

Android multimedia structure overview

AESOP(<http://www.aesop.or.kr>)

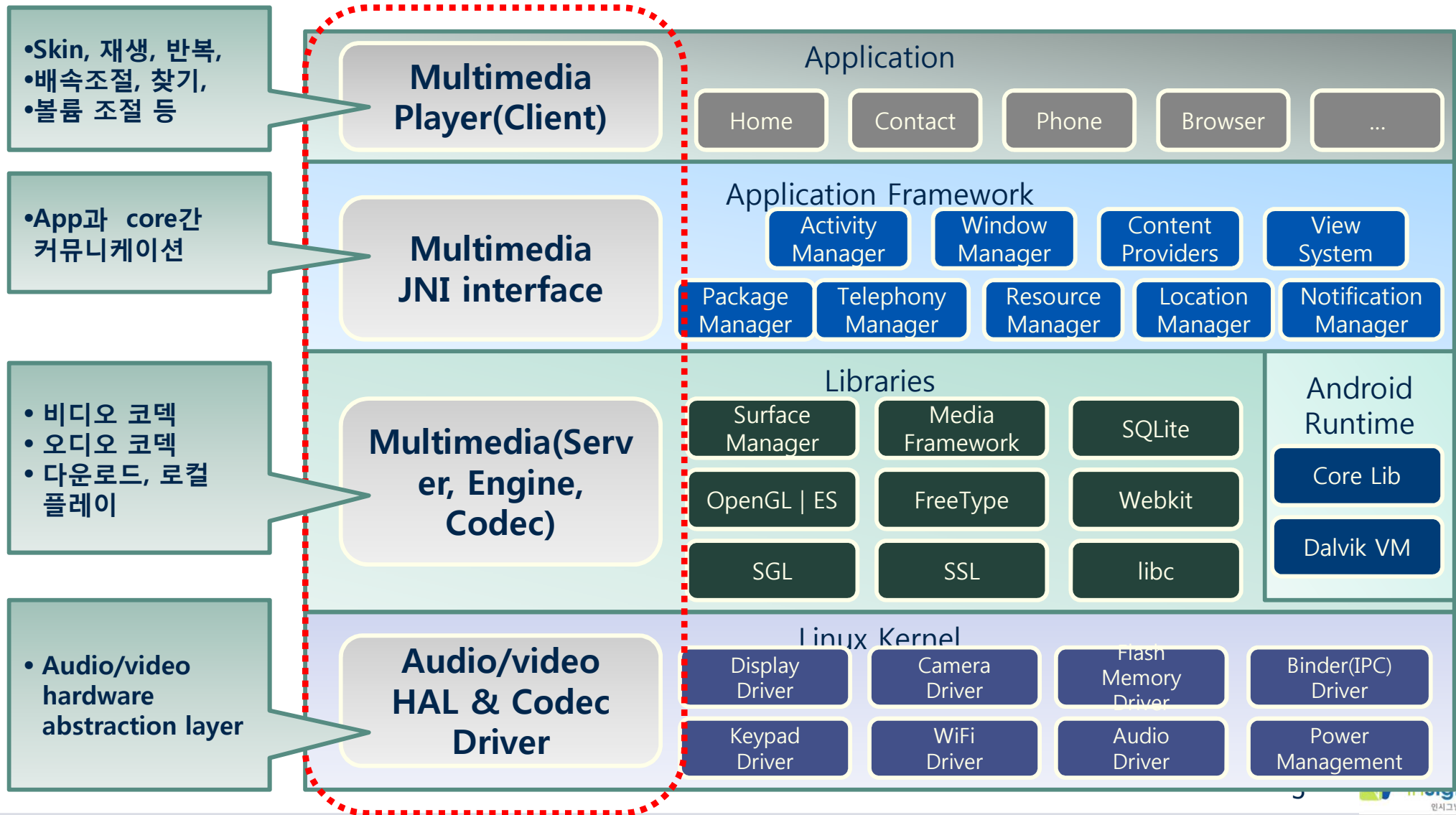
2011.09.09, 고헌철(고도리)

개요

- Android multimedia framework 구조 및 flow

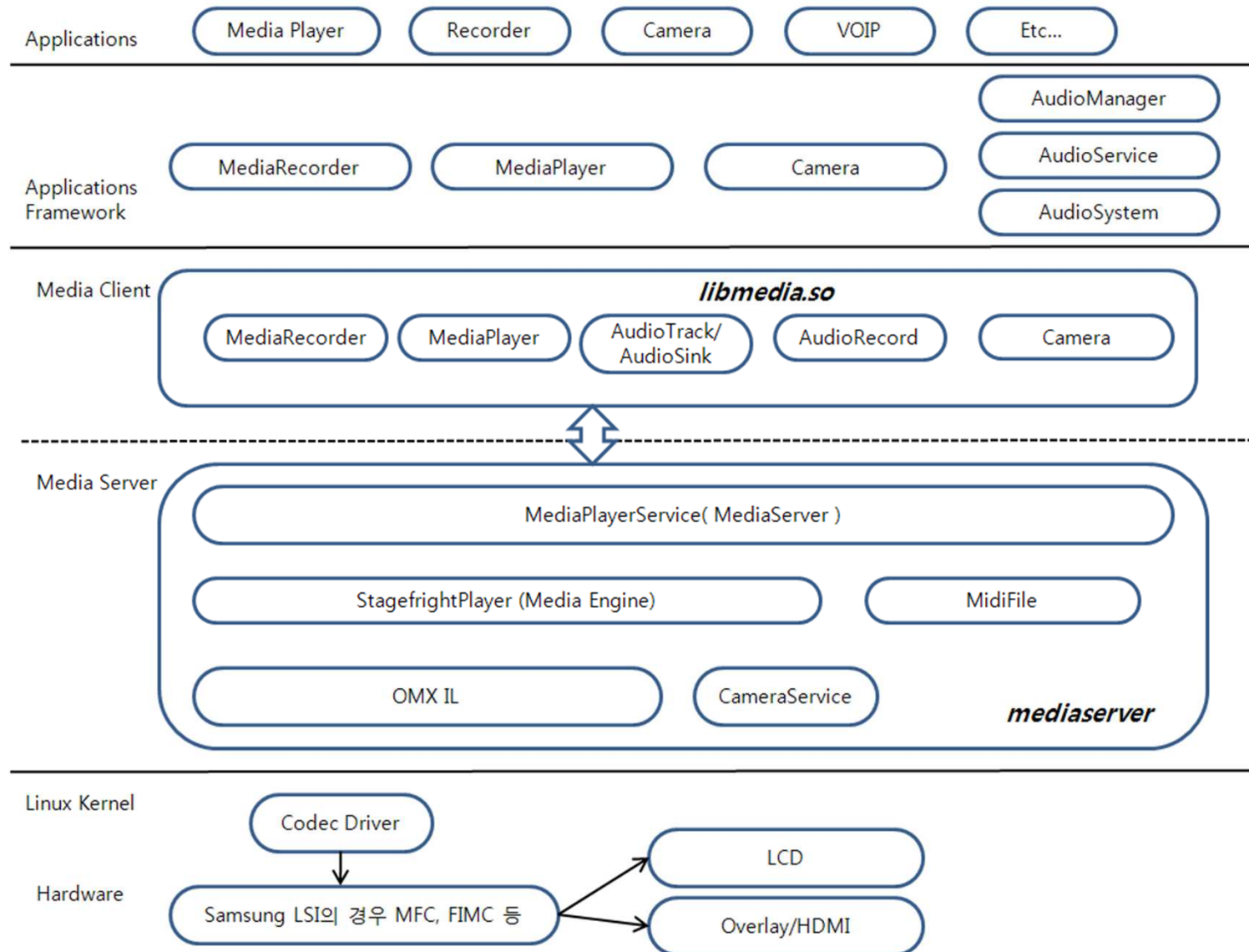
Android Multimedia Framework

- Multimedia framework은 크게 네 개로 나뉘볼 수 있다
 - ✓ Client/Server/Media Engine/Codec – 즉, Binder에 그 기본을 둔다
 - ✓ Libraries level의 media framework은 server만을 얘기한다



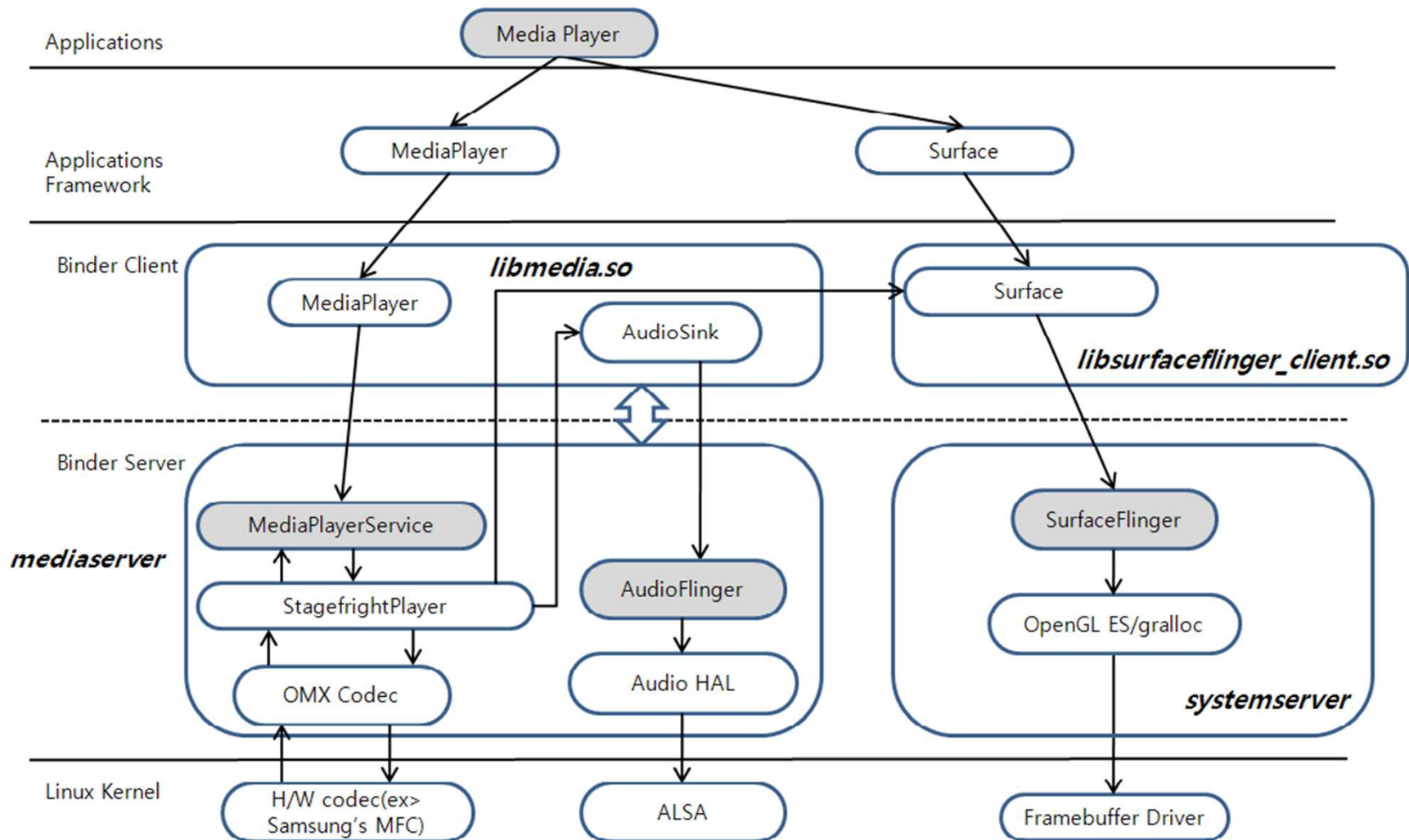
Android multimedia framework의 구조

- Android에서의 multimedia framework 구조는 다음과 같다
 - ✓ Stagefright Player는 다른 엔진으로 대체가능

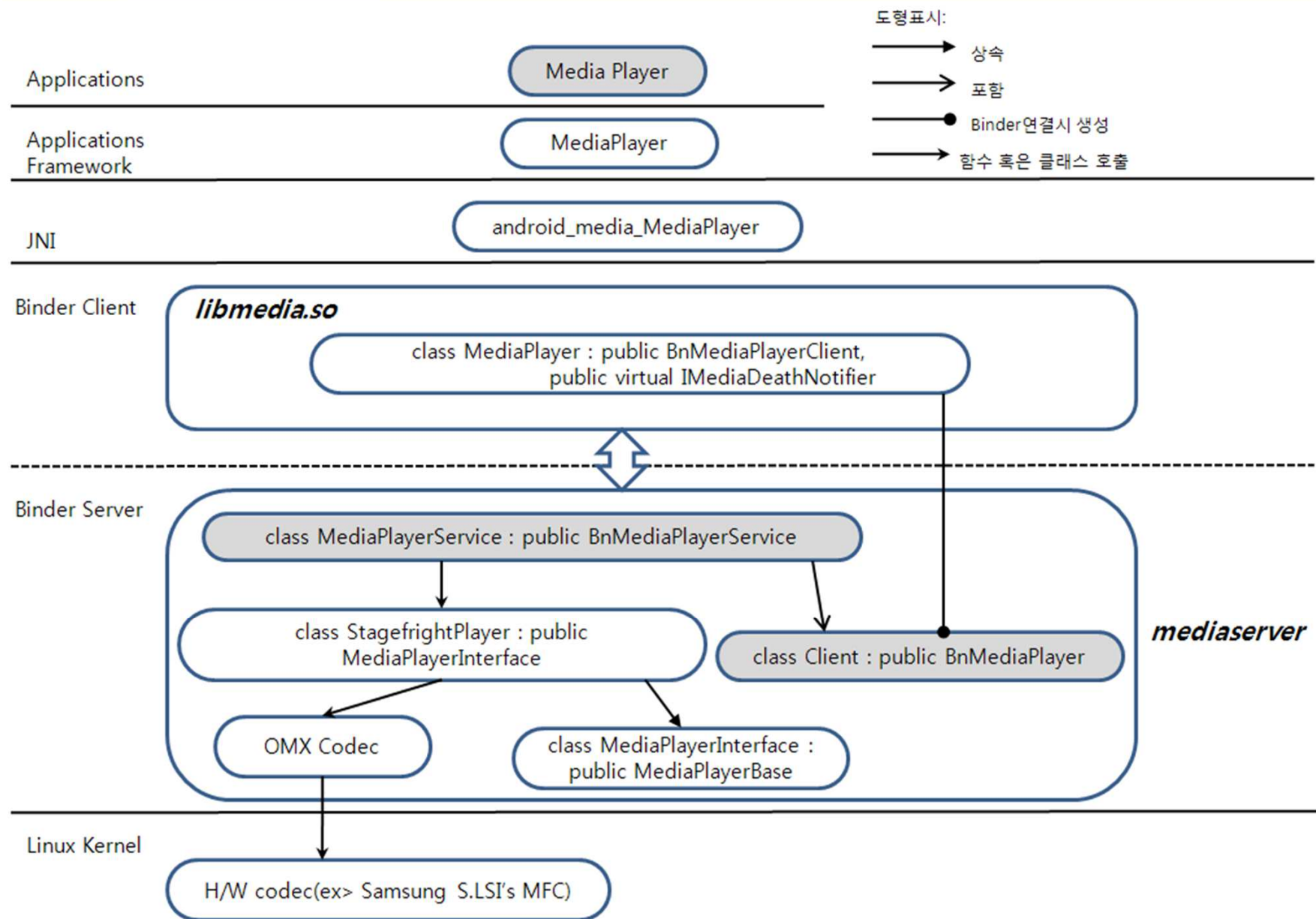


MediaPlayer call flow

- MediaPlayer application이 실행되었을 때의 call flow
 - ✓ 그림에서 StagefrightPlayer인 libstagefrightplayer.so 라이브러리는 다른 player engine으로 대체 가능

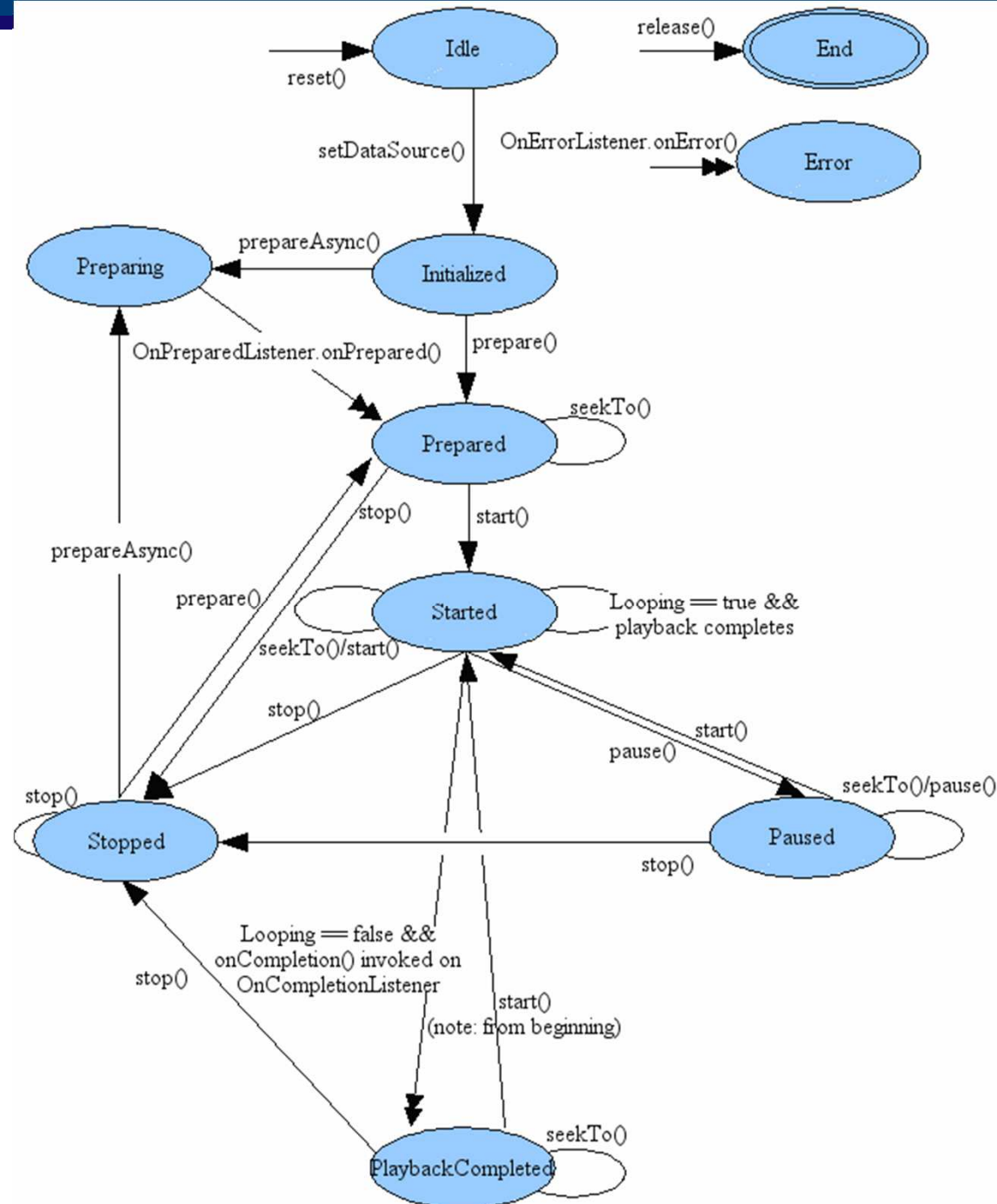


MediaPlayer class structure



Multimedia app. state diagram

- Android Multimedia App.
의 state diagram
 - ✓ 각 단계에서 호출되는 함수
들은 client(app.)에서
server(media engine)으로
호출되는 함수이름들이다

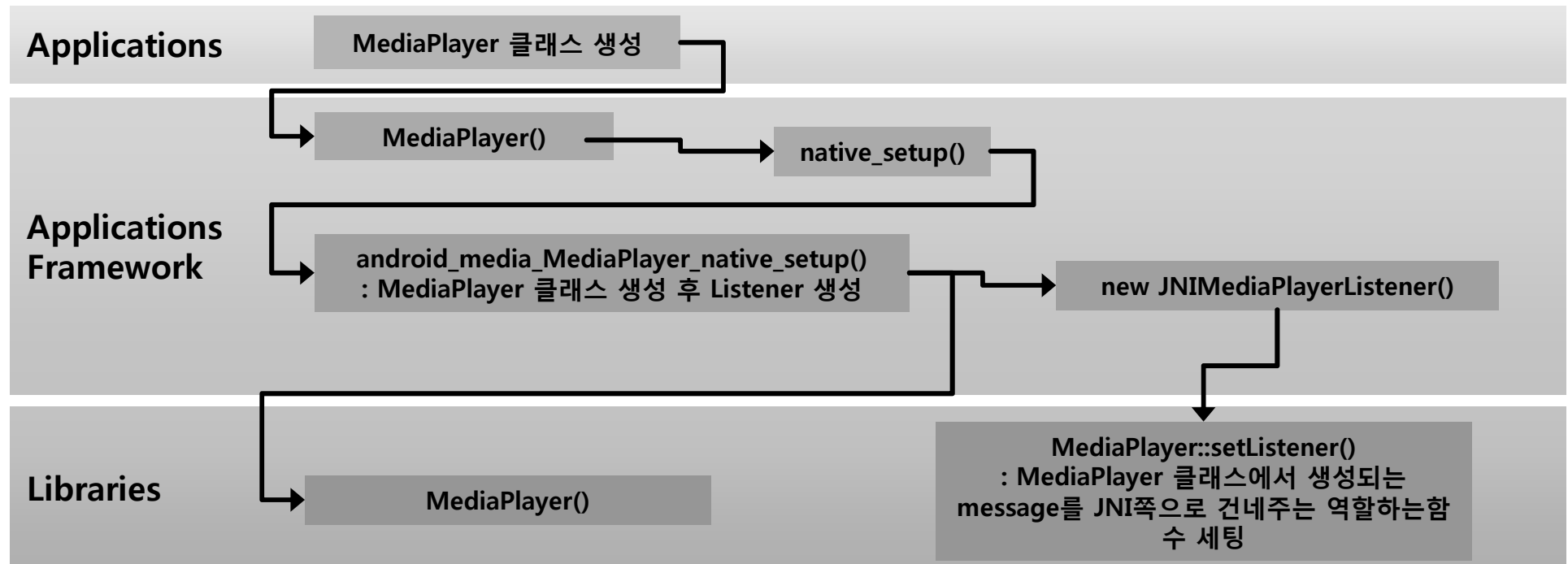


Android multimedia player 구조

- Client
 - ✓ Native console mediaplayer
 - gvideo
- Server
 - ✓ mediaserver

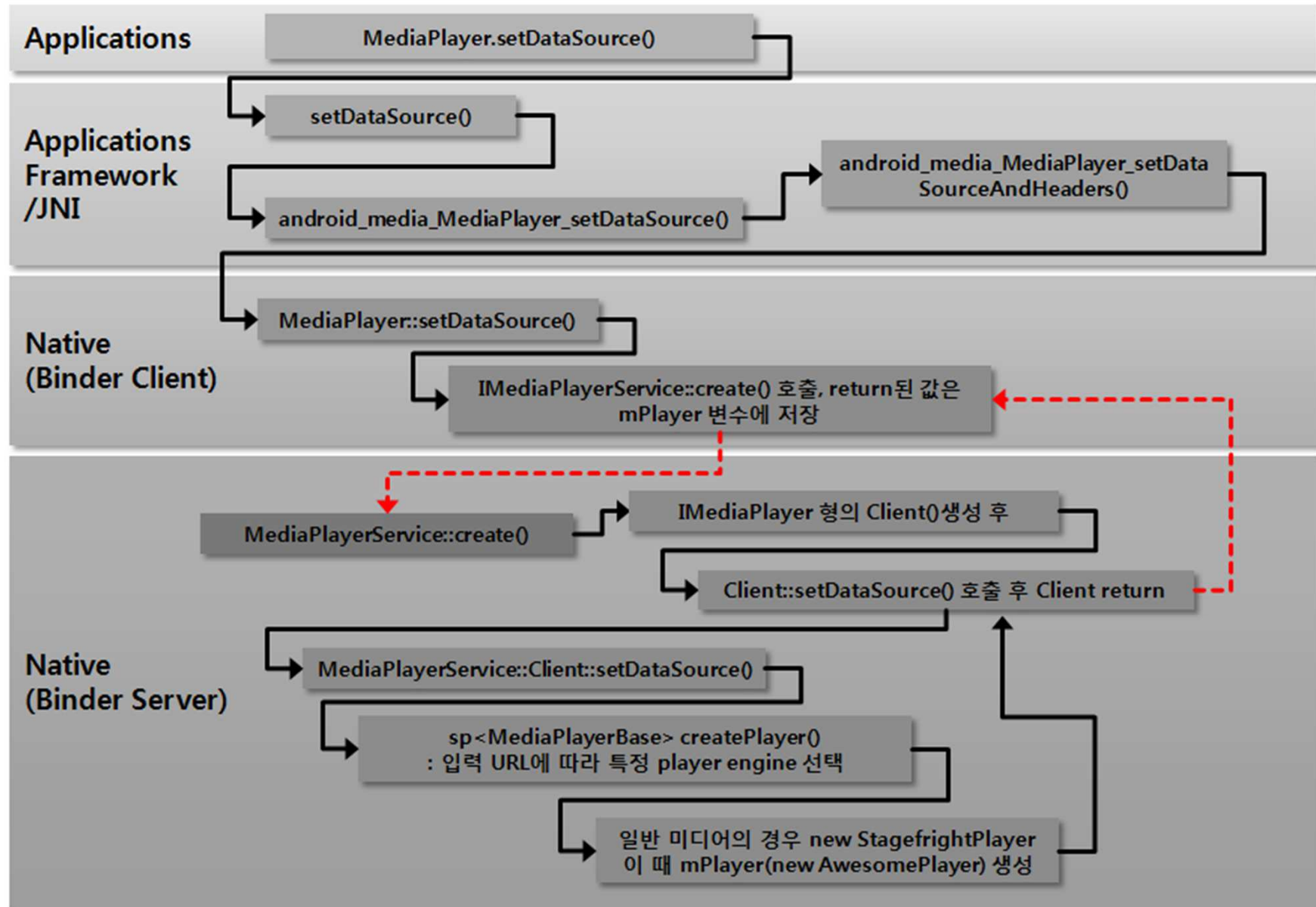
Media Player Call flow(1)

■ MediaPlayer class의 생성과 Listener 등록



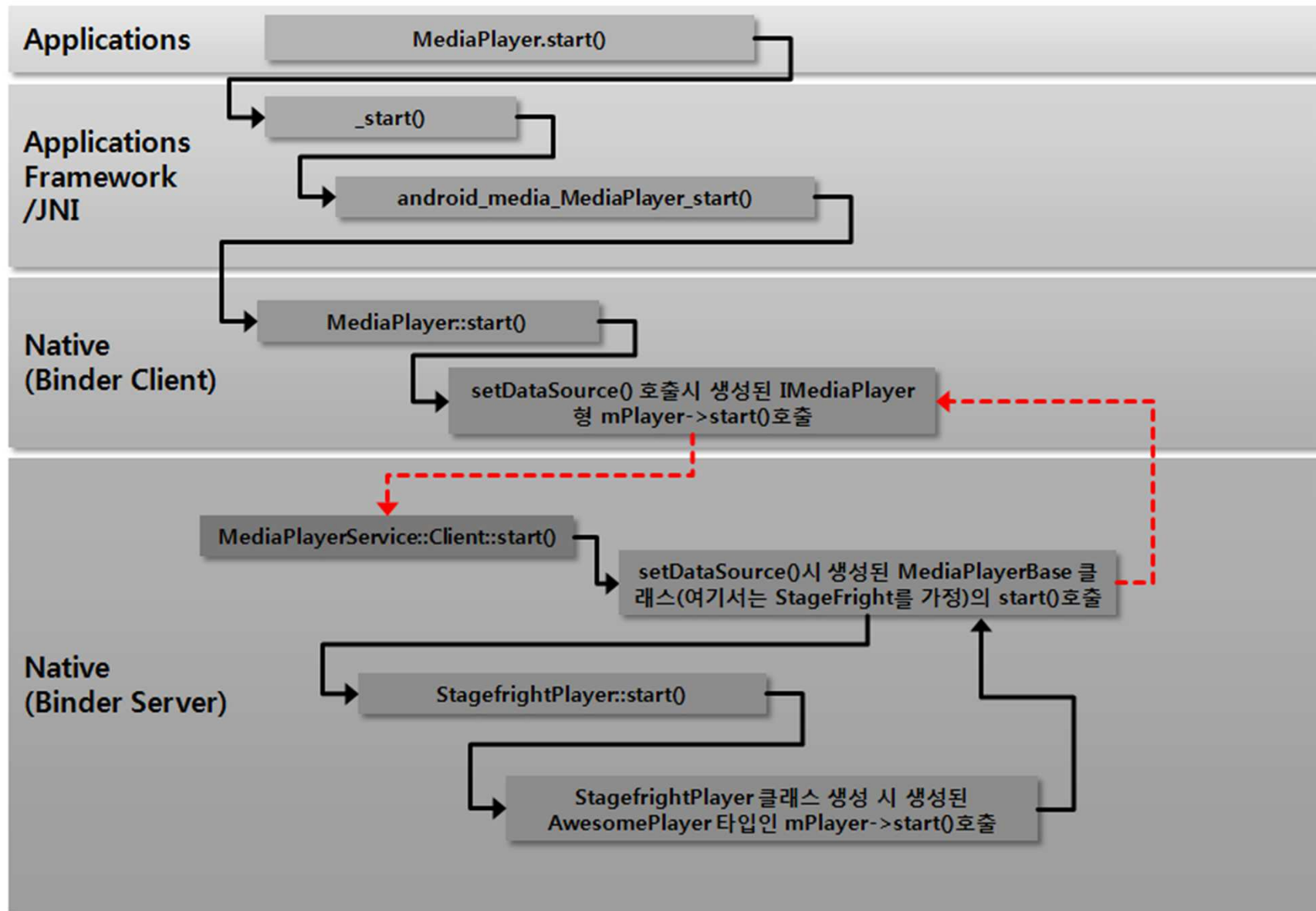
Media Player Call flow(2)

■ Play할 Media의 등록 – setDataSource()의 호출



Media Player Call flow(3)

■ Play의 시작 – start()의 호출



gvideo client

- Android native console multimedia player
 - ✓ Android에서 multimedia를 포팅할 때 사용하는 test application
 - ✓ Application을 일일이 구동하는 것보다 console에서 명령어를 이용하여 media engine을 구동
 - ✓ 개발용 code
 - ✓ 두 가지 버전
 - gvideo: only server와 engine만 run시키는 app
 - <http://freepine.blogspot.com/2009/02/1-native-console-app-for-video-playback.html>
 - 해당 사이트의 소스를 Android 2.x 대에 알맞게 수정
 - gvideo2: gvideo1를 확장해서 server에서 client로 보내는 message 처리 루틴을 추가한 버전

gvideo android makefile

```
LOCAL_PATH:= $(call my-dir)
```

```
include $(CLEAR_VARS)
```

```
LOCAL_SRC_FILES:= gvideo.cpp
```

```
LOCAL_SHARED_LIBRARIES := ₩
```

```
    libcutils ₩
```

```
    libutils ₩
```

```
    libui ₩
```

```
    libsurfaceflinger ₩
```

```
    libaudioflinger ₩
```

```
    libmediaplayerservice ₩
```

```
    libmedia
```

```
LOCAL_MODULE:= gvideo
```

```
LOCAL_C_INCLUDES := ₩
```

```
    frameworks/base/include ₩
```

```
    frameworks/base/media/libmediaplayerservice₩
```

```
    frameworks/base/media/libmedia
```

```
#LOCAL_CFLAGS := ₩
```

```
#          -DHAVE_CONFIG_H
```

```
include $(BUILD_EXECUTABLE)
```

gvideo source 1/2

```
1 #include <media/mediaplayer.h>
2 #include <media/IMediaPlayer.h>
3
4 using namespace android;
5
6 #if 1
7     #define gprintf(fmt, args...) LOGE("%s(%d): " fmt, __FUNCTION__, __LINE__, ##args)
8 #else
9     #define gprintf(fmt, args...)
10 #endif
11
12 int
13 main(int argc, char **argv)
14 {
15     gprintf("entering main...");
16     sp < ProcessState > proc = ProcessState::self();
17     proc->startThreadPool();
18     MediaPlayer mediaPlayer;
19     sp < Surface > gs;
20
21     if (argc > 1)
22     {
23         gprintf("set datasource: %s", argv[1]);
24         mediaplayer.setDataSource(argv[1], NULL);
25     }
26     else
27     {
28         gprintf("set datasource: /aa/test.mp4");
29         mediaPlayer.setDataSource("/aa/test.mp4", NULL);
30     }
31
32     gprintf("create SurfaceComposerClient");
33     int pid = getpid();
34     int nState = 0;
```

gvideo source 2/2

```
36  sp < SurfaceComposerClient > videoClient = new SurfaceComposerClient;
37
38  gprintf("create video surface");
39  sp < SurfaceControl > videoSurface(videoClient->createSurface(pid, 0, 320, 240,
40      PIXEL_FORMAT_OPAQUE,
41      ISurfaceComposer::eFXSurfaceNormal / ISurfaceComposer::ePushBuffers));
42  videoClient->openTransaction();
43
44  // set toppest z-order
45  nState = videoSurface->setLayer(INT_MAX);
46  nState = videoSurface->show();
47  videoClient->closeTransaction();
48
49  gprintf("set video surface to player");
50  gsf = videoSurface->getSurface();
51  mediaplayer.setVideoSurface(gsf); // for android 2.0
52
53  status_t retCode = mediaplayer.prepare();
54
55  if (retCode < 0)
56  {
57      gprintf("prepare failed: %d\n", retCode);
58      IPCThreadState::self()->stopProcess();
59      return -1;
60  };
61
62  mediaplayer.start();
63  for (int i = 0; i < 10; i++)
64  {
65      sleep(1);
66  }
67  mediaplayer.reset();
68
69  // close binder fd, still need waiting for all binder threads exit?
70  IPCThreadState::self()->stopProcess();
71  return 0;
72 }
```

gvideo client

- gvideo client : gvideo2

gvideo client : gvideo.cpp 수행부 해당

- 아래는 gvideo 클라이언트 동작 모습이며, gvideo 클라이언트의 내부를 살펴본다.



```
chlr bgh0@chlr bgh0-desktop: ~  
파일(E) 편집(E) 보기(V) 터미널(T) 도움말(H)  
# /data/local/gvideo /data/local/olipop_1024.MP4  
main: entering main...msg data Q initialize  
main: set datasource: /data/local/olipop_1024.MP4  
main: create SurfaceComposerClient  
main: create video surface  
main: video surface->setLayer, 0  
main: video surface->show, 0  
main: set video surface to player  
notify: event: MEDIA_INFO  
main: media player start.....
```


gvideo client

- gvideo client

setDataSource

- mediaplayer의 client와 server구동의 시작점

```
ret = mediaplayer.setDataSource(argv[1])
```

```
int main(int argc, char** argv)
{
    ... ..
    dprintf("set datasource: %s\n", argv[1]);

    ret = mediaplayer.setDataSource(argv[1]);

    if( ret != NO_ERROR)

    .....
}
```

gvideo client

- gvideo client

mediaplayer.prepare

- mediaplayer관련 함수의 두번째 함수 call

```
status_t retCode = mediaplayer.prepare()
```

```
int main(int argc, char** argv)
{
    ... ..
    status_t retCode = mediaplayer.prepare();

    if(retCode < 0)
    {
        dprintf("prepare failed: %d\\n", retCode);
    }
    ... ..
}
```

gvideo client

- gvideo client

mediaplayer.start

- mediaplayer에 start명령을 내린다. 세번째 함수 call이다.

mediaplayer.start()

```
int main(int argc, char** argv)
{
    ... ..
    mediaplayer.start();

    dprintf("media player start.....\n");
    //mediaplayer.reset();
    ... ..
}
```

gvideo client

- gvideo client

미디어 플레이어 구동 순서

```
mediaplayer.setDataSource();  
mediaplayer.prepare();  
mediaplayer.start()
```

이후로 mediaplayer service가 동작,
client는 관련 message만 처리하면 되는 구조 중지하고 싶을 때는

```
mediaplayer.reset()
```

혹은

```
mediaplayer.stop();  
mediaplayer.disconnect();
```

Android multimedia server

- mediaserver
 - ✓ mediaserver에서 실행시키는 MediaPlayerService의 동작분석
- mediaserver에서 media engine을 호출하는 루틴을 중점파악

Media server/engine관련 source

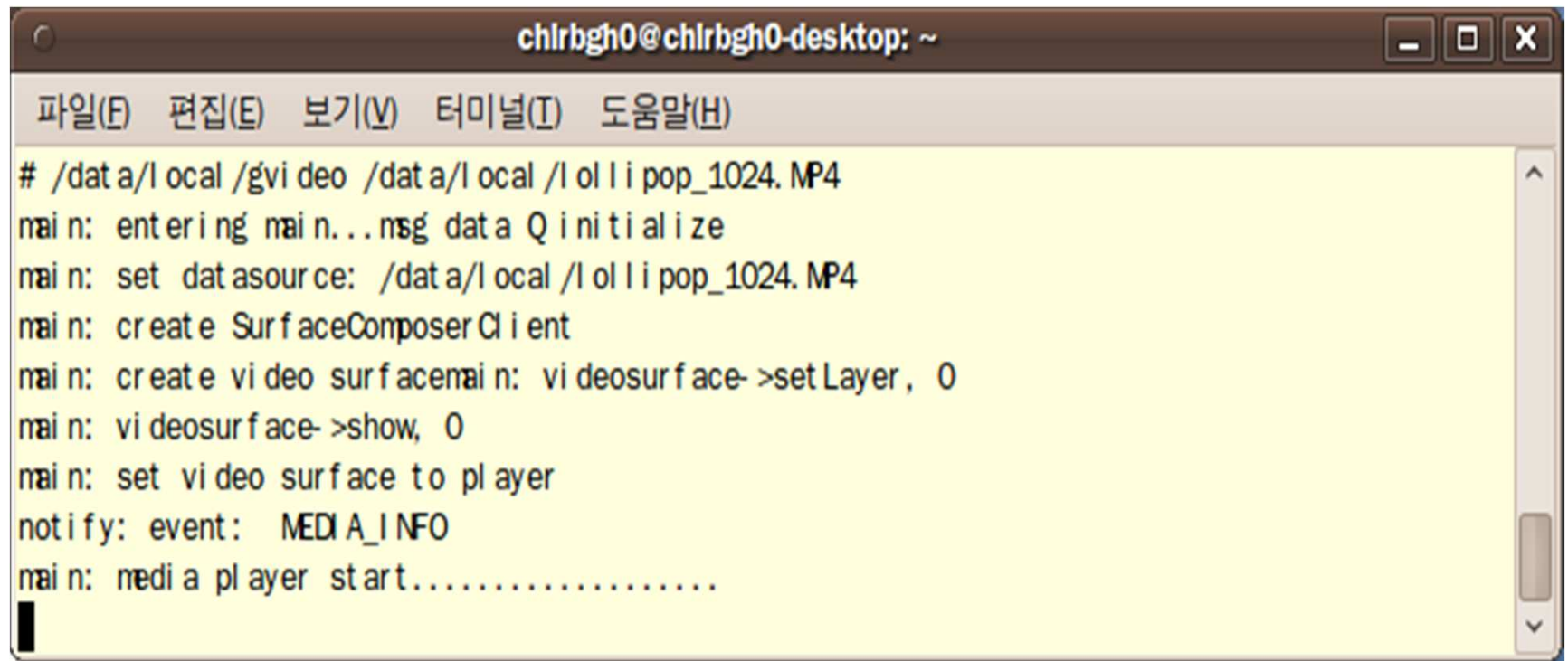
- Header
 - ✓ frameworks/base/include/media/*
- Client
 - ✓ frameworks/base/media/libmedia/*
- Server
 - ✓ frameworks/base/media/mediaserver/*
 - ✓ frameworks/base/media/libmediaplayerservice/*
- Engine
 - ✓ frameworks/base/media/libstagefright/*
- Codec
 - ✓ frameworks/base/media/libstagefright/omx : Interface
 - ✓ frameworks/base/media/libstagefright/codecs : codec source
- Renderer
 - ✓ frameworks/base/media/libstagefright/colorconversion: video renderer

mediaserver

■ mediaserver

mediaserver : MediaPlayerService 에 해당

- 아래는 앞서 살펴본 gvideo 클라이언트 동작 모습이며, gvideo 클라이언트 수행에 따른 MediaPlayerService 동작에 대해 살펴본다.



The screenshot shows a terminal window titled "chlrhgh0@chlrhgh0-desktop: ~". The terminal output displays the following log messages:

```
# /data/local/gvideo /data/local/olipop_1024.MP4
main: entering main...msg data Q initialize
main: set datasource: /data/local/olipop_1024.MP4
main: create SurfaceComposerClient
main: create video surfacemain: videosurface->setLayer, 0
main: videosurface->show, 0
main: set video surface to player
notify: event: MEDIA_INFO
main: media player start.....
```

[그림 2 - gvideo client 동작]

mediaserver

- mediaserver

- frameworks/base/media/libmediaplayerservice/MediaPlayerService.cpp

```
sp<IMediaPlayer> MediaPlayerService::create
```

- Client 담당 Binder생성 → setDataSource()가 호출되며 Android의 Player들 중 하나가 생성이 된다.(eg. stagefright)

```
sp<IMediaPlayer> MediaPlayerService::create
```

```
sp<IMediaPlayer> MediaPlayerService::create(pid_t pid, const sp<IMediaPlayerClient> & client, const char* url)
{
    sp<Client> c = new Client(this, pid, connId, client);
    if (NO_ERROR != c->setDataSource(url))
    {
        c.clear();
        return c;
    }
    wp<Client> w = c;
    Mutex::Autolock lock(mLock);
    mClients.add(w);
    return c;
}
```


mediaserver

■ mediaserver

setDataSource

- DataSource 설정 및 플레이어 객체 생성, 플레이어 객체에 DataSource 를 할당하여 호출

setDataSource

```
status_t MediaPlayerService::Client::setDataSource(const char *url)
{
    ... ..
    //File의 확장자로 Playertype을 얻는다. 만족 하는 확장자가 없으면 static play
er_type getDefaultPlayerType() 함수를 통해 PV_PLAYER나 STAGEFRIGHT_PLA
YER를 얻는다.
    player_type playerType = getPlayerType(url);

    // PlayerType으로 해당하는 Player 객체 생성
    sp<MediaPlayerBase> p = createPlayer(playerType);
```

mediaserver

■ mediaserver

setDataSource

- DataSource 설정 및 플레이어 객체 생성, 플레이어 객체에 DataSource 를 할당하여 호출

setDataSource

```
status_t MediaPlayerService::Client::setDataSource(const char *url)
{
    ... ..
    // player 객체의 setDataSource를 url을 이용해서 호출, 여기서는 stagefr
    ight관련 player임

    mStatus = p->setDataSource(url);

    ... ..
}
```

mediaserver

■ mediaserver

creatPlayer

- setDataSource 로 얻어진 플레이어 타입으로 플레이어 생성

createPlayer

```
sp<MediaPlayerBase> MediaPlayerService::Client::createPlayer(player_type playerType)
{
    ... ..
    sp<MediaPlayerBase> p = mPlayer;
    if (p == NULL) {
        p = android::createPlayer(playerType, this, notify);
    }
    return p;
}
```

mediaserver

■ mediaserver

```
sp<MediaPlayerBase> creatPlayer
```

- setDataSource 로 얻어진 플레이어 타입으로 플레이어 생성

sp<MediaPlayerBase> createPlayer

```
static sp<MediaPlayerBase> createPlayer(player_type playerType, void* cookie, notify_callback_f notifyFunc) {  
    sp<MediaPlayerBase> p;  
    switch (playerType) {  
        ... ..  
        #if BUILD_WITH_FULL_STAGEFRIGHT  
        case STAGEFRIGHT_PLAYER:  
            LOGV(" create StagefrightPlayer");  
            p = new StagefrightPlayer;  
            break;  
        ... ..  
    }  
    return p;  
}
```

mediaserver

- mediaserver

StagefrightPlayer::setDataSource

-StagefrightPlayer 타입으로 생성한 플레이어 객체의 setDataSource 수행
플레이어 initCheck() 및 AudioSink 설정 등

StagefrightPlayer::setDataSource

frameworks/base/media/libmediaplayerservice/StagefrightPlayer.h
frameworks/base/media/libmediaplayerservice/StagefrightPlayer.cpp

파일을 참조

Android multimedia engine

- Media engine
 - ✓ mediaserver로 부터 호출되는 media engine routine 파악
- Android 2.2
 - ✓ OpenCORE와 Stagefright engine이 혼재
- Android 2.3
 - ✓ *Stagefright로 변경 → 기존 OpenCORE 개발사들이 힘들어짐*

OpenCORE

- PacketVideo사의 Multimedia Engine

OpenCORE 개요(1)

- Android 2.2 version까지의 표준 Multimedia Engine
- external/opencore/*
- OpenCORE는 Google Android의 Multimedia Framework로서 PacketVideo라고도 불린다.
- OpenCORE Multimedia Framework는 PacketVideo를 포함한 Software Layer의 이름이기도 하다.
- OpenCORE Multimedia Framework 코드는 매우 크고, C++로 작성된 Full-Featured 운영체제에 통합되는 구조로 되어 있다.
- OpenCORE Multimedia Framework를 거시적인 관점에서 볼 때, 그것은 주로 다음과 같은 두 가지 측면을 포함하고 있다.
 - ✓ PVPlayer
 - 다양한 오디오 비디오 스트림에 대한 재생 기능을 갖고 있는 미디어 플레이어 위한 함수들을 제공
 - ✓ PVAuthor
 - 오디오, 비디오 스트림을 녹화, 이미지 캡처 기능을 위한 함수들을 제공
- PVPlayer 및 PVAuthor는 개발자들이 사용할 수 있는 형태로 SDK를 제공한다.

OpenCORE 개요 (2)

✓ OSCL (Operating System Compatibility Library)

➤ 운영 체제 호환성 라이브러리.

다른 운영체제간의 호환성을 위하여 기본 운영체제 동작을 지원하는 기능을 포함하고 있다.

기본 데이터 형식, Configuration, String Instruments, I/O, Error handling, Thread 등을 포함한 C++ 기본 라이브러리와 유사하다.

✓ PVMF (PacketVideo Multimedia Framework)

➤ Document Analysis(Parser)와 Composition(Composer)를 구현한 Framework.

이 안의 Codec Node는 공통적인 인터페이스를 상속할 수 있다.

사용자 계층은 Node를 생성하기 위하여 그 공통 인터페이스를 상속 할 수 있다.

✓ PVPlayer 엔진 및 PVAuthor 엔진

OpenCORE 개요 (3)

- Player의 입장에서 PVPlayer는
입력(Source)으로 Network File 혹은 Media Stream 등이 될 수 있고,
출력(Sink)은 오디오/비디오 장비의 입력이 될 수 있고,
기본적인 기능을 포함하는 미디어 흐름 제어와 Document Analysis, Video Streaming, Audio Decoder(Decode)와 그 외의 다른 특징을 갖고 있다.
Paper Document서부터 방송 미디어까지 포함하고 있고, 또한 Network-Related RTSP Streaming 기능도 포함하고 있다.
- 미디어 영역의 recording에 있어서,
PVAuthor 입력 (원본)은 카메라, 마이크 및 기타 장비,
출력(Sink) 각종 문서의 동기화의 흐름, 비디오(Encode)로 작성된 문서 등 오디오 인코딩 스트리밍 등 기능을 수행한다.
- OpenCORE SDK의 사용에 있어,
응용프로그램 계층에서 Adaptor(Adaptor)를 구현하는 것이 필요하고,
그 Adaptor는 PVMF를 위한 Node의 특정기능을 Common Interface를 이용해서 구현해야 하는데 이것은 상위 단에서의 사용을 위하여 Plug-in 형태로 구현하게 된다.

Stagefright

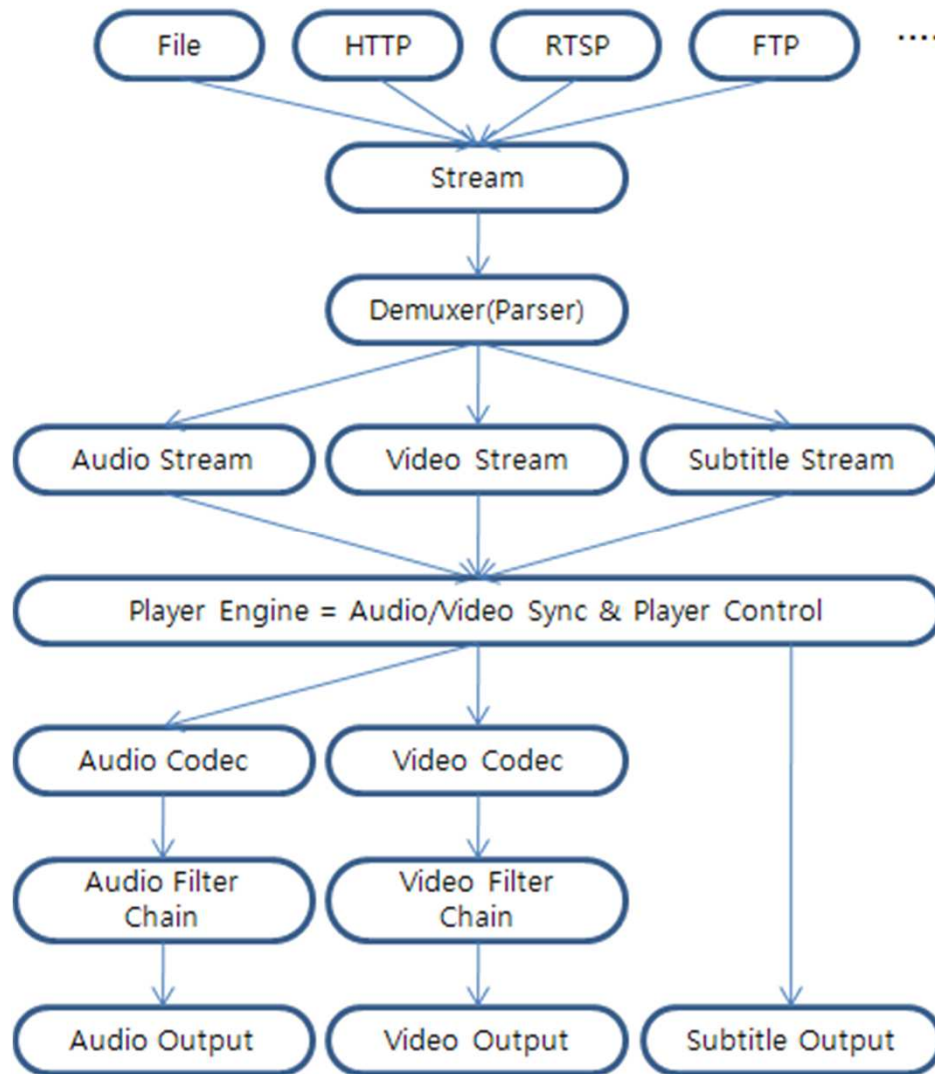
- Android 2.3부터의 표준 Multimedia Engine

StageFright

- Android 2.0서부터 새로 나타난 Multimedia Engine
 - ✓ 매우 단순하고 OpenCORE solution에 비해서 직관적인 구조를 가지고 있음.
 - ✓ Android Gingerbread서부터 공식으로 채택
 - ✓ Engine 대부분을 새롭게 구성하였고 적은 양의 코드구조를 갖는다
 - OpenCORE에 비해서 상대적으로 쉬운 구조이나, Parser의 경우 유연한 구조를 갖지는 않음.

일반 Multimedia Engine의 구조

- Linux Multimedia Engine인 mplayer의 경우 – Structure 구성



동영상에서 사용하는 용어정리

■ Mplayer에서의 동영상 처리 module 용어 정리

✓ Stream

- 동영상이 저장되어 있는 파일, 네트워크 등
- Stagefright에서는 DataSource라고 불림

✓ Demuxer

- Parser라고 불린다.
- Audio, Video, Subtitle의 세 가지를 Stream에서 분리하는 역할을 한다.
- StageFright에서는 MediaExtractor 클래스이며, 일반적으로는 분리되는 데이터 세가지를 Demuxer stream이라고 얘기하며
- StageFright에서는 MediaSource라고 불리며, AwesomePlayer에서는 mVideoTrack, mAudioTrack이라는 두 가지로 표현이 된다.

✓ Audio Codec

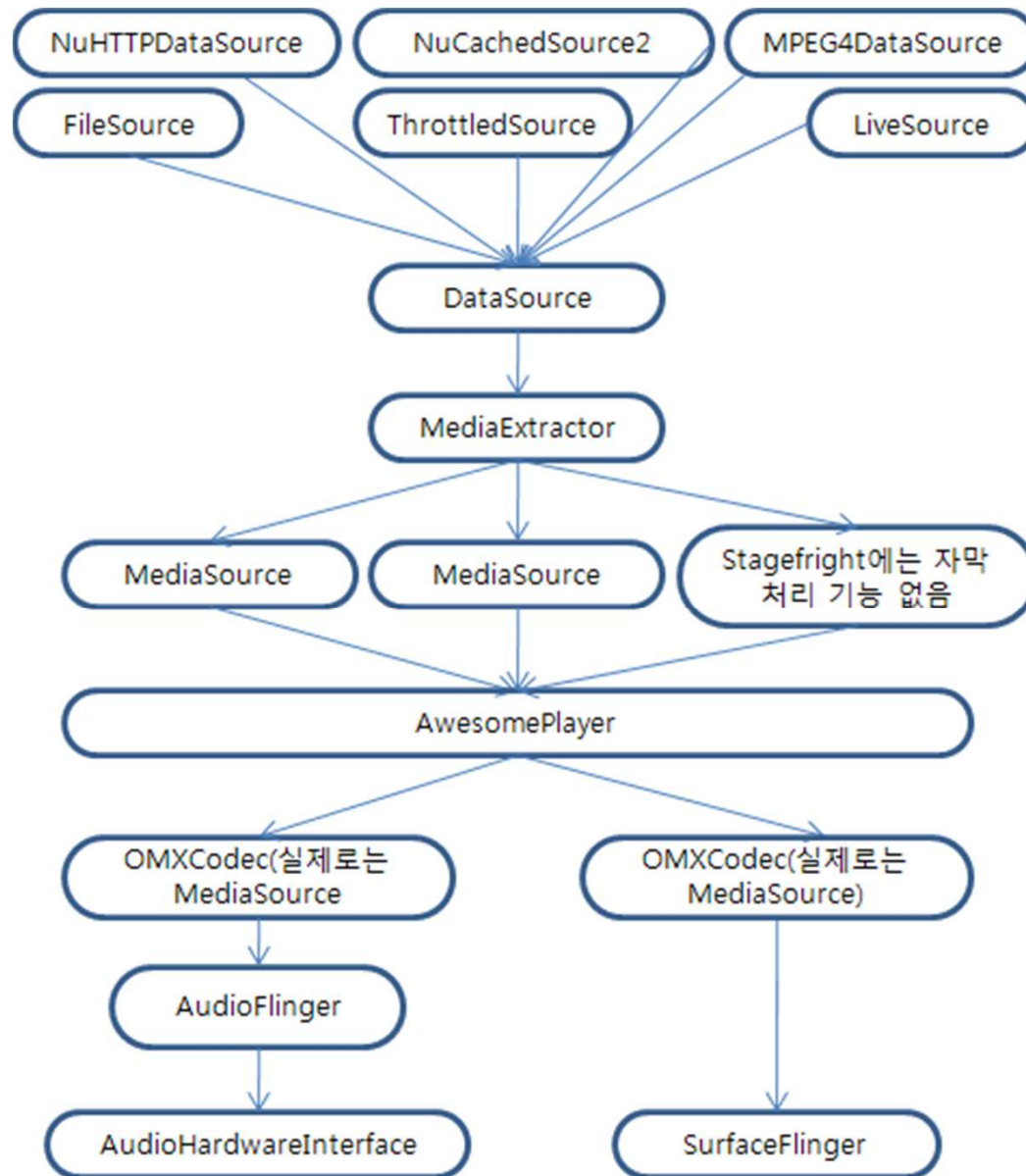
- Encoded Audio Data를 디코딩한다
- StageFright에서는 MediaSource라고 불리며, 실제로는 OMXCodec(MediaSource에서 상속됨)이다

동영상에서 사용하는 용어정리

- Mplayer에서의 동영상 처리 module 용어 정리(계속)
 - ✓ Audio Filter
 - 오디오 출력데이터에 어떤 조작을 가하기 위해서 사용한다
 - StageFright에서는 AudioFlinger에서 처리된다
 - ✓ Video Codec
 - Audio Codec과 비슷한 역할이다.
 - ✓ Video Filter
 - Audio Filter와 유사한 역할이나, StageFright에서는 사용되지 않는다.

Stagefright Player의 기본 구조 및 class

- 앞의 mplayer의 구조와 비교했을 때의 Class 구성도



Stagefright Player에서의 Class 정리

■ Stagefright에서 사용되는 Class들의 정의는 다음과 같다.

✓ DataSource

- data input type에 대한 class - ex> file, http
- 각 MediaExtractor(ex> MP3Extractor)에서는 기본적으로 Sniff함수를 DataSource에 등록시켜야 한다.
- MediaExtractor의 Create함수가 AwesomePlayer()에서 호출될 때 각 Sniff함수가 호출된다. 해서 어떤 media가 입력되었는지 선택할 수 있도록 한다.
- ==> 즉, demuxer를 선택할 수 있도록 한다.

✓ MediaSource

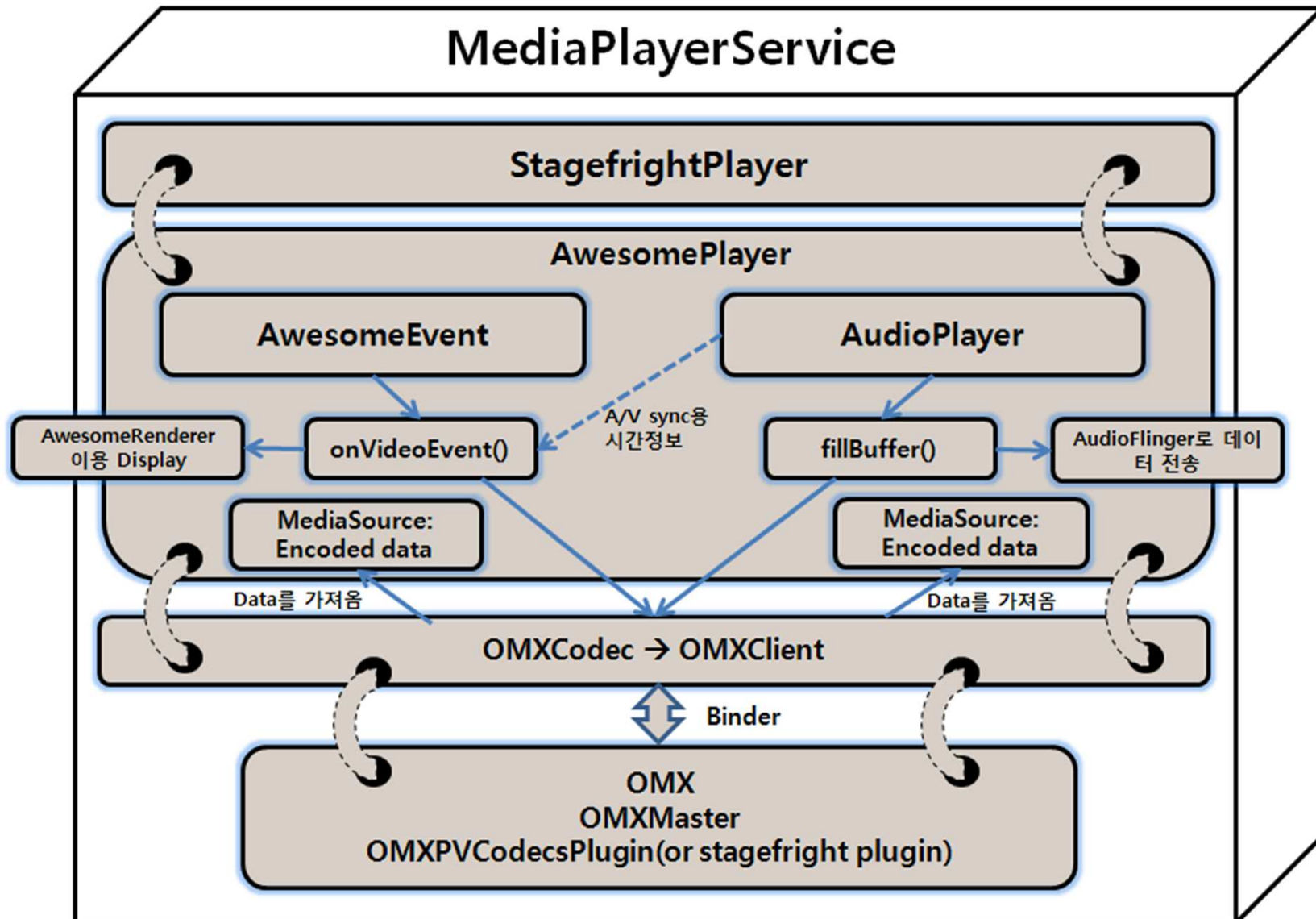
- 데이터를 입력으로 제공한다는 의미를 갖는 클래스이다. 즉, 데이터의 소스가 될 수 있다는 의미이다.
- 데이터의 입력은 세 가지로 볼 수 있는데
 - Encoded Data
 - Demuxer(Extractor)에서 분리된 데이터를 제공한다. MediaSource중 Extractor에서 분리된 mAudioTrack, mVideoTrack을 애기한다
 - Encoded Data된 데이터를 Decoding한 데이터
 - OMXCodec을 애기한다
 - Raw Data
 - CameraSource와 같은 데이터를 제공해주는 소스를 애기한다.

Stagefright Player에서의 Class 정리(계속)

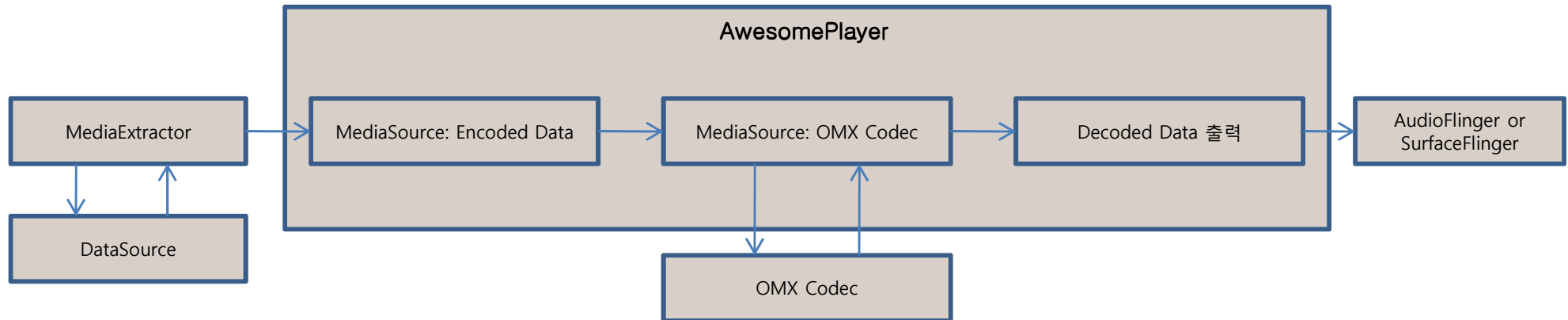
- Stagefright에서 사용되는 Class들의 정의는 다음과 같다.
 - ✓ MediaExtractor
 - Demuxer를 애기함
 - ✓ MediaBuffer
 - MediaSource서부터 다른 루틴으로 건네지는 데이터 클래스이다.
 - ✓ MediaBufferGroup
 - MediaBuffer를 링크드 리스트로 다루는 헤더 객체라고 보면 된다.

Stagefright의 기본구조

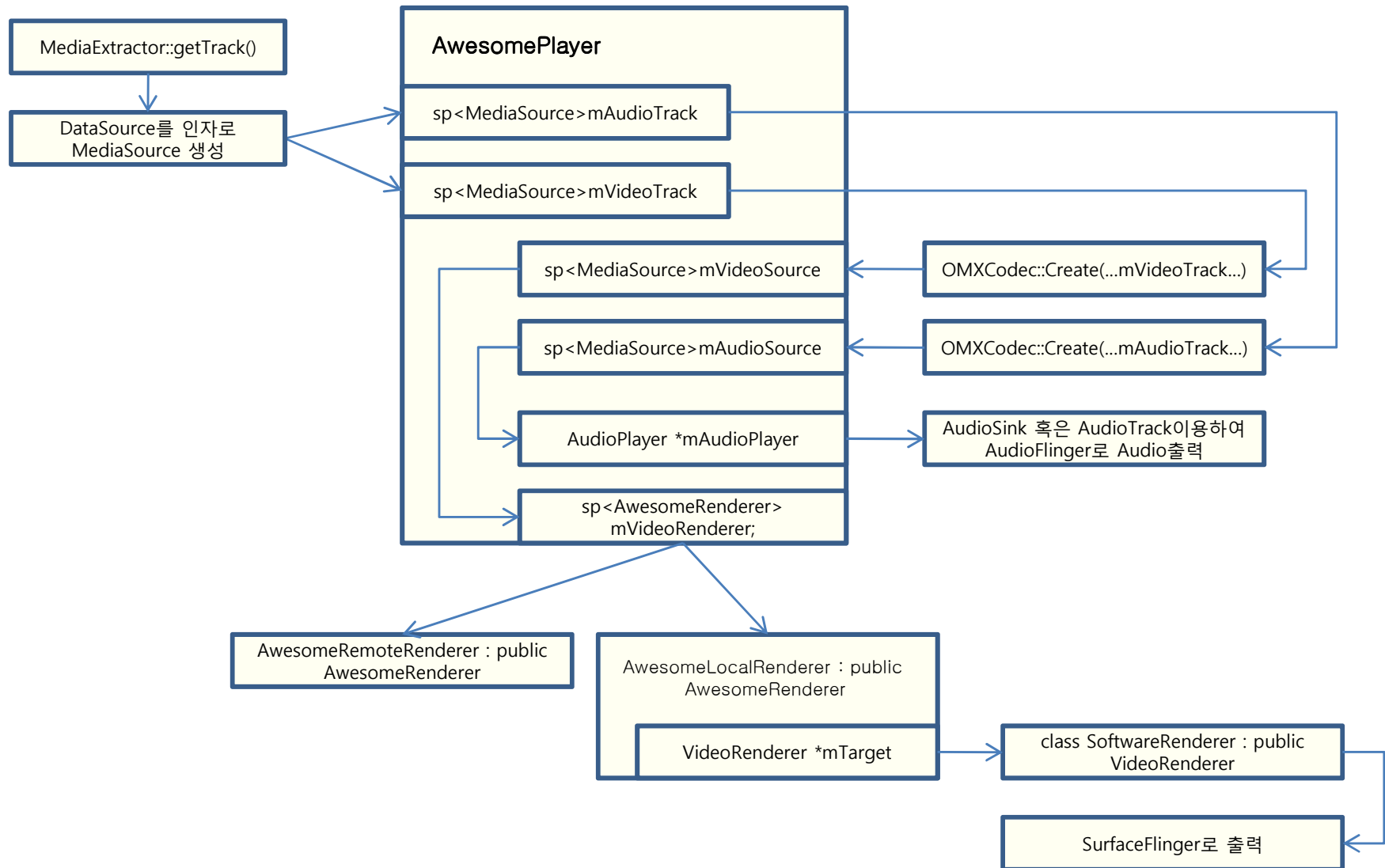
MediaPlayerService 소속인 StagefrightPlayer의 기본구조



Stagefright Class 구조도 #1

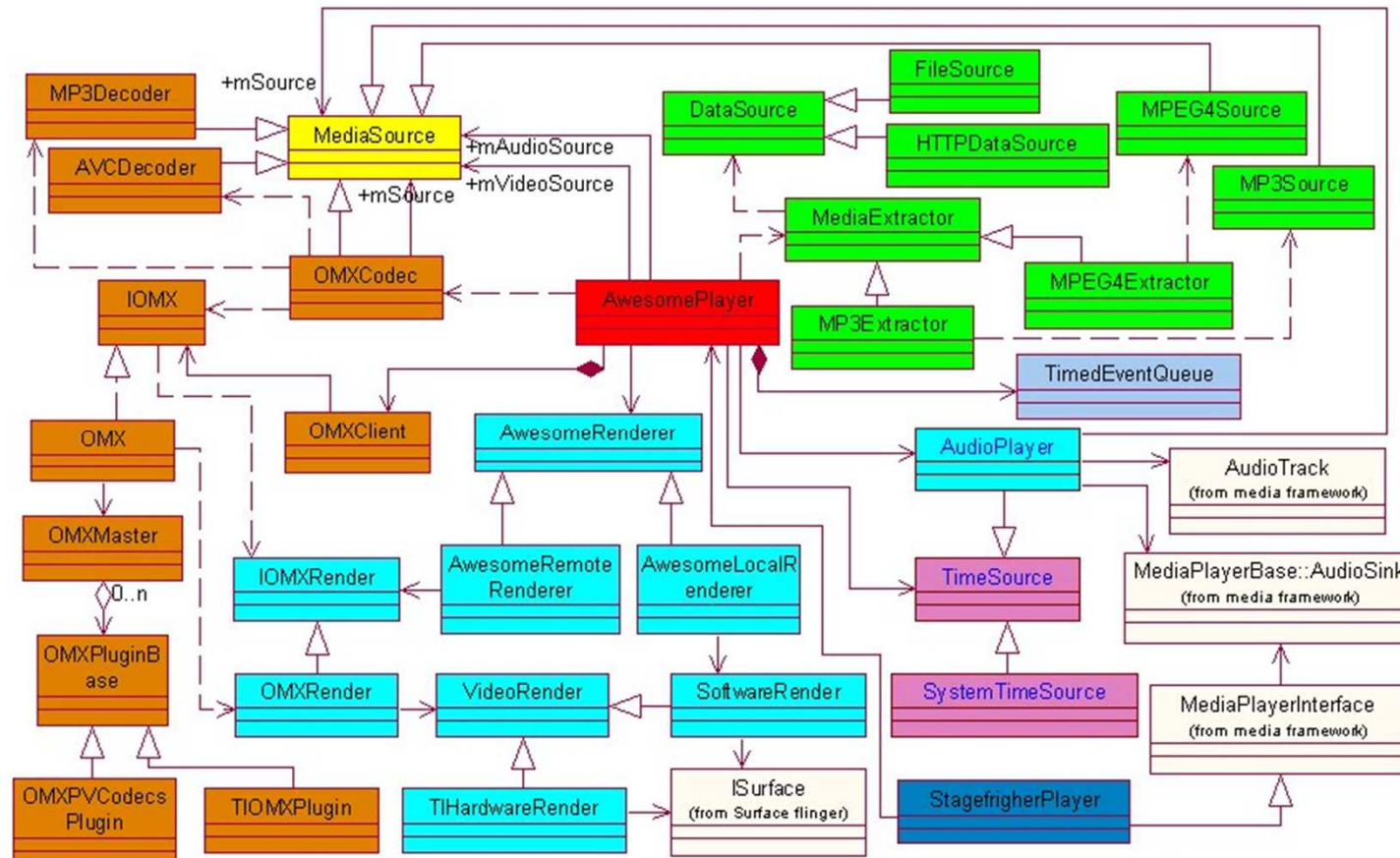


Stagefright Class 구조도 #2



Stagefright Class 구조도 #3

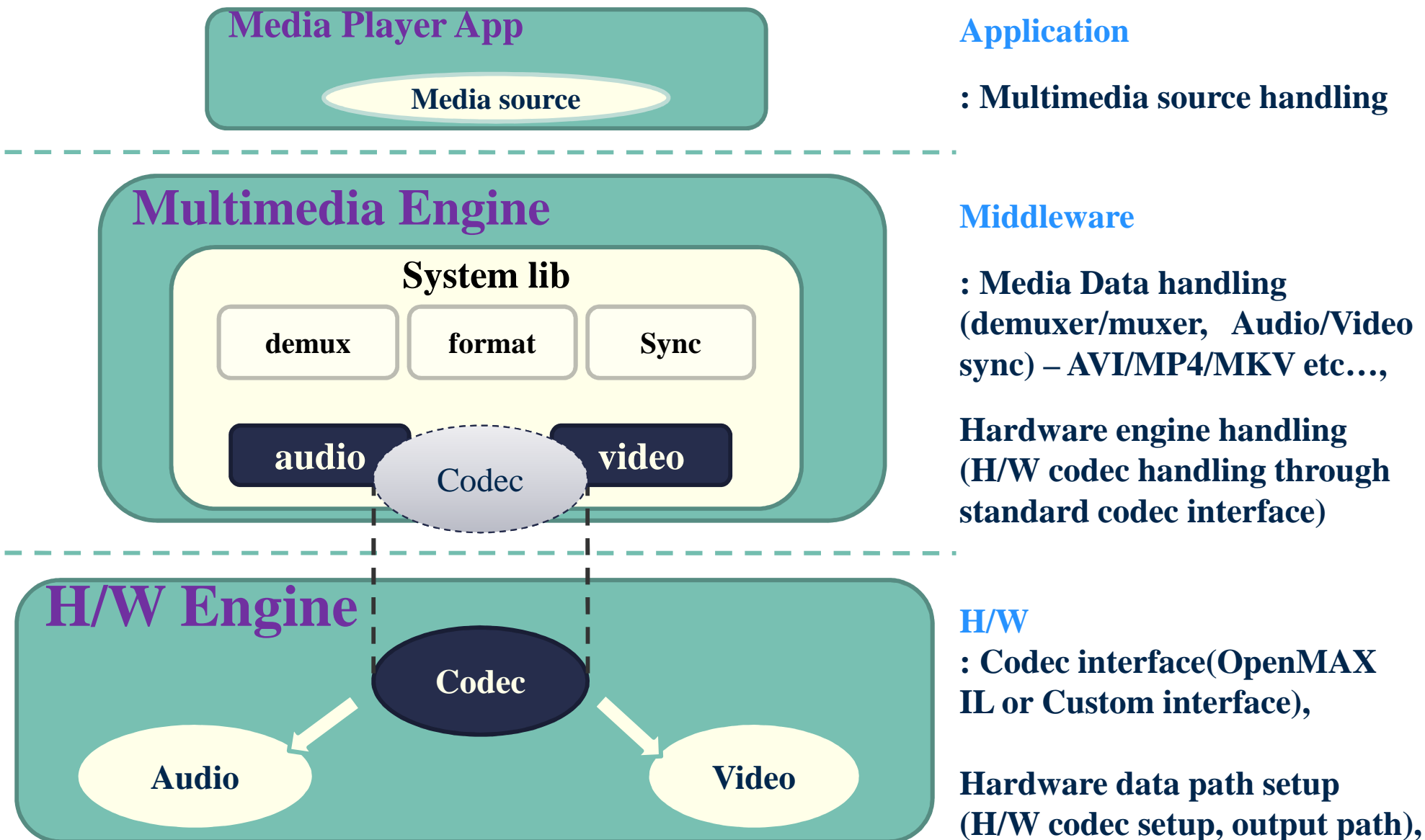
- <http://freepine.blogspot.com/2010/01/overview-of-stagefrighter-player.html> 사이트의 그림에서 발췌
 - ✓ 참고용 - 클래스의 구조는 이와 같이 복잡하나, 전체 media player의 plugin의 관점에서 접근해야만 구조 파악이 쉽다.



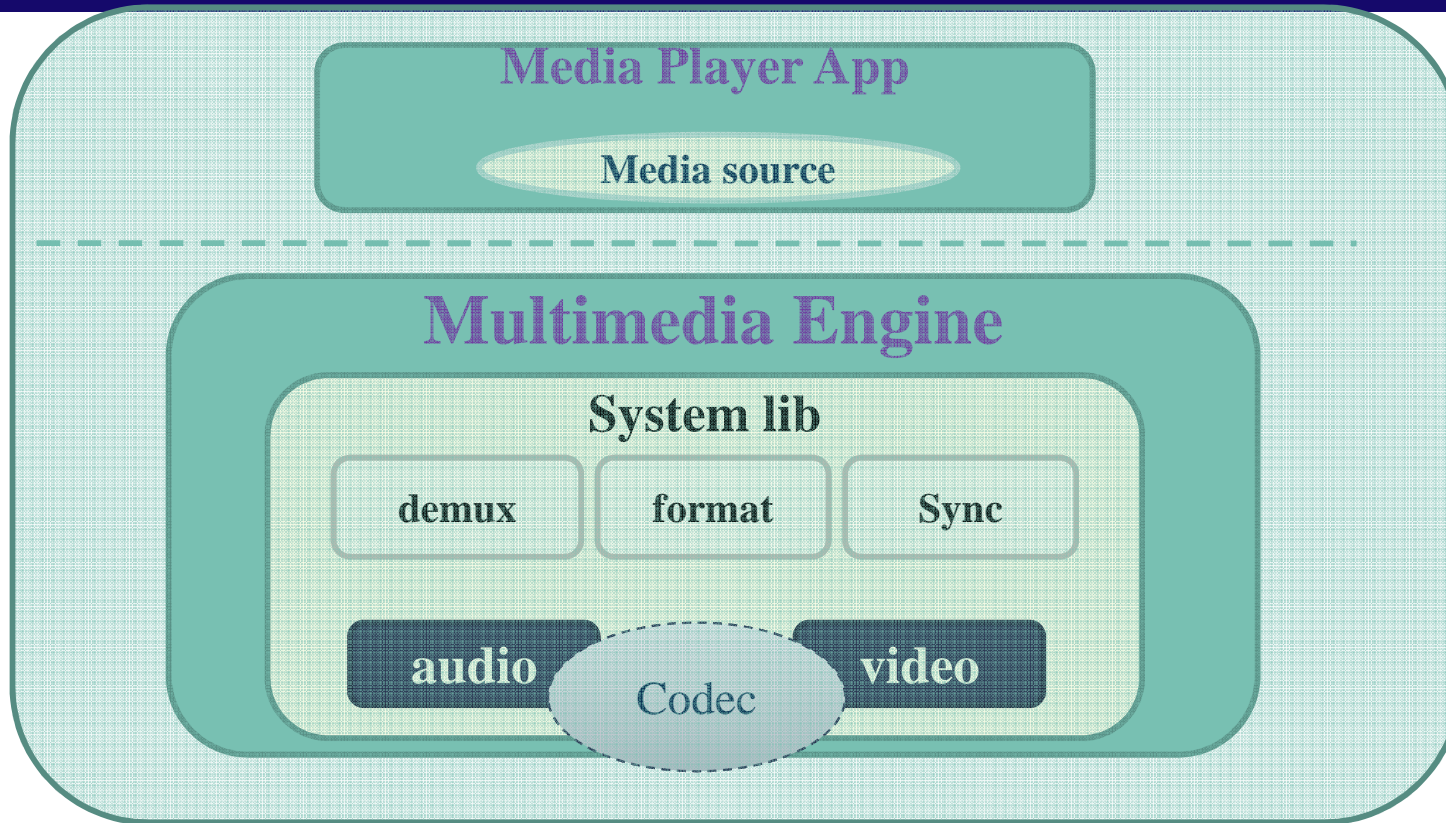
Android codec interface

- Android engine에서의 codec interface

Common Multimedia Engine(include Android)

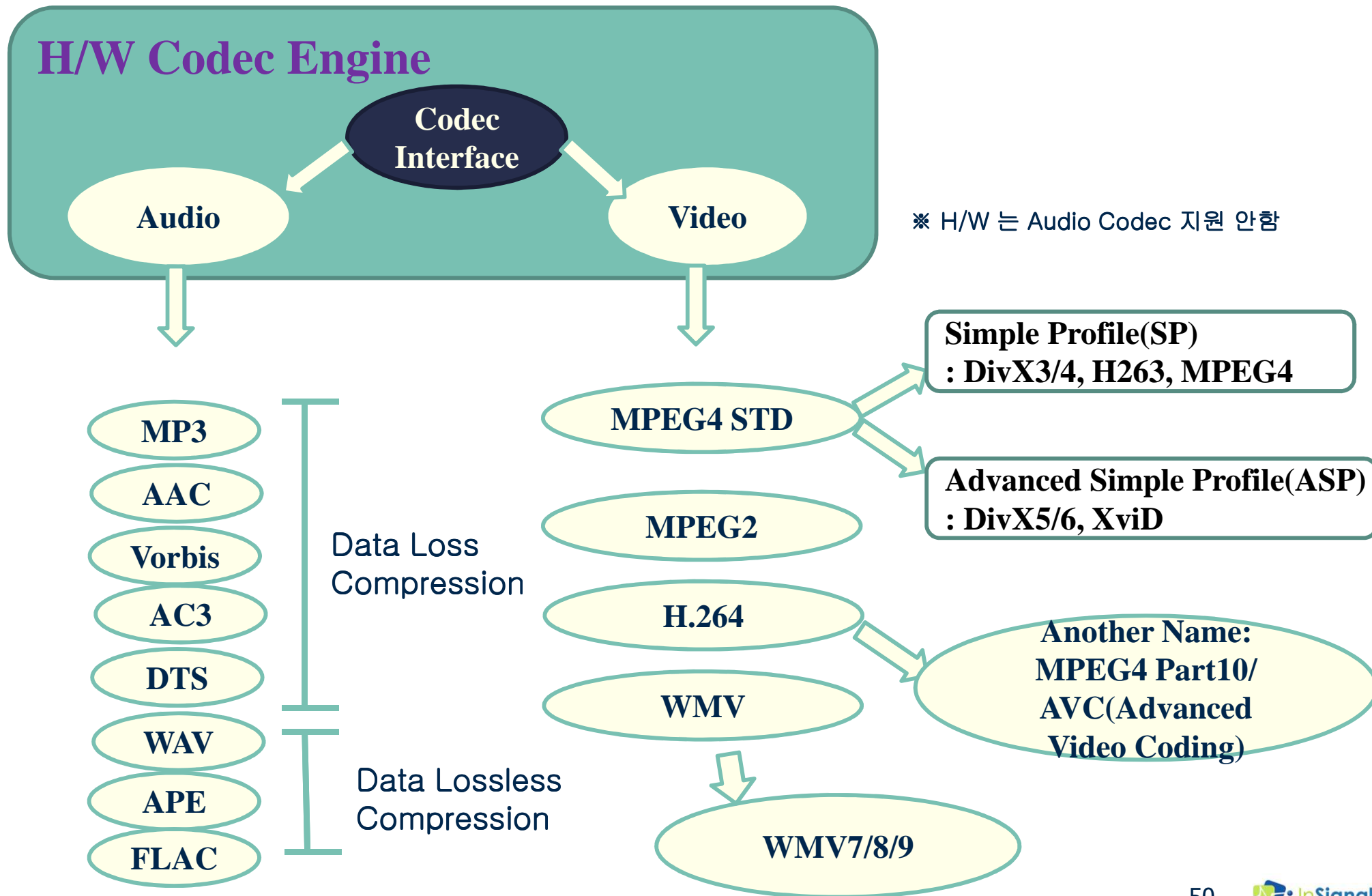


Media Player/Composer part

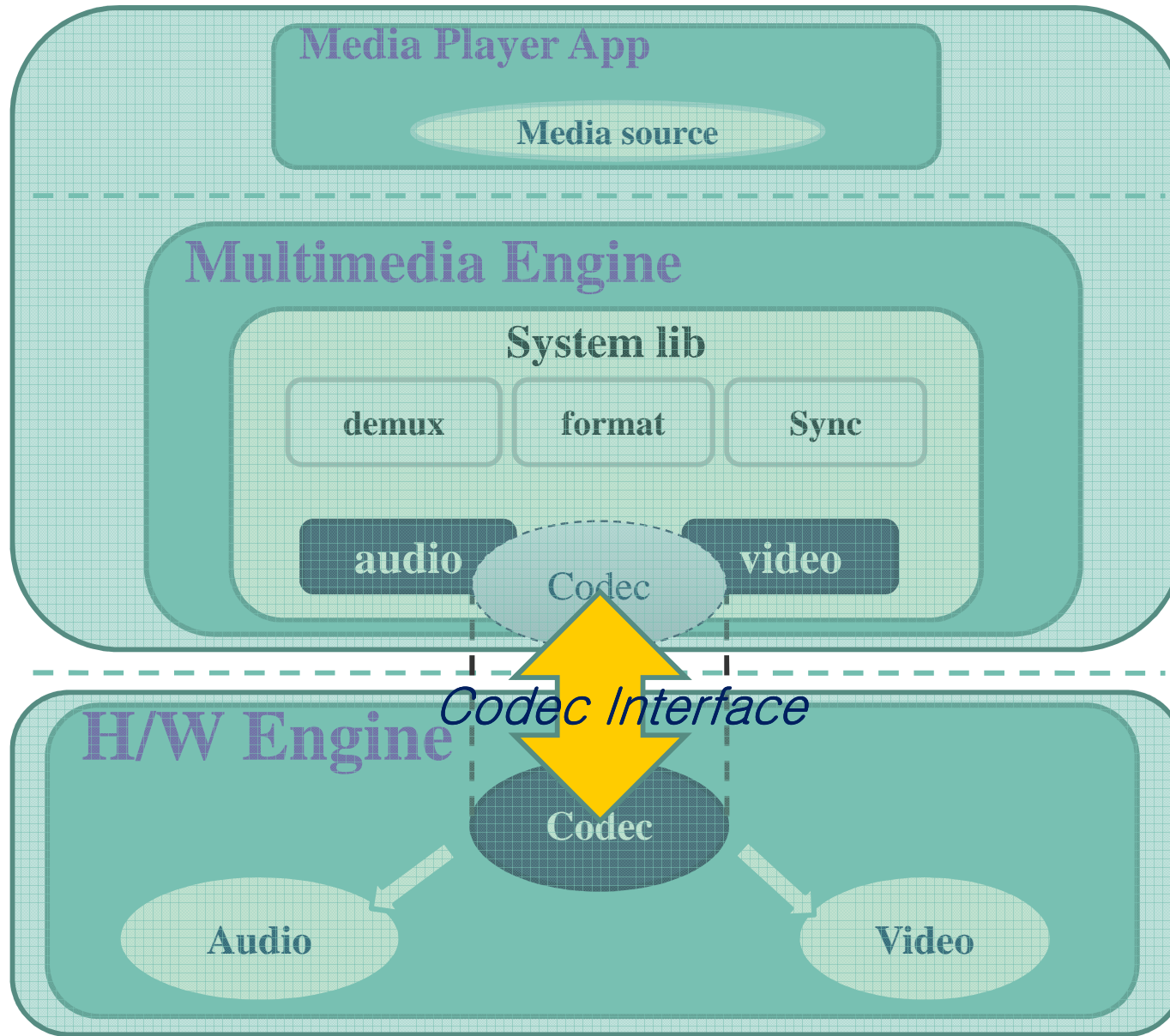


1. Player – 동영상 decoder
2. Composer – 동영상 encoder
3. PLAYER/Composer 가 다루는 부분은 위와 같다 – ex> OpenCORE/StageFright/Skype Engine/Custom engine

H/W Codec Engine



Multimedia Codec Interface



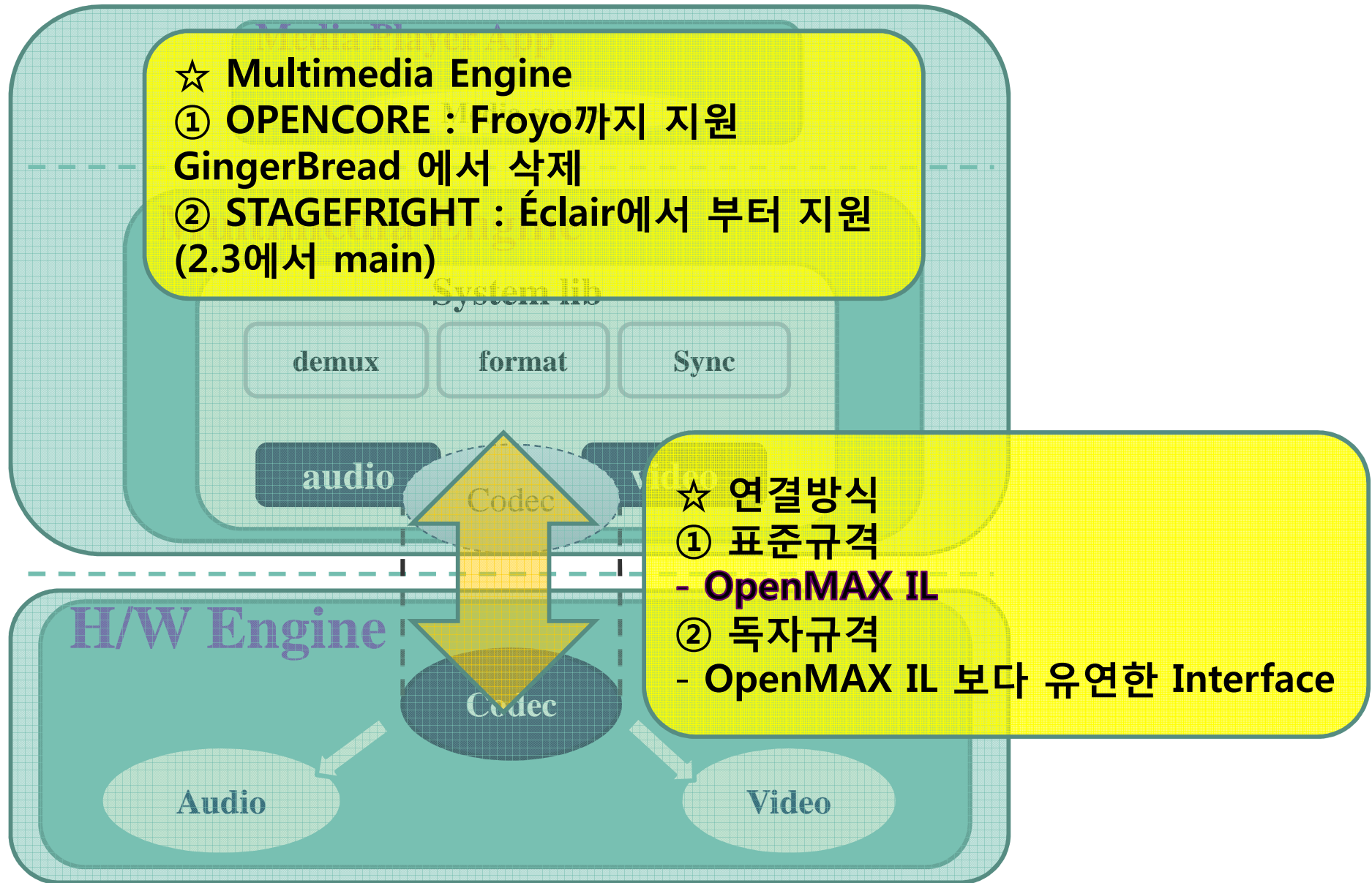
– Codec Interface: H/W or S/W codec handling software

– Codec Interface는 표준화가 되어 있을수도 있고, Player에 따라서는 직접 사용자가 자기만의 표준을 가지고 작성할 수 있음

– 표준화가 되어 있는 경우: OpenMAX IL(Integration Layer)

– 표준화가 되어 있지 않은 경우: Player에 따라 자체적으로 작성. ffmpeg/mplayer/Xine etc...

Android Multimedia Codec Interface



What is OpenMAX

- **OpenMax**

- Khronos Group에서 만들고 있는 표준 API Media Interface
- 시스템에 무관하게 미디어 프로그램 작성이 가능하도록 표준 API를 제공
- 안드로이드 플랫폼에서 OpenMAX 표준규격에 의한 코덱과 미디어 플레이어 설계 가능

- **OpenMax Layer**

- **OpenMax AL (Application Layer)**

- Platform에 무관하게 Media Interface를 제공하고 사용 가능
- 고수준 미디어 제어 프로그램만 작성이 가능
- 저수준에 해당되는 OpenMax IL을 사용하여 미디어 제어 프로그램 작성도 가능

- **OpenMax IL (Integration Layer)**

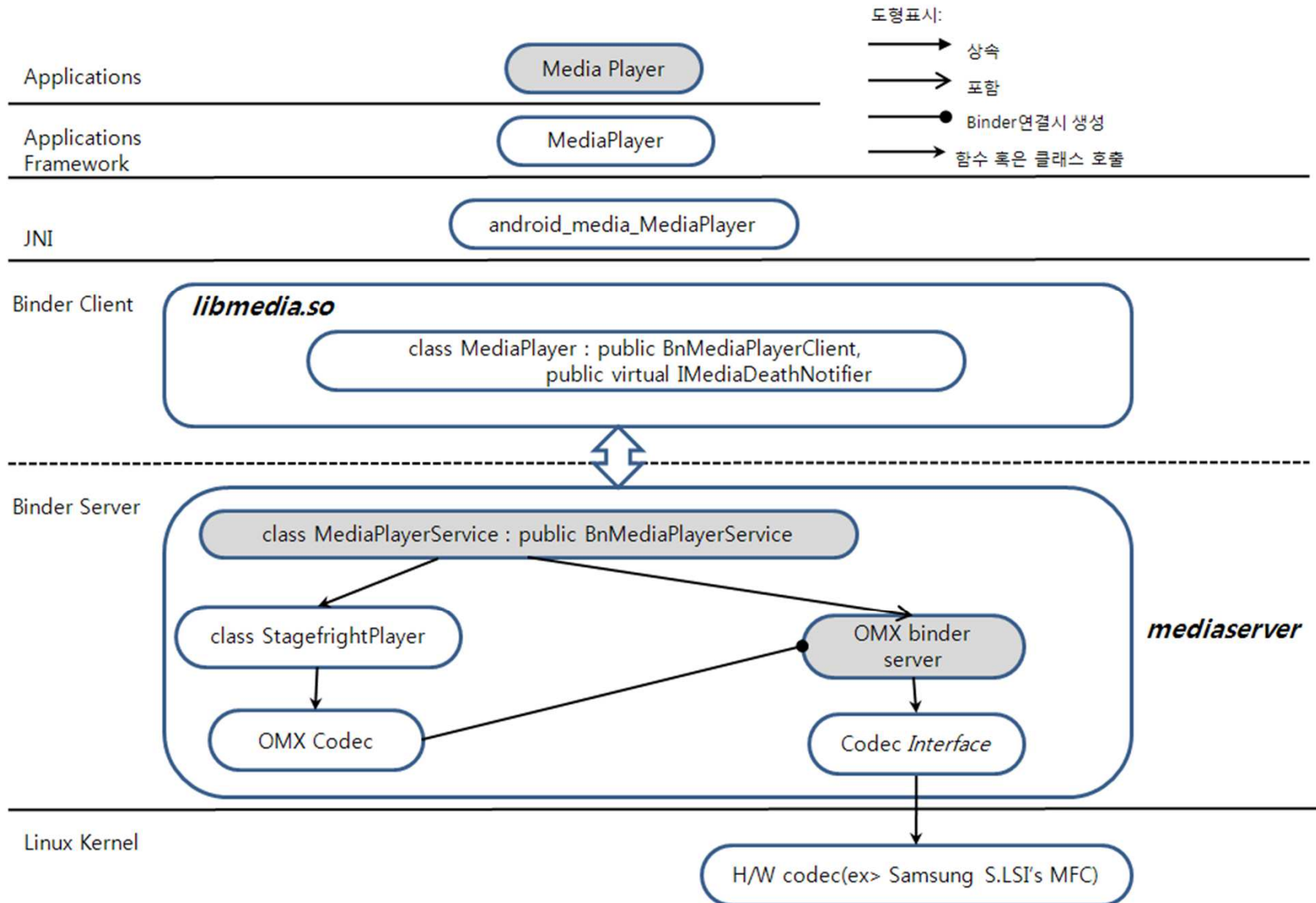
- Multimedia Codec들과 사용자들간의 인터페이스를 제공
- 컴포넌트 기반의 프로그램으로서 부품을 설계하듯이 미디어 플레이어 설계가 가능
- 이는 MS의 DirectShow 구조와 상당히 흡사함 (표준 멀티미디어 규격)

- **OpenMax DL (Development Layer)**

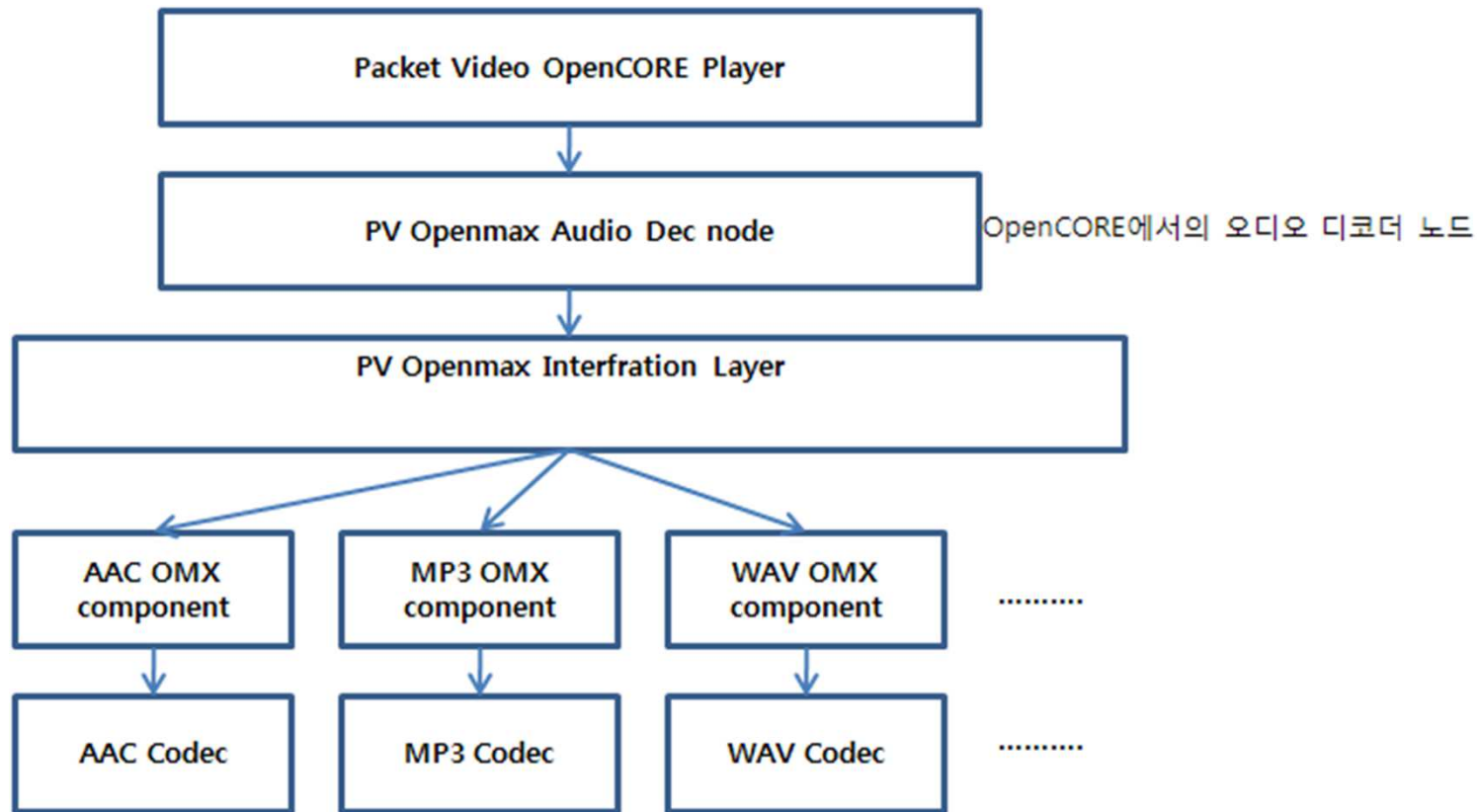
- Audio Codec, Video Codec 개발에 대한 설명
- 일반적으로 Platform 제공자가 Wrapping하여 OpenMax IL로 커버
- 즉, 미디어 플레이어 프로그래머는 접근할 필요가 없는 레이어
- 단, Codec 관련 구조 확인 시에는 참고 가능

Android 2.3 에서의 OMX 인터페이스

■ Android Gingerbread에서의 codec interface

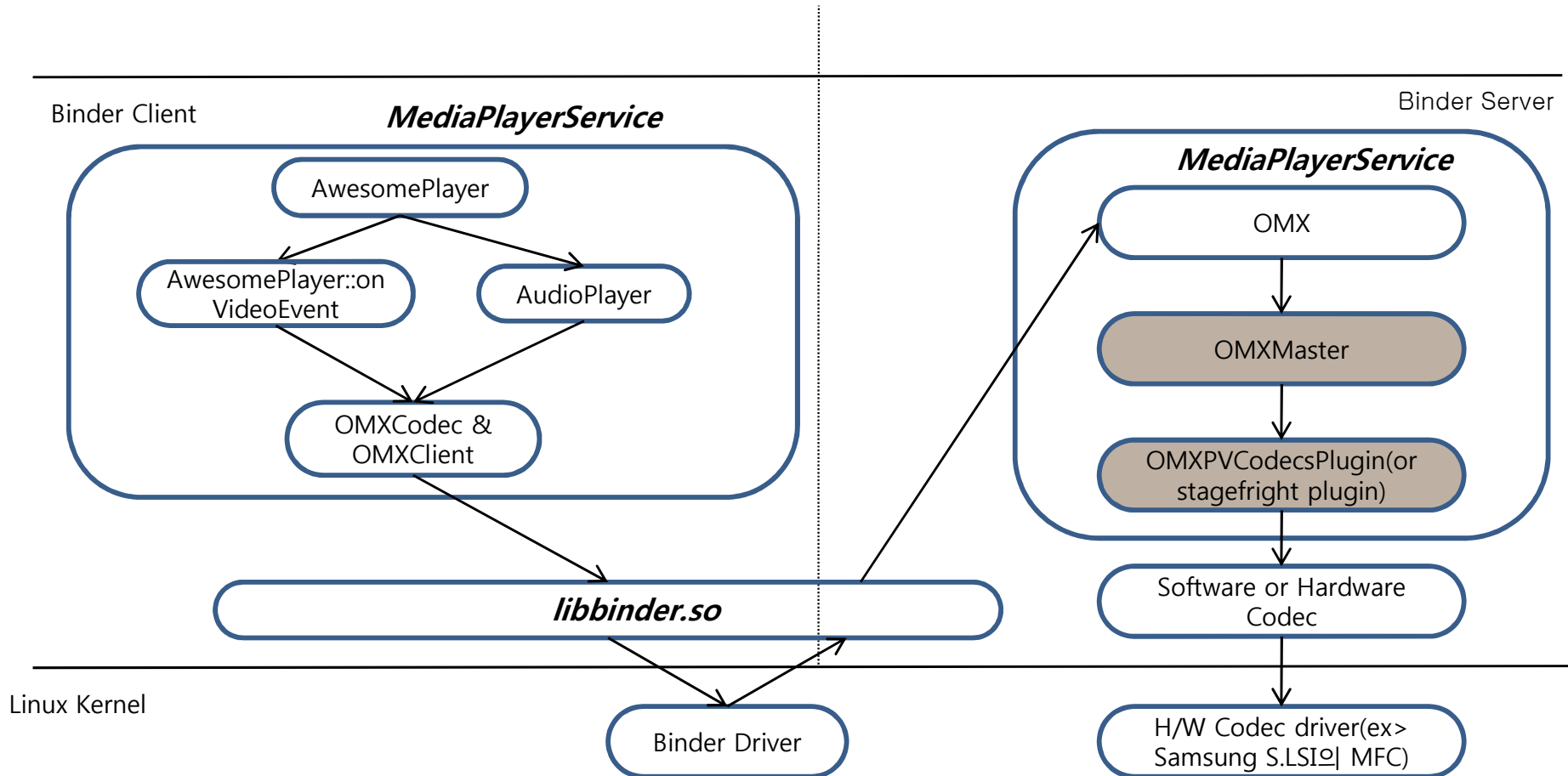


Android에서의 OPENMAX IL의 사용

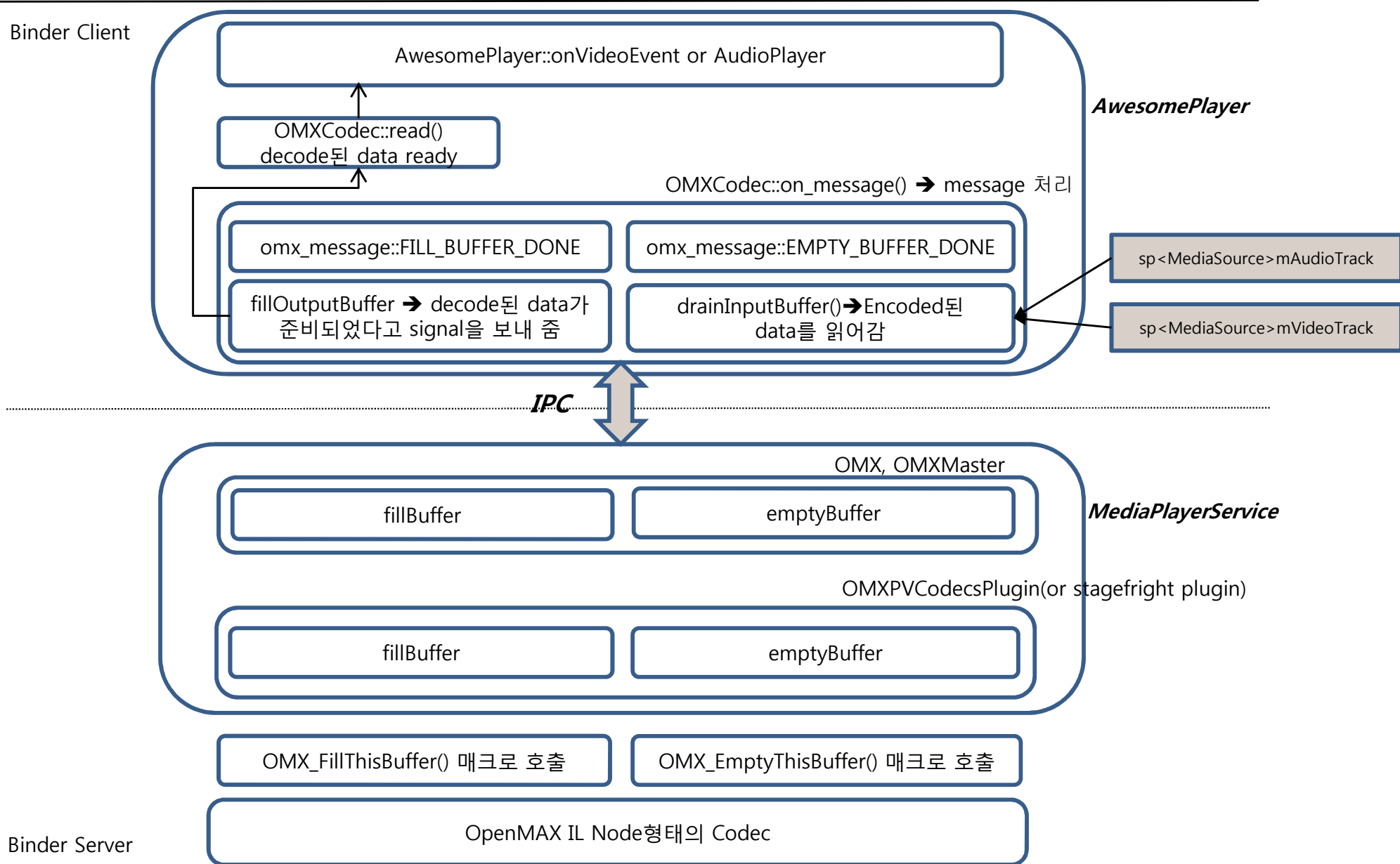


1. OpenCore 에서는 Node라는 개념(OpenCore에서 codec을 사용하는 방식)에 OpenMax IL을 연결함
2. Stagefright 에서도 역시 Codec이 OpenMax IL 형식으로 wrapping 되어있음

Android 2.3 Stagefright의 OMX 구조 #1



Android 2.3 Stagefright의 OMX 구조 #2



Android에서 지원하는 Component - Stagefright

Features : StageFright

Name	Extractor/Writer		Codec	
AAC	parser		DEC	✓
	composer		ENC	✓
AMR	parser	✓	DEC	NB ✓
				WB ✓
	composer	✓	ENC	NB ✓
				WB ✓
WAV	parser	✓		
	composer			
MP3	parser	✓	DEC	✓
	composer		ENC	
AVI	parser			
	composer			
MP4	parser	✓	DEC	✓
	composer	✓	ENC	✓
PVX	parser			
	composer			
SBC			DEC	
			ENC	
MKV	parser	✓	DEC	
	composer		ENC	
MPEG2	parser	✓	DEC	
	composer	✓	ENC	
OGG	parser	✓	DEC	✓
	composer		ENC	
on2			DEC	✓
			ENC	
G711			DEC	✓
			ENC	
H264			DEC	✓
			ENC	✓

안드로이드 지원 S/W CODEC(기본지원)

Type	Format	Enc	Dec	Details	File Types Supported
Audio	AAC LC/LTP		○	Mono/Stereo contents in any combination of standard bit rates up to 160kbps and sampling rates from 8 to 48kHz	3GPP(.3gp) and MPEG-4(.mp4, m4a). No support for raw-AAC
	HE-AACv1(AAC+)		○		
	HE-AACv2(@AAC+)		○		
	AMR-NB	○	○	4.75 to 12.2 kbps samples @8kHz	3GPP(.3gp)
	AMR-WB		○	9 rates from 6.60kbps to 23.85kbps samples @16kHz	3GPP(.3gp)
	MP3		○	Mono/Stereo 8~320kbps constant(CBR) or Variable(VBR)	MP3(.mp3)
	MIDI		○	MIDI Type 0 and 1. DLS Version 1 and 2. XMF and mobile XMF. Support for ringtone formats RTTTL/RTX, OTA, and iMelody	Type 0 and 1(.mid, xmf, mxmf) Also RTTTL/RTX(.rtttl, rts), OTA(.ota) and iMelody(.imy)
	Ogg Vorbis		○		Ogg(.ogg)
	PCM/WAVE		○	8- and 16-bit linear PCM (rates up to limit of hardware)	WAVE(.wav)
Image	JPEG	○	○	Base+progressive	JPEG(.jpg)
	GIF		○		GIF(.gif)
	PNG		○		PNG(.png)
	BMP		○		BMP(.bmp)
Video	H.263	○	○		3GPP(.3gp) and MPEG-4(mp4)
	H.264 AVC		○		3GPP(.3gp) and MPEG-4(mp4)
	MPEG-4SP		○		3GPP(.3gp)

1. Packet Video에서 작성한 OpenCORE Framework/Stagefright 내에 포함되어 있음
2. H/W Codec을 사용하고 싶다면, 이런 S/W Codec들을 제거하고, H/W Codec들을 OpenMAX IL API로 wrapping하여 플러그인 시켜야 함.
3. Interface가 복잡하기 때문에 제어가 힘들다는 단점이 있음

Android에서의 codec을 사용하는 방법(1)

- Android Multimedia Engine를 사용할 경우(Samsung SoC의 경우)
 - ✓ Froyo: OpenCORE에 포함되어 있는 OpenMAX IL 구조를 이용
 - ✓ Gingerbread: StageFright에 포함되어 있는 OpenMAX IL 구조를 이용
 - ✓ 장점
 - 기존에 구성되어 있는 코덱 코드를 그대로 사용이 가능함
 - ✓ 단점
 - Skype와 같은 외부 player engine에서 OpenMAX IL을 지원하지 않을 경우 Android version에 따라 기존의 코드(OpenCORE/StageFright)에서 사용하는 OpenMAX IL Interface를 재작업 후 Skype와 연결해야 한다
 - Skype와 같은 custom player/composer의 경우 자체 표준이 존재할 경우가 많다

Android에서의 codec을 사용하는 방법(2)

- Android Multimedia Engine을 사용하지 않는 경우
 - ✓ SoC의 코덱(ex> Samsung S.LSI의 MFC) 사용법을 기존의 코드를 참고하여 새로 작성
 - ✓ 이후에 특정 Video Engine의 Interface와 맞춰서 직접 hardware codec을 제어하는 코드를 작성한다
 - ✓ 장점
 - skype 혹은 다른 엔진에서 원하는 방식대로 코드 구성이 가능
 - ✓ 단점
 - 기존의 코드를 재작성 해야 한다.

Media In/Output(MIO)

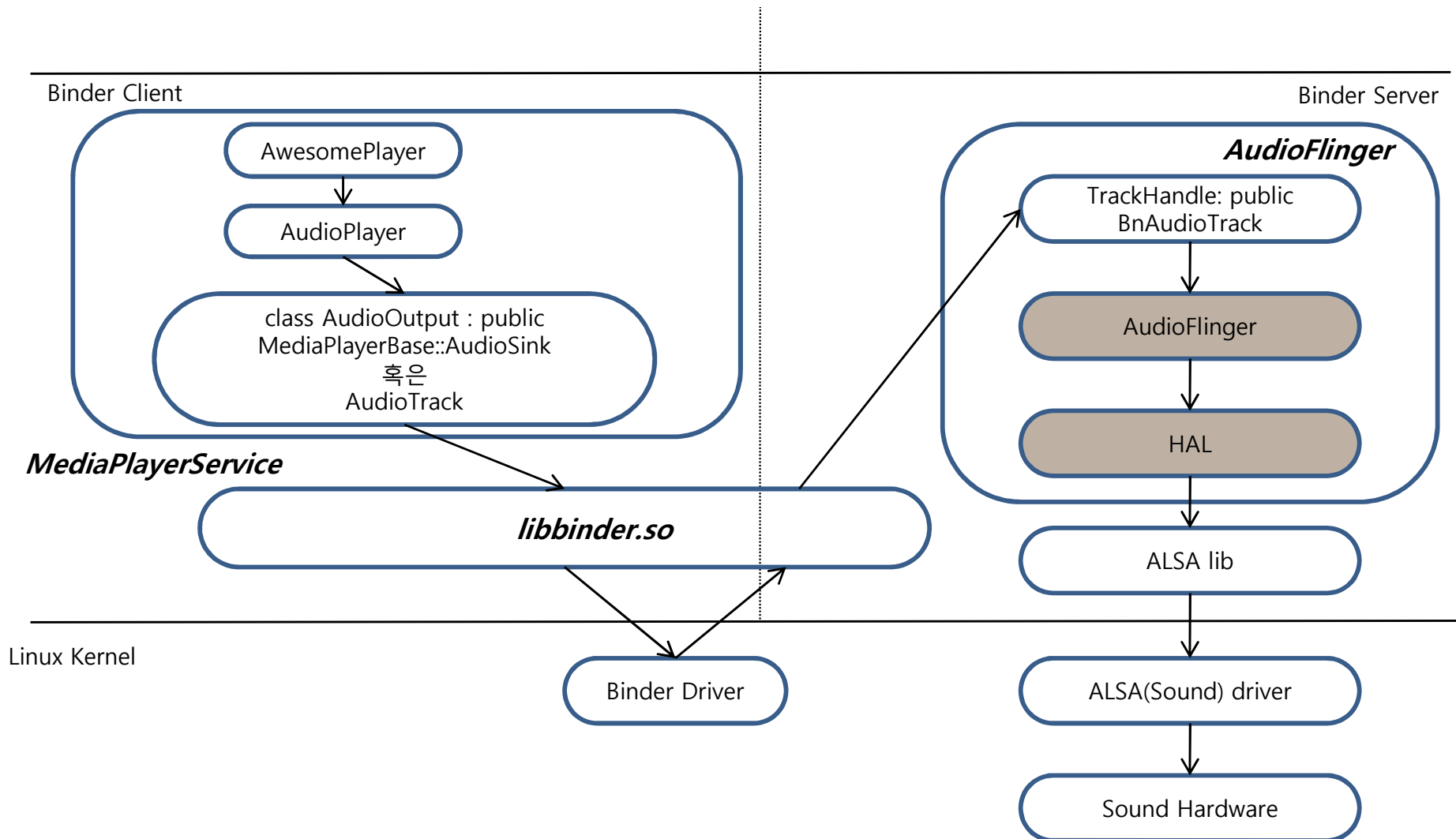
- Decoding된 최종 결과물의 출력
 - ✓ Audio
 - AudioSink
 - ✓ Video
 - SurfaceFlinger surface
 - vs.
 - Overlay

Audio output

■ AudioSink

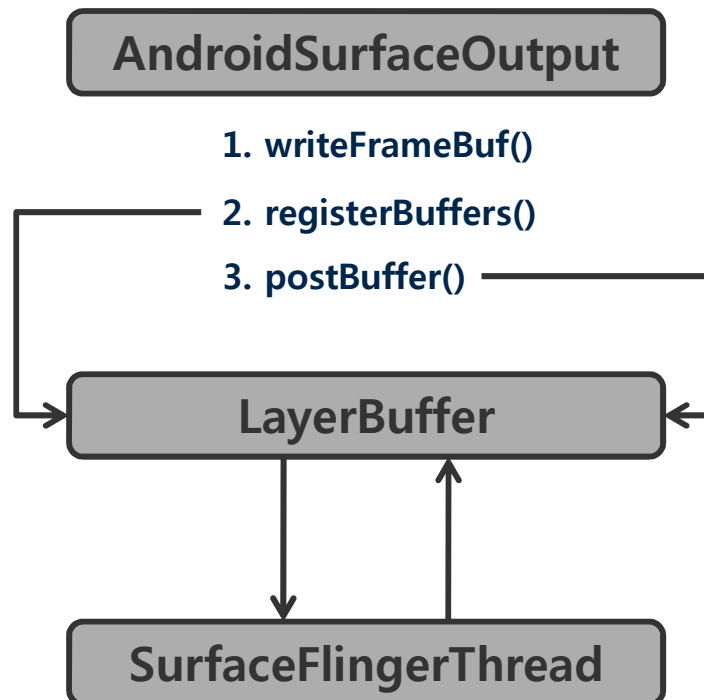
- ✓ 일반적인 application의 경우 – AudioTrack 사용
- ✓ MediaPlayerService는 AudioSink 사용
 - AudioSink로부터 상속받은 AudioOutput class를 사용한다
 - AudioSink는 실제로 AudioTrack이다
- ✓ Callback function위주의 동작 – AudioTrack은 직접 제어하는 경우가 많음
- ✓ 선언
 - AudioSink
 - frameworks/base/include/media/MediaPlayerInterface.h
 - AudioOutput
 - frameworks/base/media/libmediaplayerservice/MediaPlayerService.h
- ✓ Media Engine에서는 Audio Thread를 동작시켜서 Audio callback 함수를 이용해서 data를 output

Android 2.3 Stagefright의 Audio output

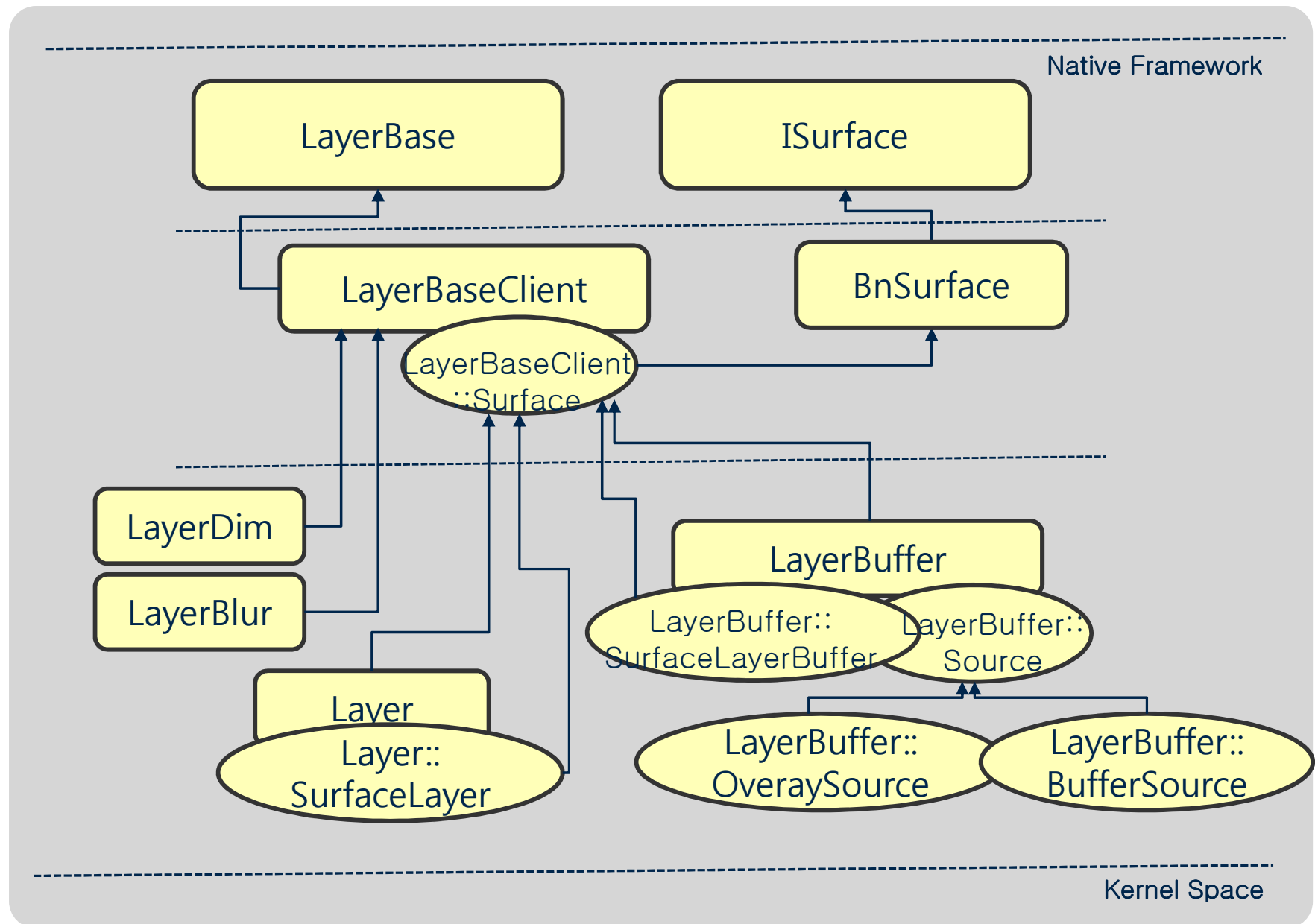


Video output

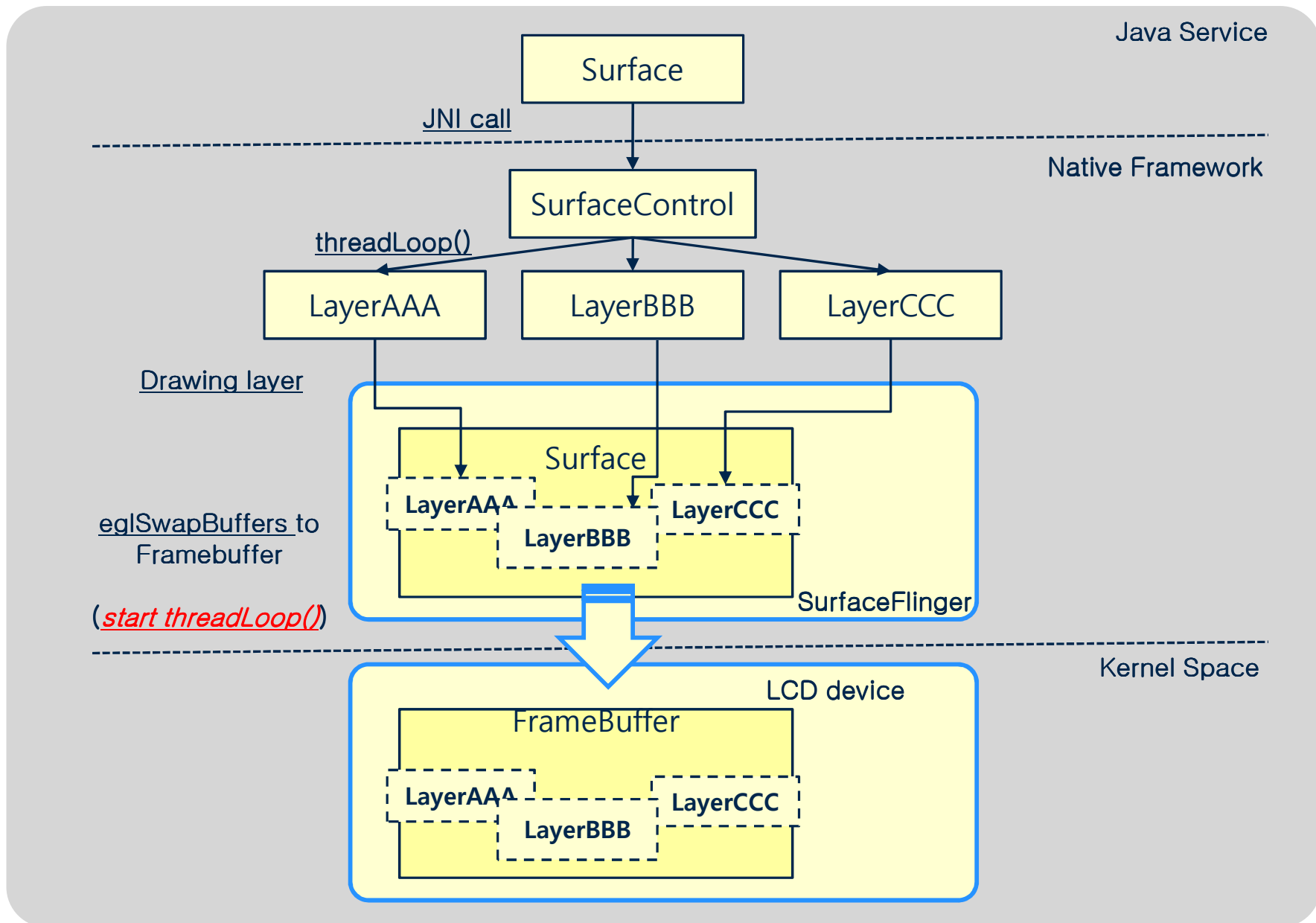
- Android surface output은 기본적으로는 SurfaceFlinger의 LayerBuffer를 이용
- 기본적으로 모든 동영상 display routine은 SurfaceFlinger사용
 - ✓ BufferSource를 사용하는 경우
 - ✓ OverlaySource를 사용하는 경우 – H/W engine 사용
- OpenCORE에서의 Surface Output routine



Video output class flow

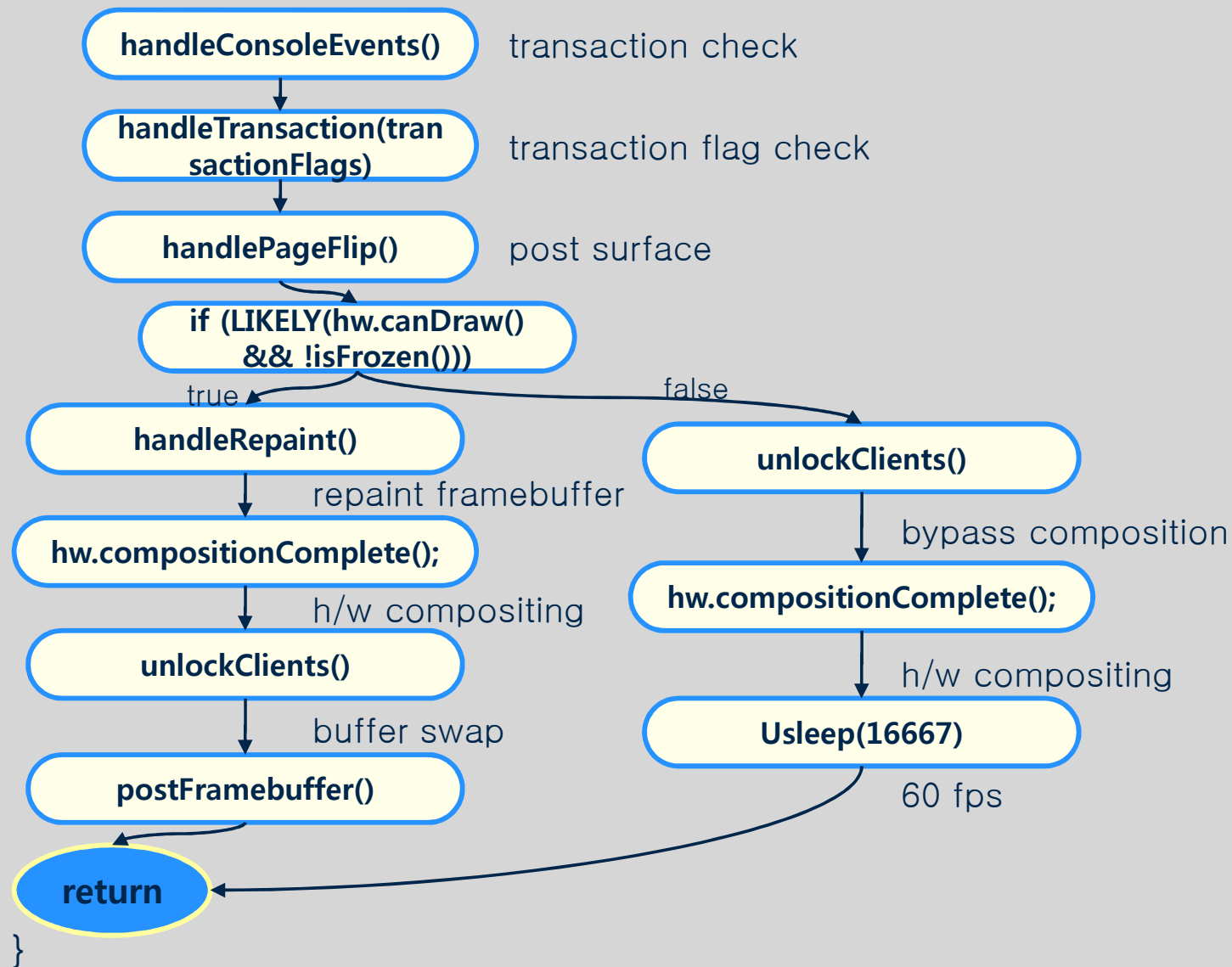


SurfaceFlinger display flow(1)



SurfaceFlinger display flow(2)

SurfaceFlinger::threadLoop(){



Overlay를 이용한 H/W engine 사용

- 기본적으로 모든 동영상 display routine은 SurfaceFlinger사용
- SurfaceFlinger의 LayerBuffer class를 이용한 Surface display routine 사용
- LayerBuffer를 사용할 경우
 - ✓ BufferSource를 사용하면 직접 LCD device driver(ex> fb)를 이용
 - ✓ OverlaySource를 사용할 경우 H/W Layer를 이용
- Overlay를 이용할 경우는 LayerBuffer를 이용해서 display Entry만 유지
 - ✓ 실제 동작은 H/W Overlay로 데이터는 직접 전송하는 구조
 - ✓ 대부분 YUV → RGB 변환 루틴은 Overlay driver를 이용해서 처리한다
- hardware/libhardware/modules/overlay/* 혹은 vendor의 overlay source참조

Multimedia 와 SurfaceFlinger와의 관계

