Introducing Upipe

Flexible data flow framework

http://www.upipe.org/

OpenHeadend

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

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**
- Sponsored by OpenHeadend
  - Intends to use Upipe in its products
  - Written by Christophe Massiot and Benjamin Cohen
  - Core under MIT, modules under GPLv2+ or LGPLv2+

OpenHeadend

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

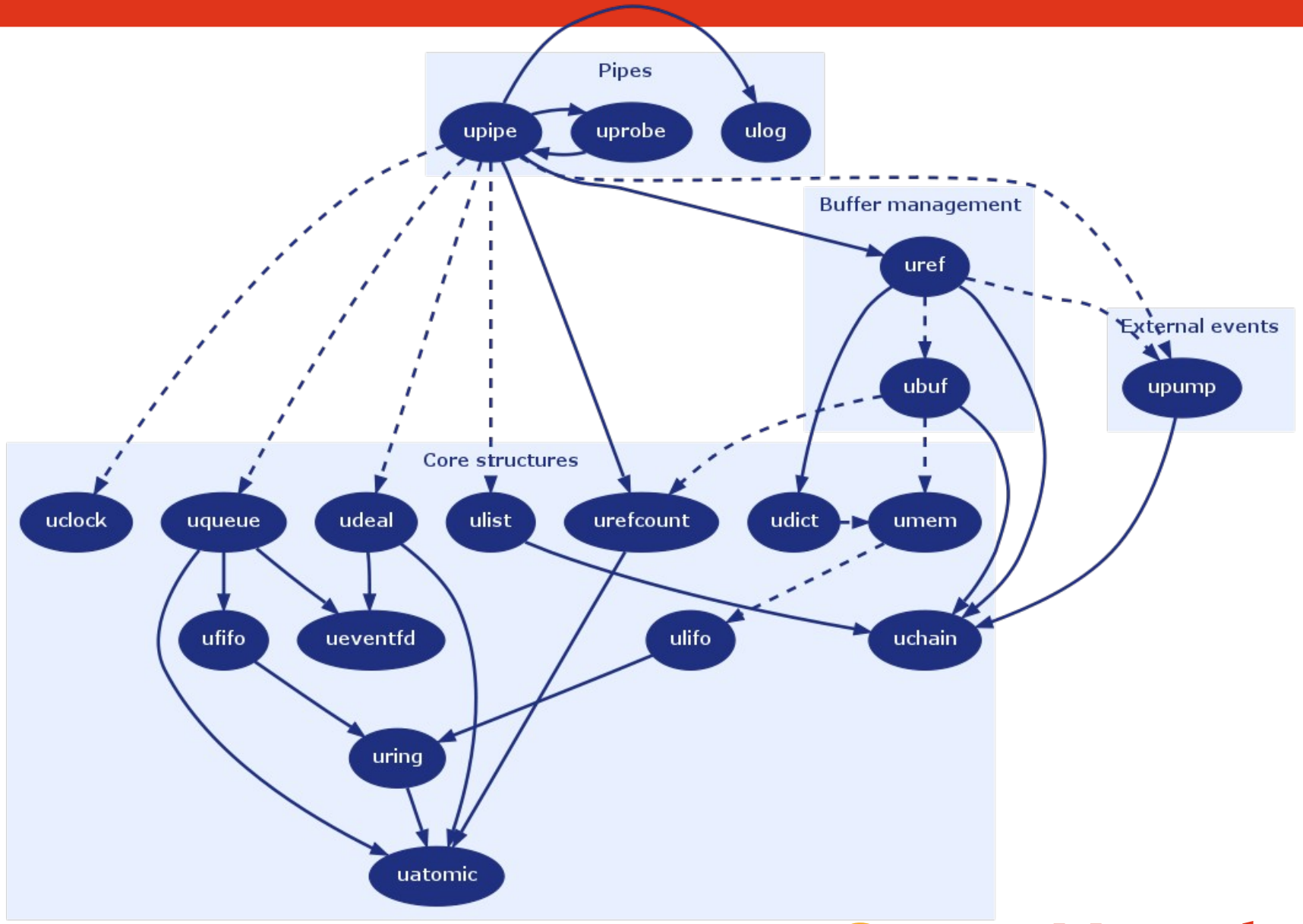
Probes

- 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
- Pipes create watchers on file descriptors, timers, and idlers and get called back
- The API needs extending for worker thread pools

In a nutshell



OpenHeadend

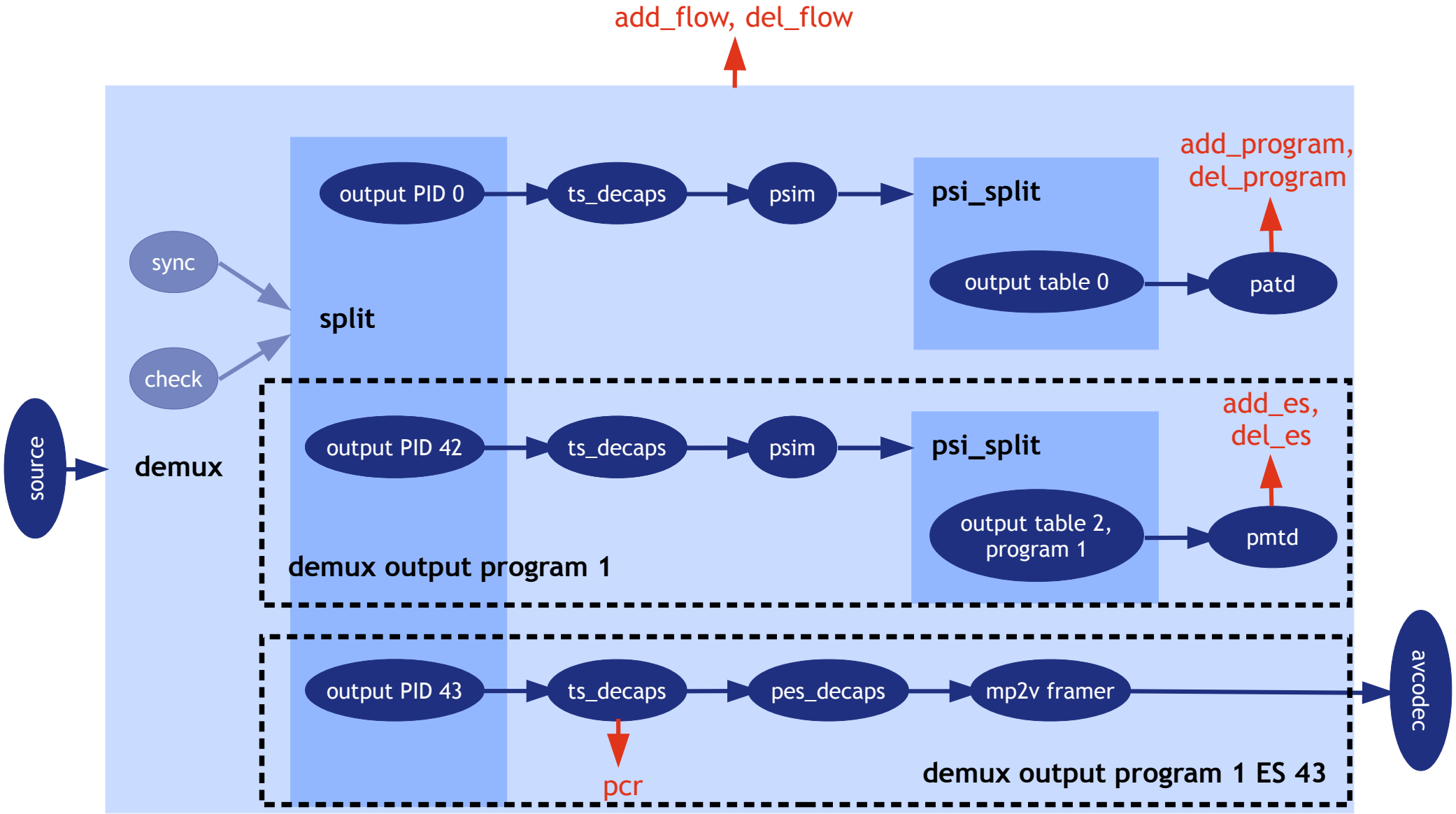
Upipe development status

- Basic structures are in place
- Pipes in development or already available:
  - File source and sink, UDP source
  - Lock-less queue between threads
  - “dup” pipe
  - Libavformat source
  - Libavcodec and swscale
  - TS demux
- API still to be considered unstable

Feedback on requirements

- Pipes are very small, modular objects that can provide a multi-level API, combined with clever “probes”
- Buffer structures are thread-safe and lock-less
- Upipe doesn't deal with “threading”:
  - The application may run pipes in different threads and place queues where needed
  - Some tasks will be off-loaded to dedicated or common thread pools

Case study: anatomy of the TS demux



OpenHeadend