Video on Android

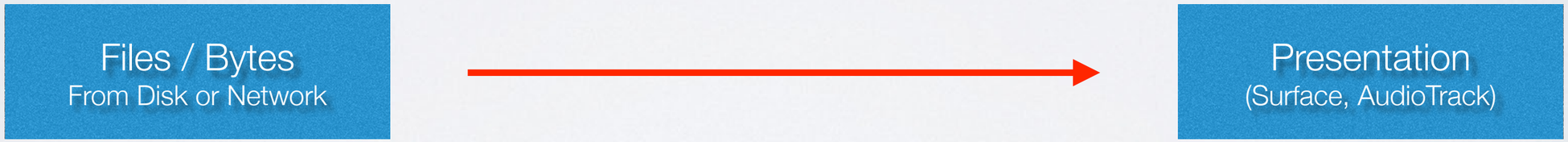
September 2014

by Edison Wang
(@wew)

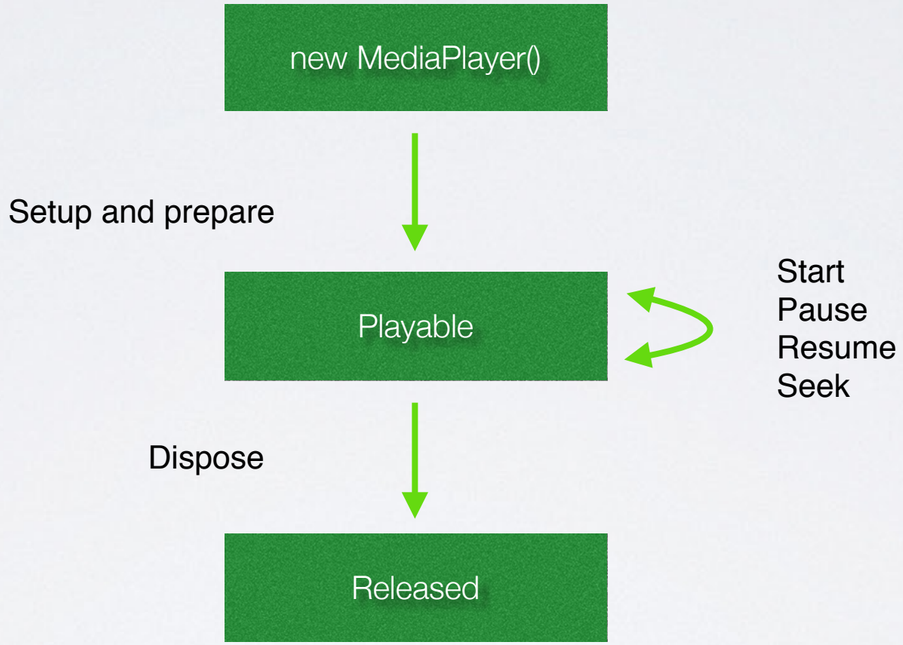
Video on Android

A stream of bytes that contains information for a **series of images** and **sound** to be presented to the on the screen and the speaker in a **predetermined pattern**.

Simple Playback



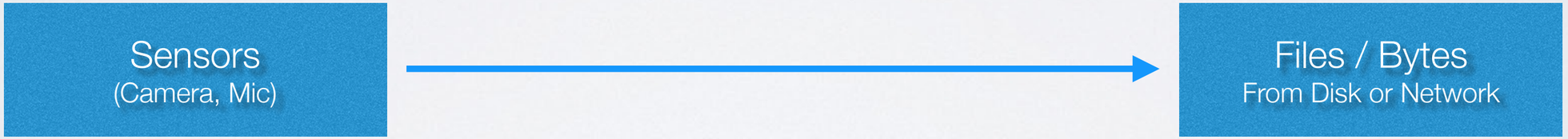
MediaPlayer



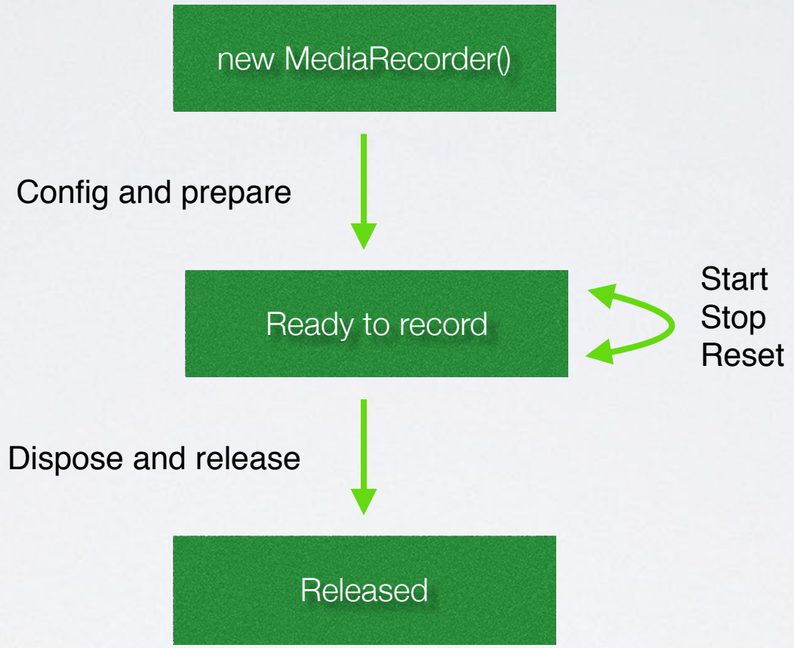
MediaPlayer

- Pros
  - API 1, great compatibility across different devices.
  - Good for most use cases.
  - Decoded frames can easily be reused.
- Cons
  - “WISWIG”, no control on how each part is done.
  - No playback control besides, pause(), resume(), and seek().
  - No way to handle custom caching, buffering, and adaptive playback

Simple Recording



MediaRecorder

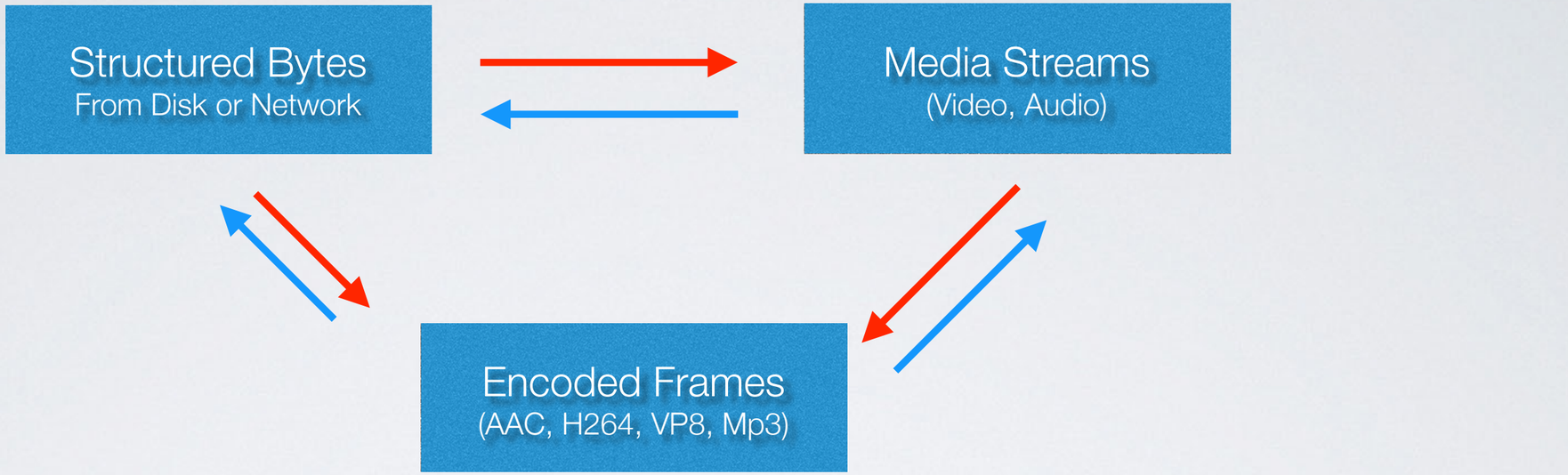


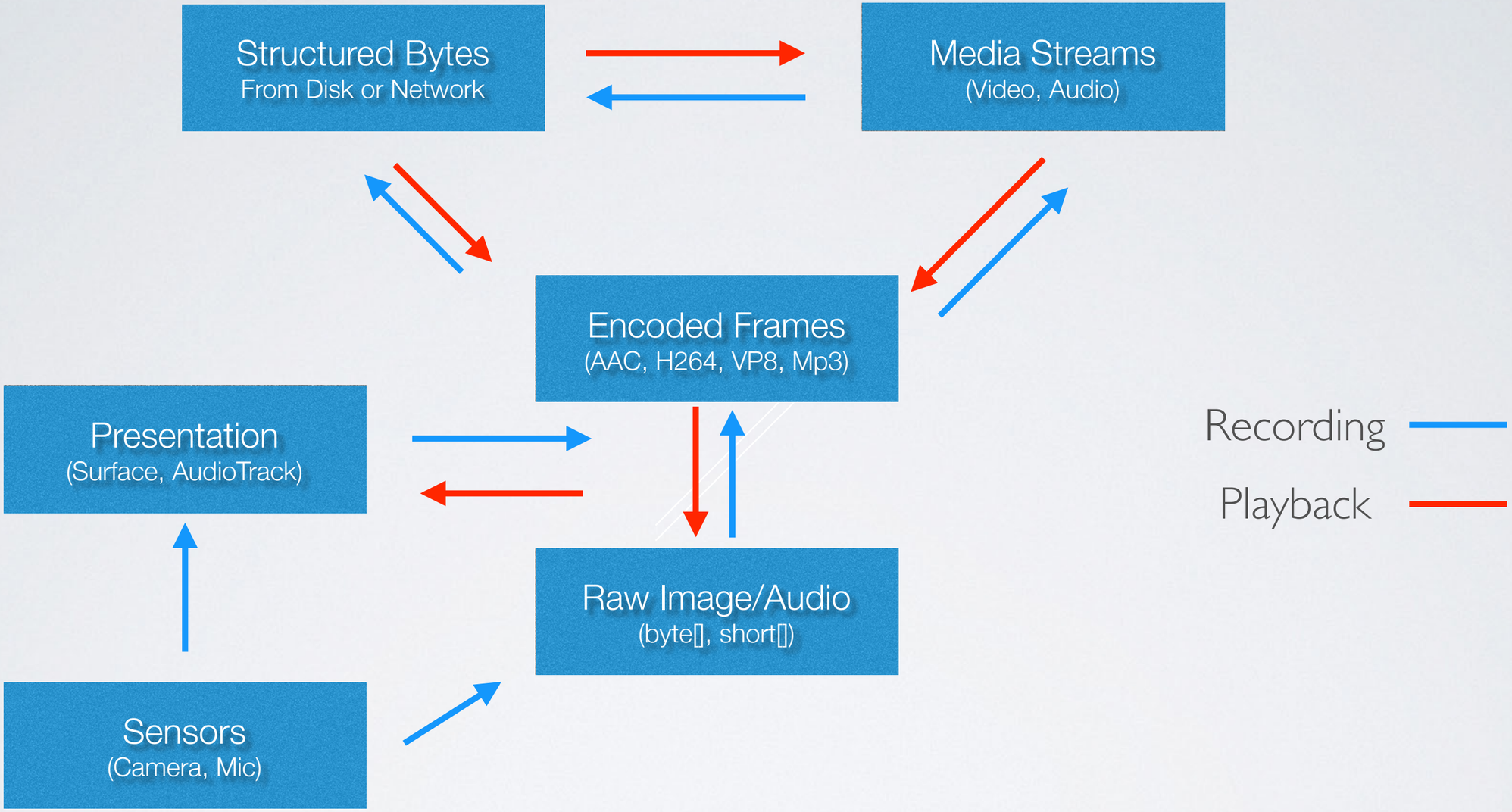
MediaRecorder

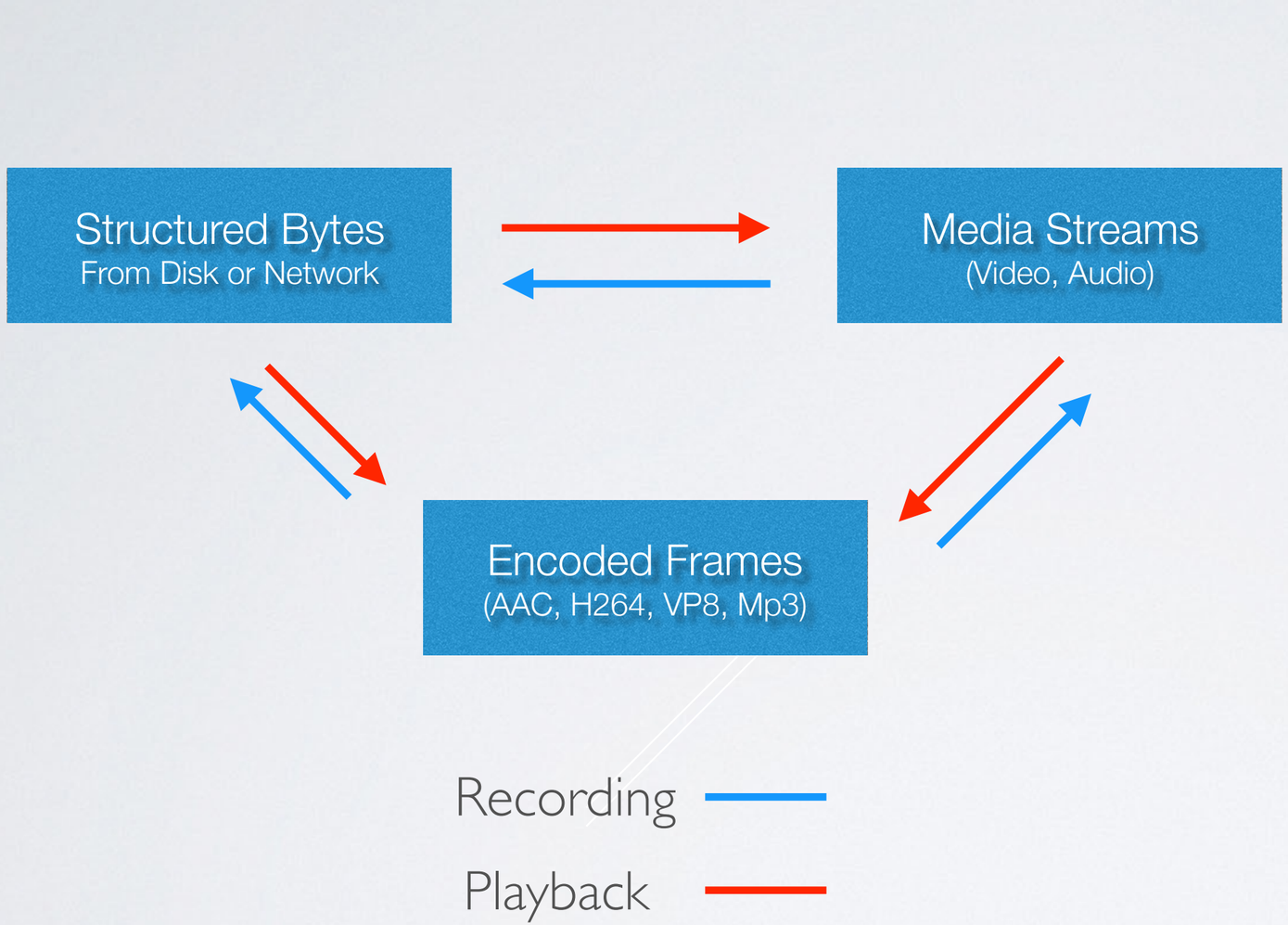
- Pros
  - API 1, great compatibility across different devices.
  - Good for most use cases.
  - Simple and have many examples.
- Cons
  - API was made for the Camera.app v1.
  - Can't make modifications to inputs easily.
  - It's slow to start and stop. (~700ms)

We want more

- What about adaptive playback?
- What about custom buffer control during playback?
- What about frame by frame recording?
- What about video effects for recording?
- What about….


Structured Bytes
From Disk or Network
Media Streams
(Video, Audio)
Encoded Frames
(AAC, H264, VP8, Mp3)
Recording
Playback


Structured Bytes
From Disk or Network
Media Streams
(Video, Audio)
Encoded Frames
(AAC, H264, VP8, Mp3)
Presentation
(Surface, AudioTrack)
Raw Image/Audio
(byte[], short[])
Sensors
(Camera, Mic)
Recording
Playback



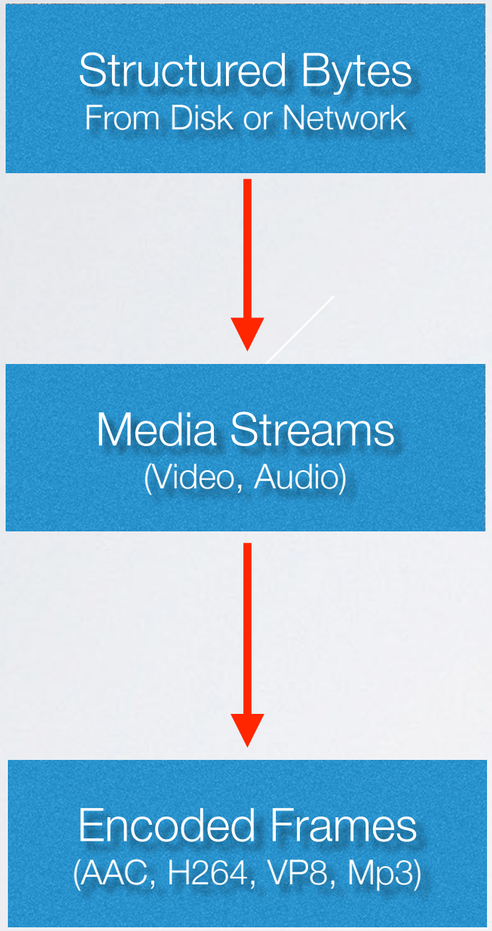
MediaExtractor

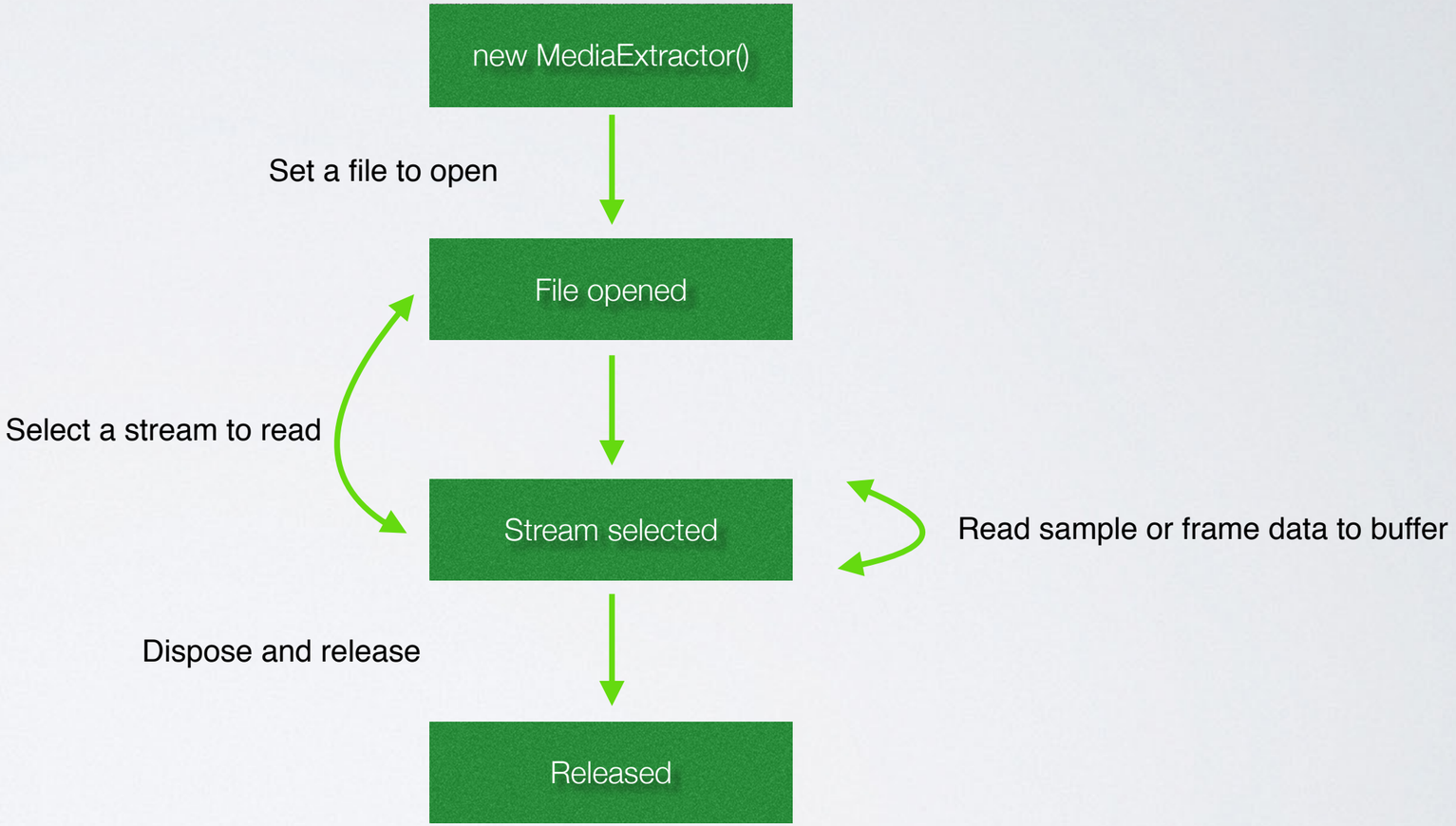
MediaMuxer

MediaMetadataRetriever

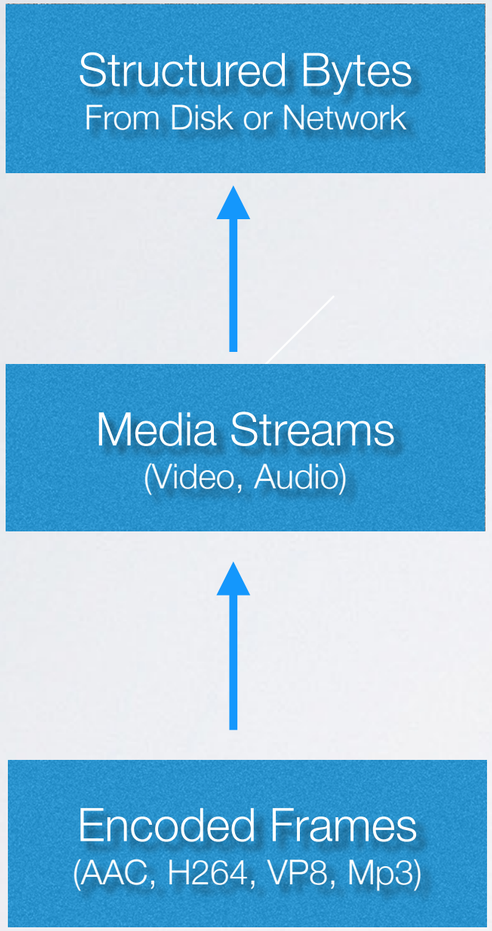
FFmpeg (libavformat)

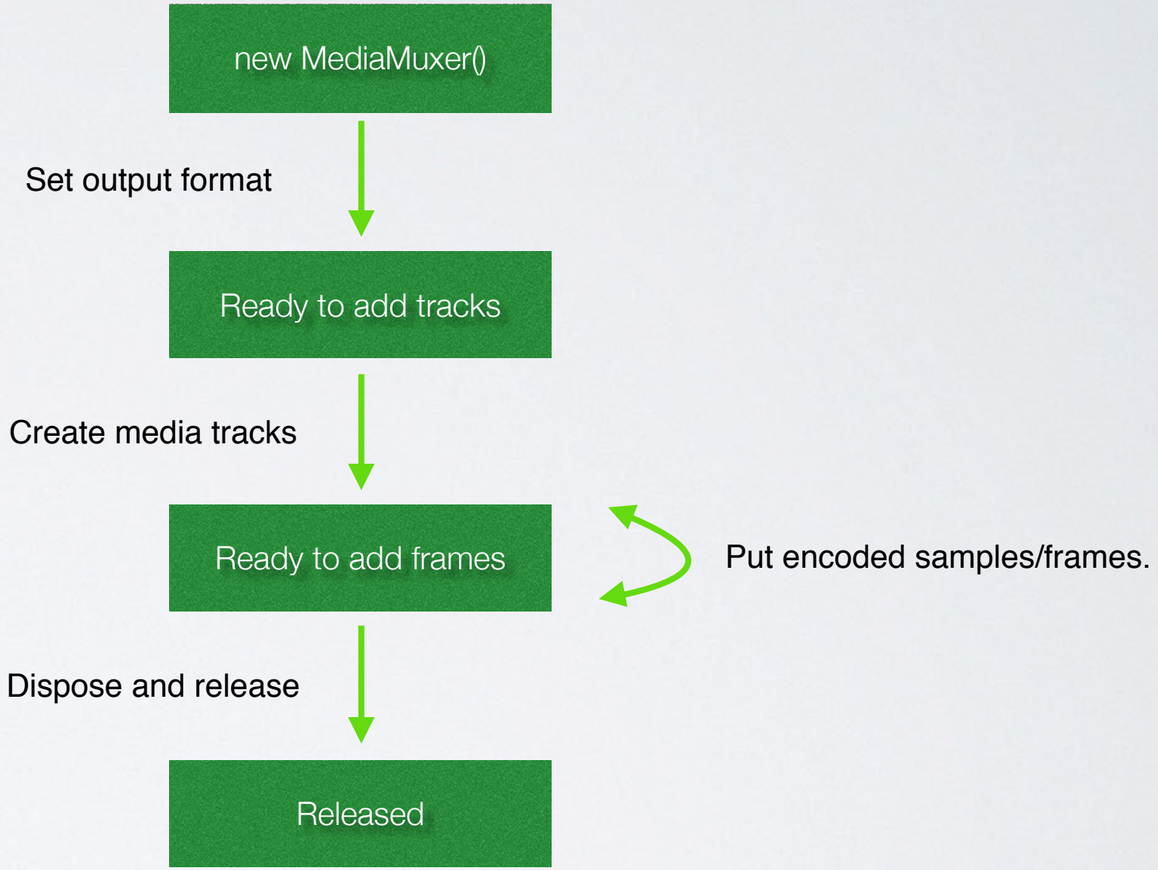
MediaExtrator (API 16)





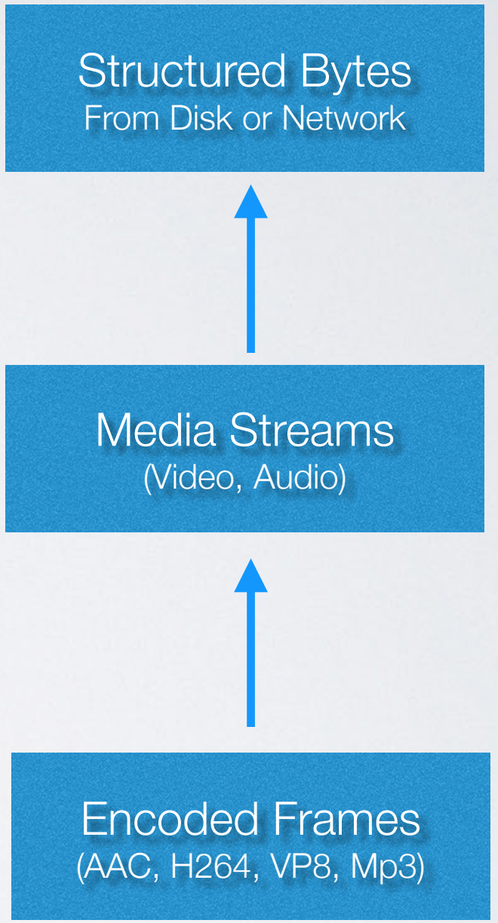
MediaMuxer (API 18)

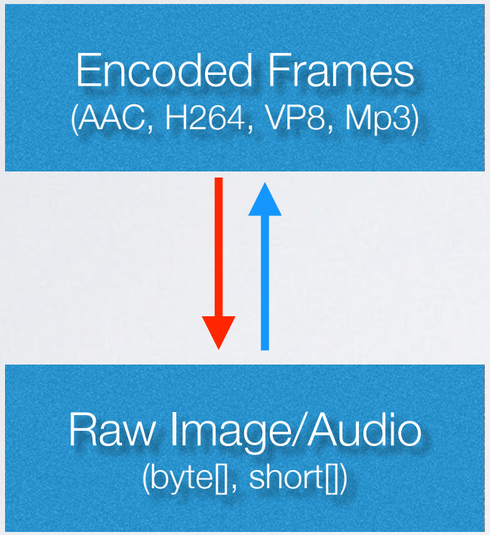


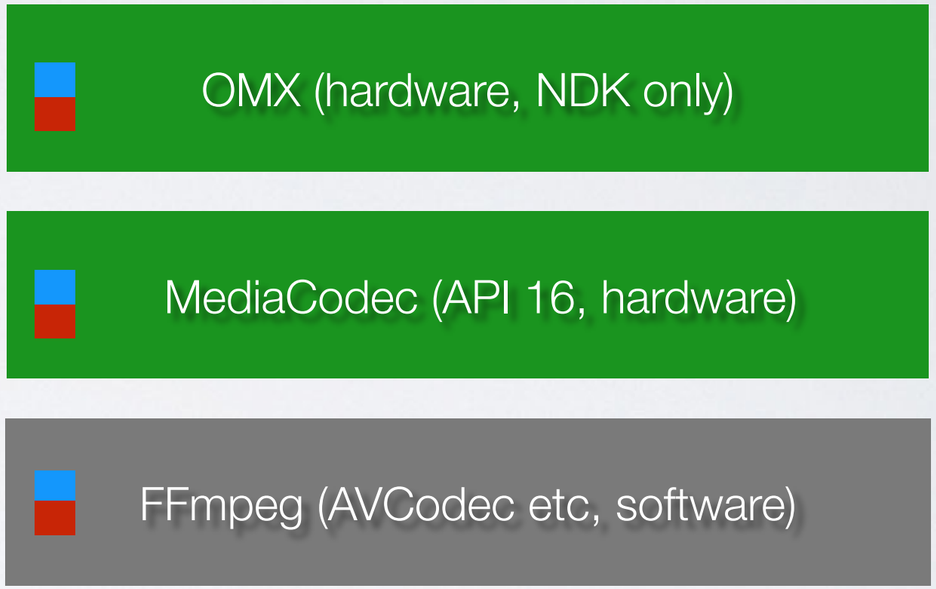


MediaMuxer (API 18)

- Muxing of elementary streams into specific container formats
- API 18 and only supports MP4.
- Suppors only1 audio stream and or 1 video stream.
- Better replacement?
  - MP4 Parser: https://code.google.com/p/mp4parser/

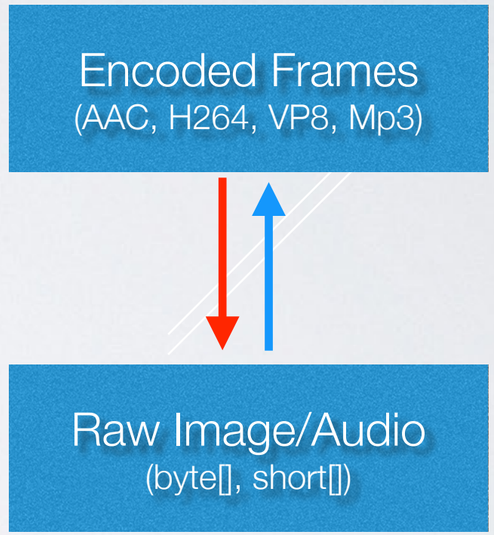



Encoded Frames
(AAC, H264, VP8, Mp3)
Raw Image/Audio
(byte[], short[])


OMX (hardware, NDK only)
MediaCodec (API 16, hardware)
FFmpeg (AVCodec etc, software)

MediaCodec

- API 16, but encoding input not usable until API 18.
- For all video frames, do things via Surface proxy.
- Create and configure MediaCodec.
  - Loop through inputs.
    - Wait until input buffer is ready to give an input.
    - Wait until output buffer is ready and read result. *
  - Dispose resources.



Goodies that put it together

Playback: ExoPlayer

- Open sourced by Google, very active development.
- Best playback and decoding example using SDK components.
- Loyalty free, good open source license.
- Handles all the timestamp and synchronizations for you.
- Super customizable.
  - Watch the Google IO 14 ExoPlayer talk!
- Vine and Youtube both use it.

Recording: JavaCV

- Open source library that demos OpenCV, FFMpeg, etc.
- FFMpeg.
  - More customizable than SDK components.
  - Good device compatibility via NDK components.
  - LGPL if you compile it yourself.
- https://github.com/bytedeco/javacv

Other Goodies

- Hidden packages inside AOSP.
  - com.android.mediadump (Video -> RGB via MediaPlayer)
  - android.filterpacks.* (image processing, recording via surface, and so much more.)
- Grafika Project: https://github.com/google/grafika
  - Various Android graphics & media hacks.
- MediaCodec examples: http://bigflake.com/mediacodec/
- All the great Google IO talks.

How Vine does it?

Playback

- MediaPlayer for regular playback when seek is not used.
  - Single MediaPlayer for GB on custom SurfaceViews.
  - Pooled MediaPlayers for ICS on custom TextureViews.
  - ListView video views are re-added when they bind to a new MediaPlayer.
- Highly modified ExoPlayer for playback when accurate seek timestamps are needed.
  - Jelly Bean only.

Recording

- Raw data -> Encoded data.
  - Video: Camera (converted to 480 x480 bitmap -> VP8)
  - Audio: Mic (PCM 16bit -> Vorbis)
  - or imported data from external source.
- Encoded data are stored in segments to allow clip editing.
  - Timestamps are relative within each segment.
- Encoded data -> Final output file.
  - WebM on device, WebM -> MP4 when distributed from server)

Import

- User select video from storage app (e.g. Gallery App).
- Use ExoPlayer to allow user to choose the accurate range and crop to import.
- Use MediaCodec and friends to decode the file in the selected range.
- Feed the decoded bytes into FFMpeg and to transcode as VP8/raw audio.
  - Each imported video will be their own segment.
  - Each segment will be in it's own frame rate, GCD is used at the end.

Audio from Mic

- AudioRecord to save the day.
- Continuous sample gathering in small chunks.
- THREAD_PRIORITY_URGENT_AUDIO.
- Avoid GC_ALLOC for world pauses.

Video from Camera API I

- PreviewCallback.onPreviewFrame(byte[] data)
- Set a callback via Camera.setOnPreviewCallback()
  - uses a new byte[] every frame -> GC -> lag.
- Use Camera.setOnPreviewCallbackWithBuffer() instead.
  - manual buffer management via Camera.addBuffer().

Super fast processing?

- Add 1 buffer to camera.
- Process the buffer before the next buffer needs to be added.
- Give buffer back to camera.
- Rinse and repeat.
- Achievable via hardware encoding only (MediaCodec).
- Avoid big buffers needed to be allocated.

Reality

- onPreviewFrame images are always landscape.
- Camera frames do not guarantee to come in fixed intervals.
- Images also have to be converted to a square before encoding for us.
- the byte[] that comes back is in NV21.
  - May need convert to RGBA for preprocessing.

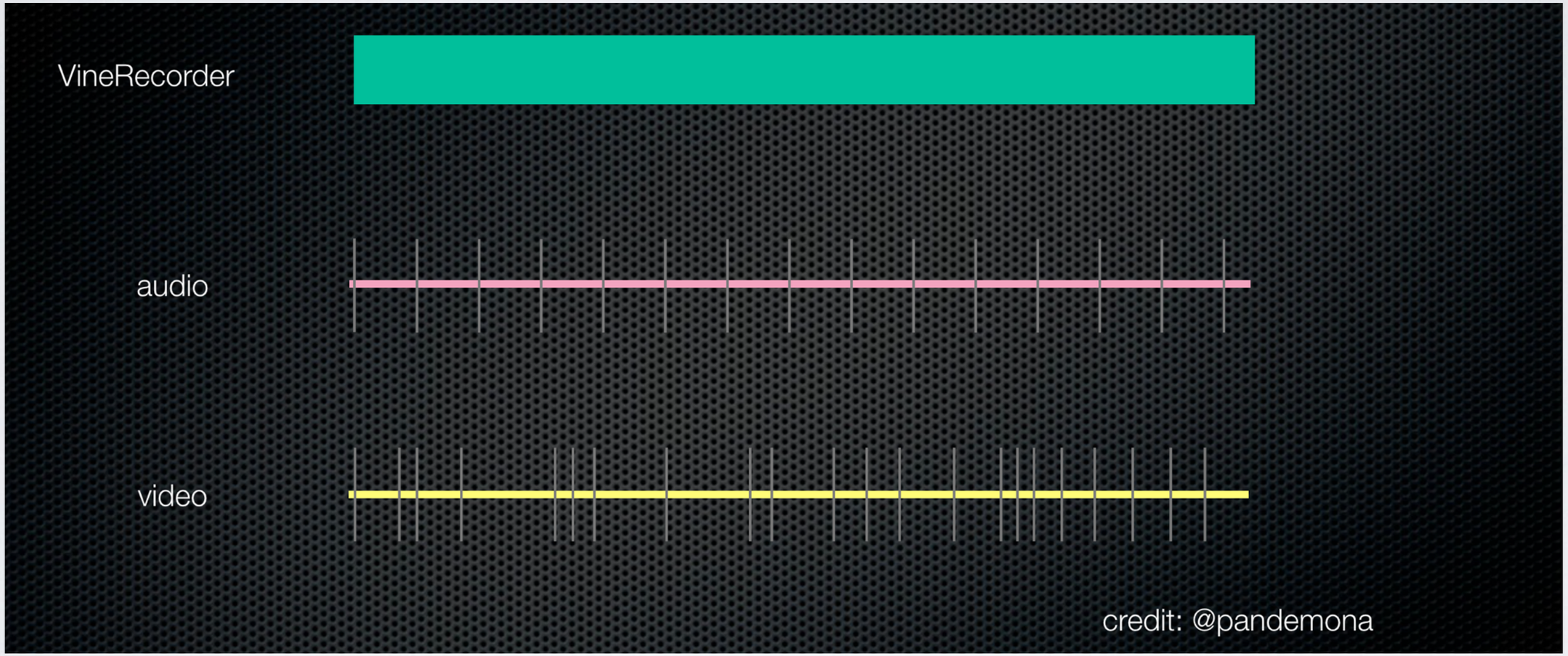
Road to encoded

- Color conversion via RenderScript
  - super optimized NDK code can achieve similar result 50x faster than do it via Java.
- Image manipulation (crop, rotation, inversion, etc.) via Matrix & Canvas.
- Encode to final video frame. (bottleneck)
- **Avoid GC: All intermediate objects are initialized before hand and reused.**
- Lots of pre-allocated buffers and objects.
  - largeHeap=true.
  - Separate dedicated process for more RAM budgets and avoid memory fragmentation.

Threads during recording

- Main thread with the final say and maintain locks and states.
- Thread 1 that takes frame from Camera -> dump into a buffer array.
- Thread 2 that takes data from that buffer array and get it encoded.
- Thread 3 that takes audio data and uses it for timestamp.
- Thread 4 that post updates to green process on the top.
- If recording via MediaCodec?
  - Few other threads to keep input buffer queue and output buffer queues busy.

AV Sync



Camera API V1 Only

AV Sync (Best Guess)

- Finger down:
  - start recording audio samples: keep track of how many samples we have recorded to get the timestamp for audio track.
  - start accepting video frames: (frame's videoTimestamp) = (current audioTimestamp) + (time difference since the audioTimestamp update).

AV Sync (Best Guess)

- Finger up:
  - stop accepting video frames and freeze videoTimestamp:
  - videoTimestamp = (last frame's timestamp) + (time per one frame)
  - wait until audioTimestamp >= videoTimestamp,
  - and then stop accepting audio samples.

New In L

- Camera API V2:
  - asynchronous frame gathering -> finally timestamped frames.
  - easier to do video effects via preview frames via live openGL tricks.
- Improved audio sampling.
  - Floating point audio sampling, audio clipping finally fixed.

Questions?

- AMA! :)
- Thank you!
- edison@edisonwang.com
- @wew
- We are hiring at Vine!