Deet

A Simple and Extensible
Graphical Debugger

Jeffrey Korn and Dave Hanson
Princeton University
January 9, 1997

Motivation

Debuggers are notorious for being:

- **Hard to port**
- **Hard to use**
- **Hard to program**
- **Hard to modify**
- **Complex**

The Deet Approach

- **Write the debugger on top of a small, simple system API:**
  - Machine-independent
  - Distributed

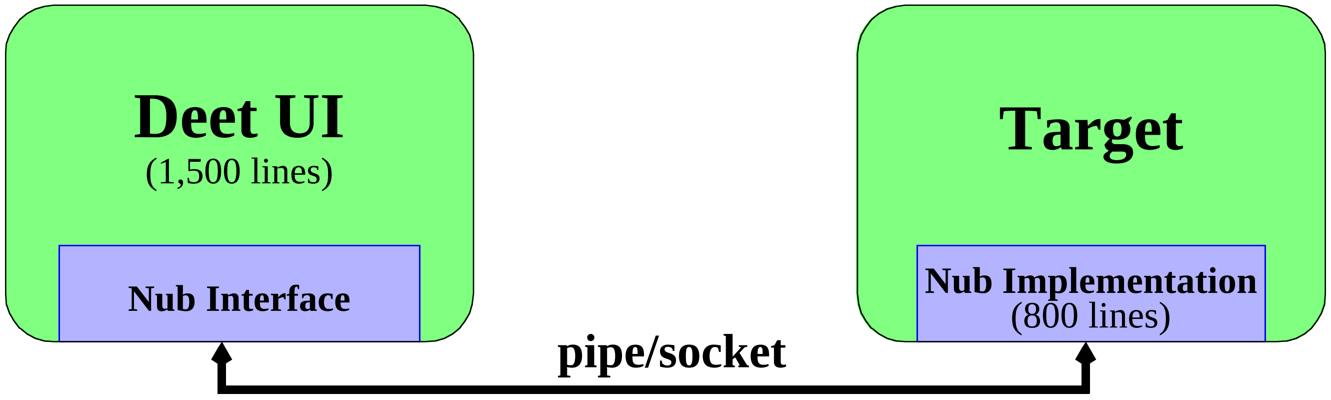
- **Use a suitable language to implement the debugger:**
  - Graphical
  - Programmable
  - Extensible

Result: Simplicity

Related Work

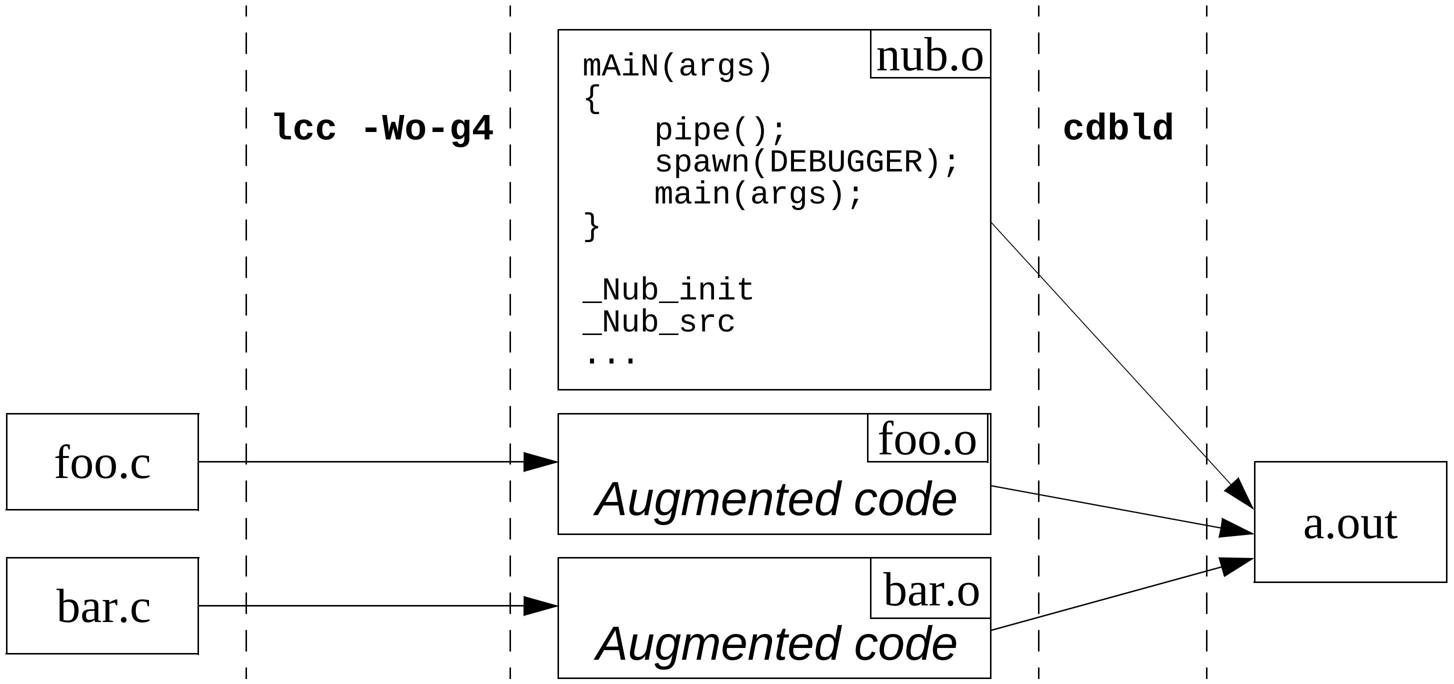
- **Machine Independence**
  - smld [LFP 90]
  - ldb [SIGPLAN 92]
- **Graphical Debuggers**
  - Blit Debugger, pi [USENIX 86]
  - ddd [SIGPLAN notices 95]
  - Microsoft Visual C++
- **Debugging Languages**
  - ups, Acid [USENIX 94]
  - Dalek [USENIX 90], duel [USENIX 93]
  - NeD [USENIX 92]
  - Solaris dbx

The Debugging Nub



- Interface between debugger and target
- Contains all dependencies
- Minimal functionality, small API
- Can have different implementations
- Allows debugger to be on different machine

A Machine Independent Nub



```
% DEBUGGER=cdb a.out
cdb> b 7
Sweep and send one of the following commands:
b test/wf.c:7.6
b test/wf.c:7.18
b test/lookup.c:7.2
cdb>
```

The Nub Interface

`_Nub_init`	Initialize nub.
`_Nub_set` `_Nub_remove`	Set and remove breakpoints.
`_Nub_src`	Walk through breakpoints with given pattern.
`_Nub_frame`	Information about stack.
`_Nub_fetch` `_Nub_store`	Manipulate memory in target.

- Symbol table implemented on top of nub interface (not specified by nub)

The Deet Language

- **Uses Tksh, a superset of Tcl**
- **Parses and interprets Tcl code or ksh scripts**
- **Written, used and programmed with Tksh**
- **Uses set of built-in nub commands:**

`deet_continue`	**Begin / resume execution**
`deet_breakpoint`	**Set and remove breakpoints.**
`deet_getval`	**Get value at given address.**
`deet_gettype`	**Get type information.**
`deet_frame`	**Get/Set current frame.**
`deet_sym`	**Lookup symbol.**

Why Debug With Tksh?

- **Good for interactive use**

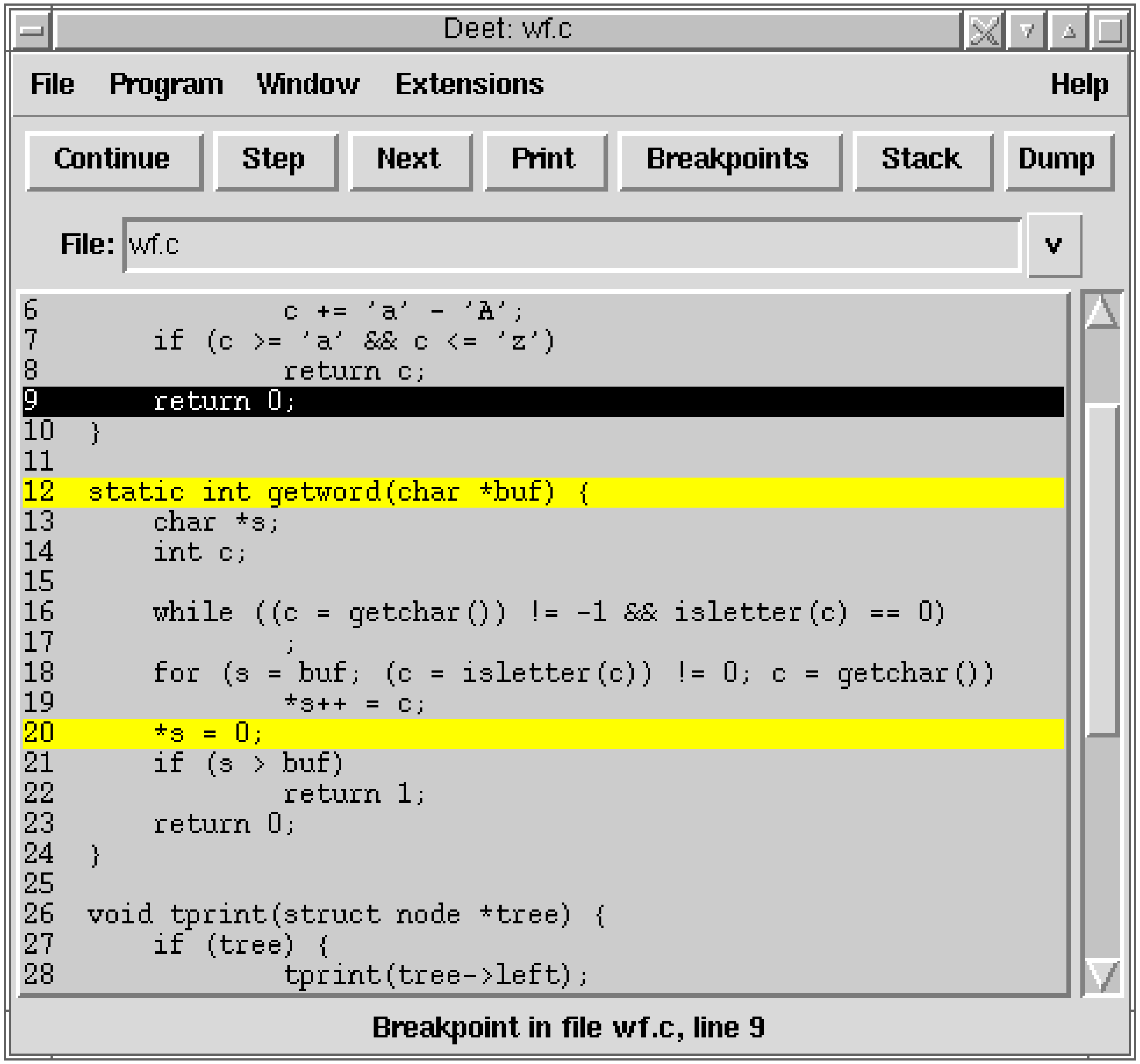
  Job control
  Command line editing
  Easy to work with files and processes

- **Backward compatibility**

  No need to learn a new language
  Conformance to standards (POSIX 1003.2 / ISO 9945-2)

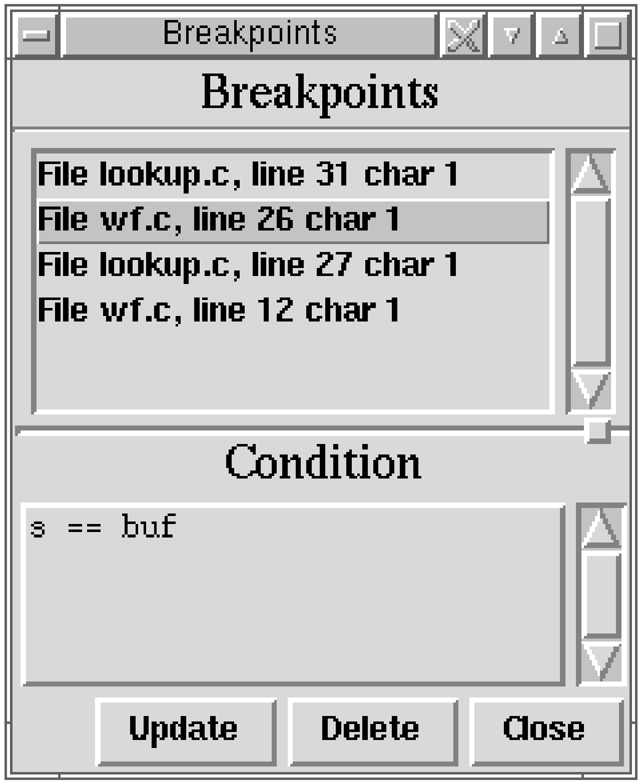
- **Ksh is a good programming language**

  Superset of Tcl, better syntax than Perl
  Good performance
  Language features



```
6                   c += 'a' - 'A';
7           if (c >= 'a' && c <= 'z')
8                   return c;
9           return 0;
10  }
11
12  static int getword(char *buf) {
13          char *s;
14          int c;
15
16          while ((c = getchar()) != -1 && isletter(c) == 0)
17                  ;
18          for (s = buf; (c = isletter(c)) != 0; c = getchar())
19                  *s++ = c;
20          *s = 0;
21          if (s > buf)
22                  return 1;
23          return 0;
24  }
25
26  void tprint(struct node *tree) {
27          if (tree) {
28                  tprint(tree->left);
```

Breakpoint in file wf.c, line 9

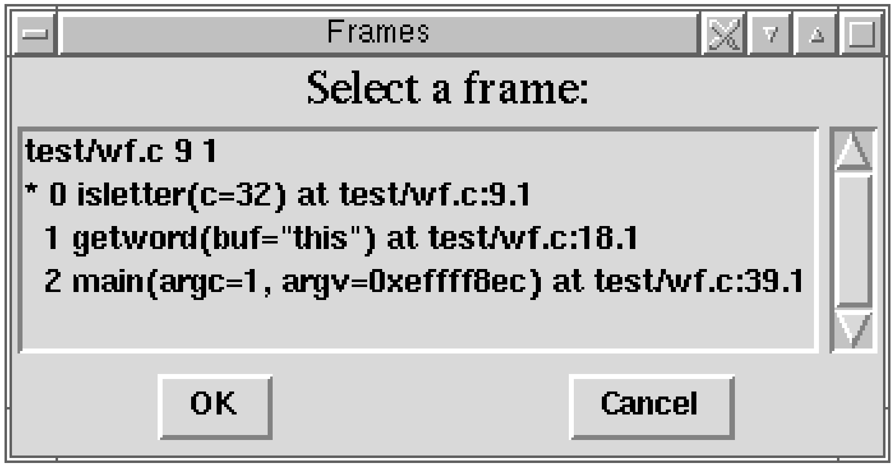


Breakpoints

```
File lookup.c, line 31 char 1
File wf.c, line 26 char 1
File lookup.c, line 27 char 1
File wf.c, line 12 char 1
```

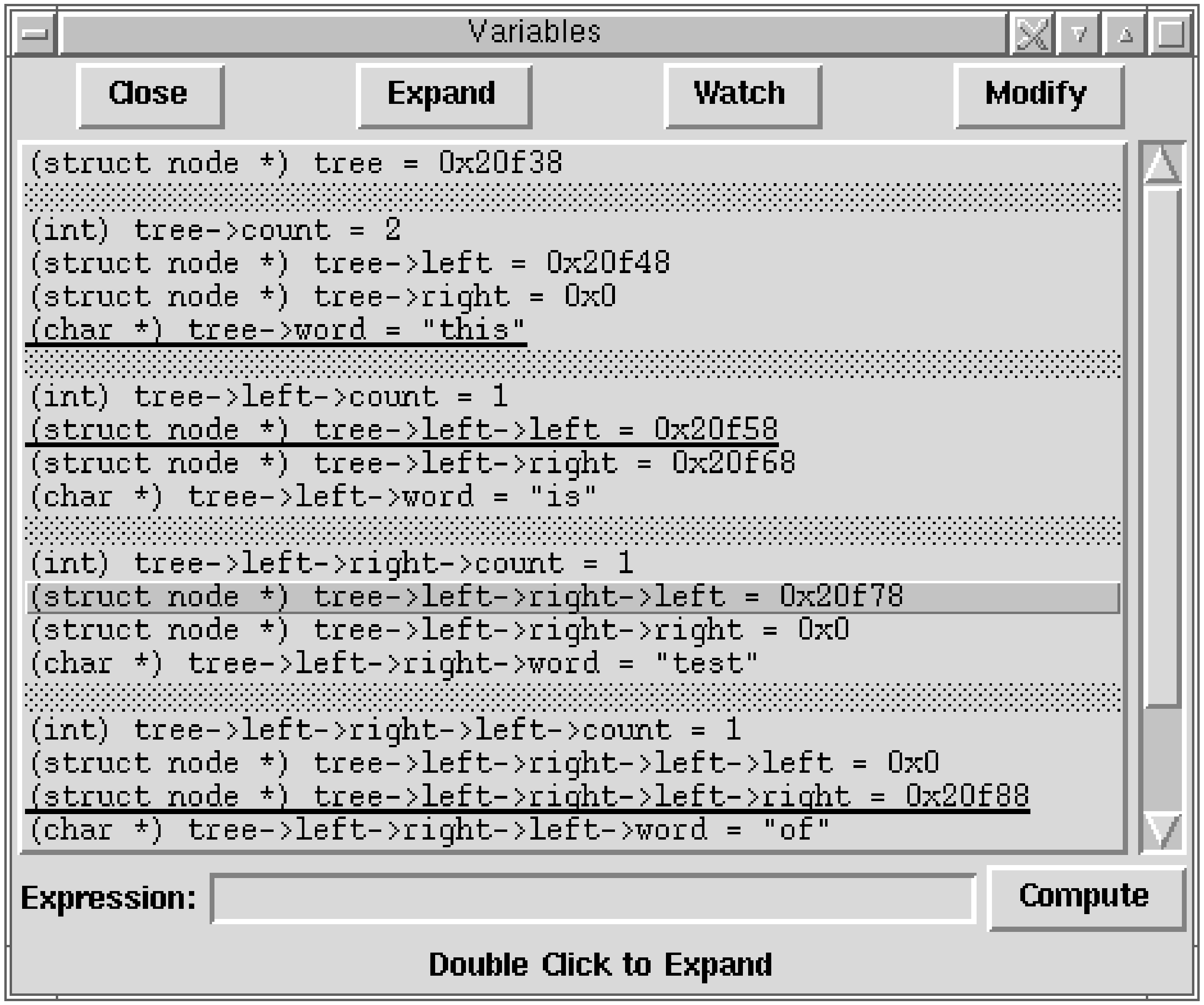
Condition

```
s == buf
```



Select a frame:

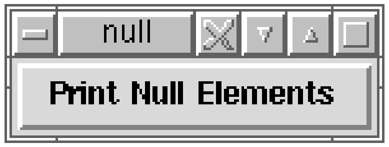
```
test/wf.c 9 1
* 0 isletter(c=32) at test/wf.c:9.1
  1 getword(buf="this") at test/wf.c:18.1
  2 main(argc=1, argv=0xeffff8ec) at test/wf.c:39.1
```



```
(struct node *) tree = 0x20f38

(int) tree->count = 2
(struct node *) tree->left = 0x20f48
(struct node *) tree->right = 0x0
(char *) tree->word = "this"

(int) tree->left->count = 1
(struct node *) tree->left->left = 0x20f58
(struct node *) tree->left->right = 0x20f68
(char *) tree->left->word = "is"

(int) tree->left->right->count = 1
(struct node *) tree->left->right->left = 0x20f78
(struct node *) tree->left->right->right = 0x0
(char *) tree->left->right->word = "test"

(int) tree->left->right->left->count = 1
(struct node *) tree->left->right->left->left = 0x0
(struct node *) tree->left->right->left->right = 0x20f88
(char *) tree->left->right->left->word = "of"
```

Programming Deet

```
function nullElements
{
    typeset arr=$1
    integer s=$(arraySize $arr)
    for (( i=0 ; i < s ; i++ ))
    do
        if [[ $(var "$arr[$i]") == 0x0 ]]
        then
            print "Element $arr[$i] null"
        fi
    done
}

toplevel .null

pack $(button .null.b -text "Print Null Elements" \
    -command "nullElements hashtable")
```



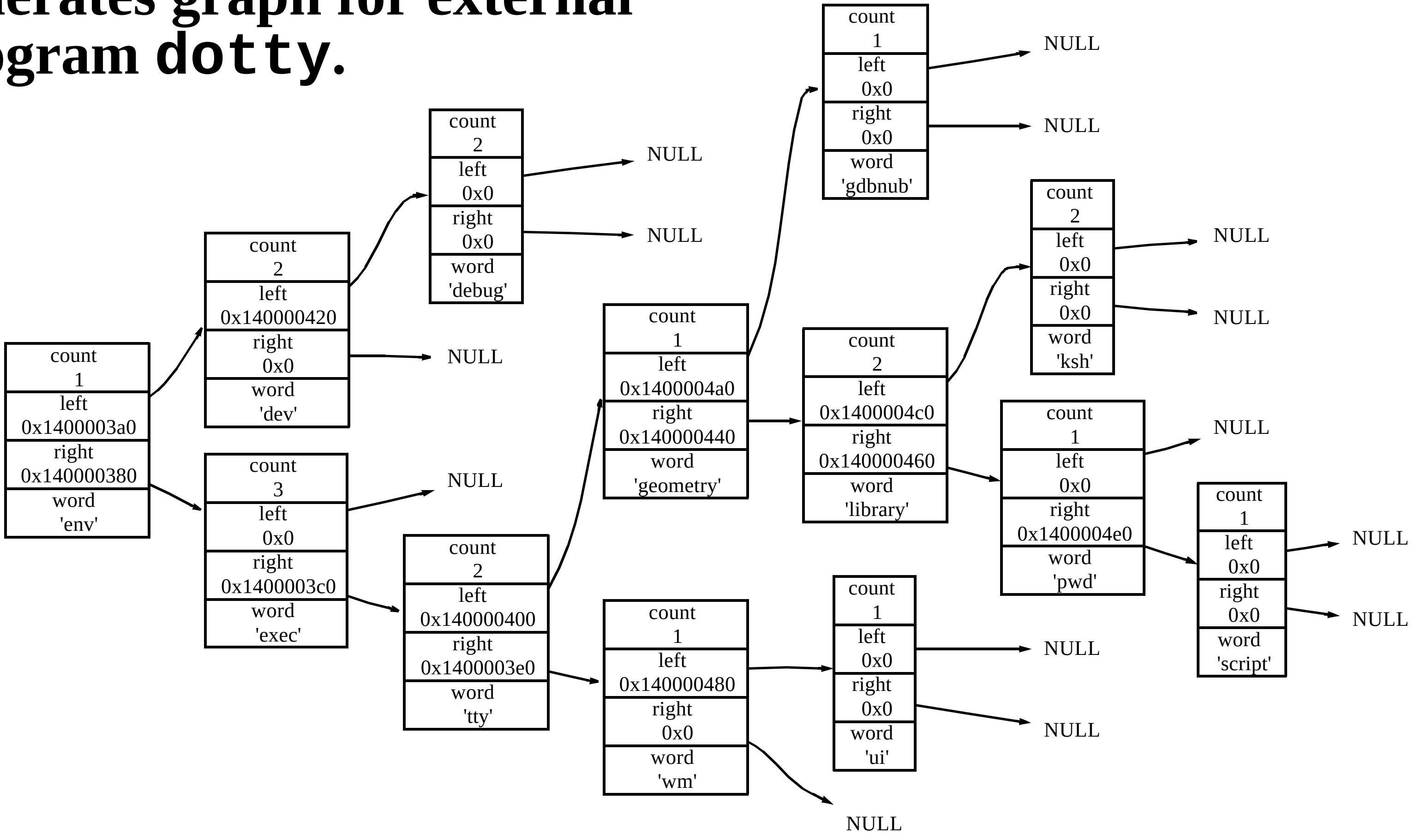
Example: Implementation of `where`

- **Complex debugger function written on top of nub interface:**

```
function where
{
    integer i=0
    while deet_frame $i 2> /dev/null
    do
        set -A frame  $(deet_frame)
        set -A params $(deet_sym -params)
        ... # Print frame (< 30 lines)
        ((i++))
    done
}
```

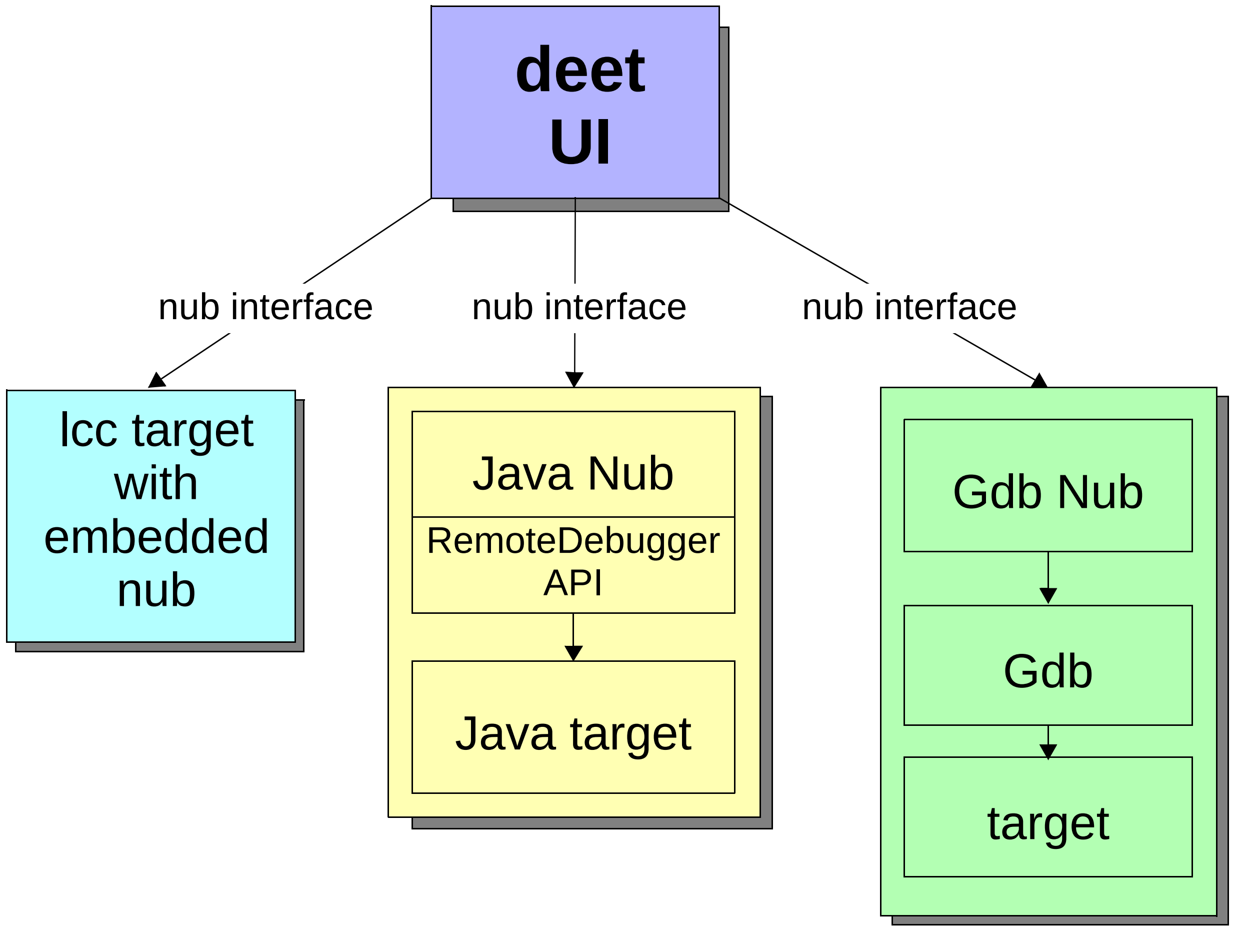
Example: Drawing Structures

- **Tksh function** `drawval` **(< 60 lines) generates graph for external program** **`dotty`**.

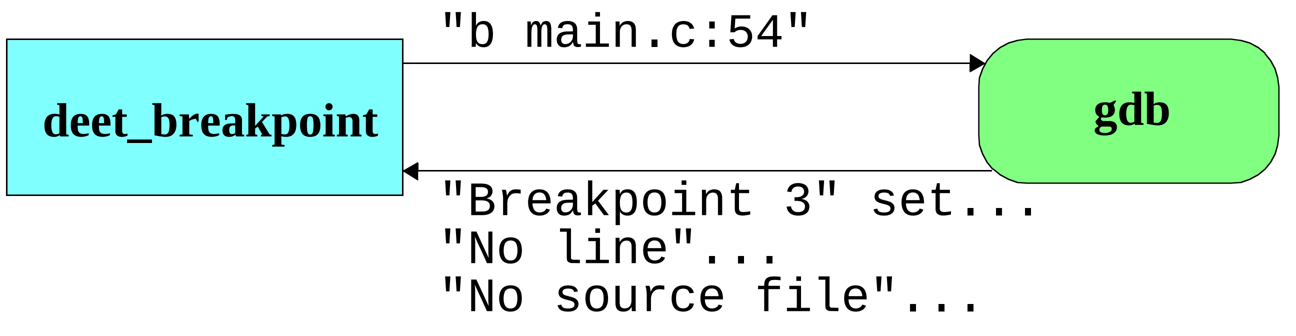


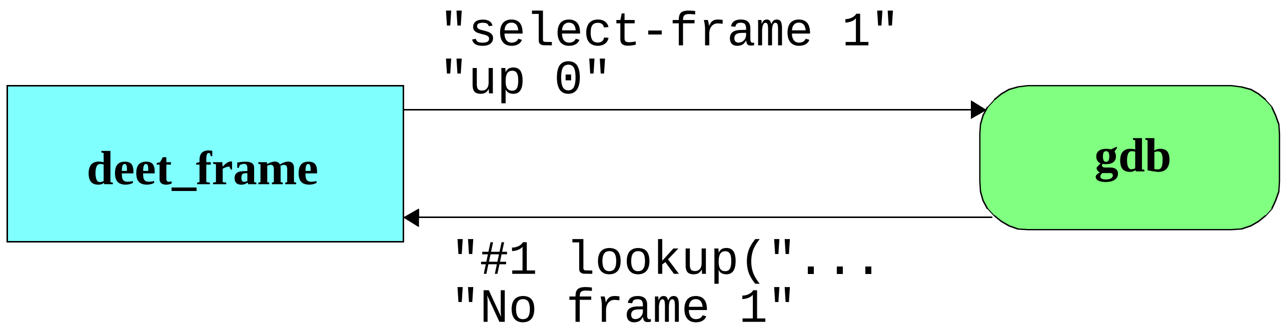
count 1 left 0x1400003a0 right 0x140000380 word 'env'
count 2 left 0x140000420 right 0x0 word 'dev'
count 3 left 0x0 right 0x1400003c0 word 'exec'
count 2 left 0x0 right 0x0 word 'debug'
count 2 left 0x140000400 right 0x1400003e0 word 'tty'
count 1 left 0x1400004a0 right 0x140000440 word 'geometry'
count 1 left 0x140000480 right 0x0 word 'wm'
count 1 left 0x0 right 0x0 word 'gdbnub'
count 2 left 0x1400004c0 right 0x140000460 word 'library'
count 1 left 0x0 right 0x0 word 'ui'
count 2 left 0x0 right 0x0 word 'ksh'
count 1 left 0x0 right 0x1400004e0 word 'pwd'
count 1 left 0x0 right 0x0 word 'script'
NULL

Piece-parts Design

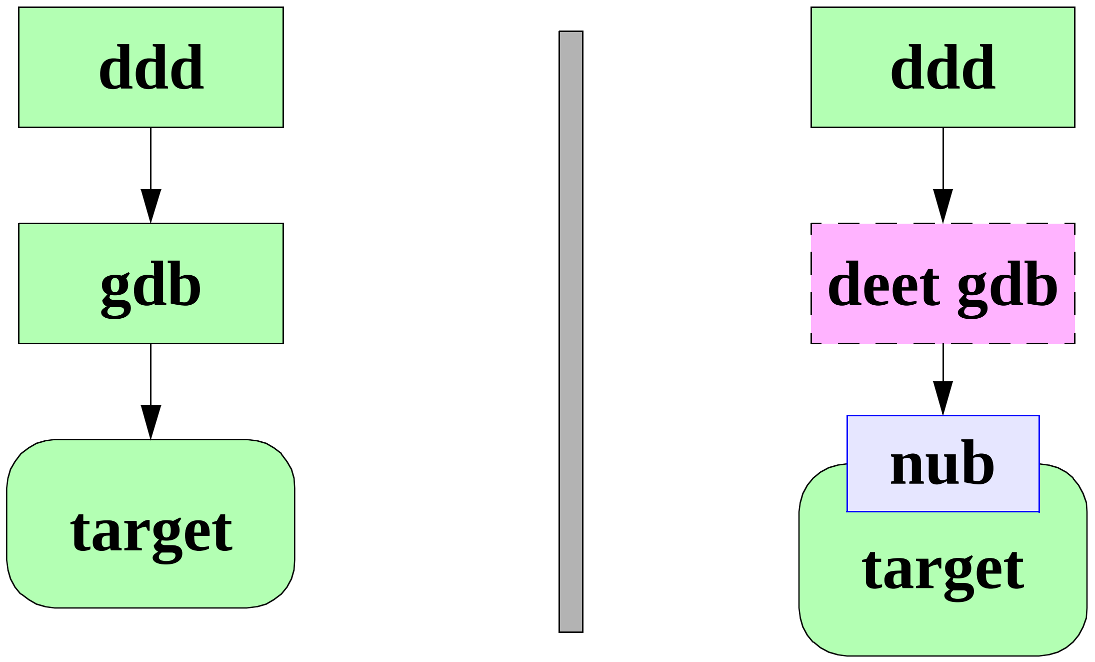


Gdb Nub Implementation





Implementation of gdb UI



- **Implements enough of gdb to support ddd**

```
(gdb) frame
#0  lookup (word=0x11ffff8e0 "a", p=0x140000010) at test/lookup.c:15
```

The Bad News...

- Deet cannot support:
    - Stepping through assembly
    - Hardware data watchpoints
- Deet can but currently does not support:
    - Interrupting the target
    - Debugging already running target
    - Calling target functions
    - Debugging core files
    - Signal handling
    - Threads

The Good News...

- **Demonstrates feasibility of nub interface**
- **Uses familiar high level debugging language**
- **Provides an extensible user interface**
- **Makes use of existing external tools**
- **Achieves simplicity**
  - Deet nub: 800 lines of C
  - Deet UI: 1,500 lines of Tksh

  - Gdb: 150,000 lines of C (47,000 machine dependent)
  - DDD: 90,000 lines of C++

Future Work

- **Support missing features**
- **Performance evaluation**
- **Expression evaluation**
- **Additional nubs:**
  - Other languages
  - Native object files (e.g. ELF)
  - Microsoft Debug API

http://www.cs.princeton.edu/~jlk/deet