CAS 765 Fall'15
Mobile Computing and Wireless Networking

Rong Zheng

Android Sensing Subsystem

Qiang Xu

What is a Sensor?

- A converter that measures a physical quantity and coverts it into a signal which can be read by an observer or by an instrument.
  - Microphone, number keys, accelerometer, gyroscope
  - **Instead of carrying around 10 separate devices, now you just need 1**

Sensor Categories

- Android sensors as separated into one of three broad categories:
  - Motion sensors – measure force and rotation
    - e.g. acceleration
  - Environmental sensors – measure environmental parameters
    - e.g. illumination, air temperature and pressure
  - Position sensors – measure physical positioning of the device
    - e.g. GPS

Hardware vs. Software

- Although it might be natural to think of all sensors as hardware, that is NOT the case
- Sensor of Android is a component that provides information about the outside world
- Some "sensors" are actually software components that fuse data from hardware sensors generate different kinds of data
- These are called "compound sensors" or "virtual sensors"

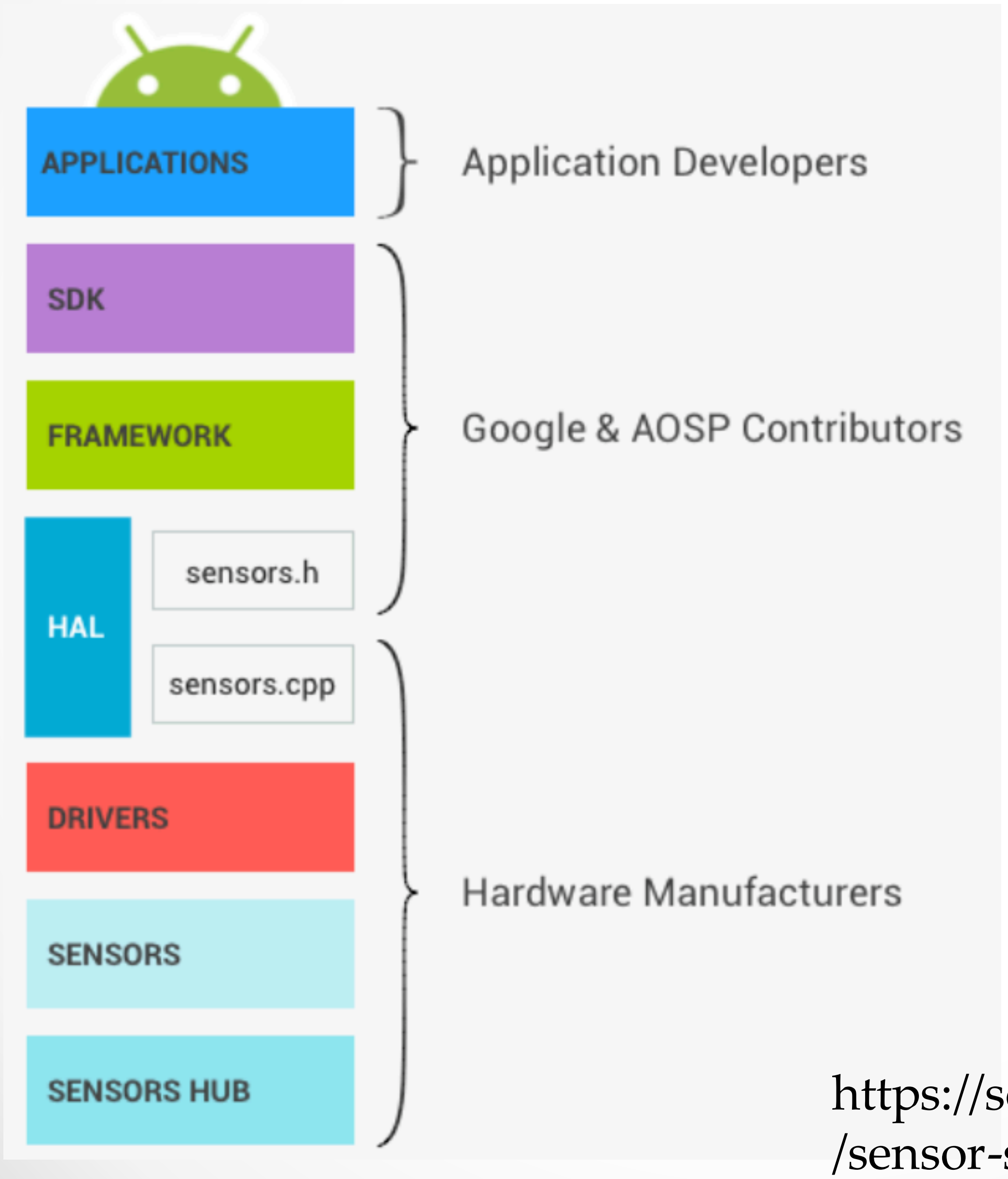
Sensors? what sensors?

- Sensors are NOT guaranteed!
- There is no Android Specs for required sensors
- You need to check for the sensors first
- You can activate more than one sensors in one app
- Different device has different capability and driver

Android Sensors

- Common
  - MIC
  - Camera
  - Temperature
  - Location (GPS or Network)
  - Orientation
  - Accelerometer
  - Proximity
  - Pressure
  - Light
- Special (Introduced since Android 4.4, only avail on Nexus 5 now)
  - Sensor batching
  - Step Detector and Step Counter

Sensor Stack

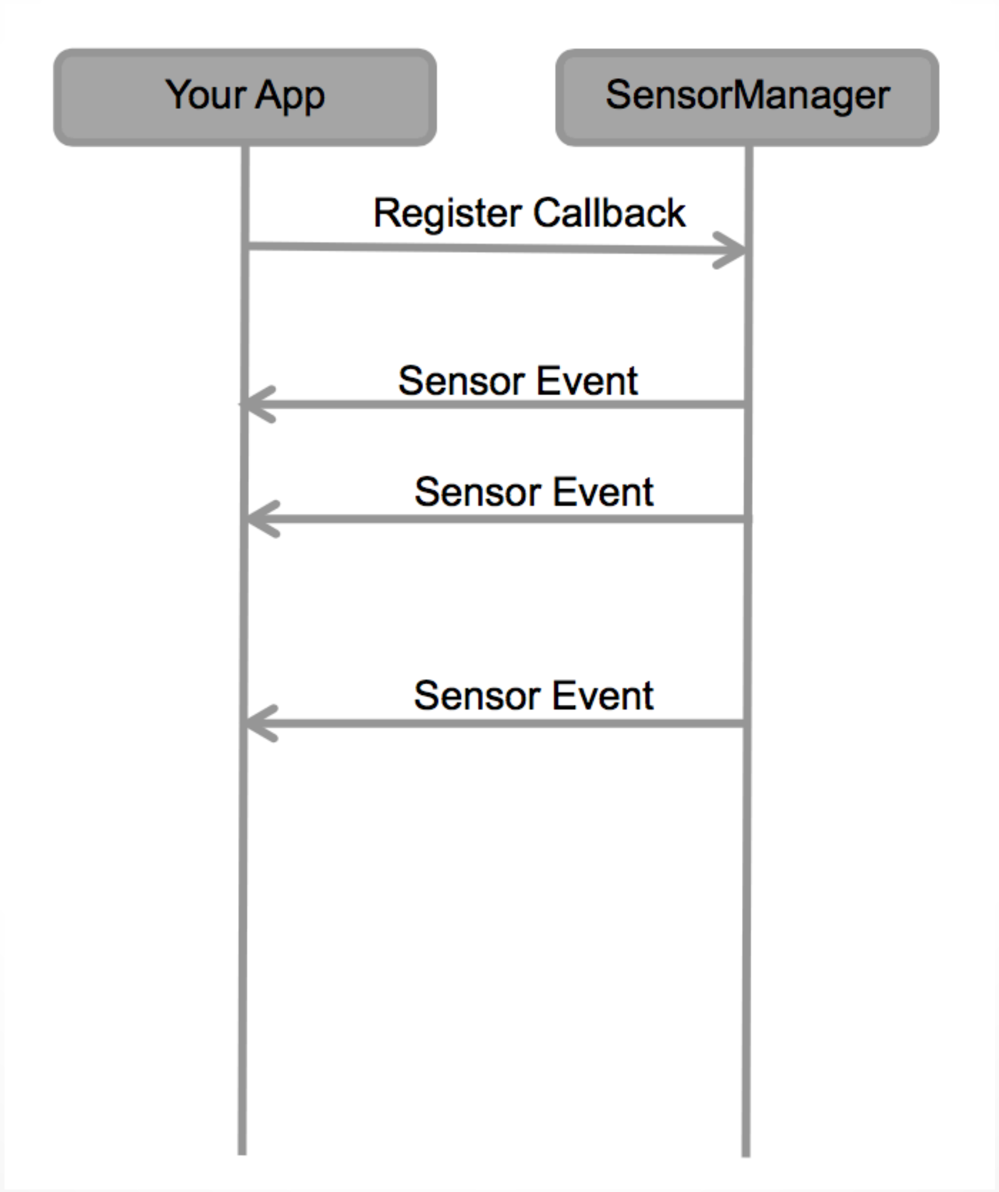


https://source.android.com/devices/sensors/sensor-stack.html

Android.hardware Package

- Camera, Camera.Area, Camera.CameraInfo, Camera.Face and other Camera.XX
  - Image related API, deprecated in API level 21, refer to android.hardware.camera2 for new apps
- ConsumerIrManager
  - operates consumer infrared on the device
- ConsumerIrManager.CarrierFrequencyRange
  - Represents a range of frequencies which the infrared transmitter can transmit
- GeomagneticField
  - Estimates magnetic filed at a given point.
- Sensor
  - Represents a sensor
- SensorEvent
  - Represents a sensor event (a sensor measurement)
- SensorManager
  - Let apps access sensors
- TriggerEvent
  - Represents a Trigger sensor-the event associated with a Trigger sensor
- TriggerEventListener
  - Listener used to handle Trigger Sensors

How to get data from sensors?



Class SensorManager

- An Android system service that gives an app access to hardware sensors
  - it is a service, so it is running in the backend as a daemon process
- Allow apps to register or unregister for sensor events
  - once registered, an app will receive sensor data from hardware
- Also, provides methods that process sensor data
  - e.g. SensorManger.getOrientation()
- Methods
  - **Sensor getDefaultSensor(int type):** get the default sensor for a given type
  - **List<Sensor> getSensorList(int type):** get the list of avail sensors of certain type
  - **boolean registerListener(SensorEventListener listener, Sensor sensor, int rate)**
  - **void unregisterListener(SensorEventListener listener, Sensor sensor)**
-
```
String service_name = Context.SENSOR_SERVICE;
SensorManager sensorManager =
(SensorManager)getSystemService(service_name)
```

Class Sensor

- Represents a sensor on a device
- Exposes information about the sensor
    - Maximum range
    - Minimum delay
    - Name
    - Power
    - Resolution
    - Type
    - Vendor
    - Version

Sensor.TYPE_XX

- TYPE_ACCELEROMETER
- TYPE_AMBIENT_TEMPERATURE
- **TYPE_GAME_ROTATION_VECTOR**
- **TYPE_GEOMAGNETIC_ROTATION_VECTOR**
- TYPE_GRAVITY
- TYPE_GYROSCOPE
- TYPE_GYROSCOPE_UNCALIBRATED
- **TYPE_HEART_RATE**
- TYPE_LIGHT
- **TYPE_LINEAR_ACCELERATION**
- TYPE_MAGNETIC_FIELD
- TYPE_MAGNETIC_FIELD_UNCALIBRATED
- **TYPE_ORIENTATION**
- TYPE_PRESSURE
- TYPE_PROXIMITY
- TYPE_RELATIVE_HUMIDITY
- **TYPE_ROTATION_VECTOR**
- **TYPE_SIGNIFICANT_MOTION**
- **TYPE_STEP_COUNTER**
- **TYPE_STEP_DETECTOR**

Note:

- Availability depends on hardware
- May deprecate or only introduced in some API levels

Interface SensorEventListener

- An interface that provides the callbacks to alert an app to **sensor events**
  - **onSensorChanged(SensorEvent event)** : Monitor sensor changes
  - **onAccuracyChanged(Sensor sensor, int accuracy)** : React to a change in Sensor Accuracy

- App need to register an concrete class that implements SensorEventListener with SensorManager

```
final SensorEventListener mySensorEventListener = new SensorEventListener() {
        public void onSensorChanged(SensorEvent sensorEvent) {
                // TODO Monitor Sensor changes.
        }
        public void onAccuracyChanged(Sensor sensor, int accuracy) {
                // TODO React to a change in Sensor accuracy.
} }
```

Register a sensor listener

```
// Usually in onResume
Sensor sensor =
sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
sensorManager.registerListener(mySensorEventListener, sensor,
SensorManager.SENSOR_DELAY_NORMAL);
// Usually in onPause
sensorManager.unregisterListener(mySensorEventListener);
```

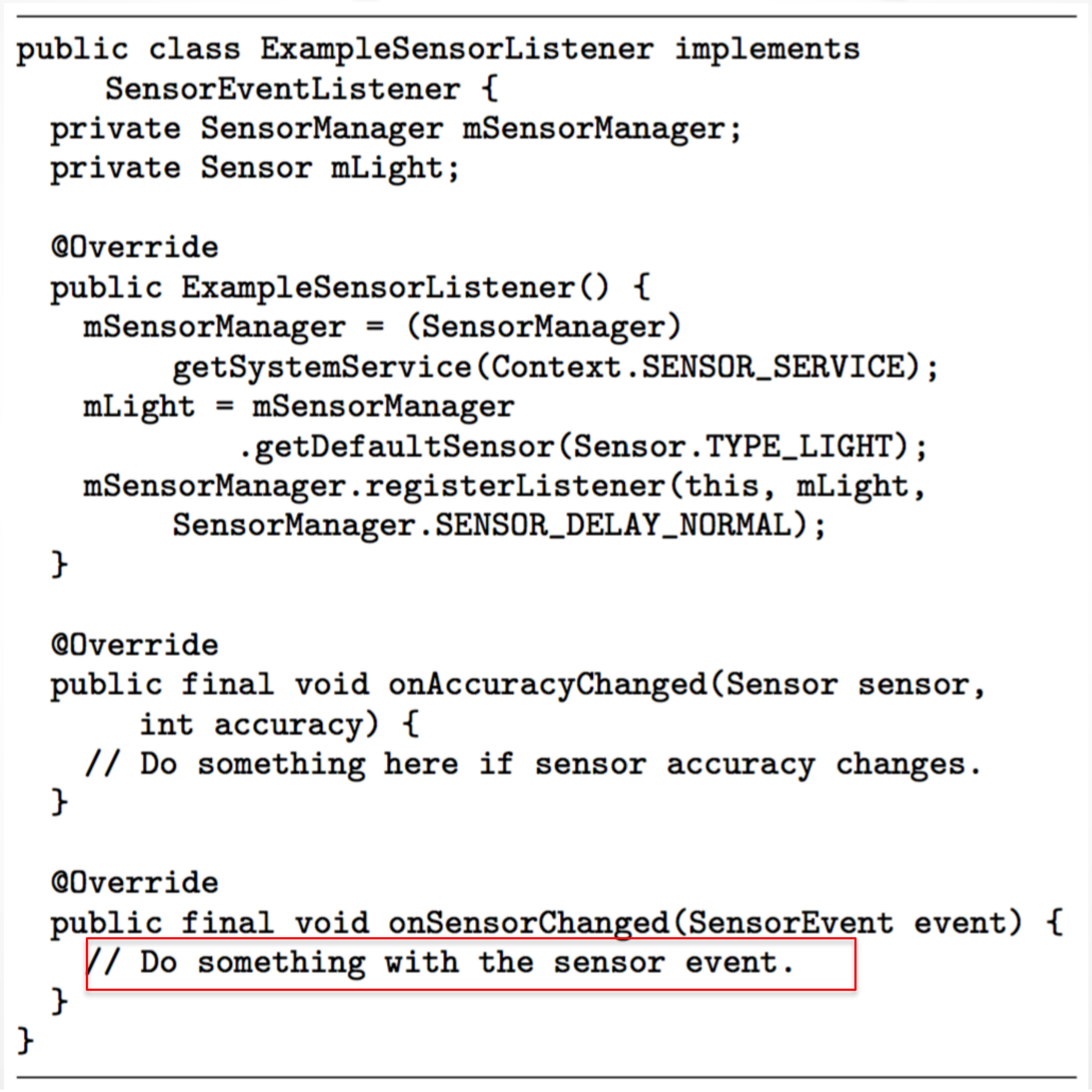
- Update Rate
  - Four options (0, 20, 67, and 200 milliseconds)
    - SENSOR_DELAY_FASTEST
    - SENSOR_DELAY_GAME
    - SENSOR_DELAY_UI (Suitable for usual user interface functions, like rotating the screen orientation.)
    - SENSOR_DELAY_NORMAL (The default value)
  - **Only intended to be hints to the system, as events may be received faster or slower than the specified delay**

Class SensorEvent

- SensorEvent parameter in onSensorChanged method includes four properties:
  - sensor: The sensor that triggered the event.
  - accuracy: The accuracy of the Sensor when the event occurred.
  - **values**: A float array that contains the new value(s) detected.
    - The sensor measurements
  - **timestamp**: The time in nanosecond at which the event occurred.
    - VERY important.
    - The data values are not necessarily evenly spaced in time, this property allows you to access the timestamp associated with the data (which is held in the SensorEvent.values field) in **nanoseconds**.

SENSOR-TYPE	VALUE COUNT	VALUE COMPOSITION	COMMENTARY
`TYPE_ACCELEROMETER`	3	value[0] : Lateral value[1] : Longitudinal value[2] : Vertical	Acceleration along three axes in m/s^2. The Sensor Manager includes a set of gravity constants of the form `SensorManager.GRAVITY_*`
`TYPE_GYROSCOPE`	3	value[0] : Azimuth value[1] : Pitch value[2] : Roll	Device orientation in degrees along three axes.
`TYPE_ LIGHT`	1	value[0] : Illumination	Measured in lux. The Sensor Manager includes a set of constants representing different standard illuminations of the form `SensorManager.LIGHT_*`
`TYPE_MAGNETIC_FIELD`	3	value[0] : Lateral value[1] : Longitudinal value[2] : Vertical	Ambient magnetic field measured in microteslas (µT).
`TYPE_ORIENTATION`	3	value[0] : Azimuth value[1] : Roll value[2] : Pitch	Device orientation in degrees along three axes.
`TYPE_PRESSURE`	1	value[0] : Pressure	Measured in kilopascals (KP).
`TYPE_PROXIMITY`	1	value[0] : Distance	Measured in meters.
`TYPE_TEMPERATURE`	1	value[0] : Temperature	Measured in degrees Celsius.

A complete example



```
public class ExampleSensorListener implements
     SensorEventListener {
  private SensorManager mSensorManager;
  private Sensor mLight;

  @Override
  public ExampleSensorListener() {
    mSensorManager = (SensorManager)
         getSystemService(Context.SENSOR_SERVICE);
    mLight = mSensorManager
             .getDefaultSensor(Sensor.TYPE_LIGHT);
    mSensorManager.registerListener(this, mLight,
         SensorManager.SENSOR_DELAY_NORMAL);
  }

  @Override
  public final void onAccuracyChanged(Sensor sensor,
       int accuracy) {
    // Do something here if sensor accuracy changes.
  }

  @Override
  public final void onSensorChanged(SensorEvent event) {
    // Do something with the sensor event.
  }
}
```

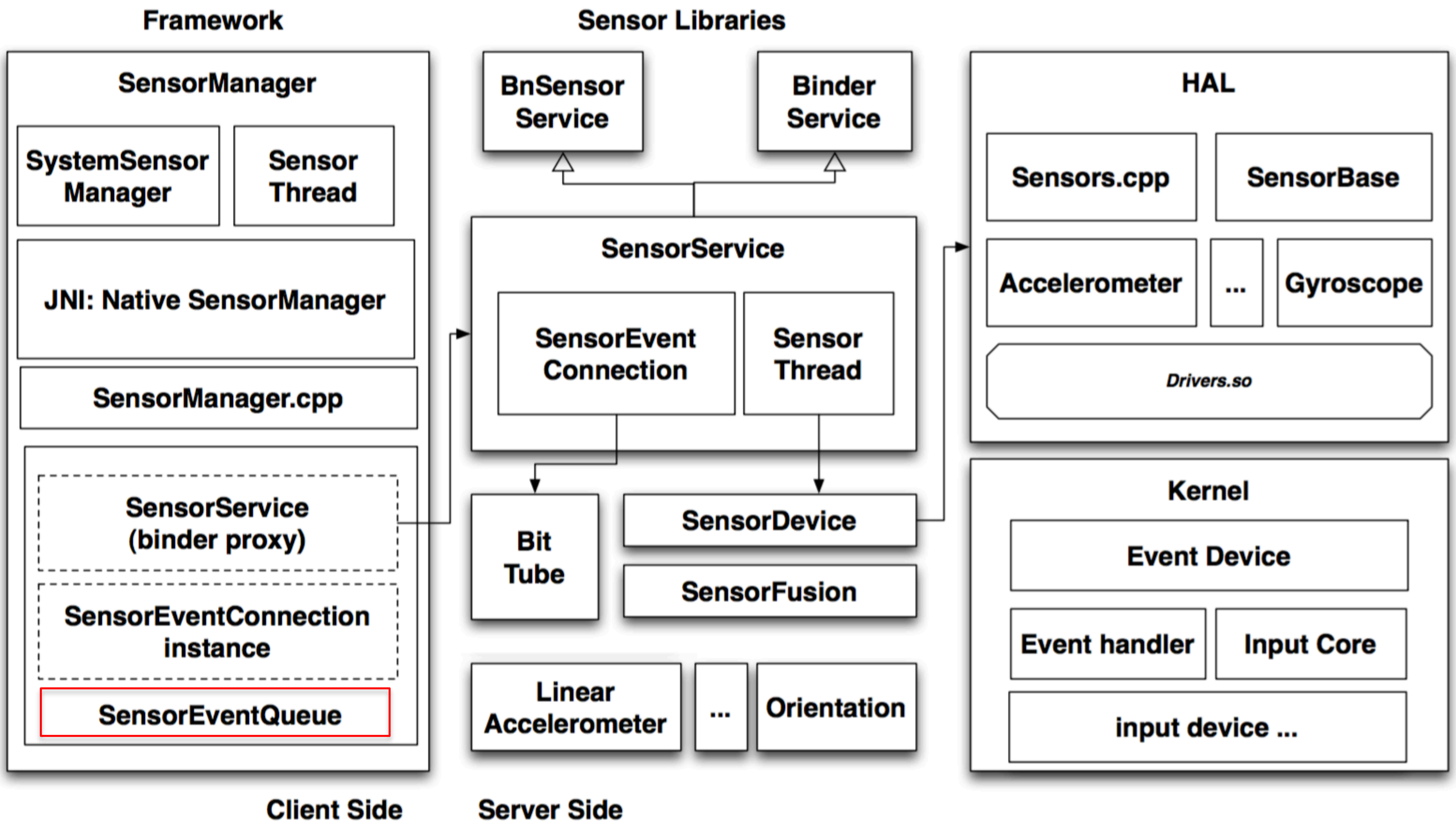
Sensor Rate configuration: A two-edged sword

- Set by *SensorManager.registerListener(SensorEventListener listener, Sensor sensor,* ***int rate****)*
  - *Just a hint, no guarantee*
- Pros
  - The timing of the sensor events is optimized for the particular device
  - Allow sensor polling to be device-agnostic and future proofed
- Cons
  - Unevenly spaced sensor data introduce inaccuracy
  - Difficult for data processing in application

Why sensor data is unevenly spaced?

Real-Time Sensing on Android, *Yin Yan, Shaun Cosgrove, Ethan Blanton, Steven Y. Ko, Lukasz Ziarek,* Proceedings of the 12th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES), 2014

Android sensor architecture



Four Layers

- Kernel
  - Bottom most layer
  - Use Linux as the base kernel and most of the mechanisms for sensor support come directly from Linux
  - Input devices (raw sensor) capture physical inputs and produce input events
  - Input events will be dispatched to any subscribed handlers, **which in turn available**
- HAL (Hardware Abstraction Layer)
  - User-space layer that interfaces with the kernel
  - **Polls the input events from kernel**
  - Provide an unifying hardware interfaces for other user-space processes
  - All the vendor-specific details

Four Layers (cont'd)

- SensorService
  - Part of a system process that starts from boot time
  - Re-format raw hardware sensor data using application-friendly data structure
  - Fuse reading from multiple hardware sensors to generate software sensor data
  - **To accomplish this, it needs to poll each sensor through HAL**
- SensorManager
  - An Android Library linked to each app at run time
  - Provides registration and deregistration calls for app-implemented event handlers
  - **Maintains a SensorEventQueue that holds sensor data pulled from the SensorService**

Two Reasons

Current Android sensing architecture does NOT provide predictable sensing

- Two primary reasons:
  - Android doesn't have any priority support in sensor data delivery. All data delivery follows a **single path** from kernel to apps (SensorEventQueue)
  - The amount of time it takes to deliver sensor data is unpredictable. Current Android sensor architecture relies heavily on polling and buffering to deliver sensor data. This happens at all layers.

Potential Remedies

- While using sensors, turn off other background processes
- Sensor Events are usually received faster if the hardware and garbage collection can keep up.
- Do not set all sensor rates to SENSOR_DELAY_FASTEST
- Unregister unnecessary sensors
- Use other Real-Time Android (e.g. RTDroid)
- Tackle inside application side
  - Interpolation makes sense (think about onSensorChange())
  - Dedicated mechanism to handle unevenly spaced measurments

Further Readings

- **Real-Time Sensing on Android,** *Yin Yan, Shaun Cosgrove, Ethan Blanton, Steven Y. Ko, Lukasz Ziarek,* Proceedings of the 12th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES), 2014
- http://developer.android.com/guide/topics/sensors/sensors_overview.html