СВД
Встраиваемые Системы

Синаптика
Интеллект реального времени

Практическое применение платформы Синаптика на примере задачи распознавания образов

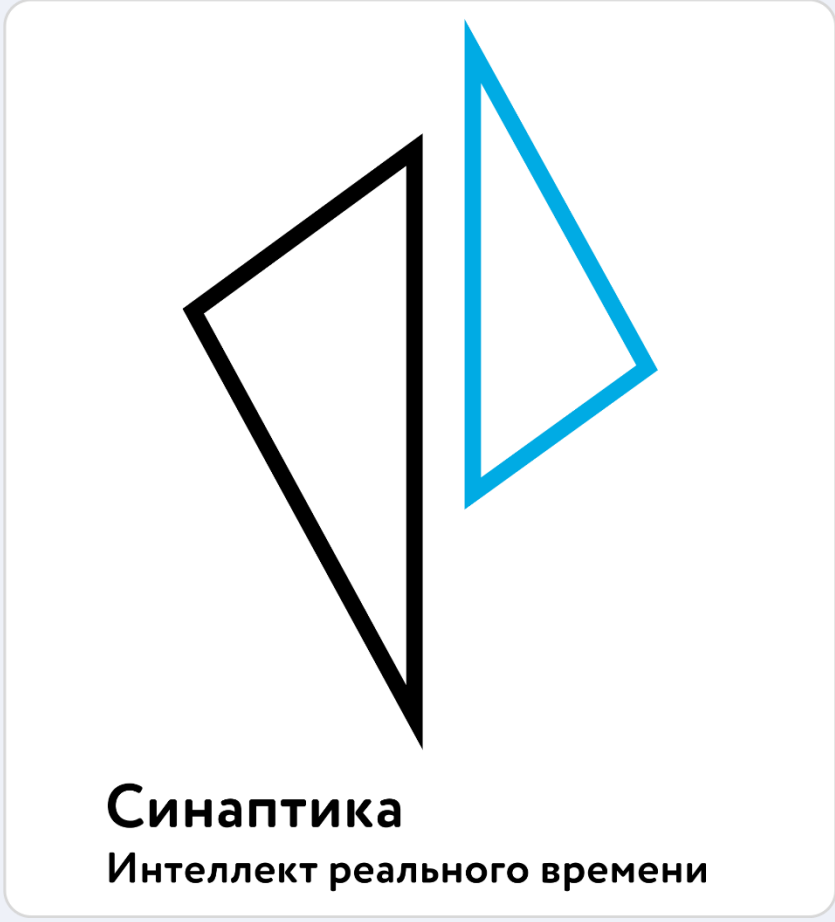
Штурнев Альберт

Техник-программист, отдел нейротехнологий

Про Синаптику

Синаптика предоставляет универсальный набор инструментов для разработчиков

- **Конвейер обработки (Pipeline)**
- **Блок выполнения (Stage)**
- **Подсистема памяти (Memory)**



Модули платформы Синаптика

vision

Модуль для решения задач машинного зрения

anomaly

Модуль для решения задач поиска аномалий

common

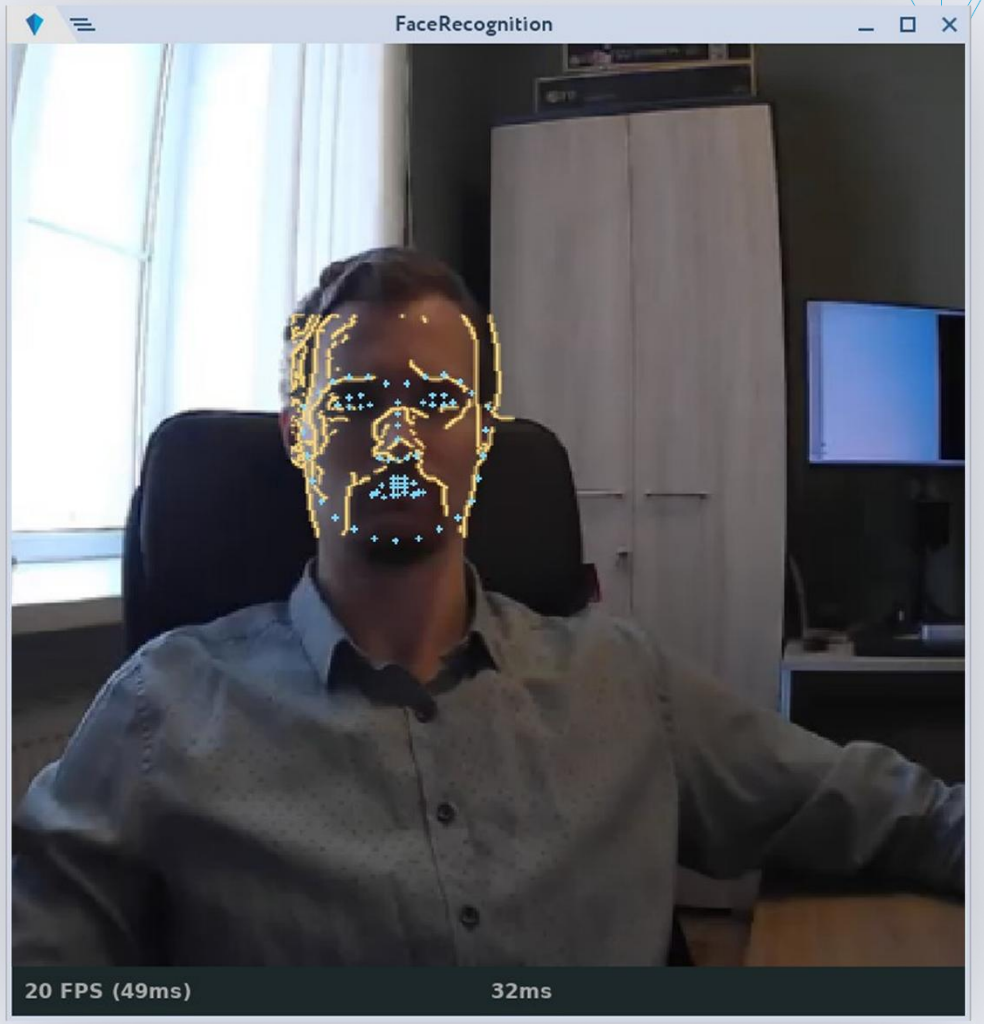
Модуль общих алгоритмов платформы

Больше модулей в рамках дальнейшего развития

Приложение для детекции лиц

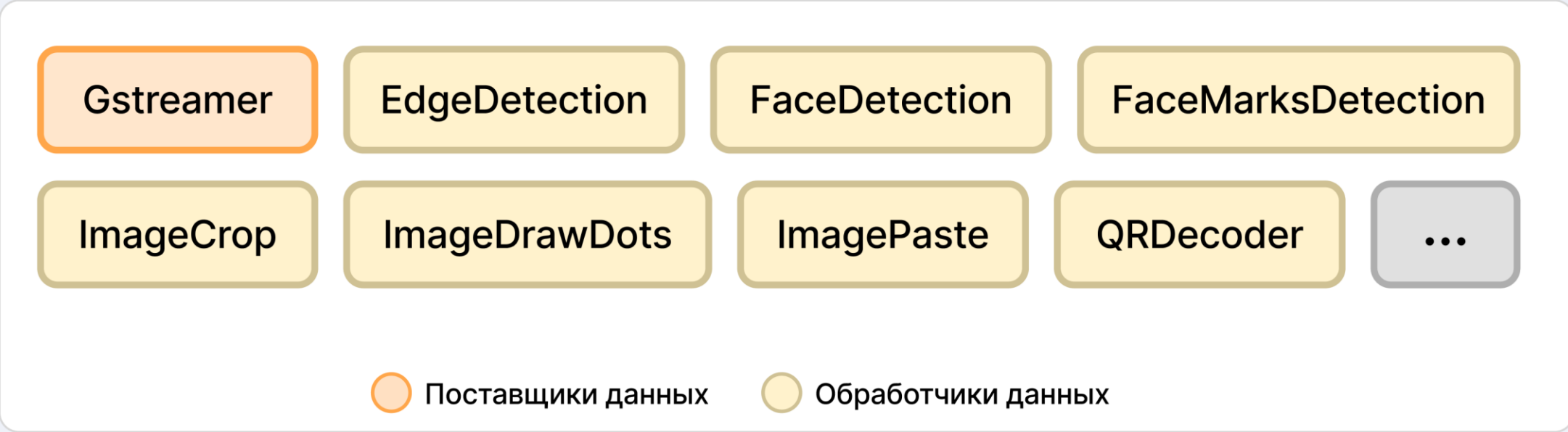
Разберём последовательность этапов:

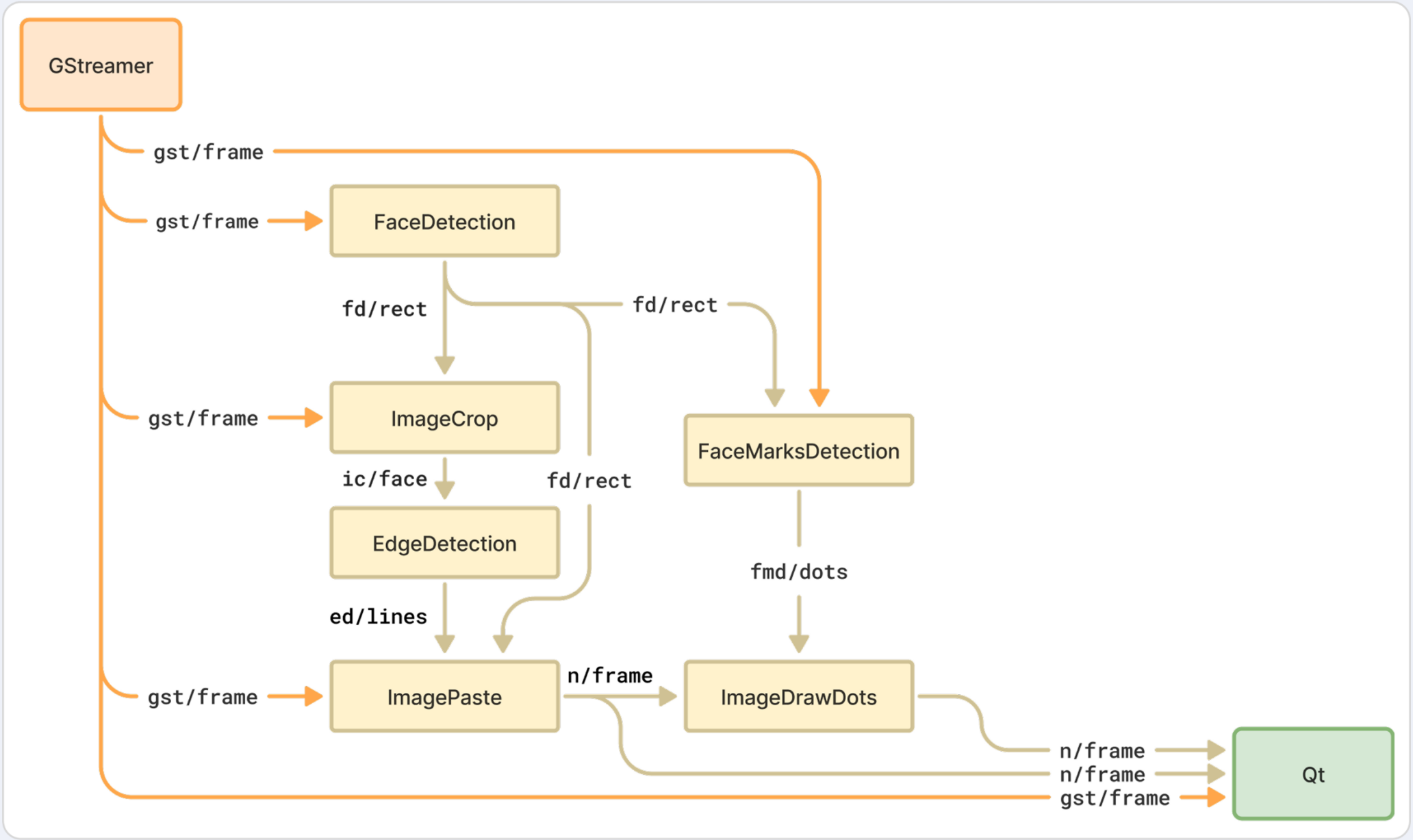
- **Получение картинки с камеры**
- **Получение координат лиц на картинке**
- **Получение и отрисовка контуров лиц**
- **Получение и отрисовка опорных точек для каждого лица**
- **Отрисовка изображения в Qt**



Рассмотрим задачу в контексте Синаптики

Для решения задачи детекции лиц будет использован модуль **vision**, который содержит следующие блоки выполнения:




GStreamer
gst/frame
gst/frame
FaceDetection
fd/rect
fd/rect
gst/frame
ImageCrop
FaceMarksDetection
ic/face
fd/rect
EdgeDetection
fmd/dots
ed/lines
gst/frame
ImagePaste
n/frame
ImageDrawDots
n/frame
n/frame
gst/frame
Qt
Поставщики данных
Обработчики данных
Средства вывода данных

Этапы разработки приложения

- Подключение Синаптики к проекту
- Создание пайплайна
- Добавление блоков выполнения в пайплайн
- Распределение блоков по потокам выполнения
- Запуск пайплайна

Этап 1: Подключить Синаптику к проекту

Подключаем Синаптику к проекту

CMakeLists.txt

```
31  target_link_libraries(
32      ${PROJECT_NAME}
33      sinaptica-core
34      sinaptica-vision
35  )
```

Подключаем заголовочные файлы

main.cpp

```
 6  #include "sinaptica/vision/providers.h"
 7  #include "sinaptica/vision/processors.h"
 8  #include "sinaptica/vision/types.h"
 9
10  #include "sinaptica/core.h"
```

Этап 2: Создание пайплайна

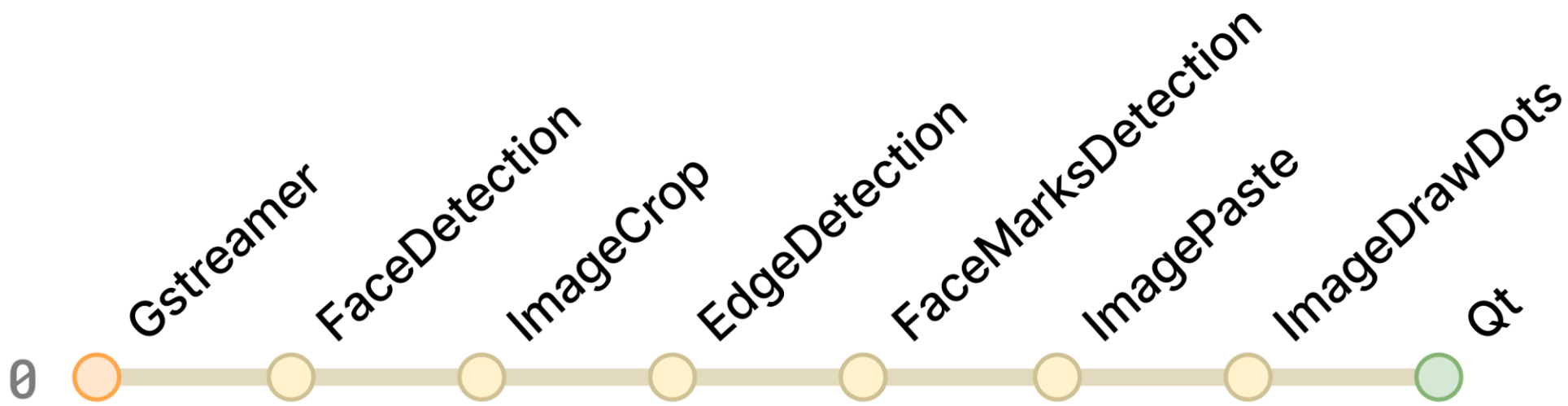
Создаём пайплайн с одним потоком выполнения

main.cpp

```
15  int main(int argc, char *argv[]) {
16      Sinaptica::Init();
17
18      // Создаём пайплайн, указывая количество потоков выполнения
19      Sinaptica::Pipeline P(1);
20
21      // В этом месте будут добавляться блоки выполнения
22      // p.doAttachStage(...)
23
24      // Указываем период для потока выполнения
25      P.setWorkerOptions(0, {.period=1});
26
27    // Запускаем пайплайн
28      P.doRunPipeline();
29      return 0;
30  }
```

Этап 3: Добавляем блоки выполнения в пайплайн

Опираясь на построенный граф данных можем построить следующую последовательность блоков выполнения:



Этап 3.1: Добавляем GStreamer

Создаём gstreamer-пайплайн для получения изображения

main.cpp

```
21    auto constexpr framerate = 20;
22    std::string uri = "rtsp://...";
23
24    auto gstpipeline = "rtspsrc location="+uri+" latency=200";
25    gstpipeline += " ! queue ! rtph264depay ! h264parse ! avdec_h264 ! videoconvert";
26    gstpipeline += " ! videorate ! video/x-raw,framerate="+std::to_string(framerate)+"/1";
27    gstpipeline += " ! videoscale ! video/x-raw,width=700,height=400";
28    gstpipeline += " ! appsink name=sink max-buffers=3 sync=1 caps=video/x-raw,format=BGR";
```

Пайплайн Gstreamer передаётся в стейдж Gstreamer

main.cpp

```
32    // Получение картинки с камеры с помощью gstreamer пайплайна
33    P.doAttachStage(0, Sinaptica::Providers::GStreamer::Stage(
34      {.gst_pipeline_parameters=gstpipeline,
35       .output_name="gst/frame"}));
```

Этап 3.2 Добавляем FaceDetection

Добавляем обработчик для детектирования лиц на изображении с камеры

main.cpp

```
37    // Получение списка прямоугольников для каждого лица на картинке
38    P.doAttachStage(0, Sinaptica::Processors::FaceDetection::Stage(
39      {.input_name="gst/frame",
40       .output_name = "fd/rect"}));
```

Описание входов и выходов блока FaceDetection:

Назначение	Описание	Допустимые типы объекта памяти
Вход	Изображение, на котором должен быть выполнен поиск лиц	Sinaptica::Types::Image
Выход	Список прямоугольников, содержащих координаты каждого лица, найденного на изображении	Sinaptica::Types::CVRectList

Этап 3.3: Добавляем ImageCrop

Добавляем обработчик для вырезания лица с изображения

main.cpp

```
42    // Получение картинки с вырезанным лицом
43    P.doAttachStage(0, Sinaptica::Processors::ImageCrop::Stage(
44      {.input1_name="gst/frame",
45       .input2_name="fd/rect",
46       .output_name="ic/face"}));
```

Описание входов и выходов блока ImageCrop:

Назначение	Описание	Допустимые типы объекта памяти
Вход 1	Исходное изображение	Sinaptica::Types::Image
Вход 2	Координаты области, из которой необходимо вырезать изображение	Sinaptica::Types::CVRectList
Выход	Вырезанное изображение	Sinaptica::Types::Image

Этап 3.4: Добавляем EdgeDetection

Добавляем обработчик для выделения контуров объектов на изображении

main.cpp

```
48    // Получение контуров лица
49    P.doAttachStage(0, Sinaptica::Processors::EdgeDetection::Stage(
50      {.input_name="ic/face",
51       .output_name="ed/lines",
52       .t1=50, .t2=55}));
```

Описание входов и выходов блока EdgeDetection:

Назначение	Описание	Допустимые типы объекта памяти
Вход	Изображение, на котором должно быть выполнено выделение контуров	Sinaptica::Types::Image
Выход	Изображение с выделенными контурами объектов	Sinaptica::Types::Image

Этап 3.5: Добавляем FaceMarksDetection

Добавляем обработчик для определения лицевых опорных точек

main.cpp

```
54    // Получение опорных точек для лица
55    P.doAttachStage(0, Sinaptica::Processors::FaceMarksDetection::Stage(
56      {.model_name="../share/resources/lbfmodel.yaml",
57       .input1_name="gst/frame",
58       .input2_name="fd/rect",
59       .output_name="fmd/dots"}));
```

Описание входов и выходов блока FaceMarksDetection:

Назначение	Описание	Допустимые типы объекта памяти
Вход 1	Изображение, содержащее лица для детектирования опорных точек	Sinaptica::Types::Image
Вход 2	Список прямоугольников, описывающих положение лиц на изображении	Sinaptica::Types::CVRectList
Выход	Облако опорных точек для каждого лица	Sinaptica::Types::CVDotCloud

Этап 3.6: Добавляем ImagePaste

Добавляем обработчик для результатов предыдущих блоков в новый объект

main.cpp

```
61    // Вставка картинки с картинки с камеры в новый объект памяти
62    P.doAttachStage(0, Sinaptica::Processors::ImagePaste::Stage(
63      {.input1_name="gst/frame",
64       .input2_name="ed/lines",
65       .input3_name="fd/rect",
66       .output_name="n/frame"}));
```

Описание входов и выходов блока ImagePaste:

Назначение	Описание	Допустимые типы объекта памяти
Вход 1	Исходное изображение	Sinaptica::Types::Image
Вход 2	Изображение для вставки	Sinaptica::Types::Image
Вход 3	Координаты области, в которую необходимо вставить изображение	Sinaptica::Types::CVRectList
Выход	Итоговое изображение, если режим модификации не активирован	Sinaptica::Types::Image

Этап 3.7: Добавляем ImageDrawDots

Добавляем отрисовку лицевых опорных точек на новом объекте памяти

main.cpp

```
68    // Отрисовка лицевых опорных точек
69    P.doAttachStage(0, Sinaptica::Processors::ImageDrawDots::Stage(
70      {.input1_name="n/frame",
71       .input2_name="fmd/dots"}));
```

Описание входов и выходов блока ImageDrawDots:

Назначение	Описание	Допустимые типы объекта памяти
Вход 1	Исходное изображение	Sinaptica::Types::Image
Вход 2	Облако точек для отрисовки	Sinaptica::Types::CVDotCloud

Этап 3.8: Добавляем вывод в Qt

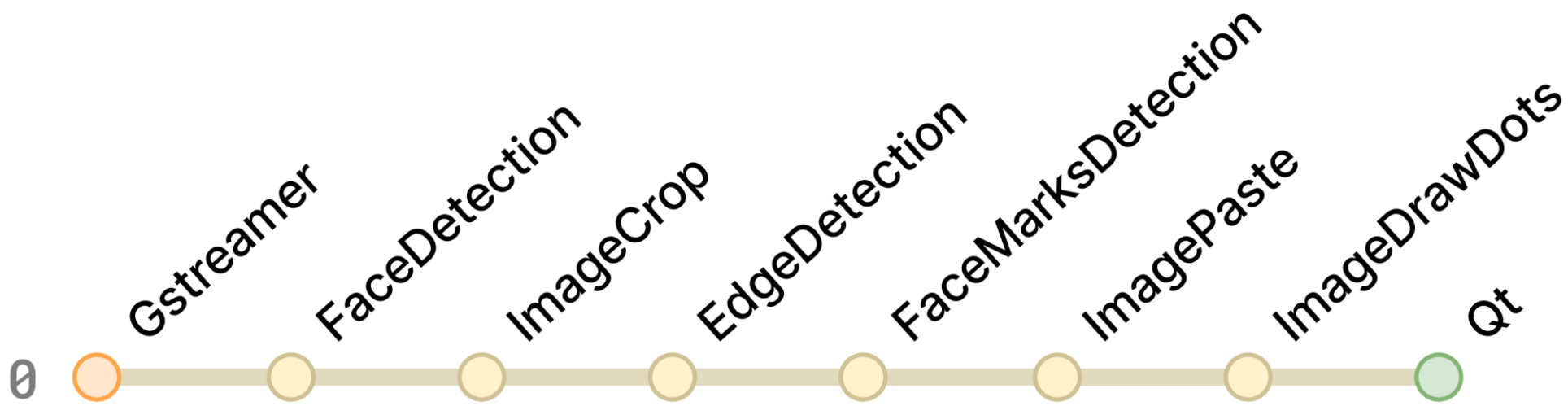
Добавляем новый стейдж с лямбда-функцией, в которой опишем логику для получения изображения через объект памяти n/frame и вывод в приложение Qt

main.cpp

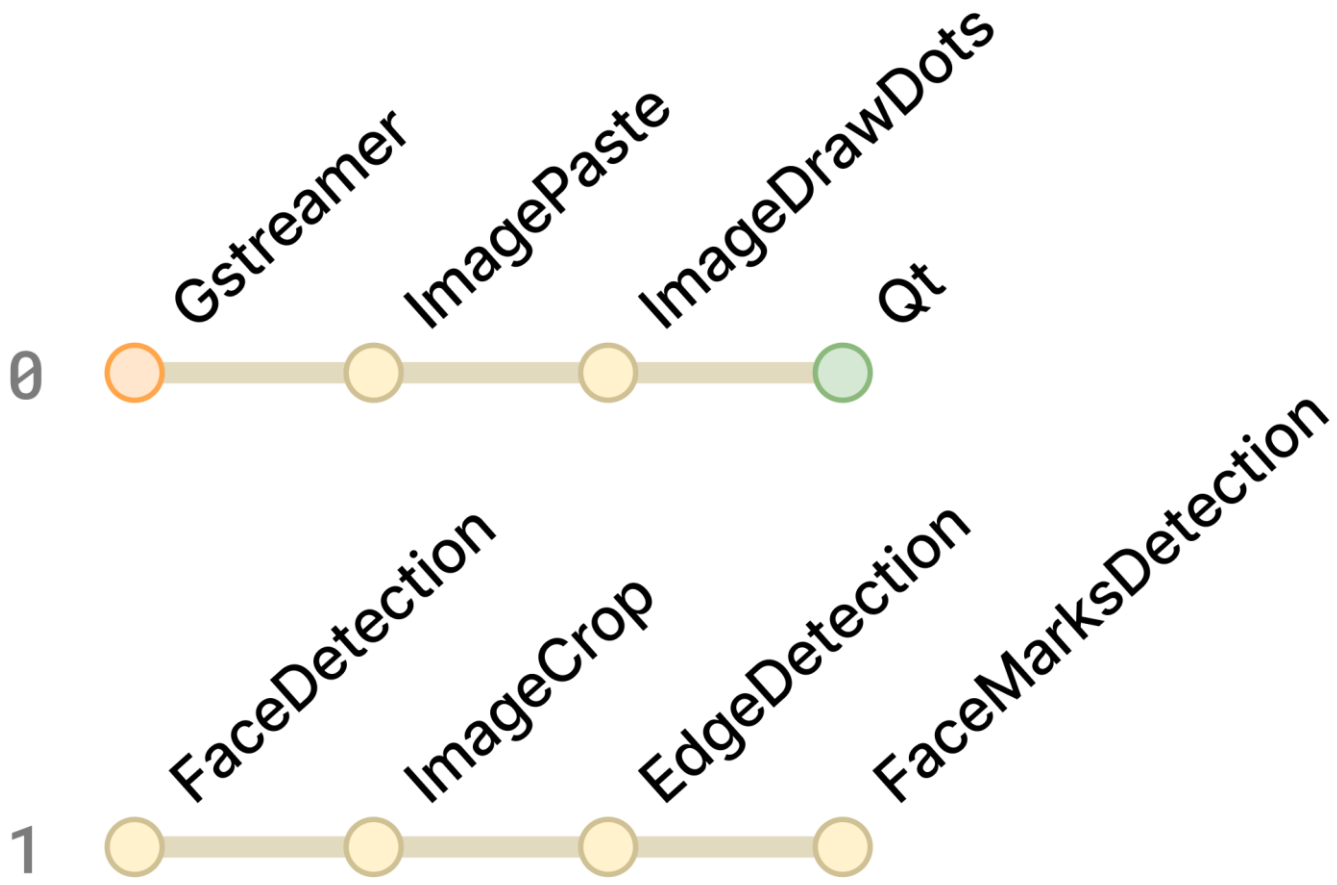
```
73      // Вывод в Qt
74      P.doAttachStage(0, Sinaptica::Pipeline::Stage([&viewer](auto info) {
75          std::string frame_name = "n/frame";
76          if (!Sinaptica::Memory::TryFindMemoryObject(frame_name)) return;
77
78          auto transaction = Sinaptica::Memory::Transaction<Sinaptica::Types::Image>(frame_name);
79          auto image = transaction -> doRead();
80          auto frame = QImage((const uchar*)image.ptr, image.width, image.height, QImage::Format_BGR888);
81          Q_EMIT viewer.doDrawProcessedFrame(frame.copy());
82      }));
```

Этап 3.9: Рассмотрим получившиеся пайплайн

Таким образом все нужные блоки выполнения были добавлены в пайплайн.
Единственная его проблема - однопоточность



Этап 4: Распределение по потокам выполнения



Этап 4.1: Реализация в коде

main.cpp

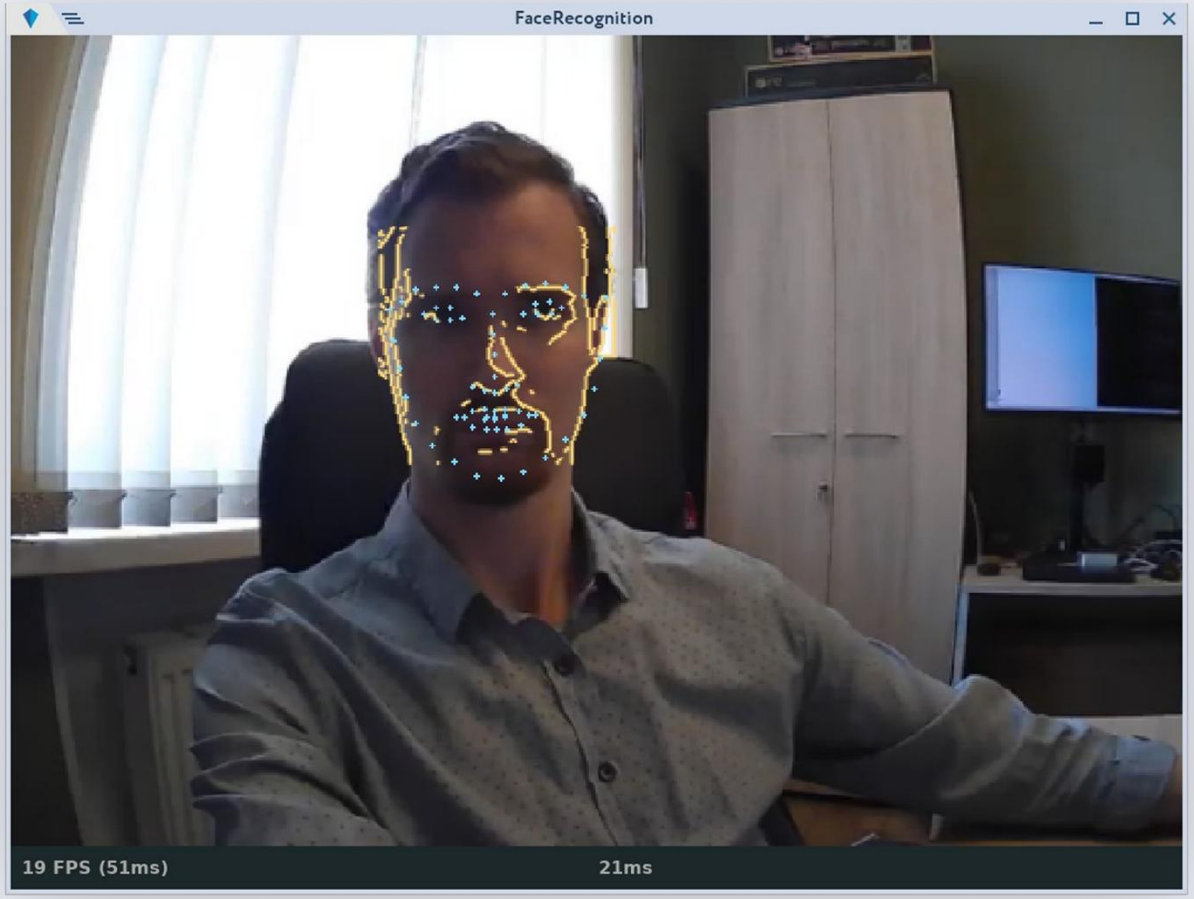
```
28      Sinaptica::Init();
29
30      Sinaptica::Pipeline P(2);
31
32      P.doAttachStage(0, Sinaptica::Providers::GStreamer::Stage({.gst_pipeline_parameters=gstpipeline, .output_name="gst/frame"}));
33      P.doAttachStage(1, Sinaptica::Processors::FaceDetection::Stage({.input_name="gst/frame", .output_name = "fd/rect"}));
34      P.doAttachStage(1, Sinaptica::Processors::ImageCrop::Stage({.input1_name="gst/frame", .input2_name="fd/rect",
35                                                                  .output_name="ic/face"}));
36      P.doAttachStage(1, Sinaptica::Processors::EdgeDetection::Stage({.input_name="ic/face", .output_name="ed/lines", .t1=50, .t2=55}));
37      P.doAttachStage(1, Sinaptica::Processors::FaceMarksDetection::Stage({.model_name="../share/resources/lbfmodel.yaml",
38                                                                           .input1_name="gst/frame", .input2_name="fd/rect",
39                                                                           .output_name="fmd/dots"}));
40      P.doAttachStage(0, Sinaptica::Processors::ImagePaste::Stage({.input1_name="gst/frame", .input2_name="ed/lines",
41                                                                   .input3_name="fd/rect", .output_name="n/frame"}));
42      P.doAttachStage(0, Sinaptica::Processors::ImageDrawDots::Stage({.input1_name="n/frame", .input2_name="fmd/dots"}));
43      P.doAttachStage(0, Sinaptica::Pipeline::Stage([&viewer](auto info) {
44          std::string frame_name = "n/frame";
45          if (!Sinaptica::Memory::TryFindMemoryObject(frame_name)) return;
46
47          auto transaction = Sinaptica::Memory::Transaction<Sinaptica::Types::Image>(frame_name);
48          auto image = transaction -> doRead();
49          auto frame = QImage((const uchar*)image.ptr, image.width, image.height, QImage::Format_BGR888);
50          Q_EMIT viewer.doDrawProcessedFrame(frame.copy());
51      }));
52
53      P.setWorkerOptions(0, {.period=1});
54      P.setWorkerOptions(1, {.period=50});
55      P.doRunPipeline();
```

Этап 5: Запуск пайплайна

Запуск пайплайна происходит в методе doRunPipeline()

main.cpp

```
53        P.setWorkerOptions(0, {.period=1});
54        P.setWorkerOptions(1, {.period=50});
55        P.doRunPipeline();
56
57      return a.exec();
58  }
```



Заключение

В мастер-классе были разобраны следующие темы:

- **Подключение Синаптики к проекту**
- **Конвейеры обработки и блоки выполнения**
- **Потоки выполнения**
- **Модуль vision**
- **Подсистема памяти**

Материалы

Документация Синаптики

Пакеты в составе Синаптики

Репозиторий с кодом из мастер-класса

СВД
Встраиваемые Системы

Синаптика
Интеллект реального времени

Спасибо за внимание!

Штурнев Альберт

Техник-программист, отдел нейротехнологий

www.kpda.ru

+7 812 317-55-56