Internet System

Multimedia Networking - GStreamer

P. Bakowski

bako@ieee.org

Multimedia networking applications

- streaming stored audio and video
- streaming live audio and video
- real-time interactive audio and video

- delay sensitive
- loss-tolerant

Multimedia networking applications

- streaming stored audio and video

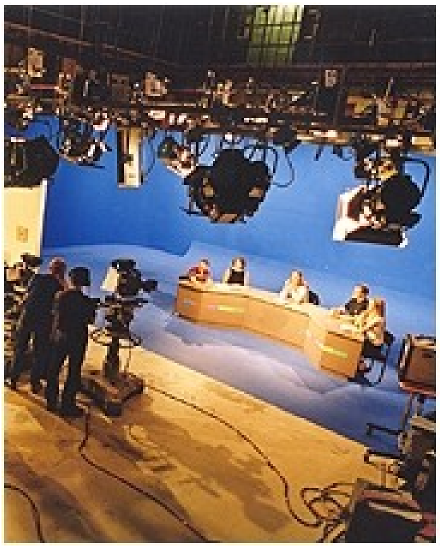
stored media

video server rack

- streaming
- continuous playout

Multimedia networking applications

- streaming live audio and video



- media not stored
- multiple streams
  broadcasting

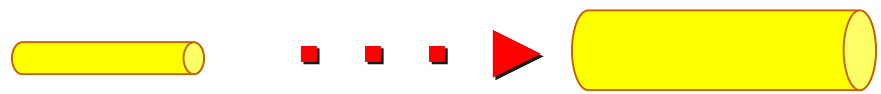
Multimedia networking applications

- real-time interactive audio and video

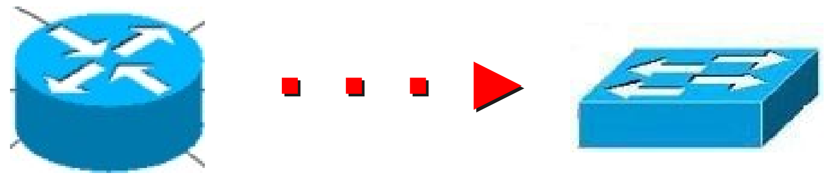
- no stored media
- strong real-time constraints
- multicasting

Internet evolution

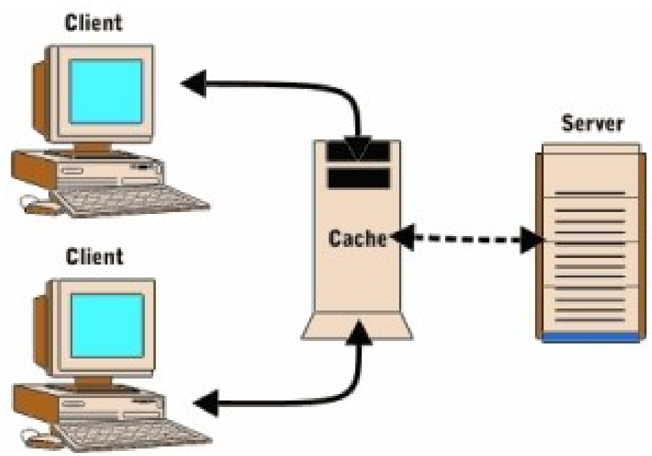
- more bandwidth and switching capacity

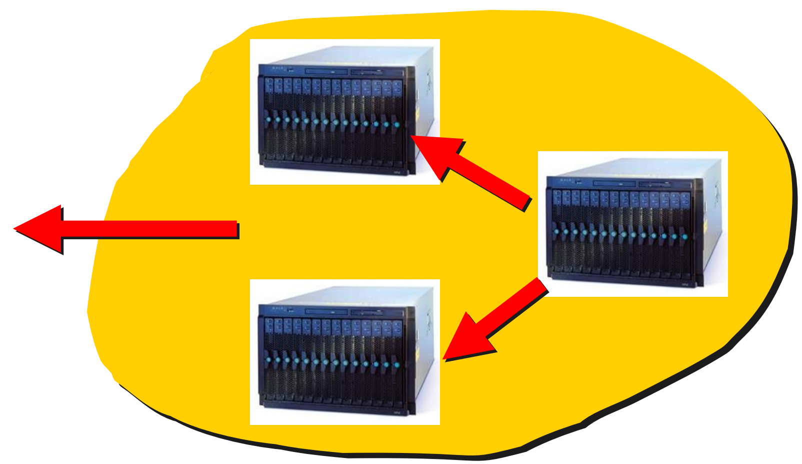


routing



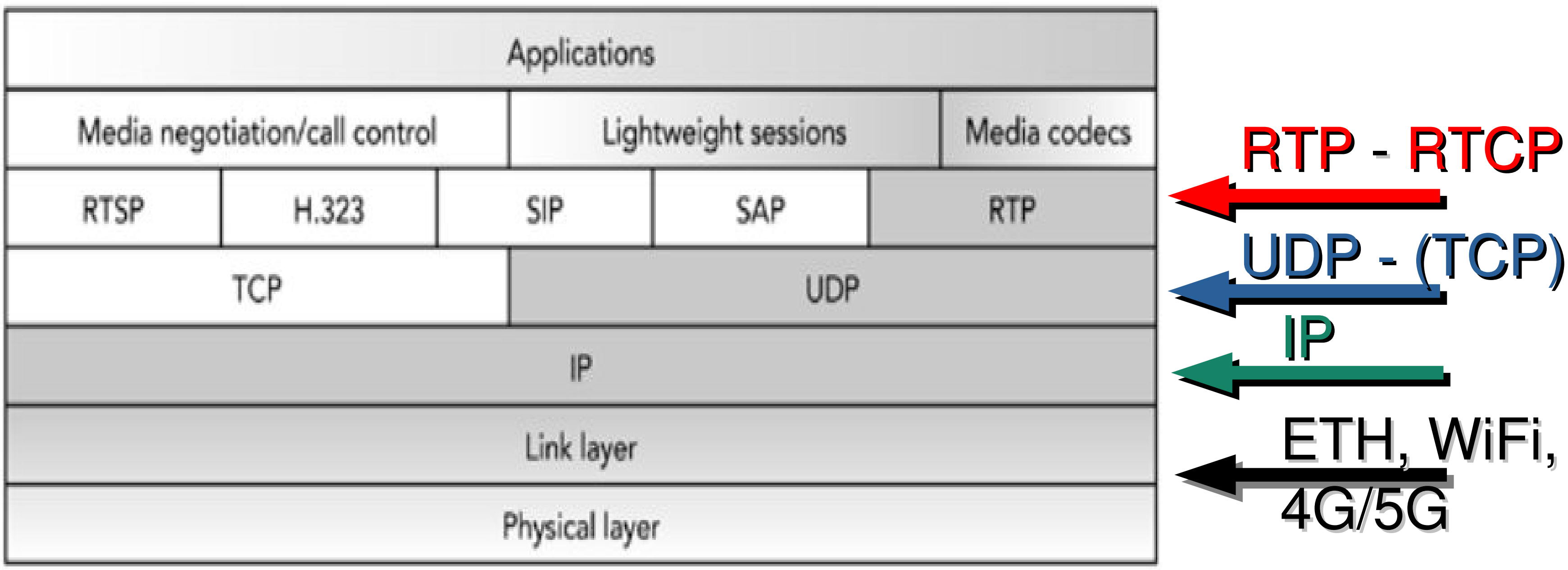
switching



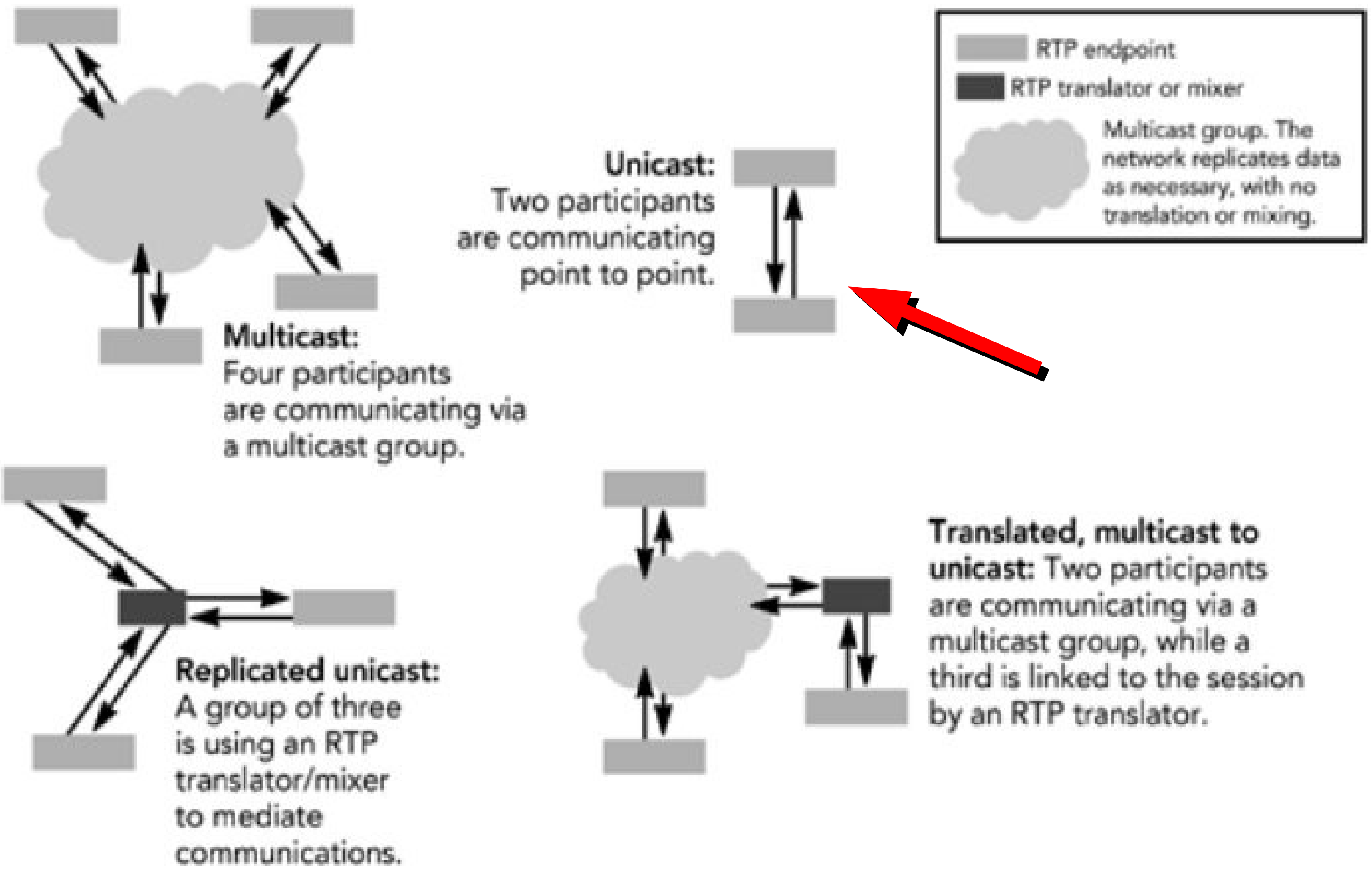


content distribution by replication and caching

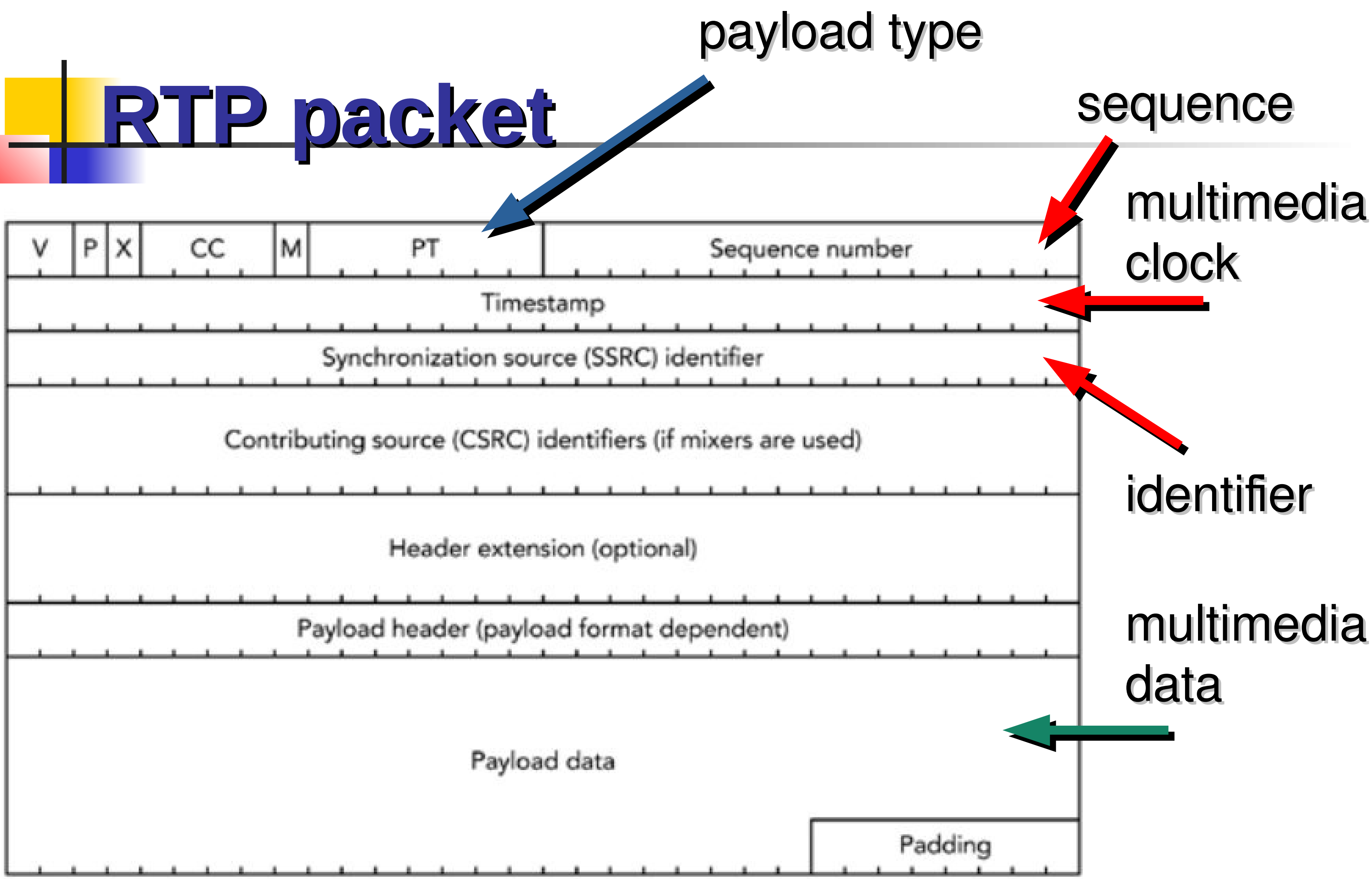
Multimedia protocol stack



RTP session types



RTP packet



V = version number
P = padding
X = extensions
CC = count of contributing sources
M = marker
PT = payload type

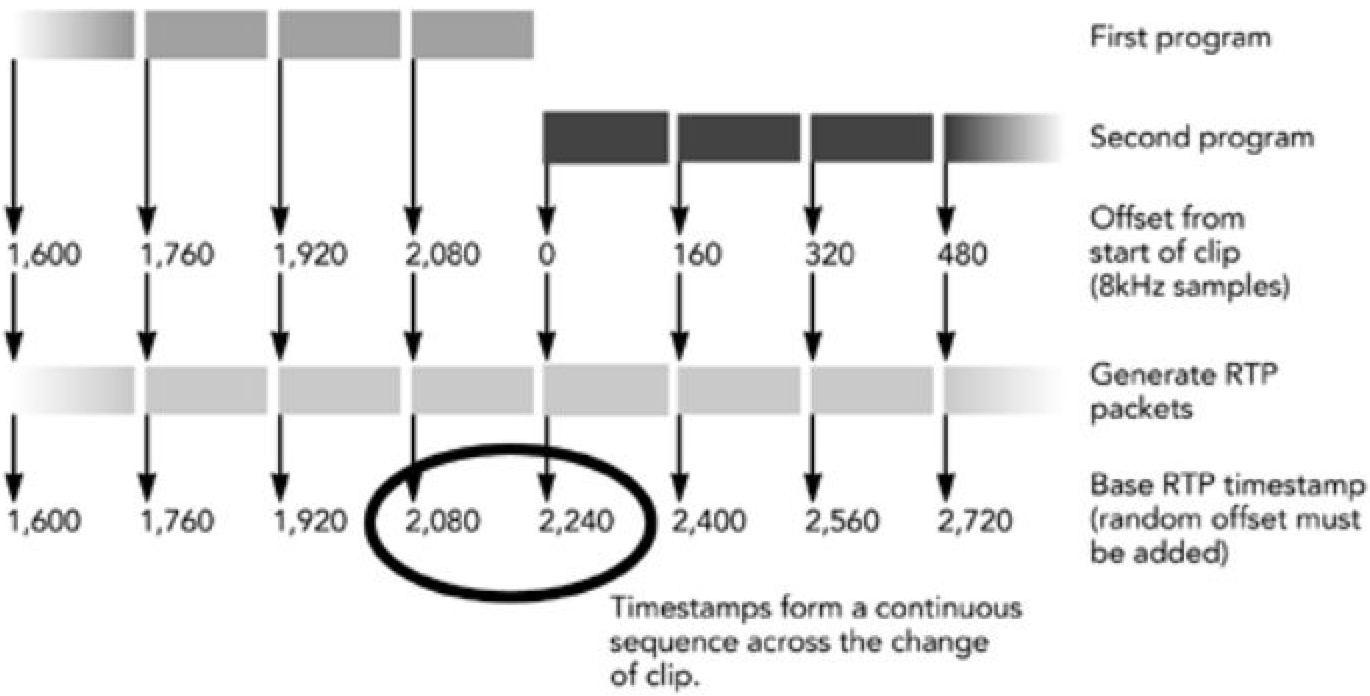
RTP packet : payload type

Payload Type Number	Payload Format	Specification	Description
0	AUDIO/PCMU	RFC 1890	ITU G.711 μ-law audio
3	AUDIO/GSM	RFC 1890	GSM full-rate audio
8	AUDIO/PCMA	RFC 1890	ITU G.711 A-law audio
12	AUDIO/QCELP	RFC 2658	PureVoice QCELP audio
14	AUDIO/MPA	RFC 2250	MPEG audio (e.g., MP3)
26	VIDEO/JPEG	RFC 2435	Motion JPEG video
31	VIDEO/H261	RFC 2032	ITU H.261 video
32	VIDEO/MPV	RFC 2250	MPEG I/II video

96 to 127 – dynamic (includes H264)

RTP packet : timestamp

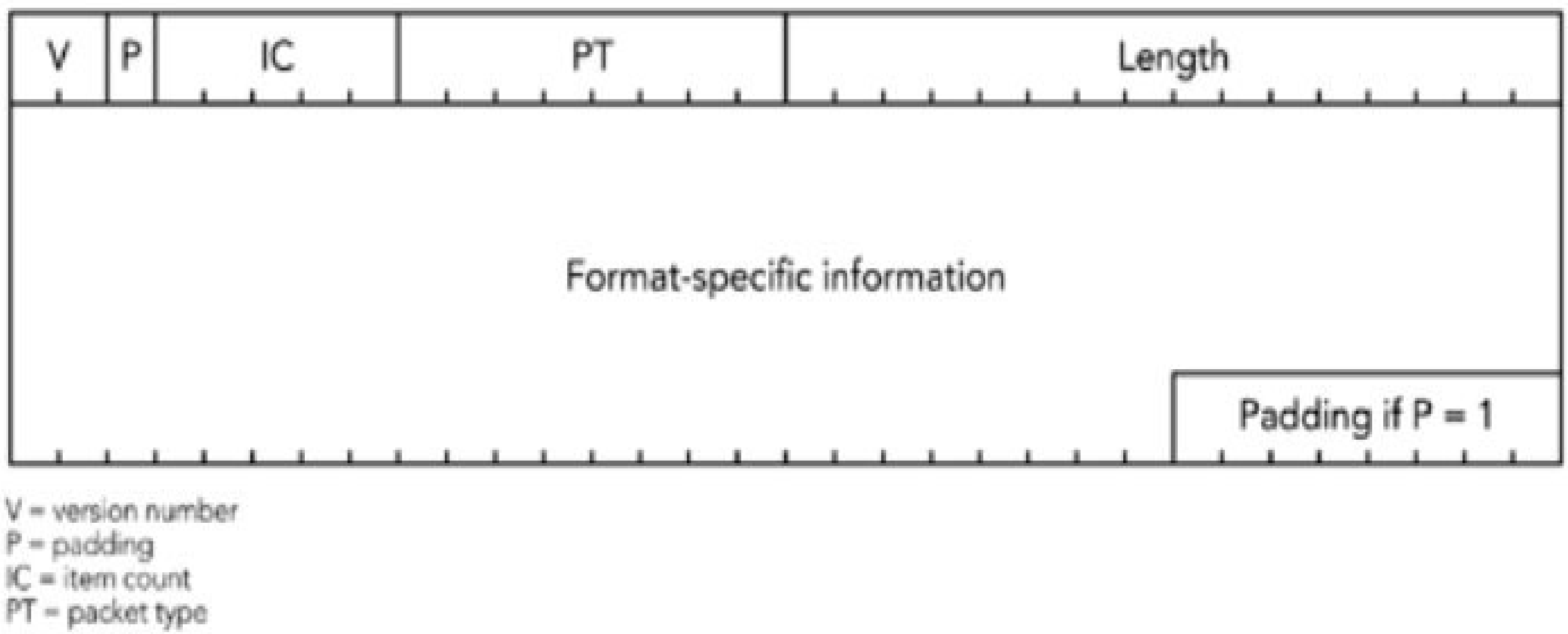
audio



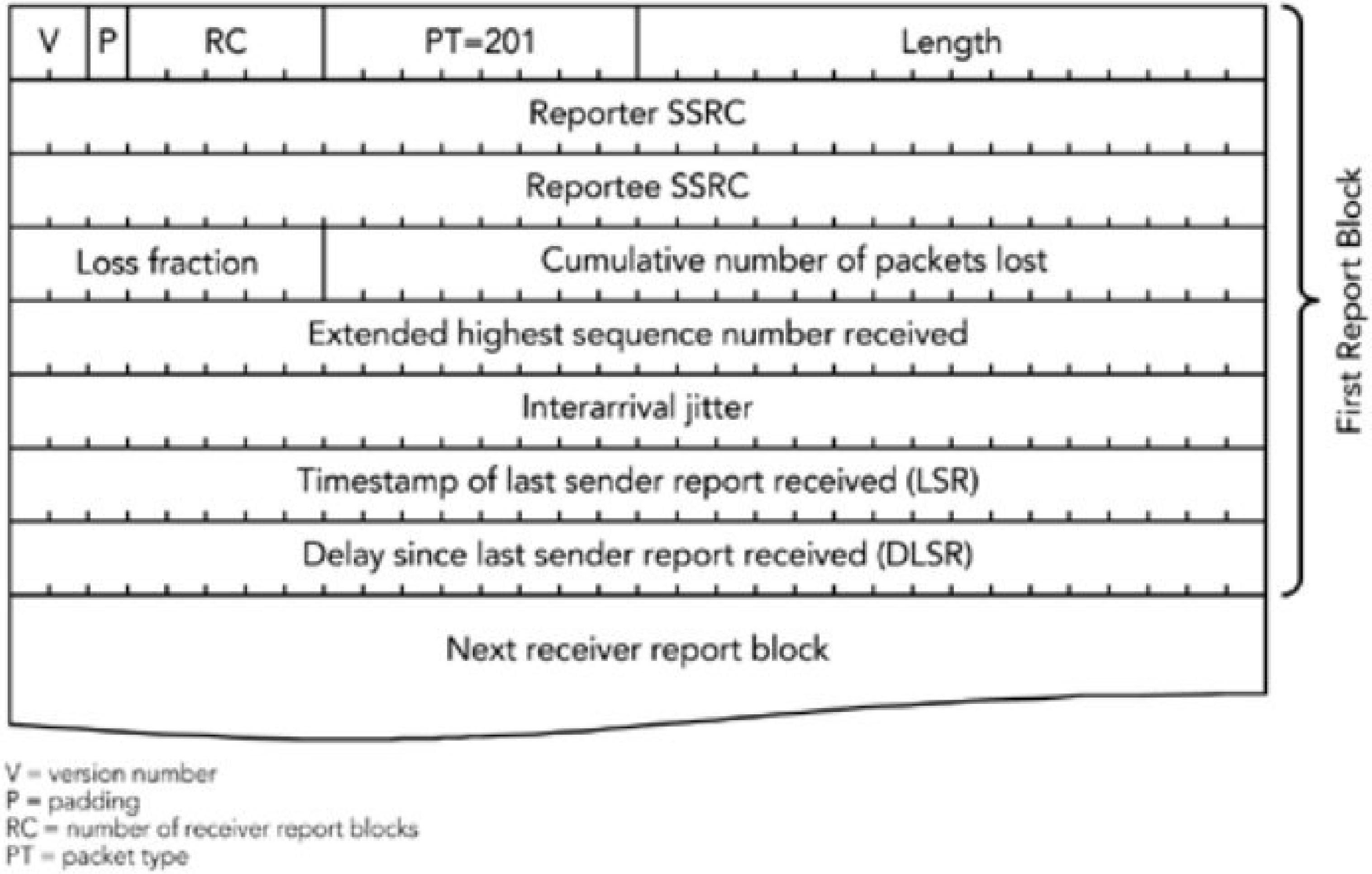
20 ms * 8000 1/s => 160 (timestamp distance)

reference – media clock audio (8kHz), video (96KHz)

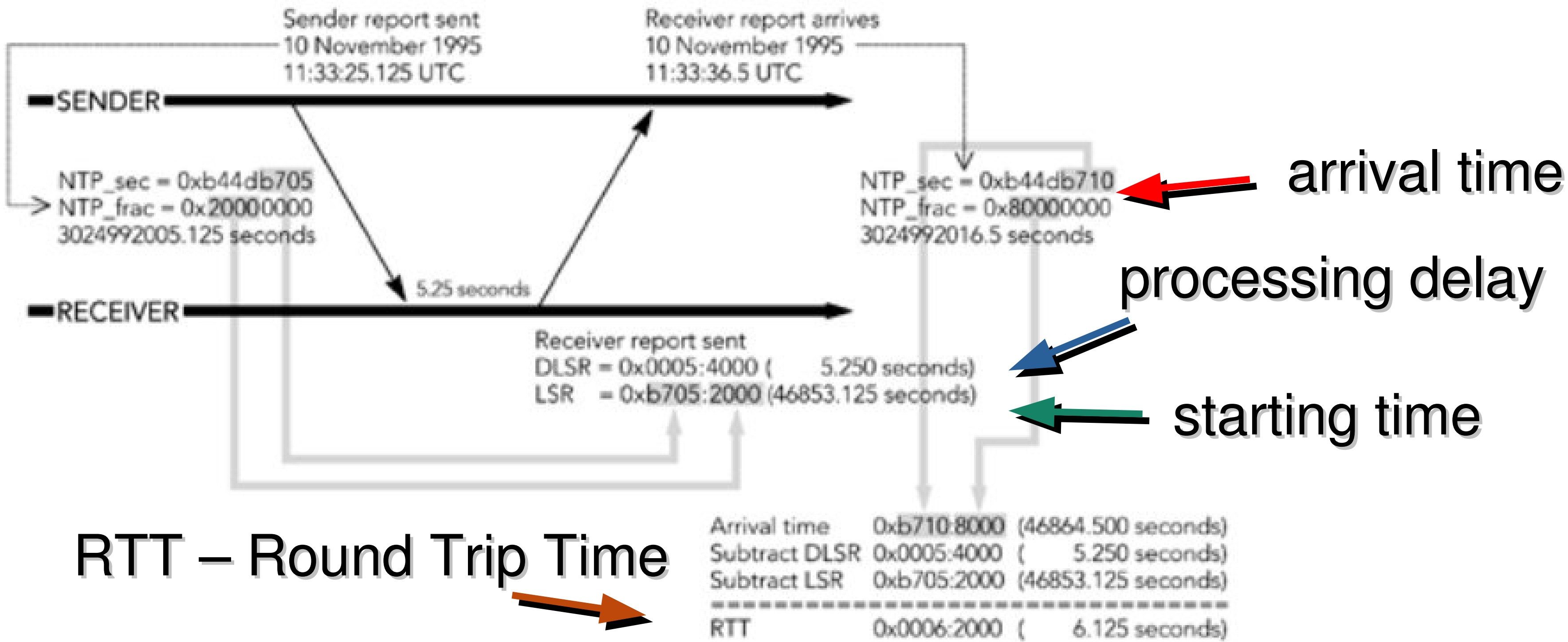
RTCP packet


V
P
IC
PT
Length
Format-specific information
Padding if P = 1
V = version number
P = padding
IC = item count
PT = packet type

RTCP packet: receiver report



RTT time computation



RTT – Round Trip Time

NTP – Network Time Protocol – data on 64 bits

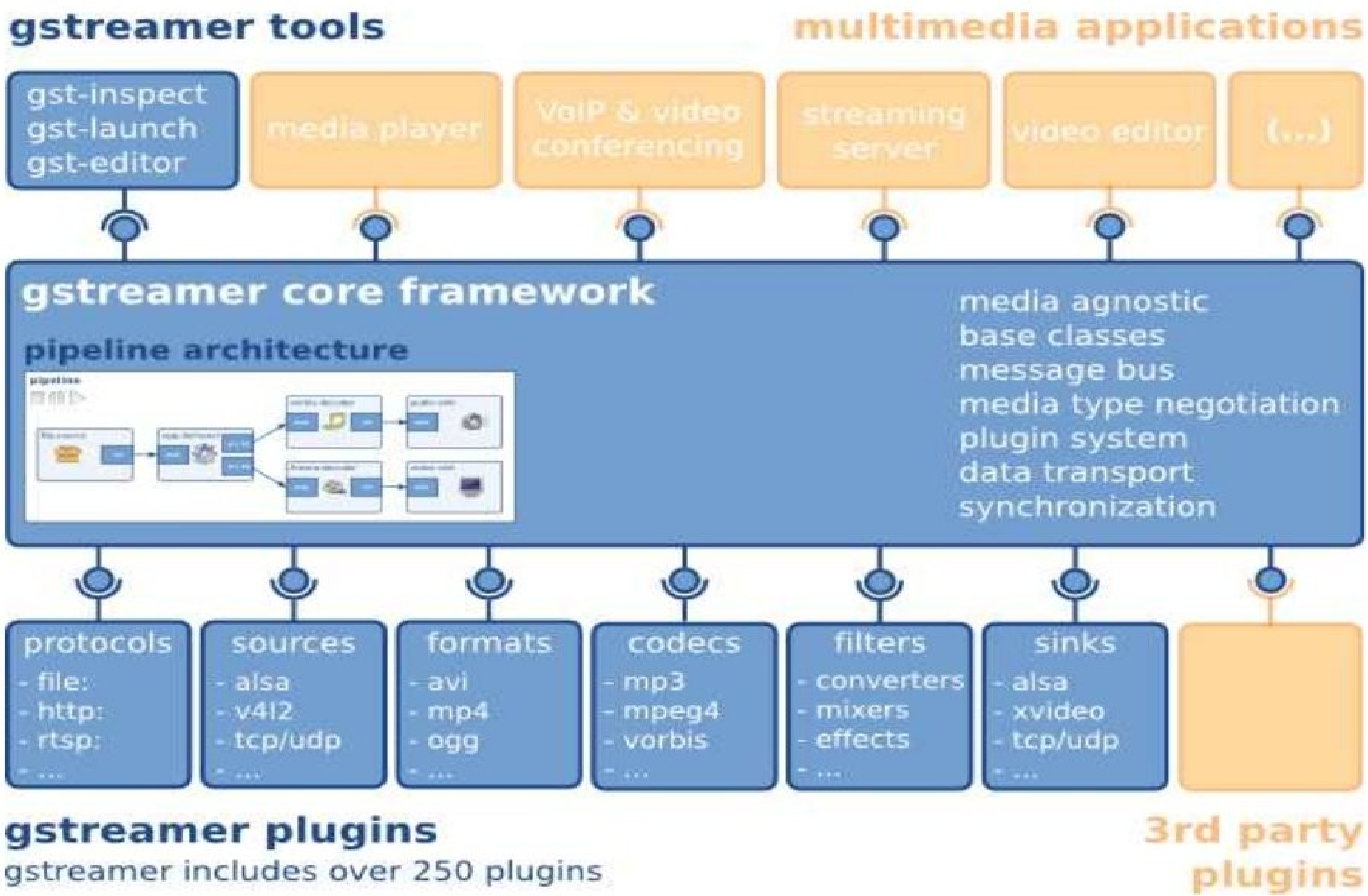
0xb44db710 – seconds:0x80000000 - fraction of second

GStreamer – global overview

tools

core

plugins



GStreamer – elements

There are different types of elements such as:

– source element: the source of stream

– sink element: the destination of stream

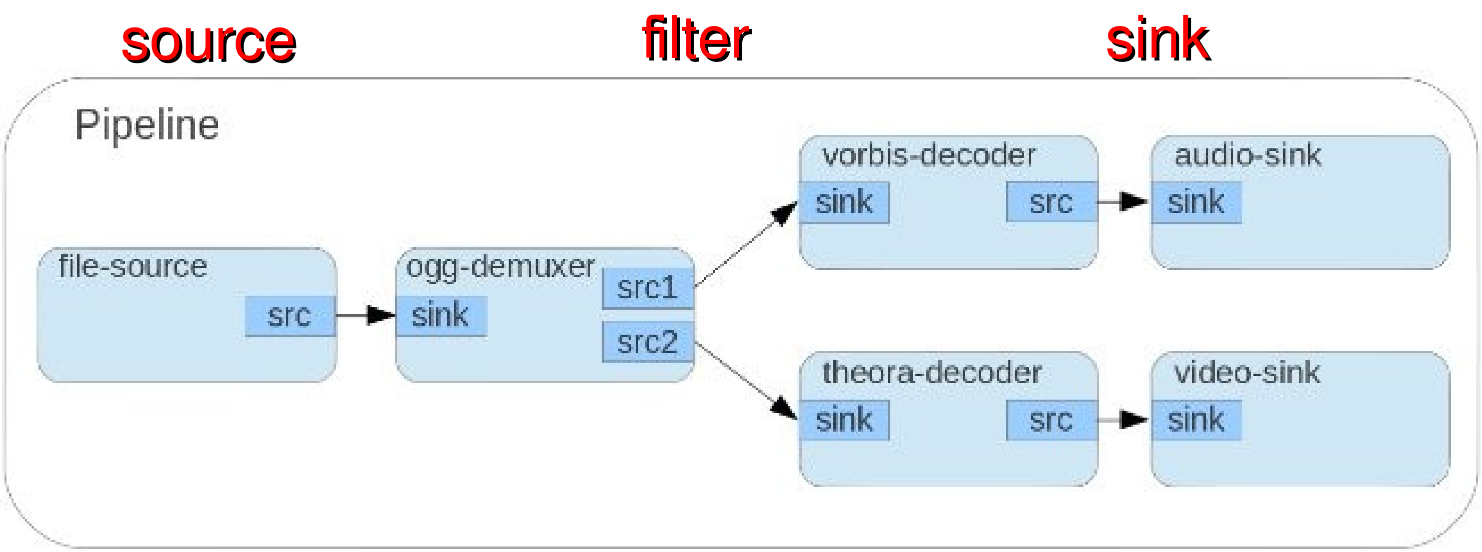
– filter: the manipulator of streams

GStreamer – pipeline

Pipeline is a top level Bin (component).

Each application must have one pipeline.

The following figure demonstrates a pipeline for an ogg player.



Using Gstreamer - `launch`

`gst-launch` also allows the use of multiple threads.

You can use **dots** to imply padnames on elements, or even omit the padname to automatically select a pad.

Using all this, the pipeline (note the use of **`queue`** element):

```
gst-launch filesrc location=file.ogg ! oggdemux
name=d d. ! queue ! theoradec ! videoconvert !
xvimagesink d. ! queue ! vorbisdec !
audioconvert ! audioresample ! alsasink
```

will play an Ogg file containing a Theora video-stream and a Vorbis audio-stream.

Using GStreamer - `inspect`

`gst-inspect` can be used to inspect all properties, signals, dynamic parameters codecs, and the object hierarchy of an element.

```
bako@mezza:~$ gst-inspect-0.10 | grep mp3
ffmpeg:  ffmux_mp3: FFmpeg MPEG audio layer 3 formatter (not recommended, use id3v2mux instead)
ffmpeg:  ffdec_mp3on4float: FFmpeg MP3onMP4 decoder
ffmpeg:  ffdec_mp3on4: FFmpeg MP3onMP4 decoder
ffmpeg:  ffdec_mp3adufloat: FFmpeg ADU (Application Data Unit) MP3 (MPEG audio layer 3) decoder
ffmpeg:  ffdec_mp3adu: FFmpeg ADU (Application Data Unit) MP3 (MPEG audio layer 3) decoder
ffmpeg:  ffdec_mp3float: FFmpeg MP3 (MPEG audio layer 3) decoder
ffmpeg:  ffdec_mp3: FFmpeg MP3 (MPEG audio layer 3) decoder
flump3dec:  flump3dec: Fluendo MP3 Decoder (liboil build)
mpegaudioparse:  mp3parse: MPEG1 Audio Parser
typefindfunctions: audio/mpeg: mp3, mp2, mp1, mpga
typefindfunctions: application/x-apetag: mp3, ape, mpc, wv
typefindfunctions: application/x-id3v1: mp3, mp2, mp1, mpga, ogg, flac, tta
typefindfunctions: application/x-id3v2: mp3, mp2, mp1, mpga, ogg, flac, tta
mad:  mad: mad mp3 decoder
lame:  lame: L.A.M.E. mp3 encoder
lame:  lamemp3enc: L.A.M.E. mp3 encoder
```

Using ChatGPT to build pipelines

`prompt :`

Write GStreamer pipeline to decode and play video audio file h264 and mp3 for example.

```
gst-launch-1.0 filesrc location=yourfile.mp4 !
qtdemux name=demux \
demux.video_0 ! queue ! h264parse ! avdec_h264
! videoconvert ! autovideosink \
demux.audio_0 ! queue ! mp3parse !
mpg123audiodec ! audioconvert ! autoaudiosink
```

Modifications :

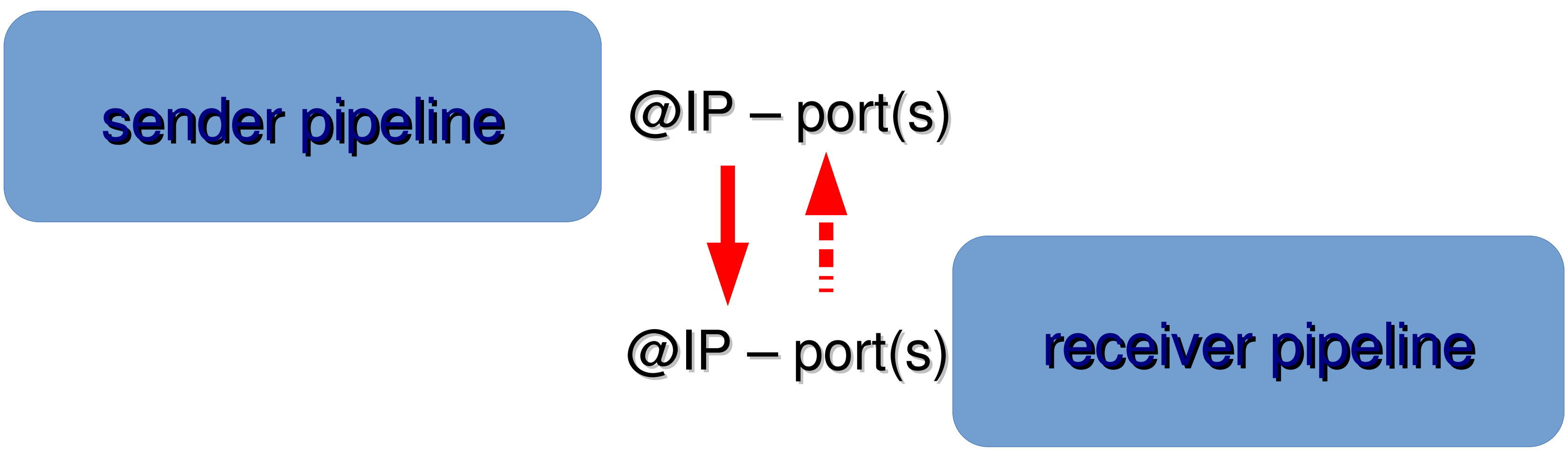
`Mpg123audiodec` should be replaced by `mad`

`autovideosink` by `ximagesink`

GStreamer – streaming

GStreamer allows us to send the multimedia flows over several Internet protocols :

- UDP, (TCP) as simple data flows or
- RTP/UDP and RTP/RTCP/UDP as controlled multimedia flows



Gstreamer – streaming on UDP

```
#config.sh
CLIENT="adresse_IP_client"
PORT_UDP_VIDEO=5000    #flux UDP video
```

```
source config.sh
gst-launch-1.0 v4l2src device=/dev/video1 ! video/x-h264, width=1280, height=720, framerate=30/1 ! udpsink host=$CLIENT port=$PORT_UDP_VIDEO
```

```
source config.sh
gst-launch-1.0 udpsrc port=$PORT_UDP_VIDEO ! queue ! h264parse ! openh264dec ! videoconvert ! xvimagesink
```

sender pipeline

udpsink

@IP – port

@IP – port

udpsrc

receiver pipeline

Streaming video on RTP/UDP

```
CLIENT_IP="192.168.1.62"
gst-launch-1.0 -v v4l2src device=/dev/video1 ! video/x-
h264, width=640,height=480,framerate=30/1 ! \
h264parse ! rtph264pay mtu=1400 ! udpsink
host=$CLIENT_IP port=8050 sync=false async=false
```

```
gst-launch-1.0 udpsrc port=8050 caps="application/x-rtp,
media=(string)video, clock-rate=(int)90000, encoding-
name=(string)H264, encoding-params=(string)1" ! \
rtph264depay ! h264parse ! avdec_h264 ! queue !
videoconvert ! xvimagesink
```

payload specifications (PT) - capacities

Streaming audio on RTP/UDP

```
gst-launch-1.0 -v alsasrc device="default" !
audioconvert ! audioresample ! audio/x-
raw,rate=16000,channels=2 ! opusenc bitrate=64000 !
rtpopuspay pt=97 ! udpsink host=127.0.0.1 port=5002
```

```
gst-launch-1.0 -v udpsrc port=5002 caps="application/x-
rtp, media=audio, encoding-name=OPUS, payload=97" !
rtpjitterbuffer ! rtpopusdepay ! opusdec !
audioconvert ! autoaudiosink
```

payload specifications (PT) - capacities

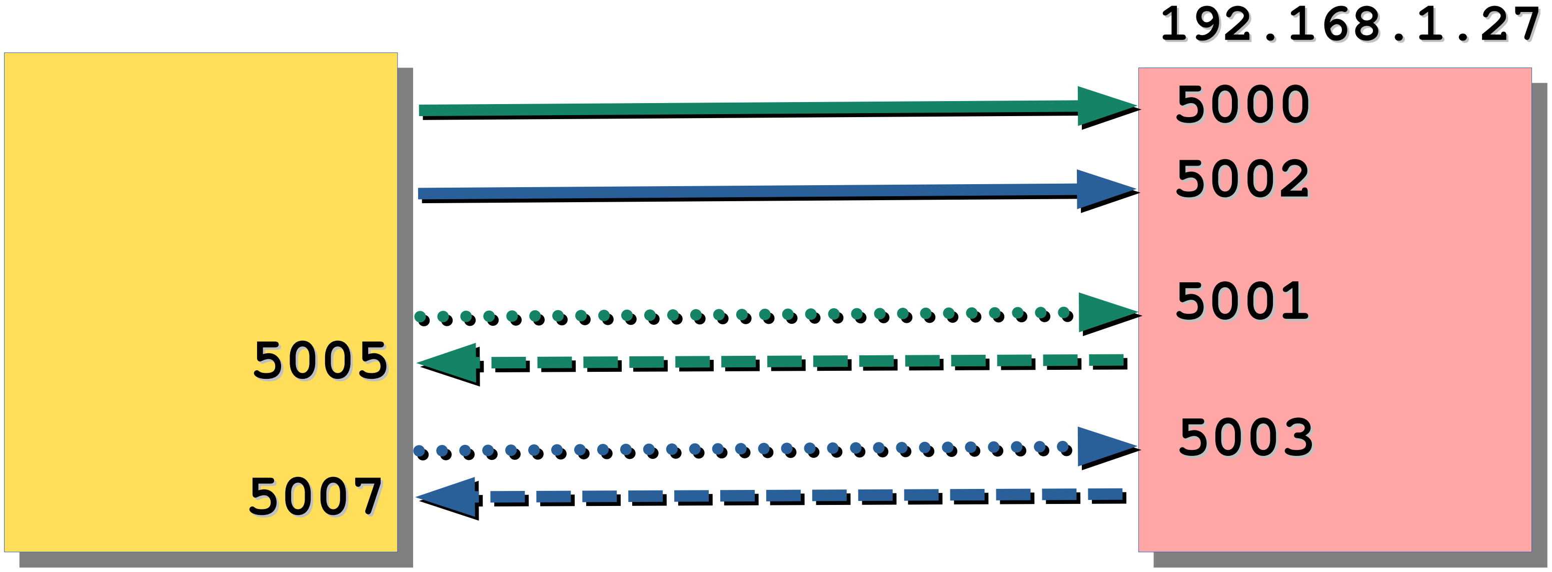
Streaming video-audio on RTP/UDP

```
gst-launch-1.0 -v v4l2src device=/dev/video0 ! video/x-h264,
height=480, width=640 framerate=30/1 ! h264parse ! rtph264pay
config-interval=1 pt=96 ! udpsink host=127.0.0.1 port=5000 alsasrc
device=”default” ! audioconvert ! audioresample ! audio/x-
raw,rate=16000,channels=2 ! opusenc bitrate=64000 ! rtpopuspay
pt=97 ! udpsink host=127.0.0.1 port=5002
```

```
gst-launch-1.0 -v \ udpsrc port=5000 caps="application/x-rtp,
media=video, encoding-name=H264, payload=96" ! rtpjitterbuffer !
rtph264depay ! avdec_h264 ! videoconvert ! ximagesink \ udpsrc
port=5002 caps="application/x-rtp, media=audio, encoding-
name=OPUS, payload=97" ! rtpjitterbuffer ! rtpopusdepay !
opusdec ! audioconvert ! autoaudiosink
```

Streaming video-audio on RTP-RTCP/UDP

```
# config.sh
CLIENT=192.168.1.27
PORT_RTP_VIDEO=5000
PORT_RTCP_VIDEO=5001
PORT_RTP_AUDIO=5002
PORT_RTCP_AUDIO=5003
PORT_RTCP_VIDEO_RET=5005
PORT_RTCP_AUDIO_RET=5007
```



Streaming video-audio on RTP-RTCP/UDP

sender

```
source config.sh

gst-launch-1.0 rtpbin name=rtpbin \
v4l2src device=/dev/video1 ! video/x-h264, width=1920, height=1080,
framerate=30/1 ! rtph264pay ! rtpbin.send_rtp_sink_0 \
rtpbin.send_rtp_src_0 ! udpsink host=192.168.1.27 port=$PORT_RTP_VIDEO \
rtpbin.send_rtcp_src_0 ! udpsink host=192.168.1.27 port=$PORT_RTCP_VIDEO \
sync=false async=false \
udpsrc port=$PORT_RTCP_VIDEO_RET ! rtpbin.recv_rtcp_sink_0 \

alsasrc device="default" ! queue ! audioconvert ! audioresample ! audio/x-raw,
rate=16000, width=16, channels=1 ! speexenc ! rtpspeexpay !
rtpbin.send_rtp_sink_1 \
rtpbin.send_rtp_src_1 ! udpsink host=192.168.1.27 port=$PORT_RTP_AUDIO \
rtpbin.send_rtcp_src_1 ! udpsink host=192.168.1.27 port=$PORT_RTCP_AUDIO \
sync=false async=false \
udpsrc port=$PORT_RTCP_AUDIO_RET ! rtpbin.recv_rtcp_sink_1
```

Streaming video-audio on RTP-RTCP/UDP

receiver

```
source config.sh

gst-launch-1.0 -v rtpbin name=rtpbin  \
     udpsrc caps="application/x-rtp, media=(string)video,clock-rate=(int)90000,
encoding-name=(string)H264, encoding-params=(string)1" \
     port=$PORT_RTP_VIDEO ! rtpbin.recv_rtp_sink_0  \
     rtpbin. ! queue ! rtph264depay ! h264parse ! avdec_h264 ! queue ! videoconvert !
xvimagesink \
      udpsrc port=$PORT_RTCP_VIDEO ! rtpbin.recv_rtcp_sink_0   \
      rtpbin.send_rtcp_src_0 ! udpsink port=$PORT_RTCP_VIDEO_RET sync=false
async=false   \

      udpsrc caps="application/x-rtp,media=(string)audio,clock-rate=(int)16000,
encoding-name=(string)SPEEX, encoding-params=(string)1, payload=(int)110" \
      port=$PORT_RTP_AUDIO ! rtpbin.recv_rtp_sink_1  \
      rtpbin. !  queue ! rtpspeexdepay ! decodebin ! audioconvert ! alsasink \
      udpsrc port=$PORT_RTCP_AUDIO ! rtpbin.recv_rtcp_sink_1  \
      rtpbin.send_rtcp_src_1 ! udpsink port=$PORT_RTCP_AUDIO_RET sync=false
async=false
```

Summary

- Multimedia protocol stack
- RTP and RTCP protocols
- GStreamer framework and components
- GStreamer pipelines – examples

Lab2 : testing GStreamer examples without or with **ChatGPT** assistance.

Summary

- Multimedia protocol stack
- RTP and RTCP protocols
-

Summary

- Multimedia protocol stack
- RTP and RTCP protocols
-