

# 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?

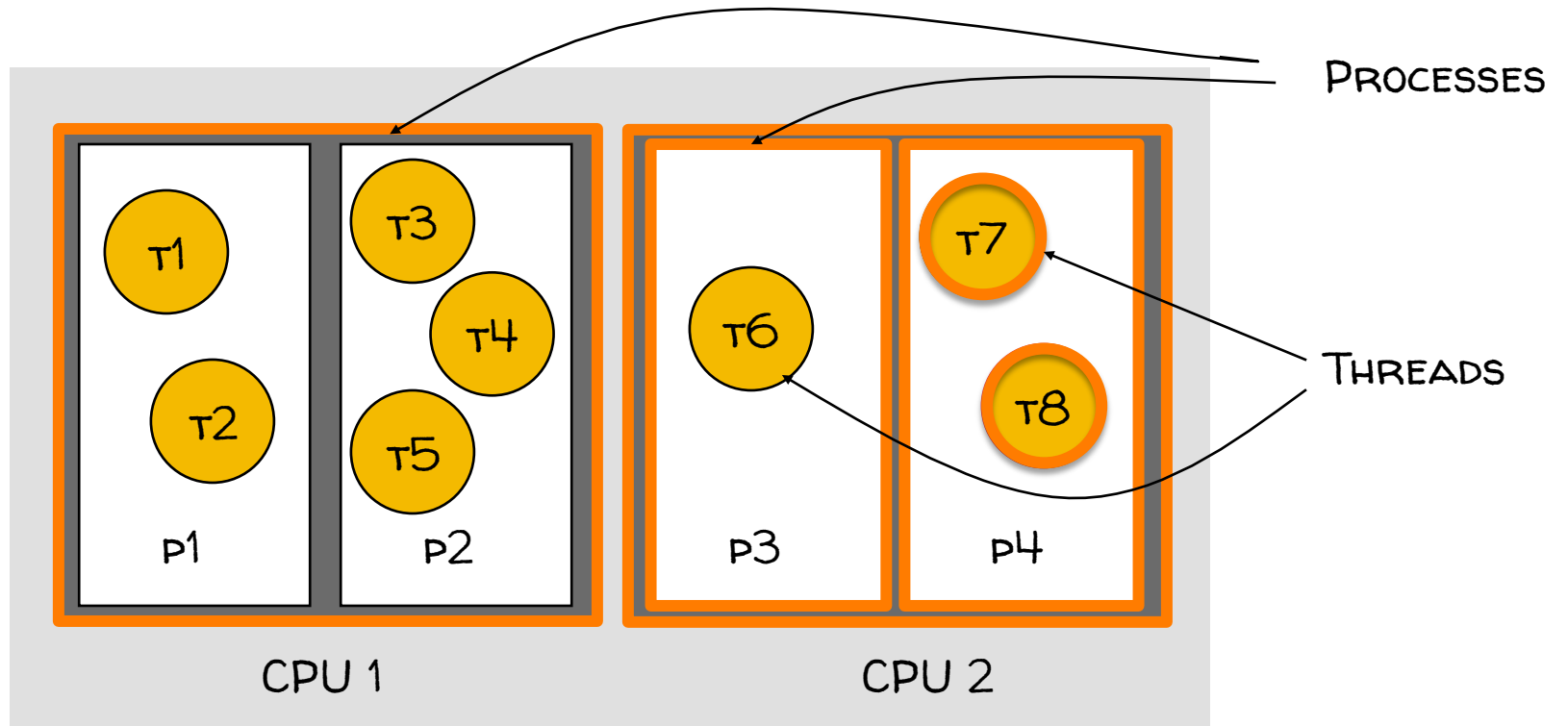
## CONCEPTUAL VIEW

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](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

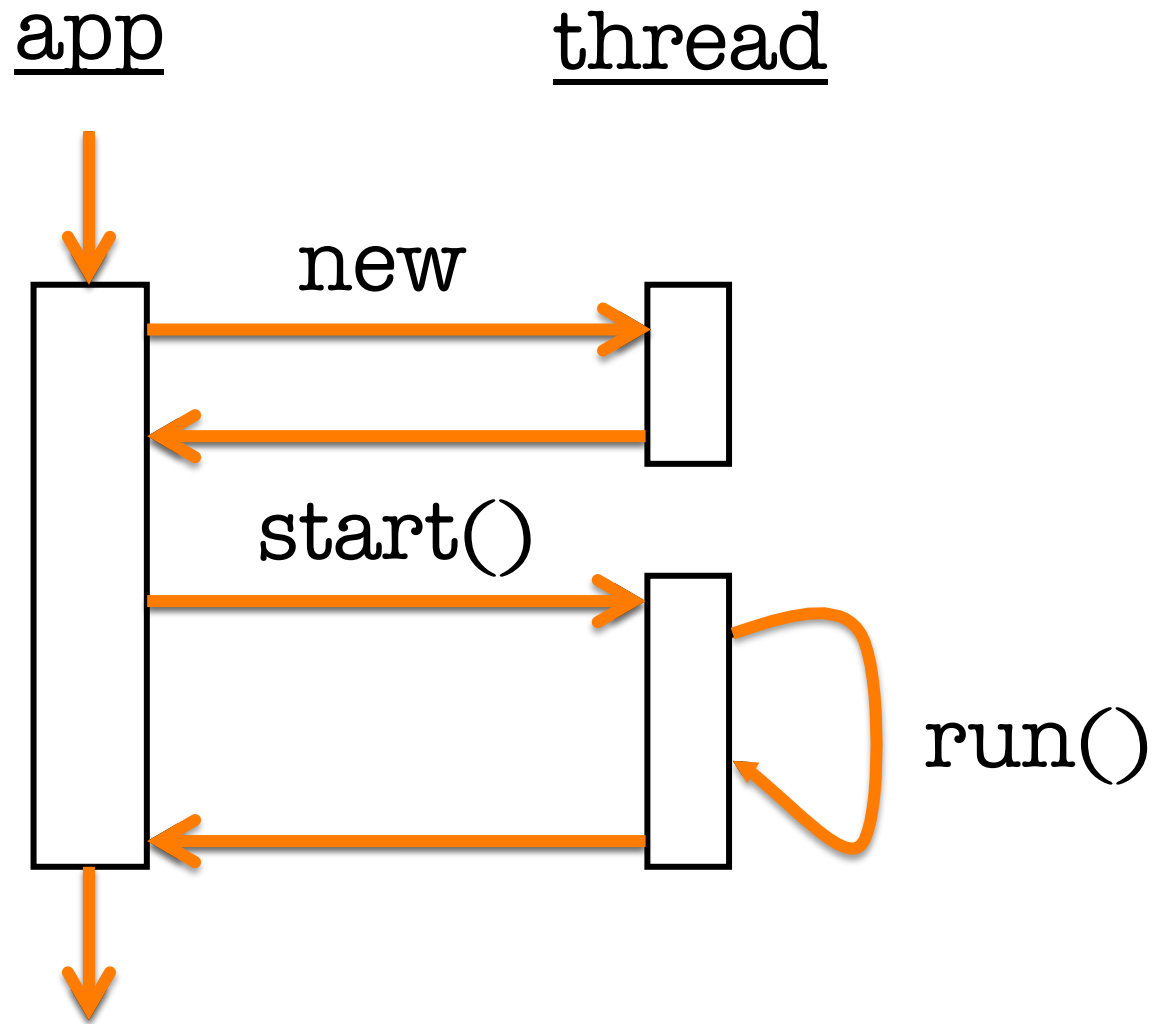
INSTANTIATE A THREAD OBJECT

INVOKE THE THREAD'S `start()` METHOD

THREAD'S `run()` METHOD GET CALLED

THREAD TERMINATES WHEN `run()` RETURNS

# BASIC THREAD USE CASE



# THREADINGNOTREADING

APPLICATION DISPLAYS TWO BUTTONS

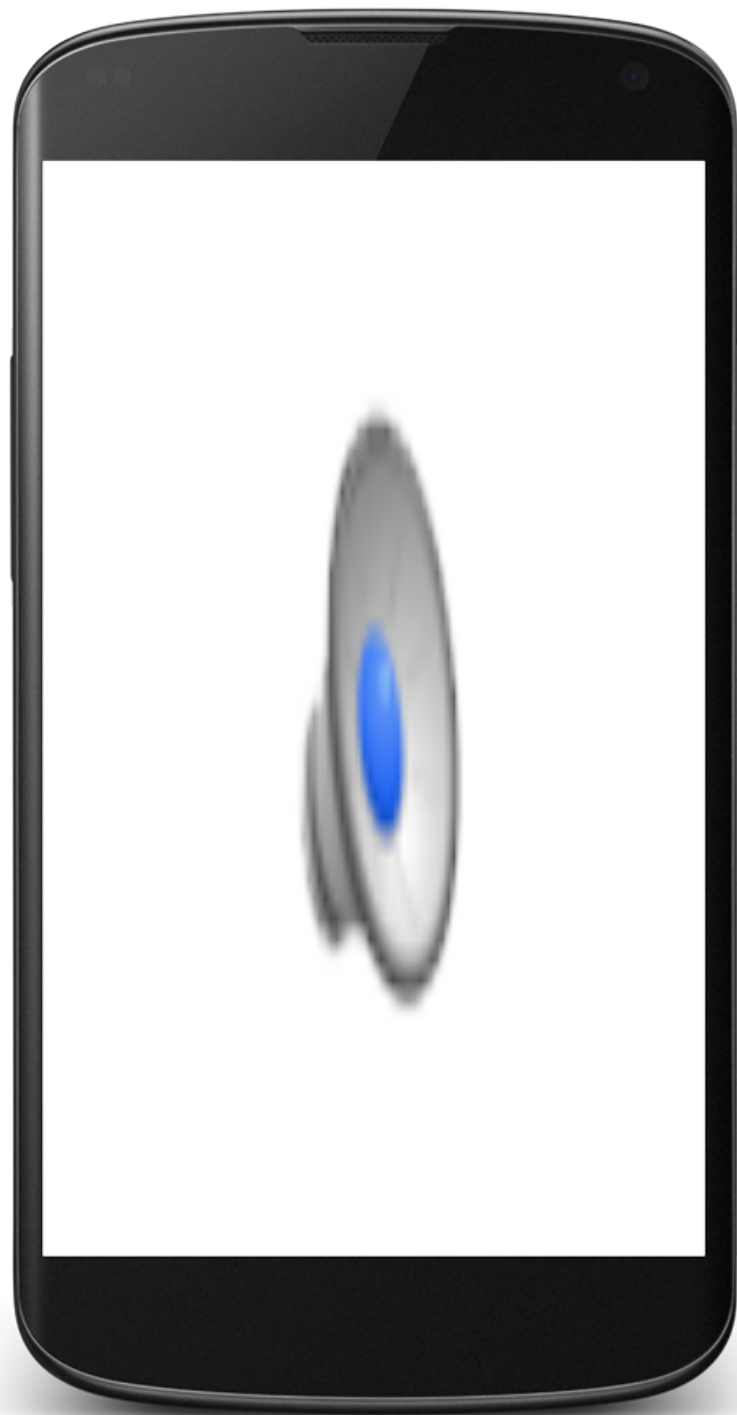
LOADICON

LOAD A BITMAP FROM A RESOURCE FILE &  
DISPLAY

SHOW LOADED BITMAP

OTHER BUTTON

DISPLAY SOME TEXT



# THREADINGSIMPLE

SEEMINGLY OBVIOUS, BUT INCORRECT, SOLUTION:

BUTTON LISTENER SPAWNS A SEPARATE  
THREAD TO LOAD BITMAP & DISPLAY IT





# 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

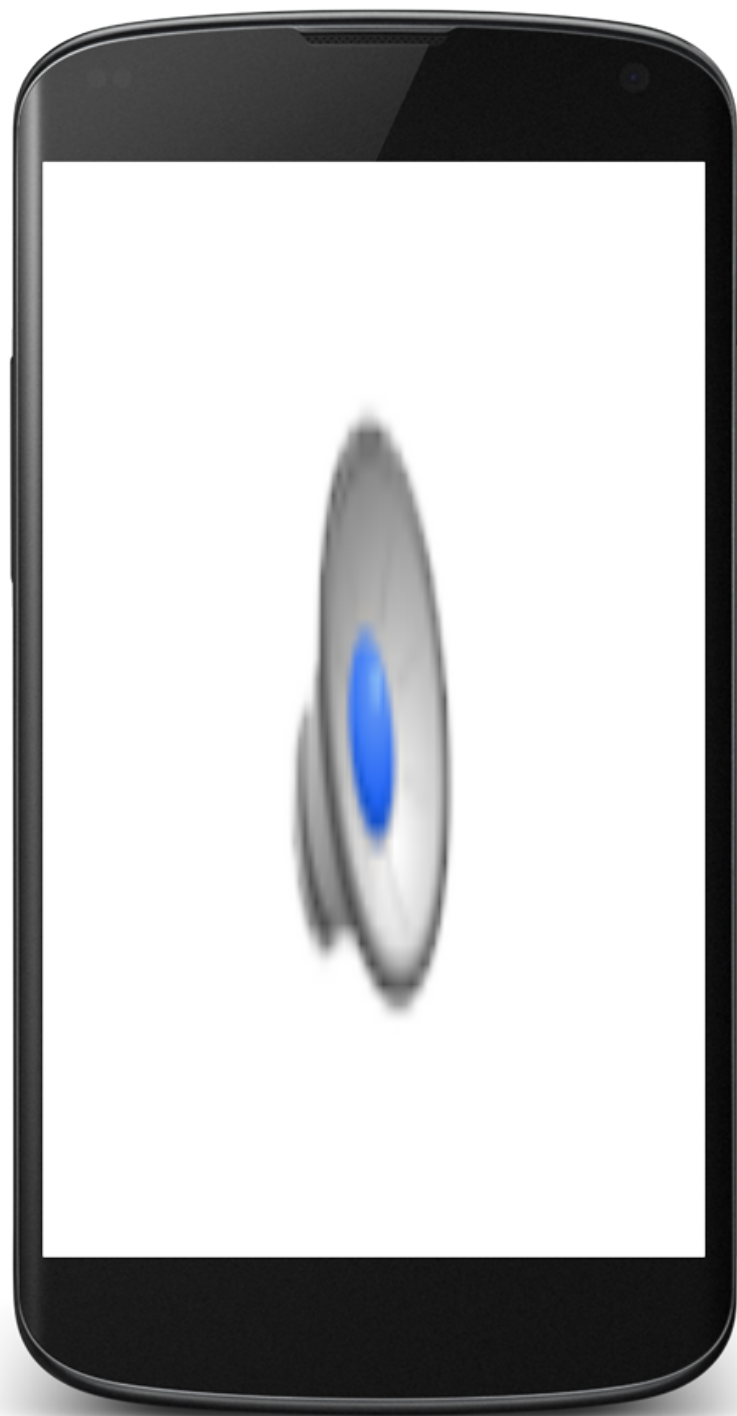
NEED TO DO WORK IN A BACKGROUND THREAD,  
BUT UPDATE THE UI IN THE UI THREAD

ANDROID PROVIDES SEVERAL METHODS THAT  
ARE GUARANTEED TO RUN IN THE UI THREAD

`boolean View.post (Runnable action)`

`void Activity.`

`runOnUiThread (Runnable action)`





# 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

CAN CALL void publishProgress(Progress... values)

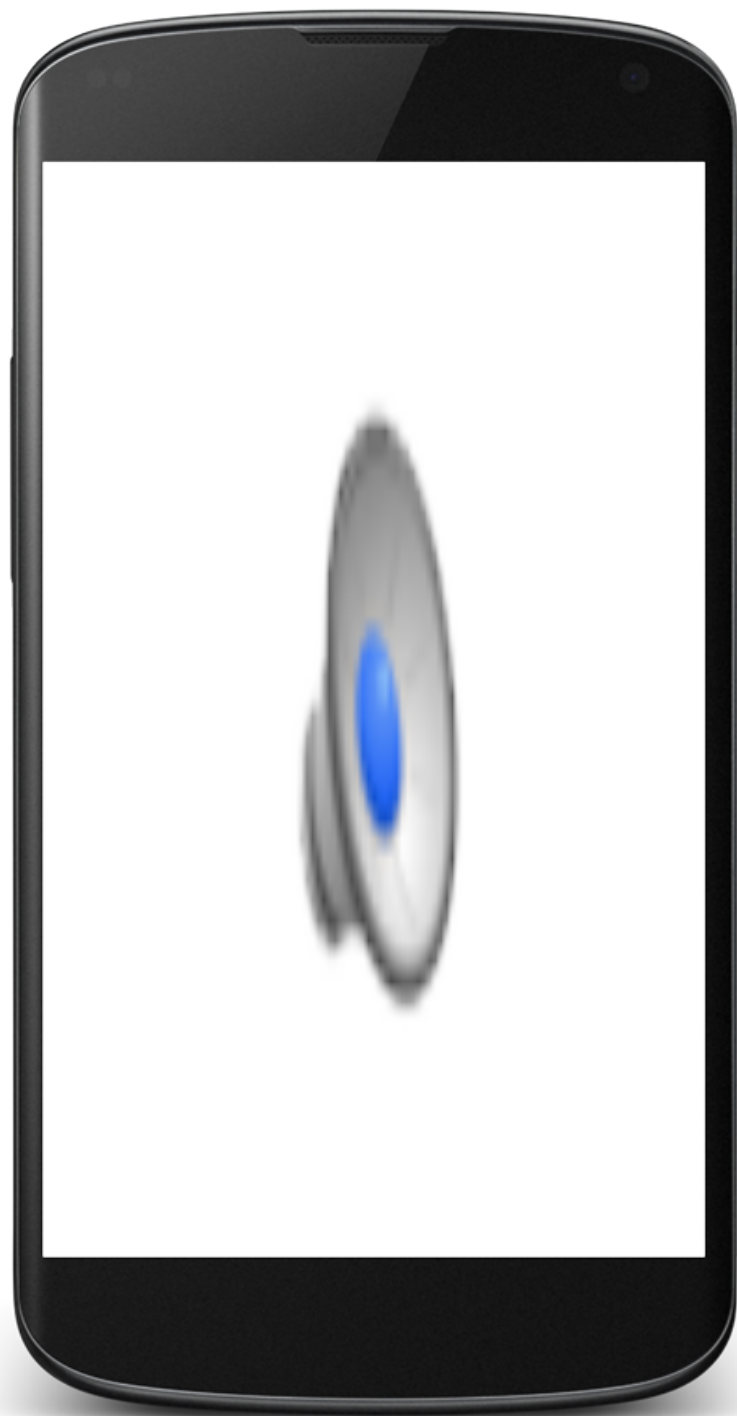
# ASYNCTASK

void onProgressUpdate (Progress... values)

INVOKED IN RESPONSE TO publishProgress()

void onPostExecute (Result result)

RUNS AFTER doInBackground()





# 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

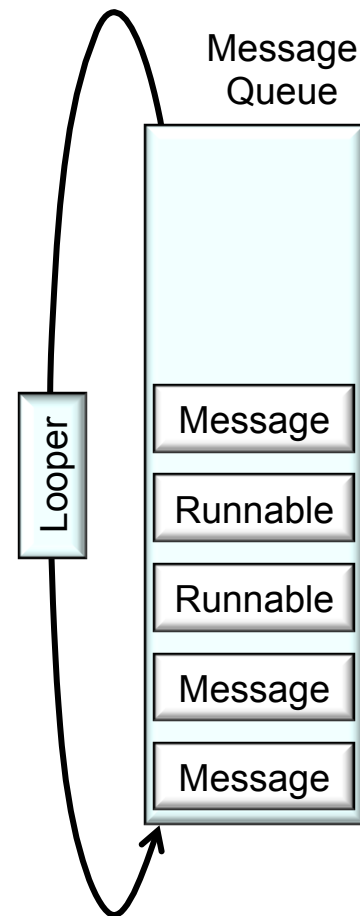
CAN CONTAIN A MESSAGE CODE, AN OBJECT &  
INTEGER ARGUMENTS

HANDLER IMPLEMENTS RESPONSE

# HANDLER ARCHITECTURE

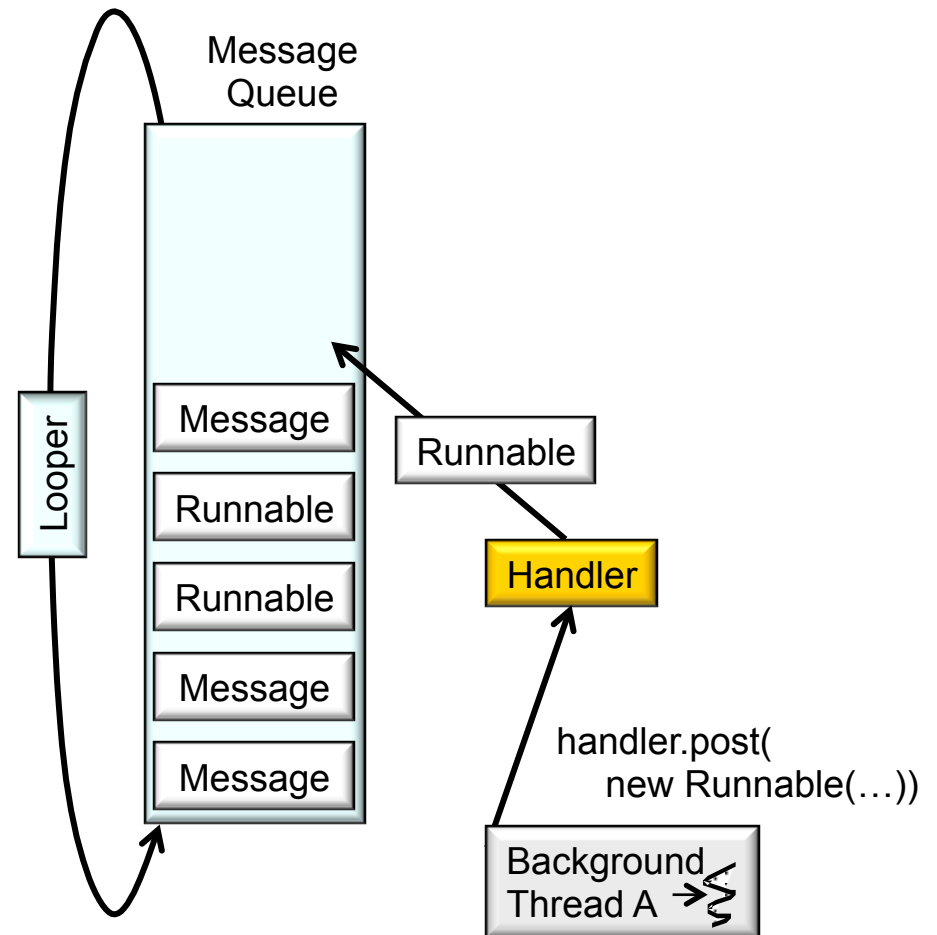
EACH ANDROID  
THREAD IS  
ASSOCIATED WITH A  
MESSAGEQUEUE & A  
LOOPER

A MESSAGEQUEUE  
HOLDS MESSAGES  
AND RUNNABLES TO  
BE DISPATCHED BY  
THE LOOPER



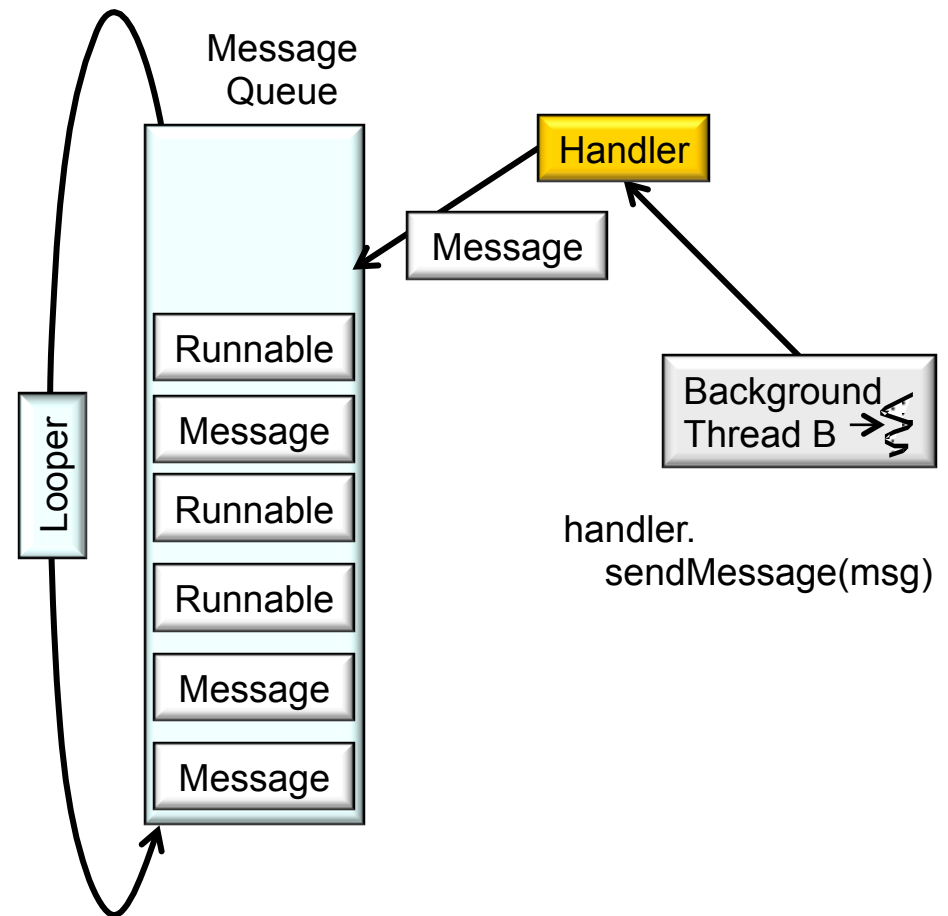
# HANDLER ARCHITECTURE

ADD RUNNABLES TO  
MESSAGEQUEUE BY  
CALLING HANDLER'S  
`post()` METHOD



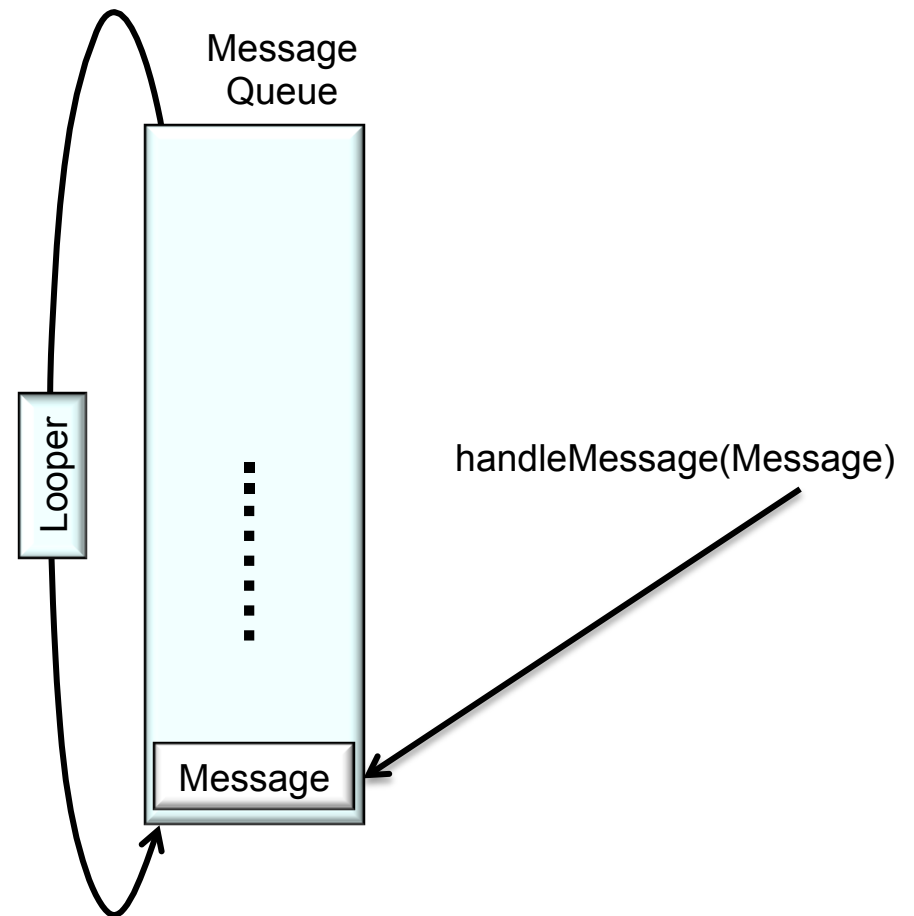
# HANDLER ARCHITECTURE

ADD MESSAGES TO  
MESSAGEQUEUE BY  
CALLING HANDLER'S  
sendMessage()  
METHOD



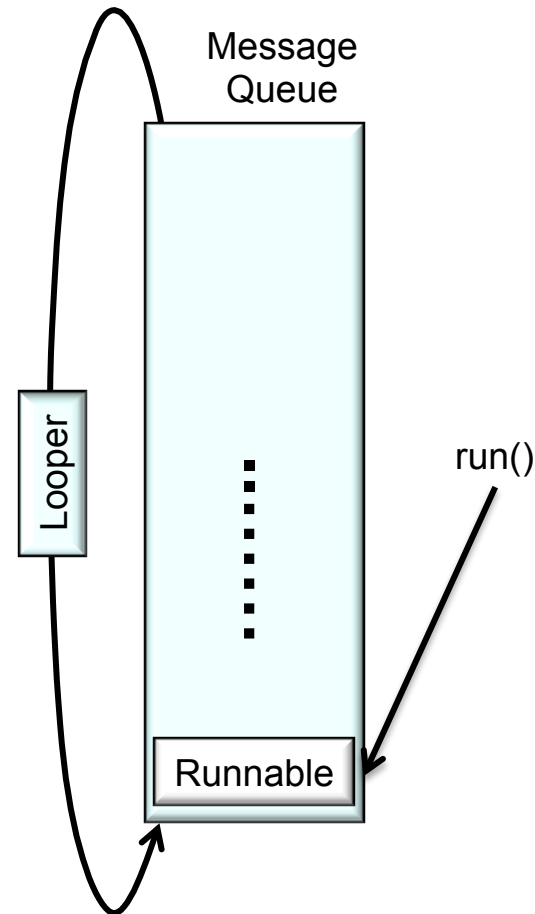
# HANDLER ARCHITECTURE

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 AT CURRENT TIME + DELAY



NEXT TIME

ALARMS