Android Developer Fundamentals V2

Background Tasks

Lesson 7

7.1 AsyncTask and AsyncTaskLoader

Contents

- Threads
- AsyncTask
- Loaders
- AsyncTaskLoader

Threads

Threads Example

Car extends Thread { // extends GameObject implements Runnable

… // name/ID/xPosition/yPosition

@Override

public void run(){ /** put code to run in parallel here; parametrizable with ID… **/ }

}

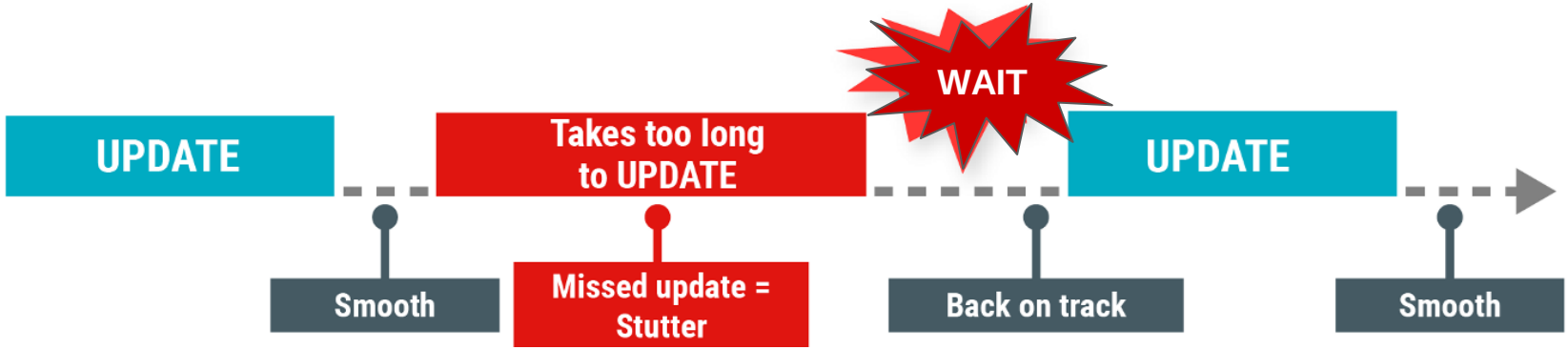
…main(){ Car c1 = new Car(…,1,…); Car c2 = new Car(…,2,…);
~~c1.run(); c2.run();~~
c1.start(); c2.start(); }

The main thread

- Independent path of execution in a running program
- Code is executed line by line
- App runs on Java thread called "main" or "UI thread"
- Draws UI on the screen
- Responds to user actions by handling UI events

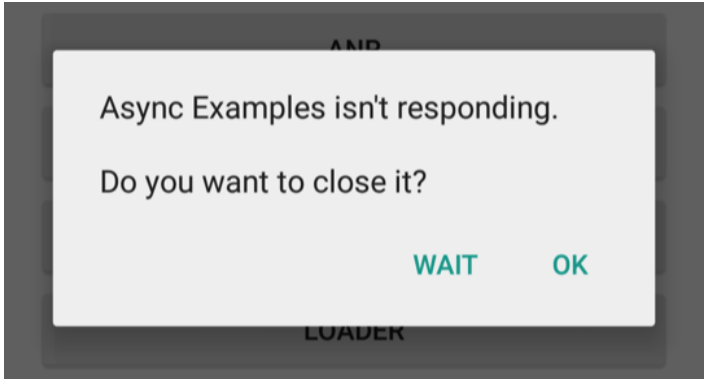
The Main thread must be fast

- Hardware updates screen every 16 milliseconds
- UI thread has 16 ms to do all its work
- If it takes too long to update (currently 5 sec), app stutters



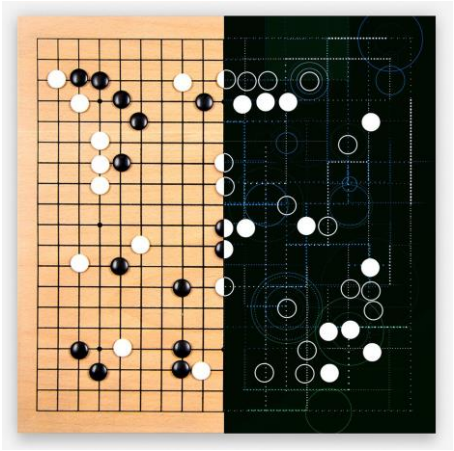
Users uninstall unresponsive apps

- If the UI waits too long for an operation to finish, it becomes unresponsive
- The framework shows an Application Not Responding (ANR) dialog



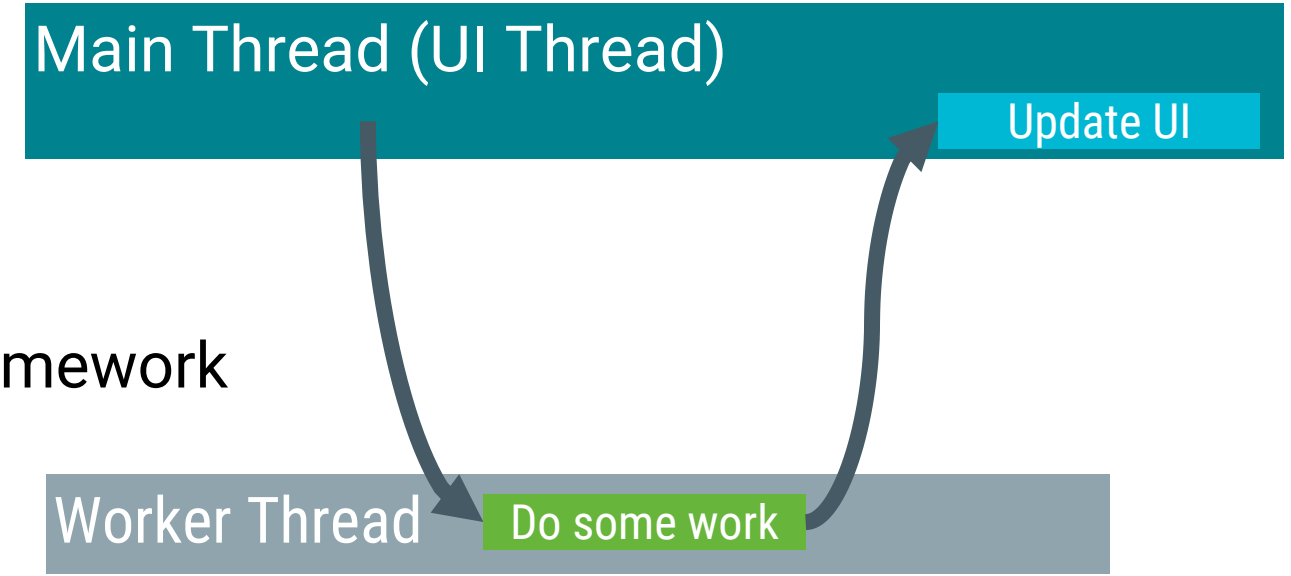
What is a long running task?

- Network operations
- Long calculations
- Downloading/uploading files
- Processing images
- Loading data



Background threads

Execute long running tasks on a **background thread**



- AsyncTask
- The Loader Framework
- Services

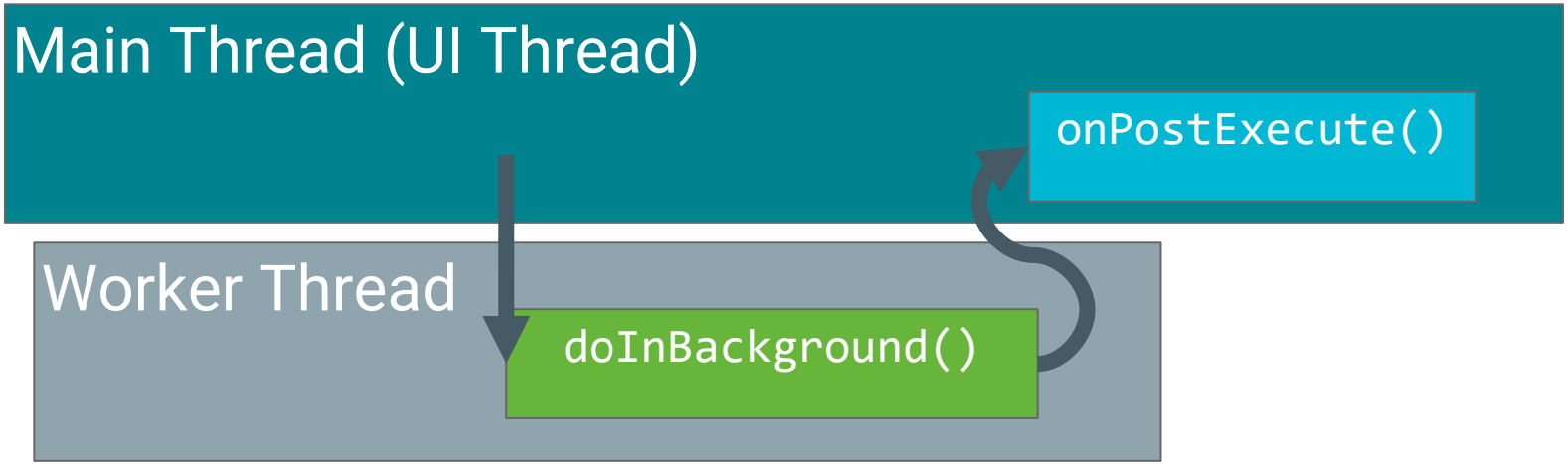
Two rules for Android threads

- Do not block the UI thread
  - Complete all work in less than 16 ms for each screen
  - Run slow non-UI work on a non-UI thread
- Do not access the Android UI toolkit from outside the UI thread
  - Do UI work only on the UI thread

AsyncTask

What is AsyncTask?

Use AsyncTask to implement basic background tasks



Override two methods

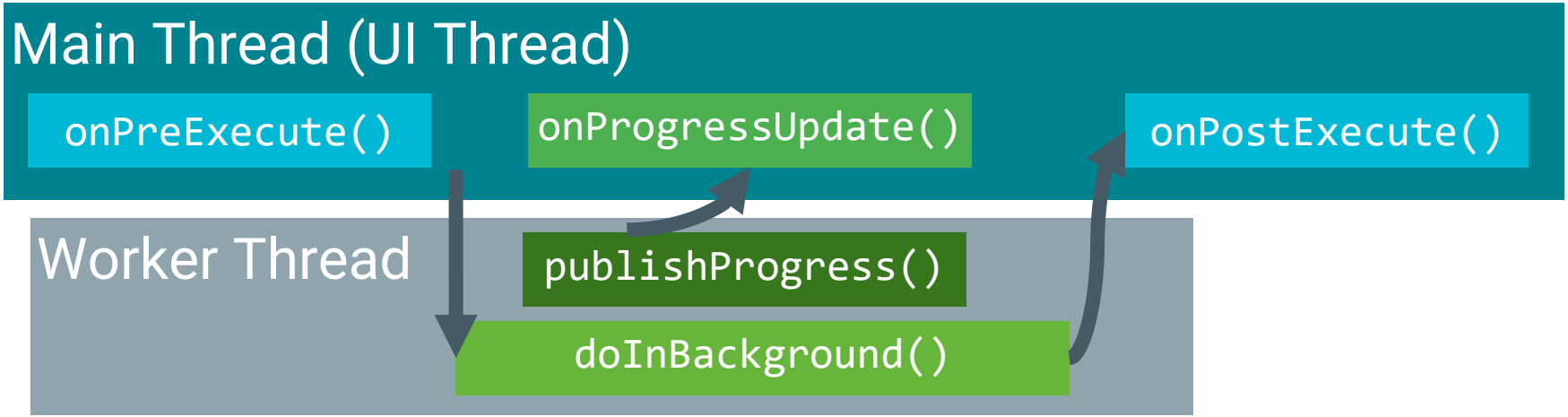
- `doInBackground()`—runs on a background thread
  - All the work to happen in the background
- `onPostExecute()`—runs on main thread when work done
  - Process results
  - Publish results to the UI

AsyncTask helper methods

- `onPreExecute()`
  - Runs on the main thread
  - Sets up the task

- `onProgressUpdate()`
  - Runs on the main thread
  - receives calls from `publishProgress()` from background thread

AsyncTask helper methods



Creating an AsyncTask

1. Subclass AsyncTask
2. Provide data type sent to `doInBackground()`
3. Provide data type of progress units for `onProgressUpdate()`
4. Provide data type of result for `onPostExecute()`

```
private class MyAsyncTask
        extends AsyncTask<URL, Integer, Bitmap> {...}
```

MyAsyncTask class definition

```
private class MyAsyncTask
    extends AsyncTask<String, Integer, Bitmap> {...}
```

- String—could be query, URI for filename
- Integer—percentage completed, steps done
- Bitmap—an image to be displayed
- Use Void if no data passed

onPreExecute()

```
protected void onPreExecute() {
    // display a progress bar
    // show a toast
}
```

doInBackground()

```
protected Bitmap doInBackground(String... query) {
    // Get the bitmap
    return bitmap;
}
```

onProgressUpdate()

```
protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }
```

onPostExecute()

```
protected void onPostExecute(Bitmap result) {
    // Do something with the bitmap
}
```

Start background work

```
public void loadImage (View view) {
    String query = mEditText.getText().toString();
    new MyAsyncTask(query).execute();
   }
```

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
     protected Long doInBackground(URL... urls) {
         int count = urls.length;
         long totalSize = 0;
         for (int i = 0; i < count; i++) {
             totalSize += Downloader.downloadFile(urls[i]);
             publishProgress((int) ((i / (float) count) * 100));
             // Escape early if cancel() is called
             if (isCancelled()) break;
         }
         return totalSize;
     }

     protected void onProgressUpdate(Integer... progress) {
         setProgressPercent(progress[0]);
     }

     protected void onPostExecute(Long result) {
         showDialog("Downloaded " + result + " bytes");
     }
 }
```

Limitations of AsyncTask

- When device configuration changes, Activity is destroyed
- AsyncTask cannot connect to Activity anymore
- New AsyncTask created for every config change
- Old AsyncTasks stay around
- App may run out of memory or crash

When to use AsyncTask

- Short or interruptible tasks
- Tasks that do not need to report back to UI or user
- Lower priority tasks that can be left unfinished
- Use AsyncTaskLoader otherwise