VideoTech conference

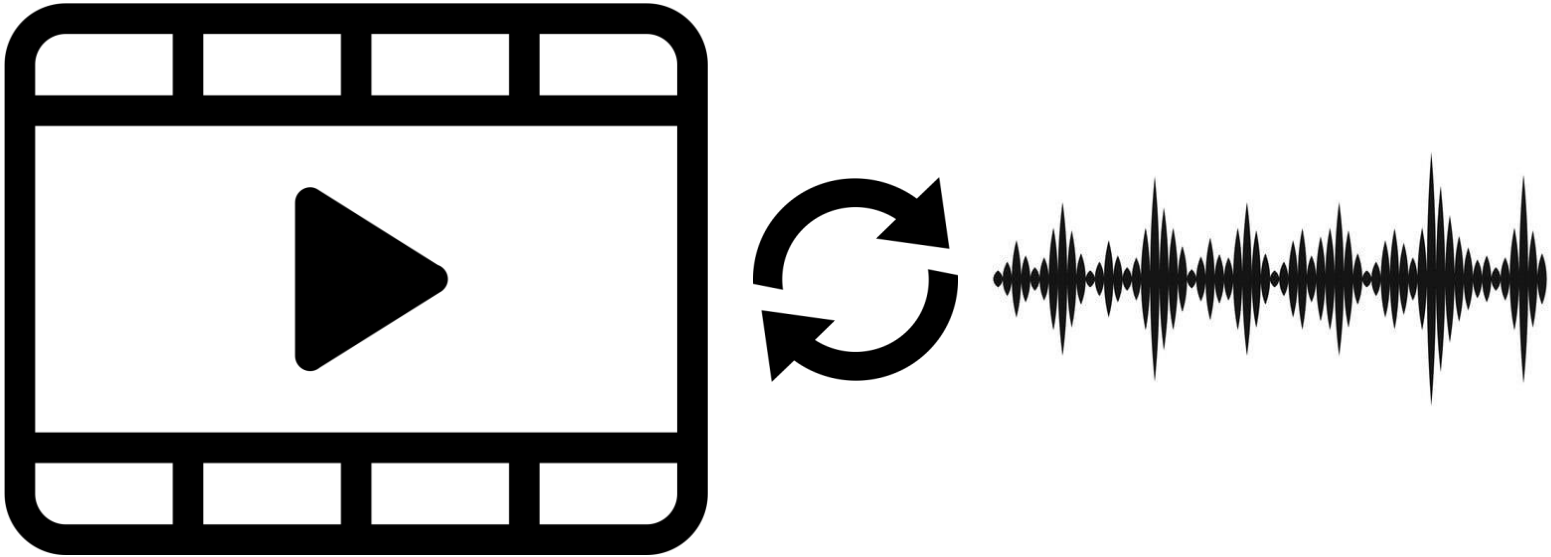
A/V Sync in Android

Oct, 2022

Orion Innovation℠
formerly MERA

ANDROID
open source project

A/V Sync = Audio and Video Synchronized



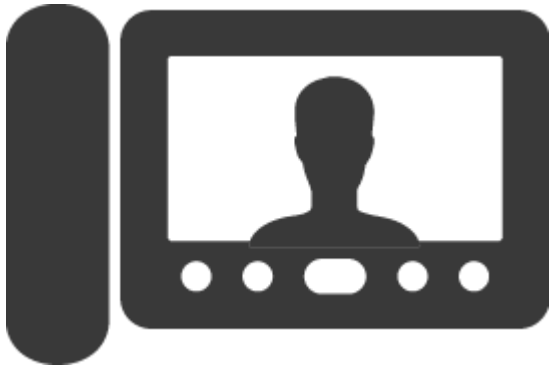
Human Sensitivity

Skew	Effect
20ms	Not visible
50ms	Something’s wrong
1000ms	Video is ignored

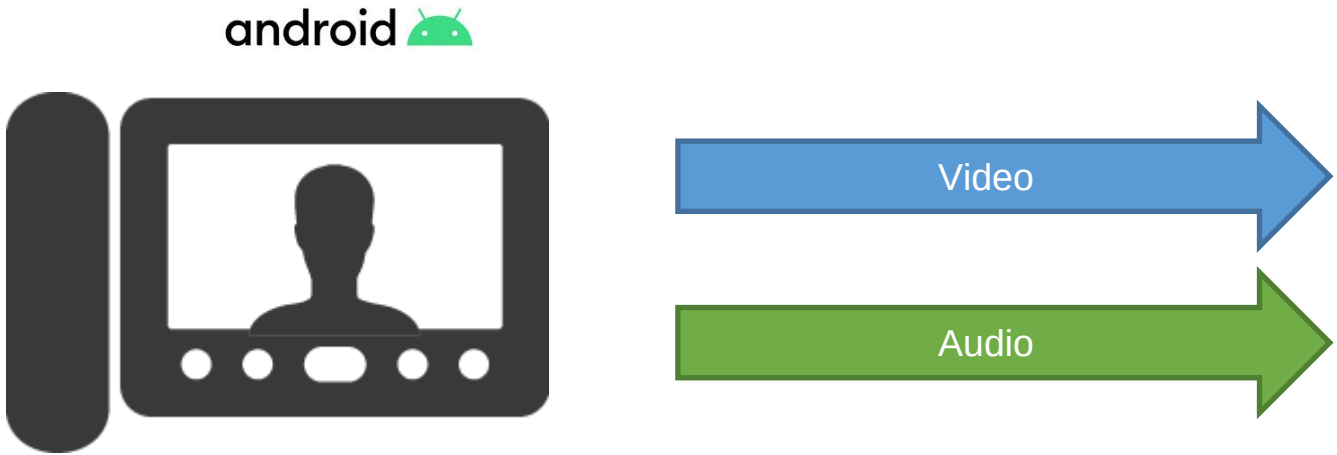
Customer Request

Task: create Android Videophone

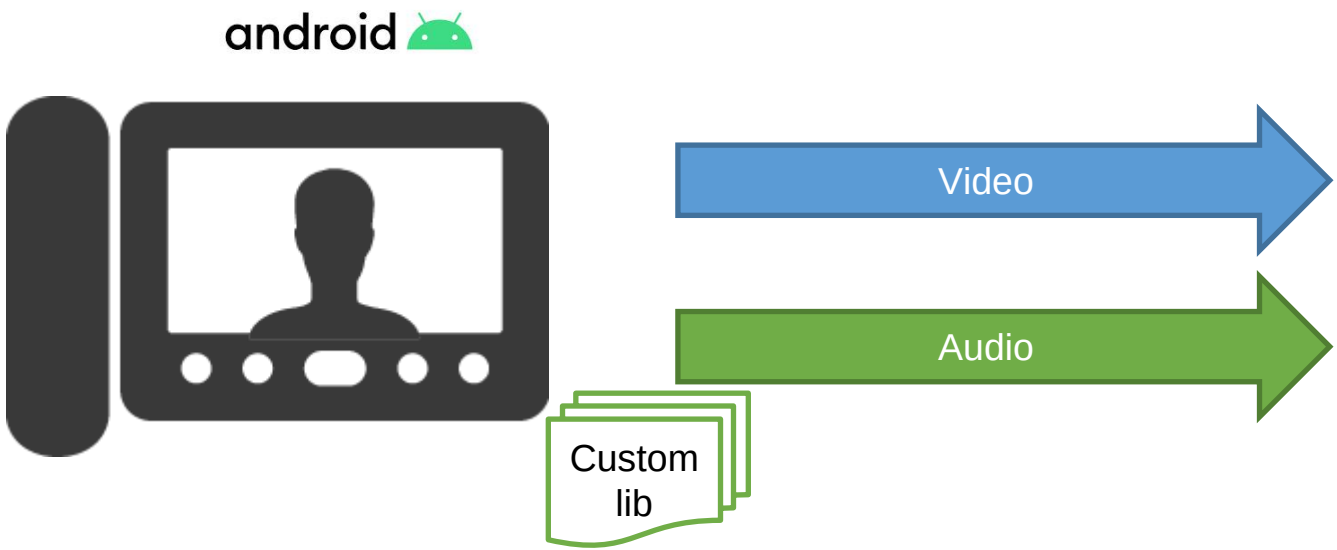
android



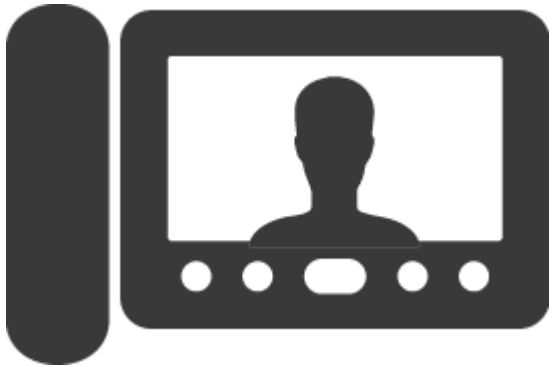
Task: create Android Videophone



Task: create Android Videophone



Task: create Android Videophone



android

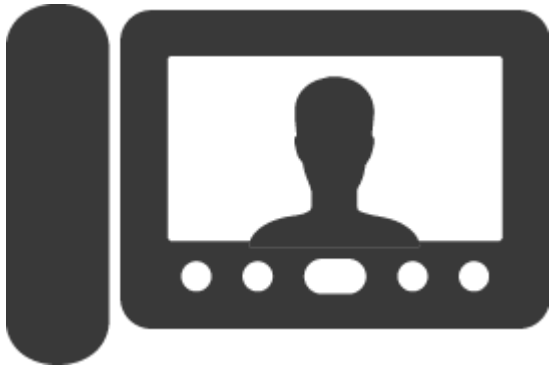
Video

Audio

Custom service

Linux

Task: create Android Videophone



android

Orion Innovation formerly MERA

Video

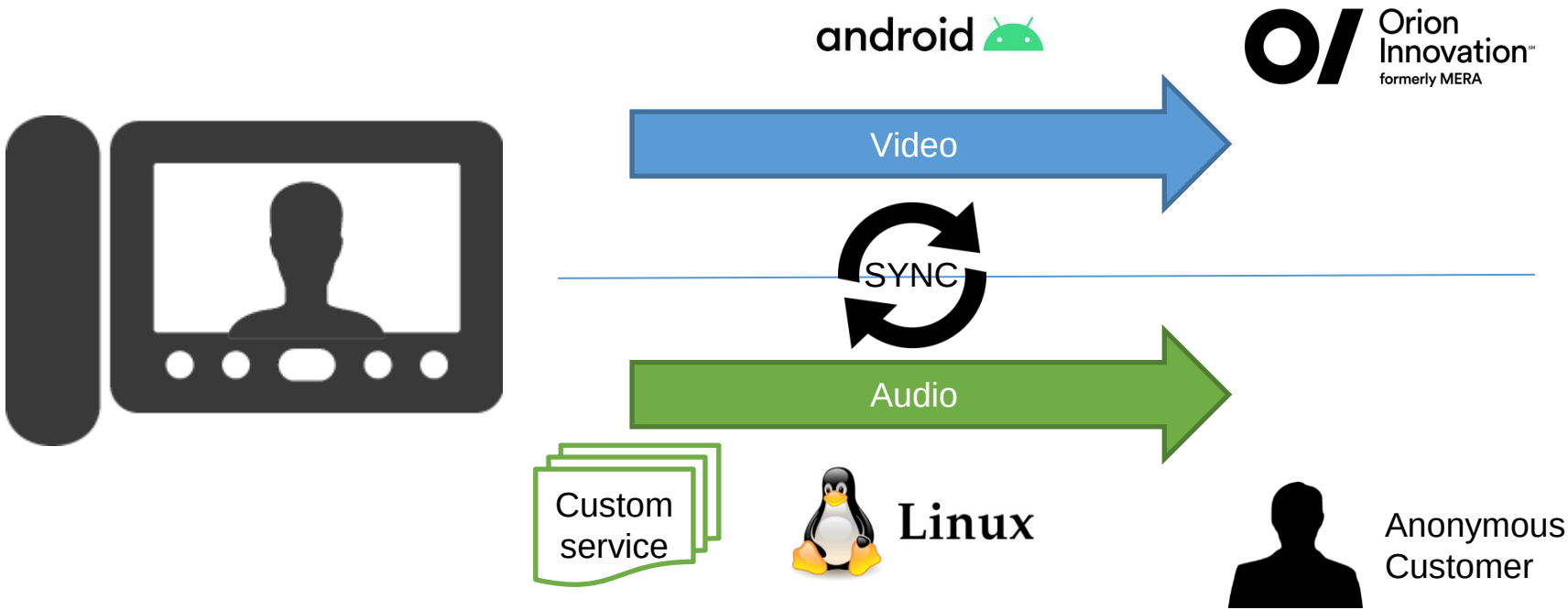
Audio

Custom service

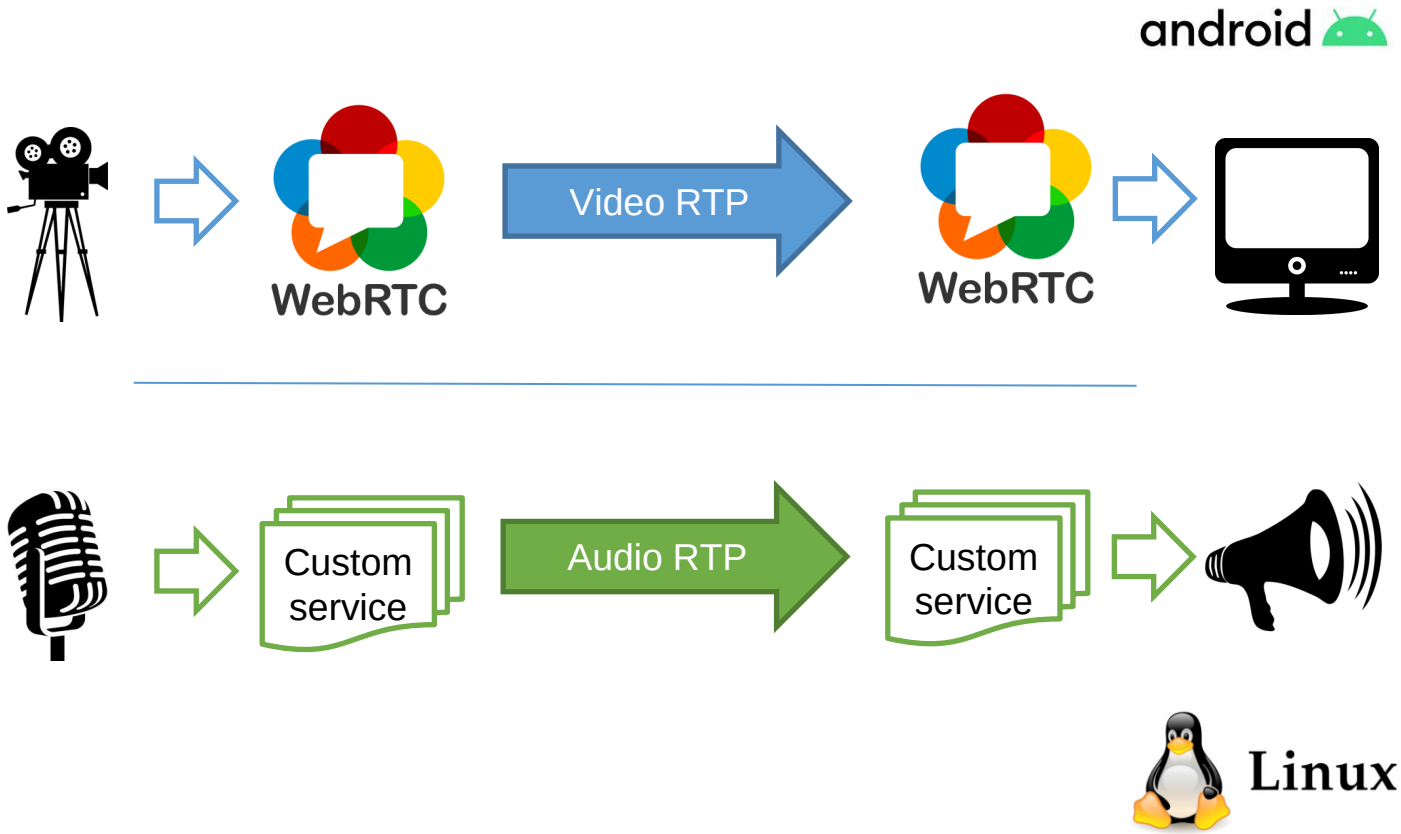
Linux

Anonymous Customer

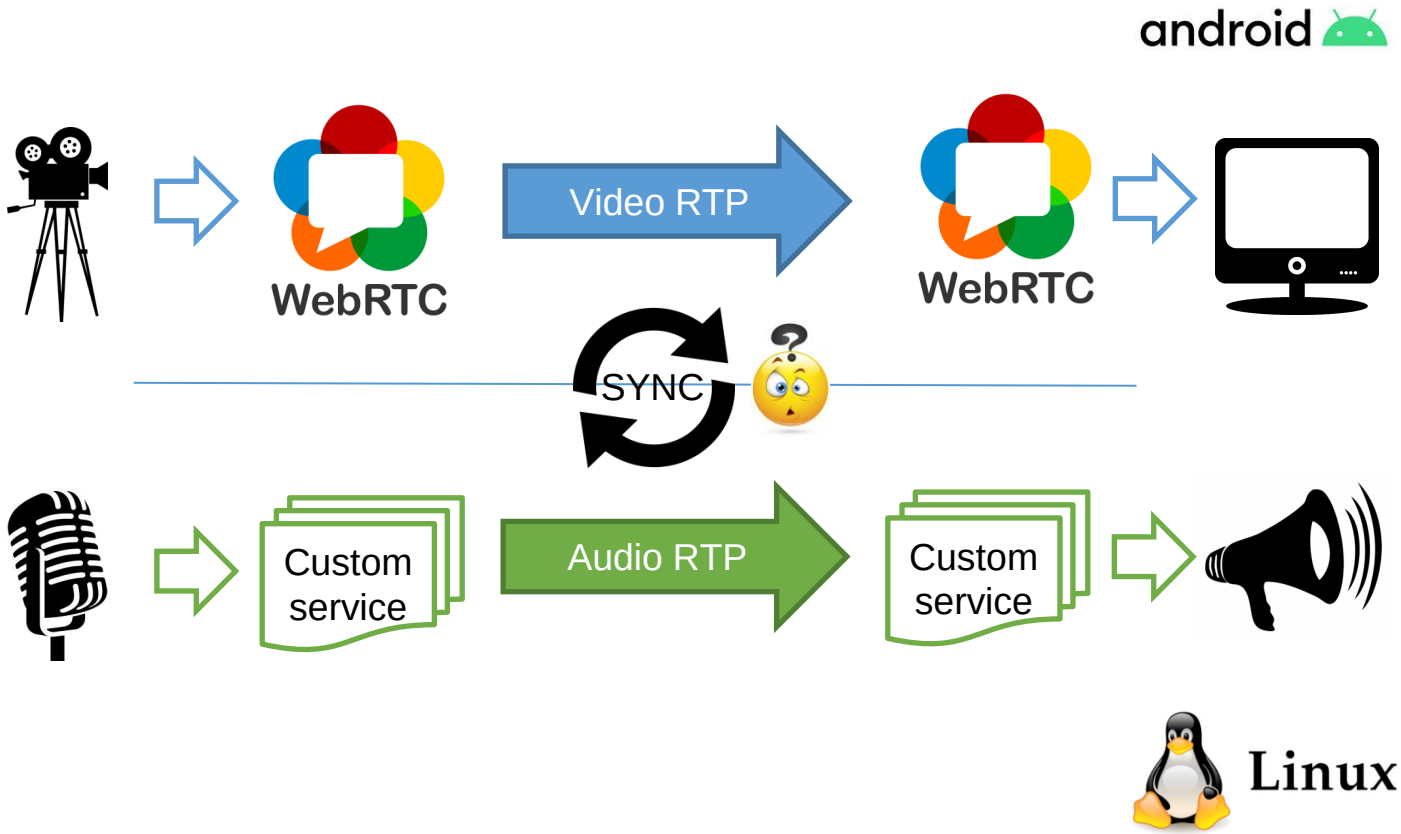
Task: create Android Videophone



Architecture



Architecture



How A/V sync works?

How A/V sync works?

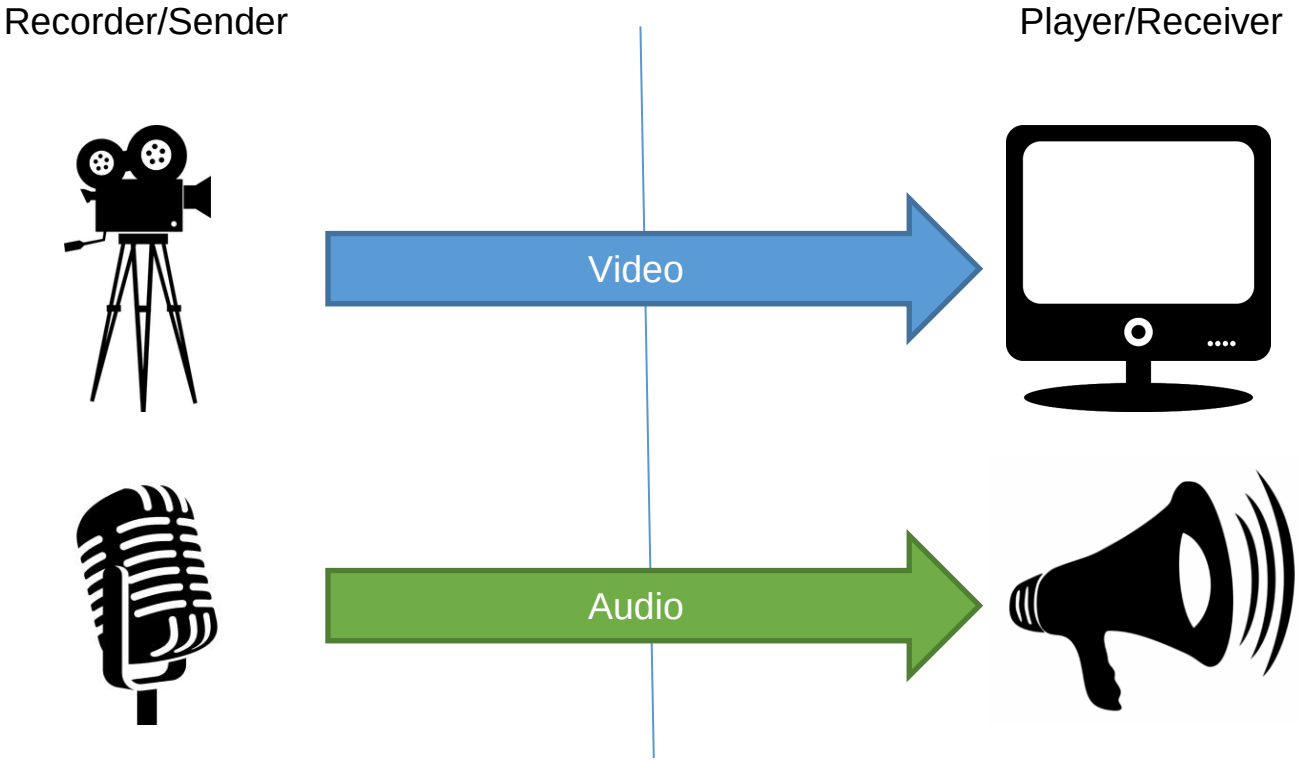
How A/V sync works?

How A/V sync works?

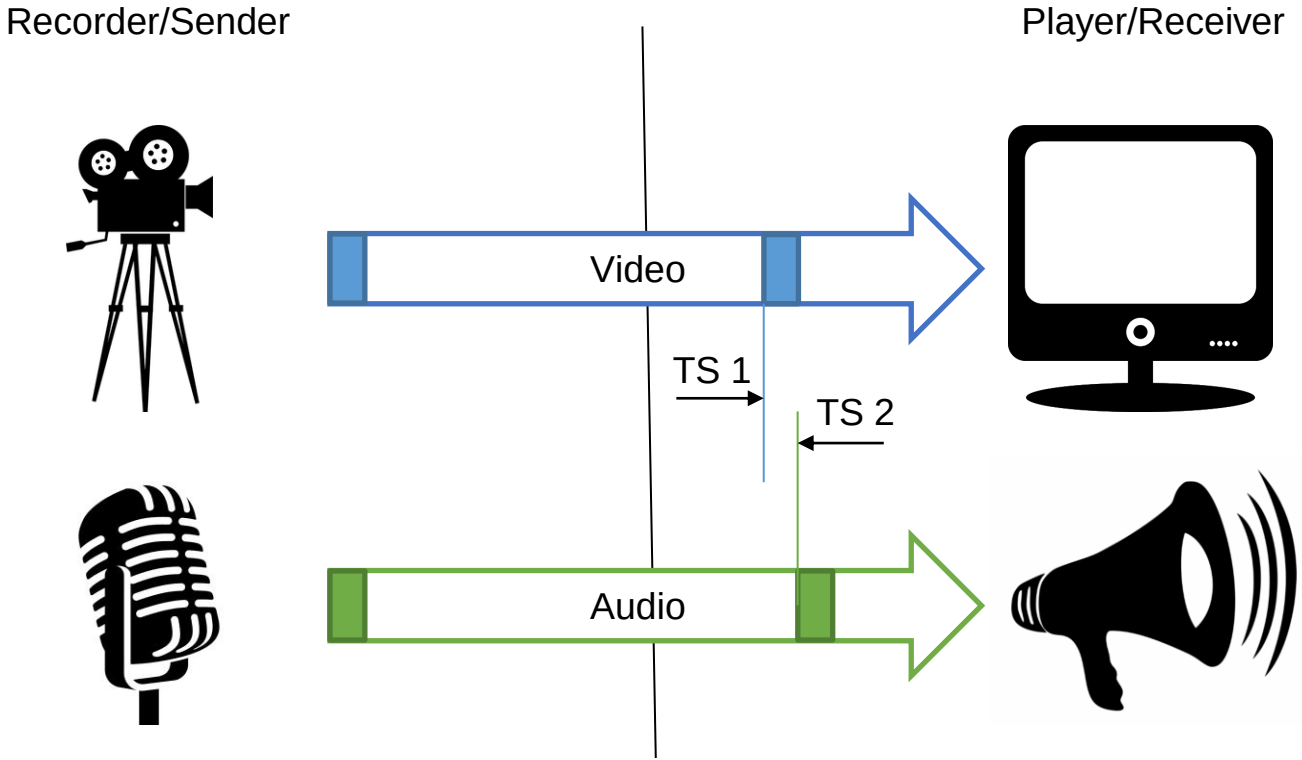
Timestamps

Theory

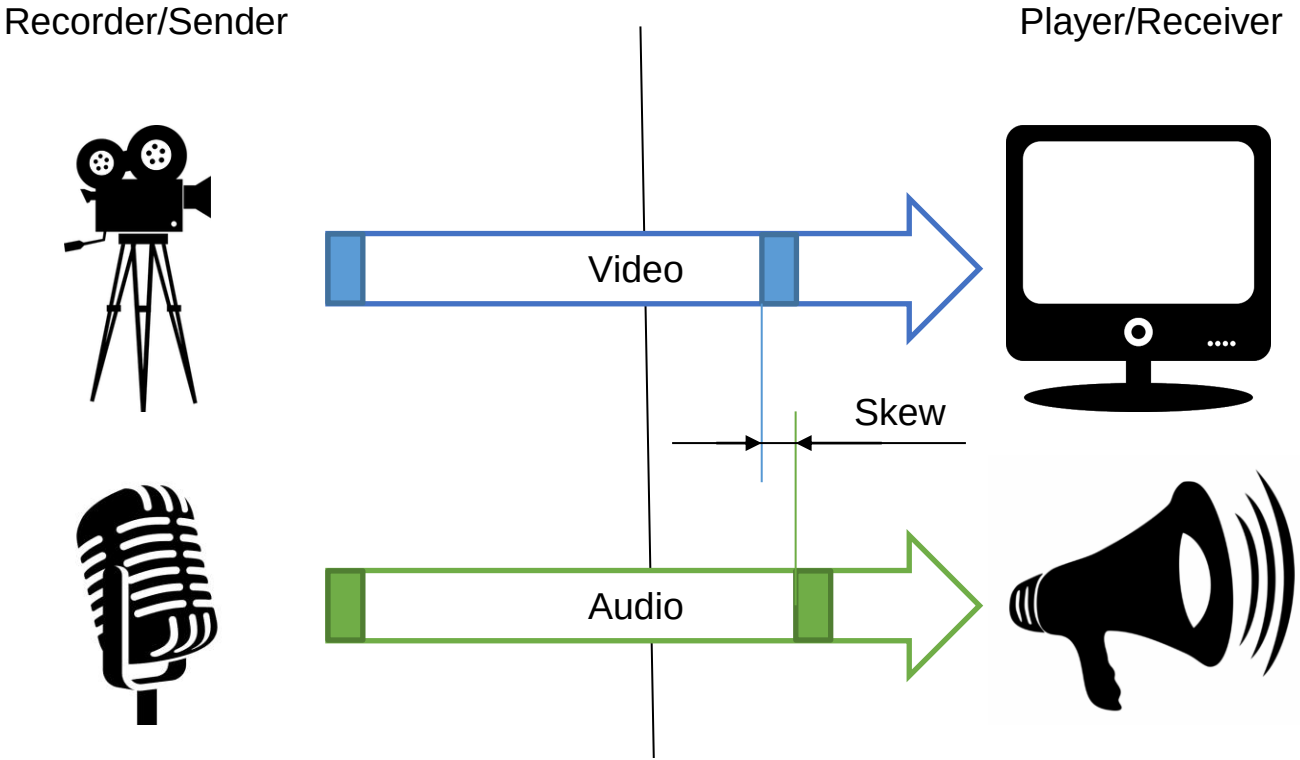
Two Streams



Time Stamps

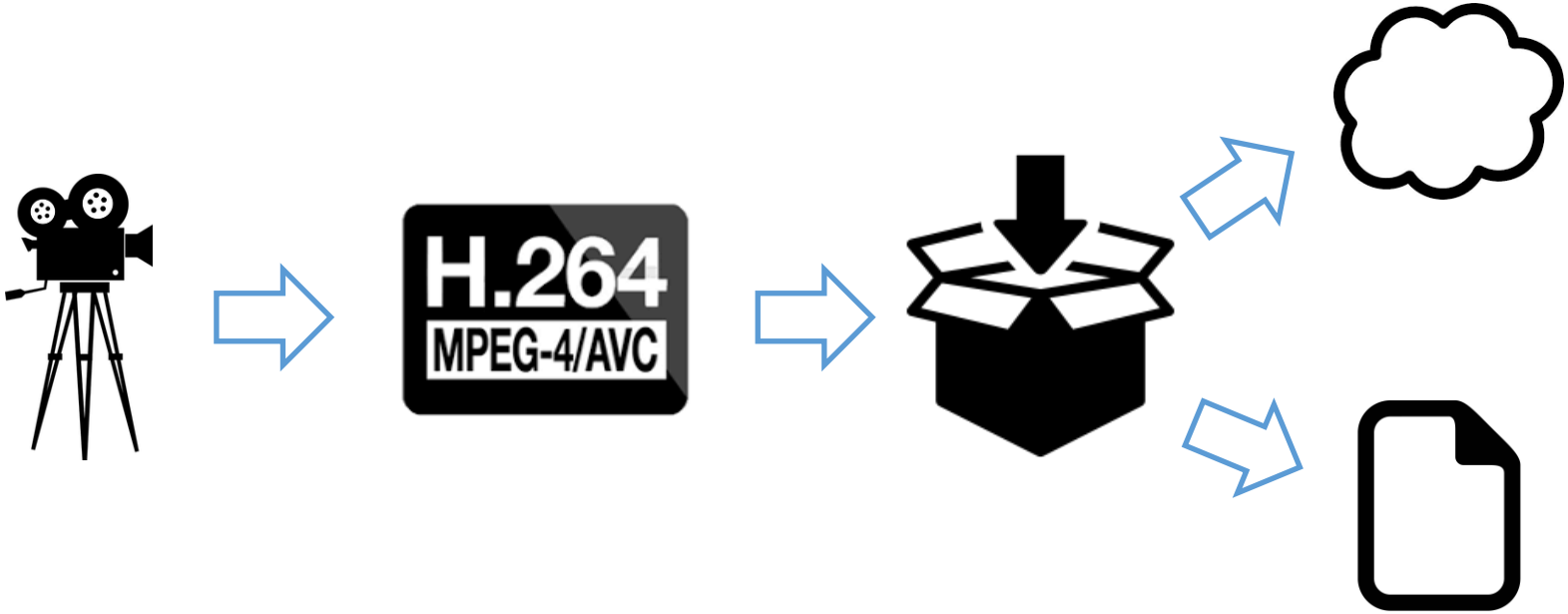


Skew

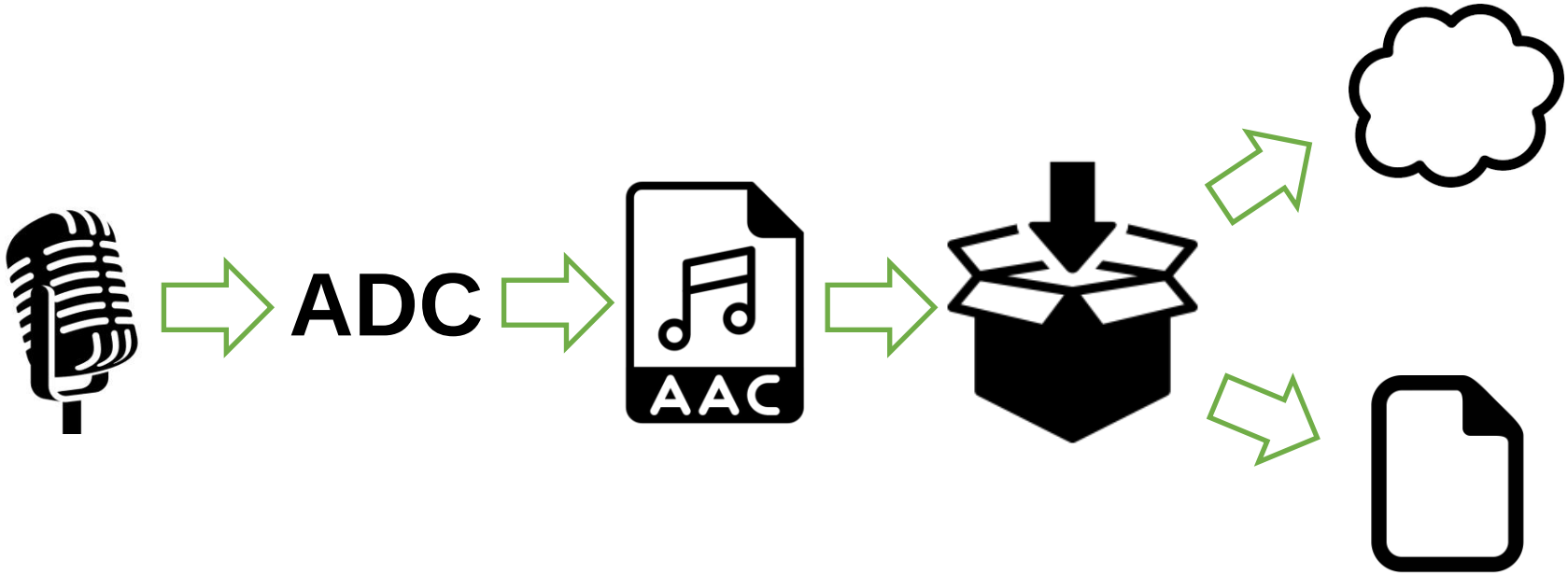


Recording side

Recorder Delays - Video

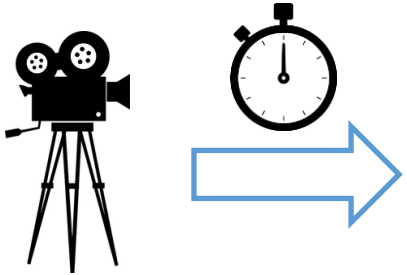


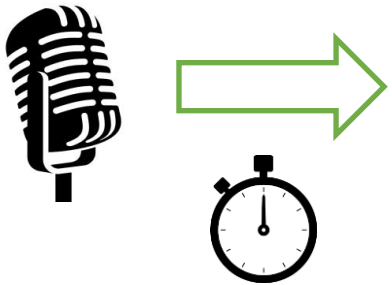
Recorder Delays - Audio



Three clocks

Video timestamps





Audio timestamps

- Video frames may carry time stamps set by the camera clock
- Audio frames may carry timestamps set by the microphone
- By default we can't guarantee that two clocks are synced and we need third clock that can be used for sync

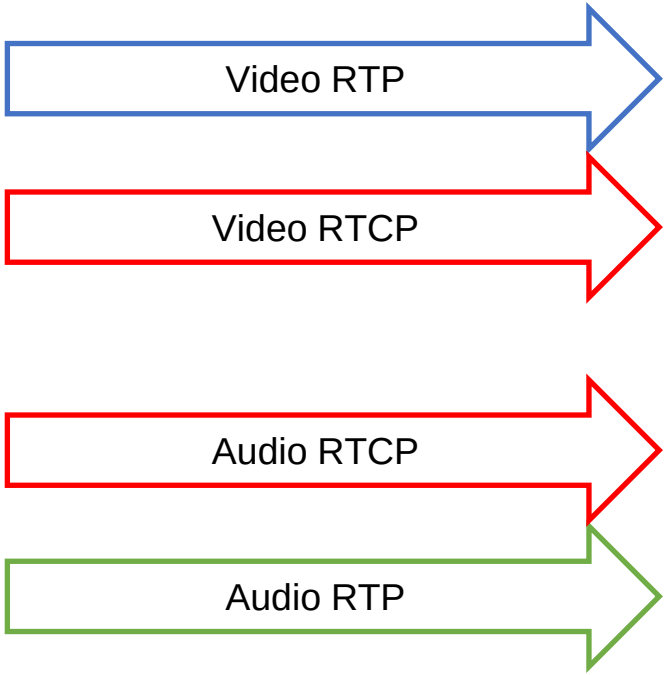
Wall Clock

How RTP handles multiple clocks

Video timestamps

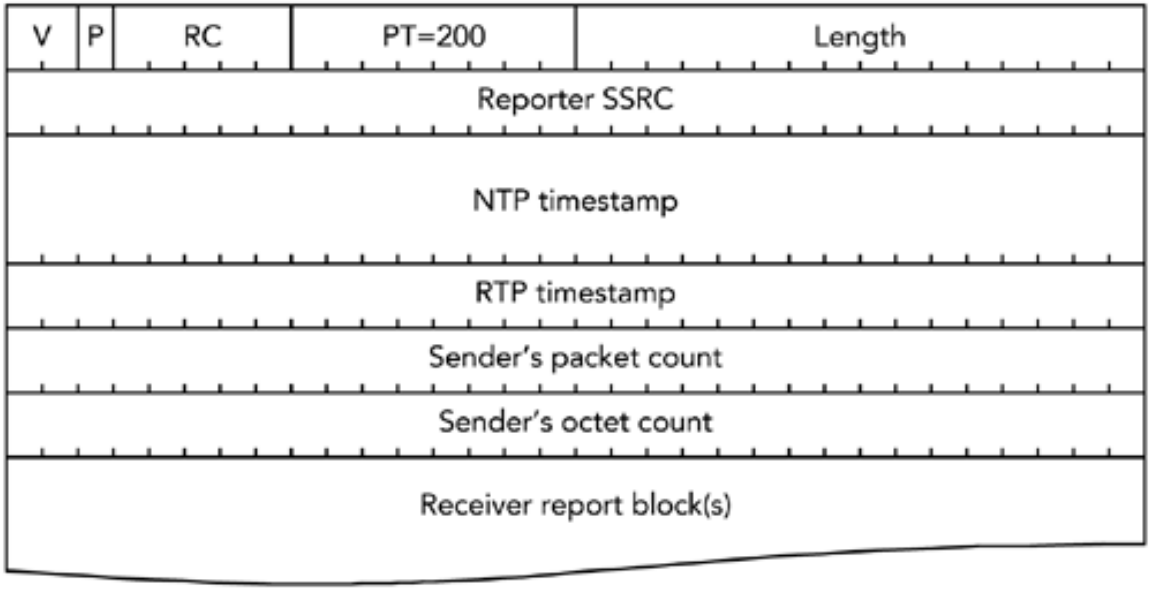
Wall Clock

Audio timestamps



- RTCP packets for Video Stream carry camera clock and wall clock timestamps
- RTCP packets for Audio Stream carry microphone clock and wall clock timestamps

RTCP packet



Wall clock

Audio/Video clock

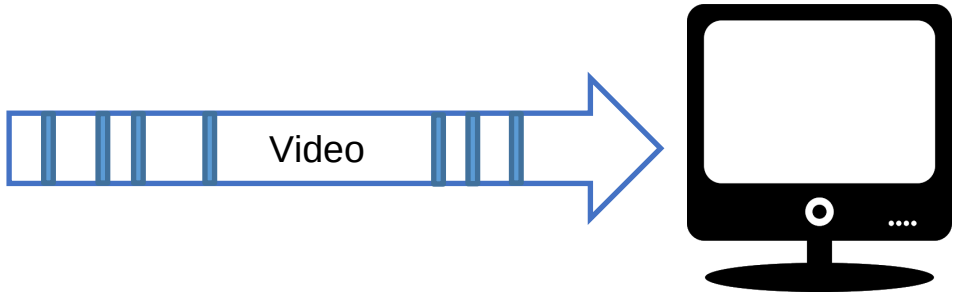
Rules for sending side

- All A/V Sync logic will happen on the receiving side before playback
- Sending side needs to provide enough info for this
- This info should include timestamps for all audio and video chunks and wall clock timestamps to synchronize streams

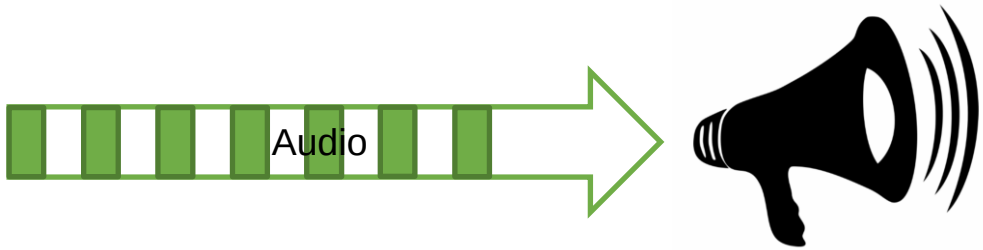
Playback side

Playback difference

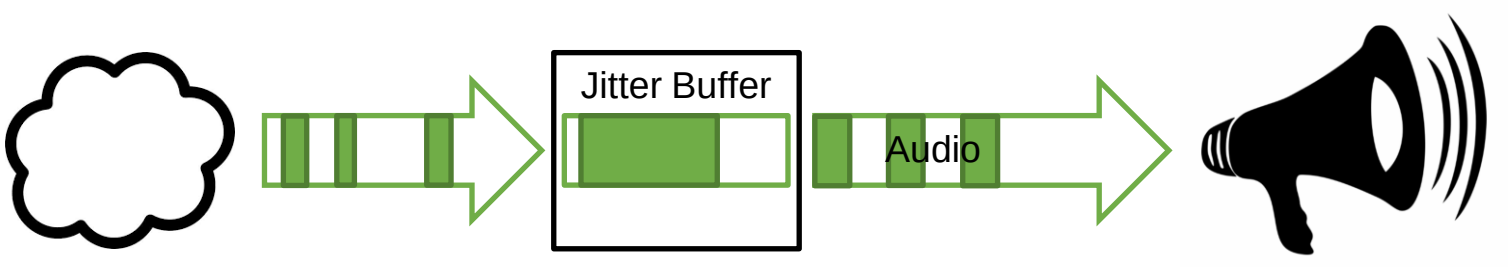
Malleable - can play a media sample on command, at any time



Nonmalleable - always consume data at a constant rate

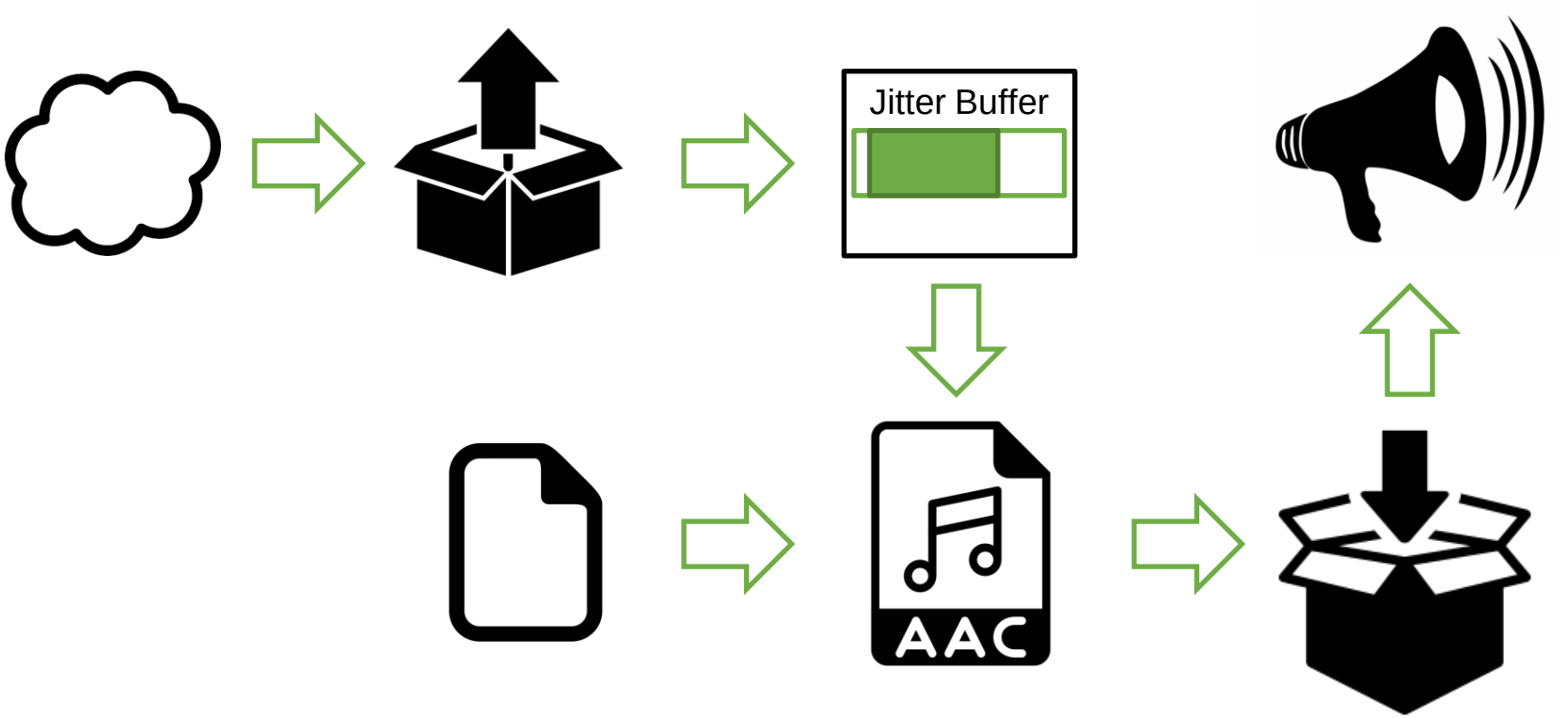


Jitter Buffer

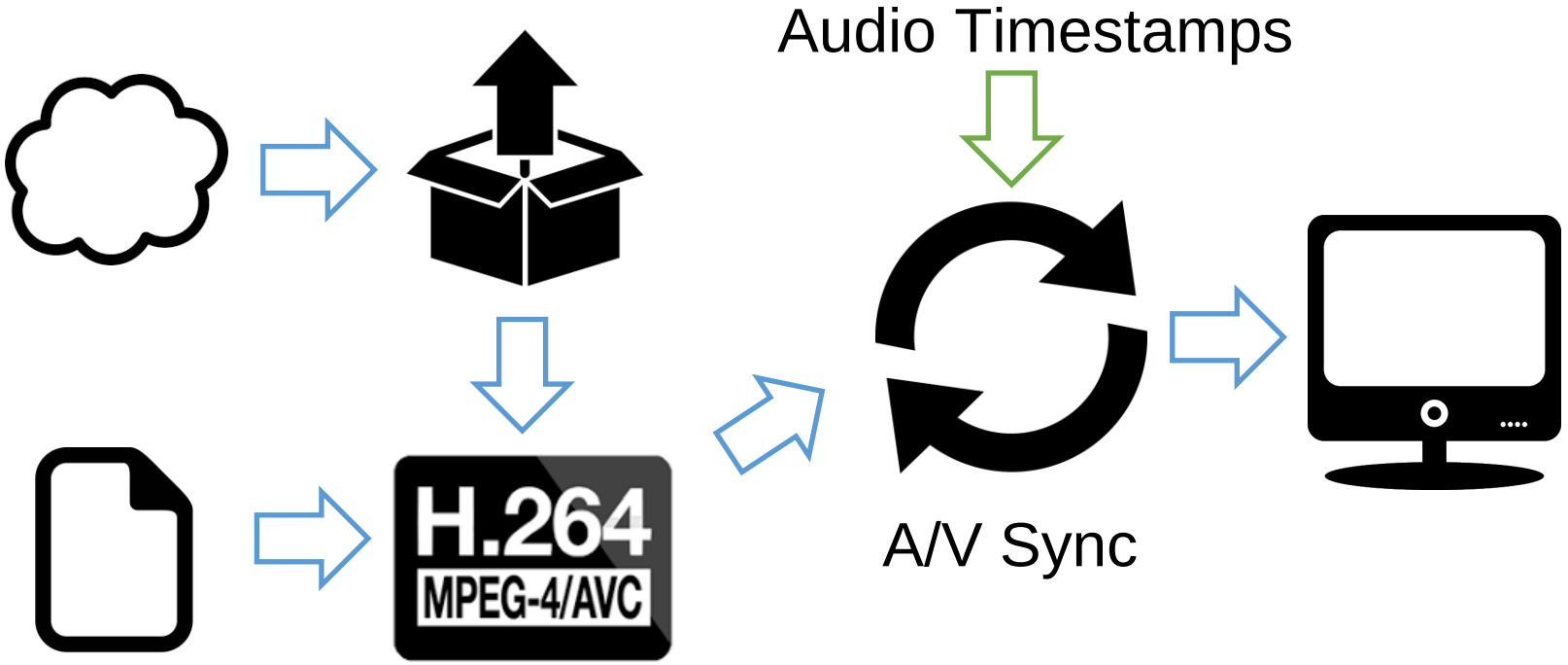


- Nonmalleable devices require Jitter Buffer

Player Delays - Audio



Player Delays - Video



Human Sensitivity

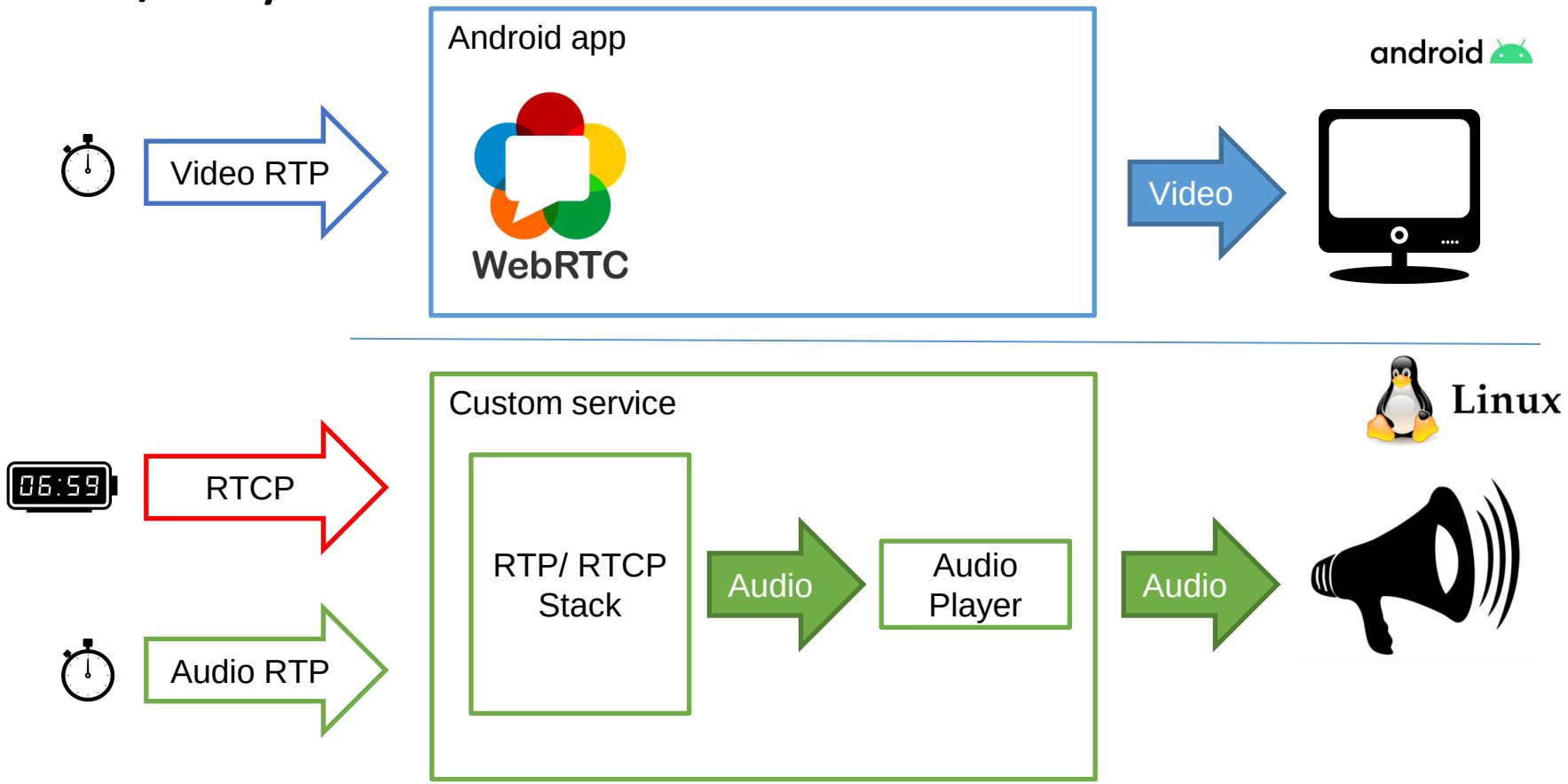
- Humans can notice very small skew. Especially if video has high resolution and frame rate
- Interrupts in audio are more noticeable
- Skew is more noticeable if audio is ahead of video

A/V Sync Rules

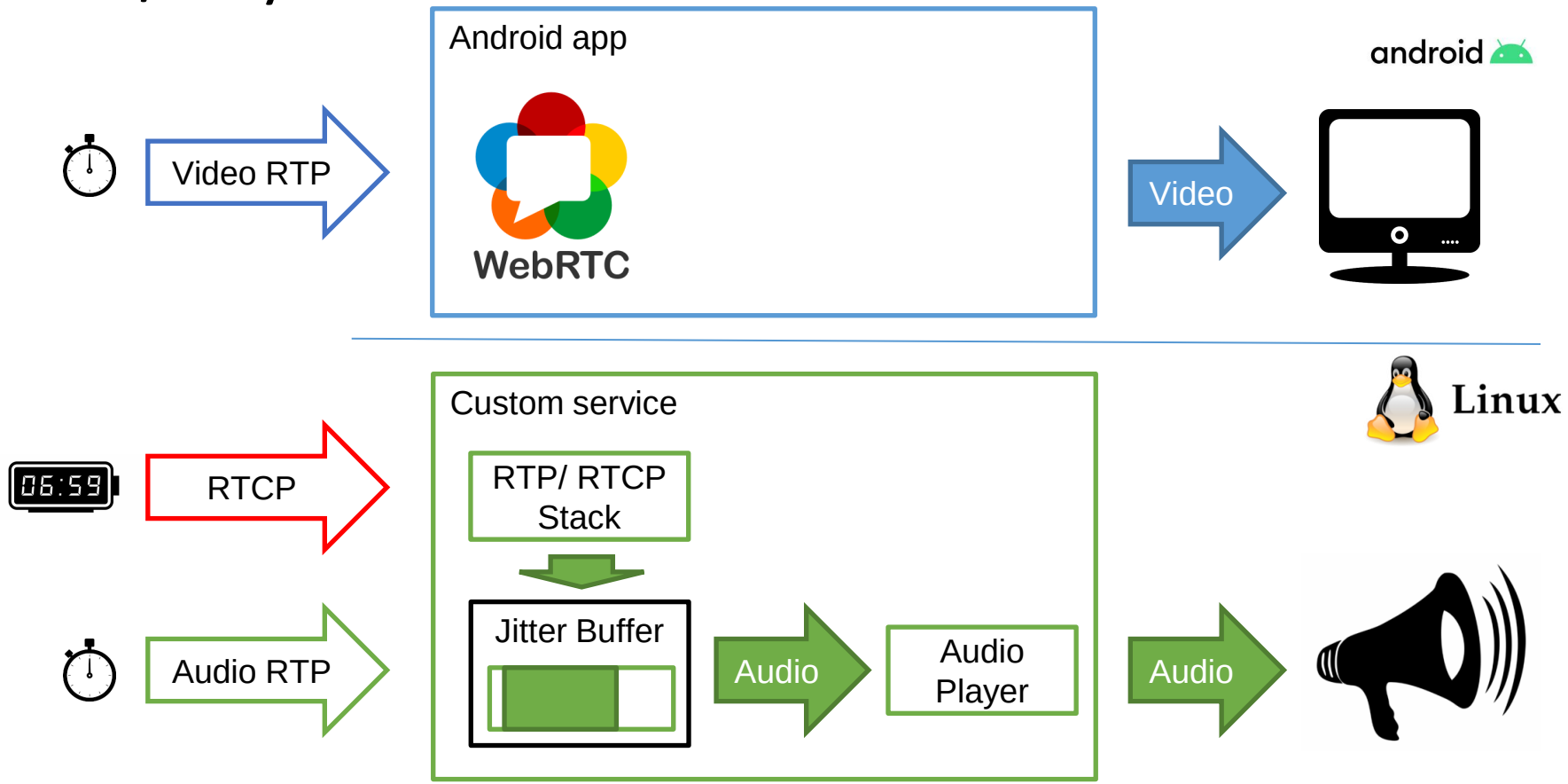
1. A/V Sync happens entirely on receiving side
2. Play audio at constant rate and adjust video to it
3. If audio packets come before video packets the whole audio stream needs to be delayed (can use Jitter Buffer for this)
4. The only way to ensure sync is to carry timestamps from start to finish
5. Make sure that timestamps for Audio and Video streams are coming from the same clock

Sync architecture

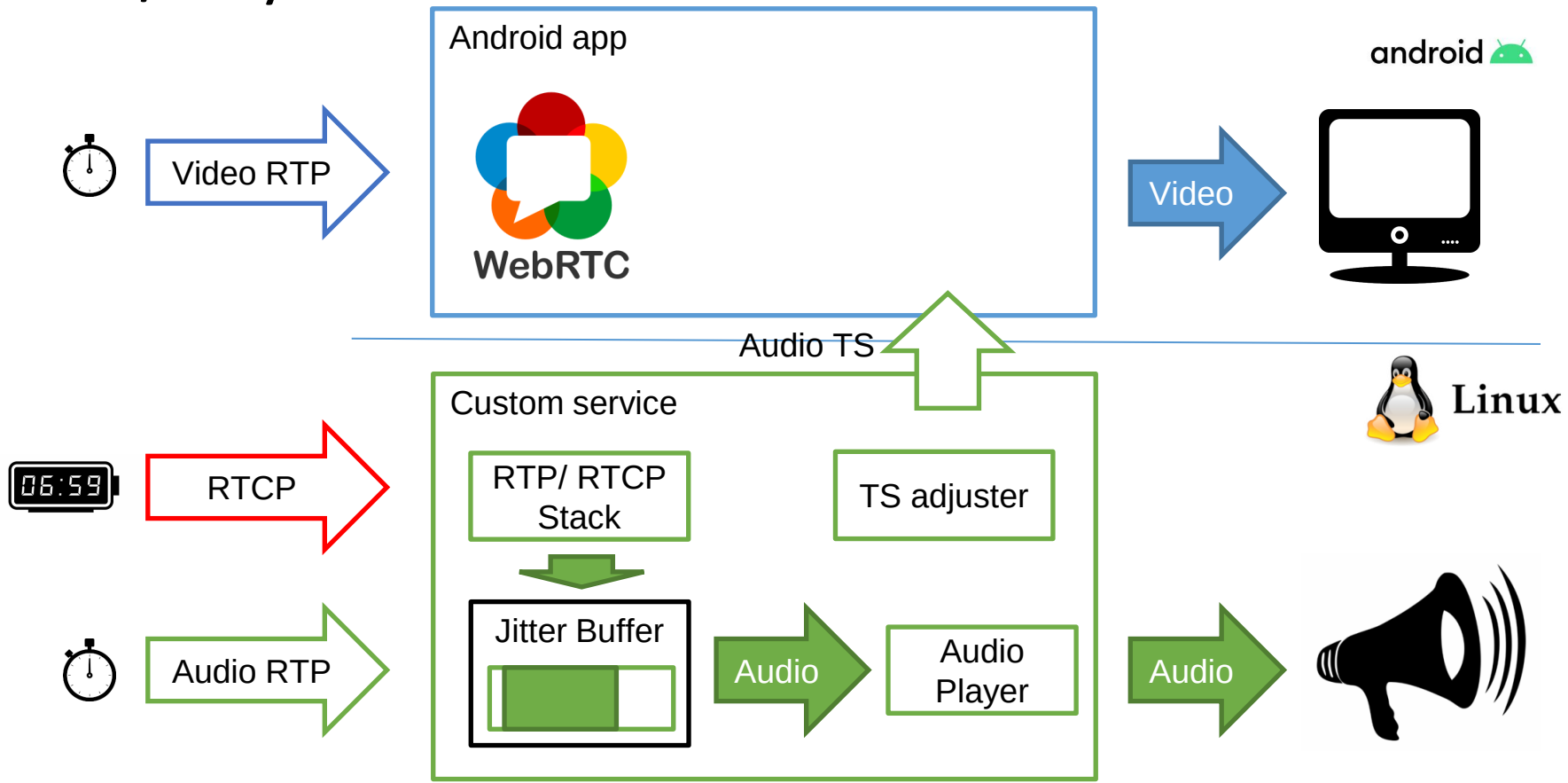
A/V Sync Architecture



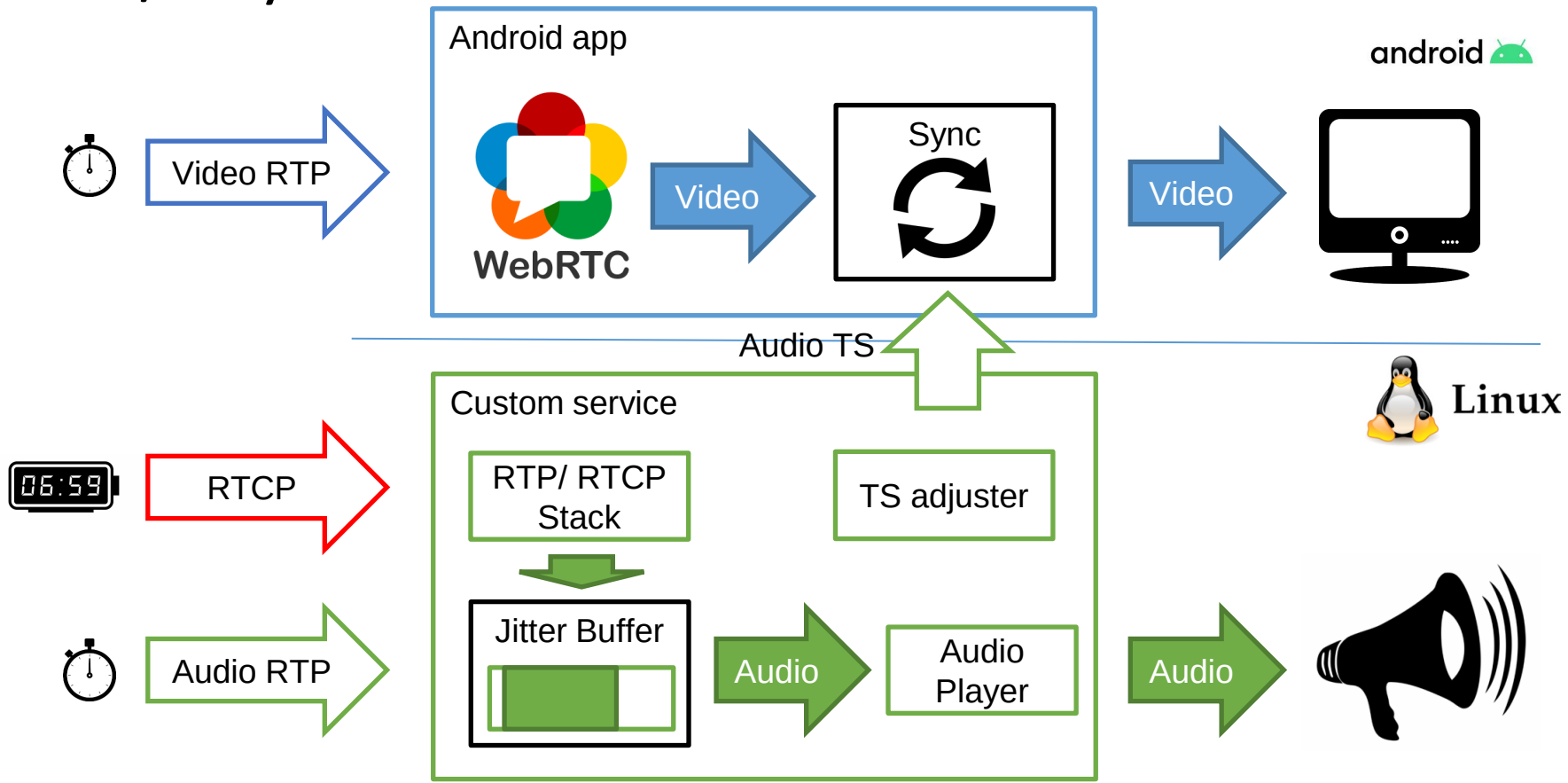
A/V Sync Architecture



A/V Sync Architecture



A/V Sync Architecture



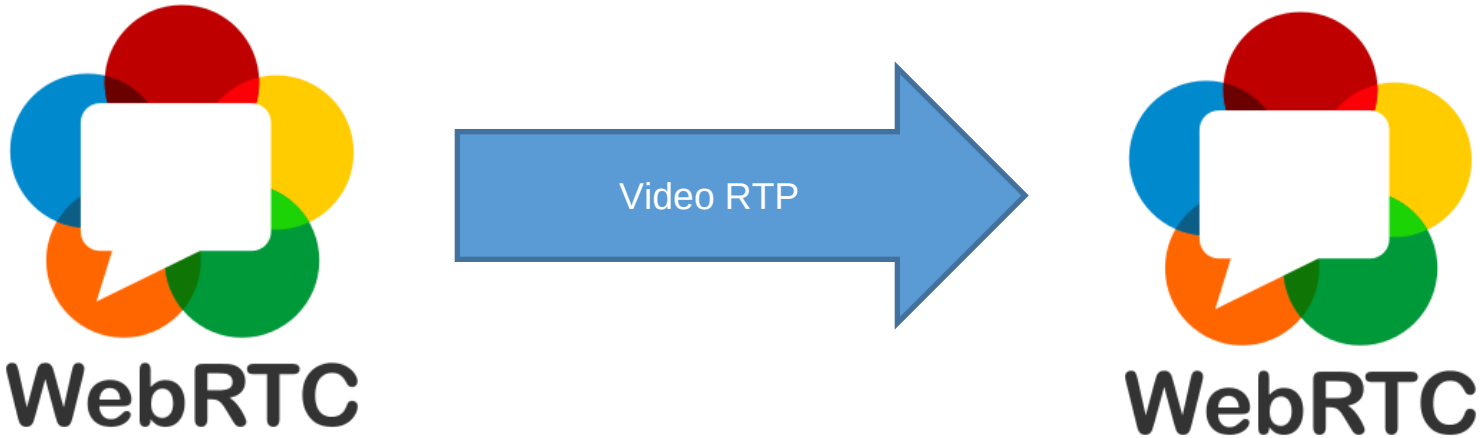
WebRTC

WebRTC captures video

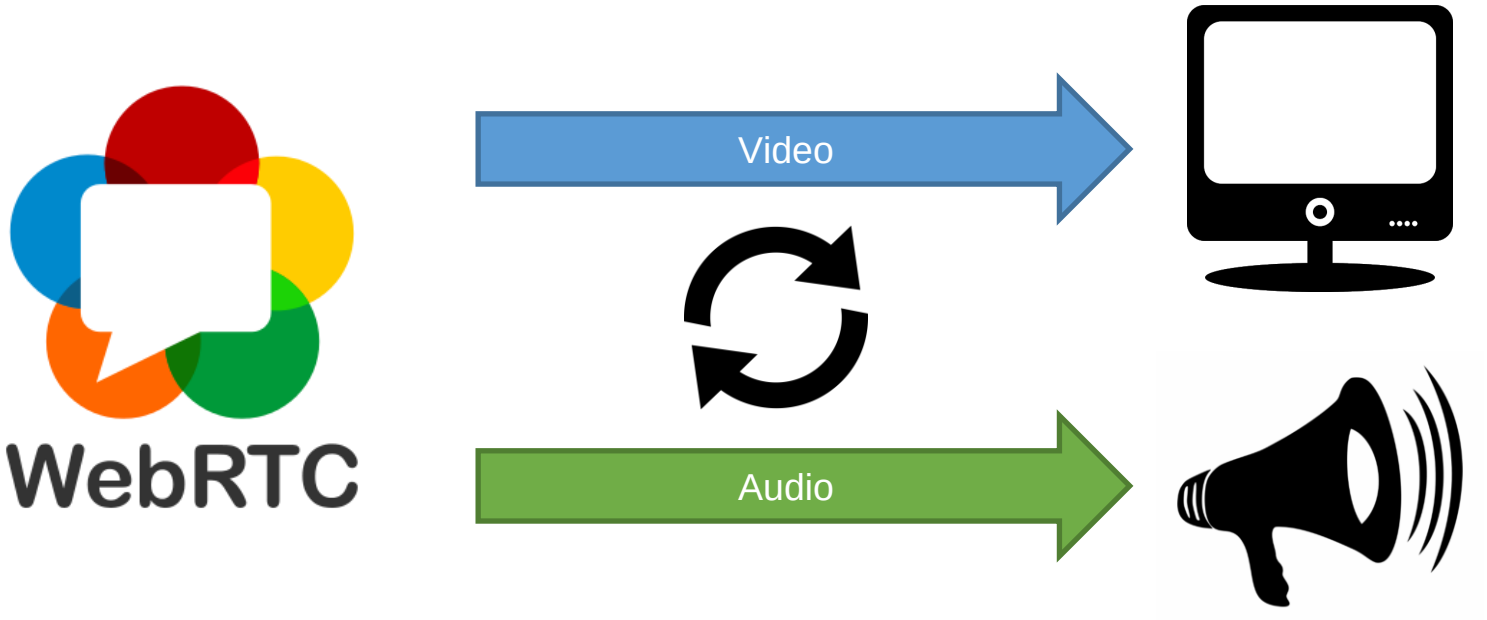
MediaRecorder

WebRTC

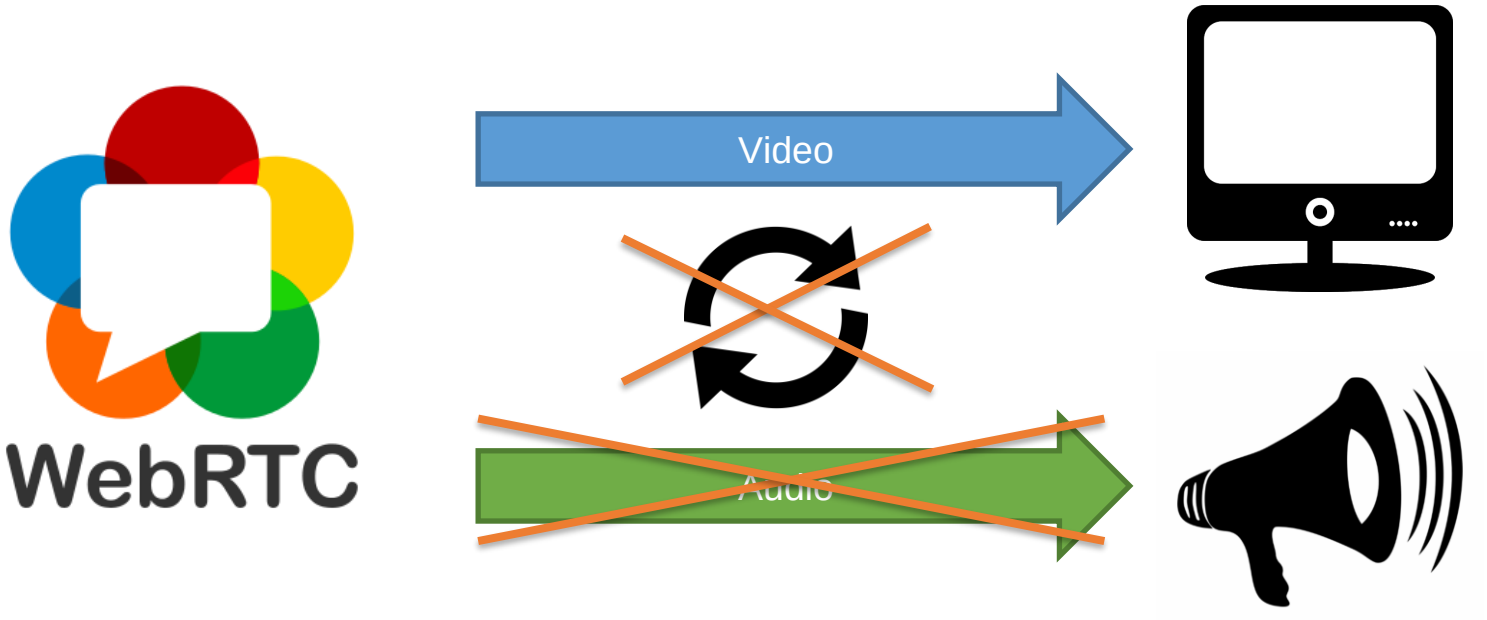
WebRTC transfers video



WebRTC can A/V Sync…



… only if it handles both audio and video



MediaSync

MediaSync – native Android framework

- Can be used to synchronously play audio and video streams
- Can be used to play audio-only or video-only stream
- Works like this:
  1. Take frame from buffer
  2. Adjust frame timestamp
  3. Send it for Playback/Rendering

How MediaSync really works

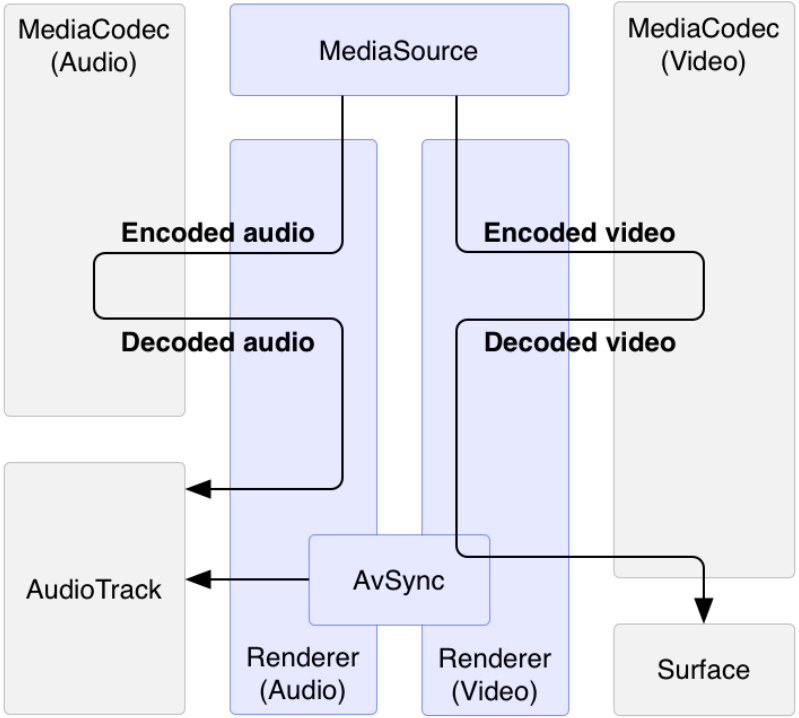
1. Configuration parameters are ignored
2. System monolithic clock is used to get “now” time
3. Video frame presentation time is calculated based on “now” time and VSYNC
4. If audio is present and video is more than 40ms behind – skip video frame to catch up

Conclusion

- MediaSync is not configurable for video playback
- MediaSync only helps with adjusting to VSYNC really, if you only have one stream
- We can MediaSync for video playback, but it does not solve our problem

ExoPlayer

A/V Sync in ExoPlayer

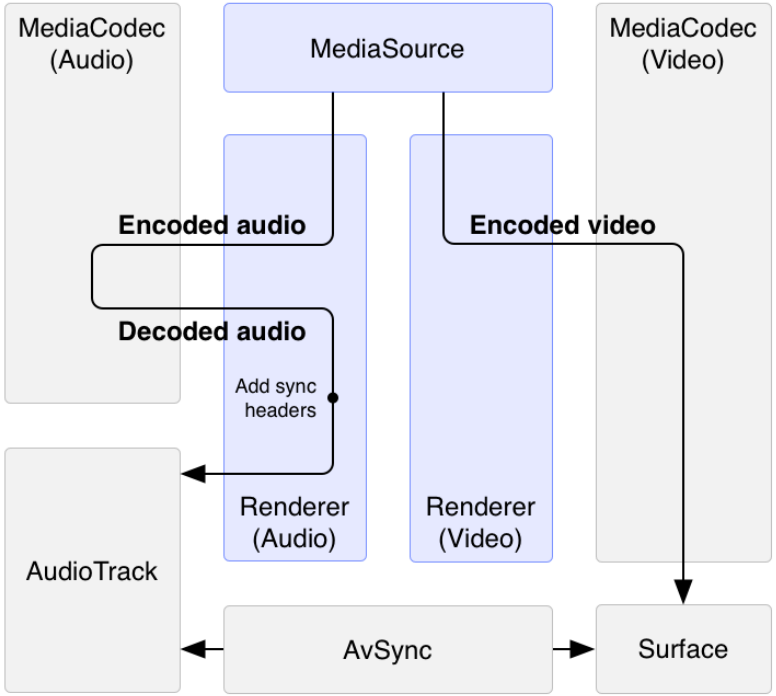


- ExoPlayer implements A/V Sync using standard Android APIs: *AudioTrack* & *MediaCodec*
- Audio and Video from *MediaSource* must carry synced timestamps for A/V Sync to work

A/V Sync in ExoPlayer

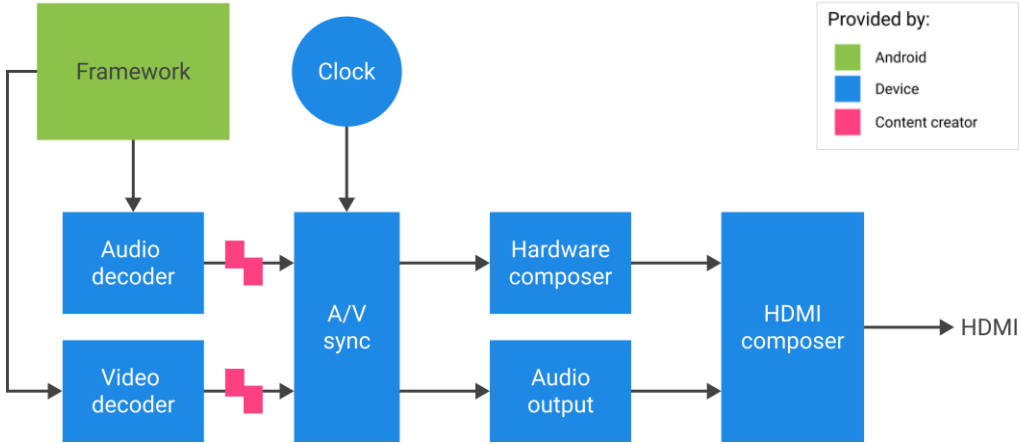
```
...
audioTrack.getTimestamp(audioTimestamp);//Get audio timestamp from AudioTrack
...

//Some very complicated logic to adjust this timestamp

...
protected void renderOutputBufferV21(
    MediaCodec codec, int index, long presentationTimeUs, long releaseTimeNs) {
  ...
  //Using adjusted timestamp to render video frame
  codec.releaseOutputBuffer(index, releaseTimeNs);
  ...
}
```

Multimedia tunneling in ExoPlayer



- ExoPlayer supports Multimedia tunneling
- This is more efficient and it offloads A/V Sync to the underlying platform

Multimedia Tunneling



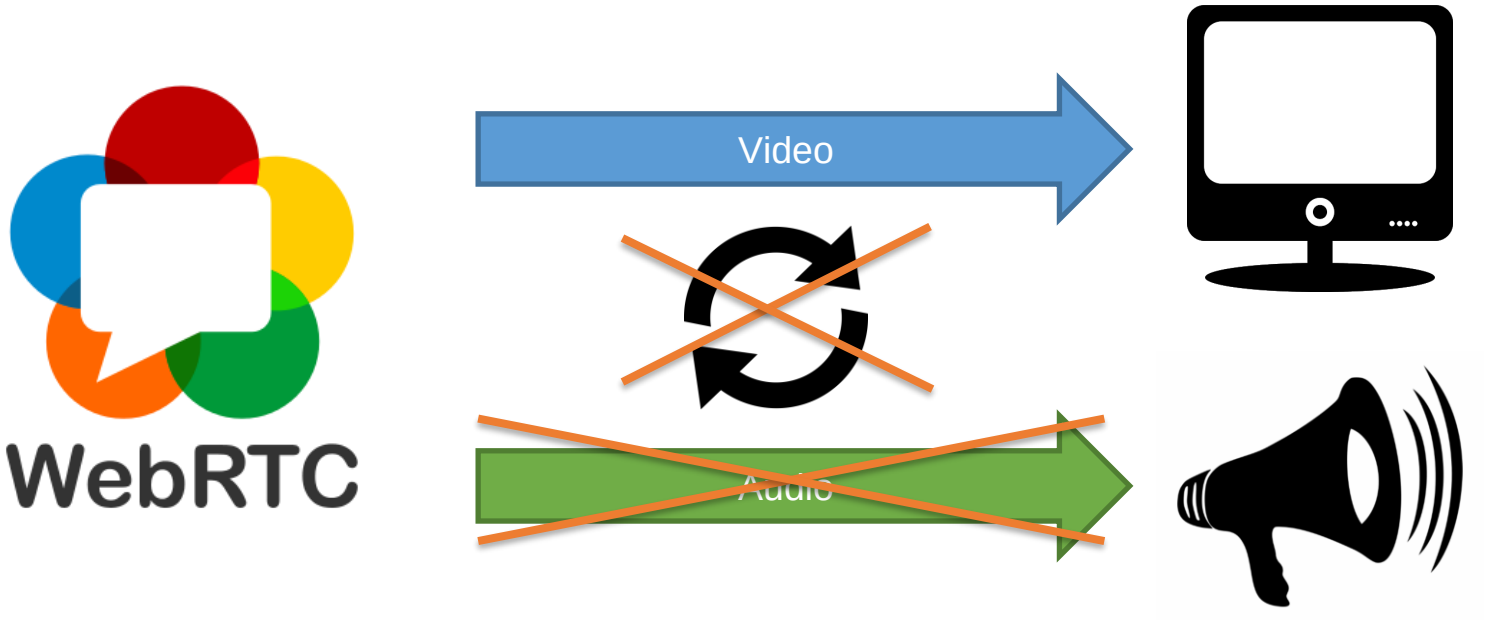
- This is an optional feature that is present on some devices (mostly Android TV)
- Usually not all codecs support it
- Useful feature if you want to play video in 4K

Conclusion

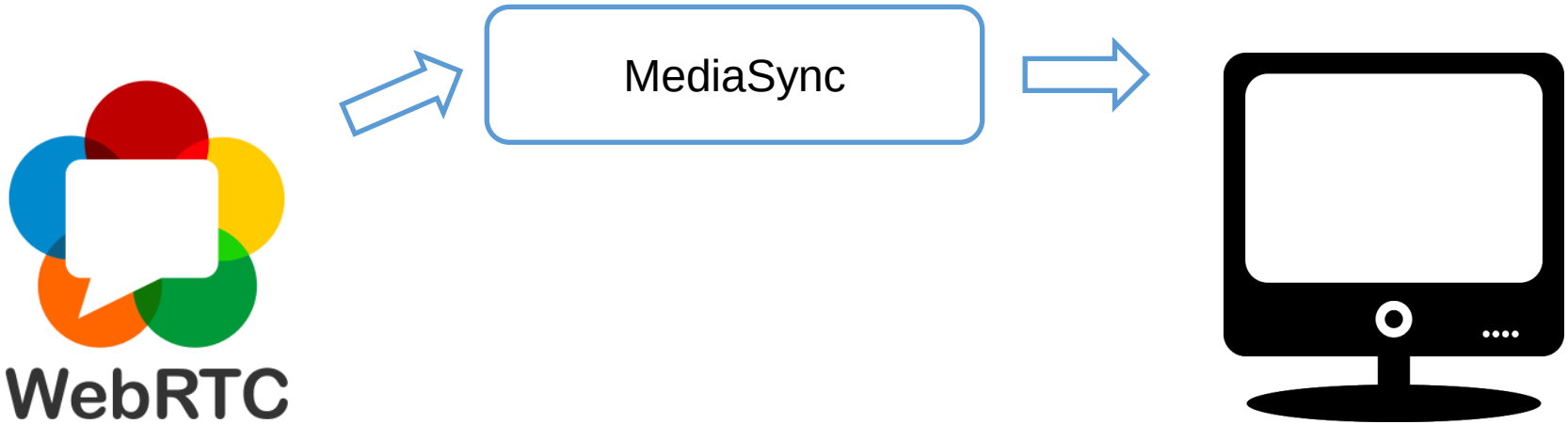
- By default, ExoPlayer implements its own complex A/V sync logic, but it only handles playback
- ExoPlayer also supports multimedia tunneling (if device can provide this functionality)
- In either case it only works if Audio and Video in MediaSource already synced somehow
- We could’ve used ExoPlayer, but decided against it:
  1. Only works for playback
  2. Complex sync logic too hard for us to understand and adjust

WebRTC again?

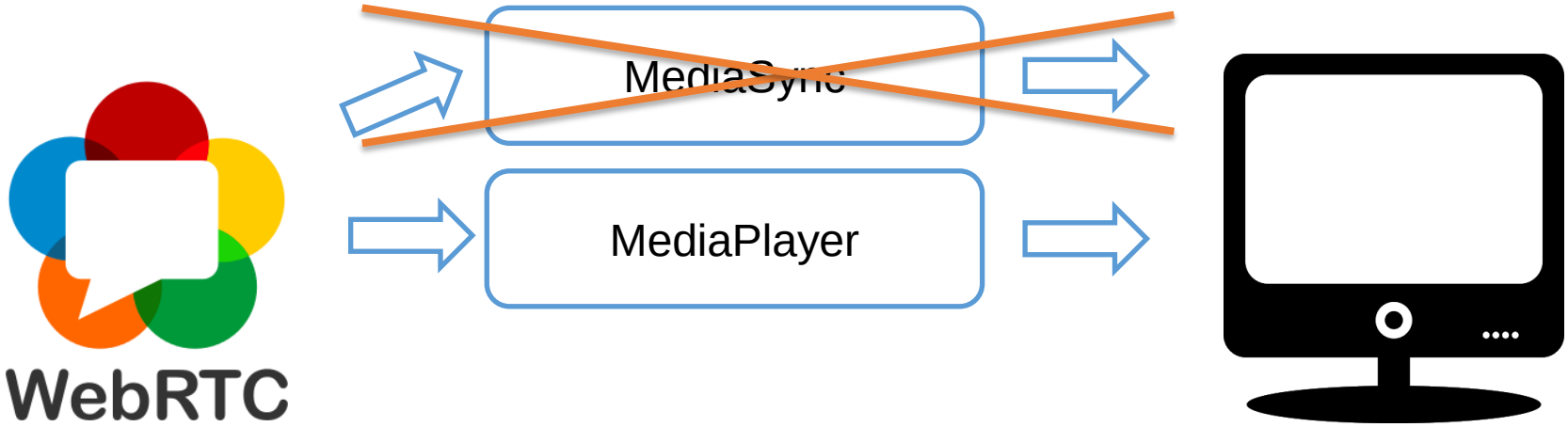
WebRTC can’t sync just video



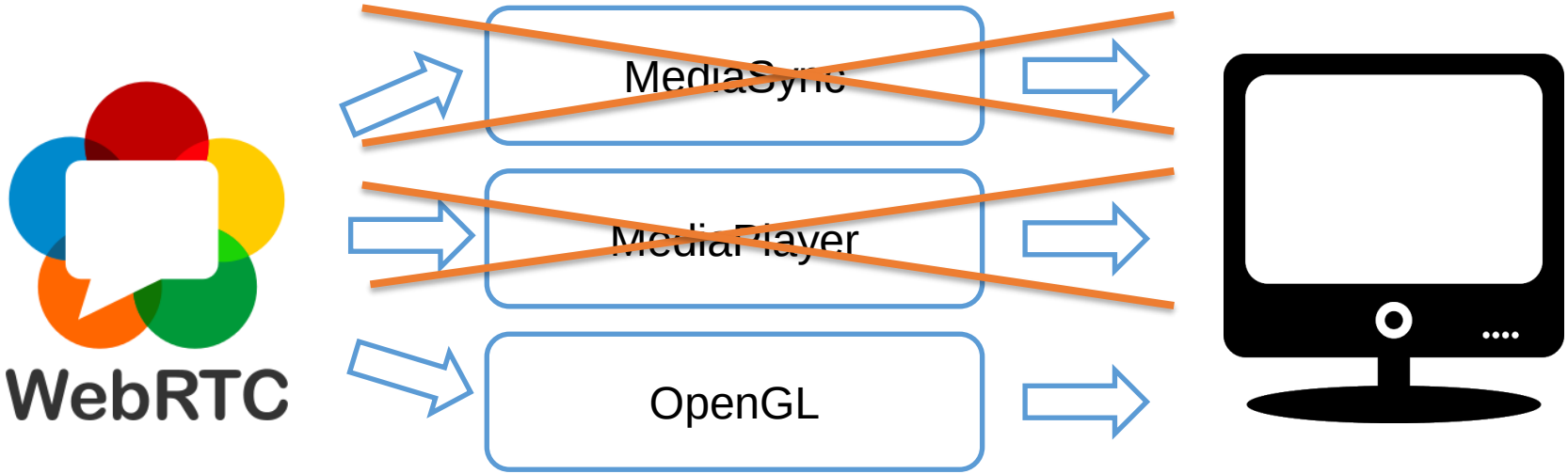
WebRTC renders video with…



WebRTC renders video with…



WebRTC renders video with…



WebRTC renders video with OpenGL

```
//defined in EglRenderer.java
private void renderFrameOnRenderThread() {
    ...
    eglBase.swapBuffers(frame.getTimestampNs());
```

```
//defined in EglBase14Impl.java
public void swapBuffers(long timeStampNs) {
    ...
    EGLExt.eglPresentationTimeANDROID(eglDisplay,eglSurface, timeStampNs);
    EGL14.eglSwapBuffers(eglDisplay, eglSurface);
```

WebRTC renders video with OpenGL

```
//defined in EglRenderer.java
private void renderFrameOnRenderThread() {
    ...
    eglBase.swapBuffers(frame.getTimestampNs());
```

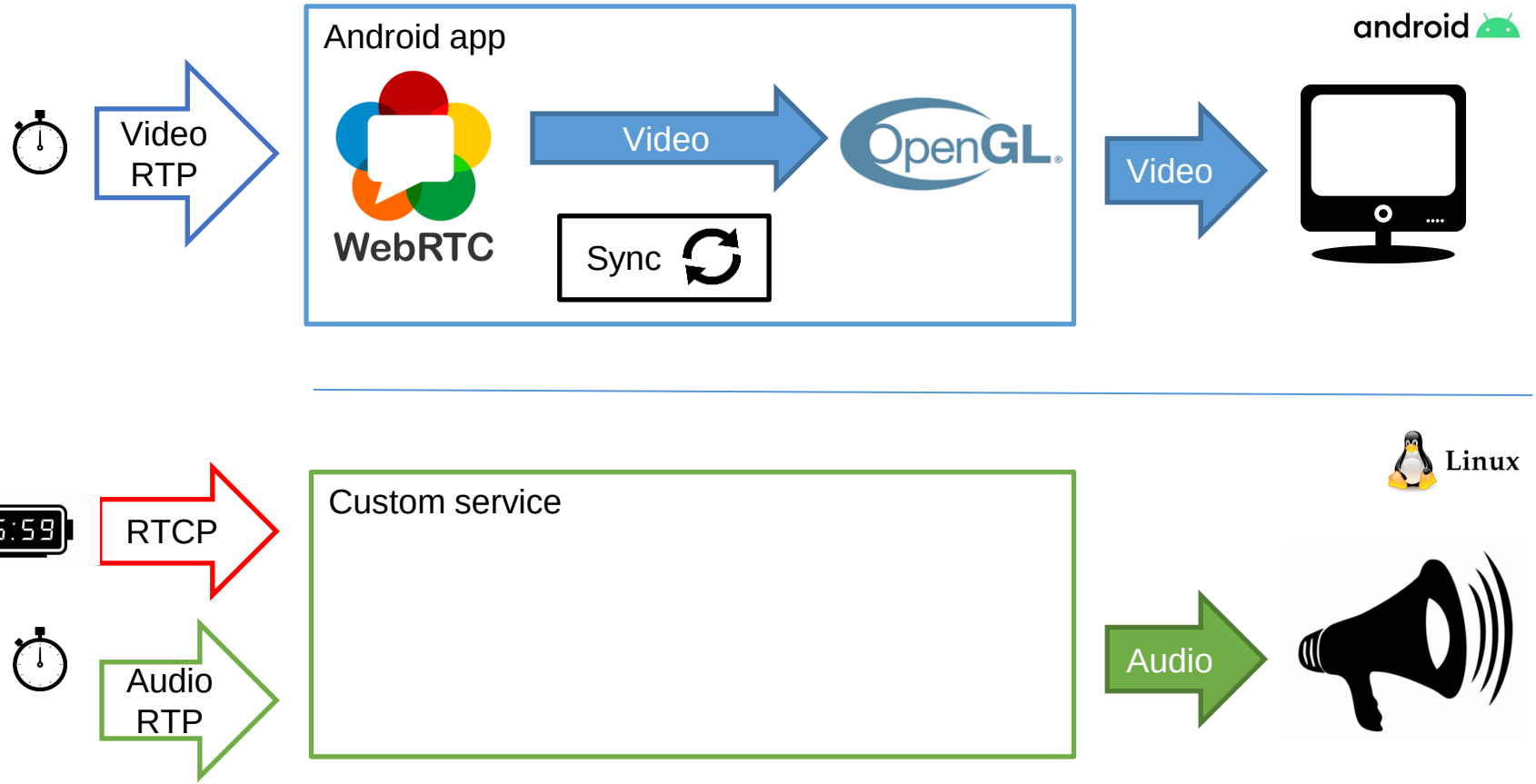
```
//defined in EglBase14Impl.java
public void swapBuffers(long timeStampNs) {
    ...
    EGLExt.eglPresentationTimeANDROID(eglDisplay,eglSurface, timeStampNs);
    EGL14.eglSwapBuffers(eglDisplay, eglSurface);
```

Conclusion

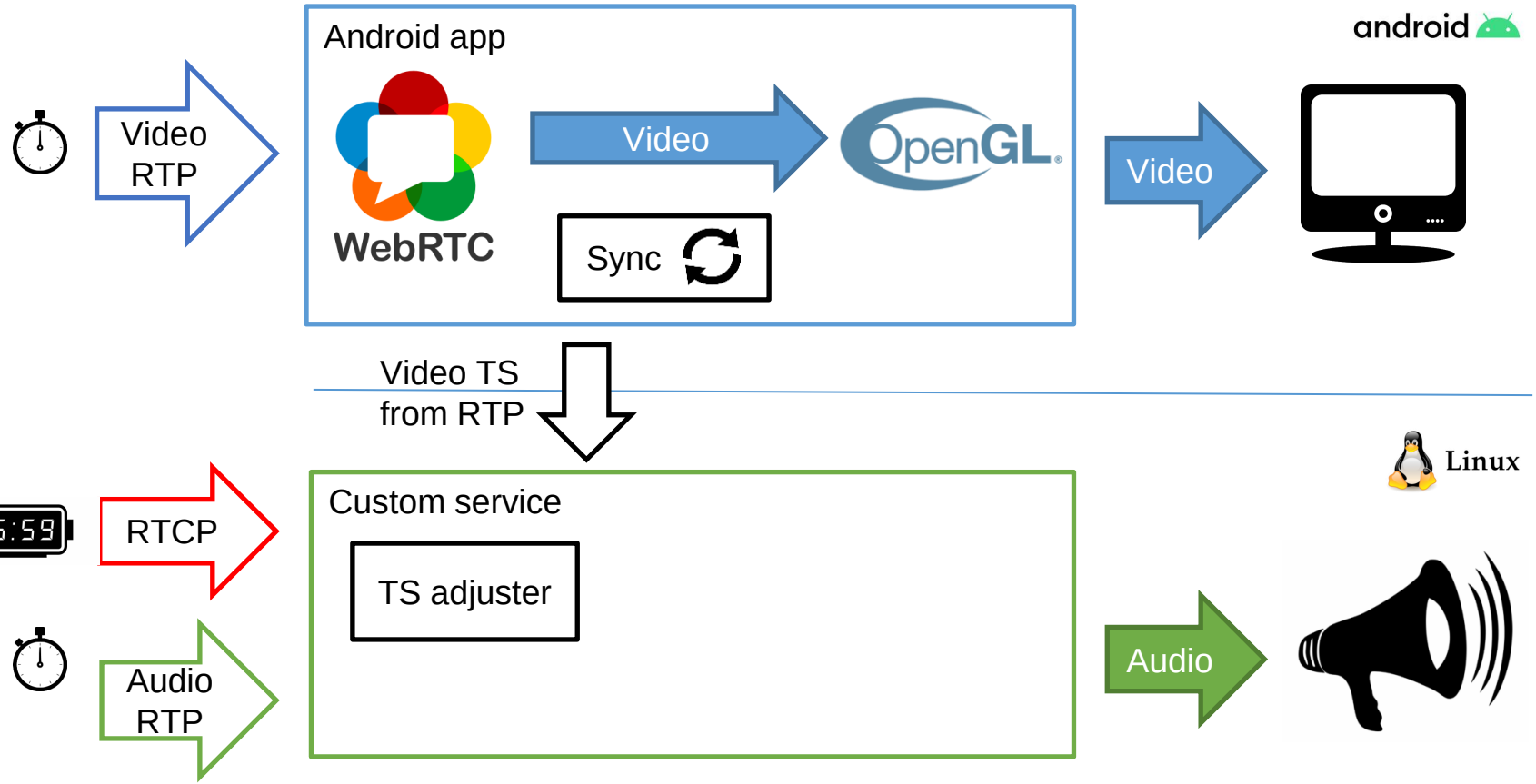
- WebRTC was the obvious choice for us, since it took care of recording video, transferring and playing it
- WebRTC has an internal A/V sync logic that we couldn’t use, since we only WebRTC for video
- Fortunately, WebRTC uses OpenGL for rendering video and it had interface for modifying video frame presentation time

Conclusion

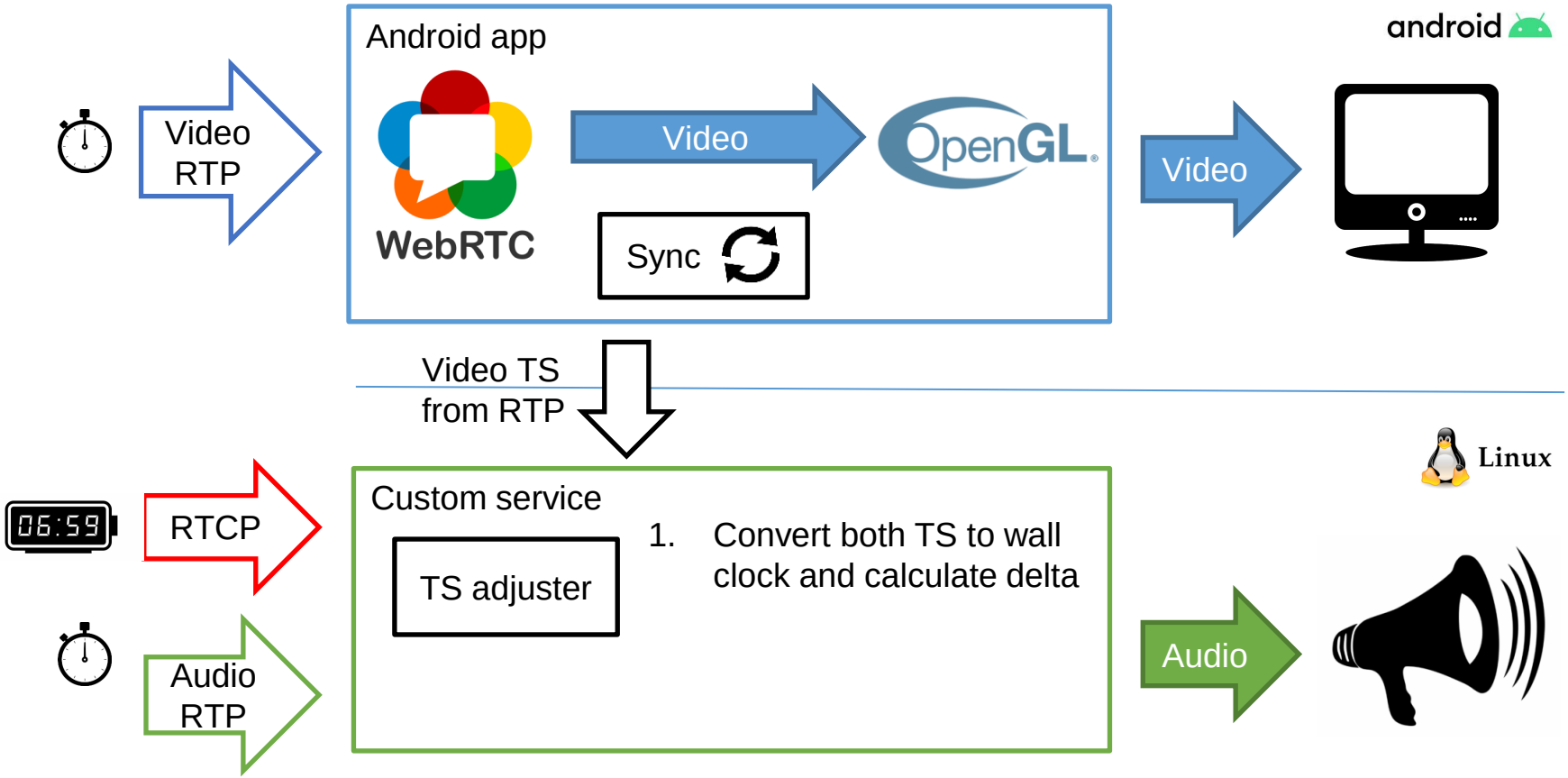
Final solution



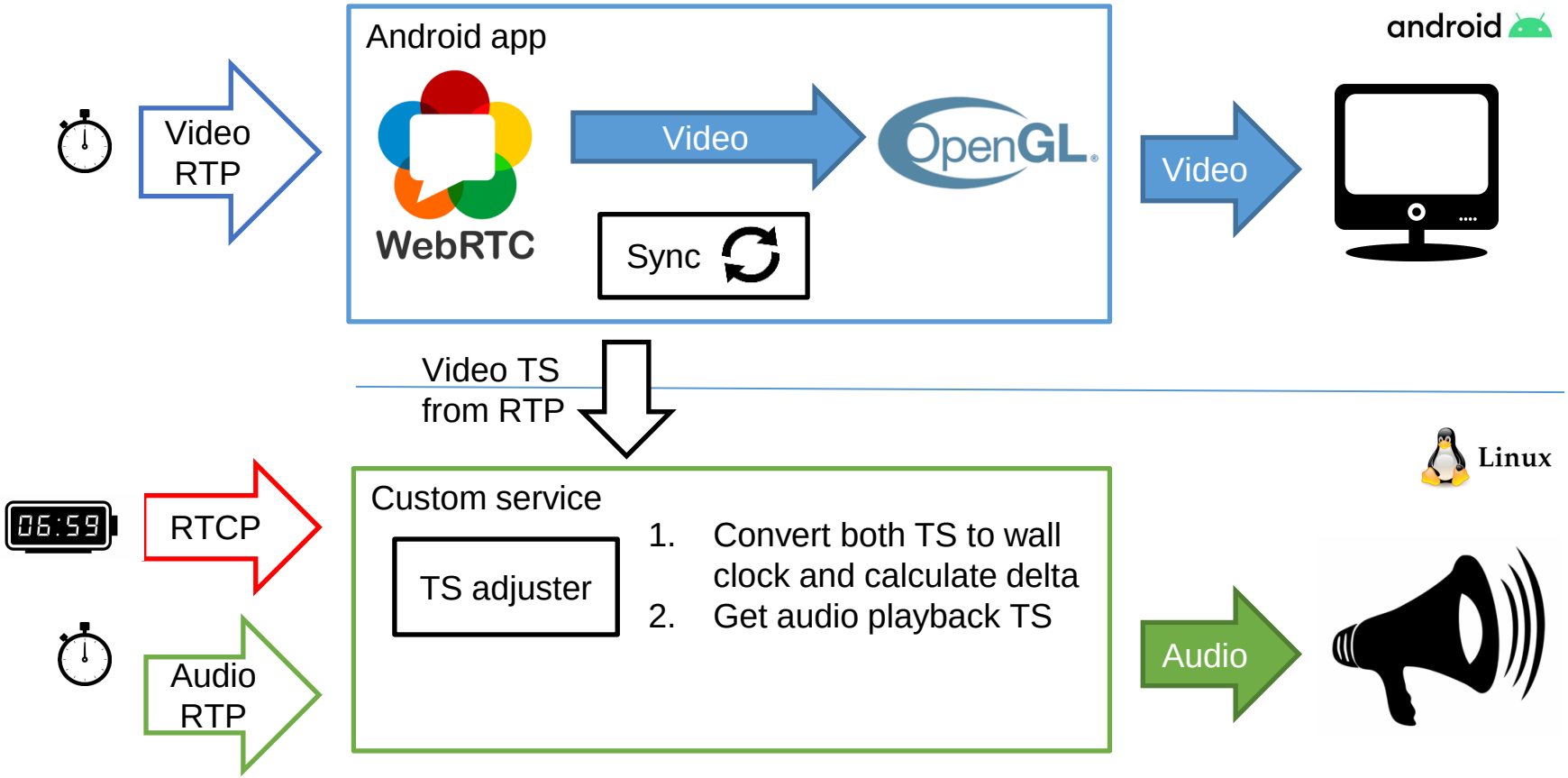
Final solution



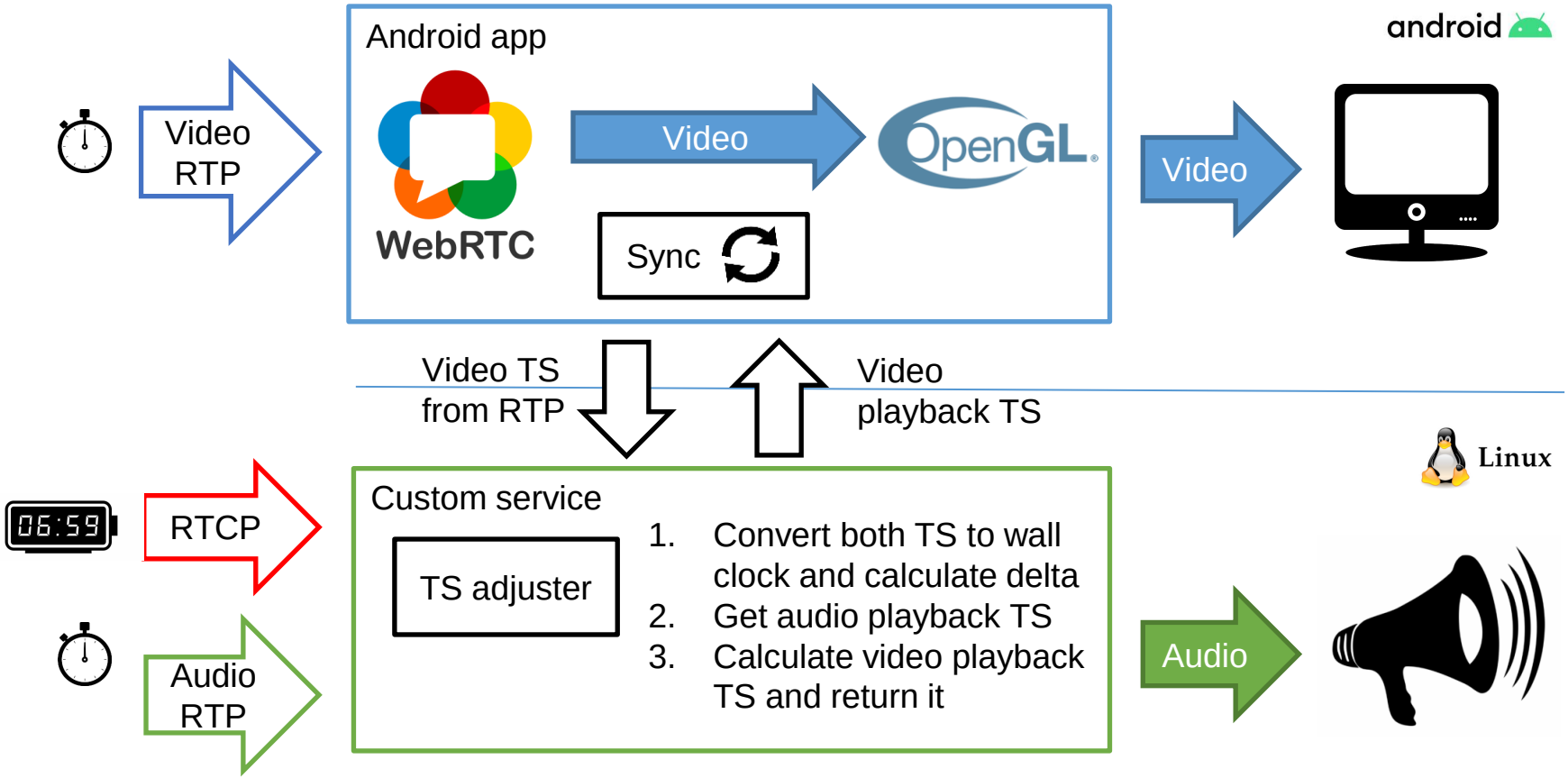
Final solution



Final solution



Final solution



Conclusion

- A/V sync is an advanced magic
- Better to use frameworks than implement it yourself
- MediaSync – can’t recommend
- ExoPlayer – excellent for playing video and supports multimedia tunneling
- WebRTC – works best if you want video calls synced

Thank You

Fedor Tcymbal

Technical Manager, CTO

fedor.tcymbal@orioninc.com

 @ftsymbal

orioninc.com