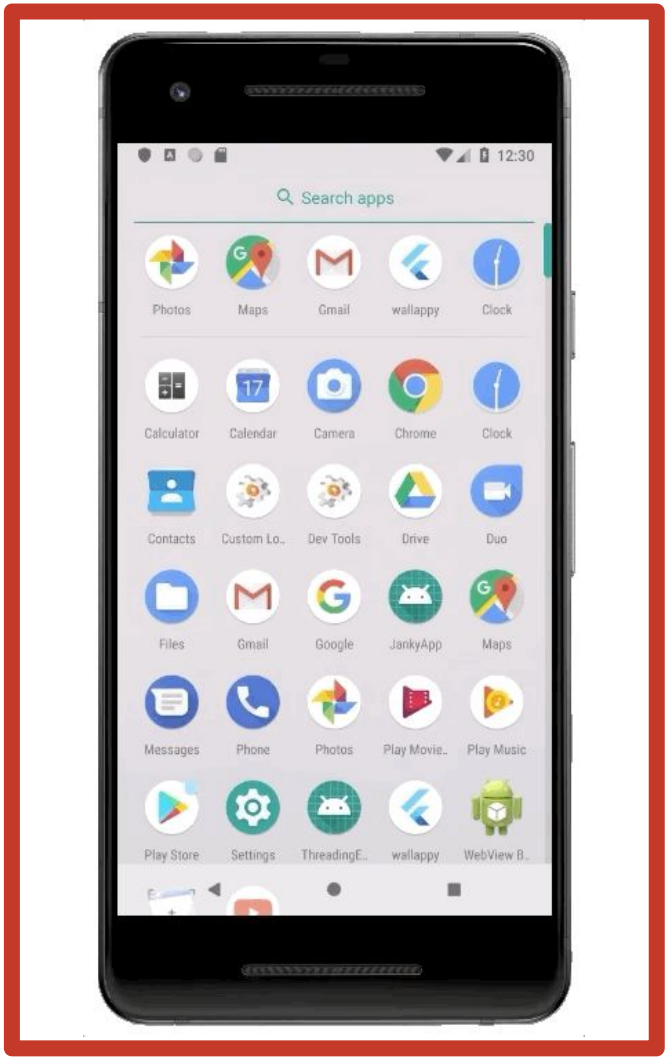
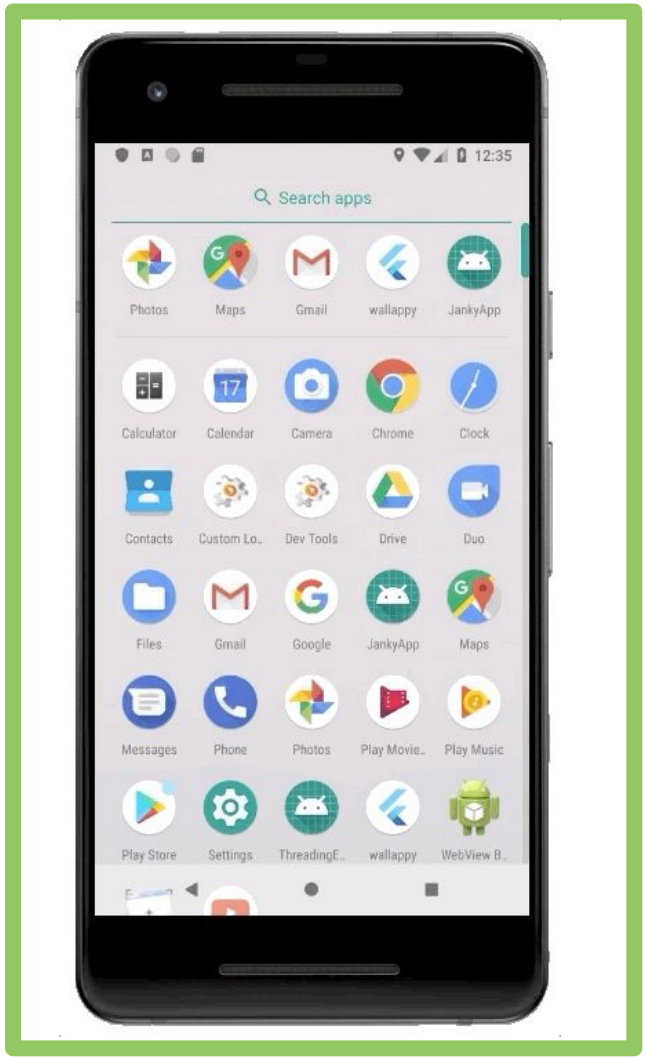
Android Lecture #8

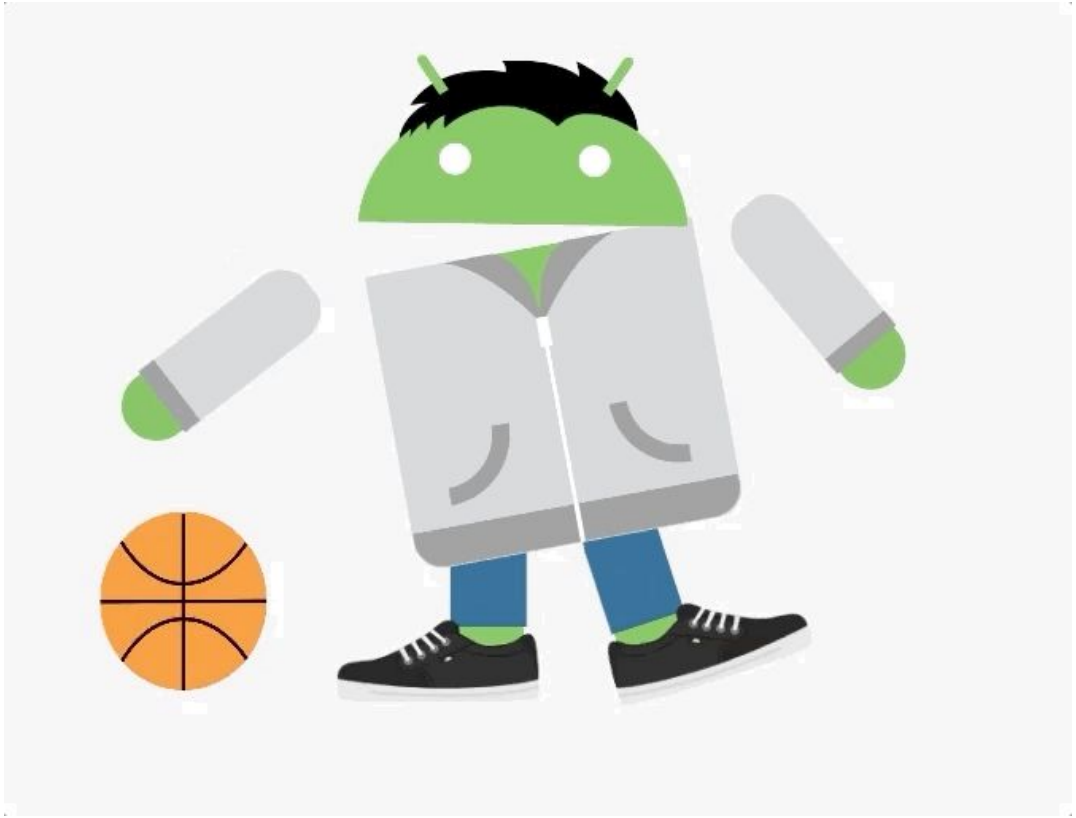
Threads

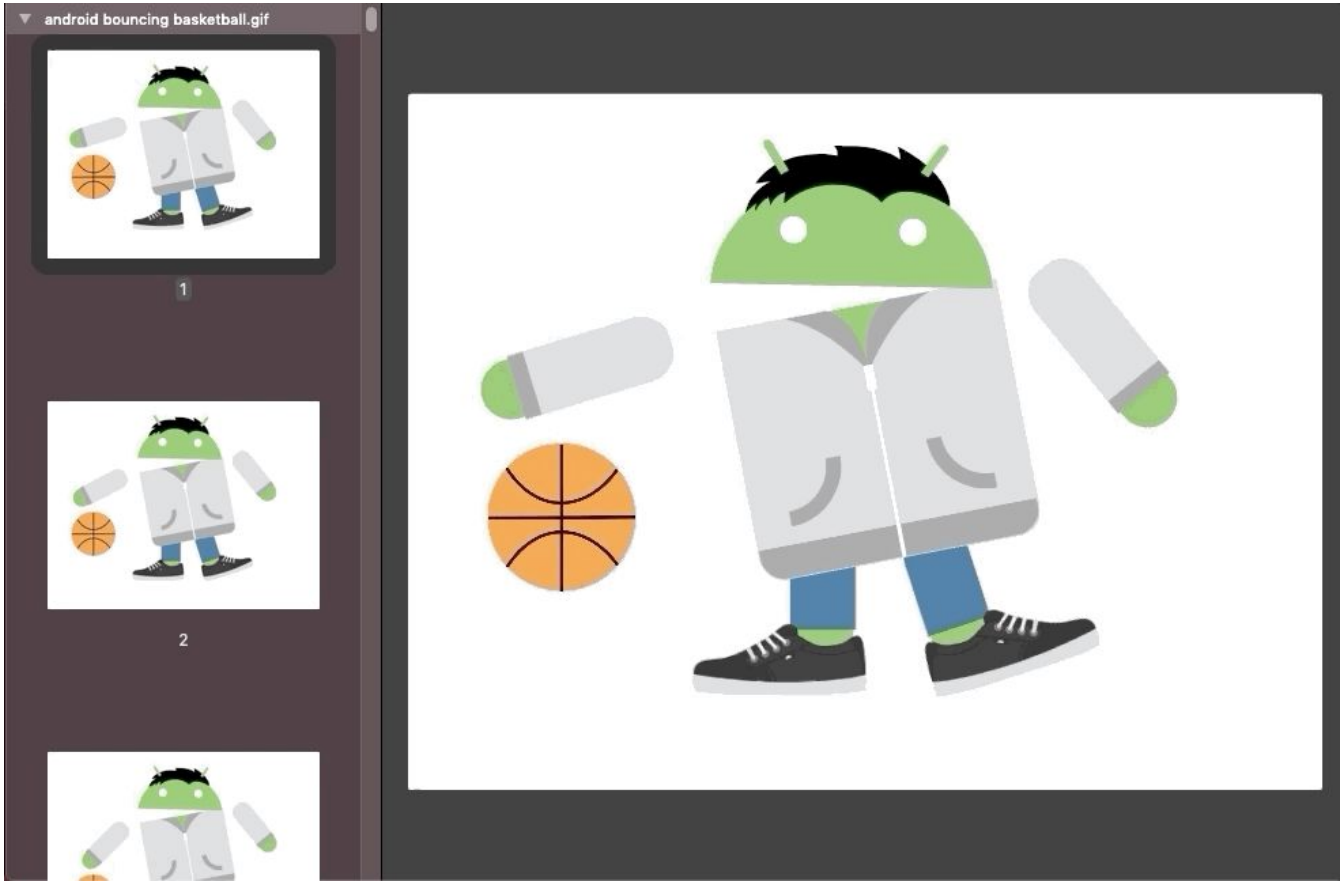

12:35
Search apps
Photos
Maps
Gmail
wallappy
JankyApp
Calculator
Calendar
Camera
Chrome
Clock
Contacts
Custom Lo..
Dev Tools
Drive
Duo
Files
Gmail
Google
JankyApp
Maps
Messages
Phone
Photos
Play Movie..
Play Music
Play Store
Settings
ThreadingE..
wallappy
WebView B..

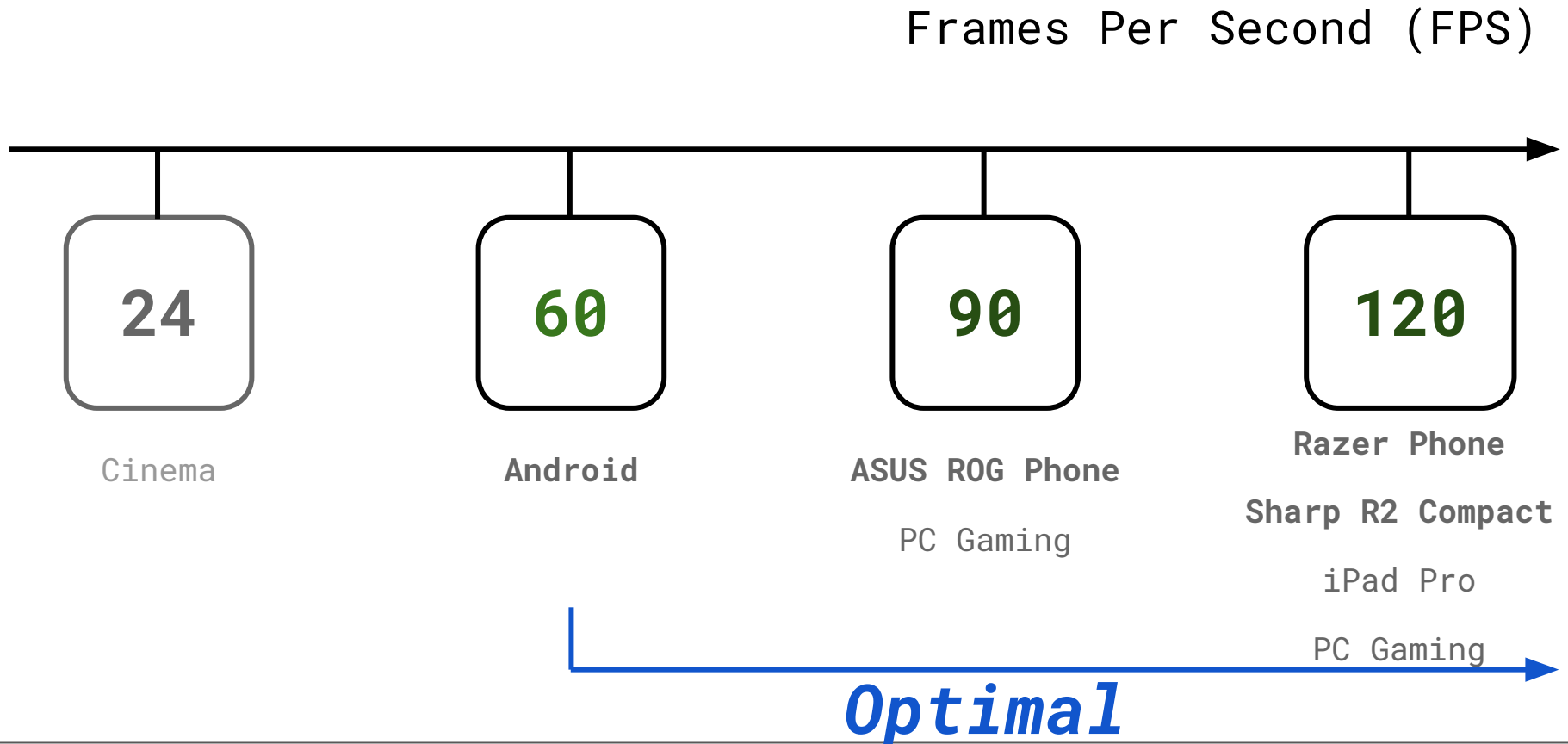

12:30
Search apps
Photos
Maps
Gmail
wallappy
Clock
Calculator
Calendar
Camera
Chrome
Clock
Contacts
Custom Lo..
Dev Tools
Drive
Duo
Files
Gmail
Google
JankyApp
Maps
Messages
Phone
Photos
Play Movie..
Play Music
Play Store
Settings
ThreadingE..
wallappy
WebView B..


12:30
Search apps
Photos
Maps
Gmail
wallappy
Clock
Calculator
Calendar
Camera
Chrome
Clock
Contacts
Custom Lo..
Dev Tools
Drive
Duo
Files
Gmail
Google
JankyApp
Maps
Messages
Phone
Photos
Play Movie..
Play Music
Play Store
Settings
ThreadingE..
wallappy
WebView B..


12:35
Search apps
Photos
Maps
Gmail
wallappy
JankyApp
Calculator
Calendar
Camera
Chrome
Clock
Contacts
Custom Lo..
Dev Tools
Drive
Duo
Files
Gmail
Google
JankyApp
Maps
Messages
Phone
Photos
Play Movie..
Play Music
Play Store
Settings
ThreadingE..
wallappy
WebView B..

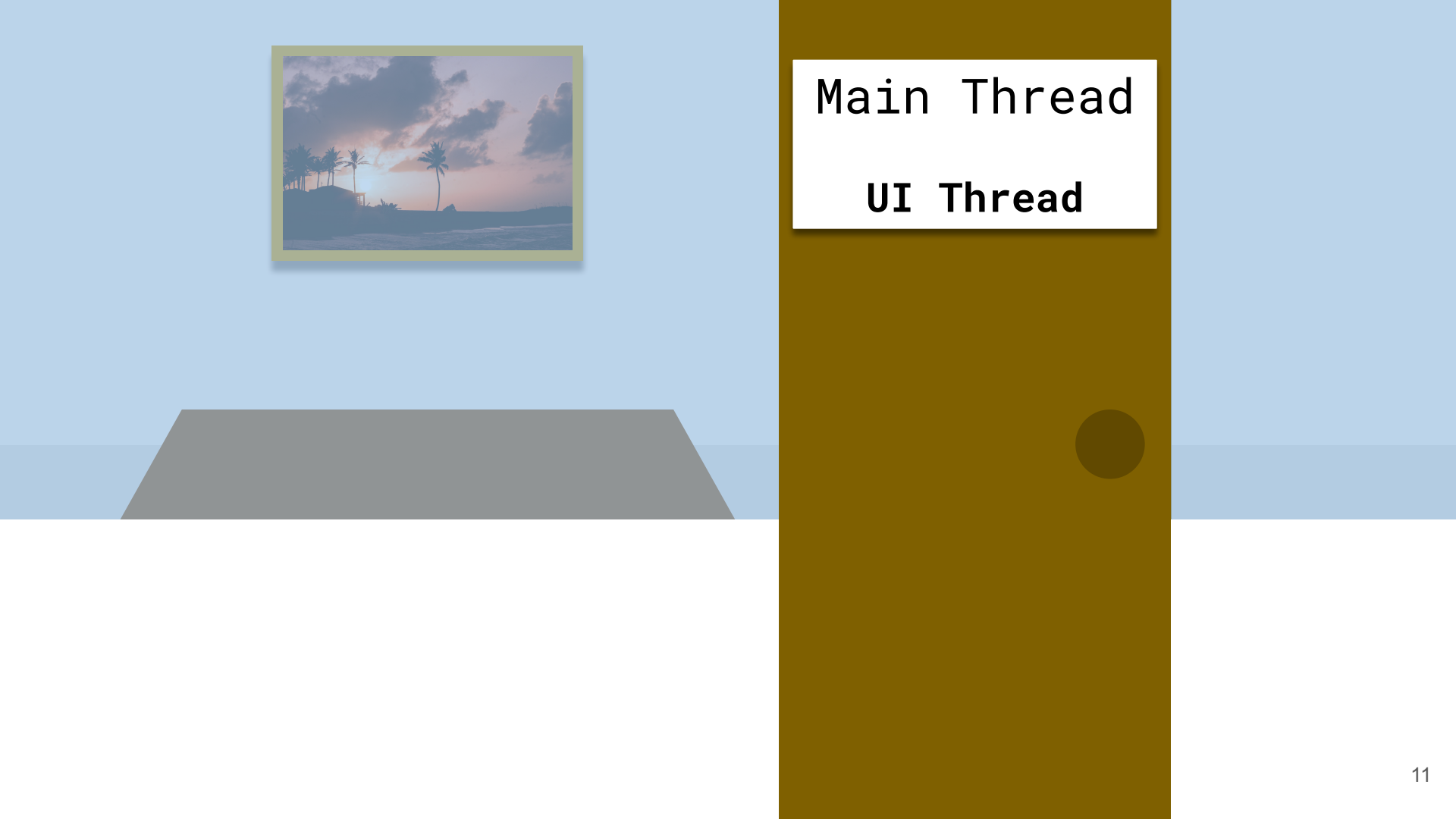


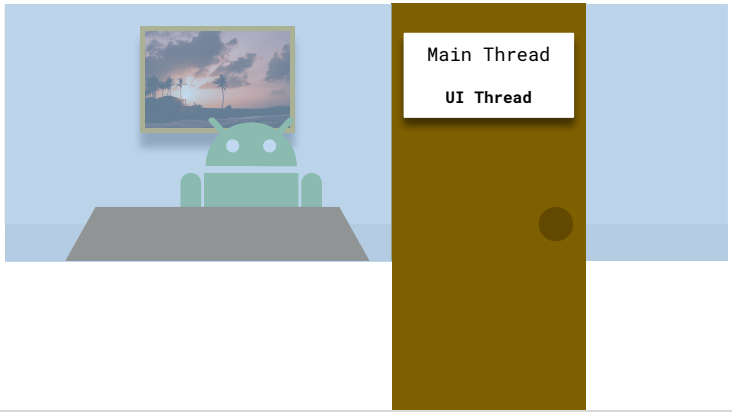

android bouncing basketball.gif
1
2

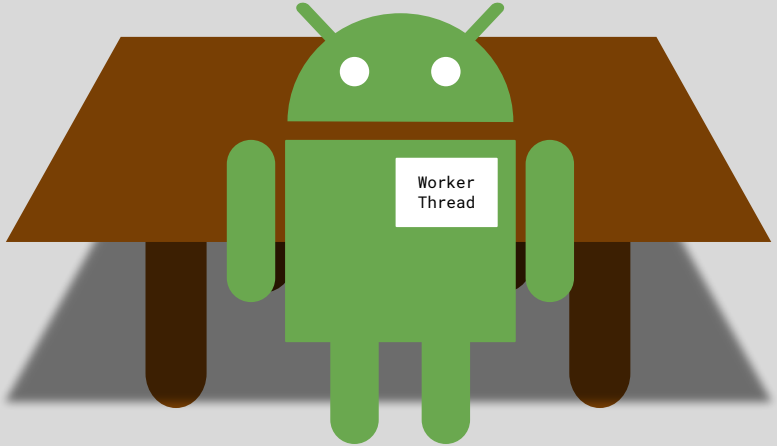

Frames Per Second (FPS)
24
Cinema
60
Android
90
ASUS ROG Phone
PC Gaming
120
Razer Phone
Sharp R2 Compact
iPad Pro
PC Gaming
Optimal

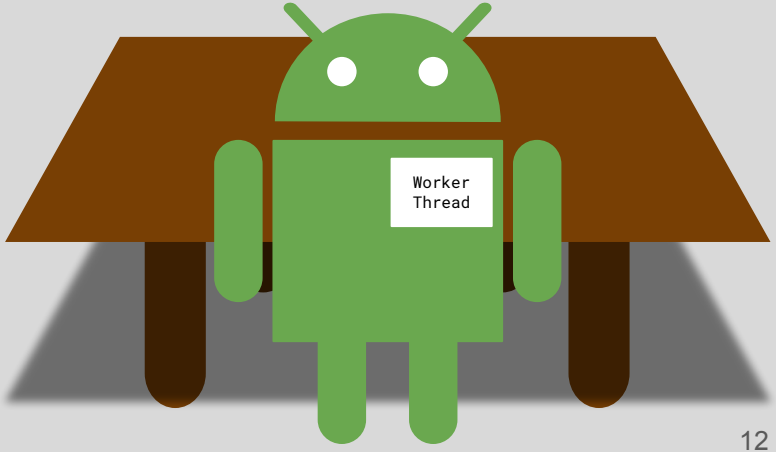
Frame
=
16.67 ms


[adult swim]
What is your problem?

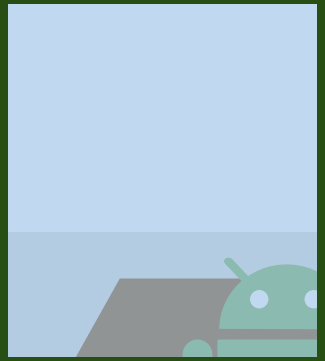

Main Thread
UI Thread

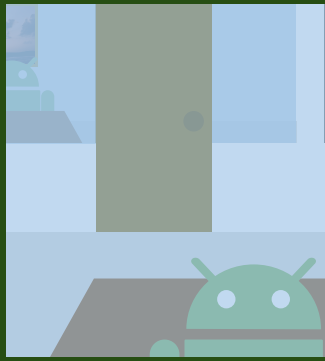

Main Thread
UI Thread

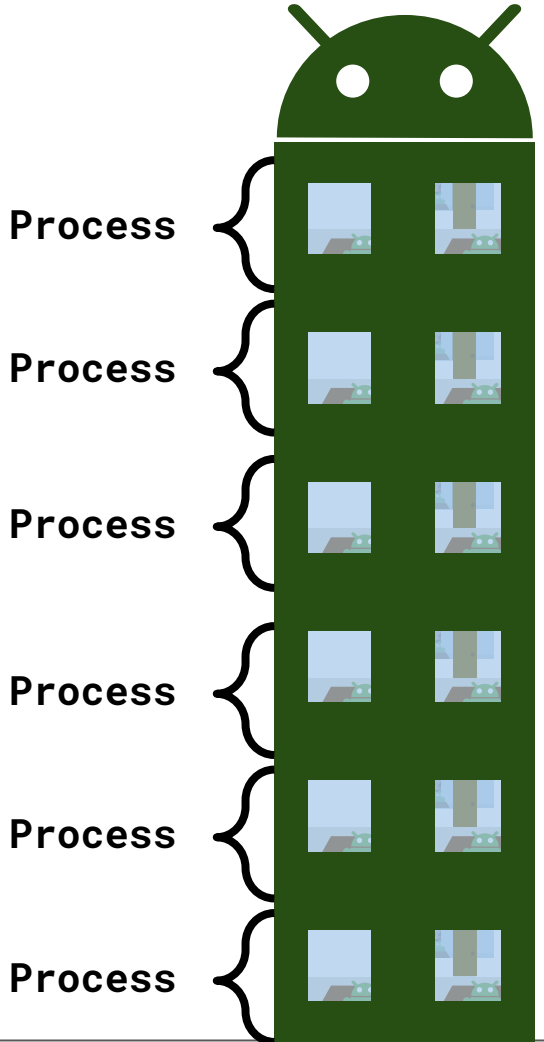

Worker
Thread


Worker
Thread

Process






Process
Process
Process
Process
Process
Process

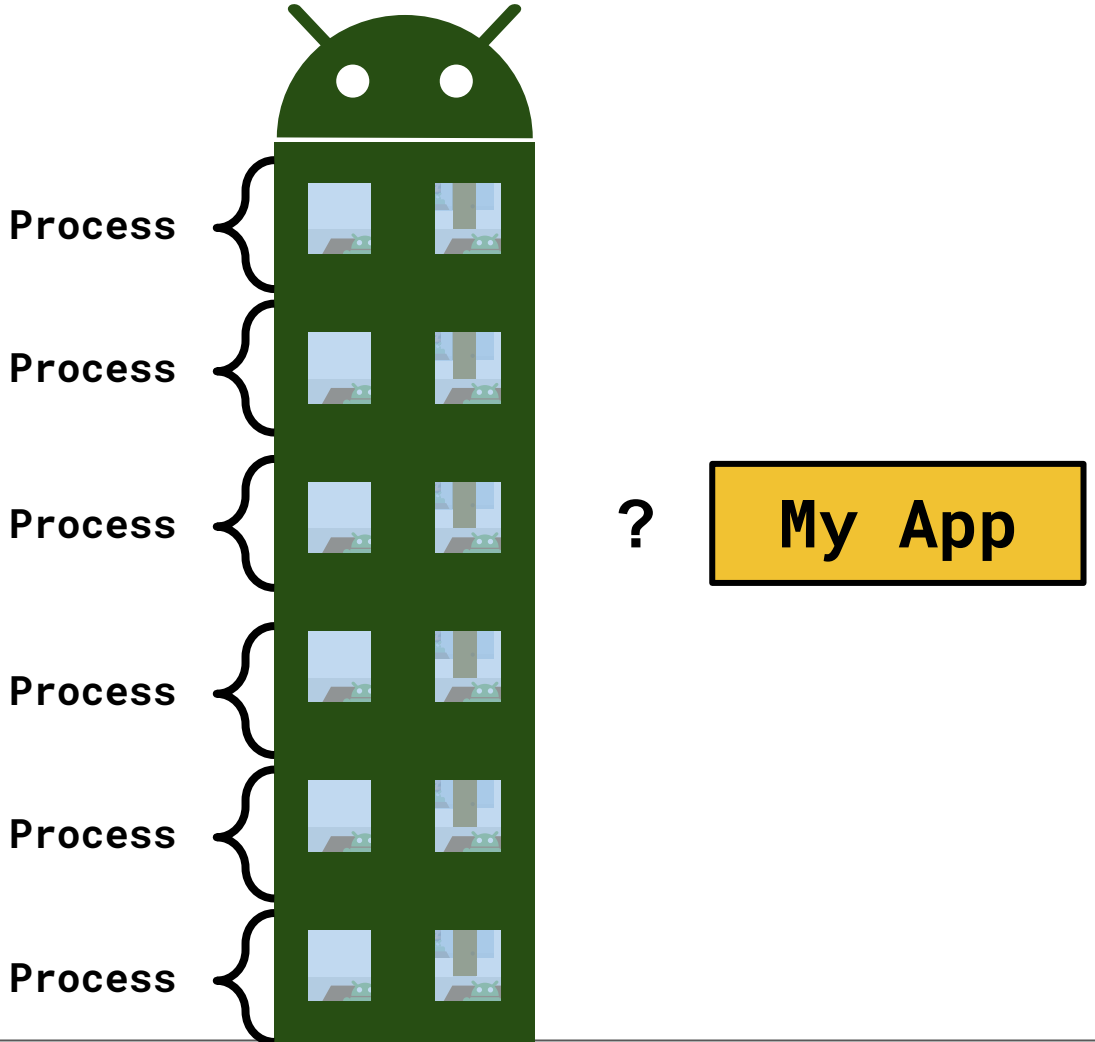
Process VS Thread

Process VS Thread

- **Process can contain a few threads.**

Process VS Thread

- Process can contain a few threads.
- **Android manages processes.**


Process
Process
Process
Process
Process
Process
?
My App

https://developer.android.com/guide/components/processes-and-threads.html

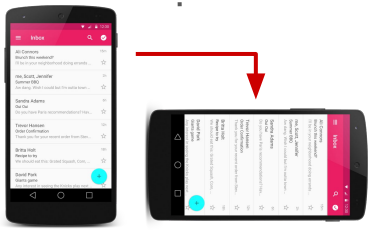
Process

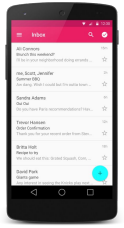
Main Thread (UI Thread)

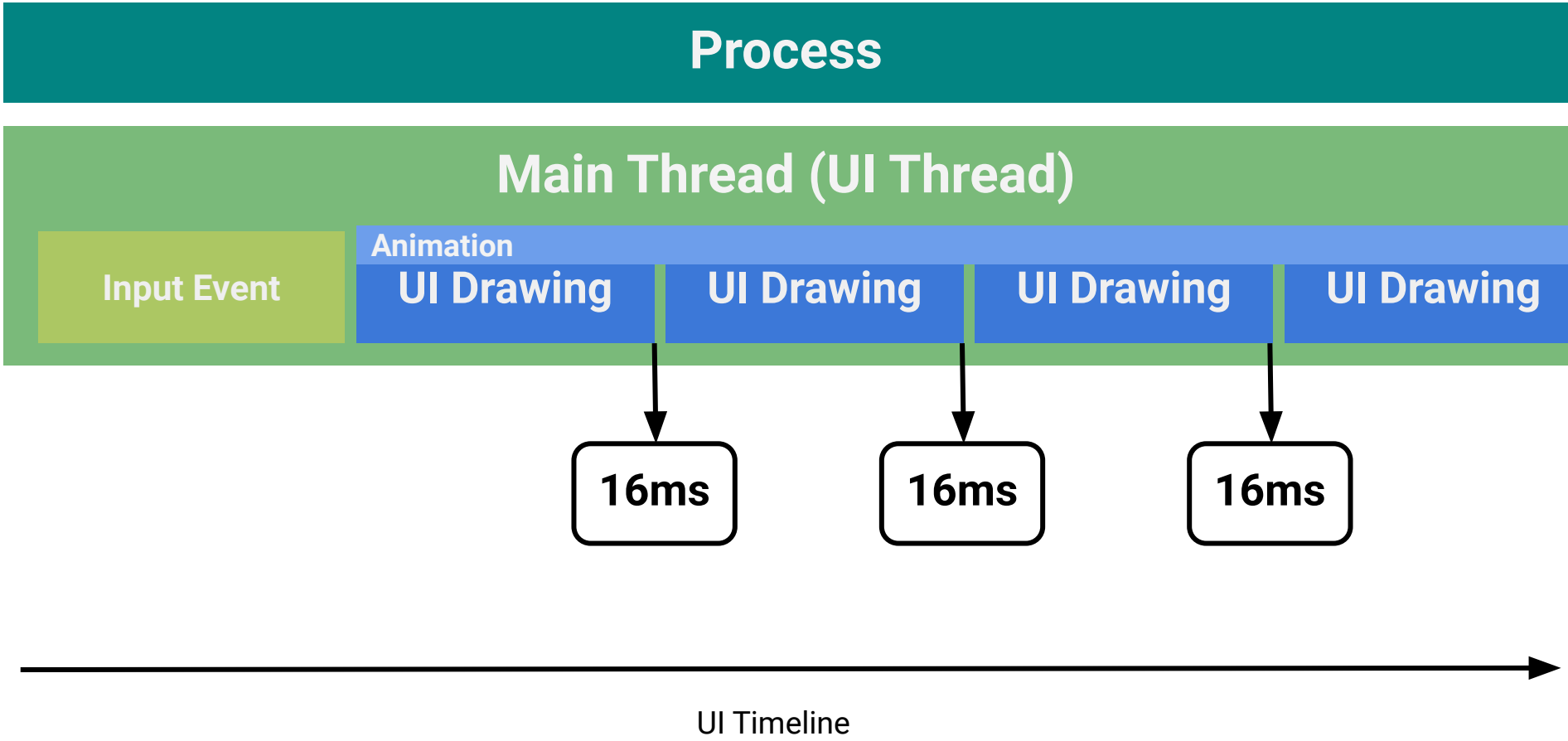
System Event

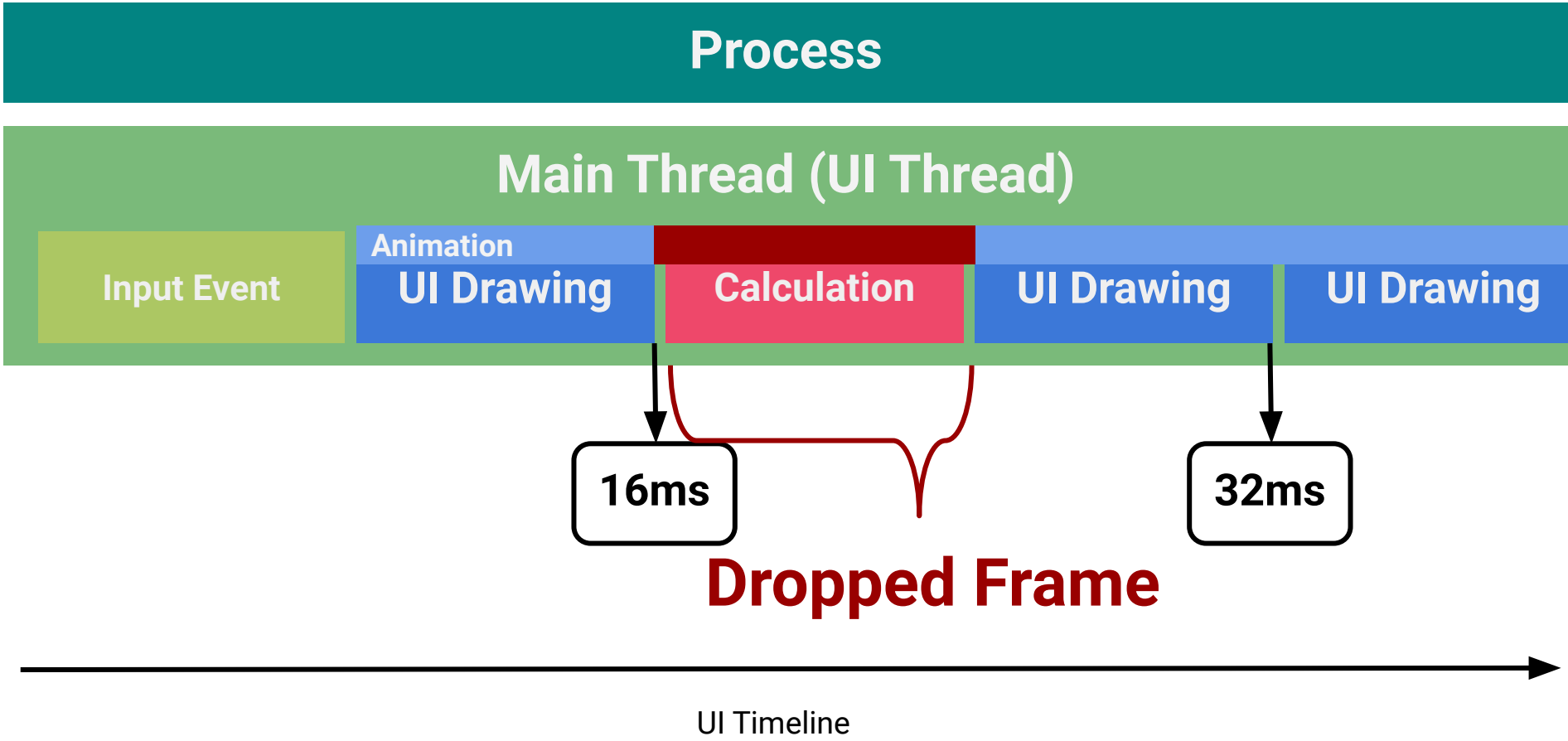
Input Event

UI Drawing

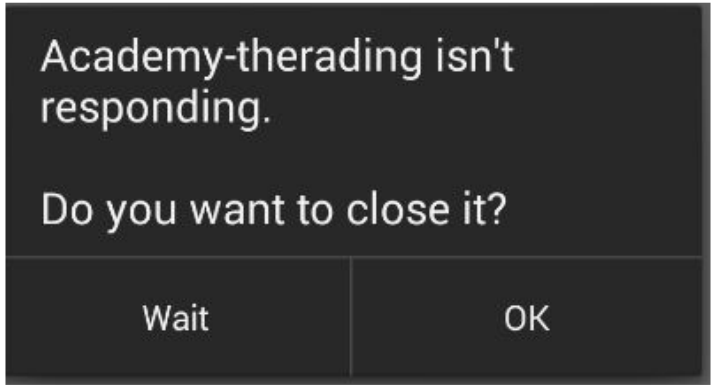





Process
Main Thread (UI Thread)
Input Event
Animation
UI Drawing
UI Drawing
UI Drawing
UI Drawing
16ms
16ms
16ms
UI Timeline


Process
Main Thread (UI Thread)
Input Event
Animation
UI Drawing
Calculation
UI Drawing
UI Drawing
16ms
32ms
Dropped Frame
UI Timeline

ANR
Application Not Responding



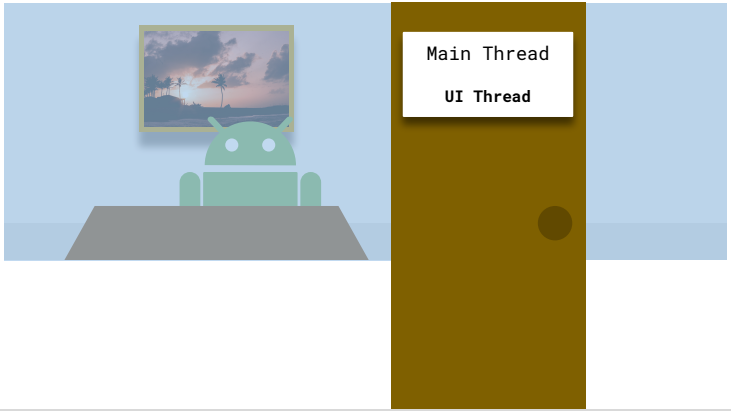
Any Questions?

What can We do?!

Process

Main Thread
UI Thread

Worker Thread



Android Multi-thread Options

- Thread
- Executor 
- HandlerThread
- AsyncTask
- Loaders
- Libraries

Android Multi-thread Options

- Thread
- Executor
- HandlerThread
- AsyncTask
- Coroutines
- Libraries

Custom Thread

```
public class CustomThread
```

Custom Thread

```
public class CustomThread extends Thread {




}
```

Custom Thread

```
public class CustomThread extends Thread {
    @Override
    public void run() {

    }
}
```

Custom Thread

```
public class CustomThread extends Thread {
    @Override
    public void run() {
        // do Hard Work
    }
}
```

Custom Thread

```
public class CustomThread extends Thread {
    @Override
    public void run() {
        // do Hard Work
    }
}


new CustomThread().start();
```

Custom Runnable

```
public class CustomRunnable {



}
```

Custom Runnable

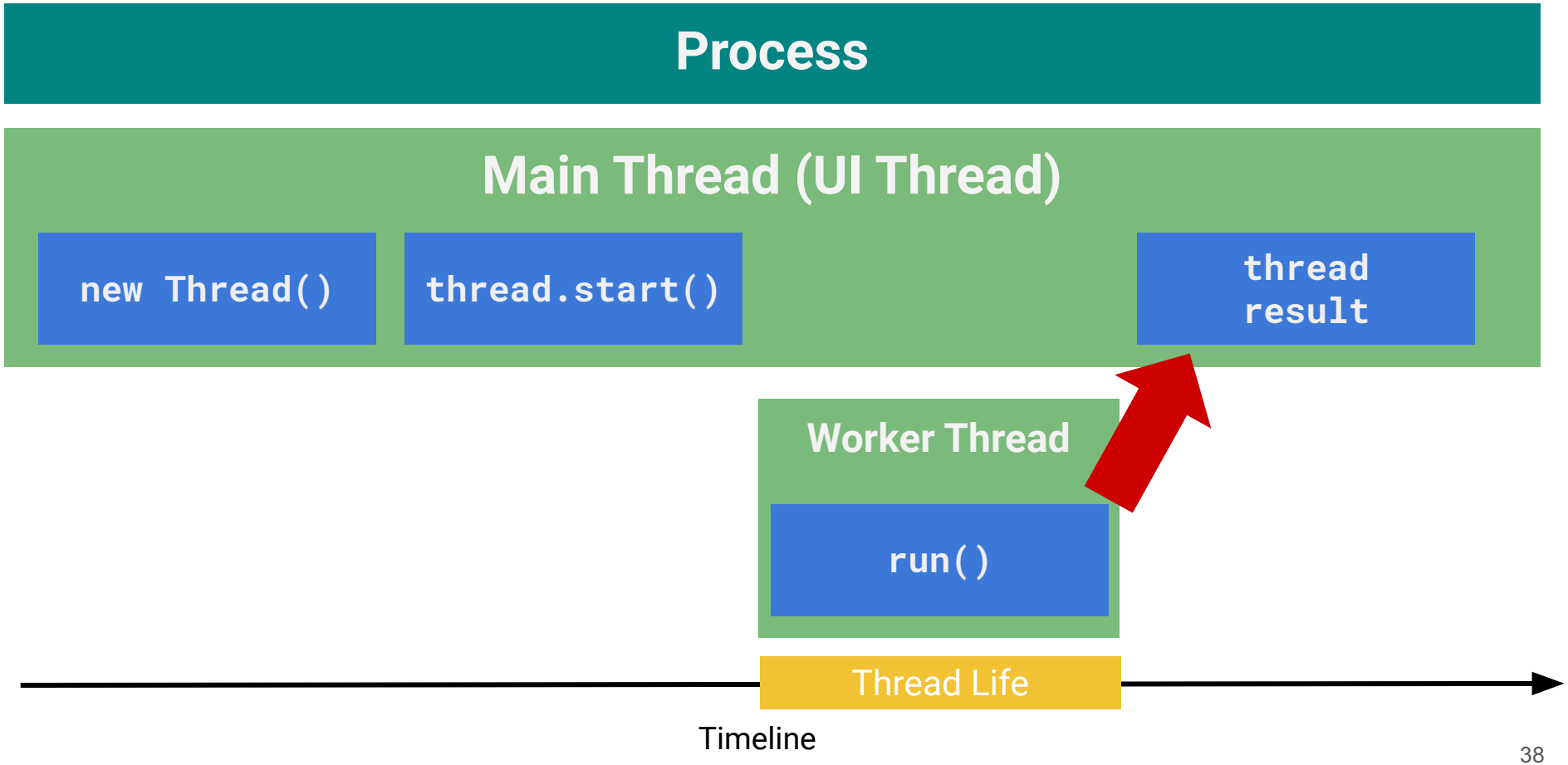
```
public class CustomRunnable implements Runnable {



}
```

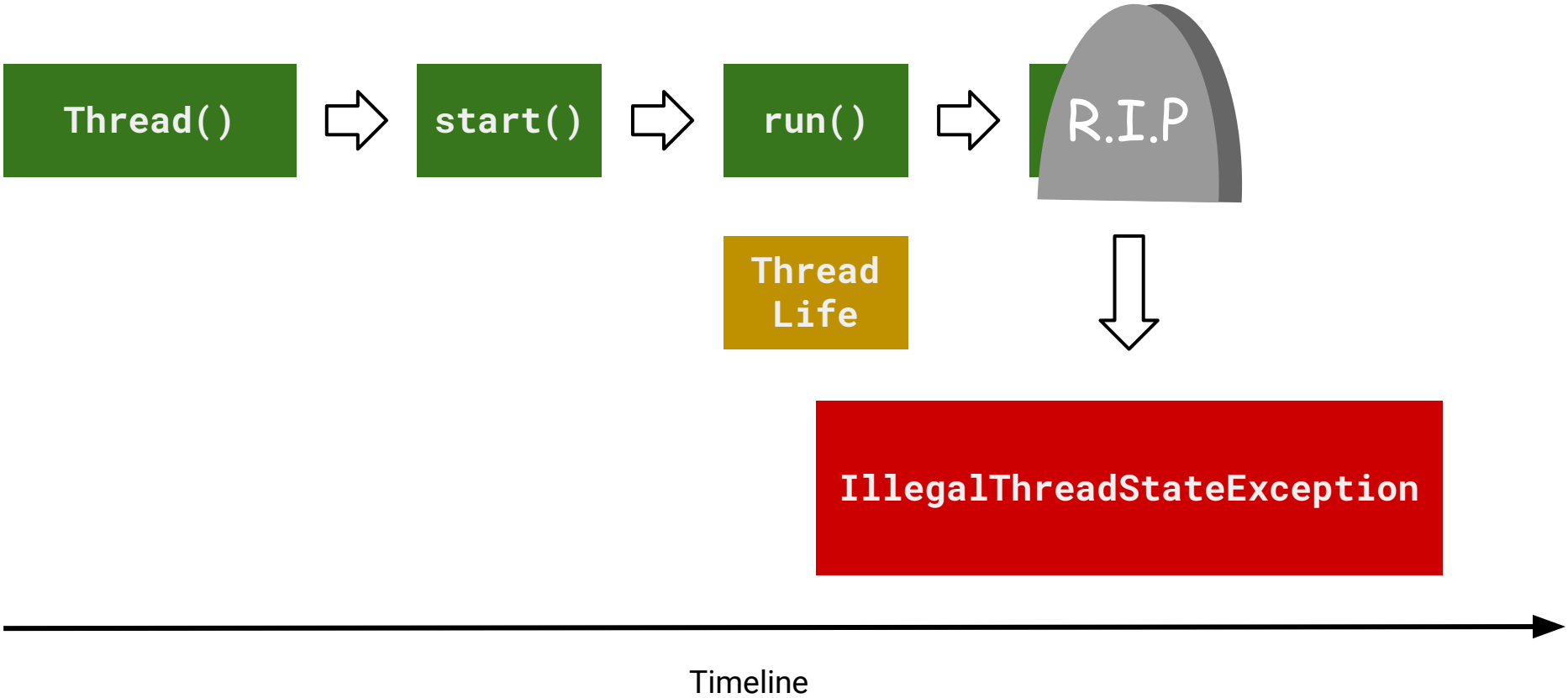
Custom Runnable

```
public class CustomRunnable implements Runnable {
    @Override
    public void run() {
        // do Hard Work
    }
}
```

Custom Runnable

```
public class CustomRunnable implements Runnable {
    @Override
    public void run() {
        // do Hard Work
    }
}


new Thread(new CustomRunnable()).start();
```


Process
Main Thread (UI Thread)
new Thread()
thread.start()
thread result
Worker Thread
run()
Thread Life
Timeline


Thread()
start()
run()
R.I.P
Thread Life
IllegalThreadStateException
Timeline

the thread life problem


ORBO

Loop Thread

```
public class CustomThread extends Thread {

    @Override
    public void run() {


            //doHardWork();

    }

}
```

Loop Thread

```
public class CustomThread extends Thread {

    @Override
    public void run() {

        while (true) {
            //doHardWork();
        }
    }
}
```

Loop Thread

```
public class CustomThread extends Thread {

    boolean running = false;

    @Override
    public void run() {
        while (true) {
            //doHardWork();
        }
    }


}
```

Loop Thread

```
public class CustomThread extends Thread {

    boolean running = false;

    @Override
    public void run() {
        running = true;
        while (running) {
            //doHardWork();
        }
    }

}
```

Loop Thread

```
public class CustomThread extends Thread {

    boolean running = false;

    @Override
    public void run() {
        running = true;
        while (running) {
            //doHardWork();
        }
    }

    public void cancel() {
        running = false;
    }
}
```

Loop Thread

```java
public class CustomThread extends Thread {

    boolean running = false;

    @Override
    public void run() {
        running = true;
        while (running) {
            //doHardWork();
        }
    }

    public void cancel() {
        running = false;
    }
}
```

```java
CustomThread thread = new
CustomThread();
```

Loop Thread

```
public class CustomThread extends Thread {

    boolean running = false;

    @Override
    public void run() {
        running = true;
        while (running) {
            //doHardWork();
        }
    }

    public void cancel() {
        running = false;
    }
}
```

```
CustomThread thread = new
CustomThread();
thread.start();
```

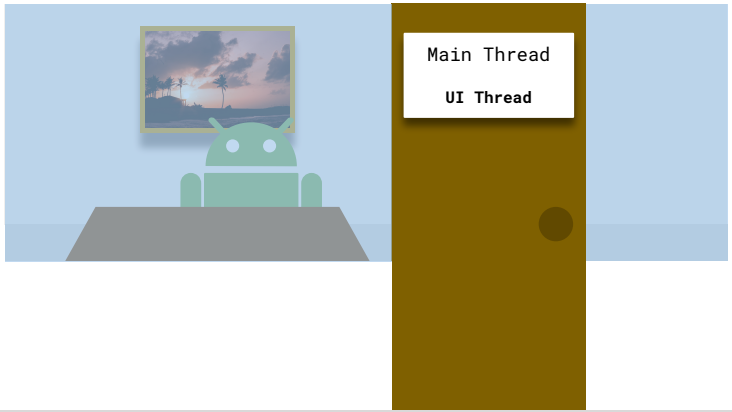
Loop Thread

```java
public class CustomThread extends Thread {

    boolean running = false;

    @Override
    public void run() {
        running = true;
        while (running) {
            //doHardWork();
        }
    }

    public void cancel() {
        running = false;
    }
}
```

```java
CustomThread thread = new
CustomThread();
thread.start();
thread.cancel();
```

Thread Factory


Executor
Outsource Company


Main Thread
UI Thread

Executor

- **An object that executes submitted Runnable tasks.**

Executor

- An object that executes submitted Runnable tasks.
- **Decouples task submission from the mechanics of how each task will be run.**

Thread Way

```
public class CustomRunnable implements Runnable {
    @Override
    public void run() {
        //do Hard Work();
    }
}
```

Thread Way

```java
public class CustomRunnable implements Runnable {
    @Override
    public void run() {
        //do Hard Work();
    }
}


Thread thread = new Thread(new CustomRunnable());

thread.start();
```

Executor Way

```
public class CustomRunnable implements Runnable {
    @Override
    public void run() {
        //do Hard Work();
    }
}
```

Executor Way

```
public class CustomRunnable implements Runnable {
    @Override
    public void run() {
        //do Hard Work();
    }
}


Executor executor = Executors. /* creation method */;
executor.execute(new CustomRunnable());
```

Executor Factory Methods

- `Executors.newCachedThreadPool()`

Executor Factory Methods

- `Executors.newCachedThreadPool()` ... ∞
- **`Executors.newFixedThreadPool(n: Int)`** ... **number**

Executor Factory Methods

- `Executors.newCachedThreadPool()` ... ∞
- `Executors.newFixedThreadPool(n: Int)` ... **number**
- **`Executors.newSingleThreadExecutor()`**

Preferred Usage

```
public class AppExecutors {

}
```

https://github.com/googlesamples/android-architecture-components/blob/master/PersistenceMigrationsSample/app/src/main/java/com/example/android/persistence/migrations/AppExecutors.java

Preferred Usage

```
public class AppExecutors {


    private final Executor diskIO




}
```

https://github.com/googlesamples/android-architecture-components/blob/master/PersistenceMigrationsSample/app/src/main/java/com/example/android/persistence/migrations/AppExecutors.java

Preferred Usage

```
public class AppExecutors {


    private final Executor diskIO = Executors.newSingleThreadExecutor();




}
```

https://github.com/googlesamples/android-architecture-components/blob/master/PersistenceMigrationsSample/app/src/main/java/com/example/android/persistence/migrations/AppExecutors.java

Preferred Usage

```
public class AppExecutors {


    private final Executor diskIO = Executors.newSingleThreadExecutor();

    private final Executor networkIO = Executors.newFixedThreadPool(THREAD_COUNT);


}
```

https://github.com/googlesamples/android-architecture-components/blob/master/PersistenceMigrationsSample/app/src/main/java/com/example/android/persistence/migrations/AppExecutors.java

Preferred Usage

```
public class AppExecutors {

    private static final int THREAD_COUNT = 3;

    private final Executor diskIO = Executors.newSingleThreadExecutor();

    private final Executor networkIO = Executors.newFixedThreadPool(THREAD_COUNT);



}
```

https://github.com/googlesamples/android-architecture-components/blob/master/PersistenceMigrationsSample/app/src/main/java/com/example/android/persistence/migrations/AppExecutors.java

Preferred Usage

```
public class AppExecutors {

    private static final int THREAD_COUNT = 3;

    private final Executor diskIO = Executors.newSingleThreadExecutor();

    private final Executor networkIO = Executors.newFixedThreadPool(THREAD_COUNT);

    private final Executor mainThread = new MainThreadExecutor();

}
```

https://github.com/googlesamples/android-architecture-components/blob/master/PersistenceMigrationsSample/app/src/main/java/com/example/android/persistence/migrations/AppExecutors.java

Example

```
public static class CustomRunnable implements Runnable {
    @Override
    public void run() {
        //doHardWork();
    }
}
```

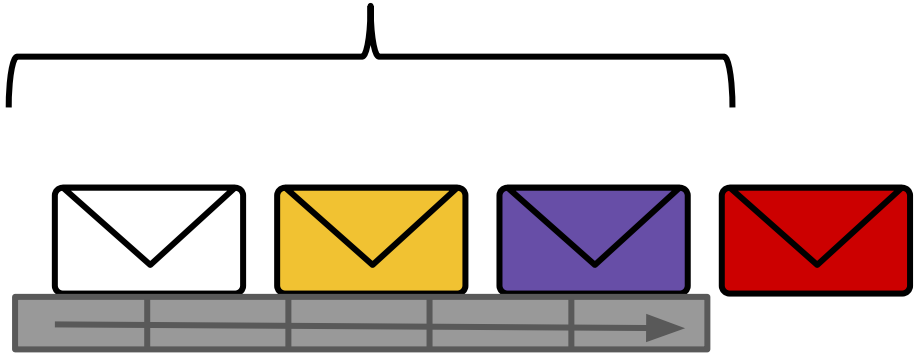
Example

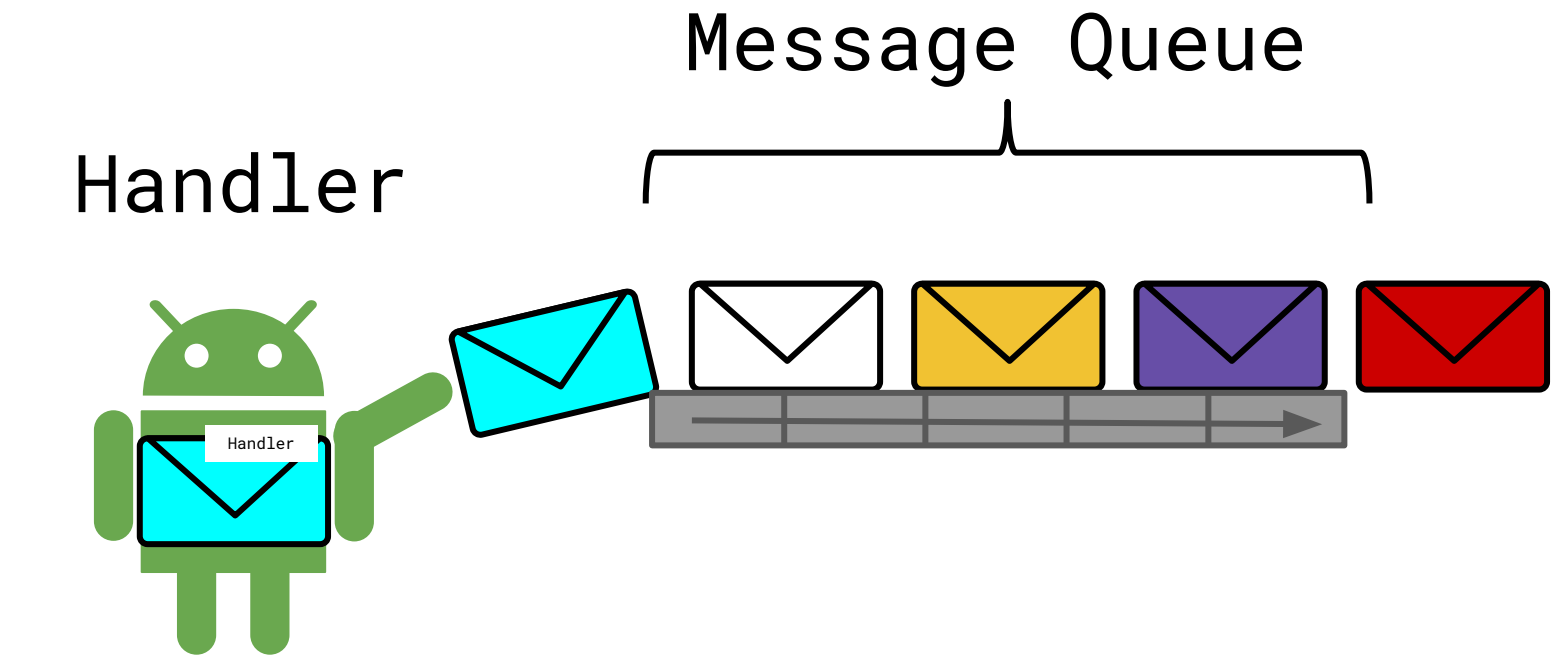
```java
public static class CustomRunnable implements Runnable {
    @Override
    public void run() {
        //doHardWork();
    }
}
```

```java
executor.diskIO().execute( new CustomRunnable());
```

How does the main thread stay alive?

Message Queue

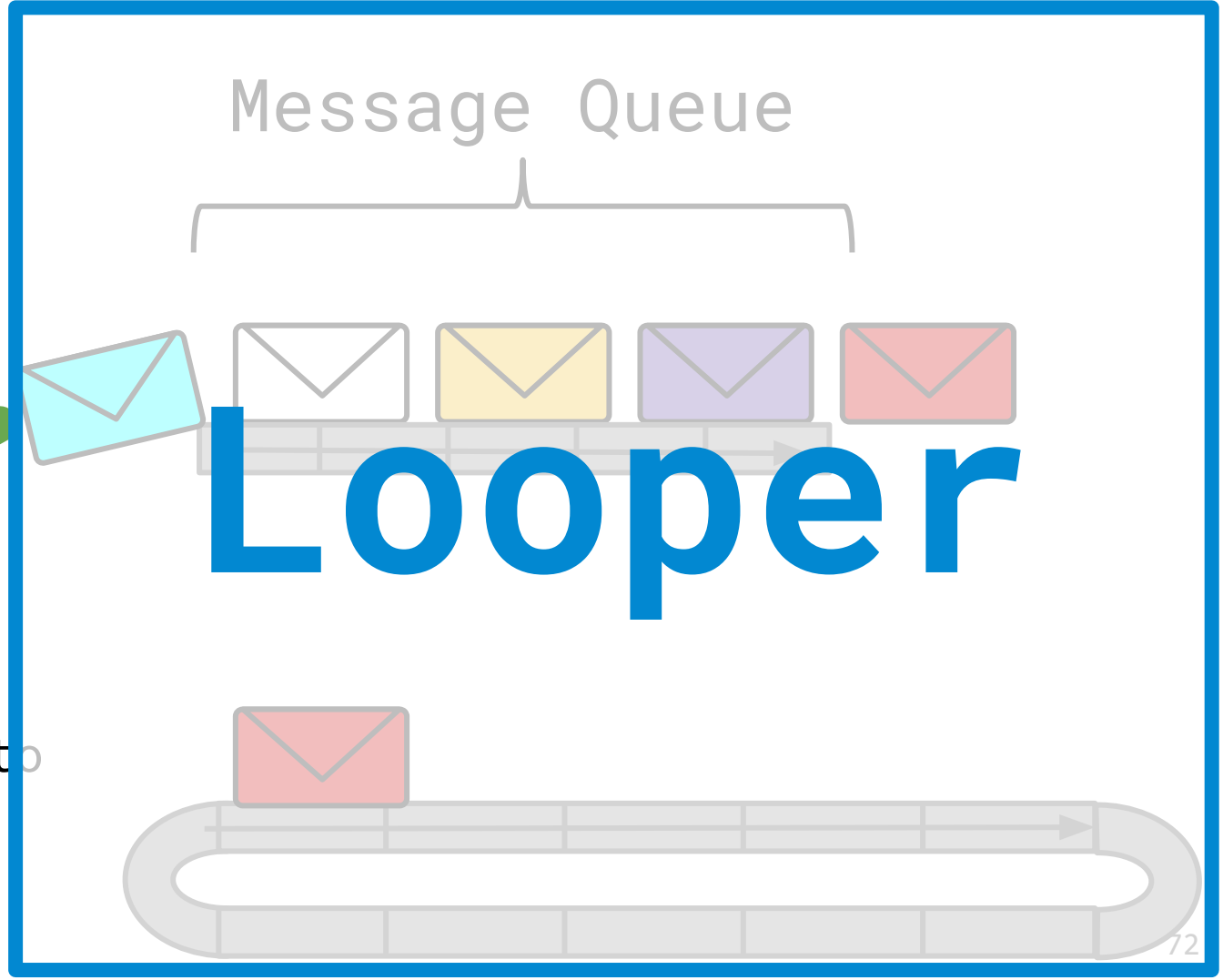




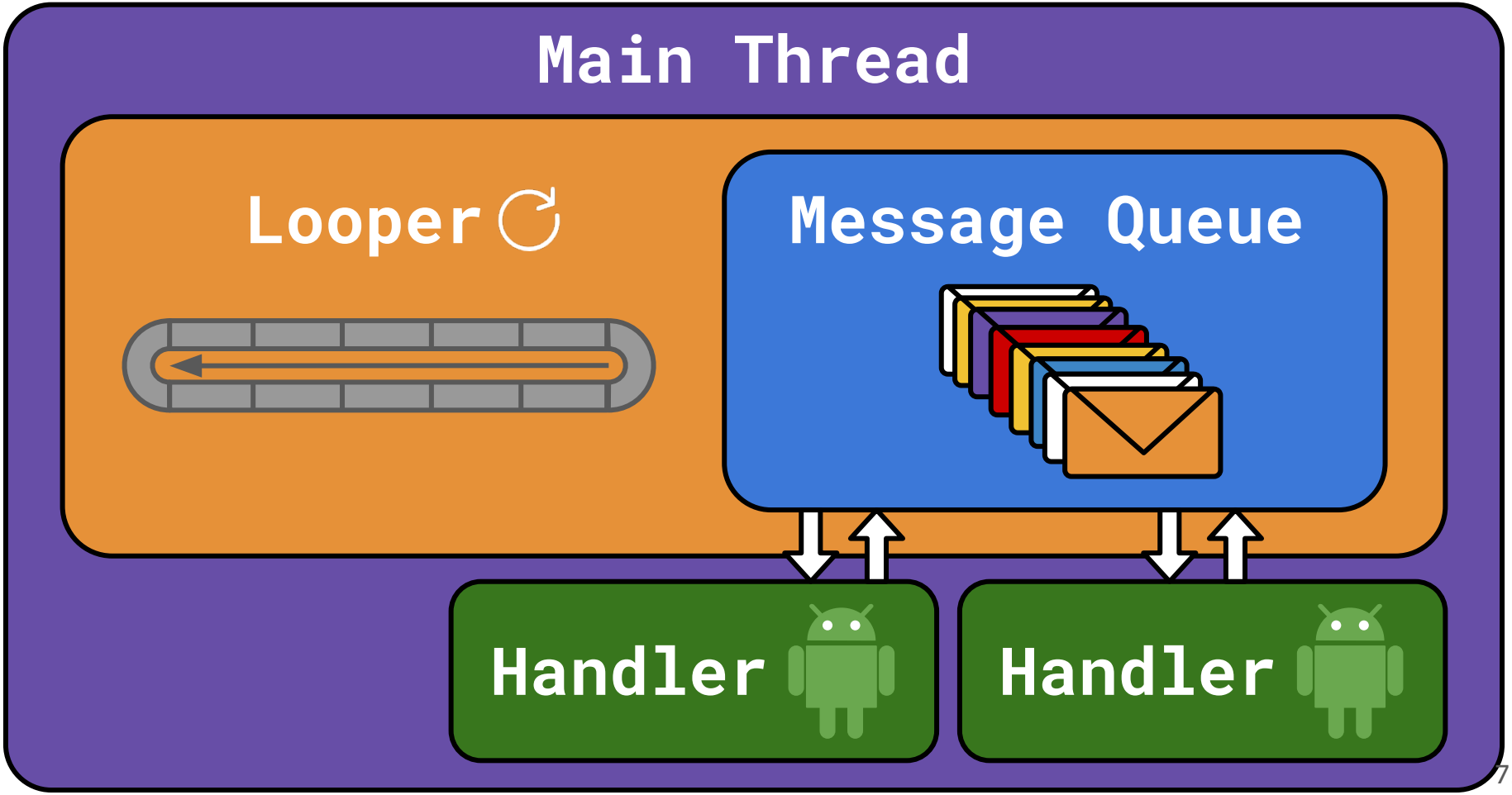
Add Messages to
the queue

Handler

Add Messages to the queue



Process



Any Questions?

Main thread communication

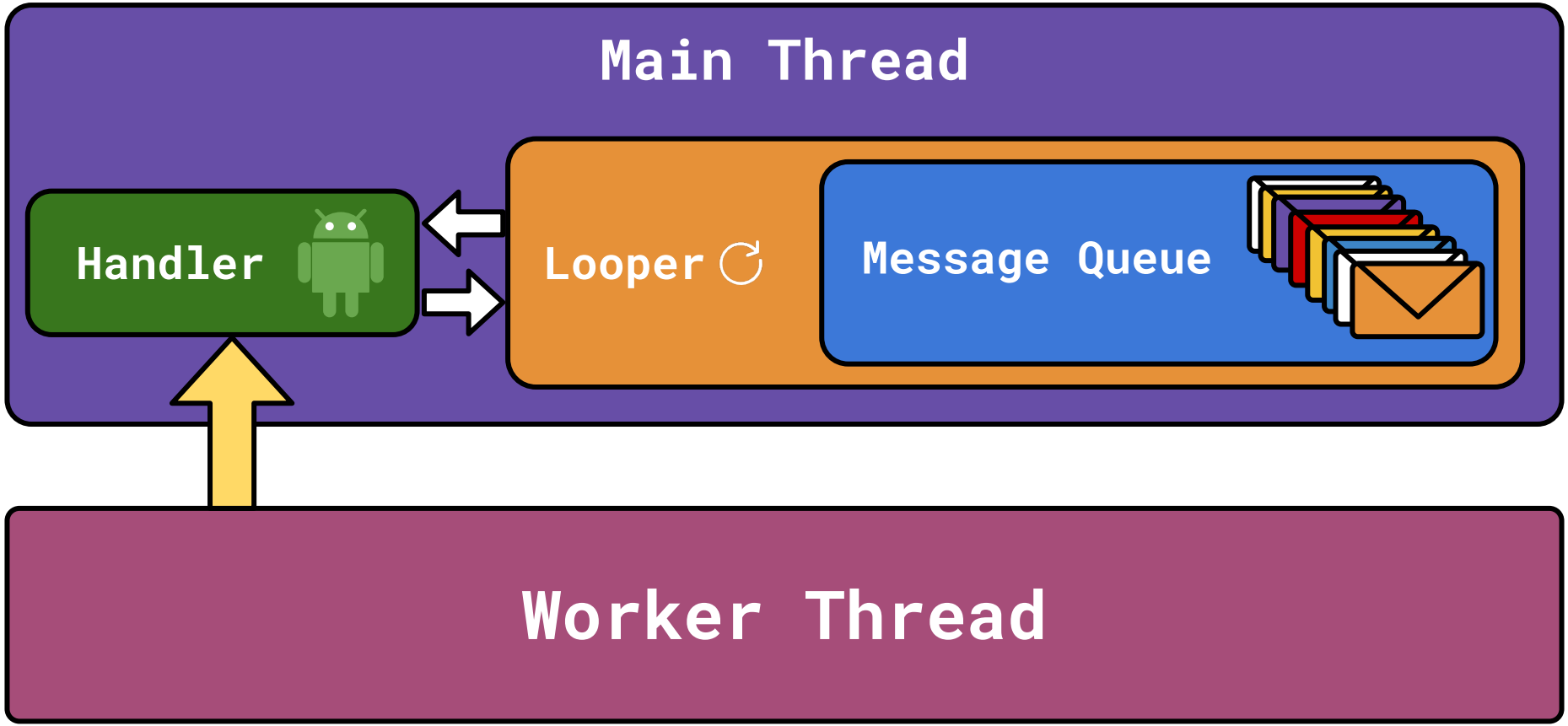
UI updates

```
public class CustomRunnable extends Runnable {
    @Override
    public void run() {
        // do Hard Work
        someTextView.setText("done!");
    }
}
```

UI updates

```
public class CustomR      le ex      Runnable {
    @Override
    public void run() {
        // do Hard Work
        someTextView.s      t(" "      ");
    }
}
```

Process



Handler Communication

- `handler.post(Runnable runnable);`

Handler Communication

- `handler.post(Runnable runnable);`
- `handler.postAtFrontOfQueue(Runnable runnable);`

Handler Communication

- `handler.post(Runnable runnable);`
- `handler.postAtFrontOfQueue(Runnable runnable);`
- `handler.postDelayed(Runnable runnable,Long dellayMillis);`

Handler Communication

- `handler.post(Runnable runnable);`
- `handler.postAtFrontOfQueue(Runnable runnable);`
- `handler.postDelayed(Runnable runnable,Long dellayMillis);`
- `handler.postAtTime(Runnable runnable,Long uptimeMillis);`

Wrappers

- `activity.runOnUiThread(Runnable runnable);`

Wrappers

- `activity.runOnUiThread(Runnable runnable);`
- `anyView.post(Runnable runnable);`

General

```
new Handler(Looper.getMainLooper()).post(Runnable runnable);
```

Main Thread Rules

#1 Rule in Android development

Never block the UI Thread

Main Thread Rules

- **Never block the UI thread.**

Main Thread Rules

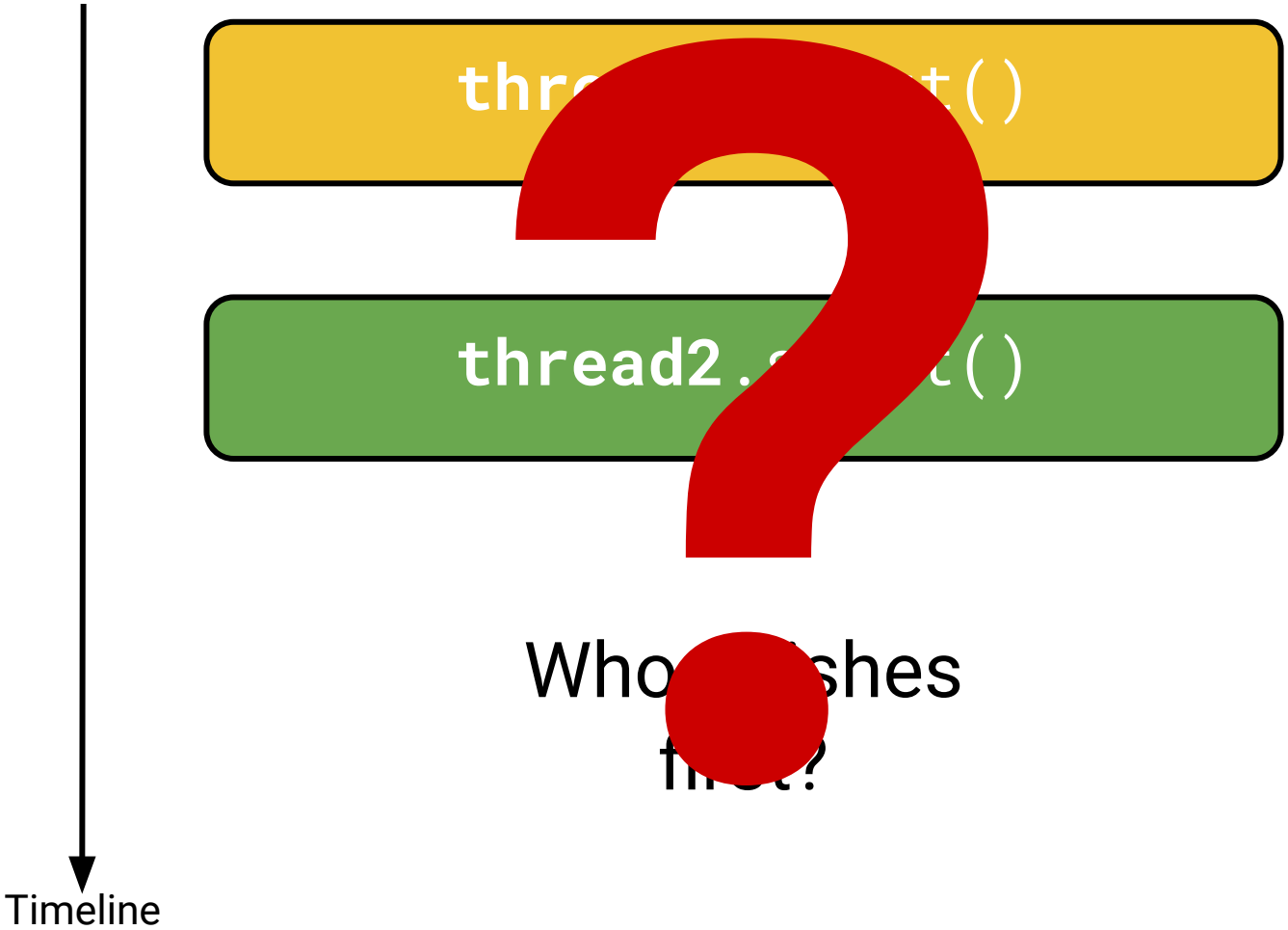
- Never block the UI thread.
- **Do not call UI components from a worker thread.**

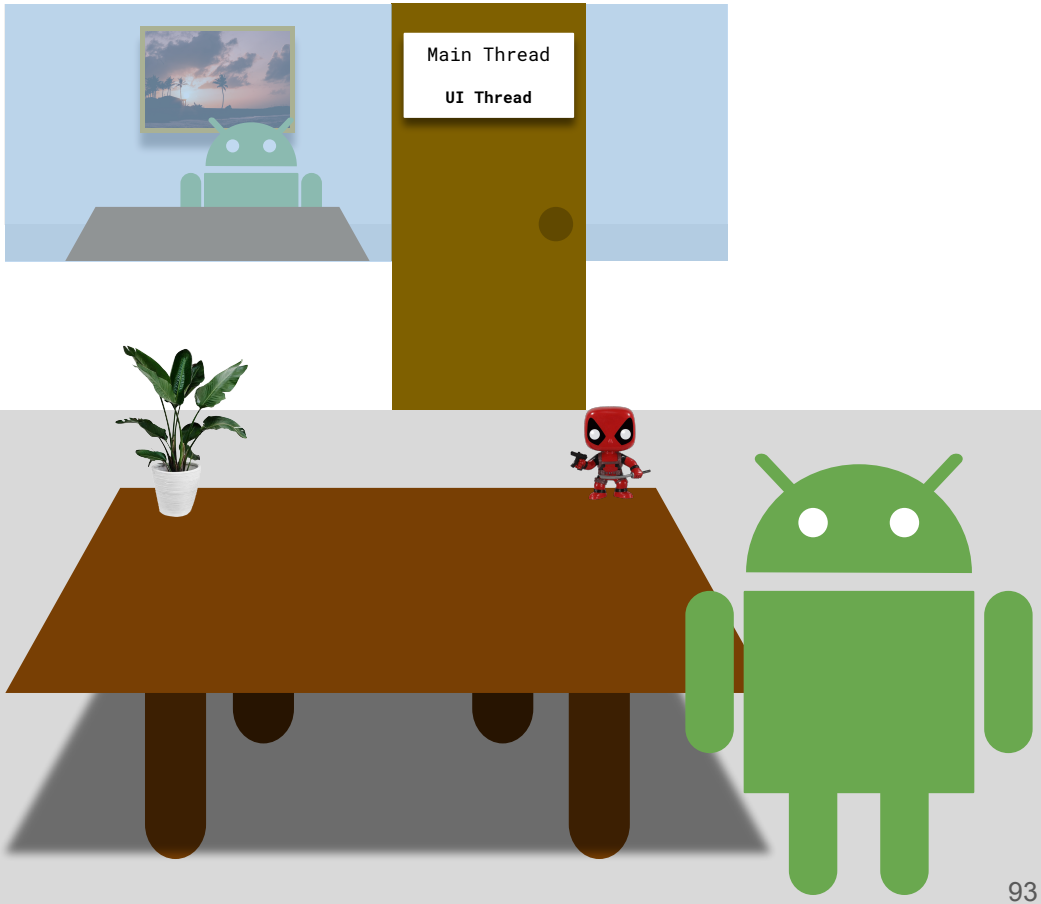
Main Thread Rules

- Never block the UI thread.
- Do not call UI components from a worker thread.
- **Tasks that are not UI related will not run on the UI.**

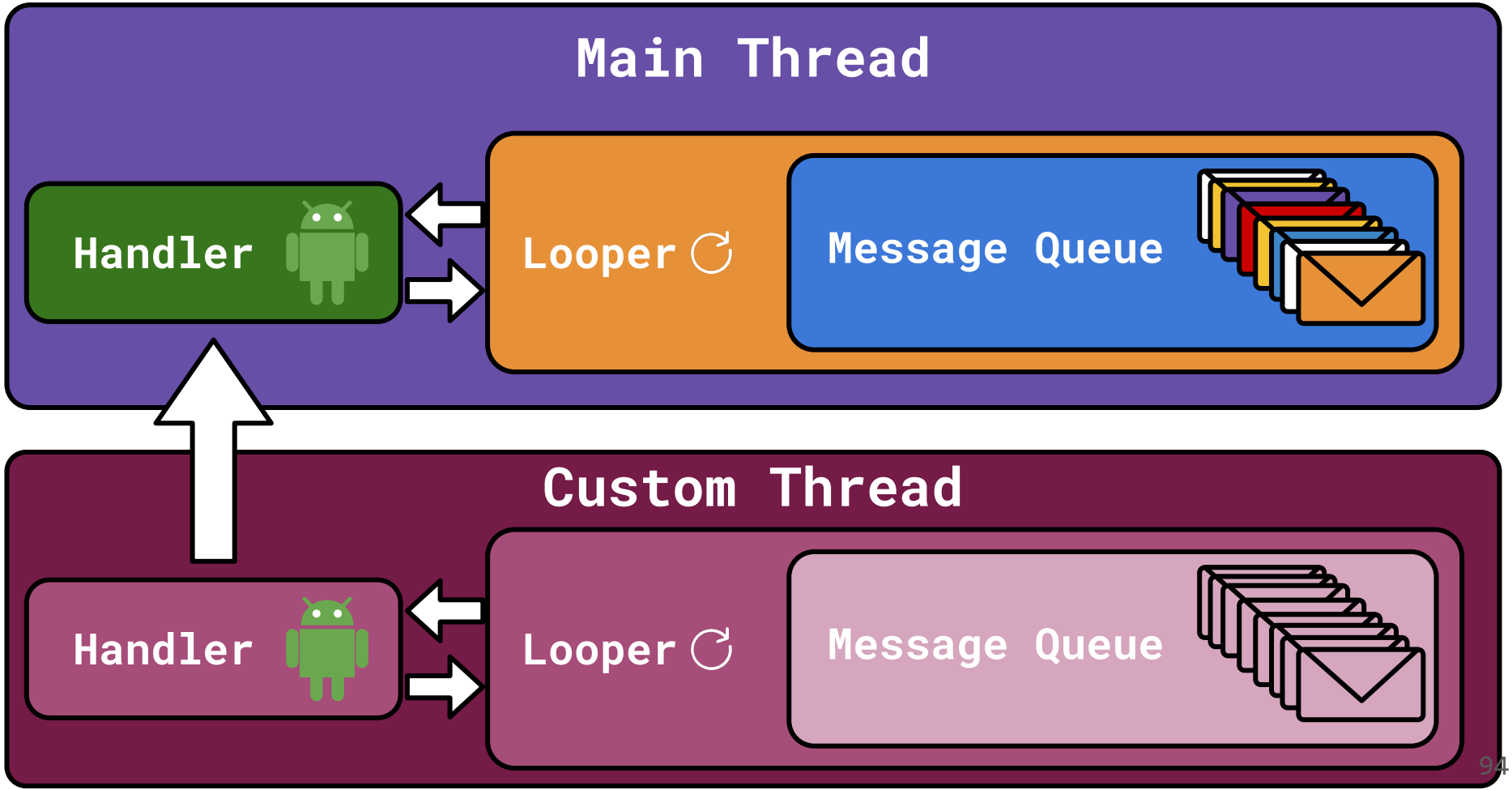
Any Questions?

running sequentially


thre t()
thread2. t()
Who shes
first?
Timeline

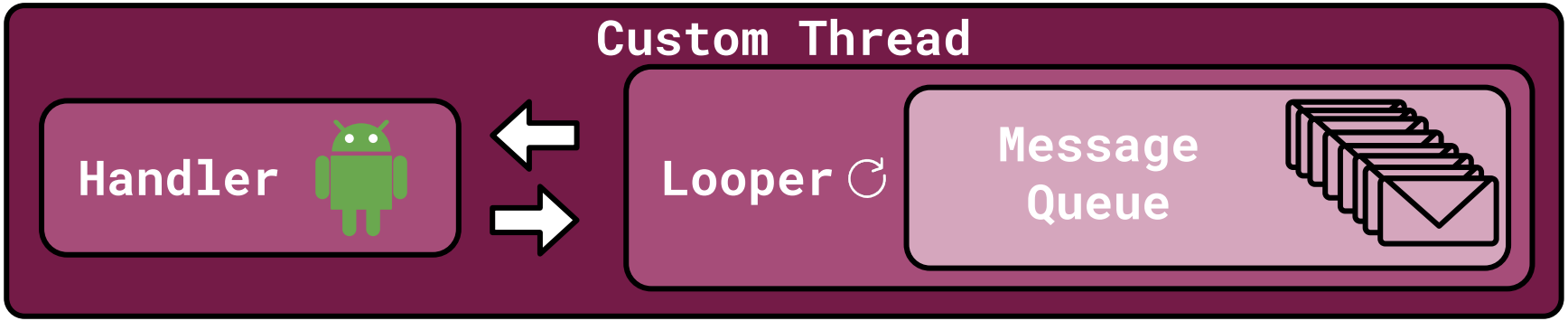

Main Thread
UI Thread

Process


Main Thread
Handler
Looper
Message Queue
Custom Thread
Handler
Looper
Message Queue

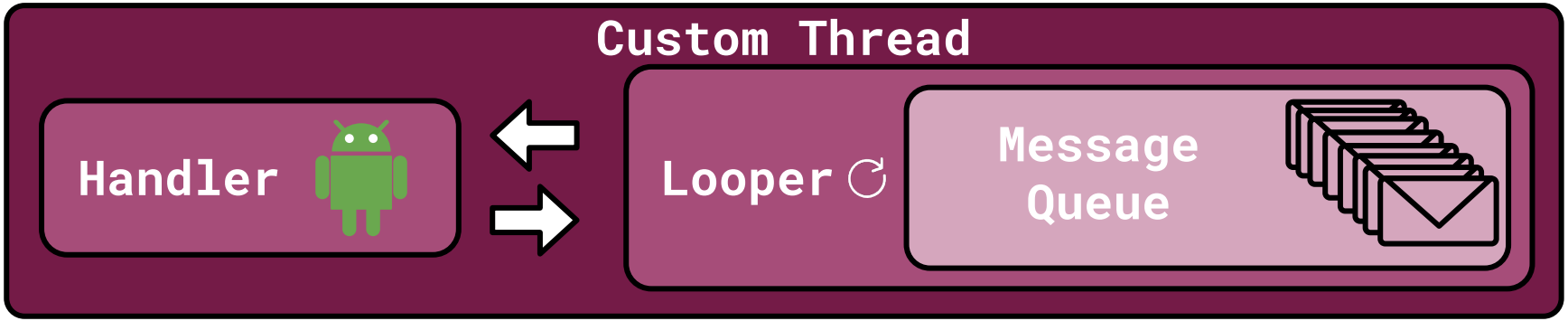
HandlerThread

```
HandlerThread handlerThread = new HandlerThread("worker");
handlerThread.start();
```



HandlerThread

```
HandlerThread handlerThread = new HandlerThread("worker");
handlerThread.start();
Looper looper = handlerThread.getLooper();
Handler handler = new Handler(looper);
```



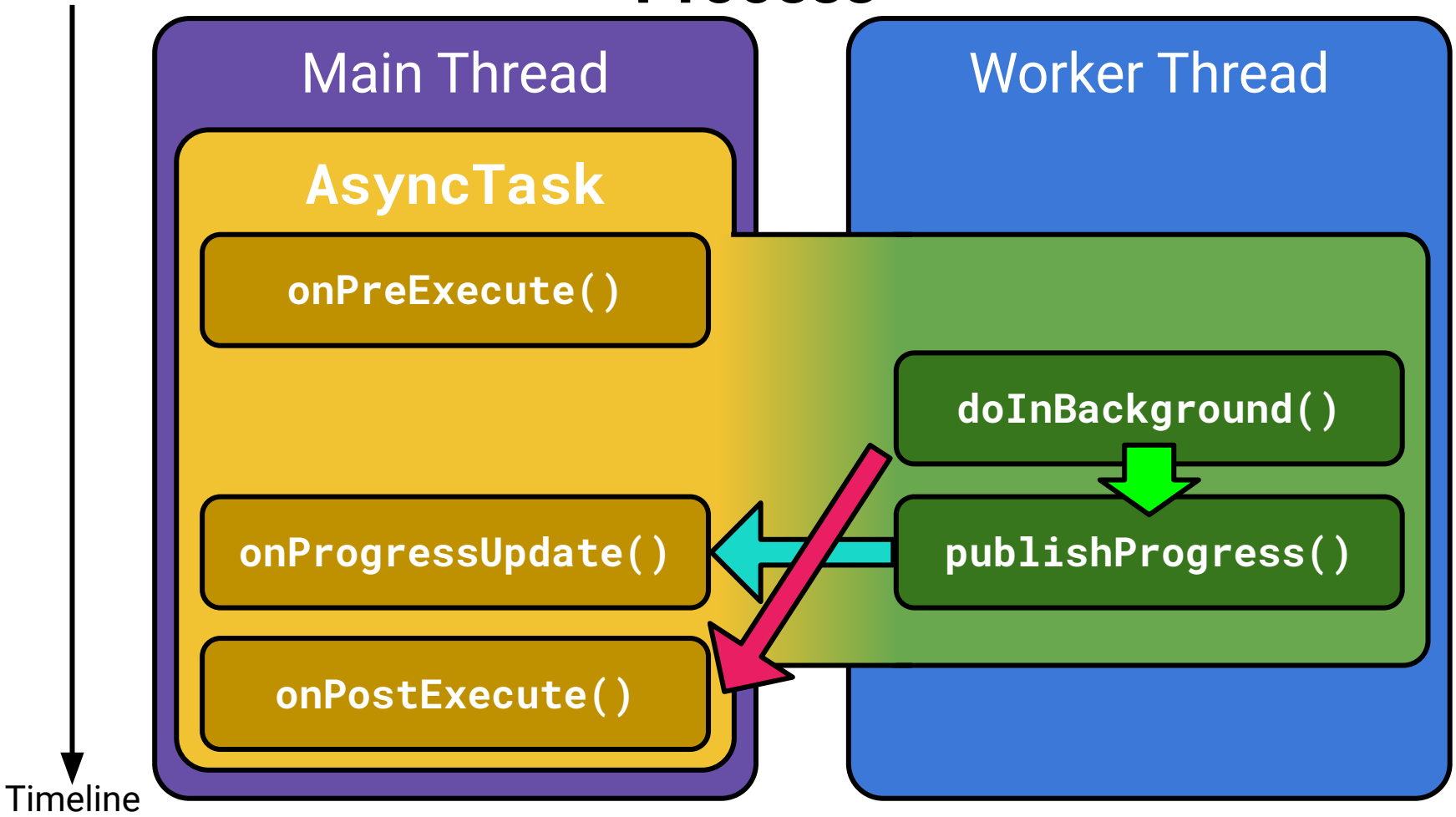
Handler Communication

- `handler.post(Runnable runnable);`
- `handler.postAtFrontOfQueue(Runnable runnable);`
- `handler.postDelayed(Runnable runnable,Long dellayMillis);`
- `handler.postAtTime(Runnable runnable,Long uptimeMillis);`

Any Questions?

Thread Abstraction

Process


Main Thread
Worker Thread
AsyncTask
onPreExecute()
doInBackground()
onProgressUpdate()
publishProgress()
onPostExecute()
Timeline

AsyncTask Initialization

```
AsyncTask<Params, Progress, Result>
```

AsyncTask Initialization

```
AsyncTask<

  Params,  →doInBackground(Params... params)

Progress,→onProgressUpdate(Progress... values)

  Result> →onPostExecute(Result result)
```

AsyncTask Initialization

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {

}
```

AsyncTask Initialization

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
    }

}
```

AsyncTask Initialization

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
    }
    @Override
    protected String doInBackground(Integer... integers) {
    }
}
```

AsyncTask Initialization

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
    }
    @Override
    protected String doInBackground(Integer... integers) {
    }
    @Override
    protected void onProgressUpdate(Integer... values) {
        super.onProgressUpdate(values);
    }
}
```

AsyncTask Initialization

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
    }
    @Override
    protected String doInBackground(Integer... integers) {
    }
    @Override
    protected void onProgressUpdate(Integer... values) {
        super.onProgressUpdate(values);
    }
    @Override
    protected void onPostExecute(String s) {
        super.onPostExecute(s);
    }
}
```

AsyncTask Initialization

```
public class MainActivity extends AppCompatActivity {
...

   public void startAsyncTask() {



   }

}
```

AsyncTask Initialization

```
public class MainActivity extends AppCompatActivity {
...

    public void startAsyncTask() {
        ExampleAsyncTask task = new ExampleAsyncTask();
        task.execute(10);
    }

}
```

AsyncTask Initialization

```
public class MainActivity extends AppCompatActivity {
    private ProgressBar progressBar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout. activity_main);
        progressBar = findViewById(R.id. progress_bar);
    }

    public void startAsyncTask() {
        ExampleAsyncTask task =  new ExampleAsyncTask();
        task.execute(10);
    }
}
```

AsyncTask Initialization

```
public class MainActivity extends AppCompatActivity {
    private ProgressBar progressBar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout. activity_main);
        progressBar = findViewById(R.id. progress_bar);
    }

    public void startAsyncTask() {
        ExampleAsyncTask task =  new ExampleAsyncTask(progressBar);
        task.execute( 10);
    }
}
```

AsyncTask Initialization

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
    private ProgressBar progressBar;
    ExampleAsyncTask(ProgressBar progressBar) {
        this.progressBar = progressBar;
    }
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        progressBar.setVisibility(View. VISIBLE);
    }
    @Override
    protected void onProgressUpdate(Integer... values) {
        super.onProgressUpdate(values);
        progressBar.setProgress(values[ 0]);
    }
}
```

AsyncTask Initialization

```
public static class ExampleAsy[illegible]sk extends AsyncTask<Integer[illegible]eger, String> {
    private ProgressBar p[illegible];
    ExampleAsyncTask(Progres[illegible]essBar) {
        this.progressBar = prog[illegible]
    }
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        progressBar.setVisibility(Vie[illegible]
    }
    @Override
    protected void onProgres[illegible]teger... values)
        super.onProgressU[illegible]es);
        progressBar.setProgr[illegible](values[0]);
    }
}
```

Garbage Collection

Collects and removes unreferenced objects.

Memory Leak

Process

Main Thread

onCreate@Activity1

onPreExecute@AsyncTask1

onDestroy@Activity1

onCreate@Activity2

onPreExecute@AsyncTask2

onPostExecute@AsyncTask1

onPostExecute@AsyncTask2

Worker Thread

doInBackground@AsyncTask1

doInBackground@AsyncTask2

Timeline

Process

Main Thread

onCreate@Activity1

onPreExecute@AsyncTask1

onDestroy@Activity1

onCreate@Activity2

onPreExecute@AsyncTask2

onPostExecute@AsyncTask1

onPostExecute@AsyncTask2

Worker Thread

doInBackground@AsyncTask1

```
protected void onPostExecute(String s) {
        activity.progressBar.setProgress(0);
}
```

Memory Leak!

doInBackground@AsyncTask2

Timeline

AsyncTask Initialization

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
    private WeakReference




}
```

AsyncTask Initialization

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
    private WeakReference<MainActivity> activityWeakReference;

}
```

AsyncTask Initialization

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
    private WeakReference<MainActivity> activityWeakReference;

    ExampleAsyncTask(MainActivity activity) {
        activityWeakReference = new WeakReference<MainActivity>(activity);
    }

}
```

AsyncTask Initialization

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
    ...
    @Override
    protected void onPreExecute() {
        super.onPreExecute();

        MainActivity activity = activityWeakReference.get();









    }
...
}
```

AsyncTask Initialization

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
    ...
    @Override
    protected void onPreExecute() {
        super.onPreExecute();

        MainActivity activity = activityWeakReference.get();
        if (activity == null || activity.isFinishing()) {
            return;
        }



    }
...
}
```

AsyncTask Initialization

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
    ...
    @Override
    protected void onPreExecute() {
        super.onPreExecute();

        MainActivity activity = activityWeakReference.get();
        if (activity == null || activity.isFinishing()) {
            return;
        }

        activity.progressBar.setVisibility(View.VISIBLE);
    }
...
}
```

AsyncTask Initialization

```
public static class ExampleAsyncTask extends AsyncTask<
                Integer,    →  doInBackground(Int.. params)
                Integer,    →  onProgressUpdate(Int... int)
                String>     →  onPostExecute(String result)
{
}
```

Preparation | optional

UI Thread

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
    ...
    @Override
    protected void onPreExecute() {
        super.onPreExecute();

        MainActivity activity = activityWeakReference.get();
        if (activity == null || activity.isFinishing()) {
            return;
        }

        activity.progressBar.setVisibility(View.VISIBLE);
    }
...
}
```

Background Work

Worker Thread

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
    ...
    @Override
    protected String doInBackground(Integer... integers) {



    }
```

Background Work

Worker Thread

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
    ...
    @Override
    protected String doInBackground(Integer... integers) {









        return "Finished!";
    }
```

Background Work

Worker Thread

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
    ...
    @Override
    protected String doInBackground(Integer... integers) {
        for (int i = 0; i < integers[0]; i++) {
            publishProgress((i * 100) / integers[0]);









        }

        return "Finished!";
    }
```

Background Work

Worker Thread

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
    ...
    @Override
    protected String doInBackground(Integer... integers) {
        for (int i = 0; i < integers[0]; i++) {
            publishProgress((i * 100) / integers[0]);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        return "Finished!";
    }
```

Progress Update | Optional

UI Thread

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
  ...
   @Override
   protected void onProgressUpdate(Integer... values) {
       super.onProgressUpdate(values);









   }

}
```

Progress Update | Optional

UI Thread

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
  ...
   @Override
   protected void onProgressUpdate(Integer... values) {
       super.onProgressUpdate(values);









       activity.progressBar.setProgress(values[0]);
   }

}
```

Progress Update | Optional

UI Thread

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
  ...
   @Override
   protected void onProgressUpdate(Integer... values) {
       super.onProgressUpdate(values);

       MainActivity activity = activityWeakReference.get();
       if (activity == null || activity.isFinishing()) {
           return;
       }

       activity.progressBar.setProgress(values[0]);
   }

}
```

Result | Optional

UI Thread

```
public static class ExampleAsyncTask  extends AsyncTask<Integer, Integer, String> {
  ...
   @Override
   protected void onPostExecute(String s) {
       super.onPostExecute(s);
   }
}
```

Result | Optional

UI Thread

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
  ...
   @Override
   protected void onPostExecute(String s) {
       super.onPostExecute(s);

       MainActivity activity = activityWeakReference.get();
       if (activity == null || activity.isFinishing()) {
           return;
       }





   }
}
```

Result | Optional

UI Thread

```
public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {
  ...
   @Override
   protected void onPostExecute(String s) {
       super.onPostExecute(s);

       MainActivity activity = activityWeakReference.get();
       if (activity == null || activity.isFinishing()) {
           return;
       }

       Toast.makeText(activity, s, Toast.LENGTH_SHORT).show();
       activity.progressBar.setProgress(0);
       activity.progressBar.setVisibility(View.INVISIBLE);
   }
}
```

Usage

UI Thread

MoviesActivity.java

```
public void startAsyncTask() {


}
```

Usage

UI Thread

MoviesActivity.java

```
public void startAsyncTask() {
    ExampleAsyncTask task = new ExampleAsyncTask(this);

}
```

Usage

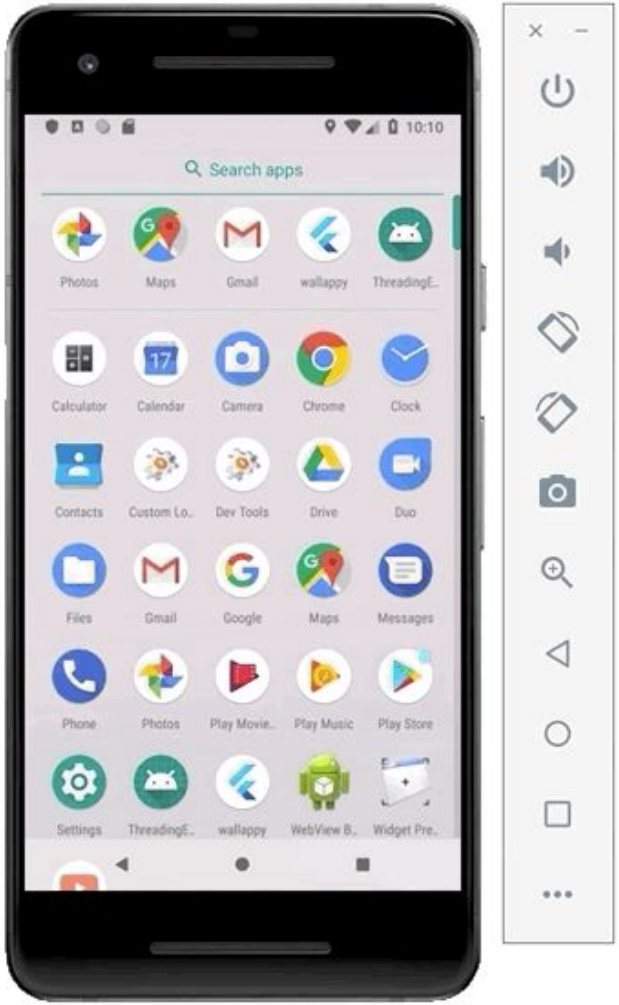
UI Thread

MoviesActivity.java

```
public void startAsyncTask() {
    ExampleAsyncTask task = new ExampleAsyncTask(this);
    task.execute(10);
}
```

Any Questions?

What about screen rotation?


10:10
Search apps
Photos
Maps
Gmail
wallappy
ThreadingE..
Calculator
Calendar
Camera
Chrome
Clock
Contacts
Custom Lo..
Dev Tools
Drive
Duo
Files
Gmail
Google
Maps
Messages
Phone
Photos
Play Movie..
Play Music
Play Store
Settings
ThreadingE..
wallappy
WebView B..
Widget Pre..

Process

Main Thread

onCreate@Activity1

onPreExecute@AsyncTask1

onDestroy@Activity1

onCreate@Activity2

onPreExecute@AsyncTask2

onPostExecute@AsyncTask1

onPostExecute@AsyncTask2

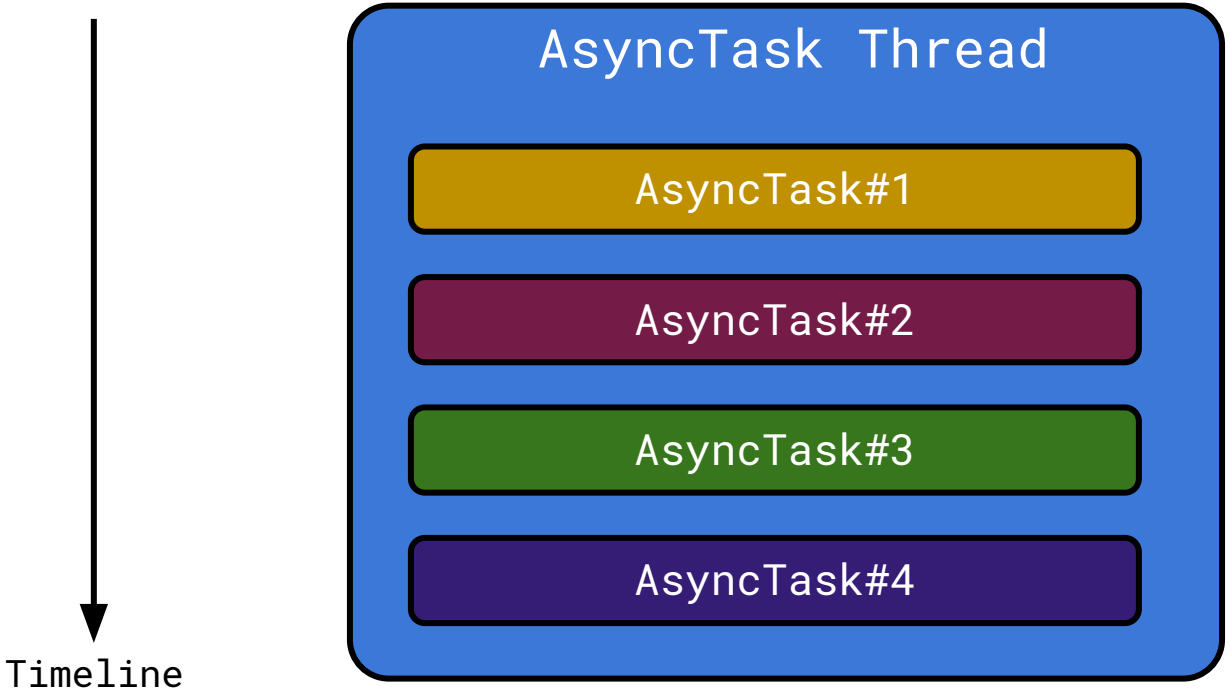
Worker Thread

doInBackground@AsyncTask1

doInBackground@AsyncTask2

Timeline

AsyncTask runs serially*

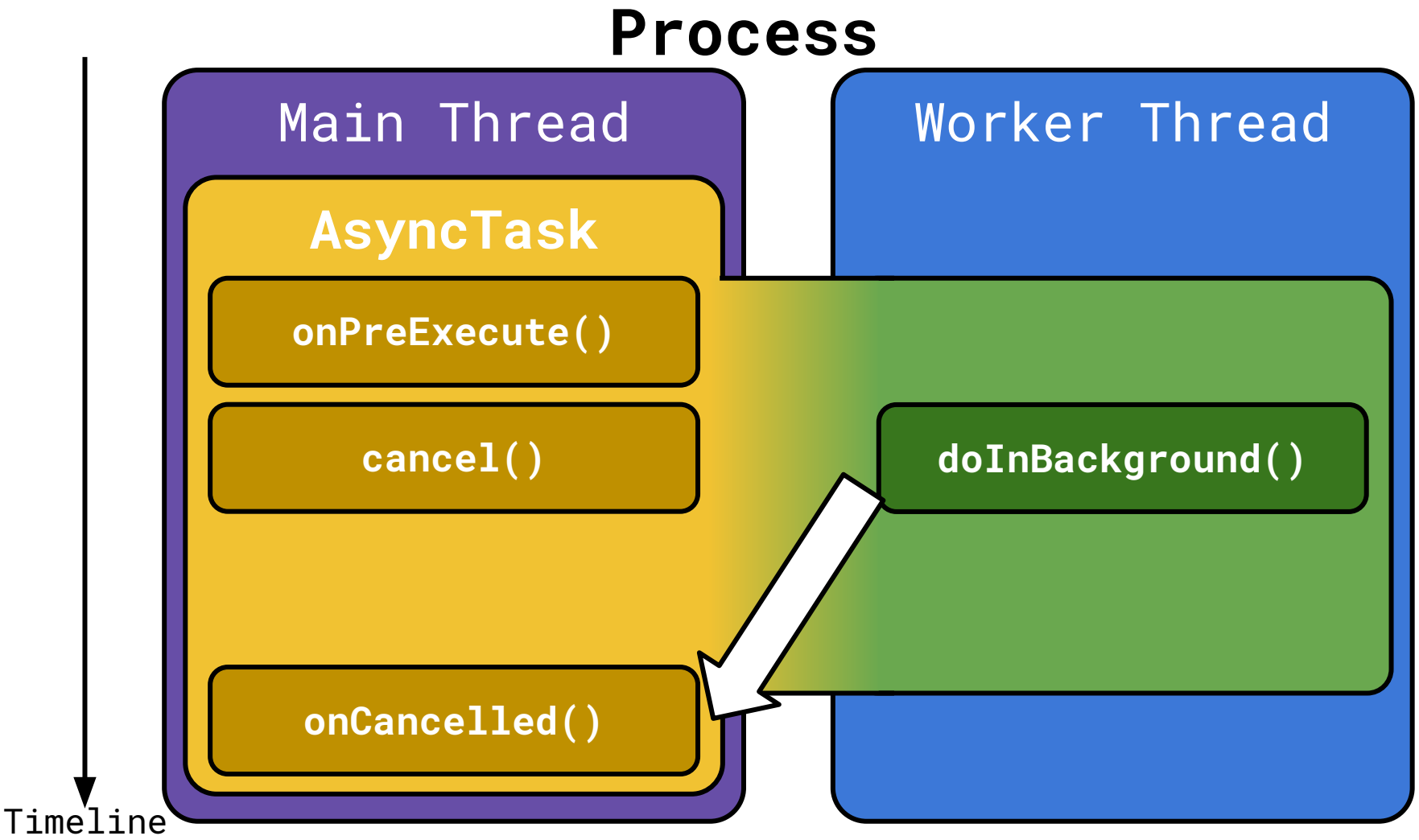


Cancellation

UI Thread

MoviesActivity.java

```
@Override
protected void onDestroy() {
    super.onDestroy();
    task.cancel(false / true);
}
```


Process
Main Thread
Worker Thread
AsyncTask
onPreExecute()
cancel()
doInBackground()
onCancelled()
Timeline

Cancellation

Worker Thread

```
@Override
protected String doInBackground(Integer... integers) {
    for (int i = 0; i < integers[0]; i++) {



        publishProgress((i * 100) / integers[0]);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    return "Finished!";
}
```

Cancellation

Worker Thread

```
@Override
protected String doInBackground(Integer... integers) {
    for (int i = 0; i < integers[0]; i++) {
        if (isCancelled()) {
            break;
        }
        publishProgress((i * 100) / integers[0]);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    return "Finished!";
}
```

Cancellation

UI Thread

```
@Override
protected void onCancelled() {
    super.onCancelled();

}
```

Cancellation

UI Thread

```
@Override
protected void onCancelled() {
    super.onCancelled();

    MainActivity activity = activityWeakReference.get();
    if (activity == null || activity.isFinishing()) {
        return;
    }
    Toast.makeText(activity, "An error occurred, " +
            "please try again later :(", Toast.LENGTH_SHORT).show();

}
```

Async Task Benefits

- **Threads abstraction.**

Async Task Benefits

- Threads abstraction.
- **Easy implementation.**

Async Task Benefits

- Threads abstraction.
- Easy implementation.
- **Easy thread communication.**

Async Task Rules

- **Create and execute on the UI thread.**

Async Task Rules

- Create and execute on the UI thread.
- **Do not call the methods manually! (onPreExecute, onPostExecute…).**

Async Task Rules

- Create and execute on the UI thread.
- Do not call the methods manually! (onPreExecute, onPostExecute…).
- **Can be executed only once.**

Memory Leak Prevention

Memory Leak - Prevention

Inner threading classes should only be static.

```
public class MainActivity extends AppCompatActivity {

...
   public static class ExampleAsyncTask extends AsyncTask<Integer, Integer, String> {

       ...
}
```

Memory Leak - Prevention

Clean threading classes on lifecycle events.

```
public class MainActivity extends AppCompatActivity {

}
```

Memory Leak - Prevention

Clean threading classes on lifecycle events.

```
public class MainActivity extends AppCompatActivity {


    @Override
    protected void onDestroy() {
        super.onDestroy();
        task.cancel(true);

    }





}
```

Memory Leak - Prevention

Clean threading classes on lifecycle events.

```
public class MainActivity extends AppCompatActivity {


    @Override
    protected void onDestroy() {
        super.onDestroy();
        task.cancel(true);

    }

    @Override
    protected void onStop() {
        super.onStop();
        task.cancel(true);

    }
}
```

Async Task Usage Rules

- **Don’t create AsyncTask as a non static inner class.**

Async Task Usage Rules

- Don’t create AsyncTask as a non static inner class.
- **Cancel according to the activity life cycle.**

Async Task Usage Rules

- Don’t create AsyncTask as a non static inner class.
- Cancel according to the activity life cycle.
- **Use WeakReference to update UI**

Recap

- **Thread\Process**

Recap

- Thread\Process
- **Main Thread**

Recap

- Thread\Process
- Main Thread
- **Executor**

Recap

- Thread\Process
- Main Thread
- Executor
- **HandlerThread**

Recap

- Thread\Process
- Main Thread
- Executor
- HandlerThread
- **AsyncTask**

Summary

- Tasks that are not UI related will not run on the UI

#1 Rule in Android development

Never block the UI Thread

Summary

- Tasks that are not UI related will not run on the UI.
- **Inner threading classes should only be static.**

Summary

- Tasks that are not UI related will not run on the UI.
- Inner threading classes should only be static.
- **Clean threading classes on lifecycle events.**

Any Questions?

2020

Threads - Exercise

Homework

Thank you