GStreamer
-
Negotiate all the things

Edward Hervey – bilboed@bilboed.com

GStreamer design

- Since 1999 : Direct A-cyclic Graph of processing elements
- Hard part
  - Keeping that simple design ...
  - ... but with optimal underlying processing
- API added to provide information between elements to come up with the “best” choice
- Potentially plenty of choices

gstreamer

Choosing the right elements

- First things first
- How do we know what's the best element for:
  - A given task ?
  - Certain source caps ?
  - Certain sink caps ?
- Choice for application
- Decodebin and playbin use same info and logic

The Registry

- For each element, registry will provide:
  - Klass : What it can do
  - Rank : Ordering amongst similar elements
  - Pad Templates
    - Direction : sink vs source
    - Caps : what it could produce/consume
- No need to instantiate elements

Choosing the right elements

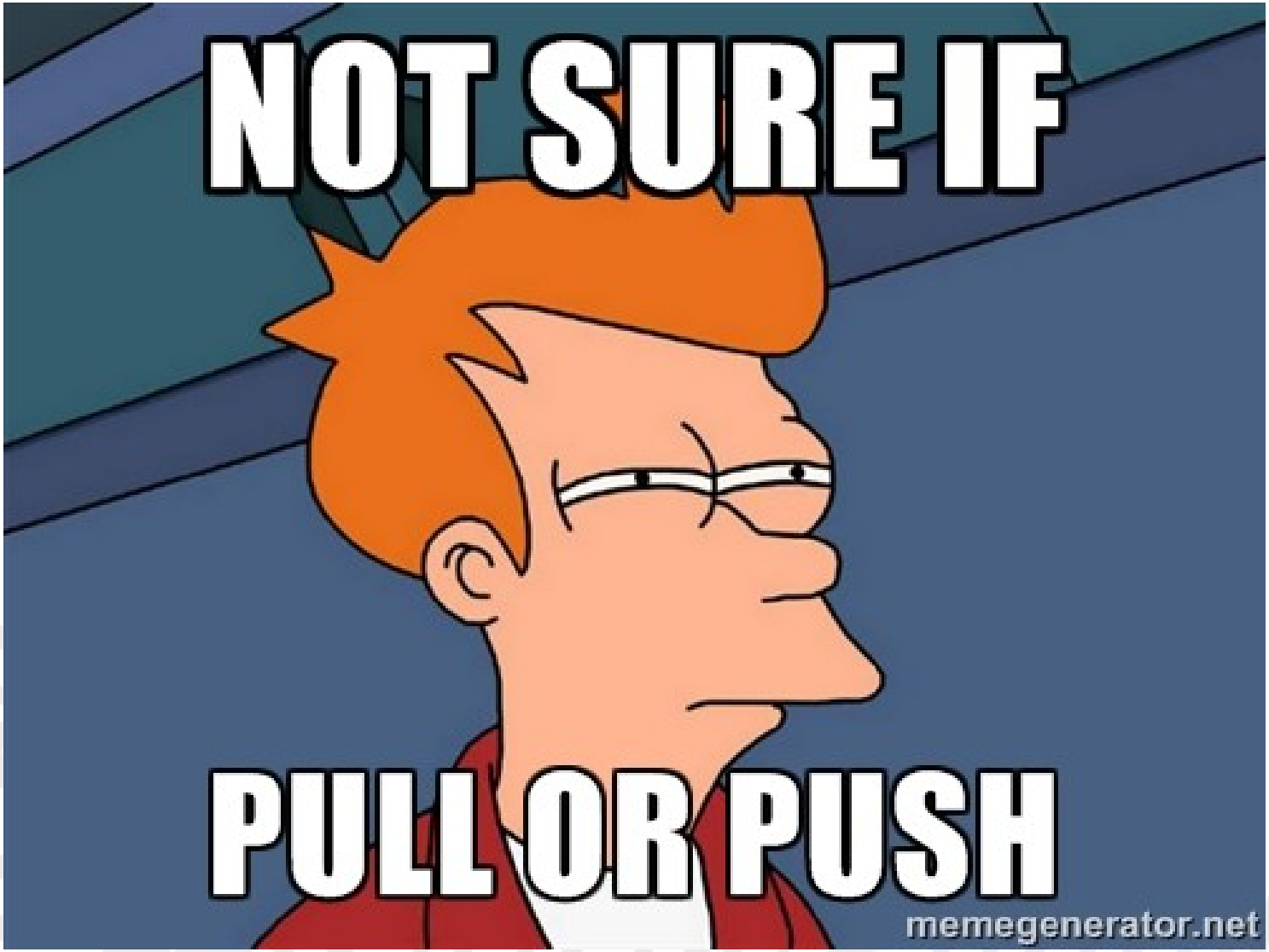
- Given certain caps, ask the registry for elements that could consume/output that
- Filter it by Klass (if needed)
- Sorted by rank
- Try elements in that order
  - !! GST_STATE_READY (initialize device/library)

Negotiating Scheduling

- In READY to PAUSED, (pads of) elements get “activated”
- Are we going to work:
  - Push-based (default, upstream pushes into downstream)
  - Pull-based (downstream does random access pull from upstream)

Negotiating Scheduling

- Some elements are more efficient in pull-mode (demuxers, know how much to pull from which offset)
- Some elements might be slow to change position if in pull mode (network source elements)
- Some elements might not provide data fast enough for real-time (network source elements)


NOT SURE IF
PULL OR PUSH
memegenerator.net

gstreamer

GST_QUERY_SCHEDULING

- Downstream sends query upstream
- Upstream fills in information
  - Mode: PUSH and/or PULL
  - Flags:
    - SEEKABLE
    - SEQUENTIAL
    - BANDWITH_LIMITED
- Downstream decides whether to configure itself in push or pull
- Want to force push-mode ? Queue !
- Pull-mode with a push-mode source ? Queue2 !

Negotiating CAPS

- Before any buffers are pushed, the Caps must be specified (GST_EVENT_CAPS)
- They must be fixed (no ambiguity, all fields with fixed values)
- There might be several possible caps to use
- How do we end up choosing the “best” fixed caps ? How do we get that information ?

gstreamer

GST_QUERY_CAPS

- `gst_query_new_caps(GstCaps* filter)`
- Ask what caps are supported
  - Optionally filtered by some other caps (source pad template caps)
- Purpose:
  - Provide Caps the element can support
  - Filtered by Pad Template Caps
  - Filtered by (optional) filter caps
  - Might be dependent on what downstream supports
- ***Downstream proposes***
- ***Upstream decides***

Choosing Caps

- GST_QUERY_CAPS returns:
  - GST_CAPS_NONE: no possible solution, negotiation fails
  - Fixed caps: you have no choice, use those.
  - Non-fixed caps: you need to choose from that.
- Order of non-fixed caps matters. Ex:
  - Video/x-raw,format=I420; video/x-raw,format=RGBA
  - “Downstream would prefer I420”

GST_QUERY_ACCEPT_CAPS

- `gst_query_new_accept_caps(GstCaps *caps)`
- Will these (fixed) caps be accepted on this pad ?
- Allows checking caps possibilities **without** pushing GST_EVENT_CAPS (which would be too late)

gstreamer

Use-case

- Fixed caps element: (decoders,demuxers), there is not choice to what they can output.
- Transformation: output caps dependent on input caps
  - Volume, encoders, videobox, ...
- Dynamic elements: output caps independent from input caps
  - Videoconvert, audioconvert, ...

Influence caps negotiation

- Capsfilter
- Refuses incompatible GST_EVENT_CAPS
- Filters results of GST_QUERY_CAPS
- Refuses incompatible GST_QUERY_ACCEPT_CAPS
- You can therefore influence what caps are negotiated on a given link

Re-using memory

- Elements can share a pool of (pre-allocated/pre-configured) buffers : GstBufferPool
  - Makes processing a lot more efficient
  - Can be using special memory they both support

GST_QUERY_ALLOCATION

- `gst_query_new_allocation(GstCaps *caps, gboolean need_pool);`
- Returns:
  - GstAllocator(s) supported
  - GstAllocationParams
  - GstBufferPool(s) (if requested)
  - GstMeta supported
  - ....

Allocation decision

- Upstream gets a potential list of pools, it can then pick one and configure it based on downstream information (AllocationParams) and upstream preferences (min/max buffers, alignment, size,..)
- Upstream knows what GstMeta/GstMemory downstream supports
  - GstVideoMeta, extra API
  - GstMemory, extra direct access to memory
- ***Downstream proposes***
- ***Upstream decides***

gstreamer

Delegating processing

- GstMeta
- Some processing could be avoided or even delegated
  - Cropping, compositing, audio mixing/level, ...
- But can't be expressed with caps, might change per buffer, ...
- Enter GstMeta
  - Provide transformationinformation per buffer
  - Let downstream elements handle it (more efficiently, or avoid the processing)

Delegating processing

- GST_QUERY_ALLOCATION provides GstMeta supported downstream
- Ex:
  - Videosink can do “cropping”
  - GstVideoCropMeta
  - Upstream decoder doesn't need to crop, just fills the meta on buffer
  - Videosink just uploads, doesn't crop, just tells hardware the cropping region
- ***Downstream proposes***
- ***Upstream decides***

gstreamer

But wait ...

- So I can use these optimized paths between elements (buffer pools, optimized memory, delegating processing ,...) ...
- ... but that's only negotiated once we've chosen elements (in PAUSED) ...
- ... what if there were **better** combination of elements (that could delegate everything) ?

gstreamer

gstreamer

And there's more !

- GST_QUERY_ALLOCATION will return supported allocator/pool/meta ... for the caps you already selected
- What if you could have picked another caps ... that provided better allocator/pool/meta ?

gstreamer

GstCapsFeatures

- New in 1.2 : GstCapsFeatures
- Provide additional description to caps
  - Memory:GstMemoryTypeName
  - Meta:GstMetaAPIName
- Stored in pad templates

GstCapsFeature usage

- Stored in the registry
  - Pick a better combination of elements
  - Sink supports GL ? Pick a GL decoder !
  - Decoder support dma-buf ? Pick a dma-buf aware sink !
- Used in Caps Negotiation
  - Glimagesink prefers memory:GLMemory, memory:EGLImage, meta:GstVideoGLTextureUploadMeta
  - Upstream elements can pick better/best combination (ex: RGBA, but accelerated)

gstreamer

Summary

- Various information exchanged
- As much as possible to make the “best” choice
- Only a walkthrough
- More details in API docs and design docs
- Any Questions ?