HTML5 Video

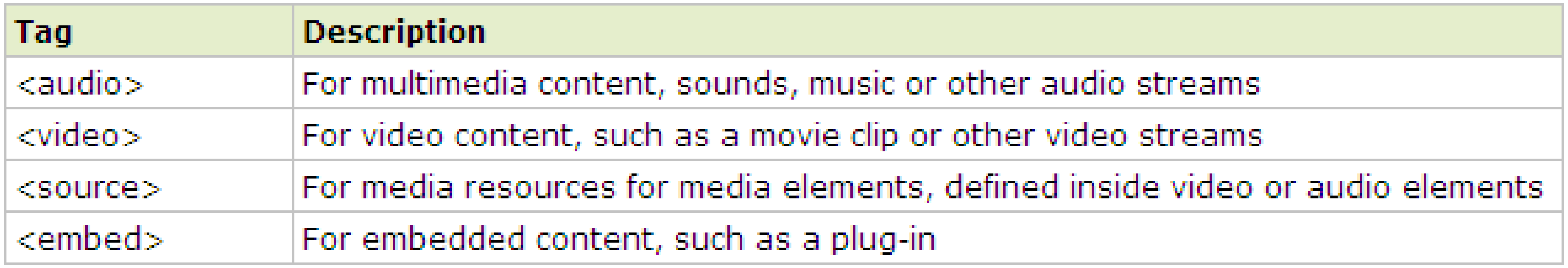
IINTERACTION WITH CSS AND JAVASCRIPT & SYNCHRONIZATION

POLITECNICO
DI TORINO

e-Lite

HTML5 media elements

- New HTML5 media elements



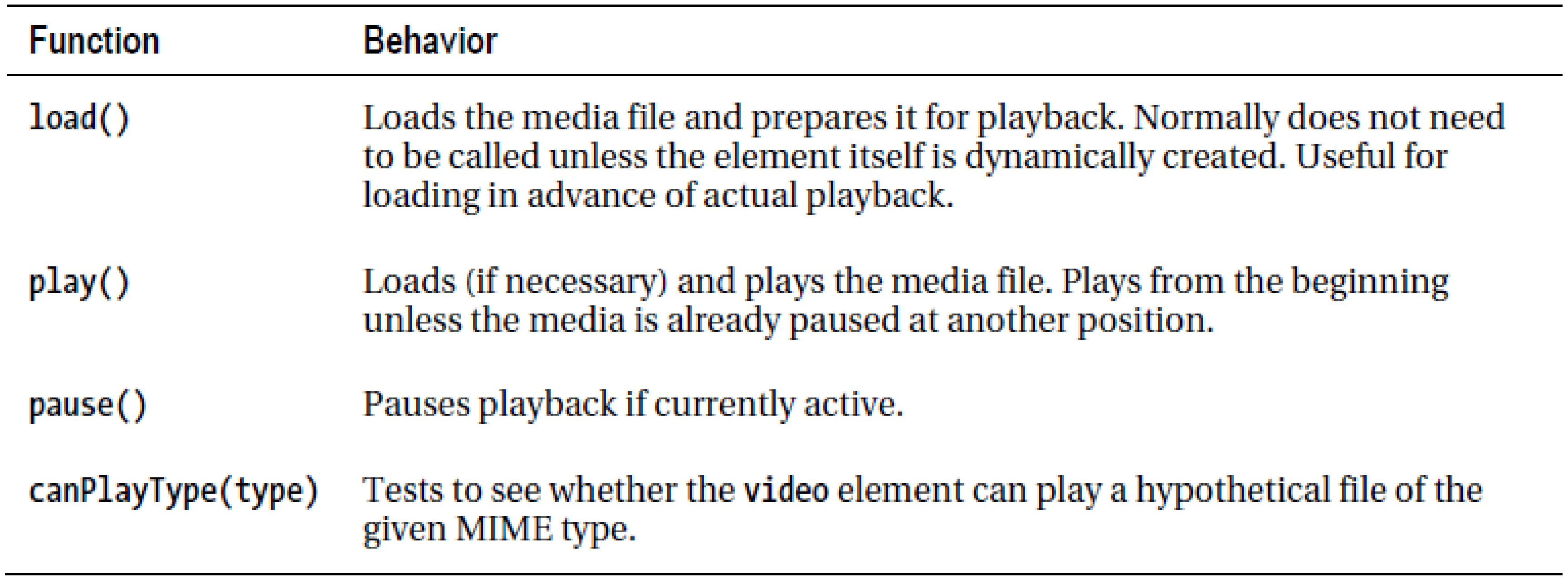
Tag	Description
<audio>	For multimedia content, sounds, music or other audio streams
<video>	For video content, such as a movie clip or other video streams
<source>	For media resources for media elements, defined inside video or audio elements
<embed>	For embedded content, such as a plug-in

- The new audio and video tags make multimedia no longer a second-class citizen on the web
  - No separate download or enabled/disabled issues
  - No separate rendering (problems with HTML elements overlap)
  - Keyboard accessibility, styling with CSS, combining video and canvas

HTML5 media elements

- The media elements expose a common, integrated, and scriptable API to the document
  - You can design and program your own multimedia controls (e.g., play, seek, etc.)
- Examples
  - http://www.craftymind.com/factory/html5video/CanvasVideo3D.html
  - http://www.craftymind.com/factory/html5video/CanvasVideo.html

JS APIs for media control



Function	Behavior
`load()`	Loads the media file and prepares it for playback. Normally does not need to be called unless the element itself is dynamically created. Useful for loading in advance of actual playback.
`play()`	Loads (if necessary) and plays the media file. Plays from the beginning unless the media is already paused at another position.
`pause()`	Pauses playback if currently active.
`canPlayType(type)`	Tests to see whether the `video` element can play a hypothetical file of the given MIME type.

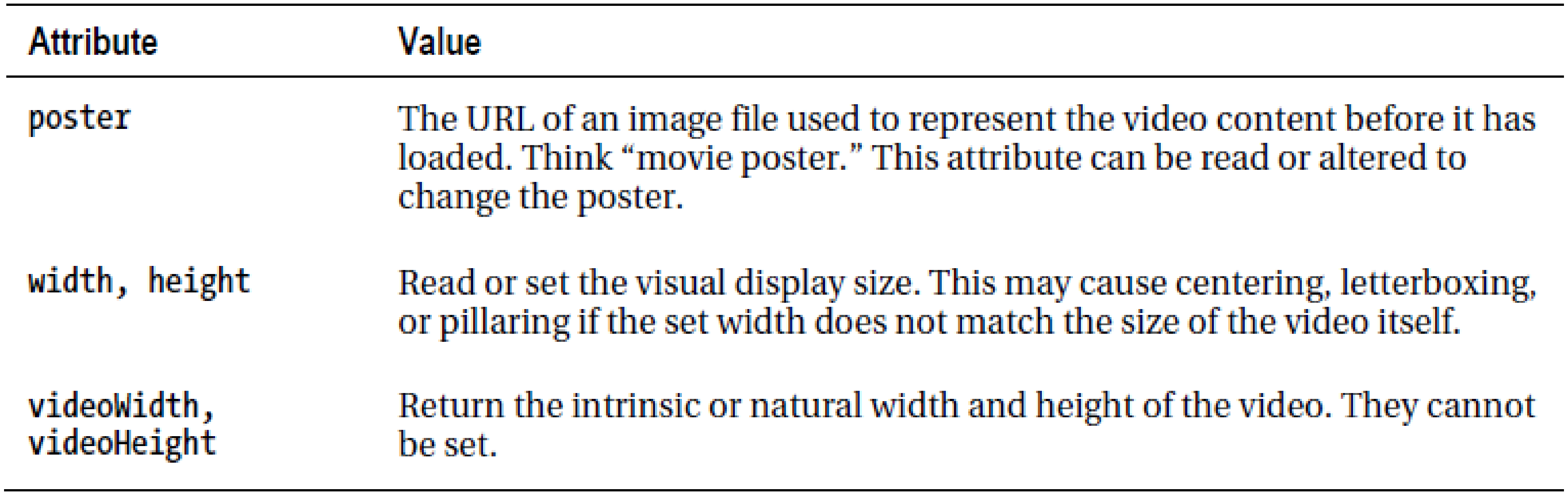
Media attributes

Read-only attribute	Value
`duration`	The duration of the full media clip, in seconds. If the full duration is not known, `NaN` is returned.
`paused`	Returns `true` if the media clip is currently paused. Defaults to `true` if the clip has not started playing.
`ended`	Returns `true` if the media clip has finished playing.
`startTime`	Returns the earliest possible value for playback start time. This will usually be 0.0 unless the media clip is streamed and earlier content has left the buffer.
`error`	An error code, if an error has occurred.
`currentSrc`	Returns the string representing the file that is currently being displayed or loaded. This will match the source element selected by the browser.

Media attributes

Attribute	Value
`autoplay`	Sets the media clip to play upon creation or query whether it is set to `autoplay`.
`loop`	Returns `true` if the clip will restart upon ending or sets the clip to loop (or not loop).
`currentTime`	Returns the current time in seconds that has elapsed since the beginning of the playback. Sets `currentTime` to seek to a specific position in the clip playback.
`controls`	Shows or hides the user controls, or queries whether they are currently visible.
`volume`	Sets the audio volume to a relative value between 0.0 and 1.0, or queries the value of the same.
`muted`	Mutes or unmutes the audio, or determines the current mute state.
`autobuffer`	Tells the player whether or not to attempt to load the media file before playback is initiated. If the media is set for auto-playback, this attribute is ignored.

Additional video attributes



Attribute	Value
`poster`	The URL of an image file used to represent the video content before it has loaded. Think "movie poster." This attribute can be read or altered to change the poster.
`width, height`	Read or set the visual display size. This may cause centering, letterboxing, or pillaring if the set width does not match the size of the video itself.
`videoWidth, videoHeight`	Return the intrinsic or natural width and height of the video. They cannot be set.

Example: mouseover video playback

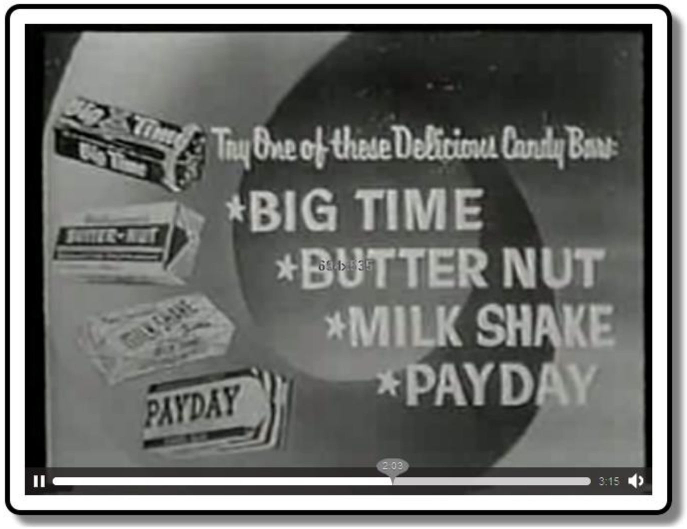
```
<!DOCTYPE html>
<html>
  <link rel="stylesheet" href="styles.css">
  <title>Mouseover Video</title>
  <video id="movies" onmouseover="this.play()"
    onmouseout="this.pause()" autobuffer="true"
    width="400px" height="300px">
    <source src="Intermission-Walk-in.ogv"
      type='video/ogg; codecs="theora, vorbis"'>
    <source src="Intermission-Walk-in_512kb.mp4"
      type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
  </video>
</html>
```

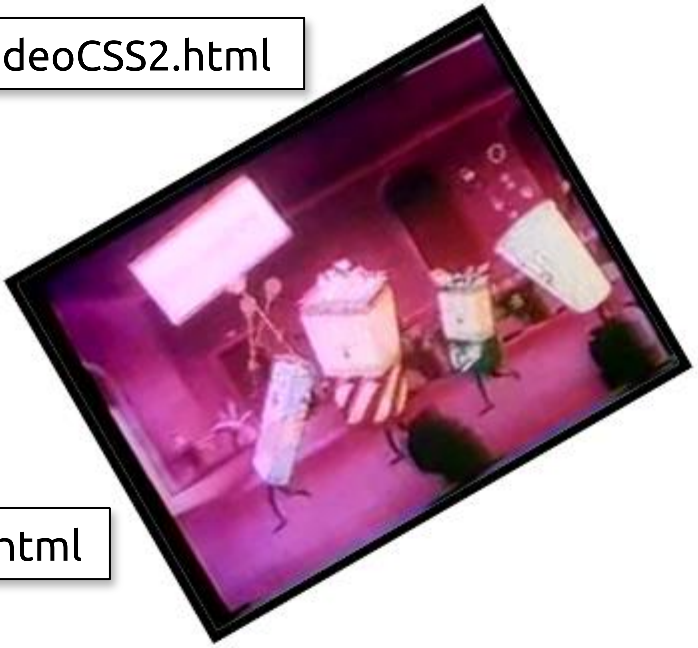
mouseoverVideo.htm

Video + CSS

- The video tag can be styled using traditional CSS (e.g. border, opacity, etc) since it is a first-class citizen in the DOM
  - You can also style it with the latest CSS3 properties like reections, masks, gradients, transforms, transitions and animations
- Examples

videoCSS2.html



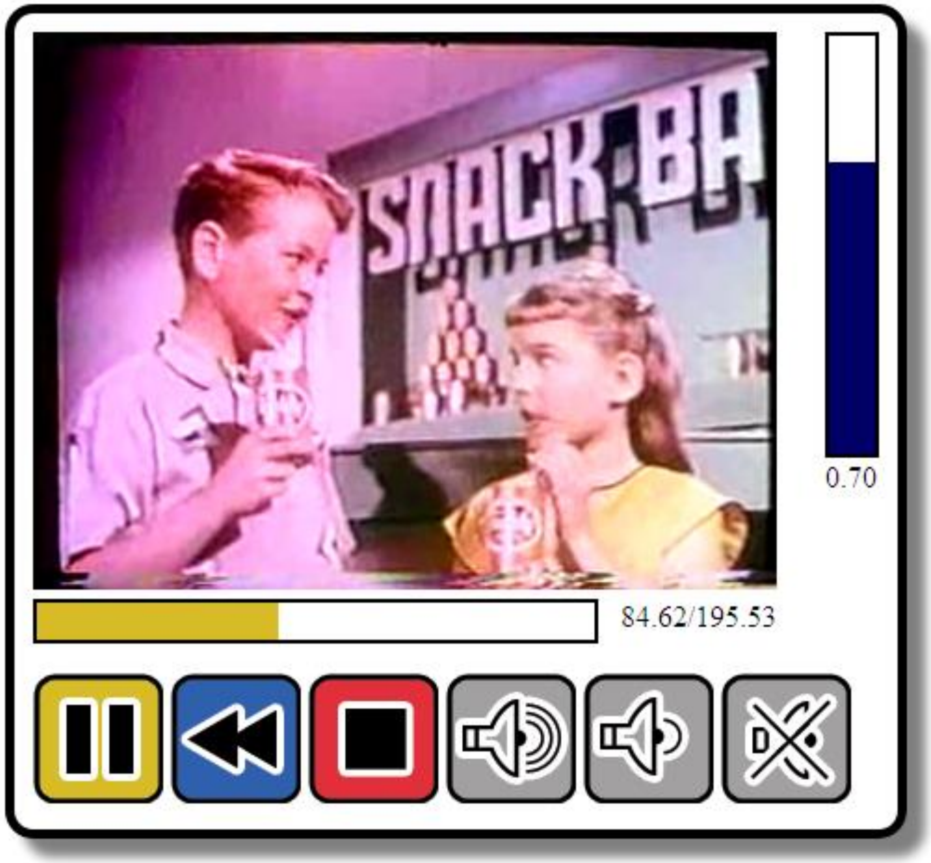


videoCSS1.html

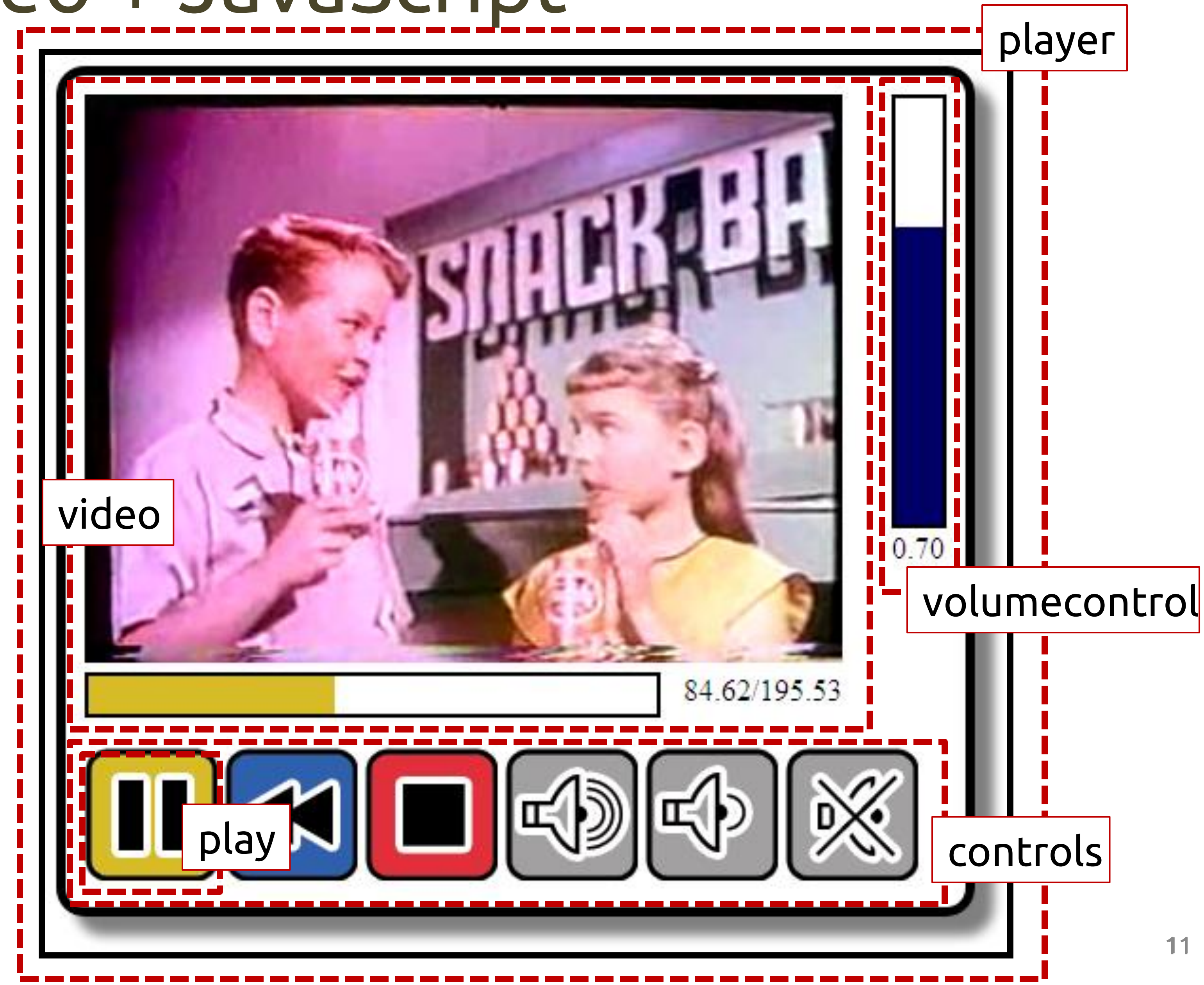
Video + JavaScript

- Example

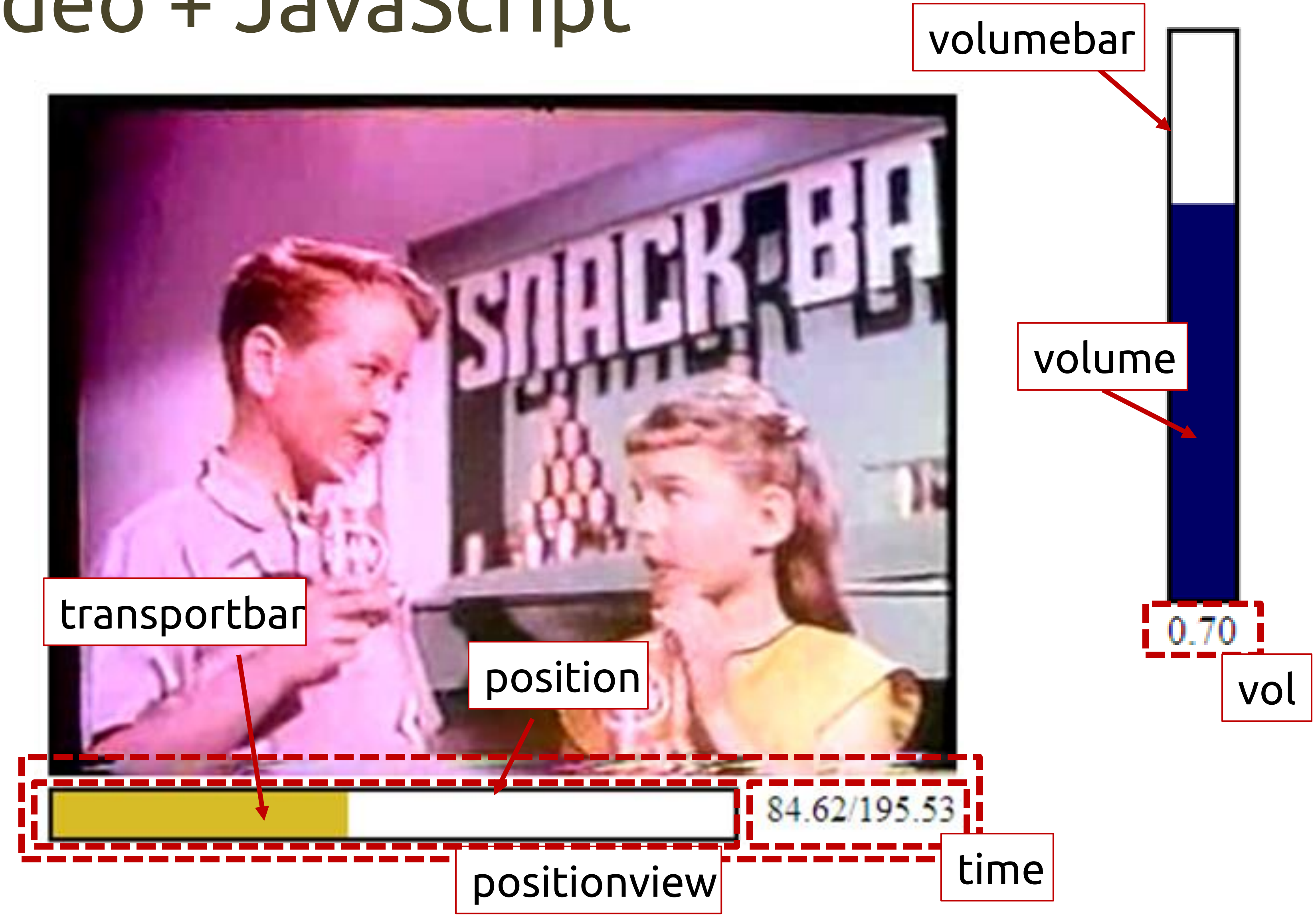
videoJS.html



Video + JavaScript



Video + JavaScript



Video and canvas

Example: video timeline viewer

- Autoplay attribute: the video starts as soon as the page loads
- Two additional event handler functions, oncanplay (when the video is loaded and ready to begin play) and onended (when the video ends)

```
<video id="movies" autoplay oncanplay="startVideo()"
       onended="stopTimeline()" autobuffer="true"
       width="400px" height="300px">
  <source src="Intermission-Walk-in.ogv"
       type='video/ogg; codecs="theora, vorbis"'>
  <source src="Intermission-Walk-in_512kb.mp4"
       type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
</video>
```

- Canvas called timeline into which we will draw frames of video at regular intervals

```
<canvas id="timeline" width="400px" height="300px">
```

Example: video timeline viewer

- Variables declaration

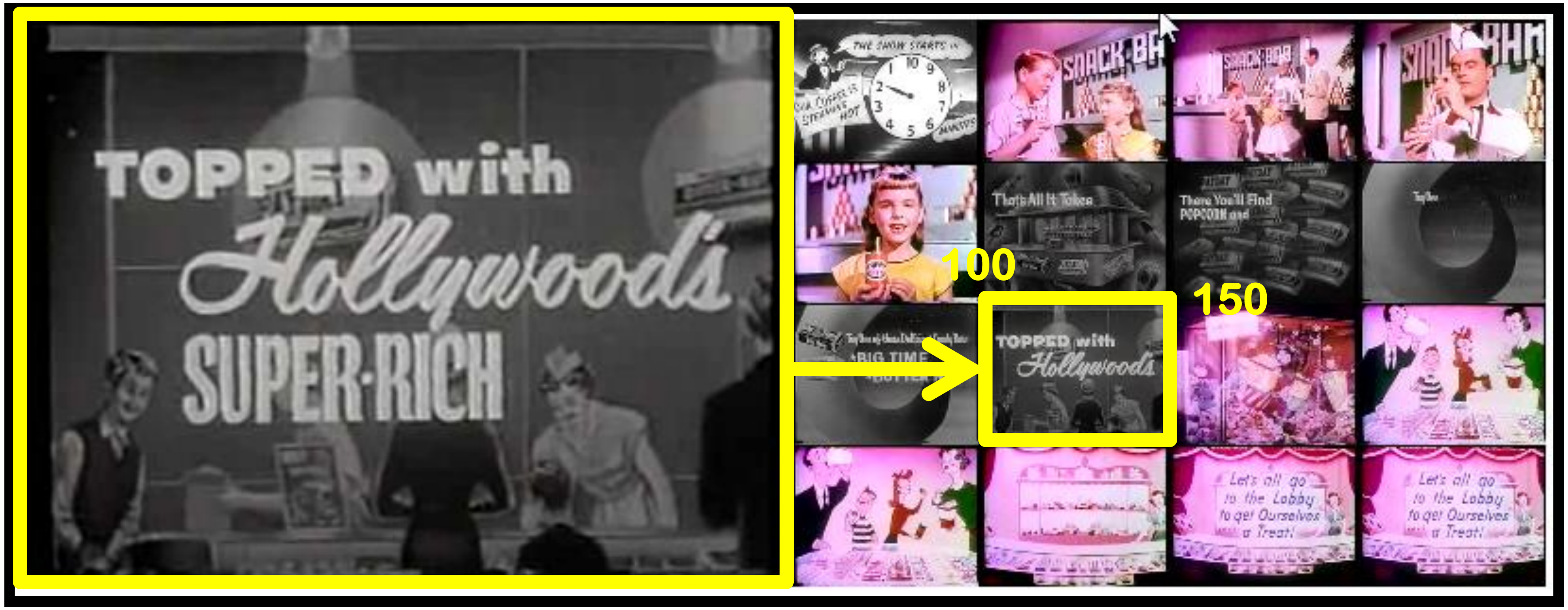
```
// # of milliseconds between timeline frame updates (5sec)
var updateInterval = 5000;
// size of the timeline frames
var frameWidth = 100;
var frameHeight = 75;
// number of timeline frames
var frameRows = 4;
var frameColumns = 4;
var frameGrid = frameRows * frameColumns;
// current frame
var frameCount = 0;
// to cancel the timer at end of play
var intervalId;

var videoStarted = false;
```

Example: video timeline viewer

- Function updateFrame: grabs a video frame and draws it onto the canvas

```
// paints a representation of the video frame into canvas
function updateFrame() {
  var video = document.getElementById("movies");
  var timeline = document.getElementById("timeline");
  var ctx = timeline.getContext("2d");
  // calculate out the current position based on frame
  // count, then draw the image there using the video
  // as a source
  var framePosition = frameCount % frameGrid;
  var frameX = (framePosition % frameColumns) * frameWidth;
  var frameY = (Math.floor(framePosition / frameRows)) *
               frameHeight;
  ctx.drawImage(video, 0, 0, 400, 300, frameX, frameY,
  frameWidth, frameHeight);
  frameCount++;
}
```

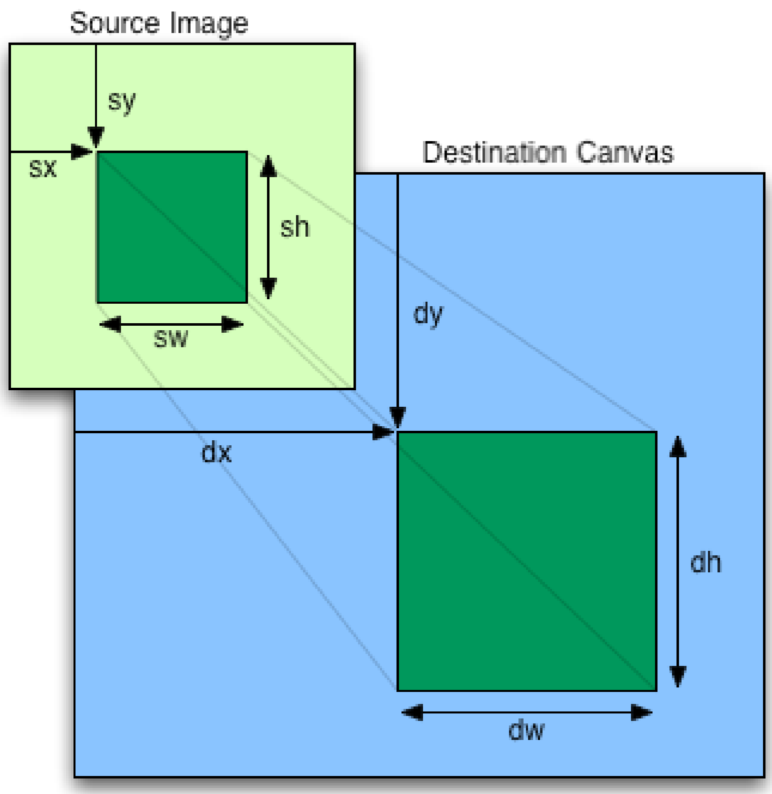
Example: video timeline viewer



```
frameCount = 25
framePosition = 25 % 16 = 9
frameX = (9 % 4) * 100 = 100
frameY = (Math.floor(9 / 4)) * 75 = 150
ctx.drawImage(video, 0, 0, 400, 300, 100, 150, 100, 75)
```

Canvas: drawImage

```
cxt.drawImage(image, dx, dy)
cxt.drawImage(image, dx, dy, dw, dh)
cxt.drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)
```



- The first argument can be an image, a canvas or a video element
- When a canvas uses a video as an input source, it draws only the currently displayed video frame
  - Canvas displays will not dynamically update as the video plays
  - If you want the canvas content to update, you must redraw your images as the video is playing

Example: video timeline viewer

- Function startVideo: updates the timeline frames regularly
  - The startVideo() function is triggered as soon as the video has loaded enough to begin playing

```
function startVideo() {
  // only set up the timer the first time the video starts
  if (videoStarted) return;
  videoStarted = true;
  // calculate an initial frame, then create
  // additional frames on a regular timer
  updateFrame();
  intervalId = setInterval(updateFrame, updateInterval);
  ...
```

- setInterval: calls a function repeatedly, with a fixed time delay between each call to that function

```
var intervalID = window.setInterval(func, delay);
```

Example: video timeline viewer

- Function startVideo: handles user clicks on the individual timeline frames

```
    // set up a handler to seek the video when a frame
    // is clicked
    var timeline = document.getElementById("timeline");
    timeline.onclick = function(evt) {
      var offX = evt.layerX - timeline.offsetLeft;
      var offY = evt.layerY - timeline.offsetTop;
```

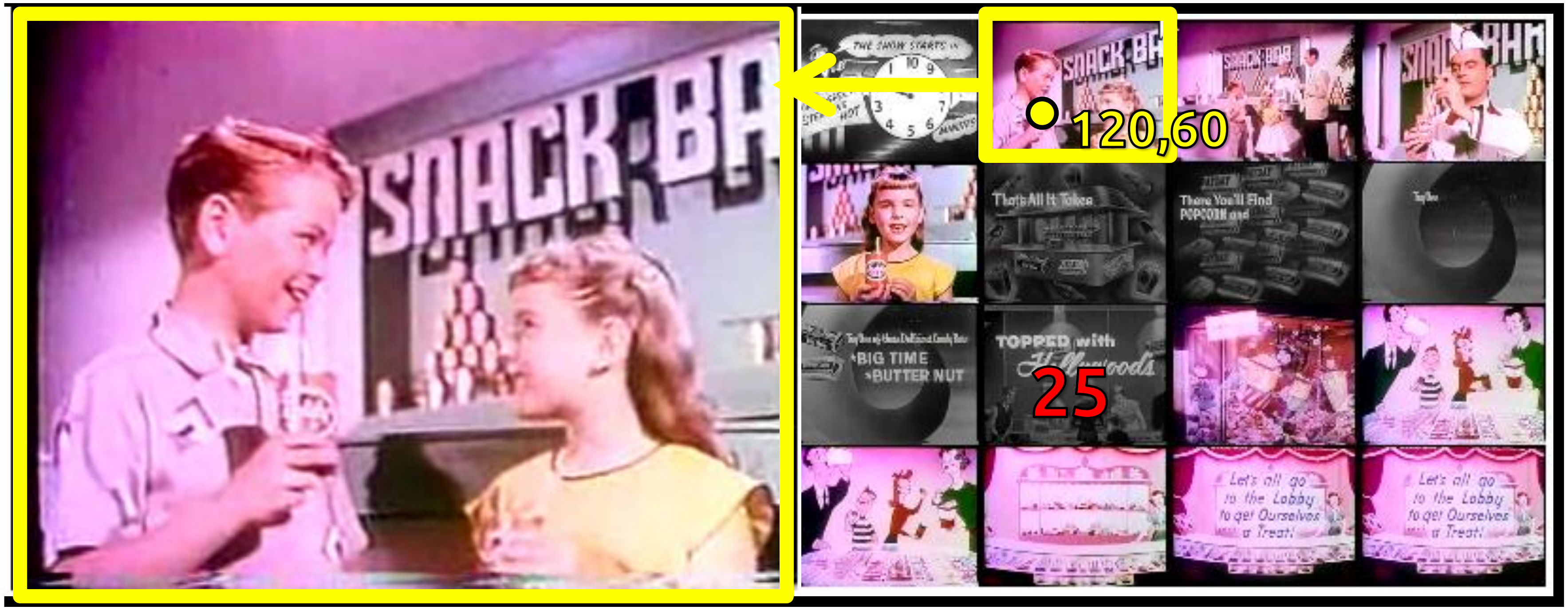
- offsetLeft: returns the number of pixels that the upper left corner of the current element is offset to the left within the parent node
- offsetTop: returns the distance of the current element relative to the top of the parent node
- layerX: returns the horizontal coordinate of the event relative to the current layer
- layerY: returns the vertical coordinate of the event relative to the current layer

Example: video timeline viewer

```
// calculate which frame in the grid was clicked
// from a zero-based index
var clickedFrame = Math.floor(offY/frameHeight)* frameRows;
clickedFrame += Math.floor(offX/frameWidth);
// find the actual frame since the video started
var seekedFrame = (((Math.floor(frameCount/frameGrid))*
  frameGrid) + clickedFrame);
```

- The clicked frame should be only one of the most recent video frames, so seekedFrame determines the most recent frame that corresponds to that grid index

Example: video timeline viewer



```
offX= 120
offY= 60
clickedFrame = Math.floor(60/75)* 4 = 0
clickedFrame += Math.floor(120/100)= 1
seekedFrame = (((Math.floor(25/16))* 16) + 1 = 17
```

Example: video timeline viewer

- Function startVideo: handles user clicks on the individual timeline frames

```
      // if the user clicked ahead of the current frame
      // then assume it was the last round of frames
      if (clickedFrame > (frameCount%16))
        seekedFrame -= frameGrid;
      // can't seek before the video
      if (seekedFrame < 0) return;
      // seek the video to that frame (in seconds)
      var video = document.getElementById("movies");
      video.currentTime = seekedFrame * updateInterval / 1000;
      // then set the frame count to our destination
      frameCount = seekedFrame;
    }
  }
```

Example: video timeline viewer

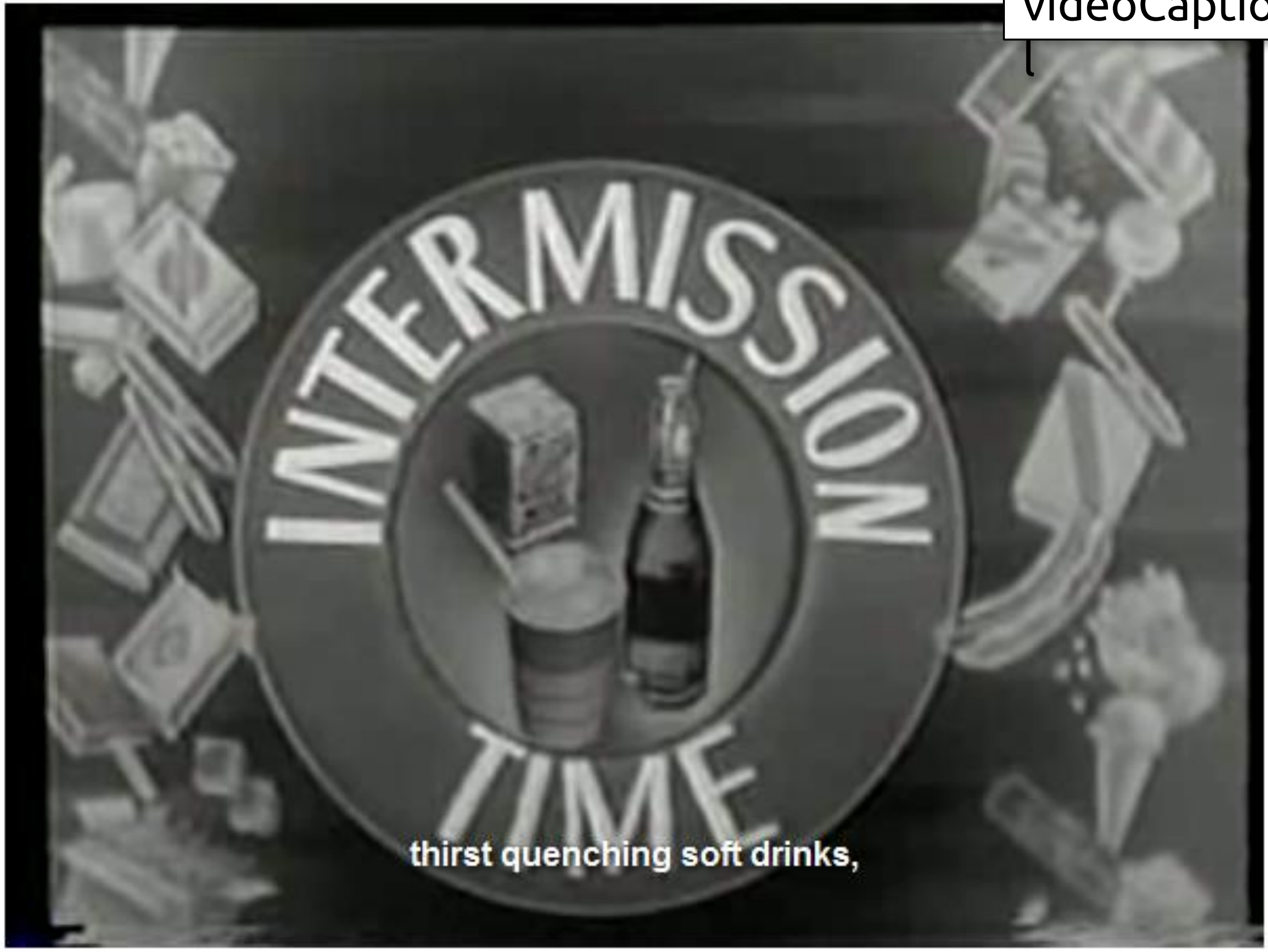
- Function stopTimeline: stops capturing frames when the video finishes playing
  - The stopTimeline handler is called when the “onended” video handler is triggered, i.e. by the completion of video playback.

```
 // stop gathering the timeline frames
function stopTimeline() {
  clearInterval(intervalId);
}
```

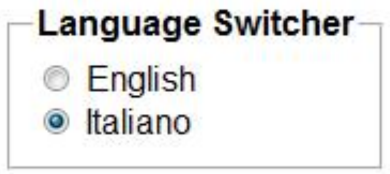
- clearInterval: cancels repeated action which was set up using setInterval()

Video with JavaScript synchronised captions

videoCaption.htm

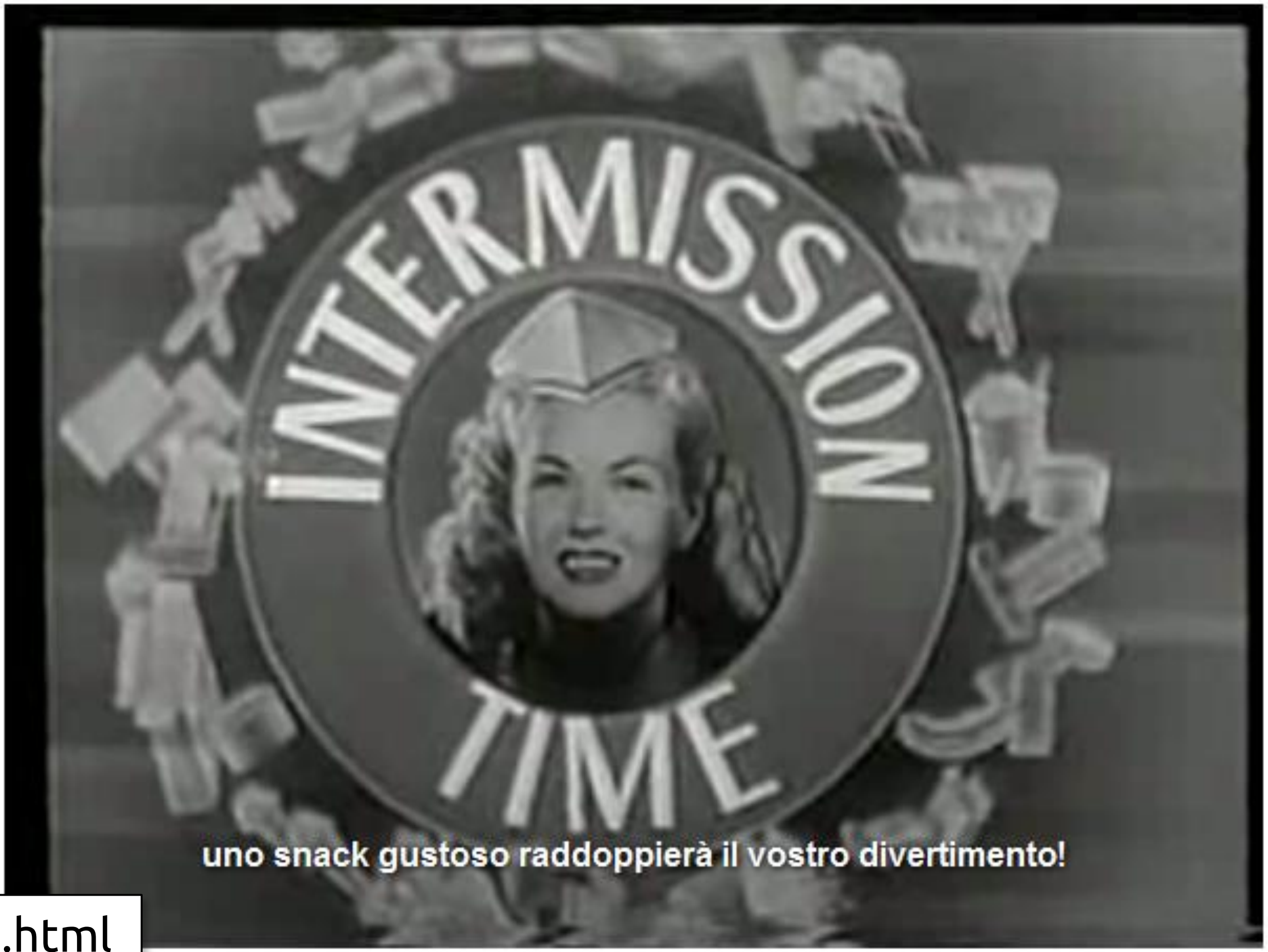


Multilingual synchronized captions



Language Switcher

- English
- [✓ Italiano]



uno snack gustoso raddoppierà il vostro divertimento!

videoCaption-lang.html

Other synchronization examples

- http://chirls.com/2011/01/13/what-im-working-on-synchronized-videos-in-html5-featuring-ok-go/

References

License