It's All Glue

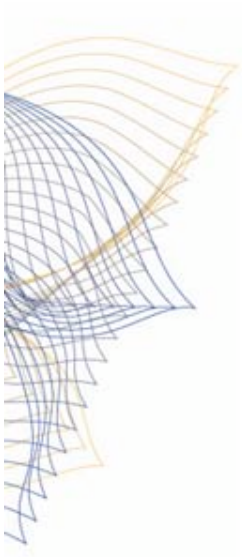
Building a desktop application with Lua

Mark Hamburg

Adobe Fellow

27 July 2005

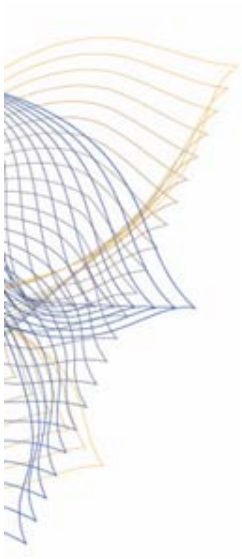
Adobe



Starting Out

- **Looking at new interaction models**
  - Particularly interested in what we could learn from games
- **Long-standing interest in extensible systems**
  - Very interested in Oberon about 11 years ago
  - AutoCAD and AutoLISP seemed like an interesting model of what deep-scripting could do for one
- **Learned about Lua via the GC mailing list**

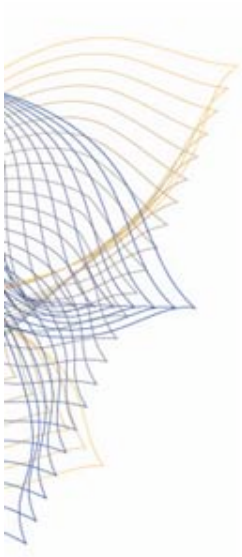
Adobe



Evolution

- **Shifting from a C++ bias to a C (Objective-C) bias**
  - C++ has grown frighteningly complex
  - Difficult to build an extensibility story around C++
- **Could we make Lua a peer to the C code?**
  - Already had Objective-C based plug-ins
  - Implemented Lua support for the plug-in loader
- **Implemented parallel namespace support for C-based APIs and Lua-based APIs**

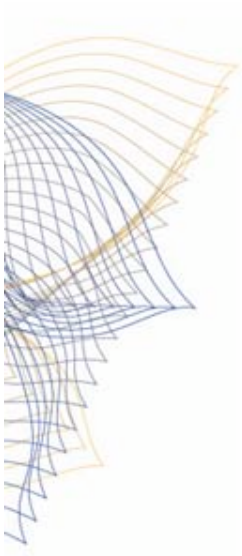
Adobe



Evolution continued

- **More pieces start getting implemented in Lua**
  - How far can we take this?
  - To what extent do we need to maintain support for a pure C path?
- **Standard platform conventions such as plists give way to Lua-based manifests**
- **Gradual absorbtion of "the Lua way"**
  - C code should be minimal and exists to handle performance critical inner loops and interfacing to the OS
  - As much of the interesting logic as possible goes into Lua
  - Build "pretty" APIs using Lua
  - Whenever possible test while coding

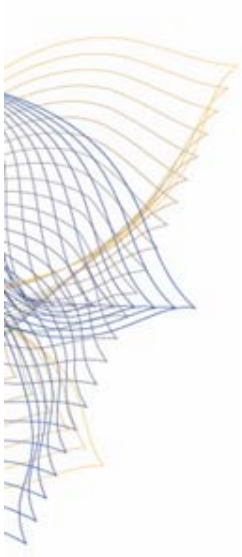
Adobe



Project Breakdown

- **About 40% Lua**
  - 100,000 lines of Lua
  - 150,000 lines of C, C++, Objective-C, etc.
    - Excludes “third-party” libraries including those from within Adobe

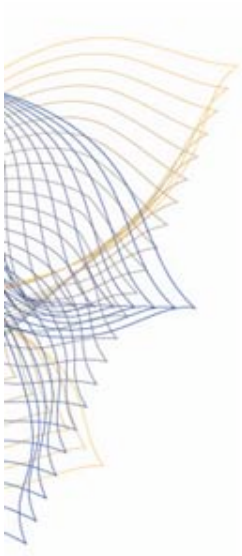
Adobe



But LOC is deceptive…

- **Lua code includes some significant subsystems**
  - Namespace management
  - Observations & Notifications
  - View layout
  - Database abstraction
  - Most of the task system logic
- **Virtually all of the actual UI logic**

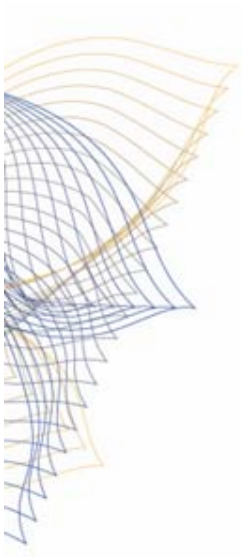
Adobe



Achievements

- Flexible event handling
- Flexible data handling
- Low project bug count
- Very low crash count
- QA engineer generating production code

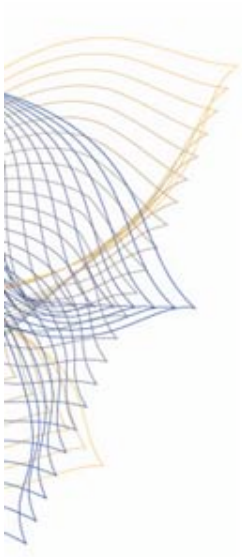
Adobe



Mechanisms

- Objective-C bridging
- Multiple universes

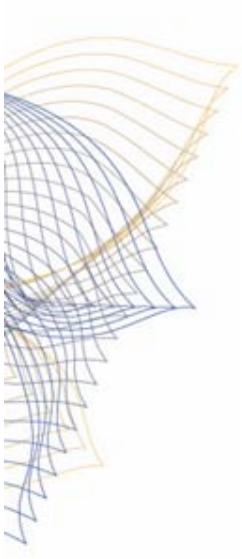
Adobe



Objective-C Bridging

- **Lots of bridges out there**
  - CocoaDev
  - Steve Dekorte had one though it seems to have moved
  - Many are more aggressive than ours
- **Enabled by the availability of introspection data in the Objective-C runtime**
  - This makes Lua to Objective-C calls easy
  - Automatic extensions have to contend with Objective-C's use of ":" [dict setValue: value forKey: key]
  - Not as easy to implement objects in Lua that are transparently callable from Objective-C

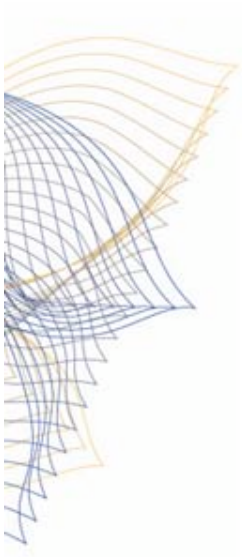
Adobe



Objective-C Bridging (continued)

- **Extension on our part to deal with naming and to allow for greater parameter list flexibility:**
  - Objective-C: - (int) myMethod_L: lua_State* L { }
  - Lua: obj:myMethod(1, 2, 3)
- **Added support for property-style access in addition to method-style access**
  - myObj.x
  - myObj:x()
  - Complicated on reads by the fact that at __index time, you don't know how the value will be used
    - We've got a __methods metamethod patch to the LuaVM
- **Primary Lesson: Languages with good introspection make it easy to export APIs to Lua**

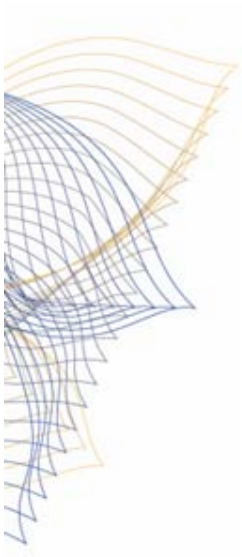
Adobe



Multiple Universes: Prelude

- **Started out by trying LuaThreads**
- **The mutex locks basically kill performance**
  - 25-50% speed hit in some tests
  - Memory synchronization bites you even in the absence of contention
- **LuaThreads is unsafe with respect to some function in the library such as ref manipulation**
  - So, we need more locks…
- **Looked at doing things to reduce lock traffic**
  - Those all became scary in their complexity

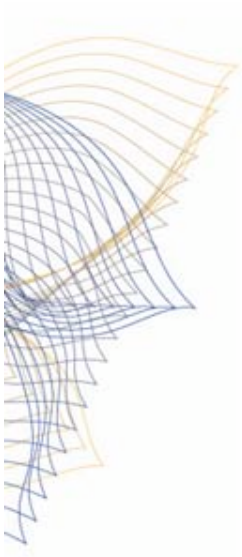
Adobe



Multiple Universes: Solution

- **Do processing in separate universes — i.e., independently opened Lua states**
- **Logic driving universes is written in C**
  - If doing it over again, it would probably be in Lua
- **Communicate via a "transit universe" that is subject to a mutex at the transit universe API level**

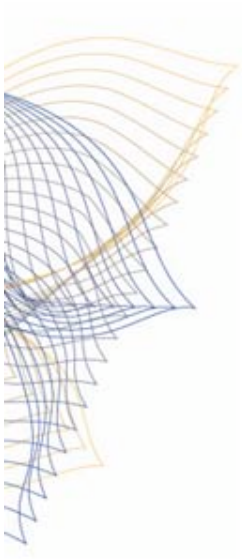
Adobe



Transit Universe

- **Supports transfer of primitive Lua types**
  - Numbers
  - Strings
  - Booleans
  - Light userdata
- **Supports transfer of tables**
  - No logic to deal with DAGs
- **Supports transfer of Objective-C objects**
- **No support for:**
  - Functions: Use dump & load
  - Metatables
  - Arbitrary userdata

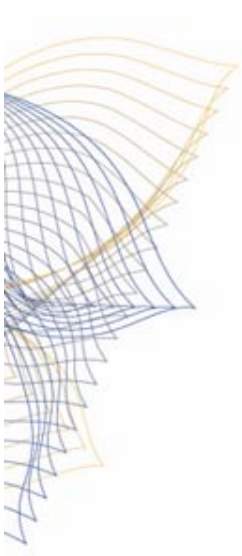
Adobe



Transit Universe

- **C-based API**
- **If it had a Lua-based API, it might look something like:**
  - transitUniverse.put(value) -- returns a token that can be transferred between universes via some other mechanism
  - transitUniverse.get(token) -- returns the value associated with a token returned by transitUniverse.put
  - transitUniverse.delete(token) -- deletes the value associated with the token in the transit universe

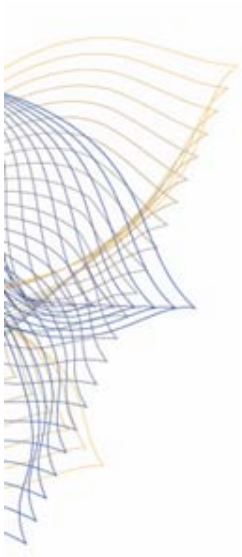
Adobe



Challenges

- Garbage collection performance
- Garbage collection cycles
- Temporary states
- Lack of static type-checking
- Performance measurement

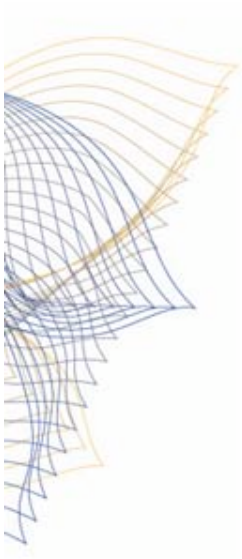
Adobe



Garbage Collection Performance

- **GC pauses are disturbing when running animations**
  - Incremental collection smooths those out
- **Heap allocation is slower than stack allocation**
  - Returning a rectangle struct on the stack in C is a lot cheaper than allocating a rectangle object, returning it, and then collecting it
  - For small structures, the solution is to work with them unpacked — i.e., pass x and y rather than a point
    - myObject:offsetBy(x, y)
    - myObject:offsetBy(point)
  - Pass in destination storage as an optional parameter
    - myObj:bounds(storage) but also myObj:bounds()
  - Careful allocation of temporaries at outer scopes

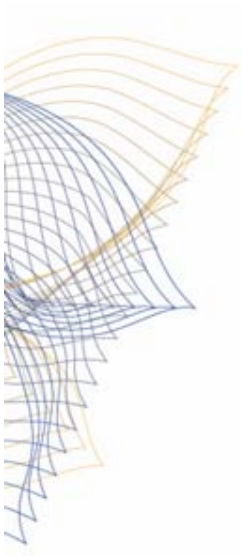
Adobe



Garbage Collection Cycles

- **C points to Lua points to C points to Lua points to…**
- **Refs created via Lua have to be manually broken — i.e., they create the moral equivalent of reference-counting cycles**
- **Worked with some really ugly hacks based on per-object metatables**
- **Lua 5.1w5's addition of environments for userdata has fixed all this**
  - Store links to other Lua objects in the environment and let the Lua GC trace them
  - Be happy that Objective-C allows one to change the behavior on retain and release calls for existing classes

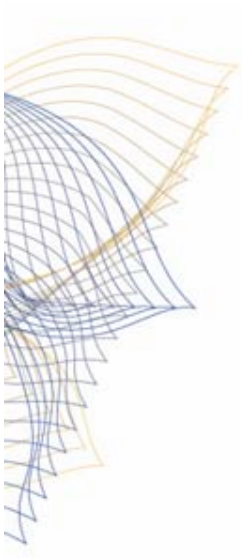
Adobe



Temporary States

- **Problem: Lots of C code doesn't have a Lua state directly available to it**
- **Solution: Maintain a pool of states for a universe (allocated via lua_newthread) so that we can just grab one**
- **Problem: Cleaning up after errors is messy if one isn't inside a pcall or a cpcall**
  - lua_cpcall is awkward to use
  - lua_cpcall is expensive: it allocates a new function every time
  - Catching exceptions without cleaning up the state is bad
- **Solution: Catch the exception but recognize that the state wasn't properly cleaned up and let the garbage collector deal with it**

Adobe



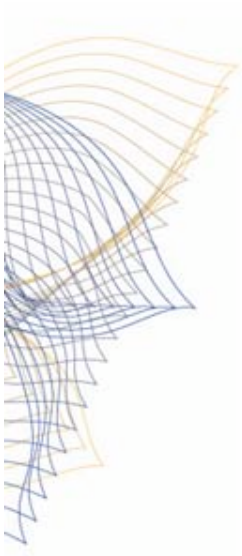
Lack of Static Type-Checking

- Programmers make typos
- Catching everything at runtime requires exhaustive testing and some bugs can be subtle
- Unit tests help but don't work as well for UI code

- Wrong name v wrong type

- Added checks at the global environment via __index and __newindex metamethods
- Added lint tool that checks the files
- Class constructor looks for an optional __fields entry and if so adds __index and __newindex metamethods to check keys

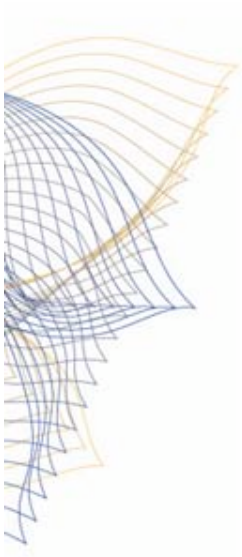
Adobe



Performance Measurement

- Lots of stack crawls just include a section doing something in the Lua VM
- Partial Solution: Put in wrappers to measure time and generate profiling information where we suspected issues

Adobe



Key Lua Strengths

- **Coroutines**
  - Coroutines and closures are more valuable than objects
- **Robust tables**
- **Metamethods make bridging easy (mostly)**
- **Data-description is natural**
- **Simplicity**
- **Vibrant community**

Adobe


Adobe