GStreamer

gstreamer

open source multimedia framework

Кратко обо мне.

- Разработчик одного из подразделений ITransition.
- Четко выраженной сферы интересов нет, по возможности использую весь багаж знаний полученный в альма-матер, и расширяемый за последние 10 лет – сетевые протоколы, микропроцессорные архитектуры, внутренние механизмы операционных систем, обработка изображений.
- На данный момент занимаюсь разработкой сервера системы видеонаблюдения на базе платформы iMX6.
- Ну и конечно же C/C++ ☺

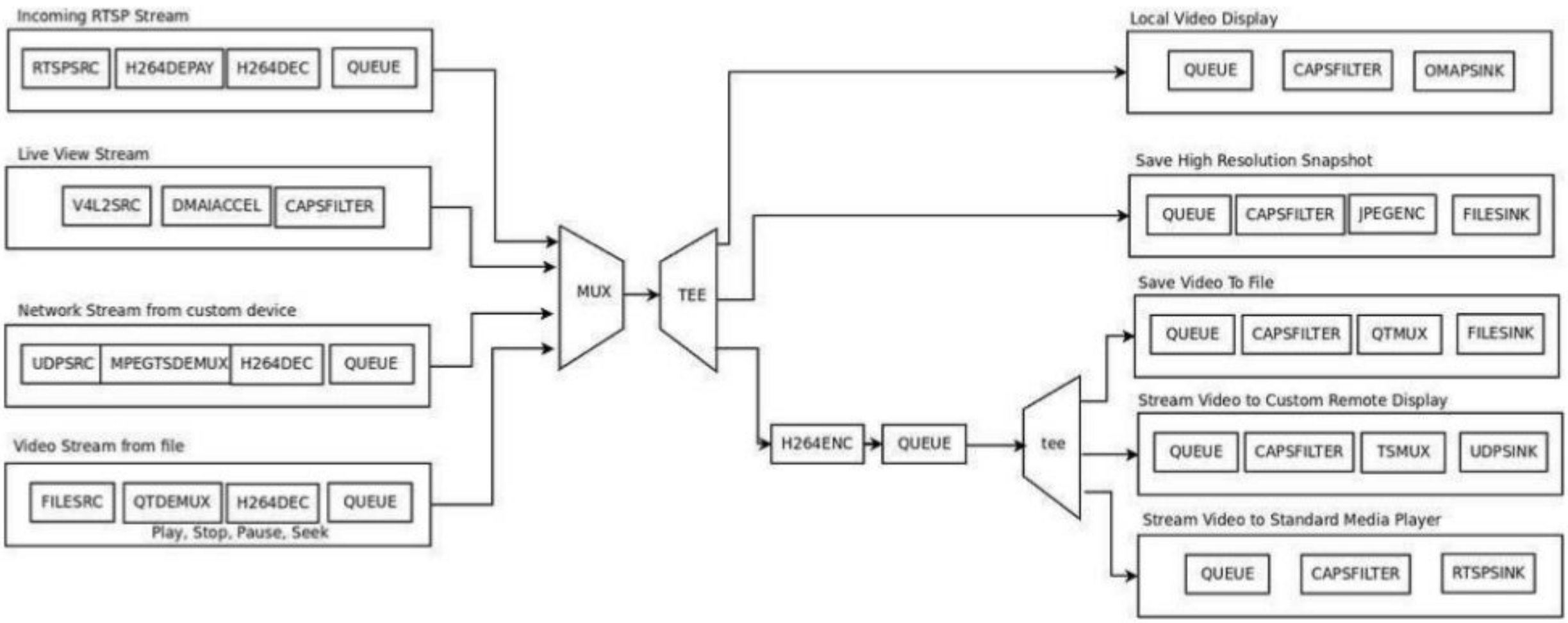
Gstreamer: что это?

- Фреймворк, для построения мультимедийных приложений.
- GStreamer написан на языке C, с использованием ООП-парадигмы, которая реализована на базе GObject.
- Приложение строится путем объединения элементарных строительных блоков – элементов, в цепочку, которая образует путь следования данных – pipeline.

Сравните

- Linux Pipeline:
- `grep 'Starting' /var/log/boot.log | sort > recently-started.txt`
- Gstreamer pipeline:
- `gst-launch-1.0 filesrc location='movie.mkv' ! matroskademux ! avdec_h264 ! avenc_mpeg2video ! filesink location='videostream.mp4'`

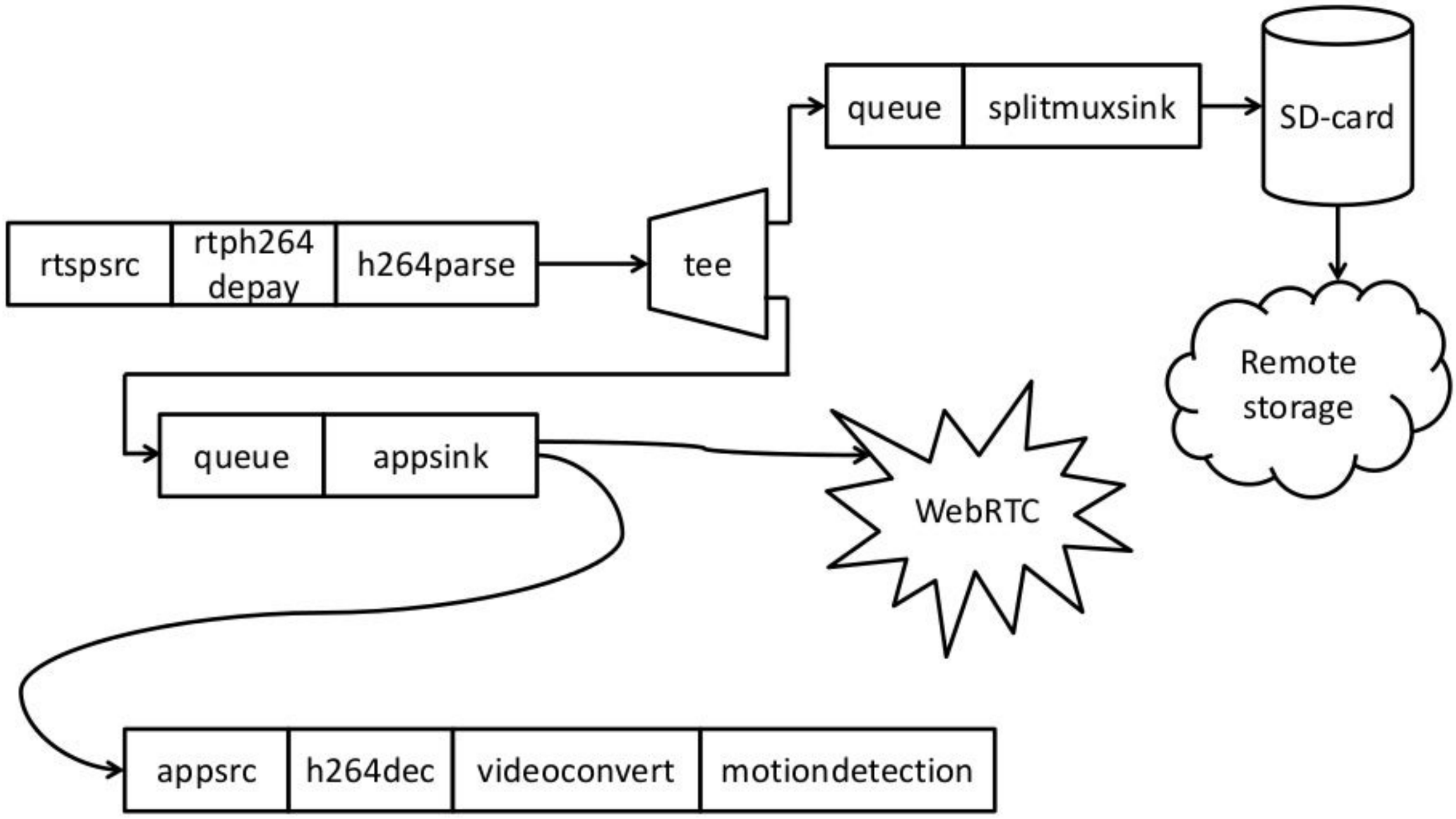
GStreamer:пример использования



gstreamer

open source multimedia framework

GStreamer:пример использования



gstreamer
open source multimedia framework

Аналоги: ffmpeg

По набору предоставляемых возможностей и возможным областям применения, наиболее близким аналогом является ffmpeg. Однако они отличаются «философией» написания приложения.

ffmpeg - Доменные сущности (файлы, потоки, кодеки) представляются разработчику в виде контекстов, для работы с контекстами используется предоставляемый API.

GStreamer – разработчику предоставляется набор функционально законченных элементов, которые комбинируются и объединяются, образуя путь следования данных в порядке их обработки.

На мой взгляд, основная сложность использования ffmpeg для обработки медиа-данных – это необходимость «ручной» синхронизации потоков к единой шкале времени.

GStreamer скрывает детали синхронизации времени, кроме того, обеспечивает механизм согласования форматов данных. Модульная архитектура способствует разработке элементов, поддерживающих аппаратные ускорители.

GStreamer: возможности

- Принимать и передавать аудио/видео данные, используя протоколы HTTP, RTSP/RTP и др.
- Разбирать и собирать потоки в разных форматах (контейнерах): MPEG, AVI, ASF, FLV, MKV и др.
- Декодировать/кодировать потоки в различных комбинациях кодеков (в том числе можно задействовать и аппаратные ресурсы, реализующие необходимый алгоритм)
- Получать потоки данных из различных источников и отправлять их (потоки данных) в различные приемники

gstreamer
open source multimedia framework

Компоненты

- Набор плагинов, поставляемых в виде динамических библиотек.
- Утилиты командной строки, предназначенные для запуска pipelin'a, перечисления списка имеющихся элементов и их свойств (gst-launch, gst-inspect).

Gstreamer: инструментарий

- Уже упомянутые утилиты командной строки: gst-launch, gst-inspect
- Возможность визуализации pipeline в формат graphviz.
- Основным WYSIWYG-построителем является GStreamer Pipeline Editor.
- Система логирования с широким диапазоном log-level

Строительные блоки: элементы

- Элементы являются минимальными строительными блоками. 5 категорий:
  - Источники данных
  - Фильтры
  - Приемники данных
  - Анализаторы потока
  - Вспомогательные элементы

Строительные блоки: элементы

- Любой элемент является конечным автоматом, в течение времени функционирования приложения переходя из одного состояния в другое.
- В соответствии с философией GLib элементы имеют именованные свойства.
- Для соединения элементов в цепочку используются точки подключения, т.н. pads:
  - sinkpad – вход потока данных в элемент
  - srcpad – вывод потока данных из элемента

  У элемента может быть несколько точек подключения.

Элементы: источники

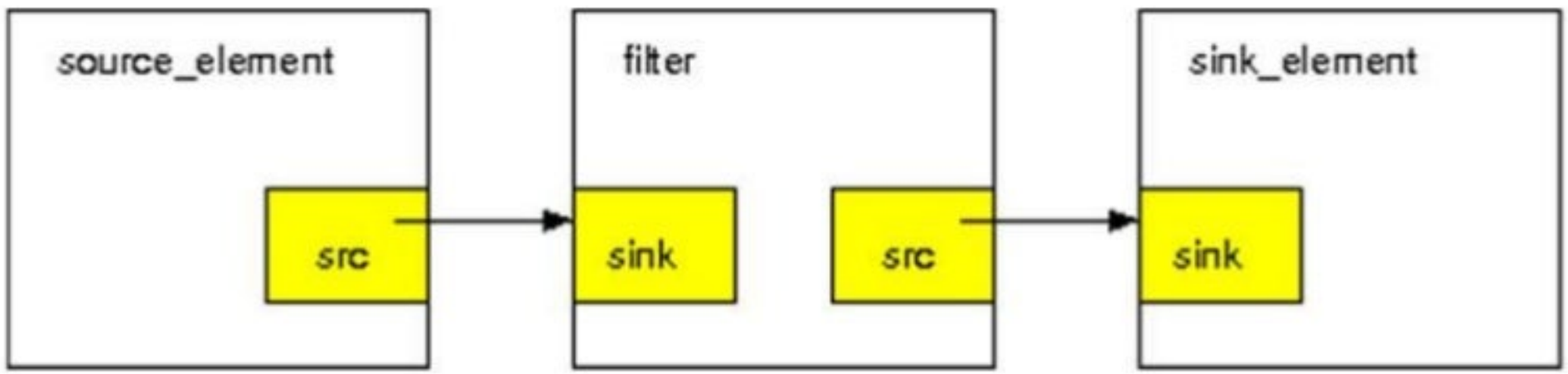
- Источники – это класс плагинов, которые позволяют читать медиаданные из различных источников:
  - Filesystem (filesrc, multifilesrc)
  - Network (souphttpsrc, rtspsrc, udpsrc)
  - Devices (alsasrc, v4l2src)
  - Others (fakesrc, audiotestsrc, videotestsrc)

Элементы: приемники

- Приемники – это оконечные элементы pipeline, они “выводят медиаданные” за его пределы.
  - Filesystem (filesink, multifilesink)
  - X-server (ximagesink, xvimagesink)
  - Network (udpsink)

Элементы: фильтры

- Фильтры – это элементы, которые выполняют различные преобразования над потоком данных. Это, пожалуй, самый обширный класс элементов, который объединяет мультиплексоры/демультиплексоры потоков, парсеры, кодеры/декодеры и многое другое.



gstreamer
open source multimedia framework

Элементы: прочие элементы

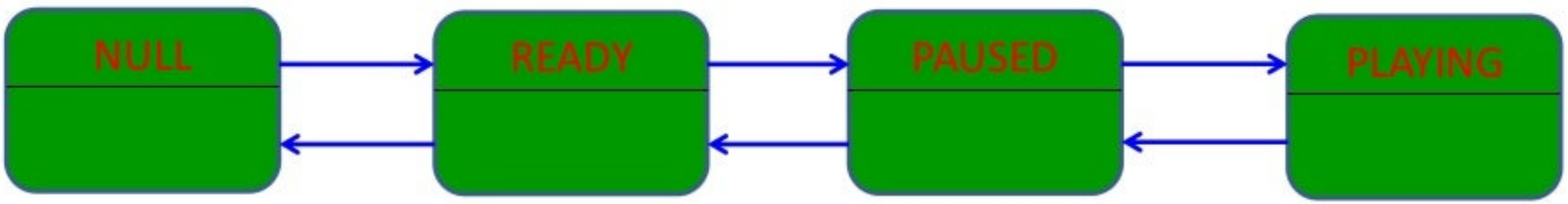
- Анализаторы потока – элементы этой категории пропускают через себя поток данных, не модифицируя его, но изменняют свое состояние и/или генерируют событие.
- Вспомогательные элементы – к этой категории относятся очереди, “разветвители” потока данных и некоторые другие элементы.

Элемент: конечный-автомат

- Состояния элемента в течение времени жизни:
  - **NULL** – элемент неактивен и не владеет никакими ресурсами
  - **READY** – элементу предоставлена часть ресурсов, не относящихся напрямую к обработке потока данных (динамические библиотеки, ресурсы аппаратуры)
  - **PAUSED** – элемент готов принимать и обрабатывать потоки данных
  - **PLAYING** – элемент обрабатывает поток данных

Элемент: конечный автомат

- Диаграмма переходов

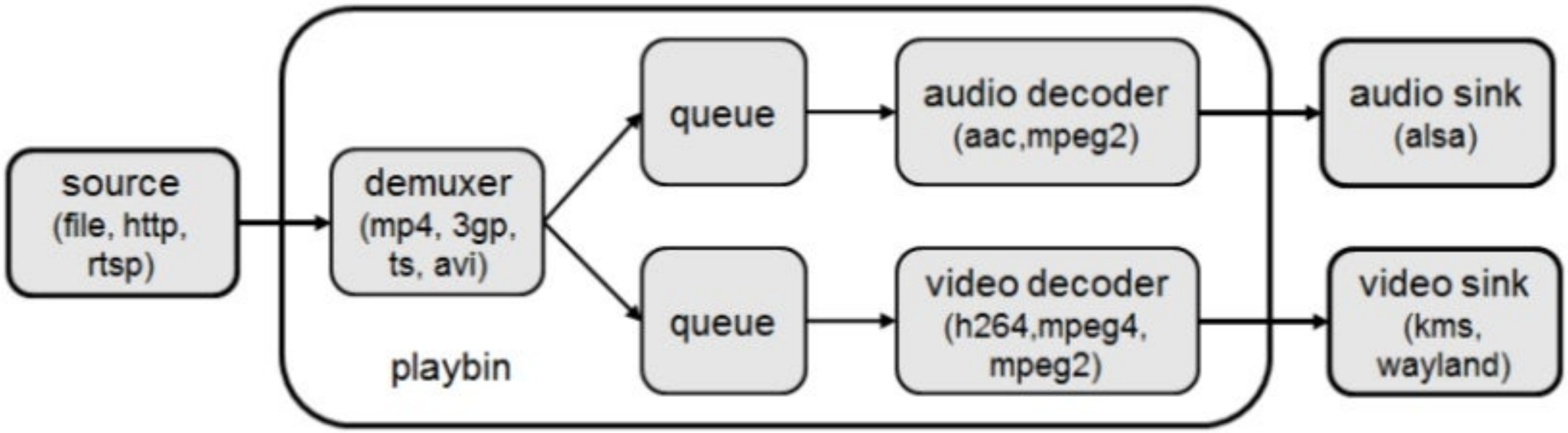


gstreamer
open source multimedia framework

Строительные блоки: контейнеры

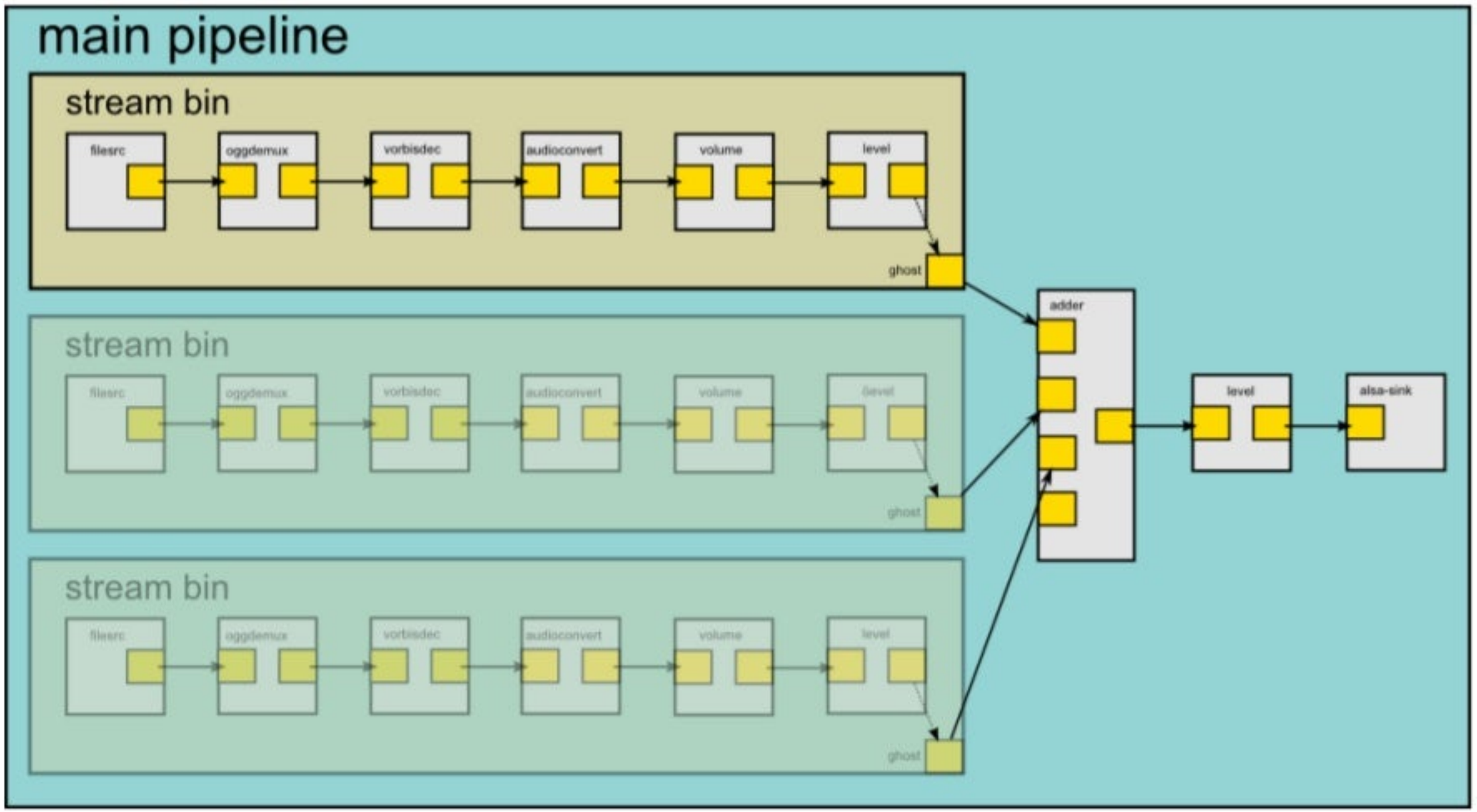
- Контейнеры представляют собой особую категорию элементов.
- Контейнеры объединяют несколько элементов, позволяя управлять ими одновременно (например, изменять состояние)

Контейнеры



gstreamer
open source multimedia framework

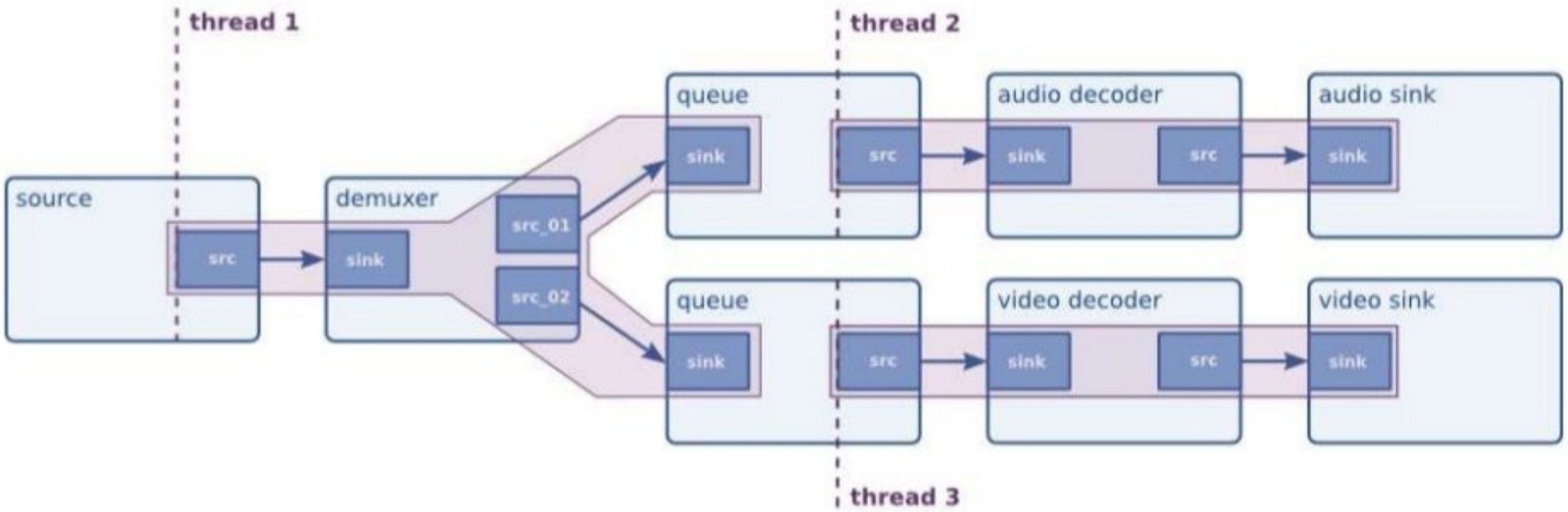
Контейнеры



gstreamer

open source multimedia framework

Pipeline: пример



gstreamer

open source multimedia framework

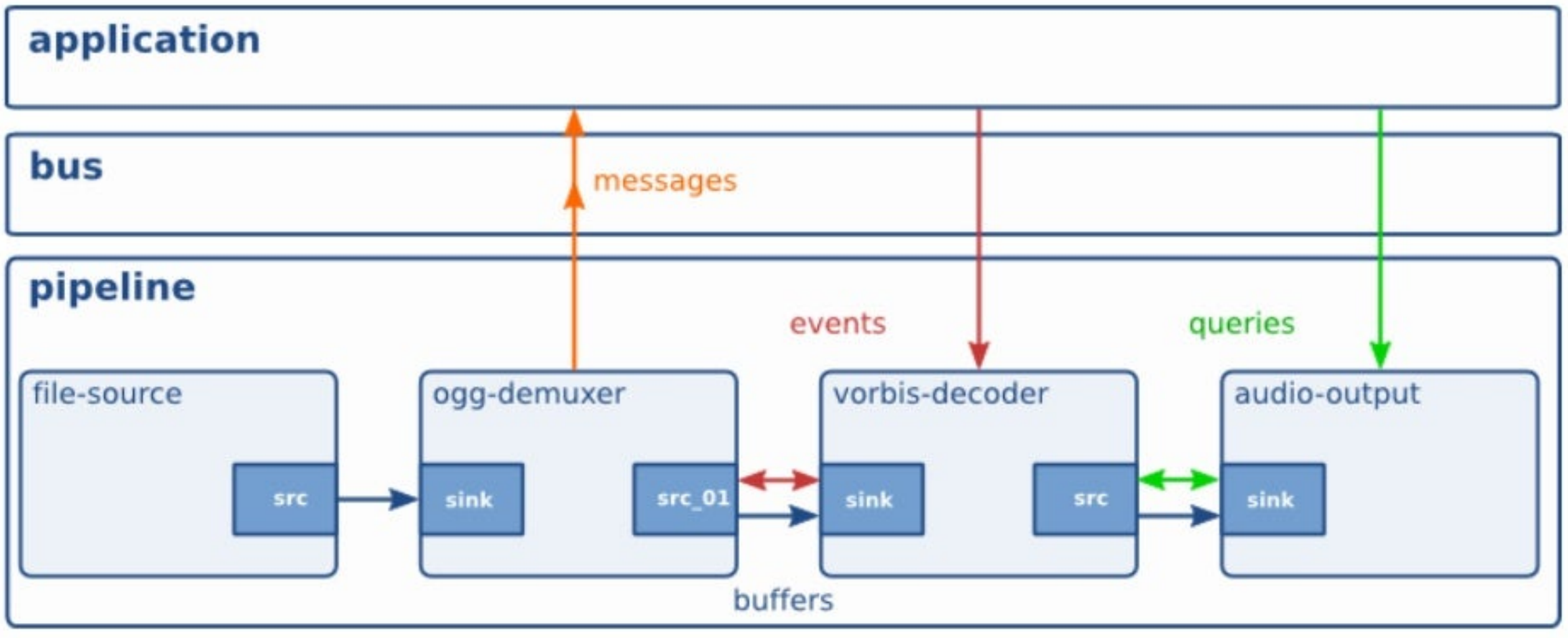
Pipeline: пример

```
gst-launch-1.0 filesrc location='movie.mkv'
  ! matroskademux name=d
  ! queue ! avdec_h264 ! xvimagesink
d. ! queue ! avdec_mp3 ! alsasink
```

Взаимодействие элементов

- Для взаимодействия элементов между собой и с приложением верхнего уровня используются следующие сущности:
  - Pad
  - Bus
  - Buffers
  - Events
  - Messages
  - Query

Взаимодействие элементов



gstreamer
open source multimedia framework

Gstreamer: пишем код

```
void Recorder::init_pipeline() {
    std::string pipeline("rtspsrc name=src ! rtph264_depay ! H264parse ! tee name=t"
                         " t. ! queue ! appsink name=h264sink "
                         " t. ! queue ! splitmuxsink max-size-time=90000000000 muxer=qtmux");
    _pipeline = gst_parse_launch(pipeline.c_str(), NULL);
    _rtspsrc = gst_bin_get_by_name(GST_BIN(_pipeline), "src");
    _h264sink = gst_bin_get_by_name(GST_BIN(_pipeline), "h264sink");
    g_object_set(_h264sink, "emit-signals", TRUE, NULL);
    _new_sample_id = g_signal_connect(_h264sink, "new-sample",
                                      G_CALLBACK(new_sample), this);
    GstBus *bus = gst_pipeline_get_bus(GST_PIPELINE(_pipeline));
    _bus_watch_id = gst_bus_add_watch(bus, on_bus_message, this);
    g_object_unref(bus);
}
```

Gstreamer: пишем код

```
GstFlowReturn Recorder::new_sample(GstElement *element, gpointer userdata) {
    Recorder *recorder = static_cast<Recorder*>(userdata);
    GstSample *sample = NULL;
    g_signal_emit_by_name(element, "pull-sample", &sample);
    if (sample) {
        if (element == recorder->_appsink)
            recorder->_detector.push_sample(sample);
        gst_sample_unref(sample);
    }
    return GST_FLOW_OK;
}
```

open source multimedia framework

Gstreamer: пишем код

```
gboolean Recorder::on_bus_message(GstBus *bus, GstMessage *msg, gpointer userdata) {
    Recorder *recorder = static_cast<Recorder*>(userdata);
    switch (GST_MESSAGE_TYPE(msg)) {
        case GST_MESSAGE_ERROR:
            recorder->on_message_error(msg);
            break;
        case GST_MESSAGE_EOS:
            recorder->on_message_eos(msg);
            break;
        case GST_MESSAGE_STATE_CHANGED:
            recorder->on_message_state_changed(msg);
            break;
        default:
            recorder->on_message_other(msg);
    }
    return TRUE;
}
```

GStreamer: пишем код

```
void Recorder::start() {
        GstStateChangeReturn ret =  gst_element_set_state(_pipeline, GST_STATE_PLAYING);
        _detector.start();
}
void Recorder::stop() {
        _detector.stop();
        GstStateChangeReturn ret =  gst_element_set_state(_pipeline, GST_STATE_NULL);
}
```

open source multimedia framework

GStreamer: пишем код

```
void Detector::init_pipeline() {

    std::string pipeline("appsrc ! "

    #if defined BUILD_PROFILE_IMX

        " vpudec ! imxvideoconvert_g2d ! capsfilter caps=\"video/x-raw,width=640,height=480\" !
          videoconvert ! "

    #elif defined BUILD_PROFILE_PC

        " avdec_h264 ! videoconvert ! videoscale ! capsfilter caps=\"video/x-
          raw,width=640,height=480, format=RGB\" !  "

#else

#error "Undefined build profile"

#endif

        " motioncells name=mdcells display=FALSE gridx=32 gridy=32 ! videorate name=vrate ");

_pipeline = gst_parse_launch(pipeline.c_str(), NULL);
```

GStreamer: пишем код

```
    _mdcells = gst_bin_get_by_name(GST_BIN(_pipeline), “mdcells”);
    _vrate = gst_bin_get_by_name(GST_BIN(_pipeline), “vrate”);
    GstBus *bus = gst_pipeline_get_bus(GST_PIPELINE(_pipeline));
    _bus_watch_id = gst_bus_add_watch(bus, on_bus_message, this);
    g_object_unref(bus);
}
```

Gstreamer: пишем код

```
gboolean Detector::on_bus_message(GstBus *bus, GstMessage *msg, gpointer userdata)
{
    Detector *detector = static_cast<Detector*>(userdata);
    if (GST_MESSAGE_TYPE(msg) == GST_MESSAGE_ELEMENT
        && GST_MESSAGE_SRC(msg) == _mdcells) {
        const GstStructure *msg_struct = gst_message_get_structure(msg);
        if (gst_structure_has_name(msg_struct, "motion") {
            if (gst_structure_has_field(msg_struct, "motion_begin")
                _recorder.start_recording();
            else if (gst_structure_has_field(msg_struct, "motion_finished")
                _recorder.stop_recording();
        }
    }
    return TRUE;
}
```

GStreamer: итоги

Подводя итоги, я хочу отметить некоторые достоинства и недостатки фреймворка:

- Достоинства
  - Возможность быстрого прототипирования сложных схем обработки аудио/видеоданных
  - Модульная архитектура
  - Поддержка широкого спектра устройств ввода-вывода
- Недостатки
  - Написан на C
  - Далеко не все элементы качественно реализованы