# GPUs for Mobile Malware, Mitigation and More

by Jared Carlson

# About Myself

I'm a researcher in the Boston area

Have worked and consulted for variety of companies

Sr. Engineer @ viaForensics

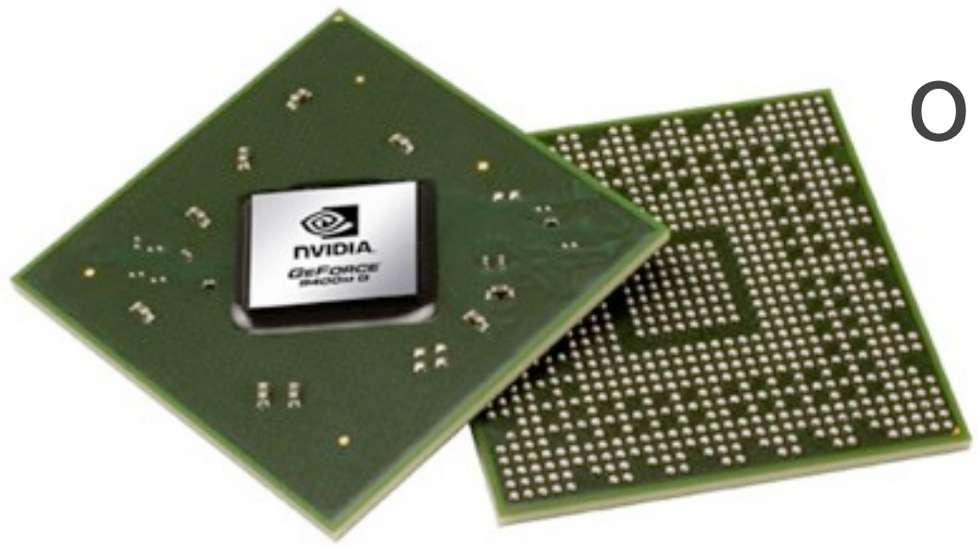
We're hiring

# Why?

# Why?

Explosive Growth

Can be used for
Offensive
or Defensive tactics




Highly Capable

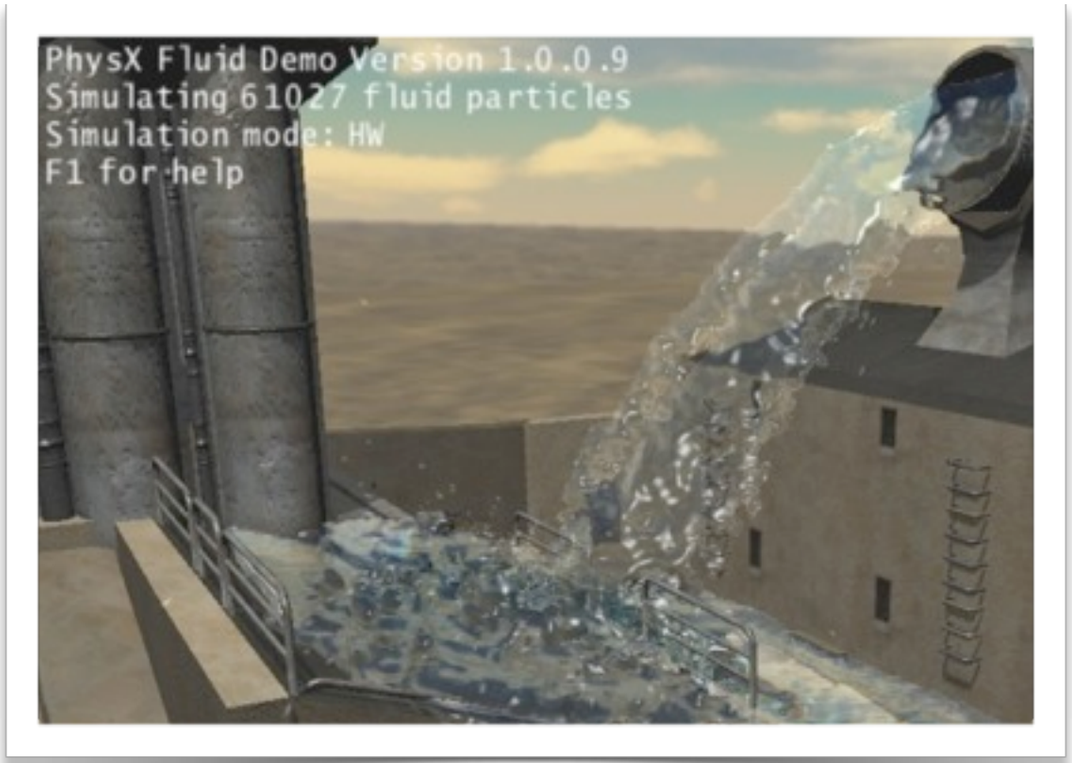
SoC/PoP on Mobile
Highly Integrated

# Generalizing GPUs

What do folks do on GPUs?

Physics

Crypto




OpenCL

Heterogenous computing

CUDA

Using GPU for general purpose

# Motivation for Mobile

# Motivation for Mobile

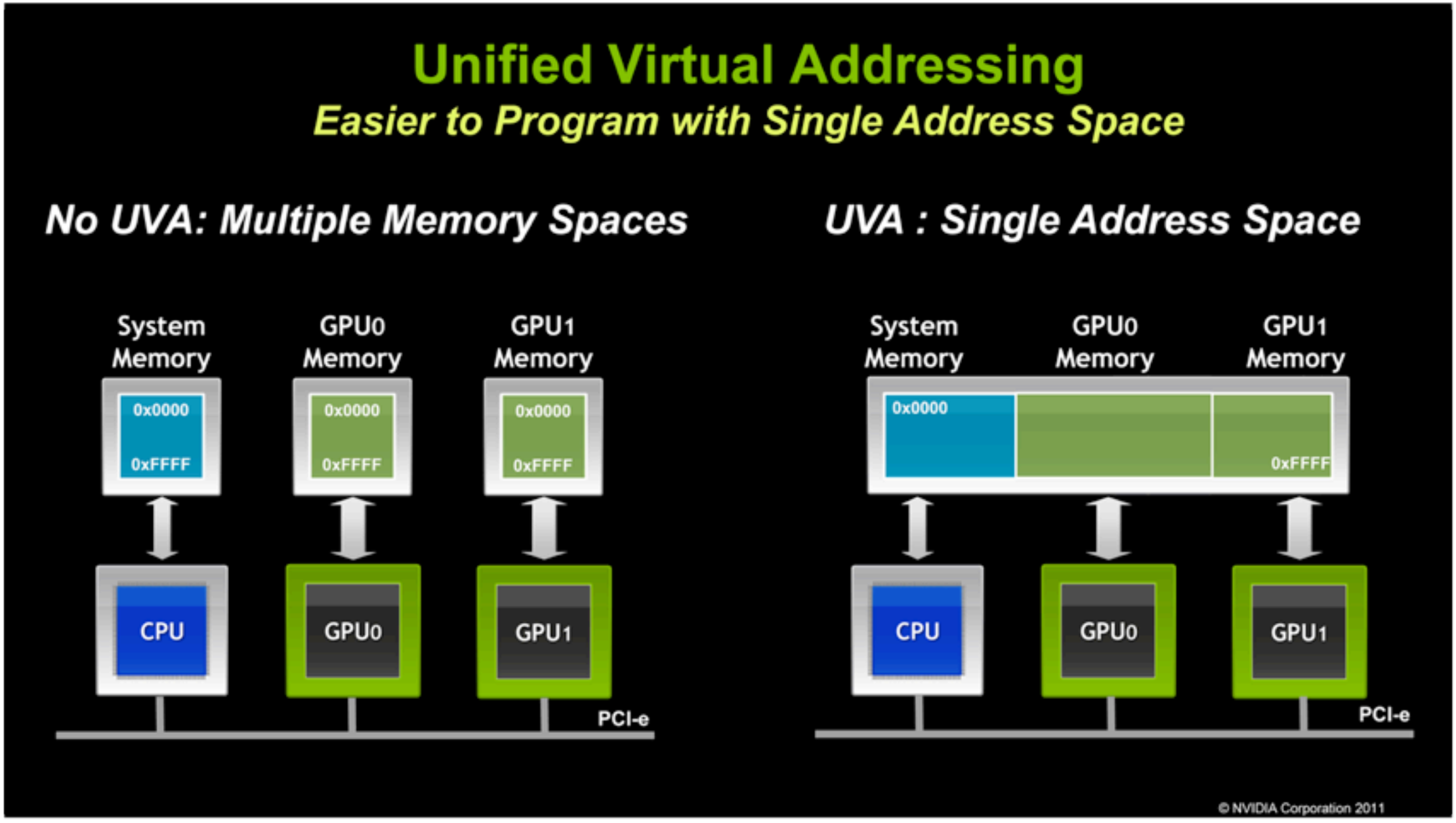
Increased surface area

Offloading tasks from the CPU

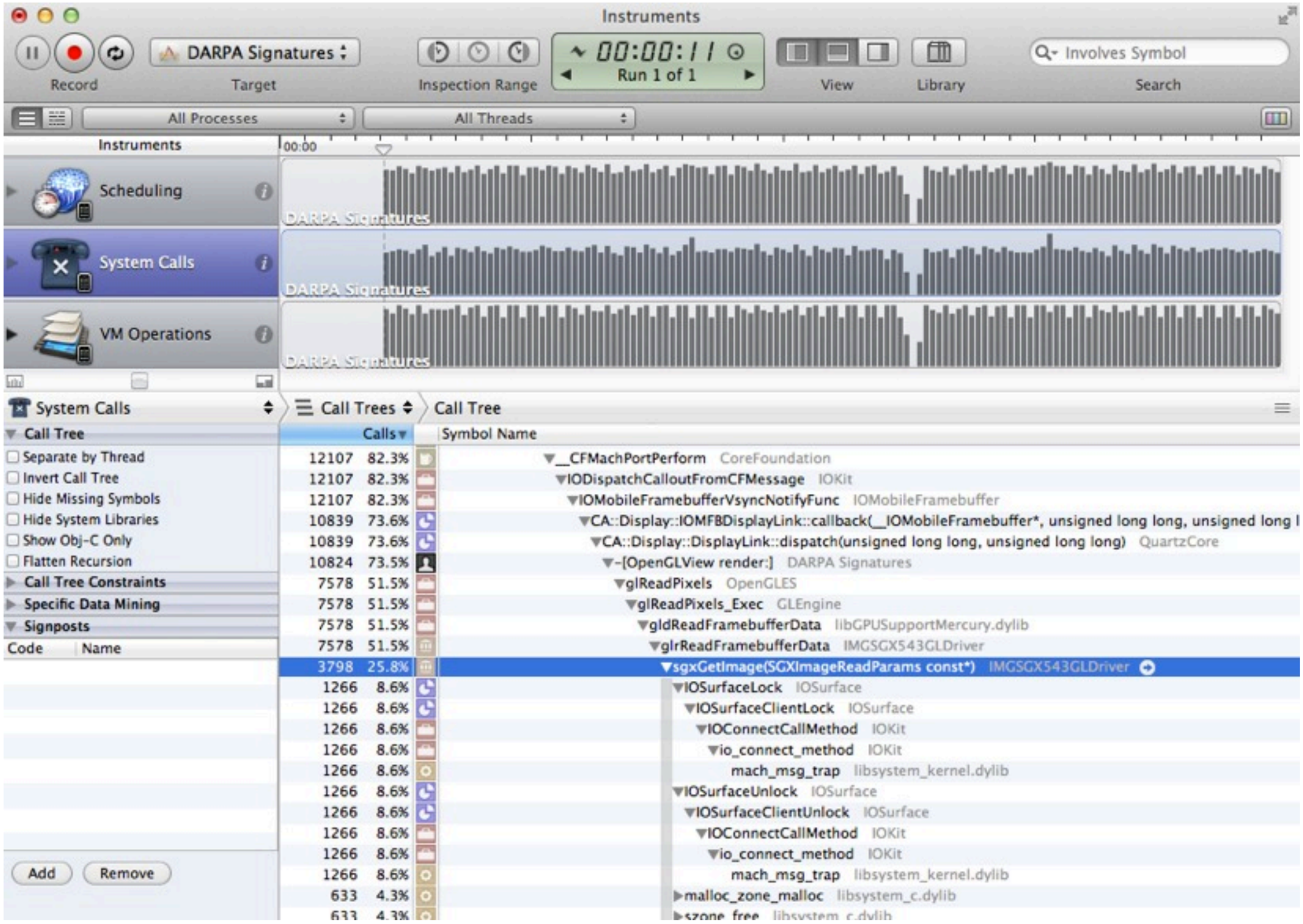
Code signing?  Nope...

Easily (re)compiled

# Unified Addressing



# Call Stack for GL



# What can we do?

# What can we do?

Signatures

Track Dynamic Memory

Disassembly

Encryption

And more...

# GL Example

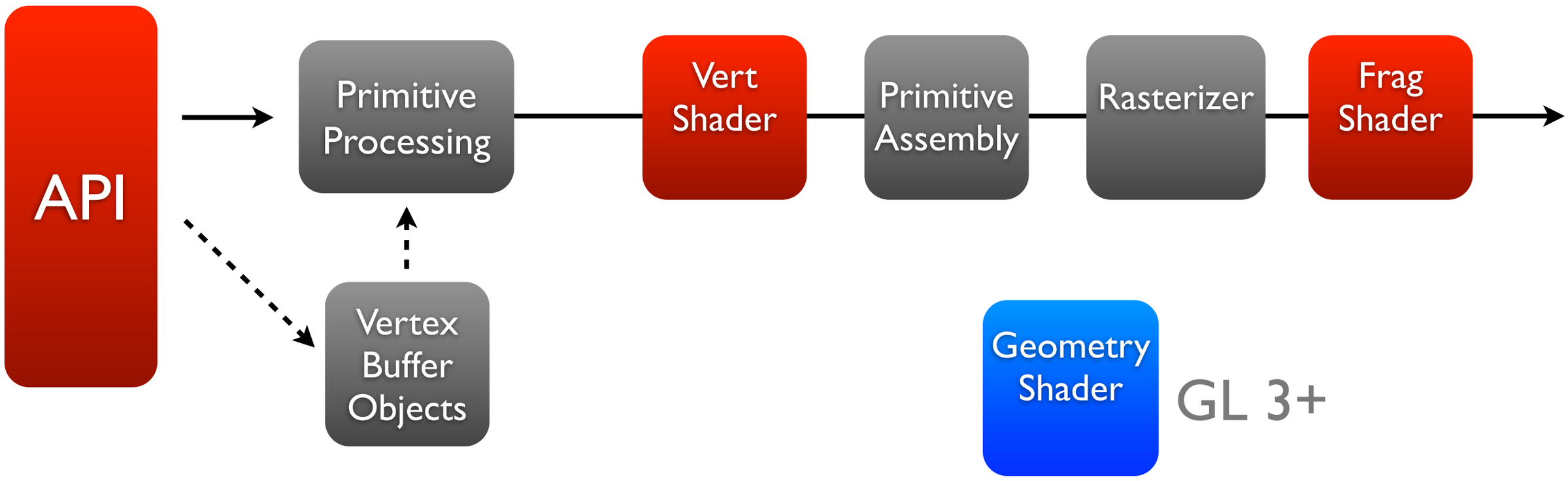
## What a shader looks like

```
#ifdef GL_ES
precision highp float;
#endif

uniform mat4 modelViewMatrix;
uniform mat4 modelViewProjectionMatrix;
uniform mat3 normalMatrix;

#if __VERSION__ >= 140
in vec3  inNormal;
in vec4  inPosition;
out vec3 varNormal;
out vec3 varEyeDir;
#else
attribute vec3 inNormal;
attribute vec4 inPosition;
varying vec3  varNormal;
varying vec3  varEyeDir;
#endif

void main (void)
{
    gl_Position    = modelViewProjectionMatrix * inPosition;
    vec4 eyePos = modelViewMatrix * inPosition;

    varNormal = normalize(normalMatrix * inNormal);
    varEyeDir = eyePos.xyz;
}
```

# Some GL Basics

Shared Memory (bandwidth considerations)

Shaders compiled at runtime



# Signatures

# Signatures

Vectorized

Sweep a texture across as a masking operation

```
// offset is a uniform we control from the CPU
// allowing us to "sweep"
gl_FragColor =  texture2D( Texture, TexCoordOut ) -
texture2D( Mask, TexCoordOut + offset );
```

# Signatures

## Offsets and other parameters controlled via uniforms

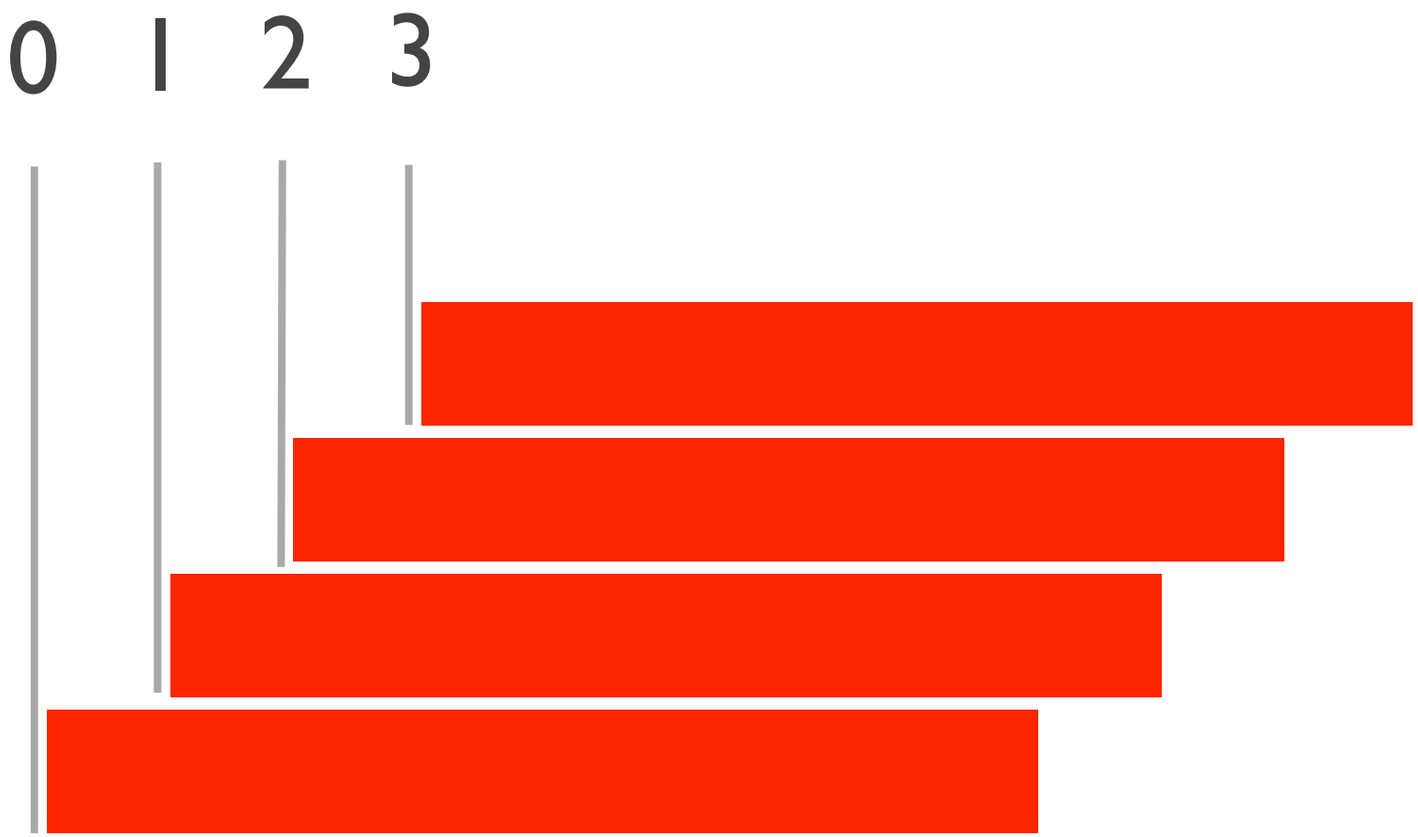
Periodic BC's

```
// offset is a uniform we control from the CPU
// allowing us to "sweep"
gl_FragColor =  texture2D( Texture, TexCoordOut ) -
texture2D( Mask, TexCoordOut + offset );
```

# Signatures in Action

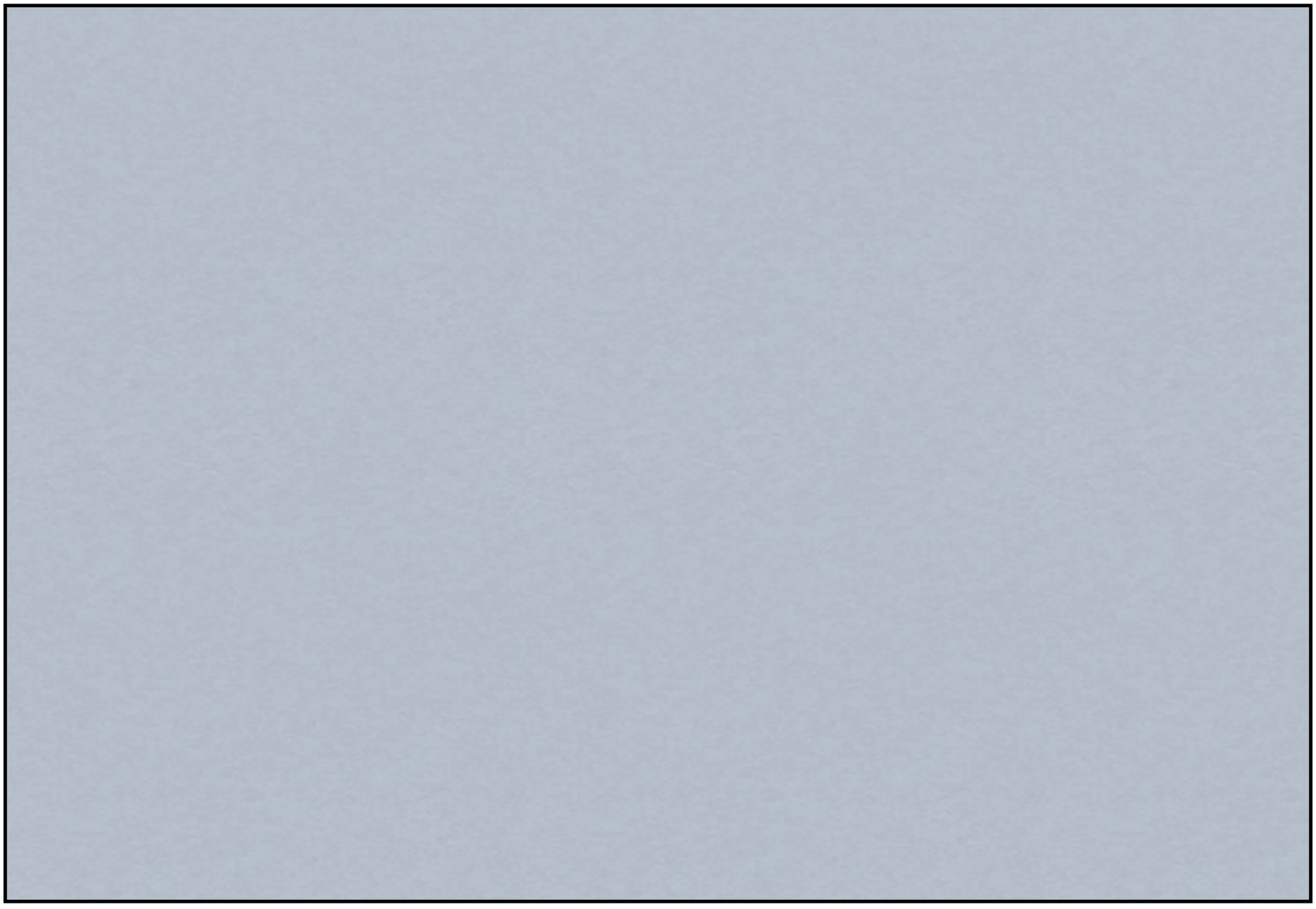
```
// placing breakpoint immediately after:
glReadPixels(0, 0, dimension, dimension, GL_RGBA, GL_UNSIGNED_BYTE, bytes);
//
// at the start... offset (0,0)
(gdb) x/20 bytes
0x6f33000:    0xff000000    0xffac420f    0xff002047    0xff000000
0x6f33010:    0xff090000    0xff5b1a00    0xff000000    0xff1c002c
0x6f33020:    0xff761900    0xff080900    0xff000001    0xff250f00
0x6f33030:    0xff000e00    0xff0d0013    0xff432500    0xff00005e
0x6f33040:    0xff1d8c1e    0xffc23646    0xffc36bb4    0xffbdc2b5

// later... offset(x,y)
(gdb) x/20 bytes
0x6f33000:    0xff070503    0xff000000    0xff000000    0xff000000
0x6f33010:    0xff000000    0xff000000    0xff000000    0xff000000
0x6f33020:    0xff000000    0xff000000    0xff000000    0xff000000
0x6f33030:    0xff000000    0xff000000    0xff000000    0xff000000
0x6f33040:    0xff000000    0xff520000    0xffc36bb4    0xffbdc2b5
```

# Byte Alignment

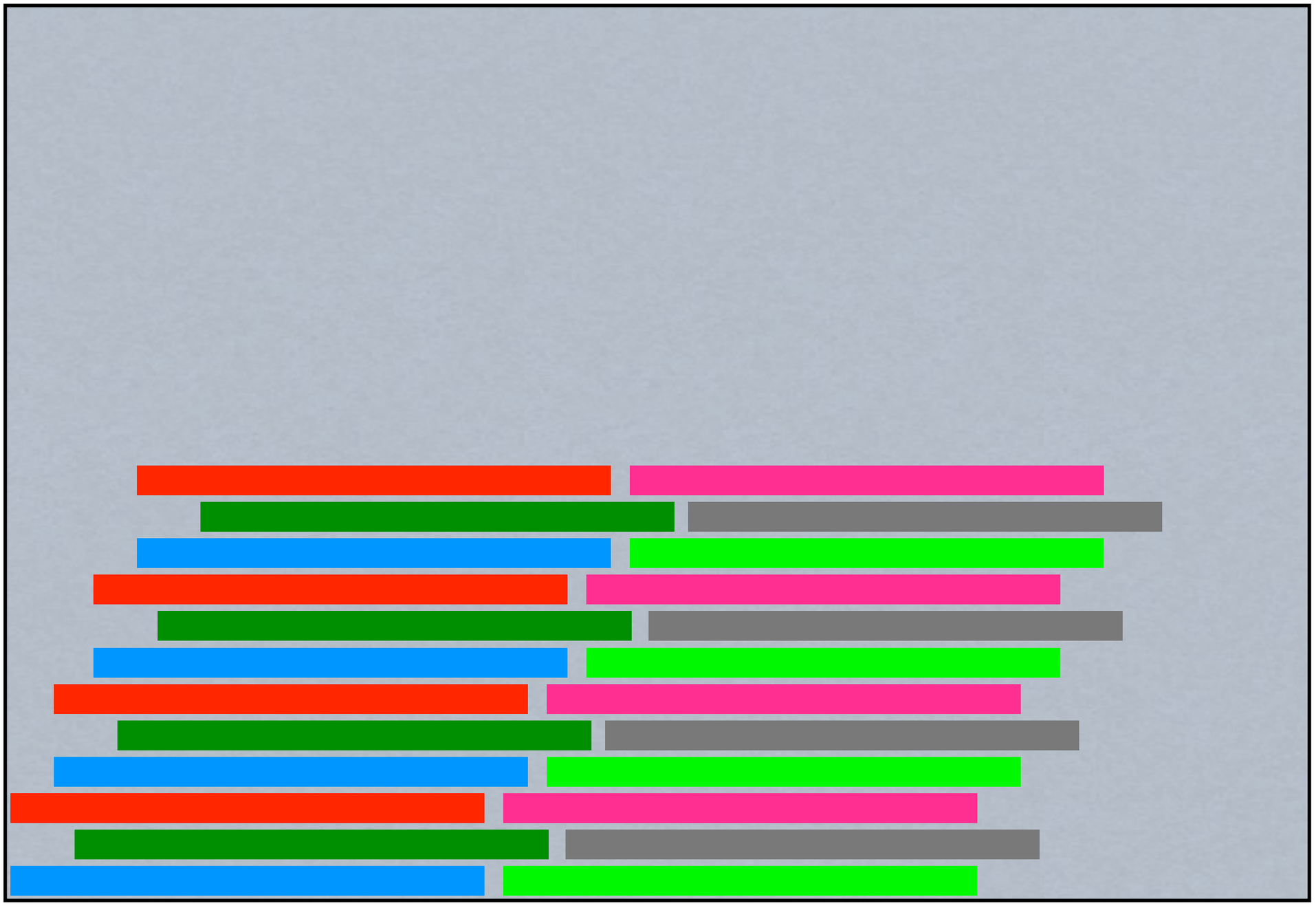



This solves the texel comparison problem, plus in practice we should fill our texture!

# Tiling



# Tiling



# Devices

iPhone 4S - 8 textures, 4096 max. texture dimension

NVIDIA Tegra 3 ASUS Prime Tablet 16 textures, 2048 max. texture dimension

NEON processors

PowerVR SGX543 vs NVIDIA Tegra 3 - NVIDIA Trailing but catching up

# Some Characteristics

Using OpenGL ES 2 exclusively

GL ES 1 lacks the control we want,
i.e. no shaders to compile!

Can use code optimized for NEON processor

EIGEN - Android

Accelerate (BLAS/LAPACK) iOS

However, this would be signed code...

# Memory Tracking

## A little setup goes along way

# Memory Tracking

```
class MemoryObject
{
    unsigned char leading[8];
    vector<string> objects;
    // other objects
    unsigned char trailing[8];


    // static methods
    static void Generator(unsigned char ptr[8]) {
        static unsigned char start = 0;
        const unsigned char interval = 0x02;
        start += interval;
        for (int i=0; i<8; i++)
            ptr[i] = start;

    }

public:

    // constructor
    MemoryObject(void) { Generator(leading); objects.push_back("Testing"); Generator(trailing);  }
```

A little setup goes along way

Can continuously monitor

Power consumption not really a problem

# Frag Shader

```
varying lowp vec4 DestinationColor;

varying lowp vec2 TexCoordOut;
uniform sampler2D Texture;
uniform sampler2D Mask;
uniform lowp vec2 offset;

// assume we have our signature in four bytes or less, we can grab neighboring texels
bool isValidSignature( lowp vec4 pixel )
{
    lowp float norm;
    norm = dot(pixel.rgb , pixel.rgb );
    bool result = false;
    if ( norm > 0.0 )
    {
        // for now we just see if the all RGB channels match, this means
        if ( pixel.r == pixel.g && pixel.g == pixel.b  )
        result = true;
    }
    return result;
}

// basic shader
void main(void) {

    if ( isValidSignature( texture2D(Texture,TexCoordOut) ) )
    {
        gl_FragColor    = vec4( 1, 0, 0, 1 );
    }
    else
    {
        gl_FragColor    = vec4( 0, 0, 0, 1 ) * texture2D( Texture,TexCoordOut);
    }

}
```

# In Practice: Render to Texture

```
- (BOOL)renderToTexture
{
    BOOL result = NO;
    unsigned char * texturedata = (unsigned char*) malloc( dimension * dimension * 4 );

    // bind our texture to render to...
    glBindFramebuffer(GL_FRAMEBUFFER, offScreenTexture);

    // gl calls, as before...
    ...

    //
    // set up our data to be sampled within the texture
    //
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, heapTexture);
    glUniform1i(sampler, 0);

    GetGLError();


    glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE, 0);

    // copy from the texture to image...
    glReadPixels(0, 0, dimension, dimension, GL_RGBA, GL_UNSIGNED_BYTE, texturedata );

    // analyze the results...
    if ( [self analyzeBuffer:texturedata] )
        result = YES;

    // unbind the frame buffer...
    glBindFramebuffer(GL_FRAMEBUFFER, frameBuffer);

    free( texturedata );

    return result;
}
```

# In Practice: Render to Texture

```
- (BOOL)renderToTexture
{
    BOOL result = NO;
    unsigned char * texturedata = (unsigned char*) malloc( dimension * dimension * 4 );

    // bind our texture to render to...
    glBindFramebuffer(GL_FRAMEBUFFER, offScreenTexture);

    // gl calls, as before...
    ...

    //
    // set up our data to be sampled within the texture
    //
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, heapTexture);
    glUniform1i(sampler, 0);

    GetGLError();


    glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE, 0);

    // copy from the texture to image...
    glReadPixels(0, 0, dimension, dimension, GL_RGBA, GL_UNSIGNED_BYTE, texturedata );

    // analyze the results...
    if ( [self analyzeBuffer:texturedata] )
        result = YES;

    // unbind the frame buffer...
    glBindFramebuffer(GL_FRAMEBUFFER, frameBuffer);

    free( texturedata );

    return result;
}
```

# In Practice: Render to Texture

```
- (BOOL)renderToTexture
{
    BOOL result = NO;
    unsigned char * texturedata = (unsigned char*) malloc( dimension * dimension * 4 );

    // bind our texture to render to...
    glBindFramebuffer(GL_FRAMEBUFFER, offScreenTexture);

    // gl calls, as before...
    ...

    //
    // set up our data to be sampled within the texture
    //
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, heapTexture);
    glUniform1i(sampler, 0);

    GetGLError();


    glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE, 0);

    // copy from the texture to image...
    glReadPixels(0, 0, dimension, dimension, GL_RGBA, GL_UNSIGNED_BYTE, texturedata );

    // analyze the results...
    if ( [self analyzeBuffer:texturedata] )
        result = YES;

    // unbind the frame buffer...
    glBindFramebuffer(GL_FRAMEBUFFER, frameBuffer);

    free( texturedata );

    return result;
}
```

# Disassembly

Can leverage vectorized NEON processor

Texture math, not that difficult

ARM = 32 bit insts = 4 channels = RGBA (it's fate)

What could you do?

# Dynamic DisASM

This work inspired by the thesis, "Approximate Disassembly using Dynamic Programming" by Shah

Basic Idea is to approximate disassembly by using optimization efforts.

Excellent candidate for GPU & Vectorized calls because of the mathematical formulation of the problem.

# Accelerate Framework

```
m = read_matrix_from_file( matrixfile );

__CLPK_integer mn = (__CLPK_integer) m->numberOpcodes;
float * vec, * result;
vec = (float*) malloc( sizeof(float) * m->numberOpcodes );
result=(float*)malloc( sizeof(float) * m->numberOpcodes );

// populate the vector...
populate_vector( vec, m->numberOpcodes, m->opcodes, argv[1] );

cblas_sgemv( CblasColMajor, CblasNoTrans, mn, mn, alpha, m->elements,
        mn, vec, 1, beta, result, 1 );

char * chosen_opcode = choose_opcode( result, m->numberOpcodes,
                                      m->opcodes );
printf("%s\n",chosen_opcode);
// print_vector( result, m->numberOpcodes );

free( vec );
free( result );
destroy_matrix( m );
return 0;
```

# Accelerate Framework

iOS's current answer to OpenCL

Can leverage vectorized algorithms - BLAS/LAPACK

Excellent for image, vector, signal processing

# Exploring DisASM

```
# let's use our C-code to on-the-fly generate
# disassembly using probability tables
last_instruction = False
application = './optimization'
args = ''
digraph = 'simple.digraph'
recreated = []
index = 0
for instr in samples:
    if ( instr == 'xxx' ):
        # call out...
        try:
            prob_instruction = subprocess.check_output([application,last_instruction,digraph])
            instr = prob_instruction
            print "replacing %s with %s " % (instructions[index],instr)
        except:
            print "Error in our C-code, time to debug..."
    # append...
    recreated.append(instr)
    last_instruction = instr
    index+=1

# Now we test our disassembly to see how much we got right...
```

# Exploring DisASM

## Simulating in Python

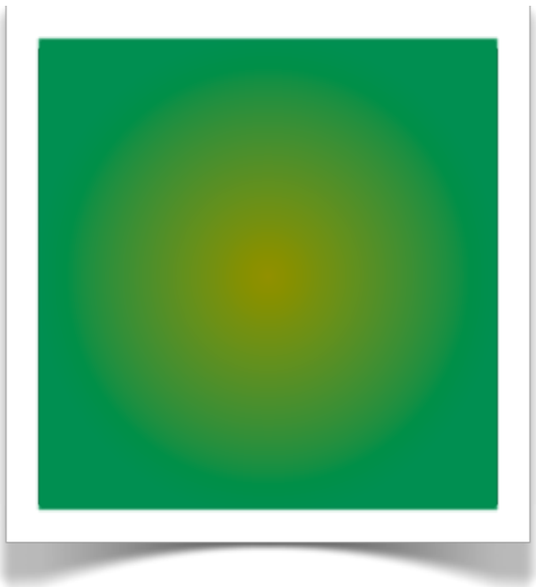
```
# let's use our C-code to on-the-fly generate
# disassembly using probability tables
last_instruction = False
application = './optimization'
args = ''
digraph = 'simple.digraph'
recreated = []
index = 0
for instr in samples:
    if ( instr == 'xxx' ):
        # call out...
        try:
            prob_instruction = subprocess.check_output([application,last_instruction,digraph])
            instr = prob_instruction
            print "replacing %s with %s " % (instructions[index],instr)
        except:
            print "Error in our C-code, time to debug..."
    # append...
    recreated.append(instr)
    last_instruction = instr
    index+=1

# Now we test our disassembly to see how much we got right...
```

# Encryption

Render to Texture A

Just another render call

Operate on Texture A



Render to Texture B

This is not atypical consumption, just an atypical usage - TinyWings

# Be Creative

A simple example...

If you understand how to work with the "signed" API calls then you can alter your shader(s).

Therefore, OTA updates for encryption algorithms? Yes...

Bottom line - you have adaptive computational resources at your disposal!

# GPU Malware

First paper used GPU decryption as a method to deliver the malicious payload

As GPGPU (general) methods become increasingly available this is likely to increase


# A Demo

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.

    unsigned int i, j;
    void * heap_space = (void*) malloc( sizeof(Matrix) * 3 );
    A = new Matrix(3,3);
    B = new Matrix(3,3);
    C = new Matrix(3,3);

    ....
```

```
- (IBAction)corruptMemory:(id)sender
{
    NSLog(@"Corrupting Memory");
    // grab memory from C and corrupt it...
    unsigned int i,s = (unsigned int) sizeof(Matrix);
    unsigned char * p = (unsigned char *) C;
    for (i=0; i<s; i++) {
        p[i] = i;
    }

    ...
```

```
...
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE, 0);

// copy from the texture to raw...
glReadPixels(0, 0, dimension, dimension, GL_RGBA, GL_UNSIGNED_BYTE, texturedata );

// analyze the results...
if ( [self analyzeBuffer:texturedata] )
    result = YES;

// unbind the frame buffer...
glBindFramebuffer(GL_FRAMEBUFFER, frameBuffer);

return result;
```

# Renderscript

Mechanism to leverage all resources on device.

Compiles C99 code to shaders, Java classes, etc.

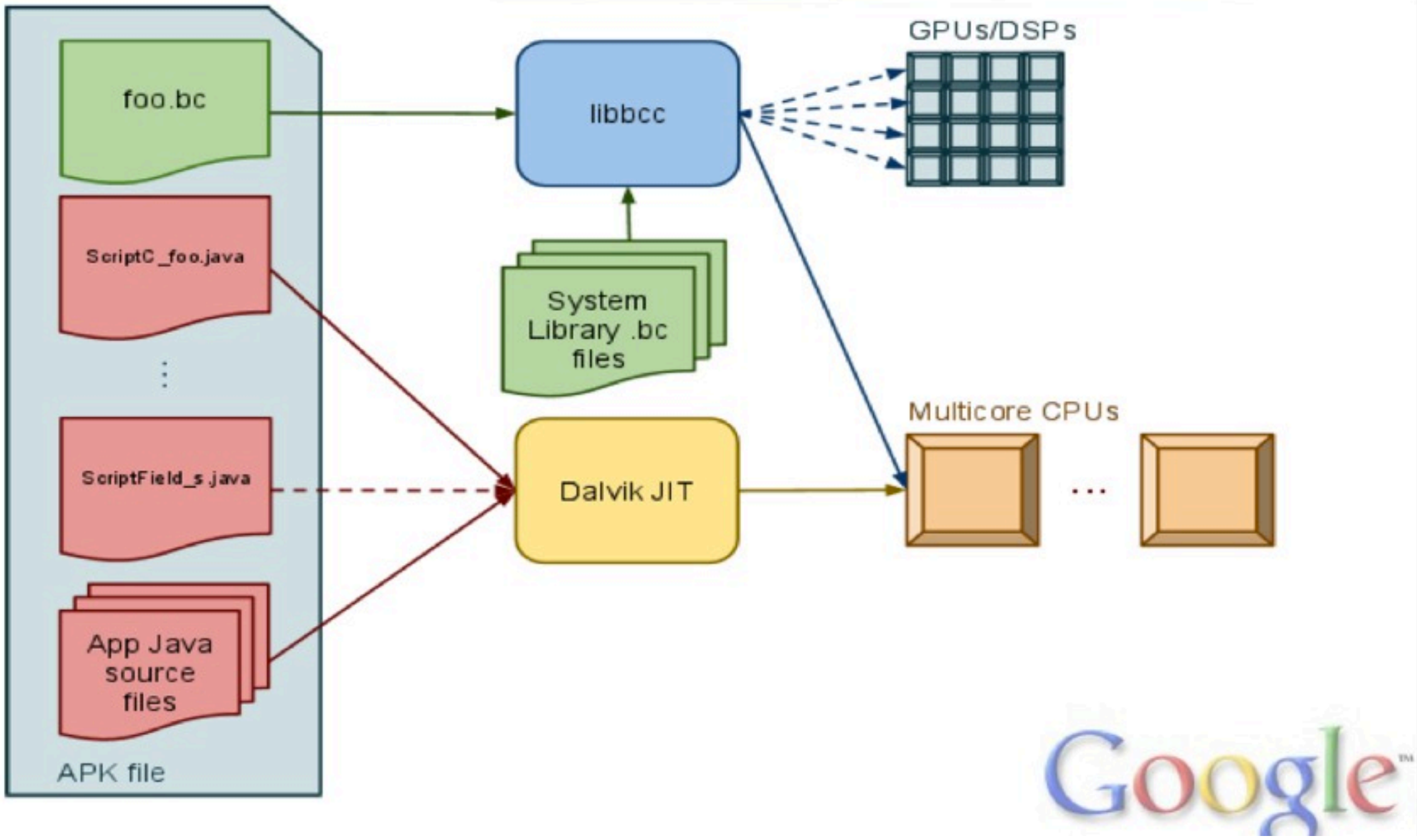
Interesting caching mechanism

No control

Shaders are... so-so...

Allows device considerations to be offloaded

# Online JIT Compiler Flow



# Use Cases

How can you use this today?

Limiting factors today for strict GPU usage is the number of textures

Mathematical, especially vectorized, techniques are best suited for more complex tasks

# Uses for Tomorrow

We've shown a variety of ways, but keep in mind that this is a rapidly growing area

Generalized techniques are clearly coming

Device access and memory architectures for Mobile an area to watch.

# Tools & Frameworks

# Tools & Frameworks

OpenGL ES 2

GLKit

Renderscript

GL Extensions

OpenCL

CUDA

# CARMA

## Ubuntu Linux ARM board with CUDA

**CPU**
**NVIDIA® Tegra® 3 ARM Cortex A9 Quad-Core**
**GPU**
**NVIDIA® Quadro™ 1000M with 96 CUDA® Cores**
**Memory**

1. **CPU Memory: 2 GB**
2. **GPU Memory: 2 GB**

**Peak Performance**
**270 Single Precision GFlops**
**CPU - GPU Interface**
**PCIe x4 Gen1 link**
**Network**
**1x Gigabit Ethernet**
**Storage**
**1x SATA Connector**
**USB**
**3x USB 2.0**
**Display**
**HDMI**
**Software**

1. **Linux Ubuntu Derivative OS**
2. **CUDA® Tool Kit**

# Follow the $$$

Economics

Attacker Math?

## Source

http://www.appleinsider.com/articles/11/11/09/apples_iosgoogle_android_command_58_of_us_portable_game_revenue.html

# Follow the $$$

Economics

Attacker Math?

Security

Games

## Source

http://www.appleinsider.com/articles/11/11/09/apples_iosgoogle_android_command_58_of_us_portable_game_revenue.html

# Follow the $$$

Economics

Attacker Math?

Security Games

Apple & Google command 58 % of portable games in US

Approx 3.5+ Billion $$

Source

http://www.appleinsider.com/articles/11/11/09/apples_iosgoogle_android_command_58_of_us_portable_game_revenue.html

# What have you learned?

GPU/Vectorized processors are ready today.

Shaders allow you a way to deliver unsigned code, OTA, across platforms

GPUs will be used as part of cyber for tomorrow

# What Can You do?

Download some source

source located at github: https://github.com/jcarlson23/gpumalware

Ask Questions

jared.carlson23@gmail.com

# Thanks

**DARPA** - @mudge, Peiter Zaitko

**MITRE** - Seth Landsman, Alan Stone, Rob Dingwell, Nick Harezga, and Ayal Spitz

**VSR** - George Gal and Dan Rosenberg

**viaForensics** - Andrew Hoog and Thomas Cannon

# Questions?