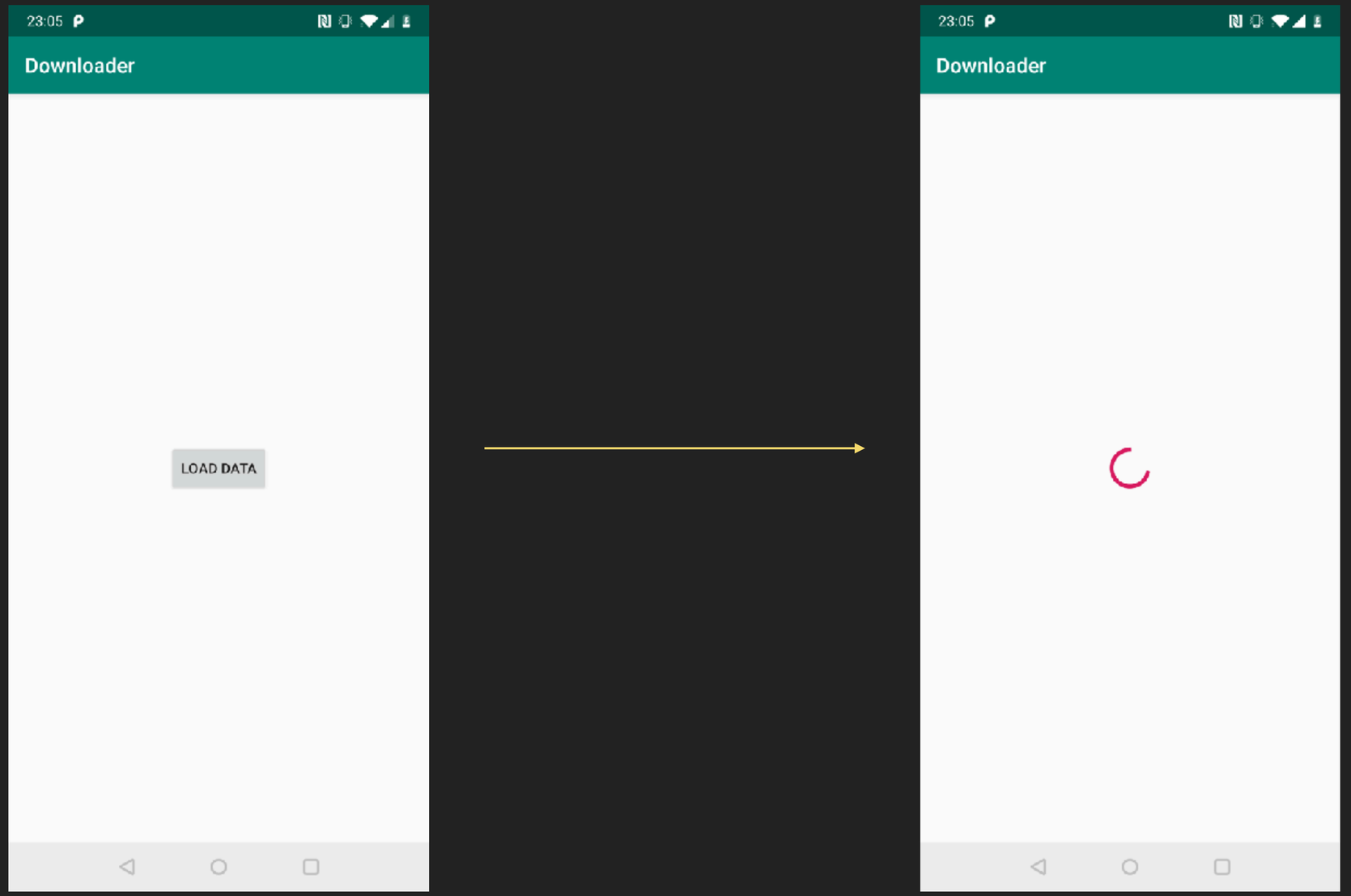
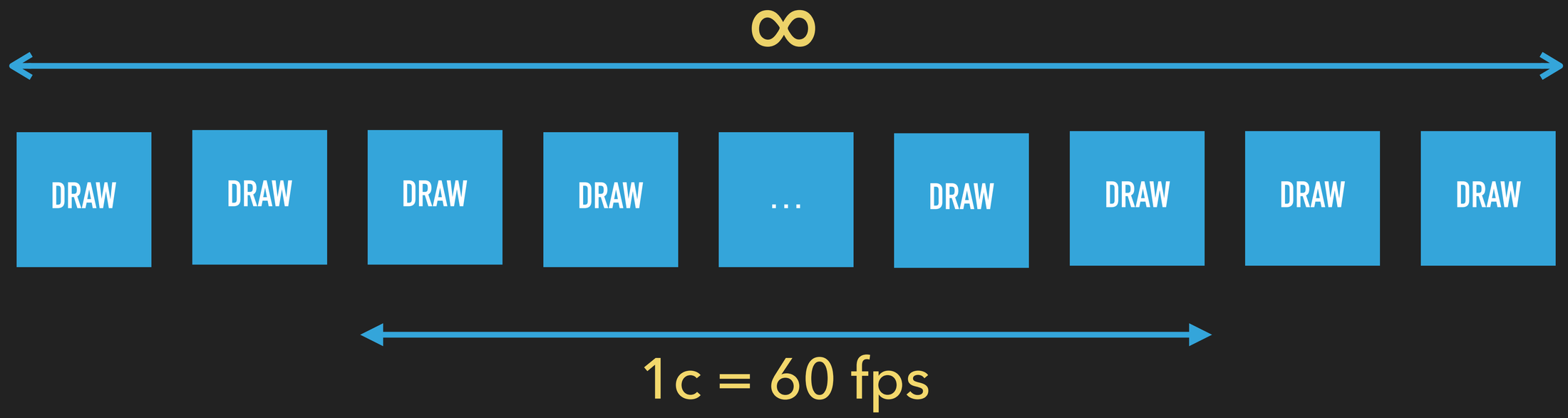
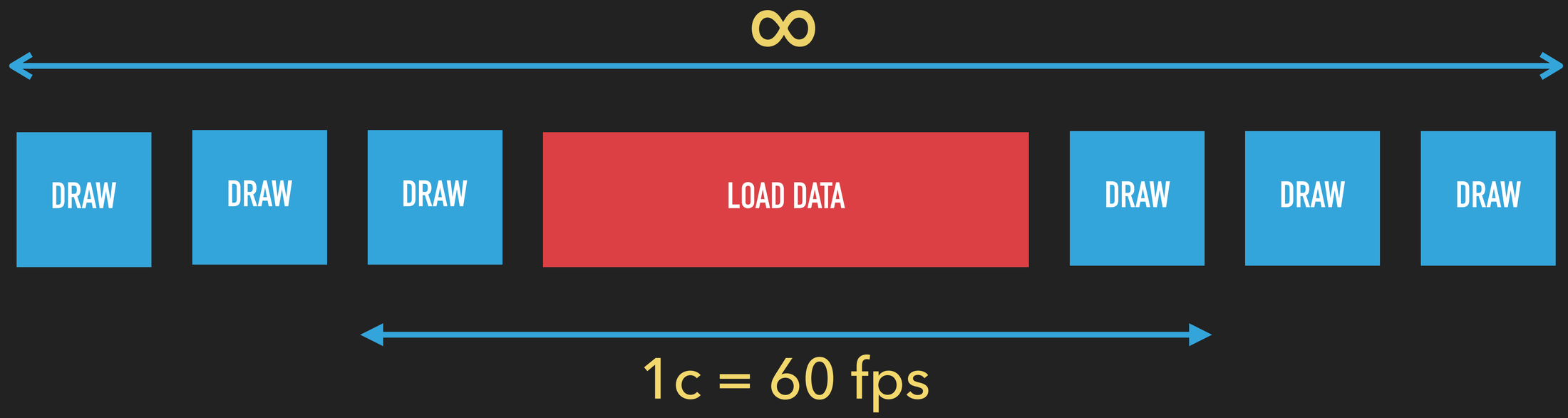
АСИНХРОННОЕ ИСПОЛНЕНИЕ


23:05
Downloader
LOAD DATA
23:05
Downloader

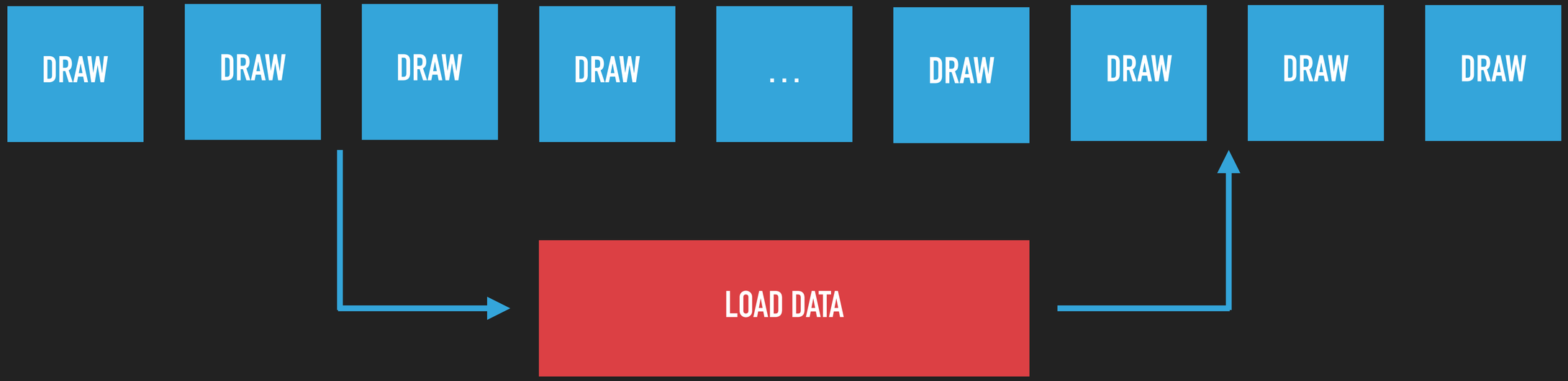
MAIN-ПОТОК



MAIN-ПОТОК



ПРОБЛЕМА



ЧТО ЕСТЬ В JAVA?

- Thread/Runnable
- Механизмы синхронизации потоков

JAVA-WAY

```
public Data load() {
    Data data;
    new Thread(new Runnable() {
        @Override
        public void run() {
            data = download();
        }
    }).start();
    while (data == null) {}
    return data;
}
```

JAVA-WAY

```
public Data load() {
    new Thread(new Runnable() {
        @Override
        public void run() {
            Data data = download();
            runOnUiThread(new Runnable() {
                // TODO
            })
        }
    }).start();
}
```

УТЕЧКИ ПАМЯТИ

В ПАМЯТИ ОСТАЮТСЯ ОБЪЕКТЫ, КОТОРЫЕ БОЛЬШЕ НЕ НУЖНЫ, И GC ИХ НЕ МОЖЕТ СОБРАТЬ

УТЕЧКИ ПАМЯТИ

- Нестатические вложенные классы держат ссылку на внешний класс
- Анонимные классы держат ссылку на внешний класс

JAVA-WAY

```
private static class Downloader {

    private Callback callback;

    public void subscribe(Callback callback) { this.callback = callback; }

    public void unSubscribe() { this.callback = null; }

    public void load() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                Data data = download();
                if (callback != null) {
                    callback.setResult(data);
                }
            }
        }).start();
    }
}

private interface Callback {

    void setResult(String result);
}
```

ССЫЛКИ

- Сильные
- Слабые (пакет java.lang.ref)
  - WeakReference – очистится, если на объект больше нет сильных ссылок
  - SoftReference – очистится, если не будет хватать памяти
  - PhantomReference – перед удалением добавится в ReferenceQueue

JAVA-WAY

```
private static class Downloader {

    private final WeakReference<Callback> callback;

    public Downloader(Callback callback) {
        this.callback = new WeakReference<>(callback);
    }

    public void load() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                Data data = download();
                if (callback.get() != null) {
                    callback.get().setResult(data);
                }
            }
        }).start();
    }
}

private interface Callback {

    void setResult(String result);
}
```

JAVA-WAY

- Код трудноподдерживаемый и немасштабируемый
- Код привязан к runOnUiThread()

ANDROID-WAY

- Looper/Handler
- AsyncTask
- AsyncTaskLoader

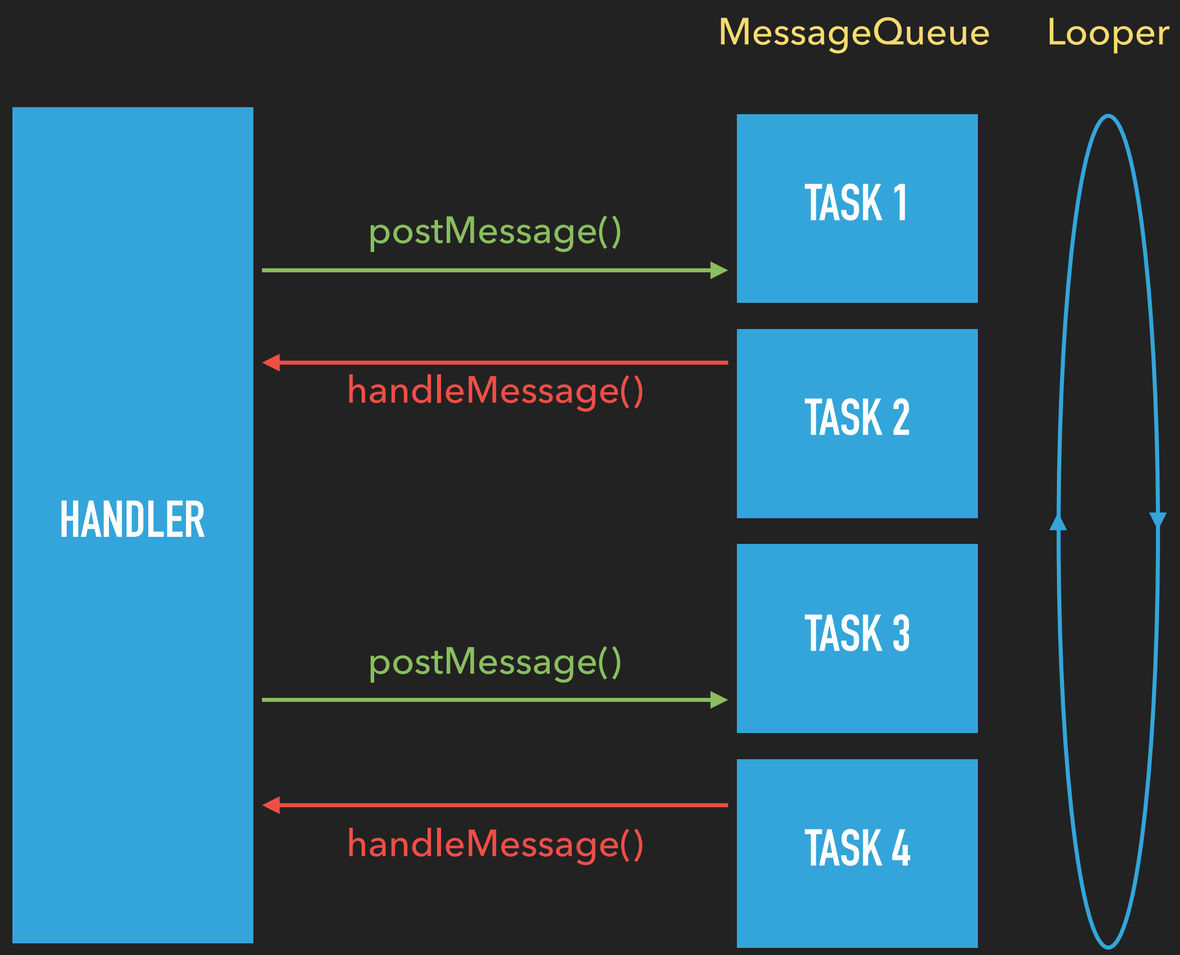
- IntentService

HANDLER

```
Handler handler = new Handler(new Handler.Callback() {
    @Override
    public boolean handleMessage(Message msg) {
        if (msg.what == SEND_RESULT) {
            Data data = (Data) msg.obj;
        }
    }
});
```

```
public void load() {
    new Thread(new Runnable() {
        @Override
        public void run() {
            Data data = download();
            Message message = handler.obtainMessage(SEND_RESULT, data);
            handler.sendMessage(message);
        }
    }).start();
}
```

HANDLER


MessageQueue
Looper
HANDLER
postMessage()
handleMessage()
postMessage()
handleMessage()
TASK 1
TASK 2
TASK 3
TASK 4

ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО ПОТОКА

```
public Handler() {
    ...
    mLooper = Looper.myLooper();
    if (mLooper == null) {
        throw new RuntimeException(
                "Can't create handler inside thread " + Thread.currentThread()
                        + " that has not called Looper.prepare()");
    }
    ...
}
```

```
public static Looper Looper.myLooper() {
    return sThreadLocal.get();
}

private static void Looper.prepare(boolean quitAllowed) {
    sThreadLocal.set(new Looper(quitAllowed));
}
```

LOOPER И MESSAGE QUEUE

- Looper.prepare() – инициализирует Looper в текущем потоке
- Looper.loop() – запускает цикл, который опрашивает MessageQueue; последний метод в цепочке вызовов

ТРЕД С ЛУПЕРОМ

```
class MyThread extends Thread {

    public Handler handler;

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

        handler = new Handler() {
            public void handleMessage(Message msg) {
                // process incoming messages here
            }
        };

        Looper.loop();
    }
}
```

~~ТРЕД С ЛУПЕРОМ~~ HANDLER THREAD

```
class MyThread extends Thread {

    public Handler handler;

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

        handler = new Handler() {
            public void handleMessage(Message msg) {
                // process incoming messages here
            }
        };

        Looper.loop();
    }
}
```

HANDLER

```
Handler handler = new Handler(new Handler.Callback() {
    @Override
    public boolean handleMessage(Message msg) {
        if (msg.what == SEND_RESULT) {
            Data data = (Data) msg.obj;
        }
    }
});
```

```
public void load() {
    new Thread(new Runnable() {
        @Override
        public void run() {
            Data data = download();
            Message message = handler.obtainMessage(SEND_RESULT, data);
            handler.sendMessage(message);
        }
    }).start();
}
```

HANDLER

```
public final boolean post(Runnable r)

public final boolean sendMessage(Message msg)

public final boolean sendEmptyMessage(int what)
```

ACTIVITY . RUN ON UI THREAD ()

```
public final void runOnUiThread(Runnable action) {
    if (Thread.currentThread() != mUiThread) {
        mHandler.post(action);
    } else {
        action.run();
    }
}
```

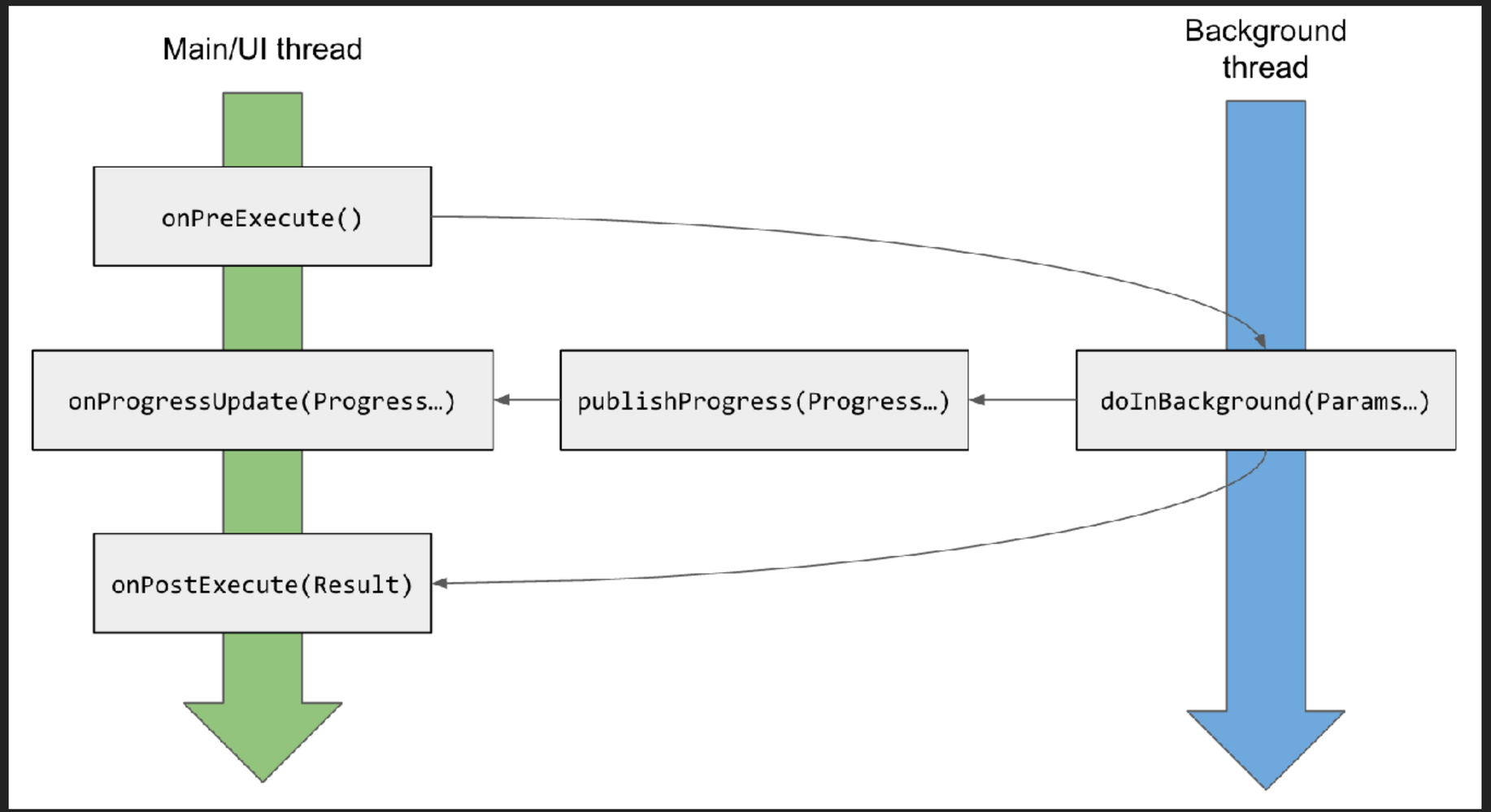
VIEW . POST ()

```
public boolean post(Runnable action) {
    final AttachInfo attachInfo = mAttachInfo;
    if (attachInfo != null) {
        return attachInfo.mHandler.post(action);
    }
    getRunQueue().post(action);
    return true;
}
```

HANDLER. МИНУСЫ:

- Создаётся каждый раз новый поток
- Логика работы с потоками разбросана по коду
- Низкоуровнево: приходится работать не со своими объектами, а с Message

ASYNC TASK


Main/UI thread
Background thread
onPreExecute()
onProgressUpdate(Progress…)
publishProgress(Progress…)
doInBackground(Params…)
onPostExecute(Result)

ASYNC TASK

```
class MyAsyncTask extends AsyncTask<Void, Void, Data> {

    @Override
    protected void onPreExecute() {
        progress.setVisibility(View.VISIBLE);
    }

    @Override
    protected Data doInBackground(Void... params) {
        Data data = download();
        return data;
    }

    @Override
    protected void onPostExecute(Data data) {
        // TODO манипуляции с data
        progress.setVisibility(View.GONE);
    }
}



new MyAsyncTask().execute();
```

ASYNC TASK. ПОТОКИ:

Order of execution

When first introduced, AsyncTasks were executed serially on a single background thread. Starting with `Build.VERSION_CODES.DONUT`, this was changed to a pool of threads allowing multiple tasks to operate in parallel. Starting with `Build.VERSION_CODES.HONEYCOMB`, tasks are executed on a single thread to avoid common application errors caused by parallel execution.

If you truly want parallel execution, you can invoke `executeOnExecutor(java.util.concurrent.Executor, Object[])` with `THREAD_POOL_EXECUTOR`.

https://developer.android.com/reference/android/os/AsyncTask

ASYNC TASK. ОТМЕНА:

Cancelling a task

A task can be cancelled at any time by invoking `cancel(boolean)`. Invoking this method will cause subsequent calls to `isCancelled()` to return true. After invoking this method, `onCancelled(Object)`, instead of `onPostExecute(Object)` will be invoked after `doInBackground(Object[])` returns. To ensure that a task is cancelled as quickly as possible, you should always check the return value of `isCancelled()` periodically from `doInBackground(Object[])`, if possible (inside a loop for instance.)

https://developer.android.com/reference/android/os/AsyncTask

ASYNC TASK. МИНУСЫ:

- Не привязан к жизненному циклу Activity
- Может приводить к утечкам Activity
- Сложности при отмене или смене конфигурации

LOADER

- Переживает изменения конфигурации
- Не приводит к утечкам контекста

ASYNCTASK LOADER

- Надстройка над AsyncTask
- Не отменяется во время поворота экрана
- Хранится в поле в ActivityManager

LOADER

- ~~Переживает изменения конфигурации~~
- ~~Не приводит к утечкам контекста~~
- Deprecated в Android P
- Сильно привязан к activity
- Код немасштабируемый и трудноподдерживаемый

INTENT SERVICE

- Расширение класса Service
- Позволяет запускать долгосрочные операции
- Не очень подходит в большинстве случаев, тяжеловесный

INTENT SERVICE

```
public abstract class IntentService extends Service {

    @Override
    public void onCreate() {
        super.onCreate();
        HandlerThread thread = new HandlerThread("IntentService[" + mName + "]");
        thread.start();

        mServiceLooper = thread.getLooper();
        mServiceHandler = new ServiceHandler(mServiceLooper);
    }

    @Override
    public void onStartCommand(Intent intent, int startId) {
        Message msg = mServiceHandler.obtainMessage();
        msg.arg1 = startId;
        msg.obj = intent;
        mServiceHandler.sendMessage(msg);
    }

    protected abstract void onHandleIntent(@Nullable Intent intent);
}
```

ANDROID-WAY

- Looper/Handler
- AsyncTask
- AsyncTaskLoader
- IntentService

ANDROID-WAY

- ~~Looper/Handler~~
- ~~AsyncTask~~
- ~~AsyncTaskLoader~~
- ~~IntentService~~

MODERN ANDROID WAY

- RxJava
- Kotlin coroutines

RXJAVA

```
download()
        .subscribeOn(io())
        .observeOn(mainThread())
        .subscribe(/* TODO */);
```

RXJAVA

- Плюсы:
  - Мощь и огромные возможности
  - Компактно и читаемо
  - Удобно обрабатывать ЖЦ
- Минусы:
  - Можно выстрелить себе в ногу

RXJAVA. ССЫЛКИ:

- http://reactivex.io/intro.html
- http://reactivex.io/documentation/observable.html
- http://reactivex.io/documentation/scheduler.html
- http://reactivex.io/documentation/subject.html

ИТОГИ:

- Поняли, что в корне всего многопоточного взаимодействия в Android лежит Handler
- Поняли, почему RxJava так популярна в Android

ДОМАШКА:

- Написать класс MyObservable:

```
MyObservable.from(callable)
        .subscribeOn(looper)
        .observeOn(mainLooper)
        .subscribe(callback);
```

- Выполнение стартует только после subscribe()
- subscribeOn – на каком лупере будет выполняться callable; если не указан, выполнится на том треде, в котором вызвали subscribe
- observeOn – на каком лупере будет выполняться callback; если не указан выполнится на треде subscribeOn
- Порядок subscribeOn и observeOn не важен
- Просьба логгировать текущий тред выполнения

СПАСИБО!