

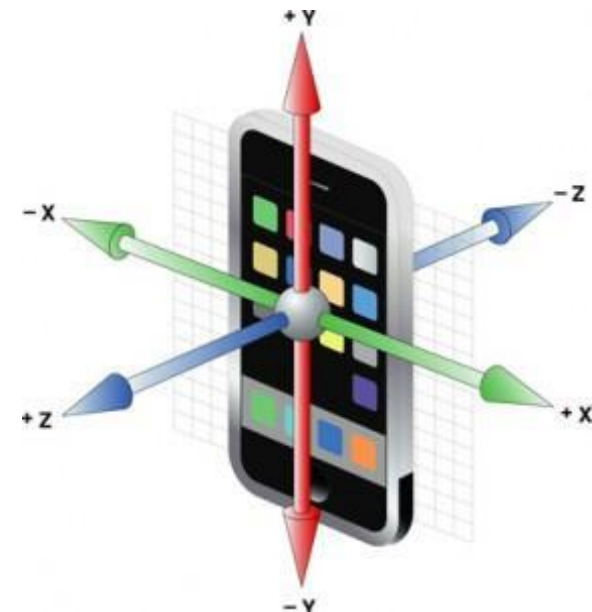
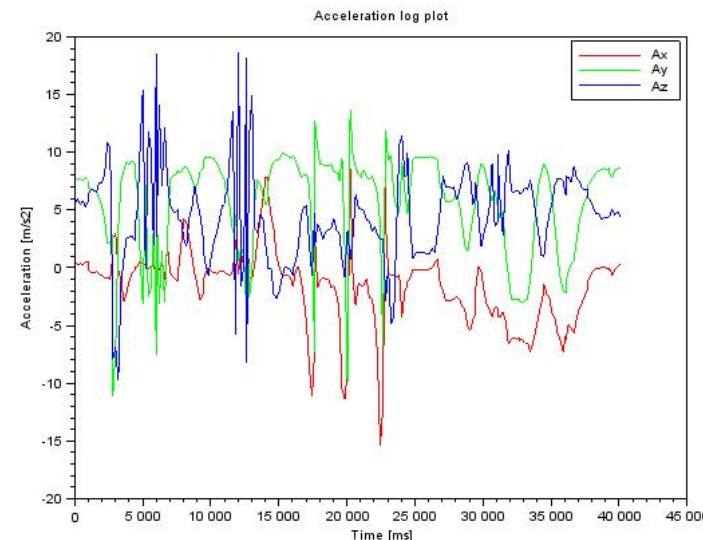
CSE 162 Mobile Computing

Mobile Sensing

Hua Huang

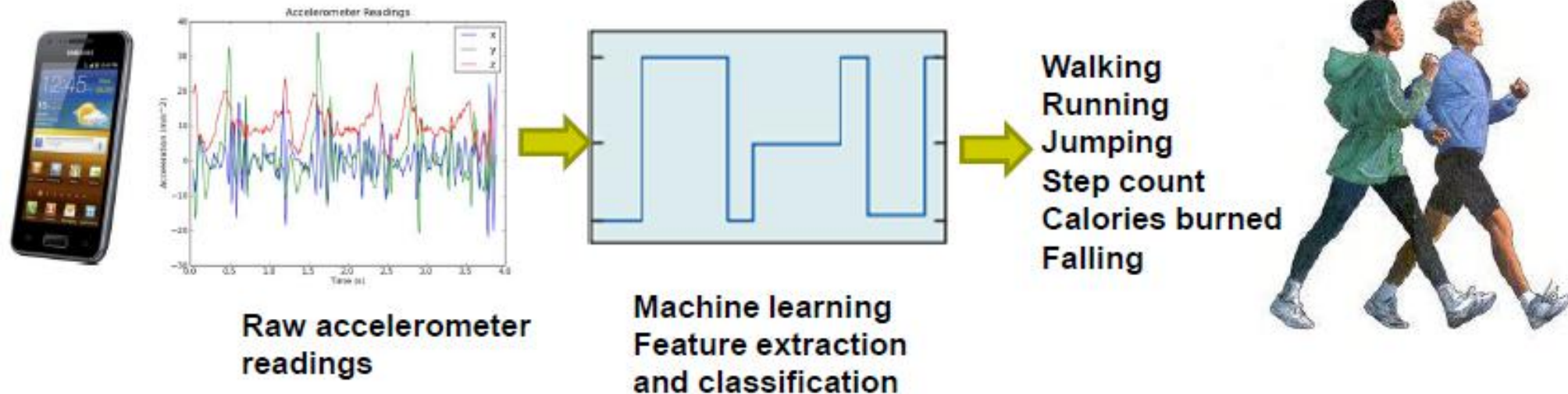
What is a Sensor?

- Converts physical quantity (e.g. light, acceleration, magnetic field) into a signal
- Example: accelerometer converts acceleration along X,Y,Z axes into signal



So What?

- Raw sensor data can be processed into useful info
- **Example:** Raw accelerometer data can be processed/classified to infer user's activity (e.g. walking running, etc)
- Voice samples can be processed/classified to infer whether speaker is nervous or not



Why are we talking about sensors?

- Sensors have been used in cellphones since they were invented ...
 - Microphone, number keys
- What about smartphones?
 - accelerometers, gyroscopes, GPS, cameras, etc ...
- Allowed cellphones explode into different markets
 - R.I.P: Garmin, Tomtom, Kodak, and more
- Instead of carrying around 10 separate devices, now you just need 1

Sensor Applications

Give some examples of sensor use

- Cars
- Computers
- Retail, logistics:
- Buildings
- Environment monitoring
- Industrial sensing & diagnostics

Definitions

- A **sensor** is a device that converts physical quality into an electrical signal.
- It is the interface between the physical world and electrical systems.
- Sensors are required to produce data that the computing system can process.
 - E.g., opening a washing machine stops the washing cycle.
 - Opening of a house door results in activation of a house alarm.

Definitions

- A **transducer** is the device that takes one form of input (energy or signal) and changes into another form.
- A transducer can be part of our earlier defined sensors.
 - Many times the terms sensor and transducer are used interchangeably
 - Sensors measure the change in physical environment and produce electrical signals using a transducer,
 - The transducer takes the measured change in the physical environment and transforms it into a different form of energy (such as an electrical signal)

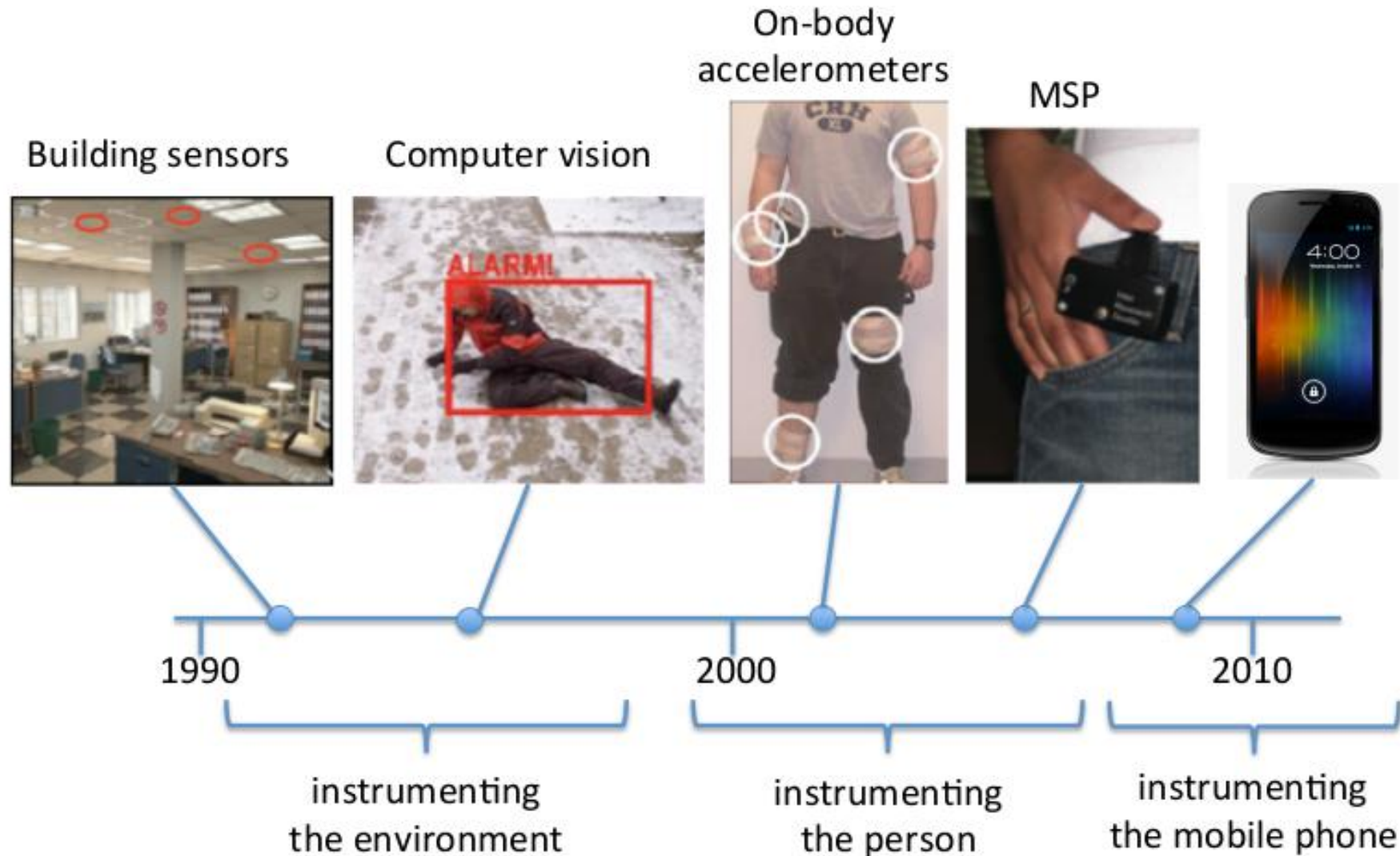
Definitions

- An **actuator** is a transducer that takes one form of energy as input and produces some form of motion, movement, or action.
 - For example, an electrical motor in an elevator converts electrical energy into the vertical movement of going from one floor to another floor of the building.

Common Sensors

- What are some sensors we use every day?
 - Thermometers
 - Radar guns
 - Automatic door openers

History of Sensing Platforms



Android Sensors

- Microphone (sound)
- Camera
- Temperature
- Location (GPS, A-GPS)
- Accelerometer
- Gyroscope (orientation)
- Proximity
- Pressure
- Light



The screenshot displays the AndroSensor application interface on an Android device. The status bar at the top shows the time as 13:16 and various system icons. The app's title bar is blue with the text "AndroSensor". The main content area is divided into several sections, each representing a different sensor type. Each section includes an icon, a title, and specific data readings. The sections are: LOCATION (with a globe icon and Google Maps link), ACCELEROMETER (with a 3D coordinate icon), LIGHT (with a lightbulb icon), MAGNETIC_FIELD (with a horseshoe magnet icon), ORIENTATION (with a compass icon), PROXIMITY (with a speaker icon), and BATTERY STATUS (with a battery icon). The bottom of the screen features a watermark for "ANDROIDFREWARE.NET".

AndroSensor

LOCATION:
Latitude: 41.9305
Longitude: 25.5567
Altitude: unavailable
Accuracy: 241.9 ft
Provider: network

ACCELEROMETER: (0.2mA)
x:0.4597 m/s²
y:-0.6129 m/s²
z:9.6534 m/s²

LIGHT: (0.8mA)
5000.0 lux

MAGNETIC_FIELD: (4.0mA)
X:12 uT
Y:-7 uT
Z:-49 uT

ORIENTATION: (4.2mA)
X:233.0°
Y:3.0°
Z:2.0°

PROXIMITY: (0.8mA)
2.0 in

BATTERY STATUS:
Temperature: 116.6 °F
Level: 66%
Voltage: 3.93 V

ANDROIDFREWARE.NET

Hardware and software sensors

- Sensors can be hardware- or software-based.
- Hardware-based sensors
 - physical components built into a handset or tablet device.
 - Examples: light sensor, proximity sensor, magnetometer, accelerometer
- Software-based sensors are not physical components, although they mimic hardware-based sensors.
 - Derive their data from one or more of the hardware-based sensors and manipulate it
 - Are sometimes called *virtual sensors* or *composite sensors*
 - Examples: linear acceleration, **orientation**.

Android Sensor Programming

Android Sensor Framework

- Enables apps to:
 - Access sensors available on device and
 - Acquire raw sensor data
- With the Android sensor framework you can:
 - Determine **which sensors are available** on a device.
 - Determine an individual **sensor's capabilities**, such as its maximum range, manufacturer, power requirements, and resolution.
 - **Acquire raw sensor data** and define the minimum rate at which you acquire sensor data.
 - **Register and unregister sensor event listeners** that monitor sensor changes.

Contd.

The following classes are the key parts of the Android sensor framework:

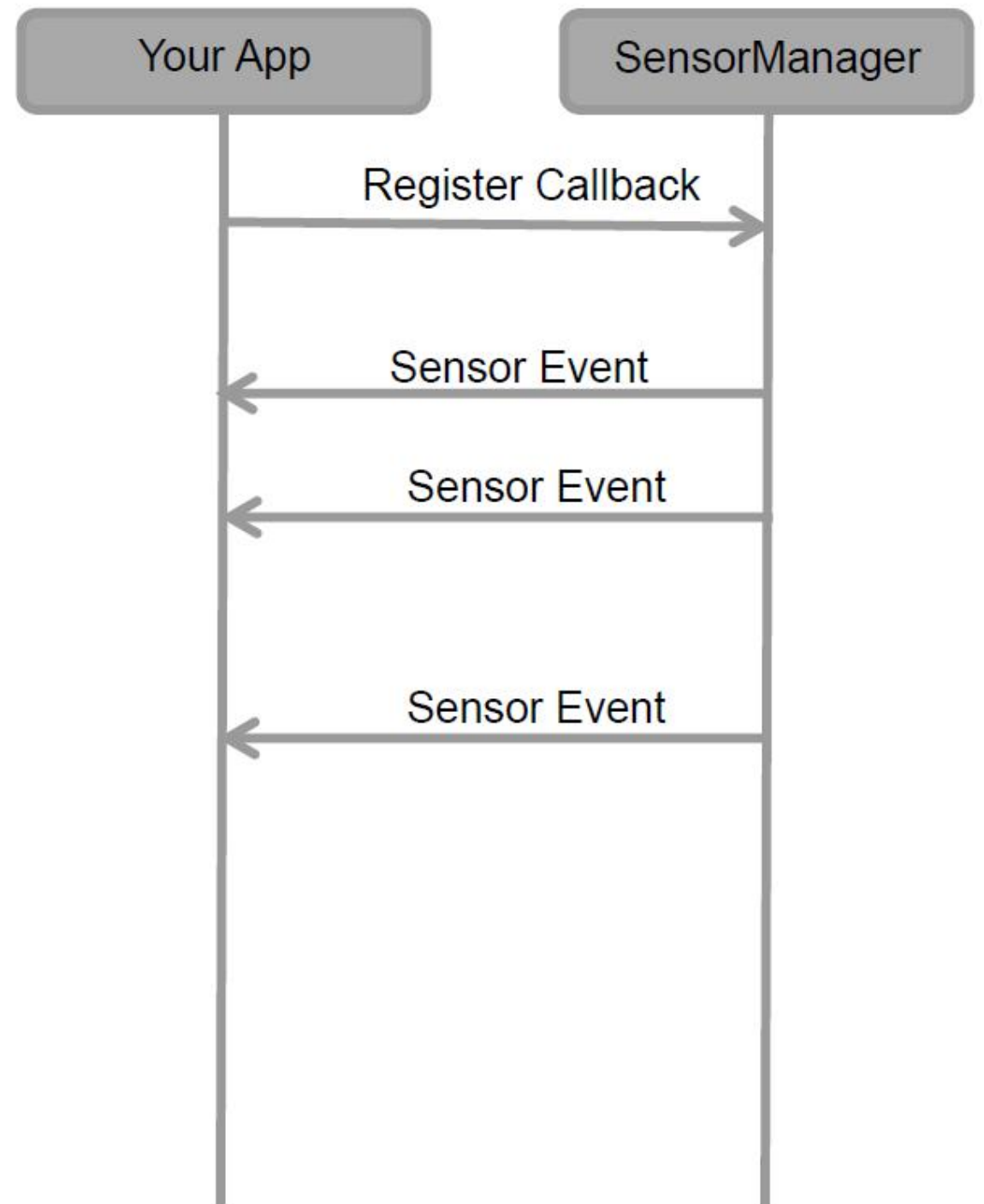
- **SensorManager**
 - Channel between your classes and sensors
- **Sensor**
 - Abstract representation of sensors on device
- **SensorEvent**
 - Represents information about a sensor event
- **SensorEventListener**
 - Register with SensorManager to listen for events from a sensor

Using Sensors

- Obtain the `SensorManager` and `Sensor` object
- Create a `SensorEventListener` for Sensor Events
 - Logic that responds to Sensor Event
 - Varying amounts of data from sensor depending on the type of sensor
- Register the sensor listener with a `Sensor` using `SensorManager`
- Unregister when done
 - A good thing to be done in the `onPause` or `onStop` method
- Override callback methods

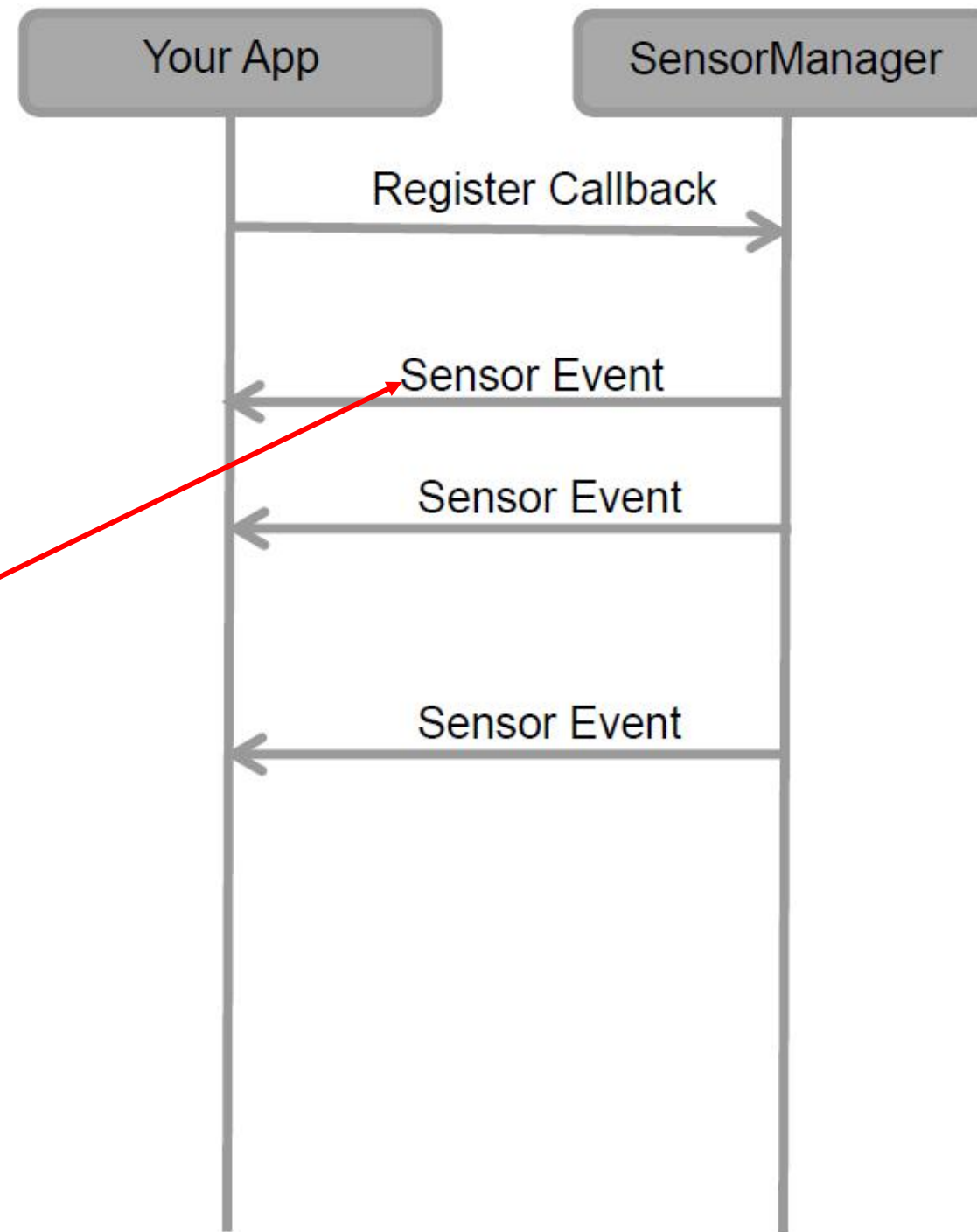
Sensor Events and Callbacks

- Sensors send events to sensor manager asynchronously, when new data arrives
- General approach:
 - App registers callbacks
 - **SensorManager** notifies app of sensor event whenever new data arrives (or accuracy changes)



Sensor Class

- A class that can be used to create instance of a specific sensor
- Has methods used to determine a sensor's capabilities
- Included in sensor event object



Sensor Availability

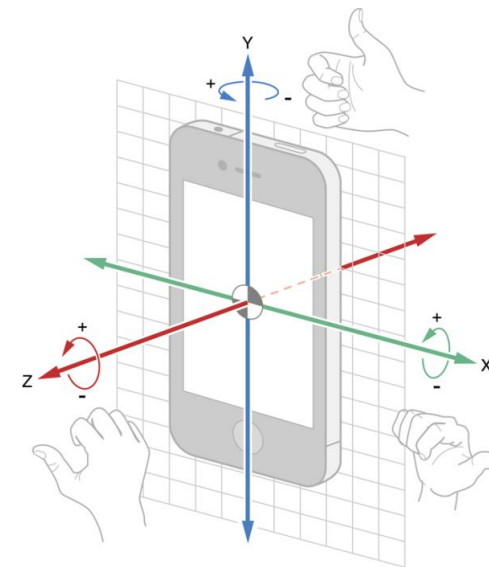
- Sensor availability varies from device to device, it can also vary between Android versions
 - Most devices have accelerometer and magnetometer
 - Some devices have barometers or thermometers
 - Device can have more than one sensor of a given type
 - Availability varies between Android versions

Show all sensors available in a device

```
public class MainActivity extends AppCompatActivity {  
    private SensorManager mSensorManager;  
    TextView tv;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        tv = findViewById(R.id.tv);  
        mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);  
        List<Sensor> deviceSensors = mSensorManager.getSensorList(Sensor.TYPE_ALL);  
        for(int i=0; i<deviceSensors.size(); i++){  
            tv.append("\n"+deviceSensors.get(i).getName()+"\n"+deviceSensors.get(i).getVendor());  
        }  
    }  
}
```

Sensor Types Supported by Android

- TYPE_PROXIMITY
 - Measures an **object's proximity to device's screen**
 - **Common uses:** determine if handset is held to ear
- TYPE_GYROSCOPE
 - Measures device's **rate of rotation** around X,Y,Z axes in rad/s
 - **Common uses:** rotation detection (spin, turn, etc)



Available Sensors in Most Android Devices

Sensor	Used for
<u>TYPE_ACCELEROMETER</u>	Motion detection (shake, tilt, and so on).
<u>TYPE_AMBIENT_TEMPERATURE</u>	Monitoring air temperature.
<u>TYPE_GRAVITY</u>	Motion detection (shake, tilt, and so on).
<u>TYPE_GYROSCOPE</u>	Rotation detection (spin, turn, and so on).
<u>TYPE_LIGHT</u>	Controlling screen brightness.
<u>TYPE_LINEAR_ACCELERATION</u>	Monitoring acceleration along a single axis.
<u>TYPE_MAGNETIC_FIELD</u>	Creating a compass.
<u>TYPE_PRESSURE</u>	Monitoring air pressure changes.
<u>TYPE_PROXIMITY</u>	Phone position during a call.
<u>TYPE_RELATIVE_HUMIDITY</u>	Monitoring ambient humidity, and dew point.
<u>TYPE_TEMPERATURE</u>	Monitoring temperatures.

Sensors

- TYPE_ACCELEROMETER
 - Hardware
 - Acceleration force in m/s^2
 - x,y,z axis
 - Includes gravity
- TYPE_AMBIENT_TEMPERATURE
 - Hardware
 - Room temperature in degree Celsius
- TYPE_GRAVITY
 - Software
 - Just gravity
 - If phone at rest same as TYPE_ACCELEROMETER

Sensors

- TYPE_GYROSCOPE
 - Hardware
 - Measures device rate of rotation in radians/seconds around 3 axis
- TYPE_LIGHT
 - Hardware
 - Light level in lx
 - Lux is SI measures illuminance in luminous flux per unit area
- TYPE_LINEAR_ACCELERATION
 - Software
 - Measures acceleration force applied to device in 3 axes excluding the force of gravity

Sensors

- TYPE_MAGNETIC_FIELD
 - Hardware
 - Ambient geomagnetic field in all 3 axes
 - μ T Micro Teslas
- TYPE_PRESSURE
 - Hardware
 - Ambient air pressure in hPa or mbar
 - Force per unit area

Sensors

- TYPE_PROXIMITY
 - Hardware
 - Proximity of an object in cm relative to the view screen of a device
 - Typically used to determine if handset is being held to person's ear during a call
- TYPE_RELATIVE_HUMIDITY
 - Hardware
 - Ambient humidity in percent (0 to 100)
- TYPE_TEMPERATURE
 - Hardware
 - Temperature of the device in degree Celcius

- TYPE_STEP_DETECTOR
 - Triggers sensor event each time user takes a step (**single step**)
 - Delivered event has value of 1.0 + timestamp of step
- TYPE_STEP_COUNTER
 - Also triggers a sensor event each time user takes a step
 - Delivers total ***accumulated number of steps since this sensor was first registered by an app,***
 - Tries to eliminate false positives
- **Common uses:** step counting, pedometer apps
- Requires hardware support, available in Nexus 5

Sensor Capabilities

Various methods in Sensor Class to get capabilities of Sensor

- Minimum Delay (in MicroSeconds) - `getMinDelay()`
- Power Consumption (in MicroAmpere) - `getPower()`
- Maximum Range - `getMaximumRange()`
- Resolution - `getResolution()`

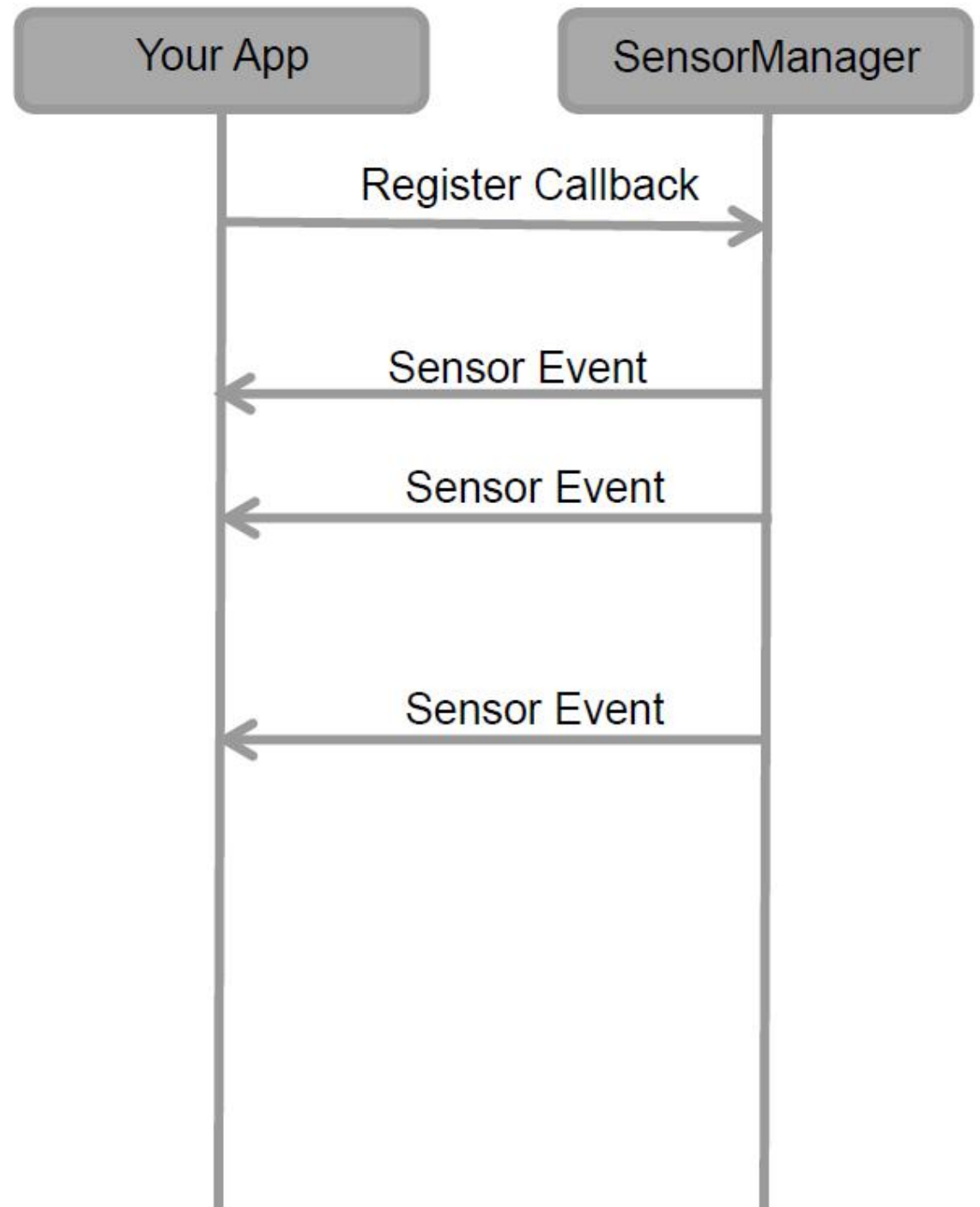
Managing Sensor Accuracy

Accuracy is represented by one of four status constants:

- **SENSOR_STATUS_ACCURACY_HIGH**
 - Indicates that this sensor is reporting data with maximum accuracy
- **SENSOR_STATUS_ACCURACY_LOW**
 - Indicates that this sensor is reporting data with low accuracy, calibration with the environment is needed
- **SENSOR_STATUS_ACCURACY_MEDIUM**
 - Indicates that this sensor is reporting data with an average level of accuracy, calibration with the environment may improve the readings
- **SENSOR_STATUS_UNRELIABLE**
 - Indicates that the values returned by this sensor cannot be trusted, calibration is needed or the environment doesn't allow readings

SensorEvent

- Android system sensor event information as a **sensor event object**
- **Sensor event object** includes:
 - **Sensor:** Type of sensor that generated the event
 - **Values:** Raw sensor data
 - **Accuracy:** Accuracy of the data
 - **Timestamp:** Event timestamp



Sensor Delay

- Data should begin to come in at the rate you specified as an argument
- The rate can be `SENSOR_DELAY_NORMAL`
- `SENSOR_DELAY_UI` (For basic UI interaction)
- `SENSOR_DELAY_GAME` (A high rate that many games require)
- `SENSOR_DELAY_FASTEST` (without any delay)
- You can also specify the delay as an absolute value (in microseconds).

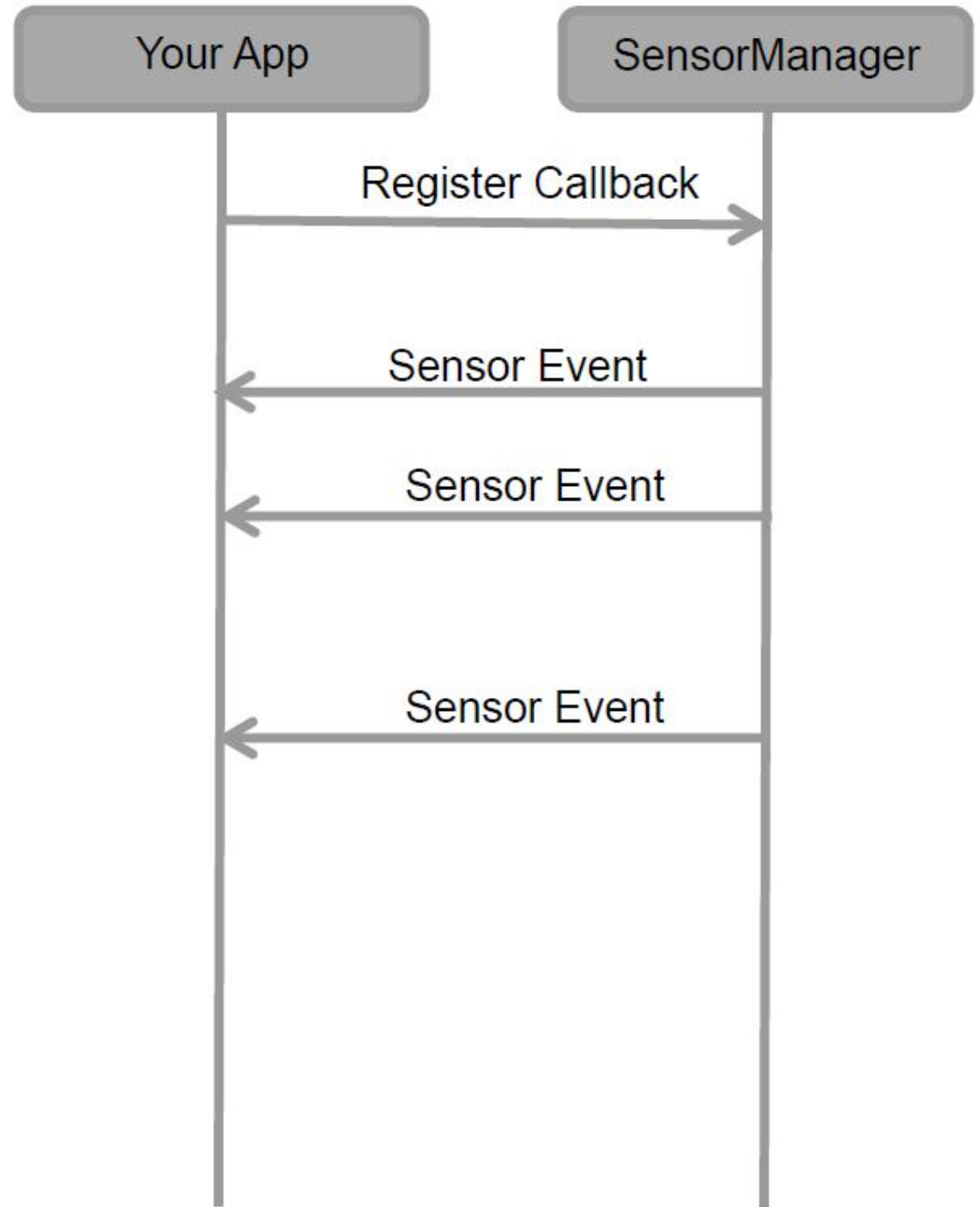
- **Sensor Values Depend on Sensor Type**

Sensor	Sensor event data	Description	Units of measure
TYPE_ACCELEROMETER	SensorEvent.values[0]	Acceleration force along the x axis (including gravity).	m/s ²
	SensorEvent.values[1]	Acceleration force along the y axis (including gravity).	
	SensorEvent.values[2]	Acceleration force along the z axis (including gravity).	
TYPE_GRAVITY	SensorEvent.values[0]	Force of gravity along the x axis.	m/s ²
	SensorEvent.values[1]	Force of gravity along the y axis.	
	SensorEvent.values[2]	Force of gravity along the z axis.	
TYPE_GYROSCOPE	SensorEvent.values[0]	Rate of rotation around the x axis.	rad/s
	SensorEvent.values[1]	Rate of rotation around the y axis.	
	SensorEvent.values[2]	Rate of rotation around the z axis.	
TYPE_GYROSCOPE_UNCALIBRATED	SensorEvent.values[0]	Rate of rotation (without drift compensation) around the x axis.	rad/s
	SensorEvent.values[1]	Rate of rotation (without drift compensation) around the y axis.	
	SensorEvent.values[2]	Rate of rotation (without drift compensation) around the z axis.	
	SensorEvent.values[3]	Estimated drift around the x axis.	
	SensorEvent.values[4]	Estimated drift around the y axis.	
	SensorEvent.values[5]	Estimated drift around the z axis.	

TYPE_LINEAR_ACCELERATION	<code>SensorEvent.values[0]</code>	Acceleration force along the x axis (excluding gravity).	m/s ²
	<code>SensorEvent.values[1]</code>	Acceleration force along the y axis (excluding gravity).	
	<code>SensorEvent.values[2]</code>	Acceleration force along the z axis (excluding gravity).	
TYPE_ROTATION_VECTOR	<code>SensorEvent.values[0]</code>	Rotation vector component along the x axis ($x * \sin(\theta/2)$).	Unitless
	<code>SensorEvent.values[1]</code>	Rotation vector component along the y axis ($y * \sin(\theta/2)$).	
	<code>SensorEvent.values[2]</code>	Rotation vector component along the z axis ($z * \sin(\theta/2)$).	
	<code>SensorEvent.values[3]</code>	Scalar component of the rotation vector $((\cos(\theta/2))^1$	
TYPE_SIGNIFICANT_MOTION	N/A	N/A	N/A
TYPE_STEP_COUNTER	<code>SensorEvent.values[0]</code>	Number of steps taken by the user since the last reboot while the sensor was activated.	Steps
TYPE_STEP_DETECTOR	N/A	N/A	N/A

SensorEventListener

- Interface used to create 2 callbacks that receive notifications (sensor events) when:
 - Sensor values change (**onSensorChange()**) or
 - When sensor accuracy changes (**onAccuracyChanged()**)



Sensor API Tasks

- **Sensor API Task 1: Identifying sensors and their capabilities**
- Why identify sensor and their capabilities at runtime?
 - Disable app features using sensors not present, or
 - Choose sensor implementation with best performance
- **Sensor API Task 2: Monitor sensor events**
- Why monitor sensor events?
 - To acquire raw sensor data
 - Sensor event occurs every time sensor detects change in parameters it is measuring

Identifying Sensors and Sensor Capabilities

- First create instance of **SensorManager** by calling **getSystemService()** and passing in **SENSOR_SERVICE** argument

```
private SensorManager mSensorManager;  
  
...  
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

- Then list sensors available on device by calling **getSensorList()**

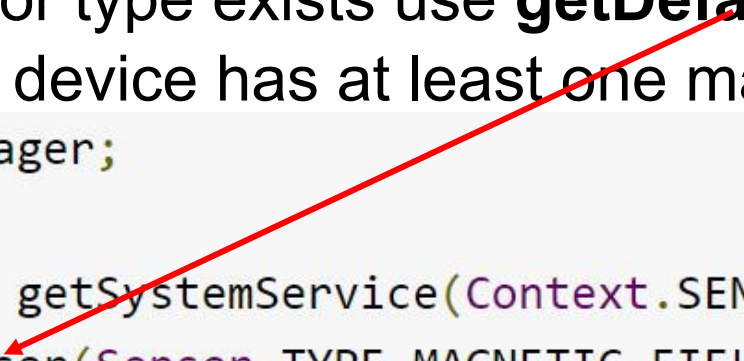
```
List<Sensor> deviceSensors = mSensorManager.getSensorList(Sensor.TYPE_ALL);
```

- To list particular type, use **TYPE_GYROSCOPE**, **TYPE_GRAVITY**, etc

Checking if Phone has at least one of particular Sensor Type

- Device may have multiple sensors of a particular type.
 - E.g. multiple magnetometers
- If multiple sensors of a given type exist, one of them must be designated “the default sensor” of that type
- To determine if specific sensor type exists use **getDefaultSensor()**
- **Example:** To check whether device has at least one magnetometer

```
private SensorManager mSensorManager;  
  
...  
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);  
if (mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null){  
    // Success! There's a magnetometer.  
}  
else {  
    // Failure! No magnetometer.  
}
```



Example: Monitoring Light Sensor Data

- **Goal:** Monitor light sensor data using **onSensorChanged()**, display it in a **TextView** defined in main.xml

Create instance of
Sensor manager

Get default
Light sensor

Called by Android
system when accuracy
of sensor being
monitored changes

```
public class SensorActivity extends Activity implements SensorEventListener {  
    private SensorManager mSensorManager;  
    private Sensor mLight;  
  
    @Override  
    public final void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
  
        mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);  
        mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);  
    }  
  
    @Override  
    public final void onAccuracyChanged(Sensor sensor, int accuracy) {  
        // Do something here if sensor accuracy changes.  
    }  
}
```

Example: Monitoring Light Sensor Data (Contd)

```
@Override
public final void onSensorChanged(SensorEvent event) {
    // The light sensor returns a single value.
    // Many sensors return 3 values, one for each axis.
    float lux = event.values[0];
    // Do something with this sensor value.
}
```

Called by Android system to report new sensor value
Provides SensorEvent object containing new sensor data

Get new light sensor value

```
@Override
protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL);
}
```

Register sensor when app becomes visible

```
@Override
protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(this);
}
```

Unregister sensor if app is no longer visible to reduce battery drain

Handling Different Sensor Configurations

- Different phones have different sensors built in
 - **E.g.** Motorola Xoom has pressure sensor, Samsung Nexus S doesn't
- If app uses a specific sensor, how to ensure this sensor exists on target device?
- Two options
- **Option 1:** Detect device sensors at runtime, enable/disable app features as appropriate
- **Option 2:** Use AndroidManifest.xml entries to ensure that only devices possessing required sensor can see app on Google Play
 - **E.g.** following manifest entry in AndroidManifest ensures that only devices with accelerometers will see this app on Google Play

```
<uses-feature android:name="android.hardware.sensor.accelerometer"  
            android:required="true" />
```

Detecting Sensors at Runtime

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
if (mSensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE) != null){
    // Success! There's a pressure sensor.
}
else {
    // Failure! No pressure sensor.
}
```

Using Sensors

- Recall basics for using a Sensor:
 - Obtain the *SensorManager* object
 - create a *SensorEventListener* for *SensorEvents*
 - logic that responds to sensor event
 - Register the sensor listener with a *Sensor* via the *SensorManager*

Sensor Best Practices

- Unregister sensor listeners
 - when done with Sensor or activity using sensor paused (onPause method)
 - `sensorManager.unregisterListener(sensorListener)`
 - otherwise data still sent and battery resources continue to be used

Sensors Best Practices

- verify sensor available before using it
- use `getSensorList` method and type
- ensure list is not empty before trying to register a listener with a sensor

Sensors Best Practices

- Avoid deprecated sensors and methods
- TYPE_ORIENTATION and TYPE_TEMPERATURE are deprecated as of Ice Cream Sandwich / Android 4.0

Sensors Best Practices

- Don't block the `onSensorChanged()` method
 - 50 updates a second for `onSensorChange` method not uncommon
 - if necessary save event and do work in another thread or asynch task

Sensor Best Practices

- Testing on the emulator

Sensor Coordinate System

- In general, the sensor framework uses a standard 3-axis coordinate system to express data values.
- For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation
 - +x to the right
 - +y up
 - +z out of the front face

