Debugging Unikernel Operating Systems

Kareem Ahmad, Alan Dearle, Jon Lewis, Ward Jaradat

School of Computer Science

University of St Andrews

Email: kareemahmad@protonmail.com

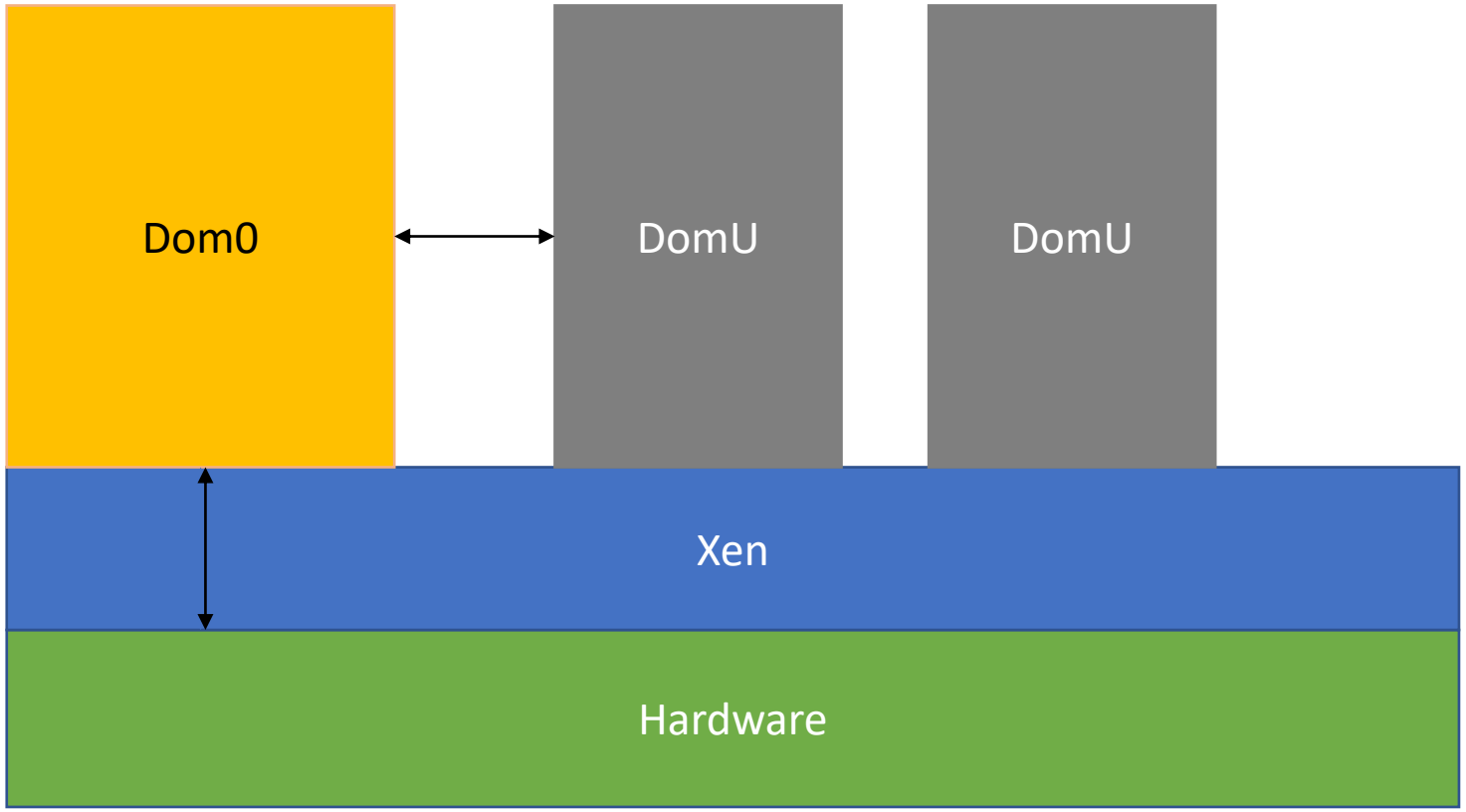
Overview

- In this talk we report on an undergraduate led project to develop debugger for unikernels running on Xen.
- Unikernels are challenging to debug as there are not many production ready debuggers for unikernels.
- Specifically, we focused on debugging support for the Stardust unikernel.
- This work is applicable to any unikernel written in C and hosted on Xen.

What Is A Unikernel?

- Specialised Operating System that runs directly above the Hypervisor (such as Xen)
- Single image that contains the OS, an application plus any required libraries
- Small image size (400KB including application)
- Fast to boot
- Fast to deploy
- Fast to provision

What is Xen?



The Problem

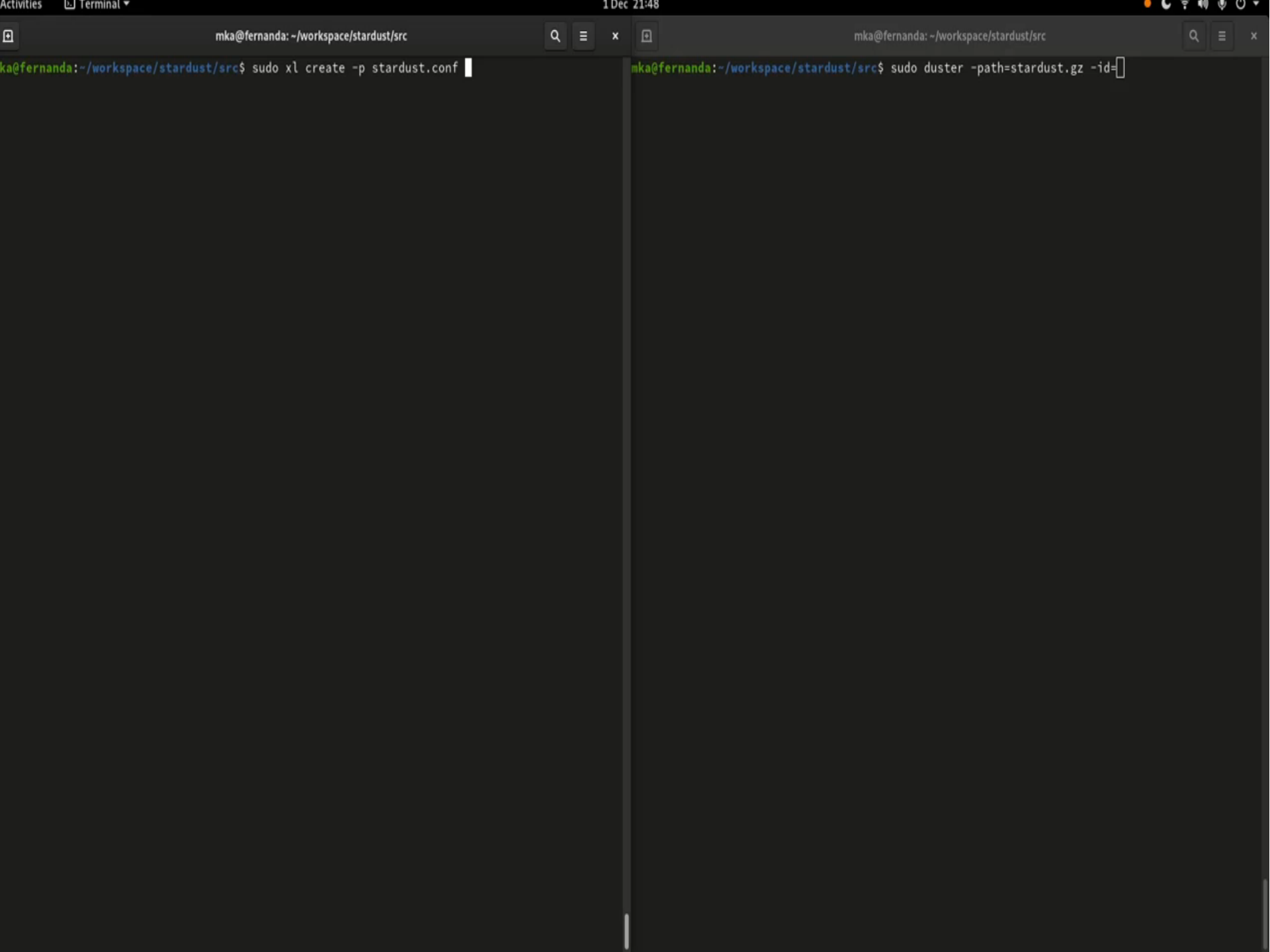
- Unikernels are difficult to debug:
  - The kernel and the application are compiled into a single image requiring embedded support.
  - An independent debugging context is needed in order to provide isolation and the ability to stop and start the Operating System.
  - Unikernels may not be designed for compatibility with conventional debugging tools like gdb
  - Gdb commonly makes use of Unix process structures, ptrace, and library calls

Approach

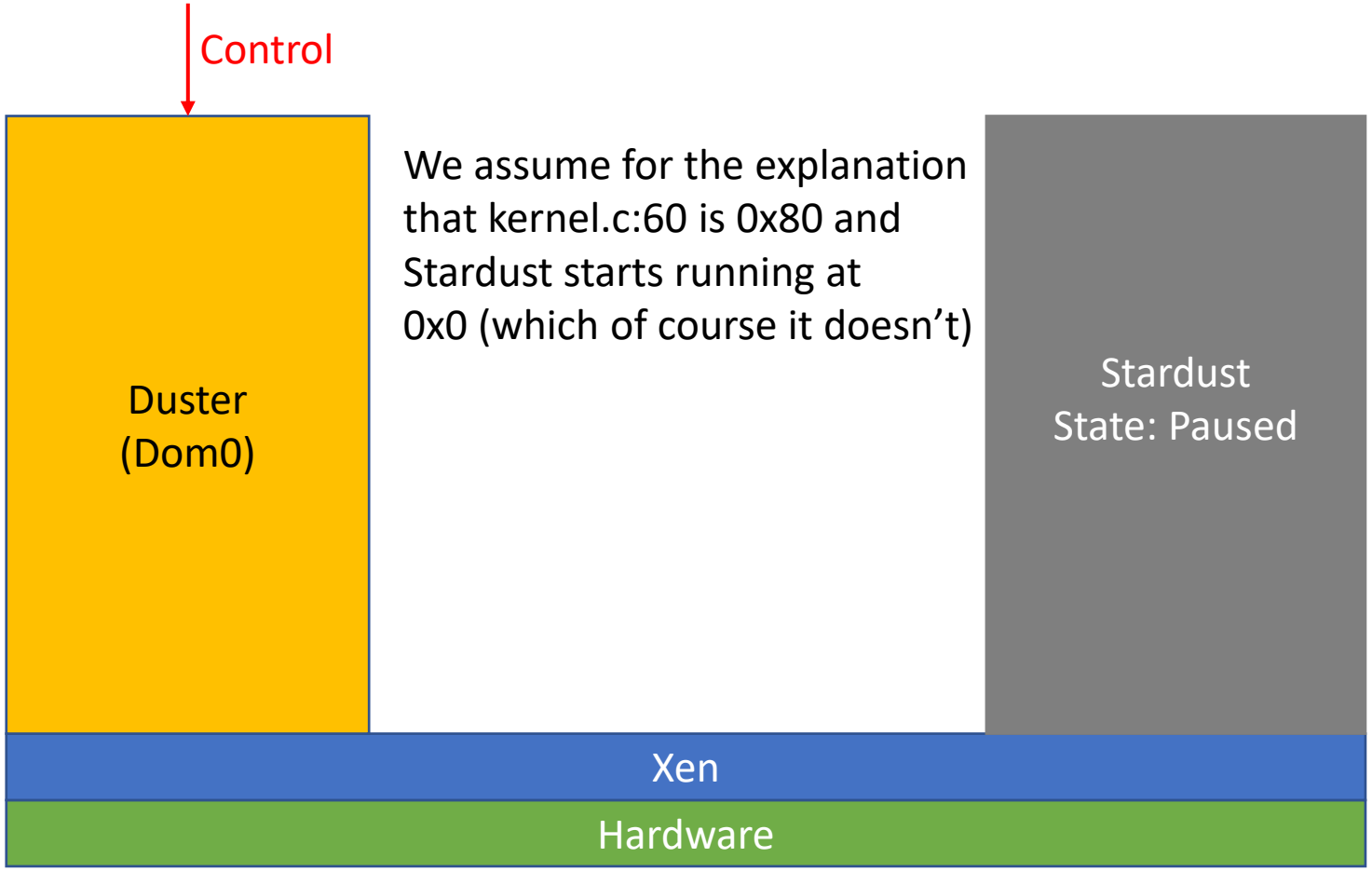
- The approach taken was based on xendbg developed at Nccgroup by Michael Spencer
- Algorithms and techniques used by Duster and xendbg are very similar.
- Duster focused on allowing the developer to use source-level constructs during debugging
- Uses Xen's Virtual Machine Introspection API for interacting with the Unikernel
- Written in C and Go
- Uses standard DWARF file format to get symbolic information (e.g. variables)

Demo Code (kernel.c)

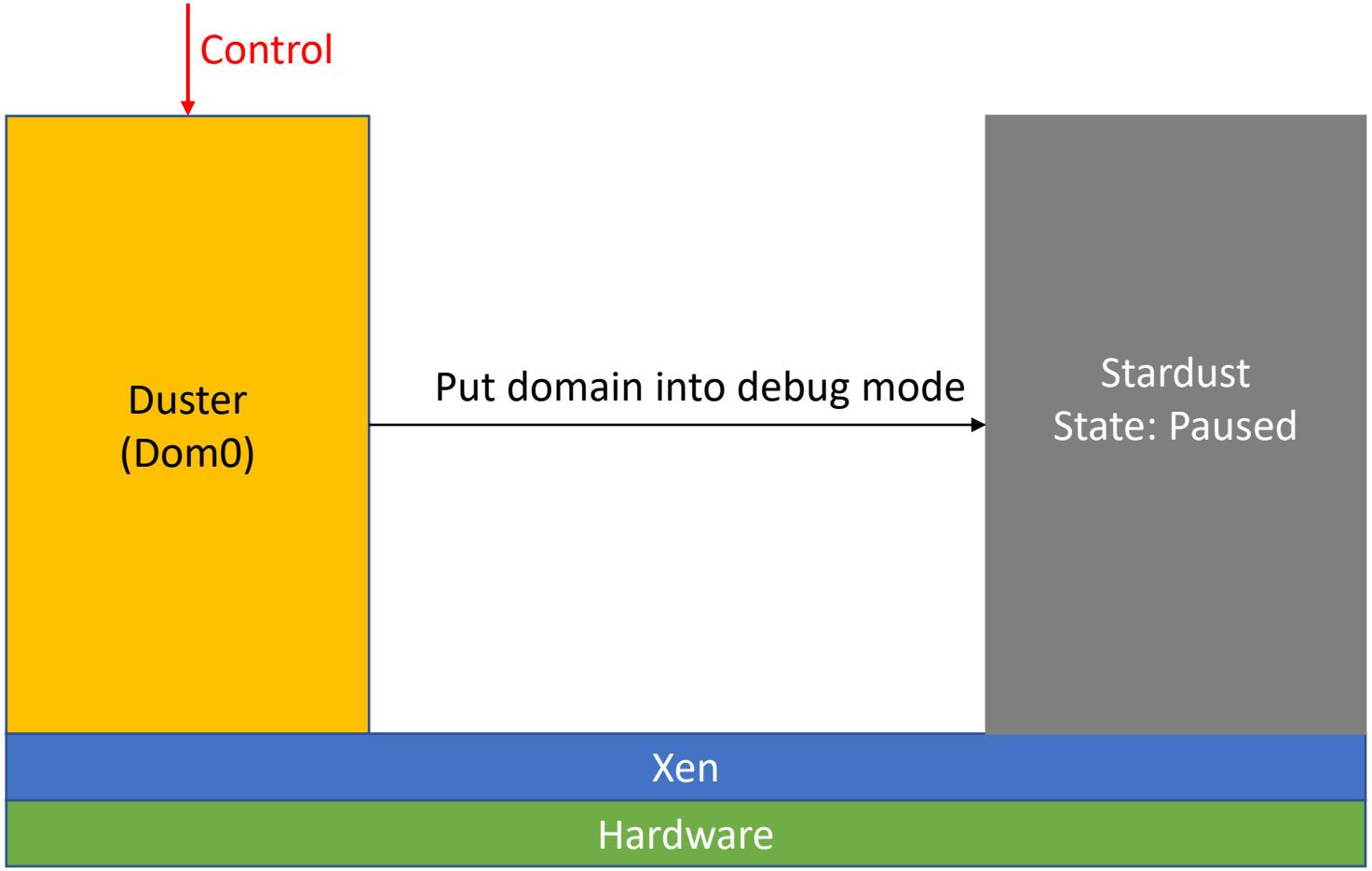
```
59 void demo() {
60         int my_integer = 0;
61         float my_float = 0.0;
62         for (int i = 0; i < 3; i++) {
63                 printf("We are on the %d iteration of the loop\n", i);
64                 my_integer += 100;
65                 my_float += 0.5;
66         }
67         bool my_boolean = true;
68         struct test *pointer_tester;
69         pointer_tester = malloc(sizeof(struct test));
70         pointer_tester->val = 20;
71         pointer_tester->no = 0.110;
72         pointer_tester->my_pointer = NULL;
73         printf("%d\n", pointer_tester->val);
74 }
```


Activities
Terminal
1 Dec 21:48
mka@fernanda: ~/workspace/stardust/src
ka@fernanda:~/workspace/stardust/src$ sudo xl create -p stardust.conf
mka@fernanda: ~/workspace/stardust/src
mka@fernanda:~/workspace/stardust/src$ sudo duster -path=stardust.gz -id=

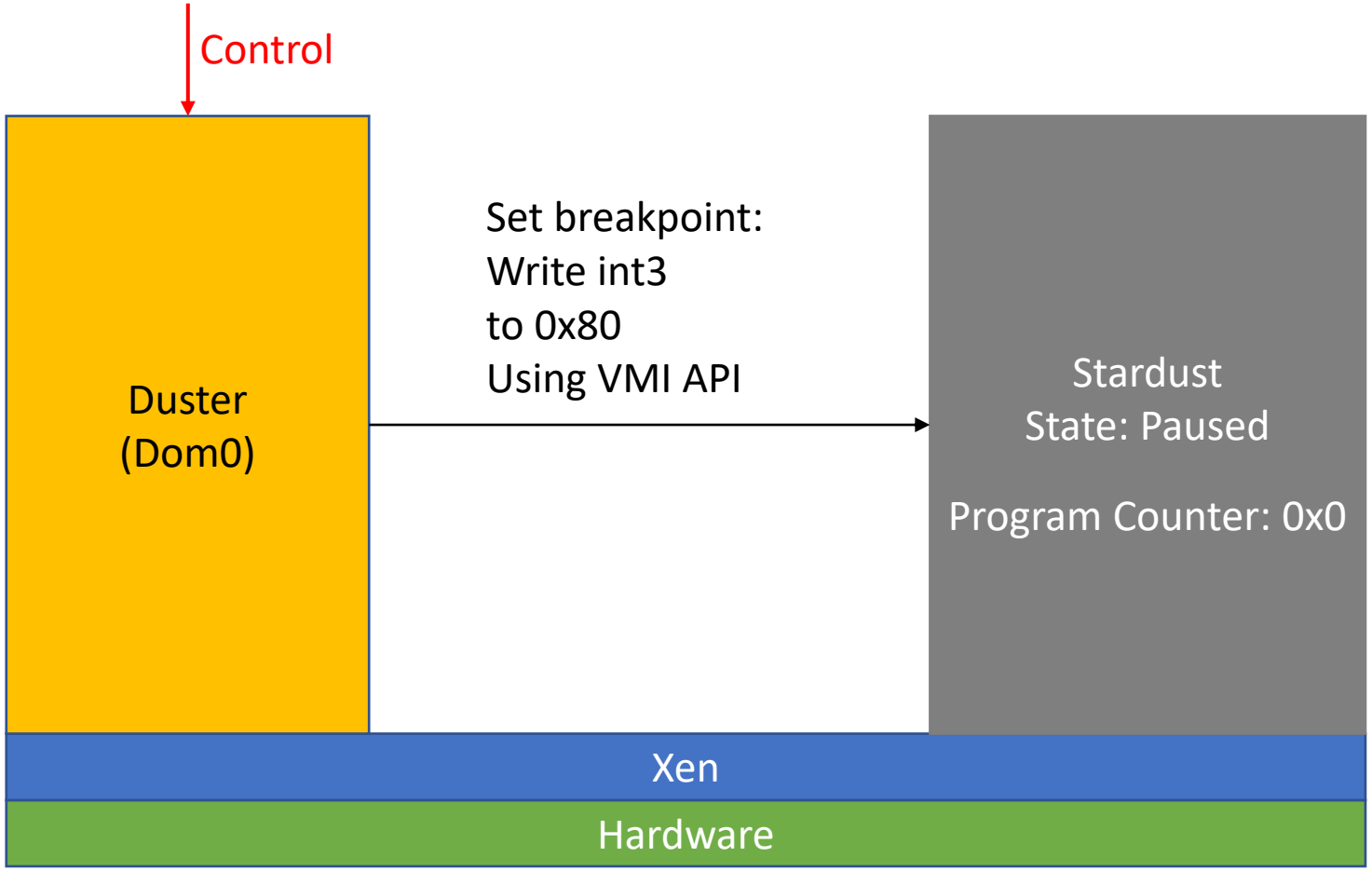
Explanation



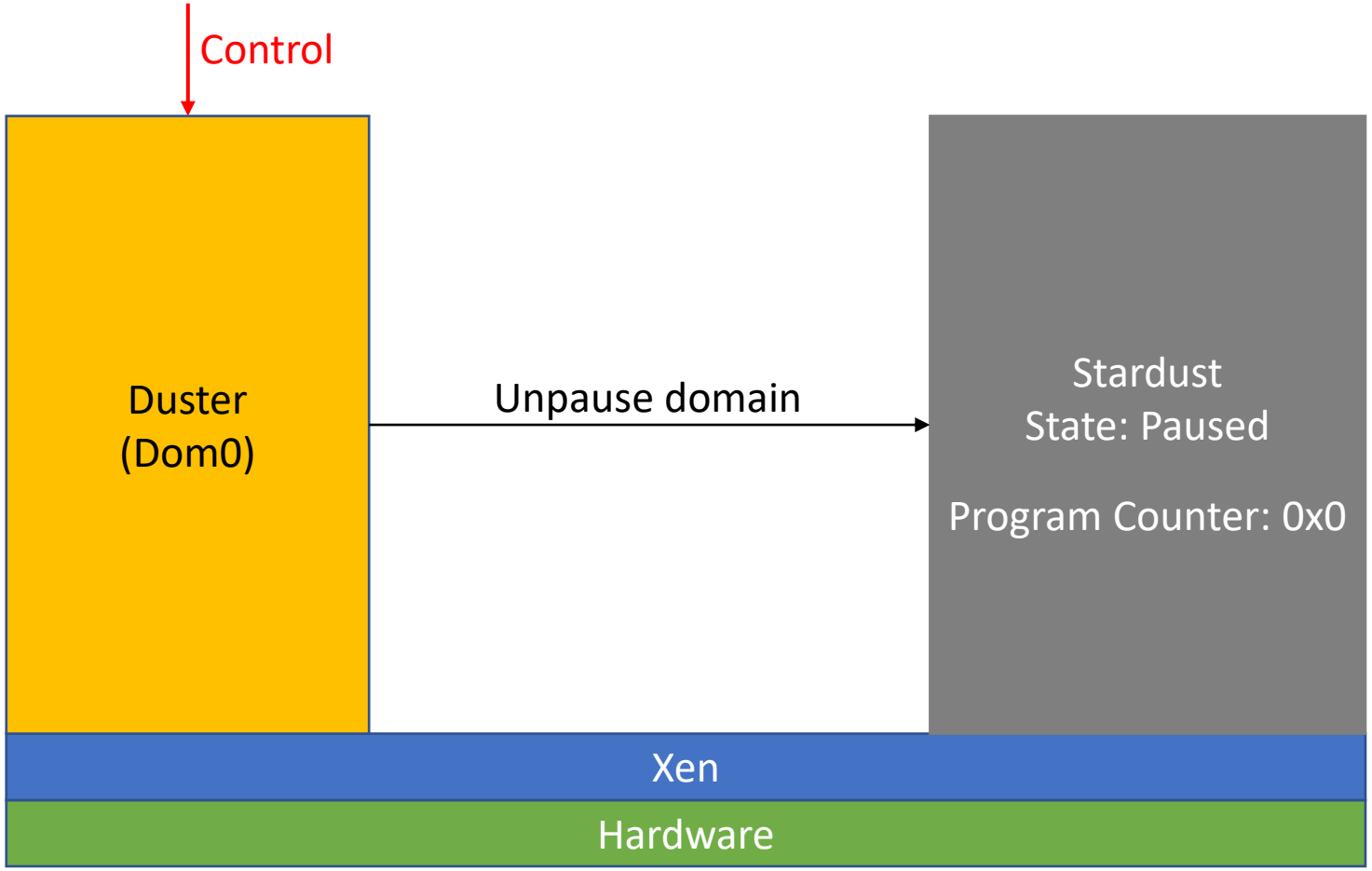
Explanation



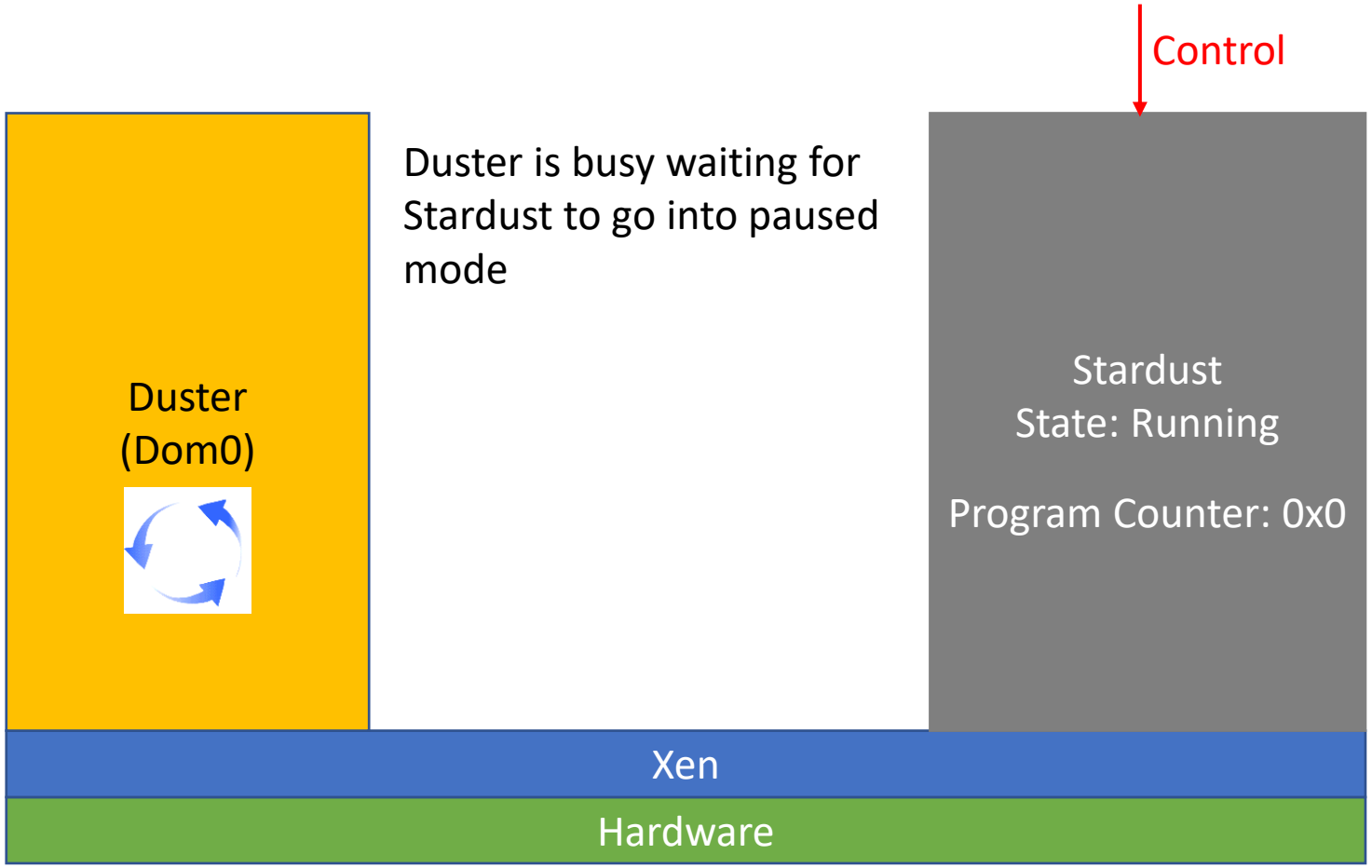
Explanation



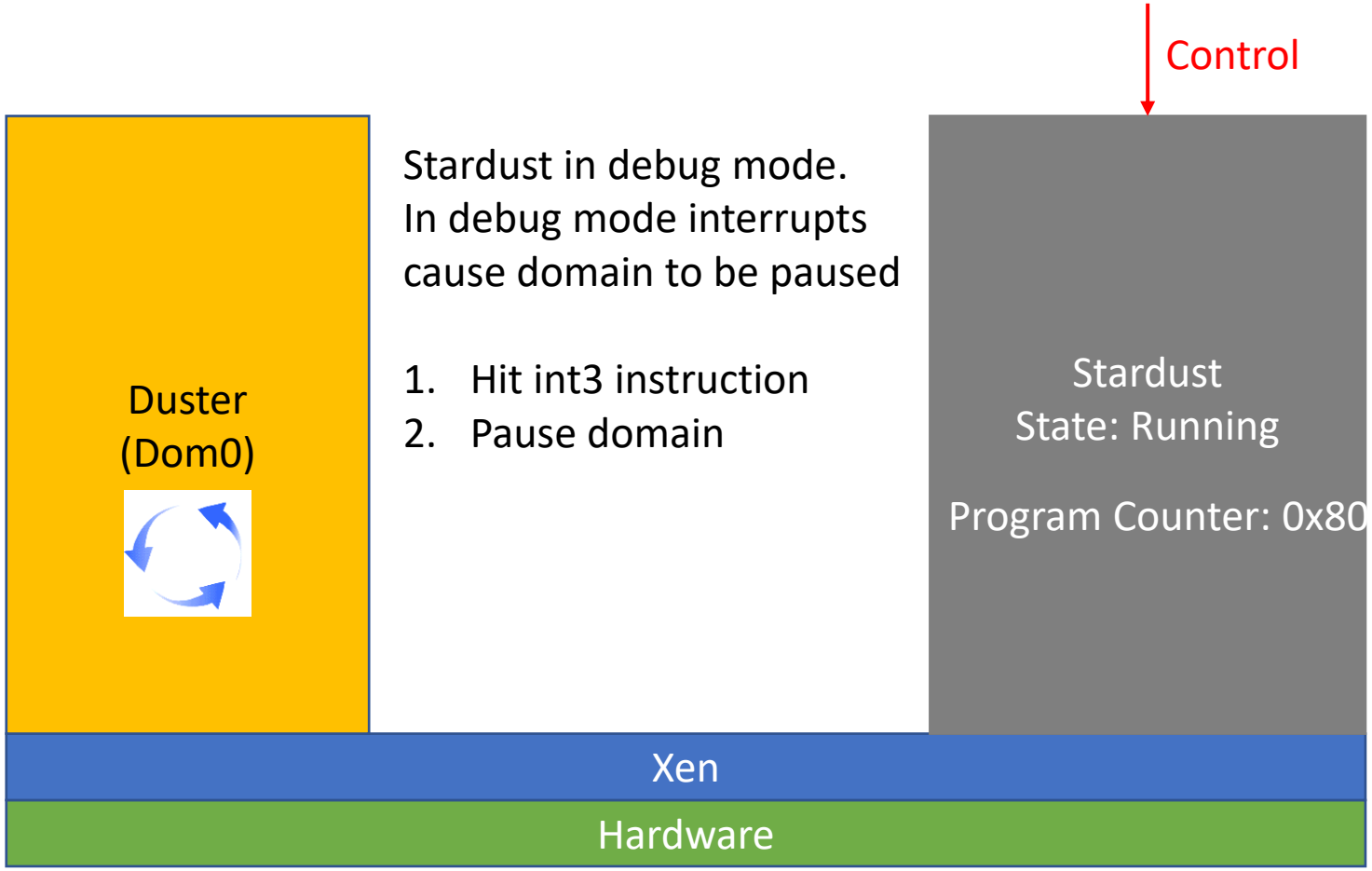
Explanation



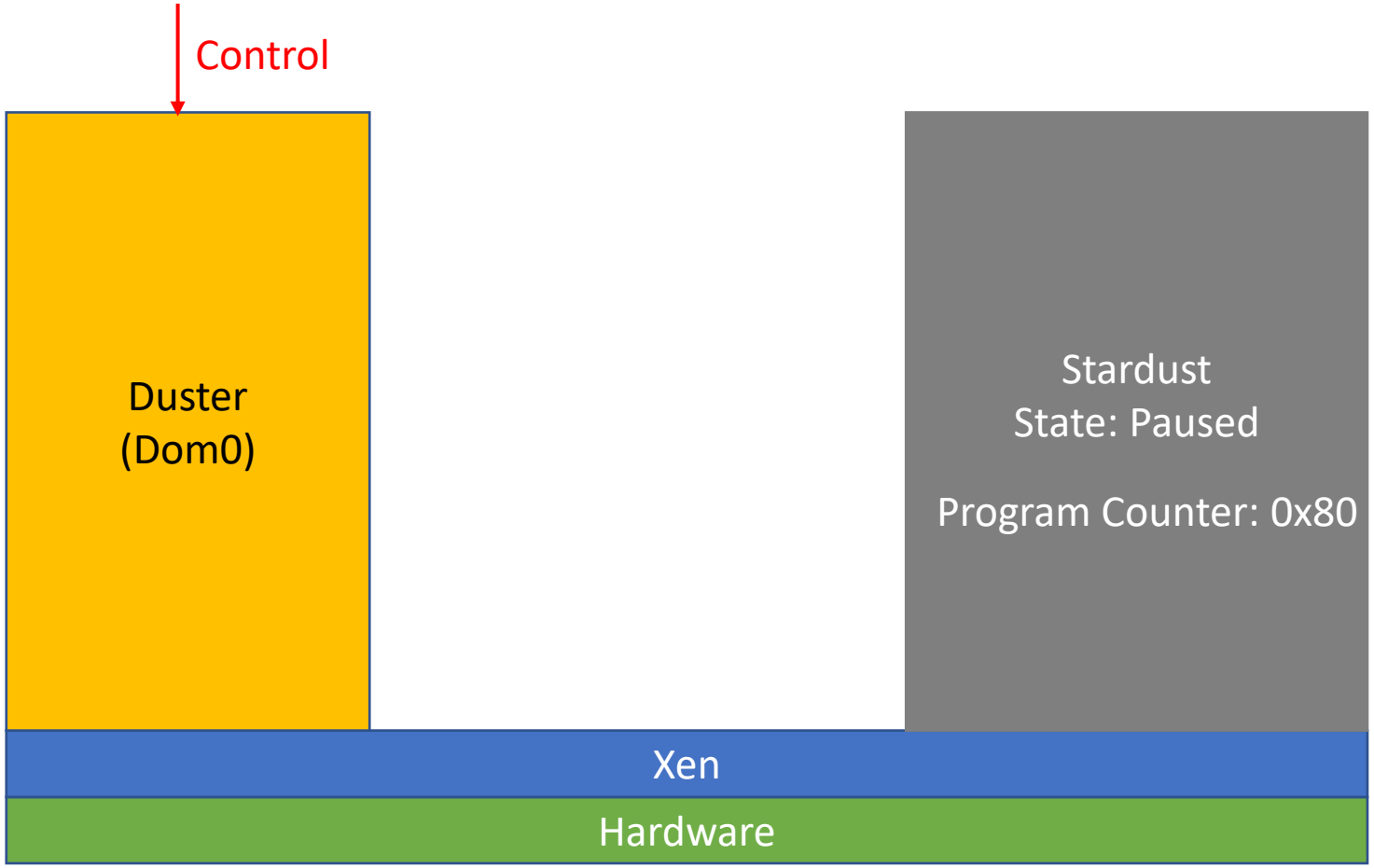
Explanation



Explanation

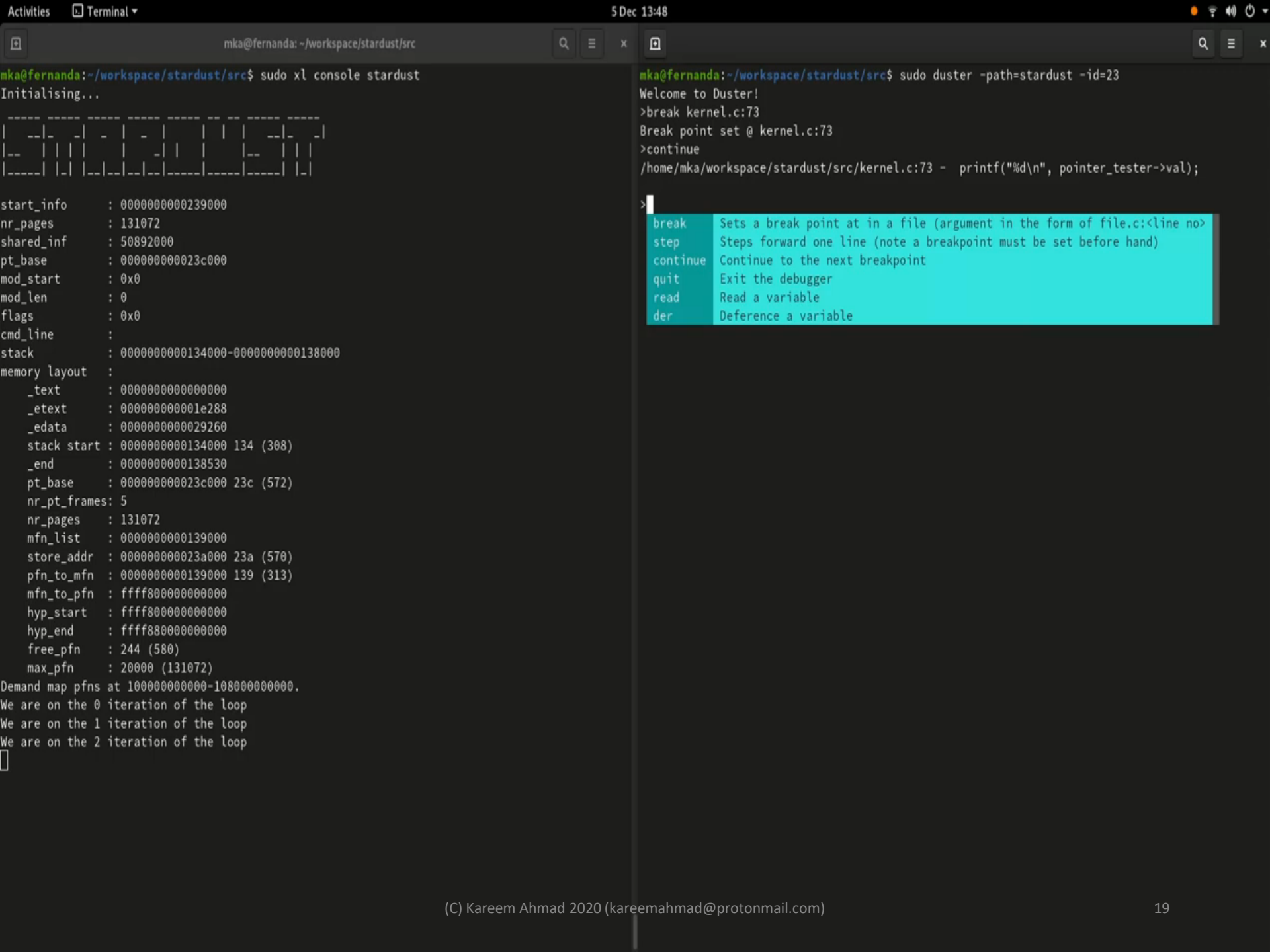


Explanation



Demo Code

```
59 void demo() {
60          int my_integer = 0;
61          float my_float = 0.0;
62          for (int i = 0; i < 3; i++) {
63                  printf("We are on the %d iteration of the loop\n", i);
64                  my_integer += 100;
65                  my_float += 0.5;
66          }
67          bool my_boolean = true;
68          struct test *pointer_tester;
69          pointer_tester = malloc(sizeof(struct test));
70          pointer_tester->val = 20;
71          pointer_tester->no = 0.110;
72          pointer_tester->my_pointer = NULL;
73          printf("%d\n", pointer_tester->val);
74 }
```


Activities
Terminal
5 Dec 13:48
mka@fernanda: ~/workspace/stardust/src
mka@fernanda:~/workspace/stardust/src$ sudo xl console stardust
Initialising...
start_info : 0000000000239000
nr_pages : 131072
shared_inf : 50892000
pt_base : 000000000023c000
mod_start : 0x0
mod_len : 0
flags : 0x0
cmd_line :
stack : 0000000000134000-0000000000138000
memory layout :
_text : 0000000000000000
_etext : 000000000001e288
_edata : 0000000000029260
stack start : 0000000000134000 134 (308)
_end : 0000000000138530
pt_base : 000000000023c000 23c (572)
nr_pt_frames: 5
nr_pages : 131072
mfn_list : 0000000000139000
store_addr : 000000000023a000 23a (570)
pfn_to_mfn : 0000000000139000 139 (313)
mfn_to_pfn : ffff800000000000
hyp_start : ffff800000000000
hyp_end : ffff880000000000
free_pfn : 244 (580)
max_pfn : 20000 (131072)
Demand map pfns at 100000000000-108000000000.
We are on the 0 iteration of the loop
We are on the 1 iteration of the loop
We are on the 2 iteration of the loop
mka@fernanda:~/workspace/stardust/src$ sudo duster -path=stardust -id=23
Welcome to Duster!
>break kernel.c:73
Break point set @ kernel.c:73
>continue
/home/mka/workspace/stardust/src/kernel.c:73 - printf("%d\n", pointer_tester->val);
>
break Sets a break point at in a file (argument in the form of file.c:<line no>
step Steps forward one line (note a breakpoint must be set before hand)
continue Continue to the next breakpoint
quit Exit the debugger
read Read a variable
der Deference a variable

Conclusions

- We have extended the Xendbg code to support the debugging of high level (C) programming language code x86-64 Para-virtualised unikernels
- Without this (we at least) had no debug support
- Does not support some GDB operations including writing to memory, stack frame analysis
- But can:
  - Set and remove breakpoints on the source level
  - Step through code line at a time
  - Read memory using symbolic names
  - Pretty print memory based on C types

Links

- Xendbg: https://github.com/SpencerMichaels/xendbg
- Duster: https://github.com/StardustOS/duster
- Stardust: https://github.com/StardustOS
- Stardust (docs): https://stardustos.gitbook.io/docs/