

# Lecture #8

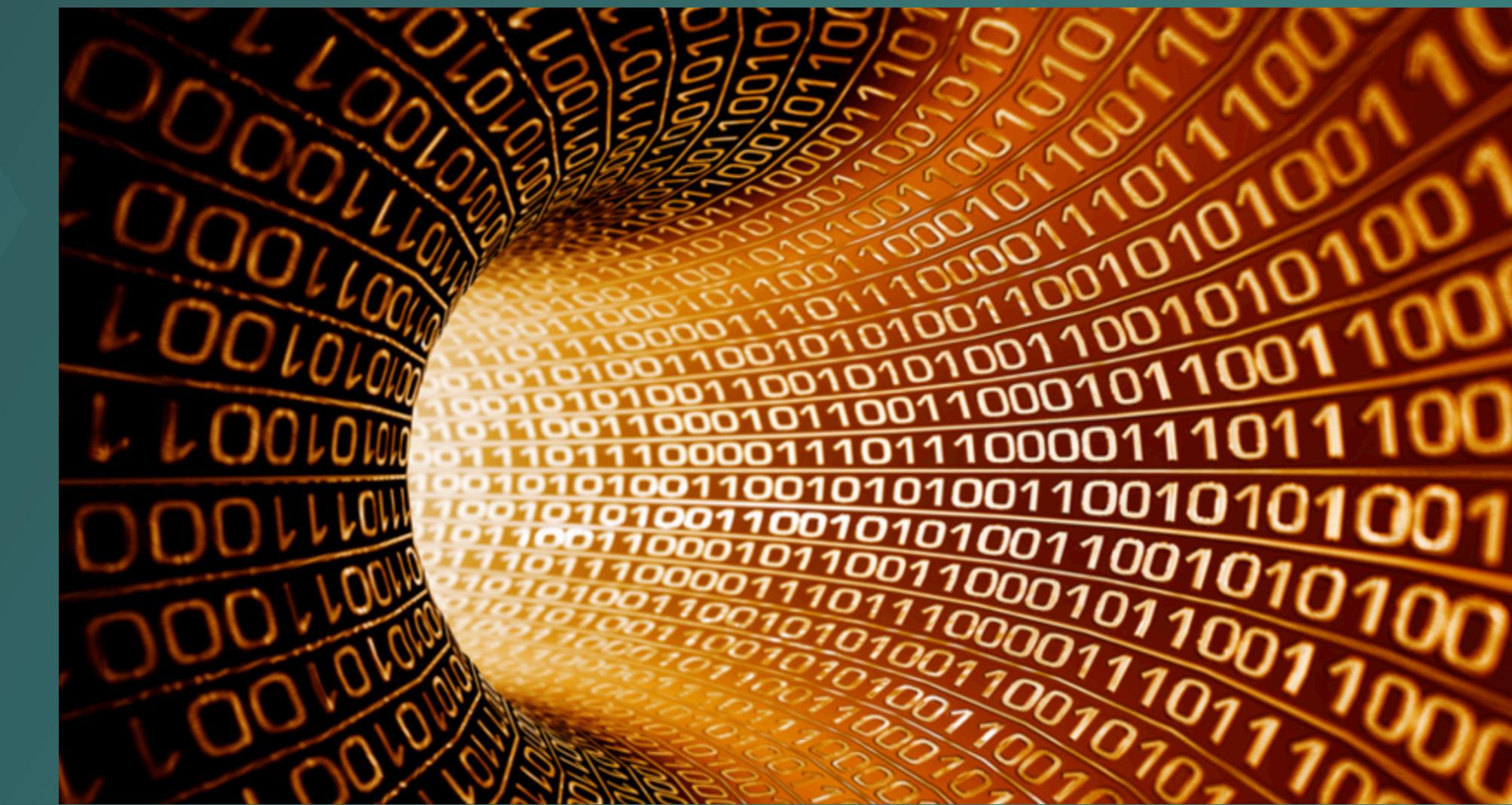
# System Services &

# Sensors

Mobile Applications 2022-2023

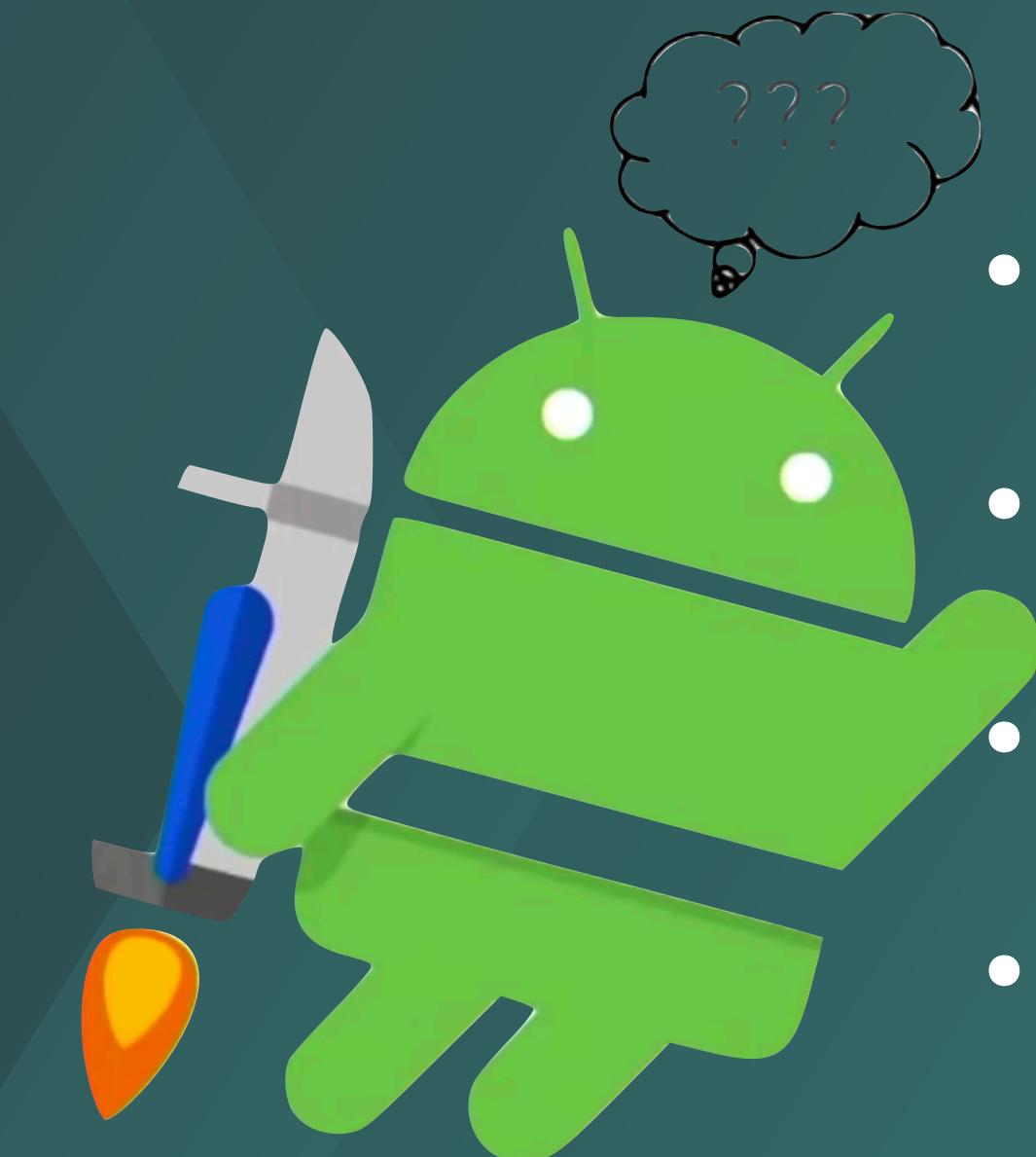
# Background Tasks

- Sending logs or tracking user progress.
- Upload images, videos or session data.
- Syncing data.
- Processing data.



# Options

- Threads
- Executors
- Services
- AsyncTasks
- Handlers and Loopers
- Coroutines
- Jobs (API 21+)
- GcmNetworkManager
- SyncAdapters
- Loaders
- AlarmManager
- WorkManager



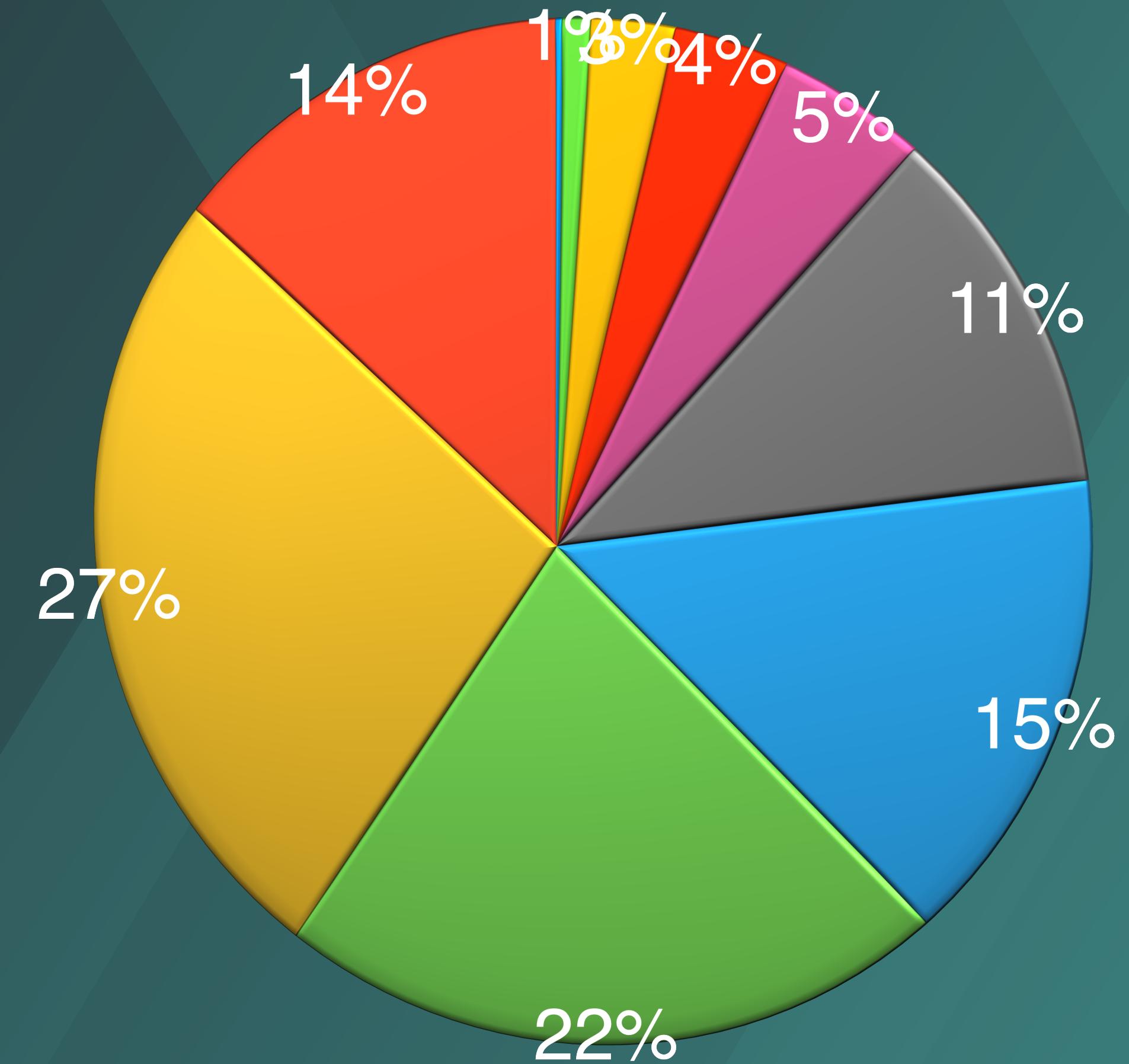
# Battery Optimizations

- Doze mode
- App standby
- Limited implicit broadcasts
- Release cached wakelocks
- Background service limitations
- App standby buckets
- Background restricted apps



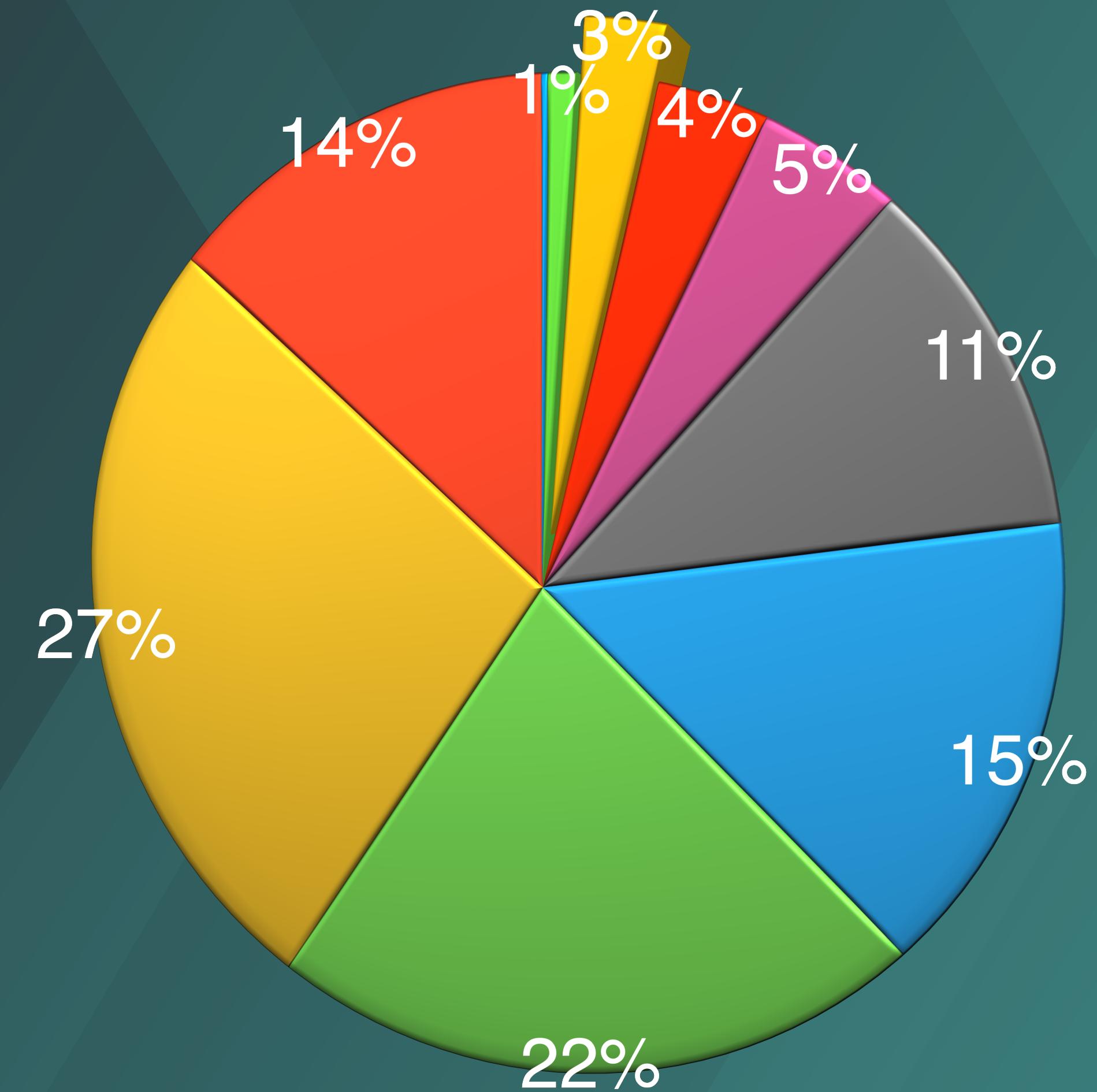
# Compatibility

- Jelly Bean
- Oreo
- KitKat
- Pie
- Lollipop
- Android 10
- Marshmallow
- Android 11
- Nougat
- Android 12

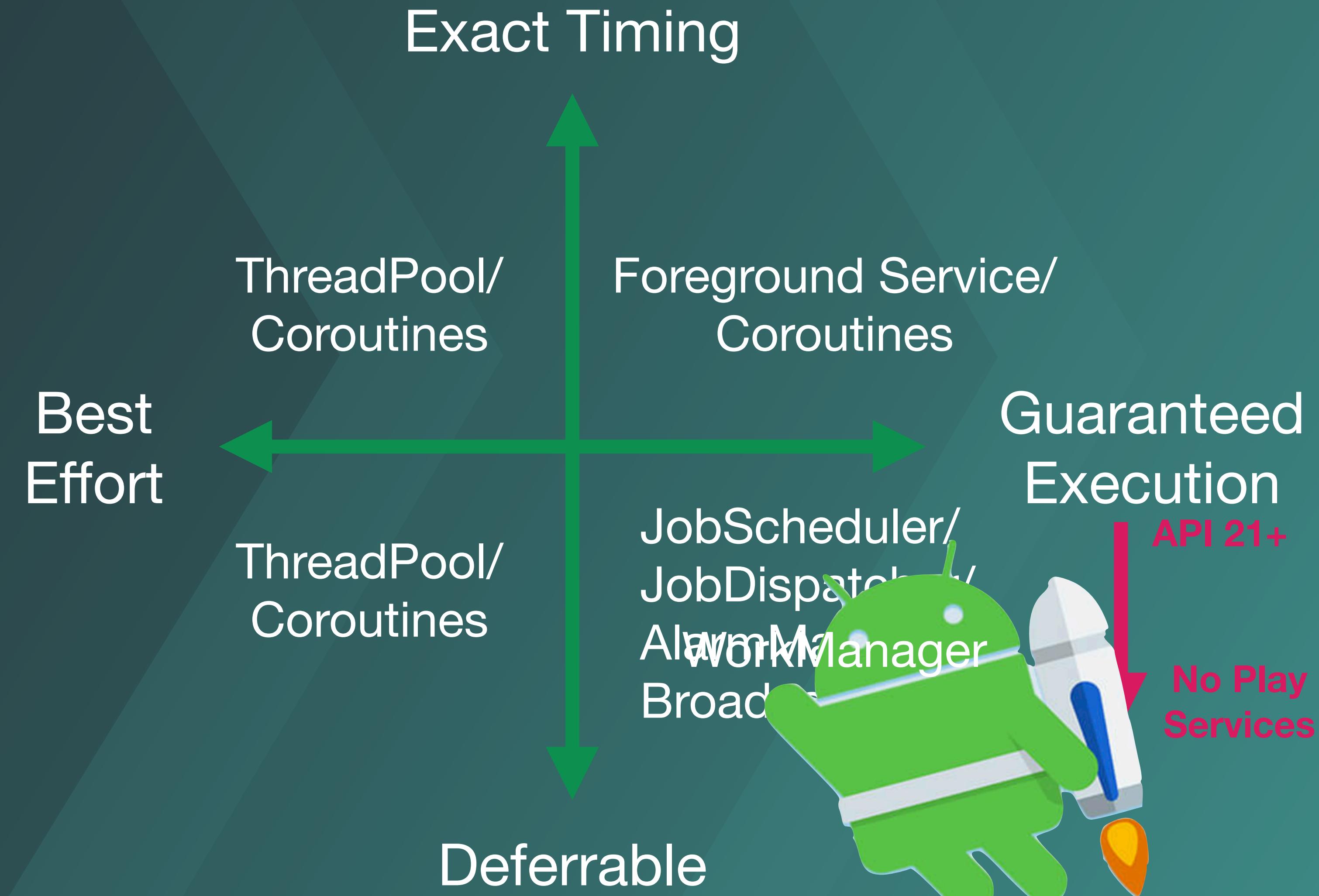


# Compatibility

- Jelly Bean
- Oreo
- KitKat
- Pie
- Lollipop
- Android 10
- Marshmallow
- Android 11
- Nougat
- Android 12



# Requirements



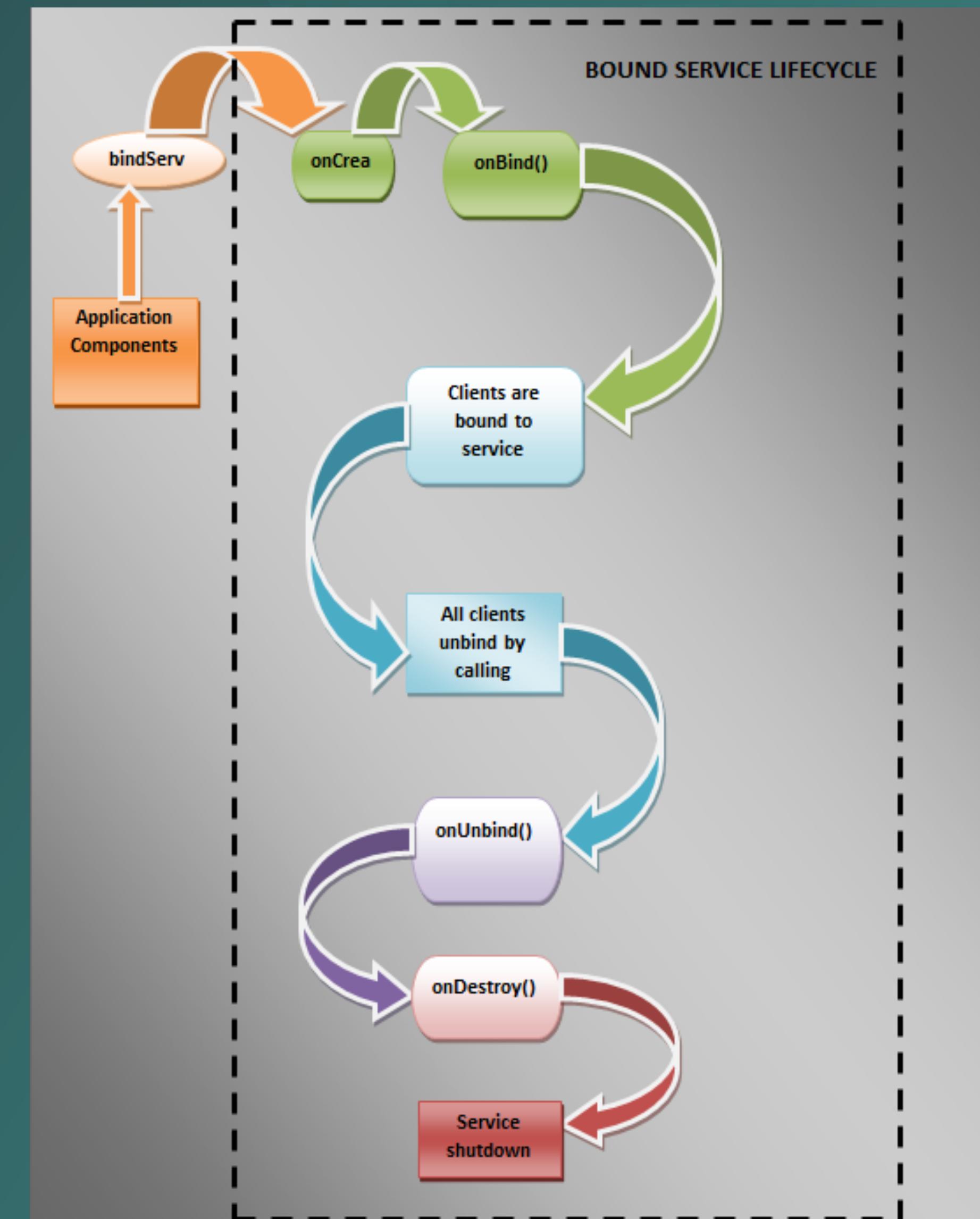
# Services

- Perform long-running operations in the background.
- It does not provide a user interface.
- Continues to run even if the user switches to another application.
- Eg.:
  - Network transactions.
  - Play music.
  - Perform I/O.
  - Use a content provider.



# Services

- Types:
  - Foreground
  - Background
  - Bound



# Basics

- onStartCommand()
- onBind()
- onCreate()
- onDestroy()

The system invokes this method by calling `startService()` when another component (such as an activity) requests that component wants to bind with the service (such as to perform RPC). The system invokes this method to open the service for the client. In your implementation of this method, you must provide an interface that clients use to communicate with the service by returning `IBinder`. It's up to you to decide if this responsibility falls on the client or the service. You must always implement this method; however, if you don't want to allow binding, you should return null.

# Declaring a service in the manifest

```
<manifest ... >  
...  
<application ... >  
    <service android:name=".ExampleService" />  
    ...android:description="Service Description"  
</application>  
</manifest>  
</application>  
</manifest>
```

# Creating a Service

- **Service**
  - Base class for all services.
  - Create and manage a new thread on your own.
- IntentService
  - Subclass of Service.
  - Uses a worker thread.
  - Implement onHandleIntent()

```
/**  
 * A constructor is required  
 * with a name for the worker thread.  
 */  
  
class HelloService extends Service {  
    private ServiceLooper mServiceLooper = null;  
    private ServiceHandler mServiceHandler = null;  
    * The IntentService calls this method  
    // Hand the thread to the worker thread with the thread  
    private inner class ServiceHandler(looper: Looper) : Handler(looper)  
        *Override fun handleMessage(msg: Message) {  
        * stopNoisyWorkIfAppropriate work here, like download a file.  
        * // For our sample, we just sleep for 5 seconds.  
        override fun onHandleIntent(intent: Intent?) {  
            try Thread.sleep(5000)  
            Thread.sleepInterruptedException) {  
        } catch (e: InterruptedException) {  
            // Thread.currentThread().interrupt()  
            Thread.currentThread().interrupt()  
        } // Stop the service using the startId, so that we don't stop  
        } // the service in the middle of handling another job  
    } stopSelf(msg.arg1)
```

# Service Management

- Starting a service
- Stopping a service

```
Intent(this, HelloService.class).also {  
    intent -> startService(intent)  
}  
        stopSelf()
```

# Notify the User

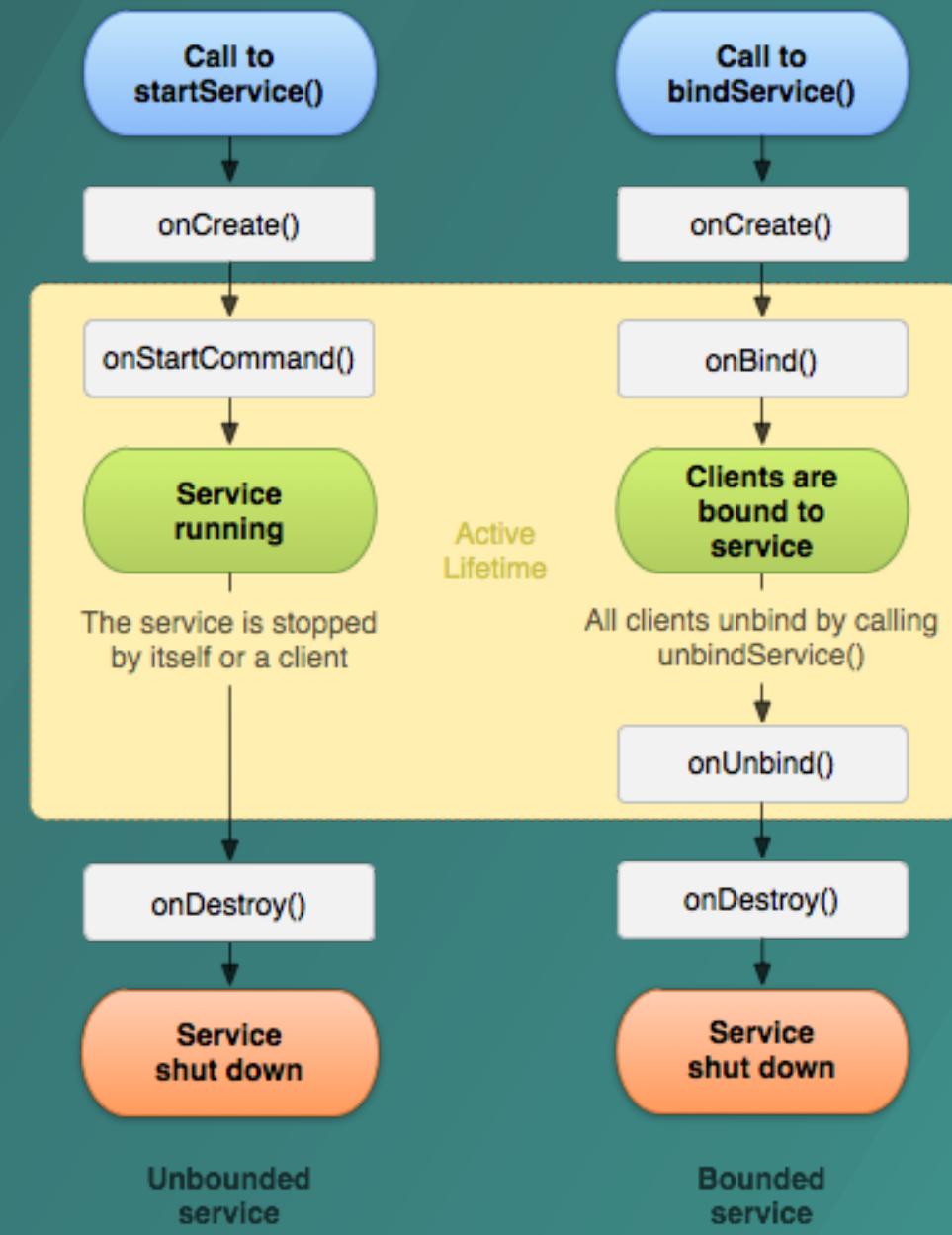
- Toast Notifications
- Status Bar Notifications

```
val pendingIntent: PendingIntent =  
    Intent(this, ExampleActivity::class.java).let { notificationIntent ->  
        PendingIntent.getActivity(this, 0, notificationIntent, 0)  
    }  
    val text = "Hello toast!"  
    val notification=NotificationCompatNotification.Builder(this,  
        CHANNEL_DEFAULT_IMPORTANCE)  
        .setCustomTitle(text)  
        .setCustomText(applicationContext.text, duration)  
        .setContentText(getText(R.string.notification_message))  
        .setSmallIcon(R.drawable.icon)  
        .setContentIntent(pendingIntent)  
        .setTicker(getText(R.string.ticker_text))  
        .build()  
  
startForeground(ONGOING_NOTIFICATION_ID, notification)
```

# Service Lifecycle

**DEMO**

```
class ExampleService : Service() {  
    private var mStartMode: Int = 0 // how to behave if the service is killed  
    private var mBinder: IBinder? = null // interface for clients that bind  
    private var mAllowRebind: Boolean = false // whether onRebind should be used  
  
    override fun onCreate() {  
        // The service is being created  
    }  
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {  
        // The service is starting, due to a call to startService()  
        return mStartMode  
    }  
    override fun onBind(intent: Intent): IBinder? {  
        // A client is binding to the service with bindService()  
        return mBinder  
    }  
    override fun onUnbind(intent: Intent): Boolean {  
        // All clients have unbound with unbindService()  
        return mAllowRebind  
    }  
    override fun onRebind(intent: Intent) {  
        // A client is binding to the service with bindService(),  
        // after onUnbind() has already been called  
    }  
    override fun onDestroy() {  
        // The service is no longer used and is being destroyed  
    }  
}
```



# Alarm Manager

- Alarm types:
  - ELAPSED\_REALTIME
  - ELAPSED\_REALTIME\_WAKEUP
  - RTC
  - RTC\_WAKEUP

```
alarmMgr?.setInexactRepeating(  
    AlarmManager.ELAPSED_REALTIME_WAKEUP,  
    SystemClock.elapsedRealtime() + AlarmManager.INTERVAL_HALF_HOUR,  
    AlarmManager.INTERVAL_HALF_HOUR,  
    alarmIntent  
)  
    // Cancel the alarm.  
    alarmMgr?.cancel(alarmIntent)
```

# Alarm at 14:00

DEMO

```
// Set the alarm to start at approximately 2:00 p.m.  
val calendar: Calendar = Calendar.getInstance().apply {  
    timeInMillis = System.currentTimeMillis()  
    set(Calendar.HOUR_OF_DAY, 14)  
}
```

```
// With setInexactRepeating(), you have to use one of the AlarmManager interval  
// constants--in this case, AlarmManager.INTERVAL_DAY.  
alarmMgr?.setInexactRepeating(  
    AlarmManager.RTC_WAKEUP,  
    calendar.timeInMillis,  
    AlarmManager.INTERVAL_DAY,  
    alarmIntent  
)
```

# JobScheduler

- Register the Service

```
<service  
    android:name=".MyJobService"  
    android:permission="android.permission.BIND_JOB_SERVICE"  
    android:exported="true"/>
```

- Schedule a Job

```
val builder = JobInfo.Builder(jobId++, serviceComponent)  
builder.setMinimumLatency(delay.toLong() * TimeUnit.SECONDS.toMillis(1))  
builder.setOverrideDeadline(deadline.toLong() * TimeUnit.SECONDS.toMillis(1))  
builder.setRequiredNetworkType(JobInfo.NETWORK_TYPE_ANY)  
builder.run {  
    setRequiresDeviceIdle(requiresIdleCheckbox.isChecked)  
    setRequiresCharging(requiresChargingCheckBox.isChecked)  
    setExtras(extras)  
}  
(getSystemService(Context.JOB_SCHEDULER_SERVICE) as JobScheduler).schedule(builder.build())
```

The Job



# Define The Job

DEMO

```
class MyJobService : JobService() {  
    override fun onStartCommand(intent: Intent, flags: Int, startId: Int): Int {  
        activityMessenger = intent.getParcelableExtra(MESSENGER_INTENT_KEY)  
        return Service.START_NOT_STICKY  
    }
```

```
    override fun onStartJob(params: JobParameters): Boolean {  
        // The work that this service "does"  
        // ...  
        // Return true as there's more work to be done with this job.  
        return true  
    }
```

```
    override fun onStopJob(params: JobParameters): Boolean {  
        // Stop tracking these job parameters, as we've 'finished' executing.  
        // ...  
        // Return false to drop the job.  
        return false  
    }  
}
```

[developer.android.com/topic/performance/scheduling](http://developer.android.com/topic/performance/scheduling)

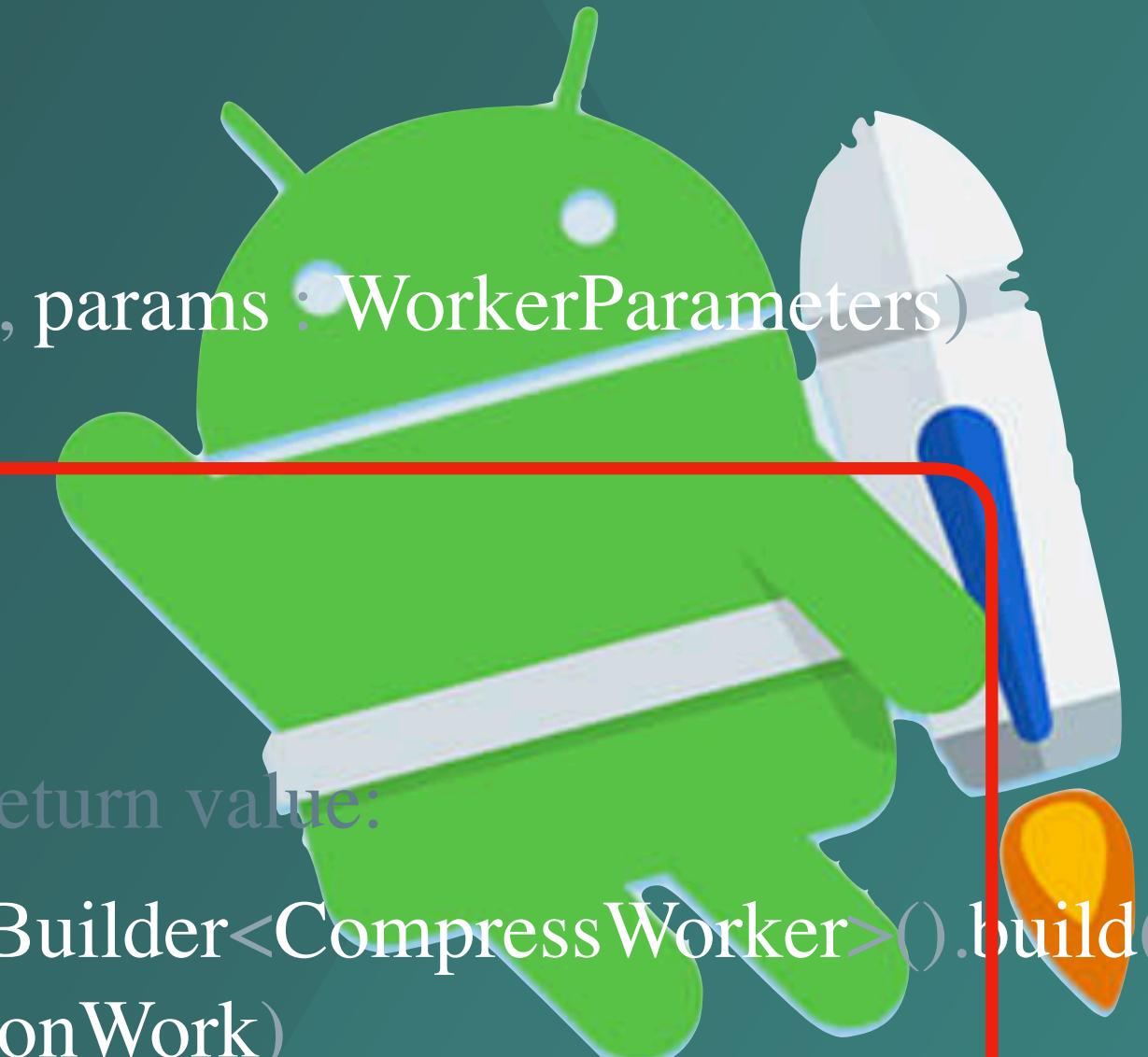
# WorkManager

- Core Classes:

- Worker
- WorkRequest

- OneTimeWorkRequest

```
class CompressWorker(context : Context, params : WorkerParameters) : Worker(context, params) {  
    override fun doWork(): Result {  
        // The actual work!  
        myCompress()  
        // Indicate success or failure with your return value:  
        return Result.SUCCESS  
    }  
    val compressionWork = OneTimeWorkRequestBuilder<CompressWorker>().build()  
    WorkManager.getInstance().enqueue(compressionWork)  
    // - RETRY tells WorkManager to try this task again later  
    // - FAILURE says not to try again.  
}  
}
```



Runs on a background thread

# Constraints



```
// Create a Constraints object that defines when the task should run
val myConstraints = Constraints.Builder()
    .setRequiresDeviceIdle(true)
    .setRequiresCharging(true)
// ...
.build()
```

```
// then create a OneTimeWorkRequest that uses those constraints
val compressionWork = OneTimeWorkRequestBuilder<CompressWorker>()
    .setConstraints(myConstraints)
    .build()
```

**Cancel the task**

```
val compressionWorkId:UUID = compressionWork.getId()
WorkManager.getInstance().cancelWorkById(compressionWorkId)
```

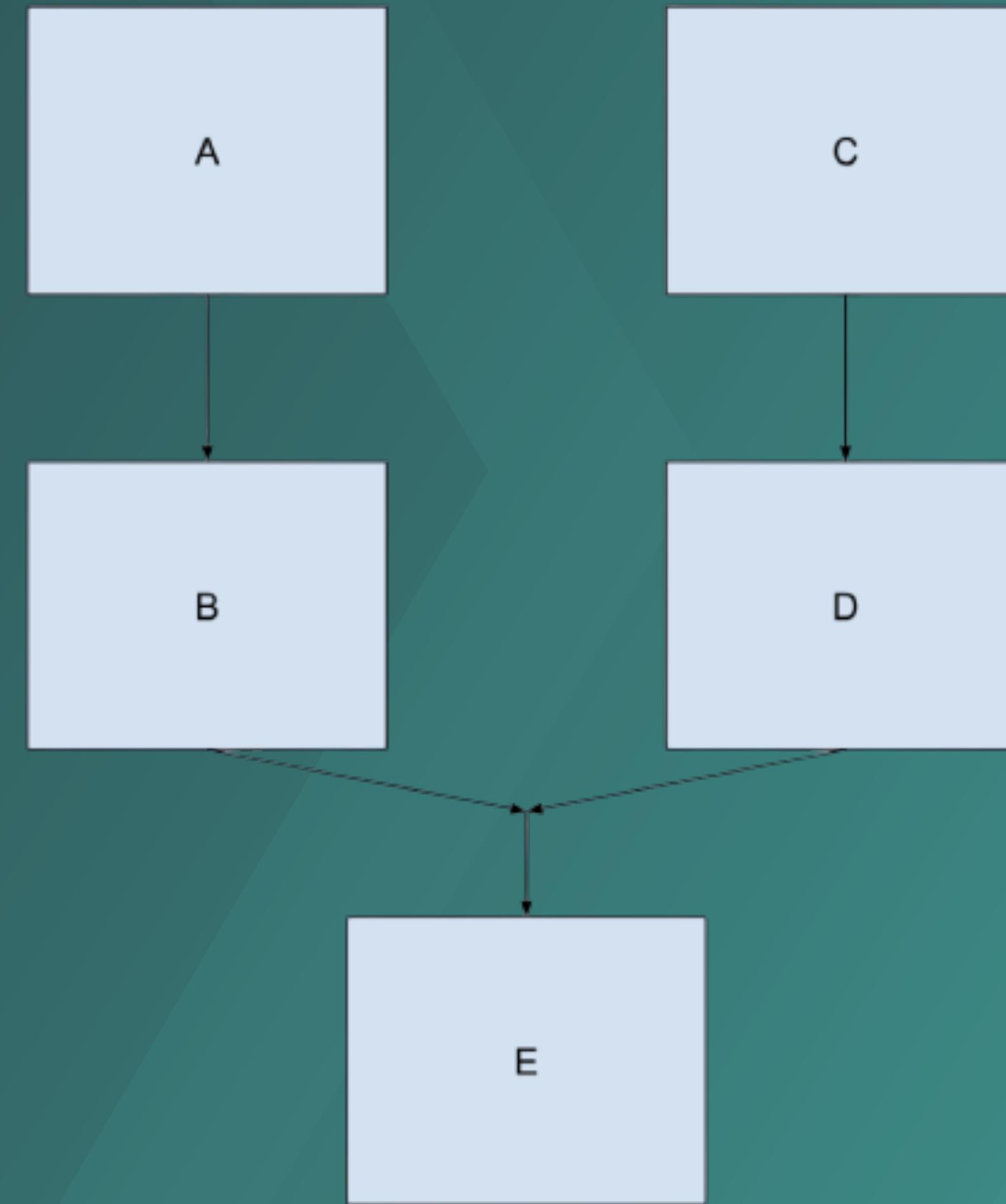
# Chained Tasks

```
WorkManager.getInstance()  
.beginWith(workA)  
// Note: WorkManager.beginWith() returns a  
// WorkContinuation object; the following calls are  
// to WorkContinuation methods  
WorkManager.getInstance()  
.then(workB) // FYI, then() returns a new WorkContinuation instance  
// First, run all the A tasks (in parallel):  
.then(workC)  
.beginWith(workA1, workA2, workA3)  
.enqueue()  
// ...when all A tasks are finished, run the single B task:  
.then(workB)  
// ...then run the C tasks (in any order):  
.then(workC1, workC2)  
.enqueue()
```

# Chained tasks

**DEMO**

```
val chain1 = WorkManager.getInstance()  
.beginWith(workA)  
.then(workB)  
val chain2 = WorkManager.getInstance()  
.beginWith(workC)  
.then(workD)  
val chain3 = WorkContinuation  
.combine(chain1, chain2)  
.then(workE)  
chain3.enqueue()
```



# Sensors

- Motion

The ~~Position~~ sensors measure acceleration forces and rotational forces along three axes.

The ~~Environment~~ sensors measure the physical position of a device.

This category includes: various environmental parameters.

- Accelerometers.

- Orientation sensors.

This category includes:

- Gravity sensors.

- Magnetometers.

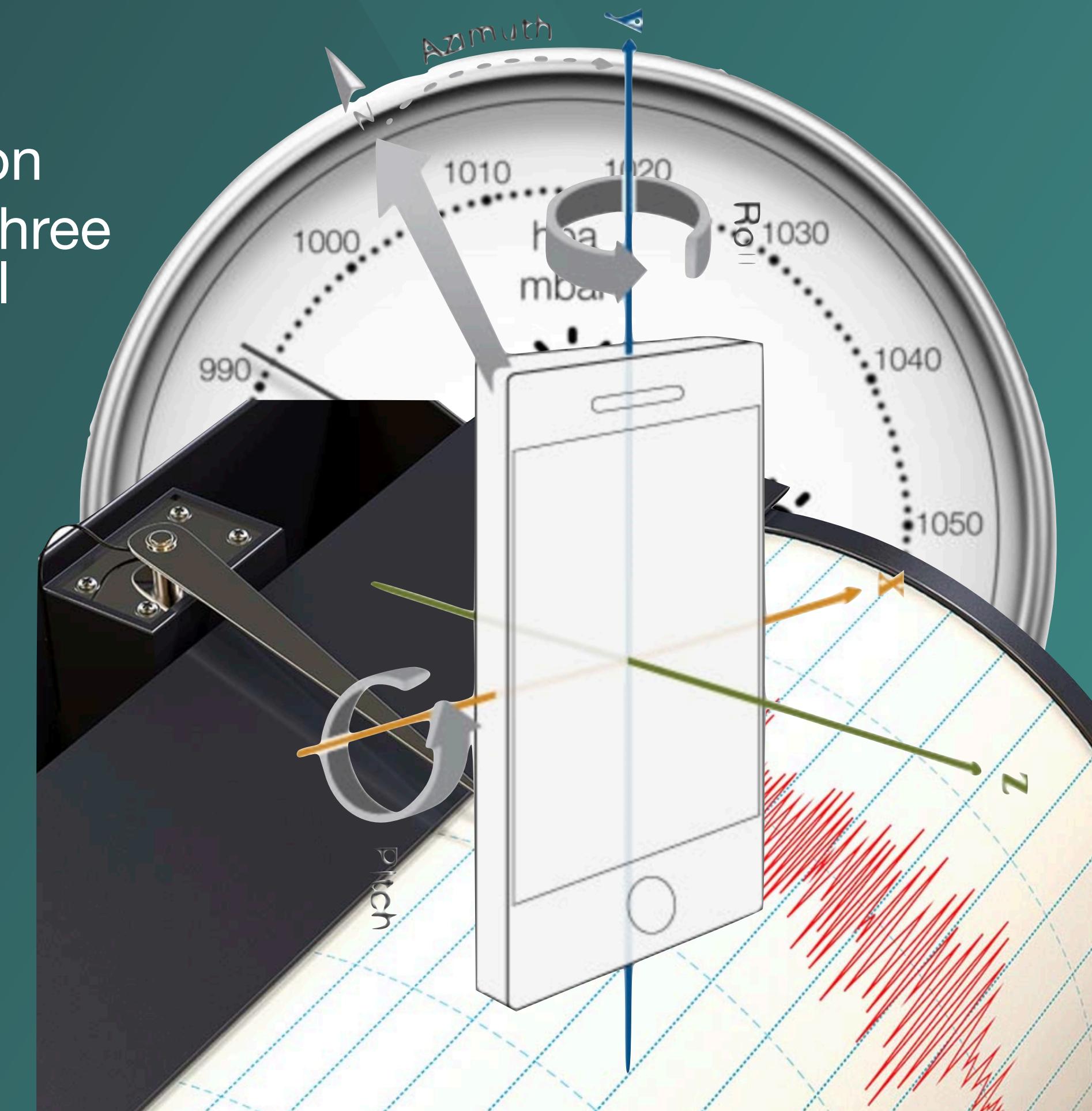
- Barometers.

- Gyroscopes.

- Photometers.

- Rotational vector sensors.

- Thermometers.



# Framework

```
private lateinit var mSensorManager: SensorManager
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
if(mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY_FIELD) != null) {
    val gravSensors: List<Sensor> = mSensorManager.getSensorList(Sensor.TYPE_GRAVITY)
    // And here we're using it.
}
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
// And here we're using it.
val deviceSensors: List<Sensor> = mSensorManager.getSensorList(Sensor.TYPE_ALL)
it.vendor.contains("Google LLC") && it.version == 3
}
class SensorActivity : Activity(), SensorEventListener {
    private lateinit var mSensorManager: SensorManager
    private var mLight: Sensor? = null
    mSensor = if (mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
public override fun onCreate(savedInstanceState: Bundle?) {
    mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
} setContentView(R.layout.main)
// Sorry, there are no accelerometers on your device.
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT)
}
```

- SensorManager
- Sensor
- SensorEvent
- SensorEventListener

# Motion

Sensor	Sensor event data	Description	Units of measure
TYPE_ACCELEROMETER	<code>SensorEvent.values[0]</code>	Acceleration force along the x axis (including gravity).	$\text{m/s}^2$
	<code>SensorEvent.values[1]</code>	Acceleration force along the y axis (including gravity).	
	<code>SensorEvent.values[2]</code>	Acceleration force along the z axis (including gravity).	
TYPE_ACCELEROMETER_UNCALIBRATED	<code>SensorEvent.values[0]</code>	Measured acceleration along the X axis without any bias compensation.	$\text{m/s}^2$
	<code>SensorEvent.values[1]</code>	Measured acceleration along the Y axis without any bias compensation.	
	<code>SensorEvent.values[2]</code>	Measured acceleration along the Z axis without any bias compensation.	
	<code>SensorEvent.values[3]</code>	Measured acceleration along the X axis with estimated bias compensation.	
	<code>SensorEvent.values[4]</code>	Measured acceleration along the Y axis with estimated bias compensation.	
	<code>SensorEvent.values[5]</code>	Measured acceleration along the Z axis with estimated bias compensation.	

# Position

Sensor	Sensor event data	Description	Units of measure
TYPE_GAME_ROTATION_VECTOR	<code>SensorEvent.values[0]</code>	Rotation vector component along the x axis $(x * \sin(\theta/2))$ .	Unitless
	<code>SensorEvent.values[1]</code>	Rotation vector component along the y axis $(y * \sin(\theta/2))$ .	
	<code>SensorEvent.values[2]</code>	Rotation vector component along the z axis $(z * \sin(\theta/2))$ .	
TYPE_GEO MAGNETIC_ROTATION_VECTOR	<code>SensorEvent.values[0]</code>	Rotation vector component along the x axis $(x * \sin(\theta/2))$ .	Unitless
	<code>SensorEvent.values[1]</code>	Rotation vector component along the y axis $(y * \sin(\theta/2))$ .	
	<code>SensorEvent.values[2]</code>	Rotation vector component along the z axis $(z * \sin(\theta/2))$ .	
TYPE_MAGNETIC_FIELD	<code>SensorEvent.values[0]</code>	Geomagnetic field strength along the x axis.	$\mu\text{T}$
	<code>SensorEvent.values[1]</code>	Geomagnetic field strength along the y axis.	
	<code>SensorEvent.values[2]</code>	Geomagnetic field strength along the z axis.	
TYPE_MAGNETIC_FIELD_UNCALIBRATED	<code>SensorEvent.</code>	Geomagnetic field strength (without hard	$\mu\text{T}$

# Environment

DEMO

Sensor	Sensor event data	Units of measure	Data description
TYPE_AMBIENT_TEMPERATURE	event.values[0]	°C	Ambient air temperature.
TYPE_LIGHT	event.values[0]	lx	Illuminance.
TYPE_PRESSURE	event.values[0]	hPa or mbar	Ambient air pressure.
TYPE_RELATIVE_HUMIDITY	event.values[0]	%	Ambient relative humidity.
TYPE_TEMPERATURE	event.values[0]	°C	Device temperature. <sup>1</sup>

 <sup>1</sup> Implementations vary from device to device. This sensor was deprecated in Android 4.0 (API Level 14).

# Lecture outcomes

- Use existing system services.
- Define custom services.
- Understand the user notifications API.
- Consume data from sensors.

