Hardware Accelerated Live Broadcasting

Johan Jino, Gareth Sylvester-Bradley | GStreamer Conference 2025

Hardware Accelerated Live Broadcasting

Why the need for Hardware Acceleration?

Look at the Data Rate!

- Typical H.265 1080p60 stream is around 5-10Mb/s
- Uncompressed 1080p60 stream, 10bit, 4:2:2 subsampling is 2.5Gb/s
- 250x higher than typical compressed streams!
- Single 4k60 streams go above 10Gb/s
- Professional production facilities need support for multiple uncompressed video streams at high res and frame rate

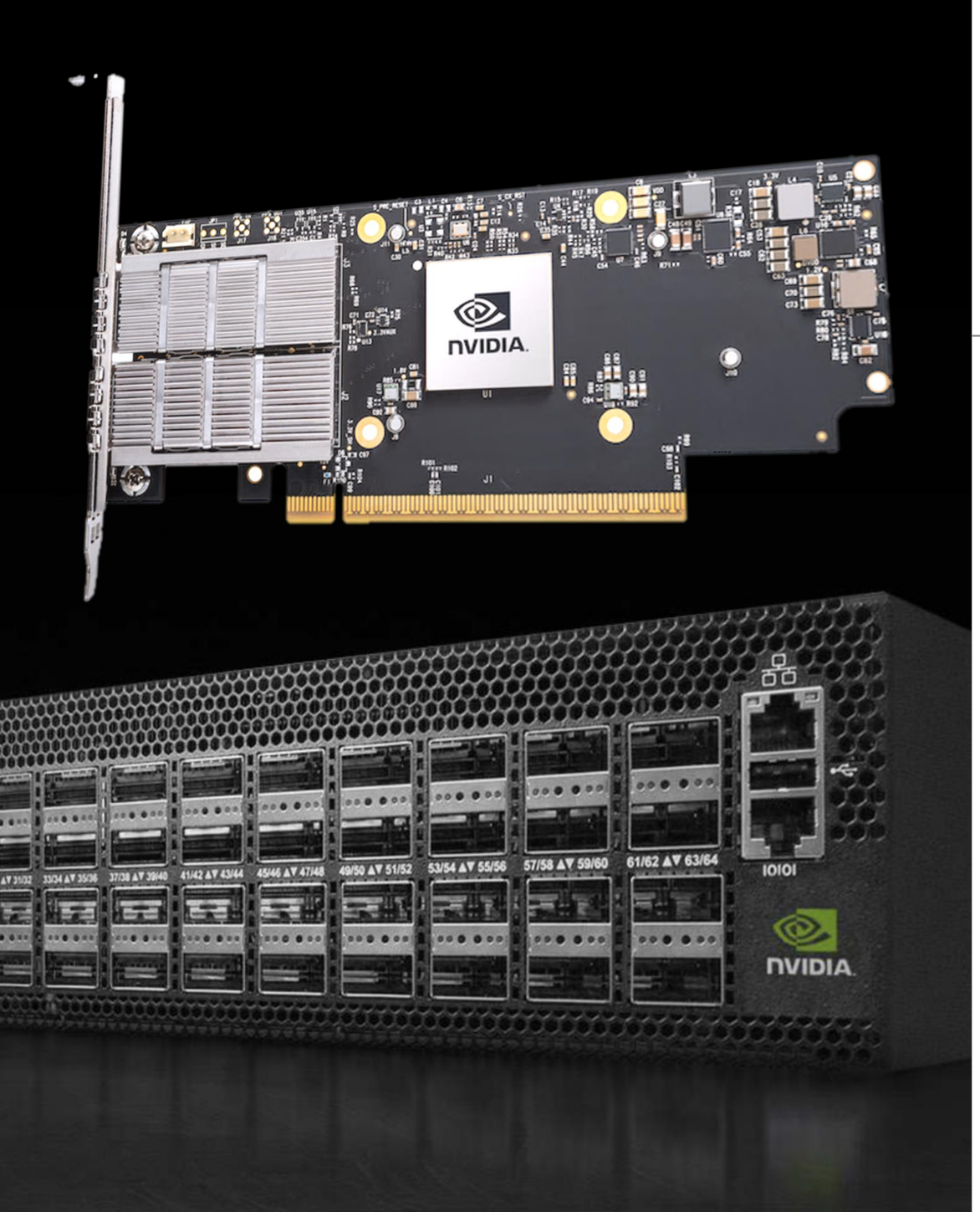
But why do professional broadcast/live production industry need uncompressed streams?

- Quality matters through multiple hops in production. No encode/decode loss.
- Minimize latency. Encoding/Decoding adds latency between every transmission. Multiple hops, this accumulate.

Serial Digital Interface (SDI)

Standardized by SMPTE

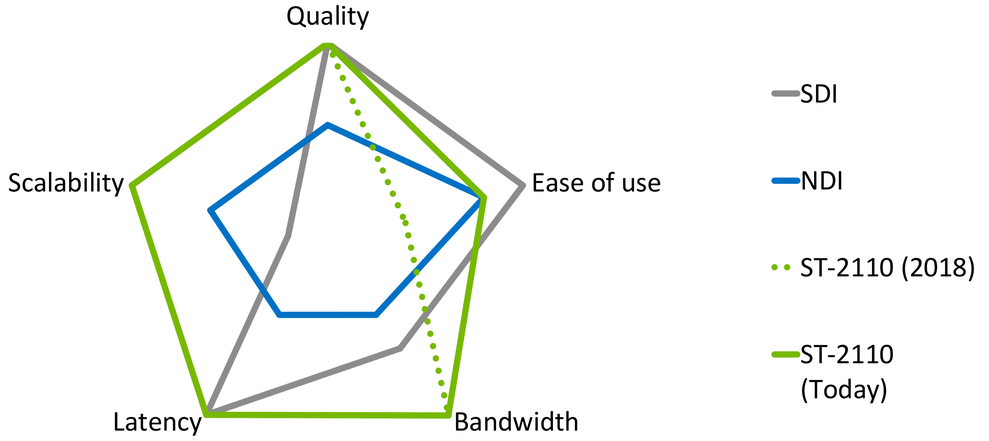
- Uncompressed unencrypted digital video signal
- First introduced in 1989 with regular refreshes until 2015
- 3G-SDI (2006) 3Gb/s 1080p60 is mostly used
- A lot of production still using 1080 interlaced cameras
- Multi-camera lock/sync, audio & ancillary data with same connectors



IP Transition

SMPTE ST 2110 – RTP over UDP with traffic shaping

Technology	SDI	NDI	SMPTE ST 2110
Image Quality	Excellent	Good	Excellent
Bandwidth	3-12 Gb/s	90 to 350 Mb/s	1 to 50 Gb/s (8K)
Network Type	SDI	1 to 10 Gb/s	10 to 400 Gb/s
Synchronization	Genlock	Possible	PTP



Hardware Accelerated Live Broadcasting

Broadcasting of Uncompressed ST 2110 Streams with GStreamer leveraging NVIDIA GPUs and NICs

- ST 2110 operates by sending media streams over IP networks, like how data travels on the internet, but with broadcast-grade precision.

- Why use GStreamer in between?
  - Brings advanced media applications and ease-of-use
  - Consistent high-level programmatic APIs (C and Python) for plugins
  - Ability to add own custom plugins and app integration

- What is needed for GStreamer to deliver full ST 2110 and more:
  - Packet pacing -> **`NvDsUdp`**
  - Hardware Optimised-> Also **`NvDsUdp`**
  - Dynamic reconfigurability -> **`NvDsNmosbin`**
  - Ancillary Data Streaming -> **`rtpsmpte291pay/depay`**

highlighted are NVIDIA built GStreamer plugins/elements

DeepStream and Rivermax

Toolkit for video, audio, and image understanding

- DeepStream Extends GStreamer via plugins
  - GPU accelerated filter elements
  - Efficient buffer passing between elements
  - Integrates with NVIDIA AI toolkits
  - SMPTE ST 2110 source and sink elements
  - NMOS support library
  - Uses Rivermax for optimized networking
- Rivermax is an optimised networking SDK for media streaming
  - API for ST 2110 Tx and Rx on COTS NICs, e.g. ConnectX-6Dx
  - Provides highly optimised network transfers
  - Used by DeepStream NvDsUdp plugin

DeepStream NvDsUdp Plugin

Technical Deepdive

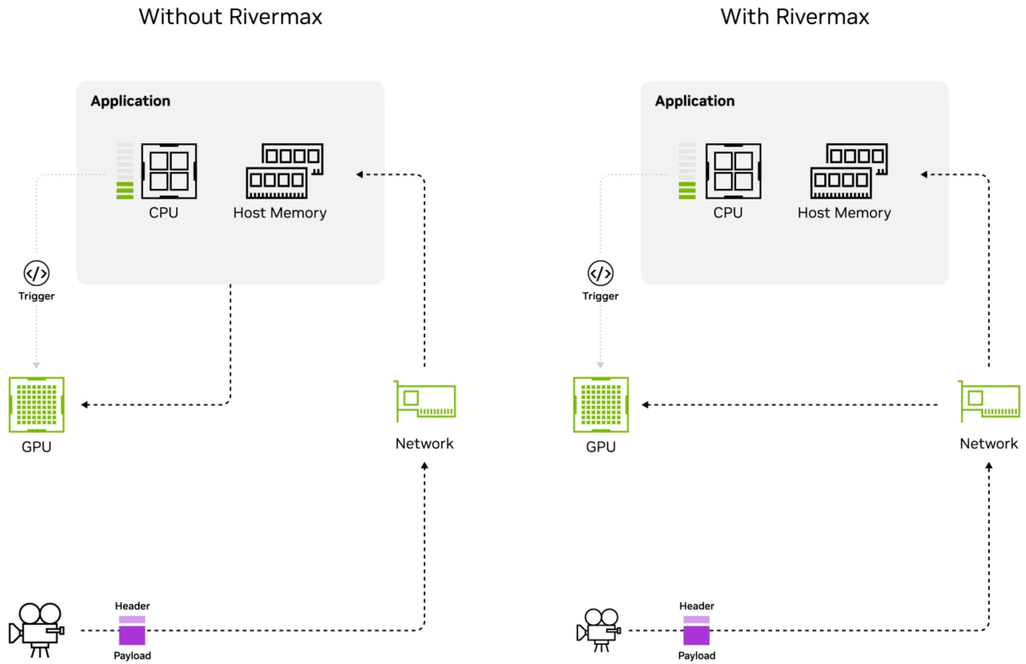
- GStreamer plugin for hardware-accelerated (Rivermax-enabled) transport that replaces generic GstUdp src/sink.
  - Like GstUdp but with built-in rtppay and rtpdepay*
  - Direct GPU-to-NIC memory transfers using kernel bypass
  - Packet paced according to SMPTE ST 2110
- Benefits:

* Currenly only for raw video and audio

DeepStream NvDsUdp Plugin

Technical Deepdive

- GStreamer plugin for hardware-accelerated (Rivermax-enabled) transport that replaces generic GstUdp src/sink.
  - **Like GstUdp but with built-in rtppay and rtpdepay***
  - Direct GPU-to-NIC memory transfers using kernel bypass
  - Packet paced according to SMPTE ST 2110
- Benefits:
  - Supports full ST 2110, which constrains some RTP packet headers more than the base RFC 4175 (rtpvraw)
  - More performant than single-threaded implementation of the OSS payloader which isn't sufficiently performant for 4k60,
  - Takes advantage of Rivermax's header-data split feature where payload goes to/from the GPU but headers are handled on the CPU.

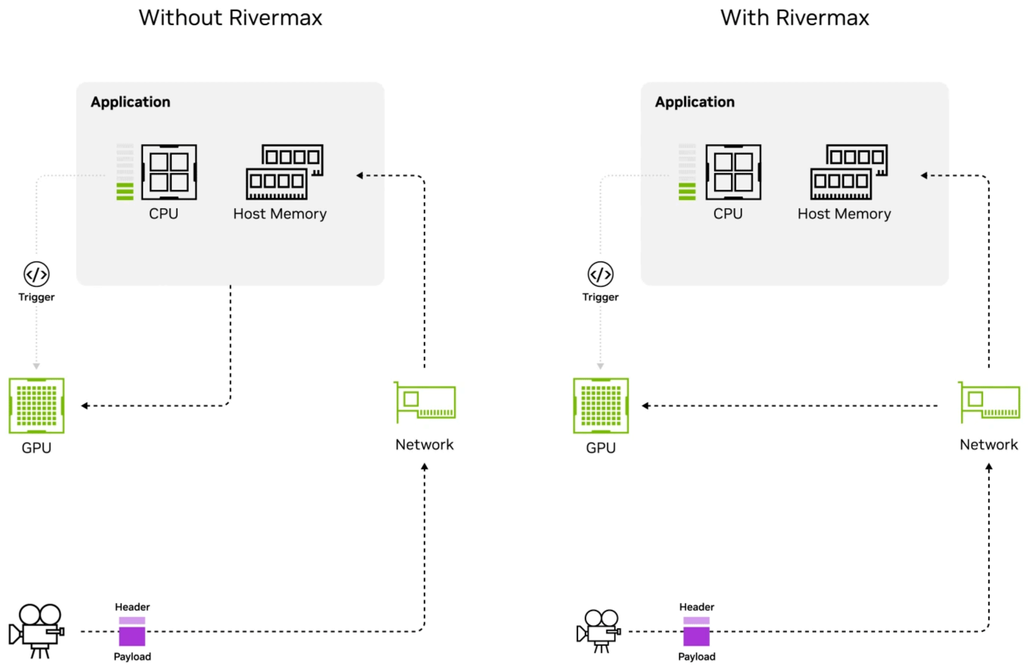


* Currenly only for raw video and audio

DeepStream NvDsUdp Plugin

Technical Deepdive

- GStreamer plugin for hardware-accelerated (Rivermax-enabled) transport that replaces generic GstUdp src/sink.
  - Like GstUdp but with built-in rtppay and rtpdepay
  - **Direct GPU-to-NIC memory transfers using kernel bypass**
  - Packet paced according to SMPTE ST 2110
- Benefits:
  - Lower CPU Utilization –10Gb/s 85% of 1 Core Vs. 5% of 1 Core with Rivermax. (10Gb/s = 1 4k60 uncompressed stream)
  - Utilize NIC HW accelerators to increase performance and simplify the solution i.e. RTP header insertion/stripping
  - Reduced host memory to GPU memory copy, allowing higher throughput.

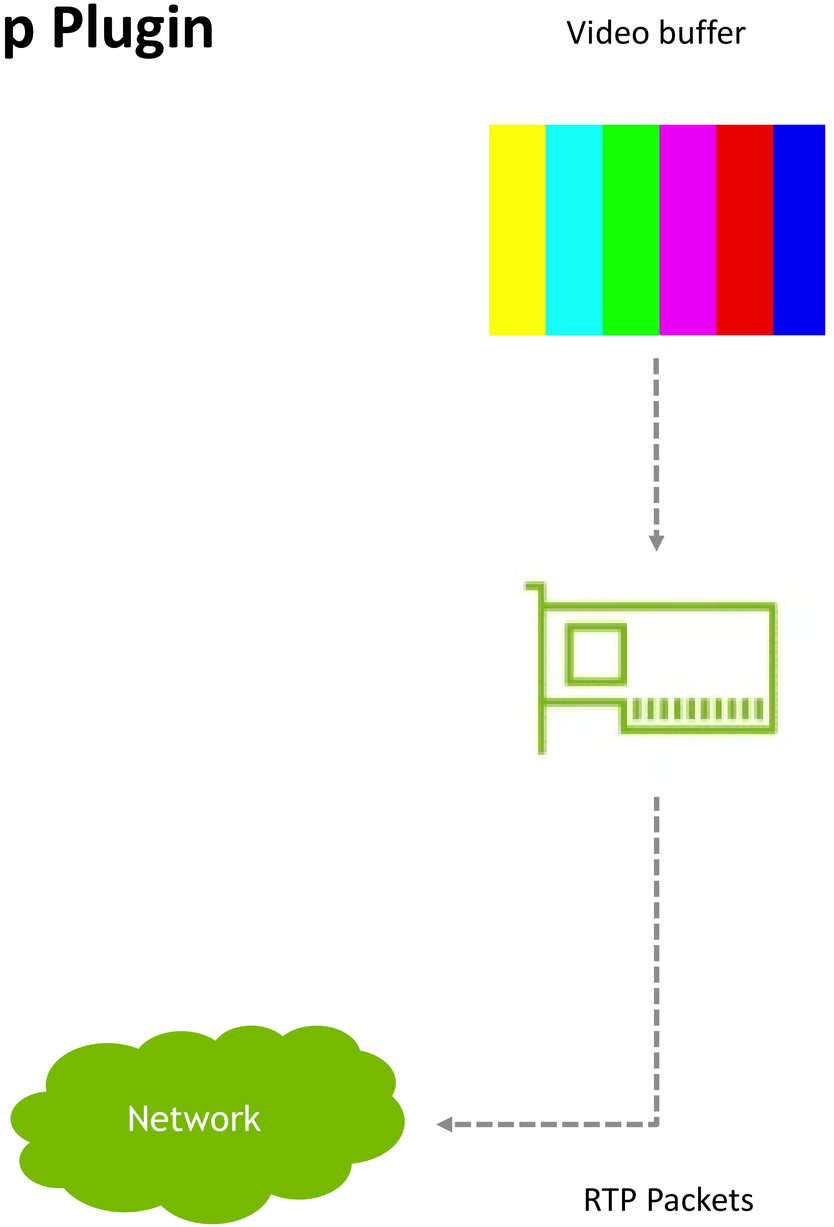


* Currenly only for raw video and audio

DeepStream NvDsUdp Plugin

Technical Deepdive

- GStreamer plugin for hardware-accelerated (Rivermax-enabled) transport that replaces generic GstUdp src/sink.
  - Like GstUdp but with built-in rtppay and rtpdepay*
  - Direct GPU-to-NIC memory transfers using kernel bypass
  - **Packet paced according to SMPTE ST 2110**
- Benefits:
  - Jitter free transmission
  - Inter Packet Gap for 4K can be < 1μs. Transmission done asynchronously in NIC HW ensures this target can be met.

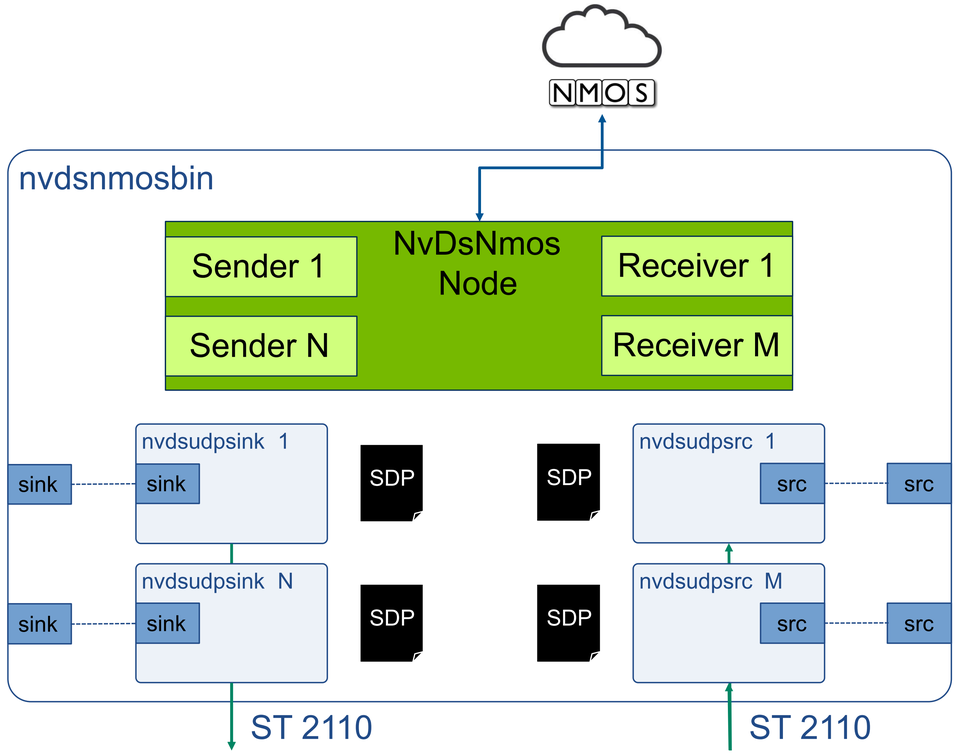


* Currenly only for raw video and audio

NMOS with GStreamer - NvDsNmosBin

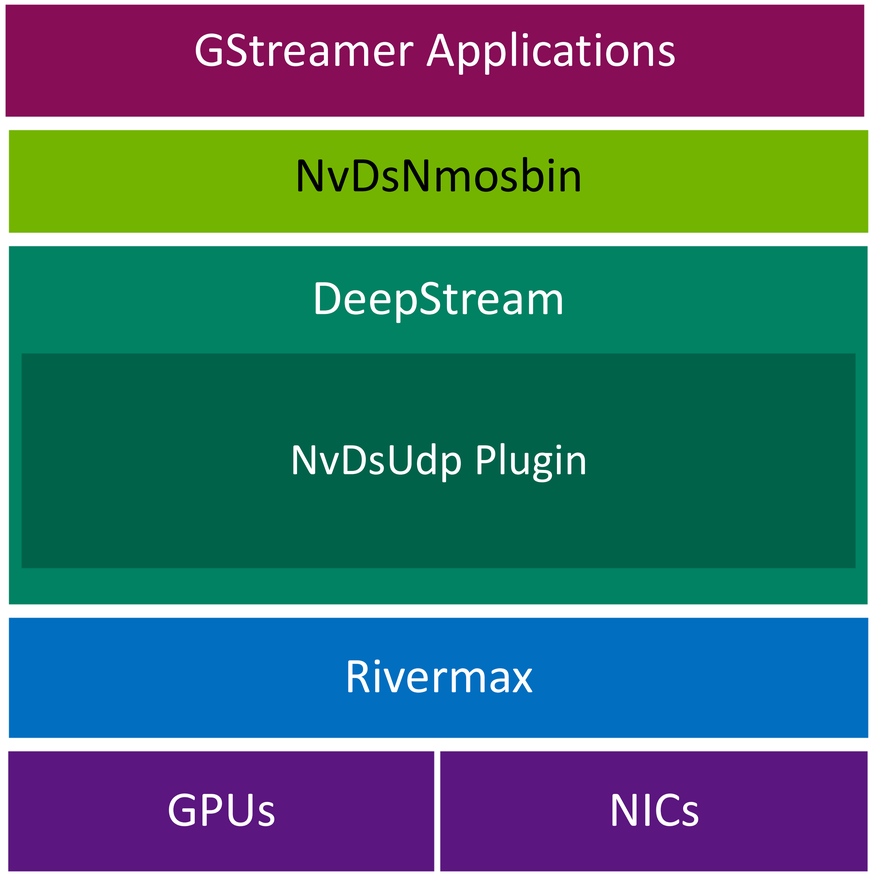
Networked Media Open Specification

- NMOS provides APIs for discovery, registration and control of ST 2110 and other media over IP networks
- **NvdsNmosBin** element bring this easy configurability to GStreamer
- Instantiates senders/receivers within pipelines and updates parameters on-the-fly using NMOS callbacks
- Uses GStreamer bins and ghost pads to dynamically configure and manage multiple sinks and sources
- Fully open source, built on top of other OSS including nvidia/nvnmos, sony/nmos-cpp and other low-level C++ libs.



Chaining them together

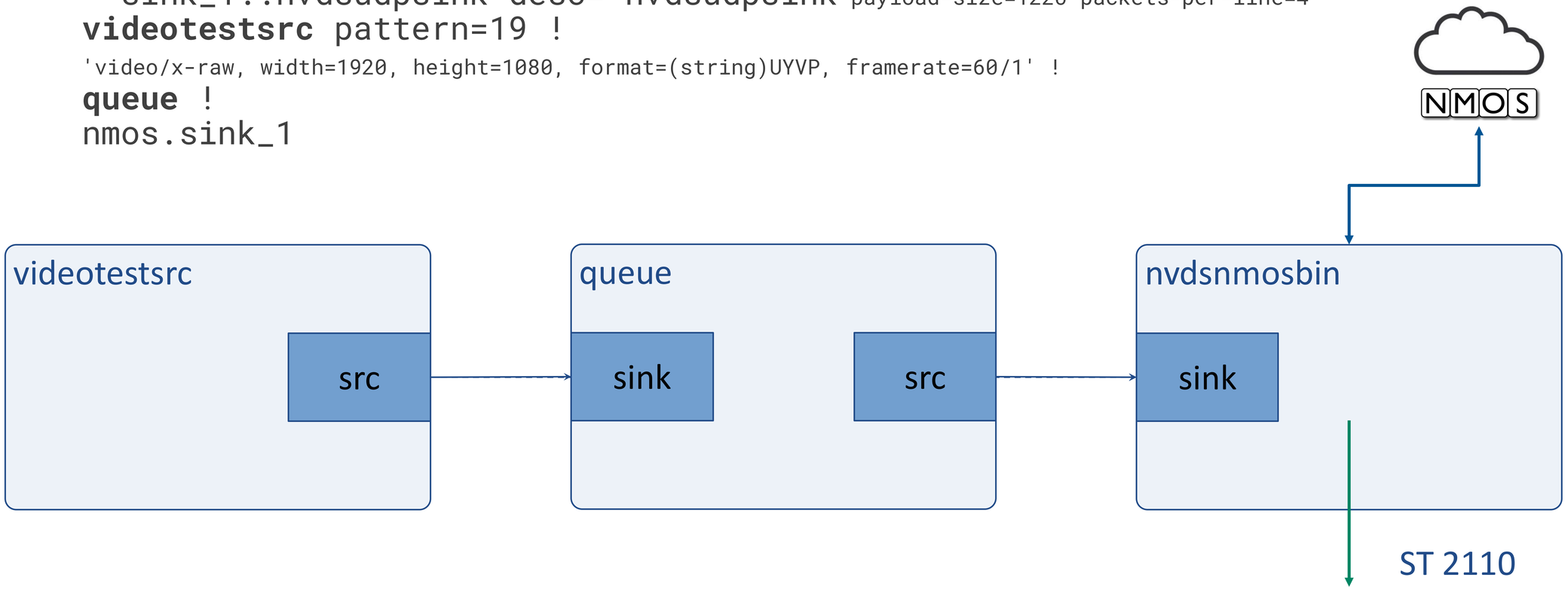
Brining performance within easy abstractions



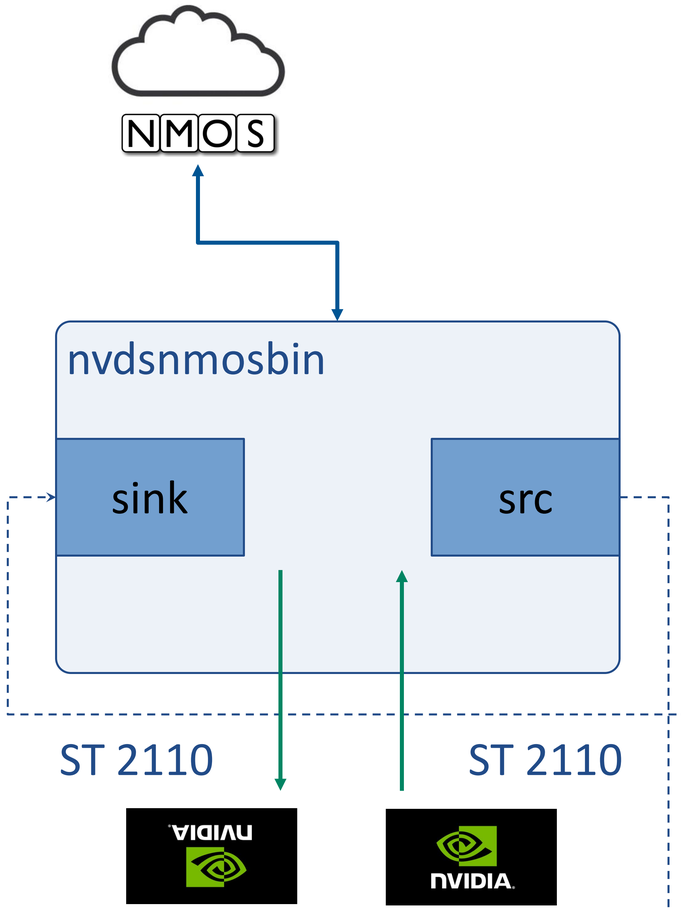
- Build simple GStreamer pipelines with just NvDsNmosbin sinks and sources
- Internally, it will create the correct NvDsUdp source and sink elements based on media and encoding type
- Result: Leverage Rivermax performance and NMOS configurability with 1 simple open source GStreamer element!
- Pass GPU (NVMM) buffers directly to other DeepStream accelerated elements
- Or connect to any GStreamer elements, via nvvideoconvert to move data from GPU to system memory

Simple ST 2110 video streaming

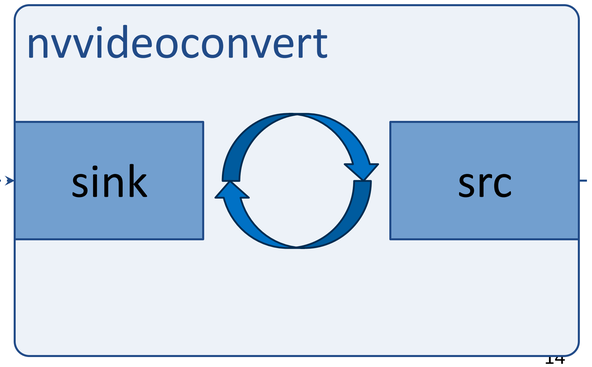
```
gst-launch-1.0
  nvdsnmosbin name=nmos
    sink_1::sdp=file:///example.sdp
    sink_1::nvdsudpsink-desc="nvdsudpsink payload-size=1220 packets-per-line=4"
  videotestsrc pattern=19 !
  'video/x-raw, width=1920, height=1080, format=(string)UYVP, framerate=60/1' !
  queue !
  nmos.sink_1
```



GPU-Accelerated Processing

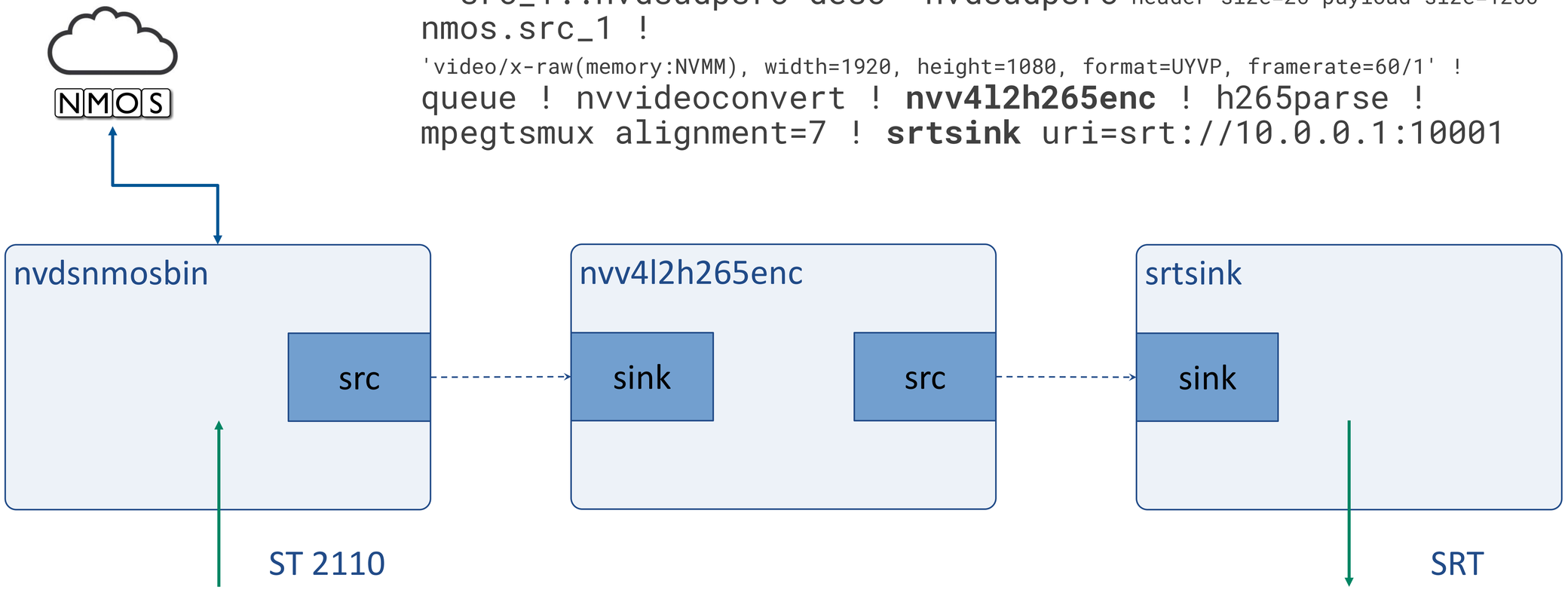



```
gst-launch-1.0
  nvdsnmosbin name=nmos
    src_1::sdp=file:///example.sdp
    src_1::nvdsudpsrc-desc="nvdsudpsrc …"
    sink_1::sdp=file:///example.sdp
    sink_1::nvdsudpsink-desc="nvdsudpsink …"
  nmos.src_1 !
  'video/x-raw(memory:NVMM), width=1920, height=1080, format=UYVP, framerate=60/1' !
  queue ! nvvideoconvert flip-method=rotate-180 !
  'video/x-raw, width=1920, height=1080, format=UYVP, framerate=60/1' !
  queue !
  nmos.sink_1
```



GPU-Accelerated Transcode

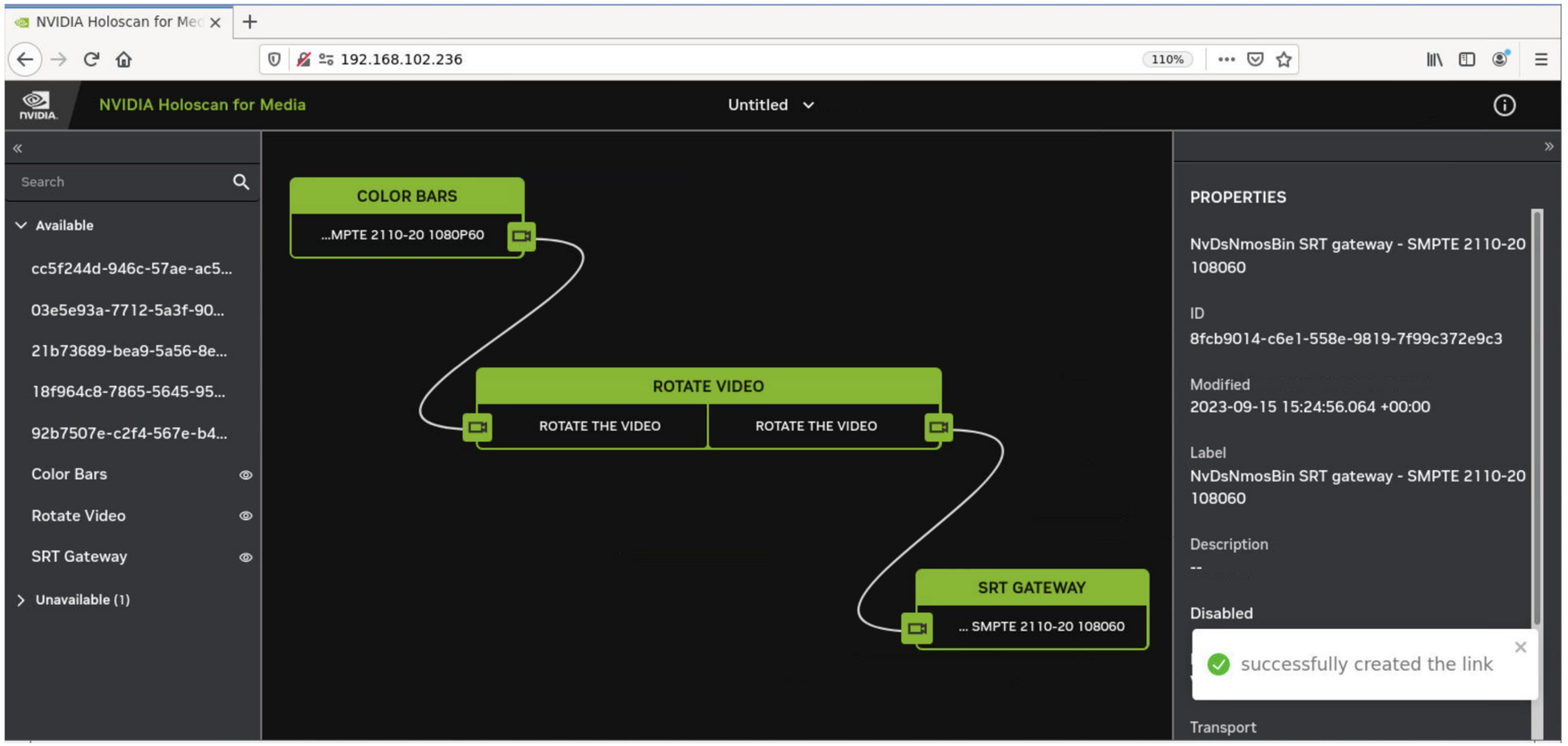
```
gst-launch-1.0
  nvdsnmosbin name=nmos
    src_1::sdp=file:///example.sdp
    src_1::nvdsudpsrc-desc="nvdsudpsrc header-size=20 payload-size=1200"
  nmos.src_1 !
  'video/x-raw(memory:NVMM), width=1920, height=1080, format=UYVP, framerate=60/1' !
  queue ! nvvideoconvert ! nvv4l2h265enc ! h265parse !
  mpegtsmux alignment=7 ! srtsink uri=srt://10.0.0.1:10001
```



Deploying Them All Together

Orchestrating GStreaming Pipelines

Each of these three Media Nodes is one of our GStreamer pipelines, connected via ST 2110 and NMOS



SMPTE Media Types

ST 2110 -20, -30, -40

- SMPTE ST 2110 specifies standards for video, audio and ancillary data formats
- Raw video can use `rtpvraw` and raw audio can use `rtpL24` (24bit) or `rtpL16` (16bit)
- What about Ancillary Data?

- How is Ancillary Data stored and streamed in GStreamer?
  - GstVideo.AncillaryMeta introduced in GStreamer 1.24
  - Contains all the fields for Ancillary data as specified by SMPTE 291
  - Propagates through the pipeline, till buffer is unreferenced
  - If video streamed over IP via RTP packets, the metadata is lost

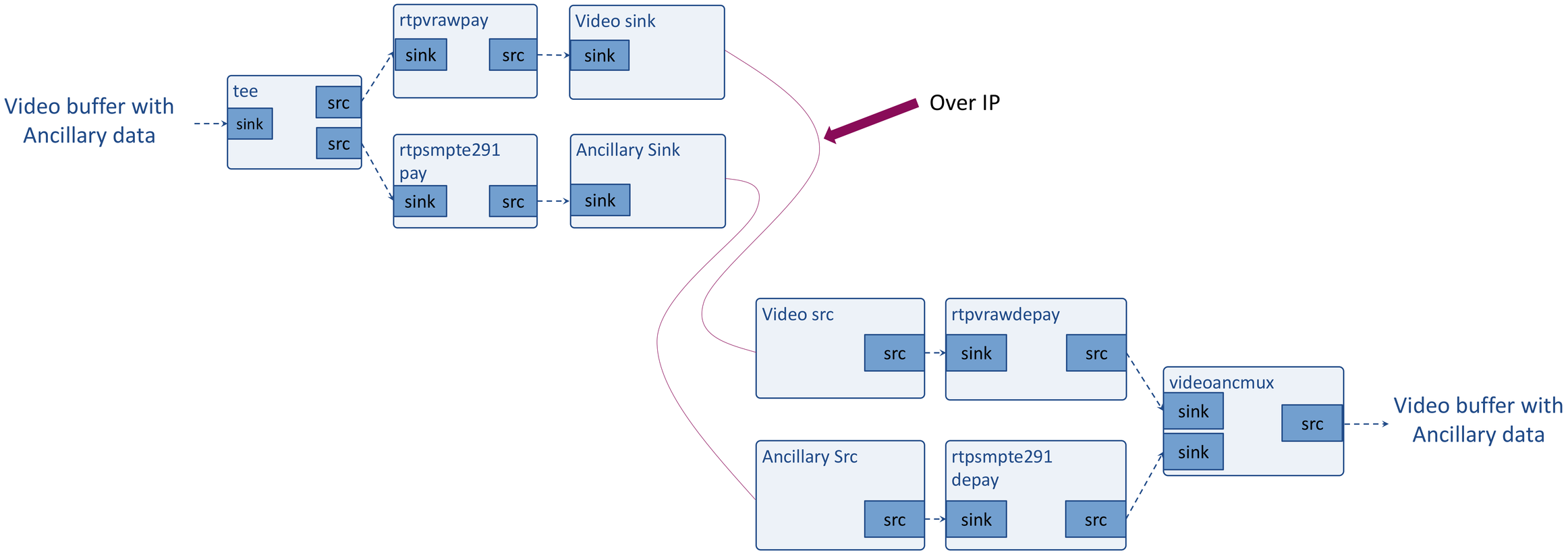
- SMPTE ST 2110-40 specifies the rules for transporting Ancillary Data over IP
- This includes format of the RTP packets which is done by reference to RFC 8331, in the same way that ST 2110-20 for video references RFC 4175, which is what the OSS rtpvraw(de)pay implement.
- However, no means currently in GStreamer to packetize Ancillary Data into RTP

SMPTE Media types

ST 2110 -20, -30, -40

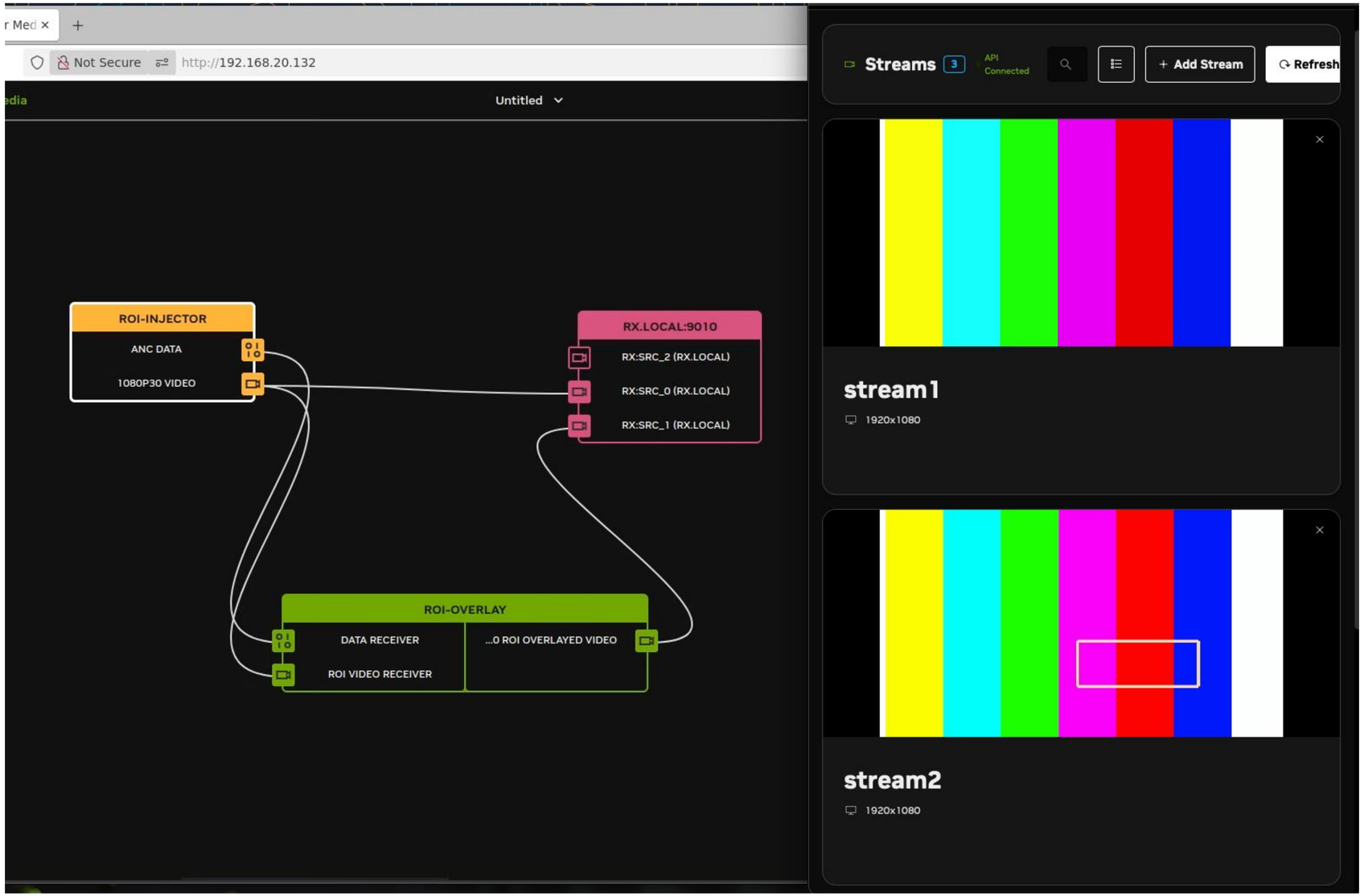
We introduce `rtpsmpte291` payloader/depayloader and a corresponding `videoancmux`

SMPTE 291 is the Ancillary Data Standard used in ST2110-40



Ancillary Streaming in Action

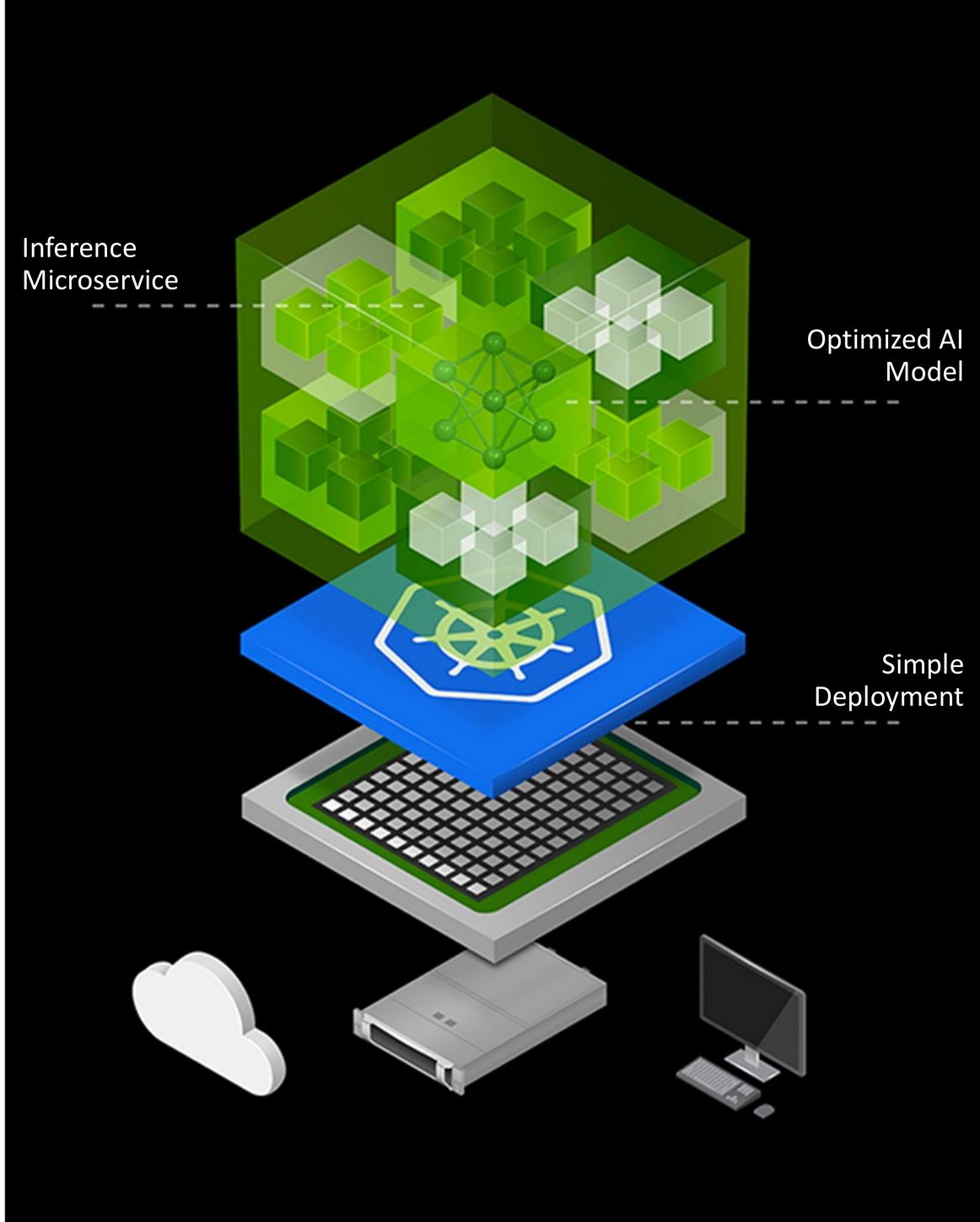
SMPTE ST2110-40



NVIDIA NIM

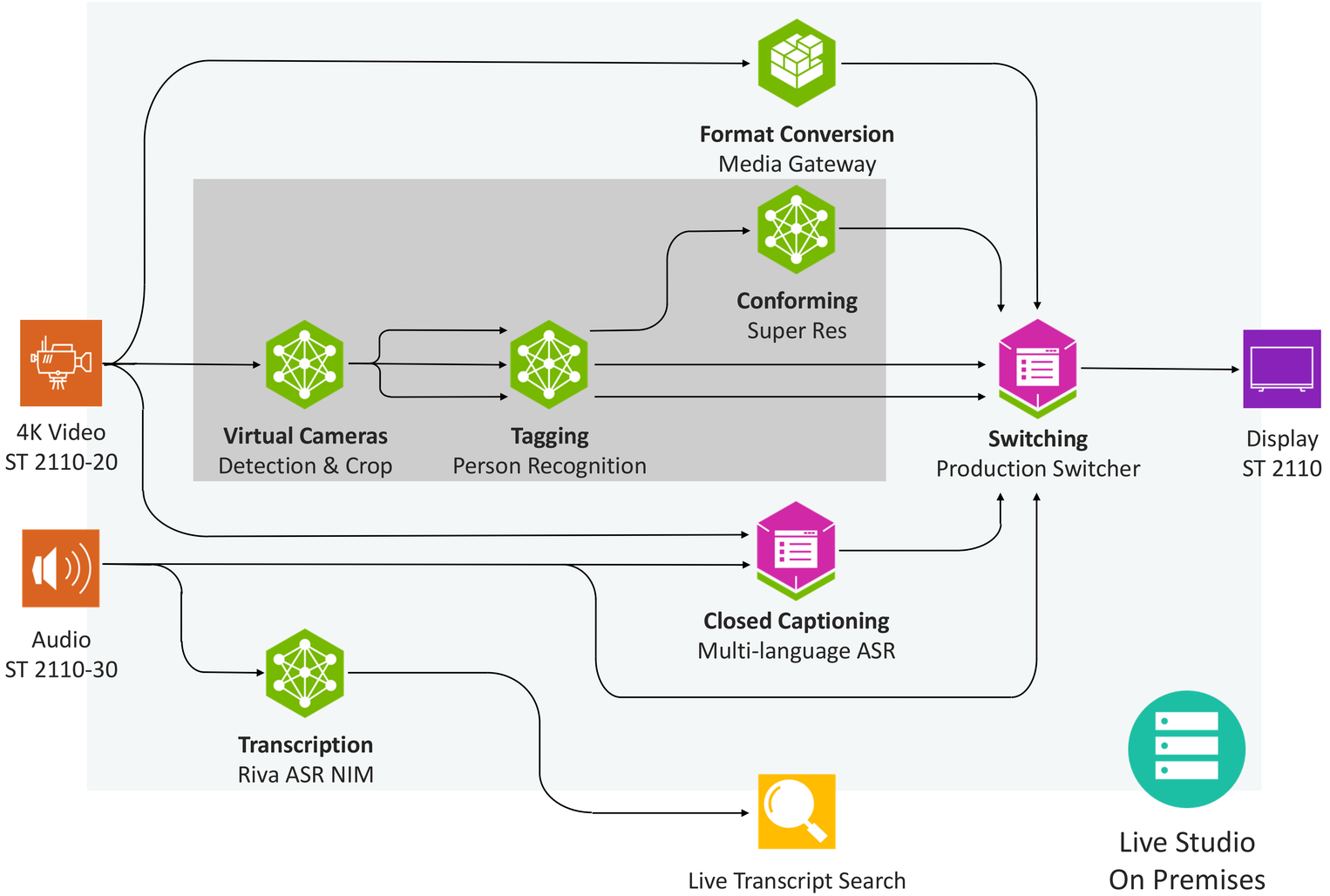
Inference Microservices

- Text, image, video and audio language models
- Optimized inference engines
- Industry-standard APIs
- Pre-configured containers for simplified deployment
- Blueprints for creating and deploying generative AI applications



Using NIMs in GStreamer Pipelines

AI For Live Media

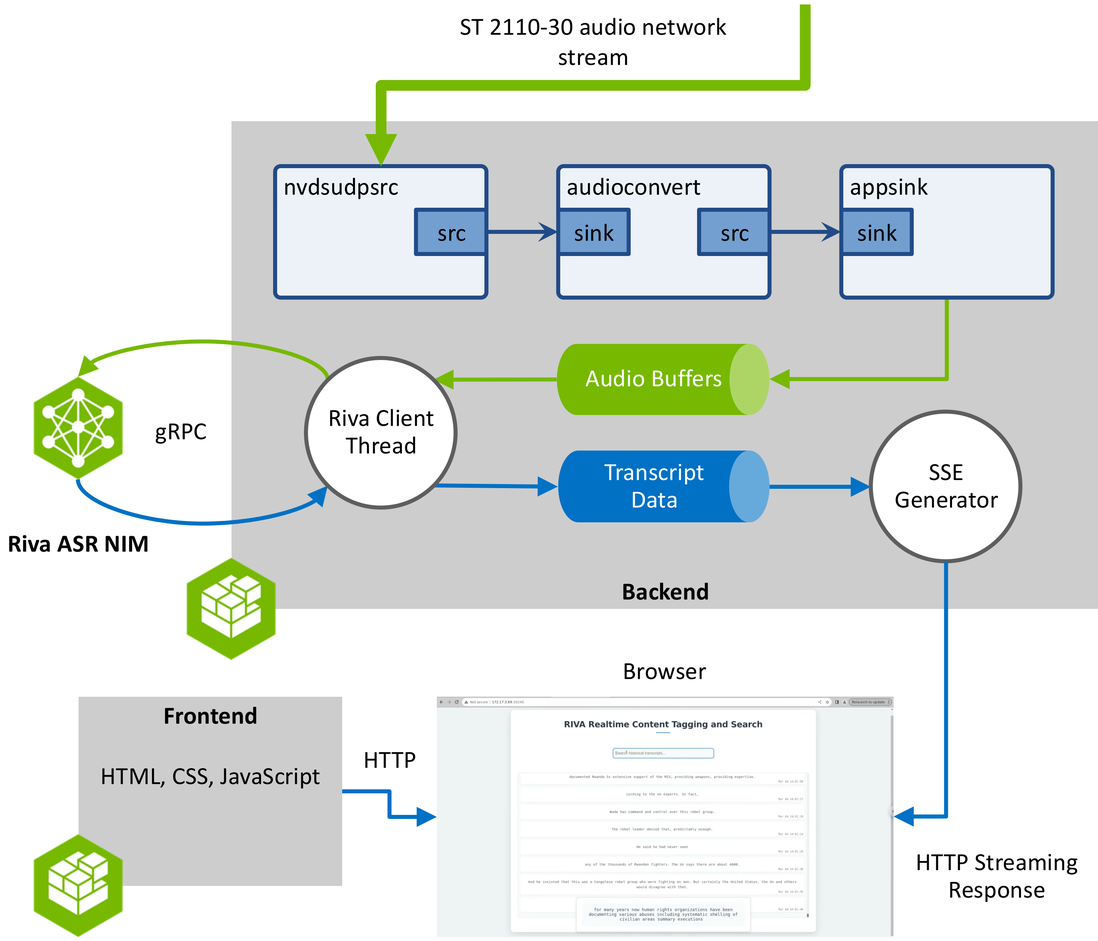


Audio Transcription

AI For Live Media

- Back-end app container - Python
  - **PyGObject** to use GStreamer
  - **Riva Python** client to talk to the NIM
  - **Flask** for server-sent events (SSE)

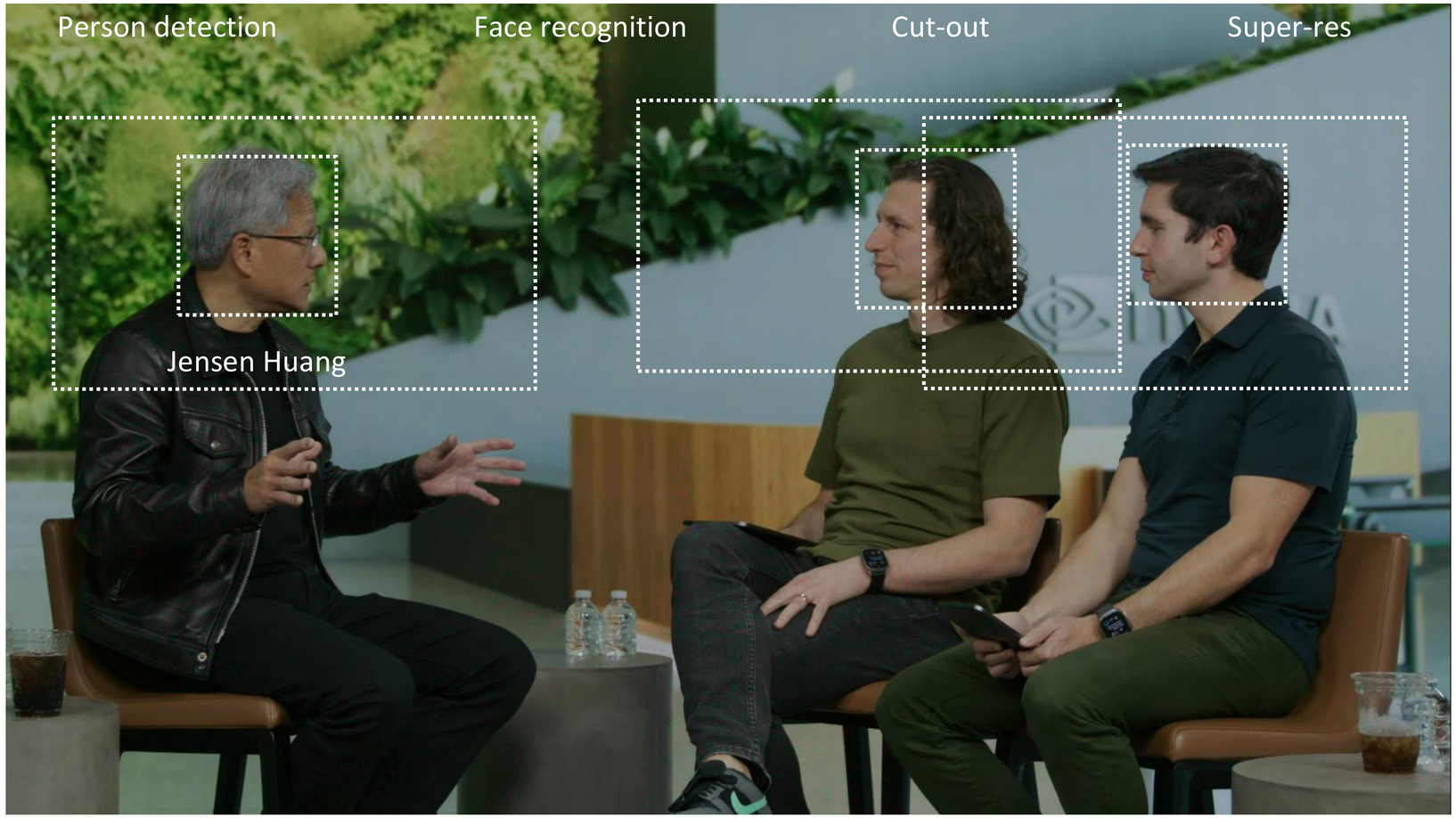
1. Pipeline now has an AppSink that emits “new-sample” callbacks in which we aggregate and enqueue the audio buffers
2. Riva client thread dequeues the audio buffers and sends to the Riva ASR NIM, and enqueues the returned transcript data
3. The server-sent event generator dequeues the transcript data and delivers it to the web UI



AI Virtual Cameras

Person detection, face recognition, super-resolution

Live ST 2110 source – 4k (3840 x 2160)



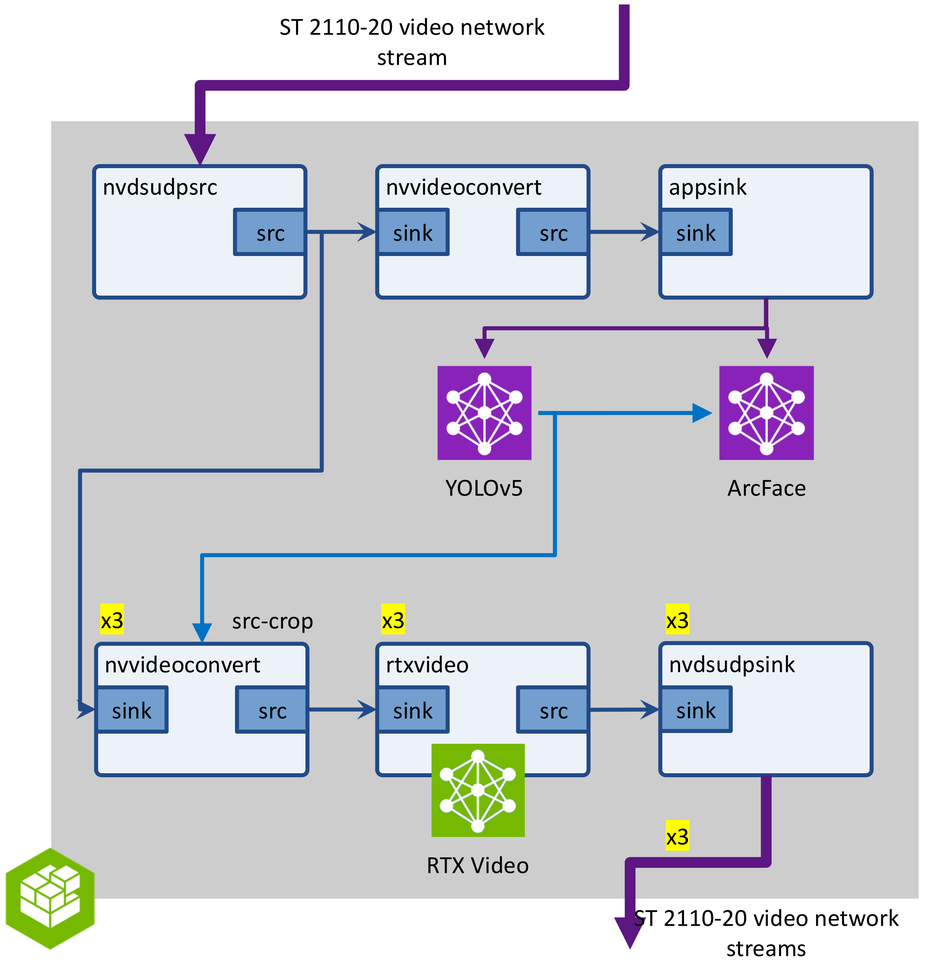
Virtual ST 2110 cameras – HD (1920 x 1080)

NVIDIA.

AI Virtual Cameras

Person detection, face recognition, super-resolution

- Input
  - DeepStream `nvdsudpsrc`
- **Person detection**
  - PyTorch YOLOv5
  - Real-time, 20ms per 4k frame on NVIDIA L40S
- **Face recognition**
  - PyTorch ArcFace + embeddings
  - 2 seconds
- Cut-out
  - DeepStream `nvvideoconvert`
- **Super resolution**
  - RTX Video
  - <3ms per frame
- Output
  - DeepStream `nvdsudpsink`



Get Started

Anything… Anywhere…

Choose your abstraction:

- Bare Metal: Use Rivermax
- Prebuilt framework: Use GStreamer and DeepStream.

Anywhere on any device:

- Intel/AMD x86_64 → Server, workstations and cloud
- ARM aarch64 → Embedded, edge and cloud

For AI ready, container orchestrated system
→ Use Holoscan for Media!

Questions? 😄