

Lecture #8

System Services &

Sensors

Mobile Applications 2018-2019

Housekeeping Notes

**Valid ONLY for December 3rd, 2018
For students attending the 936/1 laboratory!**

Orar C.d.asociat BARABAS Attila

Ziua	Orele	Frecventa	Sala	Anul	Formatia	Tipul	Disciplina
Luni	8-10	sapt. 2	L306	3 Informatica - in limba engleza	936/1	Laborator	Programare pentru dispozitive mobile
Luni	8-10	sapt. 1	L306	3 Informatica - in limba engleza	936/2	Laborator	Programare pentru dispozitive mobile

Housekeeping Notes

**Valid ONLY for December 3rd, 2018
For students attending the 936/1 laboratory!**

Orar C.d.asociat BARABAS Attila

Ziua	Orele	Frecventa	Sala	Anul	Formatia	Tipul	Disciplina
Luni	8-10	sapt. 2	L306	3 Informatica - in limba engleza	936/1	Laborator	Programare pentru dispozitive mobile
Luni	8-10	sapt. 1	L306	3 Informatica - in limba engleza	936/2	Laborator	Programare pentru dispozitive mobile

**Will be postponed to Wednesday December 5th, 2018 (same week)
L306 room, from 12:00.**

Orar C.d.asociat BARABAS Attila

Ziua	Orele	Frecventa	Sala	Anul	Formatia	Tipul	Disciplina
WED	12-14	sapt. 2	L306	3 Informatica - in limba engleza	936/1	Laborator	Programare pentru dispozitive mobile
Luni	8-10	sapt. 1	L306	3 Informatica - in limba engleza	936/2	Laborator	Programare pentru dispozitive mobile



hipMenu

December 10th, 2018

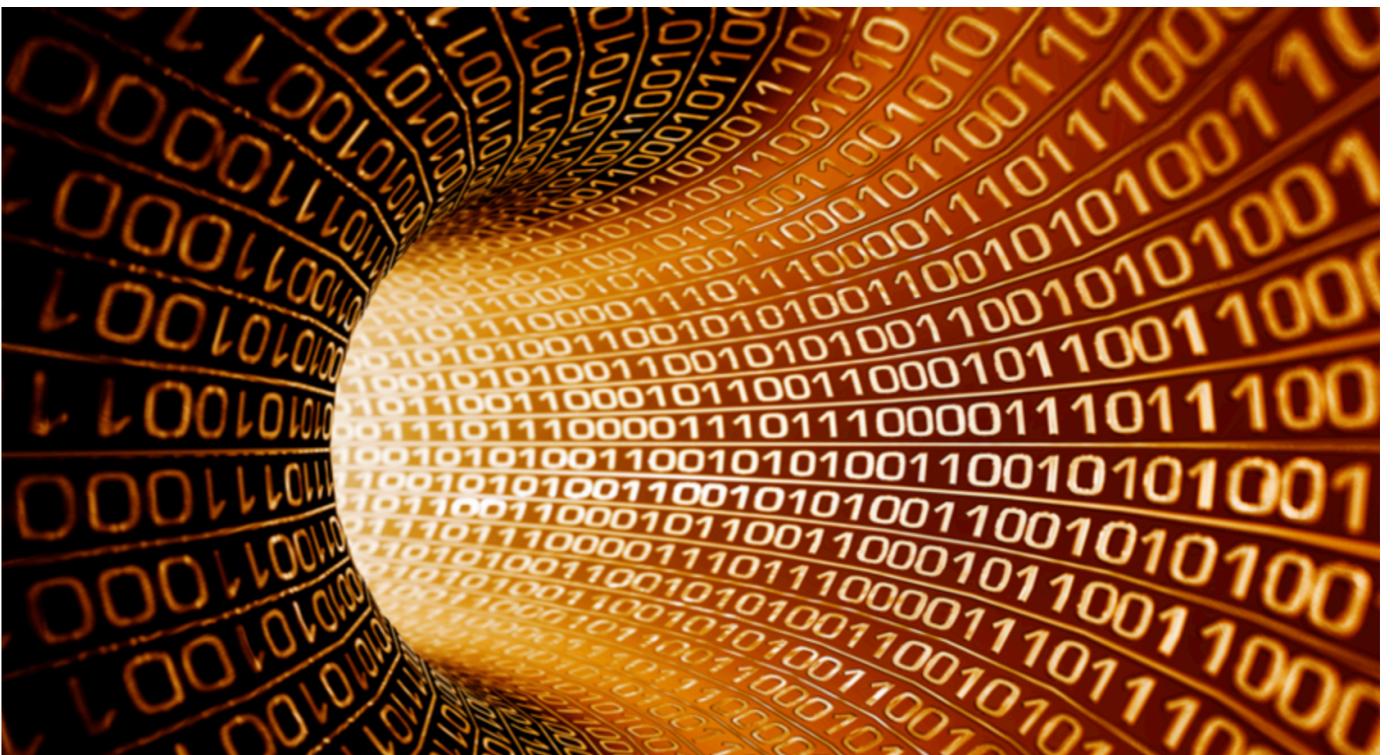


December 10th, 2018

+1p Final
Laboratory Grade

Background Tasks

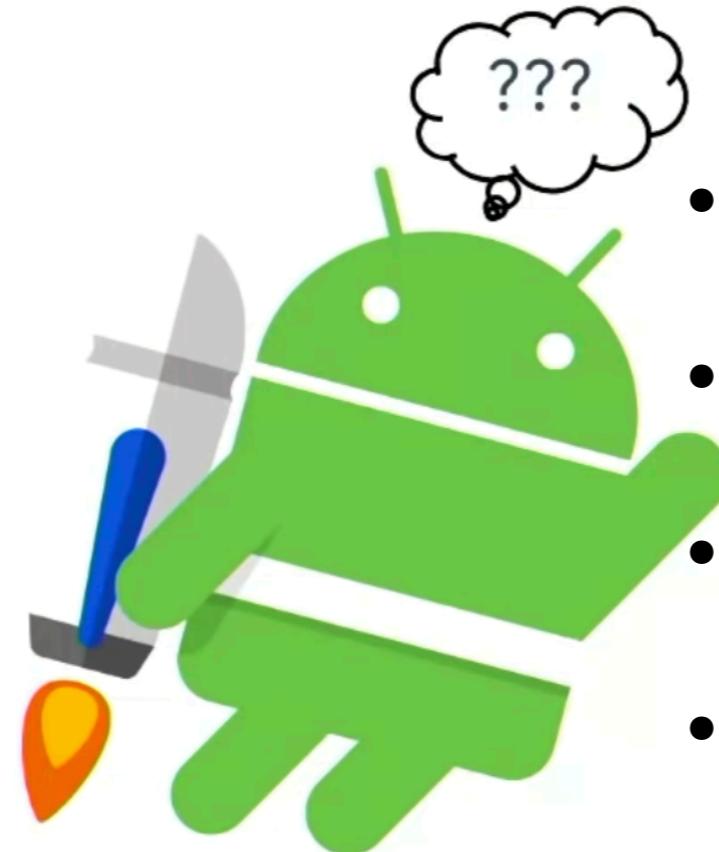
- Sending logs or tracking user progress.
- Upload images, videos or session data.
- Synching data.
- Processing data.



Options

Options

- Threads
- Executors
- Services
- AsyncTasks
- Handlers and Loopers



- Jobs (API 21+)
- GcmNetworkManager
- SyncAdapters
- Loaders
- AlarmManager

Battery Optimizations



Battery Optimizations

- Doze mode
- App standby



Battery Optimizations

- Doze mode
- App standby
- Limited implicit broadcasts



Battery Optimizations

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



Battery Optimizations

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



Compatibility

<https://developer.android.com/about/dashboards/> (October 26th, 2018)

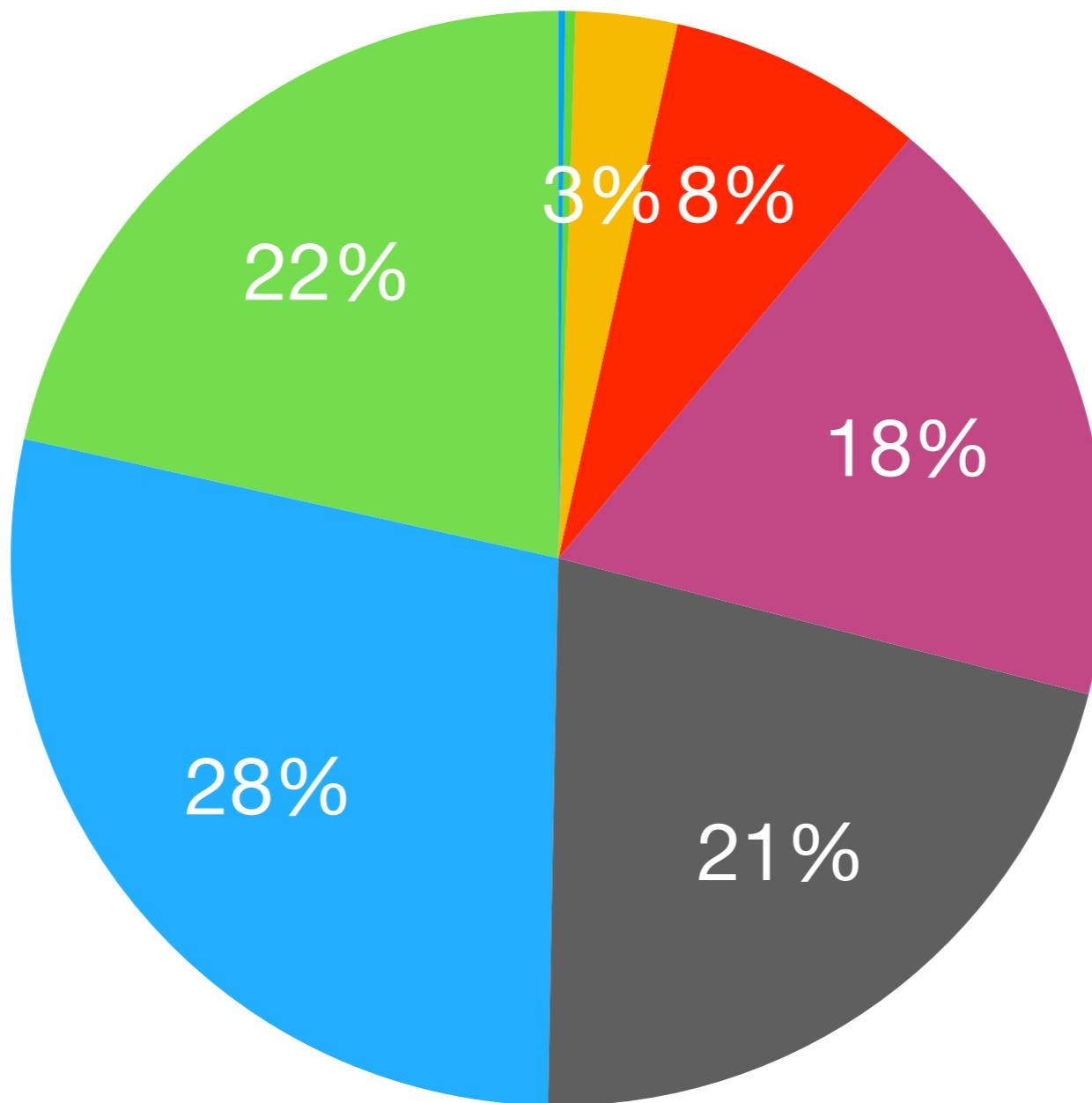
Compatibility

- Gingerbread
- Lollipop
- Ice Cream Sandwich
- Marshmallow
- Jelly Bean
- Nougat
- KitKat
- Oreo



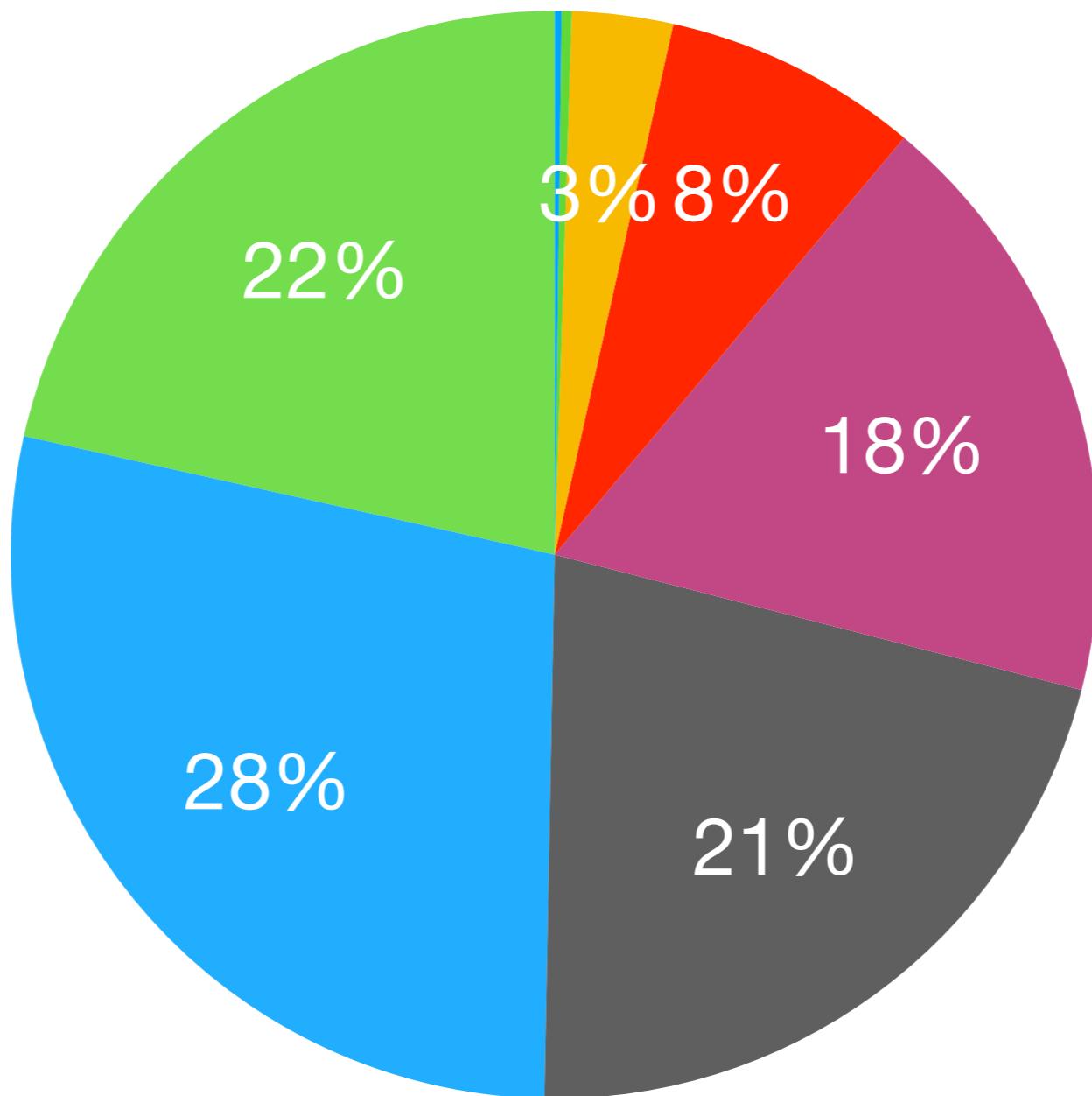
Compatibility

- Gingerbread
- Lollipop
- Ice Cream Sandwich
- Marshmallow
- Jelly Bean
- Nougat
- KitKat
- Oreo

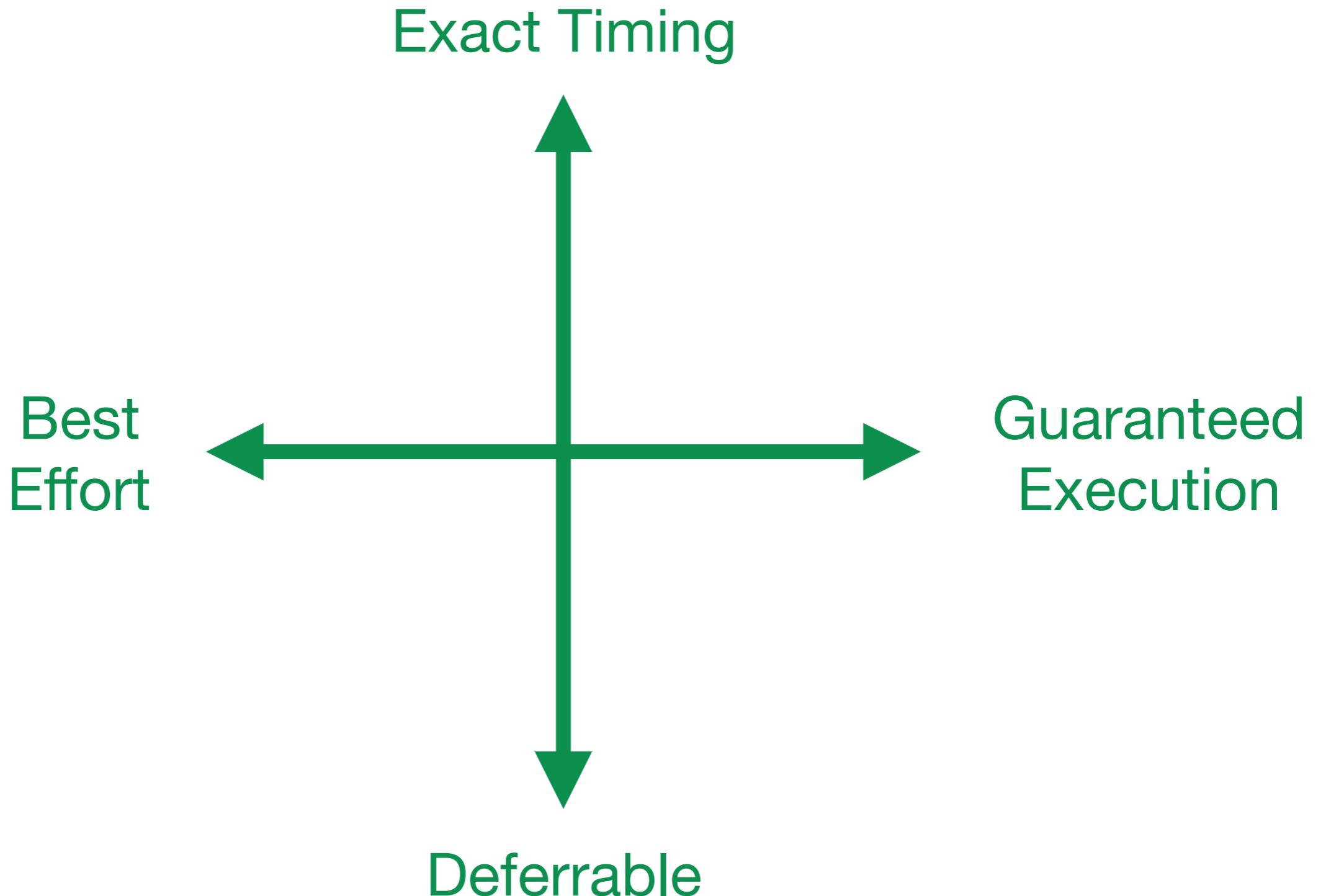


Compatibility

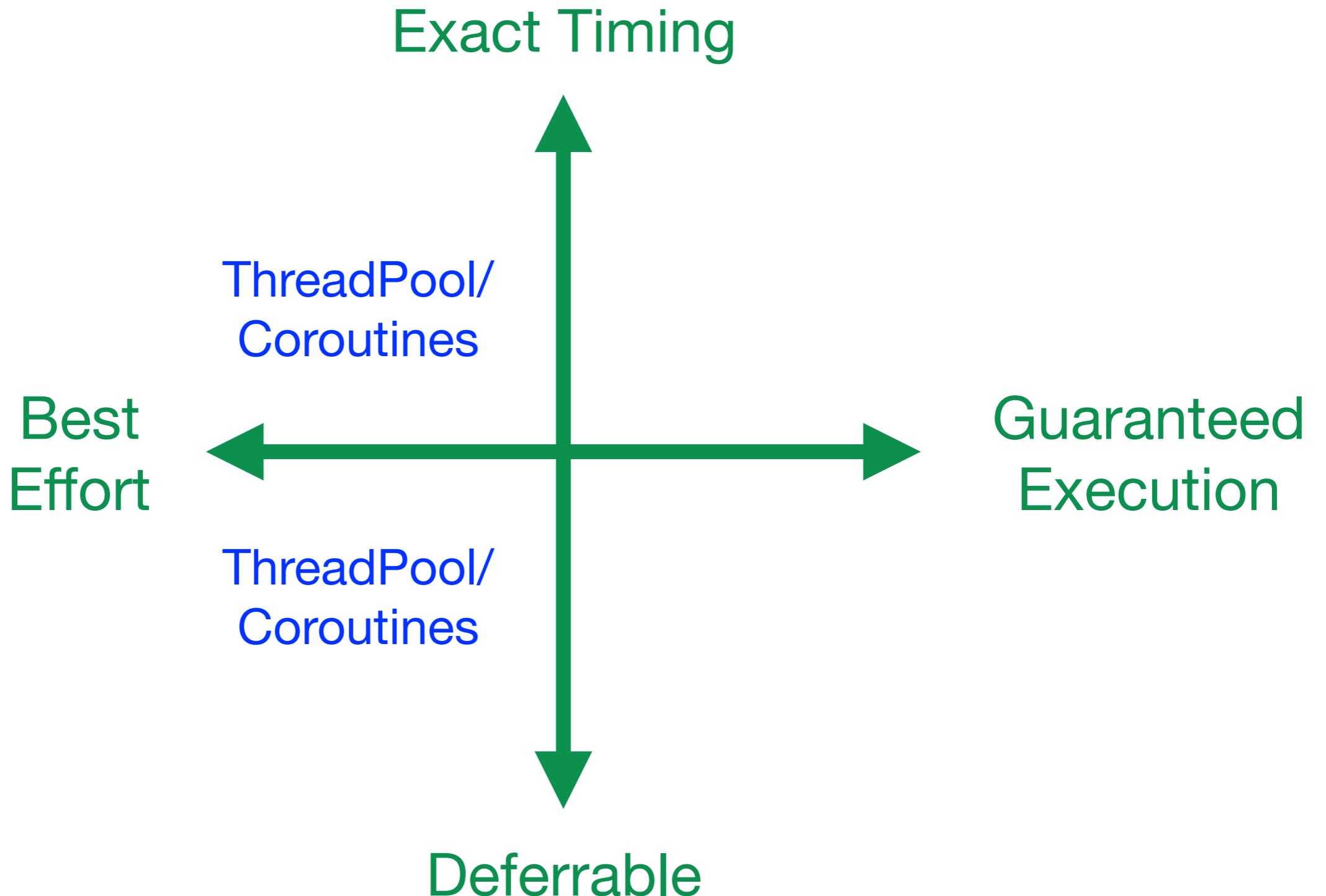
- Gingerbread
- Ice Cream Sandwich
- Jelly Bean
- KitKat
- Lollipop
- Marshmallow
- Nougat
- Oreo



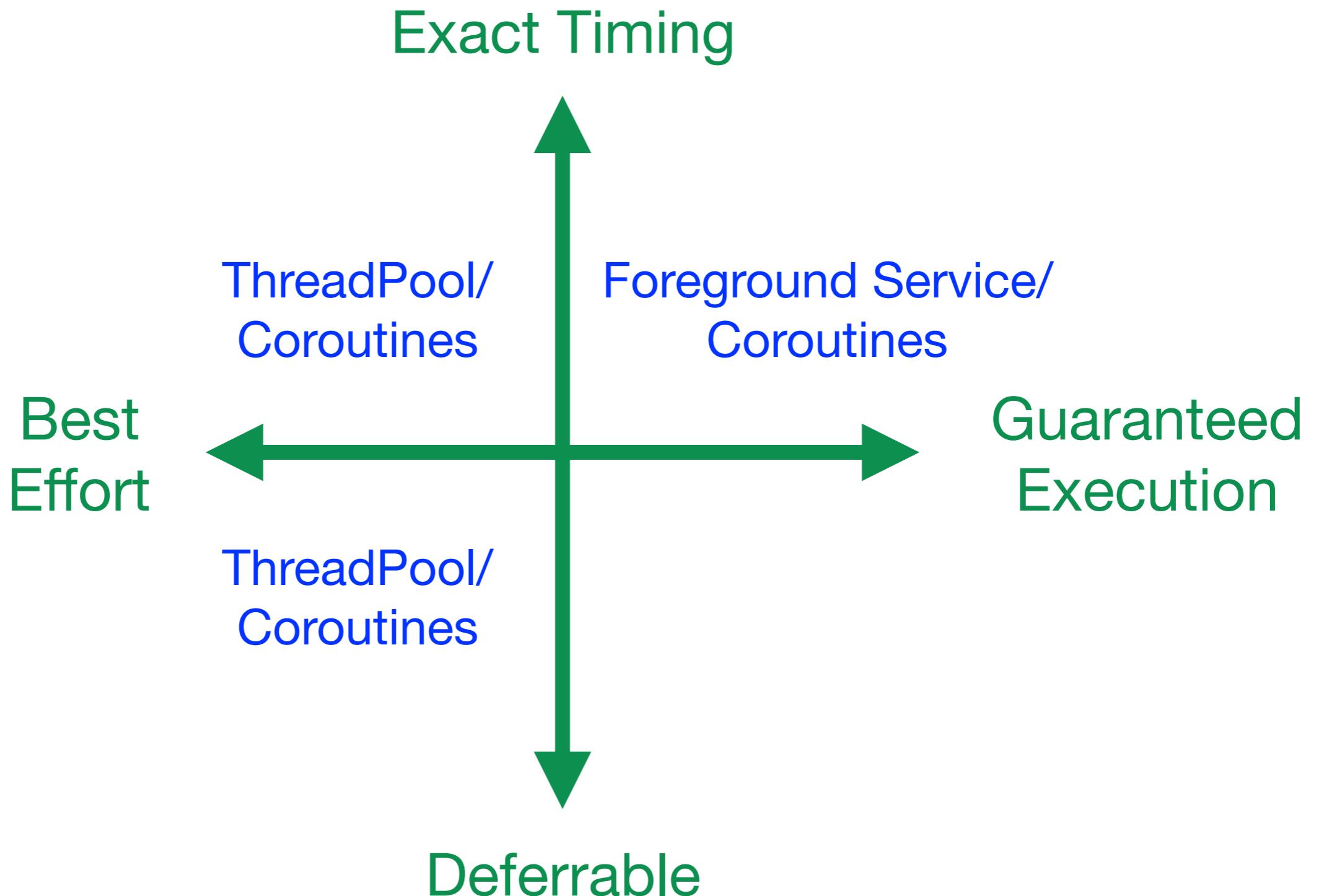
Requirements



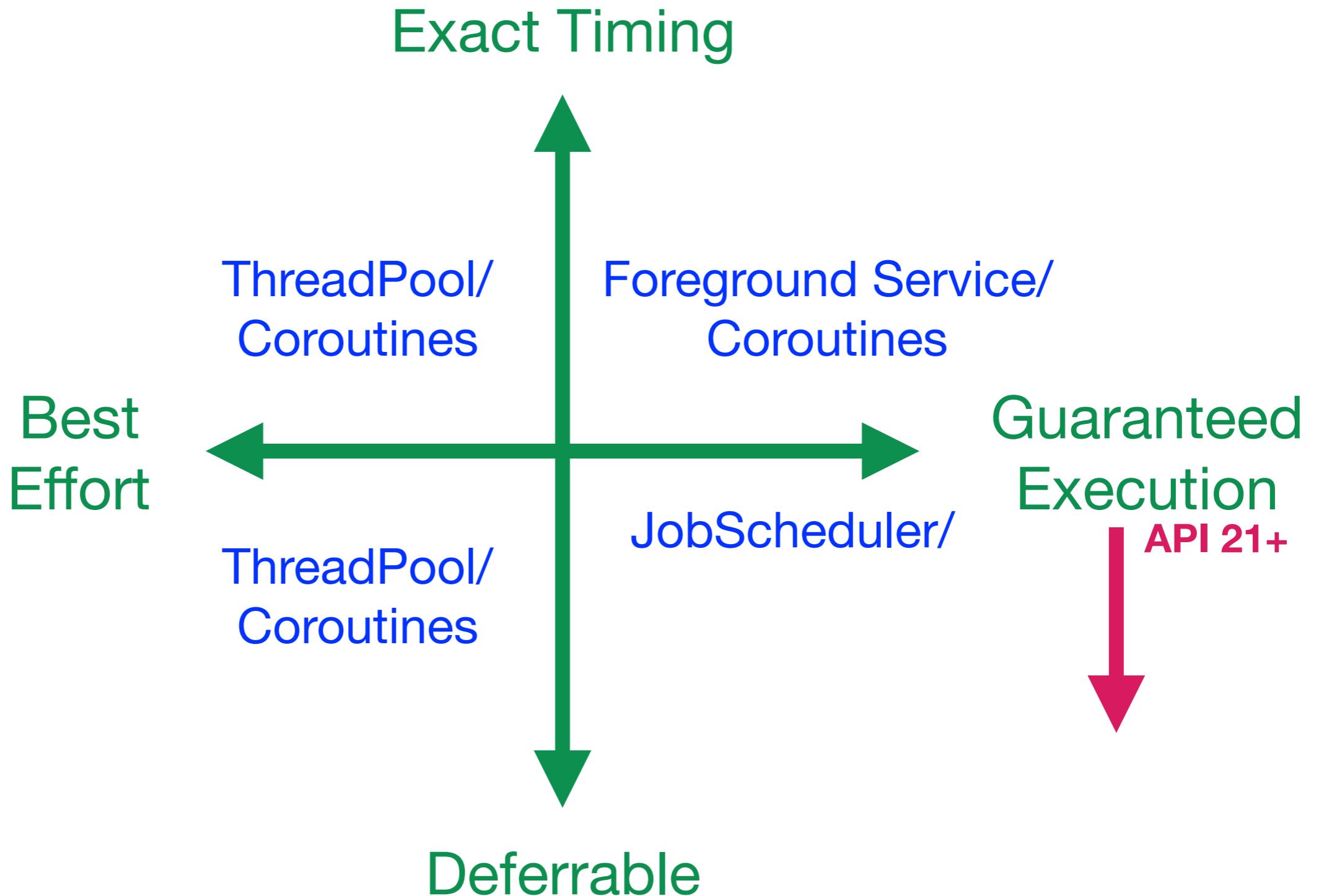
Requirements



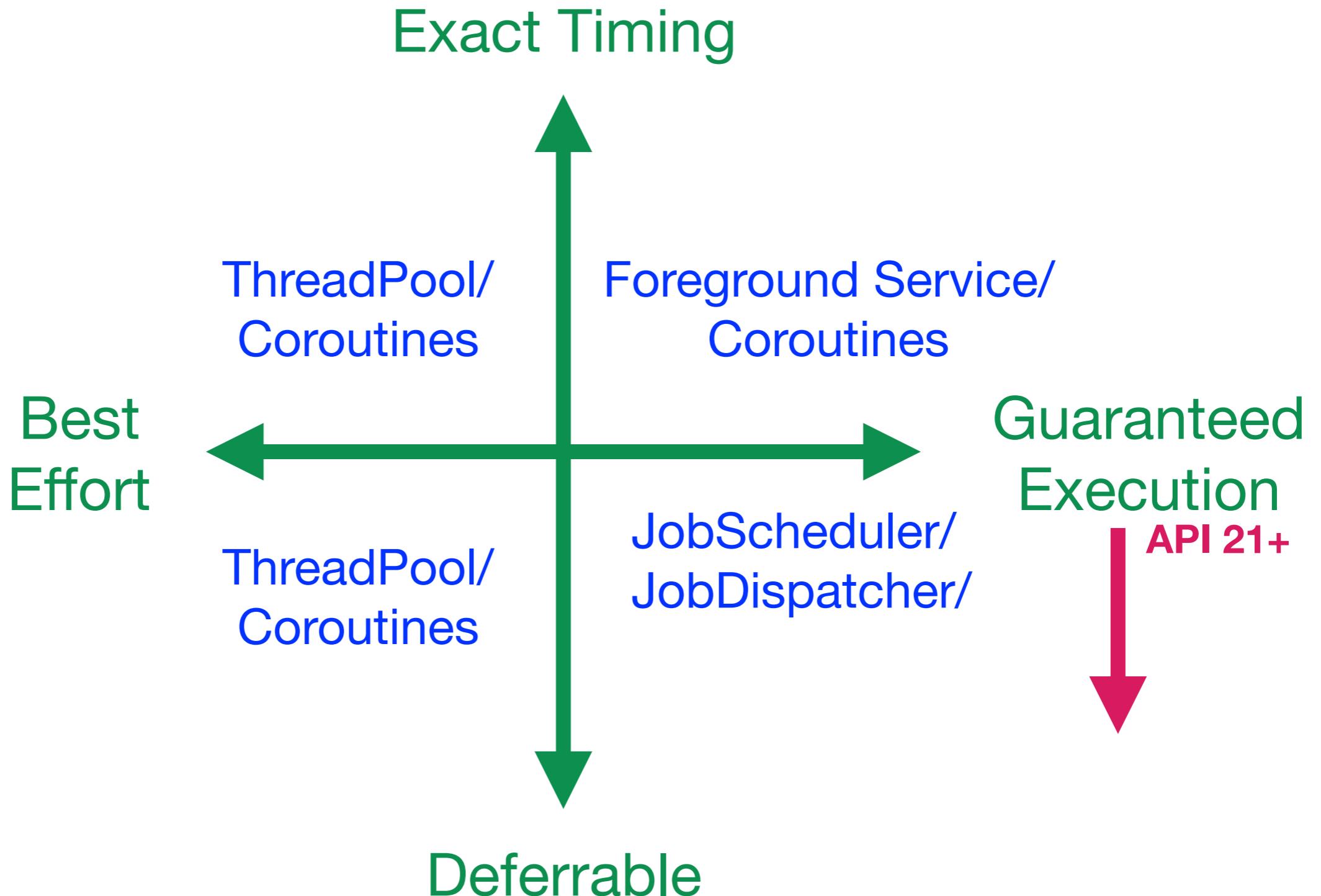
Requirements



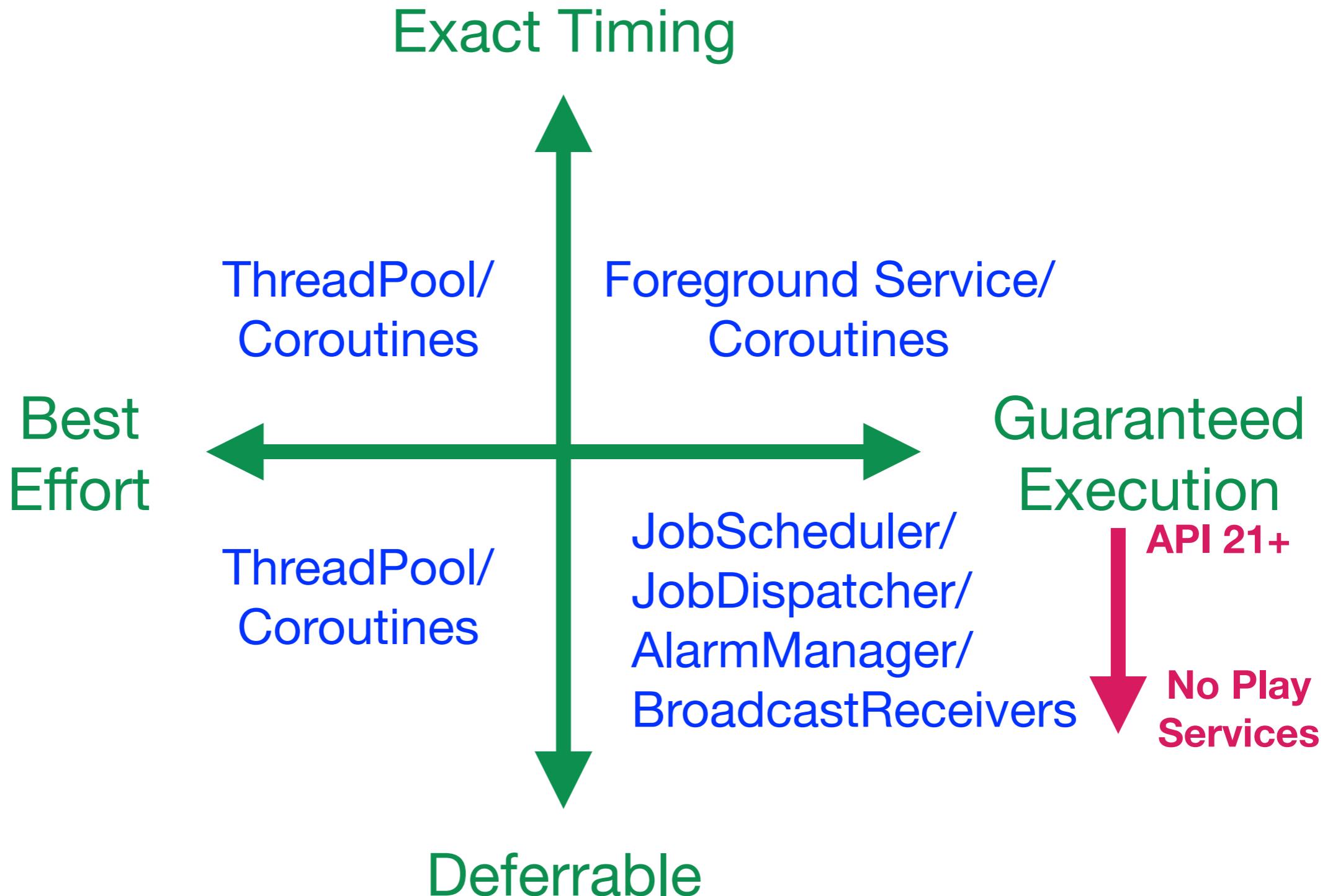
Requirements



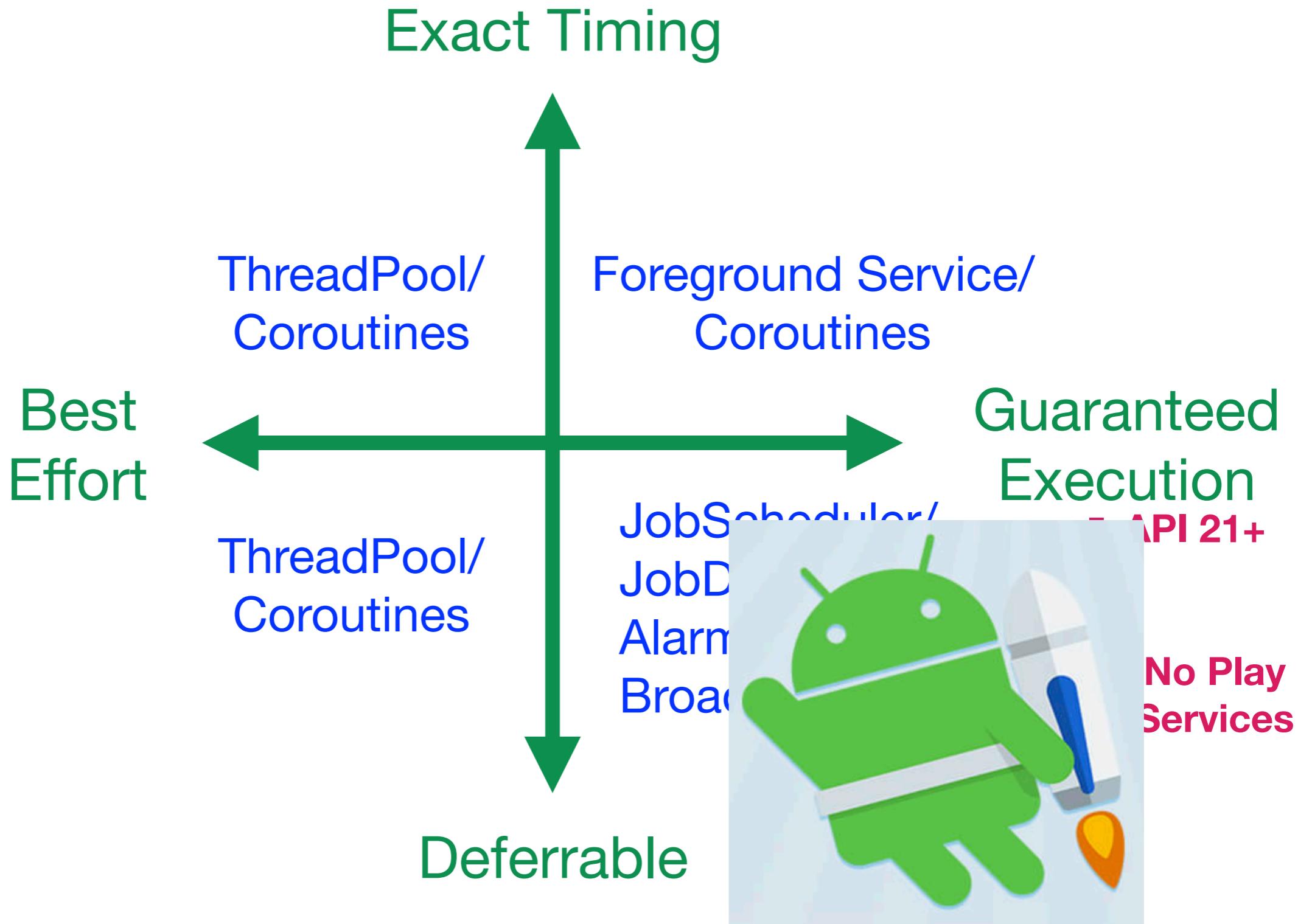
Requirements



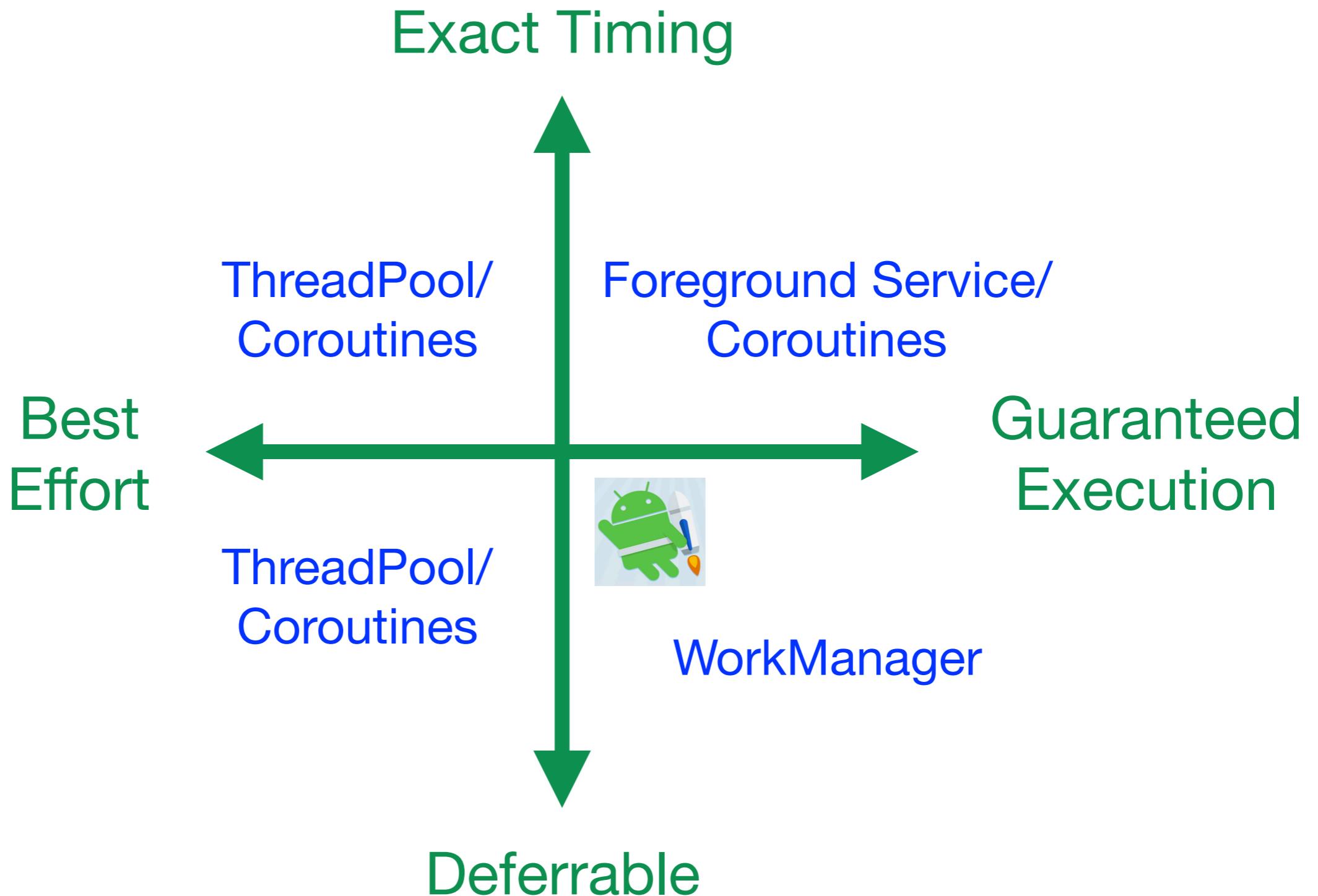
Requirements



Requirements

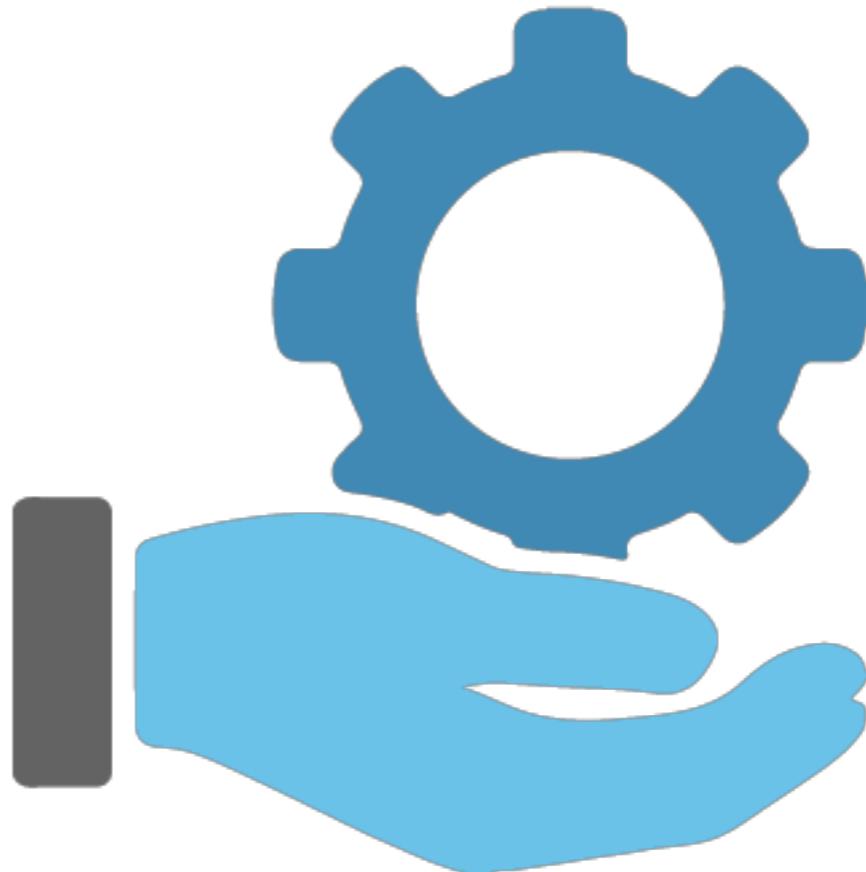


Requirements



Services

- Perform long-running operations in background.
- 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

- Perform long-running operations in background.
- 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.

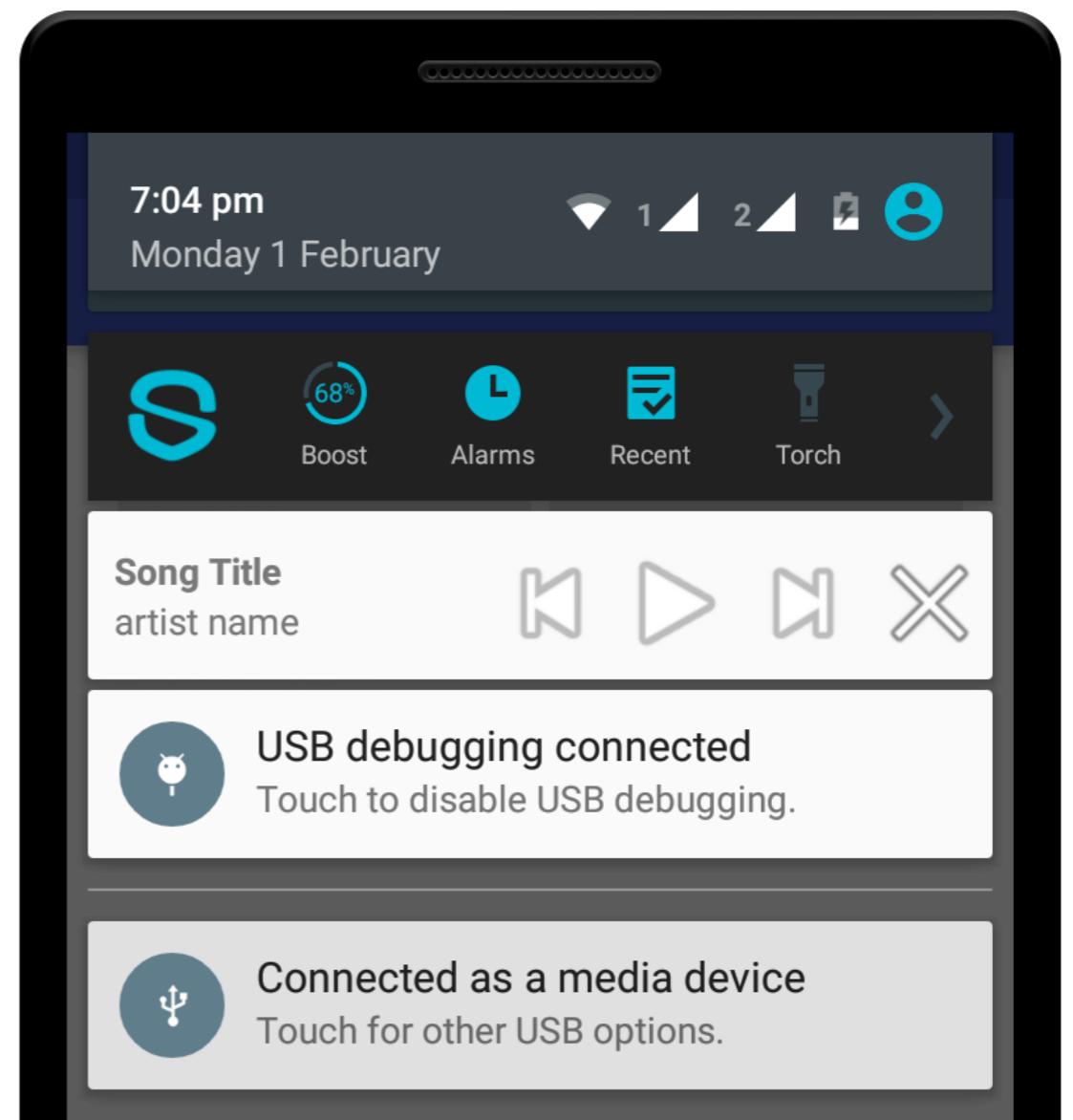


Services

Services

Performs some operation
that is noticeable to the user.

- Types:
 - Foreground

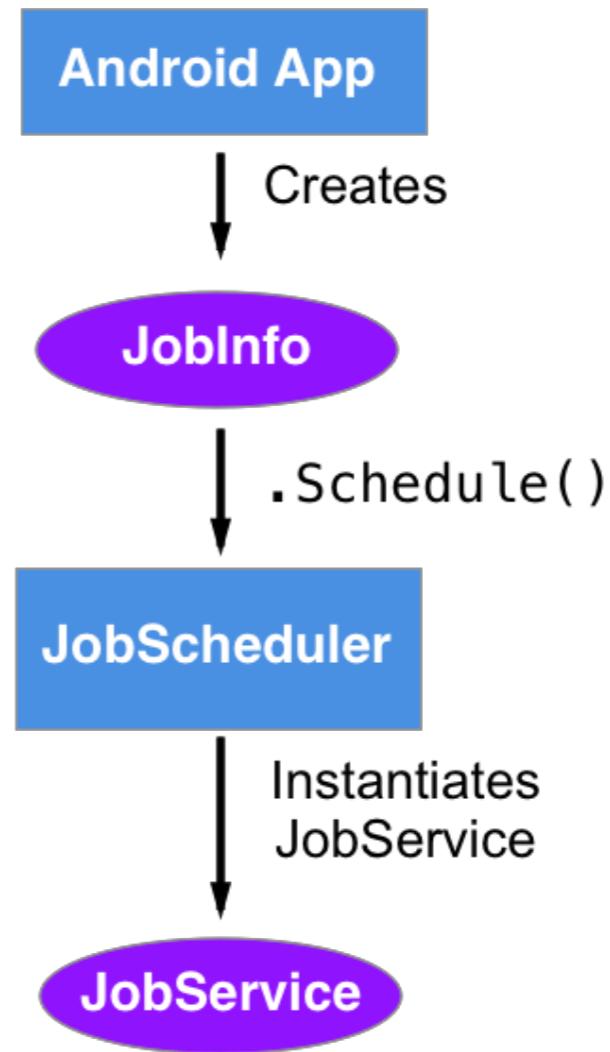


Services

Performs an operation that isn't directly noticed by the user.

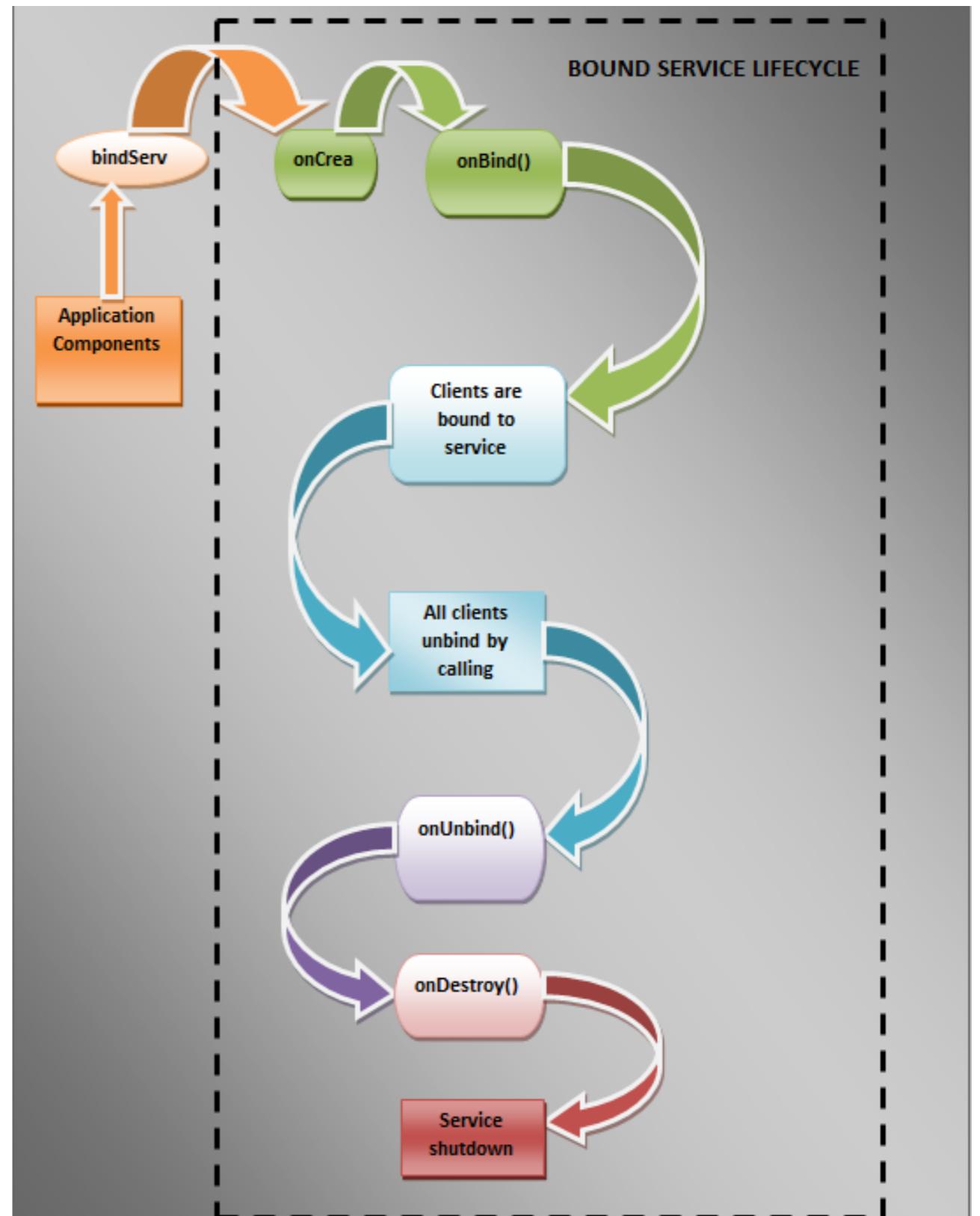
- Types:

- Foreground
- Background



Services

- Types:
 - Foreground
 - Background
 - Bound



Basics

<https://developer.android.com/guide/components/services>

Basics

- `onStartCommand()`

The system invokes this method by calling `startService()` when another component (such as an activity) requests that the service be started.

When this method executes, the service is started and can run in the background indefinitely.

If you implement this, it is your responsibility to stop the service when its work is complete by calling `stopSelf()` or `stopService()`.

If you only want to provide binding, you don't need to implement this method.

Basics

- `onStartCommand()`
- `onBind()`

The system invokes this method by calling `bindService()` when another component wants to bind with the service (such as to perform RPC).

In your implementation of this method, you must provide an interface that clients use to communicate with the service by returning an `IBinder`.

You must always implement this method; however, if you don't want to allow binding, you should return null.

Basics

- `onStartCommand()`
- `onBind()`
- `onCreate()`

The system invokes this method to perform one-time setup procedures when the service is initially created (before it calls either `onStartCommand()` or `onBind()`).

If the service is already running, this method is not called.

Basics

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

The system invokes this method when the service is no longer used and is being destroyed.

Your service should implement this to clean up any resources such as threads, registered listeners, or receivers.

This is the last call that the service receives.

Declaring a service in the manifest

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

Declaring a service in the manifest

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


Creating a Service

- Service
 - Base class for all services.
 - Create and manage a new thread on your own.

```
class HelloService : Service() {  
    private var mServiceLooper: Looper? = null  
    private var mServiceHandler: ServiceHandler? = null  
  
    // Handler that receives messages from the thread.  
    private inner class ServiceHandler(looper: Looper) : Handler(looper) {  
        override fun handleMessage(msg: Message) {  
            // Normally we would do some work here, like...  
            // For our sample, we just sleep for 5 seconds.  
            try {  
                Thread.sleep(5000)  
            } catch (e: InterruptedException) {  
                // Restore interrupt status.  
                Thread.currentThread().interrupt()  
            }  
            // Stop the service using the startId, so  
            // the service in the middle of handling a  
            stopSelf(msg.arg1)  
        }  
    }  
    override fun onCreate() {  
        // Start up the thread running the service.  
        // separate thread because the service normally  
        // runs on the main thread, which we don't want to block.  
        // (This is also why we can't use HandlerThread here.)  
        mServiceLooper = Looper.prepare()  
        mServiceHandler = ServiceHandler(mServiceLooper)  
        startService(Intent(this, HelloService::class.java))  
    }  
}
```

```
class HelloService : Service() {
    private var mServiceLooper: Looper? = null
    private var mServiceHandler: ServiceHandler? = null

    // Handler that receives messages from the thread
    private inner class ServiceHandler(looper: Looper) : Handler(looper) {
        override fun handleMessage(msg: Message) {
            // Normally we would do some work here, like download a file.
            // For our sample, we just sleep for 5 seconds.
            try {
                Thread.sleep(5000)
            } catch (e: InterruptedException) {
                // Restore interrupt status.
                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)
        }
    }

    override fun onCreate() {
        // Start up the thread running the service. Note that we create a
        // separate thread because the service normally runs in the process's
        // main thread, which we don't want to block. We also make it
        // background priority so CPU-intensive work will not disrupt our UI.
        HandlerThread("ServiceStartArguments",
            Process.THREAD_PRIORITY_BACKGROUND).apply {
            start()
            // Get the HandlerThread's Looper and use it for our Handler
            mServiceLooper = looper
            mServiceHandler = ServiceHandler(looper)
        }
    }
}
```

```
// separate thread because the service normally runs in the process's
// main thread, which we don't want to block. We also make it
// background priority so CPU-intensive work will not disrupt our UI.
HandlerThread("ServiceStartArguments",
    Process.THREAD_PRIORITY_BACKGROUND).apply {
    start()
    // Get the HandlerThread's Looper and use it for our Handler
    mServiceLooper = looper
    mServiceHandler = ServiceHandler(looper)
}
}

override fun onStartCommand(intent: Intent, flags: Int, startId: Int): Int {
    Toast.makeText(this, "service starting", Toast.LENGTH_SHORT).show()
    // For each start request, send a message to start a job and deliver the
    // start ID so we know which request we're stopping when we finish the job
    mServiceHandler?.obtainMessage()?.also { msg ->
        msg.arg1 = startId
        mServiceHandler?.sendMessage(msg)
    }
    // If we get killed, after returning from here, restart
    return START_STICKY
}

override fun onBind(intent: Intent): IBinder? {
    // We don't provide binding, so return null
    return null
}
override fun onDestroy() {
    Toast.makeText(this, "service done", Toast.LENGTH_SHORT).show()
}
```

Creating a Service

- Service
 - Base class for all services.
 - Create and manage a new thread on your own.

```
class HelloService : Service() {  
    private var mServiceLooper: Looper? = null  
    private var mServiceHandler: ServiceHandler? = null  
  
    // Handler that receives messages from the thread.  
    private inner class ServiceHandler(looper: Looper) : Handler(looper) {  
        override fun handleMessage(msg: Message) {  
            // Normally we would do some work here, like...  
            // For our sample, we just sleep for 5 seconds.  
            try {  
                Thread.sleep(5000)  
            } catch (e: InterruptedException) {  
                // Restore interrupt status.  
                Thread.currentThread().interrupt()  
            }  
            // Stop the service using the startId, so it  
            // the service in the middle of handling a  
            stopSelf(msg.arg1)  
        }  
    }  
    override fun onCreate() {  
        // Start up the thread running the service.  
        // separate thread because the service normally  
        // main thread, which we don't want to block.  
        mServiceLooper = Looper.prepare()  
        mServiceHandler = ServiceHandler(mServiceLooper)  
        HandlerThread("HelloService").start()  
        mServiceHandler?.post {  
            Looper.myLooper()?.queue {  
                mServiceHandler?.handleMessage(Message.obtain())  
            }  
        }  
    }  
    override fun onStartCommand(intent: Intent, flags: Int, startId: Int): Int {  
        // We want this service to continue running until  
        // we tell it to stop, so return sticky.  
        return START_STICKY  
    }  
}
```

Creating a Service

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

```
 /**
 * A constructor is required
 * with a name for the worker thread.
 */
class HelloIntentService :  
    IntentService("HelloIntentService") {  
    /**
     * The IntentService calls this method
     * from the default worker thread with
     * the intent that started the service.
     * When this method returns, IntentService
     * stops the service, as appropriate.
     */  
    override fun onHandleIntent(intent: Intent?) {  
        try {  
            Thread.sleep(5000)  
        } catch (e: InterruptedException) {  
            // Restore interrupt status.  
            Thread.currentThread().interrupt()  
        }  
    }  
}
```

Service Management

Service Management

- Starting a service

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

Service Management

- Starting a service
- Stopping a service

```
stopService()  
// or  
stopSelf()
```

Notify the User

Notify the User

- Toast Notifications

```
val text = "Hello toast!"  
val duration = Toast.LENGTH_SHORT  
  
val toast = Toast.makeText(applicationContext, text, duration)  
toast.show()
```

Notify the User

- Toast Notifications
- Status Bar Notifications

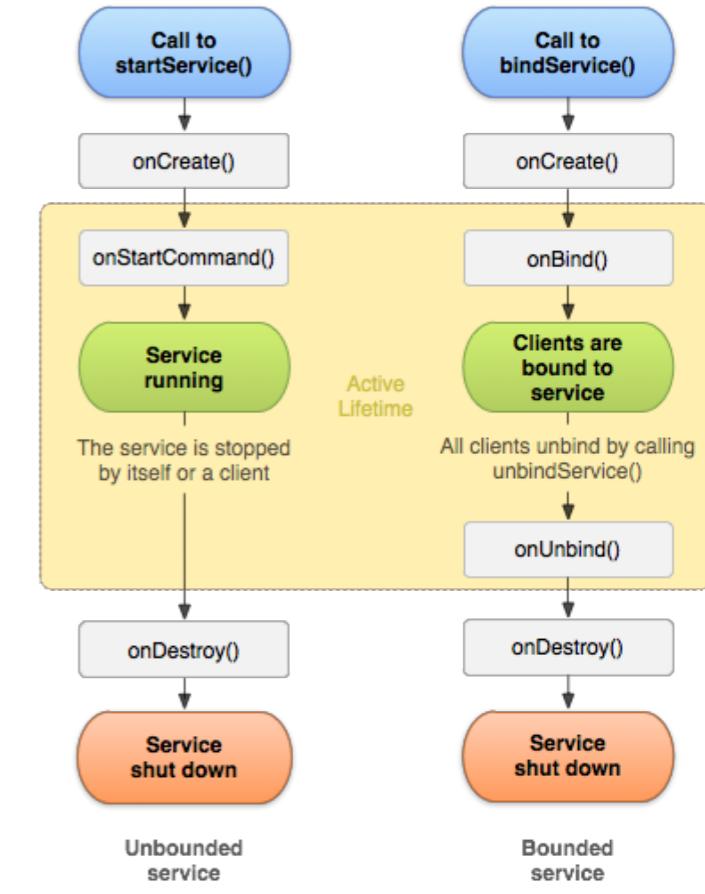
```
val pendingIntent: PendingIntent =
    Intent(this, ExampleActivity::class.java).let { notificationIntent ->
        PendingIntent.getActivity(this, 0, notificationIntent, 0)
    }

val notification: Notification = Notification.Builder(this,
    CHANNEL_DEFAULT_IMPORTANCE)
    .setContentTitle(getText(R.string.notification_title))
    .setContentText(getText(R.string.notification_message))
    .setSmallIcon(R.drawable.icon)
    .setContentIntent(pendingIntent)
    .setTicker(getText(R.string.ticker_text))
    .build()

startForeground(ONGOING_NOTIFICATION_ID, notification)
```

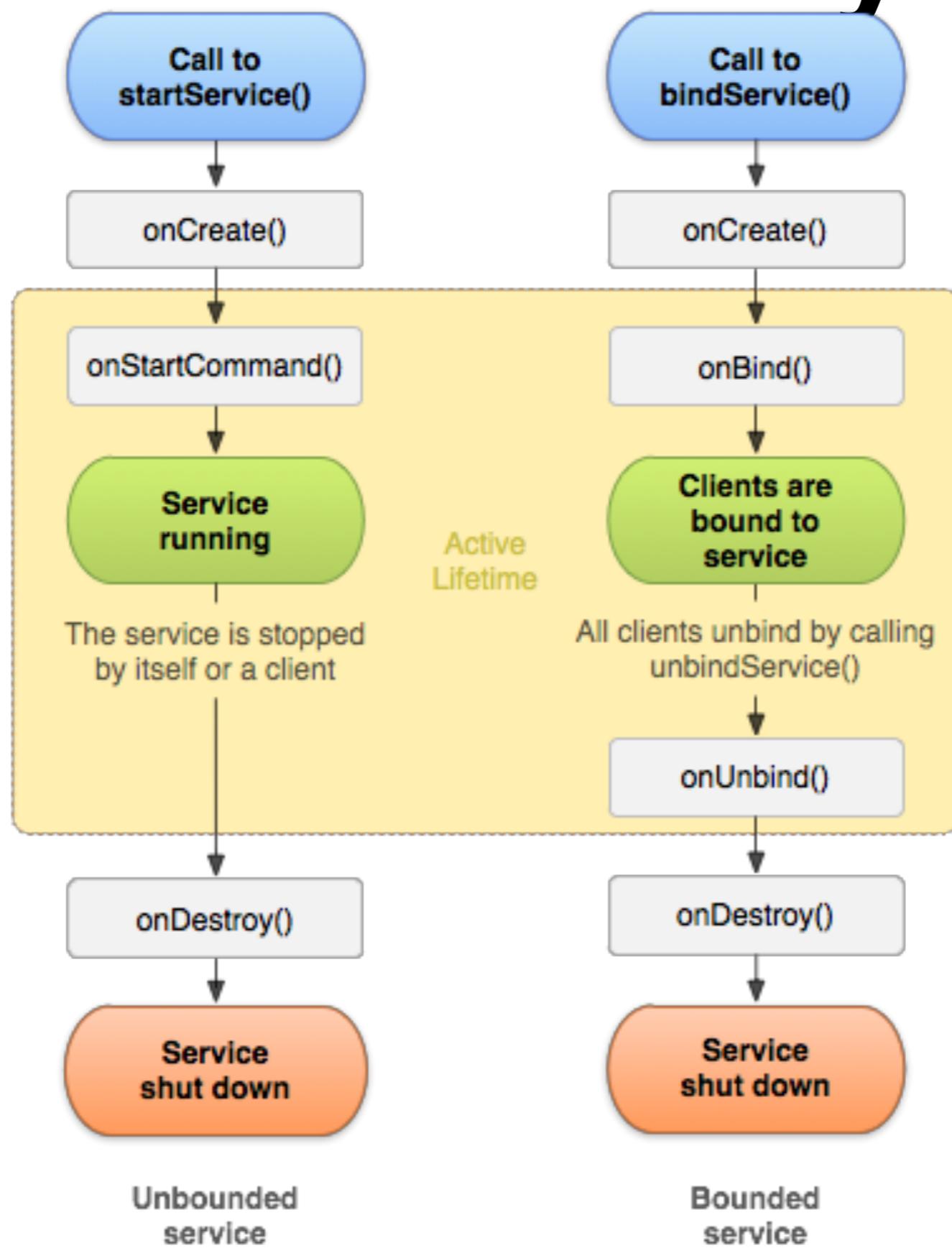
Service Lifecycle

```
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  
    }  
}
```



DEMO

Service Lifecycle



Alarm Manager

- Alarm types:
 - `ELAPSED_REALTIME`
 - `ELAPSED_REALTIME_WAKEUP`
 - `RTC`
 - `RTC_WAKEUP`

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  
)
```

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)
```

JobScheduler

- Register the Service

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

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)
```

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))
```

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))
```

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)
```

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(requirementsIdleCheckbox.isChecked)  
    setRequiresCharging(requirementsChargingCheckBox.isChecked)  
    setExtras(extras)  
}
```

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(requirementsIdleCheckbox.isChecked)
    setRequiresCharging(requirementsChargingCheckBox.isChecked)
    setExtras(extras)
}
(getSystemService(Context.JOB_SCHEDULER_SERVICE) as
JobScheduler).schedule(builder.build())
```

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(requirementsIdleCheckbox.isChecked)  
    setRequiresCharging(requirementsChargingCheckBox.isChecked)  
    setExtras(extras)  
}  
  
(getSystemService(Context.JOB_SCHEDULER_SERVICE) as  
JobScheduler).schedule(builder.build())
```

The Job



Define The Job

```
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  
    }  
}
```

WorkManager



WorkManager

- Core Classes:
 - Worker

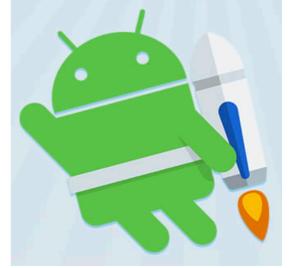




WorkManager

- Core Classes:
 - Worker

```
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
        // Returning:
        // - RETRY tells WorkManager to try this task again later
        // - FAILURE says not to try again.
    }
}
```

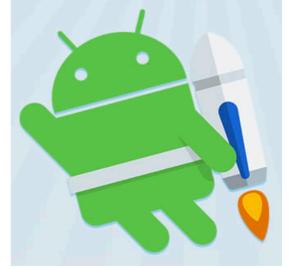


WorkManager

- Core Classes:
 - Worker

```
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
        // Returning:
        // - RETRY tells WorkManager to try this task again later
        // - FAILURE says not to try again.
    }
}
```

Runs on a background thread



WorkManager

- Core Classes:
 - Worker
 - WorkRequest
 - OneTimeWorkRequest
 - PeriodicTimeWorkRequest

```
val compressionWork = OneTimeWorkRequestBuilder<CompressWorker>().build()  
WorkManager.getInstance().enqueue(compressionWork)
```

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()
```

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  
.then(workB) // FYI, then() returns a new WorkContinuation instance  
.then(workC)  
.enqueue()
```

Chained Tasks

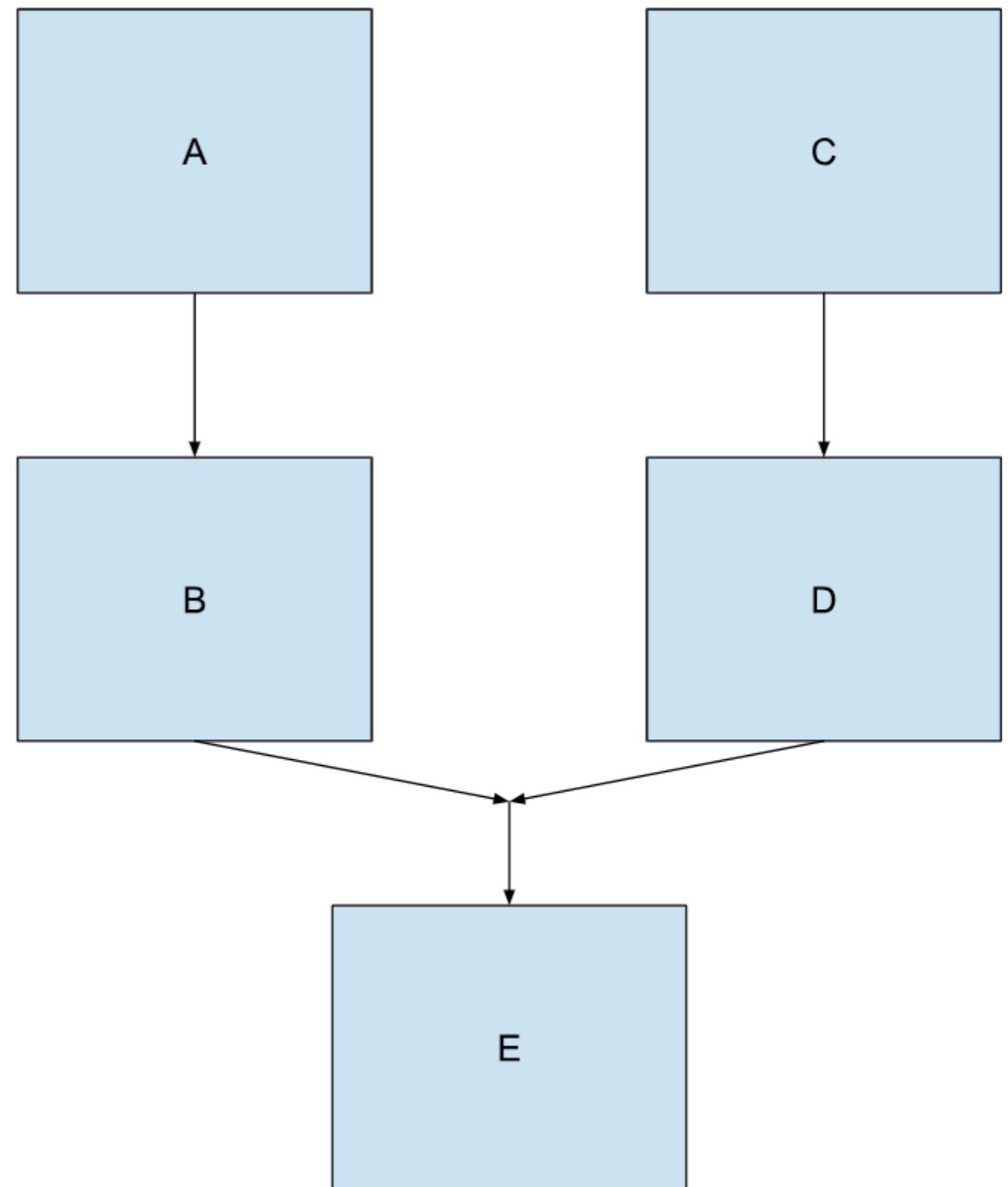
```
WorkManager.getInstance()
.beginWith(workA)
    // Note: WorkManager.beginWith() returns a
    // WorkContinuation object; the following calls are
    // to WorkContinuation methods
.then(workB) // FYI, then() returns a new WorkContinuation instance
.then(workC)
.enqueue()
```

```
WorkManager.getInstance()
    // First, run all the A tasks (in parallel):
.beginWith(workA1, workA2, workA3)
    // ...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()
```

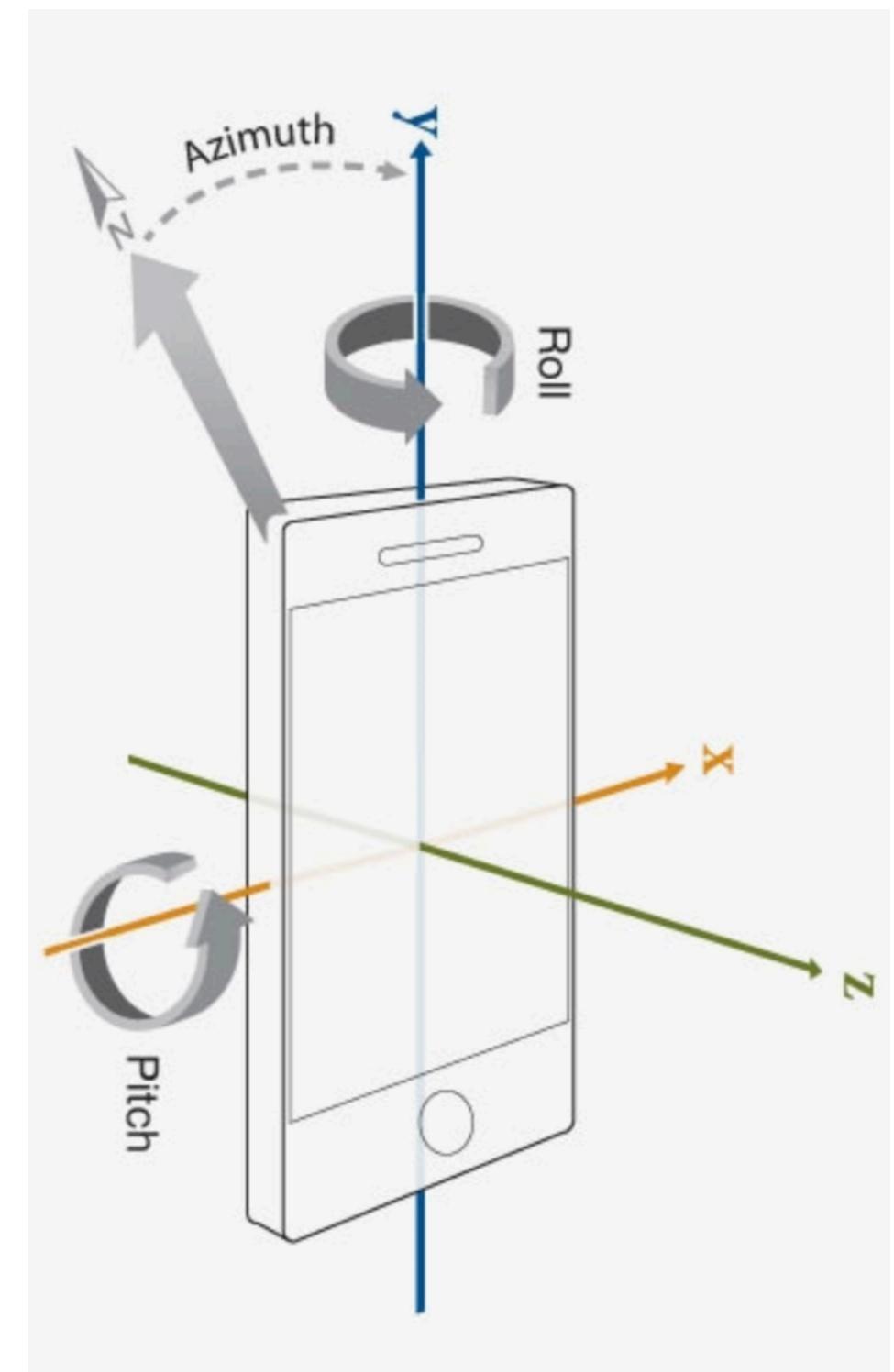
DEMO

Chained tasks

```
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



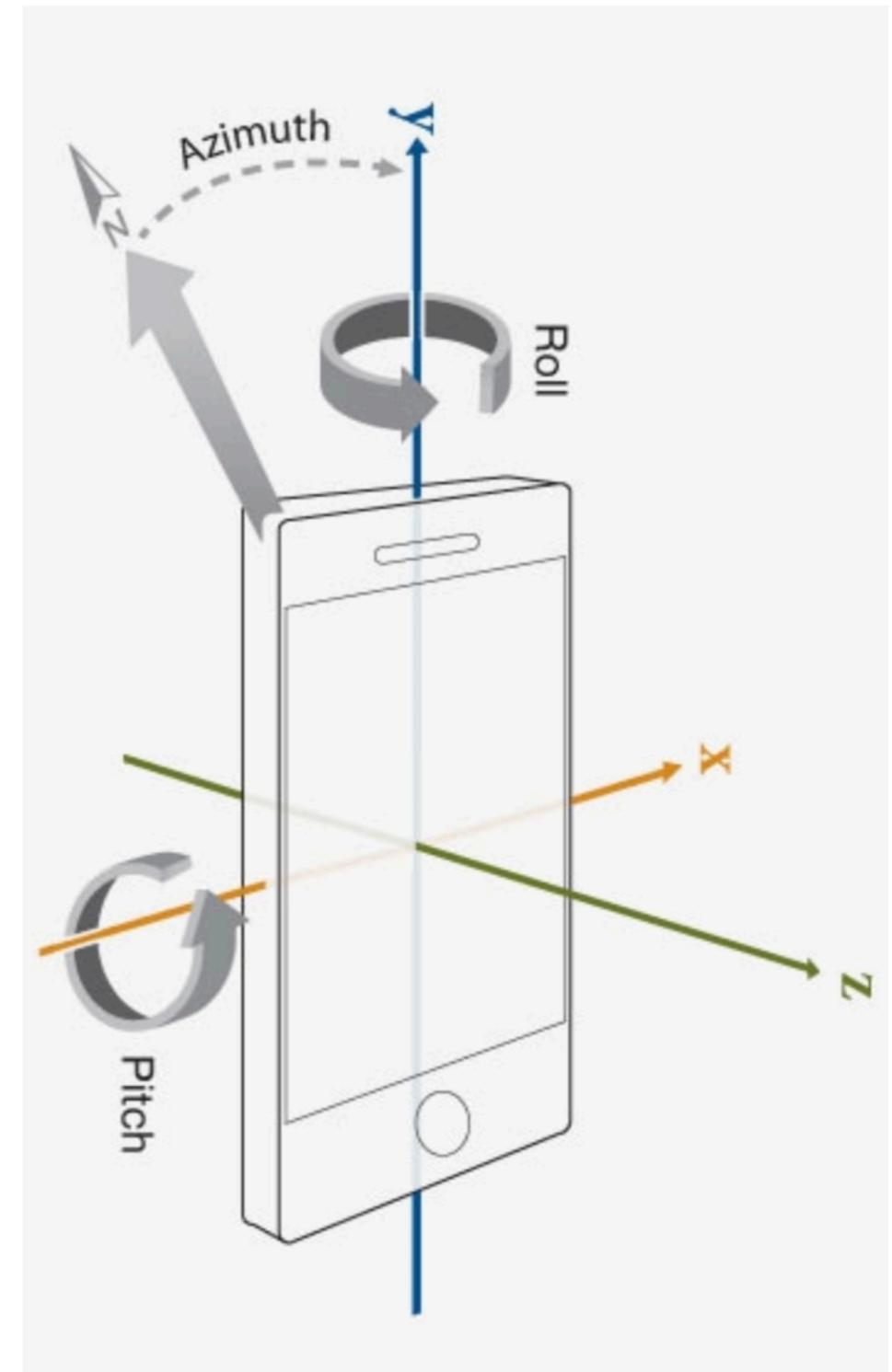
Sensors

- Motion

These sensors measure acceleration forces and rotational forces along three axes.

This category includes:

- Accelerometers.
- Gravity sensors.
- Gyroscopes.
- Rotational vector sensors.



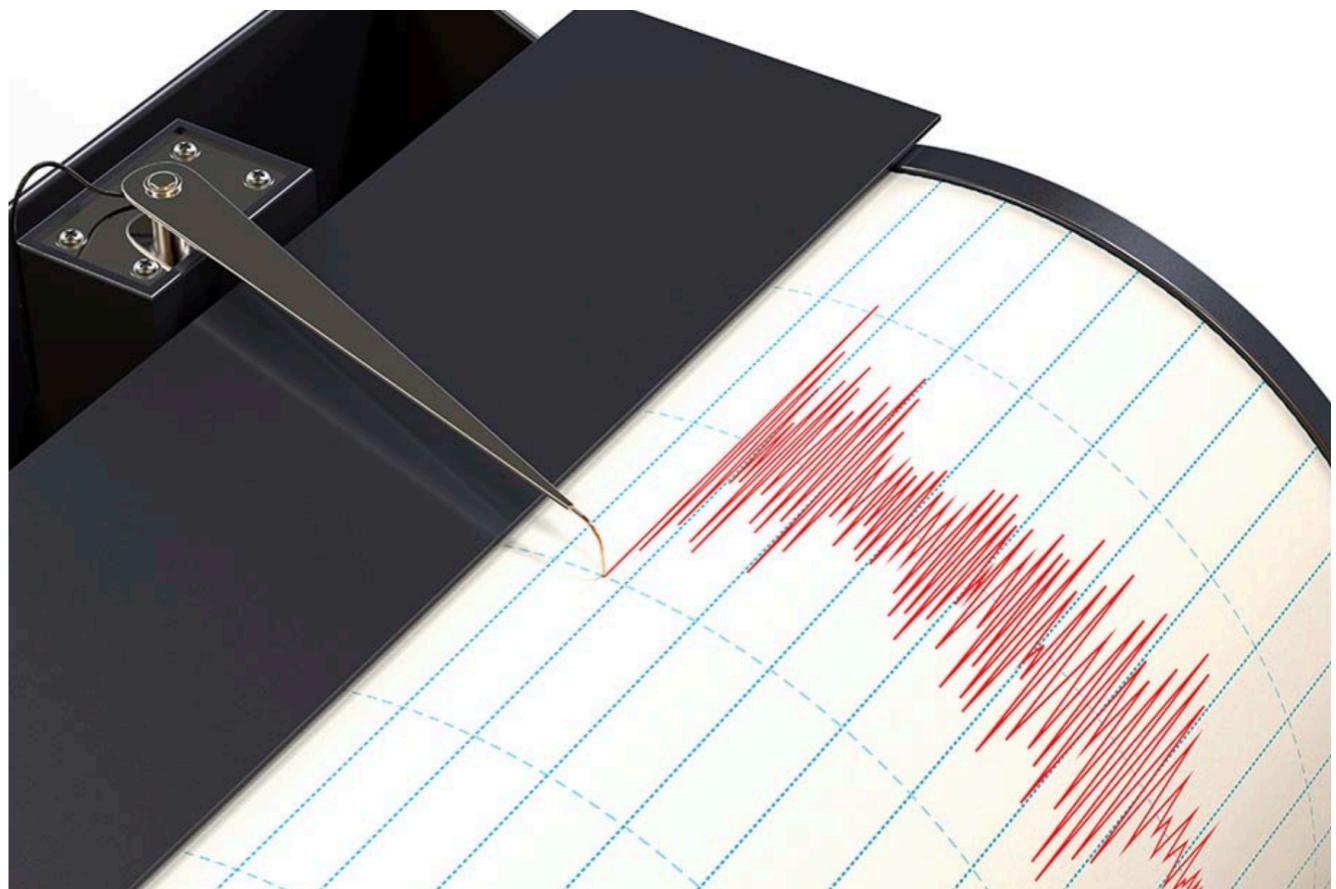
Sensors

- Motion
- Position

These sensors measure the physical position of a device.

This category includes:

- Orientation sensors.
- Magnetometers.



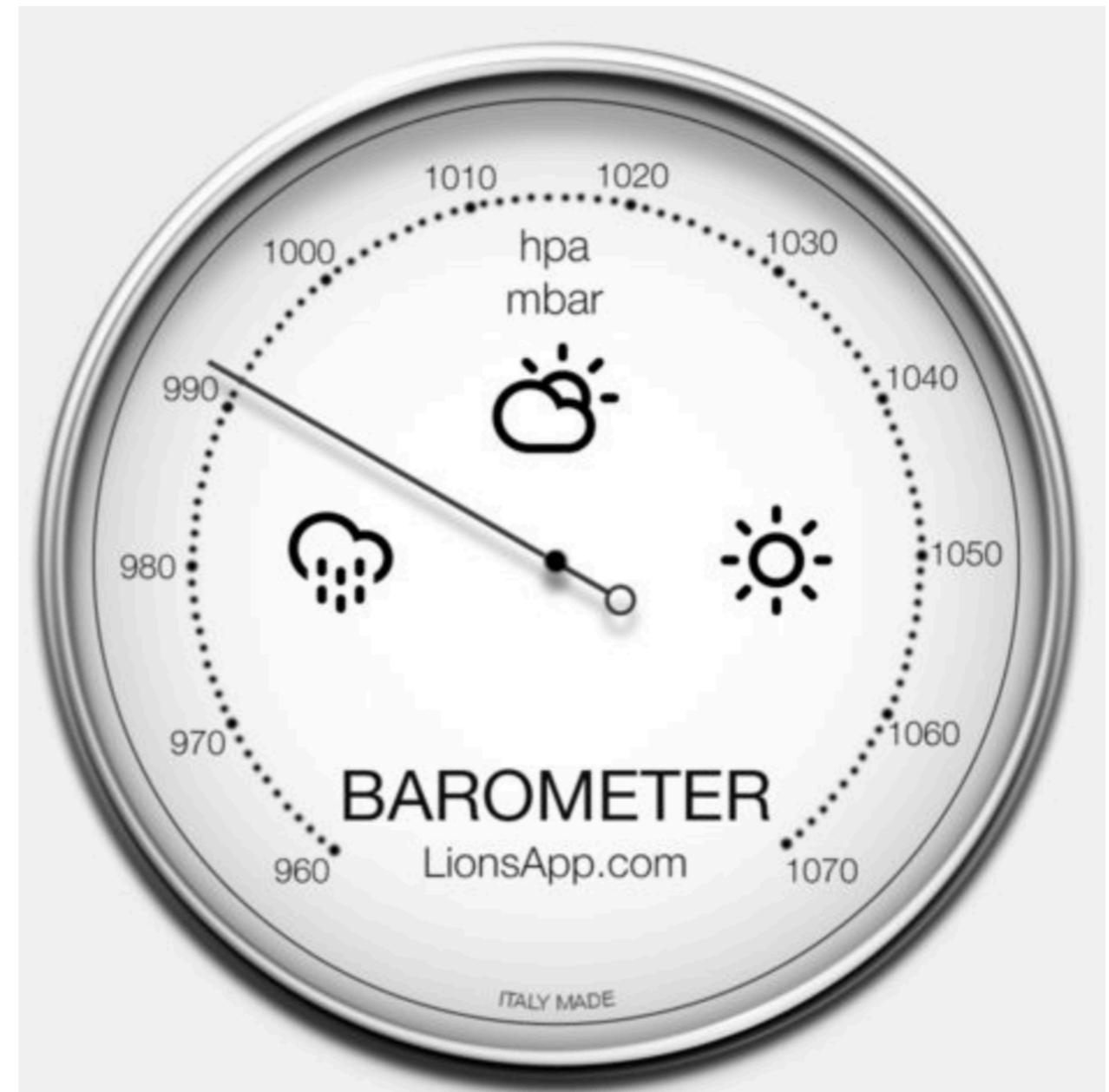
Sensors

- Motion
- Position
- Environment

These sensors measure various environmental parameters.

This category includes:

- Barometers.
- Photometers.
- Thermometers.



Framework

```
private lateinit var mSensorManager: SensorManager  
...  
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
```

Framework

- SensorManager

```
private lateinit var mSensorManager: SensorManager  
...  
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
```

Framework

- SensorManager

```
private lateinit var mSensorManager: SensorManager
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager

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

Framework

- SensorManager
- Sensor

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

Framework

- SensorManager
- Sensor

```
if (mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY) != null) {  
    val gravSensors: List<Sensor> =  
        mSensorManager.getSensorList(Sensor.TYPE_GRAVITY)  
    // Use the version 3 gravity sensor.  
    mSensor = gravSensors.firstOrNull {  
        it.vendor.contains("Google LLC") && it.version == 3  
    }  
}  
if (mSensor == null) {  
    // Use the accelerometer.  
    mSensor = if (mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)  
        != null) {  
        mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)  
    } else {  
        // Sorry, there are no accelerometers on your device.  
        null  
    }  
}
```

Framework

- SensorManager
- Sensor
- SensorEvent
- SensorEventListener

```
class SensorActivity : Activity(), SensorEventListener {  
    private lateinit var mSensorManager: SensorManager  
    private var mLight: Sensor? = null  
  
    public override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.main)  
  
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager  
        mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT)  
    }  
  
    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {  
        // ...  
    }  
}
```

```
public override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.main)

    mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
    mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT)
}

override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
    // Do something here if sensor accuracy changes.
}

override fun onSensorChanged(event: SensorEvent) {
    // The light sensor returns a single value.
    // Many sensors return 3 values, one for each axis.
    val lux = event.values[0]
    // Do something with this sensor value.
}

override fun onResume() {
    super.onResume()
    mLight?.also { light ->
        mSensorManager.registerListener(this, light,
            SensorManager.SENSOR_DELAY_NORMAL)
    }
}

override fun onPause() {
    super.onPause()
    mSensorManager.unregisterListener(this)
}
```

Motion

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

	SensorEvent. values[1]	Measured acceleration along the Y axis without any bias compensation.	
	SensorEvent. values[2]	Measured acceleration along the Z axis without any bias compensation.	
	SensorEvent. values[3]	Measured acceleration along the X axis with estimated bias compensation.	
	SensorEvent. values[4]	Measured acceleration along the Y axis with estimated bias compensation.	
	SensorEvent. values[5]	Measured acceleration along the Z axis with estimated bias compensation.	
TYPE_GRAVITY	SensorEvent. values[0]	Force of gravity along the x axis.	m/s^2
	SensorEvent. values[1]	Force of gravity along the y axis.	
	SensorEvent. values[2]	Force of gravity along the z axis.	
TYPE_GYROSCOPE	SensorEvent. values[0]	Rate of rotation around the x axis.	rad/s
	SensorEvent. values[1]	Rate of rotation around the y axis.	
	SensorEvent. values[2]	Rate of rotation around the z axis.	

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.	µT
	<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	µT

	<code>SensorEvent. values[0]</code>	axis.	
	<code>SensorEvent. values[1]</code>	Geomagnetic field strength along the y axis.	
	<code>SensorEvent. values[2]</code>	Geomagnetic field strength along the z axis.	
<code>TYPE_MAGNETIC_FIELD_UNCALIBRATED</code>	<code>SensorEvent. values[0]</code>	Geomagnetic field strength (without hard iron calibration) along the x axis.	µT
	<code>SensorEvent. values[1]</code>	Geomagnetic field strength (without hard iron calibration) along the y axis.	
	<code>SensorEvent. values[2]</code>	Geomagnetic field strength (without hard iron calibration) along the z axis.	
	<code>SensorEvent. values[3]</code>	Iron bias estimation along the x axis.	
	<code>SensorEvent. values[4]</code>	Iron bias estimation along the y axis.	
	<code>SensorEvent. values[5]</code>	Iron bias estimation along the z axis.	
<code>TYPE_ORIENTATION</code> ¹	<code>SensorEvent. values[0]</code>	Azimuth (angle around the z-axis).	Degrees
	<code>SensorEvent. values[1]</code>	Pitch (angle around the x-axis).	
	<code>SensorEvent. values[2]</code>	Roll (angle around the y-axis).	
<code>TYPE_PROXIMITY</code>	<code>SensorEvent. values[0]</code>	Distance from object. ²	cm

DEMO

Environment

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. ¹

★ ¹ 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.

