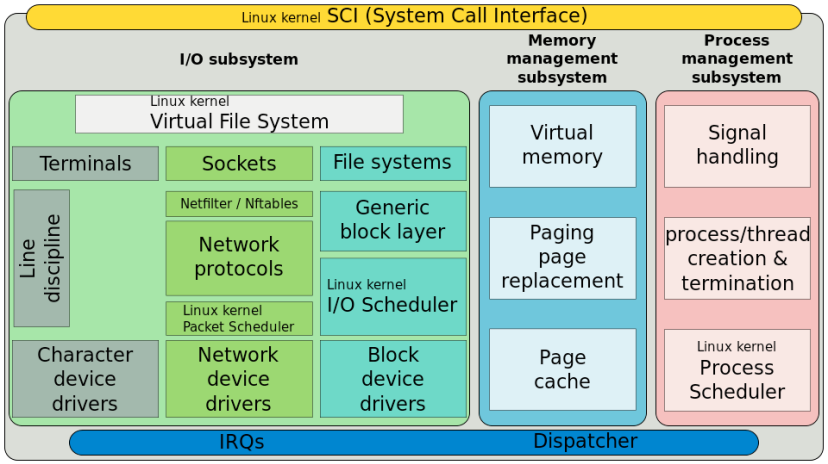
Operating systems

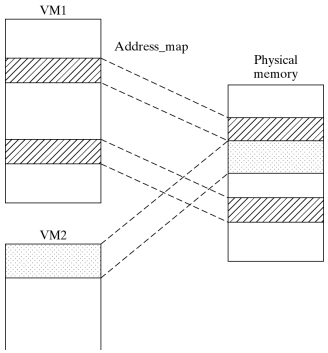
Memory: address translation



Memory management

- OS problem: how to implement virtual addressing?
- the process/mechanism: address translation
- heart of the problem: how to manage free memory?
- speeding things up: caching

Address translation

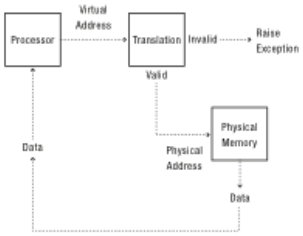


the conversion of

a virtual address to

a physical address

Goals of address translation



- transparency
- protection and isolation
- efficiency

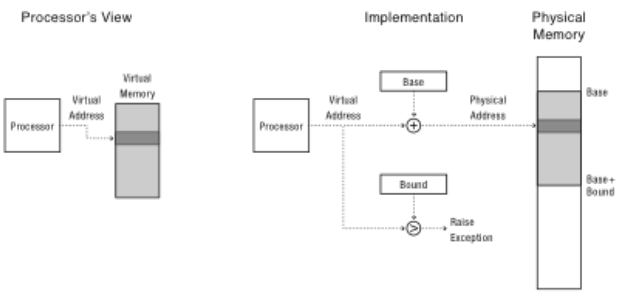
Multi-tasking problem

how to hold multiple processes in memory?

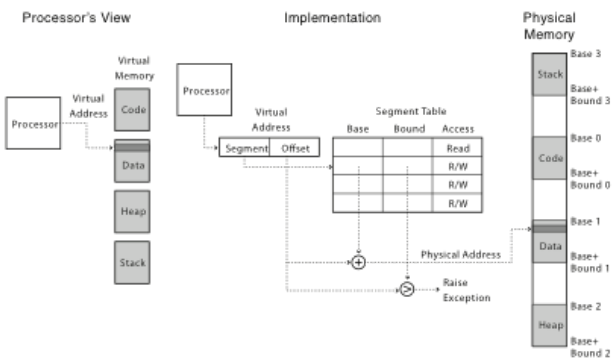
Relocation

once OS picks region in memory where program is to be stored, shift all absolute addresses by that base.

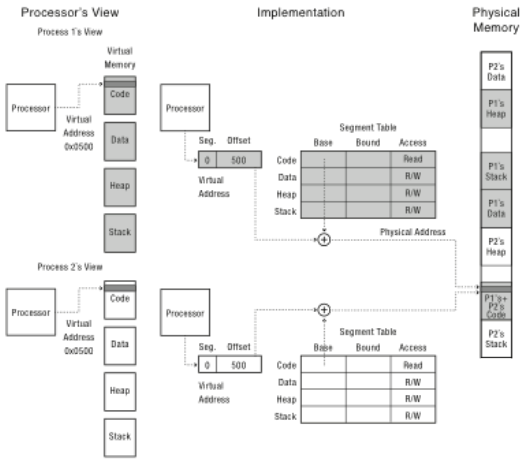
Hardware solution to protection base-and-bound



Segmented memory



Segmentation can be used to share code segments



How does the OS make sure that sensitive data from a previous process isn't accessible to new process assigned that memory?

Efficiently?

Segmentation can be used to do zero-on-reference

How big and how many segments?

fine-grained? ⟷ coarse-grained

more access control

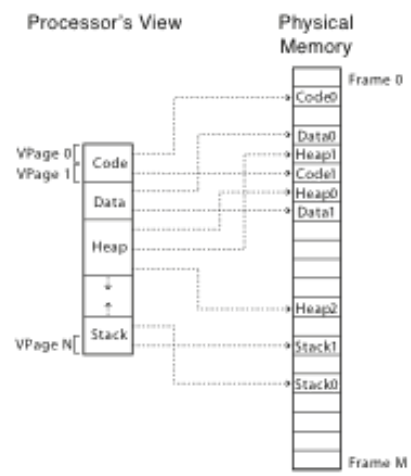
fine-grained sharing

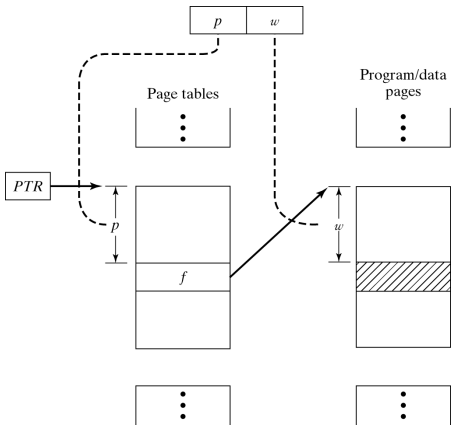
less fragmentation

less overhead

? external fragmentation

Paged memory





address translation is similar to segmented memory

Example

start of page table for current process

page table

...	...
1024	21504
1025	40960
1026	3072
1027	15360
...	...

if each page is 4KB, what physical address does virtual address (2,100) map to?

Page size

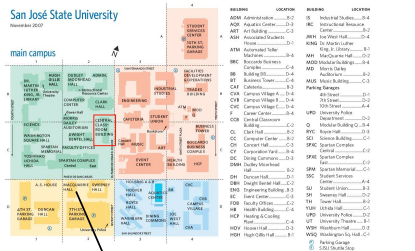
If each page is 4KB and addresses are 32-bit, how big is the page table?

Hint: one table entry per page, each entry is the address of the page frame.

small frames
-> large table

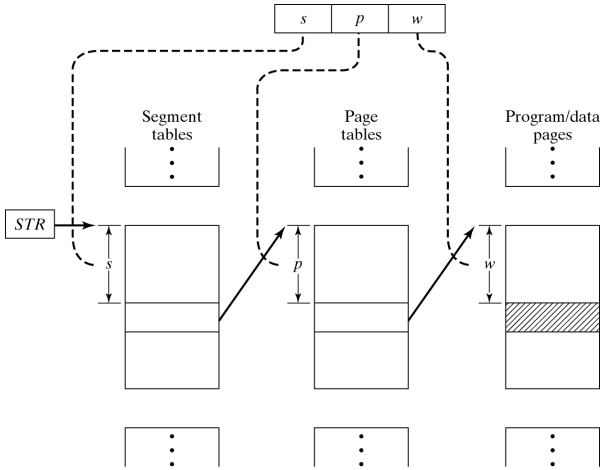
large frames
-> internal fragmentation

Multi-level translation

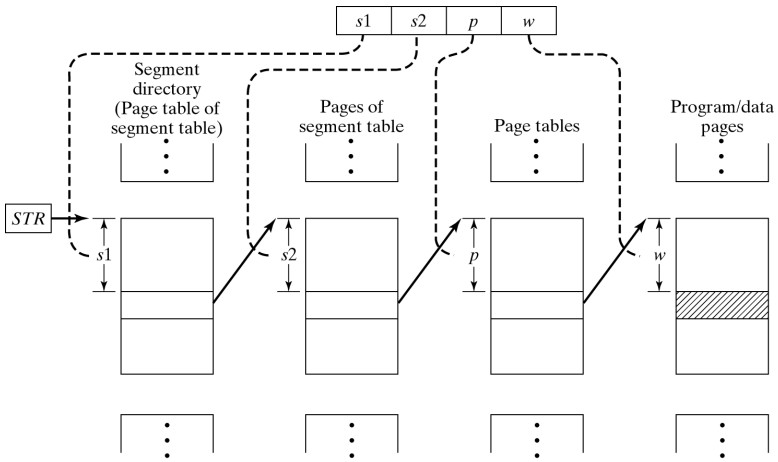


sparse adresses
-> tree based lookups

Paged segmentation

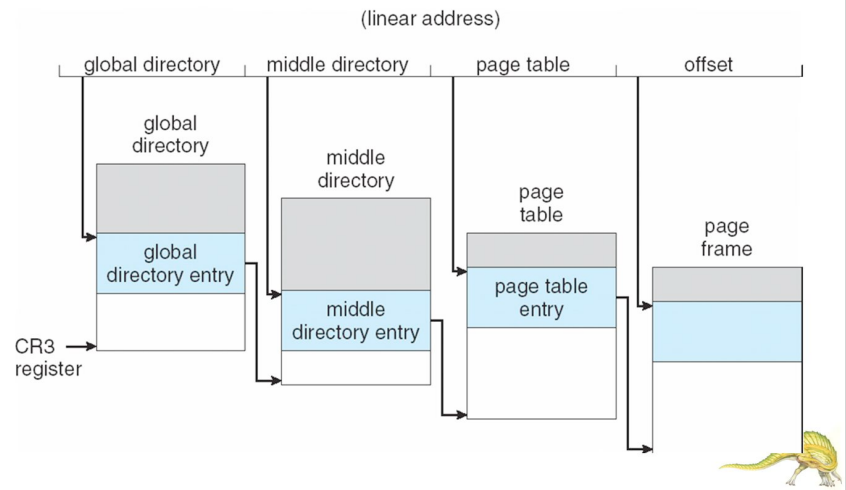


Multi-level paging

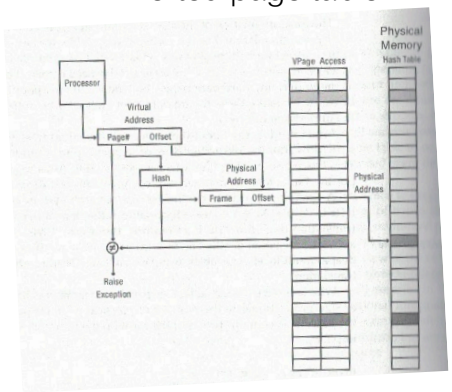


ex: Sun Microsystem SPARC processor

Three-level Paging in Linux



Inverted page table



What else is virtual addressing good for?

- efficient I/O
- memory mapped files
- virtual memory
- checkpointing and restart
- persistent data structures
- process migration
- information flow control
- distributed shared memory

Recap

- memory protection: hardware branch-and-bound
- flexible use of memory: segmenting and paging
- sparse addressing: tree-based or hash table lookups