# “decodebin3”
# Modern playback use-cases

GStreamer Conference 2015, Dublin
October 8th 2015

Edward Hervey a.k.a bilboed

edward@centricular.com

gstreamer

Centricular

- Playback ?
- Current implementation
- Pitfalls and limitations
- Some solution (no spoilers)

# What is playback ?

As a user
I want to play/pause/seek a media
Switch streams if available
....

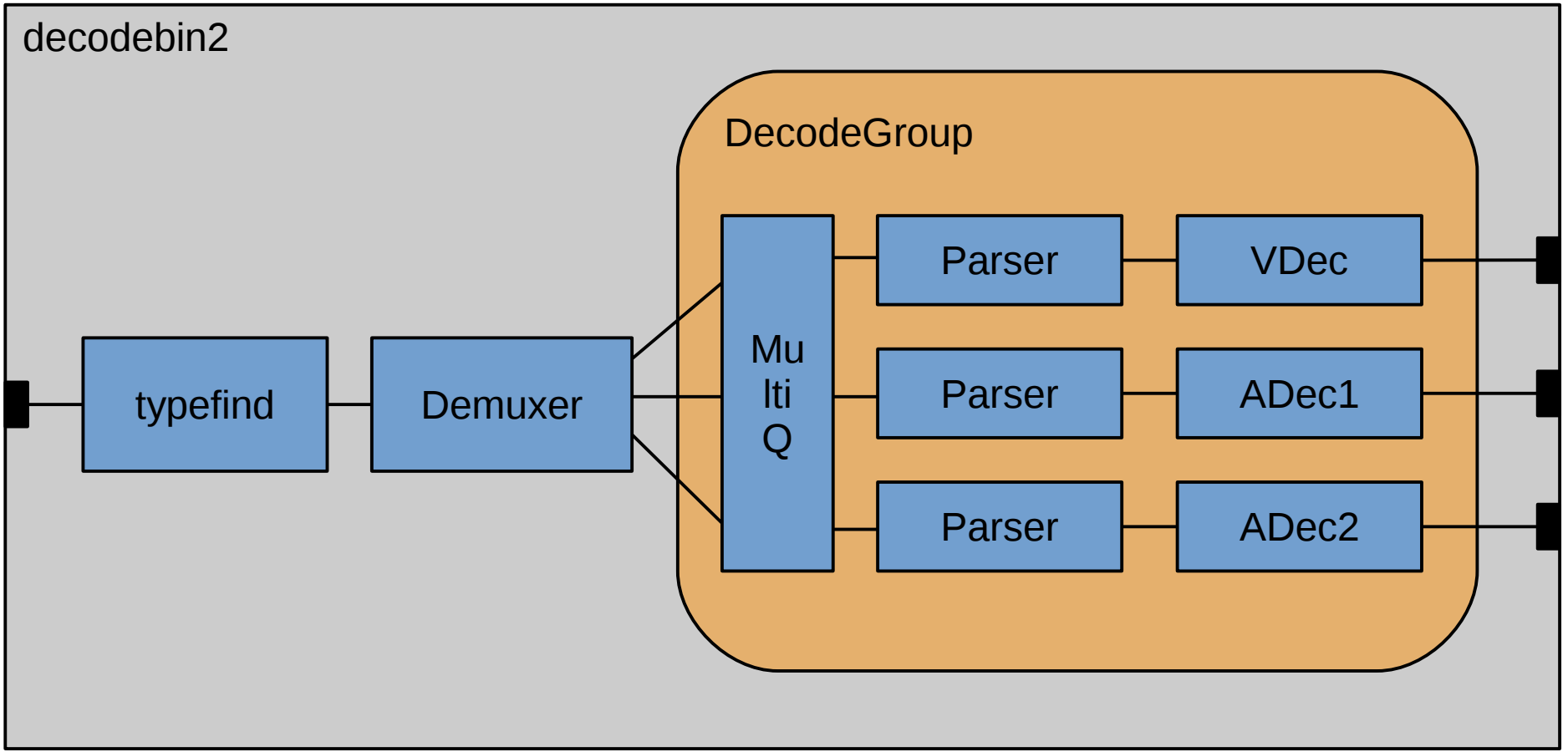
i.e. the gst-player API
*(go see slomo's talk just after)*

# Playback in GStreamer

- `playbin` (convenience pipeline)
  - `uridecodebin`
    - `decodebin`
  - `playsink`
  - And some non-reusable code

# decodebin

- Goal : “Take this input stream, figure out what elements are needed to decode it”
- Actually decodebin2 (2006, 0.10)
- Recursively figure out elements needed
- Support for hardware outputs (assisted auto-plugging)
- Stream switching without data loss
- “chained” files (ex: ogg)




gstreamer

Centricular

# Decodebin2: Stream Switching

We might switch streams later on …

We don't want the other stream to be drained completely

Else we will have a gap when switching

`multiqueue` throttles unlinked streams

***But … we're decoding everything :(***

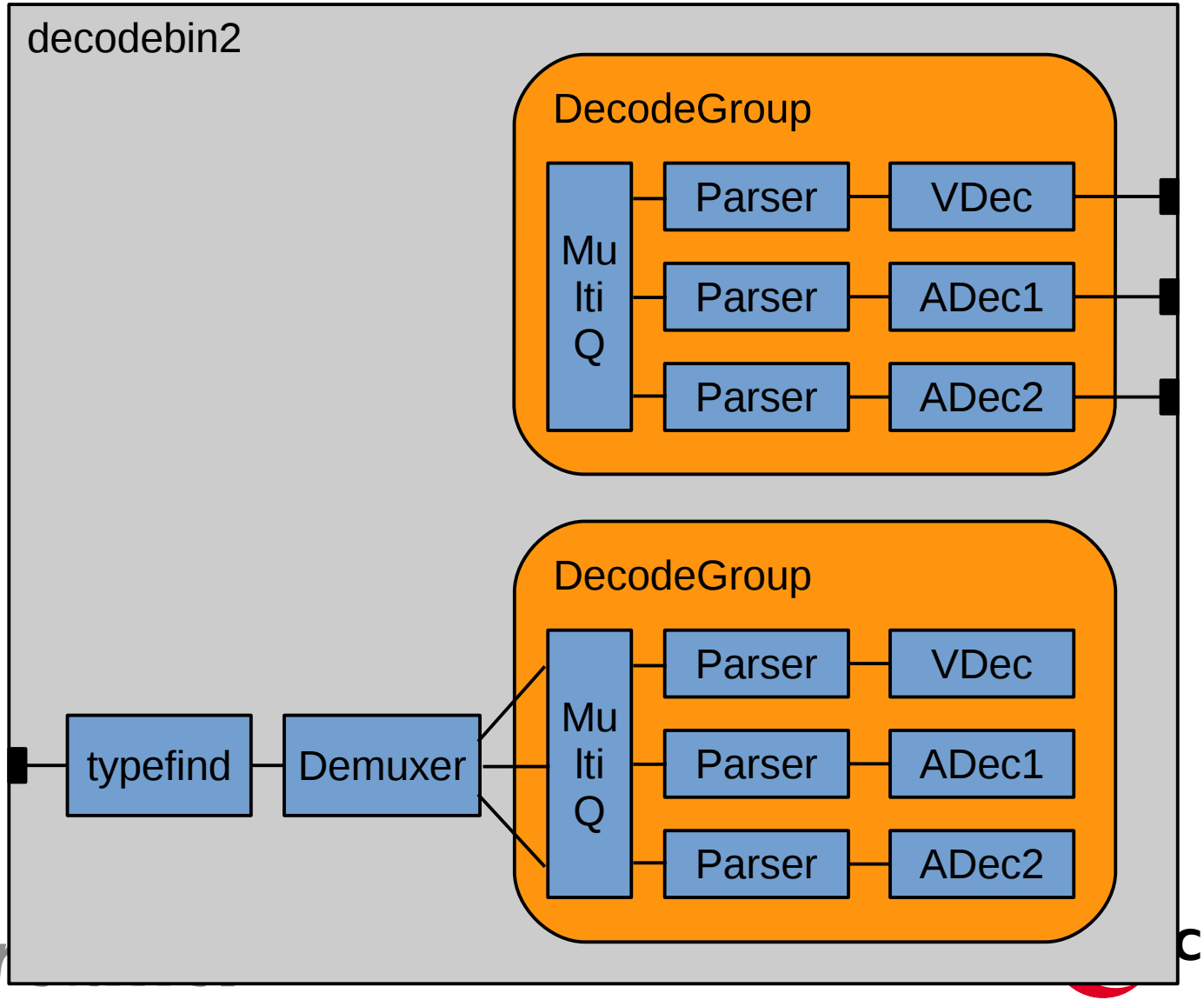
***Useless CPU/MEM/IO usage***

# Decodebin2: Chained File Support

- i.e. dealing with streaming formats (topology can change at any point in time)
- … but only with OGG in mind
- … and brings a bag of issues

# Decodebin2: Demuxer expectations

- To switch
  - Emit 'no-more-pads'
  - Add new pads
  - Send EOS on old pads
  - Remove old pads
- Decodebin2 will create a new DecodeGroup
  - Blocks new DecodeGroup
  - Waits until old DecodeGroup is drained (EOS reaching the end of that group)
  - Switches DecodeGroup over (ghostpads point to new group)


decodebin2
DecodeGroup
Mu
lti
Q
Parser
VDec
Parser
ADec1
Parser
ADec2
DecodeGroup
typefind
Demuxer
Mu
lti
Q
Parser
VDec
Parser
ADec1
Parser
ADec2

# Chain pitfalls

- New “pending” DecodeGroup
  - Increased memory usage (multiqueue)
  - Increased CPU usage (duplicated elements)
- Input and output of decodebin is no longer fully linked
  - Ex : seek event ending nowhere :(
- Want to just add/remove a stream ?
  - Still need to re-create a new bag of source pads
  - Breaks playback (switch video decoder in GOP)
  - Note: wasn't possible in 0.10 due to SEGMENT limitations

# Decodebin2 : more issues :(

- ~~Calculating~~ Guessing the ideal multiqueue size
  - Not too much (keep memory usage down)
  - But enough to cope with interleave
    - Distance in time between co-located buffers
- Hard to do because we're not dealing with timed input in multiqueue
  - Parsers (i.e. guaranteeing timed/chunked data) is after multiqueue

# uridecodebin

- Gets the proper source element to use for the URI
- If needed, add buffering/queueing (network streams)
  - Ideally this is where “network” buffering/queuing should happen
  - Not always the case (adaptive demuxers in decodebin)
- Plug result into decodebin
  - … or more than one decodebin (ex: RTSP)

# PlayBin

- Uses uridecodebin
  - Output of uridecodebin to inputselector(s) (to switch streams)
  - Into playsink
- Allows stream listing/selection
- Extra-uri (for subtitle in separate URI)
  - Creates a new parallel uridecodebin
- Gapless support (about-to-finish / next-uri)
  - Prerolls another uridecodebin and switches

# Pitfalls of Stream Selection

- Not a generic API
  - Need to duplicate that code/logic in your custom pipeline
- Expects all streams to switch from to be decoded (and switch happens in inputselector)
- How do we list streams ?
  - Entirely based on source pads from uridecodebin
  - What if there are no pads ? (ex: Alternate HLS/DASH)

# A solution to all these issues ?

- A generic API for listing streams...
  - Not tied to pads, Elements can list “hidden” streams
- … and to select streams :
  - Elements need to **know** what downstream wants
  - No longer relying on GST_FLOW_NOT_LINKED
- Reduce CPU, Memory and I/O usage to the minimum needed
- And let's use 1.0 improvements
  - Segment base (adding streams at any time)
  - Better/Saner renegotiation/reconfiguration

# Generic Stream API

- Listing Streams
  - Could just use stream-id … but not that useful for user
  - Also need type of stream, caps, tags, …
- We first need a more convenient way of dealing with “streams”
  - New GstStream high-level object

# GstStream object

- GstObject subclass (refcounted)
  - Id (same as STREAM_START stream-id)
  - GstStreamType
    - GST_STREAM_TYPE_{AUDIO, VIDEO, TEXT, …}
  - GstCaps
  - GstTagList
- It is just collecting information stored in various places
- In GST_EVENT_STREAM_START
  - Get all the information from one place

```
GstStream *gst_stream_new             (const gchar *stream_id,
                       GstCaps *caps,
                       GstStreamType type,
                       GstStreamFlags flags);

const gchar *gst_stream_get_stream_id (GstStream *stream);

void           gst_stream_set_stream_flags (GstStream *stream, GstStreamFlags flags);
GstStreamFlags gst_stream_get_stream_flags (GstStream *stream);

void          gst_stream_set_stream_type (GstStream *stream, GstStreamType
stream_type);
GstStreamType gst_stream_get_stream_type (GstStream *stream);

void        gst_stream_set_tags (GstStream *stream, GstTagList *tags);
GstTagList *gst_stream_get_tags (GstStream *stream);

void     gst_stream_set_caps (GstStream *stream, GstCaps *caps);
GstCaps *gst_stream_get_caps (GstStream *stream);
```

# GstStreamCollection

- A immutable collection of GstStream
  - Usually posted by demuxers or other elements that can offer a “collection” of streams
  - Does not need to have a GstPad associated
- GST_MESSAGE_STREAM_COLLECTION
  - User, application, bin, can be informed of available streams
- Not tied to playbin
  - Got a custom pipeline ? Win

# GstStreamCollection

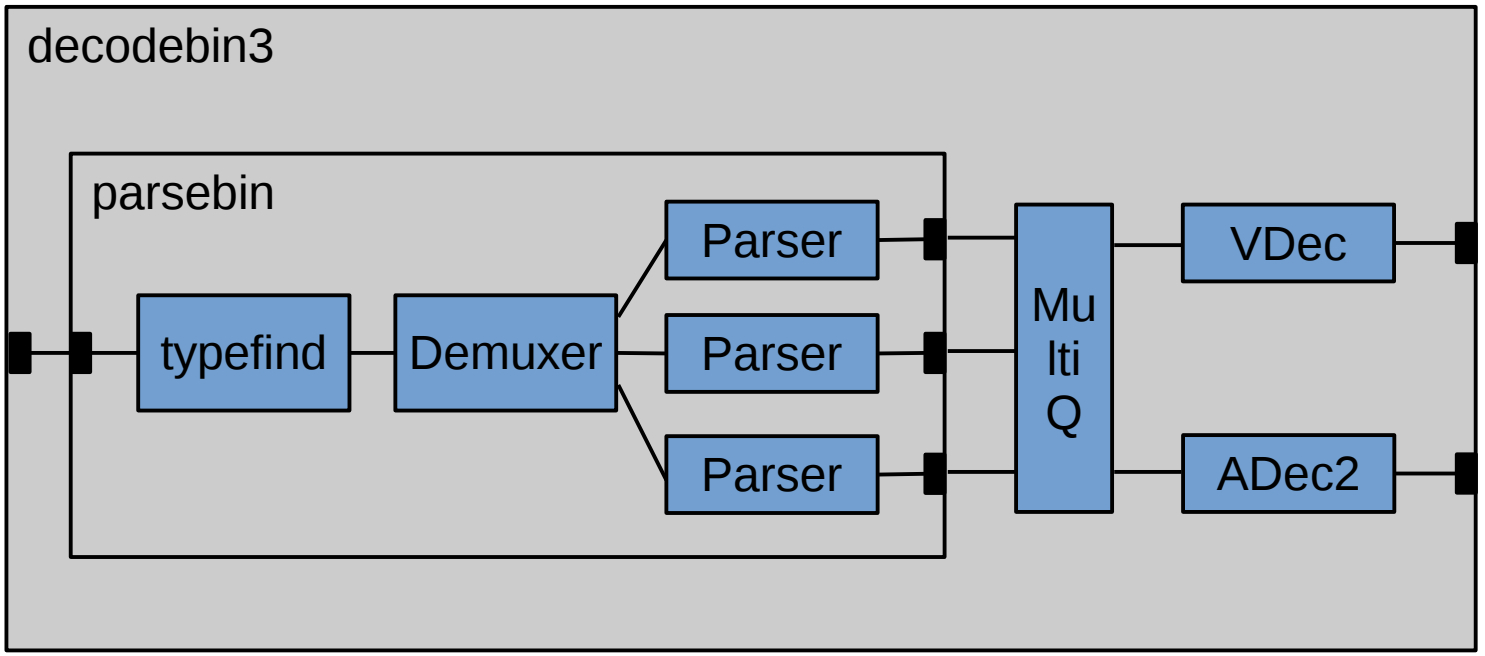
```
GstStreamCollection *gst_stream_collection_new (const gchar *upstream_id);

const gchar *gst_stream_collection_get_upstream_id (GstStreamCollection
*collection);

guint gst_stream_collection_get_size (GstStreamCollection *collection);
GstStream *gst_stream_collection_get_stream (GstStreamCollection *collection,
guint index);

gboolean gst_stream_collection_add_stream (GstStreamCollection *collection,
                        GstStream *stream);
```

# Selecting Streams

- GST_EVENT_SELECT_STREAMS
  - List of stream-id to be selected
- Elements can now reliably know **which** streams will be needed downstream
  - Avoid processing (decoding anyone ?)
  - Hidden streams to activate (Alternate HLS/DASH, switching DVB channel, ...)
- Not tied to playbin
  - Custom pipeline ? Win again

# decodebin3

- Because it's been 9 years since I committed decodebin2
- More seriously
  - Use the new stream API to reduce processing as much as possible
  - Re-use as many elements as possible
  - Reduce buffering
  - … and more




gstreamer

Centricular

# GstParseBin

- One single input sink pad
- Recursively figures out “decodable” elements needed
  - Demuxers, depayloaders, parsers … but not decoders
- No queueing
- Creates/Posts GstStreamCollection/GstStream if the element didn't create it
  - All pipelines can get new API support \o/
- Longer term : Reconfigurable
  - On input changes, re-use elements if possible, else switch

# decodebin3

- Output of GstParseBin(s) fed to multiqueue
  - Parsed Elementary Streams
  - Multiqueue “slots” (sink + src pad) are typed (audio, video,…)
  - If available slot of proper type, re-use, else create new one
- Selection is done post-multiqueue
  - Only thing that (might) need to be plugged is a decoder
  - By default only expose/decode one stream of each type
  - But you can also expose everything...
  - Or just what you selected via GST_EVENT_SELECT_STREAMS

# Selecting Streams (from upstream)

- Startup example
  - 3 streams from GstParseBin : video1, audio1, audio2
  - Do a pre-emptive decision of which streams to use
    - Pending selection : video1, audio1
  - Feed **all** streams through multiqueue
  - On multiqueue output, check STREAM_START / CAPS
    - If in Pending selection : create decoder and expose
    - Else leave unlinked

# Switching Streams

- GST_EVENT_SELECT_STREAMS
  - Ex : “video1”, “audio2”
  - Compare available, active, pending and requested streams.
  - Need to switch audio1 to audio2
  - Put audio2 in pending selection
  - Set idle probe on multiqueue output of audio1
    - Unlinks from decoder
    - Send GST_EVENT_RECONFIGURE to audio2 multiqueue source pad
    - Audio2 multiqueue source pad will use same logic as before

# Re-using decoders

- STREAM_START / CAPS on multiqueue source pad
  - Checks if in requested / pending selection
  - Check if there is an output available to be re-used
  - Check if decoder can accept CAPS
    - If so, just link
    - If not, unlink, insert new decoder, link
- No pad removed/added on decodebin3 output
  - Just a stream that reconfigures itself
  - Simpler usage in any pipeline

# Demuxer handling

- Decodebin3 backwards compatible with current behaviour
- Demuxers (ex: tsdemux) can be smarter
  - Ex: Video stream remains the same, but other streams change
  - No need to remove/add pad for the video stream
    - No breakage in data stream
- Any element can add/remove streams at any time
  - Ex: [CC] parsing in video parsers

# Decodebin3 multiple input

- A “media” might be composed of several different input streams
  - RTSP, Pro-cinema (separate files for audio/video), extra subtitle files, …
  - Instead of creating multiple decodebin, just use one
- Decodebin3 has GST_PAD_REQUEST sink pads
  - Feel all streams corresponding to one media
  - Commmon interleave (one multiqueue)
  - One GstStreamCollection (to rule them all)
- No longer dependent on playbin

# Gapless playback in decodebin3

- Can we re-use more elements ?
- Just re-use elements for gapless playback
- When EOS reaches all multiqueue slot input, emit 'about-to-finish'
  - Users have a chance to change the input of decodebin3
  - Earlier than with legacy playbin (EOS on output of decodebin)
- GstParseBin reconfigurable
  - Handles the reconfiguration
  - And we just get new streams… as if it was a demuxer update

# Playbin and uridecodebin

- Shifted most of the logic to decodebin3 (stream selection, gapless, multi-input support)
- Should not require new elements… but we'll see

# Summary

- Avoid as much unneeded processing as possible (CPU, MEM, I/O, …)
- Re-use as many elements as possible
- Still WIP

# Questions ?