The OpenMAX Integration Layer standard

Giulio Urlini (giulio.urlini@st.com)

Advanced System Technology

Topics

- Standardization organization and goals overview
- OpenMAX software layers
- OpenMAX IL architecture
- ST implementation: Bellagio
- GStreamer multimedia framework integration
- GStreamer with OpenMAX demo

Khronos & OpenMAX

- The OpenMAX working group is a part of the Khronos Group, a member-funded industry consortium focused on the creation of open standard, royalty-free APIs to enable the authoring and accelerated playback of dynamic media on a wide variety of platforms and devices.

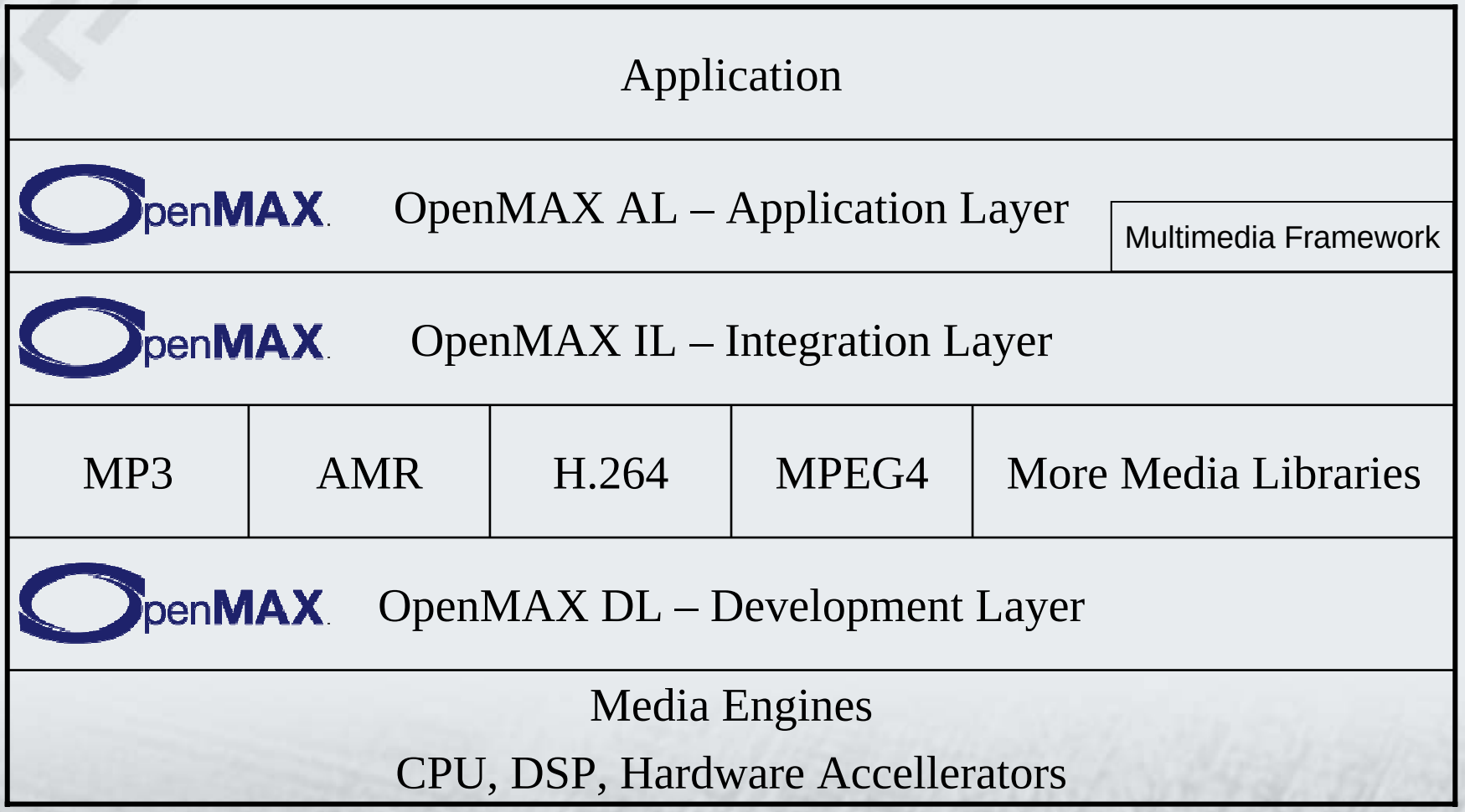
  Khronos homepage: http://www.khronos.org/

  OpenMAX home page: http://www.khronos.org/openmax/

OpenMAX general definition

- OpenMAX™ is a royalty-free, cross-platform API that provides comprehensive streaming media codec and application portability
- It enables accelerated multimedia components to be developed, integrated and programmed across multiple operating systems and silicon platforms
- The OpenMAX API will be shipped with processors to enable library and codec implementers to rapidly and effectively make use of the full acceleration potential of new silicon - regardless of the underlying hardware architecture

OpenMAX Layers



OpenMAX Application Layer

- The OpenMAX AL API defines a set of APIs providing a standardized interface between an application and multimedia middleware
- The AL provides application portability with regards to the multimedia interface.

OpenMAX Integration Layer

- OpenMAX IL serves as a low-level interface for audio, video, and imaging codecs used in embedded and/or mobile devices
- It gives applications and media frameworks the ability to interface with multimedia codecs and supporting components (i.e. sources and sinks) in a unified manner
- The codecs themselves may be any combination of hardware or software and are completely transparent to the user. Without a standardized interface of this nature, codec vendors must write to proprietary or closed interfaces to integrate into mobile devices

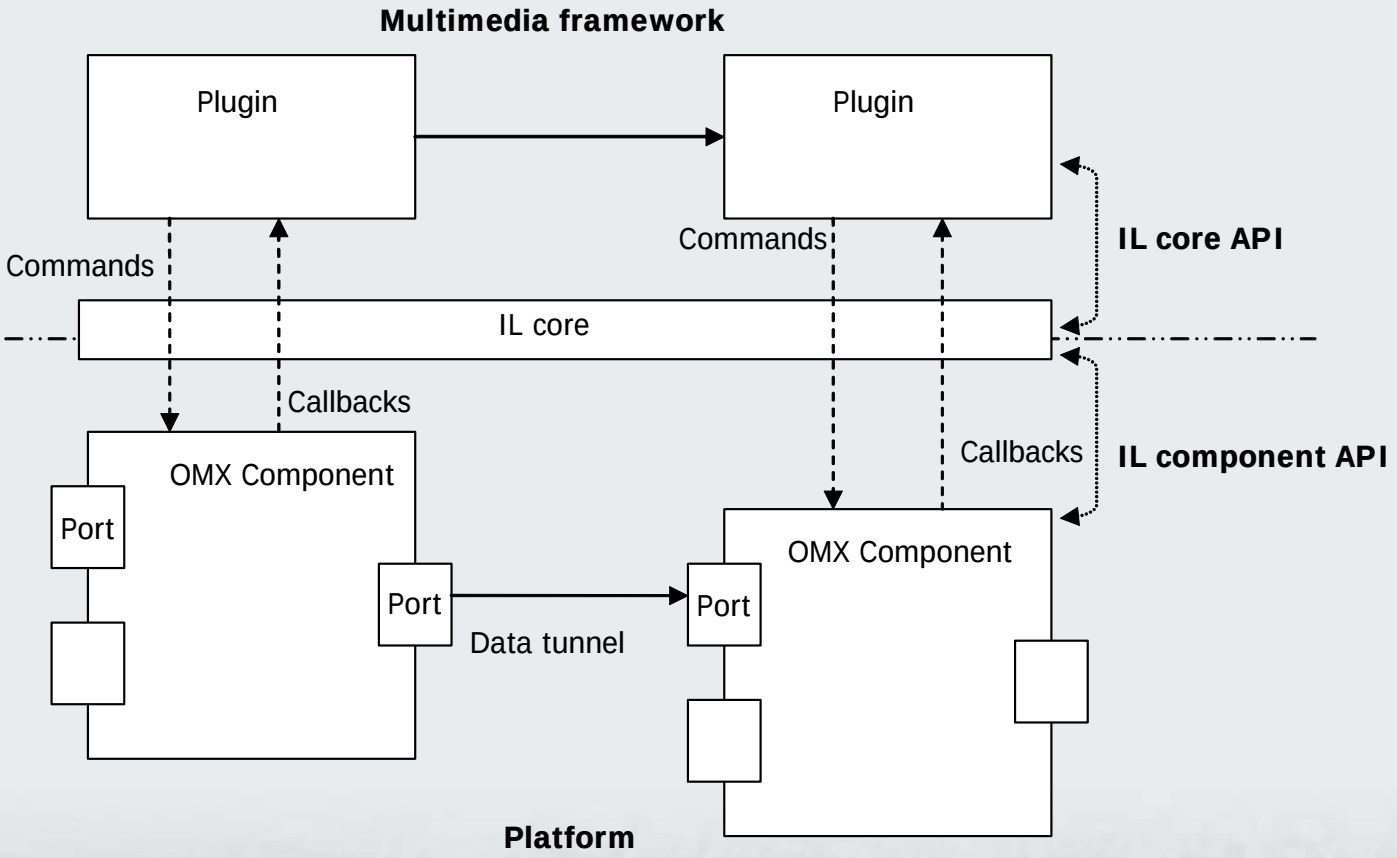
OpenMAX Development Layer

- OpenMAX DL defines an API which contains a comprehensive set of audio, video and imaging functions that can be implemented and optimized on new processors by silicon vendors and then used by codec vendors to code a wide range of codec functionality.
- It includes audio signal processing functions such as FFTs and filters, imaging processing primitives such as color space conversion and video processing primitives to enable the optimized implementation of codecs such as MPEG-4, H.264, MP3, AAC and JPEG.

OpenMAX IL domains

- The OpenMAX IL components are divided in three main domains: audio, video and imaging
- A support for other domains (timing info, bitstream) is provided. It is generally classified as “other” domain
- The other domain includes also the mixed-domain components, like video/imaging processors

OpenMAX IL architecture



OpenMAX IL Core

- Provides the functions to build and destroy each component
- It gives all the information functionalities needed
  - List of components
  - List of capabilities (roles) where supported
- It tries to set up the direct connection between two components

IL Core functions

- `OMX_Init`, `OMX_Deinit`: inintialize and de-initialize the core environment.
- `OMX_ComponentNameEnum`, `OMX_GetComponentsOfRole`, `OMX_GetRolesOfComponent`: these functions provide information about the components available in the system, the components that support a specific *role* and the *roles* supported by a given component (see later for role definition)

IL Core functions

- `OMX_GetHandle`, `OMX_FreeHandle`: these functions are used to create and destroy an instance of a given component
- `OMX_SetupTunnel`: This function tries to establish a tunnel between two ports of two components.

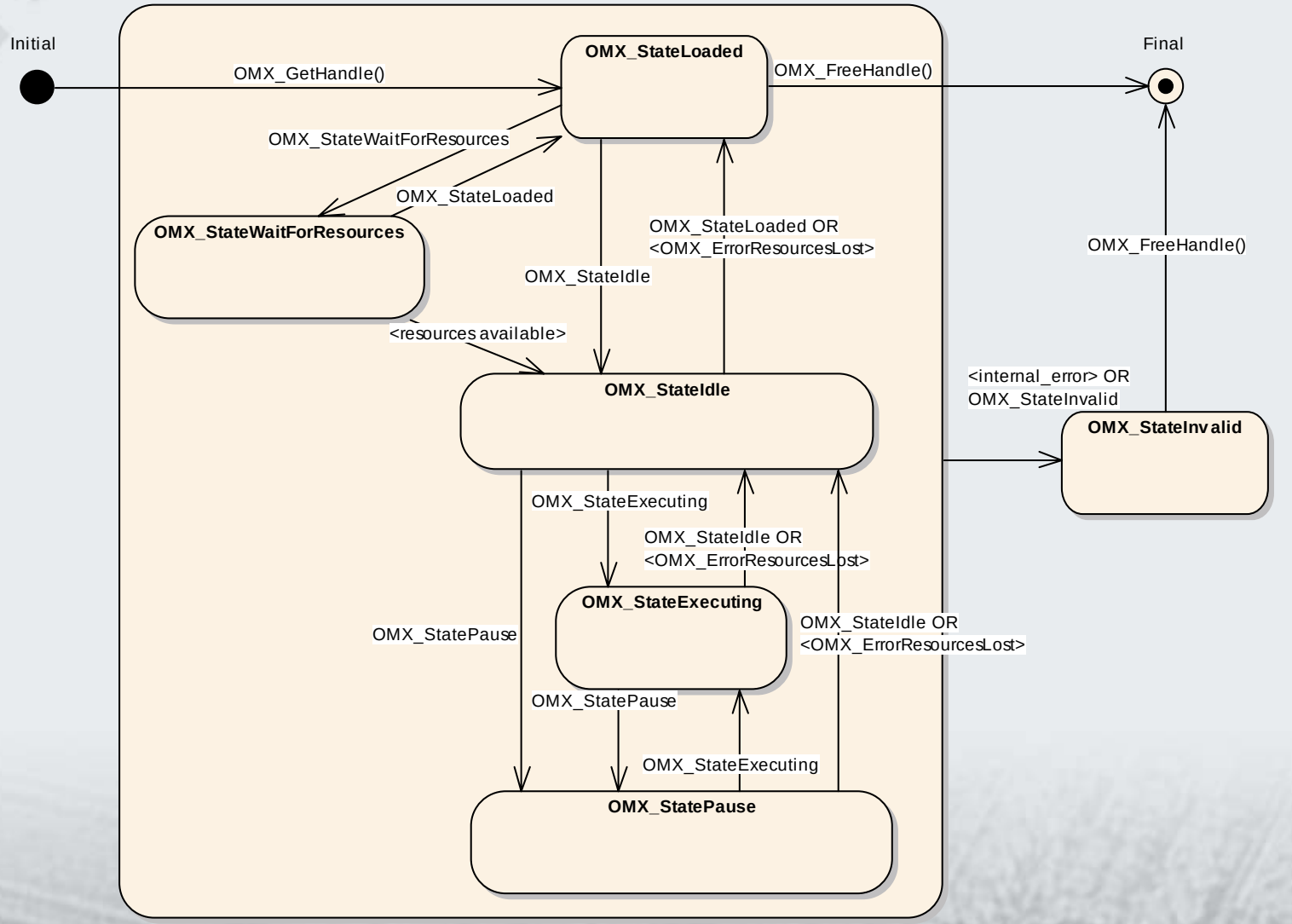
IL Components

- An OpenMAX IL component is a black box that receives instruction with asynchronous commands and transmit/receive data through ports
- It is uniquely identified by a string name, like the following

  “OMX.vendor_name.component_name”

- A component is **base profile** compliant if it supports the standard and buffer are exchanged only with an IL client
- A component is **interop profile** compliant if it supports the base profile and is able to setup the tunneling with other interop components

Component states



Component initialization

- IL client requests a component by name using `OMX_GetHandle()`
  - Naming convention: “OMX.<vendor_name>.<vendor_convention>”
  - Example: “OMX.ST.AUDIO.MP3DEC”
- IL client sets component parameters
- Component is put into IDLE state
- Resources (buffers) can be allocated either by the IL client or by the component itself
- When component is ready, it notifies the IL client using a callback

IL Component ports

- The data is transferred to/from IL client and between components through *ports*
- Any port contains the information about any buffer needed, the minimum size of them, and any other implementation specific need
- Any buffer used in a port should be registered on it using the functions `UseBuffer`/`AllocateBuffer` during the setup phase of the component (`Loaded` state)

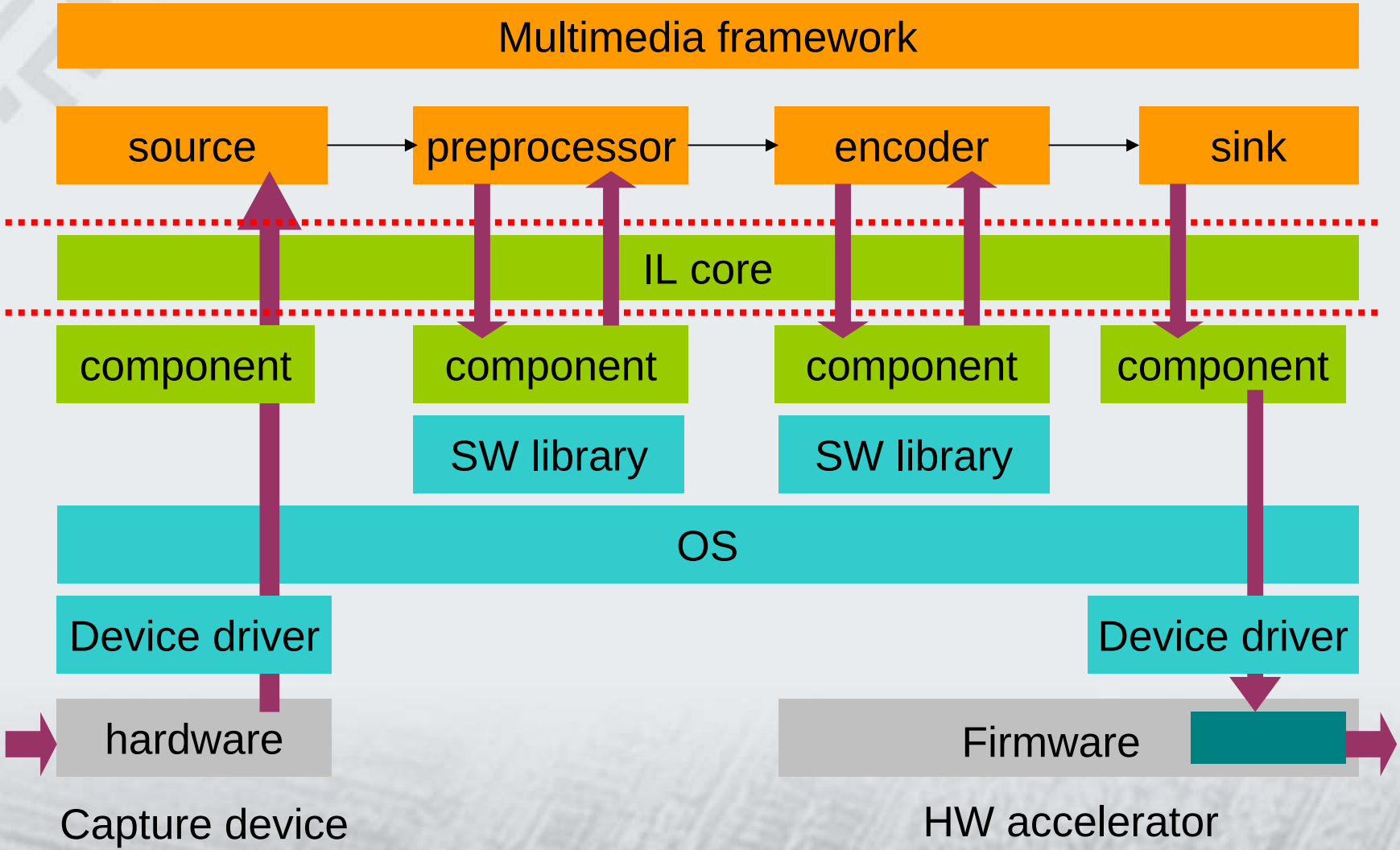
IL Component ports

- A port can be enabled or disabled independently, so that the behavior of the component is not affected
- A port contains all the information about the data format handled, and the possible capabilities

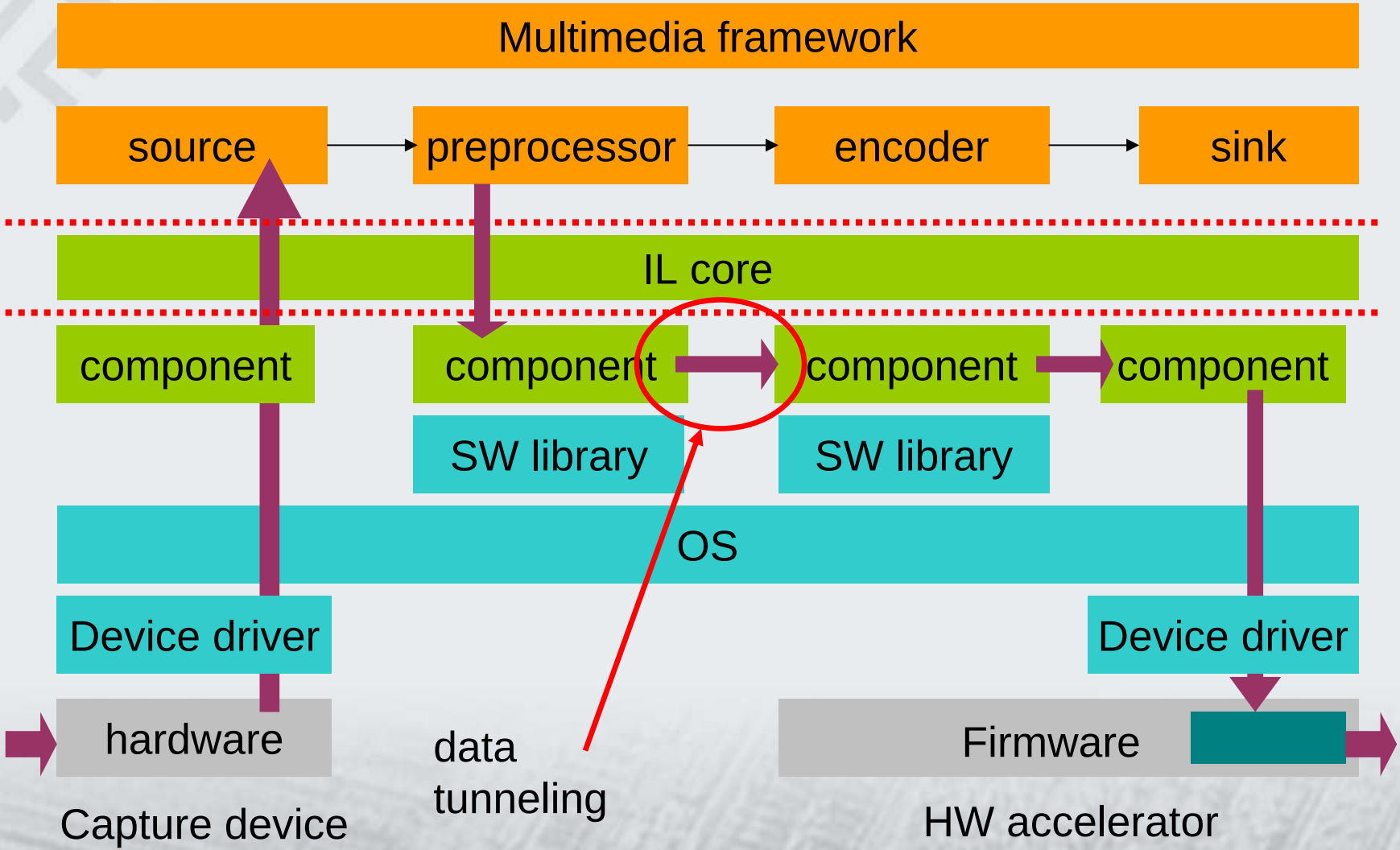
IL Component ports

- The functions used to send or retrieve data to or from the ports are:
  - `EmptyThisBuffer`: used to send a buffer to a component. This call is asynchronous
  - `FillThisBuffer`: used to retrieve a buffer from a component. This call is asynchronous

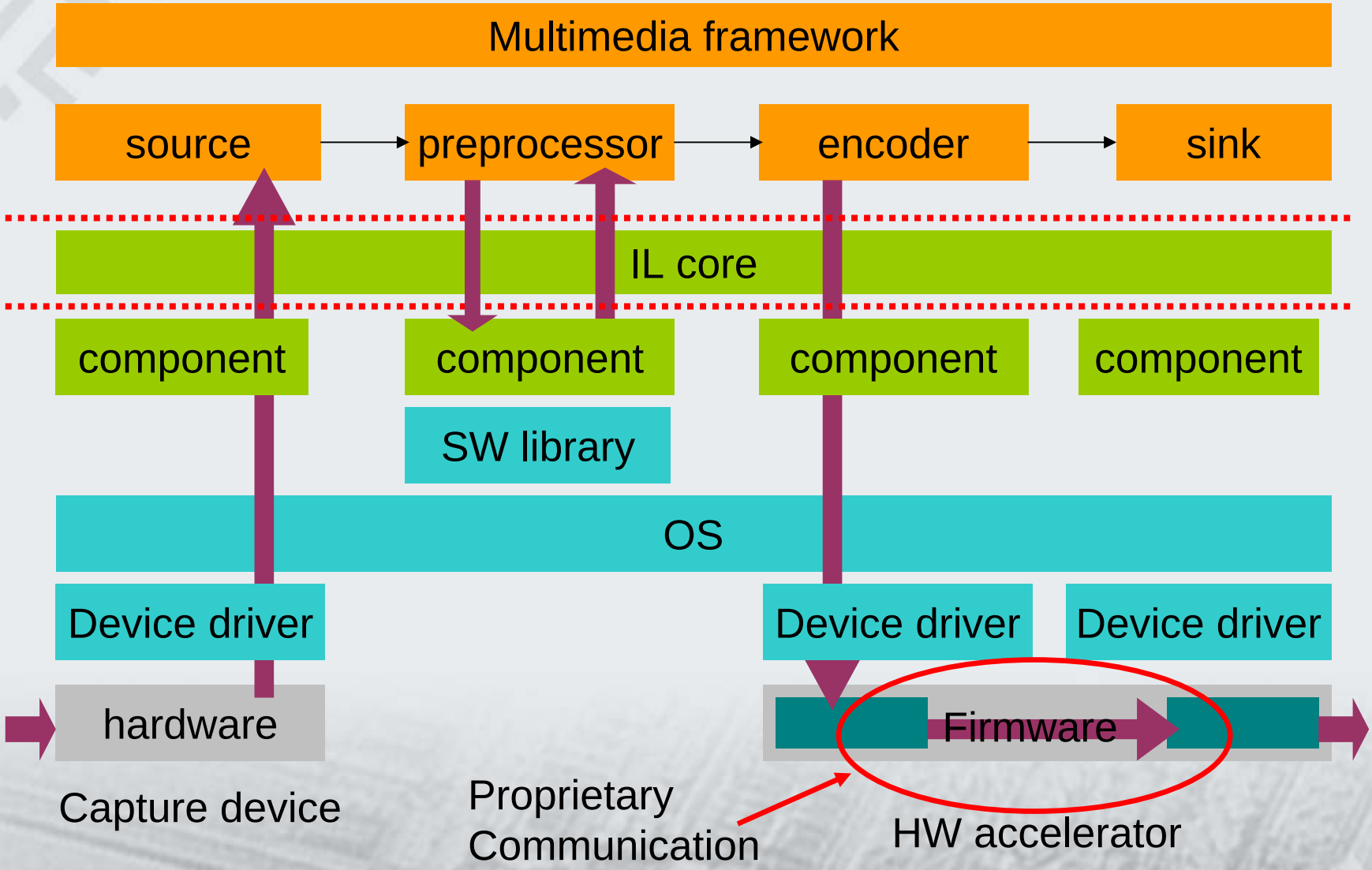
Default data flow



Data flow with tunneling



Data flow with proprietary communication



IL Component `SendCommand`

- `OMX_SendCommand`: this function sends a command to the component, that is handled asynchronously by the component. It can send the following types of commands:
  - State change
  - Flush buffers on a given port.
  - disable/enable a given port
  - Mark a buffer

Component callbacks

- There are three callbacks available for IL client to synchronize the execution
  - `EventHandler`: this function is executed by the component when a `SendCommand` request has been completed, when an automatic state transition occurs, due to the resource manager for instance, and when an error is signaled
  - `EmptyBufferDone`: when the buffer is input has been totally consumed
  - `FillBufferDone`: when a buffer in output has been totally filled

Setting and retrieving parameters

- Static parameters are set during loaded state through the `SetParameter` call. The `GetParameter` call allows the application to retrieve the current value of a setting
- Dynamic parameters can be set at any time using the `SetConfig` call. The current value is obtained with the `GetConfig` call

Resource management

- The OpenMAX IL APIs provide also some entry points for (optional) resource management
  - Each component could be characterized by a priority value
  - A `WaitForResources` special state is provided to put in stand by a component when a needed resource is not available

Enhanced Resource management

- If the component looses a dynamic resource it can be put in a suspended state, and when the resource becomes available its execution can be restored
- An optional level of quality setting can be applied to the component for saving resources

What's new in IL 1.1

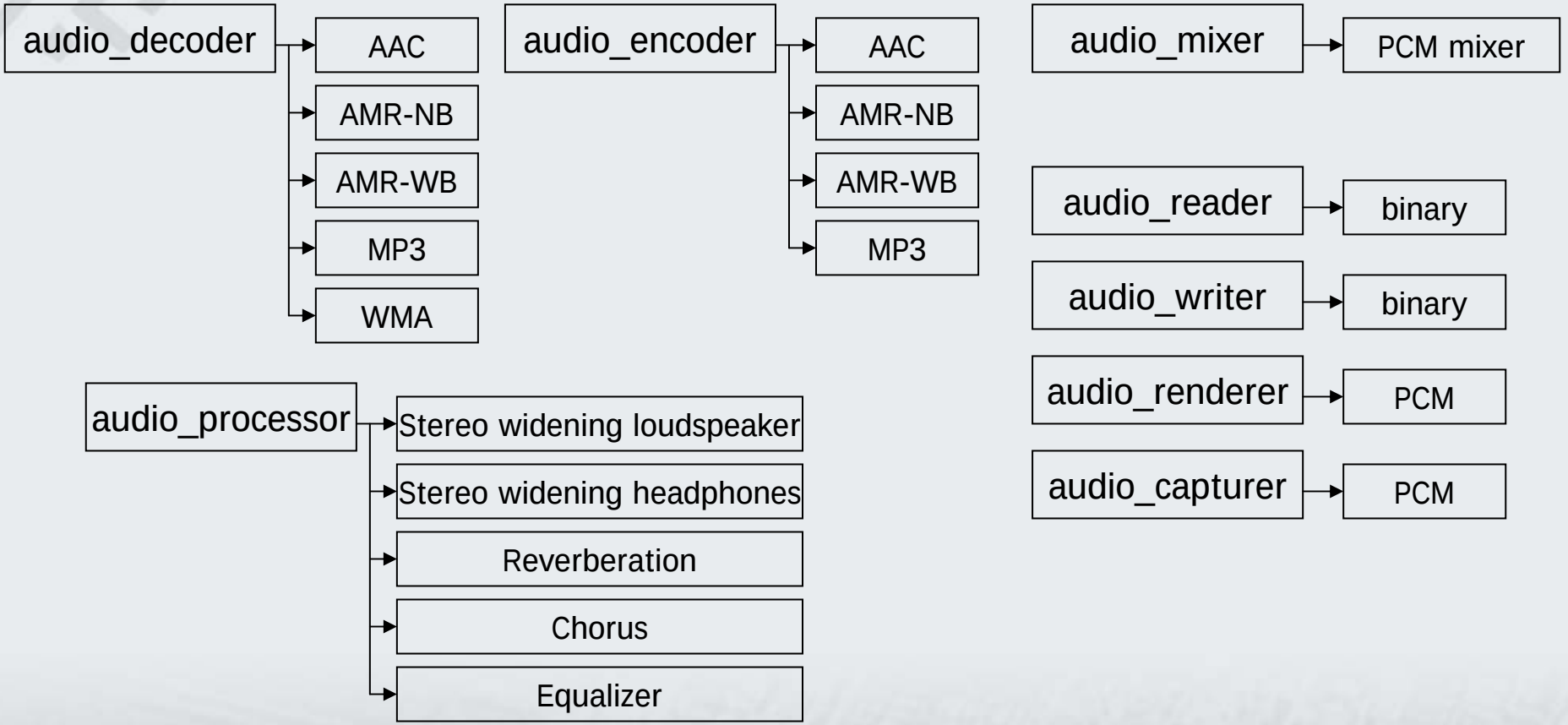
- A set of standard components (codecs, renderers, readers…)
- The "content pipes" abstract access do data storage, files or streams. The functionalities added are:
  - Open, read, write, seek pos, get pos, close
  - Check for available bytes, read or write data via pipe supplied data buffers
- Enhanced resource management
- Definition of component roles: e.g. audio decoder supports MP3 and OGG roles

IL standard components

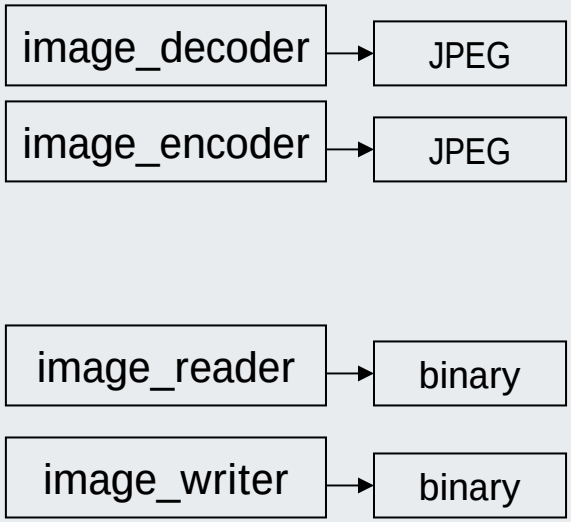
- The OpenMAX IL 1.1 release supports a list of standard components, that share the same ports and high level functionality (class)
- Each standard component supports a set of common settings, and can support additional custom settings
- The name of the component is also standard, specified as follows:

  “OMX.vendor. component_class.format”

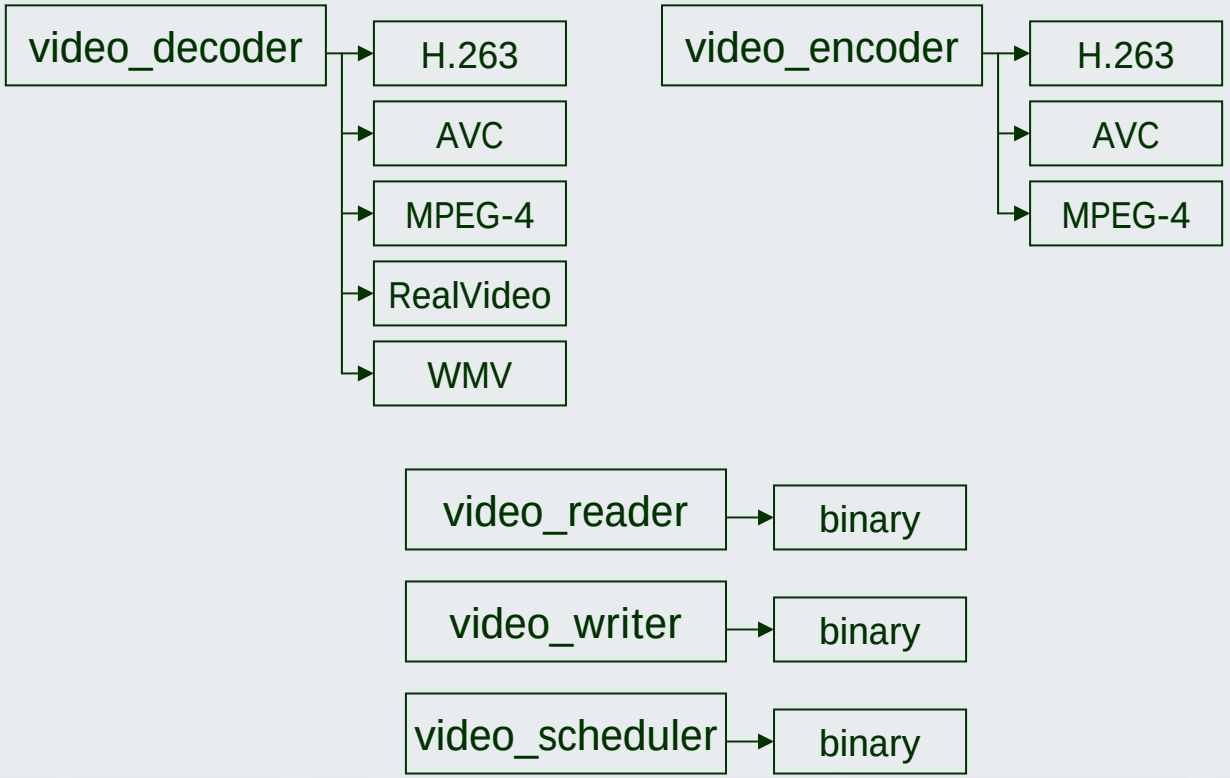
Standard audio components



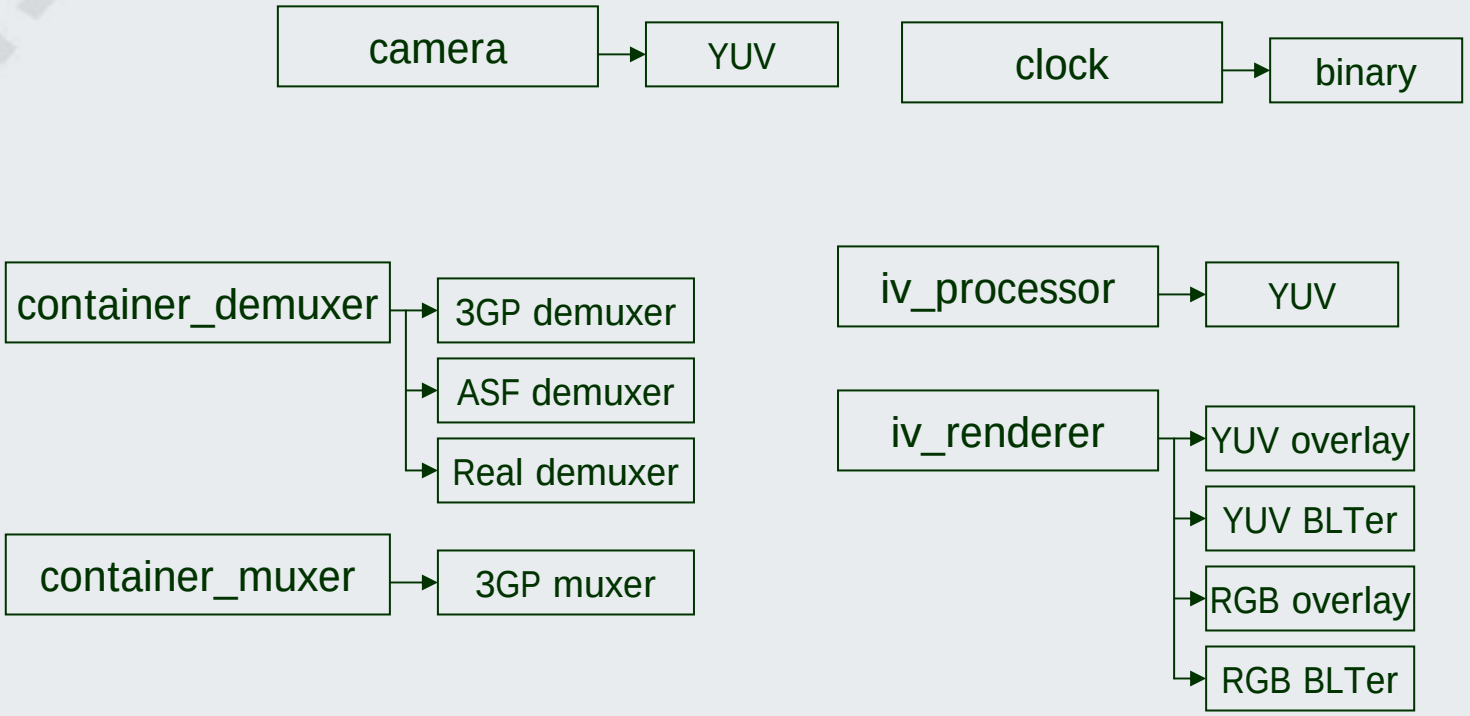
Standard imaging components



Standard video components



Other standard components



Conformance testing process

There are two ways to use OpenMAX

- **Implementers -** may create and deliver a product using the publicly released specifications, but not claim it is OpenMAX "compliant" or if it is a software or hardware engine, they cannot advertise it using OpenMAX logos or trademarks. The Implementor option carries no cost or license fees.
- **Adoption/Conformance Testing -** Adopters can download and run the conformance tests (currently only OpenMAX IL) and if the implementation passes, they can advertise and promote the product as being compliant; using the OpenMAX logos and trademarks under a royalty-free license. The procedure requires a fee, that is 10.000 $ for Khronos members, and 2.500 $ for non-members
- Any further detail can be found at:
  - **http://www.khronos.org/openmax/adopters/**

ST Bellagio implementation

- STMicroelectronics has developed a set of components that support the OpenMAX IL API
- The implementation runs on a Linux PC
- The public available implementation can be found in Source Forge at:

  https://sourceforge.net/projects/omxil/

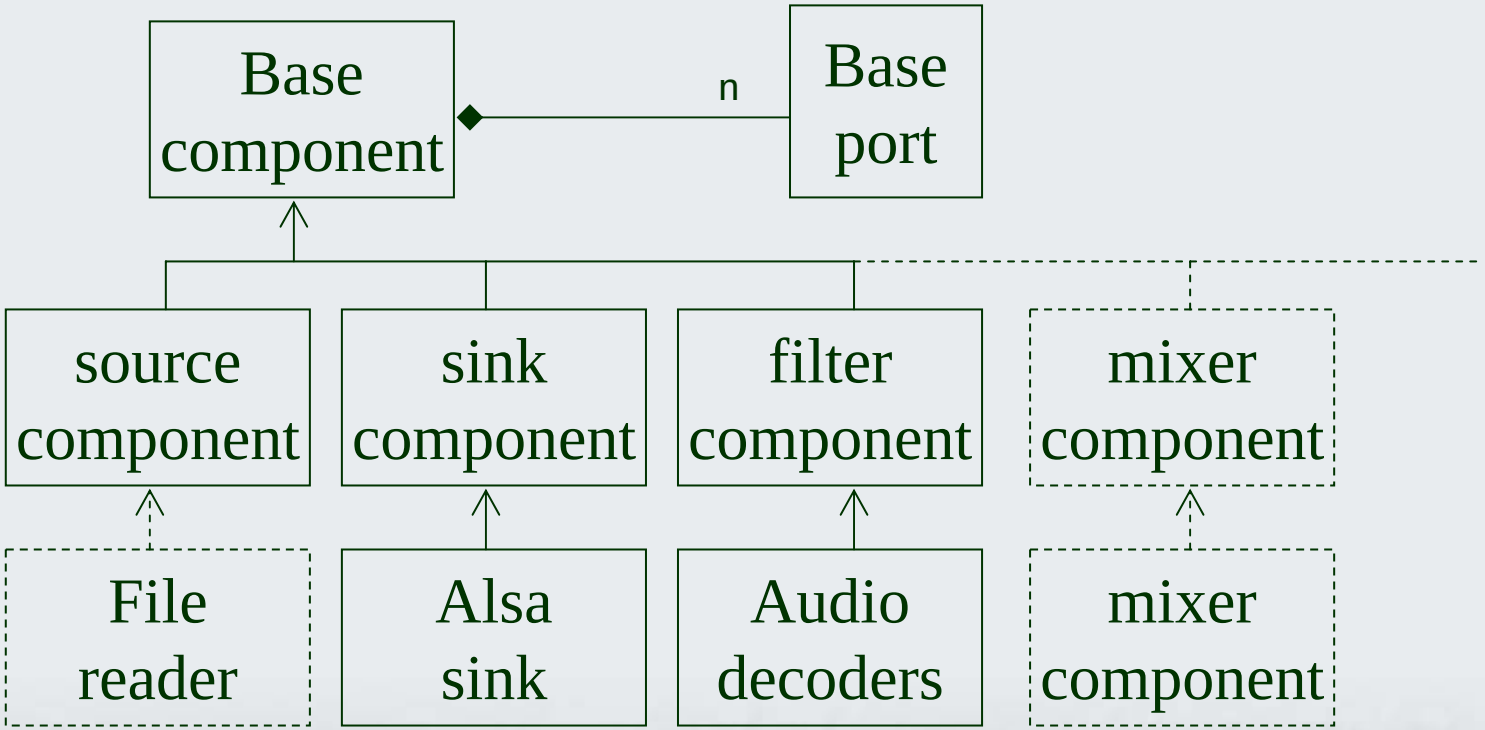
Bellagio current status

- Release 0.3 available
- Audio decoders provided:
  - Mp3 decoder based on madlib
  - Ogg vorbis decoder based on libvorbis
  - Mp3/ogg multiple component based on ffmpeg and libvorbis
- An audio PCM sink based on alsa library is also provided

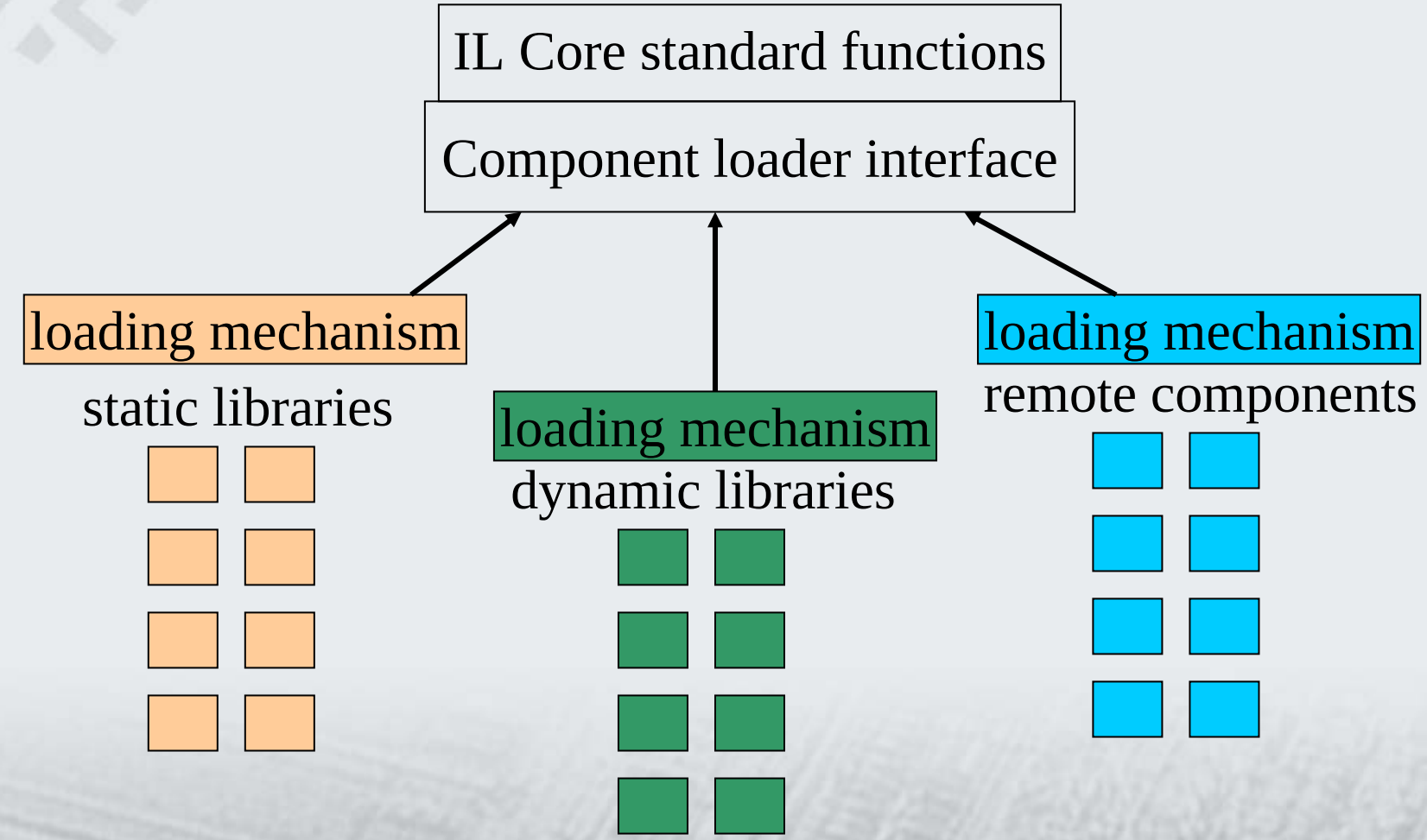
Bellagio roadmap

- Addition of video decoders components (MP4, H264)
- Addition of a color converter
- Addition of a video sink based on directFB
- Addition of an audio PCM mixer
- Addition of source/sink components that use the content pipes
- Make the project more similar to an SDK

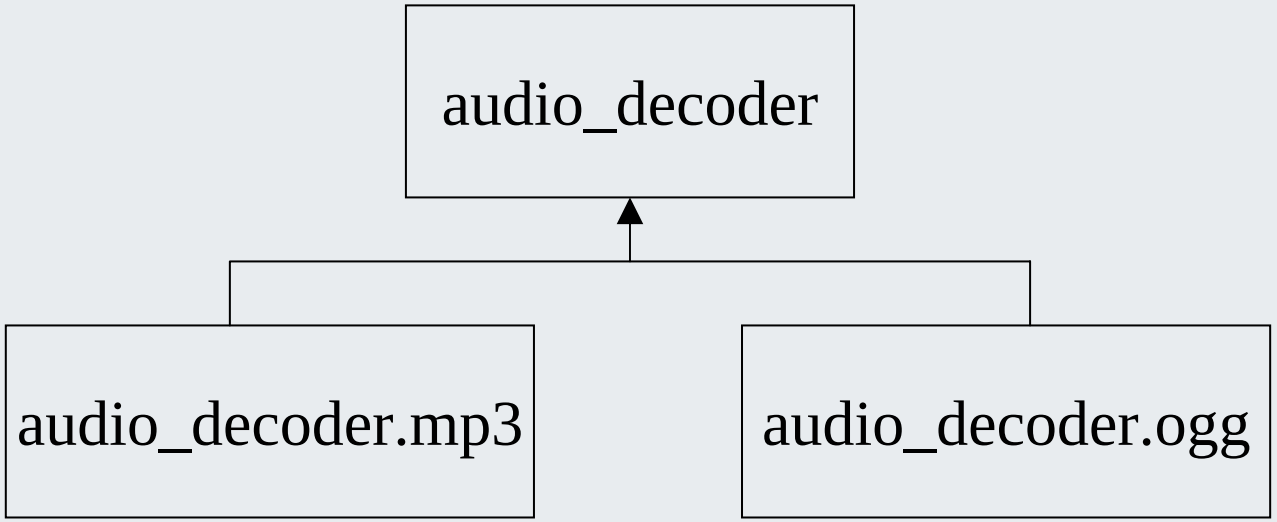
Software architecture



Component loader mechanism



Multiple roles support



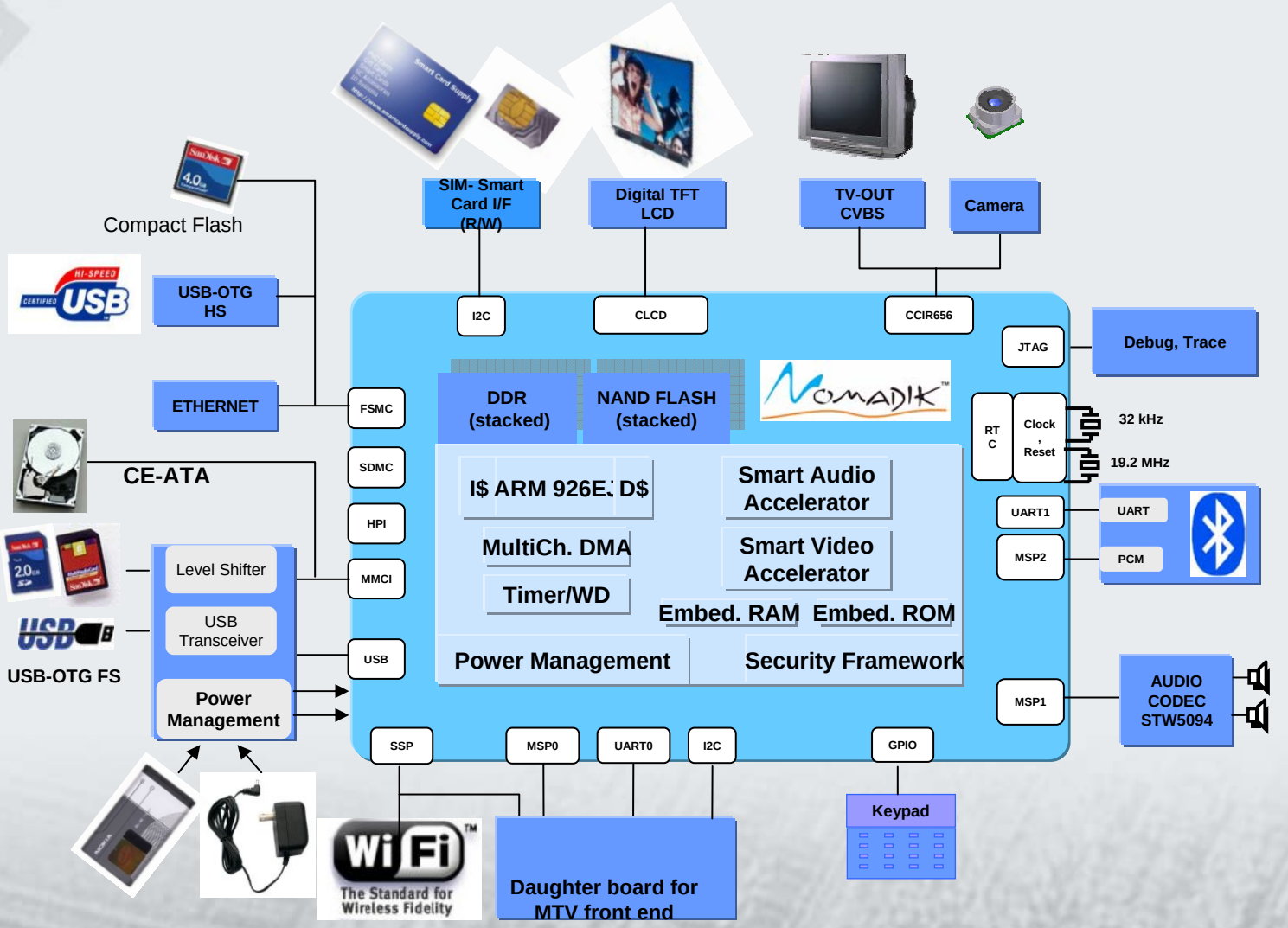
GStreamer integration

- GStreamer plugins for OpenMAX alsa sink component and mp3 decoder
- Executed on PC platform
- Without any change executed on development ARM based board
- Connected with GStreamer standard plugins

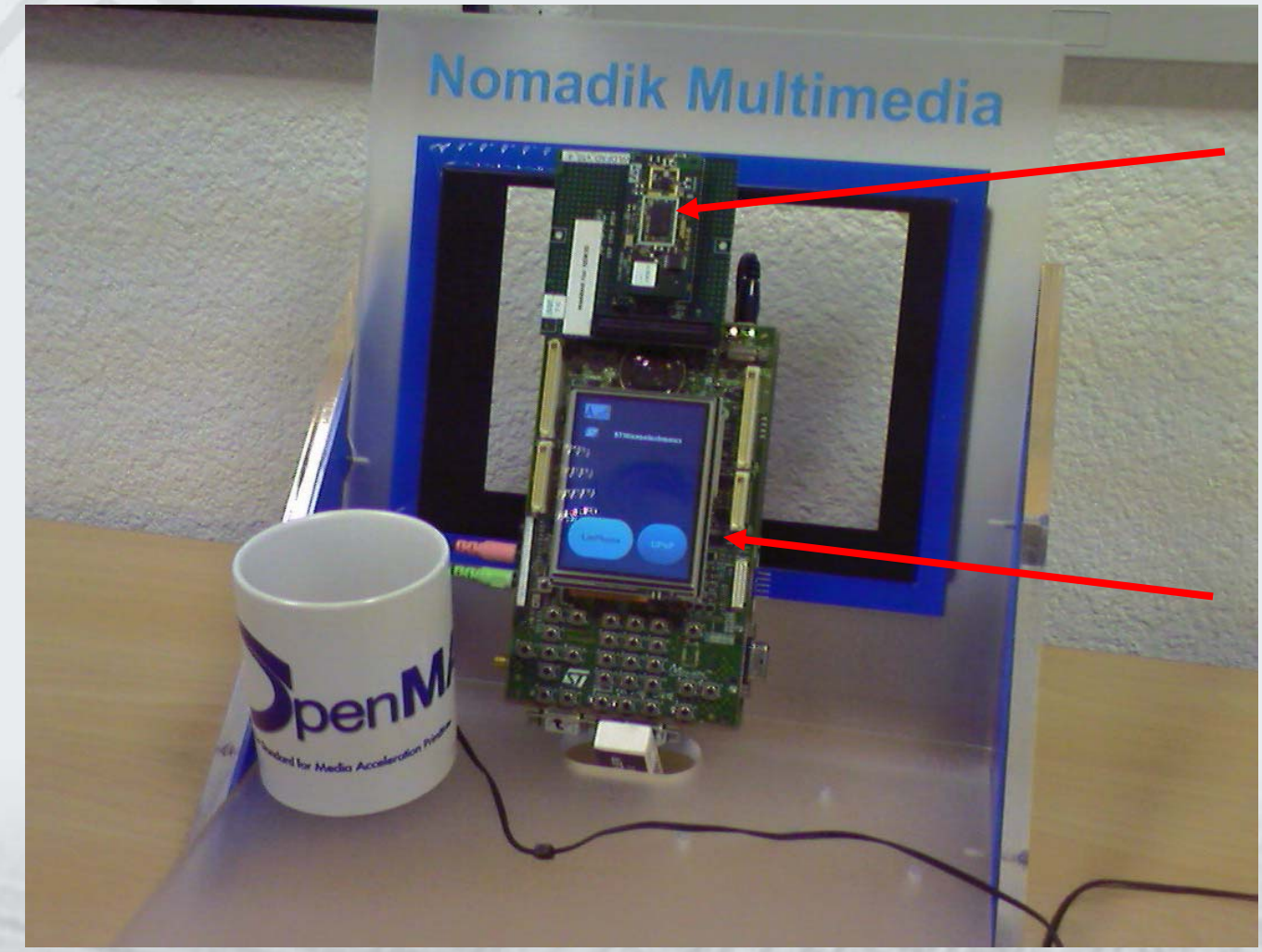
GStreamer roadmap

- Addition of multiple format plugins to use the role support in OpenMAX components
- Addition of video components
- Contribution to the GStreamer community

Embedded platform Nomadik



Prototype



Low-power WLAN chipset

Nomadik™ development board

Demo OpenMAX – GST on Nomadik