igel

Technology Consulting Company
Research, Development &
Global Standard

Integrating a Hardware Video Codec into Android Stagefright using OpenMAX IL

Damian Hobson-Garcia(Igel),
Katsuya Matsubara (Igel),
Takanari Hayama (Igel),
Hisao Munakata (Renesas Electronics)

Introduction

- Android Stagefright media server
  - – Handles video/audio playback/recording on an Android device (in 2.3 – Gingerbread)
  - – Built in software codecs
    - Enabled by default
- Hardware codec
  - – Faster than software codec
  - – Frees up the CPU for other tasks (eg. UI)
    - Require integration

Why bother?

- We want to play video at 1920x1080 @ 30fps from a mobile platform
- Stagefright S/W decoder won’t play certain high resolution videos
  - certain features not supported

Getting H/W into Stagefright

- How can we get H/W codec into Stagefright?

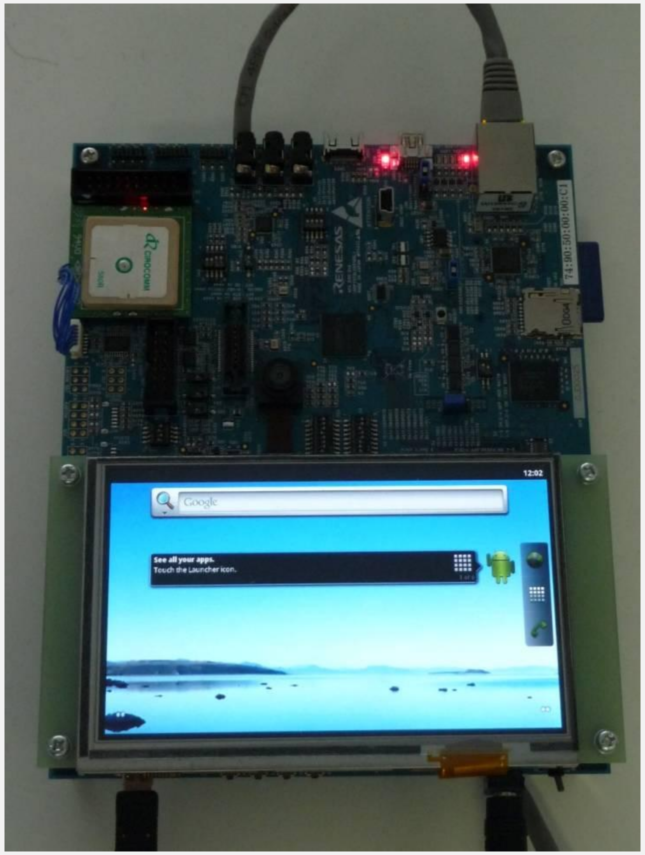
OpenMAX IL

- We integrated an AVC (H.264) decoder into Stagefright using OpenMAX IL
- Here’s what we found out.

Overview

- Hardware
- OpenMAX IL/Bellagio
- Android Stagefright Integration
  - Video Decoder Specific Considerations

Platform



- Renesas SH 7372 SoC (ARM Cortex-A8 @ 800MHz on board)

  http://www.renesas.com/prod/assp/mobile/ap4.html

- Hardware assist IP
  - Video codec (AVC, MPEG)
  - Audio codec (AAC, MP3)
  - Image processing (scaling, rotation, colour conversion, filtering)
  - JPEG codec
  - etc.

Hardware Acceleration

Video Processing Unit (VPU)

- Video AVC/MPEG codec
  - H.264 High/Main/Baseline Profile codec
  - H.263 codec
  - 1920x1080 @ 30fps throughput
  - YCbCr 4:2:0 color format

Video Engine Unit (VEU)

- Image processing
  - RGB <-> YCbCr (planar)
  - Rotation
  - Scaling

+ necessary drivers/libraries

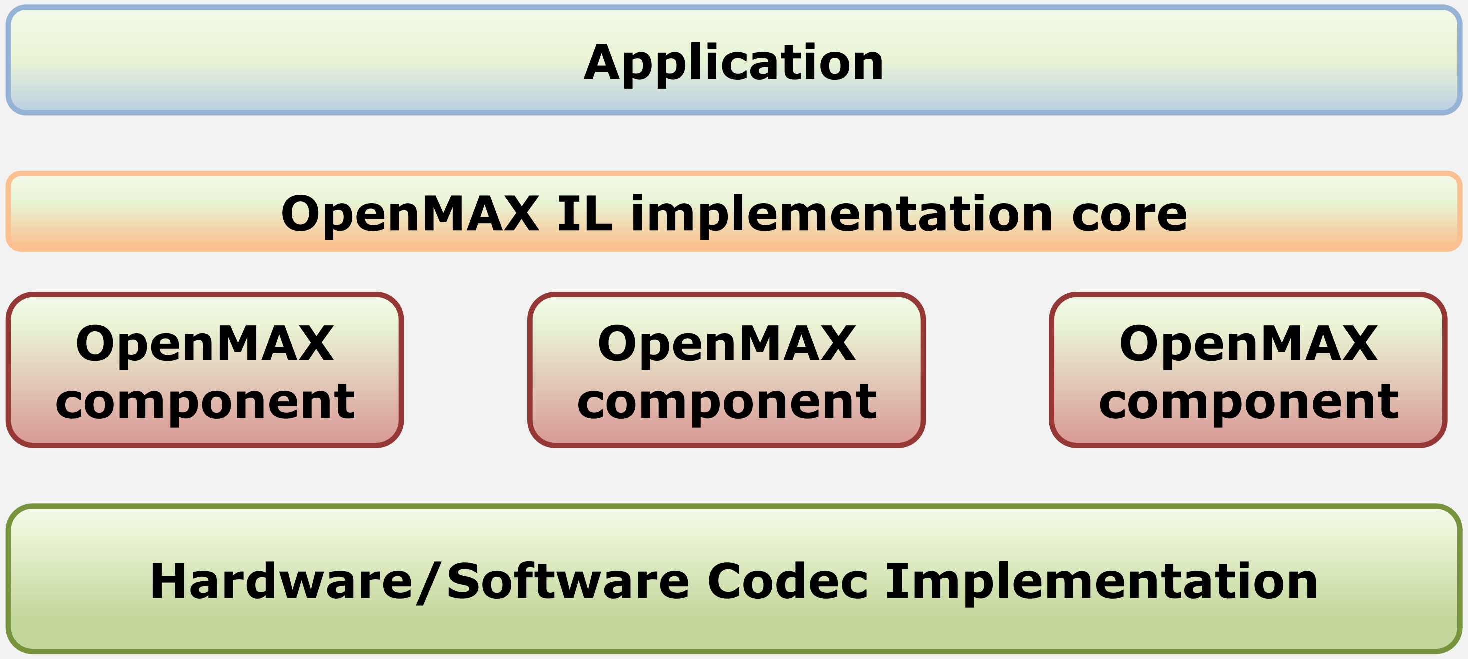
Overview

- Hardware
- OpenMAX IL/Bellagio
- Android Stagefright Integration
  - Video Decoder Specific Considerations

OpenMAX IL

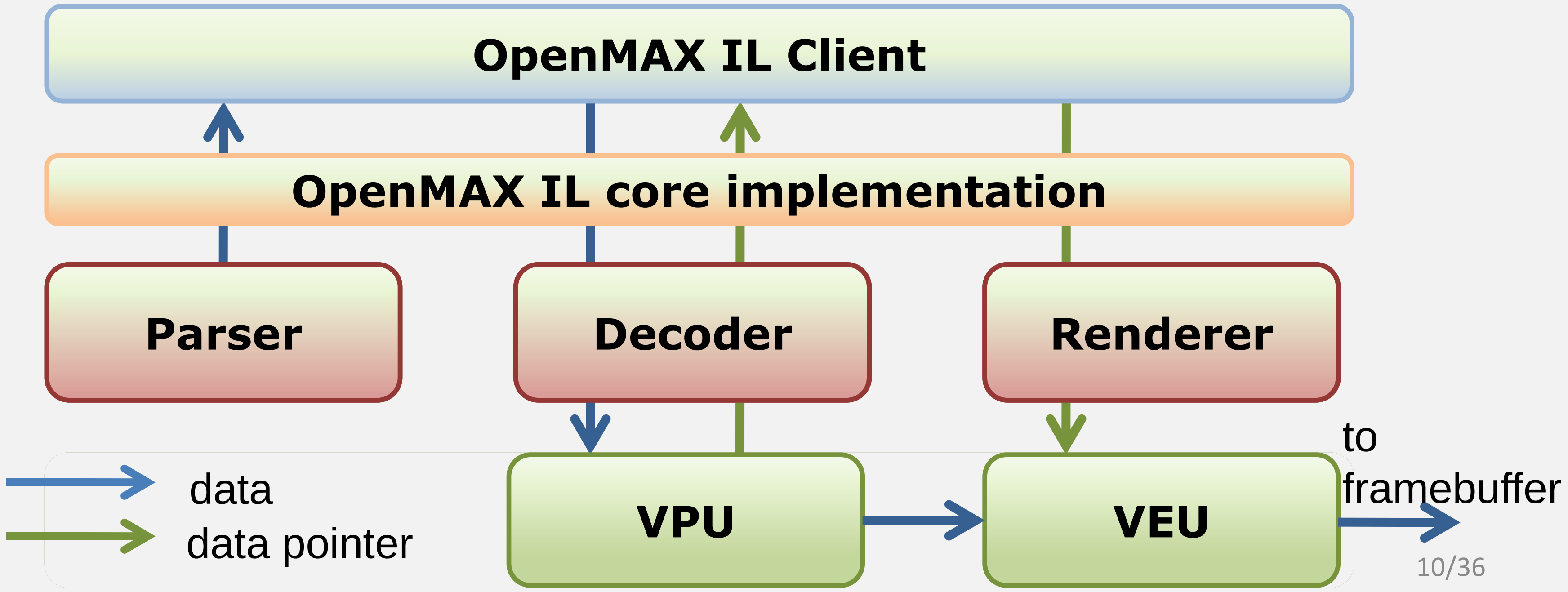
"The OpenMAX IL (Integration Layer) API defines a standardized media component interface to enable developers and platform providers to integrate and communicate with multimedia codecs implemented in hardware or software"

Khronos Group - http://www.khronos.org/openmax/


Application
OpenMAX IL implementation core
OpenMAX component
OpenMAX component
OpenMAX component
Hardware/Software Codec Implementation

Sample Configuration

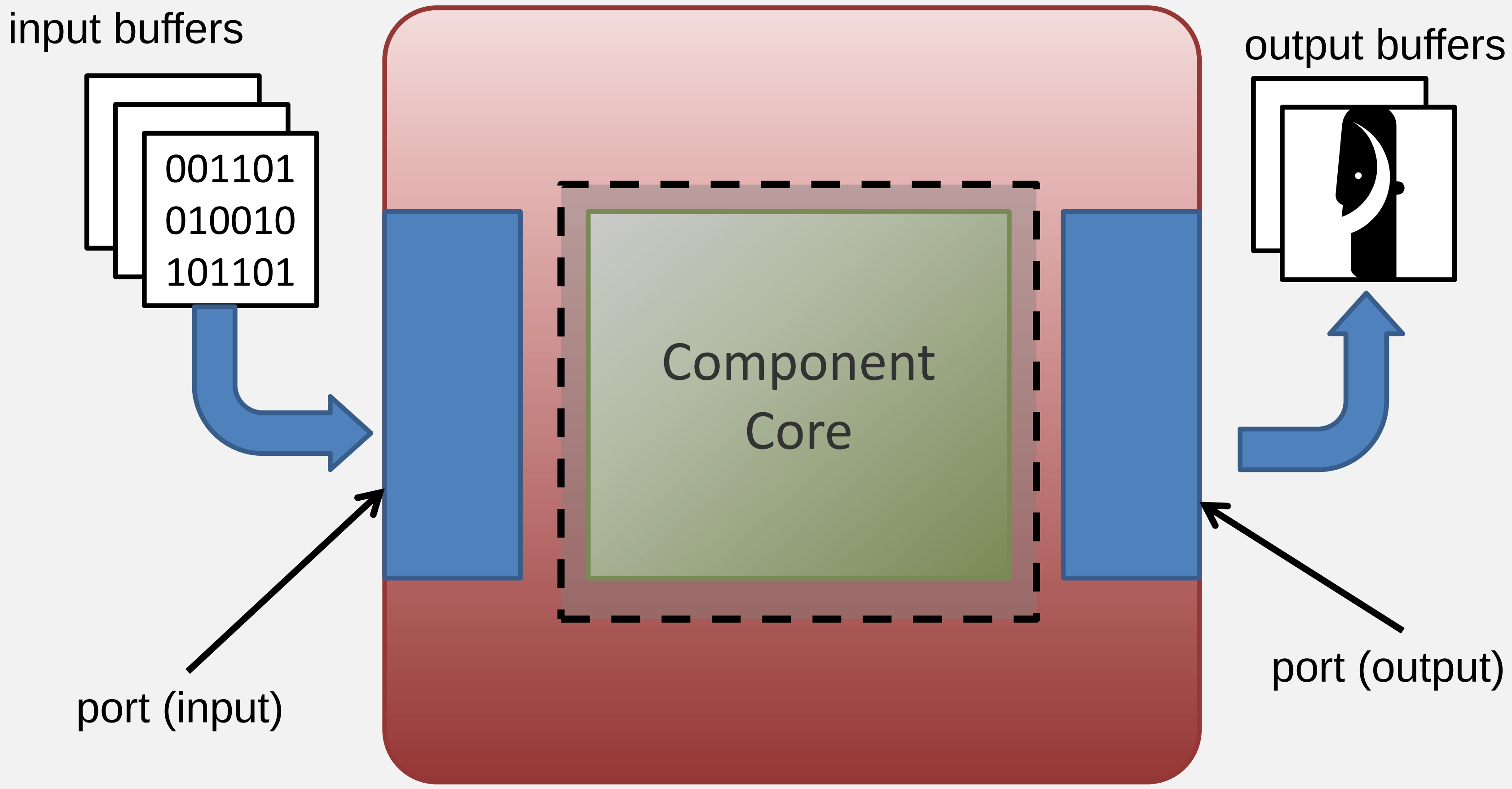
- Target application could be GStreamer , Android or something else
- Use Bellagio OpenMAX IL implementation as the core



Bellagio – What and Why?

- Open source (LGPL) OpenMAX IL library
  - http://omxil.sourceforge.net/
- OpenMAX IL core, component base and framework provided
- Provides example components, simple test programs
  - ffmpeg, camera input, jpeg, etc.

Anatomy of a component



OpenMAX IL access mainly through component ports

Making an OpenMAX IL component

1. Look at one of the Bellagio components
   - lots to reuse
   - `/src/base/omx_base_*`
2. Configuration interface
3. Data interface
4. Buffer allocation (if necessary)
5. Bellagio specific setup

OpenMAX IL functions to implement

- Component represented as `struct OMX_COMPONENTTYPE`
- Need to implement/customize (at a minimum):

  Configuration Interface
  - `(*SetParameter)(…)` – Set component properties
  - `(*GetParameter)(…)` – Get component properties
  - `(*SetCallbacks)(…)` – Set callbacks to use

  Data Interface
  - `(*EmptyThisBuffer)(…)` – Process an input buffer
  - `(*FillThisBuffer)(…)` – Process an output buffer

  Buffer allocation (if necessary)
  - `(*UseBuffer)(…)` – Application allocated buffer
  - `(*AllocateBuffer)(…)` – Component allocated buffer

All prototypes in `include/OMX_Component.h`

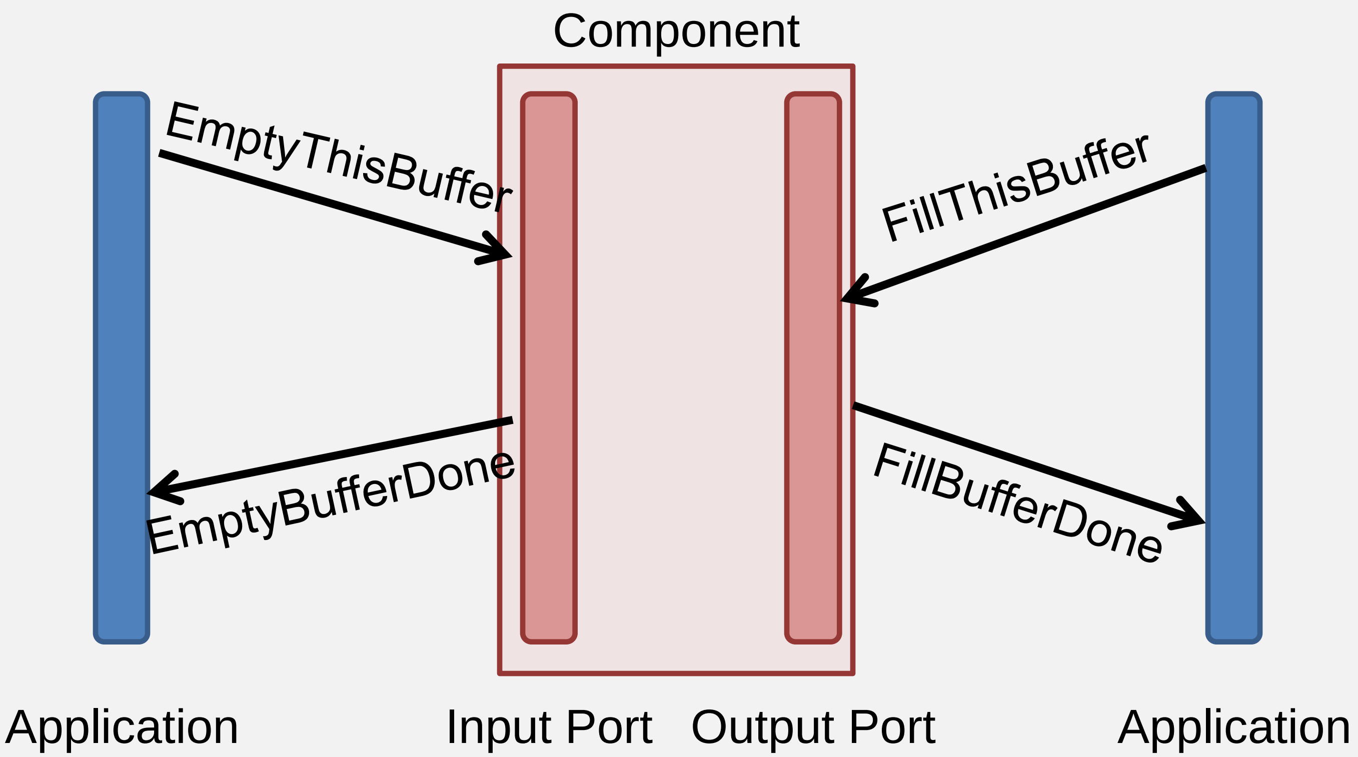
Component Implementation: Configuration Interface

- Application callbacks
  - Callback when errors or other events occur

    e.g. `OMX_EventPortSettingsChanged`
    - can be used to inform application of changes to decoded frame size, etc.

Component Implementation: Data Interface

- `FillThisBuffer()/EmptyThisBuffer()` called from application
- `FillBufferDone()/EmptyBufferDone()` event from component
- Bellagio default implementation (need to customize) through `BufferMgmtFunction()`



Component Implementation: Port Buffer Allocation

- Buffer Allocation
  - `OMX_UseBuffer()` – use application allocated buffers to transfer data
  - `OMX_AllocateBuffer()` – Ask component to allocate the buffers and return pointers to application

Bellagio base will `malloc()` buffers, but you can tailor to your H/W requirements

Bellagio Specific

1. Compile Bellagio
2. `library_entry_point.c`
   - Component name 🡐 should start with “OMX.”
   - Component role 🡐 what does the component do?
3. Compile component into `mycomponent.so`
4. Copy `mycomponent.so` to `/lib/bellagio`
5. Run `omxregister-bellagio` to create `~/.omxregister`

And finally (for this section)

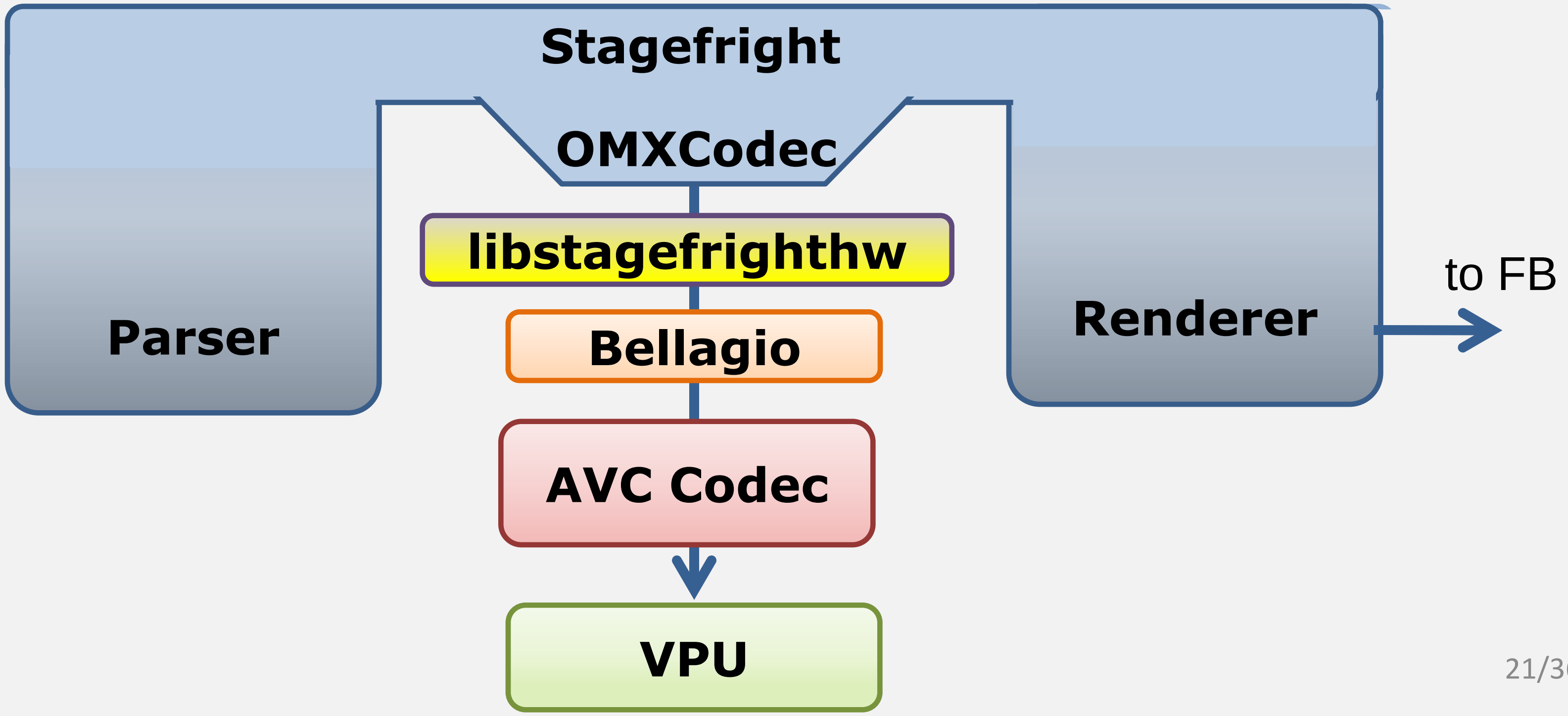
- Access from application
  - via component name from `library_entry_point.c`
- Possible applications
  - Bellagio sample application
  - GStreamer via GstOpenMAX
  - or Android Stagefright

Overview

- Hardware
- OpenMAX IL/Bellagio
- Android Stagefright Integration
  - Video Decoder Specific Considerations

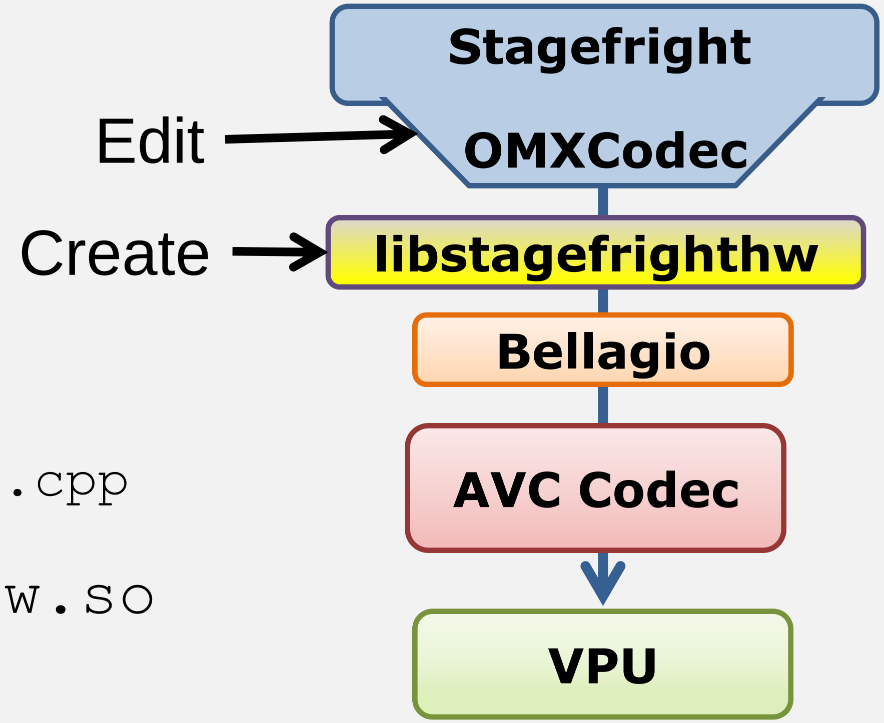
Stagefright Application

- Data input, parsing, and output are supplied natively by Stagefright.
- Link Bellagio to Stagefright through `libstagefrighthw.so`



Linking Bellagio to Stagefright

- Edit `OMXCodec.cpp` **only**
  - Register component
  - Configure component
  - Configure Stagefright
  - `frameworks/base/media/libstagefright/OMXCodec.cpp`
- Create `libstagefrighthw.so`
  - Access to Bellagio
  - see `hardware/xxx/libstagefrighthw`



OMXCodec.cpp: Component registration

Update component list in `OMXCodec.cpp`

```
const CodecInfo kDecoderInfo[] = {
    ...
    {MEDIA_MIMETYPE_VIDEO_AVC,”OMX.mydecode.avc”},
    {MEDIA_MIMETYPE_VIDEO_AVC,”OMX.another.avc”},
    {MEDIA_MIMETYPE_VIDEO_AVC,”AVCDecoder”},
    ...
}
```

Codec name must start with “`OMX.`” so Stagefright knows it’s an external codec

OMXCodec.cpp: Component Configuration

- Additional component settings
  - Stagefright configures most settings automatically
  - Sometimes we need some extra settings
  - Before sending data Stagefright calls `OMXCodec::configureCodec()`
  - Edit `OMXCodec::configureCodec()` to add any codec specific initialization you like

OMXCodec.cpp: Stagefright Configuration

- Customize Stagefright behaviour
  - return value of `OMXCodec::`**`getComponentQuirks`**`(…)`
  - quirks: properties of your component that Stagefright can adapt to.
  - bitmap constants defined in:

    `frameworks/base/include/media/stagefright/OMXCodec.h`

Stagefright configuration (cont): Example `quirks`

- Allocate buffers with OMX_AllocateBuffer() instead of OMX_UseBuffer()
  - `kRequiresAllocateBufferOnOutputPorts`
- No data (pointer) or buffer post-processing req'd.
  - `kOutputBuffersAreUnreadable`
- Output buffers allocated after frame size determined
  - `kDefersOutputBufferAllocation`

libstagefrighthw.so: OMX plugin

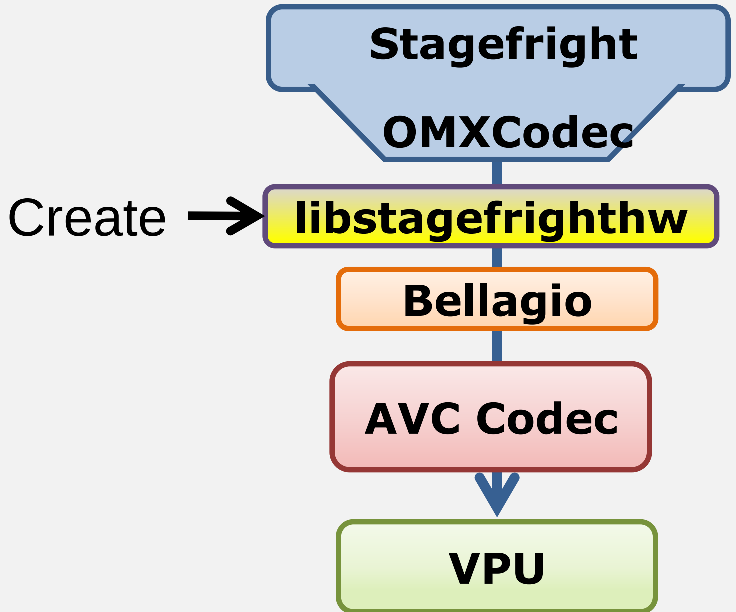
- Create `libstagefrighthw.so` with override of

```
class OMXPluginBase {
     virtual makeComponentInstance(…);
     virtual destroyComponentInstance(…);
     virtual enumerateComponents(…);
     virtual getRolesOfComponent(…);
 }
```

Make a component
Destroy a component
List available components
Get component roles

- Define class factory function

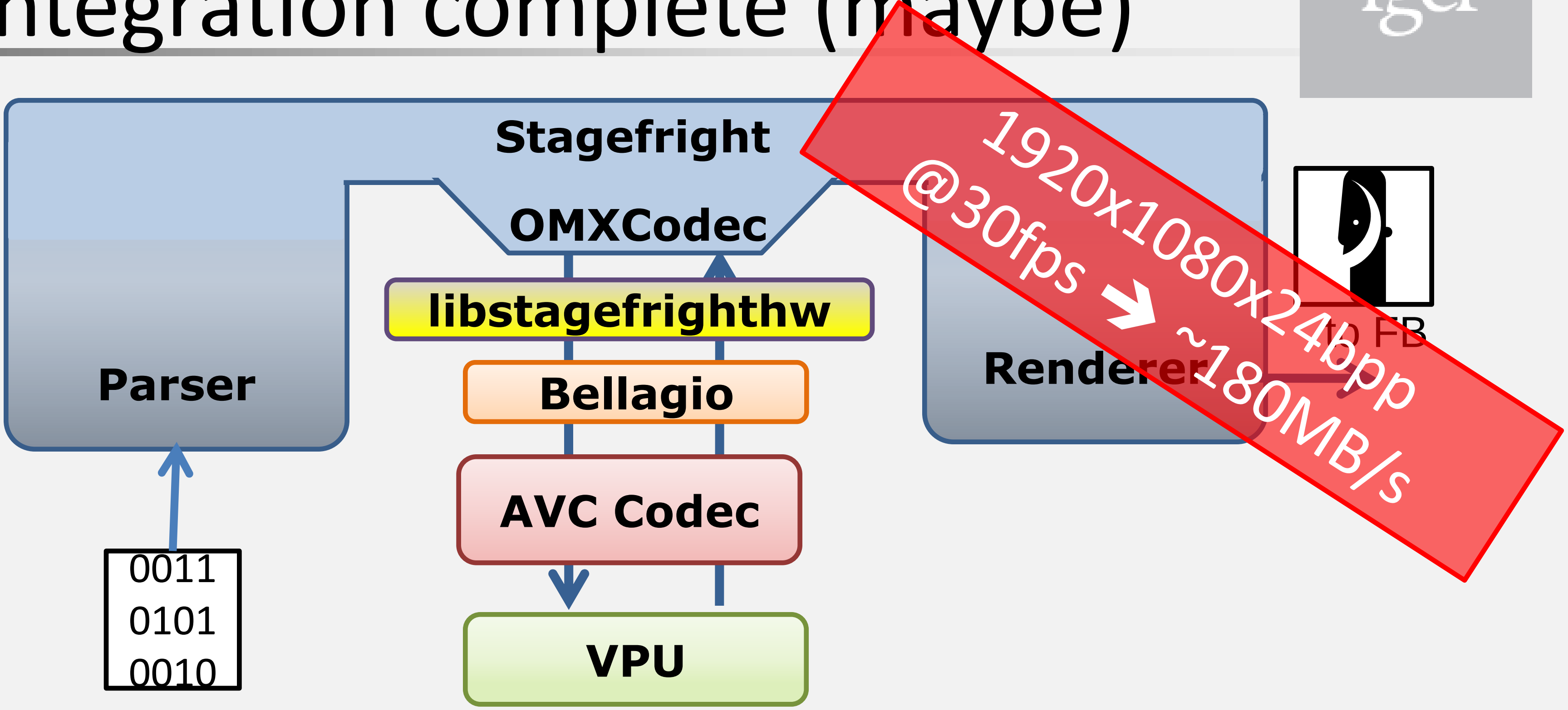
```
OMXPluginBase *createOMXPlugin() {
     return new myOMXPlugin;
}
```



Prepare Bellagio for Stagefright

- Compile Bellagio core/component on Android
  - Must use Android build environment
- Stagefright and Bellagio versions must match
  - include/bellagio/omxcore.h
    - SPECVERSIONMAJOR = 1
    - SPECVERSIONMINOR = 0
- Component registry
  - copy `.omxregistry` to Android rootfs (e.g. `/system/etc`)
  - `export OMX_BELLAGIO_REGISTRY=/<path>/.omxregistry`

Integration complete (maybe)



Can’t we get rid of all this output copying?

Can we process the video fast enough?

Video Decoder Considerations

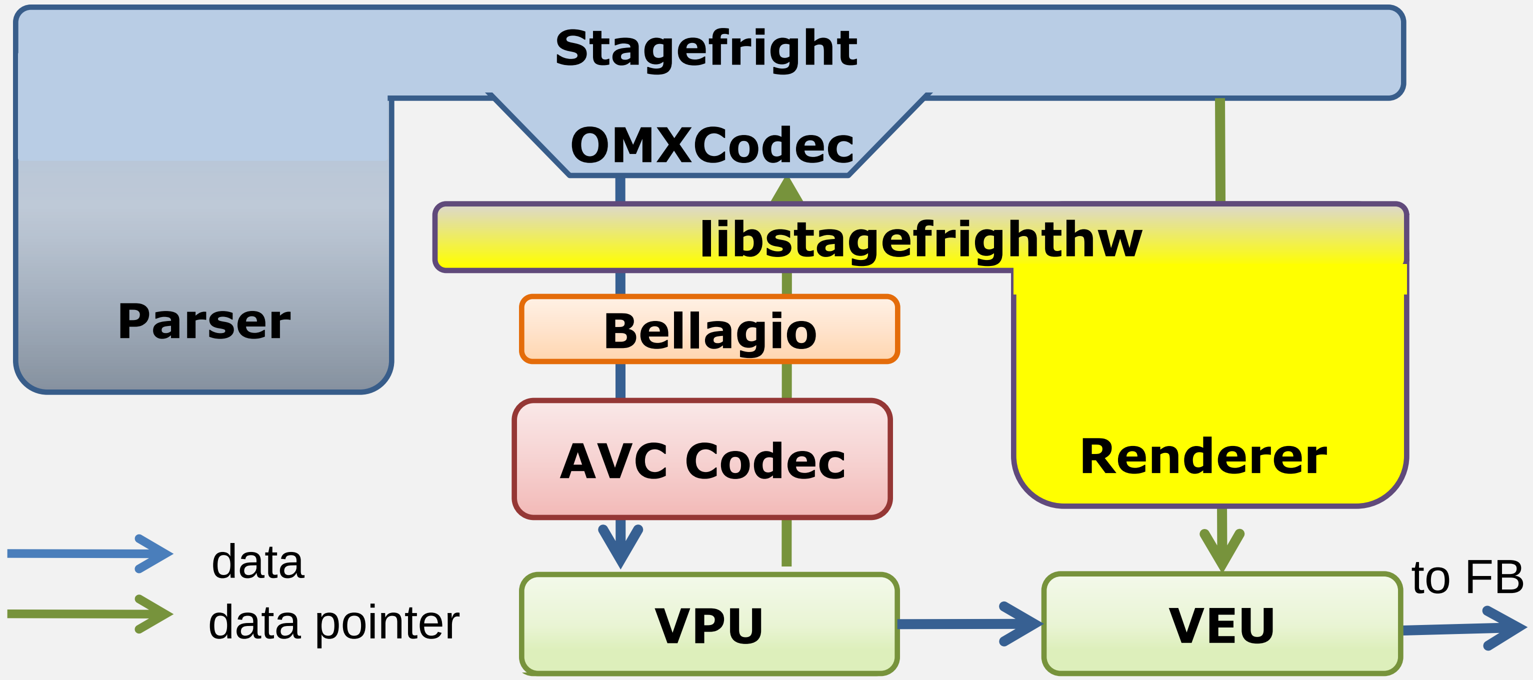
- Custom renderer
  - Default render path requires data copying
  - Custom renderer may avoid copying
  - Might have other uses
- T/L conversion (hardware dependant)
  - Increase memory efficiency and decode speed
  - (Need a custom renderer to use this)

Bypassing default renderer: Custom Renderer

– direct to output device (reduce copying)

– H/W scaling, color conversion

– process custom frame data



Custom Renderer (cont'd)

- Also in `libstagefrighthw.so`
- Renderer is NOT an OMX component
- Override

```
class VideoRenderer {
     virtual VideoRenderer(…);
     virtual render(…,void *platformPrivate);
}
```

Passed up from OpenMAX decoder with each buffer

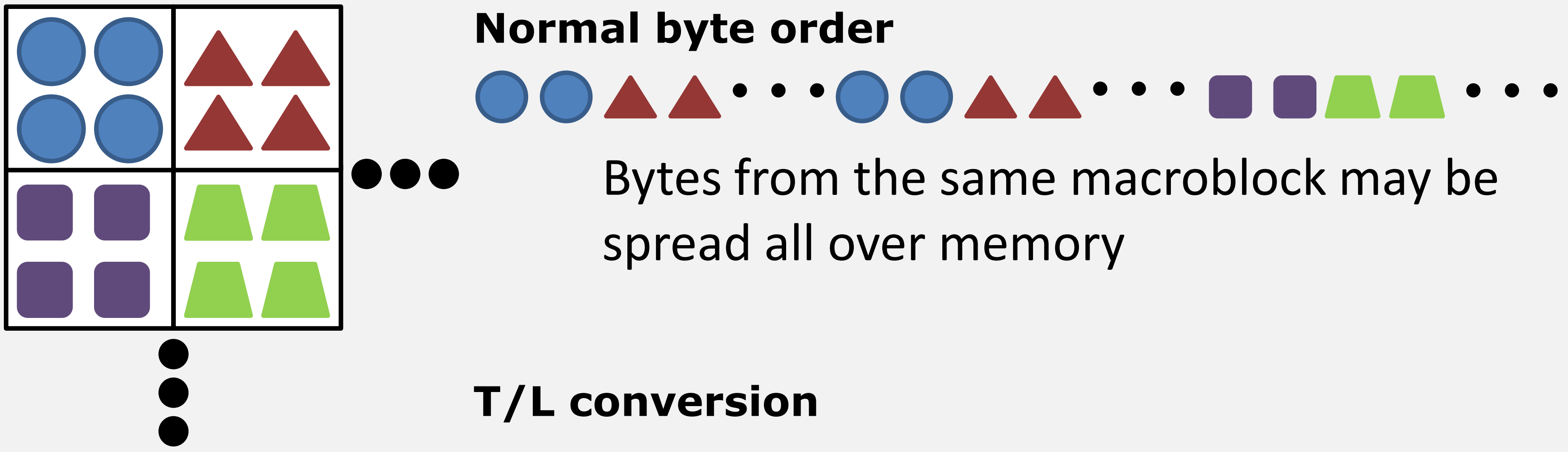
Called to render each decoded frame

- Implement class factory function

```
 VideoRenderer *createRenderer(...) {
     return new MyVideoRenderer(...);
 }
```

Faster video processing: Tiling/Linear conversion

- Tiling/Linear conversion ➔ faster memory access when coding macroblocks



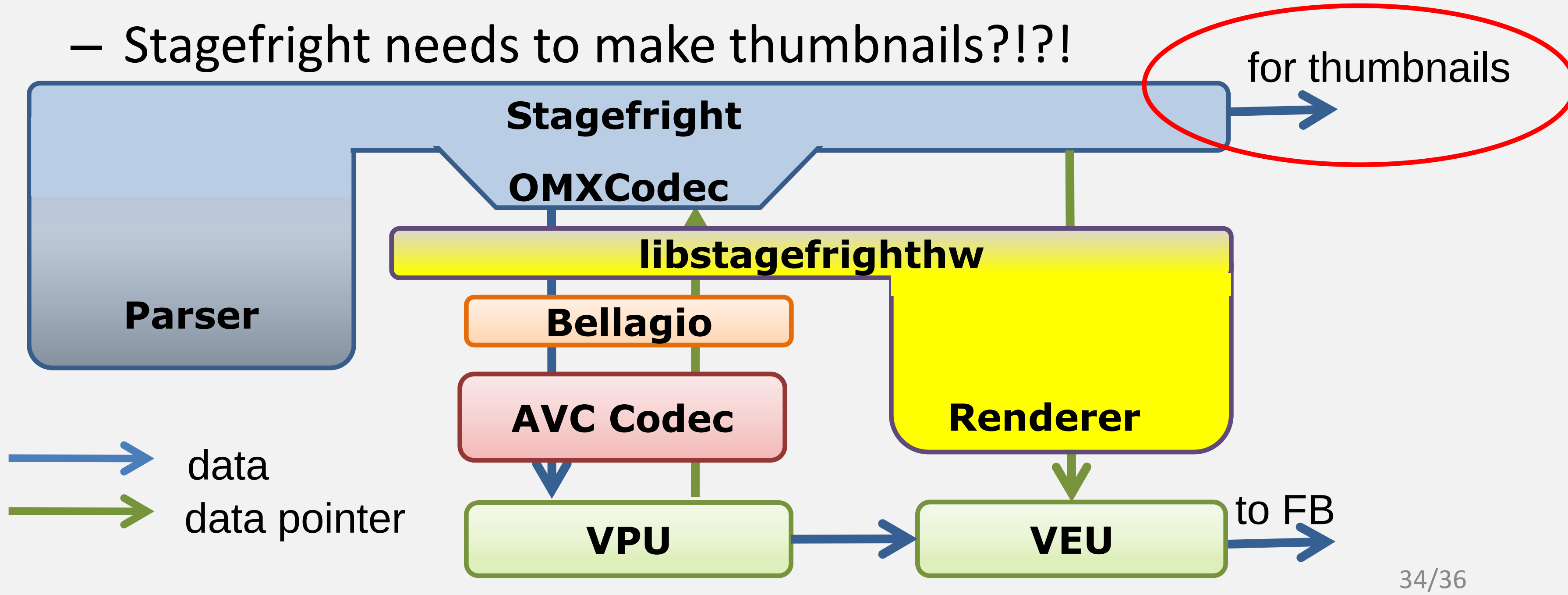
Bytes from the same macroblock may be spread all over memory

Bytes from the same macroblock stay together ➔ faster access (caching, burst memory transfers, etc)

T/L conversion and thumbnails

- Problem
  - When using T/L conversion (or other H/W features) buffers are unreadable by S/W

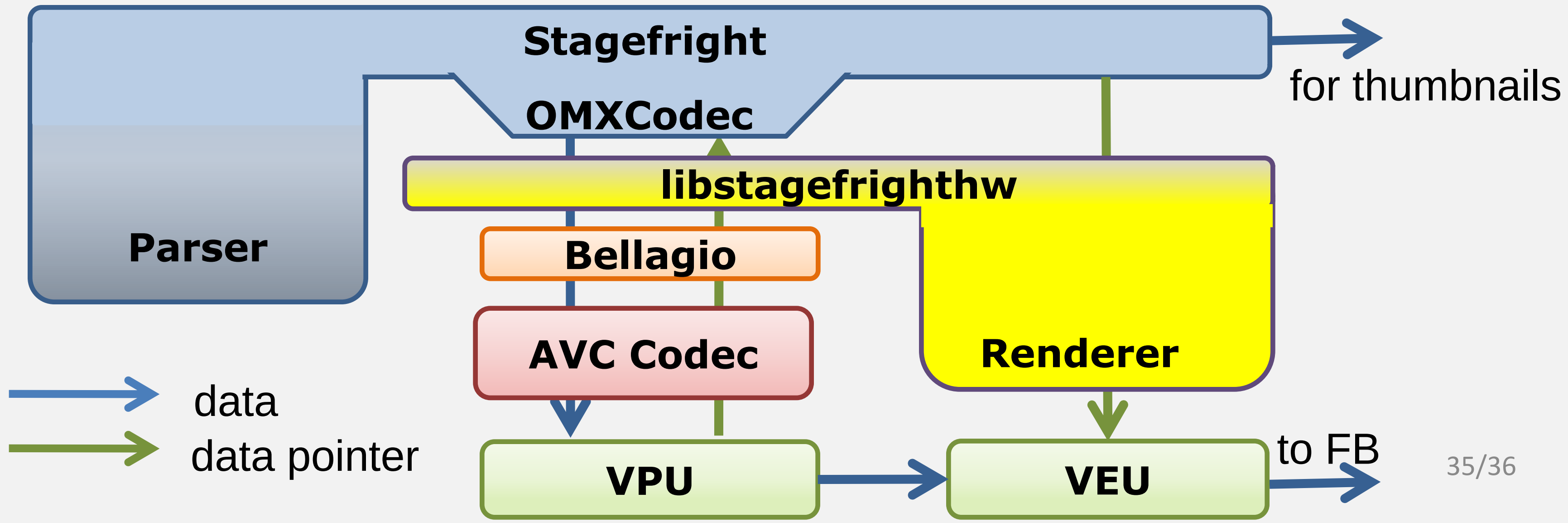
    `(kOutputBuffersAreUnreadable)`
  - Stagefright needs to make thumbnails?!?!



T/L conversion and thumbnails

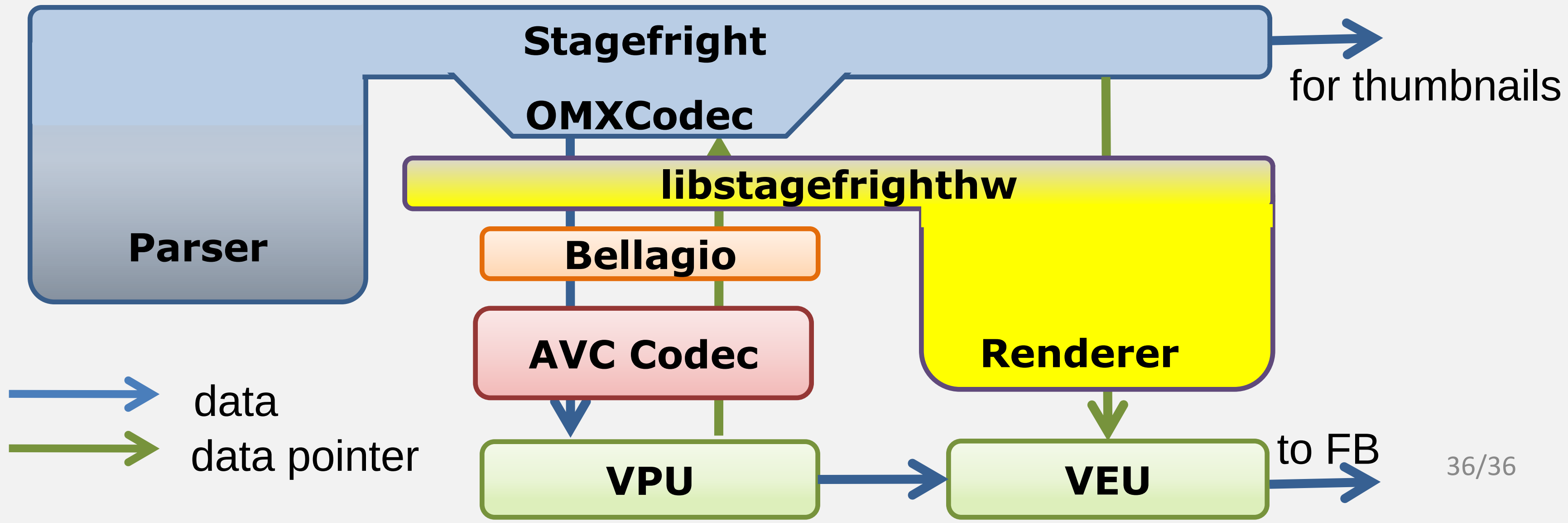
- Solution
  - Thumbnail mode: Stagefright calls `OMXCodec::configureCodec()` with `kClientNeedsFramebuffer` flag set
  - Codec settings can then be adjusted

    ➔ eg. T/L conversion disabled, necessary data copied, etc



Summary

- External video and audio codecs are linked to Stagefright through OpenMAX IL
- Bellagio is a reasonable implementation to use
- Use `quirks` to help with integration
- Check out the examples in the Android source


Stagefright
OMXCodec
for thumbnails
libstagefrighthw
Parser
Bellagio
AVC Codec
Renderer
data
data pointer
VPU
VEU
to FB