Paging

- operating systems resides at the beginning of memory
- OS know exactly how much physical memory there is
- OS can address memory by its actual **physical address** (non-translated)

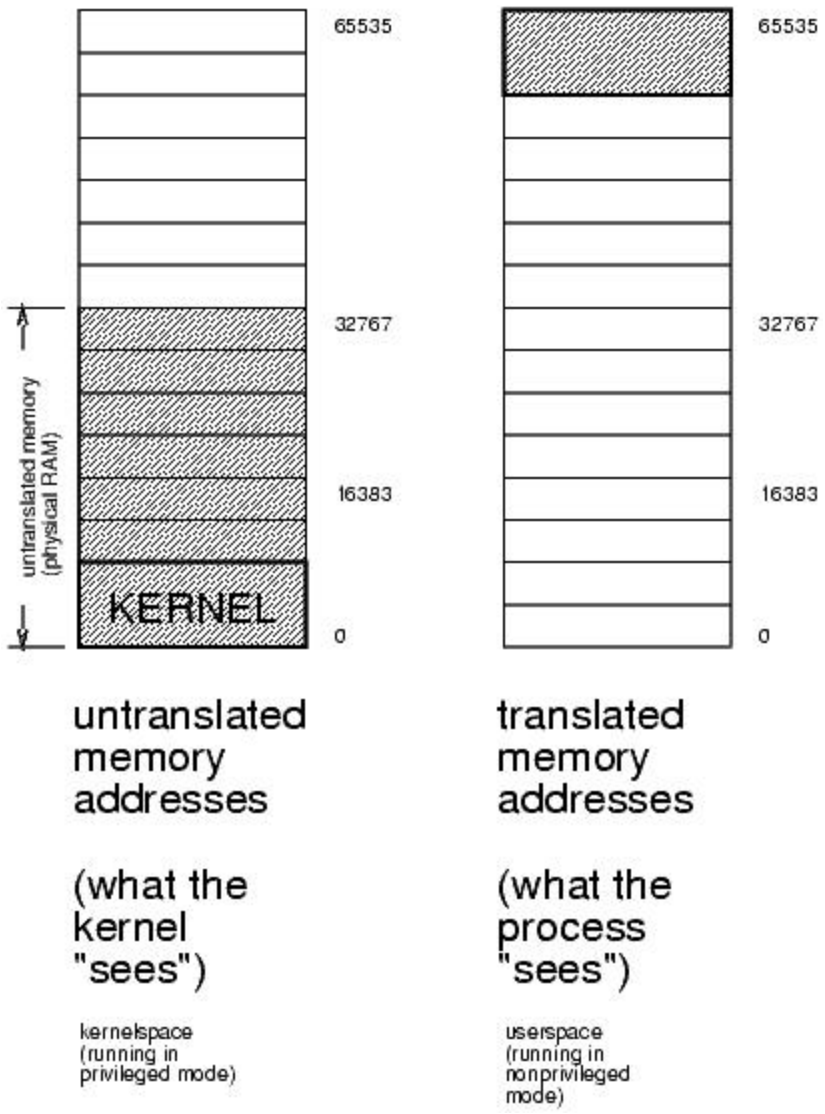
- processes running under an OS cannot see physical memory addresses
- address **translated** by the OS

result: each process sees itself
- at the **beginning** of memory
- as the **only process** in the memory

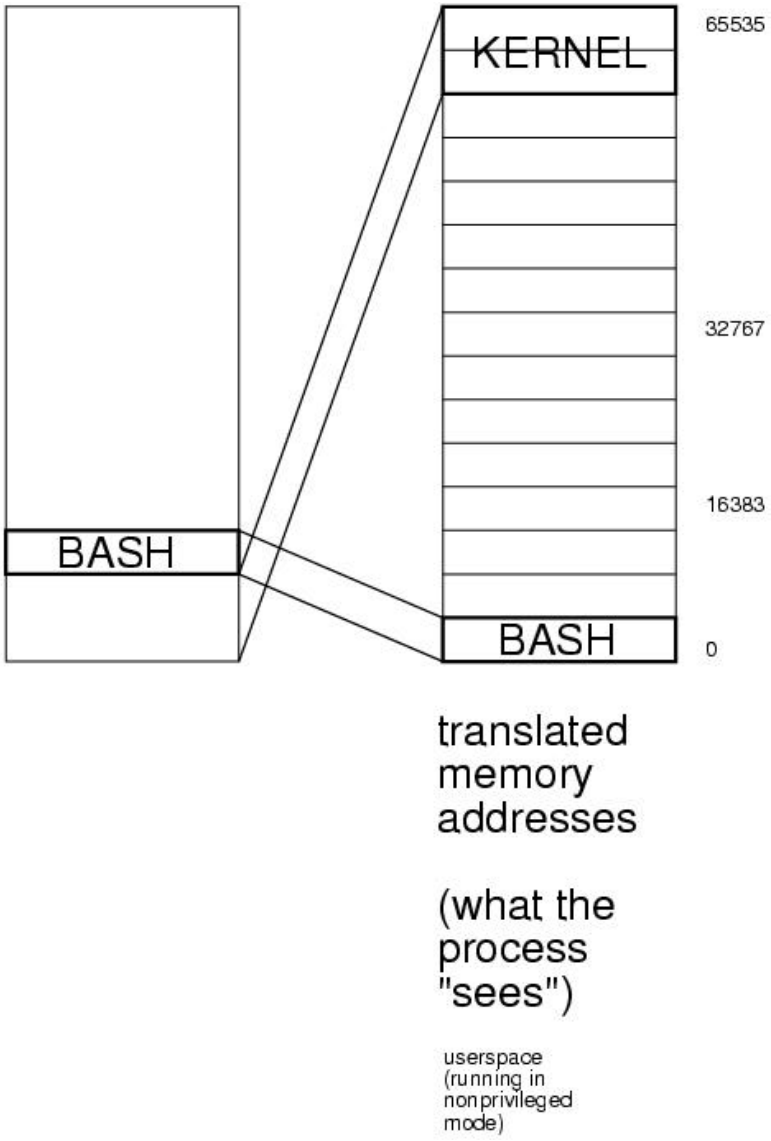
this greatly simplifies memory management from the program's view, can start addressing at #00000000 and OS takes care of translation

- each process sees **more** memory that it actually has (because it thinks it's the only one running)
- swap provides **extra** space (more on this later)
- capacity management is handled by the OS

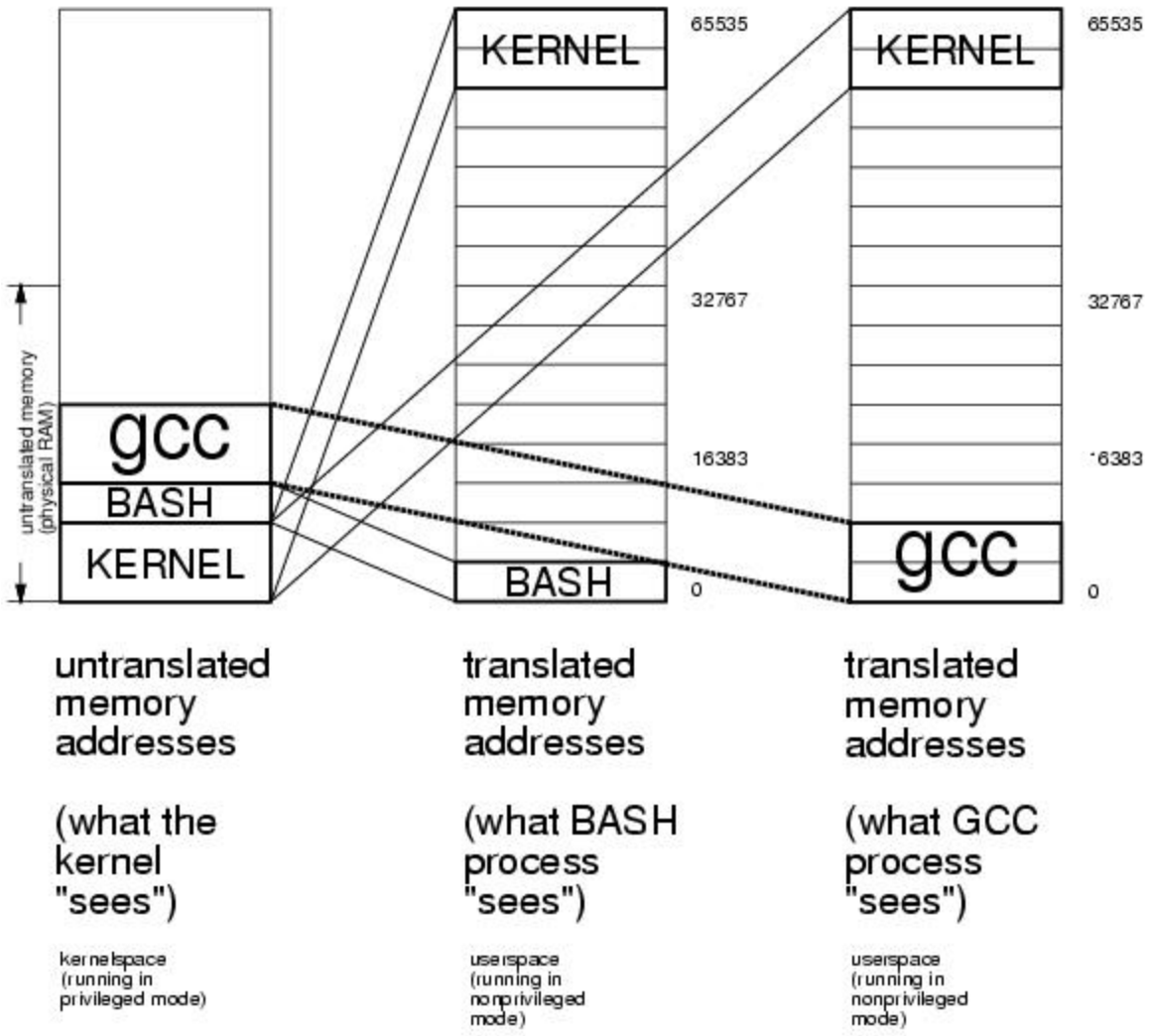
Examples (on Linux, diagrams by Prof. Mann)
For simplicity, assume kernel = OS



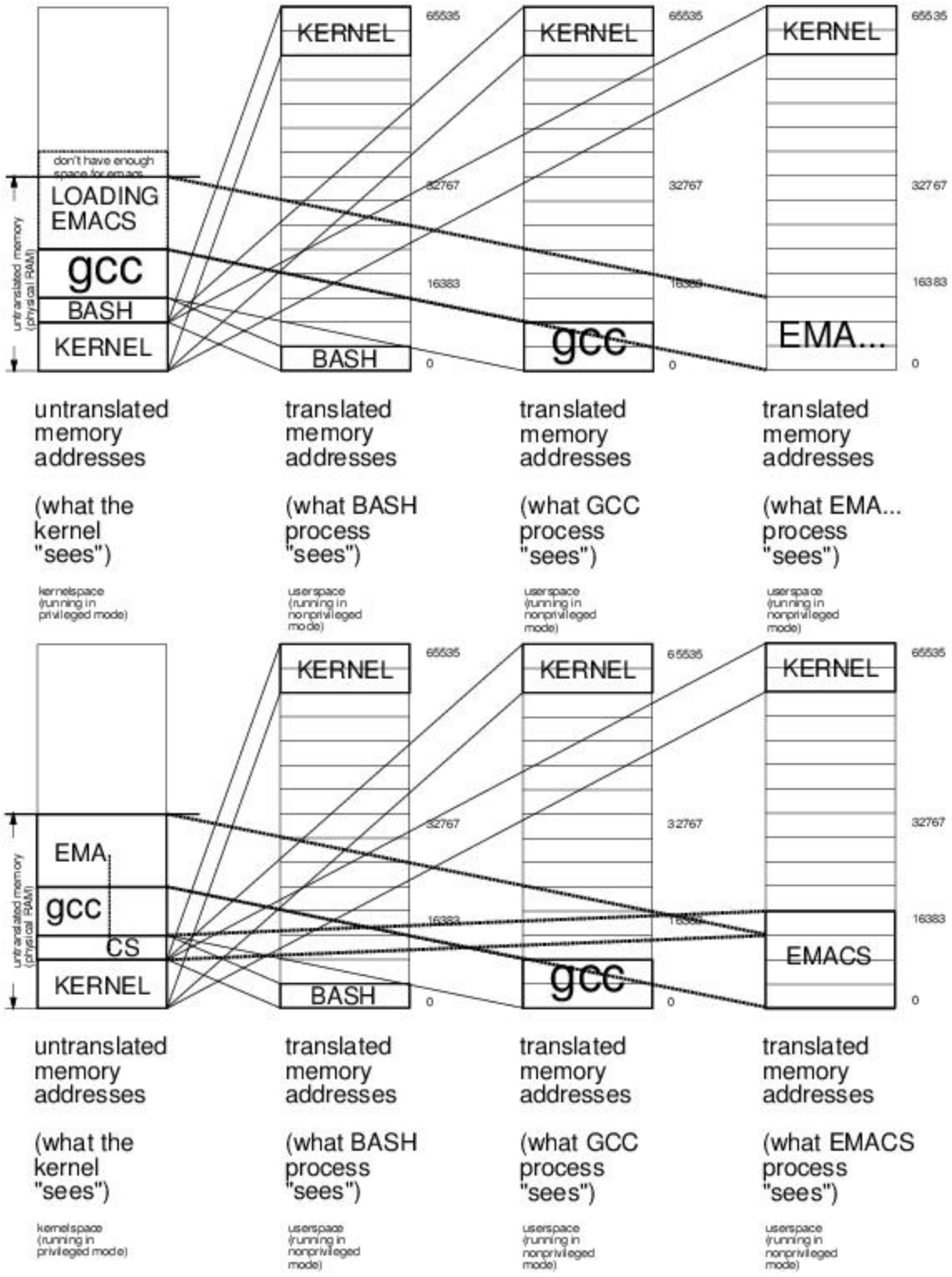
No processes are loaded, but the translation already set up, user programs see memory as empty, except the part taken by the OS, which appears at the end of available memory.



One process loaded, sees itself at the beginning, but in actuality loaded right after the kernel.



Some more processes…



No room for the Emacs process to load fully. It loads some, then swaps out Bash (which was accessed last) to disk, and loads the rest. Even though the memory space taken by the process is split, translation allows the process to think it's still all together.

This allows each process to have its own memory space, thus uncontrolled behavior can be contained.

Uncontrolled behavior by the operating system, on the other hand, makes for data loss and other mayhem, since it can access everything directly.

Now the details: which page to swap out? We got several possibilities…

First In First Out (FIFO)
- Store age of each page
- the page that has been in memory longest is the one offloaded

Good:
- straightforward to implement
- requiredslittle effort by the system to maintain age information (can just keep a counter, record and increment when page loaded)

Bad:
- say a page that has been loaded first is in heavy use
- it will be offloaded anyways, and will have to be reloaded almost right away, so performance suffers

Optimizing FIFO (Second Chance)

- Every time a page is used, set a signal bit to 1
- When trying to swap out, if bit is 0, swap it out, if 1, set to 0 and try next one

Good:
- frequently used pages will have the bit set to 1, and hopefully will not get swapped out

Bad:
- if all pages are in heavy use, system will have to try every single one
- some performance penalty to update the bit

Least Recently Used (LRU)

- record last time the page was used, i.e. update timestamp on each access

Good:
- in theory, more optimal than FIFO

Bad:
- often slows down the system so much, it's not worth it (can increase memory access time by factor of 5)

Real Life: UNIX *pagedaemon* uses a somewhat modified FIFO Second Chance.

Interrupts (hardware)

- 15 (or so) signal lines used by hardware to tell CPU they need attention
- Example: keyboard

trivial implementation of keyboard:
- ask every few ms if a key is pressed – wasteful, can miss keystroked
- this is known as polling, very wasteful CPU use

with interrupts:
- keyboard takes the keypress, and remembers it
- sets the interrupt, so the CPU can get the data asap

References:

http://www.wearcam.org/ece385/lecture7/pagedmemory.htm
http://home.lanet.lv/~sd70058/aboutos/node99.html