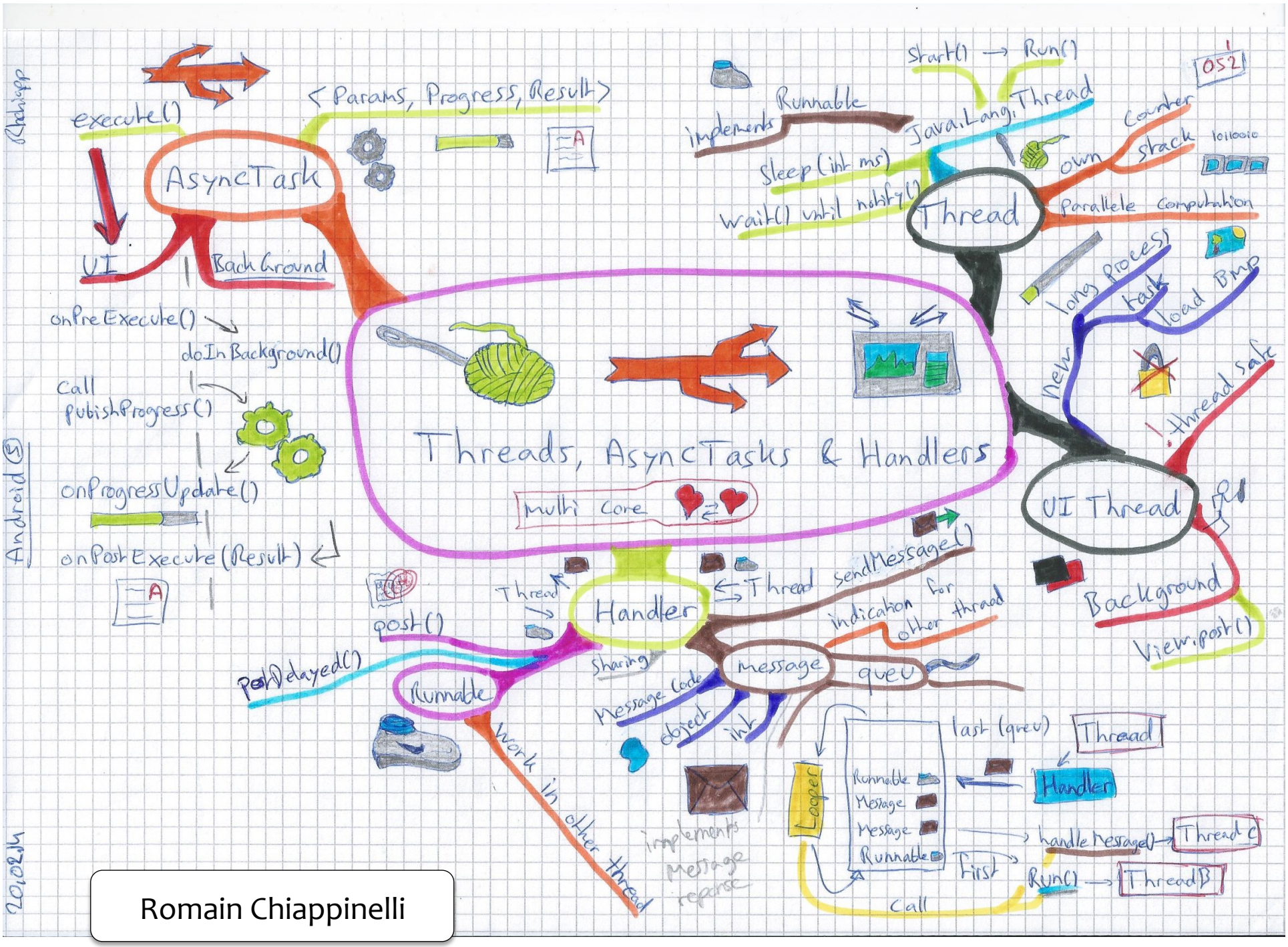


Romain Chiappinelli

Programming Handheld Systems

Adam Porter

Threads, AsyncTasks & Handlers

Today's Topics

Threading overview

Android's UI Thread

The AsyncTask class

The Handler class

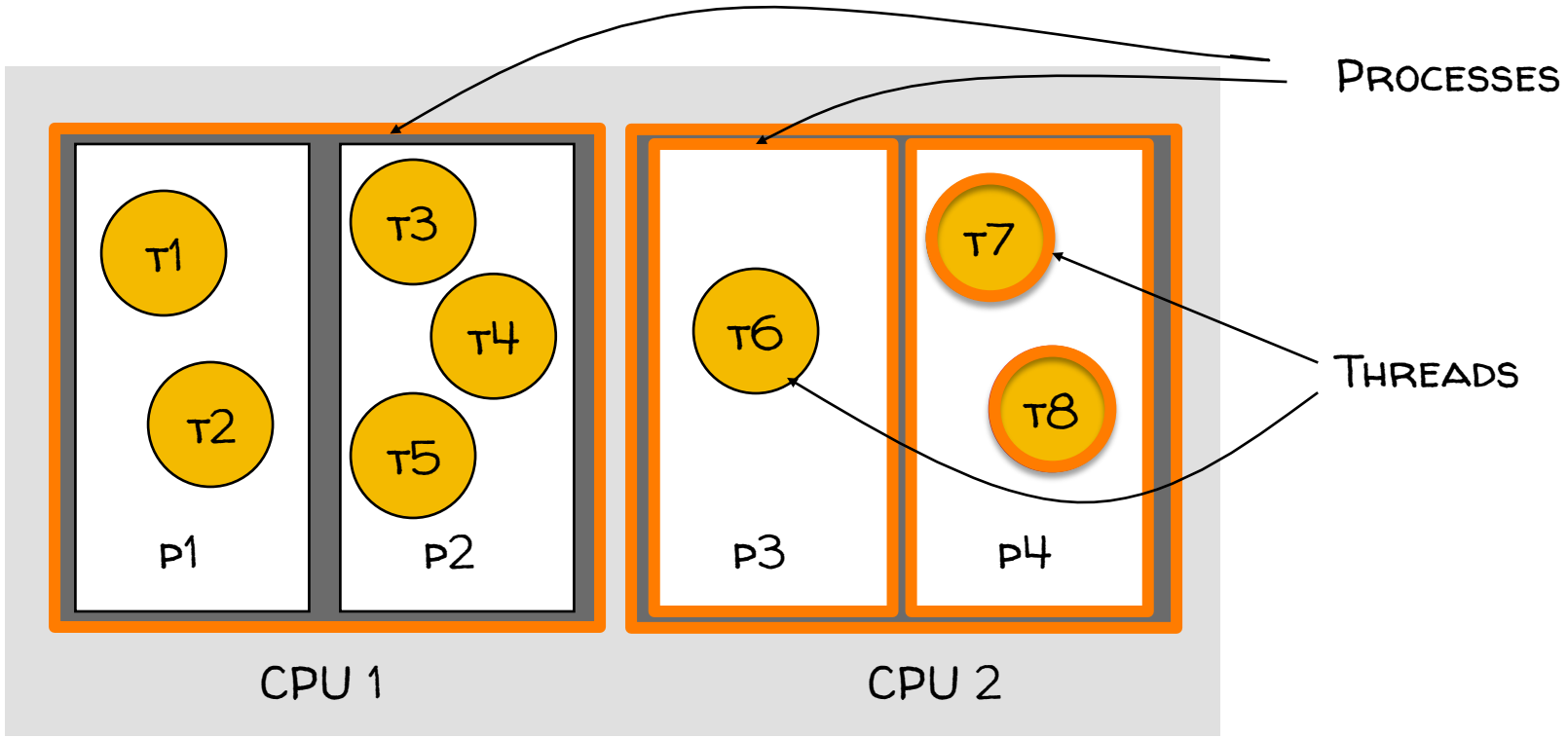
What is a Thread?

Conceptual view

- Parallel computation running in a process

Implementation view

- A program counter and a stack
- With heap and static areas that are shared with other threads



Computing Device

Java Threads

Represented by an object of type Java.lang.Thread

Threads implement the runnable interface

```
void run()
```

See:

http://docs.oracle.com/javase/tutorial/essential/concurrency/threads.html

Some Thread Methods

void start()

Starts the Thread

void sleep(long time)

Sleeps for the given period

Some Object Methods

void wait()

Current thread waits until another thread invokes notify() on this object

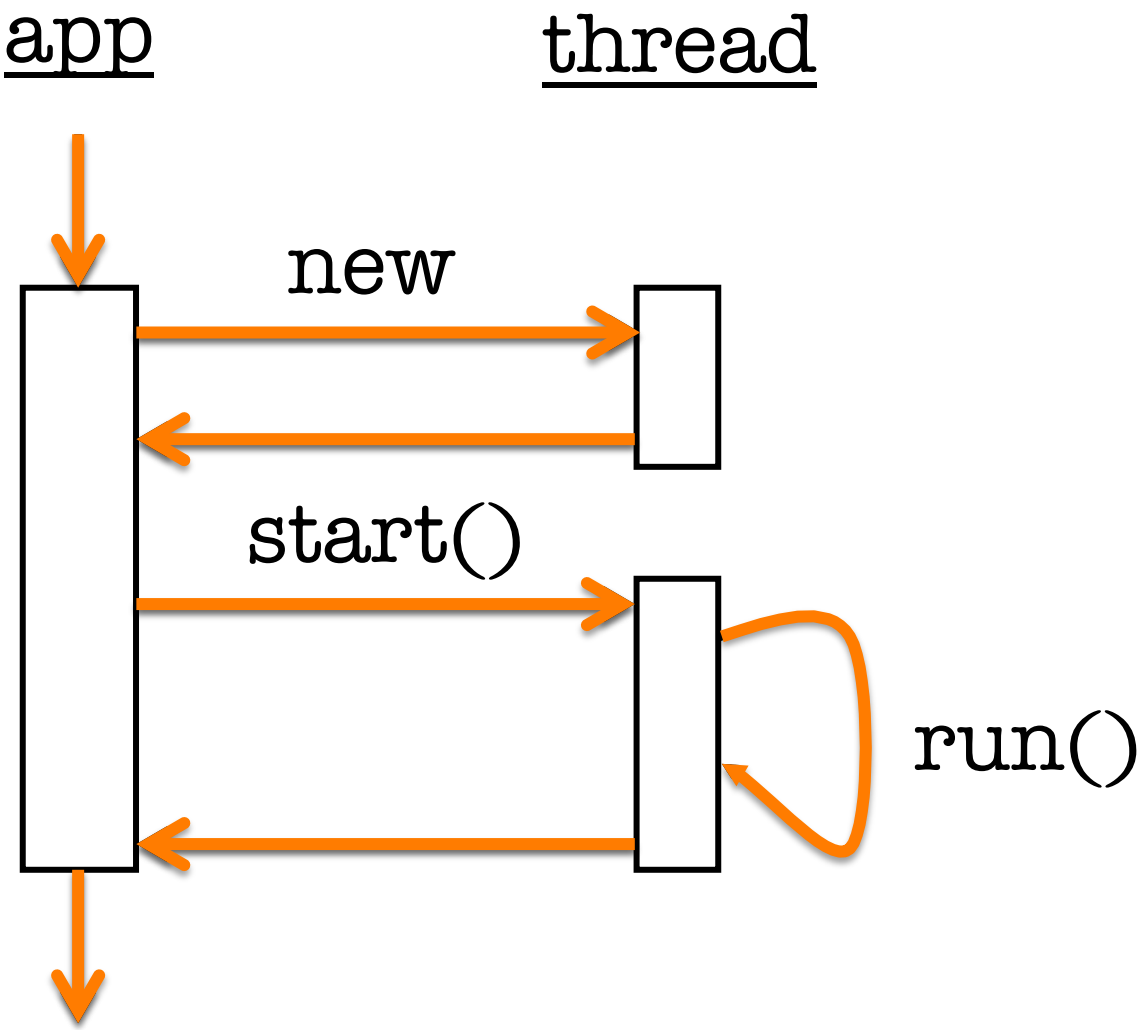
void notify()

Wakes up a single thread that is waiting on this object

Basic Thread Use Case

- Instantiate a Thread object
- Invoke the Thread's start() method
  - Thread's run() method get called
- Thread terminates when run() returns

Basic Thread Use Case



ThreadingNoThreading

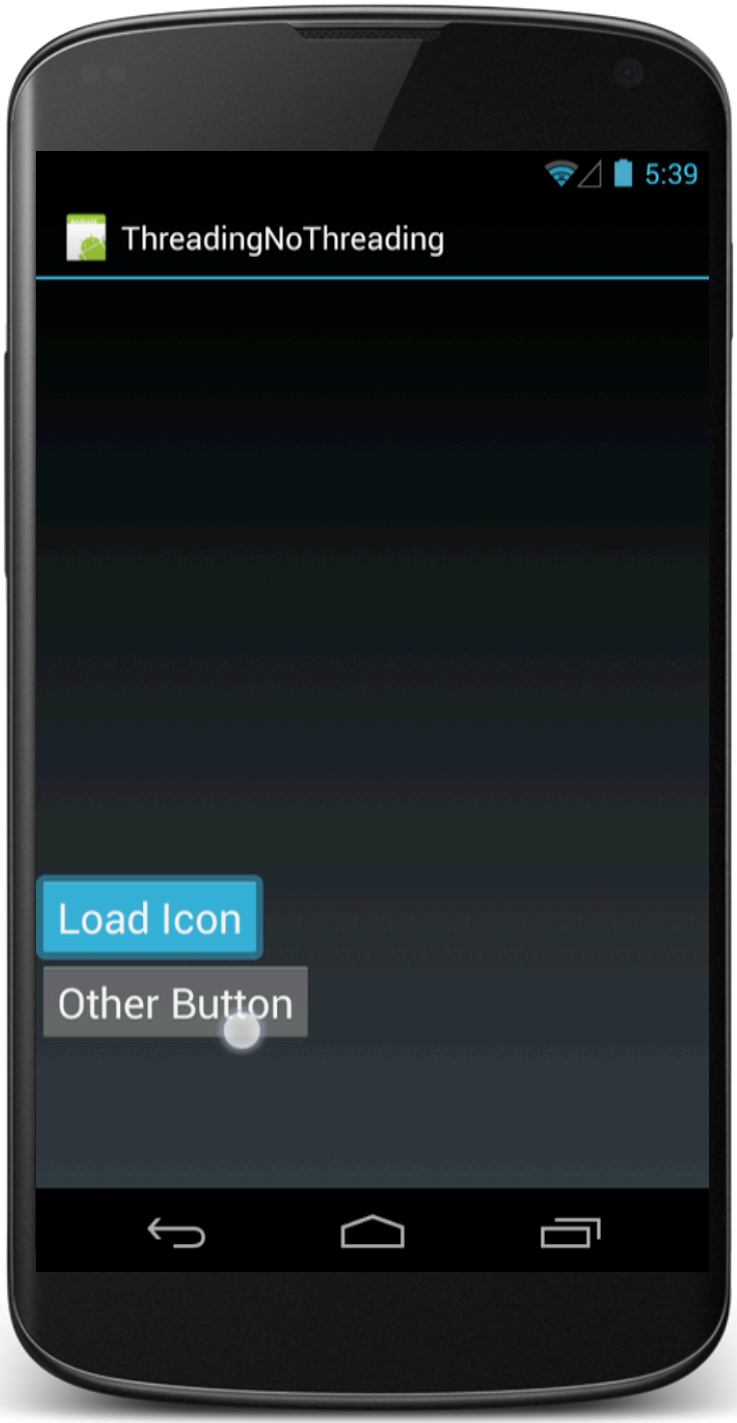
Application displays two buttons

LoadIcon

- Load a bitmap from a resource file & display
- Show loaded bitmap

Other Button

- Display some text


5:39
ThreadingNoThreading
Load Icon
Other Button

ThreadingSimple

Seemingly obvious, but incorrect, solution:

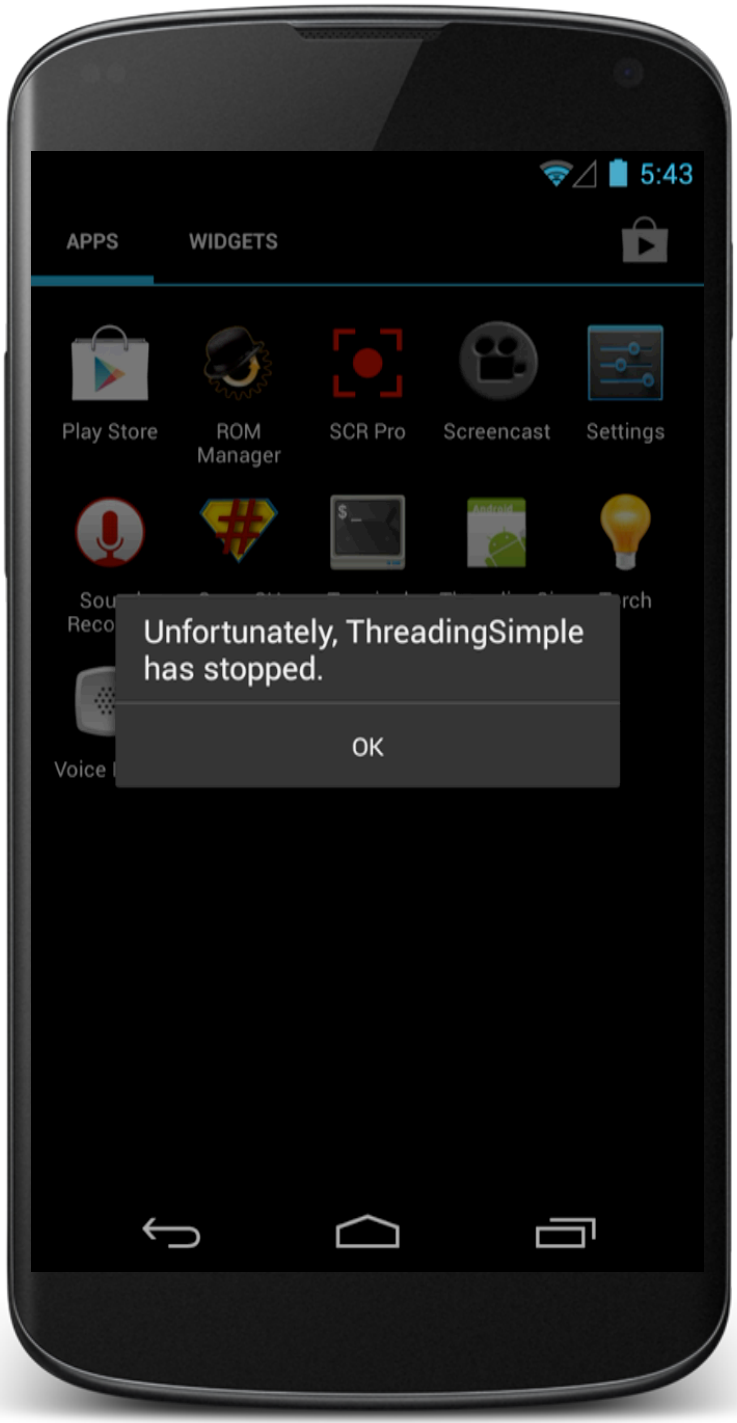
Button listener spawns a separate thread to load bitmap & display it

ThreadingSimple

```
public class SimpleThreadingExample extends Activity {

    private static final String TAG = "SimpleThreadingExample";

    private Bitmap mBitmap;
    private ImageView mIView;
    private int mDelay = 5000;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mIView = (ImageView) findViewById(R.id.imageView);

        final Button loadButton = (Button) findViewById(R.id.loadButton);
        loadButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                loadIcon();
            }
        });

        final Button otherButton = (Button) findViewById(R.id.otherButton);
        otherButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(SimpleThreadingExample.this, "I'm Working",
                        Toast.LENGTH_SHORT).show();
            }
        });
    }
```

ThreadingSimple

```
private void loadIcon() {
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                Thread.sleep(mDelay);
            } catch (InterruptedException e) {
                Log.e(TAG, e.toString());
            }
            mBitmap = BitmapFactory.decodeResource(getResources(),
                    R.drawable.painter);

            // This doesn't work in Android
            mIView.setImageBitmap(mBitmap);
        }
    }).start();
}
```


5:43
APPS
WIDGETS
Play Store
ROM Manager
SCR Pro
Screencast
Settings
Unfortunately, ThreadingSimple has stopped.
OK

The UI Thread

Applications have a main thread (the UI thread)

Application components in the same process use the same UI thread

User interaction, system callbacks & lifecycle methods handled in the UI thread

In addition, UI toolkit is not thread-safe

Implications

- Blocking the UI thread hurts application responsiveness
  - Long-running operations should run in background threads
- Don't access the UI toolkit from a non-UI thread

Improved Solution

Need to do work in a background thread, but update the UI in the UI Thread

Android provides several methods that are guaranteed to run in the UI Thread

boolean View.post (Runnable action)

void Activity.
runOnUiThread (Runnable action)


5:45
ThreadingViewPost
Load Icon
Other Button
I'm Working

ThreadingViewPost

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mImageView = (ImageView) findViewById(R.id.imageView);

    final Button button = (Button) findViewById(R.id.loadButton);
    button.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            loadIcon();
        }
    });

    final Button otherButton = (Button) findViewById(R.id.otherButton);
    otherButton.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            Toast.makeText(SimpleThreadingViewPostActivity.this, "I'm Working",
                    Toast.LENGTH_SHORT).show();
        }
    });
}
```

ThreadingViewPost

```
private void loadIcon() {
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                Thread.sleep(mDelay);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            mBitmap = BitmapFactory.decodeResource(getResources(),
                    R.drawable.painter);
            mImageView.post(new Runnable() {
                @Override
                public void run() {

                    mImageView.setImageBitmap(mBitmap);
                }
            });
        }
    }).start();
}
```

AsyncTask

Provides a structured way to manage work involving background & UI threads

AsyncTask

Background thread

- performs work
- Indicates progress

UI Thread

- Does setup
- Publishes intermediate progress
- Uses results

AsyncTask

Generic class

```
class AsyncTask<Params, Progress, Result> {
  ...
}
```

Generic type parameters

- Params – Type used in background work
- Progress – Type used when indicating progress
- Result – Type of result

AsyncTask

void onPreExecute()

Runs in UI Thread before doInBackground()

Result

doInBackground (Params...params)

Performs work in background Thread

May call

void publishProgress(Progress... values)

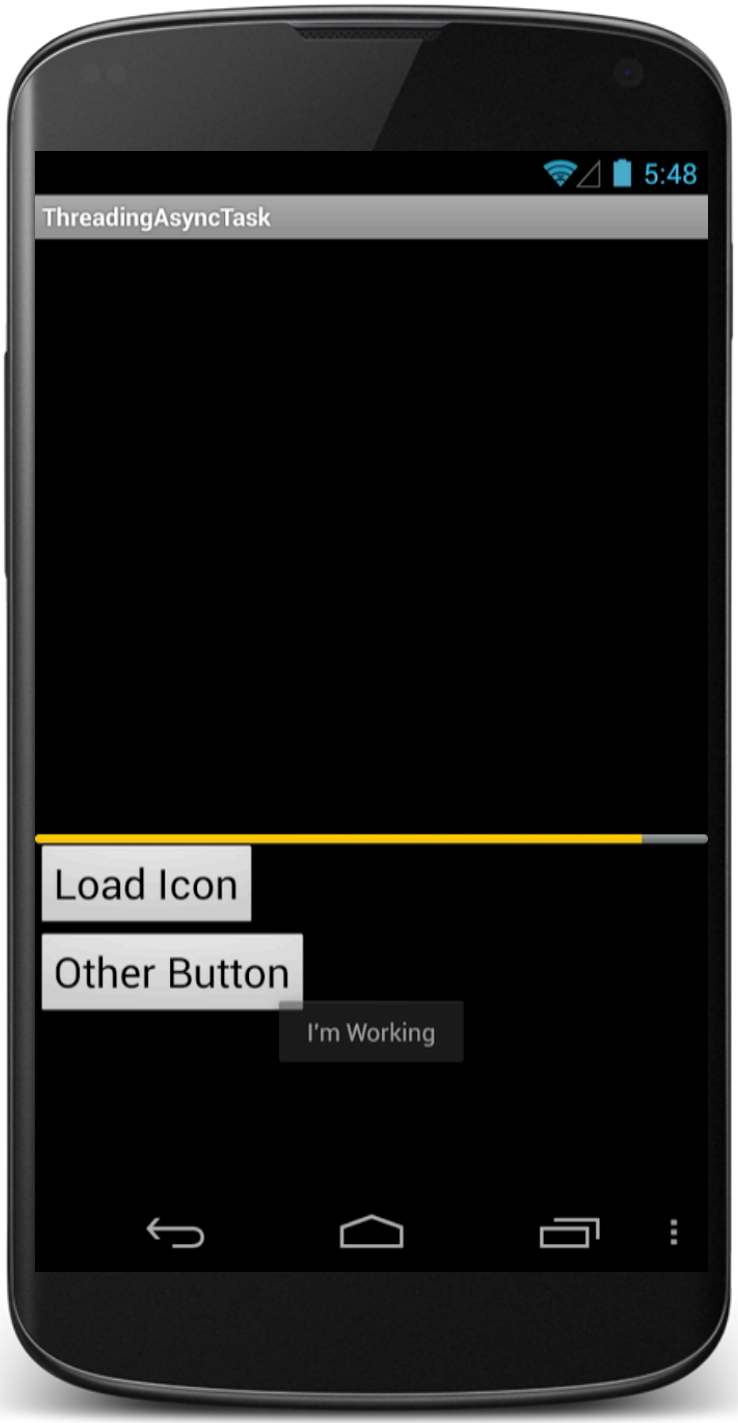
AsyncTask

void
onProgressUpdate (Progress... values)

Invoked in response to publishProgress()

void onPostExecute (Result result)

Runs after doInBackground()


5:48
ThreadingAsyncTask
Load Icon
Other Button
I'm Working

ThreadingAsyncTask

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mImageView = (ImageView) findViewById(R.id.imageView);;
    mProgressBar = (ProgressBar) findViewById(R.id.progressBar);

    final Button button = (Button) findViewById(R.id.loadButton);
    button.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            new LoadIconTask().execute(R.drawable.painter);
        }
    });

    final Button otherButton = (Button) findViewById(R.id.otherButton);
    otherButton.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            Toast.makeText(AsyncTaskActivity.this, "I'm Working",
                    Toast.LENGTH_SHORT).show();
        }
    });

}
```

ThreadingAsyncTask

```
class LoadIconTask extends AsyncTask<Integer, Integer, Bitmap> {


    @Override
    protected void onPreExecute() {
        mProgressBar.setVisibility(ProgressBar.VISIBLE);
    }

    @Override
    protected Bitmap doInBackground(Integer... resId) {
        Bitmap tmp = BitmapFactory.decodeResource(getResources(), resId[0]);
        // simulating long-running operation
        for (int i = 1; i < 11; i++) {
            sleep();
            publishProgress(i * 10);
        }
        return tmp;
    }
```

ThreadingAsyncTask

```
    }

    @Override
    protected void onProgressUpdate(Integer... values) {
        mProgressBar.setProgress(values[0]);
    }

    @Override
    protected void onPostExecute(Bitmap result) {
        mProgressBar.setVisibility(ProgressBar.INVISIBLE);
        mImageView.setImageBitmap(result);
    }

    private void sleep() {
        try {
            Thread.sleep(mDelay);
        } catch (InterruptedException e) {
            Log.e(TAG, e.toString());
        }
    }
}
```

Handler

Each Handler is associated with a Thread

One Thread can hand off work to another Thread by sending Messages & posting Runnables to a Handler associated with the other Thread

Handler

Runnable

- Contains an instance of the Runnable interface
- Sender implements response

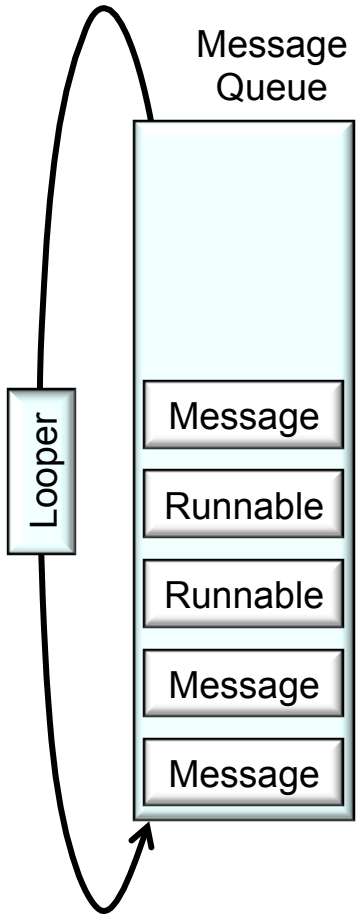
Message

- Can contain a message code, an object & integer arguments
- Handler implements response

Handler Architecture

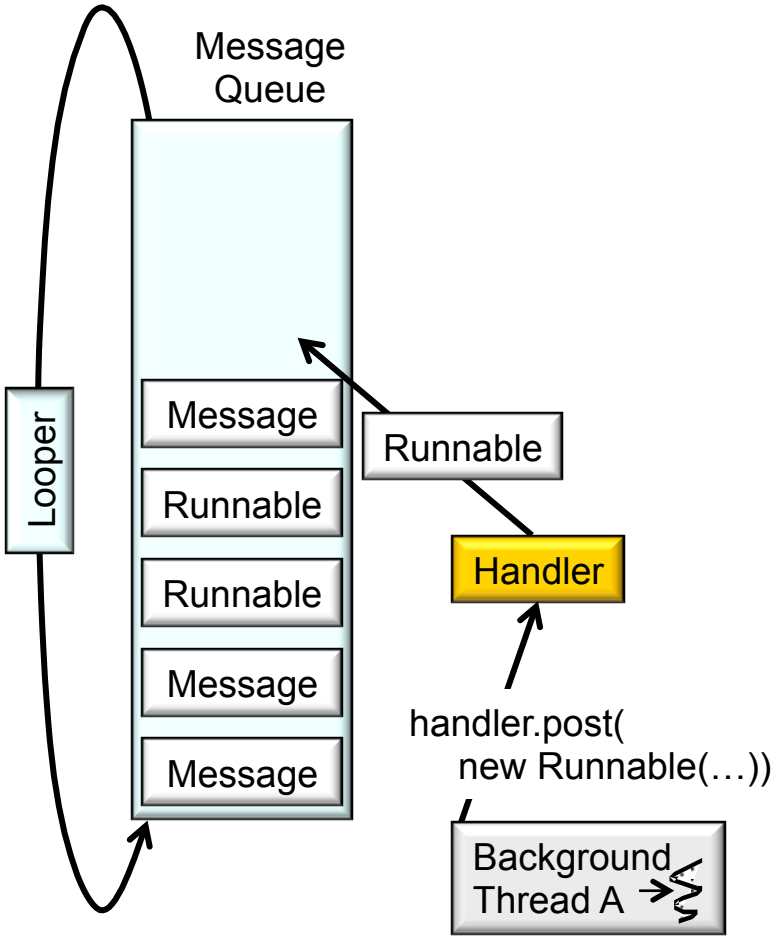
Each Android Thread is associated with a messageQueue & a Looper

A MessageQueue holds Messages and Runnables to be dispatched by the Looper



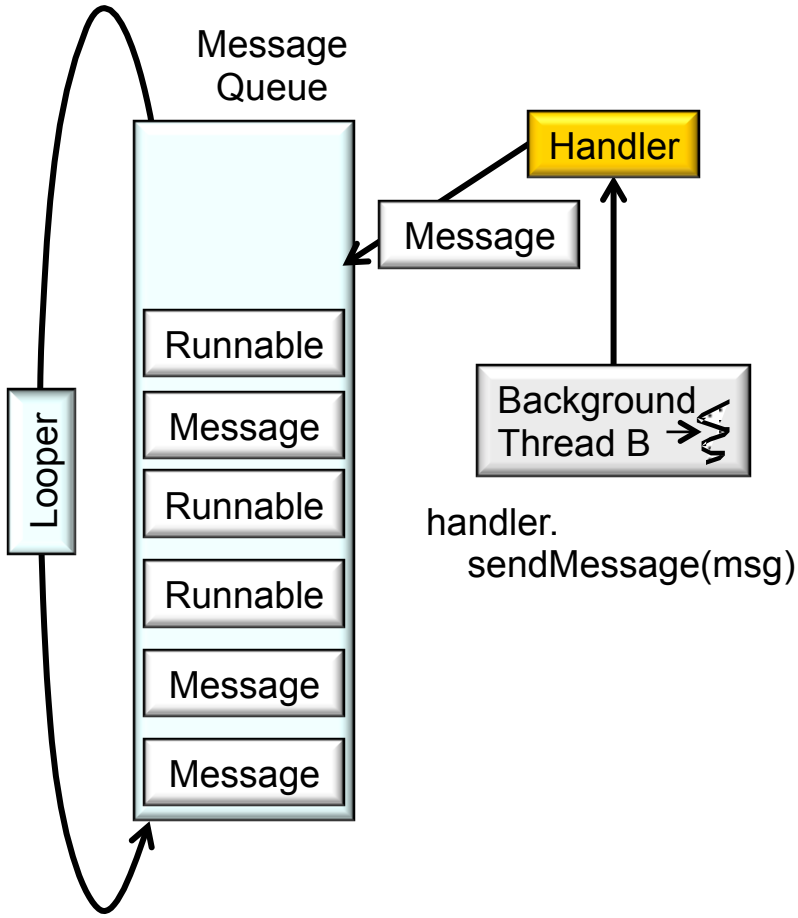
Handler Architecture

Add Runnables to MessageQueue by calling Handler's post() method



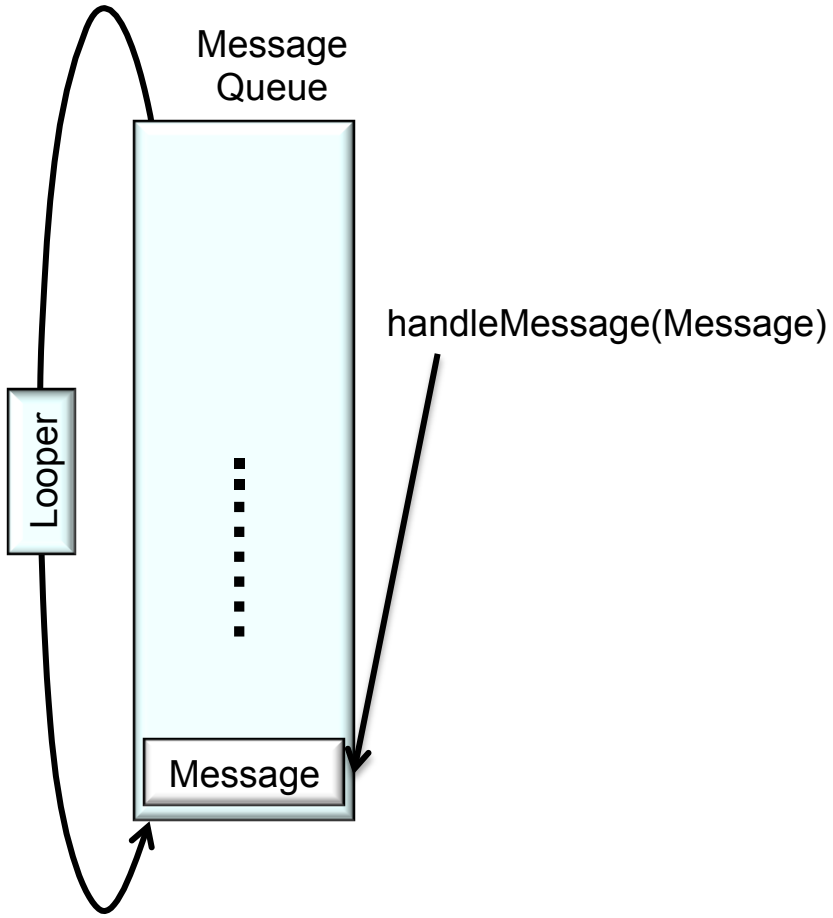
Handler Architecture

Add Messages to MessageQueue by calling Handler's sendMessage() method



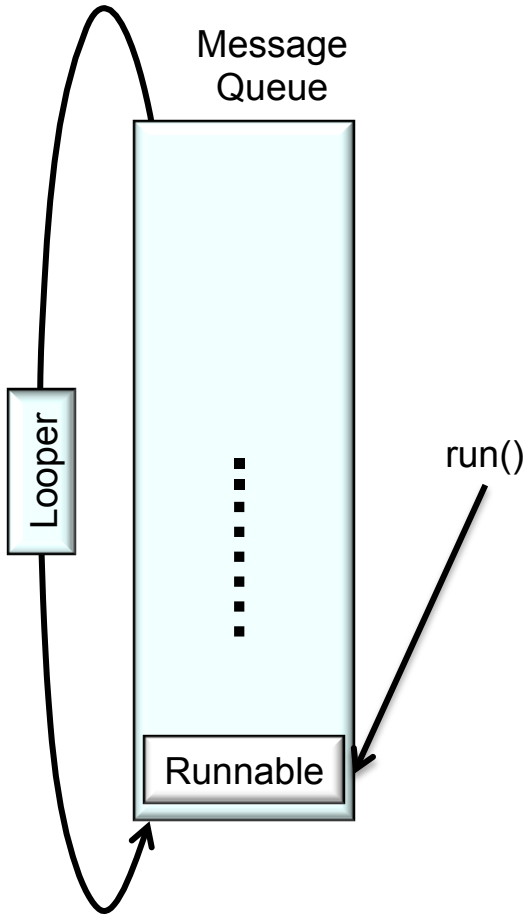
Handler Architecture

Looper dispatches Messages by calling the handler's handleMessage() method in the MessageQueue's Thread



Handler Architecture

Looper dispatches Runnables by calling their run() method in the MessageQueue's Thread



Runnables & Handlers

boolean post(Runnable r)

Add Runnable to the MessageQueue

boolean
postAtTime(Runnable r, long uptimeMillis)

Add Runnable to the MessageQueue. Run at a specific time (based on SystemClock.upTimeMillis())

boolean
postDelayed(Runnable r, long delayMillis)

Add Runnable to the message queue. Run after the specified amount of time elapses

Messages & Handlers

Create Message & set Message content

- Handler.obtainMessage()
- Message.obtain()

Message parameters include

- int arg1, arg2, what
- Object obj
- Bundle data

Many variants. See documentation

Messages & Handlers

sendMessage()

Queue Message now

sendMessageAtFrontOfQueue()

Insert Message now at front of queue

sendMessageAtTime()

Queue Message at the stated time

sendMessageDelayed()

Queue Message after delay

ThreadingHandlerMessages

```
static class UIHandler extends Handler {
    WeakReference<HandlerMessagesActivity> mParent;

    public UIHandler(WeakReference<HandlerMessagesActivity> parent) {
        mParent = parent;
    }

    @Override
    public void handleMessage(Message msg) {
        HandlerMessagesActivity parent = mParent.get();
        if (null != parent) {
            switch (msg.what) {
            case SET_PROGRESS_BAR_VISIBILITY: {
                parent.getProgressBar().setVisibility((Integer) msg.obj);
                break;
            }
            case PROGRESS_UPDATE: {
                parent.getProgressBar().setProgress((Integer) msg.obj);
                break;
            }
            case SET_BITMAP: {
                parent.getImageView().setImageBitmap((Bitmap) msg.obj);
                break;
            }
            }
        }
    }

}

Handler handler = new UIHandler(new WeakReference<HandlerMessagesActivity>(
        this));
```

ThreadingHandlerMessages

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mImageView = (ImageView) findViewById(R.id.imageView);
    mProgressBar = (ProgressBar) findViewById(R.id.progressBar);

    final Button button = (Button) findViewById(R.id.loadButton);
    button.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            new Thread(new LoadIconTask(R.drawable.painter, handler))
                    .start();
        }
    });

    final Button otherButton = (Button) findViewById(R.id.otherButton);
    otherButton.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            Toast.makeText(HandlerMessagesActivity.this, "I'm Working",
                    Toast.LENGTH_SHORT).show();
        }
    });

}
```

ThreadingHandlerMessages

```
private class LoadIconTask implements Runnable {
    private final int resId;
    private final Handler handler;

    LoadIconTask(int resId, Handler handler) {
        this.resId = resId;
        this.handler = handler;
    }

    public void run() {

        Message msg = handler.obtainMessage(SET_PROGRESS_BAR_VISIBILITY,
                ProgressBar.VISIBLE);
        handler.sendMessage(msg);

        final Bitmap tmp = BitmapFactory.decodeResource(getResources(),
                resId);

        for (int i = 1; i < 11; i++) {
            sleep();
            msg = handler.obtainMessage(PROGRESS_UPDATE, i * 10);
            handler.sendMessage(msg);
        }

        msg = handler.obtainMessage(SET_BITMAP, tmp);
        handler.sendMessage(msg);

        msg = handler.obtainMessage(SET_PROGRESS_BAR_VISIBILITY,
                ProgressBar.INVISIBLE);
        handler.sendMessage(msg);
    }

    private void sleep() {
        try {
            Thread.sleep(mDelay);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

ThreadingHandlerRunnable

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mImageView = (ImageView) findViewById(R.id.imageView);
    mProgressBar = (ProgressBar) findViewById(R.id.progressBar);

    final Button button = (Button) findViewById(R.id.loadButton);
    button.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            new Thread(new LoadIconTask(R.drawable.painter)).start();
        }
    });

    final Button otherButton = (Button) findViewById(R.id.otherButton);
    otherButton.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            Toast.makeText(HandlerRunnableActivity.this, "I'm Working",
                    Toast.LENGTH_SHORT).show();
        }
    });

}
```

ThreadingHandlerRunnable

```
private class LoadIconTask implements Runnable {
    int resId;

    LoadIconTask(int resId) {
        this.resId = resId;
    }

    public void run() {

        handler.post(new Runnable() {
            @Override
            public void run() {
                mProgressBar.setVisibility(ProgressBar.VISIBLE);
            }
        });

        mBitmap = BitmapFactory.decodeResource(getResources(), resId);

        // Simulating long-running operation

        for (int i = 1; i < 11; i++) {
            sleep();
            final int step = i;
            handler.post(new Runnable() {
                @Override
                public void run() {
                    mProgressBar.setProgress(step * 10);
                }
            });
        }

        handler.post(new Runnable() {
            @Override
            public void run() {
                mImageView.setImageBitmap(mBitmap);
            }
        });

        handler.post(new Runnable() {
            @Override
            public void run() {
                mProgressBar.setVisibility(ProgressBar.INVISIBLE);
            }
        });
    }
}
```

Next Time

Alarms