Exposing the Android Camera Stack

Balwinder Kaur, Principal Software Architect
Joe Rickson, Principal Software Engineer

Android Builder's Summit 2012
2.14.2012

© 2012 Aptina Imaging Corporation. All rights reserved. Products are warranted only to meet Aptina's production data sheet specifications. Information, products, and/or specifications are subject to change without notice. All information is provided on an "AS IS" basis without warranties of any kind. Dates are estimates only. Drawings not to scale. Aptina and the Aptina logo are trademarks of Aptina Imaging Corporation. All other trademarks are the property of their respective owners.

Agenda

- Hardware Independent Section
  - Overview of android.hardware.Camera
  - Prominent Camera Use Cases
  - High Level Architecture
  - JNI Layer
  - Native Camera service
  - Media Subsystem Interactions
- Hardware Dependent Modules
  - Camera Hardware Abstraction Layer
  - Camera Device Driver
  - Camera Hardware Architecture
- Future Trends
- Q&A

| © 2012 Aptina Imaging Corporation

Section I

Hardware Independent Camera Stack

© 2012 Aptina Imaging Corporation

Overview of android.hardware.Camera

6 Classes

Camera
Camera.CameraInfo
Camera.Parameters
Camera.Size
Camera.Face
Camera.Area

7 Callback Interfaces

- Camera.AutoFocusCallback
- Camera.ErrorCallback
- Camera.FaceDetectionListener
- Camera.OnZoomChangeListener
- Camera.PictureCallback
- Camera.PreviewCallback
- Camera.ShutterCallback

© 2012 Aptina Imaging Corporation

Handling Camera Hardware Fragmentation

Camera.Parameters class provides a “dumb” pipe to the hardware

Hardware capabilities can be queried for capabilities.

As an example, for Video Stabilization Feature

- isVideoStabilizationSupported()
- setVideoStabilization(boolean)
- getVideoStabilization()

| © 2012 Aptina Imaging Corporation

Android 4.0 Camera Features

Feature	Platform Feature with API	In-built Camera Application Code	Proprietary Solution	API Level
Face Detection	✓			14
Face Recognition			✓	14
Panoramic Stitch		✓		14
Video Snapshot	✓			14
AE & AWB Lock	✓			14
Continuous Focus Mode	✓			14
Region Of Interest (AE, AWB and AF)	✓			14
Zero Shutter Lag*				14
Video Stabilization	✓			15
Live Effects on Images / Video**	✓			14

* There is no API for ZSL. It is a hardware dependent feature.
** android.media.Effect

AE : Auto Exposure

AWB : Auto White Balance

AF : Auto Focus

© 2012 Aptina Imaging Corporation

Prominent Camera Use Cases

Main Use Cases

Live Preview of Camera Stream

Live Preview + copy of the Frame returned to the application

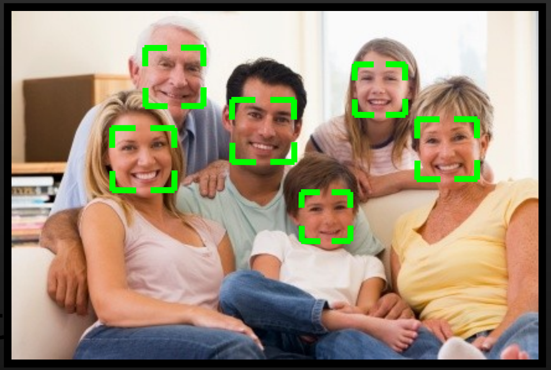
Capture a frame

Video Recording of a Camera Stream

Secondary Use Cases

Configuring the Camera

Receiving more than an image back . e.g. face detect



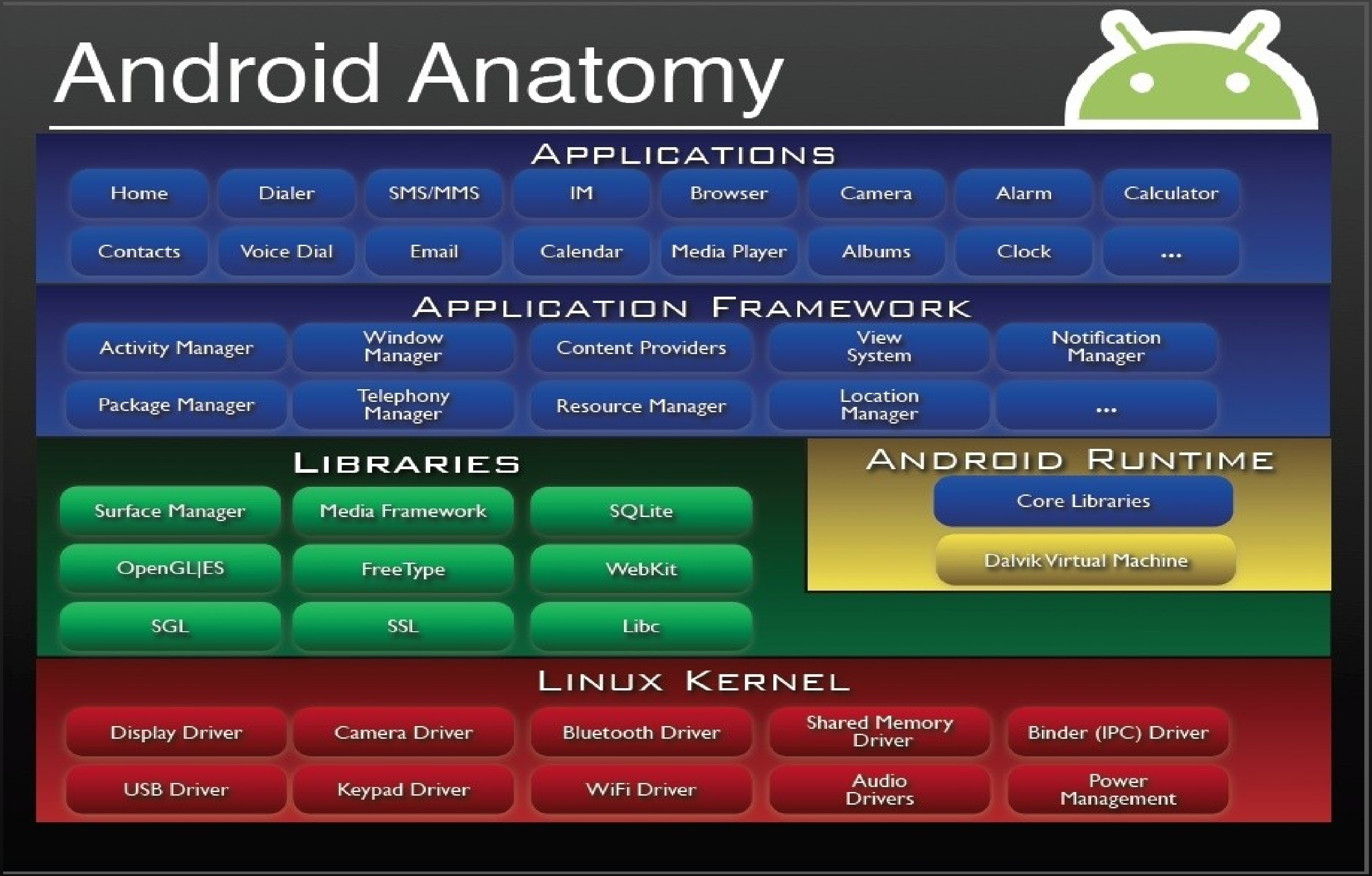
Event Callbacks: Shutter Clicked, AutoFocus Achieved

© 2012 Aptina Imaging Corporation

High Level Architecture

| © 2012 Aptina Imaging Corporation

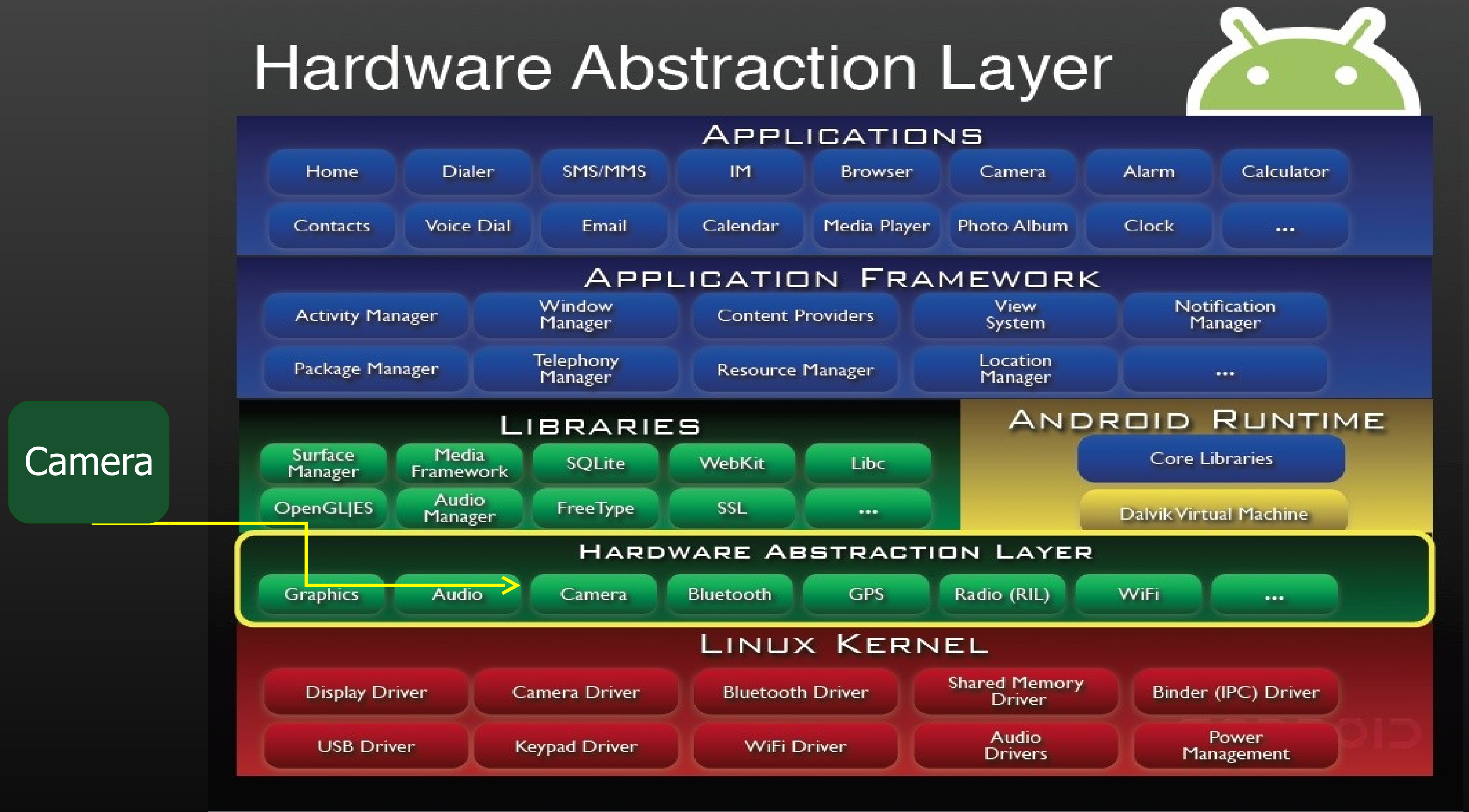
Android High Level Architecture



Source: Android Anatomy and Physiology, Google IO 2008

© 2012 Aptina Imaging Corporation

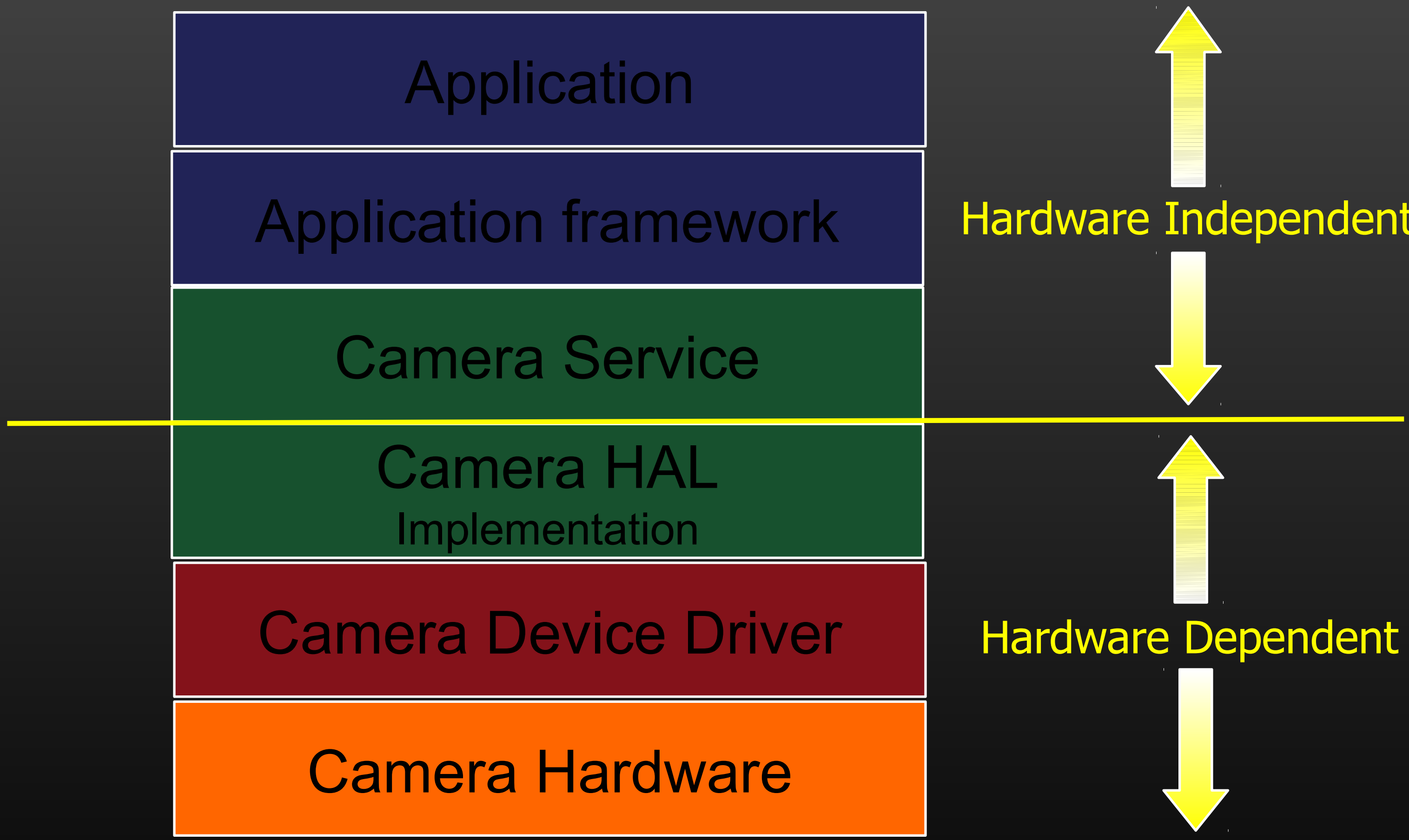
Hardware Abstraction Layer



Source: Android Anatomy and Physiology, Google IO 2008

© 2012 Aptina Imaging Corporation

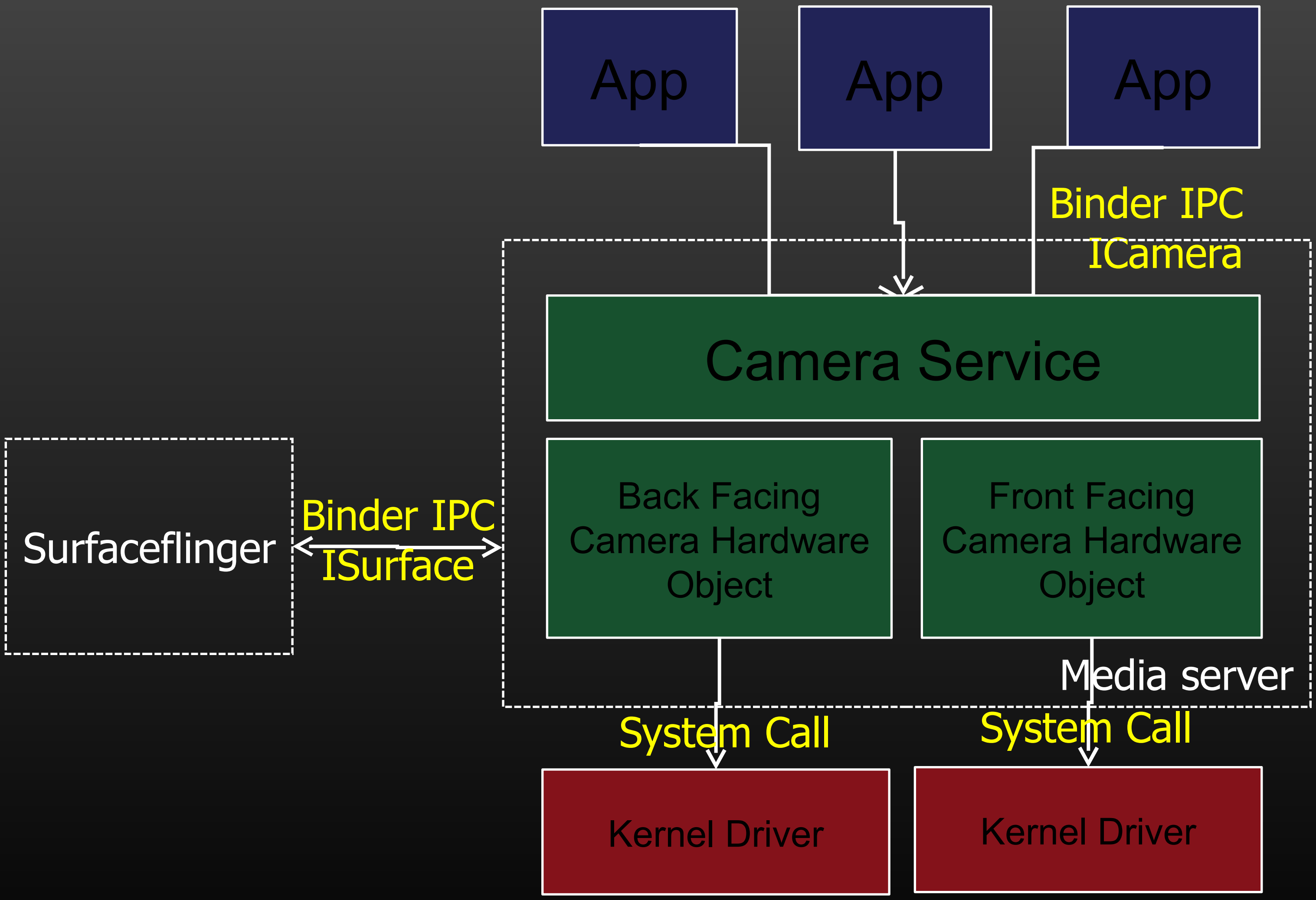
Camera Subsystem



HAL = Hardware Abstraction Layer

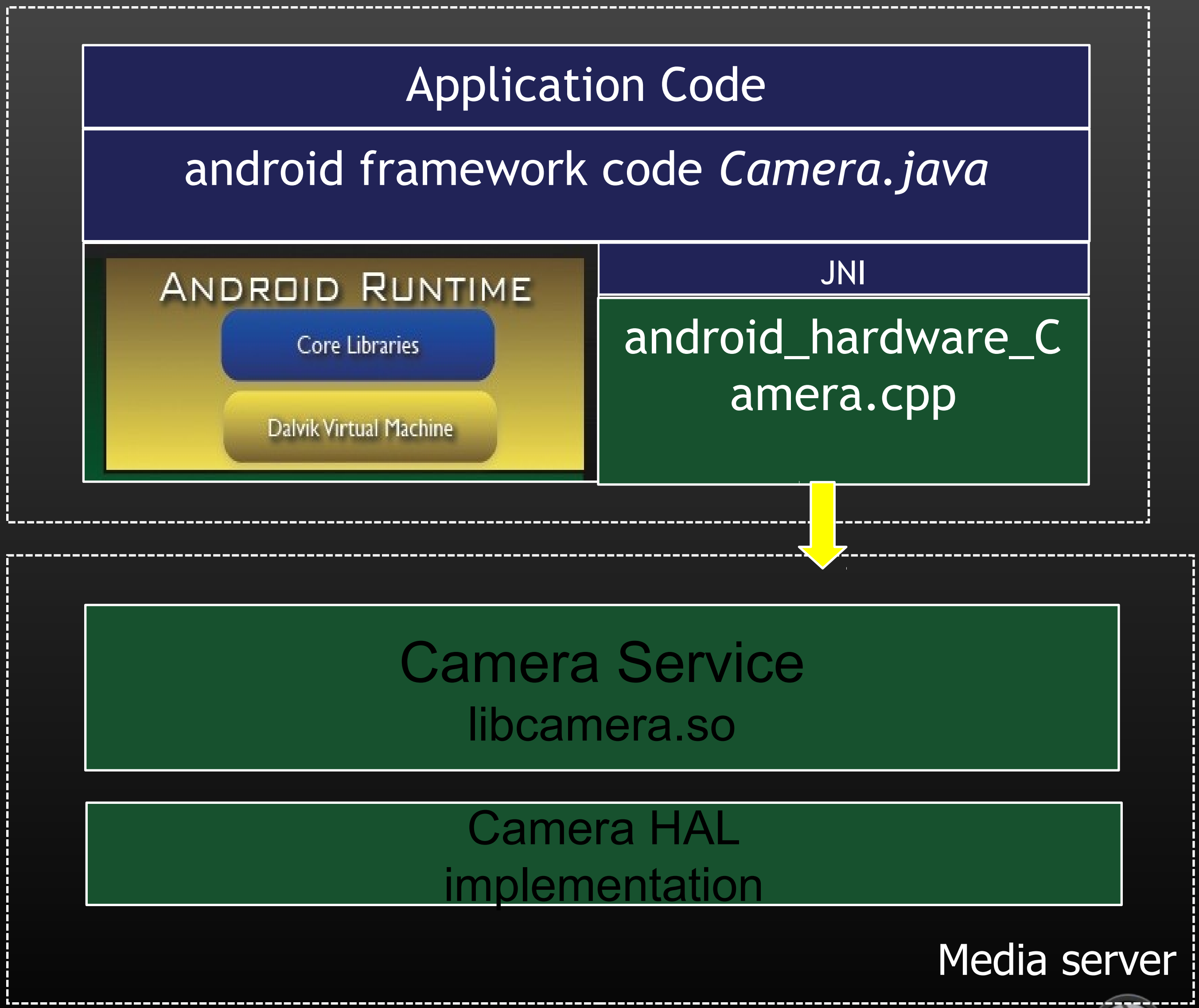
| © 2012 Aptina Imaging Corporation

Process View



 | © 2012 Aptina Imaging Corporation

Inside the Camera App

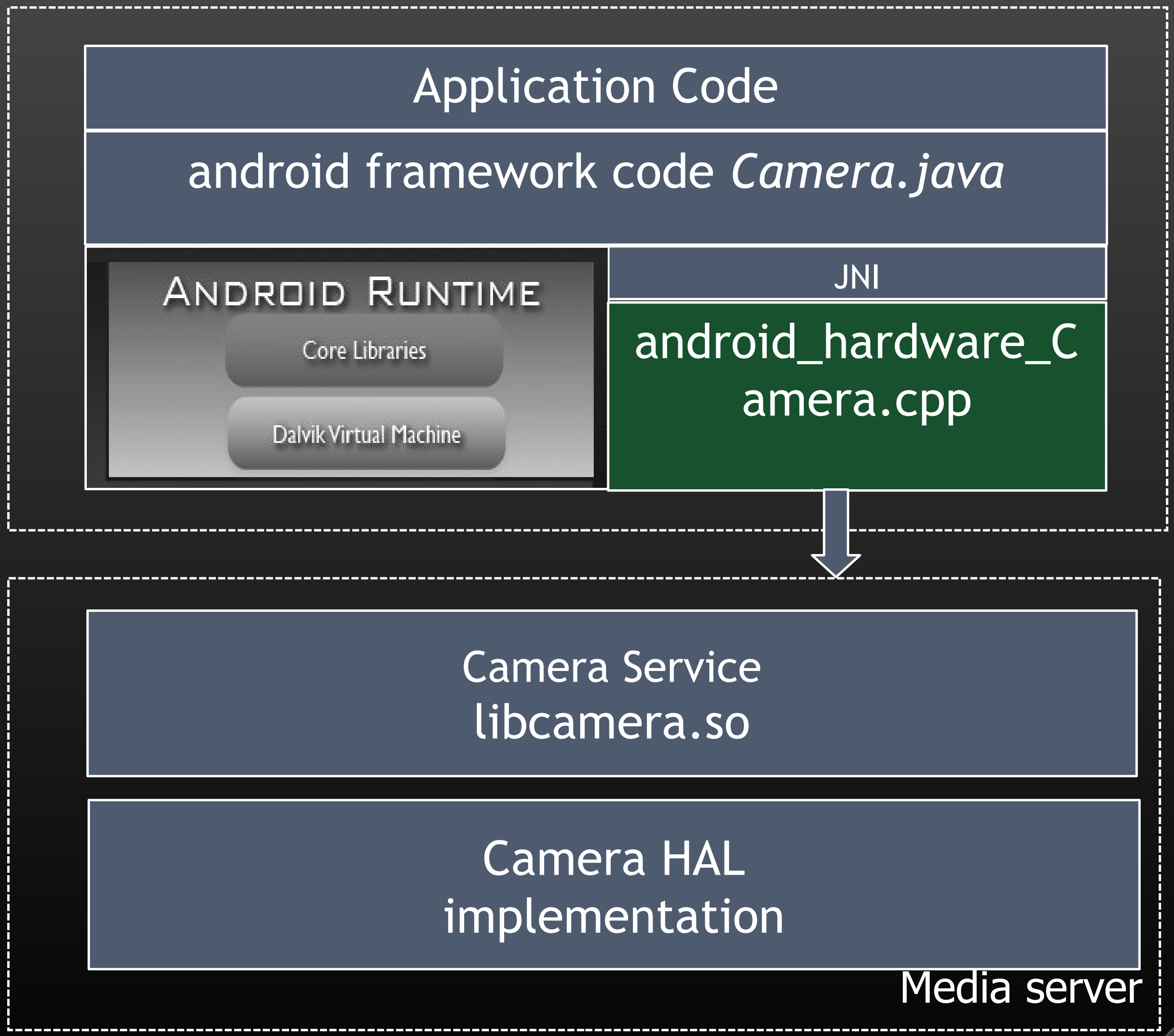


© 2012 Aptina Imaging Corporation

JNI Layer

© 2012 Aptina Imaging Corporation

JNI Layer



| © 2012 Aptina Imaging Corporation

android_hardware_Camera

Creates a persistent context for callbacks from native code to Java (JNICameraContext)

Holds references to the Java Camera, Face and Area objects.

If a Copy of the Preview Frame is requested by the app, then the copy from native to java buffers is done here.

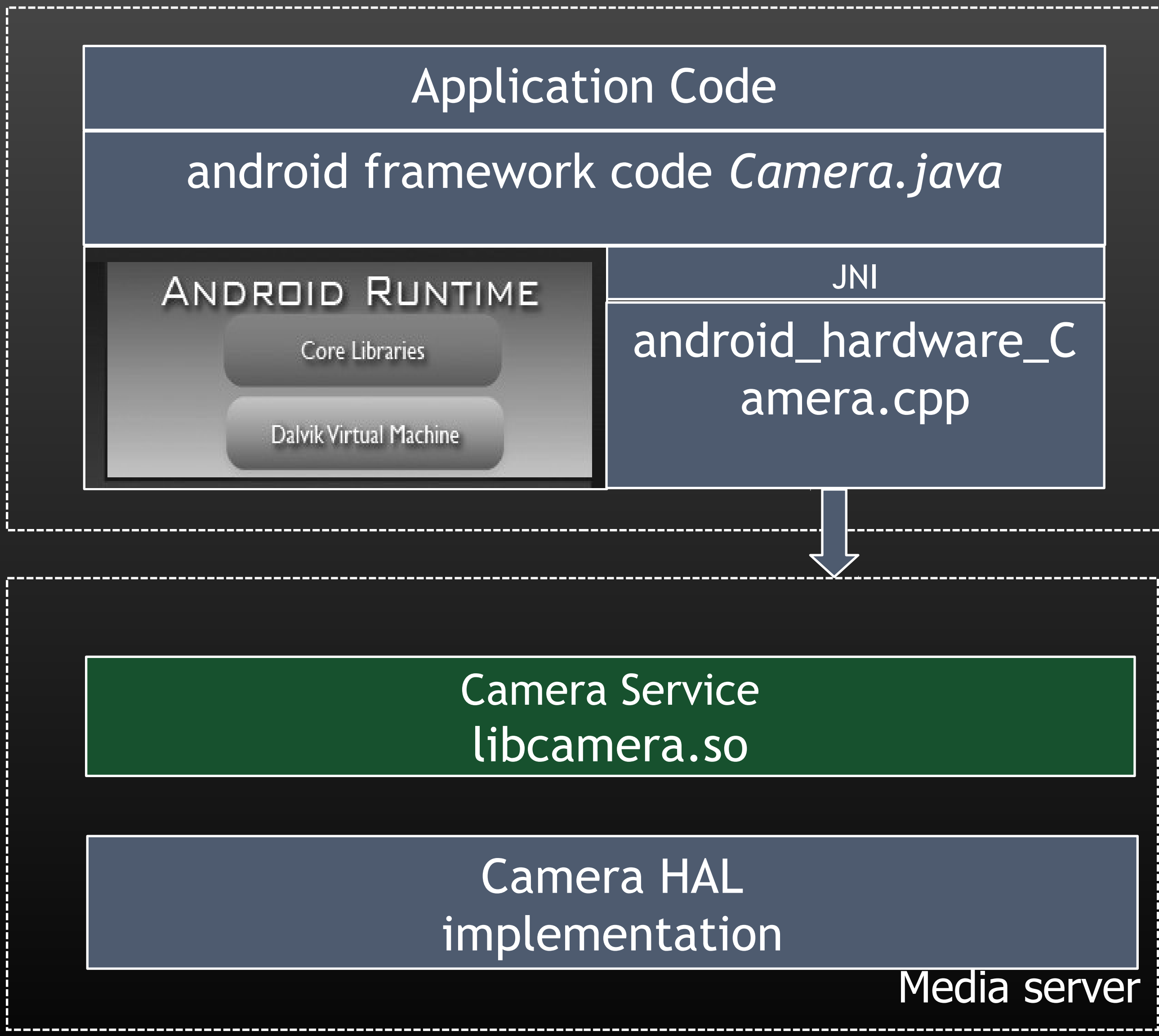
Allocates Memory from the Java memory heap for JPEG images.

| © 2012 Aptina Imaging Corporation

Camera Service

© 2012 Aptina Imaging Corporation

Camera Service



 | © 2012 Aptina Imaging Corporation

Camera Service

Resource Manager for the Camera Hardware Asset

Runs in the media server process

It is a shared library libcameraservice.so

Main Functions:

- Permission check android.permission.CAMERA
- Ensures only one Client connects to a Camera Hardware Object
- Ensures each Process connects to a single Camera Hardware Object
- Redirects callbacks back to the app layer
- Accessed over IBinder Interface
- Number of Cameras Available
- CameraInfo Details

Camera Service (contd.)

Android.mk file

frameworks/base/media/mediaserver/Android.mk

```
LOCAL_SHARED_LIBRARIES := \
    libaudioflinger \
    libcameraservice \
    libmediaplayerservice \
    libutils \
    libbinder
```

Gets instantiated as along with other components of the media server

```
AudioFlinger::instantiate();
MediaPlayerService::instantiate();
CameraService::instantiate();
AudioPolicyService::instantiate();
```


| © 2012 Aptina Imaging Corporation

Interaction with the Media Subsystem

ICameraRecordingProxy and ICameraRecordingProxyListener were introduced in Android 4.0

Allow apps to use the camera subsystem while the MediaRecorder is recording the video frames.

ICameraRecordingProxy is a proxy of Icamera

- startRecording
- stopRecording
- releaseRecordingFrame

ICameraRecordingProxyListener is an interface that allows the recorder to receive video frames during recording.

- dataCallbackTimestamp

© 2012 Aptina Imaging Corporation

Android Open Source Project (AOSP) Structure

Android Framework

Java: frameworks/base/core/java/android/hardware

JNI: frameworks/base/core/jni

Camera Service

frameworks/base/services/camera/libcameraservice/

IBinder Interfaces

frameworks/base/libs/camera/ICamera.h

Camera HAL Interface

frameworks/ base/services/camera/libcameraservice

Camera HAL

hardware/<vendor>/camera (typically)

| © 2012 Aptina Imaging Corporation

Section II

Hardware-Dependent Camera Stack

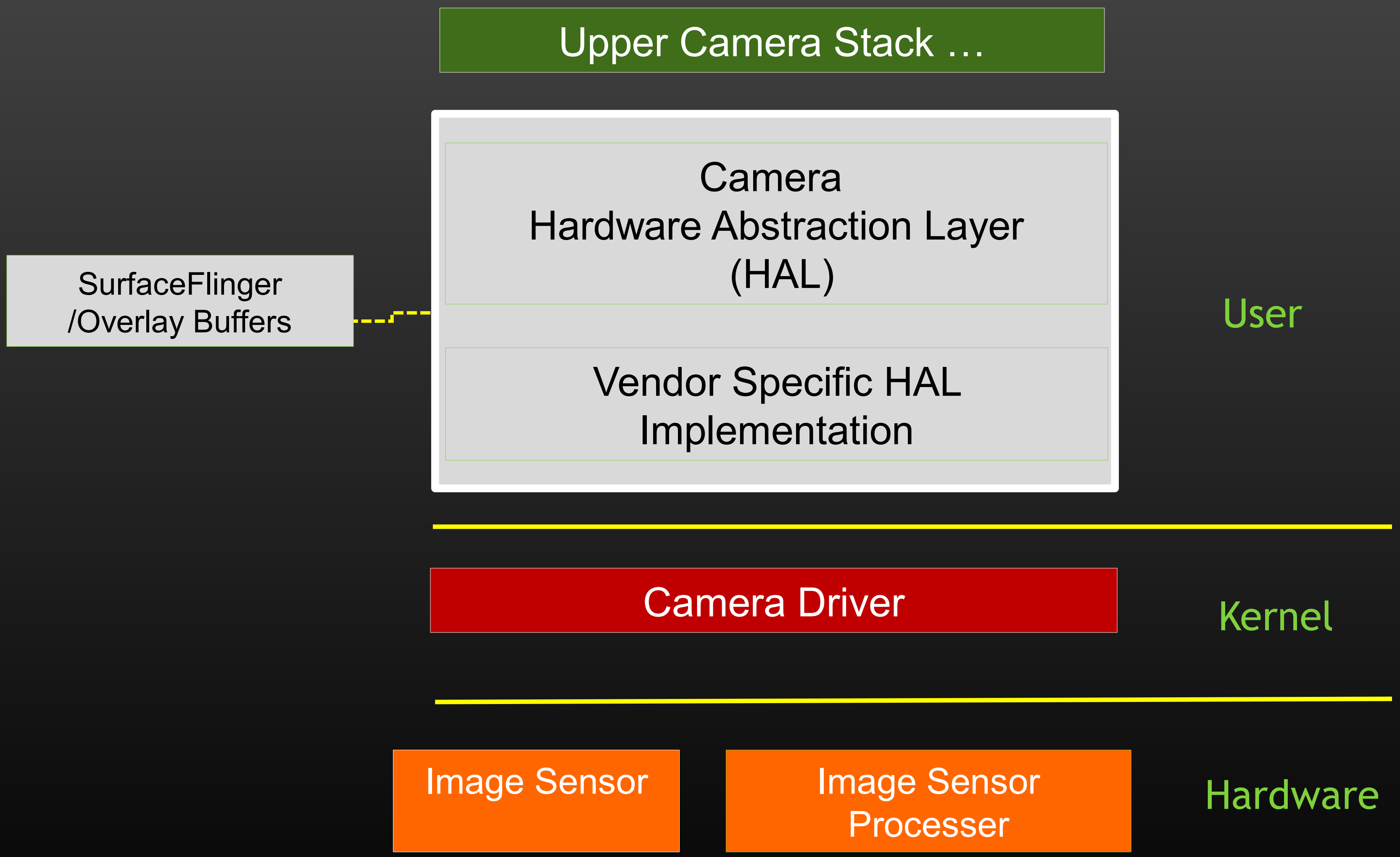
© 2012 Aptina Imaging Corporation

Camera Hardware Abstraction Layer

Review of a Typical Implementation

| © 2012 Aptina Imaging Corporation
Aptina IMAGING

Camera Stack – Camera HAL



© 2012 Aptina Imaging Corporation

Android CameraHAL Library

The Camera Hardware Abstraction Layer (HAL) is a library that is specific to the camera hardware platform

- Written by hardware vendors (Qualcomm, TI, others)

CameraHAL maps Android Camera Service calls to driver functions

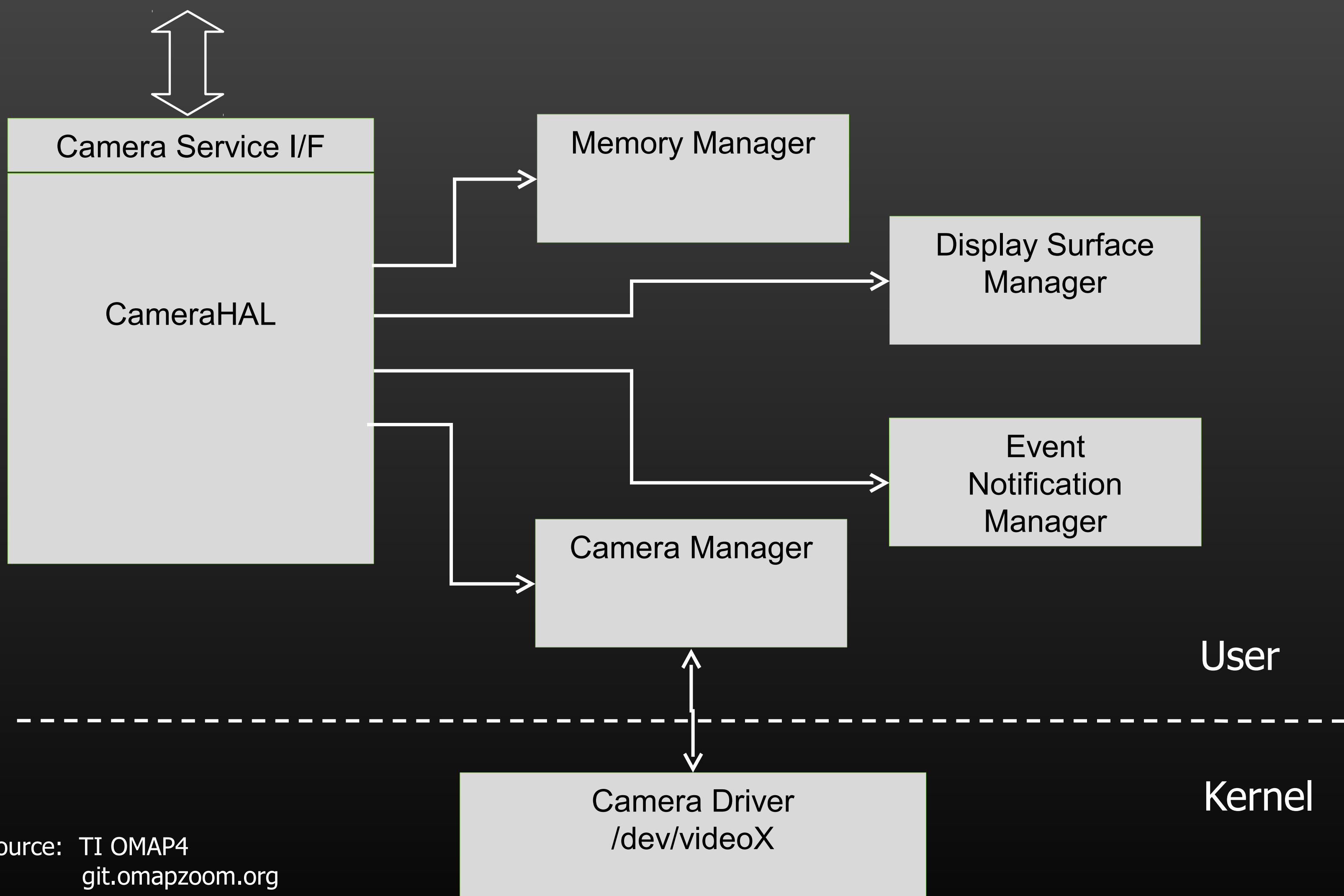
- Android Froyo uses CameraHardwareInterface.h wrapper
- Ice Cream Sandwich (ICS) and above use camera.h

CameraHAL low level interface communicates with the kernel level driver

- It can support interfaces including Video for Linux 2 (V4L2) or OpenMax (OMX)
- Communicates with the driver through file I/O calls (open, close, input/output controls (IOCTL), etc)

| © 2012 Aptina Imaging Corporation

Sample CameraHAL Functional Diagram



Source: TI OMAP4
git.omapzoom.org

© 2012 Aptina Imaging Corporation

CameraHAL Block Diagram Discussion (1)

- Parts of the previous block diagram are hardware vendor specific
  - May be different for each vendor and target platform
- CameraHAL
  - Initialization – initialize the CameraHAL block and the target device driver
  - Camera Services interface – Handle each Camera Service request, dispatch requests to the appropriate functional block
  - Camera State machine – maintain the camera state through different API calls (e.g., preview, capture, recording, focus enable, etc).
- Memory Manager
  - Cameras are memory intensive devices
  - On request, allocate buffers for preview, capture and other functions
- Display Surface Manager
  - Controls preview and video displaying - helps to coordinate with the camera manager block

| © 2012 Aptina Imaging Corporation

CameraHAL Block Diagram Discussion (2)

- Display Surface Manager (cont)
  - Communicates to the display when a frame is ready for preview
  - Signals to the Camera Manager when the image buffer can be re-queued
- Event Notification Manager
  - Supported callbacks include notify, data and timestamp
    - Notify – call on camera error, shutter, focus, zoom events or raw image notify event
    - Timestamp – call on video frame event
    - Data – call on preview, postview, compressed image, and other capture events
  - Call backs types are separated at the Camera Service level
- Camera Manager
  - Handle camera activities
    - Setting parameters
    - Preview and snapshot callback

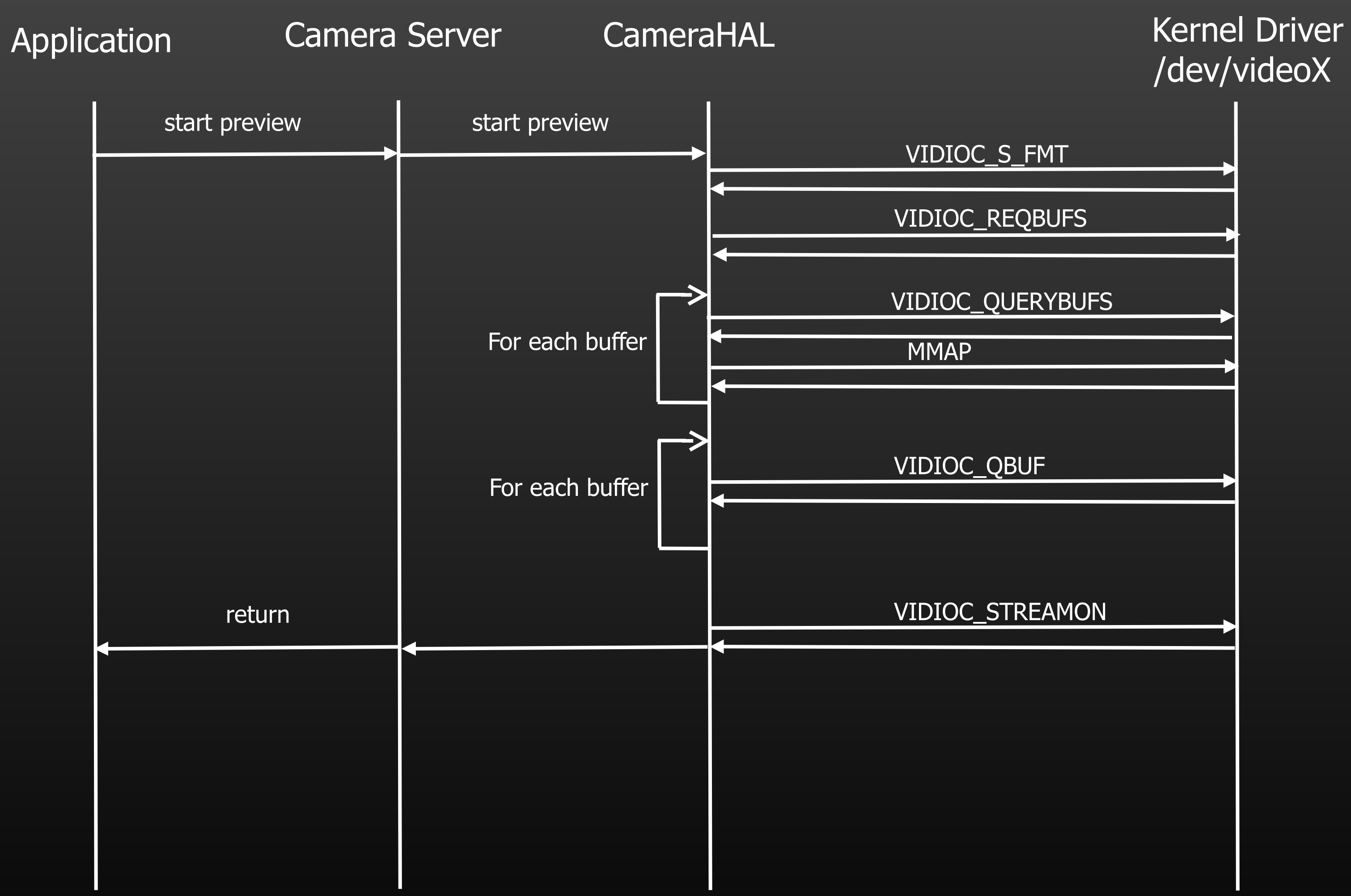
| © 2012 Aptina Imaging Corporation

CameraHAL Preview Discussion

- The following slides discuss the preview use case
- Preview – displaying the camera image on the device display in real time
- The startPreview application call initiates image preview
  - A single application level call results in a chain of CameraHAL and driver events
- Preview continues until the stopPreview() application call
  - During preview, no application interaction unless a preview callback is registered

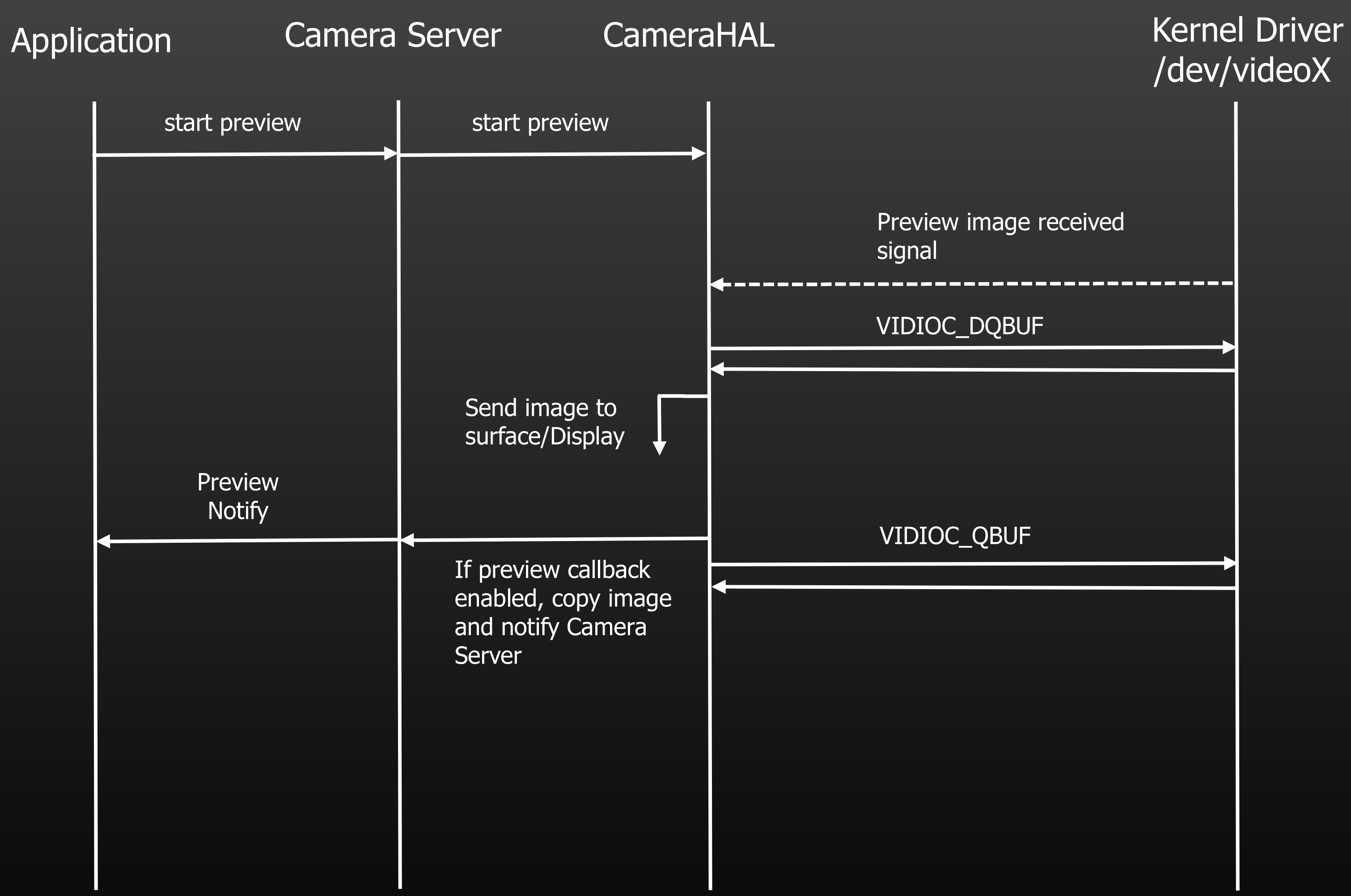
| © 2012 Aptina Imaging Corporation
Aptina IMAGING

Preview Start Up
Sequence Diagram (V4L2)



| © 2012 Aptina Imaging Corporation
Aptina IMAGING

Preview Operation Sequence Diagram (V4L2)



 | © 2012 Aptina Imaging Corporation

Camera Preview Interaction with the Display Subsystem

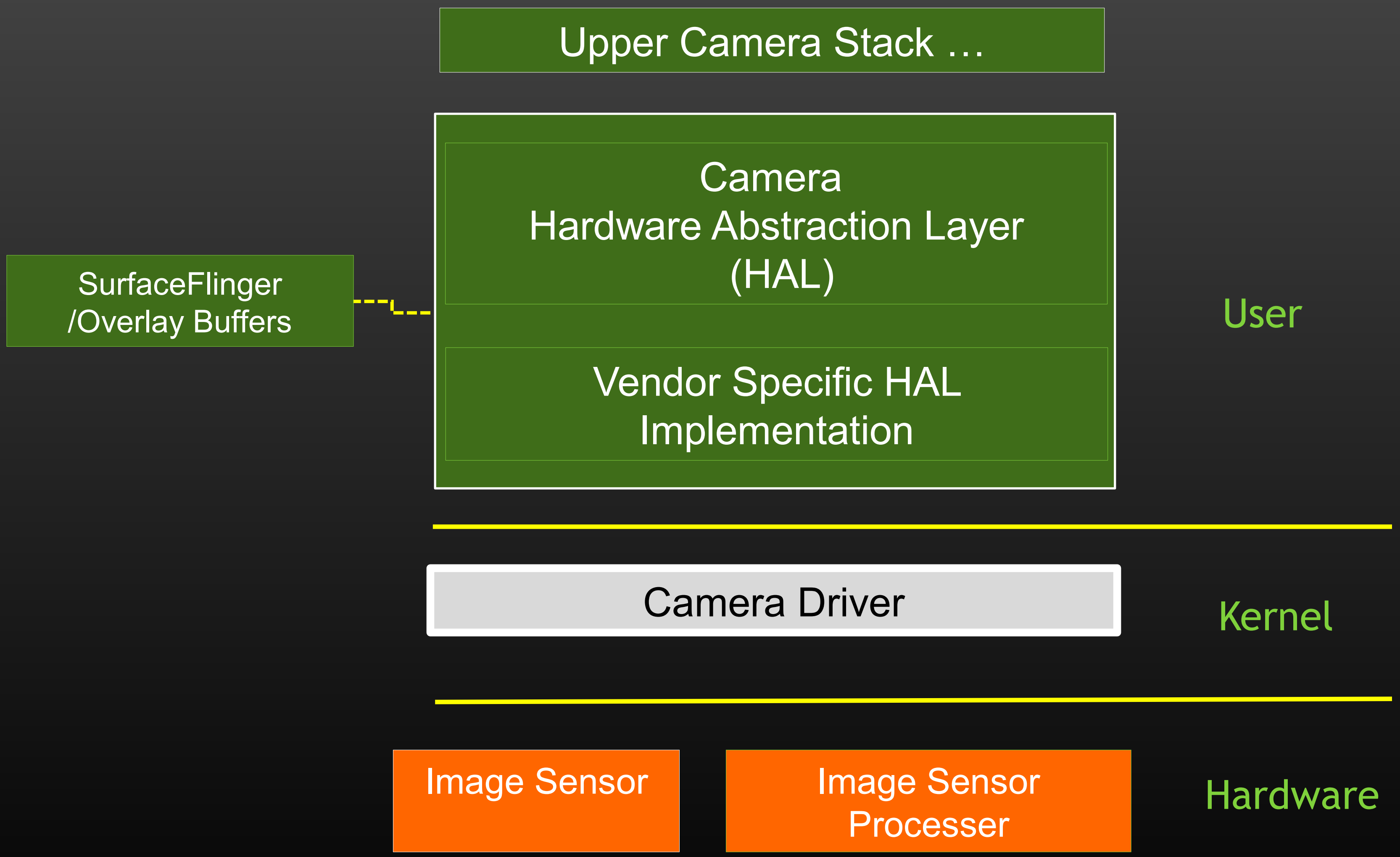
- Matching the timing of 2 events
  - Preview frames arrive asynchronously from the camera
  - The display subsystem refreshes the display at regular intervals
  - Potential mismatch between these 2 system
- Sending the preview image to the display subsystem
  - The preview frame is removed from the V4L2 queue of buffers
  - The frame is sent to the display subsystem
    - The frame memory is shared by the display subsystem
    - Or the frame is copied to a buffer for display subsystem use
  - The preview frame may be copied to a user space buffer if preview callback is enabled
  - The frame is returned to the V4L2 queue of buffers when done

| © 2012 Aptina Imaging Corporation

Camera Device Driver

© 2012 Aptina Imaging Corporation

Camera Stack – Camera Driver



© 2012 Aptina Imaging Corporation

Android Kernel Camera Driver

The kernel driver presents a standard interface for different types of camera hardware

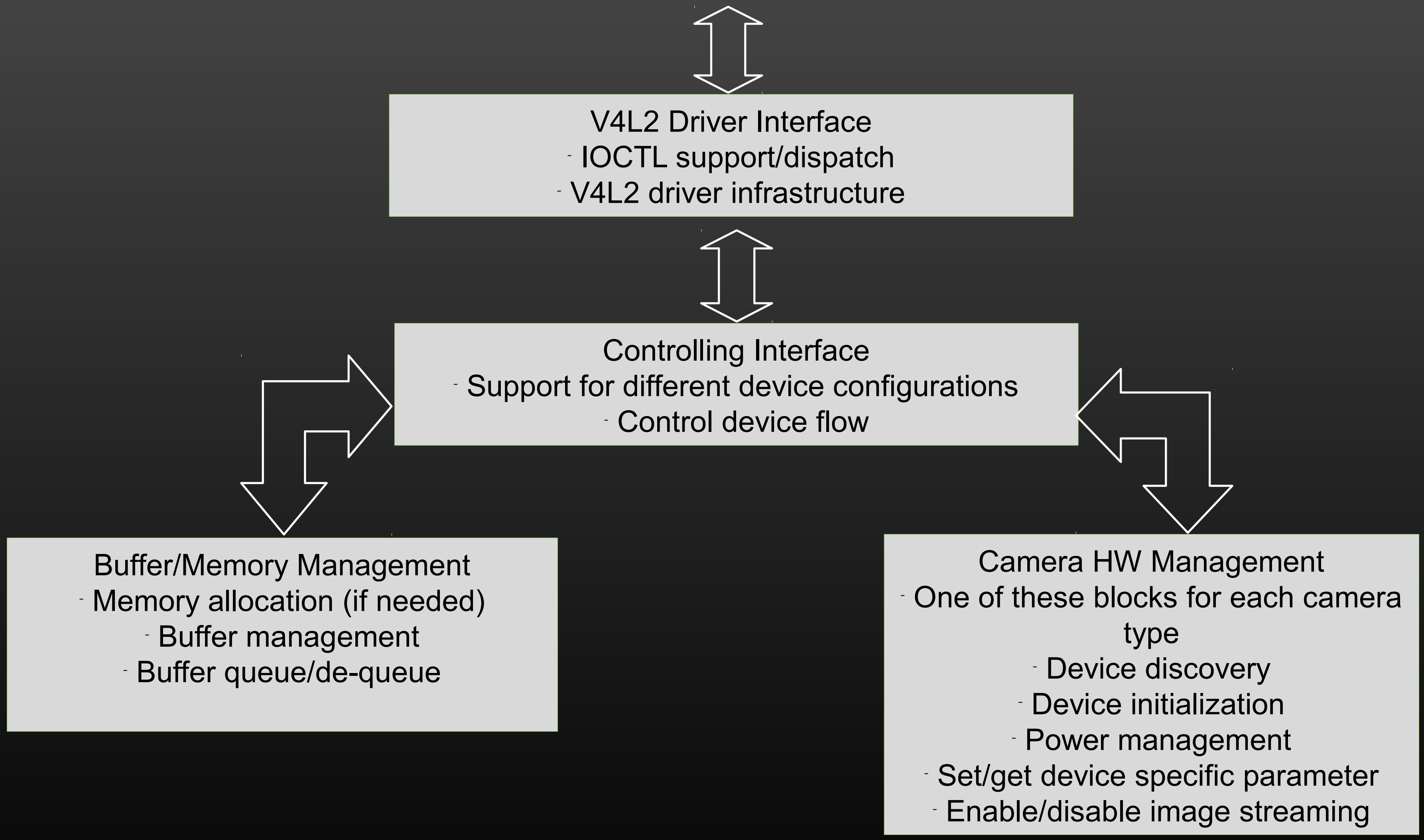
- Camera hardware specific attributes are (usually) handled by the low level kernel driver
- Image Sensor Processor (ISP) vs. smart image sensor - differences are handled at the driver level

For Android, Video for Linux 2 (V4L2) is used in many implementations

- V4L2 has been in existence for many years
- OpenMax (OMX) is also used for a low level driver interface by some vendors.

© 2012 Aptina Imaging Corporation

V4L2 Kernel Driver Block Diagram



| © 2012 Aptina Imaging Corporation

Android Linux Kernel Functionality

- Support for multiple camera types
  - Camera specific code is localized to one file (the subdev device)
  - Compile time option to add other cameras (one driver can support many different camera hardwares)
  - More cameras mean a longer start up time since the driver is searching for each device
- The driver manages the underlying hardware topology (e.g., ISP + sensor, smart sensor)
- For two or more cameras, the V4L2 driver creates additional device nodes
  - Devices show up as /dev/video0 (primary), /dev/video1 (secondary), …

| © 2012 Aptina Imaging Corporation

V4L2 Kernel Driver Resources

Memory

- Memory can be either driver-allocated or user-provided
- Moving image from the camera to memory should be done through hardware DMA (Direct Memory Access)
  - Hardware memory management required to avoid contiguous memory requirement

Interrupts

- Camera ports support for interrupts on events such as frame start, finish, focus events, etc.

Camera Control: I2C/SPI

- Communications with the camera is usually done with I2C by either writing or reading sensor registers.
- I2C is somewhat slow, this limits the number of register accesses during a frame. SPI (Serial Peripheral Interface) is an alternative to I2C.

Control Signals/Power/GPIO

- All controlled by the low level driver

Power

- Sensor power management is critical to embedded device operation
- Save power by disabling the sensor and sensor processor when not used

| © 2012 Aptina Imaging Corporation

V4L2 Driver Buffer Management

One or more buffers are supported
User buffers or kernel-allocated buffers are supported
Buffers are treated the same for preview, capture, video (output resolution does not matter)
Buffers are queued to a circular list
Buffer filling starts when the V4L2 Stream_On command is executed
Once filled, the CameraHAL de-queues a buffer, processes the buffer, then re-queues the buffer
The Stream_Off command causes all buffer to be released

| © 2012 Aptina Imaging Corporation

Typical Android V4L2 Start up Sequence

The V4L2 call for the preview are given below

V4L2 Call	Title	Comments
V4L2_Open()	Open a V4L2 Device	
VIDIOC_S_FMT	IOCTL: Set format	Set both resolution and output pixel format
VIDIOC_G_PARM	IOCTL: Get camera parameter	Get the camera frame rate
VIDIOC_S_PARM	IOCTL: Set camera parameter	Set the camera frame rate
VIDIOC_CROPCAP	IOCTL: Get the camera cropping capabilities	Get the current crop rectangle
VIDIOC_S_CROP	IOCTL: Set the cropping rectangle	Set the desired cropping rectangle
VIDIOC_REQBUFS	IOCTL: Request camera buffers	Request buffer support from the driver (user vs. kernel)
Loop: VIDIOC_QUERYBUF	IOCTL: Return buffer address information	Used for mapping buffers to user space
V4L2_MMAP	Memory map buffers to user space	Make buffers visible to user applications

© 2012 Aptina Imaging Corporation

Typical V4L2 Start up Sequence

V4L2 driver start up sequence (cont)

Typ

V4L2 Call	Title	Comments
Loop: VIDIOC_QBUF	IOCTL: Add buffer to queue	Queue of buffers the kernel manages
VIDIOC_STREAM_ON	IOCTL: Start image streaming	Camera/Driver starts filling the queued buffers

V4L2 Call	Title	Comments
VIDIOC_STREAM_OFF	IOCTL: Stop image streaming	Stop camera/driver streaming
V4L2_Close()	Close the camera device	Disable camera operations, free resources

© 2012 Aptina Imaging Corporation

Aptina IMAGING

V4L2 Driver Directions

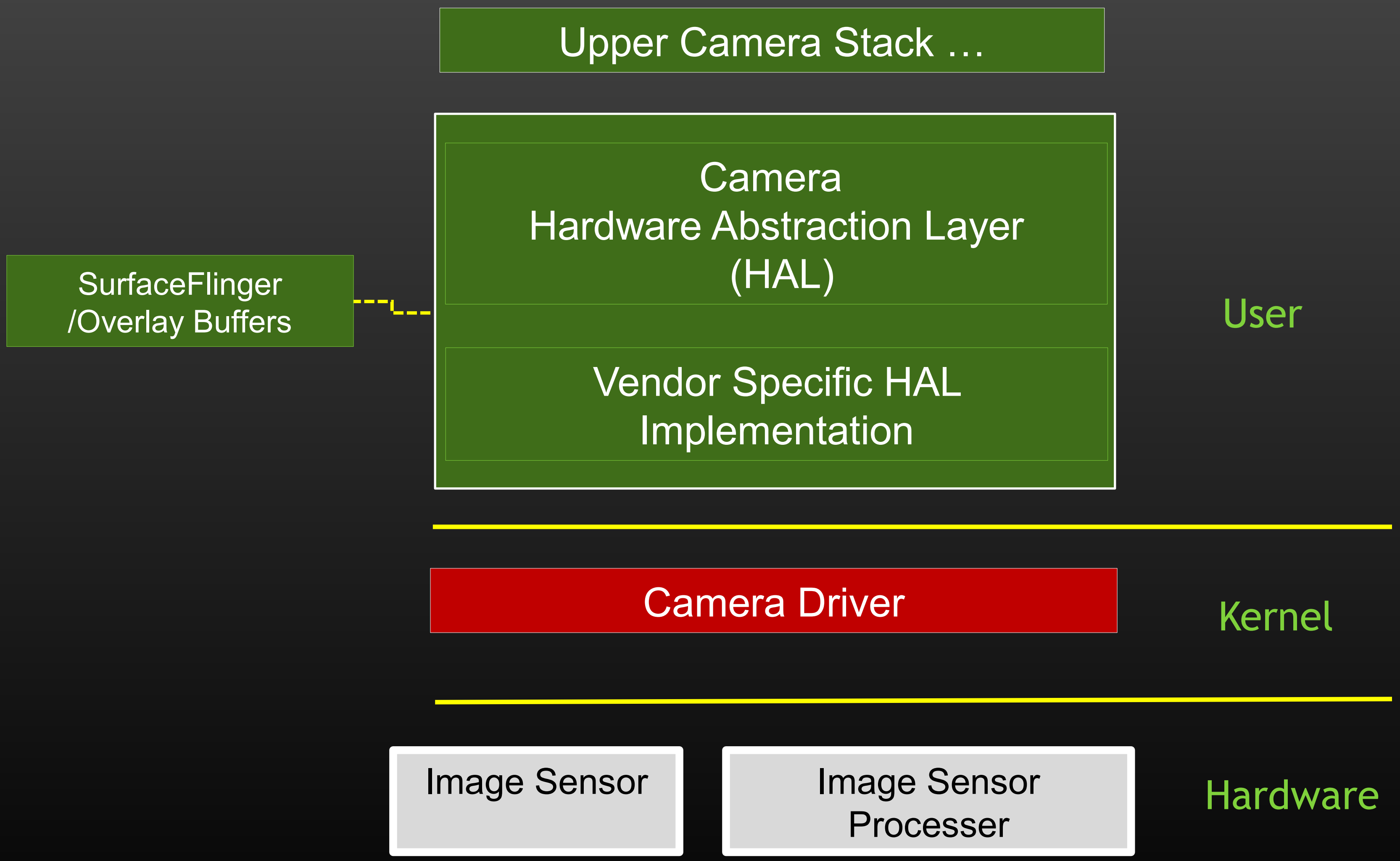
- Other Topics
  - V4L2 Media Controller Architecture
    - Exposing the hardware image processor to the calling application
    - Allows for greater programmer control
    - Supported only on open source architectures
  - Proprietary ISP software moves to user space
    - Many ISP providers wish to hide their hardware
    - Moving ISP code to user space handles this (avoid kernel open source issues)
- Driver source code location:
  - {kernel sources}/drivers/media/video

© 2012 Aptina Imaging Corporation

Camera Hardware Overview

| © 2012 Aptina Imaging Corporation

Camera Stack – Camera Hardware



© 2012 Aptina Imaging Corporation

Camera Hardware Introduction

- Types of Sensor Hardware
  - Raw or Bayer Sensor
    - Outputs a Bayer (unprocessed) image
    - Used with internal or external Image Sensor Processor (ISP)
      - Internal ISP – System Processor and ISP bundled together
      - External ISP – External companion chip
    - Controls include exposure time and analog/digital gains
  - Smart or System On a Chip (SOC) Sensor
    - ISP built into the sensor
    - Outputs processed YUV/RGB/Other formats
    - Controls include exposure, white balance, gamma correction, and many others

G1	R	G1	R
B	G2	B	G2

Bayer P

Cb_i	Y_i	Cr_i	Y_{i+1}

YCrCb Pattern

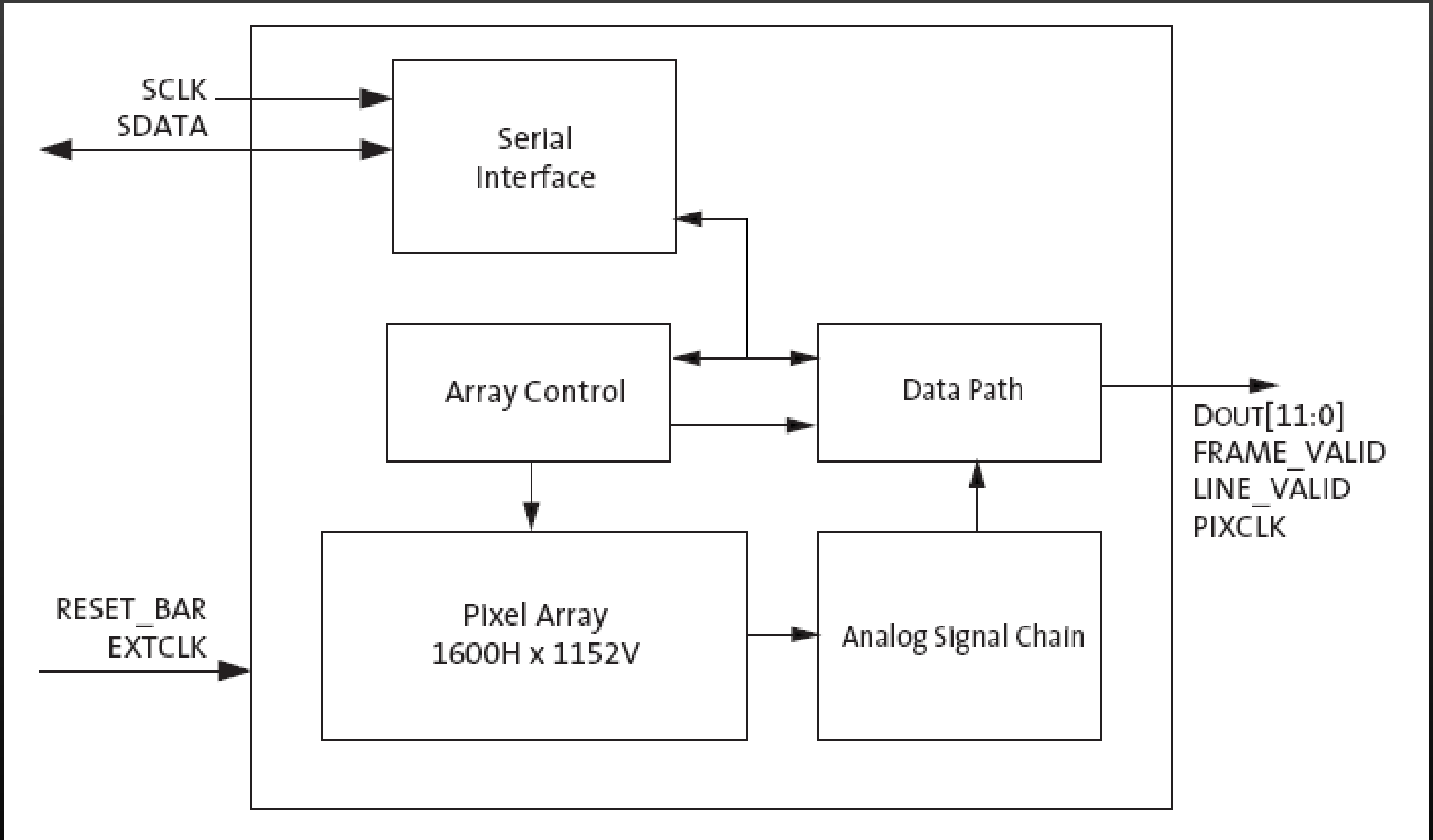
$R_7R_6R_5R_4R_3G_7G_6G_5$	$G_4G_3G_2B_7B_6B_5B_4B_3$

RGB565 Pattern

© 2012 Aptina Imaging Corporation

Bayer Sensor Block Diagram

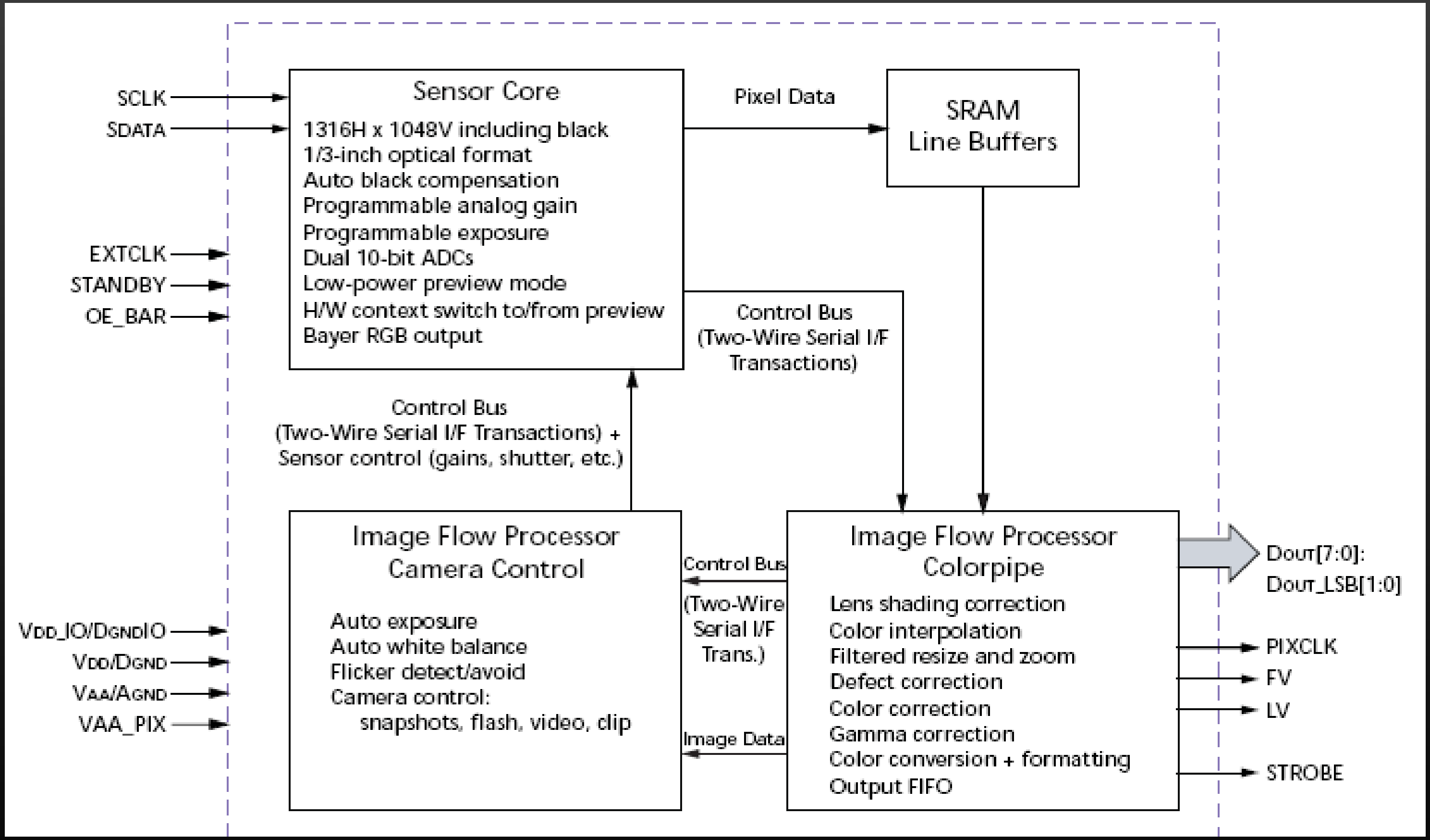
Example - MT9M032 – 1.6MP Image Sensor



© 2012 Aptina Imaging Corporation

SOC Sensor Block Diagram

Example - MT9M131 – 1.3 MP Image Sensor



© 2012 Aptina Imaging Corporation

Camera Hardware Inputs/Outputs

Controlled by the sensor driver

Inputs:

- Power/Ground (analog, digital power/grounds)
- Control Signals
  - Reset – reset the camera to a default state
  - Standby – place the camera in low power standby mode
  - GPIO, others – control camera peripherals such as autofocus, flash, etc.
- Clock In (system clock in)
- Register control through I2C, SPI, or others

Output

- Data Output
  - Parallel (8, 10, 12, 14 bits)
  - Serial (MIPI)
- Control Signals (frame/line valid)
- Clock Out (pixel clock out)

| © 2012 Aptina Imaging Corporation

A Peek into the Future

© 2012 Aptina Imaging Corporation

Camera Application Trends

Android Applications – memory limitation 16MB ~ 24MB

- Higher pixel sizes and Bursty modes put a strain on the system

Computer Vision Applications go mainstream

- APIs on Object Tracking, Gesture Recognition become more common place

Computation Photography application

- Developers get fine grained control of flash and camera

| © 2012 Aptina Imaging Corporation

Camera Hardware Trends

- Back Side Illumination(BSI) vs. Front Side Illumination(FSI)
  - BSI can add up to 30% more light gathering capability
- Smaller Pixels
  - Constant push to reduce pixel and sensor package sizes
- Faster data output rates, higher clock speeds
  - 1080p30, 1080p60
  - Serial data interfaces enable increased sensor output speeds
- High Dynamic Range
  - Ability to capture larger exposure range
- 3D Imaging
  - Use of 2 cameras to generate a 3D image

| © 2012 Aptina Imaging Corporation

Q&A

© 2012 Aptina Imaging Corporation

References

http://developer.android.com/

http://www.codeaurora.org

http://omappedia.org

http://source.android.com

© 2012 Aptina Imaging Corporation

Aptina™
IMAGING