

Upipe video pipelines for multimedia transcoders, streamers and players

Flexible data flow framework
<http://www.upipe.org/>

What is Upipe?

- Handles flows of data in a « pipeline »
- Processes them using filters called « pipes »
- Designed to be the core of a multimedia player, transcoder or streamer
- Defines APIs:
 - To configure and feed data into pipes
 - To get out-of-band events from pipes
 - To store data in an efficient manner
 - To interact with an event loop/threads

Why yet-another-multimedia-framework?

- Existing frameworks are 15 years old; new trends emerged since:
 - Super-scalar architectures
 - Event-driven loops (à la libevent)
 - Frameworks (designed for multimedia players) are more and more used for professional applications, for which a single high-level API is not convenient
- Maintenance made more difficult by:
 - Lack of modularity, complexity
 - Confusion between processing vs. decision

Developing Upipe

- Started a year ago with new principles:
 - Specified bottom-up, from the simplest to the most complicated, different API levels are possible
 - All modules of code are autonomous and are unit-tested separately
 - **SIMPLICITY**
 - Emphasis on documentation
 - Designed for professional applications – not “media players that should play every single poorly encoded file”
 - Core under MIT, modules under GPLv2+ or LGPLv2+

Who we are

- Upipe development sponsored by OpenHeadend
 - Use Upipe in products (encoders, video processing, mosaics)
 - Provide development services on Upipe
 - Written by Christophe Massiot and Benjamin Cohen
- C. Massiot co-founded “VLC media player” 15 years ago, ran the television head-end of a large telco for 10 years

Upipe buffer management

- Relies on struct ubuf
- Is designed to point to a refcounted memory area (copy-on-write) with lock-less access
- APIs to get read or write pointer and unmap
- Two implementations:
 - Picture: handles the notion of planes, pixel/line prepend/append/alignment
 - Block: allows appending, inserting, deleting (zero-copy), prepend/append/alignment

Upipe reference management

- Ubufs aren't passed to pipes – urefs are
- A uref points to a ubuf, and associates a number of “attributes” with it
- Attributes are a triplet (name, type, value) and are standard (PTS...) or totally arbitrary
- Existing types are:
 - Booleans
 - Numeric (8bit or 64bit integers, rationals, double)
 - Strings, opaque

Pipes

- Pipes have at most one input and at most one output, and do the “processing” part
- Pipes have (possibly custom) control functions to change their settings, their output, or provide them with managers to create buffers
- All methods of struct upipe must be called from a single thread
- Demuxers and muxers are implemented with subpipes for each output (resp. input)
- Pipes libraries need no runtime dependancy

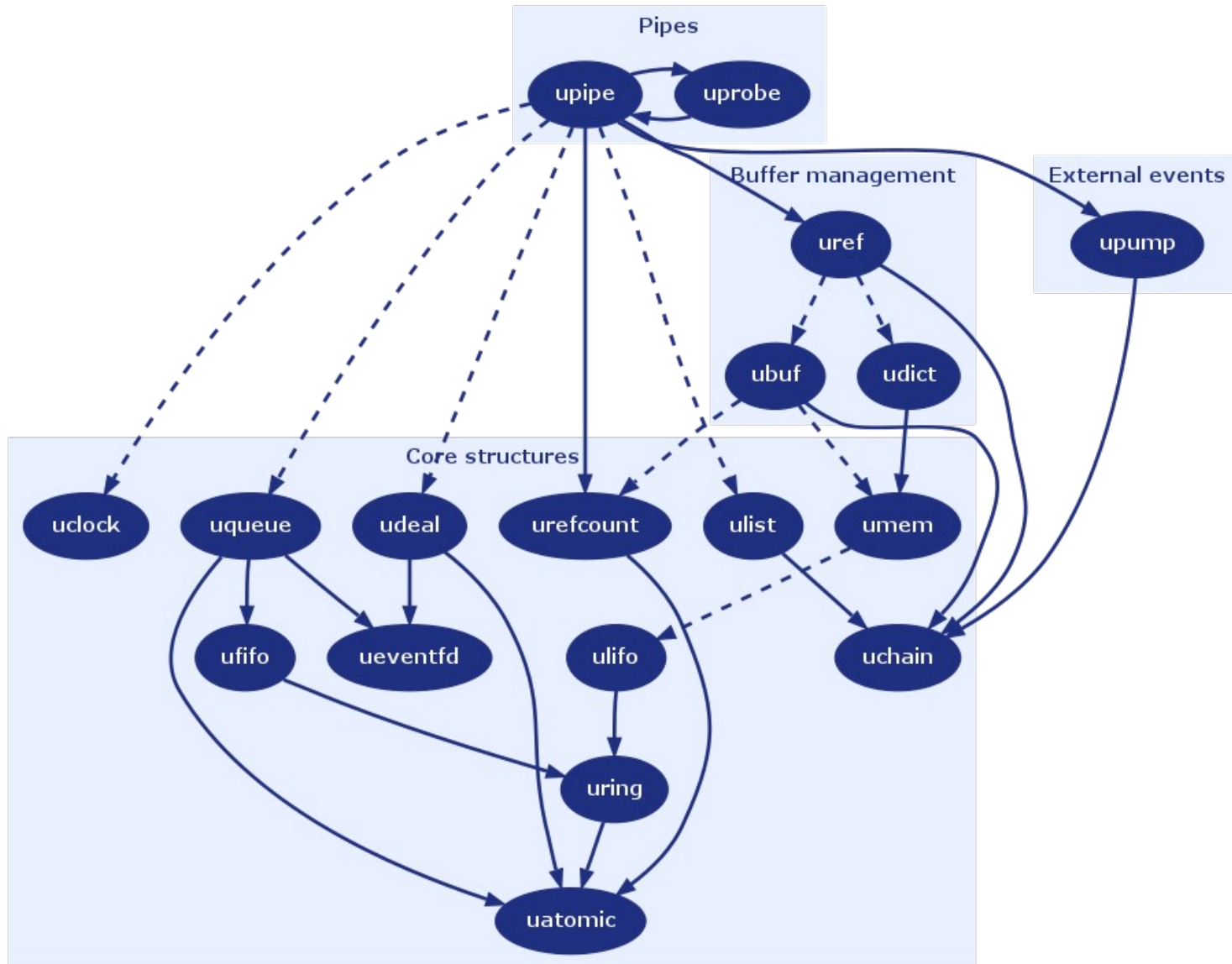
Probes

- Pipes that need to warn the application (or higher-level pipes) of something send an event to the “probe” that has been provided on allocation, or a hierarchy of probes
- Standard events `need_output`, `need_ubuf_mgr`, `need_uref_mgr` allow for a dynamic construction of the pipeline
- Probes are run in the same thread as the struct `upipe` methods, and do the “decision” part
- Custom events are possible

Event loop management

- Upipe does not rely on a specific event loop
- The upump API can map any event-based loop
- At present libev support is implemented, work in progress for ecore/evas
- Pipes create watchers on file descriptors, timers, and idlers and get called back
- Upipe provides `upipe_xfer_mgr` to transfer pipes from one thread to another

In a nutshell



Available modules for video pipelines

- Input/output: file, udp (multicast), http
- Demux: native TS demux + libavformat support
- Codecs: libavcodec & x264 support
- Filters: swscale, native deinterlacing
- Display: GLX, Mac OS X audioqueue
- Other utility modules:
 - Lock-less queue between threads
 - “dup” pipe
 - “null” pipe
 - Trick play, dejitter

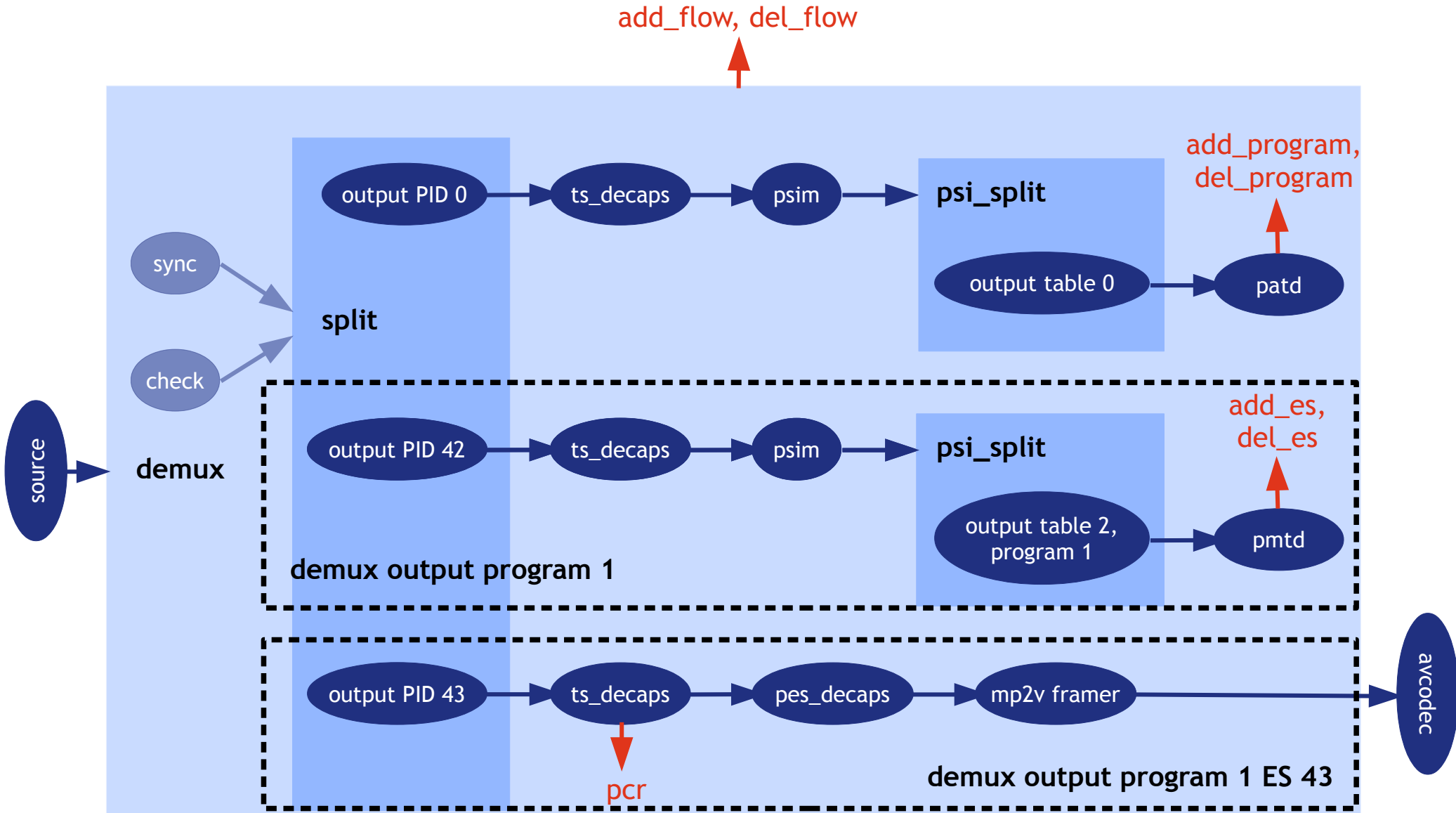
Status and development progress

- Upipe 0.1 just released, to be considered as a “preview release”
- An example “glxplay” application is provided as a tutorial
- The API may still evolve a bit, in particular:
 - Sound support (planar?)
 - Negotiation of ubuf formats within a pipeline
 - Thread pools (harmless for pipes)
 - Higher level APIs (“bin”) such as “demuxbin”, “decodebin”, “playbin”

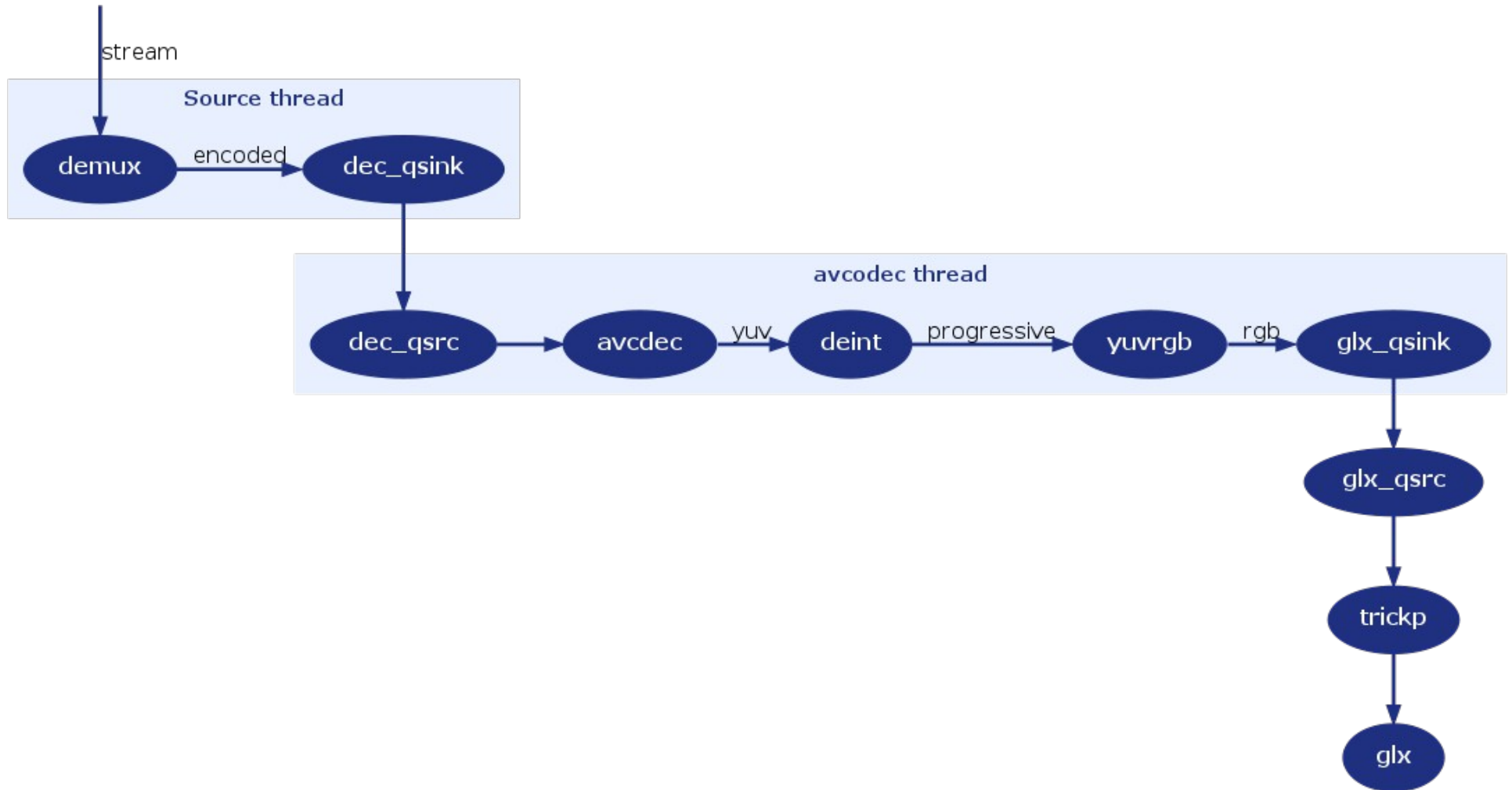
Potential applications

- Transcoders
- Multiplexers
- Play-out systems
- Mosaics
- Embedded, lightweight, media players
- Embedded demonstration platforms

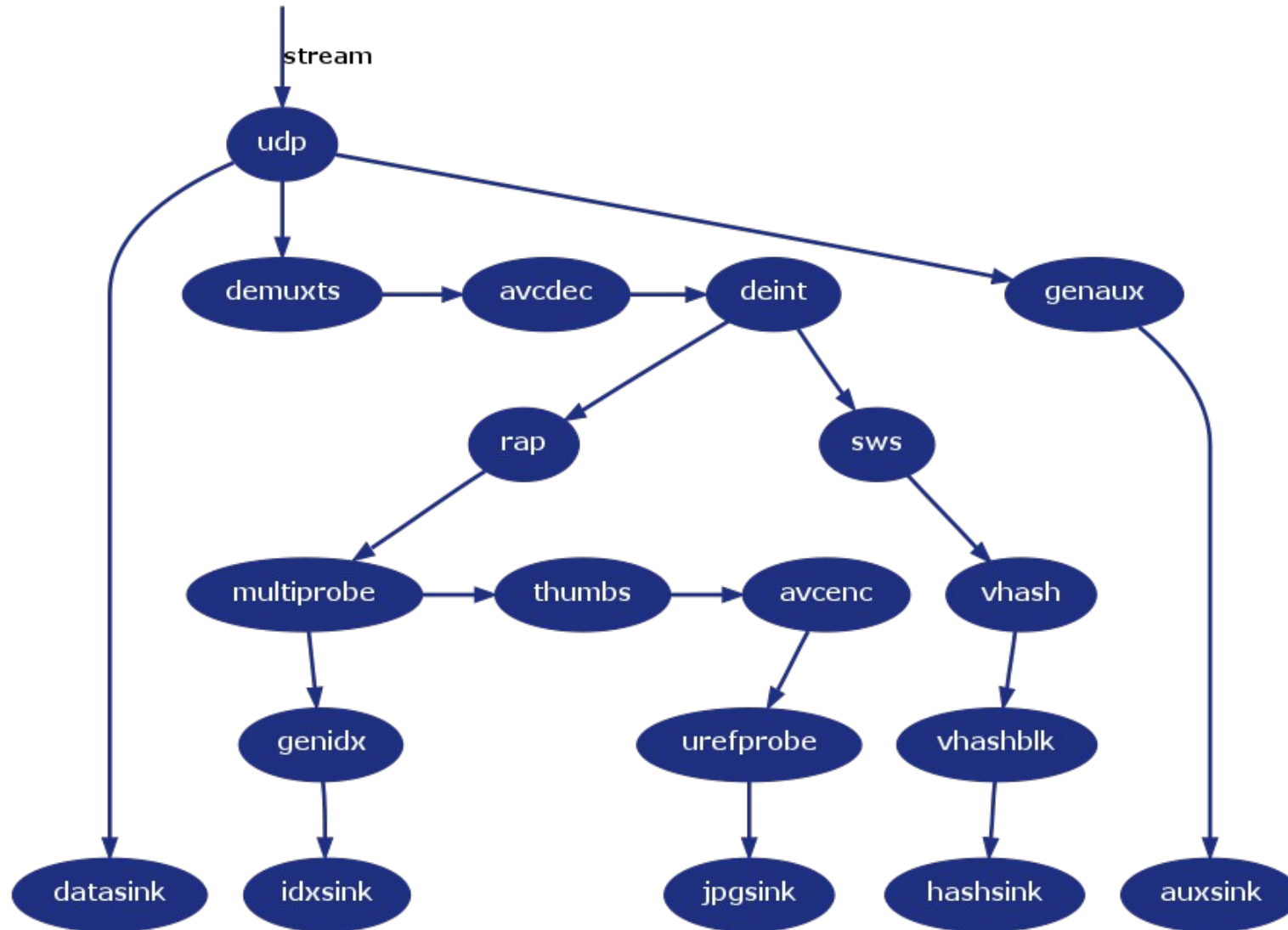
Case study: anatomy of the TS demux



Example: glxplay pipeline



Example: urecordhash pipeline



Keep in touch!



<http://upipe.org/>