Ogvjs

Bringing free codecs to Safari and IE with Emscripten

Brion Vibber, Wikimedia

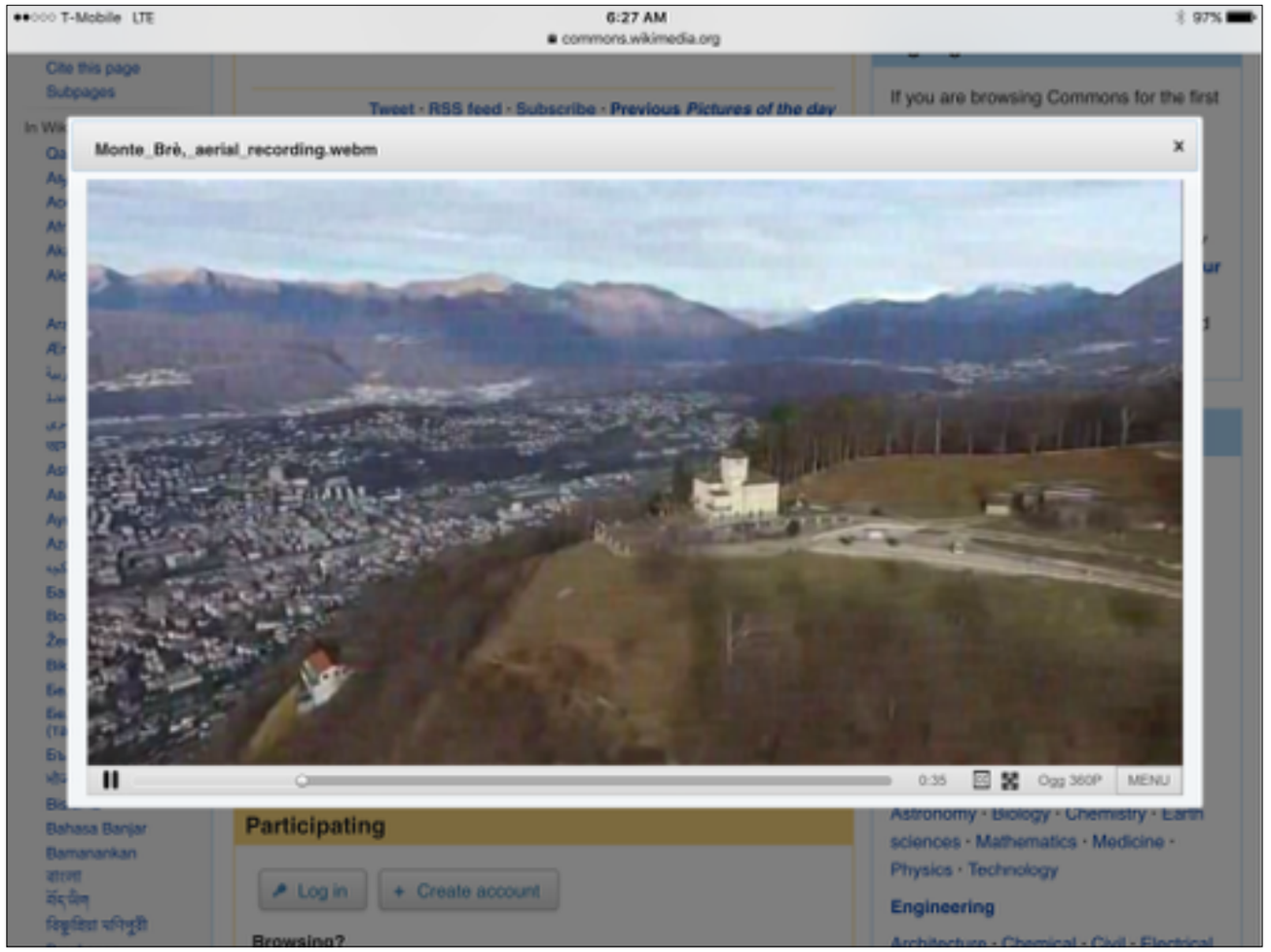
GStreamer Conference - 8 October 2015

WHAT?

- HTML5 audio/video stack implemented in JavaScript

So what.... Is it?? Roughly ogv.js is an HTML5/JavaScript implementation of the HTML5 media player stack, that sits alongside the native one and can be extended in ways the native stack cannot.

Ogg Vorbis & Opus audio and Theora video are fully supported, including seeking. WebM support is more experimental, with VP8 supported and VP9 on the agenda, but incomplete seeking and middling performance so far.



Nonetheless the Theora and Vorbis performance are good enough that we've gone ahead and deployed ogv.js as a fallback player backend for Wikipedia and Wikimedia Commons, engaged automatically only when needed.

WHY?

- Web app playback for older & newer formats
- Compatibility for Free license nerds!

Why would we do this? Well there's potential in adding support for less popular formats for specialty uses or emulation, but the main driver is patent licensing.

PATENT HELL

Remember when everybody was trying to avoid using GIF images because of the Unisys patent licensing? For the first few years of Wikipedia we only accepted JPEG and PNG over this! Well that war's not over in video and audio land...

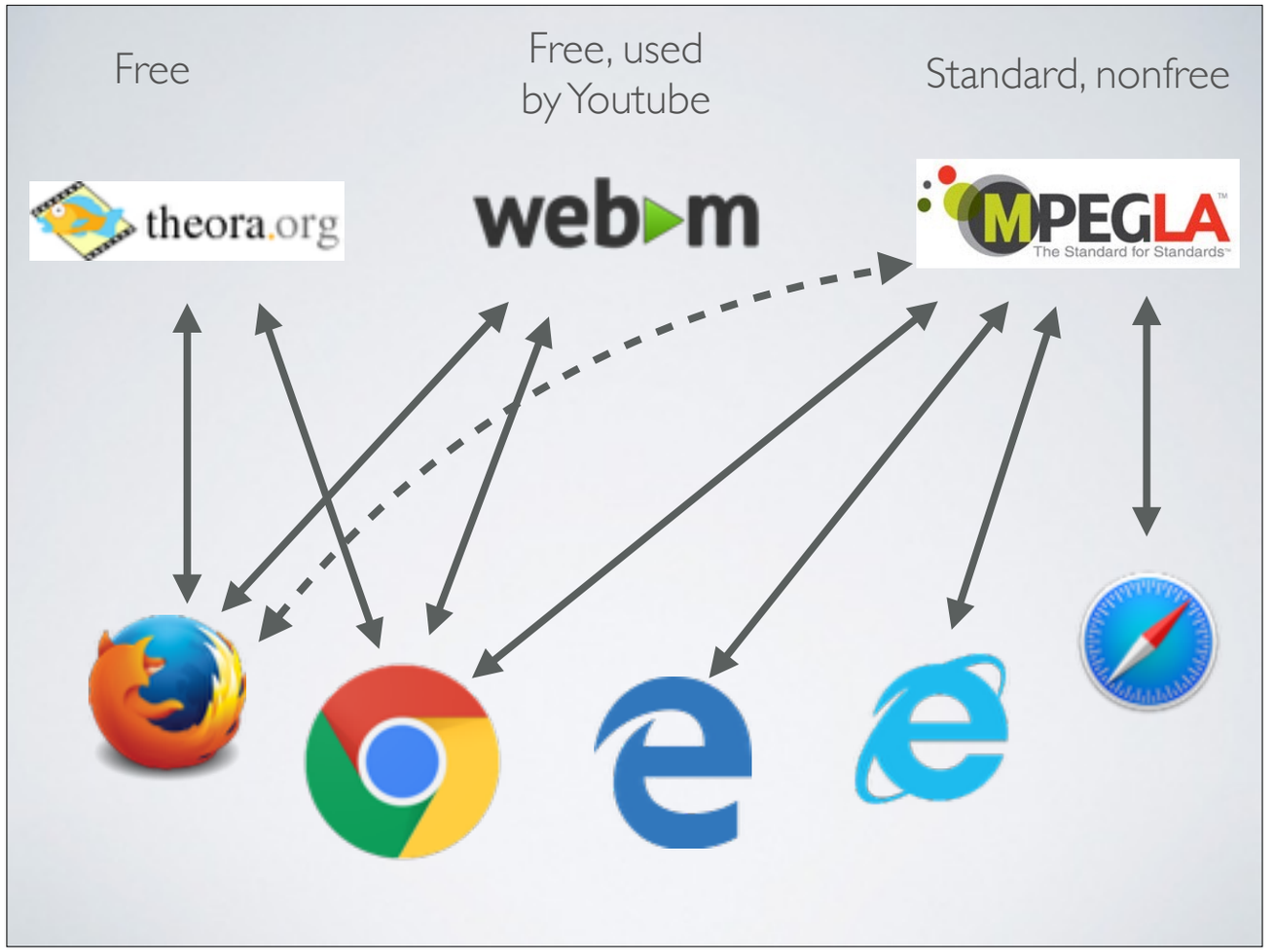
PATENT HELL

As many of you are familiar, the MPEG-LA patent licensing pools form a choke point for free software attempting to support MPEG-4 family formats, such as the widely used H.264 and AAC codecs.

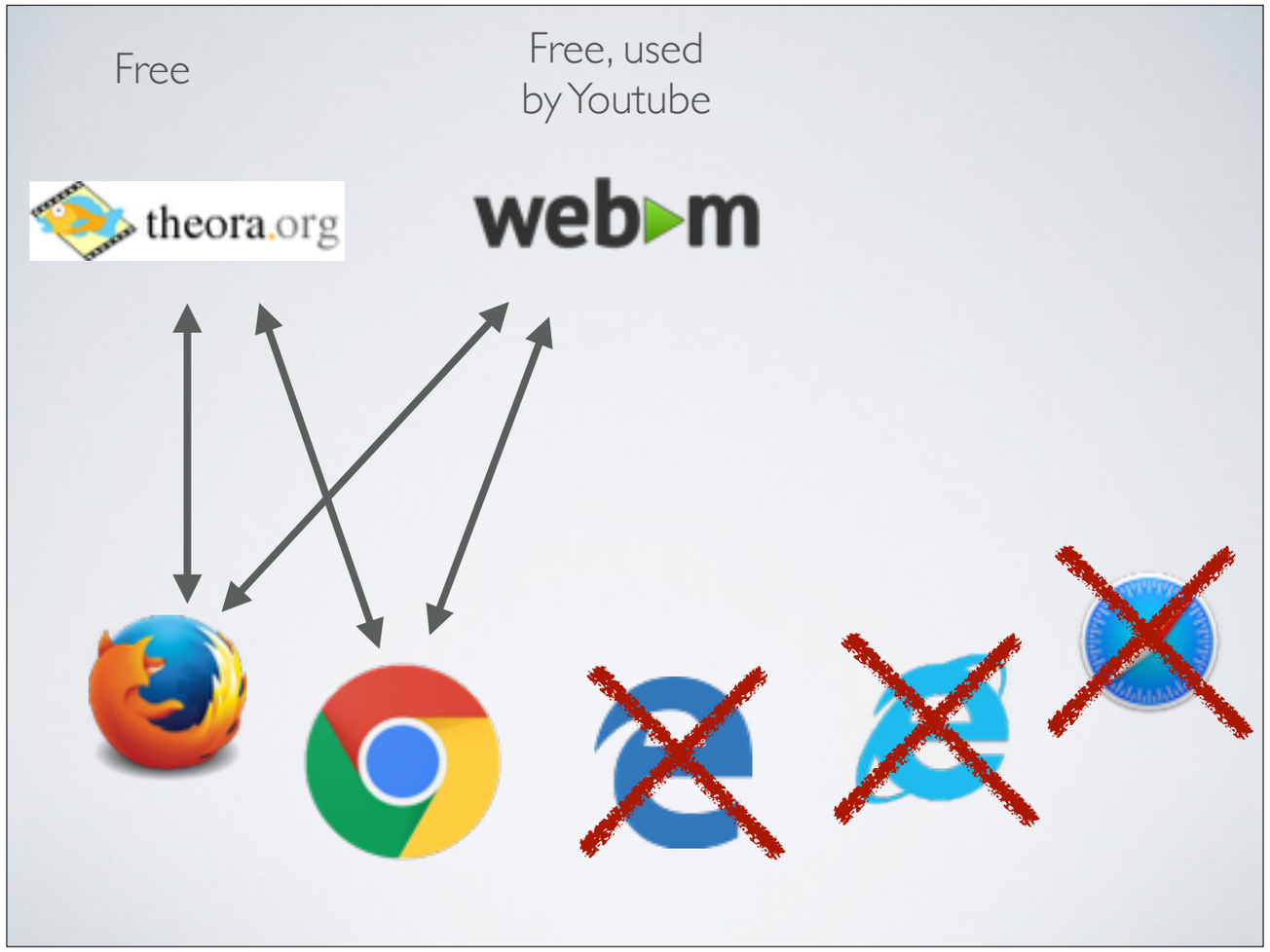
HISTORY'S MYSTERIES

- 2002: Wikipedia bans GIFs over patent concerns
- 2003: Wikipedia adopts Ogg Vorbis over MP3 due to patent concerns
- 2004: Wikipedia allows GIFs after key patent expiration
- 2007: Opera proposes video element. HTML5 draft spec briefly recommends Ogg Theora/Vorbis as base formats... Microsoft, Apple, Nokia force dropping of base format
- 2008: Wikipedia adopts Cortado Java applet as fallback for Ogg media
- 2010: Google announces WebM VP8
- 2011: Wikipedia produces WebM and Ogg transcodes
- 2014: Wikipedia community rejects MP4 support proposal
- 2015: Alliance for Open Media; Microsoft announces future VP9 support

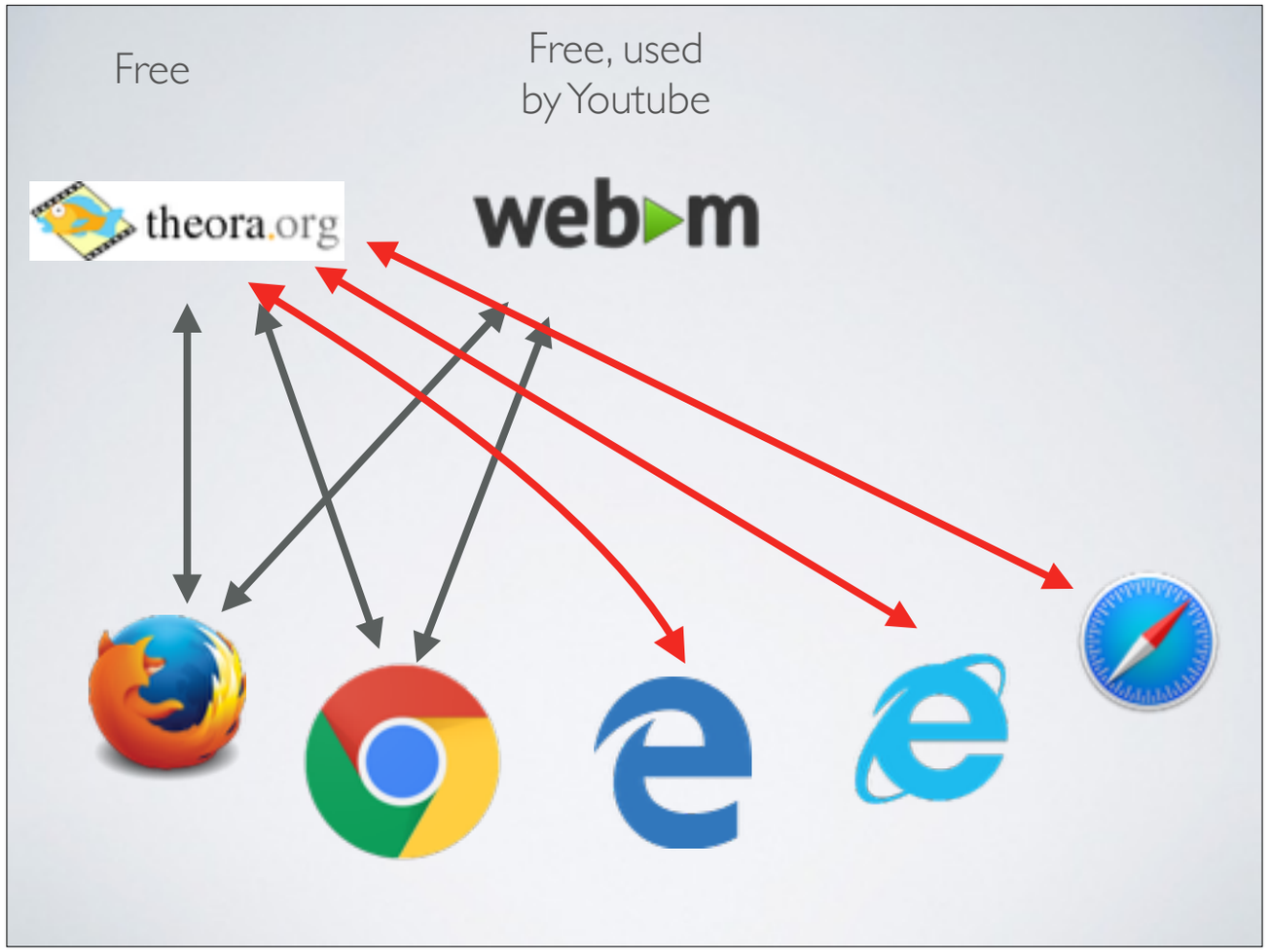
This is hardly a new problem; the browser makers spent years fighting over formats.



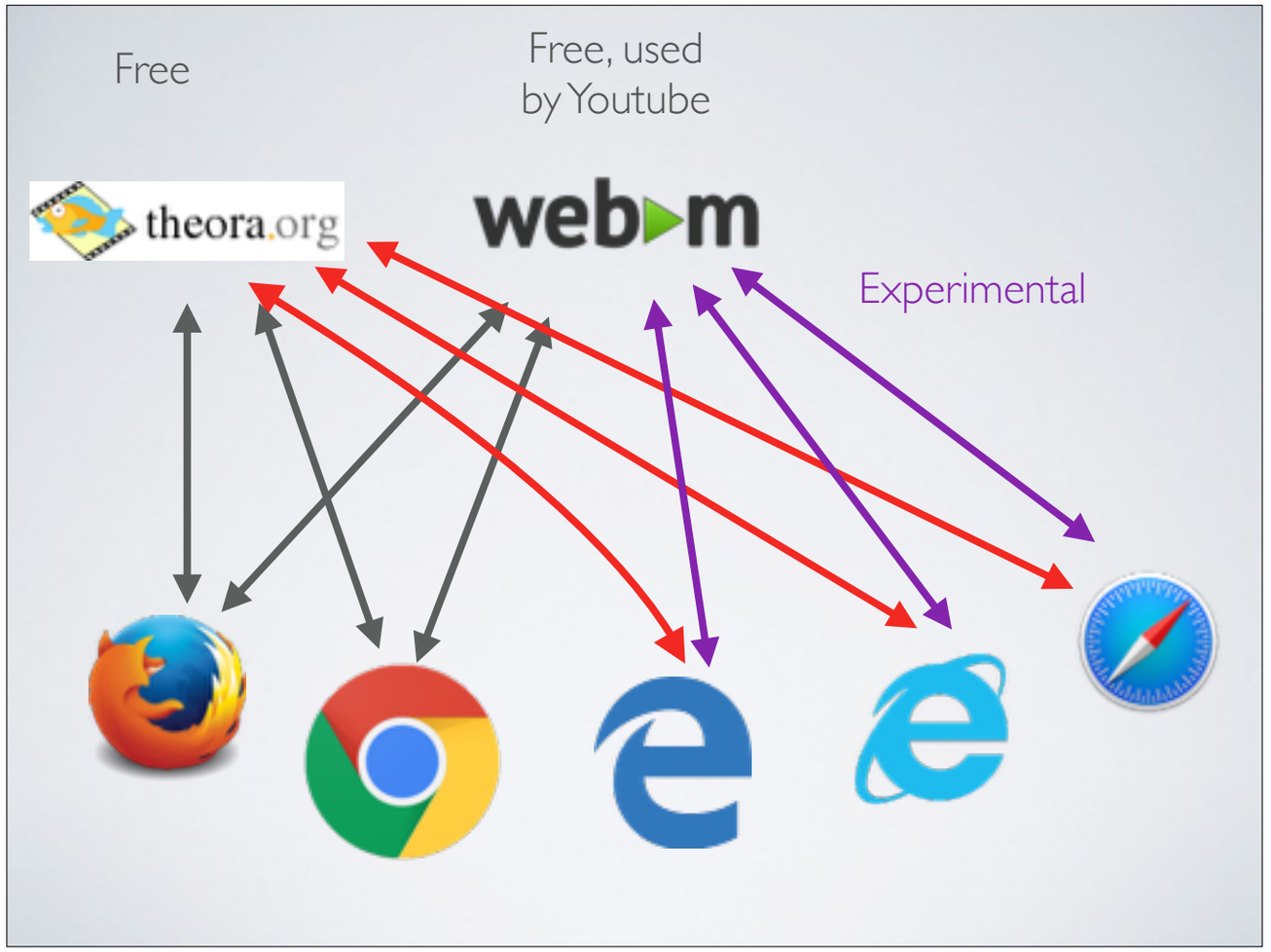
Today the world of browser media support is still inconsistent, with almost every browser supporting H.264 but only some supporting WebM, and a few still carrying on Theora support. This makes your life easy if you have a suitable license from MPEG-LA, unless you want to support Linux clients ...



If you can't do H.264/AAC then your life becomes freer but much more complicated. Until recently we basically had no good media playback solution for iE and Safari at Wikipedia because of our format licensing policies.



Wikipedia has now deployed ogv.js to provide Ogg Vorbis and Theora playback, reaching today's IE 10/11, Edge, and Safari browsers in addition to native WebM playback on Firefox and Chrome.



As WebM support improves, depending mainly on performance, we hope to roll out VP8 or VP9 video through ogv.js as well, providing better picture quality than the older Theora can achieve. Edge will also get native VP9 support somewhere down the line.

ANTECEDENTS

CORTADO

Theora, Vorbis in Java applet

The Cortado Java applet provided Vorbis and Theora playback, with adequate performance on desktop machines for the small low resolution clips we had in the late 2000s. However it was hard to maintain, with Java forks of the decoder libraries and a partial port of GStreamer. Java applets also became more user unfriendly over time, and Java itself is now on the outs in the browser world, with no mobile support, poor desktop market share, and a trickier applet signing process. Eventually we realized our copy of the applet no longer worked, and nobody wanted to try to fix it.

FVORBIS DEMO

Vorbis in Flash 10

In 2008 I stumbled on a demo Vorbis player in Flash, written in haXe and compiled to ActionScript 3 byte code. Like Cortado this involved porting libraries and didn't get much maintenance but it proved that a scripting language could be fast enough to do multimedia decoding.

BROADWAY

H.264 in JS
No audio

Some experimental Mozilla projects out there included Broadway, a JS playback engine for H.264 video meant to run in Firefox which at the time did not support it. It used emscripten for the decoder, with a JS MP4 demuxer and WebGL acceleration for YUV conversion but never quite finished audio support. It also didn't solve the license issues for sites that didn't have an mpegla license...

ROUTE9

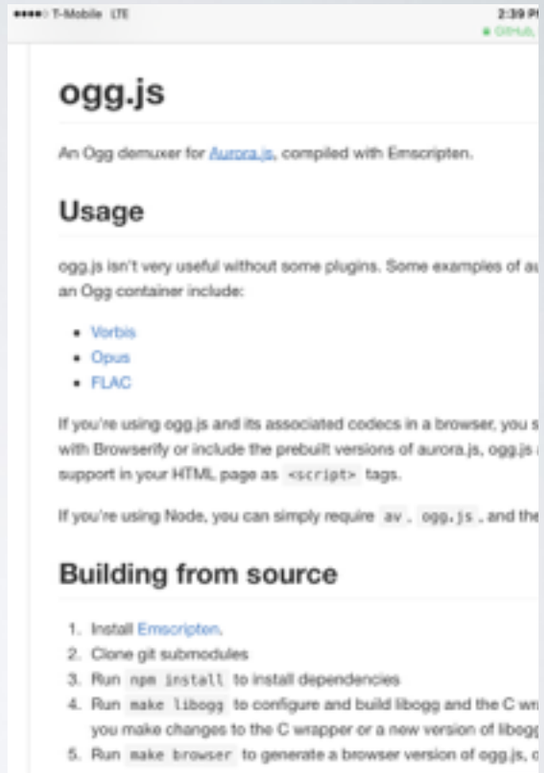
WebM VP8 in JS
No audio

Someone put together a fork of Broadway called Route 9 that played back WebM VP8 video; like Broadway it never finished audio support and was too slow.

OGG.JS

Vorbis, Opus audio for Audiocogs/Aurora.js

No video

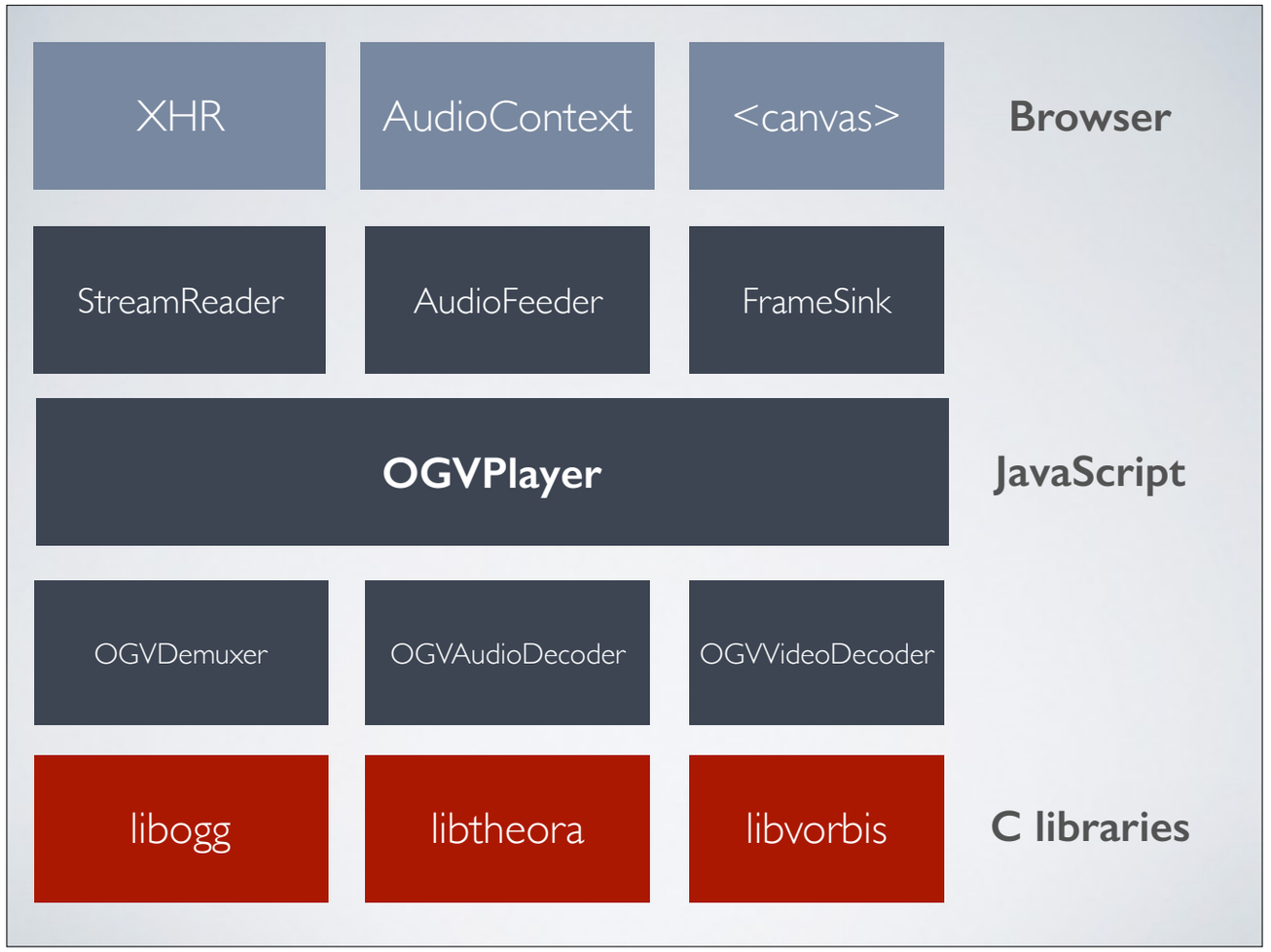


On the audio side, in 2013 I stumbled on Ogg.js, an audio-focused plugin adding Vorbis and Opus support to the audiocogs JS audio framework. This became my jumping off point for build scripts -- in fact the ogv.js git repo is a fork from Ogg.js, with very little original code left!

HOW?

C -> clang + LLVM + emscripten -> JS

Rather than maintaining forks of the decoder libraries, I thought it best to follow the lead of newer projects like Broadway and use emscripten with the battle-tested C libraries that everyone else ships: libogg, libvorbis, libtheora, libvpx. The emscripten cross compiler suite uses the clang front end C compiler and a custom LLVM backend to produce fairly portable JavaScript code which can be pretty well optimized in modern JITing JS engines.



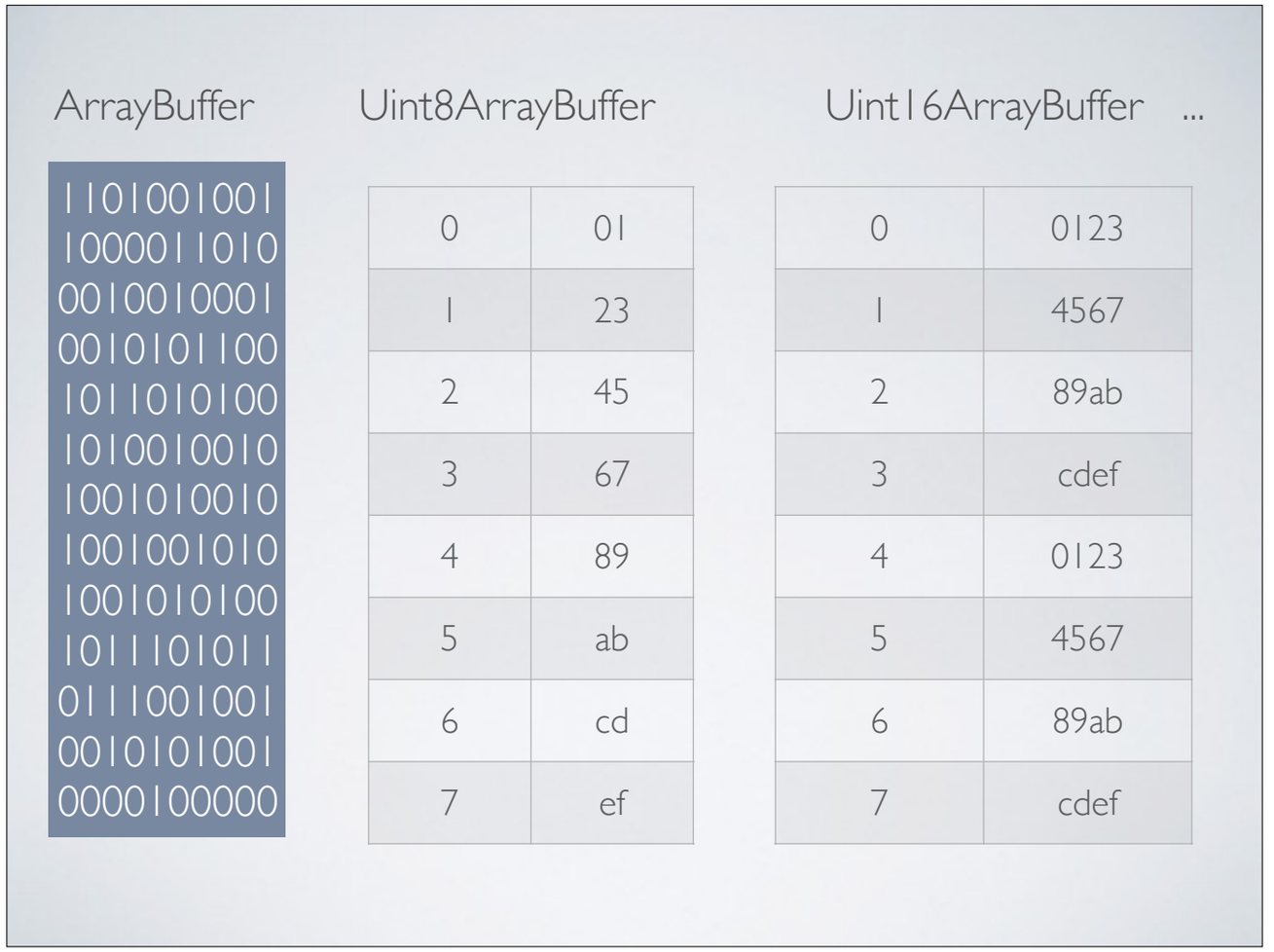
The higher levels of the ogv.js stack abstract around the HTML5 JS environment, providing a consistent interface for seekable HTTP steaming, playing audio output, and drawing YUV frames to a canvas. Lower levels wrap around the emscripten'd C libraries for demuxing and decoding data. In the middle, OGVPlayer class provides the playback logic and an HTMLMediaElement compatible API.

	JavaScript	C
Objects	Opaque refs, GC	Heap, pointers
Numeric types	Float64	Int 8/16/32, Float 32/64, & SIMD types
Threads	Separate memory	Shared memory

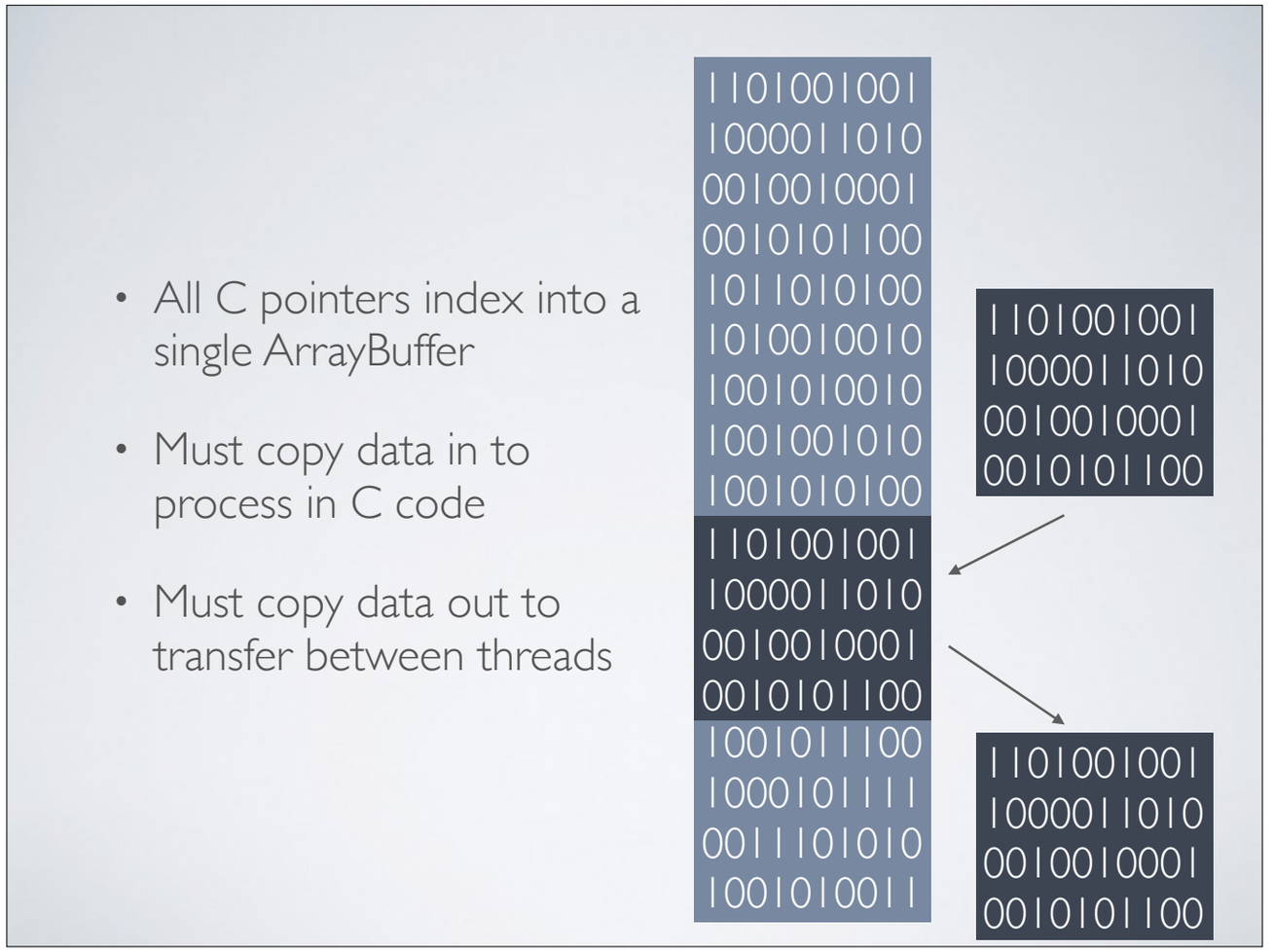
So how does this cross compiler work in the first place? At first glance, JavaScript and C are horribly mismatched. JS has garbage collected objects and only one floating point numeric value type; C has numerous numeric types, raw pointers everywhere, and allows all kinds of weird side effects which code may rely on. On top of that, threading in JS is limited, more like separate processes in C.

	Emscripten
Objects	ArrayBuffer as heap
Numeric types	TypedArray views and coercion
Threads	SharedArrayBuffer (future)

The magic comes in JS's typed arrays system, introduced originally to go with WebGL which required more low level data arrays than were previously available. Combined with explicit type coercion to ensure that JIT compilers optimize correctly, this provides a very C-like environment.



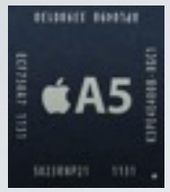
The core of the emscripten runtime is a single ArrayBuffer object, which is a fixed size byte vector. Typed Array View objects can be created as aliases of the buffer, allowing the same memory region to be accessed at a byte level, or as larger int and float types with native endianness.



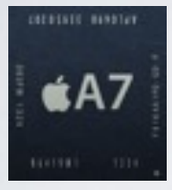
There is a big limitation though: because every C pointer is an index into that one buffer, data that comes in from other ArrayBuffers must be copied in before it can be processed. When getting data out sometimes you can use a direct aliased typed array view, but to transfer between worker threads you have to copy again, or you'll end up coming the entire heap at once.

But how's it actually perform?

THEORA ON SAFARI



iPhone 4s: 160p or audio



iPhone 5s: 360p+

MacBook Pro 2010: 480p+

MacBook Pro 2015: 1080p

Currently the main deployment target is Theora video and Vorbis audio on Safari. Audio runs adequately on even relatively slow armv7 devices like an old iPad 3 or iPhone 4s. Theora is a bit too sluggish on these, with stuttery output even at low-res 160p. But on an arm64 device like the iPhone 5s or iPad Air, 360p or even 480p run well. Newer devices should do even better. Desktops run well, with up to 1080p on a fast MacBook Pro.

INTERNET EXPLORER

- Older slower JIT; much slower than Safari on same machine
- Flash audio shim

We get middling performance out of IE 10/11, bit usually can still pump out 360p on a reasonably fast machine. 10 lacks WebGL, and both lack web audio, so there is additional output conversion overhead. The JIT compiler is also just not as good as newer browsers, and memory comes have more overhead than in other engines.

MS EDGE

- About 2x faster than IE 11
- another 2x coming with asm.js optimizations
- Native Web Audio
- Native VP9, Opus in development!

Microsoft Edge on Windows 10 runs even faster than IE, with a much better JIT, good WebGL and native Web Audio. It even works on Windows 10 Mobile, though I have only tested it on a relatively slow device so far. Edge is getting new asm.js optimizations that double the speed, as well! Eventually Edge should also get native VP9 and Opus or Vorbis support, and we'll be able to use that instead of ogv.js for playback.

WEBM VP8

- About 8X slower than Theora due to additional decode complexity
- Demuxer doesn't yet support seeking
- Too slow on mobile so far...

WebM with VP8 video is about 8x slower than Theora.the codec is more computationally complex, and is more reliant on SIMD acceleration in hot spots, which cannot yet be done in JS. VP9 has not yet been tested but is likely to be still slower.

CAPABILITIES FUTURE

- Media Source Extensions API - base for adaptive streaming
- Encoding
- Easier demux & codec plugins

PERFORMANCE FUTURE

- SIMD - requires browser support
- Threaded decoder - requires browser support or deep library hacks
- WebGL shader acceleration - requires deep hacks?
- Faster loading - web assembly?

END

https://github.com/brion/ogv.js