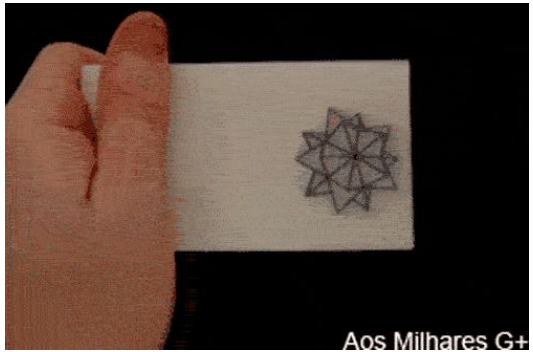
Multithreading Alternatives in Android

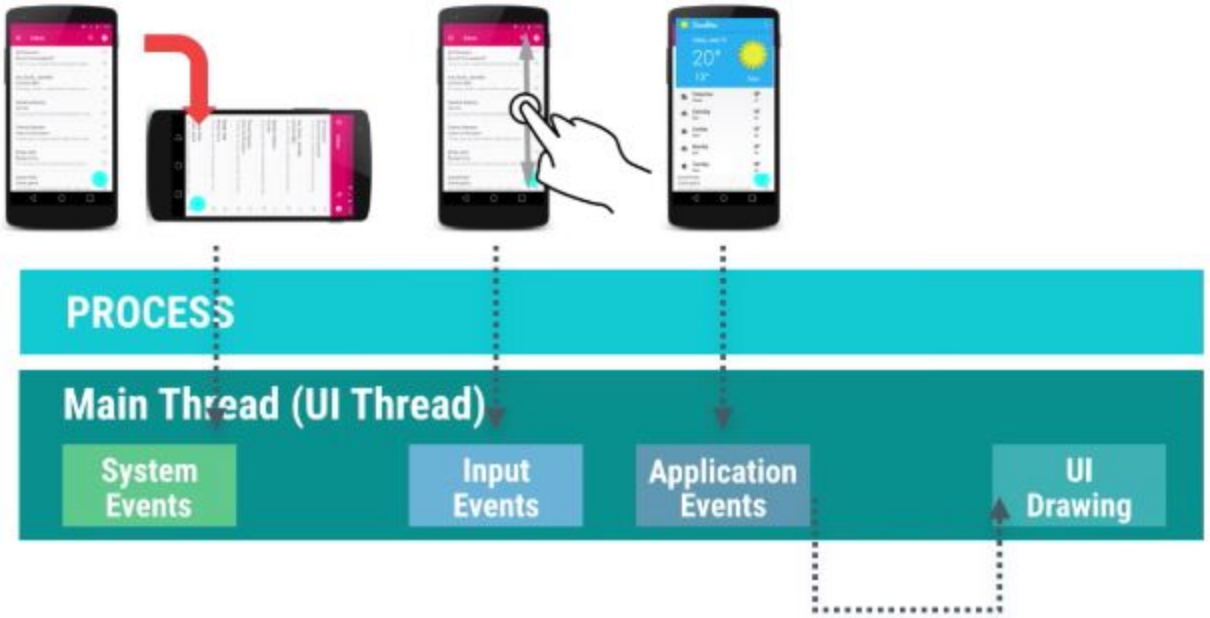
By Shiv Kumar Malik

What is this 16ms time limit

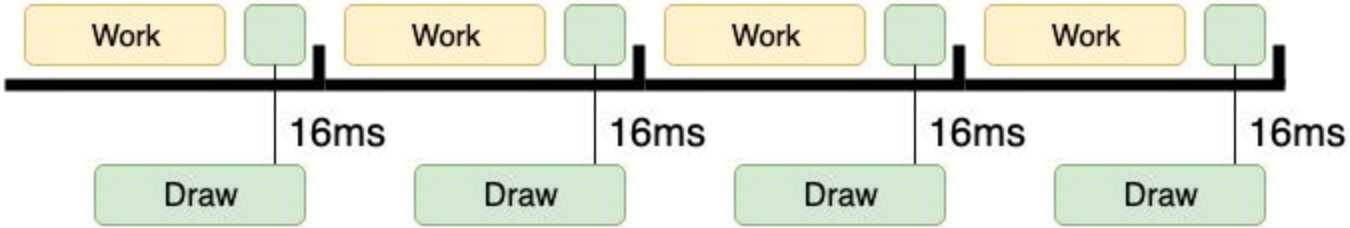
1000ms / 60 frames =

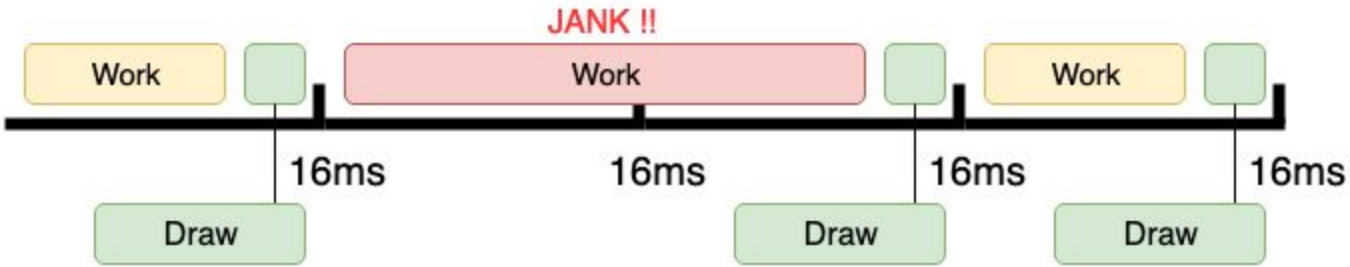
16.666 ms / frame

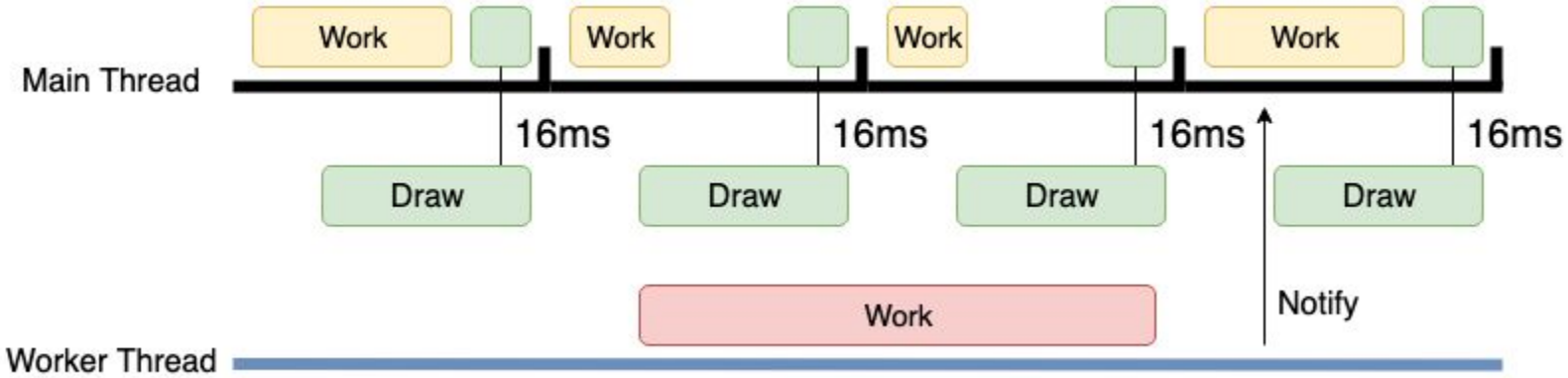



PROCESS
Main Thread (UI Thread)
System Events
Input Events
Application Events
UI Drawing


PROCESS
Main Thread (UI Thread)
System Events
Input Events
Application Events
Service
Alarm
UI Drawing


Work
Work
Work
Work
16ms
16ms
16ms
16ms
Draw
Draw
Draw
Draw


JANK !!
Work
Work
Work
16ms
16ms
16ms
16ms
Draw
Draw
Draw


Work
Work
Work
Work
Main Thread
16ms
16ms
16ms
16ms
Draw
Draw
Draw
Draw
Work
Notify
Worker Thread

Use Cases

- Sharedpref/file read/write
- Serialization/deserialization
- Data manipulation for display
- Encryption/decryption of data
- Bitmap manipulation
- DB operations
- Heavy data calculations
- Access to services like location/telephony
- Library initialisations
- Logging: GA, inhouse logging

Use Cases

- Sharedpref/file read/write
- Serialization/deserialization
- Data manipulation for display
- Encryption/decryption of data
- Bitmap manipulation
- DB operations
- Heavy data calculations
- Access to services like location/telephony
- Library initialisations
- Logging: GA, inhouse logging
- ……...

Use Cases

- Sharedpref/file read/write
- Serialization/deserialization
- Data manipulation for display
- Encryption/decryption of data
- Bitmap manipulation
- DB operations
- Heavy data calculations
- Access to services like location/telephony
- Library initialisations
- Logging: GA, inhouse logging


MS. PEREZ
WHAT CAN I DO?

Multi-Threading Alternatives

- Simple Threads
- AsyncTask
- IntentService
- Handlers
- ThreadPoolExecutor
- ...

Simple Threads

```
new Thread() {
    @Override
    public void run() {
        // Do heavy work here
        activity.onTaskComplete(taskId);
    }
}.start();
```

Simple Threads

+ Simple to implement
+ Versatile
+ Communicating back to main thread is easy

Simple Threads

+ Simple to implement
+ Versatile
+ Communicating back to main thread is easy
- Become too many
- Memory heavy
- Makes debugging difficult

Multi-Threading Alternatives

- Simple Threads
- **AsyncTask**
- IntentService
- Handlers
- ThreadPoolExecutor

AsyncTask

```
1  private static class MyAsyncTask extends AsyncTask<String, Float, Bitmap> {
2
3      protected Bitmap doInBackground(String... params) {
4          // Bg thread. Do heavy lifting here.
5          // publishProgress(progress)
6          return result;
7      }
8
9      protected void onPostExecute(Bitmap result) {
10         // on main thread
11     }
12
13     protected void onProgressUpdate(Float... values) {
14         // on main thread
15     }
16 }
17
18 MyAsyncTask asyncTask = new MyAsyncTask();
19 asyncTask.execute(IMAGE_URL);
```

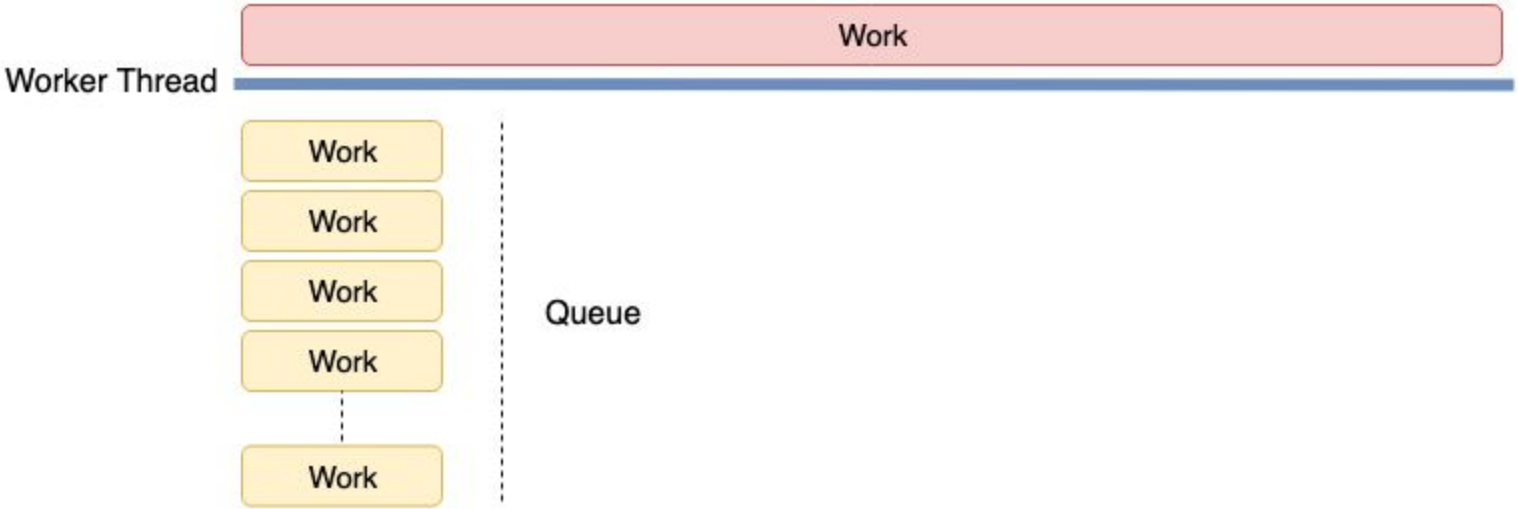
AsyncTask

+ Easy to implement
+ Communicating back to main thread is easy

AsyncTask

+ Easy to implement
+ Communicating back to main thread is easy
- **Can potentially block tasks on other AsyncTasks**

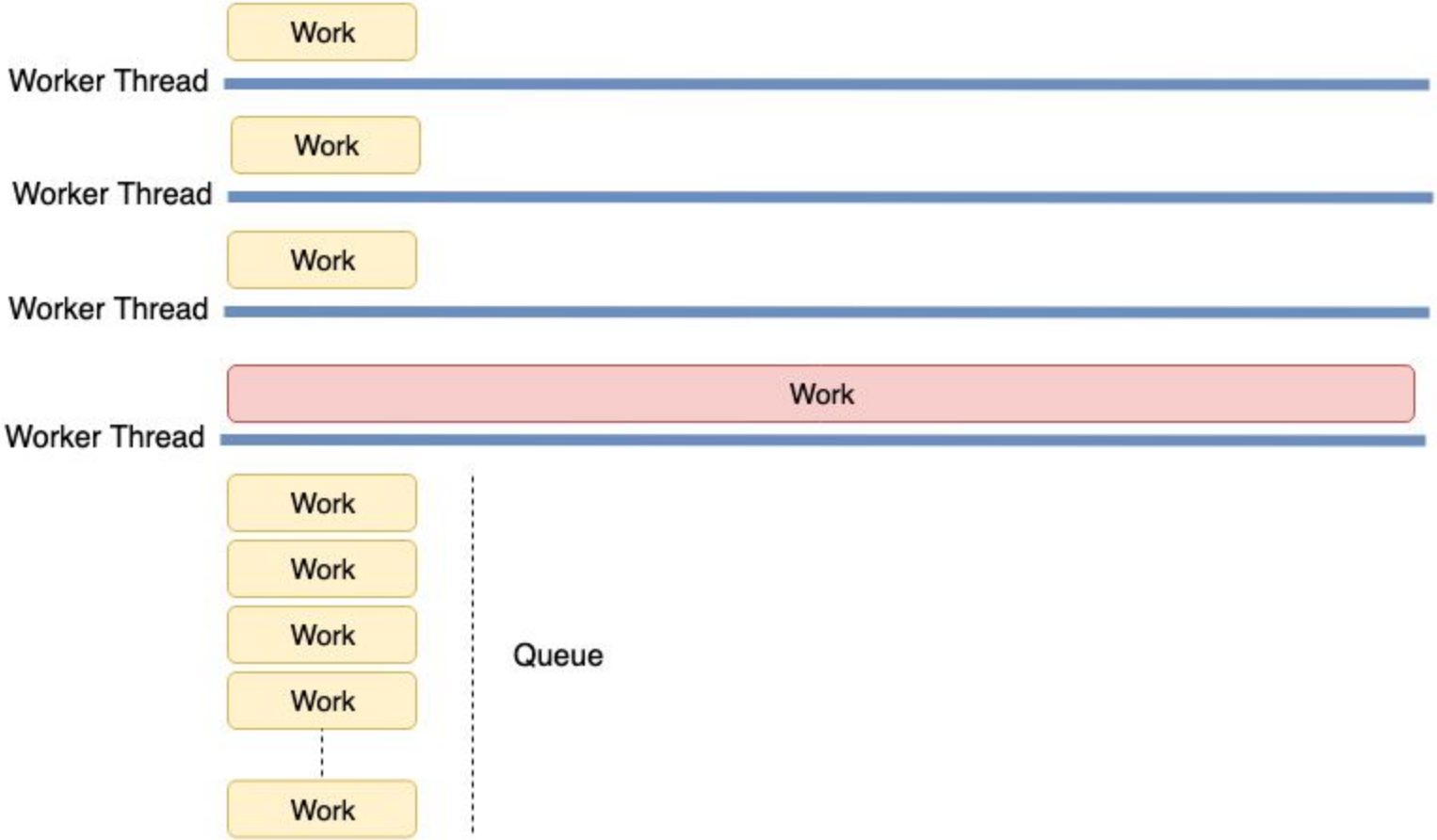
AsyncTask



AsyncTask

+ Easy to implement
+ Communicating back to main thread is easy
- **Can potentially block tasks on other AsyncTasks**

```
MyAsyncTask asyncTask = new MyAsyncTask();
asyncTask.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR, IMAGE_URL);
```


Work
Worker Thread
Work
Worker Thread
Work
Worker Thread
Work
Worker Thread
Work
Work
Work
Work
Work
Queue

Multi-Threading Alternatives

- Simple Threads
- AsyncTask
- **IntentService**
- Handlers
- ThreadPoolExecutor

Intent Service

```
Intent intent = new Intent(this, PingIntentService.class);
intent.putExtra("receiver", messageReceiver);
startService(intent);

class PingReceiver extends ResultReceiver {
    @Override
    protected void onReceiveResult(int resultCode, Bundle resultData) {
        activity.runOnUiThread(() -> {
            // Notify
        });
    }
}
```

```
public class PingIntentService extends IntentService {
    @Override
    protected void onHandleIntent(@Nullable Intent intent) {
        ResultReceiver receiver = intent.getParcelableExtra("receiver");
        // Do bg work
        receiver.send(new Random().nextInt(), null);
    }
}
```

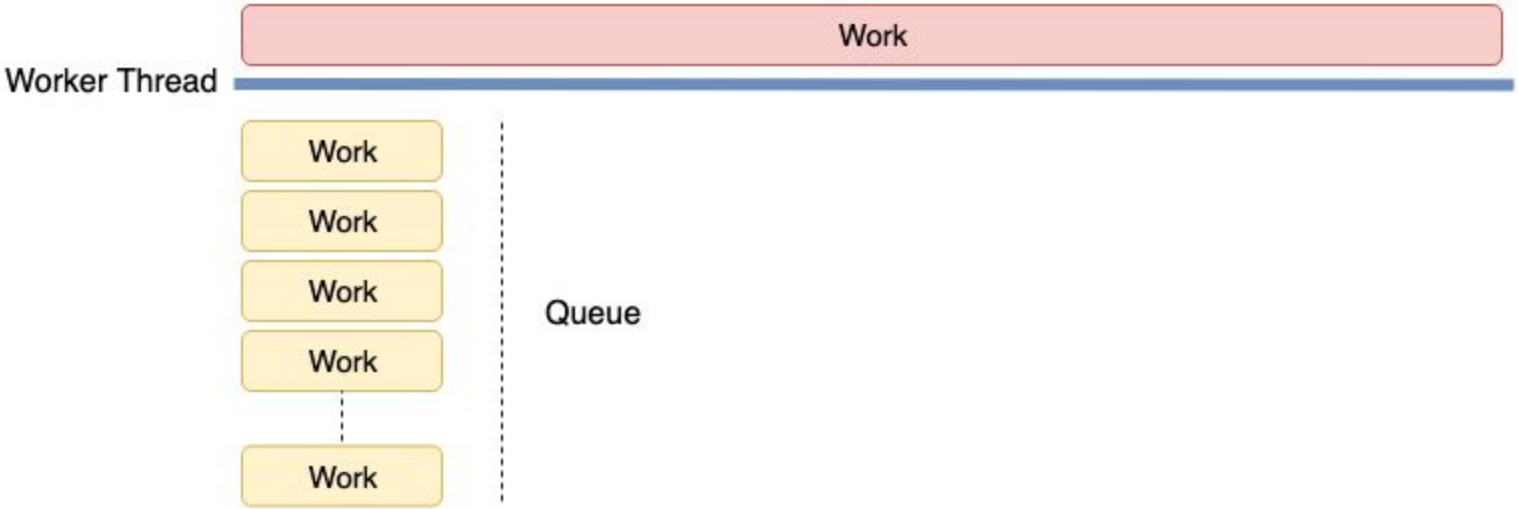
Intent Service

+ Maintains a queue of tasks
+ Quits automatically once done
+ Fail safe for single task/recent intent

Intent Service

+ Maintains a queue of tasks
+ Quits automatically once done
+ Fail safe for single task/recent intent
- Not versatile
- Sending results back to caller is not straight forward

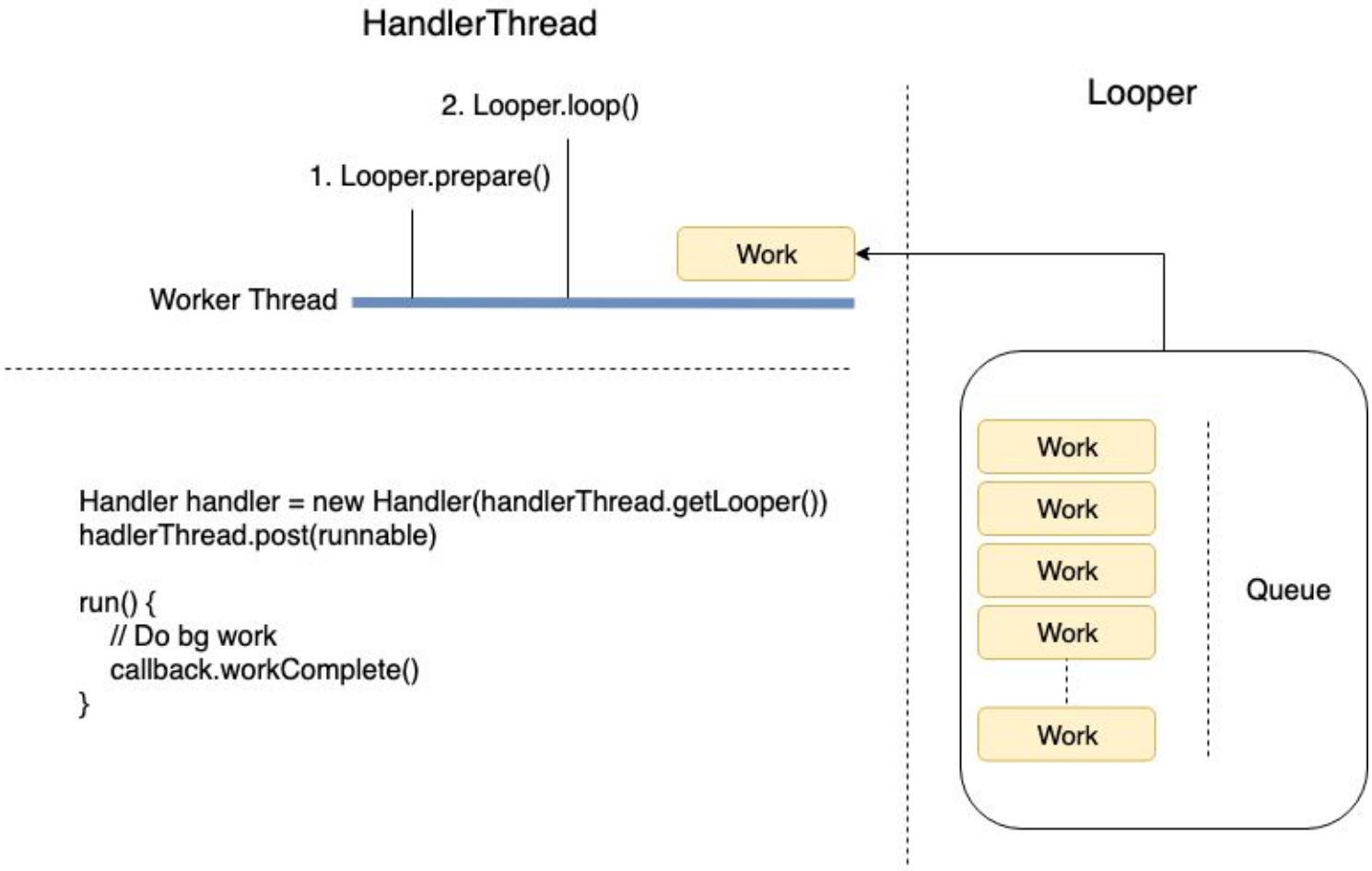
Intent Service… Same problem…?



Multi-Threading Alternatives

- Simple Threads
- AsyncTask
- IntentService
- **Handlers**
- ThreadPoolExecutor

Handlers




```
Handler handler = new Handler(handlerThread.getLooper())
hadlerThread.post(runnable)

run() {
    // Do bg work
    callback.workComplete()
}
```

Handlers

+ Maintains queue
+ Versatile
+ Sending callbacks easy

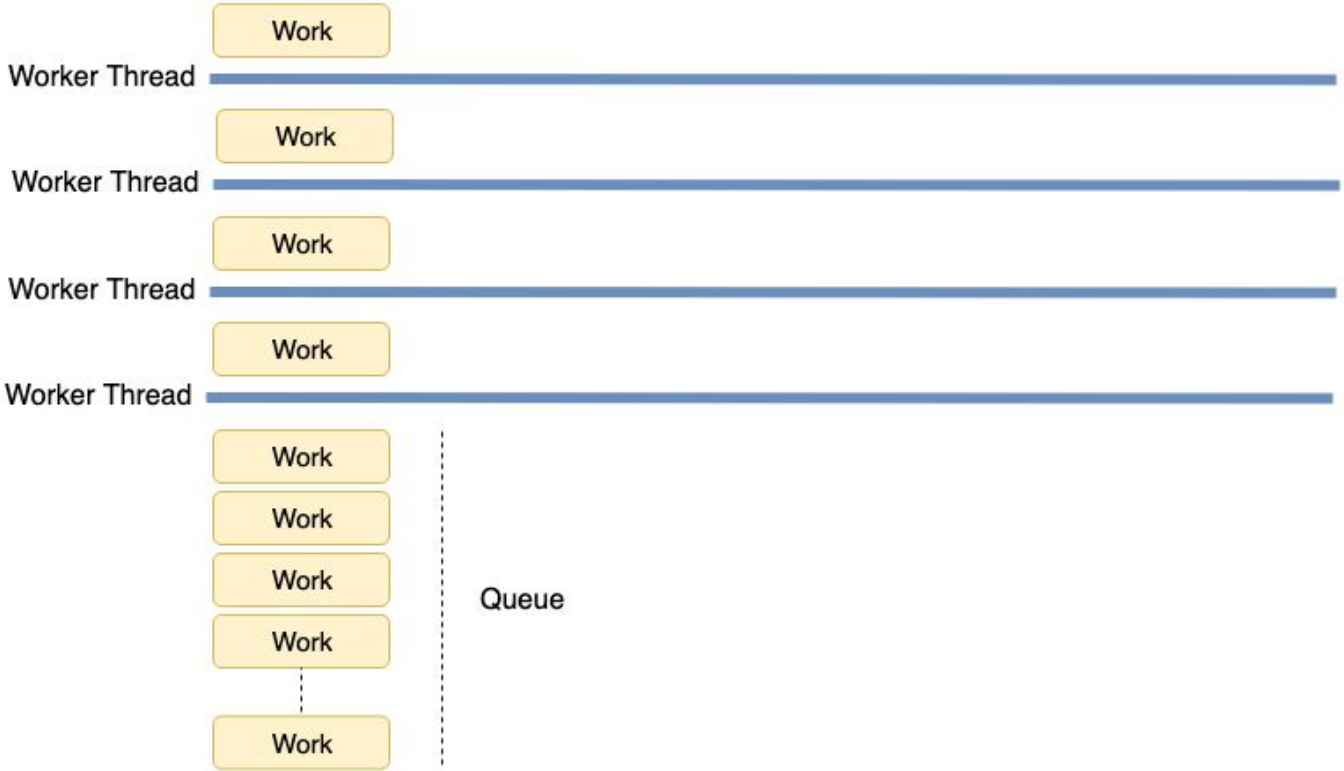
Handlers

+ Maintains queue
+ Versatile
+ Sending callbacks easy
- Needs to be quit once done

Multi-Threading Alternatives

- Simple Threads
- AsyncTask
- IntentService
- Handlers
- **ThreadPoolExecutor**

Thread Pool Executors

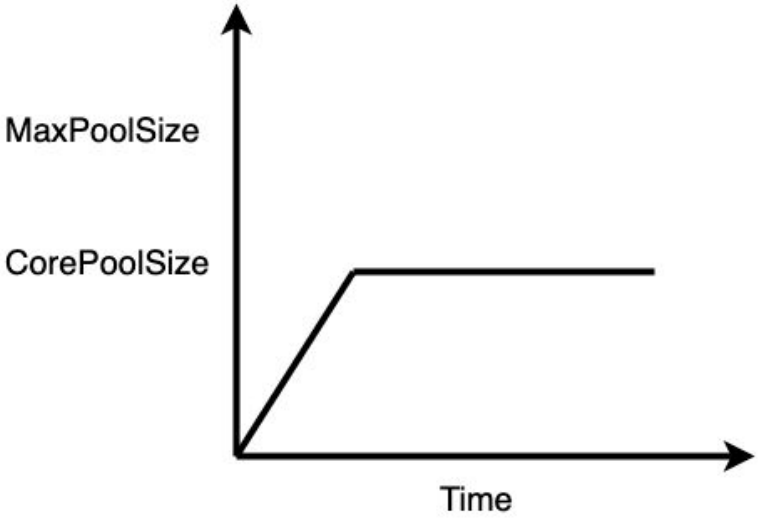


Thread Pool Executors: Configuration

```
1  corePoolSize = NUMBER_OF_CORES + 2;
2  maximumPoolSize = corePoolSize + 2;
3
4  ThreadPoolExecutor threadPoolExecutor =
5      new ThreadPoolExecutor(
6          corePoolSize,
7          maximumPoolSize,
8          KEEP_ALIVE_TIME_SECS,
9          TimeUnit.SECONDS,
10         new LinkedBlockingDeque<>());
11
12 threadPoolExecutor.execute(runnable);
```

Thread Pool Executors: Configuration

```
1  corePoolSize = NUMBER_OF_CORES + 2;
2  maximumPoolSize = corePoolSize + 2;
3
4  ThreadPoolExecutor threadPoolExecutor =
5      new ThreadPoolExecutor(
6          corePoolSize,
7          maximumPoolSize,
8          KEEP_ALIVE_TIME_SECS,
9          TimeUnit.SECONDS,
10         new LinkedBlockingDeque<>());
11
12 threadPoolExecutor.execute(runnable);
```

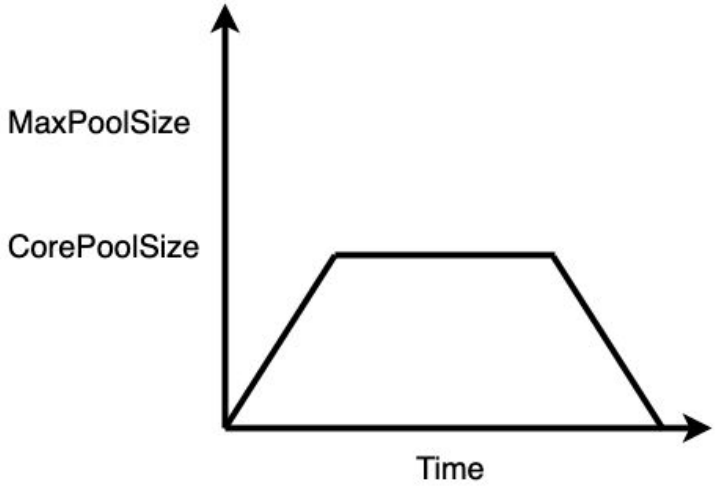


Thread Pool Executors: Configuration

```
1  corePoolSize = NUMBER_OF_CORES + 2;
2  maximumPoolSize = corePoolSize + 2;
3
4  ThreadPoolExecutor threadPoolExecutor =
5      new ThreadPoolExecutor(
6          corePoolSize,
7          maximumPoolSize,
8          KEEP_ALIVE_TIME_SECS,
9          TimeUnit.SECONDS,
10         new LinkedBlockingDeque<>());
11 allowCoreThreadTimeOut(true);
12
13 threadPoolExecutor.execute(runnable);
```

Thread Pool Executors: Configuration

```
1  corePoolSize = NUMBER_OF_CORES + 2;
2  maximumPoolSize = corePoolSize + 2;
3
4  ThreadPoolExecutor threadPoolExecutor =
5      new ThreadPoolExecutor(
6          corePoolSize,
7          maximumPoolSize,
8          KEEP_ALIVE_TIME_SECS,
9          TimeUnit.SECONDS,
10         new LinkedBlockingDeque<>());
11 allowCoreThreadTimeOut(true);
12
13 threadPoolExecutor.execute(runnable);
```



Thread Pool Executors

+ Maintains queue
+ Can execute multiple tasks at same time
+ Sending callbacks easy

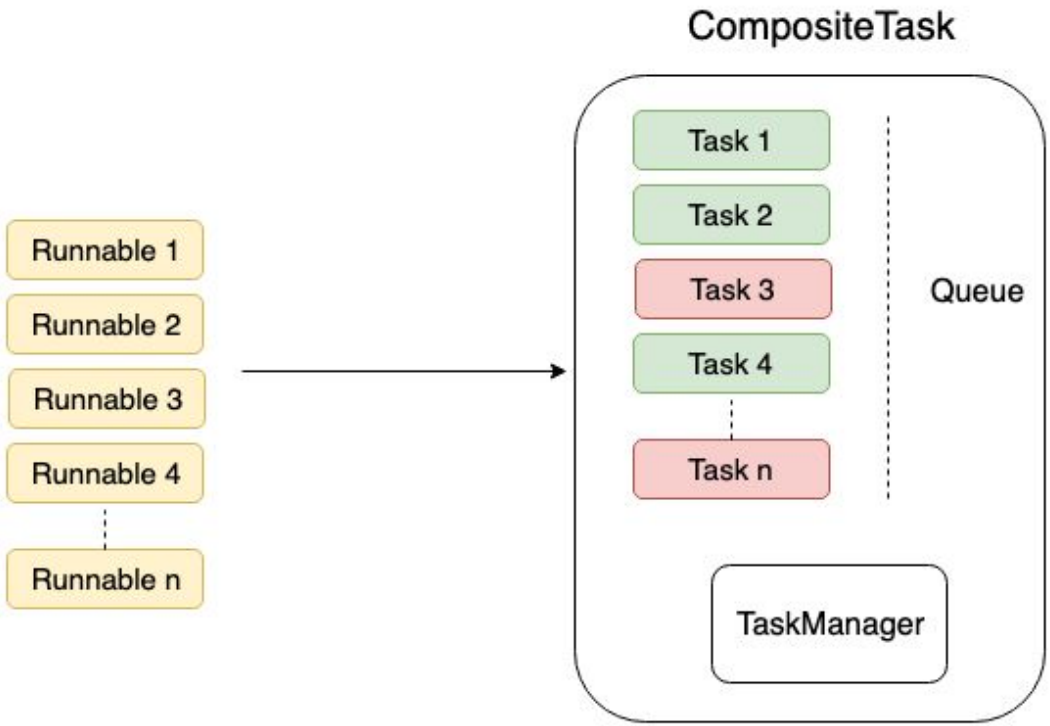
TaskManager: By OYO

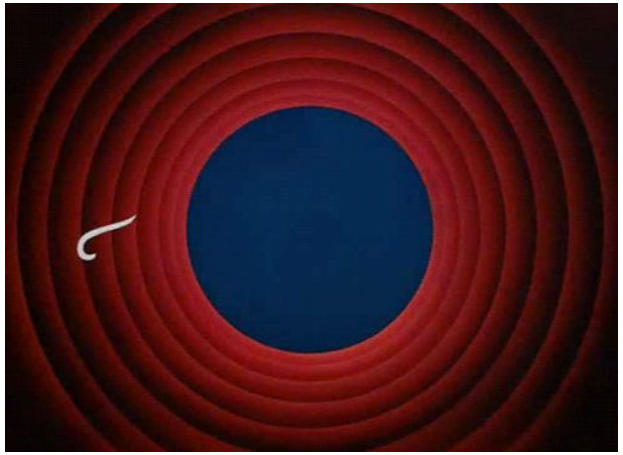
+ Can execute tasks in background as well as on main thread.
+ Can compose multiple background or foreground tasks together to execute serially.

```
1  TaskManager.get().onBackground(bgRunnable);
2
3  TaskManager.get().onMain(mainRunnable);
4
5  TaskManager.get().compose()
6      .onBackground(task1)
7      .onBackground(task2)
8      .onBackground(task4)
9      .onMain(task4));
```

TaskManager

- Uses ThreadPoolExecutor for background tasks.
- Uses Handler on UI thread for tasks to be executed on main thread.





OYO

Mail me your resume:
shiv.malik@oyorooms.com
LinkedIn:
https://www.linkedin.com/in/shiv-kumar-malik