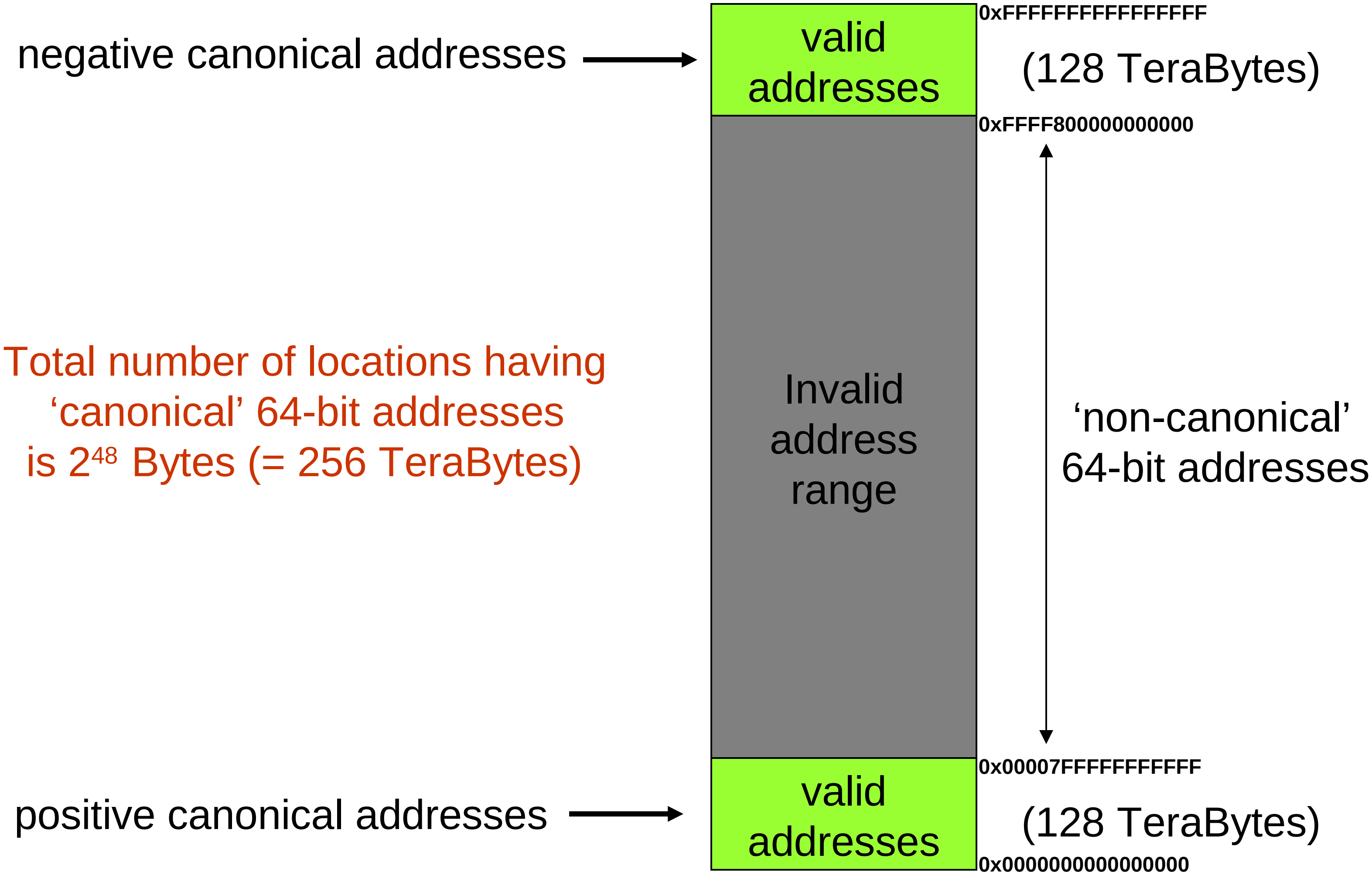
On 64-bit ‘code-relocation’

How we can launch a procedure in 64-bit mode that resides in a page-frame at a high address

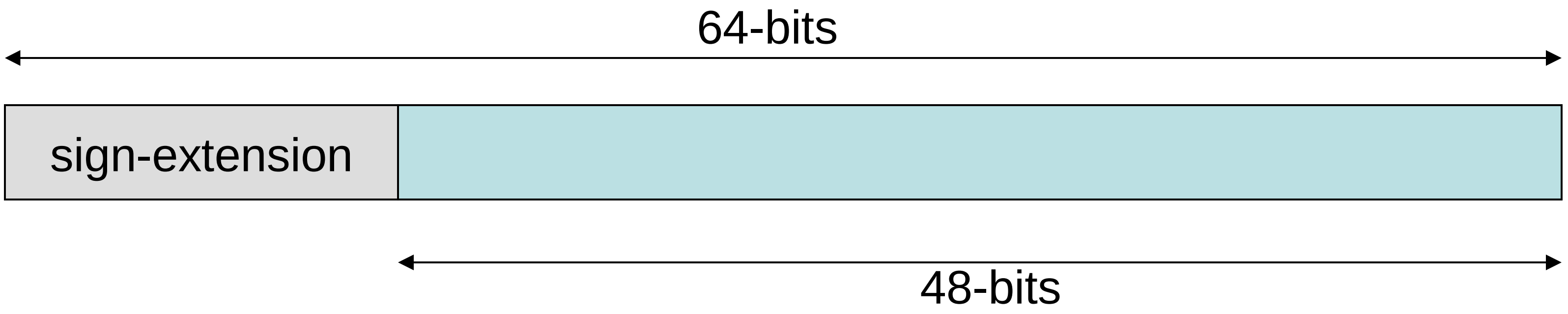
64-bit virtual address-space



Total number of locations having 'canonical' 64-bit addresses is 2^{48} Bytes (= 256 TeraBytes)

The ‘canonical’ addresses

- In a 64-bit ‘canonical’ virtual address, the uppermost 16-bits are identical to bit 47

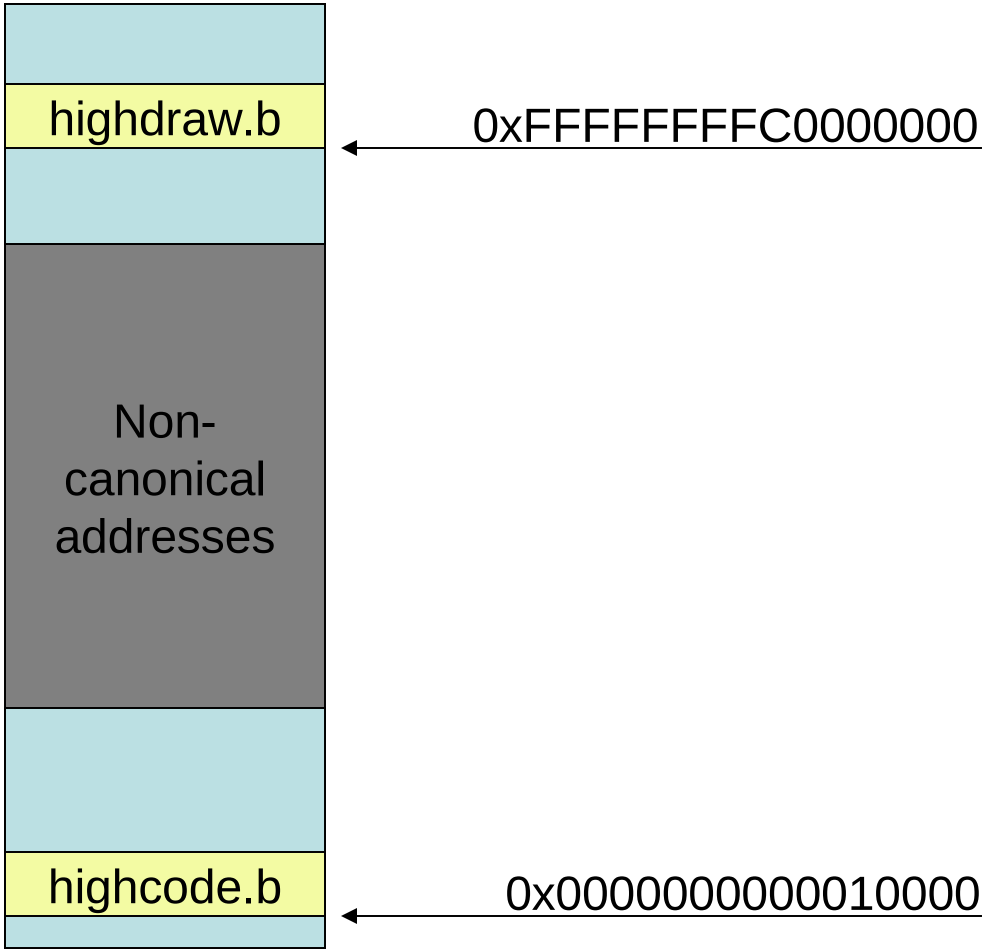


- The number of distinct virtual addresses actually implemented is 2^{48} (= 256 TB)

Our ‘highcode.s’ demo

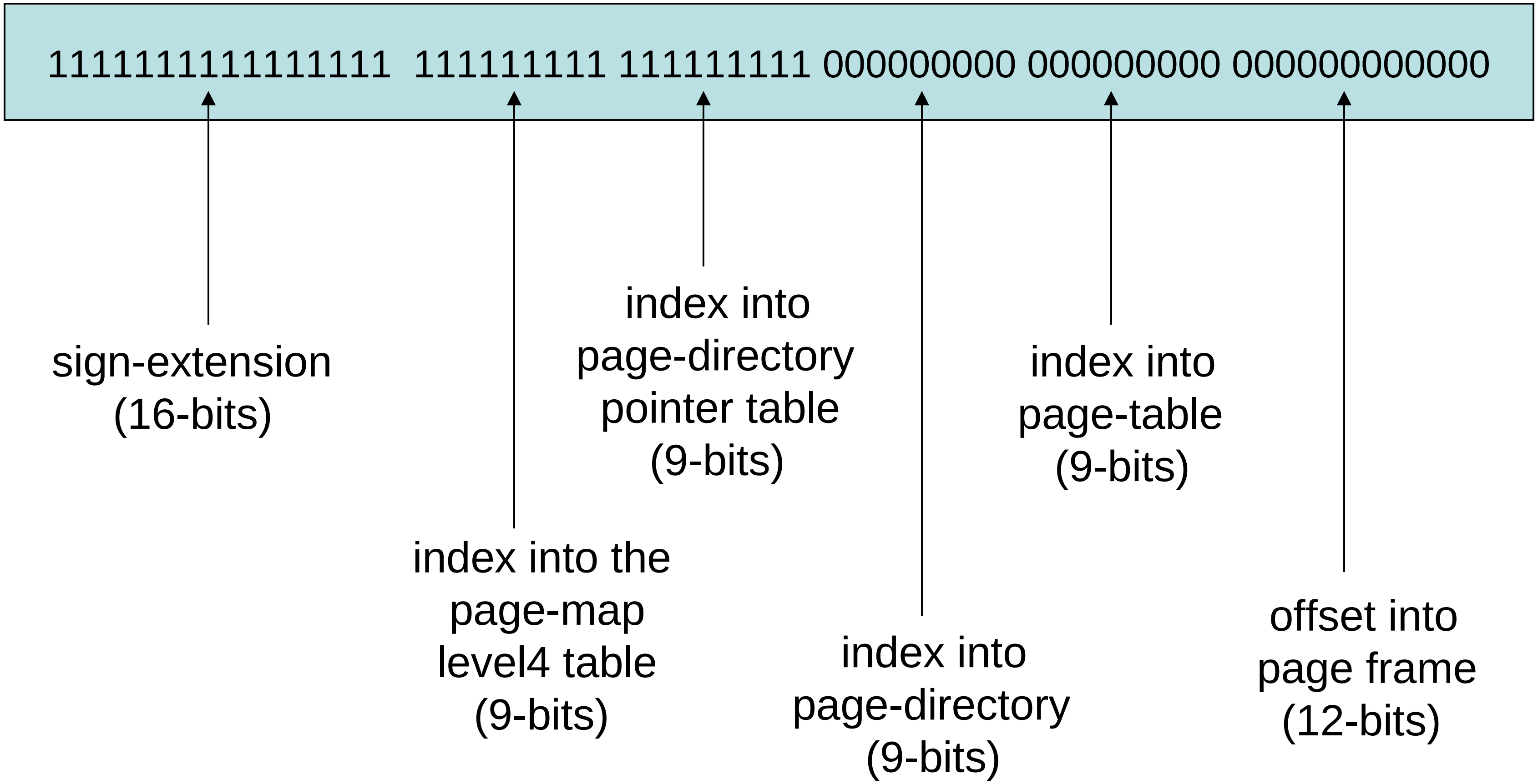
- We have created a demo-program which loads and executes a procedure that will reside at a very high location within the processor’s 64-bit virtual address space
- Some aspects of our design were shaped by limitations of our GNU software tools, and by our desire to keep the addressing ‘page-map’ tables as simple as possible

Two components



virtual address space

Subfields of 64-bit address



1111111111111111 111111111 111111111 000000000 000000000 000000000000

sign-extension
(16-bits)

index into the
page-map
level4 table
(9-bits)

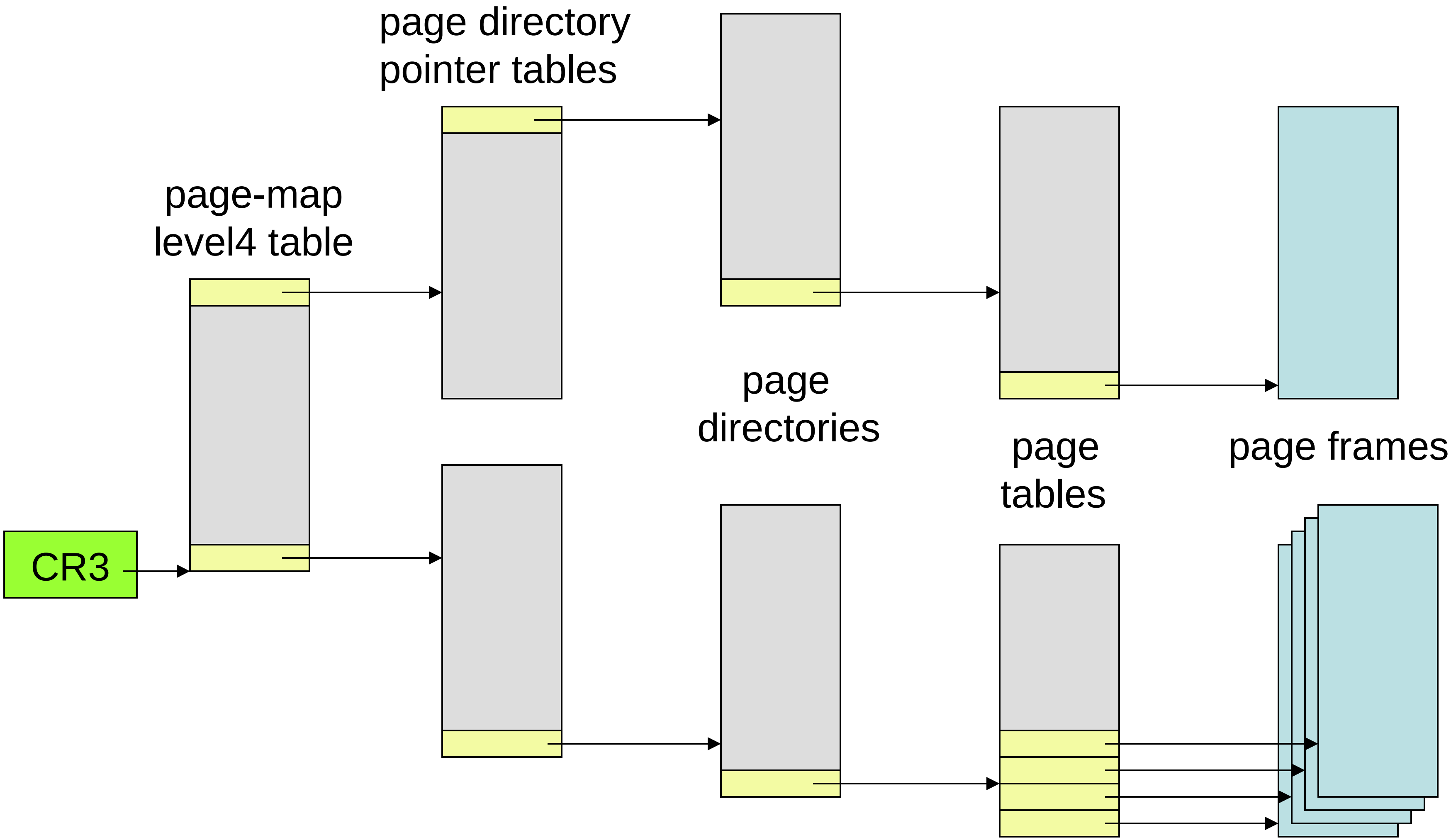
index into
page-directory
pointer table
(9-bits)

index into
page-directory
(9-bits)

index into
page-table
(9-bits)

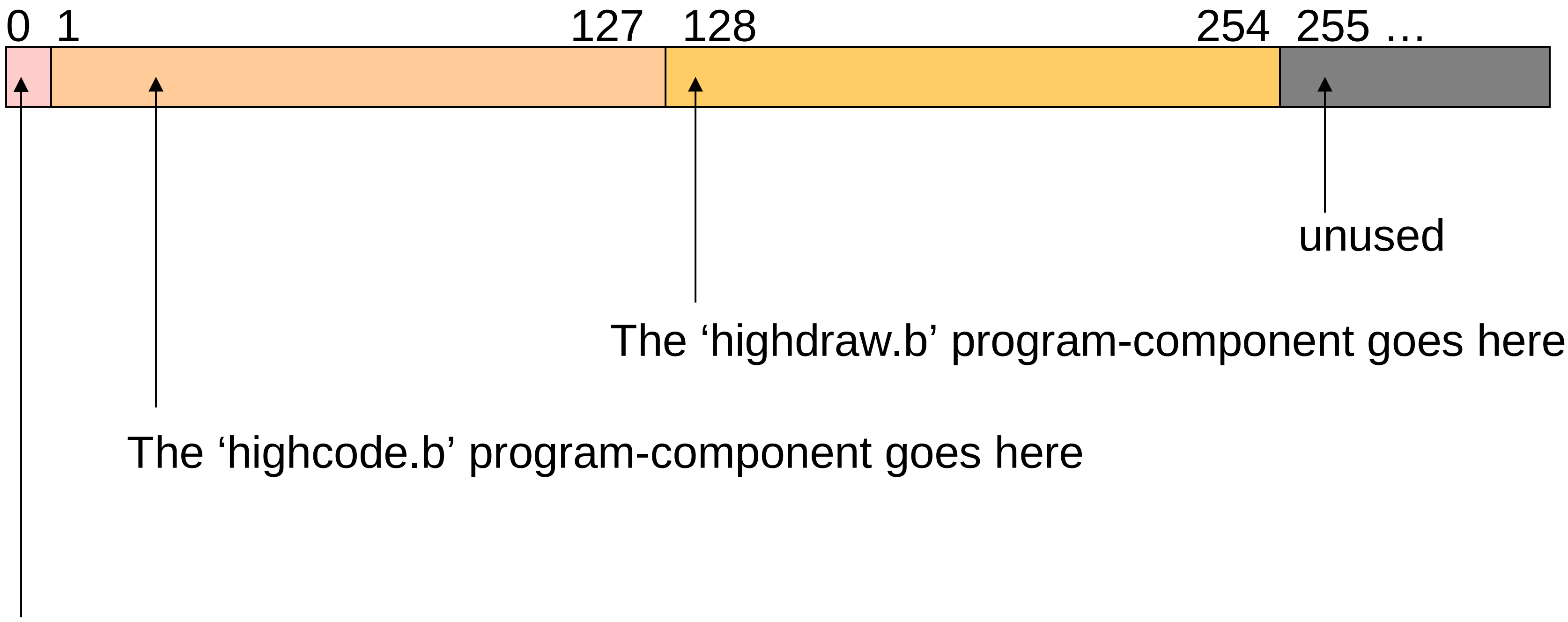
offset into
page frame
(12-bits)

4-level page-mapping tables



Disk-partition’s layout

/dev/sda4



0 1 127 128 254 255 …

unused

The ‘highdraw.b’ program-component goes here

The ‘highcode.b’ program-component goes here

Our ‘cs686ipl.b’ boot-loader goes here (in ‘boot-sector’ of this disk-partition)

‘ld’ handles 32-bit relocations

- We use a special linker-script to ‘relocate’ symbolic addresses used in ‘highdraw.s’

```
OUTPUT_FORMAT(binary);

SECTIONS {
                . = 0xFFFFFFFFC0000000;
                .text  : { *(.text) } =0x90909090
                .data : { *(.data) } = 0x00000000
                .bss   : { *(.bss)  }  = 0x00000000
          }
```

assembly commands

- Assemble the ‘highcode.s’ component:

```
$ as highcode.s –o highcode.o
```

- Assemble the ‘highdraw.s’ component:

```
$  as highdraw.s  -o  highdraw.o
```

Linker commands

- Link the ‘highcode.o’ component:

```
$ ld  highcode.o  -T ldscript  -o  highcode.b
```

- Assemble the ‘highdraw.s’ component:

```
$  ld  highdraw.o  -T hiscript   -o  highdraw.b
```

Installation commands

- Install the ‘highcode.b’ component:

```
$ dd  if=highcode.b  of=/dev/sda4  seek=1
```

- Assemble the ‘highdraw.s’ component:

```
$  dd  if=highdraw.b  of=/dev/sda4  seek=128
```

Addition to ‘cs686ipl.s’

- We made a small but useful enhancement to our ‘cs686ipl.s’ boot-loader program, so subsequent program components will not need to repeat the search for the starting Logical Block Address of the disk-partition
- That block-number is left in register EBX, where the next component can find it
- It’s also written to the ROM-BIOS ‘mailbox’

In-class exercise

- After you have successfully downloaded, assembled, linked, installed, and executed the ‘highcode.s’ demo-program, see if you can modify it so that its ‘highdraw’ code-component will reside at an even higher virtual address, namely:

  0xFFFFFFFFFFE00000

  instead of:

  0xFFFFFFFFC0000000