Address Spaces

Glenn Bruns
CSUMB

Lecture Objectives

At the end of this lecture, you should be able to:

- List and explain problems related to memory addressing
- Define “address space”, “physical address” and “virtual address”

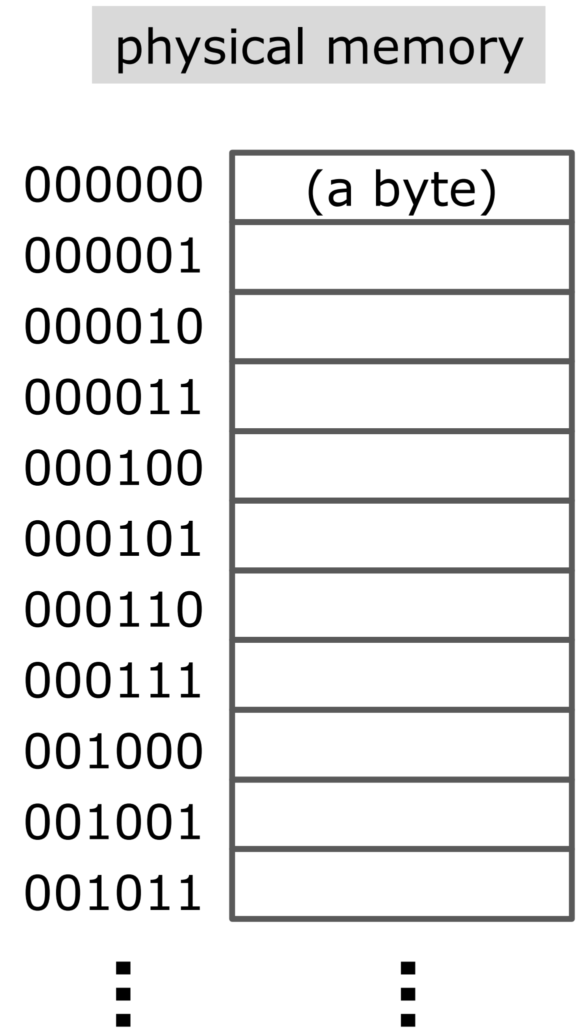
OS memory management

Memory is another resource the OS needs to manage.

For example, allocate memory to processes.

Each location in RAM memory has a physical address.

A six bit memory address



We'll usually assume memory is "byte-addressable"

Program addresses

In code we use "symbolic addresses" – variable names

When a program is compiled they are turned into memory addresses

C code

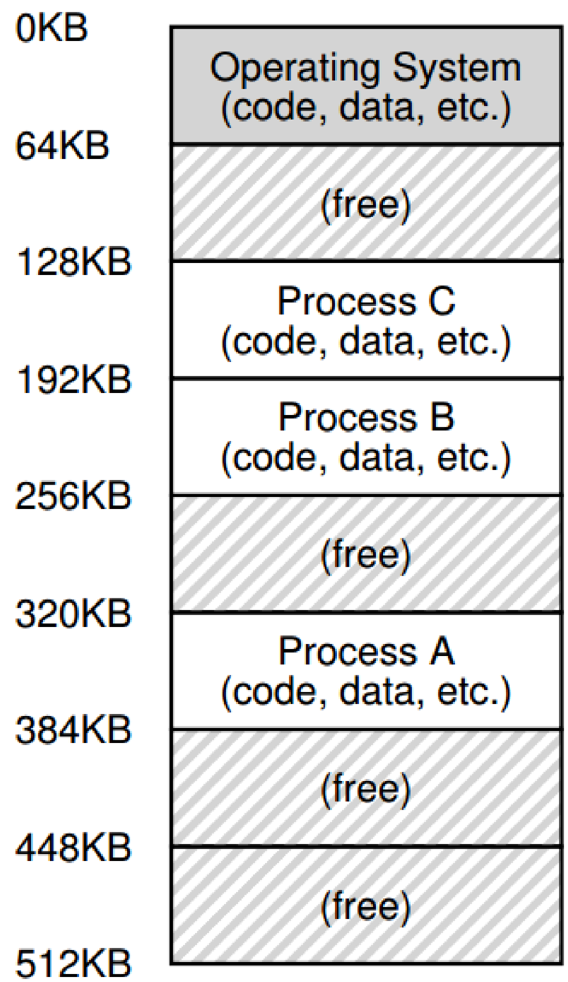
```
x++;
```

assembly code

```
inc 1065
```

Problem 1

How can the compiler generate addresses if it doesn't know where a program will be in memory?



(diagram from Operating Systems: Three Easy Pieces, Arpaci-Dusseau & Arpaci-Dusseau)

Possible solutions

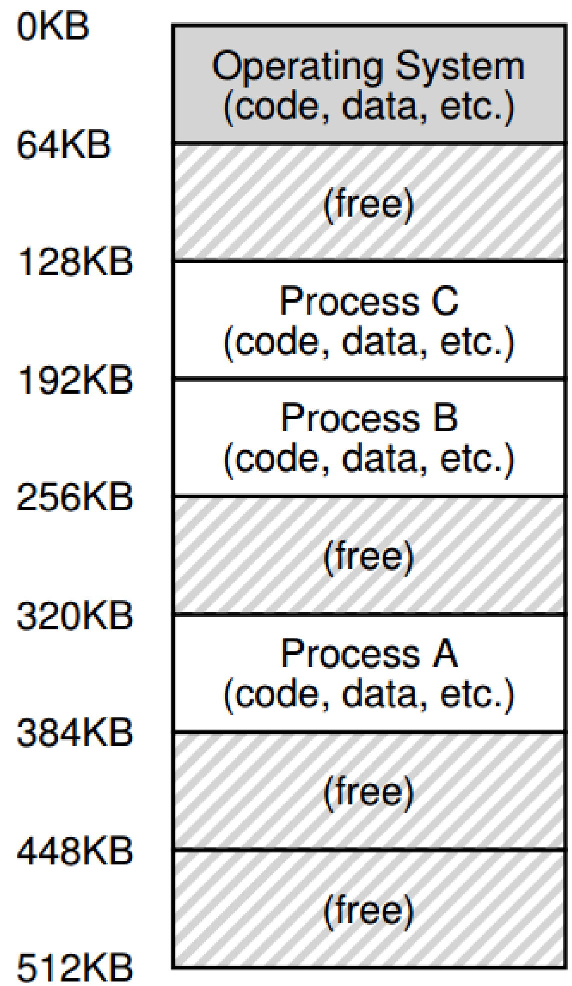
1. When program is loaded into memory, translate all the addresses
   - this slows down program loading
2. Keep only one program in memory at a time
   - this really slows down multi-programming
3. While the program is running, translate addresses to physical addresses
   - this slows down addressing

Example: translate address 100 in binary code to physical address 1200

Problem 2

How to provide protection?

For example, programs can't write into OS memory



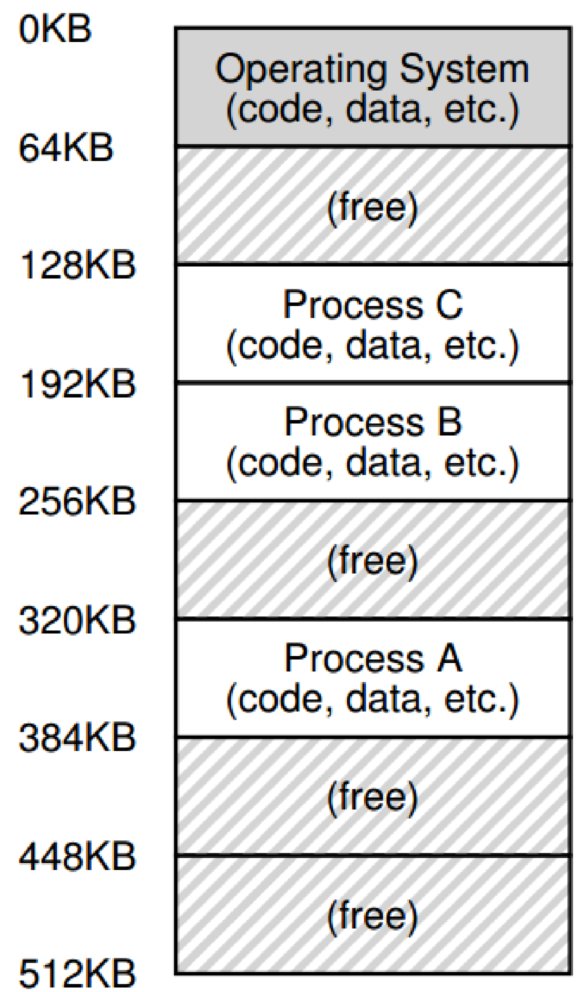
(diagram from Operating Systems: Three Easy Pieces, Arpaci-Dusseau & Arpaci-Dusseau)

Possible solutions

1. At “compile time”, try to check every address to make sure it is safe
   - is this even possible?
2. At “run time”, check every memory access to make sure it is safe
   - issues with overhead?

Problem 3

What if a program addresses more memory than is available?

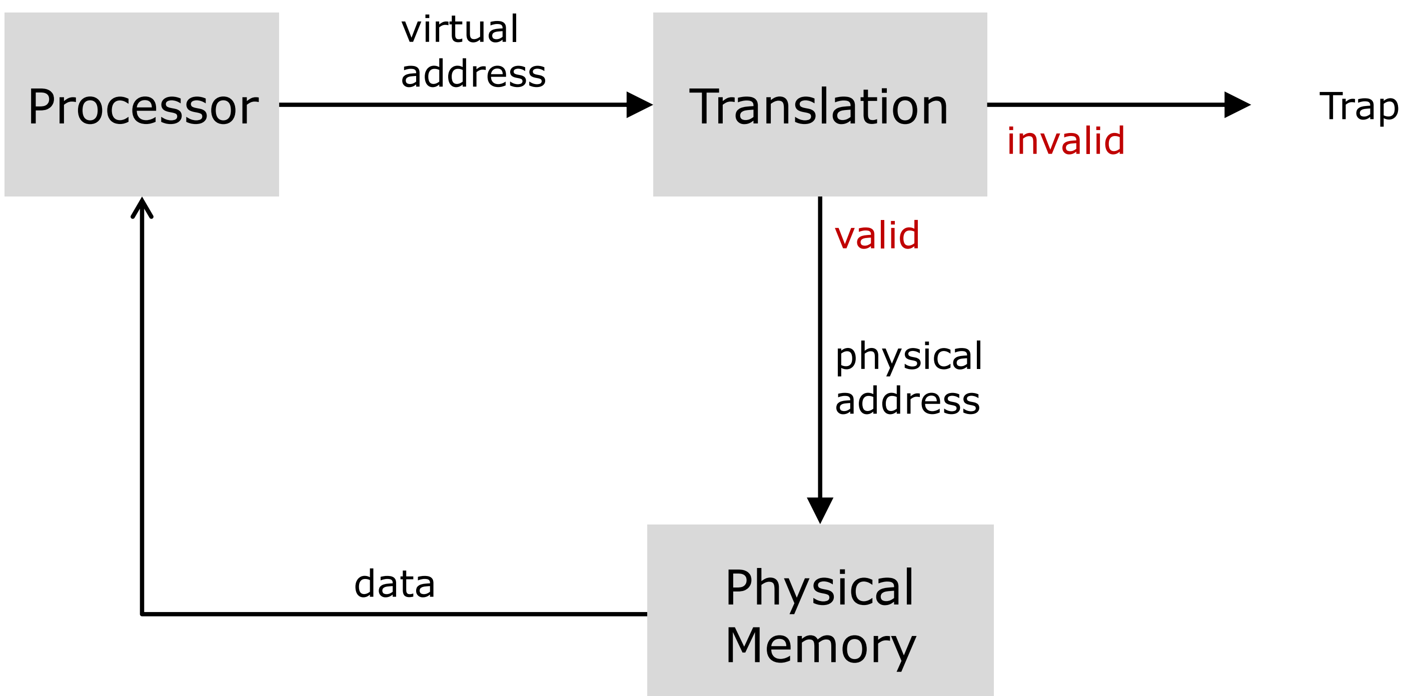


(diagram from Operating Systems: Three Easy Pieces, Arpaci-Dusseau & Arpaci-Dusseau)

Possible solutions

1. At compile time, reject program
   - programmer has to rewrite; maybe using disk
2. At run time, check addresses and terminate program if address is too big
   - issues with overhead?
3. At run time, if address too big, somehow map it onto other storage, like disk
   - issues with overhead?

The modern solution: virtual memory



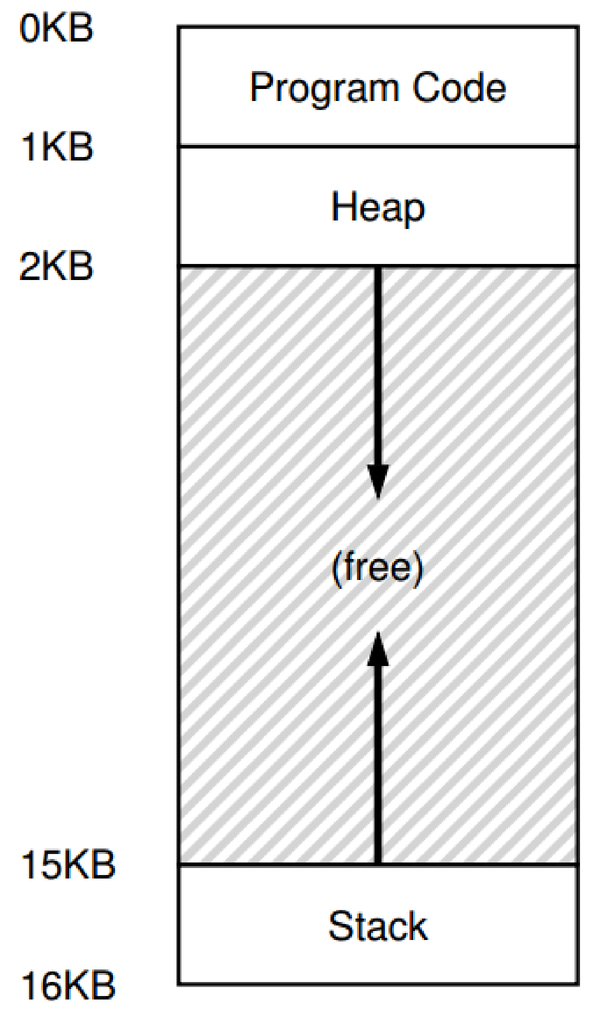
- Compiled code uses "virtual addresses"
- Virtual addresses are translated to physical addresses at run time
- Translation must be fast – so there is hardware support
- If a program tries to address memory outside its virtual address space, a trap occurs.

Address space

An **address space** is a set of addresses a process can use to address memory.

Each process has its own address space.

Repeat after me: each process has its own address space.



(diagram from Operating Systems: Three Easy Pieces, Arpaci-Dusseau & Arpaci-Dusseau)

A demonstration of transparency

```
#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char *argv[]) {
   printf("location of code : %p\n", (void *) main);
   printf("location of heap : %p\n", (void *) malloc(1));
   int x = 3;
   printf("location of stack : %p\n", (void *) &x);
   return x;
}
```

```
$ ./print-addr
location of code : 0x80483f4
location of heap : 0x8dc6008
location of stack : 0xbfbe985c
```

(code from Operating Systems: Three Easy Pieces, Arpaci-Dusseau & Arpaci-Dusseau)

Virtual memory is a powerful feature

A few of the neat things you can do with virtual memory:

- Allow processes to share code in memory
- Support persistent memory, where changes to data structures survive program and system crashes
- Allow processes to be transparently moved in memory, or moved from one server to another

Evaluating address translation

We'll look at different ways to implement virtual memory.

Use guiding questions to evaluate alternatives:

Easy to use virtualization?

- running program is unaware of address translation

Protection?

- protect processes from processes, OS from processes

Efficiency?

- speed of translation

Low overhead?

- use of memory for translation tables and other data structures

Summary

- Problems in memory addressing:
  - How to generate addresses if we don’t know where program will sit in memory?
  - How to provide protection from bad memory accesses?
  - What if program addresses more memory than available?
- Sketch of the solution:
  - compiler generates virtual addresses
  - each process has its own, simple address space
  - these are translated to physical addresses at run time with the help of hardware