Processes: A Fundamental OS Abstraction

- A process is a bundle of resources
  - Address space
  - One or more threads of execution
    - Code (i.e., machine instructions)
    - Registers (stack pointer, instruction pointer, general-purpose registers)
  - Other bookkeeping stuff like . . .
    - Open file descriptors (e.g., pipes, network sockets, on-disk files)
    - Process id (pid)
    - Process state (running, blocked, etc.)
- A single “application” contains one or more processes

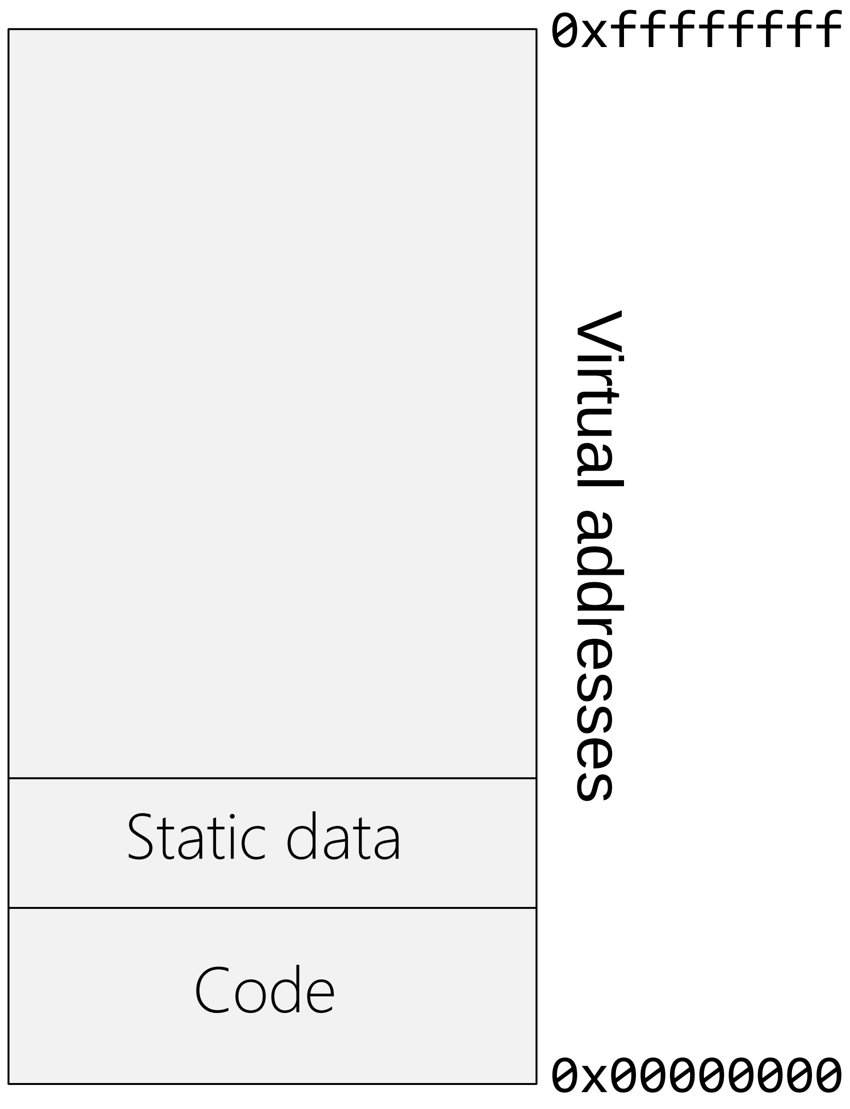
What's an Address Space?

- The set of virtual memory addresses that a process can access
  - A large array of bytes starting at 0 and going up to some large number (e.g., 4 GB)
  - Different parts of virtual memory hold different parts of the program

Code + static data

```
push %ebp
mov %esp,%ebp
sub $0x18,%esp
```

```
//At top of .c file
int foo = 42;
```

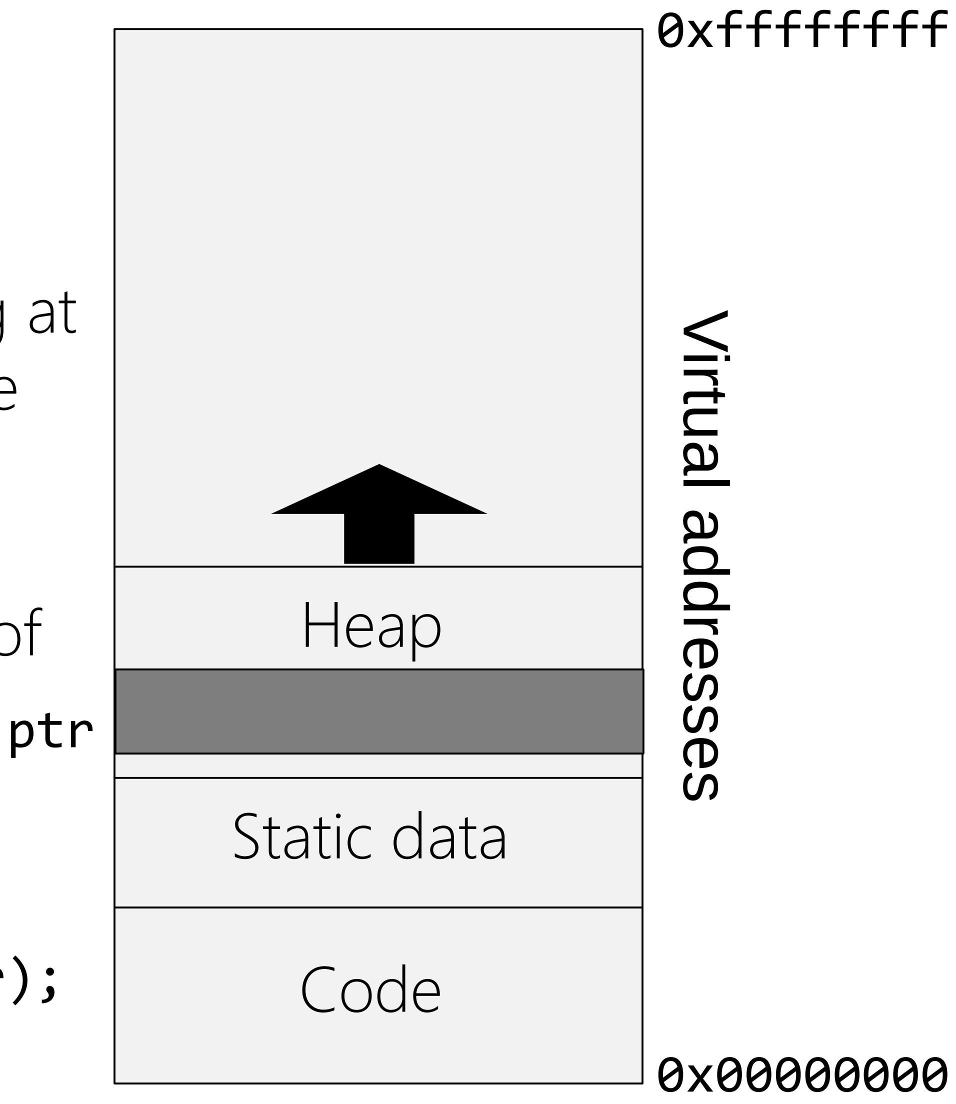


What’s an Address Space?

- The set of virtual memory addresses that a process can access
  - A large array of bytes starting at 0 and going up to some large number (e.g., 4 GB)
  - Different parts of virtual memory hold different parts of the program

Heap

```
char *ptr = malloc(4096);
printf("%p\n", (void *)&ptr);
      //“0x7ffd90590168”
```

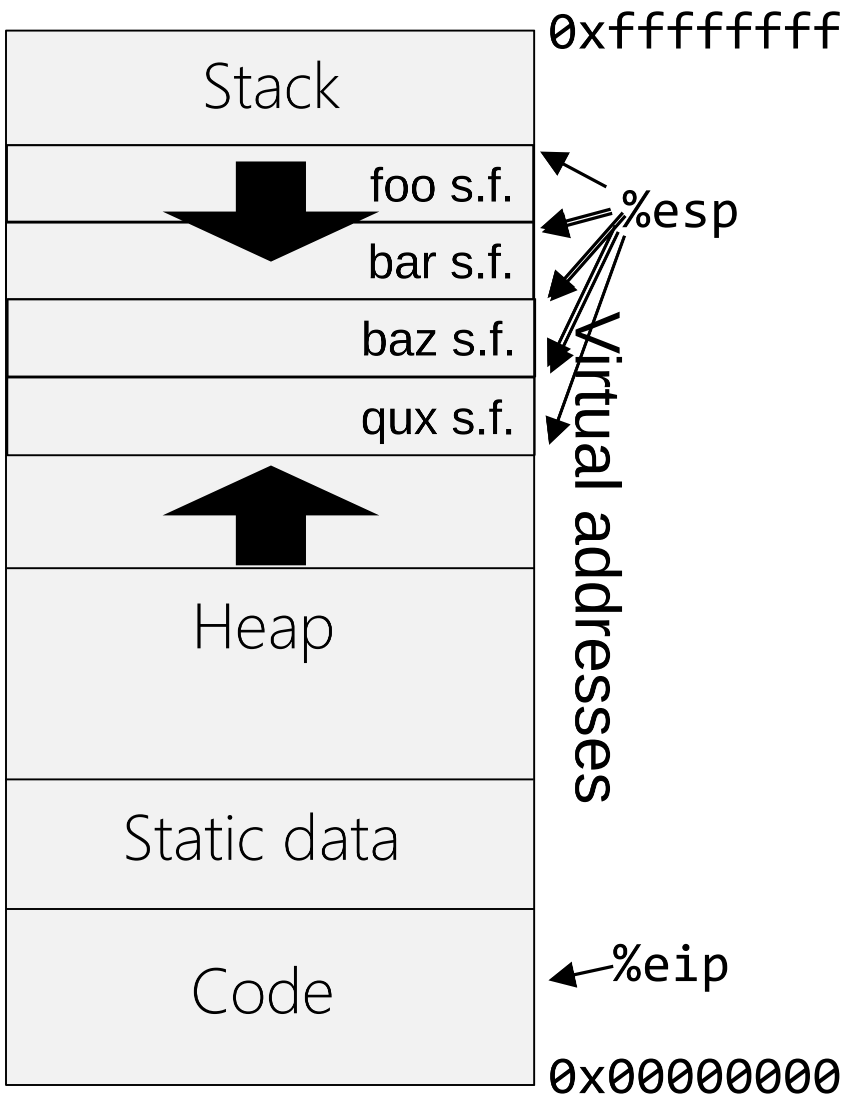


What's an Address Space?

- The set of virtual memory addresses that a process can access
    - A large array of bytes starting at 0 and going up to some large number (e.g., 4 GB)
    - Different parts of virtual memory hold different parts of the program

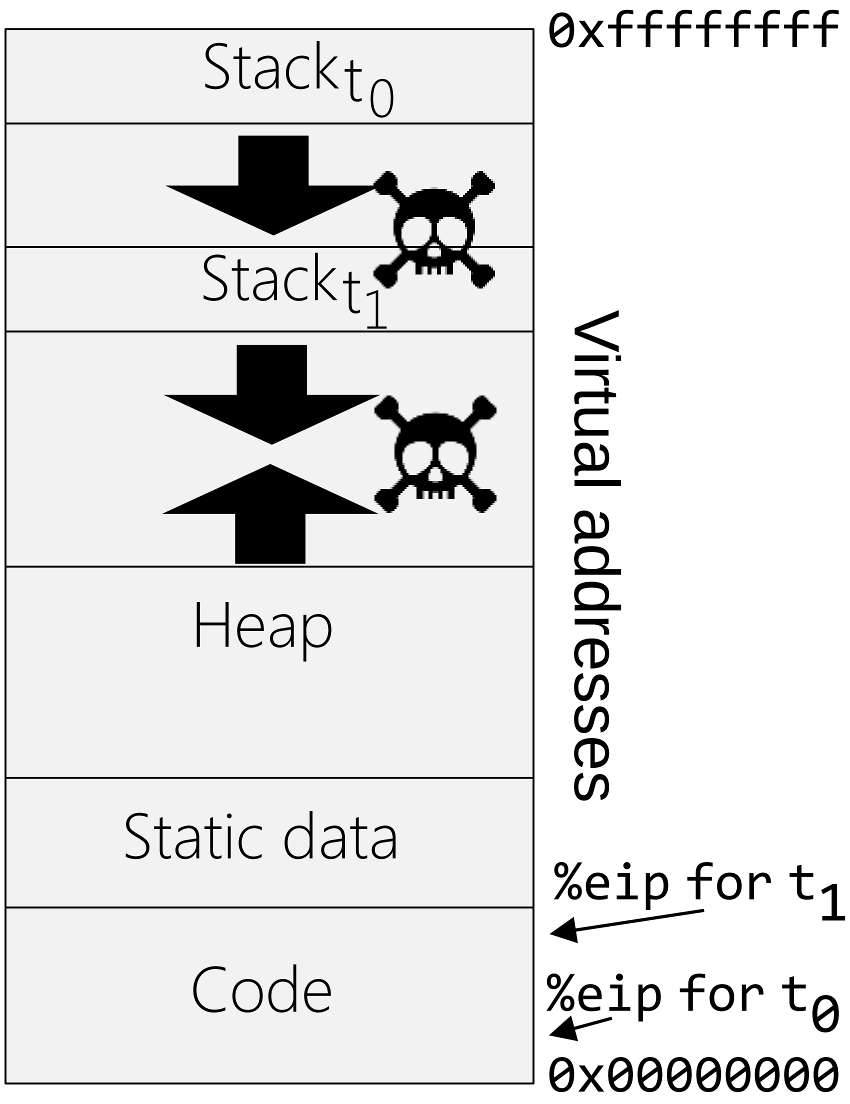
Stack

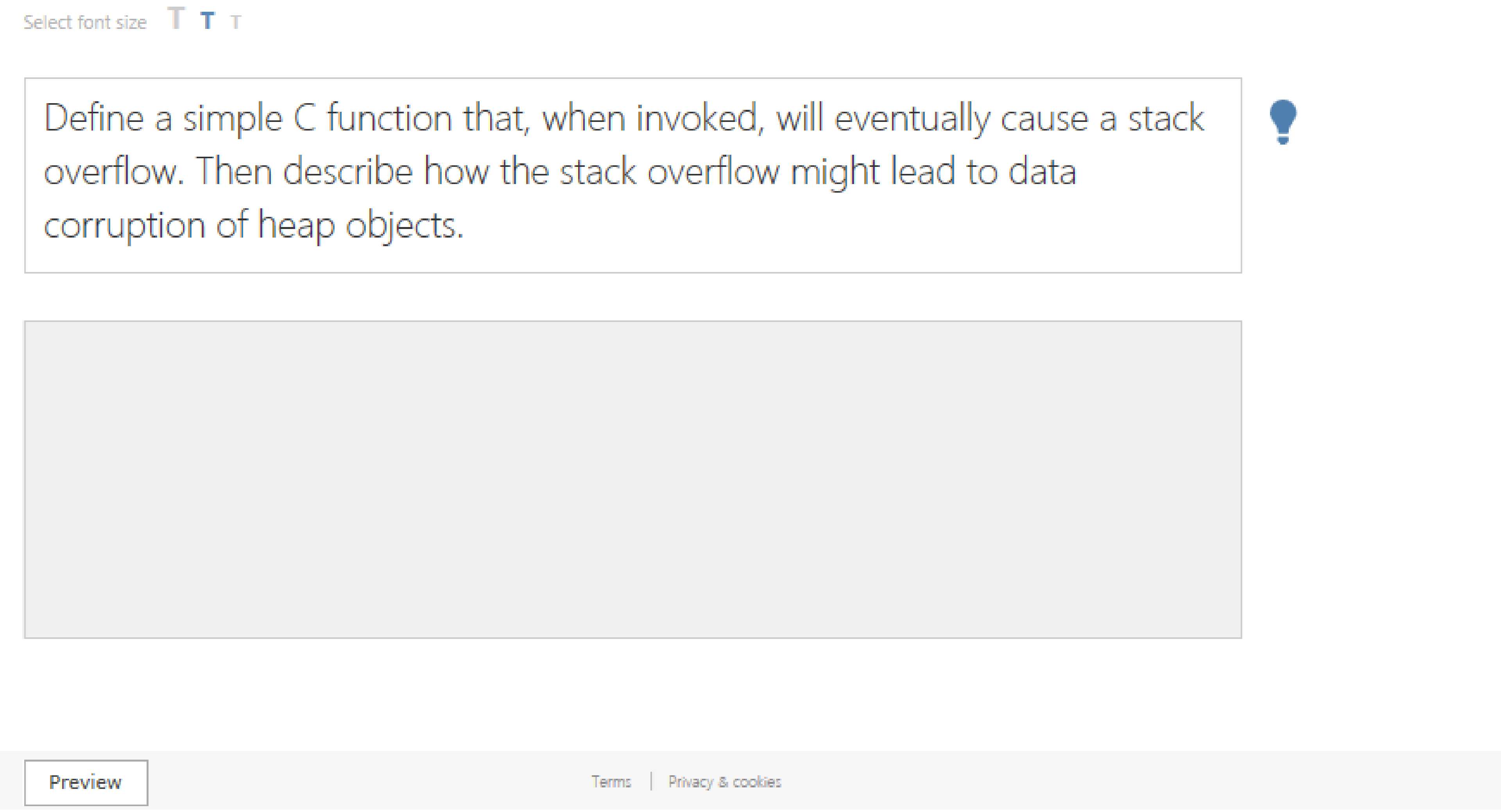
```
int qux(){return 42;}
int baz(){return qux();}
int bar(){return baz();}
int foo(){return bar();}

foo();
```



What's an Address Space?

- The set of virtual memory addresses that a process can access
  - A large array of bytes starting at 0 and going up to some large number (e.g., 4 GB)
  - Different parts of virtual memory hold different parts of the program
    - Multiple threads --> multiple stacks!





Define a simple C function that, when invoked, will eventually cause a stack overflow. Then describe how the stack overflow might lead to data corruption of heap objects.

What Processes Are Running Right Now?

On Linux, try "ps -ef | less":

```
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0  2015 ?        00:00:02 init [3]
root         2     1  0  2015 ?        00:00:00 [migration/0]
root         3     1  0  2015 ?        00:00:00 [ksoftirqd/0]
                                .
                                . //Many other processes!
                                .
cs161    21085 20995  0 23:43 pts/1    00:00:00 ps -ef
cs161    21086 20995  0 23:43 pts/1    00:00:00 less
```

We created these processes!

What Processes Are Running Right Now?

- On Windows, run the Process Explorer (Ctrl-Shift-Esc)

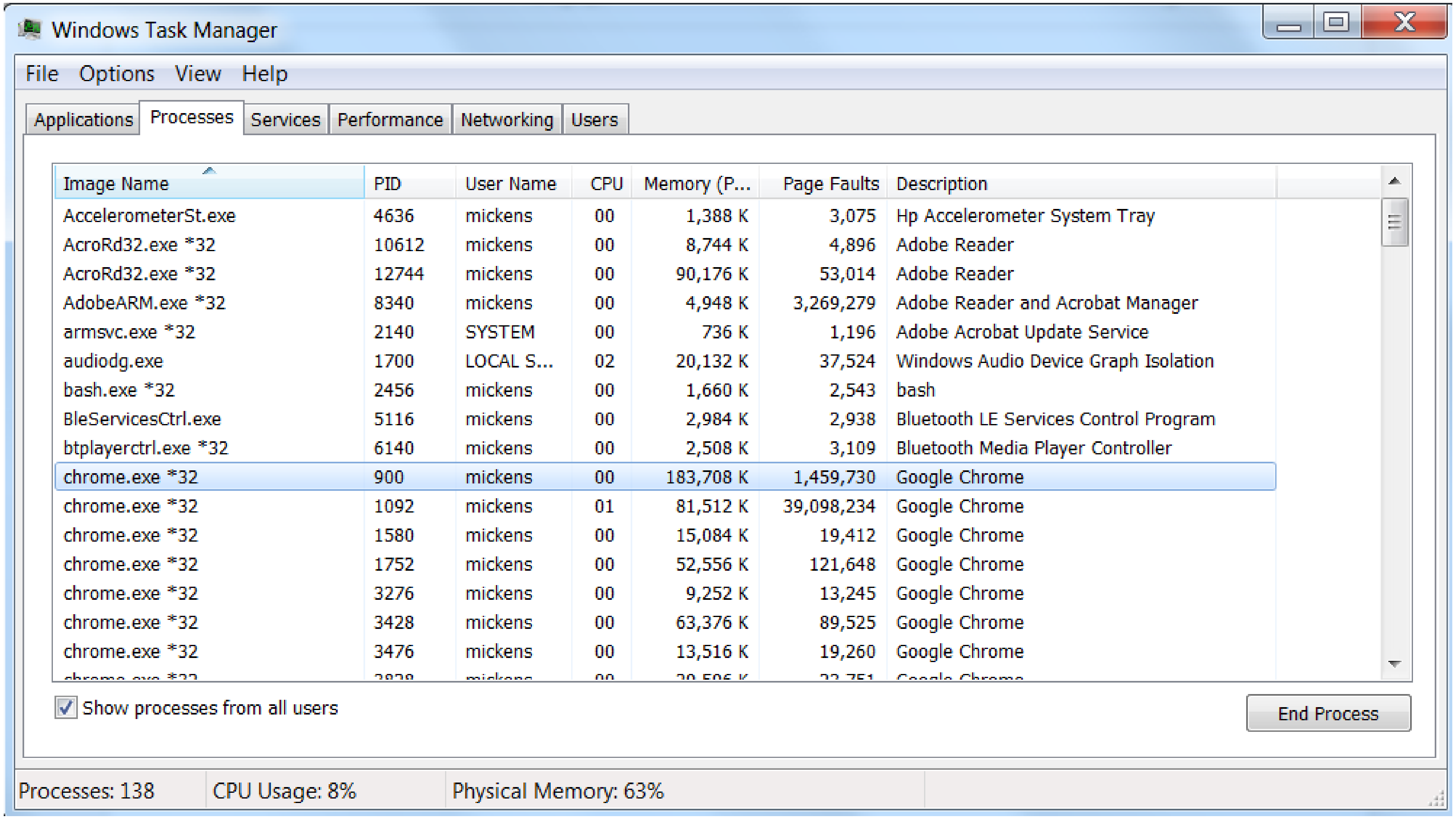

Windows Task Manager
File Options View Help
Applications Processes Services Performance Networking Users
End Process

Image Name	PID	User Name	CPU	Memory (P...	Page Faults	Description
AccelerometerSt.exe	4636	mickens	00	1,388 K	3,075	Hp Accelerometer System Tray
AcroRd32.exe *32	10612	mickens	00	8,744 K	4,896	Adobe Reader
AcroRd32.exe *32	12744	mickens	00	90,176 K	53,014	Adobe Reader
AdobeARM.exe *32	8340	mickens	00	4,948 K	3,269,279	Adobe Reader and Acrobat Manager
armsvc.exe *32	2140	SYSTEM	00	736 K	1,196	Adobe Acrobat Update Service
audiodg.exe	1700	LOCAL S...	02	20,132 K	37,524	Windows Audio Device Graph Isolation
bash.exe *32	2456	mickens	00	1,660 K	2,543	bash
BleServicesCtrl.exe	5116	mickens	00	2,984 K	2,938	Bluetooth LE Services Control Program
btplayerctrl.exe *32	6140	mickens	00	2,508 K	3,109	Bluetooth Media Player Controller
chrome.exe *32	900	mickens	00	183,708 K	1,459,730	Google Chrome
chrome.exe *32	1092	mickens	01	81,512 K	39,098,234	Google Chrome
chrome.exe *32	1580	mickens	00	15,084 K	19,412	Google Chrome
chrome.exe *32	1752	mickens	00	52,556 K	121,648	Google Chrome
chrome.exe *32	3276	mickens	00	9,252 K	13,245	Google Chrome
chrome.exe *32	3428	mickens	00	63,376 K	89,525	Google Chrome
chrome.exe *32	3476	mickens	00	13,516 K	19,260	Google Chrome

✓ Show processes from all users

Processes: 138 CPU Usage: 8% Physical Memory: 63%

What's up with Chrome?

- Chrome uses a multi-process architecture
  - One process per window
  - An additional process per tab
- Per-tab process renders HTML for that tab, sends bitmap to the main process for displaying . . .
  - . . . but has severely restricted system call privileges!
  - Per-tab process can't send network traffic, draw to screen, grab user input, or access persistent storage
  - To do so, must send IPC message (via a pipe) to main process
- Process isolation helps both security *and* robustness!
- Gory details: https://www.chromium.org/developers/design-documents/multi-process-architecture