Лекция 2

https://github.com/IFMO-Android-2016/lesson2

Многопоточность в Android

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

Multi-threading

Concurrency

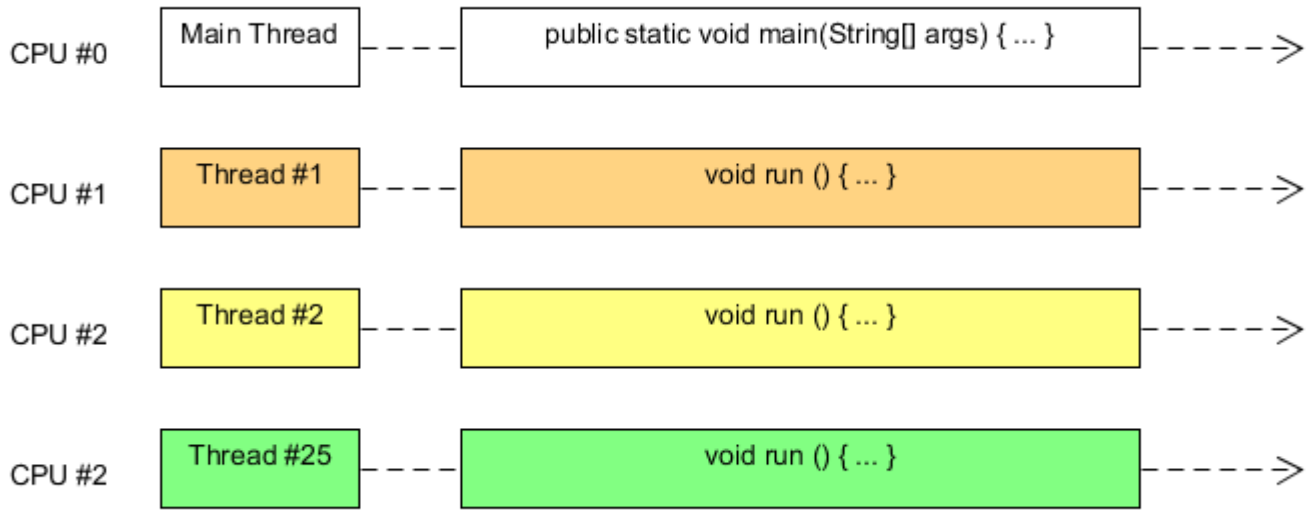
Процессы в Android

- Обычно одному приложению соответствует один процесс (*можно больше в особых случаях)
- Внутри процесса исполняется как минимум один главный (**main**) поток (Thread)
- Одна общая область памяти на один процесс. Все потоки внутри процесса могут обращаться к одной памяти.
- Один процесс не может напрямую обращаться к памяти другого.

Потоки в Java

Потоки

- Потоки – параллельно (одновременно) выполняемый код



- Могут выполняться на разных или на одном CPU
- Количество не ограничено (в пределах доступной памяти)

Потоки в Java

- Каждому потоку соответствует объект `Thread`
- Текущий поток: `Thread.currentThread()`
- Запуск нового потока: `new Thread.start()`
- Интерфейс `Runnable` для определения кода, который будет выполняться потоком:

```
public interface Runnable {
    /**
     *  Этот метод будет выполнен в отдельном потоке
     */
    public void run();
}
```

Запуск потока

```
class KillAllHumanTask implements Runnable {
    @Override
    public void run() {
        while (haveAliveHumans()) {
            killOneHuman();
        }
    }
}

KillAllHumanTask task = new KillAllHumanTask();

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

Запуск потока

- Вариант с анонимным Runnable

```
new Thread(new Runnable() {
    @Override
    public void run() {
        doSomething();
    }
}).start();
```

- Java 8 (Android 7.0)

```
new Thread(() -> { doSomething(); }).start();
new Thread(SomeClass::doSomething).start();
```

Переиспользование потоков

- Создание нового потока – дорогая операция
  (На Android выделяется 1Мб памяти для стэка)

- Вместо создания нового потока для каждой новой задачи надо использовать `Executor`

~~**new Thread**(task).start();~~

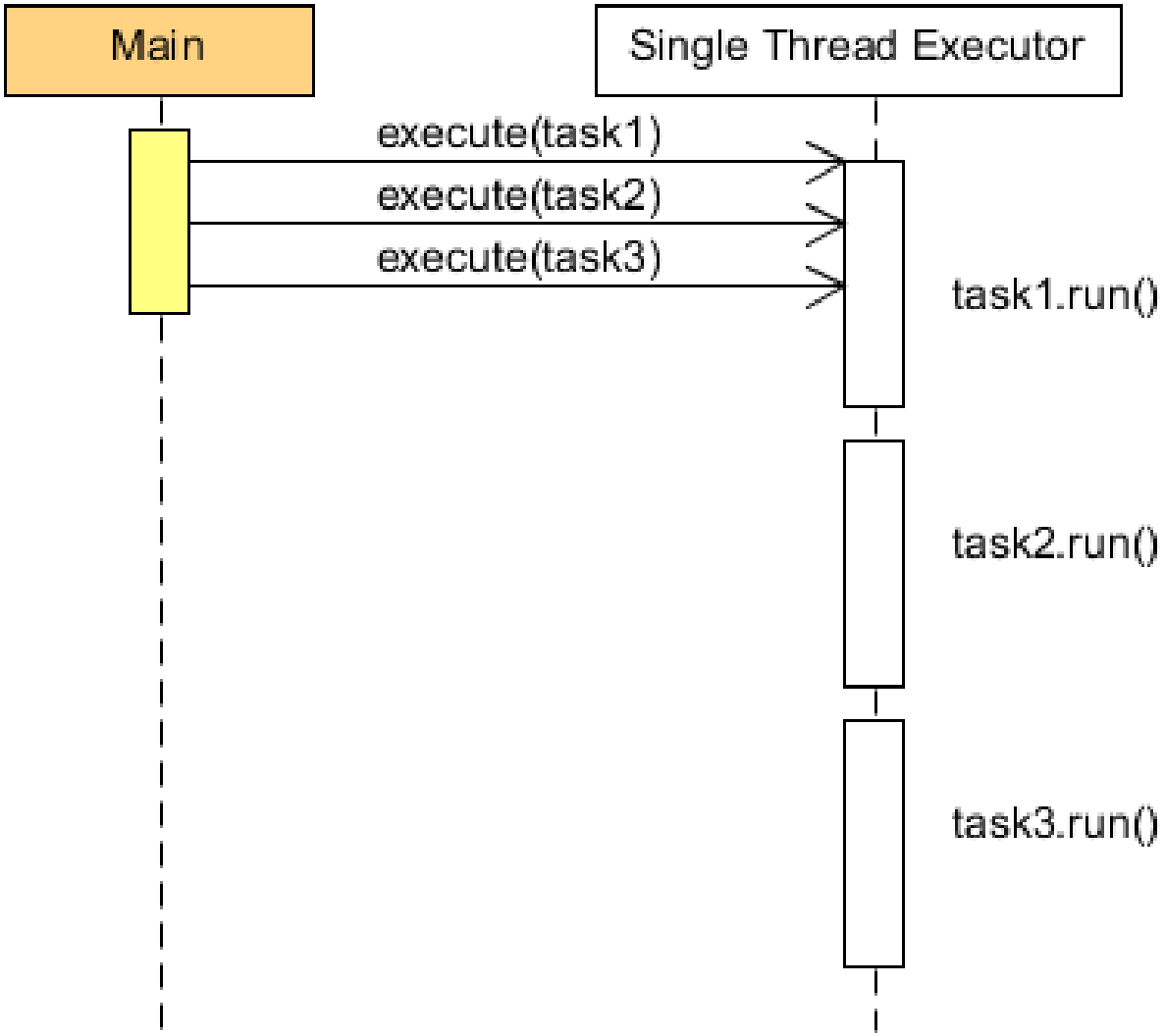
```
// Получаем и сохраняем где-нибудь объект executor-а
Executor executor = ...

// Используем его для запуска всех задач
executor.execute(new RunnableTask1());
executor.execute(new RunnableTask2()); ...
```

Single Thread Executor

```
Executor executor = Executors.newSingleThreadExecutor();
```

- Выполняет задачи по очереди в одном потоке, одна за другой.
- Одновременно выполняется только одна задача.
- Задачи выполняются в порядке поступления

Single Thread Executor

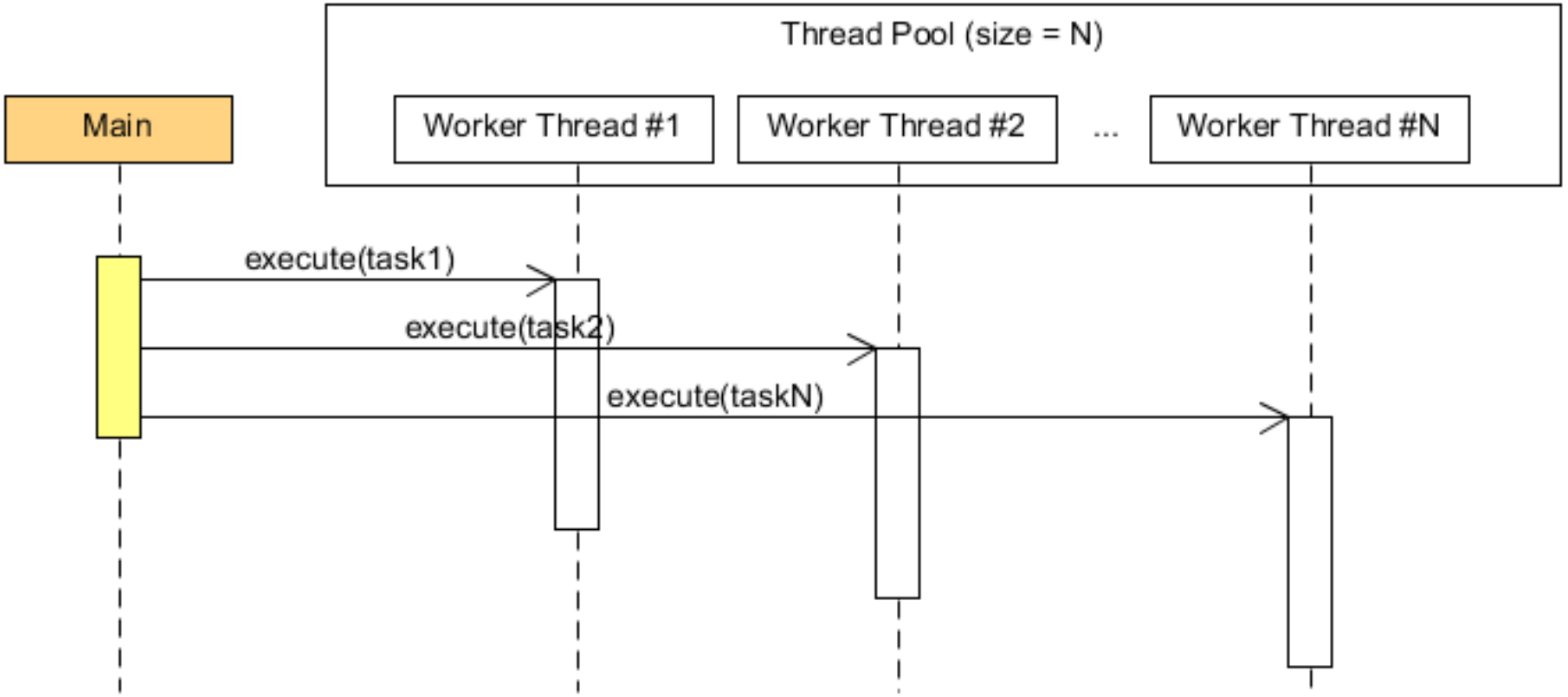


Thread Pool

```
Executor executor = Executors.newFixedThreadPool(N);
```

- Использует не более N потоков
- Выполняет до N задач одновременно
- Порядок выполнения не гарантирован

Thread Pool


Thread Pool (size = N)
Main
Worker Thread #1
Worker Thread #2
...
Worker Thread #N
execute(task1)
execute(task2)
execute(taskN)

Android / ANDROID-5161

[Crash] Кончается память на потоки на кривых самсунгах

Edit Comment Assign More Reopen

Details

Type:	Bug	Status:	Resolved
Priority:	Major	Resolution:	Done
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		

Description

Только Samsung
Только Android 5.0

Самсунг решил увеличить размер стэка с 16кб до 1мб... Начинаем экономить потоки ☹
Возможно, стоит уменьшить размер пула потоков в шине, который сейчас у нас бесконечный.

java.lang.OutOfMemoryError: pthread_create (1040KB stack) failed: Try again
at java.lang.Thread.nativeCreate(Native Method)

		Версия Android			Устройство		
33	63,5 %	Android 5.0	52	100,0 %	Galaxy S4 (jflte)	1	1,9 %
12	23,1 %				Galaxy S4 (ja3g)	42	80,8 %
7	13,5 %				Galaxy S5 (k3g)	1	1,9 %
					Galaxy Note3 (ha3g)	1	1,9 %
					Galaxy S5 (klte)	5	9,6 %
					Galaxy S5 (kltevzw)	2	3,8 %

я пользователей

Страница **7** из **28** ◀ ▶ Перейти на страницу Перейти

```
java.lang.OutOfMemoryError: pthread_create (1040KB stack) failed: Try again
    at java.lang.Thread.nativeCreate(Native Method)
    at java.lang.Thread.start(Thread.java:1063)
    at java.util.concurrent.ThreadPoolExecutor.addWorker(ThreadPoolExecutor.java:920)
    at java.util.concurrent.ThreadPoolExecutor.execute(ThreadPoolExecutor.java:1338)
```

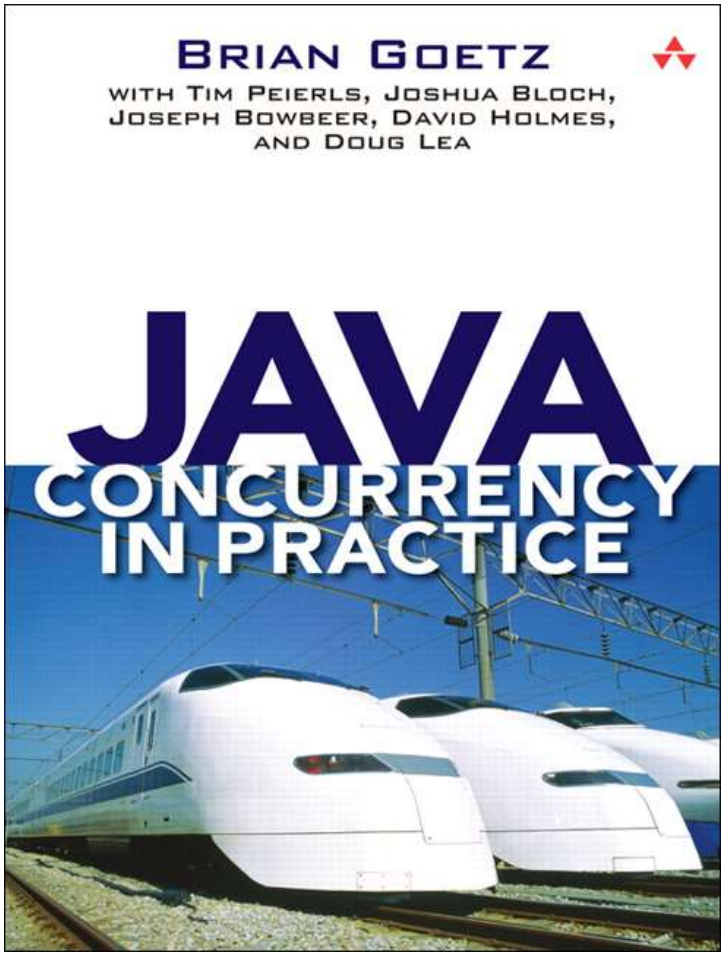
Надо избегать создания новых потоков.

Потоки можно переиспользовать:

`Thread Pool, Executor Service`

Что нужно знать еще?

- Конкурентный доступ к памяти
- Атомарные операции
- Синхронизация
- Dead Lock
- Потокобезопасность
- java.util.concurrent.*
- …



- Небольшой туториал по основам Java Concurrency: https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html

- Классическая и очень полезная книга

  Java Concurrency In Practice

  Brian Goetz и др.

Потоки в Android

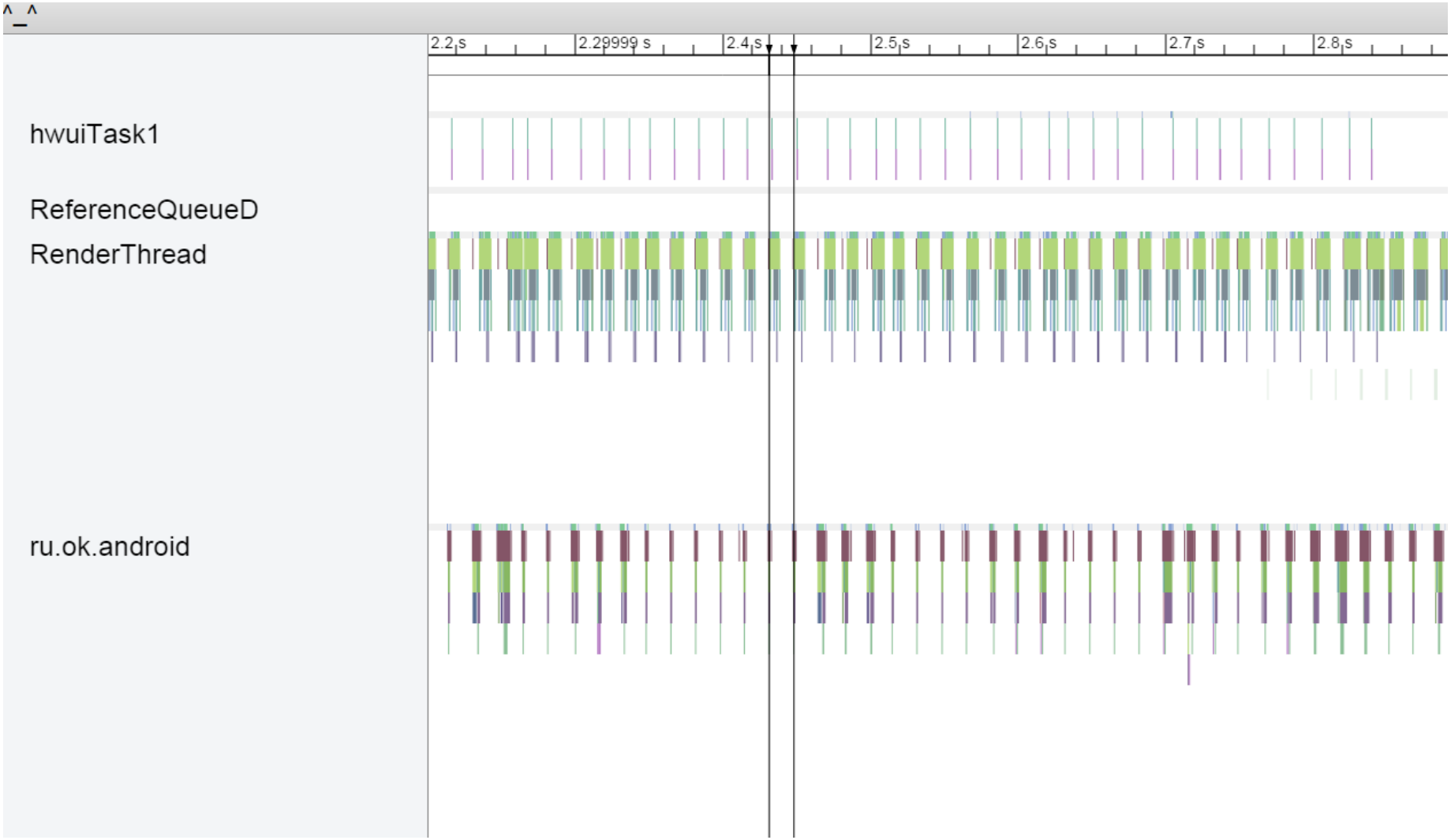
Основной поток

Основной (UI, main) поток

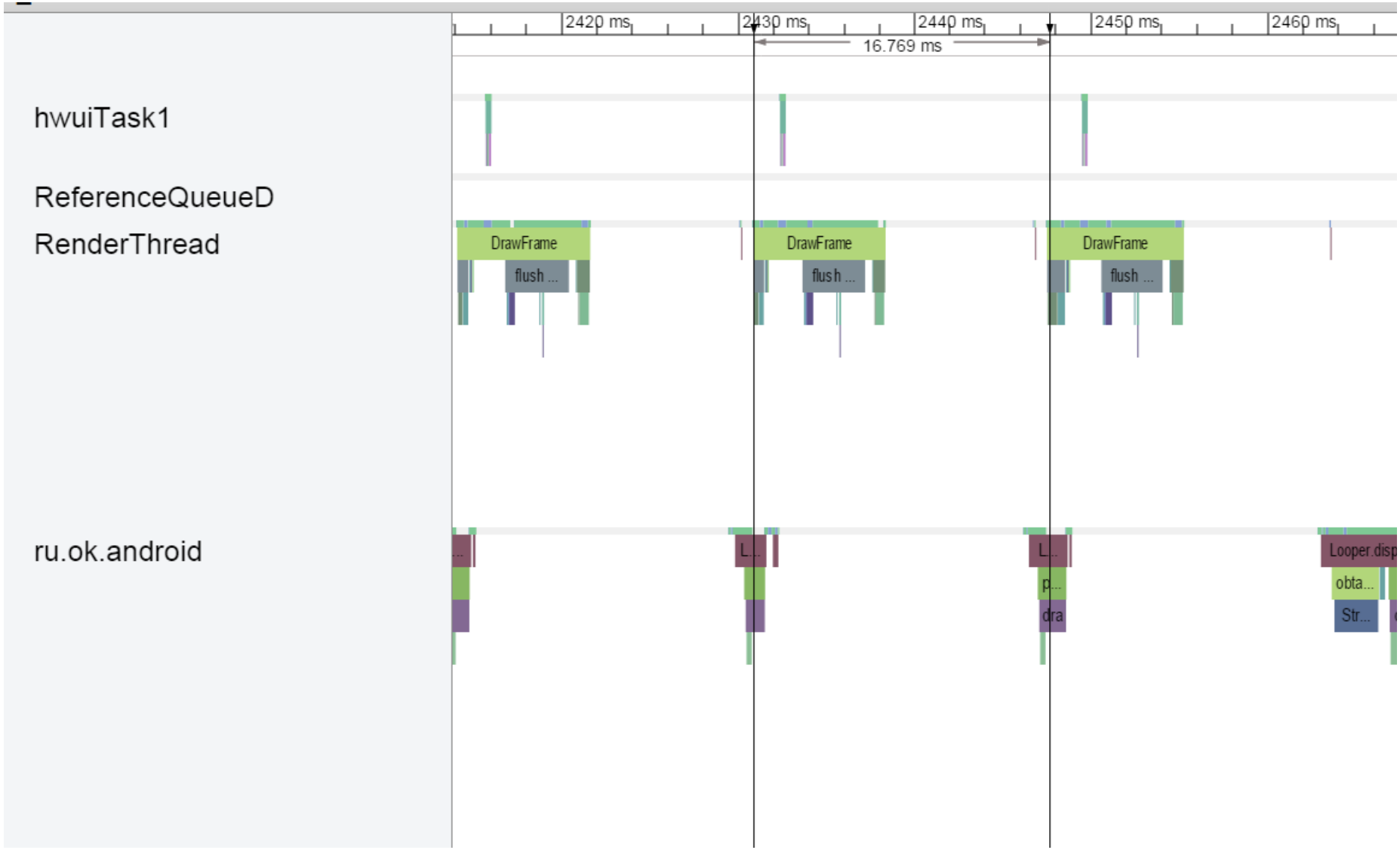
- Жизненный цикл : `Activity.onCreate`
- Верстка: `View.onMeasure, View.onLayout`
- Отрисовка: `View.onDraw`
- События интерфейса: `onTouch(), onClick()`
- Почти все методы подклассов `View`: `TextView.setText`, `ImageView.setImageBitmap`, `View.setVisibility`

Основной (UI, main) поток

- Все методы в основном потоке должны работать быстро: 60 кадров в секунду, 16 мс на кадр
- Если какой-то из UI методов будет выполняться дольше 16 мс, будут видны лаги, приложение будет «тормозить»



Systrace показывает время выполнения методов в разных потоках



Для плавного UI все методы должны отрабатывать за 16 мс или быстрее

Что нельзя делать в основном потоке?

- Операции с файлами
- Выполнение сетевых запросов
- Операции с БД (это тоже файлы)
- Любые долгие операции (например, декодирование изображений)
- Ожидание чего-либо (`sleep, wait`)
- Чтение `Shared Preferences` (тоже файлы).

Strict Mode

Особый режим, в котором система оповещает обо всех случаях выполнения долгих задач в UI потоке.

Включается в коде:

```
StrictMode.setThreadPolicy(
        new StrictMode.ThreadPolicy.Builder()
                .detectAll().penaltyLog().build());
```

Strict Mode

Пример: предупреждение в логе о чтении файла в UI потоке в методе `Application.onCreate`

```
D/StrictMode : StrictMode policy violation; ~duration=110 ms:
android.os.StrictMode$StrictModeDiskReadViolation
    at android.os.StrictMode$AndroidBlockGuardPolicy.onReadFromDisk(
            StrictMode.java:1135)
    at android.app.SharedPreferencesImpl.awaitLoadedLocked(
            SharedPreferencesImpl.java:203)
    at android.app.SharedPreferencesImpl.getString(
            SharedPreferencesImpl.java:223)
    at ru.ok.android.OkApplication.onCreate(
            OkApplication.java:186)
```

Фоновые потоки

Background threads

Фоновые потоки

Способы выполнения задач в фоновых потоках на Android:

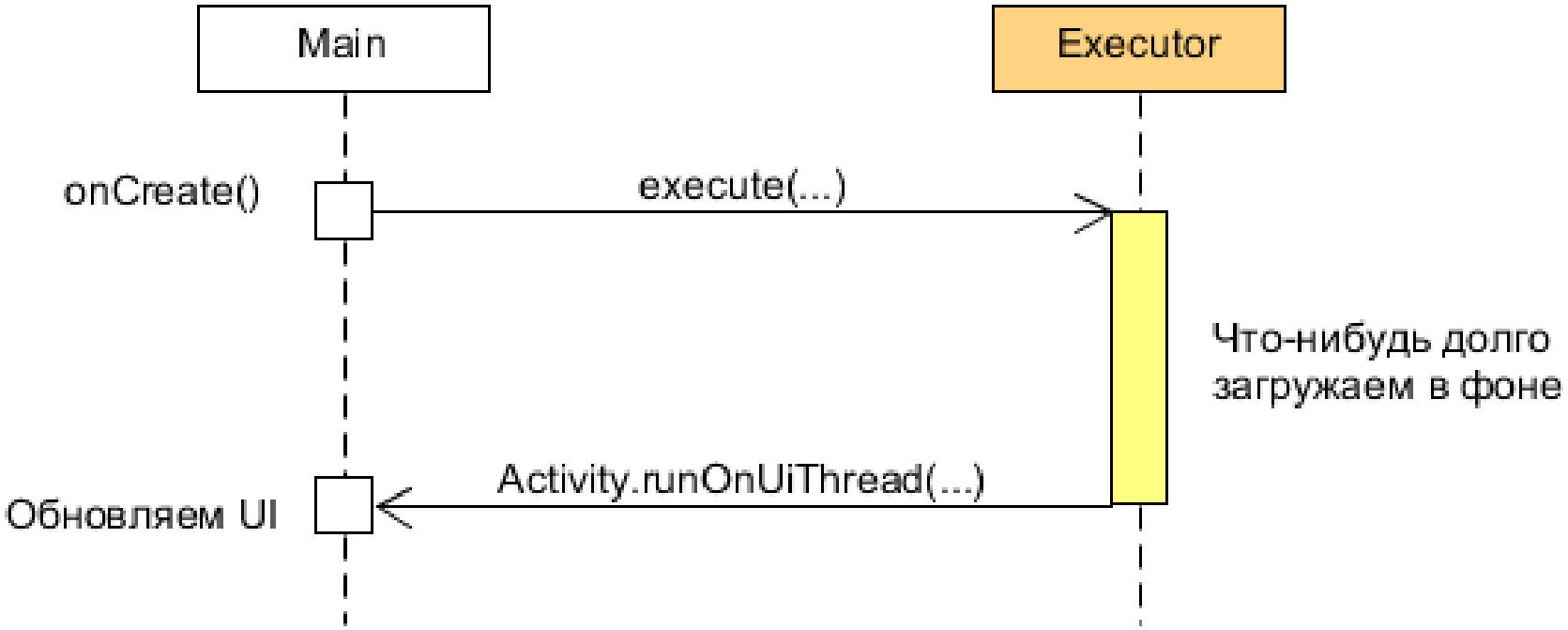
- `Executor` – как в обычной Java
- `AsyncTask` – работает на Executor, удобные методы передачи результата в main поток
- `Loader` – решает проблему жизненного цикла, основное средство загрузки данных в фоне
- `Handler, Looper, HandlerThread, Message` – обработка очереди сообщений, низкоуровневый контроль. Используются под капотом у AsyncTask, Loader и всего Android-а.

Executor

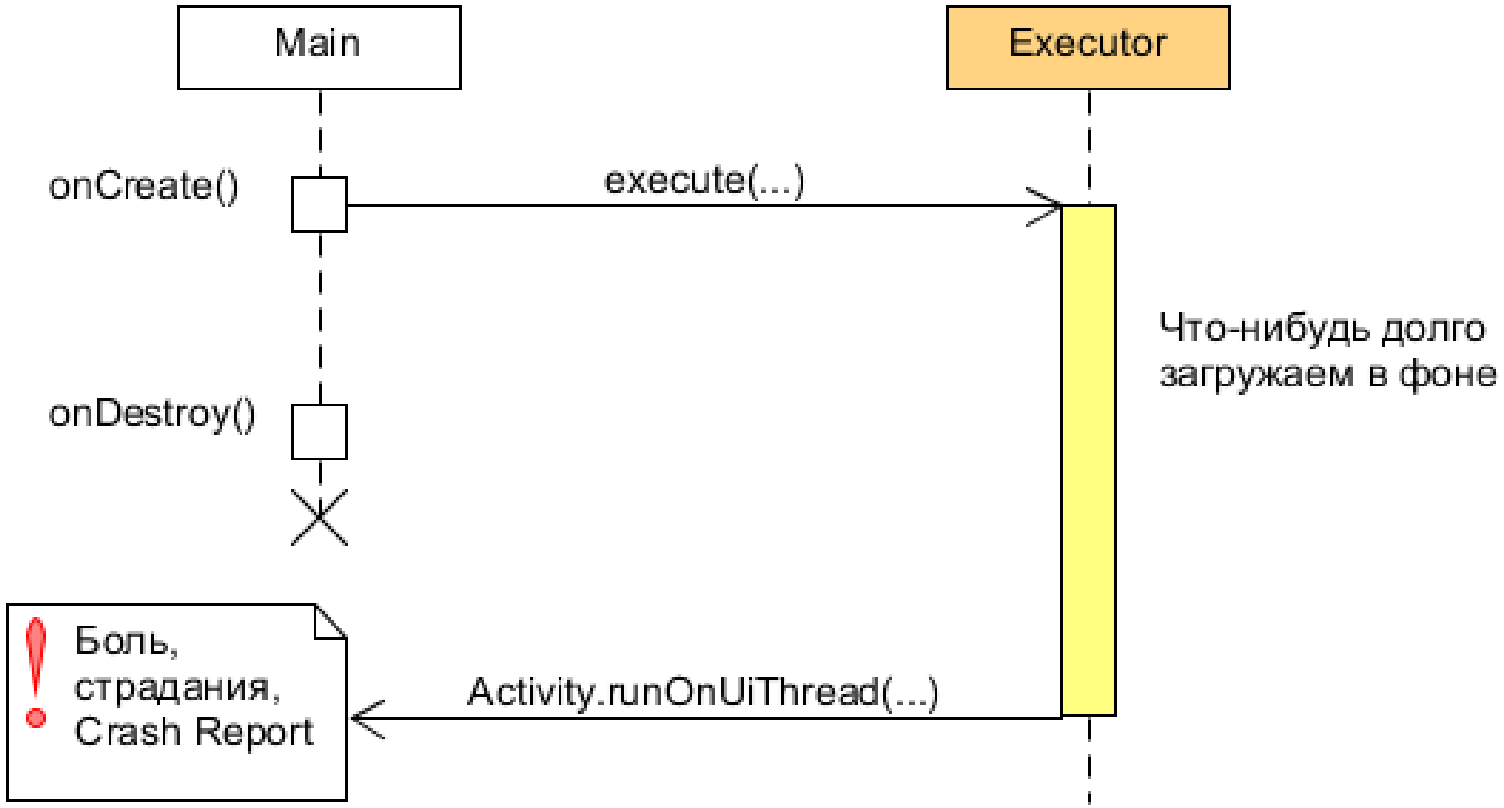
Можно просто кинуть задачу на выполнение в Executor и забыть… Но что если хотим получить результат обратно в main потоке?

```
final Activity activity = this;
final ImageView imageView = findViewById(R.id.image_view);

executor.execute(() -> {
    Bitmap bitmap = ... // Загружаем картинку из сети
    activity.runOnUiThread(() -> {
        imageView.setImageBitmap(bitmap);
    });
});
```

Executor


Main
Executor
onCreate()
execute(...)
Что-нибудь долго
загружаем в фоне
Activity.runOnUiThread(...)
Обновляем UI

Проблема жизненного цикла



Проблема жизненного цикла

В callback методах, которые «приходят» из фоновых потоков, всегда проверять, жив ли еще UI?

```
executor.execute(() -> {
    Bitmap bitmap = ... // Загружаем картинку из сети
    activity.runOnUiThread(() -> {

        if (...) { // Проверяем, что UI ещё жив

            imageView.setImageBitmap(bitmap);
        }
    });
});
```

Проблема жизненного цикла

Проверяем, что UI ещё «жив»:

- `!Activity.isFinishing()`
- `View.isAttachedToWindow()`
- `Fragment.getActivity() != null`

Потоки в Android

AsyncTask – выполнение задачи в фоновом потоке и передача результата в UI поток

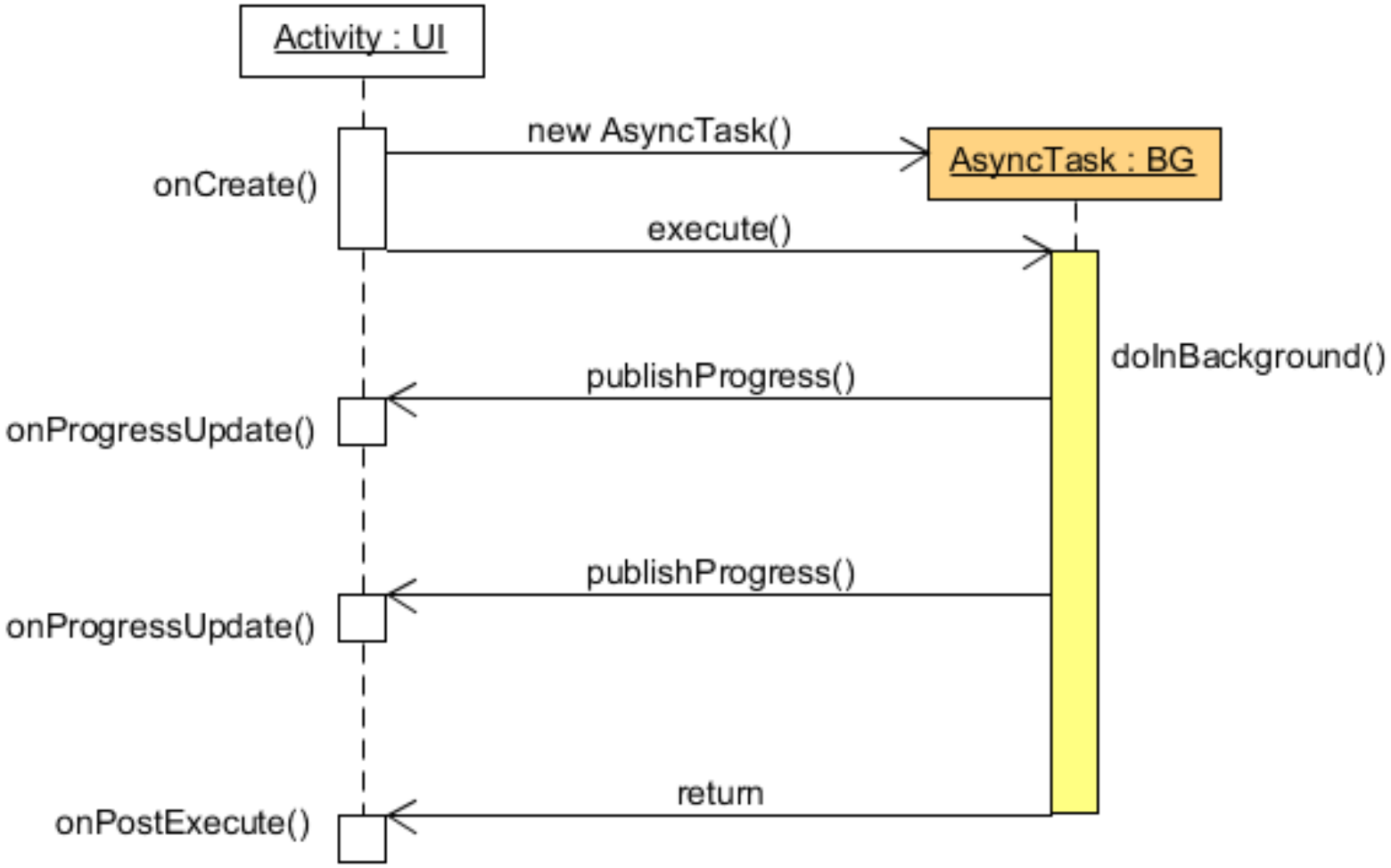
Сценарий: скачивание файла

- При старте приложение начинает скачивание изображения из сети
- Пока идет скачивание, показывает индикатор прогресса
- После завершения скачивания приложение показывает изображение на экране
- Скачивание выполняется в фоновом потоке при помощи `android.os.AsyncTask`

AsyncTask

- `doInBackground(Param... params)` выполняется в фоновом потоке
- `execute(Params... Params)` запускает задачу из UI потока
- `onPostExecute(Result result)` выполняется в UI потоке после завершения
- `publishProgress(Progress progress)` вызывается из кода `doInBackground`
- `onProgressUpdate(Progress... values)` выполняется в UI потоке

AsyncTask


Activity : UI
AsyncTask : BG
onCreate()
new AsyncTask()
execute()
doInBackground()
publishProgress()
onProgressUpdate()
publishProgress()
onProgressUpdate()
return
onPostExecute()

AsyncTask

```
class GetImageTask extends AsyncTask<Void, Void, Bitmap> {
    @Override
    protected Bitmap doInBackground(Void... ignore) {
        // Этот метод выполняется в фоновом потоке
        try {
            return downloadImage(downloadUrl);
        } catch (Exception e) {
            Log.e(TAG, "Error downloading file: " + e, e);
            return null;
        }
    }

    @Override
    protected void onPostExecute(Bitmap bitmap) {
        // Этот метод выполняется в UI потоке
        // Параметр bitmap -- это результат doInBackground
        progressBarView.setVisibility(View.INVISIBLE);
        imageView.setImageBitmap(bitmap);
    }
}
```

Запуск AsyncTask

```
public class LoadImageActivity extends Activity {

    private ProgressBar progressBarView;
    private ImageView imageView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_load_image);

        progressBarView.setVisibility(View.VISIBLE);

        new GetImageTask().execute();
    }
}
```

AsyncTask: отображение прогресса

```
/**
 * Callback интерфейс для получения уведомления о прогрессе.
 */
public interface ProgressCallback {

    /**
     * Вызывается при изменении значения прогресса.
     * @param progress новое значение прогресса от 0 до 100.
     */
    void onProgressChanged(int progress);

}
```

AsyncTask: отображение прогресса

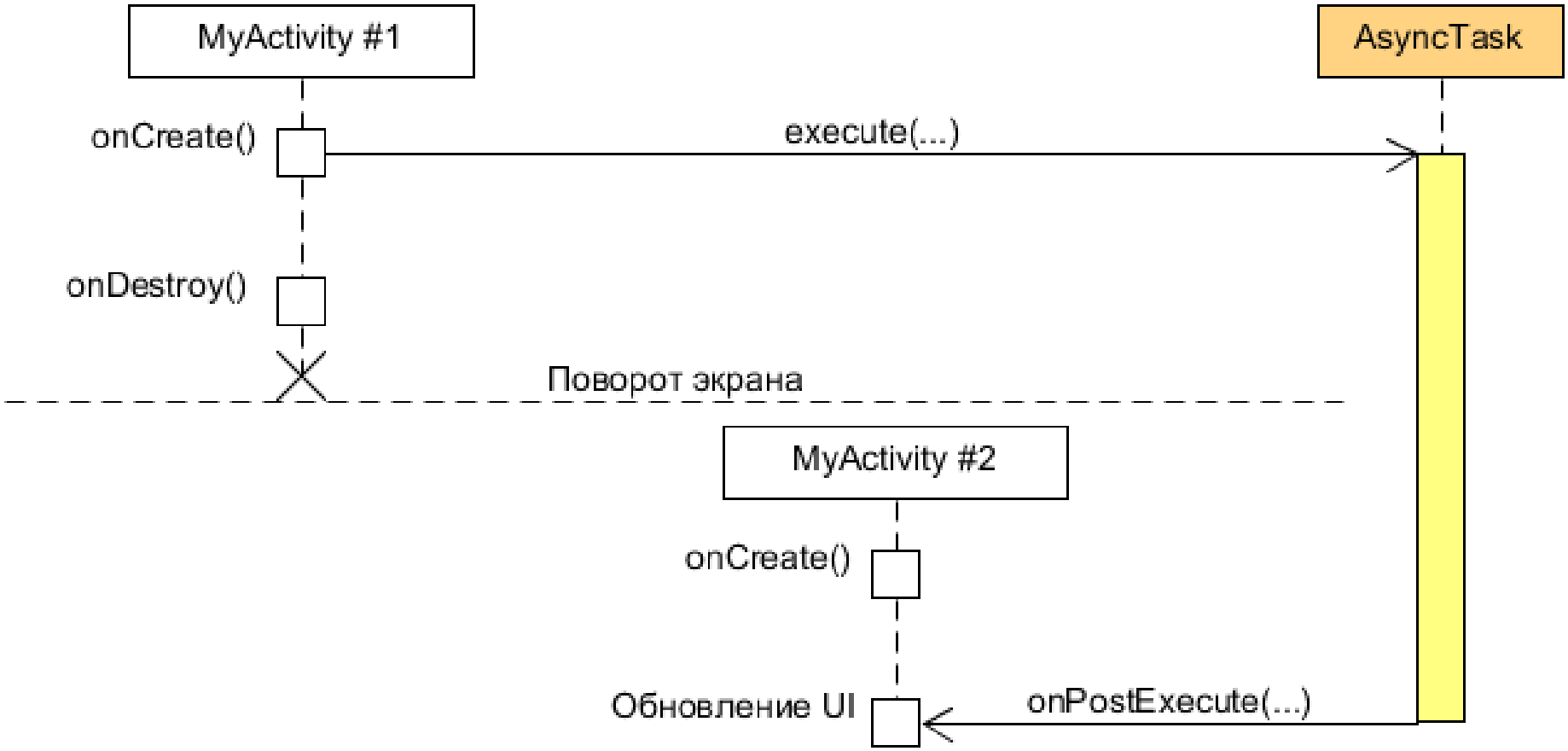
```
class GetImageTask extends AsyncTask<Void, Integer, Bitmap>
        implements ProgressCallback {

    protected Bitmap doInBackground(Void... ignore) {
        return downloadImage(url, this /*progressCallback*/);
    }

    // Метод ProgressCallback, вызывается в фоновом потоке
    public void onProgressChanged(int progress) {
        publishProgress(progress);
    }

    // Метод AsyncTask, вызывается в UI потоке
    protected void onProgressUpdate(Integer... values) {
        int progress = values[values.length - 1];
        progressBarView.setProgress(progress);
    }
}
```

Смена конфигурации (крутим экран)

Configuration Change

Runtime Change

Смена конфигурации



Смена конфигурации

- При смене конфигурации создается новый объект `Activity`
- Запущенный `AsyncTask` продолжает работать!
- `AsyncTask` должен получить ссылку на новый объект `Activity` для отображения прогресса
- Новый объект `Activity` не должен запускать новый `AsyncTask`, а должен «связаться» со старым.

Смена конфигурации

```
static class GetImageTask extends AsyncTask {

    // Текущий объект Activity, храним для обновления отображения
    private LoadImageActivity activity;

    GetImageActivity(LoadImageActivity activity) {
        this.activity = activity;
    }

    void attachActivity(LoadImageActivity activity) {
        this.activity = activity;
        updateView();
    }

    void updateView() {
        if (activity != null && !activity.isFinishing()) {
            activity.imageView.setImageBitmap(...);
            activity.progressBarView.setProgress(...);
        }
    }
```

Смена конфигурации

```
public class GetImageActivity extends Activity {

    // Выполняющийся таск загрузки изображения
    private GetImageTask getImageTask;

    @Override
    public Object onRetainNonConfigurationInstance() {
        // Этот метод вызывается при смене конфигурации,
        // когда текущий объект Activity уничтожается. Объект,
        // который мы вернем, не будет уничтожен, и его можно
        // будет использовать в новом объекте Activity
        return getImageTask;
    }
```

Смена конфигурации

```
public class LoadImageActivity extends Activity {

    protected void onCreate(Bundle savedInstanceState) {
        ...

        if (savedInstanceState != null) {
            // Пытаемся получить ранее запущенный таск
            getImageTask = (GetImageTask)
                    getLastNonConfigurationInstance();
        } if (getImageTask == null) {
            // Создаем новый таск, только если не было
            // ранее запущенного таска
            getImageTask = new GetImageTask(this);
            getImageTask.execute();
        } else {
            // Передаем в ранее запущенный таск текущий
            // объект Activity
            getImageTask.attachActivity(this);
        }
    }
```

Слишком сложно?

Используем Loader вместо AsyncTask

Loader

- Гибкий фреймворк для асинхронной загрузки чего-нибудь
- Решает проблему жизненного цикла
- Проще, чем `AsyncTask`
- Базовый класс `AsyncTaskLoader` выполняет задачу на пуле потоков (в отличие от `AsyncTask`), то есть может выполняться несколько задач параллельно

https://developer.android.com/guide/components/loaders.html