

Reaching the Multimedia Web from Embedded Platforms with WPEWebKit

Philippe Normand
Embedded Linux Conference 2021



Talk outline

- Intro
- WPE architecture overview
- Supported W3C specifications
- Cog
- WPE on Embedded platforms with Yocto
- Using WPE in GStreamer Multimedia applications

Who am I

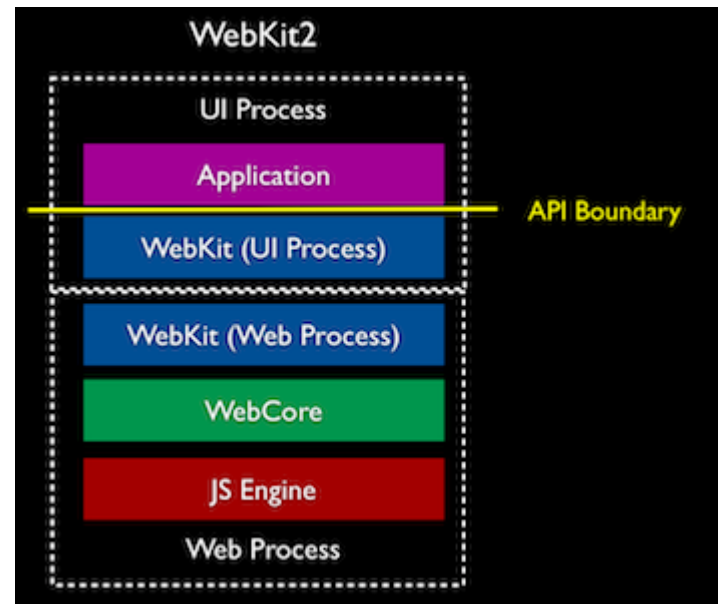
- WebKit committer and reviewer
- GStreamer committer
- Partner at Igalia:
 - Worker-owned coop, currently around 110 happy Igalians around the world
 - Provides consulting services for various Free Software projects

WPE architecture overview

What is WebKit

- WebEngine aimed for embedding HTML/CSS/JS in native applications
- Forked from KHTML by Apple in 2004
- Powers Safari, but also dozens of applications on various platforms
- APIs provided by WebKit ports

Multi-process!



WPE, a decoupled WebKit port

- Upstream in webkit.org
- 6 months release cycle, security updates
- Not tied to any widget toolkit
- Rendering and input events handling via loadable backends

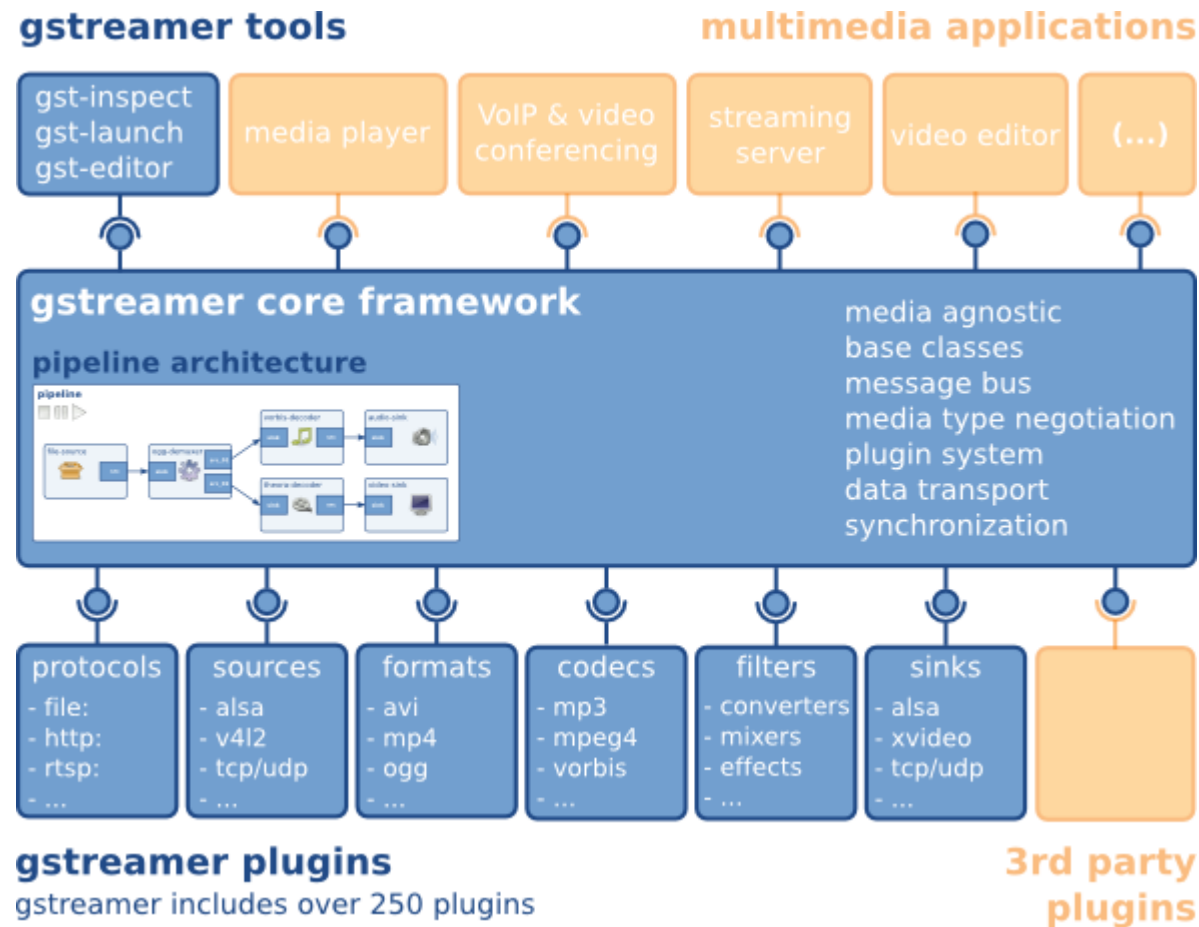
Rendering backends

- Various implementations, most notably:
 - FDO backend
 - RDK backend (various devices supported under the RDK/Comcast umbrella)

WPEBackend-FDO

- Depends on EGL, usually provided by Mesa or binary drivers
- High level API provided for applications (browsers, but also other apps!)
- Recommended by upstream WPE community

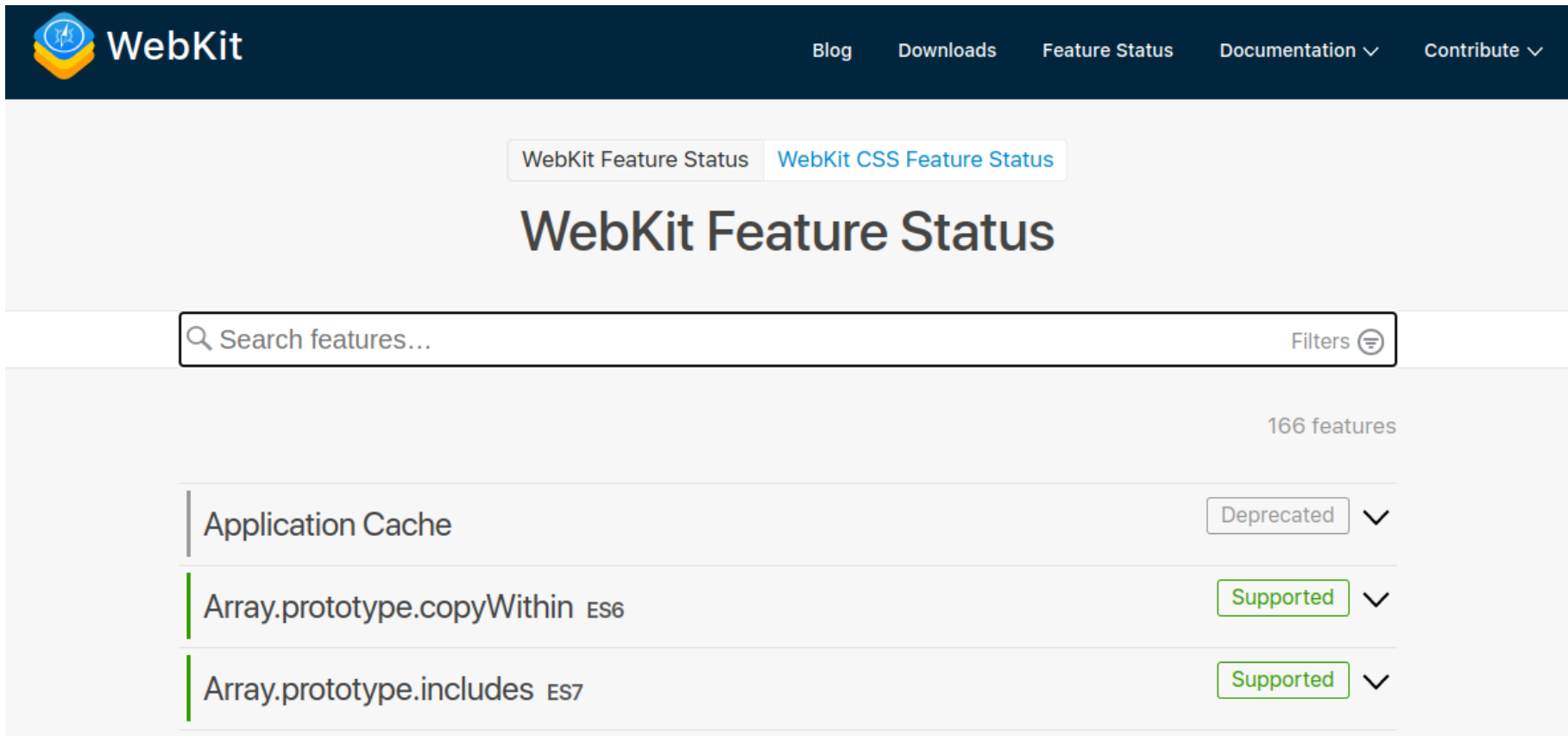
GStreamer



W3C Specs

Specs, specs everywhere

<https://webkit.org/status/>



The screenshot shows the WebKit website's 'Feature Status' page. The header includes the WebKit logo and navigation links for Blog, Downloads, Feature Status, Documentation, and Contribute. Below the header, there are tabs for 'WebKit Feature Status' and 'WebKit CSS Feature Status'. The main heading is 'WebKit Feature Status'. A search bar with the placeholder 'Search features...' and a 'Filters' button are present. Below the search bar, it indicates '166 features'. A table lists features with their status: 'Application Cache' is 'Deprecated', while 'Array.prototype.copyWithin ES6' and 'Array.prototype.includes ES7' are 'Supported'. Each status is in a colored box with a dropdown arrow.

Feature	Status
Application Cache	Deprecated
Array.prototype.copyWithin ES6	Supported
Array.prototype.includes ES7	Supported

MediaSource Extensions

- Adaptive streaming for the Web
- Widely used by most major video streaming platforms (Youtube, etc)
- Can ingest DASH (with DASH.js)
- WPE: enabled by default at build and runtime
- Spec: <https://w3c.github.io/media-source/>

MSE in WPE

- Chunks queued from JavaScript world to a SourceBuffer
- One GStreamer WebKit Append pipeline per SourceBuffer
 - Demuxing and parsing of samples
 - Samples stored at WebCore's MSE layer
- Playback pipeline using a dedicated MediaPlayerPrivate implementation
 - Playbin3-based
 - Custom source element

Encrypted Media Extensions

- Protected media content playback
- Used by most video streaming platforms (Netflix, etc)
- WPE: disabled by default due to runtime requirements (Content-Decryption-Module)
- CDMs integrated on per-product basis in WPE
- Spec: <https://www.w3.org/TR/encrypted-media/>

EME in WPE

- Protected content signaled from demuxers to WPE through GStreamer Protection events
- WPE probes for supported platform CDMs through the OpenCDM API
- Two CDM runtimes supported:
 - RDK Thunder
 - Sparkle-CDM
- Content decryption in WPE through custom GStreamer decryptors

MediaCapabilities

- Media engine decoding/encoding capabilities probing
- WPE: enabled by default at build time, disabled at runtime
- Spec: <https://w3c.github.io/media-capabilities/>

MediaCapabilities in WPE

- Probing implemented through a GStreamer registry scanner
- Runtime identification of encoders, decoders, (de)muxers
- WebKit mimetype <-> GStreamer caps

WebAudio

- Low-latency audio pipelines
- Examples of use: games, music creations apps, any app requiring audio feedback for user interactions
- WPE: enabled by default at build and runtime
- Spec: <https://www.w3.org/TR/webaudio/>

WebAudio in WPE

- Decoding pipeline relying on GStreamer `decodebin`
- Playback pipeline sourcing WebKit audio samples from custom GStreamer source element

Media Capture & Streams

- Access to Webcam/microphone/screencasting
- Used by WebRTC apps mostly (Jitsi, etc)
- WPE: enabled only in developer builds, plans for enabling by default in WPE 2.36
- Spec: <https://www.w3.org/TR/mediacapture-streams/>

Media Capture & Streams in WPE

- Capture device management with `GstDeviceMonitor`
- One capture pipeline per device, feeding one-to-many `appsinks` for consumption by WebKit
- Rendering by MediaPlayer through custom mediastream GStreamer source element
- Screencasting through PipeWire

WebRTC

- Real-Time P2P communication for browsers
- Used for audio/video chatting, low-latency one-to-many broadcasting, ...
- WPE:
 - LibWebRTC enabled only in developer builds
 - Unusable in releases (LibWebRTC is not bundled in tarballs + licensing issues due to BoringSSL)
 - Plans for a GStreamer-WebRTC backend in WPE 2.36
- Spec: <https://www.w3.org/TR/webrtc/>

MediaStream recording

- Use-case: mostly WebRTC calls recording
- Not enabled in WPE yet. WIP implementation:
 - Based on the new `GstTranscoder` GStreamer library (1.20)
 - Makes use of existing internal WPE WebRTC video encoder GStreamer element
- Spec: <https://w3c.github.io/mediacapture-record/>

Cog, the official WPE-based browser

- Minimalistic design leveraging platform renderer modules:
 - Wayland, X11 (!), GTK4
 - DRM, Headless
- Auto-probing of platform for renderer selection
- Single web view, for now
- Can be controlled through DBus

Running Cog without Wayland compositor

- App use-cases: Kiosks, Set-Top-Box UIs, ... any fullscreen display
- Wayland buffers (or DMABufs) imported as GBM Buffer objects
- Rendering through DRM backend
- Input events handled with libinput (keyboard, pointer, touch)

Headless Cog

- Use-case example: Apple Music audio player
- No rendering at all, no GPU needed.
- Need for custom DBus bridge for user-interaction: -> example: Interaction with JS Apple Music SDK for playback control

Deploying WPE/Cog on i.MX with Yocto

Yocto layers

- <https://github.com/Igalia/meta-webkit/>
 - WPEBackends
 - Cog browser
- Poky reference distro (including GStreamer 1.18.x)
- (meta-freescale)

Open-source etnaviv driver

- Nowadays well supported in the kernel and Mesa, depending on specifics of the SoC though
- Usable WPEBackends, only working in Weston:
 - WPEBackend-fdo (recommended)
 - WPEBackend-RDK/wayland
- Upstream v4l2 plugin from gst-plugins-good for hardware decoding support

i.MX6 video decoding

- CODA960 kernel driver
- Mature integration with gst-plugins-good v4l2 plugin
- On QuadPlus: H.264 1080P@30 handled but sometimes drops frames:

720P recommended: `$ export`

`WEBKIT_GST_MAX_AVC1_RESOLUTION=720P`

i.MX8M video decoding

- Hantro G1 kernel driver and integration with gst-plugins-good
- Development by Collabora and Pengutronix
- Released in kernel 5.12
- Status:
 - 1080P@30 H.264 smooth playback
 - Should allow for up to 4K@30 with VP8 and H.264 and 4K@60 with VP9 and HEVC
 - HEVC and VP9 support: Work in progress

YUV video rendering in WPE

- Internal video sink used to support RGBA only
- Since then, additional formats are now supported:
 - YUV textures through external-oes
 - I420, Y444, YV12, Y41B, Y42B, NV12, NV21, VUYA, A420

Alternative fallback: Proprietary NXP drivers

- Where etnaviv / v4l2 decoders are not usable, fallback to NXP drivers
- WPE supports the gstreamer-imx decoders as well
- GL sink makes use of `imxvideoconvert_g2d` for zero-copy rendering

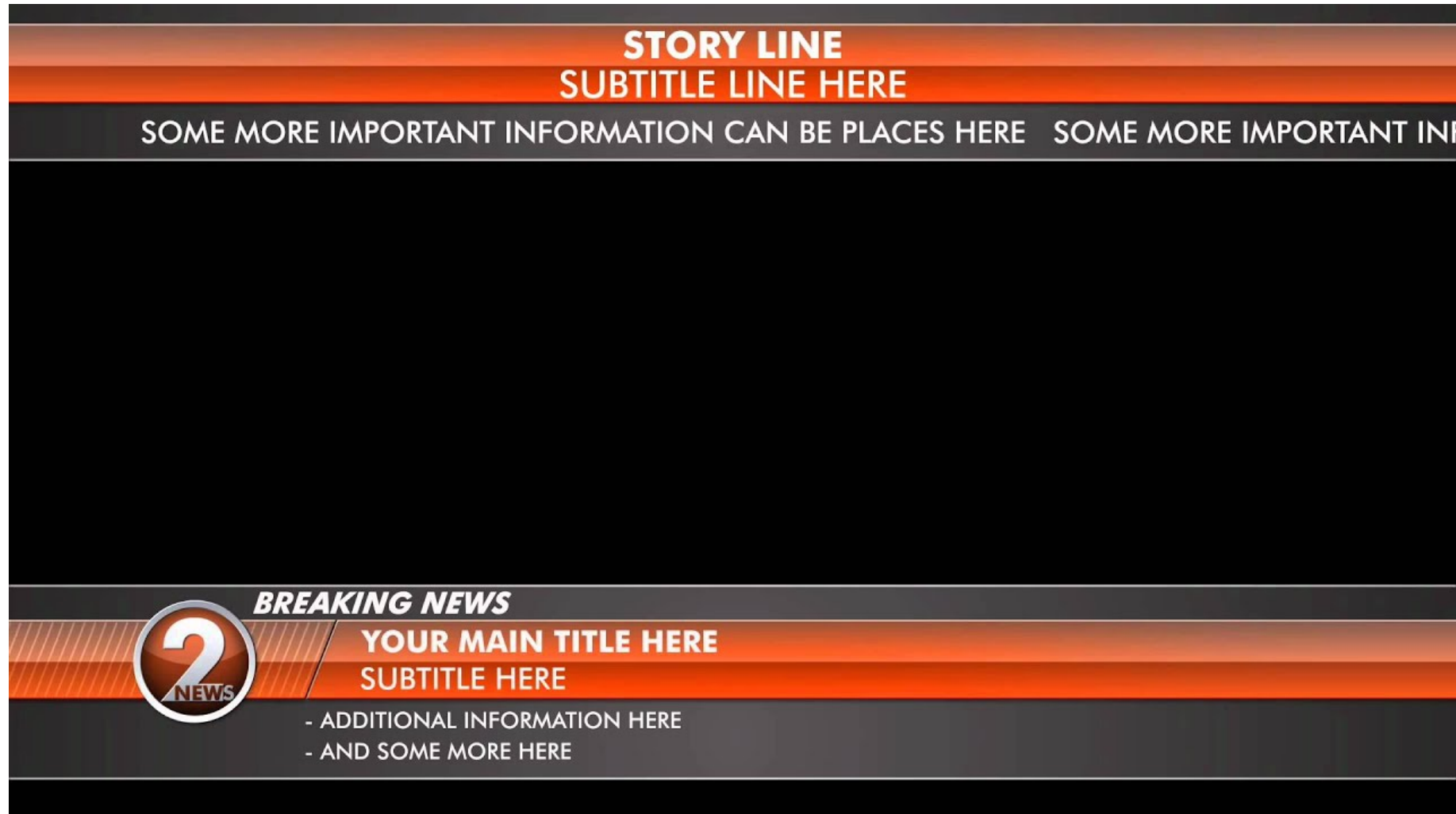
Using WPE in GStreamer applications

Use-case #1: Stinger transitions



<https://obsproject.com/wiki/Getting-Started-With-Track-Matte-Stinger-Transitions>

Use-case #2: Lower thirds



Use-case #3: Score boards



Why use WPE/GStreamer for this?

- No vendor-lockin in overlay editors
- Vast pool of Web designers
- Flexibility and broad platform support provided by GStreamer

GstWPE

- Use-cases: HTML overlays, streaming/cloud browsers
- GStreamer source element producing audio/video streams from a WPE WebView
- Two runtime modes:
 - Zero-copy from WPEBackend-FDO EGLImages to GL GStreamer sink
 - Software rasterizer with LLVMpipe to non-GL GStreamer sink

Simple examples

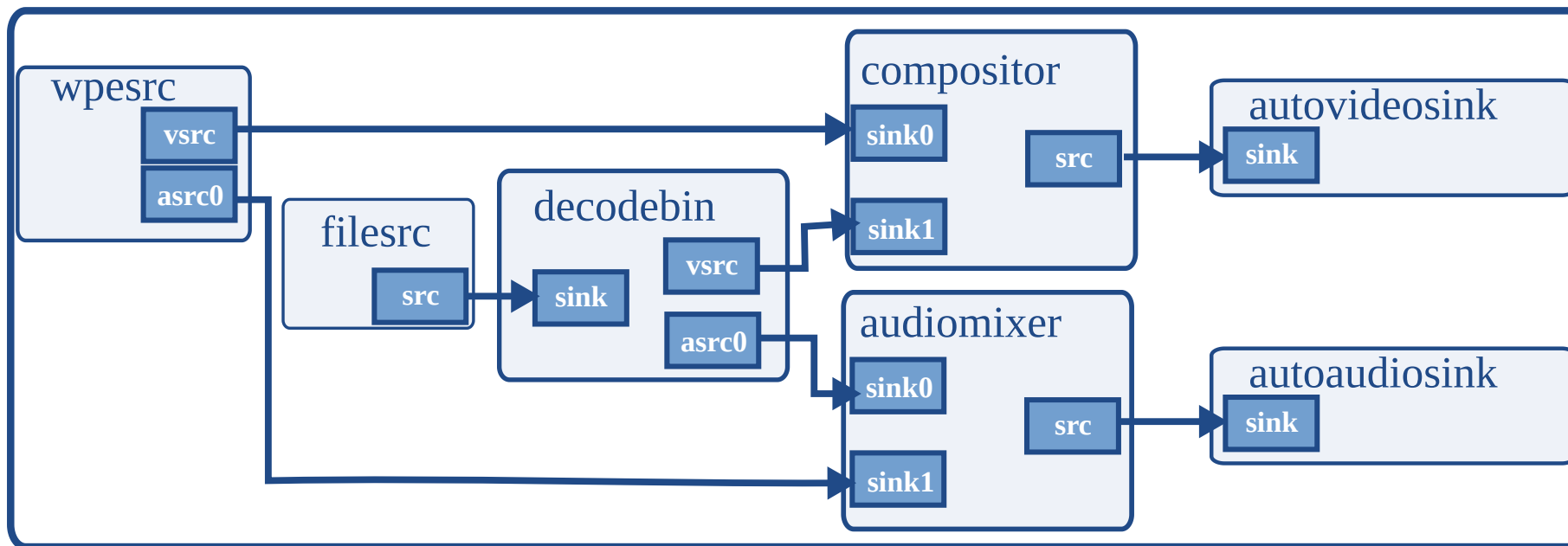
```
$ gst-play-1.0 --videosink gtkglsink \  
wpe://https://linuxfoundation.org/
```

```
# Only with GStreamer git, should work in 1.20:
```

```
$ gst-play-1.0 --videosink gtksink \  
wpe://https://linuxfoundation.org/
```

Sample pseudo-pipeline for audio/video mixing

- Don't forget to set `wpesrc::draw-background` to `0`
- Configure `zorder` value on compositor sink pads



Web-augmented one-to-many broadcasting

- Dynamic overlays controlled through a NodeJS app
- Mixed video encoded and broadcasted to a remote Janus video room
- Video consumption through WebRTC
- <https://www.youtube.com/watch?v=QNZJYOuVGiE>



Wrap-up

- WPEWebKit, WPEBackend-FDO, Cog: <https://wpewebkit.org>
- GstWPE: <https://gstreamer.freedesktop.org/documentation/wpe/>
- Yocto overlay: <https://github.com/Igalia/meta-webkit>
- Questions?

