Server-side Media Processing with GStreamer

Arun Raghavan

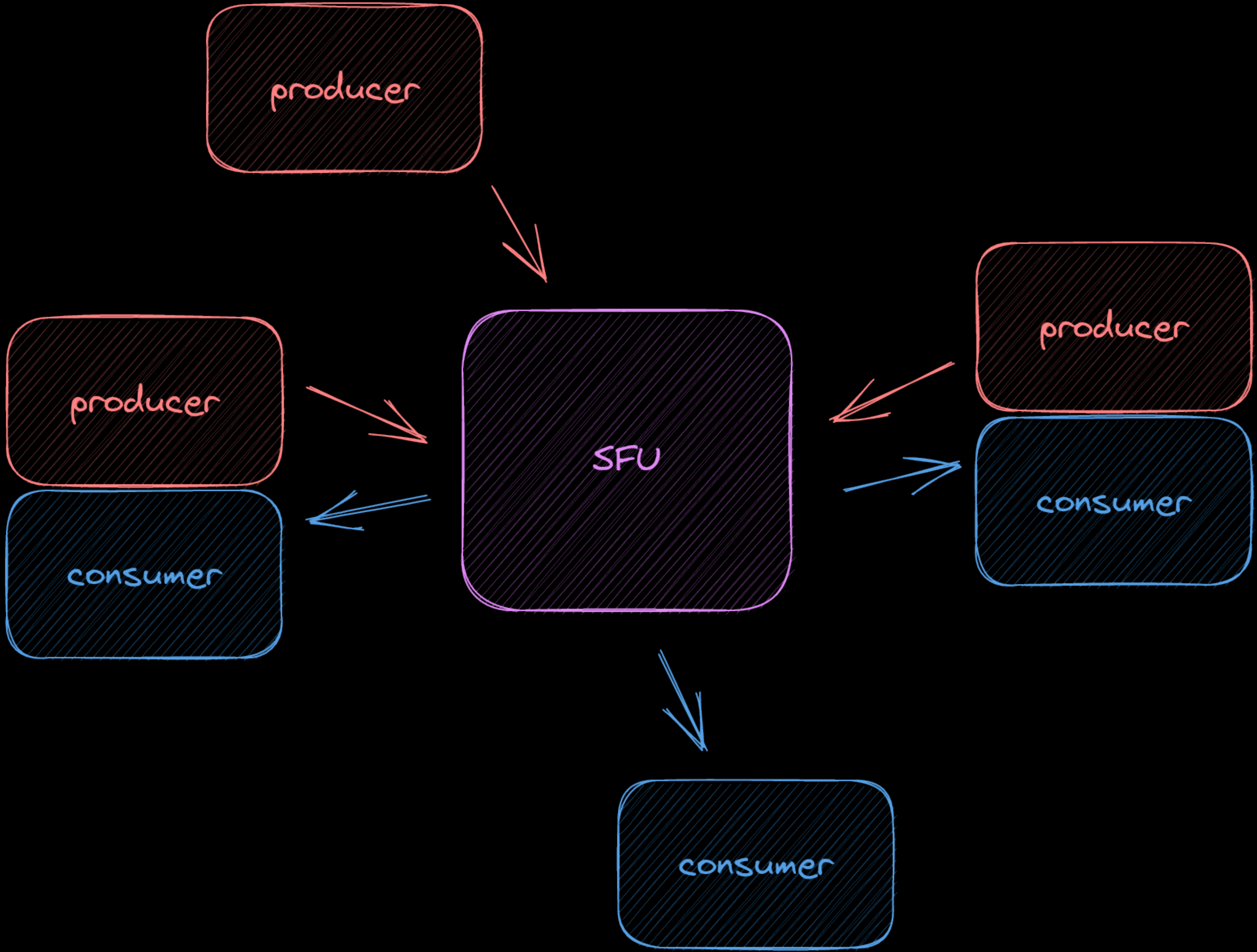
asymptotic

We all love GStreamer on our

- laptops
- phones
- speakers
- TVs
- space robots

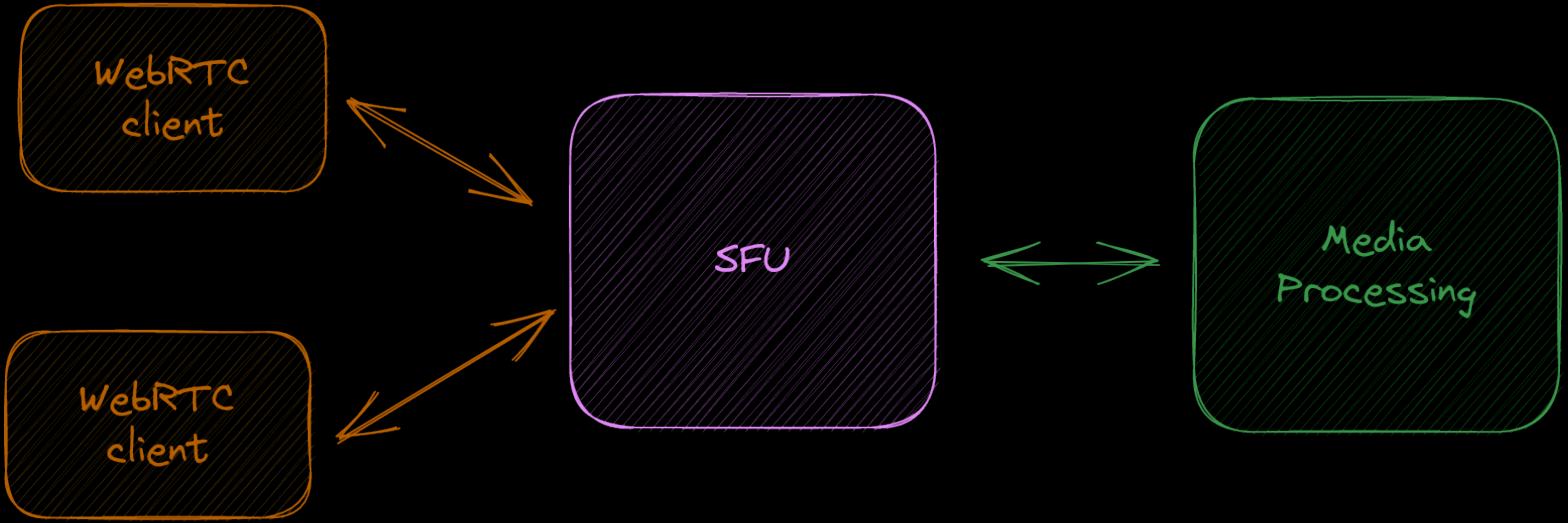
Today, let's talk about server-side processing daily.co

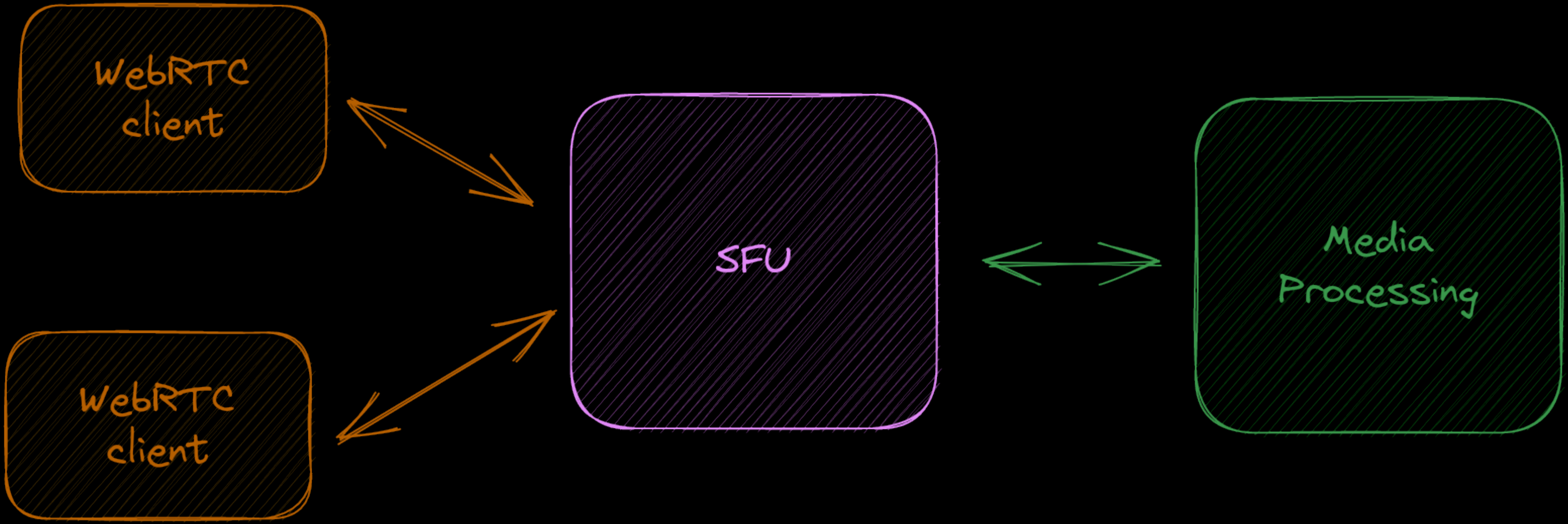
- Calling Platform as a Service ("CPaaS")
- WebRTC SDK for clients
- SFU in the backend
- A whole bunch of features on top


producer
producer
producer
SFU
consumer
consumer
consumer

`mediasoup` as the SFU

- Powerful set of features
- Built on `Producer`, `Consumer`, `Transport`
- Routes WebRTC clients between each other
- Can also route plain RTP-over-UDP


WebRTC client
WebRTC client
SFU
Media Processing



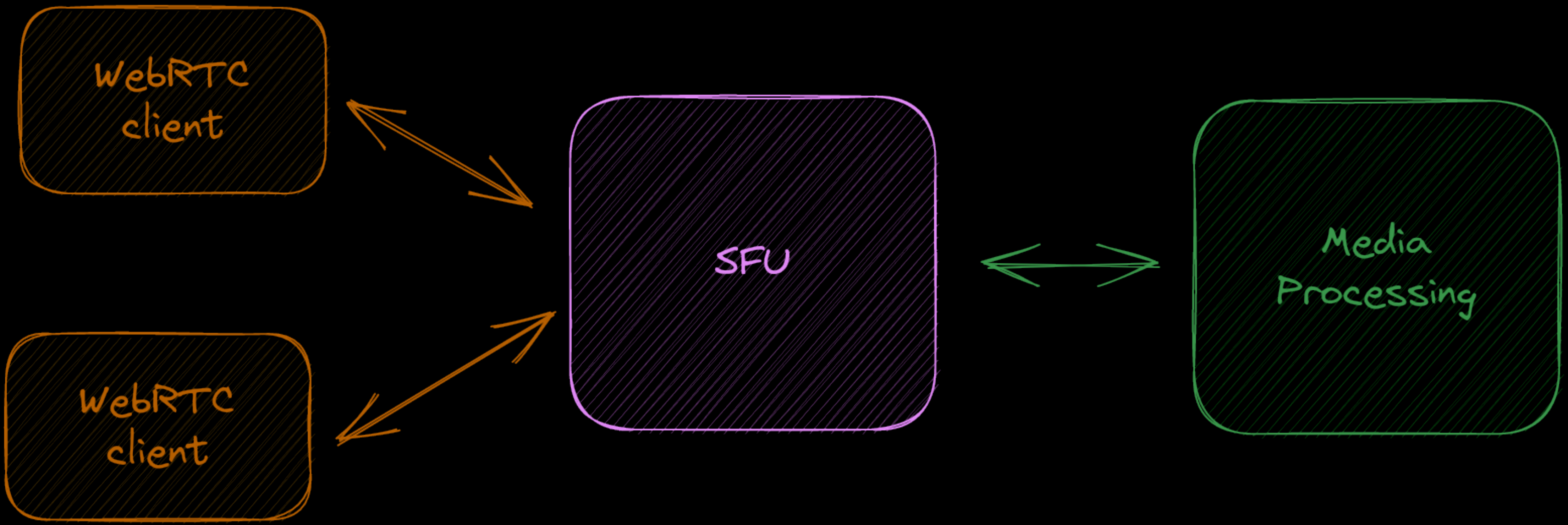
media worker can scale independently

Live streaming & recording

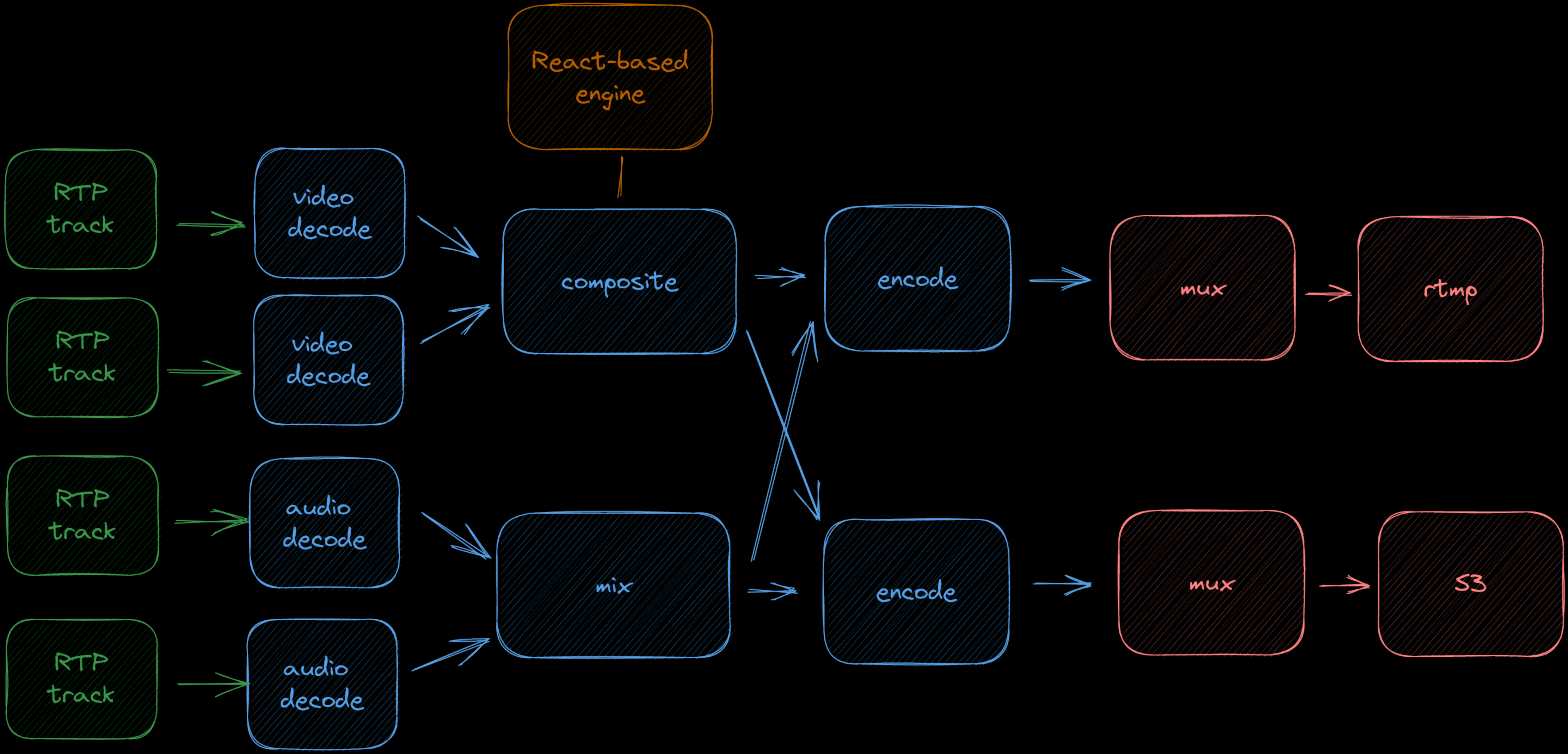
- Consume a set of audio/video streams
- Mix/composite them
- Output to
  - RTMP
  - S3 (cloud storage)
  - HLS

Live streaming & recording: approaches

❌ Record on client
❌ Headless participant (or Chrome-in-the-cloud)
✔️ Remote rendering pipeline



media worker runs a live streaming / recording pipeline

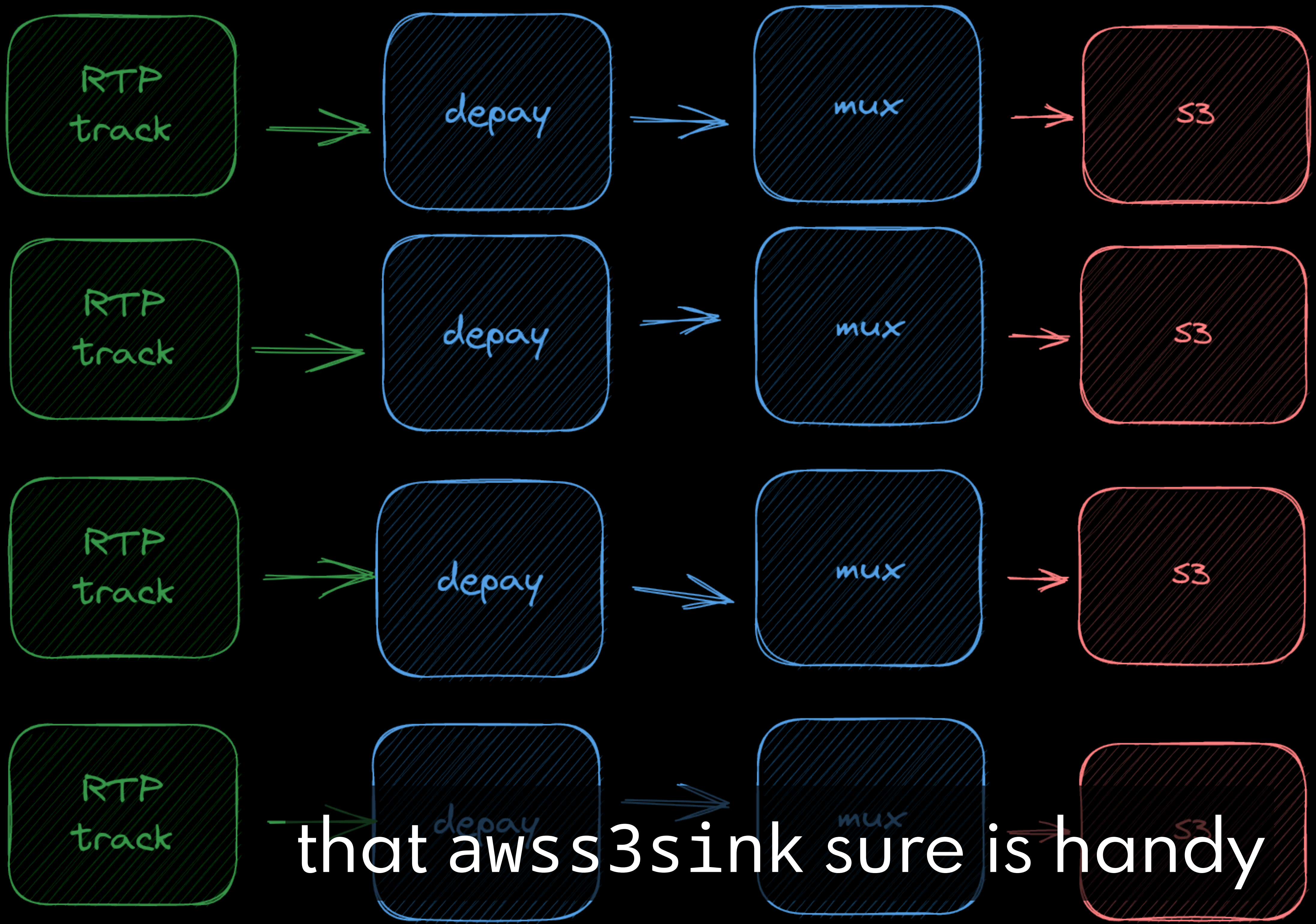

React-based engine
RTP track
RTP track
RTP track
RTP track
video decode
video decode
audio decode
audio decode
composite
mix
encode
encode
mux
mux
rtmp
S3

Live streaming & recording: learning

- React-loop + custom renderer
- Video processing performance
- Judicious use of queues + monitoring
- Dynamic pipelines vs. `StreamProducer`
  debuggability vs. resilience
- `rtmp2sink` retries: `GAsync` is hard
- `awss3hlssink`: HLS for "free"

Recording raw tracks

- Simpler version of the recording pipeline
- Record each track independently
- Review/process/composite later



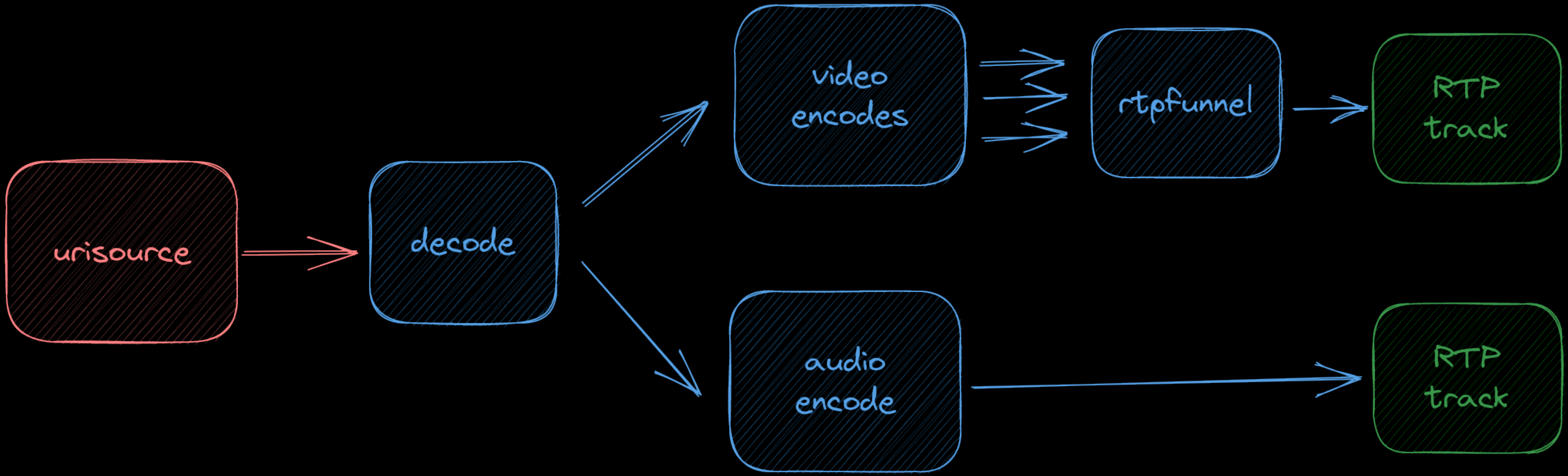
that awss3sink sure is handy

Recording raw tracks: learning

- Streaming to S3 adds resilience
- No way to correlate tracks initially
  - `webmmux cluster-timestamp-offset=...`
- Need to upstream patches for demux

Media ingestion

- We can also feed media back in
- Shared media player in a call
- Pre-recorded content, hold music, ...

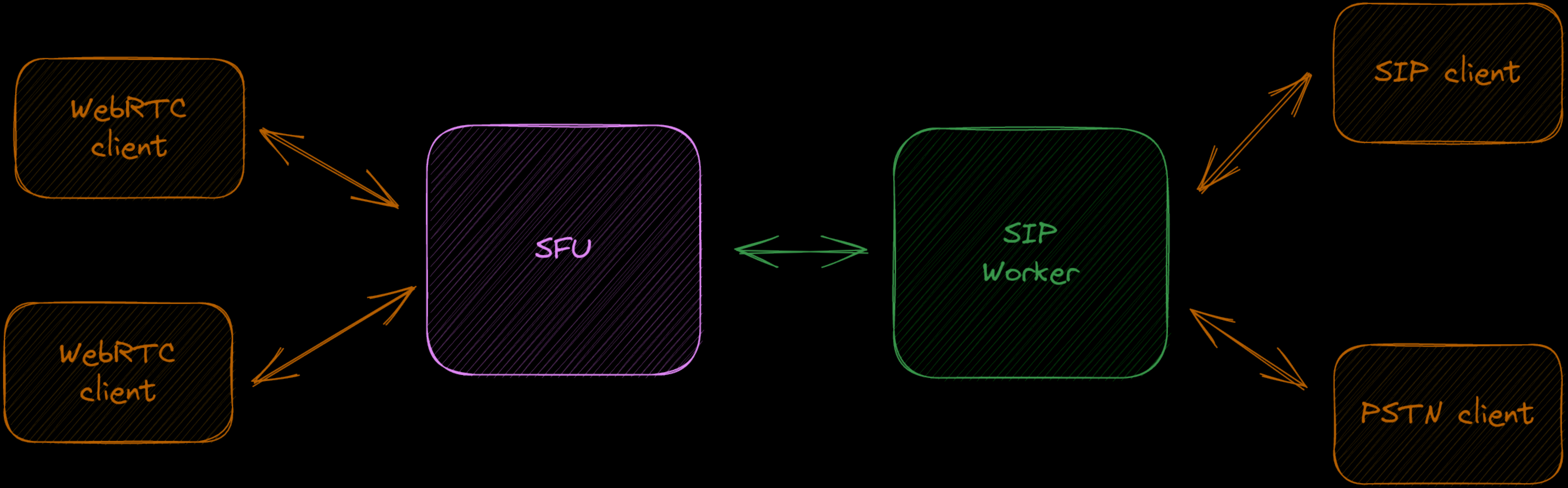

urisource
decode
video encodes
rtpfunnel
RTP track
audio encode
RTP track

Media ingestion: learning

- Simulcast layers via `rtpfunnel`
- Need to care about live vs. non-live
  - Easy enough using `clocksync`

SIP

- Hopefully you saw Sanchayan's talk
- We have all the pieces for media in/out now
- Lots of reuse for talking to phones
  and physical conference equipment



this time the media worker consumes *and* produces media

SIP: learning

- A lot of application logic to reuse
- Refactored as bins
- Modeling `webrtcbin` after `RTCPeerConnection` was a good choice

General learning

- Observability is important
- Understanding threading model key for profiling
- Bins are handy for abstracting application logic
  - JSON >>> `GstStructure` for API

Questions?

♥