Real Time Communications Conference and Expo at Illinois Tech

IEEE International Conference

WebCodecs

and the next generation of web media APIs

Chris Cunningham, Google
Paul Adenot, Mozilla
Bernard Aboba, Microsoft

Once Upon a Time… Before the Pandemic

Streaming and realtime communications technologies evolved in ***silos***, with distinct APIs and protocol stacks.

Streaming offered ***large scale*** but ***high delay***.

Realtime communications offered ***low delay***, but ***modest scale***.

The Pandemic: a Virtual Revolution

During the pandemic, Internet communications has become essential to survival for people and industries, compressing a decade of deployment and innovation into a few months. This revolution is ***all about*** virtualization.

Next generation Web media APIs

Overcome the “tyranny of OR”. Multi-threaded applications can now deliver ***both*** low-latency ***and*** large scale, through low-level access to building blocks:

- Capture
- Encode/Decode
- Transport
- Rendering

Next generation Web media APIs

- Capture
  - Media Capture and Streams Extensions
  - Mediacapture-transform
- Encode/decode
  - WebCodecs
  - MSEv2
- Transport
  - WebTransport (HTTP/3 over QUIC)
  - WebRTC data channel in Workers (SCTP/DTLS/UDP)
- Framework
  - WHATWG Streams
  - Web Assembly

The “Pipeline” Model

- Send

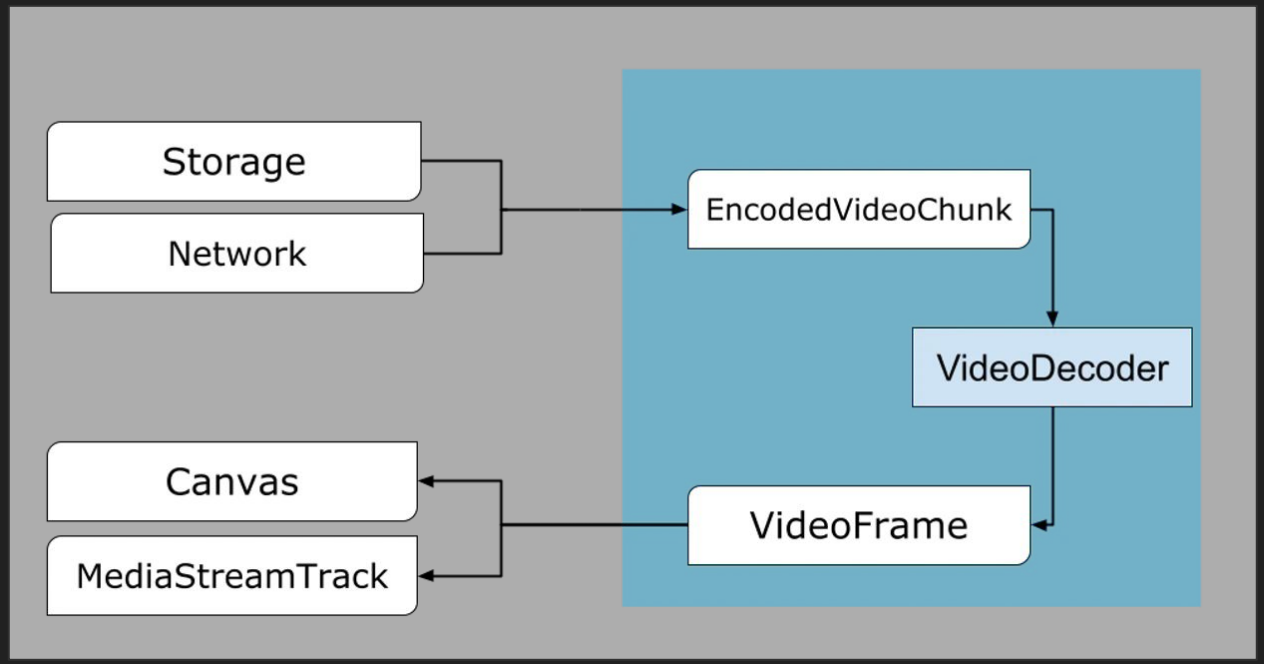
- Receive

Transport on the Web

<table>
<tr><th></th><th>Client-Server</th><th>Peer-to-peer</th></tr>
<tr><td>Reliable and ordered</td><td>WebSocket
(also WebTransport!)</td><td rowspan="3">RTCDataChannel
(SCTP/DTLS/UDP)</td></tr>
<tr><td>Reliable but unordered</td><td rowspan="2">WebTransport
(HTTP3/QUIC)</td></tr>
<tr><td>Unreliable and unordered</td></tr>
</table>

WebCodecs: Encoders & Decoders in JS

Video decoding flow



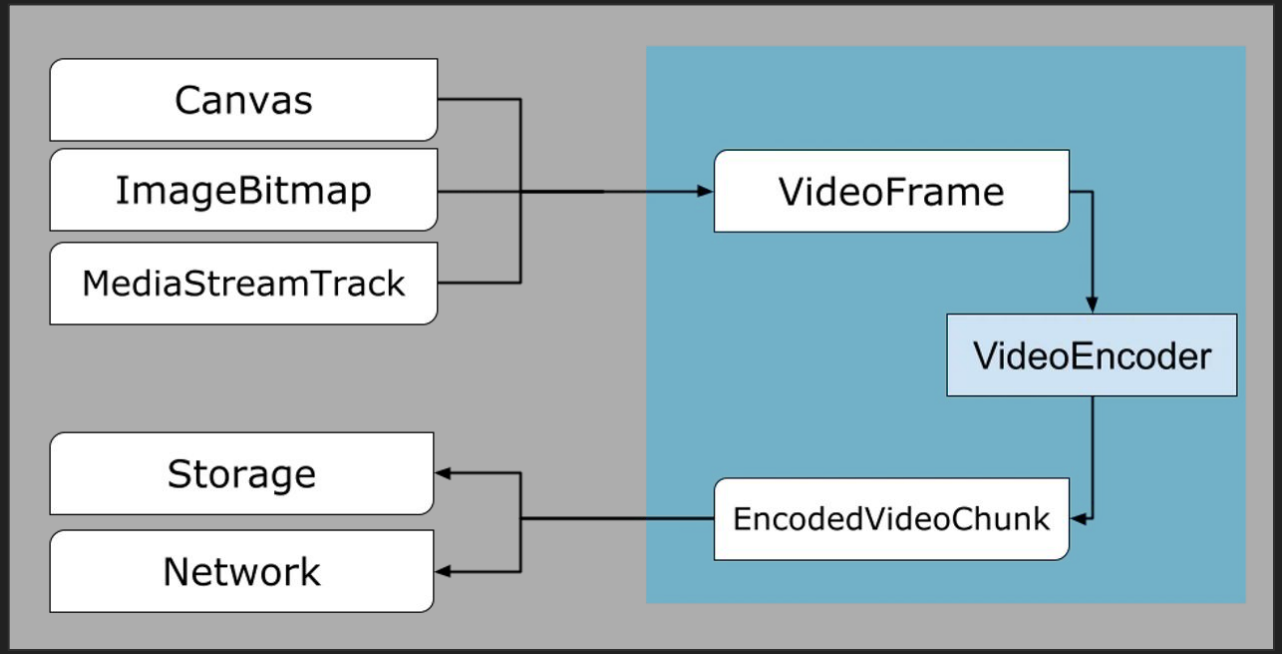
Build your own video renderer

https://chcunningham.github.io/wc-talk/rapid_video_painter.html

`AudioDecoder` looks very similar.

```
[Exposed=(Window,DedicatedWorker), SecureContext]
interface AudioDecoder {
  constructor(AudioDecoderInit init);

  readonly attribute CodecState state;
  readonly attribute long decodeQueueSize;

  undefined configure(AudioDecoderConfig config);
  undefined decode(EncodedAudioChunk chunk);
  Promise<undefined> flush();
  undefined reset();
  undefined close();

  static Promise<AudioDecoderSupport> isConfigSupported(AudioDecoderConfig config);
};

dictionary AudioDecoderInit {
  required AudioDataOutputCallback output;
  required WebCodecsErrorCallback error;
};

callback AudioDataOutputCallback = undefined(AudioData output);
```

Using Encoders looks very similar (just reversed).



Notes + FAQs

Close() your `VideoFrame` and `AudioData` objects ASAP!

Mandatory memory management

Use Workers

Keep your UI responsive and your codec well fed.

How to transfer streams to a worker? It looks like this…

```
streamWorker.postMessage(
  {
    type: "stream",
    config: config,
    url: url,
    streams: {input:  inputStream,
              output: outputStream
             }
  },
  [inputStream, outputStream]
);
```

What codecs are supported?

For Chrome:

- `VideoDecoder`: AV1, AVC (H.264), VP8, and VP9.
- `AudioDecoder`: AAC, FLAC, MP3, Opus, Vorbis, μ-law and A-law PCM formats.
- `VideoEncoder`: H264, VP8, VP9 (AV1 coming in Q4'21)
- `AudioEncoder`: Opus (AAC tentatively coming Q4'21)

Use `isConfigSupported()`

Details of support can be platform / device specific

What about WASM Codecs?

WASM codecs predate WebCodecs. The two can co-exist.

What containers are supported?

None. WebCodecs doesn't have a (de)muxing API.

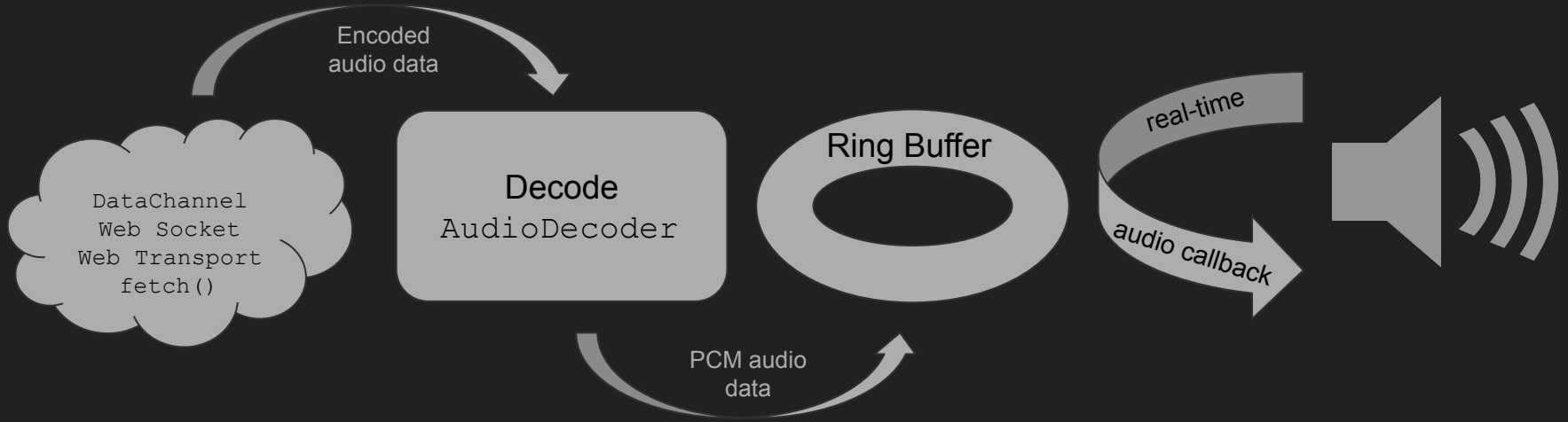
Build your own audio renderer

https://github.com/chcunningham/wc-talk/blob/main/simple_video_player.html

Build your own audio renderer

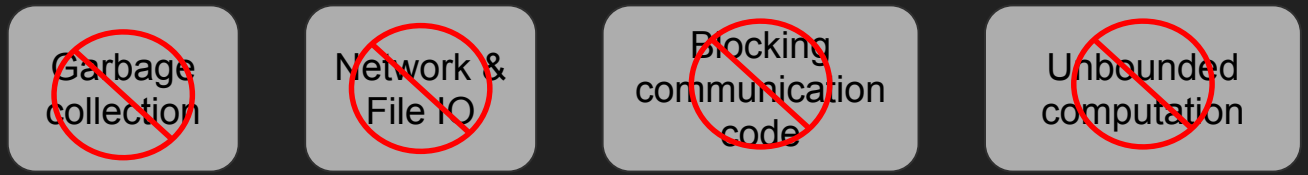
Three main steps, using an `AudioWorkletNode`:

1. Acquire and decode the encoded audio packets
2. Somehow send the decoded audio to the `AudioWorkletGlobalScope`
3. Render the audio frames



The `AudioWorklet` concept in two slides

Run arbitrary script on the high-priority real-time audio thread: the script needs to be **real-time safe**: only deterministic code is allowed, very few APIs are available



Works within an `AudioContext`, all the regular `AudioNode`s are available

Separate script file, then running in a separate global (similar to a `Worker`)

An AudioWorkletProcessor

Main thread: main.js

```
var ac = new AudioContext;
ac.audioWorklet.addModule('processo
r.js')
  .then(() => {
    var node =
      new AudioWorkletNode(ac,
'processor');
    node.port.onmessage = (m) => {
      console.log("received: " +
m);
    }
    node.port.postMessage("ping");
    node.connect(ac.destination);
  }
);
```

Real-time thread: processor.js

```
registerProcessor('processor',
  class Processor extends
AudioWorkletProcessor {
    constructor() {
      super();
      this.port.postMessage("ping");
      this.port.onmessage = () => {
        this.port.postMessage("pong");
      };
    }

    process(input, output, parameters) {
      /* White noise generator */
      for (let i = 0; i < 128; i++) {
        output[0][0][i] = Math.random()*2-1;
      }
      return true;
    }
  }
);
```

Real-time communication without generating garbage

`SharedArrayBuffer` is required, and allows writing lock-free data structures, such as an SPSC ring buffer, which is a data structure with just a few operations:

- `push(array)` enqueues PCM audio from a regular buffer into the ring buffer
- `pop(array)` dequeue PCM audio from the ring buffer into a regular buffer
- `available_write()` returns the numbers of free slots in the ring buffer
- `available_read()` return the number of elements readable from the buffer

Outline of an audio renderer: important concepts

1. The clock, for accurate audio/video synchronization
2. The pre-buffering, for a quick start
3. General operation: refilling the buffer during playback

Gettings accurate time for A/V synchronization

```
getMediaTime() {
    // Latency inherent to the OS and device in use
    let totalLatency = audioContext.outputLatency;
    // Latency of the Web Audio API implementation
    totalLatency += audioContext.baseLatency;
    return Math.max(audioContext.currentTime - totalLatency, 0.0);
 }
```

- Audio has the most robust clock: clocking the video to the audio clock is a good idea
- Range: As low as 5ms (macOS wired) up to 250-300ms (bluetooth device)
- https://blog.paul.cx/post/audio-video-synchronization-with-the-web-audio-api/

Planar vs. Interleaved, and playing the audio out

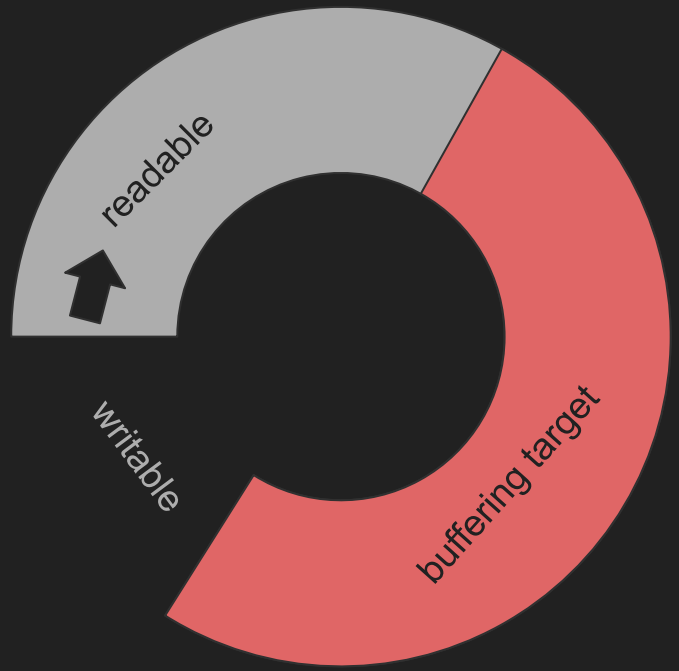
Planar audio, convenient for signal processing

Interleaved audio, convenient for IO

Filling the ring buffer

Goal: always have a few tens to a few hundreds of milliseconds of audio buffered.

While the media is not paused, periodically, check the status of the queue: if below the buffering target, kick off a few decoding operations to fill the ring buffer.



Final audio output

When the AudioWorkletProcessor's `process()` method is called :

- Dequeue just enough frames from the ring buffer (or else, an underrun happened, tune the ring buffer size and buffering target) to fulfill the request (for now 128 * number of channels)
- Deinterleave into the second parameter's planar buffers

The "Pipeline" Model

https://webrtc.internaut.com/wc/wtSender/

The “Pipeline” Model

- Send

```
inputStream
     .pipeThrough(SpecialEffects())
     .pipeThrough(EncodeVideoStream(config))
     .pipeThrough(Serialize())
     .pipeTo(Transport())
```

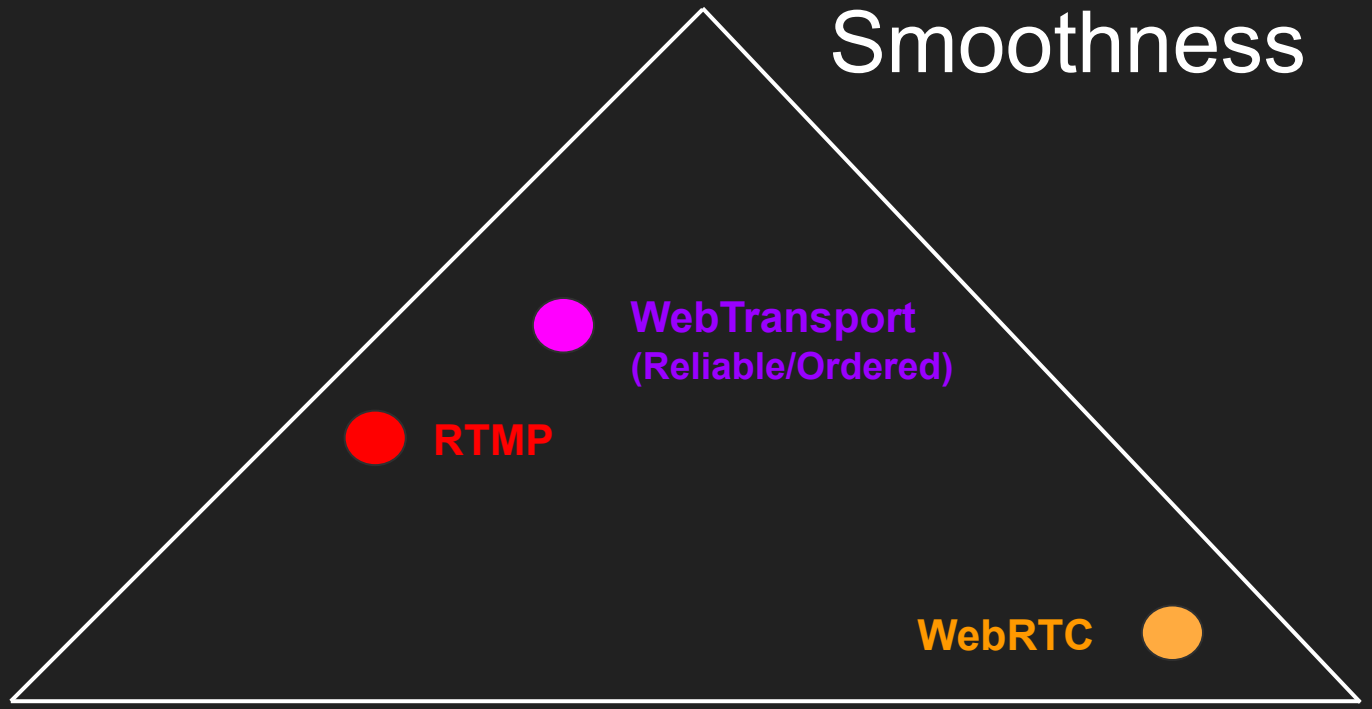
- Receive

```
receiveStream
     .pipeThrough(Deserialize())
     .pipeThrough(DecodeVideoStream())
     .pipeTo(outputStream)
```

inputStream & outputStream (getUserMedia and MediaStreamTrackProcessor/Generator)

```
async function getMedia(constraints) {
    // Get a MediaStream from the webcam.
    const mediaStream = await navigator.mediaDevices.getUserMedia(constraints);

    // Connect the webcam stream to the video element.
    document.getElementById('inputVideo').srcObject = mediaStream;

    // Create a MediaStreamTrackProcessor, which exposes a ReadableStream of VideoFrames.
    let [track] = mediaStream.getVideoTracks();
    let ts = track.getSettings();
    const processor = new MediaStreamTrackProcessor(track);
    inputStream = processor.readable;

    // Create a MediaStreamTrackGenerator, which generates a track from a
    // WritableStream of VideoFrames.
    const generator = new MediaStreamTrackGenerator({kind: 'video'});
    outputStream = generator.writable;
    document.getElementById('outputVideo').srcObject = new MediaStream([generator]);


...
}
```

Transport Tradeoffs



WebCodecs Decode as a Transform Stream

```
DecodeVideoStream() {
  return new TransformStream({
    start(controller) {
      this.decoder = decoder = new VideoDecoder({
        output: frame => controller.enqueue(frame),
        error: (e) => {
         self.postMessage({severity: 'fatal', text: `Decoder error: ${e.message}`});
        }
      });
    },
    transform(chunk, controller) {
      if (this.decoder.state != "closed") {
        this.decoder.decode(chunk);
      }
    }
  });
}
```

WebCodecs Encode as a Transform Stream - 1/3

```
EncodeVideoStream(config) {
  return new TransformStream({
    start(controller) {
      this.frame_counter = 0;
      this.pending_outputs = 0;
      this.encoder = encoder = new VideoEncoder({
        output: (chunk, cfg) => {
          if (cfg.decoderConfig) {
            self.postMessage({text: 'Decoder reconfig!'});
            decoder.configure(cfg.decoderConfig);
          }
          chunk.temporalLayerId = 0;
          if (cfg.temporalLayerId) {
            chunk.temporalLayerId = cfg.temporalLayerId;
          }
```

WebCodecs Encode as a Transform Stream - 2/3

```
    this.pending_outputs--;
        controller.enqueue(chunk);
      },
      error: (e) => {
        self.postMessage({severity: 'fatal', text: `Encoder error: ${e.message}`});
      }
    });
  VideoEncoder.isConfigSupported(config).then((encoderSupport) => {
      if(encoderSupport.supported) {
        this.encoder.configure(encoderSupport.config);
      } else {
        self.postMessage({severity: 'fatal', text: 'Config not supported:\n' + JSON.stringify(encoderSupport.config)});
        this.stopped = true;
      }
    })
    .catch((e) => {
        self.postMessage({severity: 'fatal', text: `Encoder error: ${e.message}`});
    })
  },
```

WebCodecs Encode as a Transform Stream - 3/3

```
transform(frame, controller) {
   if (this.pending_outputs <= 30) {
     if (++this.frame_counter % 20 == 0) {
       self.postMessage({text: 'Encoded 20 frames'});
     }
     this.pending_outputs++;
    const insert_keyframe = (this.frame_counter % config.keyInterval) == 0;
     try {
       if (this.encoder.state != "closed") {
        this.encoder.encode(frame, { keyFrame: insert_keyframe });
       }
     } catch(e) {
      self.postMessage({severity: 'fatal', text: `Encoder error: ${e.message}`});
     }
   }
   frame.close();
      }
    });
```

Questions?