Android Media Framework Overview

Jerrin Shaji George

Design Goals

1. Simplify application development

2. Share resources in a multi-tasked environment
   - Android – Multi tasking OS
   - Many applications could request media framework simultaneously

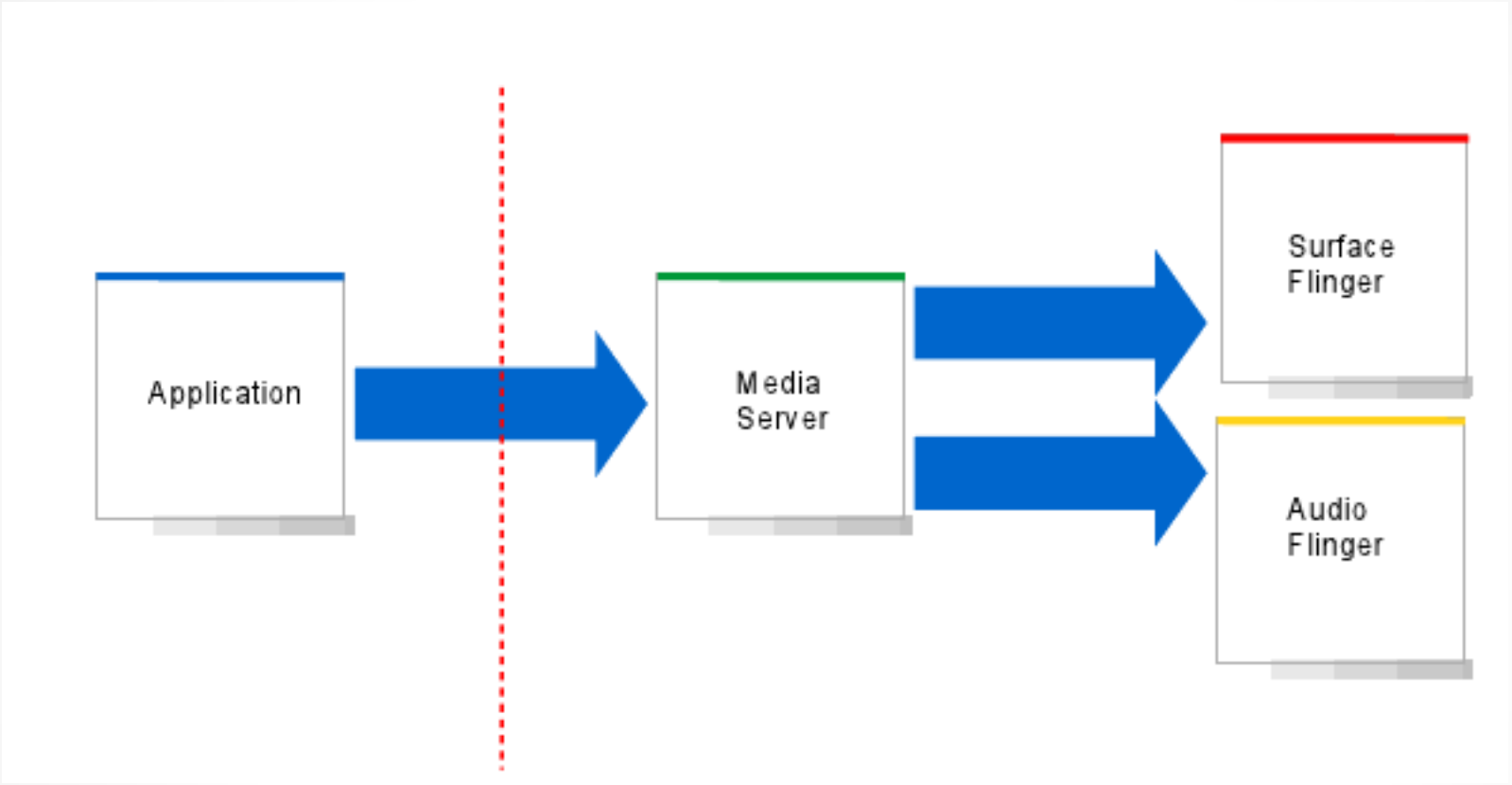
3. Strong security model
   - Framework designed to isolate parts of system vulnerable to hacking
   - Application security ensured by sandbox model

4. Room for future growth
   - Capability to add features which are backward compatible

Sandbox Model

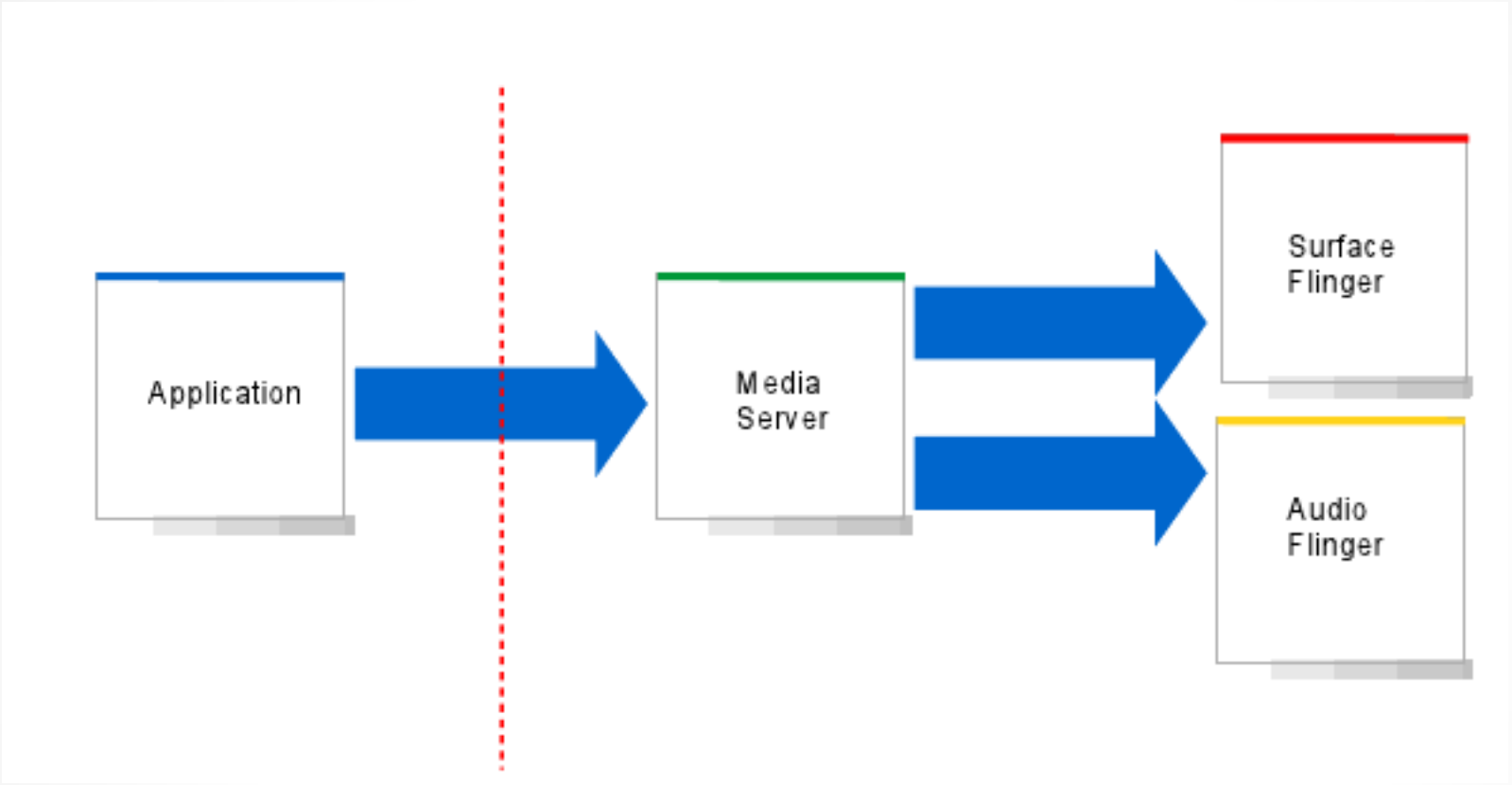
- Each application runs in its own DVM
- Both Dalvik and native applications run within the same security environment, contained within the Application Sandbox.
- Each android application assigned a unique UID. This sets up the application sandbox. By default, an application can access resources contained in its sandbox.

Media Framework Top View



- Android application and the media server run on separate processes
- Media Server is started at boot time
- Media Server : Codecs, file parsers, network stack

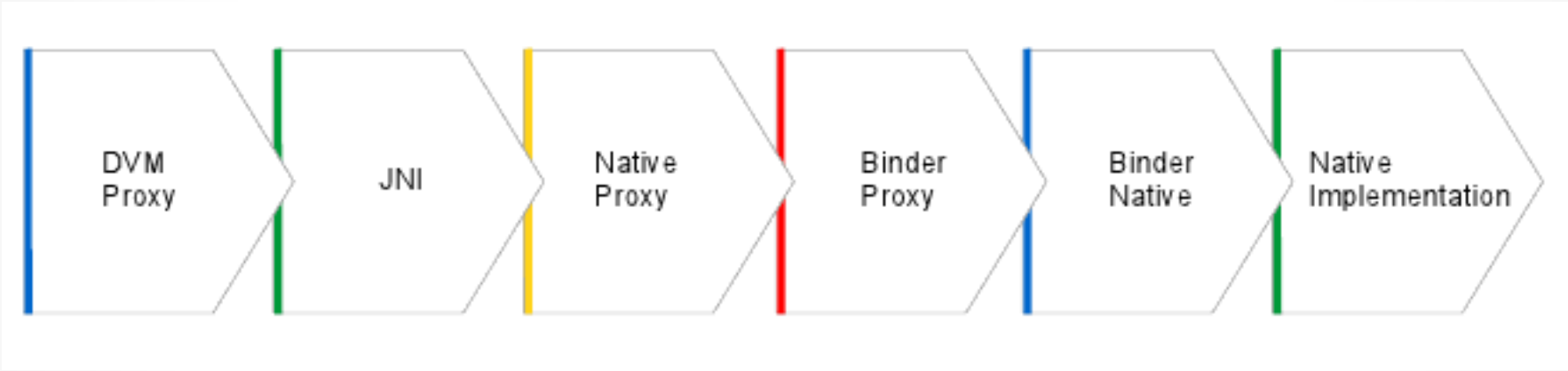
Media Framework Top View



- SF, AF : Hardware abstractions for the audio and video
- Flinger : The software implementation that combines either the visual surfaces or the audio output from multiple applications into a common stream that is sent to hardware.

Typical Stack for Media Function Call

How a media function call from an application make its way down to the framework and into the media engine

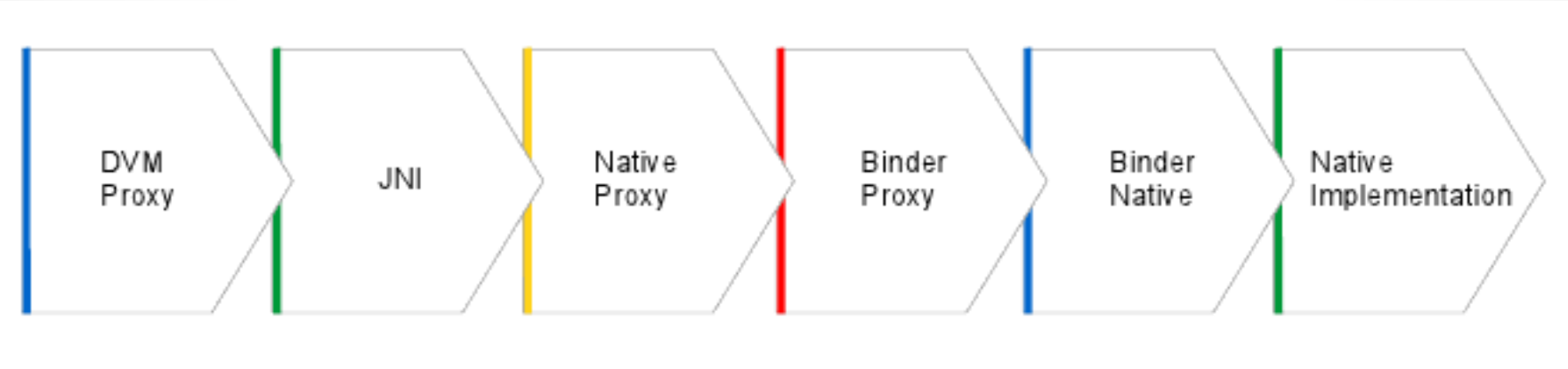


DVM Proxy : The java object, we are talking to

- E.g. – For a media player, it is the media player java object
- It is a proxy for the Native Proxy which in turn is the proxy for the actual native implementation

Typical Stack for Media Function Call

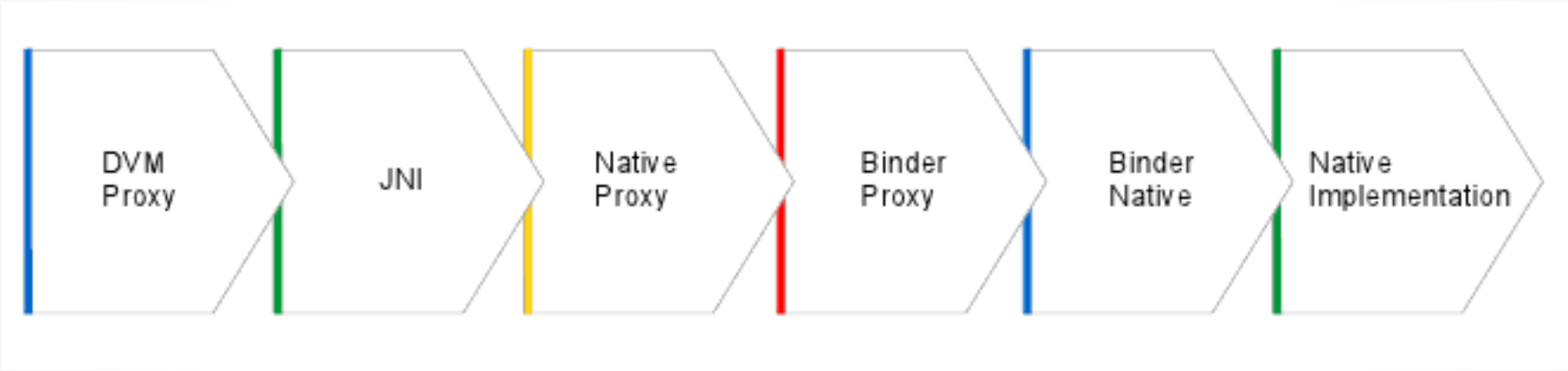
How a media function call from an application make its way down to the framework and into the media engine



- We go through the **JNI layer** (Java Native Interface)
- JNI
  - A programming framework
  - Enables Java code running in JVM to interact with native applications (Programs specific to a hardware and operating system platform) and libraries written in other languages such as C, C++ and assembly.

Typical Stack for Media Function Call

How a media function call from an application make its way down to the framework and into the media engine



- E.g. : Create a media player object in java
  - We make a call through the JNI layer to instantiate a C++ object-
  - The media player java object contains reference to the C++ object, weak references to garbage collection, etc.

Strong references

- Strong reference to an object : The garbage collector cannot collect an object in use by an application while the application's code can reach that object.
- Ordinary java objects are strong by default.
- E.g. :

```
StringBuffer buffer = new StringBuffer();
```

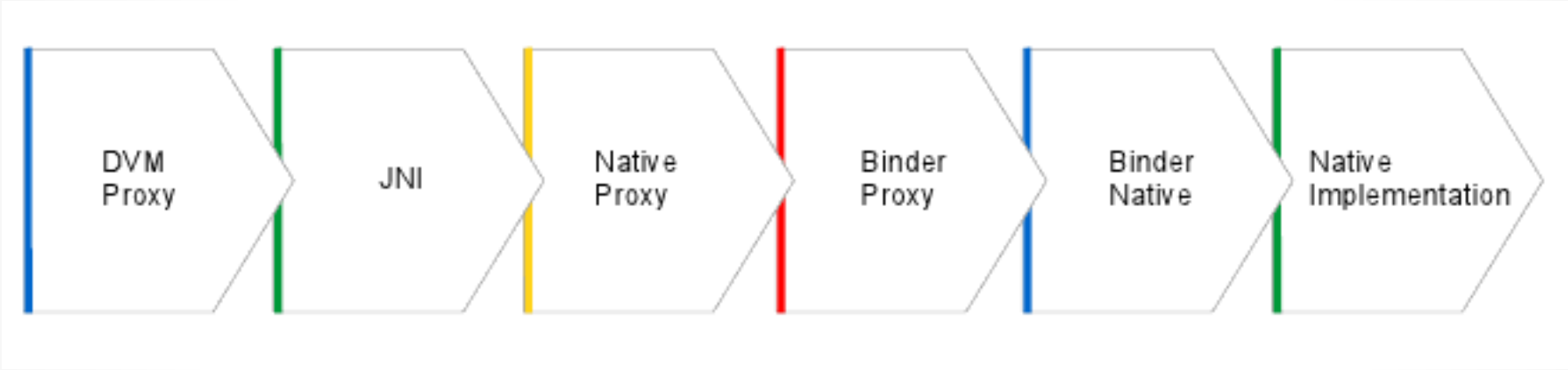
creates a new StringBuffer() and stores a strong reference to it in the variable buffer.

Weak references

- Permits the garbage collector to collect the object while still allowing the application to access the object.
- Useful for objects that use a lot of memory
- Can be recreated easily if they are reclaimed by garbage collection

Typical Stack for Media Function Call

How a media function call from an application make its way down to the framework and into the media engine

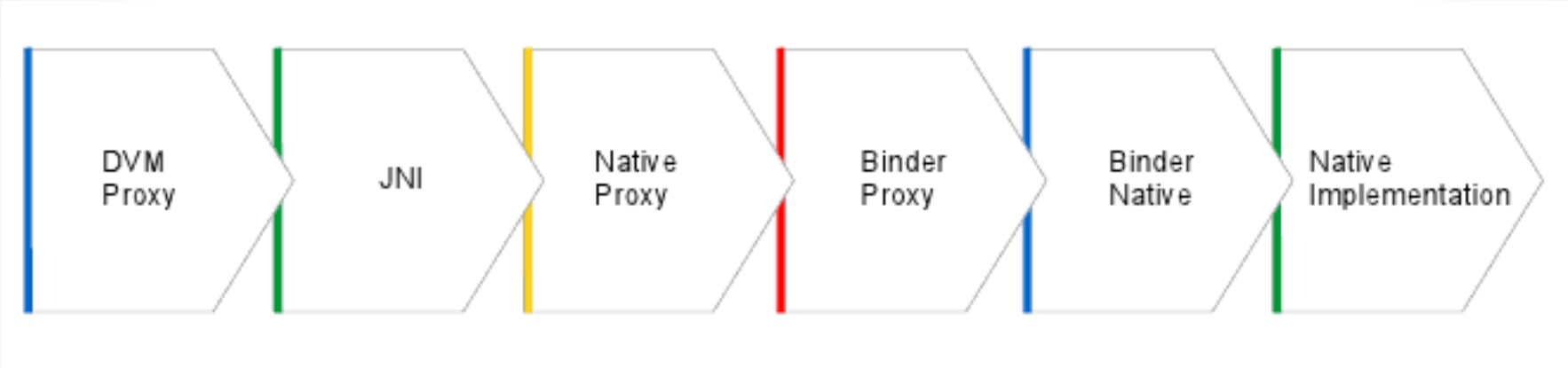


Native proxy

- Proxy object for the media service
- Native proxy is a C++ object that talks through the binder interface

Typical Stack for Media Function Call

How a media function call from an application make its way down to the framework and into the media engine

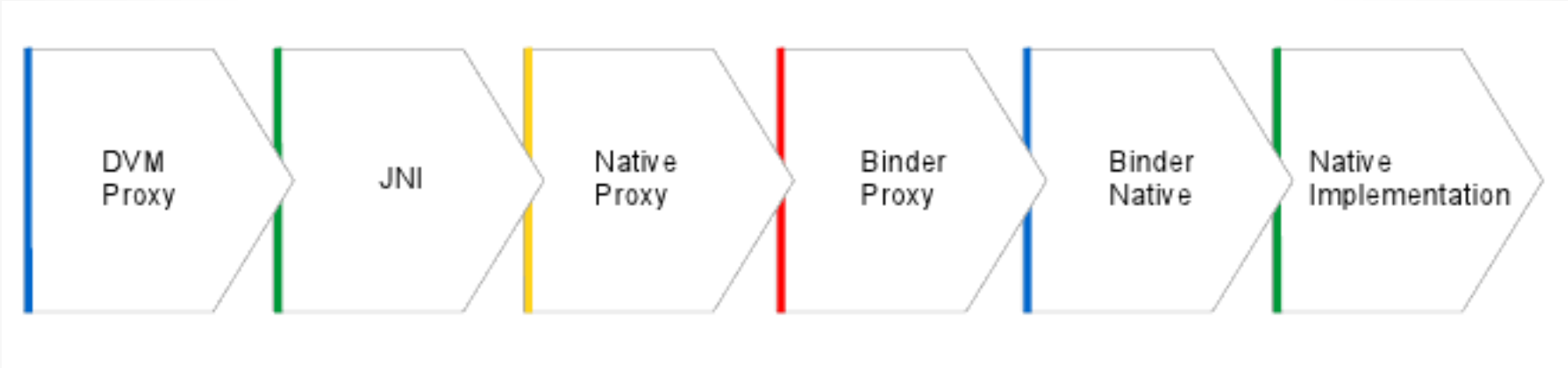


Native proxy

- We could have directly gone to the media server through the JNI layer instead of the proxy. So why do we need Native proxy ?
  - This enables to provide media service access to native applications like games
  - Native applications cannot communicate through JNI

Typical Stack for Media Function Call

How a media function call from an application make its way down to the framework and into the media engine

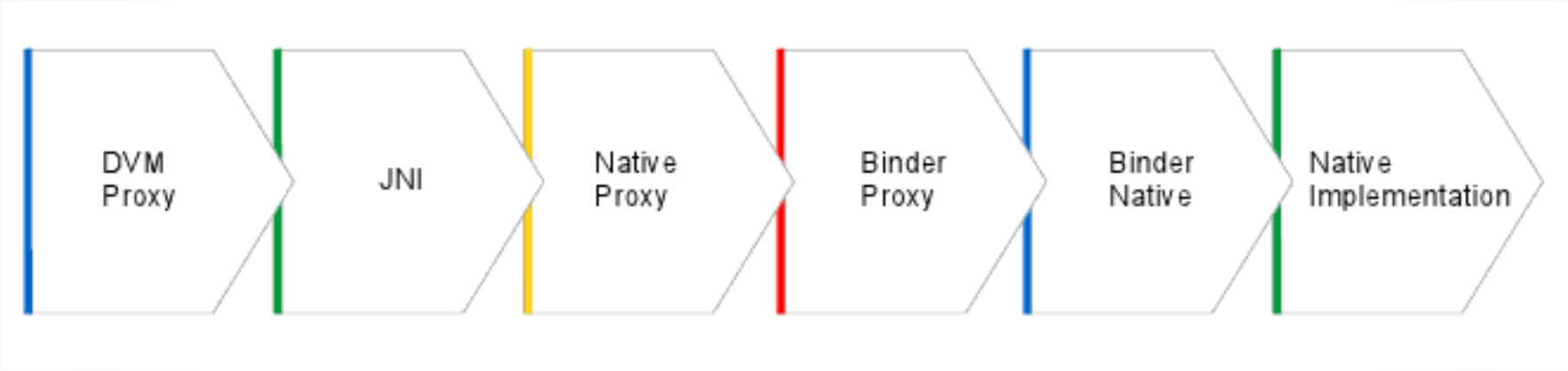


Binder

- Binder is the abstraction for IPC
- It marshals (guides) objects across process boundaries through a special kernel driver
- Processes can share memory - enables moving data back between applications and media player , move file descriptors duped across processes etc.
- Used extensively in Surface Flinger and Audio Flinger

Typical Stack for Media Function Call

How a media function call from an application make its way down to the framework and into the media engine



Binder Proxy

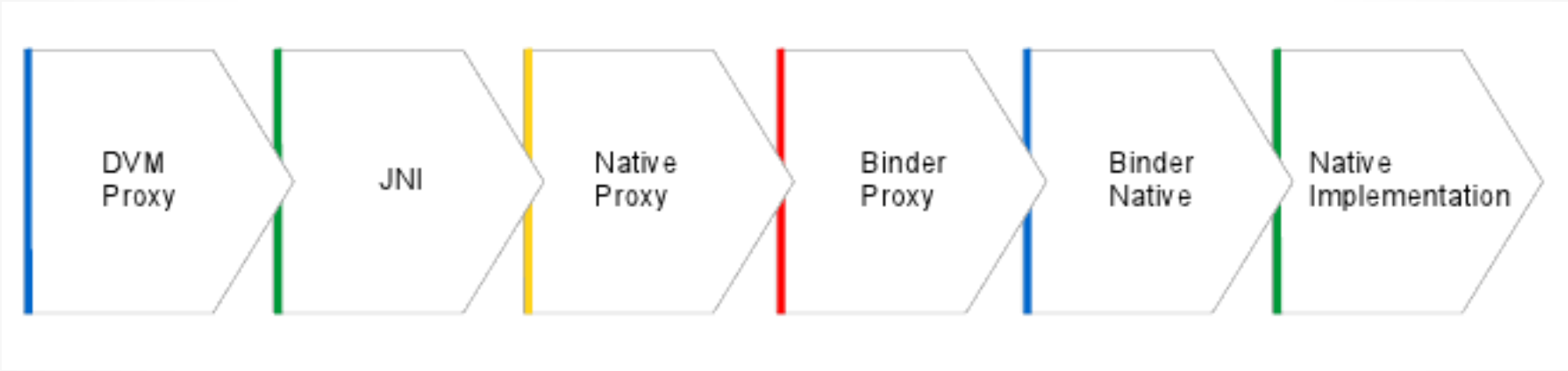
- The marshalling code in the application side

Binder Native

- The marshalling code in the server side

Typical Stack for Media Function Call

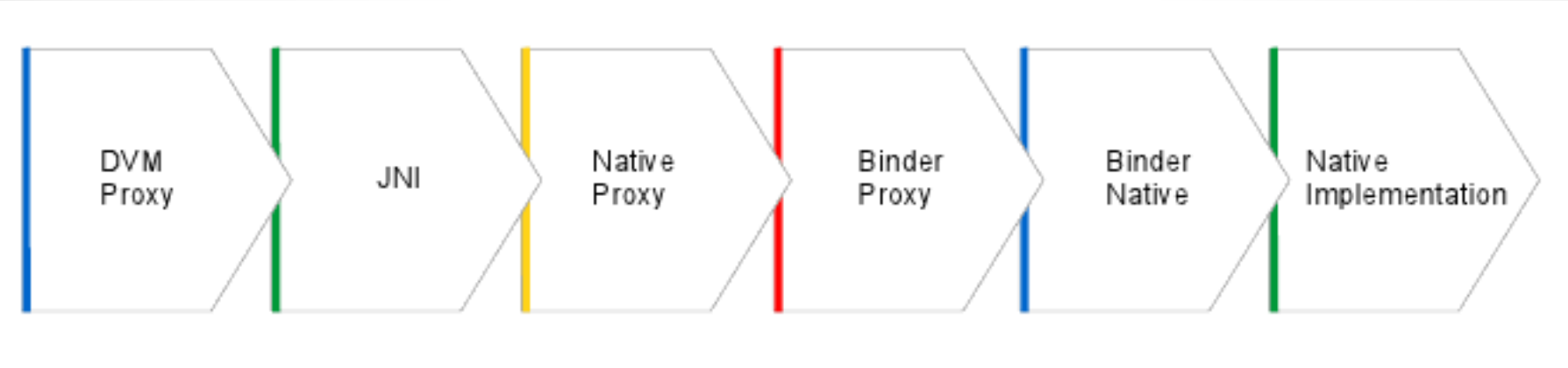
How a media function call from an application make its way down to the framework and into the media engine



Native Implementation

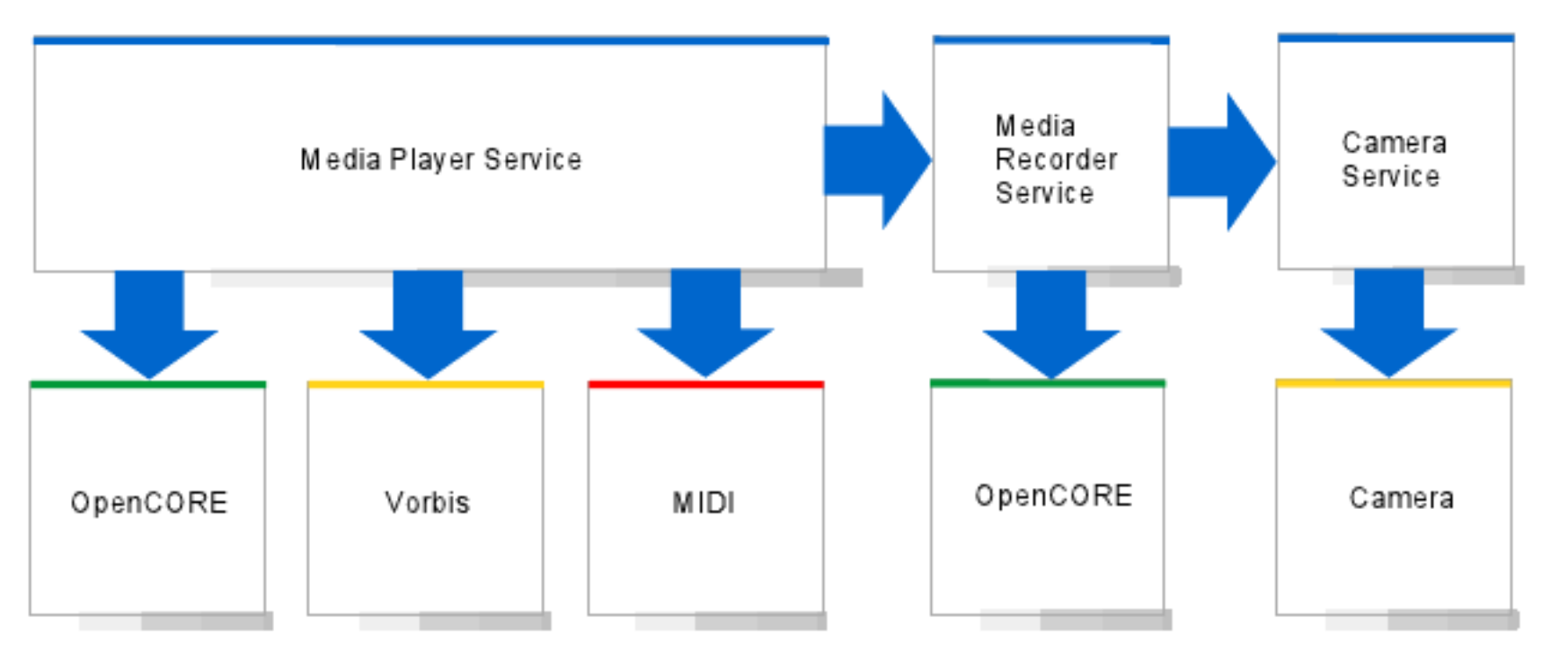
- The actual implementation
- A Media Player service instantiates a MediaPlayer object in the service which is proxied in the application by the MediaPlayer Java object

Typical Stack for Media Function Call



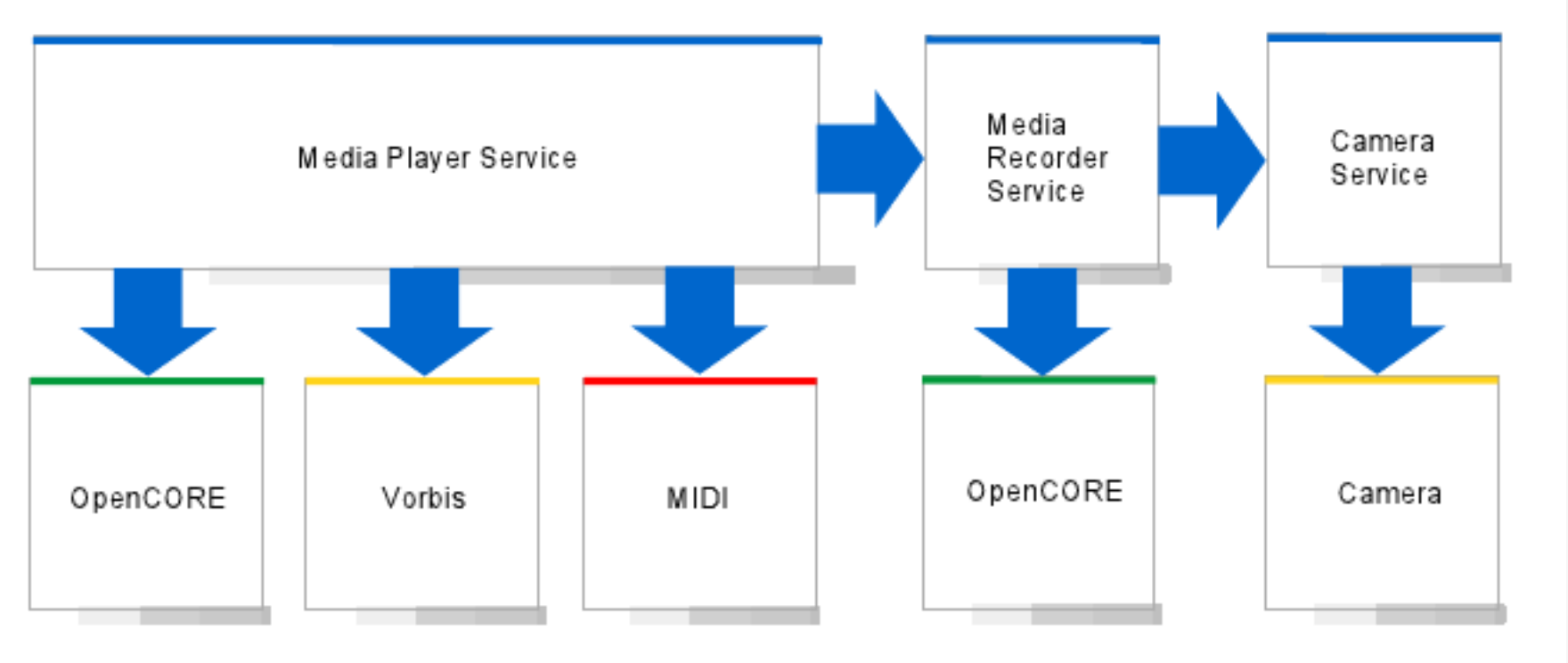
- High overhead involved in making a call through this stack
- This is acceptable since we do not make a lot of calls to the media player service.
- Media player objects must be reused as much as possible

Top Level View of Media Server Process



- It can instantiate a number of MediaPlayer objects
- OpenCore, Vorbis, MIDI : The different media player types

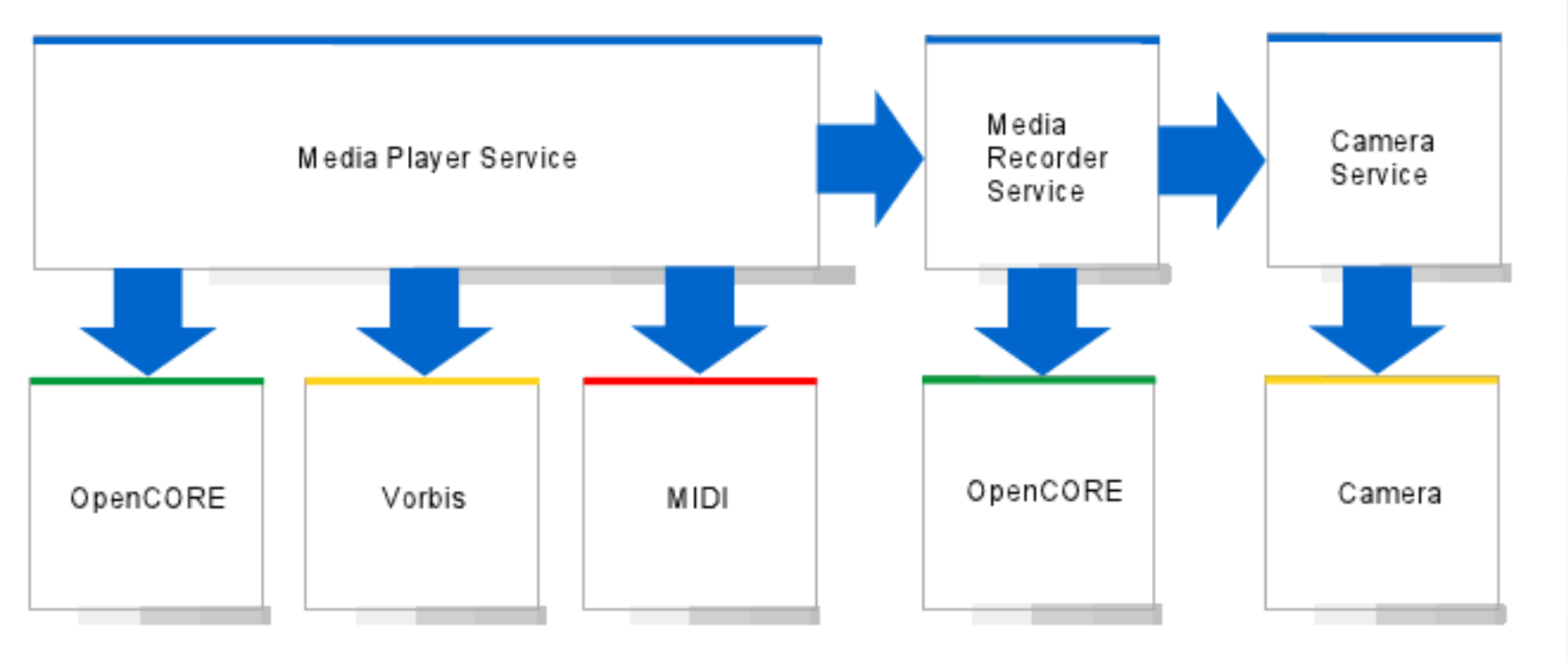
Top Level View of Media Server Process



MediaPlayer

- Each of the media player contains a codec (**co**der-**dec**oder)
- Coder encodes a data stream or signal for transmission, storage or encryption
- Decoder decodes it for playback or editing.
- In a recording device, ADC converts analog signals into digital signals which are then passed into coder

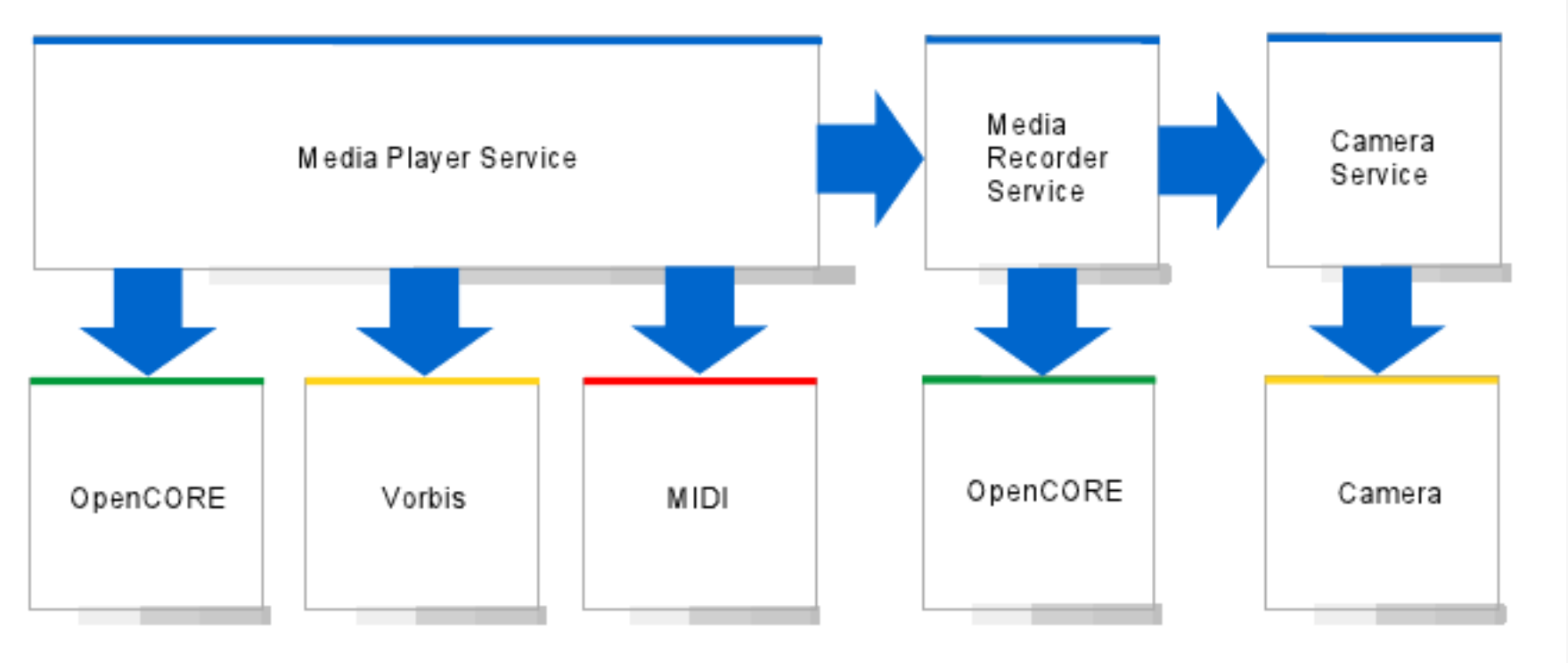
Top Level View of Media Server Process



Vorbis Player

- Plays ogg vorbis files
- A lightweight, psycho-acoustic codec
- Used in internal sounds like ringtones, application sounds /system/media/audio/alarms or ringtones

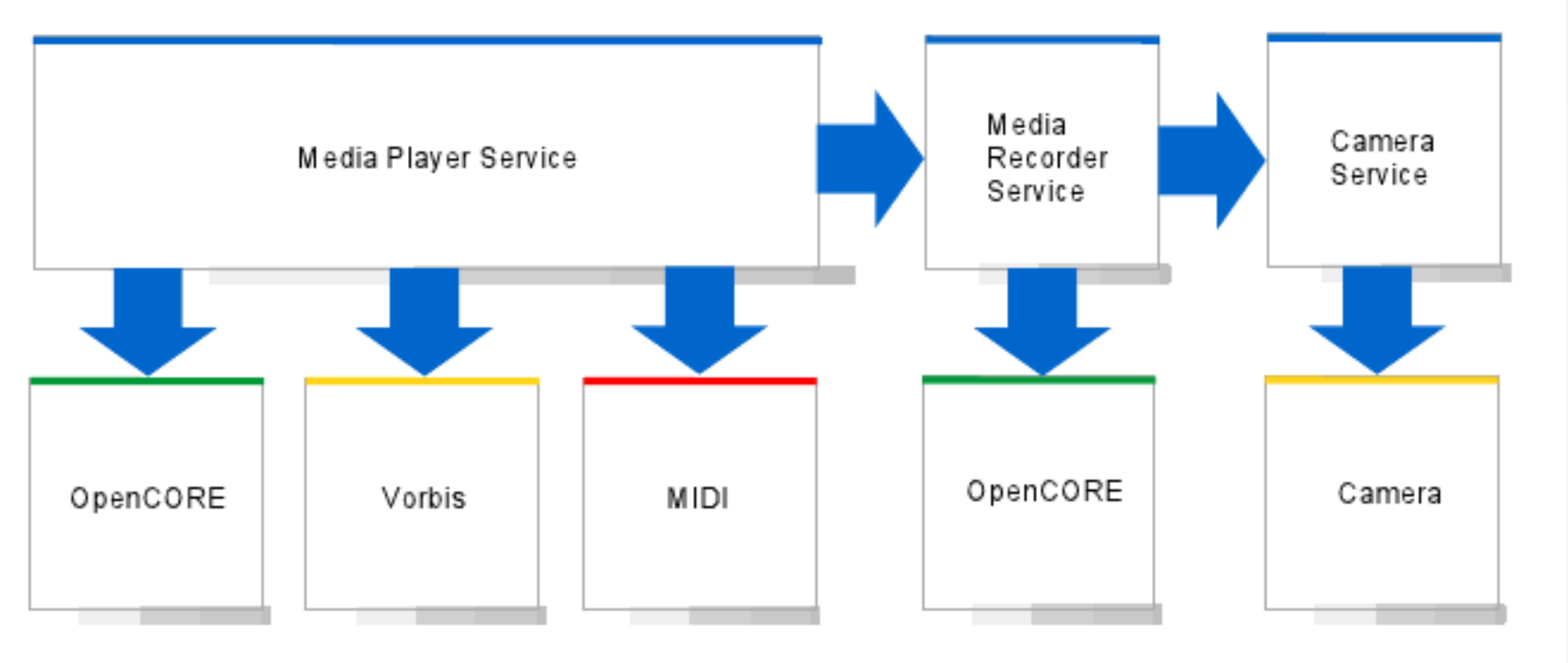
Top Level View of Media Server Process



Pyscho-Acoustic Compression

- Picks out parts of sounds which are going to be masked by other sounds, get rid of them
- Based heavily on human anatomy, a compression algorithm can assign a lower priority to sounds outside the range of human hearing

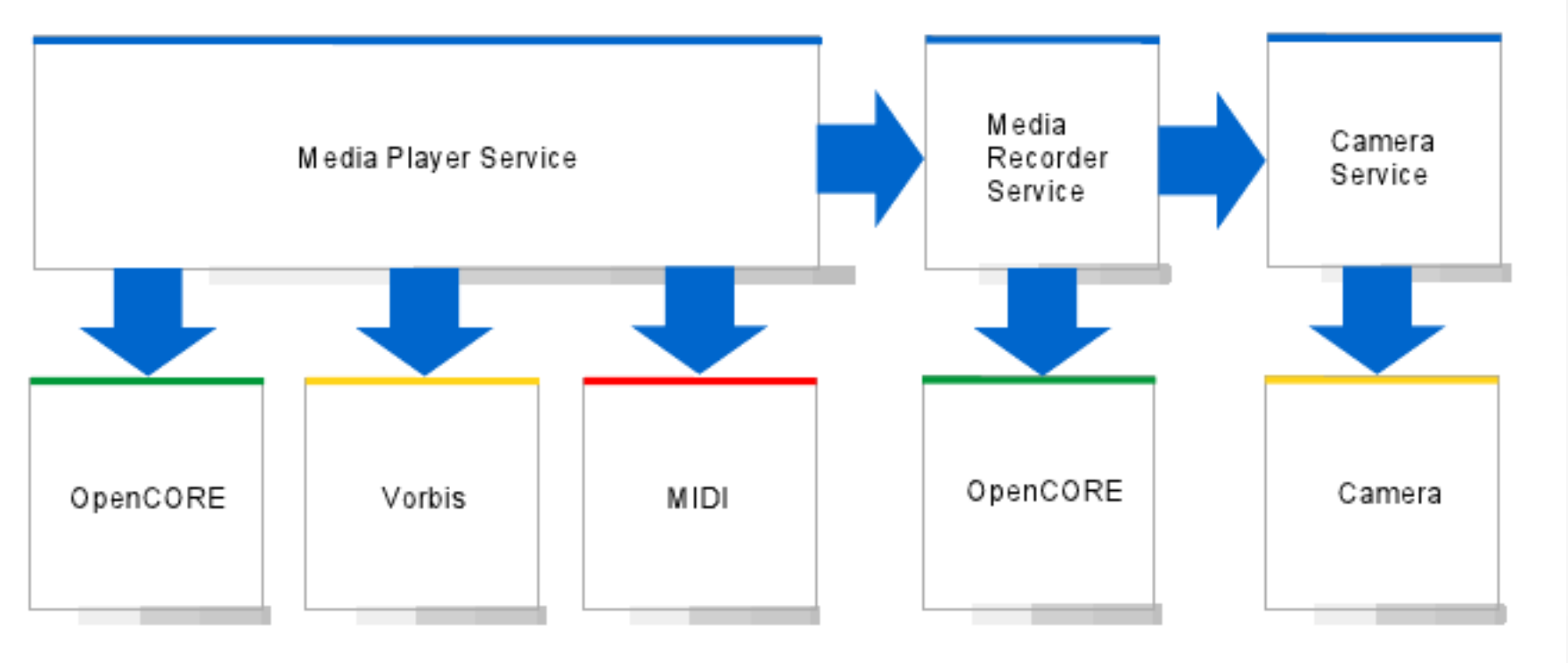
Top Level View of Media Server Process



MIDI Player

- Plays MIDI files
- MIDI : Musical Instrument Digital Interface
  - An electronic musical instrument industry specification
  - Enables wide variety of digital musical instruments, computers and other related devices to connect and communicate with one another

Top Level View of Media Server Process

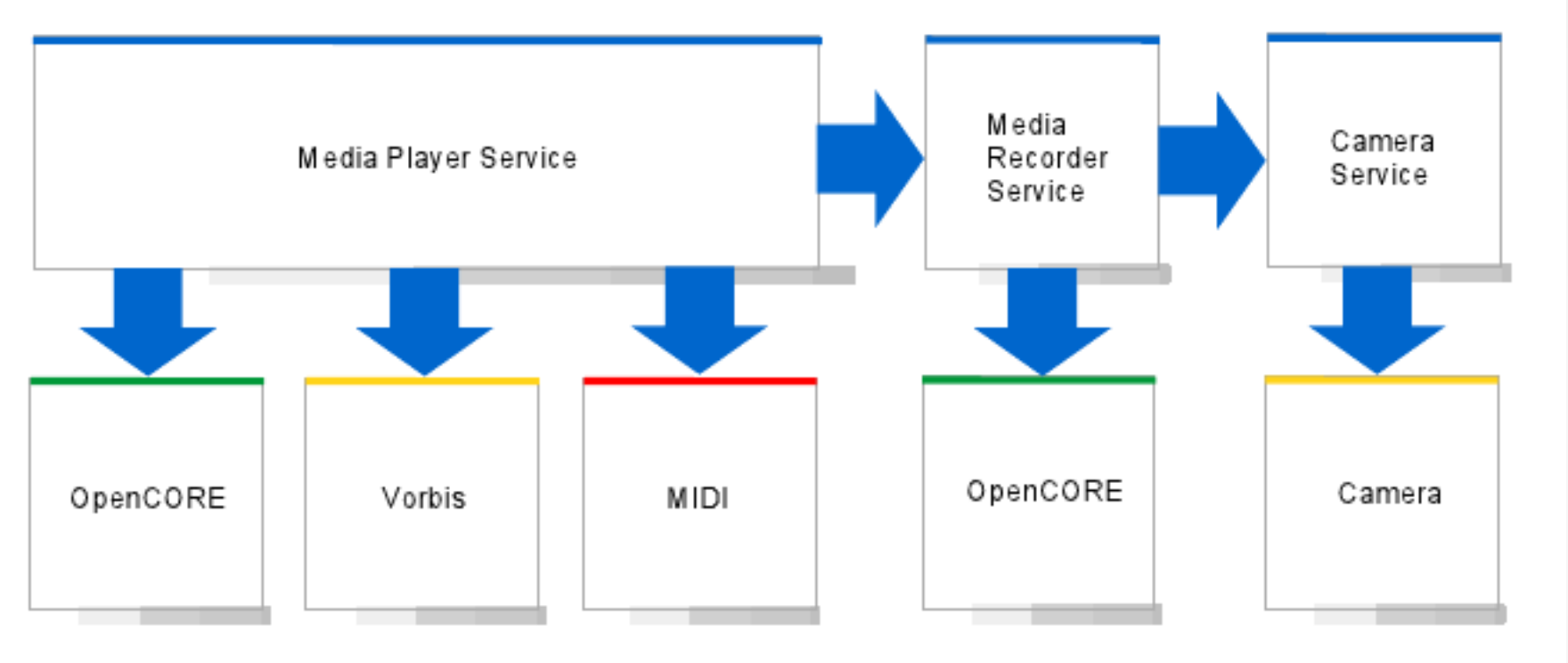


OpenCORE

- Bulk of the framework
- Anything other than Ogg and MIDI routed over to OpenCORE
- Contains major codecs like
  - Audio: MP3, AAC, AMR
  - Video : H.263, H.264, AVC

- **MediaServer is smart enough to recognize file type and call the appropriate player**

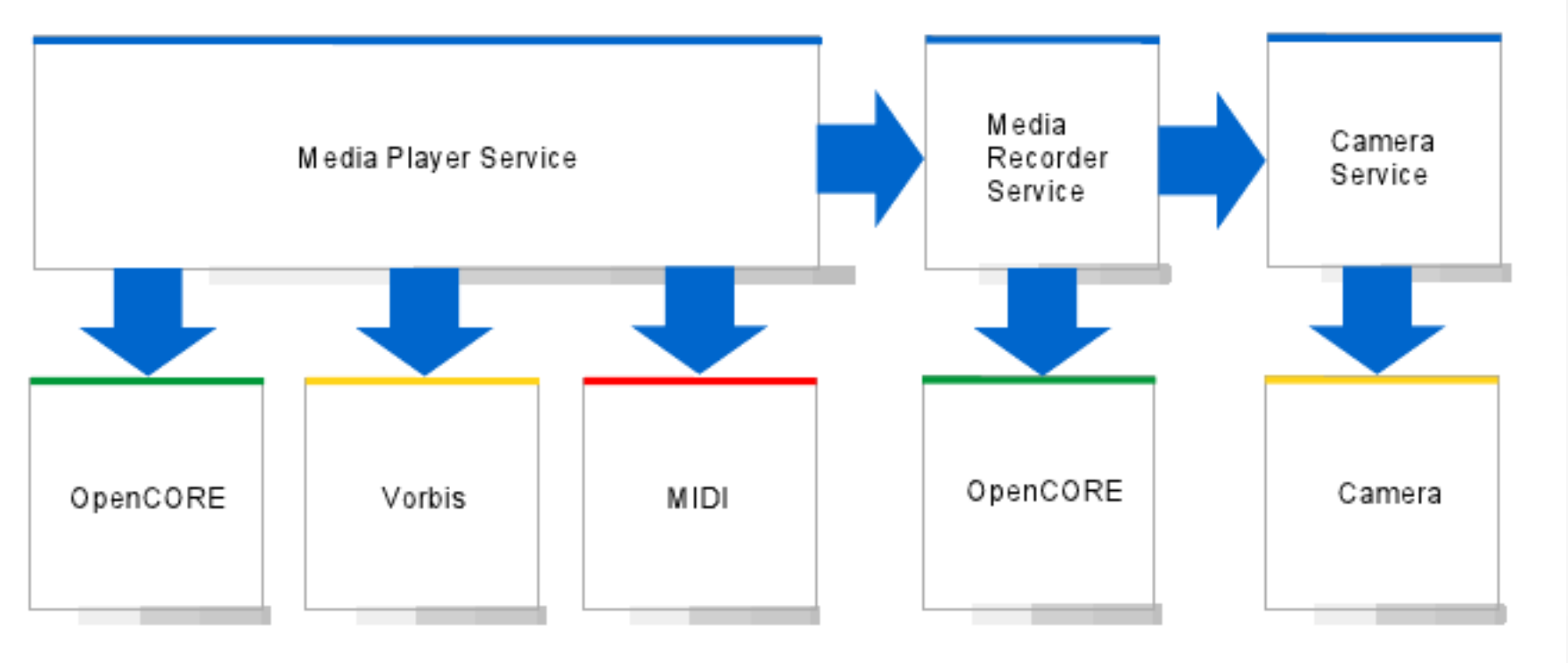
Top Level View of Media Server Process



Media Recorder Service

- It also uses a proxy model
  - A MediaRecorder Object exists in the Java layer
  - MediaRecorder Service does the actual recording
- Integrated with camera service for video recording
- Authoring engine is OpenCORE

Top Level View of Media Server Process

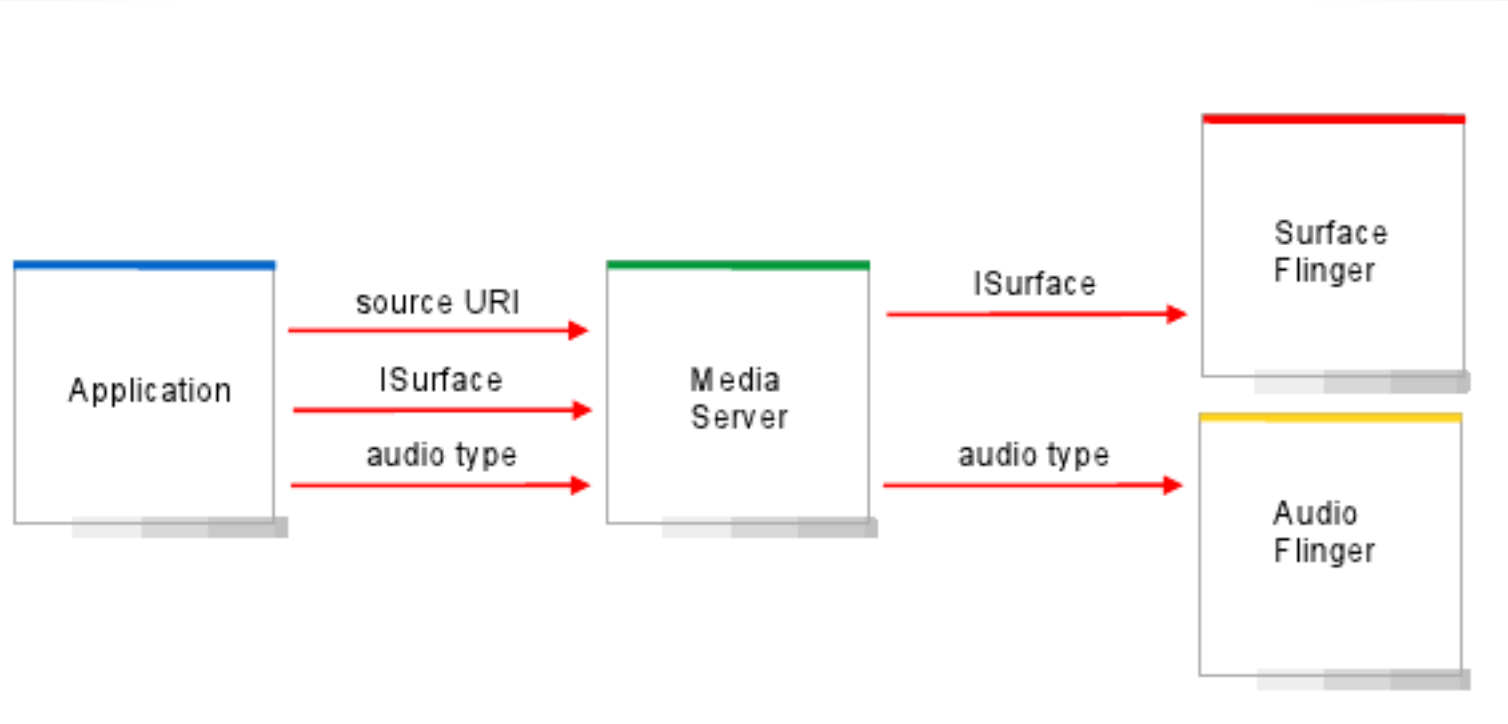


Camera Service

- Operates in conjunction with media service and independently for still images
- To take a still image
  - Instantiate a camera object (a proxy for the camera service)
  - Camera service takes care of handling preview, limiting the amount of data flowing between the application and hardware
  - For Video recording, the frames are sent to OpenCORE which does the encoding

Media Playback

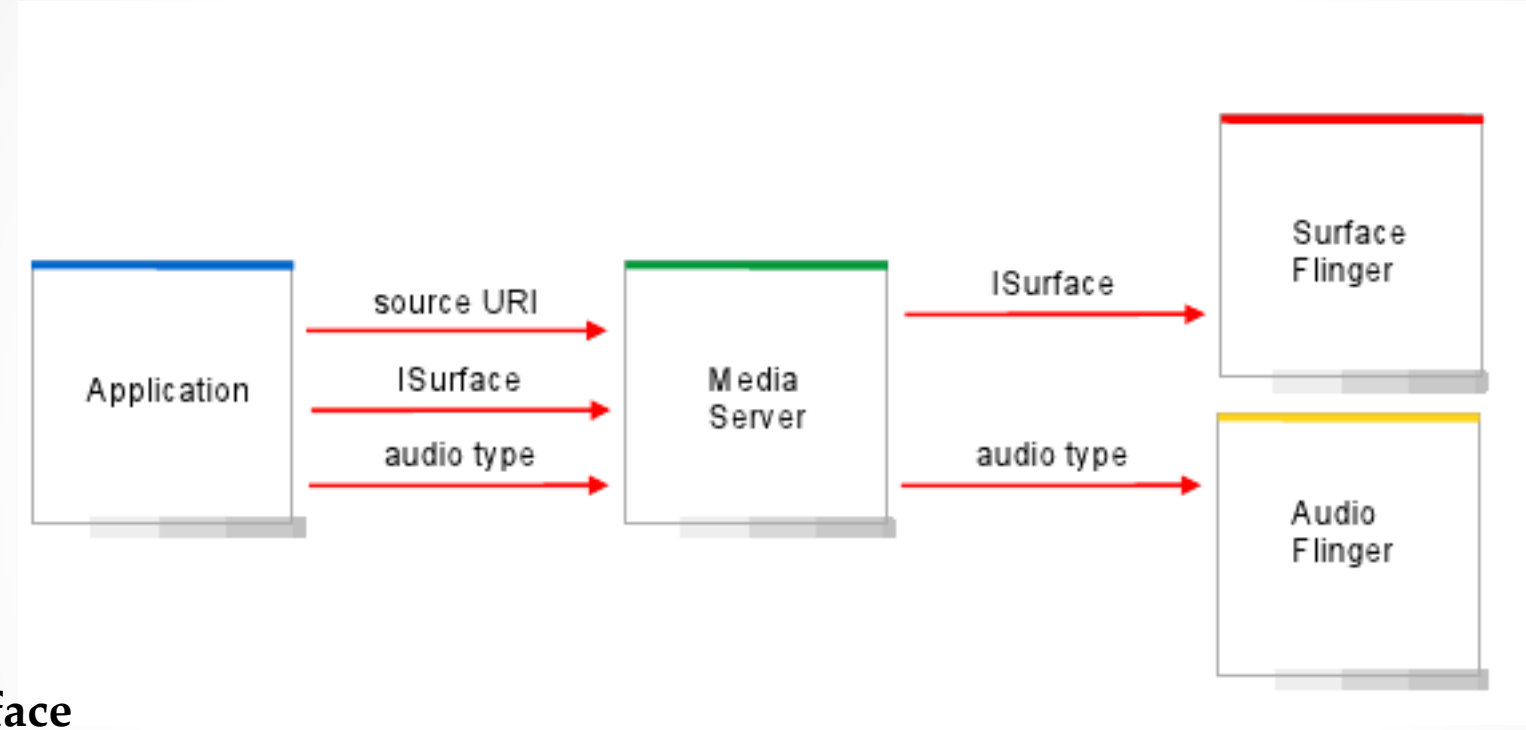
Overview of how a media playback session would look like



- Application provides three main pieces of data
  - Source URI : Location of file – SD Card , a resource in application apk, network stream
  - Surface : The abstraction for the view that you see
  - Audio type : So that media service can route the audio accordingly
- The audio/video frames are decoded inside the media server and they get output directly to the Surface/Audio Flinger
- Low overhead, since no data flowing back to the application

Media Playback

Overview of how a media playback session would look like

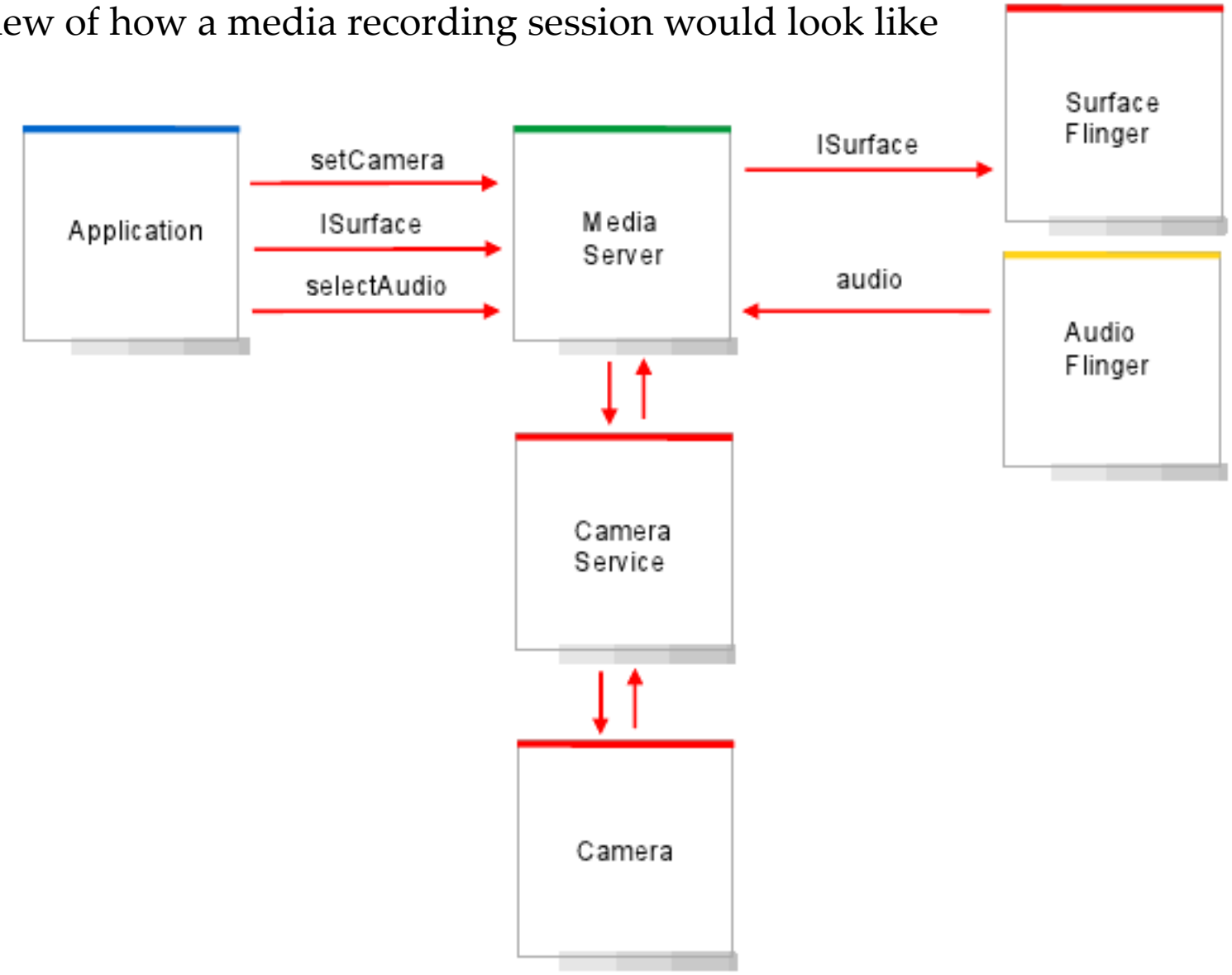


Surface

- An object holding pixels that are being composited to the screen
- Every window on the screen (a dialog, status bar) has its own surface that it draws in to, and Surface Flinger renders these to the final display
- It usually has two buffers to do double-buffered rendering: the application can be drawing its next UI state while the surface flinger is compositing the screen using the last buffer, without needing to wait for the application to finish drawing.

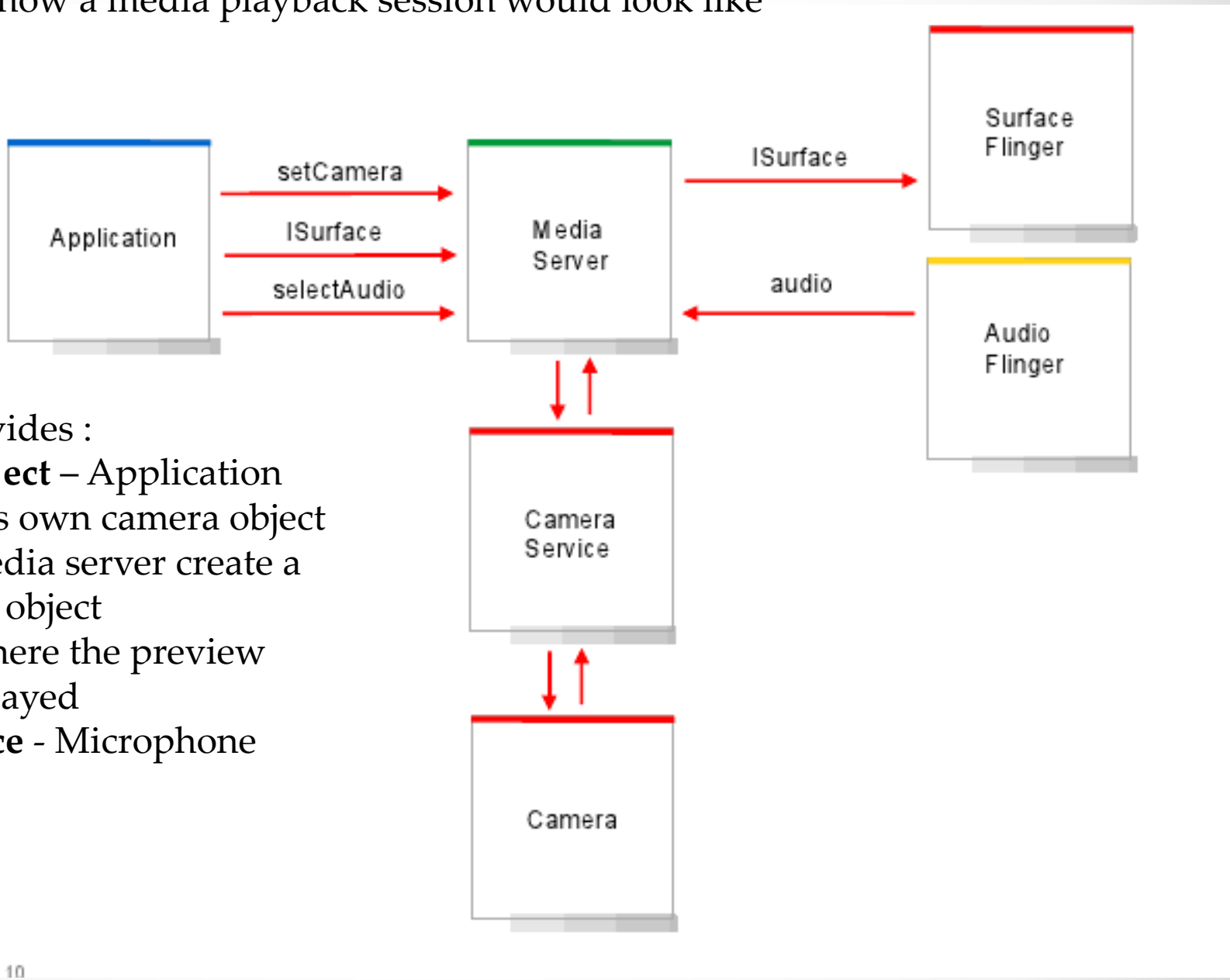
Media Recorder

Overview of how a media recording session would look like



Media Recorder

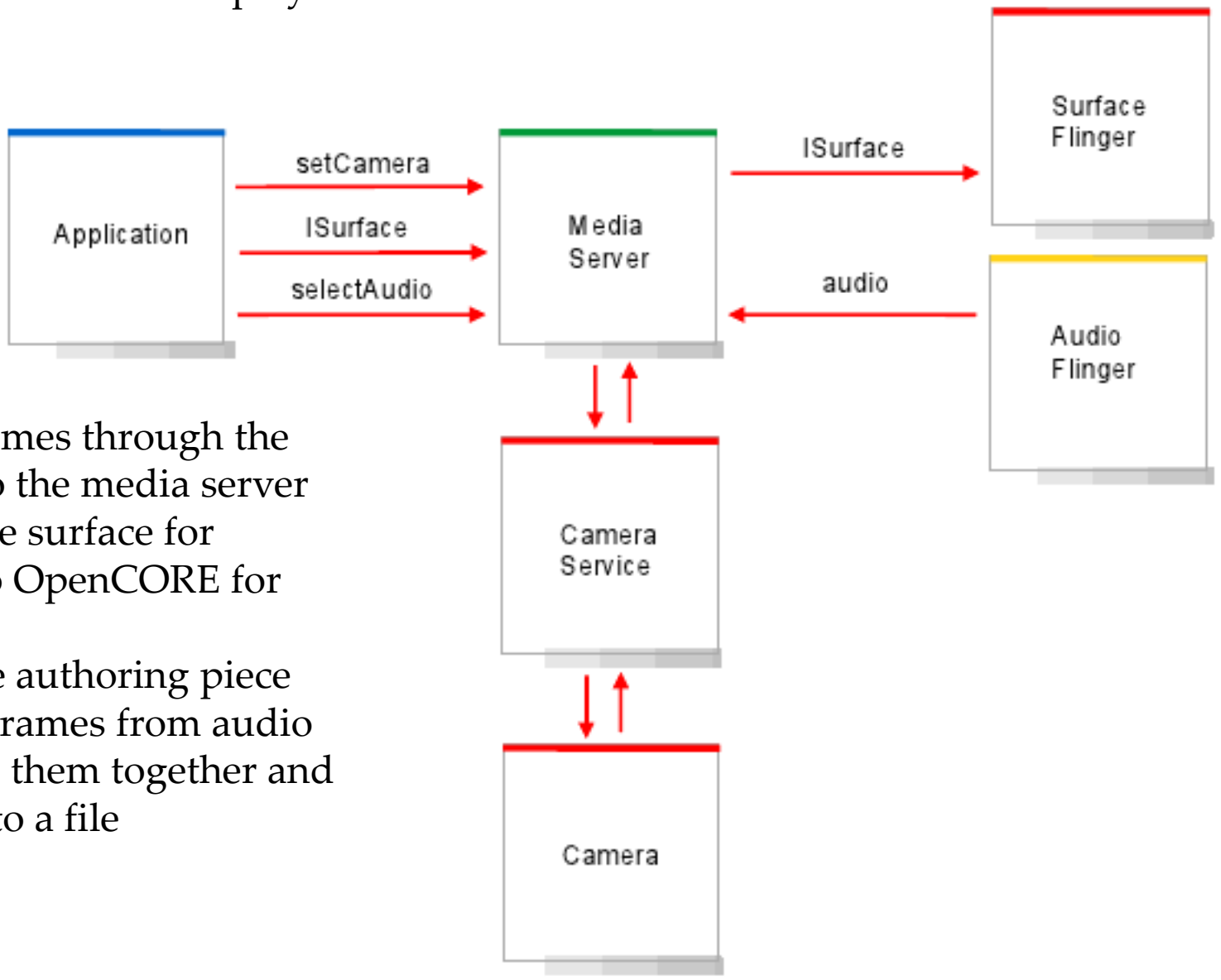
Overview of how a media playback session would look like



- Application provides :
  - **Camera Object** – Application can create its own camera object or let the media server create a new camera object
  - **Surface -** where the preview will be displayed
  - **Audio source** - Microphone

Media Recorder

Overview of how a media playback session would look like



- Camera feeds frames through the camera service to the media server
- It is pushed to the surface for preview and into OpenCORE for encoding
- There exists a file authoring piece which takes the frames from audio and video, mixes them together and writes them out to a file

Codecs

- There are three different types of video codecs
- Terminology :
  - **Bitrate**
    - Refers to the number of bits—or the amount of data—that are processed over a certain amount of time
    - E.g. A 256 kilobits per second audio file has 256 kilobits of data stored in every second of a song.
  - **3GPP**
    - The 3rd Generation Partnership Project
    - Collaboration between groups of telecommunications associations, known as the Organizational Partners.
    - Scope :
      - Make a globally applicable third-generation (3G) mobile phone system specification based on evolved Global System for Mobile Communications (GSM) specifications within the scope of the International Mobile.
      - an evolved IP Multimedia Subsystem (IMS) developed in an access independent manner
  - **MPEG**
    - The Moving Picture Experts Group
    - A working group of experts formed by ISO and IEC to set standards for audio and video compression and transmission

H.263 Video

- Originally designed for low bit-rate video conferencing
- Standardized in 1996
- Part of 3GPP standard, so adopted by number of manufactures
- Used by many websites for web stream, since supported by many mobile devices
  E.g. m.youtube.com
- Simple encoder, decoder
- Used by many streaming sites for low bit-rate video
- H.263 has a deblocking filter
  - A video filter applied to blocks in decoded video to improve visual quality and prediction performance
  - Smoothens sharp edges which can form between macroblocks (usually a 8*8 pixel block) when block coding techniques are used.
  - The filter aims to improve the appearance of decoded pictures.

MPEG4-SP (Simple Profile) Video

- Designed as replacement for MPEG1/2 codecs
- Simple encoder
- Does not have a deblocking filter
- Has singe frame references
  - Reference frames are frames of a compressed video that are used to define future frames.
  - The video encoder can choose the previously decoded frame to base each macroblock in the next frame

H.264 AVC (Advanced Video Codec)

- Better compression (e.g. multiple reference frames)
  - Allows the video encoder to choose among more than one previously decoded frame on which to base each macroblock in the next frame.
- Better quality (e.g. mandatory in-loop deblocking filter)
- Has a number of different profiles. Different profiles target different applications and devices. A profile specifies things like frame sizes, bit rates etc.
- Uses include digital cinema, HDTV broadcast, and mobile
- More complex than H.263 or MPEG4-SP

MP3

- First generation psycho-acoustic compression
- Approximately 10:1 compression @ 128kbps
- Sonic transparency at 192Kbps
  - Frequency at which most people will not be able to hear the difference between the original and compressed version

AAC (Advanced Audio Codec)

- Psycho-acoustic compression like MP3
- Better compression than MP3
- Sonic transparency at 128Kbps
- Commonly used in MPEG-4 streams

Ogg Vorbis

- An open-source codec
- Psycho-acoustic compression like MP3
- Better compression than MP3
- Low-overhead player
  - Lower latency

    Audio latency is the delay between when an audio signal enters and when it emerges from a system.

  - Uses less memory, amount of code to be loaded to play sound is very low, so used internally for ringtones and other applications

- Can loop seamlessly (unlike MP3)
  - MP3 doesn't have a native way of specifying a seamless loop, without a delay

Adaptive Multi-rate (AMR) Audio

- A speech codec, very low bit rate
- High compression rate achieved by focusing on one central tone – throws away a lot of audio
- Two flavours: Narrow band and wide band
- Narrow band is 8KHz input, bit-rates 4.75K to 12.2K bps
- Wide band to 16KHz input, bit-rates 6.6K to 23.85K bps
- Used in 3GP streams
- In the OpenCORE package, AMR Narrow band codec is the only native audio encoder we have in software
- If the hardware does not have an encoder (DSP), this will be the fallback codec
- Audio record applications like MMS use AMR

Digital container format

- The container file is used to identify and interleave different data types
- It concerns more with how data is *stored*, and not necessarily coded
- Simpler container formats can contain different types of audio formats, while more advanced container formats can support multiple audio and video streams, subtitles, chapter-information, and meta-data (tags) — along with the synchronization information needed to play back the various streams together.

Typical 3GPP Stream

- Lower quality
- H.263 video codec
- AMR-NB audio codec
- Bit rates up to 192K bps

Typical MPEG-4 Stream

- Usually higher quality than 3GPP
- H.264 video codec
- AAC audio codec
- Bit rates up to 500K bps

What container/codec type should I use?

- Authoring for Android device, use MPEG4 container with H.264 video (HVGA up to 500Kbps) and AAC 96Kbps
- Creating content on Android device for other Android devices:
  - Use 3GPP container with H.263 video (CIF up to 384Kbps) and AMR-NB audio
- Creating content on Android device for other devices: Use
  - 3GPP container with H.263 video (QCIF up to 192Kbps) and AMR-NB audio
- To stream to an android device :
  - Make sure that ‘moov’ atom is located before ‘mdat’ atom for HTTP progressive streaming
  - moov atom is index of all the frames, which tells the organization of the file
  - mdat : movie data atom
  - Most applications create the moov atom at the end of the file

References

1. Mastering the Android Media Framework, Dave Sparks, Google IO 2009
2. Android Multimedia Framework Overview, Li Li, Solution and Service Wind River