



UNIVERSITY OF CALIFORNIA  
**MERCED**

# Mobile Computing

CSE 162  
Fall 2025

Hua Huang

Department of Computer Science and Engineering

# Logistics

- Lecture time: 6-7:15 pm, Wednesday and Friday.
  - Location: Student Services Building | Room 130
- Office Hour:
  - Hua Huang: 1-2pm, Wednesday. SE2 275
  - Rahul Hoskeri: 2-3pm, SE2 lobby

# Textbooks

- Required: Head First Android Development, 2nd Edition (access the book through the university library: [link](#))
- (Optional): Manish J. Gajjar: *Mobile Sensors and Context-Aware Computing*, 2015; ISBN: 978-0-12-801660-2, (<https://www.sciencedirect.com/book/9780128016602/mobile-sensors-and-context-aware-computing>)
- (Optional): Raj Kamal: *Mobile Computing*, Oxford University Press, 2019; ISBN: 9780199455416(Any edition is fine, <https://archive.org/details/mobilecomputing000kama>)

# Textbook

- Download the book for free.
  - Head First Android Development, **2nd Edition** (access the book through the university library: [link](#))
- Actually read the book
  - It's not a manual for a quick answer look up. Read and there is much to learn.
- We are using the second edition.



BOOK  
**Head first Android development : a brain-friendly guide**  
Griffiths, Dawn, author.; Griffiths, David, author.  
Sebastopol, CA : O'Reilly Media, Incorporated; 2017; Second edition.  
Available Online >

# Grading

- Labs: 35%
  - Extra credits available
- Three Mid-term exams: 30%
- Final Exams: 30%
- Attendance: 5%

# Lab Overview

- A series of android programming projects to familiarize with mobile programming
  - UI
  - Sensor
  - Location aware services
  - Mobile AI
  - etc



# Question

- Do you?
  1. Own Android phones
  2. Can borrow android phones
  3. Do not own and cannot borrow
- Either phones or virtual machine implementations are fine.

# Lab Schedule

- Approximately One lab project every two weeks:
  - Basic tasks: implement an app. Follow the instruction of the TA
  - Bonus tasks: explore and complete an additional feature of the app
- Lab delivery:
  - Demo the features. Demonstrate that the prescribed features are up and running
  - Show your program. Be prepared to answer questions about your program.
  - Submit your functioning program through Catcourse.

# Plagiarism Policy

- Don't cheat.
  - These are exercises. You don't get punished by writing bugs. Instead, you gain experience and prepare for your future jobs.
  - We are here to help you finish them.
  - Understand every line of codes you write.
- If get caught, the consequence is grave
  - Zero grade for the assignment, fail the class, or worse

# Plagiarism Policy for Labs

- Create each app from scratch
- Naming standards required
- What are not cheating:
  - Codes generated by the IDE
  - Discussion with the TA or classmates and find out how to implement it.
  - Search tutorials about how to do it.

# Attendance Policy

- Physical attendance to the classes and labs are required
  - Lecture notes cannot replace lectures
  - In-class quizzes are not announced before hand
  - Exam questions are often discussed in lectures
- If you cannot attend, contact the instructors before hand
- Grading:
  - 80%+ attendance: 10pt
  - Between 30%-80%: 0-10pt
  - Below 30%: 0pt

# AI Usage

- The wrong way: AI is doing development, you help with debugging
  - AI can generate solutions to existing classical problems, which are also available in stackoverflow.com. It often has poor solution to new and unique problems.
  - AI-generated codes often seem to be convincing, while in fact includes hard-to-find bugs
    - You could end up spending more time debugging than coding on your own
    - You get into trouble in your job interview and your job.

# AI Usage

- The right way: You do the software development. Use AI to guide you about new features
  - Understanding of the program is required
    - Think about your interview
  - Debugging is required
    - If you are to develop anything meaningful, which is anything more complex than a leetcode question, you likely need to do the heavy lifting
    - This class offer you excellent opportunities to practice, with minimal consequence for errors
    - We are here to help with challenges
  - Treat AI as an improved stackoverflow: it finds you answers quickly. But you still need to verify the answers.

# AI Usage

- There are many noises. But AI cannot replace human in software development

[https://www.wsj.com/finance/softwares-death-by-ai-has-been-greatly-exaggerated-b639c0cd?st=icF2Qw&reflink=desktopwebshare\\_permalink](https://www.wsj.com/finance/softwares-death-by-ai-has-been-greatly-exaggerated-b639c0cd?st=icF2Qw&reflink=desktopwebshare_permalink)

# Research Projects

- Research projects are available
  - In the general topics covered by the course: networking, sensing, computing

# Introduction

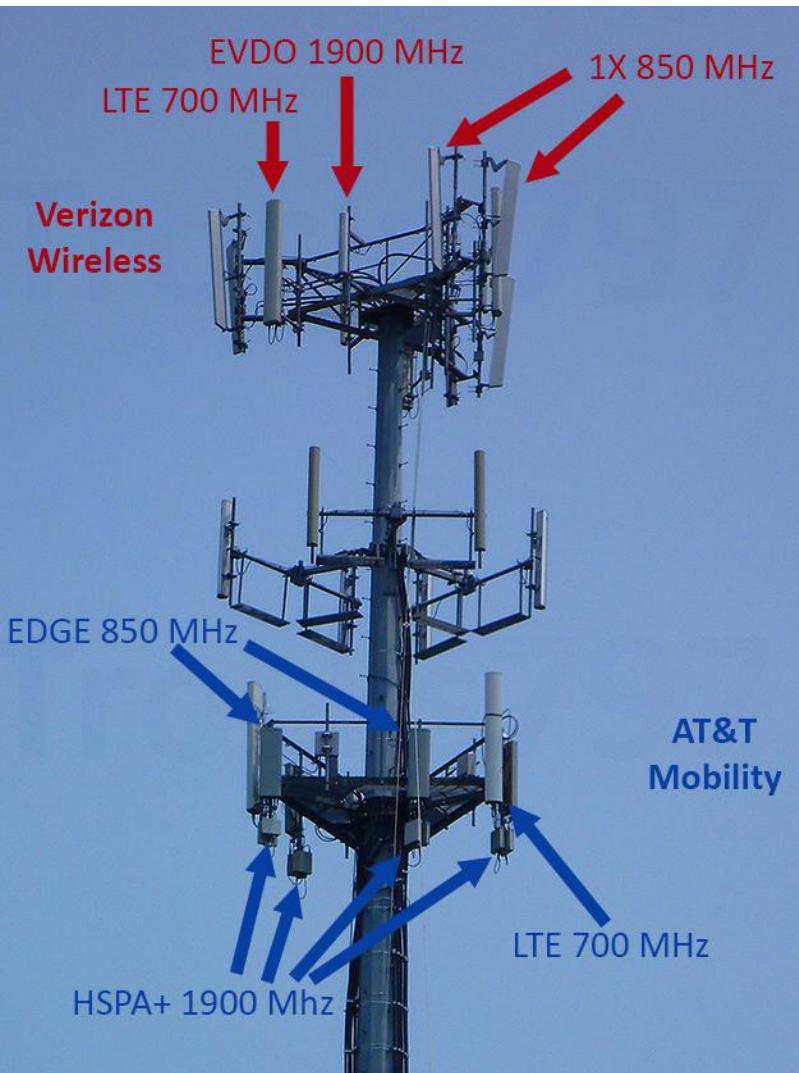
# History of Mobile Computing



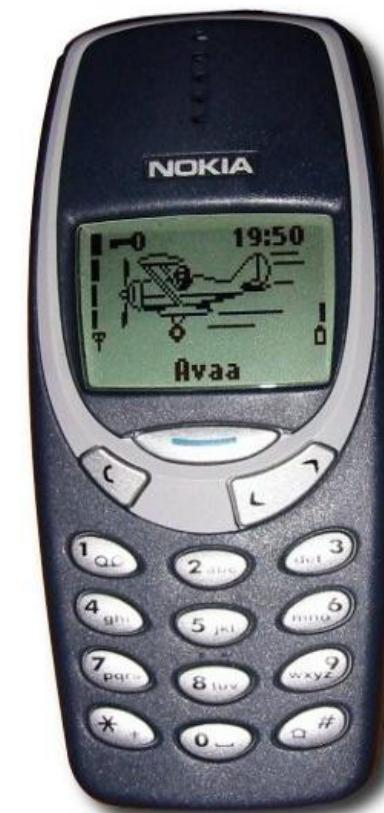
1970s to early 2000:  
Consistent innovation in wireless/mobile communication



- On April 3 1973, Motorola engineer Marty Cooper made the first cell phone call
- The DynaTAC phone weighed about 2.2 pounds and was 10 inches long



- Ubiquitous voice. different frequency bands
  - Global System for Mobile communication (GSM)
  - Code Division Multiple Access (CDMA)
- Data connectivity: 3G, 4G LTE, 5G



# Mobile Phones were just phones for a while...

- Then things began to change
  - More and more people own mobile phones
  - Batteries got better, form factors improved, coverage improved, plans were better...
  - New applications besides communication become available
  - The handset manufacturers didn't want to write all the applications for these new phones
  - However... they didn't want to open up their platform...
  - The first mobile web platform was born (client-server model)

# WAP

- Wireless Application Protocol
- Basically it's a stripped-down HTTP that was meant to be better at transmitting over the unreliable mobile network



# Mobile Phone Economies

- Before there were app stores, purchases were made through SMS
  - Send a text message to a pay-per-text number and they would respond with a ringtone or wallpaper or something else
- Some purchases could be made through platform holder services
  - V-Cast is a mobile application that allows users access and download various forms of entertainment and media from their cell phones.

# When there's money to be made...

- The Internet was full of media that people wanted to consume on the go
- Other handheld devices were selling like gangbusters (Game Boy)
- Phones seemed like an obvious next step
  - A computer that everyone carried with them and was always connected

# Bigger and bigger players get involved

- Nokia had a large portion of the mobile phone market early on
- Other players like Blackberry, Samsung, HTC, etc. also were involved
- Each had (basically) their own operating system, which made third-party development tricky
- What changed with phones?
  - Phones started running existing operating systems (Windows CE and Linux)
  - Mobile carriers started to relax the constraints on what phones could do

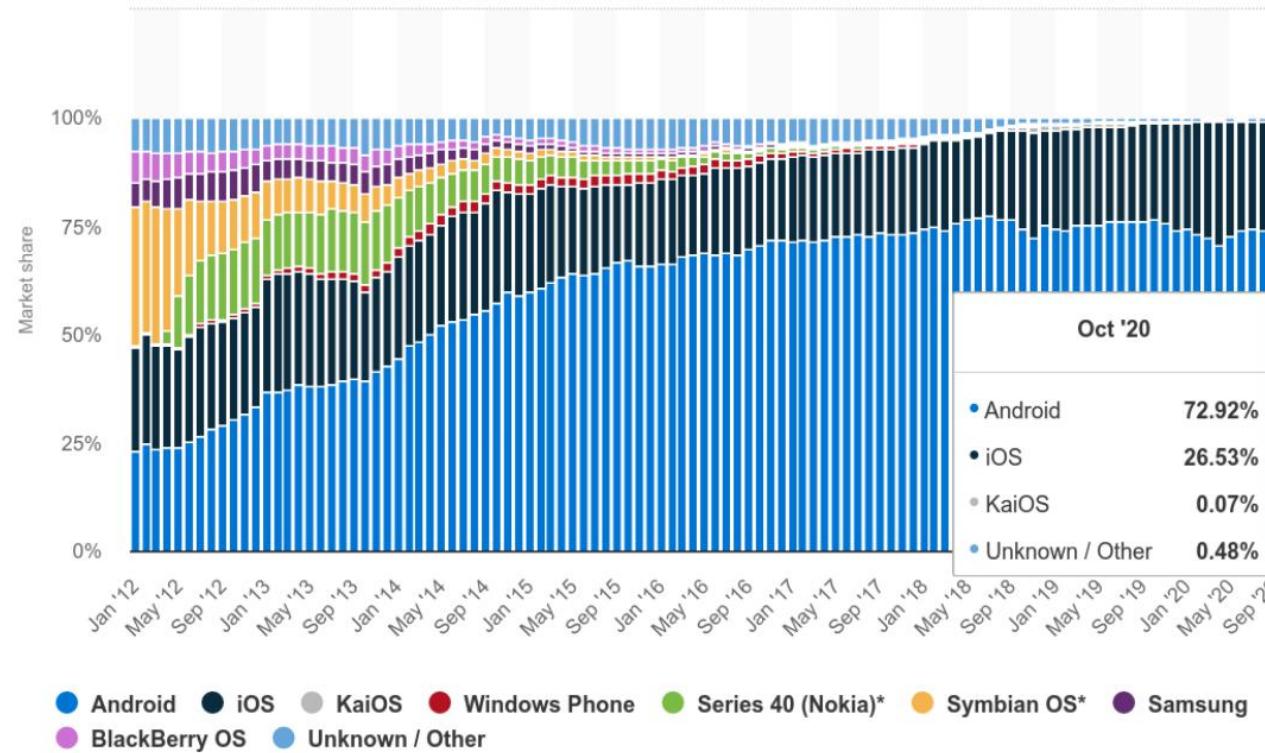
# The Market Fractures

- Microsoft
  - Tried to leverage “write once, run on any Windows device” a bit
  - Worked for a while with PDAs, but never really caught on with phones
- Apple
  - Started off as a phone that had a web browser and iPod bolted on
  - Evolved into much more once the App Store opened
- Google
  - Just provided the OS and let others build the devices (for a while)
  - Open source OS + no developer fees = lots of interest and apps

# Two Main Operating Systems Remain

- iOS
  - iPhones and iPads only
  - Objective-C or Swift using Xcode OR third party platforms (Unity, Xamarin, etc.)
  - Tightly controlled
- Android
  - Thousands of different devices of all shapes and sizes
  - Java using Android Studio OR third party platforms (Unity, Cordova, etc.)
  - Open Source, available to everyone

# Mobile OS Market Share



- Android: 72.9%
- IOS: 26.5%



Android Tablet



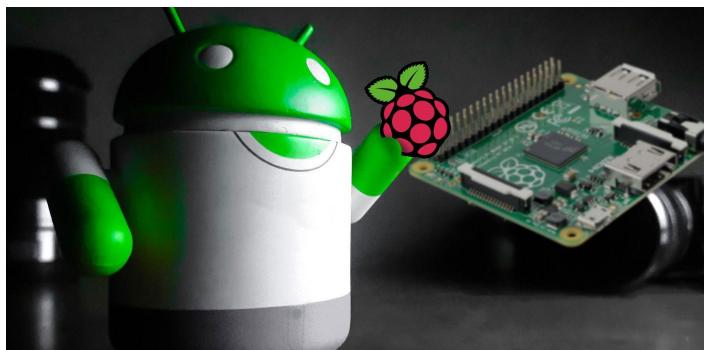
Android Auto



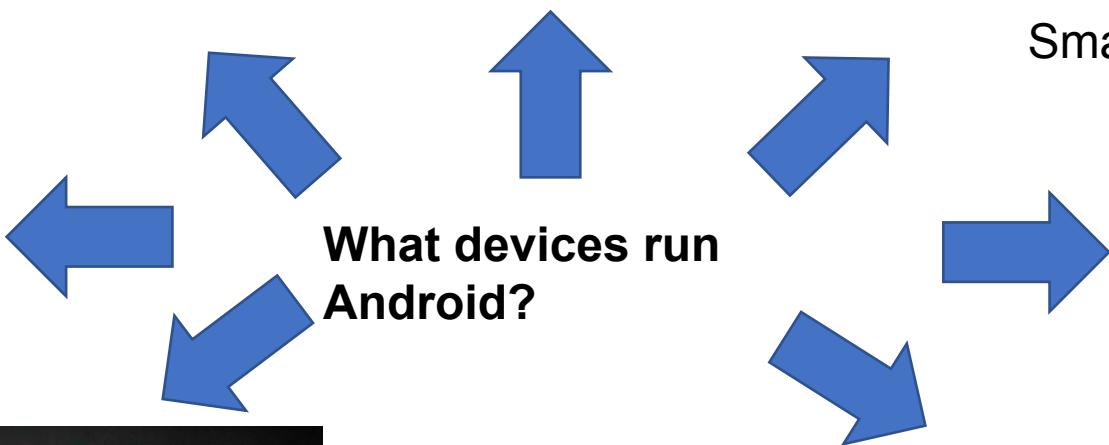
Smartwatch



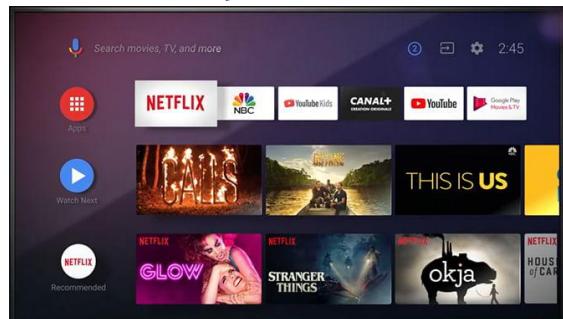
Google Glass



Embedded devices  
(e.g. Raspberry Pi)



Smartphone



Android TV

# Why Android?

- Contains rich mobile and ubicomp programming modules
  - Sensors: GPS, microphone, camera, IMU, ...
  - Data processing: machine learning
  - Application: activity recognition, bio-sign monitoring, speech recognition, ...

# Mobile Computing Concepts

# mo·bile

*adjective*

/'mōbəl, 'mō,bīl/

1. able to move or be moved freely or easily.

"he has a major weight problem and is not very mobile"

*synonyms:* able to move (around), **moving**, walking; **motile**; **ambulant**

# Main Classes of Mobile Computing

- Mobile Phones
  - Initially focused on voice calls
  - Increasingly adding computational capacities



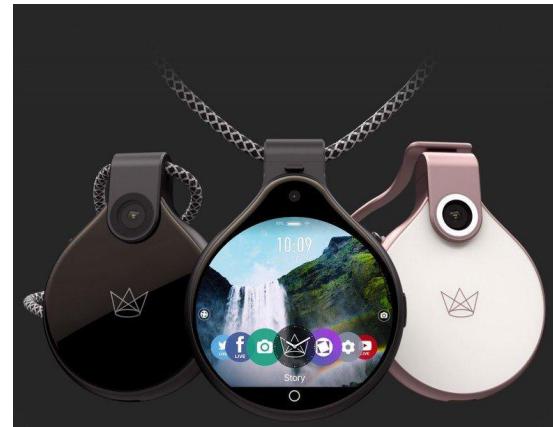
# Main Classes of Mobile Computing

- Mobile Phones
- Portable Computers
  - Portable computers are devices with only essential computing components and input/output devices
  - E.g., laptops, notebooks, tablets, notepads
  - lighter in weight than desktops, through removal of nonessential input/output devices like disc drives, use of compact hard drives, and so on.



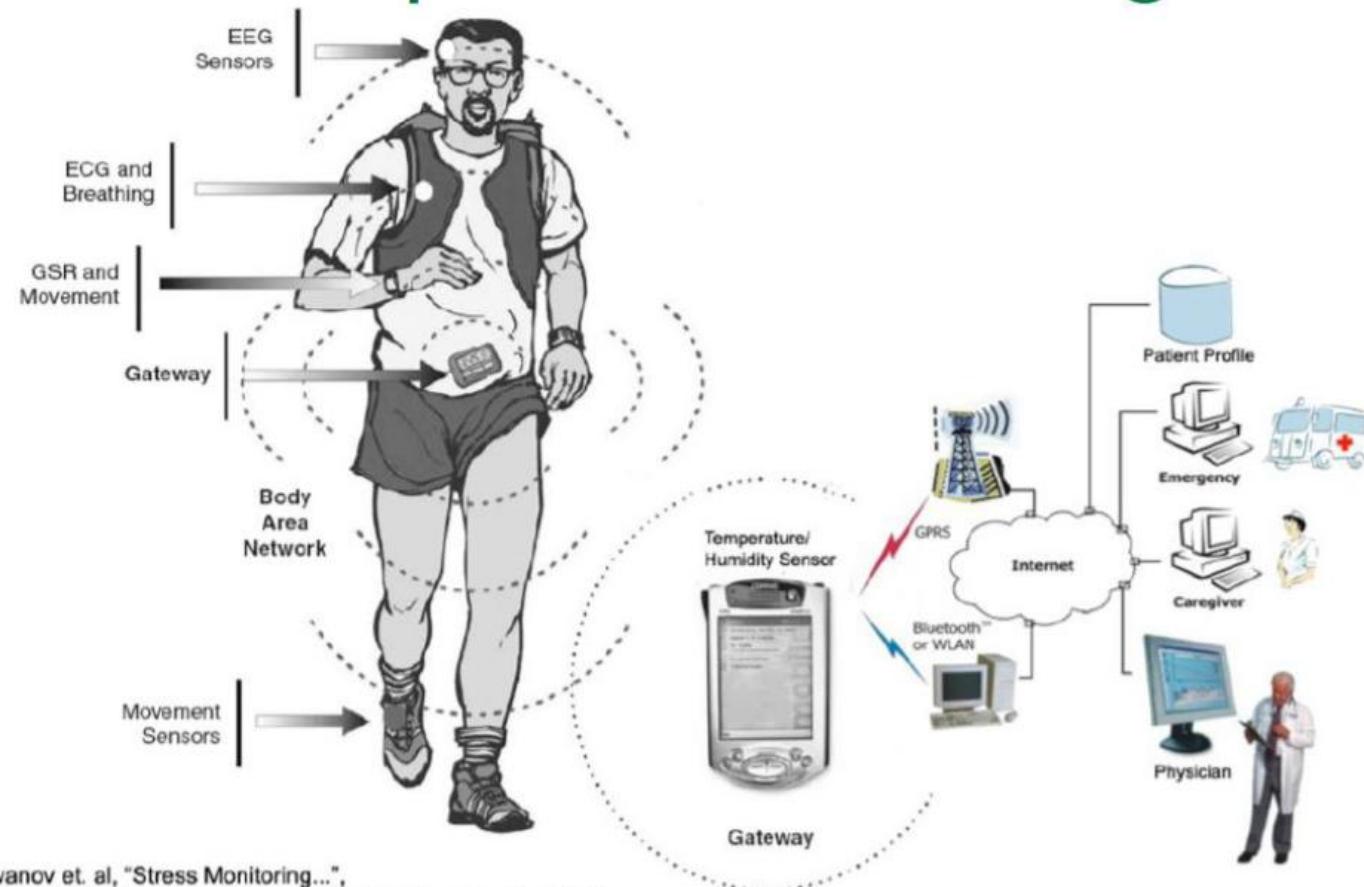
# Main Classes of Mobile Computing

- Mobile Phones
- Portable Computers
- Wearable computers
  - Devices put on the body for computation/connection as well as fashion
  - Capable of sensing, computing, reporting, and connecting
  - E.g., watches, necklaces, implants



# Ubiquitous Computing: Wearable sensors for Health

## remote patient monitoring



Jovanov et. al, "Stress Monitoring... ",  
IEEE Engineering in Medicine and Biology Mag. May/June 2003

# More Mobile Computing Devices



*Body Worn  
Activity Trackers*



*Bluetooth  
Wellness  
Devices*

# Smart Mobile Computing

- Smart = Networking + Computing + Sensing
  - **Sensors:** Camera, video, location, temperature, heart rate sensor, etc
  - **Computing:** Java apps, JVM, apps
    - Powerful processors: Quad core CPUs, GPUs
  - **Communication:** Talk, text, Internet access, chat

# Computing

# SmartPhone Computing Hardware

- Google Pixel XL phone: Quad core 1.6 GHz Snapdragon CPU, Adreno 530 GPU, 4GB RAM
  - A PC in your pocket!!
  - Multi-core CPU, GPU
  - Runs OpenGL ES, OpenCL and now Deep learning (Tensorflow)

# Sensing

# Smartphone Sensors

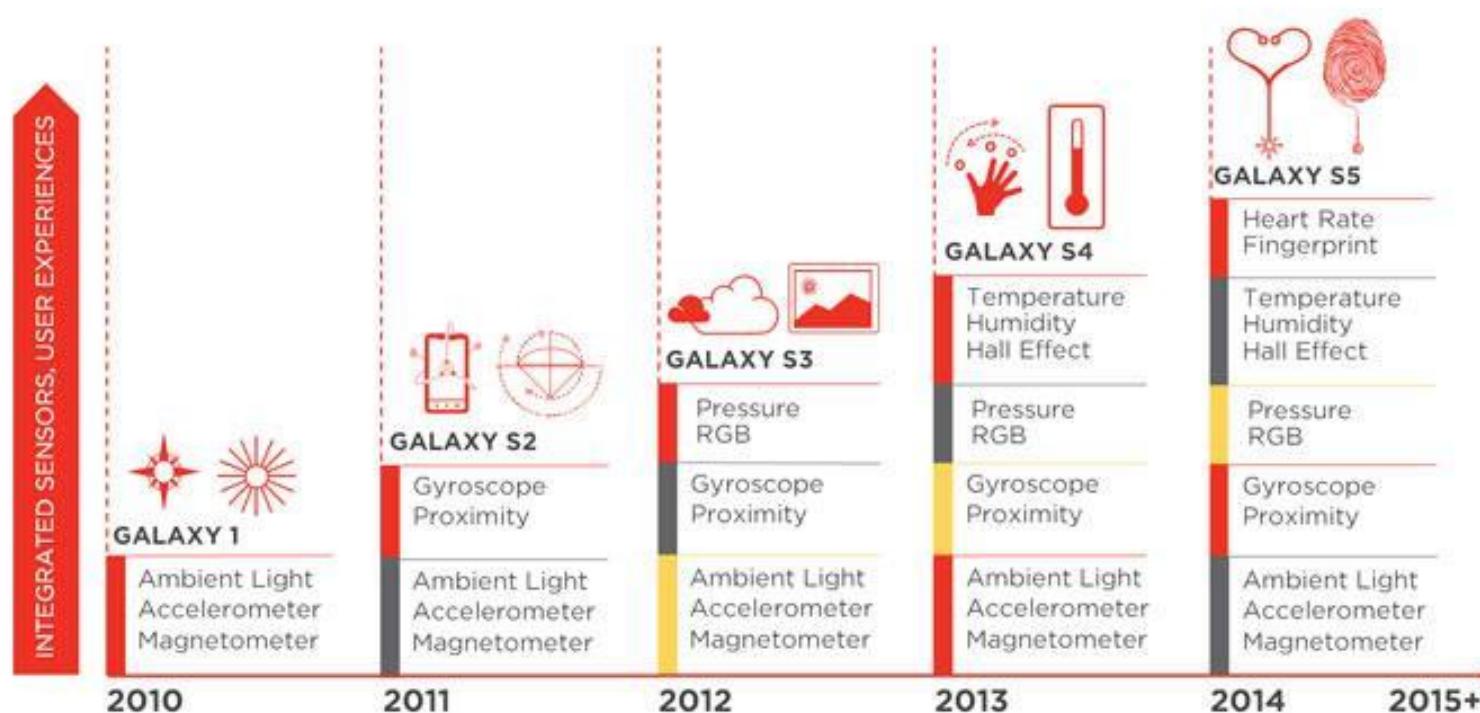
- Typical smartphone sensors today
  - accelerometer, compass, GPS, microphone, camera, proximity
- Can sense physical world, inputs to intelligent sensing apps
  - E.g. Automatically turn off smartphone ringer when user walks into a class



# Growth of Smartphone Sensors

- Every generation of smartphone has more and more sensors!!

## SENSOR GROWTH IN SMARTPHONES



More:

- LIDAR
- mmWave
- pollution sensor

# Sensor

- **Example:** E.g. door senses only human motion, opens
- **Sensor:** device that can sense physical world, programmable, multi-functional for various tasks (movement, temperature, humidity, pressure, etc)
- Device that can take inputs from physical word
  - Also includes camera, microphone, etc
- Ubicomp uses data from sensors in phone, wearables (e.g. clothes), appliances, etc.



(courtesy of MANTIS  
project, U. of Colorado)



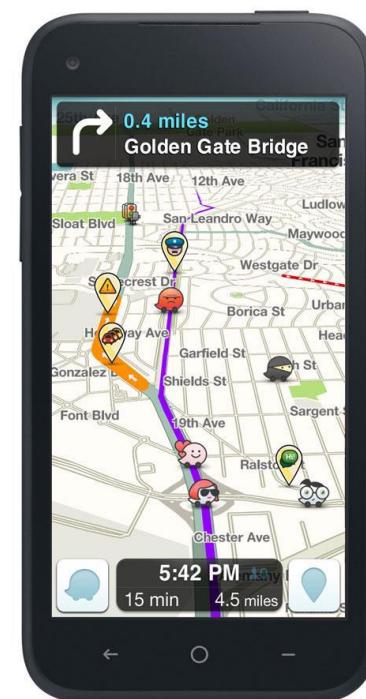
RFID tags



Tiny Mote Sensor,  
UC Berkeley

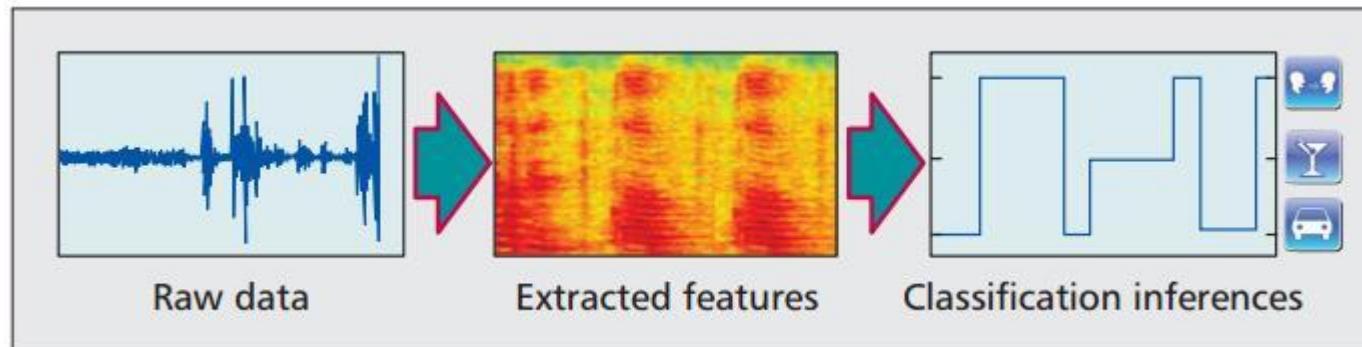
# Mobile Sensing

- Mobile devices can sense human, environment
- Example: Human activity sensing (e.g. walking, driving, climbing stairs, sitting, lying down)
- Example 2: Waze crowdsourced traffic

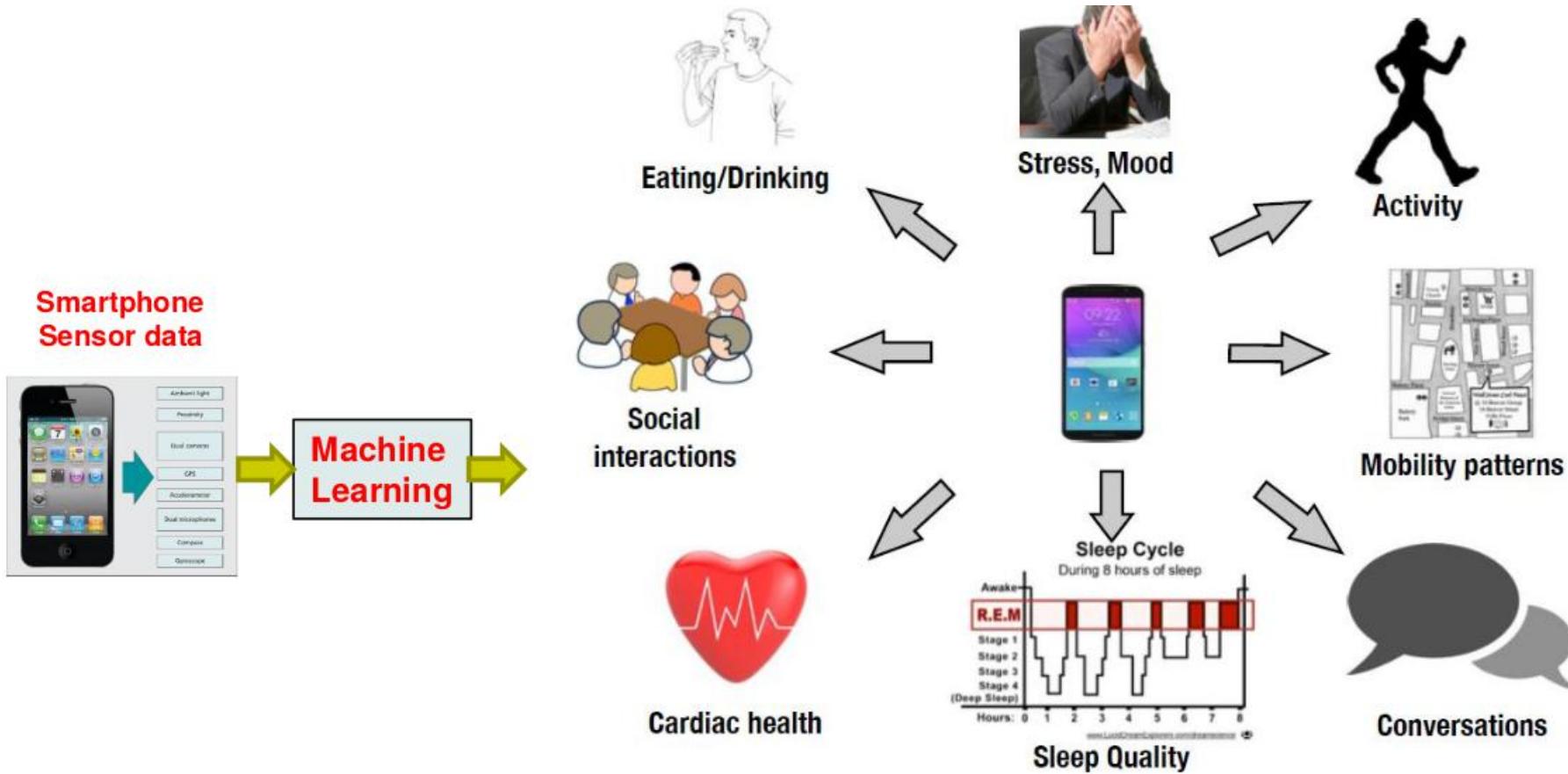


# Sensor Data Processing

- Machine learning commonly used to process sensor data
  - Action to be inferred is hand-labelled to generate training data
  - Sensor data is mined for combinations of sensor readings corresponding to action
- Example: Smartphone detects user's activity (e.g. walking, running, sitting,) by classifying accelerometer sensor data



# What can be detected by the phone?



# Networking

# Wireless Networks On a Phone

- Wi-Fi (802.11): (e.g. Starbucks Wi-Fi)
- Cellular networks: (e.g. Sprint network)
- Bluetooth: (e.g. car speaker)
- Near Field Communications (NFC)
  - e.g. Mobile pay: swipe phone at Dunkin Donuts



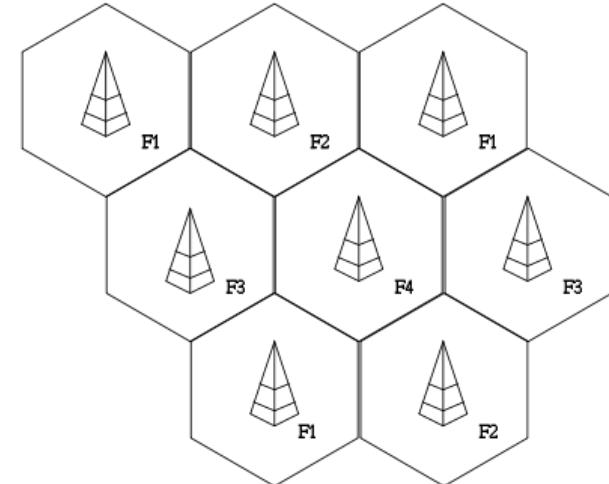
**WIFI**



**NFC**



**Bluetooth**



**Celluar Networks**

# Wireless Networks Comparison

<b>Network Type</b>	<b>Speed</b>	<b>Range</b>	<b>Power</b>	<b>Common Use</b>
WLAN	600 Mbps	45 m – 90 m	100 mW	Internet.
LTE (4G)	5-12 Mbps	35km	120 – 300 mW	Mobile Internet
3G	2 Mbps	35km	3 mW	Mobile Internet
Bluetooth	1 – 3 Mbps	100 m	1 W	Headsets, audio streaming.
Bluetooth LE	1 Mbps	100+ m	.01–.5 W	Wearables, fitness.
NFC	400 kbps	20 cm	200 mW	Mobile Payments

# Mobile Application Architecture

Hua Huang

# Assignments

- Read Chapter 1 of *Head start for Android Programming*. Getting Started: Diving In
- Complete the Android development environment setup. Generate your first running Android app.

# Computer Requirement

- Installing Android Studio:

## Windows



**Note:** Windows machines with ARM-based CPUs aren't currently supported.

Here are the system requirements for Windows:

Requirement	Minimum	Recommended
OS	64-bit Microsoft Windows 10	Latest 64-bit version of Windows
RAM	Studio: 8 GB Studio & Emulator: 16GB	32GB
CPU	Virtualization support Required (Intel VT-x or AMD-V, enabled in BIOS). CPU microarchitecture after 2017.  <a href="#">Intel 8th Gen Core i5 / AMD Zen Ryzen</a> (e.g., Intel i5-8xxx, Ryzen 1xxx).	Virtualization support Required (Intel VT-x or AMD-V, enabled in BIOS).  Latest CPU microarchitecture. Look for CPUs from the Intel Core i5, i7, or i9 series and or the suffixes H/HK/HX for laptop or suffixes S/F/K for desktop, or the AMD Ryzen 5, 6, 7, or 9 series.  Please be aware that Intel® Core™ N-Series and U-Series processors are not recommended due to insufficient performance.
Disk space	Studio: 8 GB of free space. Studio & Emulator: 16GB of free space	Solid state drive with 32 GB or more
Screen resolution	1280 x 800	1920 x 1080
GPU	Studio: None Studio & Emulator: GPU with 4GB VRAM such as Nvidia Geforce 10 series or newer, or AMD Radeon RX 5000 or newer with the latest drivers	GPU with 8GB VRAM such as Nvidia Geforce 20 series or newer, or AMD Radeon RX6600 or newer with the latest drivers.

# Resources for Laptops

- Technology Resources Program (TRP)
  - Short term laptop loan
  - <https://ue.ucmerced.edu/student-resources/technology-resources>
- School of Engineering Laptop policy
  - <https://engr-advising.ucmerced.edu/policies/soe-policies>
  - <https://engr-advising.ucmerced.edu/sites/engr-advising.ucmerced.edu/files/page/documents/2019policyonlaptopssoecomputer.pdf>

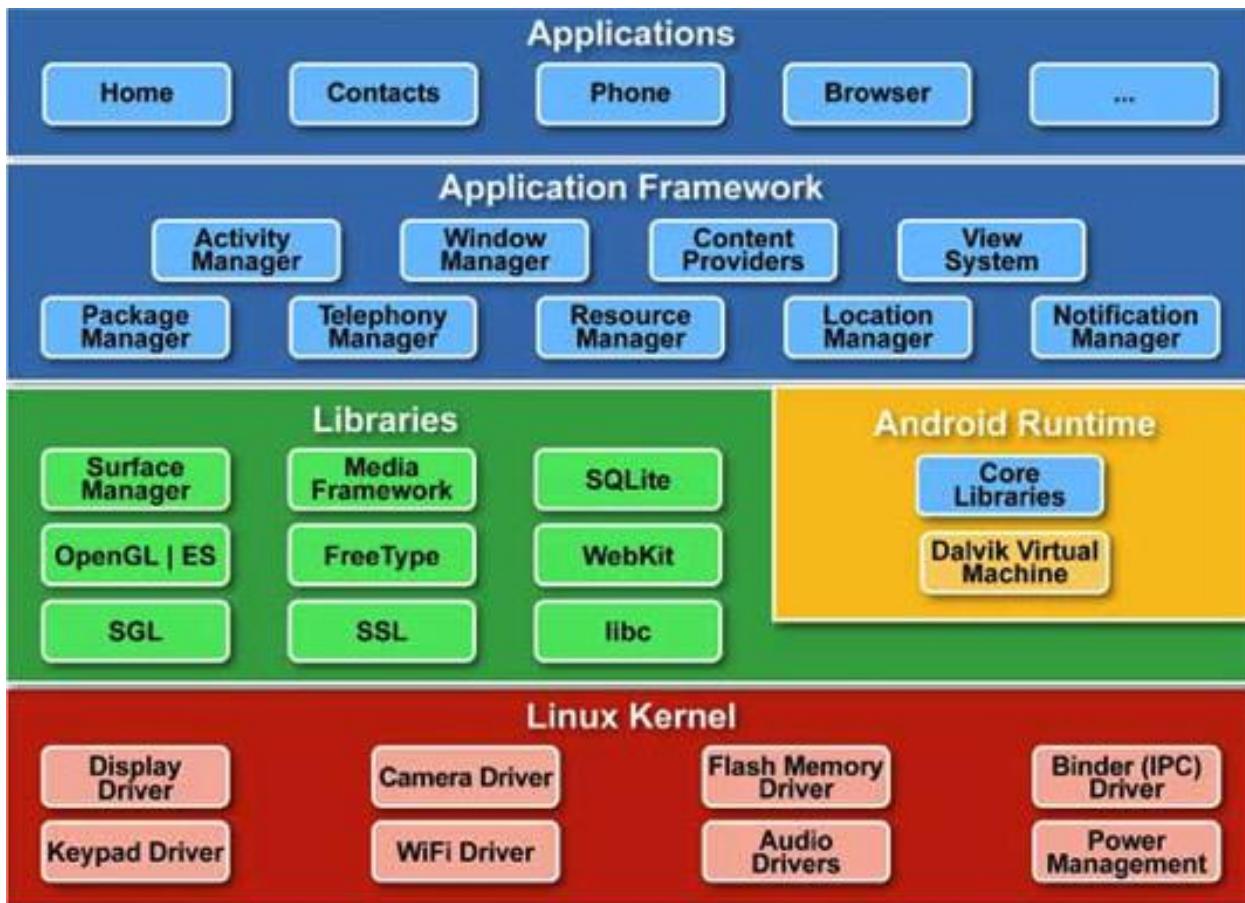
# Android Architecture

# The Basics

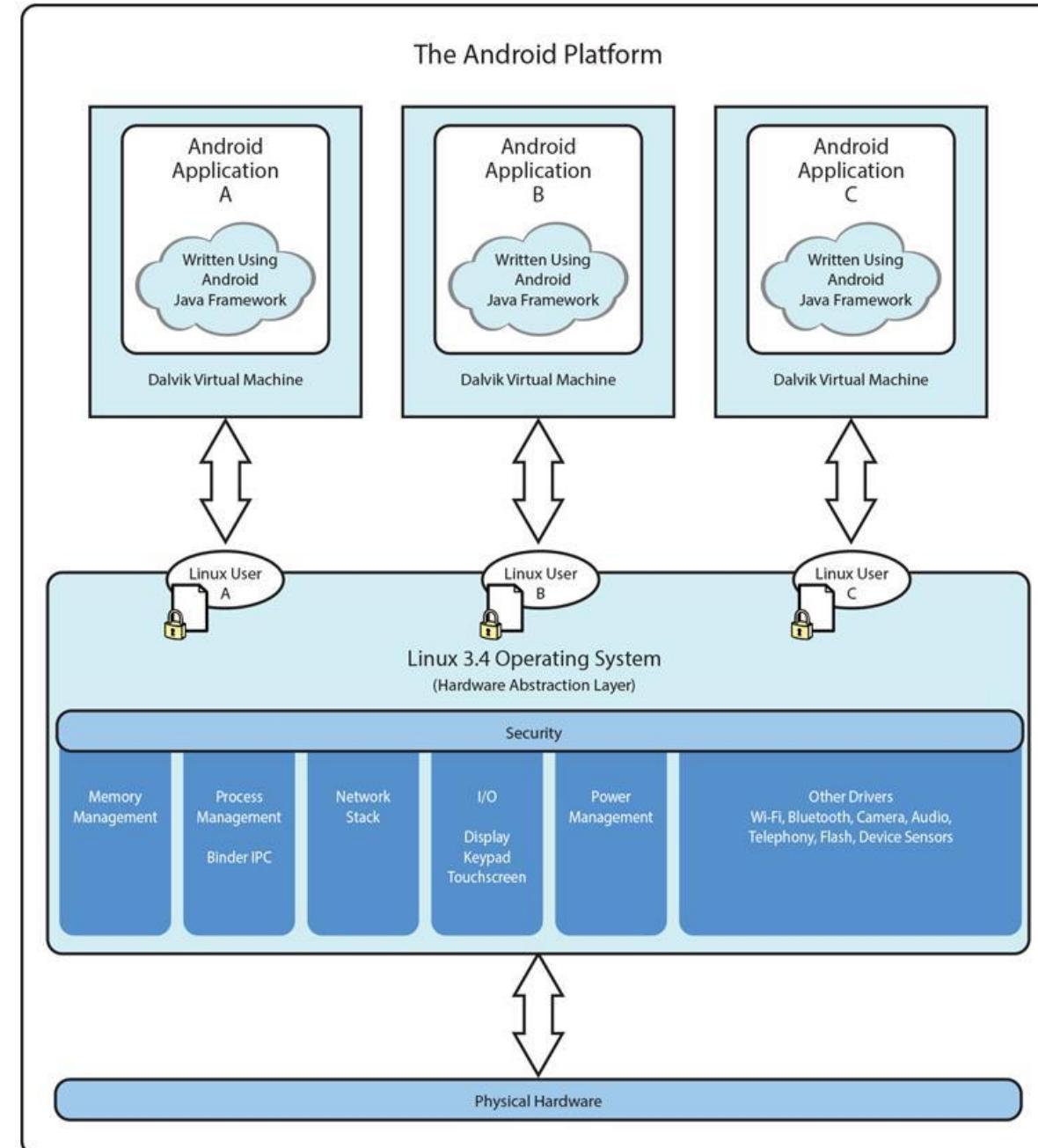
- In general, all apps are written in Java or Kotlin
- A compiled Android app is an .apk
- Android apps must be digitally signed in some way to execute
  - so that we know who releases the apk
- This digital signature can be a debug certificate that comes default with any installation

# The Android Architecture

- **OS:** Linux kernel, drivers
- **Apps:** programmed & UI in Java
- **Libraries:** OpenGL ES (graphics), SQLite (database), etc.

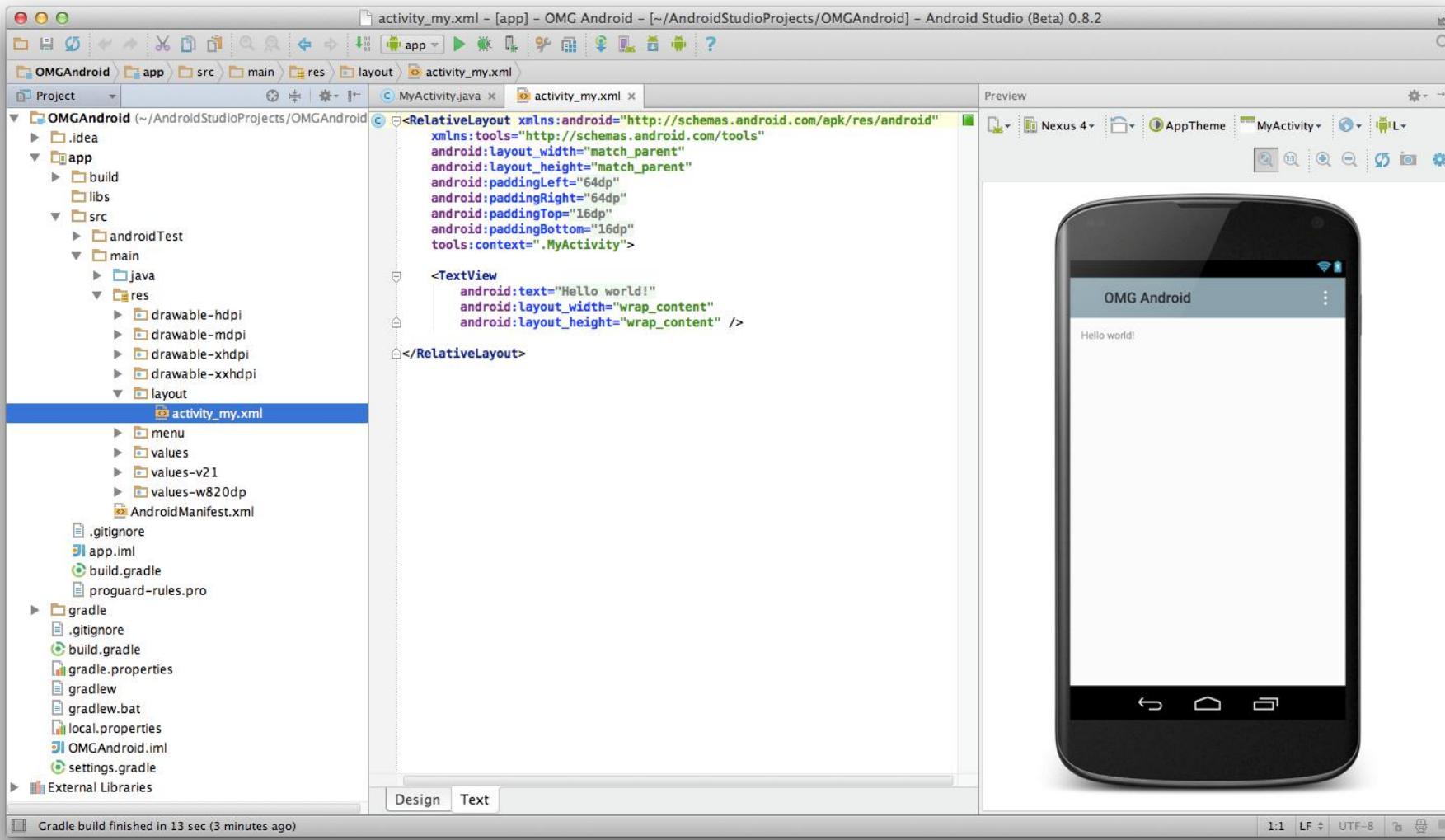


- Each Android app runs in its own security sandbox (VM, minimizes complete system crashes)
- Android OS multi-user Linux system
- Each app is a different user (assigned unique Linux ID)
- Access control: only process with the app's user ID can access its files



# Android Development Environment

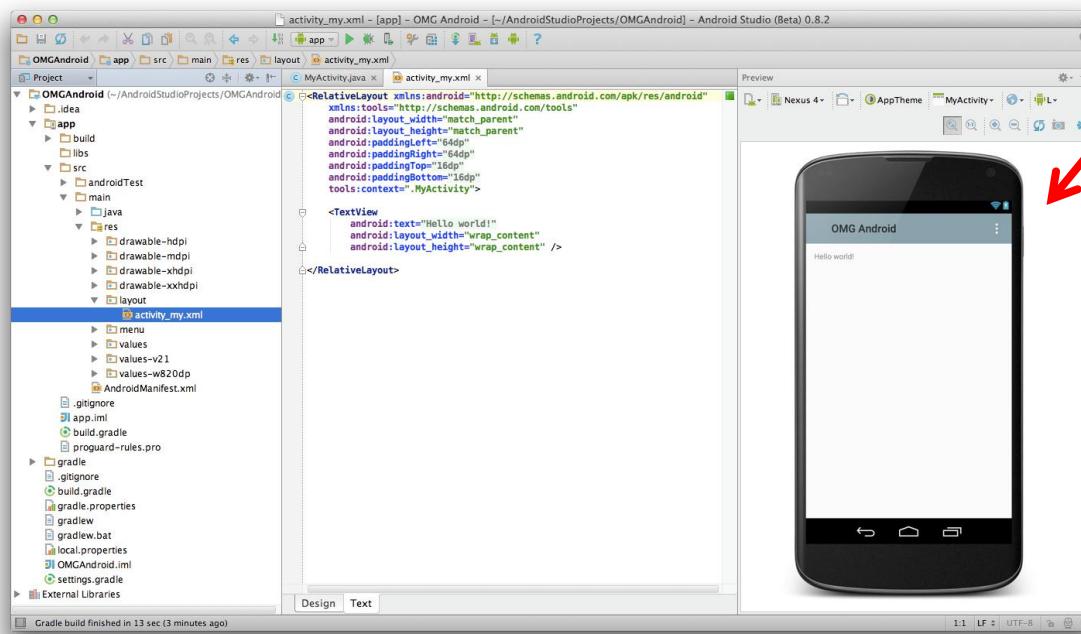
# Android Studio Layout



# Where to run Android Apps

- Android app can run on:
  - Real phone (or device)
  - Emulator (software version of phone)

Emulated Phone in  
Android Studio



# Run Android App on Real Phone

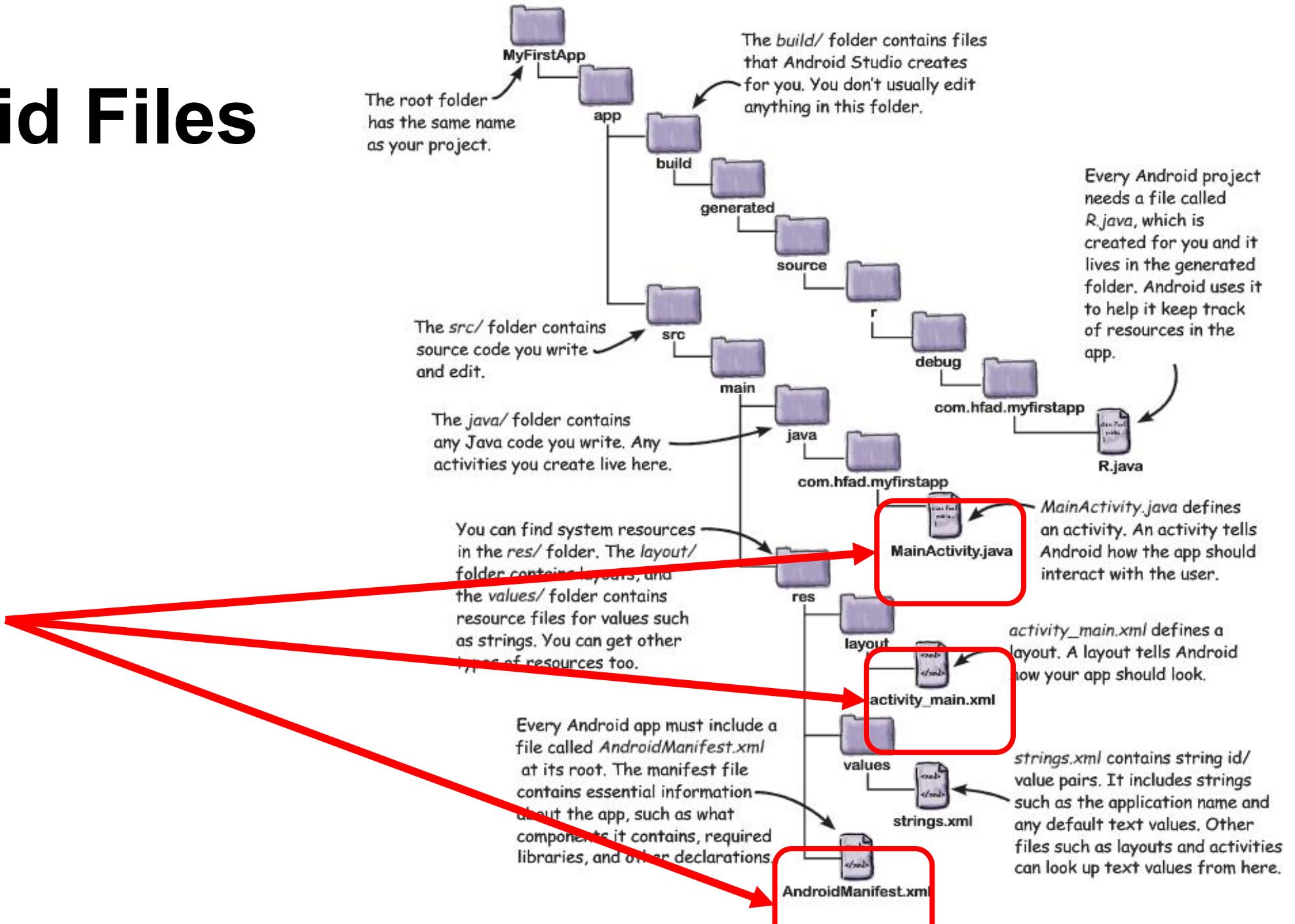
- Computer side: install Android Debug Bridge (ADB)
- Phone side: enable USB debugging
- checkout lab0

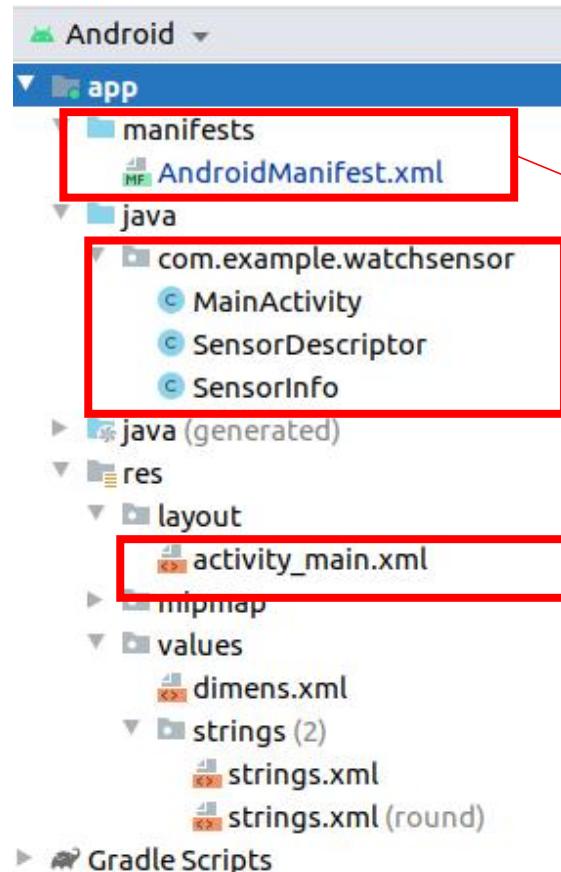


# Our First Android App

# Android Files

3 Main Files to Write Android app





Manifest

Java Program

Layout

# 3 Files in “Hello World” Android Project

- **Activity\_my.xml:** XML file specifying screen layout
- **MainActivity.Java:** Java code to define behavior, actions taken when button clicked (intelligence)
- **AndroidManifest.xml:**
  - Lists all screens, components of app
  - Analogous to a table of contents for a book
  - E.g. Hello world program has 1 screen, so AndroidManifest.xml has 1 item listed
  - App starts running here (like main( ) in C)
- **Note:** Android Studio creates these 3 files for you



# Execution Order

Start in **AndroidManifest.xml**  
Read list of activities (screens)  
Start execution from Activity  
tagged Launcher

Create/execute activities  
(declared in java files)  
E.g. **MainActivity.Java**

Format each activity using layout  
In XML file (e.g. **Activity\_my.xml**)

# Inside “Hello World” AndroidManifest.xml

package  
name  
Android  
Version

Activity List

This file is written using xml namespace and tags and rules for android

```
<?xml version="1.0"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.commonsware.android.skeleton"
    android:versionCode="1"
    android:versionName="1.0">

    <application>
        <activity
            android:name="Now"
            android:label="Now">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

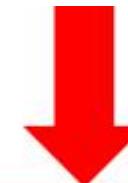
One activity (screen)  
designated LAUNCHER.  
The app starts running here

# Execution Order

Start in **AndroidManifest.xml**  
Read list of activities (screens)  
Start execution from Activity  
tagged Launcher



Create/execute activities  
(declared in java files)  
E.g. **MainActivity.Java**



Format each activity using layout  
In XML file (e.g. **Activity\_my.xml**)

# Example Activity Java file (E.g. MainActivity.java)

Package Declaration

```
package com.commonware.empublite;
```

Import needed classes

```
import android.app.Activity;
import android.os.Bundle;
```

Inherits from the  
Android Activity Class

```
public class EmPubLiteActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

App initialization:  
onCreate()

Use screen layout in  
file main.xml

# Execution Order

Start in **AndroidManifest.xml**  
Read list of activities (screens)  
Start execution from Activity  
tagged Launcher



Create/execute activities  
(declared in java files)  
E.g. **MainActivity.Java**



Format each activity using layout  
In XML file (e.g. **Activity\_my.xml**)

Declare Layout

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context=".EmPubLiteActivity">  
  
    <TextView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_centerHorizontal="true"  
        android:layout_centerVertical="true"  
        android:text="@string/hello_world"/>  
  
</RelativeLayout>
```

Add Widgets

Widget  
properties



# Simple XML file Designing UI

After choosing the layout, then widgets added to design UI

- XML Layout files consist of:
  - UI components (boxes) called **Views**
  - Different types of views. E.g
    - **TextView**: contains text,
    - **ImageView**: picture,
    - **WebView**: web page
  - **Views** arranged into layouts or **ViewGroups**

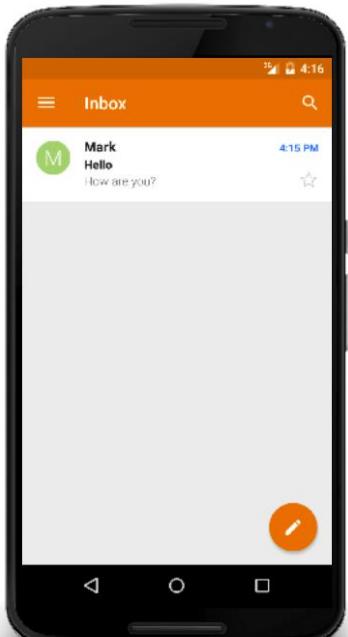
# Android Software Components

# Consider

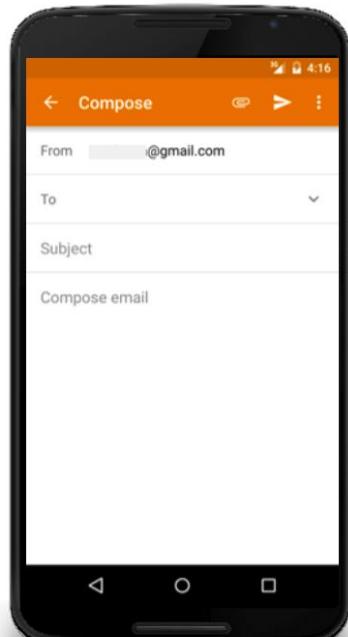
- What is a mobile app made of?
  - graphical UI: buttons, texts, video etc
  - play sounds?
  - services: sensors, locations, etc
  - more
- If one module fails, we don't want to quit entire app
  - network is disconnected and fails to download audio, but we still want to play local resources
  - Encapsulation

# Activity

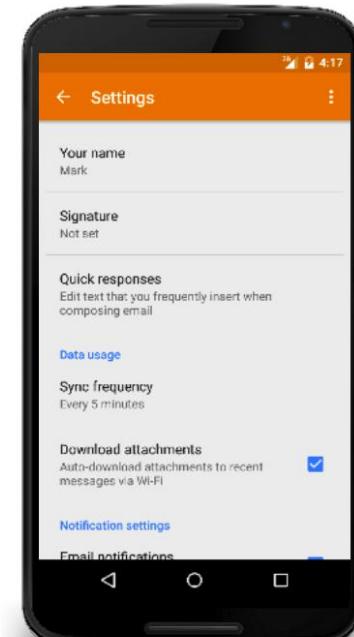
- Conceptually, an Activity shows a single screen of your application
- In other words, an App really is a collection of related Activities



Messages Activity



Compose Activity



Settings Activity

# Activity

- Consider each Activity both a screen and a feature
- Apps can activate Activities in other Apps

# Service

- A Service is a component that runs in the background to perform long-running operations
- A Service has no UI
- Examples of Services:
  - Playing music in background
  - Gathering GPS data
  - Downloading a data set from the server

# Intent

- An Intent is a message that requests an action from another component of the system
- This includes the “please start up your App” Intent that the system sends when a user clicks on your App icon

# Broadcast Receiver

- A Broadcast Receiver responds to system-wide announcements (which are manifested as Intents)
- System status information is delivered this way
  - e.g., device turned on, screen off, low battery, phone call incoming, etc.
- Broadcast Receivers typically don't have a UI, but could have a status bar icon

# Connected Apps

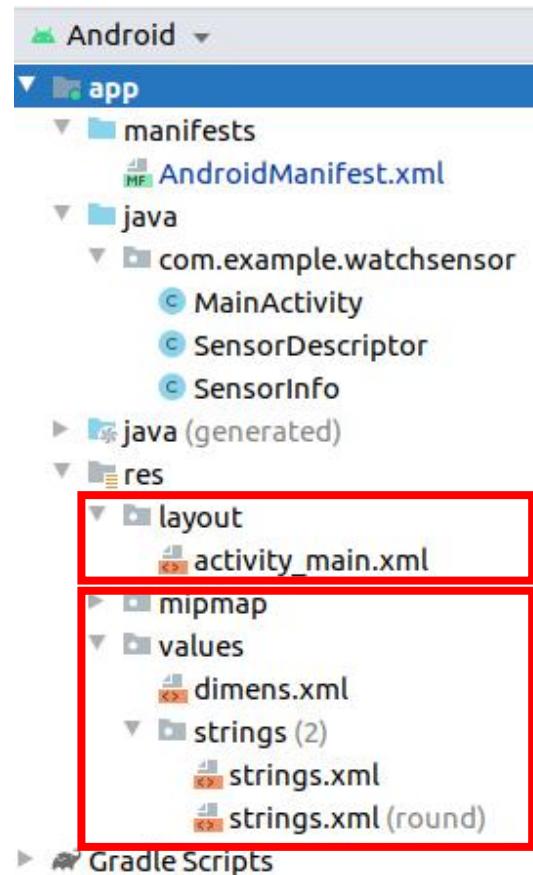
- Due to the component nature of Apps (made up of Activities, Services, etc.), it is easy to build features of your App using existing system components
- For example, if your App needs to take a picture, you can query the Camera Activity to handle that request and return the resulting image

# Put them all together

- If an App is made up of all these disparate parts, what holds them all together?
- The `AndroidManifest.xml` file!
  - Sets up all permissions the user has to agree to (i.e. Internet, GPS, contacts, etc.)
  - Declares the API level of the App
  - Requests hardware features needed
  - Needed libraries
  - Which Activities are part of this App

# Mobile Application Architecture (Continued)

# Designing UIs in Android

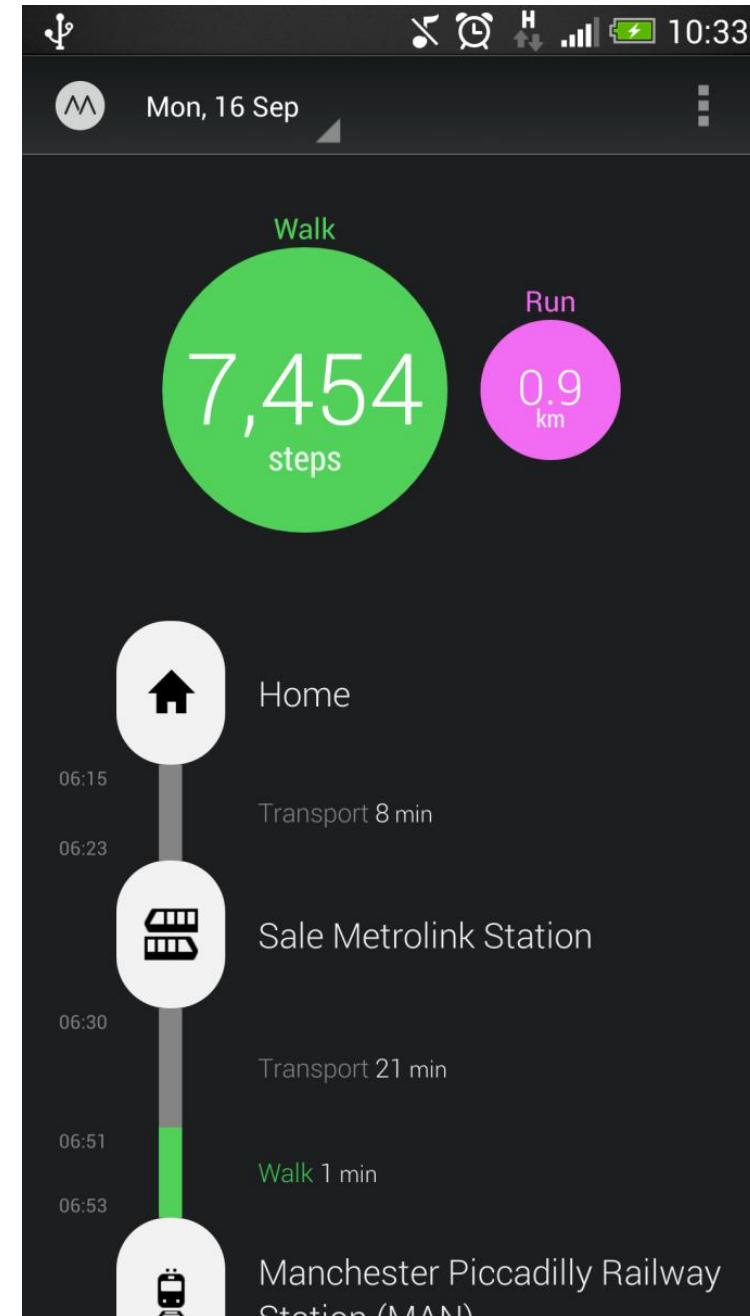


UI Design

Other stuff

# Why use XML?

- Question: Can we design UI using Java, without XML?
- Example: Shapes, colors can be changed in XML file without changing Java program
- UI designed using either:
  - Drag-and drop graphical (WYSIWYG) tool or
  - Programming Extensible Markup Language (XML)
- XML: Markup language, both human-readable and machine-readable“
- Purpose: separate UI from Logic



# Implement UI in xml

- Example 1

```
@Override  
public void onCreate(Bundle savedInstanceState)  
{  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
}
```

```
<?xml version="1.0" encoding="utf-8"?>  
<TextView xmlns:android="http://schemas.android.com/apk/res/android"  
        android:id="@+id/textview"  
        android:layout_width="fill_parent"  
        android:layout_height="fill_parent"  
        android:text="@string/hello"/>
```

# Implement UI in Java

```
@Override  
public void onCreate(Bundle savedInstanceState)  
{  
    super.onCreate(savedInstanceState);  
    TextView tv = new TextView(this);  
    tv.setText("hello");  
    LinearLayout ll = new LinearLayout(this);  
    ll.addView(tv);  
    setContentView(ll);  
}
```

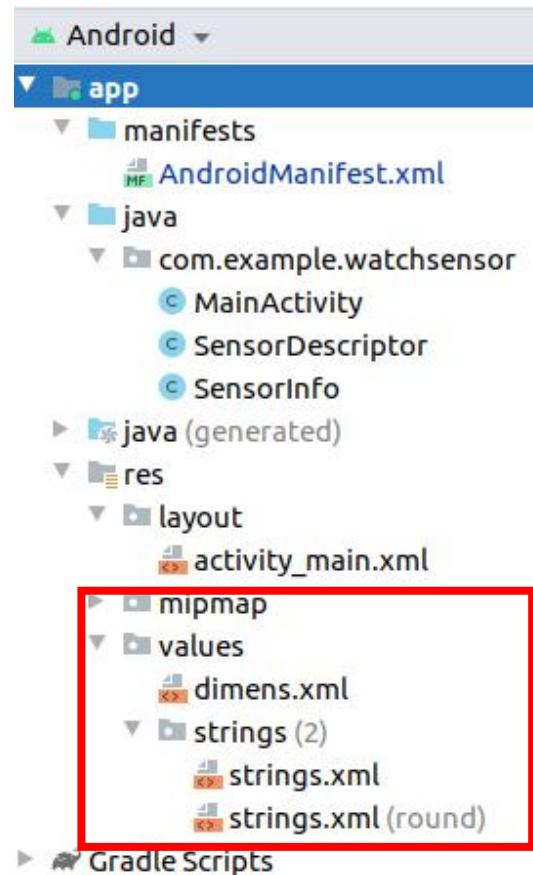
- The same UI can be implemented using java alone vs. java+xml
- Why is xml used and recommended?

## Benefits of using xml:

- Separation of logic from presentation.
  - This does help on larger projects when you need to refactor some code.
- The structure of XML correlates nicely to the structure of a user interface, i.e., a tree like structure.

# Other stuff

- Typically referred to as “assets,” anything that isn’t code is placed in the res/ folder
- String
- Music
- Images
- Some static data files



Other stuff

# Declaring Strings in Strings.xml

Can declare all strings in strings.xml

## String declaration in strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

<string name="app_name">EmPubLite</string>
<string name="hello_world">Hello world! </string>

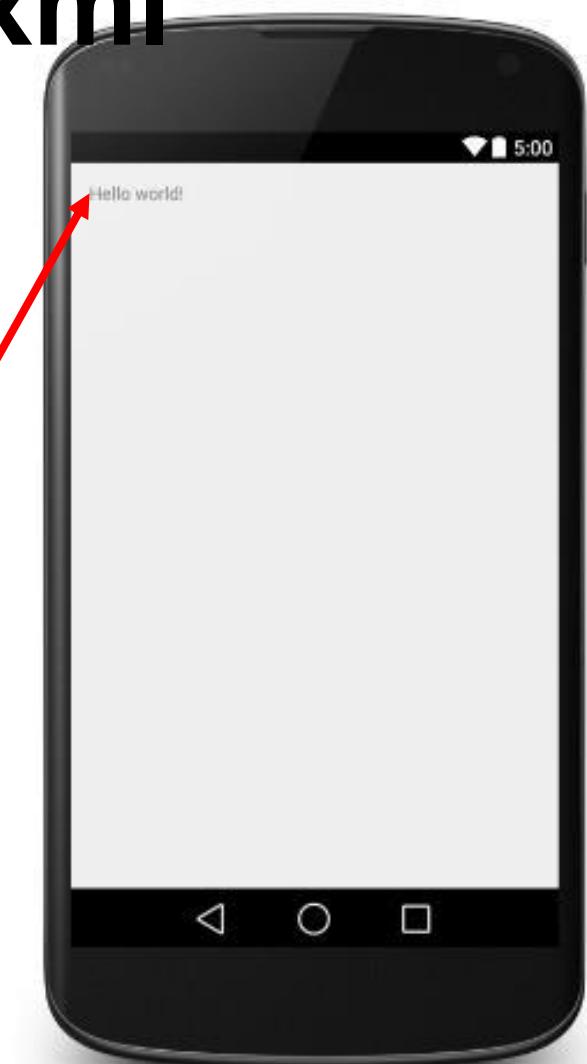
</resources>
```

Reference string in a main\_layout.xml files

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".EmPubLiteActivity">

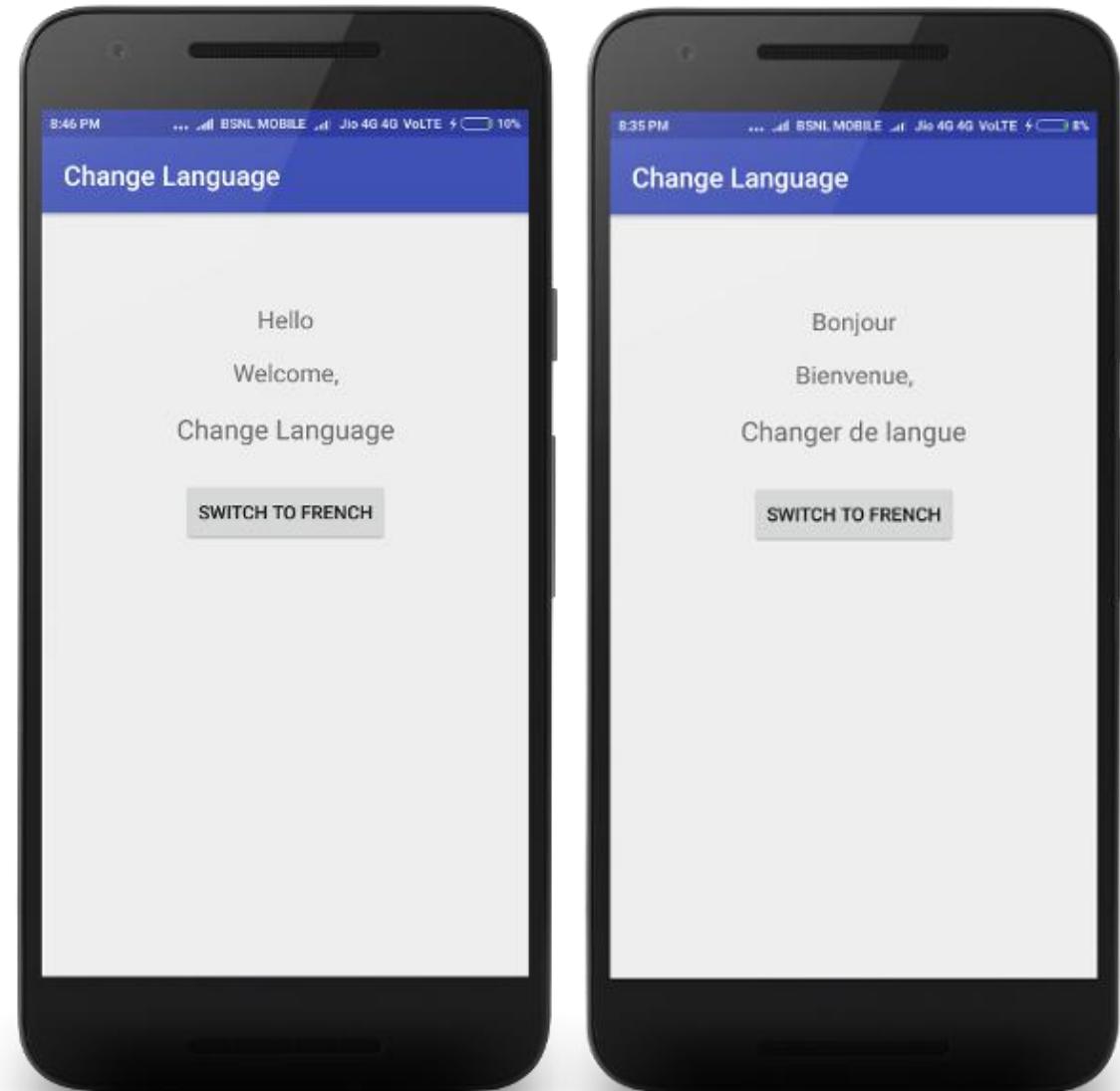
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world" />

</RelativeLayout>
```



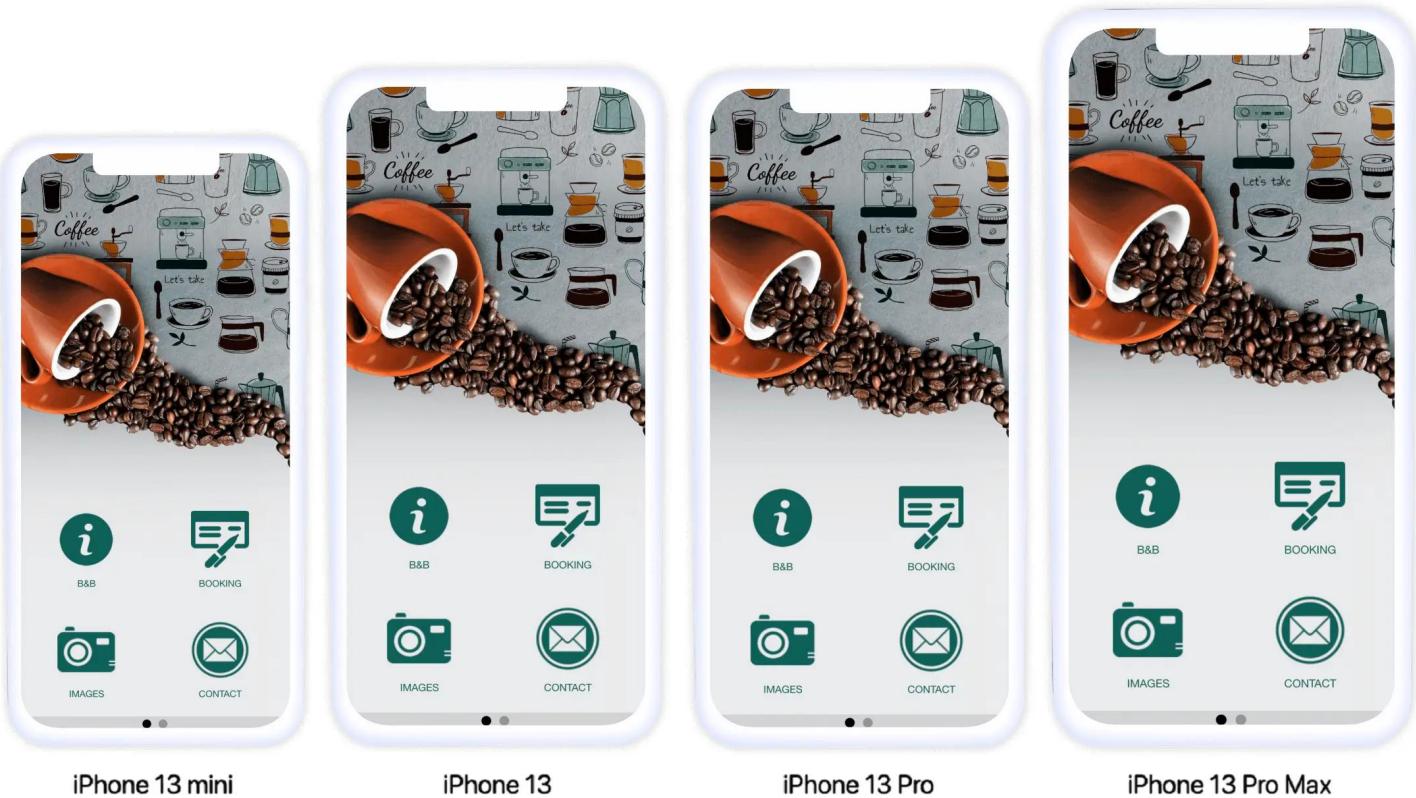
# Why bother to use an additional string.xml file?

- Think about the scenarios:
  - translate to a new language



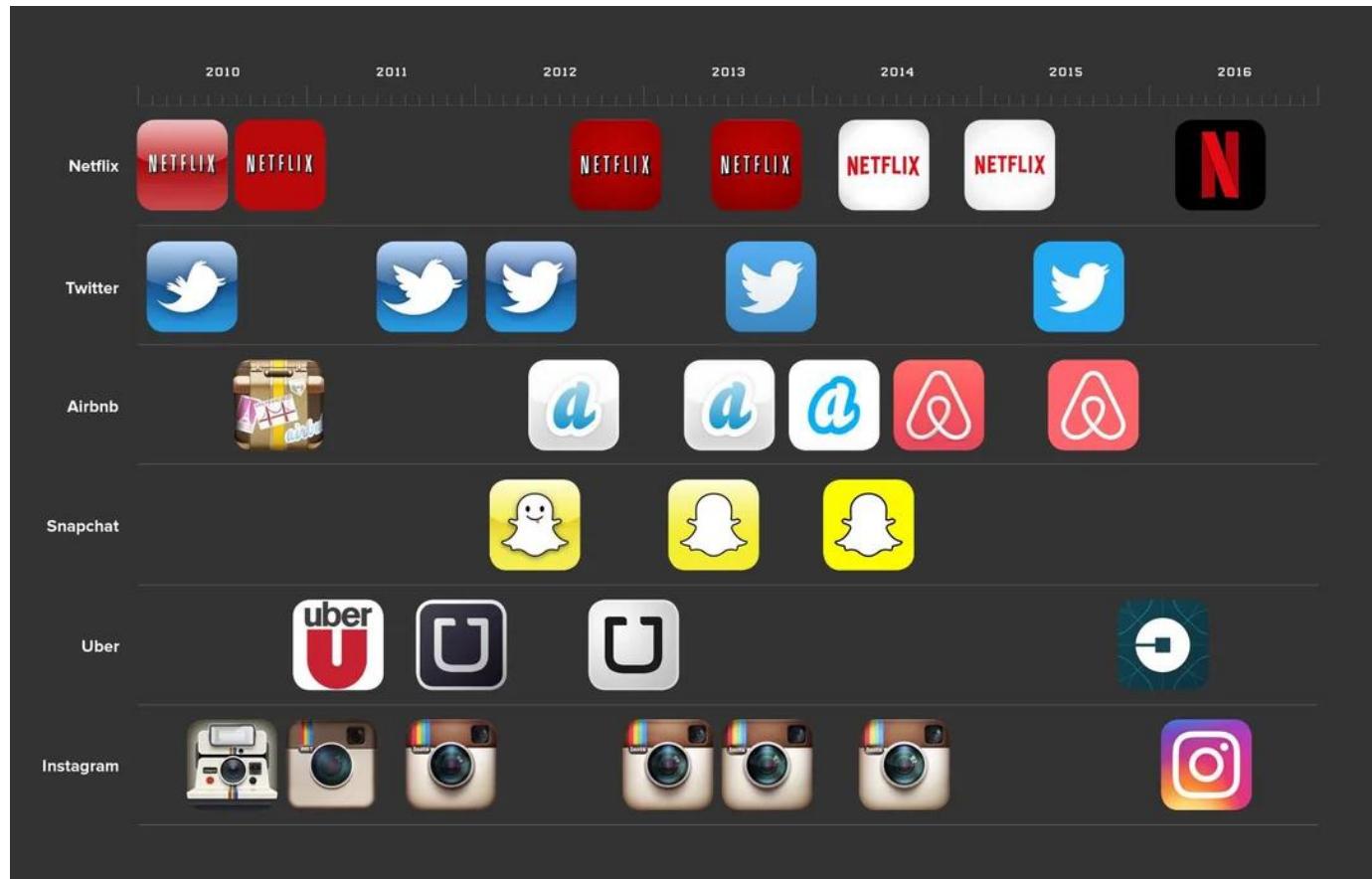
# Why dimens.xml? The same thing can be implemented in Java.

- Think about the scenarios:
  - adapt app to different screen sizes



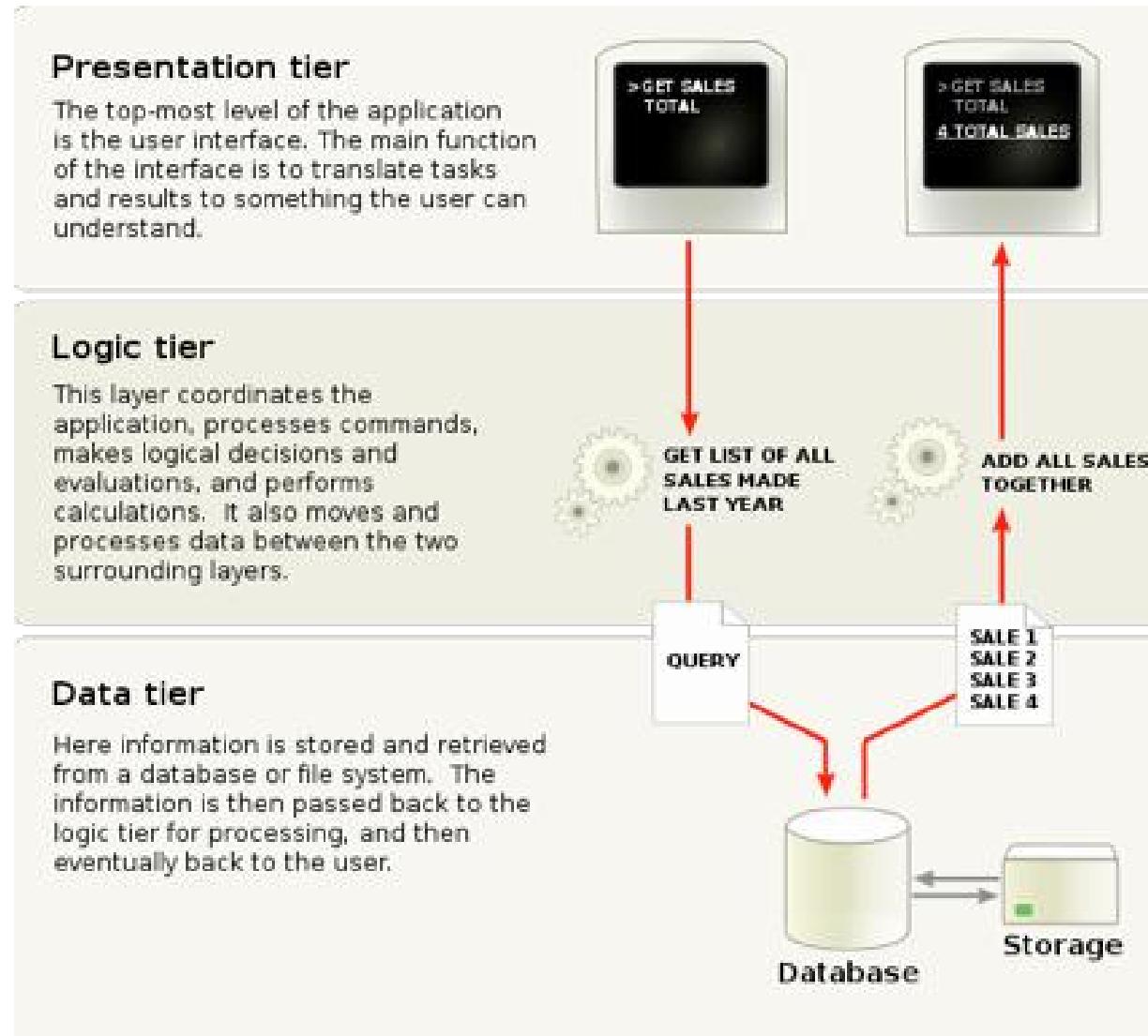
# Why put images aside?

- Think about the scenarios:
  - Logo changes
  - How to easily update logos for millions of phones?



# Design Philosophy

# Mobile Application Layered Design



# The Three-Tiered Architecture

- The concepts of the three-tiered architecture apply to many design scenarios
  - Keep the presentation separate so it's lightweight, easier to maintain, and can be tested separately
  - Keep the logic separate so you can change the logic as needed without having to change the presentation too much
  - Keep the data separate because you should NEVER build a system based on the current data values. Why?

# Client/Server Architecture

# Client/Server Architecture

- Mobile apps locally are great, but so much more can be done when apps are working with cloud services!
- The question is: how much processing / data should be done locally vs. in the cloud?

- Thick Client

- Business and some data services on the phone itself
- Good for apps that have to run “off the grid”
- Examples?



- Thin Client

- Most business and all data services on the server
- Good for apps that require phone services, but does require Internet connectivity.
- Examples?



- Which is better? Depends on the app and how it's used!

- computation vs communication. energy, time, bandwidth (Youtube)
- privacy vs accuracy (Nest smart thermostat)

# Pros and Cons of using server

- Pros
  - computation power (e.g. movies)
  - sharing (e.g., facebook, amazon )
- Cons
  - communication cost (e.g. battery, bandwidth)
  - privacy (e.g. waze)

# Challenges in Mobile Development

# Overview

- Limitations of the Wireless Network
  - heterogeneity of fragmented networks
  - frequent disconnections
  - limited communication bandwidth
- Limitations of the Mobile Devices
- Limitations of Battery

# Frequent Disconnections

- Handoff blank out (>1ms for most celluar)
- Drained battery disconnection
- Voluntary disconnection (turned off to preserve battery power, also off overnight)
- Roam-off disconnections

# Limited Wireless Bandwidth

- Orders of magnitude slower than fixed network
- Higher transmission bit error rates (BER)
- Mutual interference
  - Difficult to ensure Quality of Service (QoS)
- Limited communication bandwidth exacerbates the limitation of battery lifetime.

# Limitations of Mobile Devices

- Short battery lifetime and theft or destruction => unreliable
- Sometimes unavailable (disconnection or turned-off)
- Limited capability (display, computation, memory, input devices, and disk space)
- Device heterogeneity: phone, smartwatch, laptops, and other devices

# Battery limitations

- Most resources increasing exponentially except battery energy
- Strategies:
  - Energy harvesting: Energy from vibrations, moving humans
  - Scale content: Reduce image, video resolutions to save energy
  - Auto-dimming: Dim screen whenever user not using it. E.g. talking on phone

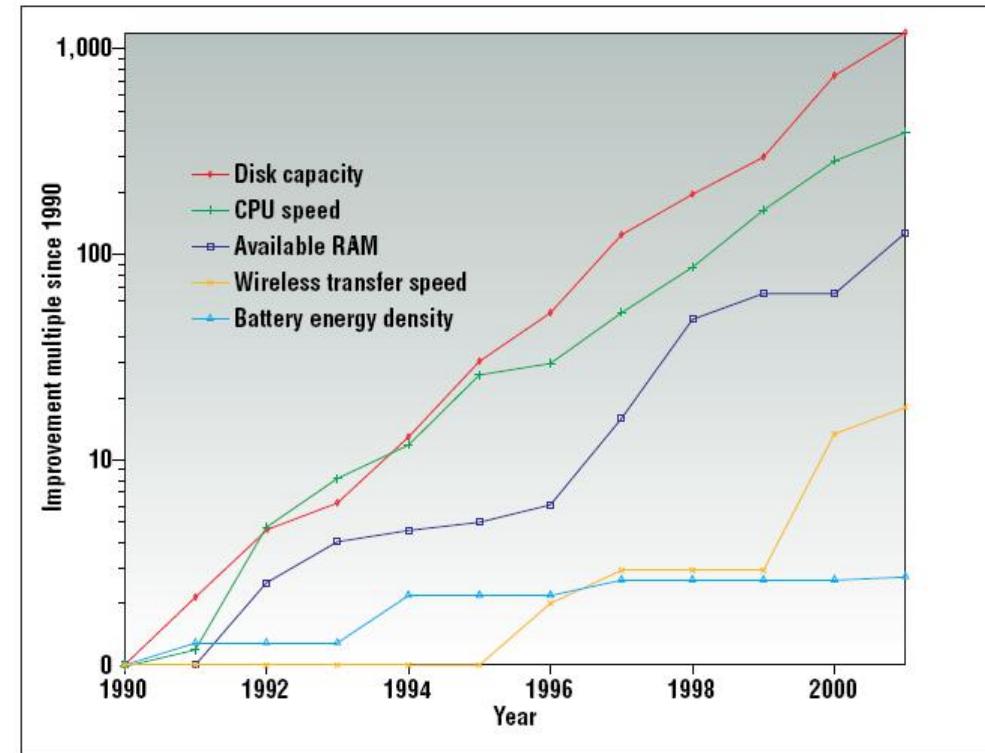


Figure 1. Improvements in laptop technology from 1990–2001.

# Mobile Application UI Design

Hua Huang

# Calendar Events

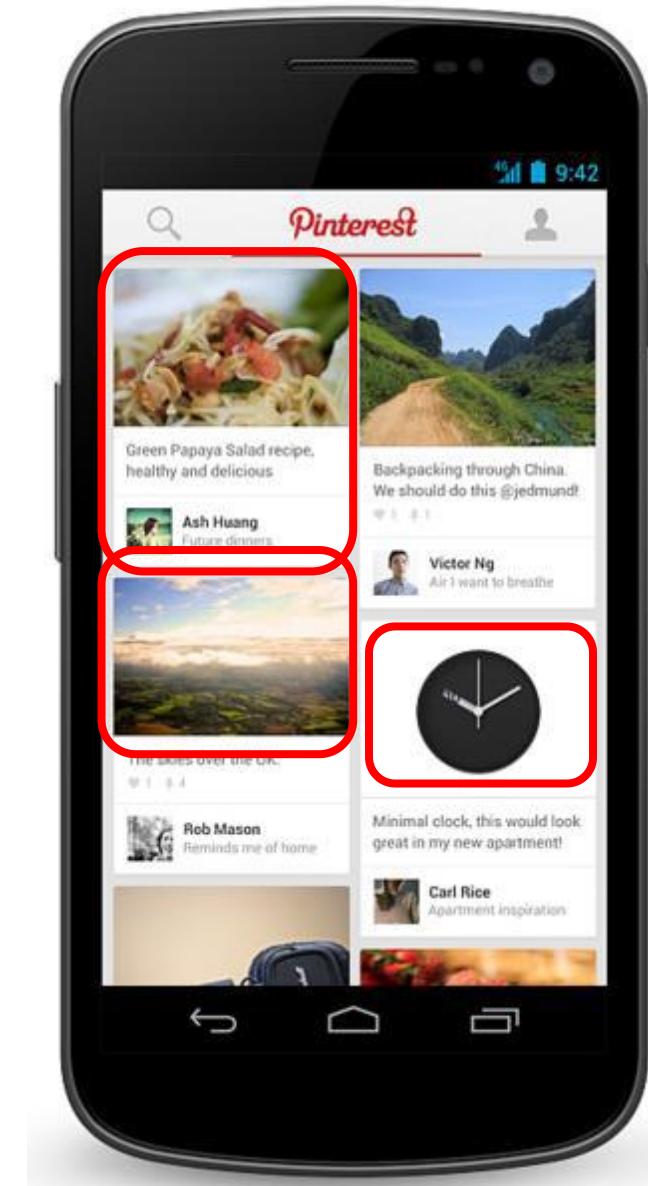
- Sept 24, Wednesday: class canceled due to travel
- Sept 26, first exam, covering the first eight lectures
  - Topics include Backgrounds, Android, UI, Activity, Service, Intents

# **Editing in Android Studio**

# Views

*Android UI design involves arranging views on a screen*

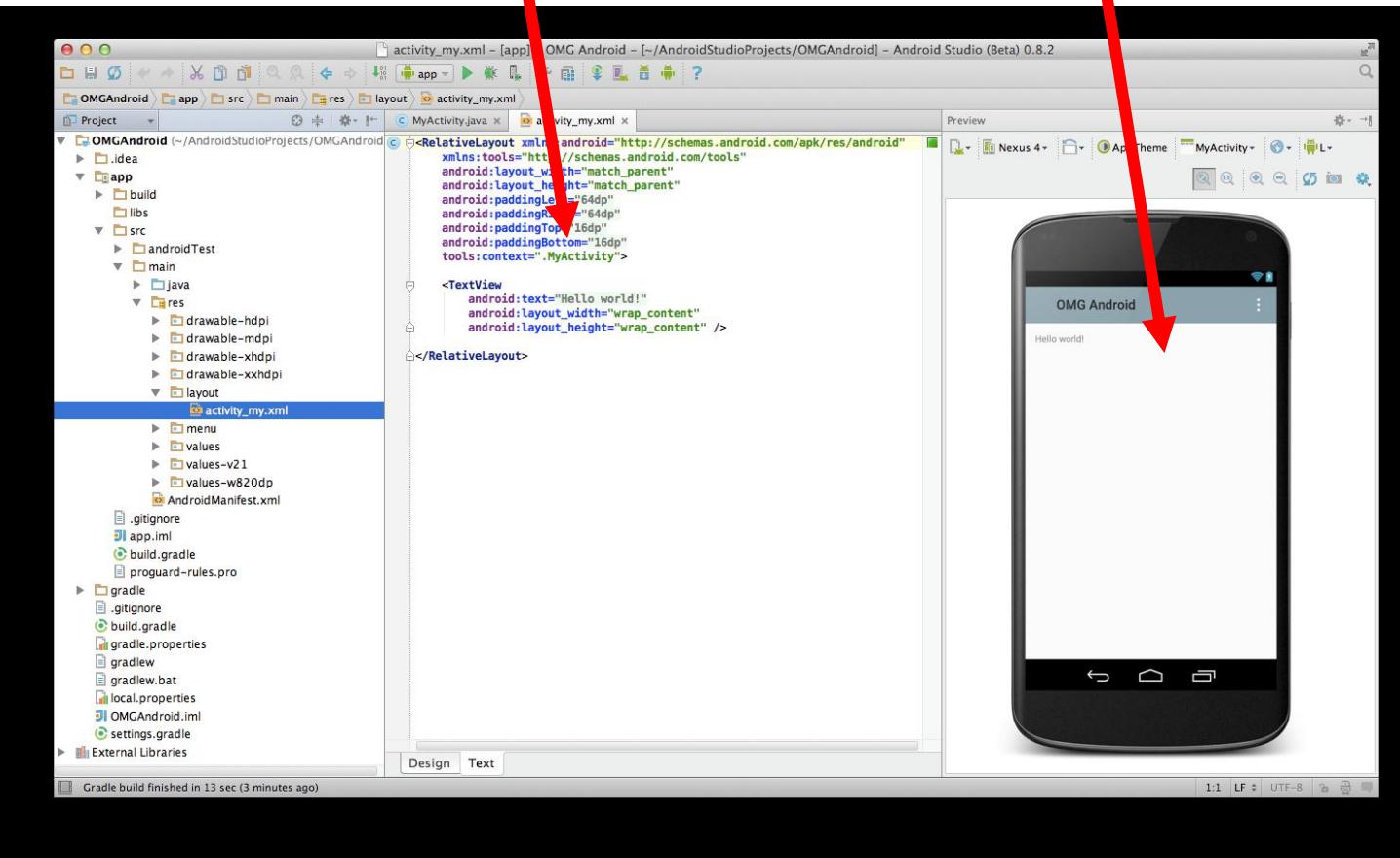
- **Views?** Rectangles containing texts, image, etc
- **Screen design:** Pick widgets, specify attributes (dimensions, margins, etc)



# Editting Android UI

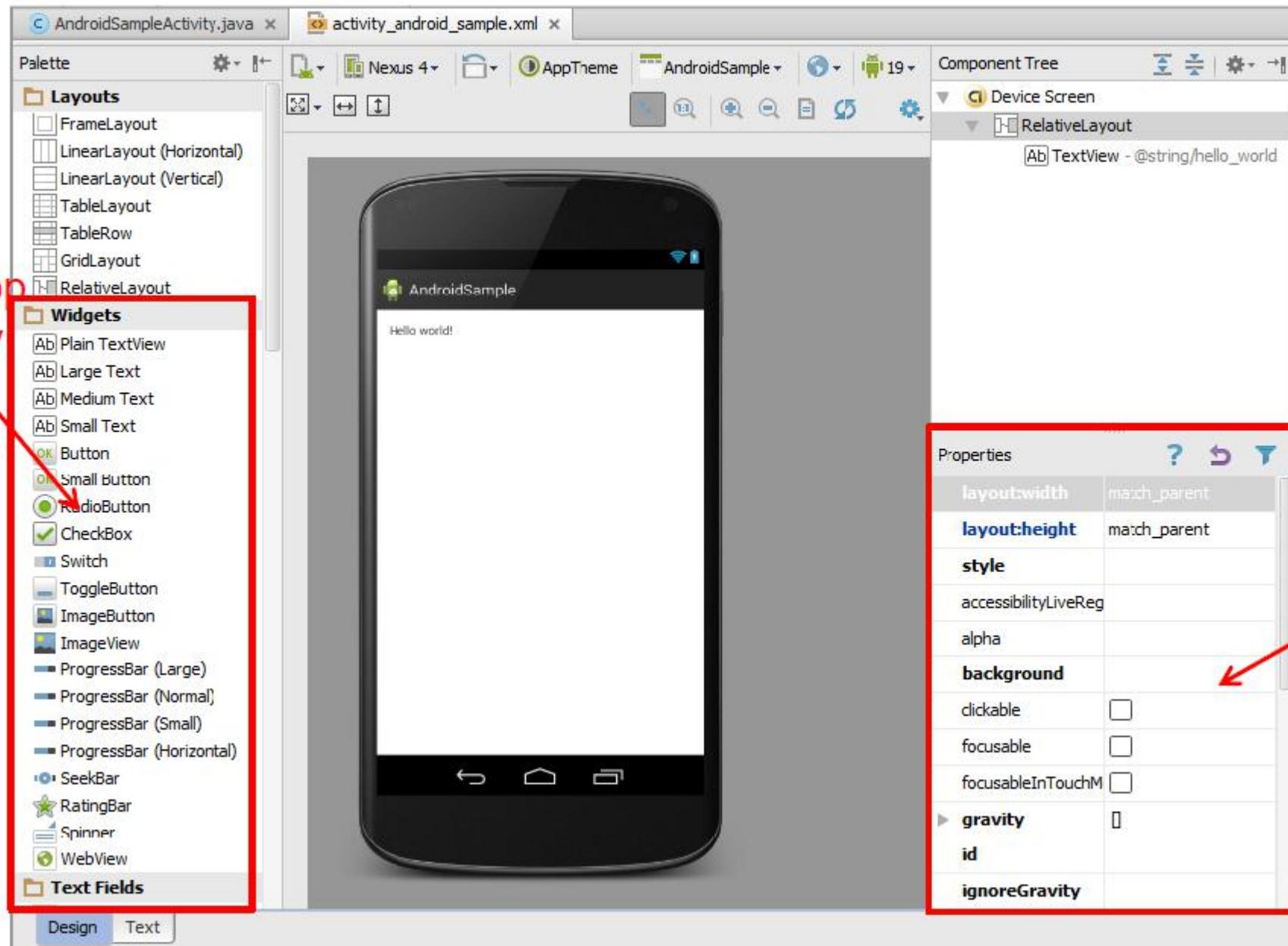
Can edit apps in:

- **Text View:** edit XML directly
- **Design View:** or drag and drop widgets unto emulated phone



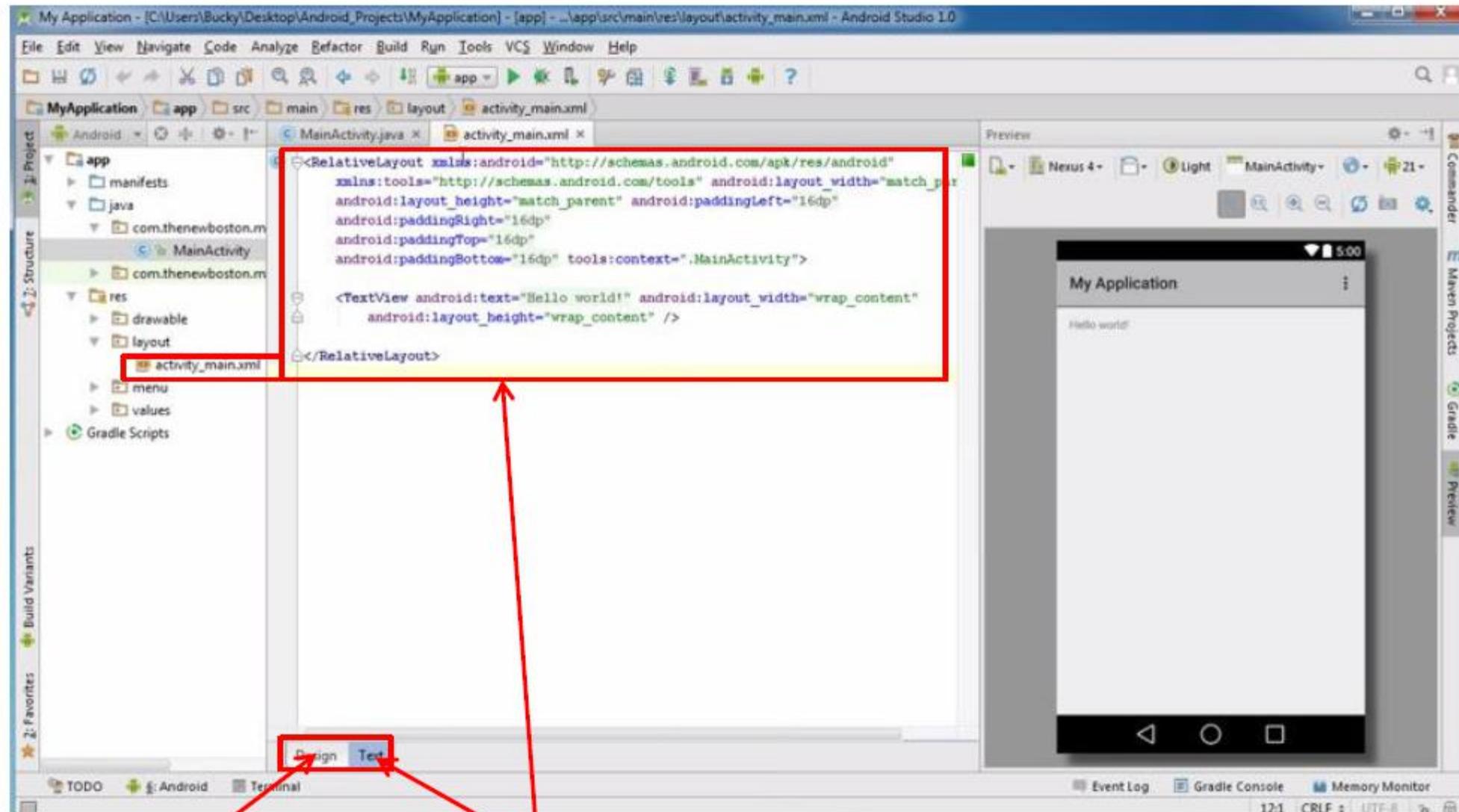
# Design Option 1: Drag and Drop Widgets

- Drag and drop widgets in Android Studio Design View
- Edit widget properties (e.g. height, width, color, etc)



# Design Option 2: Edit XML Directly

- **Text view:** Directly edit XML file defining screen (activity\_main.xml)
- **Note:** dragging and dropping widgets in design view auto-generates corresponding XML in Text view

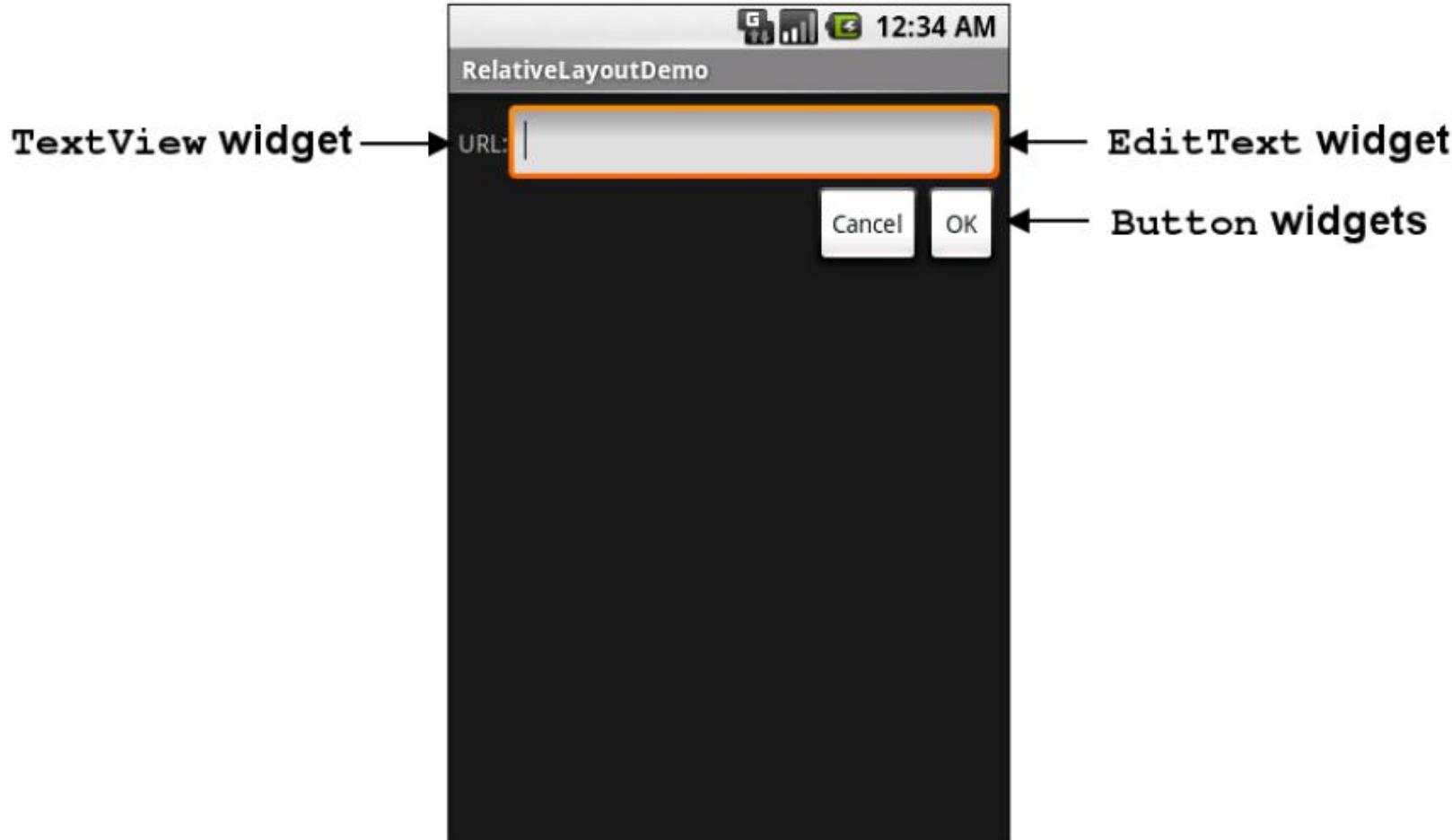


Drag and drop widget

Edit XML

# **Android Views**

- **TextView**: Text in a rectangle
- **EditText**: Text box for user to type in text
- **Button**: Button for user to click on



# General Form of Widget Declaration in XML

```
<widget type="E.g. TextView, button, EditText, etc"  
       List of attributes (e.g. format, width, length, etc)  
       .....  
       .....  
/>
```

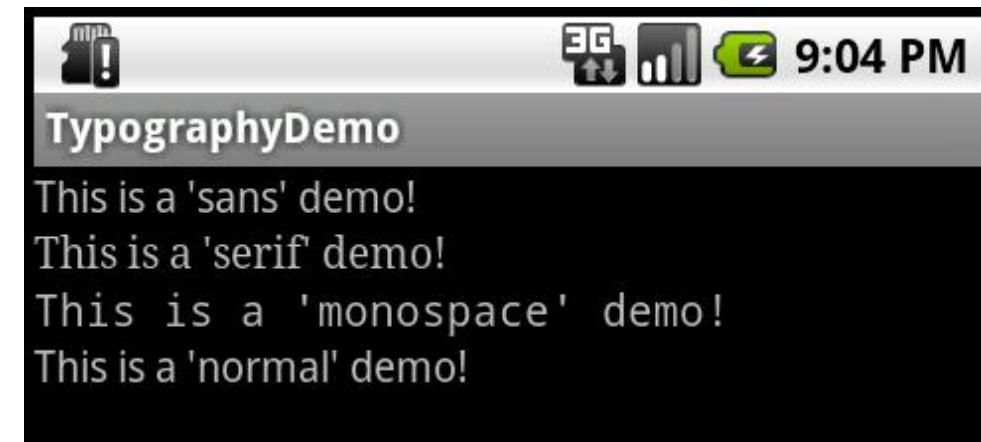
# recall

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_centerHorizontal="true"  
    android:layout_centerVertical="true"  
    android:text="@string/hello_world"/>
```

# TextView Widget

- Text in a rectangle
- Just displays text, no interaction

```
<TextView  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="This is a 'sans' demo!"  
    android:typeface="sans"  
/>
```

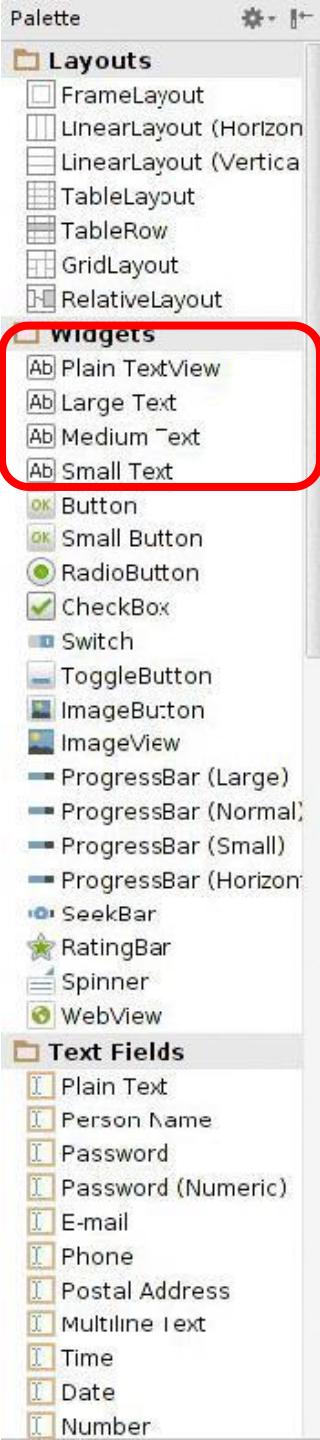
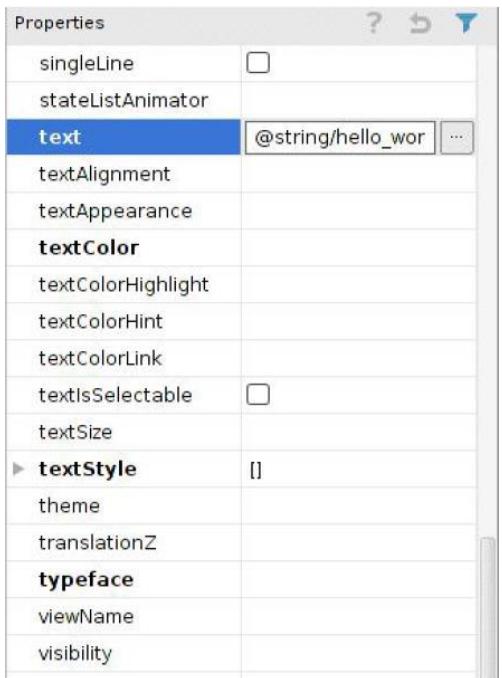


# TextView Widget

- **Common attributes:**
  - typeface (android:typeface e.g monospace), bold, italic,
  - (android:textStyle ),
  - text size,
  - text color (android:textColor e.g. #FF0000 for red),
  - width, height, padding,
  - background color
- Can also include links to email address, url, phone number,
- web, email, phone, map, etc

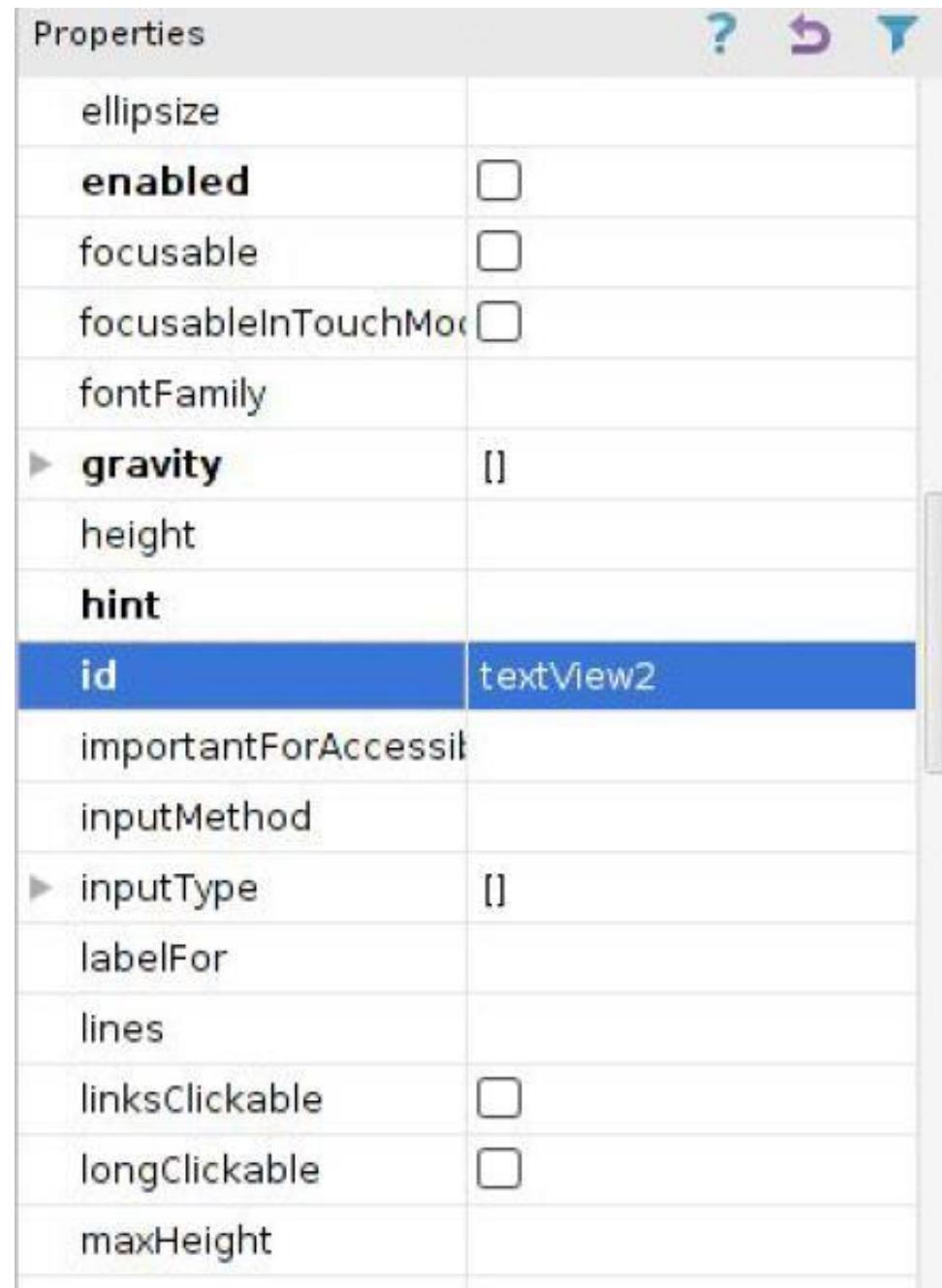
# TextView

- TextView widget is available in widgets palette in Android Studio Layout editor
  - Plain TextView, Large text, Medium text and Small text
- After dragging TextView widget in, edit properties



# Widget ID

- Every widget has ID, stored in **android:id** attribute
- Using Widget ID declared in XML, widget can be referenced, modified in java code



The screenshot shows the 'Properties' tab in the Android Studio IDE. The 'id' property is highlighted with a blue selection bar, and its value is set to 'textView2'. Other properties listed include 'ellipsize', 'enabled', 'focusable', 'focusableInTouchMode', 'fontFamily', 'gravity', 'height', 'hint', 'importantForAccessibility', 'inputMethod', 'inputType', 'labelFor', 'lines', 'linksClickable', 'longClickable', and 'maxHeight'. Each property has a corresponding input field or checkbox to its right.

Properties	
ellipsize	
<b>enabled</b>	<input type="checkbox"/>
focusable	<input type="checkbox"/>
focusableInTouchMode	<input type="checkbox"/>
fontFamily	
<b>gravity</b>	[]
height	
<b>hint</b>	
<b>id</b>	textView2
importantForAccessibility	
inputMethod	
<b>inputType</b>	[]
labelFor	
lines	
linksClickable	<input type="checkbox"/>
longClickable	<input type="checkbox"/>
maxHeight	

# Button Widget

- Clickable Text or icon on a Widget (Button)
- Examples on the right
- Appearance can be customized
- Declared as subclass of TextView so similar attributes (e.g. width, height, etc)

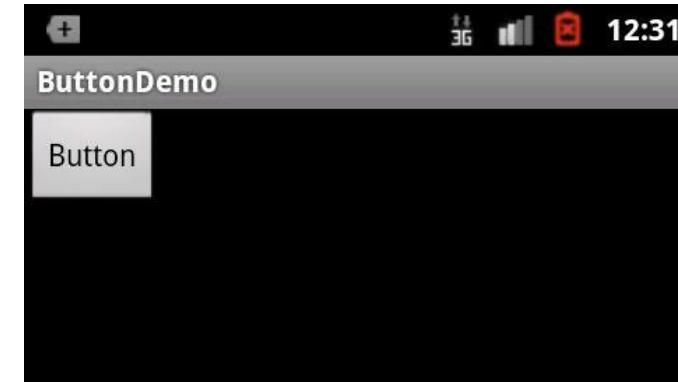
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button"/>

</LinearLayout>
```

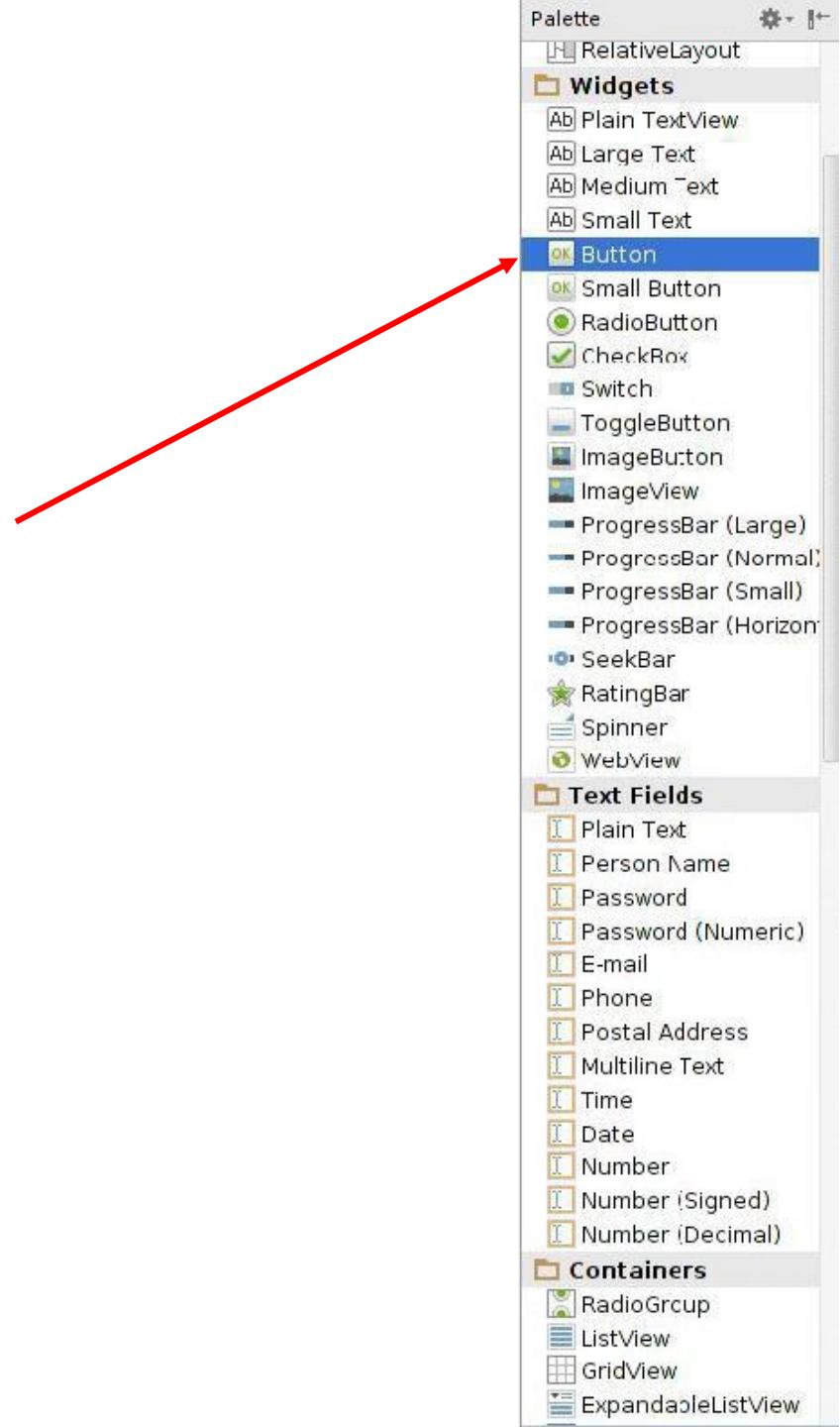


		DELETE
sin	cos	tan
In	log	!



# Button in Android Studio

- **Button** widget available in palette of Android Studio graphical layout editor
- Drag and drop button, edit its attributes



# Responding to Button Clicks

- May want Button press to trigger some action. How?

1. In XML file (e.g. Activity\_my.xml),  
set android:onClick attribute  
to specify method to be invoked

Activity\_my.xml

```
<Button  
    android:onClick="someMethod"  
    ...  
/>
```

2. In Java file (e.g. MainActivity.java)  
declare method/handler to take  
desired action

MainActivity.java

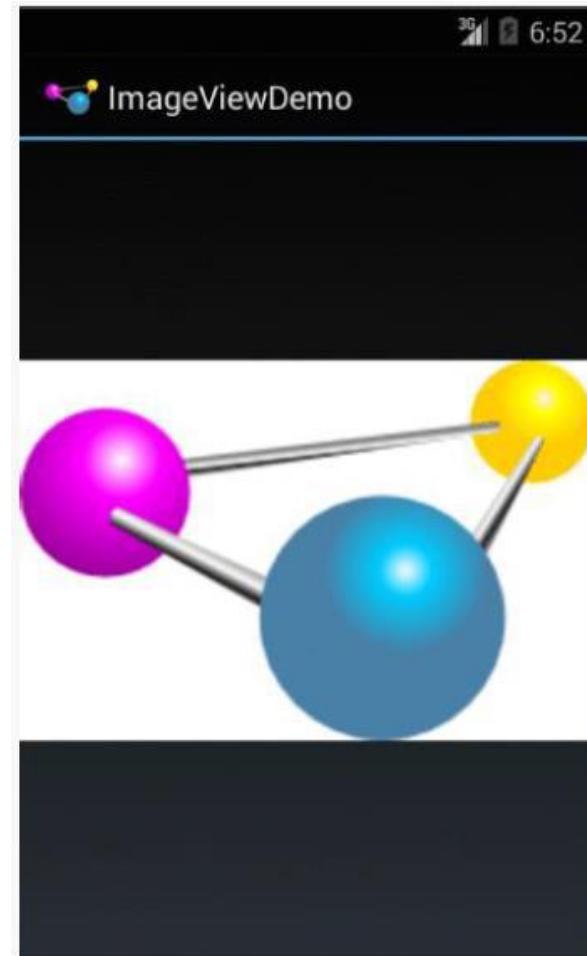
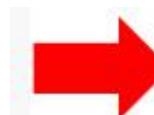
```
public void someMethod(View theButton) {  
    // do something useful here  
}
```

# Embedding Images: ImageView and ImageButton

- **ImageView:** display image (not clickable)
  - **ImageButton:** Clickable image
- 
- Use **android:src** attribute to specify image source in **drawable** folder (e.g. **@drawable/molecule**)

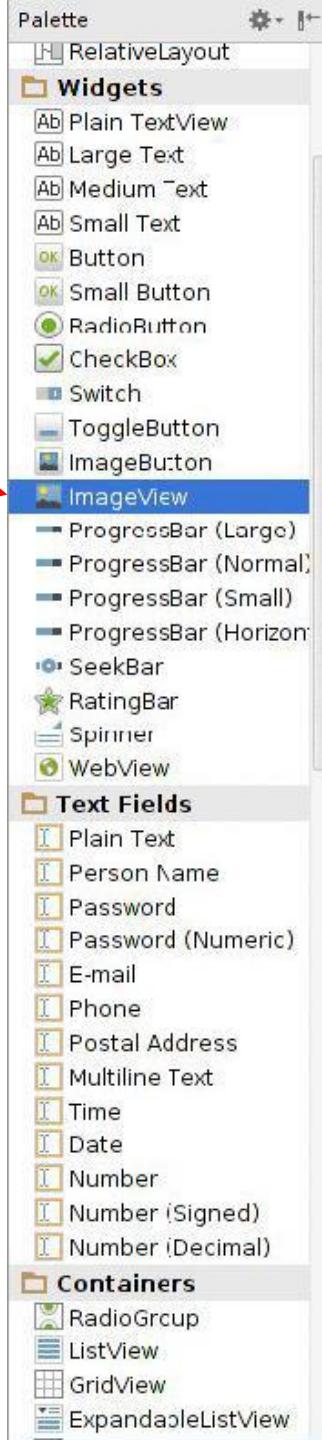
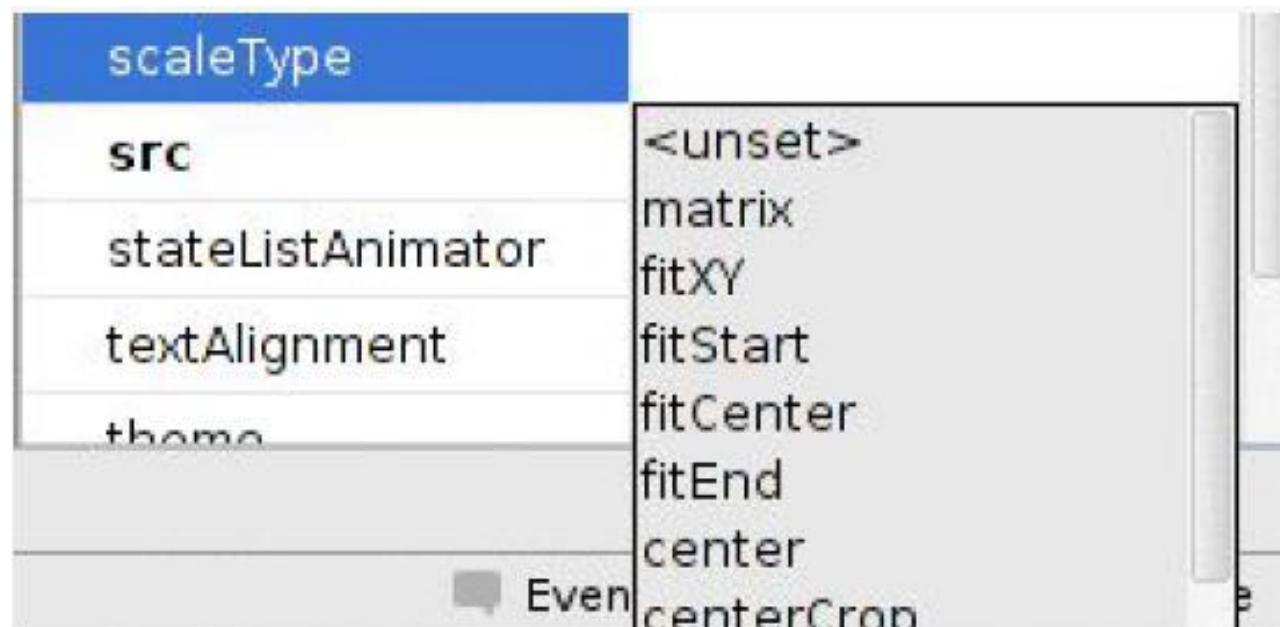
```
<?xml version="1.0" encoding="utf-8"?>
<ImageView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/icon"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:adjustViewBounds="true"
    android:src="@drawable/molecule"/>
```

File molecule.png in drawable/ folder

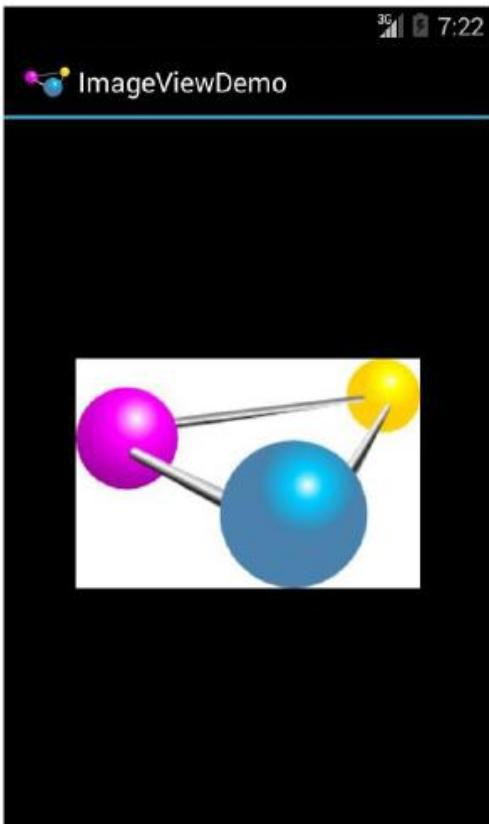


# ImageView in Widgets Palette

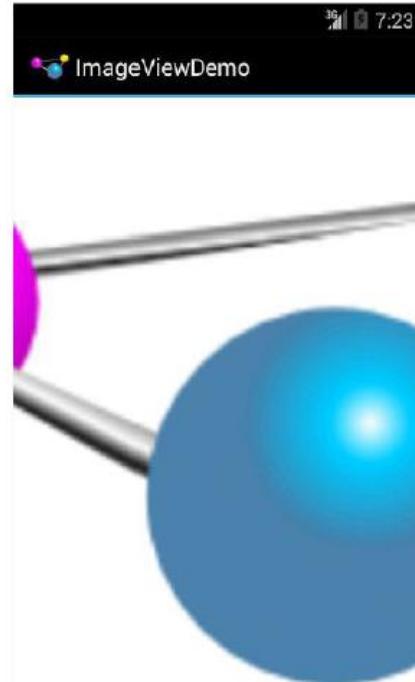
- Can drag and drop ImageView from Widgets Palette
- Use pop-up menus (right-click) to specify:
  - **src**: choose image to be displayed
  - **scaleType**: choose how image should be scaled



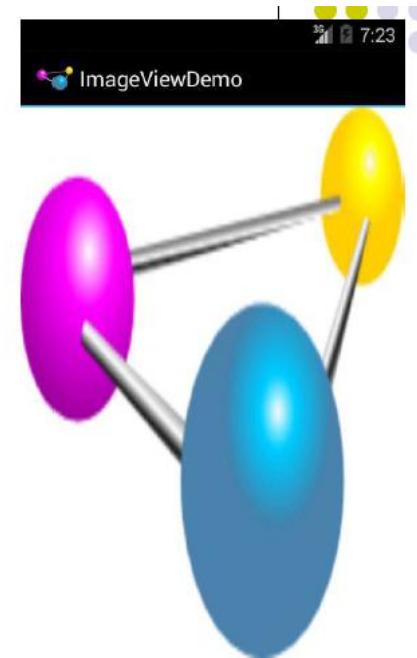
# Options for Scaling Images (scaleType)



**"center"** centers image  
but does not scale it



**"centerCrop"** centers  
image, scales it  
(maintaining aspect ratio) so  
that shorter dimension fills  
available space, and  
crops longer dimension

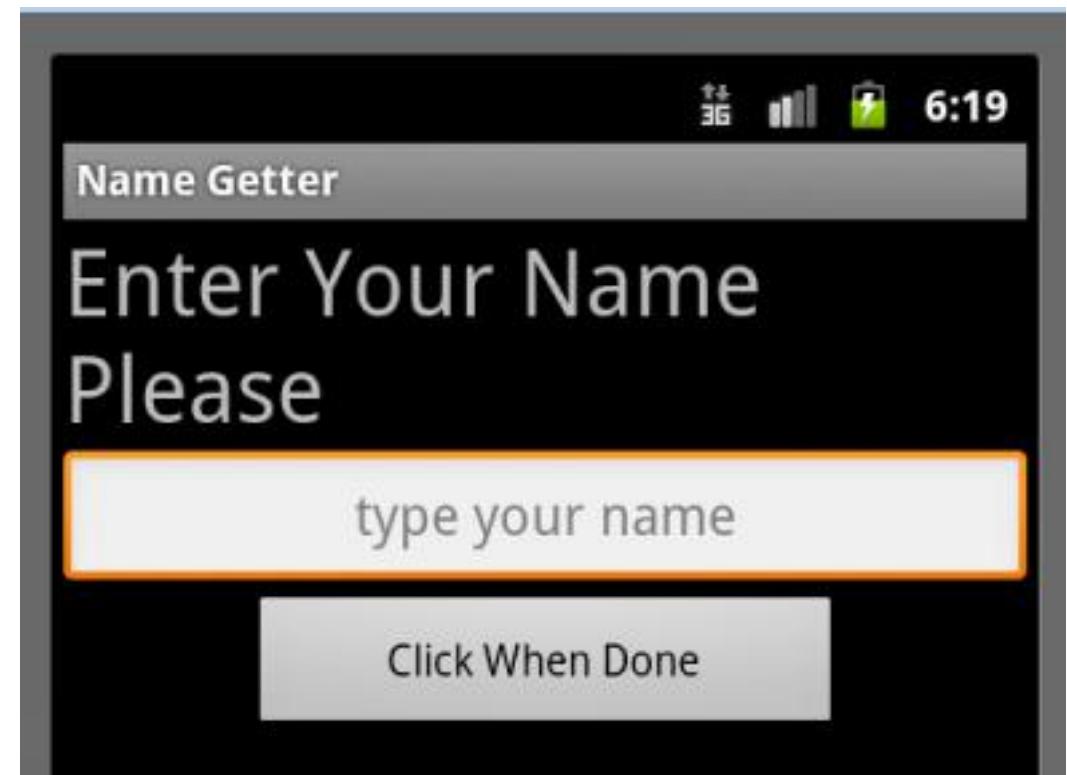


**"fitXY"** scales/distorts image  
to fit ImageView, ignoring  
aspect ratio

# EditText Widget

- Widget with box for user input

```
<EditText  
    android:id="@+id/edittext"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_gravity="center"  
    android:gravity="center"  
    android:inputType="textPersonName"  
    android:hint="type your name" />
```



- Text fields can have different input types
  - e.g. number, date, password, or email address
- android:inputType** attribute sets input type, affects
  - What type of keyboard pops up for user
  - E.g. if inputType is a number, numeric keyboard pops up

# EditText Widget in Android Studio Palette

A section of Android Studio palette has EditText widgets (or text fields)

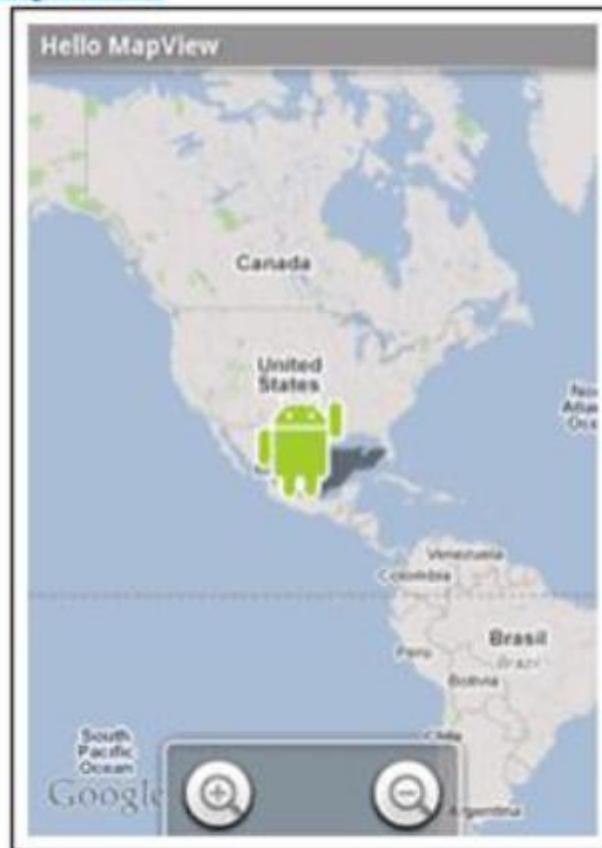
**Text Fields**  
Section of Widget palette

▼ inputType	[ ]
none	<input type="checkbox"/>
text	<input type="checkbox"/>
textCapCharacter	<input type="checkbox"/>
textCapWords	<input type="checkbox"/>
textCapSentences	<input type="checkbox"/>
textAutoCorrect	<input type="checkbox"/>
textAutoComplete	<input type="checkbox"/>
textMultiLine	<input type="checkbox"/>
textImeMultiLine	<input type="checkbox"/>
textNoSuggestion	<input type="checkbox"/>
textUri	<input type="checkbox"/>
textEmailAddress	<input type="checkbox"/>
textEmailSubject	<input type="checkbox"/>
textShortMessage	<input type="checkbox"/>
textLongMessage	<input type="checkbox"/>
textPersonName	<input type="checkbox"/>
textPostalAddress	<input type="checkbox"/>
textPassword	<input type="checkbox"/>
textVisiblePassword	<input type="checkbox"/>
textWebEditText	<input type="checkbox"/>
textFilter	<input type="checkbox"/>
textPhonetic	<input type="checkbox"/>
textWebEmailAddress	<input type="checkbox"/>
textWebPassword	<input type="checkbox"/>
number	<input type="checkbox"/>
numberSigned	<input type="checkbox"/>
numberDecimal	<input type="checkbox"/>
numberPassword	<input type="checkbox"/>
phone	<input type="checkbox"/>

**EditText  
inputType menu**

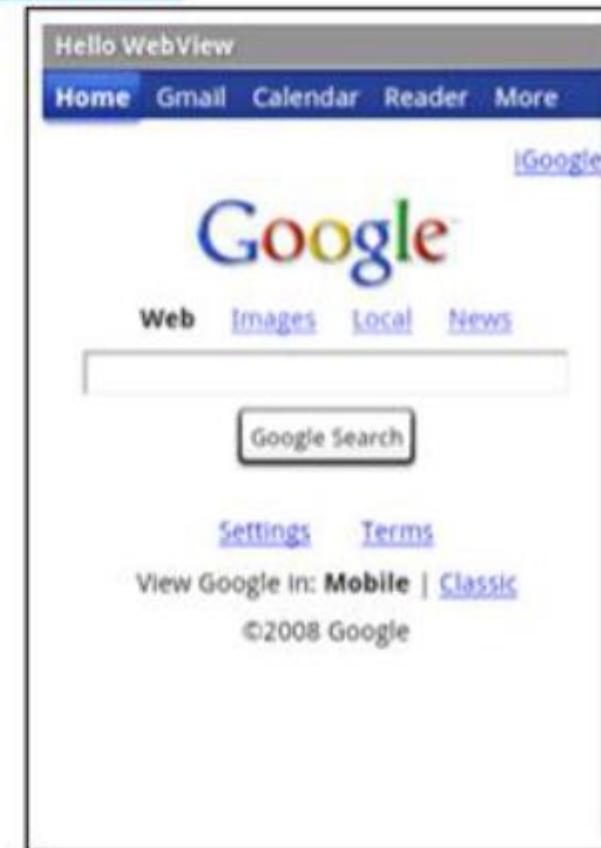
# Some Other Available Widgets

MapView



Rectangle that  
contains a map

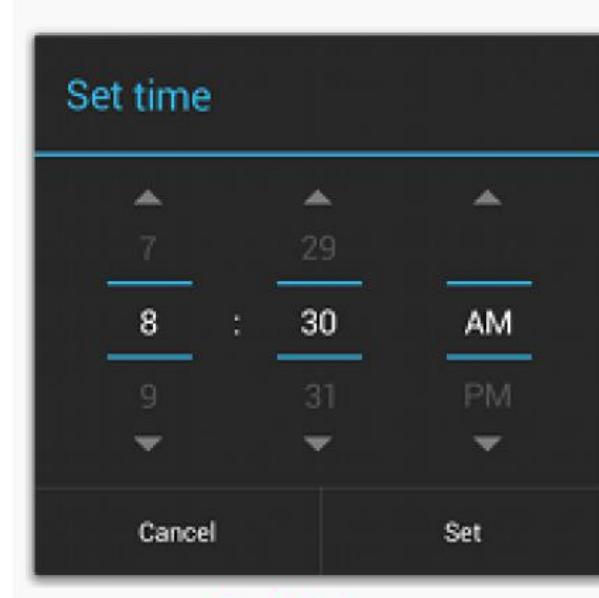
WebView



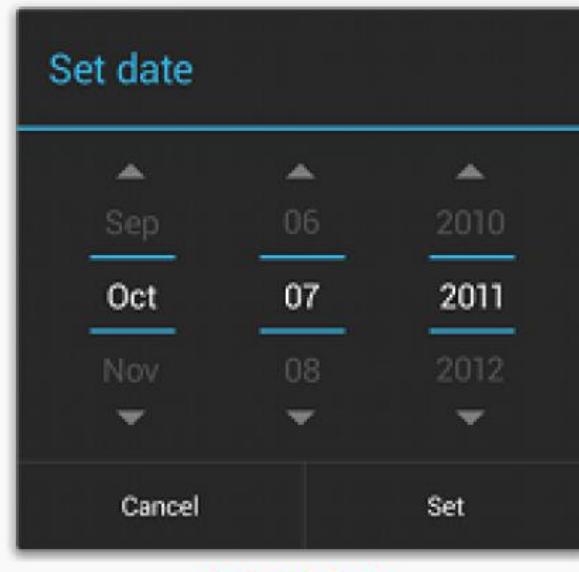
Rectangle that  
contains a web page

# Pickers

- **TimePicker:** Select a time
- **DatePicker:** Select a date
- Typically displayed in pop-up dialogs (**TimePickerDialog** or **DatePickerDialog**)



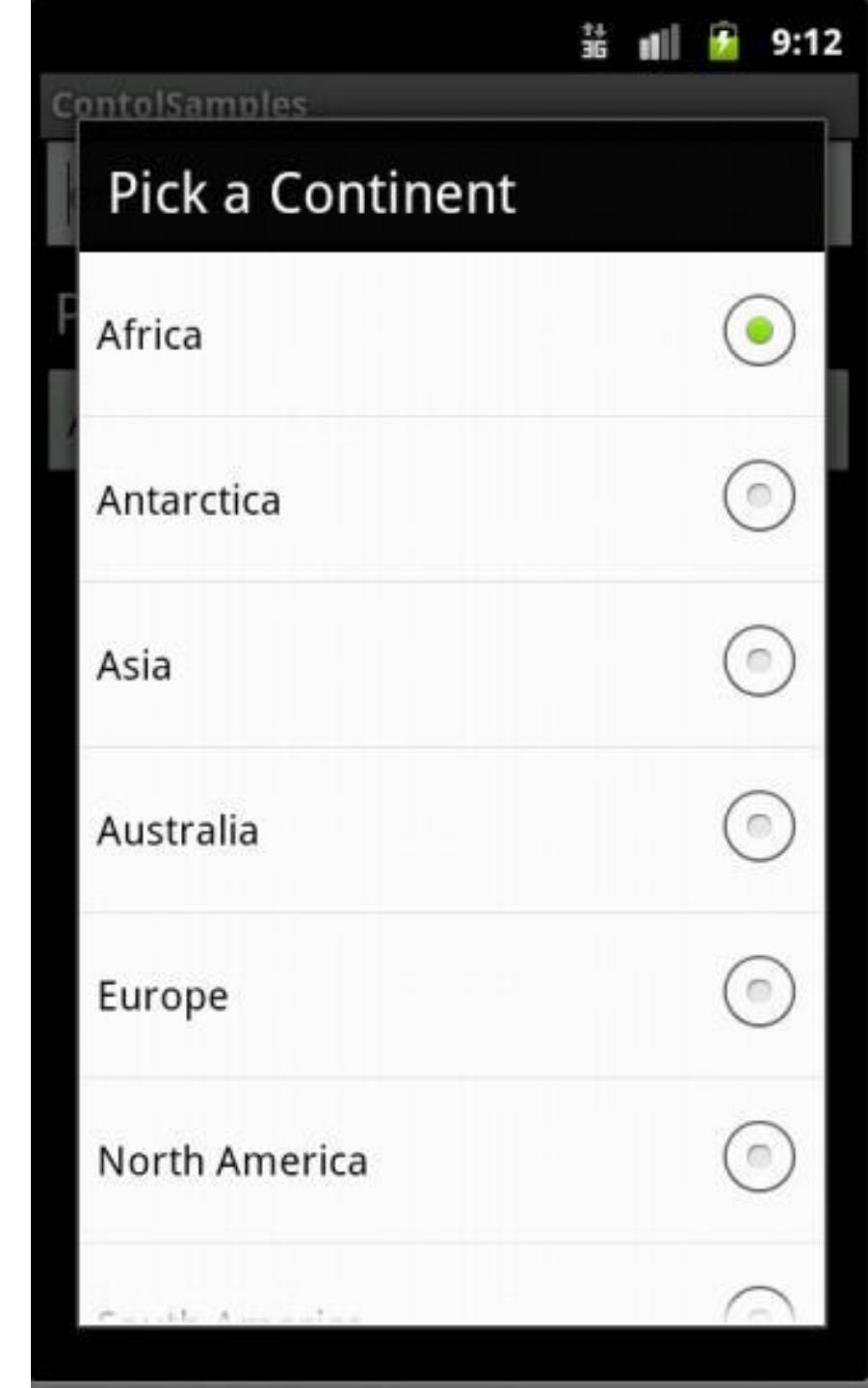
TimePicker



DatePicker

# Spinner Controls

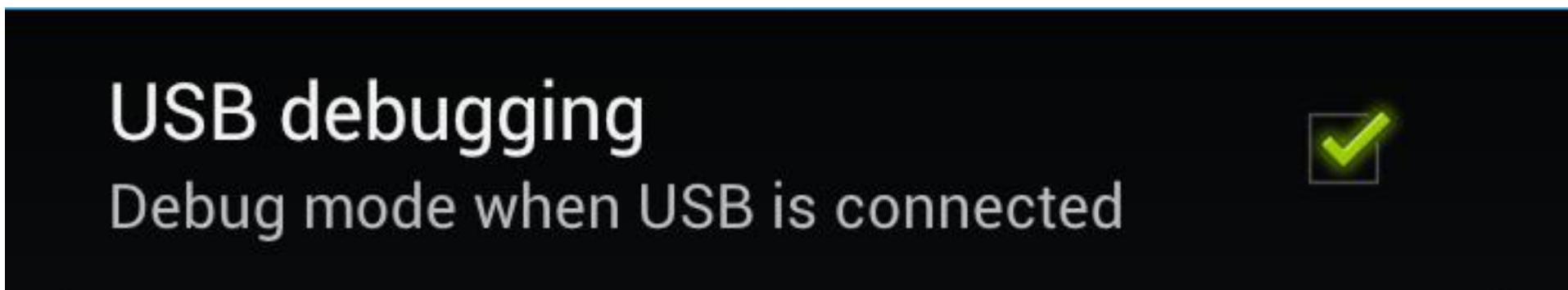
user **must** select one of a set of choices



# Checkbox

- Checkbox has 2 states: checked and unchecked
- XML code to create Checkbox

```
<?xml version="1.0" encoding="utf-8"?>
<CheckBox xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/check"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/unchecked"/>
```

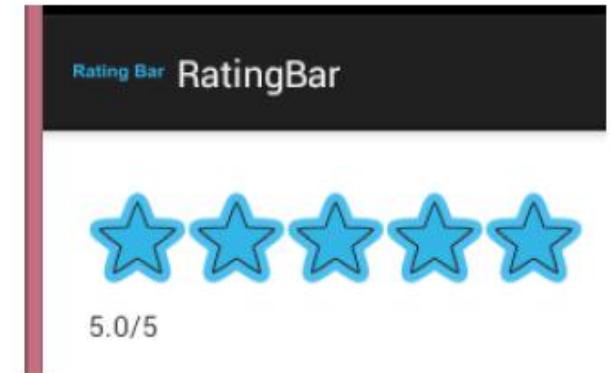


# Other Indicators & More Widgets

- ProgressBar



- RatingBar



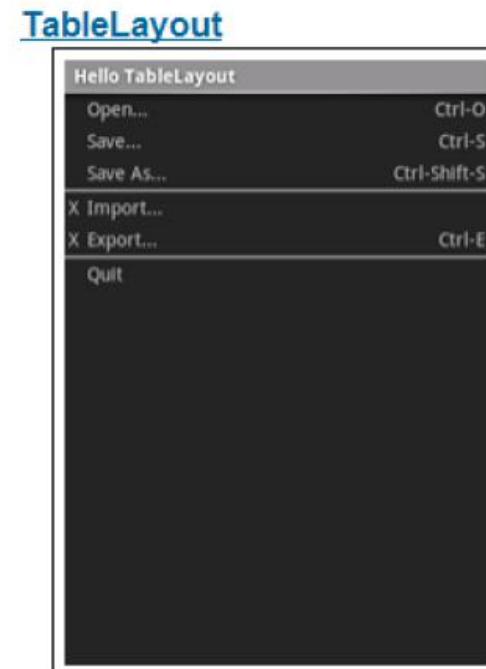
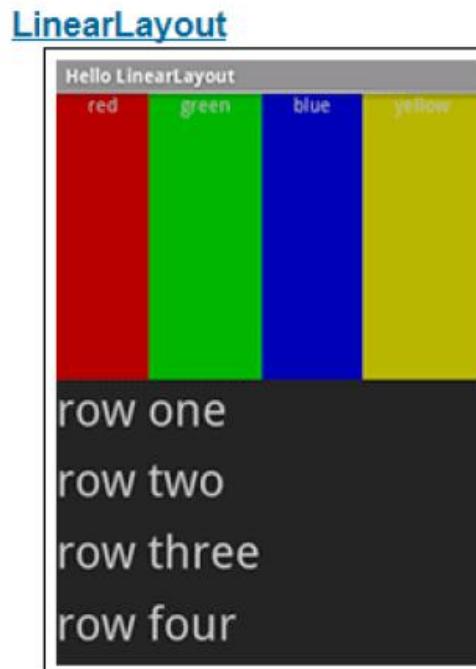
- Chronometer
- DigitalClock
- AnalogClock



# Android Layouts

# Android Layouts

- Layout? Pattern in which multiple widgets are arranged
- Layouts contain widgets
- In Android internal classes, widget is child of layout
- Layouts (XML files) stored in **res/layout**



# Views vs Layouts

## Views

- text
- buttons
- images

...

## Layouts

- linear layout
- grid layout
- relative layout

...

# Some Layouts

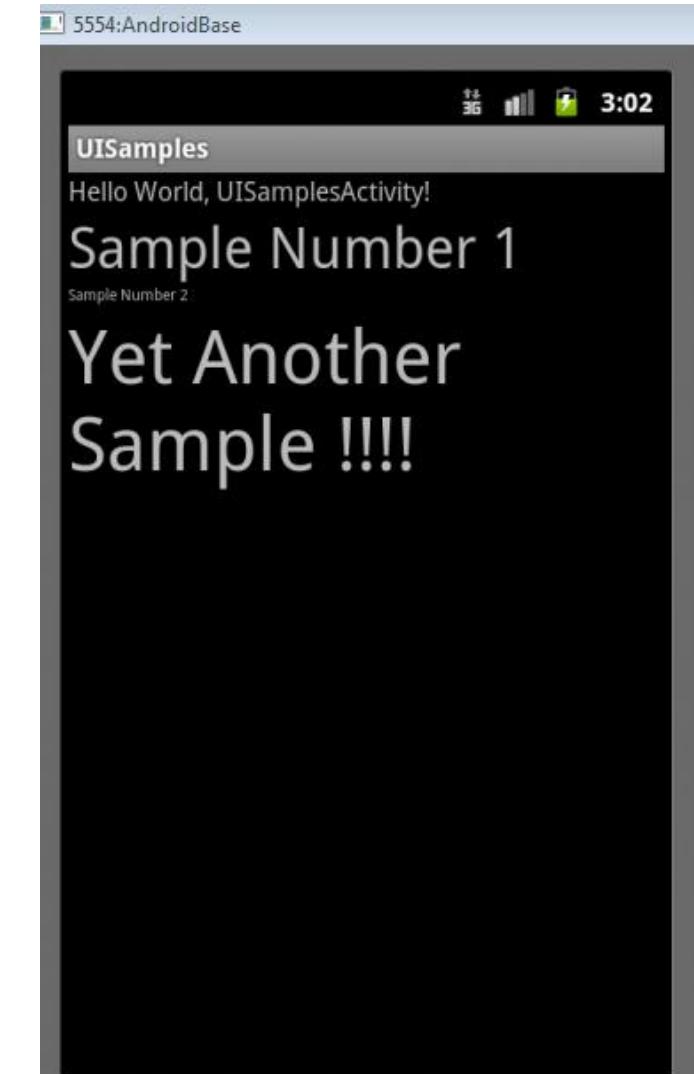
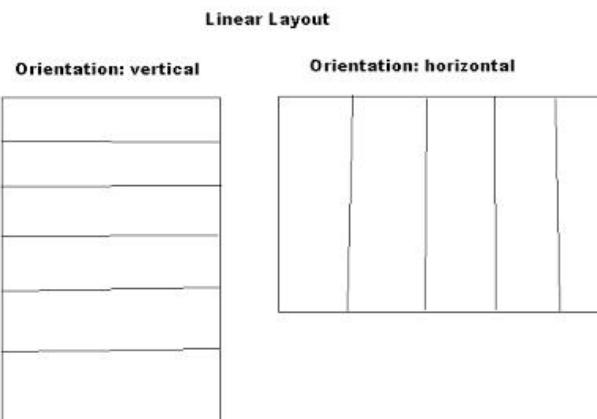
- FrameLayout,
- LinearLayout,
- TableLayout,
- GridLayout,
- RelativeLayout,
- ListView,
- GridView,
- ScrollView,
- DrawerLayout,
- ViewPager

# LinearLayout

- aligns child elements (e.g. buttons, text boxes, pictures, etc.) in one direction

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.c
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
        android:background="#ff00ff"
        android:orientation="vertical" >
```

- orientation attribute defines direction (vertical or horizontal):
  - E.g. android:orientation="*vertical*"



# Layout Width and Height Attributes

- **wrap\_content**: widget as wide/high as its content (e.g. text)
- **match\_parent**: widget as wide/high as its parent layout box
- **fill\_parent**: older form of **match\_parent**

**Text widget width  
should be as wide as  
its parent (the layout)**

**Text widget height  
should be as wide as  
the content (text)**

### Hierarchy

**Screen (Hardware)**



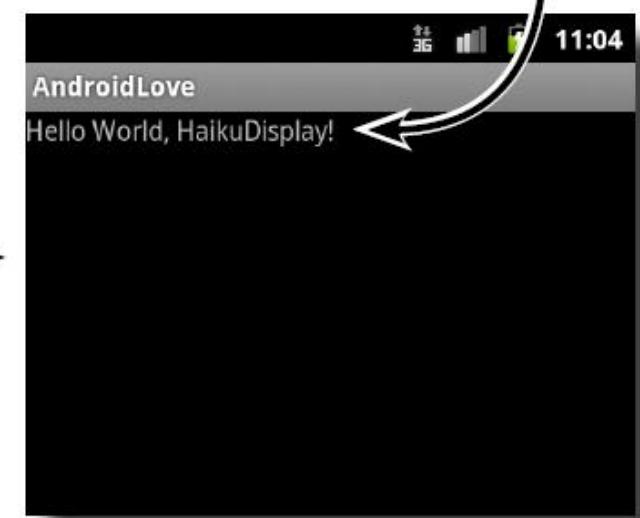
**Linear Layout**



**TextView**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        />
</LinearLayout>
```

The ViewGroup, in this case a LinearLayout, fills the screen.



The View inside the layout is a TextView, a View specifically made to display text.



main.xml

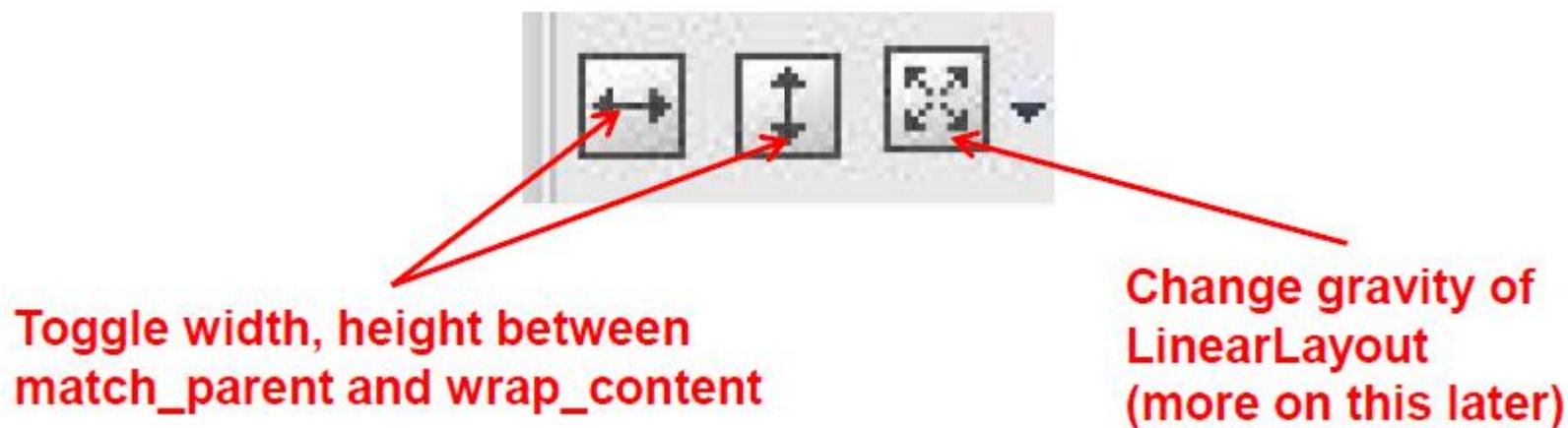
# LinearLayout in Android Studio

LinearLayout in Android Studio Graphical Layout Editor



# LinearLayout in Android Studio

- After selecting LinearLayout, toolbars buttons to set parameters



# Setting Attributes

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.c
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ff00ff"
    android:orientation="vertical" >
```

← in layout xml file

```
public class UISamplesActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void change(View v) {
        LinearLayout vg = (LinearLayout)this.findViewById(R.id.main_Layout);
        Log.d("UI SAMPLE", vg + "");
        vg.setOrientation(LinearLayout.HORIZONTAL);
    }
}
```

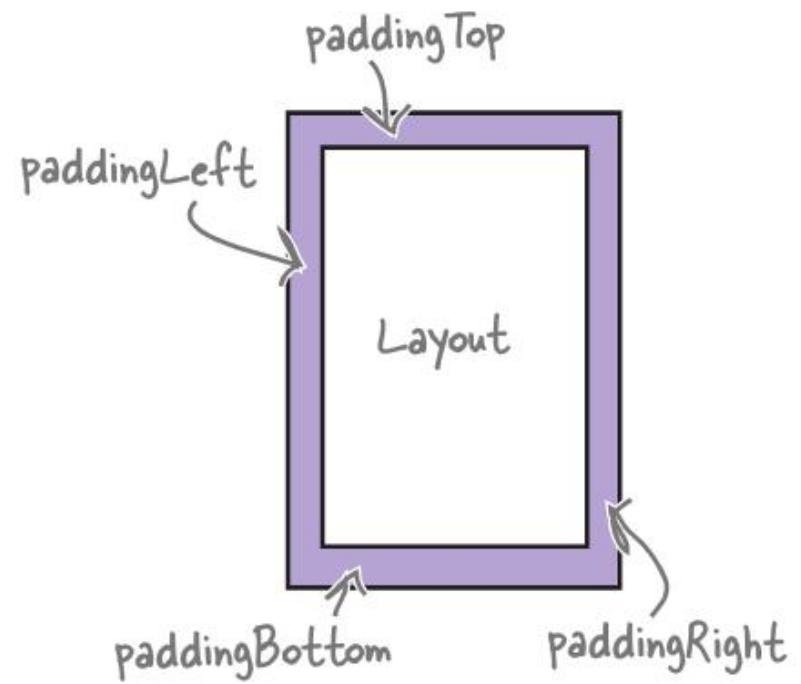
Can also design UI, set attributes in Java  
program (e.g. ActivityMain.java) (More later)

# Adding Padding

- Paddings sets space between layout sides and its parent (e.g. the screen)

```
<RelativeLayout ...  
    android:paddingBottom="16dp"  
    android:paddingLeft="16dp"  
    android:paddingRight="16dp"  
    android:paddingTop="16dp">  
    ...  
</RelativeLayout>
```

...  
Add padding of 16dp.

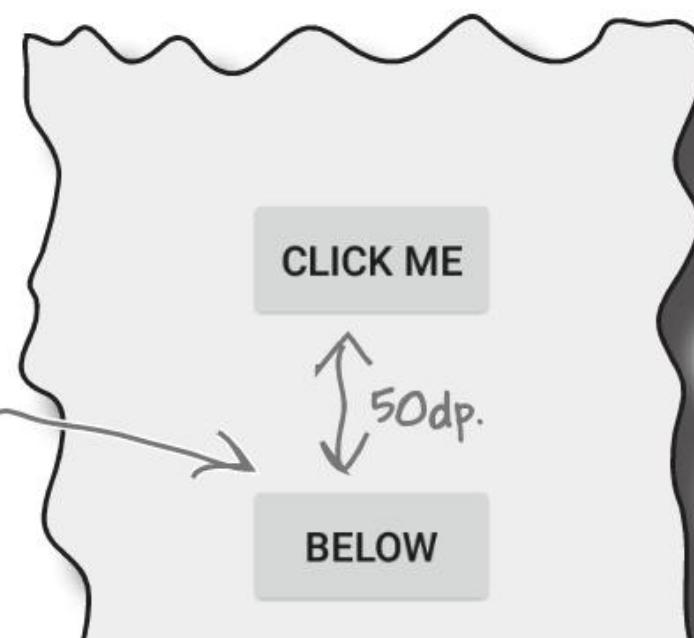


# Setting Margins

- Can increase gap (margin) between adjacent widgets
- E.g. To add margin between two buttons, in declaration of bottom button

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignLeft="@+id/button_click_me"  
    android:layout_below="@+id/button_click_me"  
    android:layout_marginTop="50dp" ←  
    android:text="@string/button_below" />  
</RelativeLayout>
```

Adding a margin to  
the top of the bottom  
button adds extra space  
between the two views.



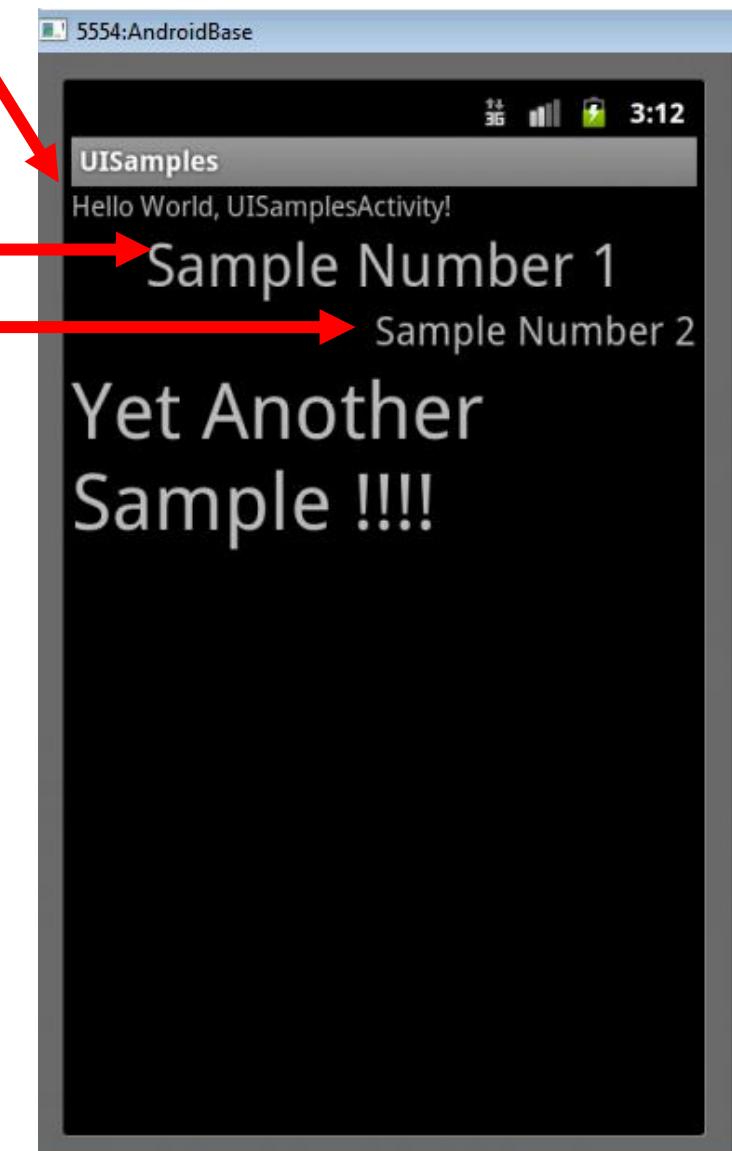
# Gravity Attribute

- By default, linearlayout left-and top-aligned
- Gravity attribute changes alignment :
  - e.g. android:gravity = “right”

left

center

right



# Linear Layout Weight Attribute

- Specifies "importance", larger weights takes up more space

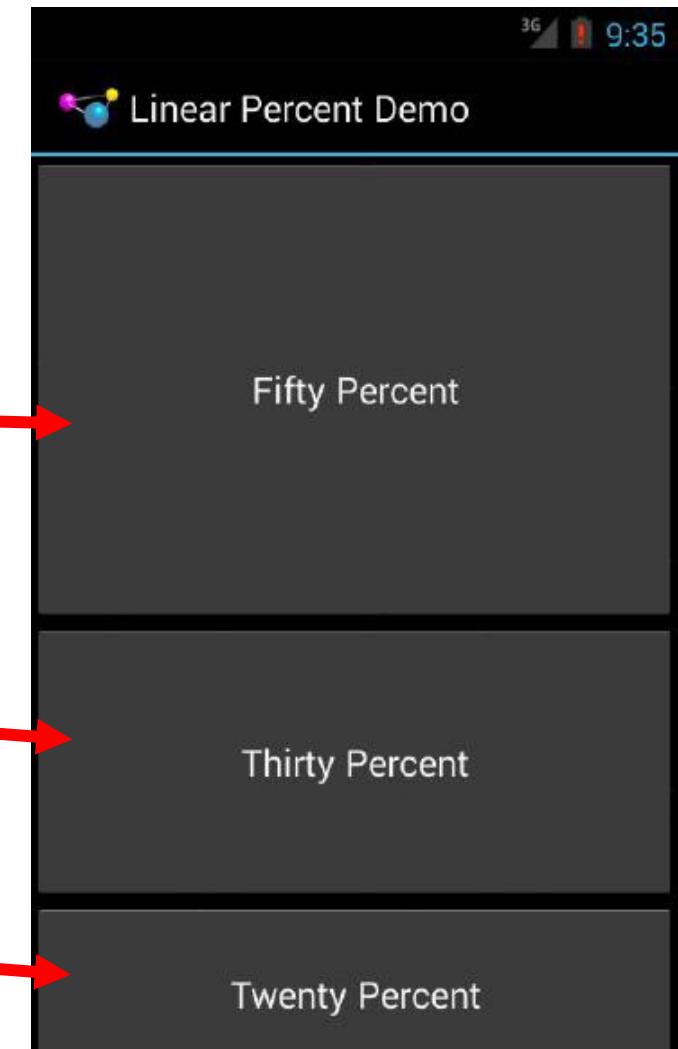
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <Button
        android:layout_width="match_parent"
        android:layout_height="0dip"
        android:layout_weight="50" -----
        android:text="@string/fifty_percent"/>

    <Button
        android:layout_width="match_parent"
        android:layout_height="0dip"
        android:layout_weight="30" -----
        android:text="@string/thirty_percent"/>

    <Button
        android:layout_width="match_parent"
        android:layout_height="0dip"
        android:layout_weight="20" -----
        android:text="@string/twenty_percent"/>

</LinearLayout>
```



# LinearLayout Attributes

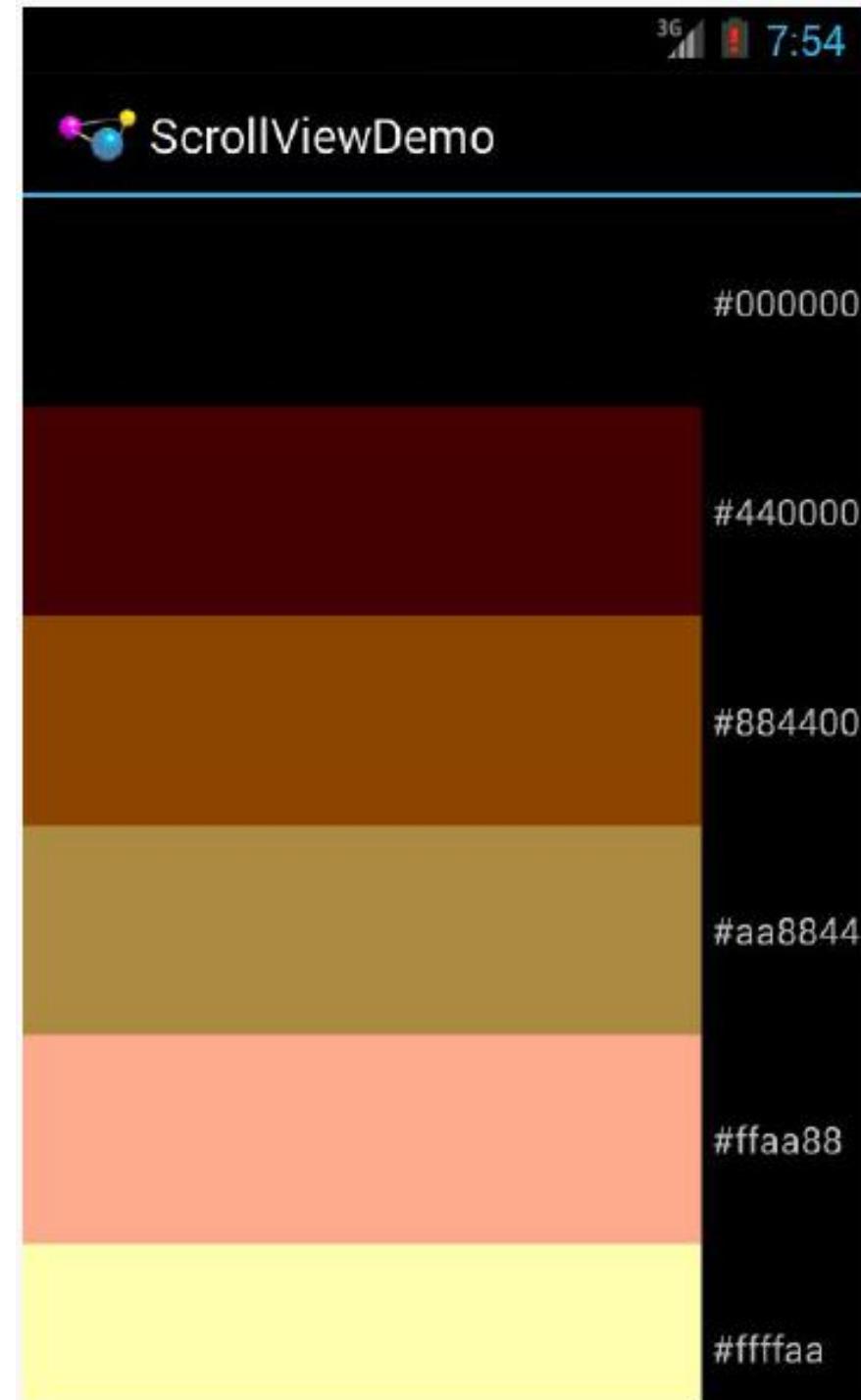
## XML attributes

<code>android:baselineAligned</code>	When set to false, prevents the layout from aligning its children's baselines.
<code>android:baselineAlignedChildIndex</code>	When a linear layout is part of another layout that is baseline aligned, it can specify which of its children to baseline align to (that is, which child TextView).
<code>android:divider</code>	Drawable to use as a vertical divider between buttons.
<code>android:gravity</code>	Specifies how an object should position its content, on both the X and Y axes, within its own bounds.
<code>android:measureWithLargestChild</code>	When set to true, all children with a weight will be considered having the minimum size of the largest child.
<code>android:orientation</code>	Should the layout be a column or a row? Use "horizontal" for a row, "vertical" for a column.
<code>android:weightSum</code>	Defines the maximum weight sum.

# Scrolling

- Phone screens are small, scrolling content helps
- Examples: Scroll through
  - large image
  - Linear Layout with lots of elements
- Views for Scrolling:
  - **ScrollView** for vertical scrolling
  - **HorizontalScrollView**
- Rules:
  - Only one direct child View
  - Child could have many children of its own

```
<ScrollView  
    ...>  
    <LinearLayout>  
        ....  
        <!-- you can have as many Views in here as you want -->  
    </LinearLayout>  
</ScrollView>
```



# Android Activities

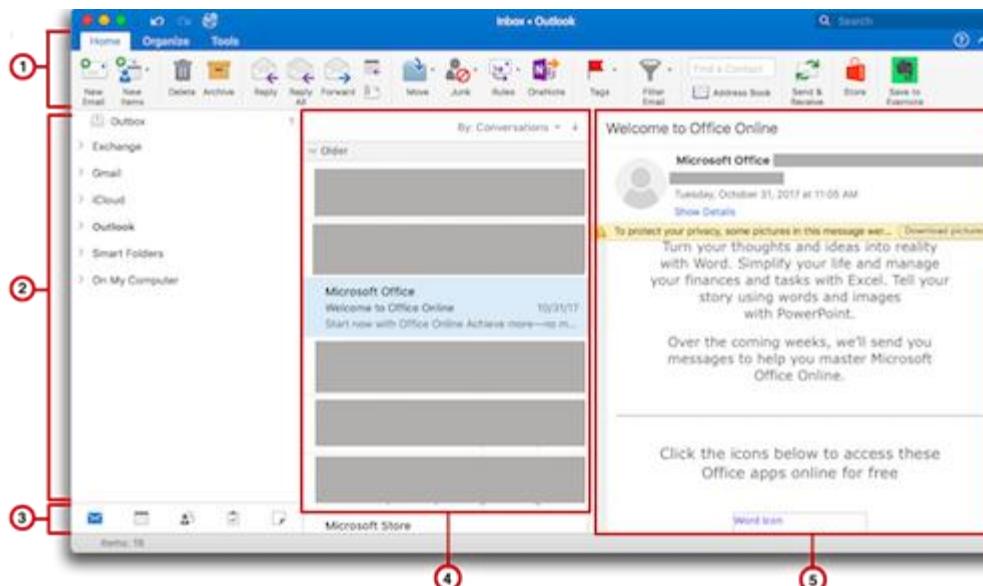
Hua Huang

# Reading Materials

- Chapter 3,4

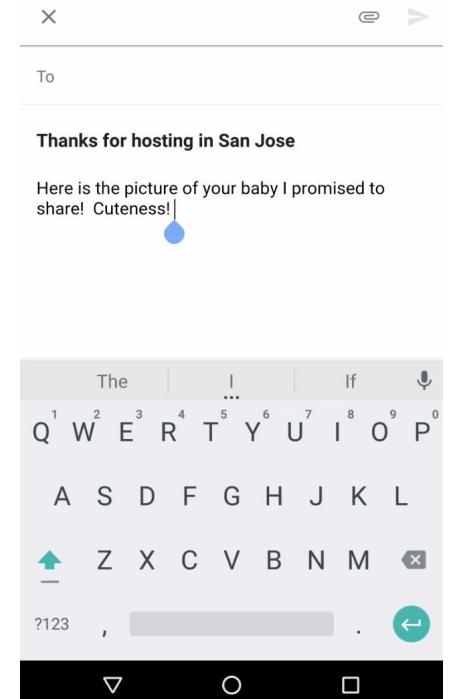
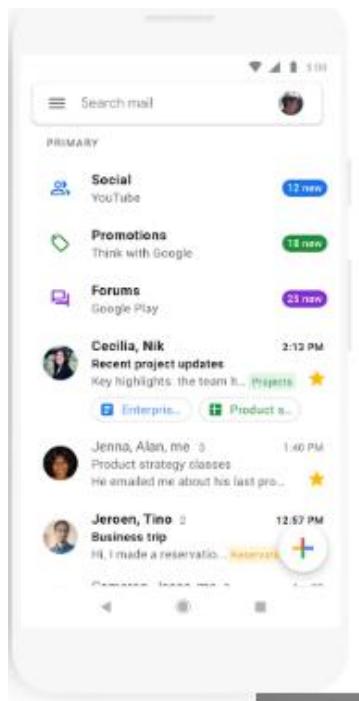
# Desktop Program vs. Mobile Apps

- A desktop program has a single entry point: main()
- A mobile app can start from different screens (non-deterministic)



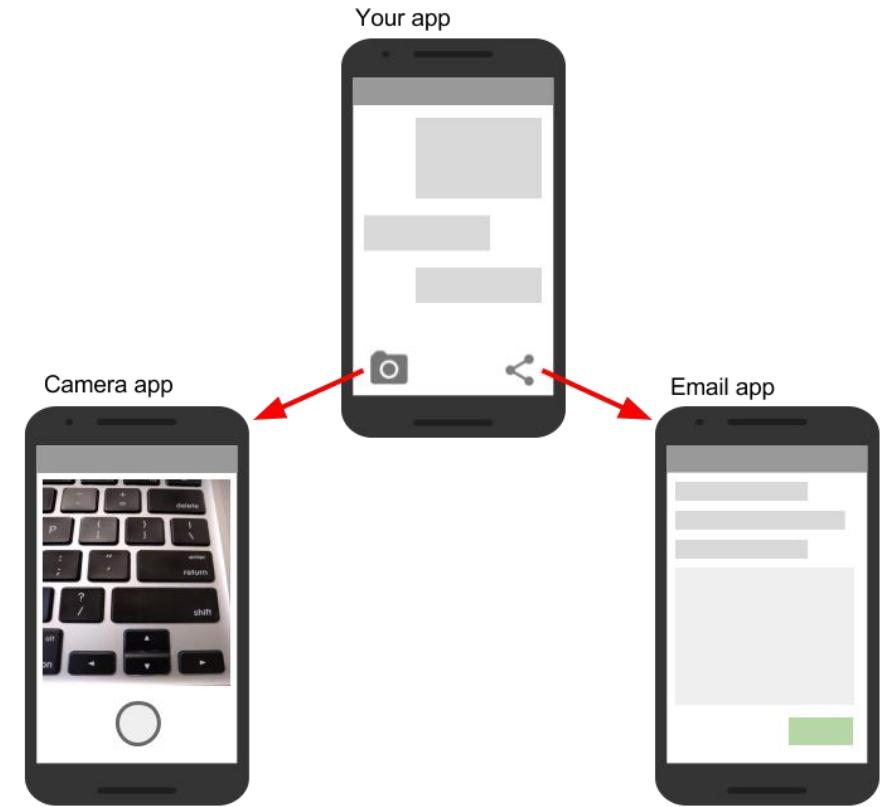
# Challenge: the starting point of an app is non deterministic

- e.g. open an email app from your home screen, you might see a list of emails.
- By contrast, use a social media app that then launches email app, you might go directly to the email app's screen for composing an email.



# Activities

- The Activity class is designed to facilitate this paradigm.
- When one app invokes another, the calling app invokes an activity in the other app, rather than the app as an atomic whole.
- In this way, the activity serves as the entry point for an app's interaction with the user.
- You implement an activity as a subclass of the Activity class.



# Activity vs Android Application

- An activity provides the window in which the app draws its UI.
  - This window typically fills the screen, but may be smaller than the screen and float on top of other windows.
- Generally, one activity implements one screen in an app.
  - e.g., one of an app's activities may implement a Preferences screen, while another activity implements a Select Photo screen.
- Android Application
  - includes activities, services, intents,data... etc
  - the manifest file itemize them

# Activities

- Most apps contain multiple screens, which means they comprise multiple activities.
- Typically, one activity the main activity, which is the first screen to appear when the user launches the app.
- Each activity can start another activity to perform different actions.
  - e.g., the main activity in a simple e-mail app may provide the screen that shows an e-mail inbox.
  - From there, the main activity might launch other activities that provide screens for tasks like writing e-mails and opening individual e-mails.

# Activities

- Each activity is only loosely bound to the other activities in the app;
  - there are usually minimal dependencies among the activities in an app.
  - In fact, activities often start up activities belonging to other apps.
  - e.g., a browser app might launch the Share activity of a social-media app.

# Inside “Hello World” AndroidManifest.xml

package  
name  
Android  
Version

Activity List

This file is written using xml namespace and tags and rules for android

```
<?xml version="1.0"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.commonsware.android.skeleton"
    android:versionCode="1"
    android:versionName="1.0">

    <application>
        <activity
            android:name="Now"
            android:label="Now">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

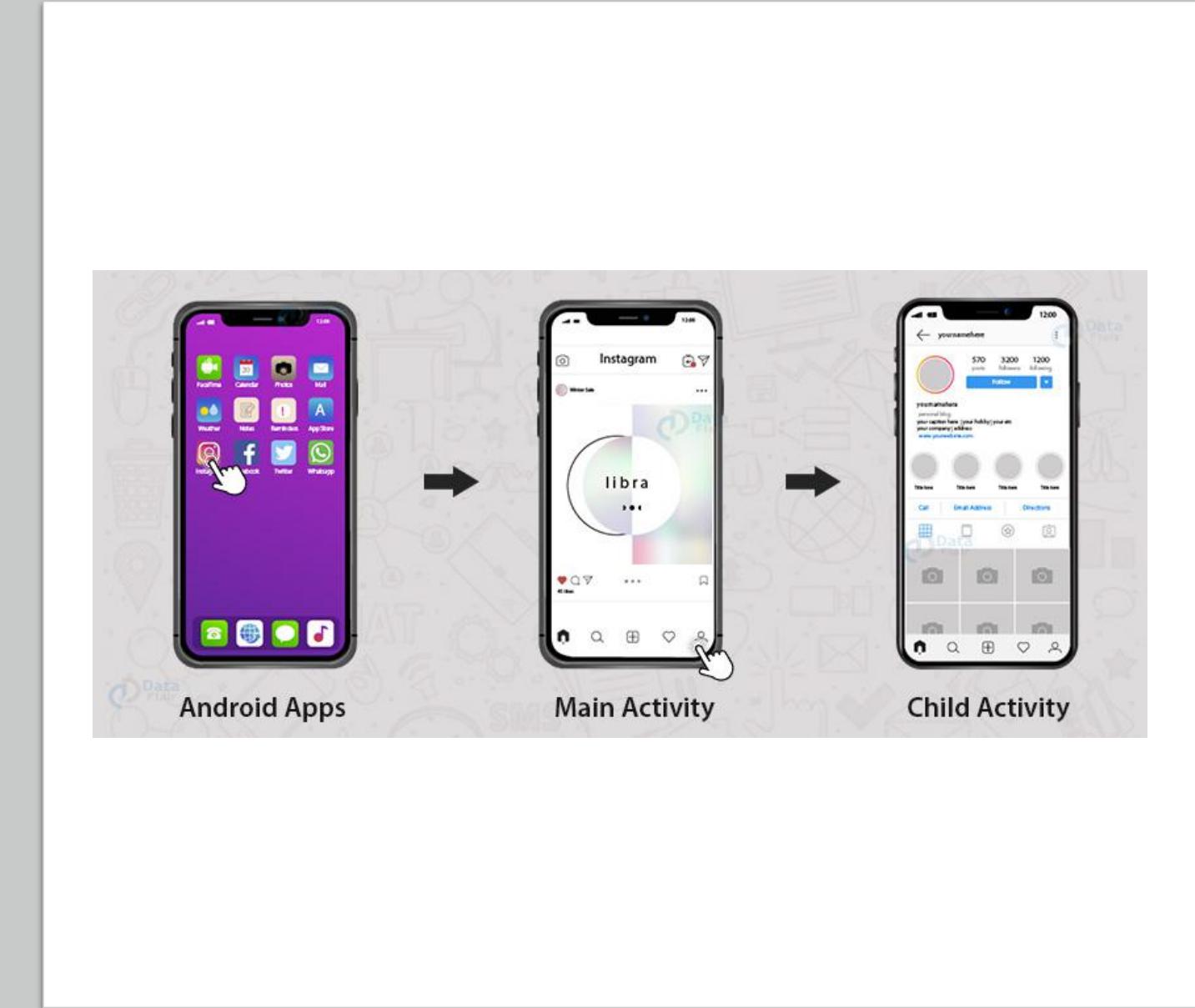
</manifest>
```

One activity (screen)  
designated LAUNCHER.  
The app starts running here

# Activity Cycles

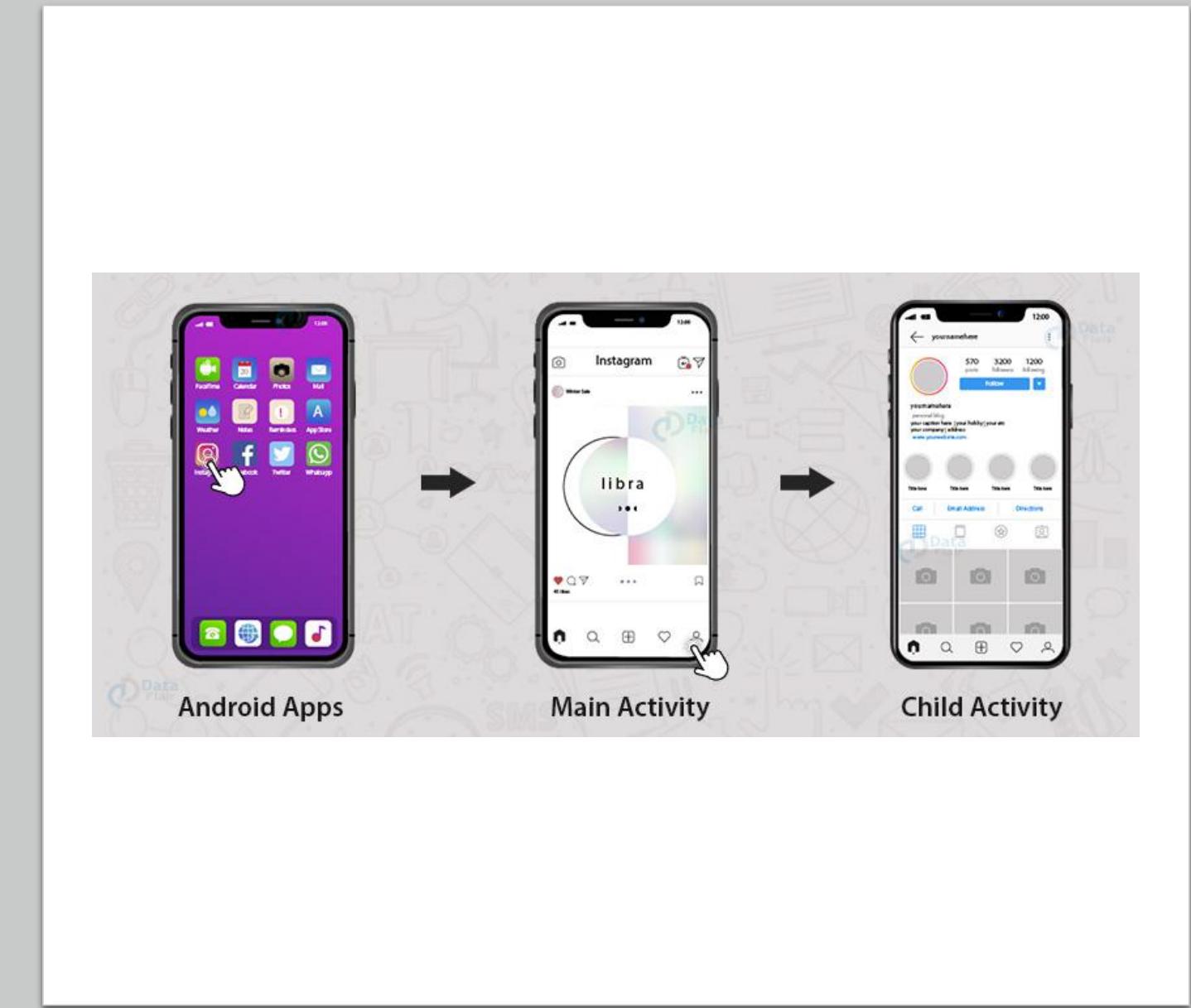
# Activity Lifecycle

- Activities keep evolving in the Apps.
  - Example: click to open the main activity. Click a button to open on child activity
  - What happens under the hood?
  - How to make an activity light up?
  - How to make it go away?
  - How to maintain the previous context?



# Activity Lifecycle

- What happens under the hood?
  - How to make old screen go away?
    - save states
    - stop sensors or GPS
    - release RAM
    - ...
  - How to make an activity light up?
    - draw the UIs
    - initiate sensors

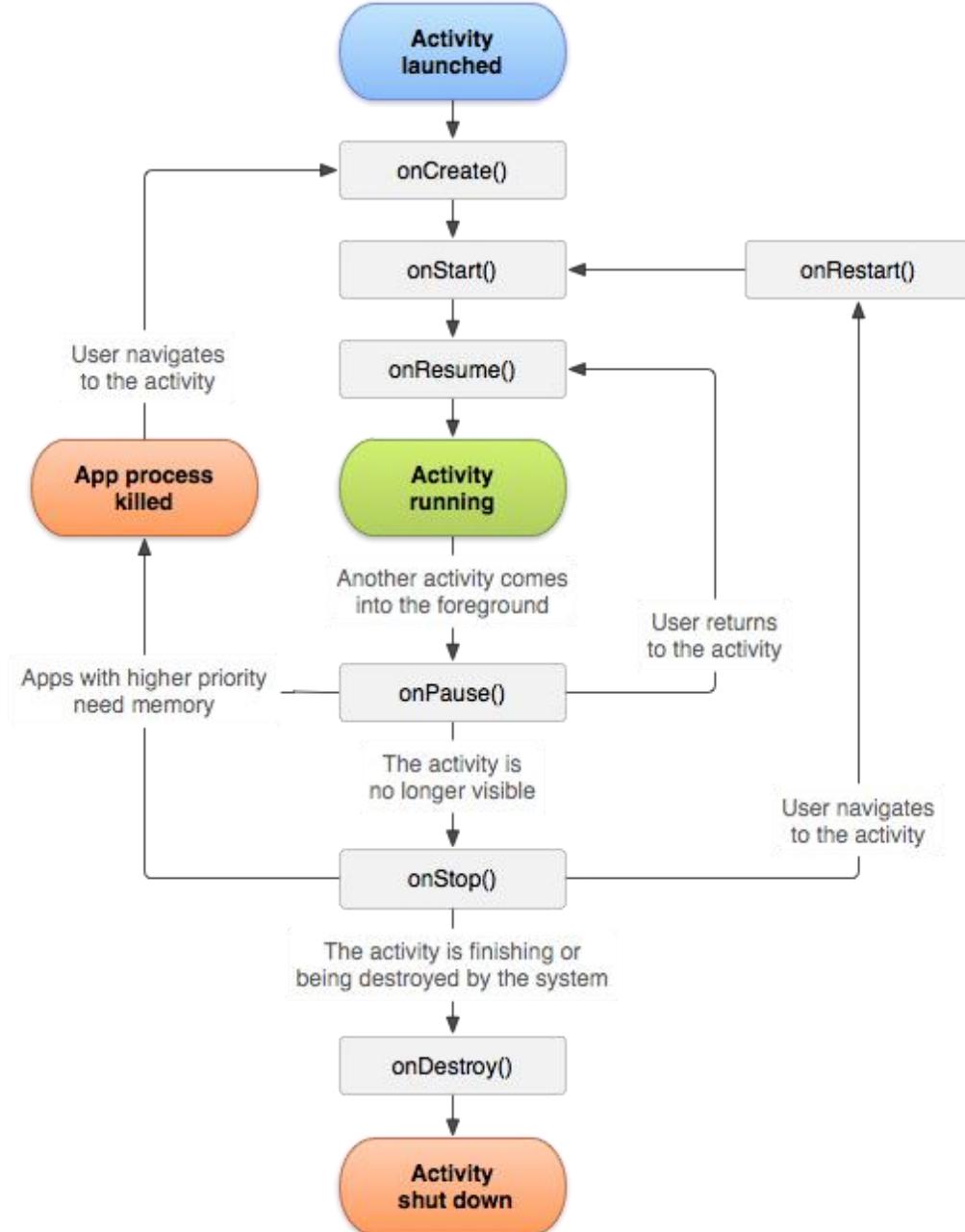


# Activity

- The Activity class provides a number of callbacks to manage the transitions
- When the system is creating, stopping, or resuming an activity, we can define the the codes to run

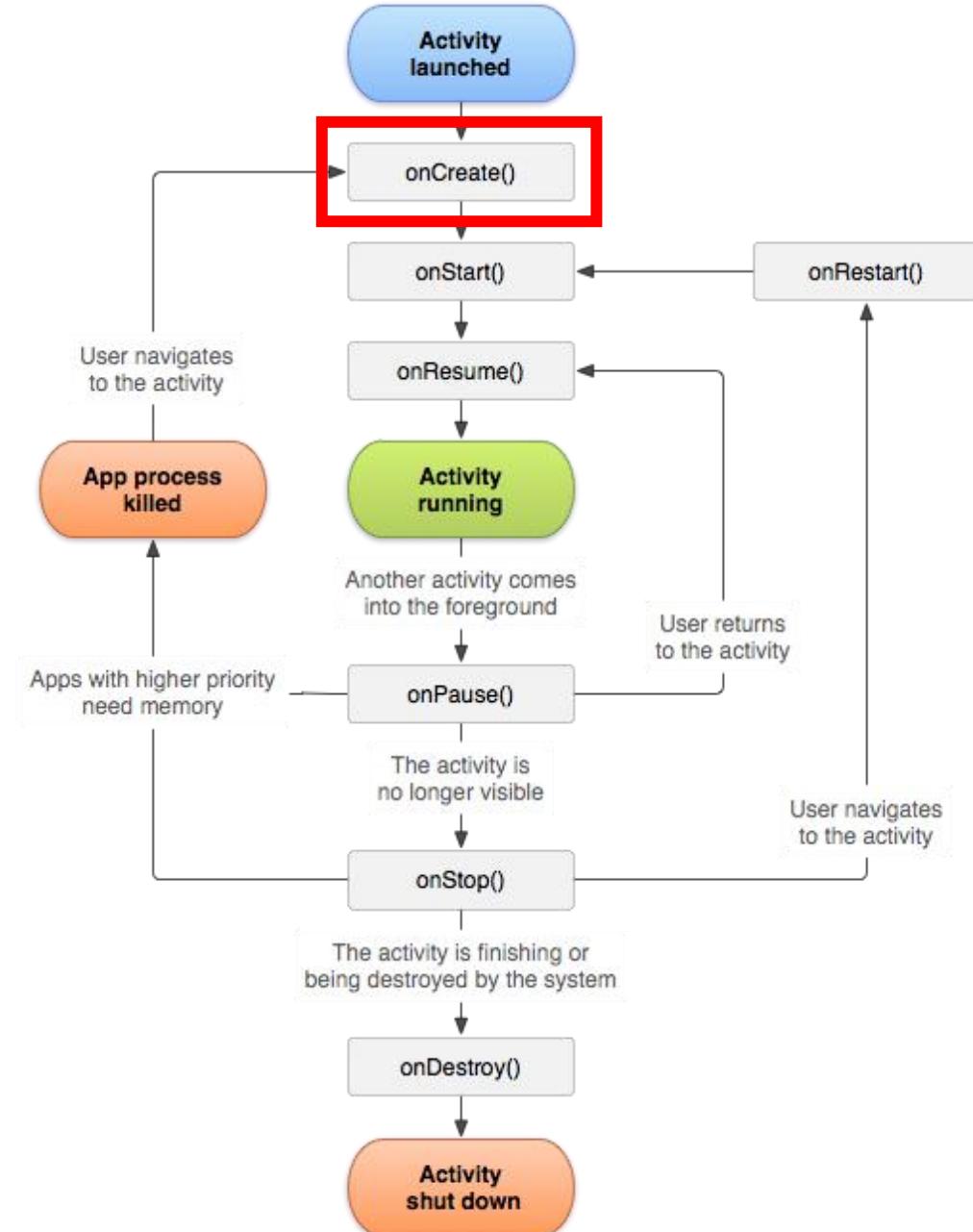
# A Simplified Activity Lifecycle

Think: Why are they necessary?



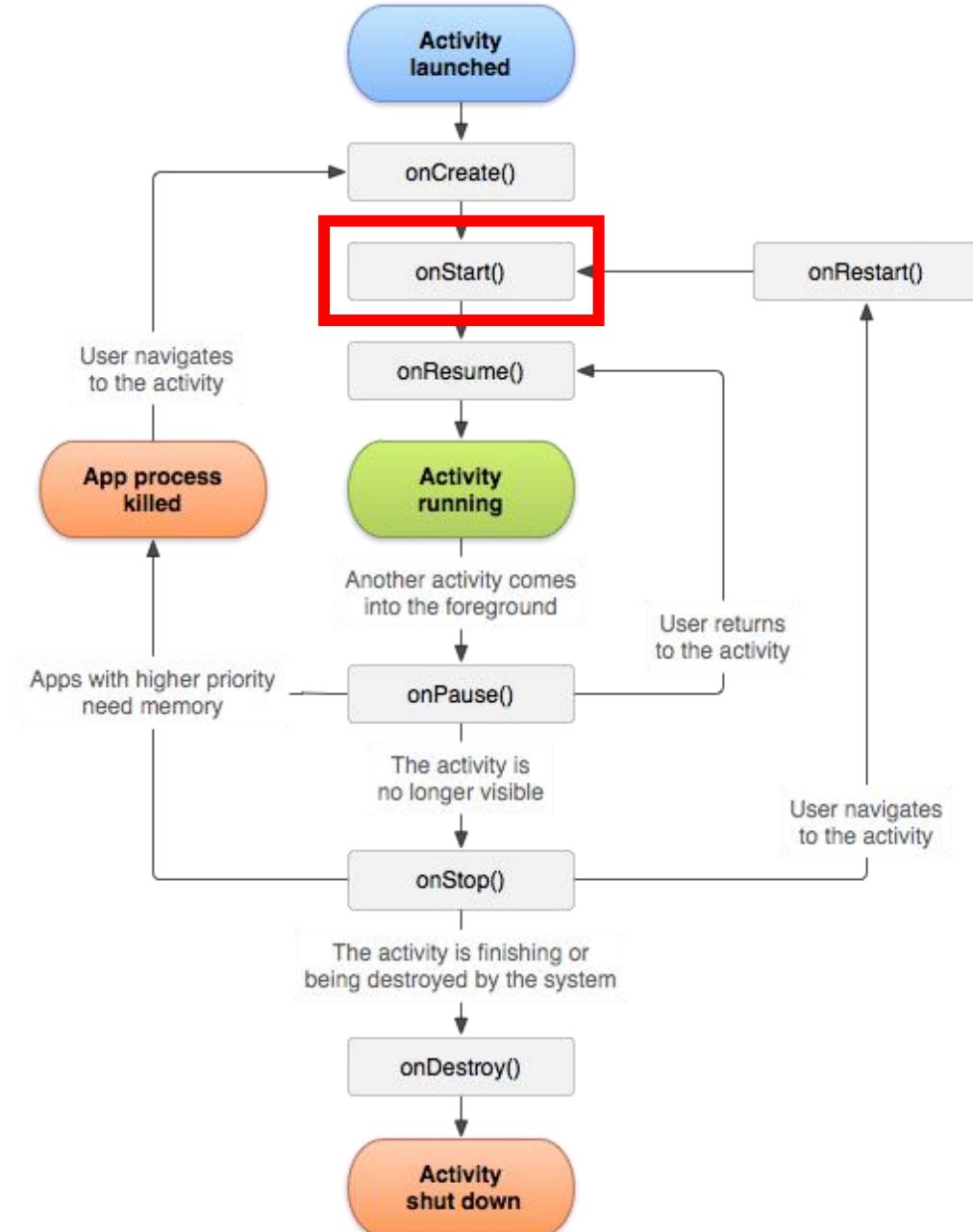
# onCreate()

- You must implement this callback, which fires when the system first creates the activity.
- Perform basic application startup logic that should happen only once for the entire life of the activity.
  - E.g., bind data to lists, associate the activity with a ViewModel, and instantiate some class-scope variables
- Enters **Created** state after execution



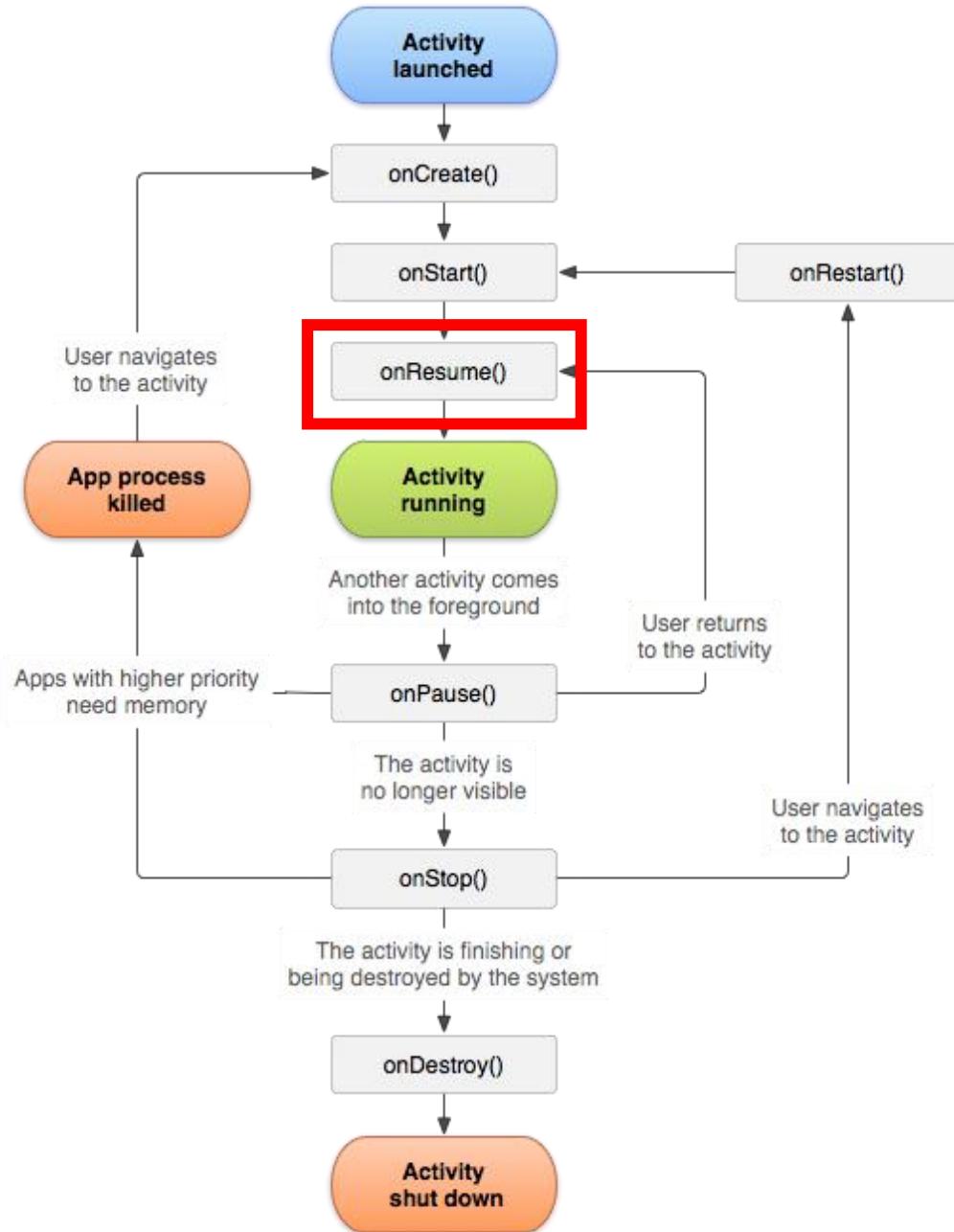
# onStart()

- Once in **Created** state, automatically calls this function.
- Makes the activity visible to the user, as the app prepares for the activity to enter the foreground and become interactive.
- The `onStart()` method completes very quickly and, as with the **Created** state, the activity does not stay resident in the **Started** state.
  - Once this callback finishes, the activity enters the **Started** state, and the system invokes the `onResume()` method.

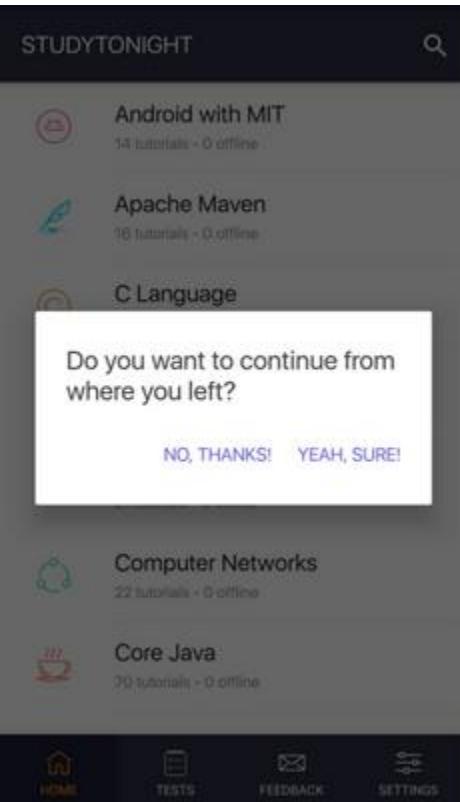


# onResume()

- When the activity enters the **Started** state, it comes to the foreground, and then the system invokes the onResume() callback.
- Resumed is the state in which the app interacts with the user.
- The app stays in this state until something happens to take focus away from the app.
  - E.g., receiving a phone call, the user's navigating to another activity, or the device screen's turning off.
- When an interruptive event occurs, the activity enters the **Paused** state, and the system invokes the onPause() callback.
- If the activity returns to the Resumed state from the Paused state, the system once again calls onResume() method.



# The Paused State



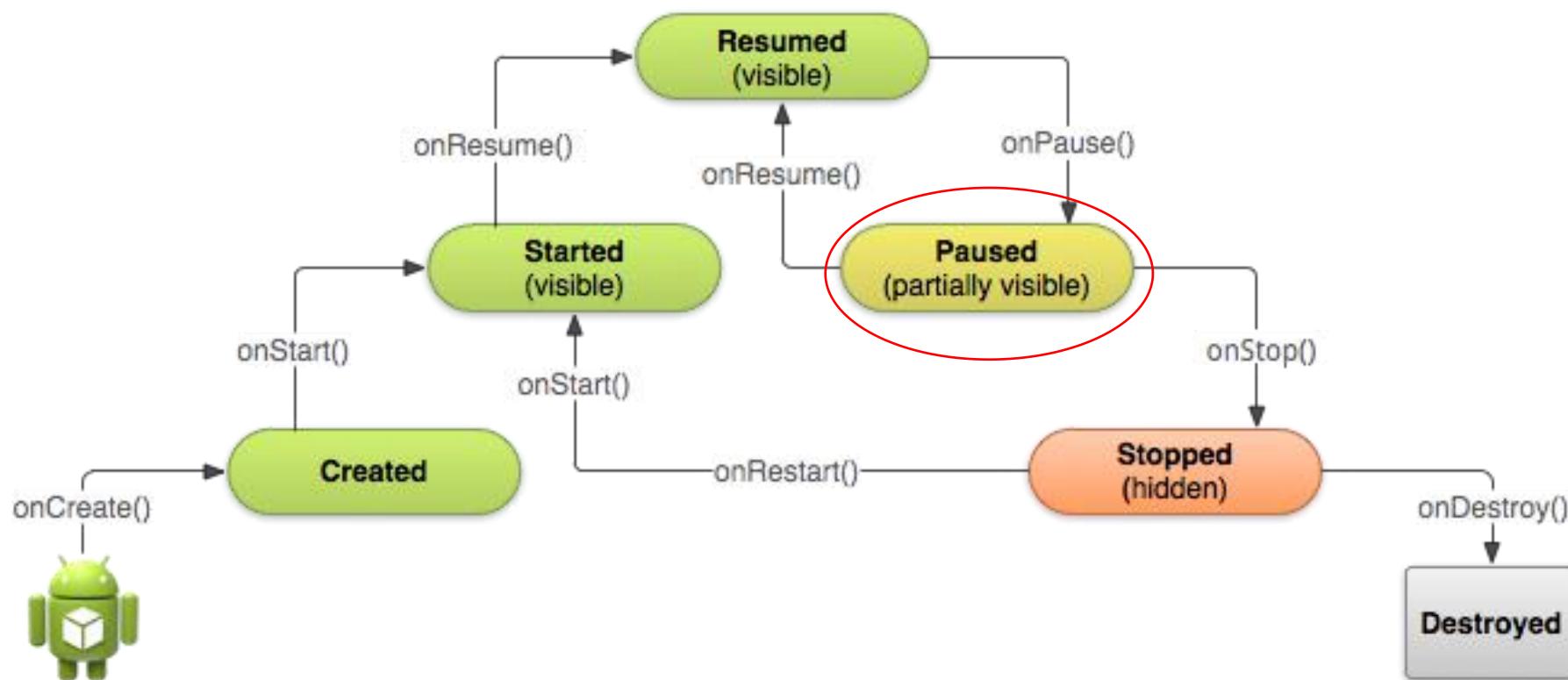
Activity State:  
**Paused**

Process state is  
Background(lost focus).

And the likelihood of killing the  
app is More.

- The app stays in **Resumed** state until something happens to take focus away from the app.
- The system calls this method as the first indication that the user is leaving your activity
  - It does not always mean the activity is being destroyed
- It indicates that the activity is no longer in the foreground

# The Paused State



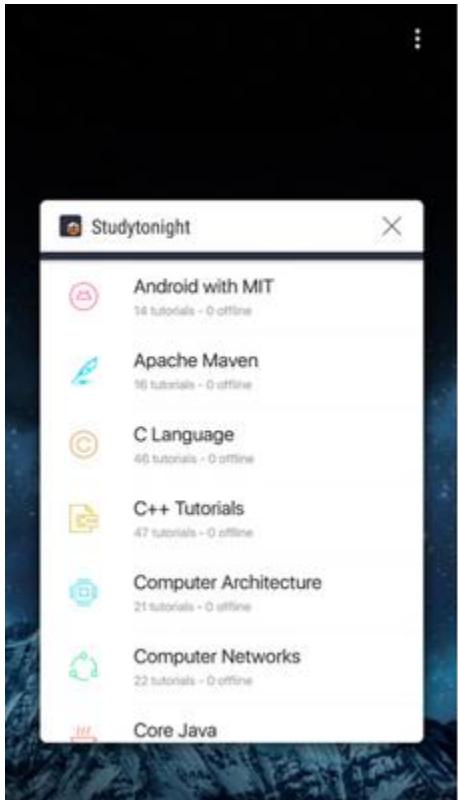
# onPause()

- Scenarios that trigger onPause():
  - Some event interrupts app execution, as described in the [onResume\(\)](#) section. This is the most common case.
  - A new, semi-transparent activity (such as a dialog) opens. As long as the activity is still partially visible but not in focus, it remains paused.
  - In Android 7.0 (API level 24) or higher, multiple apps run in multi-window mode. Because only one of the apps (windows) has focus at any time, the system pauses all of the other apps.

# onPause()

- We can stop any functionality that does not need to run while the component is not in the foreground
  - E.g., stop the camera preview
  - release system resources, handles to sensors (like GPS), or any resources that may affect battery life while your activity is paused and the user does not need them

# Stopped State

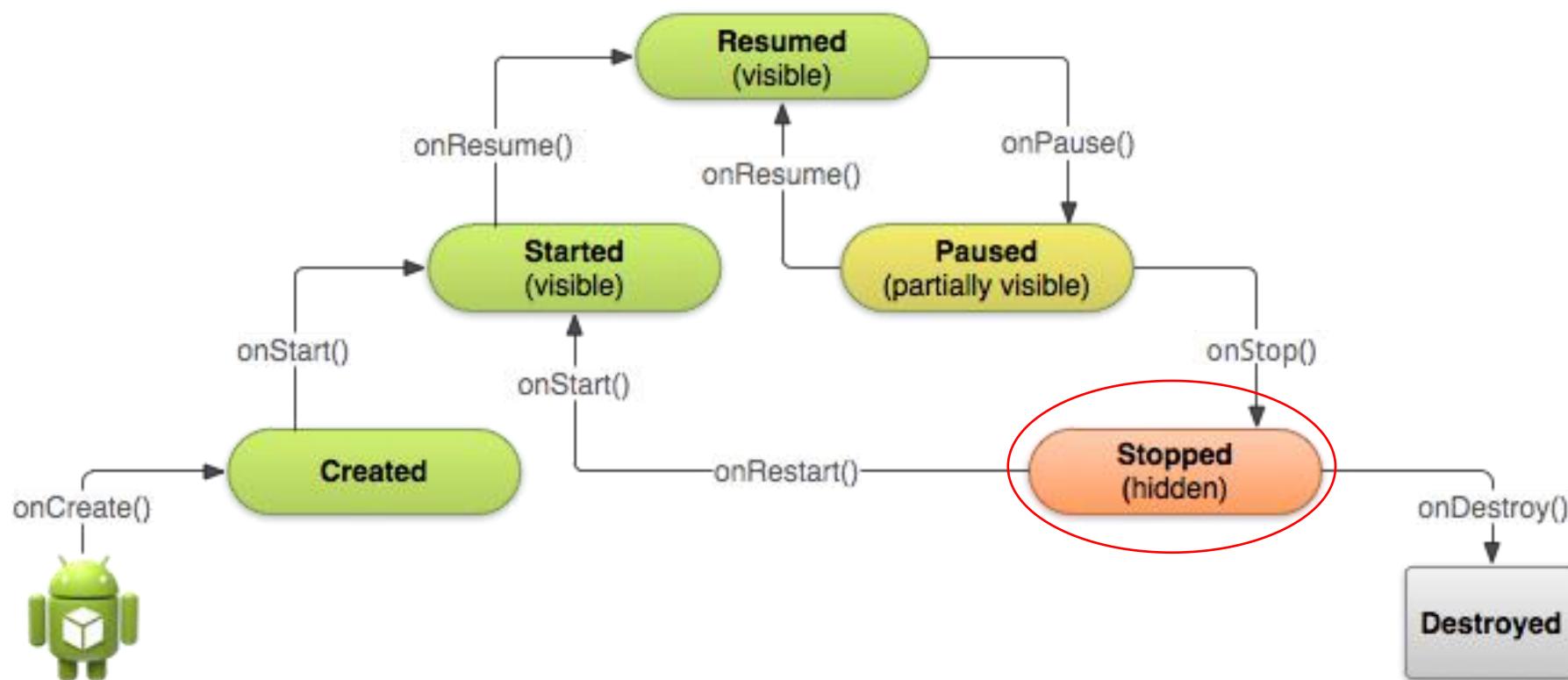


Activity State:  
**Stopped**

Process state is  
Background(not visible).

And the likelihood of killing the  
app is Most.

- When a new Activity is started on top of the current one or when a user hits the Home key, the activity is brought to Stopped state.
- The activity in this state is invisible, but it is not destroyed.
- Android Runtime may kill such an Activity in case of resource crunch.



# onStop()

- When your activity is no longer visible to the user, it has entered the Stopped state, and the system invokes the onStop() callback.
- Should release or adjust resources that are not needed while the app is not visible to the user here.
  - E.g., pause animations
  - switch from fine-grained to coarse-grained location updates
- Also use onStop() to perform relatively CPU-intensive shutdown operations.
  - E.g., save information to a database

# Question

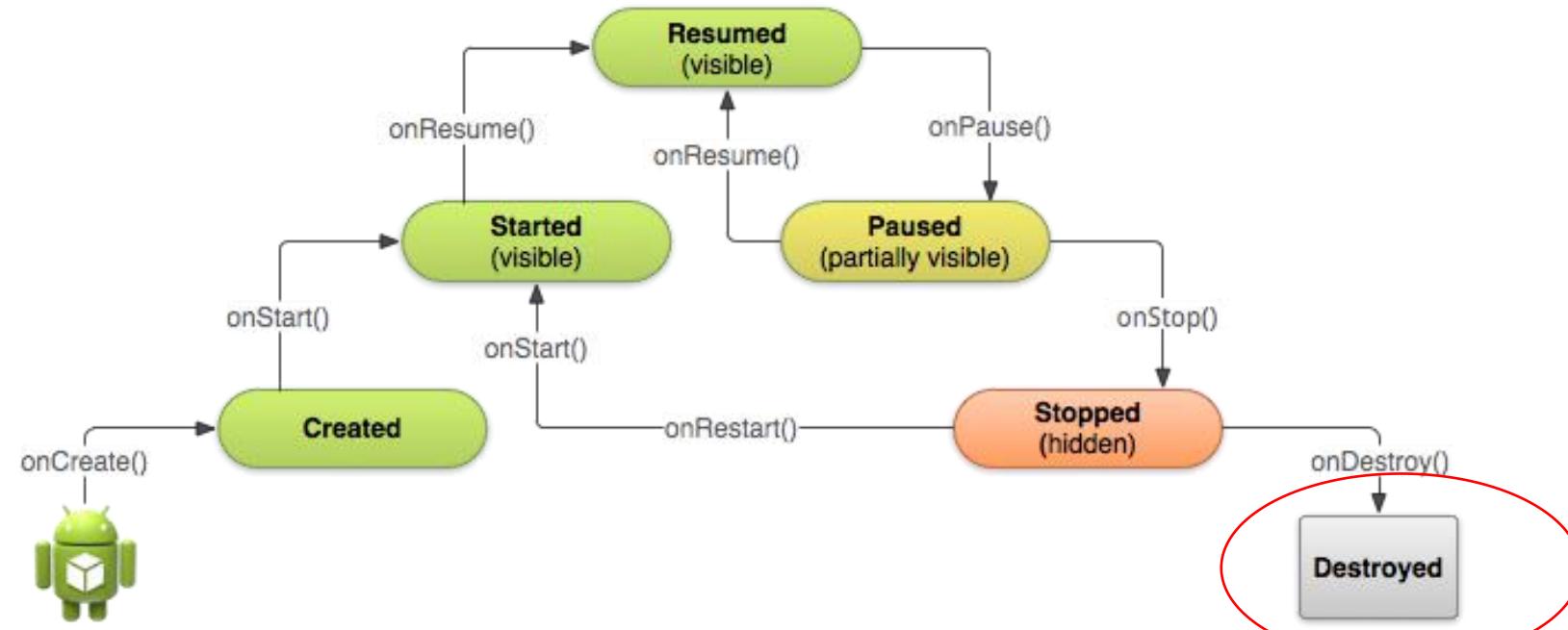
- Should we release and adjust UI components in onPause() instead? What are the pros and cons?
- Answer: a Paused activity may still be fully visible if in multi-window mode. As such, you should consider using onStop() instead of onPause() to fully release

# Question

- Should we perform relatively CPU-intensive shutdown operations in onPause() instead?
- Answer: No, because onPause() is supposed to be brief.

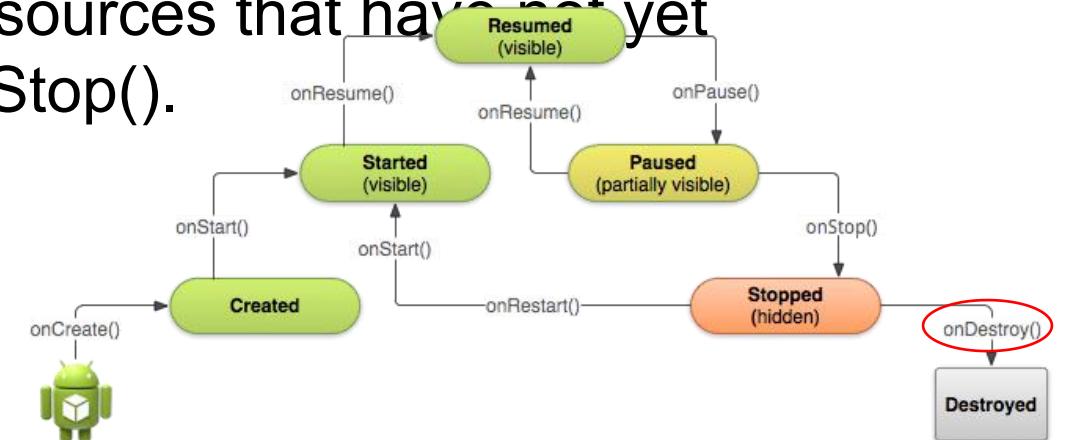
# The Destroyed State

- When a user hits a Back key or Android Runtime decides to reclaim the memory allocated to an Activity
- The Activity is out of the memory and it is invisible to the user.



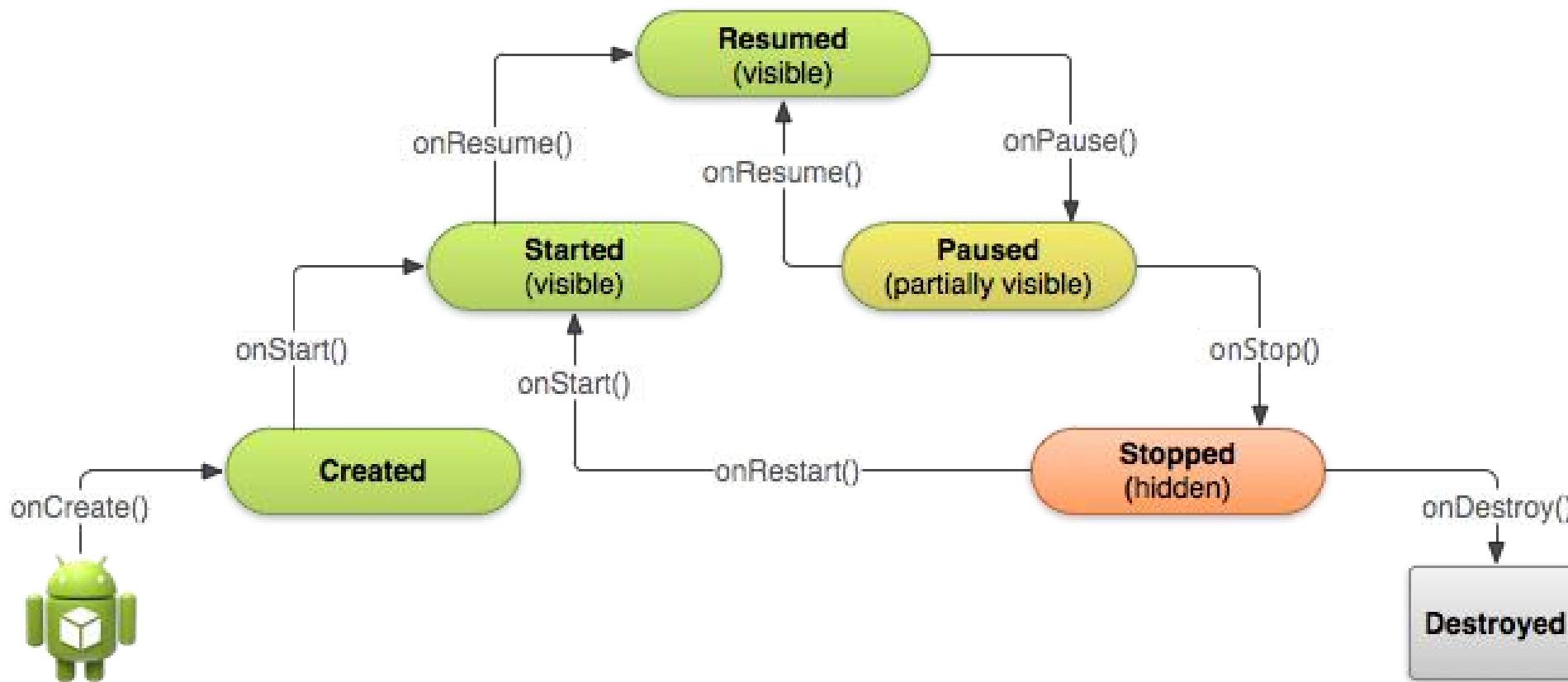
# onDestroy()

- `onDestroy()` is called before the activity is destroyed. The function is called either because:
  - The activity is finishing (due to the user completely dismissing the activity or due to `finish()` being called on the activity), or
  - The system is temporarily destroying the activity due to a configuration change (such as device rotation or multi-window mode)
- The `onDestroy()` callback should release all resources that have not yet been released by earlier callbacks such as `onStop()`.



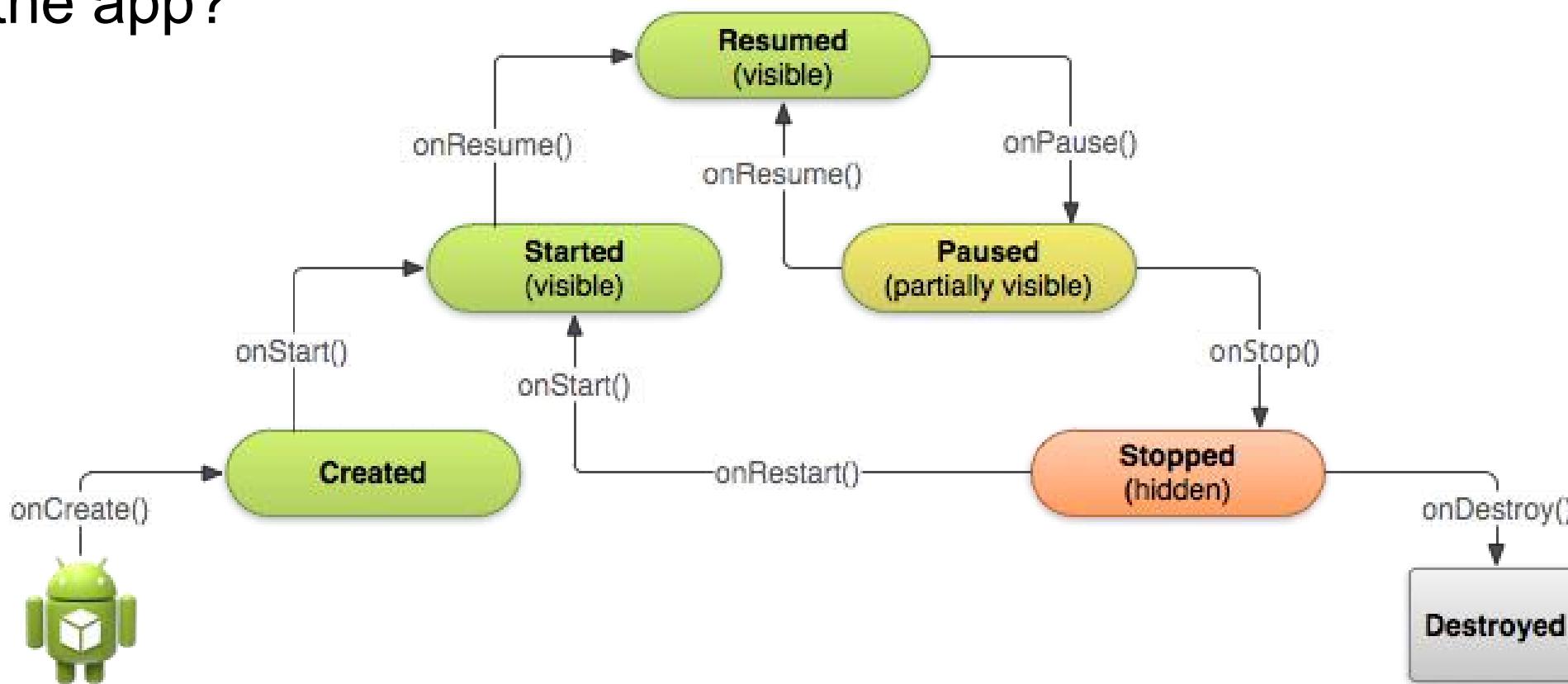
# Discussions

- What functions are called when open the app?



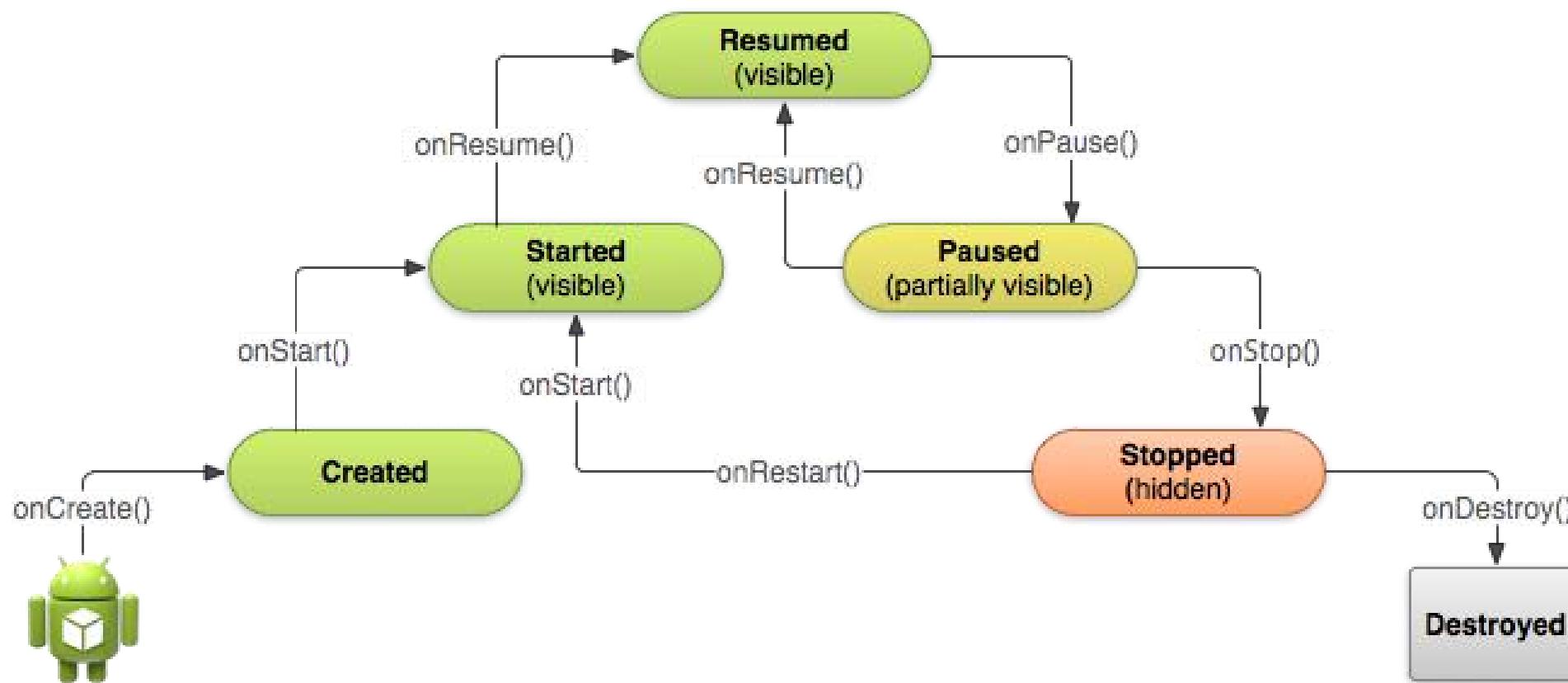
# Discussions

- What functions are called when back button pressed and exit the app?



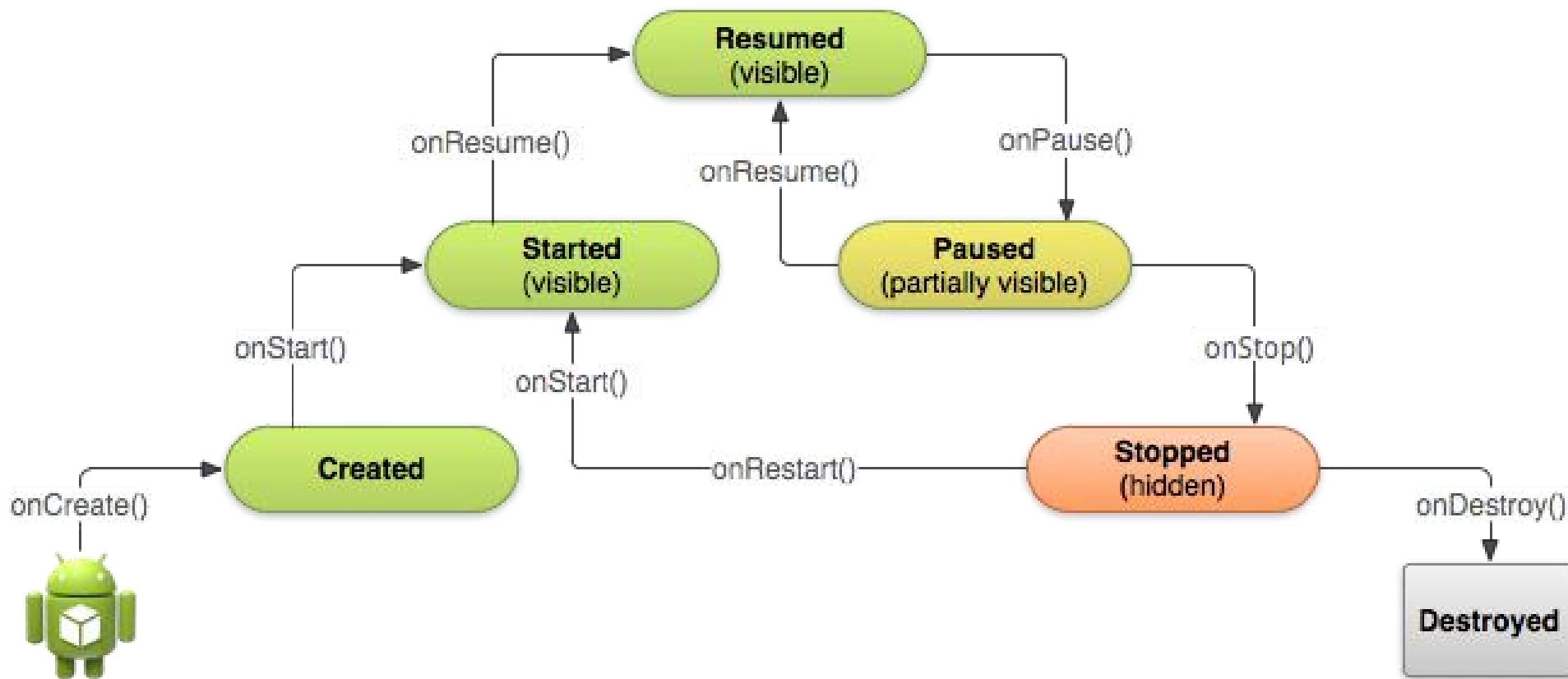
# Discussions

- What functions are called when home button pressed?



# Discussions

- What functions are called when open app from recent task list?



- Good implementation of the lifecycle callbacks can help your app avoid the following:
  - Crashing if the user receives a phone call or switches to another app while using your app.
  - Consuming valuable system resources when the user is not actively using it.
  - Losing the user's progress if they leave your app and return to it at a later time.
  - Crashing or losing the user's progress when the screen rotates between landscape and portrait orientation.

# Take Away Message

- Within the lifecycle callback methods, you can declare how your activity behaves when the user leaves and re-enters the activity.
  - For example, if you're building a streaming video player, you might pause the video and terminate the network connection when the user switches to another app.
- Each callback lets you perform specific work that's appropriate to a given change of state.
- Doing the right work at the right time and handling transitions properly make your app more robust and performant.

# CSE 162 Mobile Computing

## Lecture 6: Intents and Fragments

Hua Huang

# **Saving State Data**

# Activity Destruction

- App may be destroyed
  - On its own by calling finish
  - If user presses **back button and quit it**
- Before Activity destroyed, system calls **onSaveInstanceState**
  - Can save state required to recreate Activity later
  - E.g. Save current positions of game pieces



# **onSaveInstanceState: Saving App State**

- Systems write info about views to Bundle
- Programmer must save other app-specific information using **onSaveInstanceState( )**
  - E.g. board state in a board game such as mastermind



# onRestoreInstanceState( ): Restoring State Data

- When an Activity recreated saved data sent to **onCreate** and **onRestoreInstanceState()**
- Can use either method to restore app state data



# Saving Data Across Device Rotation

- override **onSaveInstanceState** method

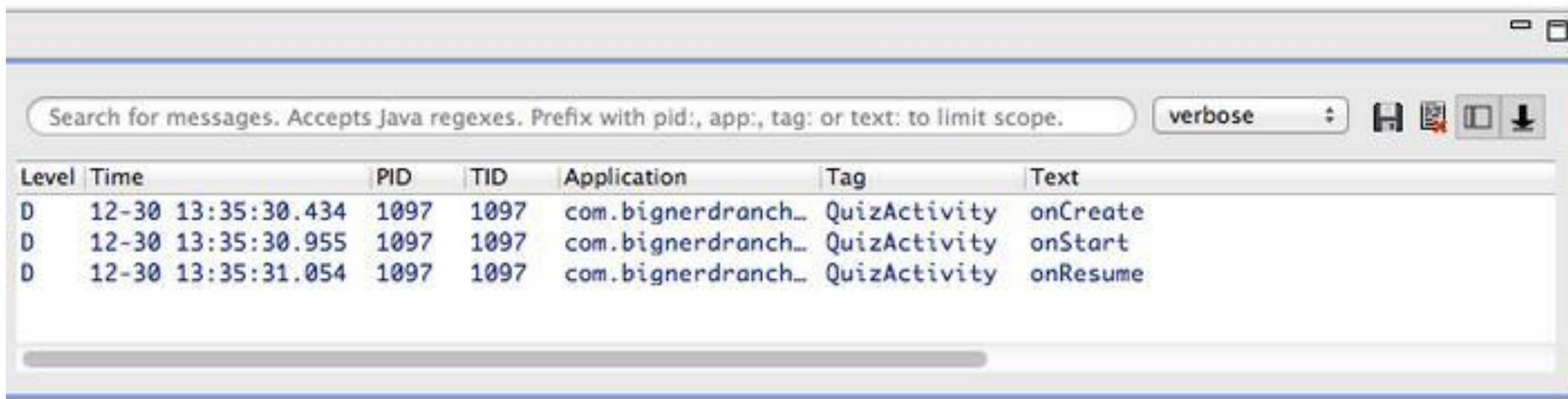
```
private static final String KEY_INDEX = "index";

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    super.onSaveInstanceState(savedInstanceState);
    Log.i(TAG, "onSaveInstanceState");
    savedInstanceState.putInt(KEY_INDEX, mCurrentIndex);
}
```

# **Logging Errors in Android**

# Logging Errors in Android

- Android can log and display various types of errors/warnings in Android Studio Window



- Error logging is in **Log** class of **android.util** package, so need to

```
import android.util.Log;
```

- Turn on logging of different message types by calling appropriate method

---

Method	Purpose
Log.e()	Log errors
Log.w()	Log warnings
Log.i()	Log informational messages
Log.d()	Log debug messages
Log.v()	Log verbose messages

---

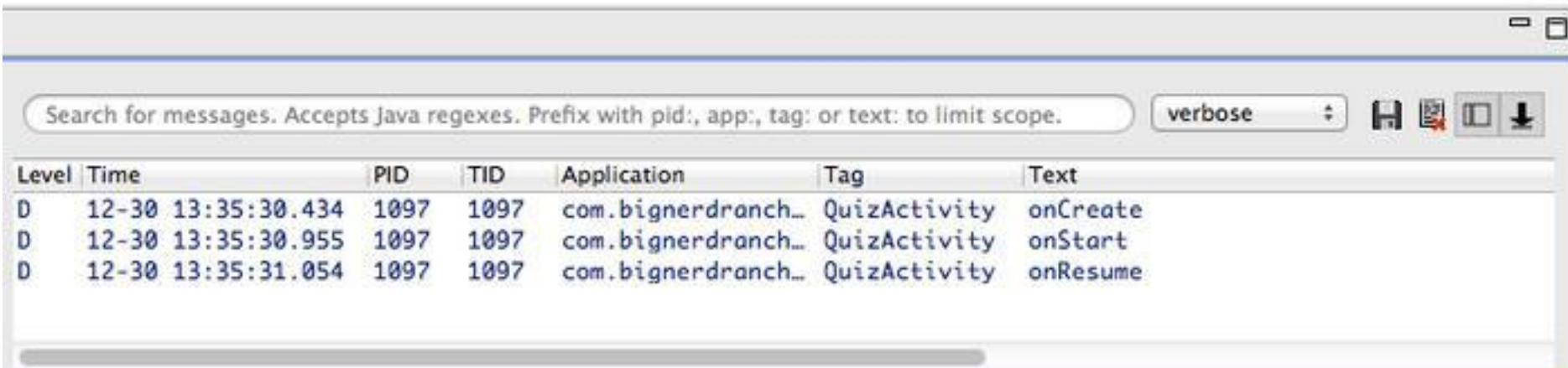
# Example

- A good way to understand Android lifecycle methods is to print debug messages in Android Studio when they are called

```
onCreate( ){  
    ... print message "OnCreate called"...
```

```
}
```

```
onStart( ){  
    ... print message "OnStart called"...
```



The screenshot shows the Android Logcat window with the following details:

- Search Bar:** Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope.
- Filter Buttons:** verbose, H, E, D, L
- Table Headers:** Level, Time, PID, TID, Application, Tag, Text
- Table Data:**

Level	Time	PID	TID	Application	Tag	Text
D	12-30 13:35:30.434	1097	1097	com.bignerdranch..	QuizActivity	onCreate
D	12-30 13:35:30.955	1097	1097	com.bignerdranch..	QuizActivity	onStart
D	12-30 13:35:31.054	1097	1097	com.bignerdranch..	QuizActivity	onResume

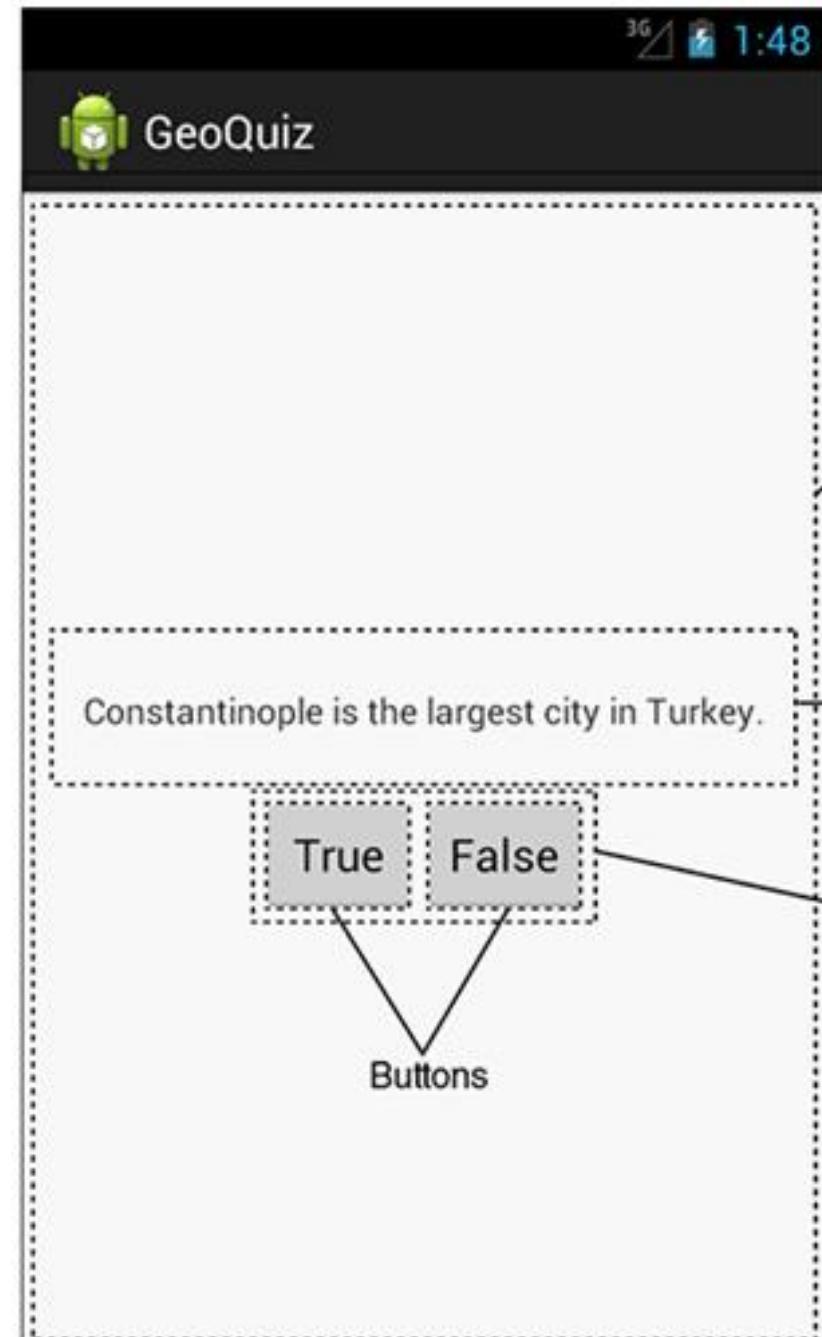
- Example: print debug message from onCreate method below

```
package com.bignerdranch.android.geoquiz;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;

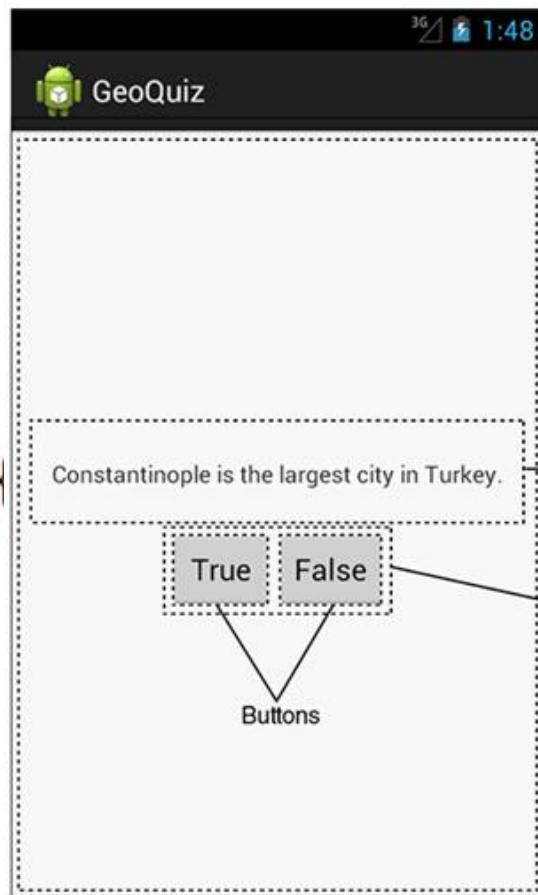
public class QuizActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);
    }
}
```



- Add the debug message

```
public class QuizActivity extends Activity {  
    ...  
  
    @Override  
    public void onCreate(Bundle savedInstanceState)  
        super.onCreate(savedInstanceState);  
        Log.d(TAG, "onCreate(Bundle) called");  
        setContentView(R.layout.activity_quiz);  
    ...
```



- Debug (d) messages have the form

```
public static int d(String tag, String msg)
```

- E.g.

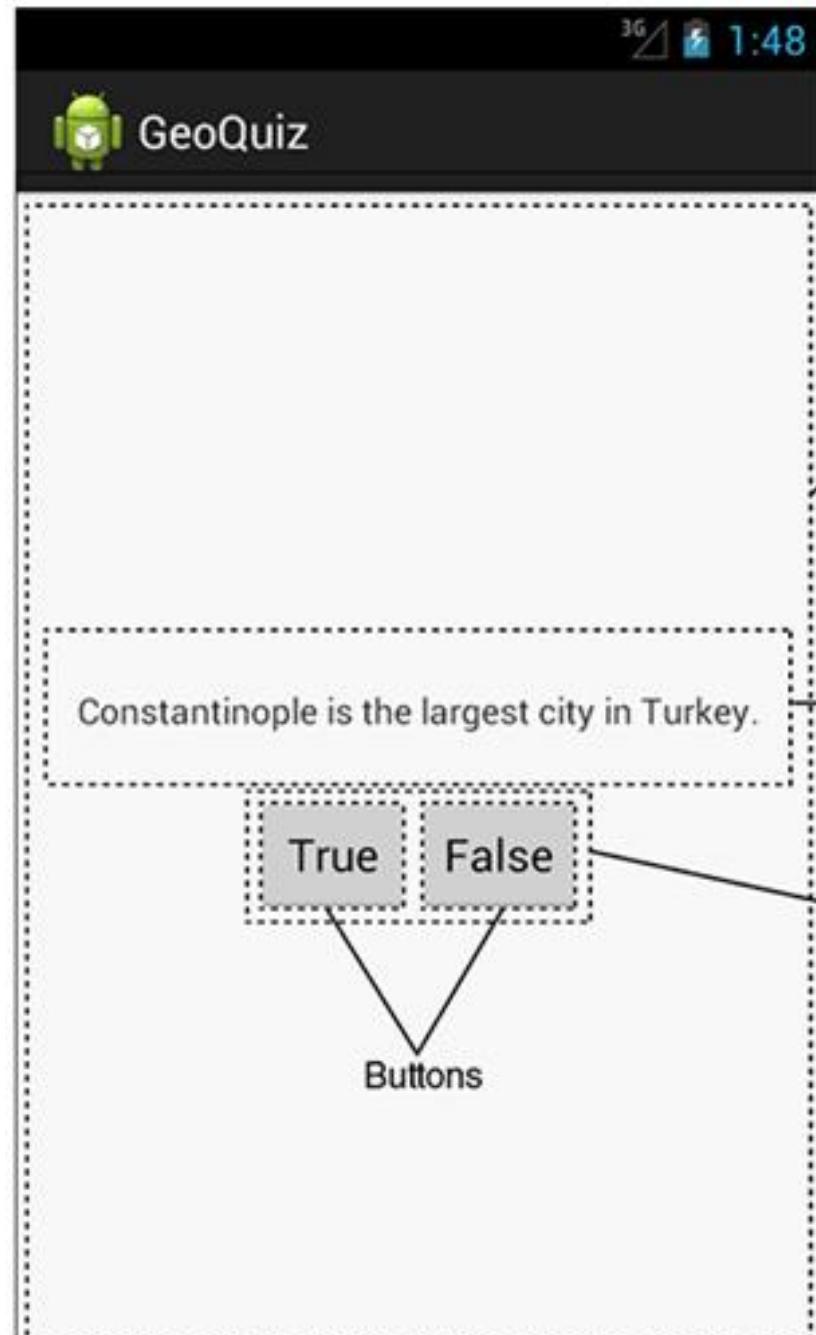
Tag                      Message  
↓                        ↓  
QuizActivity: onCreate(Bundle) called

- Example declaration:

```
Log.d(TAG, "onCreate(Bundle) called");
```

- Then declare string for TAG

```
public class QuizActivity extends Activity {  
  
    private static final String TAG = "QuizActivity";  
  
    ...  
  
}
```



## Print debug messages from each method

```
    } // End of onCreate(Bundle)

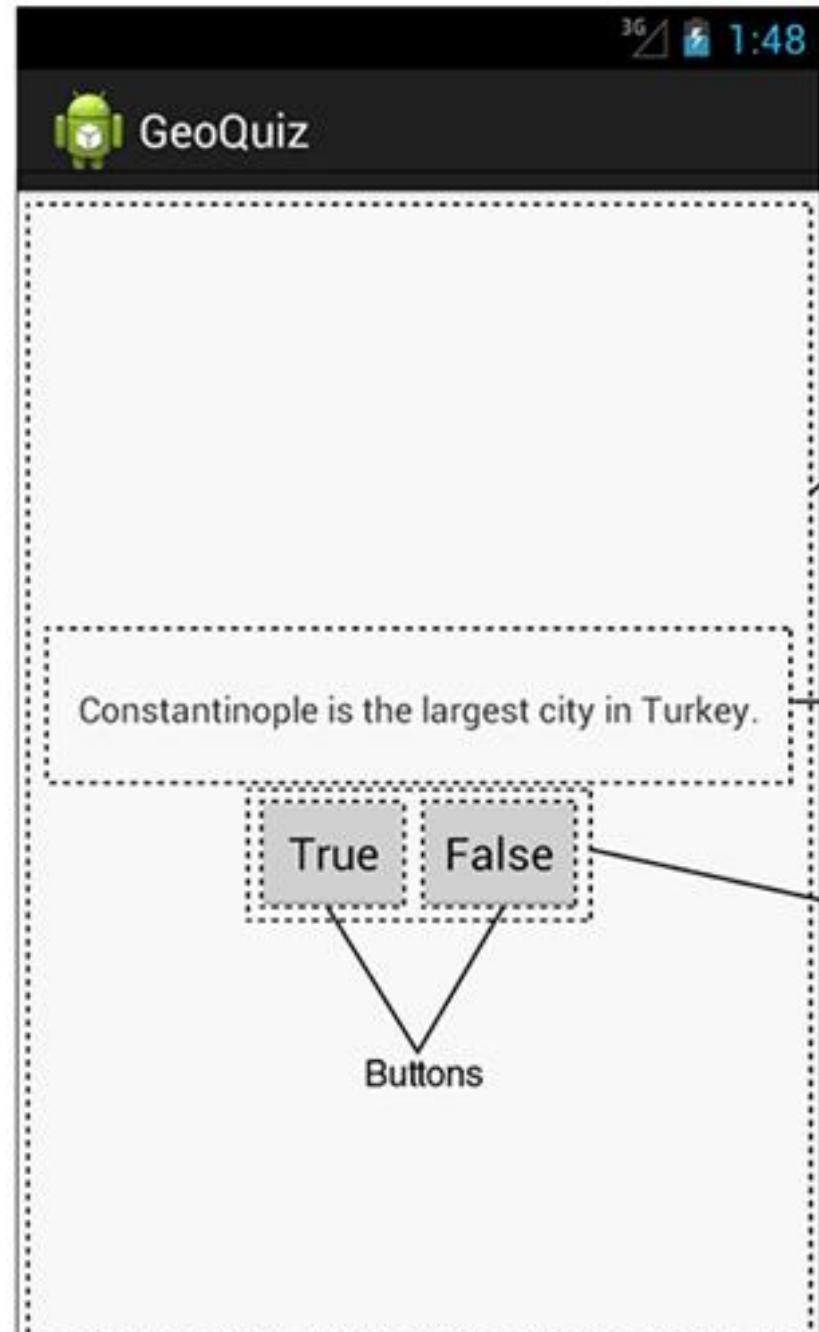
    @Override
    public void onStart() {
        super.onStart();
        Log.d(TAG, "onStart() called");
    }

    @Override
    public void onPause() {
        super.onPause();
        Log.d(TAG, "onPause() called");
    }

    @Override
    public void onResume() {
        super.onResume();
        Log.d(TAG, "onResume() called");
    }

    → @Override
    public void onStop() {
        super.onStop();
        Log.d(TAG, "onStop() called");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy() called");
    }
```

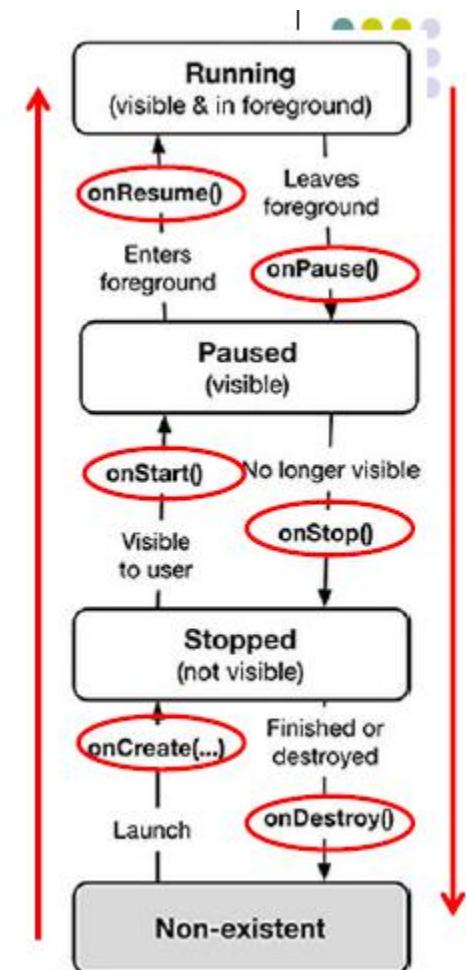


- Homework: Run the program, make sure you see all the debug messages

Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope.

verbose

Level	Time	PID	TID	Application	Tag	Text
D	12-30 13:35:30.434	1097	1097	com.bignerdranch..	QuizActivity	onCreate
D	12-30 13:35:30.955	1097	1097	com.bignerdranch..	QuizActivity	onStart
D	12-30 13:35:31.054	1097	1097	com.bignerdranch..	QuizActivity	onResume



# Intents

# What is an Intent?

- Intent is an intention to do something
- Intent contains an action carrying some information
- Intent is used to communicate between android components
  - Start an activity
  - Start a service
  - Deliver a broadcast

# Example

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType(HTTP.PLAIN_TEXT_TYPE); // "text/plain" MIME type

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getApplicationContext()) != null) {
    startActivity(sendIntent);
}
```

# Why intent?

- Intent is used to communicate, share data between components
- Intent contains the following things
  - Component name
  - Action
  - Data
  - Extras
  - Category
  - flags

# Intent Types

- Explicit and Implicit Intents

# Explicit Intents

- **Explicit Intent:** If components sending and receiving Intent are in same app
  - E.g. Activity A starts Activity B in same app
  - Activity A explicitly says what Activity (B) should be started
- It's faster
- Used if you know the specific activity to perform

# Explicit Intent Example

```
// Executed in an Activity, so 'this' is the Context  
// The fileUrl is a string URL, such as "http://www.example.com/image.png"  
Intent downloadIntent = new Intent(this, DownloadService.class);  
downloadIntent.setData(Uri.parse(fileUrl));  
startService(downloadIntent);
```

# Implicit Intents

- **Implicit Intent:** If components sending and receiving Intent are in **different apps**
  - Activity B specifies what ACTION it needs done, doesn't specify Activity to do it
    - Example of Action: take a picture, any camera app can handle this
- Useful when your app cannot perform the action
  - But other apps can do it
- Exception Handling: it is possible there isn't any app that handles your implicit intent

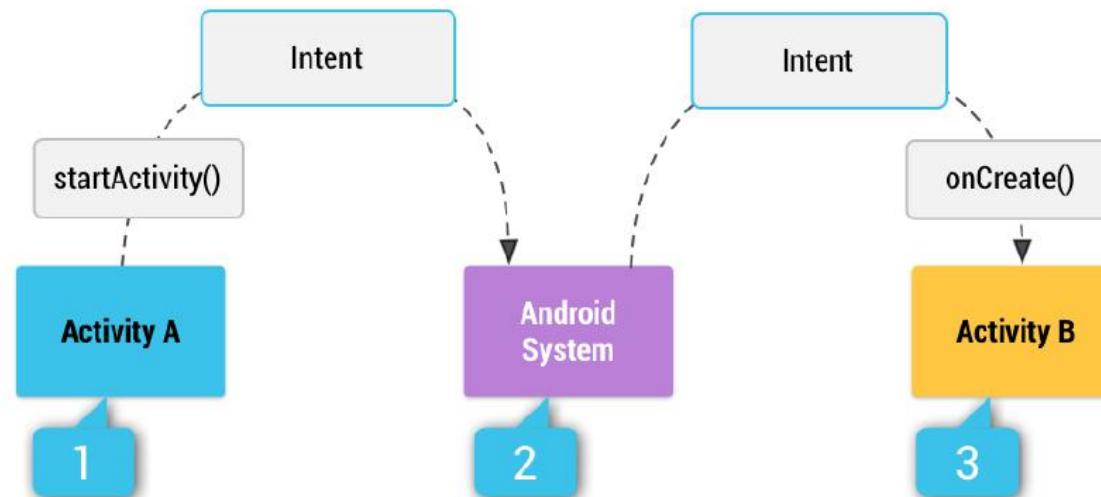
# Example

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType(HTTP.PLAIN_TEXT_TYPE); // "text/plain" MIME type

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getApplicationContext()) != null) {
    startActivity(sendIntent);
}
```

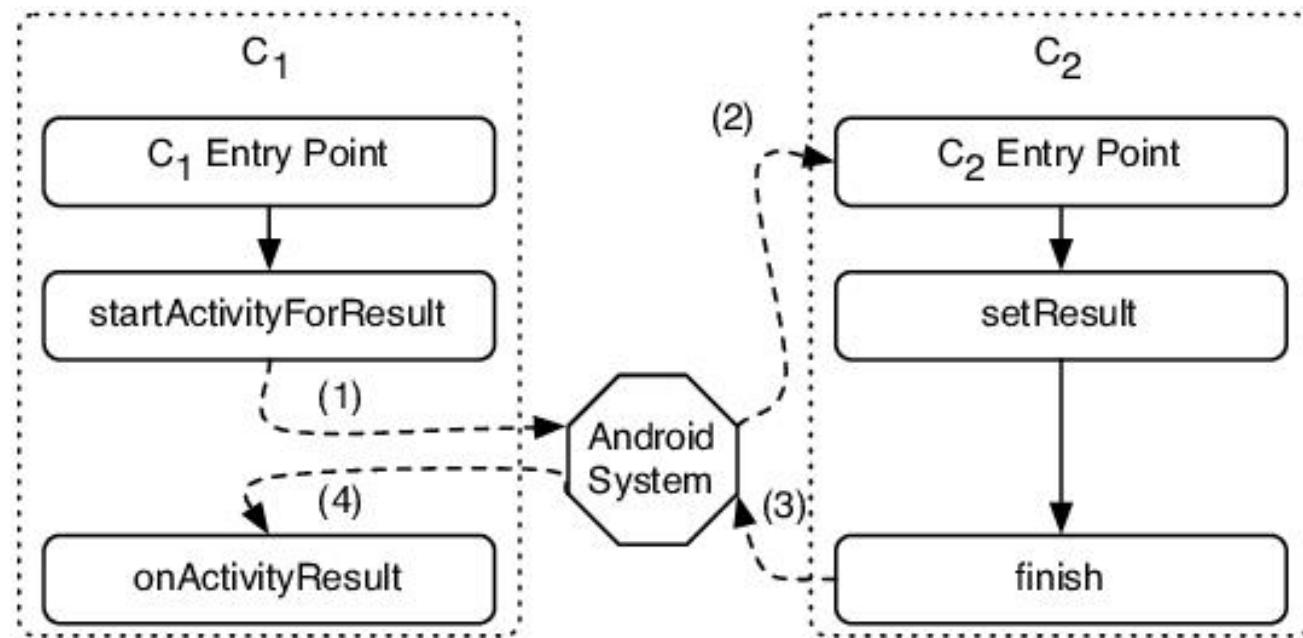
# Intent Uses

- 3 main use cases for Intents
- **Case 1 (Activity A starts Activity B, no result back):**
  - Call **startActivity( )**, pass an Intent
  - Intent has information about Activity to start, plus any necessary data



# Intent: Result Received Back

- **Case 2 (Activity A starts Activity B, gets result back):**
  - Call `startActivityForResult( )`, pass an Intent
  - Separate Intent received in Activity A's `onActivityResult( )` callback



# Intent: Result Received Back

- **Case 3 (Activity A starts a Service):**
  - E.g. Activity A starts service to download big file in the background
  - Activity A calls **StartService( )**, passes an Intent
  - Intent contains information about Service to start, plus any necessary data

# Explicit Activity launch

## Manifest Activity Registration

```
<activity android:name=".ViewActivity" android:label="View Tests">
    <intent-filter><action android:name =
        "com.acorns.intent.action.ShowView"/>
        <category android:name ="android.intent.category.DEFAULT" />
    </intent-filter></activity>
```

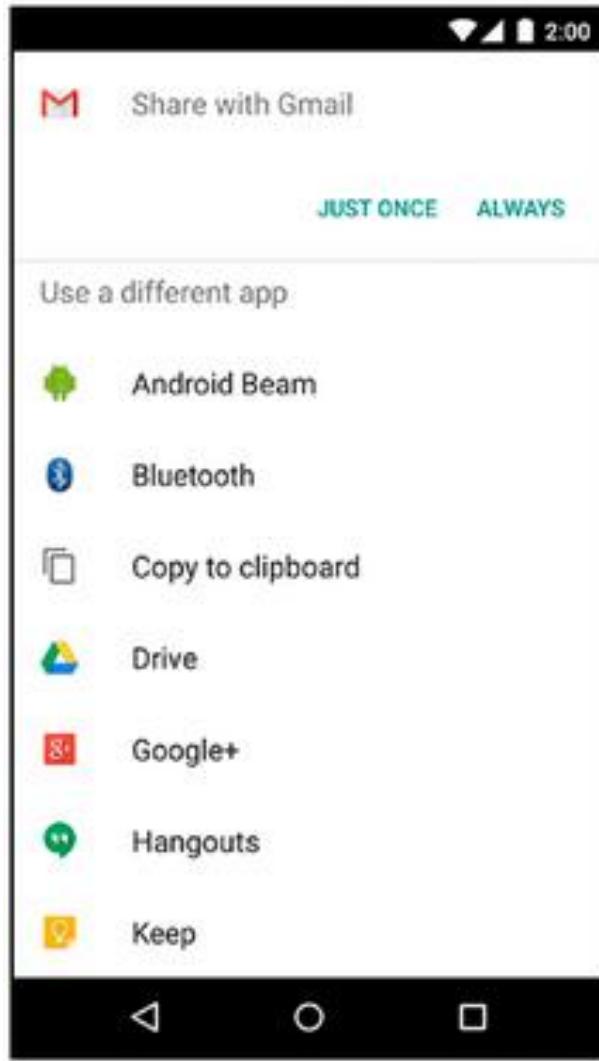
## Basic Java Code

```
public class ViewActivity extends Activity
{
    @Override public void onCreate(Bundle state)
    {
        super.onCreate(state);
        setContentView(R.layout.main);
        Intent intent = this.getIntent();
        if (intent==null) { log.d("ViewActivity","invoked without an intent"); }
    }
}
```

Default category  
Needed to allow implicit intents

## Launch

```
parentActivity.startActivity(new
Intent("com.acorns.intent.action.ShowView"));
```



```
mReportButton.setOnClickListener  
 ( new View.OnClickListener() {  
     public void onClick(View v) {  
         Intent i = new  
             Intent(Intent.ACTION_SEND);  
         i.setType("text/plain");  
         i.putExtra.EXTRA_TEXT,  
             getString(  
                 r.string.crime_report_subject));  
         startActivity(i);  
     }  
 })
```

Launching the Implicit Intent

# Intents with Results

**// Launch the sub activity expecting a result**

```
private static final int SELECT_HORSE = 1, SELECT_GUN = 2;  
private void startSubActivity(int option)  
{    startActivityForResult(new Intent(this, Other.class), option); }
```

**// Receive Result with a call back**

```
@Override  
public void onActivityResult(int requestCode, int resultCode, Intent data)  
{    super.onActivityResult(requestCode, resultCode, data);  
    if (resultCode == Activity.RESULT_OK)  
    {        switch(requestCode)  
        {            case (SELECT_HORSE): selectedHorse = data.getData(); break;  
            case (SELECT_GUN):     selectedGun = data.getData(); break;  
        }    } }
```

# How Intents are received

- Receiving Implicit Intents
- Receiving Explicit Intents

# Implicit Intent Reception

- In your manifest.xml file, declare what intents your app can handle
  - Use <intent-filter> tag
- You can mention three elements in <intent-filter>
  - Action
  - Data
  - category

# Implicit Intent Reception

- **Action:** Action to be performed must match a registered a manifest intent filter
- **Data:** Detailed specifications of data that must match in intent filter
  - **Host:** for example, google.com
  - **Mime type:** vnd.android.cursor.dir/ or vnd.android.cursor.item/ matches all or specific rows of an android SQL cursor
  - **Path, Path Pattern, Path Prefix:** match if specified (ex: /data authority/note)
  - **Data authority:** match filter specification (ex: com.google.provider.NotePad)
  - **Port:** listening port on the host machine
  - **Scheme:** examples: content or http

# Implicit Intent Reception

**Intent Categories:** Indicate to Android the general nature of an activity

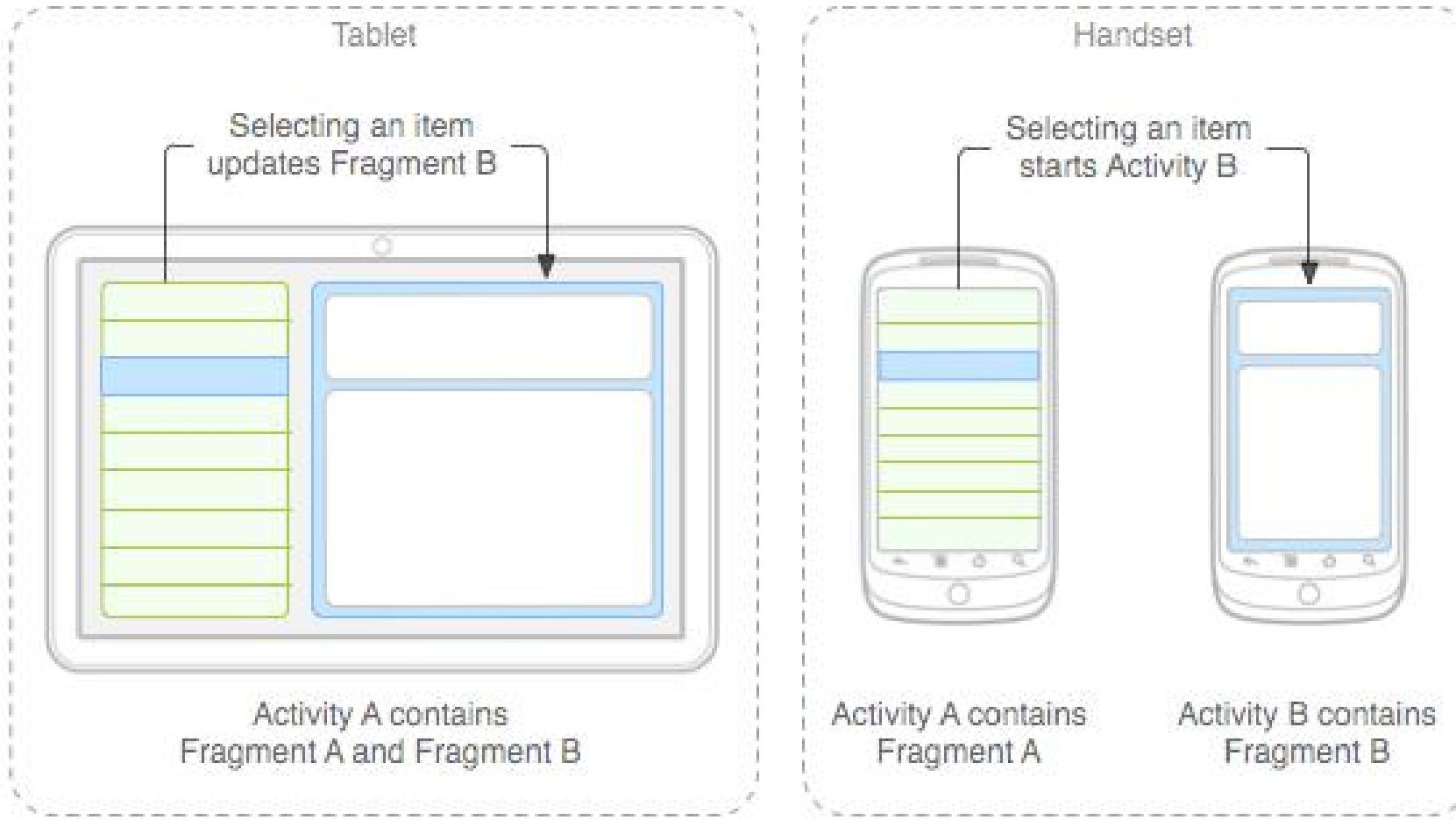
- `CATEGORY_DEFAULT`: Can invoke through implicit intents
- `CATEGORY_BROWSABLE`: Will not violate browser security restrictions
- `CATEGORY_TAB`: Embeddable in a tabbed view of a parent activity
- `CATEGORY_ALTERNATIVE`: Alternative action to displayed data
- `CATEGORY_SELECTED_ALTERNATIVE`: Alternative action to selected data
- `CATEGORY_LAUNCHER`: included on launcher screen lists
- `CATEGORY_HOME`: The home screen view (One per device)
- `CATEGORY_PREFERENCE`: Shown on the preference screen

# Lecture 7: Fragments, and Adaptive Layouts

# Fragments

# Fragment

- A Fragment represents a behavior or a portion of user interface in an Activity.
- You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities.
- You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities).



# Fragment

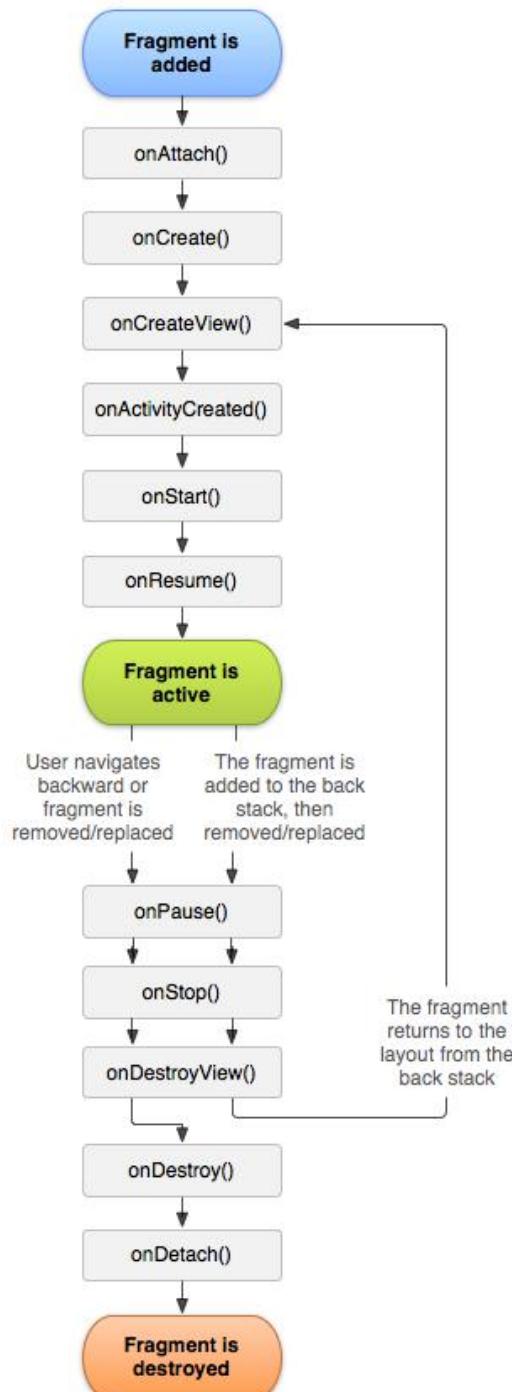
- When you have a larger screen device than a phone –like a tablet it can look too simple to use phone interface here.
- **Fragments**
  - Mini-activities, each with its own set of views
  - One or more fragments can be embedded in an Activity
  - You can do this dynamically as a function of the device type (tablet or not) or orientation

# How to Use Fragments

- First of all decide how many fragments you want to use in an activity. For example let's we want to use two fragments to handle landscape and portrait modes of the device.
- Next based on number of fragments, create classes which will extend the *Fragment* class. The Fragment class has above mentioned callback functions. You can override any of the functions based on your requirements.
- Corresponding to each fragment, you will need to create layout files in XML file. These files will have layout for the defined fragments.
- Finally modify activity file to define the actual logic of replacing fragments based on your requirement.

# Fragment Lifecycle

- Fragment in an Activity---Activity Lifecycle influences
  - Activity paused all its fragments paused
  - Activity destroyed all its fragments are destroyed
  - Activity running manipulate each fragment independently.
- Fragment transaction add, remove, etc.
  - adds it to a **back stack** that's managed by the activity—each back stack entry in the activity is a record of the fragment transaction that occurred.
  - The back stack allows the user to reverse a fragment transaction (navigate backwards), by pressing the **Back button**.



# Reading Materials

- Read Chapter 9 Fragments

# Data-Driven Layouts

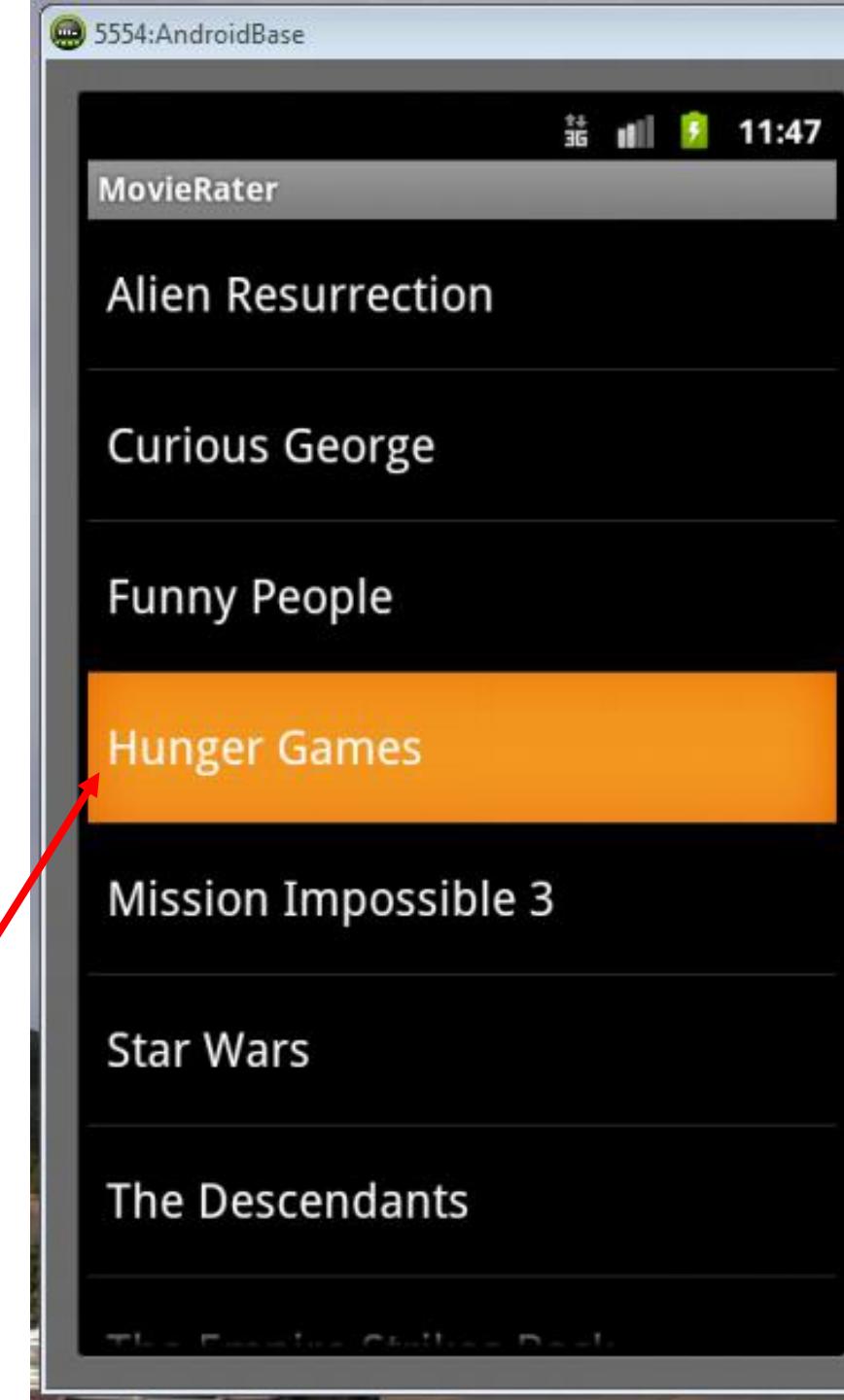
# Data-Driven Layouts

- LinearLayout, RelativeLayout, TableLayout, GridLayout useful for positioning UI elements
  - Static preset
- Other layouts dynamically composed from data (e.g. database)
  - ListView, GridView, GalleryView
  - Tabs with TabHost, TabControl



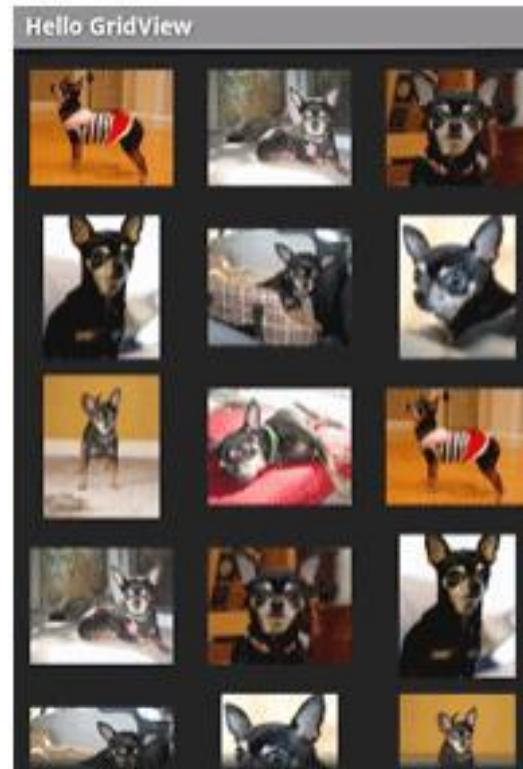
# Data Driven Layouts

- May want to populate views from a data source (XML file or database)
- Layouts that display repetitive child widgets from data source
  - ListView
  - GridView
  - GalleryView
- ListView
  - Rows of entries, pick item, vertical scroll



# Data Driven Containers

- GridView
  - List of items arranged in rows and columns
- GalleryView
  - List with horizontal scrolling, typically images



# AdapterView

- ListView, GridView, and GalleryView are sub classes of AdapterView (variants)
- **Adapter:** generates widgets from a data source, populates layout
  - E.g. Data is adapted into cells of GridView

Data

lorem  
ipsum  
dolor  
amet  
consectetuer  
adipiscing  
elit  
morbi



Adapter



# AdapterView

- Most common Adapter types:
  - **CursorAdapter**:read from database
  - **ArrayAdapter**:read from resource (e.g. XML file)

# Adapters

- When using Adapter, a layout (XML format) is defined for each child element (View)
- The adapter
  - Reads in data (list of items)
  - Creates Views (widgets) using layout for each element in data source
  - Fills the containing layout (List, Grid, Gallery) with the created Views
- Child widgets can be as simple as a TextView or more complex layouts / controls
  - simple views can be declared in a layout XML file (e.g. android.R.layout)

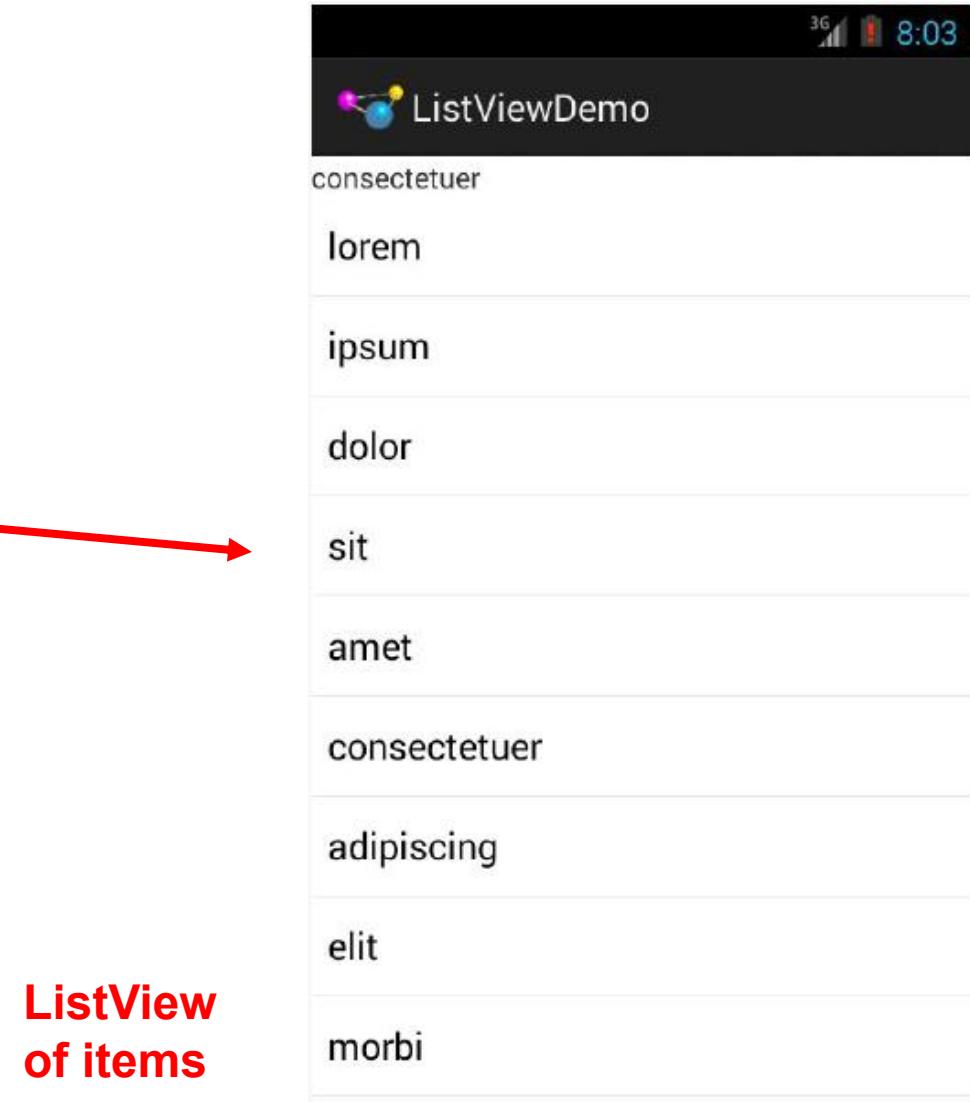


# Example: Creating ListView using AdapterArray

- Goal: create this

```
private static final String[] items={"lorem", "ipsum", "dolor",
    "sit", "amet",
    "consectetuer", "adipiscing", "elit", "morbi", "vel",
    "ligula", "vitae", "arcu", "aliquet", "mollis",
    "etiam", "vel", "erat", "placerat", "ante",
    "porttitor", "sodales", "pellentesque", "augue", "purus"};
```

Enumerated list

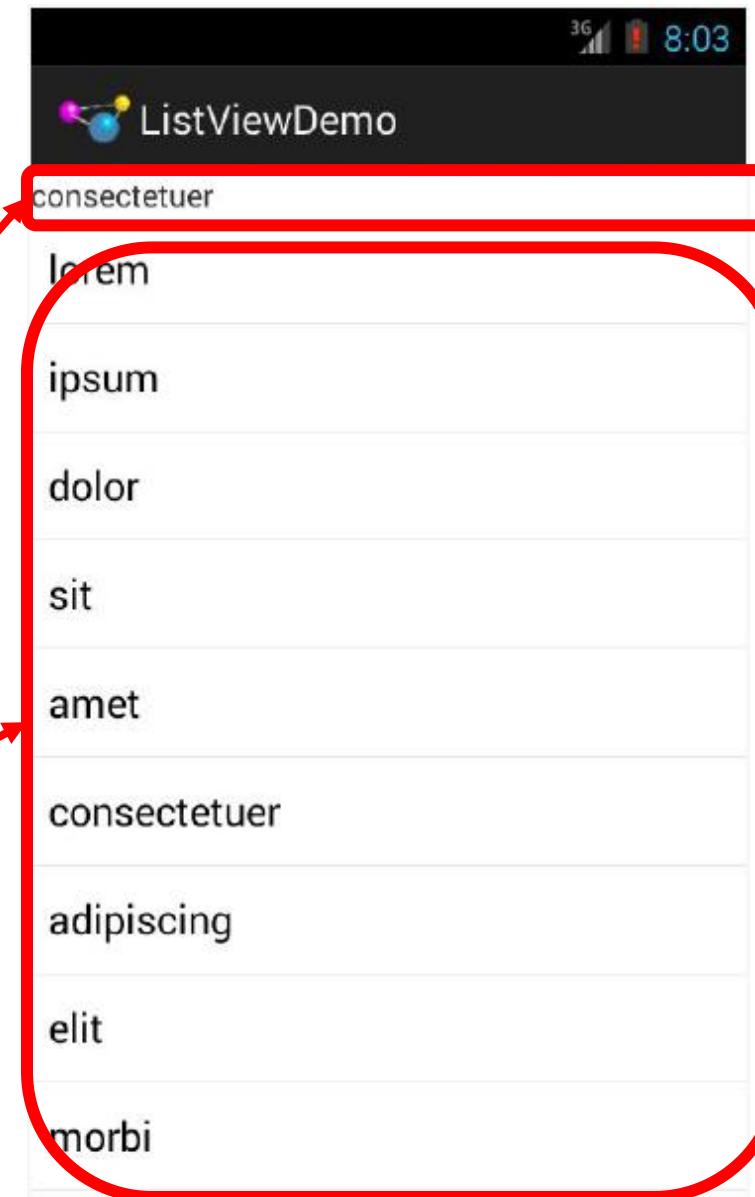


ListView  
of items

# Example: Creating ListView using AdapterArray

First create Layout file

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:id="@+id/selection"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
    <ListView
        android:id="@+android:id/list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        />
</LinearLayout>
```



# Using ArrayAdapter

Command used to wrap adapter around array of menu items or **java.util.List** instance

- E.g. **android.R.layout.simple\_list\_item\_1** turns strings into textView widgets

```
String[] items={"this", "is", "a", "really", "silly", "list"};
new ArrayAdapter<String>(this,
                           android.R.layout.simple_list_item_1,
                           items);
```

Context to use.  
(e.g app's activity)

Resource ID of  
View for formatting

Array of items  
to display

```
public class ListViewDemo extends ListActivity {  
    private TextView selection;  
    private static final String[] items={"lorem", "ipsum", "dolor",  
        "sit", "amet",  
        "consectetuer", "adipiscing", "elit", "morbi", "vel",  
        "ligula", "vitae", "arcu", "aliquet", "mollis",  
        "etiam", "vel", "erat", "placerat", "ante",  
        "porttitor", "sodales", "pellentesque", "augue", "purus"};  
  
    @Override  
    public void onCreate(Bundle icicle) {  
        super.onCreate(icicle);  
        setContentView(R.layout.main);  
        setListAdapter(new ArrayAdapter<String>(this,  
            android.R.layout.simple_list_item_1,  
            items));  
        selection=(TextView)findViewById(R.id.selection);  
    }  
  
    @Override  
    public void onListItemClick(ListView parent, View v, int position,  
        long id) {  
        selection.setText(items[position]);  
    }  
}
```

Set list adapter (Bridge Data source and views)

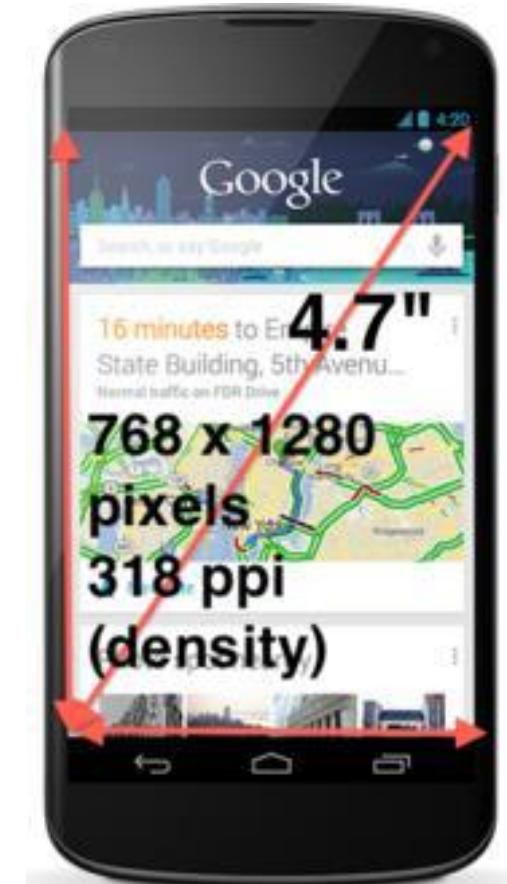
Get handle to TextView of Selected item

Change Text at top to that of selected view when user clicks on selection

# **Adaptive Image Resolution**

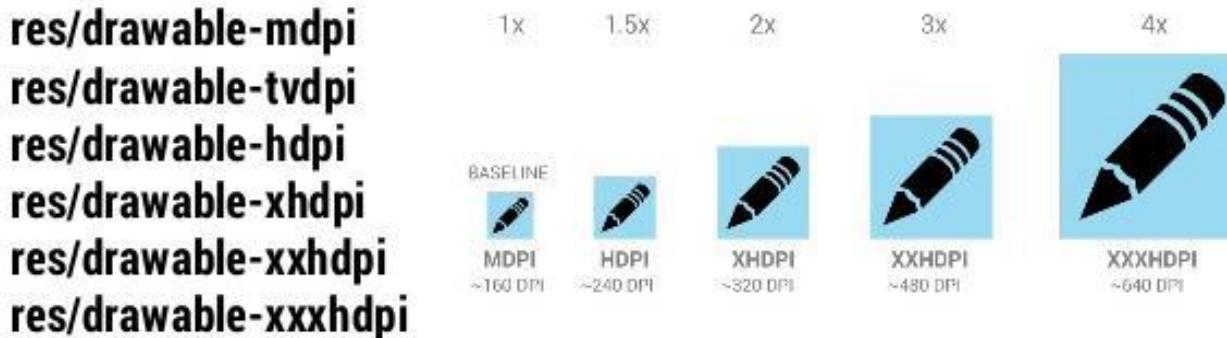
# Phone Dimensions Used in Android UI

- Physical dimensions (inches) diagonally
  - E.g. Nexus 4 is 4.7 inches diagonally
- Resolution in pixels
  - E.g. Nexus 4 resolution 768 x 1280 pixels
  - Pixels diagonally:  $\text{Sqrt}[(768 \times 768) + (1280 \times 1280)]$
- Pixels per inch (PPI) =
  - $\text{Sqrt}[(768 \times 768) + (1280 \times 1280)] / 4.7 = 318$



# Adding Pictures

- Android supports images in PNG, JPEG and GIF formats
- Put different resolutions of **same image** into different directories
  - **res/drawable-ldpi**: low dpi images (~ 120 dpi of dots per inch)
  - **res/drawable-mdpi**: medium dpi images (~ 160 dpi)
  - **res/drawable-hdpi**: high dpi images (~ 240 dpi)
  - **res/drawable-xhdpi**: extra high dpi images (~ 320 dpi)
  - **res/drawable-xxhdpi**: extra extra high dpi images (~ 480 dpi)
  - **res/drawable-xxxhdpi**: high dpi images (~ 640 dpi)



# Adding Pictures

- Use generic picture name in code (no .png, .jpg, etc)
  - E.g. to reference an image **ic\_launcher.png**
- At run-time, Android chooses which resolution/directory (e.g. –mdpi) based on phone resolution

```
<application  
    android:allowBackup="false"  
    android:icon="@drawable/ic_launcher"  
    android:label="@string/app_name"  
    android:theme="@style/AppTheme">
```

# Rotating Devices

# Rotating Device: Using Different Layouts

- Rotating device (e.g. portrait to landscape) kills current activity and creates new activity in landscape mode
- Rotation changes **device configuration**
- **Device configuration:** screen orientation/density/size, keyboard type, dock mode, language, etc.
- Apps can specify different resources (e.g. XML layout files, images) to use for different device configurations

- E.g. use different app layouts for portrait vs landscape screen orientation

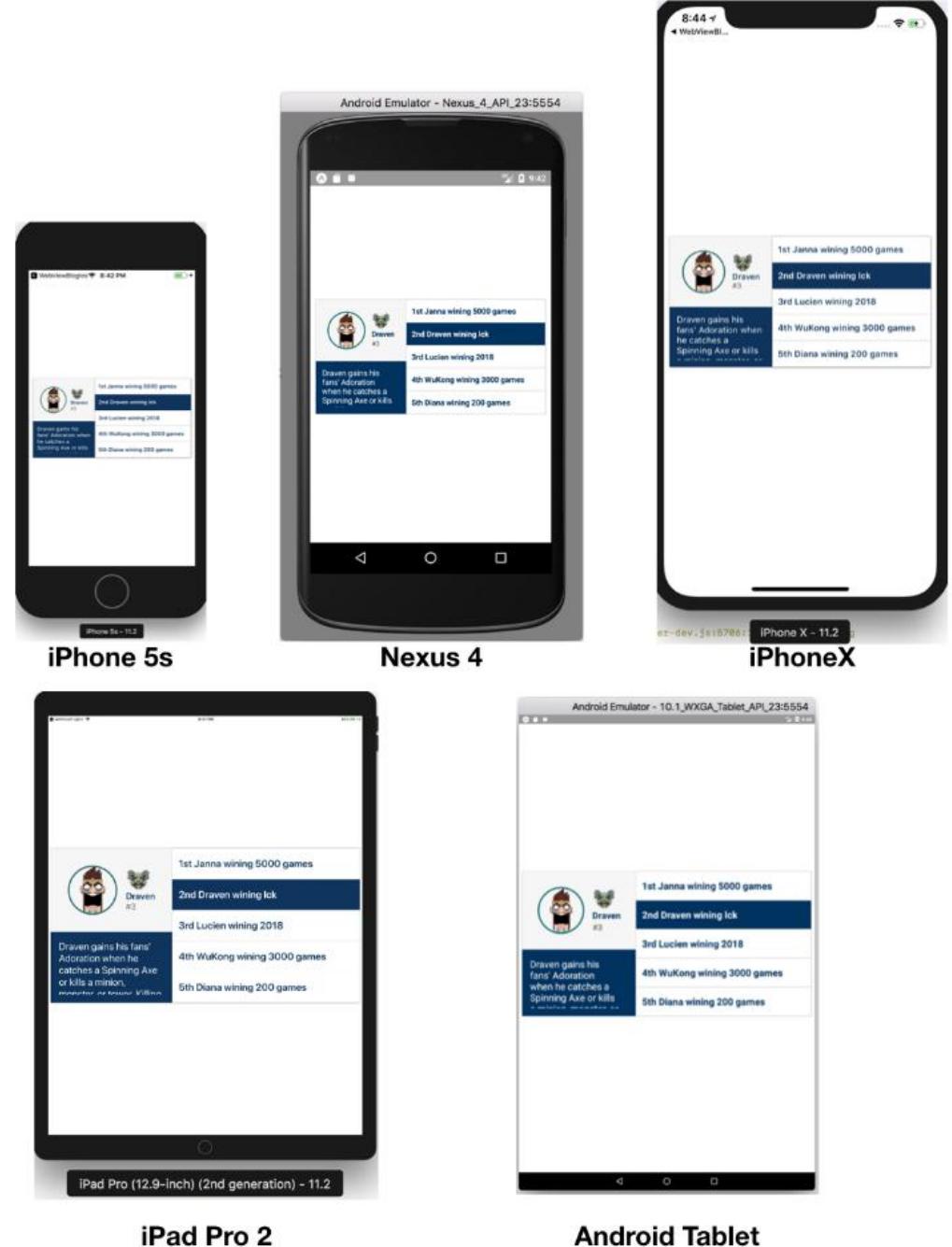
# Rotating Device: Using Different Layouts

- **Portrait:** use XML layout file in **res/layout**
- **Landscape:** use XML layout file in **res/layout-land/**
- Copy XML layout file (`activity_quiz.xml`) from **res/layout** to **res/layout-land/** and customize it
- If configuration changes, current activity destroyed, **onCreate -> setContentView (R.layout.activity\_quiz)** called again
- `onCreate` called whenever user switches between portrait and landscape

# **Mobile HCI**

# Mobile HCI

- Mobile HCI is important for an enjoyable user experience
- Can't just reuse screens originally designed for desktops. Why?
  1. Mobile screen is small, need to manage space better
  2. Does your screen look good on wide variety of mobile screen sizes?
  3. Can users reach buttons with one hand on different resolutions?
  4. Mobile device will be carried into various adverse conditions. E.g.
    1. Do colors work well indoor vs outdoor, bright vs dim light
    2. Are buttons big enough for frozen hands during winter vs summer?



# Mobile HCI: Evaluation

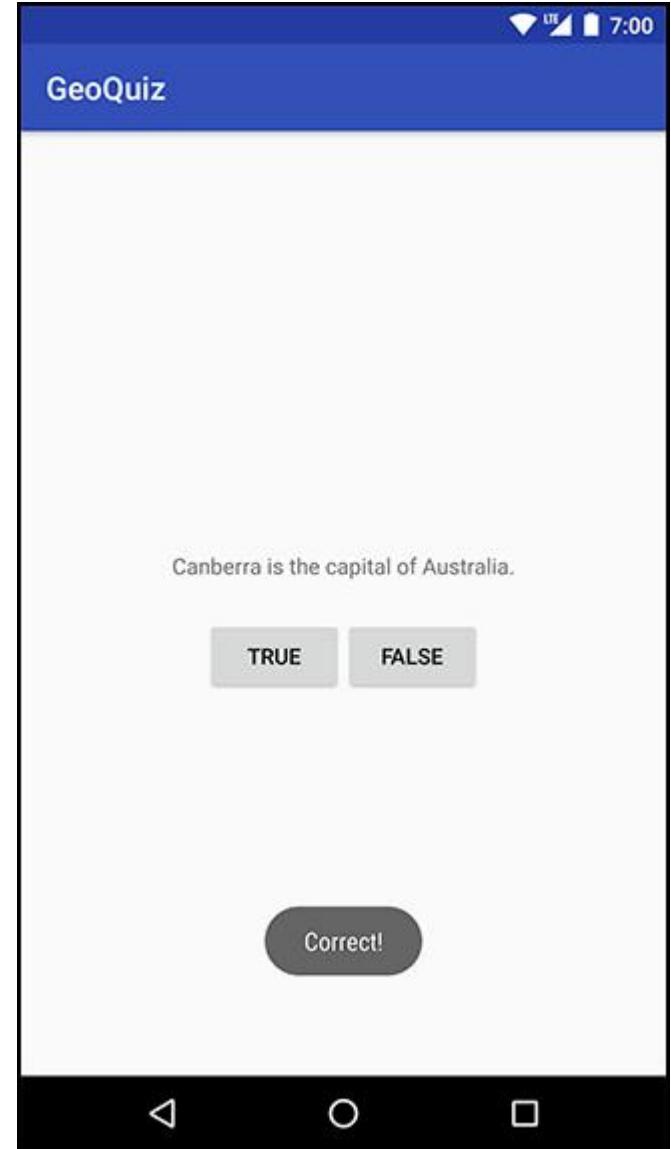
- App evaluation: iterative, user-centered
  - In lab (small) then in the field (large)
  - On wide variety of devices
    - Most poor ratings on Google Play app store are “doesn’t work on my device”
- Example: Android mobile developer tests each game on over 400 different smartphones and tablets
  - Screens
  - Aspect ratios
  - Form factors
  - Controls
  - OS versions
  - CPU/GPU
  - OpenGL/DirectX



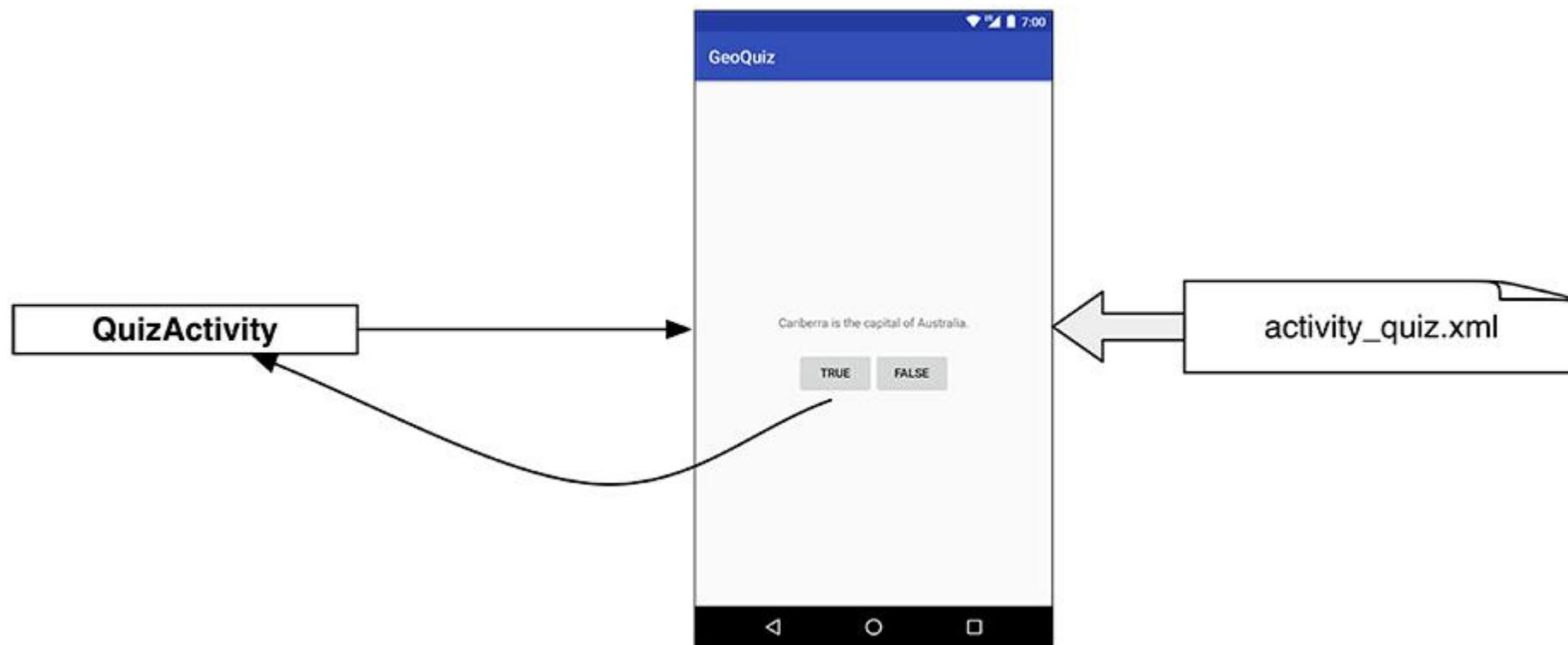
# A UI Design Example

# GeoQuiz App

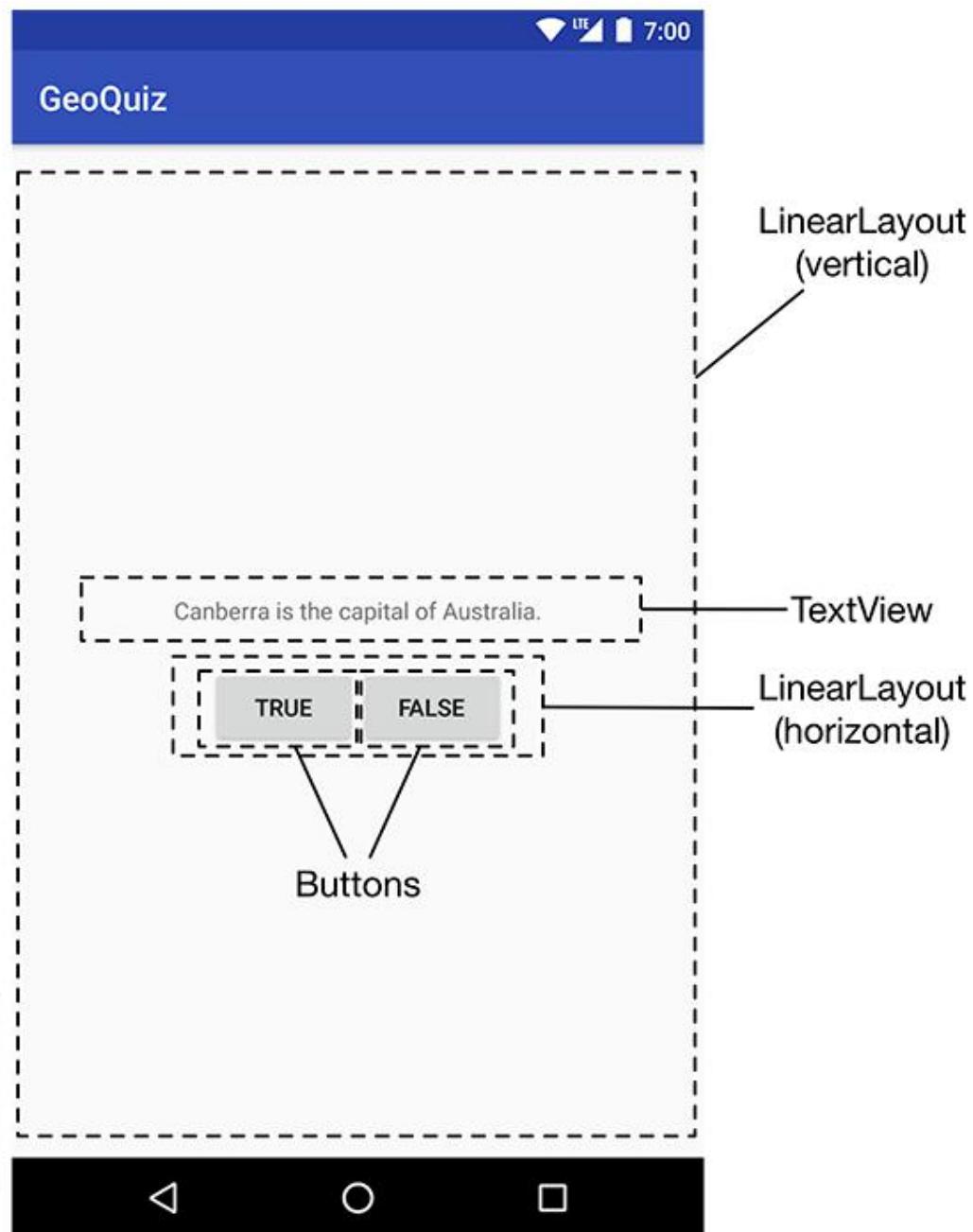
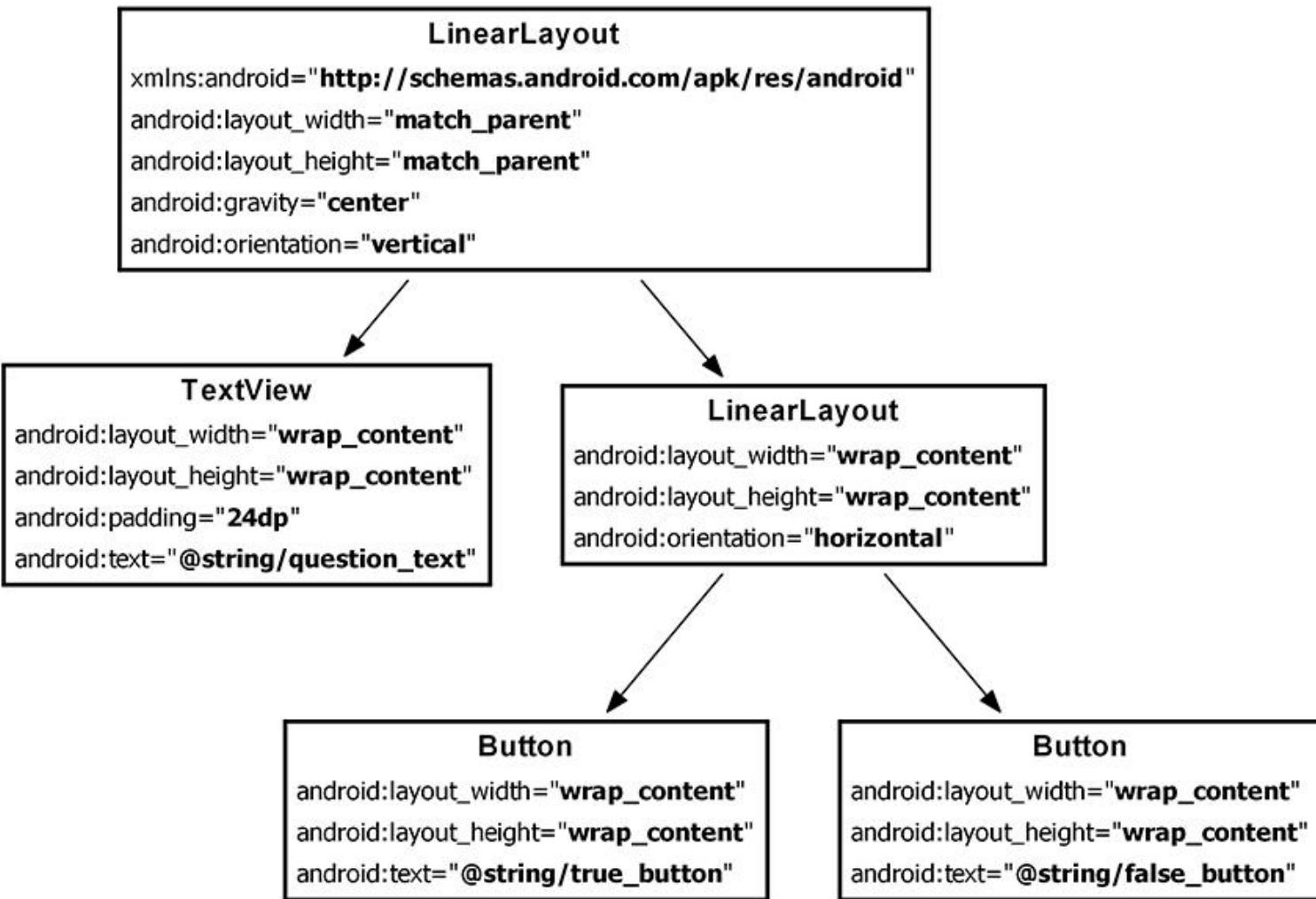
- App presents questions to test user's knowledge of geography
- User answers by pressing True or False buttons



- 2 main files:
  - `activity_quiz.xml`: to format app screen
  - `QuizActivity.java`: To present question, accept True/False response
- `AndroidManifest.xml` lists all app components, auto-generated



- 5 Widgets arranged hierarchically



- activity\_quiz.xml File listing

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical" >

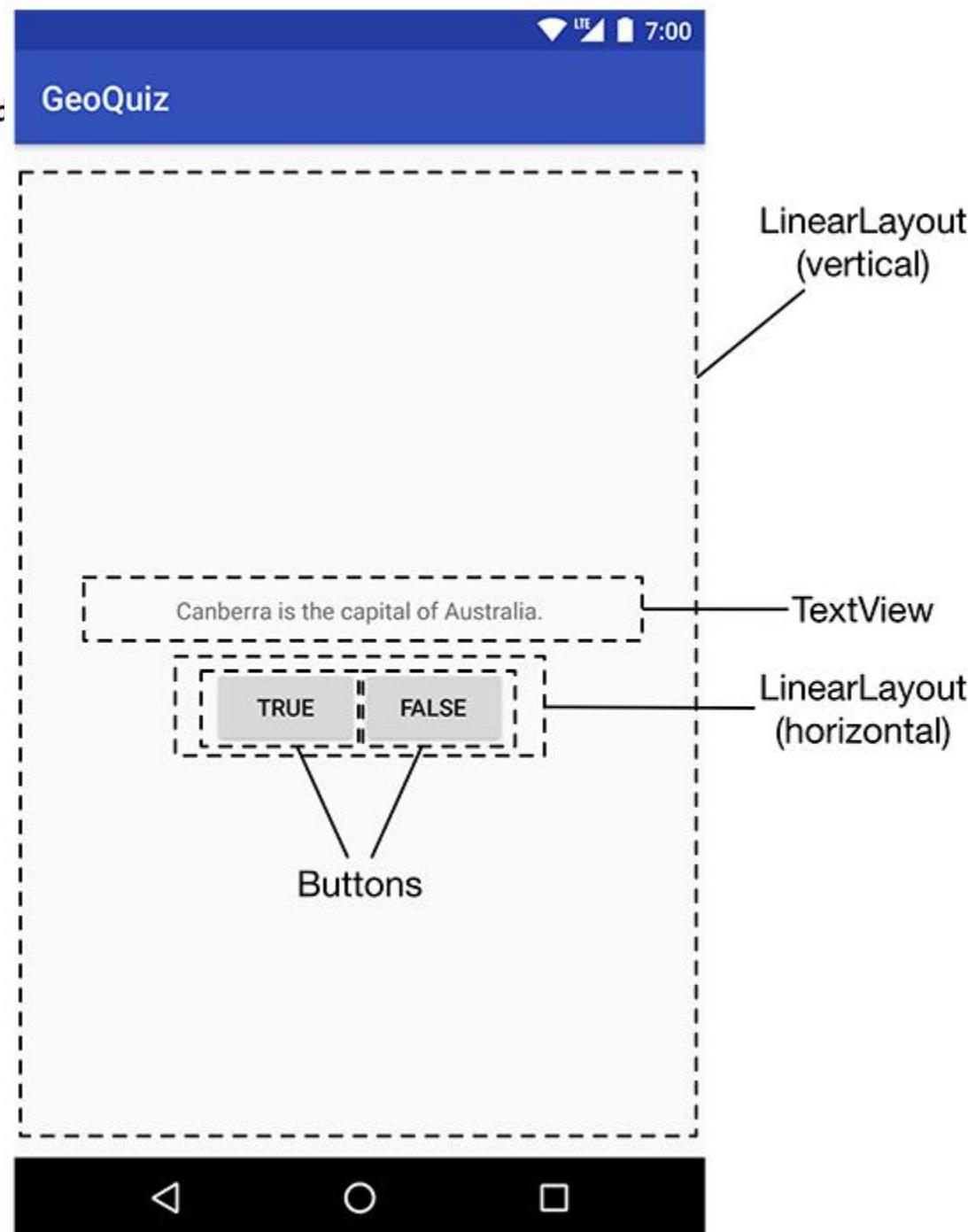
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/question_text" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/true_button" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/false_button" />

    </LinearLayout>
</LinearLayout>
```



- strings.xml File listing

Define all strings app will use

- Question: “Canberra is..”
- True
- False

```
<resources>
    <string name="app_name">GeoQuiz</string>
    <string name="question_text">Canberra is the capital of Australia.</string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
</resources>
```

- QuizActivity.java

```
package com.bignerdranch.android.geoquiz;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;

public class QuizActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);
    }
}
```

Would like java code to respond to  
True/False buttons being clicked

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
... >

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="24dp"
    android:text="@string/question_text" />

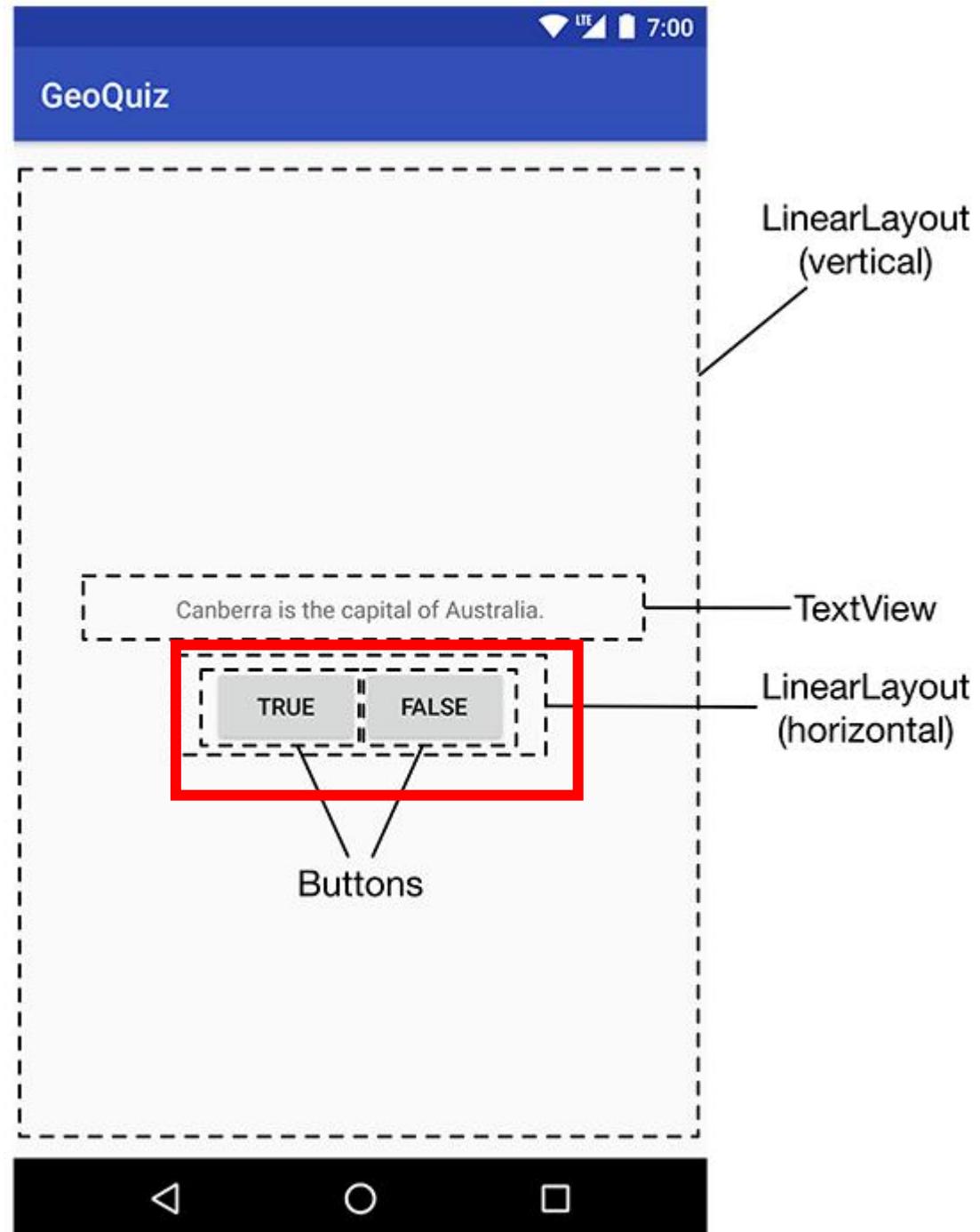
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <Button
        android:id="@+id/true_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/true_button" />

    <Button
        android:id="@+id/false_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/false_button" />

</LinearLayout>

</LinearLayout>
```



- Responding to button clicks

```
<Button  
    android:onClick="someMethod"  
    ...  
/>
```

```
public void someMethod(View theButton) {  
    // do something useful here  
}
```

# Adding a Toast

- A toast is a short pop-up message
- Does not require any input or action
- After user clicks True or False button, our app will pop-up a toast to inform the user if they were right or wrong
- First, we need to add toast strings (Correct, Incorrect) to strings.xml

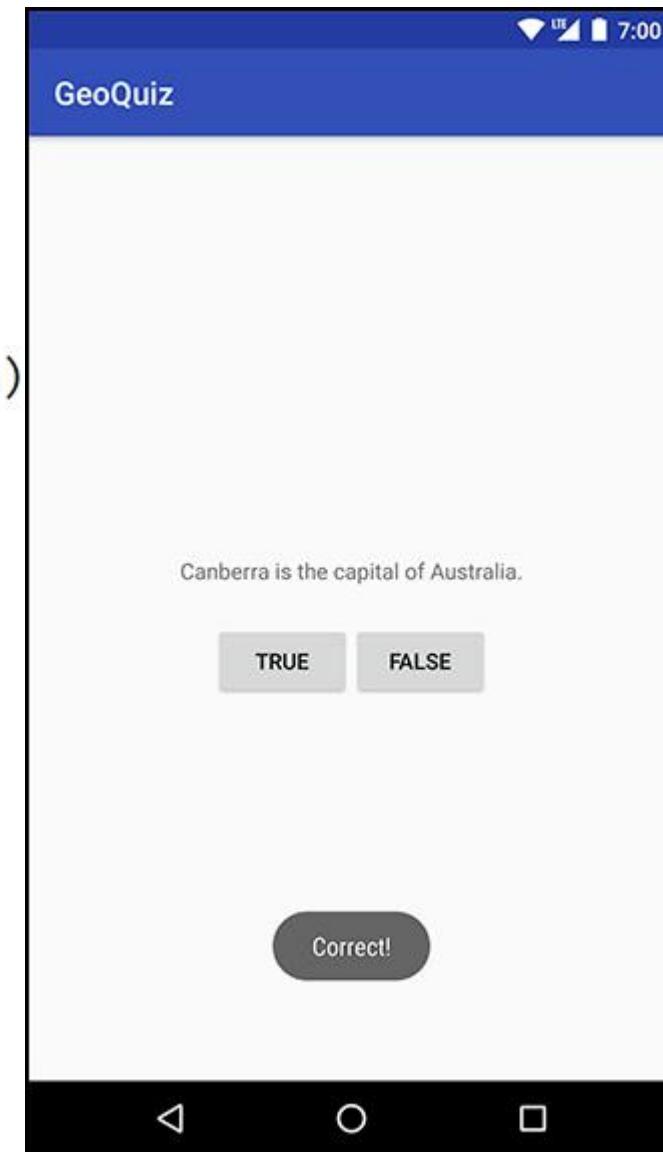
```
<resources>
    <string name="app_name">GeoQuiz</string>
    <string name="question_text">Canberra is the capital of Australia.</string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
    <string name="correct_toast">Correct!</string>
    <string name="incorrect_toast">Incorrect!</string>
</resources>
```

- To create a toast, call the method:

```
public static Toast.makeText(Context context, int resId, int duration)
```

- After creating toast, call toast.show( ) to display it
- For example to add a toast to our onClick( ) method:

```
public void onClick(View v) {  
    Toast.makeText(QuizActivity.this,  
        R.string.incorrect_toast,  
        Toast.LENGTH_SHORT).show();  
}
```



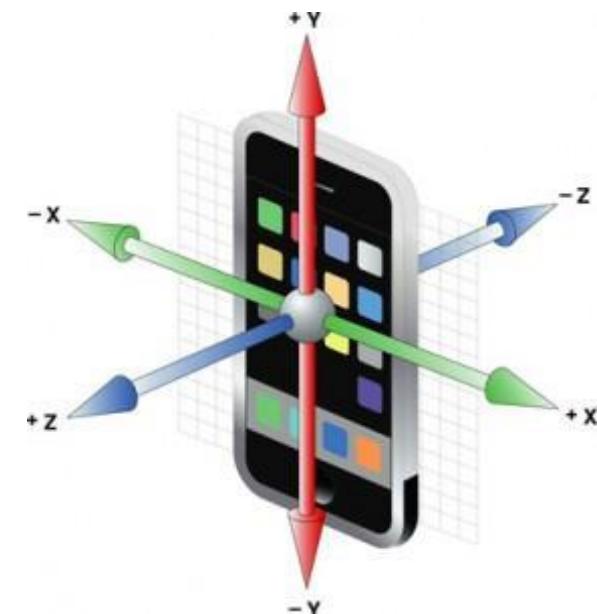
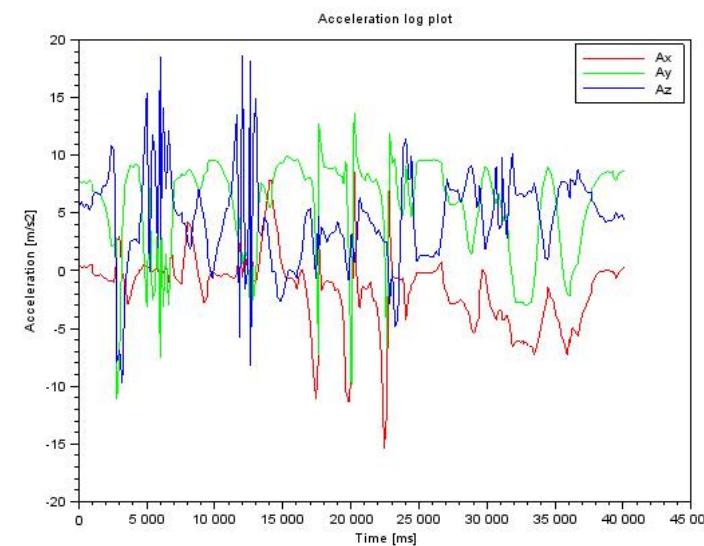
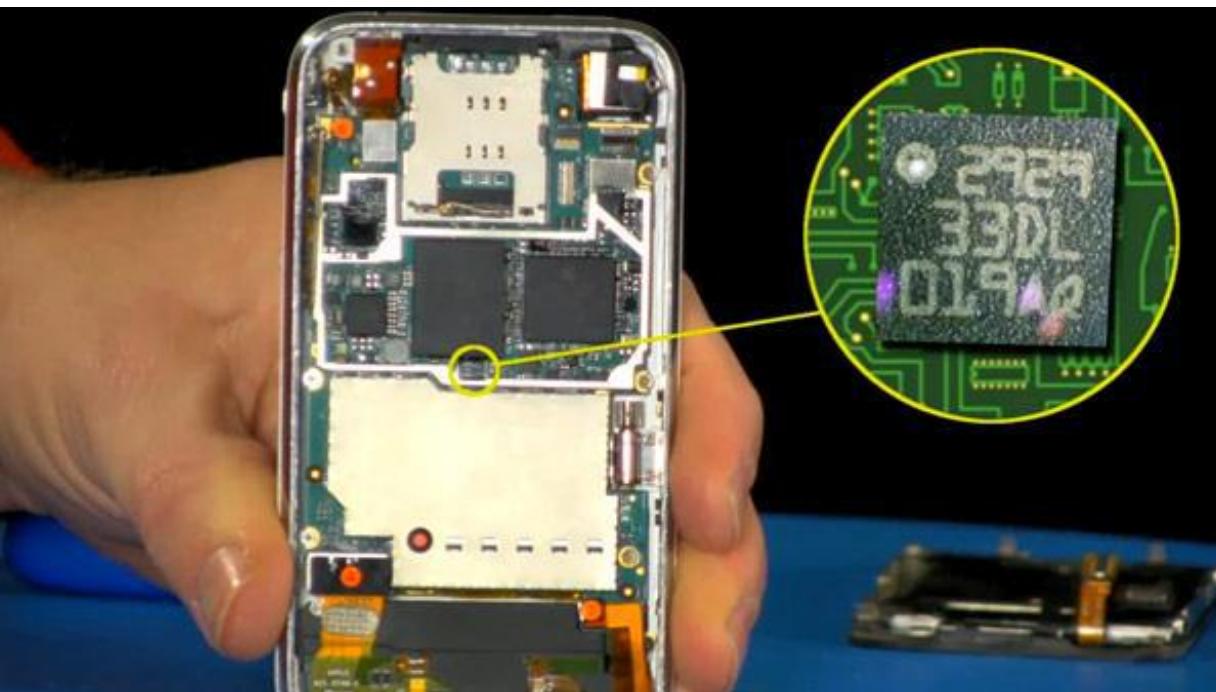


# CSE 162 Mobile Computing Mobile Sensing

Hua Huang

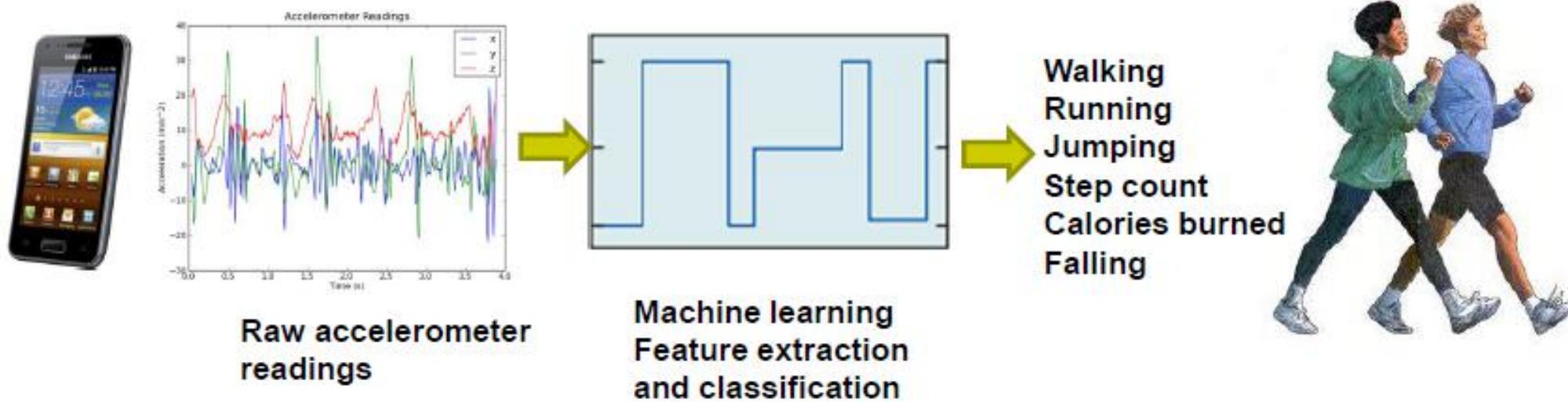
# What is a Sensor?

- 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)

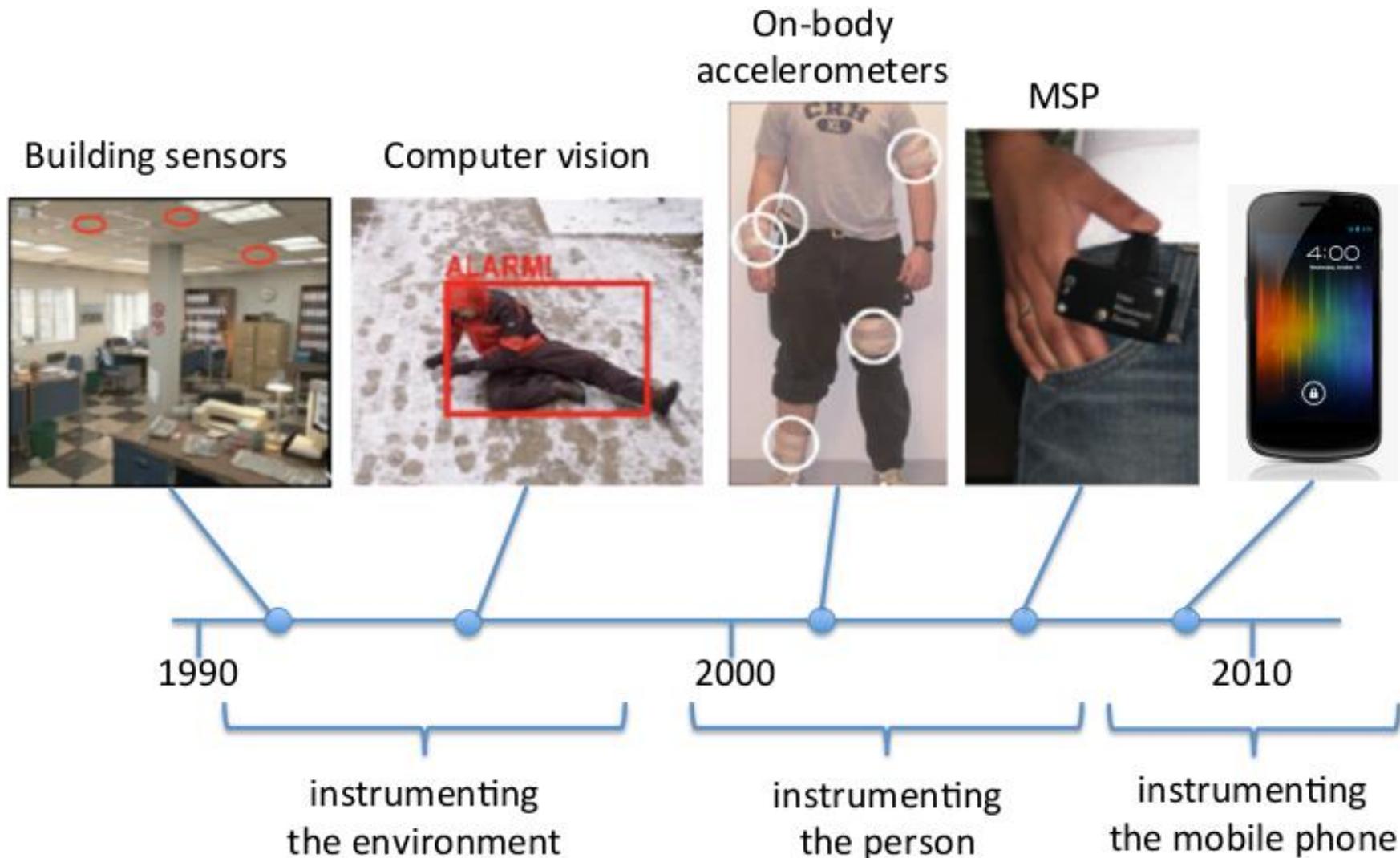
# Definitions

- 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



# 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:

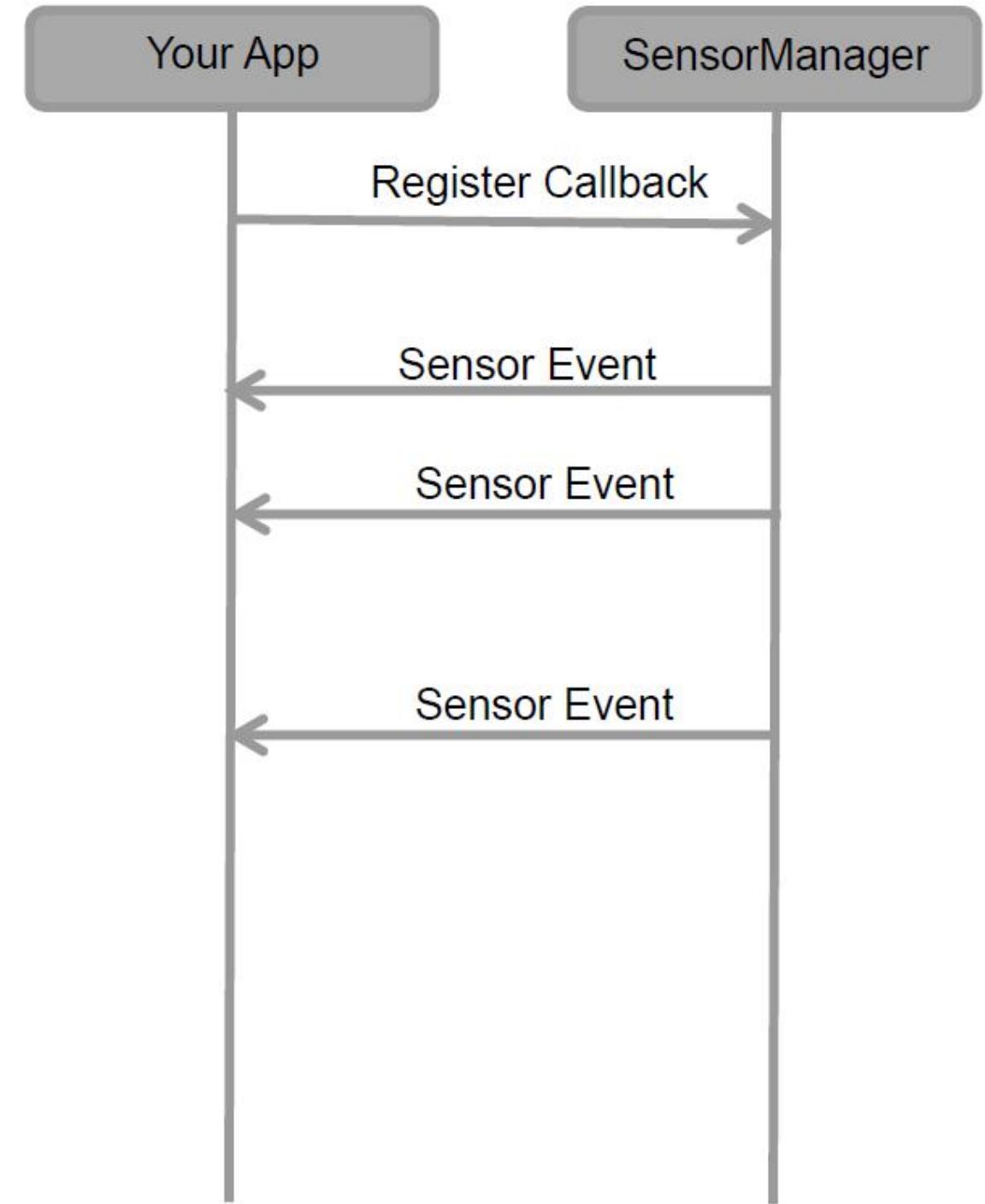
- **SensorManager**
  - Channel between your classes and sensors
- **Sensor**
  - Abstract representation of sensors on device
- **SensorEvent**
  - Represents information about a sensor event
- **SensorEventListener**
  - Register with SensorManager to listen for events from a sensor

# 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

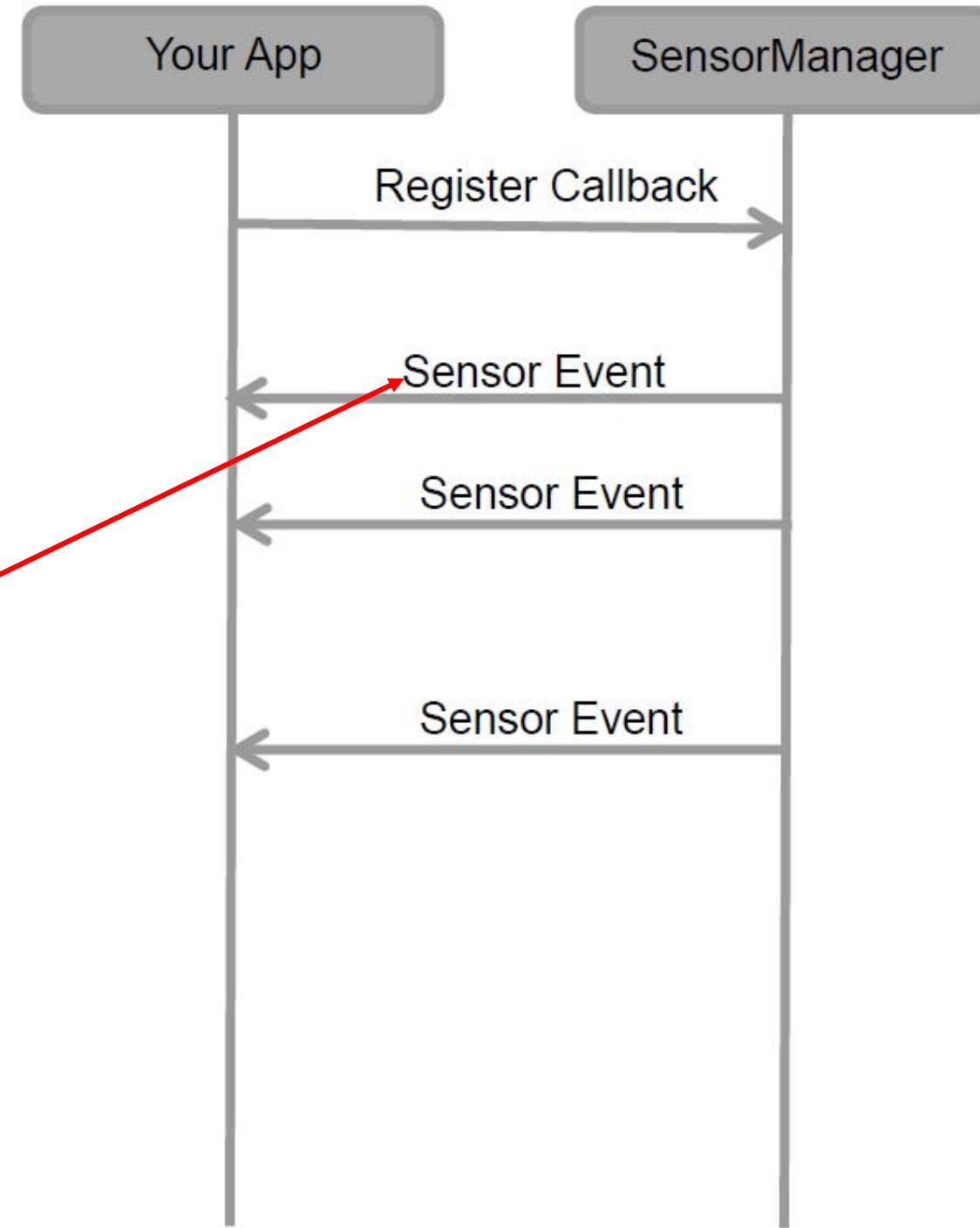
# Sensor Events and Callbacks

- 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

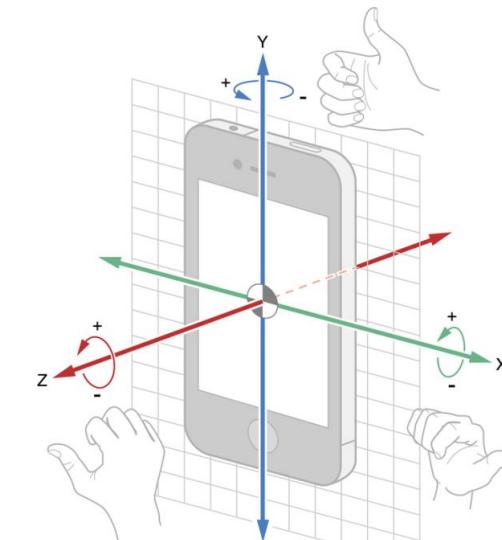
- Sensor availability varies from device to device, it can also vary between Android versions
  - Most devices have accelerometer and magnetometer
  - Some devices have barometers or thermometers
  - Device can have more than one sensor of a given type
  - Availability varies between Android versions

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

# Sensors

- TYPE\_ACCELEROMETER
  - Hardware
  - Acceleration force in  $\text{m/s}^2$
  - 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
  - $\mu\text{T}$  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()`

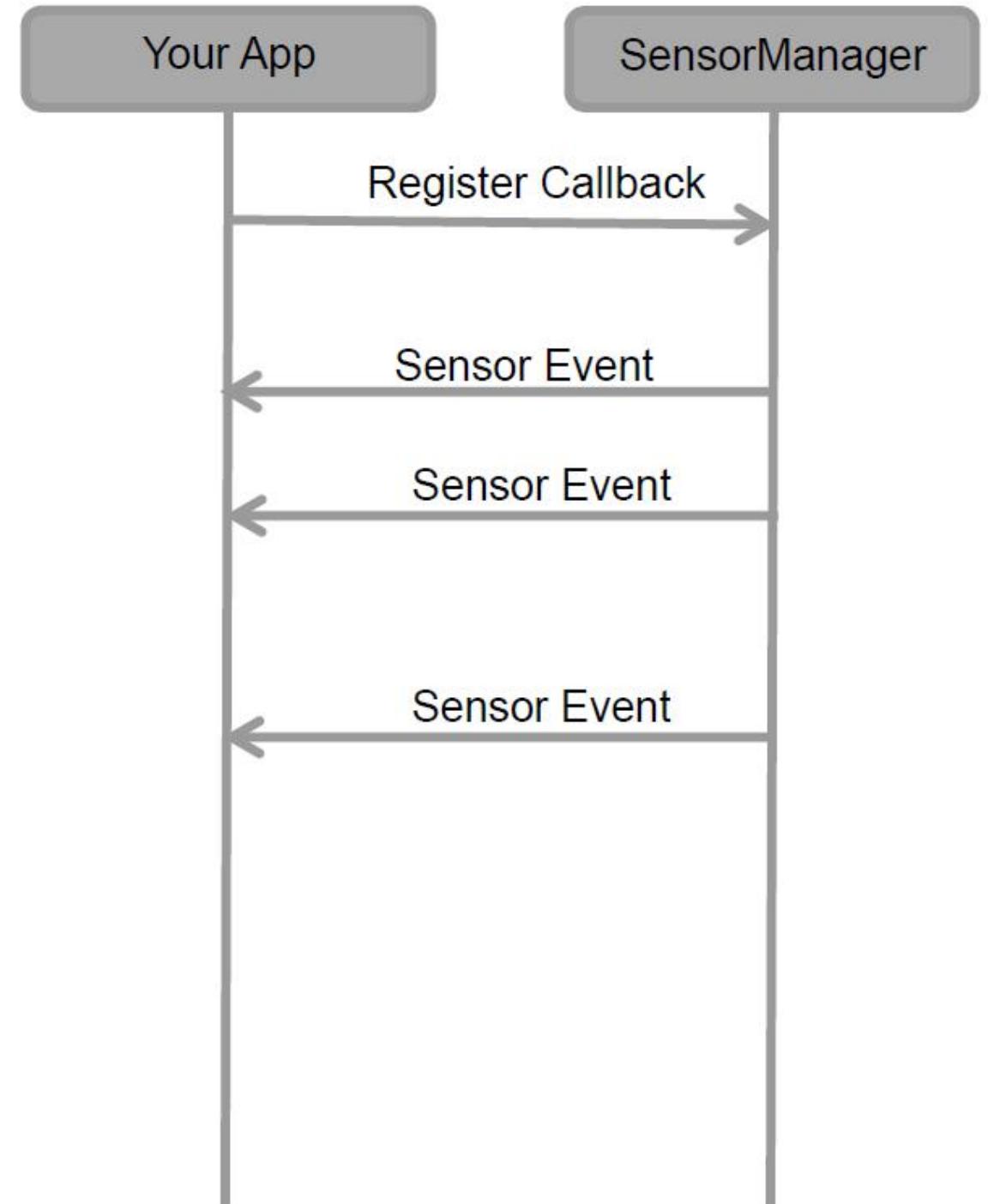
# Managing Sensor Accuracy

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).

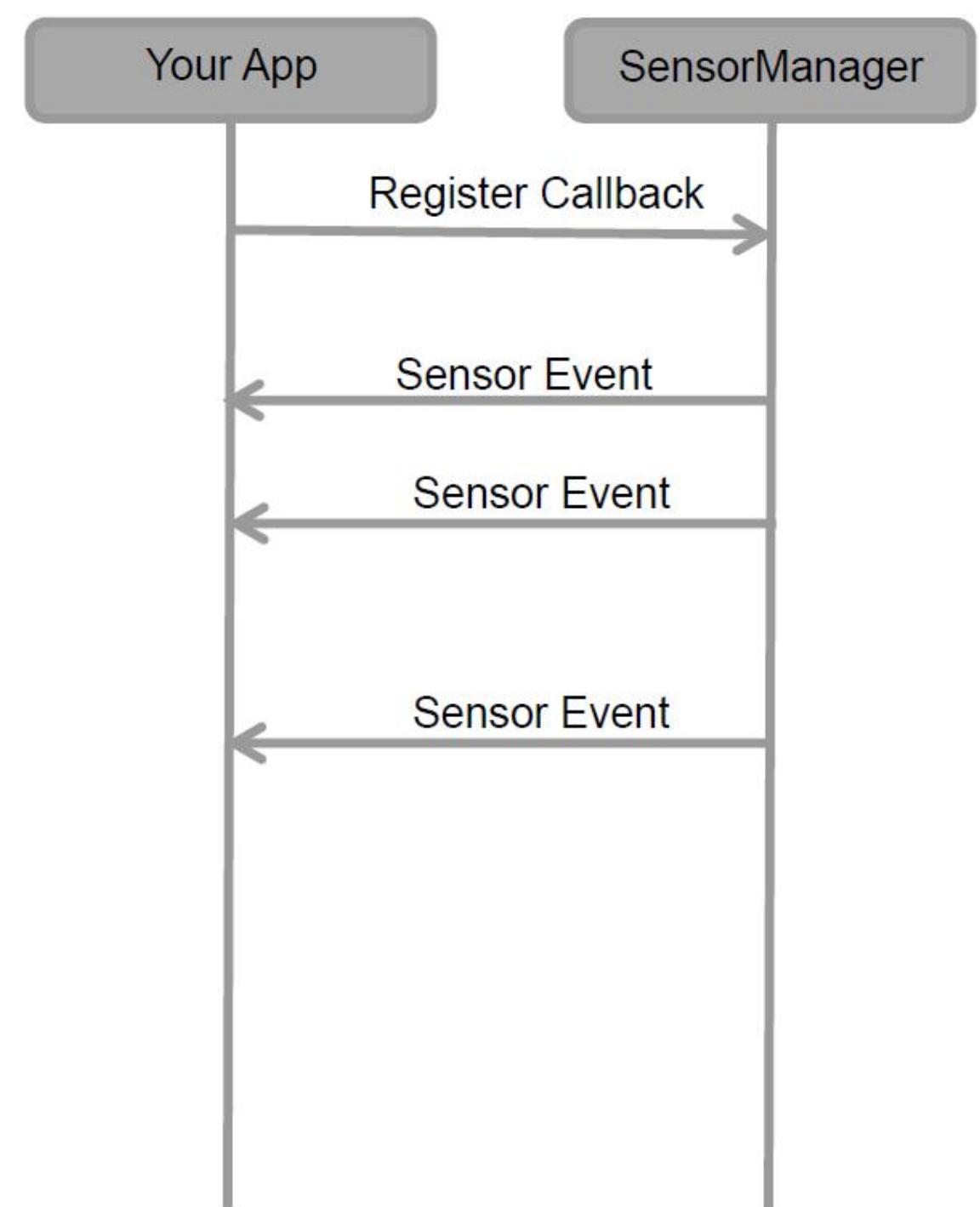
- Sensor Values Depend on Sensor Type

Sensor	Sensor event data	Description	Units of measure
TYPE_ACCELEROMETER	SensorEvent.values[0]	Acceleration force along the x axis (including gravity).	m/s <sup>2</sup>
	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 <sup>2</sup>
	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	SensorEvent.values[0]	Acceleration force along the x axis (excluding gravity).	m/s <sup>2</sup>
	SensorEvent.values[1]	Acceleration force along the y axis (excluding gravity).	
	SensorEvent.values[2]	Acceleration force along the z axis (excluding gravity).	
TYPE_ROTATION_VECTOR	SensorEvent.values[0]	Rotation vector component along the x axis ( $x * \sin(\theta/2)$ ).	Unitless
	SensorEvent.values[1]	Rotation vector component along the y axis ( $y * \sin(\theta/2)$ ).	
	SensorEvent.values[2]	Rotation vector component along the z axis ( $z * \sin(\theta/2)$ ).	
	SensorEvent.values[3]	Scalar component of the rotation vector ( $(\cos(\theta/2))$ ). <sup>1</sup>	
TYPE_SIGNIFICANT_MOTION	N/A	N/A	N/A
TYPE_STEP_COUNTER	SensorEvent.values[0]	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.  
}  
  
@Override  
protected void onResume() {  
    super.onResume();  
    mSensorManager.registerListener(this, mLIGHT, SensorManager.SENSOR_DELAY_NORMAL);  
}  
  
@Override  
protected void onPause() {  
    super.onPause();  
    mSensorManager.unregisterListener(this);  
}
```

Called by Android system to report new sensor value

Provides SensorEvent object containing new sensor data

Get new light sensor value

Register sensor when app becomes visible

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

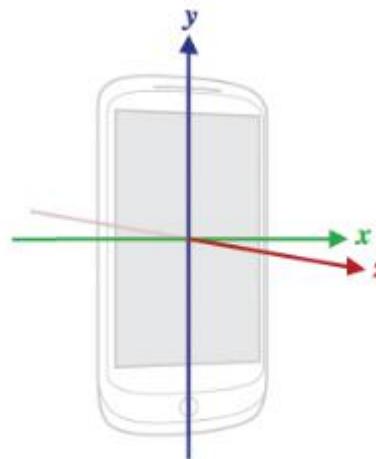
- Don't block the onSensorChanged() method
  - 50 updates a second for onSensorChange method not uncommon
  - if necessary save event and do work in another thread or asynch task

# 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



# CSE 162 Mobile Computing

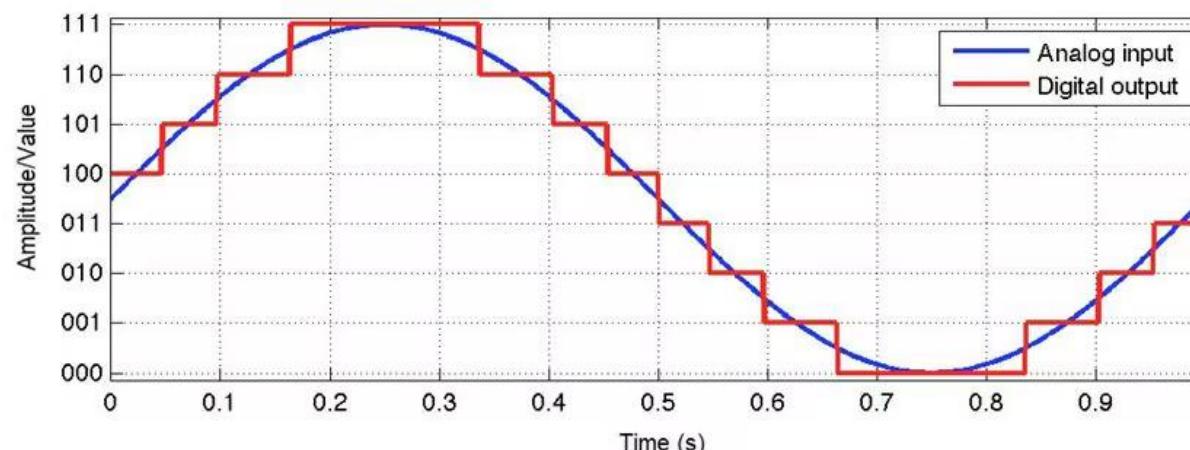
## Mobile Sensors

Hua Huang

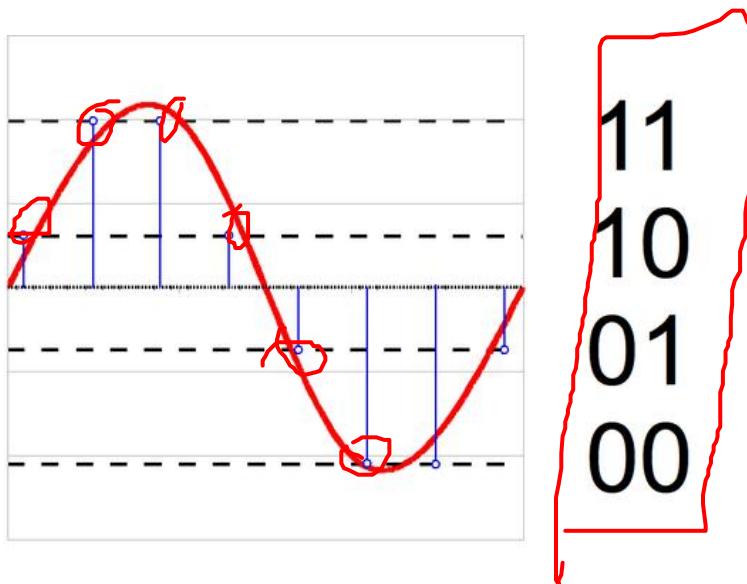
# Analog to Digital Converter

# How does an ADC work?

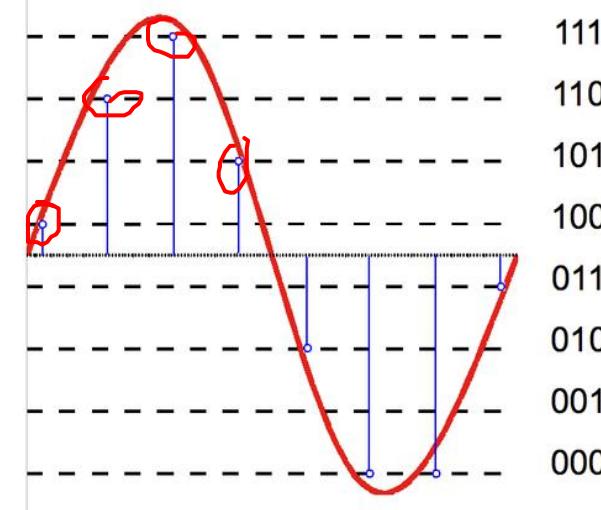
- Analog signals are signals that have a continuous sequence with continuous values
- Digital signals are represented by a sequence of discrete values broken down into sequences
- Computers can't read values unless it's digital data.
  - They can only see “levels” of the voltage



- An analog-to-digital converter (ADC) can be modeled as two processes: sampling and quantization.
  - Sampling converts a time-varying voltage signal into a discrete-time signal, a sequence of real numbers.
  - Quantization replaces each real number with an approximation from a finite set of discrete values.

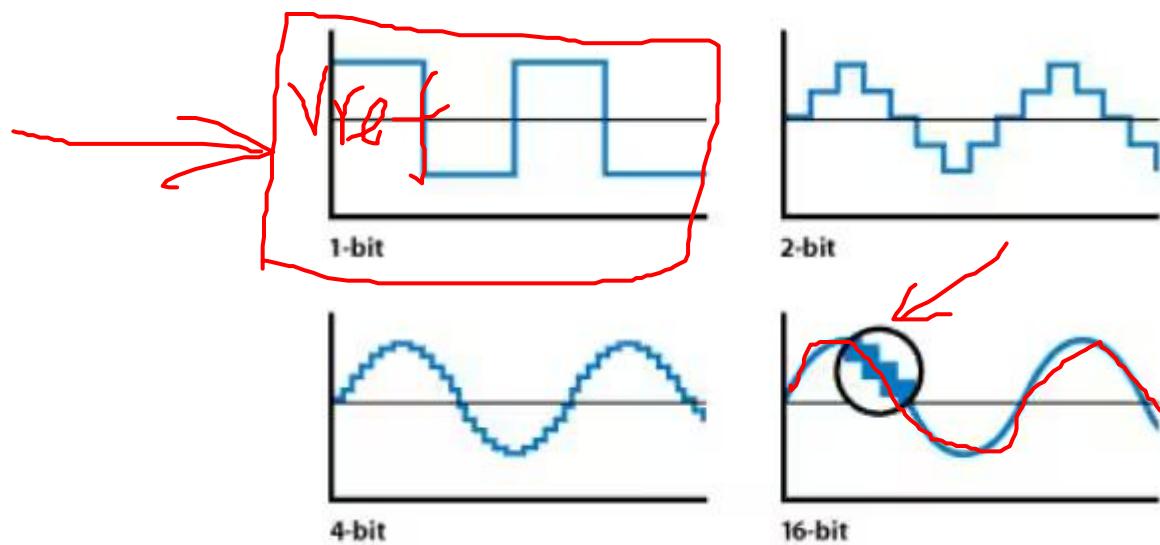


11  
10  
01  
00



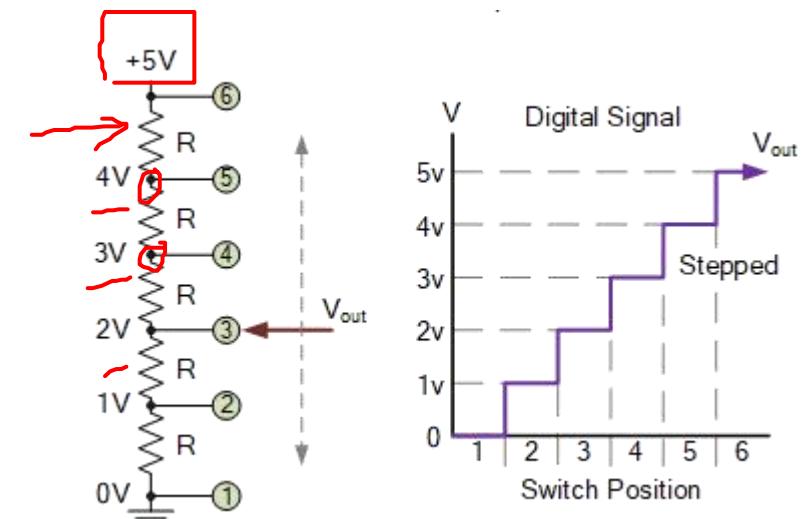
# Resolution of ADC

- The resolution of the ADC can be determined by its bit length.
  - 1-bit only has two “levels”.
  - As the bit length increases, the levels increase making the signal more closely represent the original analog signal.



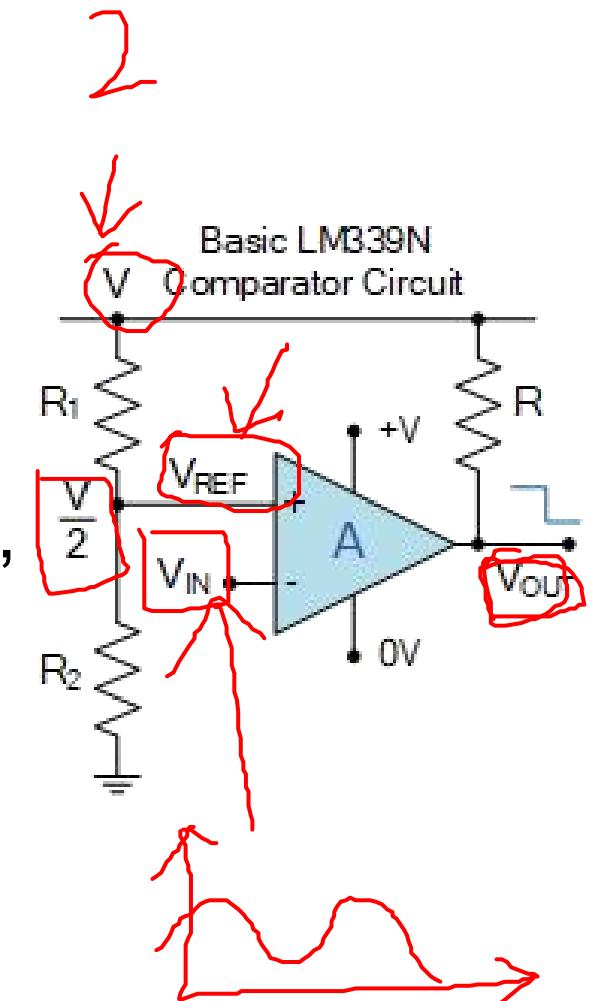
# ADC Building blocks

- How to generate the Vref?
- Series resistor chain, forms a basic potential divider network.
  - When the switch is rotated from one position (or node) to the next the output voltage, Vref changes quickly in discrete and distinctive voltage steps representing multiples of 1.0 volts on each switching action as shown.

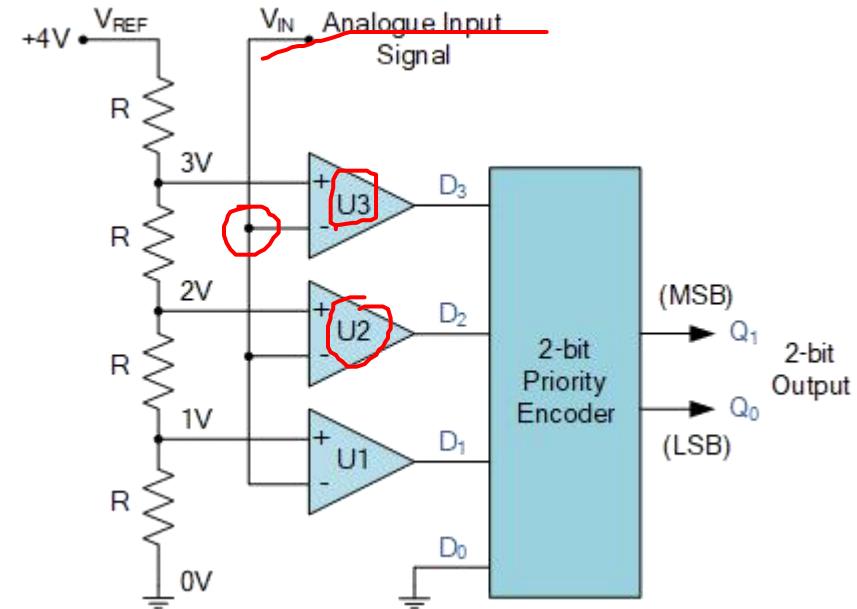


# ADC Building blocks

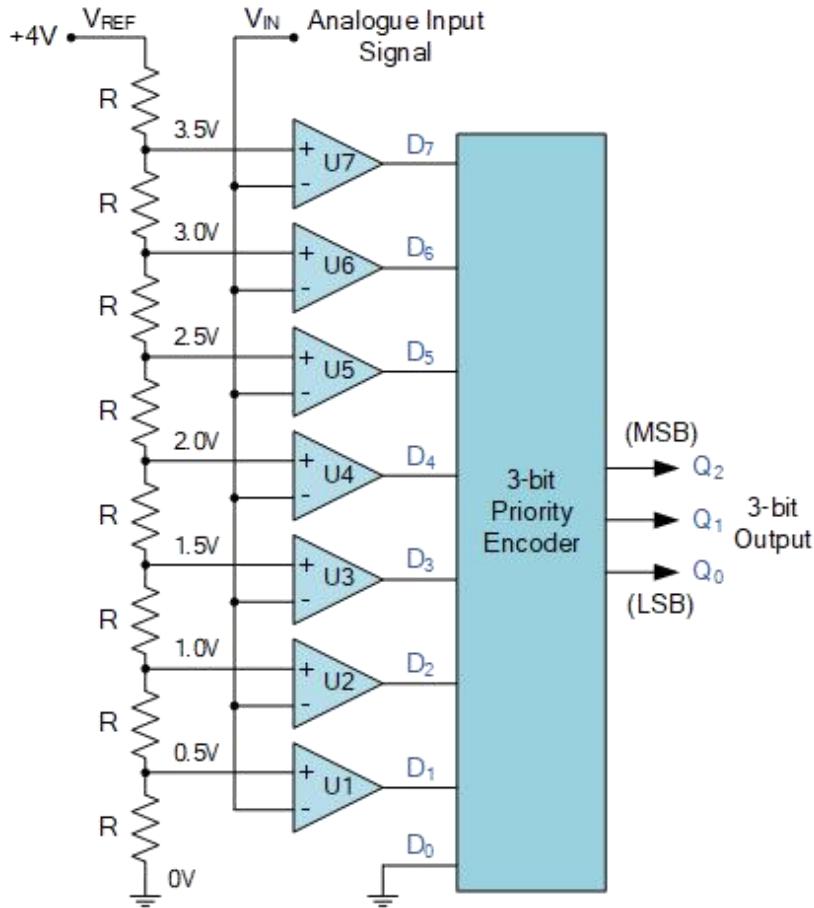
- An analogue comparator has two analogue inputs, one positive and one negative
  - Can compare the magnitudes of two different voltage levels.
- A voltage input, ( $V_{IN}$ ) signal is applied to one input of the comparator, while a reference voltage, ( $V_{REF}$ ) to the other.
- A comparison of the two voltage levels at the comparator's input is made to determine the comparators digital logic output state, either a “1” or a “0”.



- By adding more resistors to the voltage divider network we can effectively “divide” the supply voltage by an amount determined by the resistances of the resistors.
- In general,  $2^n - 1$  comparators would be required for conversion of an “n”-bit binary output, where “n” is typically in the range from 8 to 16.
- If we now create a 2-bit ADC, then we will need  $2^2 - 1$  which is “3” comparators as we need four different voltage levels corresponding to the 4 digital values required for a 4-to-2 bit encoder circuit as shown.



Analogue Input Voltage ( $V_{IN}$ )	Comparator Outputs				Digital Outputs	
	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	Q <sub>1</sub>	Q <sub>0</sub>
0 to 1 V	0	0	0	0	0	0
1 to 2 V	0	0	1	X	0	1
2 to 3 V	0	1	X	X	1	0
3 to 4 V	1	X	X	X	1	1



Analogue Input Voltage ( $V_{IN}$ )	Comparator Outputs								Digital Outputs		
	$D_7$	$D_6$	$D_5$	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$	$Q_2$	$Q_1$	$Q_0$
0 to 0.5 V	0	0	0	0	0	0	0	0	0	0	0
0.5 to 1.0 V	0	0	0	0	0	0	1	X	0	0	1
1.0 to 1.5 V	0	0	0	0	0	1	X	X	0	1	0
1.5 to 2.0 V	0	0	0	0	1	X	X	X	0	1	1
2.0 to 2.5 V	0	0	0	1	X	X	X	X	1	0	0
2.5 to 3.0 V	0	0	1	X	X	X	X	X	1	0	1
3.0 to 3.5 V	0	1	X	X	X	X	X	X	1	1	0
3.5 to 4.0 V	1	X	X	X	X	X	X	X	1	1	1

Where again "X" is a "don't care", that is either a logic "0" or a logic "1" input condition.

# Micro-electromechanical systems (MEMS)

# MEMS

- Accelerometer+Gyroscope= 6 DoF Inertia Motion Units(IMU)
- Very small and low-cost
  - less than 6 mm × 6 mm in footprint can weigh less than a gram
  - MEMS vendors have been shipping high volumes for mobiles, tablets, and automotive applications since 1990.
- Robust to environmental changes
  - The shock specifications of today's generation of devices are stated to 10,000 g, but in reality can tolerate much higher
  - Normal humans can withstand no more than 9 g's



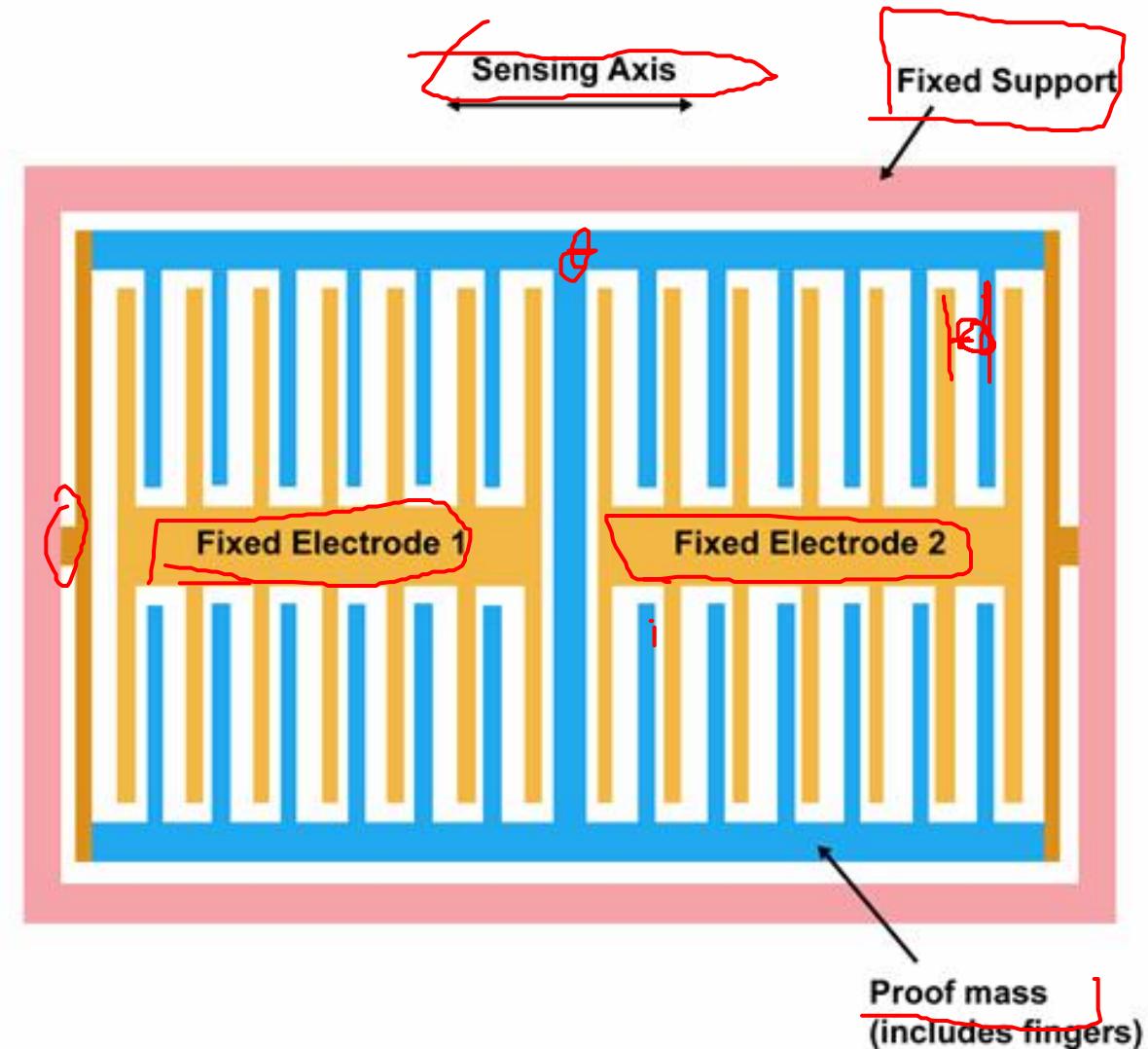
# MEMS

- Where do you see these?
  - Wii Nunchuks
  - Orientation sensing in smartphones
  - Image Stabilization in cameras
  - Collision detection in cars
  - Pedometers
  - Monitoring equipment for failure (vibrations in ball bearings, etc)



# MEMS: Accelerometer

- Accelerometers:
  - Measures change in velocity in x y z axes
- How does it work?
  - The “proof mass” shown above is allowed to move in a plane.
  - The attached fingers form a capacitor with the two plates around it.
  - The rate of change of the capacitance is measured and translated into an acceleration



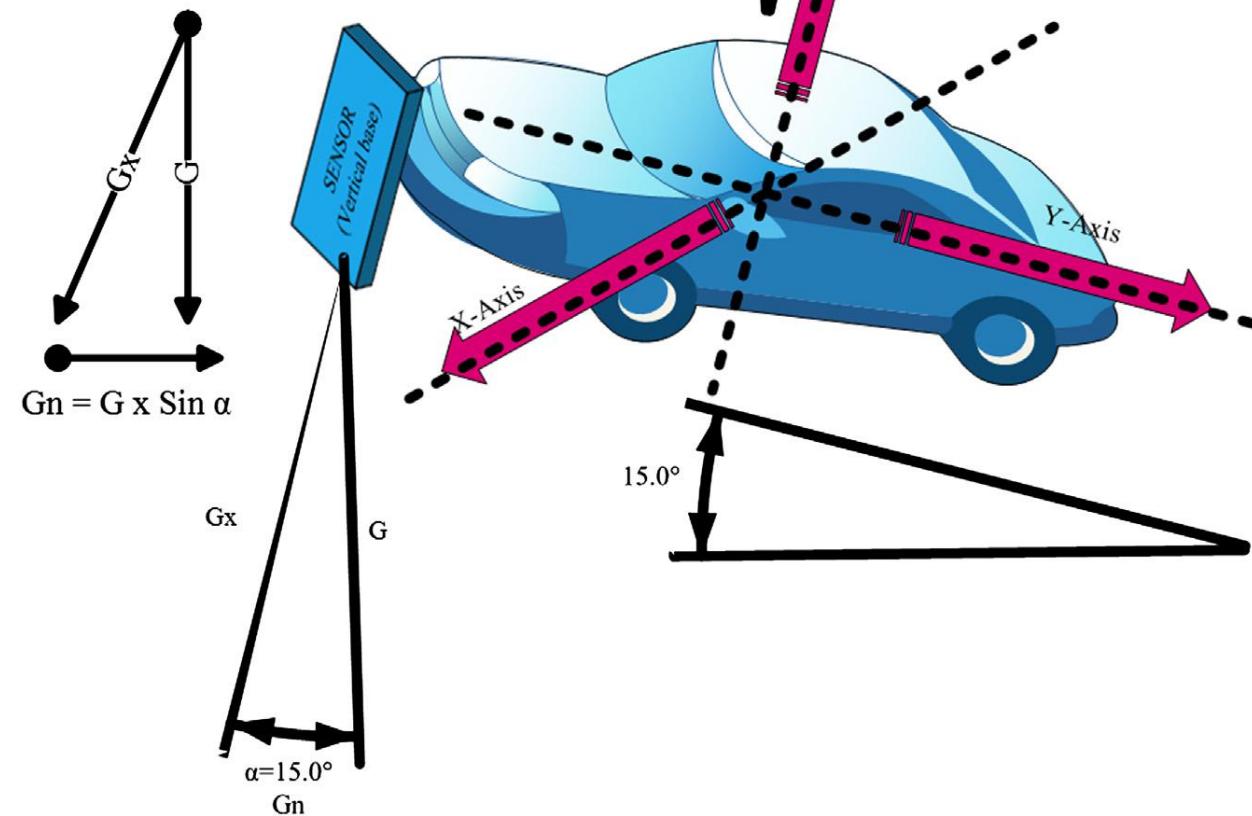
- With the device flat on a surface what, roughly, will be the magnitude of the largest acceleration?
  - 0 m/s<sup>2</sup>
  - 1 m/s<sup>2</sup>
  - 5 m/s<sup>2</sup>
  - 10 m/s<sup>2</sup>
  - 32 m/s<sup>2</sup>

# Influence of Gravity

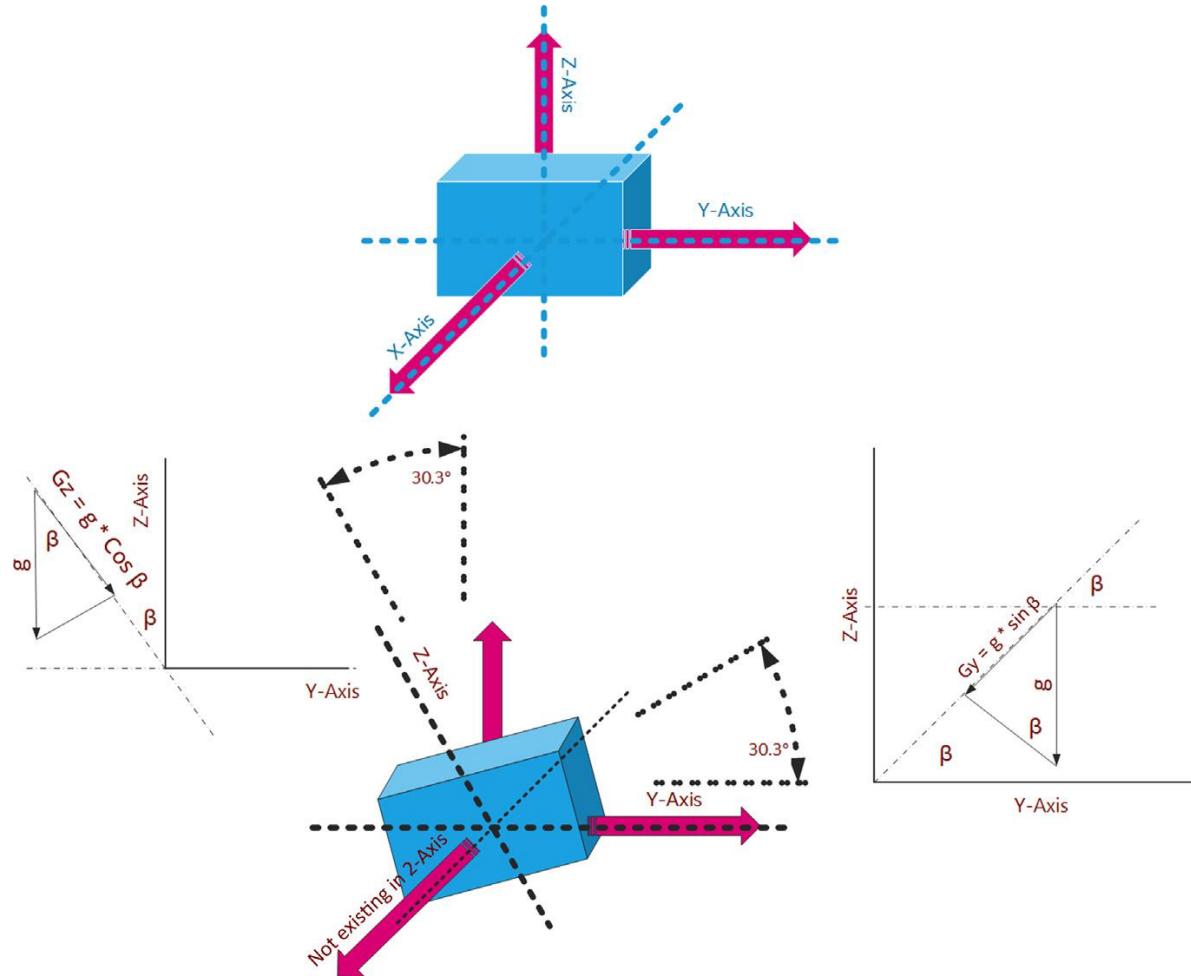
- An accelerometer at rest will not have a zero measurement. Instead, it measures  $9.8m/s^2$
- A free-falling sensor will measure 0

# Influence of Tilt Angle

- $a_z = g * \cos(\alpha)$
- $a_y = g * \sin(\alpha)$

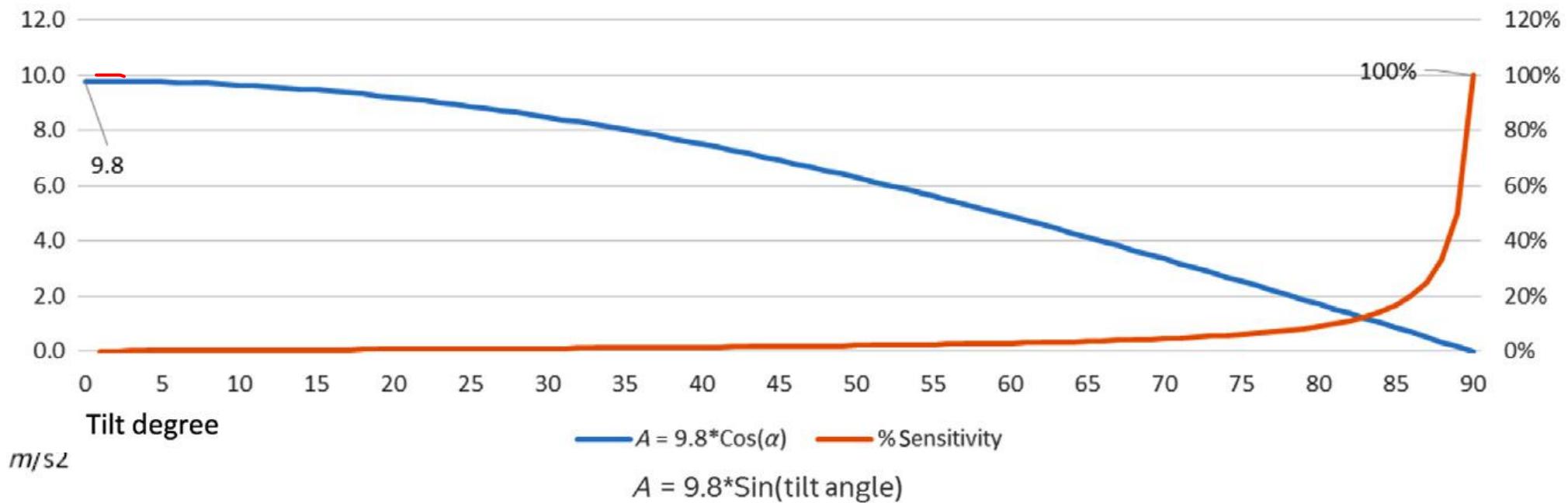


# Sensitivity to Tilt Angle



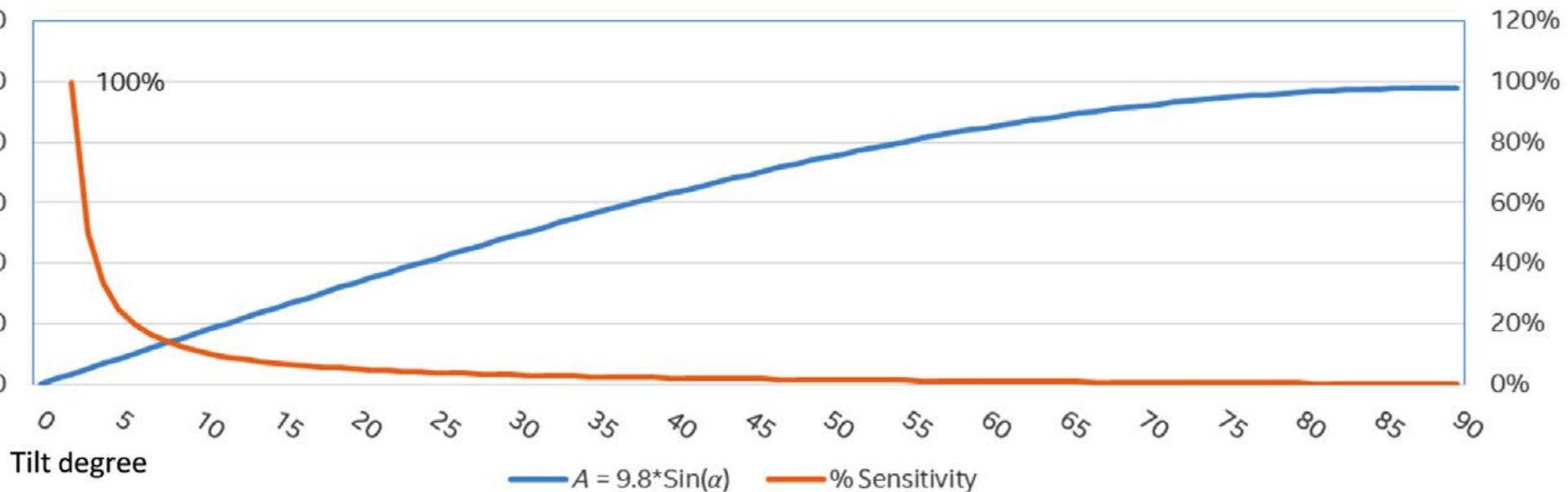
*m/s<sup>2</sup>*

$$A = 9.8 \cdot \cos(\text{tilt angle})$$



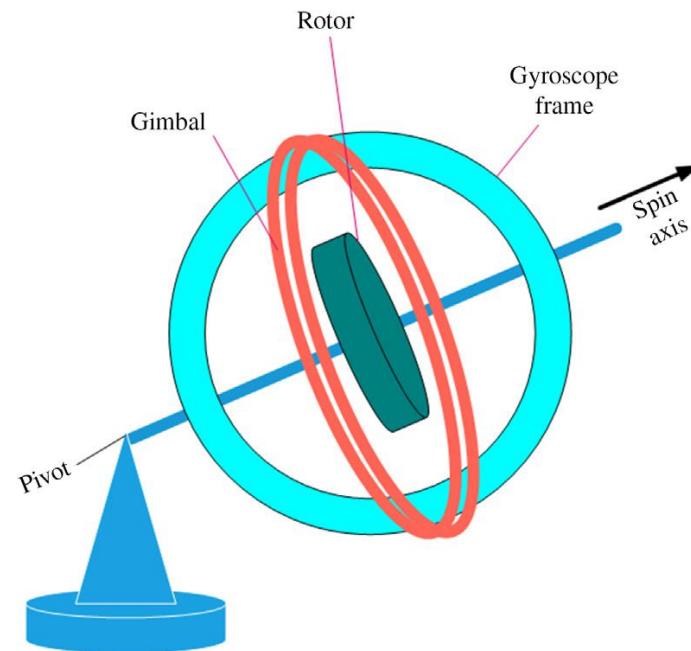
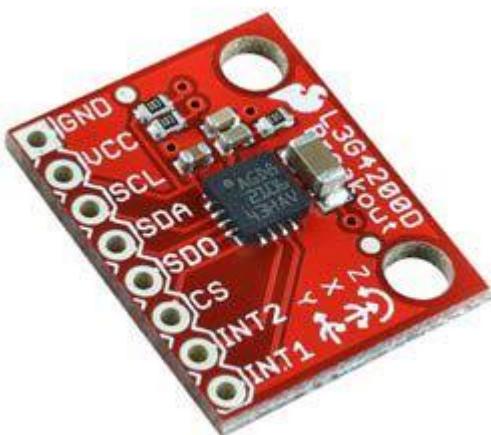
*m/s<sup>2</sup>*

$$A = 9.8 \cdot \sin(\text{tilt angle})$$

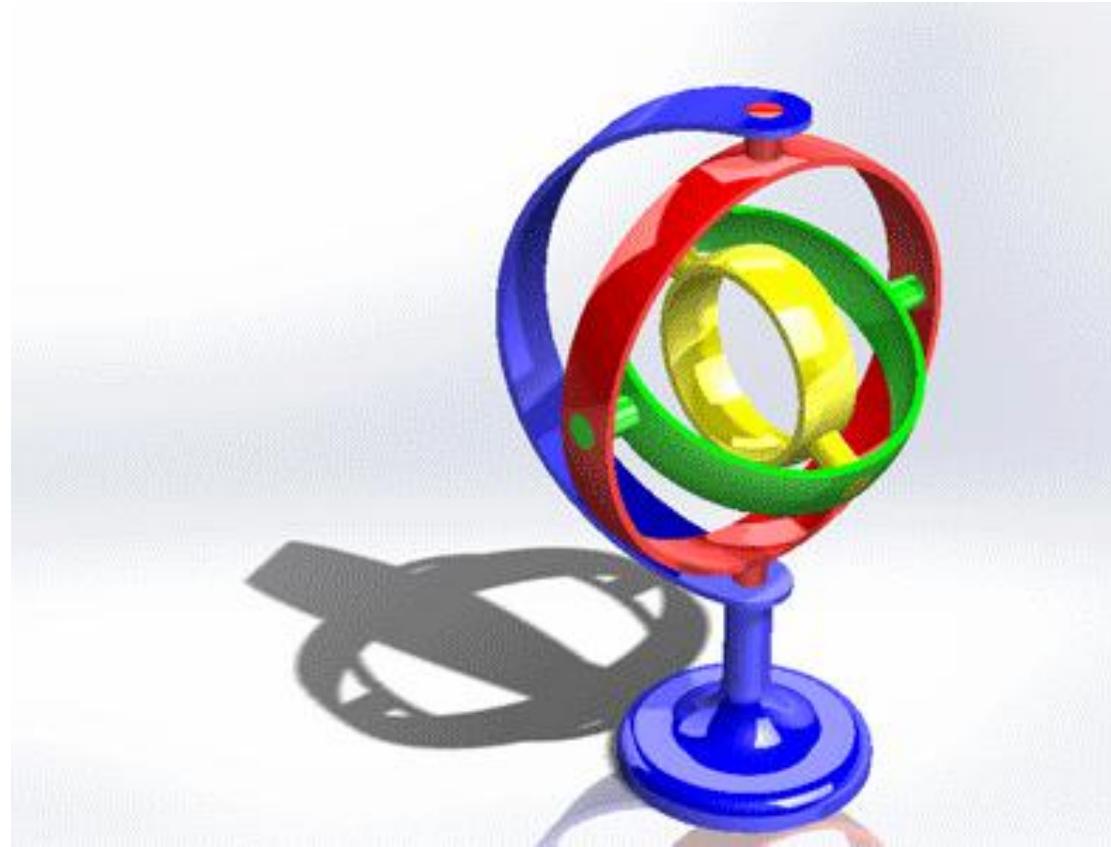


# Gyroscope

- Measures rotation speed

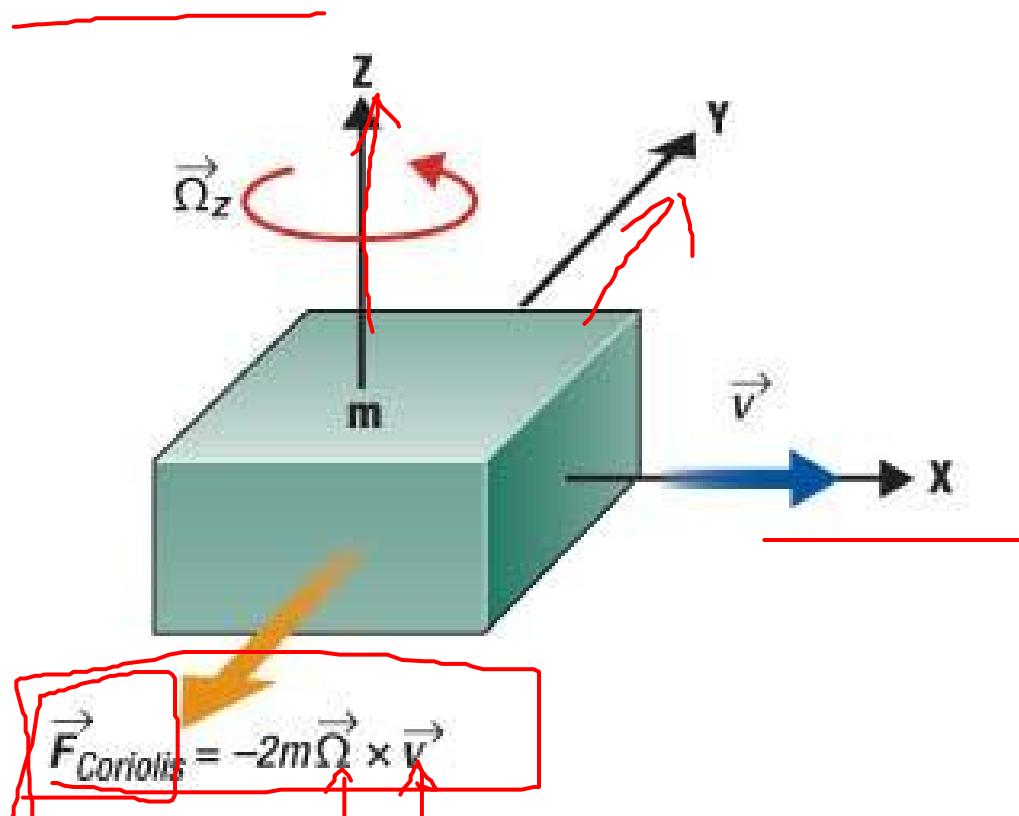


# Mechanical Gyroscope



# How MEMS Gyroscope works?

- The Coriolis force
  - If an object is moving along one axis, and it is rotated above another, it will feel a Coriolis force in the third axial direction

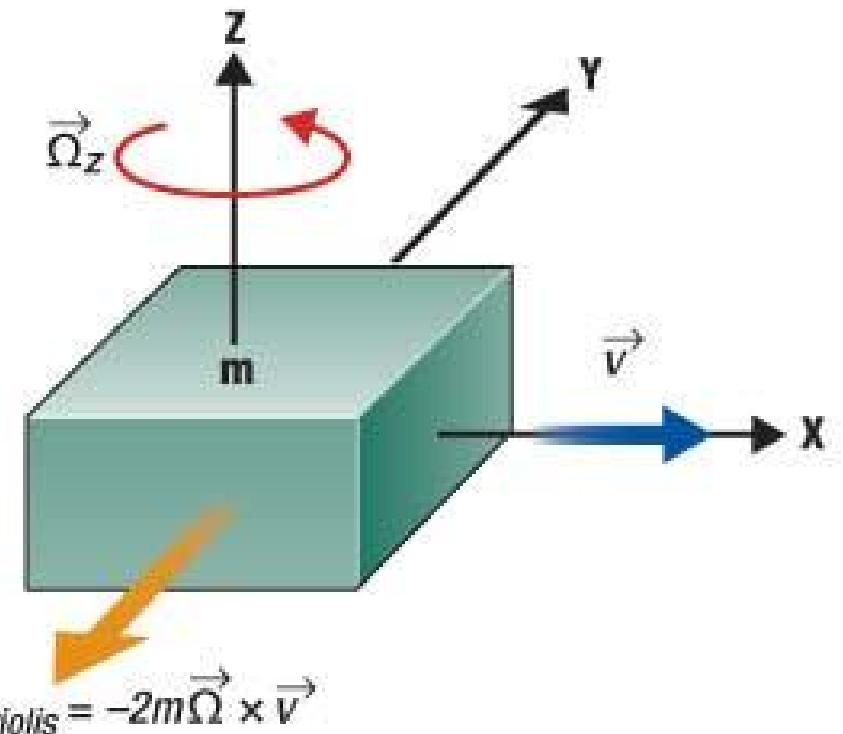


# The Direction of Coriolis Force

- To memorize using the right hand rules.

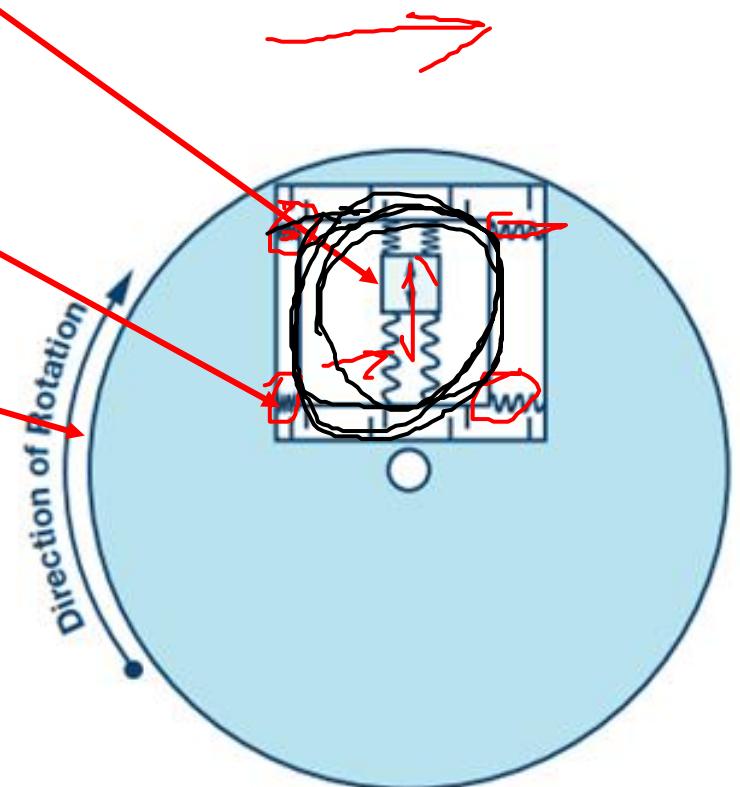
Steps:

1. Find the rotation axis using the right hand rule (z).
2. Cross product the movement direction(x) and rotation axis (z), and get the force direction. Using the right hand rule



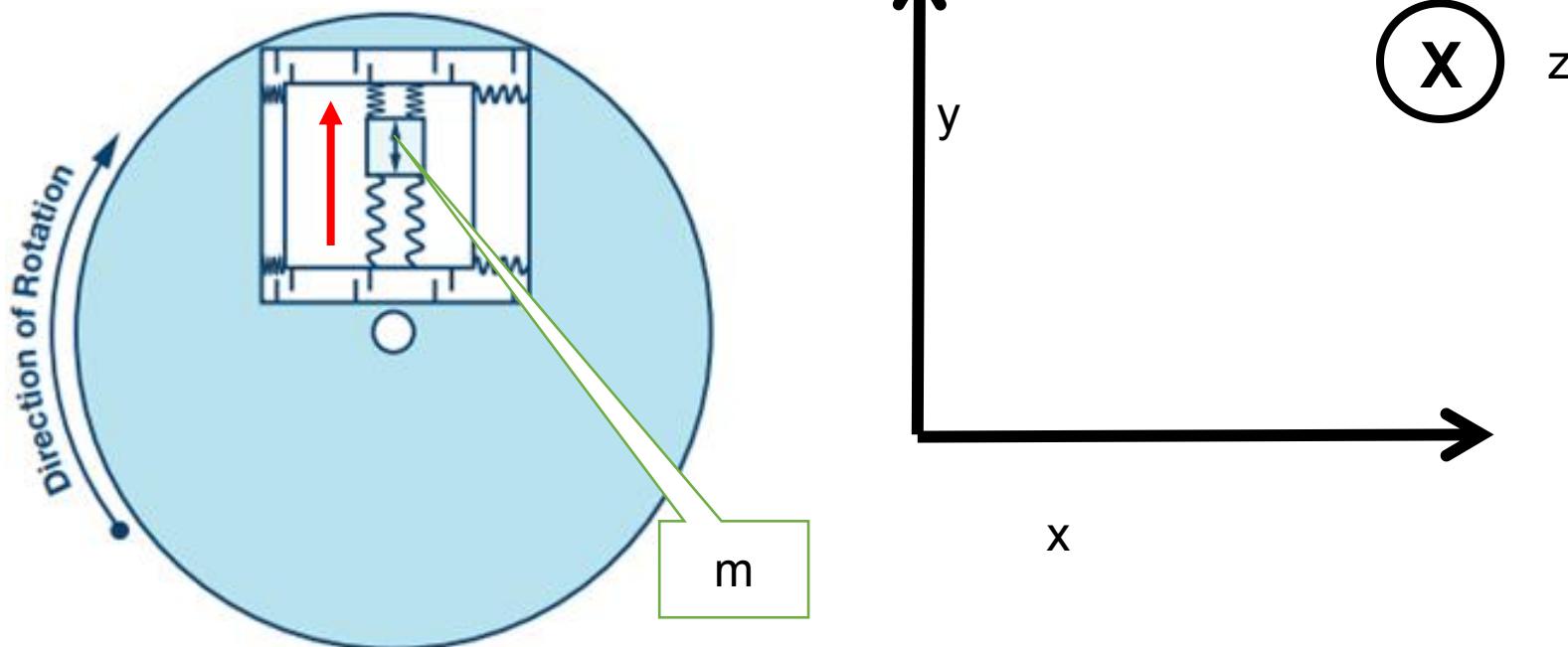
# How MEMS Gyroscope works?

- A gyroscope will have a mass oscillating back and forth along the first axis. Easy to implement using a minuscule mass.
- This oscillating mass is then placed on a second spring controlled pads. The capacitance is proportional to their positions
- These structures are placed on a disc that is free to rotate.
- When a rotation is detected around the second direction, the capacitance changes



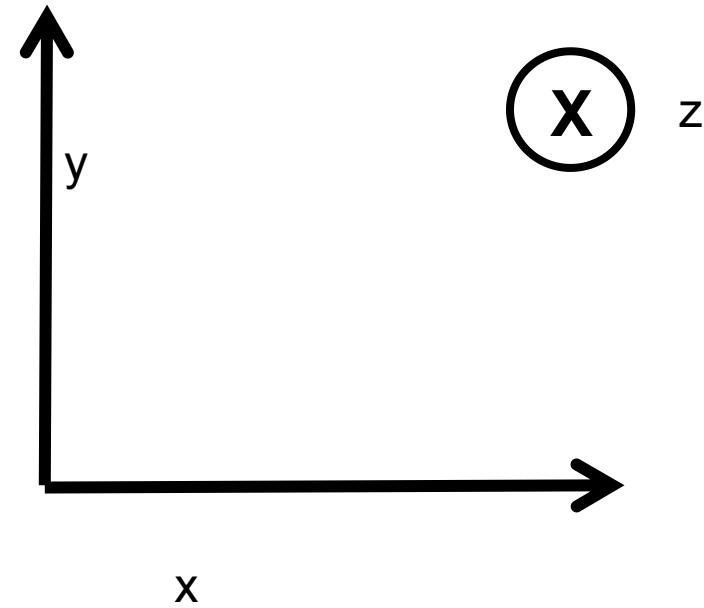
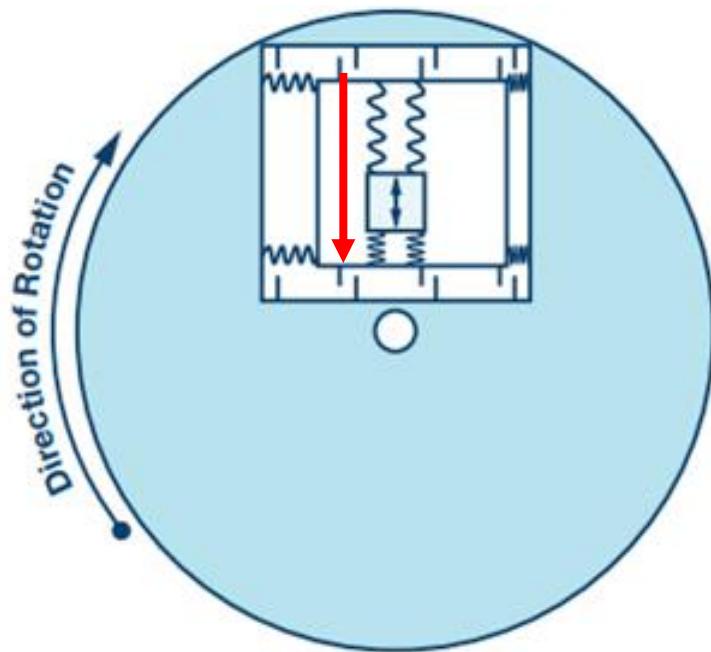
# Inside an MEMS gyroscope

- A mass,  $m$ , vibrating along y axis
- When the mass moves  $+y$  direction, rotating along z clockwise
  - What is Coriolis force direction?
  - $-x$



# Exercise

- Explain the direction of Coriolis force when object moves along  $-y$  direction



# CSE 162 Mobile Computing

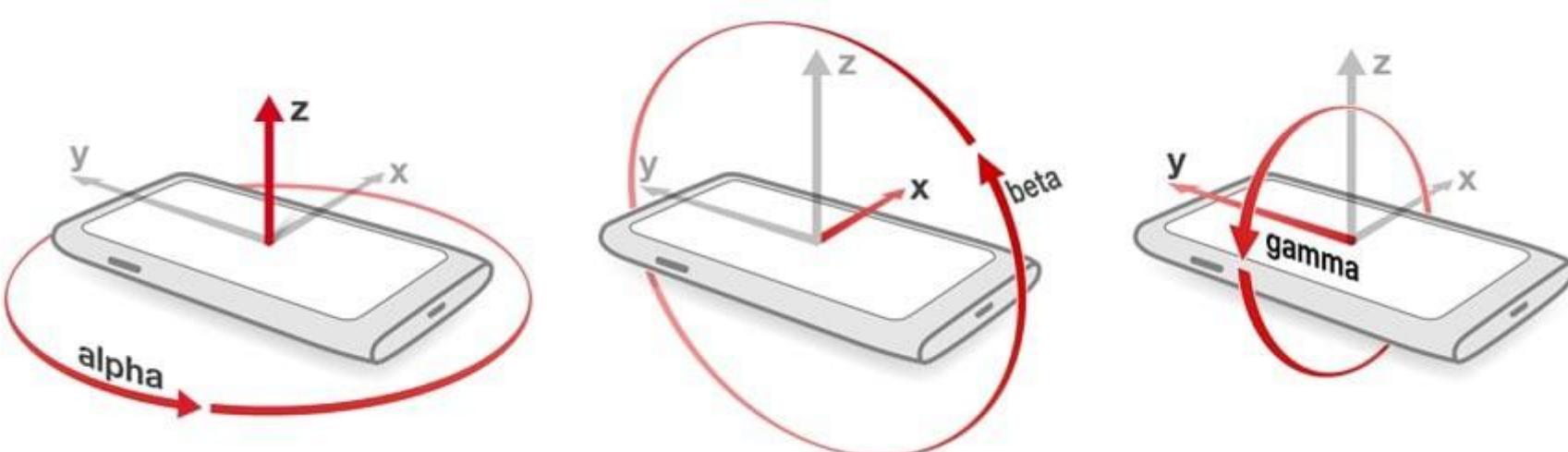
## Orientation and Step Counting

Hua Huang

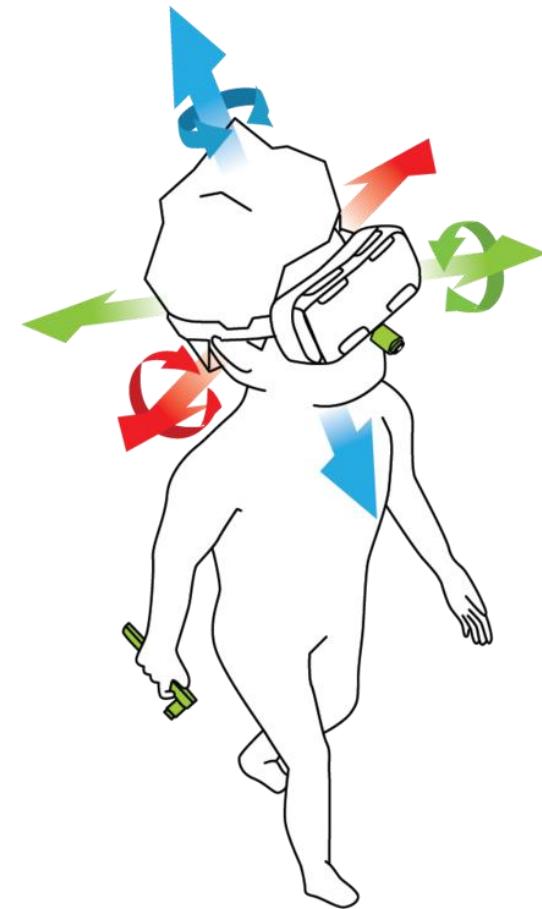
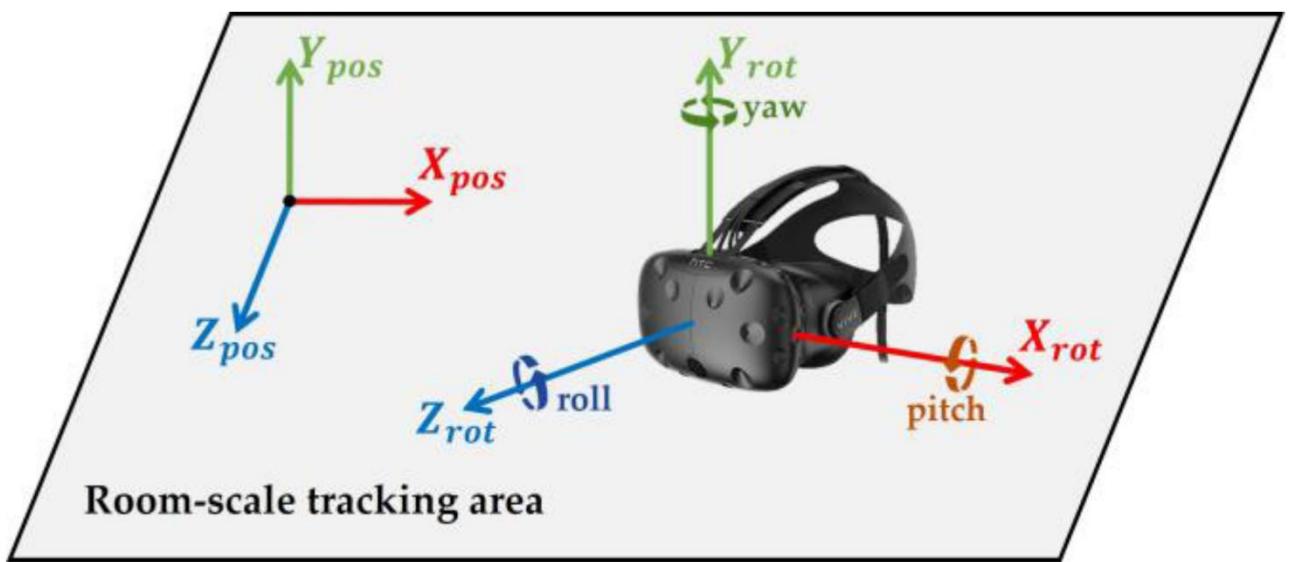
# Virtual Sensor: 3D Orientation

Question:

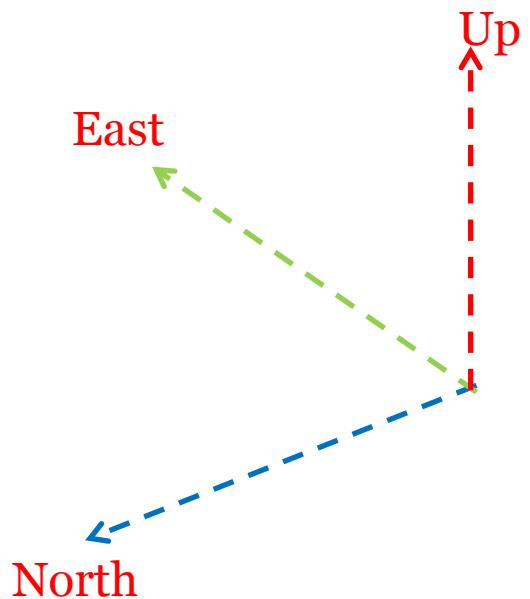
How do we know the orientation of the phone?



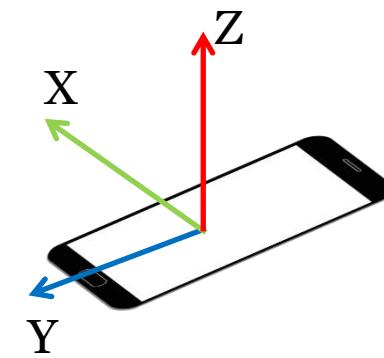
- Orientations of the VR goggle?



# Coordinate frames

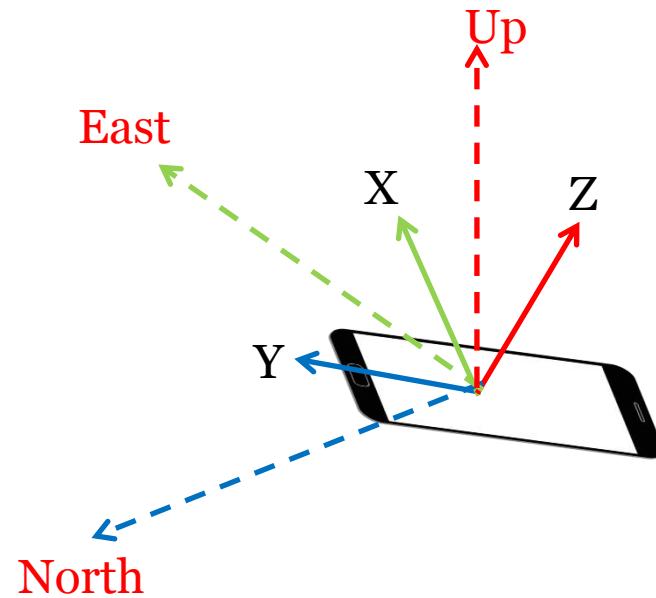


Global Frame



Local Frame

# Consider a phone in a random orientation

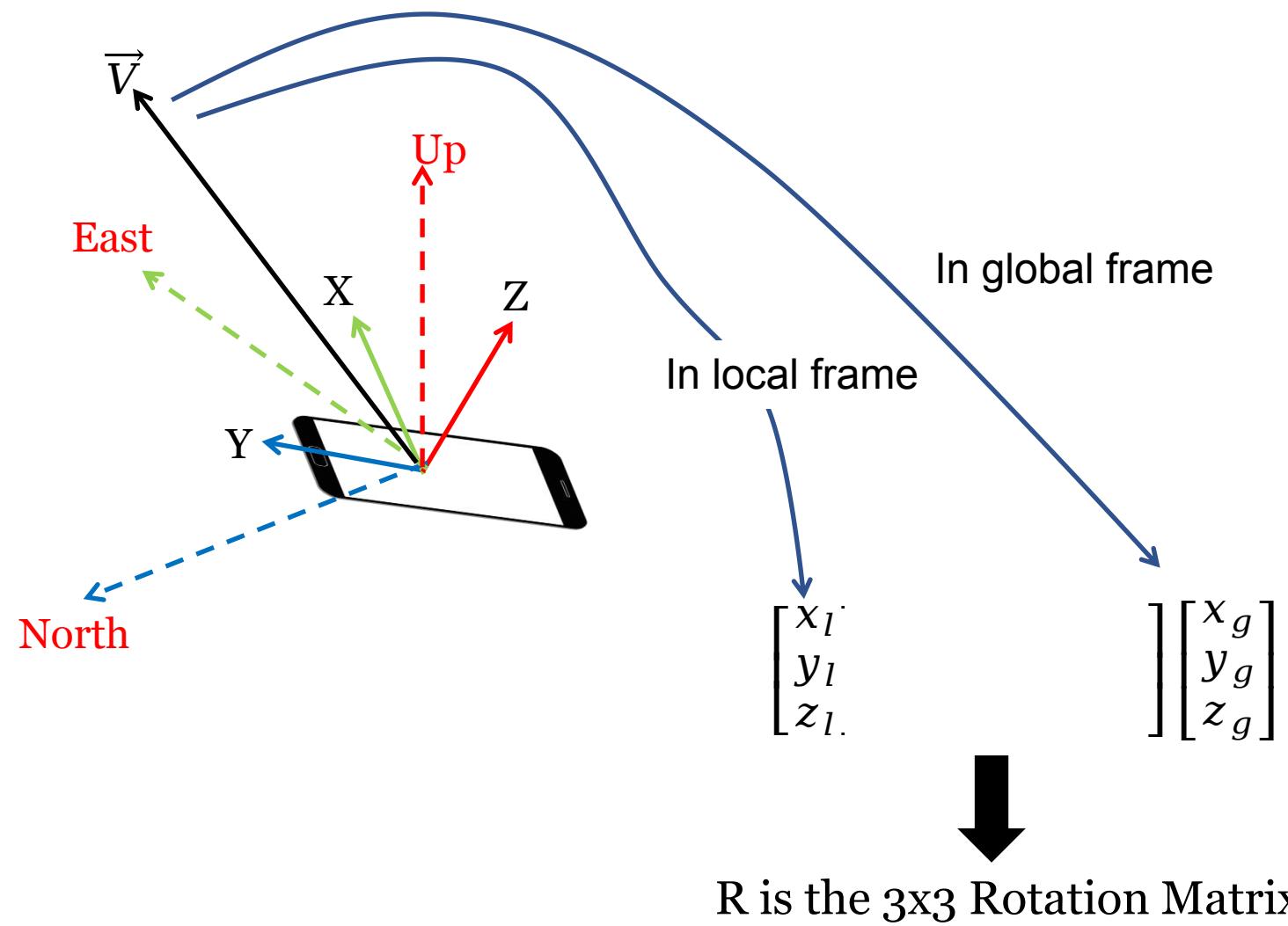


3D Orientation captures the **misalignment** between **global** and **local** frames

# Gravity Sensor

- A virtual sensor
- Calculated using accelerometer
- Always points to the earth

# Rotation Matrix



3x3 Rotation matrix captures the full 3D orientation

# How can we estimate rotation matrix?

Key idea    use globally known reference vectors  
which can also be measured in the local frame of reference

- Gravity
- Magnetic North

# Gravity equation

Gravity globally known, measurable in local frame with gravity sensor

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$$

$$\begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = \begin{bmatrix} 0 \\ M \\ 0 \end{bmatrix}$$

Magnetic north, globally known, measurable in local frame with magnetometer

6 equations and 9 unknowns (3x3 rotation matrix) can we solve ?

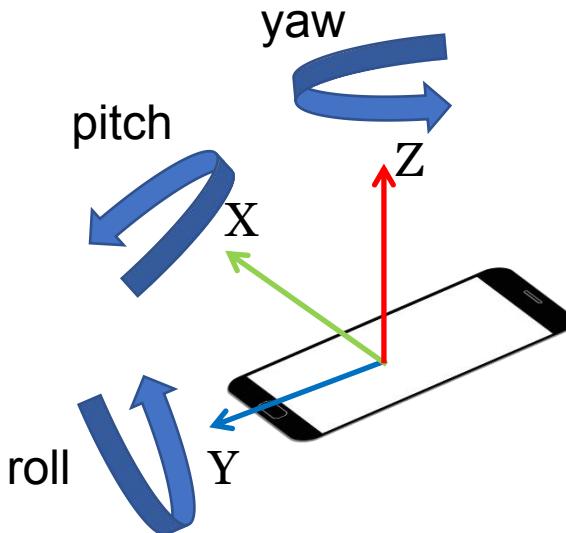
Yes, these 9 unknowns are all not independent (rotation matrix satisfies special properties)

- It does not change length of a vector
- Columns are orthogonal unit vectors

The above 6 equations are sufficient to solve the rotation matrix

Gravity Sensor and Magnetometer can be used to determine the rotation matrix (3D orientation)

# Decomposing the rotation matrix



$$\boxed{\begin{bmatrix} \text{3x3 Rotation} \\ \text{Matrix } R \end{bmatrix}} = \begin{bmatrix} \cos(pitch) & 0 & -\sin(pitch) \\ 0 & 1 & 0 \\ \sin(pitch) & 0 & \cos(pitch) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(roll) & \sin(roll) \\ 0 & -\sin(roll) & \cos(roll) \end{bmatrix} \begin{bmatrix} \cos(yaw) & -\sin(yaw) & 0 \\ \sin(yaw) & \cos(yaw) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Orientation can be represented as 3D yaw, pitch, roll

Estimating yaw, pitch, roll will determine the orientation

# Gravity equation

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \text{Rotation Matrix } R \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}$$

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} \cos(pitch) & 0 & -\sin(pitch) \\ 0 & 1 & 0 \\ \sin(pitch) & 0 & \cos(pitch) \end{bmatrix} \begin{bmatrix} 0 \\ \cos(roll) & \sin(roll) \\ -\sin(roll) & \cos(roll) \end{bmatrix} \begin{bmatrix} \cos(yaw) & -\sin(yaw) \\ \sin(yaw) & \cos(yaw) \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$$

# Gravity equation

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \text{Rotation Matrix } R \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}$$

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} \cos(pitch) & 0 & -\sin(pitch) \\ 0 & 1 & 0 \\ \sin(pitch) & 0 & \cos(pitch) \end{bmatrix} \begin{bmatrix} 0 \\ \cos(roll) \\ \sin(roll) \end{bmatrix} \begin{bmatrix} 0 \\ \sin(roll) \\ \cos(roll) \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$$

**Gravity output does not depend on yaw!**

**Hence, yaw cannot be estimated using gravity**

# Accelerometer equation

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \boxed{\text{Rotation Matrix } R} \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$$

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} -\sin(pitch) \cdot \cos(roll) \\ -\sin(roll) \cdot g \\ -\cos(pitch) \cdot \cos(roll) \end{bmatrix} g$$

The above equations estimate pitch and roll

# Magnetometer equation

$$\begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = \text{Rotation Matrix } R \begin{bmatrix} 0 \\ M \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = \begin{bmatrix} \cos(pitch) & 0 & -\sin(pitch) \\ 0 & 1 & 0 \\ \sin(pitch) & 0 & \cos(pitch) \end{bmatrix} \begin{bmatrix} 0 \\ \cos(roll) & \sin(roll) \\ 0 & -\sin(roll) & \cos(roll) \end{bmatrix} \begin{bmatrix} \cos(yaw) & -\sin(yaw) \\ \sin(yaw) & \cos(yaw) \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ M \\ 0 \end{bmatrix}$$

# Magnetometer equation

$$\begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = \text{Rotation Matrix } R \begin{bmatrix} 0 \\ M \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = \begin{bmatrix} \cos(\text{pitch}) & 0 & -\sin(\text{pitch}) \\ 0 & 1 & 0 \\ \sin(\text{pitch}) & 0 & \cos(\text{pitch}) \end{bmatrix} \begin{bmatrix} 0 \\ \cos(\text{roll}) & \sin(\text{roll}) \\ -\sin(\text{roll}) & \cos(\text{roll}) \end{bmatrix} \begin{bmatrix} \cos(\text{yaw}) & -\sin(\text{yaw}) \\ \sin(\text{yaw}) & \cos(\text{yaw}) \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ M \\ 0 \end{bmatrix}$$

Pitch, roll known from accelerometer

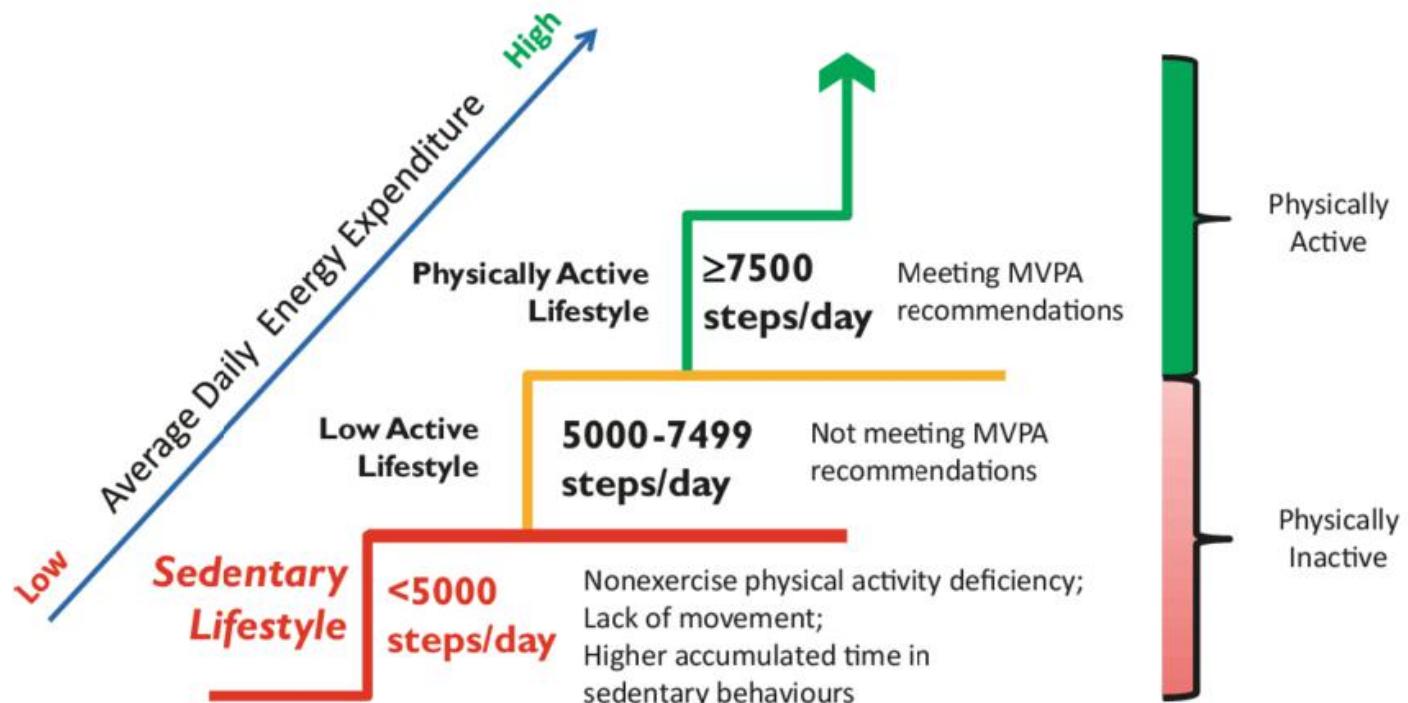
Unknown yaw can be determined from above equations

**yaw, pitch, roll together determine the rotation matrix (3D orientation) of a system**

# Virtual Sensor: Step Counting

# Sedentary Lifestyle

- Sedentary lifestyle
    - Increases risk of diabetes, heart disease, dying earlier, etc
    - Kills more than smoking!!
  - Categorization of sedentary lifestyle based on step count:
    - “A step-defined sedentary lifestyle index: < 5000 steps/day” (2013)



# Step Count Mania

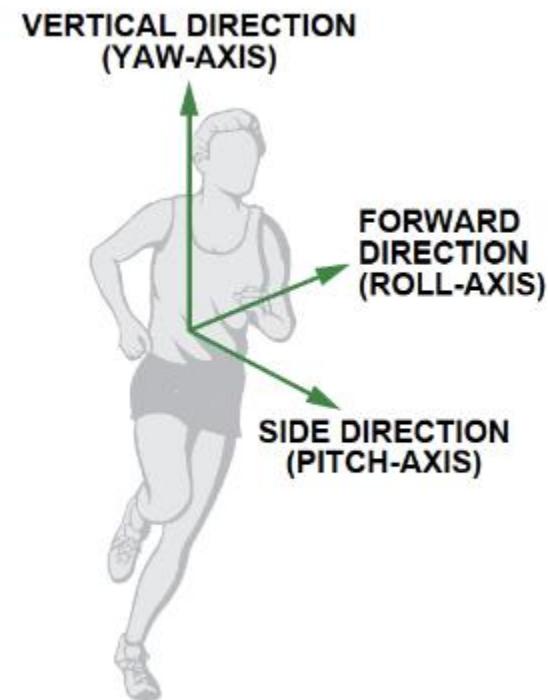
- Everyone is crazy about step count these days
- Pedometer apps, pedometers, fitness trackers, etc
- Tracking makes user aware of activity levels, motivates them to exercise more



# Benefits mobile step counters

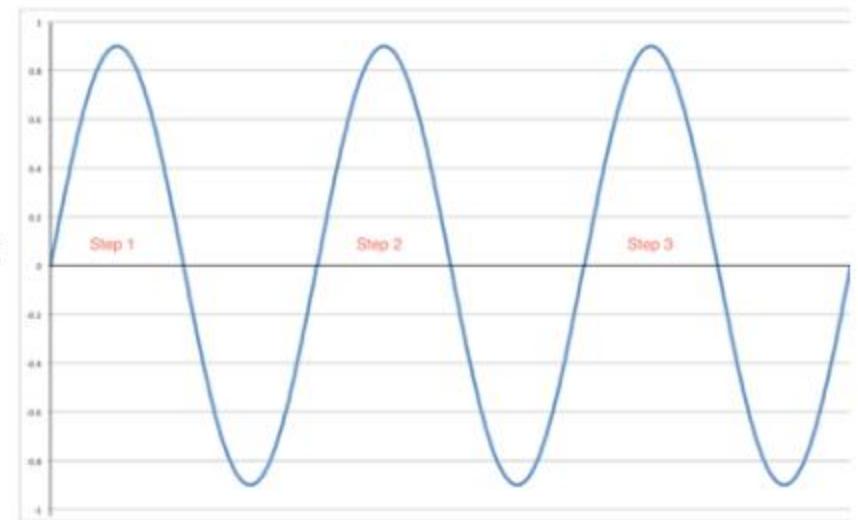
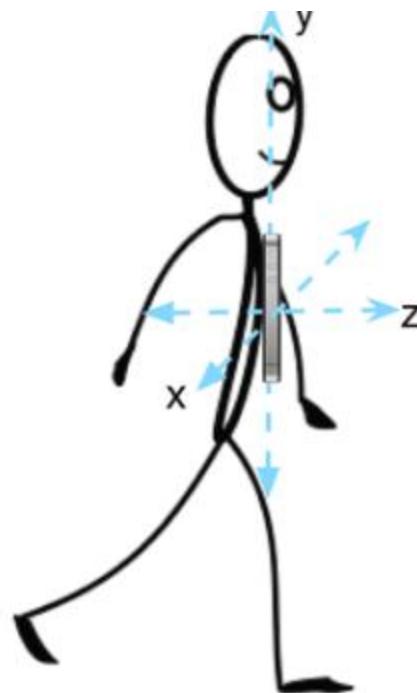
- Always on. Everywhere. Continuous monitoring.
- Low-cost
- Better privacy than computer vision

# *Definition of each axis*



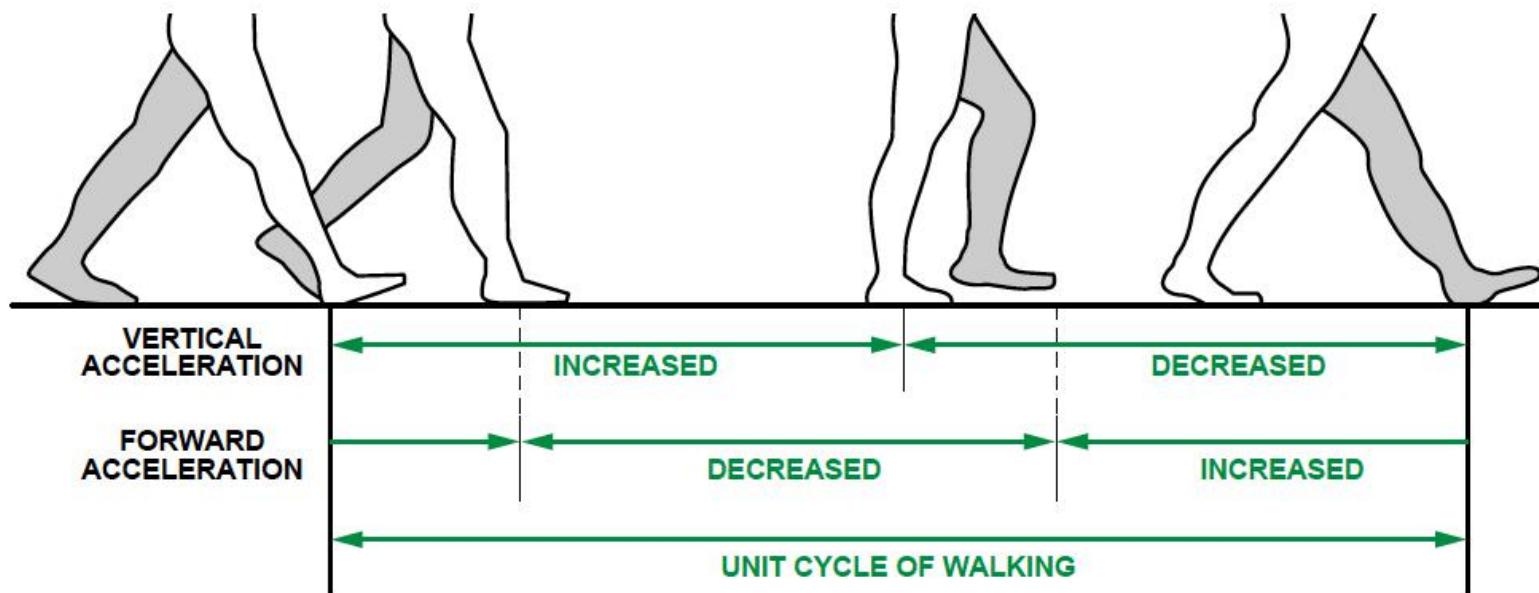
# Ideal sensor data

- In an ideal situation, you would like to see a signal like this.
  - One of the axes of the phone is along the direction of gravity, and the steps can be clearly observed in the signal.
- But in practice, the data often deviates from the ideal case



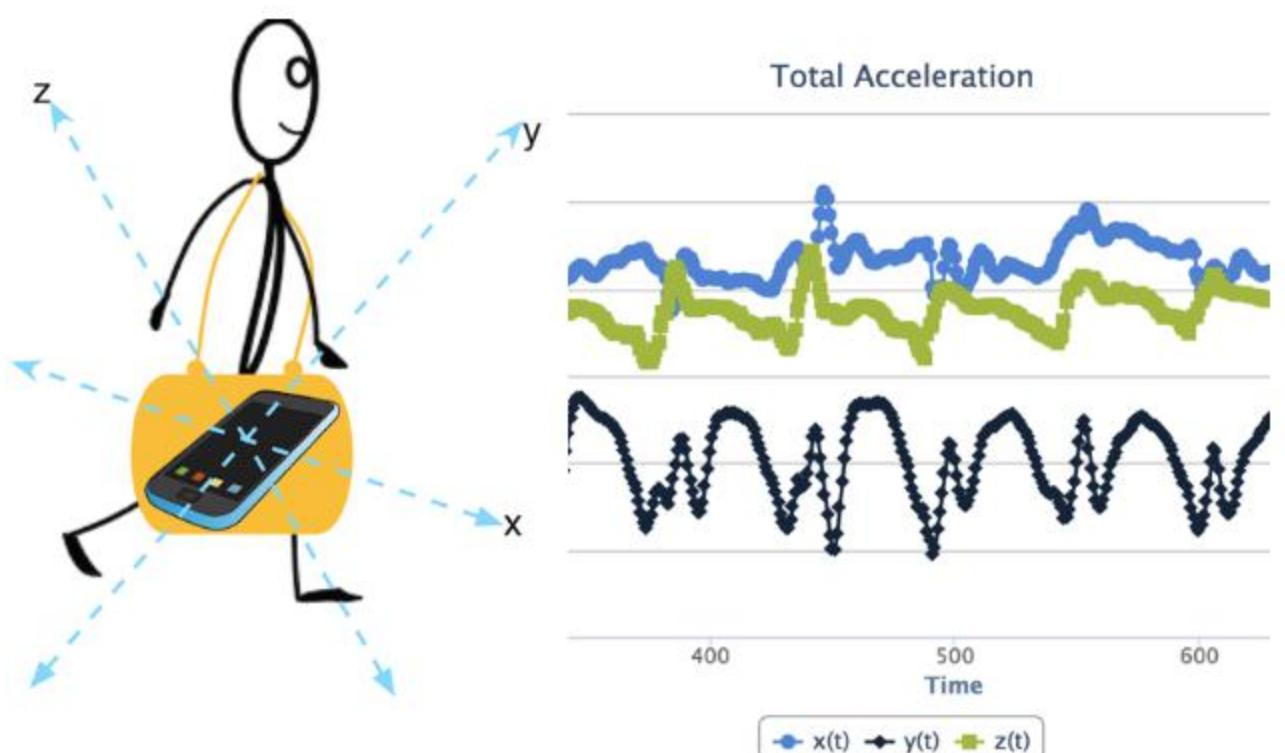
# The Nature of Walking

- Vertical and forward acceleration increases/decreases during different phases of walking
- Walking causes a large periodic spike in one of the accelerometer axes
- Which axes (x, y or z) and magnitude depends on phone orientation



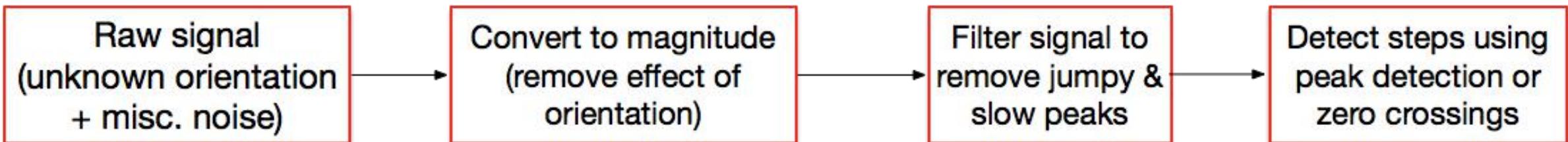
# Accelerometer signal in a real-world situation

- A more realistic signal is shown
  - some component of the user acceleration and gravity is present along all three axes.
  - The measurements are influenced by the phone orientation
- we need to design an *orientation-independent* algorithm to detect steps.



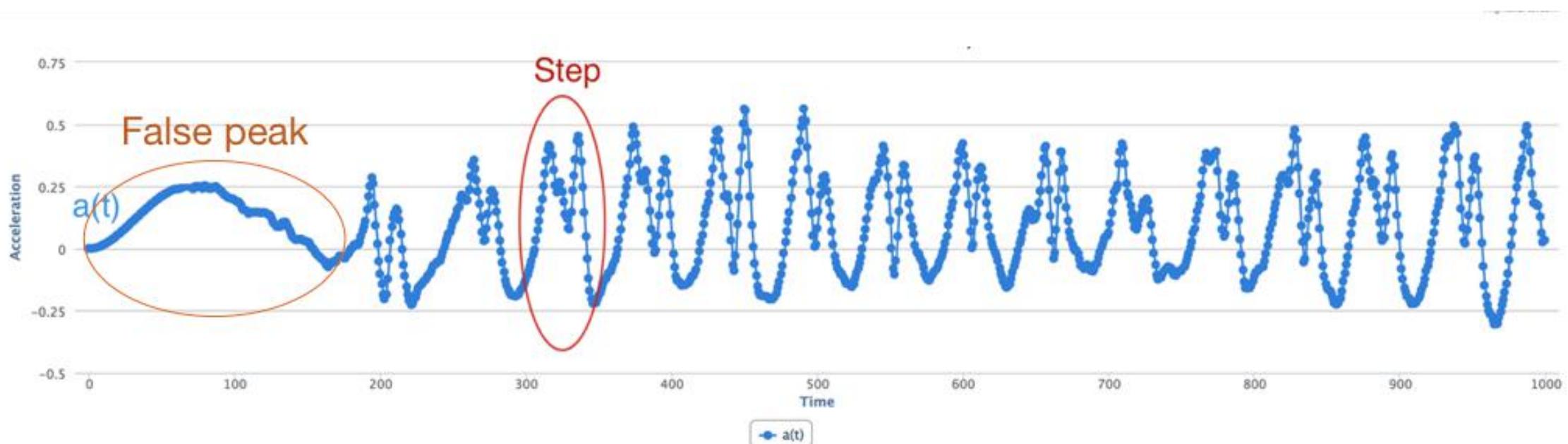
# Step Detection Algorithm

- The key insight in our method is to convert the 3-axis signal into a one axis magnitude signal, and then extract steps from this signal.



# Step 1: Extract Signal Magnitude

- take the magnitude of the entire acceleration vector i.e.  $\sqrt{x^2 + y^2 + z^2}$ , where x, y, and z are the readings of the accelerometer along the three axes.
- The signal is not dependent on phone orientation now



# Step 2: Filter the signal to remove noise

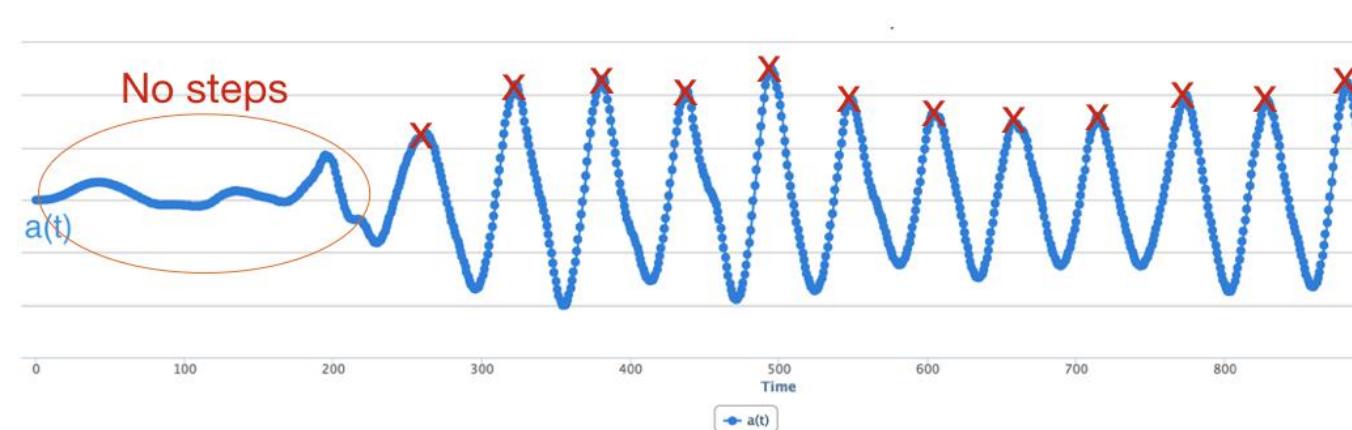
- What noises exist in the data?
  - **Jumpy peaks:** Since the phone is often carried in a pocket/purse, it can jiggle a little with each step. Also, some users have a bounce in their step, so even though they are taking a single step, the phone can bounce multiple times within this step.
  - **Short peaks:** Small peaks can occur when a user is using a phone (e.g. making a call or using an app).
  - **Slow peaks:** Slow peaks can occur when the phone is moved or due to movements of the leg while sitting (if the phone is in the pant pocket)

# Filtering

- To remove these sources of noise, we are going to use frequency-domain noise removal.
- Notice that we need to remove high frequency variations like jumpy peaks and low frequency variations like slow peaks.
- A simple solution is to use a filter that keeps only frequencies relating to walking and removes the rest.

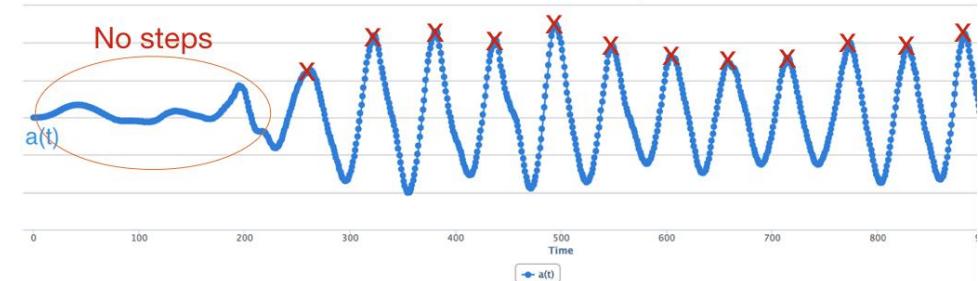
# Filtering

- Typical walking pace may be under three steps a second (3 Hz) and over half step a second (0.5Hz), so we remove all frequencies above 5 Hz and below 0.5 Hz (just to give some margin for error)
- Even after we remove low and high frequency peaks, we may be left with some short peaks.
  - A simple way to deal with this is to look only for large peaks and ignore small peaks.



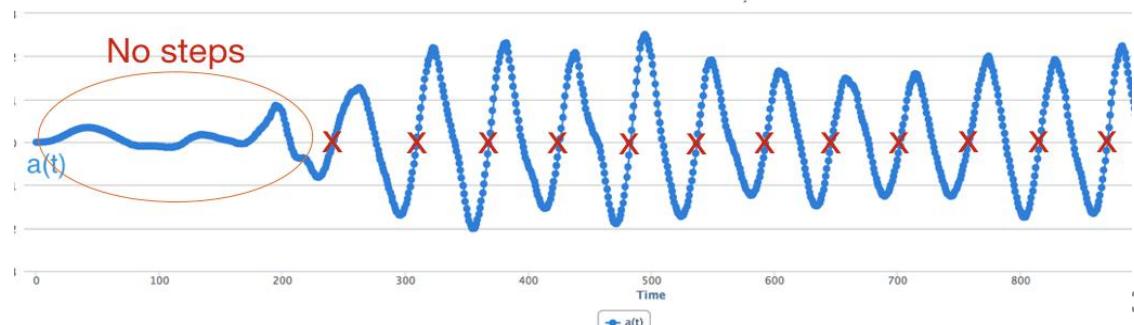
# Step 3: Detect Steps

- Approach 1: Find signal peaks



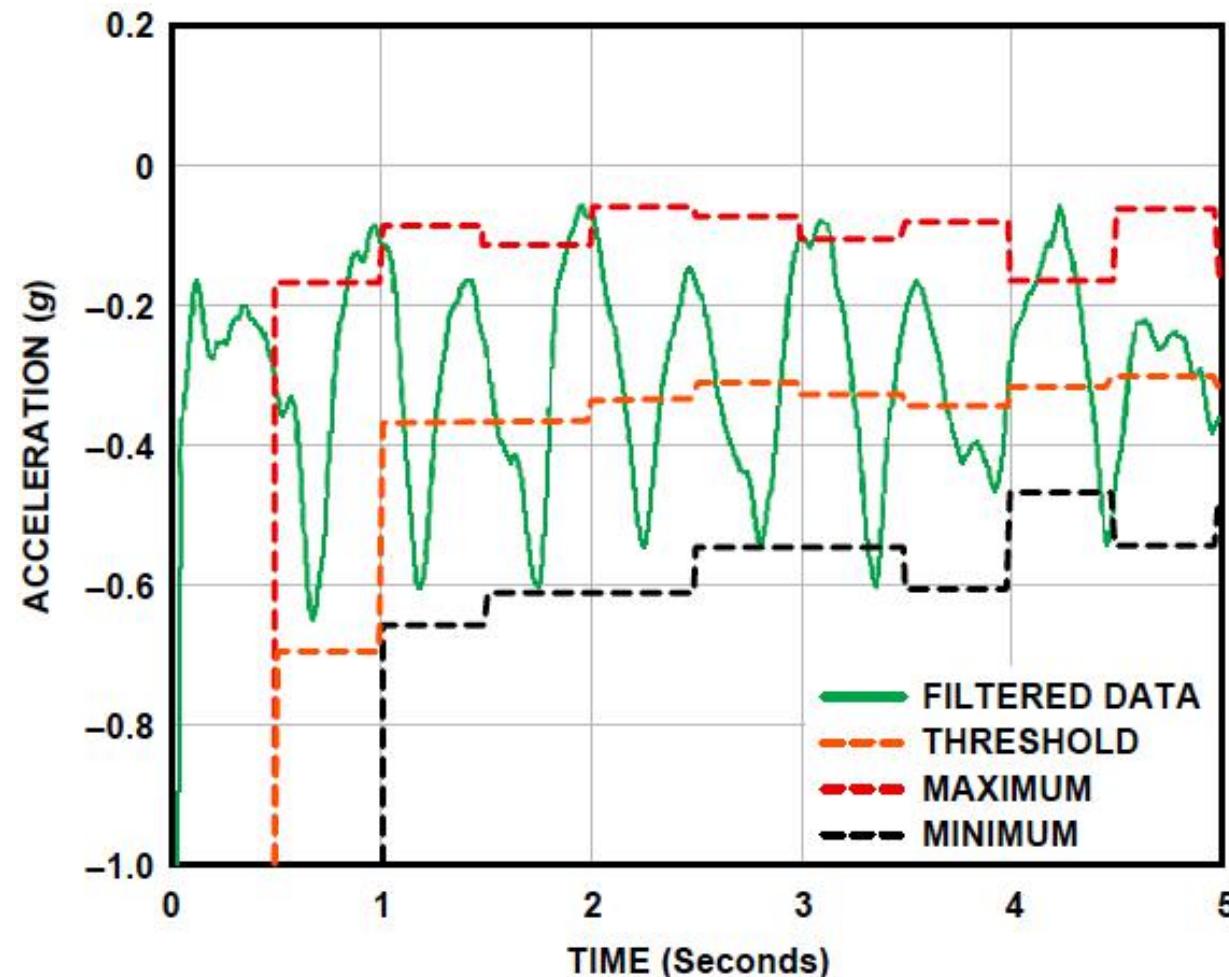
- Approach 2: Zero crossing:

- Subtract the mean for each window and look at zero crossings i.e. times when the signal crosses from the negative to positive in the upward direction



# Dynamic Threshold-based Step Detection

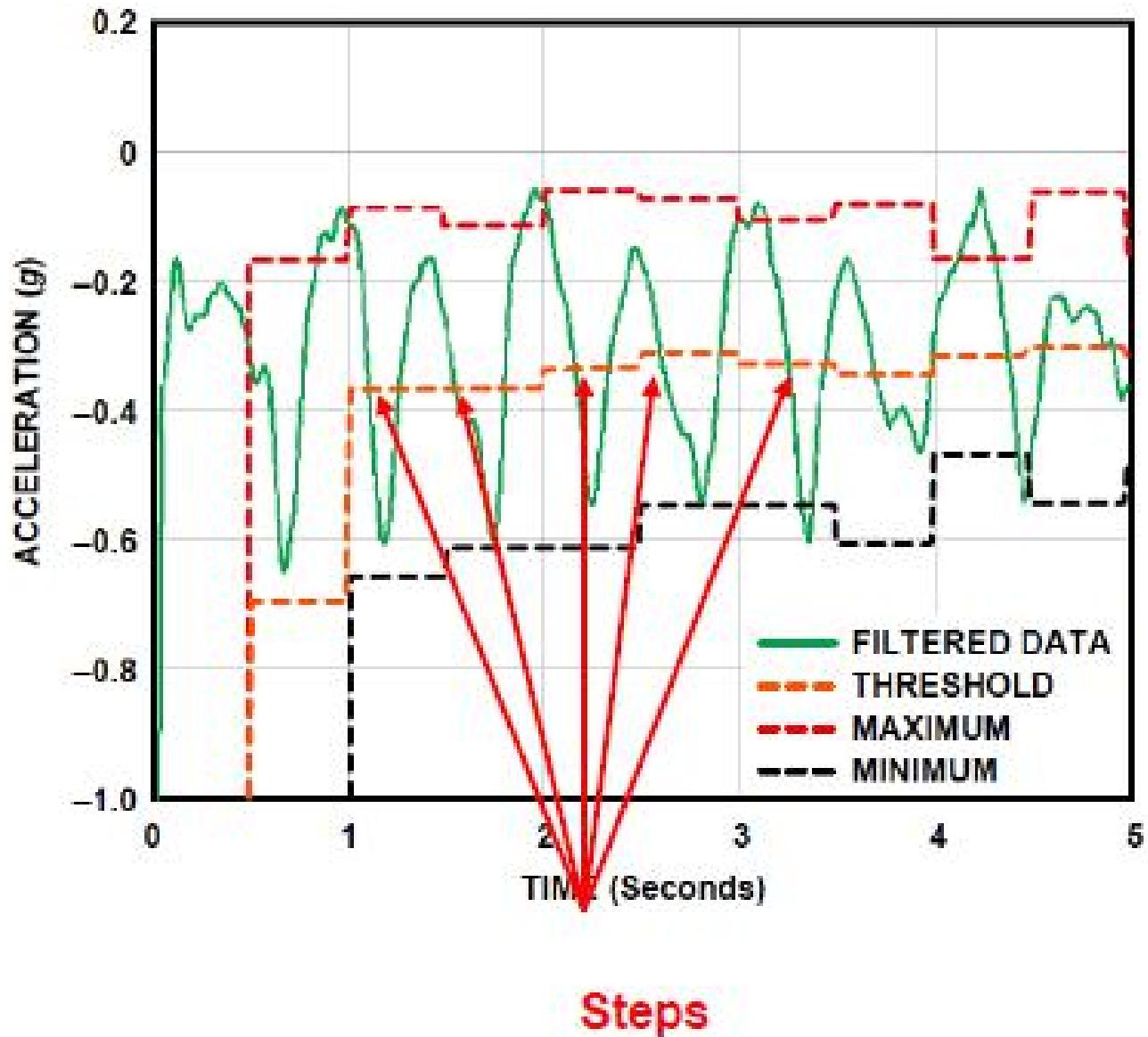
- Focus on accelerometer axis with largest peak
- Would like a threshold such that each crossing is a step
- Track min, max values observed every 50 samples
- Compute ***dynamic threshold***:  $(\text{Max} + \text{Min})/2$



# Step Detection Algorithm

A step is

- Indicated by crossings of dynamic threshold
- Defined as negative slope ( $\text{sample\_new} < \text{sample\_old}$ ) when smoothed waveform crosses dynamic threshold



# Distance Estimation

- Calculate distance covered based on number of steps taken
  - $Distance = number\ of\ steps \times distance\ per\ step$
- Distance per step (stride) depends on user's height (taller people, longer strides), and step frequency
- Using person's height, can estimate their stride, then number of steps taken per 2 seconds

Steps per 2 s	Stride (m/s)
0~2	Height/5
2~3	Height/4
3~4	Height/3
4~5	Height/2
5~6	Height/1.2
6~8	Height
$\geq 8$	$1.2 \times Height$

# Calorie Estimation

- To estimate speed, remember that speed = distance/time. Thus,
  - $\text{Speed (in m/s)} = (\text{no. steps per 2 s} \times \text{stride (in meters)})/2\text{s}$
- Calorie expenditure, which depends on many factors
  - Body weight, workout intensity, fitness level, etc
- Empirical simplified equation:
  - $\text{Calories (C/kg/h)} = 1.25 \times \text{speed (m/s)} \times 3600/1000 = 4.5 \times \text{speed (m/s)}$

# Limitations and Future Work

- Strong assumptions on how the users walk.
  - What about short-interval, high intensity exercise?
  - What about the other calorie expenditures? Standing vs sitting
- Currently, dedicated system for each activities. General activity recognition is still under research.



# CSE 162 Mobile Computing

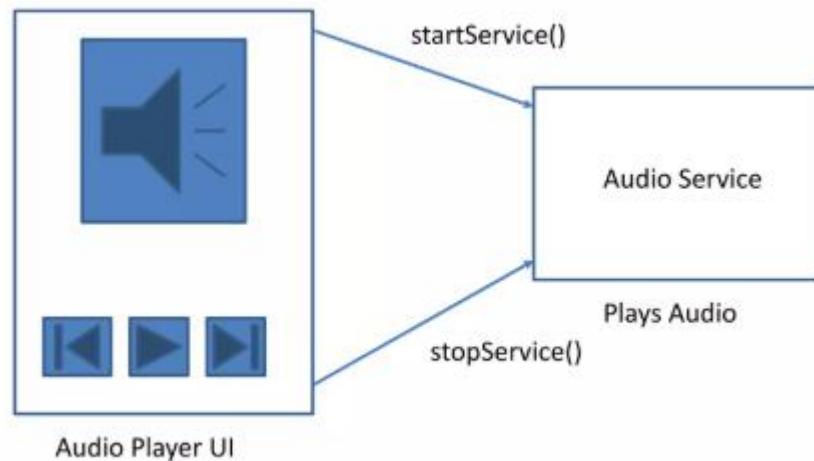
## Mobile Activity Recognition

Hua Huang

# Services

# What is Android Service?

- Services are codes that run in the background
- They can be started and stopped
- Services doesn't have UI



# What a Service is NOT?

- There are some confusions:
- A Service is not a separate process.
  - The Service object itself does not imply it is running in its own process; unless otherwise specified, it runs in the same process as the application it is part of.
- A Service is not a thread.
  - It is not a means itself to do work off of the main thread (to avoid Application Not Responding errors).

# Main Features of Service

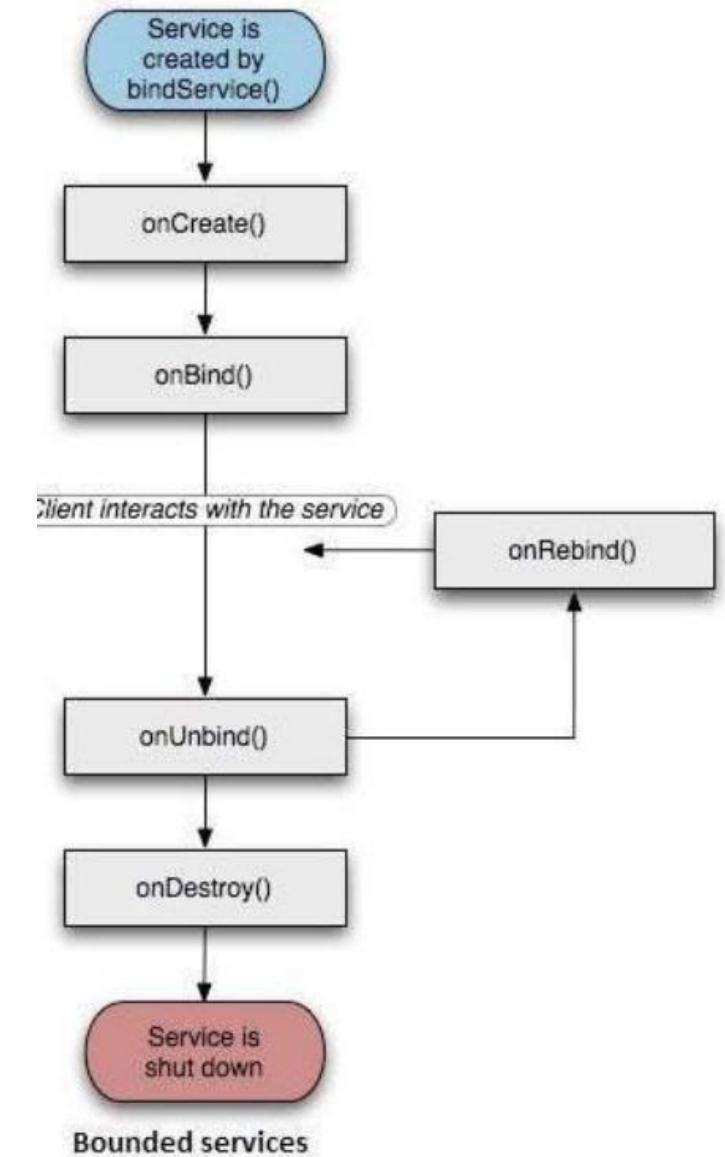
- To tell the system about something it wants to be doing in the background (even when the user is not directly interacting with the application).
- To call `Context.startService()`, which asks the system to schedule work for the service, to be run until the service or someone else explicitly stops it.

# When to use Services

- Activities are short-lived, can be shut down anytime (e.g when user presses back button)
- Services keep running in background
- Similar to Linux/Unix CRON job
- Example uses of services:
  - Periodically check/update device's GPS location
  - Check for updates to RSS feed
  - Independent of any activity, minimal interaction
- Typically an activity will control a service --start it, pause it, get data from it
- Services in an App are sub-class of Android's **Services** class

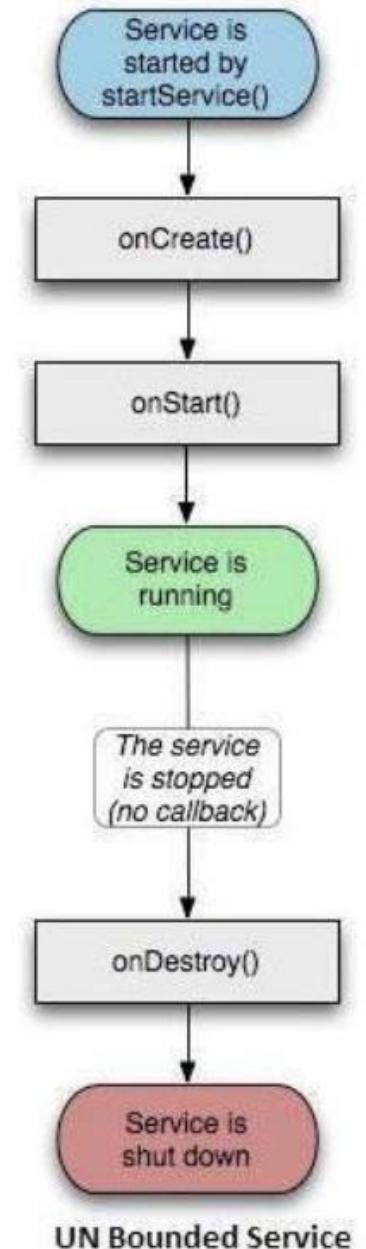
# Bound Service

- An Android component may bind itself to a Service using `bindService()`.
- A bound service would run as long as the other application components are bound to it.
- As soon as they unbind, the service destroys itself.



# Unbound Service

- In this case, an application component starts the service, and it would continue to run in the background, even if the original component that initiated it is destroyed.
- For instance, when started, a service would continue to play music in the background indefinitely.



UN Bounded Service

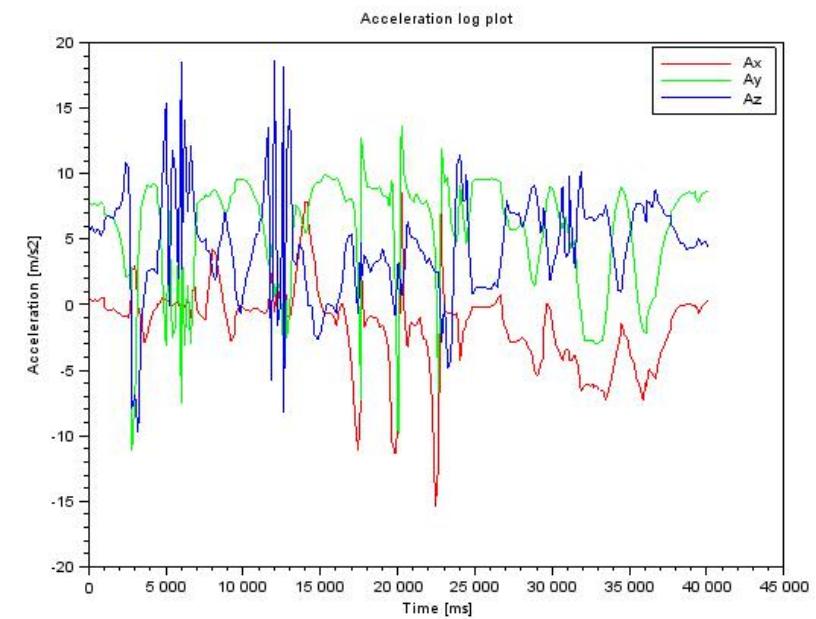
# Android Platform Services

- Android Services can either be on:
  - On smartphone or Android device (local)
  - Remote, on Google server/cloud
- Android platform local services examples (on smartphone):
  - **LocationManager**: location-based services.
  - **ClipboardManager**: access to device's clipboard, cut-and-paste content
  - **DownloadManager**: manages HTTP downloads in background
  - **FragmentManager**: manages the fragments of an activity.
  - **AudioManager**: provides access to audio and ringer controls.

# An Introduction to Human Activity Recognition

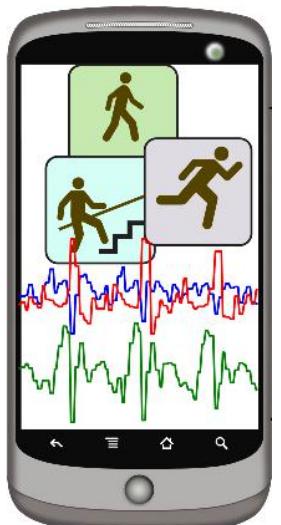
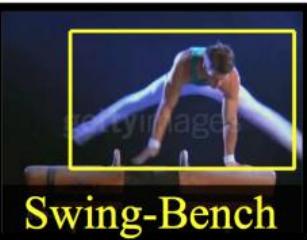
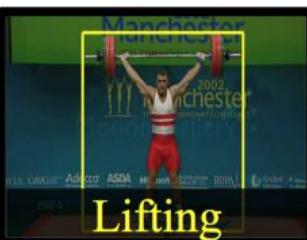
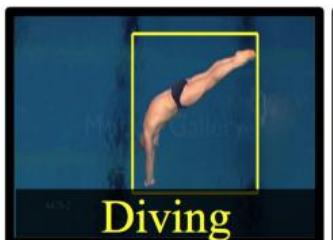
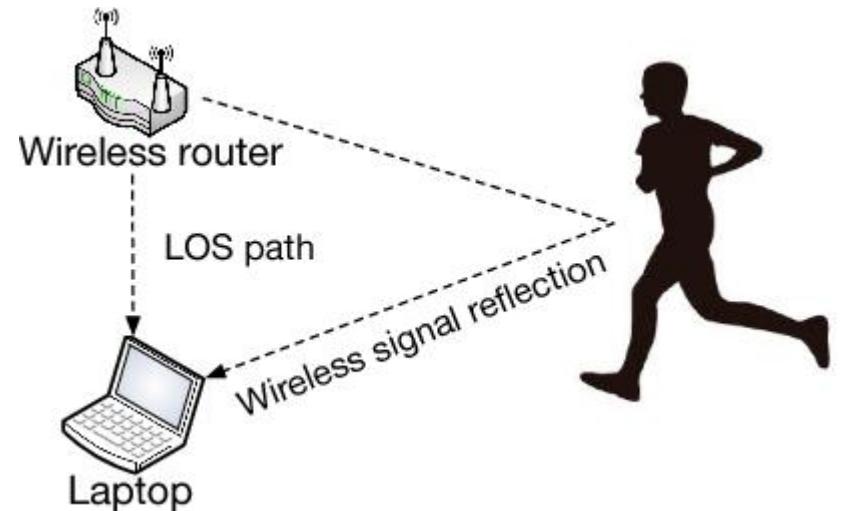
# Activity Recognition

- Goal: Want our app to detect what activity the user is doing?
- Classification task: which of these 6 activities is user doing?
  - Walking,
  - Jogging,
  - Ascending stairs,
  - Descending stairs,
  - Sitting,
  - Standing
- Typically, use machine learning classifiers to classify user's accelerometer signals

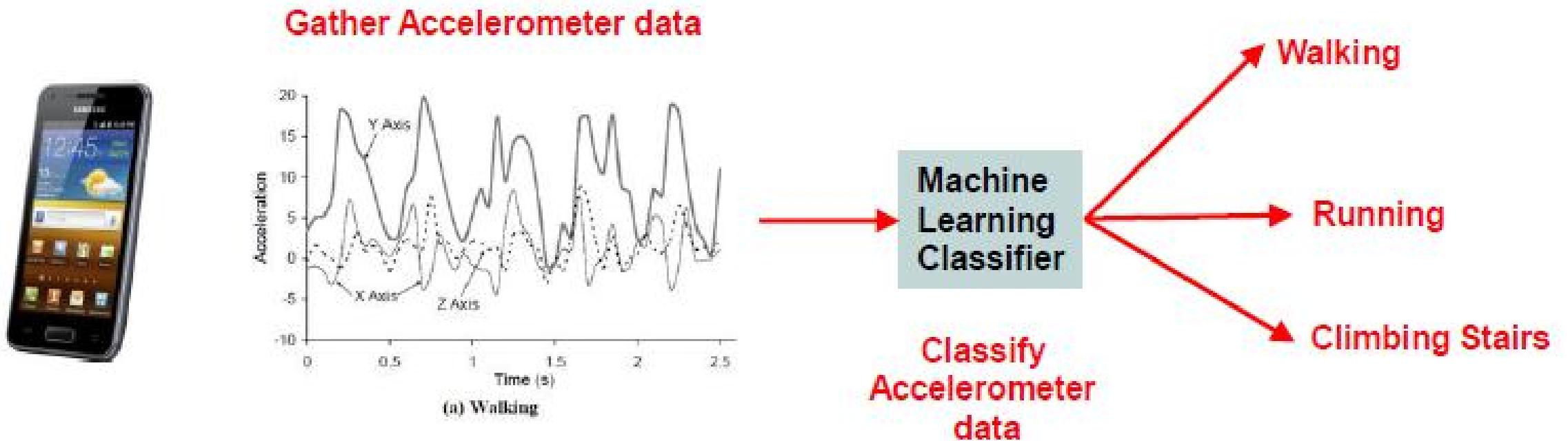


# Activity Recognition Systems

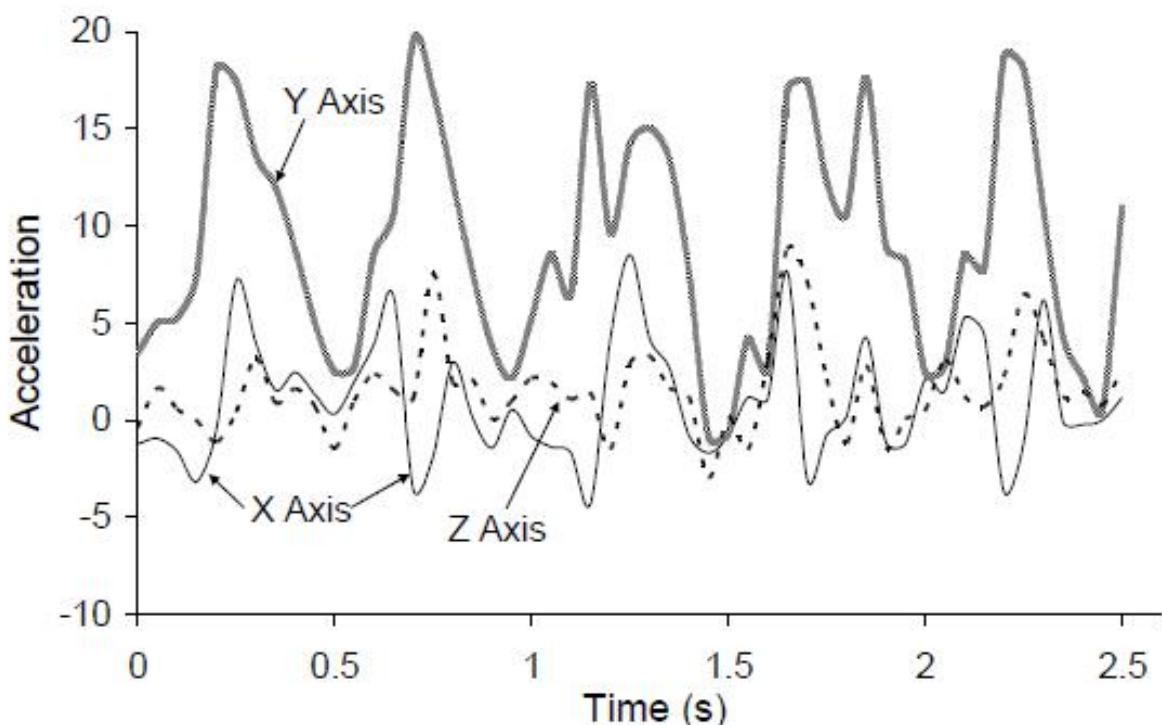
- Computer vision based
- Environmental sensor based
- Mobile sensor based



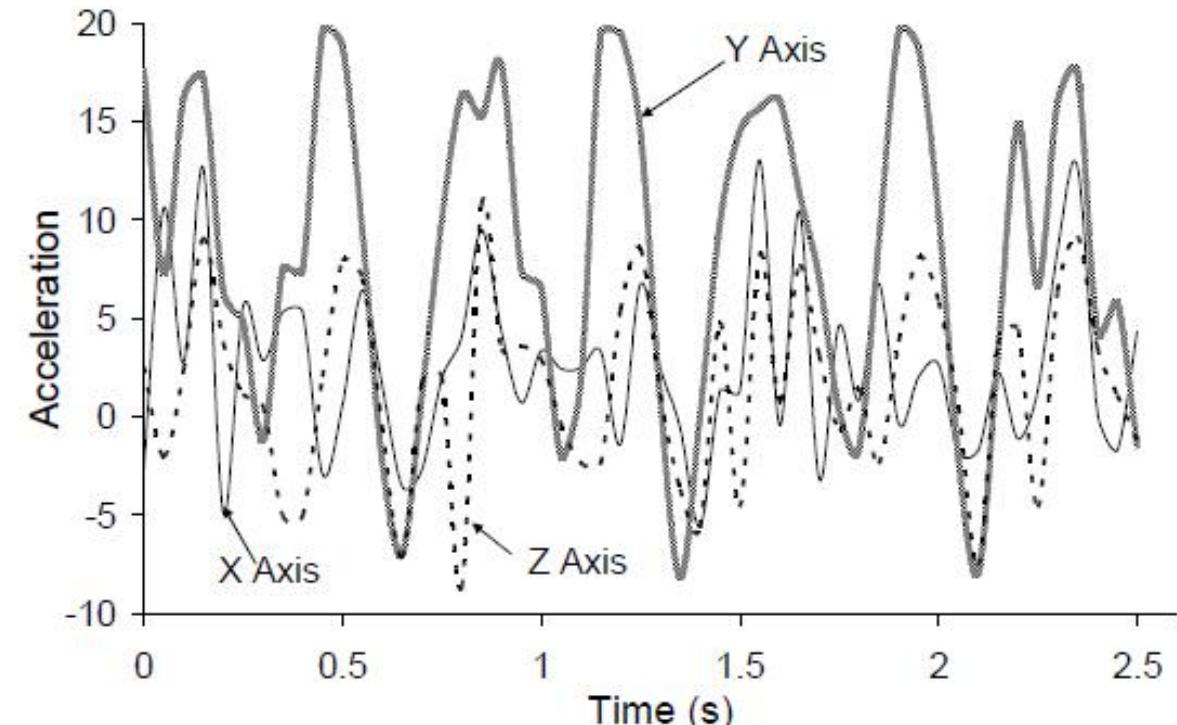
# Mobile Activity Recognition Architecture



# Sample Accelerometer during activities

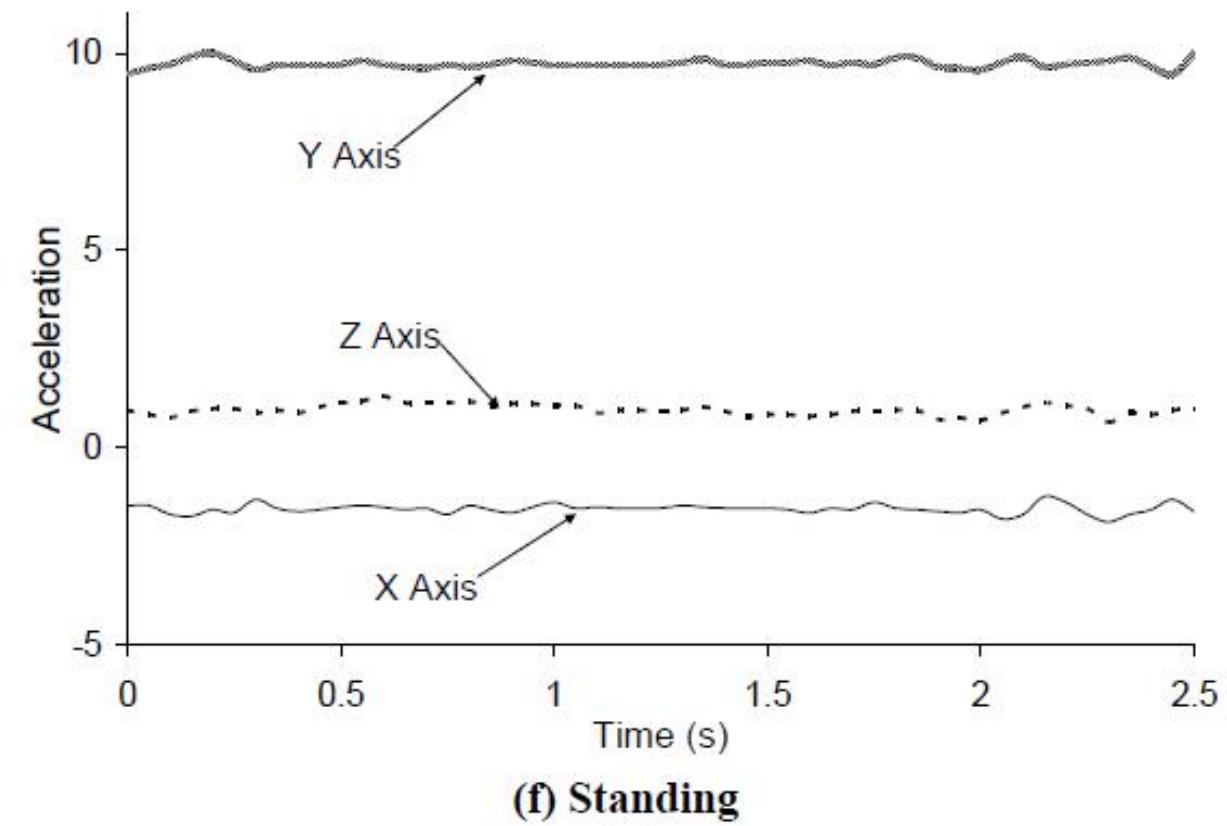
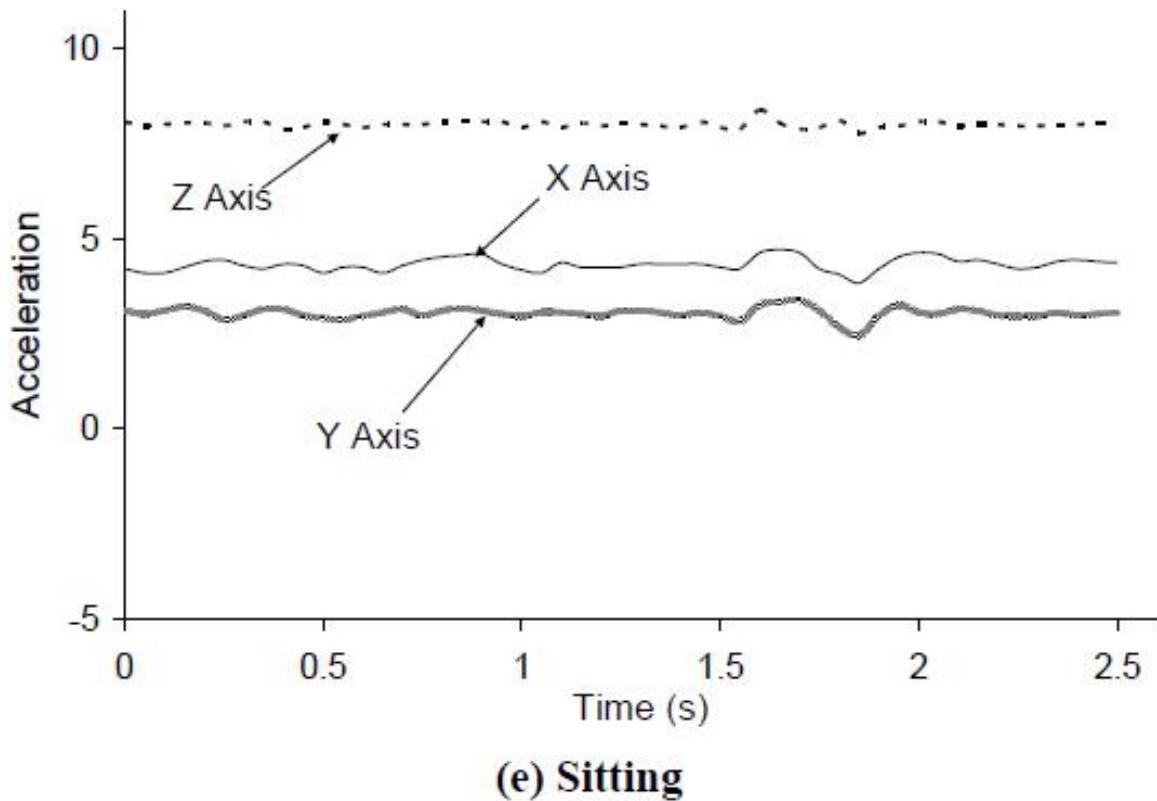


(a) Walking



(b) Jogging

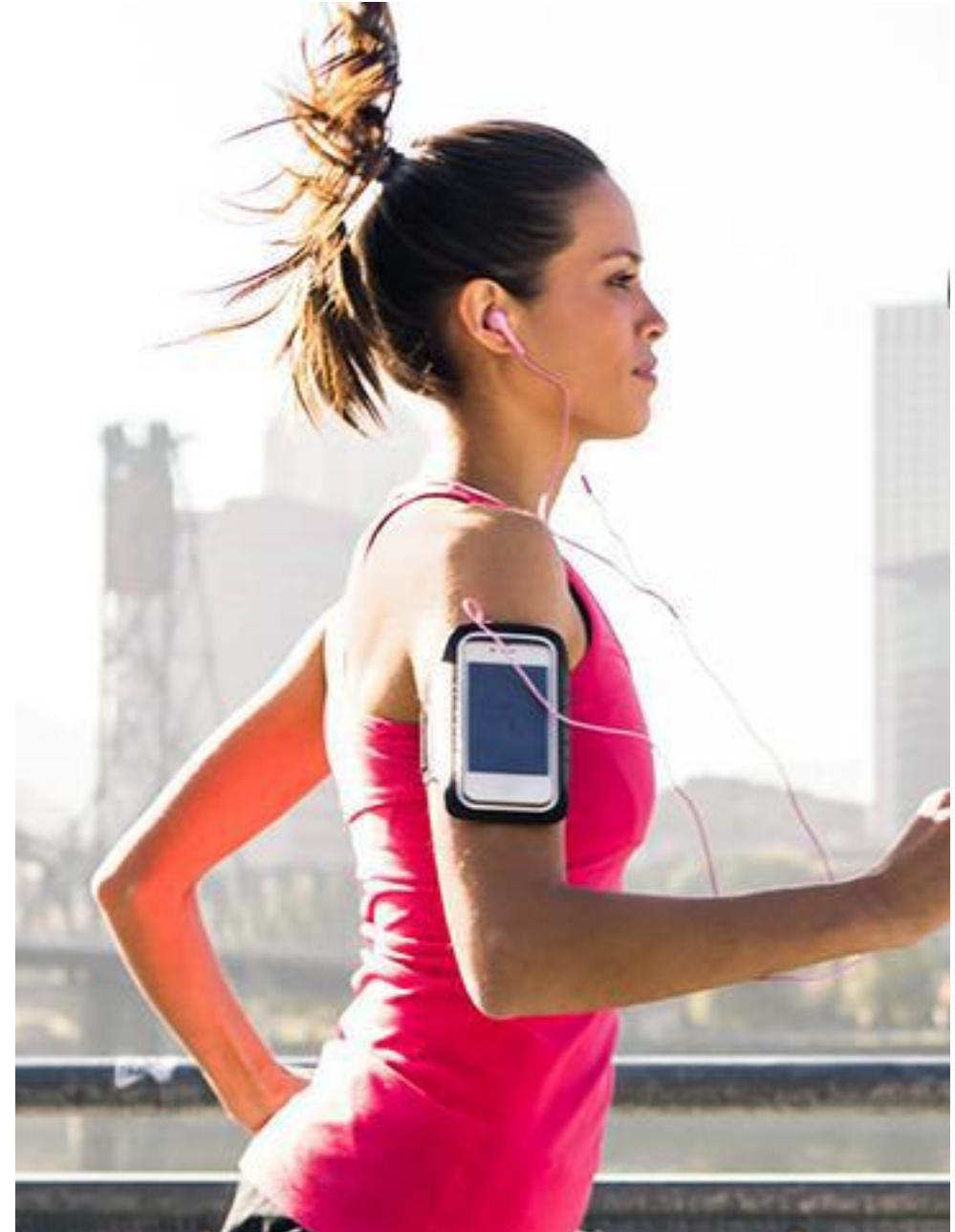
# Sample Accelerometer during activities



# Activity Recognition Applications

# Fitness Tracking

- **Initially:**
  - Physical activity type,
  - Distance travelled,
  - Calories burned
- **Newer features:**
  - Stairs climbed,
  - Physical activity (duration + intensity)
  - Activity type logging + context
  - Sleep tracking
  - Activity history



# Health Monitoring

- Make clinical monitoring pervasive, continuous, real world!!
  - Gather context information (e.g. what makes condition worse/better?)
  - E.g. timed up and go test
- Show patient contexts that worsen condition => Change behavior
  - E.g. walking in narrow hallways worsens gait freeze



Gait Freezing

**Question: What data would you need to build PD gait classifier?  
From what types of subjects?**

# Fall Detection

- A leading cause of death for seniors
- Smartphone/watch, wearable detects senior who has fallen, alert family
  - Text message, email, call relative



# Context-aware Behavior

- Study found that messages delivered when transitioning between activities better received
  - In-meeting? => Phone switches to silent mode
  - Exercising? => Play song from playlist, use larger font sizes for text
  - Arrived at work? => download email
- Adaptive Systems to Improve User Experience:
  - Walking, running, riding bike? => Turn off Bluetooth, WiFi (save energy)
  - an increase battery life up to 5x

# Smart Home

- Smart Thermostat
  - Determines if the users are at home
  - Recognize where the users are and adjust temperature accordingly
- Smart TV
  - Turns on TV when the user is at the couch



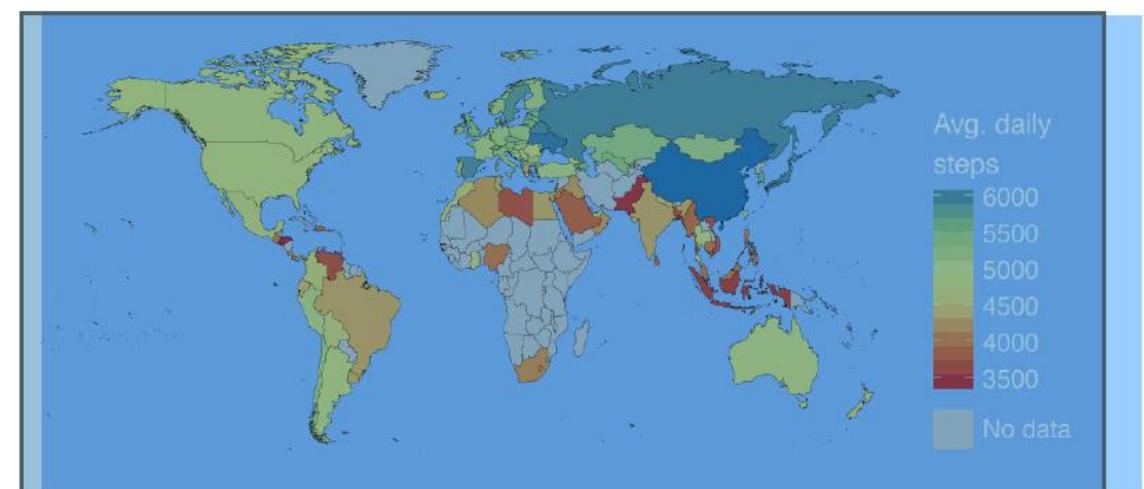
# Targeted Advertisements

- User runs a lot => Get exercise clothing ads
- Goes to pizza places often + sits there => Get pizza ads



# Research Platforms for Data Collection

- E.g. public health officials want to know how much time various people (e.g. students) spend sleeping, walking, exercising, etc
- Mobile AR: inexpensive, automated data collection
- E.g. Stanford Inequality project: Analyzed physical activity of 700k users in 111 countries using smartphone AR data
  - <http://activityinequality.stanford.edu/>



# Track, manage staff on-demand

- E.g. at hospital, determine “availability of nurses”, assign them to new jobs/patients/surgeries/cases

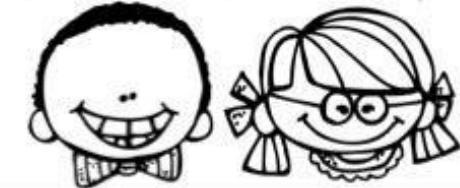


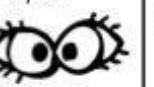
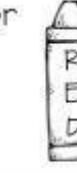
# Activity-Based Social Networking

- Automatically connect users who do same activities + live close together

**Find a friend who . . .**

name \_\_\_\_\_



has a pet dog 	has black hair 	likes to play soccer 	has a blue backpack 
has a brother 	likes to color 	has a summer birthday 	likes chocolate ice cream 
likes to eat pizza 	can play an instrument 	has a sister 	likes to swim 
has brown eyes 	is wearing white shoes 	likes the color red 	has a pet cat 

©2014 First Grade Schoolhouse

# Activity Recognition using Google API

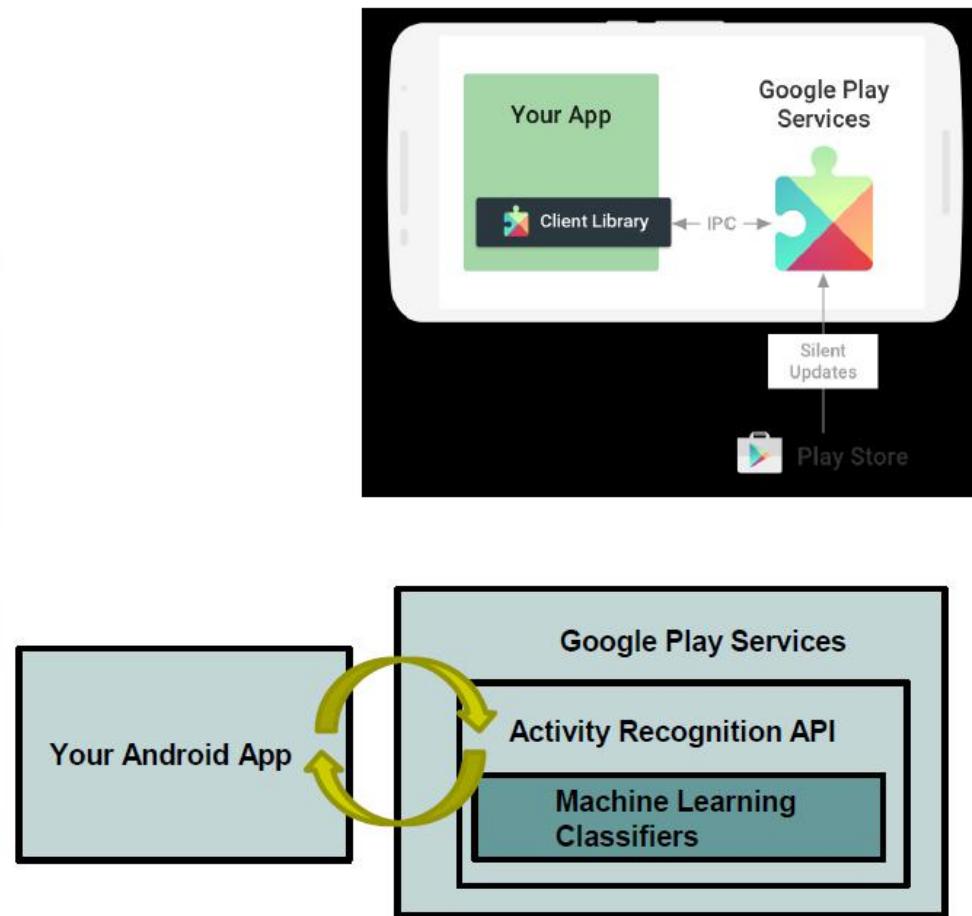
- We can adapt the mobile behaviors based on the user's behaviors
  - E.g., If the user is driving, don't send notifications

# Google Activity Recognition API

- API to detect smartphone user's current activity
- Programmable, can be used by your Android app
- Currently detects 8 states:
  - In vehicle
  - On Bicycle
  - On Foot
  - Running
  - Walking
  - Still
  - Tilting
  - Unknown

# Google Activity Recognition API

- Deployed as part of Google Play Services



# Activity Recognition API

- Understanding what users are doing in the physical world allows your app to be smarter about how to interact with them.
  - For example, an app can start tracking a user's heartbeat when she starts running,
  - another app can switch to *car mode* when it detects that the user has started driving.
- The Activity Recognition API is built on top of the sensors available in a device.
  - Device sensors provide insights into what users are currently doing.

- The Activity Recognition API automatically detects activities by periodically reading short bursts of sensor data and processing them using machine learning models.
- To optimize resources, the API may stop activity reporting if the device has been still for a while, and uses low power sensors to resume reporting when it detects movement.

# Features of Google API

- Receive information about activities using minimal resources
  - Get notified when your user starts or ends a particular activity
  - Perform an action when your app receives activity information
  - Receive detected activities that include a confidence grade

# Get notified when your user starts or ends a particular activity

- For example, a mileage tracking app could start tracking miles when a user starts driving, or a messaging app could mute all conversations until the user stops driving.
- The Activity Recognition Transition API detects changes in the user's activity.
- The app subscribes to a transition in activities
  - The API notifies your app only when needed.
- No need to implement complex heuristics to detect when an activity starts or ends.

# Perform an action when your app receives activity information

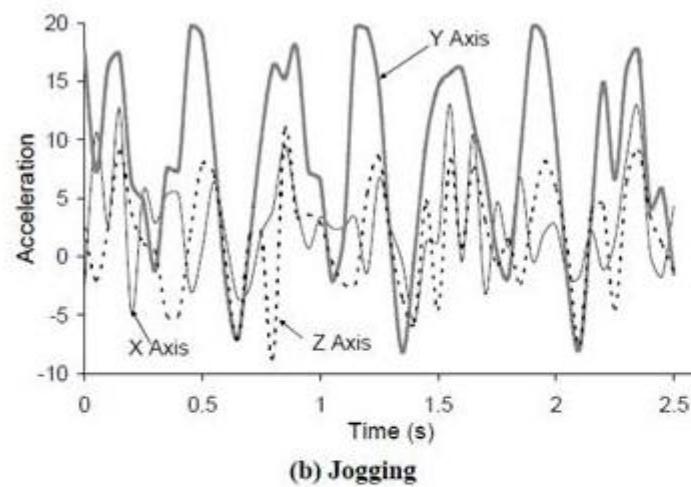
- The Activity Recognition API delivers its results to a callback, which is usually implemented as an IntentService
- The results are delivered at intervals that you specify,
  - or your app can use the results requested by other clients without consuming additional power itself.
- You can tell the API how to deliver results by using a PendingIntent
  - It removes the need to have a service constantly running in the background for activity detection purposes.
  - The app receives the corresponding Intents from the API, extracts the detected activities, and decides if it should take an action.

# Receive detected activities that include a confidence grade

- The Activity Recognition API does the heavy lifting by processing the signals from the device to identify the current activities.
- Your app receives a list of detected activities, each of which includes confidence and type properties.
  - The confidence property indicates the likelihood that the user is performing the activity represented in the result.
  - The type property represents the detected activity of the device relative to entities in the physical world, for example, the device is on a bicycle or the device is on a user who is running.

# Activity Recognition System Design

# Activity recognition on Android



New accelerometer  
Sample in real time

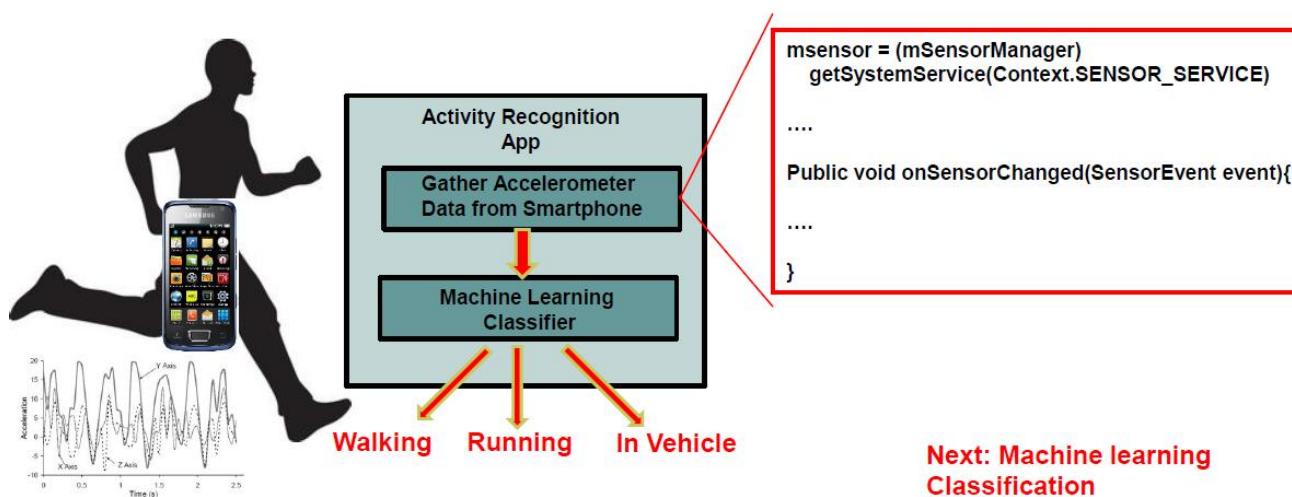


Classifier  
in  
Android app

Activity  
(e.g. Jogging)

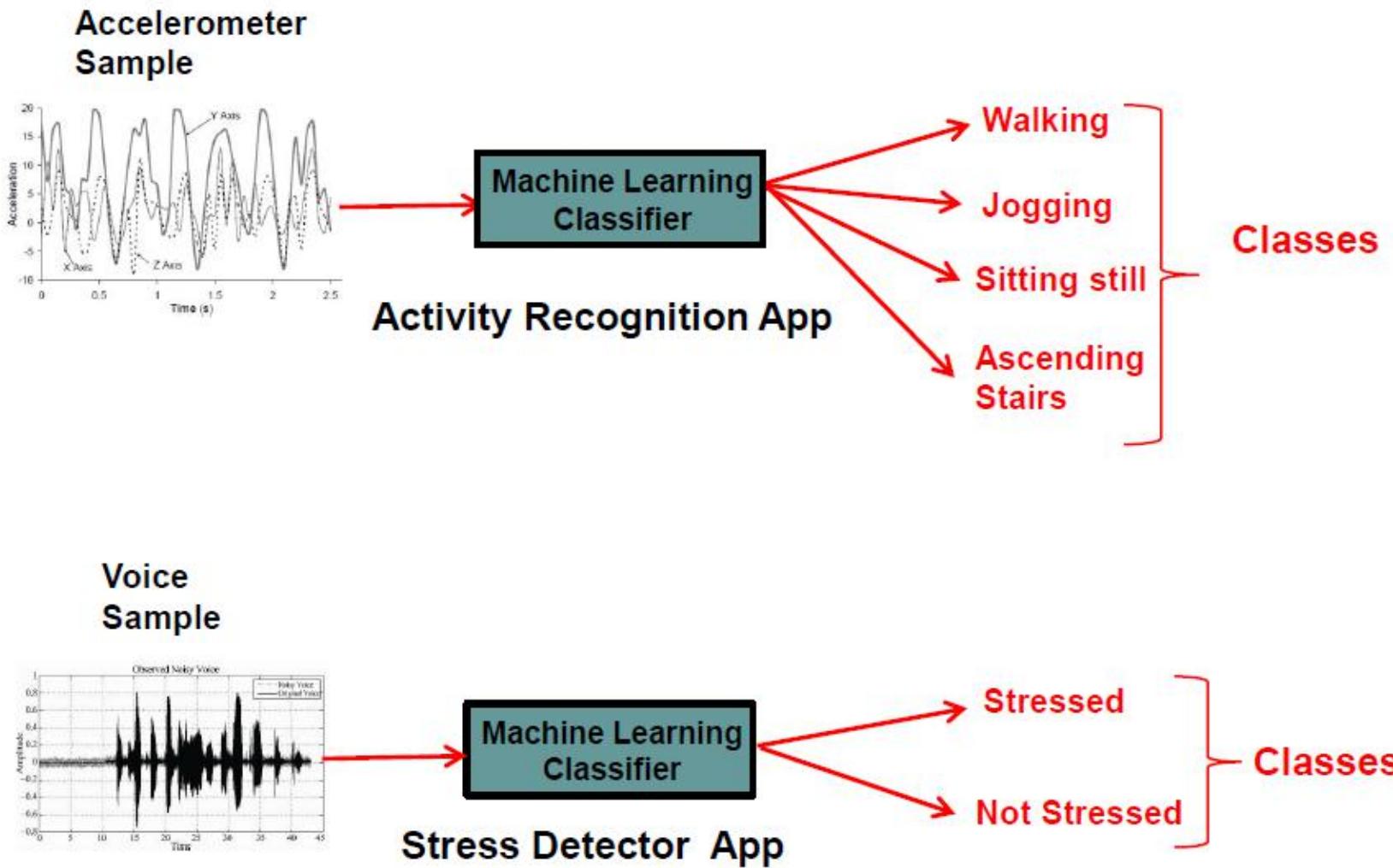
# Activity Recognition (AR) Android App

- As user performs an activity, AR app on user's smartphone
  - Gathers accelerometer data
  - Uses **machine learning classifier** to determine what activity (running, jumping, etc) accelerometer pattern corresponds to
- **Classifier:** Machine learning algorithm that guesses what activity **class**(or type) accelerometer sample corresponds to

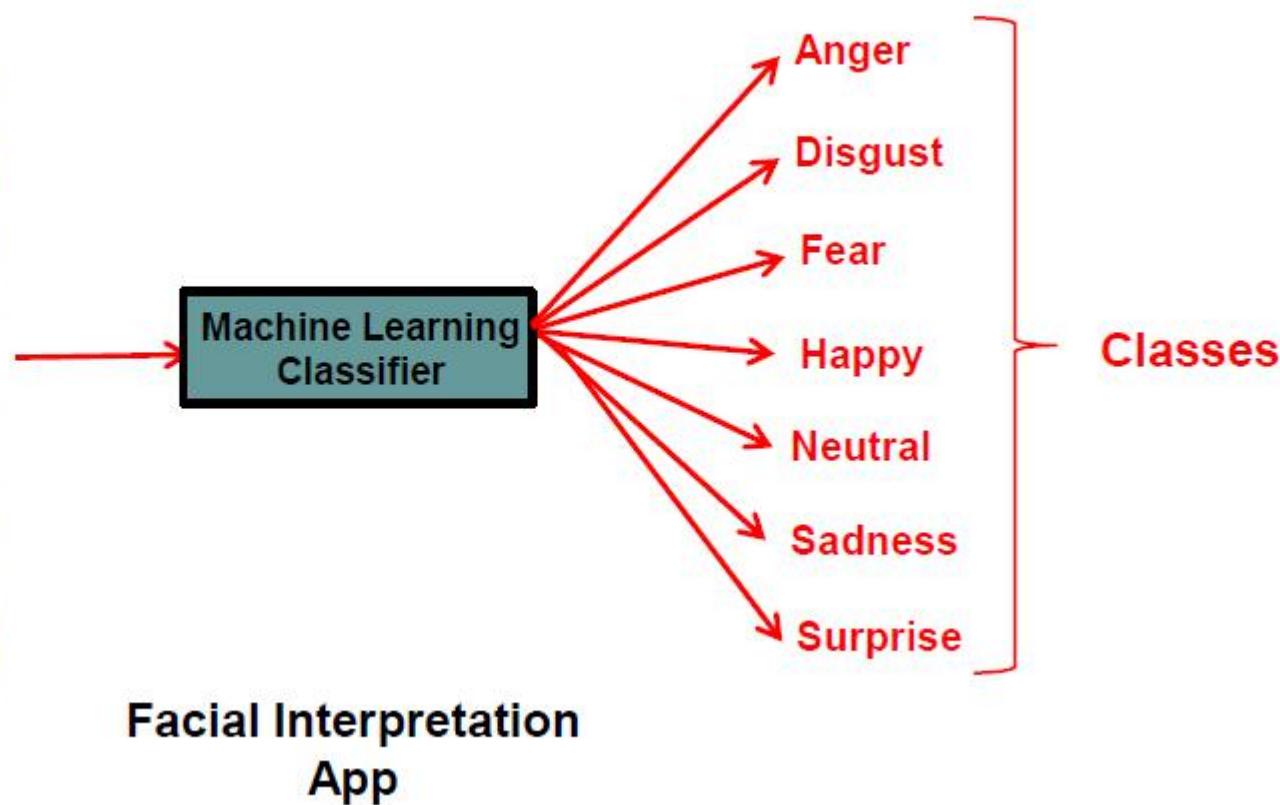
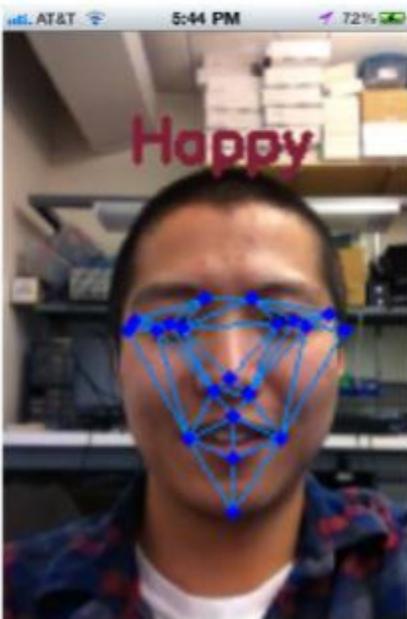


# Classification

- Classification?  
determine which **class** a sample  
(e.g. snippet of accelerometer data) belongs to.  
Examples:



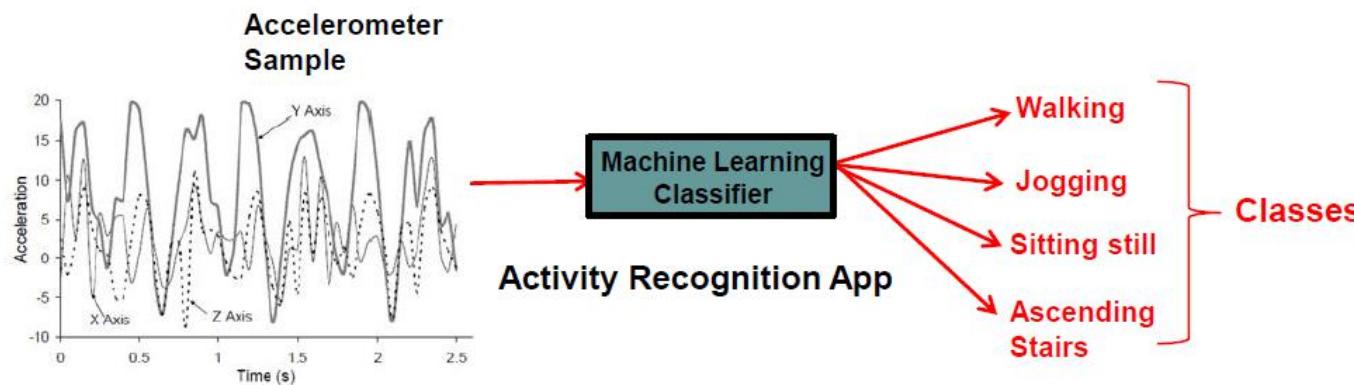
**Image showing  
Facial Expression**



# Classifier

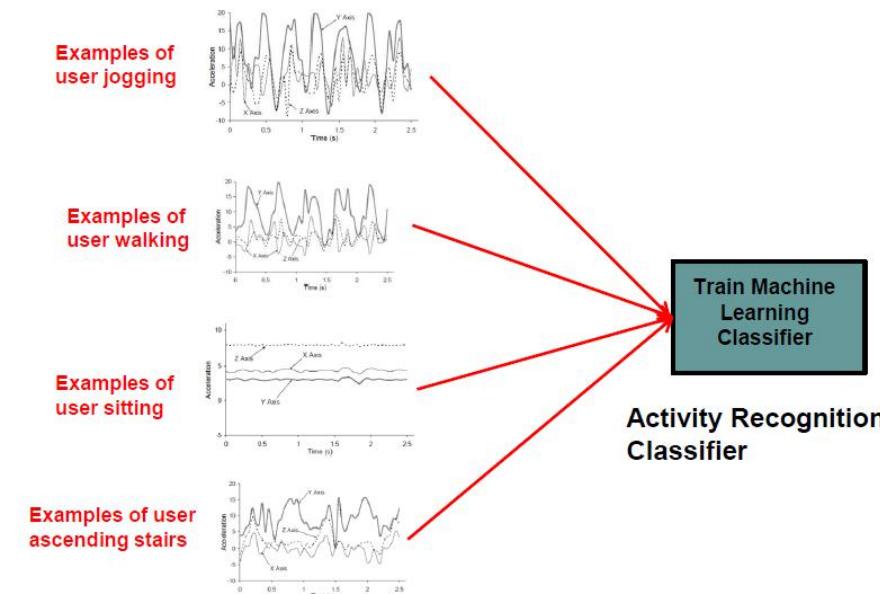
- Analyzes new sample, guesses corresponding class
- Intuitively, can think of classifier as set of rules for classification
- Example rules for classifvina accelerometer signal in Activity Recognition

```
If ((Accelerometer peak value > 12 m/s)
    and (Accelerometer average value < 6 m/s)) {
    Activity = "Jogging";
}
```



# Training a Classifier

- Created using example-based approach (called training)
- **Training a classifier:** Given examples of each class => generate rules to categorize new samples
- **E.g:** Analyze 30+ Examples (from 30 subjects) of accelerometer signal for each activity type (walking, jogging, sitting, ascending stairs) => generate rules (classifier) to classify future activities



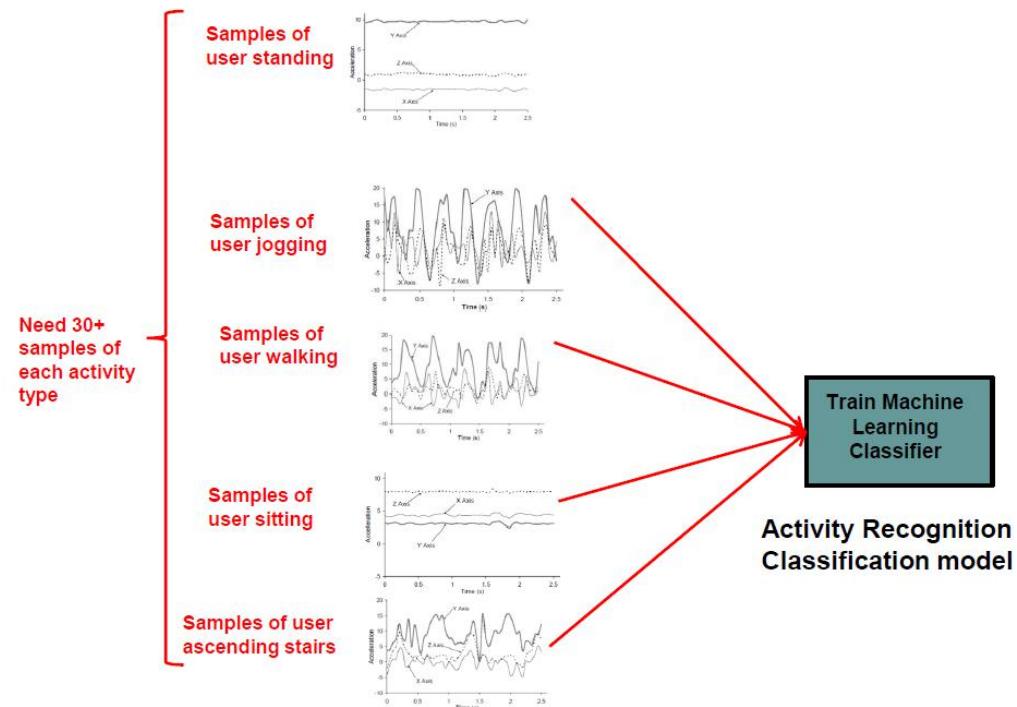
# **Training a Classifier: Steps**

# Steps for Training a Classifier

1. Gather data samples + label them
2. Import accelerometer samples into classification library (e.g. python, MATLAB)
3. Pre-processing (segmentation, smoothing, etc)
4. Extract features
5. Train classifier
6. Export classification model as JAR file
7. Import into Android app

# Step 1: Gather Sample data + Label them

- Need many samples of accelerometer data corresponding to each activity type (jogging, walking, sitting, ascending stairs, etc)



# Step 1: Gather Sample data + Label them

- Conduct a study to gather sample accelerometer data for each activity class
  - Recruit 30+ subjects
  - Run program that gathers accelerometer sensor data on subject's phone
  - Each subject:
    - Perform each activity (walking, jogging, sitting, etc)
    - Collect accelerometer data while they perform each activity (walking, jogging, sitting, etc)
  - Label data. i.e. tag each accelerometer sample with the corresponding activity
- Now have 30+ examples of each activity

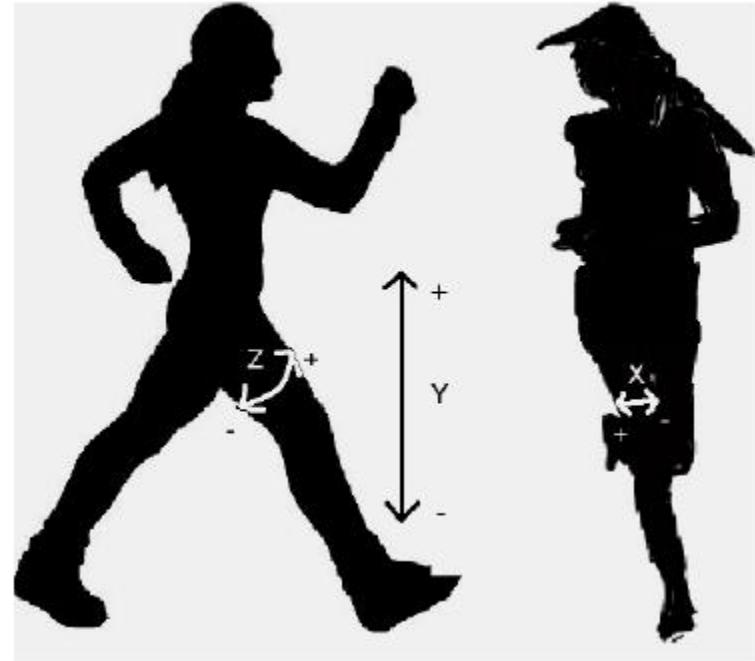
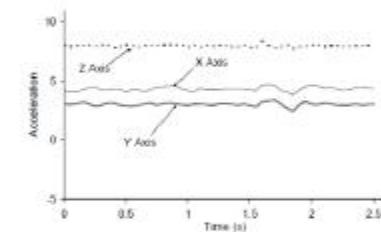
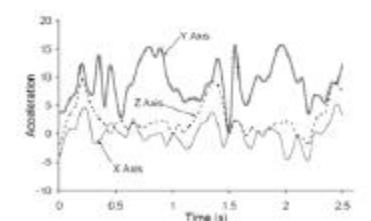


Figure 1: Axes of Motion Relative to User

30+  
Samples of  
user sitting



30+ Samples of  
user ascending  
stairs



# Step 1: Gather Sample data + Label them

## Program to Gather Accelerometer Data

- **Option 1:** Can write sensor program app that gathers accelerometer data while user is doing each of 6 activities (1 at a time)

```
msensor = (mSensorManager)
getSystemService(Context.SENSOR_SERVICE)

...
Public void onSensorChanged(SensorEvent event){

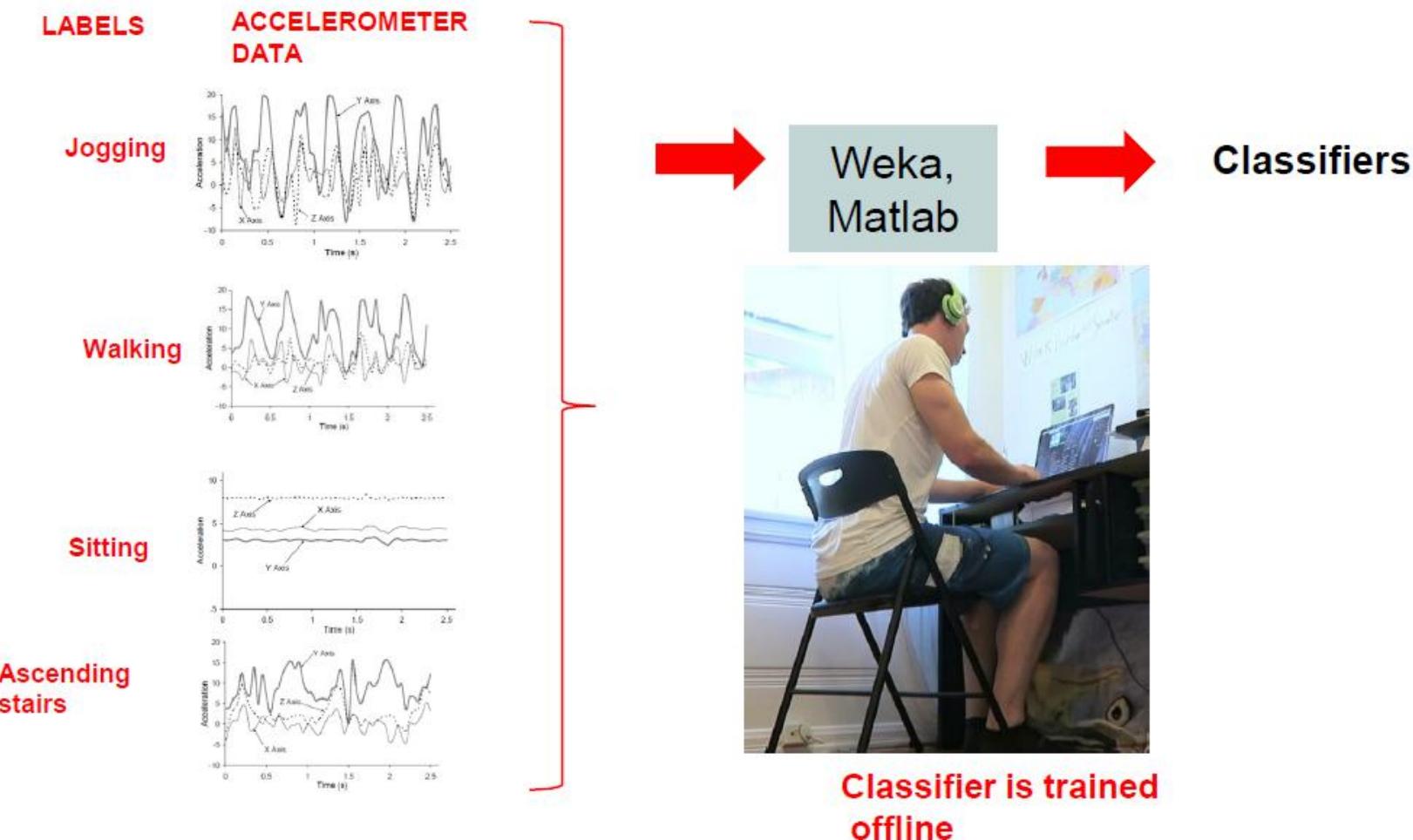
...
}
```

# **Step 1: Gather Sample data + Label them**

## **Program to Gather Accelerometer Data**

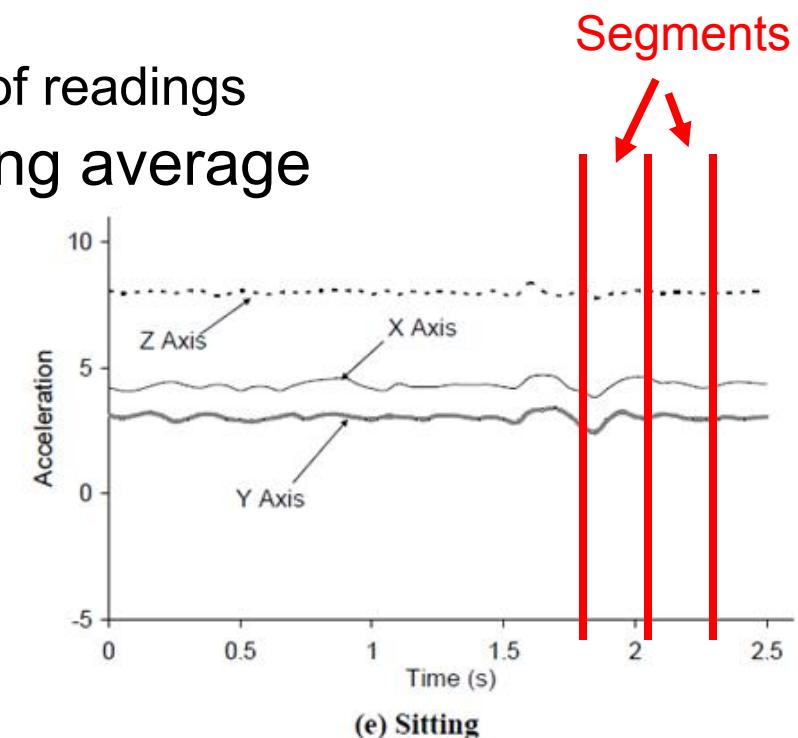
- **Option 2:** Use 3rdparty app to gather accelerometer
  - 2 popular ones: **Funf** and **AndroSensor**
  - Just download app,
  - Select sensors to log (e.g. accelerometer)
  - Continuously gathers sensor data in background

# Step 2: Import accelerometer samples into classification library (e.g. Python, MATLAB)



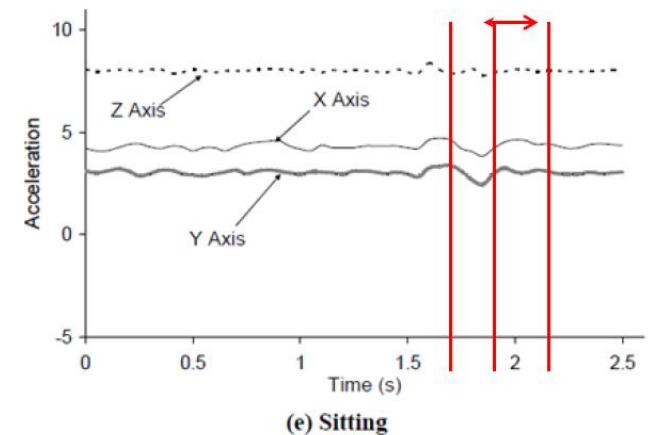
# Step 3: Pre-processing (segmentation, smoothing, etc) Segment Data (Windows)

- Pre-processing data may include segmentation, smoothing, etc
  - **Segment:** Divide data into smaller chunks. E.g. divide 60 seconds of raw time-series data into 5 second chunks
    - Note: 5 seconds of accelerometer data could be 100s of readings
  - **Smoothing:** Replace groups of values with moving average



# Step 4: Compute (Extract) Features

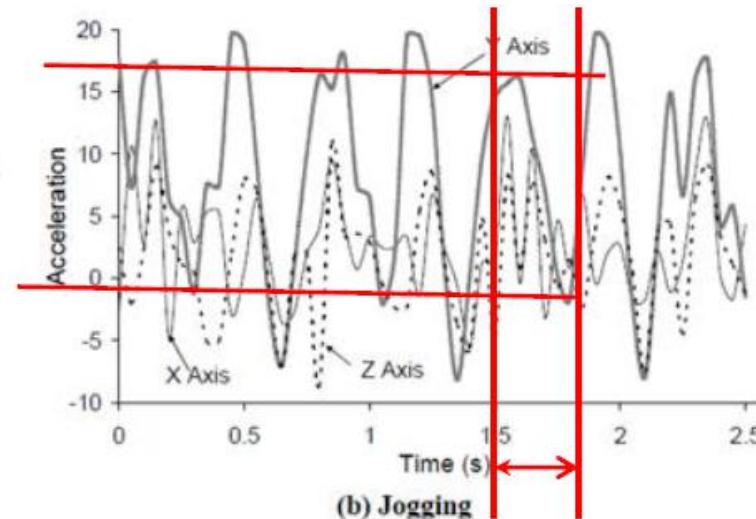
- For each 5-second segment (batch of accelerometer values) compute features
- **Features:** Formulas computed to quantify attributes of accelerometer data, captures accelerometer characteristics
- **Examples:** min-max of values within each segment, largest magnitude, standard deviation



# Step 4: Compute (Extract) Features

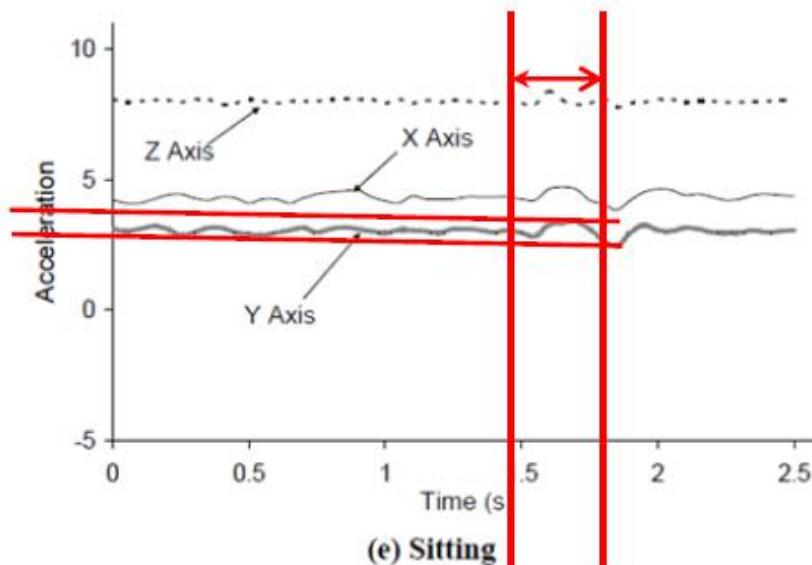
- **Important:** Ideally, values of features different for, distinguish each activity type (class)
- E.g: Min-max range feature

Large min-max  
for jogging



(b) Jogging

Small min-max  
for sitting



(e) Sitting

# Step 4: Compute (Extract) Features

Calculate  
many  
different  
features

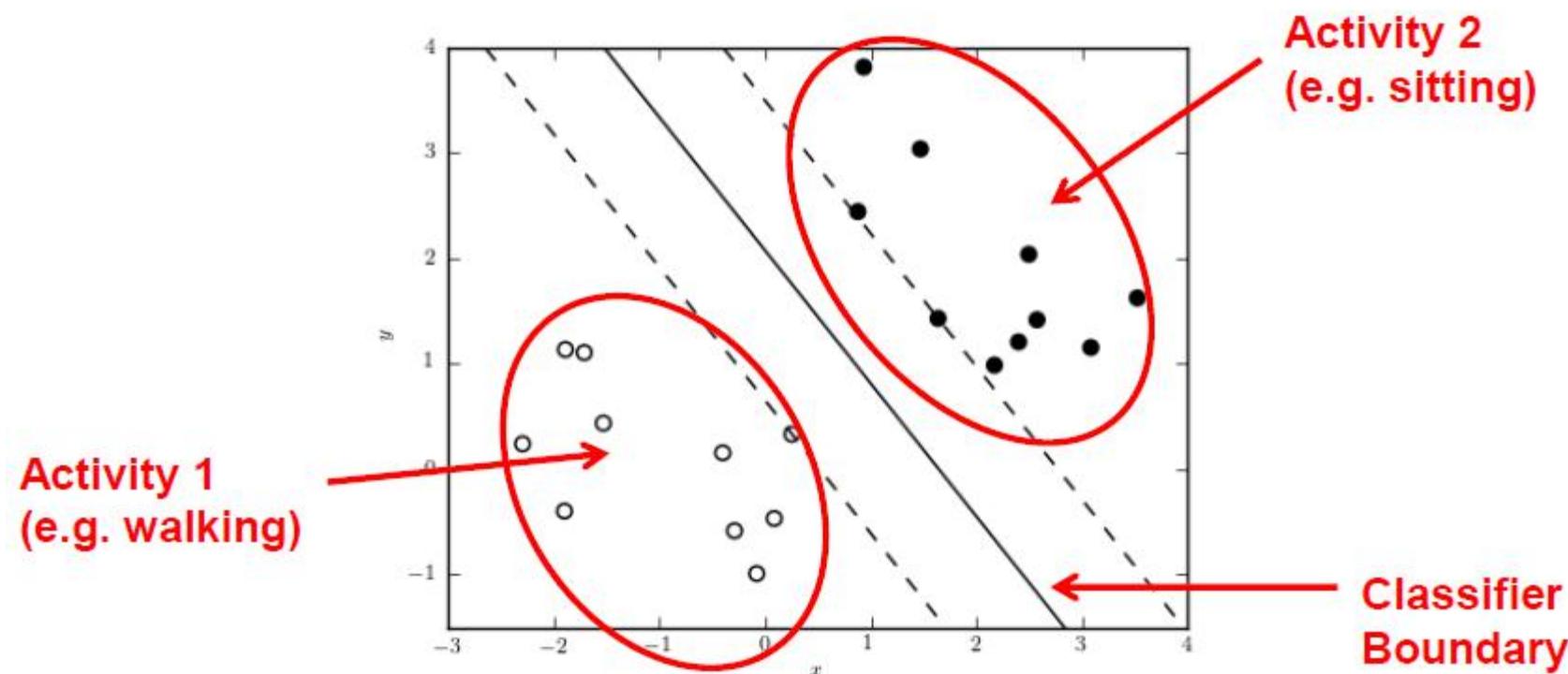
- Average[3]: Average acceleration (for each axis)
- Standard Deviation[3]: Standard deviation (for each axis)
- Average Absolute Difference[3]: Average absolute difference between the value of each of the 200 readings within the ED and the mean value over those 200 values (for each axis)
- Average Resultant Acceleration[1]: Average of the square roots of the sum of the values of each axis squared  $\sqrt{x_i^2 + y_i^2 + z_i^2}$  over the ED
- Time Between Peaks[3]: Time in milliseconds between peaks in the sinusoidal waves associated with most activities (for each axis)
- Binned Distribution[30]: We determine the range of values for each axis (maximum – minimum), divide this range into 10 equal sized bins, and then record what fraction of the 200 values fell within each of the bins.

# Step 5: Train classifier

- Features are just numbers (e.g. values of features for different subjects, activities)
- Different values for different activities
- **Training classifier:**figures out feature values corresponding to each activity
- Python, MATLAB already programmed with different classification algorithms (SVM, Naïve Bayes, Random Forest, J48, logistic regression, SMO, etc)
- Try different ones, compare accuracy

# Step 5: Train classifier

- SVM example



# Step 6: Export Classification model as JAR file

# Step 7: Import into Android app

- Export classification model (most accurate classifier type + data threshold values) as Java JAR file
- Import JAR file into Android app
- In app write Android code to
- Gather accelerometer data, segment, extract feature, classify using classifier in JAR file
- Classifies new accelerometer patterns while user is performing activity => Guess (infer) what activity

# CSE 162 Mobile Computing

## Lecture 12: Localization

Hua Huang

Department of Computer Science and Engineering

University of California, Merced

# An introduction to localization

# What is localization?

- Get the location of a mobile device
  - Some devices, e.g., cell phones, are a proxy of a person's location
- Get the location of wireless signal source
  - Wireless emitter
- Used to help derive the context and activity information
  - Location based services
  - Privacy problems

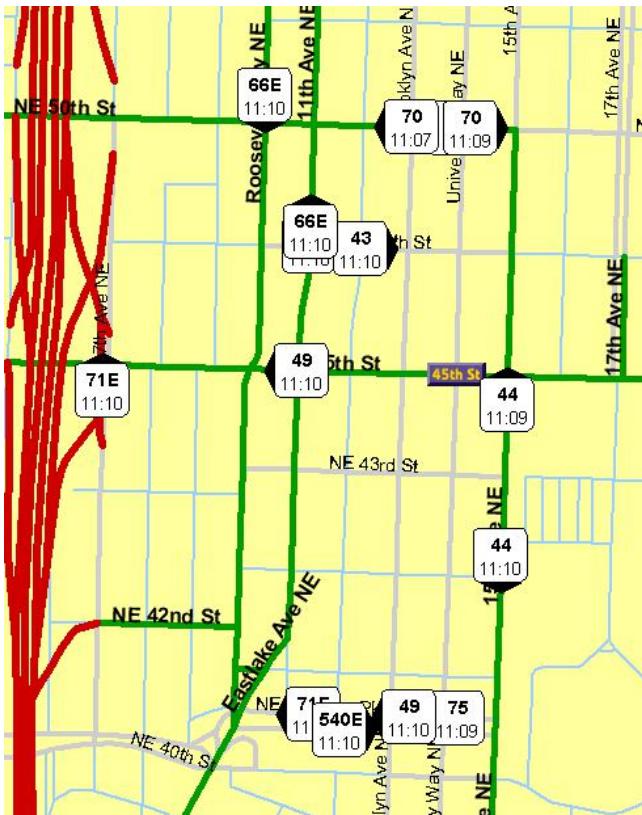
# Localization

- Well studied topic (3,000+ PhD theses??)
- Application dependent
- Research areas
  - Technology
  - Algorithms and data analysis
  - Evaluation

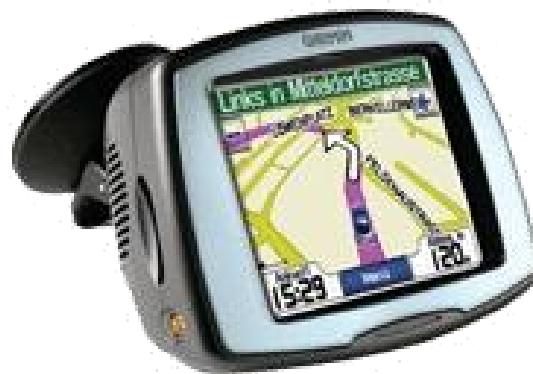
# Representing Location Information

- Absolute
  - Geographic coordinates (Lat: 33.98333, Long: -86.22444)
- Relative
  - 1 block north of the main building
- Symbolic
  - Home, road, bedroom, work

# Some outdoor applications



E-911



Car Navigation



Child tracking

# Some indoor applications



Elder care



Indoor navigation:  
mall, airport, museum, etc



Contact Tracing

# No one size fits all!

- Accurate
  - Low-cost
  - Easy-to-deploy
  - Ubiquitous
- 
- Application needs determine technology

# Lots of technologies!



GPS



WiFi Beacons



Ultrasound



Floor pressure



VHF Omni Ranging



Ad hoc signal strength



Laser range-finding



Stereo camera



Ultrasonic time of flight



Infrared proximity

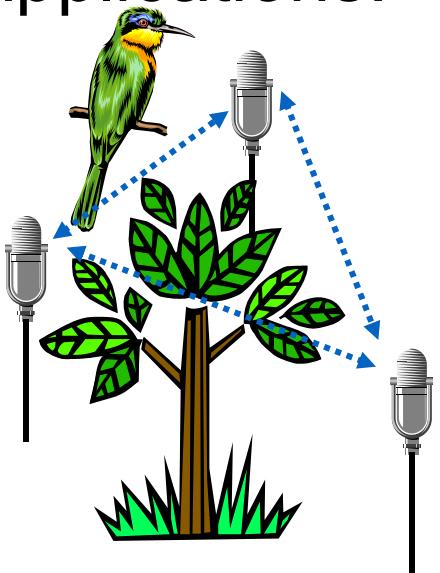


Array microphone

# Localization Applications Design Principles

# Variety of Applications

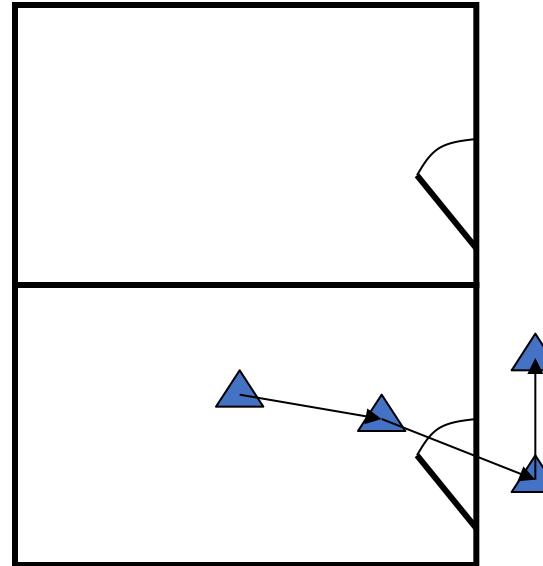
- Two applications:



## habitat monitoring:

Where is the bird?

What kind of bird is it?



## Asset tracking:

Where is the projector?

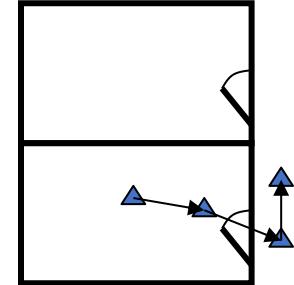
Why is it leaving the room?

# Variety of Application Requirements

■ Very different requirements!

- Outdoor operation
  - Weather problems
- Bird is not tagged
- Birdcall is characteristic but not exactly known
- Accurate enough to photograph bird
- Infrastructure:
  - Several acoustic sensors, with known relative locations; coordination with imaging systems

- Indoor operation
  - Multipath problems
- Assets are tagged
- Signals from asset tags can be engineered
- Accurate enough to track through building
- Infrastructure:
  - Room-granularity tag identification and localization; coordination with security infrastructure



# Multidimensional Requirement Space

- Granularity & Scale
- Accuracy & Precision
- Relative vs. Absolute Positioning
- Dynamic vs. Static (Mobile vs. Fixed)
- Cost & Form Factor
- Infrastructure & Installation Cost
- Environmental Sensitivity
- Cooperative or Passive Target

# Axes of Application Requirements

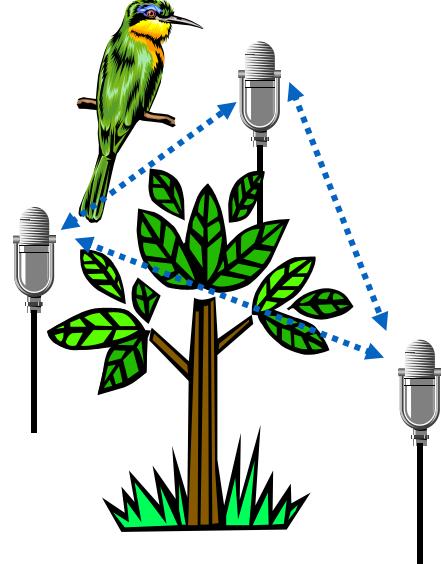
- Granularity and scale of measurements:
  - What is the smallest and largest measurable distance?
  - e.g. 50m (acoustics) vs. 25000km (GPS)
- Accuracy and precision:
  - How close is the answer to “ground truth” (accuracy)?
  - How consistent are the answers (precision)?
- Relation to established coordinate system:
  - GPS? Campus map? Building map?
- Dynamics:
  - Refresh rate? Motion estimation?

# Axes of Application Requirements

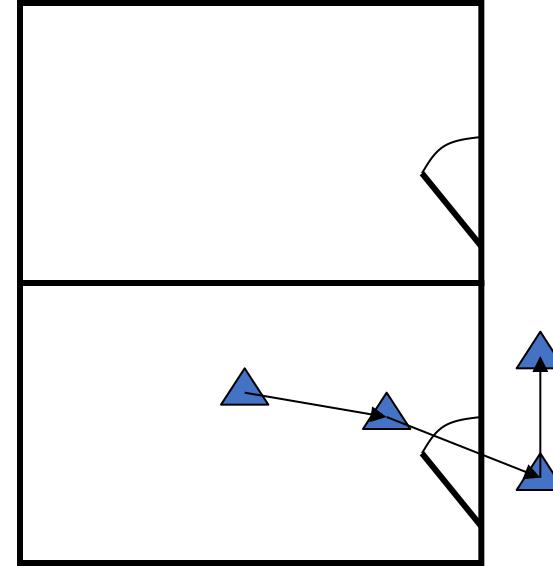
- Cost:
  - Node cost? Power? \$? Time?
  - Infrastructure cost? Installation cost?
- Form factor:
  - How big
  - Think about tracking tags on wild animals
- Communications Requirements:
  - Network topology: cluster head vs. local determination
  - What kind of coordination among nodes?
- Environment:
  - Indoor? Outdoor? On Mars?
- Is the target known? Is it cooperating?

# Returning to our two Applications...

- Choice of mechanisms differs:



**Passive habitat monitoring:**  
Minimize environ. interference  
No two bird songs are the same



**Asset tracking:**  
Controlled environment  
We know exactly what tag is like

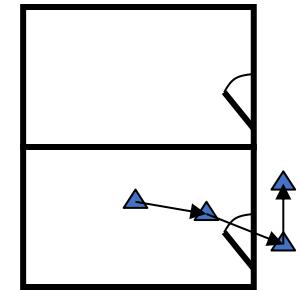
# Variety of Localization Mechanisms

## n Very different mechanisms indicated!

- Bird is not tagged
  - Passive detection of bird presence
- Birdcall is characteristic but not exactly known
- Bird does not have radio; Acoustic based ranging
- **Passive target localization**
  - Requires
    - Sophisticated detection
    - Coherent beamforming
    - Large data transfers



- Asset is tagged
  - Projector might know it had moved
- Signals from asset tag can be engineered
- Tag can use radio signal for ranging
- **Cooperative Localization**
  - Requires
    - Basic correlator
    - Simple triangulation
    - Minimal data transfers



# Wireless Technologies for Localization

Name	Effective Range	Pros	Cons
GSM	35km	Long range	Very low accuracy
LTE	30km-100km		
Wi-Fi	50m-100m	Readily available; Medium range	Low accuracy
Ultra Wideband	70m	High accuracy	High cost
Bluetooth	10m	Readily Available; Medium accuracy	Short range
Ultrasound	6-9m	High accuracy	High cost, not scalable
RFID & IR	1m	Moderate to high accuracy	Short range, Line-Of-Sight (LOS)
NFC	<4cm	High accuracy	Very short range

# Localization Algorithms

# Algorithms to obtain locations

- Range-based algorithms
- Range-free algorithms
- Fingerprinting

# Range Based Algorithms

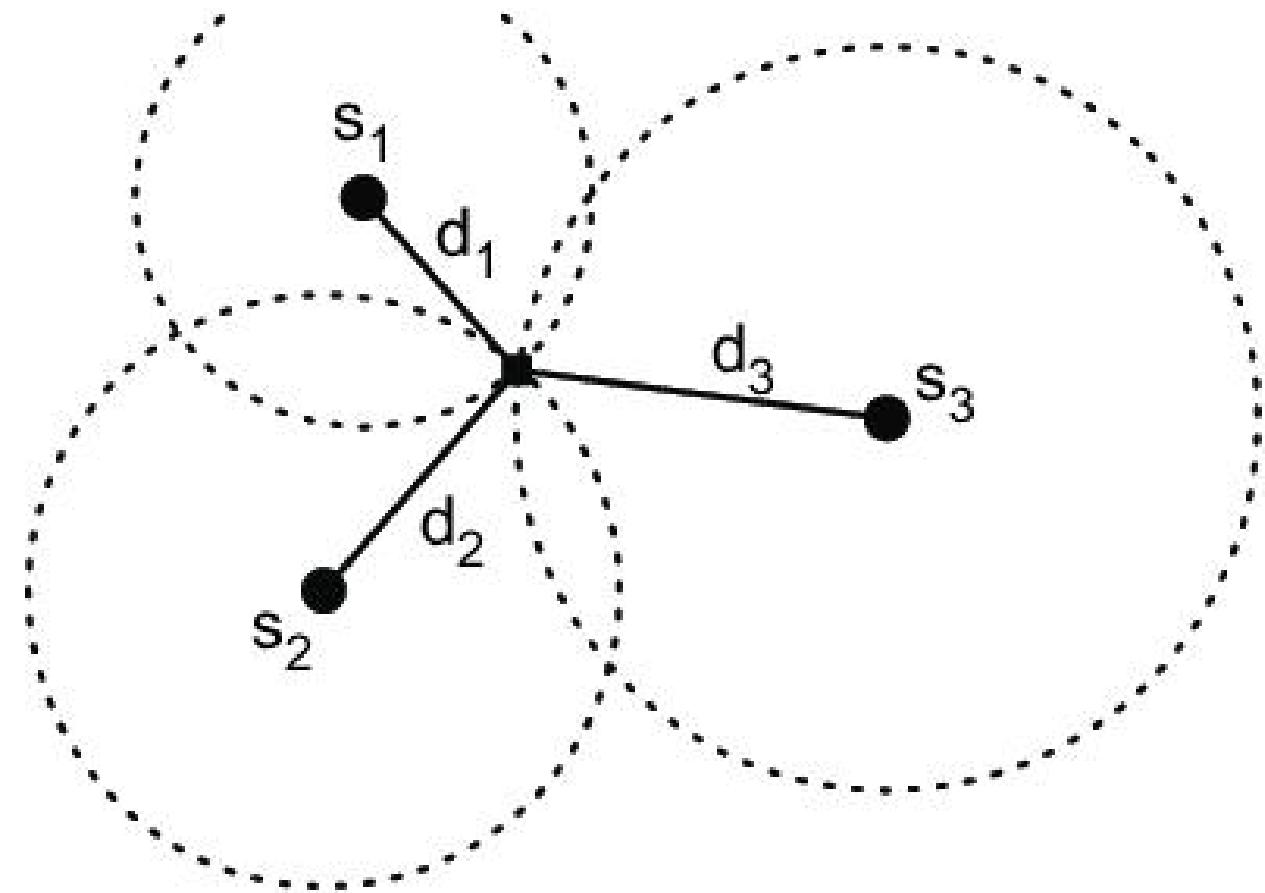
- Rely on the distance (angle) measurement between nodes to estimate the target location
- Approaches
  - Proximity
  - Lateration
  - Hyperbolic Lateration
  - Angulation
- Distance estimates
  - Time of Flight
  - Signal Strength Attenuation

# Approach: Proximity

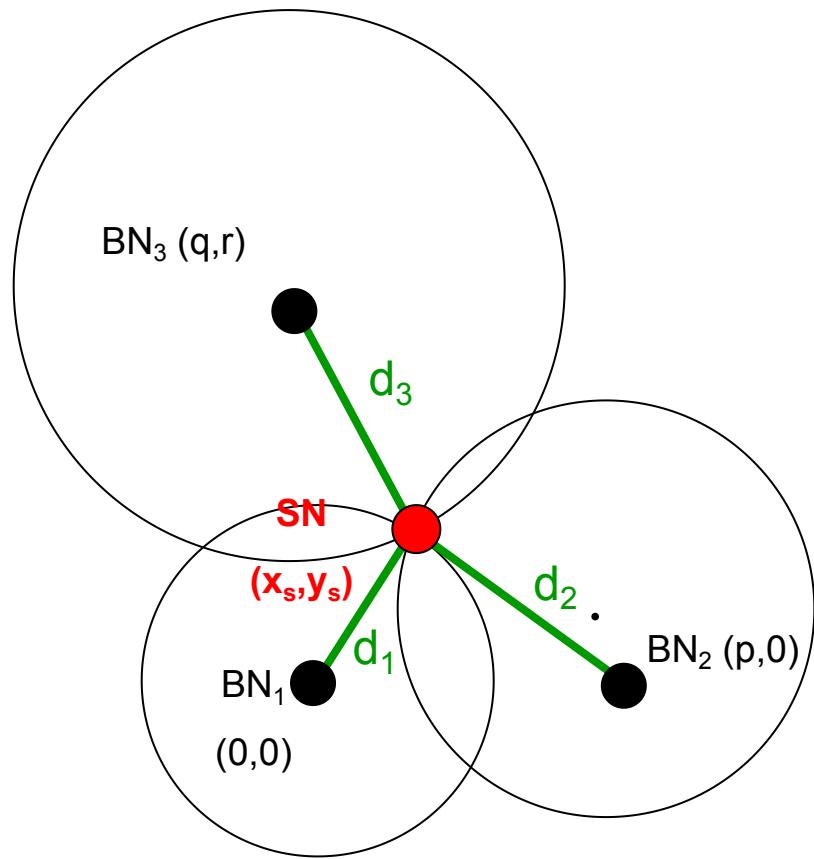
- Simplest positioning technique
- Closeness to a reference point
- Based on loudness, physical contact, etc
- Examples
  - RFID Door Access Control System

# Approach: Lateration

- Method: Measure distance between device and reference points
  - $s_1, s_2, s_3$  locations are known
  - Measure the distances  $d_1, d_2$ , and  $d_3$
  - Search for the most-likely location given the distances
- 3 reference points needed for 2D and 4 for 3D



- Assuming accurate distance measurements between nodes, apply the trilateration technique to determine the SN coordinates (unknown) using the three BNs coordinates and the r distances.
- Without loss of generality, let the coordinates of the references be  $(0,0), (0,p), (q,r)$



By definition:

$$d_1^2 = x^2 + y^2$$

$$d_2^2 = (x - p)^2 + y^2$$

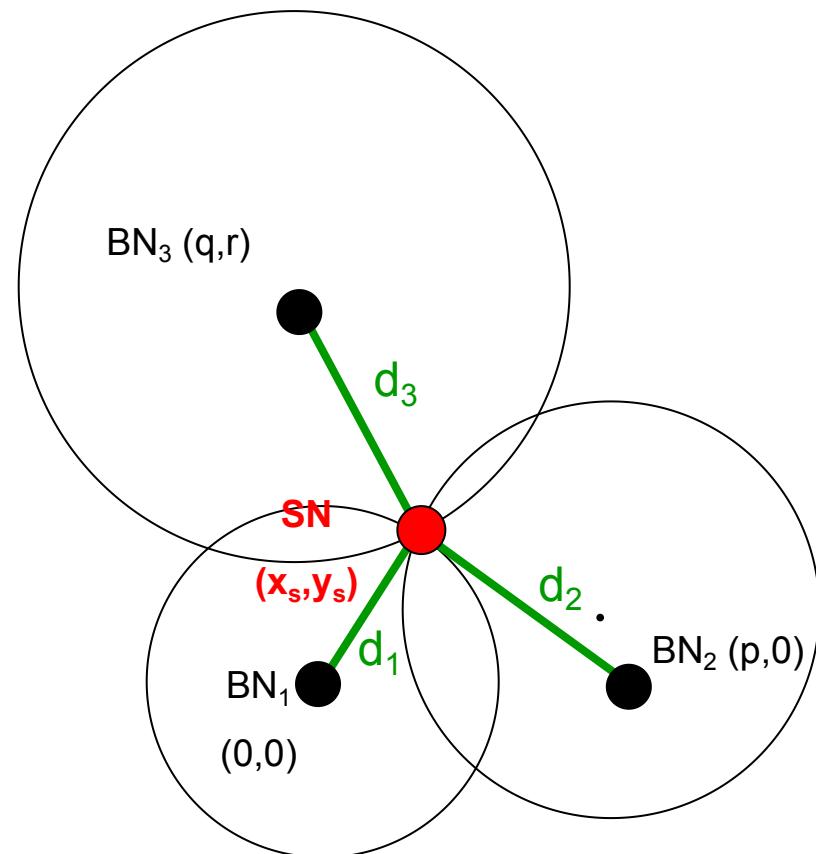
$$d_3^2 = (x - q)^2 + (y - r)^2$$

- By subtracting the second equation from the first,  $x$  is attained.

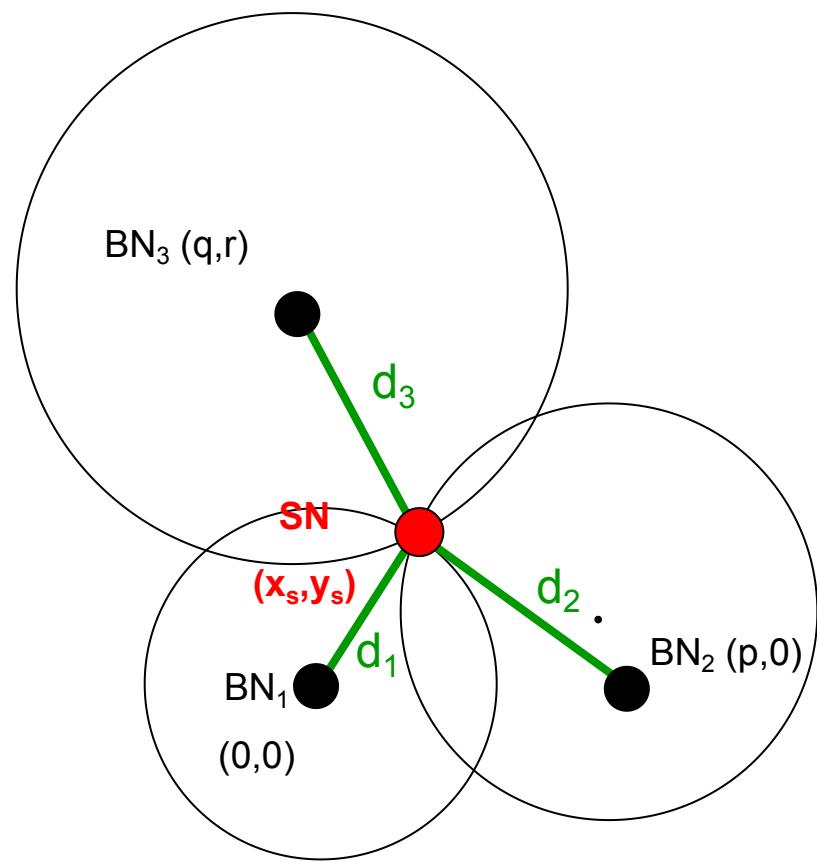
$$x = \frac{d_1^2 - d_2^2 + p^2}{2p}$$

- Substituting this value back into the first equation will result in values for  $y$ .

$$y = \pm \sqrt{d_1^2 - \left( \frac{d_1^2 - d_2^2 + p^2}{2p} \right)^2}$$

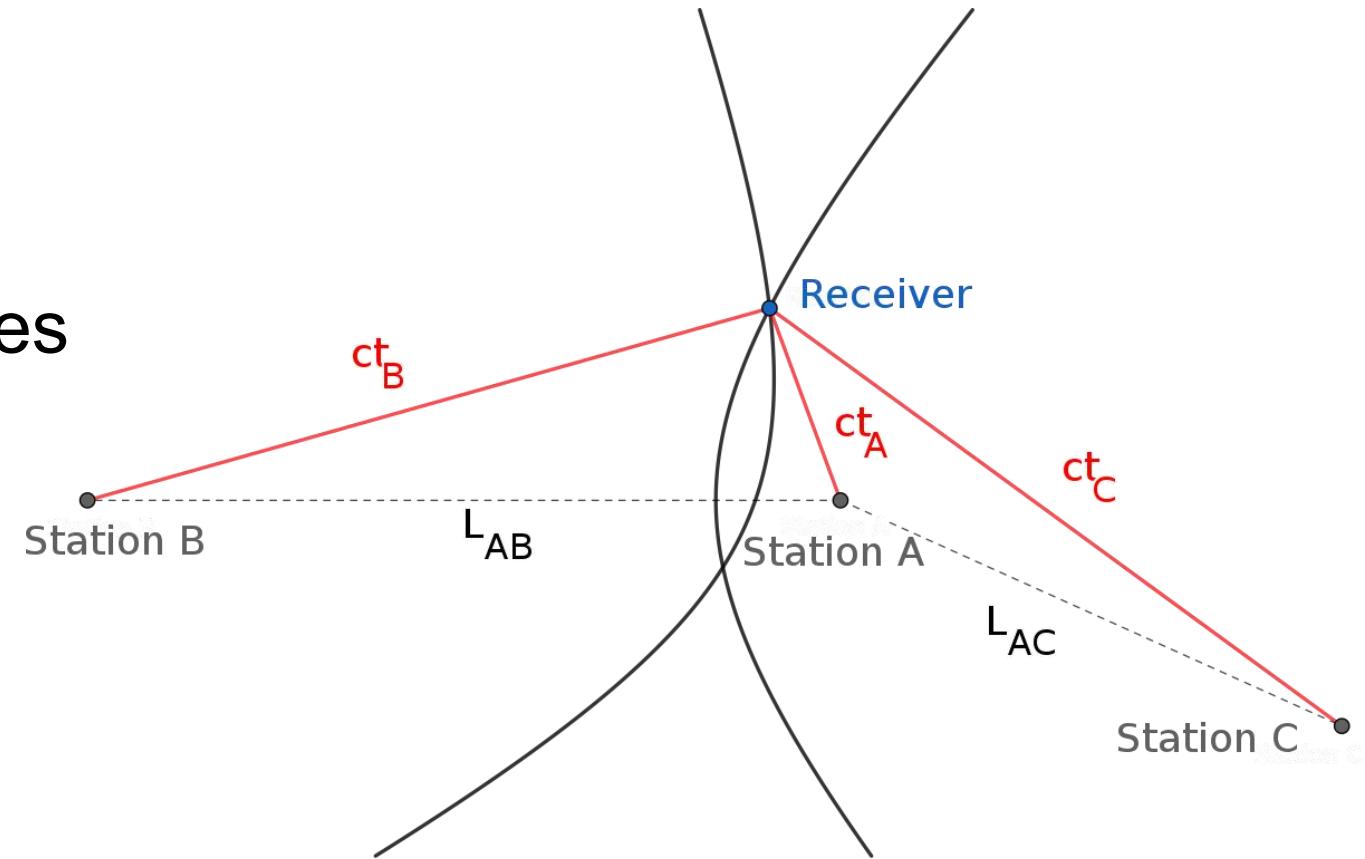


To determine which value of  $y$  correctly describes the point in question,  $x$  and  $y$  are substituted into the third equation.



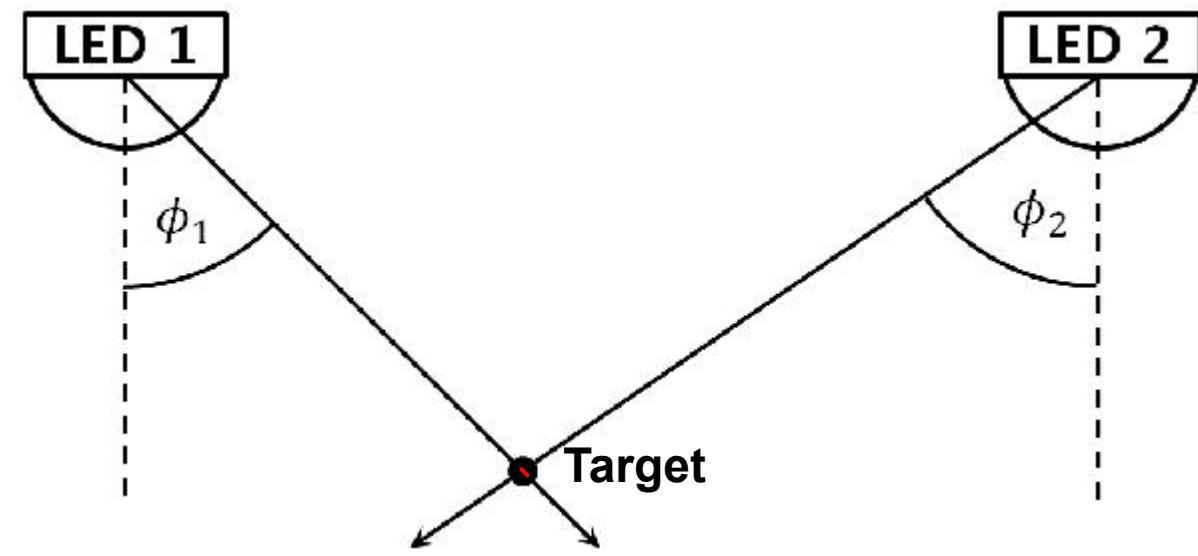
# Approach: Hyperbolic Lateration

- Known: Station A, B, C locations
- Unknown: The signal traveling times to the receiver:  $ct_A, ct_B, ct_C$
- Known:  $ct_A - ct_B, ct_A - ct_C, ct_B - ct_C$
- Each time difference locates the receiver on a branch of a hyperbola



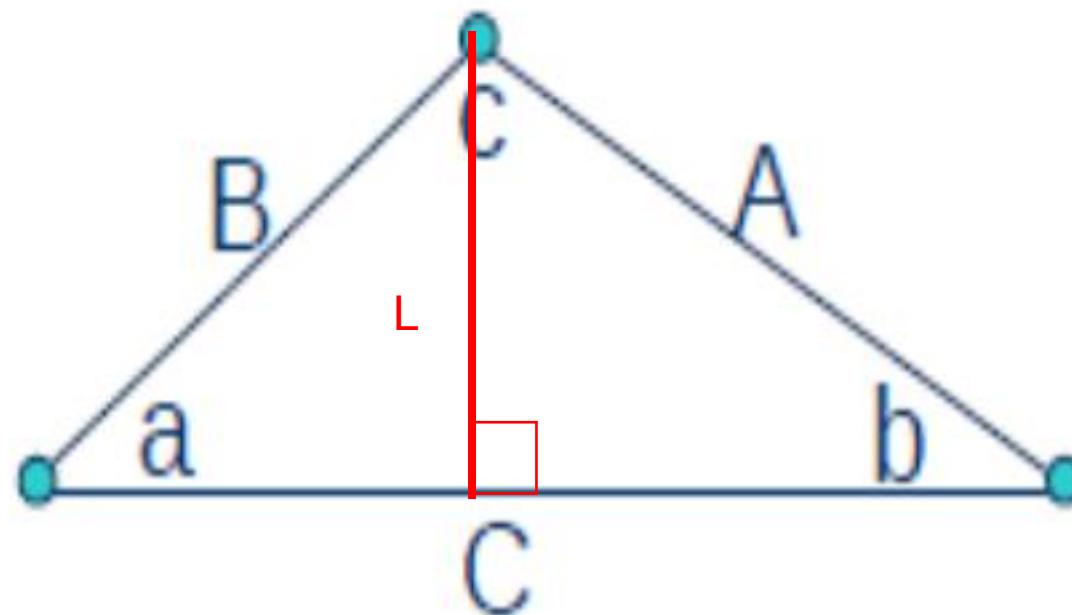
# Approach: Triangulation

- Use the location of the signal sources and the relative angles
- Uniquely identify the location of the target
- How to do it mathematically?



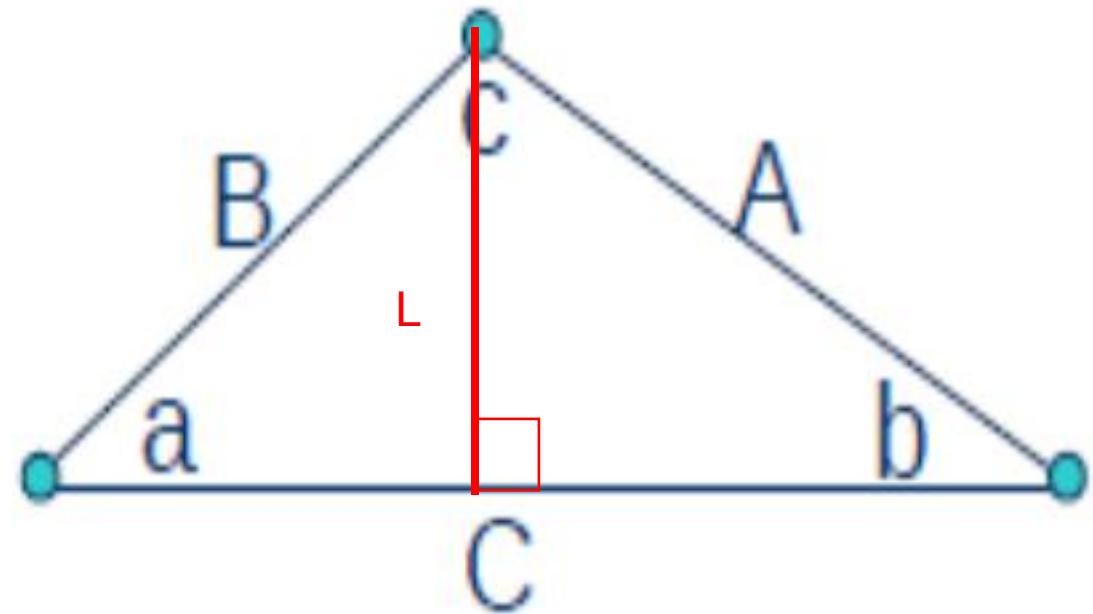
# Question

- Angles  $a$  and  $b$  are known. Distance  $C$  is known.
- What is  $L$ ?



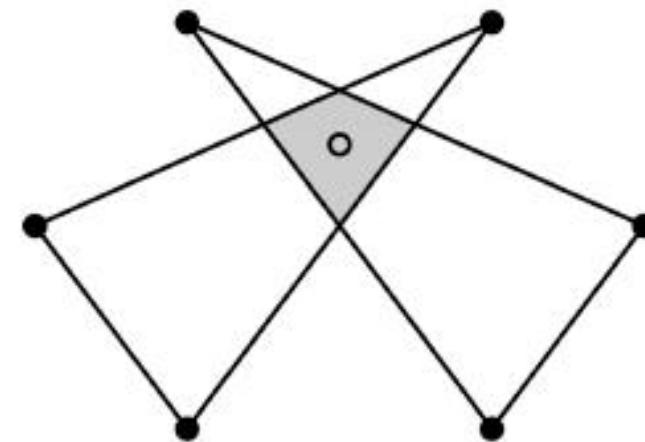
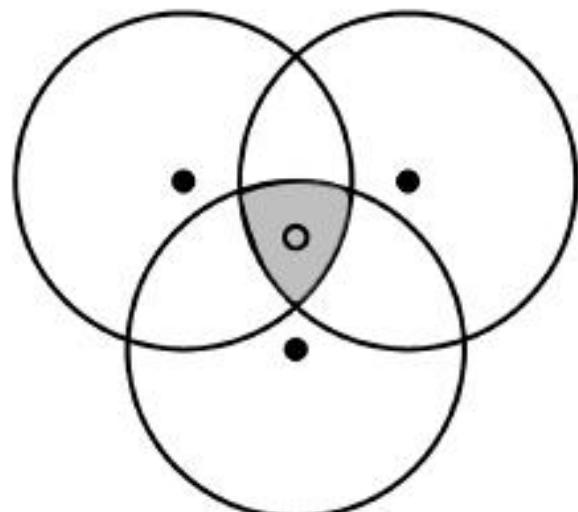
# Question

- $L/\tan(a) + L/\tan(b) = C$
- $L = C/(1/\tan(a) + 1/\tan(b))$



# Range Free Algorithms

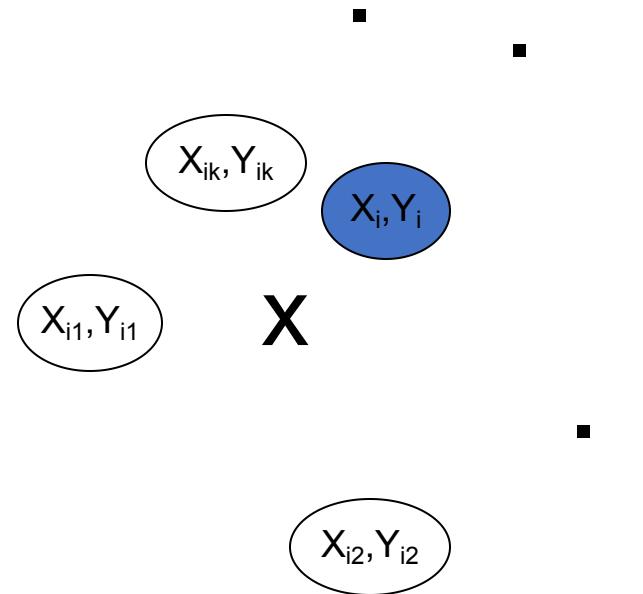
- Rely on target object's proximity to anchors with known positions
  - Neighborhood: single/multiple closest BS
  - Area estimation:



# Range-free Localization Techniques

## Centroid Algorithm

- Nodes localize themselves to the centroid of their proximate reference points



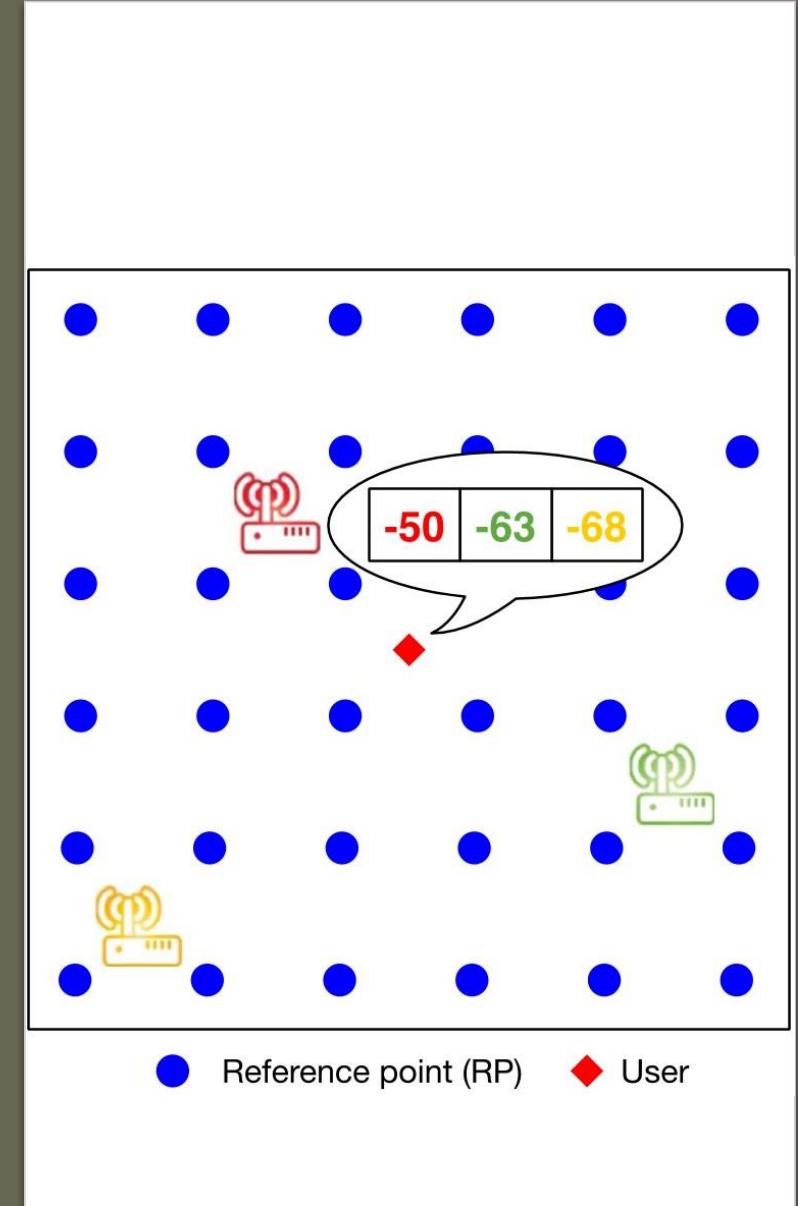
$$(X_{est}, Y_{est}) = \left( \frac{X_{i1} + \dots + X_{ik}}{k}, \frac{Y_{i1} + \dots + Y_{ik}}{k} \right)$$

# Fingerprinting

- Mapping solution
- Address problems with multipath
- Better than modeling complex RF propagation pattern

# Fingerprinting: Steps

- Step1
  - Use war-driving to build up location fingerprints (i.e. location coordinates + respective RSSI from nearby base stations)
- Step2
  - Match online measurements with the closest *a priori* location fingerprints



# Fingerprinting: Example

<b>SSID (Name)</b>	<b>BSSID (MAC address)</b>	<b>Signal Strength (RSSI)</b>
linksys	00:0F:66:2A:61:00	18
starbucks	00:0F:C8:00:15:13	15
newark wifi	00:06:25:98:7A:0C	23

# Fingerprinting: Pros and Cons

- Pros
  - Physical model not required
- Cons
  - Requires a dense site survey
- Prerequisite:
  - Spatial differentiability
  - Temporal stability

# Concluding Remarks

## Pros and Cons

- Two main types of distributed localization algorithms:
  - Range-based
    - Estimating the coordinates based on the collected information of distances or angles among nodes
    - Merit: Relatively high accuracy
    - Drawback: Costly (Hardware, Power consumption)
  - Range-free
    - Estimating the coordinates based on the connectivity relations
    - Merit: Cost-effective
    - Drawback: Not as accurate (But: coarse accuracy is sufficient for some applications)

**Hardware/Energy Cost vs Location Precision**

# Summary of Localization Algorithms

	Measurement Scheme	Accuracy	Special Requirement
Range-based	TOA	Moderate	Synchronization, dense beacons
	TDOA	High	Synchronization, LOS, dense beacons
	AOA	High	Directional antenna
	RSSI	Moderate	No
Range-free	Neighborhood	Low	No
	Area estimation	Moderate	Dense Beacons
Fingerprinting	RSSI	High	No

# CSE 162 Mobile Computing

## Lecture 13: Distance Estimation Technologies, GPS

Hua Huang

Department of Computer Science and Engineering  
University of California, Merced

# Distance Estimation Technologies

# Distance Estimation

- Multiply the radio signal velocity and the travel time
  - Time of arrival (TOA)
  - Time difference of arrival (TDOA)
- Compute the attenuation of the emitted signal strength
  - RSSI
- Problem: Multipath fading

# Distance Estimation: TOA

- Distance
  - Based on one signal's travelling time from target to measuring unit
  - $d = v_{\text{radio}} * t_{\text{radio}}$
- Requirement
  - Transmitters and receivers should be precisely synchronized
  - Timestamp must be labeled in the transmitting signal

# Distance Estimation: TDOA

- The transmitter sends a radio and a sound
- Receiver measurements: receiving time  $t_1$  and  $t_2$
- Known parameter:  $v_r$  and  $v_s$
- How to estimate the distance?

# Distance Estimation: TDOA

Let the transmission time be denoted  $t_0$ . We have

$$d = (t_1 - t_0) * v_r$$

$$d = (t_2 - t_0) * v_s$$

Therefore :  $\frac{d}{v_s} - \frac{d}{v_r} = t_2 - t_1$

Finally we get:  $d = (t_2 - t_1) * v_r * v_s / (v_r - v_s)$

# Distance Estimation: RSSI

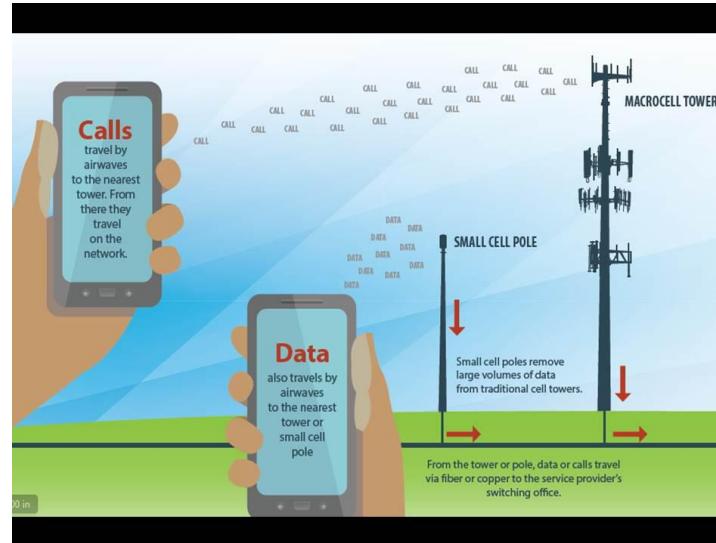
- Distance
  - Based on radio propagation model

$$P(d) = P(d_0) - \eta 10 \log \left( \frac{d}{d_0} \right) + X_\sigma$$

- Requirement
  - Path loss exponent  $\eta$  for a given environment is known

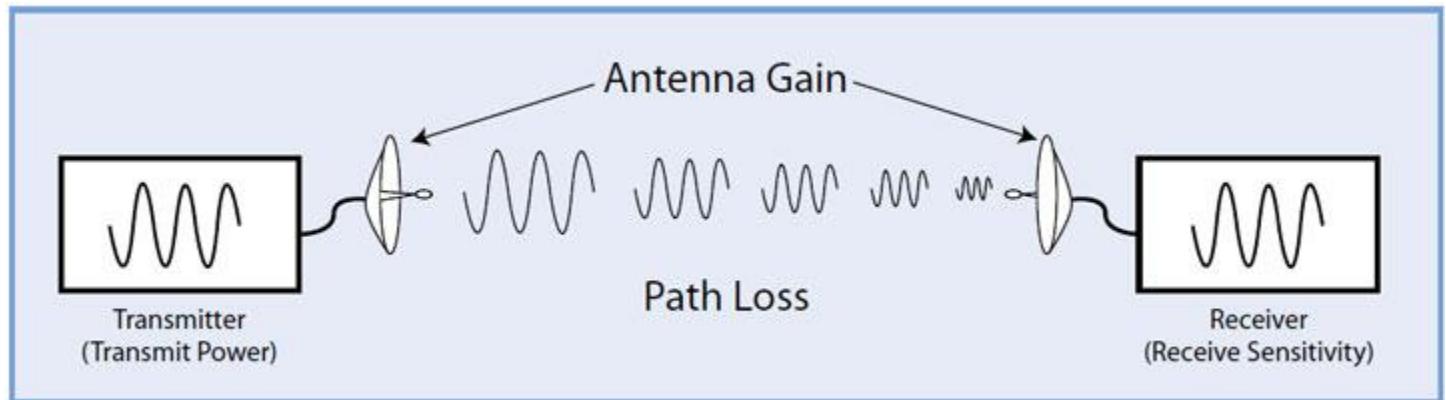
# RSSI based distance estimation

- Signal strength attenuates as distance increases

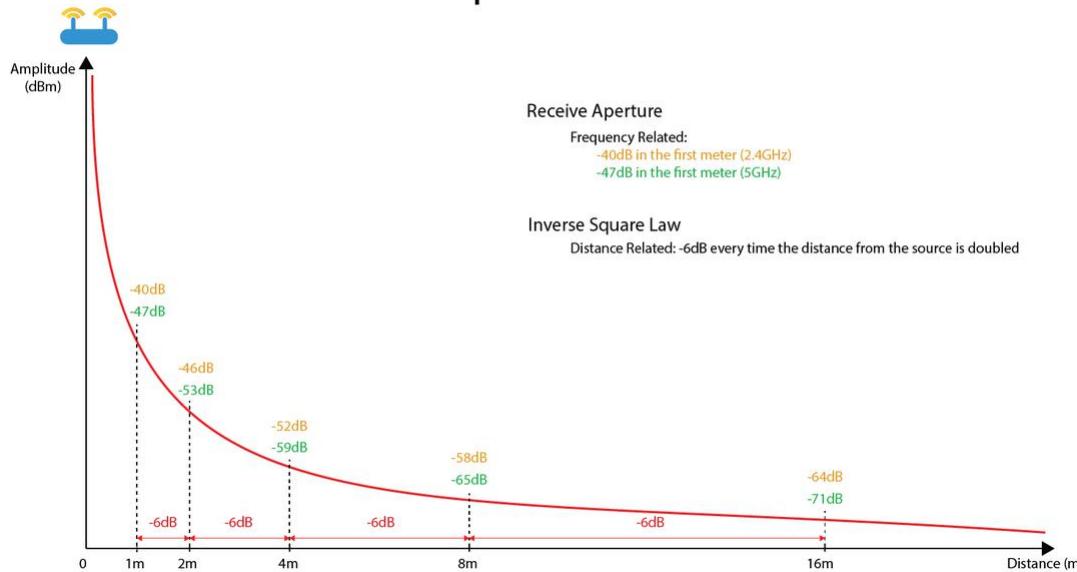


# RSSI based distance estimation

- Theoretical model

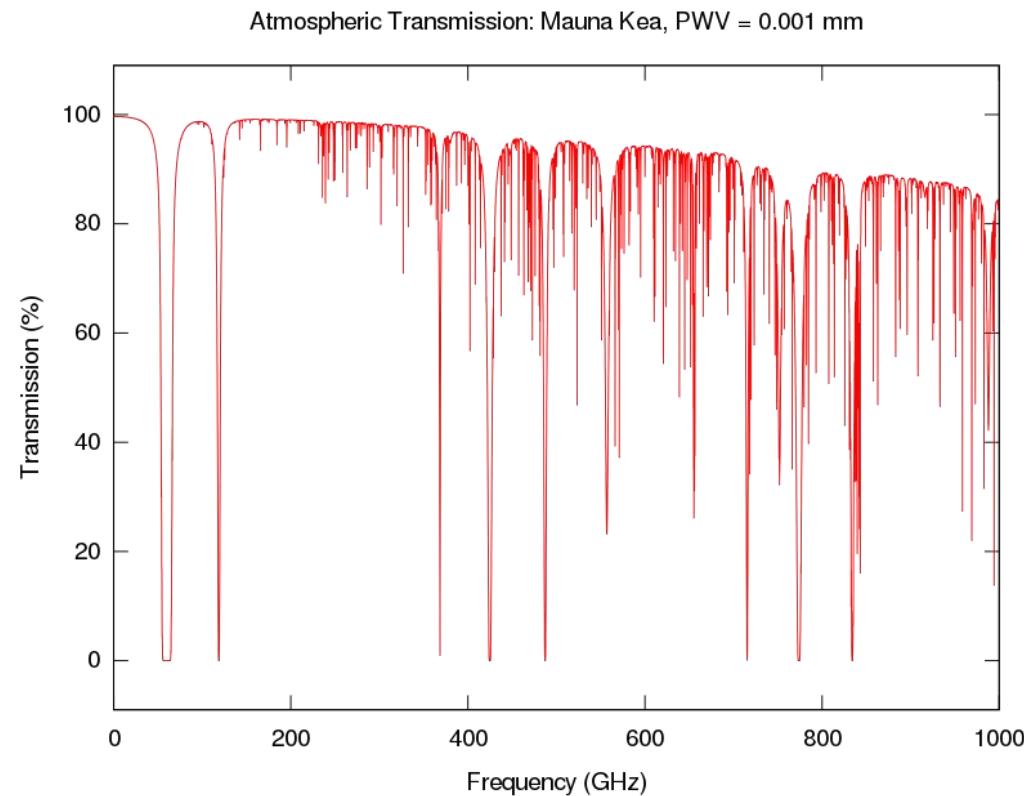


Free Space Path Loss

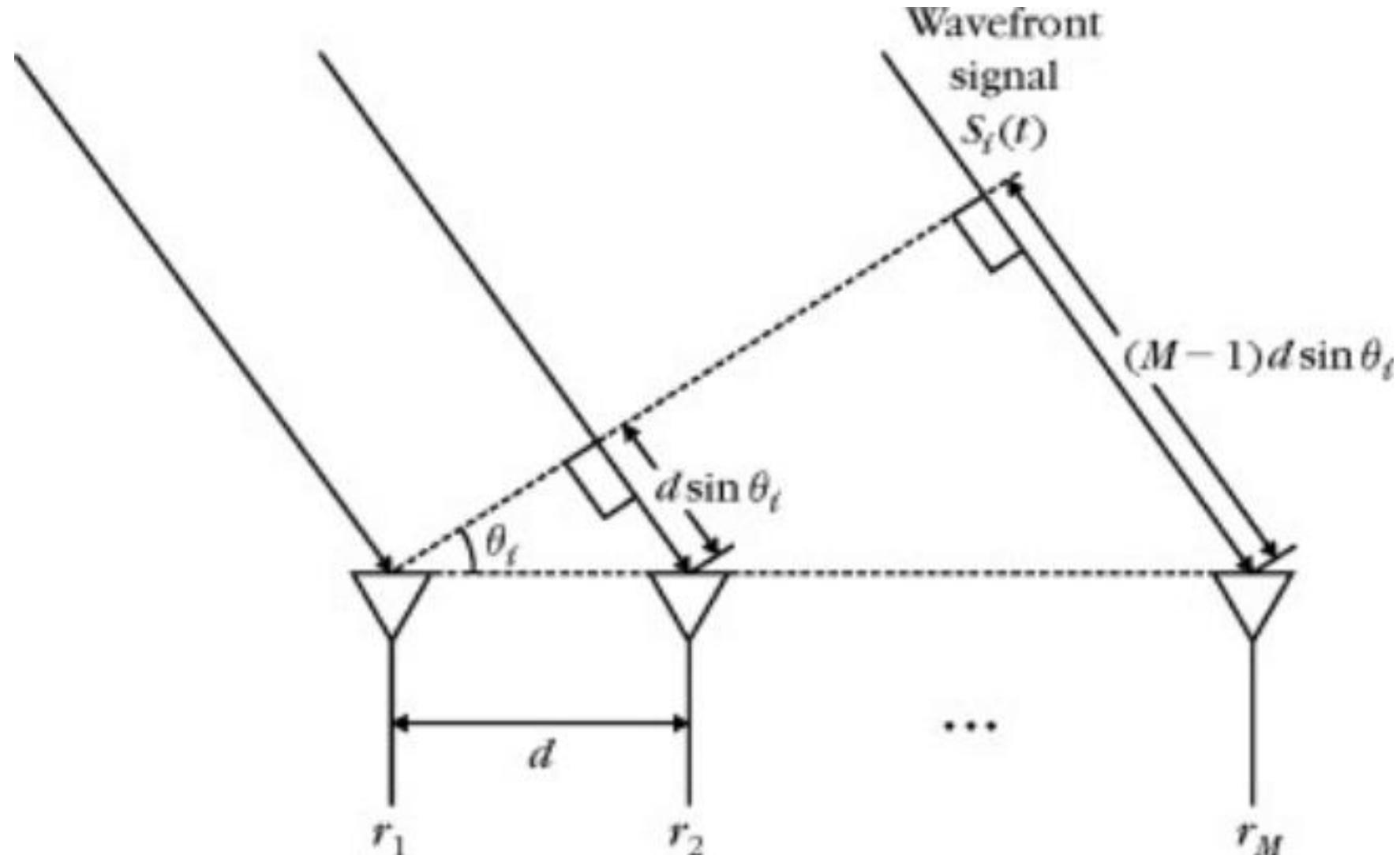


# Limitation: Actual rssi is very noisy

- The RSSI and distance are correlated, but can significantly fluctuate



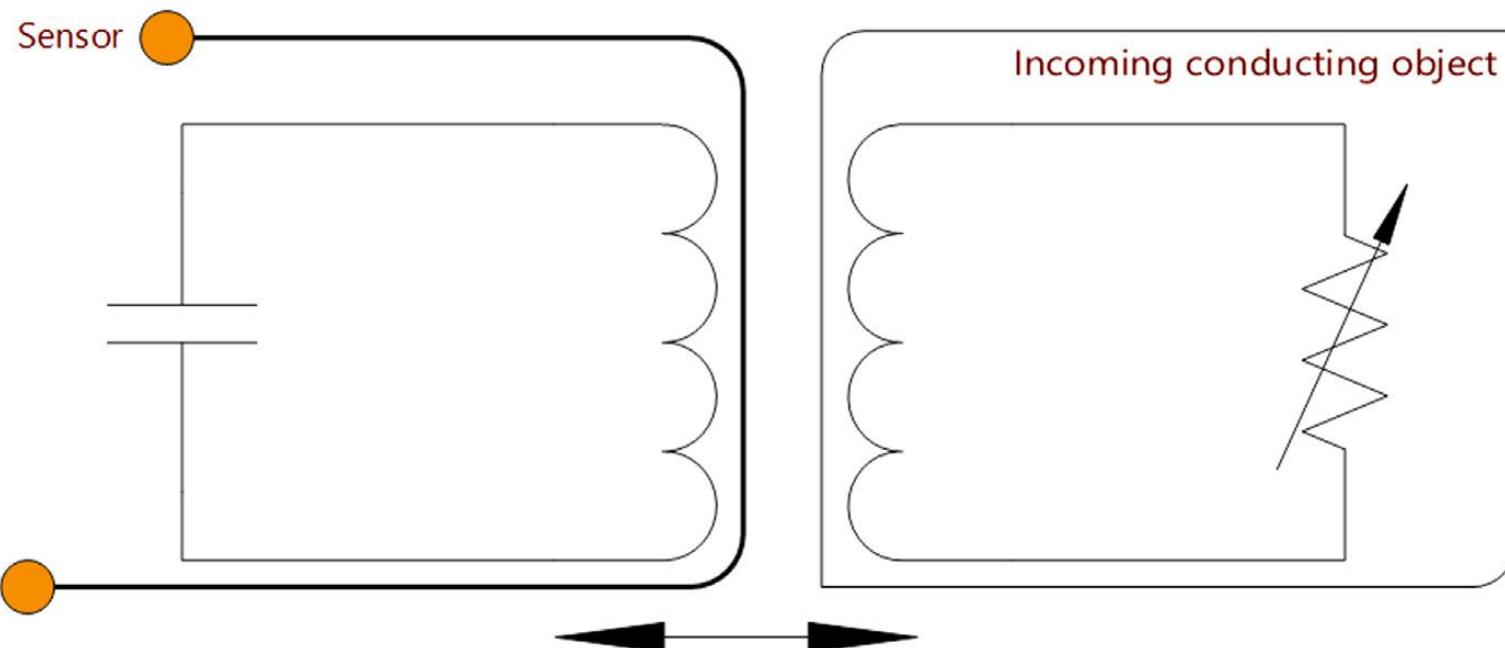
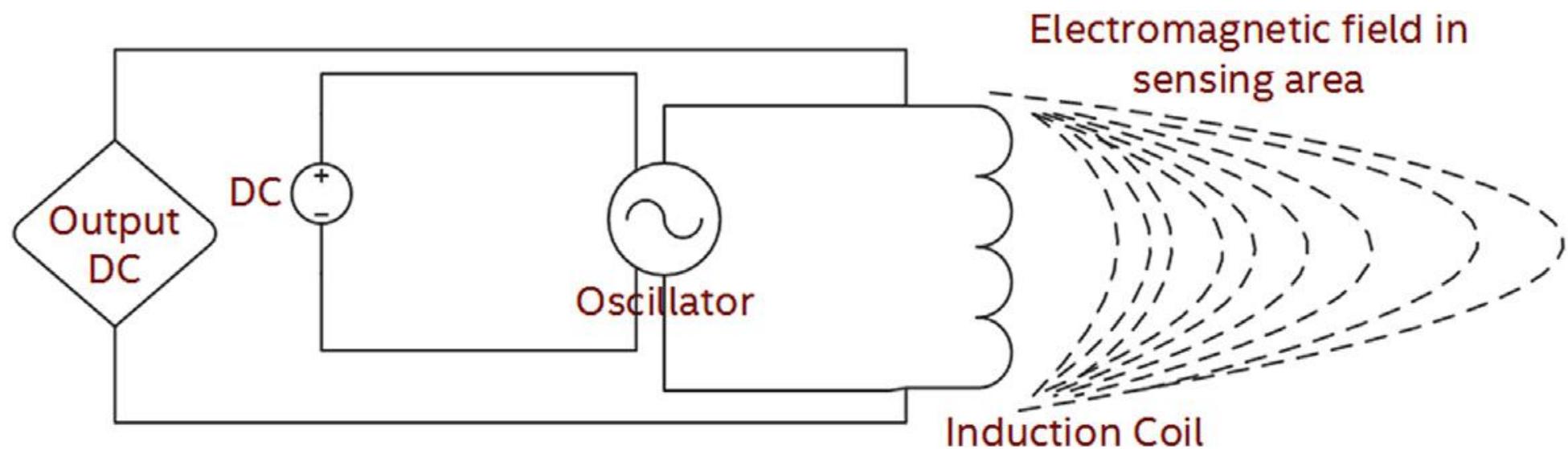
# Estimating Angle of Arrival: beamforming



# Distance Sensors

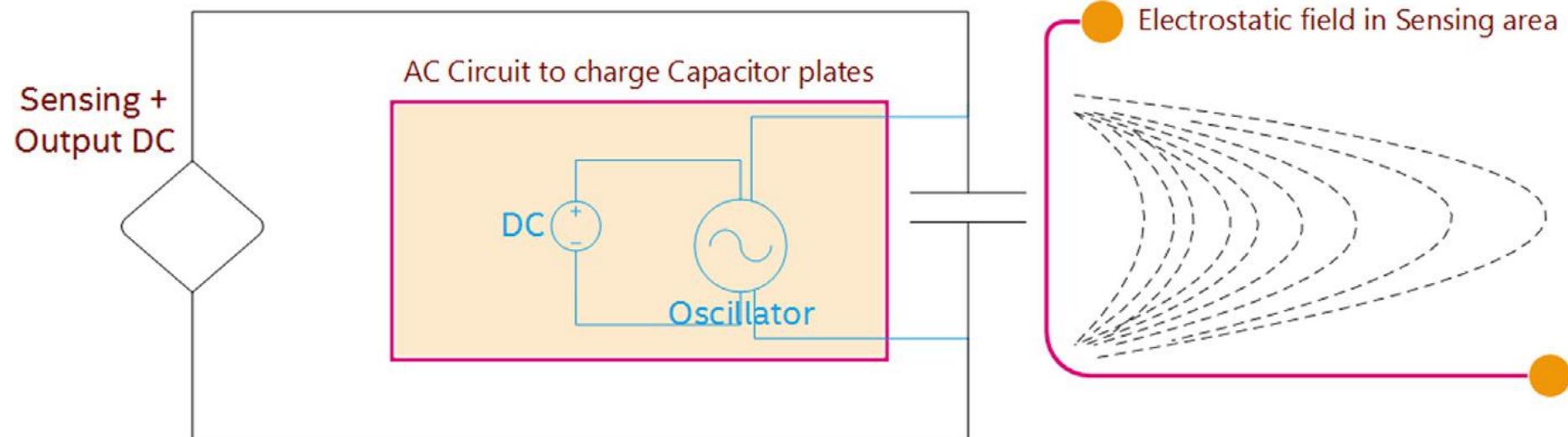
# Inductive Sensor

- An inductive proximity sensor mainly consists of a coil, an electronic oscillator, a detection circuit, an output circuit, and an energy source to provide electrical stimulation.
- Working Principle:
  - oscillator generates a changing alternating current
  - the induction coil, it generates a changing electromagnetic field
  - metal object generates a change in impedance due to eddy currents, changing the detection circuit measurements



# Capacitive Proximity Sensor

- Use electrostatic field, instead of magnetic field, for detection.
  - Thus, non metallic objects can be sensed.



# Distance and Ranging: Infrared Sensors

- Contain an infrared emitter, and an infrared detector
- Works by emitting a certain amount of infrared light, and seeing how much it gets back
- Why infrared?
  - There are not many other infrared sources in everyday life that would interfere with this sensor
  - If visible light were used, light bulbs, computer screens, cellphone screens, etc, would all interfere with the depth reading



# Distance and Ranging: Infrared Sensors

- Great at measuring shorter distances (2" –30")
- Where do you see these?
  - Touchless Switches (toilets, faucets, etc)
  - Roomba vacuums
  - Kinect
- Related: Passive Infrared (PIR) Sensors
  - No IR emitter, just detects ambient IR.
  - Detects some normal state (like a wall's IR emissions) and when something moves in front, it detects a change
  - Great for detecting motion (motion sensors for security systems)



# Distance and Ranging: Speed detector

- e.g. police radar guns
  - Microwave radars use the Doppler effect (the return echo from a moving object will be frequency shifted).
  - The greater the target speed, the greater the frequency (Doppler) shift



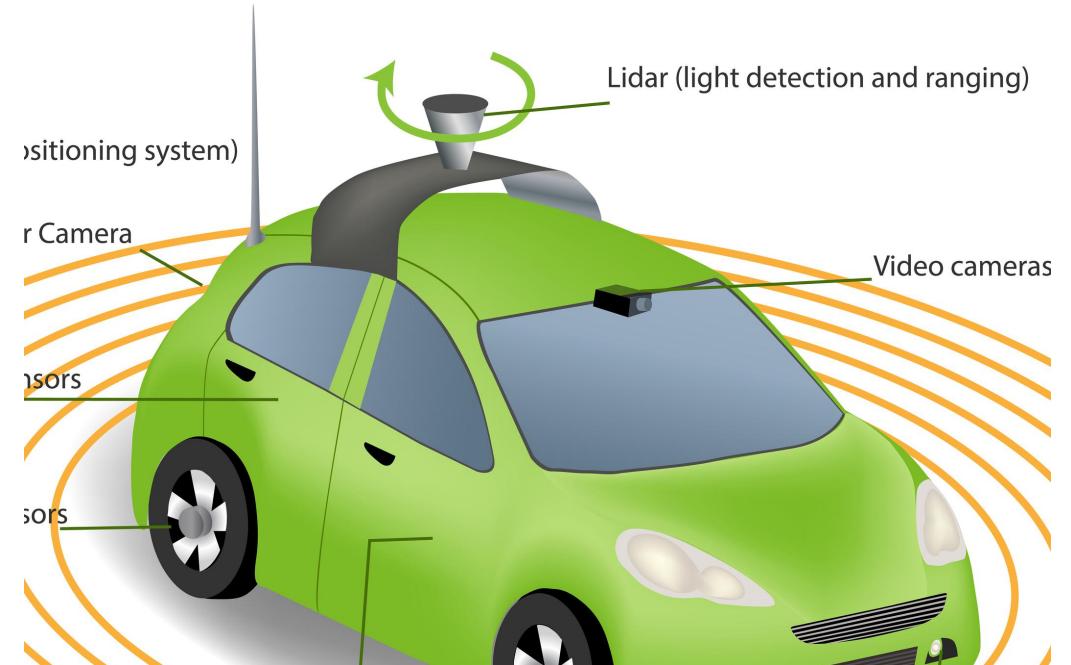
# Ultrasonic Sensors

- Contain a high frequency speaker , and a microphone
- Works just like a sonar, emitting a sound, and listening for the echo to determine range
- Why is it called ultrasonic?
  - Very high frequency sound, it is barely at the edge of what humans can hear.
  - This is nice since it is not as annoying to use
- Pros: More accurate than IR sensors at slightly longer distance (typically up to several feet)
- Cons: Almost twice the price



# Lidar

- LIDAR — Light Detection and Ranging
- How it works?
  - send pulses of light, and determine the difference in reflection time between consecutive pulses to determine speed
- Application:
  - automobile
  - Indoor mapping robot



# CSE 162 Mobile Computing

## Lecture 14: GPS

Hua Huang

Department of Computer Science and Engineering

University of California, Merced

# The Global Positioning System (GPS)

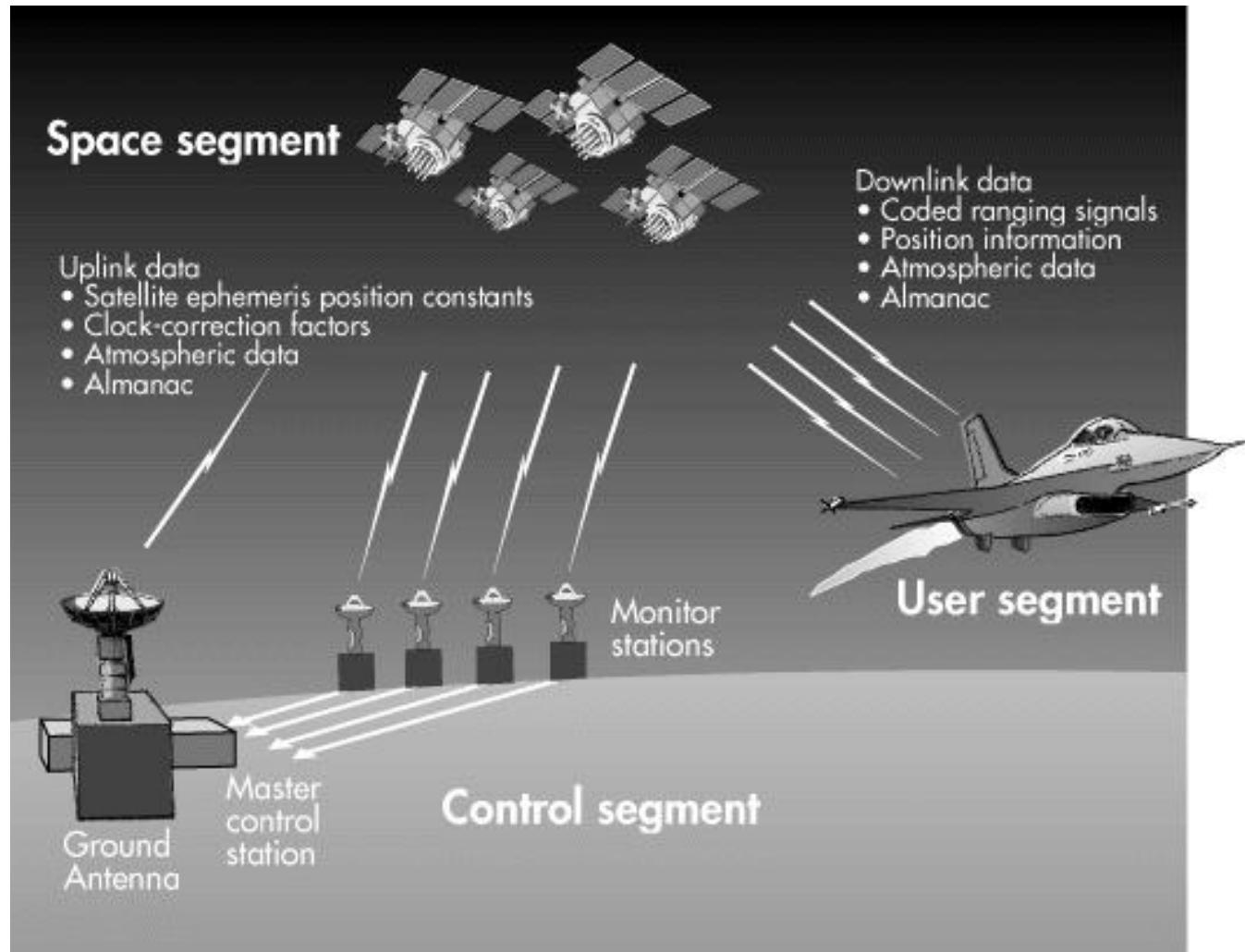
# The History of GPS

- Feasibility studies begun in 1960s.
- Pentagon appropriates funding in 1973.
- First satellite launched in 1978.
- System declared fully operational in April 1995.

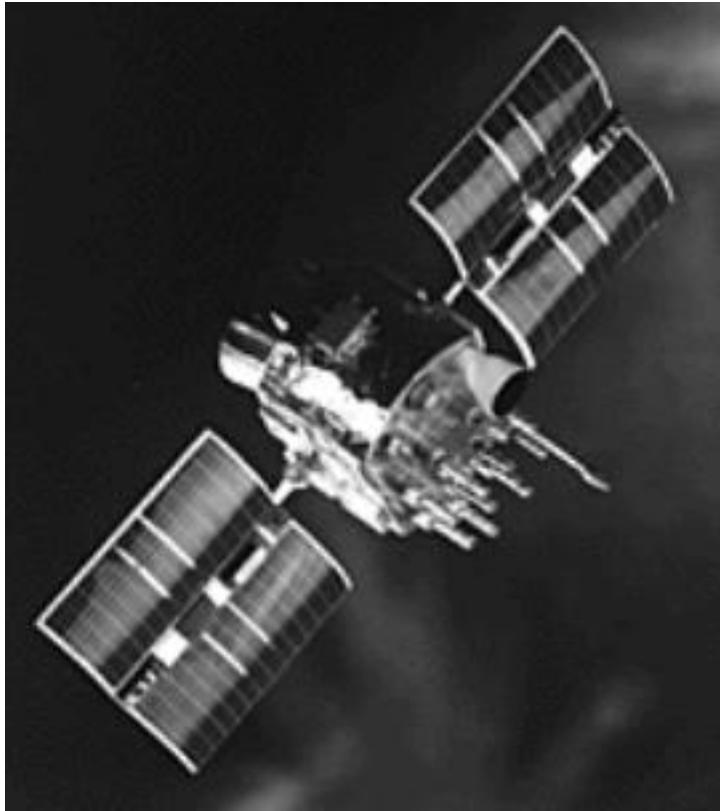


# System Components

- space (GPS satellite vehicles)
- control (tracking stations)
- users



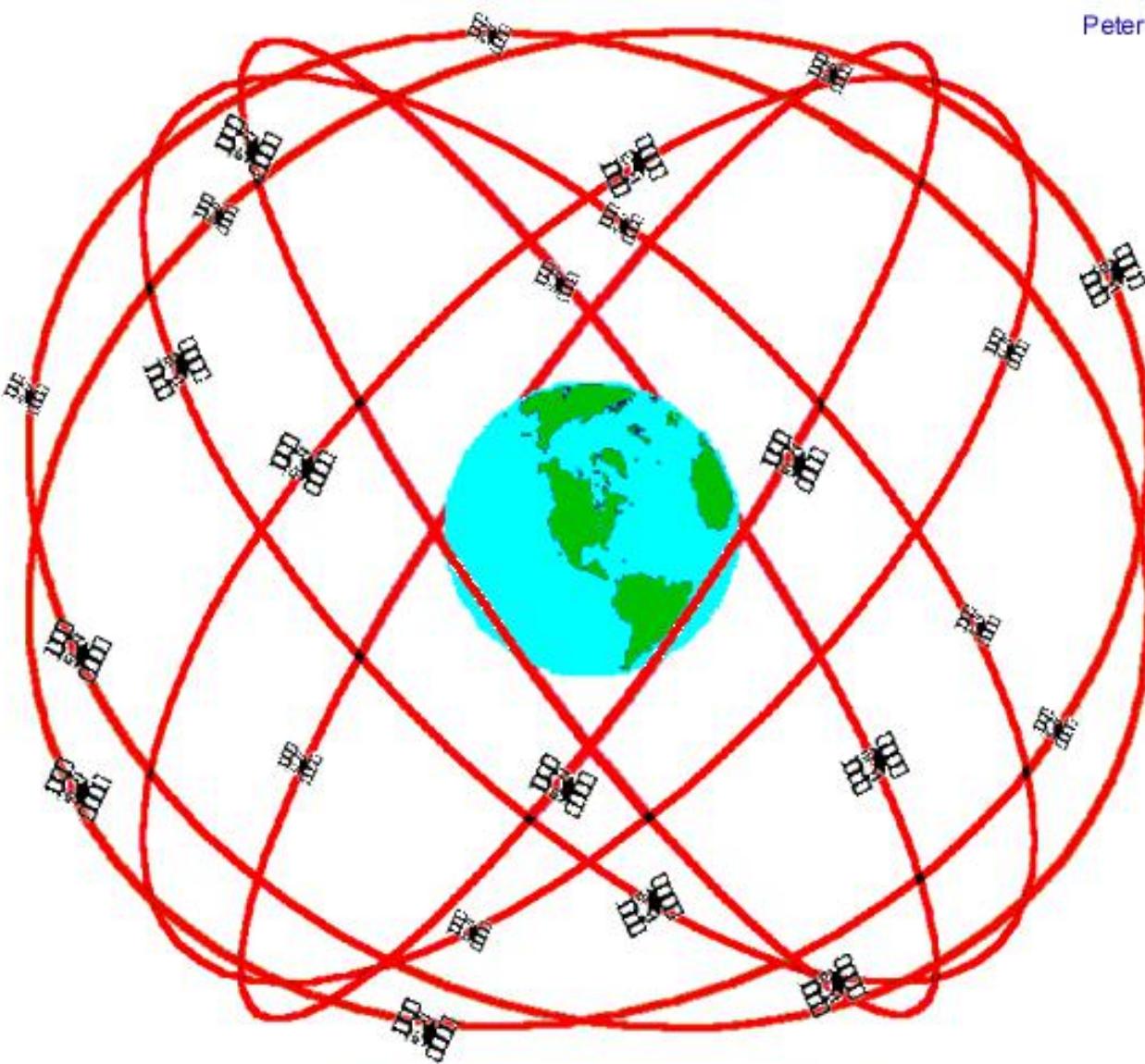
# Two generations of GPS satellite vehicles (SVs)



GPS block I:  
Experimental



GPS block II:  
Full-scale operational



**GPS Nominal Constellation**  
**24 Satellites in 6 Orbital Planes**  
**4 Satellites in each Plane**  
**20,200 km Altitudes, 55 Degree Inclination**

basic concept is that the GPS constellation replaces “stars” and gives us reference points for navigation

examples of some applications (users):

- navigation (very important for ocean travel)
- zero-visibility landing for aircraft
- collision avoidance
- surveying
- precision agriculture
- delivery vehicles
- emergency vehicles
- electronic maps
- Earth sciences (volcano monitoring; seismic hazard)
- tropospheric water vapor



examples of applications



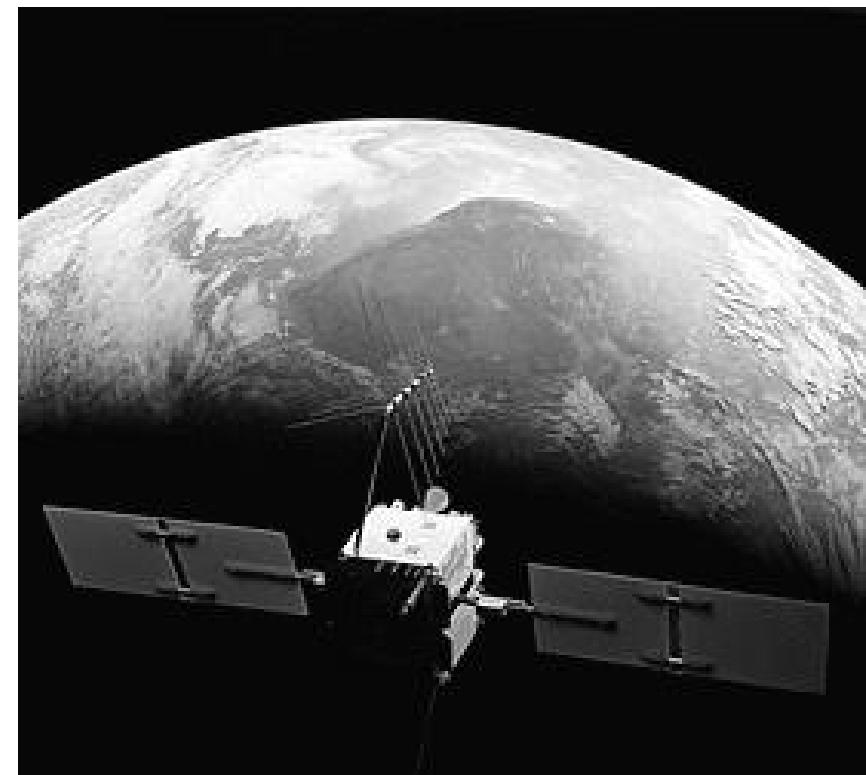
# Four Basic Functions of GPS

- Position and coordinates.
- The distance and direction between any two waypoints, or a position and a waypoint.
- Travel progress reports.
- Accurate time measurement.

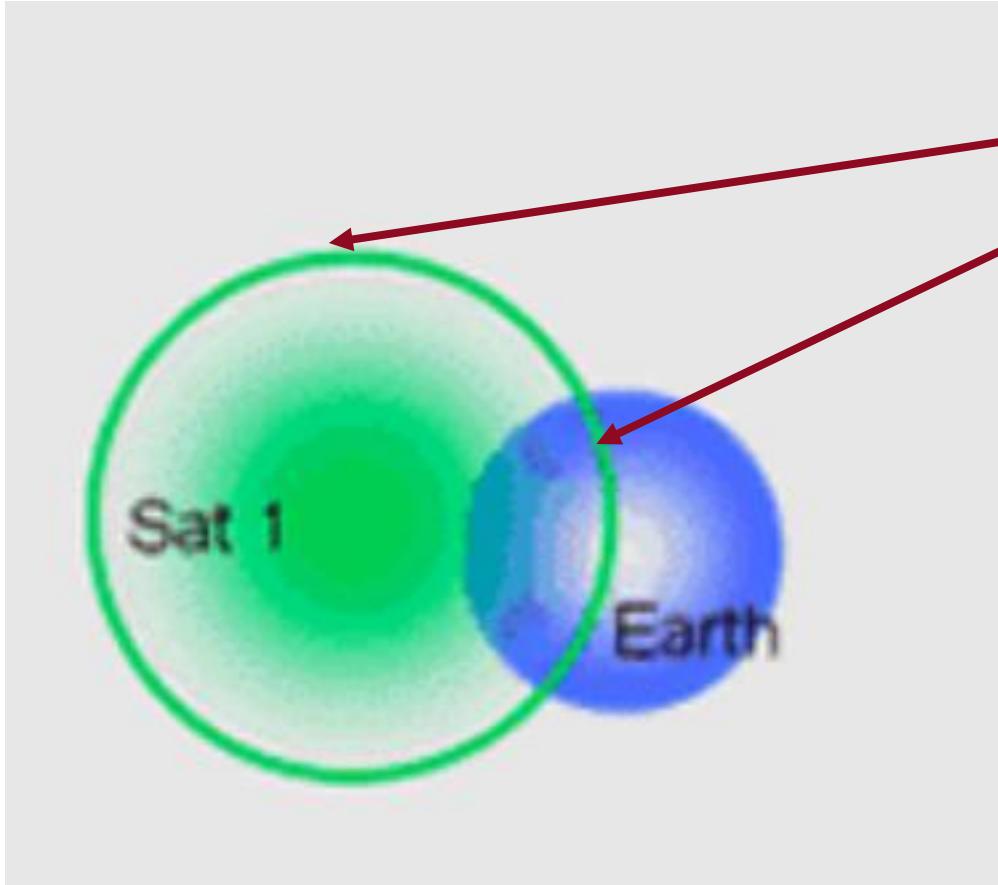


# How GPS works

- step 1: using satellite ranging
- step 2: measuring distance from satellite
- step 3: getting perfect timing
- step 4: knowing where a satellite is in space
- step 5: identifying errors

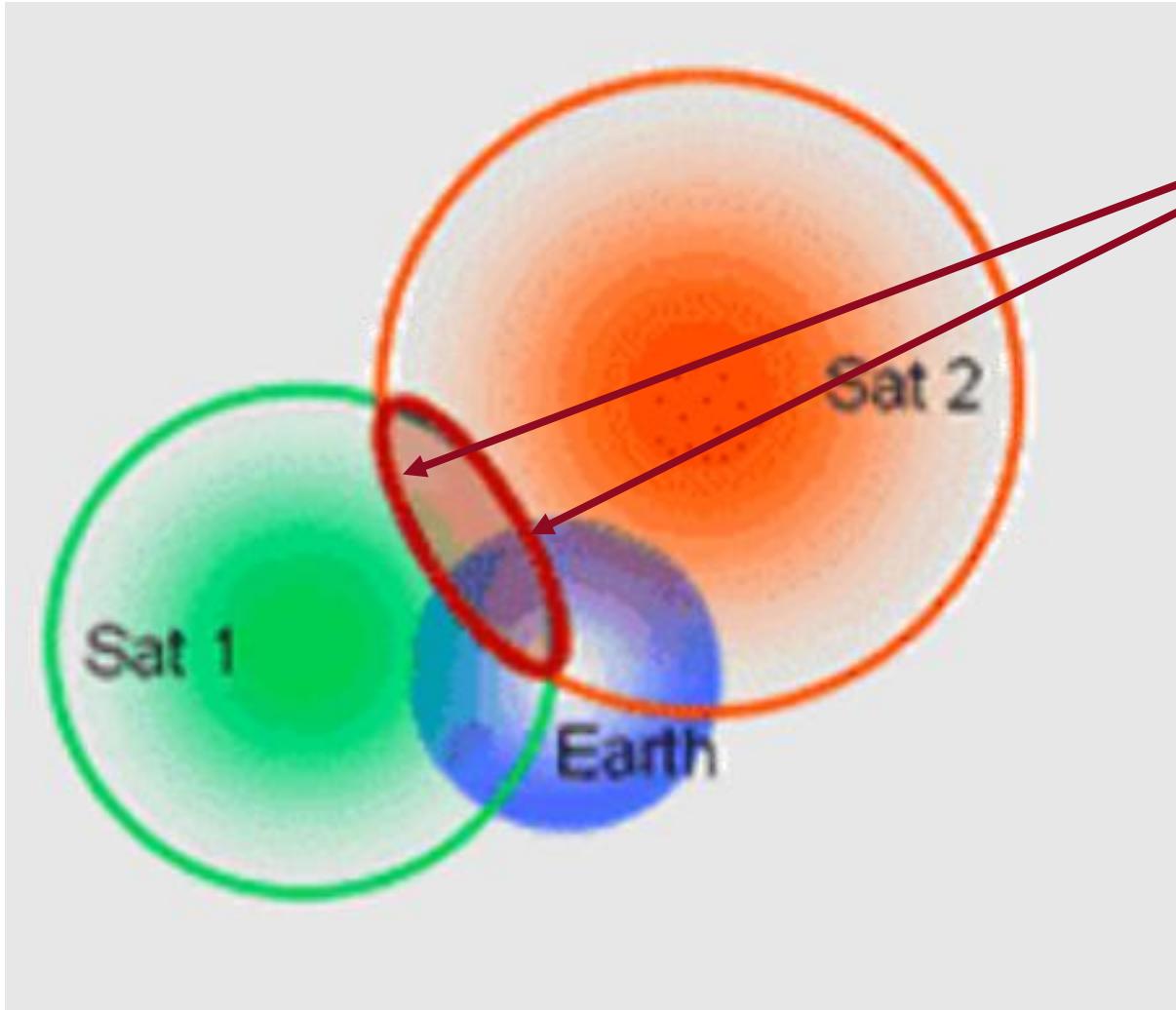


# Signal From One Satellite



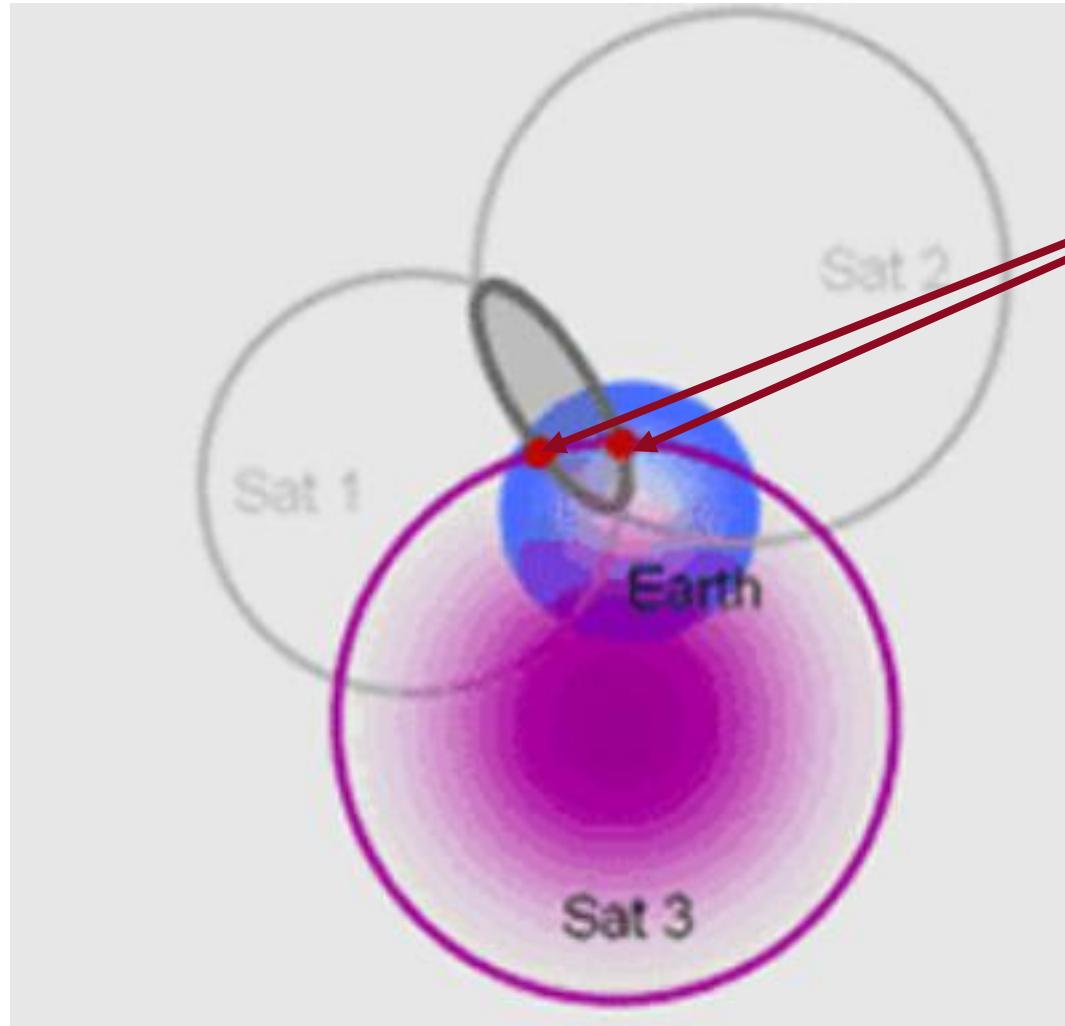
The receiver is  
somewhere on  
this sphere.

# Signals From Two Satellites



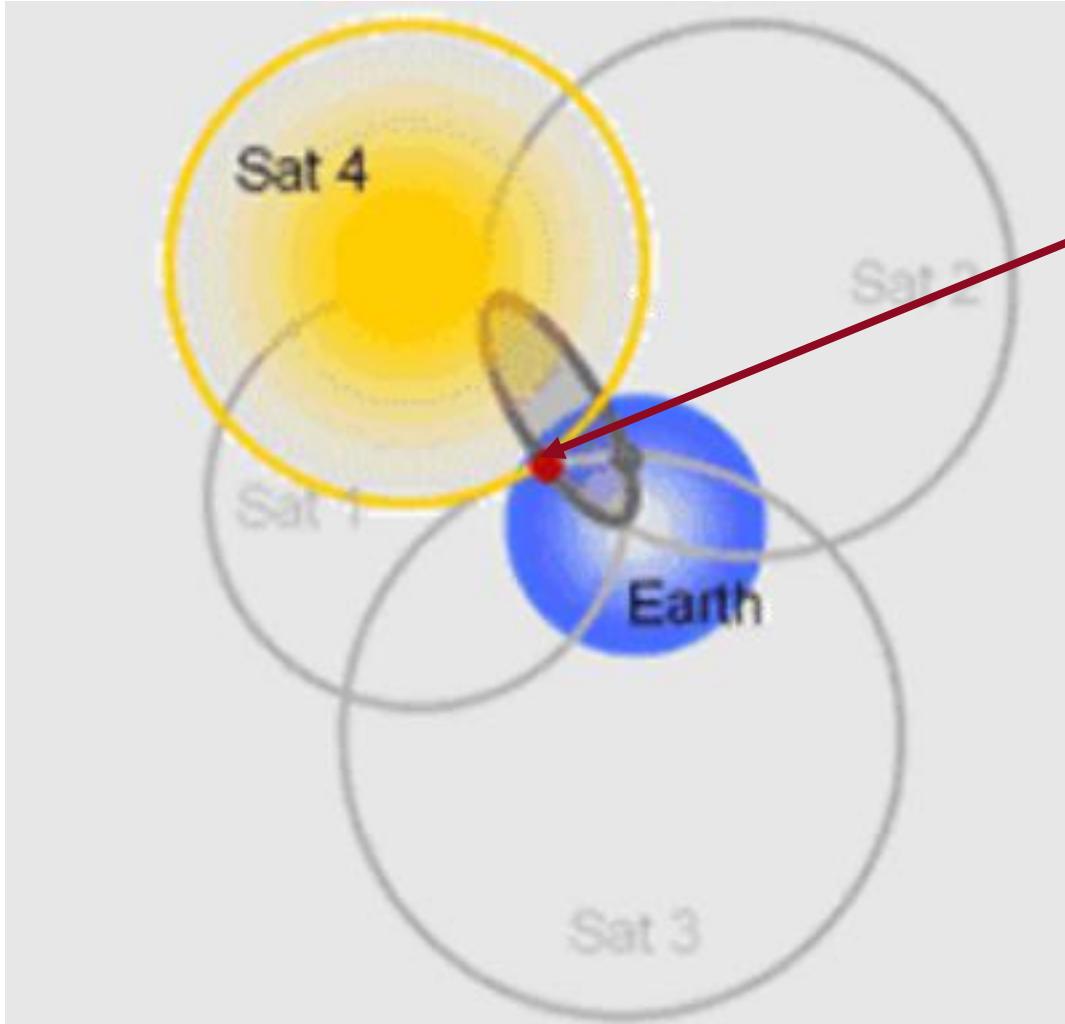
The receiver is  
somewhere on  
this circle.

# Signals From Three Satellites



The receiver is somewhere between these two points.

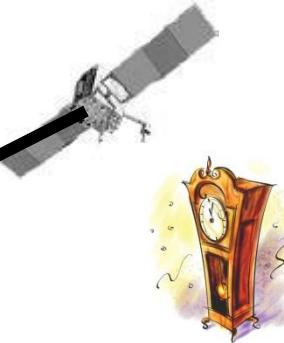
# Signals From Four Satellites



The receiver is  
here.

# Travel time

**Signal leaves at 8:03:02.12**



For example: 13,000 some miles

Radio waves travel about 186,000 miles (300,000 km) per second.



**Signal arrives at 8:03:02.19**

# High accuracy is needed

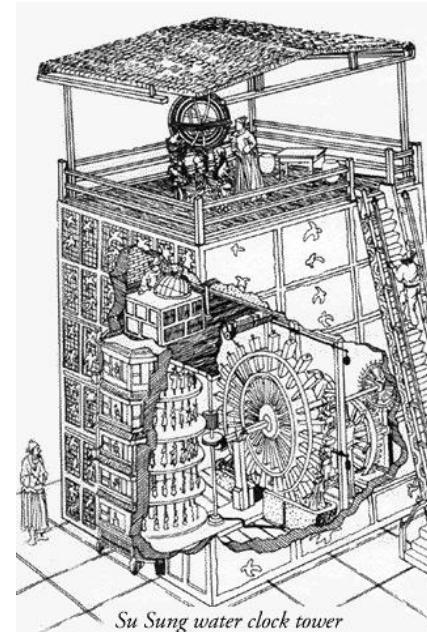
8:03:02.19

- 8:03:02.12

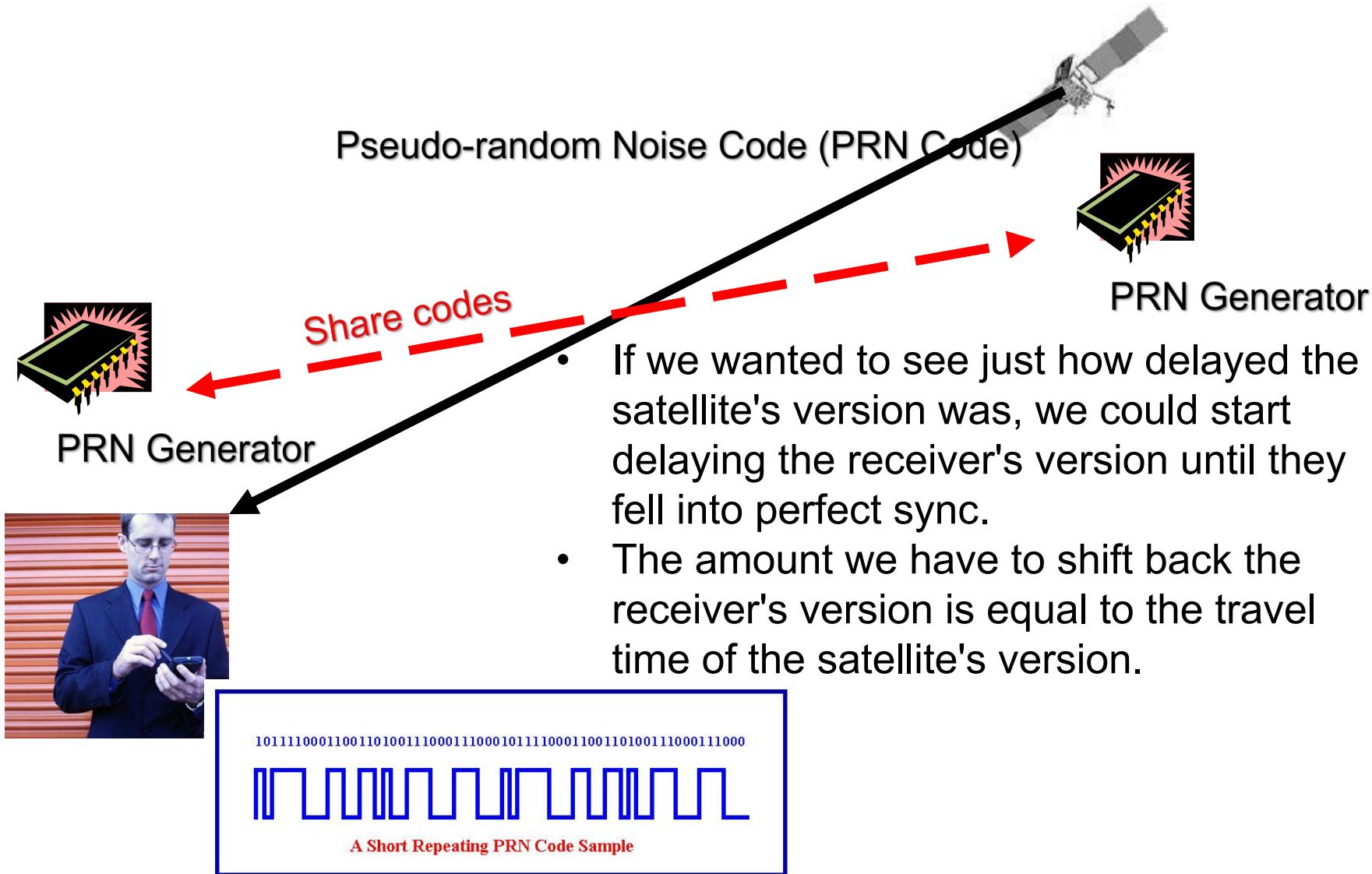
0:00:00.07

7 hundredths of a second  
difference for the 13,000 mile  
(i.e. 20,000 km) distance

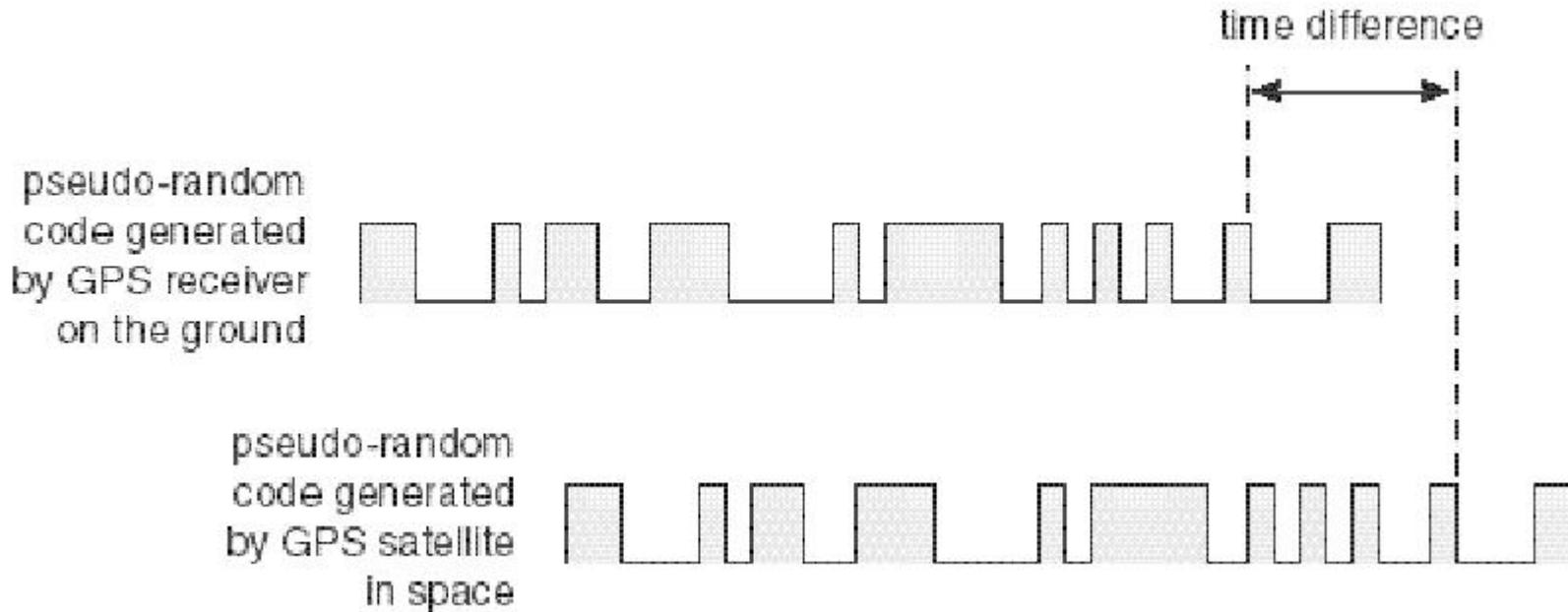
Takes some really good clocks



# So how do you measure the time difference (ranging)?



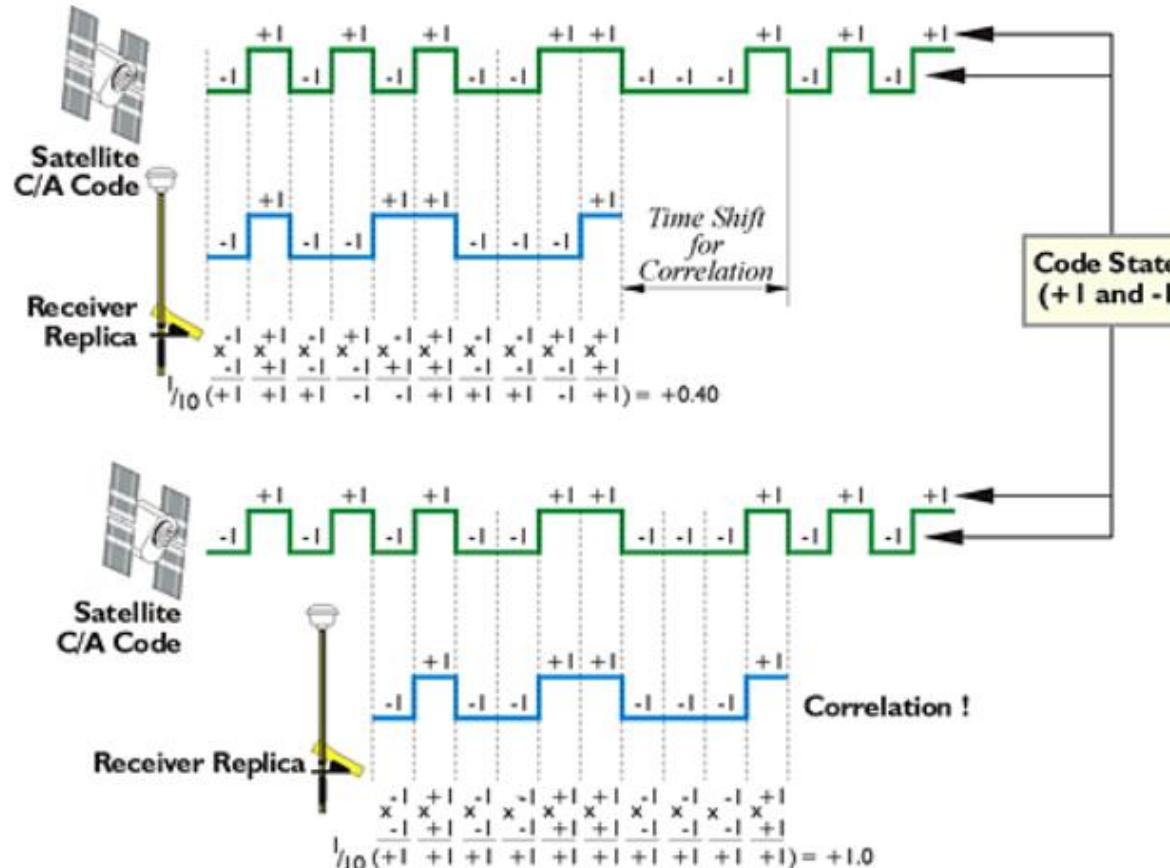
# GPS Ranging



<http://maps.unomaha.edu/Peter%20son/gis/notes/GPS.pdf>

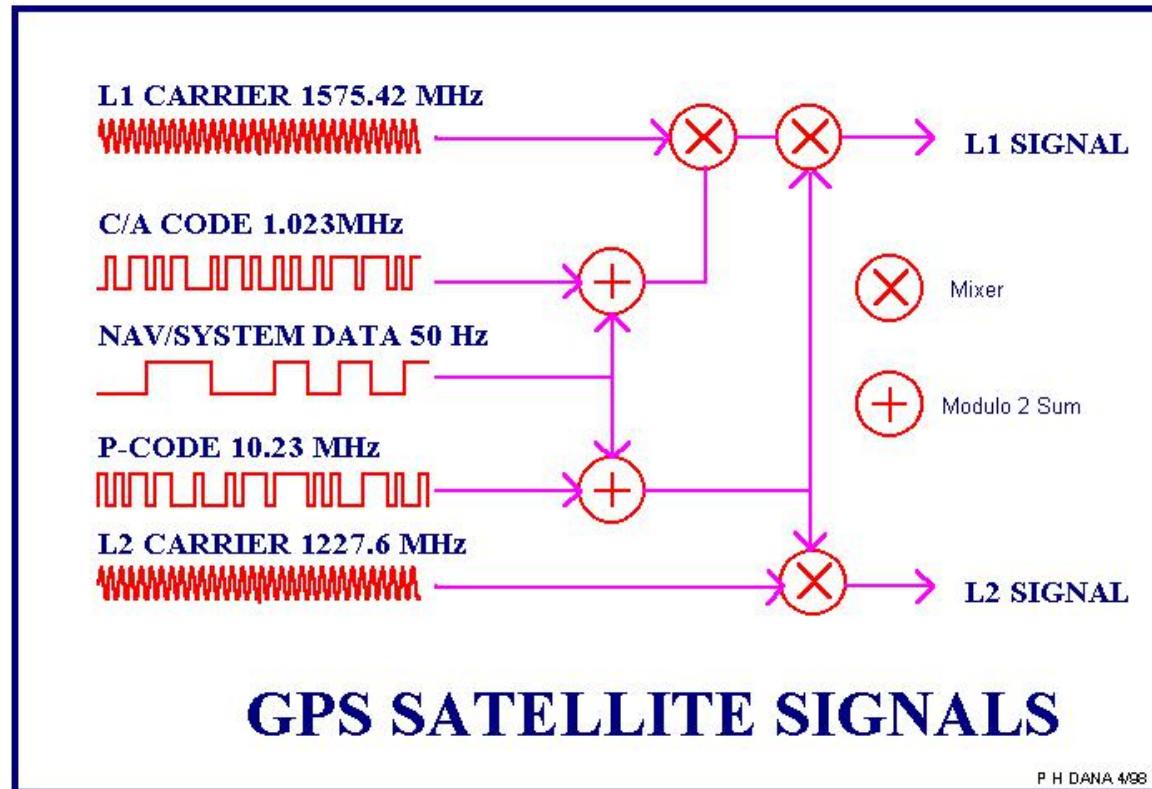
- The local receiver and the satellite generate the same pseudo-random code at the same time.
- Upon receiving the signal from the satellite, the difference of the two signal is compared.
- The time difference times speed of light determines the distance to satellite.

# Algorithm Implementation



- The signals time slices are characterized as +1 and –1 states.
- We multiply each time slices and sum them up.
- We time shift the receiver replica of the signal to find the best correlation.
- When the sum of product of the two signal equals to 1, we have a perfect match

# So what do the real signals look like?

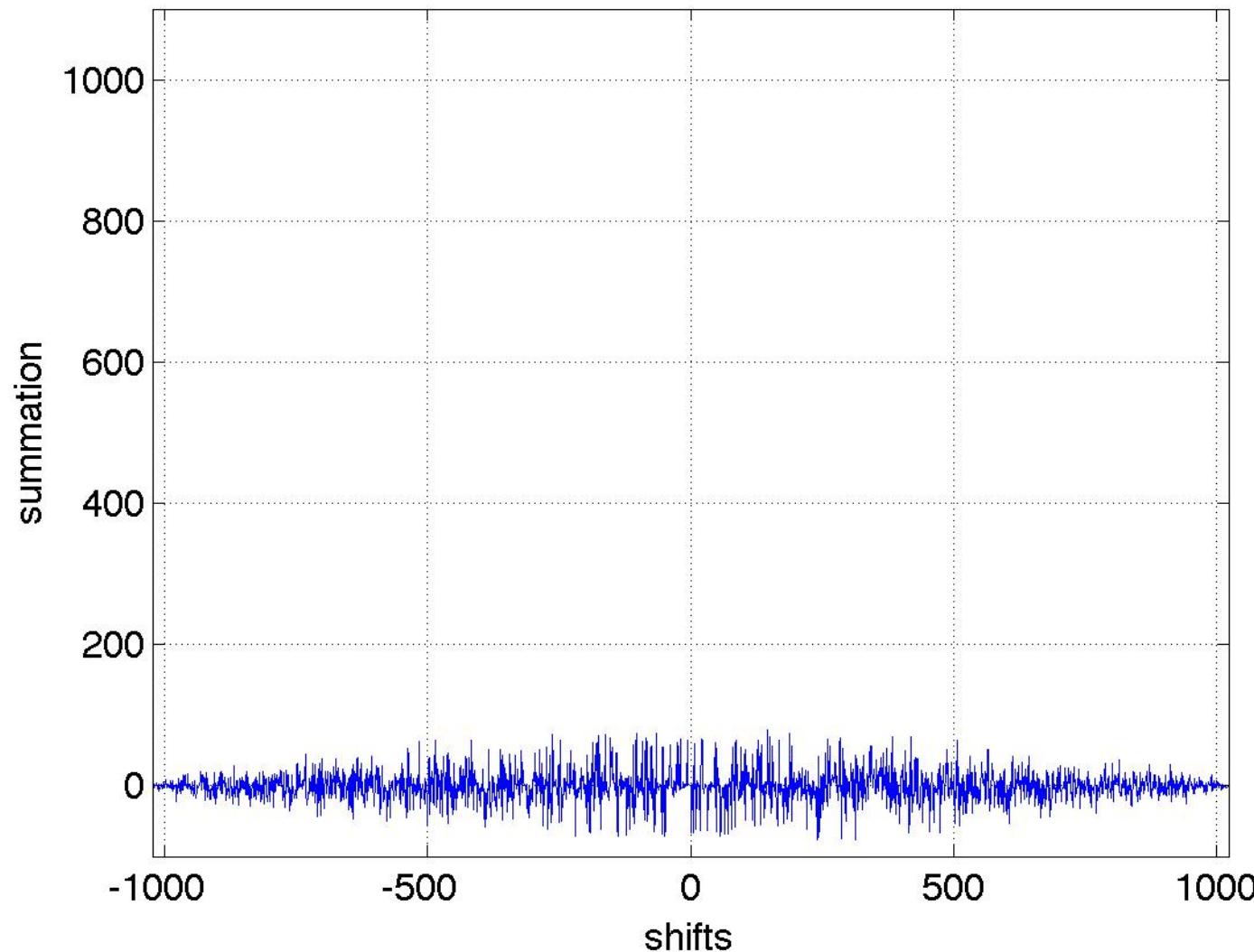


The information is sent either C/A (Course Acquisition Code) or P codes (Precision Code).

The C/A code is broadcast on L1 Carrier Frequency. 1-5 meter accuracy.  
P Code – Precision Code is used by the military (L1 and L2).

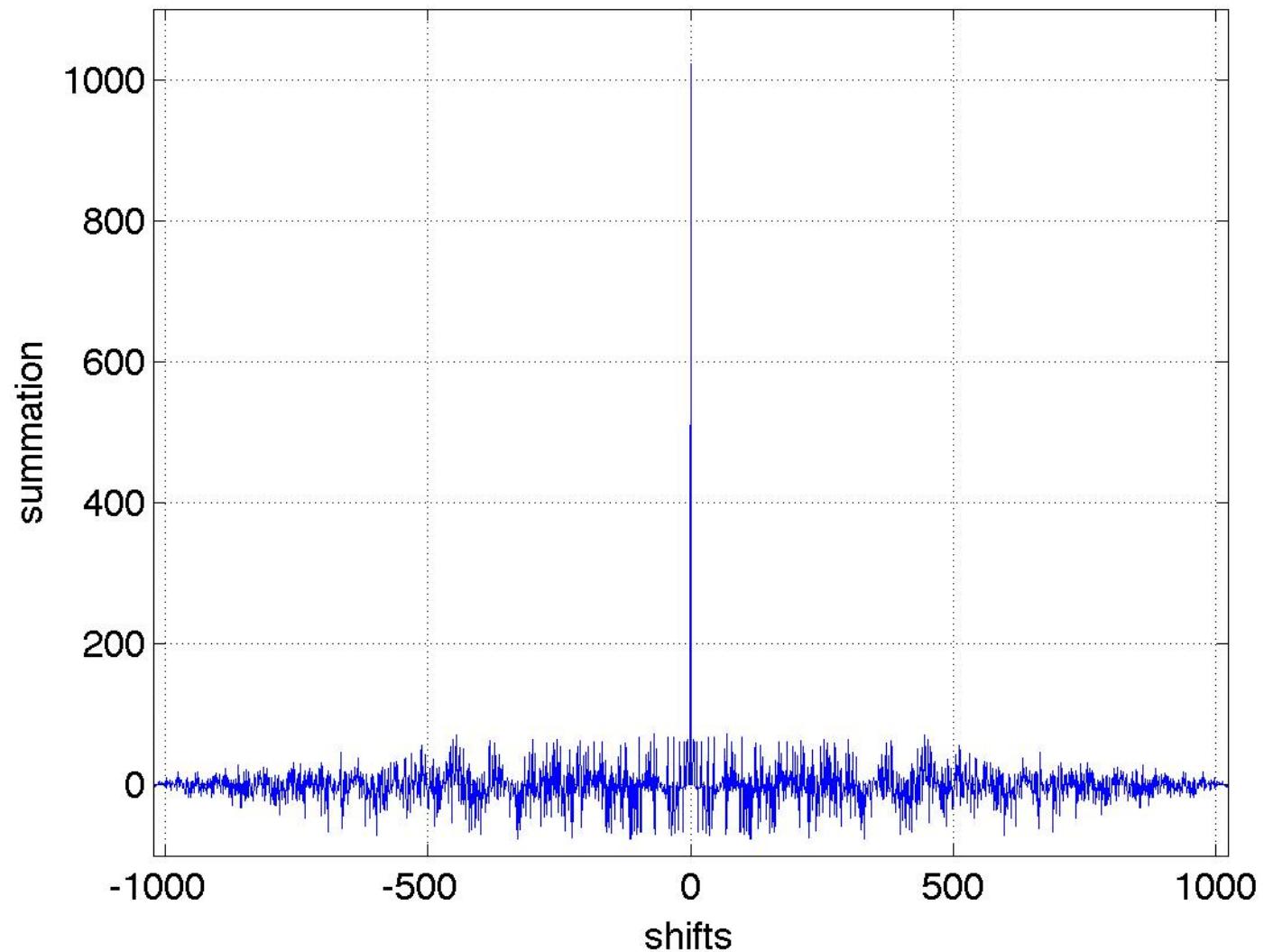
# Satellite 9 compared to Satellite 10 code

Satellite 9 compared with Satellite 10

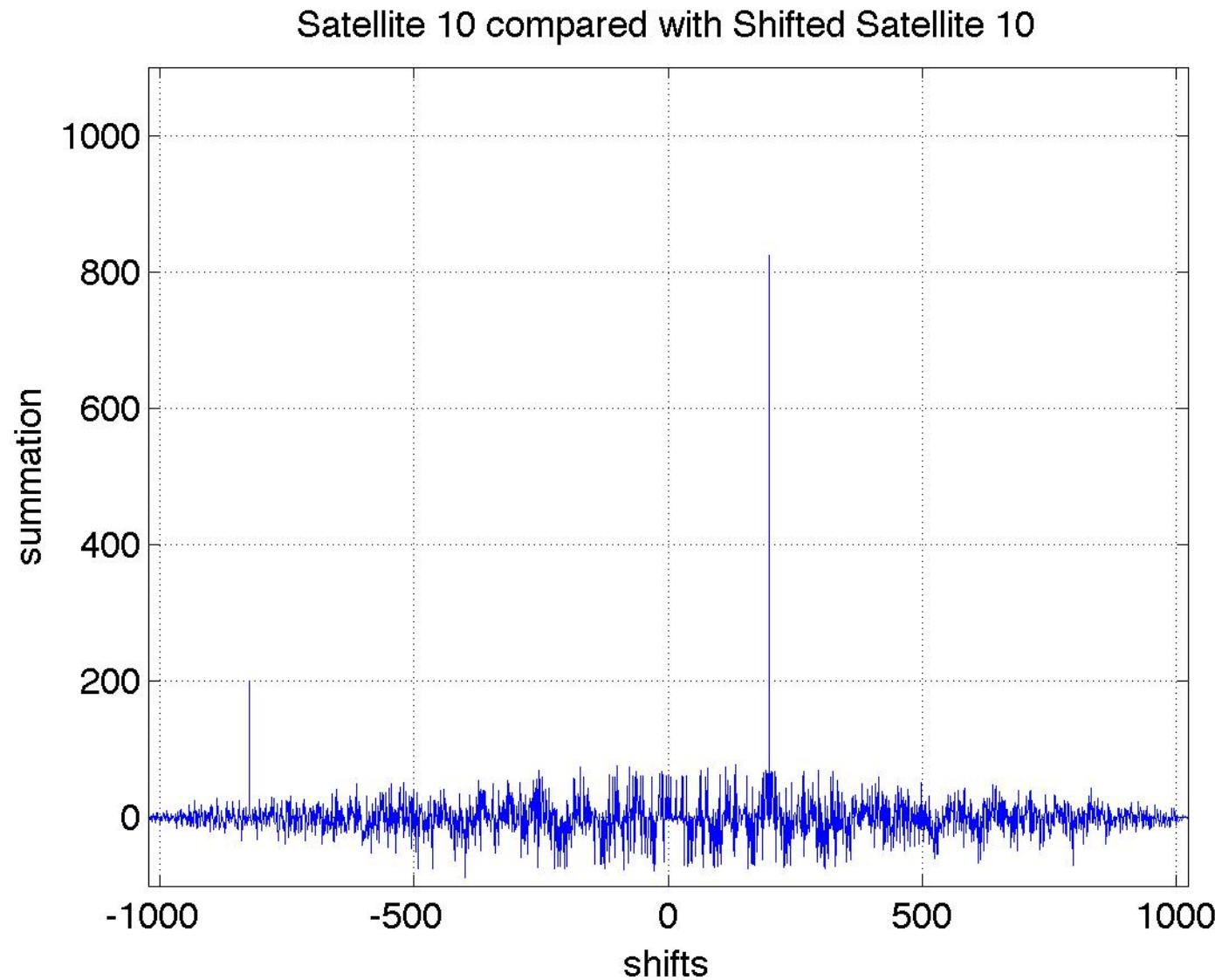


# Satellite 10 compared to Satellite 10 code

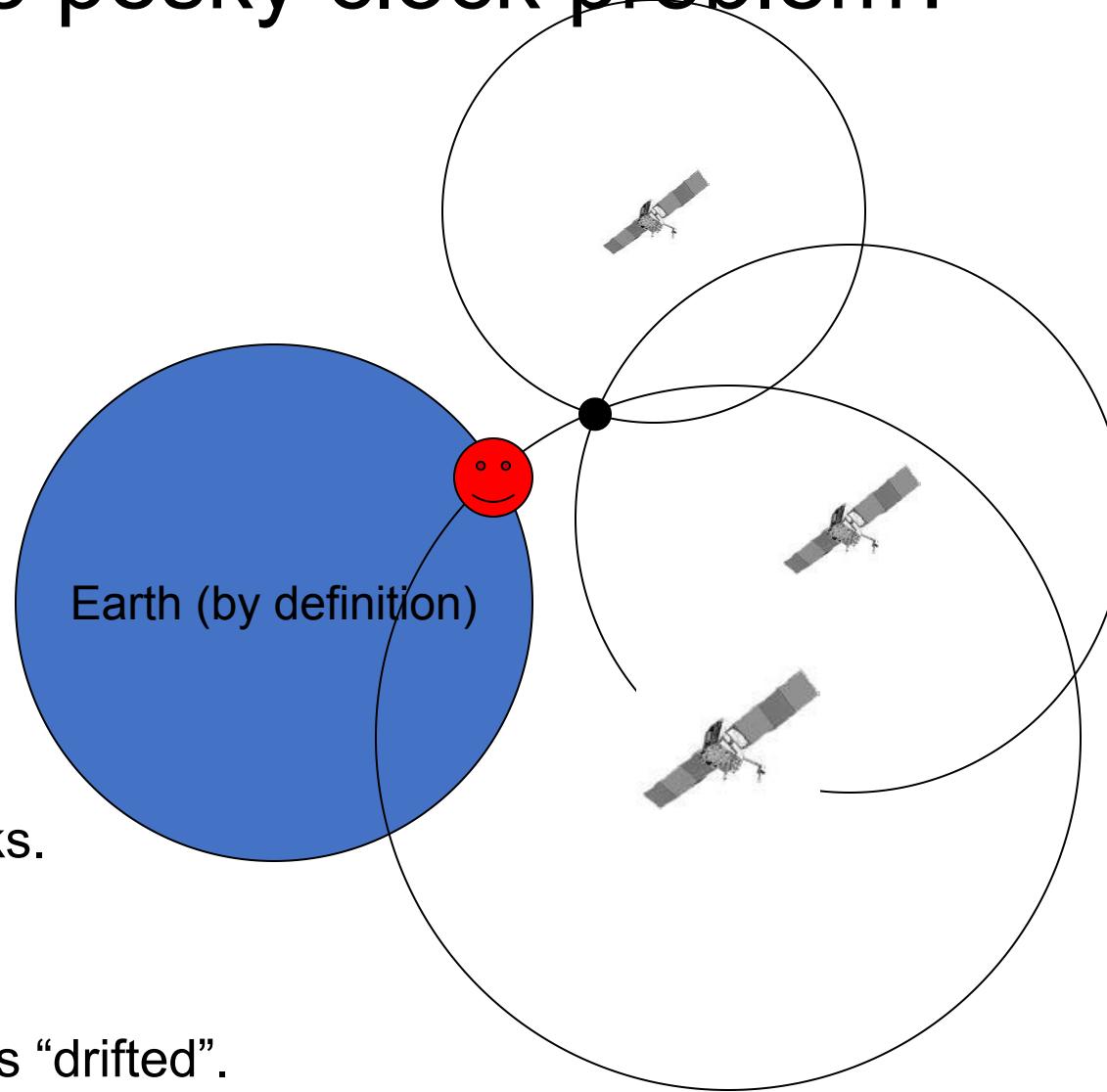
Satellite 10 compared with Satellite 10



Satellite 10 compared to Satellite 10 code that has been shifted by 200.



# Remember the pesky clock problem?

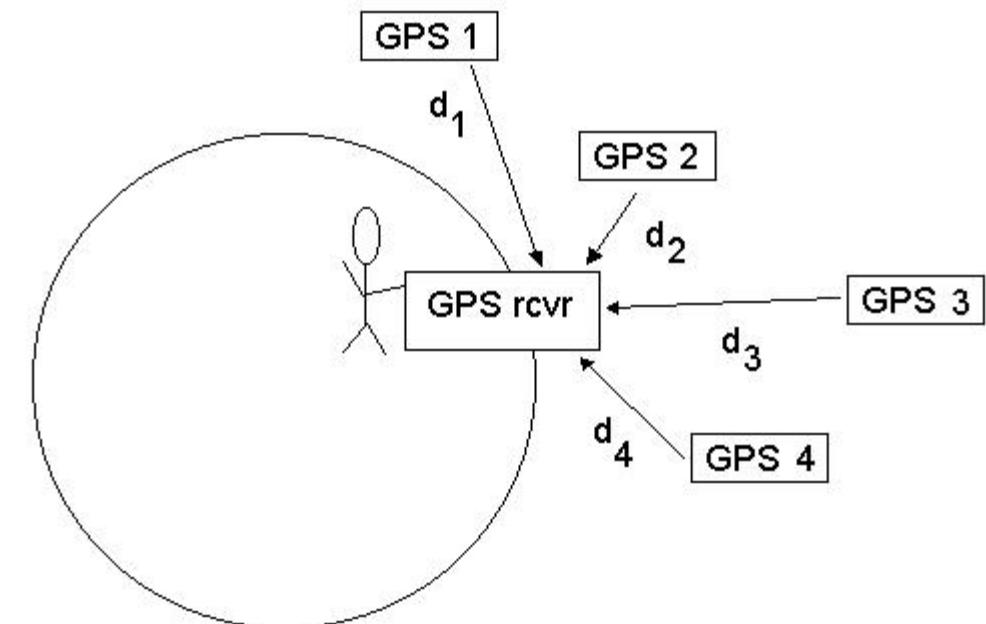


Satellites have expensive clocks.  
Our receiver doesn't!

Key idea: user clock is “drifted”.  
So our distance is off – but by a constant amount!  
Treat it as another unknown variable

# Details of how GPS works

- The GPS calculates four unknowns  $\mathbf{x}, \mathbf{y}, \mathbf{z}, t_B$ , where  $\mathbf{x}, \mathbf{y}, \mathbf{z}$  are the receiver's coordinates, and  $t_B$  is the time correction for the GPS receiver's clock.
- For satellite  $i$ , the position  $x_i, y_i, z_i$  is known



# Solving for receiver position

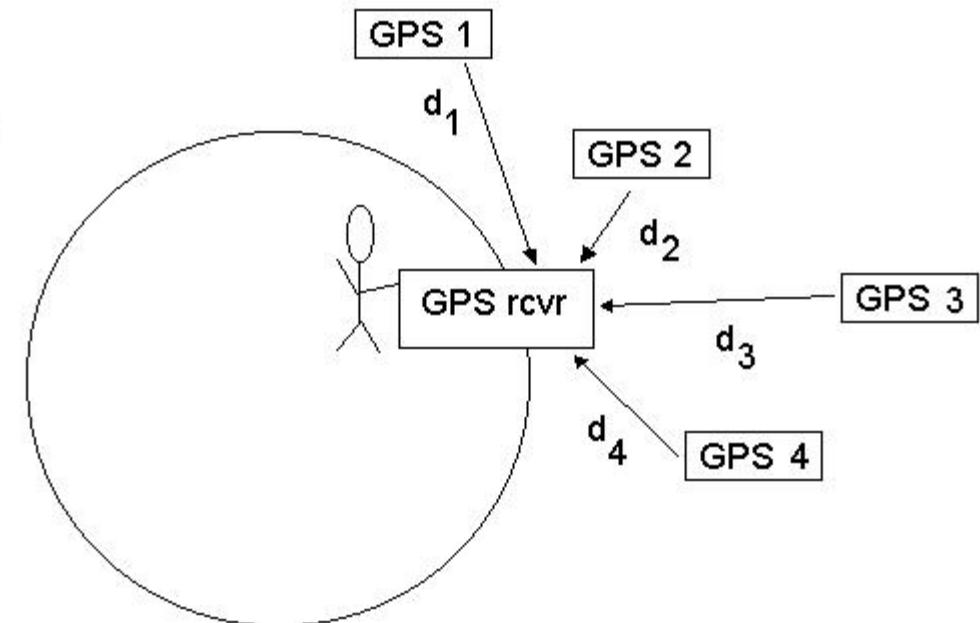
$$\sqrt{(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2} + ct_B = d_1$$

$$\sqrt{(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2} + ct_B = d_2$$

$$\sqrt{(x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2} + ct_B = d_3$$

$$\sqrt{(x - x_4)^2 + (y - y_4)^2 + (z - z_4)^2} + ct_B = d_4$$

- $c$ : speed of light
  - $x_i$ : the satellite positions, known to the receivers
- A quadratic programming problem. Solved by mature solvers



# Satellite position information

- GPS satellites transmit information about their location (current and predicted), timing and "health" via what is known as **ephemeris** data.
- This data is used by the GPS receivers to estimate location relative to the satellites and thus position on earth.

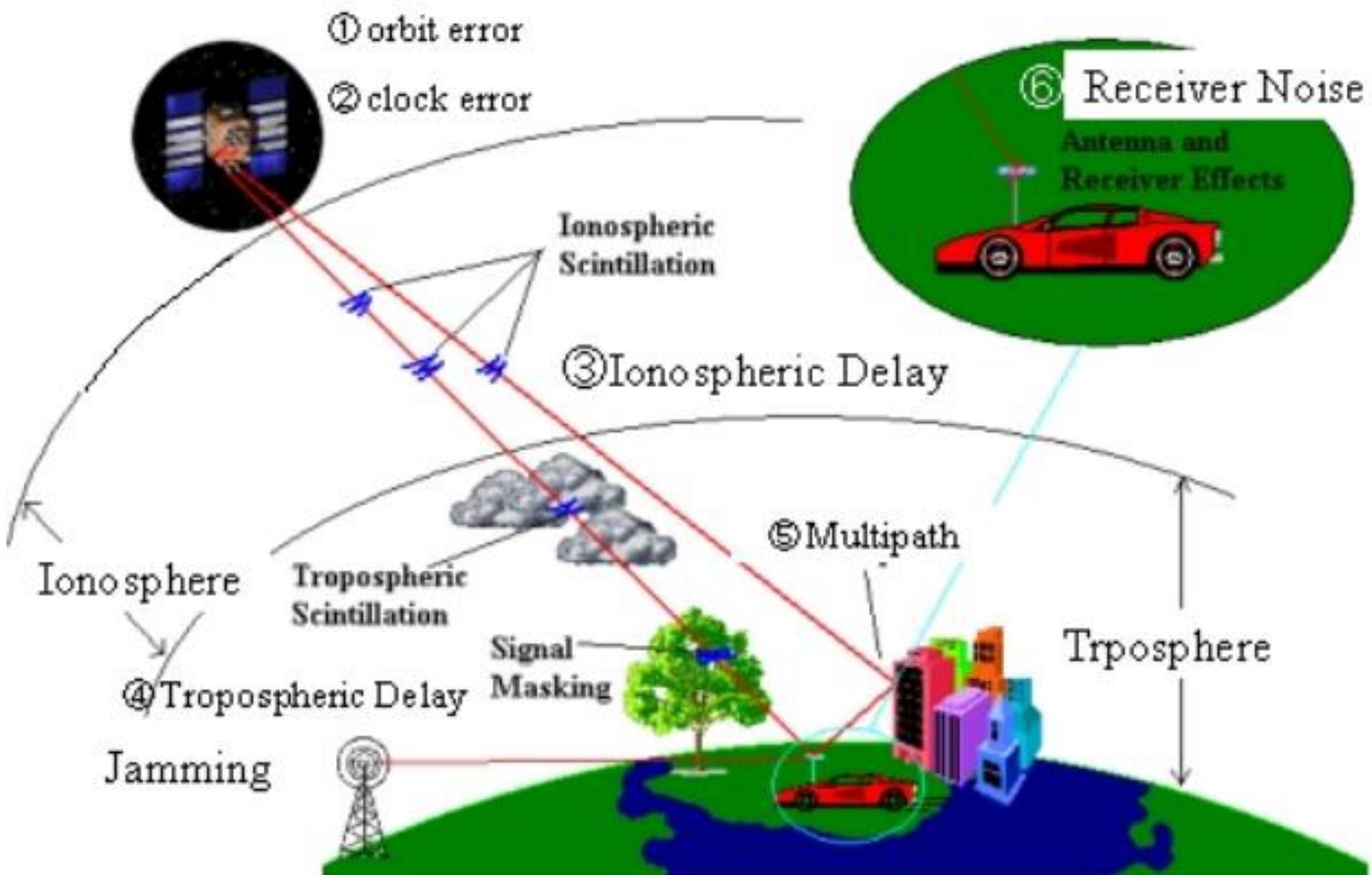
# Adafruit Ultimate GPS Module

- -165 dBm sensitivity, 10 Hz updates, 66 channels
- 5V friendly design and only 20mA current draw
- Breadboard friendly + two mounting holes
- RTC battery-compatible
- Built-in datalogging
- PPS output on fix
- Internal patch antenna
- u.FL connector for external active antenna



<https://www.adafruit.com/product/746>

# Errors on GPS Signal



# GPS error: Satellite clocks

- Satellites use atomic clocks, which are very accurate but can drift up to a millisecond. Errors: 1.5-3.6 meters
- Mitigation: minimized by calculating clock corrections (at monitoring stations) and transmitting the corrections along with the GPS signal to appropriately outfitted GPS receivers.

# GPS error: Orbital Errors

- GPS receivers calculate coordinates relative to the known locations of satellites in space. Errors<1 meter
- Mitigation: The GPS Control Segment monitors satellite locations at all times, calculates orbit eccentricities, and compiles these deviations in documents called ephemerides.
- GPS receivers that are able to process ephemerides can compensate for some orbital errors.

# GPS error: Upper Atmosphere (Ionosphere)

- As GPS signals pass through the upper atmosphere (the ionosphere 50-1000km above the surface), signals are delayed and deflected. Errors: 5-7 meters
- Mitigation: By modeling ionosphere characteristics, GPS monitoring stations can calculate and transmit corrections to the satellites, which in turn pass these corrections along to receivers.

# GPS error: Lower Atmosphere (Troposphere)

- The lower atmosphere delays GPS signals, adding slightly to the calculated distances between satellites and receivers.  
Errors: < 1 meter
- Note: weather conditions, such as clouds, storms, and rains, have limited impacts on accuracy

# GPS error: Multipath Effects

- Ideally, GPS signals travel from satellites through the atmosphere directly to GPS receivers.
- In reality, GPS receivers must discriminate between signals received directly from satellites and other signals that have been reflected from surrounding objects, such as buildings, trees, and even the ground. Errors: up to 1.2 meters
- Mitigation: use antenna technique to track signals that are at least 15 degrees above horizon.

# GPS error: Wireless Interference

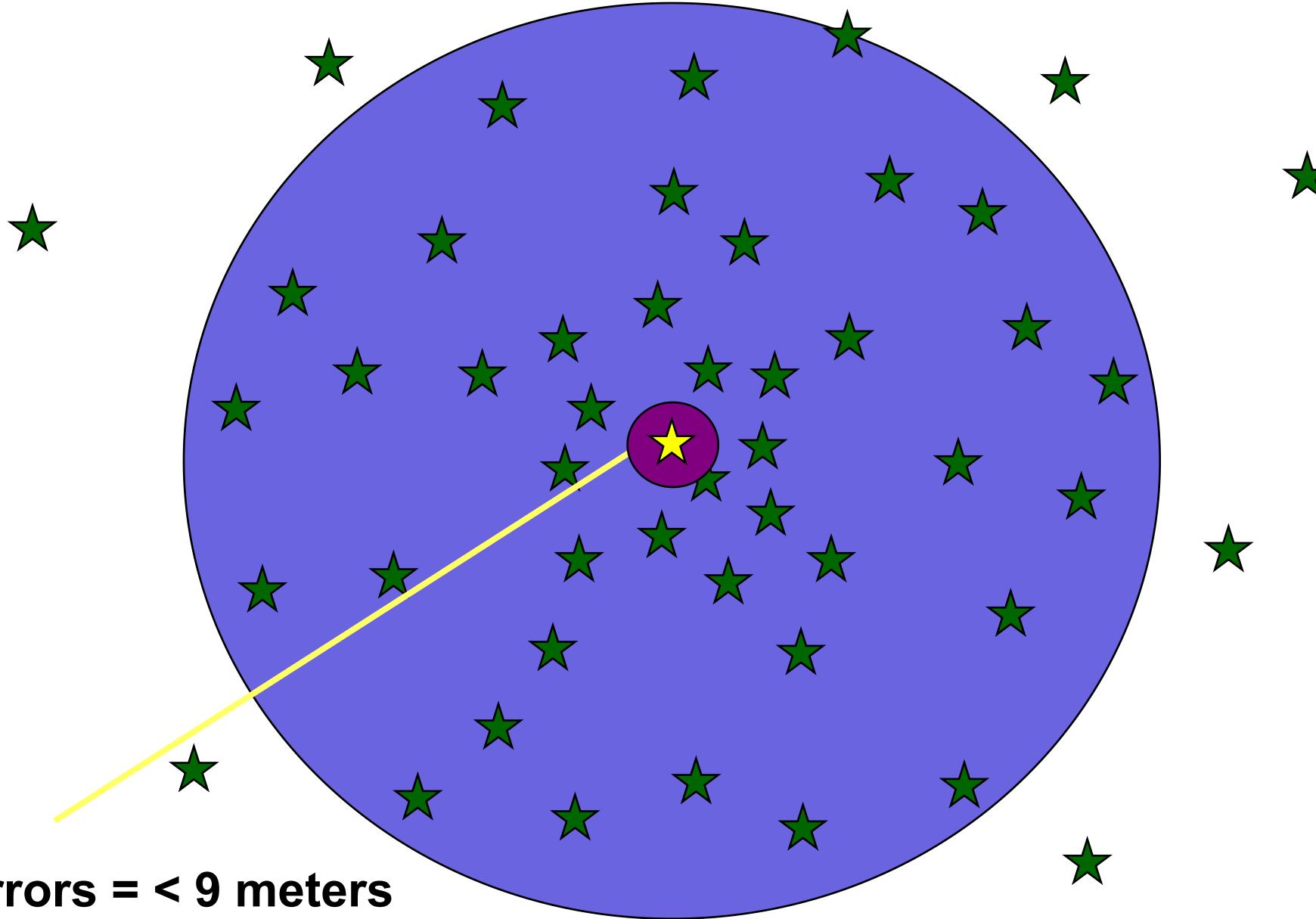
- Like any other wireless systems, GPS signals are susceptible to interference. Error: from small to unbounded.

# Sources of GPS Error

<u>Source</u>	<u>Amount of Error</u>
Ø Satellite clocks:	1.5 to 3.6 meters
Ø Orbital errors:	< 1 meter
Ø Ionosphere:	5.0 to 7.0 meters
Ø Troposphere:	0.5 to 0.7 meters
Ø Receiver noise:	0.3 to 1.5 meters
Ø Multipath:	0.6 to 1.2 meters

Errors are cumulative and increased by PDOP.

# Receiver Errors are Cumulative!



# Which sources of errors?

- GPS navigation errors in a city with tall buildings
- Drone GPS spoofing

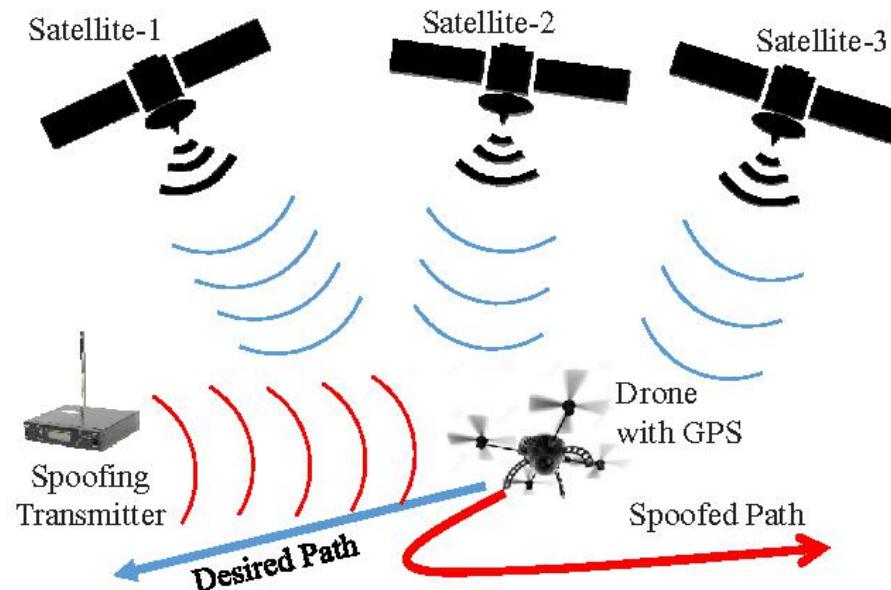
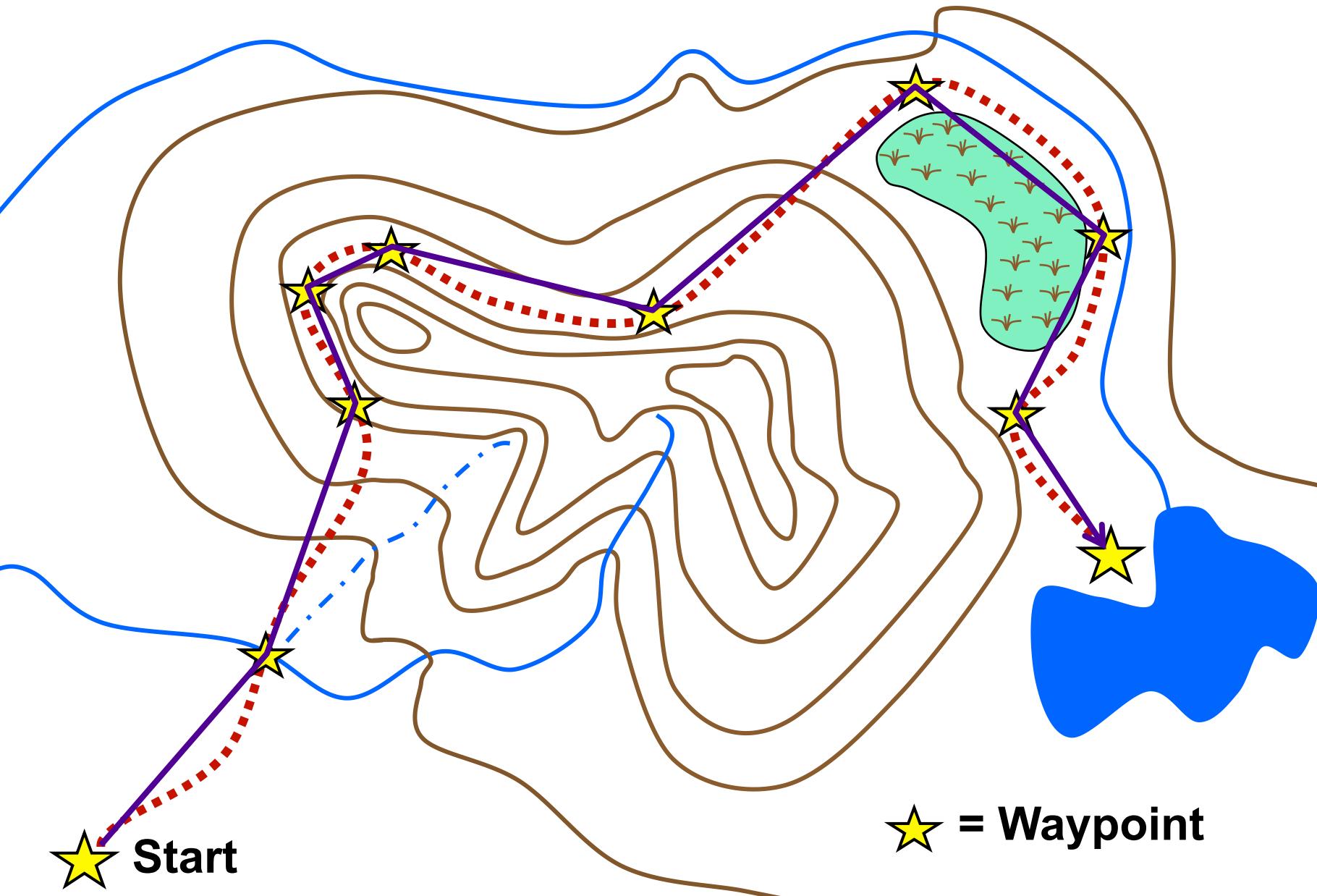


Fig. 4: GPS spoofing attack Scenario, which changes the actual

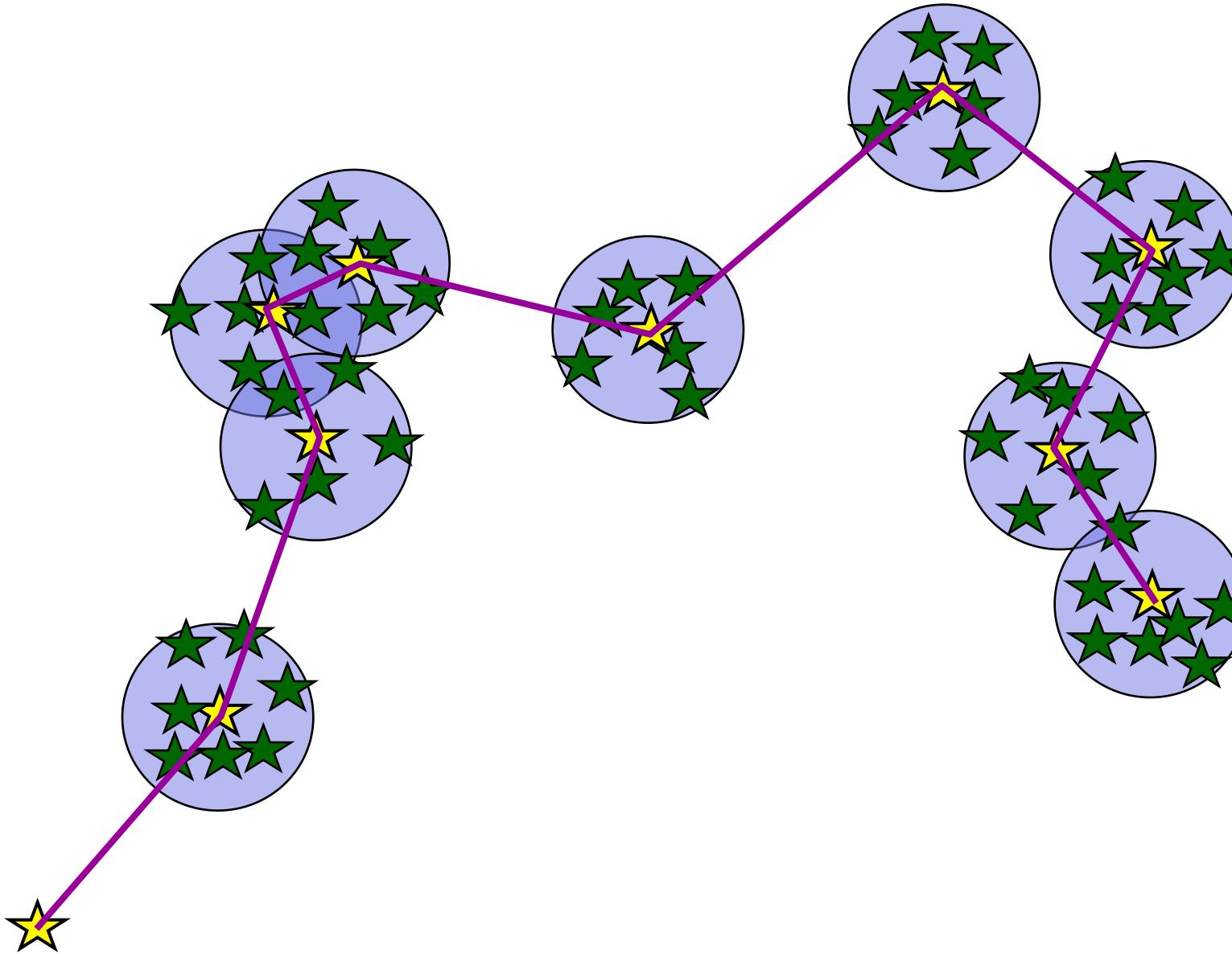
# Waypoint

- A waypoint is based on coordinates entered into a GPS receiver's memory.
- It can be created for any remote point on earth.
- It must have a receiver designated code or number, or a user supplied name.
- Once entered and saved, a waypoint remains unchanged in the receiver's memory until edited or deleted.

# Planning a Navigation Route



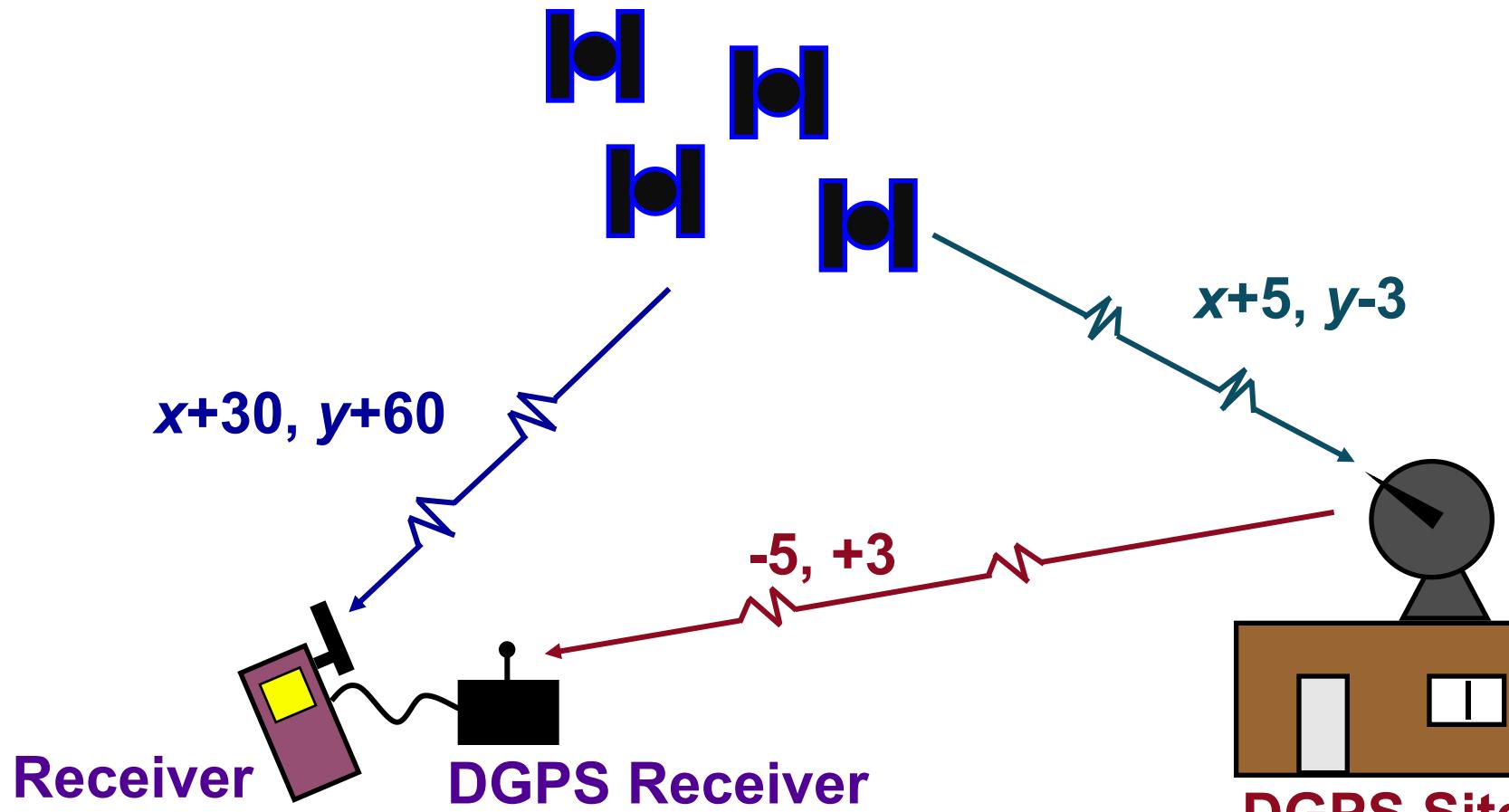
# How A Receiver Sees Your Route



# Differential GPS

- Improves nominal GPS accuracy of 15m to up to 3cm
- Basic idea:
  - Use a network of fixed reference ground-based stations to broadcast the difference between GPS result and the ground-truth position
  - The GPS receiver use this difference to correct its own errors.

# Real Time Differential GPS



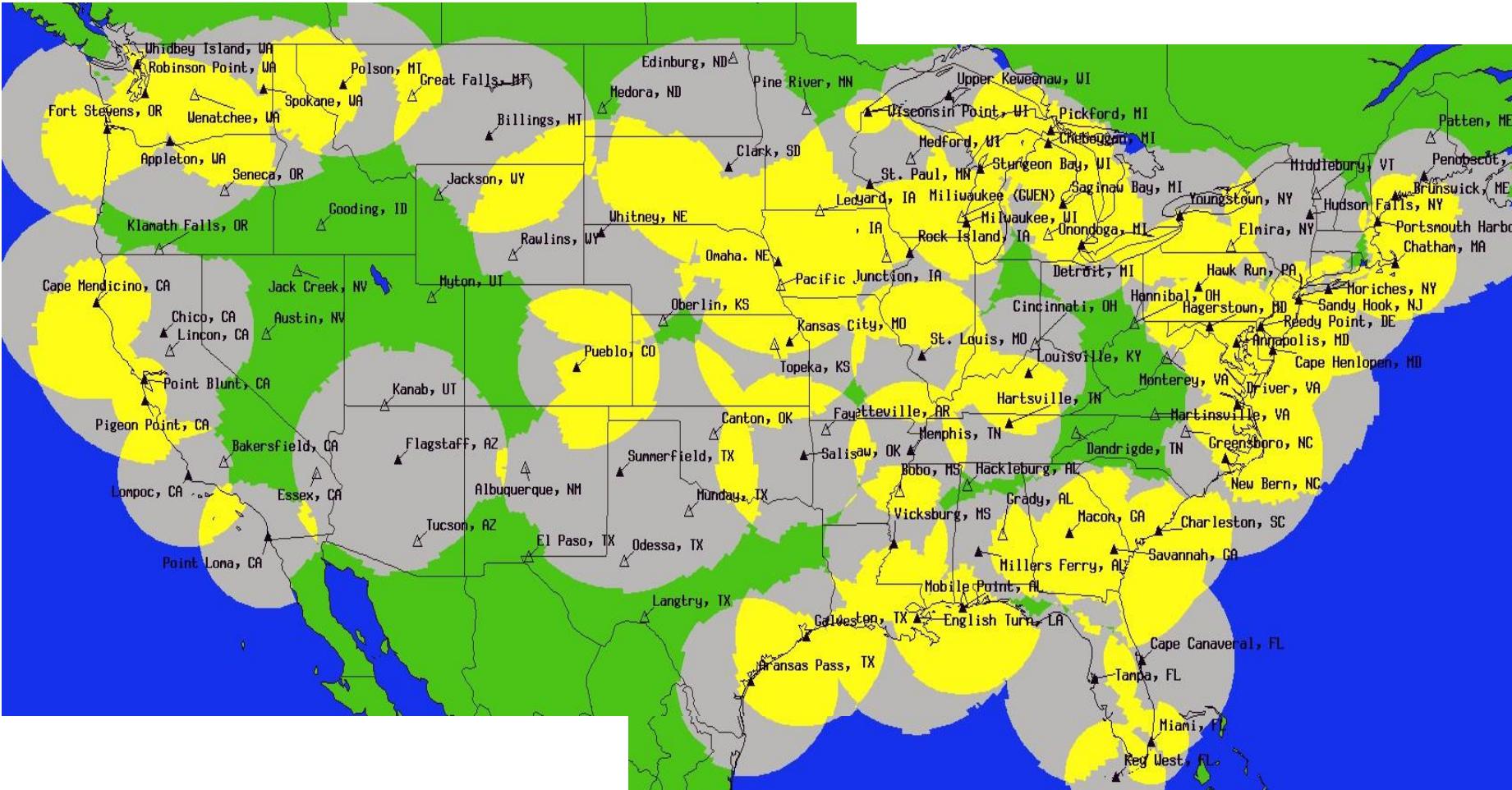
DGPS correction =  $x+(30-5)$  and  $y+(60+3)$

Corrected coordinates =  $x+25, y+63$

True coordinates =  $x, y$

Correction =  $-5, +3$

# National Differential Global Positioning System Coverage



Yellow areas show overlap between NDGPS stations. Green areas are little to no coverage.

# Exercise

Find the ground truth coordinate for receiver 1

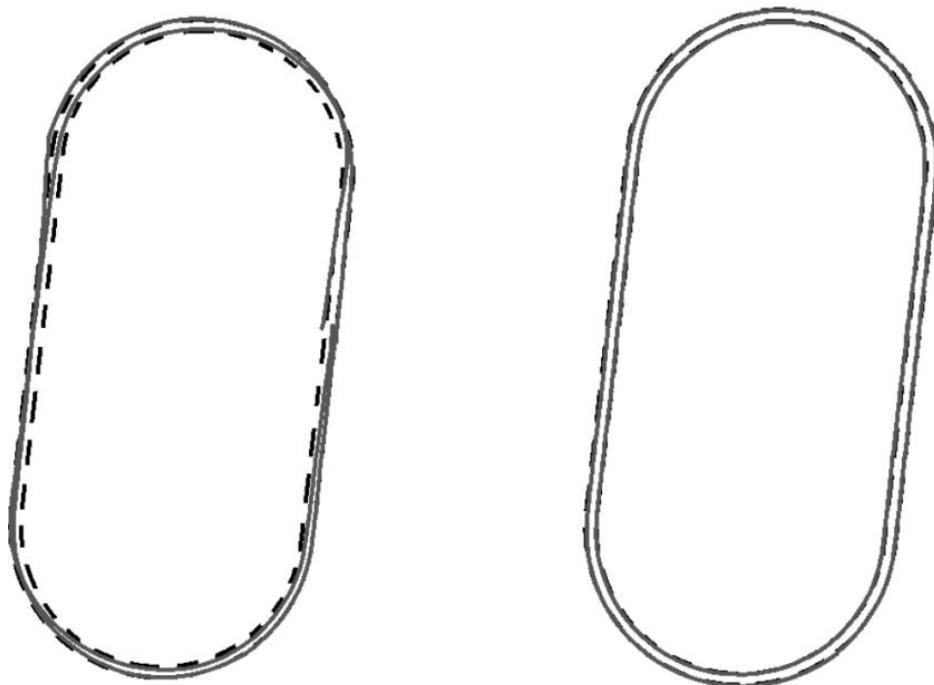
	Lat_meas	Long_meas	alt_meas	Lat_ground	Long_ground	Alt_ground
DGPS	37.0366833	35.3744433	60	37.0367146	35.3744596	64.5
Receiver 1	37.03671	35.3744467	65			

# Question

- What types of errors can be mitigated by DGPS
  - ∅ Satellite clocks
  - ∅ Orbital errors
  - ∅ Ionosphere
  - ∅ Troposphere
  - ∅ Receiver noise
  - ∅ Multipath

# Low-cost Differential GPS\*

- Use multiple GPS receivers with known relative positions
- Sub-meter accuracy



# CSE 162 Mobile Computing

## Lecture 15: Location Programming and Processing

Hua Huang

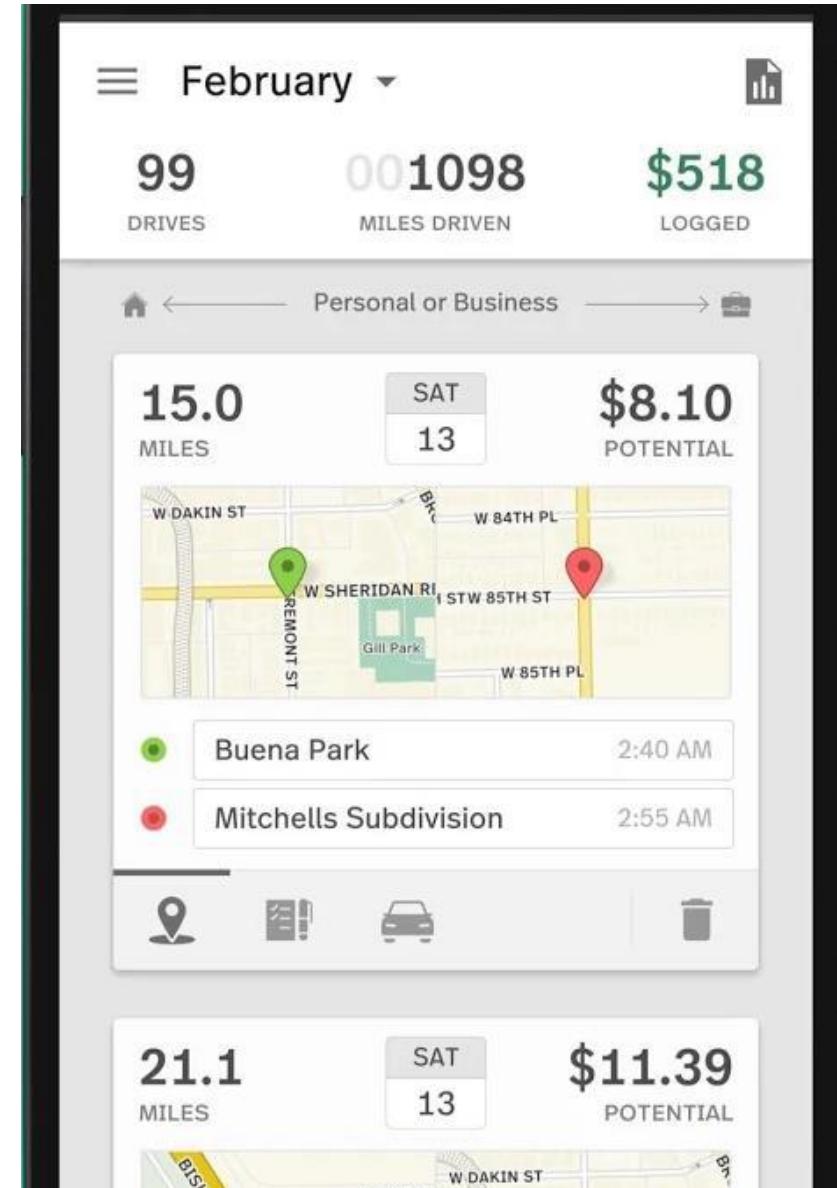
Department of Computer Science and Engineering

University of California, Merced

# **Some Interesting Location-Aware Apps**

# MileIQ

- **The Problem:** Mileage tracking is useful but a burden.
  - IRS deductions on taxes
  - Some companies reimburse employees for mileage,
- Passively, automatically tracks business mileage, IRS compliant
- Swipe right after drive to indicate it was a business trip



# Trigger

- Use geofences, NFC, bluetooth, WiFi connections, etc to set auto-behaviors
  - Battery low -> turn off bluetooth + auto sync
  - Silence phone every morning when you get to work
  - Turn off mobile data when you connect to your home WiFi
  - Silence phone and set alarm once I get into bed
  - Use geofence for automatic foursquare checkin
  - Launch maps when you connect to your car's bluetooth network

The screenshot shows the 'Suggested Tasks' screen from the IFTTT mobile application. At the top, there is a header with a menu icon and the title 'Suggested Tasks'. Below the header, a blue banner contains the text: 'Use your phone's sensors to automatically change settings, launch apps or send messages.' and 'Get started with our examples below or create your own tasks.' There are two buttons at the bottom of the banner: 'Create your own' and 'OK, got it'. Below the banner, there are three examples of suggested tasks, each in its own card:

- Save time when driving**: This task uses a car icon and the description 'turns off wifi and opens Google'.
- Silence my phone while I sleep**: This task uses a cloud with a sleep icon and the description 'silences incoming notifications while you sleep'.
- Help me save battery when my battery gets low**: This task uses a battery icon and the description 'turns off wifi and turns down screen brightness'.

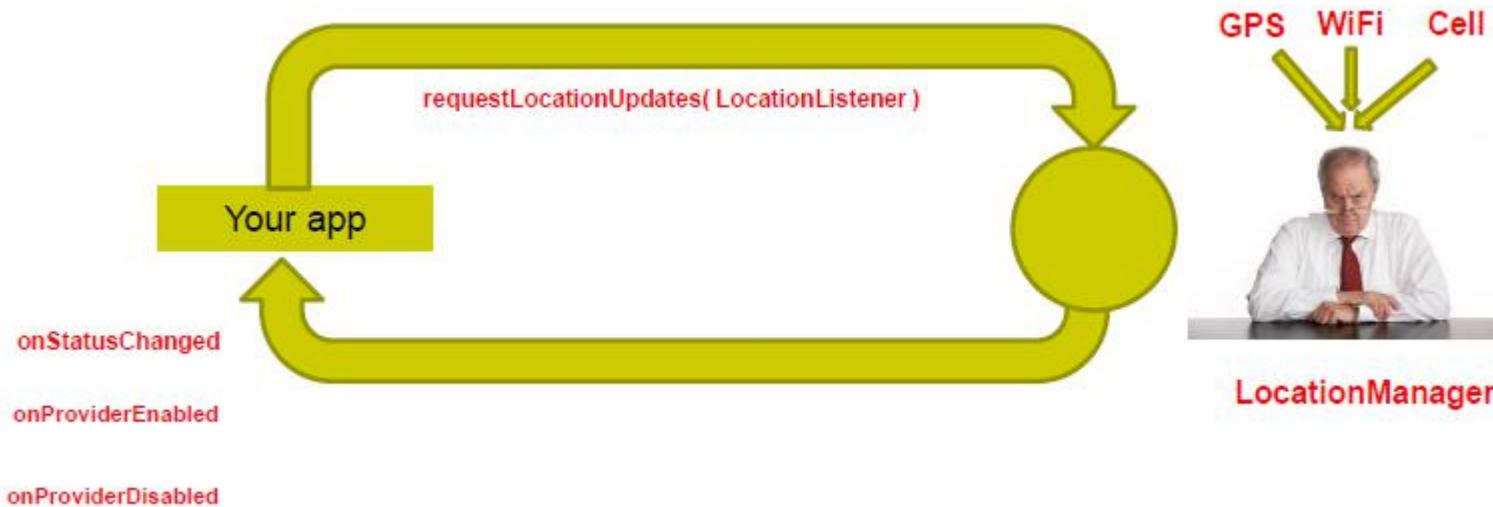
A green circular button with a '+' sign is located in the bottom right corner of the screen.

# **Location Sensing in Android Apps**

# The Basic Location APIs

- **LocationManager:**

- Android module receives location updates from GPS, WiFi, etc
- App registers/requests location updates from LocationManager



```
// Acquire a reference to the system Location Manager
LocationManager locationManager = (LocationManager) this.getSystemService(Context.LOCATION_SERVICE);

// Define a listener that responds to location updates
LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {
        // Called when a new location is found by the network location provider.
        makeUseOfNewLocation(location);
    }

    public void onStatusChanged(String provider, int status, Bundle extras) {}

    public void onProviderEnabled(String provider) {}

    public void onProviderDisabled(String provider) {}
};

// Register the listener with the Location Manager to receive location updates
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0, locationListener)
```

Create listener for location info

Callback methods called by Location manager (e.g. when location changes))

# Requesting User Permissions

- Need smartphone owner's permission to use their GPS

```
<manifest ... >
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    ...
    <!-- Needed only if your app targets Android 5.0 (API level 21) or higher. -->
    <uses-feature android:name="android.hardware.location.gps" />
    ...
</manifest>
```

- **ACCESS\_FINE\_LOCATION:** GPS
- **ACCESS\_COARSE\_LOCATION:** WiFi or cell towers

# Getting Cached Copy of Location (Fast)

- Getting current location may take a while
- Can choose to use location cached (possibly stale) from Location Manager

```
String locationProvider = LocationManager.NETWORK_PROVIDER;  
// Or use LocationManager.GPS_PROVIDER
```

```
Location lastKnownLocation = locationManager.getLastKnownLocation(locationProvider);
```

# Stopping Listening for Location Updates

- Location updates consume battery power
- Stop listening for location updates whenever you no longer need

```
// Remove the listener you previously added  
locationManager.removeUpdates(locationListener);
```

# **Location Representation in Android**

# Semantic Location

- GPS represents location as <longitude,latitude>
- **Semantic location** is better for reasoning about locations
- **E.g.** Street address (140 Park Avenue, Worcester, MA) or (building, floor, room)
- **Android supports:**
  - **Geocoding:** Convert addresses into longitude/latitude coordinates
  - **Reverse geocoding:** convert longitude/latitude coordinates into human readable address
- **Android Geocoding API:** access to **geocoding** and **reverse geocoding** services using HTTP requests

Latitude: 37.422005 Longitude: -122.084095

Address:  
1600 Amphitheatre Pkwy  
Mountain View, CA 94043  
Mountain View  
94043  
United States

# Google Places API Overview

- Access **high-quality photos** of a place
- Users can also add place information to the database
  - E.g. business owners can add their business as a place in Places database
  - Other apps can then retrieve info after moderation
- **On-device caching:** Can cache places data locally on device to avoid roundtrip delays on future requests



# Google Places

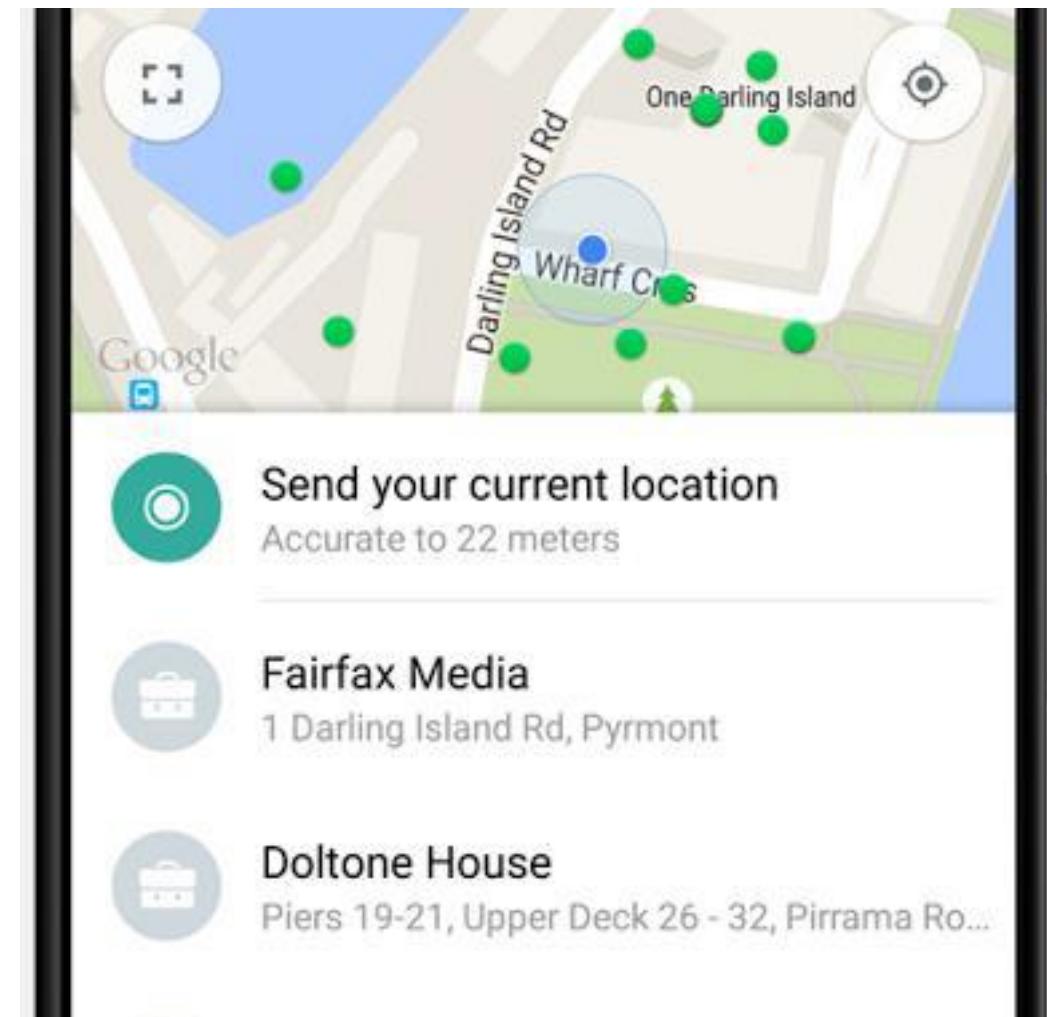
- **Place:** physical space that has a name (e.g. local businesses, points of interest, geographic locations)
  - E.g Logan airport, place type is **airport**
- **API:** Provides Contextual information about places near device.
  - **E.g:** name of place, address, geographical location, place ID, phone number, place type, website URL, etc.
- Compliments geographic-based services offered by Android location services

# Sample Place Types

accounting	hospital	city_hall	physiotherapist
airport	insurance_agency	clothing_store	place_of_worship (deprecated)
amusement_park	jewelry_store	convenience_store	plumber
aquarium	laundry	courthouse	police
art_gallery	lawyer	dentist	post_office
atm	library	department_store	real_estate_agency
bakery	liquor_store	doctor	restaurant
bank	local_government_office	electrician	roofing_contractor
bar	locksmith	electronics_store	rv_park
beauty_salon	lodging	embassy	school
bicycle_store	meal_delivery	establishment (deprecated)	shoe_store
book_store	meal_takeaway	finance (deprecated)	shopping_mall
bowling_alley	mosque	fire_station	spa
bus_station	movie_rental	florist	stadium
cafe	movie_theater	food (deprecated)	storage
campground	moving_company	funeral_home	store
car_dealer	museum	furniture_store	subway_station
car_rental	night_club	gas_station	synagogue
car_repair	painter	general_contractor (deprecated)	taxi_stand
car_wash	park	grocery_or_supermarket	train_station
		gym	transit_station
		hair_care	travel_agency
		hardware_store	university
		health (deprecated)	veterinary_care
		hindu_temple	zoo
		home_goods_store	

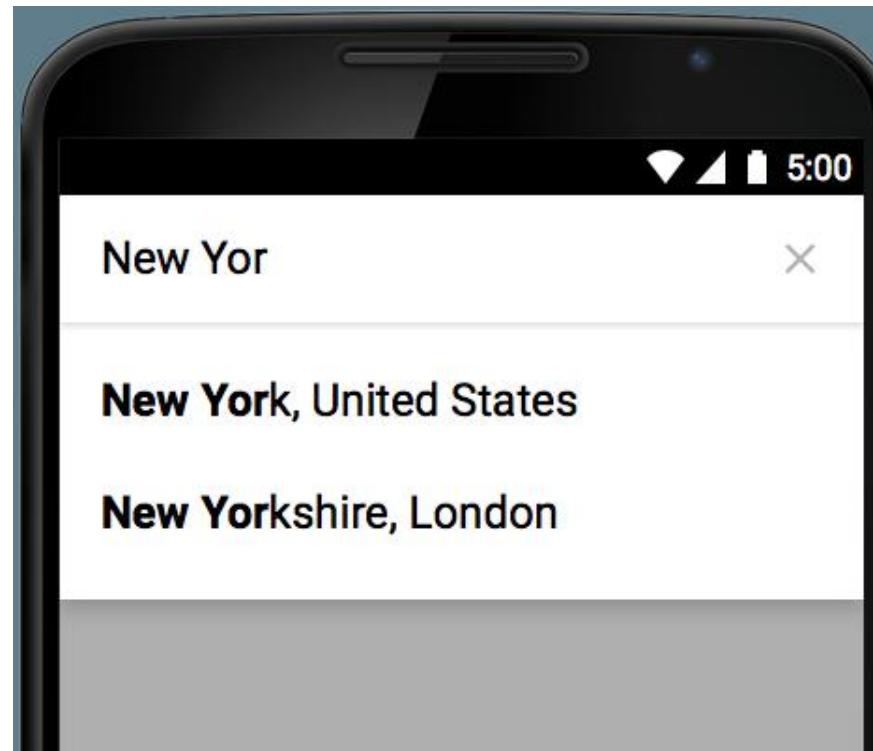
# Google Places API Overview

- **Use Place picker UI:** allows users select place from “possible place” on a map
- **Get current place:** place where device is last known to be located
  - Returns **list** of likely places + likelihood device is in that place



# Google Places API Overview

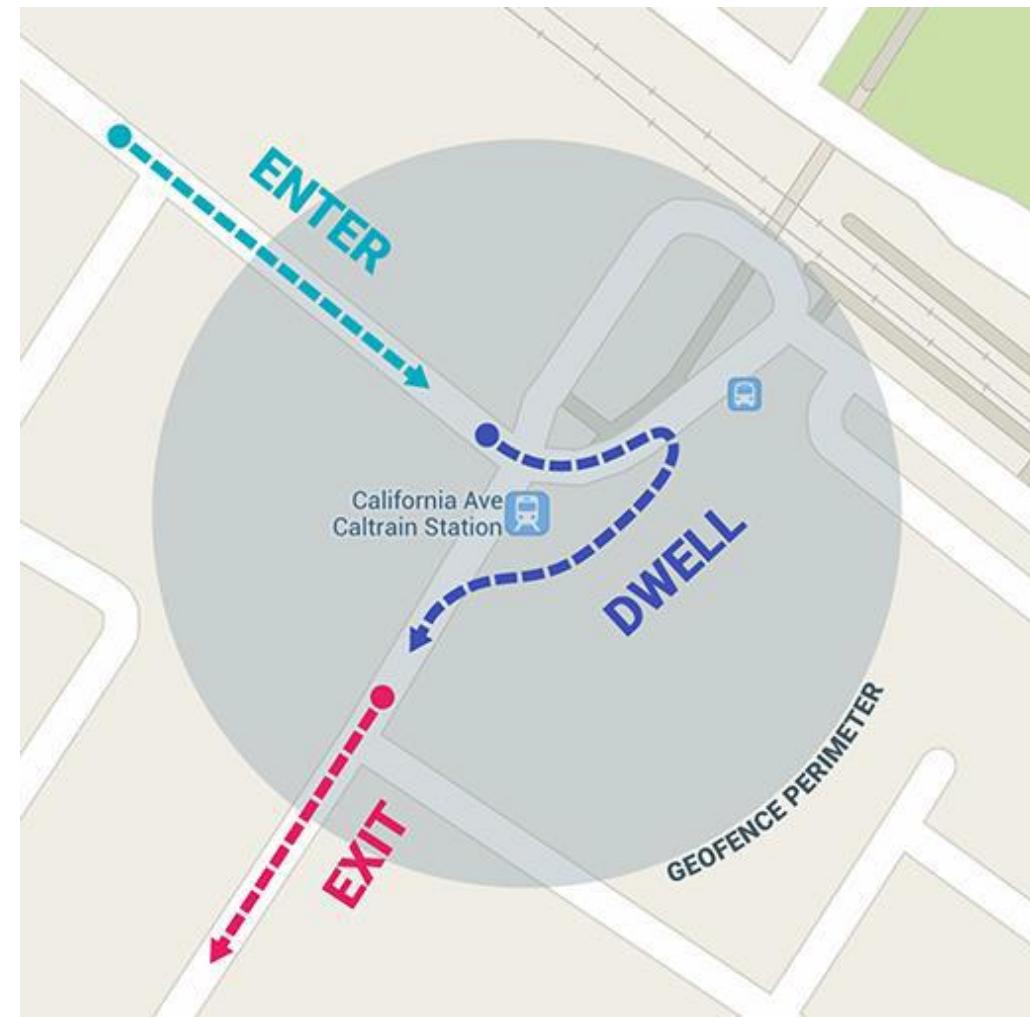
- **Autocomplete:** queries the location database as users type, suggests nearby places matching letters typed in



# **Other Useful Google Maps/Location APIs**

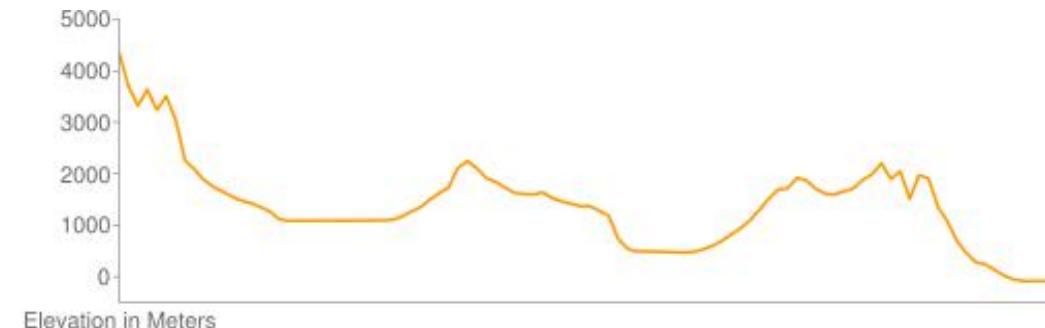
# GeoFencing

- **Geofence:** Sends alerts when user is within a certain radius to a location of interest
- Can be configured to send:
  - **ENTER** event when user enters circle
  - **EXIT** event when user exits circle
- Can also specify a duration or **DWELL** user must be in circle before triggering event



# Other Maps/Useful Location APIs

- **Maps Directions API:** calculates directions between locations (walking, driving) as well as public transport directions
- **Distance Matrix API:** Calculate travel time and distance for multiple destinations
- **Elevation API:** Query locations on earth for elevation information, calculate elevation changes along routes



# Other Useful Maps/Location APIs

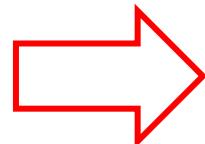
- **Roads API:**
  - sends set of GPS coordinates to road user was likely travelling on (best fit)
  - Returns posted speed limits for any road segment (premium plan)
- **Time Zone API:** request time zone for location on earth

# **GPS Clustering & Analytics**

# Determining Points of Interest from GPS Location Sequences

- **Points of Interest:** Places where a person spends lots of time (e.g. home, work, café, etc)
- **Given a sequence GPS <longitude, latitude> points,** how to infer points of interest
- **General steps:**
  - Pre-process sequence of GPS points (remove outliers, etc)
  - Cluster points
  - Convert to semantic location

LATITUDE	LONGITUDE
35.33032098	80.42152478
35.29244028	80.42382271
35.33021993	80.45339956
35.35529007	80.45222096



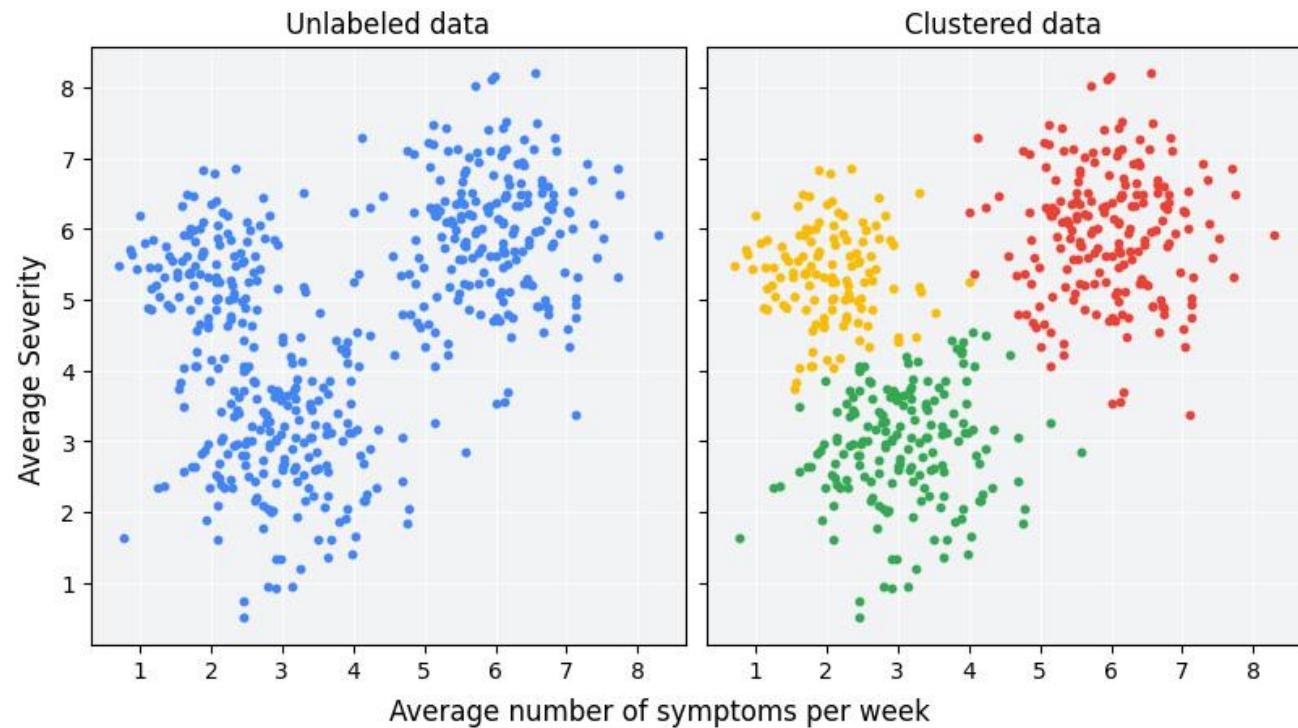
# Step 1: Pre-Processing GPS Points (Remove Noise and Outliers)

- **Remove low density points (few neighbors):**
  - i.e. places where little time was spent
  - E.g. radius of 20 meters, keep only clusters with at least 50 points
  - If GPS coordinates retrieved every minute, only considering places where you spent at least 50 minutes
- **Remove points with movement:**
  - GPS returns speed as well as <longitude, latitude> coordinates
  - If speed user is moving, discard that GPS point
- **Reduce data for stationary locations:**
  - When user is stationary at same location for long time, too many points generated (e.g. sitting at a chair)
  - Remove some points to speed up processing

# Step 2: Cluster GPS Points

- **Cluster Analysis:** Group points

- Two main clustering approaches
  - K-means clustering
  - DBSCAN



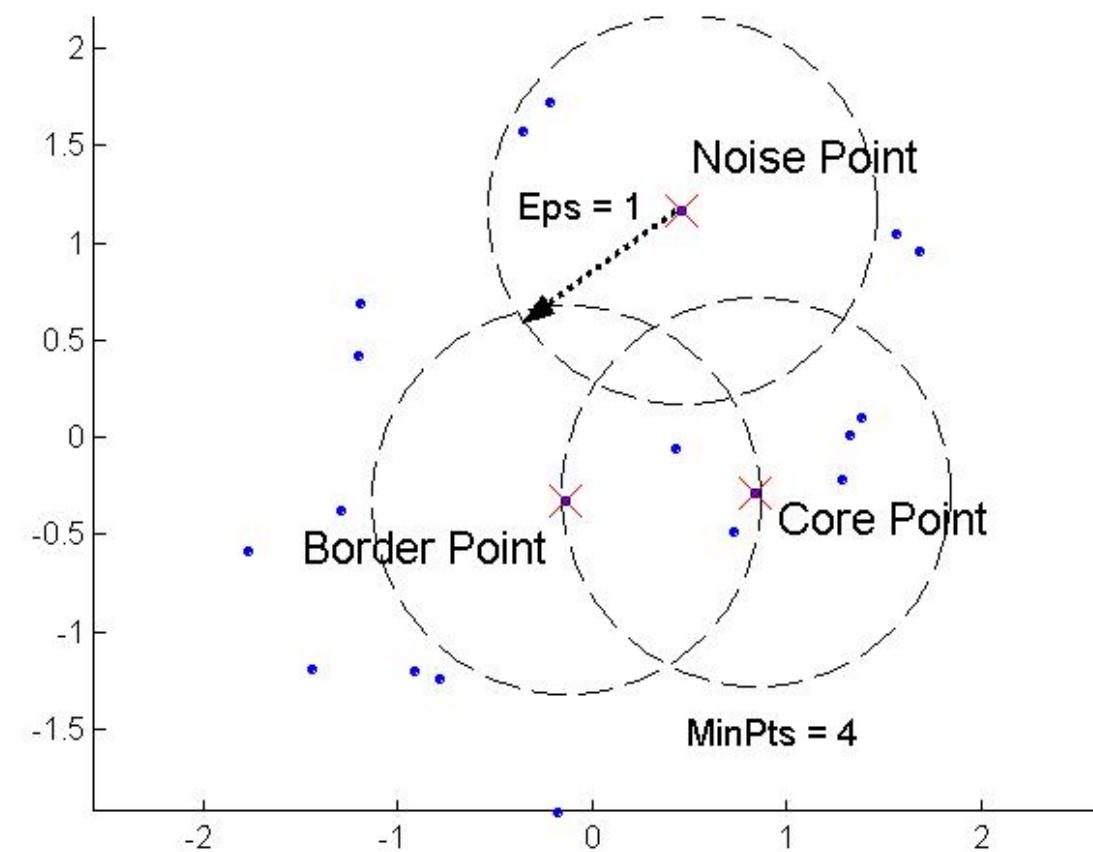
# K-Means Clustering

- Each cluster has a center point (centroid)
- Each point associated to cluster with closest centroid
- Number of clusters,  $K$ , must be specified

- 
- 1: Select  $K$  points as the initial centroids.
  - 2: **repeat**
  - 3:   Form  $K$  clusters by assigning all points to the closest centroid.
  - 4:   Recompute the centroid of each cluster.
  - 5: **until** The centroids don't change

# DBSCAN Clustering

- Density-based clustering
- **Density:** Number of points within specified radius ( $Eps$ )
- **Core points:** has  $> minPoints$  density
- **Border point:** has  $< minPoints$  density but within neighborhood of core point
- **Noise point:** not core point or border point



# DBSCAN Algorithm

- Eliminate noise points
- **Cluster remaining points**

```
current_cluster_label ← 1
for all core points do
    if the core point has no cluster label then
        current_cluster_label ← current_cluster_label + 1
        Label the current core point with cluster label current_cluster_label
    end if
    for all points in the  $Eps$ -neighborhood, except  $i^{th}$  the point itself do
        if the point does not have a cluster label then
            Label the point with cluster label current_cluster_label
        end if
    end for
end for
```

# Converting Clusters to Semantic Locations

- Can simply call reverse geocoding or Google Places on the centroid of the clusters
- Determining work? Cluster where user spends longest time most time (9-5pm)
- Determining home? Cluster where user spends most time 6pm –6am

# CSE 162 Mobile Computing

## Lecture 16: Context-Aware Computing and SQL

Hua Huang

# Context Awareness in Mobile Systems

# Ubicomp - Physical World Computing



*„In the 21st century the technology revolution will move into the everyday, the small and the invisible...“*

Mark Weiser (1952 – 1999), XEROX PARC

“Ubiquitous Computing enhances computer use by making computers **available throughout** the physical environment, while making them **effectively invisible** to the user”

- Ubicomp: a field on a physical world richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in everyday objects of lives and connected through a continuous network.
  - Mark Weiser in his last article in IBM Sys. Journal, 1999

# What are contexts

- A context represents the state or situation in the environment of a system that affects that system's (application specific) behaviour

# Types of Contexts, a systematic view

- Physical Contexts
  - What
  - Where
  - When
- Computing System Contexts
  - How
- User Context
  - Who

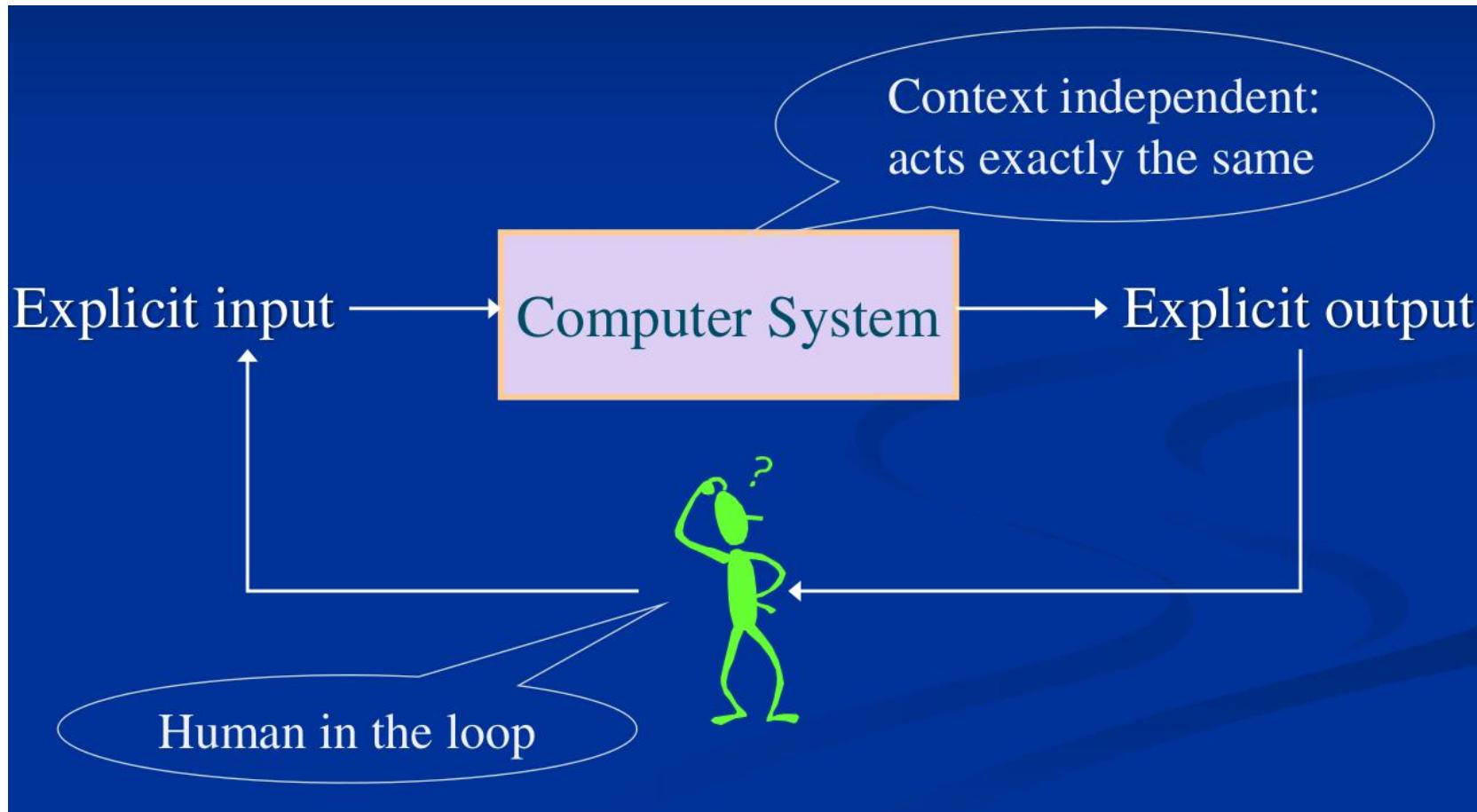
# Physical Contexts: What, Where, and When

- Physical environment or phenomena
  - Temperature, light intensity, or chemical
- Location
  - Absolute or logical locations
- Time
  - Absolute or logical time

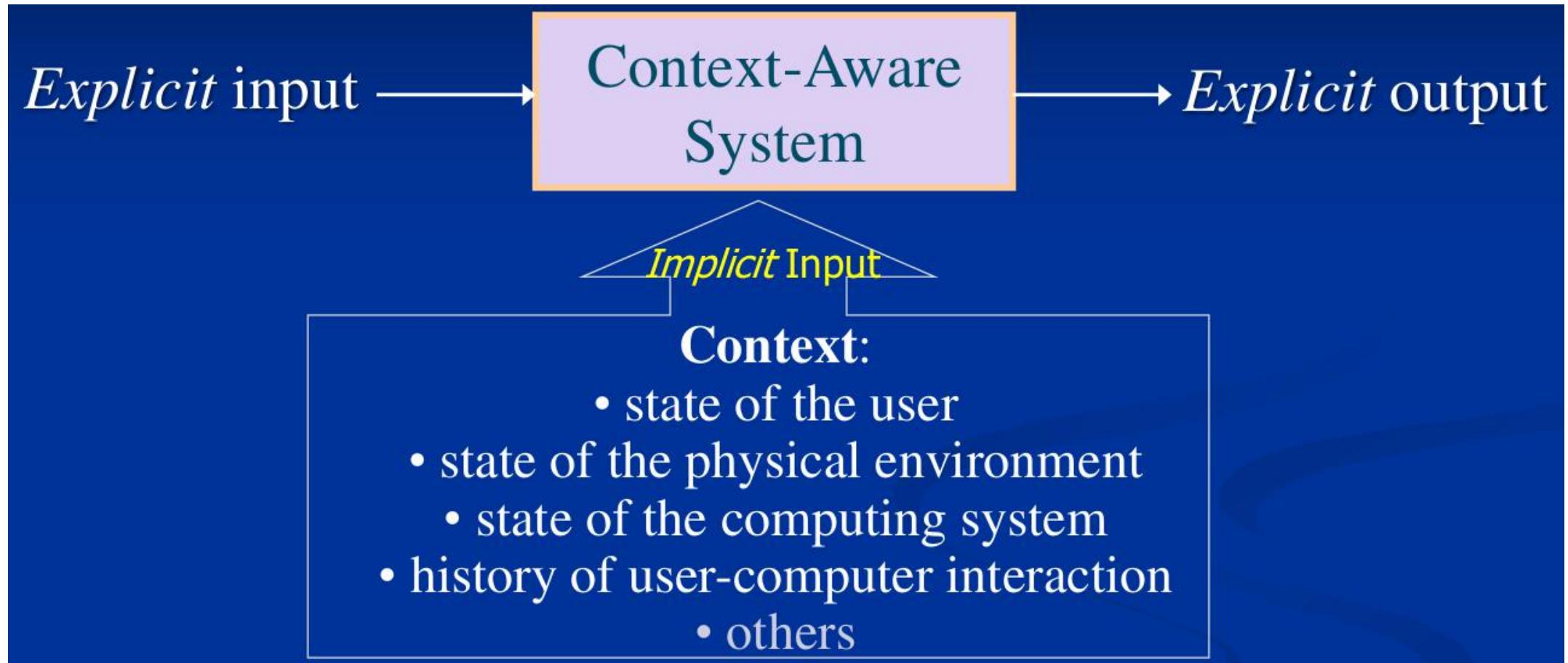
# What is Context Aware Computing

- General awareness of the surrounding environment in which the user is operating, located, or situated
  - Context examples: user's preferences, likings, dislikes, location, etc
- Context-awareness is considered to be one of the fundamental properties of ubiquitous computing systems and is a key property of smart environments.

# Traditional View of Computer Systems



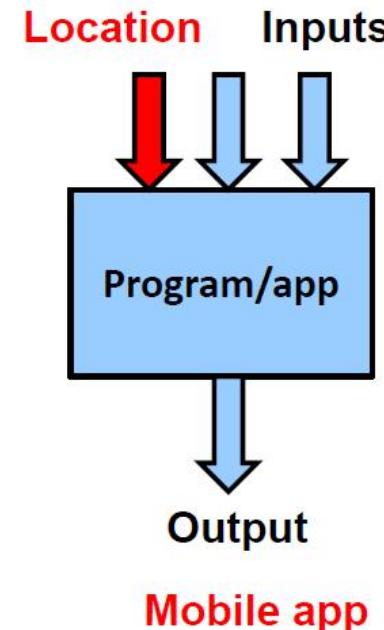
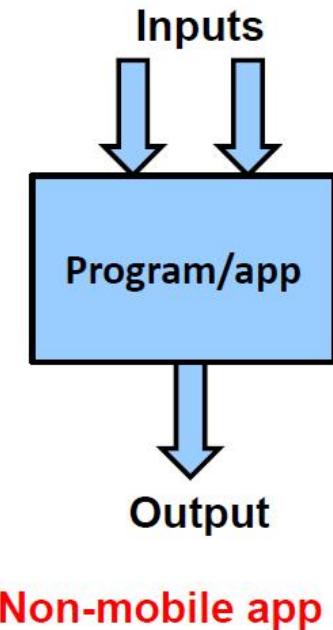
# Context as Implicit Input/Output



# Context-aware computing examples

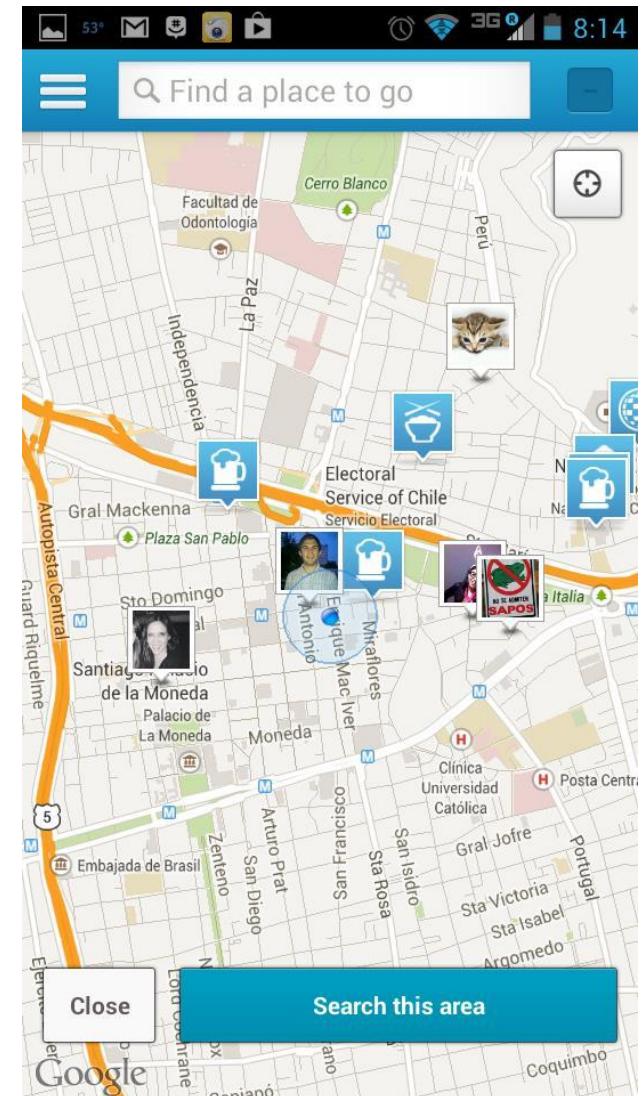
# Example: Location-aware computing

- **Location-aware:** Location must be one of app/program's inputs
- Different user location = different output (e.g. maps)



# Location-Aware Computing

- Examples:
  - Map of user's "current location"
  - Print to "closest" printer
  - Apps that find user's friends "closeby"
  - Reviews of "closeby" restaurants



# Reactive vs. Proactive LBS

- LBS can be either Reactive (“pull”) or Proactive (“push”)
- A Reactive LBS application is triggered by the user who queries the system in search of information based on the current location
- **Reactive LBS examples**
  - Finding restaurants or places of interest
  - Obtaining directions
  - Obtaining weather information
  - Sending emergency notifications to police, insurance companies, roadside assistance companies, etc.

# Reactive vs. Proactive LBS

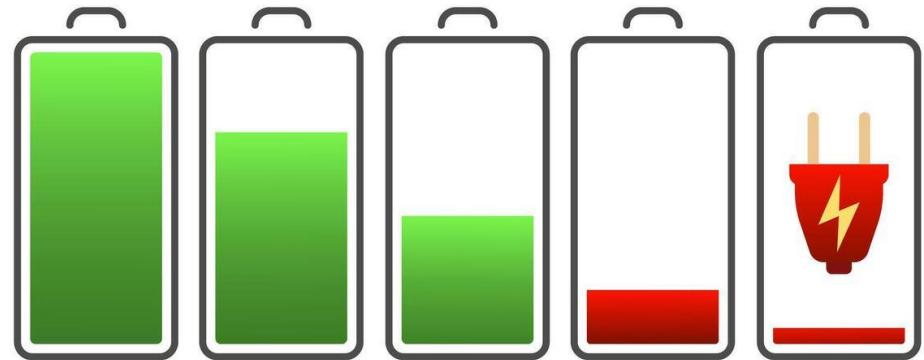
- In Proactive LBS applications, the system needs to continuously know where you are.
- Localization queries or actions are automatically triggered once a predefined set of conditions are met
- **Proactive LBS** examples
  - Geofencing, e.g., children outside predefined boundary
  - Fleet management
  - Real-time traffic congestion notifications
  - Real-time friend finding
  - Proximity-based actuation
  - Travel assistant device for riding public transportation, tourism, museum guided visits, etc

# Questions: Proactive or Reactive?

- Location-based advertisement
- Payment based on proximity (EZ pass, toll watch)
- Play a radio station in local area
- App shows grocery list when near Walmart

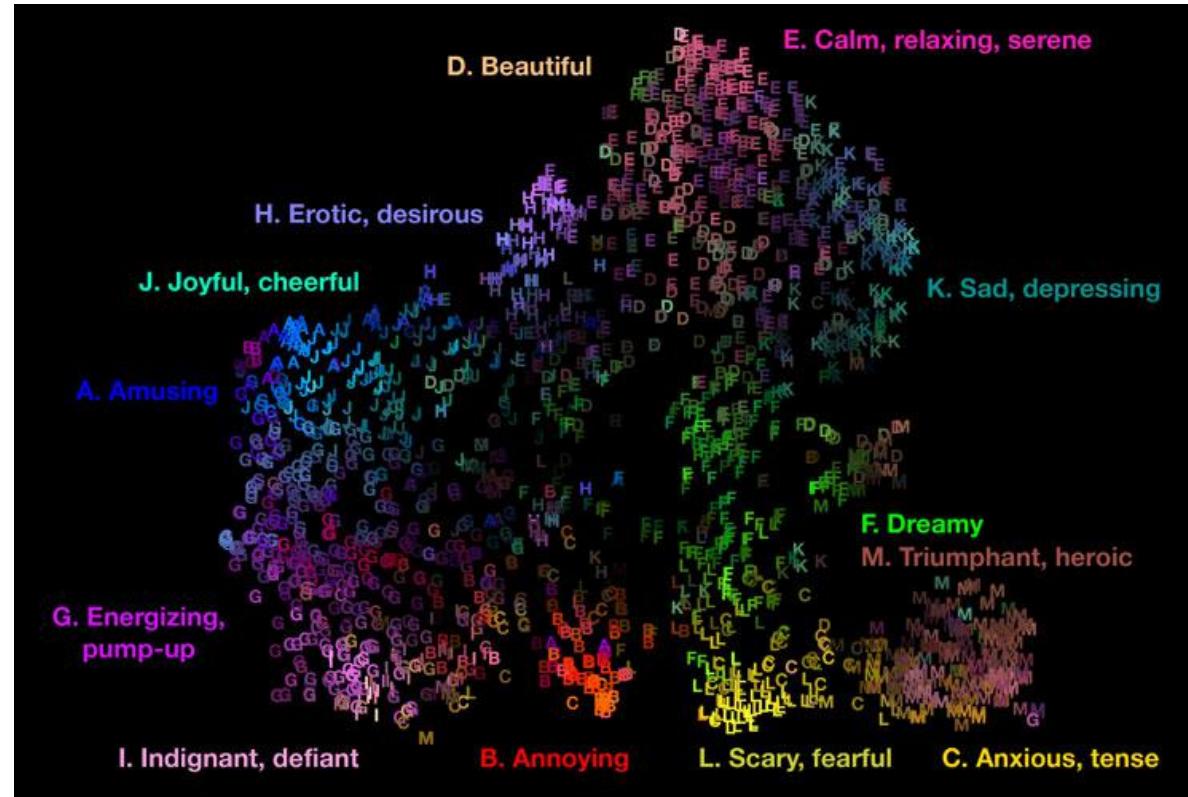
# System Context

- System Awareness
  - how any context is created and adapted over an information and computing infrastructure
  - E.g., Wireless connectivity, Charging status



# User context

- Consider a smart music player that adapts to user's mood



# User Context

- Personal Context
  - **Preference.** E.g., a referee at a sports activity may prefer to blow the whistle for minor versus major sports offences.
  - **Identity.** E.g., owner vs guest
  - **Activity and Task.** E.g., running vs standing
- Social Context
  - how the actions of someone may affect others
  - E.g., who blows the whistle? The referee, policeman, or spectator?

# Static versus Dynamic CA

- Static context
  - describes those aspects that are invariant
  - E.g., date of birth, User preference, home location, etc
- Dynamic context:
  - Information that changes change frequently
  - E.g., user activities, current locations, etc

# Three Levels of Interactions for Context-Aware Infrastructure

1. Personalization: users recognizes the context and decide how the system should behave.
2. Passive context awareness: the system provides the user with context information; however the system does not automatically change behavior based on this.
  - Instead, the user decides on the course of action based on the updated context information.
3. Active context awareness: the system recognizes the context and takes required actions.
  - It offloads the work from the user by taking active decisions.

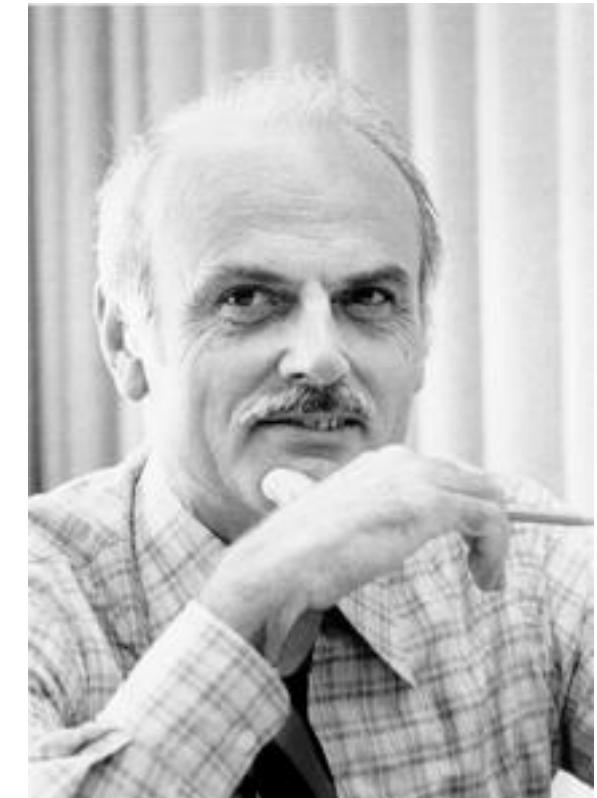
# Discussion: Which CA Type?

1. Personalization
  - A smart device uses location-based services to suggest dining locations, entertainment centers in the area, or even emergency services like hospitals and urgent care centers.
  - A thermostat that changes temperature based on schedule
  - The phone automatically disables notification when driving
  - A smart watch could automatically adjust daylight savings or time zone based on the location context.
2. Passive Context Awareness
3. Active Context Awareness

# Mobile Database

# Databases

- RDBMS
  - relational data base management system
- Relational databases introduced by
  - E. F. Codd
  - Turing Award Winner
- Relational Database
  - data stored in tables
  - relationships among data stored in tables
  - data can be accessed and viewed in different ways



# Example Database

- Wine and region tables

*Winery Table*

Winery ID	Winery name	Address	Region ID
1	Moss Brothers	Smith Rd.	3
2	Hardy Brothers	Jones St.	1
3	Penfolds	Arthurton Rd.	1
4	Lindemans	Smith Ave.	2
5	Orlando	Jones St.	1

*Region Table*

Region ID	Region name	State
1	Barossa Valley	South Australia
2	Yarra Valley	Victoria
3	Margaret River	Western Australia

# *Relational* Data

- Data in different tables can be related

*Winery Table*

Winery ID	Winery name	Address	Region ID
1	Moss Brothers	Smith Rd.	3
2	Hardy Brothers	Jones St.	1
3	Penfolds	Arthurton Rd.	1
4	Lindemans	Smith Ave.	2
5	Orlando	Jones St.	1

*Region Table*

Region ID	Region name	State
1	Barossa Valley	South Australia
2	Yarra Valley	Victoria
3	Margaret River	Western Australia

# Keys

- Each table has a key
- Column used to uniquely identify each row

KEYS

Winery Table			
Winery ID	Winery name	Address	Region ID
1	Moss Brothers	Smith Rd.	3
2	Hardy Brothers	Jones St.	1
3	Penfolds	Arthurton Rd.	1
4	Lindemans	Smith Ave.	2
5	Orlando	Jones St.	1

Region Table		
Region ID	Region name	State
1	Barossa Valley	South Australia
2	Yarra Valley	Victoria
3	Margaret River	Western Australia

# SQL and Databases

- SQL is the language used to manipulate and
- manage information in a relational database
- management system (RDBMS)
- SQL Commands:
  - **CREATE TABLE** - creates new database table
  - **ALTER TABLE** - alters a database table
  - **DROP TABLE** - deletes a database table
  - **CREATE INDEX** - creates an index (search key)
  - **DROP INDEX** - deletes an index

# SQL Commands

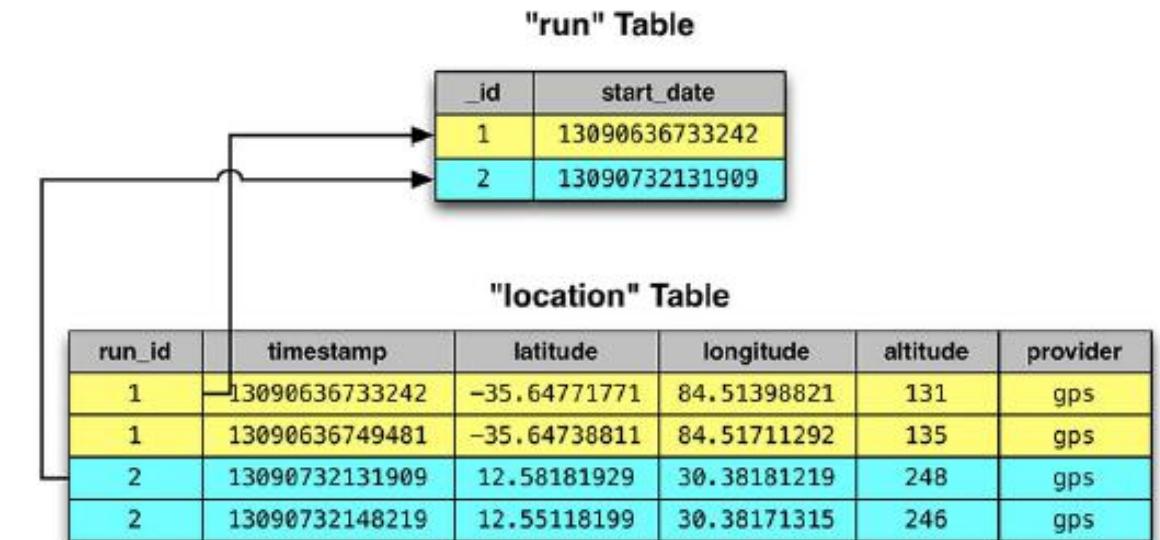
- **SELECT** - get data from a database table
- **UPDATE** - change data in a database table
- **DELETE** - remove data from a database table
- **INSERT INTO** - insert new data in a database table
- SQLite implements most, but not all of SQL
  - <http://www.sqlite.org/>

# Why Local Databases?

- Data can get large
- Example: a running tracking app
  - What would such an app be like?
- User may track their runs forever
- **Solution:** Store runTracker runs and locations in SQLite database
  - **SQLite** is open source relational database
  - **SQLiteOpenHelper** encapsulates database creation, opening and updating
  - In `runTracker`, create subclass of **SQLiteOpenHelper** called `RunDatabaseHelper`

# Use Local Databases in runTracker

- Create 1 table for each type of data
- Thus, we create 2 tables
  - **Run** table
  - **Location** table
- **Idea:** A run can have many locations visited



# Create RunDatabaseHelper

- **onCreate:** establish schema for newly created database
- **onUpgrade( ):** execute code to migrate to new version of schema
- Implement **insertRun(Run)** to write to database

```
public class RunDatabaseHelper extends SQLiteOpenHelper {  
    private static final String DB_NAME = "runs.sqlite";  
    private static final int VERSION = 1;  
  
    private static final String TABLE_RUN = "run";  
    private static final String COLUMN_RUN_START_DATE = "start_date";  
  
    public RunDatabaseHelper(Context context) {  
        super(context, DB_NAME, null, VERSION);  
    }  
  
    @Override  
    public void onCreate(SQLiteDatabase db) {  
        // Create the "run" table  
        db.execSQL("create table run (" +  
            "_id integer primary key autoincrement, start_date integer");  
        // Create the "location" table  
        db.execSQL("create table location (" +  
            " timestamp integer, latitude real, longitude real, altitude real," +  
            " provider varchar(100), run_id integer references run(_id));  
    }  
  
    @Override  
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
        // Implement schema changes and data massage here when upgrading  
    }  
  
    public long insertRun(Run run) {  
        ContentValues cv = new ContentValues();  
        cv.put(COLUMN_RUN_START_DATE, run.getStartDate().getTime());  
        return getWritableDatabase().insert(TABLE_RUN, null, cv);  
    }  
}
```

# Add ID to Run Class

- Add ID property to **Runs** class
- ID required to
  - Distinguish runs
  - Support querying runs

```
public class Run {  
    private long mId;  
    private Date mStartDate;  
  
    public Run() {  
        mId = -1;  
        mStartDate = new Date();  
    }  
  
    public long getId() {  
        return mId;  
    }  
  
    public void setId(long id) {  
        mId = id;  
    }  
  
    public Date getStartDate() {  
        return mStartDate;  
    }  
}
```

# Use RunManager to use the database

Use this method  
when a new run  
Is STARTED (When  
Start button  
is Pressed)

Use this method  
when a new run  
Is RESTARTED

Use this method when  
Run Is STOPPED  
(When STOP  
button is Pressed)

```
private void broadcastLocation(Location location) {  
    Intent broadcast = new Intent(ACTION_LOCATION);  
    broadcast.putExtra(LocationManager.KEY_LOCATION_CHANGED, location);  
    mAppContext.sendBroadcast(broadcast);  
}  
  
public Run startNewRun() {  
    // Insert a run into the db  
    Run run = insertRun();  
    // Start tracking the run  
    startTrackingRun(run);  
    return run;  
}  
  
public void startTrackingRun(Run run) {  
    // Keep the ID  
    mCurrentRunId = run.getId();  
    // Store it in shared preferences  
    mPrefs.edit().putLong(PREF_CURRENT_RUN_ID, mCurrentRunId).commit();  
    // Start location updates  
    startLocationUpdates();  
}  
  
public void stopRun() {  
    stopLocationUpdates();  
    mCurrentRunId = -1;  
    mPrefs.edit().remove(PREF_CURRENT_RUN_ID).commit();  
}  
  
private Run insertRun() {  
    Run run = new Run();  
    run.setId(mHelper.insertRun(run));  
    return run;  
}
```

- Example Usage

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_run, container, false);

    ...

    mStartButton = (Button)view.findViewById(R.id.run_startButton);
    mStartButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mRunManager.startLocationUpdates();
            mRun = new Run();
            mRun = mRunManager.startNewRun();
            updateUI();
        }
    });

    mStopButton = (Button)view.findViewById(R.id.run_stopButton);
    mStopButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mRunManager.stopLocationUpdates();
            mRunManager.stopRun();
            updateUI();
        }
    });

    updateUI();

    return view;
}
```

# Inserting Locations into Database

- Similar to inserting runs, need to insert locations when **LocationManager** gives updates
- Add **insertLocation** method

```
public class RunDatabaseHelper extends SQLiteOpenHelper {  
    private static final String DB_NAME = "runs.sqlite";  
    private static final int VERSION = 1;  
  
    private static final String TABLE_RUN = "run";  
    private static final String COLUMN_RUN_START_DATE = "start_date";  
  
    private static final String TABLE_LOCATION = "location";  
    private static final String COLUMN_LOCATION_LATITUDE = "latitude";  
    private static final String COLUMN_LOCATION_LONGITUDE = "longitude";  
    private static final String COLUMN_LOCATION_ALTITUDE = "altitude";  
    private static final String COLUMN_LOCATION_TIMESTAMP = "timestamp";  
    private static final String COLUMN_LOCATION_PROVIDER = "provider";  
    private static final String COLUMN_LOCATION_RUN_ID = "run_id";  
  
    ...  
  
    public long insertLocation(long runId, Location location) {  
        ContentValues cv = new ContentValues();  
        cv.put(COLUMN_LOCATION_LATITUDE, location.getLatitude());  
        cv.put(COLUMN_LOCATION_LONGITUDE, location.getLongitude());  
        cv.put(COLUMN_LOCATION_ALTITUDE, location.getAltitude());  
        cv.put(COLUMN_LOCATION_TIMESTAMP, location.getTime());  
        cv.put(COLUMN_LOCATION_PROVIDER, location.getProvider());  
        cv.put(COLUMN_LOCATION_RUN_ID, runId);  
        return getWritableDatabase().insert(TABLE_LOCATION, null, cv);  
    }  
}
```

# Continuously Handle Location Updates

- System will continuously give updates
- Need to receive location intents whether app is visible or not
- Implement **dedicated Location Receiver** to insert location
- Inserts location into run whenever new location is received

```
public class TrackingLocationReceiver extends LocationReceiver {  
    @Override  
    protected void onLocationReceived(Context c, Location loc) {  
        RunManager.get(c).insertLocation(loc);  
    }  
}
```

# Alternatives to sqlite

- SQLite is low level ("Down in the weeds")
- Various alternatives to work higher up the food chain
- Object Relational Mappers - ORM
- Higher level wrappers for dealing with SQL commands and sqlite databases
- Many ORMs exist

# Mobile Crowd Sensing

CSE 162 – Mobile Computing

Hua Huang

Department of Computer Science and Engineering

University of California, Merced

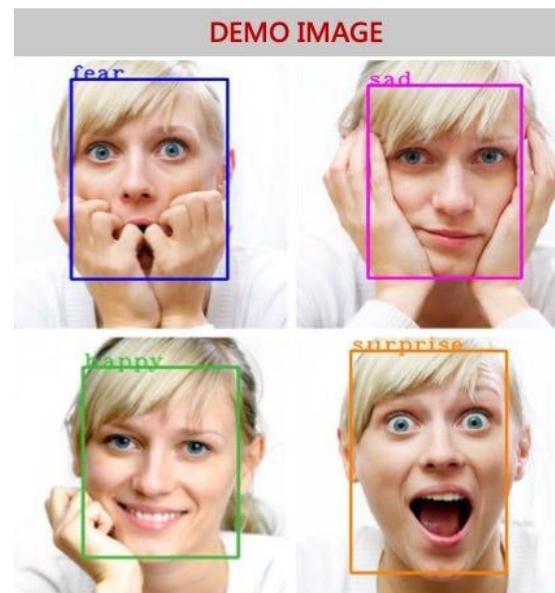
# Recap: mobile personal sensing

- **Personal sensing:** phenomena for an individual.
  - Emotions from images or voices
  - Hand movement tracking
  - Activity recognition (smoking)
  - Localization
  - etc



Sensors on iPhone 4:

- Camera
- Audio
- Accelerometer
- GPS
- Gyroscope
- Compass
- Proximity
- Ambient light



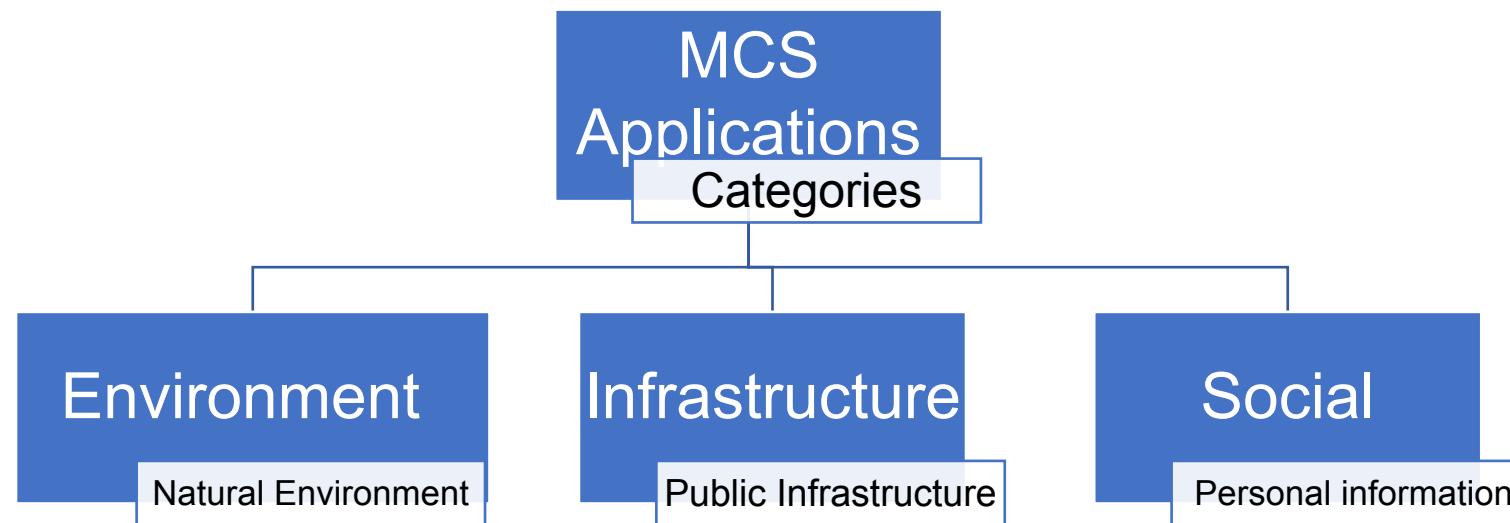
# Introduction

- Leveraging on crowd
- Data, services, ideas, contents, skills, money, ... coming from crowds
- Crowdsourcing = Crowd + outsourcing

# Introduction

- Pertains to the monitoring of large-scale phenomena that cannot be easily measured by a single individual.
- For example, intelligent transportation systems may require traffic congestion monitoring and air pollution level monitoring.
  - These phenomena can be measured accurately only when many individuals provide speed and air quality information from their daily commutes

# Mobile Crowd Sensing Applications



# Environmental MCS applications

- Environmental MCS applications,
  - pollution levels in a city
  - water levels in creeks
  - monitoring wildlife habitats

- Example: The **CommonSense** system uses specialized handheld air quality sensing devices that communicate with mobile phones (using Bluetooth) to measure various air pollutants (e.g. CO<sub>2</sub>, NO<sub>x</sub>).
  - These devices when deployed across a large population, collectively measure the air quality of a community or a large area.

# Infrastructure Applications

- Infrastructure Applications involve the measurement of large scale phenomena related to public infrastructure.
  - Traffic congestion
  - road conditions
  - parking availability
  - outages of public works (e.g. malfunctioning fire hydrants, broken traffic lights)
  - real-time transit tracking.

# Infrastructure MCS Example

- **CarTel** utilizes specialized devices installed in cars to measure the location and speed of cars and transmit the measured values using public WiFi hotspots to a central server
  - This central server can then be queried to provide information such as least delay routes or traffic hotspots
- **Nericell** utilizes individuals' mobile phones to not only determine average speed or traffic delays, but also detect honking levels (especially in countries like India where honking is common) and potholes on roads.

# Social applications

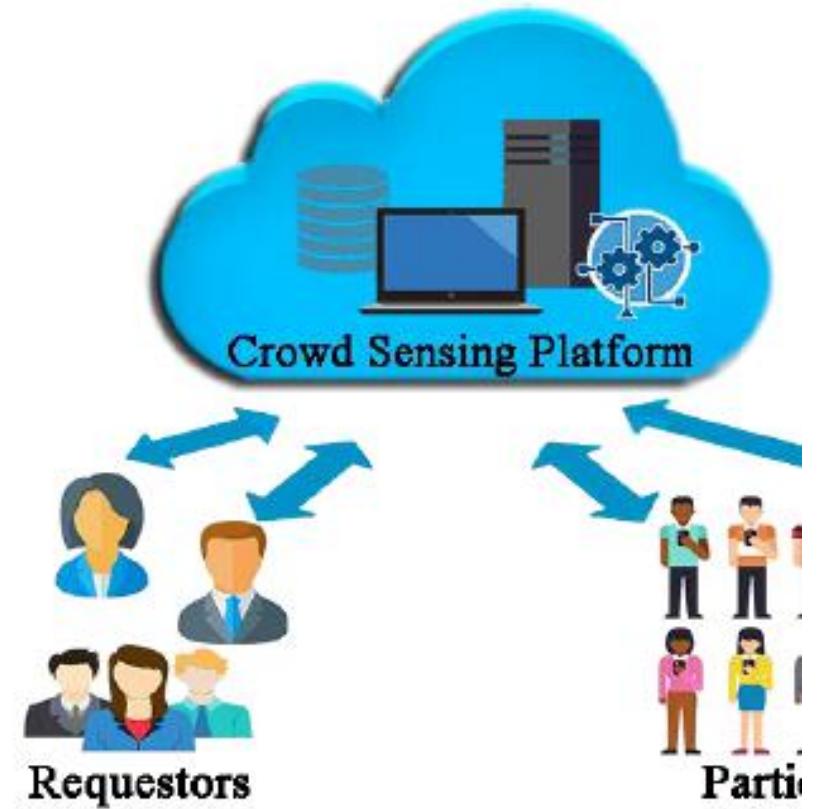
- Social applications involves individuals sharing sensed information amongst themselves.
  - Individuals can share their exercise data (e.g. how much time one exercises in a day) and compare their exercise levels with the rest of the community.
  - **BikeNet**, individuals measure location and bike route quality (e.g. CO<sub>2</sub> content on route, bumpiness of ride) and aggregate the data to obtain “most” bikeable routes
  - **DietSense**, individuals take pictures of what they eat and share it within a community to compare their eating habits.

# MCS: Unique Characteristics

- Compared to traditional mote-class sensor networks, mobile crowdsensing has a number of unique characteristics that bring both new opportunities and problems.
  1. today's mobile devices have significantly more computing, communication and storage resources than mote-class sensors, and they are usually equipped with multimodality sensing capabilities.
  2. millions of mobile devices are already “deployed in the field”: people carry these devices wherever they go and whatever they do. By leveraging these devices, we could potentially build large scale sensing applications efficiently
    - For example, instead of installing road-side cameras and loop detectors, we can collect traffic data and detect congestion levels using smartphones carried by drivers.

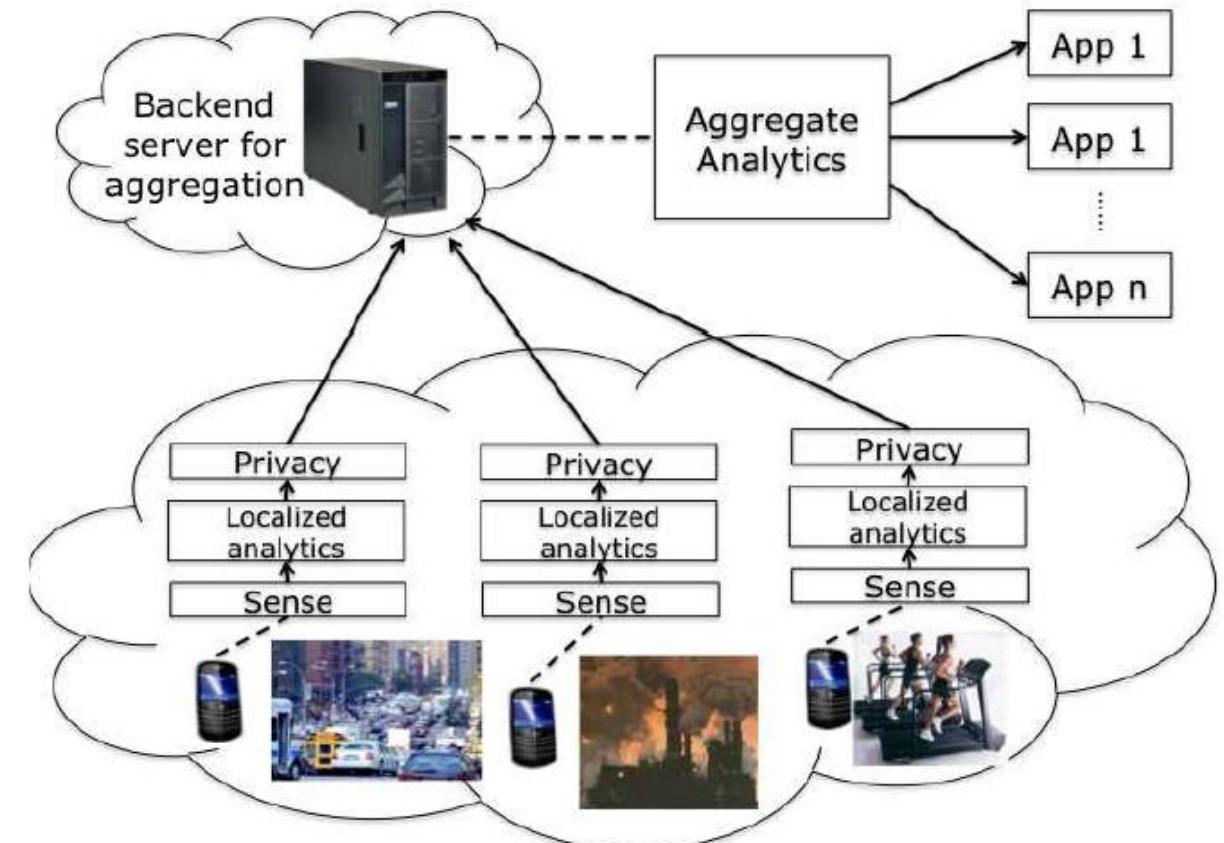
# MCS Architecture

- The general architectural design of an MCS system comprises the following main entities:
  - **Requestors**, initiate targeted data collection process by publishing a task pertinent to their interests to the crowd sensing platform and analyze the collected data from the Workers.
  - **Workers** are assigned several tasks pertinent to data collection, thus are the main source of information and play a major role in data collection
  - **Crowd sensing Platform** is the main communication link between Requestors and Workers. The platform stores, processes and analyzes data provided by Workers and the Requestors.



# MCS Workflow

- Raw sensing data is collected on devices.
- Local analytics process it to produce consumable data for applications.
- After privacy preservation, the data is sent to the backend.
- Aggregate analytics will further process it for different applications.



# Participatory and opportunistic sensing

# Participatory and opportunistic sensing

- Participatory sensing requires the **active involvement** of individuals to contribute sensor data (e.g. taking a picture, reporting a road closure) related to a large-scale phenomena.
- Opportunistic sensing is more autonomous and user involvement is minimal (e.g. continuous location sampling without the explicit action from the user).

## Participatory Sensing

## Opportunistic Sensing

Users actively engage in the data collection activity.

Users manually determine how, when, what, where to sample.

Can avoid phone context issues.

Higher burdens or costs.

Takes random sample which is application defined.

Easy to gather large amount data in small time.

Can't avoid phone context issues.

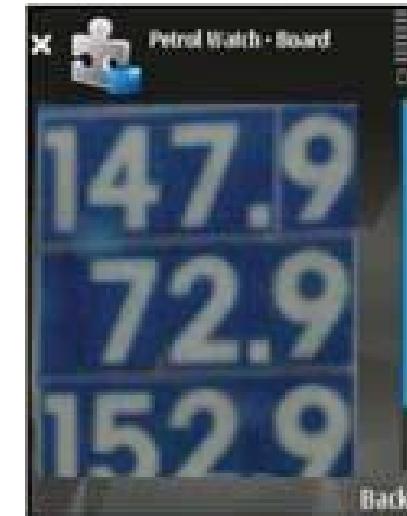
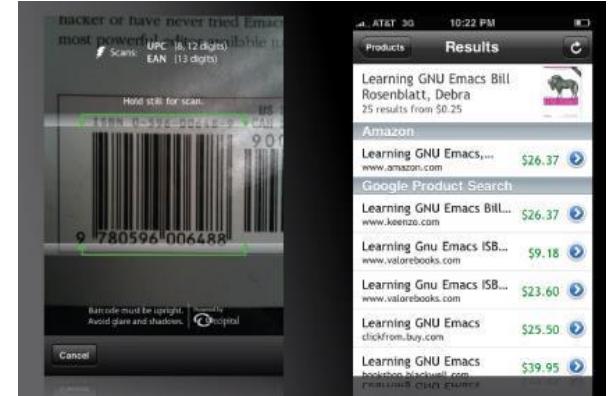
Lower burdens or costs if contextual problems are handled.

Filtering Data by Handling Privacy Issues & Localization.

Dataset is ready for research

# Participatory crowdsensing examples

- **LiveCompare**
  - User-created database of UPCs and prices
  - GPS and cell tower info used to find nearby stores
- **PetroWatch**
  - Uses phone to photograph gasoline price
  - Uses GPS to know when gas station is near



## Opportunistic crowdsensing examples



- **Pothole Monitor**
  - Combines GPS and accelerometer

# The Challenges of MCS...

*Incentivize Users*

*Resource Limitations*

*Privacy*

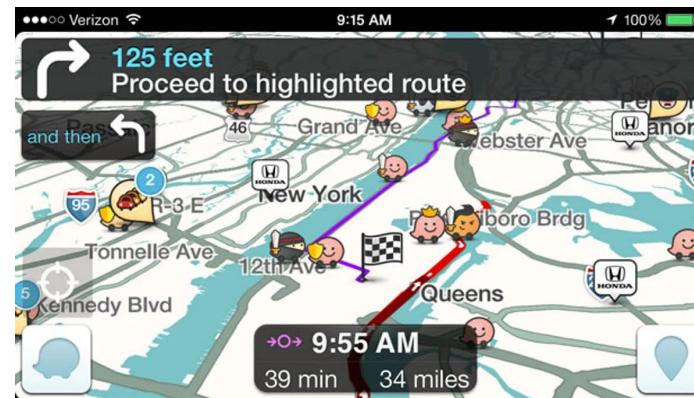
*Aggregate Analytics*

# Challenge: Incentivize users

- Three basic approaches:
  - To make **entertainment an incentive**, crowd sensing tasks are turned into sensing games, such that users can contribute computation or sensing abilities of their mobile devices when they play these games. This paradigm makes users feel enjoyable when they perform tasks
  - The rationality of taking **service as an incentive** roots in the mutual-benefit principle. Service consumers are also service providers. In other words, if a user wants to benefit from the service provided by the system, she also has to contribute to the system.
  - The last category is based on **monetary incentives**. In this case, the system has to pay a certain amount of money to motivate potential participants, such that the participants can use their resources, usually smartphone sensors, to complete the distributed tasks.

# examples: Incentivize users

- Entertainment as an incentive
  - Pokeman Go
- Service as incentive
  - Traffic monitoring
- Monetary incentive
  - measure web content usage by paying users money based on page visits to a site.



X Google Opinion Rewards

Question 1 of 3 or fewer:

What is your favorite type of dessert?

Cakes

Pastries

Ice creams

Candies & Chocolates

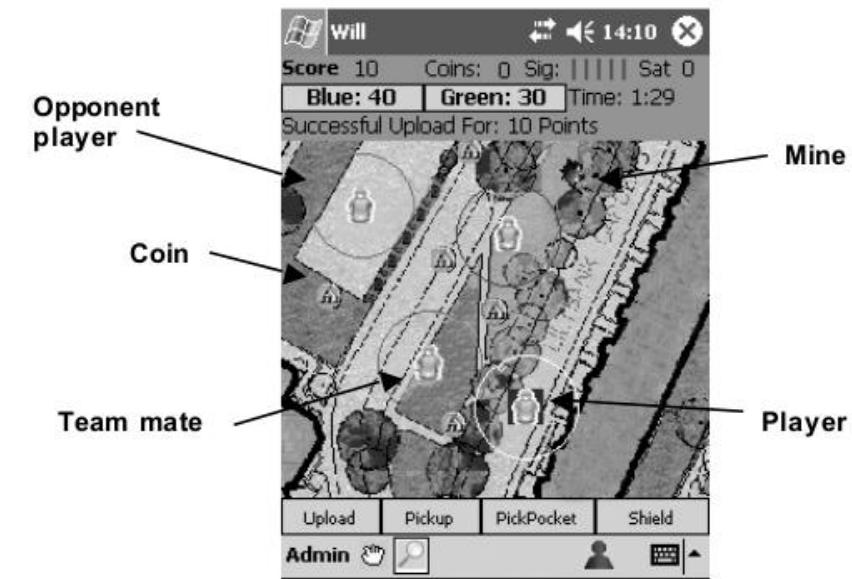
Cookies

None of the above

[NEXT](#)

# Entertainment as an incentive

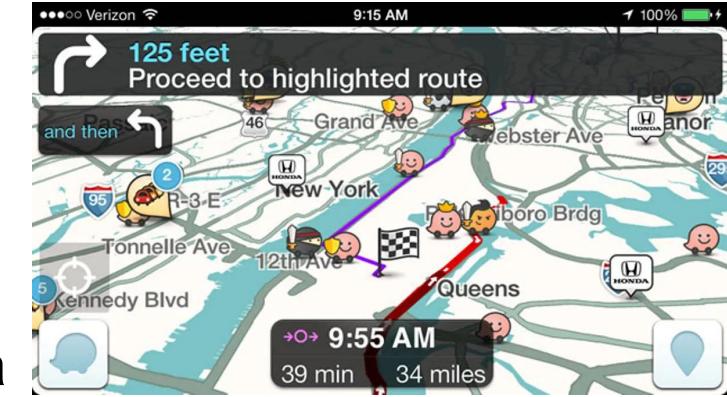
- Example: mobile game **Treasure** to build WiFi coverage maps of a given game area.
- Players carry mobile devices with GPS and WiFi. They need to pick up virtual coins scattered over the game area and then upload the coins to a server to gain game points. Better network connections give larger probabilities of uploading the collected coins successfully.
- Therefore players are motivated to find areas with stronger WiFi coverage.



**Fig. 1.** The game interface. The map shows players' locations, along with coins that are often positioned outside the network. A semi-transparent map layer of green, yellow and red squares builds up as players move around, revealing network coverage, however for printing reasons these have not been included in this figure.

# Service as incentive

- For some mobile crowd sensing systems, a participant (e.g., a smartphone user) may have two roles concurrently: a contributor and a consumer.
- Traffic monitoring is a typical example.
  - A participant acts as a contributor when she travels on a bus or car if she collects traffic data (e.g., GPS traces) to a service provider via networks such as WiFi, GPRS, and 3G.
  - The service provider then processes the data crowdsourced from a large amount of users and provides a real-time traffic information service, such as querying on traffic jams and bus crowdedness
  - In such sensing applications, to attract more users to contribute sensed data such that the system can provide services of good quality, the service provider will usually grant a participant some service quota, which determines how much service that user can receive.
  - In essence, this strategy is an exchange of contribution and consumption for each participant.



# Monetary incentive

- Paying for sensed data in crowd sensing tasks is the most intuitive incentive, as it has made sensed data become goods in a free market. Any user who would like to make some money can sell her sensed data for crowd sensing tasks.



The image shows a screenshot of a Google Opinion Rewards survey. At the top right, there is a close button (X) and the text "Google Opinion Rewards". Below that, a question is displayed: "Question 1 of 3 or fewer: What is your favorite type of dessert?". A list of six options follows, each preceded by a radio button:

- Cakes
- Pastries
- Ice creams
- Candies & Chocolates
- Cookies
- None of the above

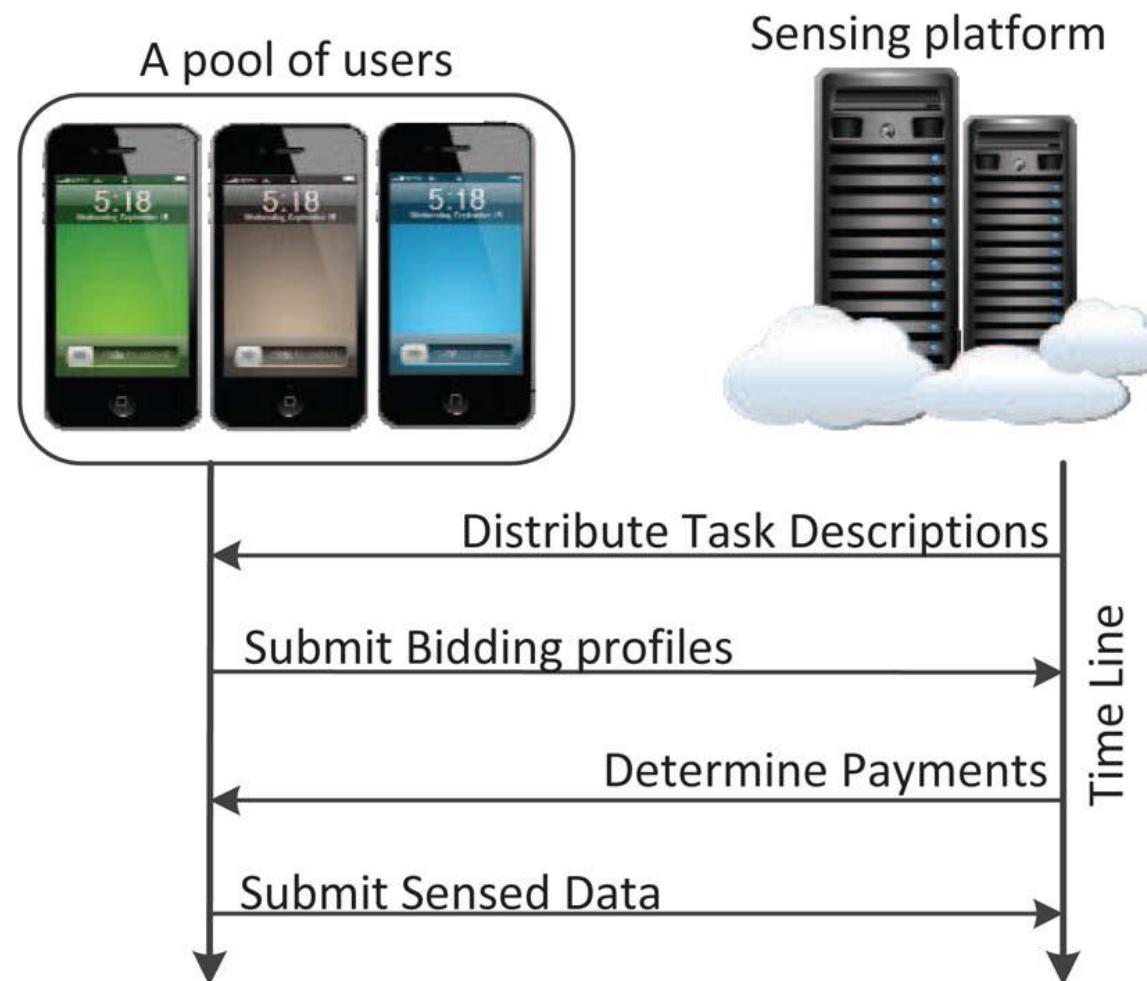
At the bottom right of the screenshot area, there is a "NEXT" button.

# Reverse Auction Mechanism

- The system involves two participating roles: a platform that distributes sensing tasks and the mobile phone users who constitute potential labor force.
- The objective is to design a task assignment and payment negotiation scheme, which ensures that both the platform and users are satisfied

# Reverse Auction Mechanism

- The platform initiates one round of task distribution by sending task descriptions.
- A set of  $n$  users are assumed to be interested in the sensing tasks after receiving the requests.
- If users participate in sensing tasks, they will consume multiple resources, including computation, communication, and energy. Thus it is rational for a user to expect certain profit based on her cost and sensing plan (e.g., sensing time).
- A participating user then submits a bidding profile (including a bidding price and a sensing plan) to the platform.
- After collecting all bidding profiles from the  $n$  users, the platform selects a subset of them and determines the payments for them.
- Finally, the selected users perform the assigned tasks and upload the sensed data to the platform.



# The Challenges of MCS...

*Incentivize Users*

*Resource Limitations*

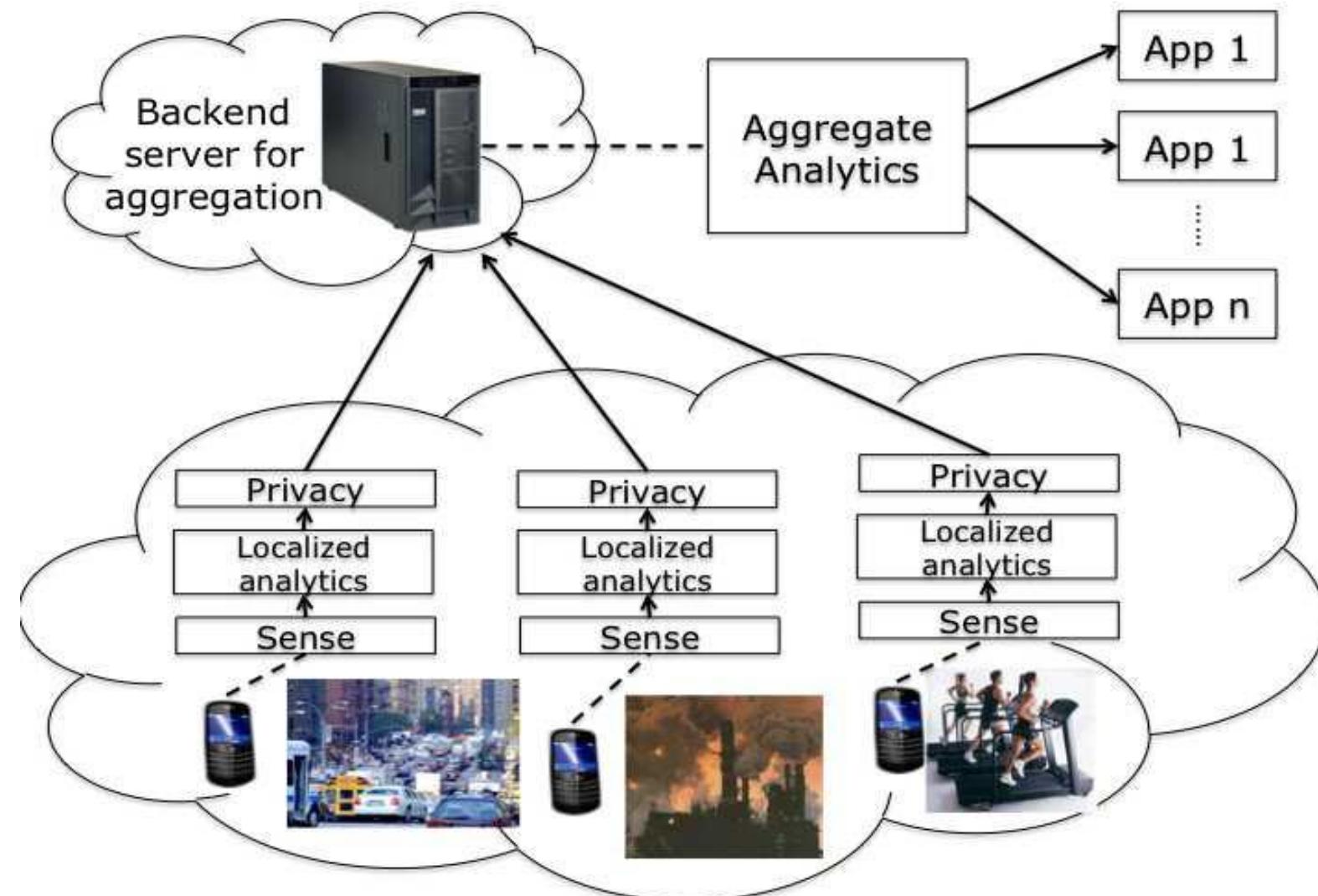
*Privacy*

*Aggregate Analytics*

# Challenge: Resource limitations

- How to run the MCS application without significantly affecting the normal device operation?
- How to avoid overloading the server?
  - Single point failure problem

# LOCALIZED ANALYTICS



# LOCALIZED ANALYTICS

- Upload less amount and appropriately processed data
  - *Data mediation*, such as filtering of outliers, elimination of noise, or makeup for data gaps.
- Reduces the amount of processing that the backend has to perform

# MCS: Localized Analytics

- Benefits
  - Transmitting data summary consumes lesser energy and bandwidth than transmitting the raw sensor readings.
  - Do the processing locally saves time.
    - Transmitting raw sensor data on intermittently connected channels can be time consuming.
    - *Example: Context inference*
      - *Transportation mode*
      - Kinetic modes of humans
      - Social settings

# The Challenges of MCS...

*Incentivize Users*

*Resource Limitations*

*Privacy*

*Aggregate Analytics*

# Privacy

- Potentially collect sensitive sensor data pertaining to individuals
  - the routes they take in daily commutes
  - their home and work locations

# Privacy Preservation techniques

- Anonymization
- Cryptographic techniques
- Data Perturbation

# Privacy Preserving Approach: Anonymization

- Remove any identifying information from the sensor data before sharing it with a third party.
  - Removes any identifying information from the sensor data before sharing it
  - Drawback: anonymized GPS (or location) sensor measurements can still be used to infer the frequently visited locations of an individual and derive their personal details.

# Privacy Preserving Approach: cryptographic techniques

- Cryptographic techniques are used to transform the data to preserve the privacy of an individual.
  - Example: privacy leakage from text auto-completion
  - Example: suggested location of interest
  - are compute intensive and are not scalable because they require the generation and maintenance of multiple keys, which also leads to higher energy consumption.



Conformal encryption.  
Talk about it.

# Privacy Preserving Approach: Data Perturbation

- Add noise to sensor data before sharing it with the community to preserve privacy of an individual
  - Requirement on the noise: Preserve the privacy of an individual, while not influencing the statistics accuracy
- For example, in a weight watchers application, it is important to compute the average weight of the population.
  - Add a random number to the weight
  - When these values are averaged, the randomized component vanishes and the average weight of the community remain accurate

# The Challenges of MCS...

*Incentivize Users*

*Resource Limitations*

*Privacy*

*Aggregate Analytics*

# Challenge: Aggregate Analytics

- MCS applications rely on analyzing the data from a collection of mobile devices, identifying spatio-temporal patterns.
- There are two possible approaches for data mining.
  - Offline approach
  - Online approach
- Consider the restaurant recommendation app based on user ratings

# Offline Aggregate Analytics

- One is a traditional approach where data is stored in a database first and then one can apply various mining algorithms against the database to detect patterns.
- However, if the application requires fast detection of patterns, this approach will not work.
- e.g., how to quickly recommend the best route considering traffic?

# Online Aggregate Analytics

- Stream data mining algorithms take as input continuous data streams and identify patterns, without the need to first store the data.
- These algorithms are less accurate due to incomplete data.
- e.g., do we recommend a lunch restaurant based on the breakfast traffic data?

# Camera, Audio, Video and Sound

CSE 162 – Mobile Computing

Hua Huang

Department of Computer Science and Engineering  
University of California, Merced

# An interesting camera app: Word Lens

## Feature of Google Translate

- Word Lens: translates text/signs in foreign Language in real time
- Example use case: tourist can understand signs, restaurant menus
- Uses Optical Character Recognition technology
- Google bought company in 2014, now part of Google Translate



[\[ Original Word Lens App \]](#)



[\[ Word Lens as part of Google Translate \]](#)

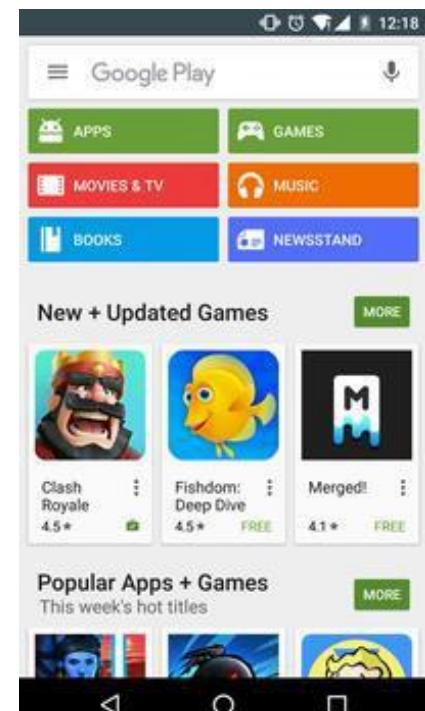
# **Taking Pictures in an Android App**

- How to take photos from your app using Android Camera app
- 4 Steps:
  - Request the camera feature
  - Take a Photo with the Camera App
  - Get the Thumbnail
  - Save the Full-size Photo

# 1. Request the Smartphone Camera Feature

- If your app takes pictures using the phone's Camera, you can allow only devices with a camera find your app while searching Google Play Store
- How?
- Make the following declaration in AndroidManifest.xml

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



# 1. Request the Smartphone Camera Feature (continued)

- Runtime permission is needed from the users.

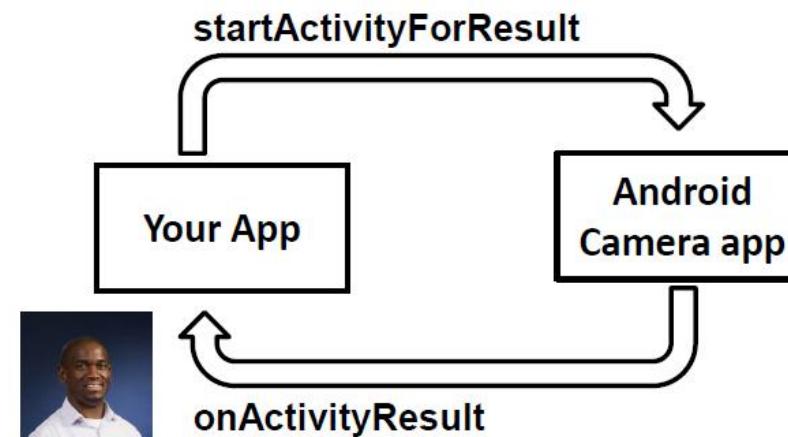
```
if (ContextCompat.checkSelfPermission(getContext(),
    Manifest.permission.CAMERA) != PackageManager.PERMISSION_GRANTED) {

    if (ActivityCompat.shouldShowRequestPermissionRationale((Activity)
        getContext(), Manifest.permission.CAMERA)) {

        } else {
            ActivityCompat.requestPermissions((Activity) getContext(),
                new String[]{Manifest.permission.CAMERA},
                MY_PERMISSIONS_REQUEST_CAMERA);
        }
    }
}
```

## 2. Capture an Image with the Camera App

- To take picture, your app needs to send **implicit Intent** requesting for a picture to be taken (i.e. action = capture an image)
- Call **startActivityForResult( )** with Camera intent since picture sent back
  - What type of intent is it? implicit or explicit?
- Potentially, multiple apps/activities can handle this/take a picture
- Check that at least 1 Activity that can handle request to take picture using **resolveActivity**



# Code to Take a Photo with the Camera App

```
static final int REQUEST_IMAGE_CAPTURE = 1;

private void dispatchTakePictureIntent() {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE);
    }
}
```

3. Send Intent requesting an image to be captured  
(usually handled by Android's Camera app)

1. Build Intent, action = capture an image

2. Check that there's at least 1 Activity that  
can handle request to capture an image  
(Avoids app crashing if no camera app available)

### 3. Get the Thumbnail

- Android Camera app returns thumbnail of photo (small bitmap)
- Thumbnail bitmap returned in “extra” of **Intent** delivered to **onActivityResult( )**

In **onActivityResult( )**, receive  
thumbnail picture sent back

```
protected void onActivityResult(int requestCode, int resultCode, Intent data
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == RESULT_OK) {
        Bundle extras = data.getExtras();
        Bitmap imageBitmap = (Bitmap) extras.get("data");
        mImageView.setImageBitmap(imageBitmap);
    }
}
```

# 4. Save Full-Sized Photo

- Android Camera app saves full-sized photo in a filename you give it
- We need phone owner's permission to write to external storage
- Android systems have:
  - **Internal storage:** data stored here is available by only your app
  - **External storage:** data stored here is available to all apps
- Would like all apps to read pictures this app takes, so use external storage

# Save Full-Sized Photo

- Android Camera app can save full-size photo to
  - **Public external storage** (shared by all apps)
    - `getExternalStoragePublicDirectory()`
    - Need to get permission
  - **Private storage** (Seen by only your app, deleted when your app uninstalls):
    - `getExternalFilesDir()`
- Either way, need phone owner's permission to write to external storage
- In `AndroidManifest.xml`, make the following declaration

```
<manifest ...>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    ...
</manifest>
```

```
static final int REQUEST_TAKE_PHOTO = 1;

private void dispatchTakePictureIntent() {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    // Ensure that there's a camera activity to handle the intent
    if (takePictureIntent.resolveActivity(getApplicationContext()) != null) {
        // Create the File where the photo should go
        File photoFile = null;
        try {
            photoFile = createImageFile(); ← Create file to store full-sized image
        } catch (IOException ex) {
            // Error occurred while creating the File
            ...
        }
        // Continue only if the File was successfully created
        if (photoFile != null) {
            Uri photoURI = FileProvider.getUriForFile(this,
                "com.example.android.fileprovider",
                photoFile); ← Put URI into intent extras
            takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, photoURI);
            startActivityForResult(takePictureIntent, REQUEST_TAKE_PHOTO); ← Take a picture
        }
    }
}
```

Create new intent for image capture

Check if a camera exists

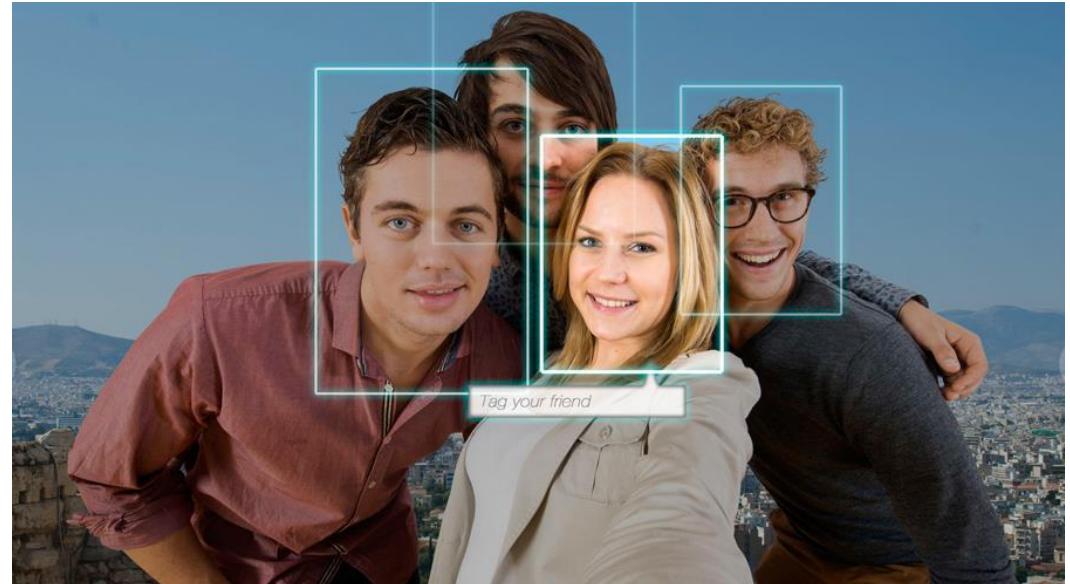
Build URI location to store captured image: file//xyz

Take a picture

# **Face Recognition**

# Face Recognition

- Answers the question:  
**Who** is this person in this picture?
- Compares unknown face to database of faces (or facial attributes) with known identity
- Neural networks/deep learning now makes comparison faster



# FindFace App: Stalking on Steroids?

- See stranger you like? Take a picture
- App searches 1 billion pictures using neural networks < 1 second
- Finds person's picture, identity, link on VK (Russian Facebook)
- You can send friend Request
  - ~ 70% accurate!
- Can also upload picture of celebrity you like
- Finds 10 strangers on Facebook who look similar, can send friend request



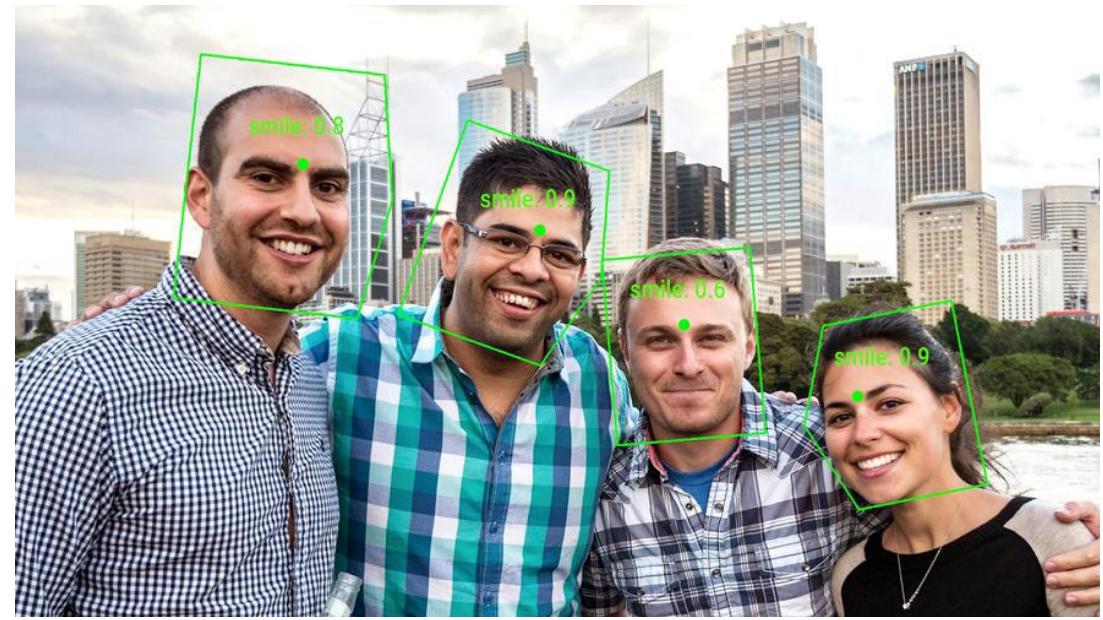
# FindFace App

- Also used in law enforcement
- Police identify criminals on watchlist

# **Face Detection**

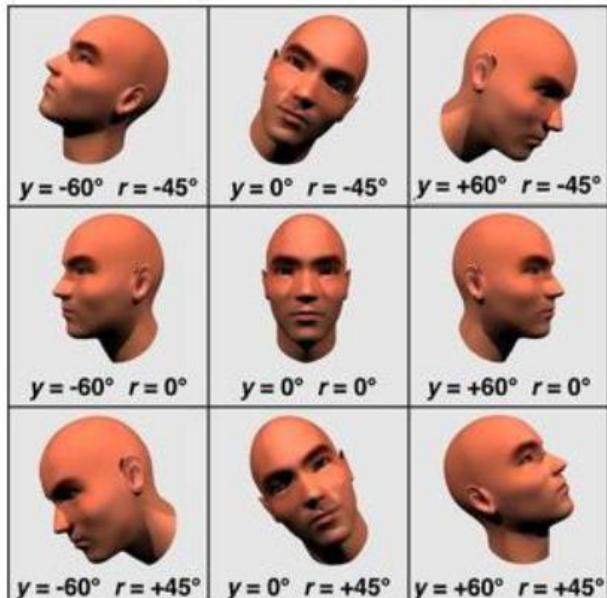
# Mobile Vision API

- **Face Detection:** Are there [any] faces in this picture?
- **How?** Locate face in photos and video and
- **Facial landmarks:** Eyes, nose and mouth
- **State of facial features:** Eyes open? Smiling?



# Face Detection: Google Mobile Vision API

- Detects faces:
  - reported at a position, with size and orientation
  - Can be searched for landmarks (e.g. eyes and nose)



Orientation



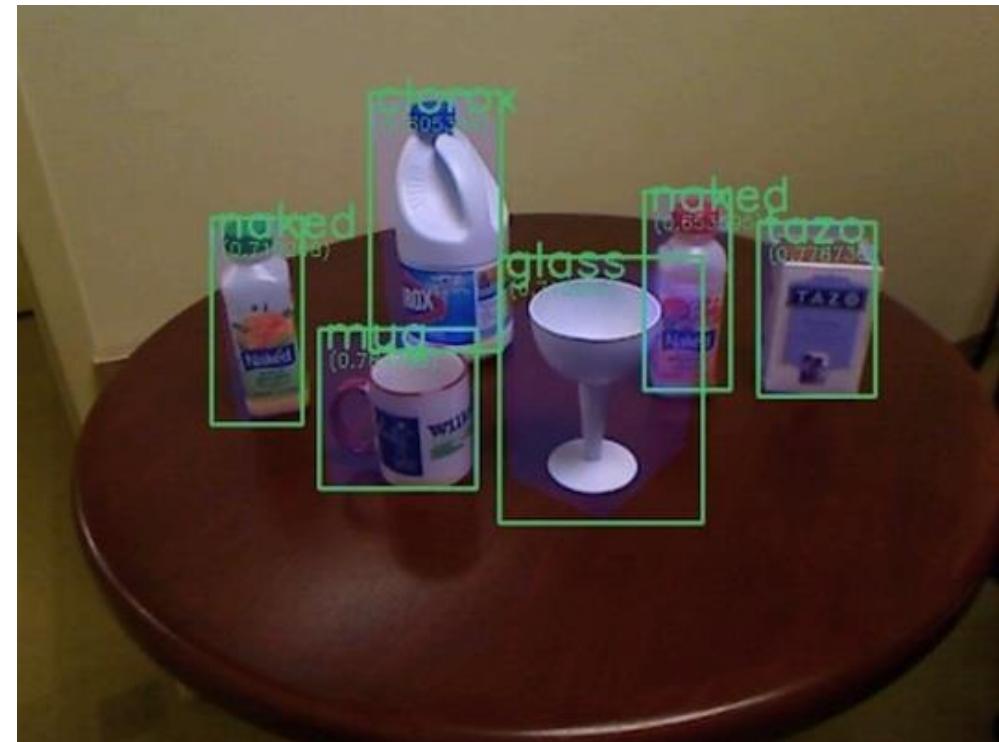
Euler Y angle	detectable landmarks
< -36 degrees	left eye, left mouth, left ear, nose base, left cheek
-36 degrees to -12 degrees	left mouth, nose base, bottom mouth, right eye, left eye, left cheek, left ear tip
-12 degrees to 12 degrees	right eye, left eye, nose base, left cheek, right cheek, left mouth, right mouth, bottom mouth
12 degrees to 36 degrees	right mouth, nose base, bottom mouth, left eye, right eye, right cheek, right ear tip
> 36 degrees	right eye, right mouth, right ear, nose base, right cheek

# Google Mobile Vision API

- Mobile Vision API also does:
  - **Face tracking:** detects faces in consecutive video frames
  - **Classification:** Eyes open? Face smiling?
- Classification:
  - Determines whether a certain facial characteristic is present
  - API currently supports 2 classifications: eye open, smiling
  - Results expressed as a confidence that a facial characteristic is present
    - Confidence  $> 0.7$  means facial characteristic is present
    - E.g.  $> 0.7$  confidence means it's likely person is smiling
- Mobile vision API does face **detection** but NOT **recognition**

# Face Detection

- **Face detection:** Special case of object-class detection
- **Object-class detection task:** find locations and sizes of all objects in an image that belong to a given class.
  - E.g: bottles, cups, pedestrians, and cars
- **Object matching:** Objects in picture compared to objects in database of labelled pictures



# Creating the Face Detector

- In app's **onCreate** method, create face detector

```
FaceDetector detector = new FaceDetector.Builder(context)
    .setTrackingEnabled(false)
    .setLandmarkType(FaceDetector.ALL_LANDMARKS)
    .build();
```

Detect all landmarks

- **detector** is base class for implementing specific detectors. E.g. face detector, bar code detector
- Tracking finds same points in multiple frames (continuous)
- Detection works best in single images when **trackingEnabled** is false

# Detecting Faces and Facial Landmarks

- Create Frame (image data, dimensions) instance from bitmap supplied

```
Frame frame = new Frame.Builder().setBitmap(bitmap).build();
```

# Detecting Faces and Facial Landmarks

- Call detector synchronously with frame to detect faces

```
SparseArray<Face> faces = detector.detect(frame);
```

# Detecting Faces and Facial Landmarks

- Detector takes **Frame** as input, outputs array of **Faces** detected
- **Face** is a single detected human face in image or video
- Iterate over array of faces, landmarks for each face, and draw the result based on each landmark's position

```
for (int i = 0; i < faces.size(); ++i) {  
    Face face = faces.valueAt(i);  
    for (Landmark landmark : face.getLandmarks()) {  
        int cx = (int) (landmark.getPosition().x * scale);  
        int cy = (int) (landmark.getPosition().y * scale);  
        canvas.drawCircle(cx, cy, 10, paint);  
    }  
}
```

Iterate through face array

Get face at position i in Face array

Return list of face landmarks,  
e.g., eyes, nose

Return landmark's xy position,  
where 0,0 is image's upper left  
corner

# Other Stuff

- To count faces detected, call **faces.size()**. E.g.

```
TextView faceCountView = (TextView) findViewById(R.id.face_count);
faceCountView.setText(faces.size() + " faces detected");
```

- Querying Face detector's status

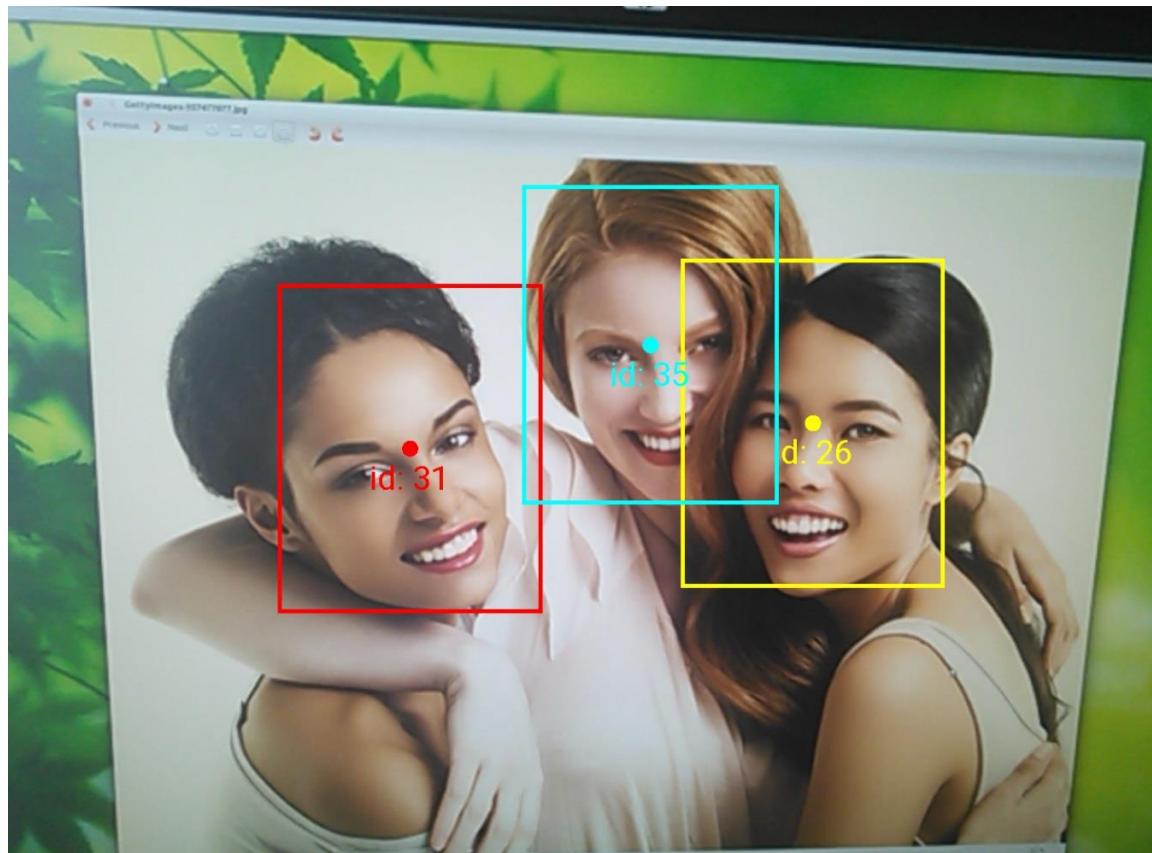
```
if (!detector.isOperational()) {  
    // ...  
}
```

- Releasing Face detector (frees up resources)

```
detector.release();
```

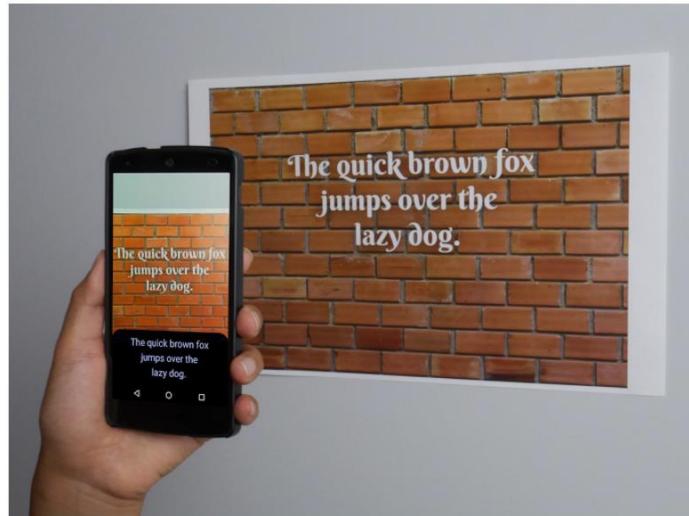
# Detect & Track Multiple Faces in Video

- Can also track multiple faces in image sequences/video, draw rectangle round each one



# Mobile Vision API: Other Functionality

- Barcode scanner
- Optical Character Recognition (OCR): Recognize text

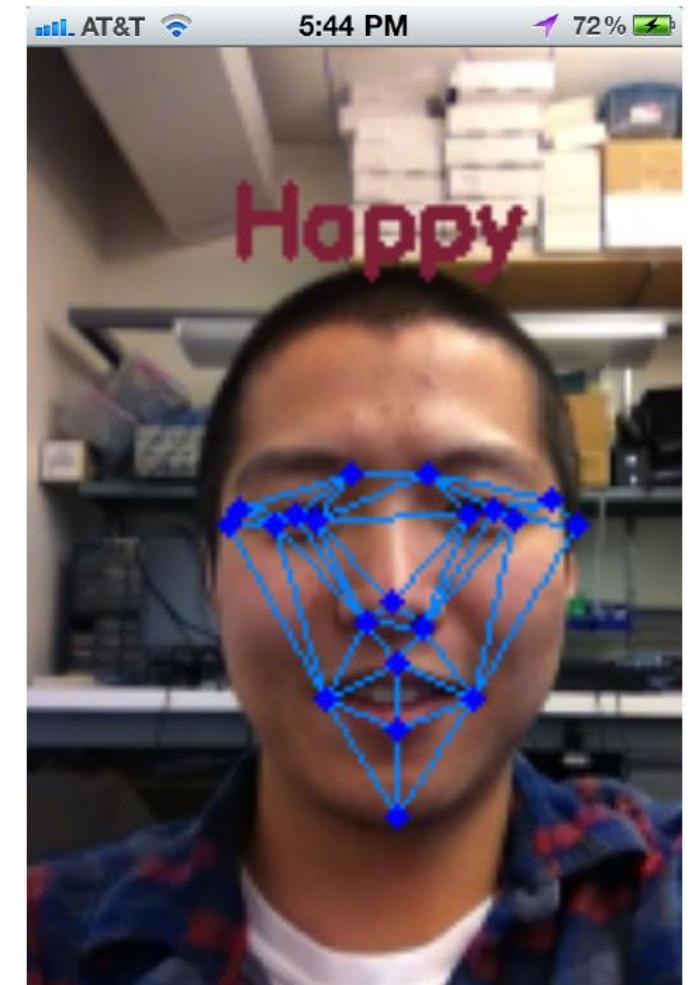


# **Face Interpretation**

- Reference:
- Yang, Xiaochao, et al. "Visage: A face interpretation engine for smartphone applications." *Mobile Computing, Applications, and Services Conference*. Springer Berlin Heidelberg, 2012. 149-168.

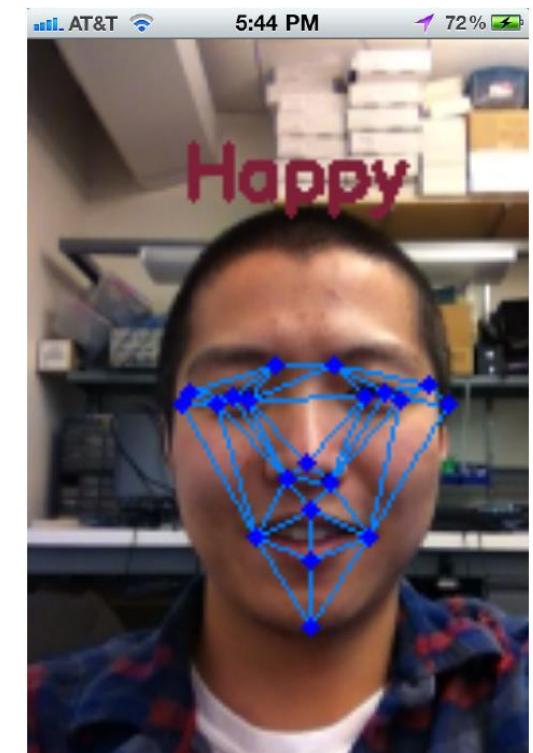
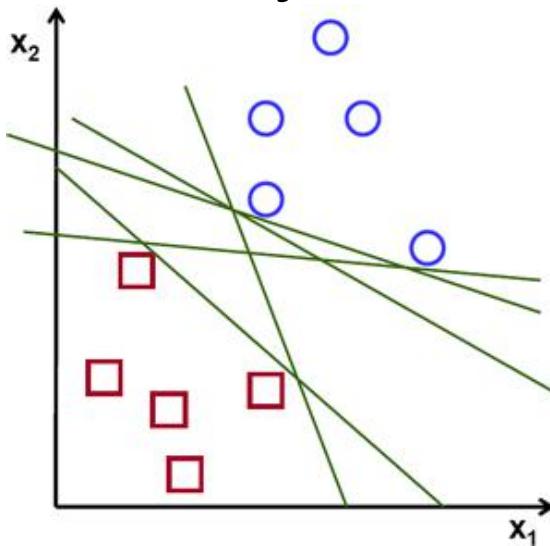
# Visage Face Interpretation Engine

- Real-time face interpretation engine for smart phones
- Tracking user's 3D head orientation + facial expression
- Facial expression, affect, emotion
  - angry, disgust, fear, happy, neutral, sad, surprise
- Use? Can be used in Mood Profiler app



# Facial Expression Inference

- Active appearance model
  - Describes 2D image as triangular mesh of landmark points
- 7 expression classes: angry, disgust, fear, happy, neutral, sad, surprise
- Extract triangle shape, texture features
- Classify features using Machine learning



# Classification accuracy

Expressions	Anger	Disgust	Fear	Happy	Neutral	Sadness	Surprise
Accuracy(%)	82.16	79.68	83.57	90.30	89.93	73.24	87.52

Table 4. Facial expression classification accuracy using the JAFFE dataset

# Camera, Audio, Video and Sound (Continued)

CSE 162 – Mobile Computing

Hua Huang

Department of Computer Science and Engineering  
University of California, Merced

# **Media (audio and video) Recording in Android**

# MediaRecorder overview

- The Android multimedia framework includes support for capturing and encoding a variety of common audio and video formats.

# Requesting permission to record audio

- To be able to record, your app must tell the user that it will access the device's audio input

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

- RECORD\_AUDIO is considered a "dangerous" permission because it may pose a risk to the user's privacy.
- Starting with Android 6.0 (API level 23) an app that uses a dangerous permission must ask the user for approval at run time.
  - After the user has granted permission, the app should remember and not ask again.

# Creating and running a MediaRecorder

- Set the audio source using `setAudioSource()`. You'll probably use MIC.
- Set the output file format using `setOutputFormat()`

```
private void startRecording() {  
    recorder = new MediaRecorder();  
    recorder.setAudioSource(MediaRecorder.AudioSource.MIC);  
    recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);  
    recorder.setOutputFile(fileName);  
    recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);  
  
    try {  
        recorder.prepare();  
    } catch (IOException e) {  
        Log.e(LOG_TAG, "prepare() failed");  
    }  
  
    recorder.start();  
}
```

# Creating and running a MediaRecorder

- Set the output file name using `setOutputFile()`. You must specify a file descriptor that represents an actual file.
- Set the audio encoder using `setAudioEncoder()`.
- Complete the initialization by calling `prepare()`.
- Start the recorder.

```
private void startRecording() {  
    recorder = new MediaRecorder();  
    recorder.setAudioSource(MediaRecorder.AudioSource.MIC);  
    recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);  
    recorder.setOutputFile(fileName);  
    recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);  
  
    try {  
        recorder.prepare();  
    } catch (IOException e) {  
        Log.e(LOG_TAG, "prepare() failed");  
    }  
  
    recorder.start();  
}
```

# Stop recording

```
private void stopRecording() {  
    recorder.stop();  
    recorder.release();  
    recorder = null;  
}
```

# Audio Project Ideas

- OpenAudio project, <http://www.openaudio.eu/>
  - Many tools, dataset available
    - OpenSMILE: Tool for extracting > 1000 audio features
    - Windowing
  - MFCC
  - Pitch
  - Statistical features, etc
- Supports popular file formats (e.g. Weka)
- OpenEAR: Toolkit for automatic speech emotion recognition
- iHeaRu-EAT Database: 30 subjects recorded speaking while eating

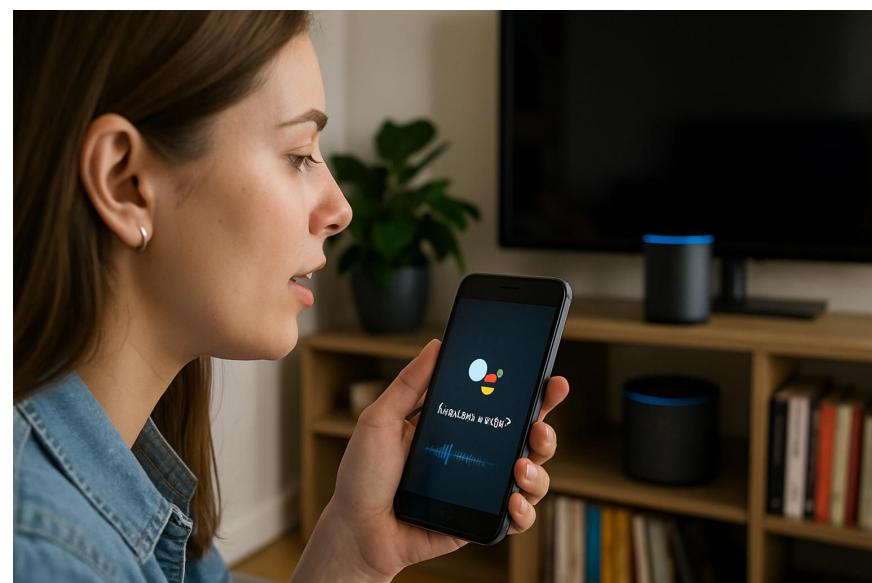
# Speech Recognition

- speech recognition, also known as automatic speech recognition (ASR), computer speech recognition or speech-to-text, is a capability that enables a program to process human speech into a written format.

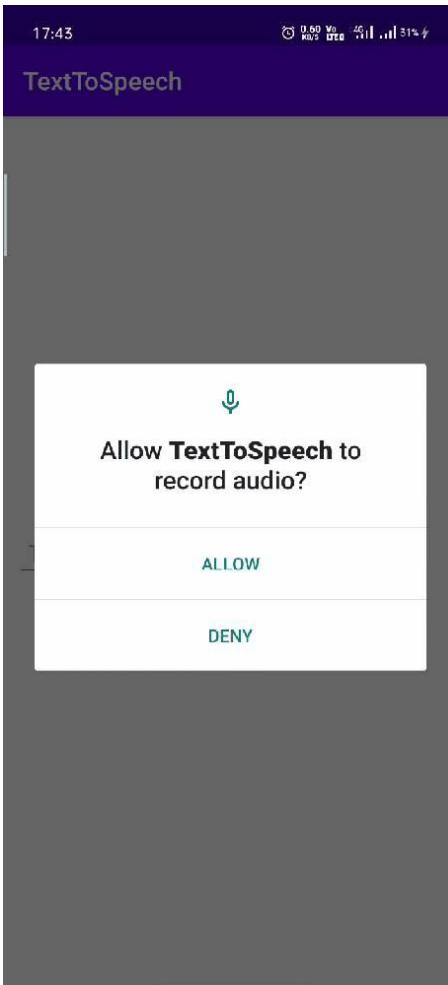


# Speech recognition use cases

- Automotive: Speech recognizers improves driver safety by enabling voice-activated navigation systems and search capabilities in car radios.
- Technology: Virtual agents are increasingly becoming integrated within our daily lives, particularly on our mobile devices.
  - We use voice commands to access them through our smartphones, such as through Google Assistant or Apple's Siri, for tasks, such as voice search, or through our speakers, via Amazon's Alexa or Microsoft's Cortana, to play music.
- Healthcare: Doctors and nurses leverage dictation applications to capture and log patient diagnoses and treatment notes.



# Build a speech recognizer in Android



# The Android SpeechRecognizer Class

- This class provides access to the speech recognition service.
- The implementation of this API is to stream audio to remote servers to perform speech recognition.
  - Not suitable for continuous recognition
  - consume a significant amount of battery and bandwidth.

# Usage of the SpeechRecognizer Class

- Create the class using *createSpeechRecognizer()*

```
1 speechRecognizer = SpeechRecognizer.createSpeechRecognizer(this);
2 final Intent speechRecognizerIntent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
3 speechRecognizerIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
4 speechRecognizerIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, Locale.getDefault());
```

- Create a speechRecognizerIntent.
  - The constant **ACTION\_RECOGNIZE\_SPEECH** starts an activity
    - The activity will prompt the user to speak and send it through a speech recognizer.
  - **EXTRA\_LANGUAGE\_MODEL**: Informs the recognizer which speech model to use
  - **EXTRA\_LANGUAGE\_MODEL**: Informs the recognizer which speech model to prefer when performing ACTION\_RECOGNIZE\_SPEECH.
  - **EXTRA\_LANGUAGE**: Optional IETF language tag (as defined by BCP 47), for example, “en-US”.

```
1 speechRecognizer = SpeechRecognizer.createSpeechRecognizer(this);
2 final Intent speechRecognizerIntent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
3 speechRecognizerIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
4 speechRecognizerIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, Locale.getDefault());
```

# Create a speechRecognitionListener class

```
speechRecognizer.setRecognitionListener(new RecognitionListener() {
    @Override
    public void onBeginningOfSpeech() {
        editText.setText("");
        editText.setHint("Listening....");
    }
    @Override
    public void onResults(Bundle bundle) {
        micButton.setImageResource(R.drawable.ic_mic_black_off);
        ArrayList<String> data = bundle.getStringArrayList(SpeechRecognizer.RESULTS_RECOGNITION);
        editText.setText(data.get(0));
    }
});
```

- We get an ArrayList of String as a result
- We can use it to parse a command, or input texts.

```
public void onResults(Bundle bundle) {  
    micButton.setImageResource(R.drawable.ic_mic_black_off);  
    ArrayList<String> data = bundle.getStringArrayList(SpeechRecognizer.RESULTS_RECOGNITION);  
    editText.setText(data.get(0));  
}
```

# **Speech Analytics**

# Voice Based Analytics

- Voice can be analyzed, lots of useful information extracted
  - Who is talking? (Speaker identification)
  - How many social interactions a person has a day
  - Emotion of person while speaking
  - Anxiety, depression, intoxication, of person, etc.
- For speech recognition, voice analytics used to:
  - Discard useless information (background noise, etc)
  - Identify and extract linguistic content



# Mel Frequency Cepstral Coefficients (MFCCs)

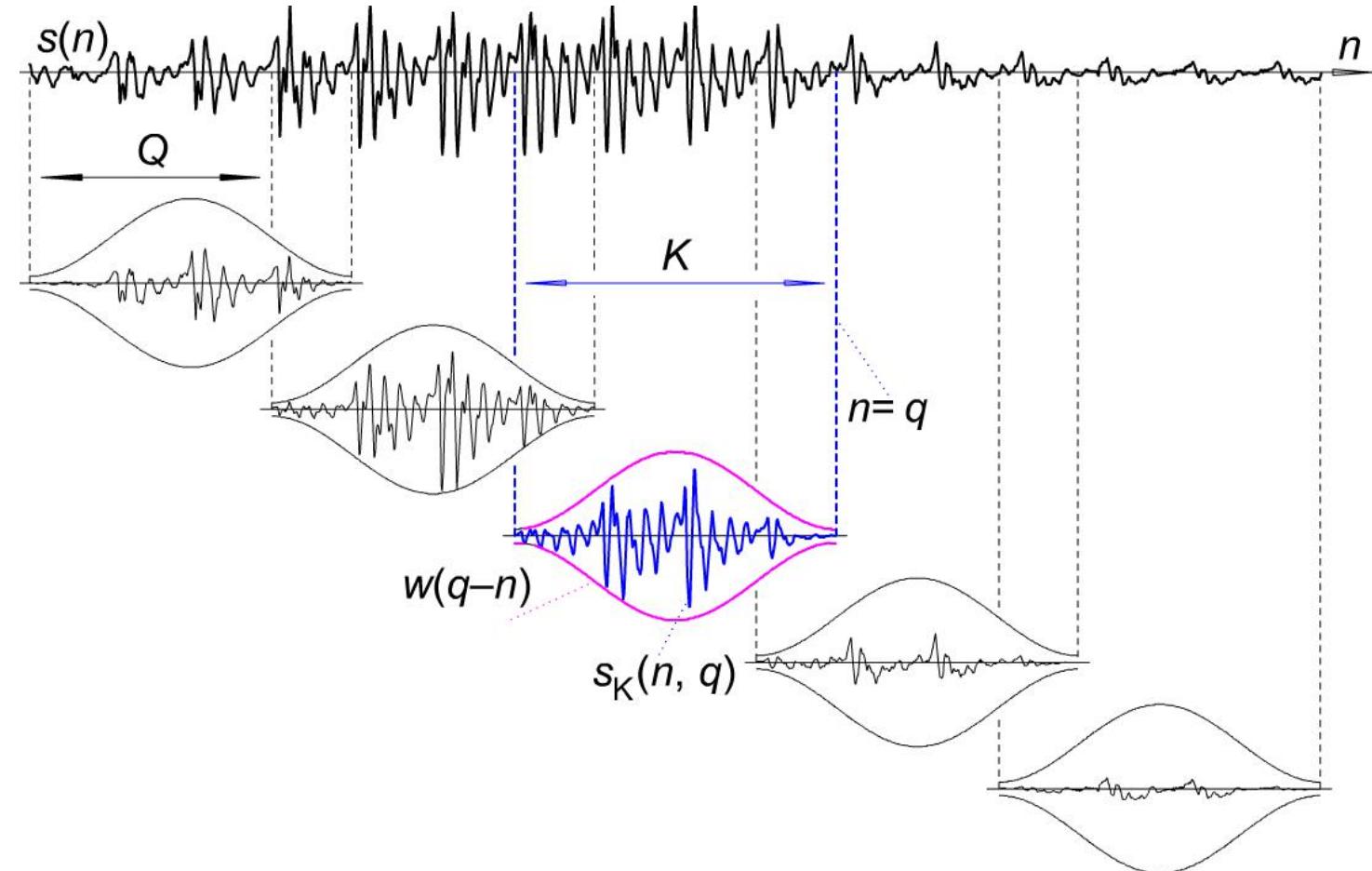
- MFCCs widely used in speech and speaker recognition for representing envelope of **power spectrum of voice**
- **Power spectrum?** Amount of power at various frequencies
- Roughly
  - Male voice: low frequency (bass)
  - Female voice: high frequency (treble)
- Popular approach in Speech recognition
  - MFCC features + Hidden Markov Model (HMM) classifiers

# MFCC Steps: Overview

1. Frame the signal into short frames.
2. For each frame calculate the periodogram estimate of the power spectrum.
3. Apply the mel filter bank to the power spectra, sum the energy in each filter.
4. Take the logarithm of all filter bank energies.
5. Take the DCT of the log filter bank energies.
6. Keep DCT coefficients 2-13, discard the rest.

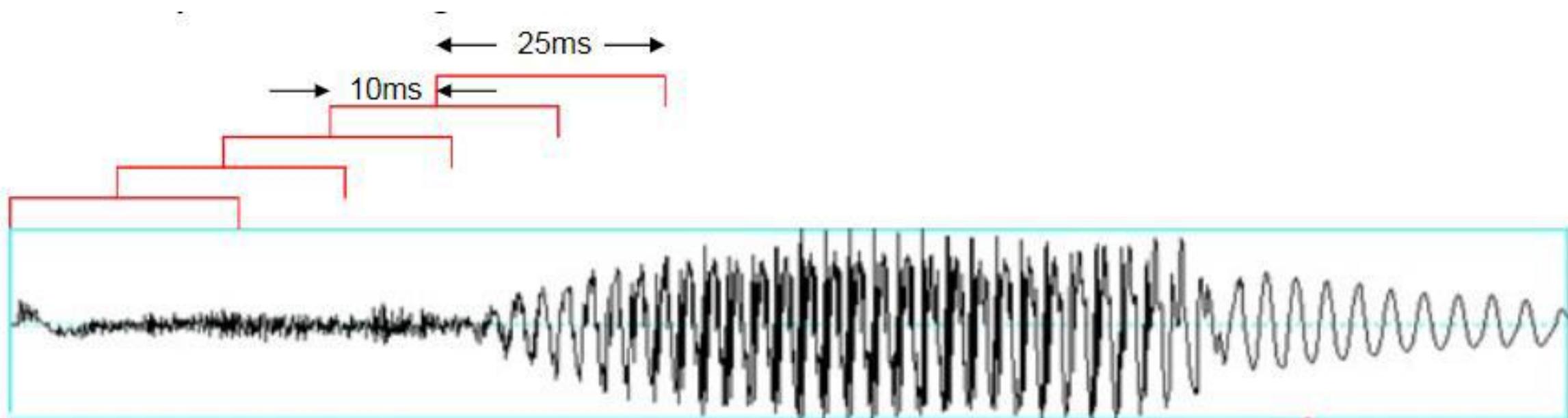
# Step 1: Windowing

- Audio is continuously changing.
- Break into short, overlapping segments (20-40 milliseconds)
- Can assume audio does not change in short window



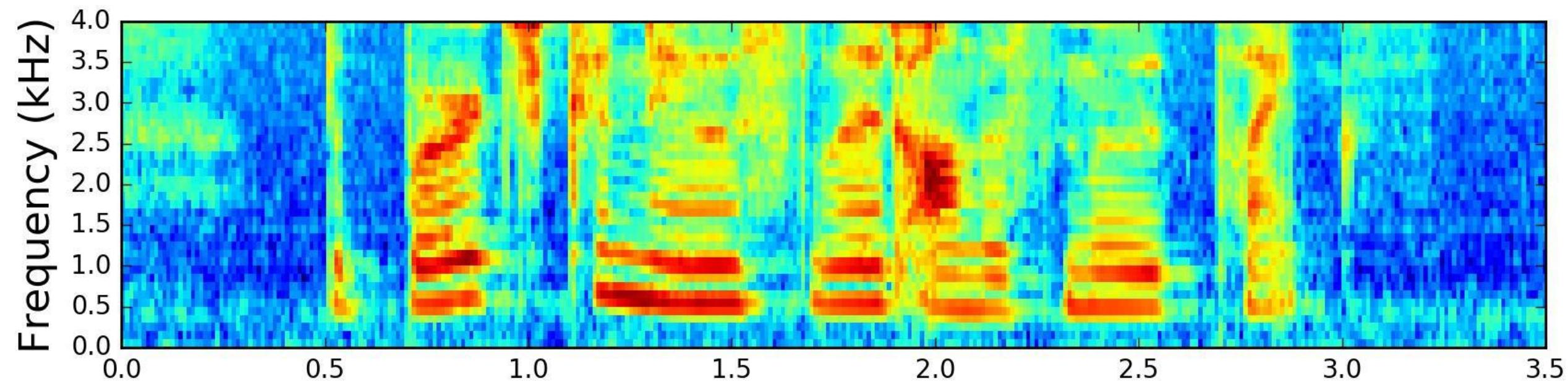
# Step 1: Windowing

- Essentially, break into smaller overlapping frames
- Need to select frame length (e.g. 25 ms), shift (e.g. 10 ms)



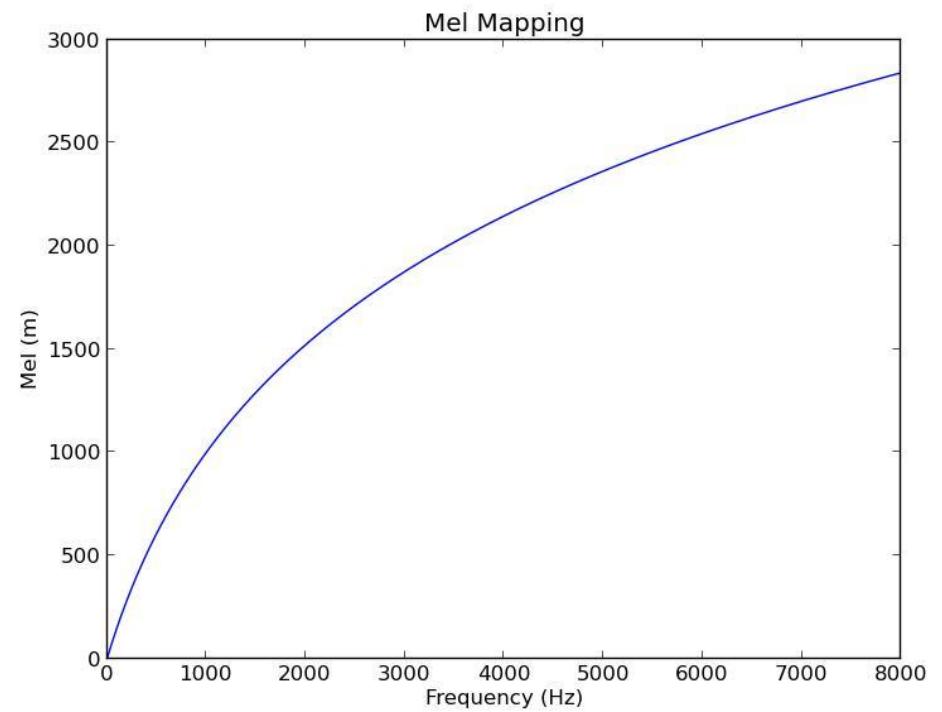
# Step 2: Calculate Power Spectrum of each Frame

- Cochlea (**kaa·klee·uh**), part of human ear, vibrates at different parts depending on sound frequency
- Power spectrum Periodogram similarly identifies frequencies present in each frame



# Background: Mel Scale

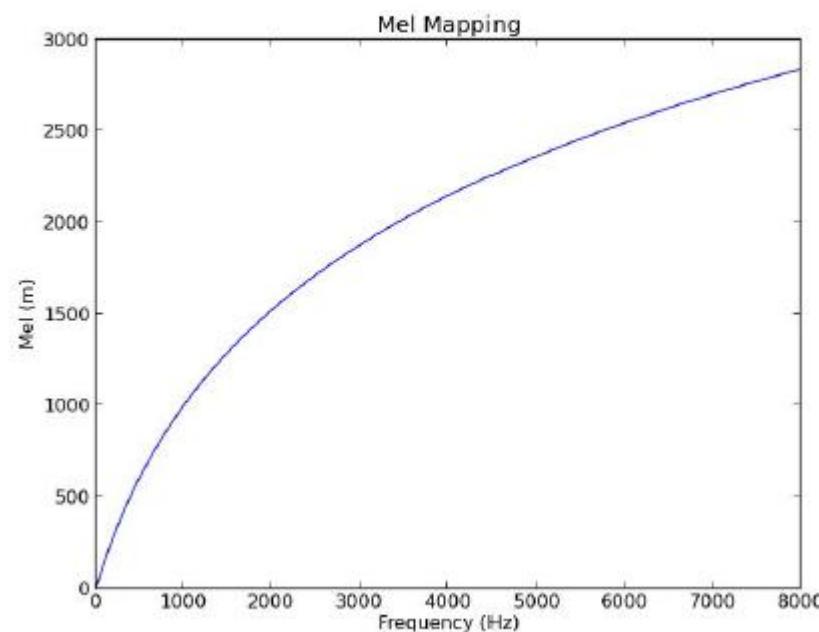
- Transforms speech attributes (frequency, tone, pitch) on non-linear scale based on human perception of voice
- Result: non-linear amplification, MFCC features that mirror human perception
  - E.g. humans good at perceiving small change at low frequency than at high frequency



# Step 3: Apply Mel FilterBank

Non-linear conversion from frequency to Mel Space

$$M(f) = 1125 \ln(1 + f/700) \quad (1)$$



# Step 4: Apply Logarithm of Mel Filterbank

- Take log of filterbank energies at each frequency
- This step makes output mimic human hearing better
- We don't hear loudness on a linear scale
- Changes in loud noises may not sound different

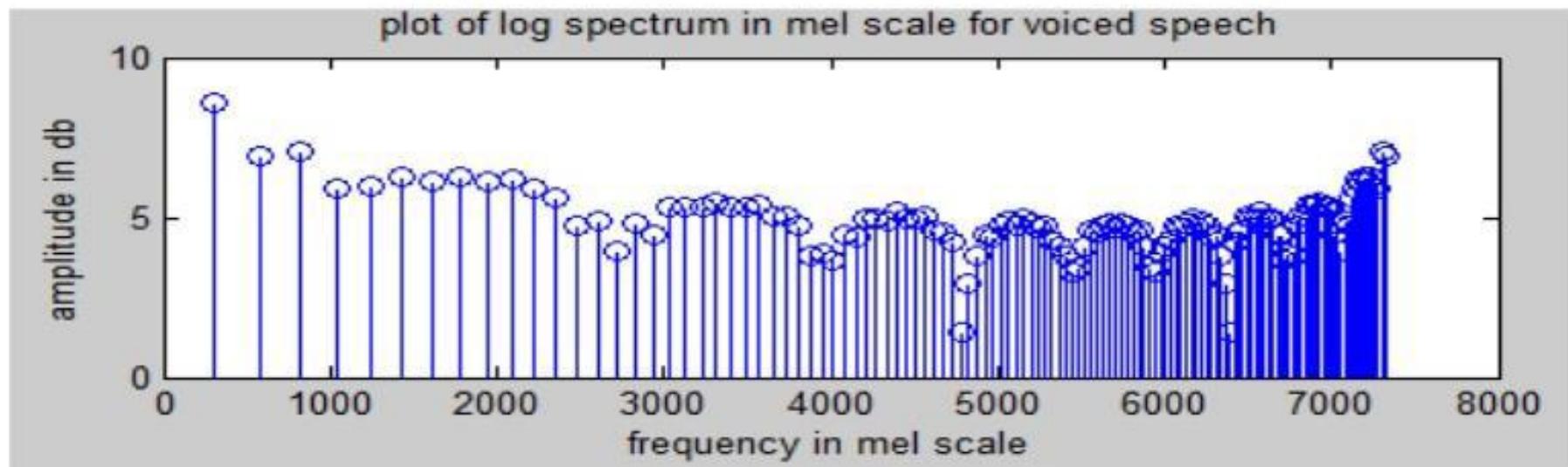


Fig.7. Spectrum of voiced speech

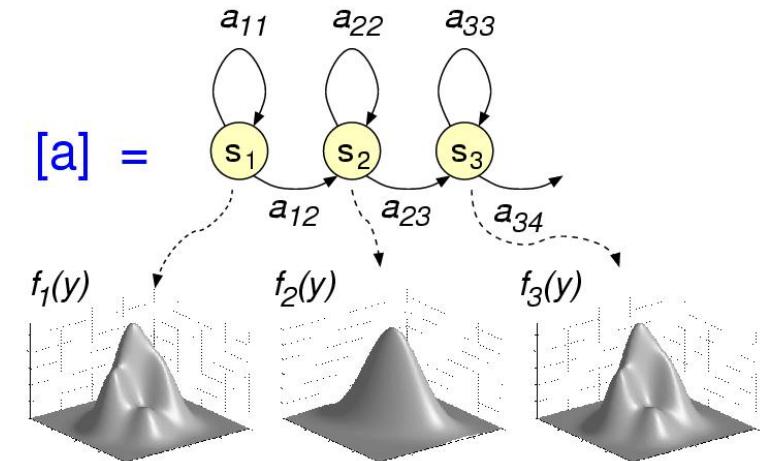
## Step 5: DCT of log filterbank:

- There are correlations between signals at different frequencies
- Discrete Cosine Transform (DCT) extracts most useful and independent features
- Final result: 39-element acoustic vector used in speech processing algorithms

# Speech Classification

- Human speech can be broken into phonemes
- Example of phoneme is /k/ in the words (**cat, school, skill**)
- Classic Speech recognition tries to recognize sequence of phonemes in a word
- Typically uses Hidden Markov Model (HMM)
  - Recognizes letters, then words, then sentences
  - Like a state machine that strings together sequence of sounds recognized

Hidden Markov Models



# **Emotion Detection**

# Definitions

- Affect
  - Broad range of feelings
  - Can be either emotions or moods
- Emotion
  - Brief, intense feelings (anger, fear, sadness, etc)
  - Directed at someone or something
- Mood
  - Less intense, not directed at a specific stimulus
  - Lasts longer (hours (4?) or days)

# Physiological Measurement of Emotion

- **Biological arousal:** heart rate, respiration, perspiration, temperature, muscle tension
- **Expressions:** facial expression, gesture, posture, voice intonation, breathing noise

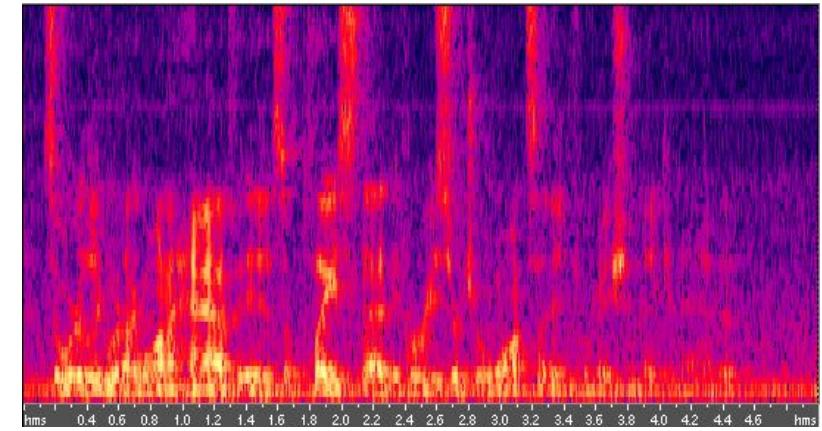
Emotion	Physiological Response
Anger	Increased heart rate, blood vessels bulge, constriction
Fear	Pale, sweaty, clammy palms
Sad	Tears, crying
Disgust	Salivate, drool
Happiness	Tightness in chest, goosebumps

# Audio Features for Emotion Detection

- MFCC widely used for analysis of speech content, Automatic Speaker Recognition (ASR)
  - Who is speaking?
- Other audio features exist to capture sound characteristics/dynamics (prosody)
  - Useful in detecting emotion in speech
- **Pitch:** the frequency of a sound wave. E.g.
  - Sudden increase in pitch => Anger
  - Low variance of pitch => Sadness

# Audio Features for Emotion Detection

- **Intensity:** Energy of speech, intensity. E.g.
  - Angry speech: sharp rise in energy
  - Sad speech: low intensity
- **Temporal features:**
  - Speech rate, voice activity (e.g. pauses)
  - E.g. Sad speech: slower, more pauses
- **Other emotion features:** Voice quality, spectrogram, statistical measures



# Uses of Affect Detection

## E.g. Using Voice on Smartphone

- Audio processing (especially to detect affect, mental health) can revolutionize healthcare
  - Detection of mental health issues automatically from patients voice
  - Population-level (e.g campus wide) mental health screening
  - Continuous, passive stress monitoring
  - Suggest interventions: breathing exercises, play relaxing music
  - Monitoring social interactions, recognize conversations (number and duration per day/week, etc)

# Voice Analytics Example: SpeakerSense (Lu et al)

- Identifies speaker, who conversation is with

# Voice Analytics Example: StressSense (Lu et al)

- Detected stress in speaker's voice
- Features: MFCC, pitch, speaking rate
- Classification using GMM
- Accuracy: indoors (81%), outdoors (76%)

# Voice Analytics Example: Mental Illness Diagnosis

- What if depressed patient lies to psychiatrist, says “I’m doing great”
- Mental health (e.g. depression) detectable from voice, can be used to detect lying patient
- Doctors pay attention to speech aspects when examining patients
- E.g. depressed people have slower responses, more pauses, monotonic responses and poor articulation

Category	Patterns
Rate of speech	slow, rapid
Flow of speech	hesitant, long pauses, stuttering
Intensity of speech	loud, soft
Clarity	clear, slurred
Liveliness	pressured, monotonous, explosive
Quality	verbose, scant

# Detection of COVID from Respiratory sounds

- large-scale crowdsourced dataset of respiratory sounds collected to aid diagnosis of COVID-19.
- Coughs and breathing to understand how discernible COVID-19 sounds are from those in asthma or healthy controls.
- Simple binary machine learning classifier is able to classify correctly healthy and COVID-19 sounds.
- Were able to distinguish
  - User who had COVID-19 + cough vs healthy user with a cough
  - Users who had COVID-19 + cough vs. Users with asthma and a cough.

# Playing Audio and Video in Android

# MediaPlayer

- Android Classes used to play sound and video
  - **MediaPlayer**: Plays sound and video
  - **AudioManager**: plays only audio
- Any Android app can create instance of/use MediaPlayerAPIs to integrate video/audio playback functionality
- MediaPlayer can fetch, decode and play audio or video from:
  - Audio/video files stored in app's resource folders (e.g. **res/raw/folder**)
  - External URLs (over the Internet)

# MediaPlayer

- MediaPlayer supports:
  - **Streaming network protocols:** RTSP, HTTP streaming
  - **Media Formats:**
    - Audio (MP3, AAC, MIDI, etc),
    - Image (JPEG, GIF, PNG, BMP, etc)
    - Video (MPEG-4, H.263, H.264, H.265 AVC, etc)
- 4 major functions of a Media Player
  - **User interface**, user interaction
  - Handle **Transmission errors**: retransmissions, interleaving
  - **Decompress** audio
  - **Eliminate jitter**: Playback buffer (Pre-download 10-15 secs of music)

# **Example: Playing Audio File using MediaPlayer**

- Use **MediaPlayer** to play audio file



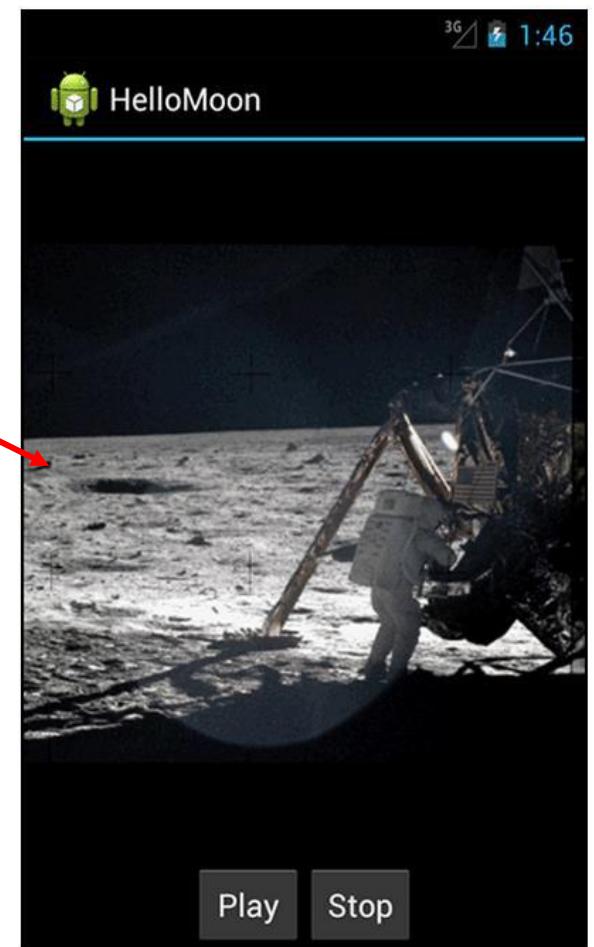
# Resources

- Put image **armstrong\_on\_moon.jpg** in **res/drawable/folders**
- Place audio file to be played back (**one\_small\_step.wav**) in **res/raw** folder
- Create **strings.xml** file for app
  - Play, Stop, Image description..

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

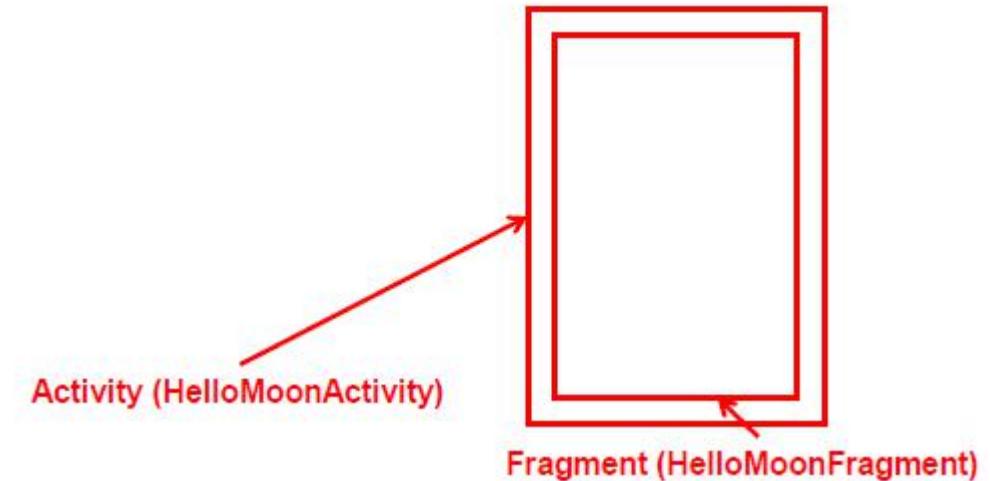
    <string name="app_name">HelloMoon</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
    <string name="hellomoon_play">Play</string>
    <string name="hellomoon_stop">Stop</string>
    <string name="hellomoon_description">Neil Armstrong stepping
        onto the moon</string>

</resources>
```

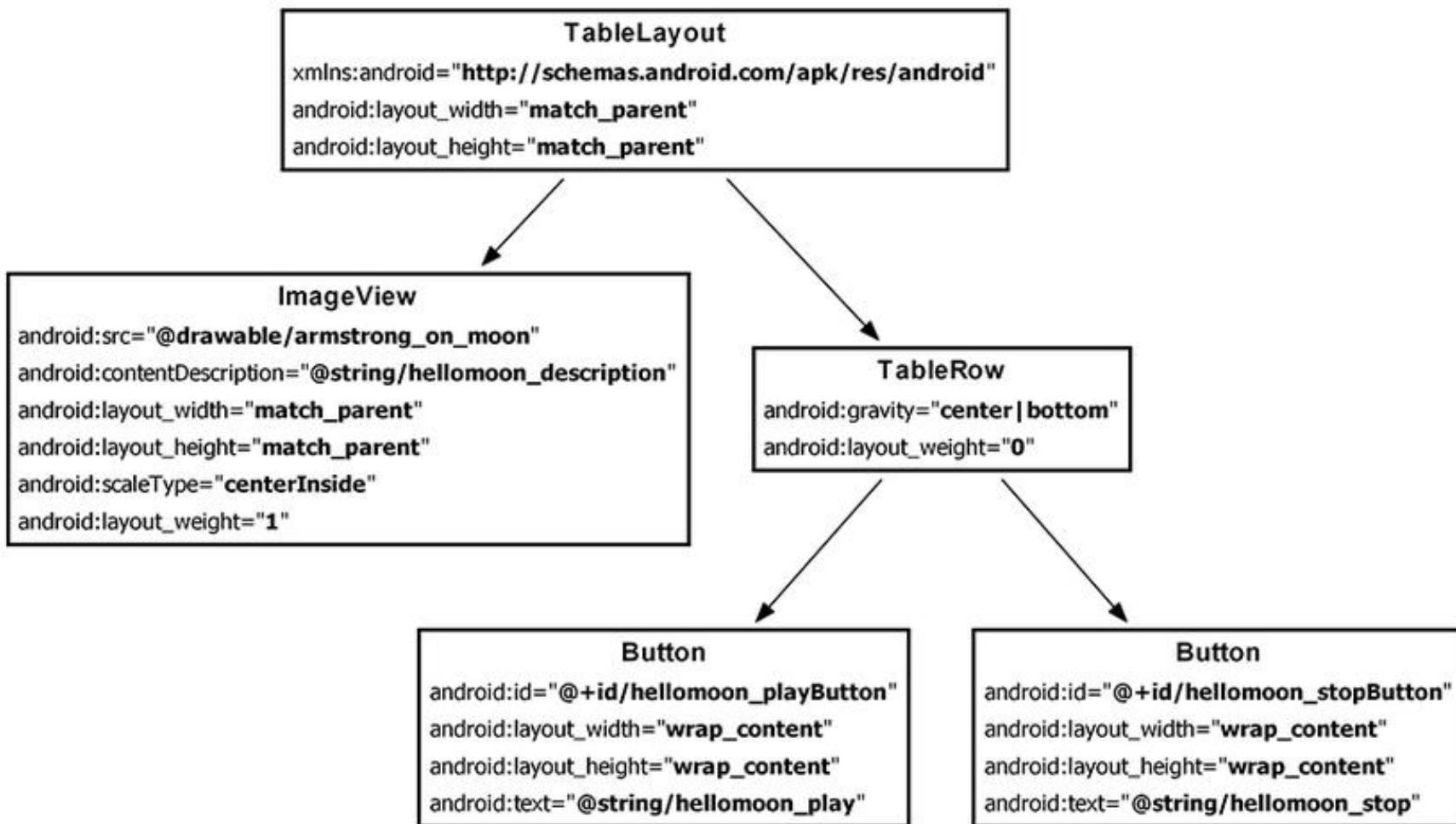


# The UI

- 1 activity (**HelloMoonActivity**) that hosts **HelloMoonFragment**
- **AudioPlayer** class will be created to encapsulate **MediaPlayer**
- First set up the rest of the app:
  - Define fragment's XML layout
  - Create fragment java class
  - Modify the activity (java) and its XML layout to host the fragment



# Defining the Layout for HelloMoonFragment



Define XML for HelloMoon UI (fragment\_hello\_moon.xml)

# Creating a Layout Fragment

- **Layout fragment:** Add fragments to hosting Activity's XML file
- Create activity's XML layout (**activity\_hello\_moon.xml**)
- **Activity's** XML layout file contains/hosts fragment



```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/helloMoonFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:name="com.bignerdranch.android.hellomoon.HelloMoonFragment">

</fragment>
```

# Using Media Player:

**Step 1: Request Permission in AndroidManifest or Place video/audio files in res/raw**

- If streaming video/audio over Internet (network-based content), request network access permission in AndroidManifest.xml:

```
<uses-permission android:name="android.permission.INTERNET" />
```



# Set up HelloMoonFragment.java

```
public class HelloMoonFragment extends Fragment {  
  
    private Button mPlayButton;  
    private Button mStopButton;  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,  
        Bundle savedInstanceState) {  
        View v = inflater.inflate(R.layout.fragment_hello_moon, parent, false);  
  
        mPlayButton = (Button)v.findViewById(R.id.hellomoon_playButton);  
        mStopButton = (Button)v.findViewById(R.id.hellomoon_stopButton);  
  
        return v;  
    }  
}
```



Get handle to Start, Stop buttons

- If playing back local file stored on user's smartphone, put video/audio files in **res/raw** folder

## Step 2: Create MediaPlayer Object, Start Player

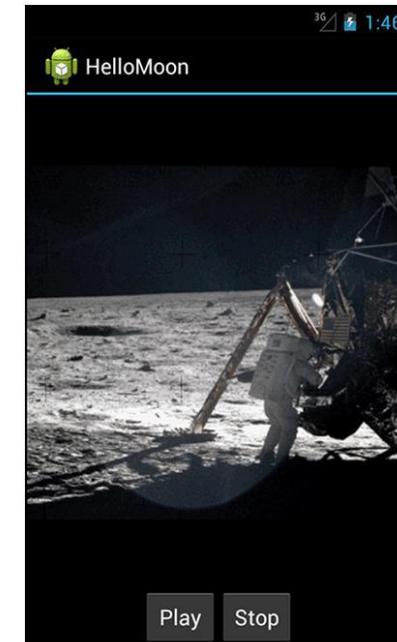
- To play audio file saved in app's **res/raw/** directory
- **Note:** Audio file opened by `create` (e.g. `one_small_step.mp3`) must be encoded in one of supported media formats

# Create AudioPlayer Class encapsulates MediaPlayer

```
public class AudioPlayer {  
  
    private MediaPlayer mPlayer;  
  
    public void stop() {  
        if (mPlayer != null) {  
            mPlayer.release();  
            mPlayer = null;  
        }  
    }  
  
    public void play(Context c) {  


mPlayer = MediaPlayer.create(c, R.raw.one_small_step);  
mPlayer.start();

  
    }  
}
```



- **Releasing the MediaPlayer**

- MediaPlayer can consume valuable system resources
- When done, call **release()** to free up system resources
- In **onStop()** or **onDestroy()** methods, call

```
MediaPlayer.release();  
MediaPlayer = null;
```

- **MediaPlayer in a Service:** Can play media (e.g. music) in background while app is not running
  - Start MediaPlayer as service

# Hook up Play and Stop Buttons

```
public class HelloMoonFragment extends Fragment {
    private AudioPlayer mPlayer = new AudioPlayer();
    private Button mPlayButton;
    private Button mStopButton;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
                            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_hello_moon, parent, false);

        mPlayButton = (Button)v.findViewById(R.id.hellomoon_playButton);
        mPlayButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                mPlayer.play(getActivity());
            }
        });

        mStopButton = (Button)v.findViewById(R.id.hellomoon_stopButton);
        mStopButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                mPlayer.stop();
            }
        });
        return v;
    }
}
```

# Extra: stream audio from internet

- To play audio from remote URL via HTTP streaming over the Internet

```
String url = "http://....."; // your URL here
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mediaPlayer.setDataSource(url);
mediaPlayer.prepare(); // might take long! (for buffering, etc)
mediaPlayer.start();
```

# **Multimedia Networking: Basic Concepts**

# Multimedia networking: 3 application types

- Multimedia refers to audio and video. 3 types

## *1. streaming, stored audio, video*

- *streaming*: transmit in batches, begin playout before downloading entire file
- e.g., YouTube, Netflix, Hulu
- Streaming Protocol used (e.g. Real Time Streaming Protocol (RTSP), HTTP streaming protocol (DASH))

## *2. streaming live audio, video*

- e.g., live sporting event

## *3. conversational voice/video over IP*

- Requires minimal delays due to interactive nature of human conversations
- e.g., Skype, RTP/SIP protocols

# Live Streaming

- Live streaming extremely popular now (E.g. going Live on Facebook)
- A person can share their experiences with friends
- Popular live streaming apps include Facebook, Periscope
- Also possible on devices such as Go Pro
- Uses RTMP (real time protocol by Adobe), or other 3 rd party APIs



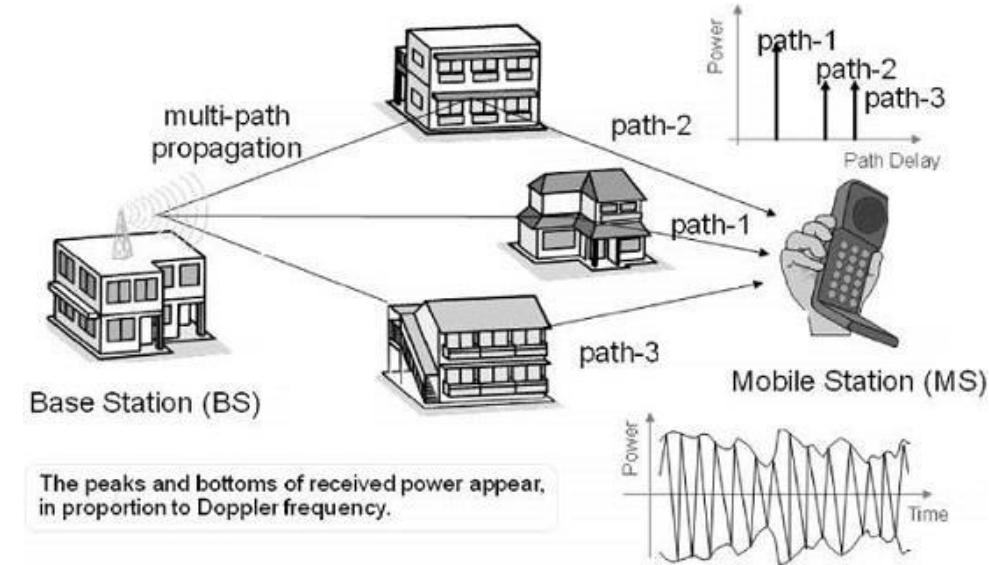
Facebook Live



Live GoPro

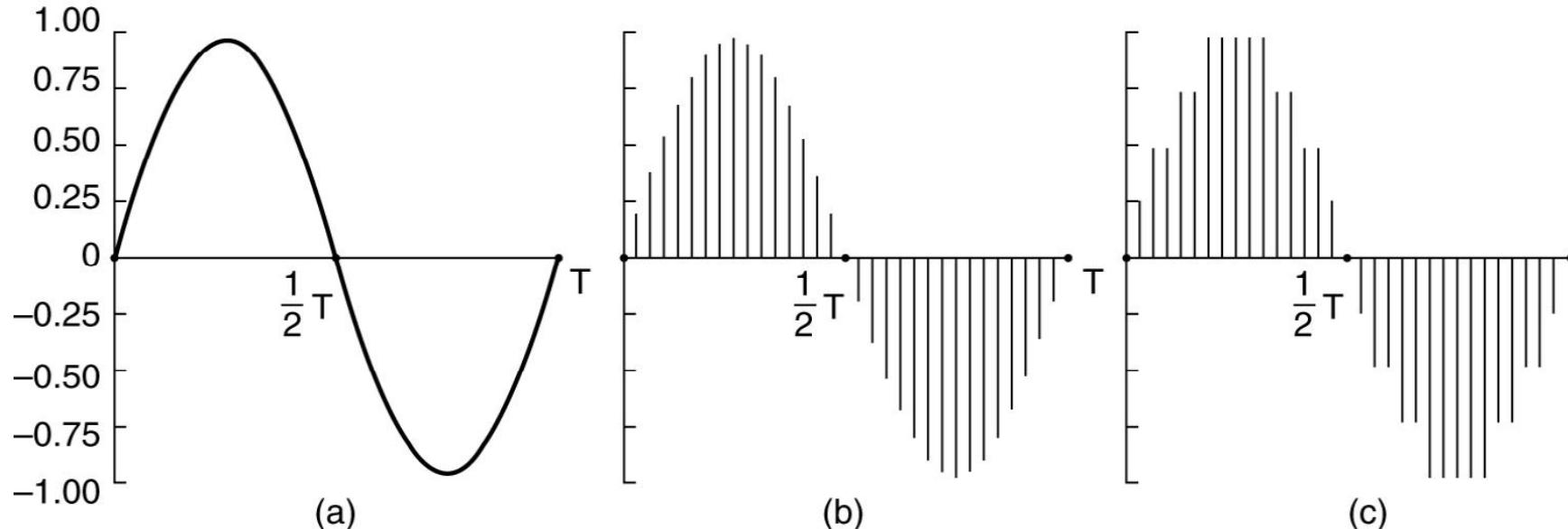
# Live Streaming Bandwidth Issues

- On WiFi, bandwidth is adequate, high quality video possible
- Cellular links:
  - Low bandwidth,
  - Variable bandwidth (multi-path fading)
    - Even when standing still
  - Optimized for download not upload
- Video quality increasing faster than cellular bandwidths
  - Ultra HD, 4k cameras makes it worse, now available on many smartphones



# Digital Audio

- Sender converts audio from analog waveform to digital signal
- E.g PCM uses 8-bit samples 8000 times per sec
- Receiver converts digital signal back into audio waveform



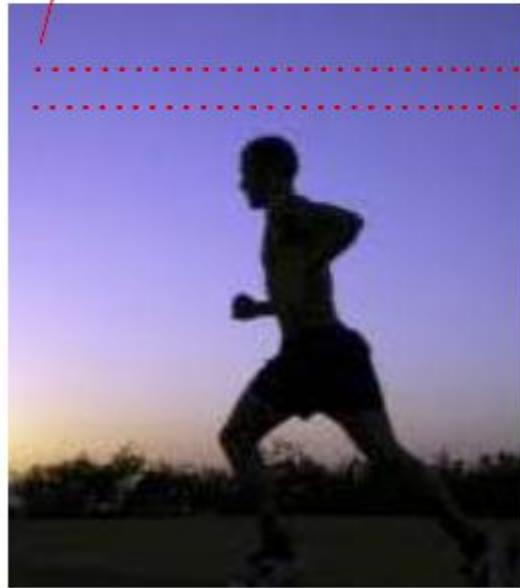
# Audio Compression

- Audio CDs:
  - 44,100 samples/second
  - Uncompressed audio, requires 1.4Mbps to transmit real-time
- Audio compression reduces transmission bandwidth required
  - E.g. MP3 (MPEG audio layer 3) compresses audio down to 96 kbps

# Video Encoding

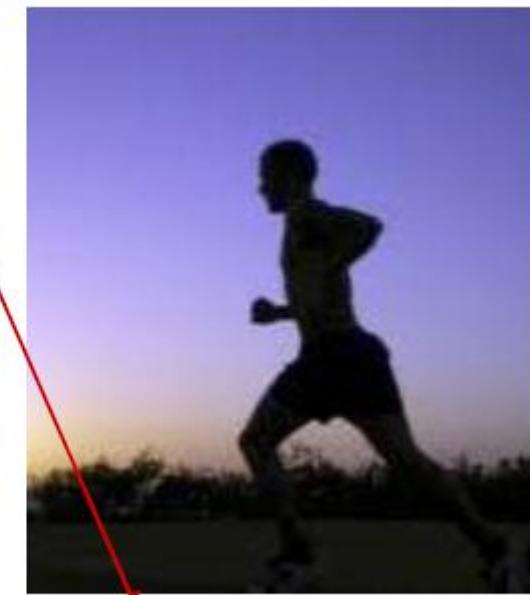
- **Digital image:** array of <R,G,B> pixels
- **Video:** sequence of images
- **Redundancy:** Consecutive frames mostly same (1/30 secs apart)
- **Video coding (e.g. MPEG):** use redundancy *within* and *between* images to decrease # bits used to encode video
  - **Spatial**(within image)
  - **Temporal**(from 1 image to next)

*spatial coding example:* instead of sending  $N$  values of same color (all purple), send only two values: color value (*purple*) and number of times repeated ( $N$ )



frame  $i$

*temporal coding example:* instead of sending complete frame at  $i+1$ , send only differences from frame  $i$



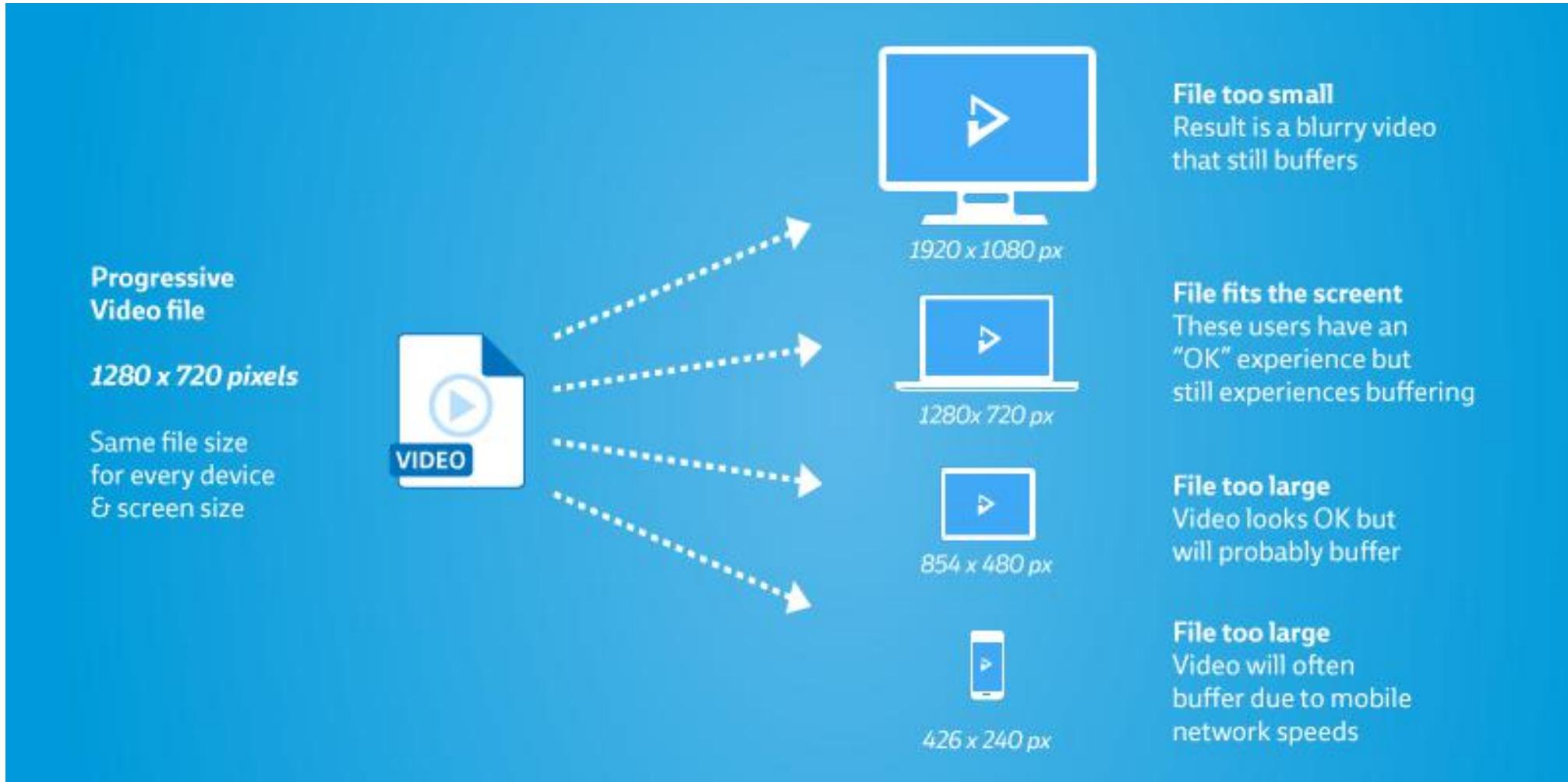
frame  $i+1$

# Adaptive Video Streaming

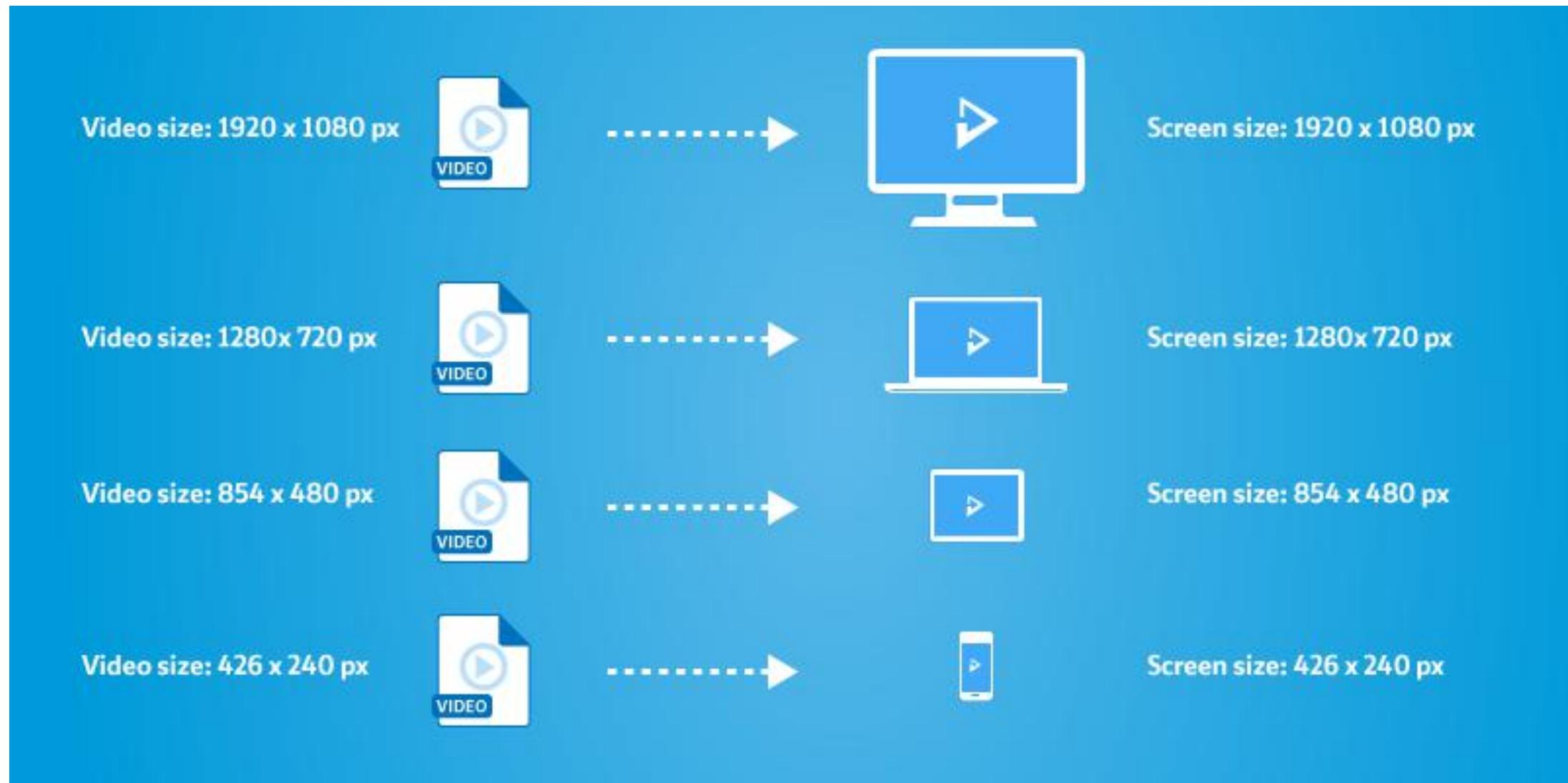
# Adaptive Video Streaming

- E.g., Dynamic Adaptive Streaming over HTTP (DASH) in Youtube
- Motivation: one size does not fit all

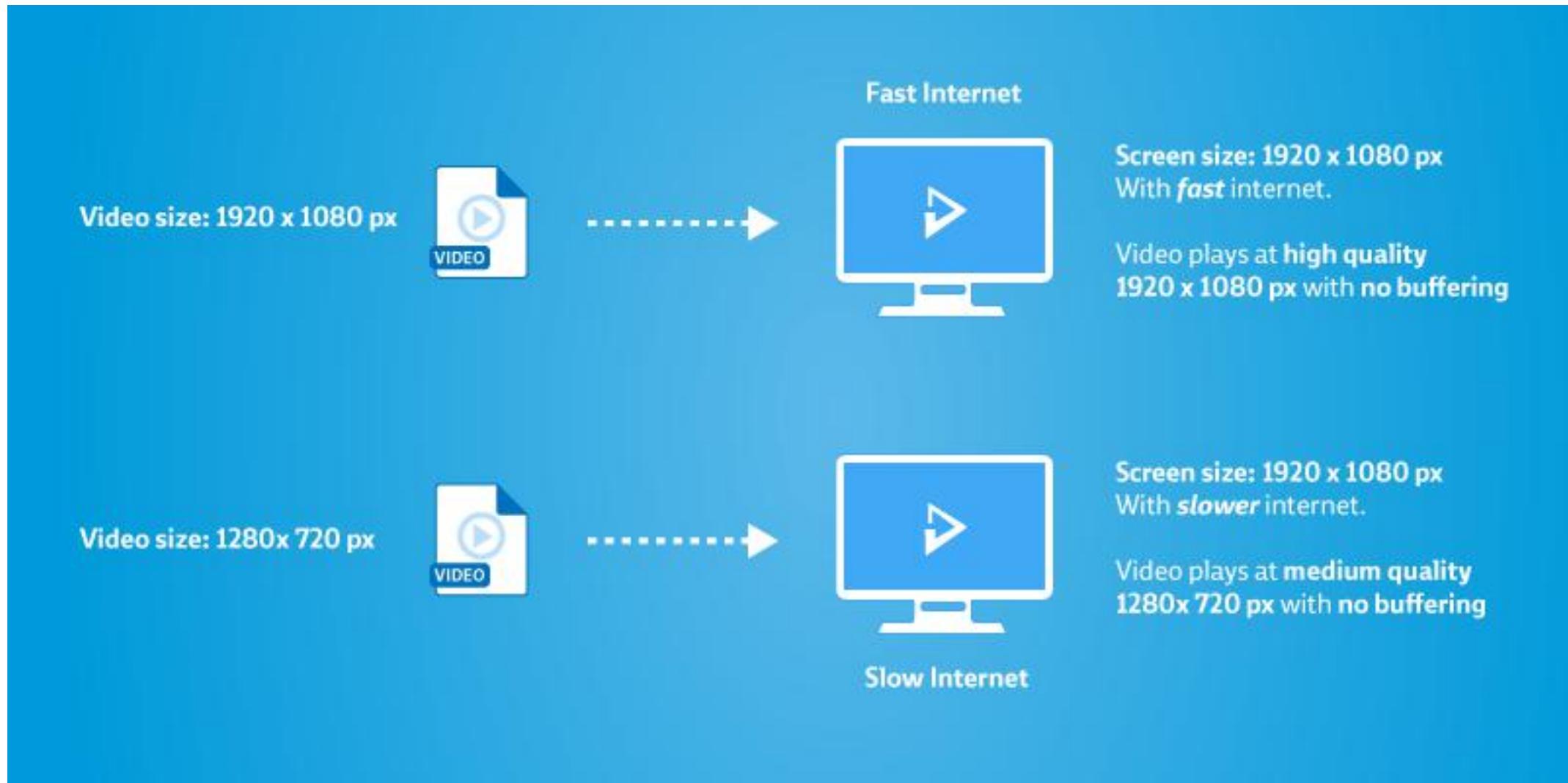
- Baseline: a progressive video stream is simply one single video file being streamed over the internet, then stretch to different screens.



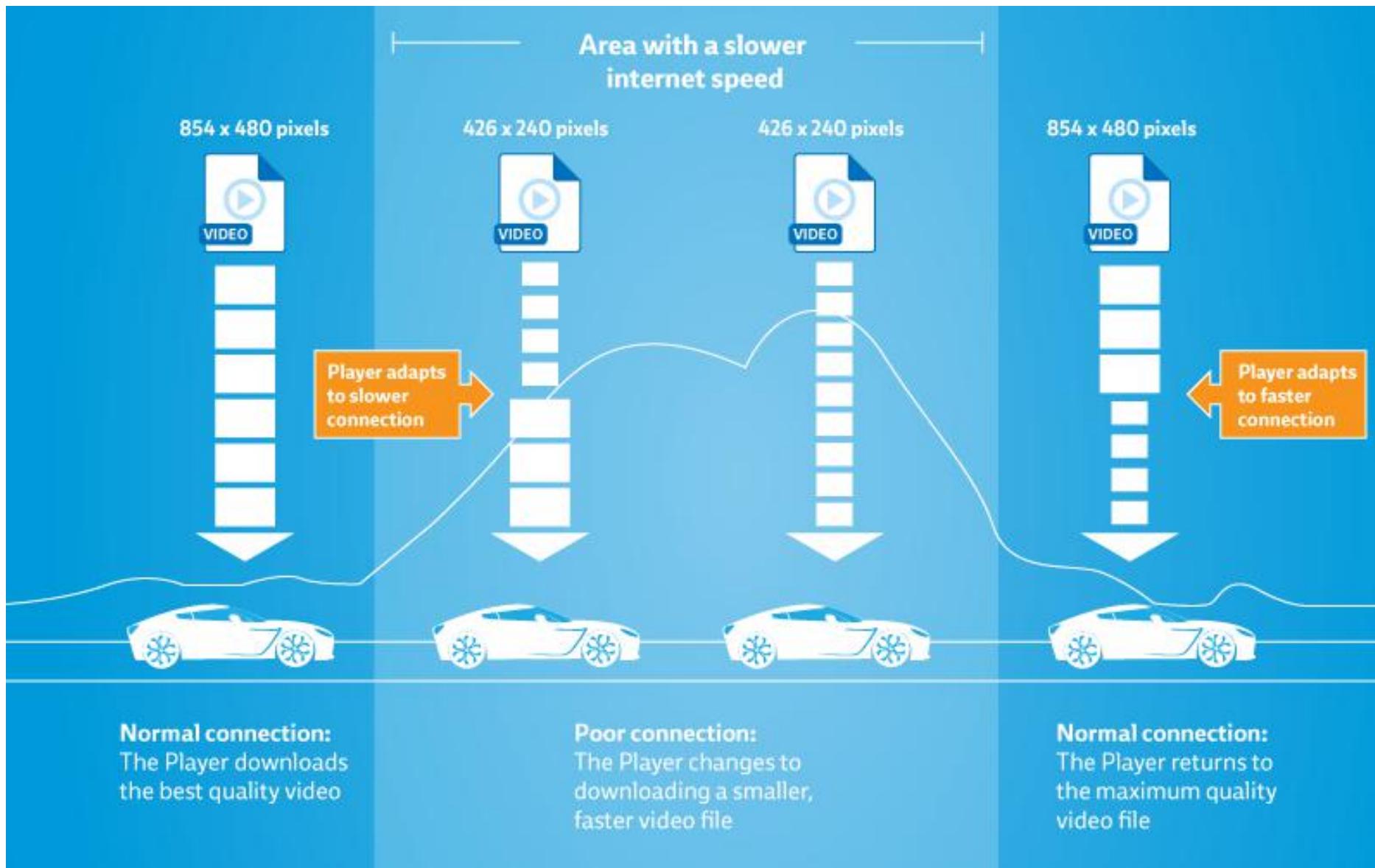
- Adaptive streaming: allows the video provider to create a different video for each of the screen sizes



- Adjust the video size based on network connection quality



- The biggest strength: adaptive bitrate



# Security, Privacy, and Ethics in Mobile Development

CSE 162 – Mobile Computing  
Hua Huang

Department of Computer Science and Engineering  
University of California, Merced

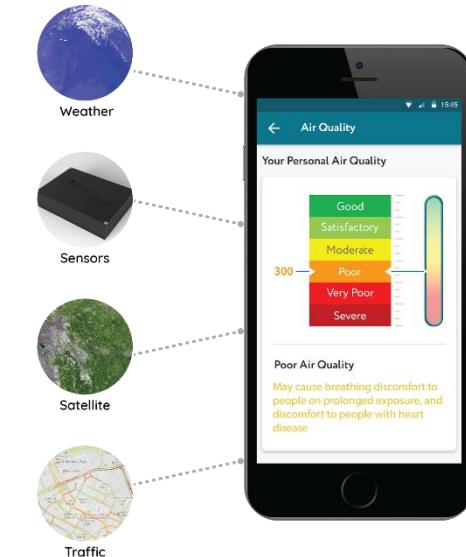
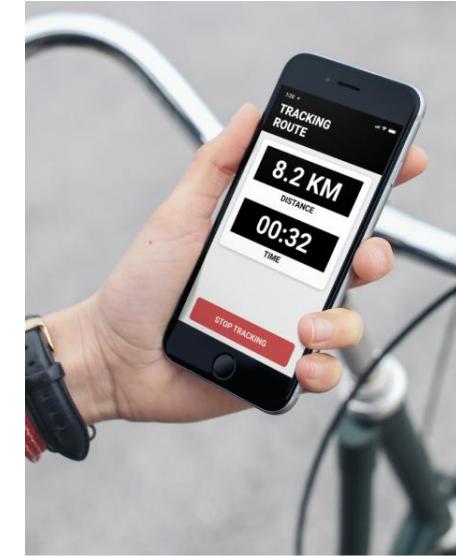
# Ethical Issues in Mobile Development

# What sensitive information can be collected by mobile systems?

- Video
- Sounds
- GPS
- Locations
- User's self reported information
- Many more!
- All with one unified API, and cheap

# Examples of Legitimate Apps

- Assess pedestrian or bike friendliness of neighborhoods.
- Use the location awareness to understand the user's exposure to air pollution as they move around.
- Use phones to snap, tag and upload photos of community events



# Ethical Issues in Mobile Sensing

- Privacy: control over personal data
- Consent: Informed permission
- Equity: fairness in how individuals are treated
- Social forgetting: purposeful discarding of information about individuals in order to enable forgiveness, recovery, or a clean slate.

# Privacy

- Mobile systems can gather significant amounts of data about the users.
- However, the data can be subpoenaed, or be demanded by U.S. authorities without warrant
- Unauthorized sharing or data theft can occur at a variety of places
- Complicated end-user licensing agreements may lead users to give away broad rights to share their data in return for services.

# Recap: Privacy Preserving in Crowdsensing

- Approaches to protect the data
- Examples:
  - **Anonymization**
  - **Encryption**
  - **Obfuscation:** Adding distracting or misleading data to a log or profile.

# Considerations: Privacy vs Functionality

- Commerce can suffer from strong privacy rights, as there is less information for both producers and consumers in the marketplace
  - Example electronic payments, and ant-money-laundering
- Truthfulness, openness, and accountability can suffer at the hands of strict privacy protections
  - Example: new algorithms for mobile sensing that allow users to replace sensitive location data with believable but fake data

# Consent

- Consent is a value central to data policies in the United States. A critical component of respect, beneficence and justice is informed consent.

# Challenges in Consent

- Consent in mobile apps is complicated by relying on ubiquitous devices.
  - Opting out of the mobile phone network is not a realistic option.
  - In 2011, Apple and Android were storing location data over and beyond what users were notified of and consented to
- Financial interest can conflict with consents
  - Use data to produce targeted advertising, sell valuable behavioral data to third parties, or use location to hone price or product discrimination
- Secondary, unforeseen purposes of data use
  - motion data can infer Parkinson's disease

# Soft Surveillance

- A technique used by agents of power, such as governments, to collect seemingly voluntary but actually mandatory data.
  - Example: searches to enter planes
  - Example: Withheld Social Security benefits if people do not “voluntarily” submit personal information
- Mobile sensing systems can easily become soft surveillance systems
  - Everywhere, all time presence
  - People can be involved by simply agreeing to data collection

# Equity

- Fairness and justice in how individuals are treated
- If powerful institutions gather data from relatively less powerful individuals, mobile sensing could tilt towards control and increased surveillance
- Alternatively, distributed sensing and analysis could shape technologies of care or even empowerment.
- Besides, the availability of mobile phones enables systematic data collection with radically lower cost, which enables data-driven decision-making to small institutions and community groups

# Social Forgetting

- Purposeful discarding of information to enable forgiveness and a clean slate
- Mobile sensing can create a record of people's movements, habits, and routines that persists
  - A subject of both celebration and concern

# Pros and Cons of persistent records

- Pros
  - Augment human memory. Can improve healthcare, and enable memory bank to relive past events.
- Cons
  - Unintended loss of fresh start
  - Increased surveillance

# Solution

- The “right to be forgotten”.
  - A combination of policies and technologies that allow for the gradual decay of digital data.

# More info

- EU General Data Protection Regulation
- California Consumer Privacy Act

# Privacy Policy

# Creating a Privacy Policy

- A privacy policy is a document created to go with a product (app, website, etc.) that describes how the product and company behind it will do the following with a customer or client's data:
  - Gather
  - Use
  - Disclose
  - Manage

# Creating a Privacy Policy

- Ask yourself some questions:
  - What data is collected?
  - How it is collected?
  - What you will/can do with it?
  - What will happen to it after X amount of time?
  - Is it anonymous?
  - Are there ads?
  - Is the data shared with another organization?
  - ... and more...

# You need a privacy policy because...

- You are collecting personal data
- You are using a third-party service
- Government regulations
- App Store regulations
- Risk alienating customers
- Open to lawsuits

# What's in a policy?

- **Information** - what personal information is being collected on the site
- **Choice** - what options the customer has about how/whether her data is collected and used
- **Access** - how a customer can see what data has been collected and change/correct it if necessary

# What's in a policy?

- **Security** - state how any data that is collected is stored/protected
- **Redress** - what customer can do if privacy policy is not met
- **Updates** - how policy changes will be communicated

# Example Policies

- Google: <https://www.google.com/policies/privacy/>
- Apple: <http://www.apple.com/legal/privacy/en-ww/>
- Facebook: <https://www.facebook.com/policy.php>
- Twitter: <https://twitter.com/privacy?lang=en>

# Example Policies

- Note that these are mainly in “regular, plain English!”
- Movement away from “legalese”
- Some privacy policies were automatically processed

# What does a privacy policy get you?

- Disclosure of what's going on
- A level of trust with developer
- Meeting requirements from publishers / government agencies

# Example

- Google Analytics is one of the most popular digital analytics software.
  - Allows you to analyze details about the visitors on your website and design strategy to improve business
- If you've enabled any Google Analytics Advertising features, you are required to notify your users:
  - What features you've implemented.
  - How you and third-party vendors use first-party
  - How visitors can opt-out of the Google Analytics Advertising

<https://support.google.com/analytics/answer/2700409?hl=en>

# Beyond Policies

- Writing down what you do is good...
- ... following it is even better
- Remember: privacy is not security
- The privacy policy says what you are collecting and what you plan to do
- And absence of this does not mean you shouldn't protect data you collect!

# Wearables (1)

CSE 162 – Mobile Computing

Hua Huang

Department of Computer Science and Engineering

University of California, Merced

# In this lecture

- What is wearable computing
- Wearable sensing technologies
- Challenges in wearable computing

- The term “wearable computing” is really built around any device that is attached or worn in some way
- So, the possibilities here are vast
  - Smartwatch
  - Continuous health monitoring
  - Brain-computer interface
  - More

# Everyday Realistic Wearable Technology

- What most people think about is watches and/or bracelets
- Smart Watches
  - Apple Watch - <https://www.apple.com/watch/>
  - Android Wear - <https://www.android.com/wear/>
  - Some proprietary watches
- Personal Fitness Devices
  - Fitbit - <https://www.fitbit.com/home>
  - Other devices

# What is a wearable computer?

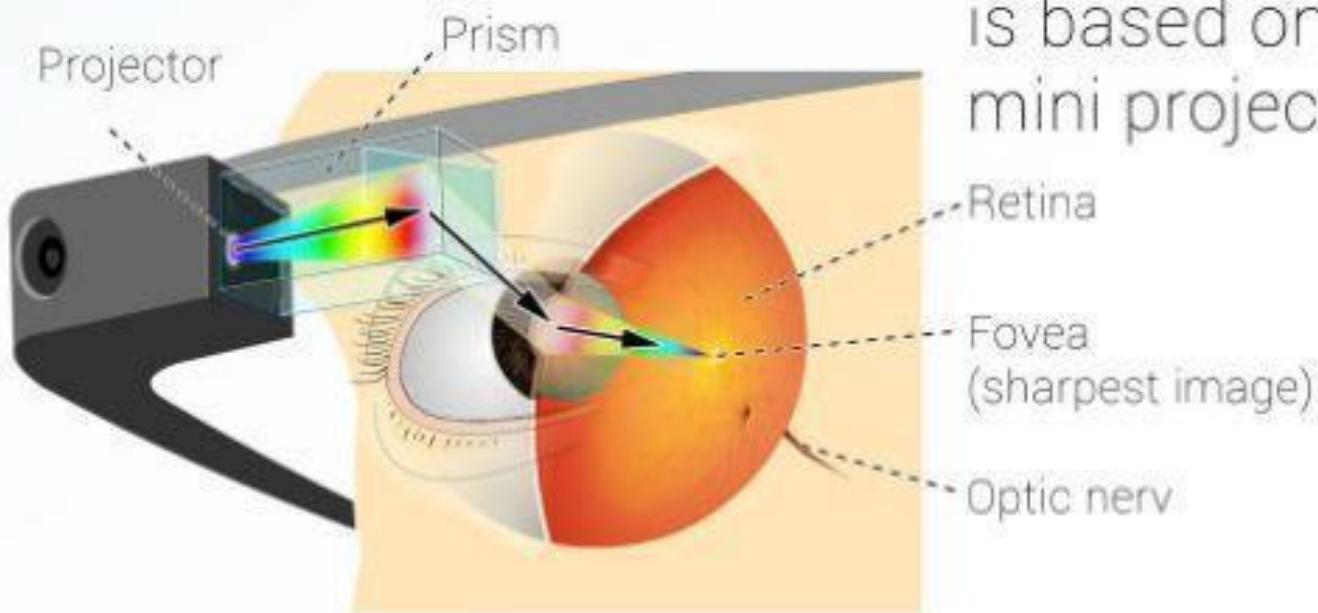
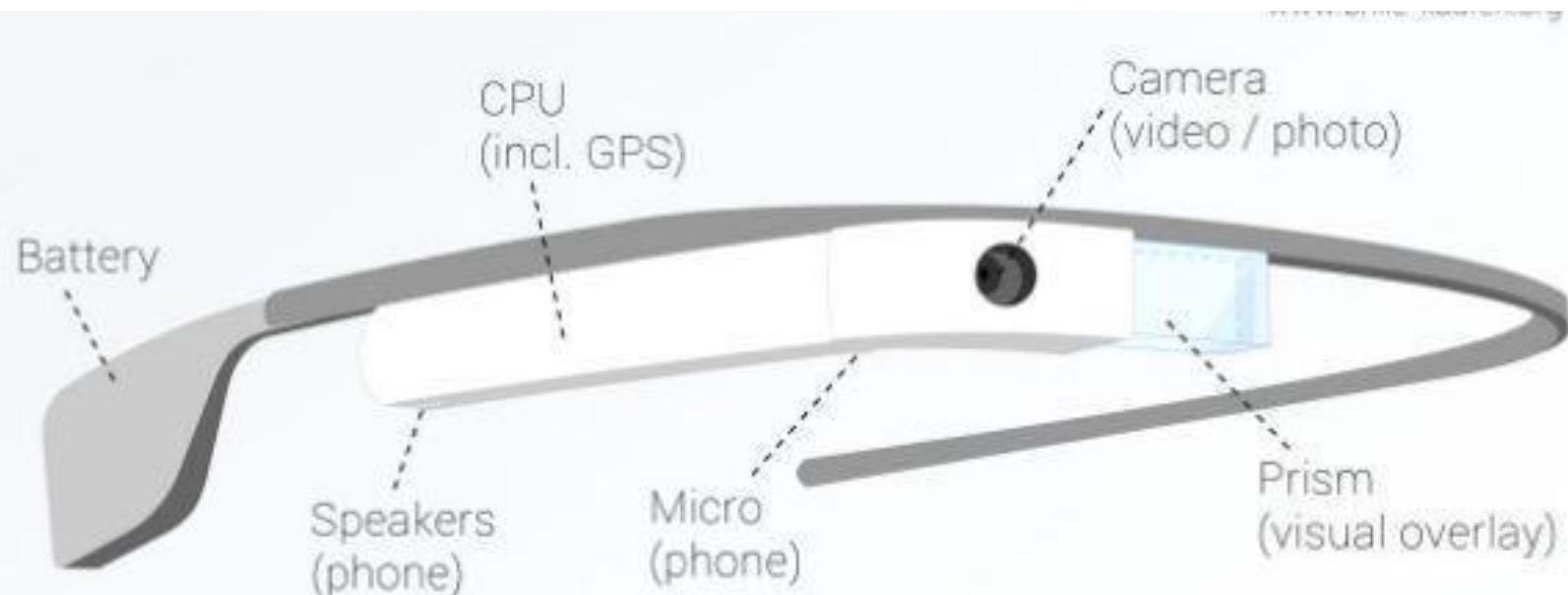
- A computer that is
  - Portable while operational
  - Enables hands-free/hands-limited use
  - Able to get the user's attention
  - Is always on, acting on behalf of the user
  - Able to sense the user's current context

Rhodes, B. J. (1997). The wearable remembrance agent: A system for augmented memory. *Personal Technologies*, 1(4), 218-224.

# Wearable Devices

# Smart glasses

- Head Mounted Display (HMD)
- Head Up Display (HUD)



The main function is based on a mini projector.

# View Through Google Glass

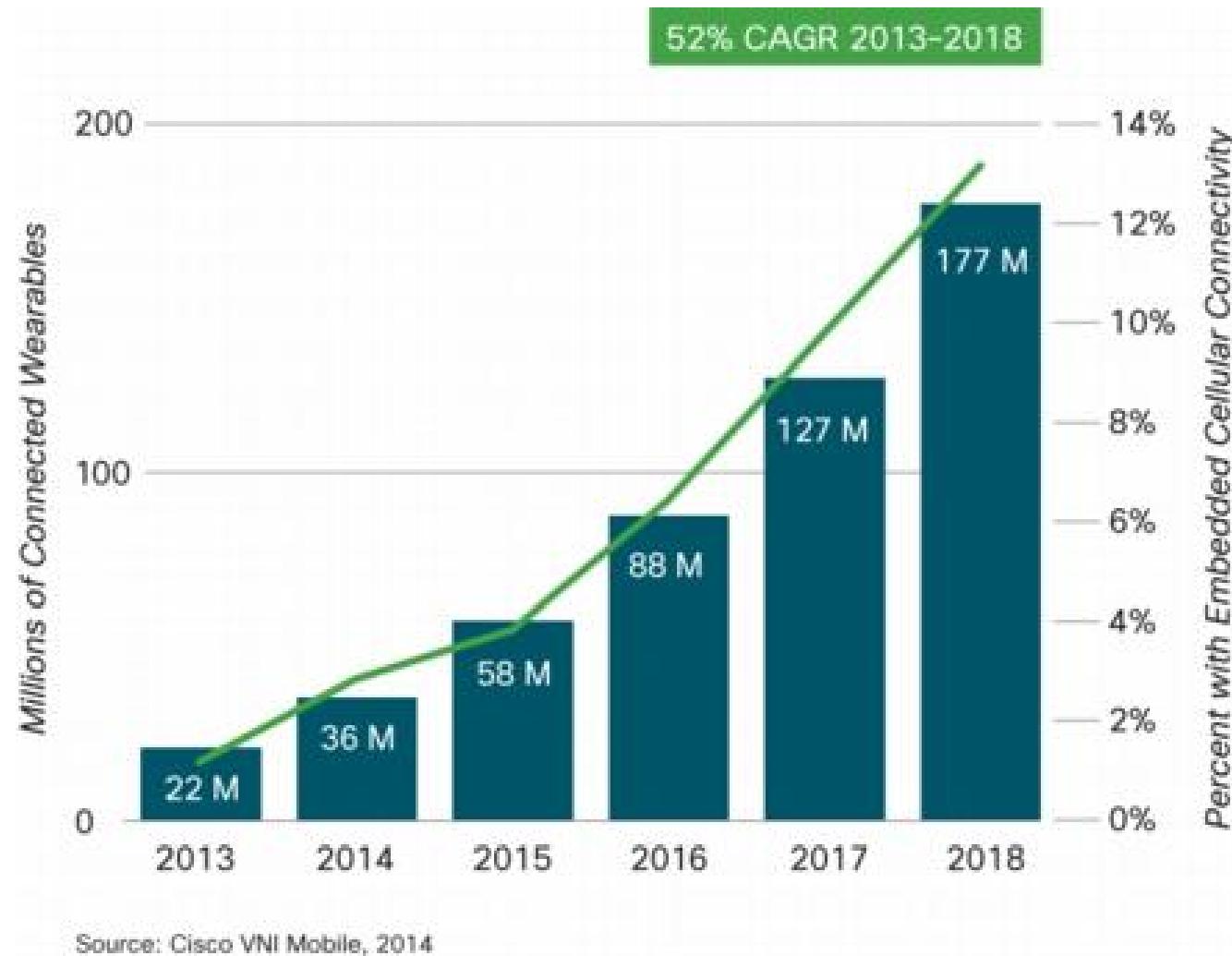
- Always available peripheral information display
  - Combining computing, communications and content capture



# Smart watches



# Number of Smartwatches Shipped



# The Predicted Wearables Boom Is All About The Wrist

Worldwide wearable device shipment forecast (in million units)



@StatistaCharts

Source: IDC

# Smart earplugs

- An emerging computing platform
- New features
  - Augmented acoustic reality
  - Real-time translation
  - Monitor biometrics
  - Fitness coaching
  - Biometric identification



# Intra-oral sensing

- “Sensor-Embedded Teeth for Oral Activity”

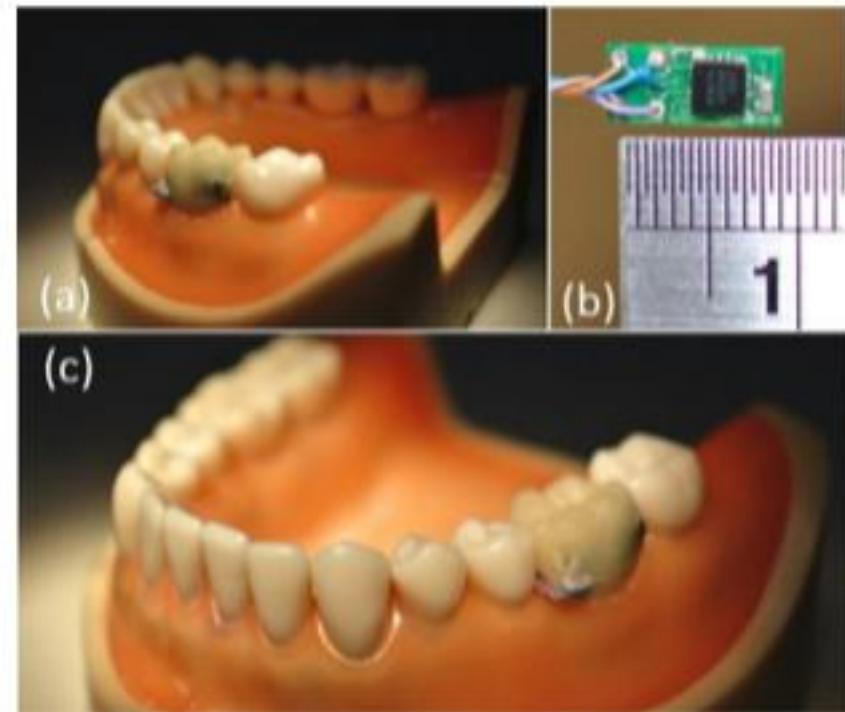


Figure 1. The breakout board with (b) tri-axial accelerometer and (a)(c) sensor embedded denture.

# Why discuss this in mobile?

- They are, by definition, mobile devices
- They often run versions of the operating systems we are already working with (Android Wear  $\Leftrightarrow$  Android)
- The wearable devices often work best when paired with a phone or other mobile device
- Many wearable apps come as part of a pair with a phone version

# Ideal Attributes

- Persist and provide constant access to information services.
  - Everyday and continuous use.
  - Wearable can interact with the user at any given time.
  - The user can access the wearable quickly and with little effort.
- Sense and model context.
  - The wearable can observe and model the user's environment, physical and mental state.
- Adapt interaction modalities based on the user's context.
  - The wearable should adapt its input and output modalities automatically to those that are most appropriate and socially graceful at the time.

# Why use wearable computers?

- Some people wear too many computers.
  - PDA, cellular phone, pager, laptop, electronic translator, and a calculator.
  - Mp3 player, audio digitizers, digital camera.
- These devices all contain very similar components.
  - Microprocessor, memory, screen, keyboard, battery, and in some cases, a wireless modem.
  - The main distinctions between these devices are the interface and the application software.
- Wearable computers could exploit the commonality in components to eliminate cost, weight and redundancy.

# Mediate interactions

- Wearable computers will help provide a consistent interface to computationally augmented objects in the physical world.
  - Example—Gesture Pendant.
  - One gesture could provide an intuitive command for many devices.

# Aid communication

- The wearable can also assist in human-to-human communication.
- Wearable computers can also help manage interruption in the user's daily life.

# Augment reality

- Augmented reality overlays information-rich virtual realities onto the physical world.
- In a sense, augmented reality is a combination of the application domains described previously.

# Wearable App Development

# Android Wear

- Released in 2014
- Android Wear devices were initially designed to be second screens, but can be used independently in more situations
- Like Android itself, Wear can be adapted to many different devices
- Uses the same UI theme that the rest of Android uses

# Use Cases

- Telling time
- OK Google
- Notifications and quick replies
- Phone app control (like music, for example)
- Fitness
- Basic functions (alarms, stopwatch, etc.)
- Mobile Payment

# When do you want a watch app?

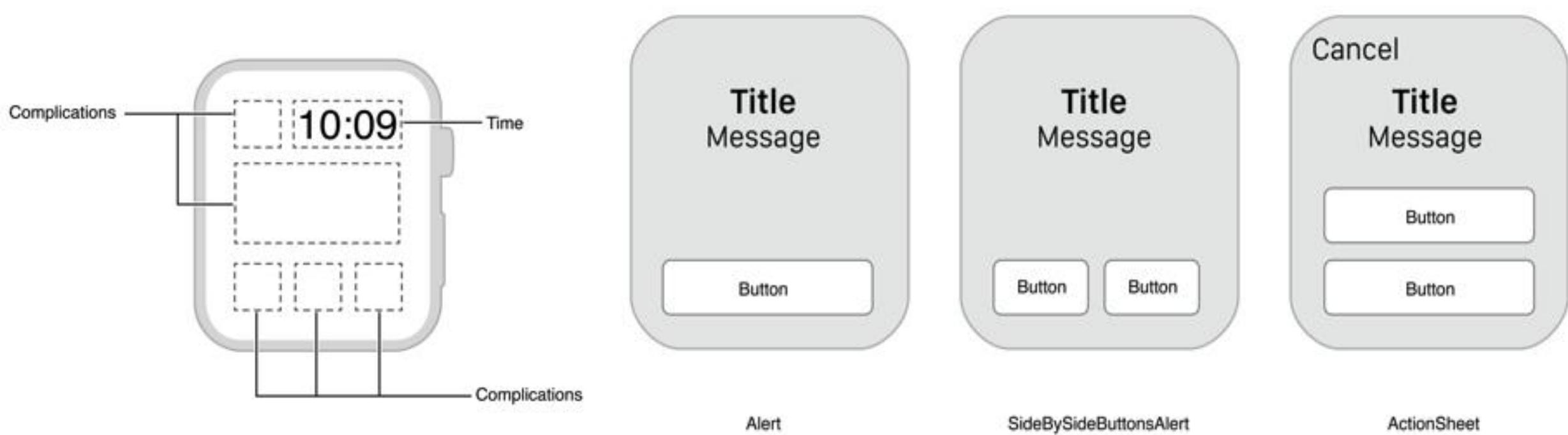
- Good Watch Apps
  - Buy Me a Pie (grocery list)
  - Seven Minute Workout (works with sensors)
  - Due (reminders)
  - Google Maps (directions on your wrist!)
  - Pandora (control your music)
  - Shazaam (what is that song?)
  - Weather Underground (quick weather forecast)

# When do you want a watch app?

- Odd Watch Apps
  - Chipotle?
  - FlightRadar?
  - Fandango?
  - AAA?
  - Amazon?

# Questions to Consider

- UI concerns regarding touch size and screen size become even more problematic...



# Questions to consider

- Would a watch app add anything to my full app?
  - What additional sensing can the watch provide?
  - Can the information be shown in a very small format?
  - Are there simple controls to the app that could be added to a watch?
- What type of interaction do you want the user to have?

# Energy Efficiency in Mobile Computing

# Heat dissipation in wearables

- Close proximity of the wearables to the human body
  - Airflow is low, limiting cooling
  - Strict limitation on temperature
- Careful design of software necessary to help avoid many heat generation crises.

## Fitbit Recalls Ionic Smartwatches Due to Burn Hazard; One Million Sold in the U.S.

Share:    



- 115 reports of the watch's battery overheating in the US, and 59 internationally.
- There were 78 reports of burn injuries in the US, including two reports of third-degree burns, and four reports of second-degree burns.
- 40 burns were reported internationally.

# Problem: Battery Power is Scarce

- Battery energy is most constraining resource on mobile device
- Most resources (CPU, RAM, WiFi speed, etc) increasing exponentially *except* battery energy (ref. Starner, IEEE Pervasive Computing, Dec 2003)

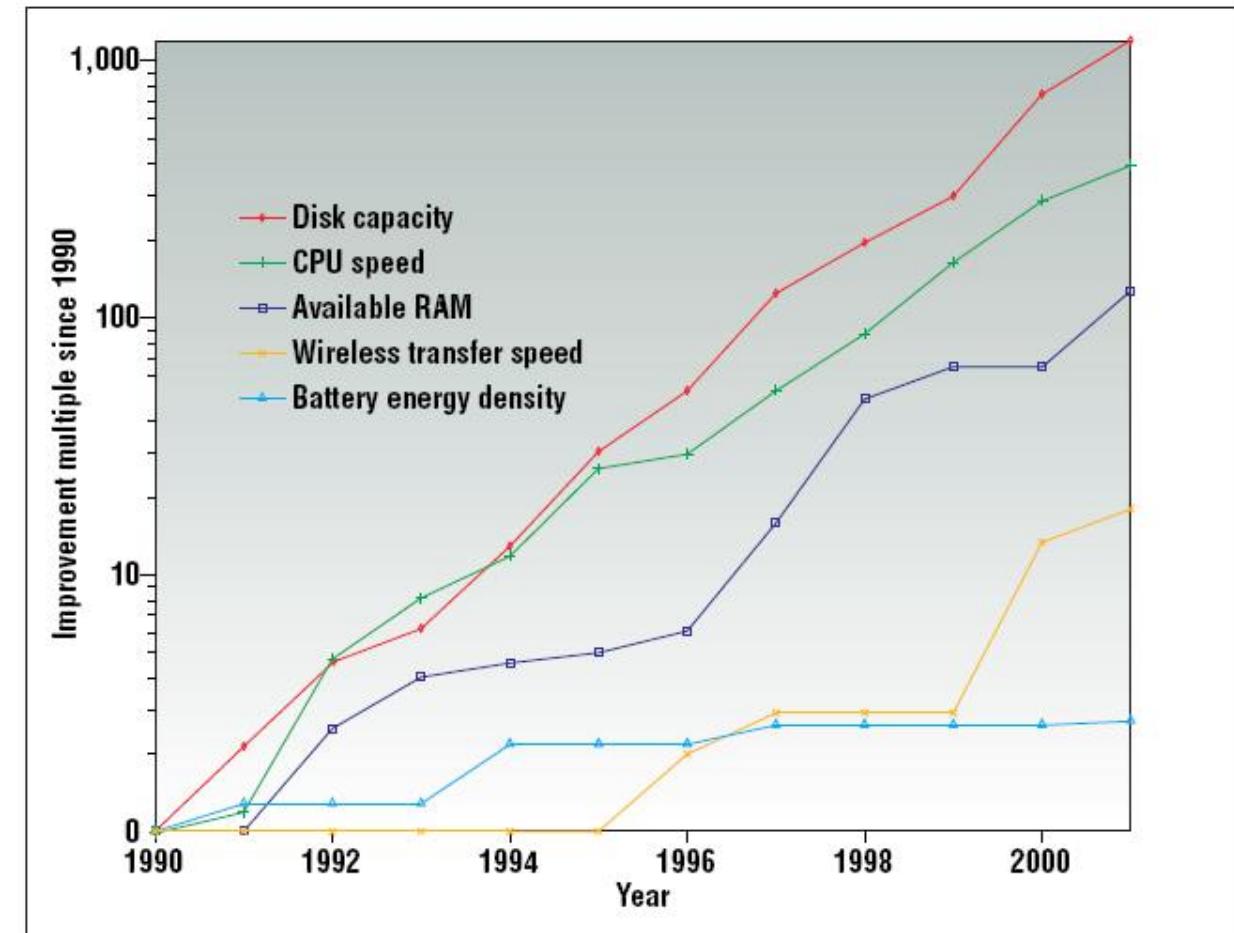


Figure 1. Improvements in laptop technology from 1990–2001.

# Ideal wearables (Recap)

- Persist and provide constant access to information services.
- Sense and model context.
- Adapt interaction modalities based on the user's context.

# Ideal wearables: challenges

- Persist and provide constant access to information services.
  - Challenge: supply the energy to support all-time access
- Sense and model context.
  - Challenge: power enough sensors
- Adapt interaction modalities based on the user's context.
  - Challenge: power sufficient computing capacity for context learning

**Summary: battery capacity continues to bottleneck the ideal wearables**

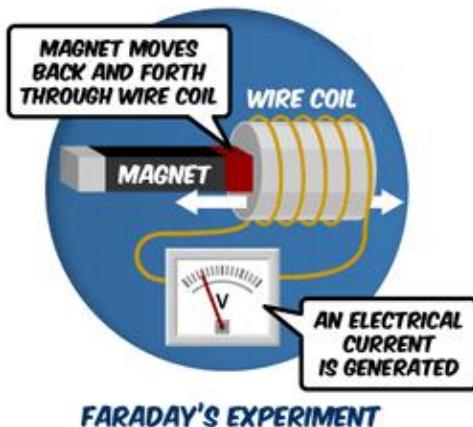
# Energy Harvesting for Wearable Systems

# Energy Harvesting

- What is it?
- Macro scale- wind and solar
- Micro scale- electromagnetic and piezoelectricity

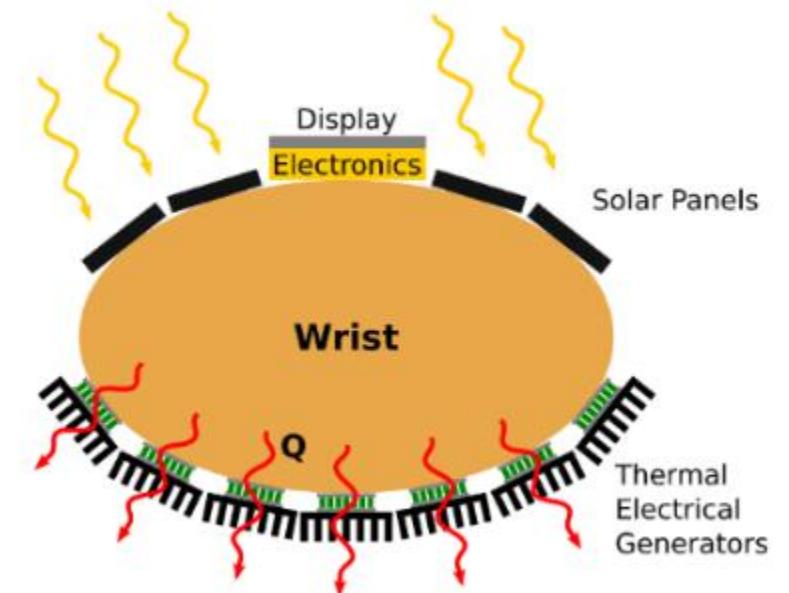


[3]



# Thermal Electrical Generator

Havest body heat and convert it to electricity



Texas Instruments Innovation Challenge: Europe Design Contest 2015

# Self-Sustainable Smart Wearable Device with Energy Harvesting for Context Recognition Applications

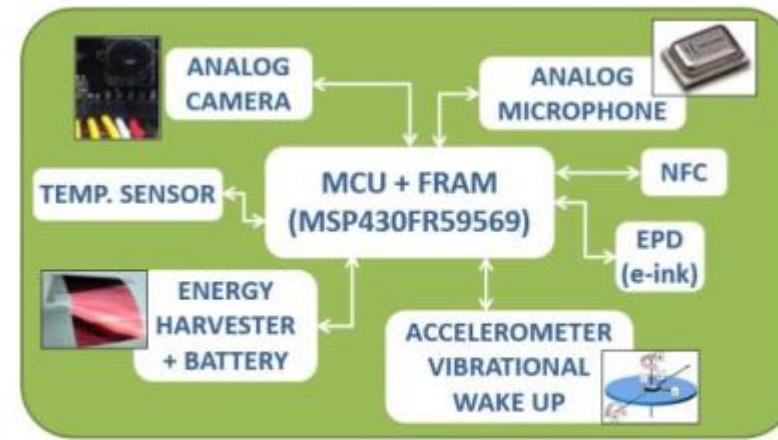
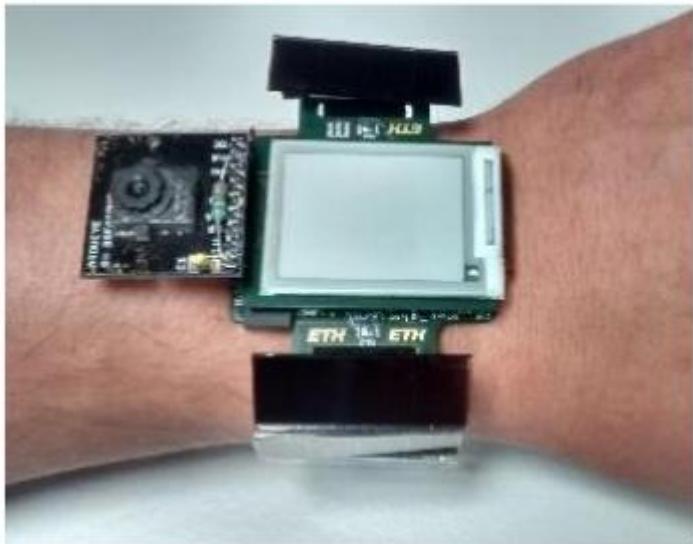
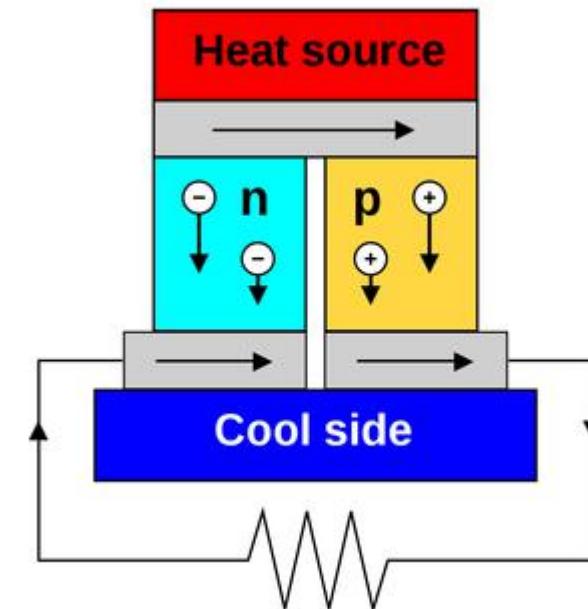


Fig. 2: Block diagram of the smartwatch

- A thin conductive material takes advantage of the temperature difference between its two sides to produce electricity.
- This is known as the Seebeck effect.



- If a thermoelectric device is located on skin, it will produce power as long as the ambient air is at a temperature that is lower than the skin.
- A device that is one square centimeter in area can yield up to 30 microwatts.
- If these generators are located side by side, the amount of power being harvested is increased.



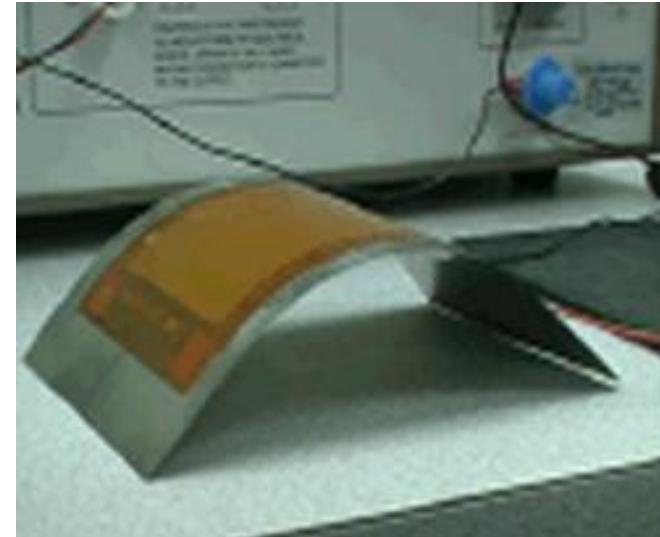
# Piezoelectricity

- Deform to get voltage



[4]

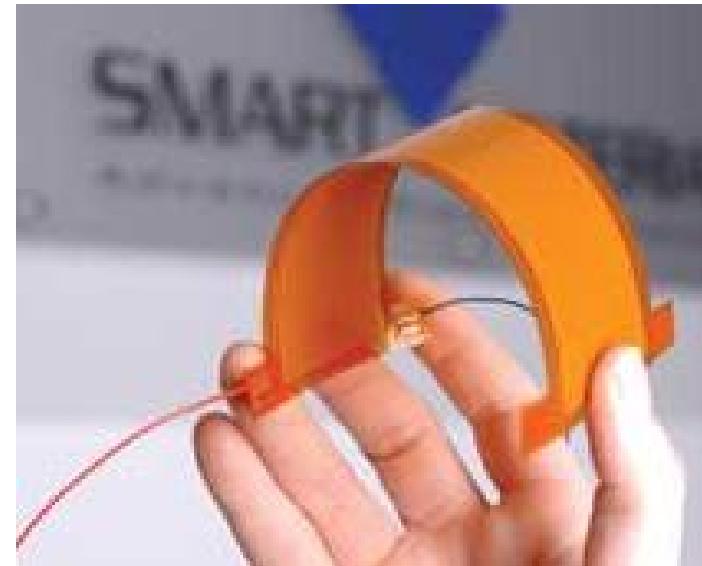
- Apply voltage to deform



[5]

# Piezoelectricity

- What is it?
- Occurs in polycrystalline materials
  - (For example: Quartz, lead-zirconate-titanate, Rochelle salt, etc.)



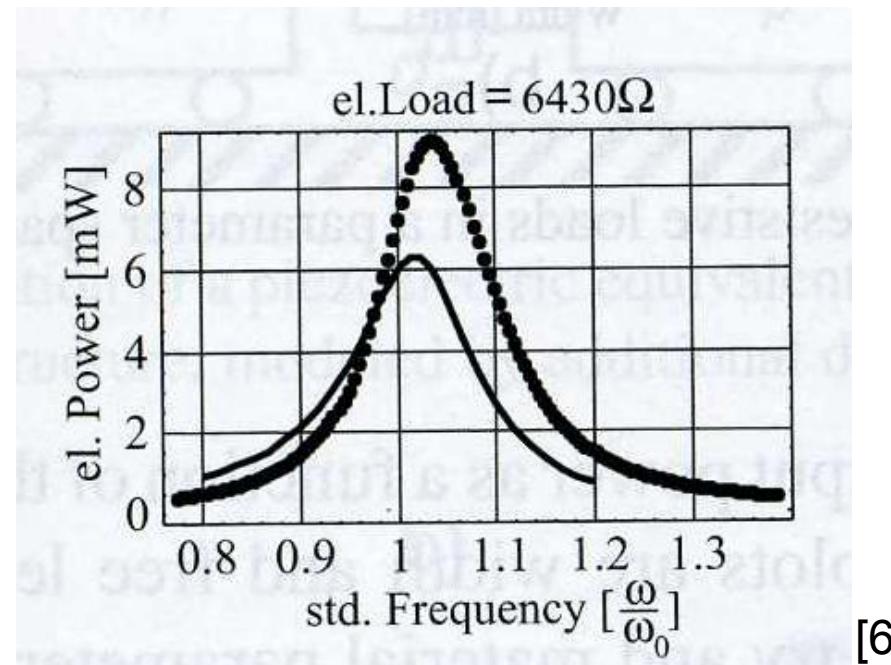
[4]

# Piezoelectricity- Output

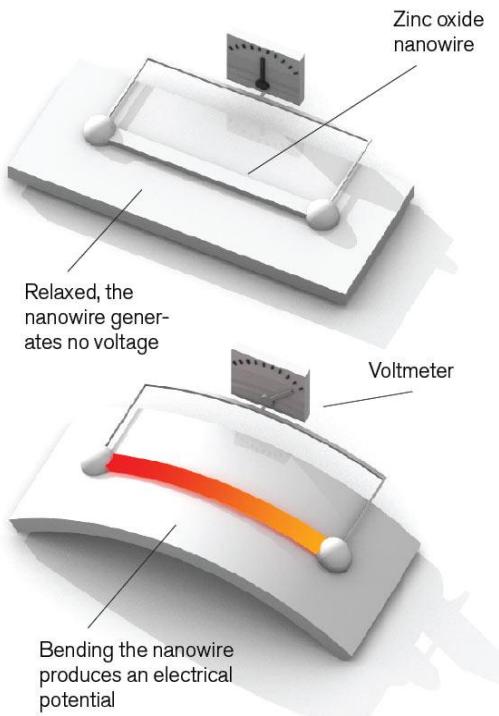
Output always different

High voltage, low current

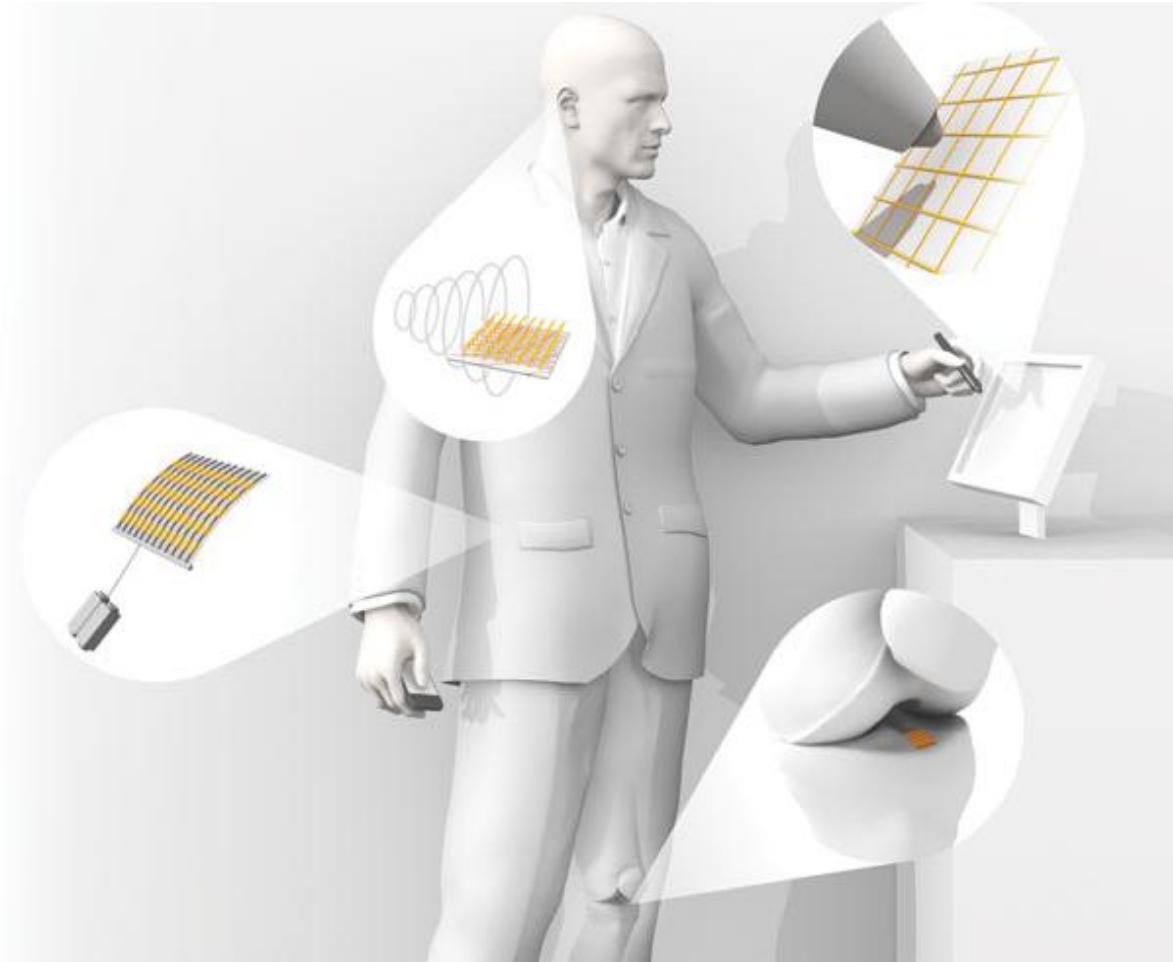
Sample output:



# Future Outlook



[7]



# Power Management in Android

# Android Power Management Design

- Apps and services must request CPU resource with ‘wake locks’ through the Android application framework to keep power on, otherwise Android will shut down the CPU
- Android PM uses wake locks and time out mechanism to switch state of system power, so that system power consumption is managed.

# Wakelock

- By default, Android tries to put the system into a sleep or better a suspend mode as soon as possible
- Some applications need to assure that the screen stays on or the CPU stays awake to react quickly to inputs
  - To do so, we use wakelock
  - Without active wake locks, the CPU will be turned off
  - Partial wakelocks can turn off screen or keyboards.

# Android Wakelock Architecture

- Wakelocks are power-managing software mechanisms that make sure that your Android device doesn't go into deep sleep
- Because a given application needs to use your system resources.
- Class PowerManager.WakeLock is mechanism to indicate that the application requires the device to stay on.

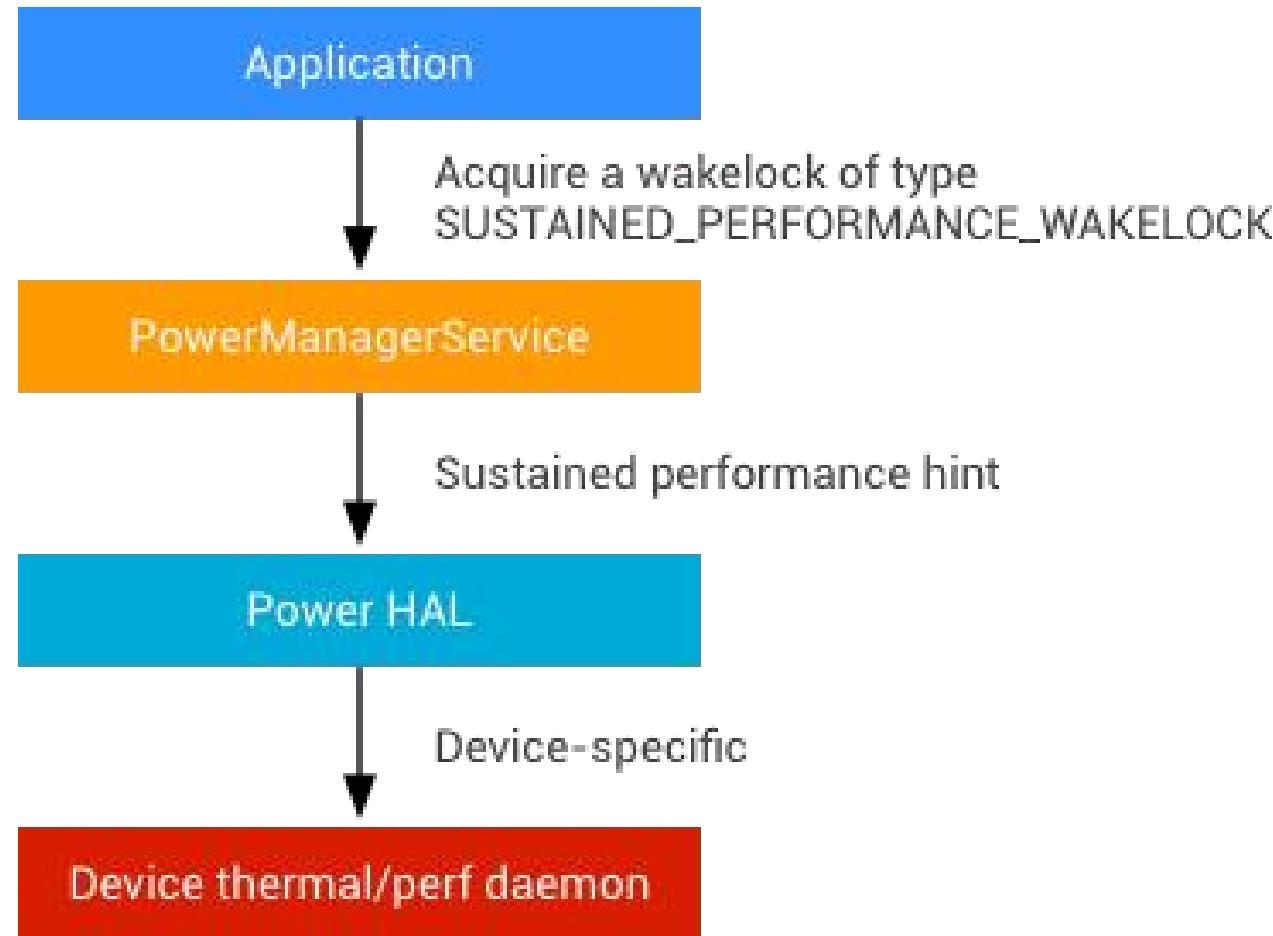
# Wakelock Levels

Experiment with it in your upcoming camera programming lab

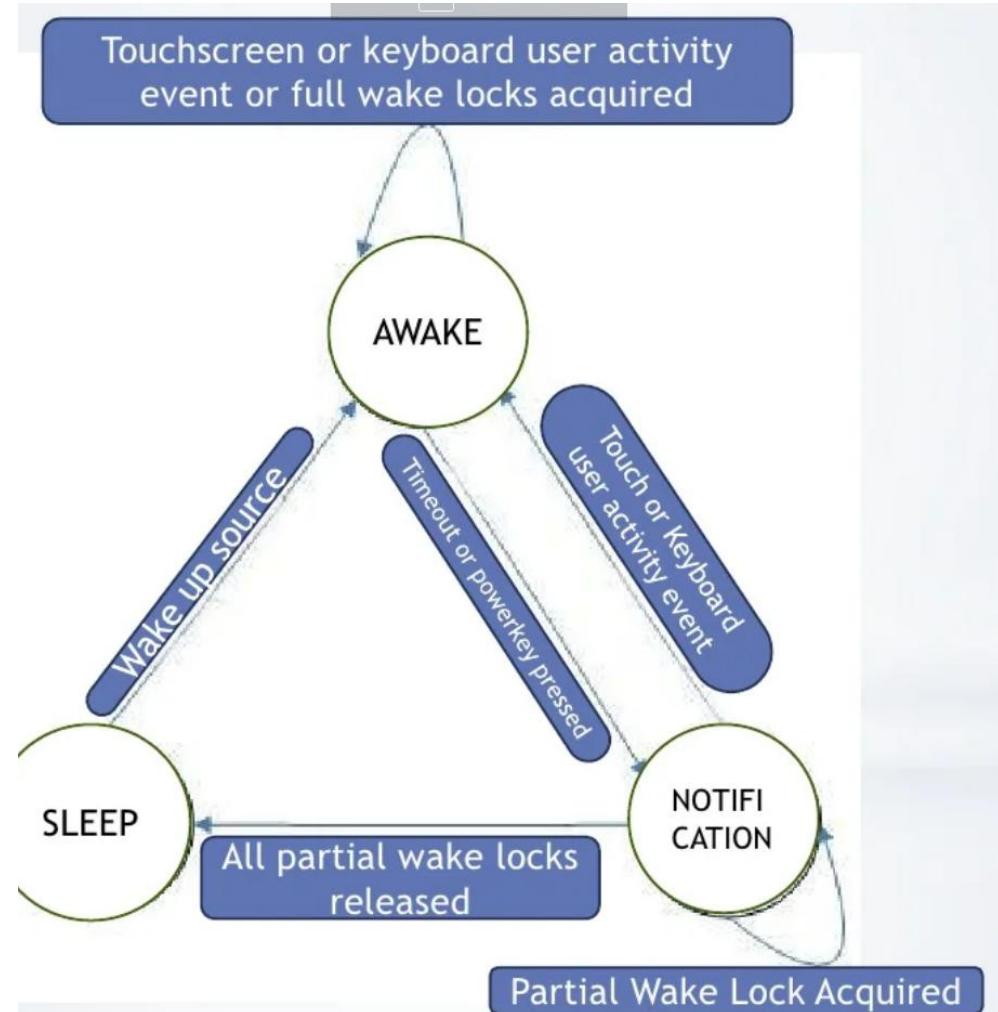
Flag Value	CPU	Screen	Keyboard
PARTIAL_WAKE_LOCK	On	Off	Off
SCREEN_DIM_WAKE_LOCK	On	Dim	Off
SCREEN_BRIGHT_WAKE_LOCK	On	Bright	Off
FULL_WAKE_LOCK	On	Bright	Bright

# Android PM Design

The PowerManager class can control the power state of the device

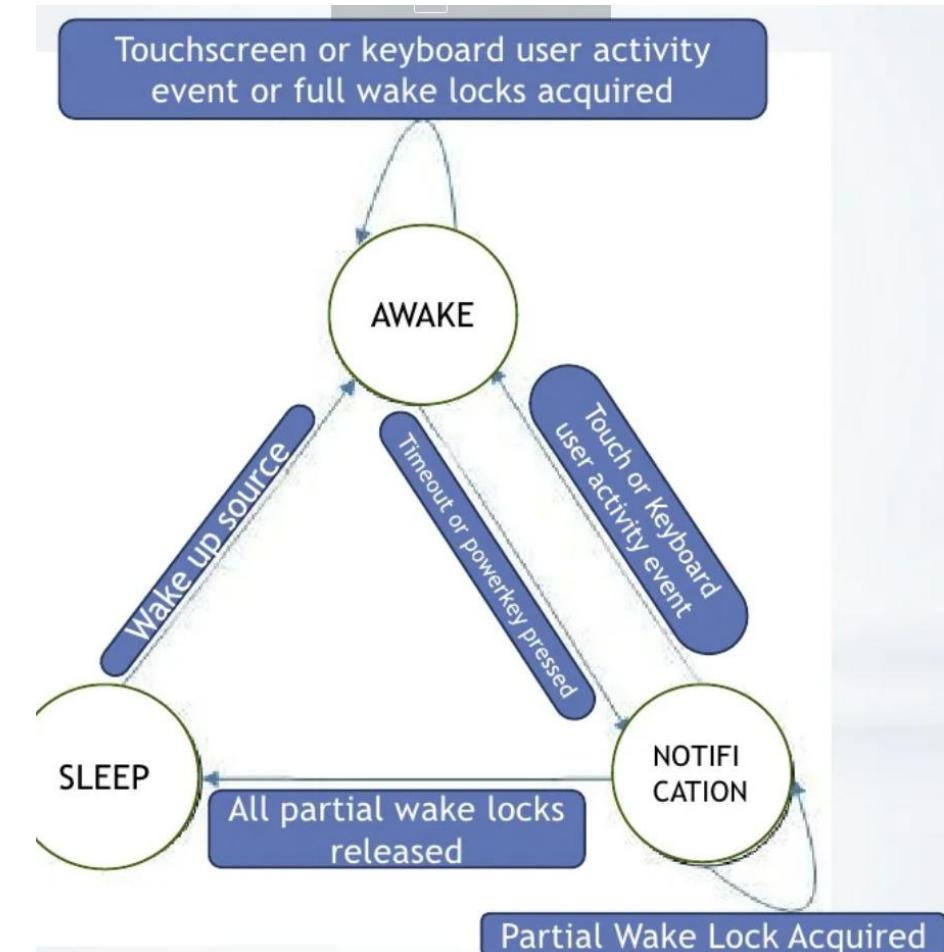


# States for Android Power Management



# States for Android Power Management

- When an application acquires full wake lock or an UI activity event occurs, the machine enters ‘wake’ state
- If timeout happens, the machine enters ‘notification’ state
  - If partial wake locks are acquired, it will remain in ‘Notification’
  - If all partial locks are released, the machine go into ‘sleep’



# System Sleep

- API to bring device to sleep when we press the power button
- Require DEVICE\_POWER permission
- Can only be called in system process context by verifying UID and PID
- When the power button is pressed, an API goToSleep() is called in the PowerManager
- goToSleep() will force release all wake locks

# Software Techniques for Battery Conservation in Mobile Devices

- Three things to help
  - Make apps *Lazy First*
  - Take advantage of platform features
  - Use tools to identify battery draining components

# Lazy First

- **Reduce**
  - Are there redundant operations your app can cut out?
- **Defer**
  - Does an app need to perform an action right away?
- **Coalesce**
  - Can work be batched, instead of putting the device into an active state many times?

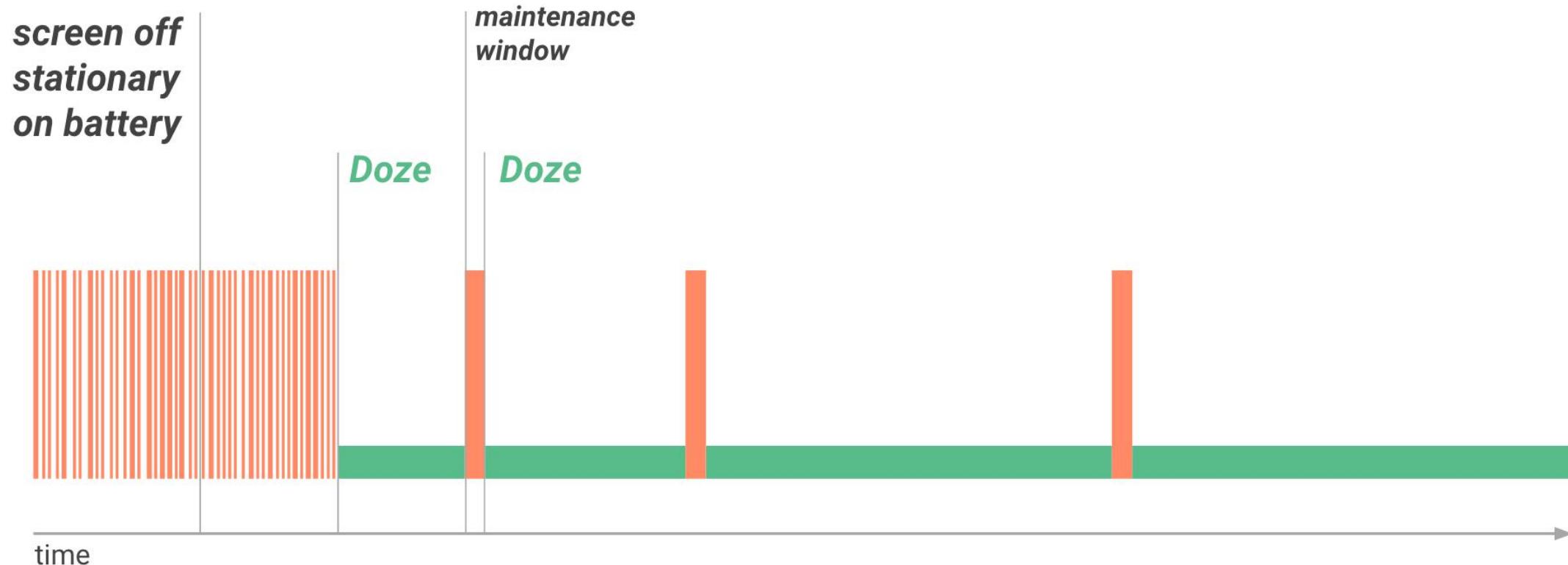
- Three things to help
  - Make apps *Lazy First*
  - **Take advantage of platform features**
  - Use tools to identify battery draining components

# Android platform features

- **Doze and App Standby:**
  - Two power saving features
  - Manage how apps behave when a device is not connected to a power source
- Doze
  - Defers background CPU and network activities when the device is unused for a long time
- App Standby
  - Defers network activity when user has not recently interacted

# Doze Mode

- Starts when a device is **unplugged** and **stationary** for a **period of time**, with the **screen off**
- Periodically, the system exits Doze to let apps complete their deferred activities, such as running pending syncs, jobs, alarms, and network access



# Doze Mode

- Apps on your phone will have no network access, the system will ignore “wakelocks” when apps try to keep the device from going to sleep, and no background tasks will be allowed to run.
  - High-priority push messages will still show up.
  - So for example, a Hangouts message will appear on a device that’s in Doze mode.

# Android App Standby

- The system puts an app into Standby mode and defers its network access if for a certain period of time,
  - the user has not explicitly launched the app, and
  - the app doesn't have a foreground process (activity), and
  - the app doesn't generate notifications, and
  - the app is not an active device admin app.
- When the phone is plugged to power, apps are released from standby state
- If the device is idle for a long period of time, idle apps access network around once a day

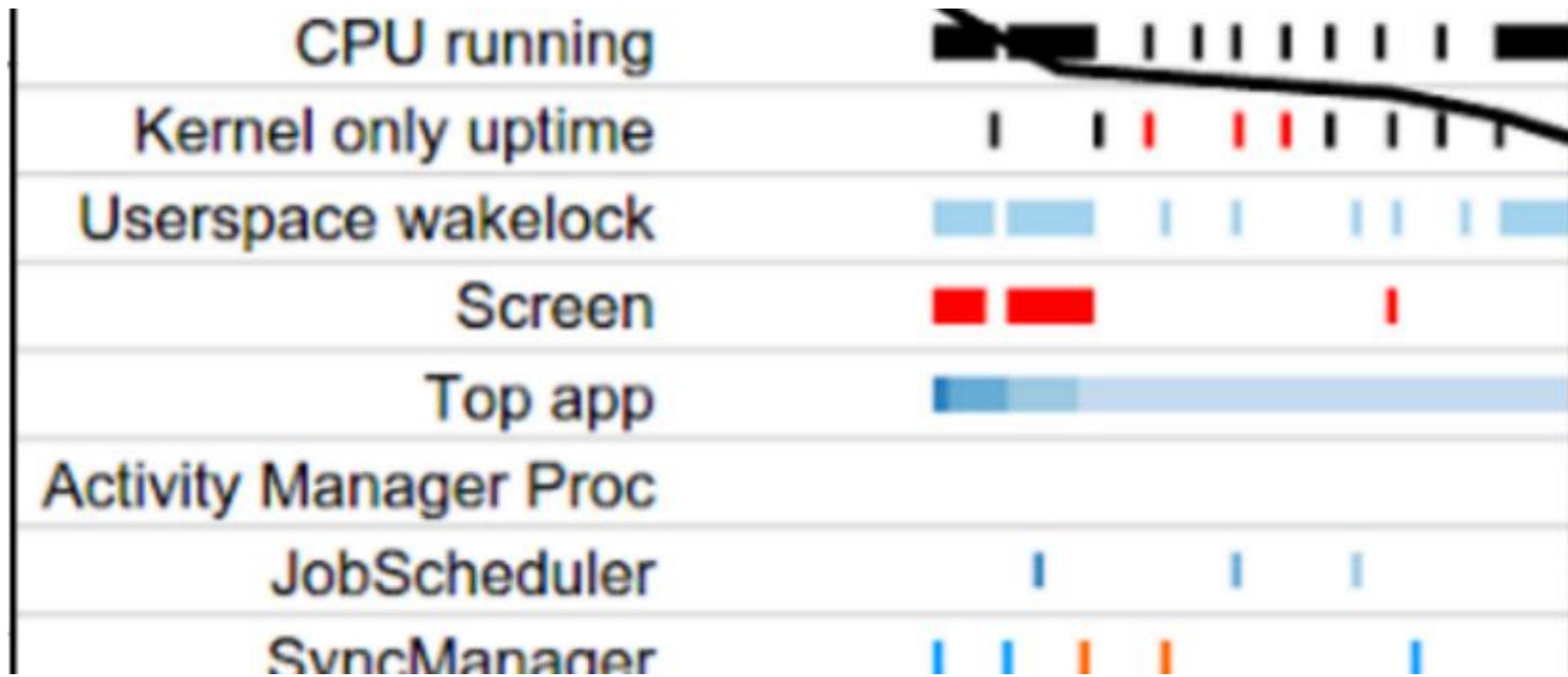
- Three things to help
  - Make apps *Lazy First*
  - Take advantage of platform features
  - **Use tools to identify battery draining components**

# Battery Historian: A tool to analyze power use in Android

provides a system-wide visualization of various app and system behaviors, along with their correlation against battery consumption over time.



# A closeup look



# Battery historian: App-specific view

- Provides tables for the following data:
  - The app's estimated power use on the device.
  - Network information.
  - Wakelocks.
  - Services.
  - Process info.

# Display the ranking of app powers

Sort apps by

Device estimated power use

com.curlytail.pugpower (Uid: 10372)

## Tables

▼ System Stats

Aggregated Checkin Stats

Sorted by Device estimated power use

Device's Power Estimates

Userspace Wakelocks

SyncManager Syncs

Mobile Radio Activity Per App

Seems  
suspicious

Device's Power Estimates:	
Show 10 entries	
Ranking	Name
0	OVERCOUNTED
1	ANDROID_SYSTEM
2	ROOT
3	SYSTEM_UI
4	CELL
5	com.fungames.pokerific
6	com.android.chrome
7	SCREEN
8	com.curlytail.pugpower

# Other cases where Battery Historian can help

- Examples:
  - Firing wakeup alarms overly frequently (every 10 seconds or less).
  - Continuously holding a GPS lock.
  - Scheduling jobs every 30 seconds or less.
  - Scheduling syncs every 30 seconds or less.
  - Using the cellular radio more frequently than you expect.

# Mobile Security, Malwares, and App Stores

CSE 162 – Mobile Computing  
Hua Huang

Department of Computer Science and Engineering  
University of California, Merced

# **Mobile Security**

# Introduction

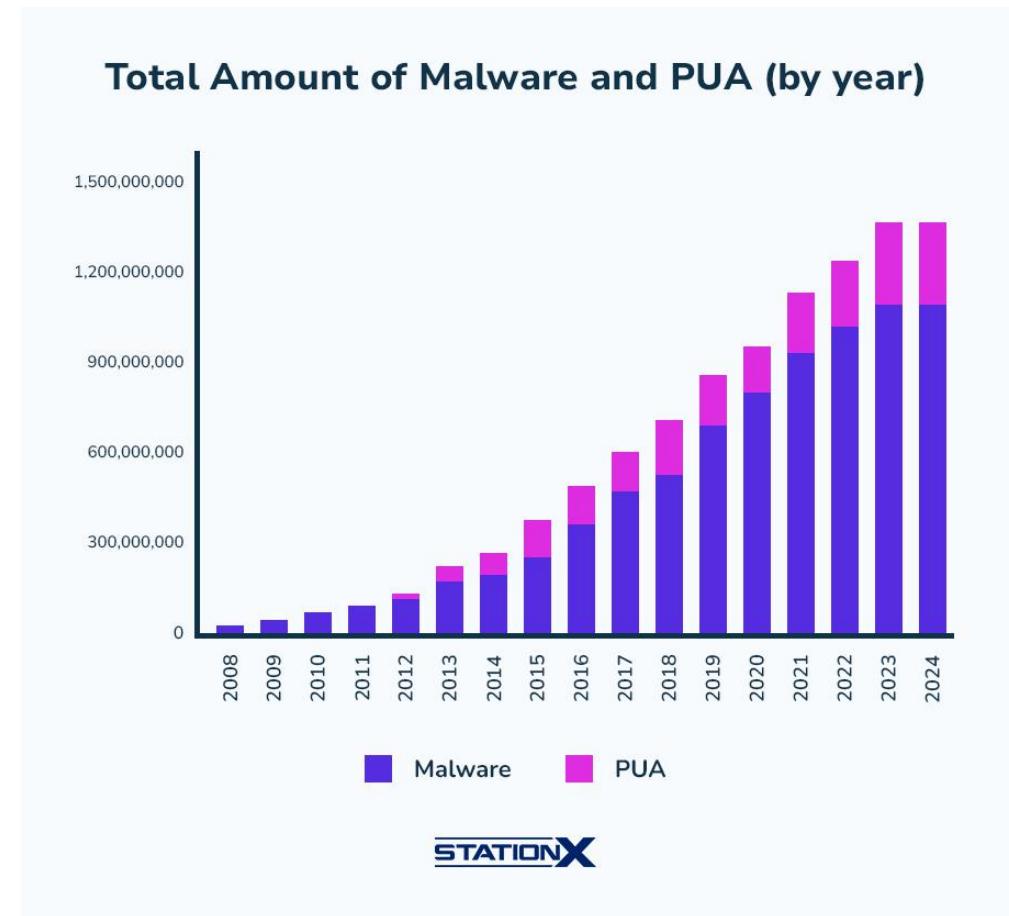
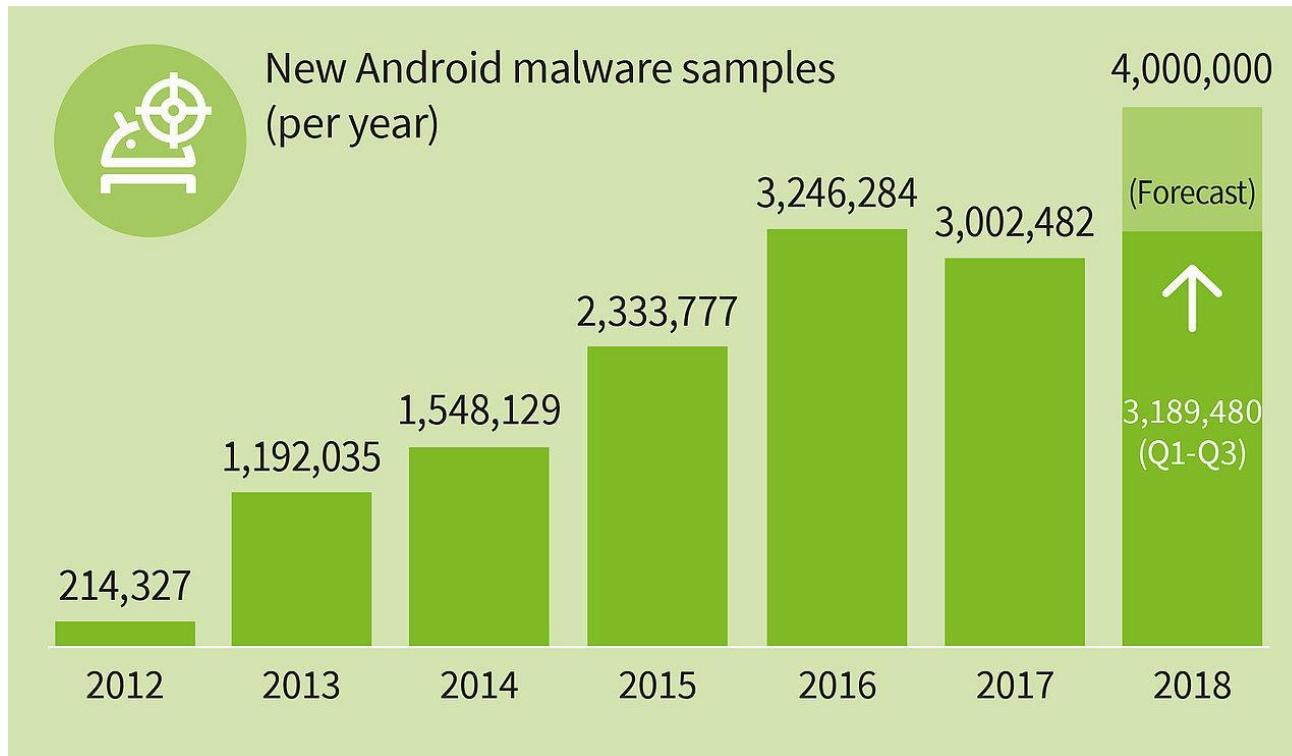
- So many cool mobile apps
- Access to web, personal information, social media, etc
- Security problems (not previously envisaged) have resulted
- Examples:
  - Malicious apps can steal your private information (credit card information, etc)
  - Jogging map generated from paths of Fitbit users can expose locations/behavioral habits of users. E.g. US soldiers at German base
  - Malware can lock your phone till you pay some money (ransomeware)
- Users/developers need better understanding of mobile security

# **Malware Evolution**

# Threat Types: Malware, Grayware & Personal Spyware

- **Malware:**
  - Gains access to a mobile device in order to steal data, damage device, or annoying the user, etc. **Malicious!!**
- **Personal Spyware:**
  - Collects user's personal information over of time
  - Sends information to app **installer** instead of author
  - E.g. spouse may install personal spyware to get info
- **Grayware:**
  - Collect data on user, but with no intention to harm user
  - E.g. for marketing, user profiling by a company

# Growth of Android Malware



# Mobile Malware Survey (*Felt et al*)

# Mobile Malware Study?

*A survey of mobile malware in the wild* Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner in Proc SPSM 2021

- First major mobile malware study in 2021 by Adrienne Porter Felt *et al*
  - Prior studies mostly focused on PC malware
- Analyzed 46 malwares that spread Jan. 2009 –June 2021
  - 18 –Android
  - 4 –iOS
  - 24 –Symbian (discontinued)
- Analyzed information:
  - in databases maintained by anti-virus companies
    - E.g., Symantec, F-Secure, Fortiguard, Lookout, and Panda Security
  - Discover malware based on mentions of malware in news sources
- Just analyzed malware. Did not analyze spyware and grayware

# Categorized Apps based on Behaviors

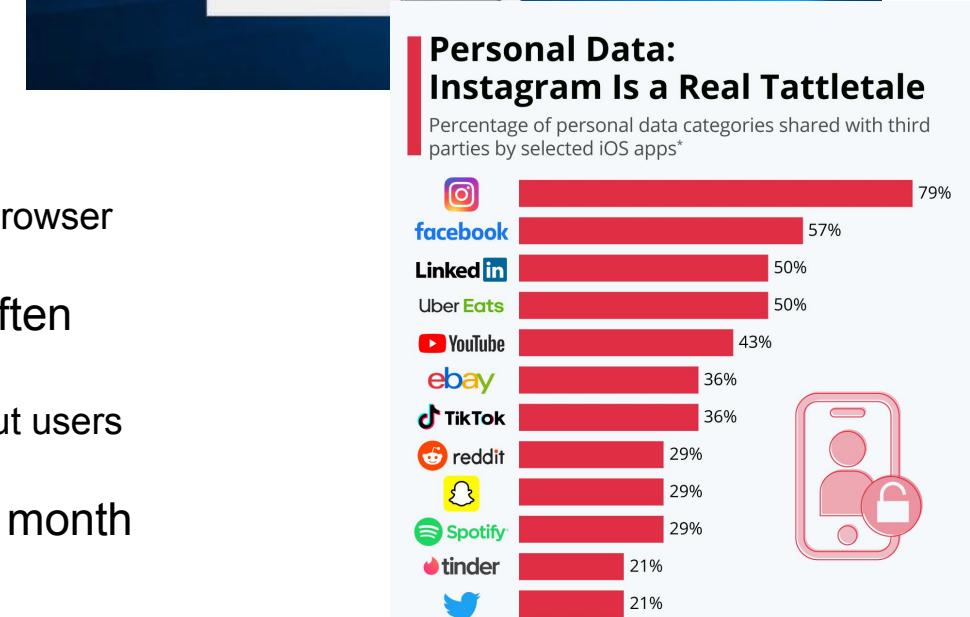
## 1. Novelty and amusement

- Designed primarily for minor mischief rather than serious harm
- Typically alters non-critical user settings
- Example: Changing the user's wallpaper or interface theme



## 2. Selling user information

- Collects personal data through system/API access
  - Possible data harvested: location, contacts, app download history, browser activity, user preferences
- Collected information is packaged and sold to third parties, often advertisers
  - Example: A coffee chain (e.g., Dunkin Donuts) purchasing data about users who frequent competitor locations
- Typical price range: Approximately \$1.90–\$9.50 per user per month



\* Based on privacy labels in the App Store, which group user data into 14 categories and inform users what data a given app collects and how it is used.

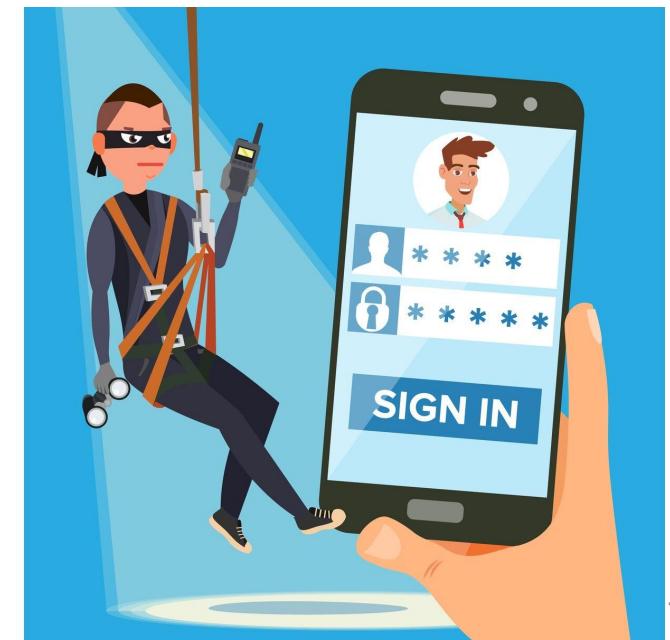
Source: pCloud



# Categorized Apps based on Behaviors

## 3. Stealing user credentials

- People use smartphones for activities that require them to input their passwords and payment information. E.g. shopping, banking, e-mail
- Malwares can log keys typed by user (keylogging), scan their documents for username + password
- User credentials can be sold
  - In 2018, black market price of:
    - Bank account credentials: \$10 to \$1, 000,
    - Credit card numbers: \$.10 to \$25,
    - E-mail account passwords: \$4 to \$30



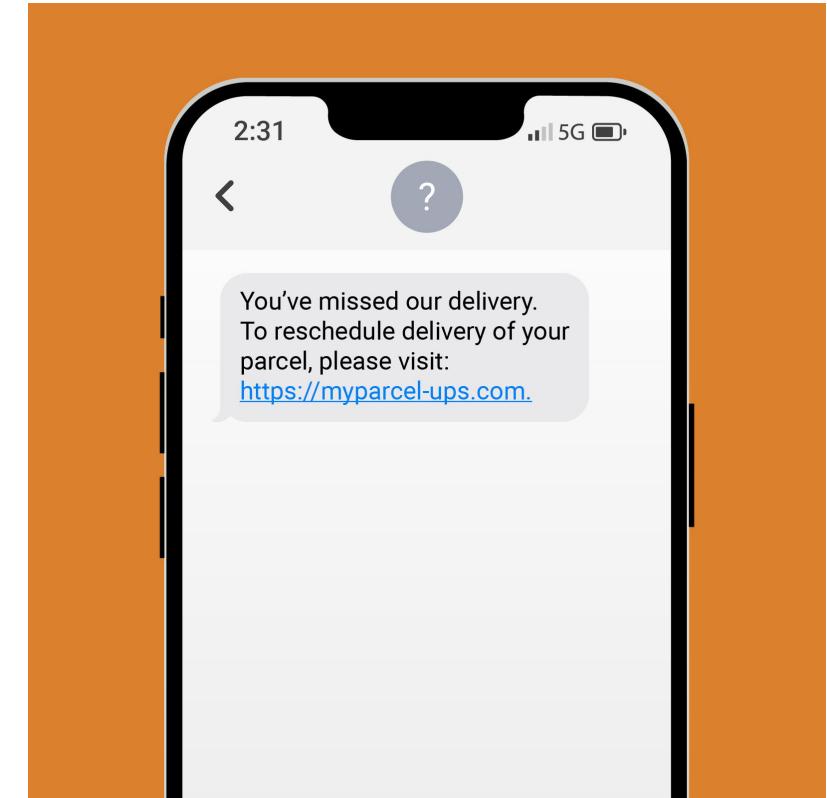
# Categorized Apps based on Behaviors

## 4. Make premium-rate calls and SMS

- Premium rate texts to specific numbers are expensive (E.g. 1-900.. Numbers)
- Attacker can set up premium rate number, Malware sends SMS there
- User is billed by their cell carrier (e.g. sprint), attacker makes money

## 5. SMS spam

- Used for commercial advertising and phishing
- Sending spam email is illegal in most countries
- Attacker uses malware app on user's phone to send SPAM email
- Harder to track down senders



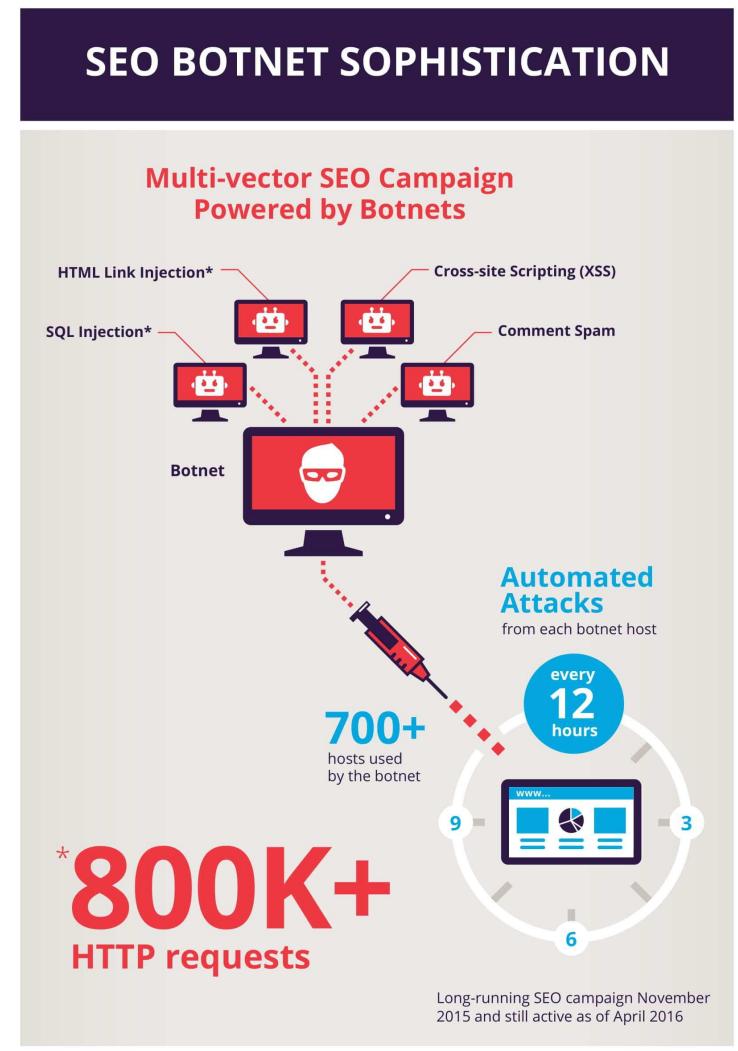
# Categorized Apps based on Behaviors

## 6. Search Engine Optimization (SEO):

- Malware makes HTTP requests for specific pages to increase their search ranking (e.g. on Google)
- Increases popularity of requested websites

## 7. Ransomeware

- Possess device, e.g. lock screen till money is paid
- Kenzero—Japanese virus inserted into pornographic games distributed on P2P networks
  - Publishes user's browser history on public website
  - Asked **5800 Yen**(~\$60) to delete information from website
  - About 12 % of users (661 out of 5510) actually paid



Long-running SEO campaign November 2015 and still active as of April 2016

Botnets in the past were running SEO campaigns, but usually just using one exploit like link injection or comment spam. The H1 report from the Imperva Defense Center investigates the botnet driven multi-vector SEO campaign that is still active and monitored since November 2015. The botnet deploys SQL injection, HTML link injection, cross-site scripting, and comment spam in an automated fashion to promote rankings of illegal/counterfeit websites.

Read the [full report](#) and learn how you can protect yourself from SEO botnet attacks.

# Recent High Profile Ransomware Attacks

- Colonial Pipeline (USA, 2021)
  - Ransom paid: \$4.4 million
  - Impact: Shutdown of the largest US fuel pipeline, causing fuel shortages across the East Coast.
- JBS Foods (Global, 2021)
  - Ransom paid: \$11 million
  - Impact: Forced shutdown of meat-processing plants in the US, Canada, and Australia.
- CNA Financial (USA, 2021)
  - Ransom paid: \$40 million (one of the largest known payments)
  - Impact: Shutdown of internal systems for weeks.



This device is locked due to the violation of the federal laws of the United States of America

# Malware Detection based on Permissions

- Does malware request more permissions?
- Analyzed permissions of 11 Android malware
- **Findings: Yes!**
- 8 of 11 malware request SMS permission (73%)
  - Only 4% of non-malicious apps ask for this
- Dangerous permissions: requests for personal info (e.g. contacts), etc
- Malware requests 6.18 dangerous permissions
  - 3.46 for Non-malicious apps

Number of Dangerous permissions	Number of non-malicious applications	Number of malicious applications
0	75 (8%)	-
1	154 (16%)	1
2	182 (19%)	1
3	152 (16%)	-
4	140 (15%)	2
5	82 (9%)	1
6	65 (7%)	-
7	28 (3%)	2
8	19 (2%)	1
9	21 (2%)	1
10	10 (1%)	1
11	6 (0.6%)	1
12	7 (0.7%)	-
13	4 (0.4%)	-
14	4 (0.4%)	-
15	2 (0.2%)	-
16	1 (0.1%)	-
17	1 (0.1%)	-
18	-	-
19	-	-
20	1 (0.1%)	-
21	-	-
22	-	-
23	1 (0.1%)	-
24	-	-
25	-	-
26	1 (0.1%)	-

Table 2: The number of “Dangerous” Android permissions requested by 11 pieces of malware and 956 non-malicious applications [28].

# **Android Security Model**

# Android Security

- Android security goals are to
  - Protect user data, system resources (hardware, software)
  - Isolate applications (e.g. app 1 from app 2)

## Foundations of Android Security

### 1. Application Isolation:

- Application sandboxing: App 1 cannot interact directly with app 2
- Apps can only communicate using secure inter-process communication

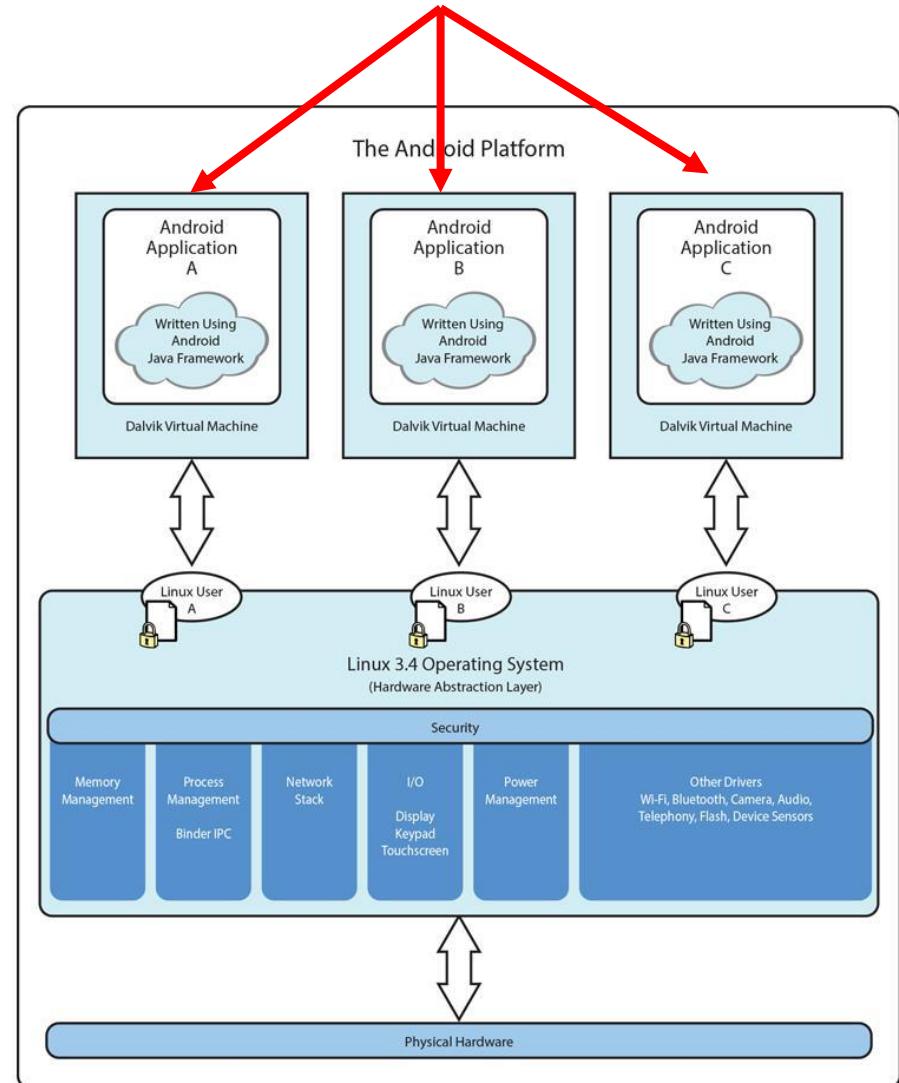
### 2. Permission Requirement:

- Supports default system, and user-defined permissions
- All apps must be signed: identifies author, ensures future updates are authentic

# Recall: Android Software Framework

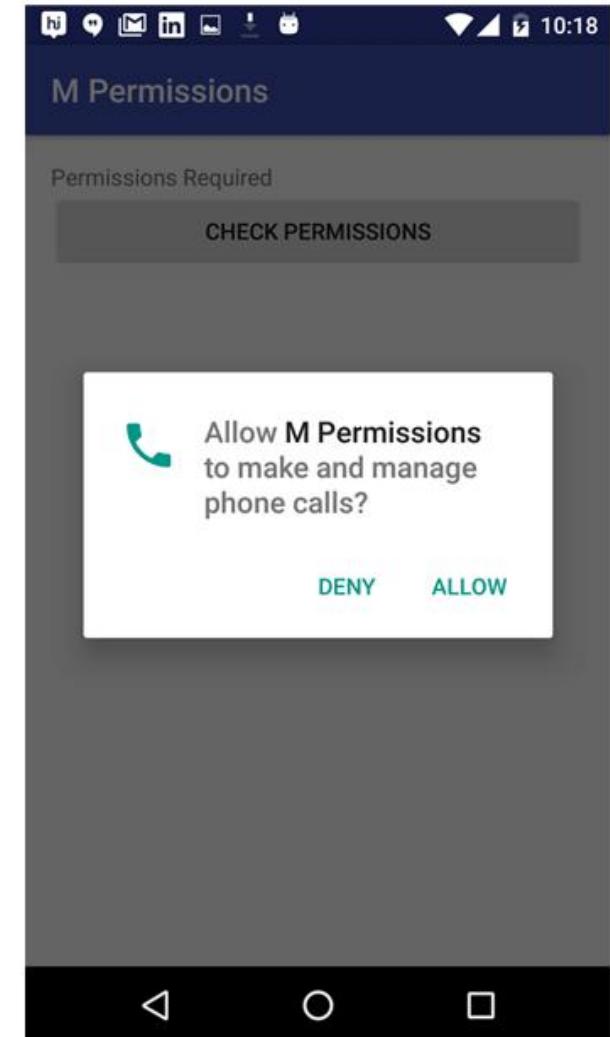
- Each Android app runs in its own security sandbox (VM, minimizes complete system crashes)
- Android OS multi-user Linux system
- Each app is a different user (assigned unique Linux ID)
- Access control: only process with the app's user ID can access its files
- Apps talk to each other only via intents, IPC or ContentProviders

Apps are isolated from each other



# Android Run-Time Permissions Changed in Marshmallow (Android 6.0)

- Pre Android 6.0: Permissions during install
- Android 6.0: Changes:
  - “Normal” permissions don’t require user consent
    - E.g. change timezone
    - Normal permissions can do very little to harm user
    - Automatically granted
  - Dangerous permissions (e.g. access to contacts can harm user)
    - Android 6.0: Run-time permissions now required for “dangerous” permissions



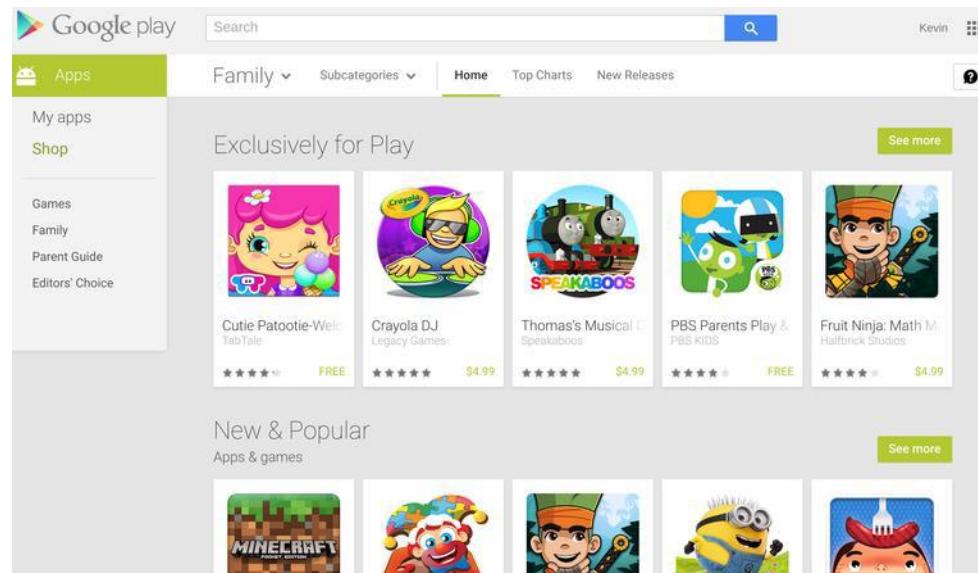
# Android Encryption

- Encryption encodes data/information, unauthorized party cannot read it
- **Full-disk encryption:** Android 5.0+ supports full filesystem encryption
  - Single key used to encrypt all the user's data
  - User password needed to access files, even to boot device
- **File-based encryption:** Android 7.0+ allows specific files to be encrypted and unlocked independently
  - Different keys used to encrypt different files

# **App Markets and Security Scanning**

# App Markets & Distribution

- Major OS vendors manage their own markets for “certified” apps
  - Android: Google Play Store
  - iOS: App Store (only way to download iPhone apps)



# App Market Scanning

- Google App Store: scanning called **Google Play Protect**
  - Antivirus scans apps on Google Play for threats, malware
  - New “peer grouping system”:
  - similar apps (e.g. all calculators) are grouped on app market.
  - If an app requests more permissions than similar apps, human takes a look
  - Also scans apps already installed on device, warns user if app looks malicious

# App Market Scanning

- Apple App Store
  - Highly regulated
  - All applications are reviewed by human
  - iOS devices can only obtain apps through official app store, unless jailbroken
- Many malware developers target third-party app stores (e.g. Amazon, getJar)
  - Weaker/no restrictions or analysis capabilities

# **Android Analysis Tools**

# Analyzing Android Apps

- Attacker can use analysis tools to get more information about an Android app
- **Source code recovery:** generate app source code from executable
- **Static analysis (binaries or source code):** Understand app design without running it.
  - Examine application logic, flow, APIs used
- **Dynamic analysis:** Observe how app executes
  - App memory usage, network usage, response time, performance, etc
- Many available (open source?) tools for all of the above!

# Android Analysis Tools

- APKinspector
- Androguard
- AndroBugs
- Qark
- Epicc / IC3
- FlowDroid
- DidFail
- DroidBox
- MobSF

# **Mobile Ad: Monetization or Grayware?**

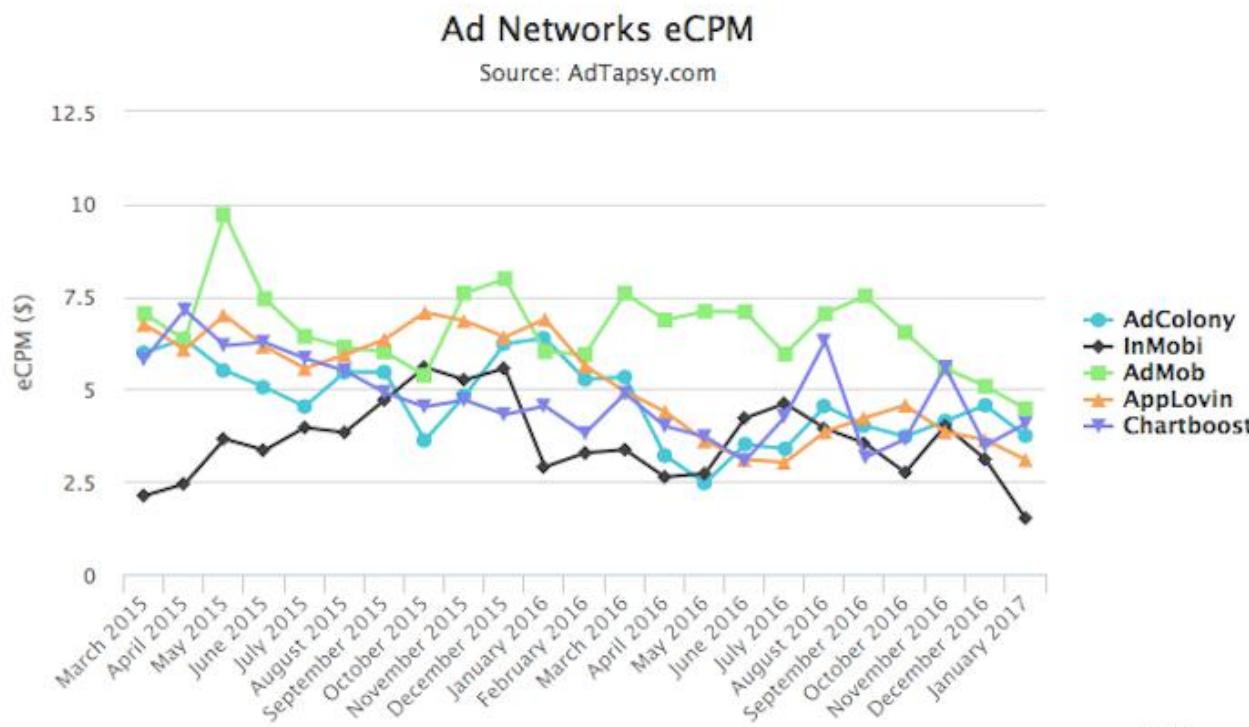


# Ad Services

- App developers make money from apps in 2 main ways:
  - Charge users fee for apps
  - Getting \$\$\$ from advertisers to include ads in apps
- To make money from ads, app author integrates ad services into app
- Mobile ad company serves ads to device

# AdMob

- AdMob: Most popular mobile ad company
  - Acquired by Google in 2009



# Permissions Requested by Ad Services

- Ad Services can also add requests to app's Android Manifest file
- Total permissions an app's AndroidManifest.xml  
= permissions requested by app + **permissions requested by ad service**

# Rogue? Ad Services

- Google is careful about permissions requested by AdMob
  - Some other mobile ad libraries require more permissions:
    - Access location data, camera, account details, calendar, call logs, browser bookmarks, contact lists, phone information, phone number, SMS, etc
    - Make phone calls, send SMS messages, vibrate
    - Change calendar and contacts

		Included in Apps	Probes Permissions	Uses Obfuscation	Uses Reflection	Uses JavaScript	Read Installed Packages	Location Data	Place Phone Call	Camera	List Accounts	Read Calendar	Read Contact/Call Logs	Read Browser Bookmarks	Read Phone Information	Read Phone Number	Send SMS	Change Calendar	Change Contacts	Use Vibrator	ClassLoader
admob/android/ads	27235	✓	✓	✓	✓	✓	.	✓	.	.	✓	.	✓	.	✓	.	✓	.	✓	.	
google/ads	16323	✓	.	✓	.	✓	.	✓	.	.	✓	.	✓	.	✓	.	✓	.	✓	.	
flurry	5152	✓	✓	.	✓	✓	.	✓	.	.	✓	.	✓	.	✓	.	✓	.	✓	.	
google/..../analytics	4551	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
millennialmedia	4228	✓	.	.	✓	✓	.	✓	.	.	✓	.	✓	.	✓	.	✓	.	✓	.	
mobclix	4190	✓	✓	.	✓	✓	.	✓	.	.	✓	✓	✓	.	✓	.	✓	.	✓	.	
adwhirl	3915	✓	.	✓	.	✓	.	✓	.	.	✓	.	✓	.	✓	.	✓	.	✓	.	
qwapi	1745	✓	.	.	.	.	.	✓	.	.	✓	.	✓	.	✓	.	✓	.	✓	.	
yoomi	1699	✓	✓	.	.	.	.	✓	.	.	✓	.	✓	.	✓	.	✓	.	✓	.	
mobfox	1524	✓	.	.	.	.	.	✓	.	.	✓	.	✓	.	✓	.	✓	.	✓	.	
zestadz	1514	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
cauly	1249	.	.	.	.	.	.	✓	✓	.	.	.	.	✓	.	.	.	.	.	.	
inmobi	1229	✓	.	.	.	.	.	✓	.	.	.	.	.	.	.	.	.	.	.	.	
wooboo	1183	✓	✓	.	.	.	.	✓	.	.	.	.	✓	.	✓	.	✓	.	.	.	
admarvel	1101	✓	.	.	.	✓	.	✓	.	.	.	.	✓	.	✓	.	.	.	.	.	
smaato	1077	✓	.	.	.	✓	.	✓	.	.	.	.	✓	.	✓	.	.	.	.	.	
mobclick	1058	✓	.	.	.	✓	.	✓	.	.	.	.	✓	.	✓	.	.	.	.	.	

# Final Words: Mobile Ad Services

- Many apps use multiple ad services
  - Angry Birds app (a game) includes 7+ ad services
- Example of rogue requests:
  - One version of the Dictionary.com app requests permissions to **monitor phone calls** and **access location**



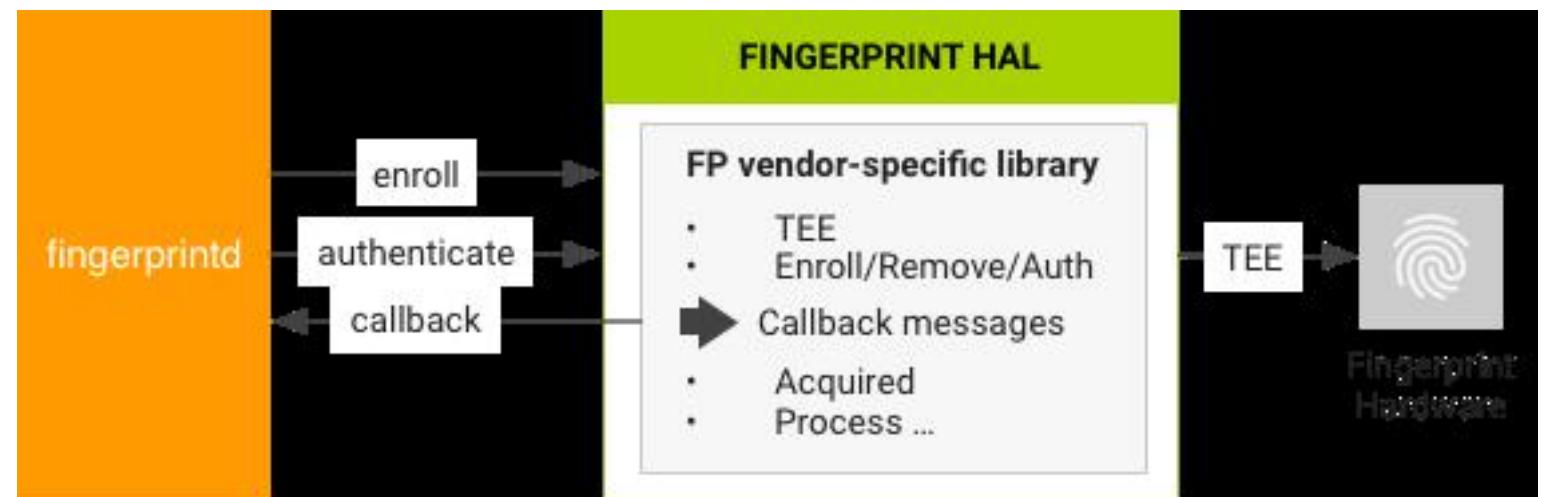
# **Authentication using Biometrics**

# Biometrics

- Passwords tough to remember, manage
- Many users have simple passwords (e.g. 1234) or do not change passwords
- Biometrics are unique physiological attributes of each person
  - Fingerprint, voice, face
- Can be used to replace passwords
  - No need to remember anything. Just be you. Cool!!

# Android Biometric Authentication: Fingerprints

- **Fingerprint:** On devices with fingerprint sensor, users can enroll multiple fingerprints for unlocking device



# Face ID

- Apple's Face ID is a facial-recognition technology that launched in 2017.
- The technology replaces Apple's Touch ID fingerprint scanning system
- Face ID uses a "TrueDepth camera system"
  - It consists of sensors, cameras, and a dot projector at the top of the iPhone display
  - It creates a detailed 3D map of your face



# **Continuous Passive Authentication using Behavioral Biometrics**

# User Behavior as a Biometric

- User behaviors patterns are unique personal features. E.g
  - Each person's daily location pattern (home, work, places) + times
  - Walk pattern
  - Phone tilt pattern
- **General idea:** Continuously authenticate user as long as they behave like themselves
- If we can measure user behavior reliably, this could enable **passive authentication**

# BehavioMetrics

- Derived from Behavioral Biometrics
  - Behavioral: the way a human subject behaves
  - Biometrics: technologies and methods that measure and analyzes biological characteristics of the human body
    - Fingerprints, eye retina, voice patterns
- BehavioMetrics:
  - Measurable behavior to recognize or verify a human's identity

# Mobile Sensing → BehavioMetrics

- Accelerometer
  - Activity & movement pattern, hand trembling, driving style
  - sleeping pattern
  - Activity level, steps per day, calories burned
- Motion sensors, WiFi, Bluetooth
  - Indoor position and trajectory.
- GPS
  - outdoor location, geo-trace, commuting pattern
- Microphone, camera
  - From background noise: activity, type of location.
  - From voice: stress level, emotion
  - Video/audio: additional contexts
- Keyboard, taps, swipes
  - User interactions, tasks .....

# BehavioMetrics → Security

- Track smartphone user behavior using sensors
- Continuously extract and classify features from sensors = Detect contexts, personal behavior features (pattern classification)
- Generate unique pattern for each user
- **Trust score:** How similar is today's behavior to user's typical behavior
- Trigger authentication schemes with different levels of authentication based on trust score

# Using Hand Gestures to Curb Mobile Malware *(Shrestha et al)*

# Malware Protection using Hand Movements

***Curbing Mobile Malware Based on User-Transparent Hand Movements*** Babins Shrestha, Manar Mohamed, Anders Borg, Nitesh Saxena and Sandeep Tamrakar in Proc IEEE Percom 2015

- **General idea:** Use real world hand movements to distinguish malware from real user
- Real user will make certain natural hand gestures when:
  - Making phone call
  - Taking a picture
  - Swiping to use NFC reader
- These hand gestures will be missing if activity is by malware
- **Main idea:** Check for these gestures (gesture recognition) to distinguish malware requests from valid user requests



# Sensors used for Gesture Identification

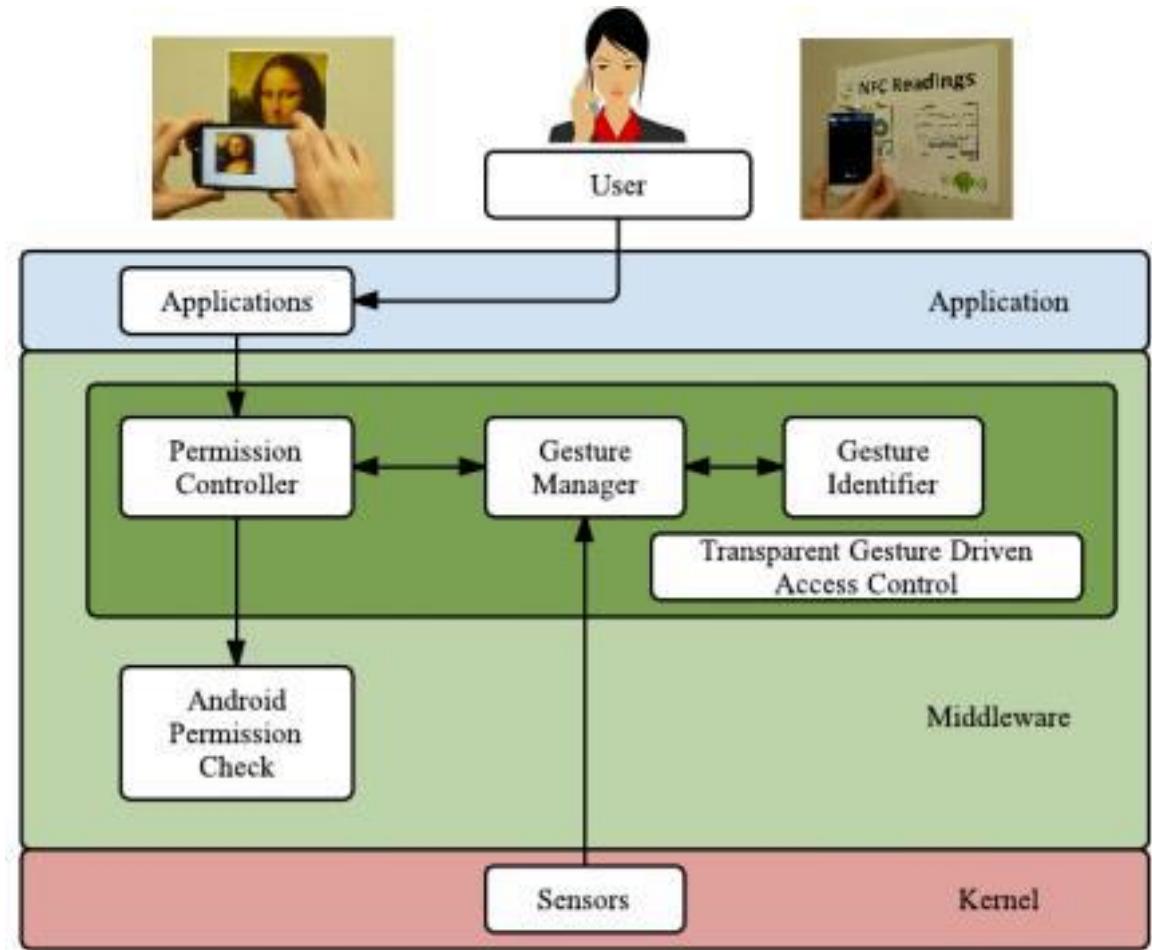
- Gesture Identifier used sensors to detect natural hand movements associated with phone dialing, taking picture, NFC usage
  - **Motion Sensors:** Accelerometer and gyroscope
  - **Position Sensors:** Magnetometer and orientation sensors
  - **Environmental Sensors:** Temperature, pressure and illuminance

TABLE I. SENSORS UTILIZED FOR GESTURE DETECTION

Type	Sensor	Description
Motion	Accelerometer (A)	The acceleration force including gravity
Motion	Gyroscope (Gy)	The rate of rotation
Motion	Linear Acceleration (LA)	The acceleration force excluding gravity
Motion	Rotation Vector (R)	The orientation of a device
Motion	Gravity (G)	The gravity force on the device
Position	Game Rotation (GR)	Uncalibrated rotation vector
Position	Magnetic Field (M)	The ambient magnetic field
Position	Orientation (O)	The device orientation
Environment	Pressure (P)	The ambient air pressure

# System Architecture

- 3 Entities
  - **Gesture Identifier:** classifier to identify gesture
  - **Permission Controller:** checks permissions granted by Android
  - **Gesture Manager:** compares gestures with permissions
- **Results:** > 85% accuracy (user gesture detection)



# Mobile Tagging

CSE 162 – Mobile Computing

Hua Huang

Department of Computer Science and Engineering

University of California, Merced

# Mobile Tagging

# Tagging the physical world

- When tags are attached to or linked to physical objects, they provide a way to audit physical spaces and processes.



# History of Barcode

- 1920's most complex business existed in the grocery store.
- Herman Hollerith built the “Hollerith Machine.”



# Groceries

- Consumers would get their cards punched depending on selected products.
- Pay for their selection at counter.
- Customer would receive groceries on a conveyor belt.
- Although the idea was ahead of time, it lead to the development of barcodes that are used in the modern supply chain.

# Barcode

- UPC (Universal Product Code)



- broadly accepted.
- Succeeded well in the marketplace resulting in the creation of EAN(European Article Numbering) and JAN (Japanese Article Numbering).

# QR Code for Mobile Tagging

- What is a QR (Quadratic) Code?
  - 2D barcode made of black/white modules in a square grid
  - “Quadratic” because data is laid out in two dimensions (x and y)
  - Easily scanned by smartphone cameras
- Why it works for Mobile Tagging
  - Encodes URLs, text, contacts, Wi-Fi configs, etc. in a tiny symbol
  - Users just point the phone camera → automatic decode → open app / webpage
  - Bridges physical objects (posters, products, tickets) to digital content
- Typical Mobile Tagging Use Cases
  - Advertising: posters → promo pages or videos
  - Payments and authentication (e.g., login via QR, pay-at-counter)
  - Ticketing and check-in: boarding passes, event tickets
  - Public info: museum exhibits, signage, transport schedules



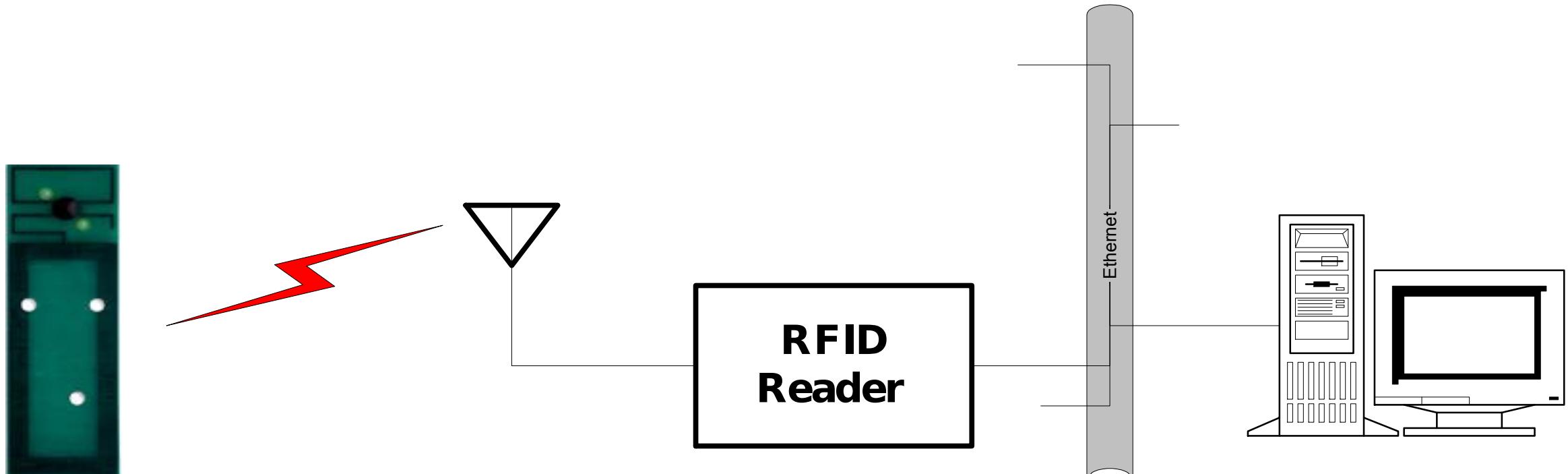
# Drawbacks of QR code and Barcode

- Limited in terms of information capacity.
- Easily damaged or lost.
- Need for human in the loop

# What is RFID?

- RFID = Radio Frequency IDentification.
- An ADC (Automated Data Collection) technology that:
  - uses radio-frequency waves to transfer data between a reader and a movable item to identify, categorize, track..
  - Is fast and does not require physical sight or contact between reader/scanner and the tagged item.
  - Performs the operation using low cost components.
  - Attempts to provide unique identification and backend integration that allows for wide range of applications.

# RFID system components



**RFID Tag**

**RF Antenna**

**Network**

**Workstation**

# RFID system components

- There are four parts to a RFID system:

## RFID Tag

- Programmed with information

## Antenna

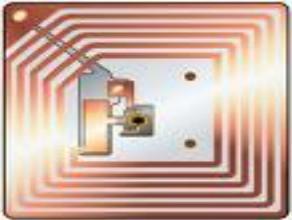
- Provides a means of communication and energy to communicate with RFID tag

## RFID Reader

- Has a decoder to interpret the data

## Work station

- Data management



# RFID vs. QR/Bar Codes

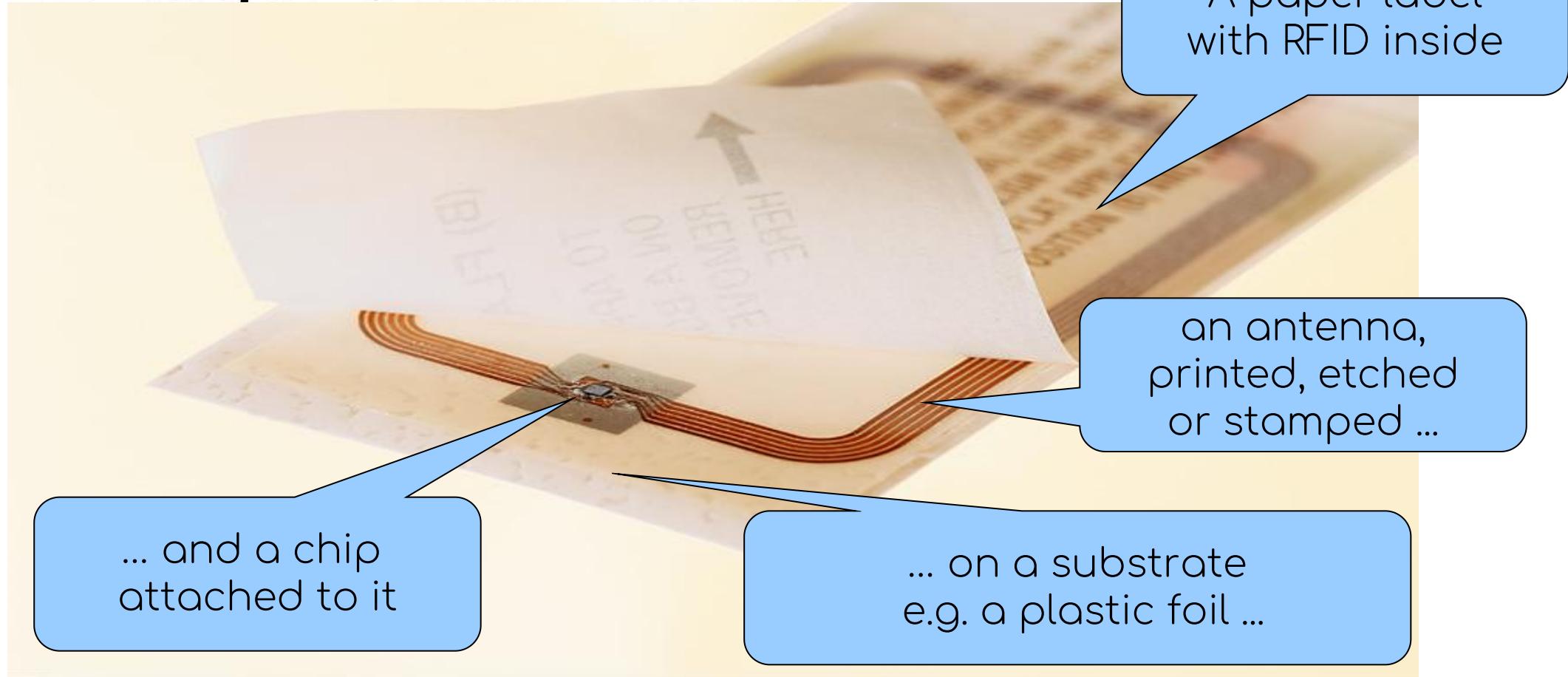


- How is RFID different from QR/Bar Codes?
  - Tag **does not need to be on the surface** of the object
    - No need for a direct line of sight, tags can be embedded or hidden.
  - Tags are **applicable in harsh environments**
    - e.g., outdoors, around chemicals, moisture and high temperatures.
  - **Faster:** RFID tags can be read at a rate of forty or more tags per second
    - Bar Codes: usually half a second or more per bar code.
  - **Longer Range:**
    - RFID tags can be read at distances up to 300 feet; Bar Codes no more than 15 feet.
  - **RFID Tags can be re-programmed;** Bar Codes do not have the read/write capability

# RFID vs. Bar Codes

	Manual Process	Bar Code	RFID
Data Accuracy	Least Accurate	Most Accurate	More Accurate
Data Collection Time/Labor	Most Time/Labor	Some Time/Labor	Least Time/Labor
Data Input Time/Labor	Most Time/Labor	Some Time/Labor	Least Time/Labor
Equipment Costs (tags, readers/scanners)	N/A	Some	More
Can Track Assets Out of Line of Sight	No	No	Yes
Amount of Data Storage on Tag	N/A	Less	More
Two way communication	No	No	Yes
Ability to Reprogram Tags	N/A	No	Yes

# RFID tags: Smart labels



# Tag Attachments

- Embedded
  - Usually aimed for permanent or long-term implantation, such as animal traceability
- Attached
  - Designed to be attached on the surface of identified objects with permanent, semi-permanent or temporary attachment
- Injected
  - Small enough to be implanted by injection through large-gauge needles
- Digested
  - The tags would be covered with soft gelatin that takes a while to dissolve in the stomach. Stop working when exposed to gastric acid for a specific period of time



# RFID tag memory

- Read-only tags
  - Tag ID is assigned at the factory during manufacturing
    - Can never be changed
    - No additional data can be assigned to the tag
- Write once, read many (WORM) tags
  - Data written once, e.g., during packing or manufacturing
    - Tag is locked once data is written
    - Similar to a compact disc or DVD
- Read/Write
  - Tag data can be changed over time
    - Part or all of the data section can be locked

# RFID readers

- Reader functions:
  - Remotely power tags
  - Establish a bidirectional data link
  - Inventory tags, filter results
  - Communicate with networked server(s)
  - Can read 100-300 tags per second
- Readers (interrogators) can be at a fixed point such as
  - Entrance/exit
  - Point of sale
- Readers can also be mobile/hand-held



# Types of Readers



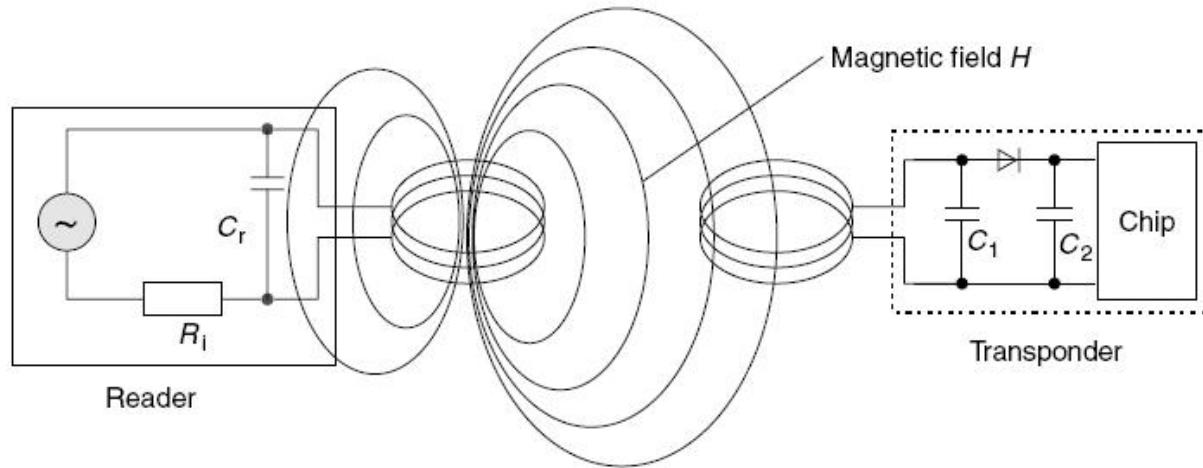
- Dumb
  - Can read only one type of tag using one frequency and one protocol
- Agile
  - can read tags at different frequencies or using different methods of communication
- Intelligent
  - Agile reader+ applications such as data filtering

# Some RFID readers



# Working of Passive RFID

# Inductive Coupling

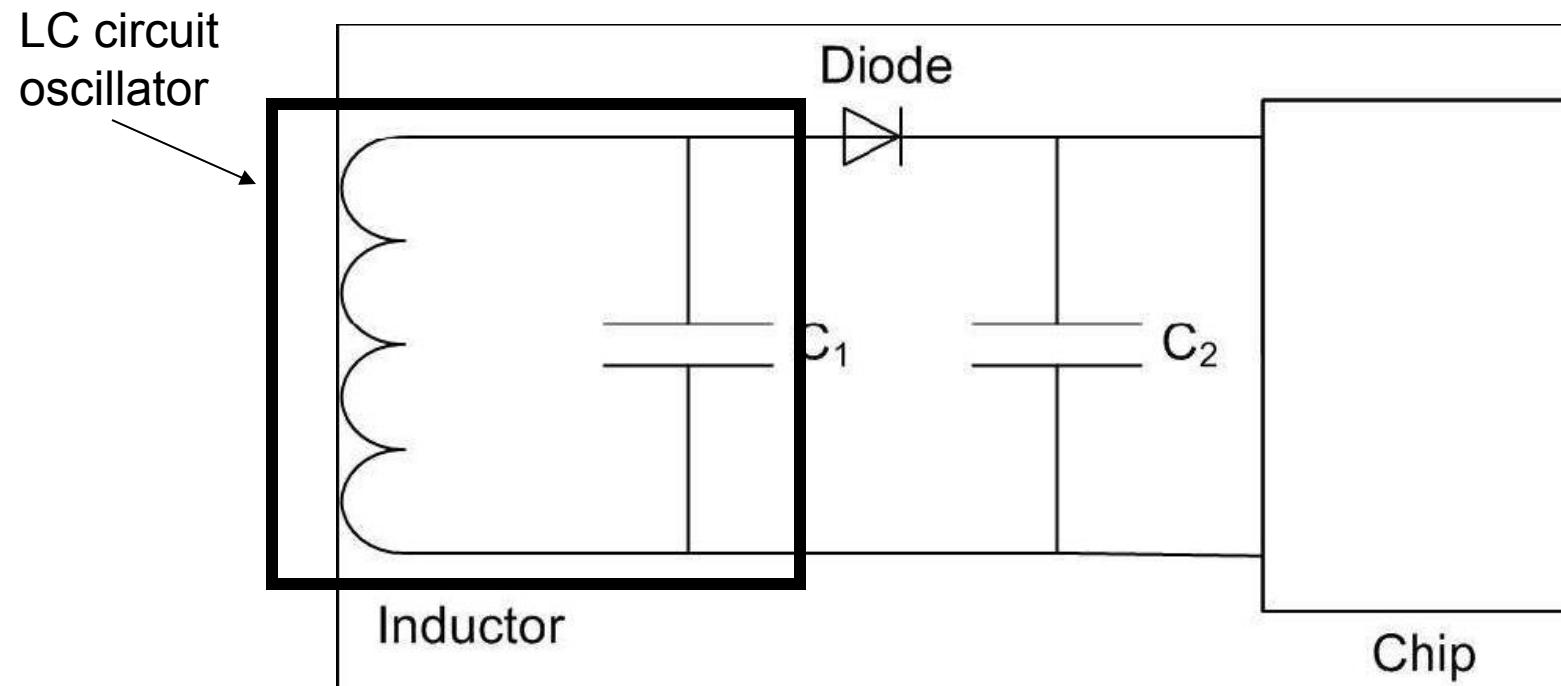


Inductive coupling means that the transponder and the antenna are coupled by the magnetic flux through both coils, much like a transformer.

All the energy used in the tag is drawn from the primary coil of the antenna.

# The rest of the picture

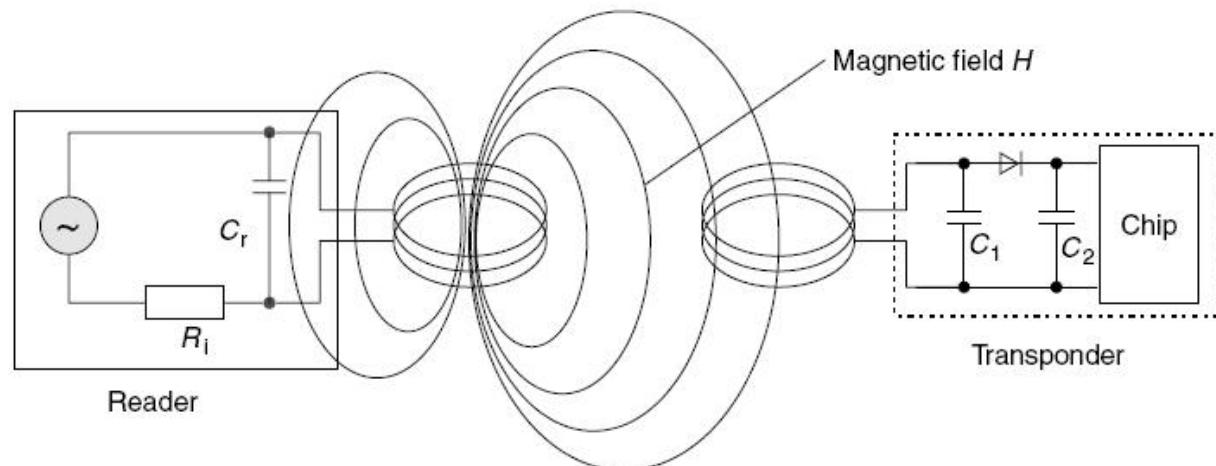
- The energy in the coil can be harvested using diodes and capacitors ( $C_2$ ) to rectify the current.



# Introduction

- For low frequency systems (100kHz-30MHz) the tag is typically read in the near field of the reader

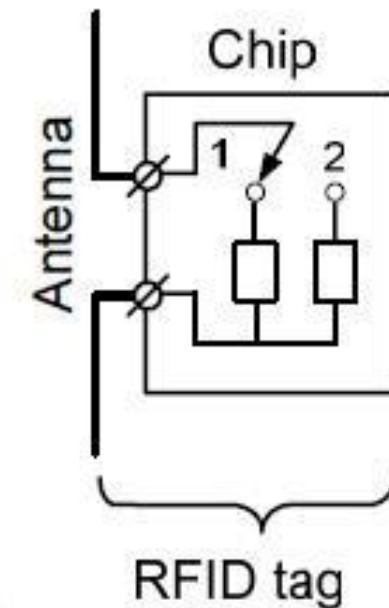
$$Z_1 = Z_{11} - \frac{Z_{12}^2}{Z_{22} + Z_{TAG}}$$



# Digital Modulation

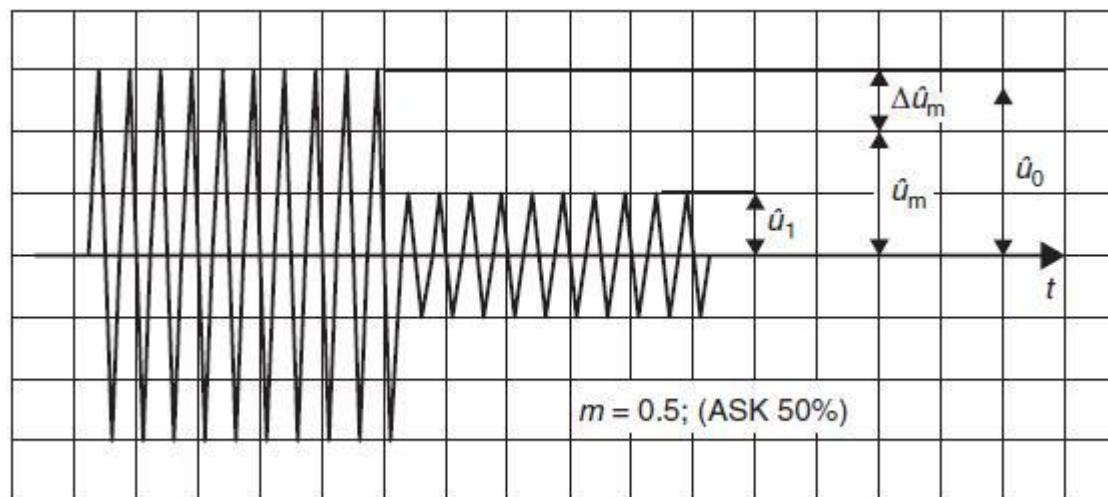
- Circuit in the tag
- The inductance load,  $Z_{tag}$ , is electronically adjusted by switching 1 or 2.

$$Z_1 = Z_{11} - \frac{Z_{12}^2}{Z_{22} + Z_{TAG}}$$



# Digital Modulation

- As the load  $Z_{tag}$  is adjusted, in the reader side, the current amplitude will change accordingly.

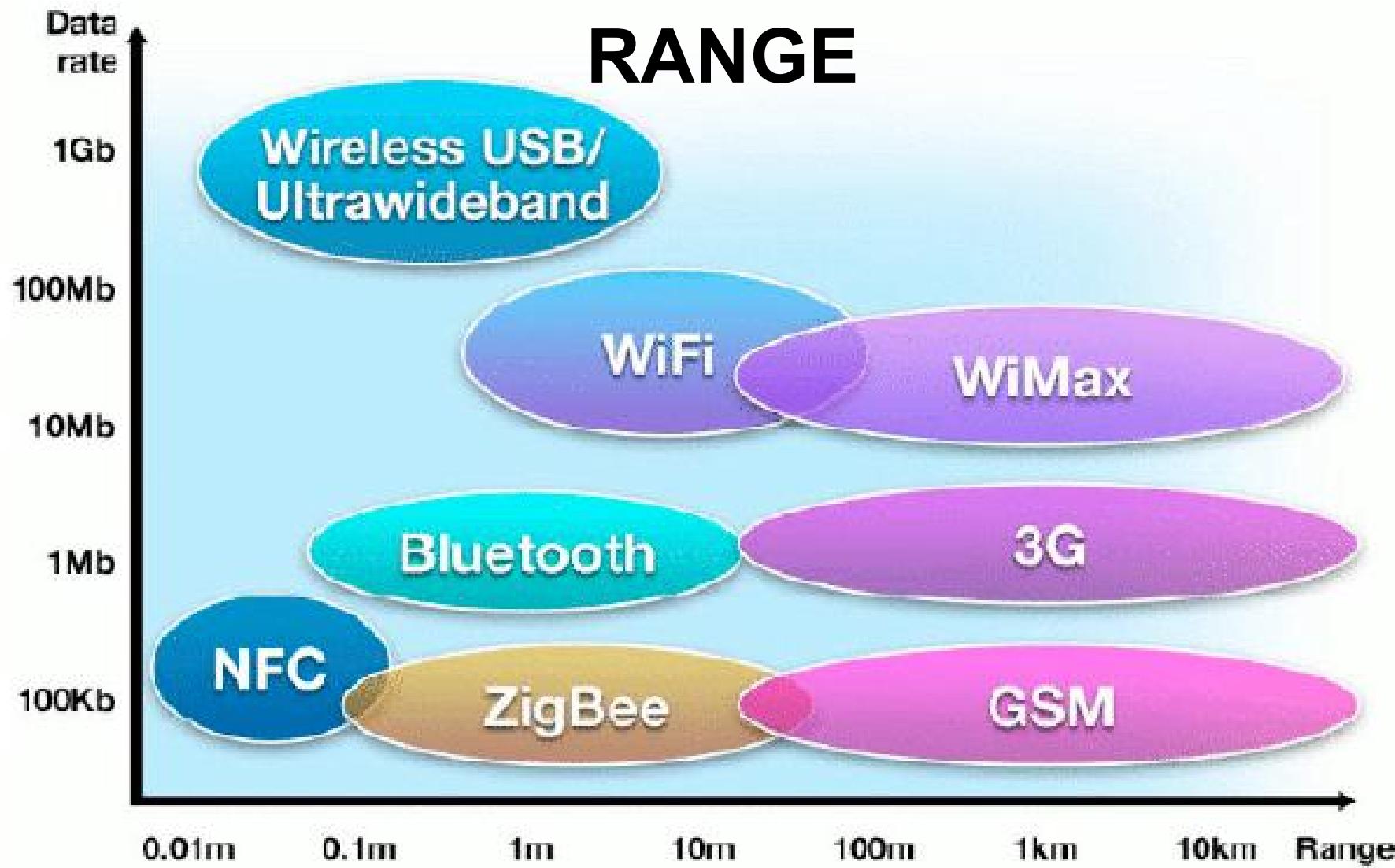


# Near Field Communication (NFC)

# What is NFC?

- NFC or Near Field Communication is a type of High frequency RFID
- NFC is mainly aimed for mobile or handheld devices.
- It allows for simplified transactions, data exchange, and wireless connections between two devices.
- Allows communication between
  - Two powered (active) devices
  - Powered and non self-powered (passive) devices

# DATA RATE & RANGE

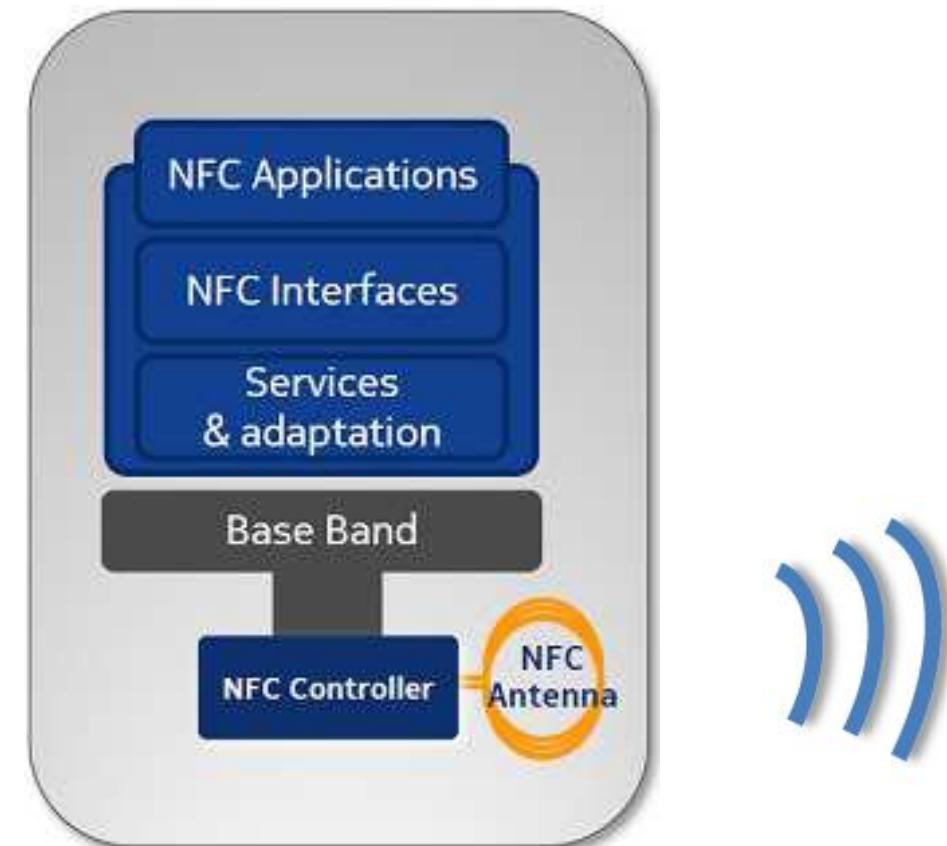


# Features

- NFC is an extension of **Radio frequency identification (RFID)** technology
- It operates within the globally available and unlicensed radio frequency band of **13.56 MHz**, with a bandwidth of **14 kHz**.
- Working distance with compact standard antennas: up to **10 cm (mostly)**
- Supported data rates: **106, 212 and 424 Kbit/s**

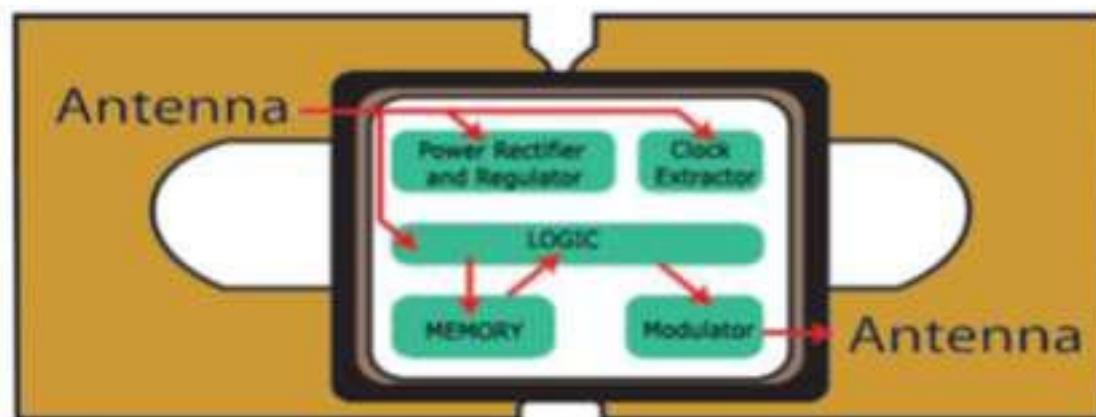
# NFC Reader

- Usually a microcontroller-based (e.g., smartphones) with an integrated NFC circuits
- The reader continuously emits RF carrier signals, and keeps observing the received RF signals for data



# NFC Tag

- The passive tag absorbs a small portion of the energy emitted by the reader (phone), and starts sending modulated information when sufficient energy is acquired from the RF field generated by the reader.



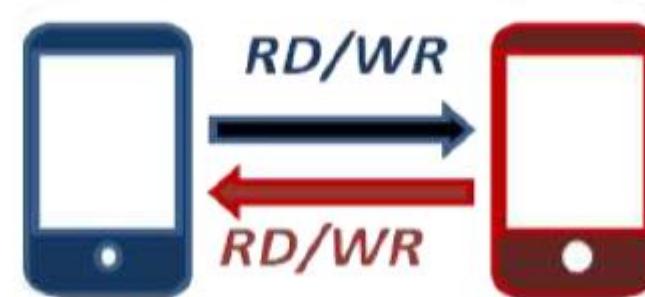
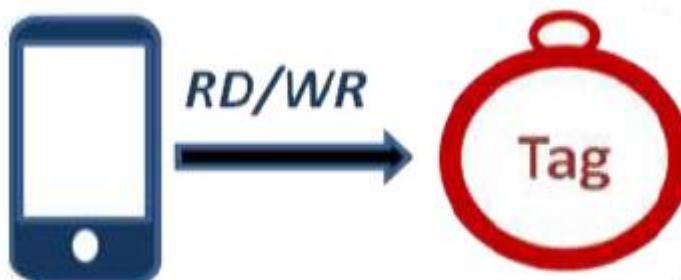
# Operation Of NFC

- NFC has two communicative terminals
  - The **INITIATOR** is the one who wishes to communicate and starts the communication.
  - The **TARGET** receives the initiator's communication request and sends back a reply



# Communication modes Of NFC

- **Passive Communication Mode**
- The Initiator device provides a carrier field and the target device answers by modulating existing field.
- **Active Communication Mode**
- Both Initiator and Target device communicate by alternately generating their own field.



# Operating Modes of NFC devices

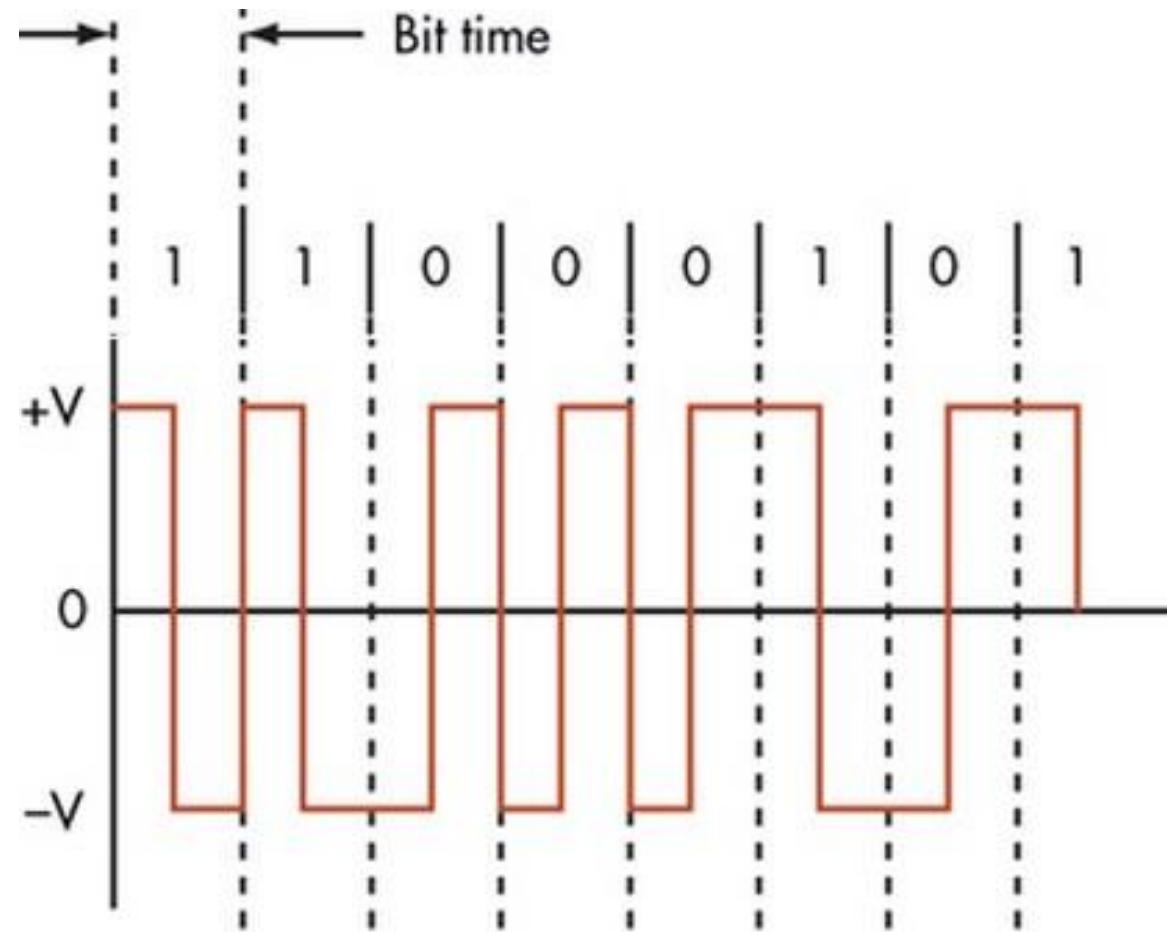
- **Reader/writer mode**, allowing the NFC device to read and/or write passive NFC tags and stickers.
- **P2P mode**, allowing the NFC device to exchange data with other NFC peers;
  - this operation mode is used by Android Beam.
- **Card emulation mode**, allowing the NFC device itself to act as an NFC card.
  - The emulated NFC card can then be accessed by an external NFC reader, such as an NFC point-of-sale terminal.



# Signal Encoding in NFC

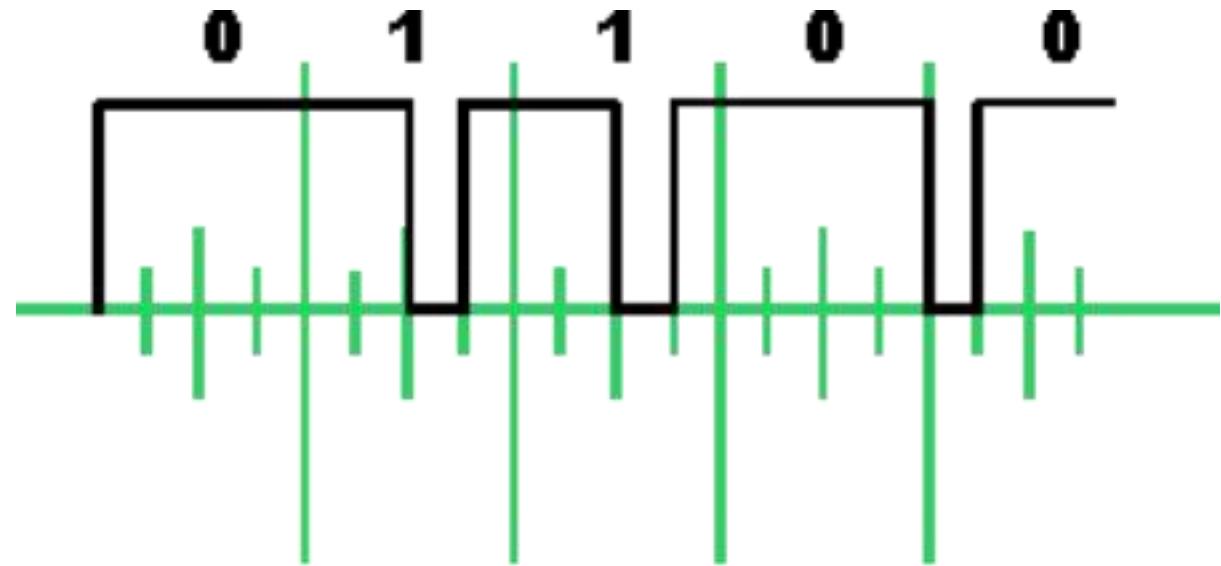
# NFC Manchester Coding

- High to Low transition is mapped as binary logic-1
- Low to High transition is mapped as binary logic-0
- Transition occurs exactly in the middle of bit period.



# Modified Miller Coding

- A high or "1" is always encoded in the same way, but a low or "0" is encoded differently dependent upon what preceded it.
- Pros
  - Has fewer transitions. This characteristic saves bandwidth



**Key:**

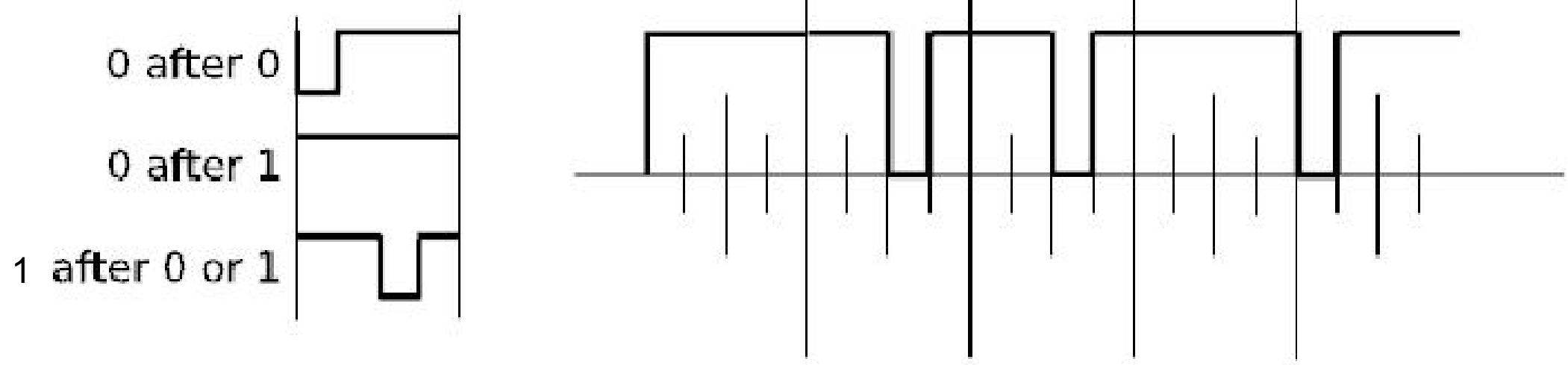
0 after 0

0 after 1

1 after 0 or 1

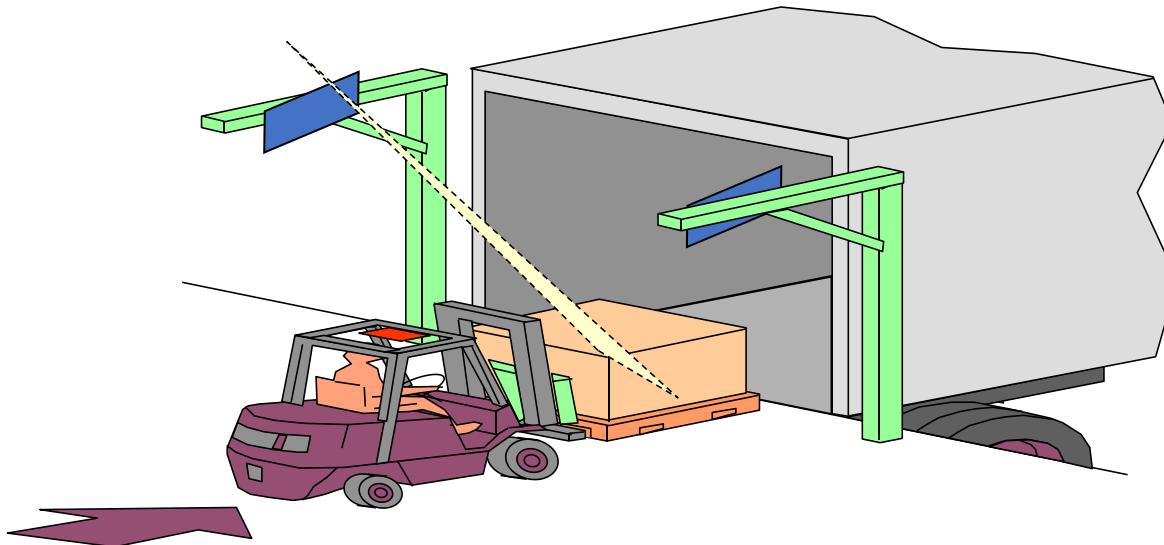
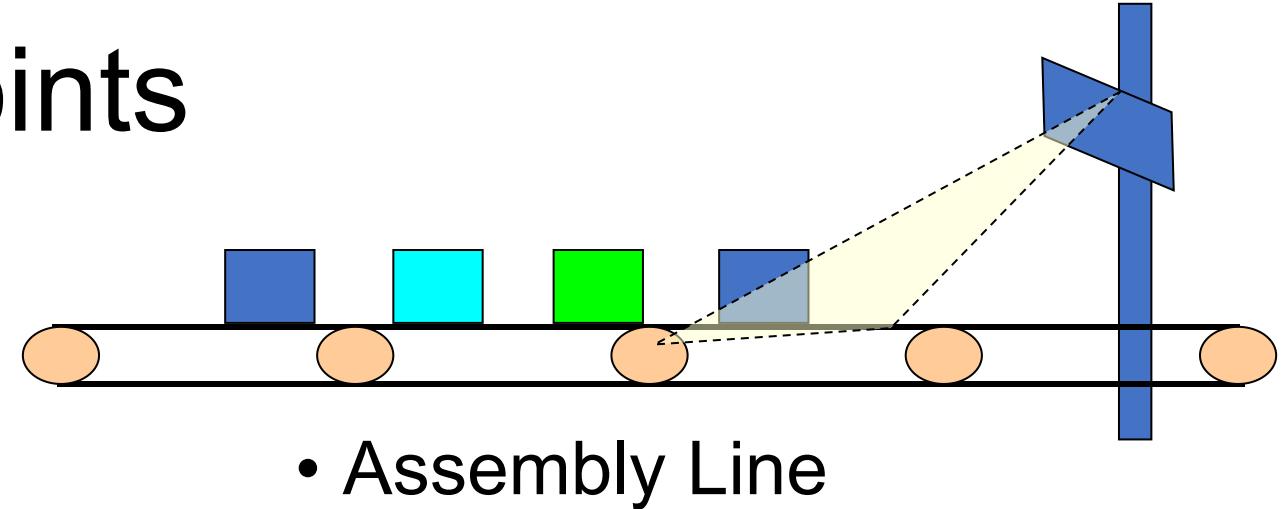
# Exercise

- Decode the signal modulated in modified miller code



# Applications

# RFID application points



§ Shipping Portals

§ Handheld Applications

# Smart groceries

- Add an RFID tag to all items in the grocery.
- As the cart leaves the store, it passes through an RFID transceiver.
- Check out in seconds.

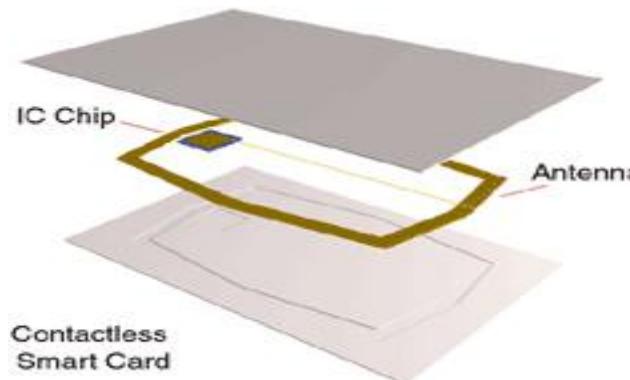


# Smart fridge

- Recognizes what's been put in it
- Recognizes when things are removed
- Creates automatic shopping lists
- Notifies you when things are past their expiration
- Shows you the recipes that most closely match what is available

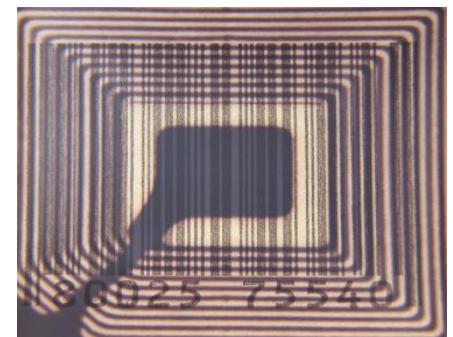
# Access control and identification

- Cards for access to secure areas.
- Wristbands to provide access to unattended buildings.



# Tracking people and objects

- Children in theme parks
- Protection of expensive objects
- Personnel activities inside a facility
- Inmates identification and tracking



# Ticketing

- Trains, subways, buses, concerts, amusement parks, fitness facilities, ski resorts.
- Reduces counterfeit
- RFID tickets used in the Soccer World Cup in Germany in 2006. 4.8 millions Tickets sold with no counterfeiting problems.



# Libraries

- From barcodes to RFID tags.
- Faster and automatic checkout and return.
- Faster inventory process.

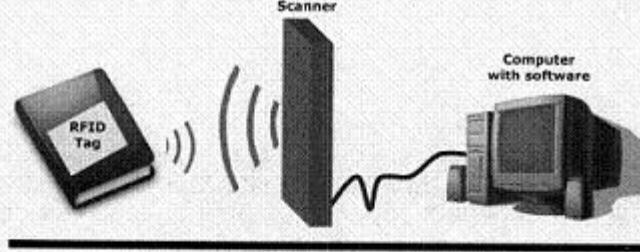


Figure 1. Components of an RFID system

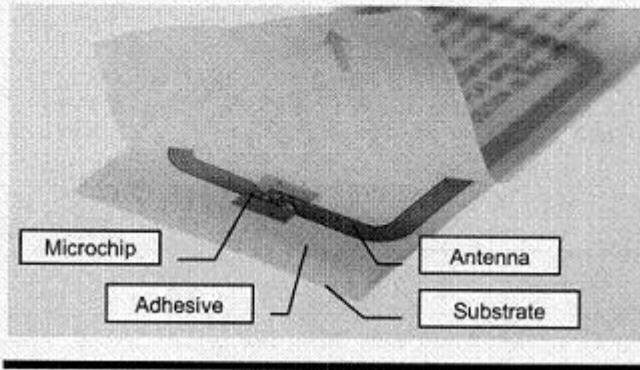


Figure 2. Typical RFID tag

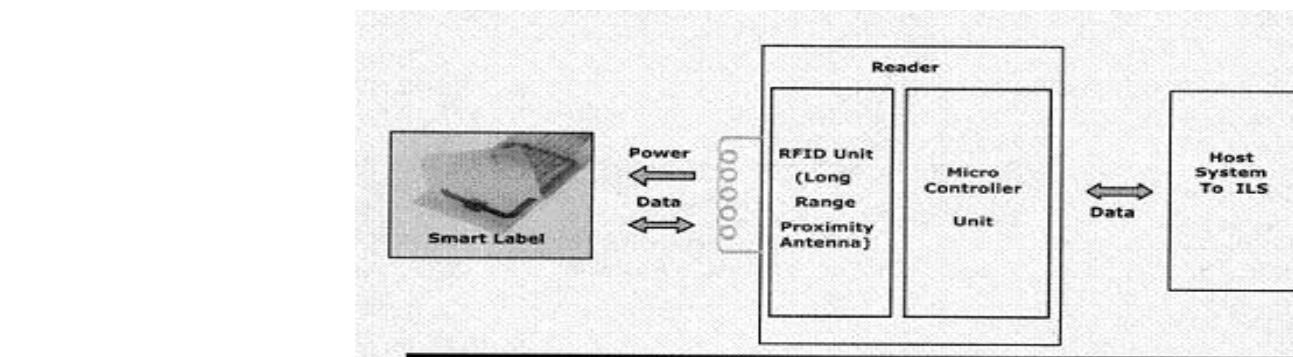


Figure 3. Information exchange for a typical RFID-based library (source: Libramation Library Systems)

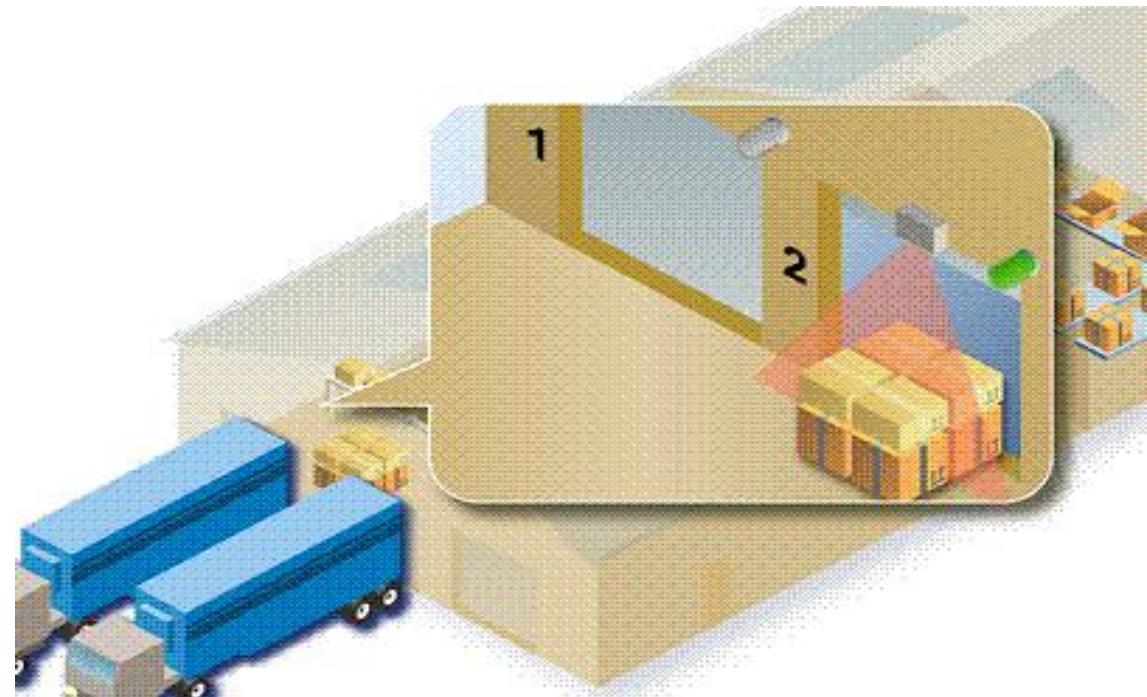
# Production Line Control and Monitoring

- Car Body Production: Flow of information along the assembly line for process verification.
- Identify vehicles through assembly line prior to the execution of a given assembly task.



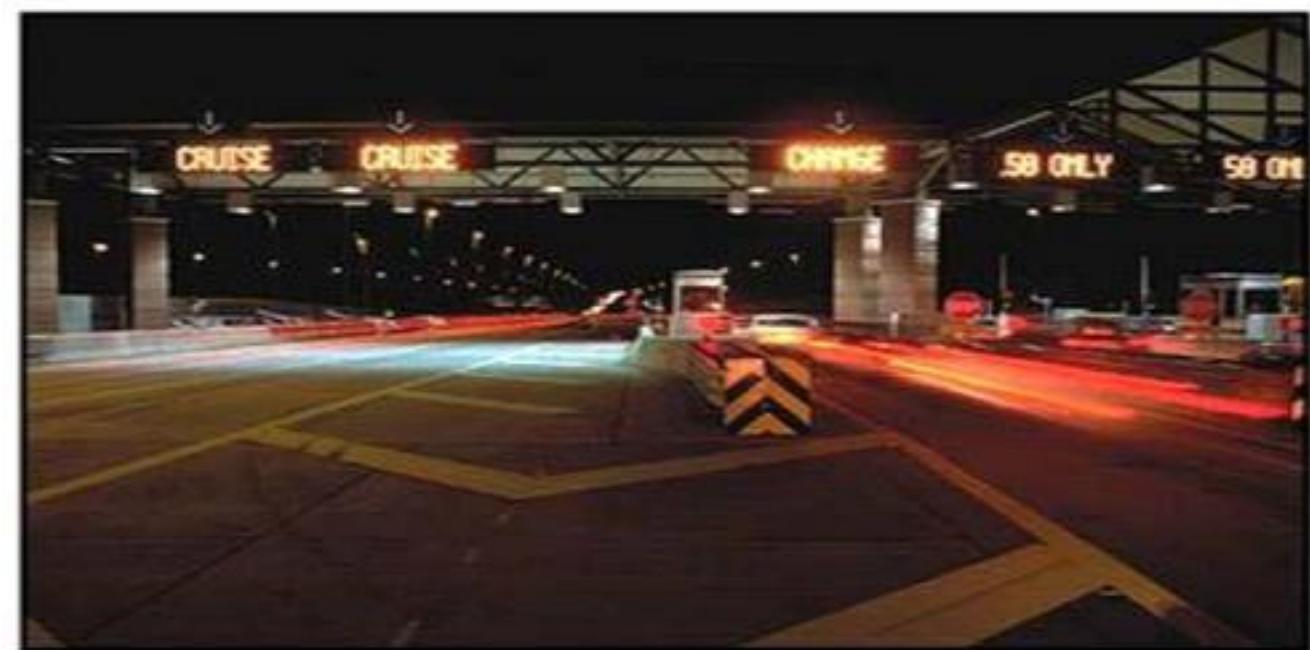
# Distribution and transportation

- Receiving and check-in
- Put away and replenishment
- Order Filling
- Shipping
- Product and asset tracking



# Vehicle Identification

- Fleet management
- Access to parking lots
- Railway industry
- Electronic toll collection



# Baggage handling

- Replace bar code stick labels with RFID inlay stick labels.
- Many tags can be read at one time.
- Read-write capability is used to record information along the way as the bag makes its way through the handling system.



# Animal Identification

- Livestock tracking
- Data critical for the safety of food supply
- Can also be used on pets.
- Ear tags, injectable tags, RFID tattoos



# Mobile Tagging (2)

CSE 162 – Mobile Computing

Hua Huang

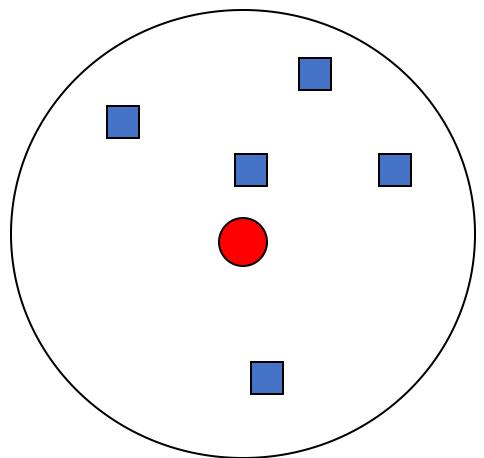
Department of Computer Science and Engineering

University of California, Merced

# Collision Avoidance in RFID

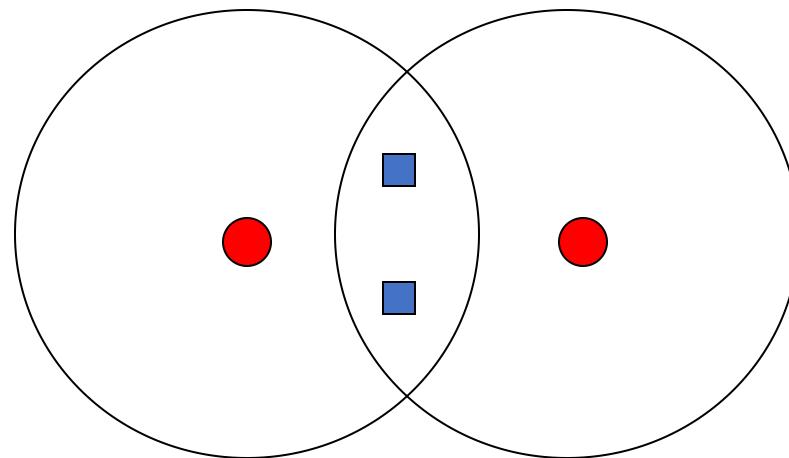
- Tag collision
- Reader collision

Tag collision



Probability-based  
Deterministic-based (Prefix-based)

Reader collision



Centralized  
Distributed

# Tag-to-Tag Collision Avoidance

# Tag Collision Problem

When multiple tags are in range of the reader:

- All the tags will be excited at the same time.
- Makes it very difficult to distinguish between the tags.

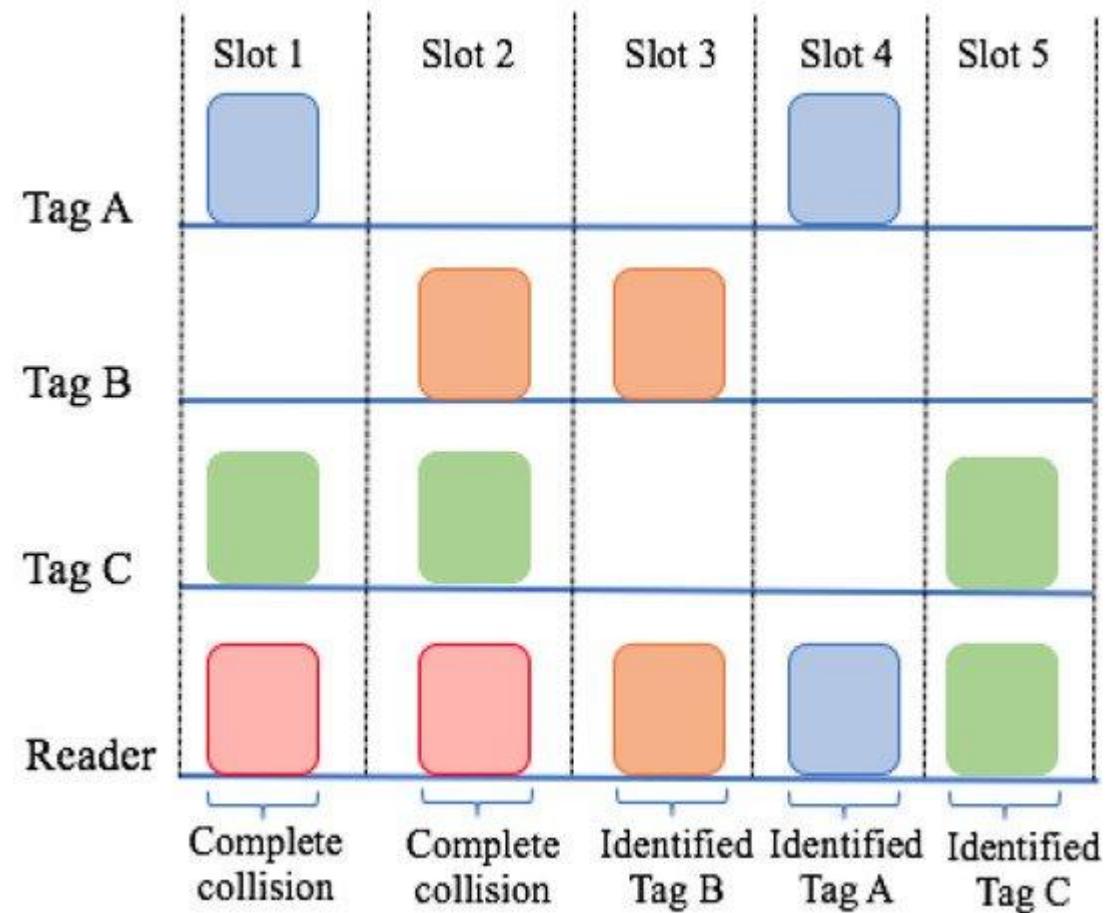
# Tag Collision Problem

- Collision avoidance mechanisms:
  - Probabilistic: Tags return at random times.
  - Deterministic: Reader searches for specific tags.

# Aloha Algorithm

# Tag Collision avoidance: ALOHA

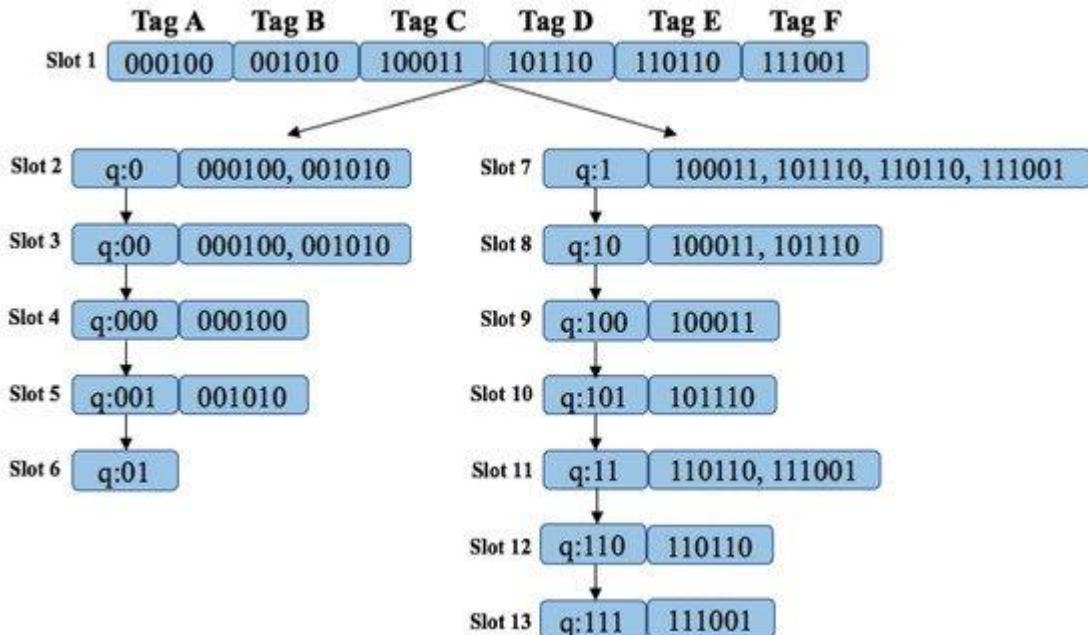
- Tags detect when a collision has occurred, and attempt to resend after waiting a random interval.
- Pros: simple
- Cons: when tags are dense, the network can reach congestion collapse.



# Query Tree

# Query Tree

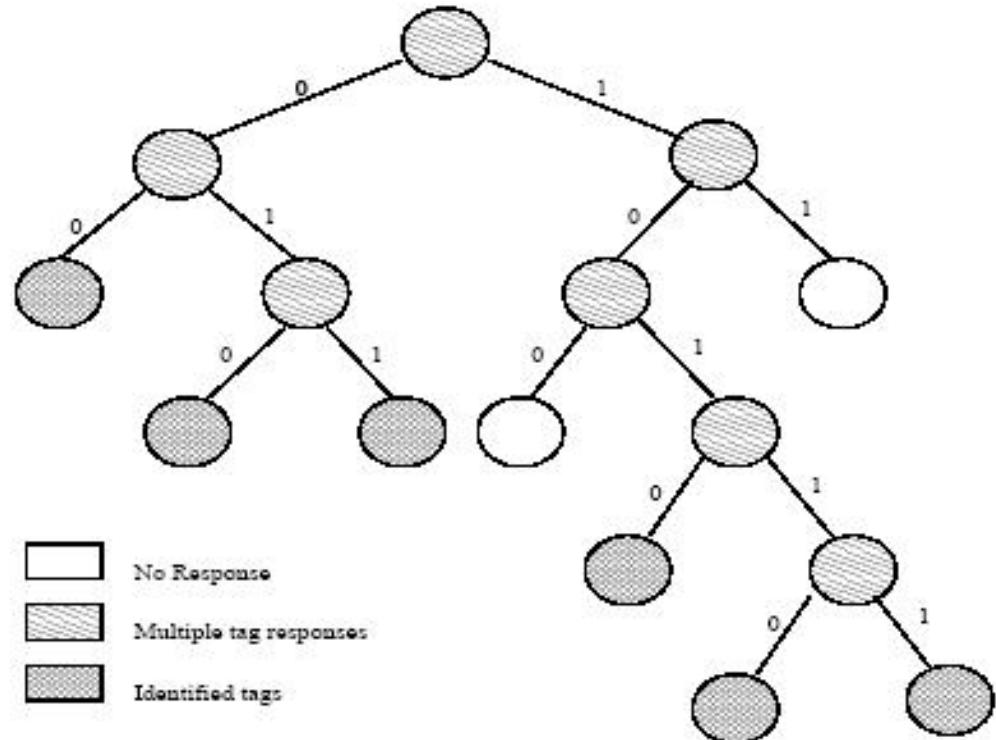
- Assumption: tagID stored in  $k$  bit binary string
- Algorithm
  - Reader queries for prefix of length  $p$
  - In case of collision queries prefix of length  $p+1$



# Query Tree– Example

- Reader queries for prefix of length  $p$
- In case of collision, reader queries for prefix of length  $p+1$
- Example: consider tags with prefixes: 00111, 01010, 01100, 10101, 10110 and 10111

Step	Query Prefix	Response
1	0	Collision
2	1	Collision
3	00	00111 (Identified)
4	01	Collision
5	10	Collision
6	11	No Response
7	010	01010 (Identified)
8	011	01100 (Identified)
9	100	No Response
10	101	Collision
11	1010	10101 (Identified)
12	1011	Collision
13	10110	10110 (Identified)
14	10111	10111 (Identified)



# Query Tree - Pros and Cons

- Pros
  - Simple
  - Bounded Query time
- Cons
  - Require prior knowledge about the tags
    - It can be unavailable

# Binary Search Protocol

# Binary Search Protocol

- Reader transmitting a serial number from the reader to all the tags in the interrogation area.
- Only tags which have an equal or lower ID value than the received serial number will respond to the request
- The reader checks the tags' responses bit by bit using Manchester coding and if a collision is detected, the reader divides the tags into subsets based on the collided bits.
  - Requirement: the reader can detect which bit has collisions
- Benefit: no prior knowledge about the Tag IDs is required

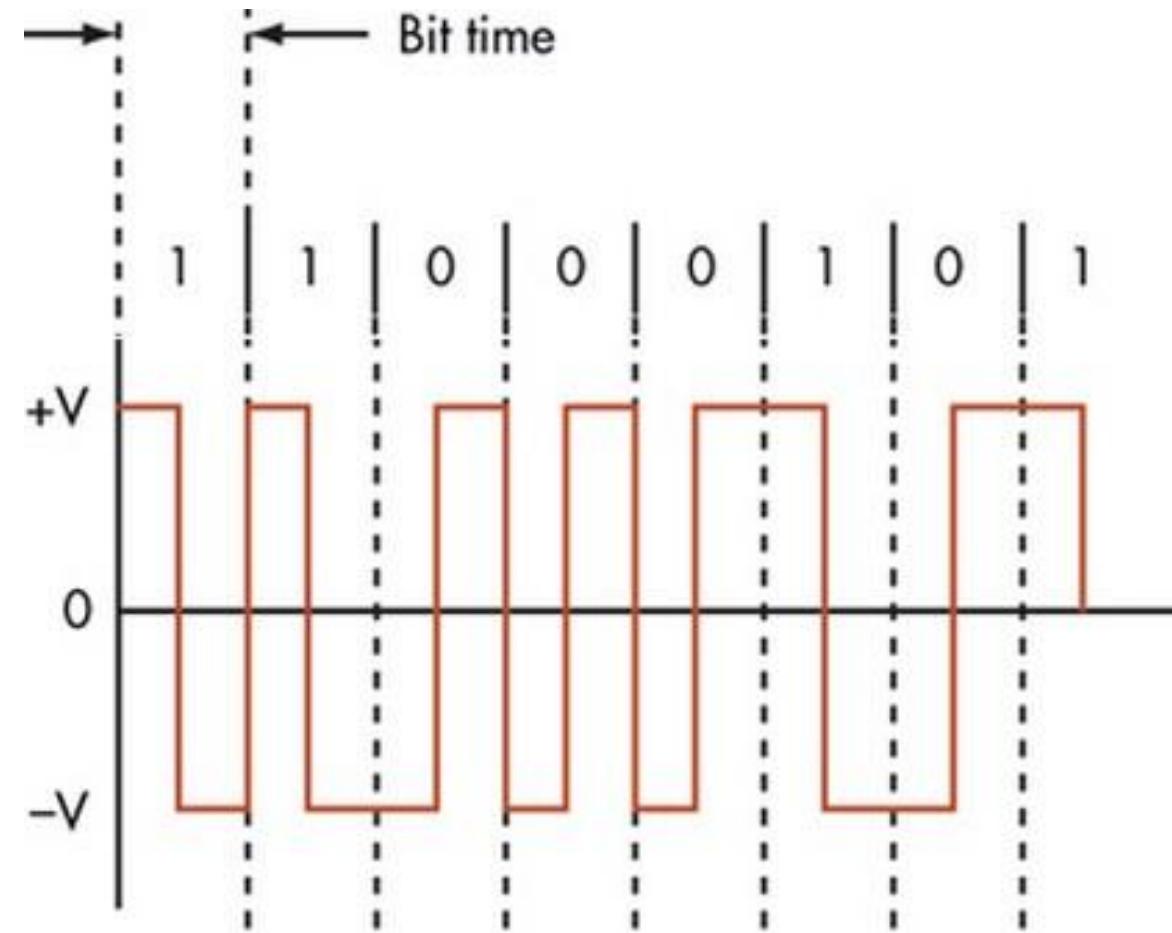
# Binary Search Protocol

Slot Number	Reader Command	Tag A (010)	Tag B (011)	Tag C (100)	Tag D (110)	Result	Type of Slot
Slot 1	111	010	011	100	110	XXX	Collision
Slot 2	011	010	011			01X	Collision
Slot 3	010	010				010	Success
Slot 4	111		011	100	110	XXX	Collision
Slot 5	011		011			011	Success
Slot 6	111			100	110	1X0	Collision
Slot 7	101			100		100	Success
Slot 8	111				110	110	Success

- The reader begins by interrogating tags with the maximum ID value 111. Tags with a value of less than 111 will respond to the query. Their answer results in collision XXX
- In the next slot, the reader transmits a new query by replacing the most significant collided bit (MSB) with a 0. The reader transmits a new query, 011, and all tags compare their ID with the received value. Communication in this slot again results in a collision (01X).
- In the third slot, the reader replaces the third bit of the command with a 0 and transmits the next query, 010. In the new interrogation round (slot 3) only Tag A has a value equal to or lower than 010, and therefore it is successfully identified.
- After this slot, the reader restarts the query value with the initial value 111 and transmits it. This procedure is repeated until all of the tags are identified.

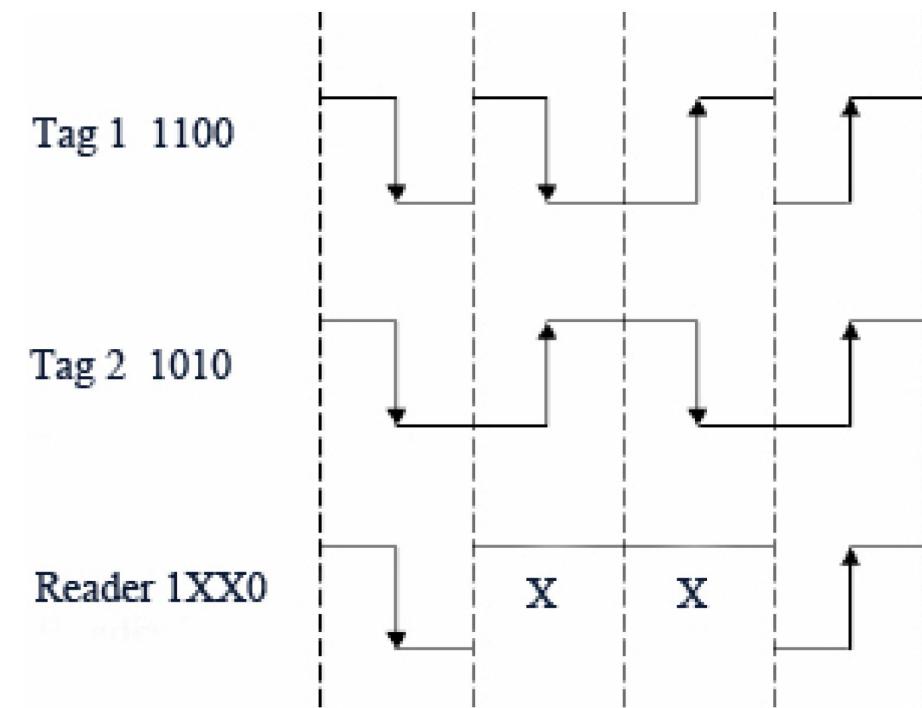
# Manchester Coding

- High to Low transition is mapped as binary logic-1
- Low to High transition is mapped as binary logic-0
- Transition occurs exactly in the middle of bit period.



# Collision Detection in Manchester Coding

- When two tags respond with different data, abnormal signals can be detected



# Memoryless Algorithm

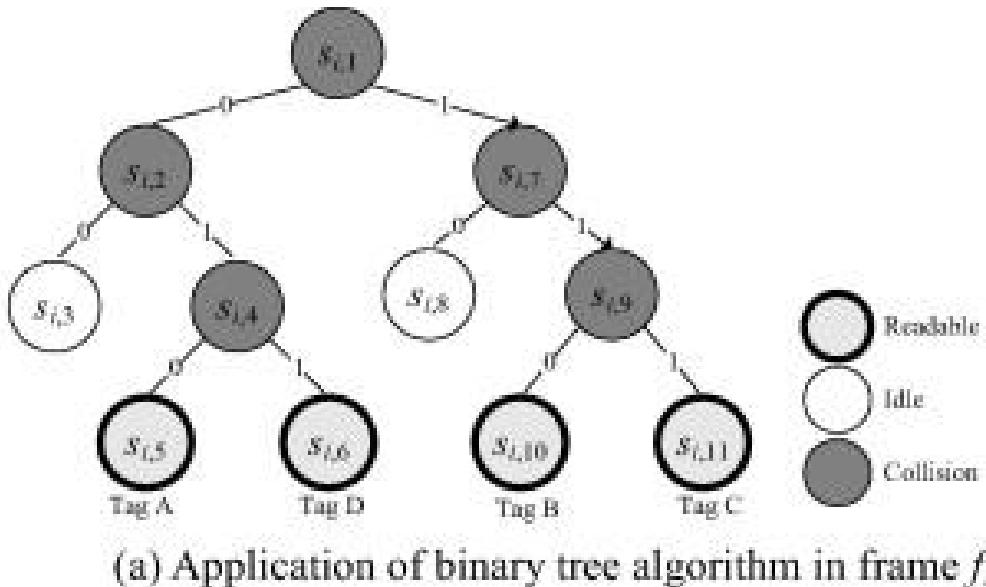
# Memoryless Algorithm

Can be applied when the reader does not know all the IDs

- Reader queries for tags
- Reader informs in case of collision and tags generates 0 or 1 randomly
- If 0 then tag retransmits on next query
- If 1 then tag becomes silent and starts incrementing its counter (which is initially zero)
- Counter incremented every time collision reported and decremented every time identification reported
- Tag remains silent till its counter becomes zero

# Example

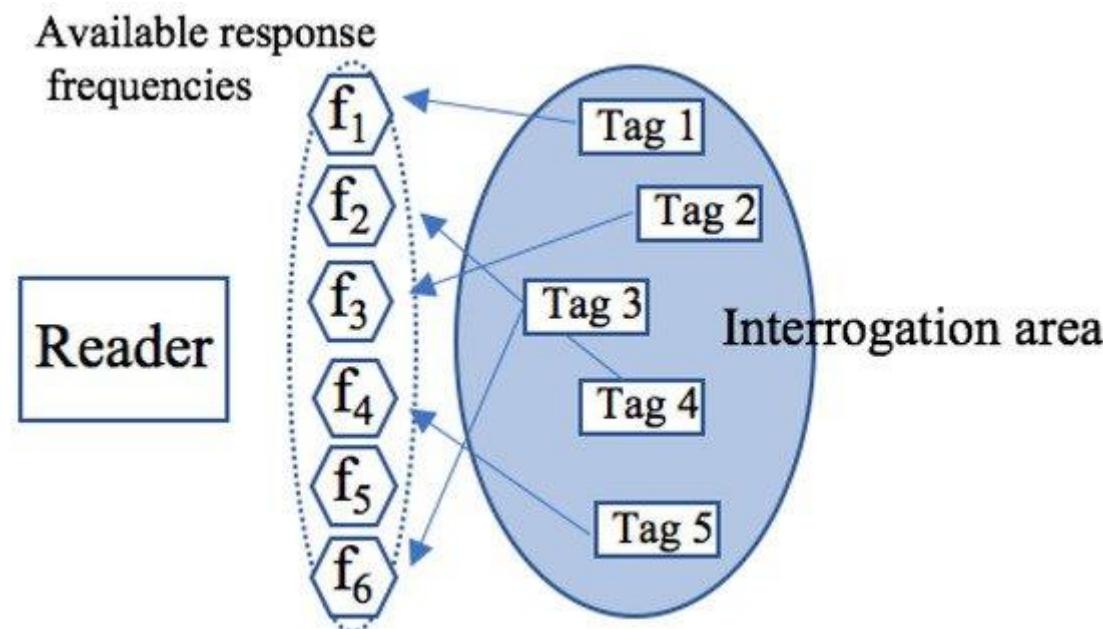
- Reader informs tags in case of collision and tags generate 0 or 1 randomly
- If 0 then tag retransmits on next query, else tag becomes silent and starts a counter.
- Counter incremented every time collision reported and decremented otherwise.



Slot	Counter value					Responding tag	Feedback message
	Reader	Tag A	Tag B	Tag C	Tag D		
1	0	0	0	0	0	A,B,C,D	Collision
2	1	0	1	1	0	A,D	Collision
3	2	1	2	2	1		Idle
4	1	0	1	1	0	A,D	Collision
5	2	0	2	2	1	A	Readable
6	1		1	1	0	D	Readable
7	0		0	0		B,C	Collision
8	1		1	1			Idle
9	0		0	0		B,C	Collision
10	1		0	1		B	Readable
11	0			0		C	Readable
12	-1						Terminate

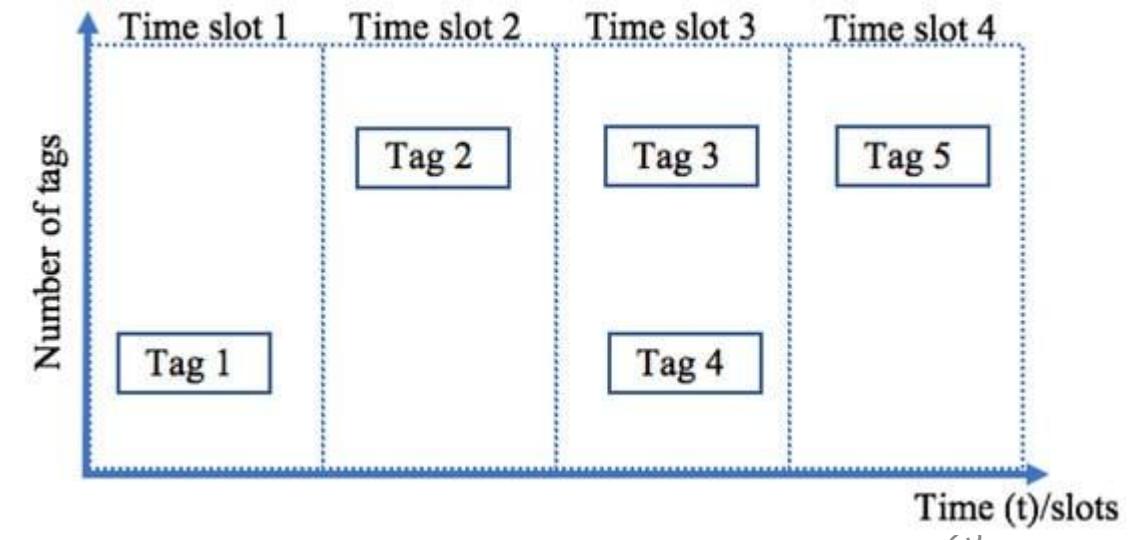
# FDMA

- **FDMA** : Interfering readers transmit at different frequency  
Adding tuning circuitry to the tags will increase the cost



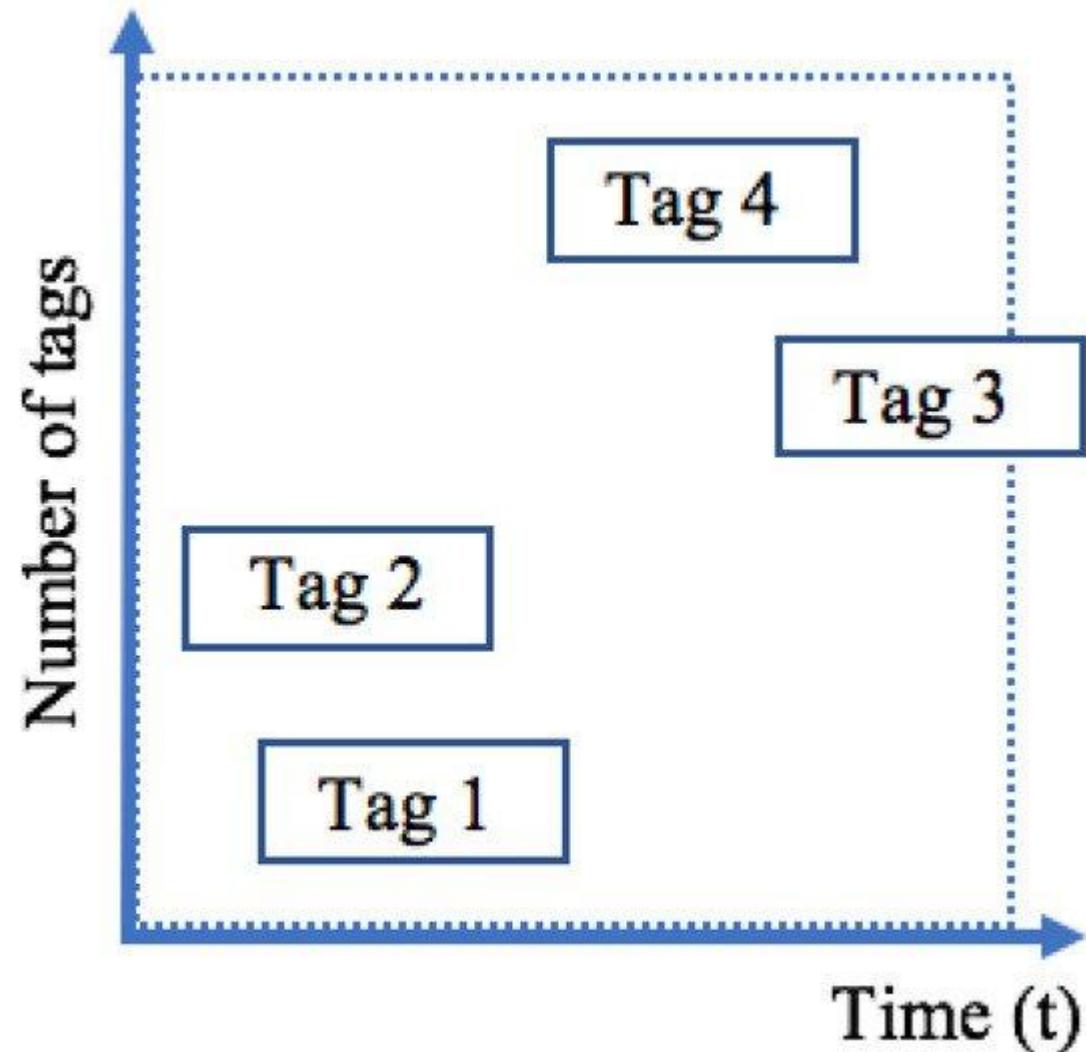
# TDMA

- TDMA (Time Division Multiple Access)
  - The largest group of RFID anti-collision protocols
  - Tag driven (tag talk first, TTF)
    - Tag transmits as it is ready
    - Aloha
    - SuperTag
      - Tags keep quiet and retransmit until reader acknowledges
  - Reader driven (reader talk first, RTF)
    - Polling, splitting, I-code, contactless



# CDMA

- **CDMA** : Interfering readers modulate signals with orthogonal codes
  - Requires complex circuitry at tags which will increase the cost of passive tags
  - Too complicate and too computationally intense for RFID tags as well

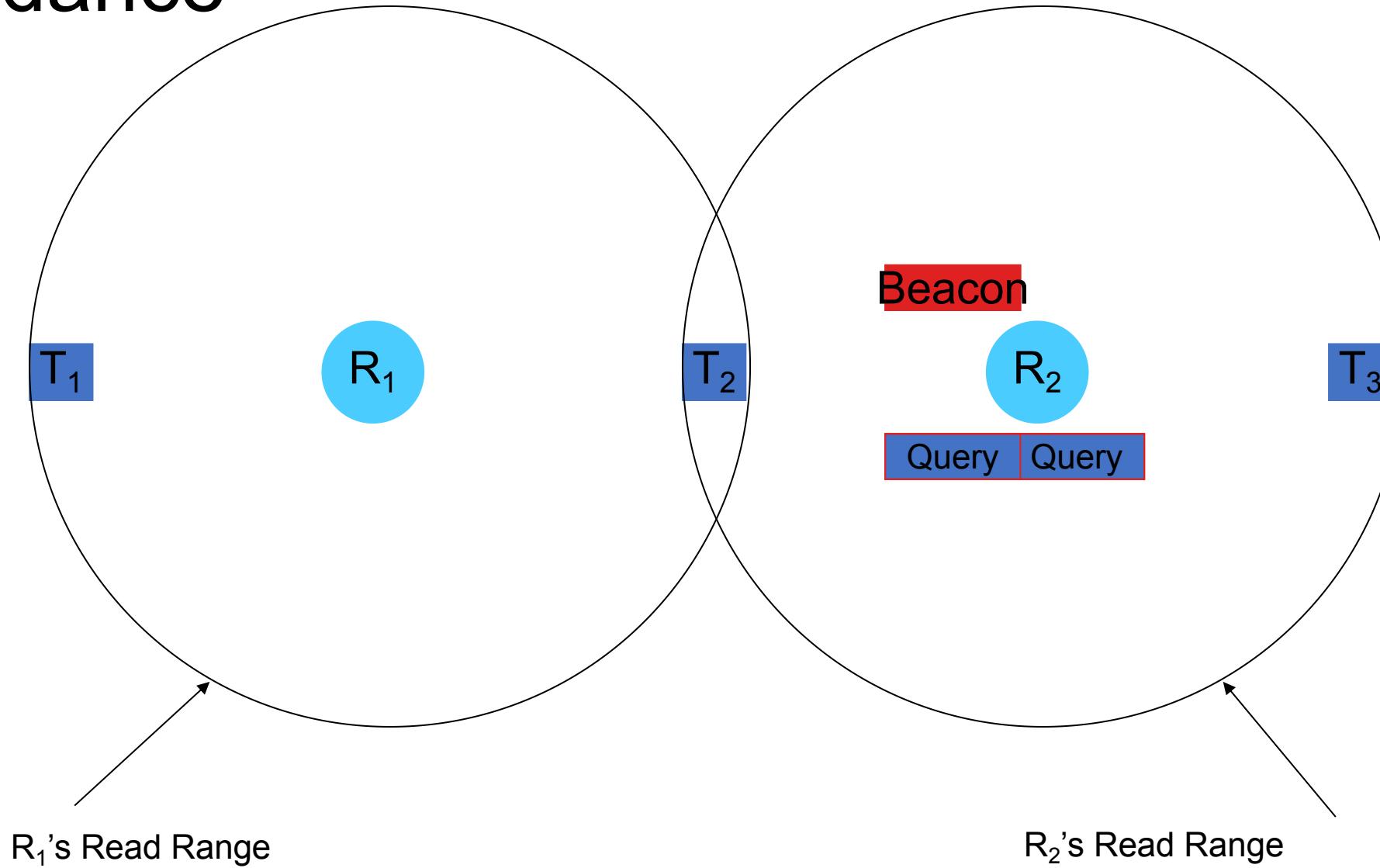


# Reader-to-Reader Collision Avoidance

# Reader-reader collision avoidance: beacon based solution

- A reader while reading tag, periodically sends a beacon on the control channel
- Assumptions
  - The range in the control channel is sufficient for a reader to communicate with all the possible readers that might interfere in the data channel

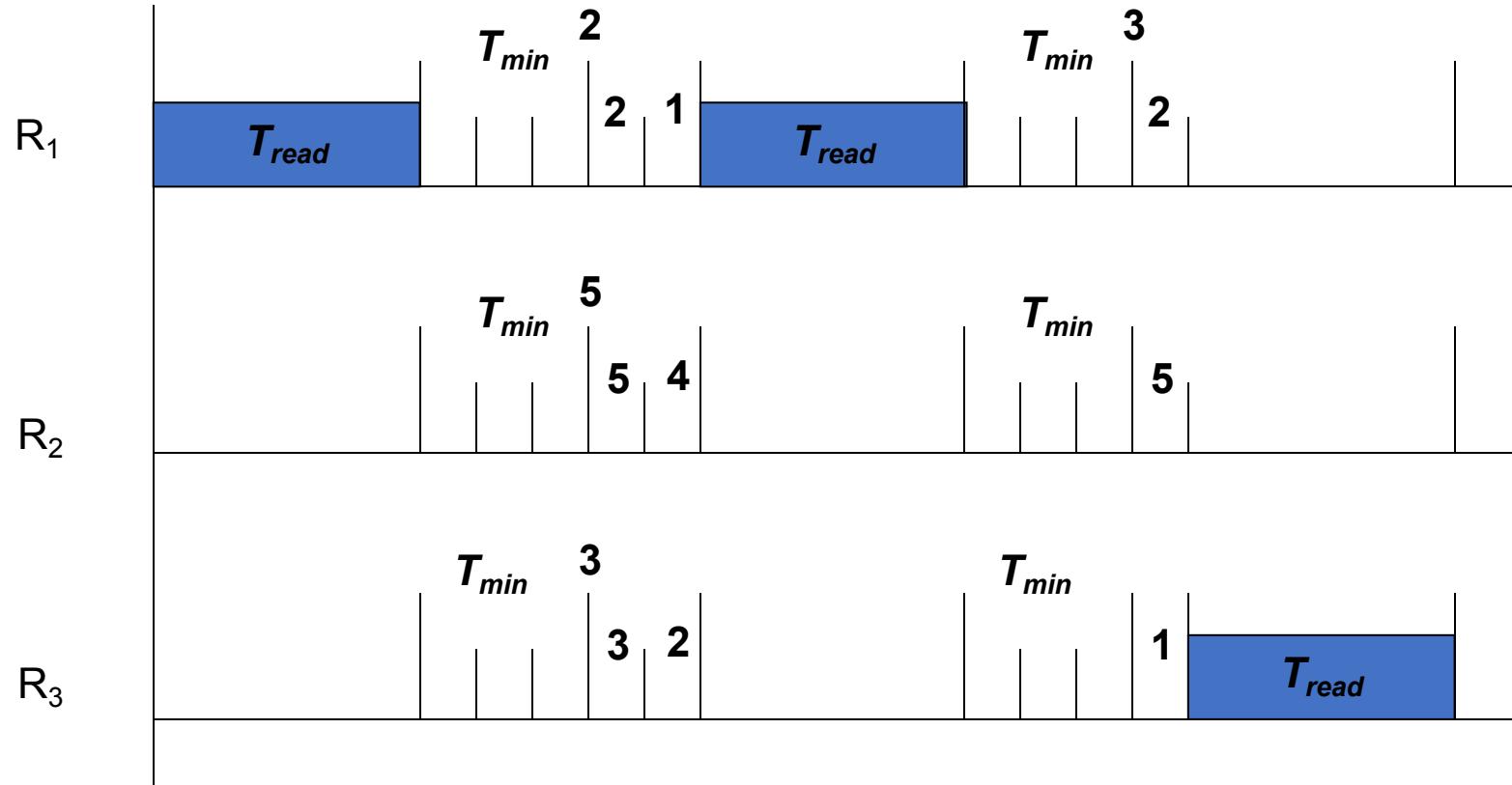
# PULSE Protocol for RFID Reader Collision Avoidance



# PULSE Protocol Overview

- Before communicating, a reader listens on the control channel for any beacon for  $T_{min}$  time
- If no beacon on the control channel for  $T_{min}$ , start communication on the data channel
- Reader periodically transmits a beacon on the control channel while communicating with the tags

# Contend backoff



R<sub>1</sub> chooses 2 BI, R<sub>2</sub> chooses 5 BI, R<sub>3</sub> chooses 3 BI