Mate Nadasdi / @matenadasdi

Frontend Tech Lead Engineer

Watson Media / Cloud Video — IBM Budapest Lab

VIDEO THE UPSIDE DOWN

Let’s skip the Flash part

It was a huge thing back then, but it’s history now!

Me some years ago:

Flash gonna die?
Let’s write a Player from scratch, it’s just JS!

So what is “video” on the web?

Be careful, if you open the door, it will haunt you!

The <video> media element

- **plugin-free** and **javascript-free** usage is possible
- it survived the patent law war already
- requires specific codec + container combinations per browser
- Works in most browsers: h.264 + AAC or mp3 in MP4, VP8 + Vorbis in WebM

`HTMLVideoElement` **object**

- Some important **Events:**

  `canplay, play, playing, progress, seeked, seeking, timeupdate, stalled, waiting, ended, pause`

- Some important **Props:**

  `videotracks, audiotracks, texttracks, currentTime, playbackRate, duration`

Level I. - <video src=“”>

Usage:

```
<video src="demo.mp4" controls autoplay></video>
```

Purpose:

- To play simple small video files directly from a CDN.
- If the server accepts, it uses **ByteRange requests.**

Level II. - <source> & <track>

Usage:

```
<video>
   <source src="demo.mp4" type="video/mp4" />
   <source src="demo.webm" type="video/webm" />
   <source src="demo.ogv" type="video/ogg" />

   <track kind="subtitles" src="sampleSubtitles_en.vtt" srclang="en">
   <p>Sad Panda, HTML5 video element is not supported in your browser</p>
 </video>
```

Purpose:

- <source> tag to support multiple formats for different browsers
- <track> to add text tracks to your video/audio for a11y purposes.

Level III? - Direct HLS `<video src=“*.m3u8”>`

Usage:

```
<video src=“*.m3u8”>
```

Purpose:

- Apple’s own standard to play video/live content out-of-the-box without JS.
- **iOS/Android + Desktop Safari/Edge only**
- Chrome is not planning to add native support on desktop
- It works, but it’s fully closed, **no control.**

Ooh this is easy!

Yeah, for basic usage.

Ok, let’s write REAL a Player!

This is the point where the boogie starts!

So many acronyms!

MSE

HLS

DASH

EOS

VTT

QoE

QoS

ABR

DRM

EME

CENC

...

Some requirements

It should play the content ASAP with excellent quality without any issues at any time even if my internet is slow! 🤓

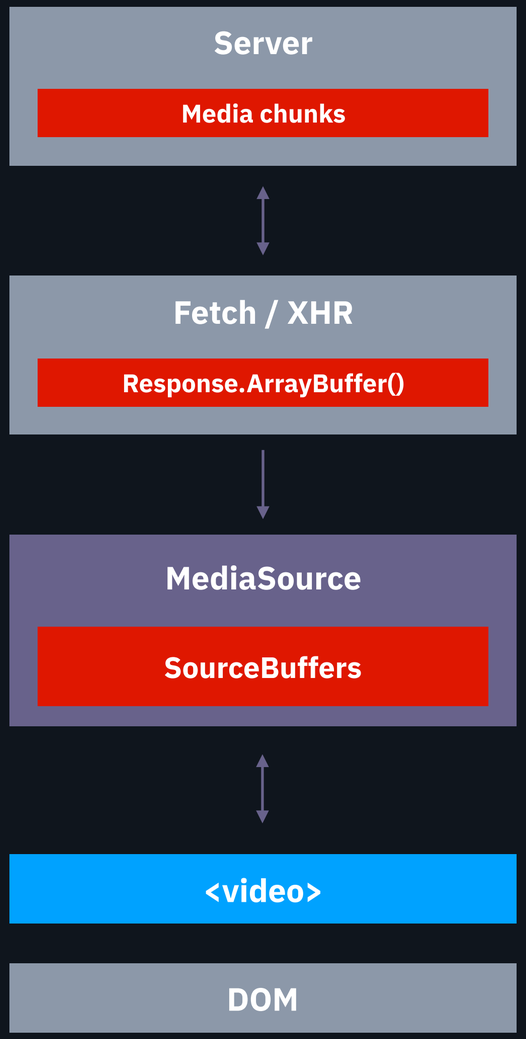
- **Control over everything**: Fast startup, low re-buffering ratio, failover handling
- Adaptive Streaming (**ABR**)
- Multi language audio, captions / subtitles
- Security
- n+1 features: Live, DVR, Multi camera, 360 / VR

<video> **alone is not enough.**

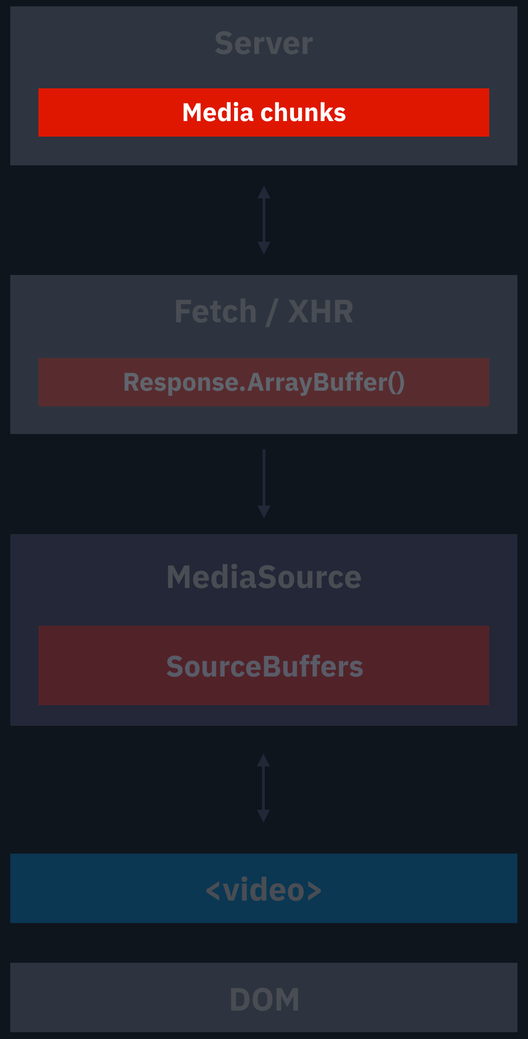
You cannot control the playback itself, you cannot change the source without breaking up your video playback. You cannot control how/when it downloads your content and you cannot specify quality constraints.

We need to be able to download and play our video piece-by-piece.

Media Source Extensions for the rescue ✌️



Media Source Extensions API

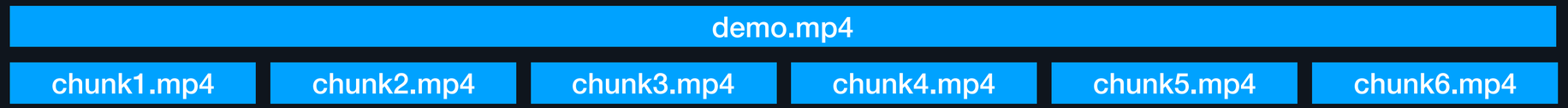


Server storage

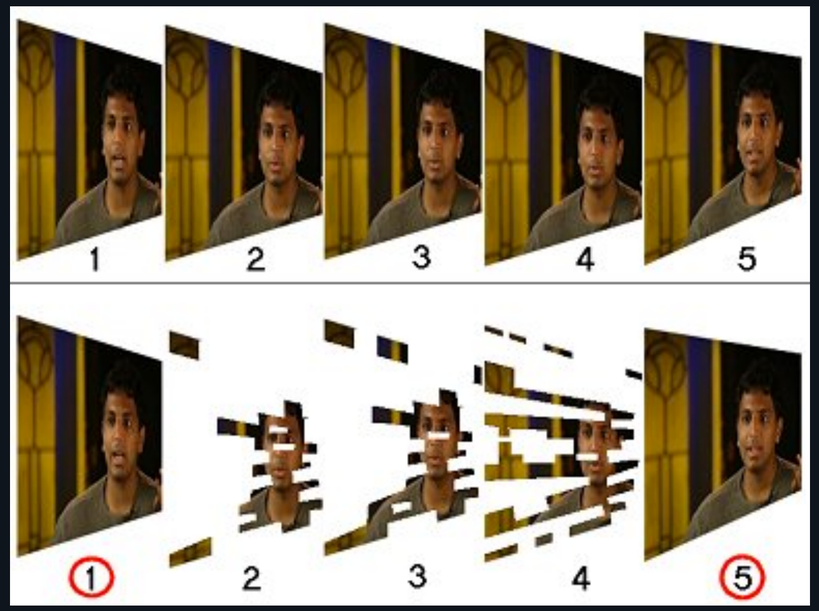
- Multiple bitrates & formats transcoded from the original media
- Video / Audio / Text content should be separated
- Multiple media container + codec combinations are needed to work in all browsers

Media Source Extensions API

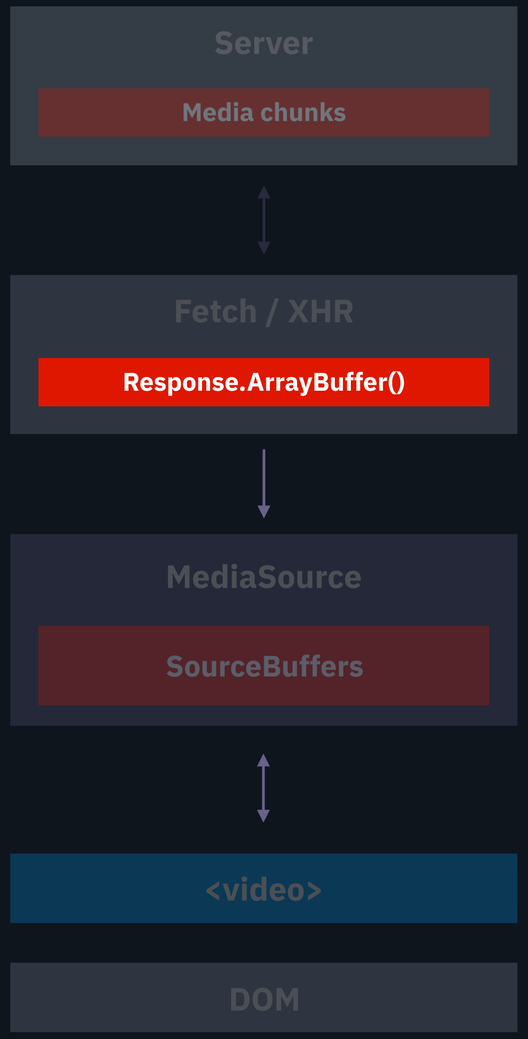
Chunks / Fragments / Segments: *pieces of our media content starting with a keyframe*



Keyframe: *reference points in your video*



Media Source Extensions API

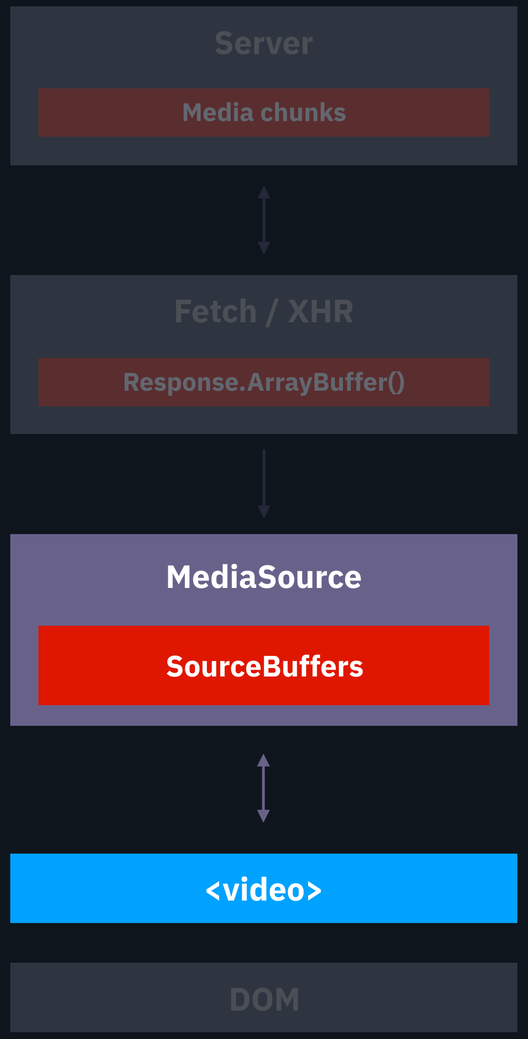


Fetching the data

- Since the fetched content is binary, we need `ArrayBuffers`
- `response.arrayBuffer()` for fetch
- `xhr.responseType = “arraybuffer”` for XHR

The `ArrayBuffer` object is used to represent a generic, fixed-length raw binary data buffer.

Media Source Extensions API



MediaSource & SourceBuffer

- **SourceBuffer:** Represents a chunk of media to be passed into an HTMLMediaElement and played via a MediaSource object.
  - Literally it’s an array for your media type where you can append new ArrayBuffers
- **MediaSource:** Represents a source of media data for an HTMLMediaElement object.
  - It will feed from the attached SourceBuffers.

Does this solve our requirement list? 🤞

- We can download **any bitrate** we think is the best, and feed it continuously.
- We can **mix content type versions** like audio / text / video without pausing
  - Multi language, different camera, different time (DVR)
- We can **change our source without pausing**, by appending a new chunk next time.
- We can manage memory consumption, we can use workers, whatever we want, it’s simple data fetching & pure JS.

MSE for the win!

Is it the new Demogorgon? 🤔

Creating content

- We need to create fragmented and segmented mp4 from original content
- `ffmpeg, Bento4, mp4box, shaka-packager, gpac, etc.`
- **DASH (.mpd) / HLS (.m3u8) / your own solution** 😱

```
./packager \
 ‘in=h264_720p.mp4,stream=video,init_segment=video/720/init.mp4,segment_template=video/720/$Number$.m4v’ \
 ‘in=h264_360p.mp4,stream=video,init_segment=video/480/init.mp4,segment_template=video/480/$Number$.m4v' \
 ‘in=h264_480p.mp4,stream=video,init_segment=video/360/init.mp4,segment_template=video/360/$Number$.m4v’ \

 'in=h264_720p.mp4,stream=audio,init_segment=audio/init.mp4,segment_template=audio/$Number$.m4a' \

 --generate_static_mpd --mpd_output multi.mpd --generate_static_mpd
```

```
<?xml version="1.0" encoding="UTF-8"?>
<MPD xmlns=“…” xmlns:xsi=“…” xmlns:xlink=“…” xsi:schemaLocation=“…” xmlns:cenc=“…”
profiles=“…” minBufferTime="PT2S" type="static" mediaPresentationDuration="PT170.92S">
  <Period id="0">
    <AdaptationSet id="0" contentType="video" maxWidth="1280" maxHeight="720"
frameRate="12800/512" par="16:9">
      <Representation id="0" bandwidth="995947" codecs="avc1.4d401f" mimeType="video/mp4"
sar="1:1" width="1280" height="720">
        <SegmentTemplate timescale="12800" initialization="video/720/init.mp4" media="video/
720/$Number$.m4v" startNumber="1">
          <SegmentTimeline>
            <S t="0" d="165888"/>
            <S t="165888" d="112128"/>
            <S t="278016" d="134656"/>
            <S t="1540096" d="124416"/>
          </SegmentTimeline>
        </SegmentTemplate>
      </Representation>
    <Representation id="1" bandwidth="587389" codecs="avc1.42c01e" mimeType="video/mp4"
    sar="1:1" width="640" height="360">
      <SegmentTemplate timescale="12800" initialization="video/480/init.mp4" media="video/
    480/$Number$.m4v" startNumber="1">
        <SegmentTimeline>
          <S t="0" d="147456"/>
```

Fetching: **What & When?**

- **When:**
  - video tag **`timeupdate`** or **`stalled`** event is a good moment to calculate what's needed exactly into the buffer at the **`currentTime`**
- **What:**
  - SegmentTimeline: contains start timestamp and length for every segment
  - Segment numbering + fix chunk length

```
video.addEventListener('timeupdate', () => {
  if (video.currentTime) {
    let videoChunkIndex = parser.getSegmentIndexForTime(video.currentTime, CONTENT_TYPES.video);
    let audioChunkIndex = parser.getSegmentIndexForTime(video.currentTime, CONTENT_TYPES.audio);

    fetchChunkForType(CONTENT_TYPES.video, videoChunkIndex);
    fetchChunkForType(CONTENT_TYPES.audio, audioChunkIndex);
  }
});
```

ABR

Using fetch/XHR with ArrayBuffers is trivial, but there is a bigger question if you would like a smart fetching algorithm.

- use **constraints** like **screen size, CPU** `(video.totalVideoFrames / video. droppedVideoFrames)`
- try to fetch the most suitable quality for the user’s bandwidth
  - measuring bandwidth is not trivial since we don’t have native support
  - **Resource Timing API** can be a good starting point
- Bandwidth vs Buffer based approaches

```
return fetchChunk(chunkFilePath)
  .then(arrayBuffer => {
    let timing = window.performance.getEntriesByName(chunkFilePath)[0];


    // you can write your own ABR based on window.performance data
    if (timing.responseEnd - timing.responseStart > downloadThreshold) {
      // anything
    }
```

Display your binary data


```
const mediaSource = new MediaSource();
video.src = window.URL.createObjectURL(mediaSource);

mediaSource.addEventListener('sourceopen', () => {
  let sourceBuffer =
    mediaSource.addSourceBuffer('video/webm; codecs="vorbis,vp8"');

  // get video segments and append them to the source buffer
  fetch(videoUrl).then(arrayBuffer => sourceBuffer.appendBuffer(arrayBuffer));
});
```

Display what you want 😯

Video is all about images after each other, do what you want with them.

```
let video = document.querySelector('#myVideoElement');
let videoCanvas = document.createElement('canvas');
let videoContext = videoCanvas.getContext('2d');

videoCanvas.width = video.videoWidth;
videoCanvas.height = video.videoHeight;

requestAnimationFrame(() => {
  // do anything with your image on the canvas!
  videoContext.drawImage(video, 0, 0, videoImage.width, videoImage.height);
});
```

360 / VR player? 🤨

- Thanks to the **canvas <-> video tag** “integration” you can do anything with your frames
- 360 streams are just like any other stream for your player
- Use **ThreeJS with a sphere + a perspective camera** in the inside

Why should you use MSE?

Because you can!*

*Because you can+should optimise your experience for your users

Thank you!

Questions?

Mate Nadasdi / @matenadasdi