

THE SALAMI METHOD

ADI SHAVIT

X-PLATFORM

- ASIC
- EMBEDDED
- MOBILE: iOS, ANDROID
- DESKTOP: WINDOWS, LINUX, OSX
- BROWSER: PLUGINS, NaCL, JS, WASM
- CLOUD: SERVERS, DOCKER, Node.js, Serverless
- ...AND MORE:
 - VEHICLES, IoT, ROBOTS, CPU/GPU

C and C++ are probably the only viable languages for true cross-platform development.

UNFORTUNATELY...



POPULAR PASTIME: CODE PASTA

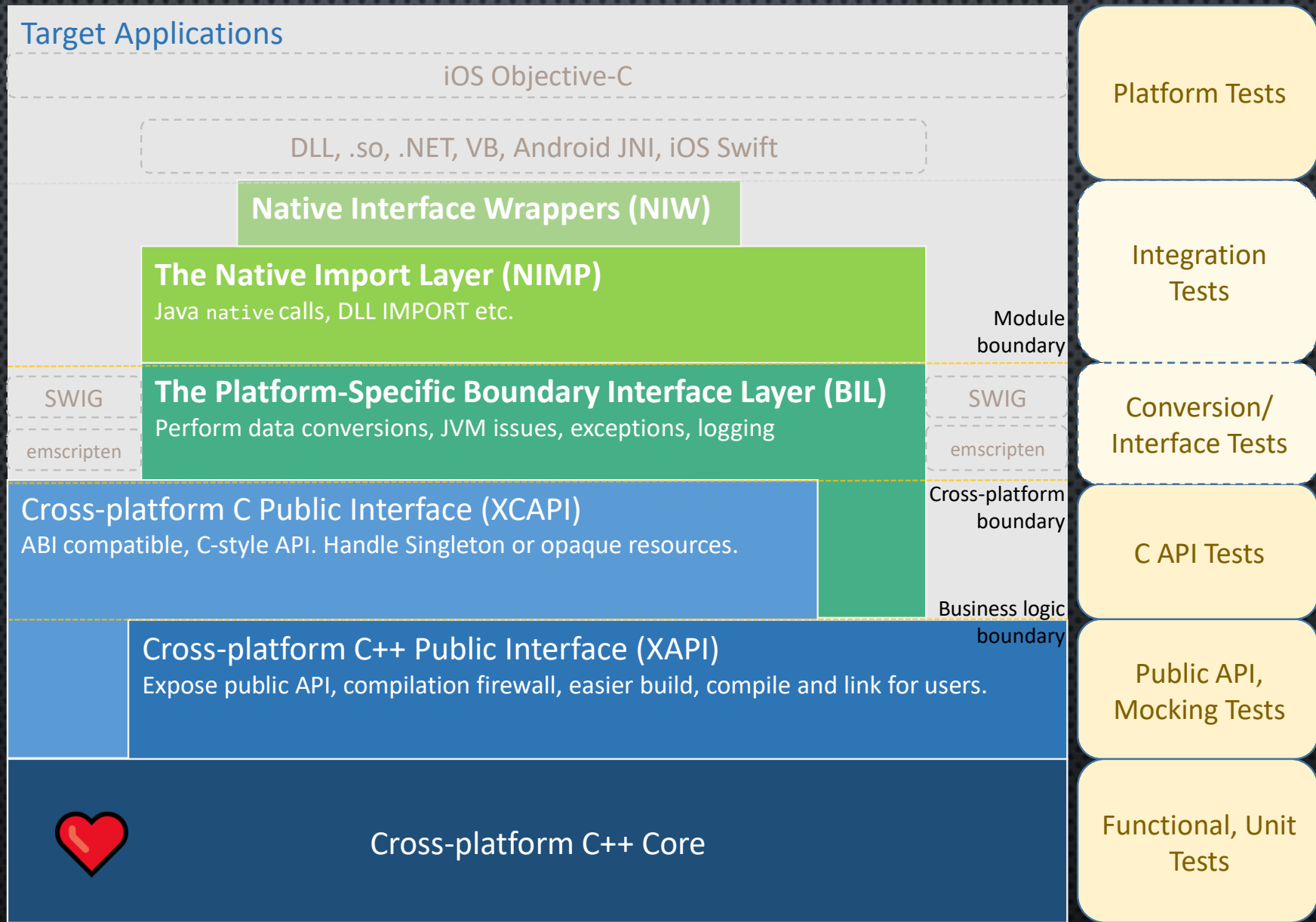
- REFACTORING AND MAINTENANCE NIGHTMARE
- NON-PORTABLE
- UNTESTABLE
- BUG-PRONE
- MIXING OF SKILLS
- MODULE BOUNDARIES (DLL, SO APIs):
 - ERROR HANDLING
 - EXCEPTIONS: MAYBE TERMINATION OR UNDEFINED-BEHAVIOR



THE SALAMI METHOD

- BORN OF FRUSTRATION, BLOOD, SWEAT AND TEARS
- A THIN, TRANSPARENT LAYER FOR EACH ASPECT
- MORE EASILY:
 - BUILD
 - TEST
 - DEBUG
 - MANAGE
 - MAINTAIN





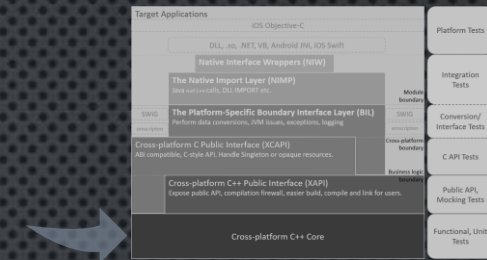
SLICING BENEFITS

- THE DRY PRINCIPLE: AVOIDS DUPLICATION AND REIMPLEMENTATION
- SINGLE RESPONSIBILITY AND TESTABILITY == TRANSPARENCY
- CONSISTENCY : BUSINESS LOGIC IS *ISOLATED* AND *SHARED*
- NEW PLATFORM READY
- DEVELOPER SKILLS
- REFACTORING

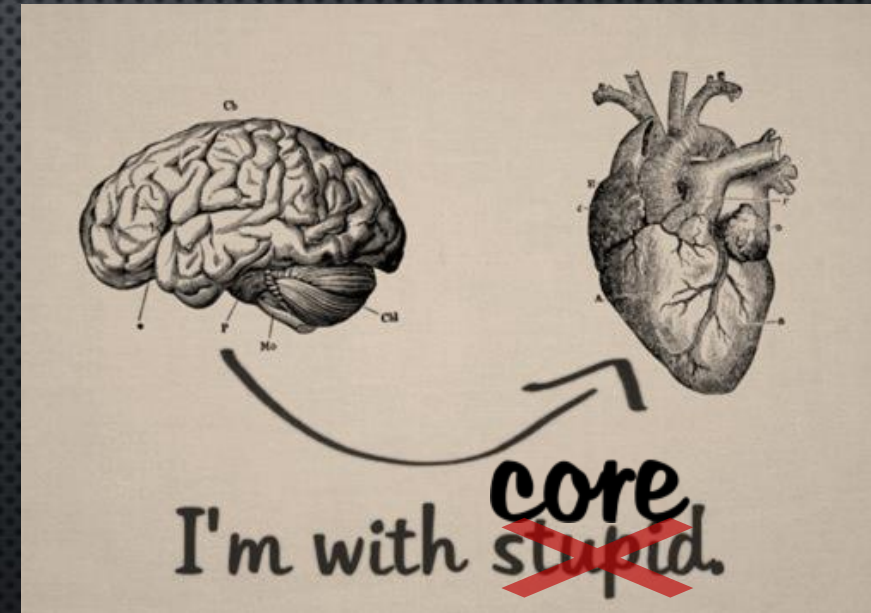


Like all good architectures, the Salami Method tries to cleanly separate concerns. Nevertheless, it lacks the greasy, heart-attack-inducing goodness of a good salami.

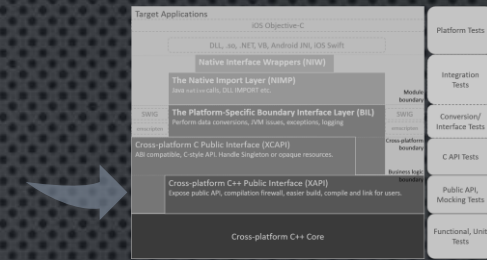
CROSS PLATFORM C++ CORE



- THE CORE
- THE BUSINESS LOGIC
- PROPRIETARY CODE, IP
- IDIOMATIC C++ CODE
- X-PLATFORM BUILD
- STATIC LIBRARY
- TESTING: UNIT, FUNCTIONAL



XAPI: X-PLATFORM PUBLIC C++ INTERFACE



- THE *PUBLIC* C++ CORE API
- APPLY GOOD API DESIGN PRINCIPLES – FOCUS ON THE SERVICE CONSUMERS
- CONSIDER:
 - INITIALIZATION AND SHUTDOWN; LIFETIME MANAGEMENT; SESSIONS; CONFIGURATION; SERIALIZATION ETC.
- HIDES PROPRIETARY CODE/DEPENDENCIES, REDUCE DEPENDENCIES, PIMPL IDIOM
- C++ MODULES EXPORT POSTERCHILD (C++20)
- TESTING: MOCK AND UNIT TEST THE API
- NAMING CONVENTION:
 - WITH CORE FILE: `core.cpp` → CORRESPONDING: `core_api.cpp`
- MAY SKIP FOR SMALL PROJECTS



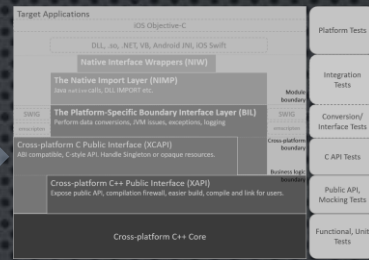
WARNING

WARNING

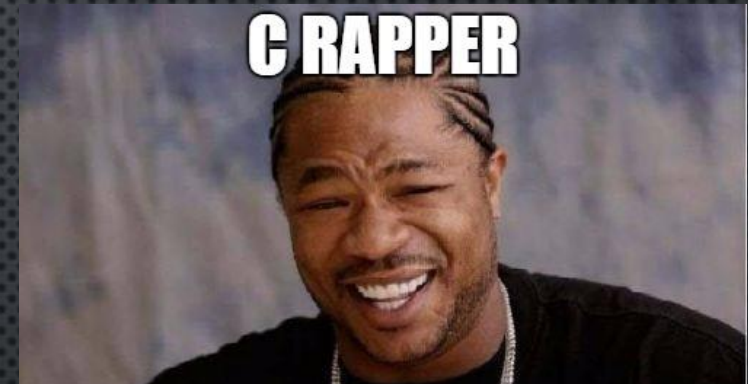
WARNING

**No Business
Logic Beyond
this Point !!!**

XCAPI: THE X-PLATFORM PUBLIC C INTERFACE



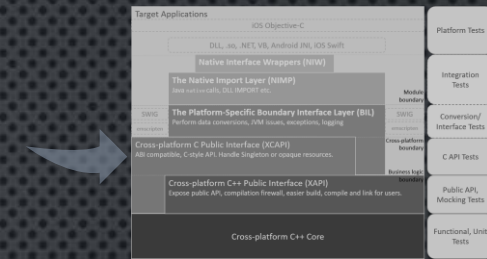
- THE TRUE *LINGUA-FRANCA* OF COMPUTING IS C: SIMPLE ABI
- A *THIN C API WRAPPER* OVER XAPI:
 - extern "C" FOR LINKING ABI
 - PORTABLE TYPE CONVERSIONS: E.G. `std::string` → `char*`
 - C++ OBJECT LIFETIME MANAGEMENT VIA C INTERFACE:
 - GLOBAL / SINGLETON
 - OPAQUE HANDLES
 - OVERLOADED C++ FUNCTIONS → MULTIPLE C FUNCTIONS
- TESTING: MOCK AND UNIT TEST THE C API/SDK
- NAMING CONVENTION:
 - WITH XAPI FILE: `core_api.h` → CORRESPONDING: `core_c_api.cpp`



YO DAWG

In wrapper implementation, use type helpers like `gsl::span` or `std::string_view` to lift weaker low-level C types to stronger, safer types as early as possible!

XCAPI



```
// foo_session_c_api.h
extern "C"
{
    bool initFromFileName(std::string const& fileName);
    bool initFromCount(int count);
    bool processBuffer(uint8_t* buffer, int size);
    bool isReady();
}
```

"C" will sometimes mean a *C-like* stand-alone function interface
(i.e. not necessarily pure C)

XCAPI

```
// foo_session_c_api.cpp
#include <foo_session_api.hpp> // C++ API
#include "foo_session_c_api.h" // header for this file

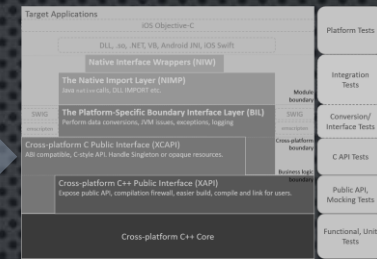
Foo::Session the_session; // use a single global "singleton".

bool initFromFileName(std::string const& fileName)
{ return the_session.init(fileName); }

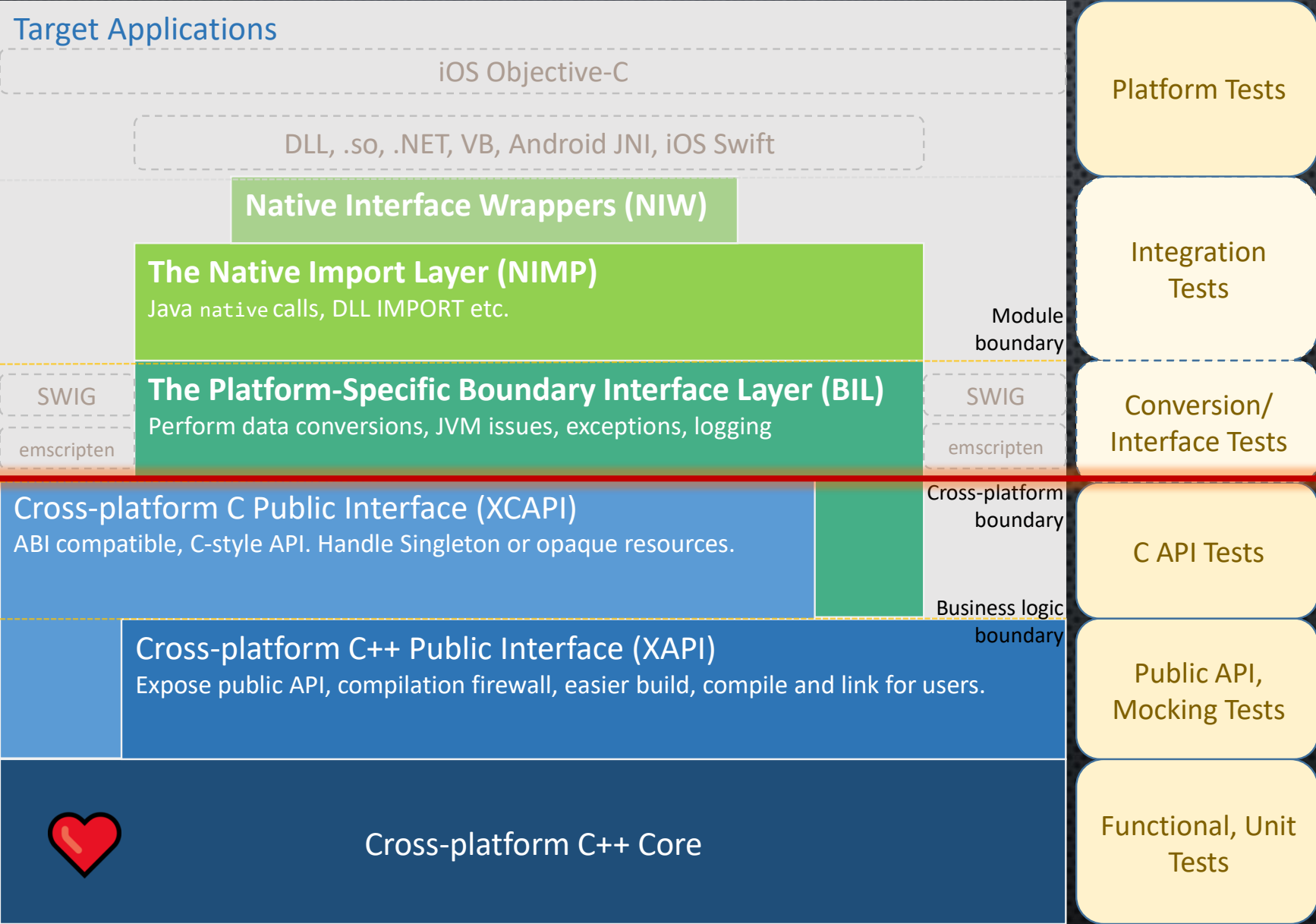
bool initFromCount(int count)
{ return the_session.init(count); }

bool processBuffer(uint8_t* buffer, int size)
{ return the_session.process(gsl::span<uint8_t>(buffer, buffer+size)); }
// Use gsl::span<> -----^^^^^^^^^^^^^^^^^^^^^^

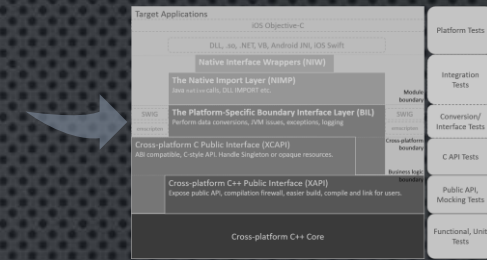
bool isReady()
{ return the_session.isReady() }
```



C-Rapper says:
If you have multiple ctors and no `init()` you could use e.g. `std::optional` or `unique_ptr<>`.



BIL: PLATFORM BOUNDARY INTERFACE LAYER

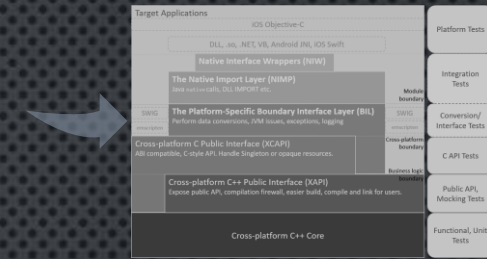


- PLATFORM SPECIFIC CODE!
- IMPLEMENTED PER TARGET PLATFORM:
 - CONVERSIONS: TYPES AND VALUES
 - CONSTRAINTS
 - CONVENTIONS
 - SET UP PLATFORM-SPECIFIC LOGGING LOGIC
- MODULE BOUNDARY: DLL, .SO
 - NO EXCEPTIONS CAN ESCAPE!
- TESTING: ? LMK!
- NAMING CONVENTION:

XC-API FILE: `core_c_api.h` → `core_c_api_dll.cpp`, `core_c_api_jni.cpp`

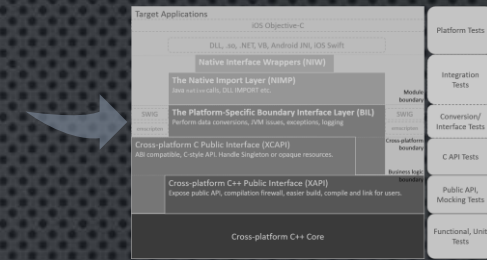


BIL – DLL/SO EXAMPLE



```
// foo_session_c_api_dll.h
extern "C"
{
    bool DLL_EXPORT FooSession_initFromFileName(LPCSTR fileName);
    bool DLL_EXPORT FooSession_initFromCount(int count);
    bool DLL_EXPORT FooSession_processBuffer(unsigned char* buffer, int size);
    bool DLL_EXPORT FooSession_isReady();
}
```


BIL – DLL EXAMPLE



```
// foo_session_c_api_dll.cpp
#include <foo_session_c_api.h> // C API
#include "foo_session_c_api_dll.h" // header for this file

bool DLL_EXPORT FooSession_initFromFileName(LPCSTR fileName) try
{ return ::initFromFileName(fileName); } // automatic LPCSTR conversion to std::string
catch (...) { return false; }

bool DLL_EXPORT FooSession_initFromCount(int count) try
{ return ::initFromCount(count); }
catch (...) { return false; }

>bool DLL_EXPORT FooSession_processBuffer(uint8_t* buffer, int size) try
{ return ::processBuffer(buffer, size); }
catch (...) { return false; }

bool DLL_EXPORT FooSession_isReady() try
{ return ::isReady(); }
catch (...) { return false; }
```

So Thin!

BIL – ANDROID JNI EXAMPLE

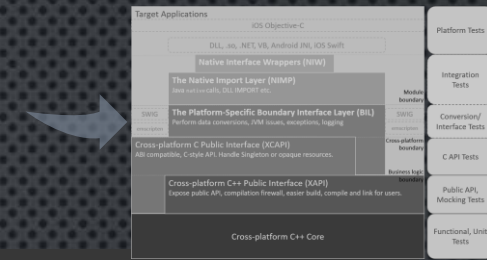
```
#include <jni.h> // JNI headers
#include <android/log.h> // Android logging facilities
#include <foo_session_c_api.h> // C API
#include "jni_utils.h" // For JNIByteArrayAdapter and exceptionHandler
```

```
JNIEXPORT jboolean JNICALL Java_initFromFileName(JNIEnv* env, jobject thiz, jstring fileName) try
{ return ::initFromFileName(jni_utils::getString(env, fileName)); } // JNI string helper
catch(...) { return exceptionHandler(); }
```

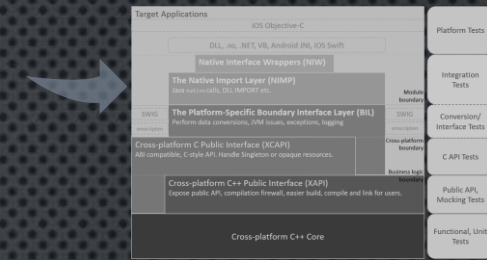
```
JNIEXPORT jboolean JNICALL Java_initFromCount(JNIEnv* env, jobject thiz, jint count) try
{ return ::initFromCount(count); }
catch(...) { return exceptionHandler(); }
```

```
JNIEXPORT jboolean JNICALL Java_processBuffer(JNIEnv* env, jobject thiz, jbyteArray buffer) try
{
    jni_utils::JNIByteArrayAdapter buffer_span(env, buffer); // JNI helper wrapper
    return ::processBuffer(buffer_span.ptr(), buffer_span.size());
}
catch(...) { return exceptionHandler(); }
```

```
JNIEXPORT jboolean JNICALL Java_isReady(JNIEnv* env, jobject thiz) try
{ return ::isReady() }
catch(...) { return exceptionHandler(); }
```



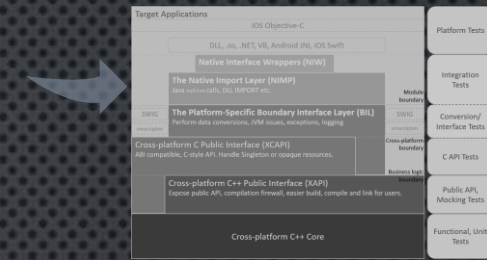
NIMP: NATIVE IMPORT LAYER



- ON TARGET PLATFORM:
 - DEVICE, HW, OS, LANGUAGE
- USES “NATIVE-NATIVE” INTERFACE
 - REFLECTS BIL
 - USUALLY LOW LEVEL
- TESTING: TARGET INTEGRATION TESTS
- NAMING CONVENTION:
 - core_native.java, core_native_dll_wrapper.cs , core_native.js



NIMP

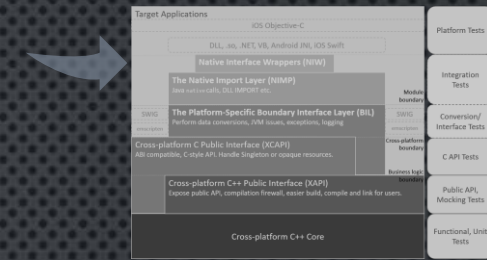


```
// foo_session_native.java
// imports ...
public class FooSession
{
    static { System.loadLibrary("native_foosession"); } // load the DLL

    public static native boolean initFromFileName(String fileName);
    public static native boolean initFromCount(int count);
    public static native boolean processBuffer(byte[] buffer);
    public static native boolean isReady();
}
```

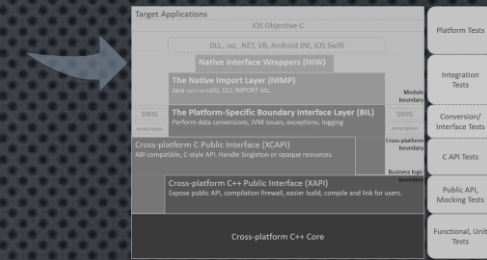

NIW: INTERFACE WRAPPERS

- HIGH LEVEL NIMP WRAPPERS
- MORE NATURAL, FAMILIAR SYNTAX
- HIGHER LEVEL TYPES
 - DEVICE, HW, OS, LANGUAGE



- TESTING: TARGET INTEGRATION TESTS
- NAMING CONVENTION: DEPENDS ON TARGET AND CONTEXT

NIW

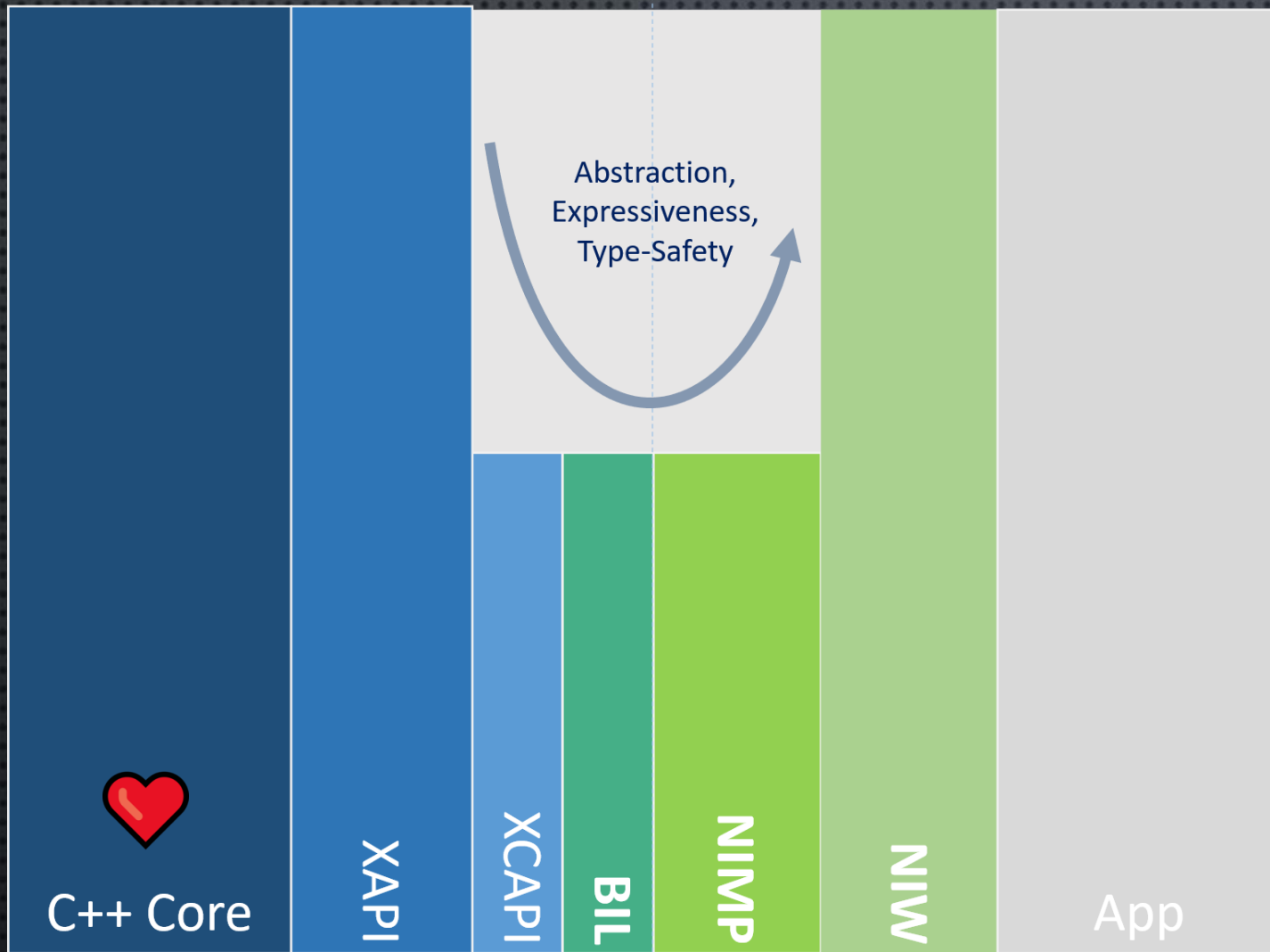


```
// face_detector_native.java
import android.graphics.PointF; // Android point type

public class FaceDetector
{
    static { System.loadLibrary("native_facedetector"); } // load the DLL

    // native import function/method, returns a float array
    public static native float[] getFaceCenterPoint();

    // Java-ized wrapper: return proper 2D point type
    public static PointF GetFaceCenterPoint()
    {
        float[] centerPt = getFaceCenterPoint(); // call native function
        return new PointF(centerPt[0], centerPt[1]); // return as Android Java type: PointF
    }
}
```

BEAUTIFUL SYMMETRY





QUICK EXAMPLE



(time allowing)

github.com/adishavit/party_parrot

XAPI / XCAPI color_cycle.h

```
namespace color_cycle
{
    void rotate_hue(cv::Mat3b const& img, cv::Mat3b& result_img, int hsteps);
    void clear_all();
}
```

BIL color_cycle_js.cpp

```
#include <emscripten.h>
cv::Mat3b bgr_g, bgr_out_g; // global data
extern "C"
{
    bool EMSCRIPTEN_KEEPALIVE rotate_colors(int width, int height, cv::Vec4b* frame4b_ptr, cv::Vec4b* frame4b_ptr_out,
                                             int hsteps) try
    {
        cv::Mat4b rgba_in(height, width, frame4b_ptr); // wrap memory pointers with proper
        cv::Mat4b rgba_out(height, width, frame4b_ptr_out); // cv::Mat images (no copies)

        bgr_g.create(rgba_in.size()); // allocate 3-channel images if needed
        bgr_out_g.create(rgba_in.size());

        cv::cvtColor(rgba_in, bgr_g, CV_RGBA2BGR); // rearrange channels and drop alpha channel

        color_cycle::rotate_hue(bgr_g, bgr_out_g, hsteps); // do the actual work!!

        // mix BGR + A (from input) => RGBA output
        const Mat in_mats[] = { bgr_out_g, rgba_in };
        constexpr int from_to[] = { 0,2, 1,1, 2,0, 6,3 };
        mixChannels(in_mats, std::size(in_mats), &rgba_out, 1, from_to, std::size(from_to)/2);
        return true;
    }
    catch (std::exception const& e) // ...
}
```


NIMP color_cycle.js

```
// Compute and display the next frame
fp.renderFrame = function () {
  // Acquire a video frame from the video element
  fp.ctx.drawImage(fp.video, 0, 0, fp.video.videoWidth,
    fp.video.videoHeight, 0, 0, fp.width, fp.height);
  var img_data = fp.ctx.getImageData(0, 0, fp.width, fp.height);

  // allocate Emscripten Heap memory buffer only when needed:
  if (!fp.frame_bytes) {
    fp.frame_bytes = _arrayToHeap(img_data.data);
  }
  else if (fp.frame_bytes.length !== img_data.data.length) {
    _freeArray(fp.frame_bytes); // free heap memory
    fp.frame_bytes = _arrayToHeap(img_data.data);
  }
  else {
    fp.frame_bytes.set(img_data.data);
  }

  // Perform operation on copy, no additional conversions needed, direct pointer manipulation
  // results will be put directly into the output param.
  Module._rotate_colors(img_data.width, img_data.height,
    fp.frame_bytes.byteOffset, fp.frame_bytes.byteOffset,
    fp.color_change_speed);

  // copy output to ImageData
  img_data.data.set(fp.frame_bytes);
  // Render to viewport
  fp.viewport.putImageData(img_data, 0, 0);
};
```

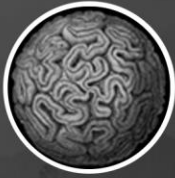
```
// Given a JS TypedArray, Module._malloc() a buffer of the same size
function _arrayToHeap(typedArray) {
  var numBytes = typedArray.length * typedArray.BYTES_PER_ELEMENT;
  var ptr = Module._malloc(numBytes);
  heapBytes = Module.HEAPU8.subarray(ptr, ptr + numBytes);
  heapBytes.set(typedArray);
  return heapBytes;
}

// Free the malloced data. No GC works on this heap.
// Alas, no dtors in JS either :(
function _freeArray(heapBytes) {
  Module._free(heapBytes.byteOffset);
}
```



THE HOST APP index.html

```
<div id="video_place"></div>
<script src='color_cycle_asm.js'></script>
<script src='color_cycle.js'></script>
<script>
    var fp = makeFrameProcessor("sirocco.mp4");
    function updateColorChangeSpeed(newValue) { fp.color_change_speed = newValue; }
</script>
<input type="range" min="0" max="20" value="1"
oninput="updateColorChangeSpeed(this.value)"
onchange="updateColorChangeSpeed(this.value)"/>
```

Video Cortex

Insights and Visions

Blog



A now, without further ado...

The Party Parrot App

And here's the app in all its glory (I literally pasted the code above into the post's Markdown):



On the left, the original, a regular HTML5 video.

On the right, the same video being processed live, frame-by-frame cycling of the frame's hue channel.

This is a live, real time view, running in the browser!

Use the slider to change the color cycling speed.

<http://videocortex.io/2017/opencv-web-app>

Thank
You



videocortex.io



[@AdiShavit](https://twitter.com/AdiShavit)

