What happens when a DWARF and a daemon start dancing by the light of the silvery moon?

The use of DWARF debug information to dynamically project the embedded extension language Lua's global environment onto the NetBSD kernel's internal state.

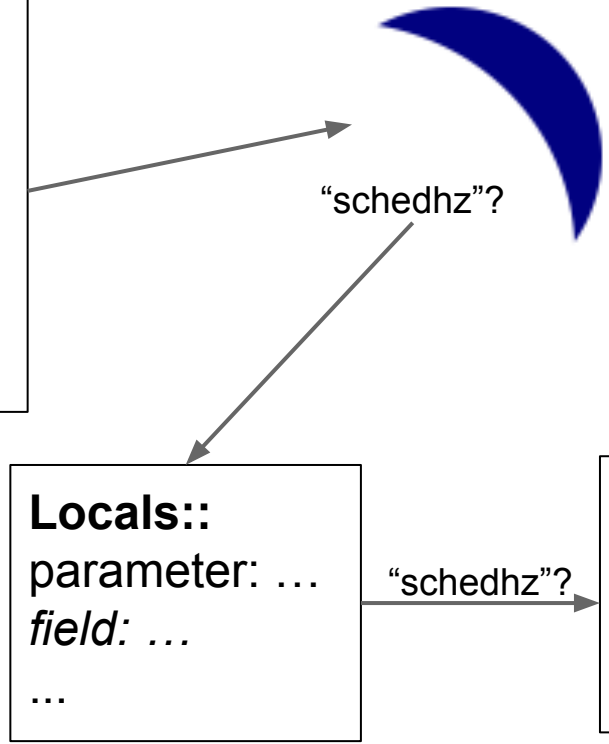
Andrew Cagney, BSDCan 2105

What?

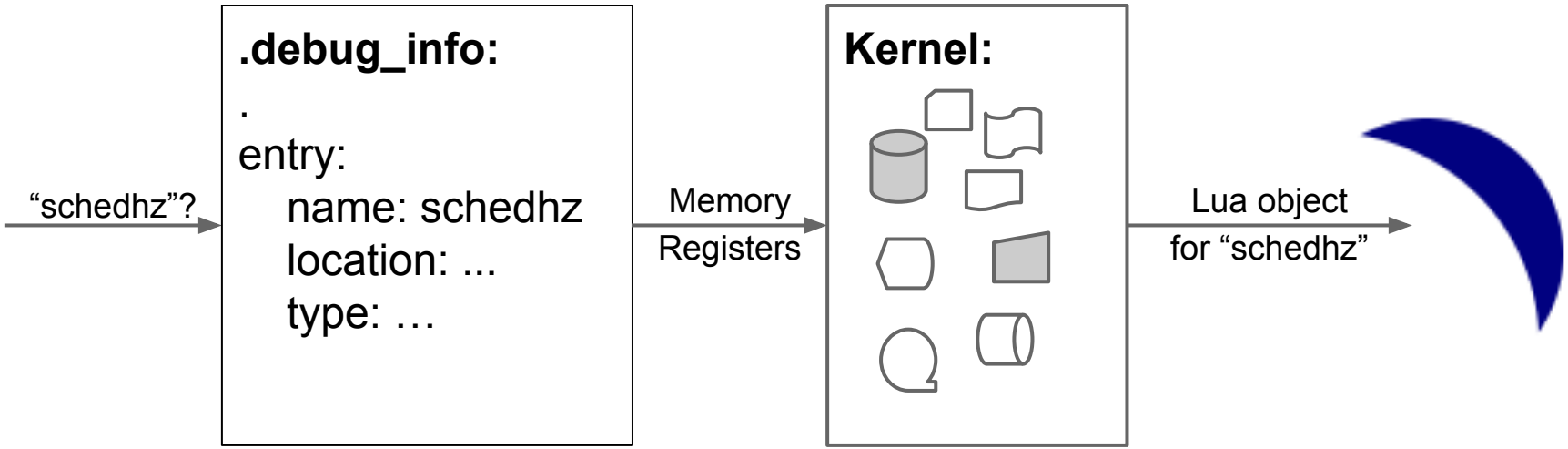
In “30” seconds or less

Lua ...

```
> function f(i)
    local hz = schedhz
    schedhz = i
    return hz
  end
> print(f(32))
```



… debugger


“schedhz”?
.debug_info:
.
entry:
name: schedhz
location: ...
type: …
Memory
Registers
Kernel:
Lua object
for “schedhz”

Why?

(Learn Lua of course?)

A problem ...

Subject: set a watchpoint programatically

To: tech-kern@netbsd; From: Emmanuel Dreyfus

I am tracking a memory corruption problem that pops up on a field of struct in a chained list. I would like to set a watchpoint on the field, but the problem is that the structures are added and removed from the list, and I cannot reproduce reliabily the bug.

Is there a way to programatically set a watchpoint, without having to do it by hand on ddb prompt? I would add it when a struct is added on the list, and delete it when a struct is removed.

http://mail-index.netbsd.org/tech-kern/2014/11/15/msg018003.html

That is ...

```
struct s {
  struct s *next;
  int f; // corrupted
};
```

```
void add(struct s *p) {
  …
}

void del (struct s *p) {
  …
}
```

Use ~~DDB~~ a debugger?

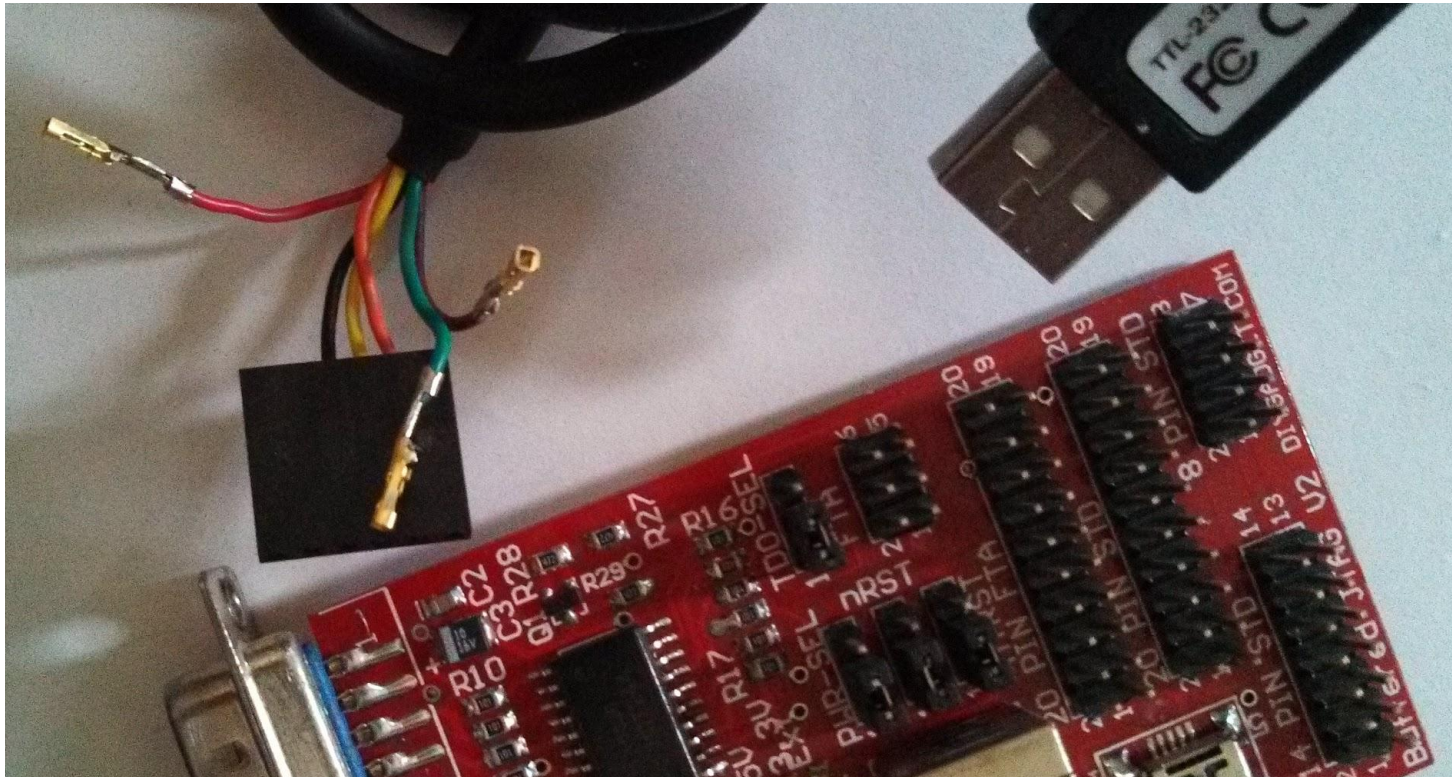
(Perhaps not what
Emmanuel Dreyfus had in mind)

~~1990~~ ~~1995~~ 2011: Try this ...

```
(gdb) break add if p->f == 1
(gdb) commands
silent
watch -location p->f
set $watching = $bpnum
continue
end
```

```
(gdb) break del if p->f == 1
(gdb) commands
silent
delete $watching
continue
end
```

… but avoid this



Use a Debugger Extension Language?

1999: Insight: GDB + TCL/TK

- ➔ Who extended who?
  Initially TCL/TK just invoked GDB’s interpreter
- ➔ fast track visual debugger tool
- ➔ not targeted at end users

2003: GDB/MI Interface

- written so that GDB could be embedded
- “designed” for extension languages
- shared code with Insight
- GDB’s official extension language shall be Guile

- more at 8

2005: Frysk (Java) + Jython

```
h = Manager.host
me = h.getSelf ()
h.requestCreateAttachedProc (["sleep","1000"])
child = me.getChildren()[0]
```

https://sourceware.org/ml/frysk/2005-q4/msg00012.html

GDB-MI based scripting

2008: GDB-MI + Python

```
class MyBreakpoint (gdb.Breakpoint):
    def stop (self):
        inf_val = gdb.parse_and_eval
("foo")
        if inf_val == 3:
            return True
        return False
```

(from GDB Manual)

2014: GDB-MI + Guile

```
(define (my-stop? bkpt)
  (let ((int-val
         (parse-and-eval "foo")))
    (value=? int-val 3)))
  (define bkpt
      (make-breakpoint "main.c:42"))
  (register-breakpoint! bkpt)
  (set-breakpoint-stop! bkpt my-stop?)
```

(from GDB Manual)

Use a Trace Tool?

systemtap

- shows more promise
  - event based syntax
  - “context variables” expressions like
    `$foo->bar` and `$foo[i]`
- has access to debug information
- ahead-of-time / static
- “context variables” have restrictions
- watchpoints seem limited to simple static symbols

```
probe kernel.data(“udp_table”).write           // ok
probe kernel.data("udp_table->hash").write // not
```

… and dtrace?

Would something like ...

```
> break(add, function()
  if p.f == 1 then
    w = watch(p.f)
  end
end)
```

```
> break(del, function()
  if p.f == 1 then
    delete(w)
  end
end)
```

… be possible?

Another problem ...

Subject: worrying differences in object code due to different build host!
To: tech-toolchain; From: Greg Woods

So in my quest to build a NEtBSD/i386 5.2_STABLE kernel that would boot on my Xen-4.5 amd64 servers, I've discovered there seems to be a quite substantial difference in the object code depending on the build host.

[...]

For context, this code is in the ibcs2_sys_getdents() function.

```
<  2595 16b3 8D8DDCFD   leal -548(%ebp),%ecx
>  2595 16b3 8D8DD0FD   leal -560(%ebp),%ecx
```

From: Andrew Cagney

Did the size or alignment of "struct ibcs2_dirent idb" change?

Now where did that come from?

< 2595 16b3 8D8DDCFD leal **-548**(%ebp),%ecx
> 2595 16b3 8D8DD0FD leal -560(%ebp),%ecx

That's a stack variable ... Rooting around the .debug_info, by luck, I find …

<2><31040e>: Abbrev Number: 73 (DW_TAG_variable)
<31040f> DW_AT_name : idb
<310416> DW_AT_type : <0x30ceb4>
<31041a> DW_AT_location : 3 byte block: 91 dc 7b (DW_OP_fbreg: **-548**)
<1><30ceb4>: Abbrev Number: 14 (DW_TAG_structure_type)
<30ceb5> DW_AT_name : (indirect string, offset: 0x214db): ibcs2_dirent

Programmatically?

```
def die_info_rec(die, indent_level='    '):
    print(indent_level + 'DIE tag=%s' % die.tag)
    child_indent = indent_level + '  '
    for child in die.iter_children():
        die_info_rec(child, child_indent)
```

https://github.com/eliben/pyelftools

… not really easier

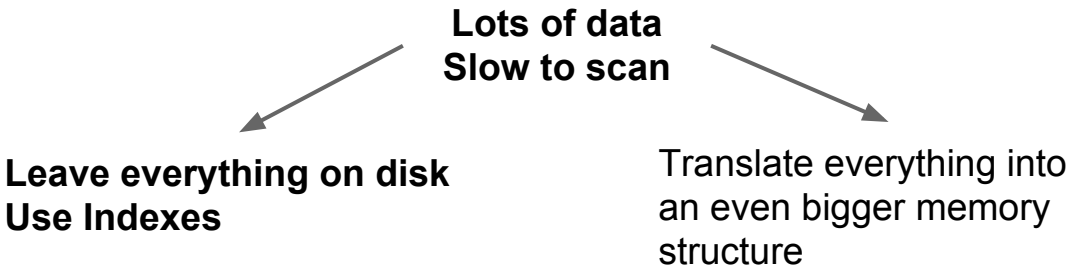
Would something like ...

```
local dwarf = getmetatable(ibcs2_sys_getdents)
for l in pairs(dwarf.variables) do
  if l.location == 548 then
    print(l)
  end
end
```

… be possible?

A problem trialing DWARF

- Improving speed
- Improving size
- Trial new proposals for DWARF
  - Need a test environment
  - Need a large C program



How?

(In theory …)

Compilation Units (.debug_info):

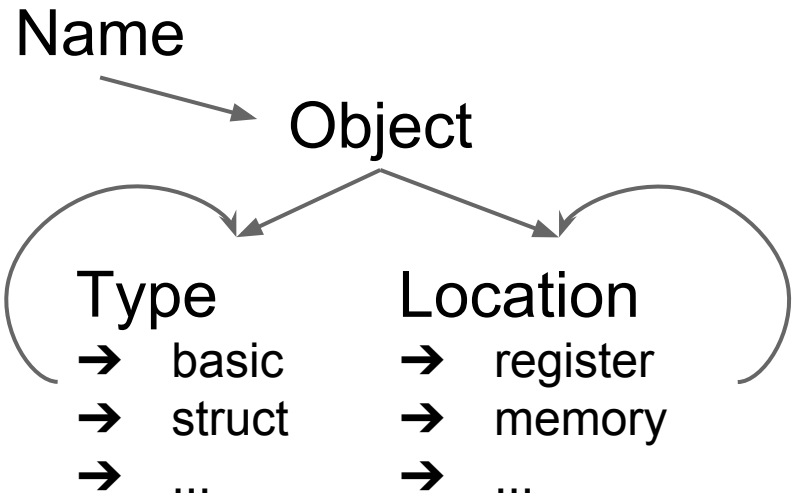
Compilation Unit (CU)

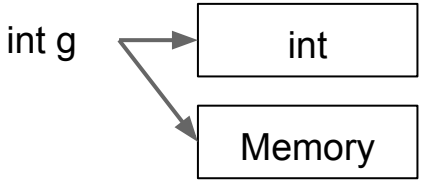
```
int g;
void f(int p)
{
  g = p * 2;
}
```

Debugging Information Entries (DIE)

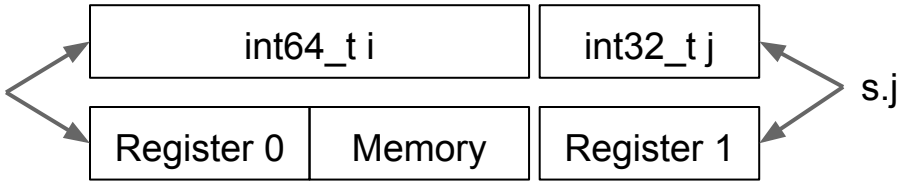
```
Compilation Unit @ offset 0x0:
  DW_TAG_subprogram <2d>:
      DW_AT_name: f
      DW_AT_low_pc: 0x1000
      DW_AT_frame_base: DW_OP_call_frame_cfa
      DW_TAG_formal_parameter:
          DW_AT_name: p
          DW_AT_type: <61>
          DW_AT_location: DW_OP_fbreg: -36
DW_TAG_base_type <61>:
      DW_AT_byte_size: 4
      DW_AT_encoding: signed
      DW_AT_name: int
DW_TAG_variable  <68>:
      DW_AT_name: g
      DW_AT_type: <0x61>
      DW_AT_location:  DW_OP_addr: 0x2000
```

DWARF objects






```
struct {
  int64_t i;  int32_t j
} s;
```



Primitive Lua primitives

Memory:

peek(addr) -> byte

poke(addr, byte)

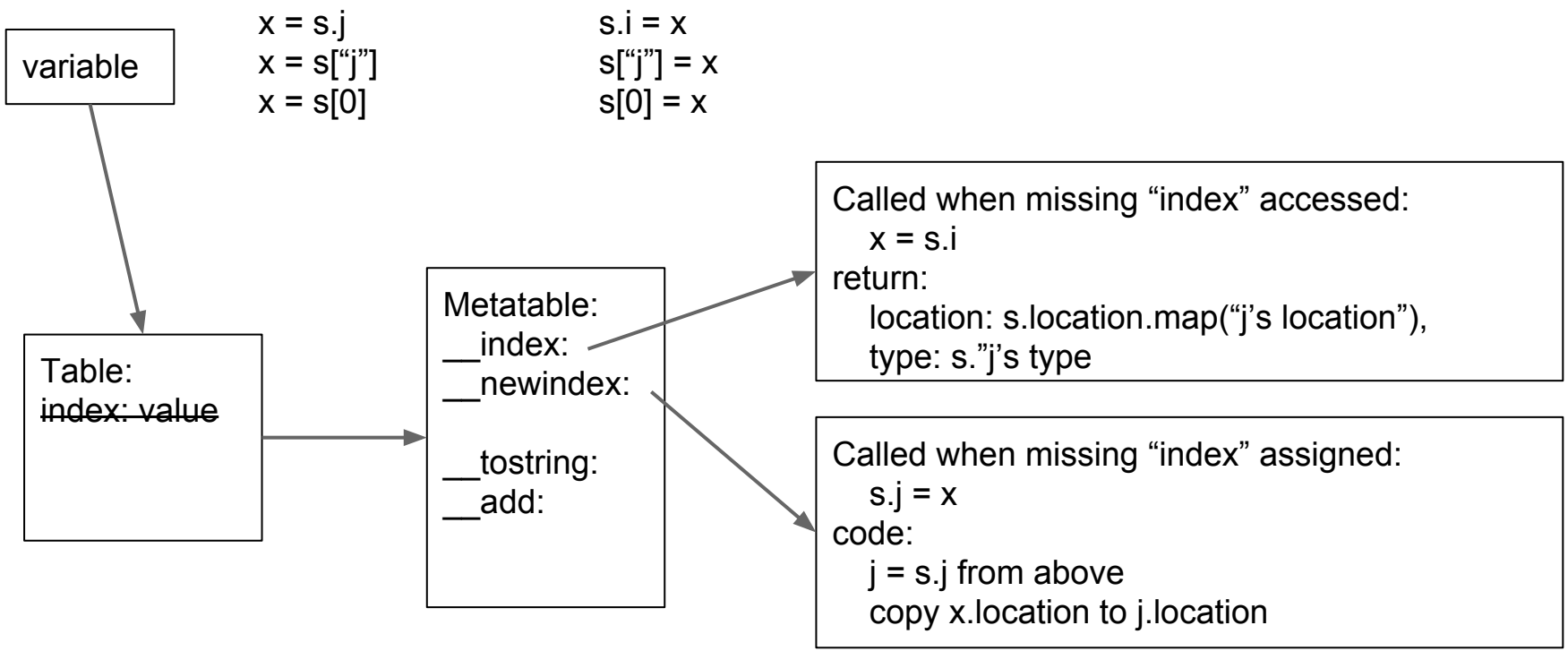
- implement “location” as an “array of bytes”

Registers:

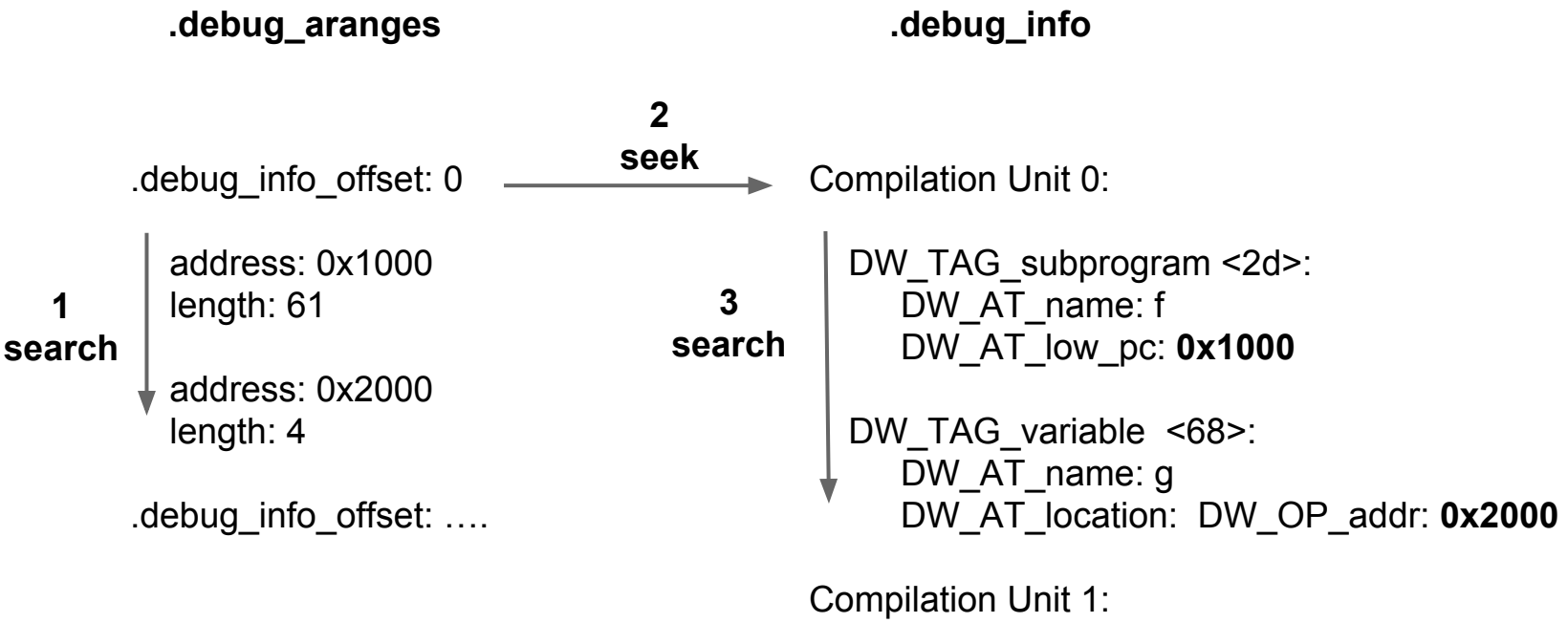
register(num) -> addr

- “num” from DWARF
- registers like memory
  peek(register(0))

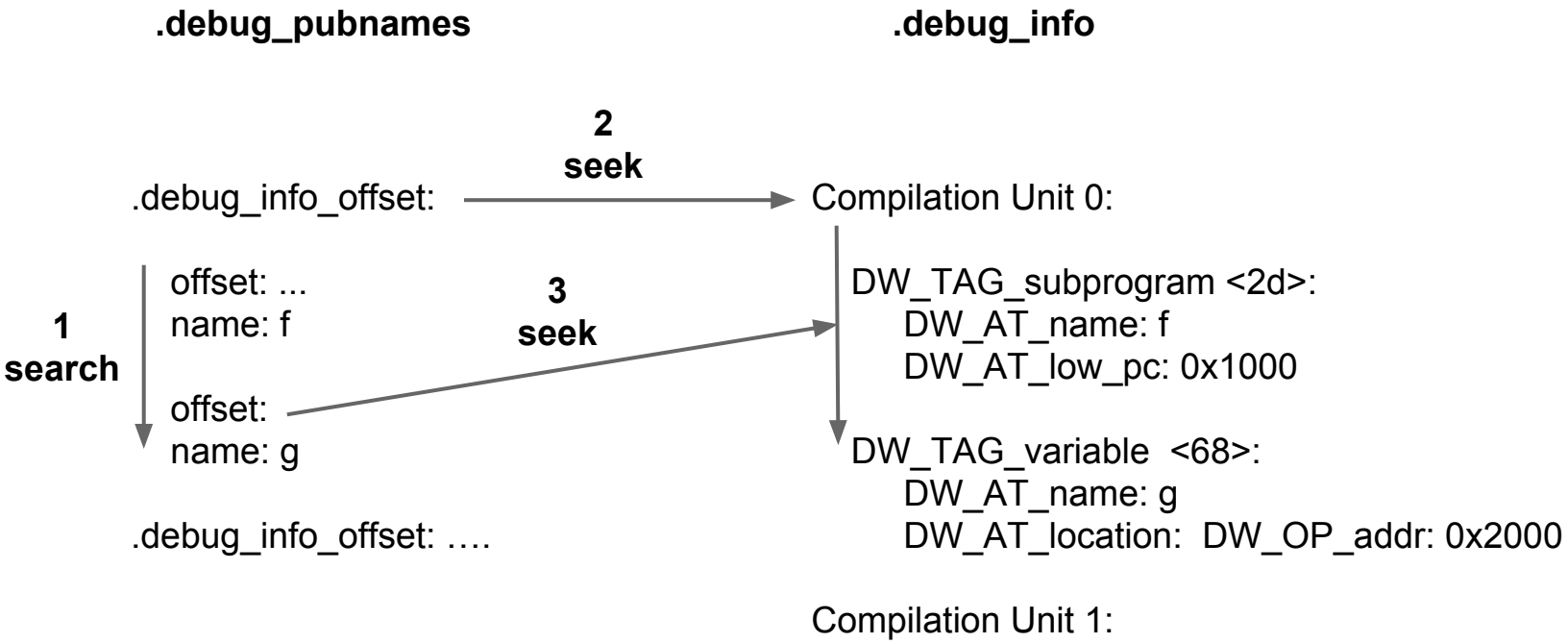
Using Lua tables (and metatables)



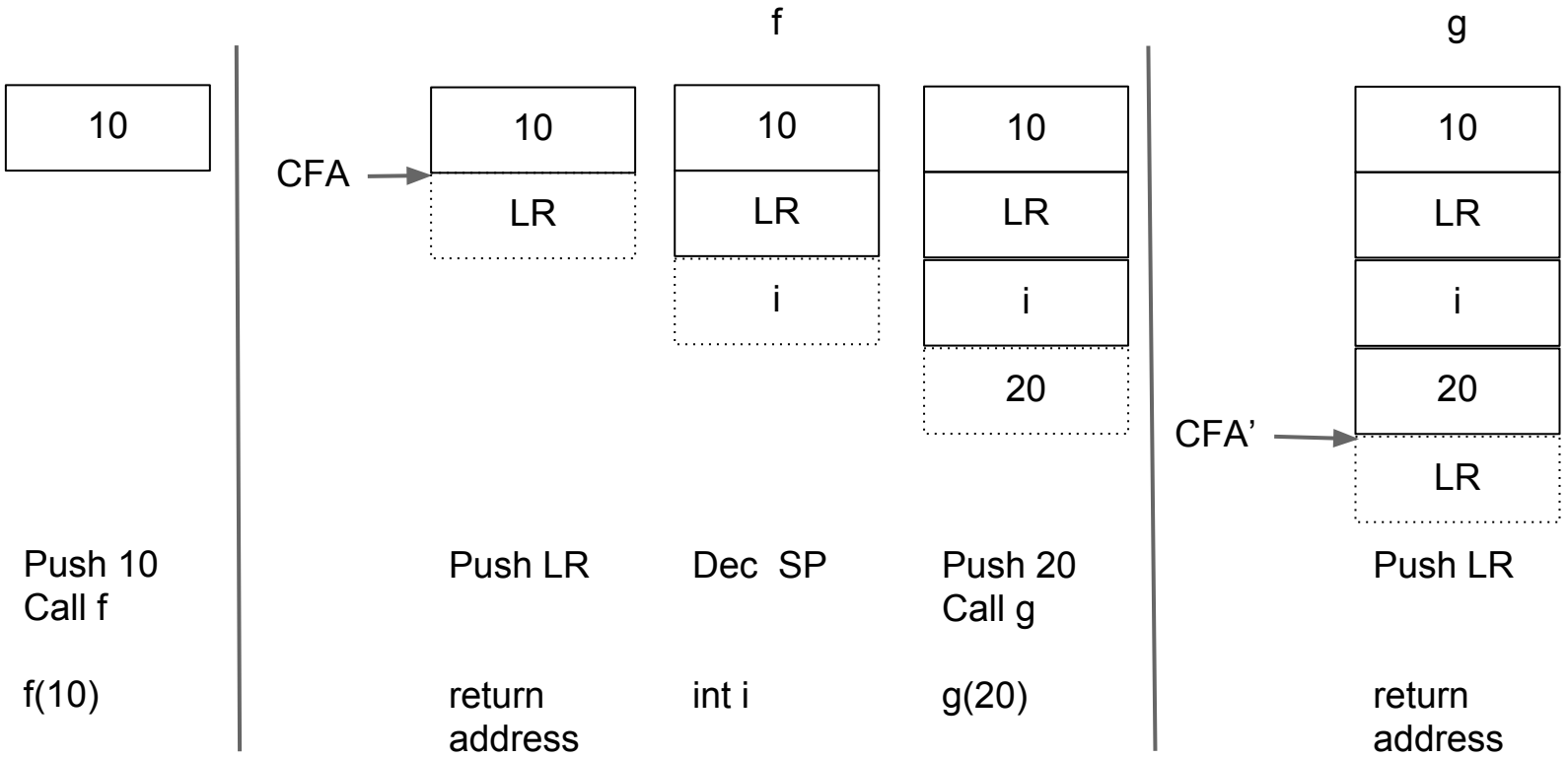
Address to CU (.debug_aranges)



Name to CU (.debug_pubnames)



Call Frame Address (.debug_frame)



Progress?

(Slow)

Add lua to kernel (files.ddb)

```
makeoptions ddb CPPFLAGS+="-I$S/..
/external/mit/lua/dist/src"
makeoptions ddb CPPFLAGS+="-I$S/sys"
makeoptions ddb CPPFLAGS+="-Wno-error=cast-qual"
makeoptions ddb CPPFLAGS+="-Wno-error=shadow"
file    ../external/mit/lua/dist/src/lapi.c ddb
file    ../external/mit/lua/dist/src/lcode.c ddb
...
```

➔ time for a lua kernel library?

Add lua to DDB

```
lua_State *L = lua_newstate(lua_alloc, NULL/*ud*/);   /* opens Lua */
luaL_openlibs(L);
while (1) {
    size_t i = db_readline3("lua", buf, sizeof(buf) - 1);
    int error = luaL_loadbuffer(L, buf, i, "line") || lua_pcall(L, 0, 0, 0);
    if (i <= 1) break;
    if (error) {
        printf("%s\n", lua_tostring(L, -1));
        lua_pop(L, 1);  /* pop error message from the stack */
    }
}
lua_close(L);
```

… find a bug

root device: ddb
Stopped in pid 0.1 (system) at netbsd:cpu_Debugger+0x4: bx r14
db> lua
Starting lua, enter an empty line to exit
lua> print(string.format(**"%d"**, 1000000))
100

-#define sprintf(s,fmt,...) snprintf(s, **sizeof(s)**, fmt, __VA_ARGS__)

```
            char *buff = luaL_prepbuffsize(&b, MAX_ITEM);
-           nb = sprintf(buff, form, n);
+           nb = snprintf(buff, MAX_ITEM, form, n);
```

Create netbsd.debug

objcopy --only-keep-debug netbsd.gdb netbsd.debug

.text	~2.3mb
.data	~0.3mb
netbsd.debug	18.6mb
Total	21.7mb

Yes, netbsd.debug is big!

.debug_info	11.7mb	describes program	functions, variables, types, scope, ...
.debug_abbrev	0.6mb	encoding information for .debug_info	
.debug_str	0.3mb	string table	names of variables, types, ...
.debug_aranges	0.1mb	map address to .debug_info	
.debug_line	1.0mb	map address to file/line	
.debug_pubnames	2.0mb	map name to .debug_info	needs -gpubnames - has problems
.debug_loc	2.1mb	object location lists	value in register, memory, both
.debug_ranges	0.2mb	instruction address ranges	instructions for block-scope; inline
.debug_frame	0.2mb	unwinding	needs -fno-unwind-tables (.eh_frame in .text)

Embed netbsd.debug in NetBSD ...

- like “makeoption COPY_SYMTAB=1”
  - #define DEBUG_SIZE $(wc -c < netbsd.debug)
  - 16 byte fudge as DWARF grows
  - char db_debug[DEBUG_SIZE+16] array
- like memory-disk device
  - mdsetimage ... netbsd netbsd.debug
- easy!?!

… find a bug

NetBSD/evbarm (EVBARM_BOARDTYPE) booting ...
panic: pmap_alloc_specials: no l2b for 0xc1000000

```
-#define KERNEL_VM_BASE      (KERNEL_BASE + 0x01000000)
+#define KERNEL_VM_BASE      (KERNEL_BASE + 0x02000000)
-#define KERNEL_VM_SIZE      0x0C000000
+#define KERNEL_VM_SIZE      0x0B000000
```

➢ still easy!

Add Lua access to netbsd.debug

```
lua> print(db_peek(db_debug_buf),db_peek
(db_debug_buf+1),db_peek(db_debug_buf+2),db_peek
(db_debug_buf+3))
127    69    76    70
```

Embed Lua source blob in kernel ...

- “Precompiled chunks are not portable …”
- see above for how to embed kernel.debug
- invent LAR format:

  { <name> NUL <file> NUL } *
- implement Lua’s “require” to load entries

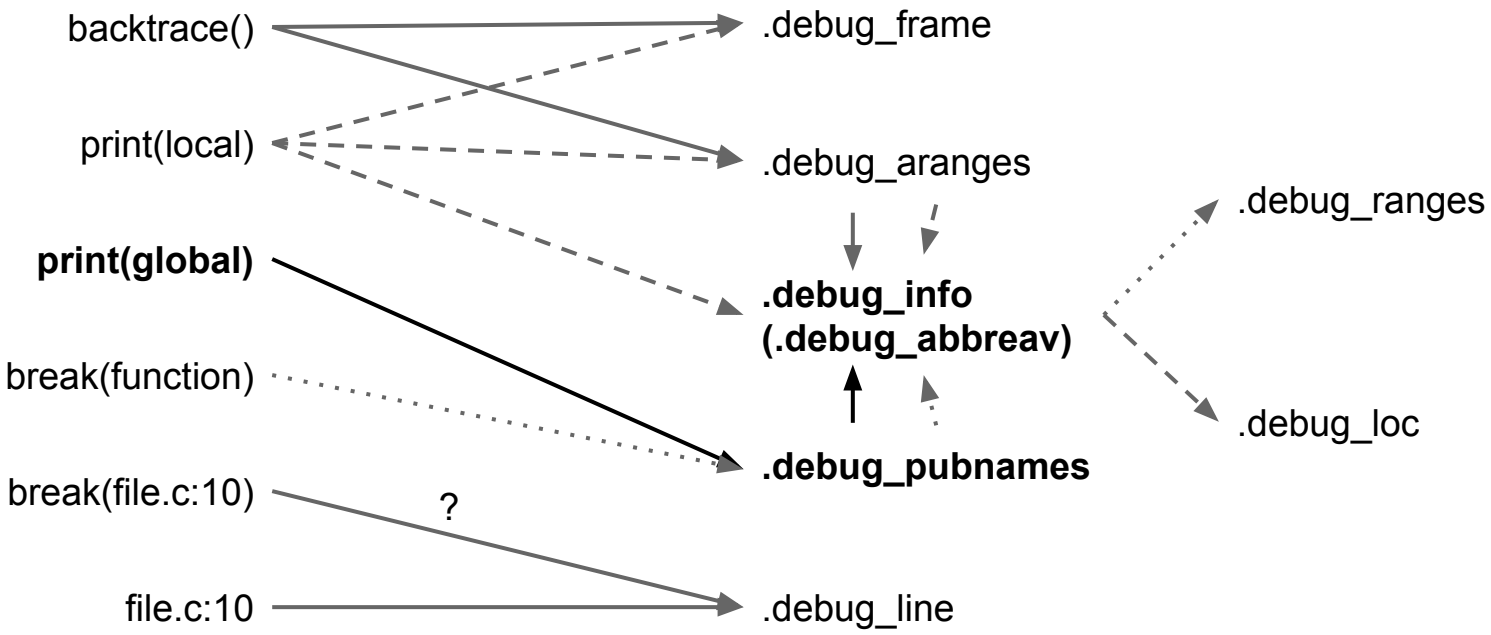
- better?
- file system (could include source)

Look for a DWARF library

DAs libdwarf	C	LGPL 2.1	Requires libelf Generate new DWARF?
elfutils	C	GPLv3	Has attitude
libunwind	C	X11	Not “remote only”
llvm	C++	Hybrid MIT/BSD	C, C++
inua	Java	GPLv2+Exception	(Mine) We’re desperate Not so slow

… sigh!

Plan “Plan B” ...


backtrace()
print(local)
print(global)
break(function)
break(file.c:10)
file.c:10
?
.debug_frame
.debug_aranges
.debug_info
(.debug_abbreav)
.debug_pubnames
.debug_line
.debug_ranges
.debug_loc

Hack ELF in lua

```
lua> e=require("ddb")()
class:    32-bit objects
data:    2's complement, little endian
type:    Executable file
machine:    Advanced RISC Machines ARM
version:    Current version
```

Parse pubnames (lookups)

```
lua> ddb=require("ddb")()
lua> m=ddb.dwarf.pubnames["main"]
main    main    2407119    42072
```

- “worst case” linear search
- 30 seconds on simulator
- hashtable proposed for DWARF

So?

So what’s been learn’t so far?

Lua has a dark side ...

```
> x = 1
> mt = {}
> setmetatable(x, mt)
table expected, got number
```

- no __tonumber
- must implement __add et.al.

… copy paste test

```
> print(x->y)
unexpected symbol near '>'
```

- ➢ fails copy/paste test
- ➢ can’t evaluate arbitrary expressions unchanged

… overloading “==”

> getmetatable(t).__eq == …

> if t == “string” then ...

- can’t overload “==” correctly
- “Lua will try a metamethod only when the values being compared are either both tables or both full userdata”
- needing “String(“string”)” would be silly
- hack “string”?

Action Items (or what is next)

- bypass DDB - dispatch events to Lua
- DWARF
- test
- more DWARF
- more tests
- still more DWARF …
- answer any questions

https://bitbucket.org/cagney/netbsd

References

- Dwarf Standards: http://dwarfstd.org/
- Lua: http://lua.org/
- Related NetBSD Lua/ddb discussion:
  - Alexander Nasonov: http://mail-index.netbsd.org/tech-kern/2013/10/19/msg015772.html
  - Marc Balmer http://mail-index.netbsd.org/tech-kern/2013/10/19/msg015773.html
- https://bitbucket.org/cagney/netbsd branch “debug”