Android Concurrency & Synchronization: Part 6

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

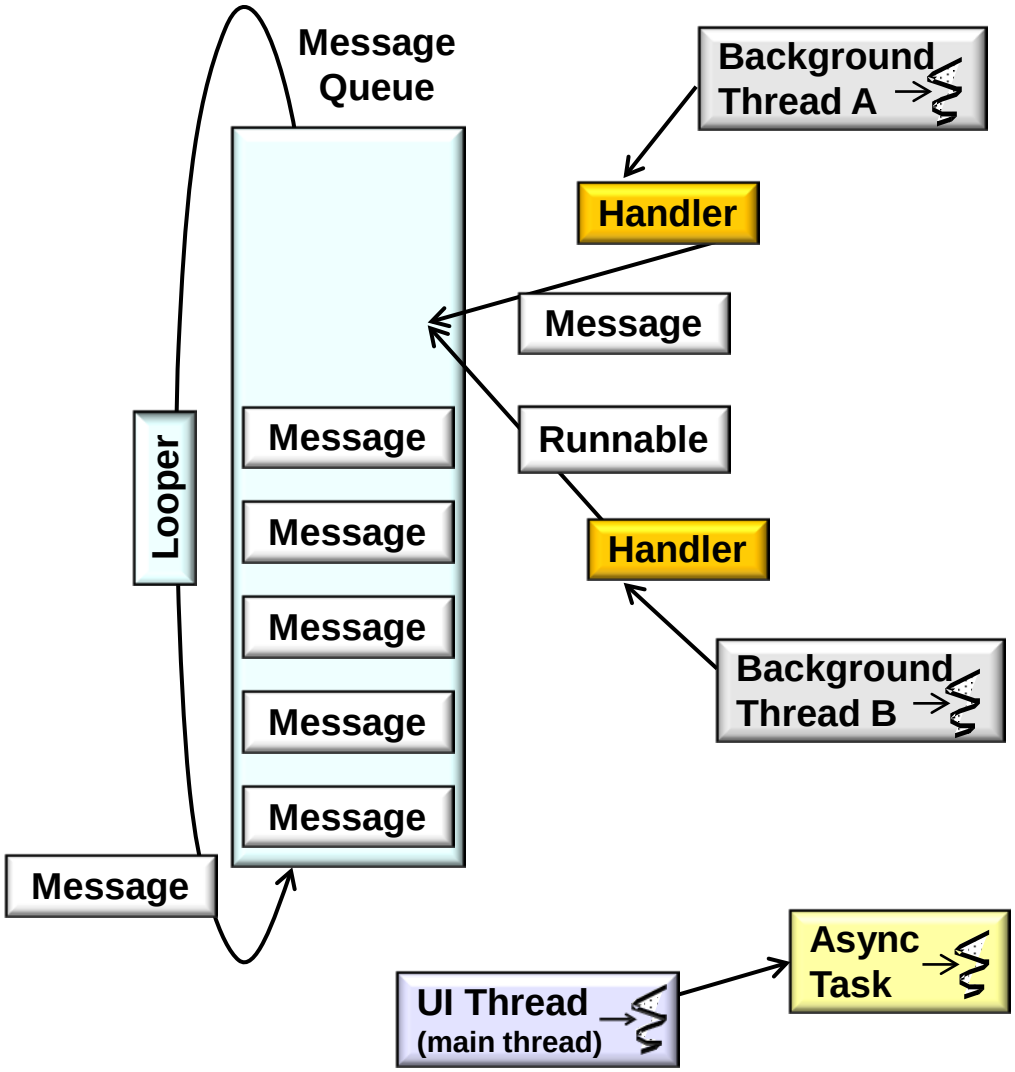
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

CS 282 Principles of Operating Systems II

Systems Programming for Android

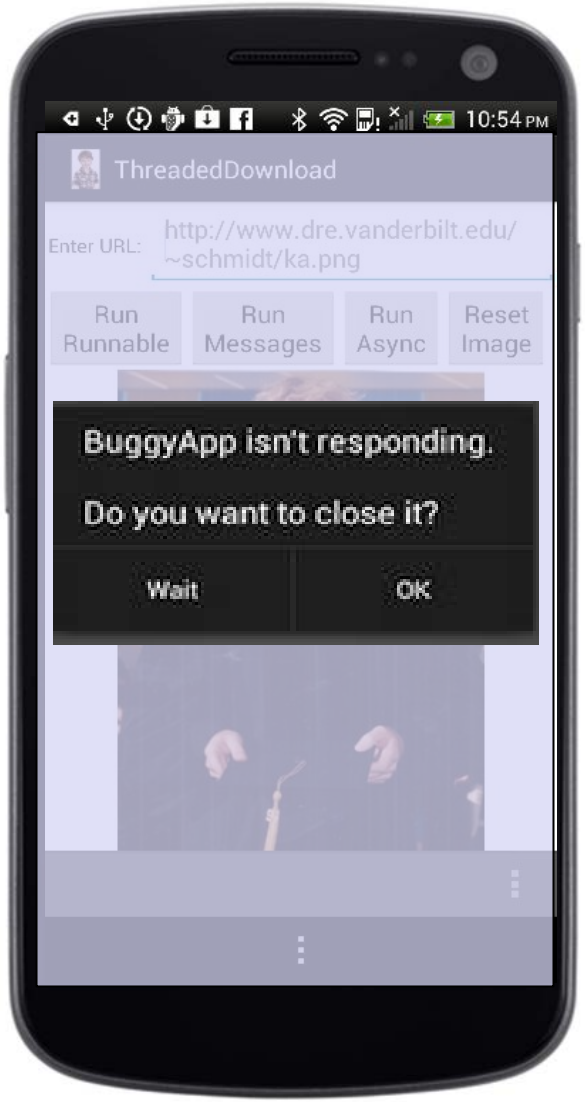
Learning Objectives in this Part of the Module

- Understand Android concurrency idioms & associated programming mechanisms



Motivating Android Concurrency Idioms

- Android's UI has several design constraints
  - An "Application Not Responding" (ANR) dialog is generated if app's UI Thread doesn't respond to user input within a short time



See developer.android.com/training/articles/perf-anr.html for more on ANRs

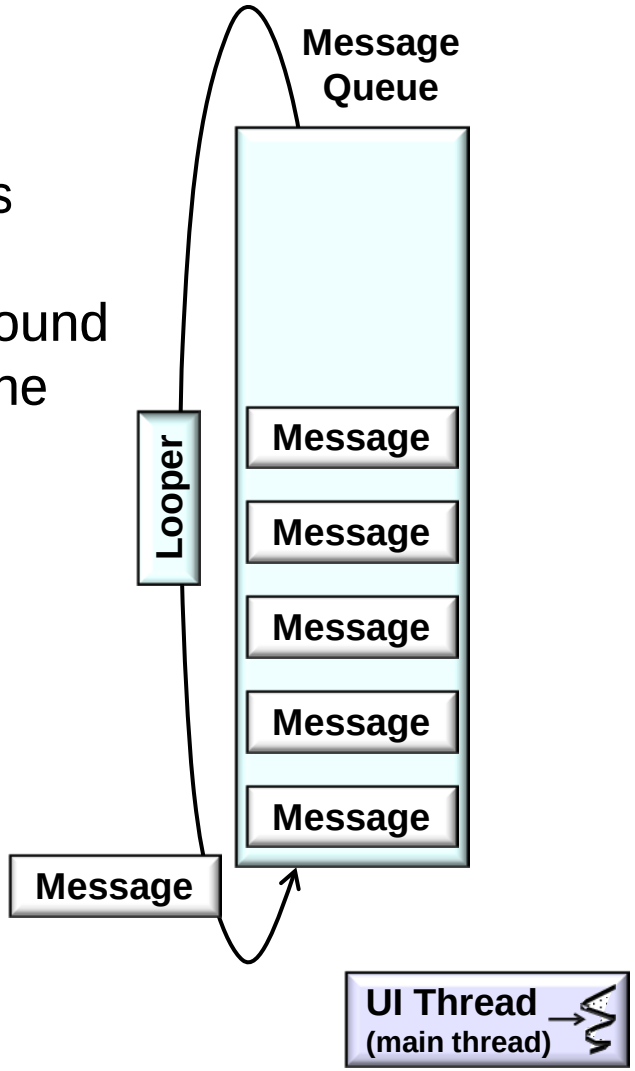
Motivating Android Concurrency Idioms

- Android's UI has several design constraints
  - An "Application Not Responding" (ANR) dialog is generated if app's UI Thread doesn't respond to user input within a short time
  - Non-UI Threads can't access widgets in the UI toolkit since it's not thread-safe

android-developers.blogspot.com/2009/05/painless-threading.html has more

Motivating Android Concurrency Idioms

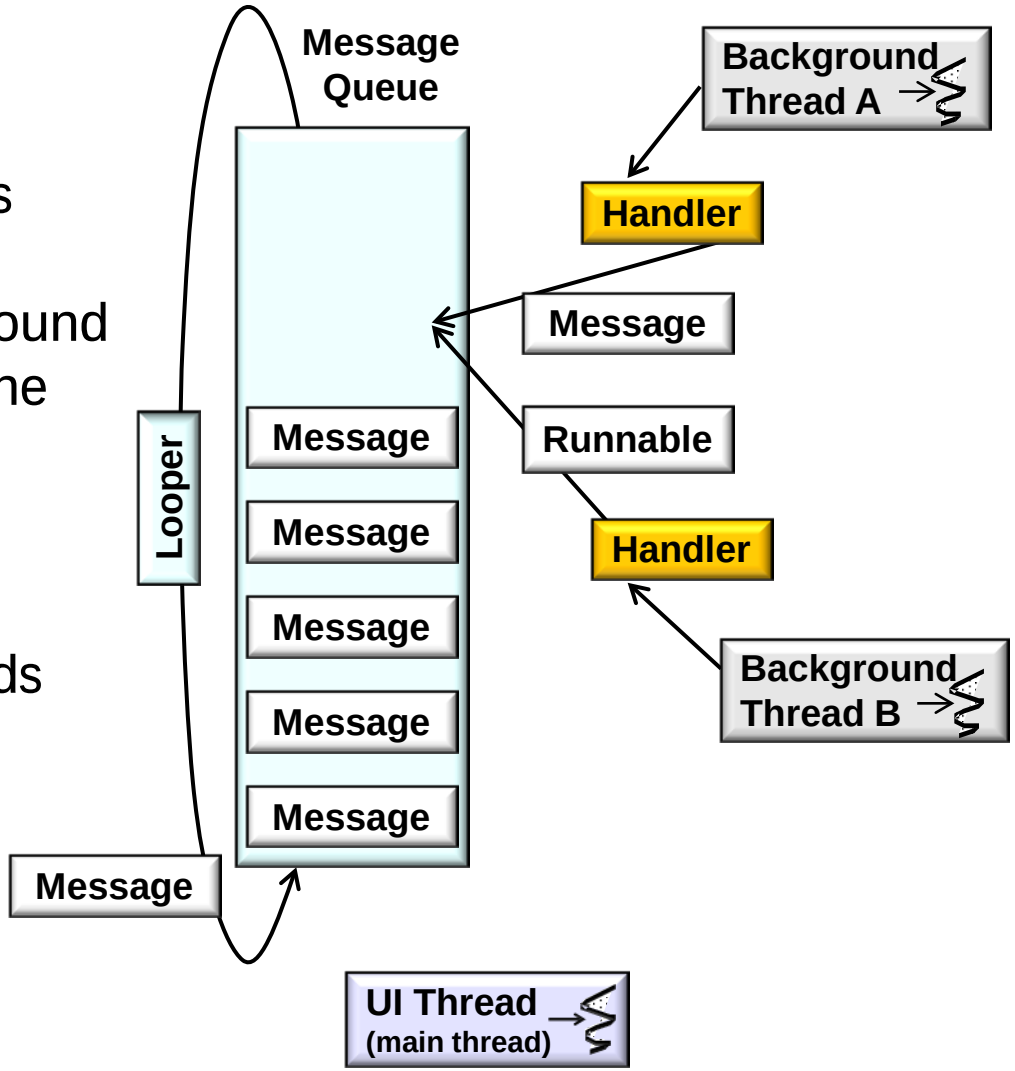
- Android's UI has several design constraints
- Android therefore supports various concurrency idioms for processing long-running operations in background thread(s) & communicating with the UI Thread



See developer.android.com/training/multiple-threads/communicate-ui.html

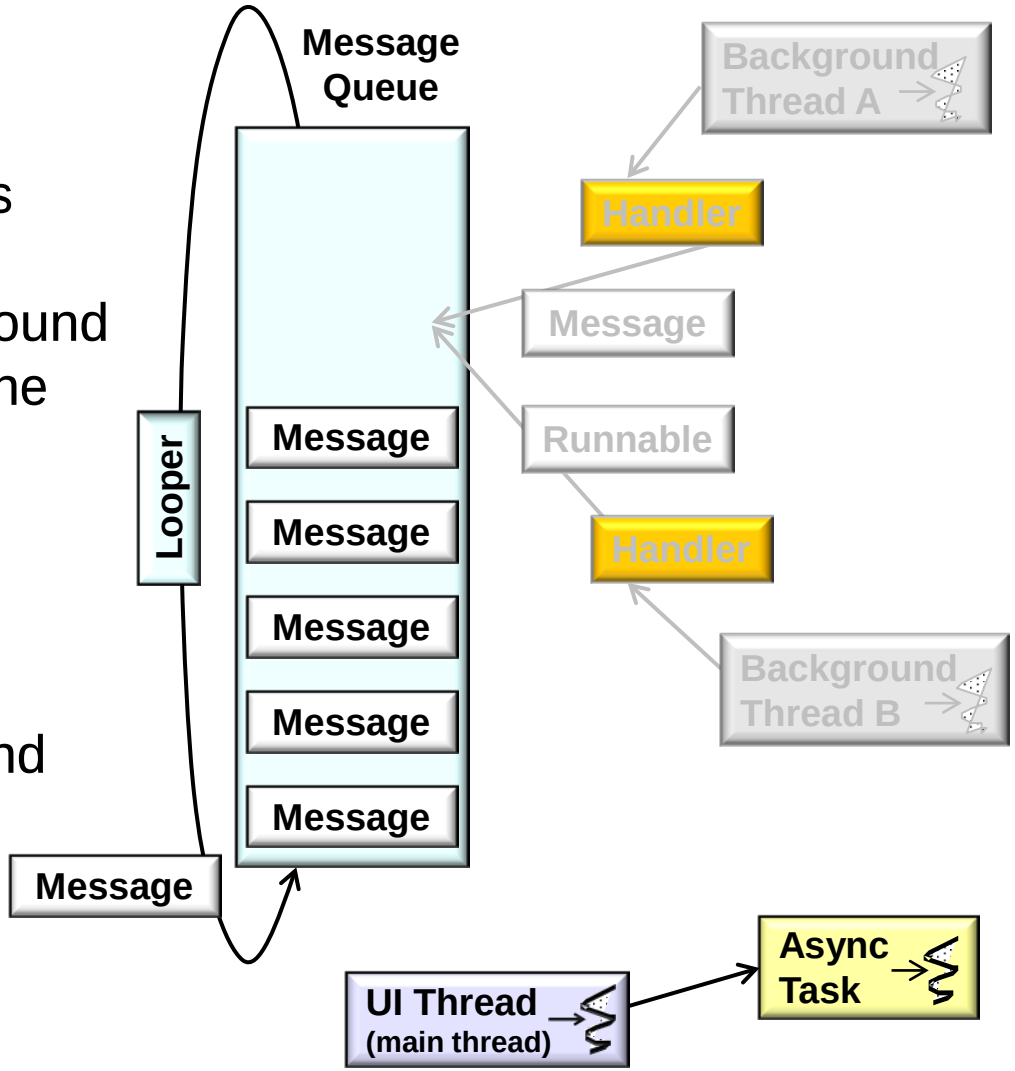
Motivating Android Concurrency Idioms

- Android's UI has several design constraints
- Android therefore supports various concurrency idioms for processing long-running operations in background thread(s) & communicating with the UI Thread
  - **Handlers, Messages, & Runnables**
    - Allows an app to spawn threads that perform background operations & publish results on the UI thread



Motivating Android Concurrency Idioms

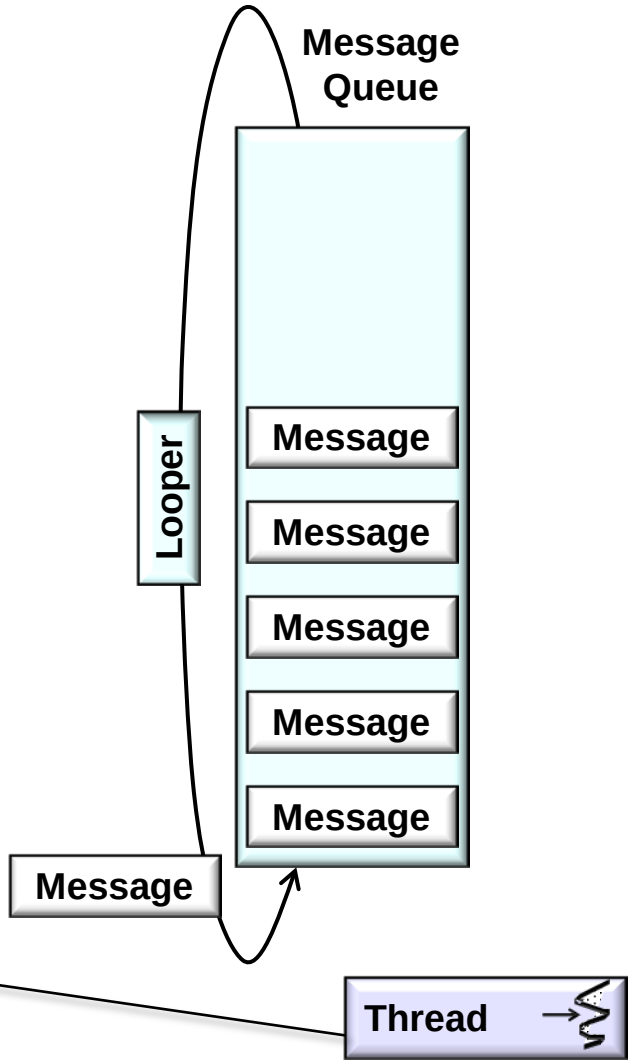
- Android's UI has several design constraints
- Android therefore supports various concurrency idioms for processing long-running operations in background thread(s) & communicating with the UI Thread
  - **Handlers, Messages, & Runnables**
  - **AsyncTask**
    - Allow an app to run background operations & publish results on the UI thread without manipulating threads or handlers



www.vogella.com/articles/AndroidBackgroundProcessing/article.html has more

The Android Looper Class

- A Looper provides a message queue to a thread
  - Only one Looper is allowed per Thread



The UI Thread has a Looper, but Loopers can also be used in non-UI threads

developer.android.com/reference/android/os/Looper.html has more info

The Android Looper Class

- A Looper provides a message queue to a thread
  - Only one Looper is allowed per Thread
  - The Looper.loop() method runs a Thread's main event loop, which waits for Messages & dispatches them to their Handlers

```
public class Looper {
  ...
  final MessageQueue mQueue;

  public static void loop() {
    ...

    for (;;) {
      Message msg =
        queue.next();
      ...

      msg.target.
        dispatchMessage(msg);
      ...
  }
  ...
```

The Android Looper Class

- A Looper provides a message queue to a thread
  - Only one Looper is allowed per Thread
  - The Looper.loop() method runs a Thread's main event loop, which waits for Messages & dispatches them to their Handlers

```
public class Looper {
  ...
  final MessageQueue mQueue;

  public static void loop() {
    ...

    for (;;) {
      Message msg =
        queue.next();
      ...

      msg.target.
        dispatchMessage(msg);
      ...
  }
  ...
```

This call can block

The Android Looper Class

- A Looper provides a message queue to a thread
  - Only one Looper is allowed per Thread
  - The Looper.loop() method runs a Thread's main event loop, which waits for Messages & dispatches them to their Handlers

```
public class Looper {
  ...
  final MessageQueue mQueue;

  public static void loop() {
    ...

    for (;;) {
      Message msg =
        queue.next();
      ...

      msg.target.
        dispatchMessage(msg);
      ...
  }
  ...
```

Note inversion of control

The Android Looper Class

- A Looper provides a message queue to a thread
  - Only one Looper is allowed per Thread
  - The Looper.loop() method runs a Thread's main event loop, which waits for Messages & dispatches them to their Handlers

```
public class Looper {
  ...

  public void prepare() {
    ...
  }

  public static void loop() {
    ...
  }

  public void quit() {
    ...
  }

  ...
```

frameworks/base/core/java/android/os/Looper.java has the source code

The Android Looper Class

- A Looper provides a message queue to a thread
- By default Threads don't have a message loop associated with them

```
public class Thread
       implements Runnable {
  public static Thread
    currentThread() {
    ...
  }

  public final void join() {
    ...
  }

  public void interrupt() {
    ...
  }

  public synchronized void start() {
    ...
  }
```

developer.android.com/reference/java/lang/Thread.html has more info

The Android Looper Class

- A Looper provides a message queue to a thread
- By default Threads don't have a message loop associated with them
  - To create one, call
    - prepare() in the thread that is to run the loop & then

```
class LooperThread extends Thread
{
  public Handler mHandler;

  public void run() {
    Looper.prepare();

    mHandler = new Handler() {
      public void handleMessage
                    (Message msg) {
        // process incoming msgs
    }};

    Looper.loop();
  }
```

developer.android.com/reference/android/os/Looper.html has more info

The Android Looper Class

- A Looper provides a message queue to a thread
- By default Threads don't have a message loop associated with them
  - To create one, call
    - prepare() in the thread that is to run the loop & then
    - Create Handlers to process incoming messages (need not go here)

```
class LooperThread extends Thread
{
  public Handler mHandler;

  public void run() {
    Looper.prepare();

    mHandler = new Handler() {
      public void handleMessage
                    (Message msg) {
        // process incoming msgs
    }};

    Looper.loop();
  }
```

developer.android.com/reference/android/os/Looper.html has more info

The Android Looper Class

- A Looper provides a message queue to a thread
- By default Threads don't have a message loop associated with them
  - To create one, call
    - prepare() in the thread that is to run the loop & then
    - Create Handlers to process incoming messages (need not go here)
    - loop() to have it process messages until the loop is stopped

```
class LooperThread extends Thread
{
  public Handler mHandler;

  public void run() {
    Looper.prepare();

    mHandler = new Handler() {
      public void handleMessage
                    (Message msg) {
        // process incoming msgs
    }};

    Looper.loop();
  }
```

developer.android.com/reference/android/os/Looper.html has more info

The Android Looper Class

- A Looper provides a message queue to a thread
- By default Threads don't have a message loop associated with them
- HandlerThread is a helper class for starting a new Thread that automatically contains a Looper

```
class HandlerThread extends Thread {
  Looper mLooper;
  ...

  public void run() {
    Looper.prepare();
    synchronized (this) {
      mLooper = Looper.myLooper();
      ...
    }
    ...
    onLooperPrepared();
    Looper.loop();
    ...

  protected void onLooperPrepared() {
  }
}
```

Note the use of the Template Method pattern to handle fixed steps in the algorithm

The Android Looper Class

- A Looper provides a message queue to a thread
- By default Threads don't have a message loop associated with them
- HandlerThread is a helper class for starting a new Thread that automatically contains a Looper

```
class HandlerThread extends Thread {
  Looper mLooper;
  ...

  public void run() {
    Looper.prepare();
    synchronized (this) {
      mLooper = Looper.myLooper();
      ...
    }
    ...
    onLooperPrepared();
    Looper.loop();
    ...

  protected void onLooperPrepared() {
  }
}
```

This hook method enables subclasses to create Handlers

The Android Looper Class

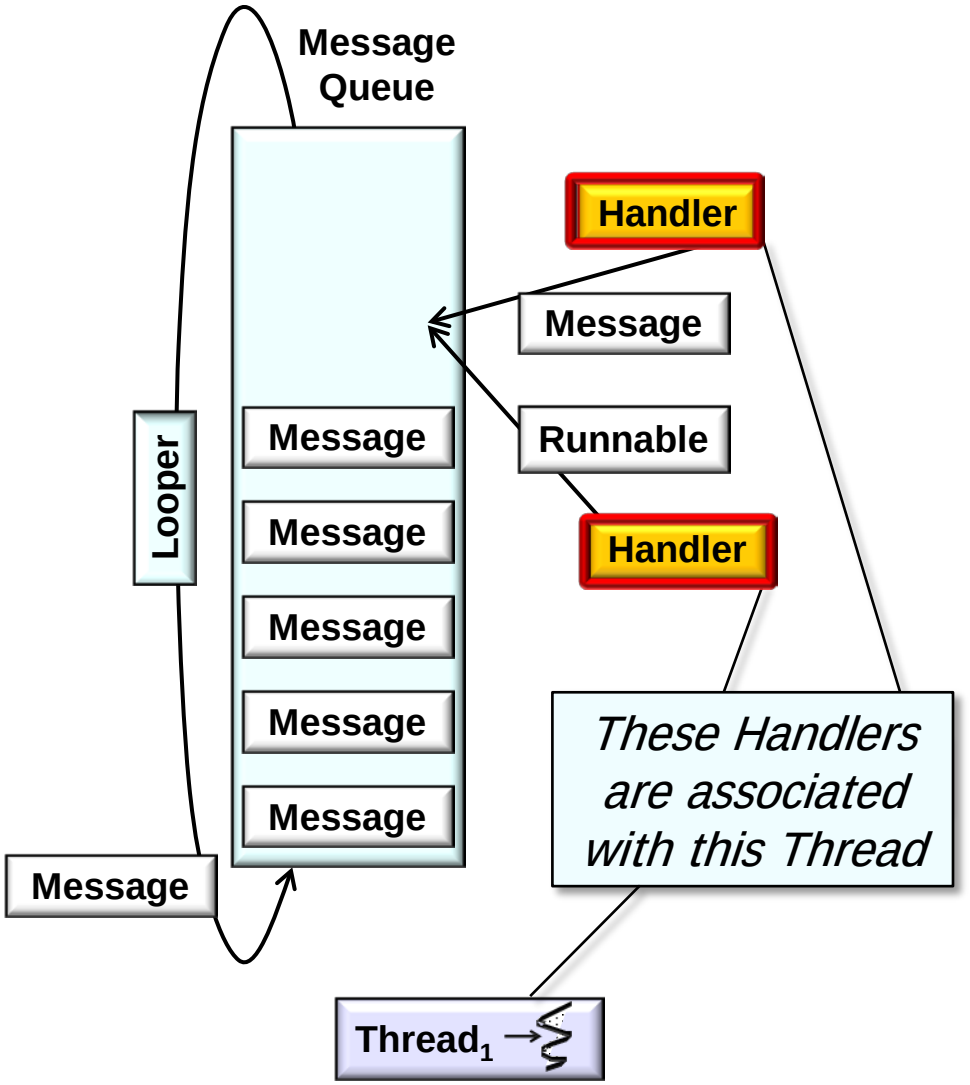
- A Looper provides a message queue to a thread
- By default Threads don't have a message loop associated with them
- HandlerThread is a helper class for starting a new Thread that automatically contains a Looper
  - The start() method must still be called by client code to launch the thread

```
class HandlerThread extends Thread {
  Looper mLooper;
  ...

  public void run() {
    Looper.prepare();
    synchronized (this) {
      mLooper = Looper.myLooper();
      ...
    }
    ...
    onLooperPrepared();
    Looper.loop();
    ...

  protected void onLooperPrepared() {
  }
}
```

frameworks/base/core/java/android/os/HandlerThread.java has the source code

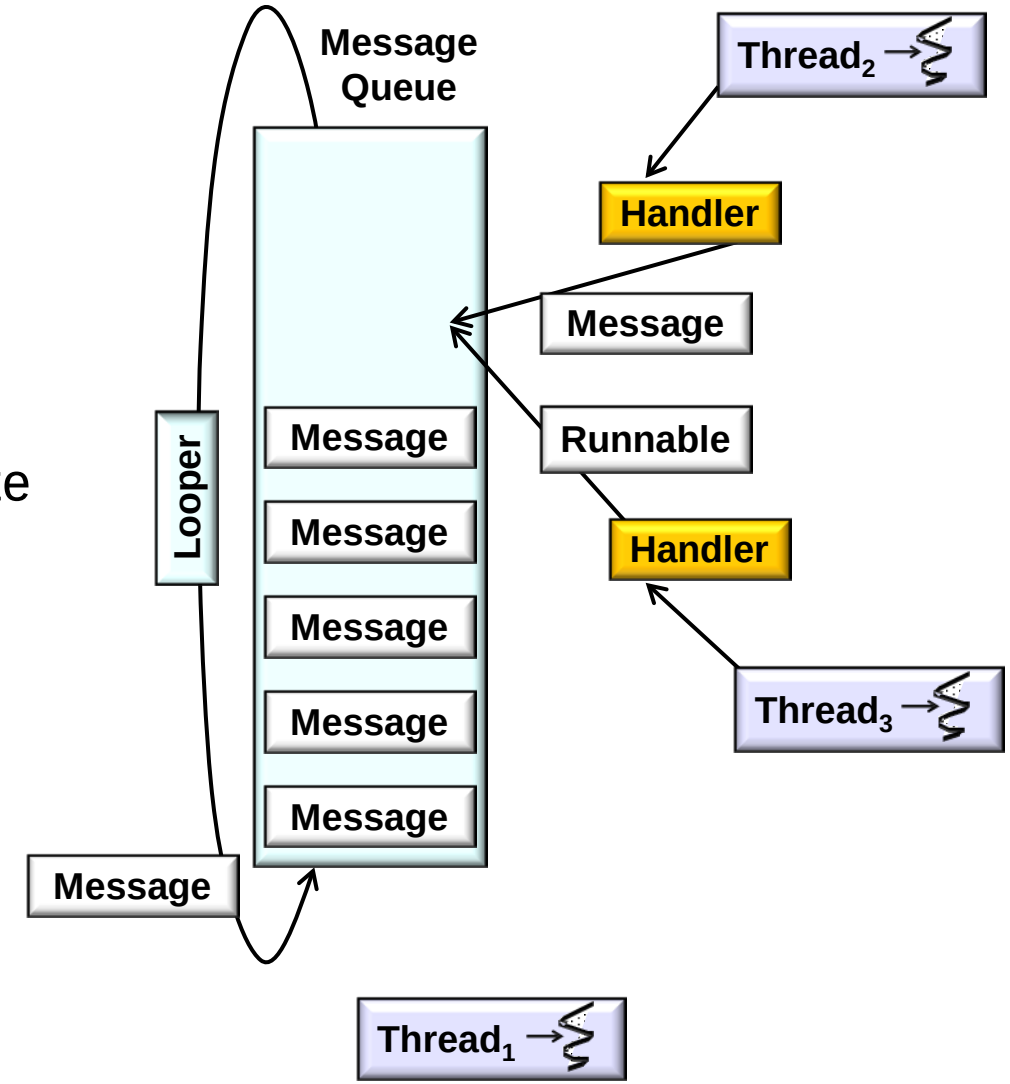
The Android Handler Class

- Most interaction with a message loop is through Handlers
  - A Handler allows sending & processing of Message & Runnable objects associated with a Thread's MessageQueue



The Android Handler Class

- Most interaction with a message loop is through Handlers
  - A Handler allows sending & processing of Message & Runnable objects associated with a Thread's MessageQueue
  - Other Threads can communicate by exchanging Messages & Runnables via a Thread's Handler(s)



developer.android.com/reference/android/os/Handler.html has more info

The Android Handler Class

- Most interaction with a message loop is through Handlers
- Each Handler object is associated with a single Thread & that Thread's MessageQueue

```
public class Handler {
  ...
  public void handleMessage
                (Message msg) {
  }
```

Subclasses must override this hook method to process messages

The Android Handler Class

- Most interaction with a message loop is through Handlers
- Each Handler object is associated with a single Thread & that Thread's MessageQueue
  - When you create a new Handler, it is bound to the Looper Thread (& its MessageQueue) of the Thread where it is created

```
public class Handler {
  ...
  public void handleMessage
                (Message msg) {
  }

  public Handler() {
    mLooper = Looper.myLooper();
    if (mLooper == null)
      throw new RuntimeException(
        "Can't create handler
         inside thread that hasn't
         called Looper.prepare()");

    mQueue = mLooper.mQueue;
    ...
```

Handler constructor ensures that the object is used within an initialized Looper

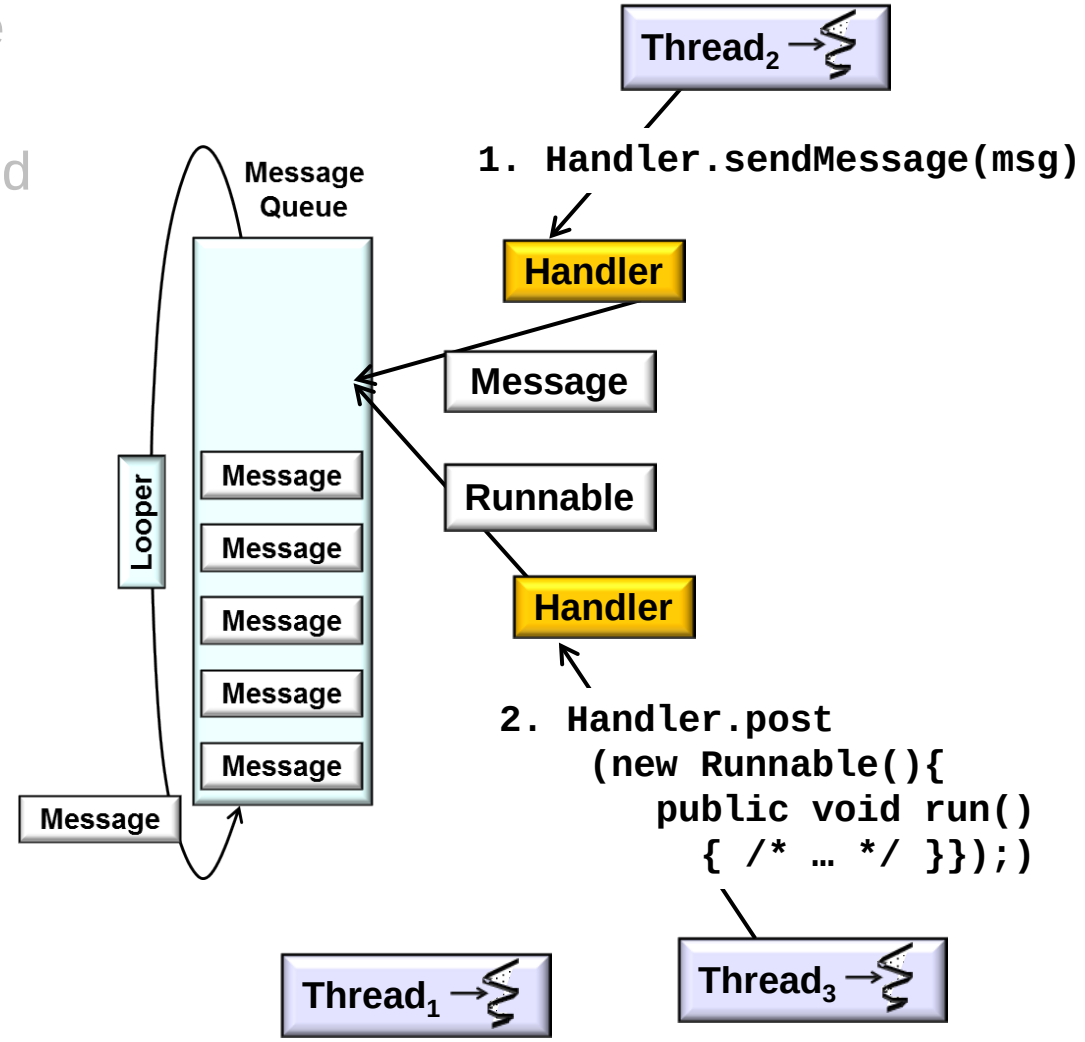
The Android Handler Class

- Most interaction with a message loop is through Handlers
- Each Handler object is associated with a single Thread & that Thread's MessageQueue
  - When you create a new Handler, it is bound to the Looper Thread (& its MessageQueue) of the Thread where it is created
  - From that point on, it will deliver Messages and Runnables to that Looper Thread's MessageQueue & execute them as they come out of the queue

```
public class Looper {
  ...
  final MessageQueue mQueue;

  public static void loop() {
    ...

    for (;;) {
      Message msg =
        queue.next();
      ...

      msg.target.
        dispatchMessage(msg);
      ...
  }
  ...
```

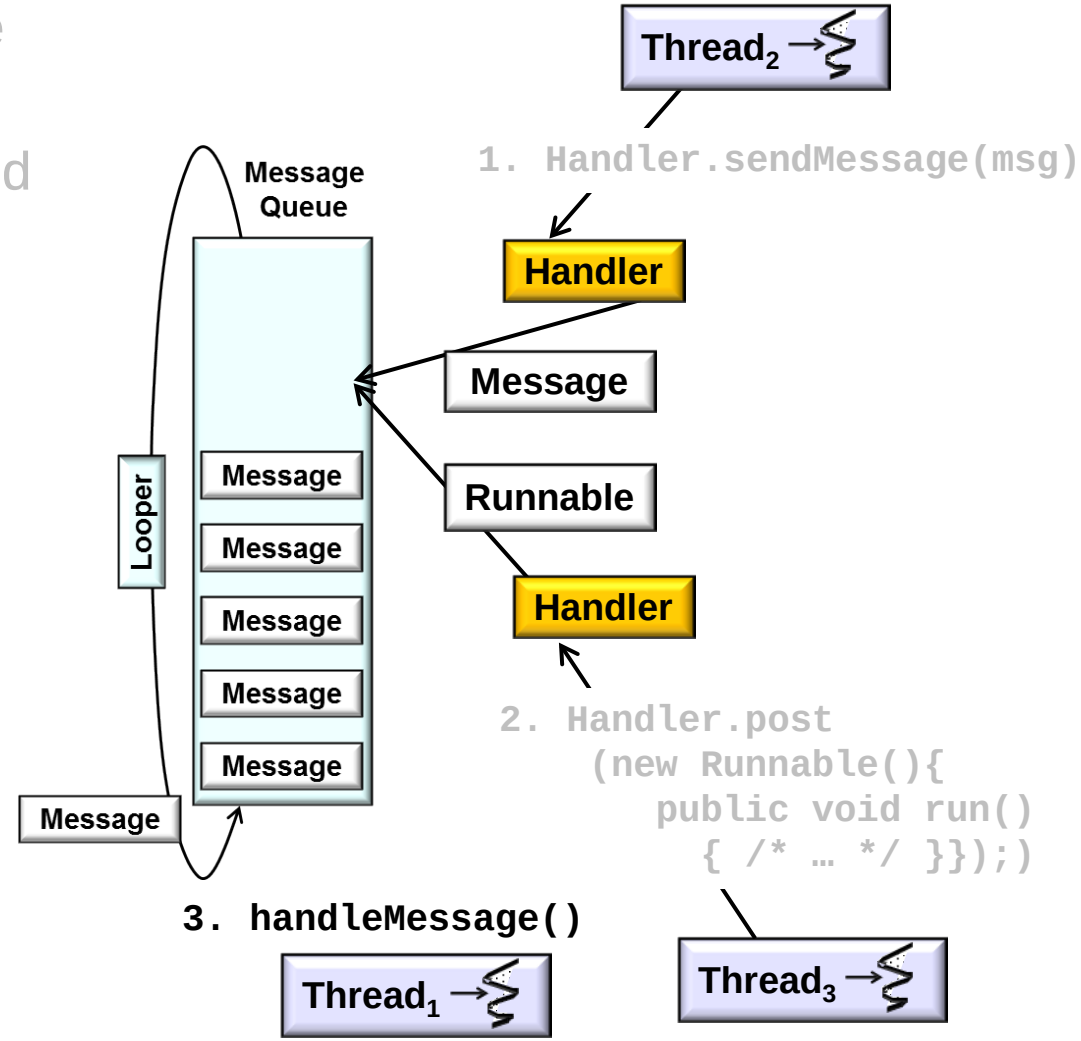
The Android Handler Class

- Most interaction with a message loop is through Handlers
- Each Handler object is associated with a single Thread & that Thread's MessageQueue
- Capabilities of a Handler
  - Sends Messages & posts Runnables to a Thread
    - Thread's MessageQueue enqueues/schedules them for future execution



The Android Handler Class

- Most interaction with a message loop is through Handlers
- Each Handler object is associated with a single Thread & that Thread's MessageQueue
- Capabilities of a Handler
  - Sends Messages & posts Runnables to a Thread
  - Implements thread-safe processing for Messages
    - In current Thread or different Thread



frameworks/base/core/java/android/os/Handler.java has source code

The Android Handler Class

- Most interaction with a message loop is through Handlers
- Each Handler object is associated with a single Thread & that Thread's MessageQueue
- Capabilities of a Handler
  - Sends Messages & posts Runnables to a Thread
  - Implements thread-safe processing for Messages
  - Handler methods associated with Runnables

`boolean post(Runnable r)`

- Add Runnable to MessageQueue

`boolean postAtTime(Runnable r, long uptimeMillis)`

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

`boolean postDelayed(Runnable r, long delayMillis)`

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

developer.android.com/reference/android/os/Handler.html has more info

The Android Handler Class

- Most interaction with a message loop is through Handlers
- Each Handler object is associated with a single Thread & that Thread's MessageQueue
- Capabilities of a Handler
  - Sends Messages & posts Runnables to a Thread
  - Implements thread-safe processing for Messages
  - Handler methods associated with Runnables
  - Handler methods associated with Messages

`boolean sendMessage(Message msg)`

- Puts msg at end of queue immediately

`boolean sendMessageAtFrontOfQueue (Message msg)`

- Puts msg at front of queue immediately

`boolean sendMessageAtTime (Message msg, long uptimeMillis)`

- Puts msg on queue at stated time

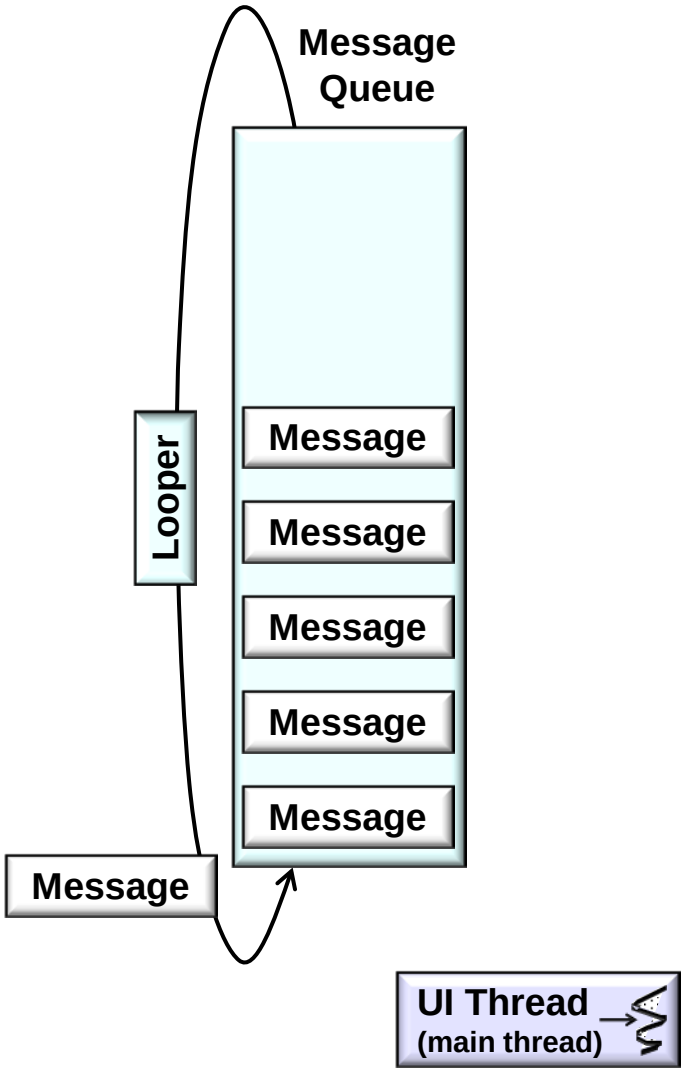
`boolean sendMessageDelayed (Message msg, long delayMillis)`

- Puts msg after delay time has passed

developer.android.com/reference/android/os/Handler.html has more info

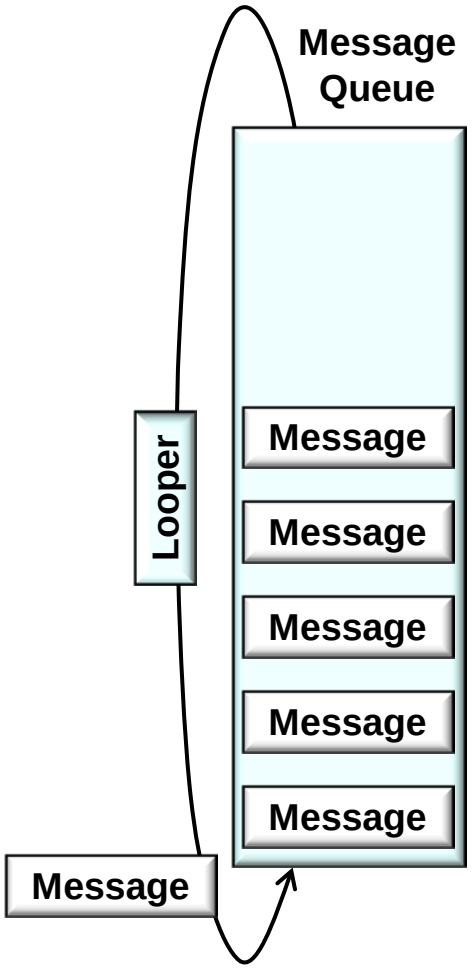
Summary

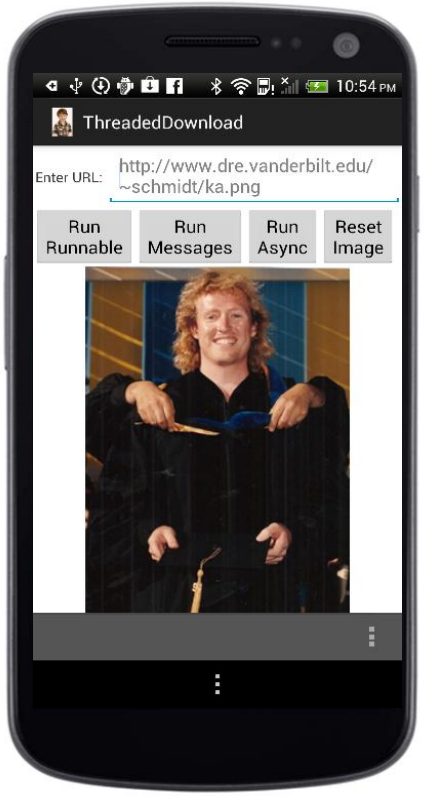
- Android apps have a UI Thread
  - The UI Thread is a Looper



Summary

- Android apps have a UI Thread
- App components in the same process use the same UI Thread
  - User interaction, system callbacks, & lifecycle methods are handled in the UI Thread

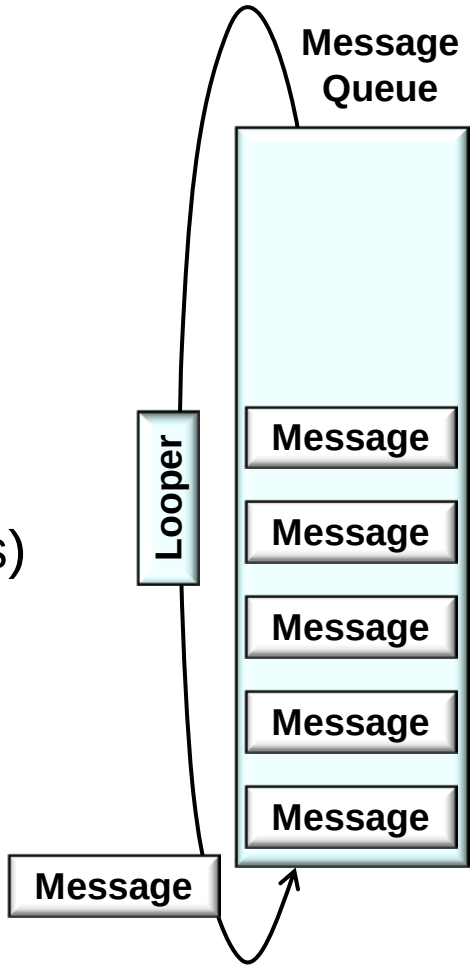


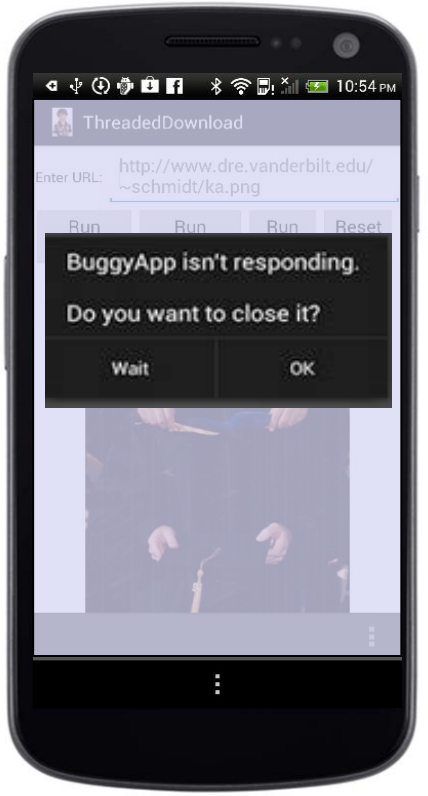


UI Thread
(main thread)

Summary

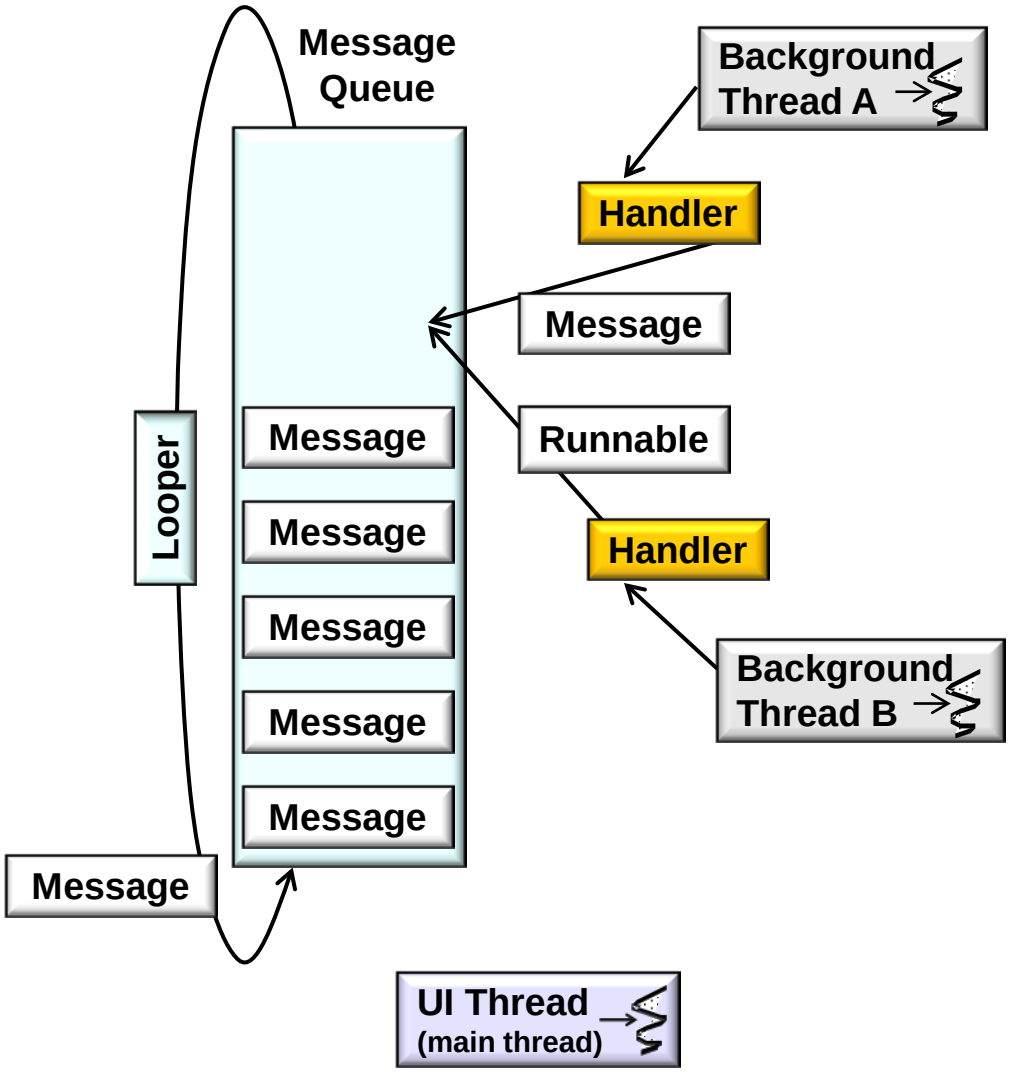
- Android apps have a UI Thread
- App components in the same process use the same UI Thread
- Don't access widgets in the UI toolkit from non-UI Thread or block the UI Thread
  - Long-running operations should execute in background Thread(s)





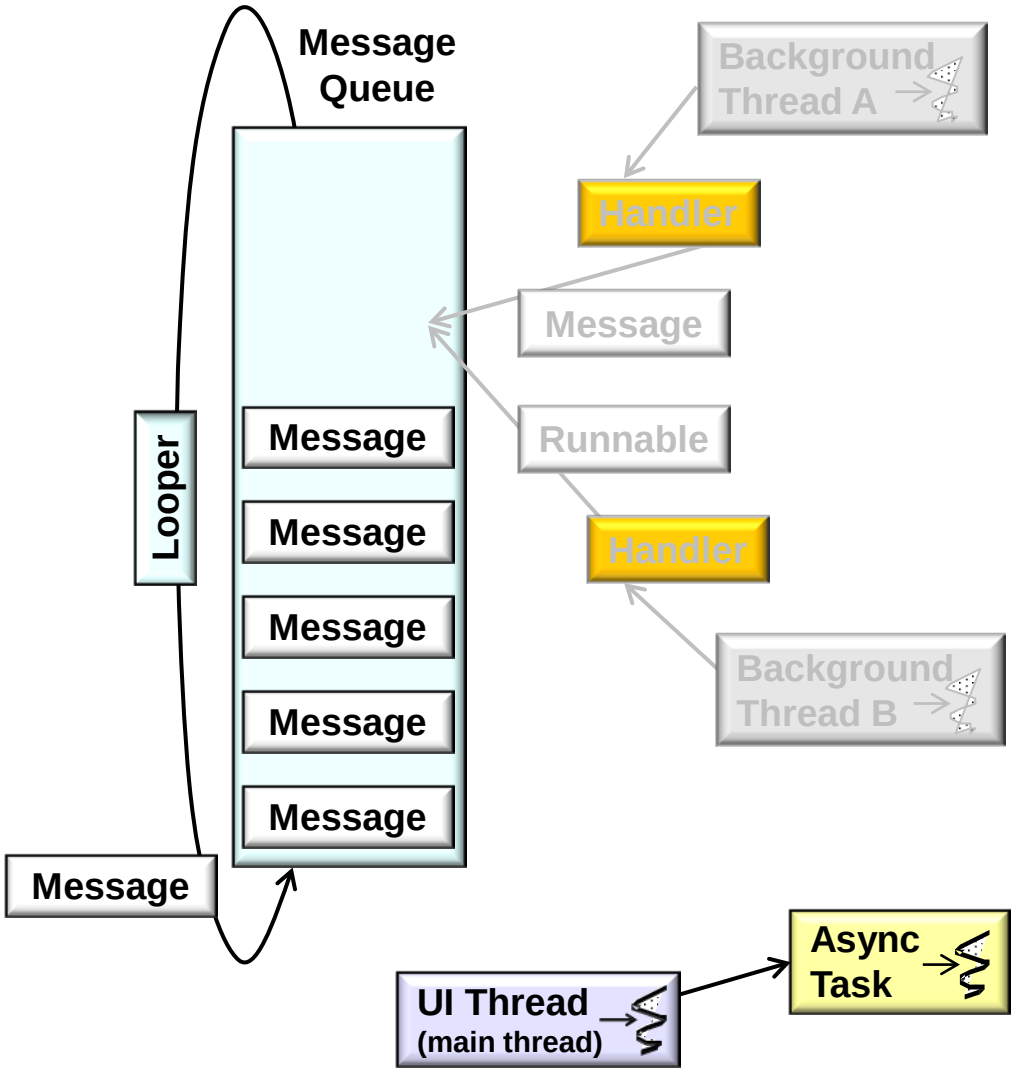
Summary

- Android apps have a UI Thread
- App components in the same process use the same UI Thread
- Don't access widgets in the UI toolkit from non-UI Thread or block the UI Thread
- UI & background threads will need to communicate via
  - Sending Messages or posting Runnables to the Looper Thread's MessageQueue



Summary

- Android apps have a UI Thread
- App components in the same process use the same UI Thread
- Don't access widgets in the UI toolkit from non-UI Thread or block the UI Thread
- UI & background threads will need to communicate via
  - Sending Messages or posting Runnables to the Looper Thread's MessageQueue
  - Executing operations in the background using AsyncTask



Android Concurrency & Synchronization: Part 7

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

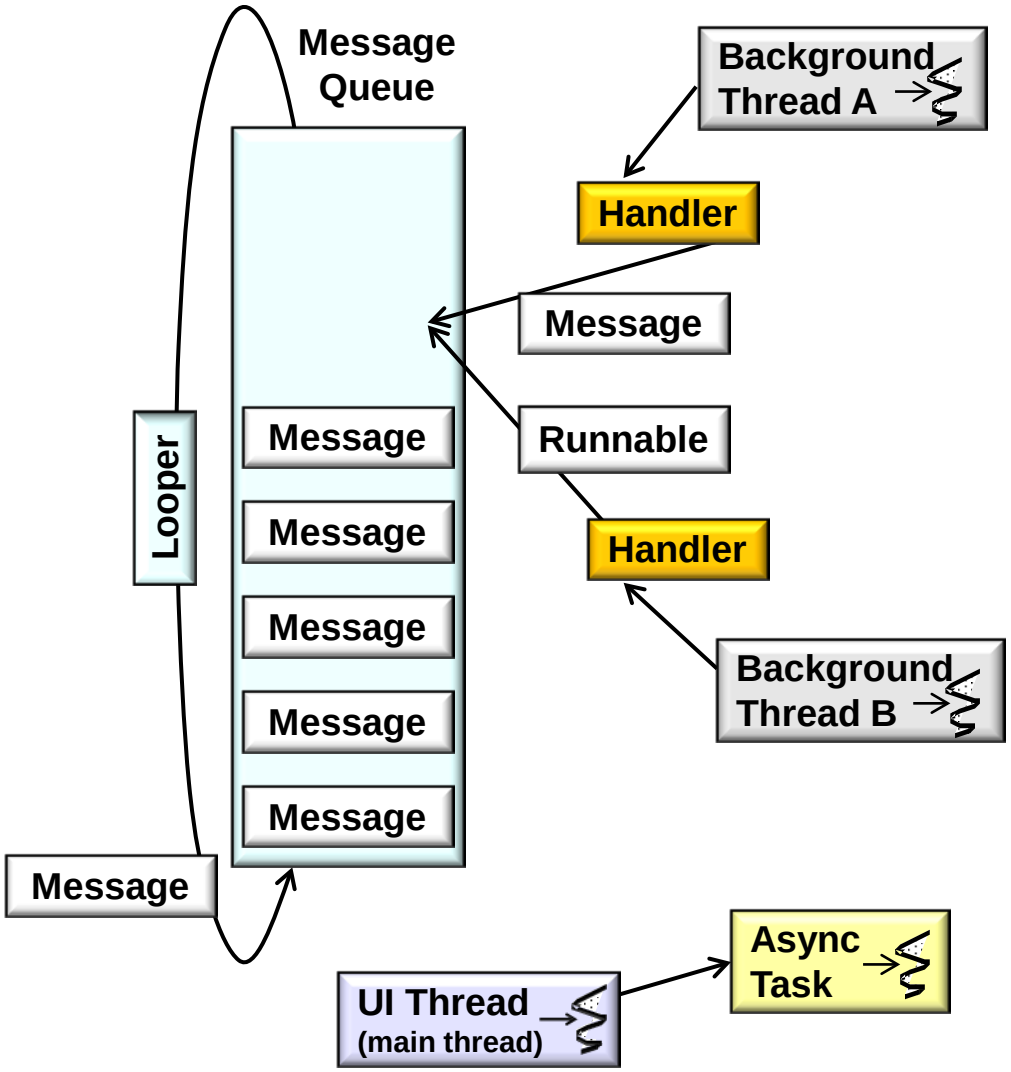
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

CS 282 Principles of Operating Systems II

Systems Programming for Android

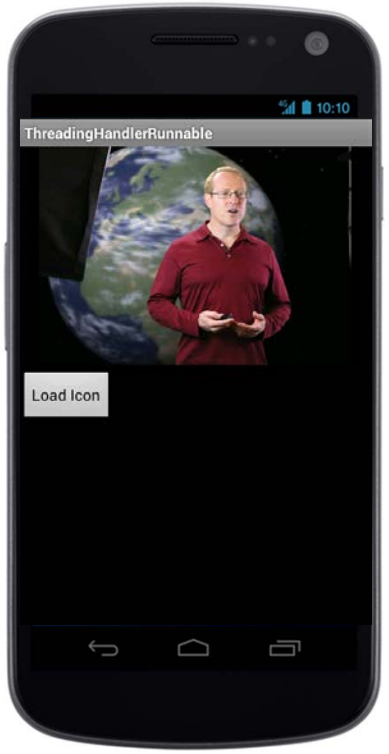
Learning Objectives in this Part of the Module

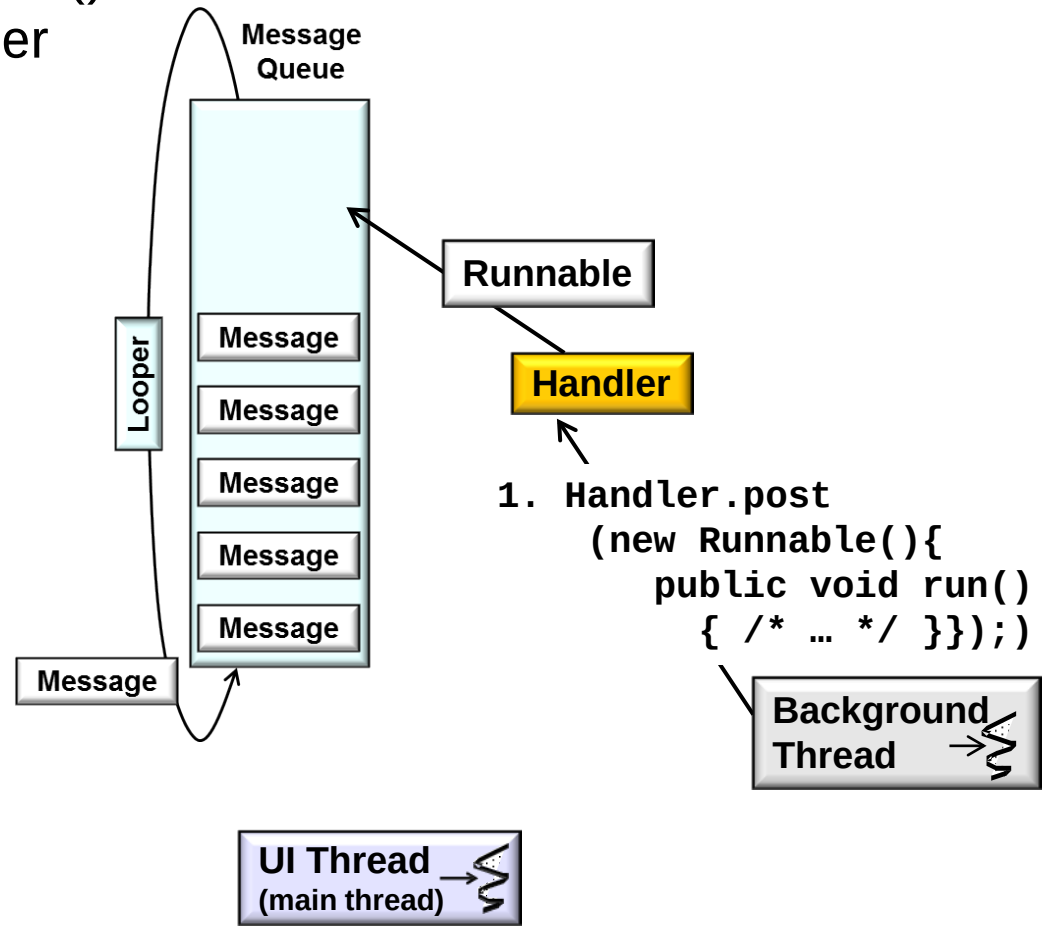
- Understand how to program with the Android concurrency idioms
  - Handlers & Runnables
  - Handlers & Messages
  - AsyncTask



Programming with the Handler & Runnables

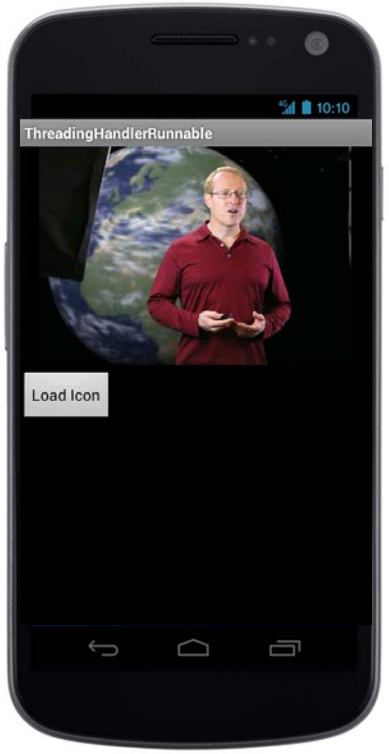
- Create a Runnable, override its run() hook method, & pass to a Handler

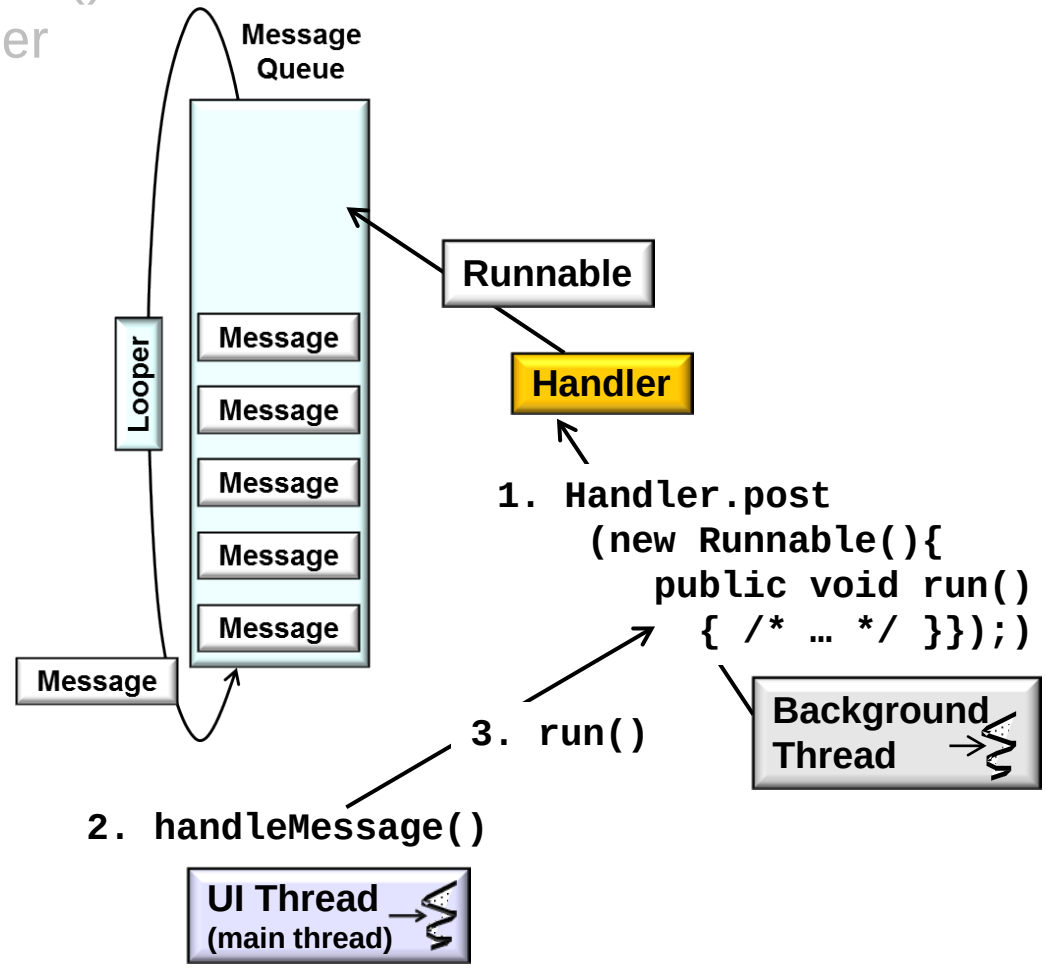




Programming with the Handler & Runnables

- Create a Runnable, override its run() hook method, & pass to a Handler
- Looper framework calls run() method in the UI Thread





Example of Runnables & Handlers

```
public class SimpleThreadingExample extends Activity {
   private ImageView iview;

   private Handler h = new Handler();
   public void onCreate(Bundle savedInstanceState) {
      ...
      iview = ...
      final Button = ...
      button.setOnClickListener(new OnClickListener() {
         public void onClick(View v) {
            new Thread(new
                  LoadIcon(R.drawable.icon)).start();
         }
      });
   }
   ...
```

Create new Handler in UI Thread

Create/start a new thread when user clicks a button

Pass the resource ID of the icon

This code runs in the UI Thread

Example of Runnables & Handlers

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

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

   public void run()
      final Bitmap tmp =
        BitmapFactory.decodeResource(getResources(),
                                     resId);
      h.post(new Runnable() {
          public void run() {
              iview.setImageBitmap(tmp);
          }
       });
   }
...
```

Cache resource ID

Convert resource ID to bitmap

Create a new Runnable & post it to the UI Thread via the Handler

This code runs in a background thread

Posting Runnables on UI thread

```
public class SimpleThreadingExample extends Activity {
  private Bitmap bitmap;
  public void onCreate(Bundle savedInstanceState) {
    ...
    final ImageView iview = ...; final Button b = ...;
    b.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        new Thread(new Runnable() {
          public void run() {
            Bitmap = ...
            iview.post(new Runnable() {
              public void run() {
                iview.setImageBitmap(bitmap);}
              });
          }
        }).start();
  ...
```

Create a new Runnable & post it to the UI Thread via the ImageView

developer.android.com/reference/android/view/View.html#post(java.lang.Runnable)

Posting Runnables on UI thread

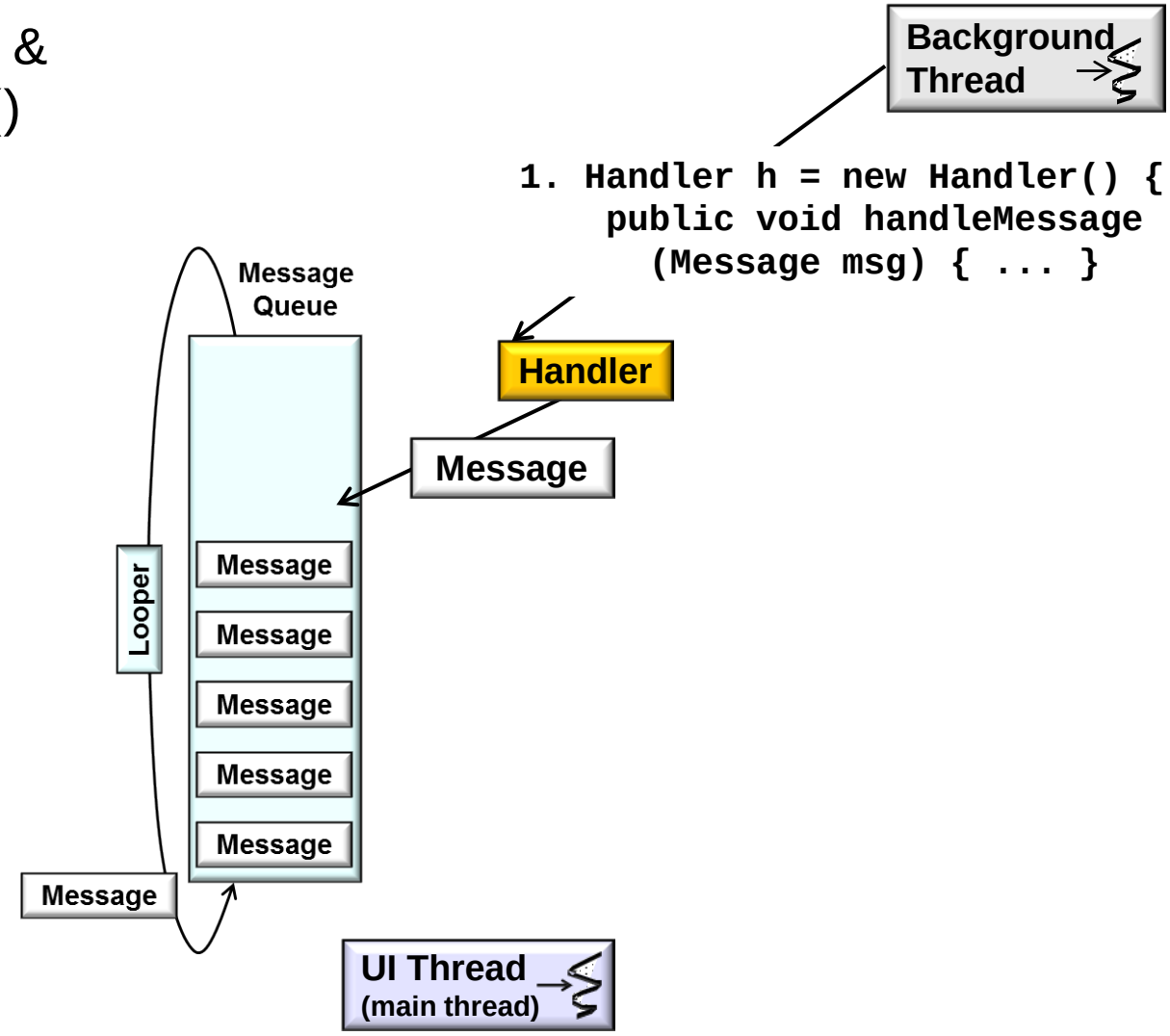
```
public class SimpleThreadingExample extends Activity {
  private Bitmap bitmap;
  public void onCreate(Bundle savedInstanceState) {
    ...
    final ImageView iview = ...; final Button b = ...;
    b.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        new Thread(new Runnable() {
          public void run() {
            Bitmap = ...
            SimpleThreadingExample.this
            .runOnUiThread(new Runnable() {
               public void run() {
                 iview.setImageBitmap(bitmap);}
               });}
        }).start();
```

Create a new Runnable & post it to the UI Thread via the Activity

developer.android.com/reference/android/app/Activity.html#runOnUiThread(java.lang.Runnable)

Programming with the Handler & Messages

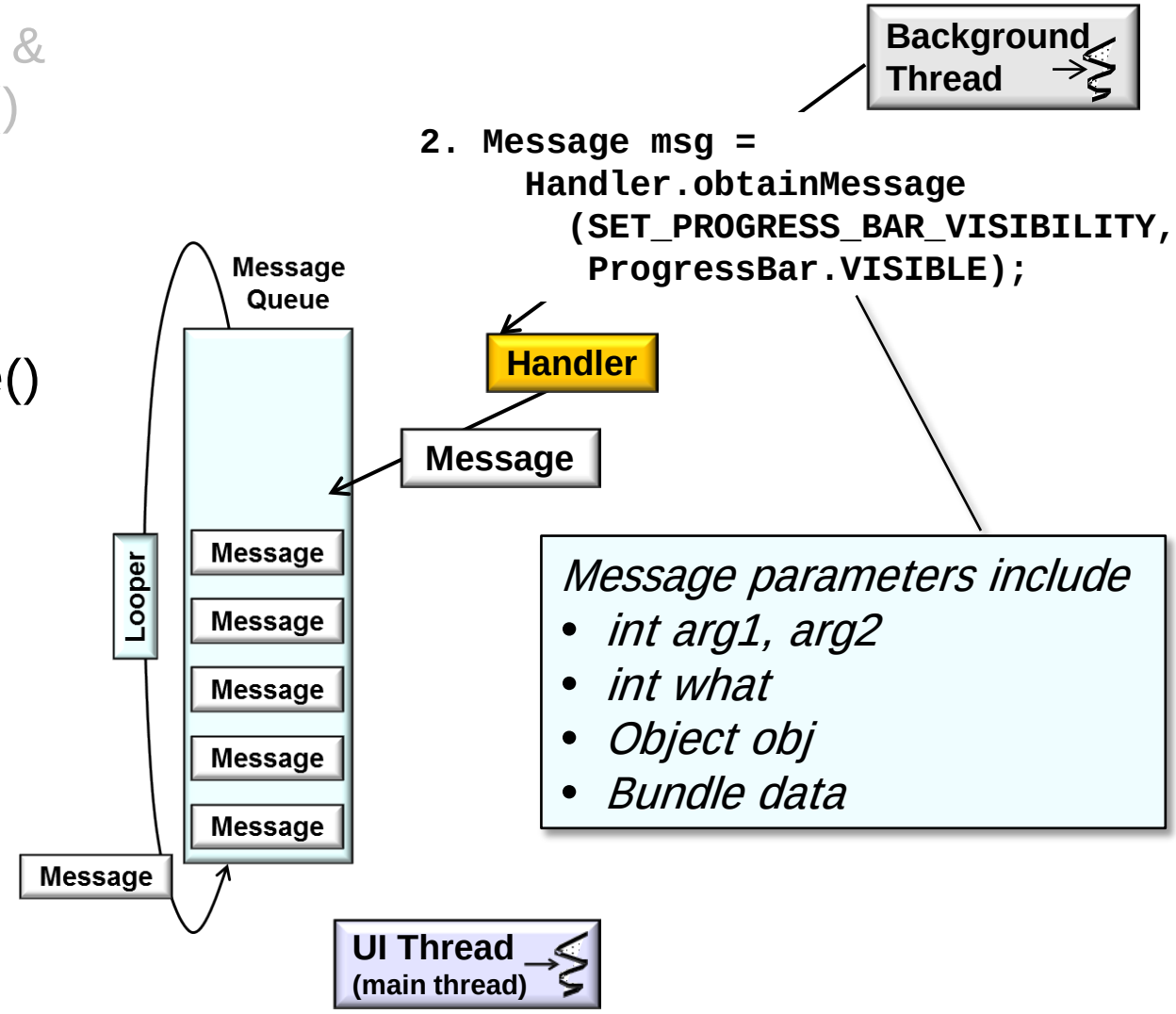
- Extend the Handler class & override handleMessage() hook method



developer.android.com/reference/android/os/Handler.html#handleMessage(android.os.Message)

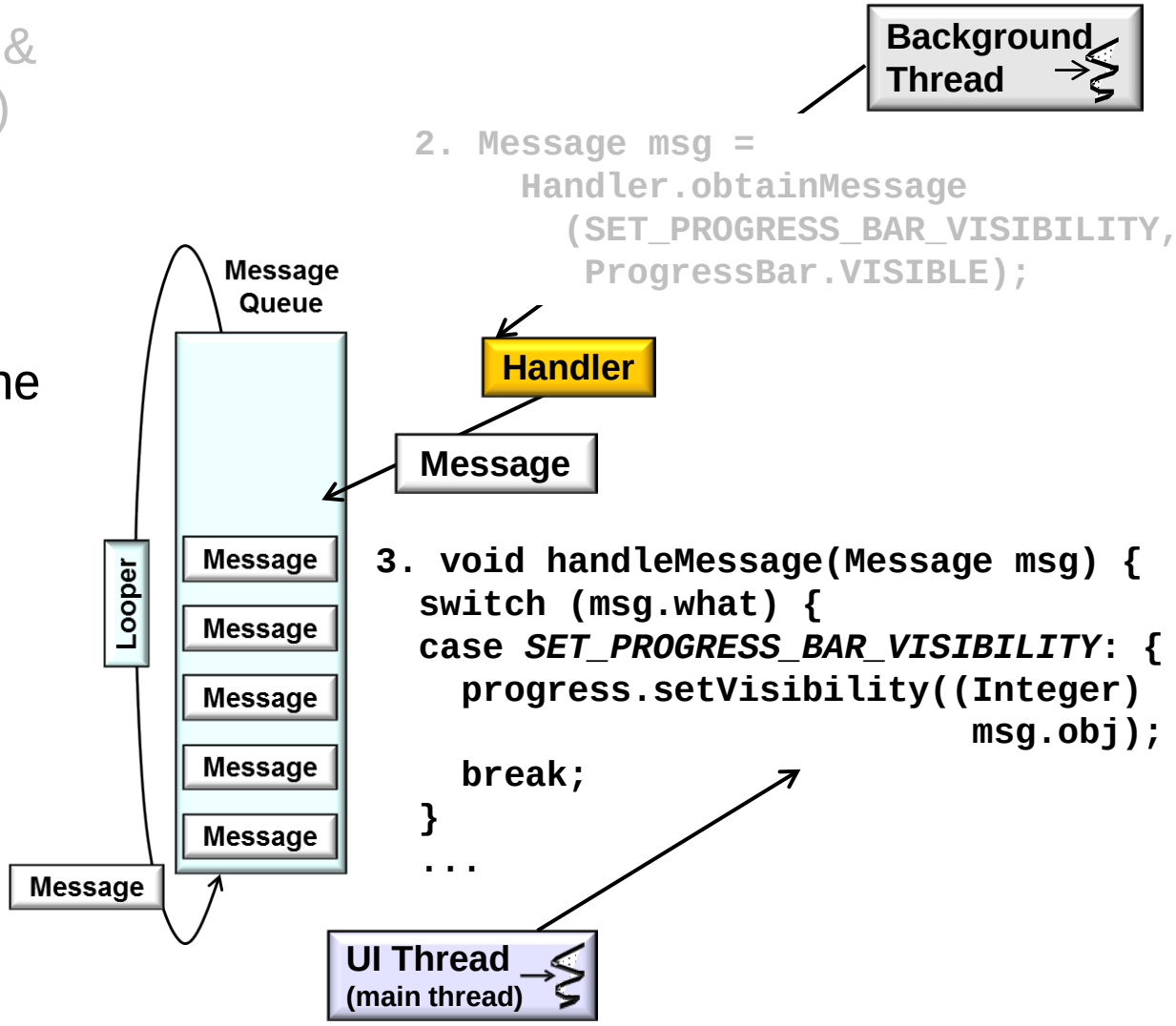
Programming with the Handler & Messages

- Extend the Handler class & override handleMessage() hook method
- Create Message & set Message content
  - Handler.obtainMessage()
  - Message.obtain()
  - etc.



Programming with the Handler & Messages

- Extend the Handler class & override handleMessage() hook method
- Create Message & set Message content
- Looper framework calls the handleMessage() method in the UI Thread



Example of Messages & Handlers

```
public class SimpleThreadingExample extends Activity {
  ...
  Handler h = new Handler() {

    public void handleMessage(Message msg) {
      switch (msg.what) {
      case SET_PROGRESS_BAR_VISIBILITY: {
        progress.setVisibility((Integer) msg.obj); break;
      }
      case PROGRESS_UPDATE: {
        progress.setProgress((Integer) msg.obj); break;
      }
      case SET_BITMAP: {
        iview.setImageBitmap((Bitmap) msg.obj);  break;
      }
    }
   ...
```

Called back by Looper framework in UI Thread

Example of Messages & Handlers

```
public void onCreate(Bundle savedInstanceState) {
  ...
  iview = …
  progress = …
  final Button button = …
  button.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
      new Thread(new LoadIcon(R.drawable.icon,
                  h)).start();
      }
   });
 }
...
```

Create/start a new thread when user clicks a button

Pass the resource ID of the icon

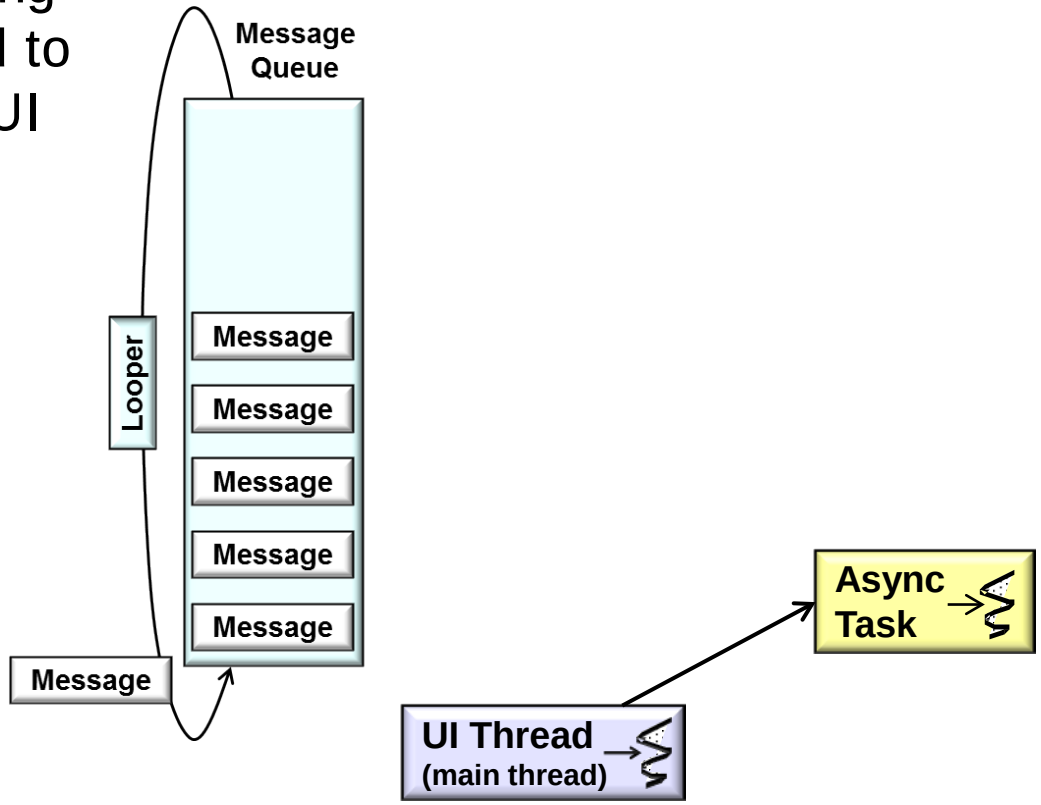
Example of Messages & Handlers

Send various Messages

```
private class LoadIcon implements Runnable {
  public void run() {
    Message msg = h.obtainMessage
      (SET_PROGRESS_BAR_VISIBILITY, ProgressBar.VISIBLE);
    h.sendMessage(msg);
    final Bitmap tmp =
      BitmapFactory.decodeResource(getResources(), resId);
    for (int i = 1; i < 11; i++) {
      msg = h.obtainMessage(PROGRESS_UPDATE, i * 10);
      h.sendMessageDelayed(msg, i * 100);
    }
    msg = h.obtainMessage(SET_BITMAP, tmp);
    h.sendMessageAtTime(msg, 11 * 200);
    msg = h.obtainMessage(SET_PROGRESS_BAR_VISIBILITY,
                          ProgressBar.INVISIBLE);
    h.sendMessageAtTime(msg, 11 * 200);
    ...
```

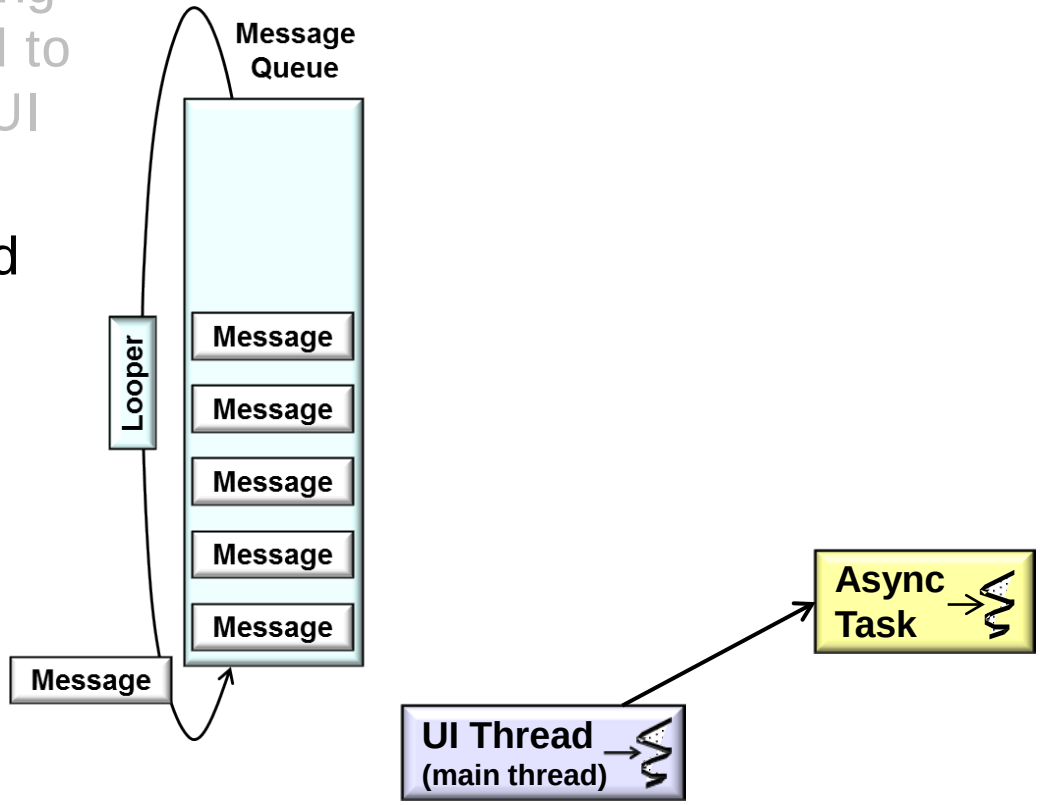
Programming with AsyncTask

- AsyncTask provides a structured way to manage work involving background & UI threads
  - Simplifies creation of long-running tasks that need to communicate with the UI



Programming with AsyncTask

- AsyncTask provides a structured way to manage work involving background & UI threads
  - Simplifies creation of long-running tasks that need to communicate with the UI
  - AsyncTask is designed as a helper class around Thread & Handler

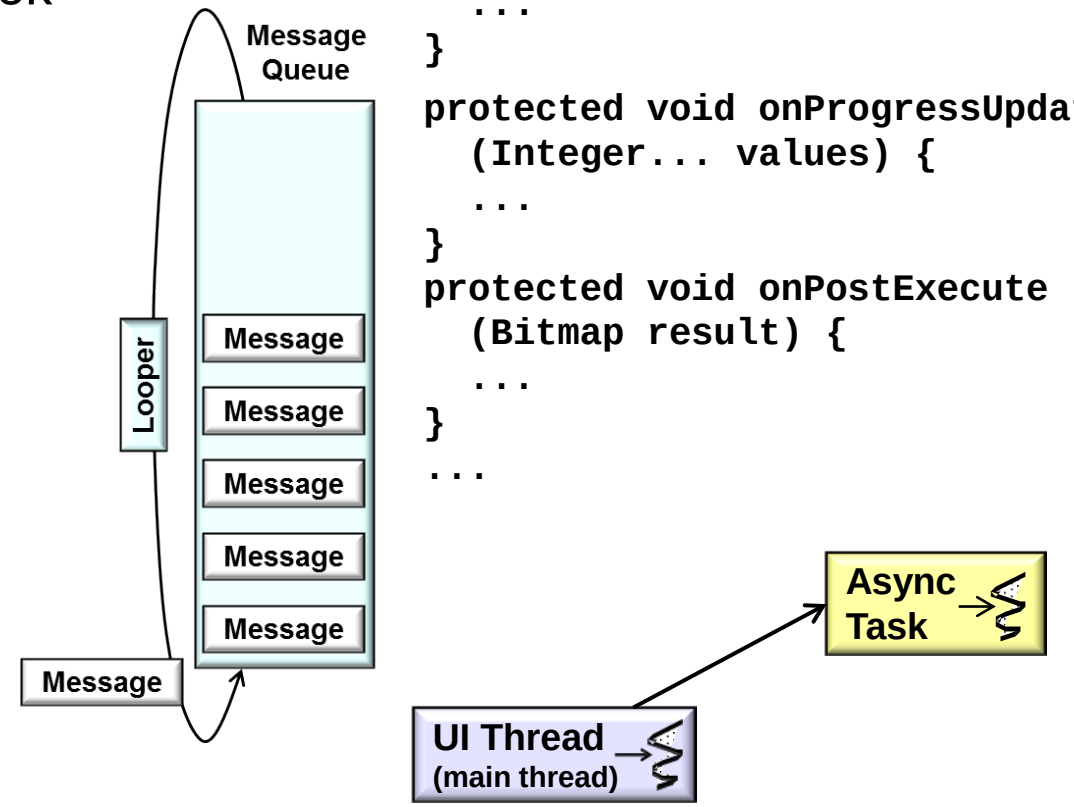


frameworks/base/core/java/android/os/AsyncTask.java has the source code

Programming with AsyncTask

- AsyncTask provides a structured way to manage work involving background & UI threads
- Must be subclassed & hook methods overridden

```
class LoadIcon extends
      AsyncTask<Integer,Integer,Bitmap>
{
  protected Bitmap doInBackground
    (Integer... resId) {
    ...
  }
  protected void onProgressUpdate
    (Integer... values) {
    ...
  }
  protected void onPostExecute
    (Bitmap result) {
    ...
  }
  ...
```



frameworks/base/core/java/android/os/AsyncTask.java has the source code

Example of Android AsyncTask

```
public class SimpleThreadingExample extends Activity {
  ImageView iview;
  ProgressBar progress;
  public void onCreate(Bundle savedInstanceState) {
    ...
    iview = …
    progress = …
    final Button button = …
      button.setOnClickListener(new OnClickListener() {
         public void onClick(View v) {
           new LoadIcon().execute(R.drawable.icon);
         }
    });
 }
 ...
```

Create/start a new AsyncTask when user clicks a button

Pass the resource ID of the icon

Example of Android AsyncTask

```
class LoadIcon extends
      AsyncTask<Integer, Integer, Bitmap> {
  protected void onPreExecute() {
    progress.setVisibility(ProgressBar.VISIBLE);
  }

  protected Bitmap doInBackground(Integer... resId) {
    Bitmap tmp =
      BitmapFactory.decodeResource(getResources(),
                                   resId[0]);
    publishProgress(...);
    return tmp;
  }
  ...
```

Customize AsyncTask

Runs before doInBackground() in UI Thread

Runs in background thread

Convert resource ID to bitmap

Simulate long-running operation

Example of Android AsyncTask

```
class LoadIcon extends
      AsyncTask<Integer, Integer, Bitmap> {
  ...
```

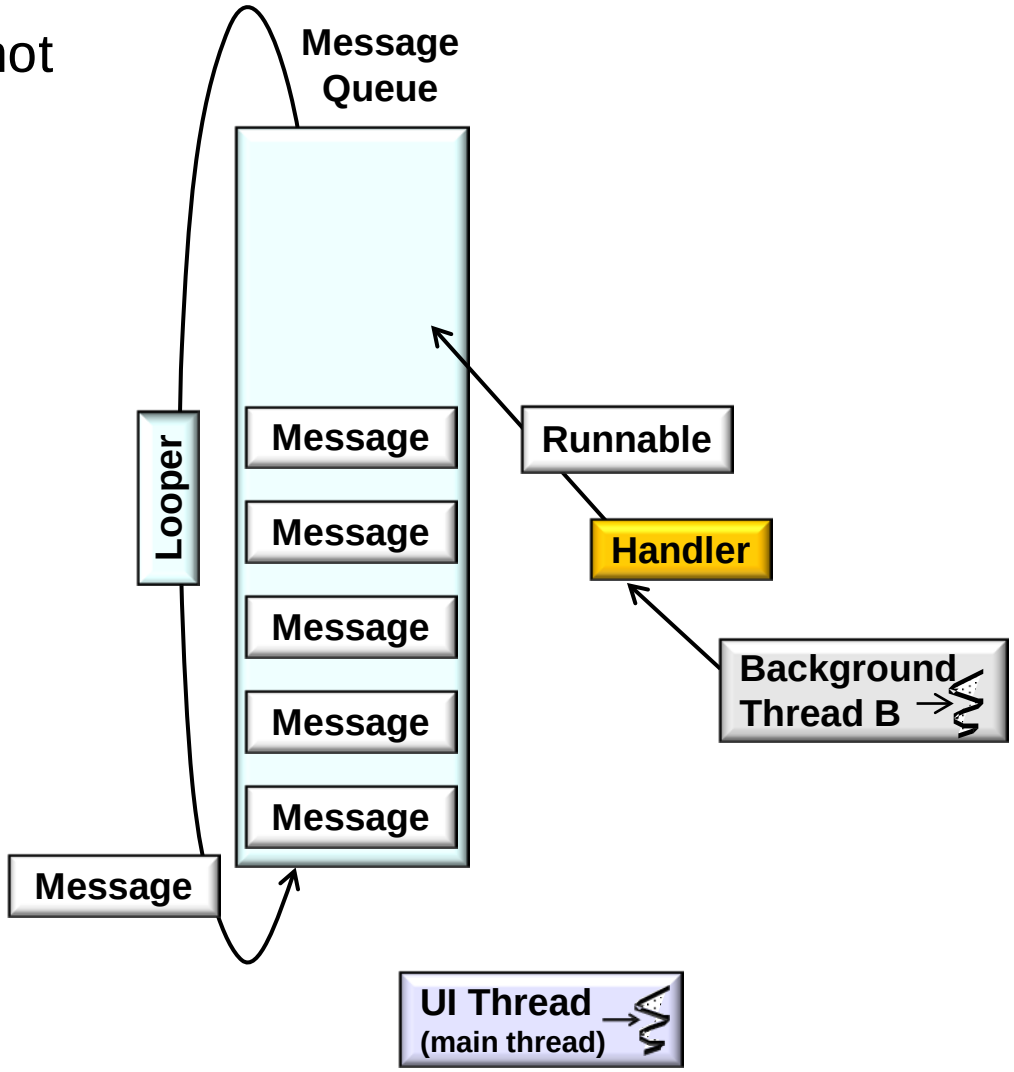
Invoked in response to publishProgress() in UI Thread

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

  protected void onPostExecute(Bitmap result) {
    progress.setVisibility(ProgressBar.INVISIBLE);
    iview.setImageBitmap(result);
  }
 ...
```

Runs after doInBackground() in UI Thread

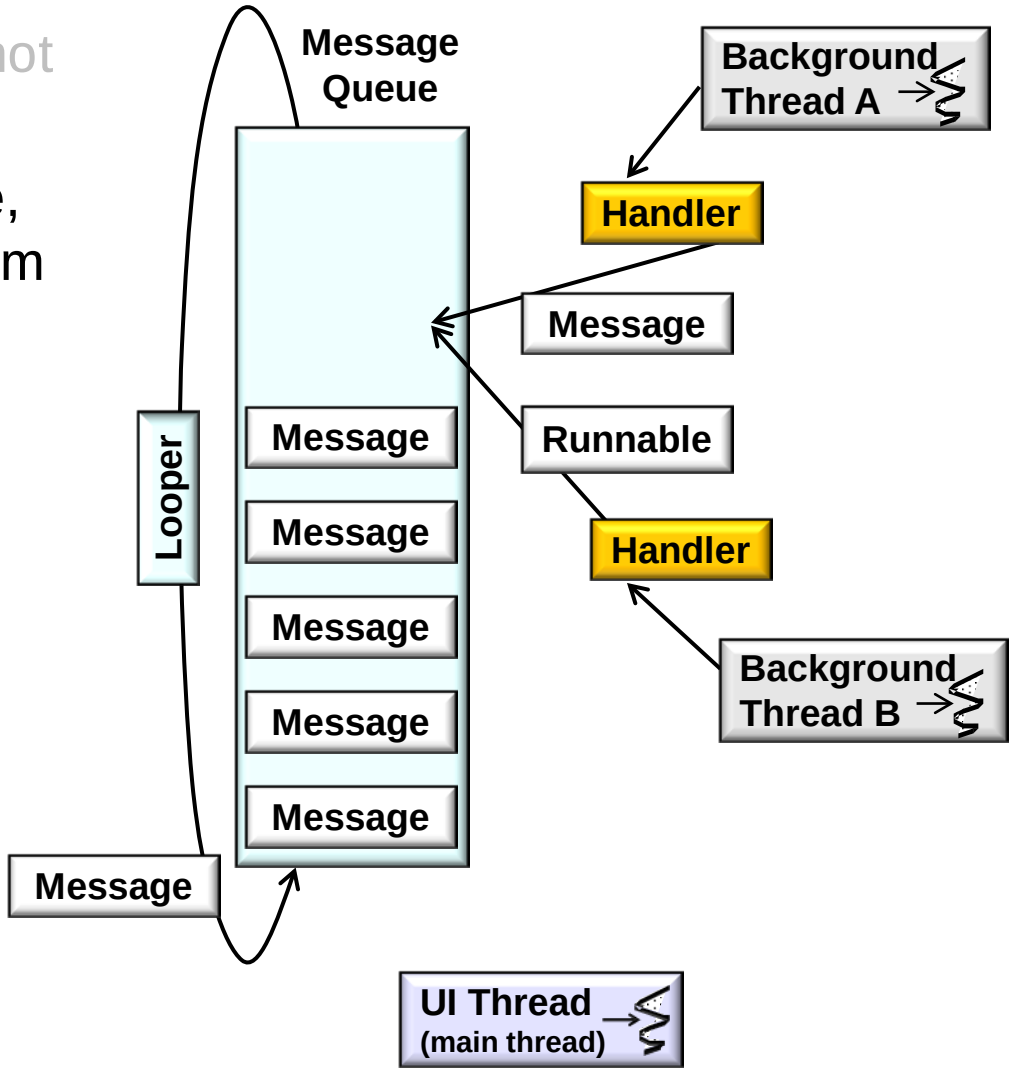
Summary

- Posting Runnables is simple, but not particularly flexible



Summary

- Posting Runnables is simple, but not particularly flexible
- Sending Messages is more flexible, but is more complicated to program



Summary

- Posting Runnables is simple, but not particularly flexible
- Sending Messages is more flexible, but is more complicated to program
- AsyncTask is powerful, but is more complicated internally & has more overhead due to potential for more thread synchronization & schedulting