A GStreamer Tutorial

Centricular
Jan Schmidt
Email: jan@centricular.com
IRC: thaytan Twitter: @thaytan

Goals

- Provide you with a high-level understanding of GStreamer concepts
- Make you comfortable with using GStreamer tools, applications and plugins
- Show techniques for debugging GStreamer, plugins and apps
- Provide you with more understanding how to best use it
- Encourage you to help improve it ! :-)

gstreamer

Centricular

House Rules

- Don't just sit back
- Ask me questions! (at any time)
- Relate topics to situations you might have already encountered while working with GStreamer
- GStreamer is a massively vast code base and framework – it's easy to forget to mention something or make some connection – please help me fill any gaps

Tutorial Code

- https://github.com/thaytan/gst-tutorial-lca2018 (https://goo.gl/HvLFB1)
- GStreamer development packages:
  - dnf install gstreamer-tools gstreamer1-devel gstreamer1-plugins-* gstreamer1-libav gstreamer1-rtsp-server gstreamer1-rtsp-server-devel
  - apt-get install gstreamer1.0-tools libgstreamer1.0-dev gstreamer1.0-plugins-* gstreamer1.0-libav libgstrtspserver-1.0-0 libgstrtspserver-1.0-dev

Introduction

My credentials

Introduction

The depths of UTS
(but 2003)

What is GStreamer?

- ***An Open Source Pipeline-based Cross-Platform Extensible Multimedia Framework***
- Not:
  - A Media Player
  - A Codec or protocol library
  - A Transcoding Tool
  - A Streaming Server
- *But can be (and is) used to implement all that*

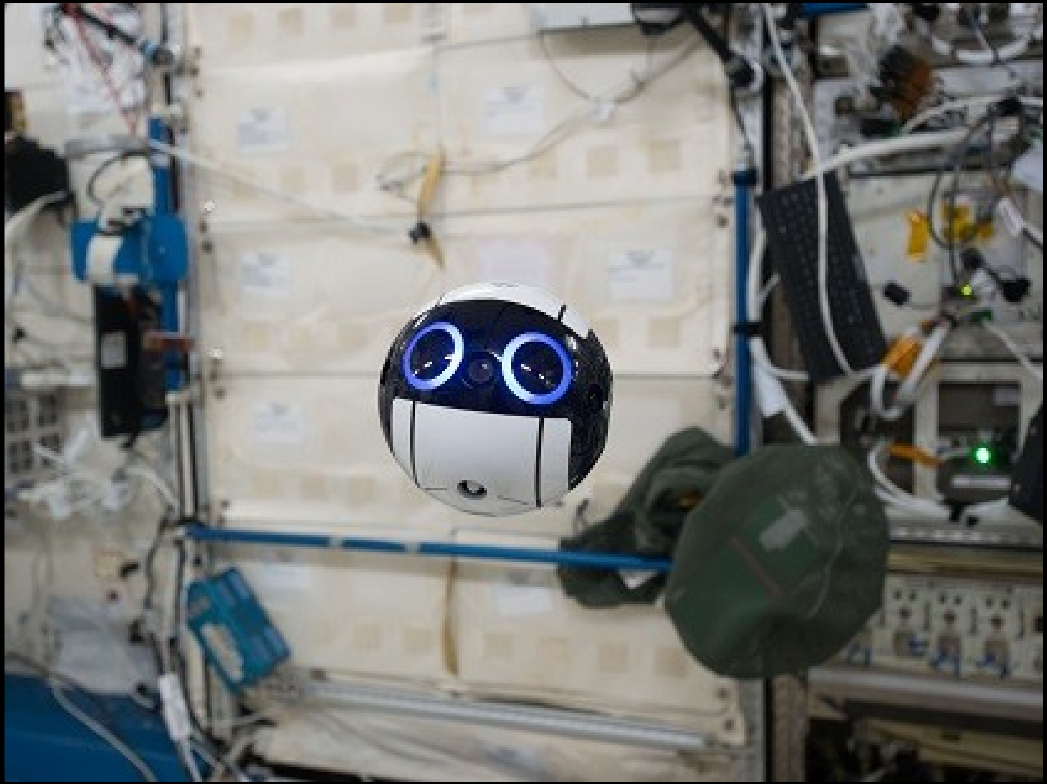
Overview

- Goals
  - Flexible and extensible design
  - Easy to integrate with other software (in both directions)
- Large, active developer and user community
- Ecosystem of companies providing services around GStreamer and companies building their own products and services on top of GStreamer

GStreamer Applications

- Media players
- Audio/video editors, music composers
- Voip/video communication
- Web browsers
- Transcoders
- Streaming servers and clients
- *… and many more*

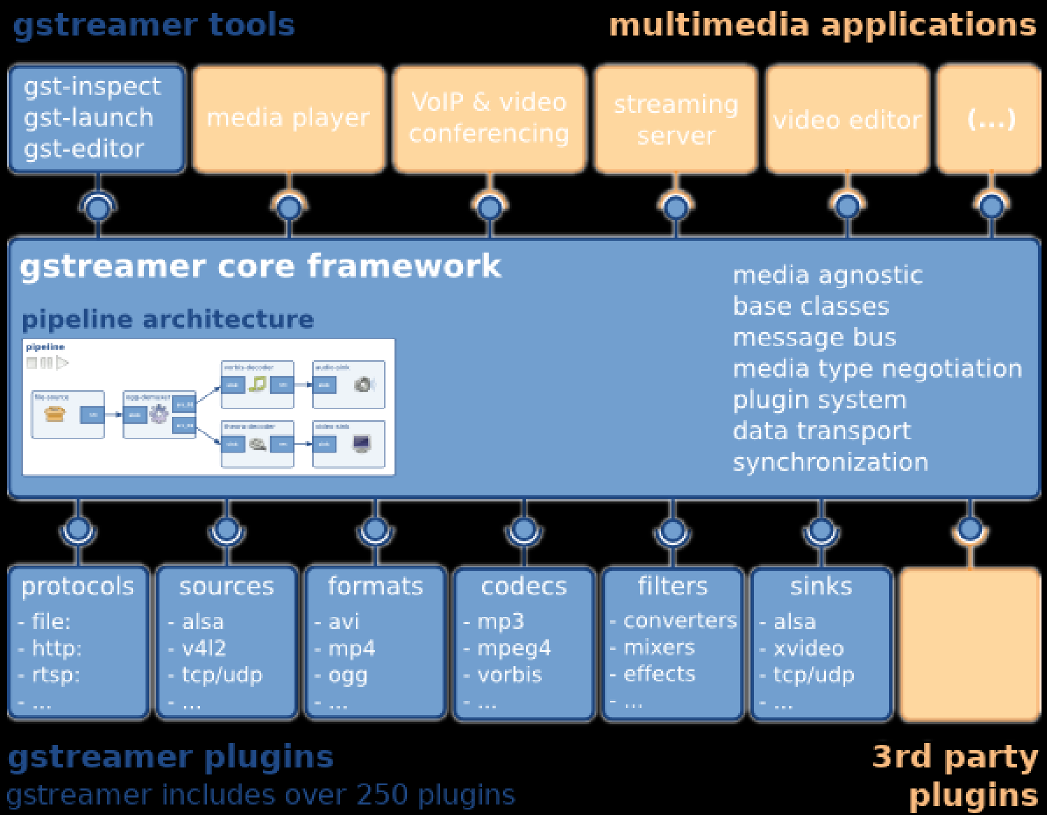
JAXA Int-Ball



GStreamer 0.10 vs. 1.x

- 0.10 no longer supported by the community
- All major software ported to 1.0
- No reason to develop new software with 0.10

Architecture



gstreamer

Centricular

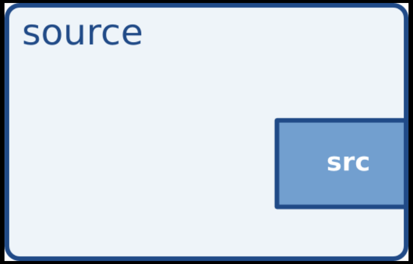
GStreamer Core

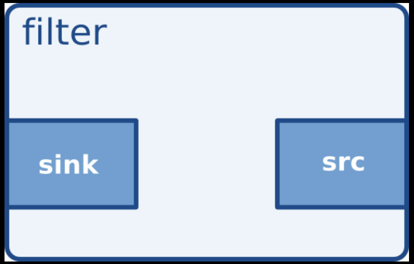
- Hierarchical pipelines
  - *Bins* containing *Elements*, linked by *Pads*
- Communication
  - *Buffers*, *Events*, *Queries*, *Messages*
- Format negotiation
- Scheduling and synchronization
- Plugin registry
- Media/format agnostic: does not know anything about audio/video/etc.

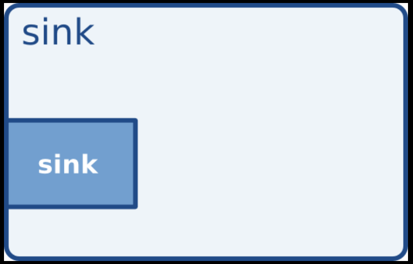
https://www.flickr.com/photos/23925401@N06/24342686634
[CC BY 2.0]

Pipeline Building Blocks – Elements

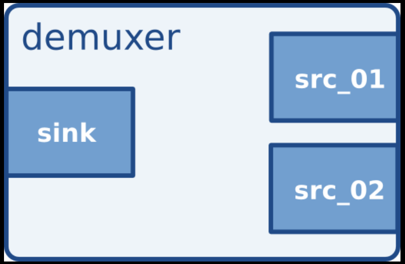
- With always Pads

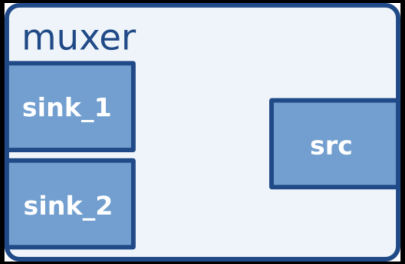


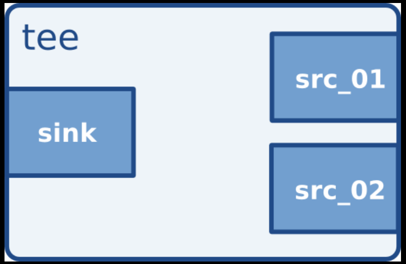




- or sometimes or request Pads







gstreamer

Centricular

Pipeline Building Blocks – Pads

- **Source** pads produce data
- **Sink** pads consume data

Caveat: terminology

- Beware of confusing terminology overlap
  - „Source“ is often abbreviated as „src“ in class names, enums or variable names
  - Depending on the context, „source“ or „src“ may refer to a *source element* or a *source pad*
  - Similarly, „sink“ may refer to a *sink element* or a *sink pad* depending on the context

Element States

- NULL: Deactivated, element occupies no resources
- READY: Check and allocate resources
- PAUSED: pre-roll, i.e. get a buffer to each sink
- PLAYING: active dataflow, running-time is increasing
- State changes always go through intermediate states, i.e. NULL-READY-PAUSED-PLAYING
  - GStreamer core handles that automatically
- Upward state changes can be asynchronous
- Downward state changes are always synchronous

Plugins & Registry

- Plugins provide additional features dynamically
  - Element factories to instantiate elements
  - Type-finders to detect the format of a file
  - Device monitors for listing available devices and capabilities
  - Contain introspectable information about features
  - e.g. new codecs, filters, …
- Per-user and system wide plugin paths
- Stored in a registry, cached on the file system
  - Applications can check available features without loading plugins by checking the Registry
- Plugins may be linked statically (e.g. Android)

GStreamer Tools – gst-inspect-1.0

- Prints details and features of a plugin or of a GstElement factory
- Uses the GStreamer registry
- Examples:
  - gst-inspect-1.0
  - gst-inspect-1.0 -a (good for grepping)
  - gst-inspect-1.0 /path/to/libgstcoreelements.so
  - gst-inspect-1.0 coreelements (plugin name)
  - gst-inspect-1.0 identity (element/feature name)

gstreamer

Centricular

GStreamer Tools – gst-inspect-1.0

```
$ gst-inspect-1.0 coreelements

Plugin Details:
  Name                    coreelements
  Description             GStreamer core elements
  Filename                /usr/lib64/gstreamer-1.0/libgstcoreelements.so
  Version                 1.12.4
  License                 LGPL
  Source module           gstreamer
  Source release date     2017-12-07
  Binary package          Fedora GStreamer package
  Origin URL              http://download.fedoraproject.org

  capsfilter: CapsFilter
  fakesrc: Fake Source
  fakesink: Fake Sink
```

GStreamer Tools – gst-inspect-1.0

```
$ gst-inspect-1.0 identity
Factory Details:
  Long name:   Identity
  Class:       Generic
  Description: Pass data without modification
  Author(s):   Erik Walthinsen <omega@cse.ogi.edu>
  Rank:        none (0)

Plugin Details:
  Name:        coreelements
  Description:     standard GStreamer elements
  Filename:        /usr/lib/x86_64-linux-gnu/gstreamer-1.0/libgstcoreelements.so
```

GStreamer Tools – gst-launch-1.0

- Provides a simple language to build pipelines and run them
- Mostly a debugging tool but can be very useful
  - Available as C API too for integration into applications
- Verbose (-v) parameter for more information

GStreamer Tools – Example 1

- Run a pipeline:
  - gst-launch-1.0 audiotestsrc ! audioconvert ! autoaudiosink
- This is audio, of course you can do the same with video
  - Check with gst-inspect-1.0 what the names of the corresponding video elements are and use them

Element Properties & Signals

- Elements can have
  - Properties, used by the application to modify the behaviour, mostly used for configuration
  - Signals, that the application can hook into to get notifications or to execute a function call on the element (action signals)
- Different for every type of element
- Introspectable at runtime
  - Names, types/signatures, descriptions
- Plugins provide no header files,
  GObject and GStreamer API is the only API there is !

Element Properties & Signals

```
$ gst-inspect-1.0 playbin
[...]
Element Properties:
  name                : The name of the object
                      flags: readable, writable
                      String. Default: "playbin0"
[...]
  connection-speed    : Network connection speed in kbps (0 = unknown)
                      flags: readable, writable
                      Unsigned Integer64. Range: 0 - 18446744073709551 Default: 0

  buffer-size         : Buffer size when buffering network streams
                      flags: readable, writable
                      Integer. Range: -1 - 2147483647 Default: -1
[...]
```

GStreamer Tools – Example 2

- Building on the pipeline from before:
  - gst-launch-1.0 audiotestsrc ! audioconvert ! autoaudiosink
  - gst-launch-1.0 videotestsrc ! videoconvert ! autovideosink
- You can set properties in gst-launch-1.0 too, e.g.
  - audiotestsrc prop=value ! ...
- Check in gst-inspect-1.0 what kind of properties the source elements have and set different ones
- What happens if you set unknown properties or values that are outside the valid range?
- Try doing audio and video in the same pipeline

Element Linking – Pads

- Elements can be linked on their Pads to define the dataflow
  - Must be compatible: opposite direction (src→sink) and compatible capabilities (*Caps*)
- Pads are created from *Pad Templates*
  - Containing name (template), direction, availability and all possible Caps of the Pads
  - Availability: always, sometimes or request Pads
  - Pad templates are what is shown in gst-inspect-1.0

Caps

- Define a media format, e.g.

```
video/x-h264, width=(int)1920, height=(int)1080,
   stream-format=(string)byte-stream, alignment=(string)au,
   level=(string)5,framerate=(fraction)25/1
```

- Terminology:
  - **video/x-h264** = **„media type“** (not MIME type)
  - **width** / **height** / **stream-format** / **alignment** / **level** = **„field“**
  - **int** / **string** / **fraction** / etc.: **„field type“** (often omitted)
  - media type + optional fields = a **„structure“** (GstStructure)

Caps

- made of one GstStructure („**simple caps**“):

  ```
  video/x-h264, width=...;
  ```

  or multiple GstStructures:

  ```
  video/x-h264, … ; video/mpeg, … ;
  ```

- Can be ***fixed*** or ***unfixed***, i.e. multiple structures or one of the fields has ranges or lists.

  ```
  video/x-h264; video/mpeg,mpegversion=[1,2]
  ```

Caps

- Special caps: **ANY** and **EMPTY** caps
- Generic set operations defined on them:
  - Intersect, is-subset, can-intersect
- Caps features for advanced use cases
  - `video/x-raw(memory:GstGLMemory), width=1920, height=1080`
  - Conjunction of additional constraints for the media type
- Pads negotiate a single fixed caps for data flow

gstreamer

Centricular

Caps

- Conventions:
  - Naming conventions for media types, caps features and fields (lower case letters, numbers, no spaces)
  - Types often omitted, GStreamer will try to guess the right type when converting a caps string into a caps object internally. Which mostly works, unless you do things like framerate=30 which would end up being framerate=(int)30 and not (fraction)30/1.
  - Sometimes fields have unexpected types, e.g. level=(string)5 (here level=2b is a possibility too)

Caps

```
$ gst-inspect-1.0 vorbisenc
[…]
Pad Templates:
  SRC template: 'src'
    Availability: Always
    Capabilities:
      audio/x-vorbis

  SINK template: 'sink'
    Availability: Always
    Capabilities:
      audio/x-raw
                 format: F32LE
                   rate: [ 1, 200000 ]
               channels: [ 1, 255 ]
                 layout: interleaved
[...]
```

GStreamer Tools – gst-typefind-1.0

- Uses the typefinders to detect the Caps of a file
- Examples:

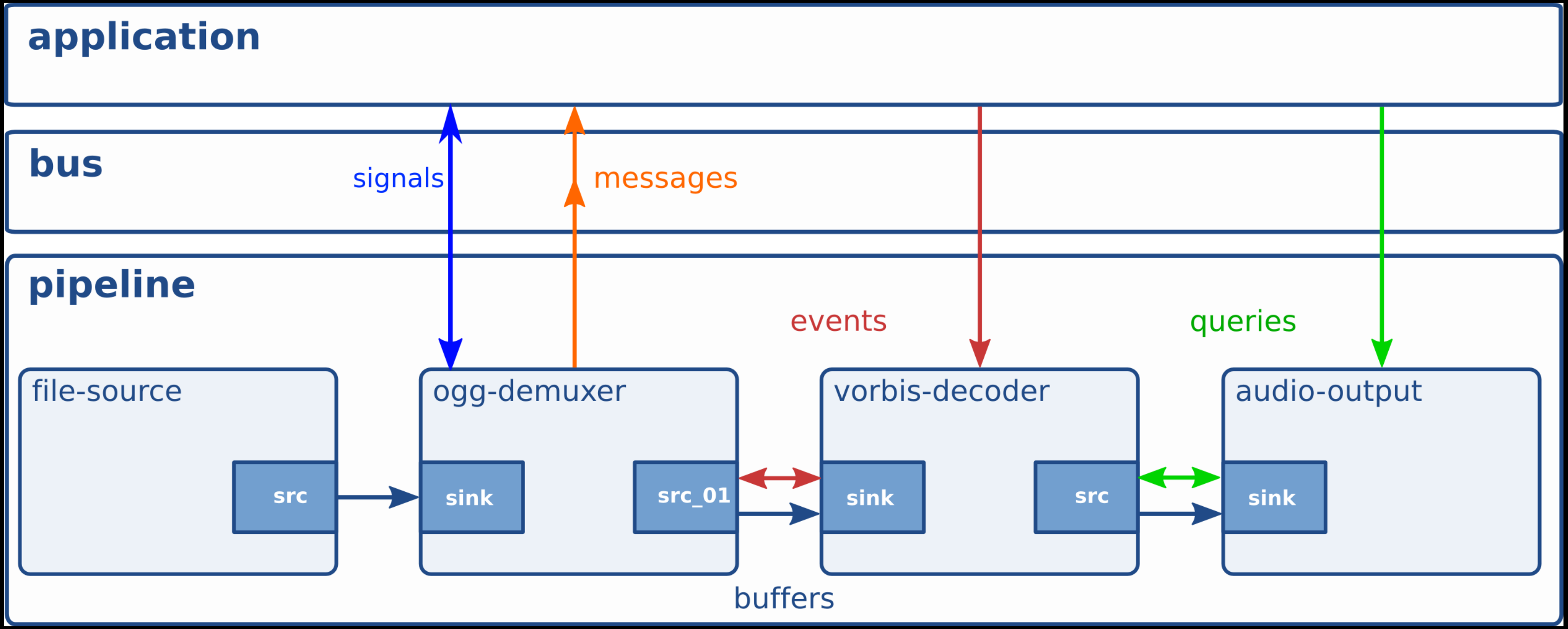
  -
    ```
    $ gst-typefind-1.0 test.mp3
    test.mp3 – application/x-id3
    ```

  -
    ```
    $ gst-typefind-1.0 test.avi
    test.avi – video/x-msvideo
    ```

GStreamer Tools – Example 3

- Let‘s go back to the video pipeline
  - gst-launch-1.0 videotestsrc ! videoconvert ! autovideosink
- This always shows a small 320x240 video
- Enforce a higher or smaller resolution by inserting a capsfilter after the source
  - You can just write caps between two ! or use the capsfilter element and set the caps property on it
  - What happens if you set incompatible caps? Try audio/x-raw instead of video/x-raw

Application <-> GStreamer Communication



GStreamer Tools – Example 4

- Add verbosity and message output to gst-launch output:
  - gst-launch-1.0 -vm audiotestsrc num-buffers=2 ! fakesink silent=false
- Can you find all the different messages and events in the output? What other parts do you find?

Threads

- GStreamer relies **heavily** on threads
- Data-flow and serialized events happen in streaming threads of pads
- Queues allow explicit insertion of new threads

Threads

- Generally need to add a queue before N-1 elements and after 1-N elements
- gst-launch example:

  -
    ```
    gst-launch-1.0 \
      audiotestsrc freq=440 volume=0.3 ! queue ! a. \
      audiotestsrc freq=880 volume=0.3 ! queue ! a. \
      adder name=a ! audioconvert ! autoaudiosink
    ```

Threads

- gst-launch example:
  - gst-launch-1.0 \

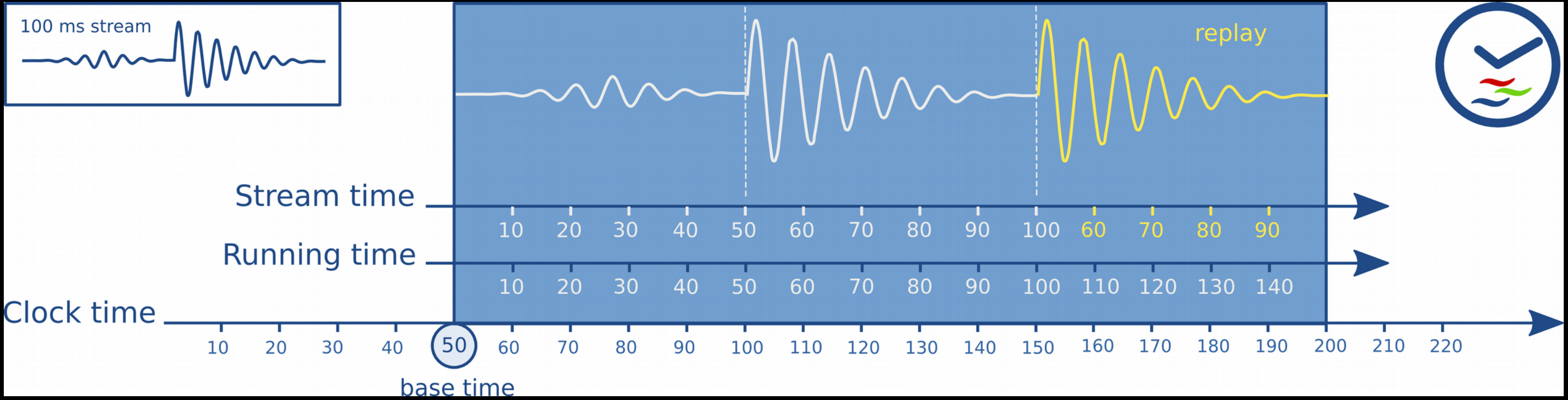
```
 filesrc location=test.ogg ! oggdemux name=d \
    d. ! queue ! vorbisdec ! audioconvert ! audioresample !
autoaudiosink \
    d. ! queue ! theoradec ! videoconvert ! videoscale !
autovideosink
```

Data-flow

- Data-flow can happen in pull- or push-mode
  - Demuxers generally prefer working in pull-mode but should be able to work in push mode too
  - Decided in READY->PAUSED during pad activation with scheduling query
- Element that starts a streaming thread and pushes data downstream is said to drive the (part of) the pipeline

Clocks & Synchronization

- Elements can provide a clock, pipeline selects
- Sinks and live sources synchronize to that clock
- Different kinds of time inside the pipeline



gstreamer

Centricular

Synchronization & Seeks

- Buffer timestamps mapped to running time with Segment, configured via *segment* event
  - start and stop position, buffers outside are dropped
  - base offsets and rate to shift and scale the timestamps
- *seek* Events instruct elements to jump to a different position (stream time) and configure a new segment
  - Different start/stop, rate can be adjusted
  - Flags to specify accuracy and speed of seeks
- **Sinks** render a Buffer when clock_time – base_time reaches the running time of the Buffer
- The **sync** property on **sink** elements controls whether they synchronise their output to the clock

Example 5

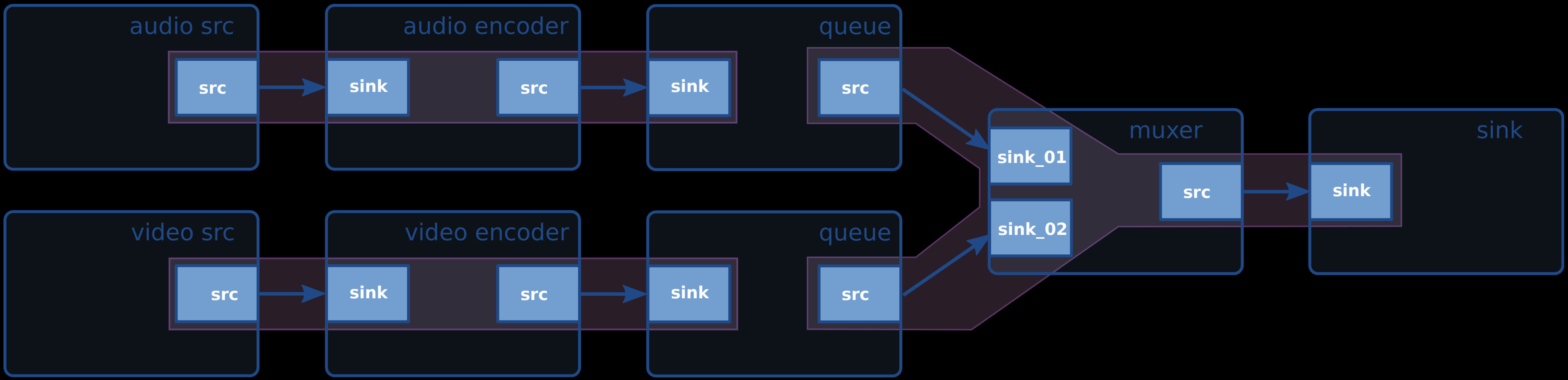
- gst-launch-1.0 \
  filesrc location=cooldance.ogg ! oggdemux name=d \
    d. ! queue ! vorbisdec ! audioconvert ! audioresample ! autoaudiosink \
    d. ! queue ! theoradec ! videoconvert ! videoscale ! autovideosink
- Now try adding 'sync=false' to the sinks

Example 6

- Playbin – easy playback
- gst-launch-1.0 playbin uri=https://gstreamer.freedesktop.org/media/incoming/Pixar%20-%20Geri\'s%20Game.avi
- gst-launch-1.0 playbin uri=file:///$PWD/big-buck-bunny_trailer-streamable.webm video-sink="glupload ! gleffects_sobel ! glimagesink"

Creating Content

- Capture/input -> encoders -> muxers -> output



gstreamer

Centricular

Synchronisation

- What keeps audio and video in sync?
- Audio and video devices likely provide timestamps on a clock that's not the pipeline clock
- Sources translate the timestamps
- Buffers from live sources are timestamped according to the running time of the pipeline using the pipeline clock
  - *Whatever clock that is*

Writing Apps

Documentation

- GLib
  - https://developer.gnome.org/glib/stable/
  - https://developer.gnome.org/gobject/stable/
- GStreamer
  - http://gstreamer.freedesktop.org/documentation/
    - Application Developer Manual
    - Core Reference
    - Core Libraries Reference
    - GStreamer Base Plugins Libraries Reference
    - Plugin Modules Reference (esp. gst-plugins-base)

Example 7

- Converting gst-launch-1.0 lines:
  - gst_parse_launch(“pipeline string”)
- gst_element_factory_make()
  - Create individual elements
- See the playback.c example in the repo

Fancy Clocks

- GStreamer includes implementations of remote clocks
  - GStreamer NetClock
  - NTP
  - PTP
- These can be used to synchronise to a master timeline
- Distributed recordings / capture
- Multi-room/device synchronised playback

Example 8

- Network synchronised playback using the GStreamer netclock
  - See the network-clocks/ sub-directory in the repo
- ./network-clocks/netclock-server
- on another machine:
- ./playback-sync -c $serverIP -p $port -b $basetime ../big-buck-bunny_trailer-streamable.webm

Clean Shutdown of Live Sources

- If you are capturing from a live source and recording to a file in a non-streaming format (i.e. one where headers need to be updated at the end and an index needs to be written), you can't just stop recording by setting the pipeline to NULL state
- The file would be closed, and the muxer won't be able to finalise headers and index

Clean Shutdown of Live Sources

- To shut down a live recording pipeline (or any other running pipeline before it 'naturally' EOSes), do this:
  - gst_element_send_event (pipeline, gst_event_new_eos())
  - This will inject an EOS event at the source(s), which will then make its way down the pipeline through the muxer, which will then finalise the file
  - When the EOS event reaches the sink, an EOS message will be posted on the bus
  - The application can pick the EOS message off the bus and knows it's safe to set the pipeline to NULL state now.
  - Examples: MP4, Matroska/WebM (to some extent)
- **gst-launch-1.0 does this when passed the -e argument**

Example 9

- Record something from your webcam:
- gst-launch-1.0 -e v4l2src ! videoconvert ! x264enc ! mp4mux ! filesink location=test.mp4
  - This may take some time to start, depending on which GStreamer version you have
  - Hit ctrl-c to end the recording

Network Transmission

- GStreamer has extensive support for network protocols
  - RTP
  - RTMP
  - HTTP
  - RTSP
- For playback, and for production

Example 10

- Send content over the network via RTP
- gst-launch-1.0 videotestsrc ! avenc_mpeg2video ! mpegvideoparse ! rtpmpvpay ! Udpsink
  - Generate a test pattern and send as MPEG2 RTP/UDP
- gst-launch-1.0 udpsrc caps="application/x-rtp,clock-rate=90000,payload=32" ! rtpjitterbuffer ! rtpmpvdepay ! decodebin ! autovideosink
  - Receive UDP and depayload, decode, display

RTSP Server

- GStreamer provides the gst-rtsp-server module
- Makes it easy to turn the output of a pipeline into an RTSP URL
- We also have (or WIP) HLS, DASH, WebRTC

Example 11

- See the test-rtsp-uri example in the repo
- ./test-rtsp-uri ./big-buck-bunny_trailer.webm
- In a player:
  - gst-play-1.0 rtsp://127.0.0.1:8554/test
  - (or on another machine, of course)

Device Specific

- gst-omx – OpenMAX integration
- Android / iOS capture/encode/decode
- Raspberry Pi
  - gst-rpicamsrc
- i.MX6
  - gstreamer-imx
- V4l2 video encoder/decoder APIs

Example 12

- From the gst-rtsp-server examples

```
./examples/test-uri "(rpicamsrc bitrate=750000 intra-refresh-type=both drc=medium ! \
 video/x-h264,profile=baseline,width=1280,height=720,framerate=10/1 ! \
 h264parse config-interval=1 ! rtph264pay name=pay0 pt=96 )"
```

Thank you!

Further questions?

Jan Schmidt
Email: jan@centricular.com
IRC: thaytan Twitter: @thaytan