媒体播放器技术
原理、实现及其演进

李大龙，PCG 视频产品技术部

Oct. 2018, v1

Jun. 2019, v2

个人简介

- 2007：毕业于武汉大学国家多媒体软件工程技术研究中心
  - Data Hiding / Digital Watermarking for Videos
  - AVS Codec
  - Fast Transcoding From MPEG4 Visual to AVS Streams
- 2007 ~ 2008：多媒体研究室，PC QQ视频聊天
  - VCODEC：H.26L私有化、服务从PC到Mobile
- 2008 ~ 2010：桌面产品中心，QQ影音
  - 主岗是万能播放引擎（基于Win32 DirectShow架构）
  - 万金油：UI控件、字幕后台、版本升级运维、软文等
- 2010 ~ 2012：桌面产品中心，QQ电脑管家
- 2012 ~ 2013：手机QQ视频聊天
  - WebRTC + VP8
- 2013 ~ ：腾讯视频
  - 移动侧统一播放框架
  - 播放质量体系及其重要升级（HLS、HEVC等）
  - PCG技术负责人联席会议：播放器中台

NERCMS 国家多媒体软件工程技术研究中心 National Engineering Research Center for Multimedia Software

腾讯视频

什么是播放器

什么是“播放器”？

一种能把视频文件里面的视频或者/和音频数据解码出来的系统和装置

那不如叫“解码器”吧

解码完还需要音画同步和渲染呀

一种解码并音画同步后再渲染的系统和装置

。。。

还有各种协议、格式的解析！！

字幕！你英语0.1级的水平看得了好莱坞？！怕只懂 f*, sh*t, what’s your name吧

还有用户操作呐，播放、暂停、上一集/下一集、快进/快退、切换、跳转、截图，…（省略500字）

等等，还有高端大气上档次的音频特效，和酷炫狂拽吊炸天的图像滤镜

再加点弹幕…

播放器的「定义」

A media player is a computer program/software for playing multimedia files like audios, videos, movies and music. Media players commonly display standard media control icons known from physical devices such as tape recorders and CD players, such as play (▶), pause (⏸), fastforward, backforward, and stop (■) buttons. In addition, they generally have progress bars (or "playback bars") to locate the current position in the duration of the media file.

以解析媒体文件格式、解码音视频数据并提供信号渲染为主要功能，并提供必要性播放控制用户操作响应机制的软件系统和装置。

当我们谈播放器时，我们谈些什么

腾讯视频
播放器

腾讯视频
ID体系接入

视频广告

播放场景
适配

OSD（字幕、弹幕等）

版权水印

模块升级

云端配置

日志管理

增值产品 播放器

QQ空间 播放器

业务层
- 业务定制
- 不可统一性：ID体系、传输通道(HTTP/MSF/WNS)、交互行为定义等

质量监控

信息展示（状态、错误、消息）

播放接入

传输代理

控制策略

能力模型

播放器Adapter

PCG中台播放器
THUMB Player

Android/iOS
系统播放器

其他渠道
3rd 播放器

应用层
- 平台相关、业务无关
- 集成直接影响播放行为/体验的通用型模块
- 参考：IJKPlayer、ExoPlayer
- 业内开源的典型分界处

数据
传输
引擎

下载

缓存

P2P

标准协议

自定义IO

格式解析

DRM

解码

图像处理

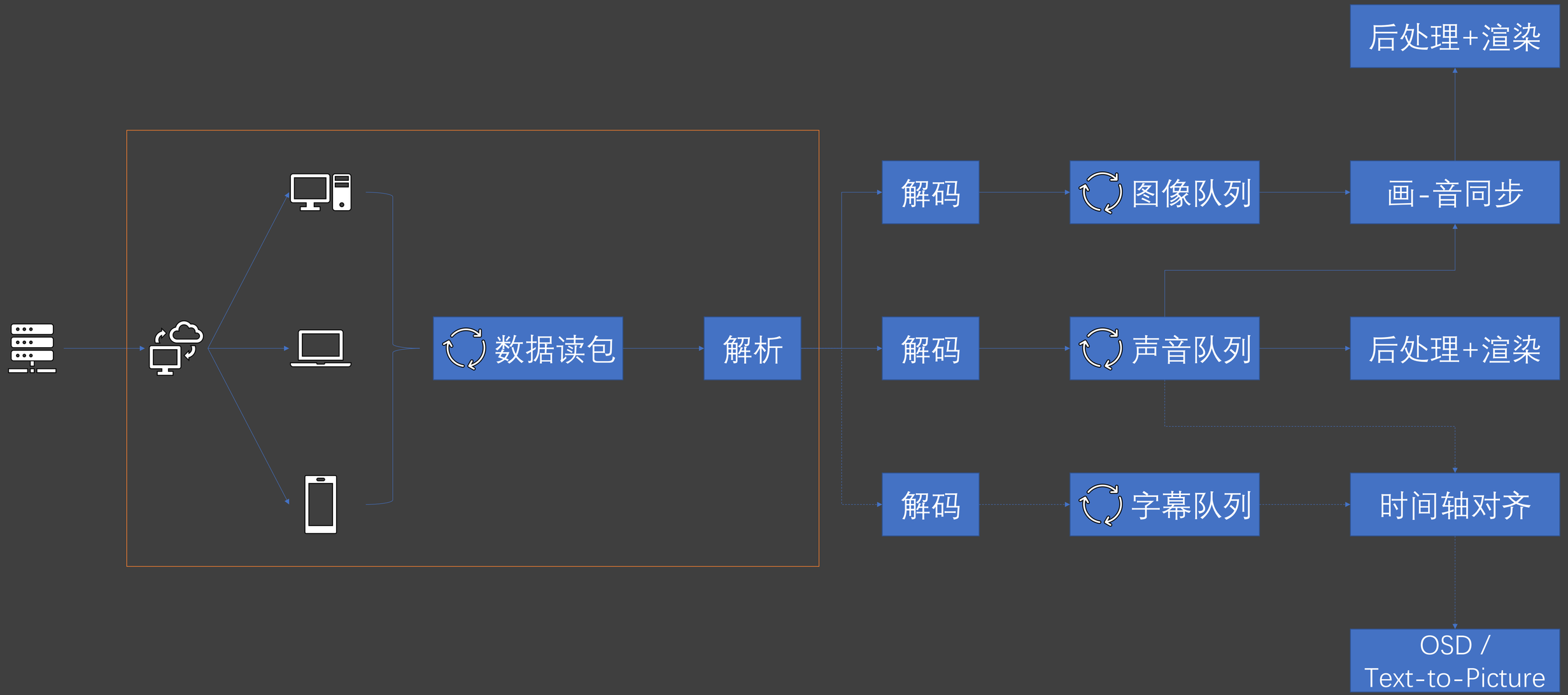
音频处理

渲染

播放
内核
引擎

内核层
- 跨平台的双引擎
- 完全的业务无关
- 参考：FFPlay、mplayer、VLC

播放器流程框图


后处理+渲染
解码
图像队列
画-音同步
数据读包
解析
解码
声音队列
后处理+渲染
解码
字幕队列
时间轴对齐
OSD /
Text-to-Picture

「传输协议」与「文件格式」

- 告知能力
- 提供机制
  - 格式→终端寻找解码器
  - 分辨率/采样率→终端创建输出设备
  - 时间戳→终端何时显示
  - 关键帧列表→终端快速定位文件

文件格式：
如何利用音
视频数据

传输协议：如何获取
音视频数据

- 数据如何分发、传输和访问
- 音视频“会话”：建立、维护、状态变更、终止
  - RTSP/RTMP：会话状态、能力交换
  - HTTP：无状态化、流式
  - ABS：清单+文件

Response headers (635 B)

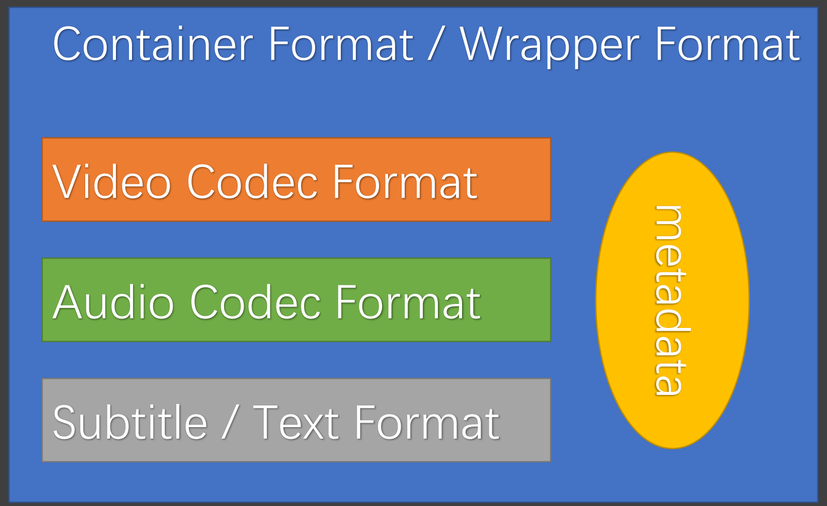
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: DNT,X-CustomHeader,Keep-Alive,...uested-With,If-Modified-Since
Access-Control-Allow-Methods: GET,POST,OPTIONS
Access-Control-Allow-Origin: *
Connection: keep-alive
Content-Length: 58635405
Content-Range: bytes 2359296-60994700/60994701
Content-Type: video/mp4
Date: Sun, 23 Dec 2018 09:08:17 GMT
ETag: "5bcf204c-3a2b48d"
Last-Modified: Tue, 23 Oct 2018 13:21:16 GMT
Server: NWSs
X-Daa-Tunnel: hop_count=1
X-NWS-LOG-UUID: 28b11153-99f3-4e71-8ed1-7390fc4312d5
X-Verify-Code: 70145bf4ef4d45182da156c9857644d6

Request headers (1.031 KB)

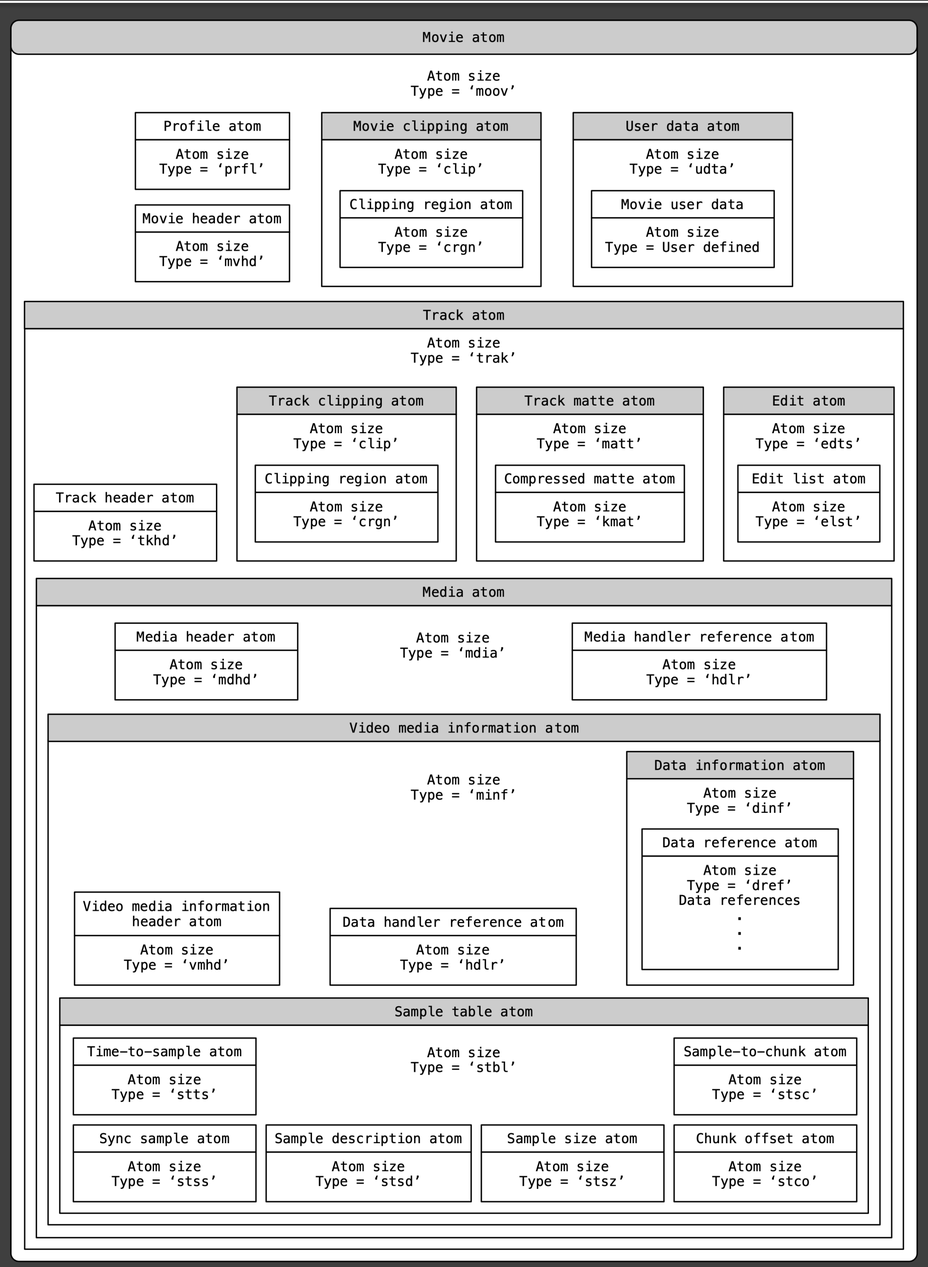
Accept: video/webm,video/ogg,video/*;q...q=0.7,audio/*;q=0.6,*/*;q=0.5
Accept-Language: zh-CN,en-US;q=0.7,en;q=0.3
Cache-Control: max-age=0
Connection: keep-alive
Cookie: pgv_pvi=3078301696; pgv_pvid=9... _qpsvr_localtk=1545556023340
Host: sec.video.qq.com
Range: bytes=2359296-
Referer: *https://sec.video.qq.com/p/tes...ic/fmp4/r0739ivm8sf.321005.mp4*
User-Agent: Mozilla/5.0 (Macintosh; Intel ...) Gecko/20100101 Firefox/66.0

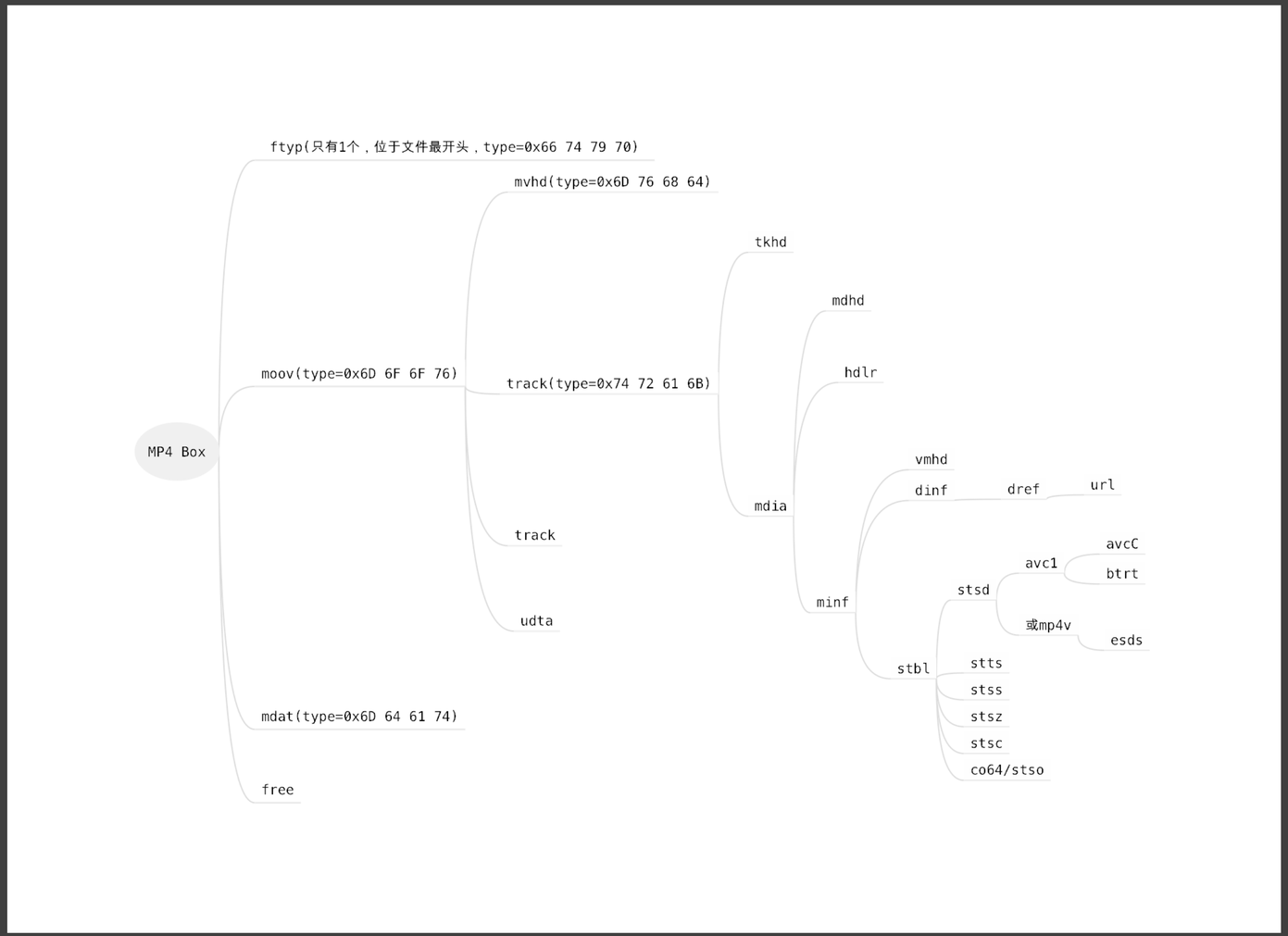
“格式”的层次定义

- 传输协议：以流媒体协议为例
  - C/S两端能力交互
  - AV会话通道的建立与维护
  - AV数据的定位与寻址
- 容器格式（封装格式）
  - 概要信息：
    - ✓ Stream counts：纯音频吗？
    - ✓ Duration：总时长、播放器的进度条
    - ✓ Video Codec /Res. /Bit-rate：创建相应的video decoder与render
    - ✓ Audio Codec /Sample-rate /Channels /Bit-rate：创建相应的audio decoder与render
  - AV数据的组织（interleaved-writing）
  - AV timestamp
  - Key frames index /list /location：播放seek的实施
- Video /Audio Codec Format
  - 角度：信源编码
  - 商业：遵循一定标准，一般情况与Container Format有弱绑定关系



容器格式举例：MP4（ISO/IEC 14496-12）





容器格式举例：MP4获取关键帧列表

Box类型						说明
ftyp						file type，表明文件类型
moov						metadata container，存放媒体信息的地方
	mvhd					movie header，文件的总体信息，如：时长、创建时间等
	track					track or stream container，存放视频/音频流的容器
		tkhd				track header，track的总体信息，如：时长、宽高等
		mdia				track media information container
			mdhd			media header，定义Time Scale，track需要通过Time Scale换算成自然时间
			hdlr			handler，表明本track类型，指明是video/audio还是hint
			minf			media information container，数据在子box中
				stbl		sample table box，存放时戳/偏移的映射关系表，数据在子box中
					stsd	sample descriptions
					stts	(decoding) time-to-sample，”时戳-sample序号”的映射表
					stsc	sample-to-chunk，sample和chunk的映射表
					stsz	sample size，每个sample的大小
					stsz2	compact sample size，另一种size的存储算法
					stss	sync sample table，可随机访问的sample列表（关键帧列表）
					stco	chunk offset，每个chunk的偏移，sample的偏移可根据其他box推算
					co64	64-bit chunk offset
mdat						media data container，音视频实体数据

MP4获取SPS/PPS

```
ADDR    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
------ -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
000000 00 00 00 2e 61 76 63 43 01 42 c0 1e ff e1 00 16
000010 67 42 c0 1e d9 02 04 68 40 00 00 03 01 40 00 00
000020 03 00 83 c5 8b 92 01 00 05 68 cb 83 cb 20
```

```
ADDR    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
------ -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
000000 00 00 00 2e 61 76 63 43 01 42 c0 1e ff e1 00 16
       ~~~~~~~~~~~
                  ~~~~~~~~~~~
                              ~~
                                 ~~
                                    ~~
                                       ~~
                                          ~~
                                             ~~~~~
000010 67 42 c0 1e d9 02 04 68 40 00 00 03 01 40 00 00
       ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
000020 03 00 83 c5 8b 92 01 00 05 68 cb 83 cb 20
       ~~~~~~~~~~~~~~~~~~
                         ~~
                            ~~~~~
                                  ~~~~~~~~~~~~~~
```

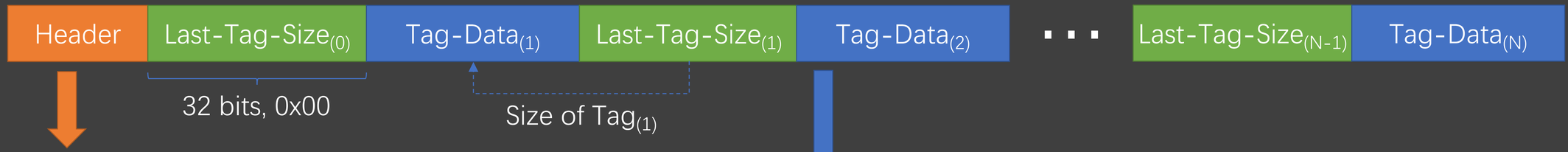
bytes	filed	说明
00 00 00 2e	BoxSize	'avcC' Box大小
61 76 63 43	BoxType	'avcC'
01	configurationVersion	通常为固定值0x01
42	AVCProfileIndication	0x42 (66), AVC 档次 (PROFILE_IDC)
c0	profile_compatibility	SPS兼容性信息
1e	AVCLevelIndication	30, 是Level。30即为Level3.0
ff	lengthSizeMinusOne	111111 + 11, NALUnit 长度中的字节数通常为4。也可以将其更改为1、2、4个字节。3是一个坏值。一个减去实际字节数。
e1	numOfSequenceParameterSets	111 + 00001, SPS 中的参数集数
00 16	sequenceParameterSetLength	0x0016(22). SPS NALU字节数
67 42 .. 8b 92	sequenceParameterSetNALUnit	22byte的SPS NALU内容
01	numOfPictureParameterSets	0x01. PPS NALU是参数集的个数
00 05	pictureParameterSetLength	0x05. PPS NALU字节数
68 cb 83 cb 20	pictureParameterSetNALUnit	5byte的PPS NALU内容

```
->BoxSize
->BoxType
->configurationVersion
->AVCProfileIndication
->profile_compatibility
->AVCLevelIndication
->lengthSizeMinusOne
->numOfSequenceParameterSets
->sequenceParameterSetLength

->sequenceParameterSetNALUnit

->sequenceParameterSetNALUnit
->numOfPictureParameterSets
->pictureParameterSetLength
->pictureParameterSetNALUnit
```

容器格式举例：FLV



字段	字段类型	字段含义
Signature	UI8	签名，固定为'F' (0x46)
Signature	UI8	签名，固定为'L' (0x4c)
Signature	UI8	签名，固定为'V' (0x56)
Version	UI8	版本，比如 0x01 表示 FLV 版本 1
TypeFlagsReserved	UB[5]	全为0
TypeFlagsAudio	UB[1]	1表示有audio tag，0表示没有
TypeFlagsReserved	UB[1]	全为0
TypeFlagsVideo	UB[1]	1表示有video tag，0表示没有
DataOffset	UI32	FLV header的大小，单位是字节

字段	字段类型	字段含义
TagType	UI8	tag类型 8：audio 9：video 18：script data 其他：保留
DataSize	UI24	tag body的大小
Timestamp	UI24	相对于第一个tag的时间戳（单位是毫秒） 第一个tag的Timestamp为0
TimestampExtended	UI8	时间戳的扩展字段，当 Timestamp 3个字节不够时，会启用这个字段，代表高8位
StreamID	UI24	总是0
Data	取决于根据TagType	TagType=8，则为AUDIODATA **TagType=9，则为VIDEODATA** TagType=18，则为SCRIPTDATAOBJECT

video frame

字段	字段类型	字段含义
FrameType	UB[4]	重点关注1、2： 1: keyframe (for AVC, a seekable frame) —— 即H.264的IDR帧； 2: inter frame (for AVC, a non-seekable frame) —— H.264的普通I帧； 3: disposable inter frame (H.263 only) 4: generated keyframe (reserved for server use only) 5: video info/command frame
CodecID	UB[4]	编解码器，主要关注 7（AVC） 1: JPEG (currently unused) 2: Sorenson H.263 3: Screen video 4: On2 VP6 5: On2 VP6 with alpha channel 6: Screen video version 2 7: AVC
VideoData	取决于CodecID	实际的媒体类型，主要关注7:AVCVIDEOPACKE 2: H263VIDEOPACKET 3: SCREENVIDEOPACKET 4: VP6FLVVIDEOPACKET 5: VP6FLVALPHAVIDEOPACKET 6: SCREENV2VIDEOPACKET 7: AVCVIDEOPACKE

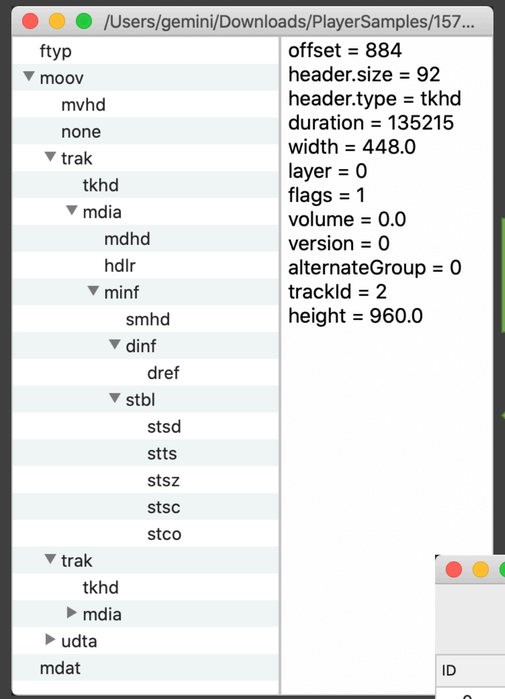
播放器WHY：为什么需要不同的「格式」

技术

- Codec兼容性
- 特性：streaming, Key Frame Index, etc.
- 扩展：DRM

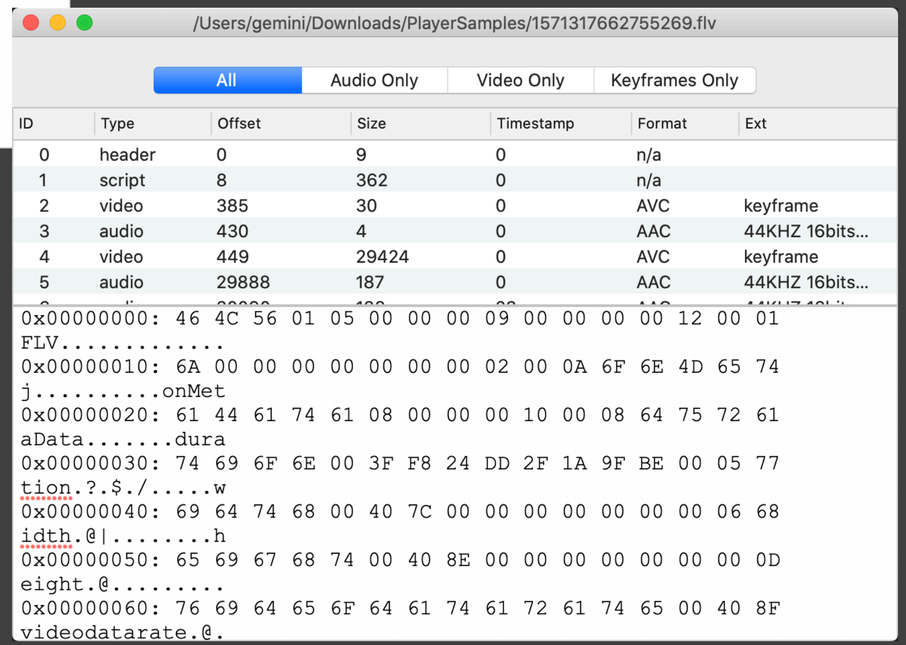
商业

- 配套生态：Video, Audio, System
- 产业链：采编、制作、分发、播放
- Adobe Flash vs. H5



- 无损转换

```
ffmpeg -i test.mp4 -c:v copy -c:a copy test.flv
```



传输/协议格式：“流媒体”的两种定义

广义

使音频和视频形成稳定和连续的传输流和回放流的一系列技术、方法和协议的总称。

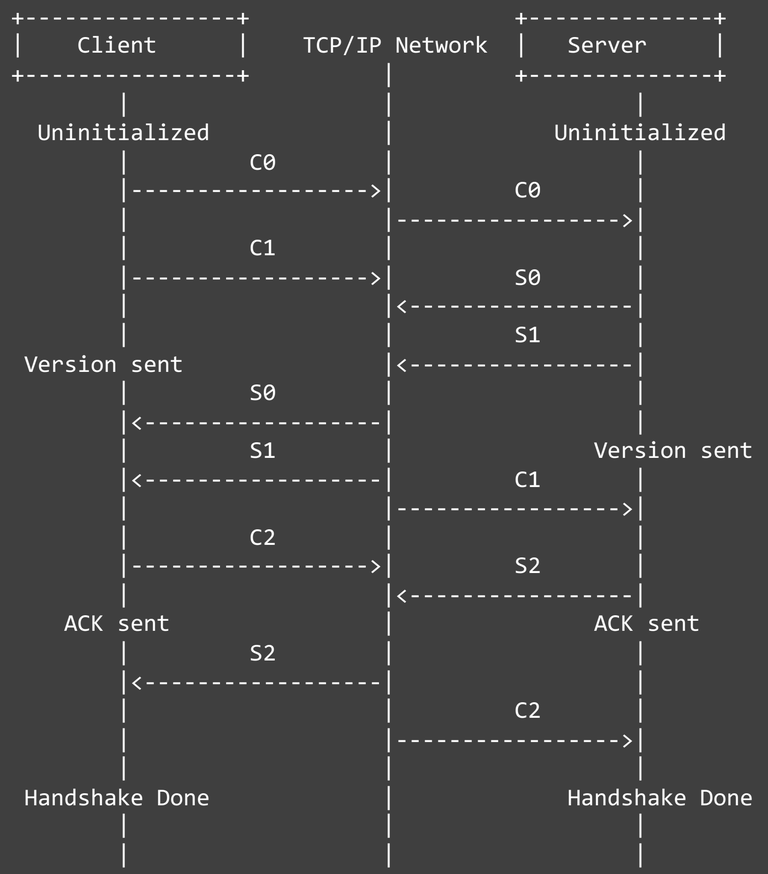
狭义

相对于传统的下载-回放方式而言，是一种从INTERNET上获取音频和视频等多媒体数据的新方法，能够支持多媒体数据流的实时传输和实时播放。

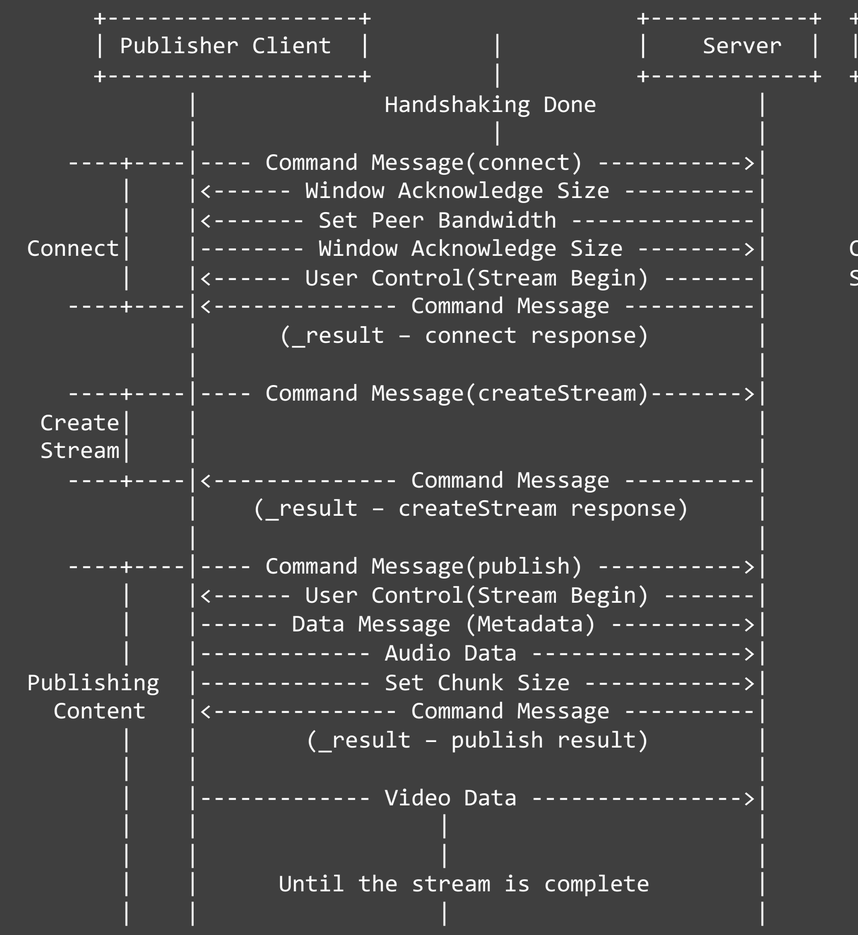
传输/协议格式：流媒体三代技术架构演进

	RTMP/RTSP/RTP	HTTP渐进式下载	自适应流媒体
服务器类型	流媒体服务器	Web服务器	Web服务器
客户端实现	复杂	很容易	较容易
系统部署	复杂	很简单	较简单
支持业务类型	多用于直播	多用于点播	直播、点播
启播延时	很低	很长	较长
客户端缓冲区	较小	较大	较小
网络带宽适应性	试服务器策略/能力	无	客户端触发、灵活切换
服务器故障保护	不支持	无	支持

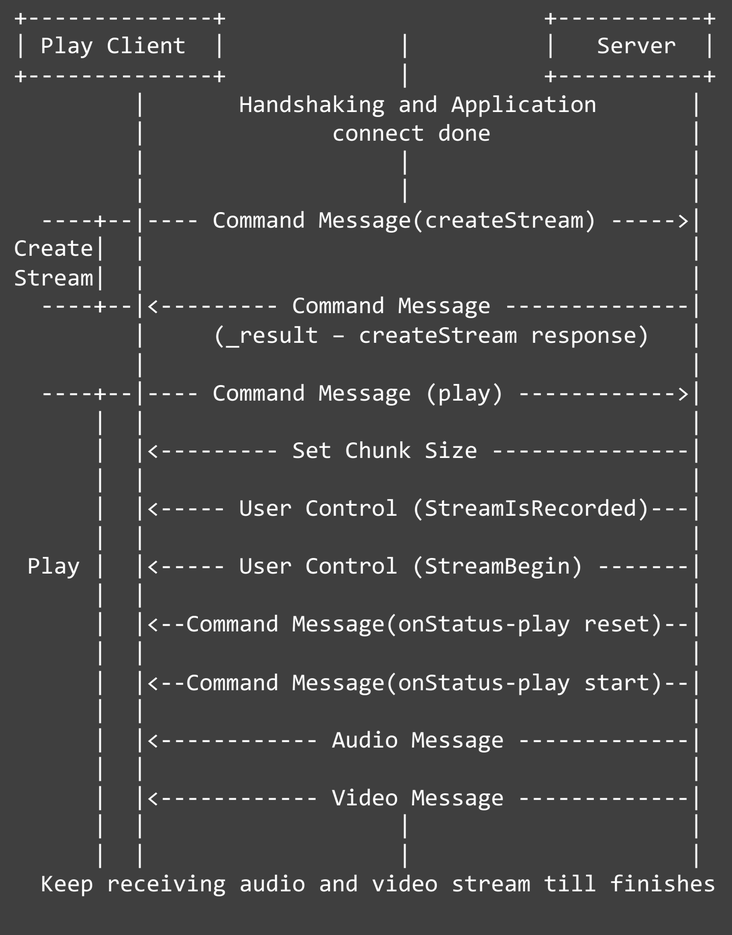
1st典型方案：RTMP



Representation of Handshake

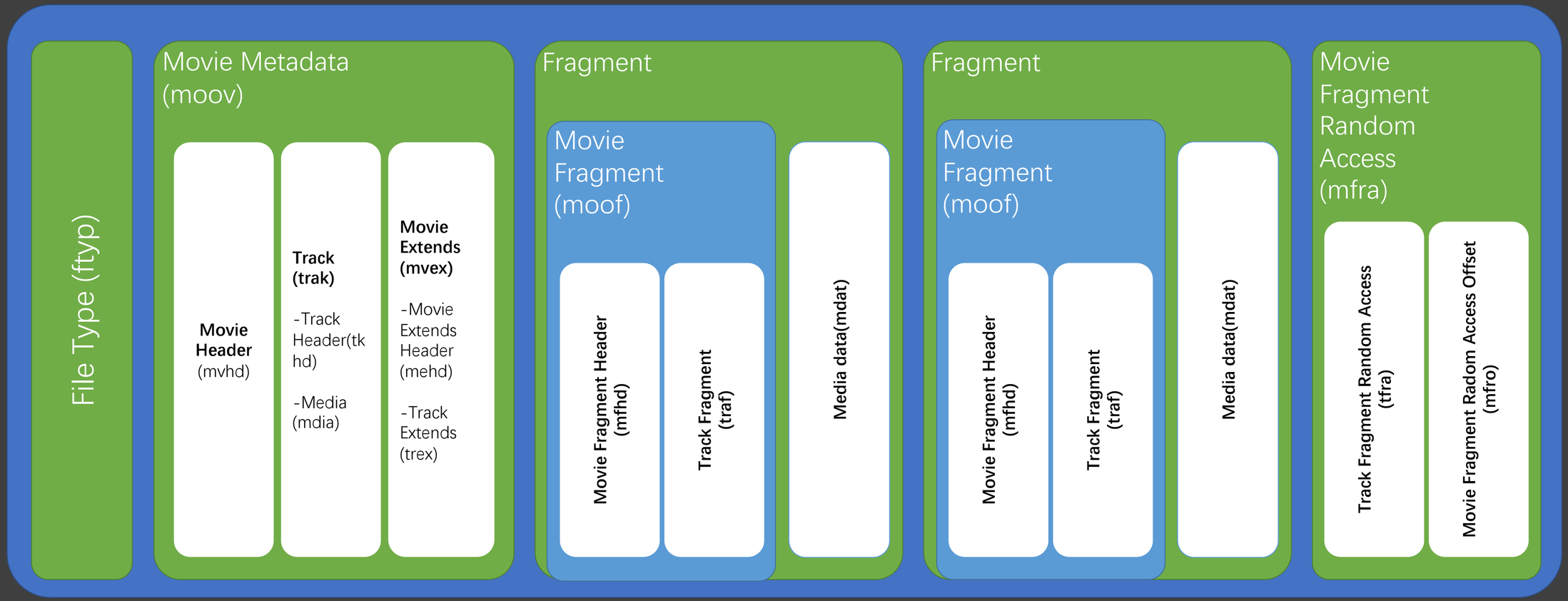


Message flow in publishing a video stream



Message flow in the play command

3rd典型方案：Microsoft Smooth Streaming



Smooth Streaming File Format

3rd典型方案：Apple HLS (HTTP Live-Streaming)

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=500000, RESOLUTION=720x480mid_video_index.M3U8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=800000, RESOLUTION=1280x720wifi_video_index.M3U8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=3000000, CODECS="avc1.4d001e,mp4a.40.5", RESOLUTION=1920x1080h264main_heaac_index.M3U8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=64000, CODECS="mp4a.40.5"aacaudio_index.M3U8
```

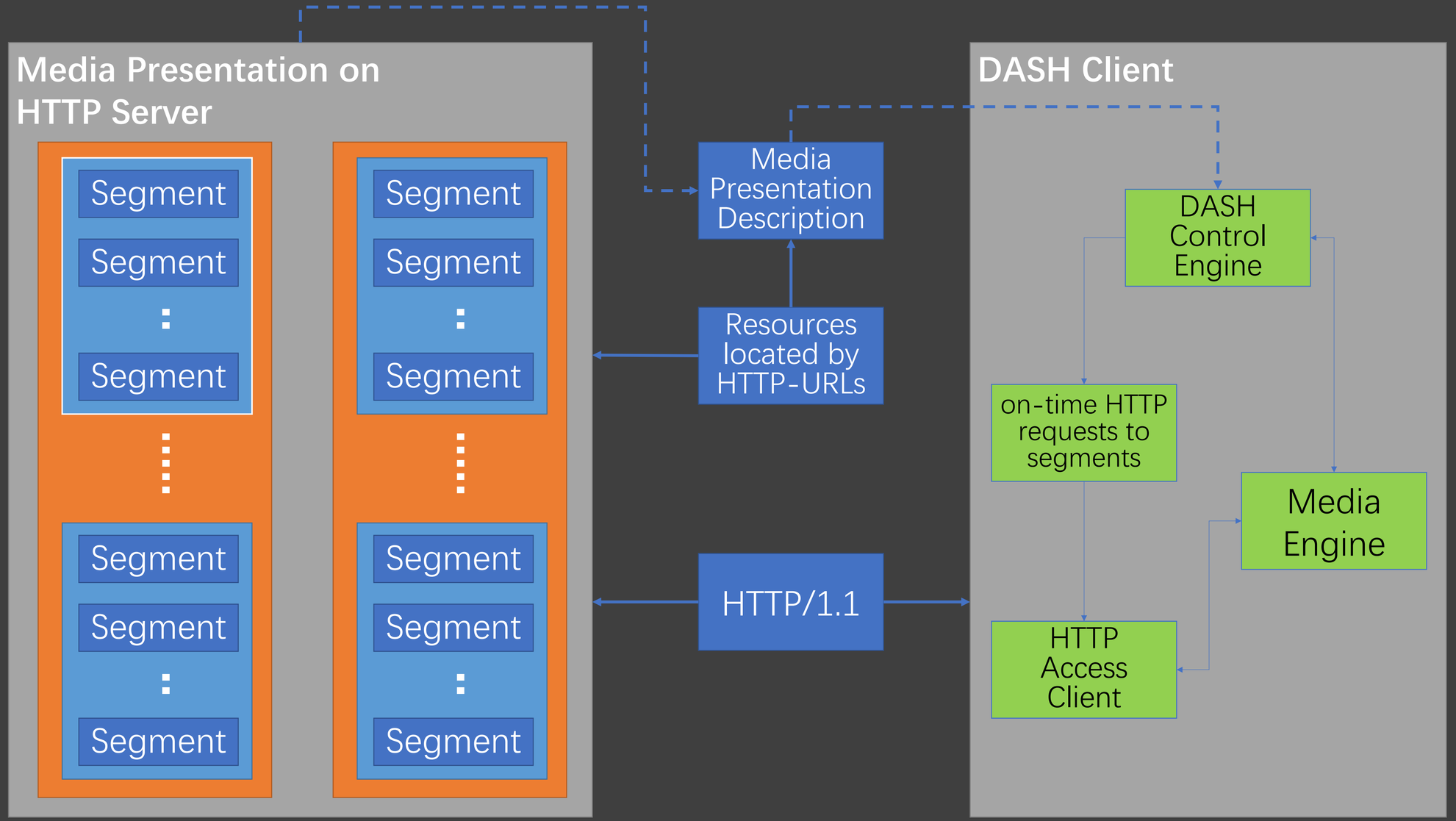
```
#EXT-X-VERSION:3
#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:1

# Old-style integer duration; avoid for newer clients.
#EXTINF:10,
http://media.example.com/segment0.ts

# New-style floating-point duration; use for modern clients.
#EXTINF:10.0,
http://media.example.com/segment1.ts
#EXTINF:9.5,
http://media.example.com/segment2.ts
#EXT-X-ENDLIST
```

- 使用MPEG TS为载体(188Byte定长分包)
- 通常5 ~ 10s切片(直播时延的症结)
- 为了保证自适应切换必须切片GOP对齐
- 每个切片都是一个Switching Point
- file seeking一般以切片为粒度，速度更快！

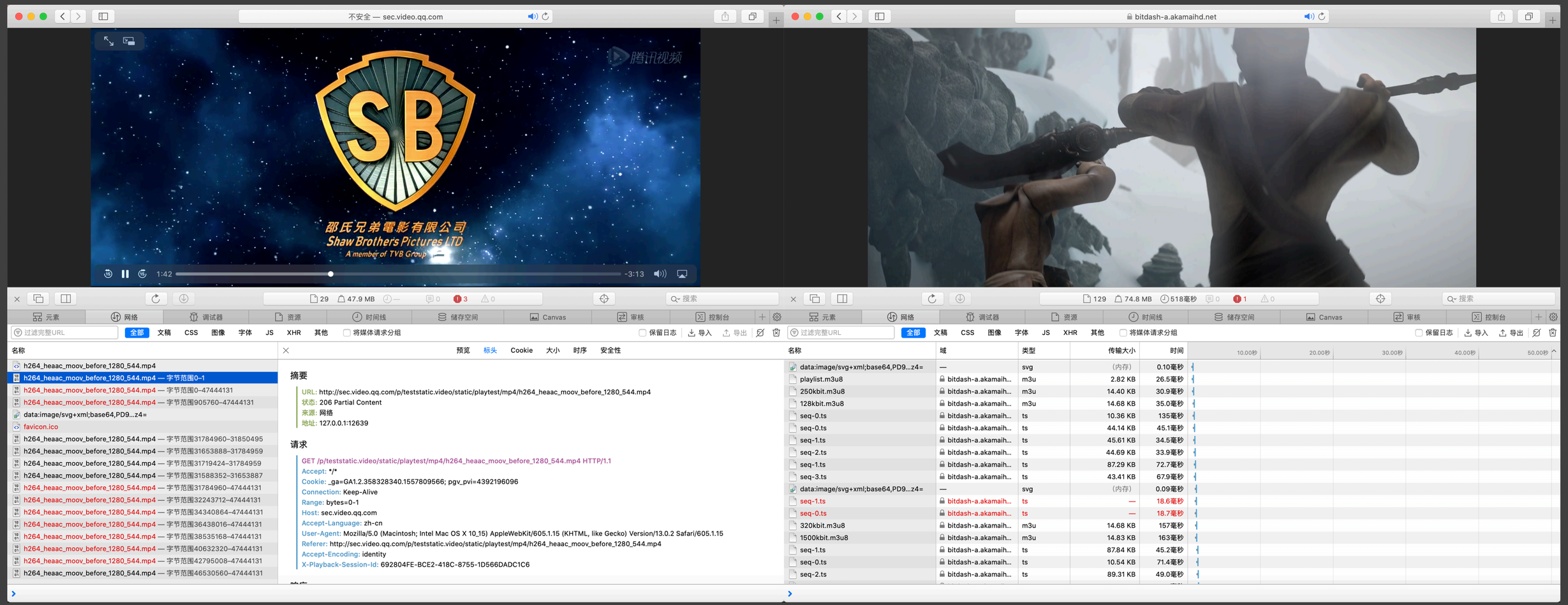
3rd典型方案：MPEG DASH (Dynamic Adaptive Streaming of HTTP)



3rd典型方案：对比小结

	Microsoft Smooth Streaming	Apple HLS	MPEG DASH
Data Description	Manifest(XML)	Playlist file(M3U8)	Media Presentation Description(XML)
Video Codec	H.264, VC-1	H.264	any
Audio Codec	AAC, WMA	AAC(HE,LC), MP3, AC3	any
File Format	fragment MP4	M2TS	MP4, M2TS
Segment Length	2s	5~10s	not specified
Deployment on ordinary Servers	no	yes	yes
Client Compatibility	only Windows	• Apple devices • Android 4.0+ • OTT devices • H5, maybe	ISO devices

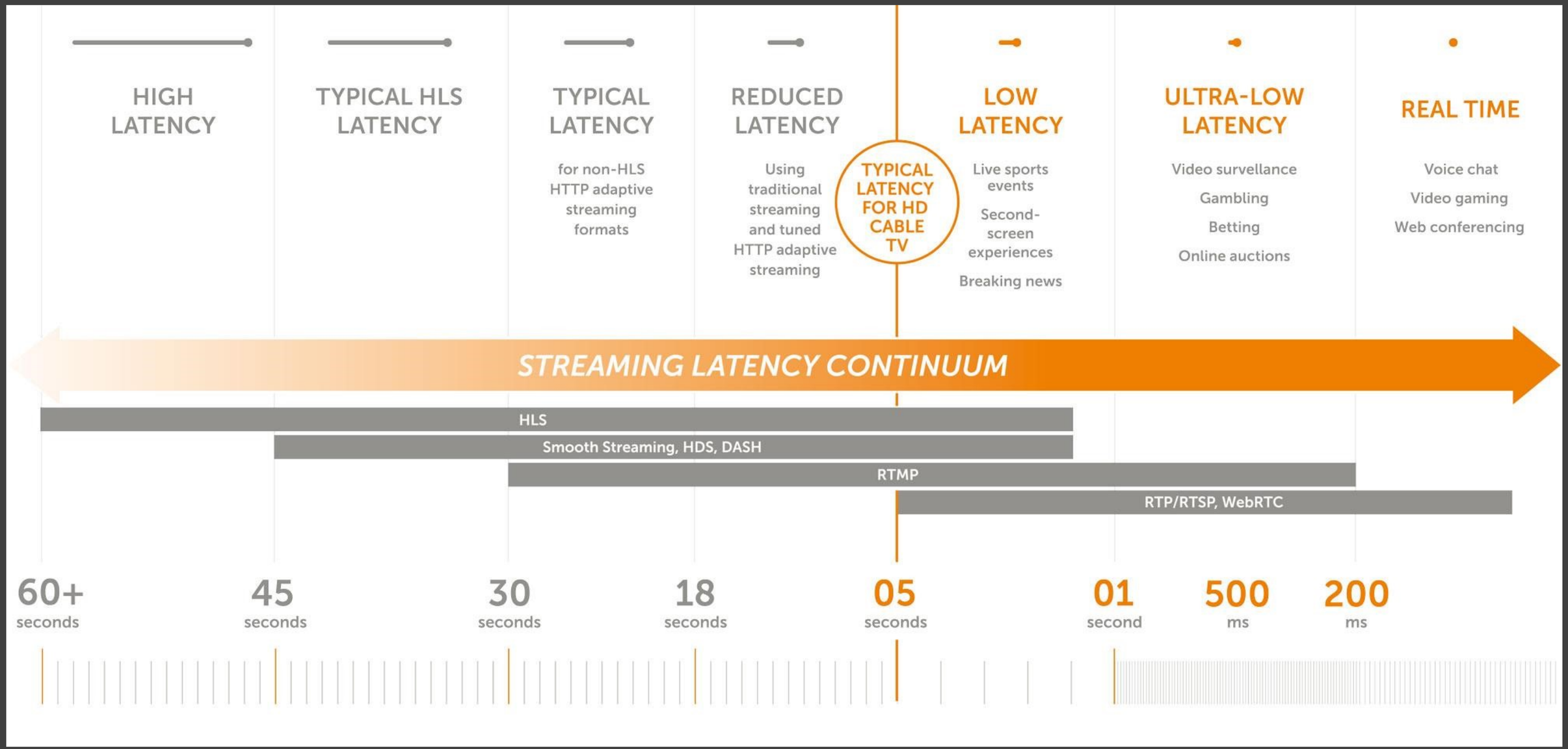
典型方案：2nd➔3rd的烦恼



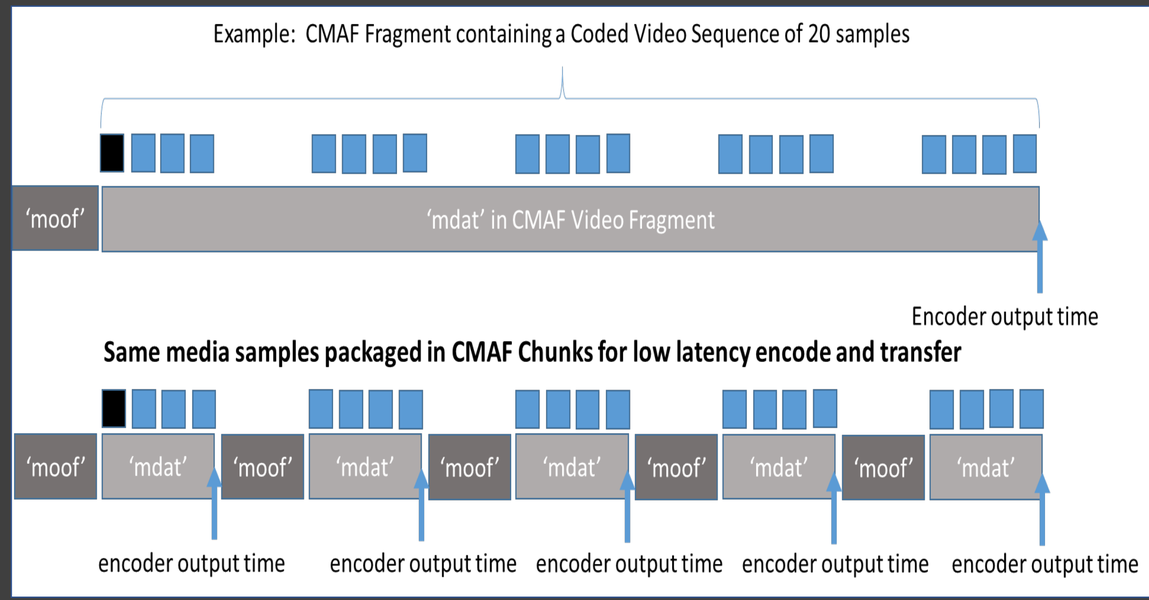
- 多了交互：获取m3u playlist
- 时延：依赖切片、播放器策略
- 多次请求、文件切片分布在不同CDN
- payload耗损：playlist, TS 188-bytes填充字节

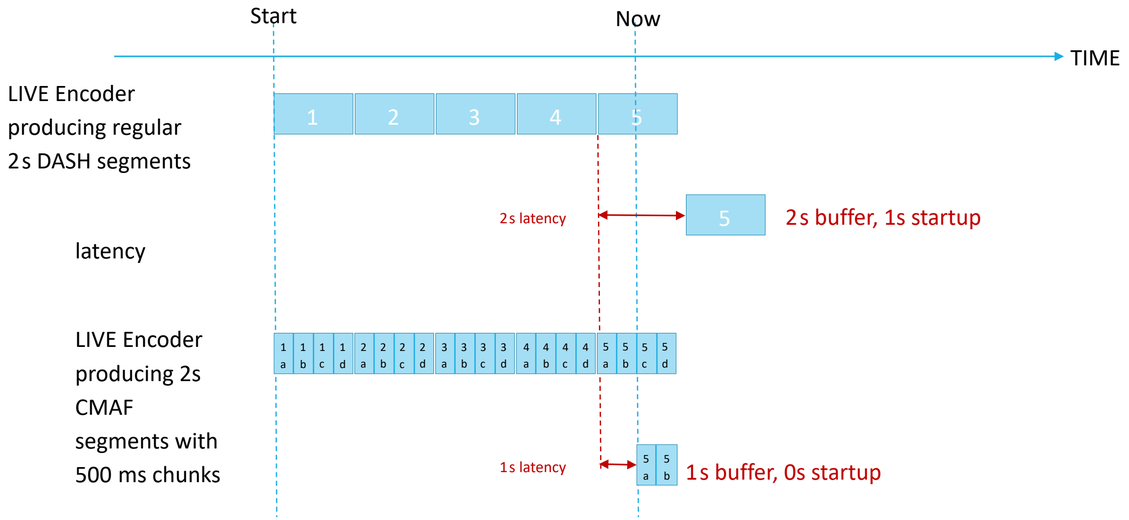
- 规范：前提是明确、简单，如何标注key frame？
- 整合：广告、DRM
- 自适应：网络变数下的自救

演进——来自「时延」的推动


HIGH LATENCY
TYPICAL HLS LATENCY
TYPICAL LATENCY
for non-HLS HTTP adaptive streaming formats
REDUCED LATENCY
Using traditional streaming and tuned HTTP adaptive streaming
TYPICAL LATENCY FOR HD CABLE TV
LOW LATENCY
Live sports events
Second-screen experiences
Breaking news
ULTRA-LOW LATENCY
Video surveillance
Gambling
Betting
Online auctions
REAL TIME
Voice chat
Video gaming
Web conferencing
STREAMING LATENCY CONTINUUM
HLS
Smooth Streaming, HDS, DASH
RTMP
RTP/RTSP, WebRTC
60+ seconds
45 seconds
30 seconds
18 seconds
05 seconds
01 second
500 ms
200 ms

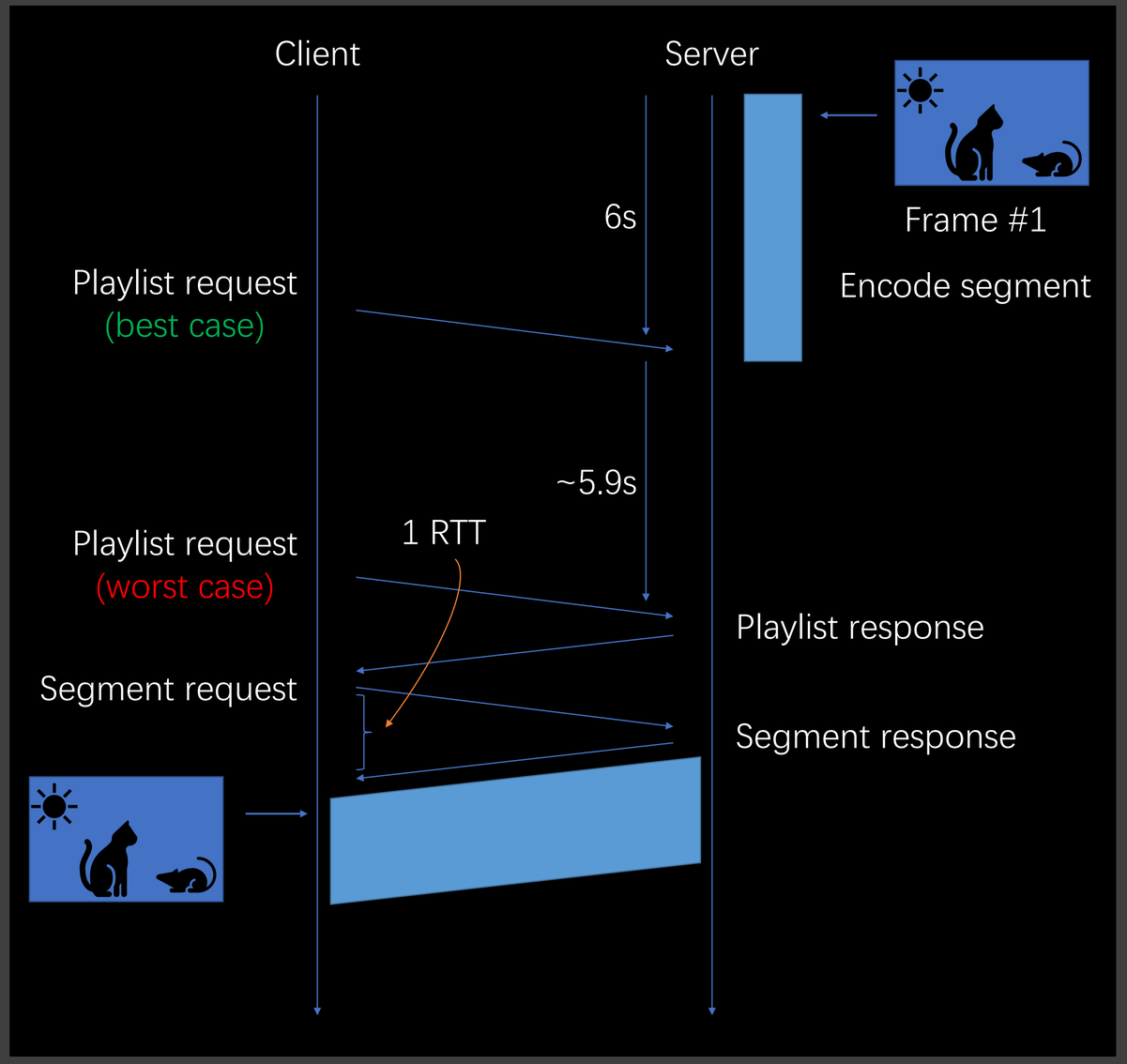
演进——CMAF

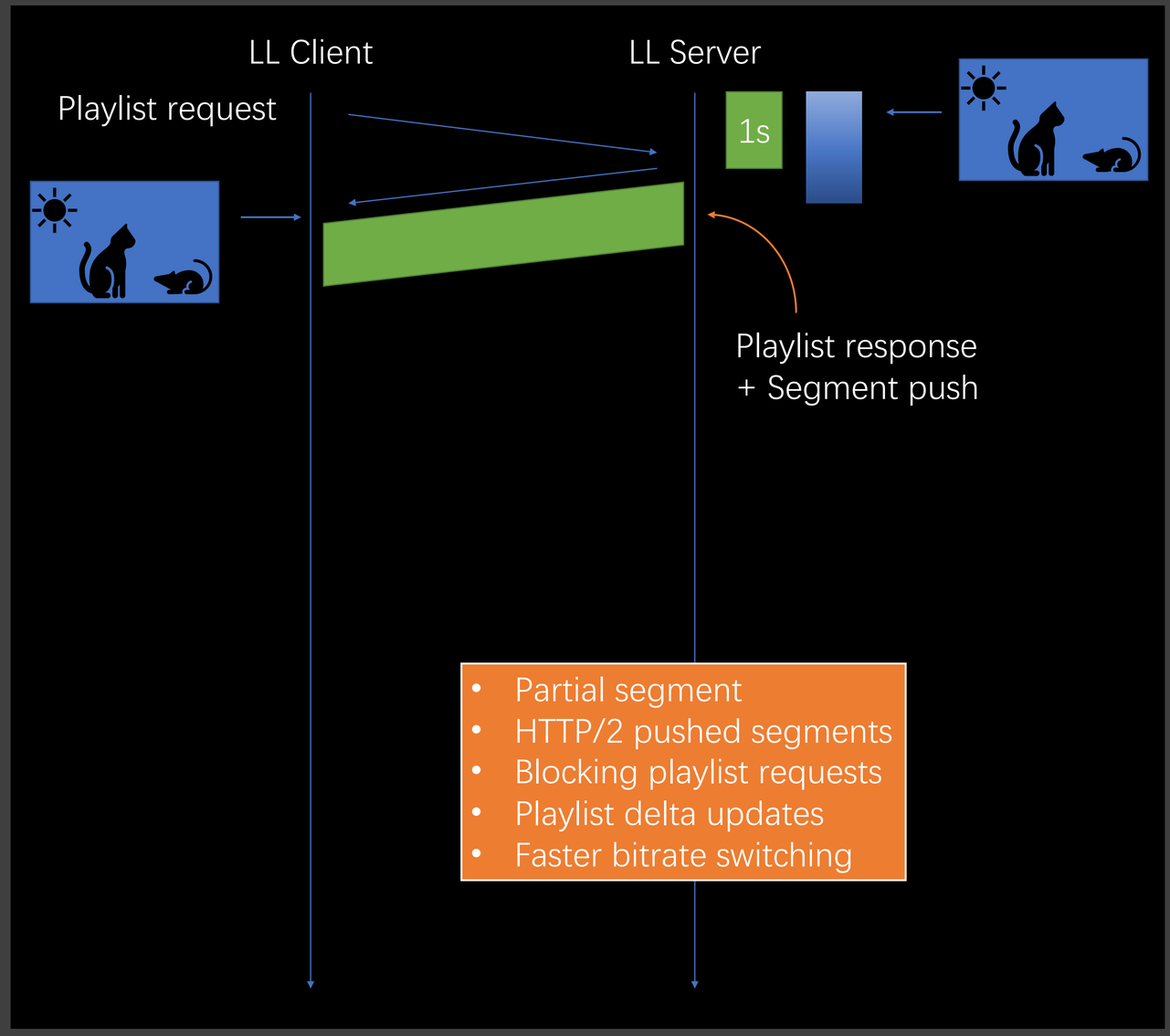




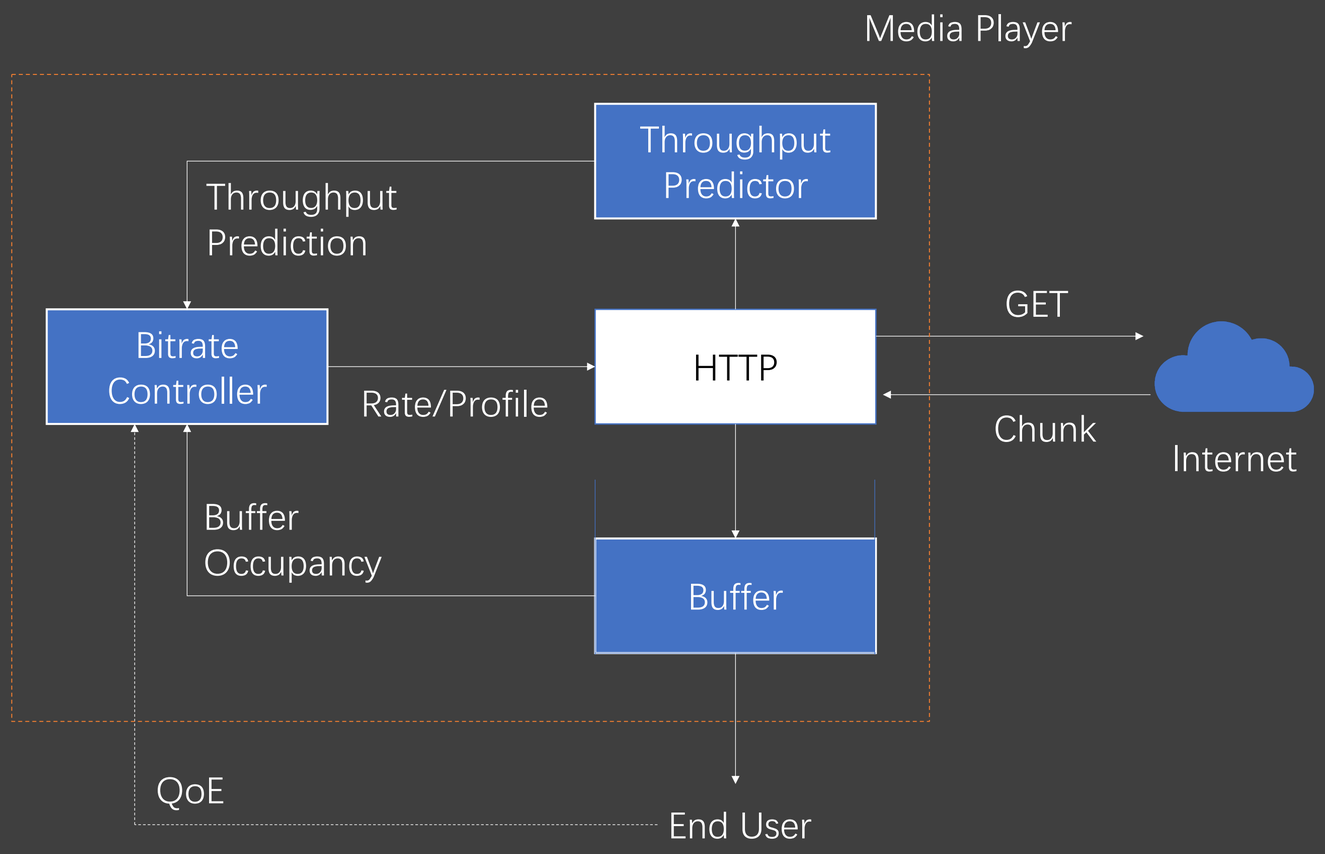
- 2016年Apple宣布支持
- HLS/MPEG-DASH = Common Media Application Format
- 自身不直接减少延迟，但提供低延迟模式
- 扩展：与CENC-ISO/IEC 23001-7标准关联，组成DRM系统

演进——Apple Low-latency HLS





ABS的实施

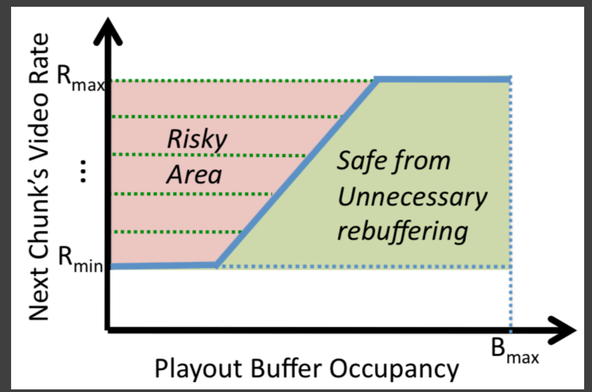


一般意义下的衡量指标

- 尽量减少数据二次缓冲（rebuffering/stalling）
- 尽可能传输高质量视频码流（更高清晰度、更高码率）
- 减少初始加载延时
- 避免过度频繁地码流档次切换，或者过大质量变化的切换

ABS的实施

- Rate-based algorithms
  - ➢ Picking a Video Streaming Rate is Hard
- Buffer-based algorithms
  - ➢ A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service
- AI-based algorithms
  - ➢ Neural Adaptive Video Streaming with Pensieve

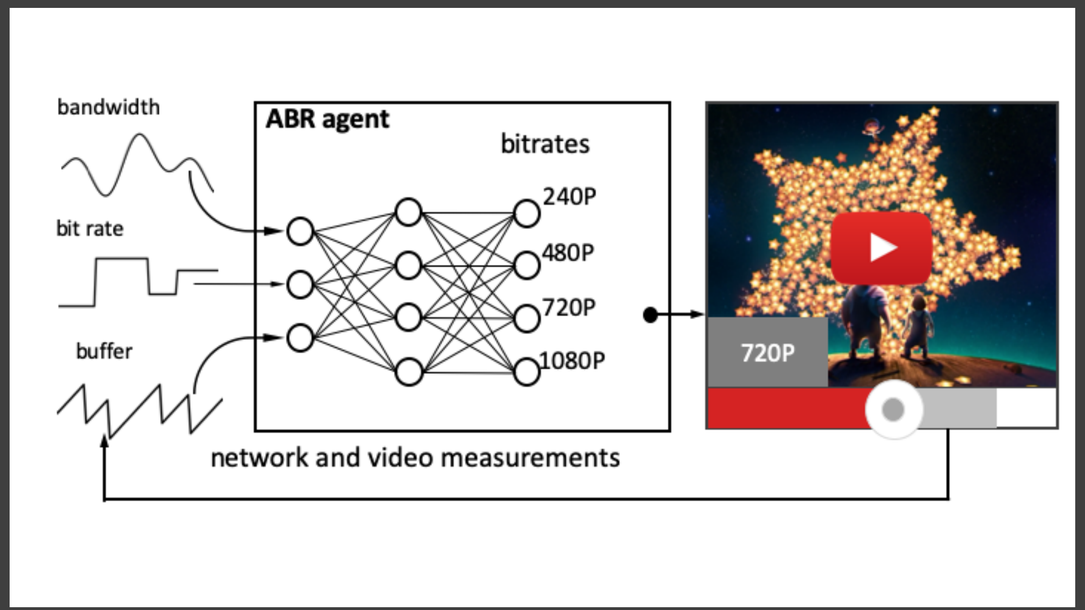


PENSIEVE

- State：吞吐量、下载时间、块大小、buffer大小、剩余chunk数量、上次策略
- Action：码率档位（bit-rate profile）
- Policy：神经网络
- Reward：单步QoE函数
- Training：A3C算法快速异步更新
- Environment：仿真器模拟客户端下载行为，高加速比
- Dataset：真实trace或者人工合成trace

$$QoE = \alpha \sum_{n=1}^{N} q(R_n) - \beta \sum_{n=1}^{N} T_n - \gamma \sum_{n=1}^{N-1} |q(R_{n+1}) - q(R_n)|$$

质量　卡顿　切换

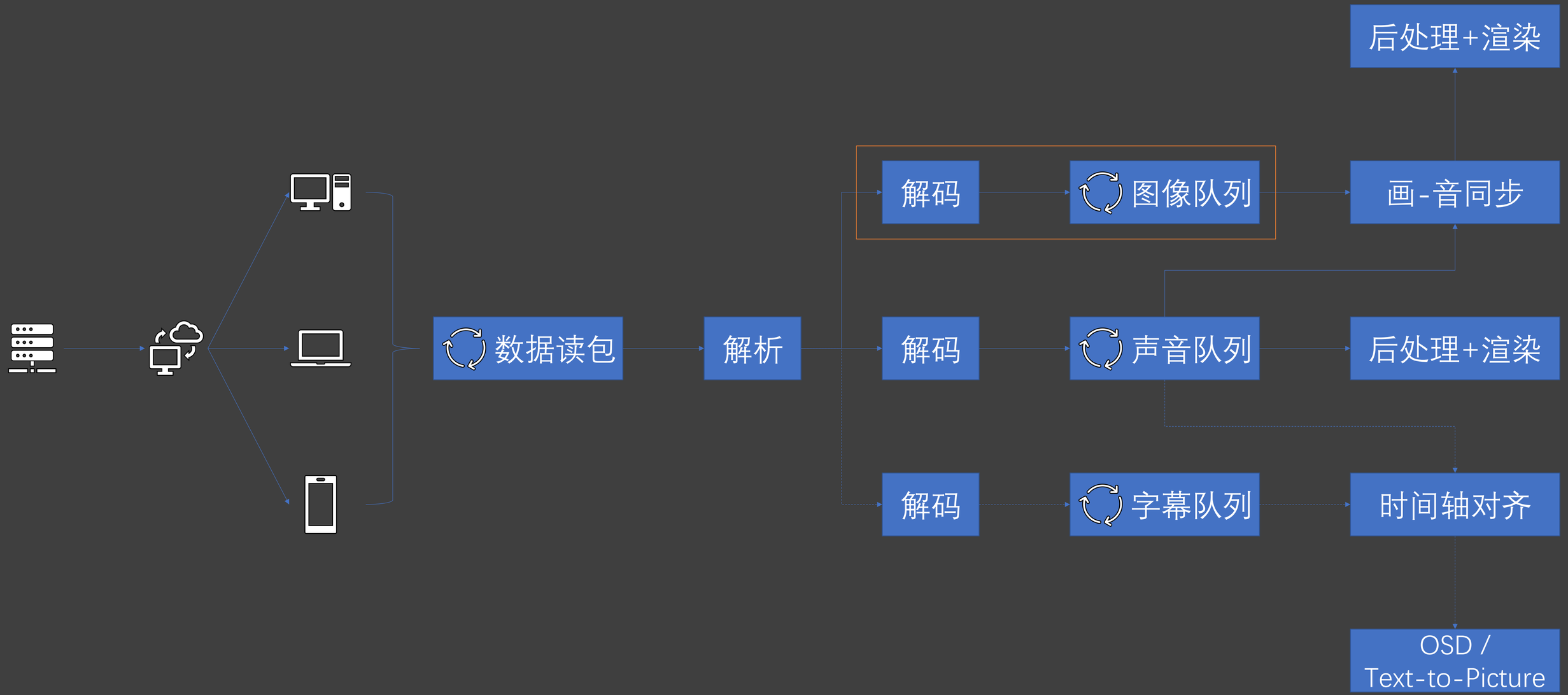


Demo

Pensieve

MPC

播放器流程框图


后处理+渲染
解码
图像队列
画-音同步
数据读包
解析
解码
声音队列
后处理+渲染
解码
字幕队列
时间轴对齐
OSD /
Text-to-Picture

视频格式基础：色彩空间

- RGB
- 每个颜色sample由3个数字组成，以表征红、绿、蓝三基色的相对比例
- 显示设备（CRT/LCD）根据每个像素点红、绿、蓝三个分量的强度分别进行展示

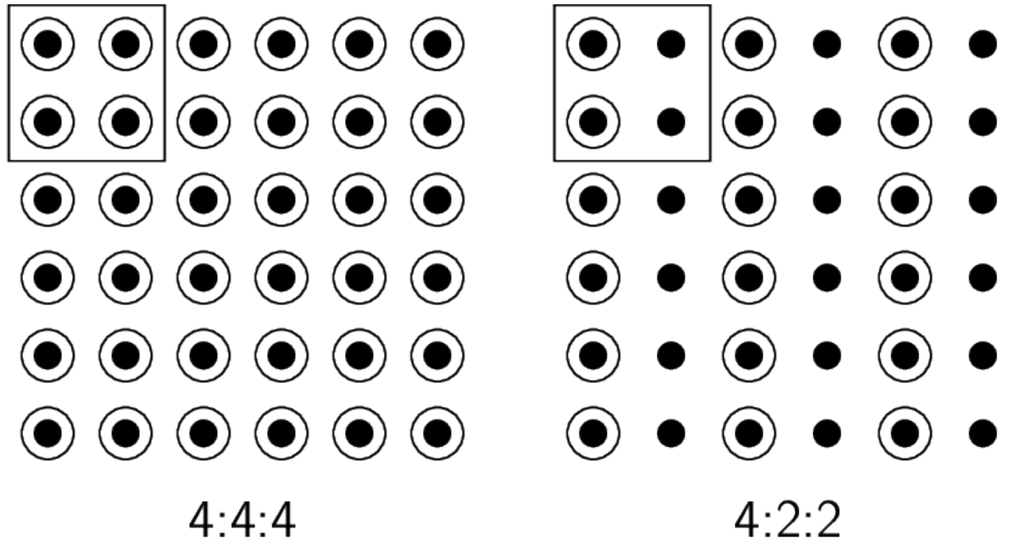
- YCbCr/YUV
- 人类视觉系统(Human Visual System, HVS)对颜色的敏感度较之对亮度的敏感度要低
- 在RGB色彩空间中，三基色对视觉的重要程度是相同的
- YCbCr色彩空间由1个亮度分量Y和2个色度分量Cb/Cr组成，利用人眼敏感度的差异，存储时可以通过降低色度分量的分辨率来减少数据量

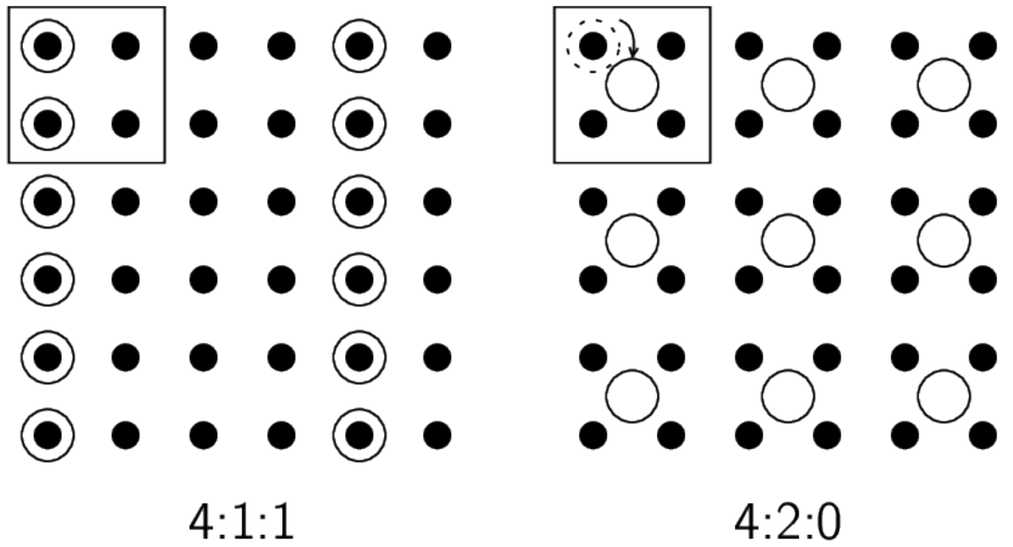
$$\begin{cases} Y = 0.299R + 0.587G + 0.114B \\ Cb = 0.564(B - Y) \\ Cr = 0.713(R - Y) \end{cases}$$

- 精度：数学上可逆、但工程上往往采用快速算法
- 速度：可能有采样（插值）操作
- 亲和度：Codec、Render

视频格式基础：YUV常见格式

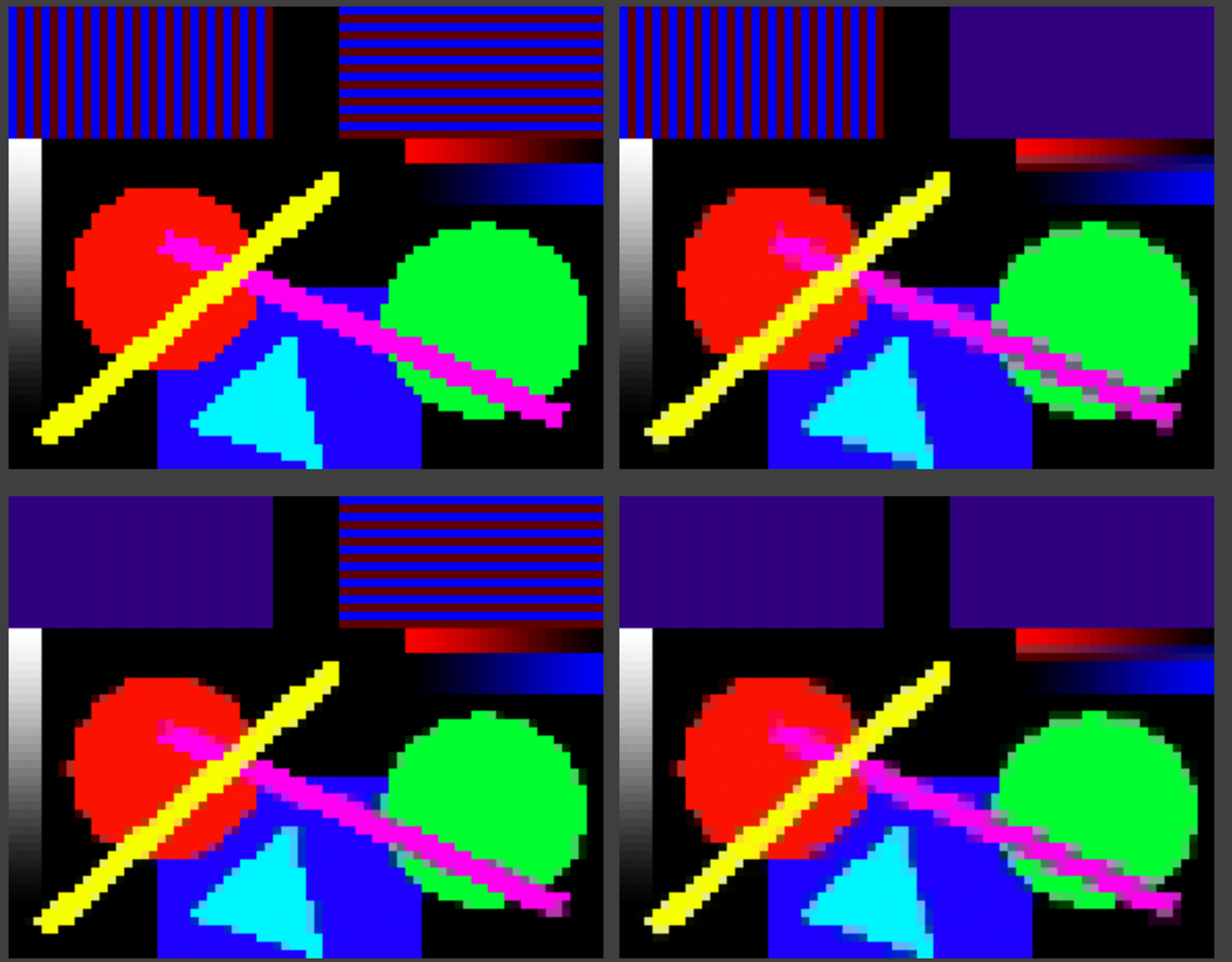
Chroma format	pixels / line Y	lines / frame Y	pixels / line Cb, Cr	lines / frame Cb, Cr	H sub-sampling factor	V sub-sampling factor
4:4:4	720	480	720	480	none	none
4:2:2	720	480	360	480	2:1	none
4:2:0	720	480	360	240	2:1	2:1
4:1:1	720	480	180	480	4:1	none





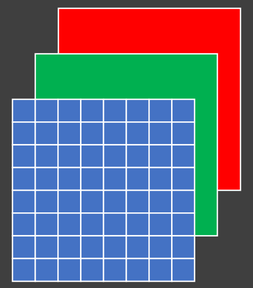
视频格式基础：YUV降采样

1x1 No Subsampling	1x2 Subsampling
2x1 Subsampling	2x2 Subsampling



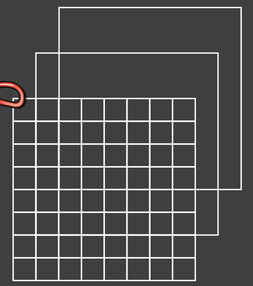
- JPEG压缩色度降采样
  - RGB to YCbCr
  - Cb和Cr分量：在水平/垂直方向上将每一对相邻像素值平均后得到输出值
  - 亮度分量：完全保持一致
- JPEG的色度降采样处理是基于人眼特征设计的。
  特殊场景：医学
- 商业软件的做法可更加灵活：Adobe Photoshop中的“分级导出”：低质量对应2x2 Subsampling、高质量对应1x1 No Subsampling

视频格式基础：JPEG图像压缩流程

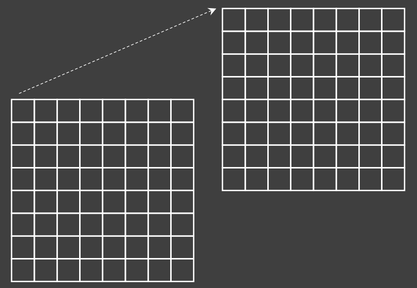


RGB空间

视觉冗余



YCbCr变换

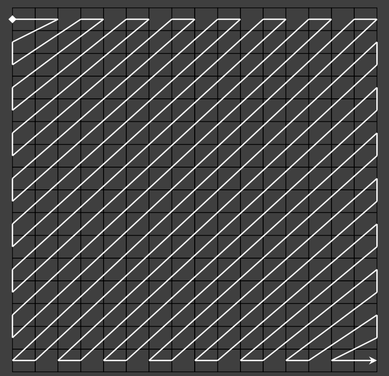


8x8图像分块

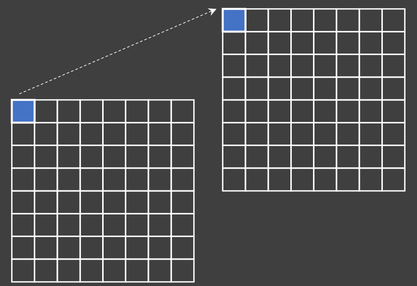
离散余弦变换（DCT）

视觉冗余

量化



Zig-Zag扫描



DC系数和AC系数分离

空间冗余

DC

AC

DPCM编码

游程编码

信息熵冗余

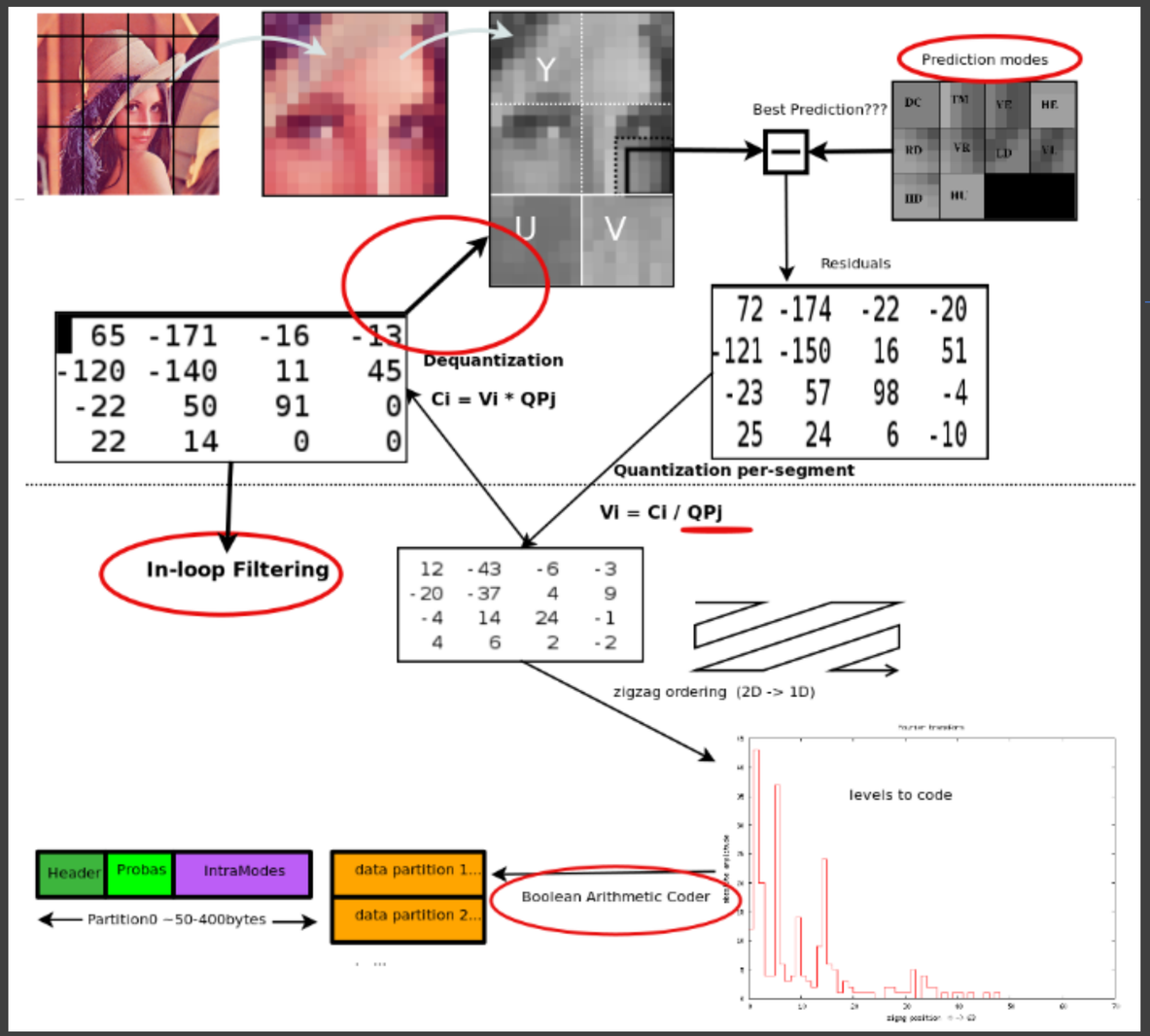
Huffman编码（VLC编码）

帧开始

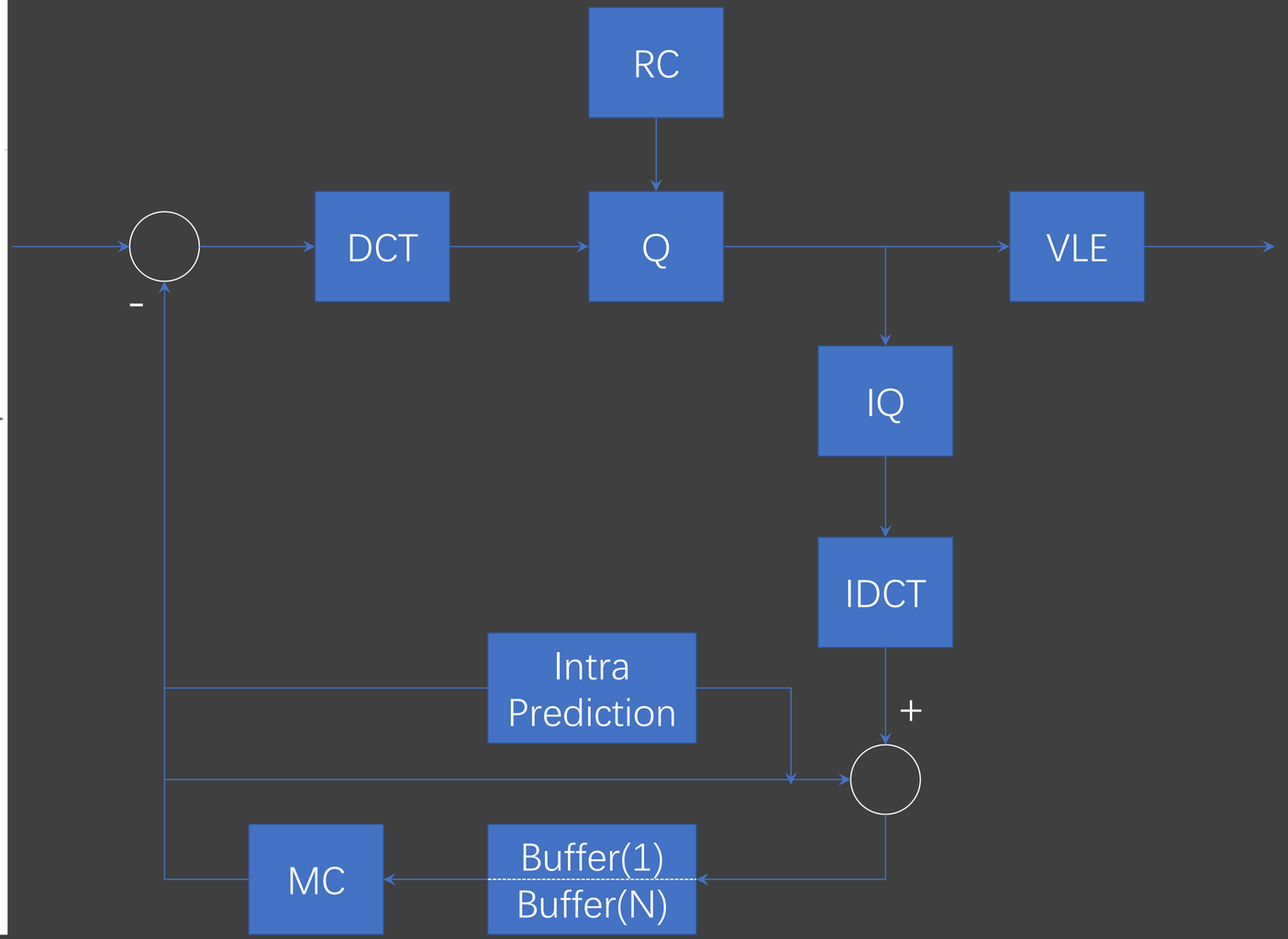
帧数据

帧结尾

视频格式基础：WebP图像压缩流程

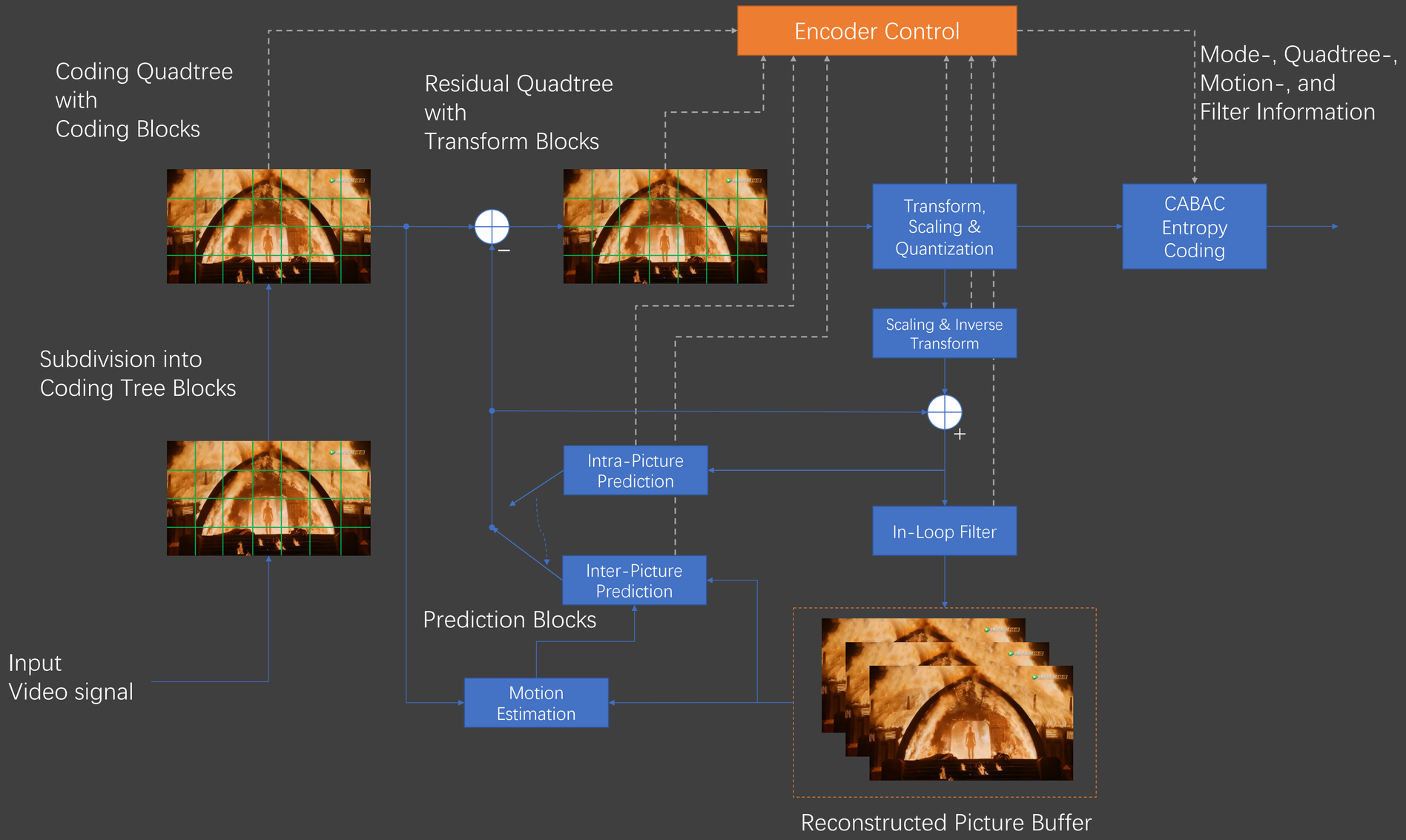


Still Image(e.g. WebP)

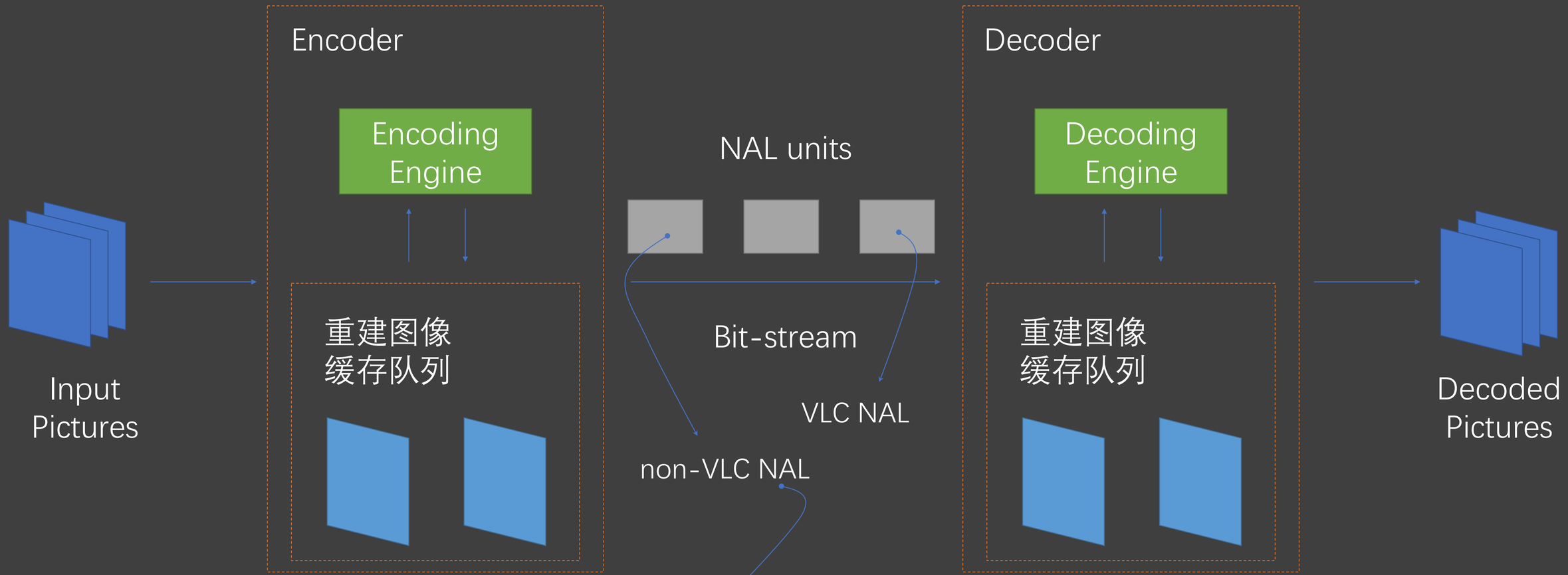


Moving Picture

视频格式基础：混合编码框架

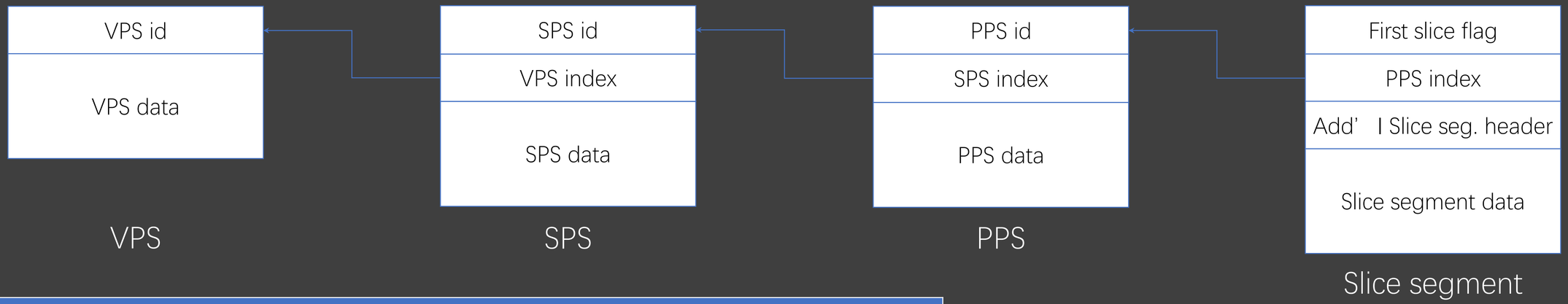


视频格式基础：AVC/HEVC压缩码流结构




```
+---------------+---------------+
|0|1|2|3|4|5|6|7|0|1|2|3|4|5|6|7|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|F|  NALType  |  LayerId  | TID |
+-------------+-----------------+
```

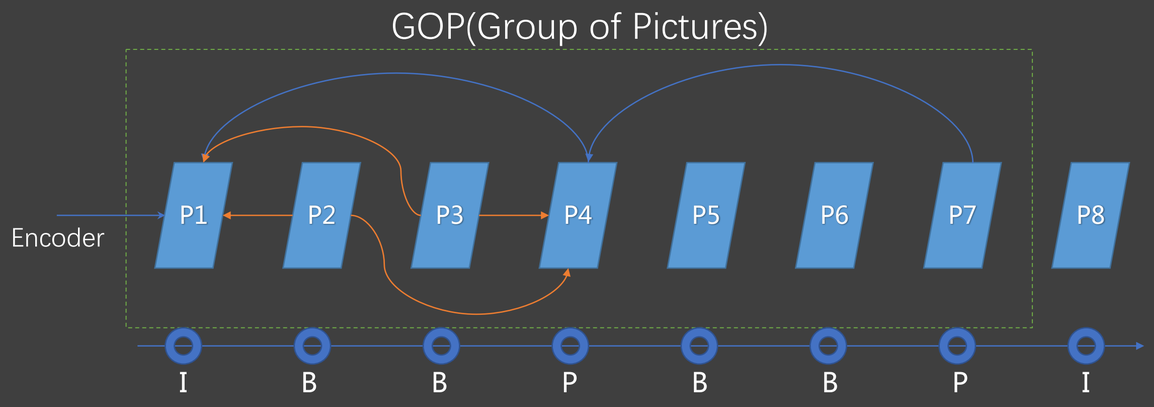
视频格式基础：Non-VCL NALU

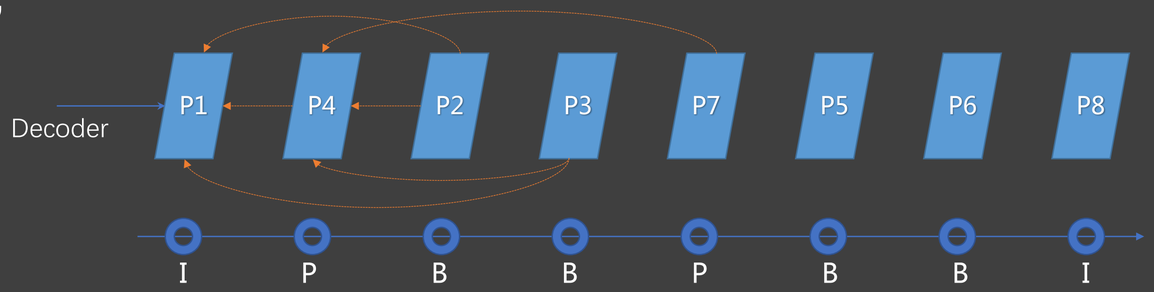


Non-VCL NAL unit types			
Parameter sets	32	VPS_NUT	Video parameter set
	33	SPS_NUT	Sequence parameter set
	34	PPS_NUT	Picture parameter set
Delimiters	35	AUD_NUT	Access unit delimiter
	36	EOS_NUT	End of sequence
	37	EOB_NUT	End of bit-stream
Filler data	38	FD_NUT	Filler data
Supplemental enhancement information (SEI)	39	PREFIX_SEI_NUT	
	40	SUFFIX_SEI_NUT	
Reserved	41-47	RSV	
Unspecified	48-63	UNSPEC	

视频格式基础：GOP (Group Of Pictures)

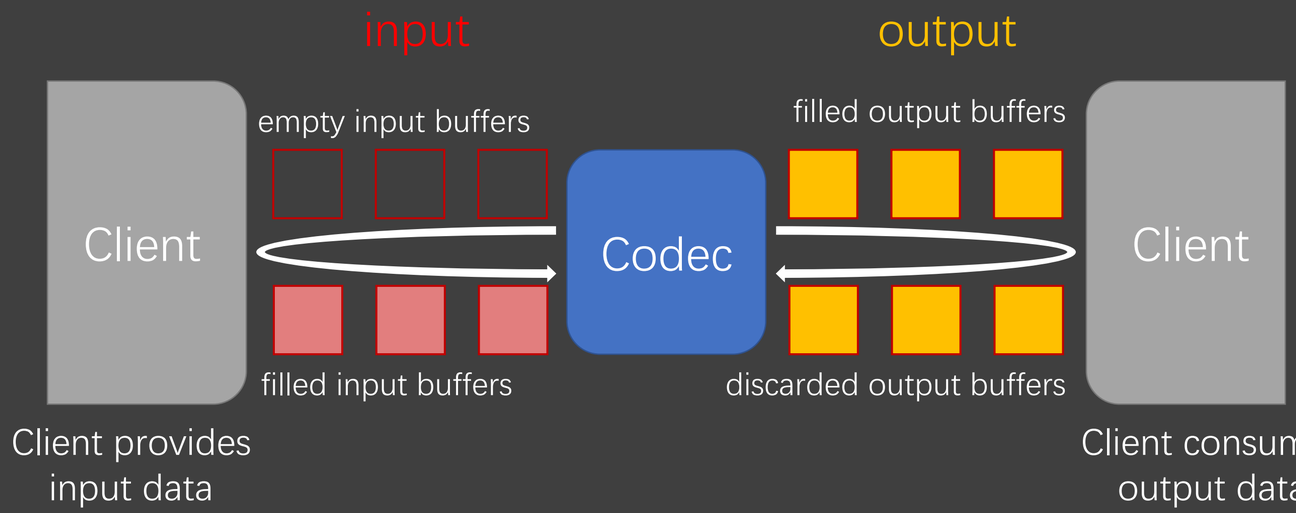
- I帧：Intra Frame/Key Frame
  - 类JPEG压缩流程。压缩效率低，无需参考其他图像，可作为其他图像的参考图像。
- P帧：Prediction Frame
  - 仅使用前序图像作为参考图像。压缩效率提高，但需要一帧参考图像，可作为其他图像的参考图像。
- B帧：Bi-prediction Frame
  - 可使用前序与后序图像作为参考图像。压缩效率高，但需要前后双向参考图像，且会引发解码端的图像重排。不作为参考图像使用。考虑时延问题，一般在实时会话业务中，不使用B帧。
- SP帧：Synchronization-predictive Frame
  - 使用不同的参考图像得到相同的重构图像。用于多码率流间切换、错误恢复等场景。

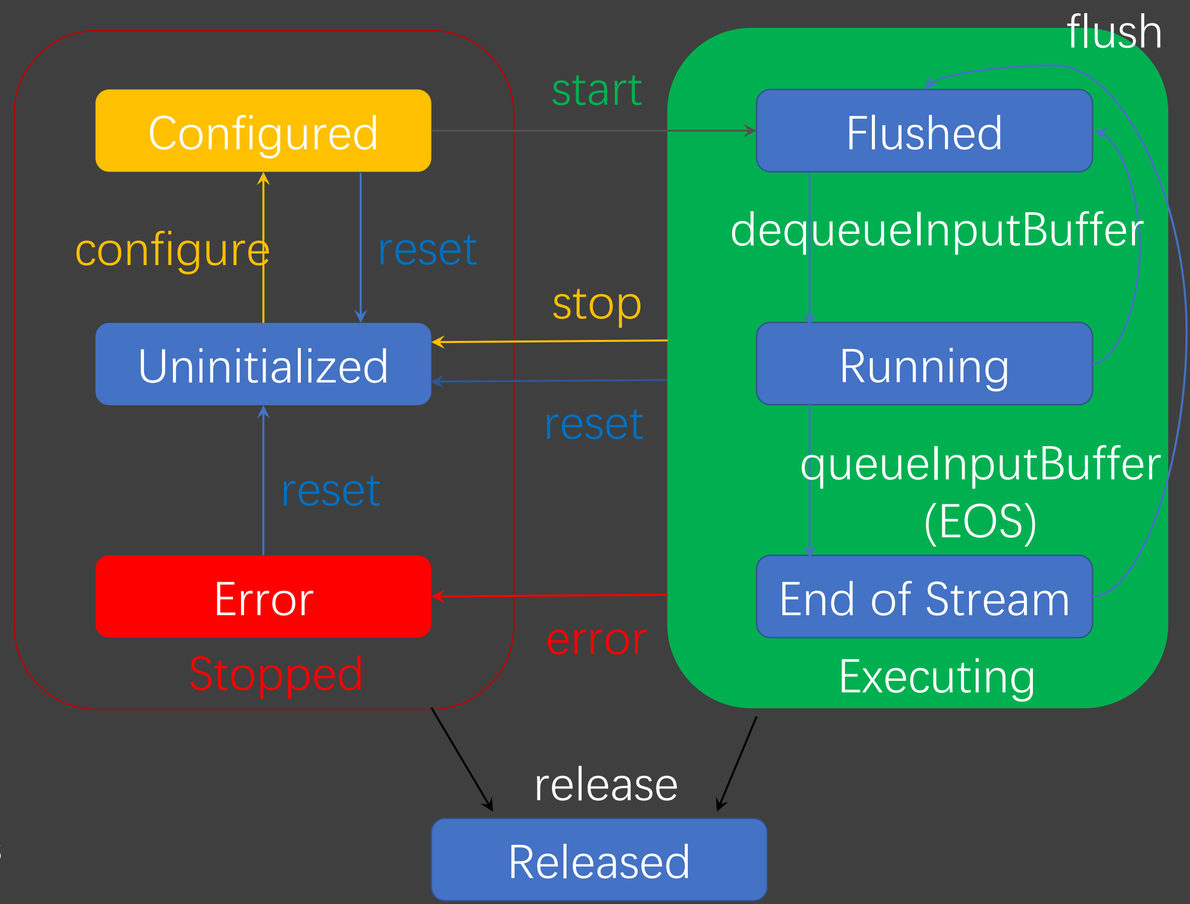




视频格式解码：Android平台方案

- Low-level接口：*vs.* StageFright, *vs.* openMAX
- Encoder与Decoder、音频与视频共用一套API/Workflow
- MediaExtractor+MediaCodec+MediaSync=Player





视频格式解码：Android代码DEMO

- 同步模式≧4.1、异步模式≧5.0
- hardware codec性能没有显著区别
- 实际使用同步模式时，可以用不同线程对应input buffers pipeline和output buffers pipeline

同步模式(synchronous mode)

```
MediaCodec codec = MediaCodec.createByCodecName(name);
codec.configure(format, …);
MediaFormat outputFormat = codec.getOutputFormat(); // option B
codec.start();
for (;;) {
  int inputBufferId = codec.dequeueInputBuffer(timeoutUs);
  if (inputBufferId >= 0) {
    ByteBuffer inputBuffer = codec.getInputBuffer(…);
    // fill inputBuffer with valid data
    …
    codec.queueInputBuffer(inputBufferId, …);
  }
  int outputBufferId = codec.dequeueOutputBuffer(…);
  if (outputBufferId >= 0) {
    ByteBuffer outputBuffer = codec.getOutputBuffer(outputBufferId);
    MediaFormat bufferFormat = codec.getOutputFormat(outputBufferId); // option
A
    // bufferFormat is identical to outputFormat
    // outputBuffer is ready to be processed or rendered.
    …
    codec.releaseOutputBuffer(outputBufferId, …);
  } else if (outputBufferId == MediaCodec.INFO_OUTPUT_FORMAT_CHANGED) {
    // Subsequent data will conform to new format.
    // Can ignore if using getOutputFormat(outputBufferId)
    outputFormat = codec.getOutputFormat(); // option B
  }
}
codec.stop();
codec.release();
```

异步模式(asynchronous mode)

```
MediaCodec codec = MediaCodec.createByCodecName(name);
MediaFormat mOutputFormat; // member variable
codec.setCallback(new MediaCodec.Callback() {
  @Override
  void onInputBufferAvailable(MediaCodec mc, int inputBufferId) {
    ByteBuffer inputBuffer = codec.getInputBuffer(inputBufferId);
    // fill inputBuffer with valid data
    …
    codec.queueInputBuffer(inputBufferId, …);
  }

  @Override
  void onOutputBufferAvailable(MediaCodec mc, int outputBufferId, …) {
    ByteBuffer outputBuffer = codec.getOutputBuffer(outputBufferId);
    MediaFormat bufferFormat = codec.getOutputFormat(outputBufferId); // option A
    // bufferFormat is equivalent to mOutputFormat
    // outputBuffer is ready to be processed or rendered.
    …
    codec.releaseOutputBuffer(outputBufferId, …);
  }

  @Override
  void onOutputFormatChanged(MediaCodec mc, MediaFormat format) {
    // Subsequent data will conform to new format.
    // Can ignore if using getOutputFormat(outputBufferId)
    mOutputFormat = format; // option B
  }

  @Override
  void onError(…) {
    …
  }
});
codec.configure(format, …);
mOutputFormat = codec.getOutputFormat(); // option B
codec.start();
// wait for processing to complete
codec.stop();
codec.release();
```

视频格式解码：Android踩坑点

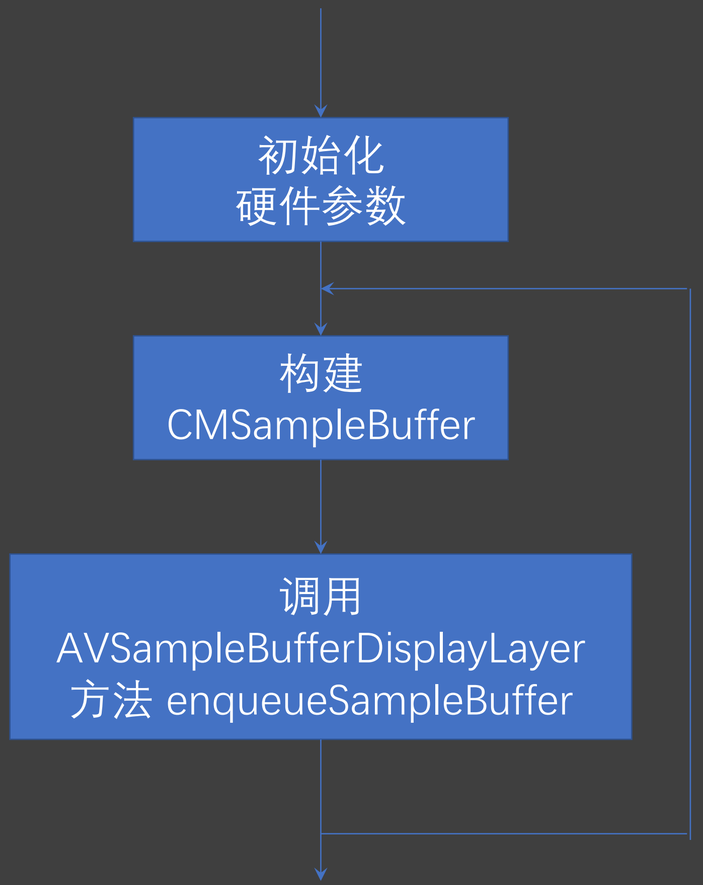
```
MediaCodec codec = MediaCodec.createByCodecName(name);
codec.configure(format, …);
MediaFormat outputFormat = codec.getOutputFormat(); // option B
codec.start();
for (;;) {
  int inputBufferId = codec.dequeueInputBuffer(timeoutUs);
  if (inputBufferId >= 0) {
    ByteBuffer inputBuffer = codec.getInputBuffer(…);
    // fill inputBuffer with valid data
    …
    codec.queueInputBuffer(inputBufferId, …);
  }
  int outputBufferId = codec.dequeueOutputBuffer(…);
  if (outputBufferId >= 0) {
    ByteBuffer outputBuffer = codec.getOutputBuffer(outputBufferId);
    MediaFormat bufferFormat = codec.getOutputFormat(outputBufferId); // option A
    // bufferFormat is identical to outputFormat
    // outputBuffer is ready to be processed or rendered.
    …
    codec.releaseOutputBuffer(outputBufferId, …);
  } else if (outputBufferId == MediaCodec.INFO_OUTPUT_FORMAT_CHANGED) {
    // Subsequent data will conform to new format.
    // Can ignore if using getOutputFormat(outputBufferId)
    outputFormat = codec.getOutputFormat(); // option B
  }
}
codec.stop();
codec.release();
```

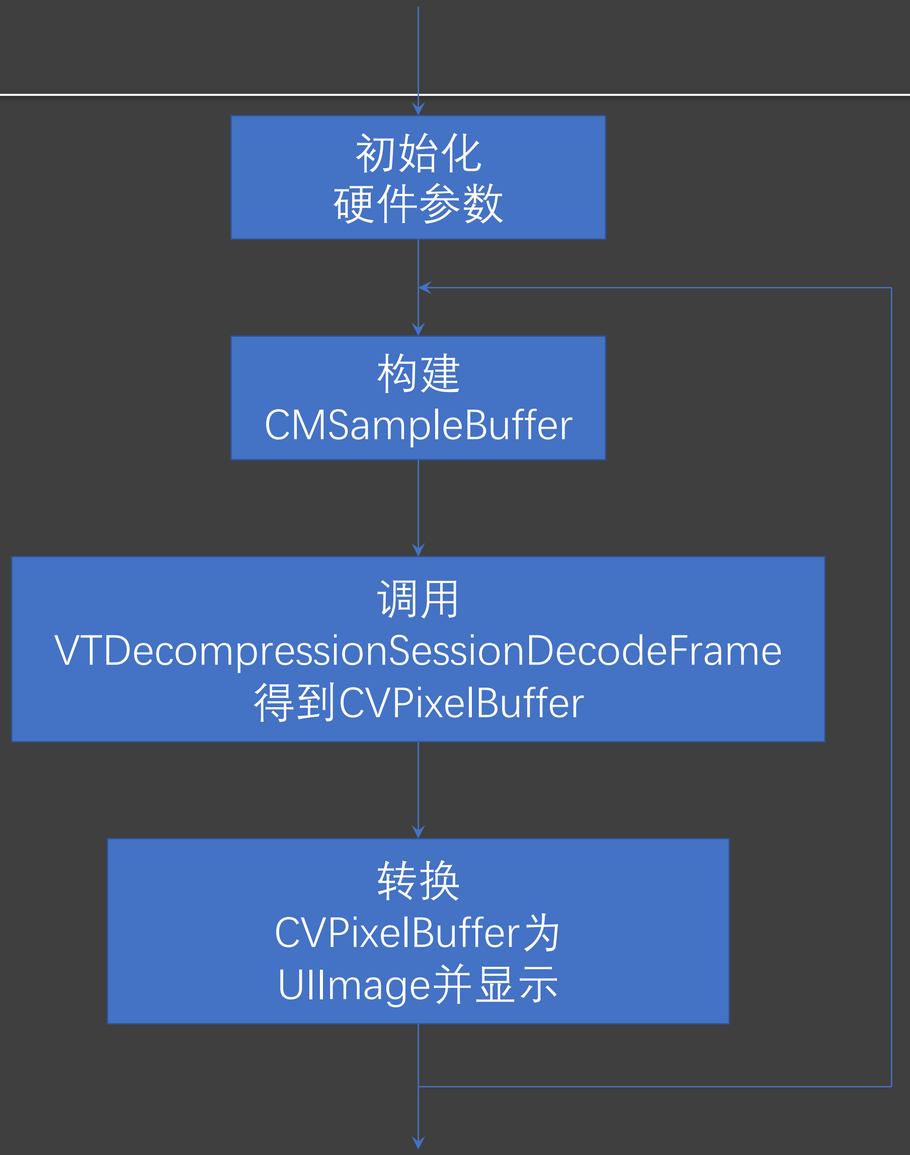
- View的类型：SurfaceView, TextureView, NULL, etc.
- Color Space：老虎的屁股摸不得
- Codec-specific Data：VPS/SPS/PPS for H.264/H.265,
  ‘esds’ box for AAC

- fill data：注意数据帧格式，MP4 *vs.* MPEG TS
- 部分芯片注意区分 CONFIG_DATA和AV stream data
- 当参数发生变化时，应该重新MediaCodec::configure，不要过度信任底层容错机制

- 保险起见，以此为准来创建Render及View
- 底层codec可能没有对input PTS作re-ordering

视频格式解码：iOS平台方案





- 硬解+渲染，内聚绑定
- 没有额外的buffer操作，理论上效率最高

- 硬解与渲染分离，方便 post-processing
- buffer操作的性能损耗、color space自行接管

iOS代码DEMO

```
CMVideoFormatDescriptionRef __fmtDesp;
CMVideoFormatDescriptionCreateFromH264ParameterSets(kCFAllocatorDefault,
                                    2,
                                    paramSetPointers,   // <-- addr: sps+pps
                                    paramSetSizes,      // <-- size: sps+pps
                                    4,
                                    __fmtDesp);
/* ******** for HEVC ********
CMVideoFormatDescriptionCreateFromHEVCParameterSets(kCFAllocatorDefault,
                                    3,
                                    paramSetPointers,   // <-- addr: vps+sps+pps
                                    paramSetSizes,      // <-- size: vps+sps+pps
                                    4,
                                    nil,
                                    __fmtDesp);
*/

VTDecompressionOutputCallbackRecord __decoderCB;
//__decoderCB.decompressionOutputCallback = ..;
//typedef void (*VTDecompressionOutputCallback)(void * CM_NULLABLE decompressionOutputRefCon,
//                                    void * CM_NULLABLE sourceFrameRefCon,
//                                    OSStatus status,
//                                    VTDecodeInfoFlags infoFlags,
//                                    CM_NULLABLE CVImageBufferRef imageBuffer,
//                                    CMTime presentationTimeStamp,
//                                    CMTime presentationDuration);

CFMutableDictionaryRef __buffAttr = CFDictionaryCreateMutable(..);
// kCVPixelBufferPixelFormatTypeKey

VTDecompressionSessionRef __decSess;
VTDecompressionSessionCreate(kCFAllocatorDefault, __fmtDesp, NULL, __buffAttr, &__decoderCB,
&__decSess);
```

```
for (;;) {
    CMBlockBufferRef videoBlock;
    CMBlockBufferCreateWithMemoryBlock(kCFAllocatorDefault,
                                       addr_inputVideoFrame,    // in NAL capsulate format
                                       size_inputVideoFrame,
                                       kCFAllocatorNull,
                                       NULL,
                                       0,
                                       size_inputVideoFrame,
                                       0,
                                       &videoBlock);

    CMSampleTimingInfo timeInfo;
    CMSampleTimingInfo timeInfoArray[1];
    timeInfo.presentationTimeStamp = CMTimeMake(pts, 1);
    timeInfo.decodeTimeStamp = CMTimeMake(dts, 1);
    timeInfoArray[0] = timeInfo;

    CMSampleBufferRef sampleBuffer = nil;
    CMSampleBufferCreate(kCFAllocatorDefault, videoBlock, true, NULL, NULL, __fmtDesp, 1, 1, timeInfoArray, 0, NULL,
&sampleBuffer);

    VTDecompressionSessionDecodeFrame(__decSess,
                                      sampleBuffer,
                                      kVTDecodeFrame_EnableAsynchronousDecompression,
                                      NULL,
                                      &flagOut);
    VTDecompressionSessionWaitForAsynchronousFrames(__decSess);

    // ...
    // ... fetch the decompressed picture data in asynchronous way
    CVPixelBufferRef image_buffer; // return by the callback from the decoder
    CVPixelBufferLockBaseAddress(image_buffer, kCVPixelBufferLock_ReadOnly);
    void *yData = CVPixelBufferGetBaseAddressOfPlane(image_buffer, 0);
    void *uvData = CVPixelBufferGetBaseAddressOfPlane(image_buffer, 1);
    int yStride = (int)CVPixelBufferGetBytesPerRowOfPlane(image_buffer, 0);
    int uvStride = (int)CVPixelBufferGetBytesPerRowOfPlane(image_buffer, 1);
    CVPixelBufferUnlockBaseAddress(image_buffer, kCVPixelBufferLock_ReadOnly);
    CVPixelBufferRelease(image_buffer);

    // Render YUV data, in most cases, with NV12 plane format
    // ...
}

VTDecompressionSessionFinishDelayedFrames(__decSess);
VTDecompressionSessionInvalidate(__decSess);
CFRelease(__decSess); CFRelease(__fmtDesp); CFRelease(__buffAttr);
```

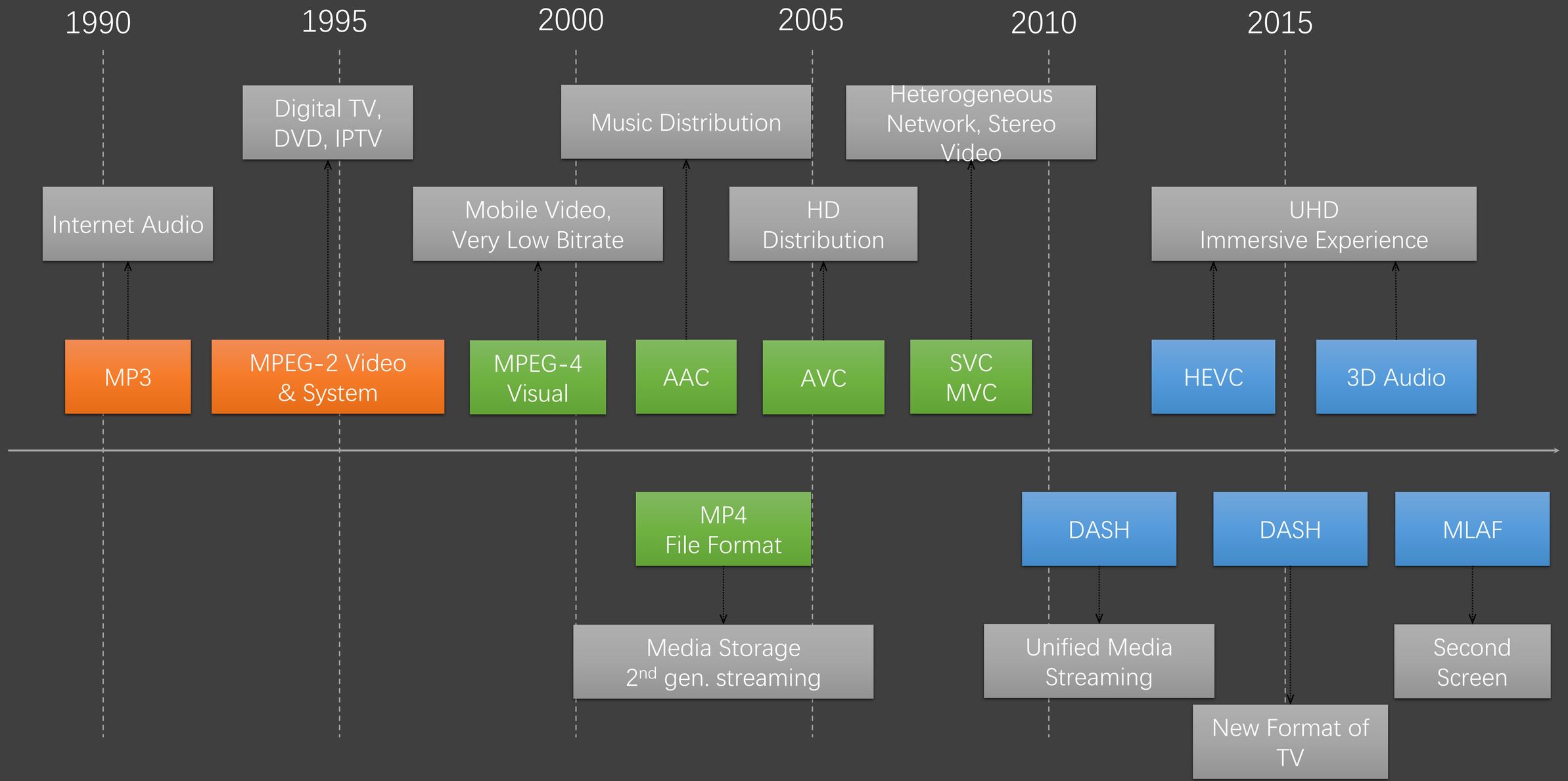
视频编码：关键技术的演进发展

算法工具	MPEG-2	H.264/AVC	HEVC
块大小	16x16 (Macroblock)	16x16 (Macroblock)	16x16 to 64x64 (Coding Tree Unit)
划分模式	Intra 8x8, Inter 16x16	子块最小到4x4	Intra：最小到4x4（对称） Inter：单向4x8或者8x4；双向尺寸更大（对称/非对称）
变换	Floating DCT	4x4/8x8 Integer DCT	32x32到4x4大小的DCT、亮度Intra 4x4块DST
帧内预测	DC系数预测	9类模式	35类模式
运动预测	单邻居	空域中值（3 blocks）	空时域结合的高级运动向量预测
Motion-copy	/	Direct Mode	Merge Mode
运动精度	½ 精度双线性	½ 精度6-tap滤波；1/4精度双线性	亮度分量1/4精度7-或8-tap滤波；色度分量1/8精度4-tap滤波
熵编码	VLC	CABAC或者CAVLC	并行化CABAC
滤波	/	De-blocking Filter	De-blocking Filter & Sample Adaptive Offset
并行编码	Slices	Slices	WPP, tiles, slices

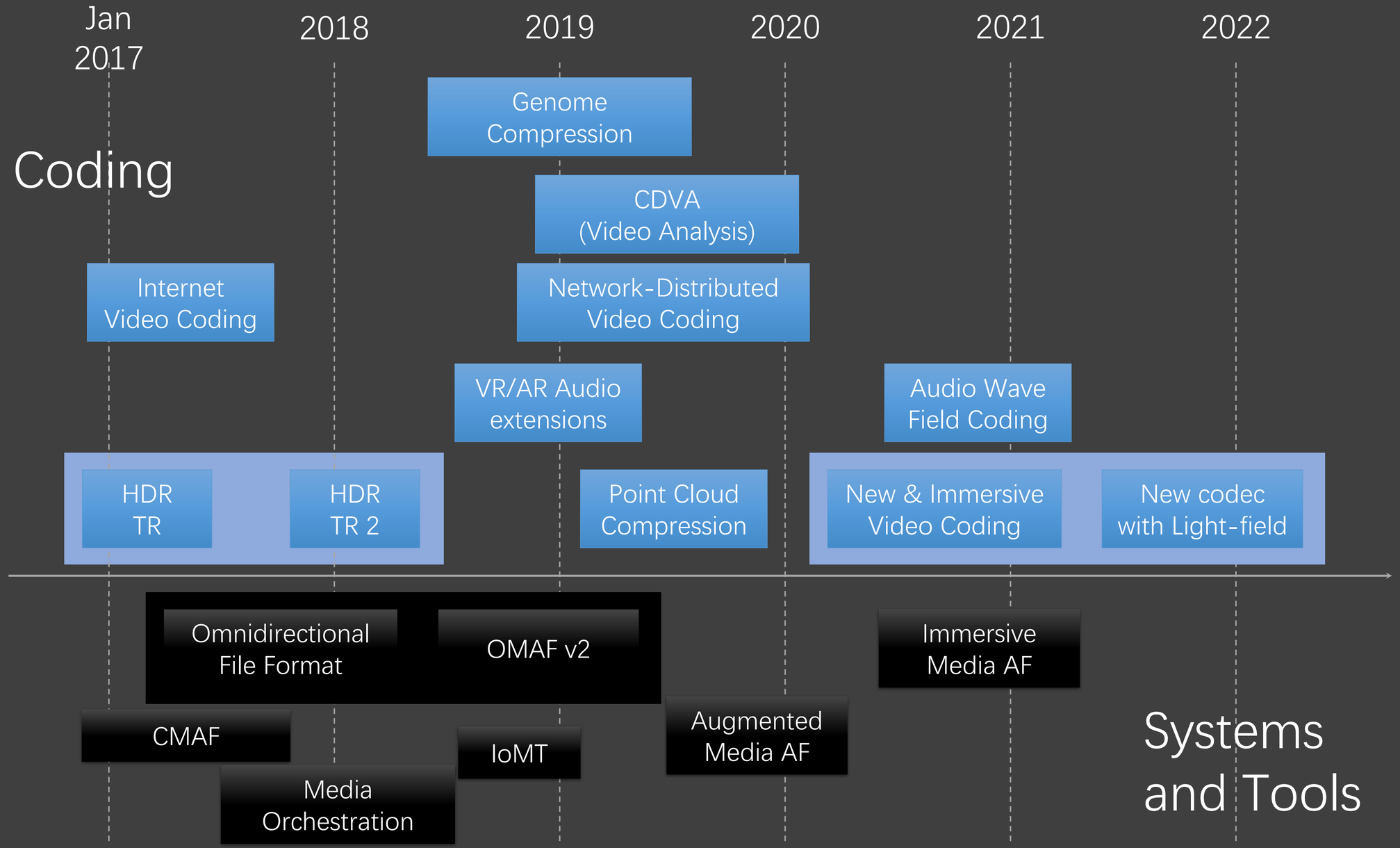
视频编码：国内外标准化发展之路

ISO/IEC JTC1/SG29/WG11	MPEG-1	VCD
	MPEG-2	• DVD、数字电视 • D1(720x480, 720x576), 2MB~20MB • 支持Progressive/Interlaced、兼容电视格式
	MPEG-4	• 甚低比特率(128kbps) • 基于面向对象的编码框架(10+年的神话、眼下…) • DCT变换系数预测(帧内预测的鼻祖)、1/2像素精度预测、高级运动向量预测、多模式预测、Post Processing…
	MPEG-7/21	• 多媒体系统接口框架 • “走两步、走两步…”
ITU-T	H.261	
	H.263, H.263+, H.263++	加强版MPEG-4、傻傻分不清楚…
JVT	MPEG4 part10 AVC/H.264	编码性能相比MPEG2提高1倍
	HEVC/H.265	编码性能相比H.264/AVC提高1倍
On2	VP6/VP7（商业） VP8(卖身后开源)	• FLV上的传奇 • 人有多大胆、地有多大产
Google	VP9, AV1	• 打倒MPEG：直接对标HEVC • 没有版权？？
Microsoft	WMV8, WMV9, VC1	
天朝	AVS、AVS2	打倒万恶的资本主义、突破xxx的技术封锁、填补xxx的技术空白，为我国xxx的发展奠定了xxx的作用

视频编码：路线图，by now

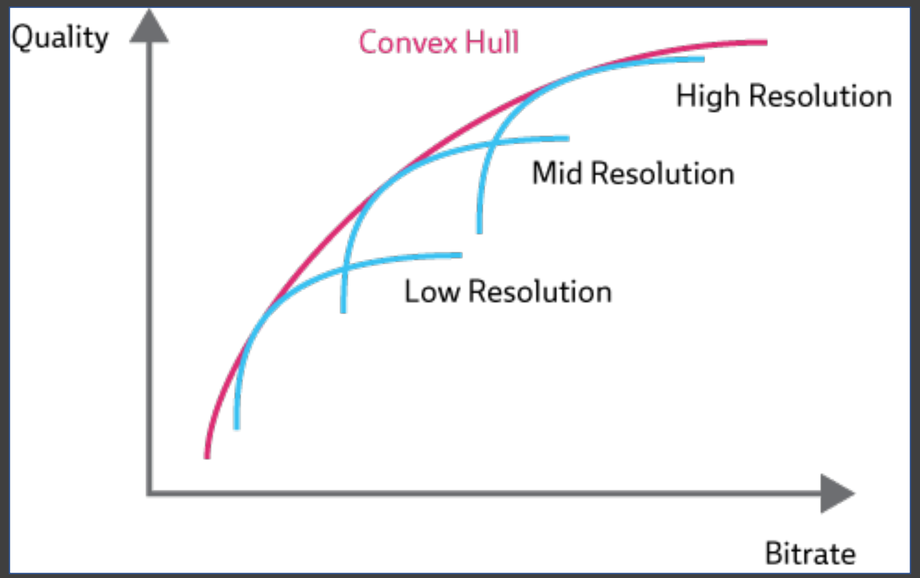


视频编码：路线图，after now…



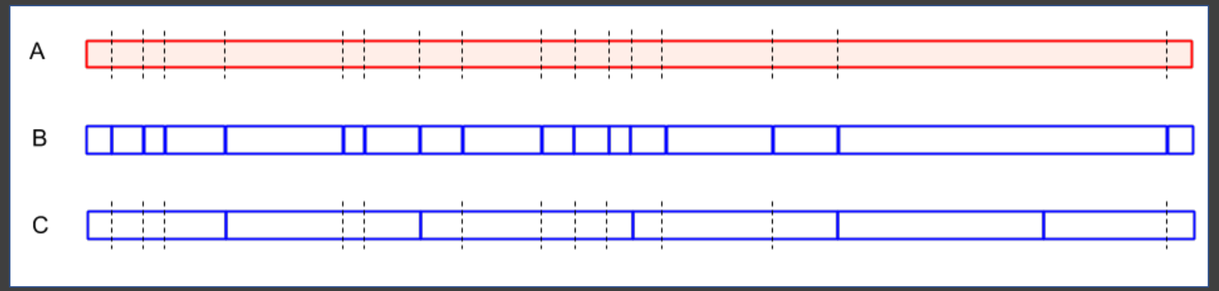
播放器HOWTO：极致提高视频压缩率

- 利用算力暴力试探性编码
- 颗粒度：全段视频



Per-title Encoding

- 利用算力暴力试探性编码
- 颗粒度：Chunk / Shot

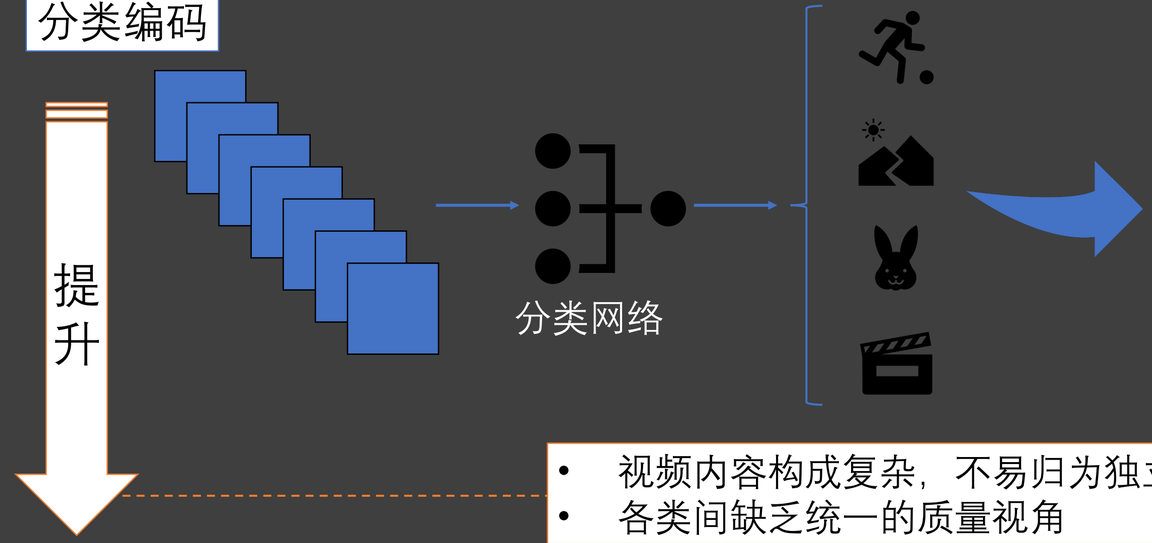


Collation of shots into ‘chunks’. **(A) Representation of a video timeline**. The dashed vertical black lines represent shot boundaries. **(B) One shot in one chunk**: Each shot is assigned a chunk. **(C) Collate shots into a chunk:** Accumulate integer number of shots within a target chunk duration.

——来自Netflix的暴力美学

播放器HOWTO：CAE(Content Aware Encoding)

分类编码

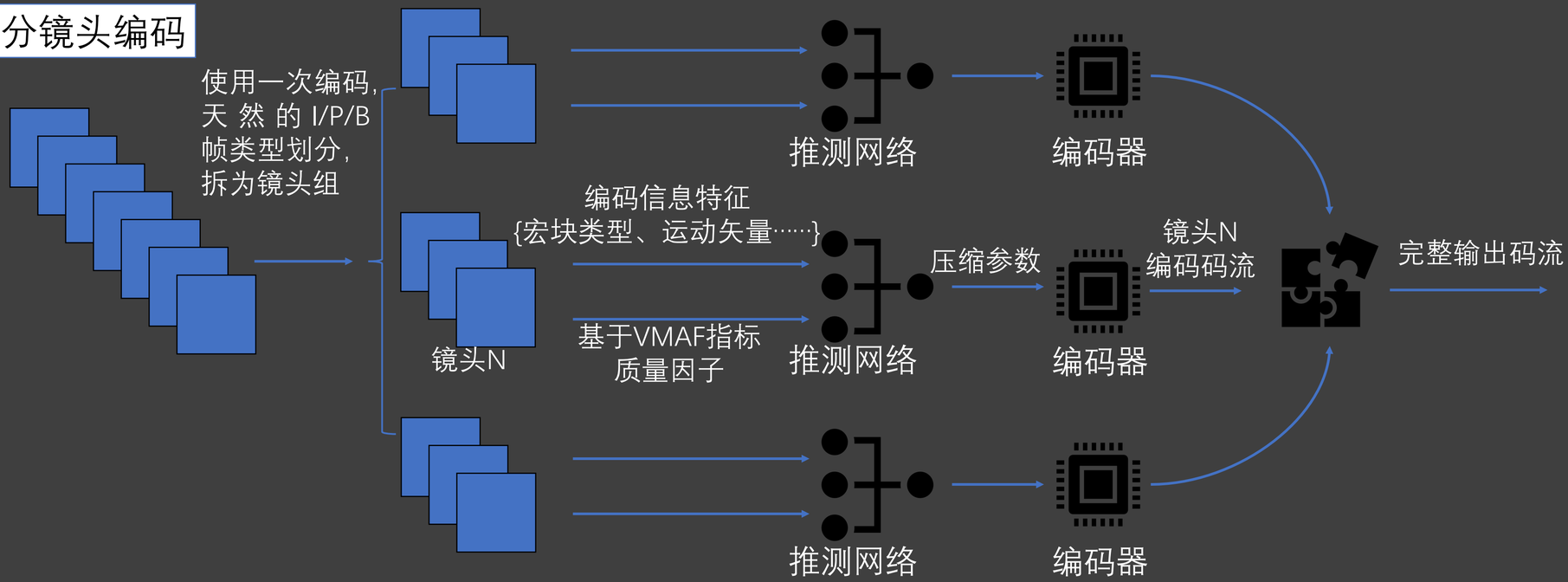


	2D动画	影视	体育
码率设定	↓	→	↑
运动检测范围	小	普通	很大
区块划分	较大	中等	较小
参考图像范围	很少	中等	多

按照类型预设编码配置

- 视频内容构成复杂，不易归为独立某一类
- 各类间缺乏统一的质量视角

分镜头编码



播放器HOWTO：视频压缩算法选型

业务场景

- 主流分辨率/码率/帧率
- 片源特征：影视剧、UGC
- 核心用户体验指标：延时、缓冲、卡顿
- 头部功能：无缝切换、多字幕/音轨、ABR

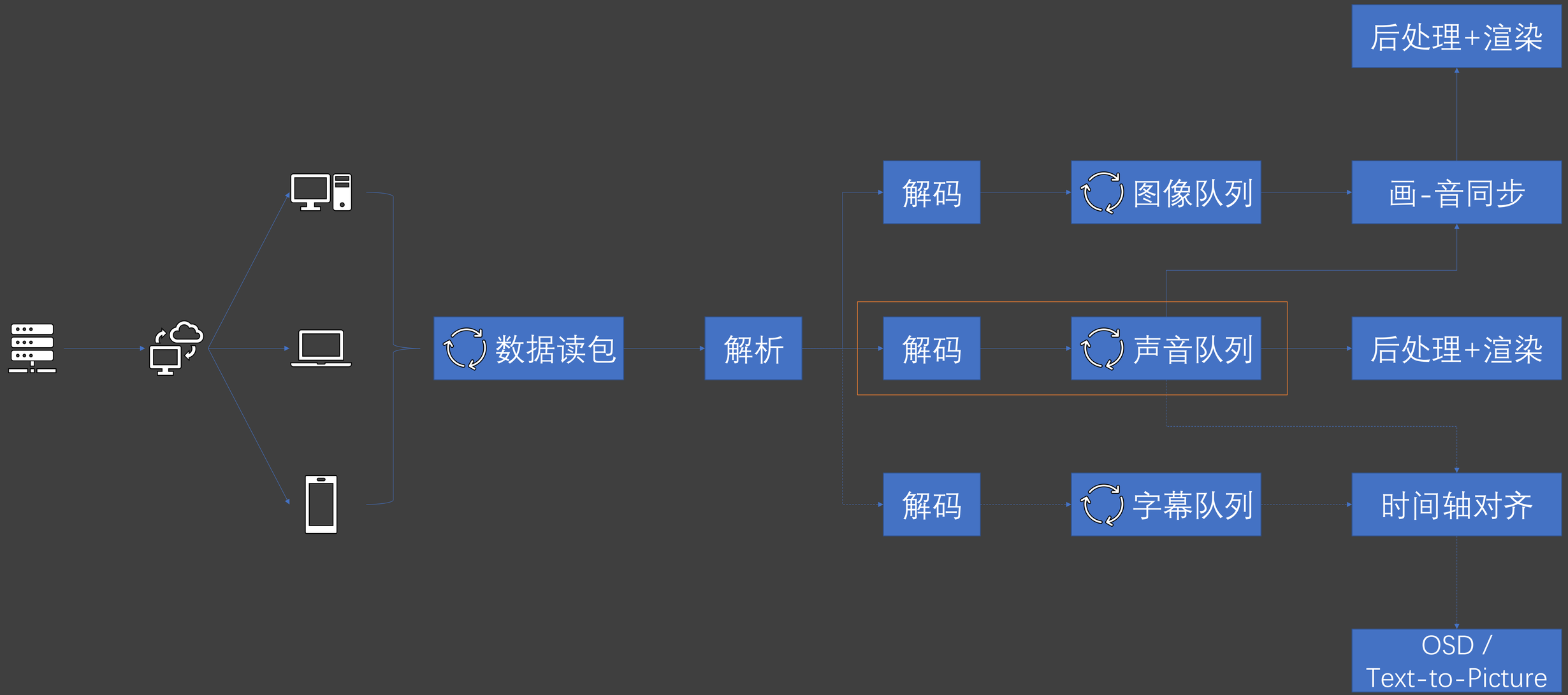
商业模式

- 地域：出海？
- 会员权益：4K、HDR、DRM
- 运营需求：直播转点播
- 成本控制：带宽、存储

技术生态

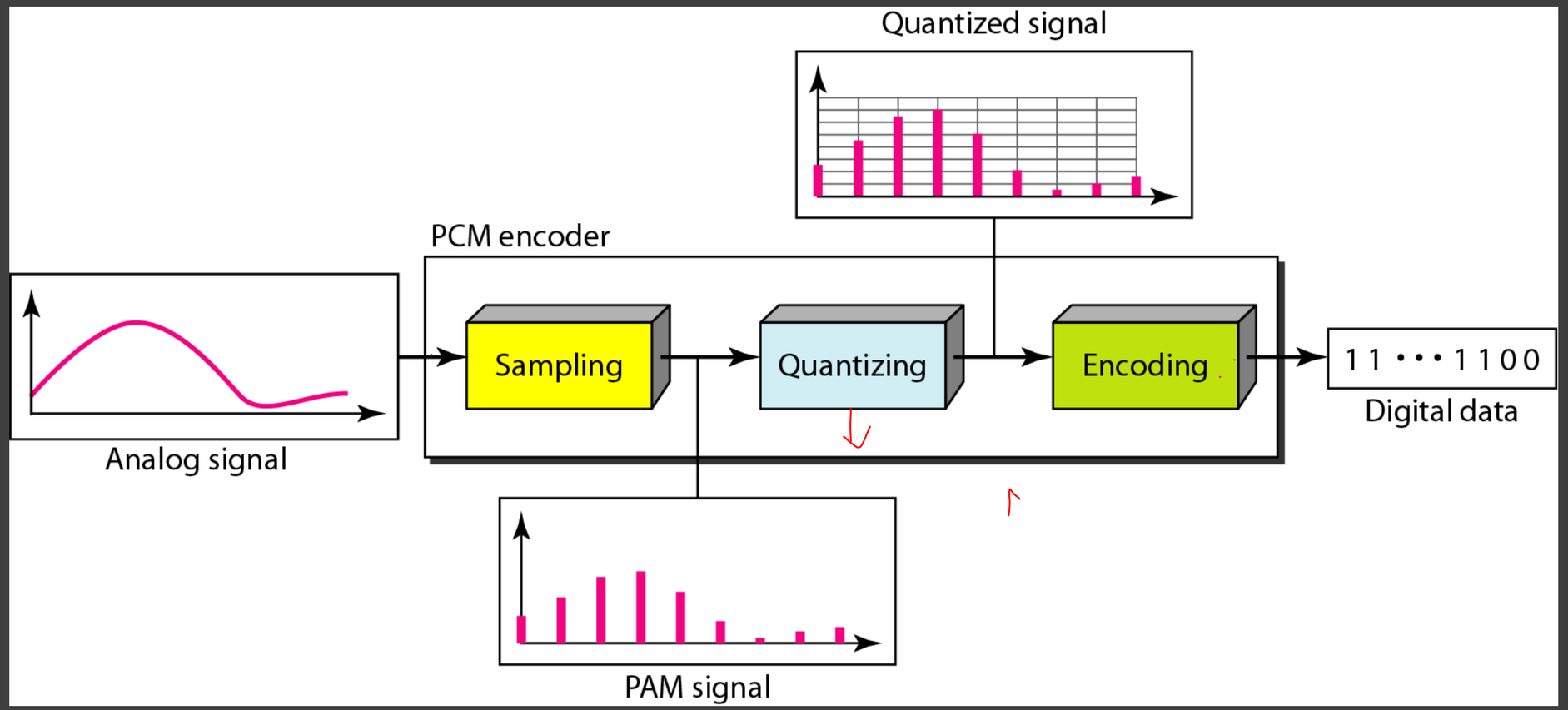
- 硬件：芯片+屏幕
- 软件：FFMPEG、商业Codec
- 协议：容器格式、流媒体协议
- 平台：Android/iOS/OTT/PC/Web

播放器流程框图


后处理+渲染
解码
图像队列
画-音同步
数据读包
解析
解码
声音队列
后处理+渲染
解码
字幕队列
时间轴对齐
OSD /
Text-to-Picture

音频格式基础：PCM编码流程

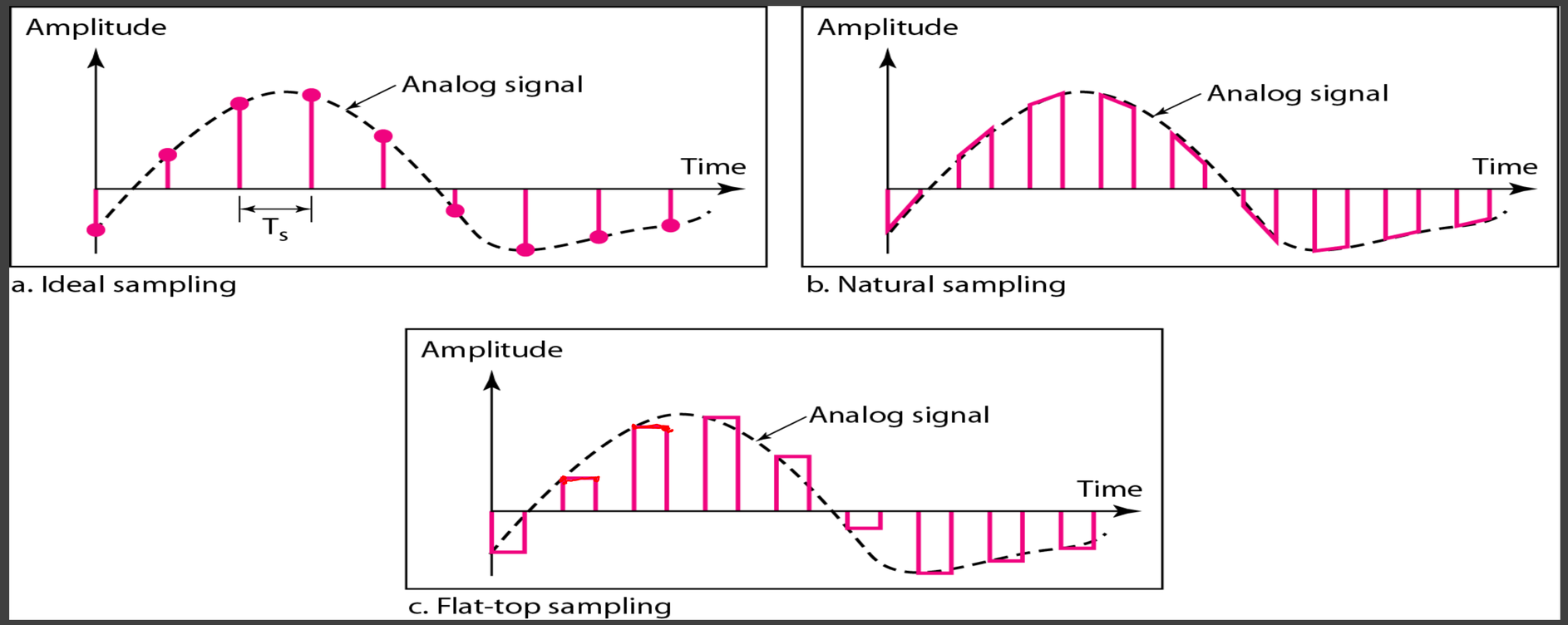
The most common technique to change an analog signal to digital data (digitization) is called pulse code modulation (PCM). A PCM encoder has three processes.



The analog signal is sampled every Ts, where Ts is the sample interval or period. The inverse of the sampling interval is called the sampling rate or sampling frequency and denoted by fs, where fs = 1/Ts.

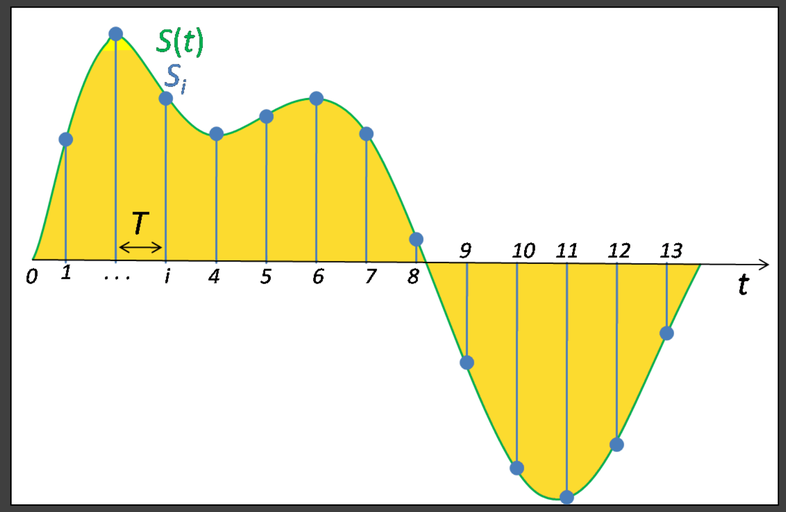
音频格式基础：PCM中的采样与量化

•There are three sampling methods: ideal, natural, and flat-top.
•In ideal sampling, pulses from the analog signal are sampled. This is an ideal sampling method and cannot be easily implemented.
• In natural sampling, a high-speed switch is turned on for only the small period of time when the sampling occurs. The result is a sequence of samples that retains the shape of the analog signal.
•The most common sampling method, called sample and hold, however, creates flat-top samples by using a circuit.



音频格式基础：理解Sampling Rate

- 采样（sampling），将时间连续信号转换为时间离散信号
- 采样率（sampling rate），单位时间内的平均采样点数量
  - Nyquist-Shannon sampling theorem
  - Whittaker-Shannon interpolation formula



- Why 44.1 kHz?

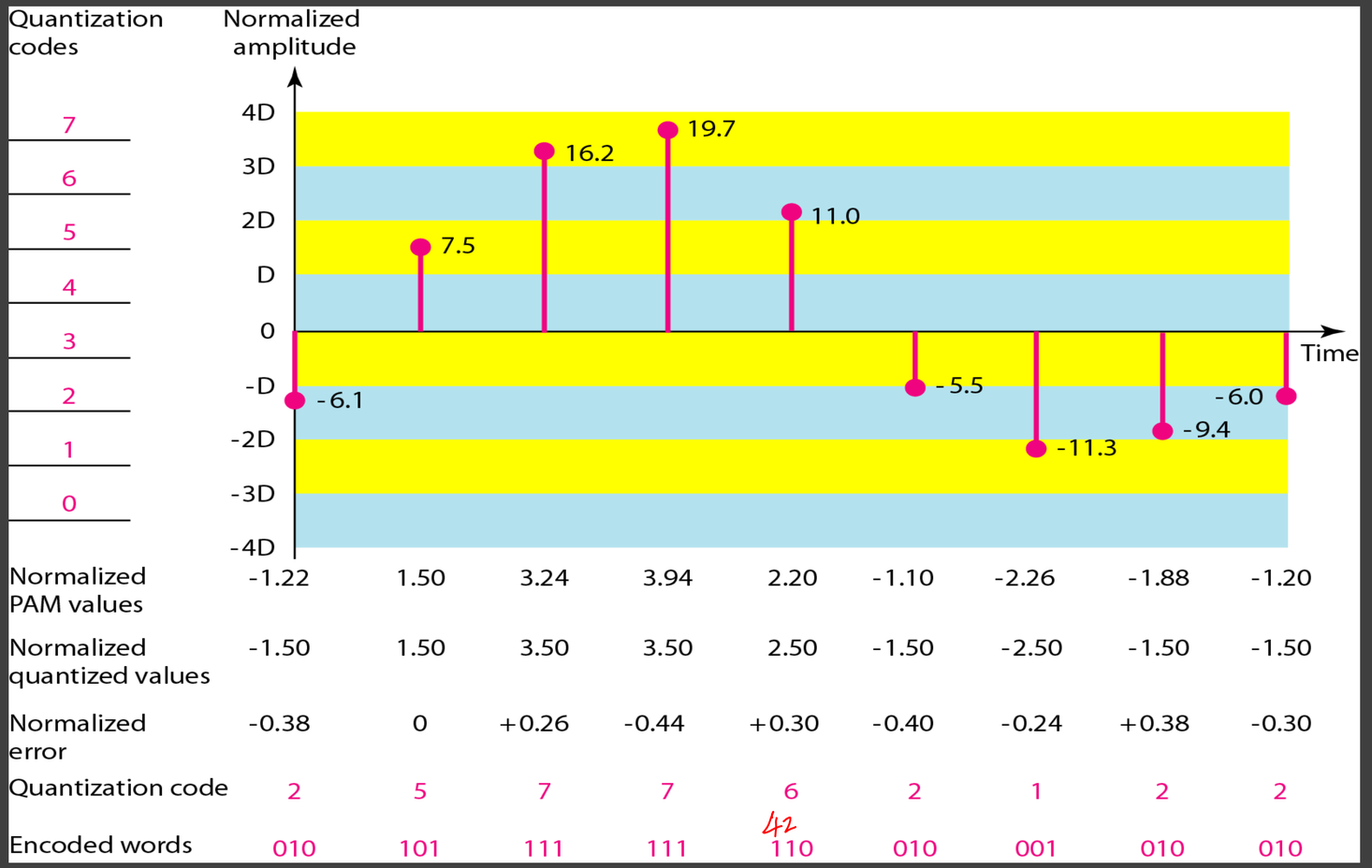
The Nyquist–Shannon sampling theorem says the sampling frequency must be greater than twice the maximum frequency one wishes to reproduce. *Since human hearing range is roughly 20 Hz to 20,000 Hz, the sampling rate had to be greater than 40 kHz.*

In addition, signals must be low-pass filtered before sampling to avoid aliasing. While an ideal low-pass filter would perfectly pass frequencies below 20 kHz (without attenuating them) and perfectly cut off frequencies above 20 kHz, such an ideal filter is theoretically impossible (it is noncausal), so in practice a transition band is necessary, where frequencies are partly attenuated. The wider this transition band is, the easier and more economical it is to make an anti-aliasing filter. The 44.1 kHz sampling frequency allows for a 2.05 kHz transition band.

音频格式基础： 理解Quantization

Quantization and encoding of a sampled signal

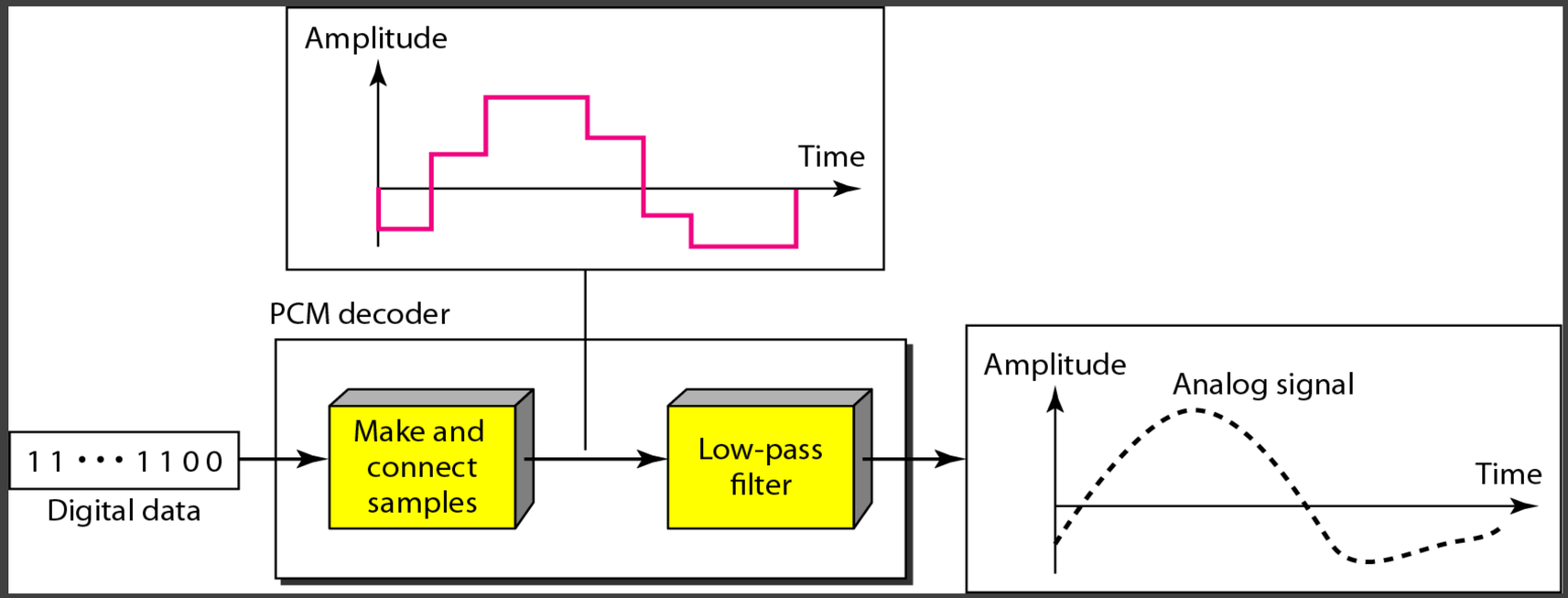
$$\Delta = \frac{V_{max} - V_{min}}{L}$$



Normalized PAM values	-1.22	1.50	3.24	3.94	2.20	-1.10	-2.26	-1.88	-1.20
Normalized quantized values	-1.50	1.50	3.50	3.50	2.50	-1.50	-2.50	-1.50	-1.50
Normalized error	-0.38	0	+0.26	-0.44	+0.30	-0.40	-0.24	+0.38	-0.30
Quantization code	2	5	7	7	6	2	1	2	2
Encoded words	010	101	111	111	110	010	001	010	010

音频格式基础： PCM的信号解码与重建

The recovery of the original signal requires the PCM decoder. The decoder first uses circuitry to convert the code words into a pulse that holds the amplitude until the next pulse. After the staircase signal is completed, it is passed through a low-pass filter to smooth the staircase signal into an analog signal. The filter has the same cutoff frequency as the original signal at the sender. If the signal has been sampled at (or greater than) the Nyquist sampling rate and if there are enough quantization levels, the original signal will be recreated. Note that the maximum and minimum values of the original signal can be achieved by using amplification.



音频格式基础： MPEG压缩的关键原理

- 频域分析
- 划分子带
- 对接心理声学模型

- 根据目标码率和剩余码字控制量化颗粒度
- 量化误差：引入信号失真
- 人耳对误差 “不可察觉”

PCM audio input

Frequency domain mapping

Bit/noise allocation Quantization & coding

Output stream formatting

Compressed output

Signal to mask ratio

Psychoacoustic model

Auxiliary data (optional)

- 描述声音的掩蔽特征：频域掩蔽、时域掩蔽
- 对各子带分别描述
- 人耳特征、大量试验得到

Compressed input

Unpacking

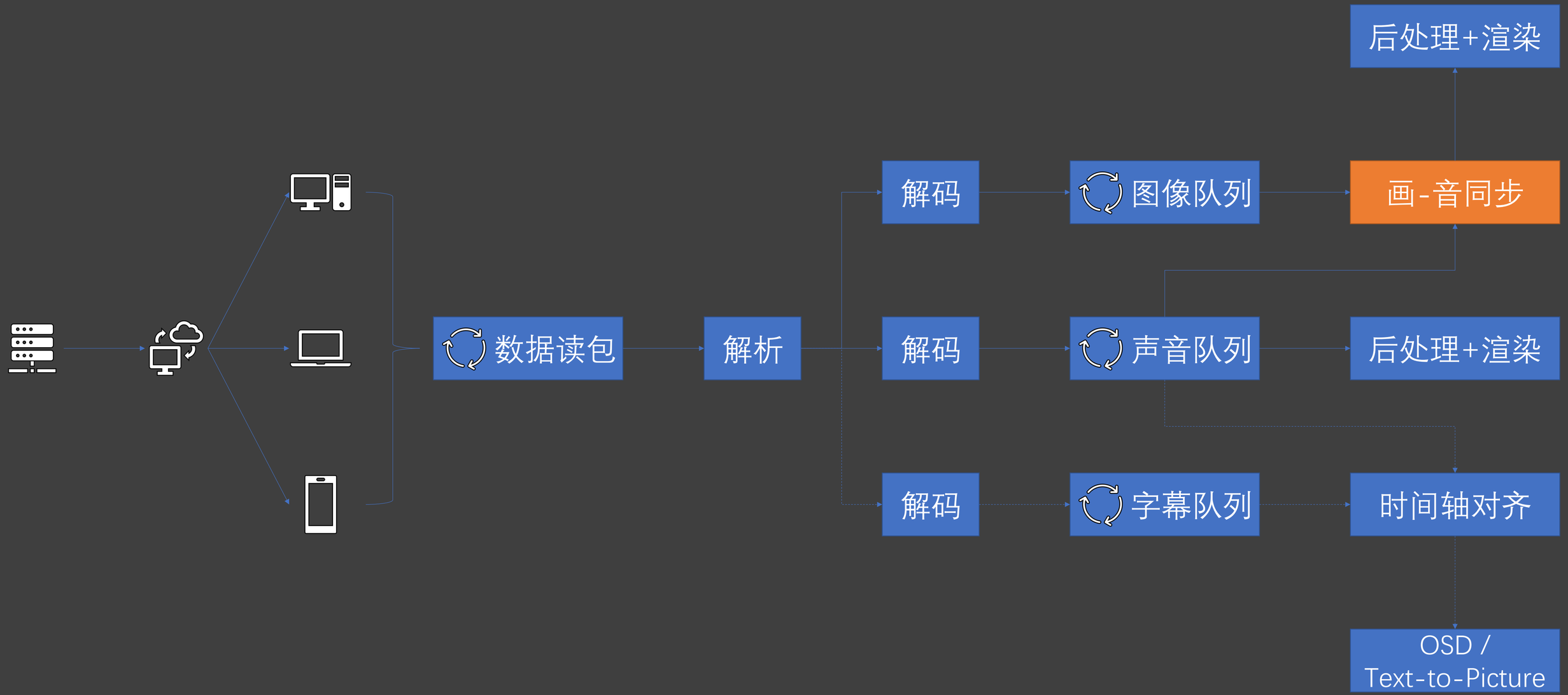
Frequency sample reconstruction

Frequency to time mapping

Decoded PCM audio output

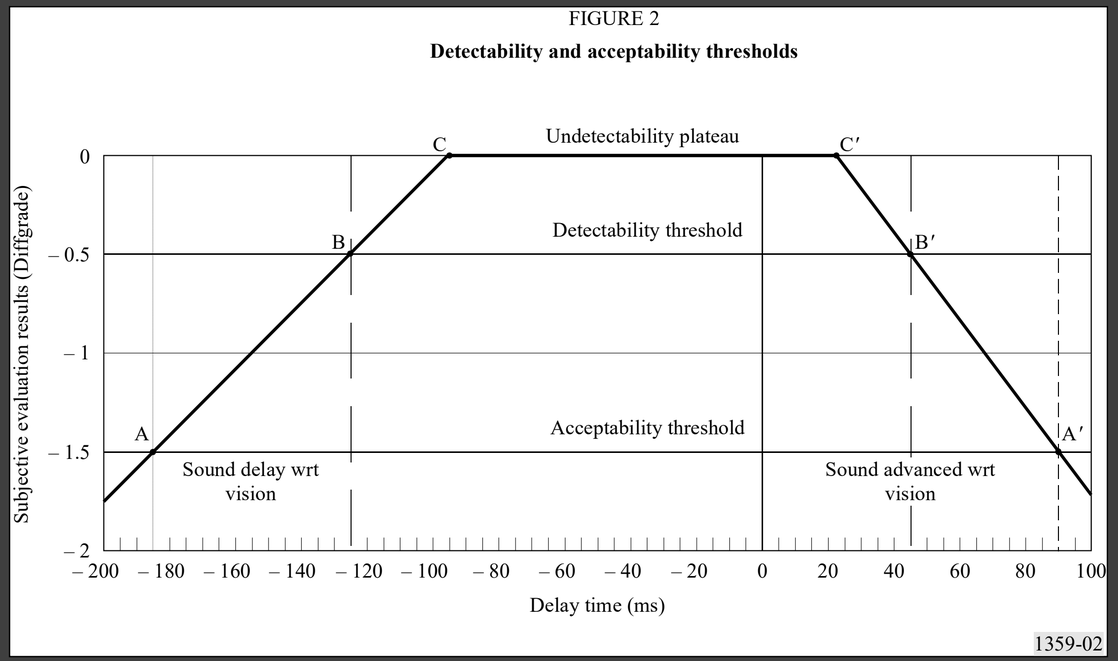
Auxiliary data if included

播放器流程框图


后处理+渲染
解码
图像队列
画-音同步
数据读包
解析
解码
声音队列
后处理+渲染
解码
字幕队列
时间轴对齐
OSD /
Text-to-Picture

终端渲染的基础：Audio-to-Video Sync (Lip Sync)

- ITU-R BT.1359-1 (1998)
- EXPERT viewers主观测试结果
  - SDTV+20’’ CRT
  - 固定距离
- 不可感知(C-C’)：+25ms ~ -100ms
- 可感知(B-B’)：+45ms ~ -125ms
- 可接受(A-A’)：+90ms ~ -185ms
- 正值：音频领先于视频；负值：音频落后于视频
- 人对声音落后于图像的宽容度更高



终端渲染的基础：播放器同步原理

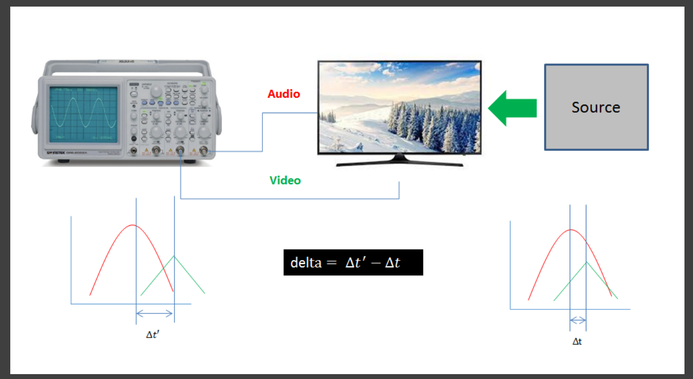
	方案A：以音频为基准	方案B：以视频为基准	方案C：A/V分离对齐
原理	• 声音是连续的（图像可认为是离散的） • 当Sample-rate、Bit-Depth、Channels参数确定时，给定Frame-size的音频数据能够客观计算播放时间 • 以当前正在播放的音频帧为基准，直接比较待输出图像帧的时戳	• 构建（模拟）系统时钟，用于对齐图像时戳 • 以当前正在播放的图像帧为基准，比较待输出音频帧的时戳	• 为A/V分别构建（模拟）系统时钟，并对应用于图像/声音时戳
优点	• 无需参考时钟：简单 • 理论上可以比较精准	• 与输出设备无关：算法跨平台	• A/V可单独调整：纠正异常PTS、跳帧 • 算法跨平台、适用性强
缺点	• 音频设备的数据缓存 • 对于单一图像序列源无效	• 自行管理时钟对齐策略，有一定复杂度 • 视频线的波动（PTS异常、主动跳帧等）会影响音频线 • 对于纯音频源无效	• 复杂度高，时钟之间需要有纠正机制

终端渲染的基础：从「原理」到「入坑」——来自华为开发手册的梳理

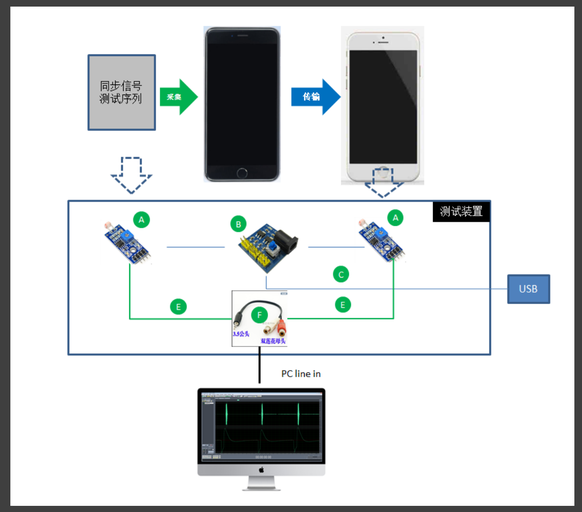
- 原理：PCM帧根据sample rate / frame size等可精确计算其timestamp / position
- 天坑：表面上对Audio Render的控制是write而非真正的output，蓝牙耳机等涉及时延场合更为明显

方式	获取音频信息	实现步骤	优点	缺点
第三方定制媒体播放器	时间戳	audiotrack.getTimeStamp()	可以精准获取端侧播放音频的时间戳	无
	音频缓存区头部位置+输出设备时延	audiotrack.getPosition()+AudioSystem.getOutputLatency()	可以相对精准获取端侧播放音频的时间戳	无
EXOPlayer	音频时间戳	audiotrack.getTimeStamp()	第三方播放器，已经实现AV同步，相对较精准。	扩展性差
	音频缓存区头部位置+输出设备时延	audiotrack.getPosition()+AudioSystem.getOutputLatency()	第三方播放器，已经实现AV同步，相对较精准。	扩展性差
标准Android NuPlayer	音频时间戳	audiotrack.getTimeStamp()	Android播放器，已经实现AV同步，相对较精准。	扩展性差
	音频缓存区头部位置+输出设备时延	audiotrack.getPosition()+AudioSystem.getOutputLatency()	Android播放器，已经实现AV同步，相对较精准。	扩展性差
Android NDK OpenSL ES框架	音频通路时延	AudioSystem.getOutputLatency()获取latency(JAVA)	NA	需要进行JAVA-Native侧交互
	音频通路时延	AudioSystem.getOutputLatency()获取latency(Native)	NA	需要加载libaudioclient.so才可以调用（不同Android版本，名字有变化。）
	音频通路时延	OpenSLES.getconfig()获取latency(Native)	第三方播放器修改最少；Native侧交互，高效。	目前OpenSLES没有实现扩展，需实现扩展。

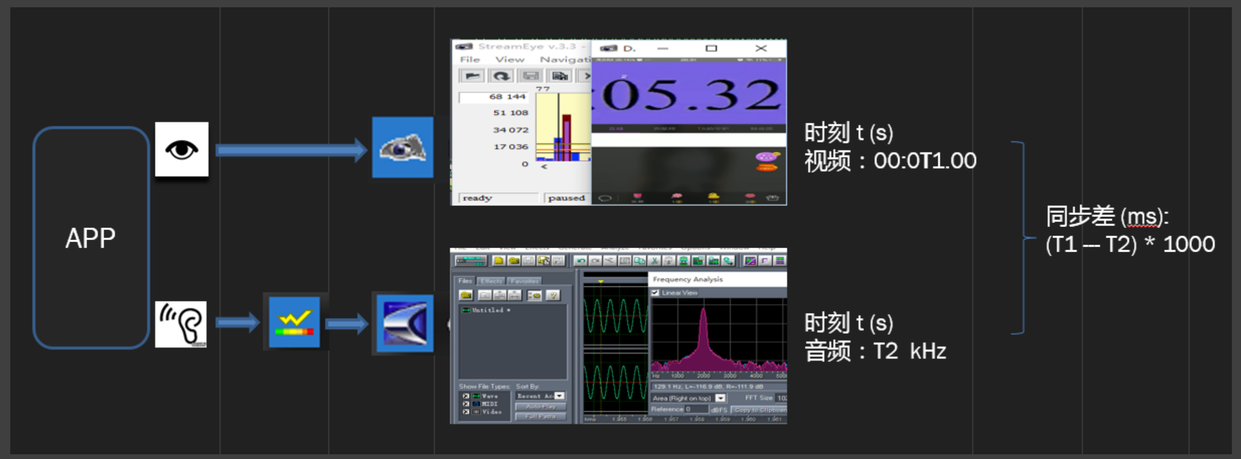
音视频同步的检测



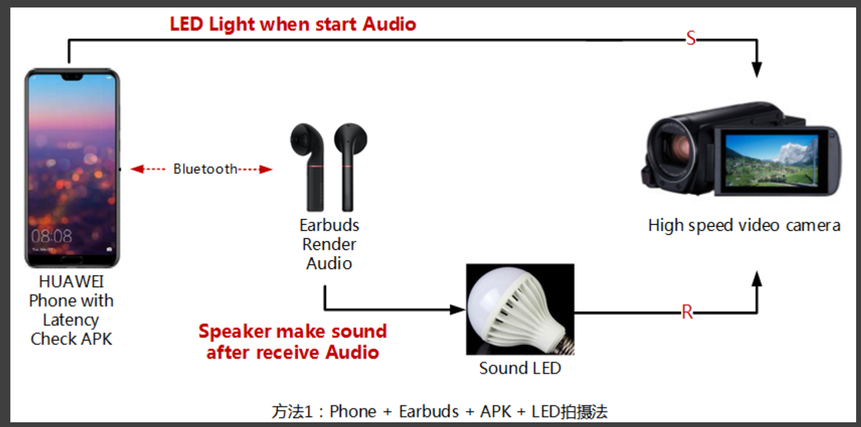
- 示波器：A/V分离后查看波峰偏移变化
- 广电、设备厂商



- CSIG音视频实验室
- 利用光敏电阻低成本实现
- RTC时延、唇音同步



- PCG研发部
- 构造特殊片源：秒表视频+与时间相关、频率线性递增的音频
- 可黑盒测试



- 华为
- 基于声控LED、高速相机
- 华为：第三方应用音视频同步接口规范

播放器架构演进

第一代

不同协议、格式片源码流

适配逻辑

Android 业务封装

iOS 业务封装

Android 系统播放器

iOS 系统播放器

- 完全依赖平台
- 无法能力扩展、性能黑盒
- 复杂适配逻辑

第二代

统一协议、格式片源码流

本地127.0.0.1代理模块

适配逻辑

Android 业务封装

iOS 业务封装

Android 系统播放器

iOS 系统播放器

- 协议和数据可与平台解耦
- 仍无法能力扩展、性能黑盒
- 对播放器行为需长期跟进适配

第三代

协议格式统一片源码流

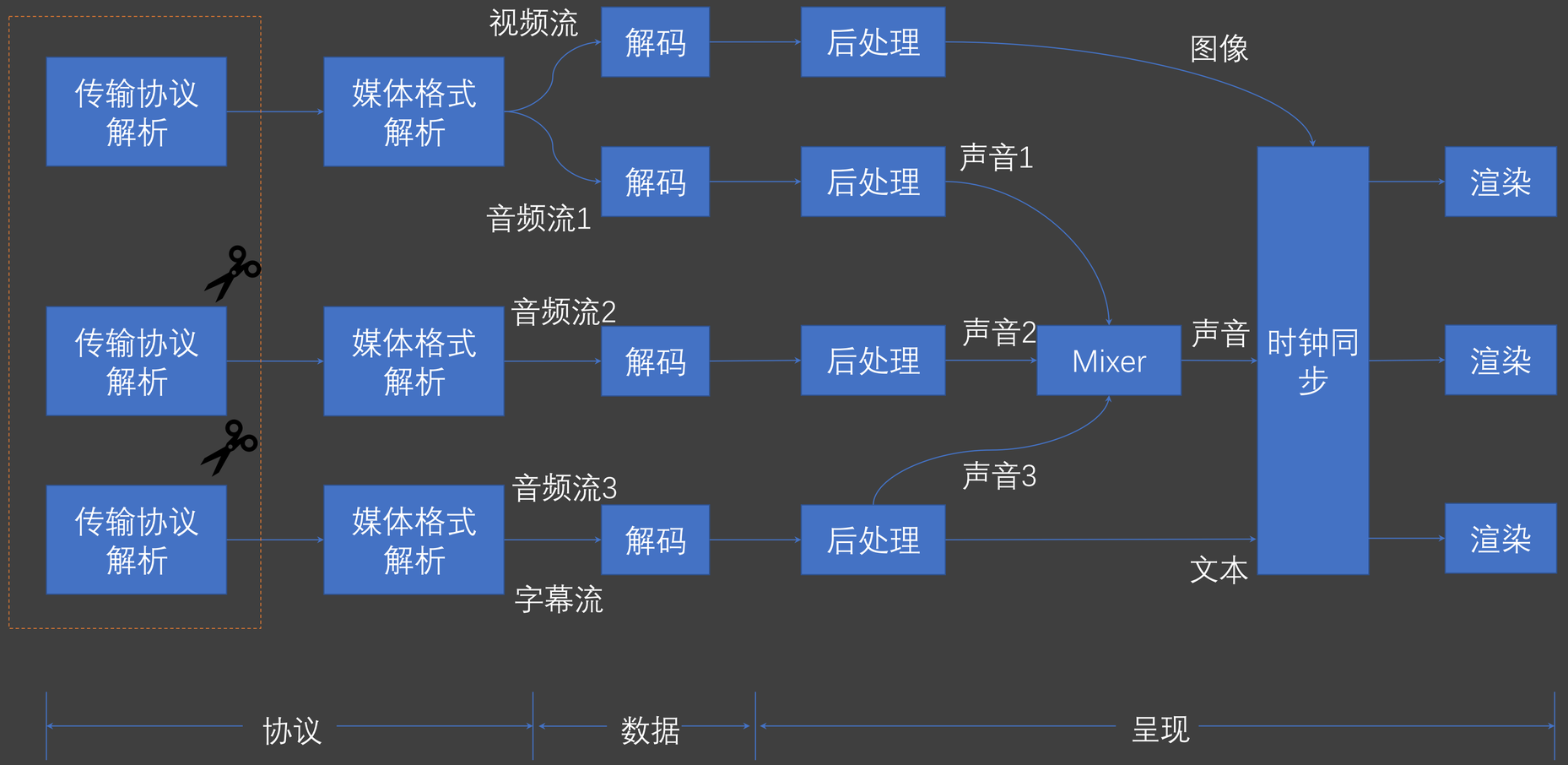
业务封装

下载 | 解析器 | 解码器 | 后处理 | 渲染

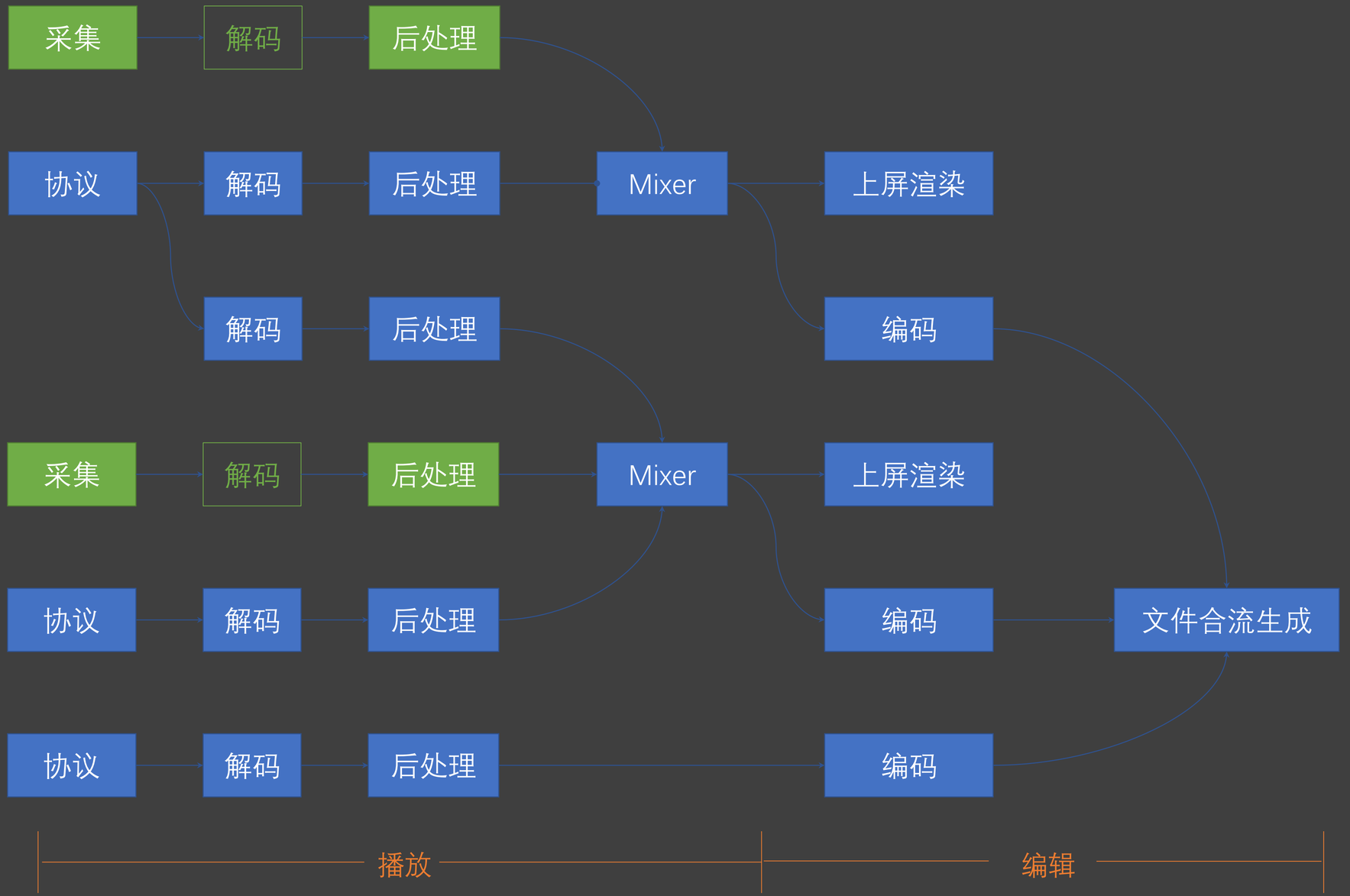
Android Phone | iPhone | iPad | OTT

- 与平台解耦、又深度利用硬件加速
- ✓ 跨平台
- ✓ 组件式，可灵活拆解
- ✓ 硬件独立适配
- 黑盒→白盒
- ✓ 灵活的能力扩展
- ✓ 随场景、全流程的深度优化
- 适配工作范围聚焦

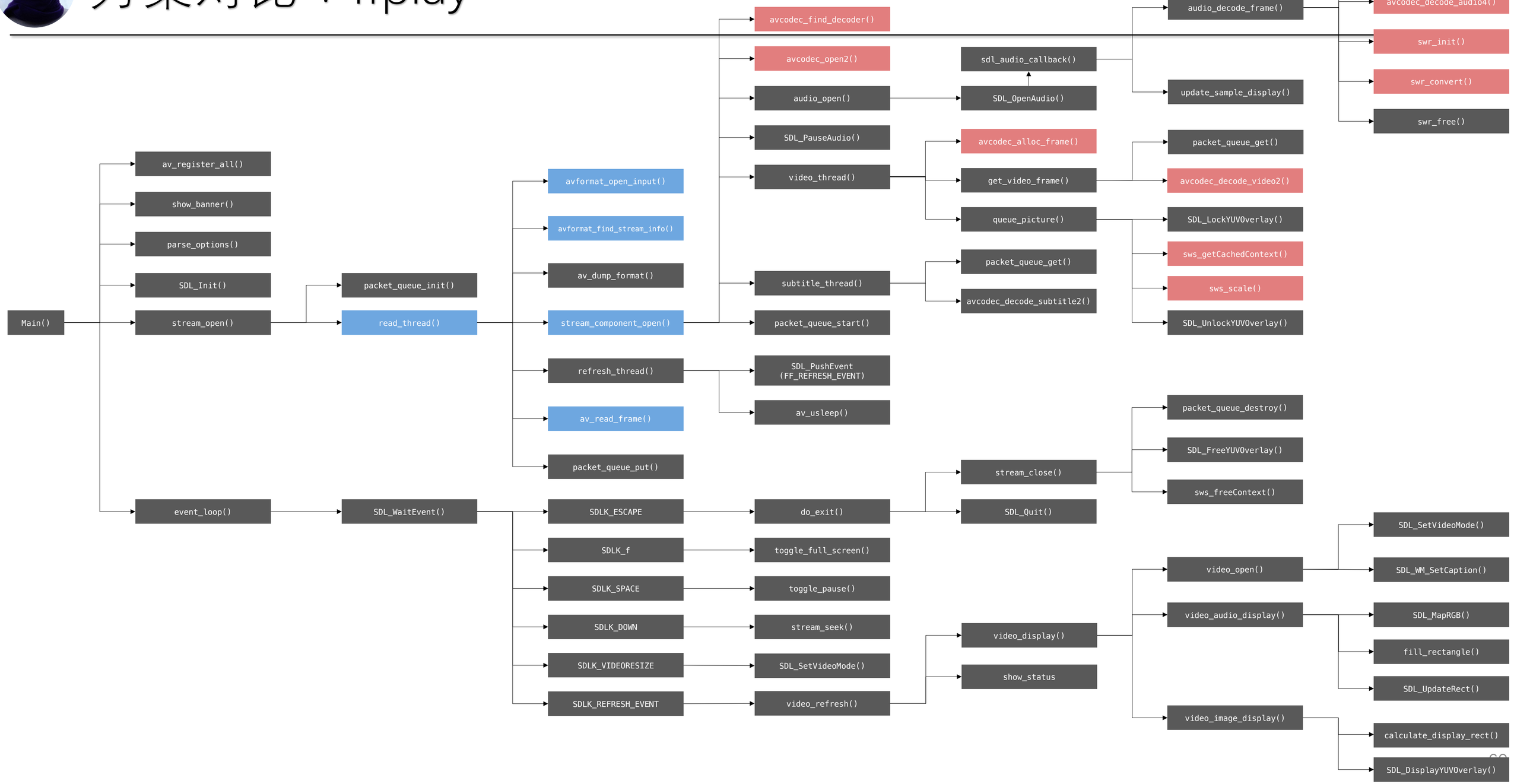
播放器架构演进：3.0+，动态可扩展



播放器架构演进：3.0+，融合



方案对比：ffplay


Main()
av_register_all()
show_banner()
parse_options()
SDL_Init()
stream_open()
packet_queue_init()
read_thread()
avformat_open_input()
avformat_find_stream_info()
av_dump_format()
stream_component_open()
refresh_thread()
av_read_frame()
packet_queue_put()
avcodec_find_decoder()
avcodec_open2()
audio_open()
SDL_PauseAudio()
video_thread()
subtitle_thread()
packet_queue_start()
SDL_PushEvent
(FF_REFRESH_EVENT)
av_usleep()
sdl_audio_callback()
SDL_OpenAudio()
avcodec_alloc_frame()
get_video_frame()
queue_picture()
packet_queue_get()
avcodec_decode_subtitle2()
audio_decode_frame()
update_sample_display()
packet_queue_get()
avcodec_decode_audio4()
swr_init()
swr_convert()
swr_free()
packet_queue_get()
avcodec_decode_video2()
SDL_LockYUVOverlay()
sws_getCachedContext()
sws_scale()
SDL_UnlockYUVOverlay()
event_loop()
SDL_WaitEvent()
SDLK_ESCAPE
SDLK_f
SDLK_SPACE
SDLK_DOWN
SDLK_VIDEORESIZE
SDLK_REFRESH_EVENT
do_exit()
toggle_full_screen()
toggle_pause()
stream_seek()
SDL_SetVideoMode()
video_refresh()
stream_close()
SDL_Quit()
video_display()
show_status
packet_queue_destroy()
SDL_FreeYUVOverlay()
sws_freeContext()
video_open()
video_audio_display()
video_image_display()
SDL_SetVideoMode()
SDL_WM_SetCaption()
SDL_MapRGB()
fill_rectangle()
SDL_UpdateRect()
calculate_display_rect()
SDL_DisplayYUVOverlay()

播放器HOWTO：流媒体架构选型

容器格式的搭配➔冗余转码

Codec格式的搭配➔HEVC/HDR/Dolby

协议封装负载➔Manifest/Index/Header/Description

存储➔文件管理、302回源

成本

兼容

CS交互➔启播/seek过程最多（最少）几次HTTP

体验

平台支持➔Web Browser的支持、外部分享、OTT厂商

播放特性➔file seeking、Def./Track switch (in seamless way)

播放器HOWTO：「启播/秒开」(1/3)原理

获取数据 → 数据解析 → 视频（音频）解码 → 渲染输出

获取数据

获取地址

- 播放地址换取服务质量
- 与应用上下文结合：详情页、Feeds流、列表等直出
- 单出 vs. 批量
- 播放需求与业务逻辑：防盗链、付费、会员等
- 有效期的更新管理

连接

- 域名解析：缓存、IP直出、接入点竞速
- 特殊情况：网络切换
- 就近接入
- 302陷阱：CDN回源
- Streaming：HTTP/ HTTPS/ QUIC
- RTC：TCP/ UDP

传输

- 并行下载/ 多连接
- 连接复用
- 赢在起步线
  - 少传输：HLS直出、控制码率
  - 懒传输：短小视频观看呈现两极化

数据解析

格式嗅探

0x00 0x00 0x00 0x18 **0x66 0x74 0x79 0x70**	ftyp
0x46 0x4c 0x56 0x01	FLV
0x52 0x49 0x46 0x46 0x50 0xE0 0x81 0x01 **0x41 0x56 0x49 0x20**	RIFF..AVI
0x47	

头数据

- 必须获取的信息：Streams/ Codec/ Profile/ Level/ SPS/ PPS及Byte Offset
- 非必要：Bit-rate/ Frame-rate
- 关键帧列表、Chunk地址
- 严格意义的按字节顺序读取、避免跳跃式读取
- 避免文件全文解析：建立索引

缓存

- 分场景：直播、点播
- 勿过度依赖文件结构组织
- 异常情况：非A/V交错组织

视频（音频）解码

解码

- GOP残缺：直播CDN缓存、RTC开头短GOP
- 解码器的创建、初始化和启动可分阶段进行
- 解码跳帧
- 勿hard code色彩空间
- 分辨率裁剪与色彩空间转换可合并
- 充分利用GPU

缓存与reordering

- GOP结构：B帧、最长参考帧、跨GOP参考
- 并行解码

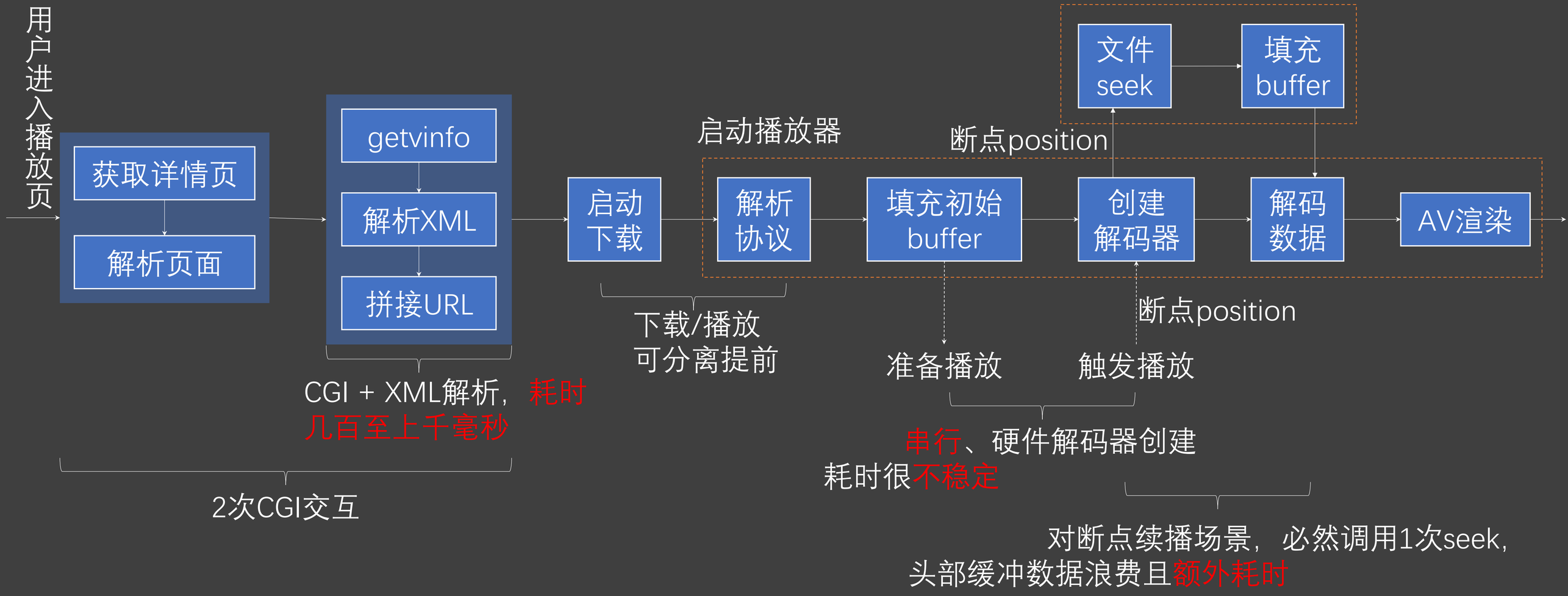
渲染输出

同步

- 严格规范A/V数据的交错组织：最长间隔
- 同步机制考虑备选与切换：容错、部分切片单轨
- 异常情况：未见其人、先闻其声
- 校准A/V两路流的时间戳偏移

等待

播放器HOWTO：「启播/秒开」(2/3)案例优化前



播放器HOWTO：「启播/秒开」(3/3)案例优化后

用户进入播放页

getvinfo

后台访问

获取详情页

解析页面

后台换取播放地址，
减少1次CGI，
且可批量获取

启动播放器$_N$

断点position

下载$_N$
高优先

解析
协议

文件
seek

创建
解码器

填充初始
buffer

解码
数据

AV渲染

并行创建
解码器

填充buffer前即seek，
避免流量浪费、省时

准备播放

触发播放

（低优先）预下载$_{N+1}$

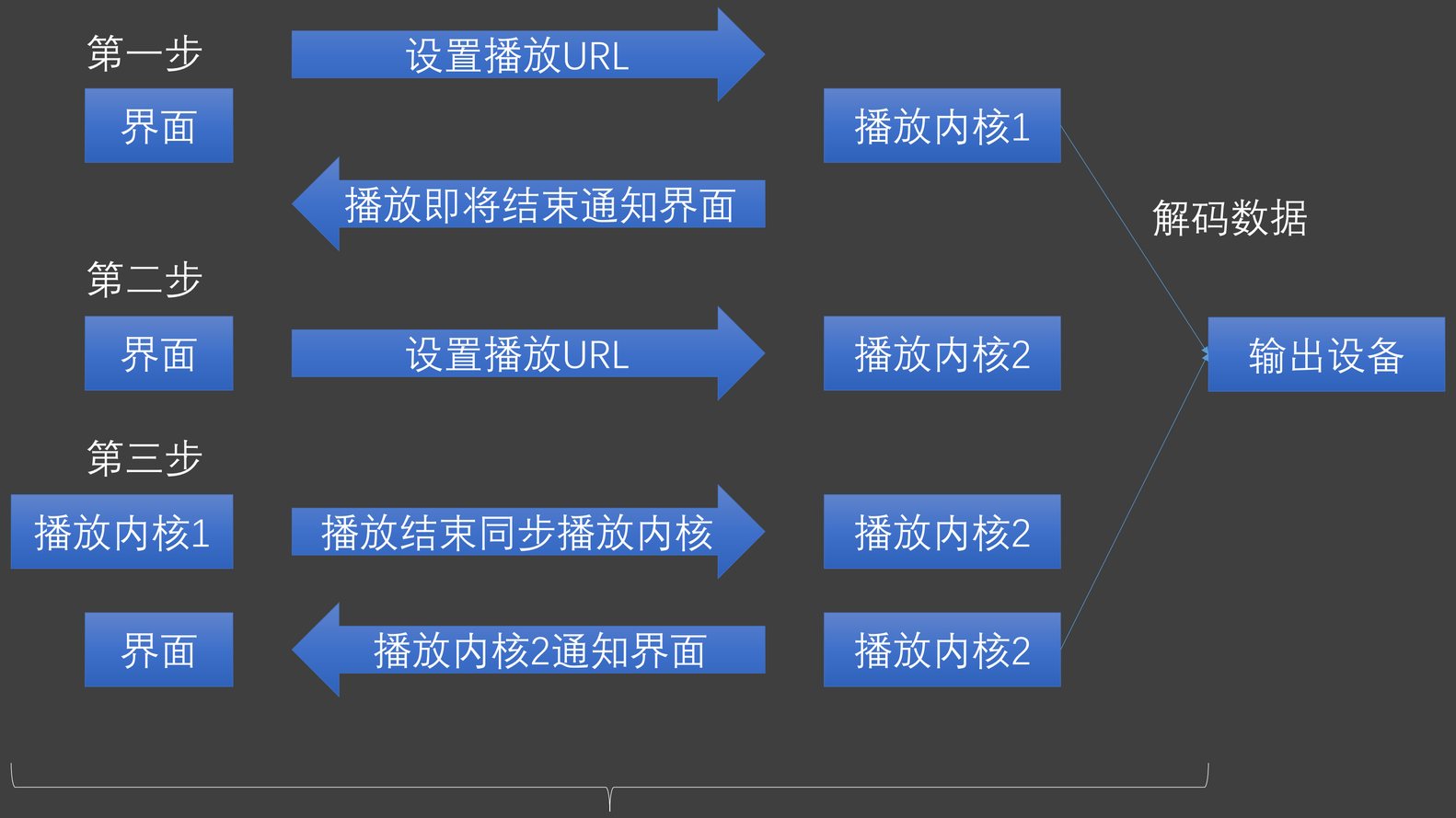
预启动播放器$_{N+1}$

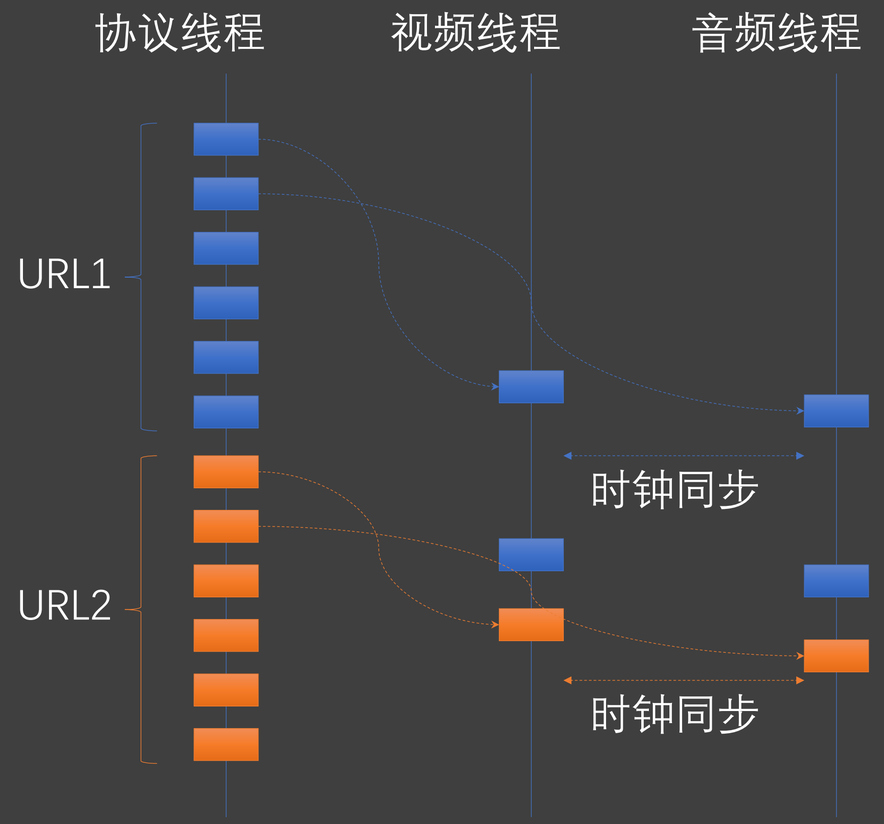
（低优先）预下载$_{N+2}$

预启动播放器$_{N+2}$

预下载、预启动，
连续播放时可直接复用

播放器HOWTO：清晰度无缝切换





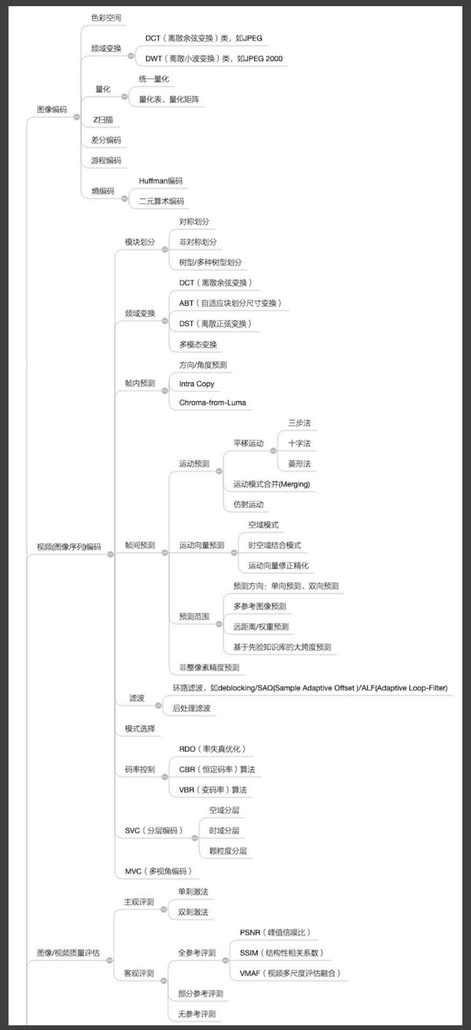
参考方案A
- 多个播放实例：更重
- 需监控输出设备数据负载、衔接时机

参考方案B
- 1个播放实例：很轻
- 完全线形、遵循AV数据PTS输出：衔接更自然

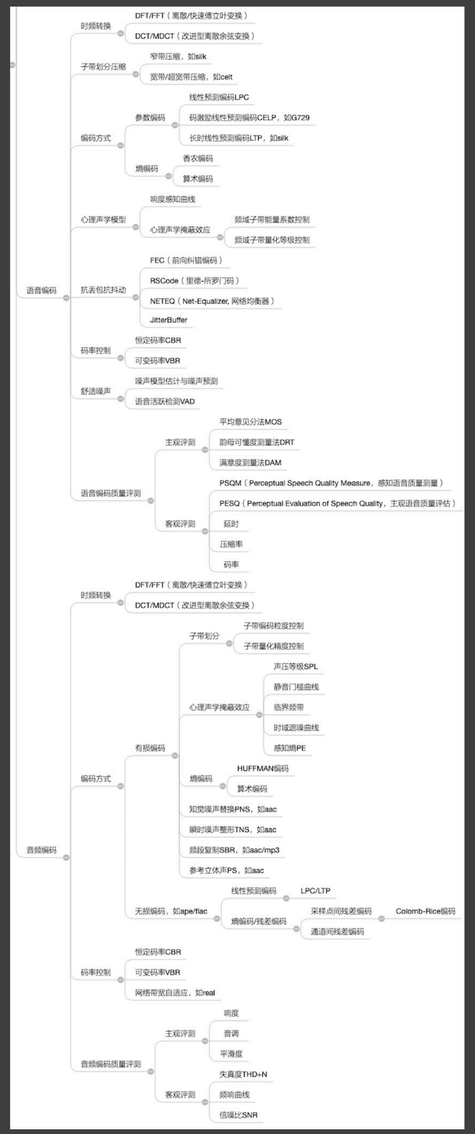
播放器HOWTO：花屏问题分析

- 输出绘制时考虑图像数据plane的padding
  - 原始图像分辨率非16倍数时
  - 色彩空间经过转换后
  - Gamma校正及色域映射
- Drift Error
  - 本质：参考偏差，即Decoder使用的参考数据或者生成的预测数据与Encoder不一致
  - 码流错误：编码模式、参考帧索引、运动向量etc.
  - Encoder使用的最长参考帧范围超过Decoder硬件能力
  - 勿混淆IDR（Instantaneous Decoding Refresh）与普通I帧
- 头信息不匹配
  - SPS – Sequence Parameter Set
  - PPS – Picture Parameter Set

后记：流媒体知识图谱

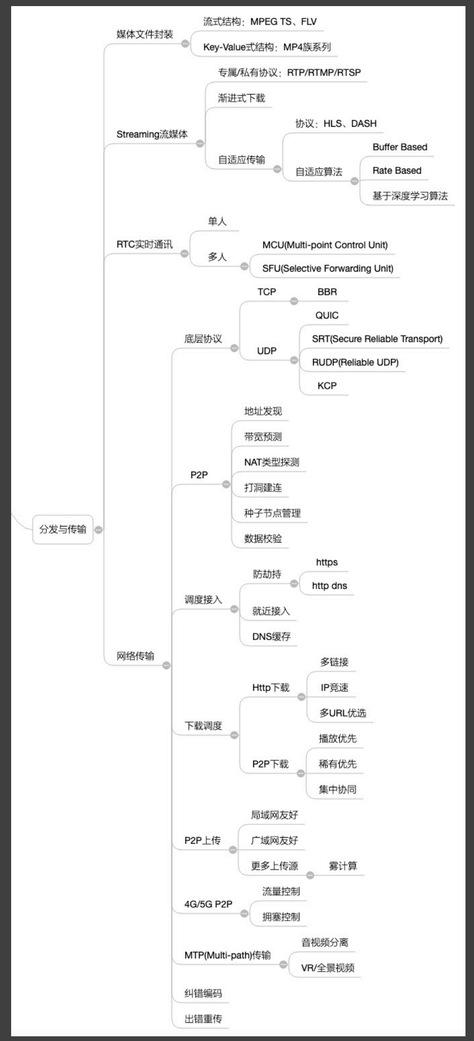


图像与视频编码



语音与音频编码

终端播放



分发与传输

Thank You
Q & A

Contact Me:
WeChat – geminili
Phone – 17688718118
E-mail – gemini@vip.qq.com