Noisy Androids

Mastering the Android Media Framework

Dave Sparks
15-August-2009

Agenda

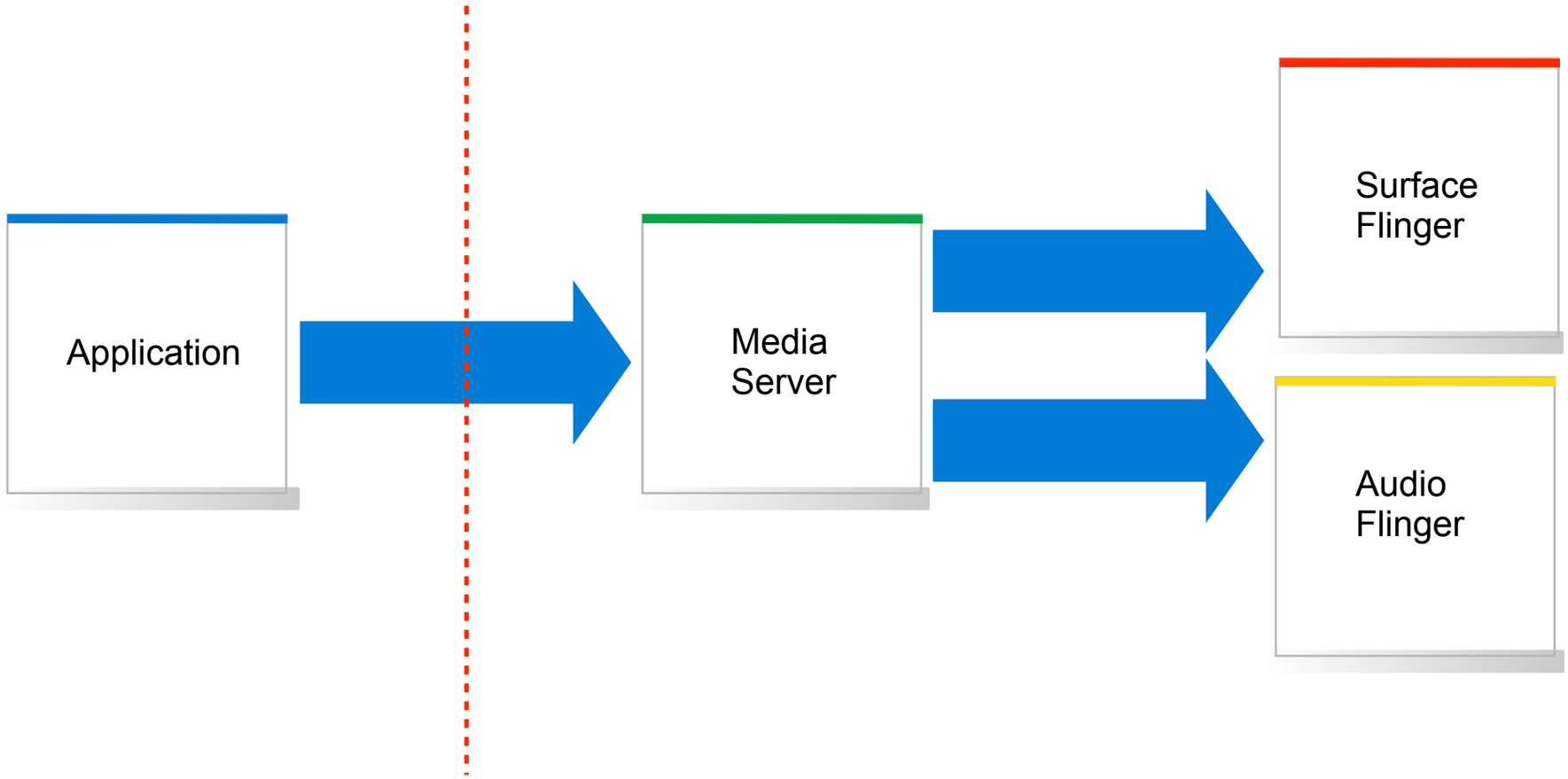
- Media Framework architecture
- Security
- Android 1.5 “Cupcake”
- Common Problems
- Q & A

Architecture

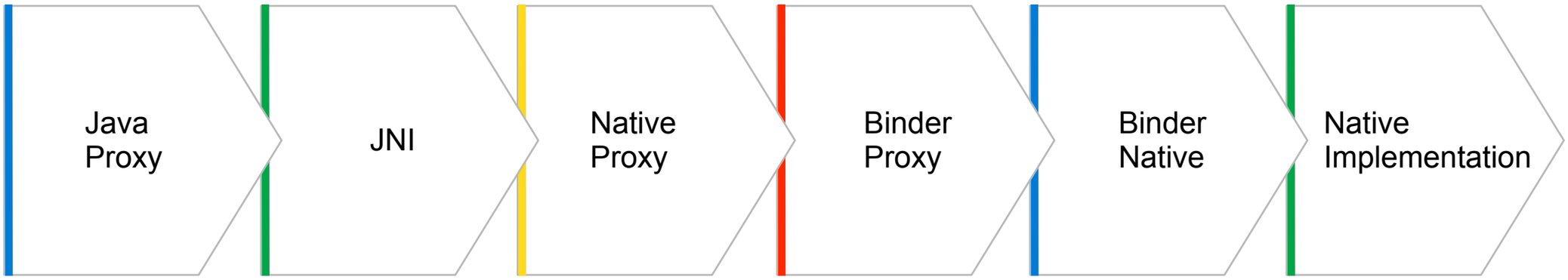
Design Goals

- Simplify application development
- Share resources in multi-tasked environment
- Strong security model
- Room for future growth

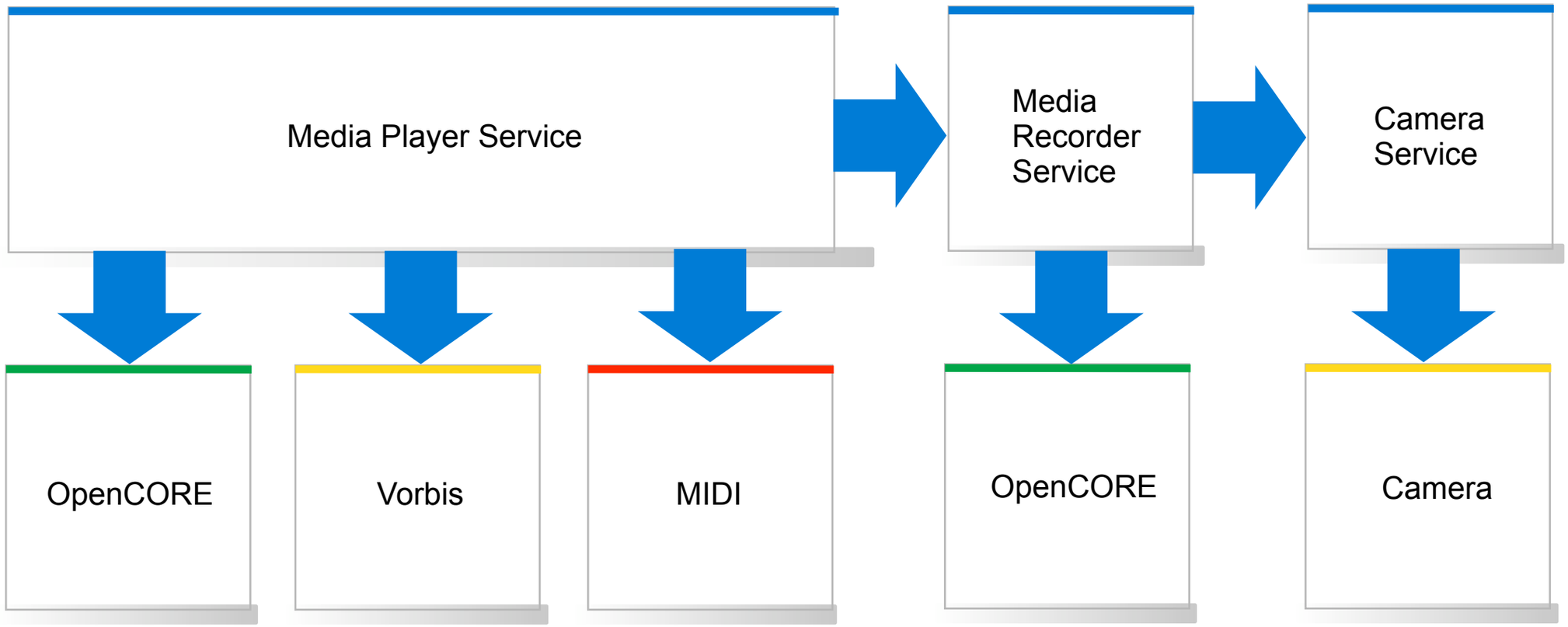
Media Framework



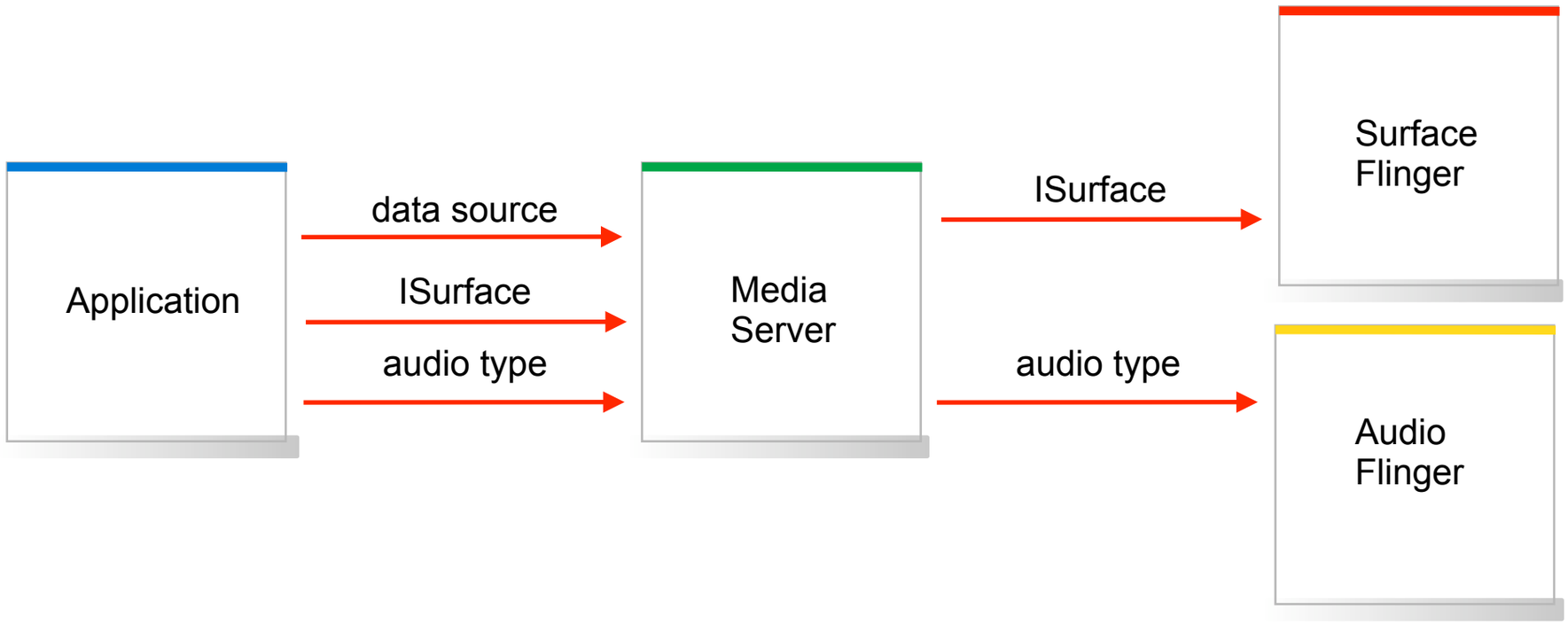
Typical Stack for Media Function Call



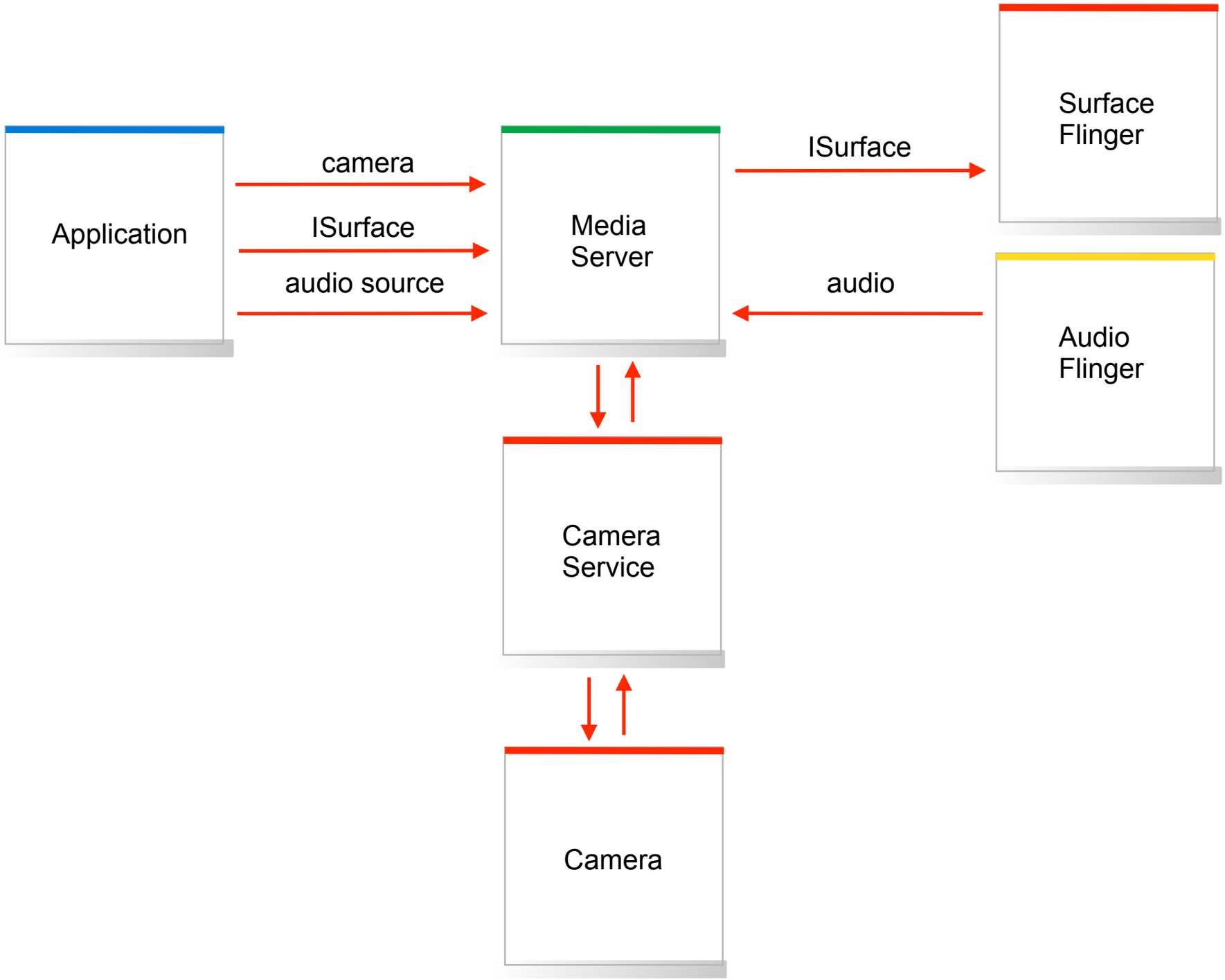
Media Server Process



Media Playback



Media Recorder



Security

Media Server Security

- Isolate file parsers and codecs in a separate process
- Centralize hardware resource management and security
- Enforce permissions on camera and audio record
- Media server runs at reduced privileges
- Applications grant file access on as-needed basis

Media Server Security (continued)

- Media server has read/write access to SD card only
- Applications typically pass in an open file descriptor
- Only the media server process has direct access to hardware resources
- Applications must include permissions in their manifest to access camera and microphone

How does sandboxing defeat a hypothetical attack?

- Assume we have a buffer overrun vulnerability in a codec
- Assume that it is possible to construct a malicious stream that can run arbitrary code in the media server process
- Media server runs in another process as a user with limited privileges and cannot access memory in the application process
- Media server can access SD card, but not the data or system partitions
- No data from application is vulnerable unless the application has explicitly given permission to read or write the file

V1.5 Features

AudioTrack and AudioRecord

- Expose raw PCM audio streams to Java
- AudioTrack: Write PCM audio directly to mixer engine
- AudioRecord: Read PCM audio directly from mic
- Callback mechanism for threaded applications
- Static buffer for playing sound effects

JET Interactive MIDI Engine

- Based on MIDI - file sizes can be small
- DLS support allows for better quality instruments
- Precise synchronization for layered MIDI tracks
- Native code - very efficient
- Synchronization callbacks to Java for rhythm games
- Open source engine and creation tools
- VST plugin - use it inside your favorite DAW tool

Common Problems

Problem: Volume control behavior is inconsistent

- Volume control is overloaded
- AudioManager has a default strategy for controlling volume:
  - If in-call, adjust in-call volume
  - If ringing, mute ringer
  - If media track is active, adjust media volume
  - Otherwise, adjust ringtone volume
- In an application that plays sounds periodically, the volume behavior is not consistent

Solution: Set the default stream type

```
import android.app.Activity;
import android.media.AudioManager;
...
public void onCreate(Bundle icicle)
{
  super.onCreate(icicle);
  setVolumeControlStream(AudioManager.STREAM_MUSIC);
  ...
}
```


Problem: Unable to play file from resource?

```
MediaPlayer mp = new MediaPlayer();
try {
  mp.setDataSource(“res:com.myapp.raw.test”);
  mp.prepare();
} catch (IOException e) {
  Log.e(“Error ” + e.print() + “ opening media player”);
}
```


Solution: Use AssetFileDescriptor

```
AssetFileDescriptor afd =
  context.getResources().openRawResourceFd(resId);

set.DataSource(afd.getFileDescriptor(),
               afd.getStartOffset(),
               afd.getLength());

afd.close();
```


Problem: Out of MediaPlayers!

```
MediaPlayer mpArray = new Object[50];
for (int i = 0; i < 50; i++) {
  MediaPlayer mp = MediaPlayer.create(soundName[i]);
  mp.prepare();
  mpArray[i] = mp;
}
```


Solution: Reuse MediaPlayer

- Call release() and set to null
- Or call reset(), then setDataSource()
- Limit to 2 or 3 maximum

Problem: CPU Overloaded

```
MediaPlayer sound1 = new MediaPlayer();
MediaPlayer sound2 = new MediaPlayer();
MediaPlayer sound3 = new MediaPlayer();
// call setDataSource, prepare, etc.
...
// later on...
sound1.start();
sound2.start();
sound3.start();
...
// CPU is bogging down here...
```

Solution: Use SoundPool

```
import android.media.SoundPool;
import android.media.AudioSystem;

Context context = getApplicationContext();
sp = new SoundPool(maxStreams, AudioSystem.STREAM_MUSIC, 0);
int snd1 = sp.load(context, res.Raw.pow);
int snd2 = sp.load(context, res.Raw.blam);
int snd2 = sp.load(context, res.Raw.biff);
...
sp.play(snd1, leftVol, rightVol, priority, loop, rate);
...
sp.play(snd2, leftVol, rightVol, priority, loop, rate);
...
sp.play(snd3, leftVol, rightVol, priority, loop, rate);
```

Q & A