RenderScript

Plan

- Introduction
- RenderScript Architecture
- Hello RenderScript!
- More RenderScript
- RenderScript and Multithreading

What is RenderScript?

- Framework for running **computationally intensive** tasks at **high performance** on Android.
- Oriented for use with **data-parallel computation**.
- RenderScript runtime parallelizes work across all available processors
  - CPUs,
  - GPUs,
  - or DSPs
- Focus on **expressing algorithms** rather than scheduling work or load balancing.
- Applications performing
  - image processing,
  - computational photography
  - computer vision

RenderScript design goals

- **Portability**: to be able to run across all devices with radically different hardware.
  - ARM and its several variants (NEON, VPS)
  - X86 CPUs
  - GPU and DSP
- **Performance**: get as much performance as possible within the constraints of Portability.
- **Usability**: simplify development as much as possible and automate the developing process.

RenderScript Vs NDK

- The performance gain comes from executing **native code** on the device.
- Unlike NDK, this is cross-platform.
- Compiled to a device-agnostic intermediate format during the development process and placed into the application package.
- Scripts are compiled to machine code and optimized on the device **when the app is run**.
- No need to target a specific machine architecture

CPU or GPU? ... Or?

- Decision on where code has to run made on the device **at runtime**.
- On desktop system best performances when GPU and CPU are stressed
- **On mobile application responsiveness matters**!
  - Better stress the GPU than CPU
  - CPU is needed for keeping the system responsive
  - Android UI is Hw accelerated, so better not to fill 100% GPU
- The **CPU also serves as a fallback** to ensure that scripts are always able to run even if a suitable GPU or other accelerator is not present.
- Meant to be transparent to the developer.

RenderScript Kernel

- Based on **Kernel**
- Basic function to apply to a single element of a collection
- Based on C99 language with extensions
- File *.rs

```
#pragma version(1)
#pragma rs java_package_name(fr.enseeiht.hellorscompute)
#pragma rs_fp_relaxed

// this kernel inverts the colors of the input image
uchar4 RS_KERNEL invert(uchar4 in)
{
   uchar4 out = in;
   out.r = 255 - in.r;
   out.g = 255 - in.g;
   out.b = 255 - in.b;
   return out;
}
```

Plan

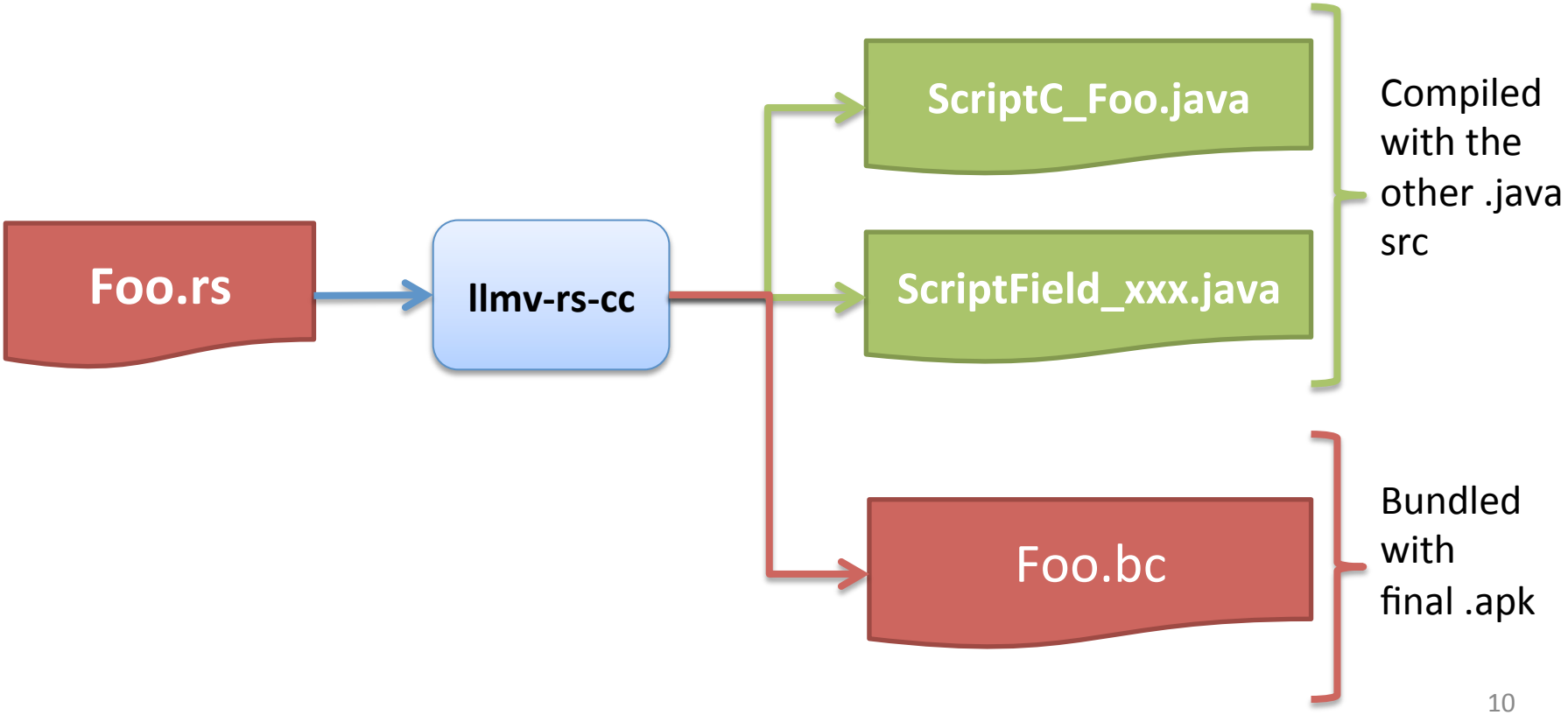
- Introduction
- RenderScript Architecture
- Hello RenderScript!
- More RenderScript
- RenderScript and Multithreading

RenderScript Architecture

- **Offline compiler** (llvm-rs-cc)
  - Convert script files into portable bitcode and reflected Java layer
- **Online JIT compiler** (libbcc)
  - Translate portable bitcode to appropriate machine code (CPU/GPU/DSP/...)
- **Runtime library support** (libRS)
  - Manage scripts from Dalvik layer
  - Also provide basic support libraries (math functions, etc.)

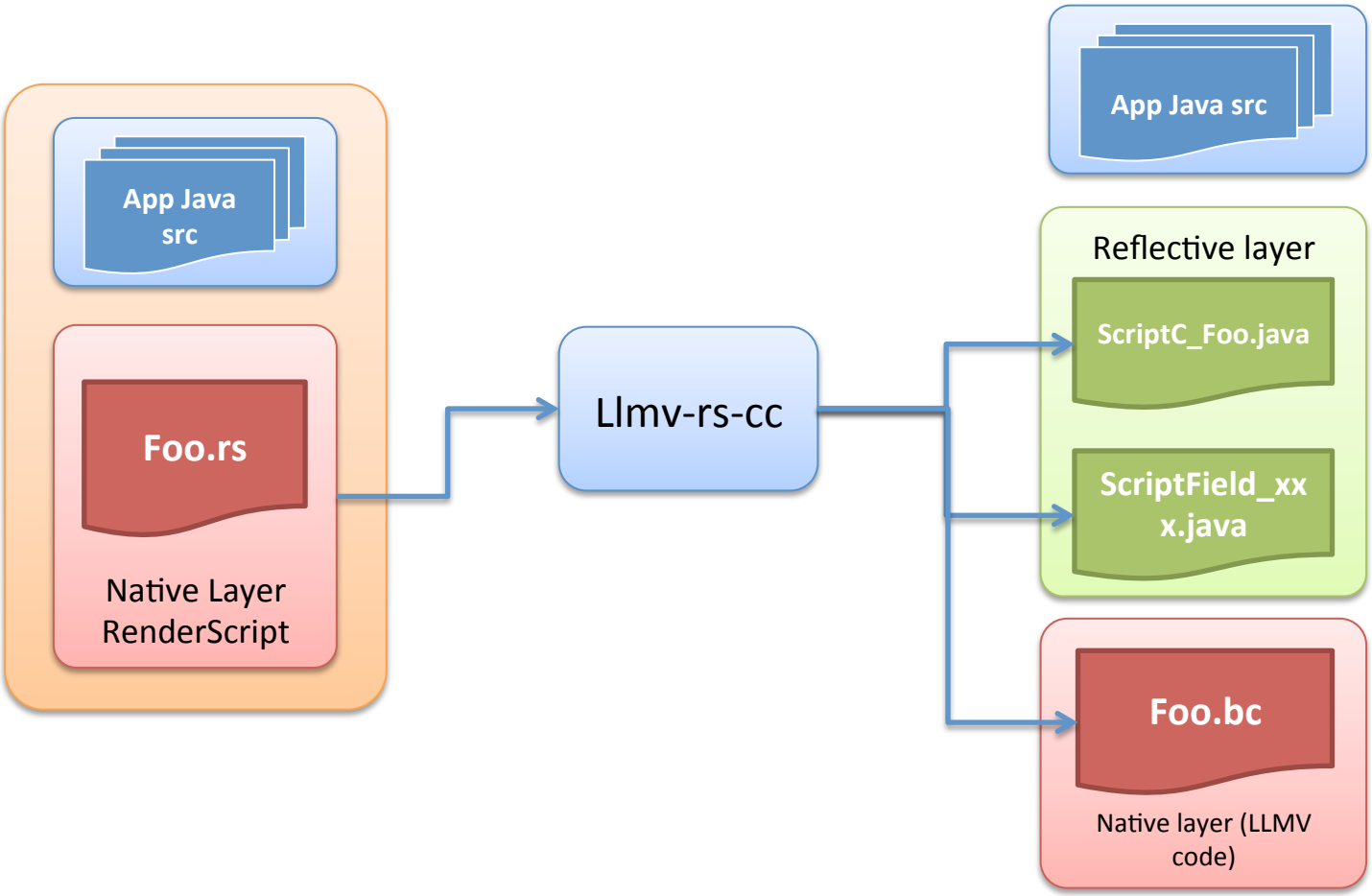
Offline Compiler – llvm-rs-cc

- Convert script files into **portable bitcode** and reflected Java layer
- Based on Clang/LLVM popular open source compiler (**slang**)
- All bitcode supplied as a resource within .apk container



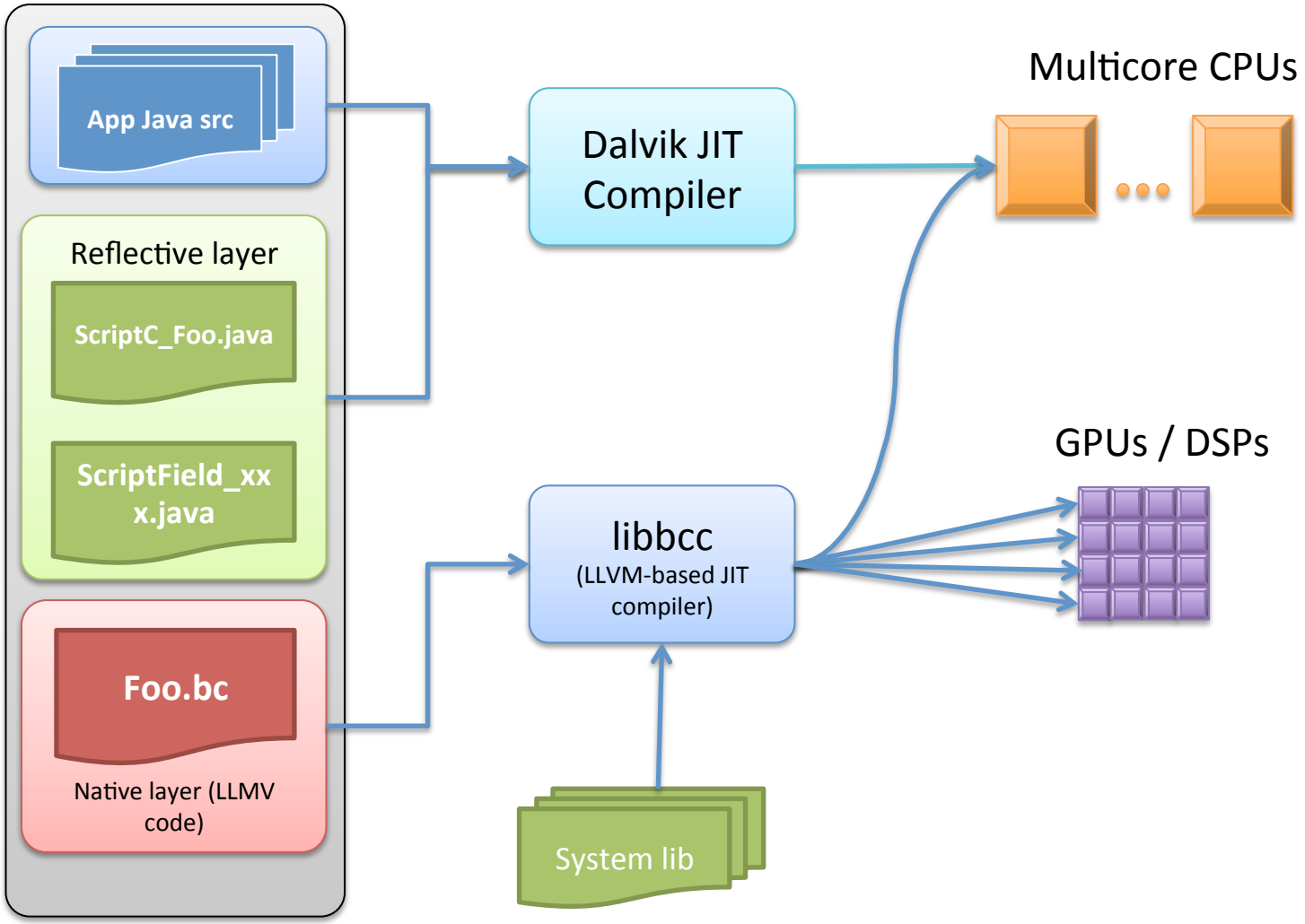
Offline Compiler – llvm-rs-cc

- Convert script files into **portable bitcode** and reflected Java layer
- Based on Clang/LLVM popular open source compiler (**slang**)
- All bitcode supplied as a resource within .apk container



Online JIT compiler – libbcc

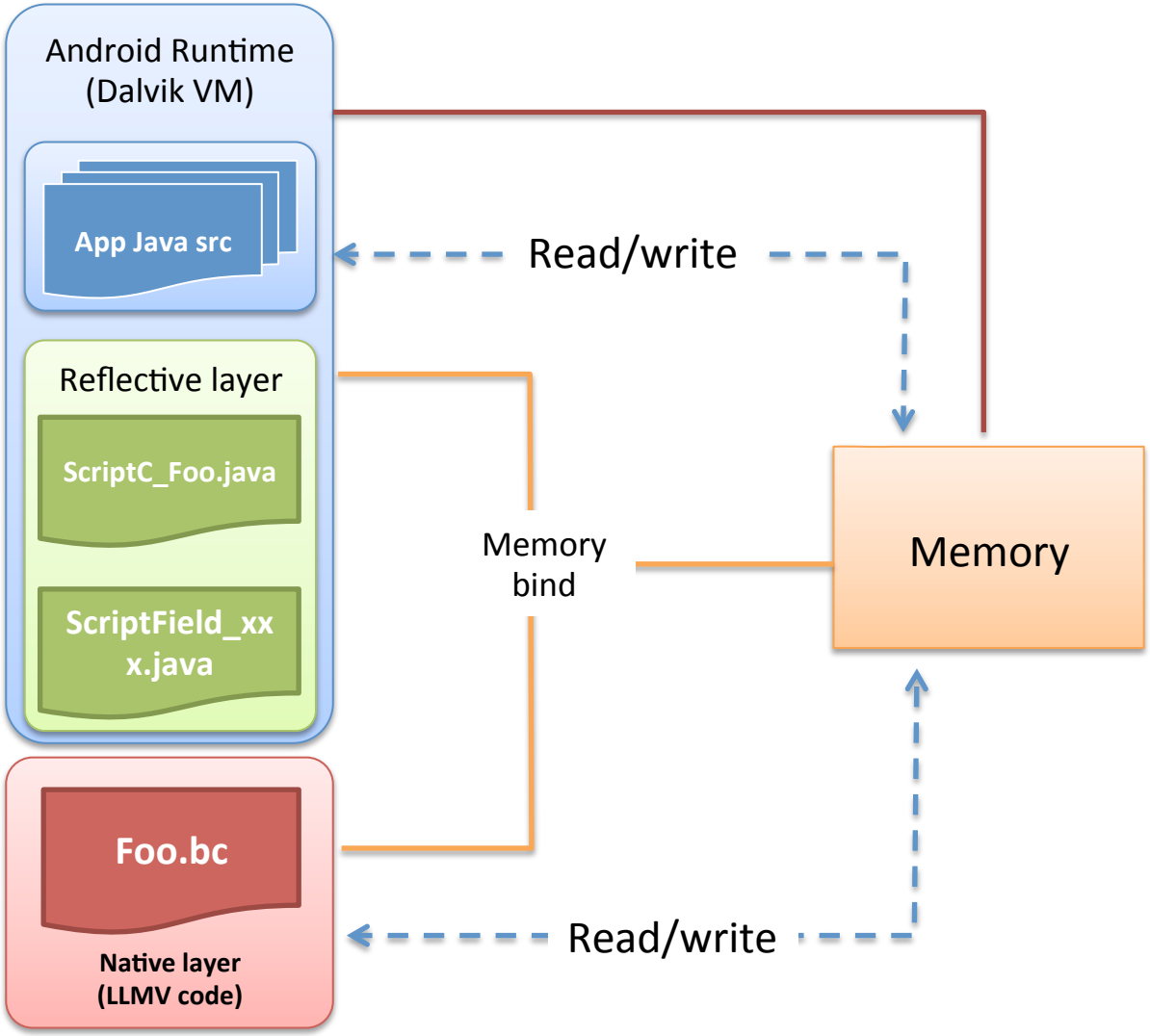
- Translate portable bitcode to appropriate machine code (CPU/GPU/DSP/...)



Runtime library support – libRS

- Manage scripts from Dalvik layer
- Also provide basic support libraries (math functions, etc.)
- Allows allocation, binding of objects to script globals
- Work distribution (multi-threading)
- Message-passing between Dalvik and script-side

At runtime


Android Runtime
(Dalvik VM)
App Java src
Reflective layer
ScriptC_Foo.java
ScriptField_xx
x.java
Foo.bc
Native layer
(LLMV code)
Read/write
Memory
bind
Memory
Read/write

Plan

- Introduction
- RenderScript Architecture
- Hello RenderScript!
- More RenderScript
- RenderScript and Multithreading

First example - HelloRS

First example - HelloRS

- Simple kernel to invert the color of the input image

ImageProcessing.rs

```
#pragma version(1)
#pragma rs java_package_name(fr.enseeiht.hellorscompute)
#pragma rs_fp_relaxed

// this kernel inverts the colors of the input image
uchar4 RS_KERNEL invert(uchar4 in)
{
   uchar4 out = in;
   out.r = 255 - in.r;
   out.g = 255 - in.g;
   out.b = 255 - in.b;
   return out;
}
```

First example - HelloRS

- Simple kernel to invert the color of the input image

ImageProcessing.rs

```
#pragma version(1)
#pragma rs java_package_name(fr.enseeiht.hellorscompute)
#pragma rs_fp_relaxed
```

#pragmas are used to tell the compiler:

- Renderscript version
- Package name of the Java classes
- Floating point precision
  - **#pragma rs_fp_full** (default): floating point precision as IEEE 754-2008 standard.
  - **#pragma rs_fp_relaxed**: can tolerate less precision -> faster
  - **#pragma rs_fp_imprecise**: no stringent precision requirements → even faster

Mapping Kernel

- Parallel function that operates on a collection of Allocations of the same dimensions.
- It executes once for every coordinate in those dimensions.
- Typically used to transform a collection of input Allocations to an output Allocation one Element at a time.
- One or more input Allocations, a single output Allocation, or both.
- The RenderScript runtime checks dimensions and Element types of I/O are the same;
  - Otherwise throws an exception.

Element

- The type of data being processed
- Some predefined types as
  - uchar
  - int
  - float
- And their vectorized version such as
  - uchar2, uchar3, uchar 4 (a vector of X uchar) etc..
  - They are packed in a struct that can be accessed like:

```
uchar4 out;
out.r = out.x = out.s0 = out.S0;
out.g = out.y = out.s1 = out.S1;
out.b = out.z = out.s2 = out.S2;
out.a = out.w = out.s3 = out.S3;
```

First example – HelloRS Java side

```
// the bitmap with the original image
private Bitmap mBitmapIn;
// the bitmap with the processed image
private Bitmap mBitmapOut;

// the renderscript context
private RenderScript mRS;
// input buffer used to call the renderscript
private Allocation mInAllocation;
// output buffer used to call the renderscript
private Allocation mOutAllocation;
// the object containing the RS kernels
private ScriptC_imageProcessing mScript;

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

   // load the original image
   mBitmapIn = loadBitmap(R.drawable.data);
   // create a clone bitmap with the same size
   mBitmapOut = Bitmap.createBitmap(mBitmapIn.getWidth(), mBitmapIn.getHeight(),
         mBitmapIn.getConfig());

   // widget for the original image
   ImageView in = (ImageView) findViewById(R.id.displayin);
   in.setImageBitmap(mBitmapIn);
   // widget for the processed image
   ImageView out = (ImageView) findViewById(R.id.displayout);
   out.setImageBitmap(mBitmapOut);

   // initialize the RS
   createScript();
}
```

RS “stuff”

Just the usual setting-up for the UI

First example – HelloRS Java side

How to create and use the RS script

```
/**
 * Initialize the script and apply the BW script
 */
private void createScript()
{
   mRS = RenderScript.create(this);

   // initialize the input buffer for RS
   mInAllocation = Allocation.createFromBitmap(mRS, mBitmapIn,
         Allocation.MipmapControl.MIPMAP_NONE,
         Allocation.USAGE_SCRIPT);
   // initialize the output buffer for RS
   mOutAllocation = Allocation.createFromBitmap(mRS, mBitmapOut,
         Allocation.MipmapControl.MIPMAP_NONE,
         Allocation.USAGE_SCRIPT);

   // create the object containing the RS kernels
   mScript = new ScriptC_imageProcessing(mRS);

   // apply the B/W filter
   mScript.forEach_mono(mInAllocation, mOutAllocation);

   // copy result to the output bitmap
   mOutAllocation.copyTo(mBitmapOut);
}
```

Context

Allocations

Create RS

Apply

Result

Allocation

createFromBitmap

added in API level 11

```
Allocation createFromBitmap (RenderScript rs,
                Bitmap b,
                Allocation.MipmapControl mips,
                int usage)
```

Creates an Allocation from a `Bitmap`.

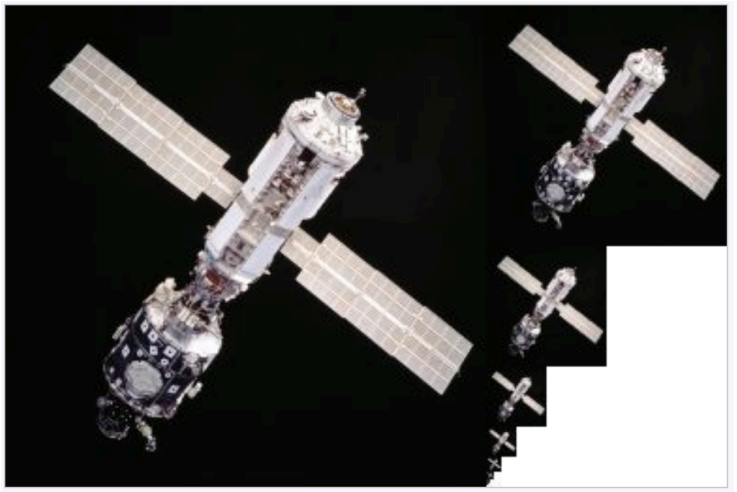
Parameters	
`rs`	`RenderScript`: Context to which the allocation will belong.
`b`	`Bitmap`: Bitmap source for the allocation data
`mips`	`Allocation.MipmapControl`: specifies desired mipmap behaviour for the allocation
`usage`	`int`: bit field specifying how the allocation is utilized

Returns	
`Allocation`	Allocation containing bitmap data

Allocation

Enum values	
`Allocation.MipmapControl`	`MIPMAP_FULL` A full mipmap chain will be created in script memory.
`Allocation.MipmapControl`	`MIPMAP_NONE` No mipmaps will be generated and the type generated from the incoming bitmap will not contain additional LODs.
`Allocation.MipmapControl`	`MIPMAP_ON_SYNC_TO_TEXTURE` The Type of the Allocation will be the same as MIPMAP_NONE.

mipmaps (or **pyramids)** are pre-calculated, optimized sequences of images, each of which is a progressively lower resolution representation of the same image.



https://en.wikipedia.org/wiki/Mipmap

Allocation

createFromBitmap

added in API level 11

```
Allocation createFromBitmap (RenderScript rs,
                Bitmap b,
                Allocation.MipmapControl mips,
                int usage)
```

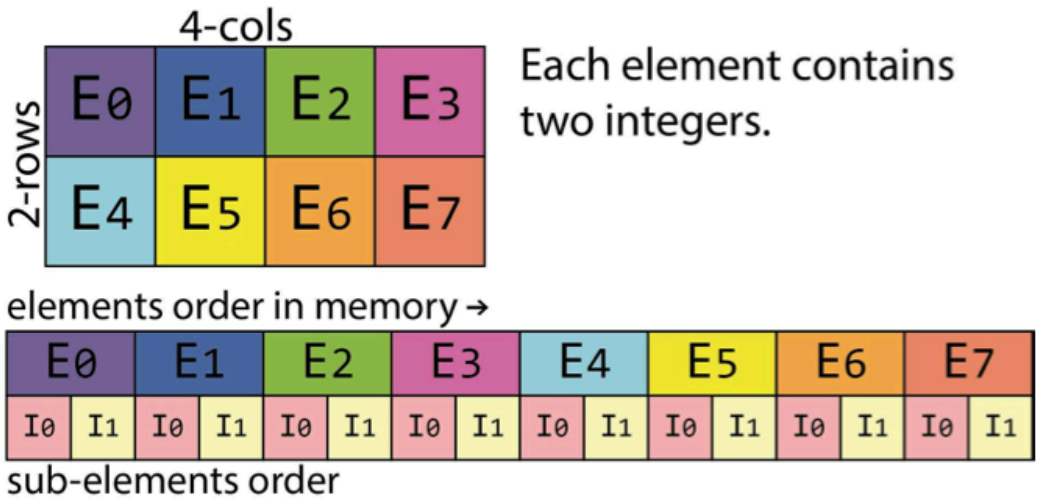
Creates an Allocation from a `Bitmap`.

Parameters	
`rs`	`RenderScript`: Context to which the allocation will belong.
`b`	`Bitmap`: Bitmap source for the allocation data
`mips`	`Allocation.MipmapControl`: specifies desired mipmap behaviour for the allocation
`usage`	`int`: bit field specifying how the allocation is utilized

`int`	`USAGE_SCRIPT` The Allocation will be bound to and accessed by scripts.
`int`	`USAGE_SHARED` The Allocation's backing store will be inherited from another object (usually a `Bitmap`); copying to or from the original source Bitmap will cause a synchronization rather than a full copy.

Allocation

- Memory blocks with bytes sequentially ordered
- Memory is contiguous with row-major indexing
- API provide many methods to create, copy data from other data structures
  - Bitmap, 1D-4D arrays,



First example – HelloRS Java side

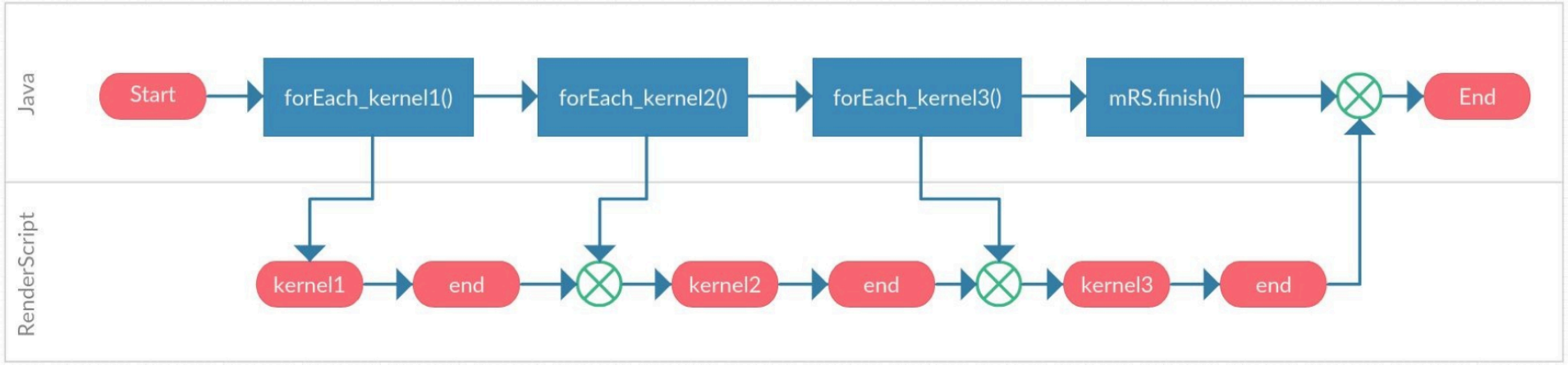
```
/**
 * It loads a bitmap from a resource
 * @param resource the resource ID to load
 * @return the bitmap corresponding to the resource ID
 */
private Bitmap loadBitmap(int resource)
{
    final BitmapFactory.Options options = new BitmapFactory.Options();
    options.inPreferredConfig = Bitmap.Config.ARGB_8888;
    return BitmapFactory.decodeResource(getResources(), resource, options);
}
```

Using RS in Java code

1. **Initialize a RenderScript context** create(Context), lifetime control of all subsequent RenderScript objects.
2. **Create Allocation(s) to be passed to a script.**
3. **Create whatever scripts are necessary.**
   - **ScriptC**: `filename.rs` is mapped into `ScriptC_filename.java`
   - **ScriptIntrinsic**: These are built-in RenderScript kernels for common operations,
4. **Populate Allocations with data.** createFromBitmap(), or one of the "copy" methods in Allocation.
5. **Launch the appropriate kernels and invokable functions.** Each kernel reflected method named `forEach_`*`mappingKernelName`*`()`
6. **Retrieve data from Allocation objects.**
7. **Tear down the RenderScript context.**

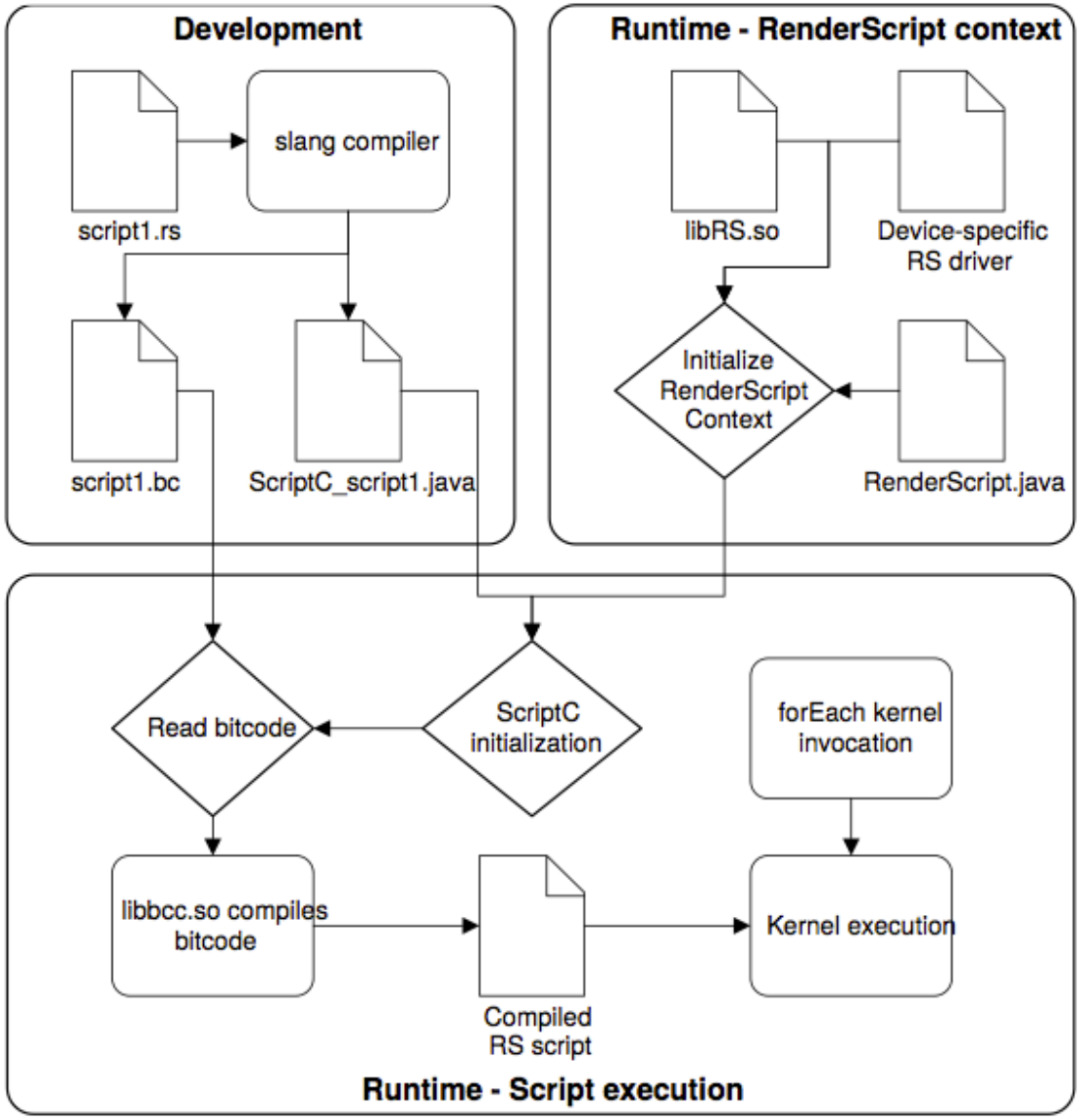
Kernel execution is asynchronous

- Kernel execution is asynchronous
- Queue of thread to be executed managed by the RS run-time
- Method mRS.finish() is a “join” equivalent that waits for the kernel(s) to finish



Asynchronous RenderScript kernels execution

To resume – workflow



Plan

- Introduction
- RenderScript Architecture
- Hello RenderScript!
- More RenderScript
- RenderScript and Multithreading

Kernel signature

- Kernels can operate on one input and one output at most
- E.g. only one input

```
// x and y are the indices of the current element (optional)
void RS_KERNEL kernelName(float in, uint32_t x, uint32_t y) {...}

// or
void kernelName(float *in, uint32_t x, uint32_t y) {...}
```

- Java side

```
// note x and y are NEVER passed as argument as the script works
// on the whole allocation
mScript.forEach_kernelName(mInAllocation);
```

Kernel signature

- Kernels can operate on one input and one output at most
- E.g. only one input and one output

```
// x and y are the indices of the current element (optional)
uint4 RS_KERNEL kernelName(float in, uint32_t x, uint32_t y) {...}

// or
void kernelName(float *in, uint4 *out, uint32_t x, uint32_t y) {...}
```

- Java side

```
// note x and y are NEVER passed as argument as the script works
// on the whole allocation
mScript.forEach_kernelName(mInAllocation, mOutAllocation);
```

Custom elements

- Custom elements can be declared as struct:

```
typedef struct MyElement
{
    int x;
    int y;
    bool simpleBool;
} MyElement;
```

- And they will be mapped in a (auto-generated) java class **ScriptField_MyElement**

```
public class ScriptField_MyElement extends android.renderscript.Script.FieldBase {
    static public class Item {
        public static final int sizeof = 12;

        int x;
        int y;
        boolean simpleBool;

        Item() {
        }
…
```

Creating Allocations (Java side)

- Allocations can be created from other objects (e.g. Bitmaps)
- Or "manually" using Type and Type.Builder

Example: plain array of 54 integers

```
// build a type builder of integers --> Element.I32
Type.Builder tb = new Type.Builder(mRS, Element.I32(mRS));

// set the x size
tb.setX(54);

// create the equivalent type
Type mType = tb.create();

// create the allocation from this type
Allocation mAlloc = Allocation.createTyped(mRS, mType, Allocation.USAGE_SCRIPT);

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// or, equivalently, in just one shot
mAlloc = Allocation.createSized(mRS, Element.I32(mRS), 54);
```

Creating Allocations (Java side)

- Allocations can be created from other objects (e.g. Bitmaps)
- Or "manually" using Type and Type.Builder

Example: a 5-rows per 6-columns matrix of floats

```
// build a type builder of float--> Element.F32
Type.Builder tb = new Type.Builder(mRS, Element.F32(mRS));

// set the x and y sizes
tb.setX(6);
tb.setY(5);

// create the equivalent type
Type mType = tb.create();

// create the allocation from this type
Allocation mAlloc = Allocation.createTyped(mRS, mType, Allocation.USAGE_SCRIPT);
```

Creating Allocations (Java side)

- Allocations can be created from other objects (e.g. Bitmaps)
- Or "manually" using Type and Type.Builder

Example: an array of 9 custom struct elements

```
// create an object of type MyElement
Element myElement = ScriptField_MyElement.createElement(mRS);

// build a type builder of MyElement
Type.Builder tb = new Type.Builder(mRS, myElement );

// set the x size
tb.setX(9);

// create the equivalent type
Type mType = tb.create();

// create the allocation from this type
Allocation mAlloc = Allocation.createTyped(mRS, mType, Allocation.USAGE_SCRIPT);

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// or, equivalently, in just one shot
mAlloc = Allocation.createSized(mRS, myElement, 9);
```

```
typedef struct MyElement
{
   int x;
   int y;
   bool simpleBool;
} MyElement;
```

Script Variables

- RS scripts allow to declare (global) variables
- They can be static or not
- Static variables is only visible inside the RS script
- non-static variables can be accessed from the Java side only to set its value, not to retrieve it.
- It is possible to retrieve only the compile-time values directly

Foo.rs

```
int myInt = 3;

void fooFunction(){
   myInt++;
    ...
```

```
mScript = new ScriptC_Foo(mRS);

mScript.set_myInt(15);

if(mScript.get_myInt() == 3)  //this will always return 3!
...
```

!

ScriptC_Foo will provide get and set accessors for myInt but get() will always return the compile time value

Script functions

- RS scripts allow to declare functions
- They can be static or not
- Static functions are only visible inside the RS script
- Non-static function are callable from Java side

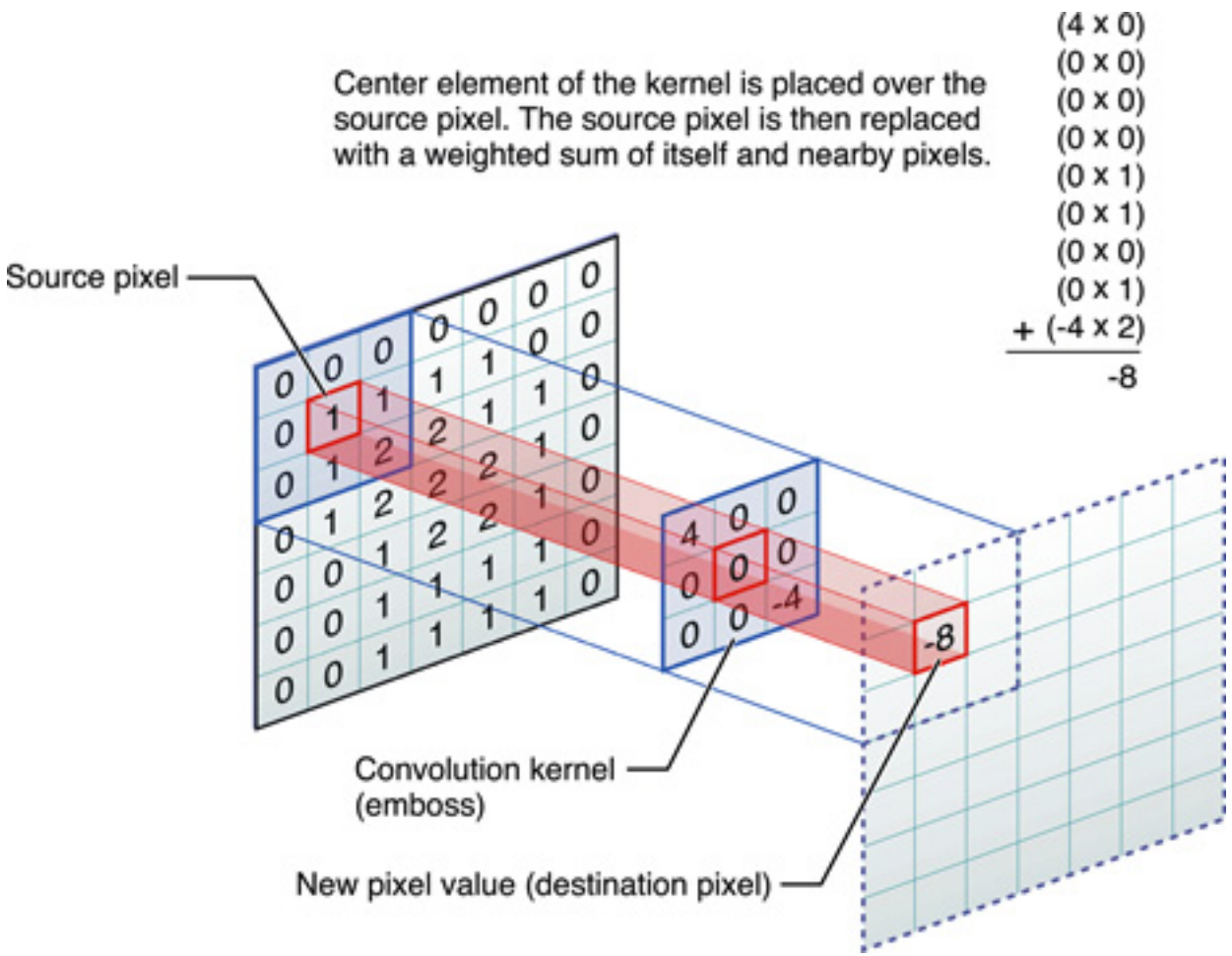
Foo.rs

```
int myInt = 3;

void modifyMyInt(int v) { myInt = v; }
```

```
mScript = new ScriptC_Foo(mRS);

mScript.invoke_modifyMyInt (15);
```

Example – Gaussian Blur

- Convolution with a gaussian kernel $$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



Gaussian kernel of size 3

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

The RS kernel

```
#pragma version(1)
#pragma rs java_package_name(fr.enseeiht.imageprocessing)

// the gaussian kernel
static const float k[9] = { 1.0f / 16.0f, 2.0f / 16.0f, 1.0f / 16.0f,
                            2.0f / 16.0f, 4.0f / 16.0f, 2.0f / 16.0f,
                            1.0f / 16.0f, 2.0f / 16.0f, 1.0f / 16.0f };
// the input allocation
rs_allocation in;

uchar4 RS_KERNEL make_gauss(uint32_t x, uint32_t y) {
   float4 f4out = { 0, 0, 0, 0};

   for (int kIdx = 0; kIdx < 9; kIdx++) {
      int kX = kIdx % 3 - 1;
      int kY = kIdx / 3 - 1;

      f4out += convert_float4(rsGetElementAt_uchar4(in, x + kX, y - kY)) * k[kIdx];

   }

   return convert_uchar4(f4out);
}
```

Java side

```
private void createScript()
{
   mRS = RenderScript.create(this);

   // initialize the input buffer for RS
   mInAllocation = Allocation.createFromBitmap(mRS, mBitmapIn,
         Allocation.MipmapControl.MIPMAP_NONE,
         Allocation.USAGE_SCRIPT);
   // initialize the output buffer for RS
   mOutAllocation = Allocation.createFromBitmap(mRS, mBitmapOut,
         Allocation.MipmapControl.MIPMAP_NONE,
         Allocation.USAGE_SCRIPT);

   // create the object containing the RS kernels
   mRSGauss3x3 = new ScriptC_gauss_for_3x3(mRS);

   // set the input allocation
   mRSGauss3x3.set_in(mInAllocation);

   // apply the B/W filter
   mScript.forEach_make_gauss(mOutAllocation);

   // copy result to the output bitmap
   mOutAllocation.copyTo(mBitmapOut);
}
```

The RS kernel

- We can do better
- Define radius of the kernel
  - Size in pixel of the kernel
- Let the user choose the radius
  - Expose the radius variable
  - And recompute the proper kernel

- OR…
- Use `android.renderscript.ScriptIntrinsicBlur` ☺

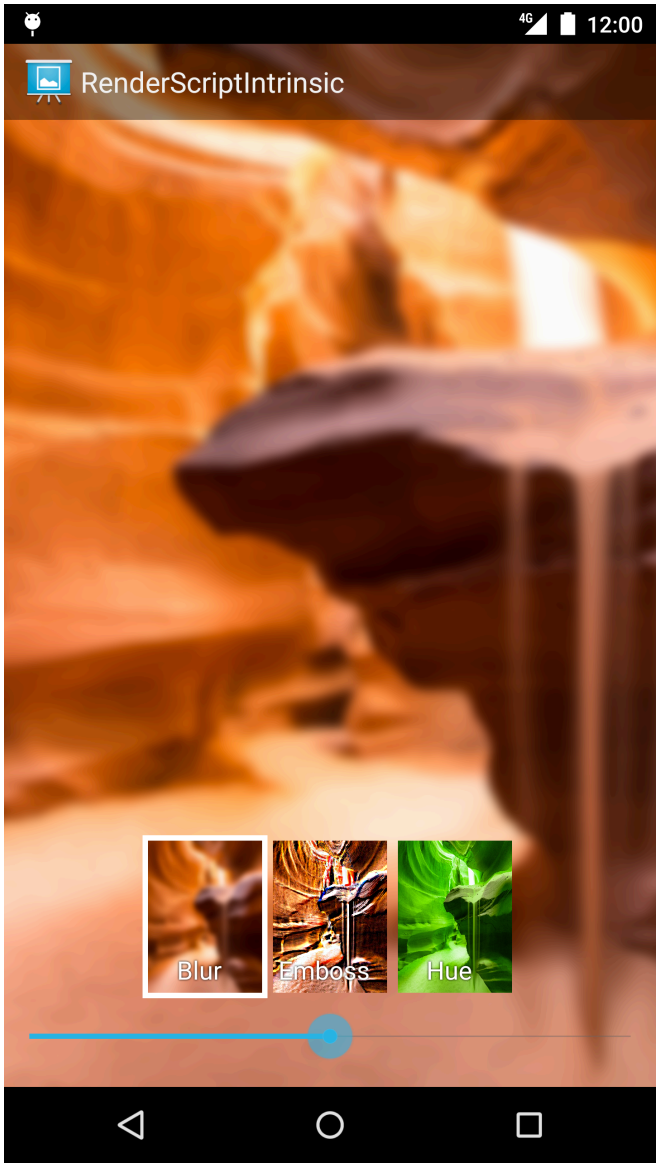
Public methods	
`static ScriptIntrinsicBlur`	`create(RenderScript rs, Element e)` Create an intrinsic for applying a blur to an allocation.
`void`	`forEach(Allocation aout)` Apply the filter to the input and save to the specified allocation.
`void`	`setInput(Allocation ain)` Set the input of the blur.
`void`	`setRadius(float radius)` Set the radius of the Blur.

RenderScriptIntrinsic

- Intrinsics provide effecient implementations of common operations.

android.renderscript.*	
ScriptIntrinsic3DLUT	converting RGB to RGBA by using a 3D lookup table
ScriptIntrinsicBlend	blending two Allocation objects.
ScriptIntrinsicBlur	Applies a gaussian blur of the specified radius
ScriptIntrinsicBLAS	Wrapper around BLAS routines
ScriptIntrinsicConvolve3x3	applying a 3x3 convolve
ScriptIntrinsicConvolve5x5	applying a 5x5 convolve
ScriptIntrinsicHistogram	calculating the histogram of an image
ScriptIntrinsicLUT	applying a per-channel lookup table
ScriptIntrinsicResize	performing a resize of a 2D allocation
ScriptIntrinsicYuvToRGB	converting an Android YUV buffer to RGB

RenderScriptIntrinsic



RenderScriptIntrinsic

```
private ScriptIntrinsicBlur mScriptBlur;
private ScriptIntrinsicConvolve5x5 mScriptConvolve;
private ScriptIntrinsicColorMatrix mScriptMatrix;
```

```
private void performFilter(Allocation inAllocation,
                    Allocation outAllocation, Bitmap bitmapOut, float value)
{
   switch (mFilterMode)
   {
      case MODE_BLUR:
         // Invoke filter blur kernel
         break;
      case MODE_CONVOLVE:
      {
         // Invoke filter convolve kernel
         break;
      }
      case MODE_COLORMATRIX:
      {
         //Invoke filter color matrix kernel
      }
      break;
   }

   // Copy to bitmap and invalidate image view
   outAllocation.copyTo(bitmapOut);
}
```

The value of the seek bar

RenderScriptIntrinsic

- The ScriptIntrinsicBlur

```
private void performFilter(Allocation inAllocation,
                     Allocation outAllocation, Bitmap bitmapOut, float value)
{
   switch (mFilterMode)
   {
      case MODE_BLUR:
         // Set blur kernel size
         mScriptBlur.setRadius(value);

         // Invoke filter kernel
         mScriptBlur.setInput(inAllocation);
         mScriptBlur.forEach(outAllocation);
         break;
}
```

RenderScriptIntrinsic

- The ScriptIntrinsicConvolve5x5

```
private void performFilter(Allocation inAllocation,
                    Allocation outAllocation, Bitmap bitmapOut, float value)
{
     case MODE_CONVOLVE:
      {
         float f1 = value;
         float f2 = 1.0f - f1;

         // Emboss filter kernel
         float coefficients[] = {-f1 * 2, 0, -f1, 0, 0, 0, -f2 * 2, -f2, 0,
               0, -f1, -f2, 1, f2, f1, 0, 0, f2, f2 * 2, 0, 0, 0, f1, 0,
               f1 * 2,};
         // Set kernel parameter
         mScriptConvolve.setCoefficients(coefficients);

         // Invoke filter kernel
         mScriptConvolve.setInput(inAllocation);
         mScriptConvolve.forEach(outAllocation);
         break;
```

RenderScriptIntrinsic

- The ScriptIntrinsicColorMatrix

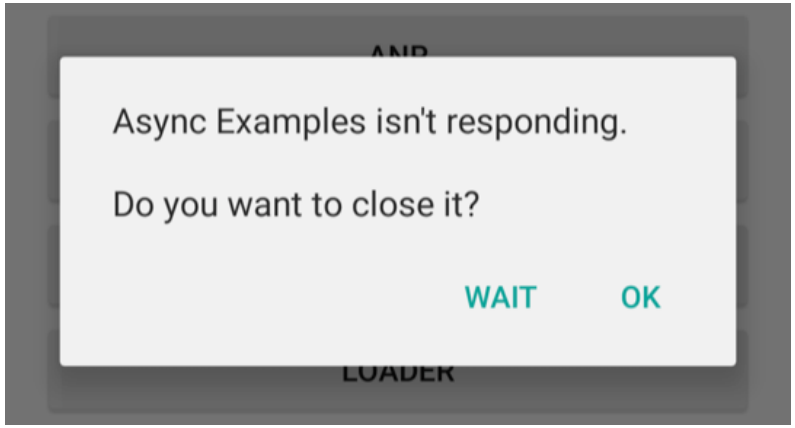
```
case MODE_COLORMATRIX:
      {
         /* Set HUE rotation matrix
          * The matrix below performs a combined operation of,
          * RGB->HSV transform * HUE rotation * HSV->RGB transform
          */
         float cos = (float) Math.cos((double) value);
         float sin = (float) Math.sin((double) value);
         Matrix3f mat = new Matrix3f();
         mat.set(0, 0, (float) (.299 + .701 * cos + .168 * sin));
         mat.set(1, 0, (float) (.587 - .587 * cos + .330 * sin));
         mat.set(2, 0, (float) (.114 - .114 * cos - .497 * sin));
         mat.set(0, 1, (float) (.299 - .299 * cos - .328 * sin));
         mat.set(1, 1, (float) (.587 + .413 * cos + .035 * sin));
         mat.set(2, 1, (float) (.114 - .114 * cos + .292 * sin));
         mat.set(0, 2, (float) (.299 - .3 * cos + 1.25 * sin));
         mat.set(1, 2, (float) (.587 - .588 * cos - 1.05 * sin));
         mat.set(2, 2, (float) (.114 + .886 * cos - .203 * sin));
         mScriptMatrix.setColorMatrix(mat);

         // Invoke filter kernel
         mScriptMatrix.forEach(inAllocation, outAllocation);

         break;
```

Plan

- Introduction
- RenderScript Architecture
- Hello RenderScript!
- More RenderScript
- RenderScript and Multithreading
  - (Multi-)Threading in Android
  - RS and multithreading

Threads in Android

- Application starts in a new Linux process with a single thread of execution.
- "main" thread or aka **UI thread**
  - By default, all components (Activities, Services, Broadcast Receiver etc) of the same application run in the same process and thread
- It is responsible for the graphical interface, drawing the UI, user interactions
- UI thread **cannot get blocked by "intensive work"**
- Otherwise we lose responsiveness and application can be killed by Android
- i.e. the infamous "application not responding" (ANR) dialog



Threads in Android

- When the app has to performs a potentially lengthy operation, **do not perform it on the UI thread**
  - E.g. downloads, network connections etc.
- Instead create a worker thread and do most of the work there.
- This keeps the UI thread running (responsiveness) and avoid frozen appearance of the app.
- The Andoid **UI toolkit is *not* thread-safe**. ==> you **MUST NOT** manipulate your UI from a worker thread
  - All UI manipulations must be done from the UI thread.

Rules for Android thread model
1. Do not block the UI thread
2. Do not access the Android UI toolkit from outside the UI thread

Multi-Threading in Android

- Use different thread to complete time-consuming tasks
- The threads will run in the same process alternating the execution on the CPU
- Responsiveness is maintained
- Could we use the `Thread` class from Java though?
- It depends... If the thread needs to access the UI to update it, it cannot be done
  - UI objects are NOT thread-safe.
- Android provides a class specifically designed for this --> `AsyncTask`

AsyncTask

- AsyncTask enables proper and easy use of the UI thread.
- It allows you to perform background operations and publish results on the UI thread without having to manipulate threads.
- Designed to be a helper class around Thread.
- Used for short operations (max few seconds
- An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread.

AsyncTask

```
private class myATask extends AsyncTask<Integer,Integer, Integer> {

   @Override
   protected void onPreExecute()
   {
      super.onPreExecute();
   }

   @Override
   protected Integer doInBackground(Integer... params)
   {
      publishProgress(50);
      return null;
   }

   @Override
   protected void onProgressUpdate(Integer... values)
   {
      super.onProgressUpdate(values);
   }

   @Override
   protected void onPostExecute(Integer result)
   {
      super.onPostExecute(result);
   }
}
```

AsyncTask

```
private class myATask extends AsyncTask<Integer,Integer, Integer> {

    @Override
    protected void onPreExecute()
    {
        super.onPreExecute();
    }
```

- invoked **on the UI thread** before the task is executed.
- E.g, setup of the task by showing a progress bar in the user interface.

```
    @Override
    protected void onProgressUpdate(Integer... values)
    {
        super.onProgressUpdate(values);
    }

    @Override
    protected void onPostExecute(Integer integer)
    {
        super.onPostExecute(integer);
    }
}
```

AsyncTask

```
private class myATask extends AsyncTask<Integer,Integer, Integer> {

    @Override
    protected void onPreExecute()
    {
        super.onPreExecute();
    }

    @Override
    protected Integer doInBackground(Integer... params)
    {
        publishProgress(50)
        return null;
    }

    }
}
```

- **invoked on the background thread** after onPreExecute() finishes.
- perform background computation that can take a long time.
- publishProgress(Progress...) used to publish one or more units of progress.
- This triggers onProgressUpdate(Progress...).

AsyncTask

```
private class myATask extends AsyncTask<Integer,Integer, Integer> {
```

- invoked **on the UI thread** after publishProgress(Progress...).
- Used to display any form of progress in the UI while the background computation is still executing.
  - animate a progress bar or show logs in a text field.

```
    @Override
    protected void onProgressUpdate(Integer... values)
    {
        super.onProgressUpdate(values);
    }

    @Override
    protected void onPostExecute(Integer integer)
    {
        super.onPostExecute(integer);
    }
}
```

AsyncTask

```
private class myATask extends AsyncTask<Integer,Integer, Integer> {

   @Override
   protected void onPreExecute()
   {
      super.onPreExecute();
   }
```

- **invoked on the UI thread** after the background computation finishes.
- The result of the background computation is passed to this step as a parameter.

```
   @Override
   protected void onPostExecute(Integer result)
   {
      super.onPostExecute(result);
   }
}
```

AsyncTask

Threading rules :

- The AsyncTask class must be loaded on the UI thread.
- The task instance must be created on the UI thread.
- execute(Params...) must be invoked on the UI thread.
- Do not call onPreExecute(), onPostExecute(Result), doInBackground(Params...), onProgressUpdate(Progress...) manually.
- The task can be executed only once (an exception will be thrown if a second execution is attempted.)

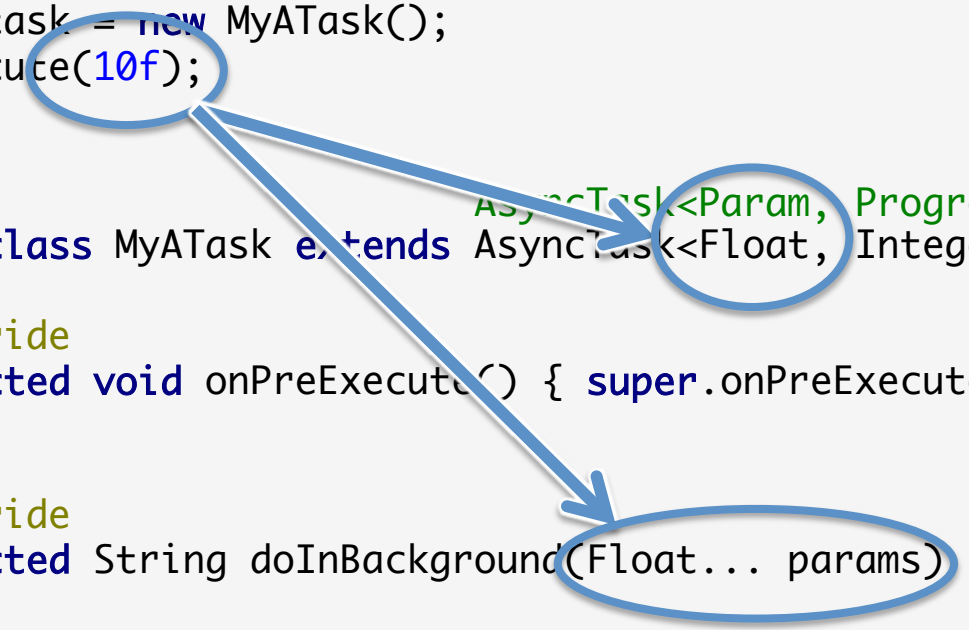
```
// create the task and execute it
MyATask task = new MyATask();
task.execute(10);

// or, equivalently, in just one shot without explicitly creating the object
new MyATask().execute(10);
```

AsyncTask

```
// create the task and execute it
MyATask task = new MyATask();
task.execute(10f);


//                              AsyncTask<Param, Progress, Result>
private class MyATask extends AsyncTask<Float, Integer, String> {

   @Override
   protected void onPreExecute() { super.onPreExecute(); }


   @Override
   protected String doInBackground(Float... params)
   {
      publishProgress(50);
      return "I'm done";
   }

   @Override
   protected void onProgressUpdate(Integer... values) { super.onProgressUpdate(values); }

   @Override
   protected void onPostExecute(String result) { super.onPostExecute(result); }

}
```

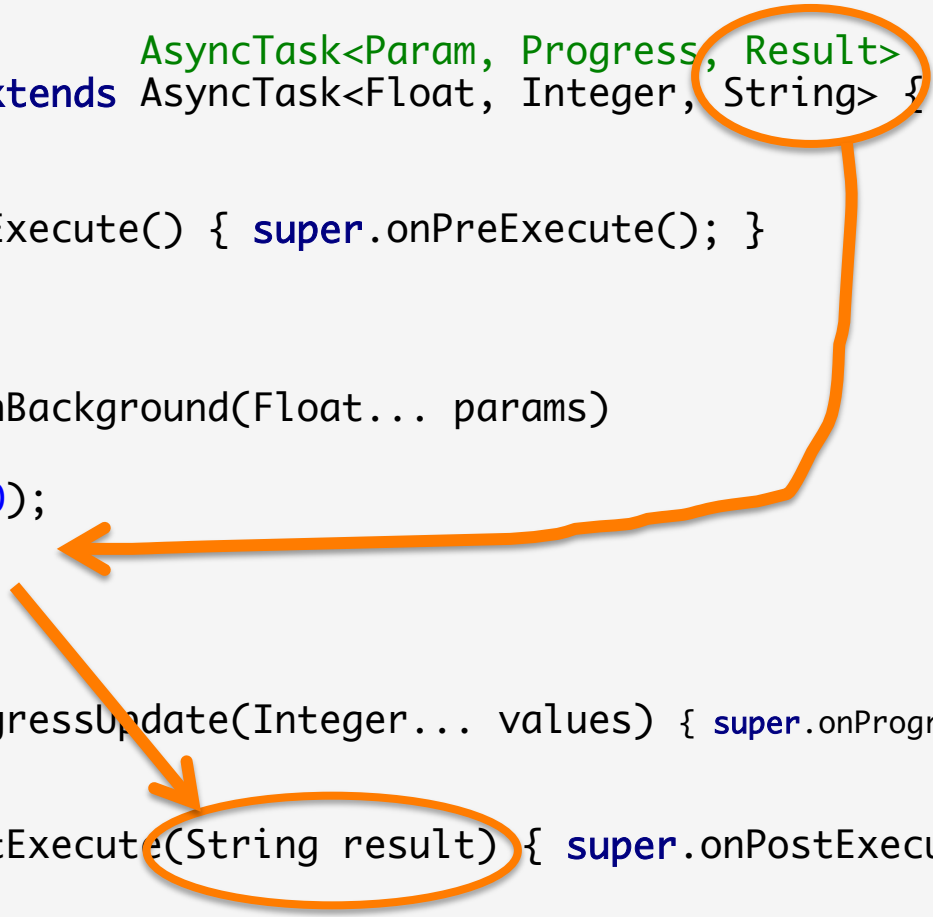
AsyncTask

```
// create the task and execute it
MyATask task = new MyATask();
task.execute(10f);


//                                  AsyncTask<Param, Progress, Result>
private class MyATask extends AsyncTask<Float, Integer, String> {

    @Override
    protected void onPreExecute() { super.onPreExecute(); }


    @Override
    protected String doInBackground(Float... params)
    {
        publishProgress(50);
        return "I'm done";
    }

    @Override
    protected void onProgressUpdate(Integer... values) { super.onProgressUpdate(values); }

    @Override
    protected void onPostExecute(String result) { super.onPostExecute(result); }

}
```



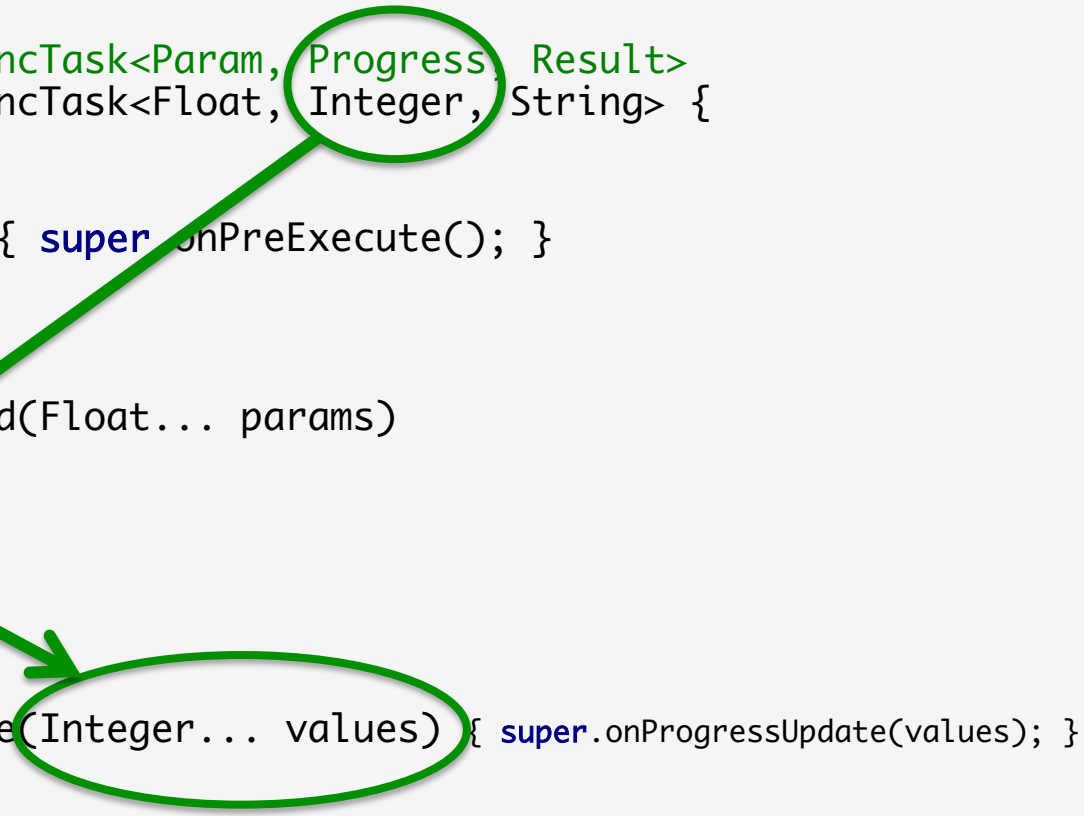
AsyncTask

```
// create the task and execute it
MyATask task = new MyATask();
task.execute(10f);


//                              AsyncTask<Param, Progress, Result>
private class MyATask extends AsyncTask<Float, Integer, String> {

   @Override
   protected void onPreExecute() { super.onPreExecute(); }


   @Override
   protected String doInBackground(Float... params)
   {
      publishProgress(50);
      return "I'm done";
   }

   @Override
   protected void onProgressUpdate(Integer... values) { super.onProgressUpdate(values); }

   @Override
   protected void onPostExecute(String result) { super.onPostExecute(result); }

}
```

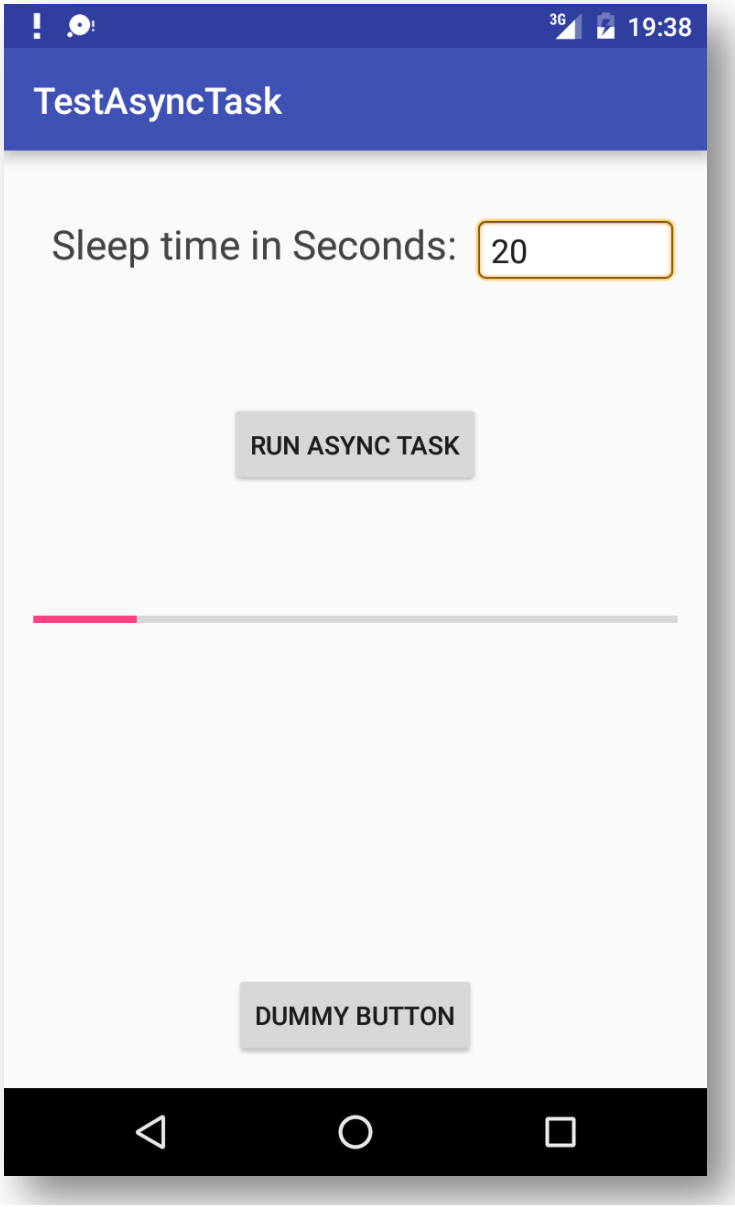


AsyncTask

```
// create the task and execute it
MyATask task = new MyATask();
task.execute(10f);


//                                AsyncTask<Param, Progress, Result>
private class MyATask extends AsyncTask<Float, Integer, String> {

   @Override
   protected void onPreExecute() { super.onPreExecute(); }


   @Override
   protected String doInBackground(Float... params)
   {
      publishProgress(50);
      return "I'm done";
   }

   @Override
   protected void onProgressUpdate(Integer... values) { super.onProgressUpdate(values); }

   @Override
   protected void onPostExecute(String result) { super.onPostExecute(result); }

}
```



AsyncTask - Example



AsyncTask - Example

```
private final String TAG = "TestAsync";
private final static int PROGRESS_MAX = 100;
private ProgressBar mProgressBar;
private Button mStartButton;
private EditText mTimeInput;

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

   mProgressBar = (ProgressBar) findViewById(R.id.progressBar);
   mProgressBar.setMax(PROGRESS_MAX);

   mTimeInput = (EditText) findViewById(R.id.in_time);

   mStartButton = (Button) findViewById(R.id.btn_run);
   mStartButton.setOnClickListener(new View.OnClickListener()
   {
      @Override
      public void onClick(View v)
      {
          // get the number of second to sleep from the edit text
          int msToSleep = Integer.parseInt(mTimeInput.getText().toString());
          new AsyncTaskRunner().execute(msToSleep);
          Log.d(TAG, "launched runner for sleeping " + msToSleep);
      }
   });

}
```

AsyncTask - Example

```
// AsyncTask<Params, Progress, Result>
private class AsyncTaskRunner extends AsyncTask<Integer, Integer, Void>
{

   @Override
   protected void onPreExecute()
   {
      super.onPreExecute();
      mProgressBar.setProgress(0);
   }

   @Override
   protected Void doInBackground(Integer... params)
   {
      // how many 100ms in given seconds
      // param is given in seconds
      int numLoops = params[0] * 10;
      Log.d(TAG, "AsyncTaskRunner is gonna do " + numLoops + " loops");
      for(int i = 0; i < numLoops; ++i)
      {
          // sleep 100ms
          Thread.sleep(100);

          // publish update as % of the number of loops
          publishProgress((int) PROGRESS_MAX * i / numLoops);

          Log.d(TAG, "AsyncTaskRunner publishing " + ((int) PROGRESS_MAX * i / numLoops) + " progress");
      }
      return null;
   }
```

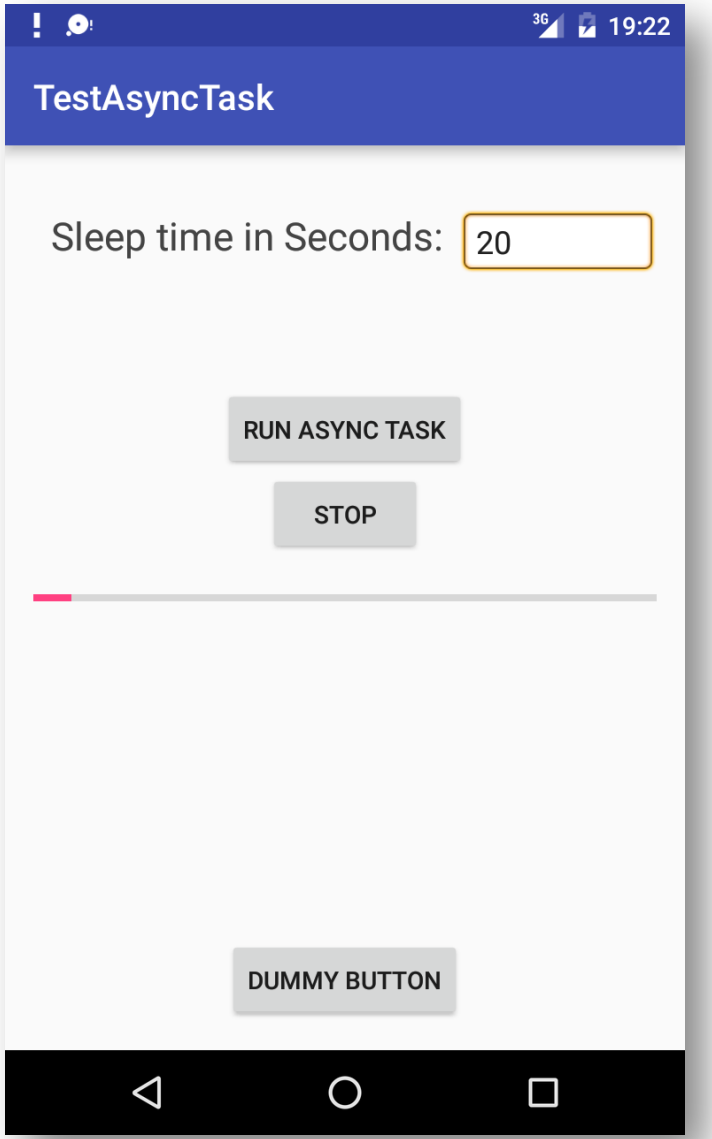
Remember, this will be executed by the UI thread, that's why it is safe to manipulate `mProgressBar`

AsyncTask - Example

```
// AsyncTask<Params, Progress, Result>
private class AsyncTaskRunner extends AsyncTask<Integer, Integer, Void>
{


   @Override
   protected void onProgressUpdate(Integer... values)
   {
      super.onProgressUpdate(values);
      // update the progress bar
      mProgressBar.setProgress(values[0]);

      Log.d(TAG, "AsyncTaskRunner setting progress bar to " + values[0] + " progress");
   }

   @Override
   protected void onPostExecute(Void aVoid)
   {
      super.onPostExecute(aVoid);
      // set the progress bar to full
      mProgressBar.setProgress(PROGRESS_MAX);
   }
}
```

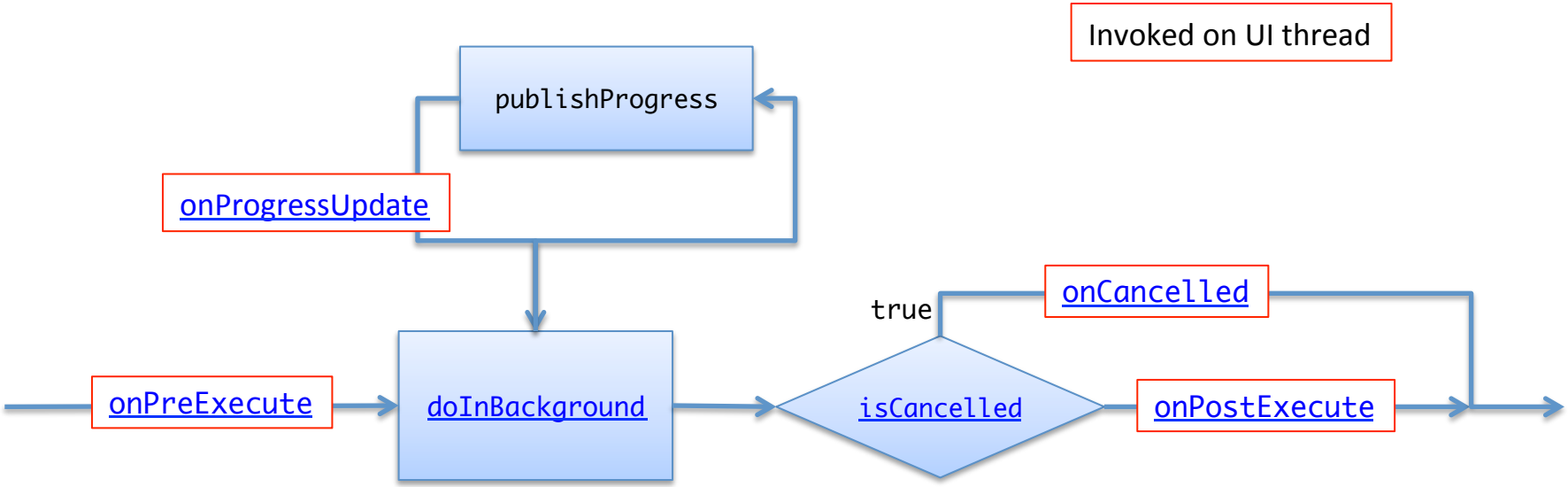
Again, the 2 callbacks will be executed by the UI thread, that's why it is safe to manipulate `mProgressBar`

AsyncTask – Cancel the task



AsyncTask – Cancel the task

- A task can be cancelled at any time by invoking `cancel(boolean)`.
- Then method `isCancelled()` will return true.
- After `cancel()`, `onCancelled(Object)`, instead of `onPostExecute(Object)` will be **invoked on the UI thread** after `doInBackground(Object[])` returns.
- To cancel as quickly as possible, check the return value of `isCancelled()` periodically from `doInBackground(Object[])`



AsyncTask – Cancel the task

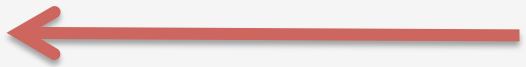
```
mStopButton = (Button) findViewById(R.id.btn_stop);
// initially the stop button is disabled
mStopButton.setEnabled(false);
mStopButton.setOnClickListener(new View.OnClickListener()
{
   @Override
   public void onClick(View v)
   {
      // if we have already a task
      if (mAsyncTask != null)
      {
         // cancel it and disable the stop button
         mAsyncTask.cancel(true);
         mStopButton.setEnabled(false);
      }
   }
});

mStartButton = (Button) findViewById(R.id.btn_run);
mStartButton.setOnClickListener(new View.OnClickListener()
{
   @Override
   public void onClick(View v)
   {
      ...
```

AsyncTask – Cancel the task

```
mStartButton = (Button) findViewById(R.id.btn_run);
mStartButton.setOnClickListener(new View.OnClickListener()
{
   @Override
   public void onClick(View v)
   {
         int msToSleep = Integer.parseInt(mTimeInput.getText().toString());

         // if we have already a task, cancel it
         if (mAsyncTask != null)
            mAsyncTask.cancel(true);

         // create a new task
         mAsyncTask = new AsyncTaskRunner();
         mAsyncTask.execute(msToSleep);

         // enable the stop button
         mStopButton.setEnabled(true);

         Log.d(TAG, "launched runner for sleeping " + msToSleep);

   }
});
```

AsyncTask – Cancel the task

```
// AsyncTask<Params, Progress, Result>
private class AsyncTaskRunner extends AsyncTask<Integer, Integer, Void>
{

   @Override
   protected Void doInBackground(Integer... params)
   {
      // how many 100ms in given seconds
      // param is given in seconds
      int numLoops = params[0] * 10;

      for(int i = 0; i < numLoops; ++i)
      {
         if(isCancelled())
            return null;

         // sleep 100ms
         Thread.sleep(100);
         // publish update as % of the number of loops
         publishProgress((int) PROGRESS_MAX * i / numLoops);
      }
      return null;
   }
```



AsyncTask – Cancel the task

```
// AsyncTask<Params, Progress, Result>
private class AsyncTaskRunner extends AsyncTask<Integer, Integer, Void>
{


   @Override
   protected void onCancelled(Void aVoid)
   {
      super.onCancelled(aVoid);
      Log.d(TAG, "AsyncTaskRunner has been cancelled :-(");
   }


   @Override
   protected void onPostExecute(Void aVoid)
   {
      super.onPostExecute(aVoid);
      // set the progress bar to full
      mProgressBar.setProgress(PROGRESS_MAX);
      mStopButton.setEnabled(false);
   }
}
```

Plan

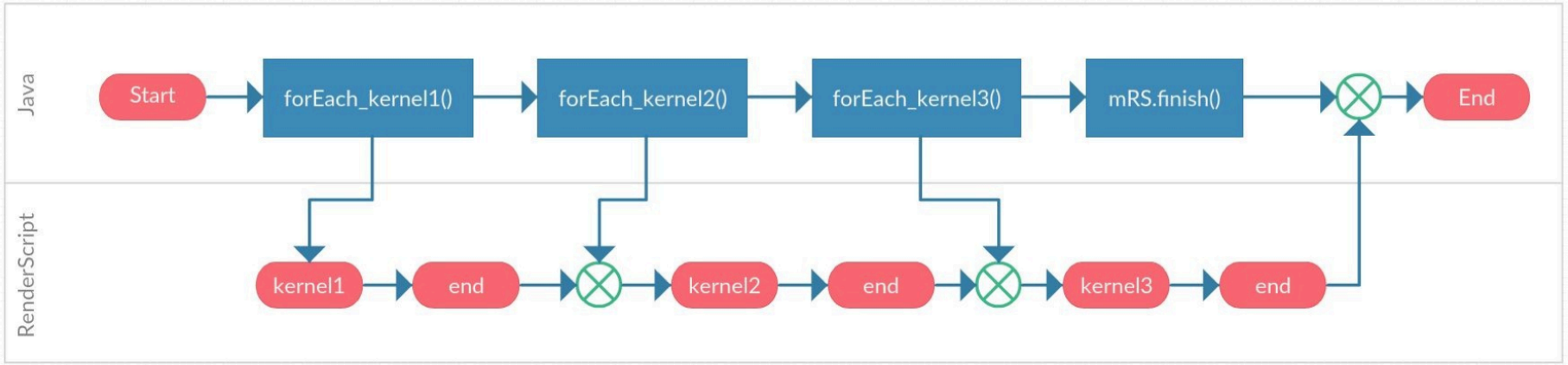
- Introduction
- RenderScript Architecture
- Hello RenderScript!
- More RenderScript
- RenderScript and Multithreading
  - (Multi-)Threading in Android
  - RS and multithreading

Improved RS with AsyncTask

- Use AsyncTask to launch RS computation
- The UI thread is not blocked waiting for the results
- Once the computation is done the UI updates are done through onPostExecute

```
// create the object containing the RS kernels
mScript = new ScriptC_imageProcessing(mRS);

// apply the B/W filter
mScript.forEach_mono(mInAllocation, mOutAllocation);

// copy result to the output bitmap
mOutAllocation.copyTo(mBitmapOut);
```

This is blocking, implicit "join" to copy only after kernel finishes



Asynchronous RenderScript kernels execution

Saturation with RS and user interaction

RS saturation

```
#pragma version(1)
#pragma rs java_package_name(com.example.android.basicrenderscript)
#pragma rs_fp_relaxed

const static float3 gMonoMult = {0.299f, 0.587f, 0.114f};

float saturationValue = 0.f;

/*
RenderScript kernel that performs saturation manipulation.
*/
uchar4 RS_KERNEL saturation(uchar4 in)
{
    float4 f4 = rsUnpackColor8888(in);
    float3 result = dot(f4.rgb, gMonoMult);
    // mix -> result + ((f4.rgb - result) * saturationValue)
    result = mix( result, f4.rgb, saturationValue );

    return rsPackColorTo8888(result);
}
```

If we do it without multi-threading

```
private RenderScript mRS;
private Allocation mInAllocation;
private Allocation mOutAllocations;
private ScriptC_saturation mScript;

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

   // Initialize UI
   mBitmapIn = loadBitmap(R.drawable.data);
   mBitmapsOut = Bitmap.createBitmap(mBitmapIn.getWidth(), mBitmapIn.getHeight(), ...);

   mImageView = (ImageView) findViewById(R.id.imageView);
   mImageView.setImageBitmap(mBitmapsOut);

   SeekBar seekbar = (SeekBar) findViewById(R.id.seekBar1);
   seekbar.setProgress(50);
   seekbar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener()
   {
      public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser)
      {
         float max = 2.0f;
         float min = 0.0f;
         float f = (float) ((max - min) * (progress / 100.0) + min);
         updateImage(f);
      }
```

This calls the RS – see next slides

If we do it without multi-threading

Function to initialize the script, exactly as before

```
/*
 * Initialize RenderScript
 * In the sample, it creates RenderScript kernel that performs saturation
manipulation.
 */
private void createScript()
{
   // Initialize RS
   mRS = RenderScript.create(this);

   // Allocate buffers
   mInAllocation = Allocation.createFromBitmap(mRS, mBitmapIn);
   mOutAllocations = Allocation.createFromBitmap(mRS, mBitmapsOut);

   //Load script
   mScript = new ScriptC_saturation(mRS);
}
```

If we do it without multi-threading

Function launching the RS script from the UI thread, just like before...

```
private void updateImage(final float f)
{

   // Set global variable in RS
   mScript.set_saturationValue(f);

   // Invoke saturation filter kernel
   mScript.forEach_saturation(mInAllocation, mOutAllocations);


   // Copy to bitmap and invalidate image view
   mOutAllocations.copyTo(mBitmapsOut);

   mImageView.setImageBitmap(mBitmapsOut);
   mImageView.invalidate();

}
```

Ehi, it works fine to me!

Yes!
But we are lucky because the kernel is actually executed quite quickly and it does not affect the responsiveness.

But try this...

```
// Set global variable in RS
mScript.set_saturationValue(f);

// Invoke saturation filter kernel
for(int i = 0; i < 1000; ++i)
  mScript.forEach_saturation(mInAllocation, mOutAllocations);


// Copy to bitmap and invalidate image view
mOutAllocations.copyTo(mBitmapsOut);
```

Not so fine now, isn't it?

With multithreading

- We just need to define a proper AsyncTask to do the job of launching the RS and update the UI when done.
- Everything is the same but...
- The updateImage() function becomes just

```
/**
 * @brief Invoke AsynchTask and cancel previous task.
 * When AsyncTasks are piled up (typically in slow device with heavy kernel),
 * Only the latest (and already started) task invokes RenderScript operation.
 * @param f the value for the kernel parameter
 */
private void updateImage(final float f)
{
     // if there is already a running task cancel it
    if (currentTask != null)
        currentTask.cancel(false);
    // create a new one
    currentTask = new RenderScriptTask();
    currentTask.execute(f);
}
```

With multithreading

```
/*
 * In the AsyncTask, it invokes RenderScript intrinsics to do a filtering.
 * After the filtering is done, an operation blocks at Allication.copyTo() in AsyncTask
thread.
 * Once all operation is finished at onPostExecute() in UI thread, it can invalidate and
 * update ImageView UI.
 */
private class RenderScriptTask extends AsyncTask<Float, Integer, Integer>
{
   Boolean issued = false;

   protected Integer doInBackground(Float... values)
   {
      if (isCancelled() == false)
      {
         issued = true;

         mScript.set_saturationValue(values[0]);

         // Invoke saturation filter kernel
         mScript.forEach_saturation(mInAllocation, mOutAllocations);

         // Copy to bitmap and invalidate image view
         mOutAllocations.copyTo(mBitmapsOut);
      }
      return 0;
   }
```

With multithreading

```
/*
 * In the AsyncTask, it invokes RenderScript intrinsics to do a filtering.
 * After the filtering is done, an operation blocks at Allication.copyTo() in AsyncTask
thread.
 * Once all operation is finished at onPostExecute() in UI thread, it can invalidate and
 * update ImageView UI.
 */
private class RenderScriptTask extends AsyncTask<Float, Integer, Integer>
{
   Boolean issued = false;

   void updateView(Integer result)
   {
      if (result != -1)
      {
         // Request UI update
         mImageView.setImageBitmap(mBitmapsOut);
         mImageView.invalidate();
      }
   }

   protected void onPostExecute(Integer result)
   {
      updateView(result);
   }

   protected void onCancelled(Integer result)
   {
      if (issued)
         updateView(result);
   }
}
```

Ok, but what if I put that for loop?


```
// Set global variable in RS
mScript.set_saturationValue(values[0]);

for(int i = 0; i < 1000; ++i)
{
   if (isCancelled())
      return -1;

   // Invoke saturation filter kernel
   mScript.forEach_saturation(mInAllocation, mOutAllocations);
}
```

This allows early exit

Much better, isn't it? ☺

References

- Introducing RenderScript, https://android-developers.googleblog.com/2011/02/introducing-renderscript.html
- Alberto Marchetti, RenderScript: parallel computing on Android, the easy way.
- RenderScript, https://developer.android.com/guide/topics/renderscript/compute.html
- Processes and Threads, https://developer.android.com/guide/components/processes-and-threads.html
- Keeping Your App Responsive, https://developer.android.com/training/articles/perf-anr.html
- AsyncTask API, https://developer.android.com/reference/android/os/AsyncTask.html