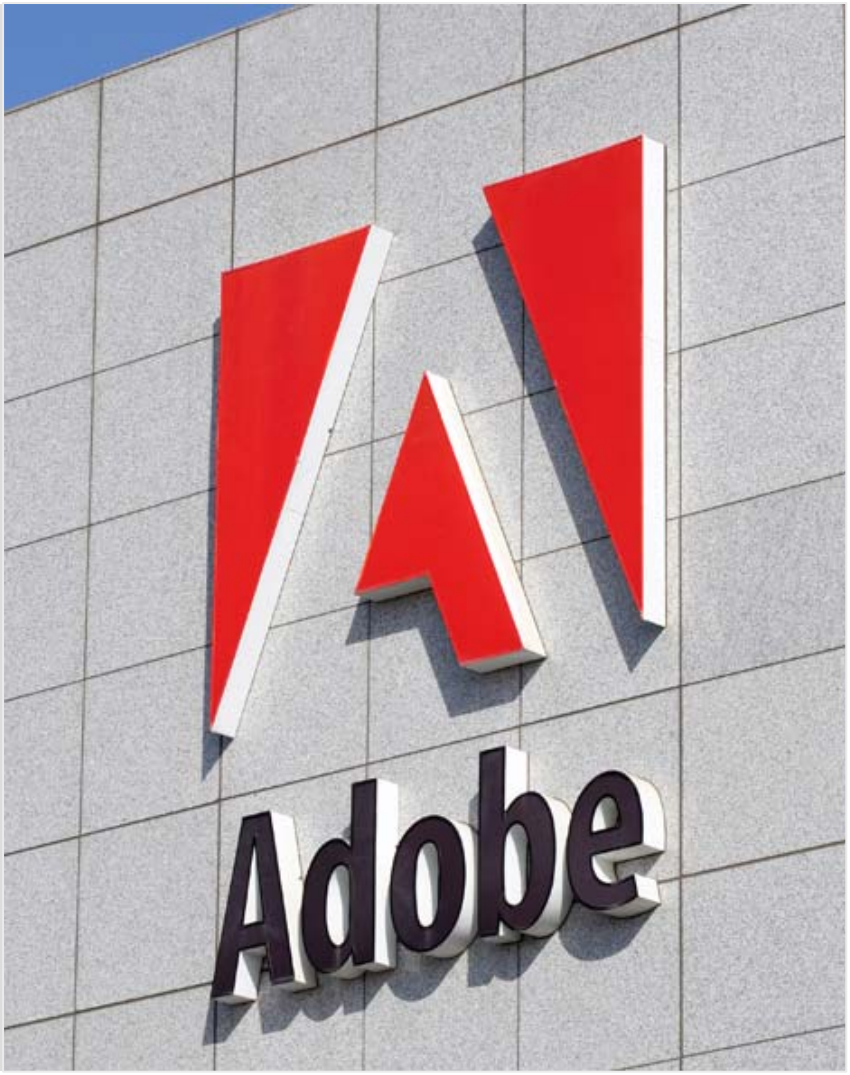
FlaCC

8/1/2008

Scott Petersen



Adobe

What is FlaCC?

- **F**lash **C** **C**ompiler
- Research project
  - No release schedule
- C/C++ in, SWF out
  - C/C++ routines
  - C/C++ libraries
  - C/C++ apps

Background

- Flash Player – Adobe's ubiquitous graphical platform
  - Primarily web centric
- AIR – Adobe's desktop development platform based on Flash technology
  - Desktop centric – provides file system access, other privileged operations
- SWF – Flash file format, AIR content type
- SWC – Flash/AIR library format
- ActionScript 3 (AS3) – Flash/AIR primary programming language
  - Similar to JS plus classes (or ES 3 minus eval)
  - Optionally strongly typed, compiles to bytecode
- AVM2/AVMPlus/Tamarin Central – Runs ABC
  - **A**ctionScript **V**irtual **M**achine
  - Interpreter + JIT compiler executes **A**ctionScript **B**yte**C**ode
- ASC – **A**ction**S**cript **C**ompiler

Motivation

- Flash/AIR developers want to reuse code too!
- C developers take thorough lib support for granted
- C developers can easily move code to most platforms
  - Why not Adobe's platform too?

Brief evolutionary overview

- Prototype 1: C => C via IR
  - MSILWriter based C FSM emitter
  - Linked to live C libs
- Prototype 2: C => AS3 via IR
  - MSILWriter based ActionScript FSM emitter
  - Handwritten subset of C standard libs, SDL
- Current: C => AS3 via 'real' codegen
  - Sparc backend derived ActionScript backend
  - C standard lib ported using tools
  - Low level system services hand-written in AS3

Implementation

- “march=avm2”
  - ~3700 lines of CPP
  - ~1025 lines of TableDesc

Machine Model

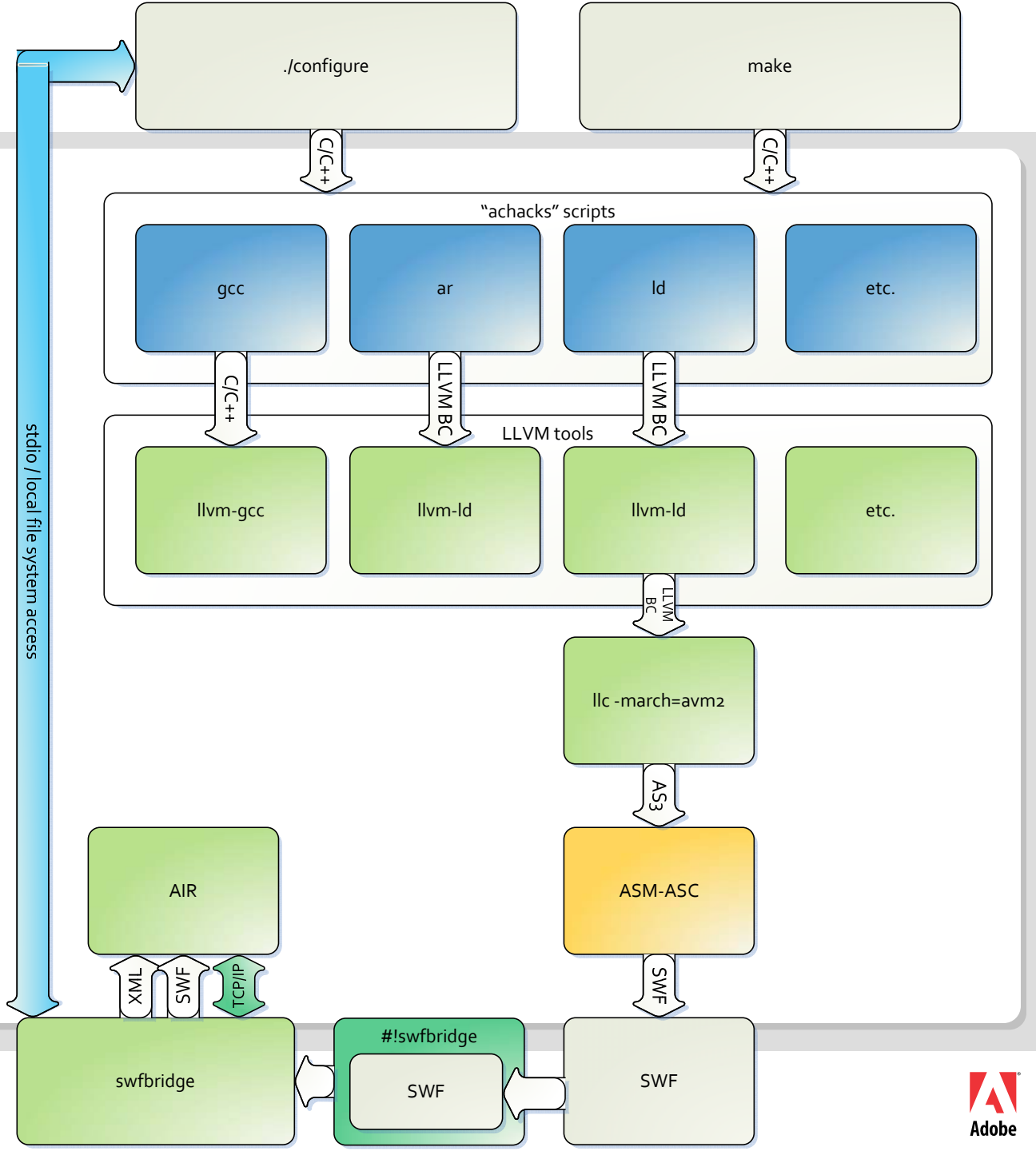
- Partial x86 semantics
- Registers
  - x86 subset: ebp, esp, eax, edx, cf(eflags), st0 **(global)**
  - 32 gp regs (int32) **(local/member)**
  - 32 fp regs (double) **(local/member)**
  - 32 fake single fp regs (double) **(local/member)**
- x86-32 param passing via virtual stack
- x86-32 return values via virtual eax, eax+edx, st0
- “ram” is a monolithic ByteArray
  - New AVM2 Ops make this fast

Example generated AS3

```
    __asm(lbl("___vfprintf_state0"))
    __asm(lbl("___vfprintf__XprivateX__BB79_0_F"))
        mstate.esp -= 4; __asm(push(mstate.ebp), push(mstate.esp),
op(0x3c), stack(-2))
        mstate.ebp = mstate.esp
        mstate.esp -= 2640
        i0 =  (0)
        i1 =  ((__xasm<int>(push((mstate.ebp+16)), op(0x37))))
        __asm(push(i1), push((mstate.ebp+-84)), op(0x3c), stack(-2))
        __asm(push(i0), push((mstate.ebp+-86)), op(0x3a), stack(-2))
        i0 =  ((__xasm<int>(push((mstate.ebp+8)), op(0x37))))
        i1 =  ((__xasm<int>(push((mstate.ebp+12)), op(0x37))))
        __asm(push(i1), push((mstate.ebp+-2295)), op(0x3c), stack(-2))
        i1 =  ((__xasm<int>(push(___mlocale_changed_2E_b), op(0x35))))
        i2 =  ((mstate.ebp+-1504))
        i3 =  ((mstate.ebp+-1808))
        __asm(push(i3), push((mstate.ebp+-2259)), op(0x3c), stack(-2))
        i3 =  ((mstate.ebp+-1664))
        __asm(push(i3), push((mstate.ebp+-2097)), op(0x3c), stack(-2))
        i3 =  ((mstate.ebp+-304))
        __asm(push(i3), push((mstate.ebp+-2115)), op(0x3c), stack(-2))
        i3 =  ((mstate.ebp+-104))
        __asm(push(i3), push((mstate.ebp+-2277)), op(0x3c), stack(-2))
        __asm(push(i1!=0), iftrue,
target("___vfprintf__XprivateX__BB79_2_F"))
    __asm(lbl("___vfprintf__XprivateX__BB79_1_F"))
        i1 =  (1)
        __asm(push(i1), push(___mlocale_changed_2E_b), op(0x3a),
stack(-2))
    __asm(lbl("___vfprintf__XprivateX__BB79_2_F"))
        i1 =  ((__xasm<int>(push(___nlocale_changed_2E_b), op(0x35))))
        __asm(push(i1!=0), iftrue,
target("___vfprintf__XprivateX__BB79_4_F"))
    __asm(lbl("___vfprintf__XprivateX__BB79_3_F"))
        i1 =  (1)
        __asm(push(i1), push(_ret_2E_993_2E_0_2E_b), op(0x3a), stack(-
2))
        __asm(push(i1), push(_ret_2E_993_2E_2_2E_b), op(0x3a), stack(-
2))
        __asm(push(i1), push(___nlocale_changed_2E_b), op(0x3a),
stack(-2))
    __asm(lbl("___vfprintf__XprivateX__BB79_4_F"))
        i1 =  (__2E_str1881)
        i3 =  ((__xasm<int>(push(_ret_2E_993_2E_0_2E_b), op(0x35))))
        i4 =  ((__xasm<int>(push((i0+12)), op(0x36))))
        i1 =  ((i3!=0) ? i1 : 0)
        __asm(push(i1), push((mstate.ebp+-2124)), op(0x3c), stack(-2))
        i1 =  (i0 + 12)
        __asm(push(i1), push((mstate.ebp+-2025)), op(0x3c), stack(-2))
        i1 =  (i4 & 8)
        __asm(push(i1==0), iftrue,
target("___vfprintf__XprivateX__BB79_7_F"))
    __asm(lbl("___vfprintf__XprivateX__BB79_5_F"))
        i1 =  ((__xasm<int>(push((i0+16)), op(0x37))))
        __asm(push(i1!=0), iftrue,
target("___vfprintf__XprivateX__BB79_9_F"))
    __asm(lbl("___vfprintf__XprivateX__BB79_6_F"))
        i1 =  (i4 & 512)
        __asm(push(i1!=0), iftrue,
target("___vfprintf__XprivateX__BB79_9_F"))
    __asm(lbl("___vfprintf__XprivateX__BB79_7_F"))
        mstate.esp -= 4
```

Usage

- Build env
  - ~1600 lines of Perl
  - ~800 lines of C
    - (swfbridge)



Adobe

Synchronous C / Asynchronous AS3

```c
// our main function...
// doesn't exit until program is done!
int main(int argc, char **argv)
{
        // ... do lots of stuff ...

        // exit program
        return 0;
}
```

```actionscript
// handle an event...
// return back to the Player when done
someObj.addEventListener(
    MouseEvent.CLICK,
    function(event:MouseEvent):void {
        // … do lots of stuff…
    }
);
```

- C is typically synchronous
  - Drawing, event handling, etc. is often done from within a synchronous event loop
- AS3 in Flash/AIR should be asynchronous
  - Player/AIR will “freeze” if script code executes continuously
- Synchronous code can be converted to asynchronous code

Synchronous C / Asynchronous AS3

Synchronous C Method

```
int sum(int a, int b){
    return a + b * 2;
}
```

Asynchronous FSM Class

```
switch(state){
    case 0:
        i2 = i1 * 2; // b * 2
        state++;
        return;
    case 1:
        i3 = i2 + i0; // a + ...
        state++;
        return;
    case 2:
        result = i2; // return ...
        gcurmachine = caller;
        return;
}
```

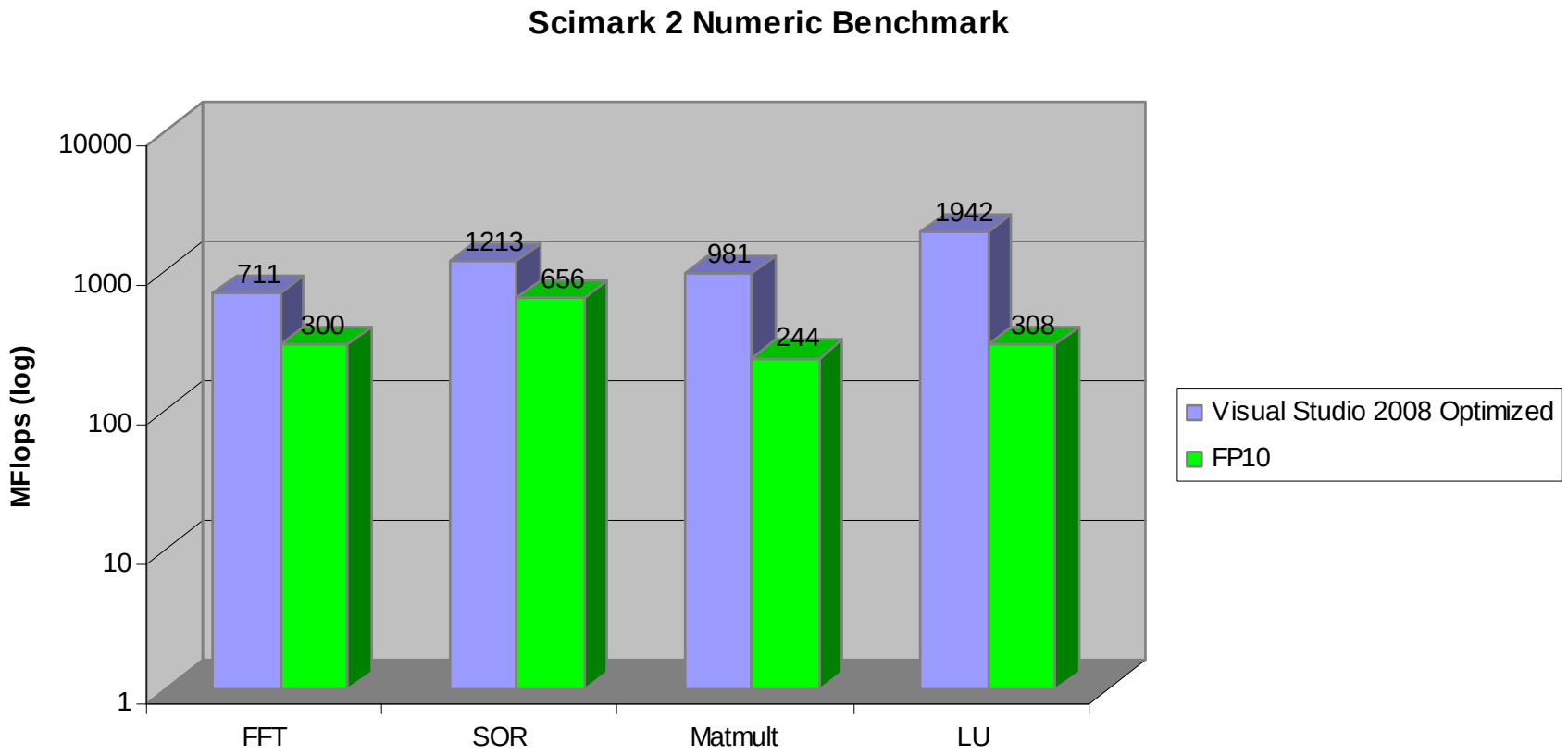
Asynchronicity

- FSMs for async functions
  - Function is genned as AS3 class
  - Registers are members of class
  - Function invoke translates to object instantiation
- Regular AS3 functions for sync functions
  - Still use virtual stack / x86 regs for parameter passing
- FSMs “chain”
  - Each FSM instance is conceptually a call frame
  - FSM instances contain links to the “caller” FSM
- FSMs given timeslices off of a Flash Timer object
- FSMs can emulate threads by slicing time between multiple FSM instances

System Services

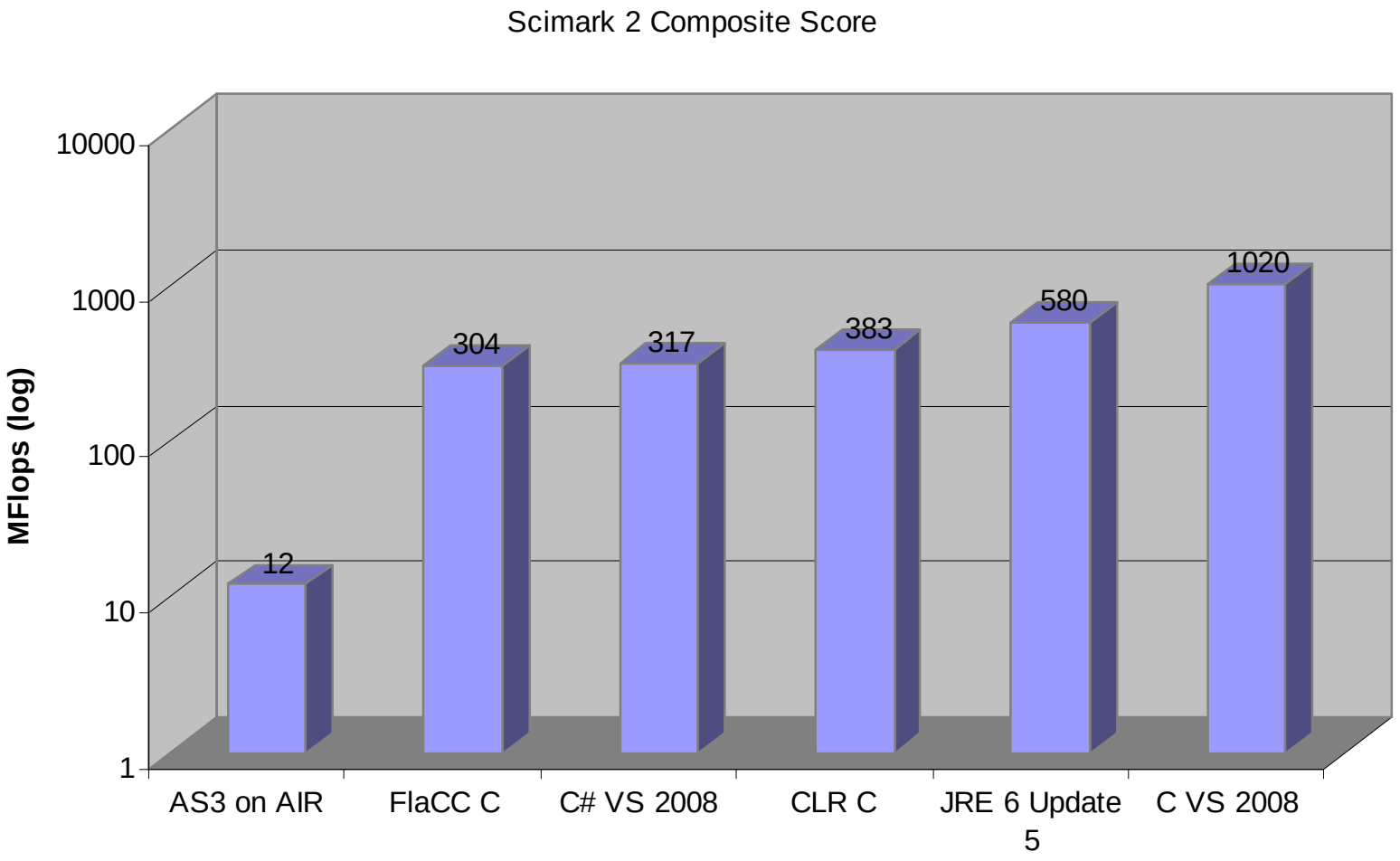
- Basic system services hand written in AS3 w/ C glue
  - ~3800 lines of AS3 / ~1775 lines of C
  - Low level I/O (through driver interface – open, read, write, close, etc.)
  - Low level memory management (mmap, sbrk)
  - setjmp/longjmp
- Rest of C standard library ported from BSD C standard library
  - FILE *operations
  - malloc/free
  - printf
  - Vast majority of C standard library provided this way

Performance



Adobe

Performance



Performance

- Demo

Fast memory ops

- Store opcodes
  - [stack: (coerced to int or Number) value, (coerced to int) address ->]
  - 0x35 **si8** – store an 8 bit integer to global memory
  - 0x36 **si16** – store a 16 bit integer to global memory
  - 0x37 **si32** – store a 32 bit integer to global memory
  - 0x38 **sf32** – store a 32 bit (IEEE 754) float to global memory (truncating the input double/Number to 32 bit IEEE 754)
- Load opcodes
  - [stack: (coerced to int) address -> value (int or Number)]
  - 0x3a **li8** – load an 8 bit unsigned integer from global memory
  - 0x3b **li16** – load a 16 bit unsigned integer from global memory
  - 0x3c **li32** – load a 32 bit integer from global memory
  - 0x3d **lf32** – load a 32 bit (IEEE 754) float from global memory and promote to 64 bit (IEEE 754) double/Number
  - 0x3e **lf64** – load a 64 bit (IEEE 754) float from global memory

Fast memory ops

- Sign extension opcodes
  - [stack: (coerced to int) value -> value (int)]
  - 0x50 **sxi1** – sign extend from 1 bit to 32
  - 0x51 **sxi8** – sign extend from 8 bits to 32
  - 0x52 **sxi16** – sign extend from 16 bits to 32
- AS3 API
  - ApplicationDomain.domainMemory:ByteArray
  - static ApplicationDomain. MIN_DOMAIN_MEMORY_LENGTH:uint
- Impl
  - On x86, JITs to direct memory accesses to hard-coded addresses
  - On x86, range checking JITs to compares against immediates
  - Hard-coded addresses and immediate range check nums modified on-the-fly in live machine code

Debugging

- DWARF debug info embedded into SWFs as static data
  - Describes every variable's type, name, etc.
  - Describes complex types (structs, unions, arrays)
- AS3 structures embedded into SWFs
  - Source file names
  - Function names
  - Variable scope info, frame location (and mappings to corresponding DWARF structures)
- Debug pseudo-instructions in generated methods
  - "debugLoc" function call inserted for each debug location / step point (usually a single line of code)
  - "debugLoc" is passed line number info, etc. for the location is corresponds to
    - Checks line number, etc. against current breakpoint set and notifies framework if it corresponds to a live BP

Debugging

- System service boilerplate can speak GDB/MI over a socket
  - ~670 lines of AS3
  - Provides limited functionality in Eclipse
    - Source file / line # info
    - Breakpoints
    - Integer-only variable / register inspection
  - Theoretically could work w/ XCode too

C AS3 API

- API

```
#ifndef AS3_H
#define AS3_H

/*
** AS3 <=> C bridging API
*/

#ifdef __cplusplus
extern "C" {
#endif

/* ref counted AS3 value type */
typedef struct _AS3_Val *AS3_Val;

/* just a char * but was malloced (so should be freed) */
typedef char *AS3_Malloced_Str;

#ifndef AS3_NO_C_API

/* all values are ref counted
   and you must release all values
   EXCEPT the params object passed
   to a thunk
*/
void AS3_Acquire(AS3_Val obj);
void AS3_Release(AS3_Val obj);

/* ns::[prop] */
AS3_Val AS3_NSGet(AS3_Val ns, AS3_Val prop);
AS3_Val AS3_NSGetS(AS3_Val ns, const char *prop);

/* obj[prop] */
AS3_Val AS3_Get(AS3_Val obj, AS3_Val prop);
AS3_Val AS3_GetS(AS3_Val obj, const char *prop);

/* obj[prop] = val */
AS3_Val AS3_Set(AS3_Val obj, AS3_Val prop, AS3_Val val)
AS3_Val AS3_SetS(AS3_Val obj, const char *prop, AS3_Val
val);

/* typeof obj */
AS3_Malloced_Str AS3_TypeOf(AS3_Val obj);

/* AS3 value creation */
AS3_Val AS3_String(const char *str);
AS3_Val AS3_StringN(const char *str, int len);
AS3_Val AS3_Int(int n);
AS3_Val AS3_Number(double n);
AS3_Val AS3_True();
AS3_Val AS3_False();
AS3_Val AS3_Null();
/* undefined is guaranteed to be (AS3_Val)NULL
   (null is NOT!)
*/
AS3_Val AS3_Undefined();

/* AS3 value conversion */
AS3_Malloced_Str AS3_StringValue(AS3_Val obj);
int AS3_IntValue(AS3_Val obj);
double AS3_NumberValue(AS3_Val obj);

/* utility to create an array from a type template
```

C AS3 API

- Example

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#include "AS3.h"

void flyield();

const char *TestThunkStr = NULL;

AS3_Val TestThunk(void *data, AS3_Val params)
{
 static char buf[256];

 sprintf(buf, "%p", data);
 sztrace(buf);
 TestThunkStr = buf;
 return NULL;
}

/* trace(new Date()); trace("foo");
  Wordy version with correct ref counting
*/
void TraceTest()
{
 AS3_Val DateClass = AS3_NSGetS(NULL, "Date");
 AS3_Val emptyParams = AS3_Array("");
 AS3_Val DateObj = AS3_New(DateClass, emptyParams);
 AS3_Val trace = AS3_NSGetS(NULL, "trace");
 AS3_Val fooParams = AS3_Array("StrType", "foo");
 AS3_Val traceParams = AS3_Array("AS3ValType",
DateObj);

 /* trace(new Date()) */
 AS3_Release(AS3_Call(trace, AS3_Undefined(),
traceParams));
 /* trace("foo") */
 AS3_Release(AS3_Call(trace, AS3_Undefined(),
fooParams));

 AS3_Release(traceParams);
 AS3_Release(fooParams);
 AS3_Release(trace);
 AS3_Release(DateObj);
 AS3_Release(emptyParams);
 AS3_Release(DateClass);
}

int main()
{
 /* to perusers -- this test leaks like a sieve!
    all AS3_Vals EXCEPT one passed to your Thunk
    must be AS3_Release-ed
    and char *s must be free-ed!
 */
 fprintf(stderr, "Starting\n");
 fprintf(stderr, "typeof Date: %s\n",
  AS3_TypeOf(AS3_NSGetS(NULL, "Date")));
 fprintf(stderr, "typeof trace: %s\n",
  AS3_TypeOf(AS3_NSGetS(NULL, "trace")));
 fprintf(stderr, "typeof string: %s\n",
  AS3_TypeOf(AS3_String("foo")));
```

C library as SWC

- C glue

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#include "AS3.h"

// a C thunk for an AS3 function
AS3_Val someEntry(void *data, AS3_Val args)
{
 // set up some vars w/ default values
 int n = 34, k = 1234;
 double d = 12.34;
 char buf[256];

 // parse the input args (if there are <3 args, not all vars
 // will be set and will retain default values
 AS3_ArrayValue(args, "IntType, DoubleType, IntType",
&n, &d, &k);

 // put something in the buf
 sprintf(buf, "%d %f %d", n, d, k);

 // return it
 return AS3_String(buf);
}
```

```
int main()
{
 // regular function
 AS3_Val someEntryVal = AS3_Function(NULL, someEntry)

 // async function
 AS3_Val someEntry2Val = AS3_FunctionAsync(NULL,
someEntry);

 // construct an object that holds refereces to the 2
functions
 AS3_Val result = AS3_Object("someEntry: AS3ValType,
someEntry2: AS3ValType",
   someEntryVal, someEntry2Val);

 AS3_Release(someEntryVal);
 AS3_Release(someEntry2Val);

 // notify that we initialized -- THIS DOES NOT RETURN!
 AS3_LibInit(result);

 // XXX never get here!
 return 0;
}
```

C library as SWC

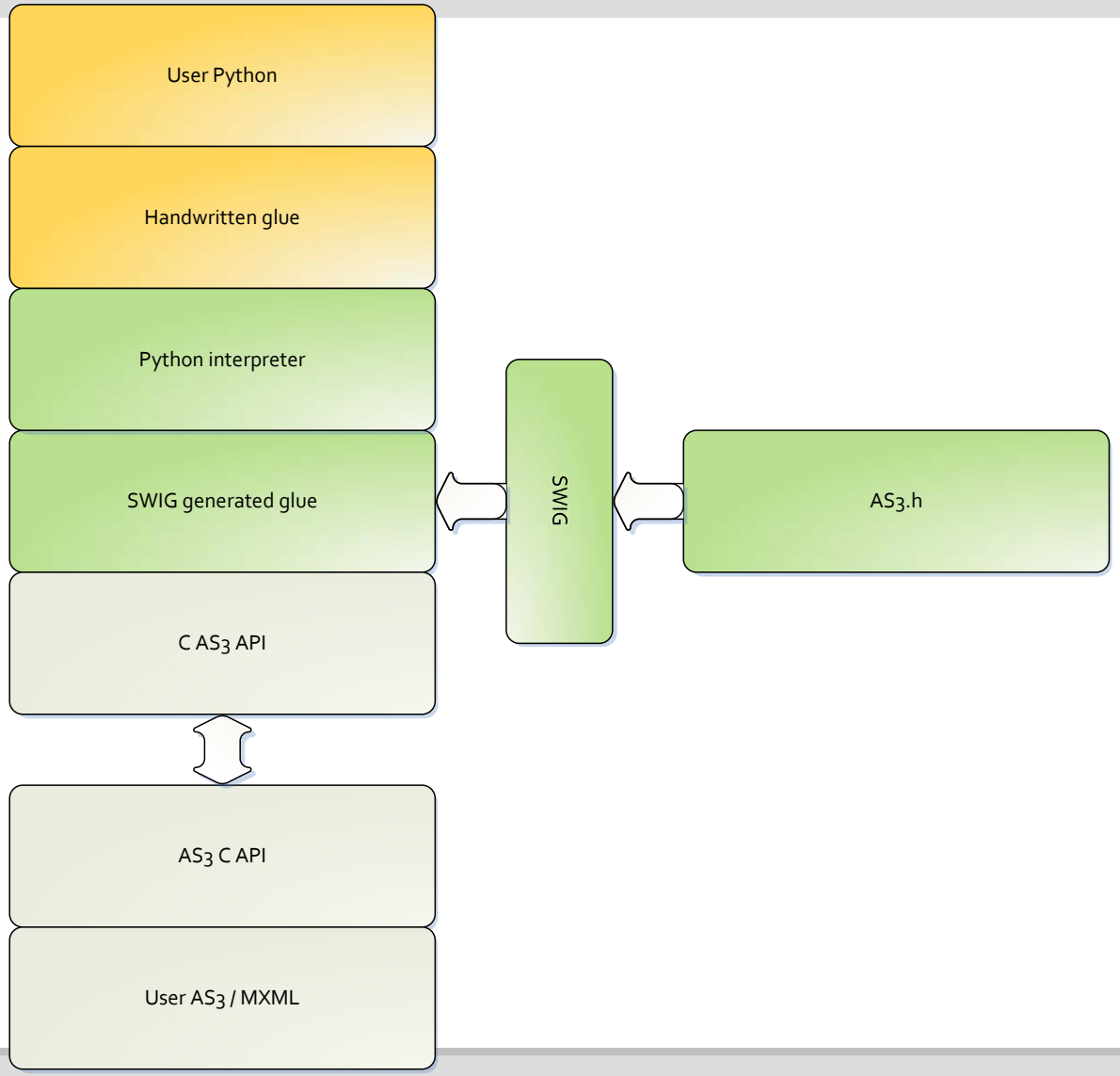
- Example

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute">
    <mx:Script>
        <![CDATA[
            import cmodule.test.CLibInit;

            function test():void
            {
                // create lib initialization object
                var lib:CLibInit = new CLibInit();
                textInp.text = "Initting lib...";
                // initialize...
                var res:* = lib.init();
                textInp.text = "Initted lib";
                // call someEntry w/ default params
                textInp.text += "\n" + res.someEntry();
                // call someEntry w/ all params
                textInp.text += "\n" + res.someEntry(1,2,3);
                // call someEntry2 (which is async!) w/ 1
                // params
                res.someEntry2(function(res:*):void
                {
                    // we get called back with the one
                    // parameter being the result
                    // of the async function
                    textInp.text += "\n" + res;
                }, 17 /* arg */);
                textInp.text += "\n" + "---";
            }
        ]]>
    </mx:Script>
    <mx:TextInput x="10" y="10" width="325" height="
="textInp" creationComplete="test()"/>
</mx:Application>
```

Interpreter integration

- Low level API:
  - AS3.h x AS3.swig
- AS3.py



Interpreter integration

- AS3.py

```
import AS3Glue

class AS3Namespace:
 def __init__(self, *args):
  ns = ""
  for i in args:
   if(len(i)):
    if(len(ns)): ns = ns + "."
    ns = ns + i
  self.__ns = ToAS3(ns)
 def __str__(self):
  return FromAS3(self.__ns)
 def __setattr__(self, item, value):
  if item == "_AS3Namespace__ns":
   self.__dict__.__setitem__(item, value)
  else:
   raise AttributeError(item)
 __setitem__ = __setattr__
 def __getattr__(self, item):
  if item == "_AS3Namespace__ns":
   return self.__dict__.__getitem__(item)
  v = AS3Glue.Value()
  v.Set(AS3Glue.AS3_NSGetS(self.__ns.Get(), item))
  if v.Get() == AS3Glue.AS3_Undefined():
   return AS3Namespace(FromAS3(self.__ns), item)
  else:
   return FromAS3(v)
 __getitem__ = __getattr__
```

```
class AS3Object:
 def __init__(self, val):
  self.__val = val
 def __str__(self):
  return self.toString()
 def __call__(self, *args, **kwargs):
  aargs = ToAS3(args)
  return FromAS3(AS3Glue.AS3_New(self.__val.Get(),
aargs.Get()))
 def __setattr__(self, item, value):
  if item == "_AS3Object__val":
   self.__dict__.__setitem__(item, value)
  else:
   val = ToAS3(value)
   AS3Glue.AS3_SetS(self.__val.Get(), item, val.Get())
 __setitem__ = __setattr__
 def __getattr__(self, item):
  if item == "_AS3Object__val":
   return self.__dict__.__getitem__(item)
  return FromAS3(AS3Glue.AS3_GetS(self.__val.Get(),
item), self.__val)
 __getitem__ = __getattr__


AS3global = AS3Namespace()
flash = AS3Namespace("flash")
stage = FromAS3(AS3Glue.AS3_Stage())
```

Interpreter integration

- Flash.py
- Demo

```
from AS3 import *

class MySprite:
  def __init__(self, color):
    self.sprite = flash.display.Sprite()
    spriteg = self.sprite.graphics
    spriteg.beginFill(color, 0.5)
    spriteg.drawCircle(0, 0, 50)
    spriteg.endFill()
    self.sprite.addEventListener(flash.events.MouseEvent.MOUSE_DOWN, self.mouseDown)
    self.sprite.addEventListener(flash.events.MouseEvent.MOUSE_UP, self.mouseUp)
    stage.addChild(self.sprite)
  def mouseDown(self, event):
    self.sprite.addEventListener(flash.events.MouseEvent.MOUSE_MOVE, self.mouseMove)
    self.sprite.startDrag()
  def mouseUp(self, event):
    self.sprite.removeEventListener(flash.events.MouseEvent.MOUSE_MOVE, self.mouseMove)
    self.sprite.stopDrag()
  def mouseMove(self, event):
    event.updateAfterEvent()

a = MySprite(0xff0000)
b = MySprite(0x00ff00)
c = MySprite(0x0000ff)
d = MySprite(0x00ffff)
e = MySprite(0xff00ff)
f = MySprite(0xffff00)
```

Better by Adobe.™

Adobe