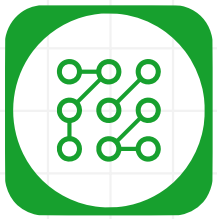
Multithreading and other developments in the FFMPEG transcoder

Anton Khirnov

FFlabs

2024-02-04
FOSDEM



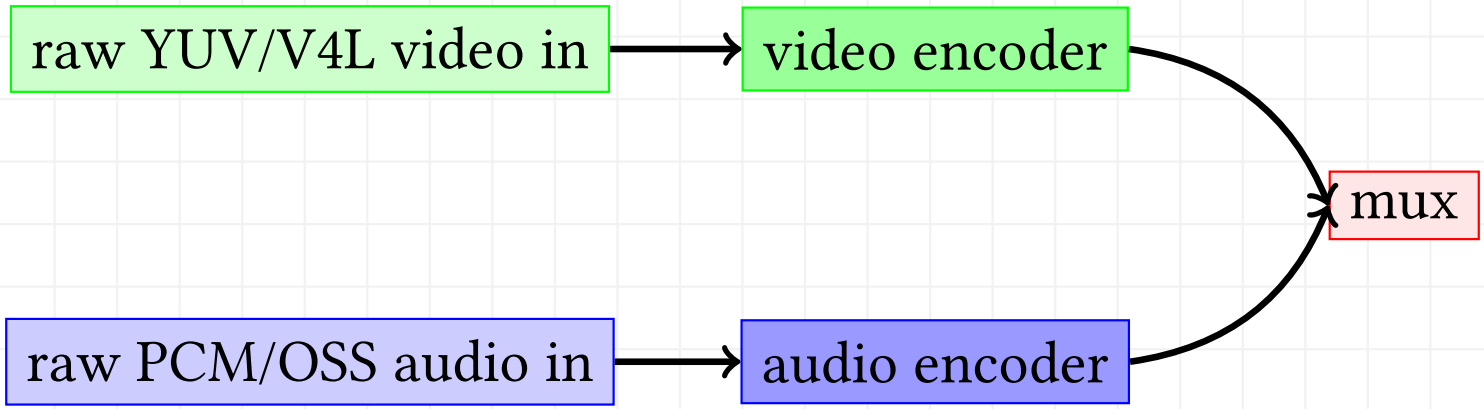
The FFMPEG project

- the LIBAV* libraries
  - LIBAVCODEC: decoders, encoders, bitstream filters, ...
  - LIBAVFORMAT: demuxers, muxers, IO, ...
  - LIBAVFILTER: audio/video filters
  - ...
  - widely used as the backend for multimedia playback and processing
  - media players, web browsers, transcoders, thumbnailers, ...
- commandline tools (CLI)
  - FFMPEG transcoder
  - FFPROBE prober/analyzer
  - FFPLAY player

- most widely used multimedia transcoder on at least two planets
- uses LIBAV* libraries to demux, decode, filter, encode, mux, ...
- almost all format-specific logic is in the libraries
- is usually the first user of new library features and APIs
- covers more use cases than any other comparable tool
- all scales — from individual users to giant corporations

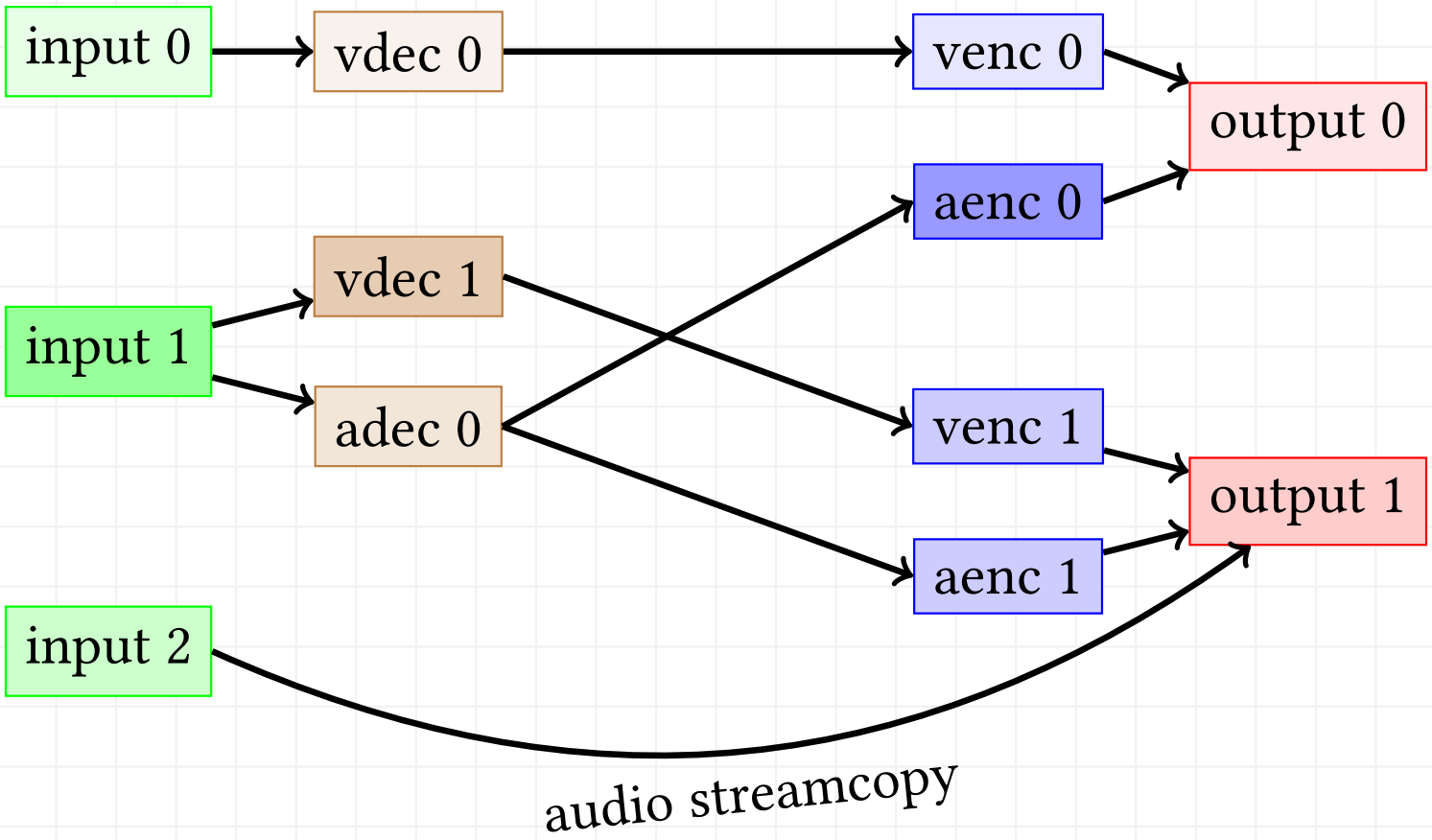
A brief history: 2000

- ~700 LoC
- raw input only, no decoding
- encoding and muxing



A brief history: 2001

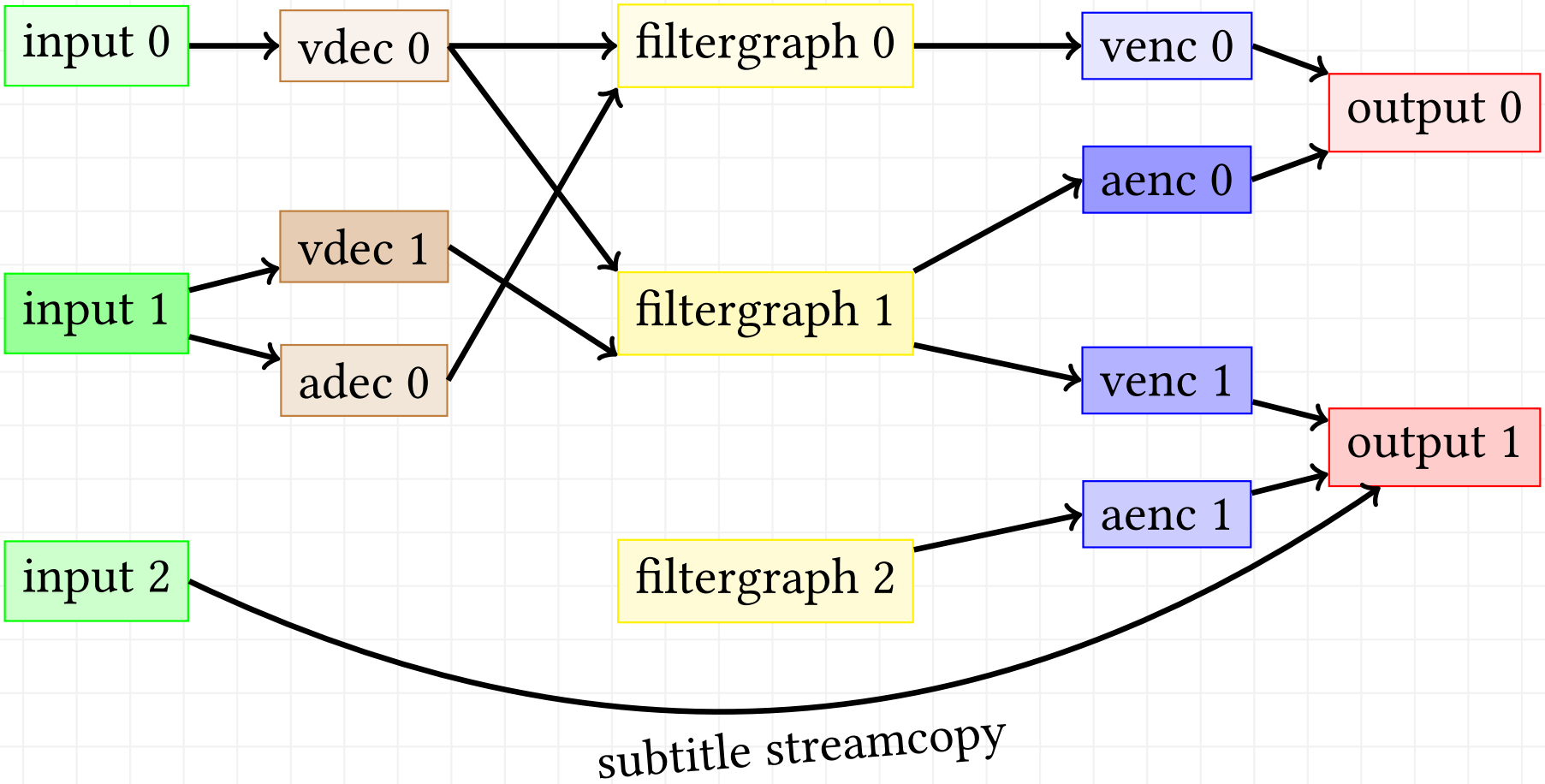
- ~2000 LoC
- demuxing and decoding
- multiple input and output files with multiple streams each



A brief history: up to 2022

- 2005 — subtitles (~4.5 kLoC)
- 2010 — simple video filtering with LIBAVFILTER (~4.5 kLoC)
- 2012 — complex filtergraphs (~5 kLoC)
- 2013 — basic hardware acceleration (~6 kLoC)
- 2016 — full hwaccel pipelines become possible (~8 kLoC)
- 2022 (project start):
  - ~11 kLoC
  - dynamic stream parameter changes
  - more options than anyone can remember
  - options interact in highly nontrivial ways

Current general transcoding pipeline



How did we get from 2000 to 2022?

```
while (1) {
```

- somebody needs a shiny new feature
- they implement it, optimizing for
  - smallest amount of work
  - smallest diff
- usually NOT optimizing for
  - ease of future development
  - clean overall design

```
}
```

- every such step adds a multiplicative factor to overall program complexity
- IOW complexity grows exponentially

…in programming simplicity and clarity —in short: what mathematicians call "elegance"— are not a dispensable luxury, but a crucial matter that decides between success and failure

E. W. Dijkstra
EWD648

Project goals

- bring code structure in alignment with actual data flow
- this is achieved by
  - making the code more explicitly object-oriented
  - clearly defined interfaces and responsibilities
  - separation of public and private state
  - every major component in its own thread
  - information flows downstream through the pipeline
- the code is easier to understand and maintain
- implementing major new features becomes feasible
- improved throughput under the right conditions

Progress & status

- project started in late 2021
- upstreamed continually in ~50 patchsets of small-moderate size
- 700+ commits overall, almost every line of code in `fftools/ffmpeg*` touched
- most of the work — moving things around, making state private
- final set merged in December 2023, will appear in upcoming 7.0 release
- extras
  - demuxing bitstream filters
  - latency probes
  - opaque passthrough
  - frame duration handling
  - timestamps handling improvements
  - sync queues

Ongoing & future work

- separate decoders from demuxers
  - looping an encoded stream back to a decoder
- separate encoders from muxers
  - every encoder currently coupled to an output stream
  - sending an encoded stream to multiple muxers
- dynamic pipelines
- scripting (Lua?)
- event loop-based architecture?