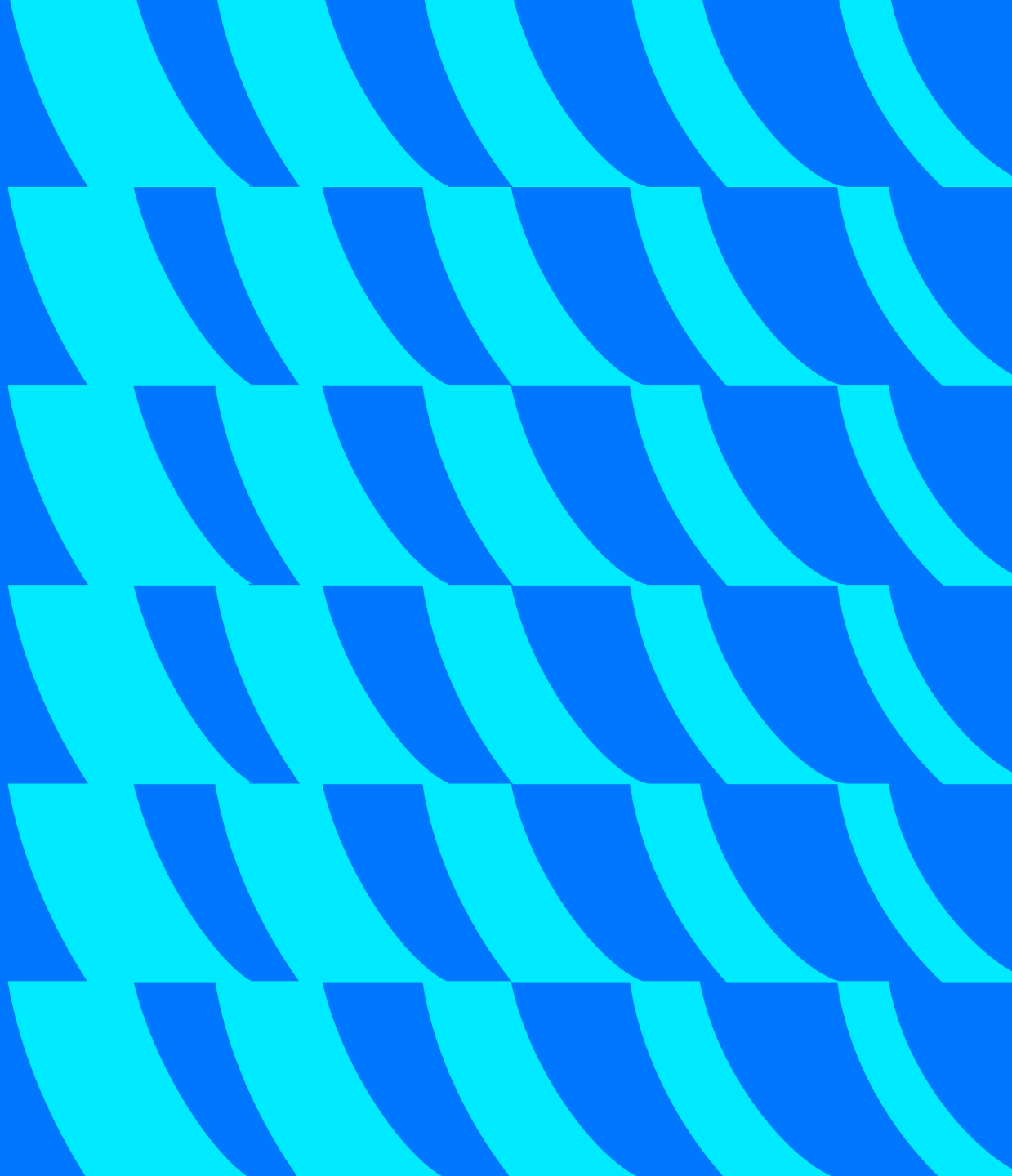
День 100. Nice-to-have feature — итоги

- Можно было бы запускать в продакшен
- Но предстоит еще протестировать в Действительно Больших Звонках

День 100. Nice-to-have feature — итоги

- Можно было бы запускать в продакшен
- Но предстоит еще протестировать в Действительно Больших Звонках
- Начали использовать OpenGL — если что, можно будет добавить ещё

День 300.
Bottlenecks.
Tips & tricks



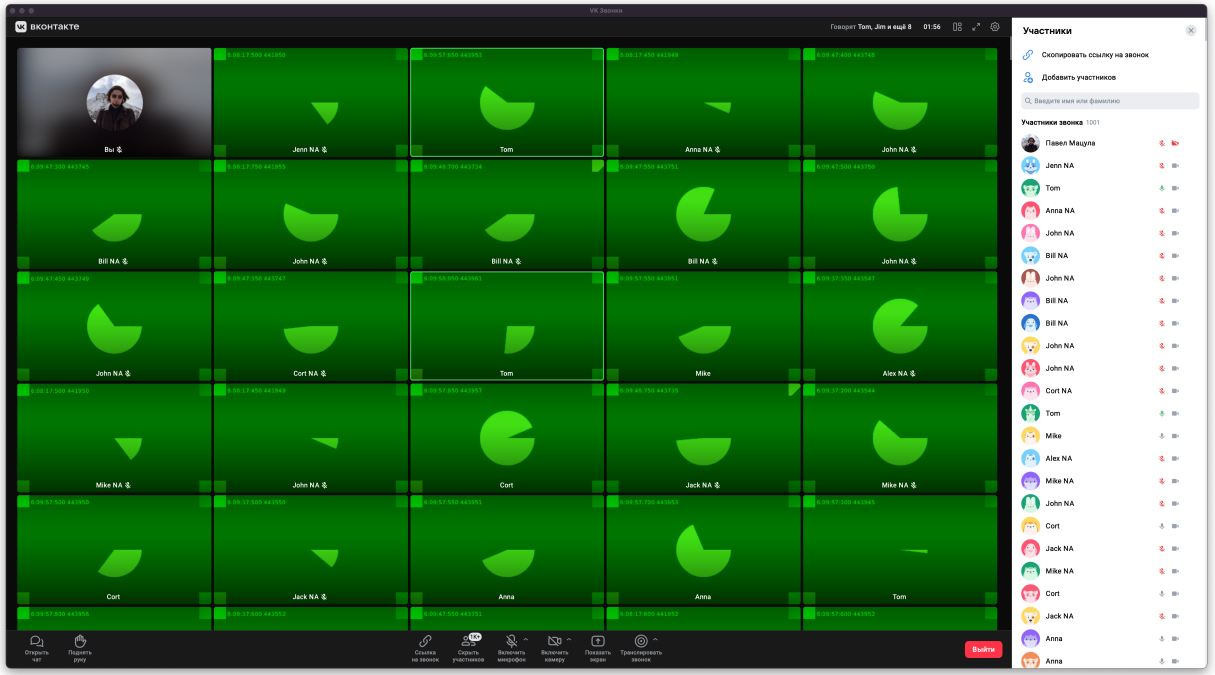
Привет, проклятие 10K!

Привет, проклятие 10K!

В нашем случае — 10K участников звонка.

Самые узкие места:

- Графические зоны с плитками видео, выложенные в «сетку» или «карусель»
- Список участников в виде виджетов с иконками и кнопками



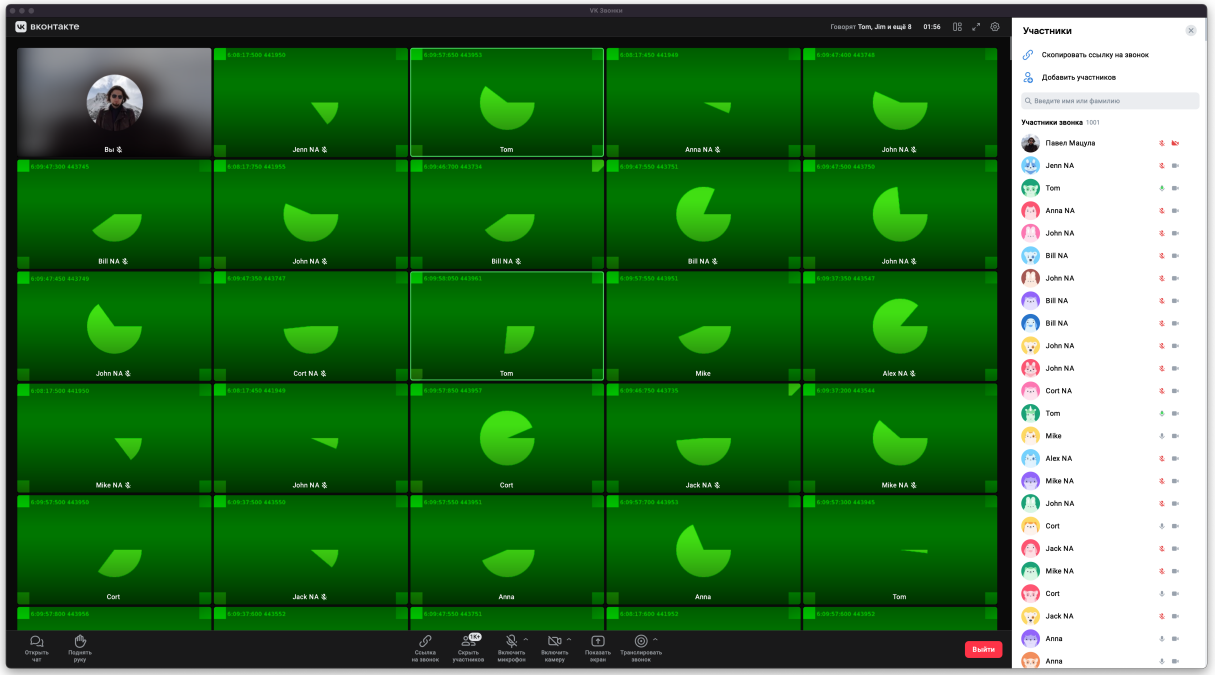
Привет, проклятие 10K!

В нашем случае — 10K участников звонка.

Самые узкие места:

- Графические зоны с плитками видео, выложенные в «сетку» или «карусель»
- Список участников в виде виджетов с иконками и кнопками

Что делать? Думать! И профилировать.



Привет, проклятие 10K!

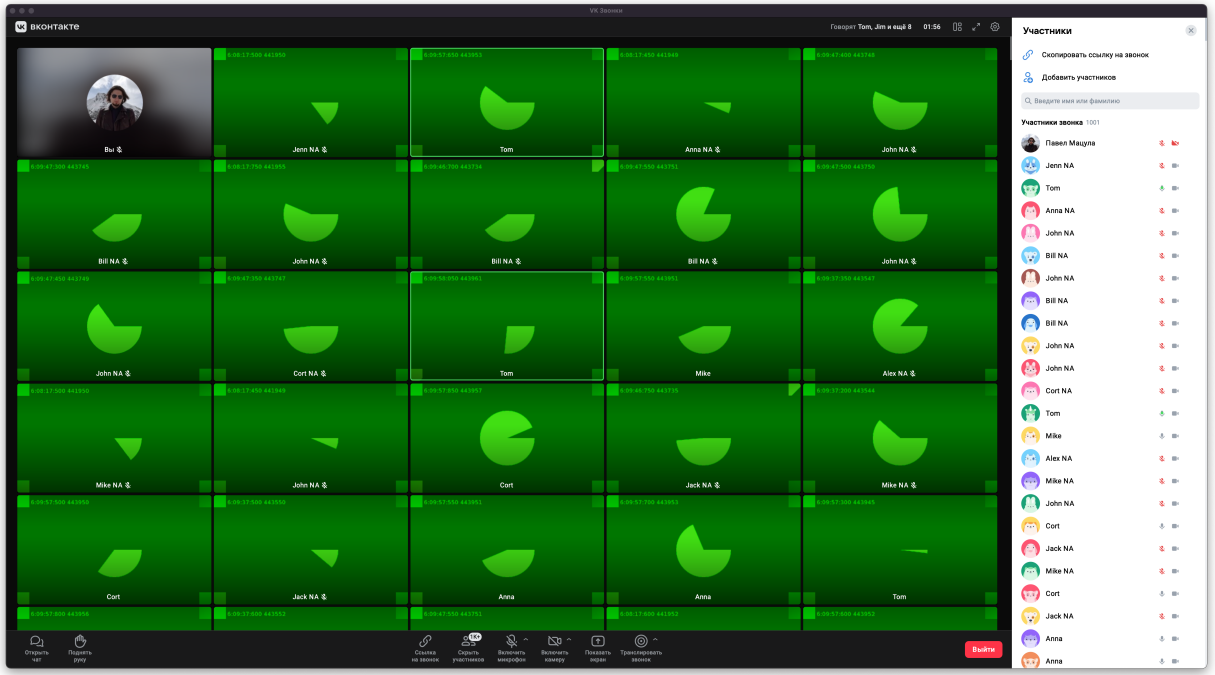
В нашем случае — 10K участников звонка.

Самые узкие места:

- Графические зоны с плитками видео, выложенные в «сетку» или «карусель»
- Список участников в виде виджетов с иконками и кнопками

Что делать? Думать! И профилировать.

Почему просто не игнорировать лишнее?



Привет, проклятие 10K!

В нашем случае — 10K участников звонка.

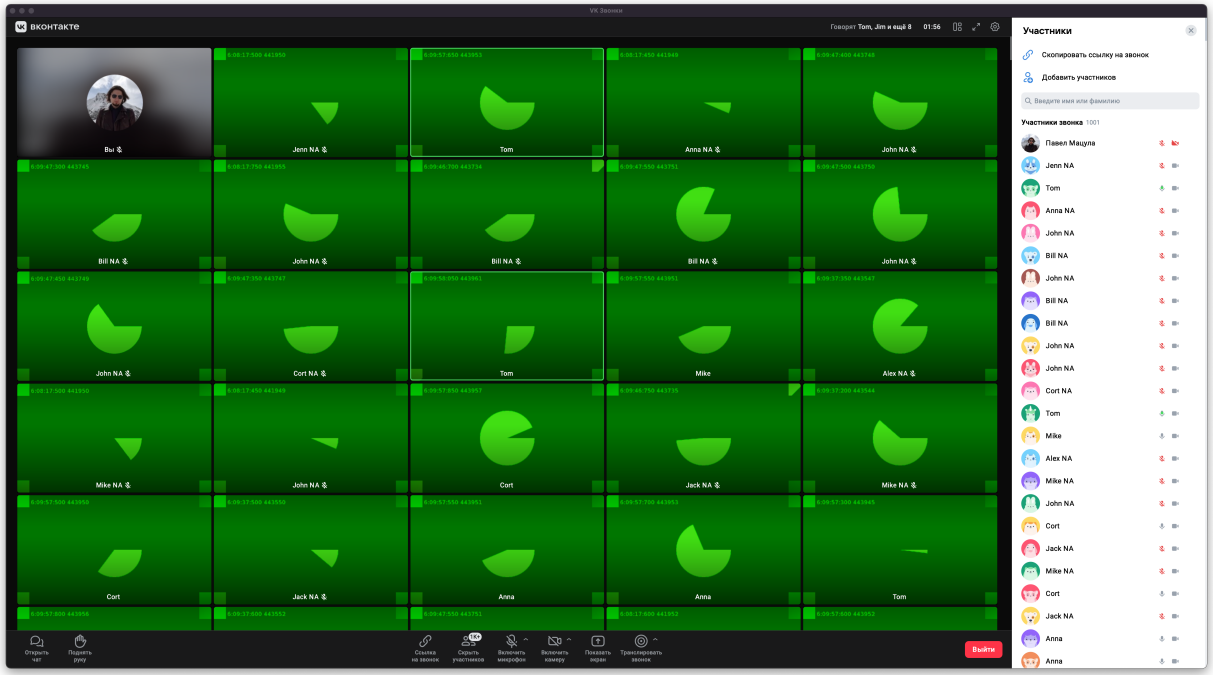
Самые узкие места:

- Графические зоны с плитками видео, выложенные в «сетку» или «карусель»
- Список участников в виде виджетов с иконками и кнопками

Что делать? Думать! И профилировать.

Почему просто не игнорировать лишнее?

- Уже так делаем для стримов и запросов
- Ухудшает UX — мы хотим плавный скролл!
- Усложняет код — нет готовых виджетов
- Можем попробовать обойтись без этого

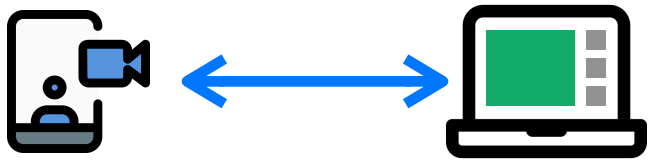


Архитектура бэкенда

Топология звонка

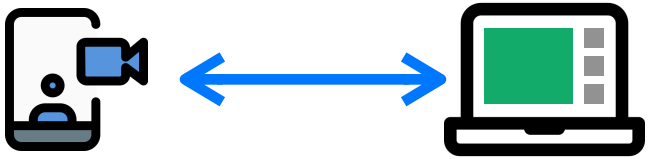
Топология звонка

1 на 1
peer-to-peer

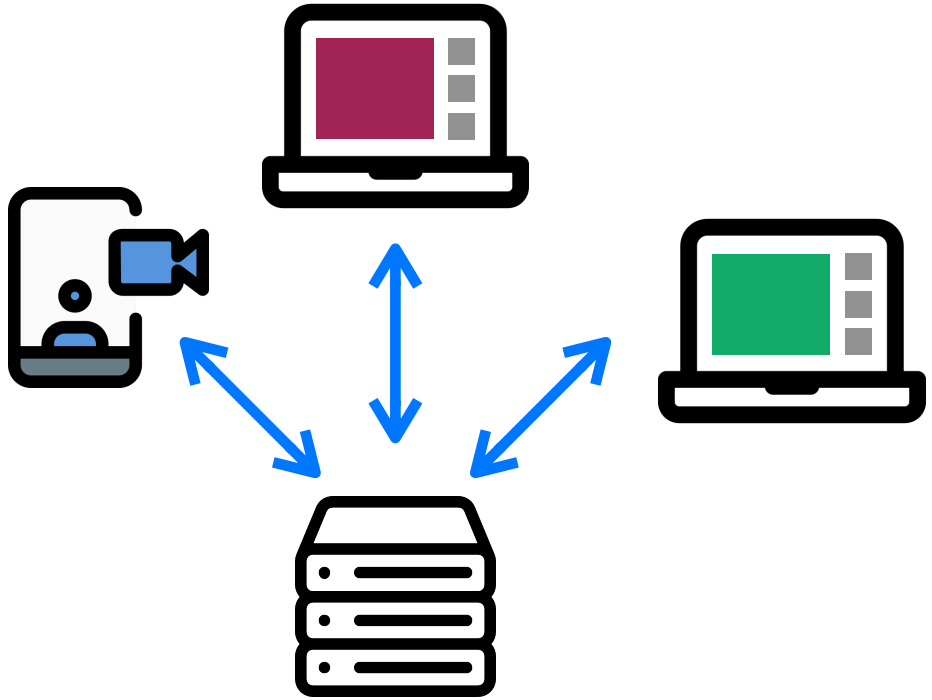


Топология звонка

1 на 1
peer-to-peer

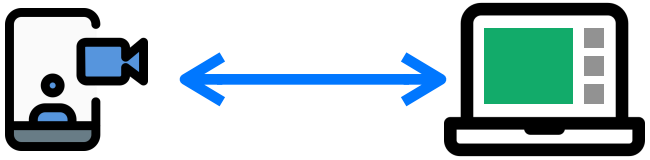


3+
server

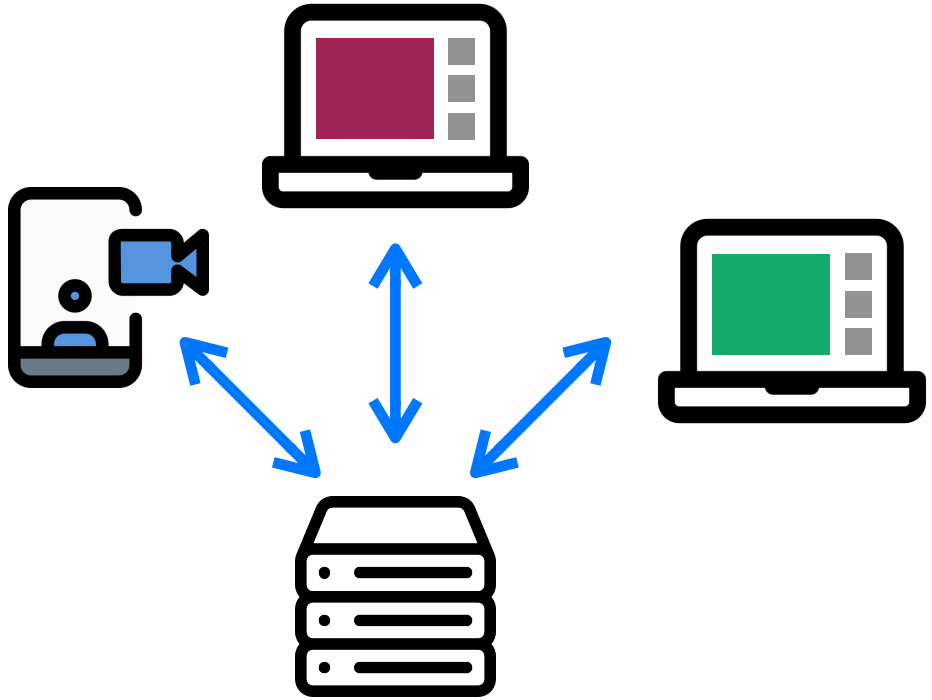


Топология звонка

1 на 1
peer-to-peer



3+
server



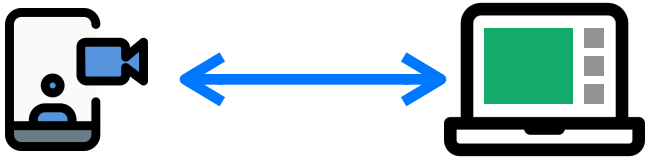
Аудио: MCU

+Просто для клиента

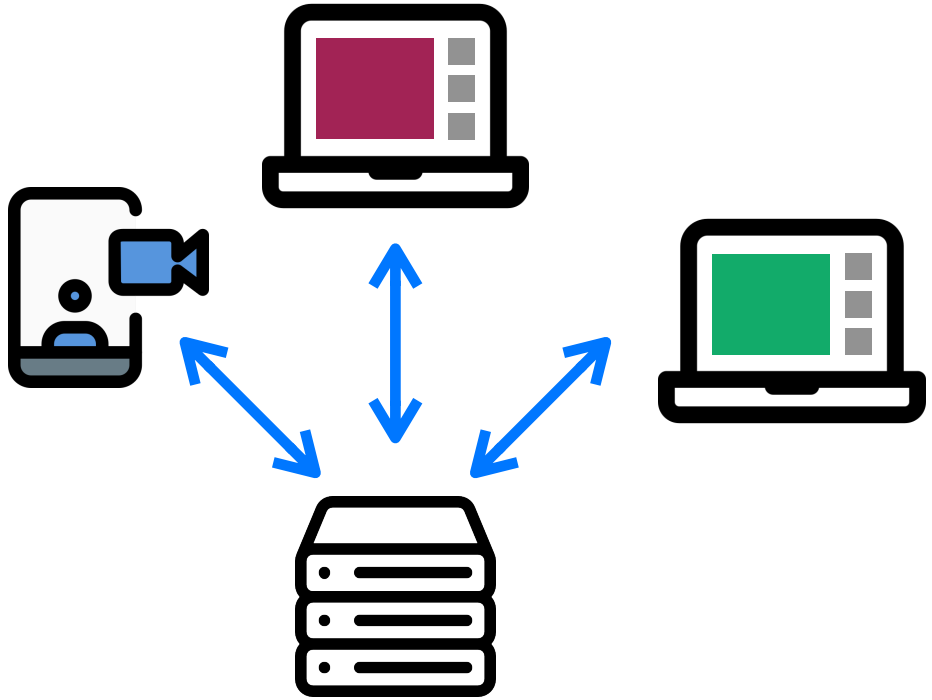
-Нагрузка на сервер

Топология звонка

1 на 1
peer-to-peer



3+
server



Аудио: MCU

+Просто для клиента

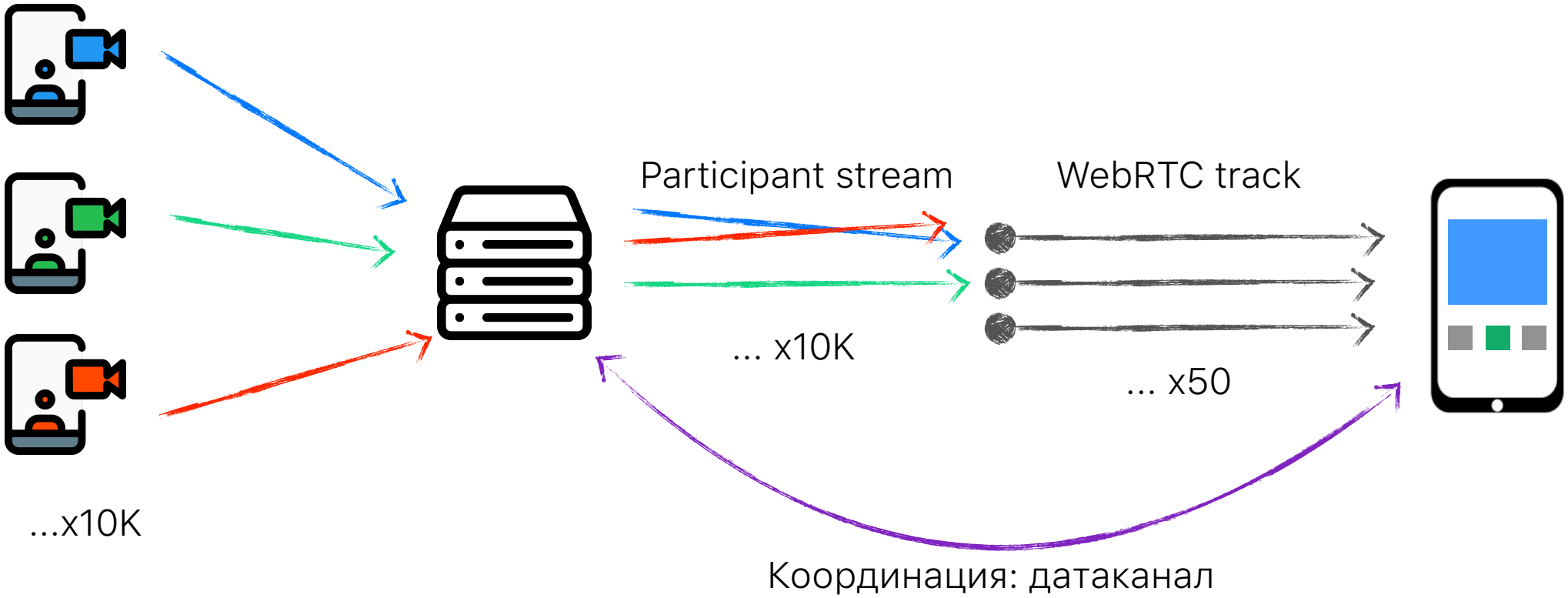
-Нагрузка на сервер

Видео: SFU

+Гибкий UI на клиенте

-Тяжело клиенту

Слоты для видео



Архитектура бэкенда — подробнее

VK Звонки: платформа видеоконференций без ограничений на количество участников звонка / Иван Григорьев

https://live.jugru.org/video?v=MTAwMTE3iiM3NjQ2ijA

Графическая сцена

QGraphicsScene управляет расположением объектов на плоскости.

Графическая сцена

QGraphicsScene управляет расположением объектов на плоскости.

QGraphicsView управляет «камерой», в которую попадает часть объектов.

Быстрый поиск таких объектов — с помощью Binary Space Partition Tree.

Графическая сцена

QGraphicsScene управляет расположением объектов на плоскости.

QGraphicsView управляет «камерой», в которую попадает часть объектов.

Быстрый поиск таких объектов — с помощью Binary Space Partition Tree.

Для отрисовки можно вместо QWidget подставить QOpenGLWidget.

Есть и другие Qt-обертки над OpenGL, которые заменяют glew.

Графическая сцена

QGraphicsScene управляет расположением объектов на плоскости.

QGraphicsView управляет «камерой», в которую попадает часть объектов.

Быстрый поиск таких объектов — с помощью Binary Space Partition Tree.

Для отрисовки можно вместо QWidget подставить QOpenGLWidget.

Есть и другие Qt-обертки над OpenGL, которые заменяют glew.

По сути — те же технологии, что в геймдеве.

Пайплайн работы с видео

Самые узкие места при работе с видео:

- Масштабирование
  - Решаем с помощью OpenGL-текстурирования

Пайплайн работы с видео

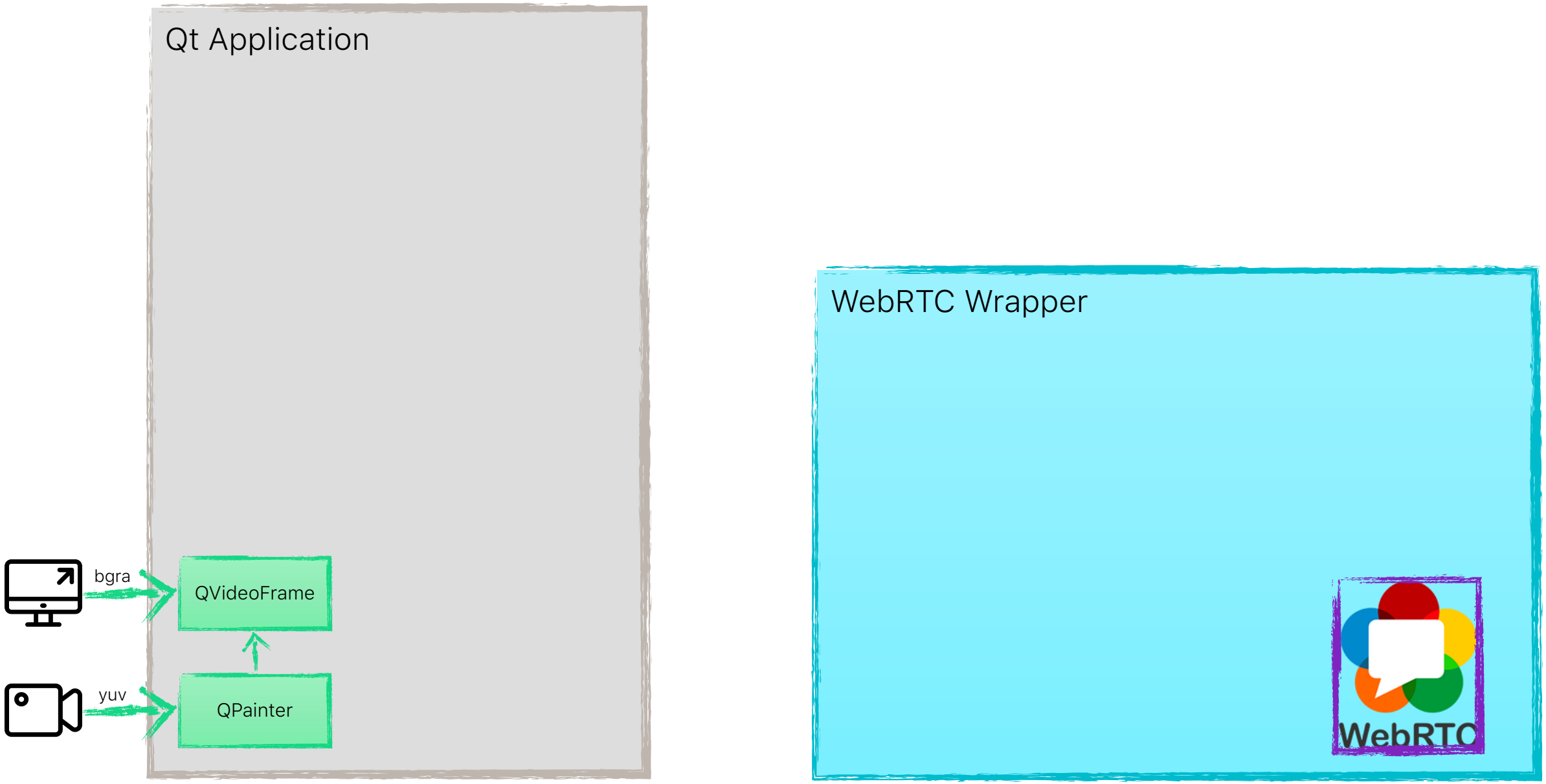
Самые узкие места при работе с видео:

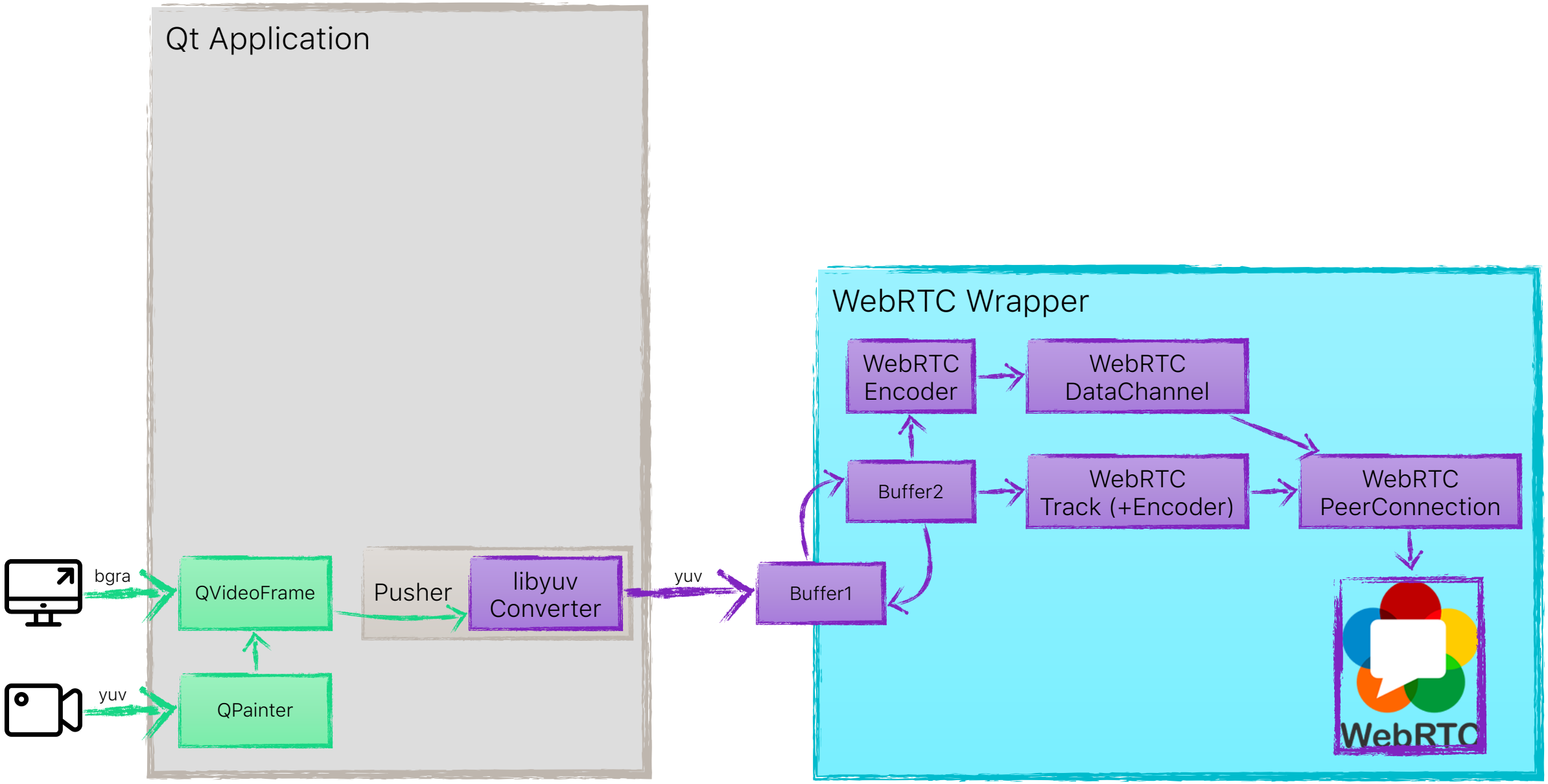
- Масштабирование
  - Решаем с помощью OpenGL-текстурирования
- Конвертация формата кадра
  - libyuv хорош, но не подошёл для камеры

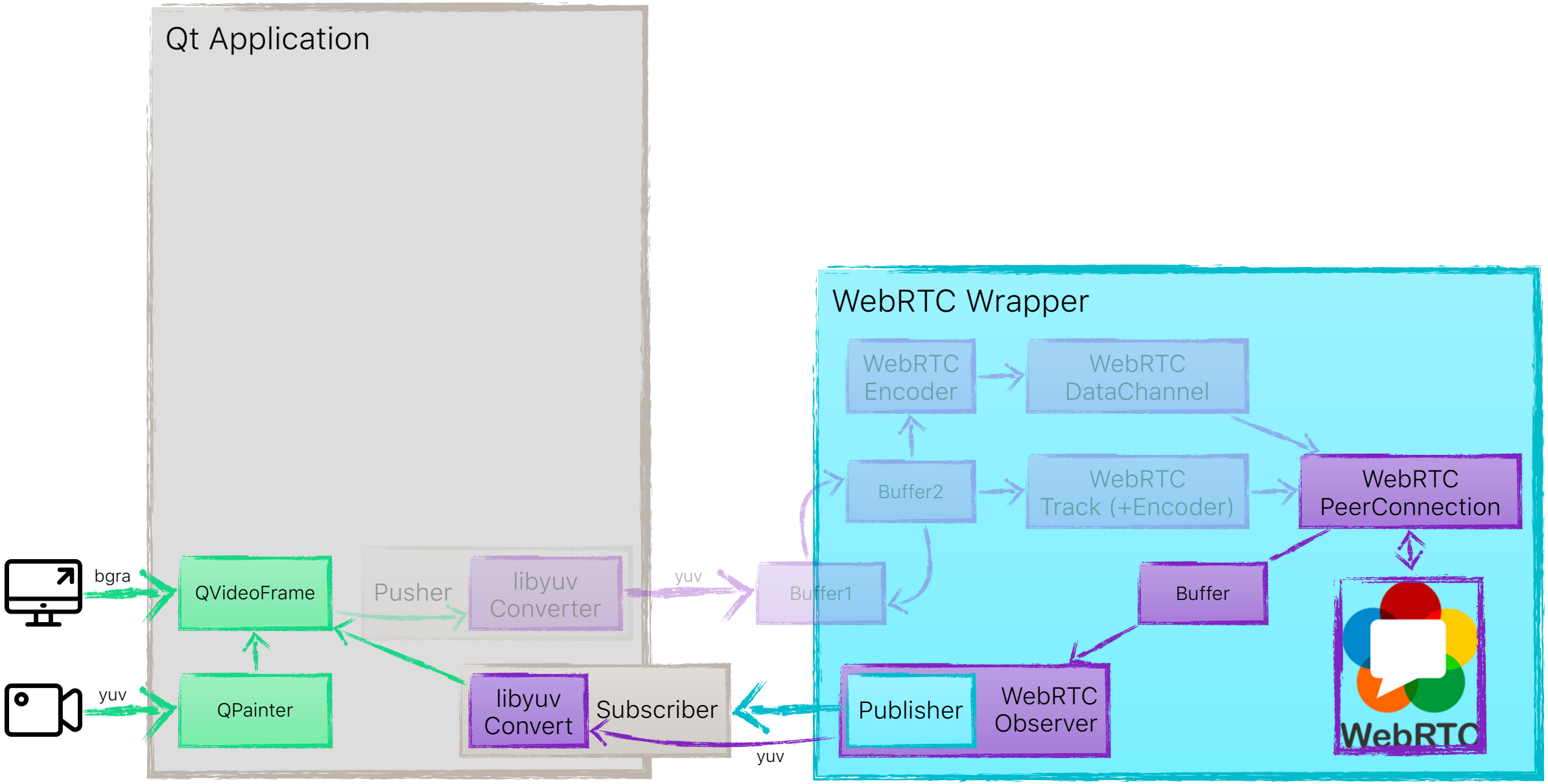
Пайплайн работы с видео

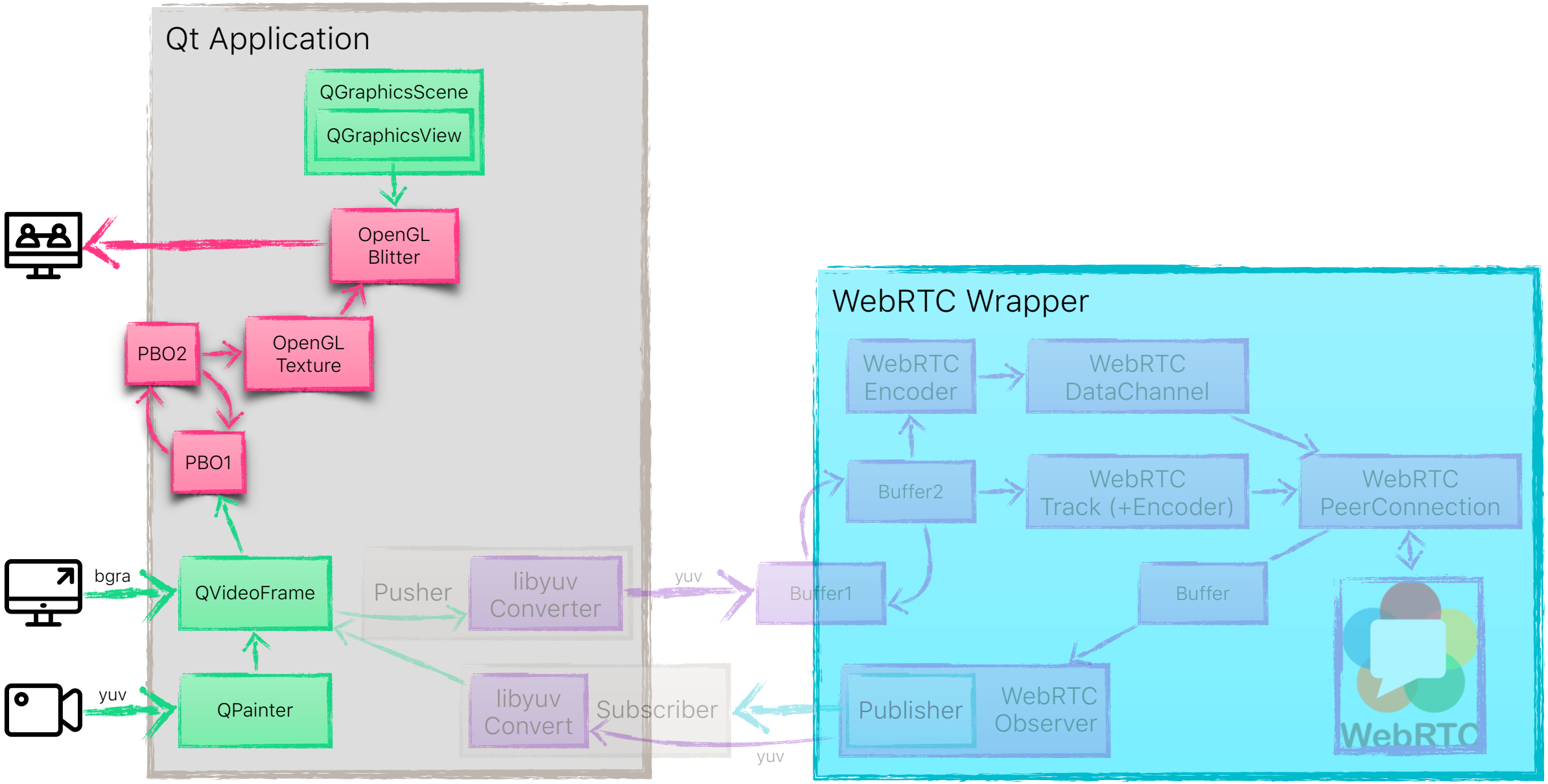
Самые узкие места при работе с видео:

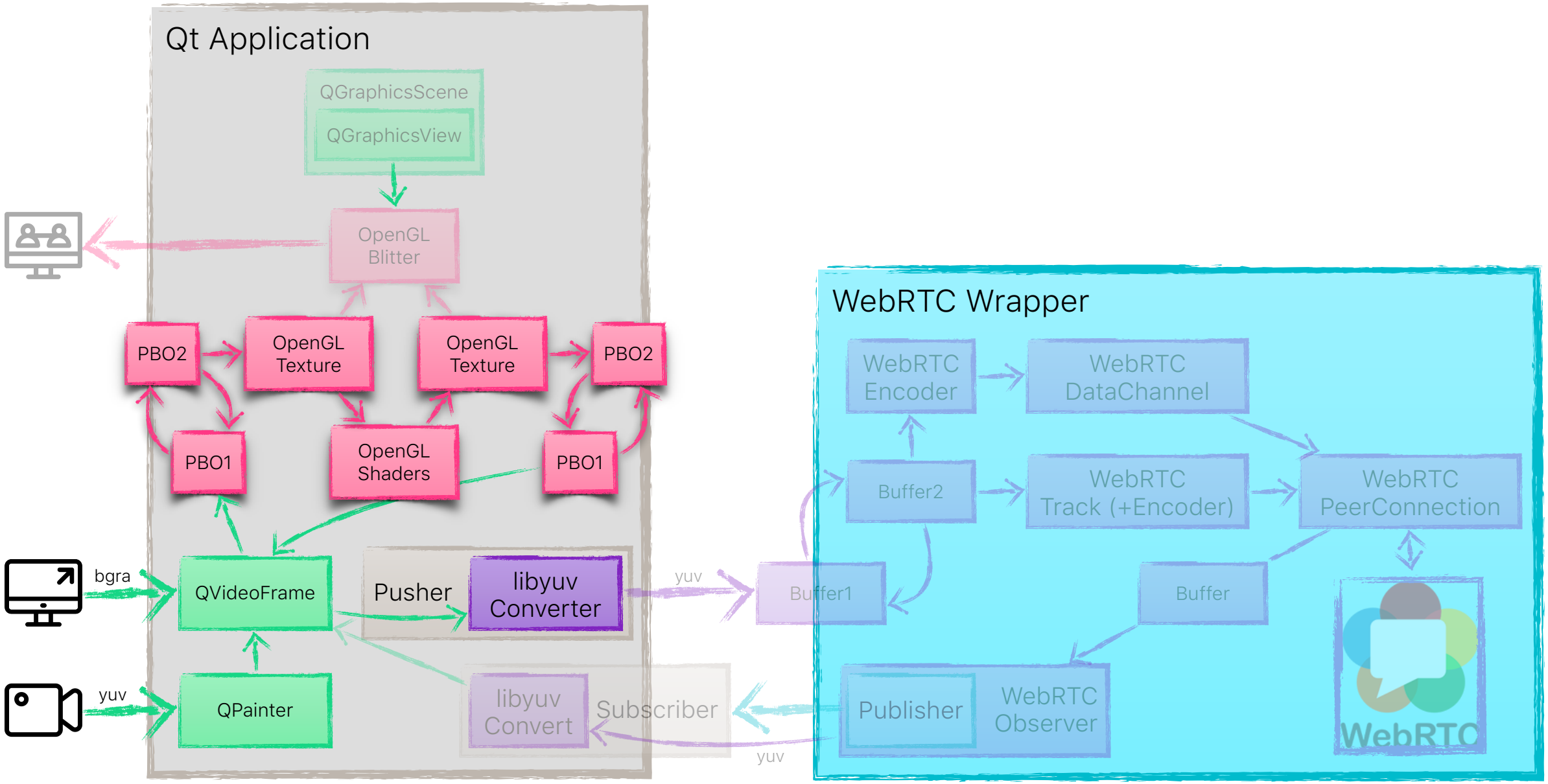
- Масштабирование
  - Решаем с помощью OpenGL-текстурирования
- Конвертация формата кадра
  - libyuv хорош, но не подошёл для камеры
- Тупо копирование
  - Используем общие буферы и двойную буферизацию
  - Не копируем, а сразу конвертируем


Qt Application
bgra
QVideoFrame
yuv
QPainter
WebRTC Wrapper
WebRTC


Qt Application
WebRTC Wrapper
WebRTC Encoder
WebRTC DataChannel
Buffer2
WebRTC Track (+Encoder)
WebRTC PeerConnection
bgra
QVideoFrame
Pusher
libyuv Converter
yuv
Buffer1
yuv
QPainter
WebRTC


Qt Application
WebRTC Wrapper
WebRTC Encoder
WebRTC DataChannel
Buffer2
WebRTC Track (+Encoder)
WebRTC PeerConnection
bgra
QVideoFrame
Pusher
libyuv Converter
yuv
Buffer1
Buffer
yuv
QPainter
libyuv Convert
Subscriber
Publisher
WebRTC Observer
yuv
WebRTC


Qt Application
QGraphicsScene
QGraphicsView
OpenGL Blitter
PBO2
OpenGL Texture
PBO1
bgra
QVideoFrame
Pusher
libyuv Converter
yuv
yuv
QPainter
libyuv Convert
Subscriber
yuv
WebRTC Wrapper
WebRTC Encoder
WebRTC DataChannel
Buffer2
WebRTC Track (+Encoder)
WebRTC PeerConnection
Buffer1
Buffer
Publisher
WebRTC Observer
WebRTC


Qt Application
QGraphicsScene
QGraphicsView
OpenGL Blitter
PBO2
OpenGL Texture
OpenGL Texture
PBO2
PBO1
OpenGL Shaders
PBO1
bgra
QVideoFrame
Pusher
libyuv Converter
yuv
yuv
QPainter
libyuv Convert
Subscriber
yuv
WebRTC Wrapper
WebRTC Encoder
WebRTC DataChannel
Buffer2
WebRTC Track (+Encoder)
WebRTC PeerConnection
Buffer1
Buffer
Publisher
WebRTC Observer
WebRTC

Пайплайн работы с видео

Планы на будущее:

- Полностью перейти на YUV

Qt tips & tricks

Как мы ускоряли список участников:

- Main-тред нужно разгружать до абсолютного минимума. Можно оставить:
  - Отрисовку графики
  - Обработку событий

Qt tips & tricks

Как мы ускоряли список участников:

- Main-тред нужно разгружать до абсолютного минимума. Можно оставить:
  - Отрисовку графики
  - Обработку событий
- Количество вызовов qMetaCall() нужно сводить к минимуму.
  - Нельзя отправлять много однотипных сигналов
  - Можно и нужно собирать их в единый батч

Qt tips & tricks

Как мы ускоряли список участников:

- Main-тред нужно разгружать до абсолютного минимума. Можно оставить:
  - Отрисовку графики
  - Обработку событий
- Количество вызовов qMetaCall() нужно сводить к минимуму.
  - Нельзя отправлять много однотипных сигналов
  - Можно и нужно собирать их в единый батч
- Надо отказаться от использования findChildren() в пользу ручного перебора

Qt tips & tricks

Как мы ускоряли список участников:

- Main-тред нужно разгружать до абсолютного минимума. Можно оставить:
  - Отрисовку графики
  - Обработку событий
- Количество вызовов qMetaCall() нужно сводить к минимуму.
  - Нельзя отправлять много однотипных сигналов
  - Можно и нужно собирать их в единый батч
- Надо отказаться от использования findChildren() в пользу ручного перебора
- Надо искать любые возможные способы отказаться от использования setVisible()

Qt tips & tricks

Как мы ускоряли список участников:

- Main-тред нужно разгружать до абсолютного минимума. Можно оставить:
  - Отрисовку графики
  - Обработку событий
- Количество вызовов qMetaCall() нужно сводить к минимуму.
  - Нельзя отправлять много однотипных сигналов
  - Можно и нужно собирать их в единый батч
- Надо отказаться от использования findChildren() в пользу ручного перебора
- Надо искать любые возможные способы отказаться от использования setVisible()
- Часть функционала надо явным образом отключать
  - Переменная окружения QT_NO_SUBTRACTOPAQUESIBLINGS

День 365.
Итоги

Итоги

Итоги

Мы выпустили продукт, которым очень гордимся!

Итоги

- Мы выпустили продукт, которым очень гордимся!
- Достигли всех поставленных продуктовых целей

Итоги

- Мы выпустили продукт, которым очень гордимся!
- Достигли всех поставленных продуктовых целей
- Используем современные, интересные и сложные технологии

Итоги

- Мы выпустили продукт, которым очень гордимся!
- Достигли всех поставленных продуктовых целей
- Используем современные, интересные и сложные технологии
- Нашли много красивых решений для организации кода