Breakpoint Shenanigans (bps)

Kevin Sheldrake
rtfc.org.uk

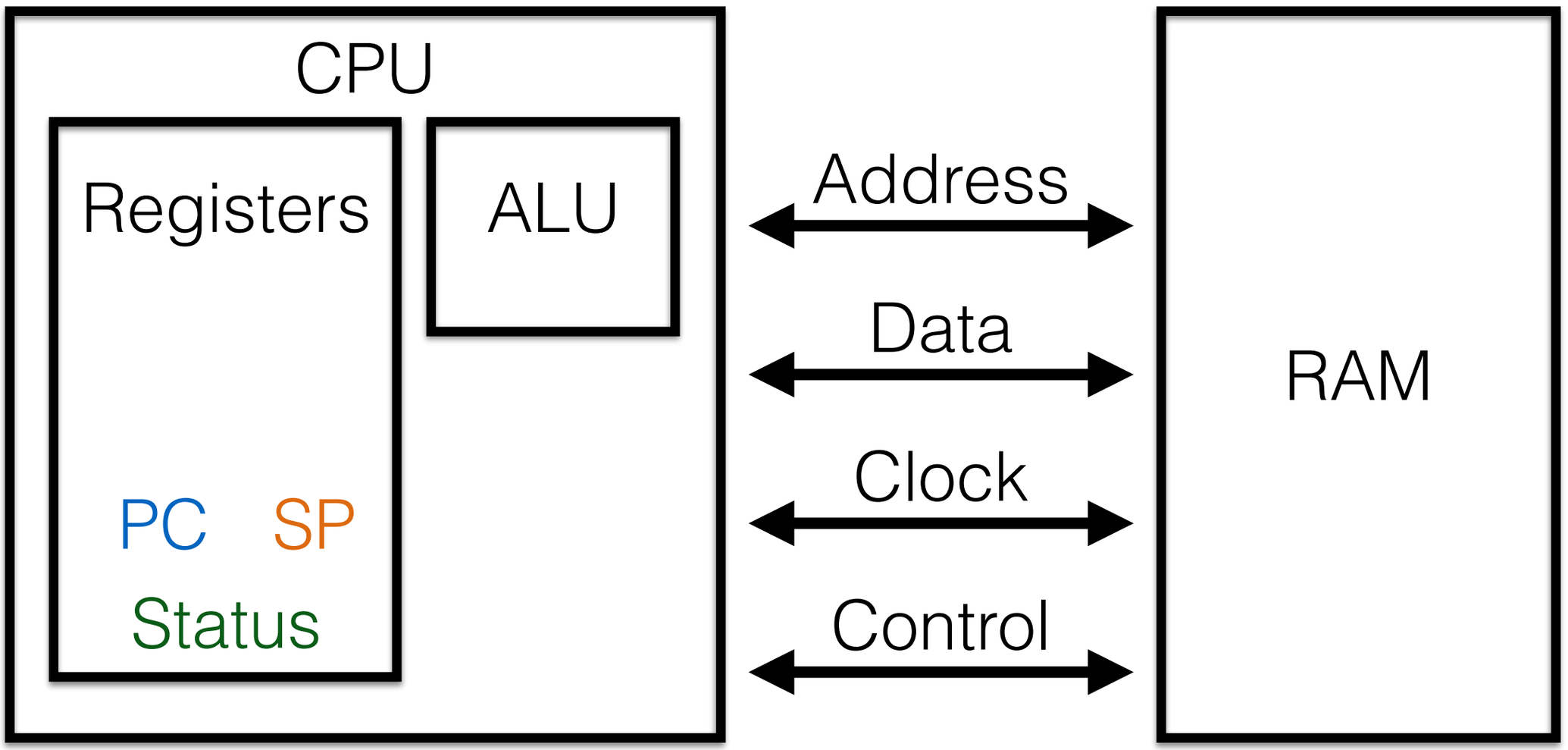
WTF?

- bps is a tool to set and manipulate breakpoints.
- It is essentially a debugger based on the linux ptrace() function/framework (like gdb).
- Unlike gdb, it is non-interactive so it can keep up with time-critical processes.
- bps is controlled from command-line arguments and, in the future, a configuration file.
- Unlike writing your own code, bps already works (subject to terms and conditions, ymmv, /home is at risk, etc).

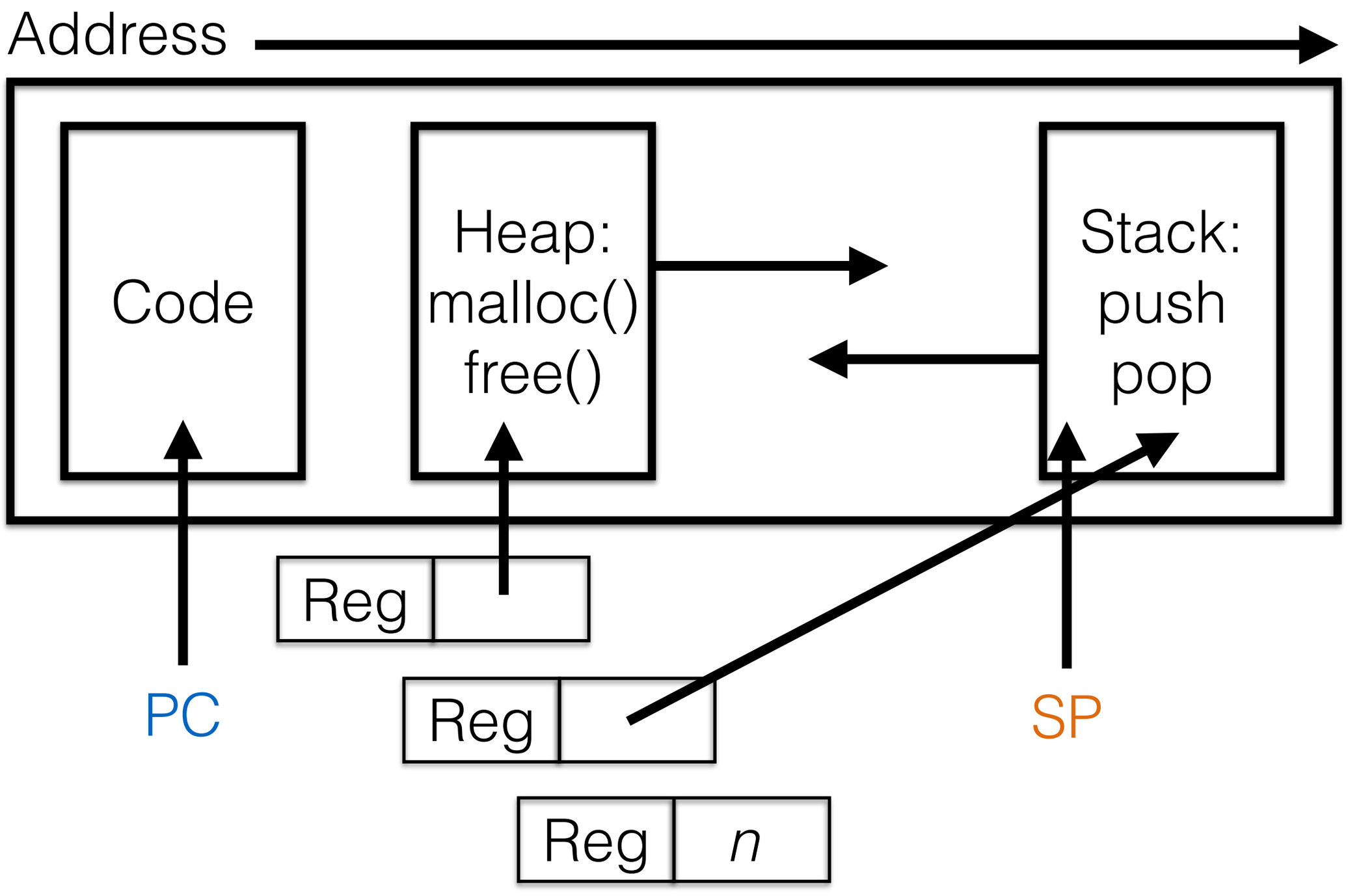
What's a breakpoint?

- To understand breakpoints, we really need to understand programming.
- To understand programming, we really need to understand assembler.
- To understand assembler, we really need to understand machine code and processor architectures.
- (sorry)

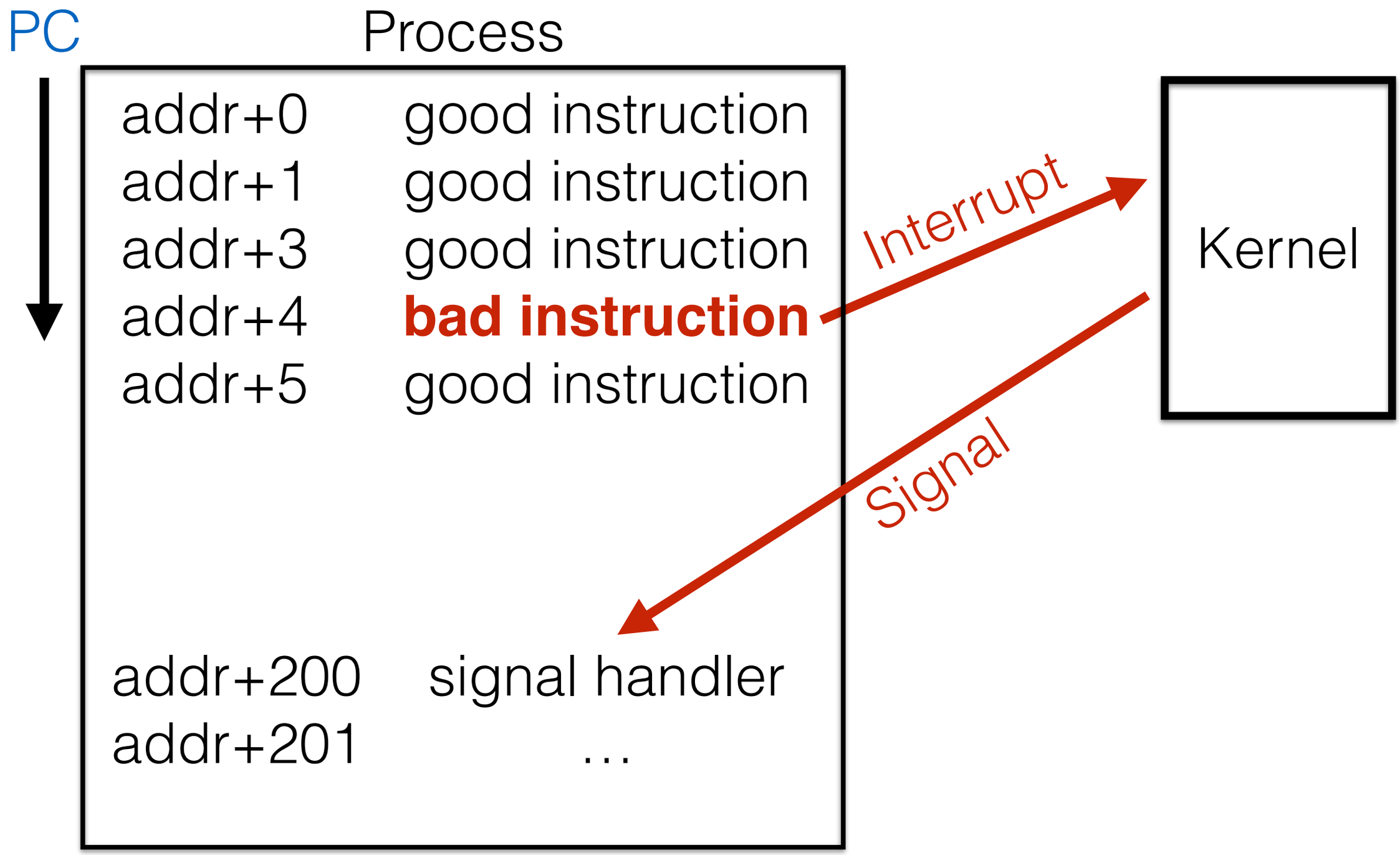
Processors


CPU
Registers
ALU
PC
SP
Status
Address
Data
Clock
Control
RAM

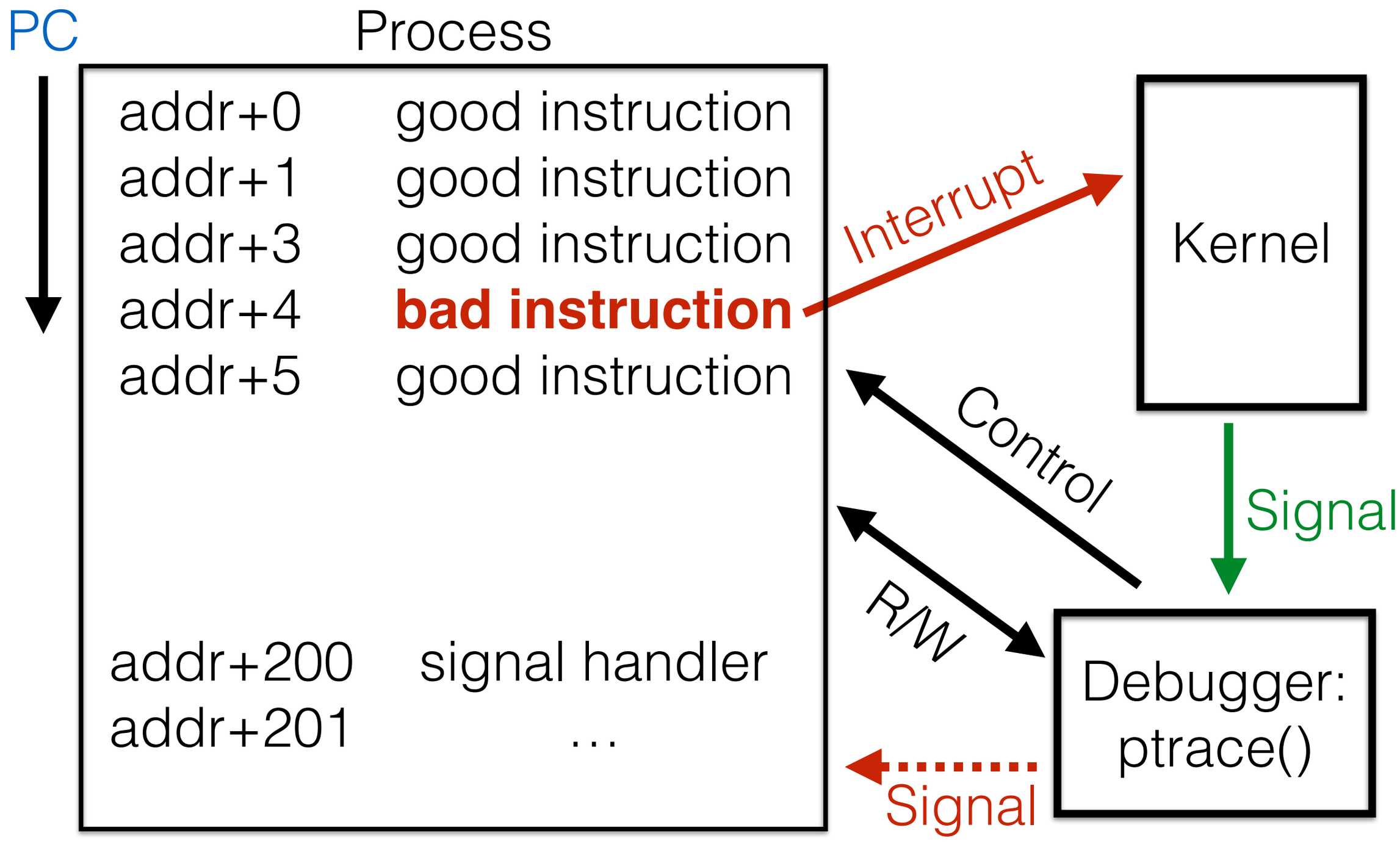
Processes


Address
Code
Heap:
malloc()
free()
Stack:
push
pop
Reg
Reg
Reg
n
PC
SP

Exceptions


PC
Process
addr+0 good instruction
addr+1 good instruction
addr+3 good instruction
addr+4 bad instruction
addr+5 good instruction
addr+200 signal handler
addr+201 …
Interrupt
Signal
Kernel

Enter ptrace()



Bad instructions

- TRAP:
  - 0xCC on x86. (PC := location of TRAP opcode + 1)
  - Causes SIGTRAP (5).
- Illegal opcode:
  - 0xE7FFFFFF on ARM.
  - 0xDEFF on Thumb.
  - Choose your own from the processor spec.
  - Causes SIGILL (4).

ptrace()

```
#include <sys/ptrace.h>
#include <sys/wait.h>
#include <sys/user.h>
…
child = fork();
if (child == 0) {
    ptrace(PTRACE_TRACEME, 0, NULL, NULL);
    execve(cmd, args, env);
} else if (child > 0) {
    exe = waitpid(-1, &status, __WALL);
    …
    ptrace(PTRACE_CONT, exe, NULL, signal);
    exe = waitpid(-1, &status, __WALL);
    ptrace(PTRACE_GETREGS, exe, NULL, &regs);
```

Breakpoints

- fork()
- child - ptrace(PTRACE_TRACEME) && execve()
- waitpid() receives SIGTRAP from child.
- breakopcode = (breakaddr).
- (breakaddr) = TRAP/ILL opcode.
- ptrace(PTRACE_CONT). Child runs until breakpoint.
- waitpid() receives SIGTRAP/SIGILL.
- (breakaddr) = breakopcode.

Breakpoints /cont

- Interact with process - ptrace(PTRACE_{GET|SET}REGS), ptrace(PTRACE_{PEEK|POKE}TEXT).
- Single step - ptrace(PTRACE_SINGLESTEP) or your own implementation.
- waitpid() receives SIGTRAP/SIGILL.
- (breakaddr) = TRAP/ILL opcode.
- ptrace(PTRACE_CONT). Child runs until breakpoint.
- waitpid() receives SIGTRAP/SIGILL.
- …

bps

- CLI debugger, non-interactive. Command line specifies:
  - Executable to run, with arguments.
  - Breakpoint function names or addresses.
  - Registers and memory locations to display on traps.
  - Breakpoints to enable/disable on traps.
  - Number of times each breakpoint can fire.
  - An optional initial breakpoint on which to set up the specified breakpoints.

Example

```
$ test/hello5
calling newfn
This is newfn: 5
after newfn
strfn, msg( 0x8048813 )='hello', msg2( 0x804880b )='goodbye'
output( 0x8552008 ) = 'hello - goodbye'
strfn returned 'hello - goodbye'
hello world

$ ./bps.0.3 -f strfn -R esp:w:16 -R esp:S:4 -R esp:S:8 -- test/hello5
calling newfn
This is newfn: 5
after newfn
breakpoint 1: strfn (0x804859f)
   register pointers:
      esp+0x0:
      00000000  0804868f 08048813  0804880b b754ebe0
      00000010  0804873b 00000001  bfc49fd4 00000005
      00000020  08048713 bfc49f40  b76d4000 00000000
      00000030  b7537e5e 00000000  08048420 00000000
      (esp+0x4) hello
      (esp+0x8) goodbye
strfn, msg( 0x8048813 )='hello', msg2( 0x804880b )='goodbye'
output( 0x837d008 ) = 'hello - goodbye'
strfn returned 'hello - goodbye'
hello world
```

Crazy successes

- Multi-thread, multi-process executables work well.
- Own implementation of arm_singlestep() appears to work correctly for ARM and Thumb modes on ARMv4-ARMv6 (ARM7-ARM11).
- Compiles on 32bit x86 and ARMv6.
- Learned a lot about ptrace() and ARM assembler.

Immediate plans

- Port to MIPS32 little-endian (and maybe big-endian).
- Permit registers to be used as parameters to buffer specs.
- Tainting of registers and buffers.
- Conditional breakpoints.
- Configuration file to supply options.
- Option to run executable as different user/group.
- Produce libbps and app template for more exotic uses.
- Port to ARMv7 (Thumb2) and x86 64bit.

Tips

- Not everything is in the ptrace() manual page.
- No PTRACE_SINGLESTEP on ARM or MIPS.
- No TRAP instruction on ARM - use illegal opcode instead.
- Use Raspberry Pi as dev platform for ARMv6.
- Use Creator CI20 as dev platform for MIPS32.
- Forking and threading expands the problem.
- Porting is an interesting challenge.

Usage

```
bps - Breakpoint Shenanigans - v0.3 K Sheldrake

Automatically display interesting info on breakpoints.

bps -l exe                                          - list functions in exe
bps <options> -- exe [arg1 [arg2 ...]]              - run exe with breakpoints

    -z/-Z                                           - enable/disable copy of breakpoints on fork
                                                      (initial configuration)
    -y/-Y                                           - ditto for new threads
    -G                                              - breakpoint enable/disable affects all processes
                                                      (default is to only affect process trapped on)
    -b init break fn | -B init break address        - optionally specify an initial breakpoint
                                                      at which to enable the real breakpoints
    -T                                              - initial breakpoint is thumb (arm only)
    -I                                              - establish breakpoints on all processes on initial
                                                      breakpoint
    -f function name | -F address                   - specify a breakpoint
     -t                                              - specify thumb rather than arm (arm only)
     -r register                                     - register to display
     -R register:format[:size][:offset]              - register buffer to display
     -A address:format[:size][:offset]               - address buffer to display
     -c count                                        - number of times to trap
     -D                                              - disable breakpoint on launch
     -k                                              - kill exe after count traps
     -e breakpoint number                            - breakpoint to enable after count traps
     -d breakpoint number                            - breakpoint to disable after count traps
     -z/-Z                                           - enable/disable copy of breakpoints on
                                                       following forks
     -y/-Y                                           - ditto for new threads
     -g                                              - reverse effect of -G for following enable/disable
                                                       options for this breakpoint - if -G specified, then
                                                       -g makes following options local, and vice versa
    -o                                              - send output to stderr rather than stdout
    -p pipe for output                              - send output to named pipe rather than stdout,
                                                      for times when stderr isn't far enough away
    -v                                              - verbose; use multiple times for extra info
    -- exe [arg1 [arg2 ...]]                        - command line to run
```

Any questions?

Can you debug a debugger?

Why didn't you write it in perl or python?

In Thumb, can a MOV to $pc cause a switch to ARM mode?

Can you quickly describe your data structures?

* Please note, Macbook Air is for pictorial purposes only.
I haven't even *tried* to compile it on OS X yet.