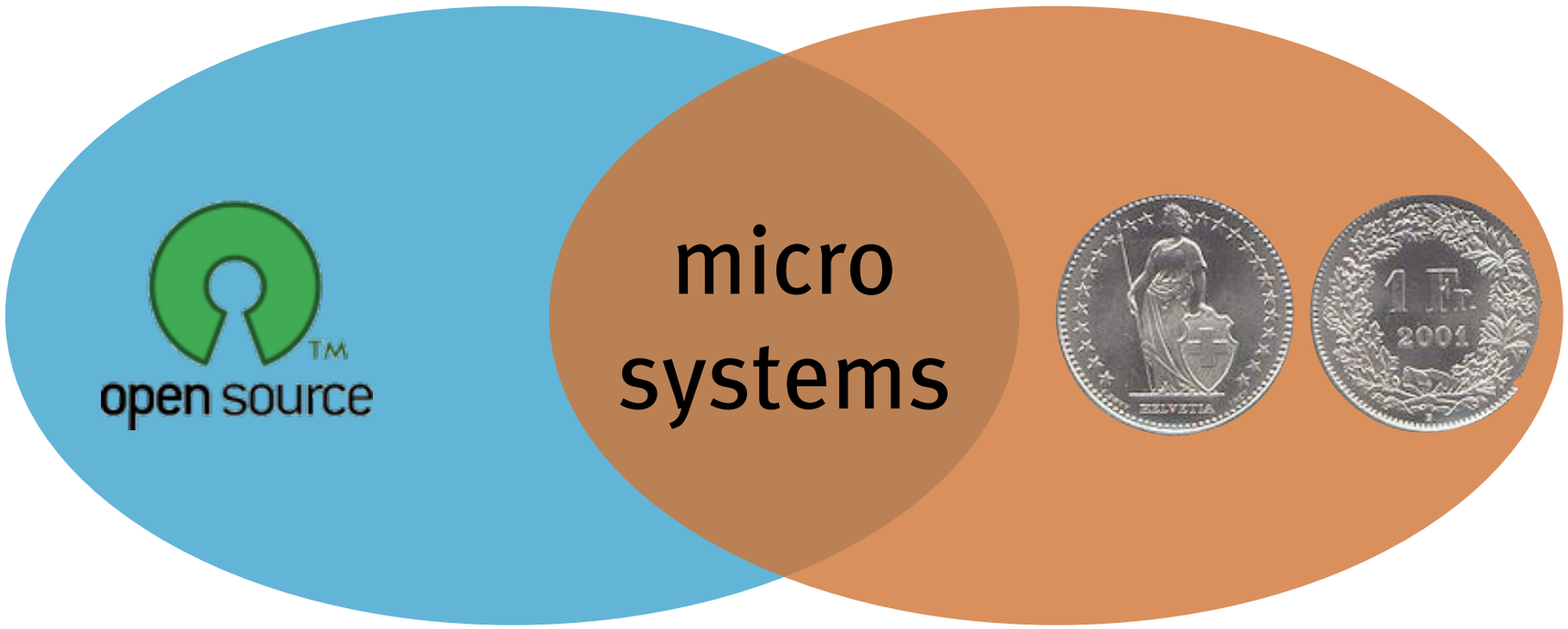
```
-- Marc Balmer, micro systems, <marc@msys.ch>
-- FOSDEM 2011, Brussels



function presentation()
    print(„Lua meets BSD“);
end
```


*micro systems*

[Who we are and what we do and why I like Lua]

# *Free Software, Commercial Applications*

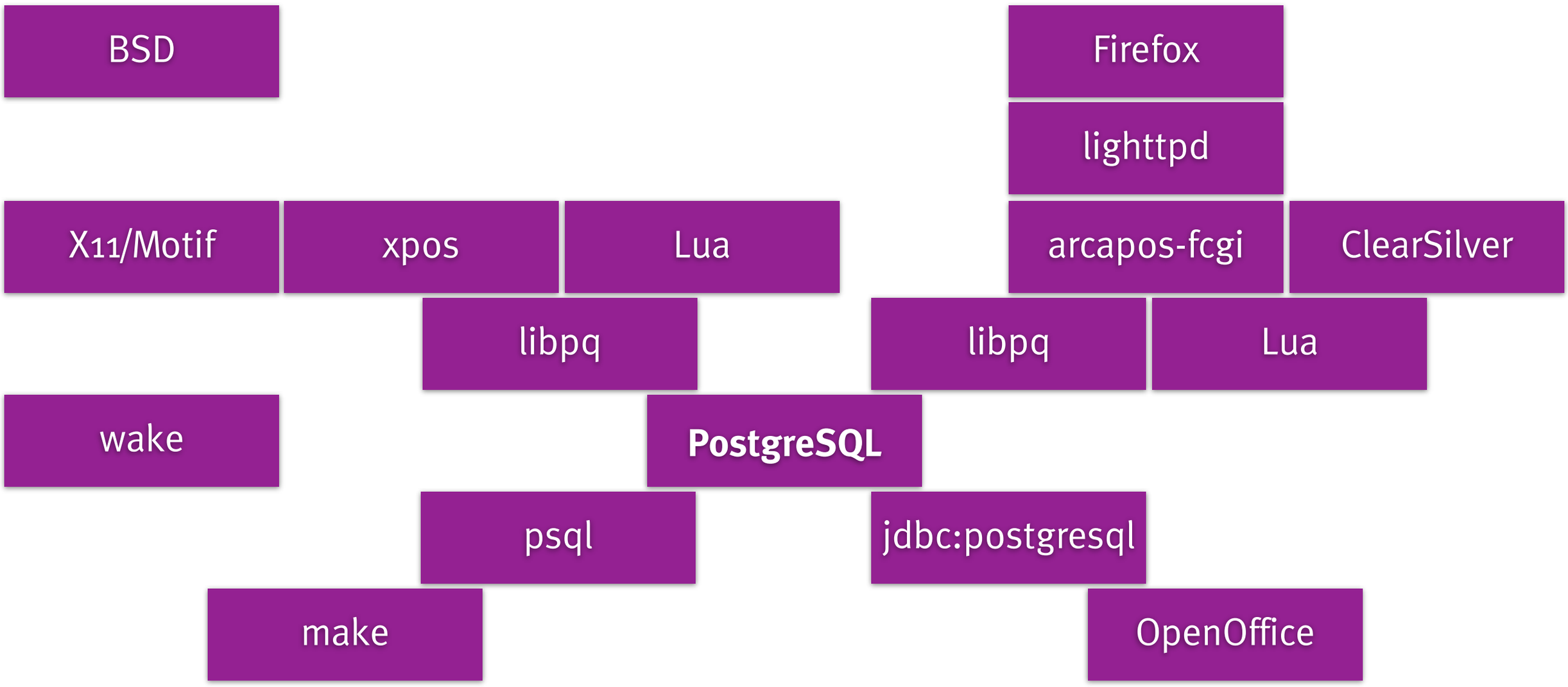


# *IANAL*

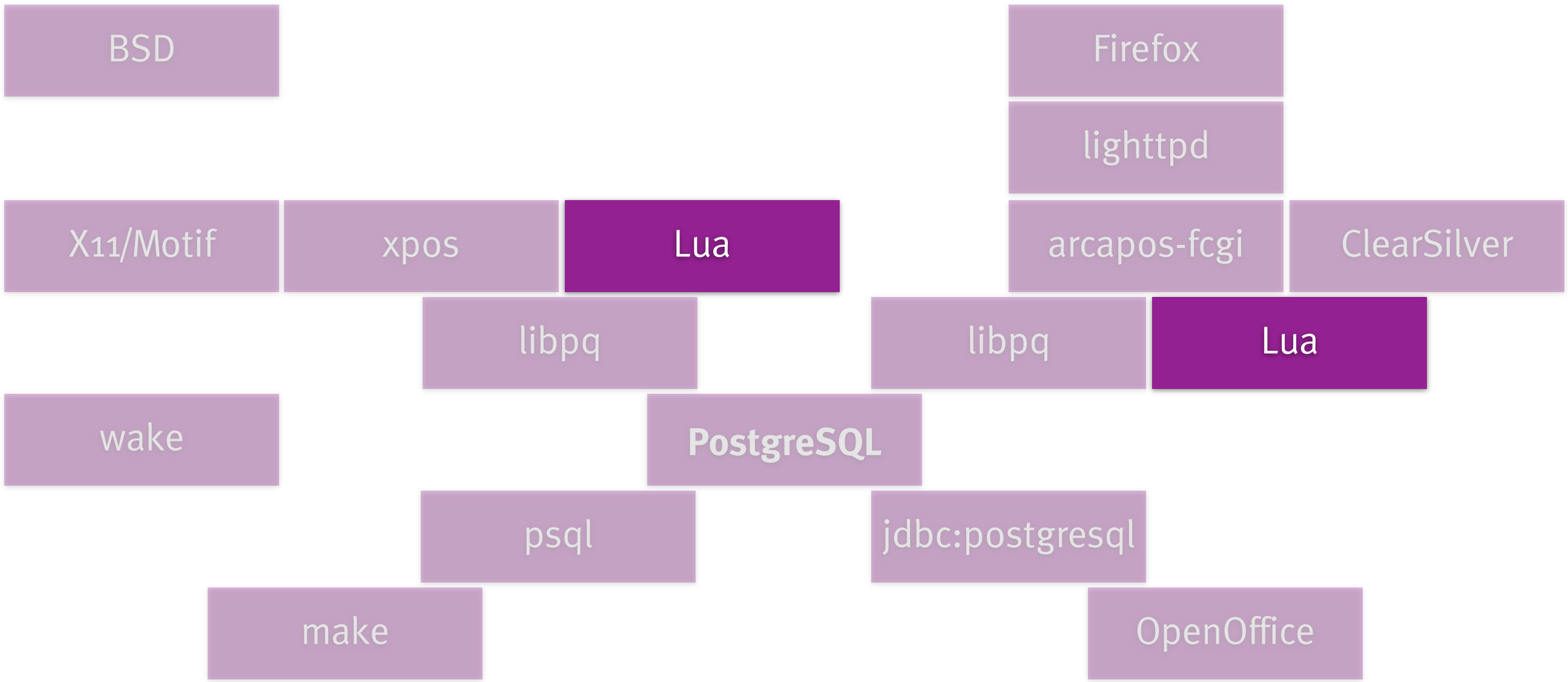

# *Short and Free of Restrictions*

```
Copyright (c) CCYY YOUR NAME HERE
<user@your.dom.ain>

Permission to use, copy, modify, and distribute this
software for any purpose with or without fee is
hereby granted, provided that the above copyright
notice and this permission notice appear in all
copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR
DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE
FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS
OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF
CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION,
ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

# *Our Products run On NetBSD, use Lua for Extenstions*



# *Our Products run On NetBSD, use Lua for Extenstions*



# *My Vision*

Modifying software written in C is hard^wimpossible for users

Give the power to modify and extend to the user

Let the user explore the system

*This was *NOT* my Goal:*

Provide a language to write system software in

# *Considering Some Alternatives*

Python

Java

But not Perl, Tcl, Javascript

# *Python*

Not to difficult to integrate in C

Huge library

Memory consumption

Difficult object mapping

# *Java*

Easy to integrate

Difficult object mapping

Memory considerations

# FAST
# POWERFUL
# LIGHTWEIGHT
# EMBEDDABLE
# SCRIPTING
# LANGUAGE
# & VM


**Builds in all platforms with an ANSI/ISO C compiler**
**Fits into 128K ROM, 64K RAM per interpreter state[1]**
**Fastest in the realm of interpreted languages**
**Well-documented C/C++ API to extend applications**
**One of the fastest mechanisms for call-out to C**
**Incremental low-latency garbage collector**
**Sandboxing for restricted access to resources**
**Meta-mechanisms for language extensions,**
**e.g. class-based object orientation and inheritance**
**Natural datatype can be integer, float or double**
**Supports closures and cooperative threads**
**Open source under the OSI-certified MIT license**

[1] Complete Lua SOC, practical applications in 256K ROM / 64K RAM

Designed, implemented and maintained at the
Pontifical Catholic University of Rio de Janeiro

**www.lua.org**

# *The Lua Programming Language*

Very simple syntax with some syntactic sugars

Easy to learn

Tables are *THE* data structure

# *E.g. Declarative Style GUI Programming*

[A small demonstration]

# *Embedding Lua in C Programs*

Create one or more Lua states

Execute Lua code in them

Perfect for sandboxing

# *The Stack Based Approach*

All data exchange between C and Lua goes over a virtual stack

Push parameters to the stack

Pull results from the stack

# *Calling C Code from Lua*

Wrap the C function

Export the wrapper function to Lua, either globally or as a table element

Call it from Lua

# *SYSLOG(3)*

```
static int
lua_syslog(lua_State *L)
{
     const char *str;
     int level;

     level = lua_tointeger(L, 1);
     str = lua_tostring(L, 2);
     syslog(level, "%s", str);
     return 0;
}
```

# *Export the C Function to Lua*

```
lua_register(L, "syslog", lua_syslog);
```

# *Exporting #defines*

```
struct constant {
    char *name;
    int value;
};

static struct constant syslog_level[] = {
    { "LOG_EMERG",        LOG_EMERG },
    { "LOG_ALERT",        LOG_ALERT },
    { "LOG_CRIT",         LOG_CRIT },
    { "LOG_ERR",      LOG_ERR },
    { "LOG_WARNING", LOG_WARNING },
    { "LOG_NOTICE",       LOG_NOTICE },
    { "LOG_INFO",         LOG_INFO },
    { "LOG_DEBUG",        LOG_DEBUG },
    { NULL,           0 }
};
```

# *Mapping them to Global Symbols*

```
for (n = 0; syslog_level[n].name != NULL; n++) {
    lua_pushinteger(L, syslog_level[n].value);
    lua_setfield(L, LUA_GLOBALSINDEX, syslog_level[n].name);
};
```

# *Calling it from Lua*

```
function startup()
    -- do important stuff

    -- log that we just started
    syslog(LOG_INFO, ,The application has started‘)
end
```

# *Calling Lua Code From C*

Define a function in Lua

Call it by name or Lua reference

# *Call by Name*

```
extern char *reason;

int
call_by_name(lua_State *L, const char *fname)
{
    int ret;

    lua_getfield(L, LUA_GLOBALSINDEX, fname);
    ret = lua_pcall(L, 0, 0, 0);
    if (ret)
        reason = (char *)lua_tostring(L, -1);
    return ret;
}
```

# *Get a Reference*

```
int ref;

lua_pushstring(L, „foobar“);
ref = luaL_ref(L, LUA_GLOBALSINDEX);

call(L, ref);
```

# *Call by Lua Reference*

```
int
call(lua_State *L, int ref)
{
     int ret;

     lua_rawgeti(L, LUA_GLOBALSINDEX, ref);
     ret = lua_pcall(L, 0, 0, 0);
     if (ret)
          reason = (char *)lua_tostring(L, -1);
     return ret;
}
```

# *Lua as a Configuration Language*

Replace lex/yacc

Flexible configuration

Easy to generate, not easy to parse by others

# *xpos.conf*

```
-- xpos configuration using Lua

Addimat.Device = '/dev/dty00'

BarcodeScanner.Device = '/dev/ttyq2'
BarcodeScanner.Type = 'usb'

CustomerDisplay.Device = '/dev/ttyq1'

ReceiptPrinter.Device = '/dev/ttyq0'
ReceiptPrinter.Type = 'ibm'
ReceiptPrinter.PrintOnDemand = true
ReceiptPrinter.PrintUsername = true

TicketPrinter.Host = 'pf4i'
```

# Reading the Config File

```
int
parse_config(char *cfgfile)
{
      lua_State *L;
struct stat sb;
char *s;

L = luaL_newstate();
config_openlib(L, "", luaopen_base);
config_openlib(L, LUA_LOADLIBNAME, luaopen_package);

/* prepare tables to be populated by the config file */
lua_newtable(L);
lua_setfield(L, LUA_GLOBALSINDEX, "Addimat");

/* Run the config file */
if (!stat(cfgfile, &sb)) {
     if (luaL_loadfile(L, cfgfile) || lua_pcall(L, 0, 0, 0))
          errx(1, "config boo boo: %s", lua_tostring(L, -1));
}

/* evaluate the configuration */
lock_dev = cfg_string(L, "Addimat", "Device", NULL);
```

# Accessing a Config Variable

```
static char *
cfg_string(lua_State *L, char *table, char *elem, char *dflt)
{
    char *r;

    lua_getglobal(L, table);
    lua_getfield(L, -1, elem);
    if (lua_isnil(L, -1)) {
        lua_pop(L, 2);
        return dflt;
    }
    if (!lua_isstring(L, -1))
        errx(1, "string expected for %s.%s", table, elem);
    r = strdup((char *)lua_tostring(L, -1));
    lua_pop(L, 2);
    return r;
}
```

# *Lua in NetBSD Userland*

Library (liblua.so) and binaries (lua, luac) committed to -current

Will be part of NetBSD 6

No back port to NetBSD 5 stable

# *Lua in the NetBSD Kernel*

## GSoC 2010 Project „Lunatic“

Scripting of subsystems (firewall)

Device drivers, tinkering with hardware

Research type of project

# *Lua in FreeBSD (not yet...)*

Userland parts can be considered done

Time restraints

# *Future Work in NetBSD*

sysinst

bluetooth

gpio

tty line disciplines

*In god we trust, in C we code!*

**Marc Balmer**

marc@msys.ch, m@x.org,

mbalmer@NetBSD.org

www.msys.ch, www.arcapos.com