Multithreading in Android

Mobile and Ubiquitous Computing

MEIC/MERC 2016/17

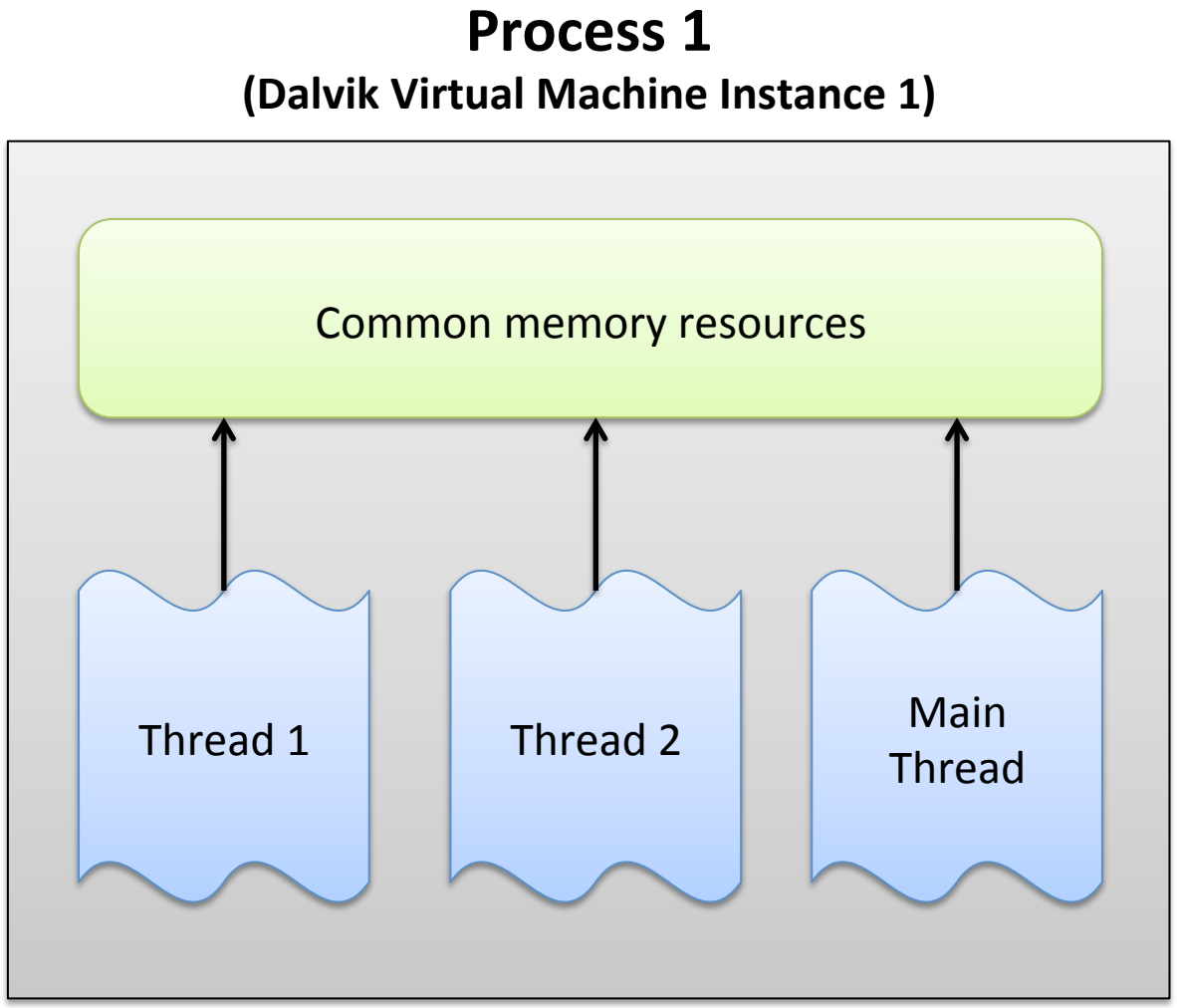
Nuno Santos

Processes, Threads, and Components

- When an app is launched, Android starts a new Linux process
  - The process executes a Dalvik virtual machine instance
- Dalvik starts a single thread of execution called main thread
- The main thread handles all components of the app
- If an app component starts and a process exists for that app, then:
  - The component is started within that process
  - Uses the same thread of execution
- Components can be arranged to run in separate processes
- It is possible to create additional threads for any process

Multithreading

- A thread is a concurrent unit of execution
- Each thread has its own call stack
- The call stack is used on method calling, parameter passing, and storage for the called method's local variables



Creating a Thread

- Implement the thread code:

```
Runnable tLogic= new Runnable {
    public void run() {
        // do some work
    }
}
```

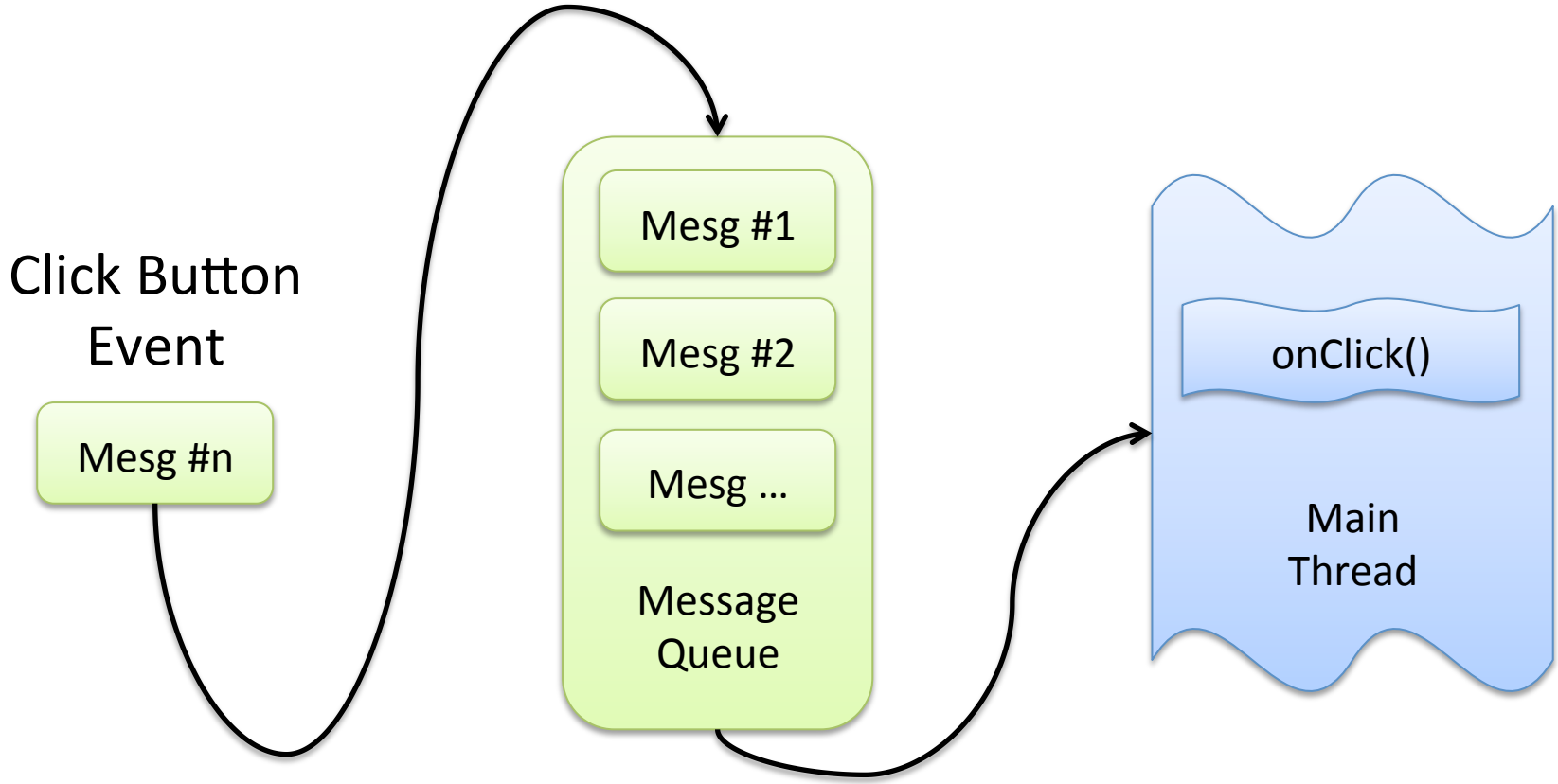
- Create the thread and launch it:

```
Thread t = new Thread(tLogic, “My Thread”);
t.start();
```

Main Thread (aka UI Thread)

- Very important thread: handles UI
  - It's in charge of dispatching events to the appropriate user interface widgets, including drawing events
- The system does not create a separate thread for each instance of a component
  - All components that run in the same process are instantiated in the UI thread
- System calls to each component are dispatched from that thread
  - Therefore, methods that respond to system callbacks (e.g., onKeyDown()) always run in the UI of the process

Handling UI Events



Beware of Long Operations in UI Thread!

```
public void onClick(View v) {
      Bitmap b = loadImageFromNetwork(…);  // long op
      mImageView.setImageBitmap(b);        // update UI
}
```

- Long running operations will block the whole UI
  - No event can be dispatched: the apps appears hung
  - If blocked for too much time “application not responding” dialog pops up

1st Attempt: Long Ops in Worker Thread

```
public void onClick(View v) {
  new Thread(new Runnable() {
    public void run() {
      Bitmap b = loadImageFromNetwork(…);  // do long ops
      mImageView.setImageBitmap(b);        // update UI
    }
  }).start();                              // execute thr
}
```

- Good: does not block UI thread
- Problem: Android UI toolkit is not thread-safe and must be always manipulated in the UI thread
  - In this code, `ImageView` is manipulated on a worker thread
  - Could be the source of nasty bugs!

Access UI Thread from Worker Threads

- Multiple ways, but the code gets pretty complicated…
  - Activity.runOnUiThread(Runnable)
  - View.post(Runnable)
  - View.postDelayed(Runnable, long)
  - Handler

```
public void onClick(View v) {
  new Thread(new Runnable() {
    public void run() {
      final Bitmap b = loadImageFromNetwork();
      mImageView.post(new Runnable() {
        public void run() {
          mImageView.setImageBitmap(b);
        }
      });
    }
  }).start();
}
```

Long-Running Tasks with AsyncTask

- Simplify the creation of long-running tasks that need to communicate with the UI

```
public void onClick(View v) {
  new DownloadImageTask().execute("http://…");
}

private class DownloadImageTask extends AsyncTask {
     protected Bitmap doInBackground(String... urls) {
         return loadImageFromNetwork(urls[0]);
     }

     protected void onPostExecute(Bitmap result) {
         mImageView.setImageBitmap(result);
     }
 }
```

AsyncTask

- AsyncTask must be used by subclassing it
- It has to be created in the UI thread and can be executed only once
- Overview:
  - You can specify the type, using generics, of the parameters, the progress values and the final value of the task
  - doInBackground() executes automatically on a worker thread
  - onPreExecute(), onPostExecute() and onProgressUpdate() are all invoked on the UI thread
  - The value returned by doInBackground() is sent to onPostExecute()
  - You can call publishProgress() at anytime in doInBackground() to execute onProgressUpdate() on the UI thread
  - You can cancel the task at any time, from any thread

Useful Pointers

- Processes and Threads
  - http://developer.android.com/guide/components/processes-and-threads.html
- Android Thread Model
  - http://mcatr.blogspot.pt/2013/06/android-thread-model.html
- Common Tasks and How to Do Them in Android
  - http://developer.android.com/guide/faq/commontasks.html#threading