Adaptive HTTP Streaming

Jean Le Feuvre

jean.lefeuvre@telecom-paristech.fr

Overview

- **HTTP Refresher**
- **HTTP Streaming**
  - Principles
  - VoD, Live
  - Adaptive Streaming
- **Low Latency HTTP Streaming**
- **HTTP streaming solutions**

- **Annex: MPEG DASH**
  - Study of the standard

HTTP Refresher

HTTP 1.1

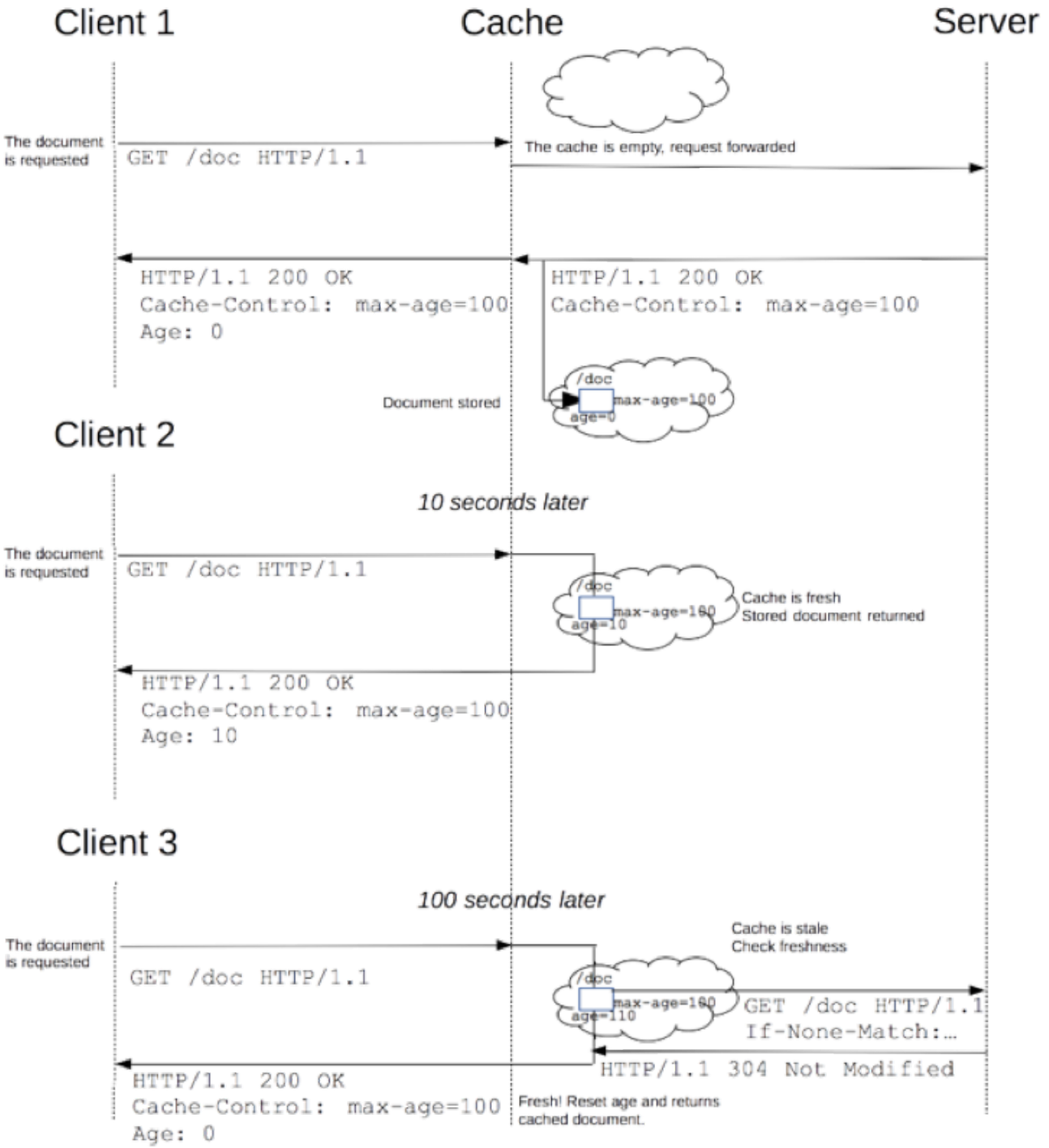
- **Basics**
  - Goal: delivery of files or bytes identified by a URL
  - Textual protocol
  - Request/answer model
    - Request: Method + URL + Headers (+ data)
    - Answer: Response Code + Headers (+ data)
- **Methods:**
    - HEAD: get info about URL
    - GET: retrieves URL data
    - POST: post data associated with a URI
    - PUT: uploads a new resource with new URL
    - DELETE: deletes the URL resource
    - TRACE: get HTTP request received by the server
    - OPTIONS: get HTTP options available at the server
    - CONNECT: for use in SSL tunnels

HTTP 1.1

Caching

- Servers may assign expiration date to resources
  - Cache-Control: public: any caching entity may store
  - Cache-Control: private: usually browser cache

- Clients may check for modifications on resources
  - Avoids re-downloading the same resource


Client 1
Cache
Server
The document is requested
GET /doc HTTP/1.1
The cache is empty, request forwarded
HTTP/1.1 200 OK
Cache-Control: max-age=100
Age: 0
HTTP/1.1 200 OK
Cache-Control: max-age=100
Document stored
/doc
max-age=100
age=0
Client 2
10 seconds later
The document is requested
GET /doc HTTP/1.1
/doc
max-age=100
age=10
Cache is fresh
Stored document returned
HTTP/1.1 200 OK
Cache-Control: max-age=100
Age: 10
Client 3
100 seconds later
The document is requested
Cache is stale
Check freshness
GET /doc HTTP/1.1
/doc
max-age=100
age=110
GET /doc HTTP/1.1
If-None-Match:…
HTTP/1.1 304 Not Modified
HTTP/1.1 200 OK
Cache-Control: max-age=100
Age: 0
Fresh! Reset age and returns cached document.

HTTP 1.1

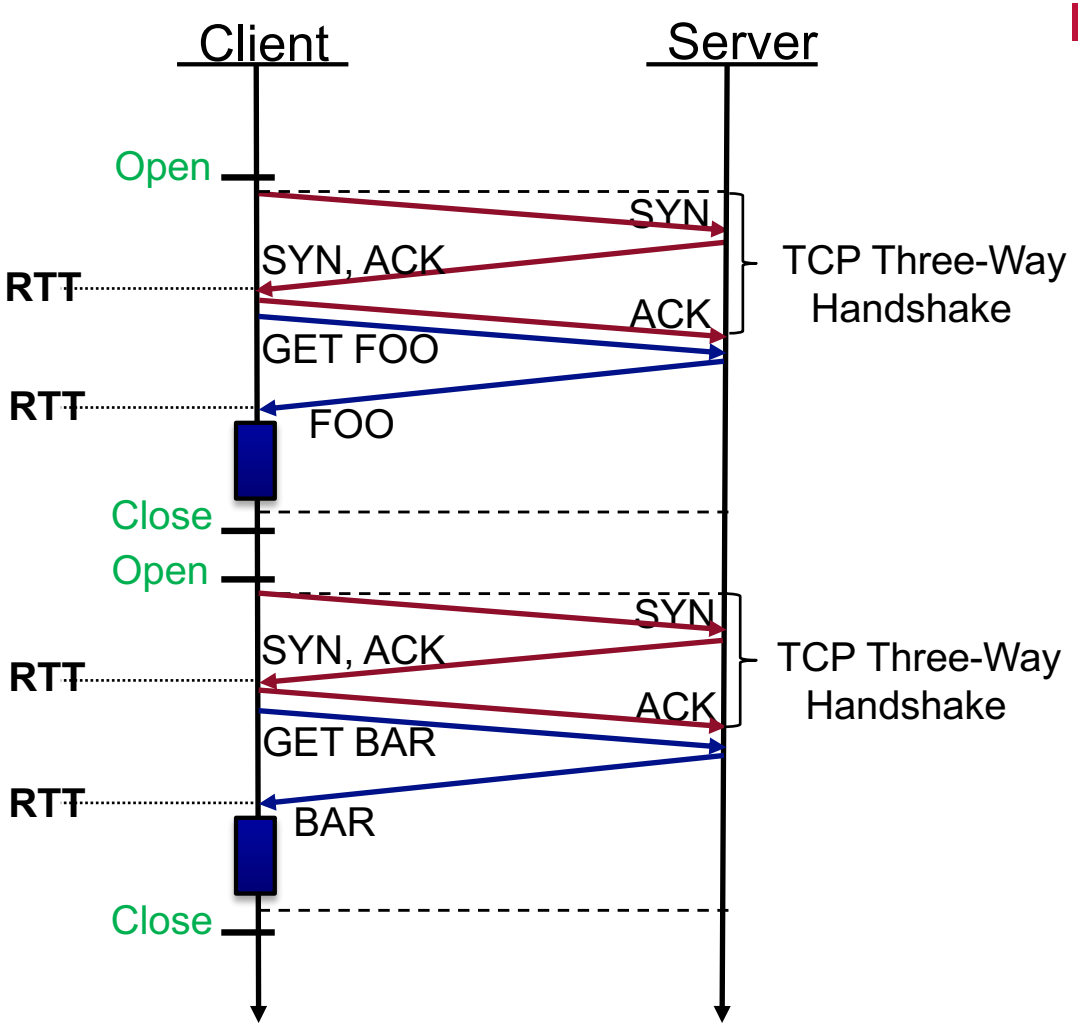
- **Resource Addressing**
  - Complete resource
  - Byte-range:
    - Closed: bytes=0-200
    - Open: bytes=1024-
    - End ranges: bytes=-500
    - Multiple: bytes=0-200,-500
- **Resource Packing**
  - Multipart messages
    - A server may pack several resources in one GET response
    - Content-Type: multipart/mixed;boundary=MyFooBarDelim

MultiPart Example

ContentType: multipart/mixed;type=….
[otherheaders…]
boundary="MyFooBarDelim—"

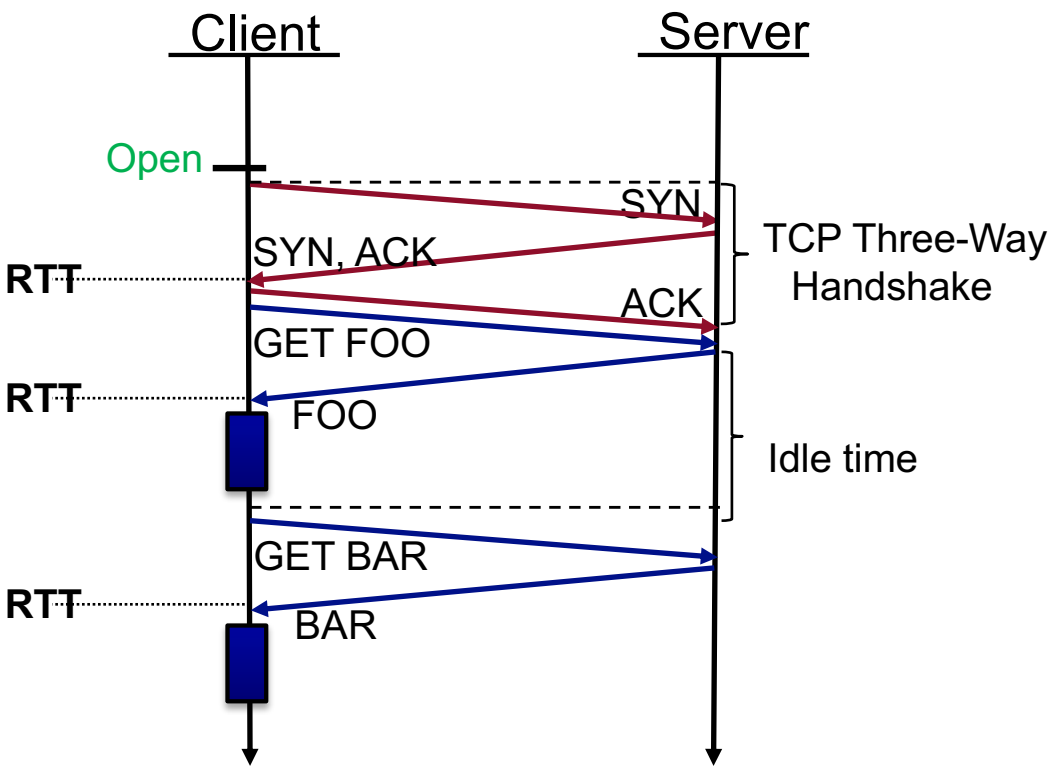
MyFooBarDelim—
ContentType: application/xml
ContentLocation: res/test.xml
[otherheaders…]
BODY_#1_BYTES
MyFooBarDelim—
ContentType: video/mp4
ContentLocation: res/logo.mp4
[otherheaders…]
BODY_#2_BYTES
MyFooBarDelim—

Optimizing HTTP 1.1 Latency



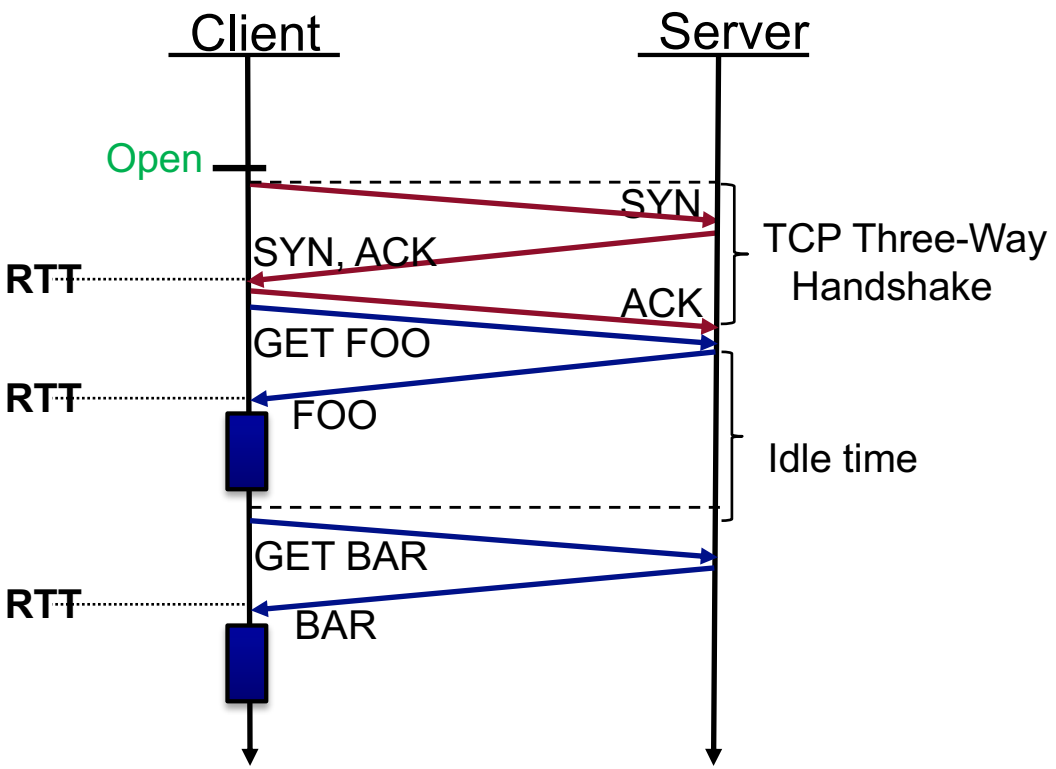
- **Connecting to the server**
  - TCP Connection takes time
  - At least 2 RTTs before the first byte arrives

Optimizing HTTP 1.1 Latency



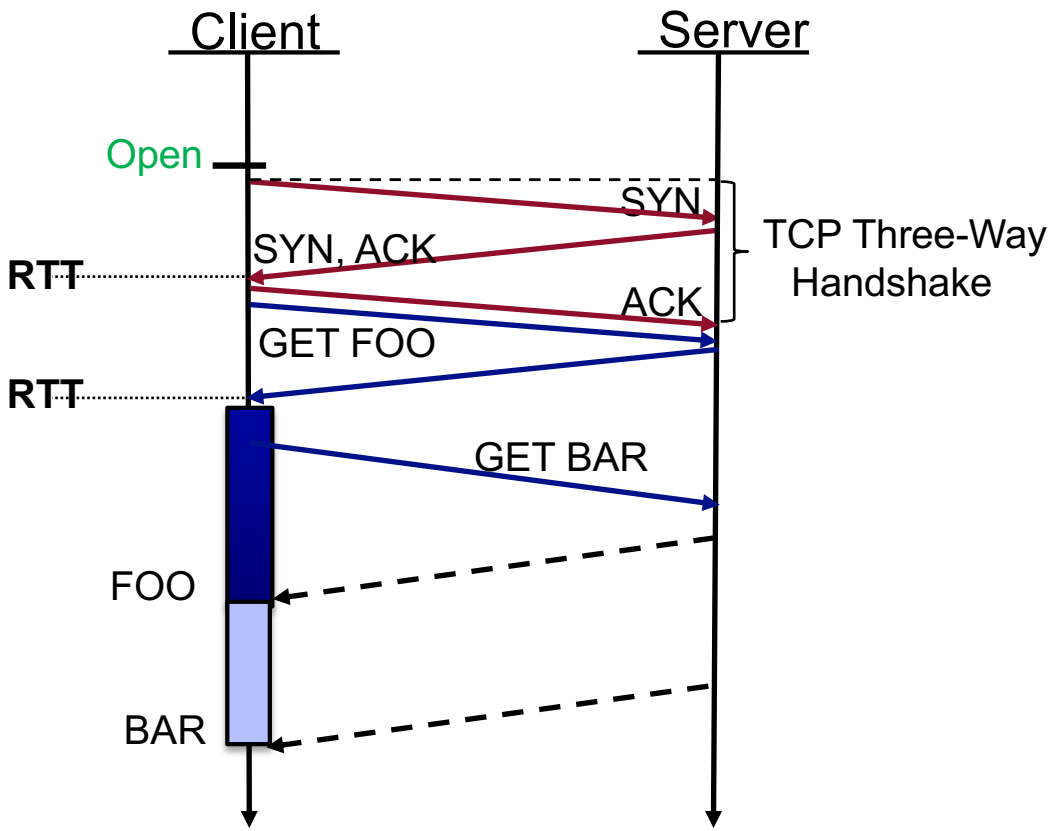
- **Persistent Connection**
  - Same connection used
  - Saves one RTT for each subsequent request

Optimizing HTTP 1.1 Latency



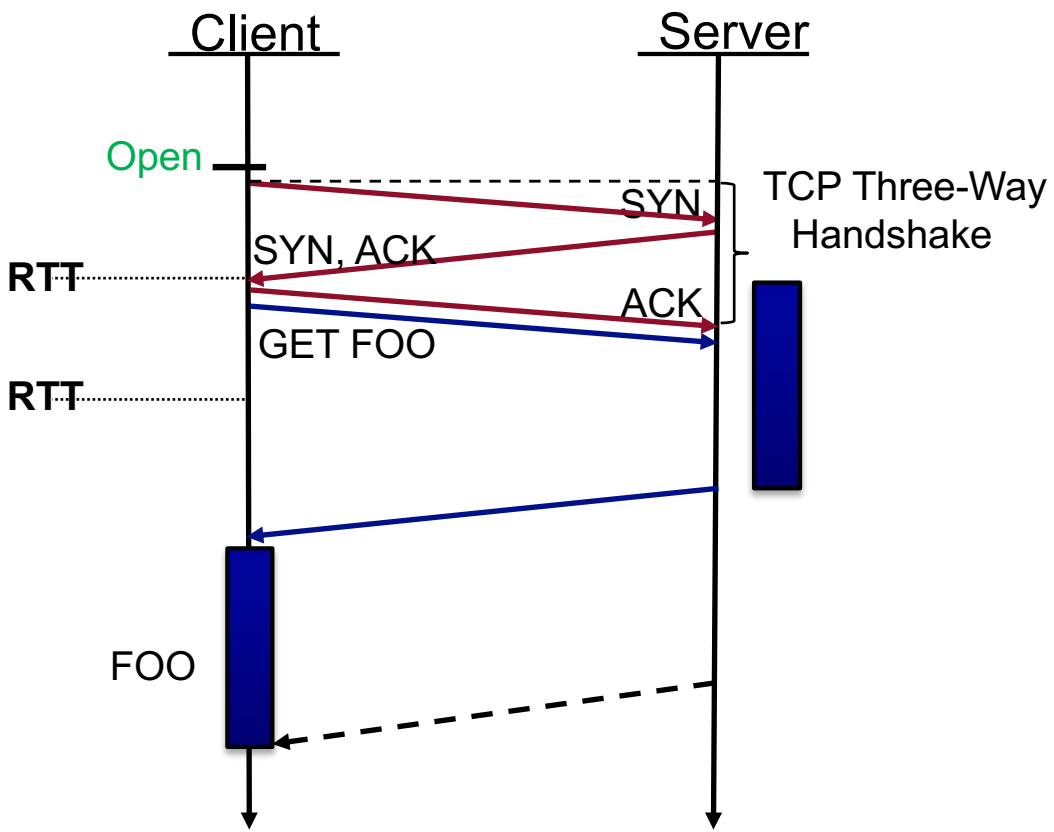
- **Persistent Connection**
  - Same connection used
  - Saves one RTT for each subsequent request
    - But still 1 RTT for request

Optimizing HTTP 1.1 Latency



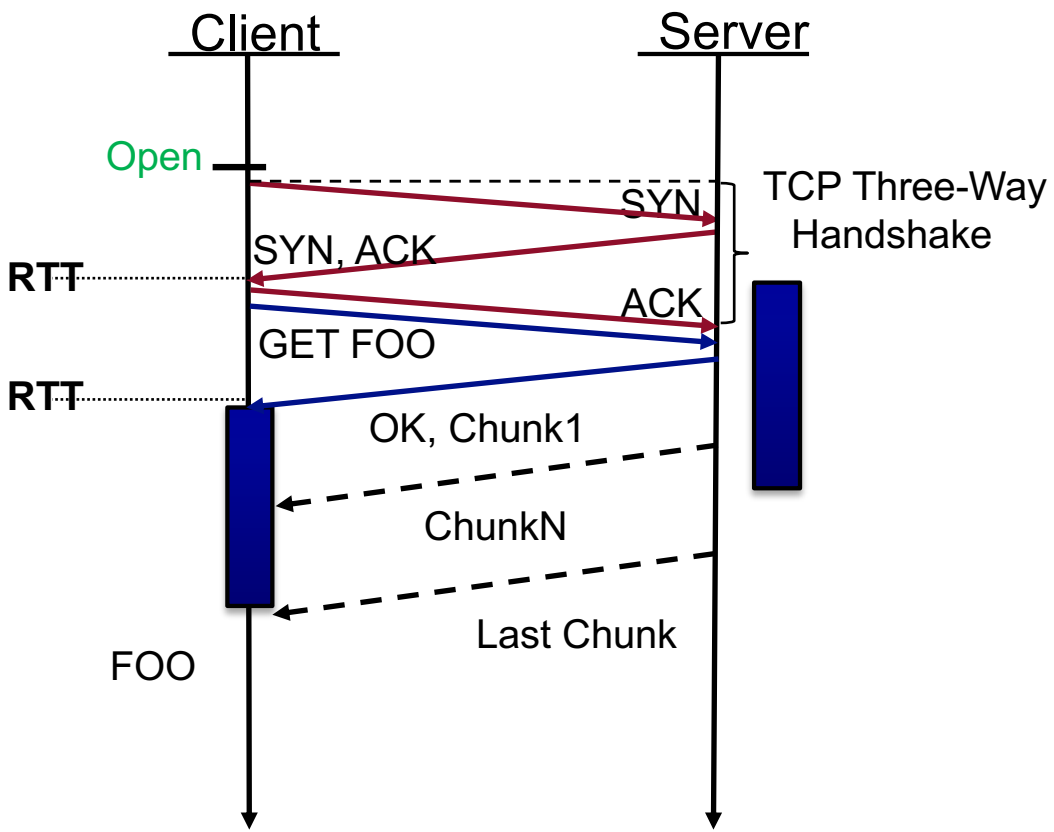
- **Pipeline Requests**
  - Send subsequent requests while downloading one request
  - Same connection used
  - No extra RTT for each subsequent request

Optimizing HTTP 1.1 Latency



- **Resource size must be known**
  - Wait for resource to be completely available at server side
  - Problematic for consumption of resource while it is being produced (live)

Optimizing HTTP 1.1 Latency



- **Chunk transfer**
  - Available parts of the resource may be sent right away

HTTP 1.1 Issues

- **No multiplexing**
  - All request are served sequentially
    - Head of Line blocking
  - Multiplexing can be simulated by using multiple connections to the server
    - Resource costly
    - Often limited by browser and/or server
    - Hard to prioritize resources
- **No cancel**
  - If resource no longer needed while still being downloaded
  - Connection has to be closed
    - Bad for HTTP streaming
  - Connection has to be re-setup
    - Bad for latency
- **Headers cost**
  - Usually always similar headers for subsequent resources

HTTP/2 Design Principles

- **Binary format**
  - Data exchanged through frames
- **Transport Layer for HTTP1.1**
  - Backward compatibility for applications
  - Multiplexing of responses and requests
    - Single connection to the server
    - Persistent connection
  - Header Compression
    - All headers from HTTP1.1 are used
- **Resource management**
  - Each resource transferred is called a stream
  - State machine for each stream at both client and server side

HTTP/2 Goodies

- **Stream Cancel**
  - A client may cancel any stream (pending or current) at any time
- **Server Push:**
  - a server may decide to push resources without the client asking
    - Ex: speed up load HTML time by pushing CSS, Javascript
- **Stream Priority**
  - Weight streams to share available bandwidth accordingly
- **Stream Dependencies**
  - Indicate sequence order for sending resources
- **Flow Control**
  - how much bytes per stream can be stored at receiver's side

HTTP/2 Downsides

- **TCP**
  - Head-of-line blocking if packet loss
    - Not convenient for multiplexed transport format (eg HTTP/2)
  - TCP retransmission ambiguity
    - Packet sent twice after timeout, ACK received: ACK of first or second packet?
  - Connection migration
    - TCP uses IP address to identify the connection, changing when migrating a session
- **TCP/TLS**
  - 3 RTTs to setup the connection
- **On-going work to fix this**
  - HTTP/3 (formerly QUIC: **Q**uick **U**DP **I**nternet **C**onnections)
    - https://datatracker.ietf.org/doc/draft-ietf-quic-http/
    - https://www.chromium.org/quic

HTTP Streaming

HTTP Adaptive Streaming (HAS) Overview

- **Transport of media files over HTTP/TCP**
  - Divide media timeline into segments of fixed duration
  - Playlist of segments
- **Why? Benefits?**
  - Simplified deployment compared to RTP
    - No need for UDP or multicast
    - Removes router configuration problems
  - Reuse of Internet Infrastructures (servers, caches, proxy, browsers …)
    - Rate adaptation does not require dedicated servers
  - Files instead of streams
    - Less overhead than RTP
    - Simplified error handling (at TCP layer rather than at application layer)

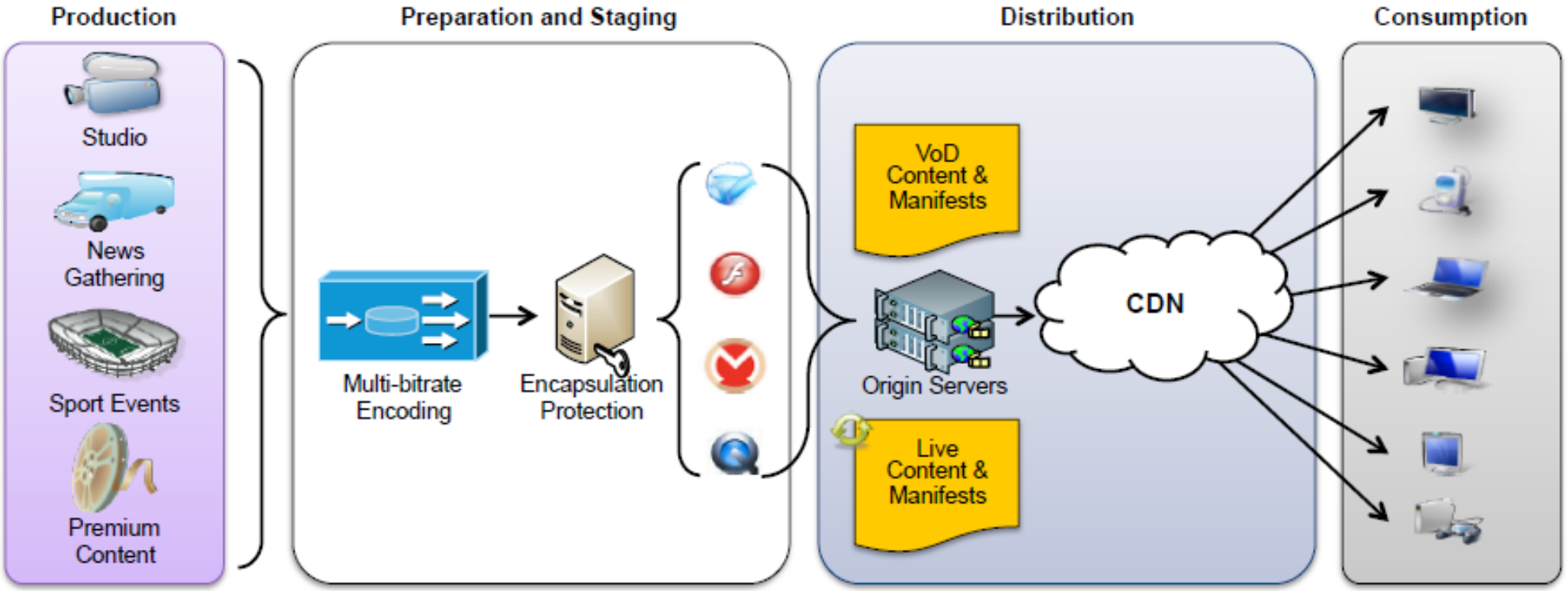
HTTP Streaming Overview

- **Challenges**
  - HTTP was made for file delivery not stream, unlike RTP
  - HTTP can suffer from
    - Large RTT
    - TCP congestion
    - Head-of-line blocking
  - Need to support broad requirements
    - Support live use cases
      - Tune-in capabilities
    - Support VoD use cases
      - Play/Pause
      - Rewind, Fast Forward

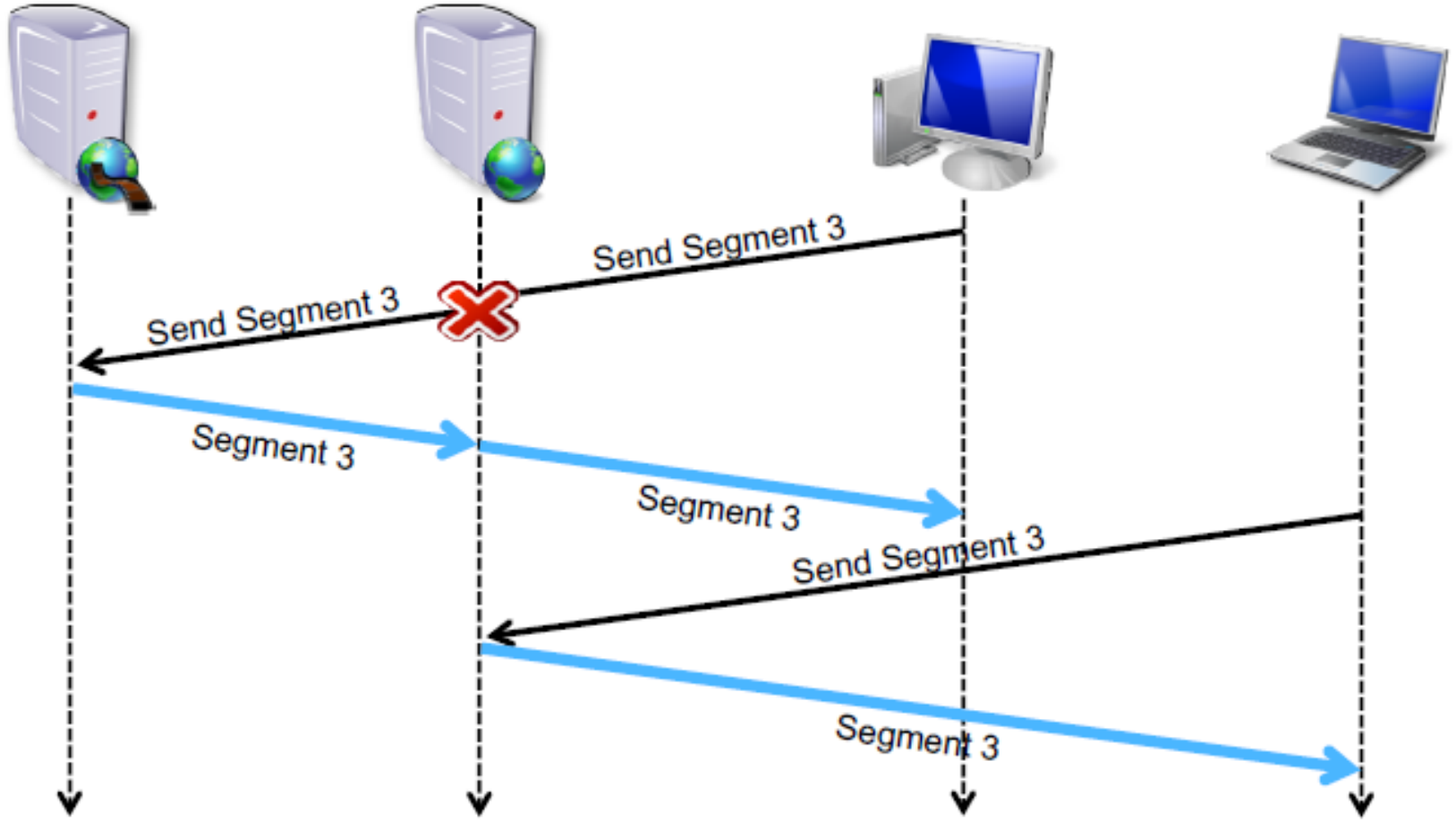
HTTP Adaptive Streaming Principles

- **Create**
  - Several bitrates / qualities
  - One file per quality
- **Split the media** in relatively short duration pieces
  - Physically: separated files (live)
  - Virtually: 1 file with byte-range<->time map (VoD)
  - Each piece can be independently decoded
- **Tell the client** about available qualities
- **Let the client**
  - Initially select the most adequate pieces to play
  - Dynamically decide the quality/bitrate to play based on available bandwidth
- **Update the description of streams**
  - Live
  - Advertisement insertion

HAS Distribution Chain



HAS: caches and CDN in the ditribution



ICME 2013

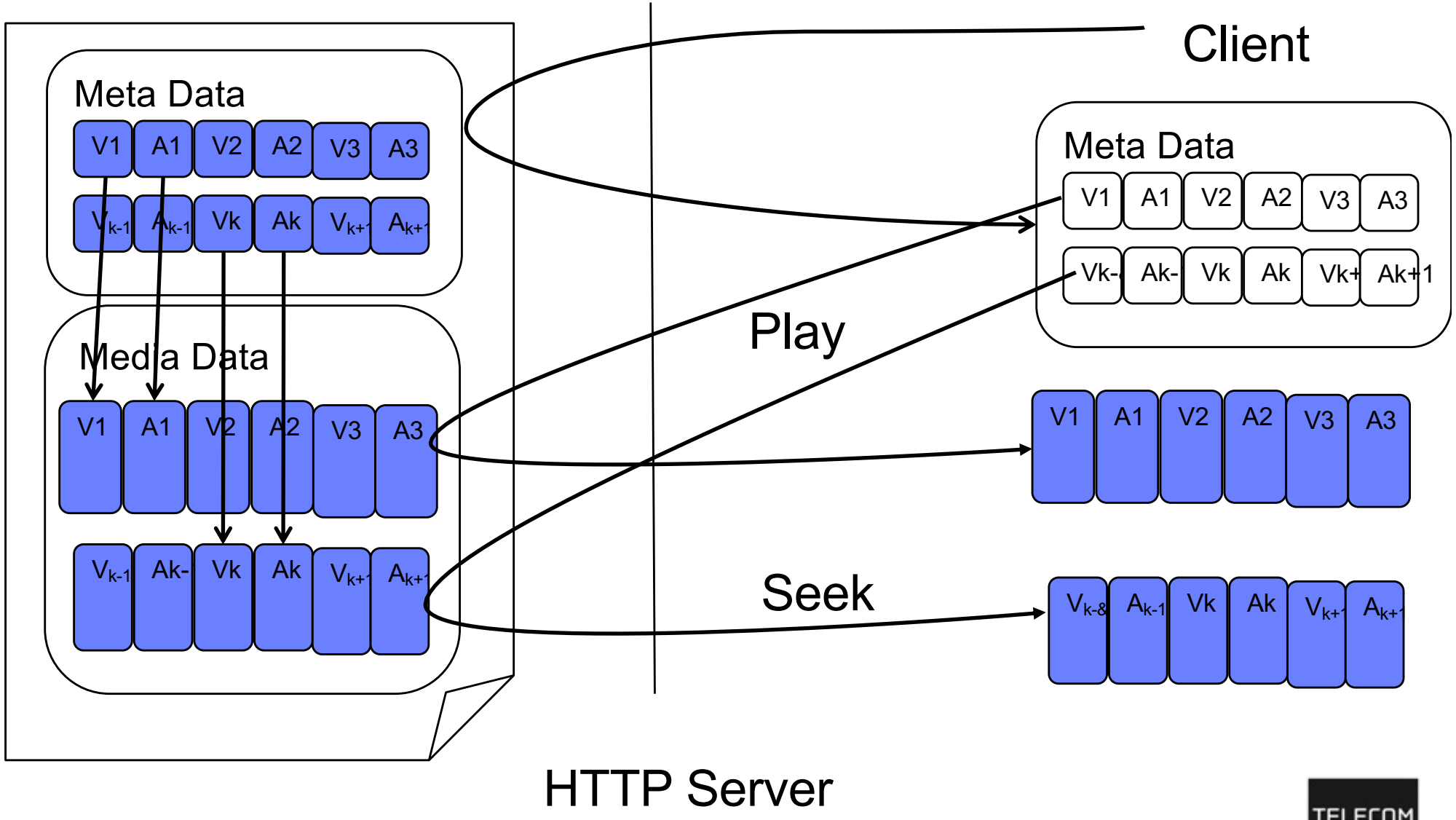
HTTP Streaming: Web Radio Example

- **Already deployed in 1999 !**
  - Solutions/protocols: Ice-Cast, Shout-Cast
- **Principles**
  - Send an indefinite file over HTTP
    - Content-length = 0 (not really standard)
  - File is read by client but never stored
- **Limitations**
  - Single stream
    - Elementary or multiplexed
  - Hard to cache: same URL for different content at different times
  - Difficult to support VOD
  - Difficult to support bandwidth adaptation

HTTP Download and Play

- **Simple Playback**
  - Descriptive meta-data at the beginning of the file
    - Number of tracks, codecs, …
  - Data placed after meta-data
  - Fetch done via HTTP GET
    - Whole file is being downloaded
- **Advanced Playback**
  - Filter data
    - Eg, HTTP GET for video and audio track 1 only
    - Requires precise description of media AUs position in file
  - Random Access (seeking)
    - Requires time to byte range/offset map
  - Requires HTTP Partial GET (HTTP 1.1)
    - Data is fetched by byte range in the file

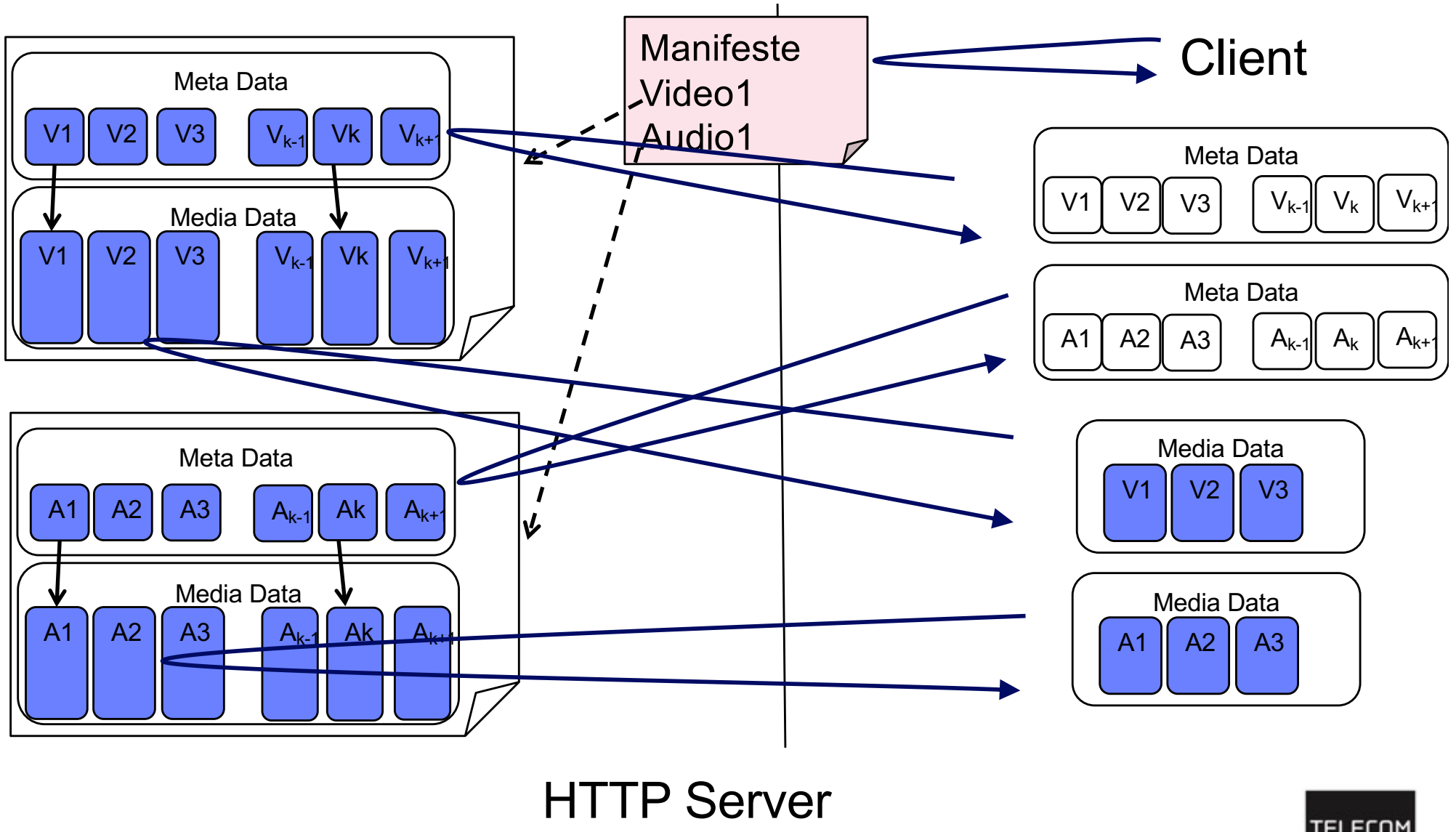
Advanced HTTP playback



HTTP VoD

- **Problematic**
  - All streams are described in one metadata block
    - Large size, initial loading time long
    - Some streams might not be used by the client
      - Unsupported codecs
      - Undesired language
- **Solution**
  - Higher-level description of the streams
    - Small description file (text, xml): « manifest »
    - codecs, languages, bitrates, etc …
  - Split the streams
    - One media stream per file
      - Exception HLS+MPEG-2 TS: one audio stream multiplexed with each video quality
    - Client selects useful streams
      - Fetches associated meta-data blocks
    - Inter-stream synchronization is done by the client
      - All streams use a common time base regardless of number of files
    - « Late Binding »
- **Requires media-time/byte-offset mapping**
  - In-band mapping: download the file header first
    - For formats supporting this (ISOBMFF)
  - Out-of-band mapping: use an additional file
    - For formats not supporting this (MPEG-2 TS)

Lecture HTTP avancée: Exemple



Live Streaming over HTTP

- **Problems**
  - Live data needs to be delivered frequently
  - HTTP Infrastructures work best with files (cache, proxys)
  - Difficult to cache indefinite duration files (web radios)

- **Solution**
  - Divide the “virtual” file into small « Segments » of duration Δt

    $$\infty = \Sigma_0^{\infty} \Delta t$$

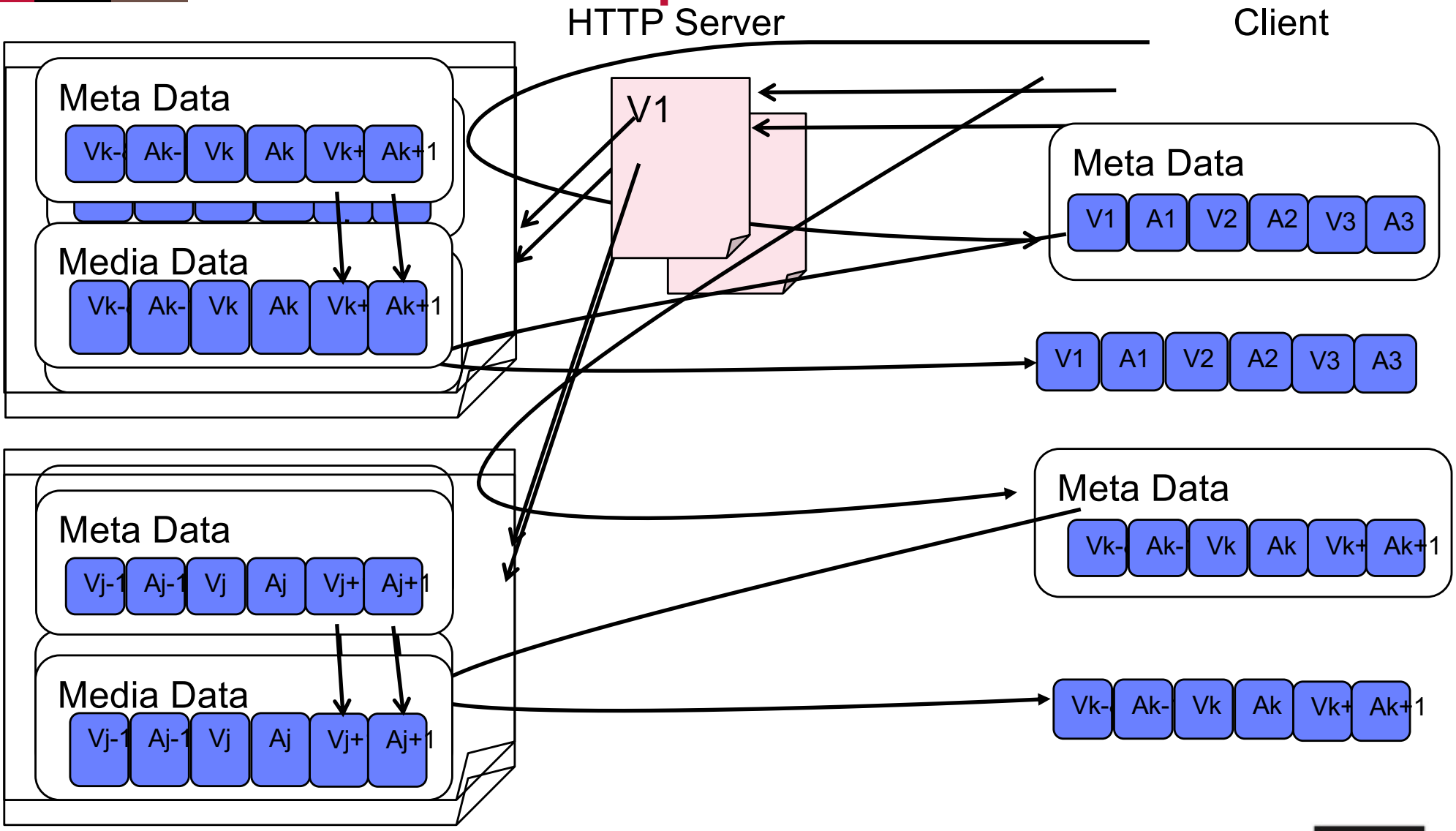
    - latency increased by **Δt**

  - Segments to download are described
    - Solution 1: In-band: (n-1)-th segment indicates the n-th segment URL
    - Solution 2: Out-of-band: by a playlist/manifest
      - Explicitly naming of each URL
      - Implicitly naming based on time or numbering, template URL

  - Players start by buffering some segments
    - Latency in the order of a segment duration (Nb_{Seg}*Δt)
    - Low latency mode: buffer less than a segment, playback while download live edge

  - Players regularly
    - download next segments
    - update the playlist

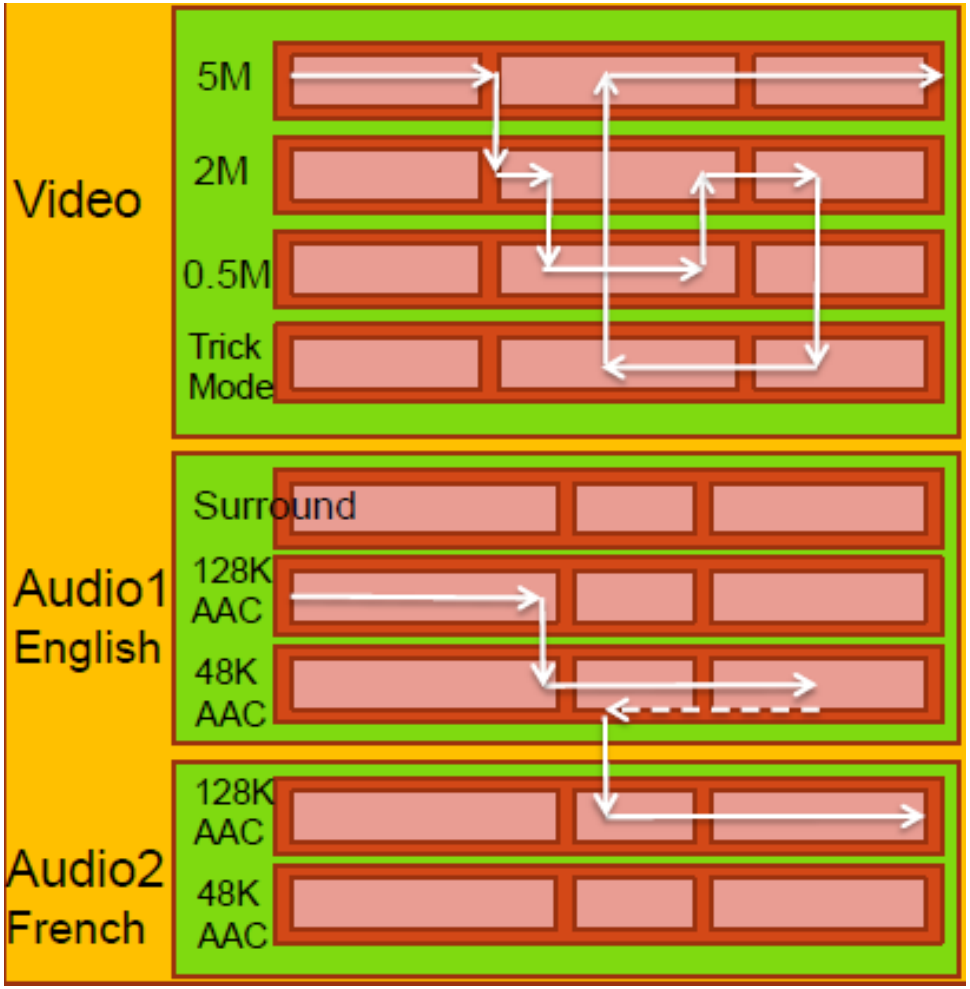
HTTP Live: Exemple



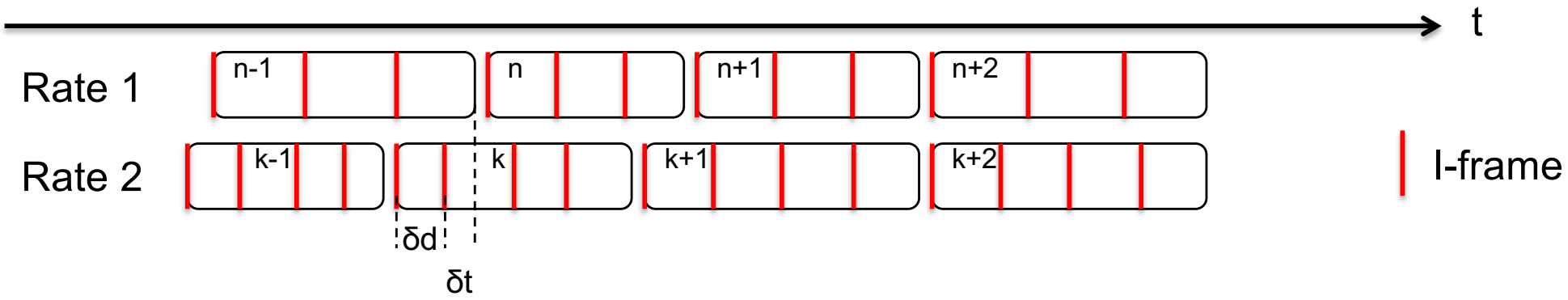
Principles for Adaptive Streaming

- **Problems**
  - Avoid re-/buffering issues due to TCP congestion
  - Offer different qualities to different clients
- **Solution**
  - Use of alternative files corresponding to different bitrates of the same segment
    - Concatenation of files from different bitrates still makes a valid stream
  - Alternatives indicated in a playlist
  - Delivered independently
- **Increased client complexity**
  - Bandwidth/buffer estimation
  - Adaptation logic
  - Switching strategies (double download, abandonment …)
- **Streams Encoding**
  - Make sure that 2 segments at different qualities/resolution can be chained
    - Ideal case: physical concatenation of the files result in a valid file containing a valid media bitstream

HTTP Adaptive Streaming Session



Examples and Problems in Bitstream Switching



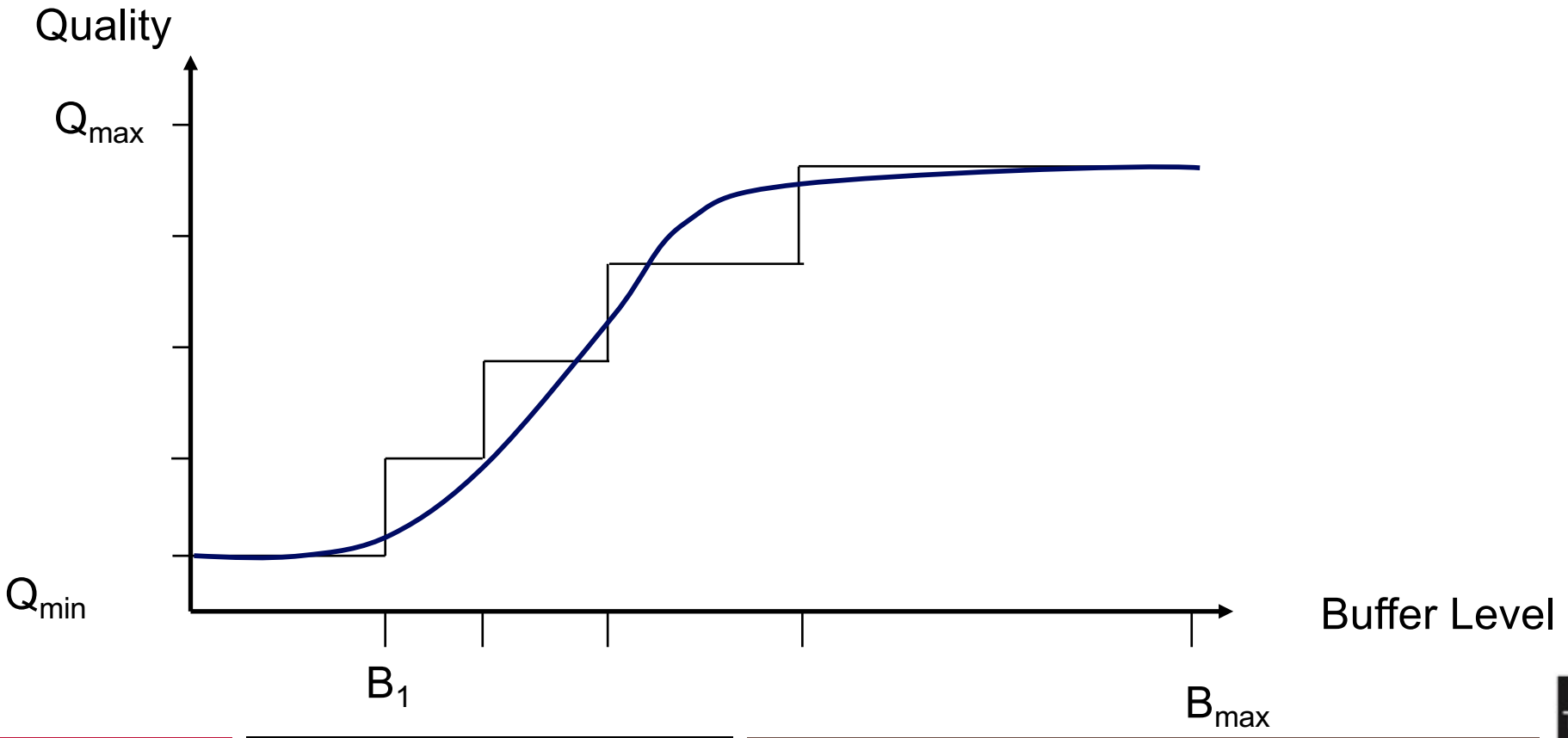
- **$R_{1,n-1}$ -> $R_{2,k}$**
  - Useless download of δd media
  - Double decoding of δt media
- **Solution: Temporal Alignment of Segments**
  - Removes double download (δd=0) & double decoding (δt=0)
  - Simplifies temporal description of segments

Adaptation Logic Challenges

- **Bandwidth-based adaptation *(Conventional, PANDA)***
  - Estimate the current bandwidth to select best quality
  - Issues
    - Congestion may happen
    - Cache miss, cache hit, local cache alter estimation
    - Server processing time unknown
- **Buffer-based adaptation *(BBA, BOLA)***
  - Monitors amount of buffered media time to select best quality
    - Can be seen as a smoothed version over time of bandwidth-based
  - Issues
    - Requires long buffer times (memory impact)
    - Less reactive
    - Buffer maps hard to predict (VBR)
- **Mixed approaches *(ABMA+)***
  - Buffer-based + segment download time

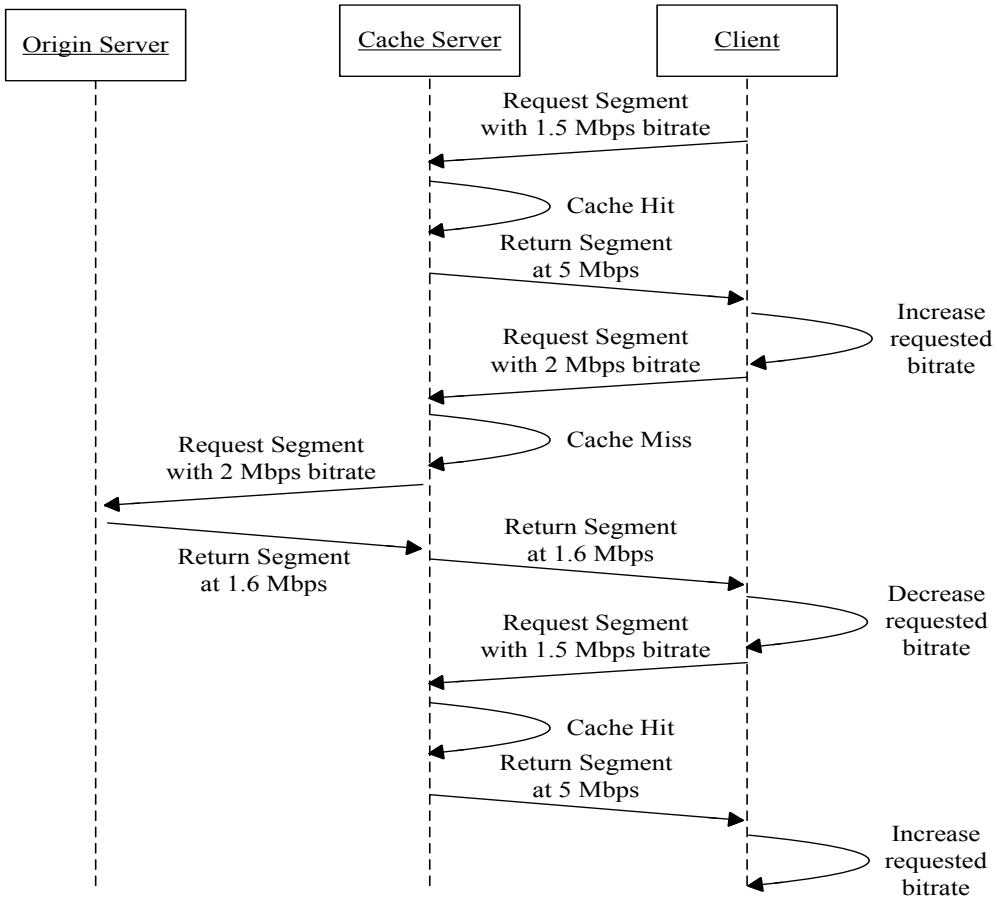
Buffer-based adaptation example

- **Define/precompute a buffer map**
  - Switch quality when buffer occupancy index change



HTTP Cache

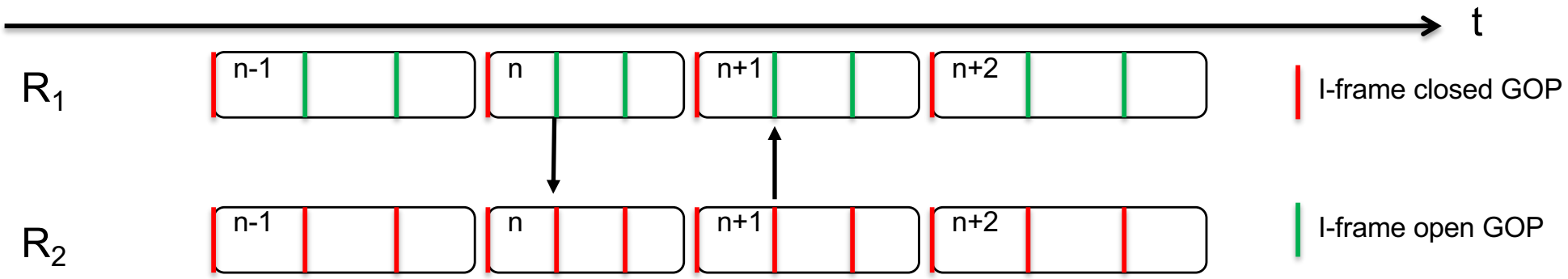
- **Reliable estimation of available throughput**
  - Cache miss, cache hit, local cache ?
  - Estimation of waiting time at server side ?



Live Adaptation Logic Challenges

- **Latency**
  - Lower the amount of media buffer
    - Consume segments while downloading them
  - Compromise between buffer-based and bandwidth-based
- **Live edge**
  - Most recent segment produced
  - If request too late
    - Longer latency towards edge time
  - If request too early
    - 404 !
- **Client logic: 404 or not 404 ?**
  - HTTP / OTT (no loss : TCP)
    - Usually implies request too early
  - File Datacast : (losses : multicast, broadcast)
    - May imply request too early or lost segment !
- **Segments duration**
  - longer: better coding efficiency, less HTTP requests
  - shorter: better reactivity to network condition changes, less latency

Random Access vs Bitstream Switching

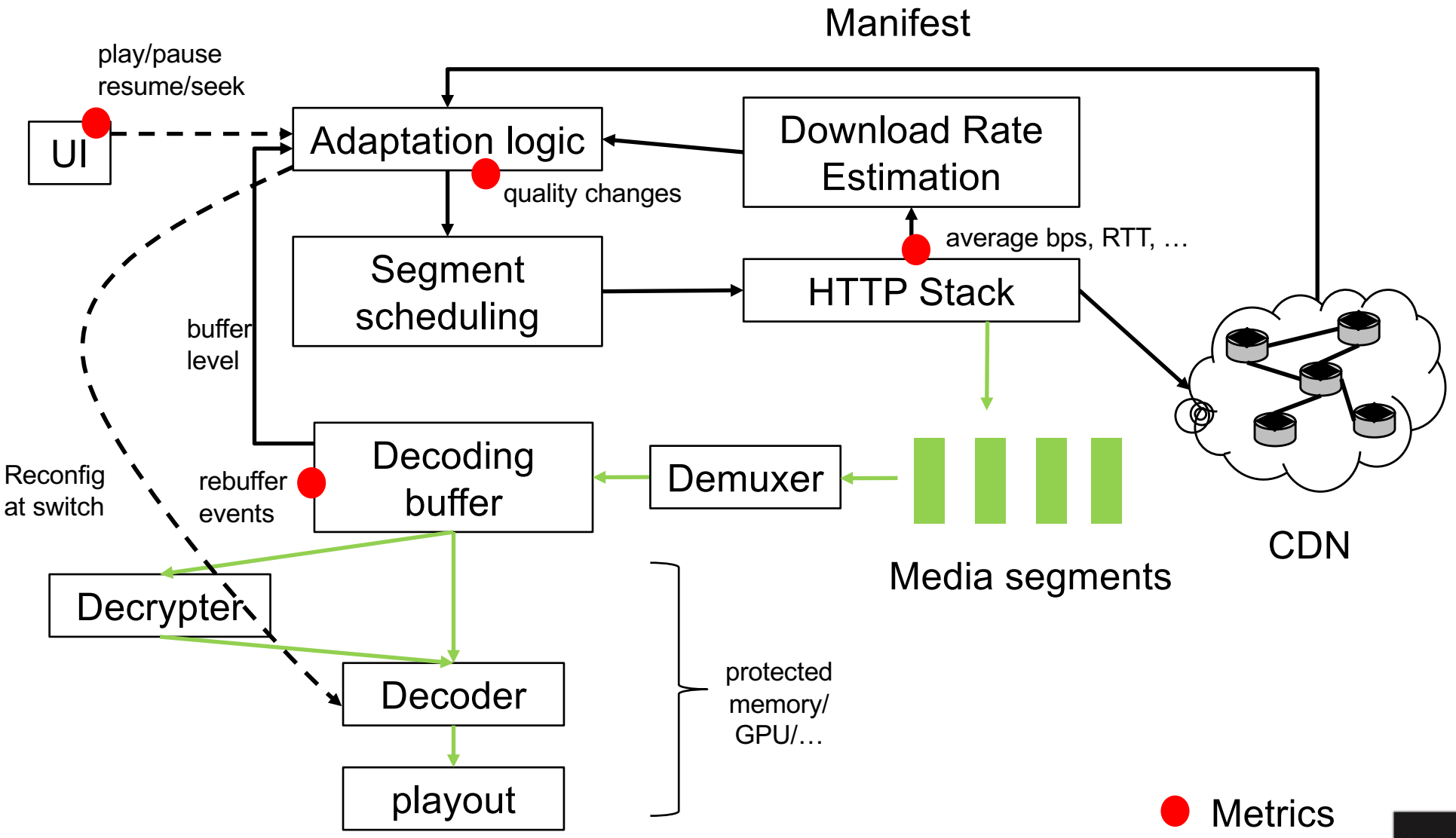


- **Tune in**
  - R_1 and R_2: possible within the segment
- **Switching**
  - R_1->R_2: possible within the segment
  - R_2->R_1: not possible within segment
    - Depends on reference picture compatibility between R_1 and R_2

HAS impact on network

- **Variable cache bloc size**
  - VOD: large files (Go), Live: small segments (Ko, Mo)
  - Not the same storage configuration (file systems) for the caches
- **Number of files**
  - From millions (Vod) to billions (live)
  - Impact on file database capacity of caches
  - Impact on cache logs
- **HTTP stateless**
  - Hard to track in the CDN the QoE of a session
  - Hard to track in the CDN the number of simultaneous users (only the number of simultaneous downloads)
  - Access Authorization more complex: manifest and segments
- **Thundering Herd**
  - Concurrent access to the same URL can trigger multiple request towards the origin server!
  - Cache for HAS usually more clever than cache for static web content
- **Non normative clients**
  - Hard to predict their behaviour (nb requests, bandwidth estimation based on client activity), hence hard to optimise the cache.
- **Network overloaded by requests**
  - Signalling Overhead (manifest)
  - Uplink traffic important (request headers)
  - TCP (number of ACKs)

Example HAS Client

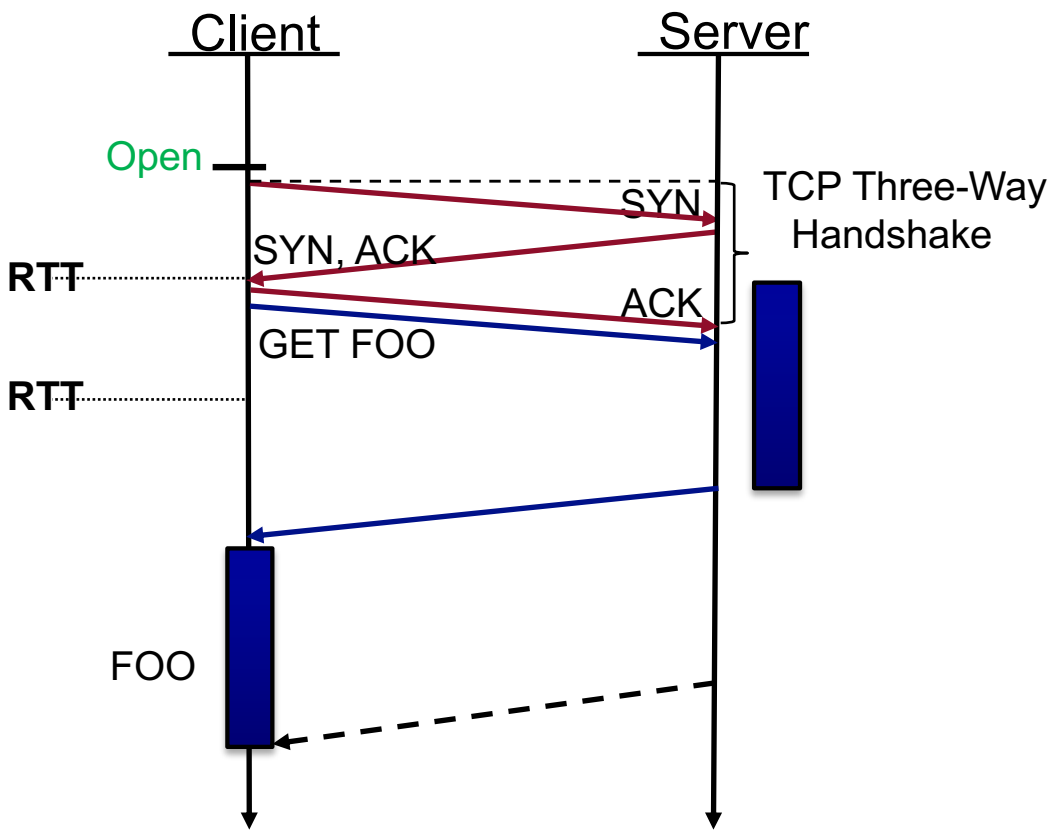


Low-Latency Adaptive HTTP Streaming

Why reduce the latency ?

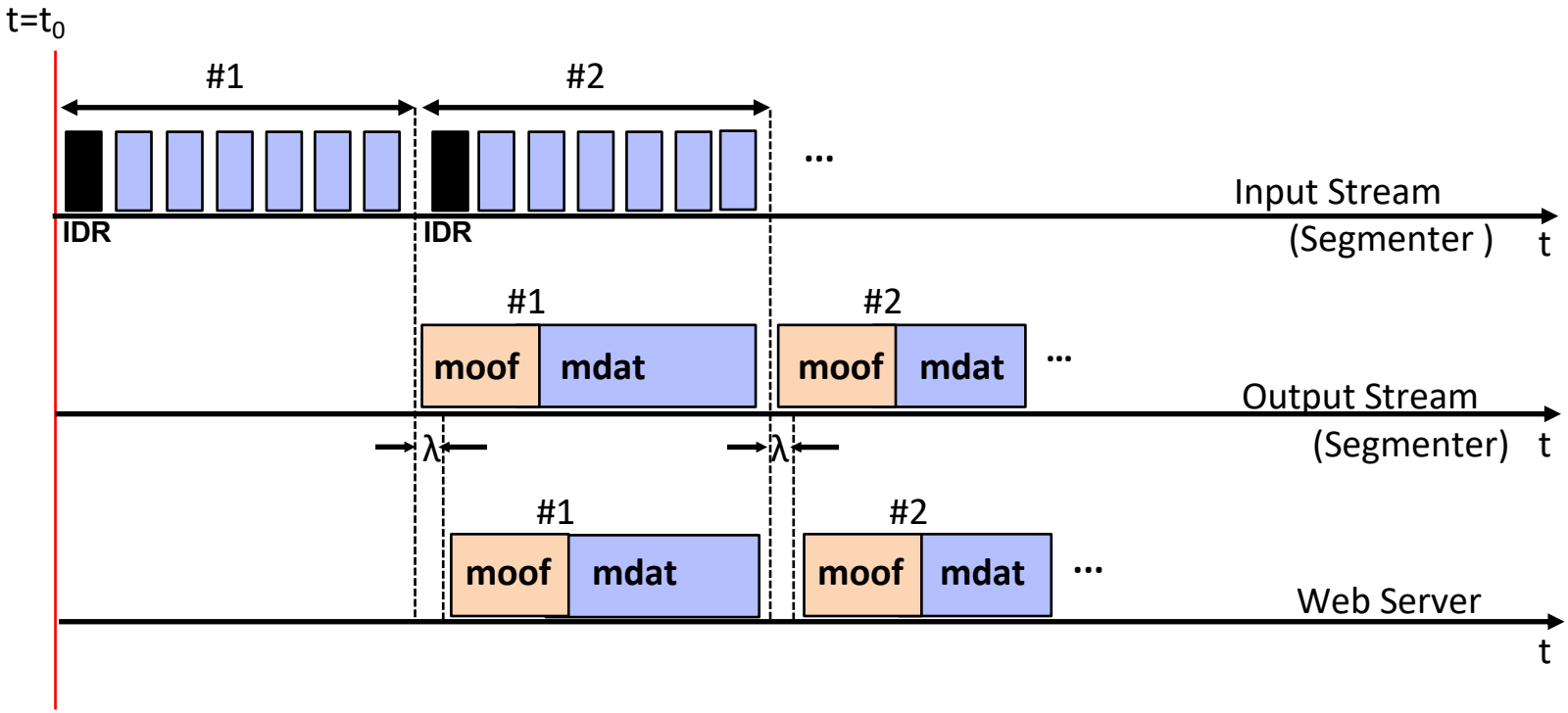
- **Interactives apps**
  - Video conferencing
  - Medical
  - Gaming
- **Mutliple distribution network**
  - Align broadcast and HAS latency
- **Hybrid broadcast / broadband delivery**
  - HAS latency may imply pausing live broadcast

Optimise HTTP 1.1 latency



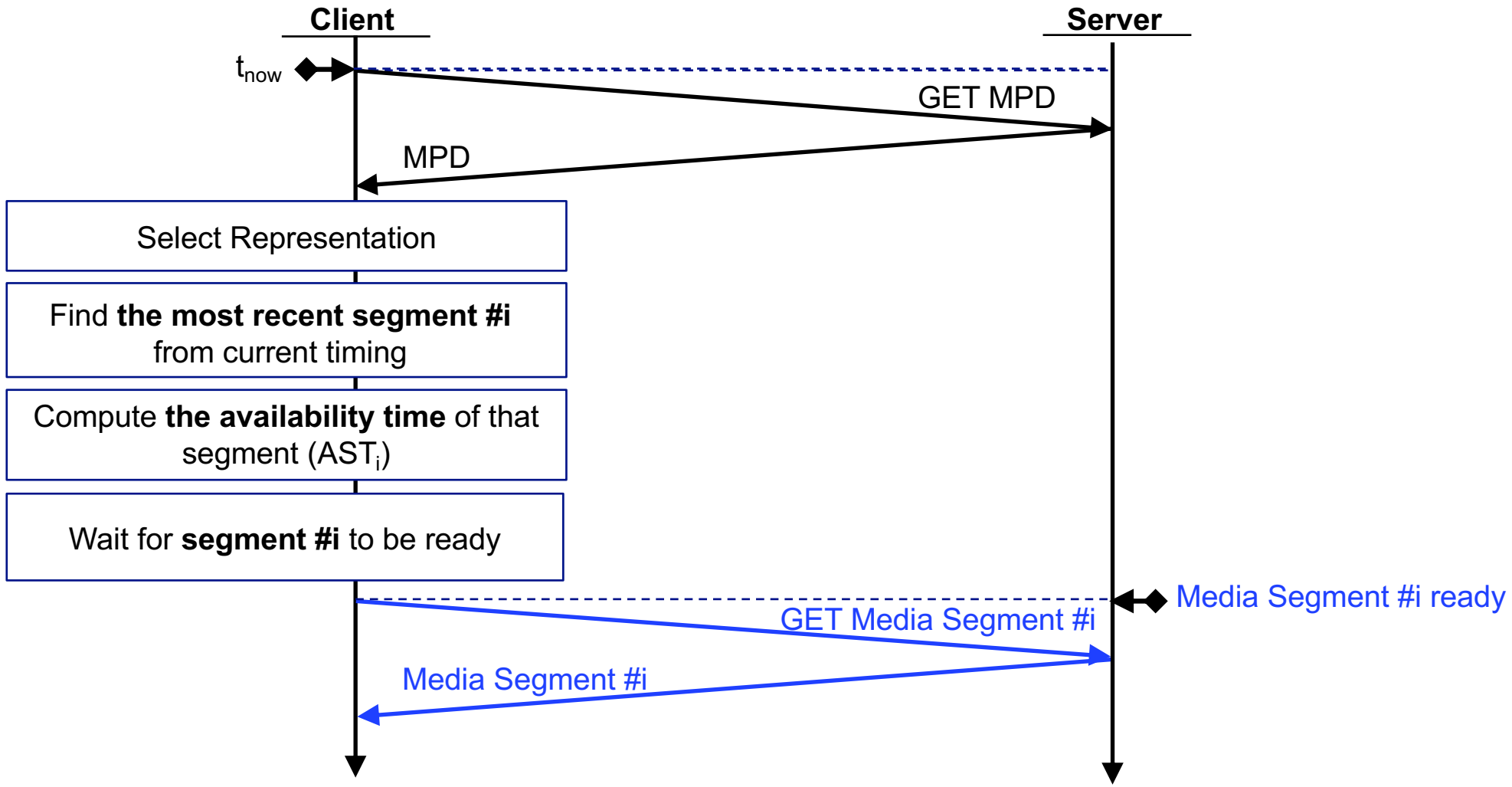
- **Resource size must be known**
  - Wait for resource to be fully available on server
  - Problematic for latency (need to wait segment end)

Live Segmentation Process

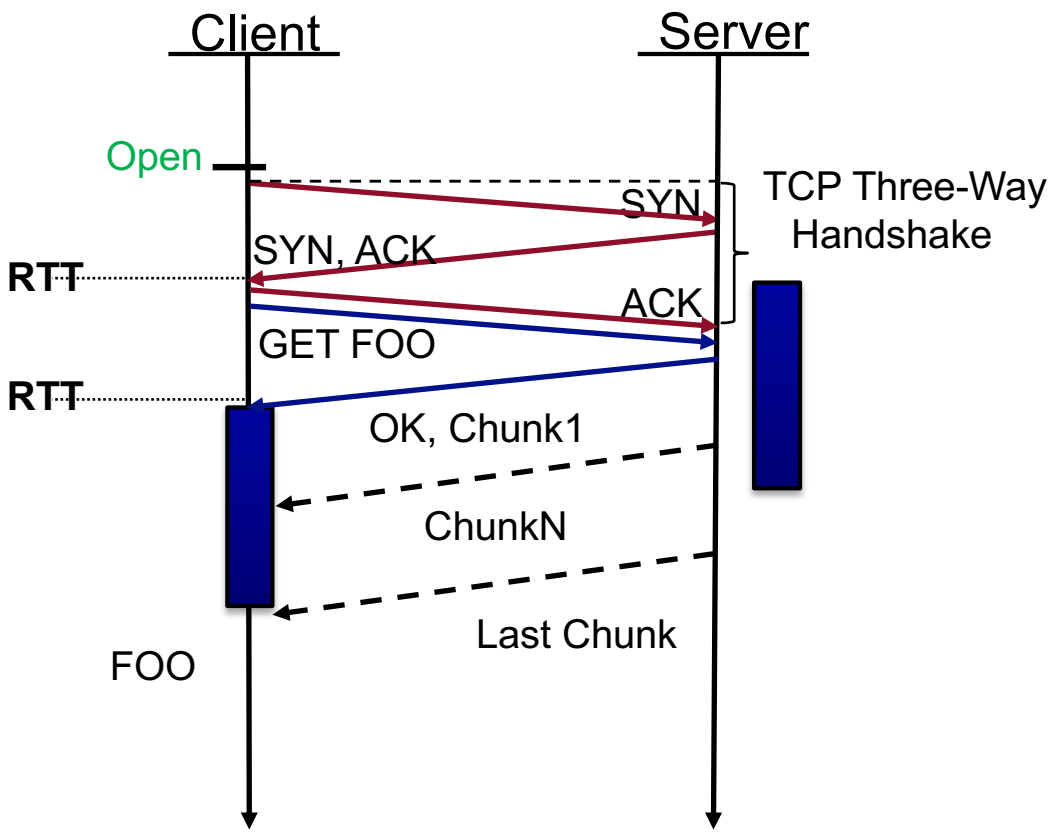


- **Added latency is at least duration of one segment**
- Reducing latency by reducing segment duration:
  - less efficient for compression (more IDR frames)
  - Higher number of requests hence server load

HAS LIVE STREAMING

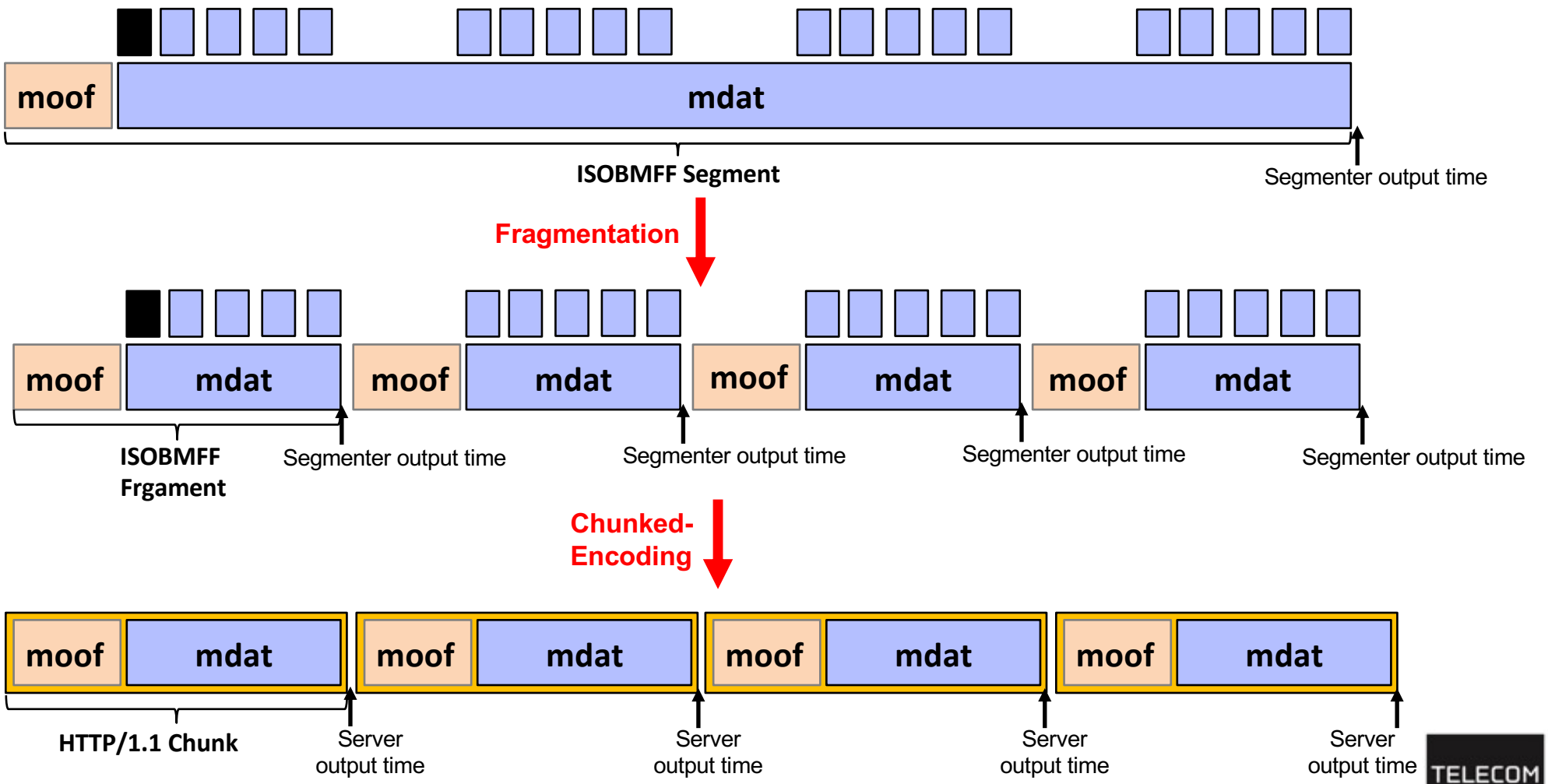


Optimise HTTP 1.1 latency



- **Chunk transfer**
  - Send resource while it is produced/uploaded
  - Allows HAS client to decode a segment while downloading it
  - Not always properly handled by CDNs

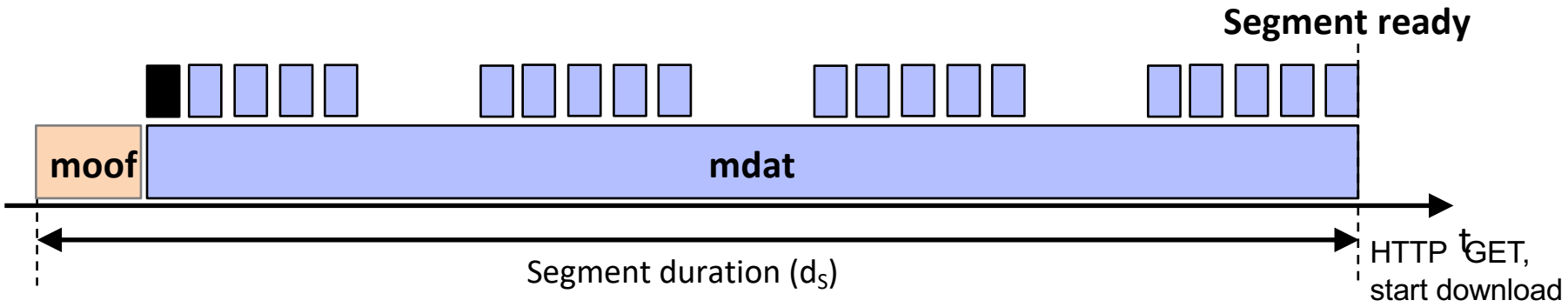
ISOBMFF Fragmentation & HTTP/1.1 Chunk

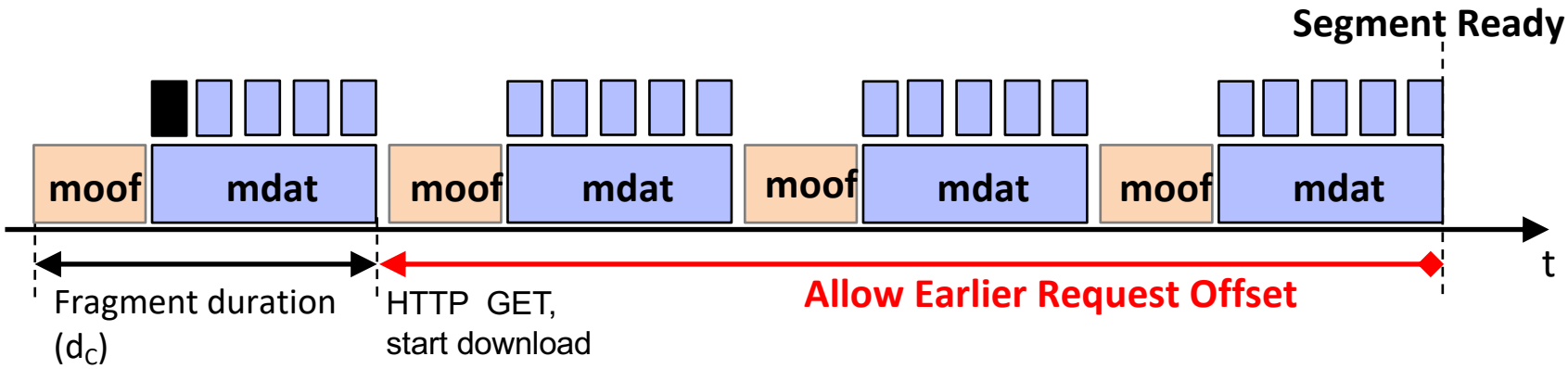


HAS Low Latency

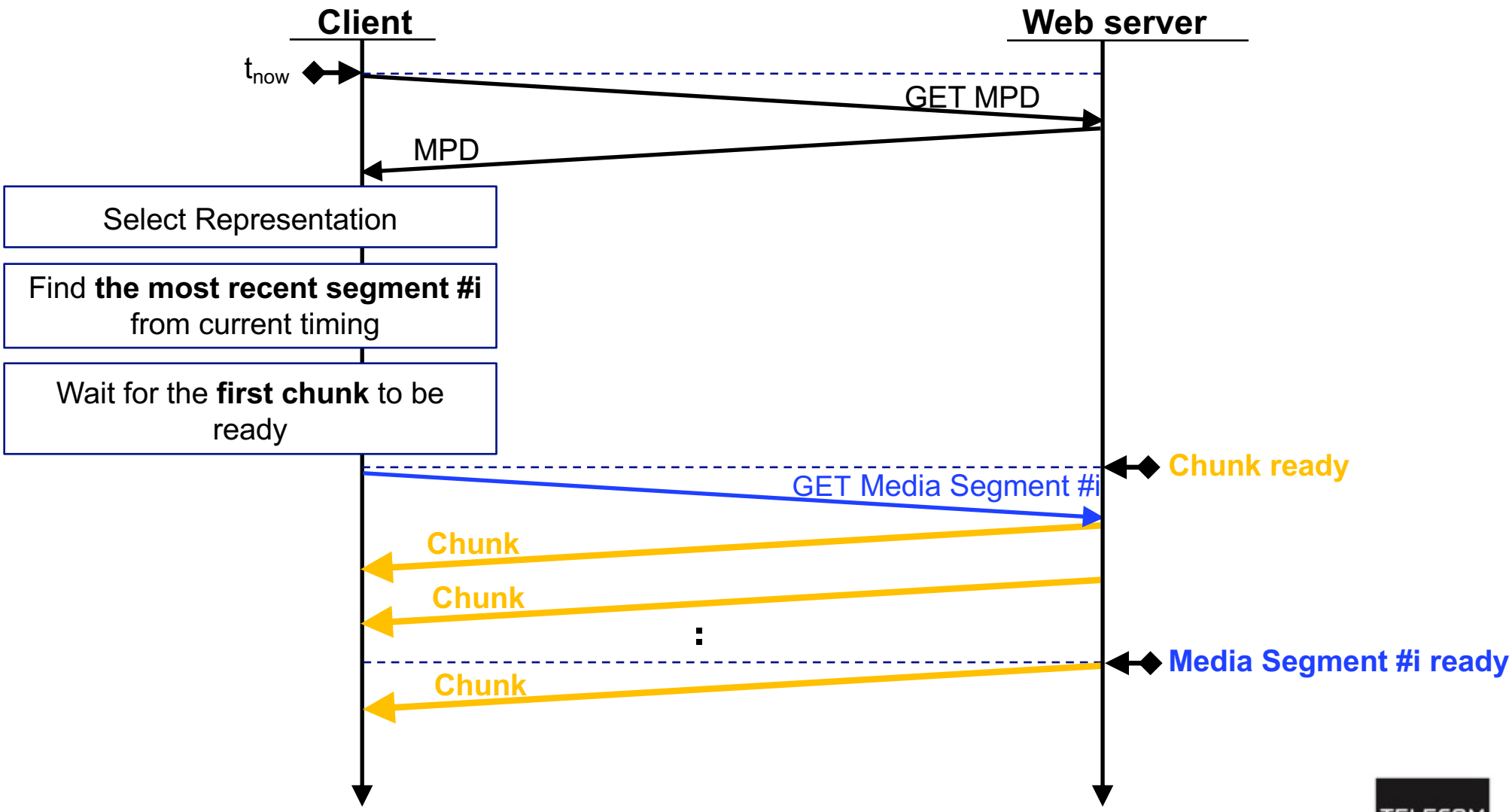
Principle:

- Chunk transfer
- Let the client the possibility to request segments before they are completely produced

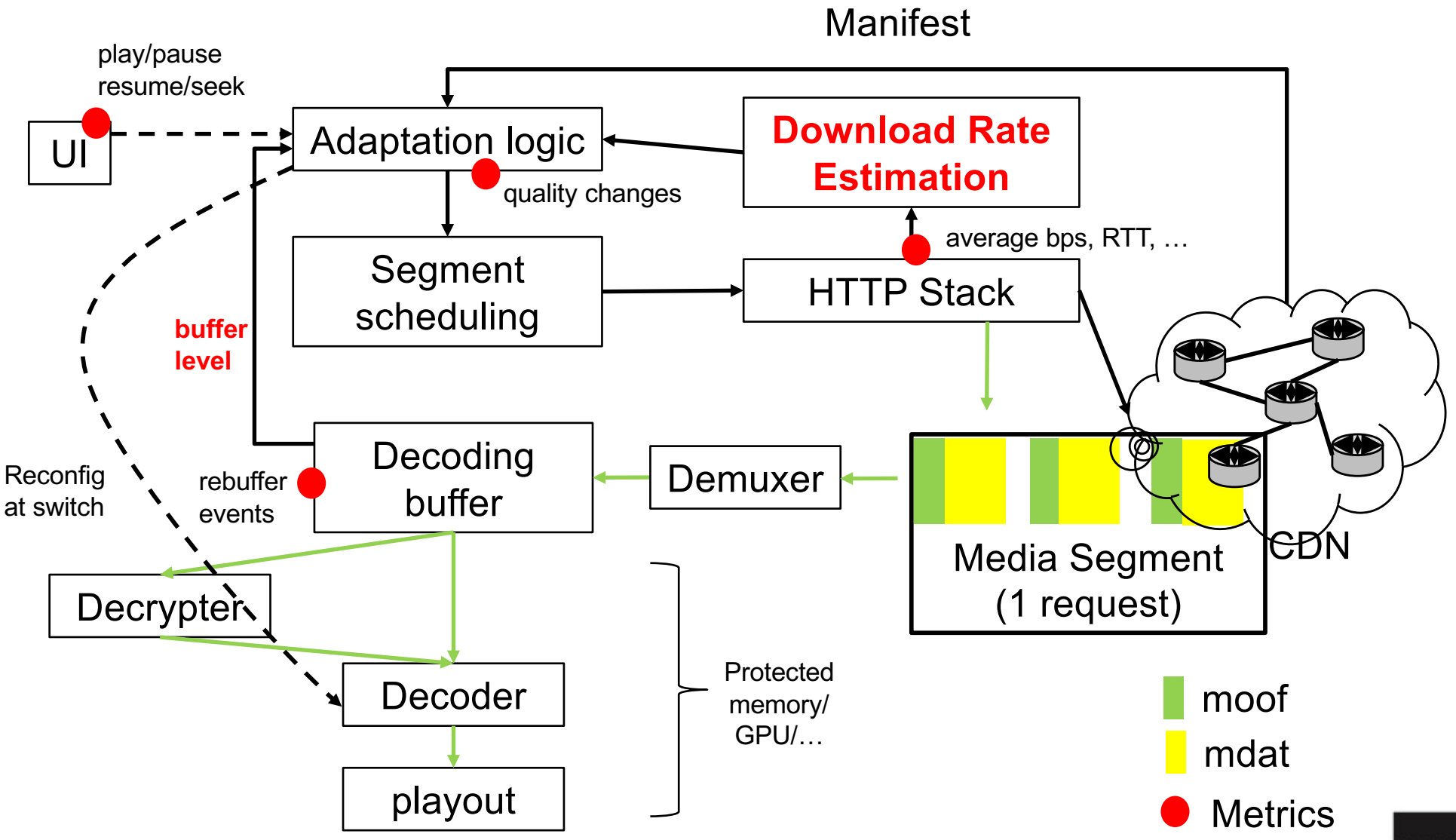




HAS Low Latency



Low Latency Client Architecture



HAS Low Latency Issues

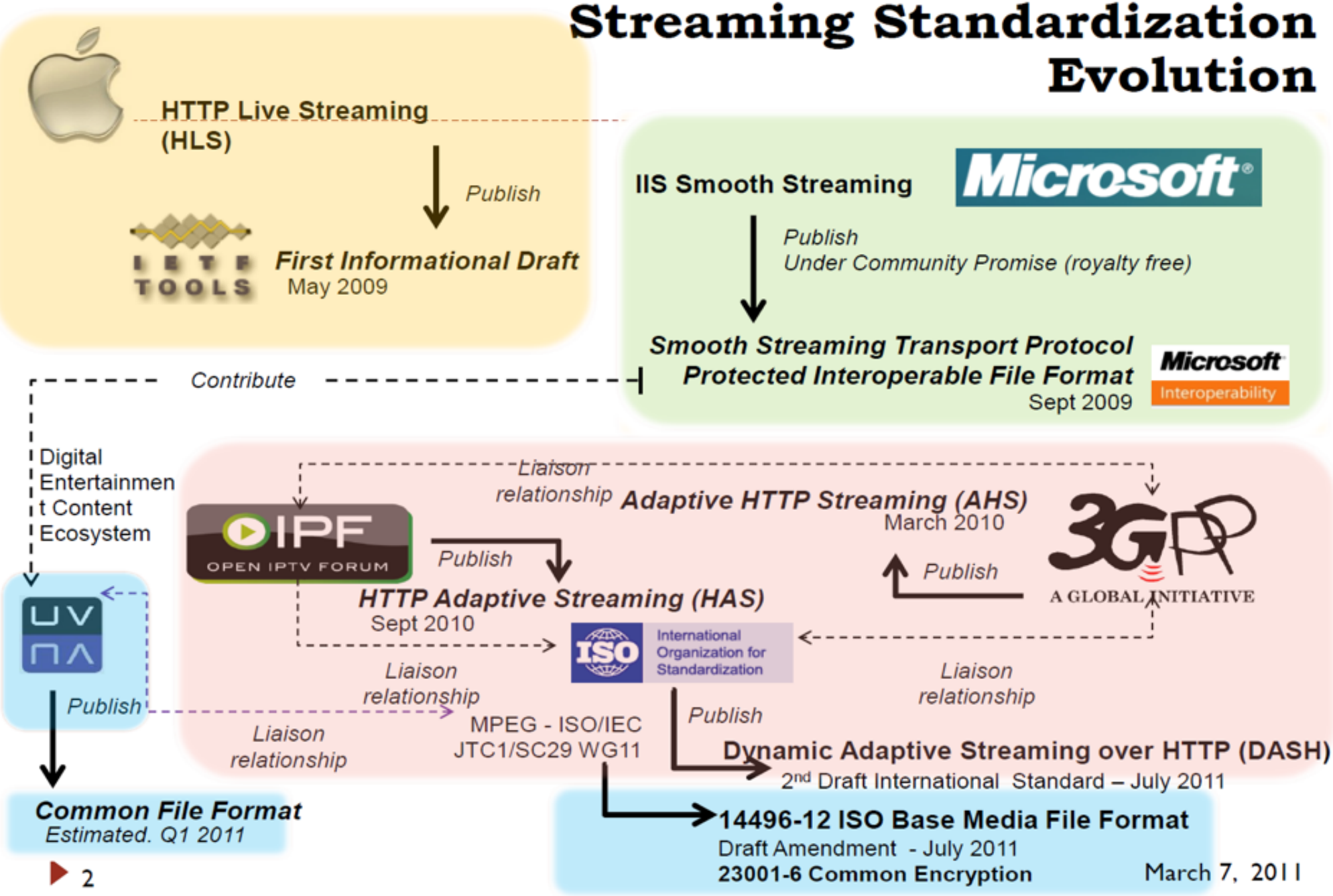
- **UTC/NTP precision of server/client**
- **HTTP chunk-transfert support by CDN**
  - Usually only supported for content upload, not download
- **Adaptation Logic**
  - Buffer-based adaptation no longer possible
    - Low latency implies very small buffer !
  - Available throughput estimation
    - Hard to estimate since real-time delivery
- **Client implementation**
  - HTTP and ISOBMFF stacks no longer separated
  - HTML5-based
    - Impossible using XMLHttpRequest() API
    - Possible through Fetch() API

HTTP Streaming Solutions

Commercial solutions

- 3GPPs' Adaptive HTTP Streaming (AHS)
- Adobes' Dynamic HTTP Streaming (HDS)
- Apples' HTTP Live Streaming (HLS)
- Microsoft’s Smooth Streaming (MSS)
- Open IPVT Forum (OIPF)

Streaming Standardization Evolution



Apple HTTP Live Streaming

- **Solution for Live, VOD & Adaptive streaming over HTTP**
  - RFC 8216 @ IETF
  - Supported on iPhone, iPad and some desktop players
- **Manifest**
  - Based on M3U8 (extended MP3 playlists, not XML)
  - Extended for live streaming, bitrate switching:
    - One master playlist describing available qualities
    - One child playlist per quality describing media segment URLs
      - Before v4: one file per segment
      - After v4: one file per segment or byte-range
  - MIME types:
    - « audio/x-mpegURL »
    - « video/x-mpegURL »
    - « application/x-mpegURL »
  - Live edge: last entry in media playlists

Apple HTTP Live Streaming

- **Media Segments**
  - « chunks »
    - Recommended duration 10 seconds
    - Typically 3 chunks in buffer before playback
  - Before version 7 (2016)
    - AAC+ADTS for audio only
    - MPEG-2 Transport Stream for AAC Audio & AVC Video
    - WebVTT for subtitles
  - After version 7
    - ISOBMFF segments un-multiplexed (single media / file)
- **Protection**
  - AES-128 ***CBC***
- **Client behaviour not normative**

HLS Example

Master.m3u8:

#Q1,BW=1mbps,codecs=…
http://foo.bar/video_low.m3u8

#Q1,BW=5mbps,codecs=…
http://foo.bar/video_high.m3u8

Video_low.m3u8:

#TIME,SEQ_NUM
segment_hlow_10.ts

#TIME,SEQ_NUM
segment_low_11.ts

#TIME,SEQ_NUM
segment_low_12.ts
#EOF

Video_high.m3u8:

#TIME,SEQ_NUM
segment_high_10.ts

#TIME,SEQ_NUM
segment_high_11.ts

#TIME,SEQ_NUM
segment_high_11.ts
#EOF

MS Smooth Streaming

- **Manifest**
  - XML
  - Describe all media (quality/rate/language/…)
  - Template for deriving URLs
    - Ex: `Url="QualityLevels({bitrate})/Fragments(video={start time})"`
  - Decoders configuration
    - base 64 version in the manifest
    - ***No initialization segment (moov) mandatory in ISOBMFF, emulated by client***
- **Live**
  - Edge: current time indicated in manifest
  - URL of segments:
    - tfrf box in segment N gives timing of segment N+1
    - No need to update the manifest

MS Smooth Streaming

- **Media Segments**
  - « fragments »
  - One or more ISOBMFF Fragments (moof)
    - Not ASF ☺
    - AVC|H264 + HeAACv2, VC-1 + WMAPro
    - ***No initialization segment (moov) mandatory in ISOBMFF, emulated by client***
  - A single file per quality
    - Modified HTTP server in charge of converting the time request into a byte range
- **Encryption**
  - AES-128 ***CTR***
- **Ad Insertion**
  - Server-side
- **Client behaviour not normative**
- **Demo**

Example de manifeste SmoothStreaming

```
<?xml version="1.0" encoding="UTF-8"?>
<SmoothStreamingMedia IsLive="TRUE" MajorVersion="2" MinorVersion="2"
DVRWindowLength="17400000000" Duration="0" TimeScale="10000000" LookAheadFragmentCount="2">
 <StreamIndex Chunks="868" Type="video" Url="QualityLevels({bitrate})/Fragments(video={start
time})" QualityLevels="9" Subtype="" Name="video" TimeScale="10000000">
  <QualityLevel Index="4" Bitrate="2962000" FourCC="H264" MaxWidth="1280" MaxHeight="732"
CodecPrivateData="0000000167640020AC2CAC05005DFBFFC10000FBD4808080A0000030020000060C080005A648
0002D325FE31C6040002D32400016992FF18E1DA12253800000000168EA535250" />
  <QualityLevel Index="6" Bitrate="331000" FourCC="H264" MaxWidth="284" MaxHeight="160"
CodecPrivateData="000000016764000DAC2CAC1215EFFFC10000FF9483030320000003002000000060C040028660002
867FF8C718080050CC00050CFFF18E1DA12253800000000168EA535250" />
…
  <c t="602643506043544" d="20000000" r="230" n="25503876" />
  <c d="20158617" /><c d="19841383" /><c d="20158617" />
  <c d="20000000" r="3" /><c d="19841383" />
  <c d="20158617" /><c d="20000000" r="3" />
  <c d="19841383" /><c d="20158617" />
…
 </StreamIndex>
</SmoothStreamingMedia>
```

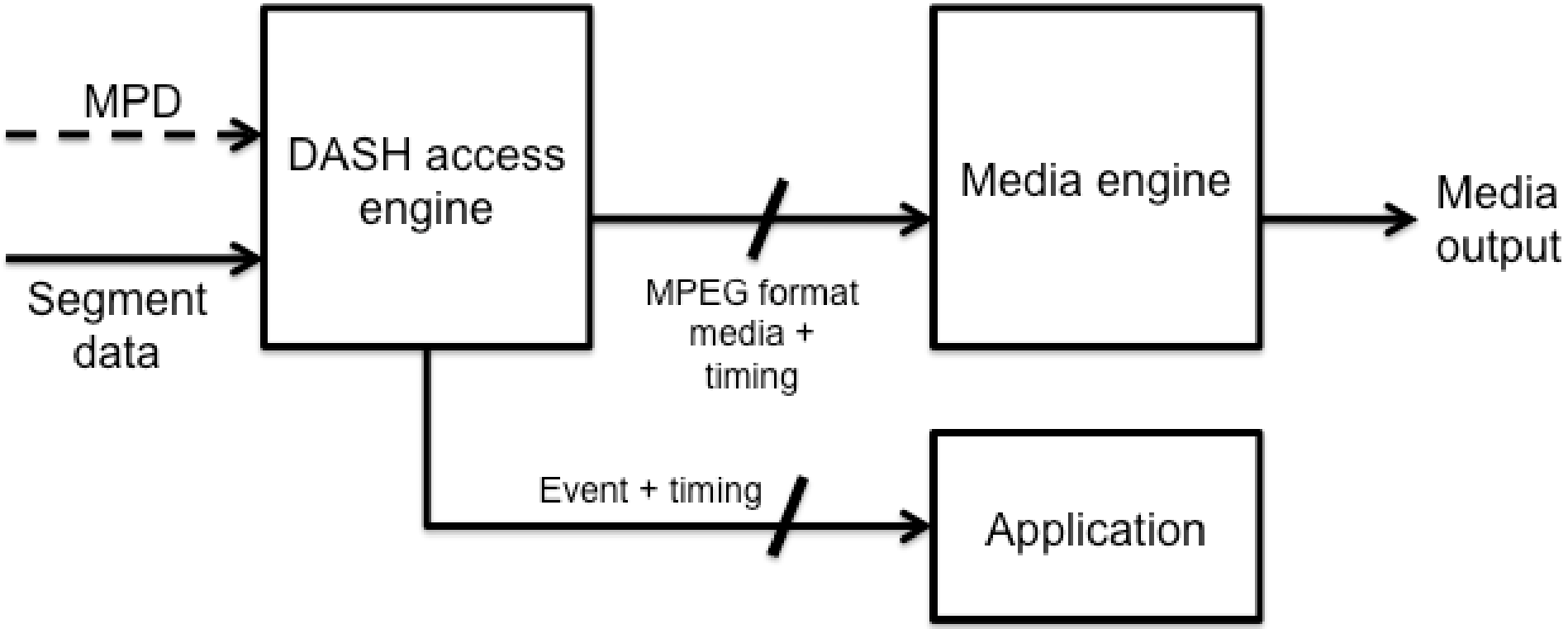
MPEG-DASH

- **ISO/IEC 23009-1**
- **Unification of existing solutions**
  - HLS, Smooth, 3GPP, OIPF
  - Same principles
- **Manifest**
  - XML, extensible via XML NS
  - Describes all media (quality/rate/language/…)
  - Lists of URL or URL construction rules
- **Segments Media**
  - ISOBMFF Fragments or MPEG-2 TS
    - Codecs: anything
    - Extensible for other formats (e.g., mkv)
- **Encryption**
  - AES-128 CBC and AES-128 CTR
- **Ad Insertion**
  - Server-side and client-side
- **Client behaviour not normative**

Manifest MPEG-DASH

```
<MPD type="static" minBufferTime="PT1.5S" mediaPresentationDuration="PT0H10M0.00S »>
 <ProgramInformation moreInformationURL="http://gpac.sourceforge.net">
  <Title>Media Presentation Description for file ZZZ</Title>
 </ProgramInformation>
 <Period start="PT0S" duration="PT0H10M0.00S">
  <AdaptationSet>
   <ContentComponent id="1" contentType="video"/>
   <ContentComponent id="2" contentType="audio" lang="und"/>
   <SegmentTemplate initialization="counter-10mn_I25_openGOP_init.mp4"/>
   <Representation id="1" mimeType="video/mp4" codecs="avc1.64000d,mp4a.40.02" width="320"
height="180" sampleRate="44100" numChannels="1" lang="und" startWithSAP="3"
bandwidth="109952">
    <SegmentTemplate timescale="1000" duration="9880" media="seg40_$Number$.m4s"
startNumber="1"/>
   </Representation>
   <Representation id="2" mimeType="video/mp4" codecs="avc1.64000d,mp4a.40.02" width="320"
height="180" sampleRate="44100" numChannels="1" lang="und" startWithSAP="3"
bandwidth="182078">
    <SegmentTemplate timescale="1000" duration="9880" media="seg112_$Number$.m4s"
startNumber="1"/>
   </Representation>
  </AdaptationSet>
 </Period> </MPD>
```

DASH Client model



MPEG-DASH Specifics

- **Timed division of manifest**
  - Series of « Periods »
  - Period = time interval within which the media configuration does not change
    - TV program
    - ad
    - movie
  - Period elements are ordered in increasing start time.

- **Chaining Periods**
  - seamless
    - splicing, ad-insertion
  - Non-seamless (reinitialization of decoders, DRM, …)
    - Changes in service configuration (codecs, number of streams)
- **Client-side Ad-Insertion**
  - Period pointing to an external document (xlink)
  - The return text is a list of one or more periods for the same duration
  - Ad personalization possible through client profiles

MPEG-DASH VoD

- **Single file containing all segments for a given quality**
  - Single URL in the manifeste
- **Size and timing info of segments**
  - In an index at the begining of the file (« sidx »)
  - Requires analysing the begining of the ISOBMFF file
  - Compact

DASH live

- **One file per segment**
  - Smooth-like operation (one single file per quality) still possible but requires a modified HTTP server
- **URL Template for segments**
  - Ex: `<SegmentTemplate timescale="1000" duration="9880" media="seg40_$Number$.m4s" startNumber="1"/>`
  - Via segment sequence number
  - Via segment timing
    - Possibility to compute next segment timing from current one (cf smooth)
- **Segments duration**
  - Average D with min/max in [-50%,+50%] of D: no specific signaling
  - Otherwise, time/duration per segment indicated in the manifest
    - May require reloading of the manifest

Live edge in DASH live

- **Via UTC clock**
  - Manifest indicates a session start time in UTC
  - Each period indicates its start time in the session
  - Average duration of segments is known

```
UTC(now) = manifest@start + period@start + duration_in_period
```

=>

```
duration_in_period = UTC - manifest@start - period@start
```

=>

```
segment_index = (UTC - manifest@start - period@start) / segment_duration
```

Live edge in DASH live

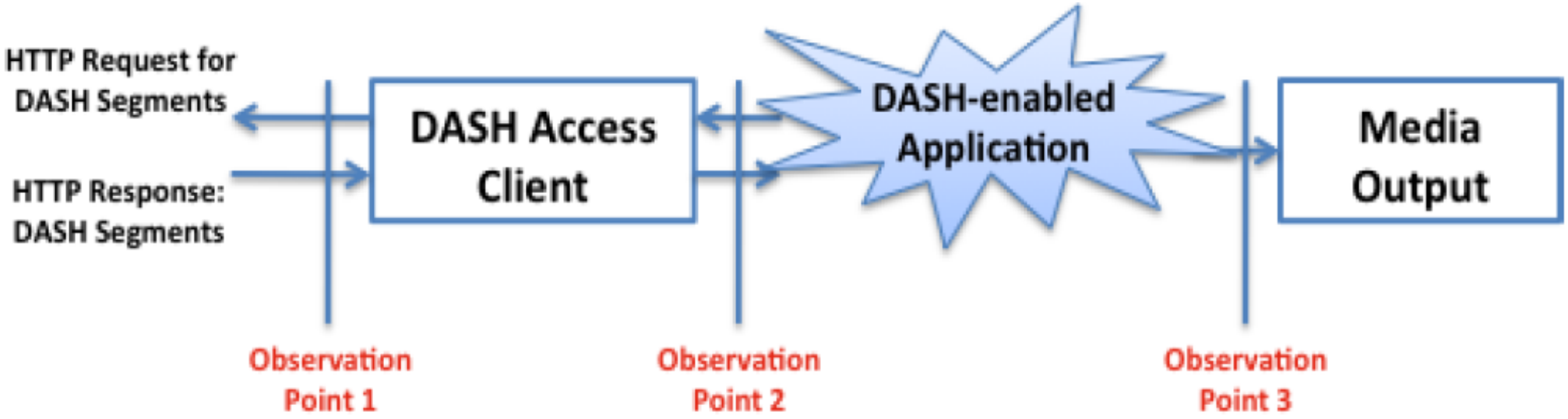
- **Problematic: UTC Time**
  - Likely a drift between client and server
  - Various NTP server could give different NTP timing
- **Solutions**
  - indicate live edge in manifest
    - Drawback: requires frequent update of manifest
    - Ex: seg 1 sec , updated every 10 s => 10s delay!
  - Add address of UTC server (NTP, HTTP) in manifest
    - Drawback: sensitive to RTT

DASH Segments

- **Scalable coding support**
  - Stream dependencies indicated in manifest
  - Order of segment concatenation = dependency order
- **Multiple Sources**
  - Several base URLs can be defined
    - For the entire session
    - For each period
    - For each media object
    - For each quality
  - Allows delivering the same content through multiple CDNs

DASH Quality Metrics

- To report back the Quality of Experience (QoE) to the reporting server.
- 3 conceptual observation points for measurements:



DASH Metrics

- **Collected information**
  - Start and duration of analysis
  - Type of data
  - Indicated in the MPD
- **Observation points**
  - TCP stack
  - Segment assembly
  - Player output
- **Network reports**
  - TCP:
    - IP adresses and interfaces
    - Opening/closing/connect times
  - HTTP
    - URL before and after redirects + byte-range
    - Request/response times + code
    - Download rates at different times
- **Switching reports**
  - Switching times (UTC and media), representations ID
  - Buffer levels (segments)
- **Playback reports**
  - Actions: play/pause/stop/…
  - Times

Questions ?

Jean Le Feuvre
jean.lefeuvre@telecom-paristech.fr

ANNEX - MPEG-DASH

Jean Le Feuvre

jean.lefeuvre@telecom-paristech.fr

MPEG DASH
Dynamic Adaptive Streaming over HTTP

- **ISO/IEC 23009-1**
  - Support for Live, VOD and Adaptive Streaming
  - Different profiles
    - Live
    - On Demand
- **Joint standard development with 3GPP**
  - Important industrial consortium: Qualcomm, Microsoft, Adobe, Ericsson, Apple, …
  - Freely available but not sure about royalty free

(Some) DASH Design Principles

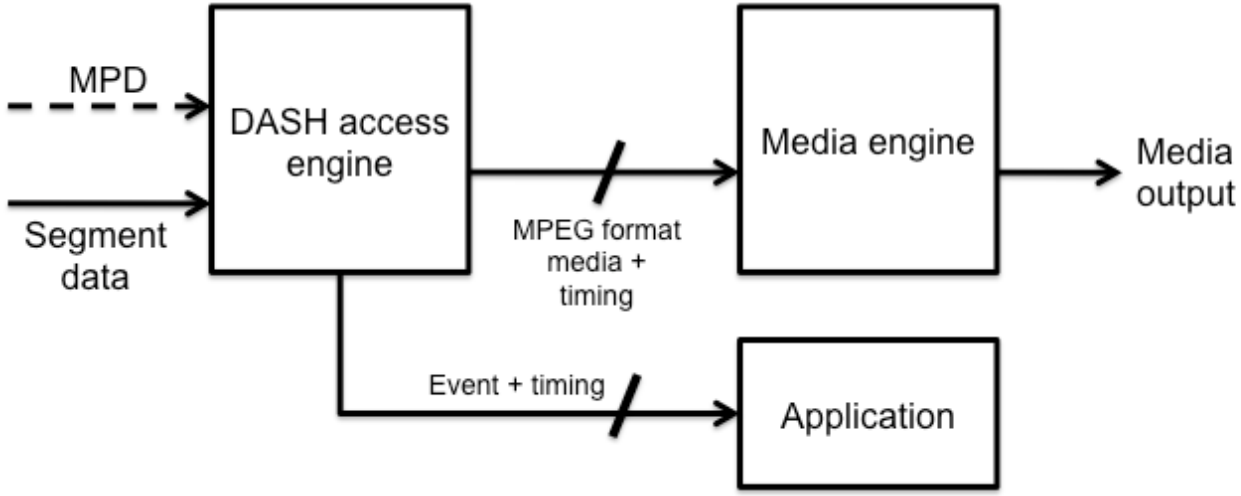
- **DASH is not:**
  - system, protocol, presentation format, codec, client specification
- **DASH is an enabler**
  - It provides formats to enable efficient and high-quality delivery of streaming services over the Internet
  - It is considered as one component in an e2e service
  - System definition left to other organizations (SDOs, Fora, Companies, etc.)

(Some) DASH Design Principles

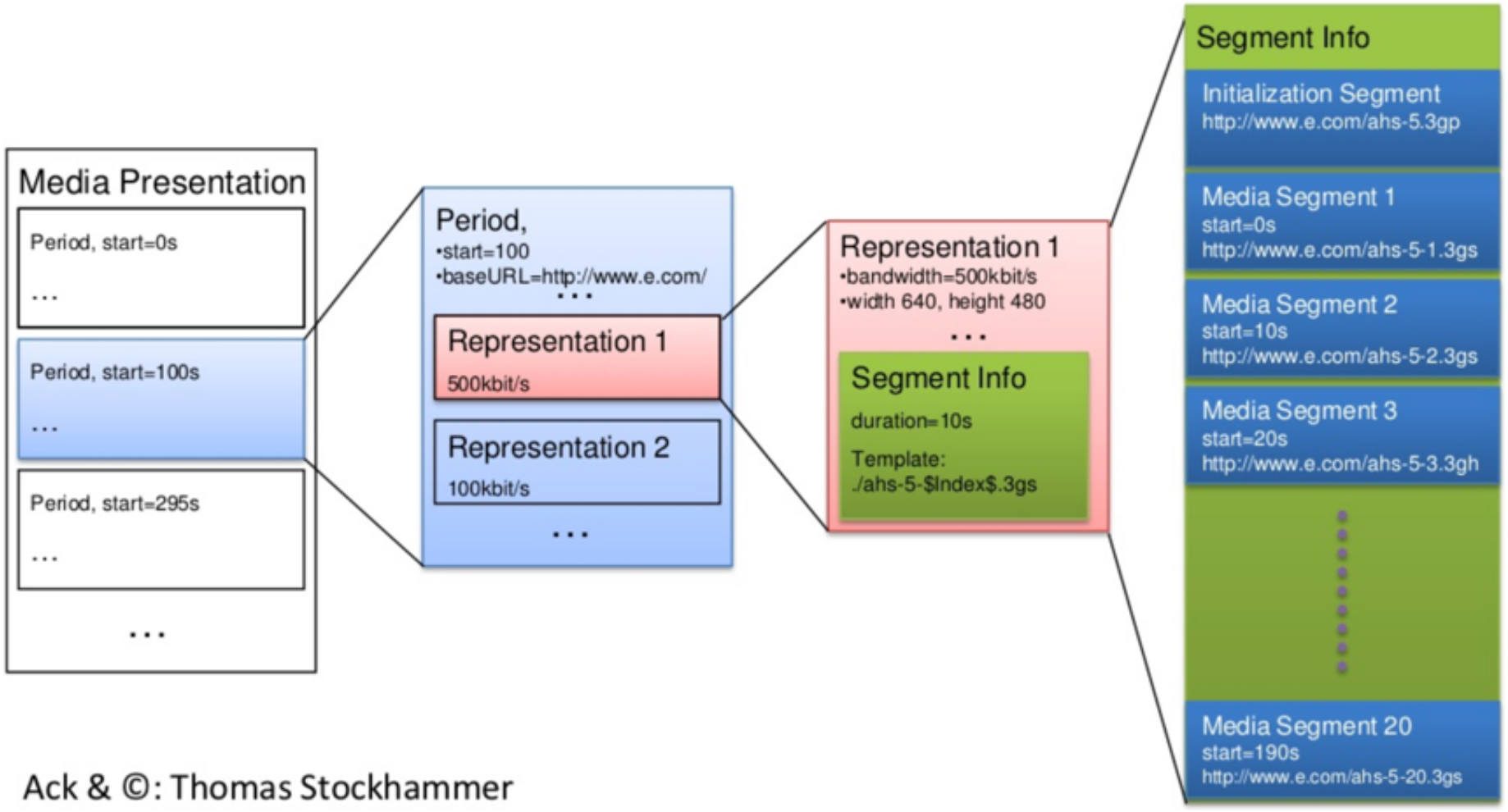
- **It attempts to be very good in what is to be addressed by the standard**
  - Enable reuse of existing technologies (containers, codecs, DRM etc.)
  - Enable deployment on top of HTTP-CDNs (Web Infrastructures, caching)
  - Enable very high user-experience (low start-up, no rebuffering, trick modes)
  - Enable selection based on network and device capability, user preferences
  - Enable seamless switching
  - Enable live and DVD-kind of experiences
  - Move intelligence from network to client, enable client differentiation
  - Enable deployment flexibility (e.g., live, on-demand, time-shift viewing)
  - Provide simple interoperability points (profiles)
  - Enable industry requirements: Ad-Insertion, DRMs

DASH Scope

- **Playlist/manifest format:**
  - XML language
  - MPD: Media Presentation Description
- **Segment formats:**
  - ISO Base media files
  - MPEG-2 TS
  - Extensible to other formats
    - WebM
    - WebVTT
    - ...

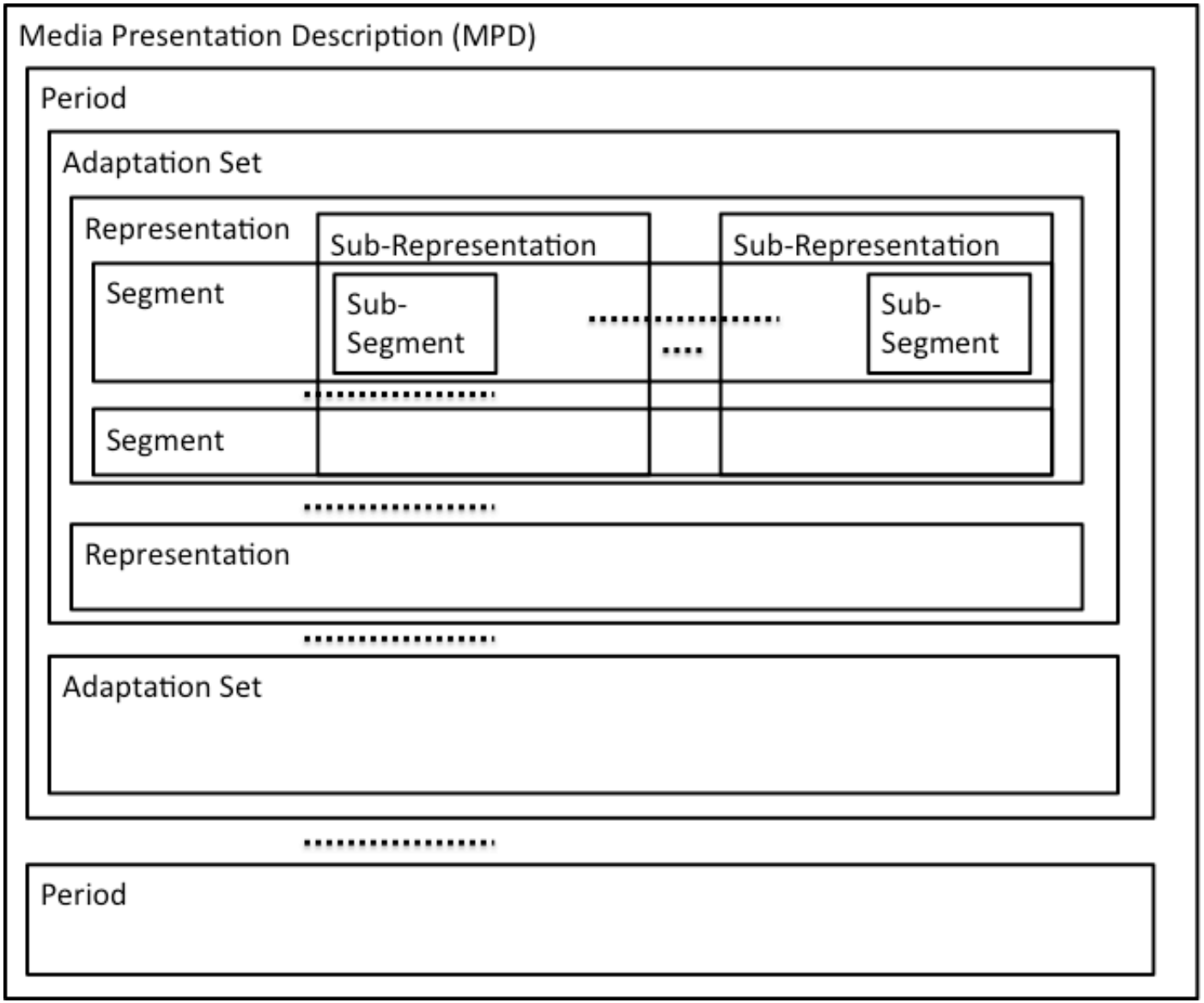


DASH Description Model



Ack & ©: Thomas Stockhammer

DASH Description Model (2)



MPD (Media Presentation Description)

- **XML document**
  - Containing exactly one <MPD> element
    - Containing in turn several <Period> elements
- **Description of the segments**
  - HTTP-URLs absolute or relative, templated or not.
  - The MIME type of the MPD is defined: application/dash+xml
- **Out of scope**
  - Delivery of the MPD
  - Actual playback is not controlled

Example MPEG-DASH Manifest

```
<MPD type="static" minBufferTime="PT1.5S" mediaPresentationDuration="PT0H10M0.00S">
 <ProgramInformation moreInformationURL="http://gpac.sourceforge.net">
  <Title>Media Presentation Description for file ZZZ</Title>
 </ProgramInformation>
 <Period start="PT0S" duration="PT0H10M0.00S">
  <AdaptationSet>
   <ContentComponent id="1" contentType="video"/>
   <ContentComponent id="2" contentType="audio" lang="und"/>
   <SegmentTemplate initialization="counter-10mn_I25_openGOP_init.mp4"/>
   <Representation id="1" mimeType="video/mp4" codecs="avc1.64000d,mp4a.40.02" width="320"
height="180" sampleRate="44100" numChannels="1" lang="und" startWithSAP="3"
bandwidth="109952">
    <SegmentTemplate timescale="1000" duration="9880" media="seg40_$Number$.m4s"
startNumber="1"/>
   </Representation>
   <Representation id="2" mimeType="video/mp4" codecs="avc1.64000d,mp4a.40.02" width="320"
height="180" sampleRate="44100" numChannels="1" lang="und" startWithSAP="3"
bandwidth="182078">
    <SegmentTemplate timescale="1000" duration="9880" media="seg112_$Number$.m4s"
startNumber="1"/>
   </Representation>
  </AdaptationSet>
 </Period> </MPD>
```

Period

- **Media Presentation = one or more Periods.**
  - Period = time interval within which the media configuration does not change
    - TV program
    - ad
    - movie
  - Period elements are ordered in increasing start time.

- **Main attributes**
  - @start (optional)
  - @duration (optional)

- **Main Content**
  - <AdaptationSet> elements

Adaptation Sets

- **1 Period = one or more AdaptationSets**
- **1 AdaptationSet = N Representations**
  - Alternative but “equivalent” media, typically contain different encoded versions of the same source
  - Language, media component type, picture aspect ratio, role, accessibility, viewpoint, rating.
- **Adaptationsets can be arranged using @group:**
  - Either one Representation from group 0, or the combination of at most one Representation from each non-zero group.

Representations

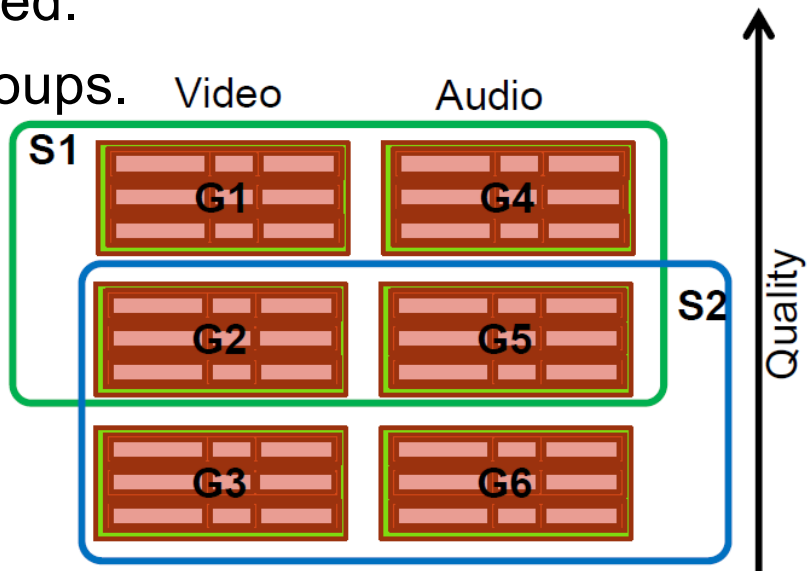
- **Media content**
  - Aligned within the period's boundaries.
  - Consists of one or more Segments.
  - Contains an initialization segment or all segments are self-initializing.
  - May contain zero or more SubRepresentations.
- **Multiplexed Elementary Streams**
  - In one AdaptationSet
  - Different components identified using <ContentComponent>
  - Storage cost (e.g. duplicated audio streams)
- **Individual Elementary Stream**
  - One AdaptationSet par ES
  - Possibly different Segment durations
- **Type identification by @mime & @codecs**

Sub Representations

- **Provide the ability for accessing a lower quality version of the Representation.**
  - Audio track in a multiplexed Representation.
  - Lower frame rate for efficient fast-forward.

Subsets

- **Optional mechanism for the content creator to create collections of media presentations for various application/devices within the same MPD.**
  - Restrict the combination of active AdaptationSets.
  - DASH client must use one subset.
  - Empty subsets are not allowed.
  - No subset can contain all groups.



Segments

- **Segment = part of a bitstream**
  - Exact definition depending on the bitstream format (TS, ISOBMF, …)
  - Sub-segment = part of a segment
  - Several types of segments are defined
- **A Segment is a unit that can be referenced by an HTTP-URL included in the MPD.**
  - “http://” and optionally with a byte range.
- **Segments have an availability window**
  - Can be accessed by the HTTP-URL.
- **Each representation has at most one SegmentInfo element which provides:**
  - Presence or absence of Initialization and Index Segment information.
  - HTTP-URL and byte range for each segment.
  - Segment availability start time and availability end time for live case.
  - Approximated media start time and duration of each segment.
    - Fixed or variable duration.

Initializations Segments

- Each representation may have at most one Initialization Segment to initialize the media engines for play-out.
- If no Initialization Segment URL is present, then each Media Segment is self-initializing.

Media Segments

- **Each representation has a list of consecutive Media Segments**
- **Media segment information:**
  - URL, possibly restricted by a byte range.
  - Index of segment.
  - Approximate start time and duration.

Bitstream Switching Segment

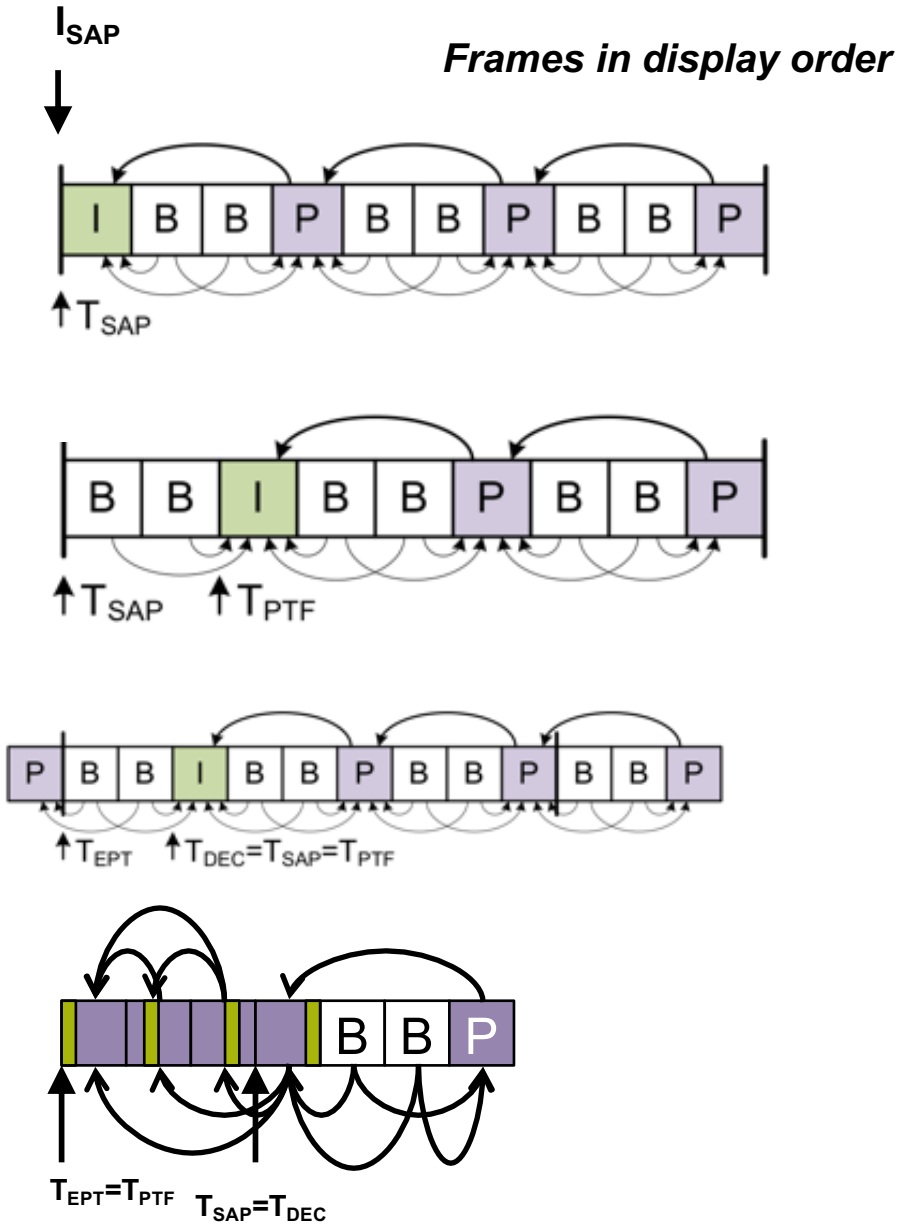
- **Additional data needed for switching**
  - Insures that concatenation of segments coming from different representations is correct
  - Example: DVB-CSA information at switch: system info (CAT) and key info (ECM/EMM)

DASH: Random Access Points

- **Historical, simple notion**
  - I / IDR frame
- **Does not cover more complex cases**
  - Open GOP: Frames from GOP N+1 use Frames from GOP N as references
  - Gradual Decoding Refresh: No « I » or « IDR » frames but progressive reconstruction over N>1 frames
- **SAP: Stream Access Point**
  - I_{SAP}: Point/position in the bitstream from which the decoding can start with data before I_{SAP}
  - T_{SAP}: Presentation time from which the decoding can happen without data prior to I_{SAP}
  - T_{PTF}: Smallest presentation time for all frames after I_{SAP}
  - T_{EPT}: Presentation time of the frame at the I_{SAP} position

SAP Types

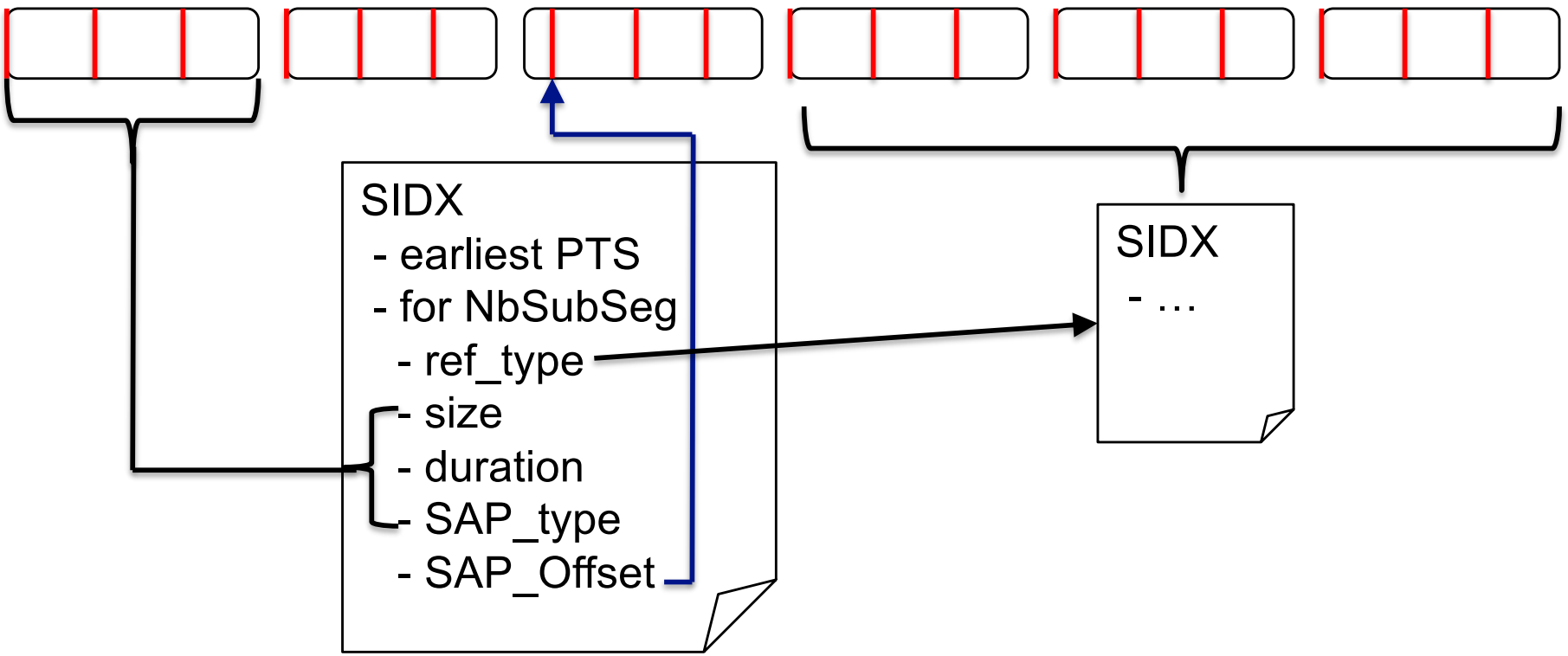
- 1: Closed GOP
- 2: Closed GOP
- 3: Open GOP
- 4: GDR



DASH: Segment Indexing

- **Allows random access into a media segment without downloading the segment**
  - Very hard to do with MPEG-2 TS
  - Hard to do in ISOBMF
- **New indexing tool, the « sidx » box**
  - A table per segment
  - Describes N entries (aka subsegments)
    - N is configurable by the encoder/packager/indexer
  - Each entry has:
    - Duration and size
    - SAP Type
    - Time difference between the subsegment start and the SAP
  - Indexing can be flat, hierarchical, or chained
- **« sidx » can be**
    - In the media segment (ISOBMF): “Indexed Segment”
    - Outside of the segment (TS or ISOBMF): “Indexing Segment”
    - Fetched via HTTP GET (w/ or w/o byte-range)

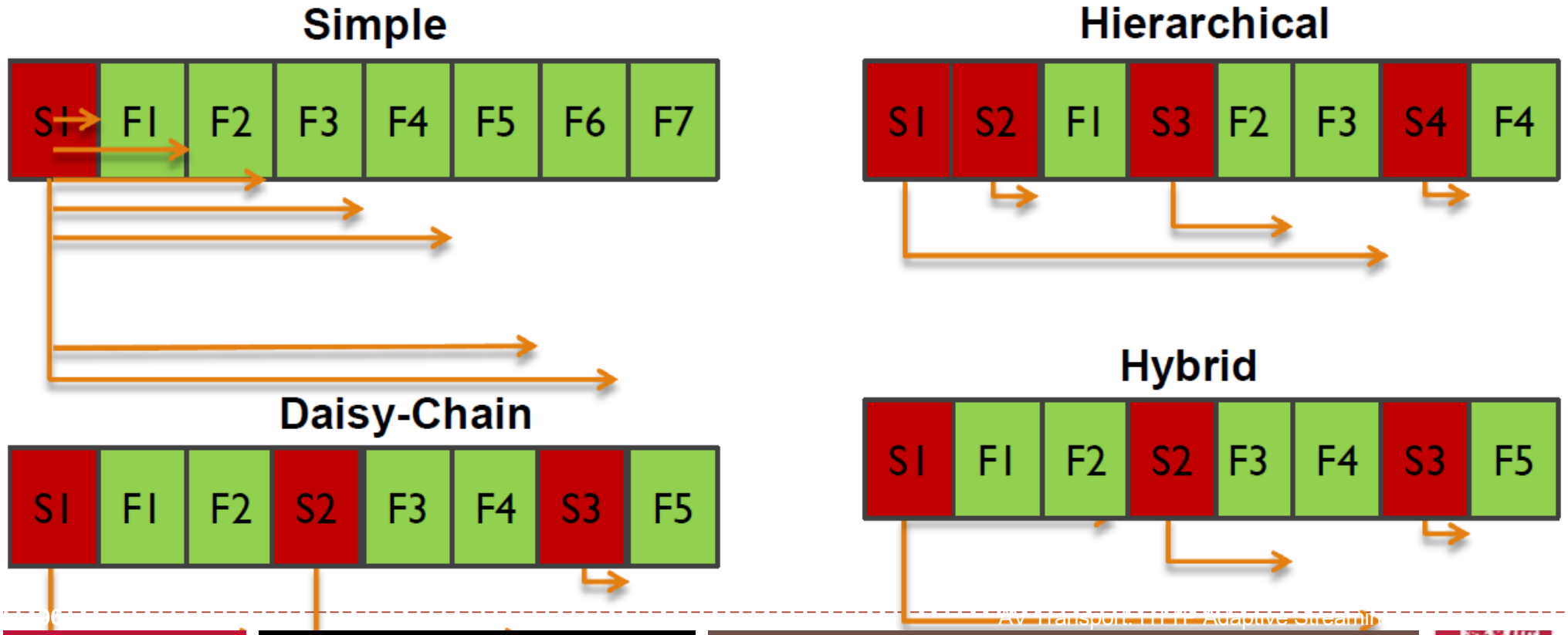
SIDX example for a segment



- **SIDX describes the whole segment (no holes)**

SIDX (Segment InDeXing)

- **Describes sub-segments and Stream Access Points (SAP)s in the segment.**
  - Byte offset and duration.
  - access the sub-segments by the use of HTTP partial



Summary of ISOBMFF Segments

- **Initialization Segment**
  - "ftyp" box
  - "moov" box with empty sample tables
  - Movie fragment defaults
  - Decoder configurations (for all bitrates!)
- **Media Segment (or subsegment)**
  - « styp » Box
  - 1 or more movie fragments (moof+mdat)
    - Media data in the segment file
    - Position (byte offset) of AU relative to the « moof » start
  - A segment contains a least a Random Access Point
  - Decoding time of the first sample of each track is given with the tfdt box for each fragment
    - DTS are stored as delta not as absolute time in mp4
- **Indexed Segment**
  - Segment with « sidx » before the first fragment
    - May have other « sidx » boxes in the segment
- **Bitstream Swiching Segment**
  - Not used

Summary of MPEG-TS Segments

- **Segment (or subsegment)**
  - Sequence of TS packets (188 bytes)
  - One program only
  - An integer number of video frames
    - Not necessarily integer number of audio frames
- **Initialization Segment (optional)**
  - PAT, PMT & PSI constant over the stream
  - ECM if constant
  - Optional
    - If not present, PAT/PMT/ECM at the beginning of each segment
- **Bitstream Switching Segment (optional)**
  - Used to make concatenation of segments conformant with TS (PCR, buffer model …)
- **Indexing Segment**
  - ISOBMF Structure not stored in the TS
    - styp, sidx
    - pcrb: gives the PCR of the first byte of each segment

DASH: Types of URLs

- **String representing an HTTP URL**
  - *SegmentBase*, *SegmentList*
- **URL Template**
  - Enables constructing a URL from some segment parameters
    - $RepresentationID$: ID of the representation
    - $Number$: segment number
    - $Bandwidth$: representation bandwidth
    - $Time$: time of the segment start

```
<SegmentTemplate startNumber="1"
media="mp4-live-$RepresentationID$-$Number$.m4s"
initialization="mp4-live-$RepresentationID$-.mp4"/>
```

URL Addressing

- **URLs at each level of the MPD are resolved with respect to the BaseURL elements of levels above.**
- **The base URL information may present on the following levels:**
  - MPD, Period, AdaptationSet, Representation
- **Alternative base URLs may be provided through the BaseURL element**
  - the identical segments are accessible at multiple locations.

DASH: Timelines (1/3)

- **MPD presentation timeline (T_{MPD})**
  - T_{MPD} = 0 at an arbitrary moment, typically when the first period starts (if period@start=0)
- **Period timeline (T_{Period})**
  - Time relative to the beginning of the period (T_{Period}=0 at the beginning of each period)
- **Mapping of Period time to MPD time:**
  - Linear mapping: $T_{Period}=0 \Leftrightarrow T_{MPD} = period@start$
    - period@start : time in the MPD time line from which the media in the period start to be processed
- **Media timeline (T_{media})**
  - T_{media} = 0 when the media presentation starts (typically when the first frame is presented)
    - For ISOBMFF, movie timeline (i.e. with edit list)
    - For TS, PTS
- **Mapping of Media Time to Period Time**
  - Linear mapping: $T_{Period} = T_{Media} – PTO$
    - PTO = Representation@presentationTimeOffset
  - Consequences:
    - If T_{period}(FirstAU) < 0 (i.e. T_{Media}(FirstAU) < PTO), the media is not presented (may need to be decoded)
    - If T_{period}(FirstAU) > 0, may have a gap in the presentation

DASH: Timelines (2/3) – Segment Times

MPD Start Time of a Segment:

- Reference Time in the Period Time Line for a segment
  - Used in live scenarios
- First segment in the period:
  - $T_{Period}(segment(0)) = 0$
- Segment K in the period:
  - $T_{Period}(segment(k)) = Sum(segmentDuration(j), 0, k-1)$

Drift Constraint

- $ABS(MPDStartTime(k) - T_{period}(FirstAU(k)) < 50\%*segmentDuration$

DASH: Timelines (3/3) – Live case

Use of UTC Time line

- *mpd@availabilityStartTime (AST):*
  UTC time used to determine the availability of each segment
  - To issue HTTP requests
  - Synchronization server/client

Live edge

- If T_{UTC} = *now*, find segment k such that ?

```
now >= AST + period@start + MPDStartTime(k)
```

- The segment will be available at :

```
T_UTC = AST + period@start  + MPDStartTime(k) + duration(k)
```

- Adjust k depending on buffering constraints

DASH: SegmentTimeline

- **Goal**
  - Segments with variable duration
    - Drift in segment availability
  - Gaps in the content generation (black-out)
- **Principle**
  - Explicit signaling of each MPD StartTime
    - Run-length encoding of segments with the same duration
  - List of segments, for each segment
    - @t: MPD StartTime(k)
    - @d: segment duration
      - <0: duration not yet known
    - @r: number of consecutive segments with the same duration
      - <0: until the next element or until the end of the period

Which Segment addressing to choose?

- **List (playlist) : No**
  - `<SegmentURL media="seg2.m4s"/>`
- **Base (byte offsets): perfect for VoD**
  - `<SegmentBase indexRangeExact="true" indexRange="867-1618"/>`
- **Number: ok**
  - ```
    <SegmentTemplate timescale="25000"
     media="seg_$Number$.m4s" startNumber="1"
     duration="250000" initialization="seg_init.mp4"/>
    ```
- **Timeline: MSS compatible**
  - ```
    <SegmentTimeline>
      <S t="0" d="10000" r="59"/>
    </SegmentTimeline>
    ```

DASH: Live Issues

- **Problem: UTC timing**
  - NTP servers can provide different UTC values !!
  - Possible drifts between clients and servers
- **Solution 1 – Indicate the live-edge segment**
  - Using SegmentTimeline
  - Live edge = last entry in the SegmentTimeline
  - Problem:
    - Requires frequent MPD updates
    - Or requires client-guessing of segments not advertised
- **Solution 2 – DASH 2015**
  - Addition of <UTCTiming> to use adjust the provide UTC information
    - NTP, SNTP servers
    - HTTP servers:
      - HTTP HEAD/GET to get
        - the « Date » header
        - An xs:dateTime value in the body
        - An ISO 8601 value in the body
        - An RFC 5905 value in the body
    - A date value in the MPD!

DASH: Switching

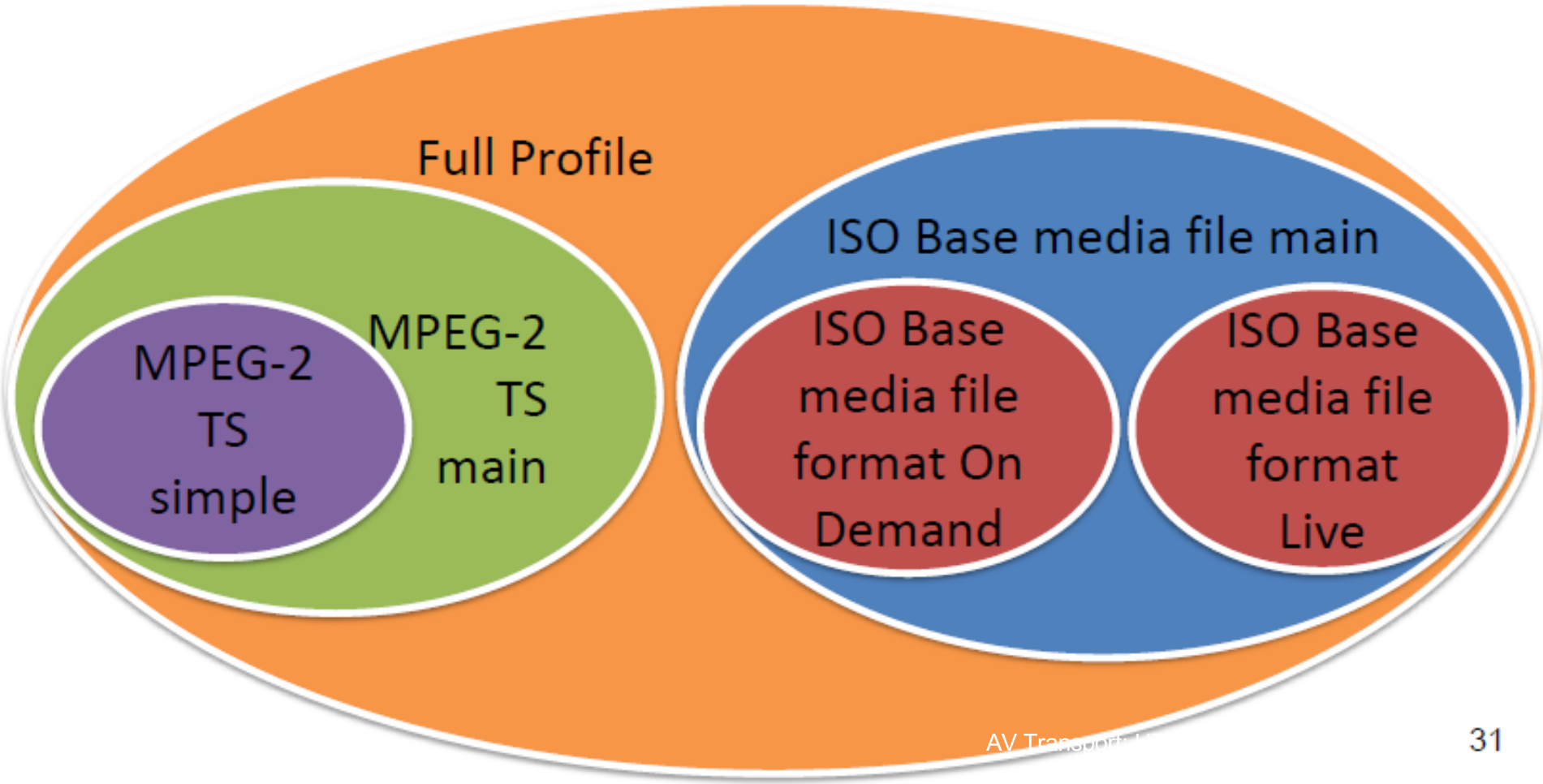
- **Based on the presentation times of 2 representations**
  - i.e. using the MPD Media Time, i.e. ISOBMFF Movie Time
  - Not the decoding times (I/P/B <-> I/P)
- **Requires determining**
  - Random Access Points
  - Presentation times of those RAPS
- **@bitstreamSwitching**
  - Indicates if the concatenation of media segments is valid
  - Otherwise, need to use an initialization segment or switching segment

DASH Descriptors

- **Basic Descriptor**
  - Common syntax: @schemeIdUri, @value, @id
  - Common behavior:
    - If 2 descriptors are present at the same level, with the same @id, only one of the two needs to processed
- **Derived descriptors**
  - Generic descriptors
    - EssentialProperty: should not process if not understood
    - Supplemental Property: additional information
  - Role, Accessibility
    - Define media role (main, caption, subtitle, alternate …)
    - Mapping to HTML 5
  - Media-Specific descriptors
    - Viewpoint, Frame-Packing, Audio Config
  - Others
    - Content Protection
    - Asset Identifier

Profiles

- Set of restrictions on the offered Media Presentation (MPD & Segments)
- can also be understood as permission for DASH clients that only implement the features required by the profile to process the Media Presentation
- Profiles defined in ISO/IEC 23009 (as below). More restrictions may be added



DASH ISOBMFF onDemand Profile

- **Netflix profile**
- **Only one segment per representation**
  - Temporal alignment of sub-segments
  - Sub-segments start with SAP Types 1, 2 or 3
- **Mandatory Indexing using « sidx »**
  - All « sidx » boxes before the first « moof »
- **« Self-initializing » Segment**
  - Segment contains « moov » box:
    - Segment = styp, moov, sidx, moof,mdat … moof, mdat
- **Cannot be used for live streaming**

DASH ISOBMFF Live Profile

- **N segments per representation**
  - Temporal alignment of segments
  - Segments can start with SAP Type 1, 2 or 3
- **Optional Indexing**
  - All « sidx » boxes before the first « moof »
- **Segment addressing**
  - Only SegmentTemplate
    - Possibly with SegmentTimeline
- **Useful for Live or VoD scenarios**
  - ***Used by HBBTV 1.5!***

DASH ISOBMFF Main Profile

- **N segments per representation**
  - Temporal alignment of segments
  - Segments can start with SAP Type 1, 2 or 3
- **Optional Indexing**
  - All « sidx » boxes before the first « moof »
- **Segment addressing**
  - SegmentTemplate or SegmentList
    - Possibly with SegmentTimeline
  - Byte-Range requests allowed
- **Useful for Live or VoD scenarios**

ISOBMFF Profiles

MPD item	On Demand	Live	Main
MPD@type	Static	Dynamic or Static	Dynamic or Static
Segmentation	Single	Single or multiple	Single or multiple
Alignment	Yes (subsegment)	Yes (segment)	Static: Any Dynamic: Yes (segment)
StartWithSAP (1, 2, 3*)	Yes (subsegment)	Yes (segment)	Static: 1,2,3 Dynamic: Yes (segment)
StartWithSAP (>3)	No		No
Segment Timeline	No	Yes	Yes
Subsets	May be ignored		
Multiple Periods	Yes		
Multiplexed	Yes		
Non-multiplexed	Yes		

DASH MPEG-2 TS Profiles

- **Main Profile**
  - N segments per representation
    - Temporal alignment of segments
    - Segments can start with SAP Type 1, 2 or 3
  - Optional Indexing

- **Simple Profile**
  - Same PSI in all representations
  - Optional Indexing
  - If CAS, ECM valid for the whole duration of a subsegment or segment if no sidx

MPEG-2 Profiles

MPD item	M2TS Simple	M2TS Main
MPD@type	Dynamic or Static	Dynamic or Static
Segmentation	Single or multiple	Single or multiple
Alignment	Yes (subsegment) Yes (segment)	Any
StartWithSAP (1, 2, 3*)	Yes (subsegment) Yes (segment)	Any
StartWithSAP (>3)	No	Yes
Segment Timeline	May be ignored	
Subsets	May be ignored	
Multiple Periods	Yes	
Multiplexed	Yes	
Non-multiplexed	No	

Other Profiles

- **ISOBMFF Extended onDemand and live**
  - Base: onDemand and live
  - Adds support for xlink:href
    - Content personalization
    - Dynamic ad insertion
- **ISOBMFF Common Profile**
  - Combined use of onDemand et live profiles
  - Profiles signaled at the MPD and Period levels
    - Allows mixing a Live Period with an OnDemand Period

Profiles adopted by HBBTV 2.0 et DVB-DASH profiles

Other Profiles

- **ISOBMFF Broadcast profile**
  - Base: ISOBMFF live
  - Adds support for segment sequences addressing
    - Different SAP frequencies in multiple representations
    - Different segment duration in multiple representations
  - Allows fast bootstrap of broadcast through broadband
    - Eg broadcast segment 10s with broadband segment 1s

Profiles adopted by ATSC 3.0 profiles