SCIENCES SORBONNE UNIVERSITÉ

Threads

Renault@lrde.epita.fr

Overview

- **An Android Application is executed by a process**
  - A thread, called **main thread** or **UI thread**, is in charge of updating GUI
- **One thread per component**
  - The UI Thread of the component at the top of the back stack runs
  - Thread UI manage callbacks
- **An application performing a lot of computing must use different threads**

Rules

- 1 - Do not block UI Thread
- 2 - Only UI Thread can modify UI
  - Android Toolkit UI is not thread safe

```
public void onClick(View v) {
  new Thread(new Runnable() {
    public void run() {
      Bitmap b =
        loadImageFromNetwork("http://example.com/image.png");
      mImageView.setImageBitmap(b);
   }
 }).start();
}
```

How to update GUI then ?

To update GUI from another Thread

- Use Asynchronous tasks
- Use dedicated methods that take a thread as parameter
  - **Activity.onRunUI(Runnable)**
  - **View.post(Runnable)**
  - **View.postDelayed(Runnable, long)**

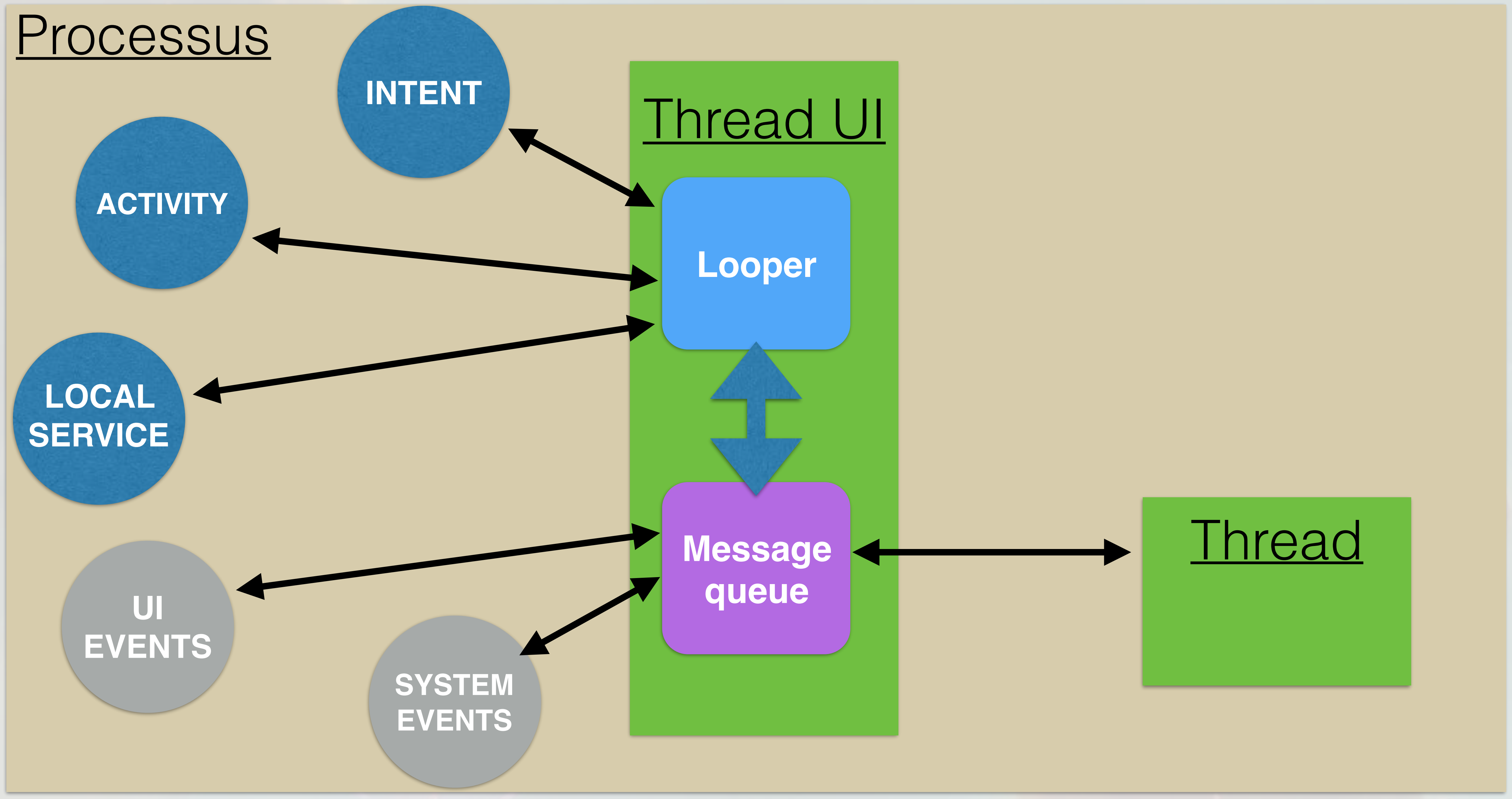
```
public void onClick(View v) {
 new Thread(new Runnable() {
    public void run() {
      final Bitmap bitmap =
         loadImageFromNetwork("http://example.com/image.png");
      mImageView.post(new Runnable() {
          public void run() {
              mImageView.setImageBitmap(bitmap);
          }
      });
    }
   }).start();
```

Message Queue & Looper

- **A thread holds a message queue**
  - For all action and callback to perform later
- **This queue is thread-safe**
- **Messages from this queue will be flushed by the Looper**
  - When a message is received, the looper treat it
  - To do so, handlers are used
- **By default, only the UI Thread have a looper and a message queue**

How does the looper work?

- We can force the UI Thread to do something thanks to the looper



Creating its own Looper

We can build a callback looper in every thread

- To do so, use Handlers

```
class LooperThread extends Thread {
  public Handler mHandler;
  public void run() {
    //Initialize the current thread as a looper.
    Looper.prepare();

    //instance a Handler of the current thread
    mHandler = new Handler() {
       // process incoming messages here
      public void handleMessage(Message msg) {
      }
    };

    //Run the message queue in this thread.
    Looper.loop();
 }
```

Define Handler in the UI Thread

The handler is associated to a given thread

```
private ThreadCompute mThread;
private Handler mHandler;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mHandler = new Handler(Looper.getMainLooper()) {
        @Override
        public void handleMessage(Message inputMessage) {
            Toast.makeText(getApplicationContext(),
               inputMessage.toString(), Toast.LENGTH_SHORT)
               .show();
        }
   };
   mThread = new ThreadCompute();
   mThread.start();
}
```

Define Handler in the UI Thread

The handler is associated to a given thread

```
void notifyUI(){
   Message completeMessage =  mHandler.obtainMessage();
   completeMessage.sendToTarget();
}

class ThreadCompute extends Thread {
    @Override
    public void run() {
        try {
           Thread.sleep(10000);
           notifyUI();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
};
```

Priorities

THREAD_PRIORITY_AUDIO	THREAD_PRIORITY_URGENT_AUDIO
THREAD_PRIORITY_BACKGROUND	THREAD_PRIORITY_LOWEST
THREAD_PRIORITY_DISPLAY	THREAD_PRIORITY_URGENT_DISPLAY
THREAD_PRIORITY_FOREGROUND	THREAD_PRIORITY_MOST_FAVORABLE
THREAD_PRIORITY_LESS_FAVORABLE	….

Summary

- **Only the UI Thread can modify the UI**
- **If another component want to modify the UI, it has to trigger an action on the UI Thread**
  - with AsyncTask
  - with predefined methods
  - with handler
- **An application can handle multiple threads**
  - onPrepare should be called to setup the looper
- **Knowing how to manage threads is important**