Internet & Multimedia

Gstreamer – concepts and examples

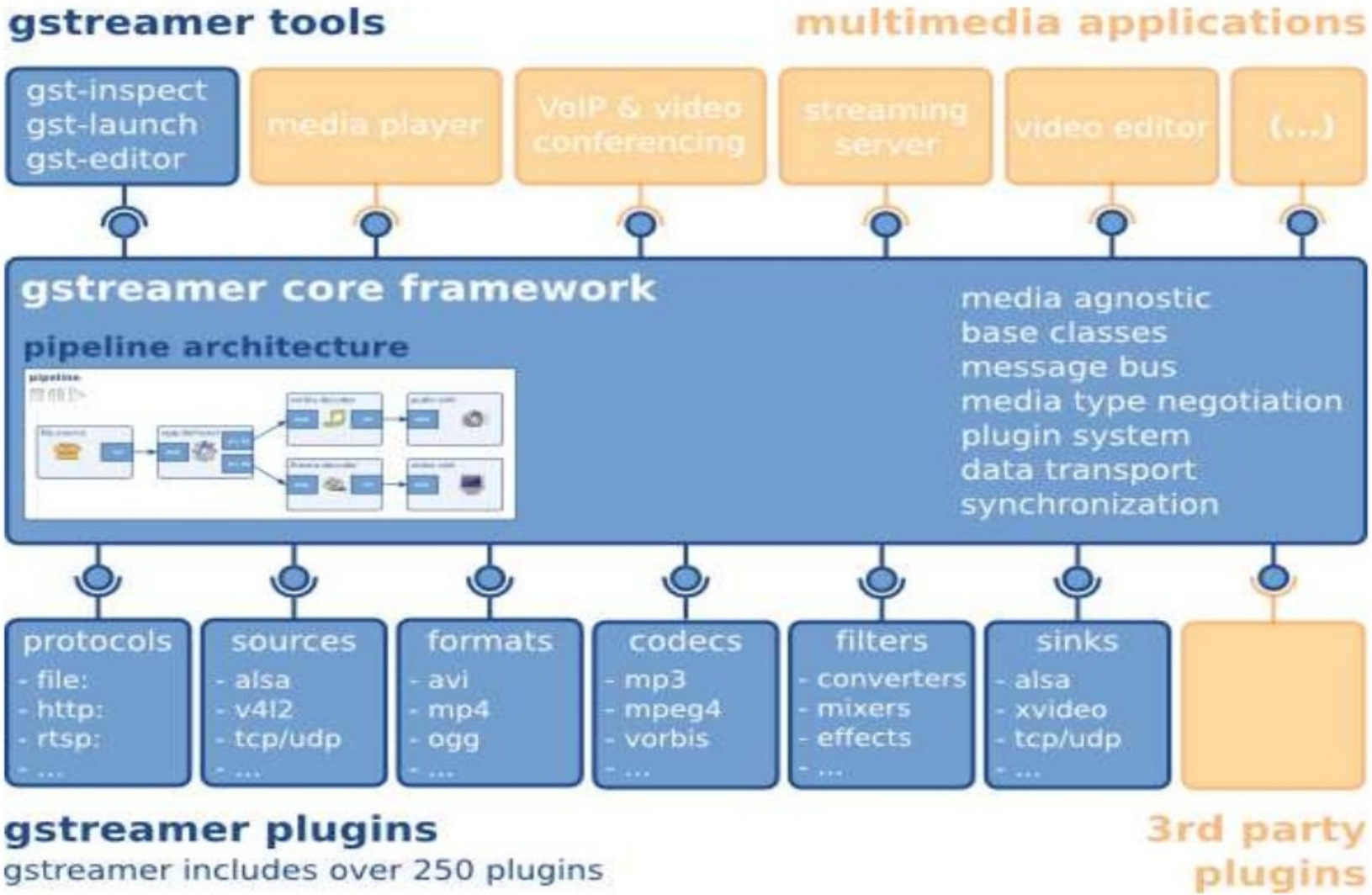
P. Bakowski

bako@ieee.org

What is Gstreamer

- GStreamer is a pipeline-based multimedia framework written in the C programming language with the type system based on **GObject**.
- GStreamer allows a programmer to create a variety of media-handling components, including simple audio playback, audio and video playback, recording, streaming and editing.
- The pipeline design serves as a base to create many types of multimedia applications such as video editors, streaming media broadcasters and media players.

Gstreamer – global overview



Gstreamer – technical overview

GStreamer processes media by connecting a number of processing elements into a pipeline.

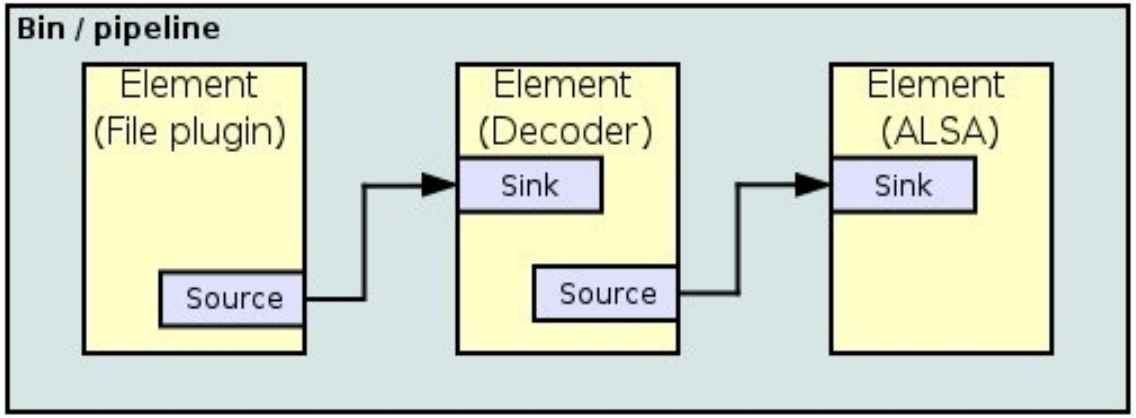
Each element is provided by a plug-in. Elements can be grouped into bins, which can be further aggregated, thus forming a hierarchical graph.

Elements communicate by means of pads. A source pad on one element can be connected to a sink pad on another.

When the pipeline is in the playing state, data buffers flow from the source pad to the sink pad. Pads negotiate the kind of data that will be sent using capabilities.

Gstreamer – technical overview

The diagram below could exemplify playing an MP3 file using GStreamer.
The file source reads an MP3 file from a computer's hard-drive and sends it to the MP3 decoder.
The decoder decodes the file data and converts it into PCM samples which then pass to the ALSA sound-driver.
The ALSA sound-driver sends the PCM sound samples to the computer's speakers.



Gstreamer – plugins

GStreamer uses a plug-in architecture which makes the most of GStreamer's functionality implemented as shared libraries.
GStreamer's base functionality contains functions for registering and loading plug-ins and for providing the fundamentals of all classes in the form of base classes.
Plug-in libraries get dynamically loaded to support a wide spectrum of codecs, container formats, input/output drivers and effects.

Since version 0.10, the plug-ins come grouped into three sets (named after the film The Good, the Bad and the Ugly),There's also a separate GStreamer FFmpeg plug-in, which is a FFmpeg-based plug-in that supports many media formats such as MPEG-1, MPEG-2, MPEG-4, H.261, H.263, H.264, RealVideo, MP3, WMV, FLV, etc.

```
$ sudo apt-get install libgstreamer0.10-0
gstreamer0.10-ffmpeg gstreamer0.10-alsa gstreamer0.10-plugins-base
gstreamer0.10-plugins-good
gstreamer0.10-plugins-bad
gstreamer0.10-plugins-ugly
```

Clint Eastwood, Lee Van Cleef, and Eli Wallach

Gstreamer – elements

There are different types of elements such as:

- source element: the source of stream
- sink element: the destination of stream
- filter: the manipulator of streams

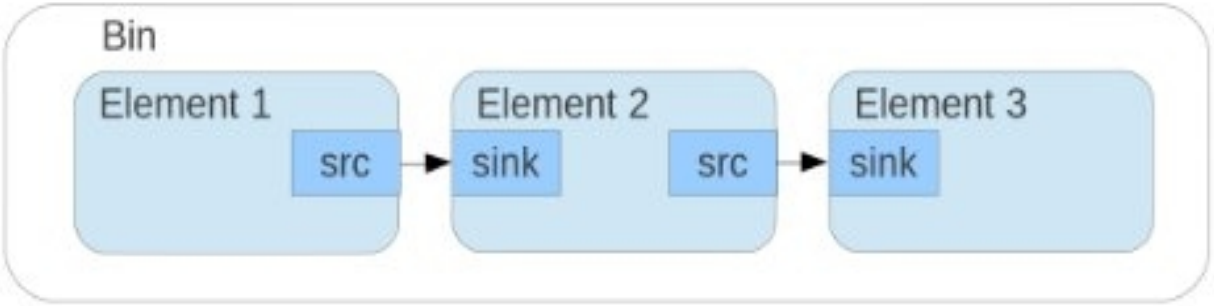
Gstreamer – elements

Each element can change state during the execution of application. The states are:

- null: no resource is allocated
- ready: resources are allocated, but the streams are not opened
- pause: streams are opened but no processed
- play: streams are being processed

Gstreamer – bin

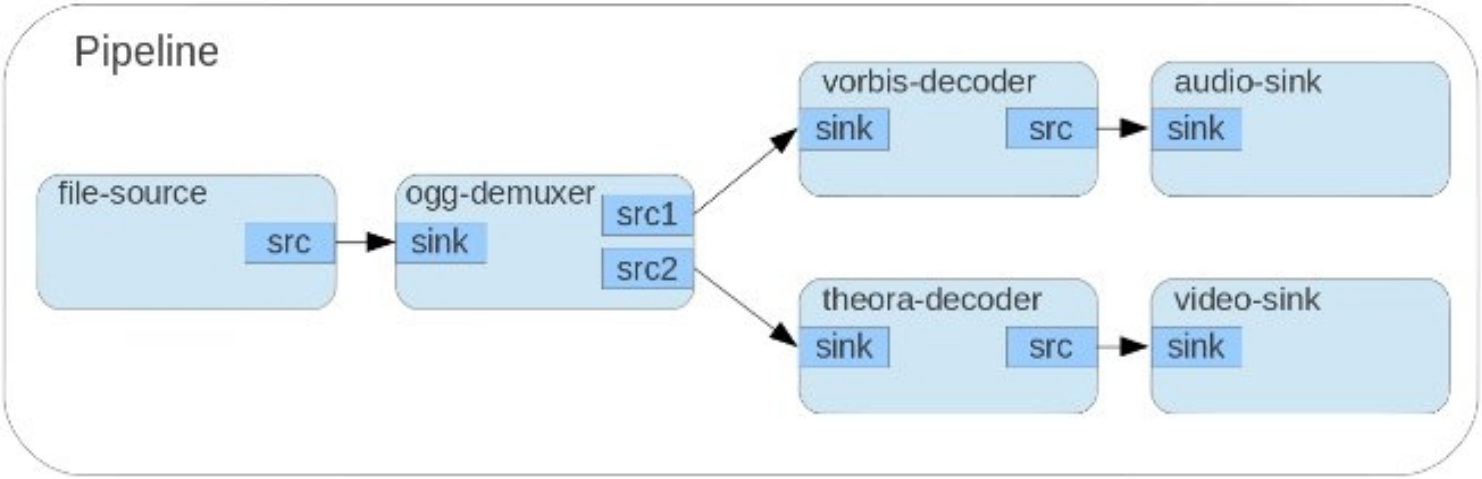
Bin is an element which contains other elements.
Since it is an element it can have the same states as mentioned for element.
By changing the state of a bin it automatically changes the states of its elements



Gstreamer – pipeline

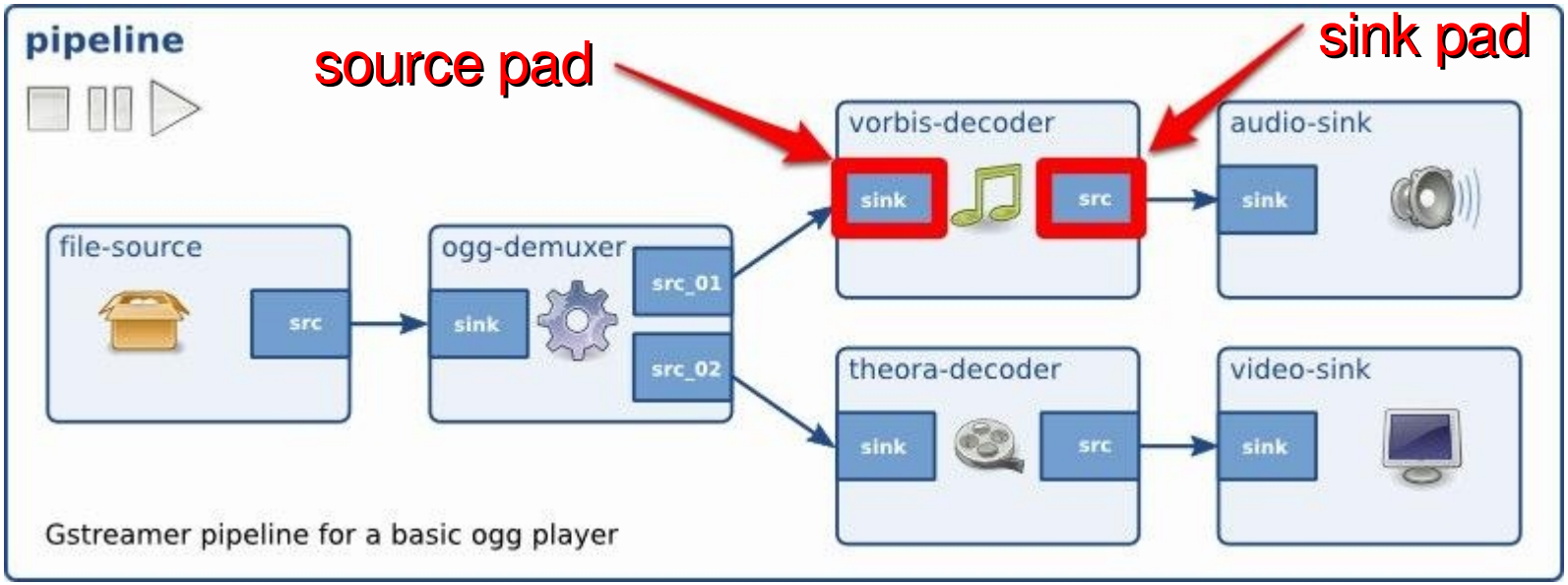
Pipeline is a top level Bin. Each application must have one pipeline.

The following figure demonstrates a pipeline for an ogg player



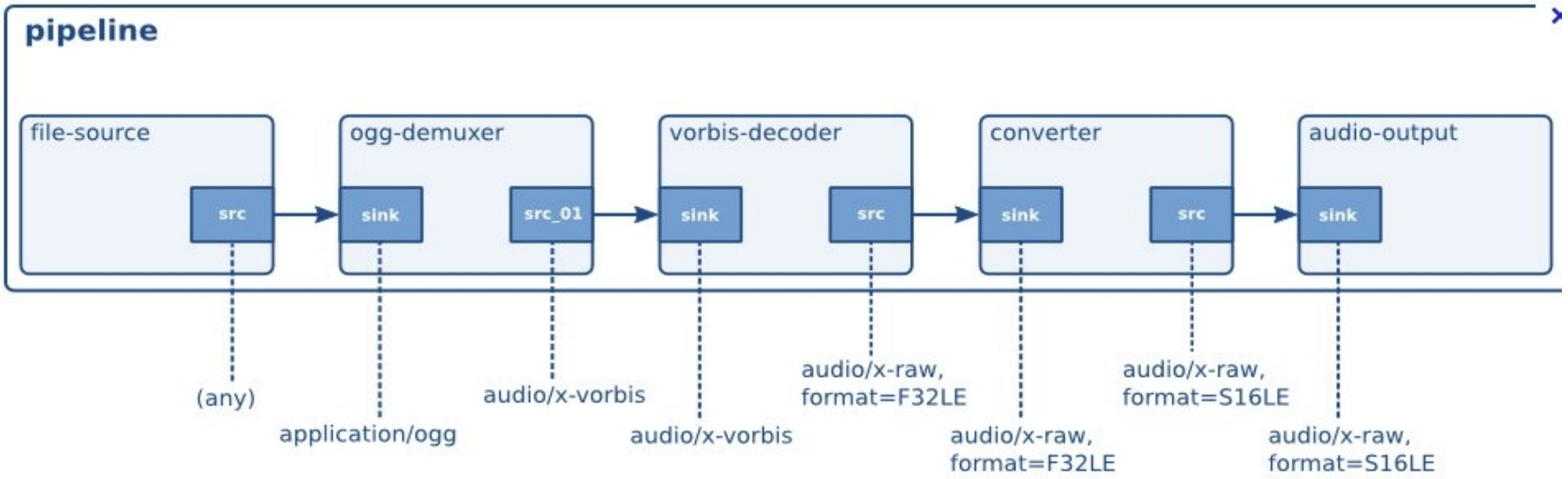
Gstreamer – pads

The pads are element's interface. Streams flow from an element's source pad to another element's sink pad. Pads can be built before execution or during runtime. Pads can have different capabilities, so before connecting two elements together we must make sure that two elements can so to speak talk together.



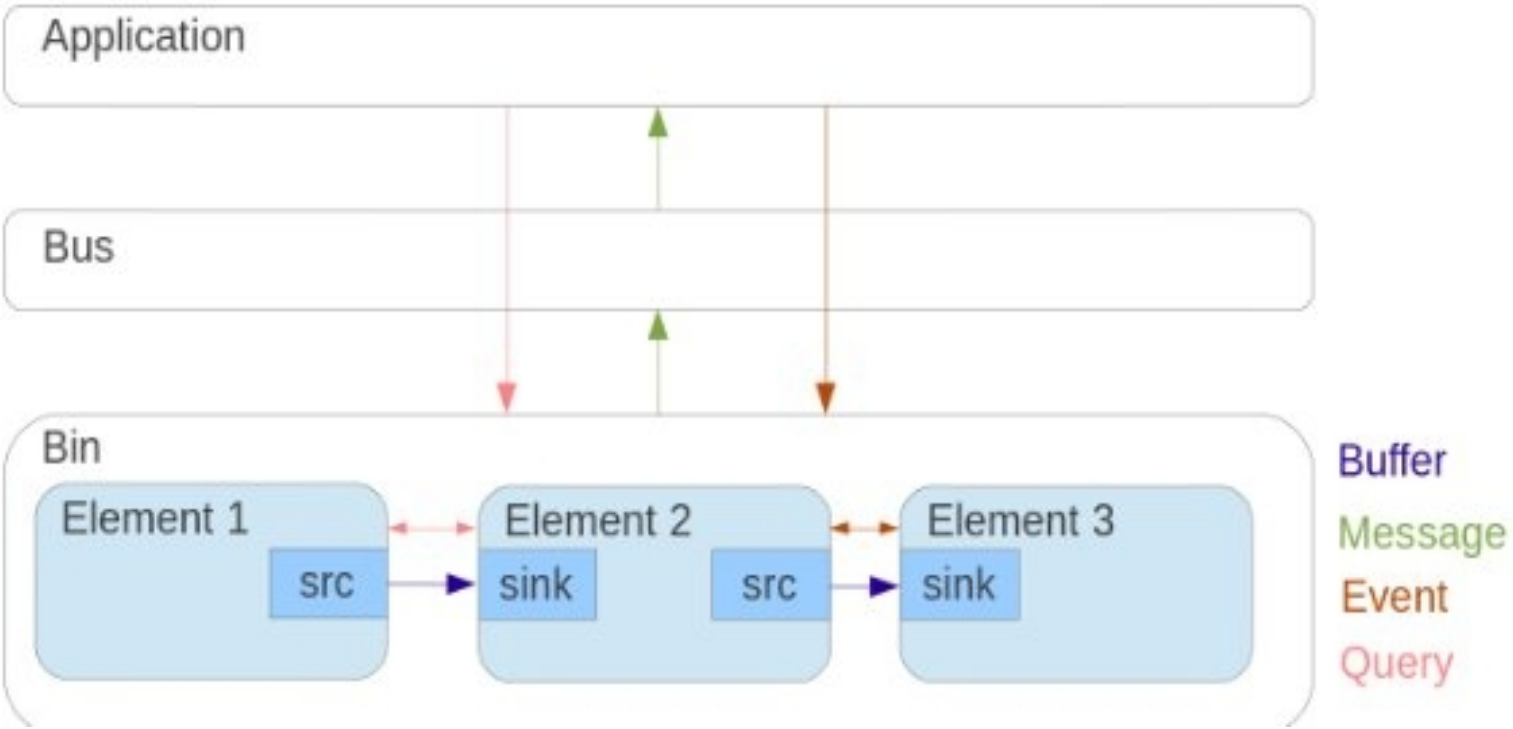
Gstreamer – autoplugging

The audio pipeline with the plugins - a simple media player for Ogg/Vorbis files:



Gstreamer – bus, buffers, ..

Bus : Each element can communicate with other elements via a bus. Bins usually transfer the bus messages of their children to applications.



Gstreamer – bus, buffers, ..

Buffers - They can pass streaming data between elements in the bin. They always travel downstream.

Events - Objects used to notify elements which are waiting for a particular event to occur. They can be sent from application and can travel downstream and upstream.

Message - Objects posted on pipeline bus to notify the application of any particular information such as bugs, state changes, end of streams, etc..

Query - Objects sent from application or between elements to ask a particular information such as duration, positions, etc.. In a bin they can travel downstream and upstream.

Using Gstreamer - launch

After the installation Gstreamer offers several commands :

`gst-launch` - is a simple script-like commandline application that can be used to build and test pipelines.

For example, the command :

```
gst-launch audiotestsrc ! audioconvert !
audio/x-raw,channels=2 ! alsasink
```

will run a pipeline which generates a sine-wave audio stream and plays it to your ALSA audio card.

Using Gstreamer - launch

gst-launch also allows the use of multiple threads.

You can use dots to imply padnames on elements, or even omit the padname to automatically select a pad.

Using all this, the pipeline (note the use of **queue** element):

```
gst-launch filesrc location=file.ogg ! oggdemux
name=d d. ! queue ! theoradec ! videoconvert !
xvimagesink d. ! queue ! vorbisdec !
audioconvert ! audioresample ! alsasink
```

will play an Ogg file containing a Theora video-stream and a Vorbis audio-stream.

Using Gstreamer - inspect

`gst-inspect` can be used to inspect all properties, signals, dynamic parameters and the object hierarchy of an element. This can be very useful to see which GObject properties or which signals (and using what arguments) an element supports.

```
bako@mezza:~$ gst-inspect-0.10 | grep mp3
ffmpeg:  ffmux_mp3: FFmpeg MPEG audio layer 3 formatter (not recommended, use id3v2mux instead)
ffmpeg:  ffdec_mp3on4float: FFmpeg MP3onMP4 decoder
ffmpeg:  ffdec_mp3on4: FFmpeg MP3onMP4 decoder
ffmpeg:  ffdec_mp3adufloat: FFmpeg ADU (Application Data Unit) MP3 (MPEG audio layer 3) decoder
ffmpeg:  ffdec_mp3adu: FFmpeg ADU (Application Data Unit) MP3 (MPEG audio layer 3) decoder
ffmpeg:  ffdec_mp3float: FFmpeg MP3 (MPEG audio layer 3) decoder
ffmpeg:  ffdec_mp3: FFmpeg MP3 (MPEG audio layer 3) decoder
flump3dec:  flump3dec: Fluendo MP3 Decoder (liboil build)
mpegaudioparse:  mp3parse: MPEG1 Audio Parser
typefindfunctions: audio/mpeg: mp3, mp2, mp1, mpga
typefindfunctions: application/x-apetag: mp3, ape, mpc, wv
typefindfunctions: application/x-id3v1: mp3, mp2, mp1, mpga, ogg, flac, tta
typefindfunctions: application/x-id3v2: mp3, mp2, mp1, mpga, ogg, flac, tta
mad:  mad: mad mp3 decoder
lame:  lame: L.A.M.E. mp3 encoder
lame:  lamemp3enc: L.A.M.E. mp3 encoder
```

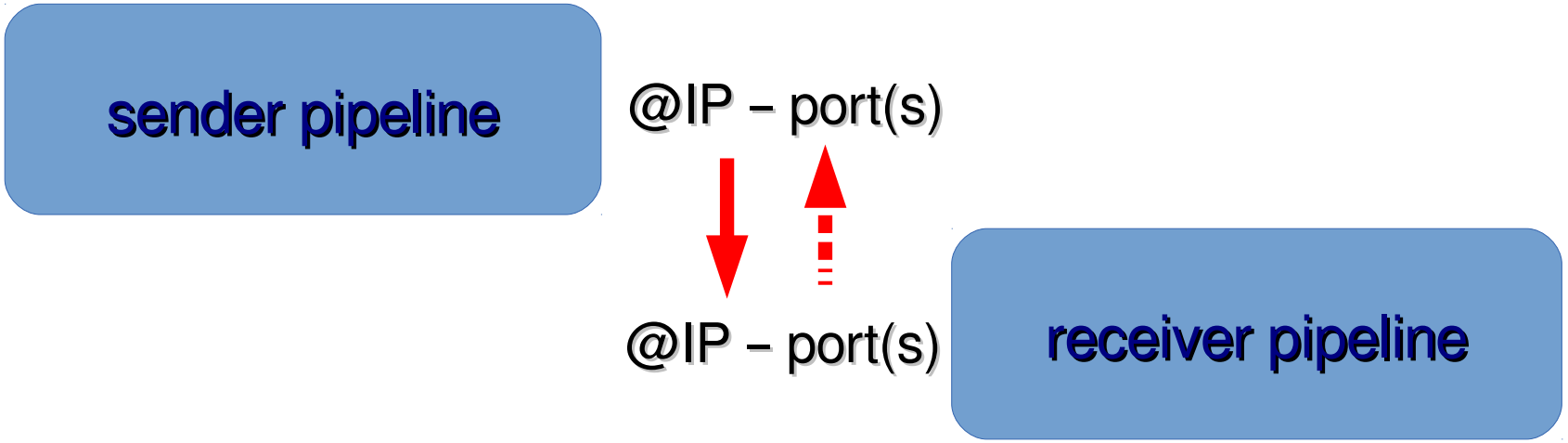
Using Gstreamer - inspect

```
bako@mezza:~$ gst-inspect-0.10 mad
Factory Details:
  Long name:    mad mp3 decoder
  Class:        Codec/Decoder/Audio
  Description:  Uses mad code to decode mp3 streams
  Author(s):    Wim Taymans <wim@fluendo.com>
  Rank:         secondary (128)

Plugin Details:
  Name:                 mad
  Description:          mp3 decoding based on the mad library
  Filename:             /usr/lib/x86_64-linux-gnu/gstreamer-0.10/libgstmad.so
  Version:              0.10.19
  License:              GPL
  Source module:        gst-plugins-ugly
  Source release date:  2012-02-20
  Binary package:       GStreamer Ugly Plugins (Ubuntu)
  Origin URL:           https://launchpad.net/distros/ubuntu/+source/gst-plugins-ugly0.10
```

Gstreamer – streaming

Gstreamer allows us to send the multimedia flows over several Internet protocols :

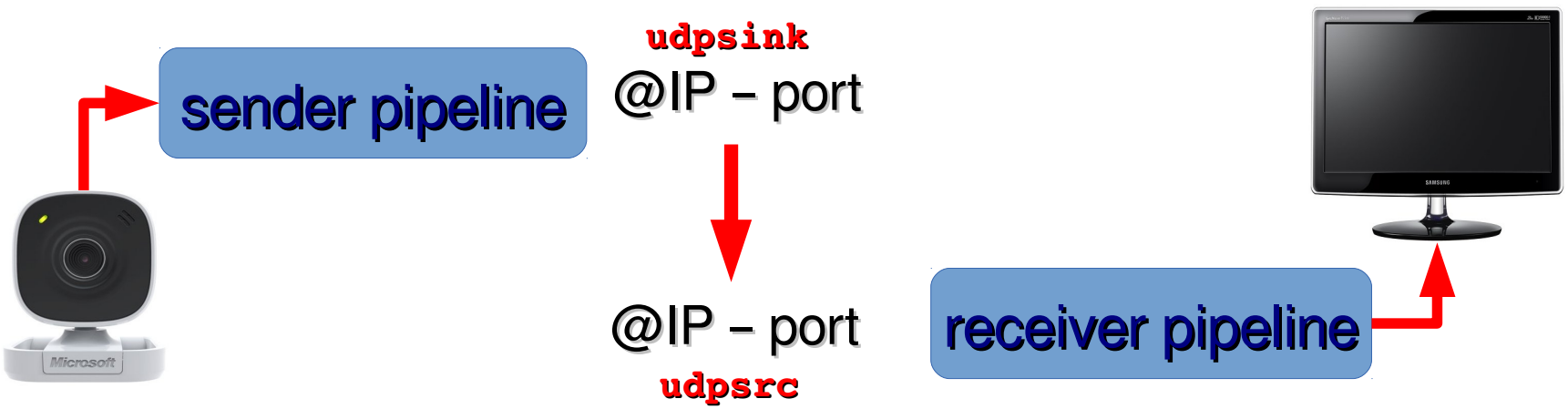
- directly over UDP, TCP as simple data flows or
- with RTP/UDP and RTP/RTCP/UDP as controlled multimedia flows



Gstreamer – streaming on UDP

```
gst-launch -v v4l2src ! 'video/x-raw-yuv, width=320,
height=240, framerate=30/1' ! queue ! videorate !
'video/x-raw-yuv,framerate=30/1' ! jpegenc quality=50 !
udpsink host=172.19.64.141 port=5000
```

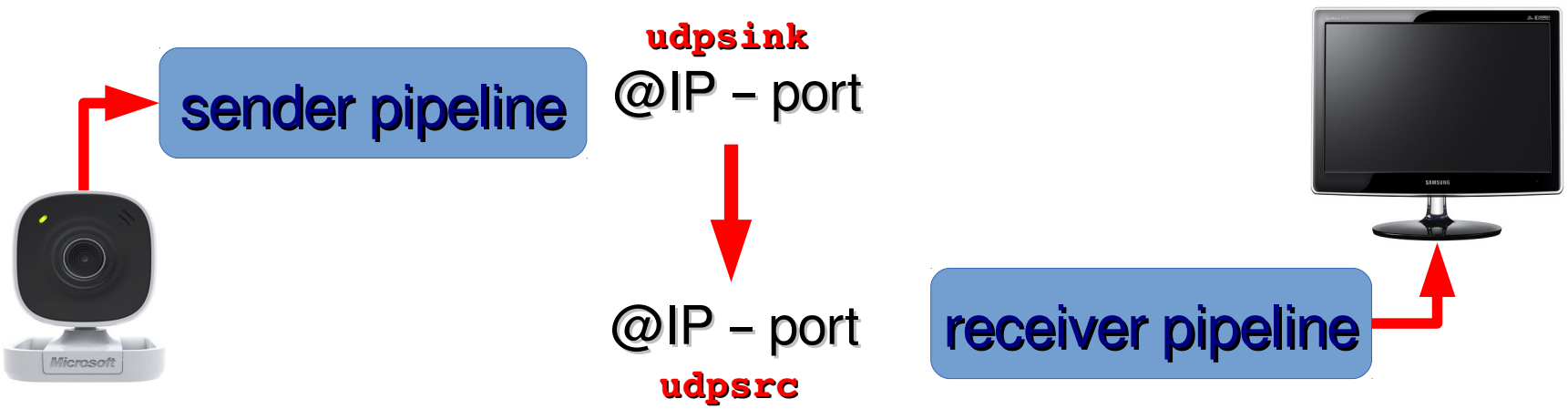
```
gst-launch udpsrc port=5000 ! jpegdec ! ffmpegcolorspace !
autovideosink
```



Gstreamer – streaming on UDP

```
gst-launch-1.0 -v v4l2src ! 'video/x-raw, width=320,
height=240, framerate=30/1' ! queue ! videorate !
'video/x-raw,framerate=30/1' ! jpegenc quality=50 ! \
udpsink host=172.19.64.141 port=5000
```

```
gst-launch-1.0 udpsrc port=5000 ! jpegdec ! videoconvert !
autovideosink
```



Gstreamer – streaming on TCP

```
#the client of tcpserver - to be launched first !
echo 'waiting for the connection from server'
gst-launch tcpserversrc host=172.19.64.141 port=5002 !
filesink location=Coffee.mp3
```

```
#sending data to the client on host=172.19.69.141 port=5002
gst-launch -v filesrc location=Coffee.mp3 ! tcpclientsink
host=172.19.64.141 port=5002
```

sender pipeline

tcpclientsink

@IP – port

@IP – port

tcpserversrc

receiver pipeline

Streaming on RTP/UDP

```
gst-launch -v alsasrc device=hw:1 ! audioconvert ! audioresample
! 'audio/x-raw-int,rate=16000,width=16,channels=1' ! speexenc !
rtpspeexpay ! udpsink host=172.19.64.141 port=50013

gst-launch udpsrc port=5001 caps="application/x-rtp,
media=(string)audio, clock-rate=(int)16000,
encoding-name=(string)SPEEX, encoding-params=(string)1,
payload=(int)110" ! gstrtpjitterbuffer name=jbuf latency=70
drop-on-latency=True ! rtpspeexdepay ! speexdec ! audioconvert !
audioresample ! volume volume=10 ! autoaudiosink
```

alsasrc
device=hw:1

sender pipeline

udpsink
@IP – port

payload

@IP – port
udpsrc

receiver pipeline

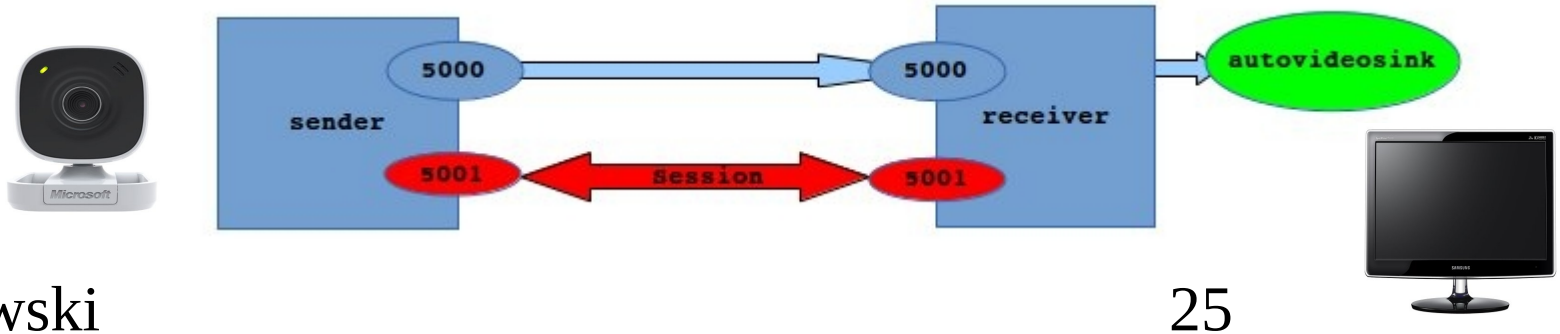
Streaming on RTP/TCP

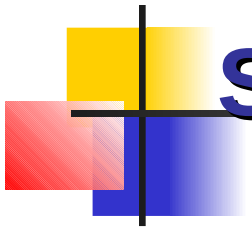
```
raspivid -t 0 -w 1280 -h 720 -fps 25 -b 2500000 -p 0,0,640,480
-o - | gst-launch -v fdsrc ! h264parse ! rtph264pay
config-interval=1 pt=96 ! gdppay ! tcpserversink
host=192.168.0.9 port 5000

gst-launch-1.0 -v tcpclientsrc host=192.168.0.9 port=5000  !
gdpdepay ! rtph264depay ! avdec_h264 ! videoconvert !
autovideosink sync=false
```

When using **gdp** with TCP, the sender can stream data using a **tcpserversink**, which can be received by multiple clients using **tcpclientsrc**. You are able to start the sender before the receiver.

Streaming with RTCP session

```
gst-launch -tv v4l2src ! videorate ! videoscale method=1 \
! video/x-raw-yuv,width=320,height=240 \
! jpegenc ! rtpjpegpay ! .send_rtp_sink gstrtpsession
name=session \
.send_rtp_src ! udpsink port=5000 host=172.19.64.141 \
session.send_rtcp_src ! udpsink port=5001 host=172.19.64.141

gst-launch -tv udpsrc port=5000 \
caps="application/x-rtp, media=(string)video,
clock-rate=(int)90000, encoding-name=(string)JPEG,
encoding-params=(string)1, payload=(int)26" !\
.recv_rtp_sink gstrtpsession name=session .recv_rtp_src \
! rtpjpegdepay ! jpegdec ! autovideosink \
udpsrc port=5001 caps="application/x-rtcp" ! session.recv_rtcp_sink
```





Summary

- Gstreamer - overview
- Gstreamer plugins
- Gstreamer: bins, pipelines ..
- Gstreamer inspect and launch
- Streaming on UDP
- Streaming on TCP
- Streaming with RTP/UDP and RTP/TCP
- Streaming session with RTP and RTCP