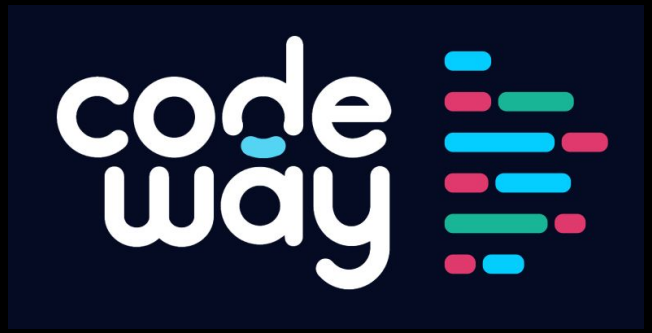



# Mobile Video: Challenges and Opportunities

Michael Dale

@michael_dale

# Who am I?

- Long time open video advocate at Wikimedia, Mozilla etc.
- Co-organizer of FOMS ( link )
- Working on delivering premium video experiences across screens
- Now at Ellation / Crunchyroll

# What are we talking about?

- Standards that enable a premium video experience
- A technical look at these standards and how to use them

# Early web video was a painful


# Many Competing standards

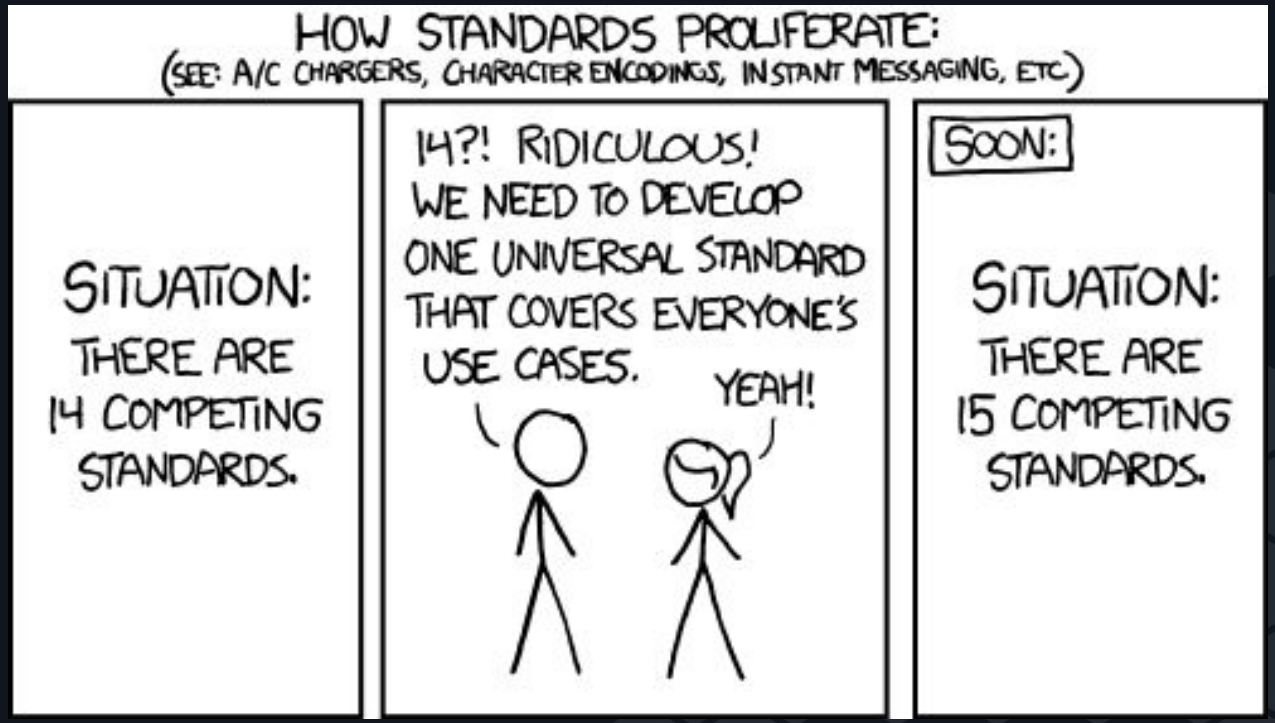
Delivery:

- Microsoft Smooth Streaming ( playback with silverlight )
- Adobe HDS ( playback with flash players )
- Apple HLS ( apple quicktime contexts )
- Set top box proprietary IP delivery protocols

Codecs / Containers:

- Fragmented Mp4
- MPEG transport stream
- Webm / Matroska

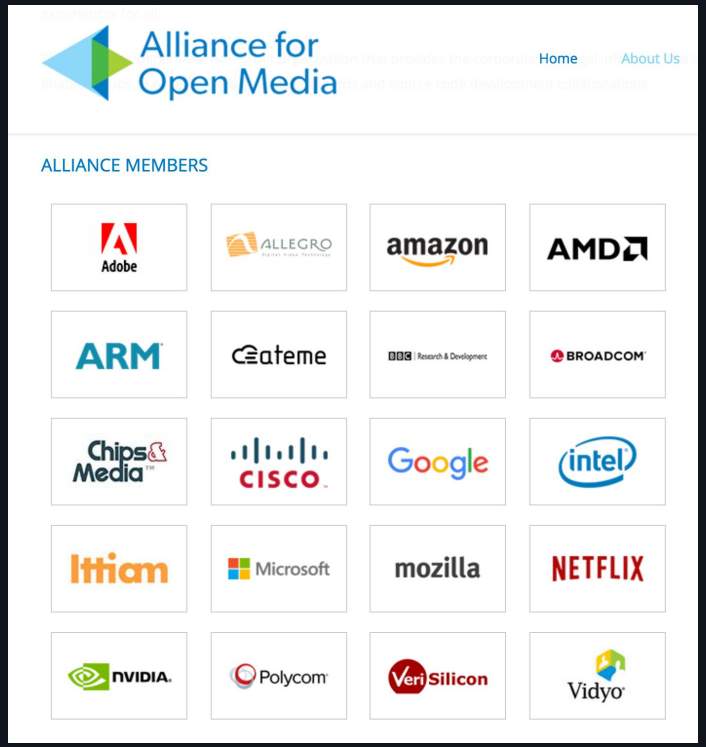
# Standards without coordination:

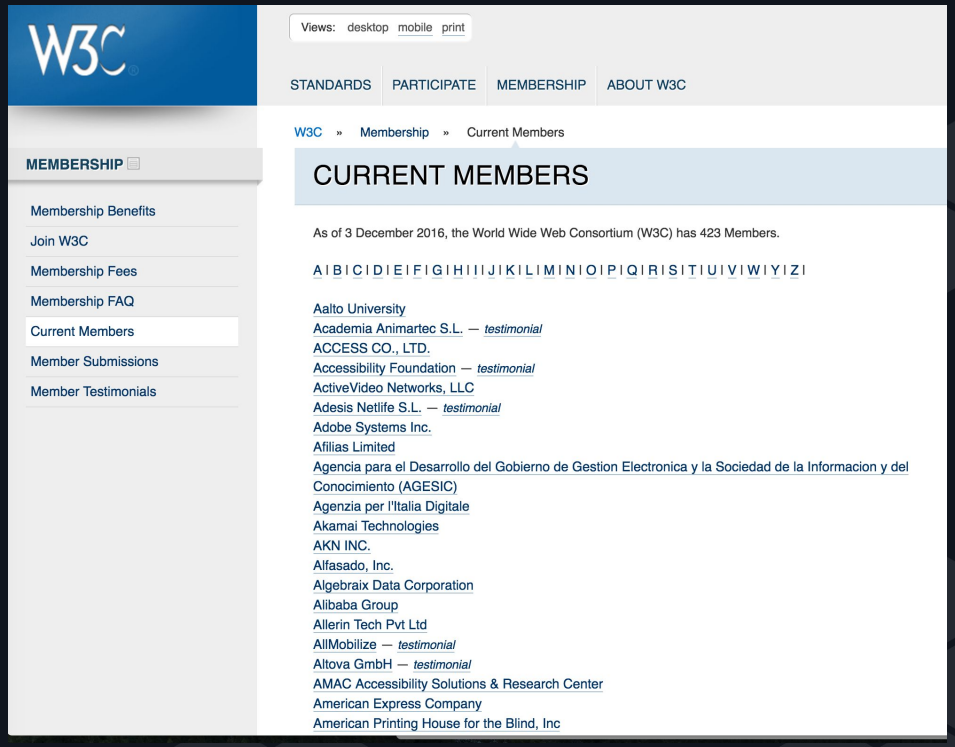


# Standards are great ... but

## ultimately driven by implementation and market share

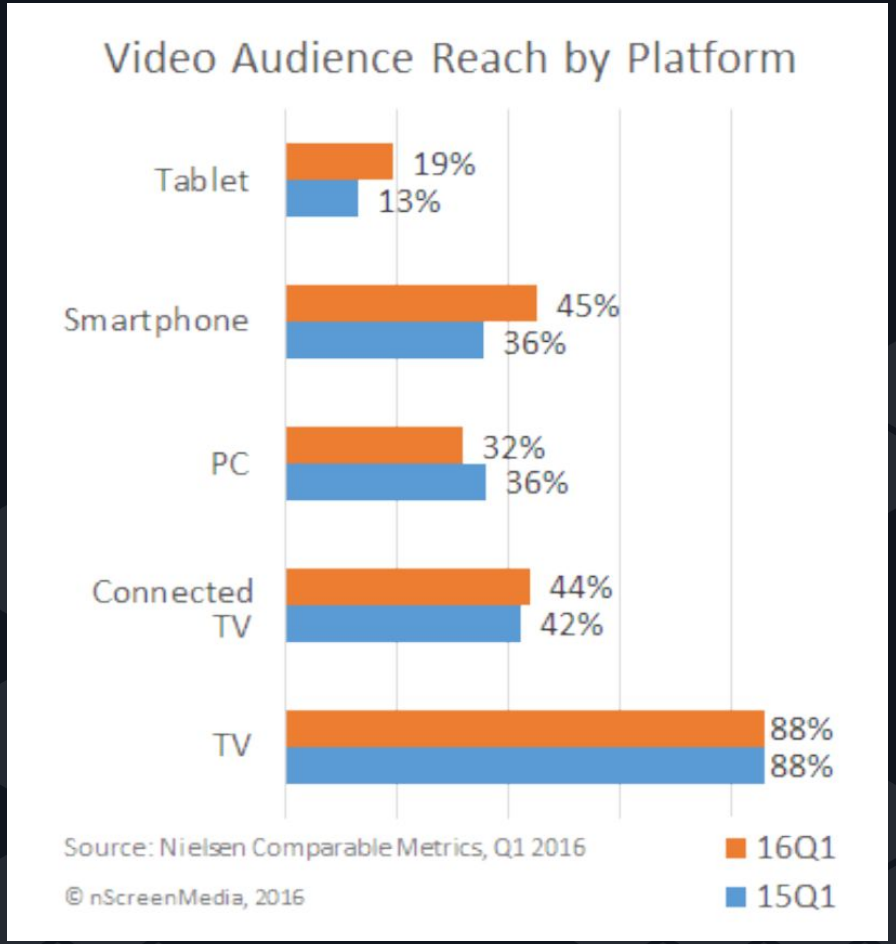
# Generally recognized today as critical by all major players



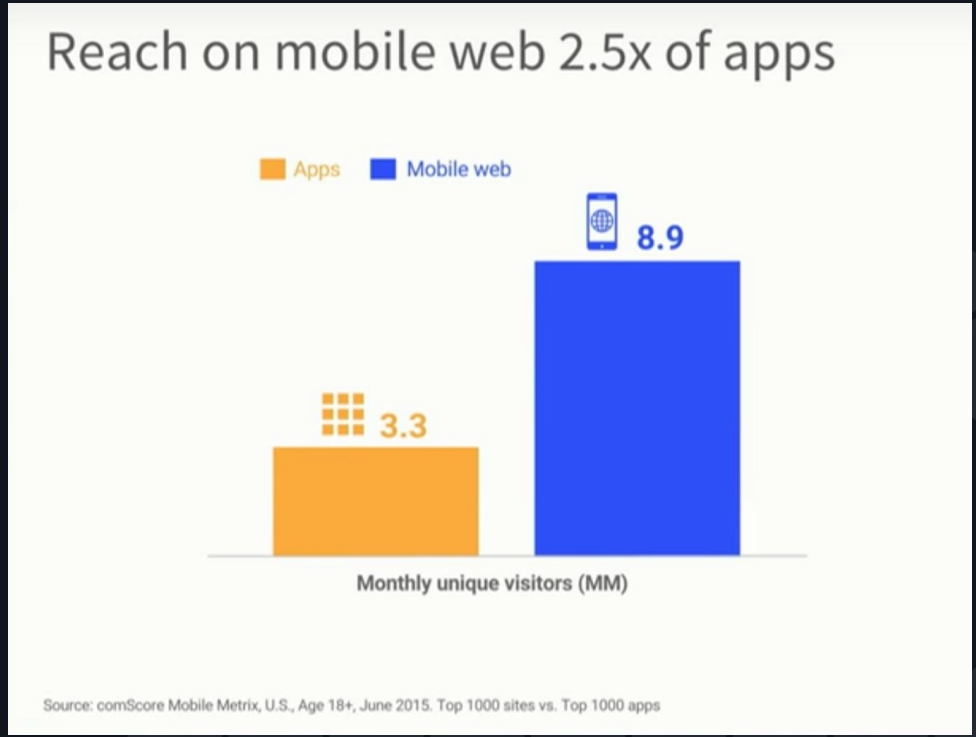
# Mobile Video

- Mobile has overtaken PC video viewing this year
- Connected TVs have similar reach to smartphones and have overtaken PC for IP video.



# Mobile Web is a big part of extended reach

- Mobile web within mobile has more reach but limitations around premium video features




Top 1000 sites vs top 1000 apps

# What are “premium” video requirements ?

- Adaptive streaming -- reduce buffering startup time
- Protected content -- digital rights management.
- Seamless Ads insertion -- reduce monetization falloff
- Low latency live video -- interactivity requires low latency

# Beyond HTML5 for Premium video experience

Standards addressing these challenges:

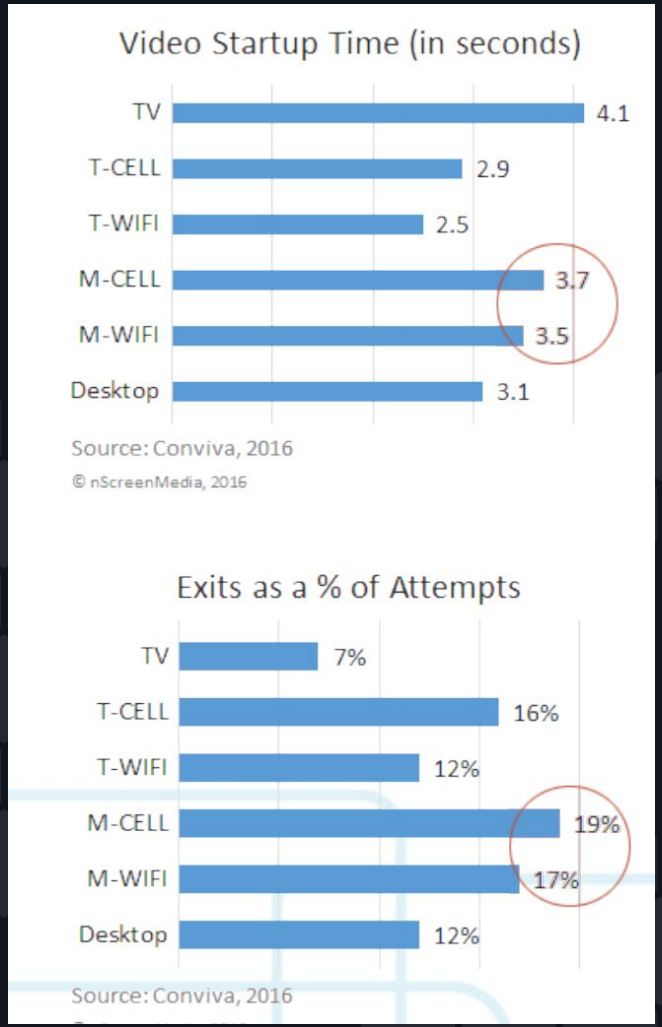
- Beyond <video> tag.
- MSE - Media source extensions
- EME - Encrypted media extensions
- Media Capabilities API
- HTTP2 - Delivery

# Web standards are mobile web standards

- Vertical integration of platforms consolidates browser based platform across screens
- Robust HTML5 support is everywhere.

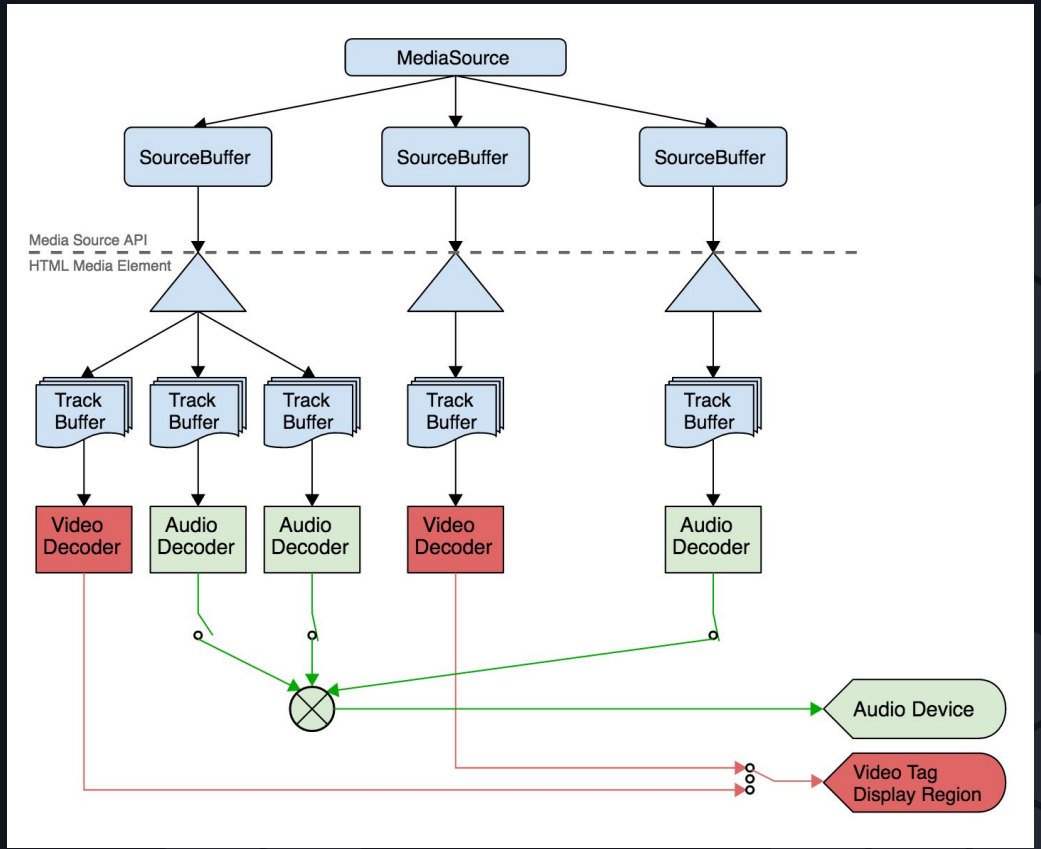
# MSE - Media source extensions

- Ease of context switching in mobile means low tolerance for long start time
- Controlling the video buffer and stream selection with MSE helps control startup time.



# MSE - Media source extensions
# How it works?

- Exposes a source buffer for adaptive streaming
- Enables lower level control over video data flow

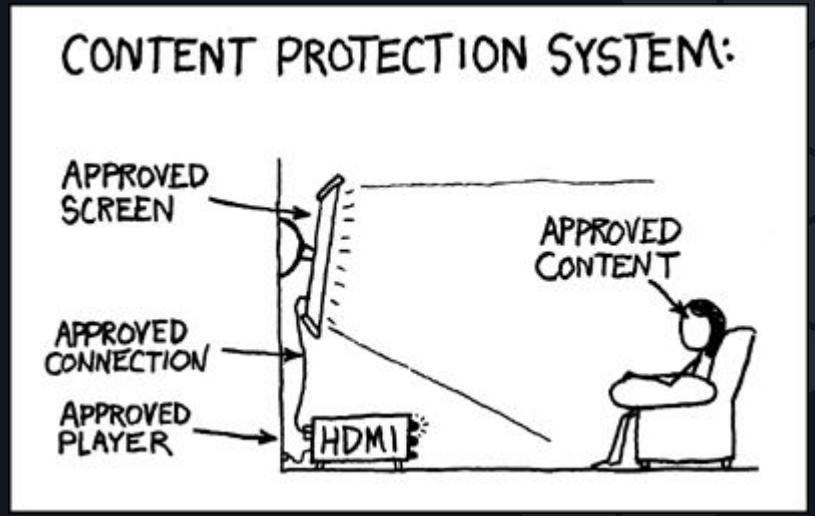


# MSE - Media source extensions

```
var video = document.getElementById('v');
var mediaSource = new MediaSource();
mediaSource.addEventListener('sourceopen', onSourceOpen.bind(this, video));
video.src = window.URL.createObjectURL(mediaSource);

function onSourceOpen(videoTag, e) {
  var mediaSource = e.target;

  …

  // The segment can be retrieved via XHR request, loaded from HTML5 storage etc.
  var mediaSegment = getVideoSegment();

  // once you have video bytes you can append them:
  mediaSource.sourceBuffers[0].appendBuffer(mediaSegment);
```

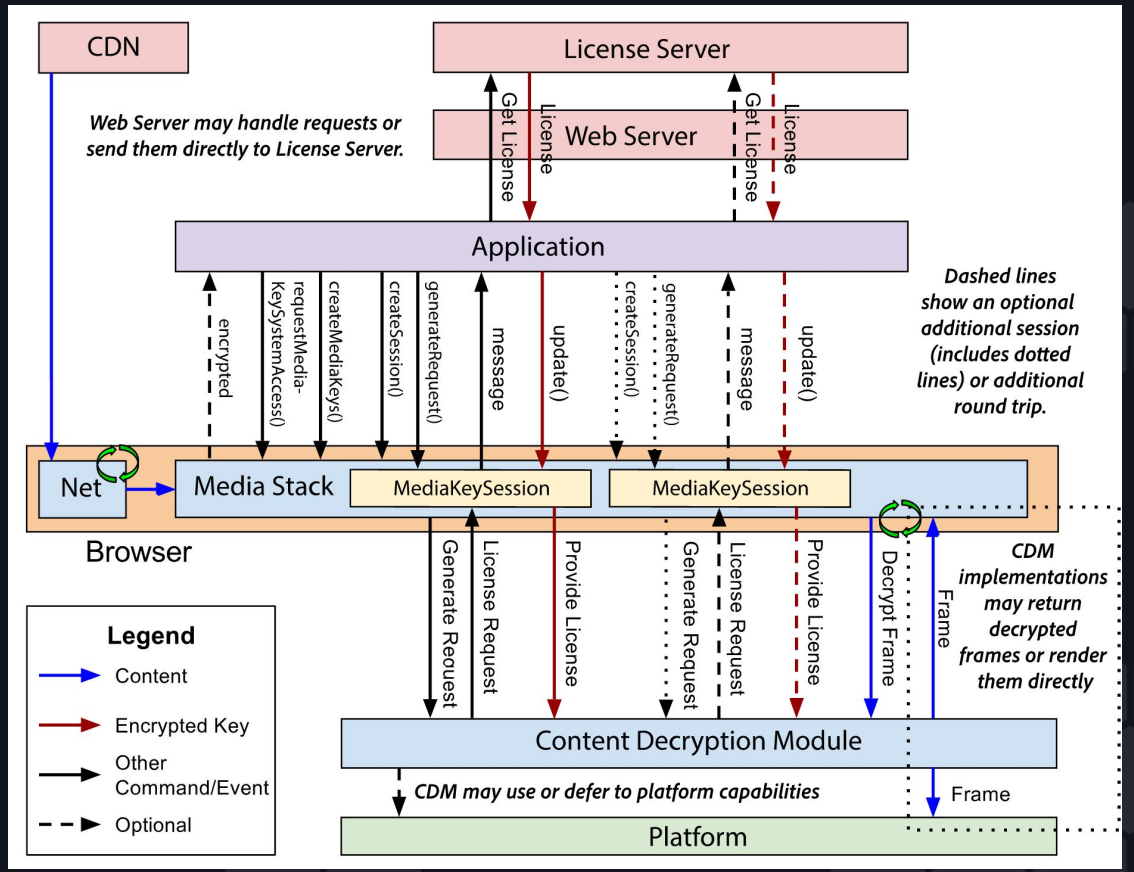
# EME - Encrypted media extensions - Why ?

- Delays freshly released content from being available
- Part of content licensing requirements
- Enables “protected” content in web context



# EME - Encrypted media extensions - how it works?

Standard way to interface with the CDM ( content decryption module )



# EME - Encrypted media extensions - example

```
var video = document.getElementById('video');
if (!video.mediaKeys) {
  navigator.requestMediaKeySystemAccess('org.w3.clearkey', [
    { initDataTypes: ['webm'],
      videoCapabilities: [{ contentType: 'video/webm; codecs="vp8"' }] }
  ]).then(
    function(keySystemAccess) {
      var promise =  keySystemAccess.createMediaKeys();
      promise.then(
        function(createdMediaKeys) {
          return video.setMediaKeys(createdMediaKeys);
        }
      )
      promise.then(
        function(createdMediaKeys) {
          var te = new TextEncoder();
          var initData = te.encode( '{"kids":["LwVHf8JLtPrv2GUXFW2v_A"]}');
          var keySession = createdMediaKeys.createSession();
          keySession.addEventListener("message", handleMessage, false);
          return keySession.generateRequest('keyids', initData);
        }
      )
  );
}
```

# Media Capabilities API ( in proposal stage )

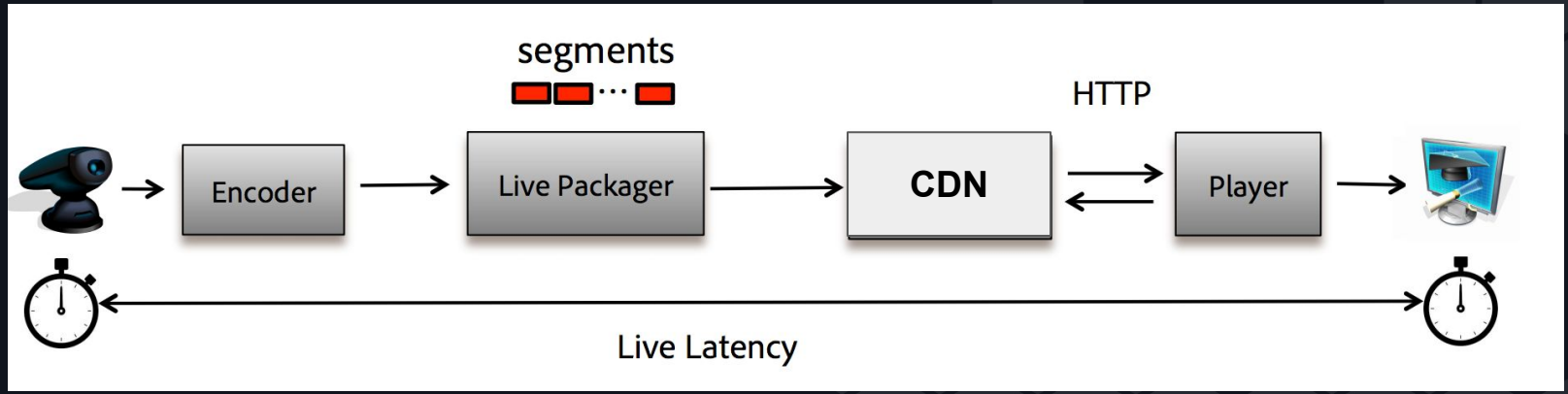
- Today we only have canPlayType

```
var obj = document.createElement('video');
console.log(obj.canPlayType('video/mp4')); // "maybe"
```

- Soon describes capabilities:
  - Will playback have high quality (smoothness)? Will it be power efficient?
  - Do output capabilities (e.g. color gamut, dynamic range, audio channels) match content?
  - Are security requirements (e.g. encryption, HDCP, etc) supported or playback-impacting?
  - Given multiple possible media formats, which is preferable?
- Still being reviewed, see proposal here.

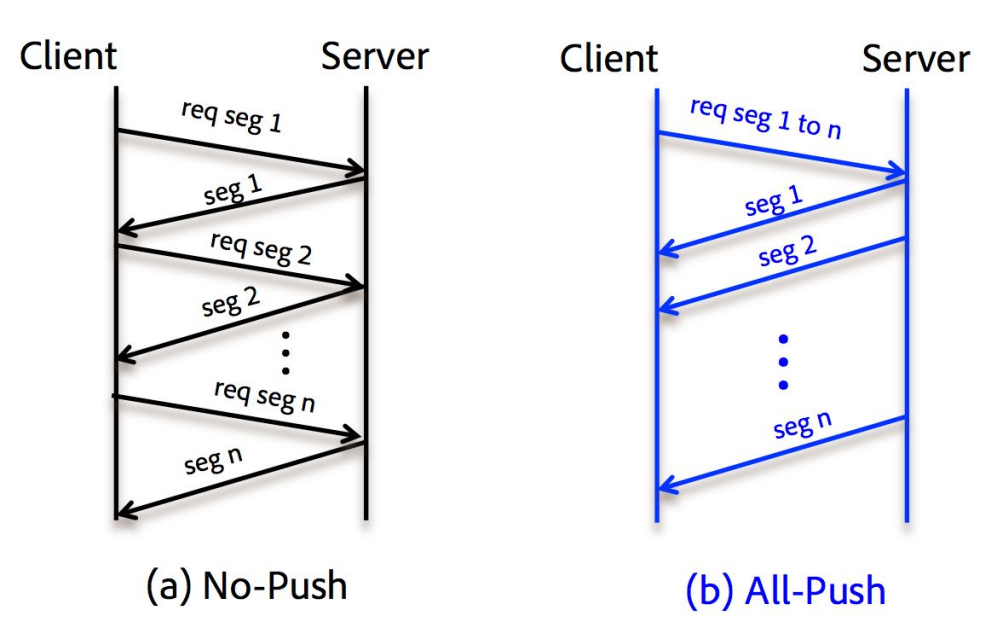
# Low latency live?

- Live streaming, a huge market segment.
- Interactivity requires low latency.
- Scalable delivery requires http based delivery



# HTTP2 enables lower latency live streaming

- Standard update to http,
- Supported by Chrome/Firefox/Opera/IE
- HTTP2; update to HTTP; enables continuous delivery of segments
- HTTP2 “Push” enables lower delay as there is no manifest round trip request.



# HTTP2 how do you use it?

- Depends on your CDNs implementation
- Example here shows http2 push via “link” header on cloudflare[1]
- Video Streaming server implementations are coming to market.

```
HTTP/2.0 200
Server: nginx/1.9.15
Date: Fri, 13 May 2016 10:52:13 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
Link: </images/drucken.jpg>; rel=preload; as=image
Link: </images/empire.jpg>; rel=preload; as=image
```

```
<?php
  function pushImage($uri) {
     header("Link: <{$uri}>; rel=preload; as=image", false);
     return <<<HTML
  <img src="{$uri}">
  HTML;
  }

  $image1 = pushImage("/images/drucken.jpg");
  $image2 = pushImage("/images/empire.jpg");
  ?>
```

# Thanks, questions ?

@michael_dale