Demand Paging

Address Translation Summary

- Protection

- Sharing

- Sparse addresses

- Efficiency

- Portability

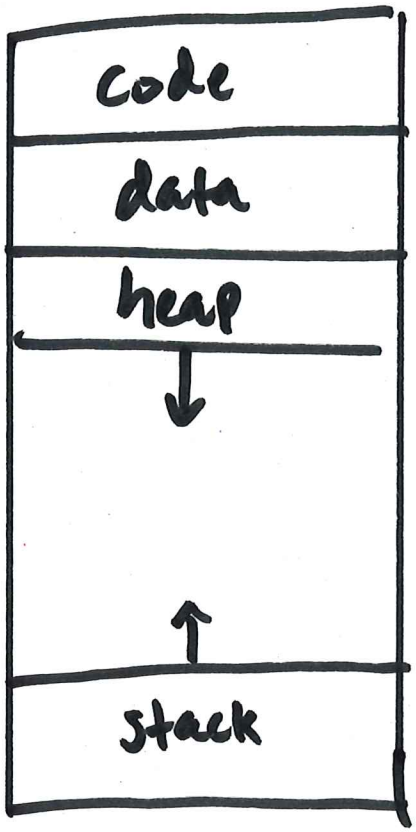
Additional features

Translation hardware notifies OS whenever a read/write occurs that is not mapped.

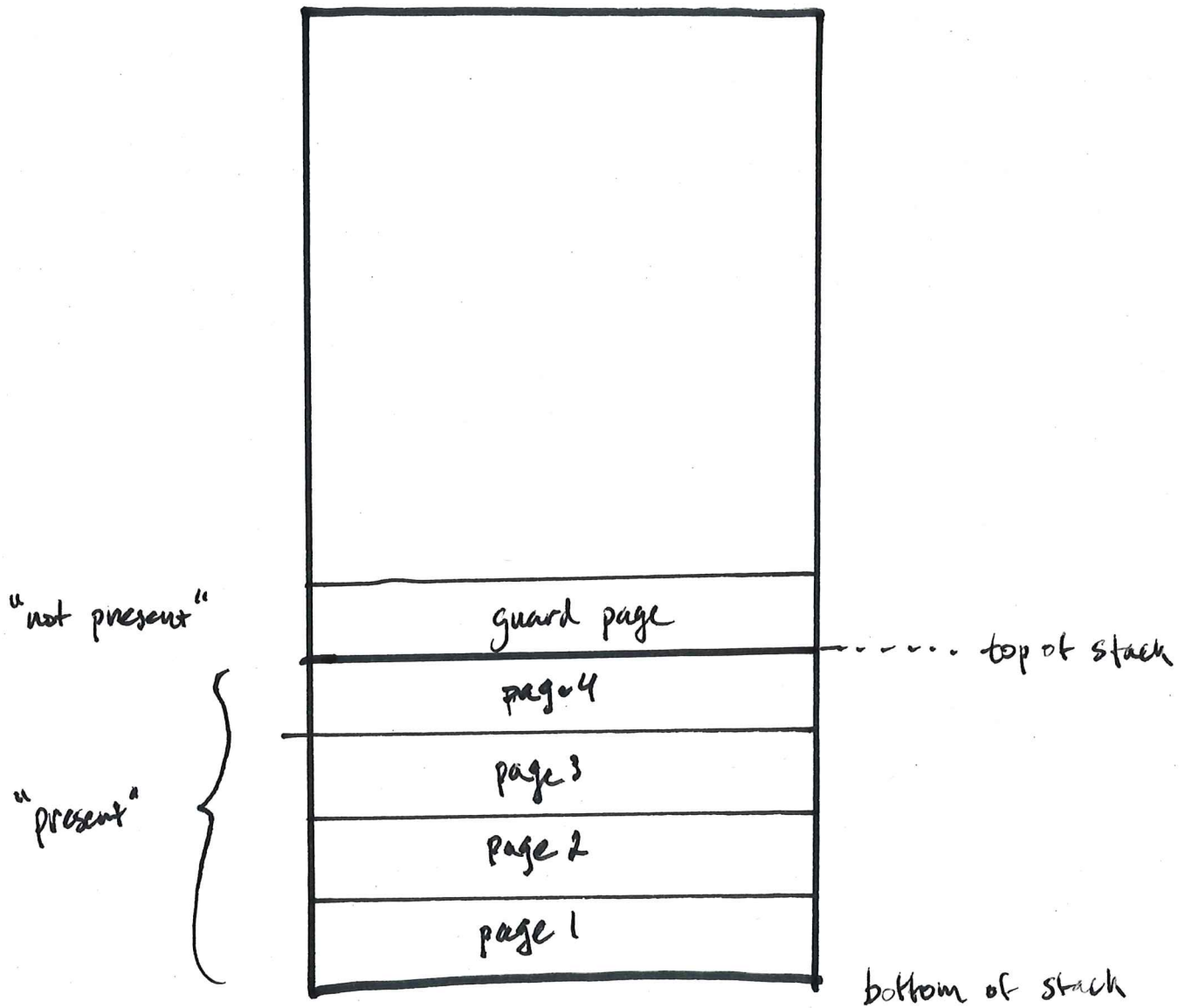
Can use this to do many things:

- copy-on-write
- zero-on-reference
- expanding stack
- demand paging
- memory mapped files
- ...

Idea: set up page tables so that fault handler gets called



How big should the stack be?


"not present"
guard page
top of stack
page 4
"present"
page 3
page 2
page 1
bottom of stack

Expanding Stack on reference

- steps:
  - set up guard page to cause page fault
  - later, page fault occurs
  - page fault handler detects that fault is to guard page
  - allocate new frame, map it to next page past top of stack
    - zero the page to avoid information leaks
  - return from fault handler, re-execute faulting instruction

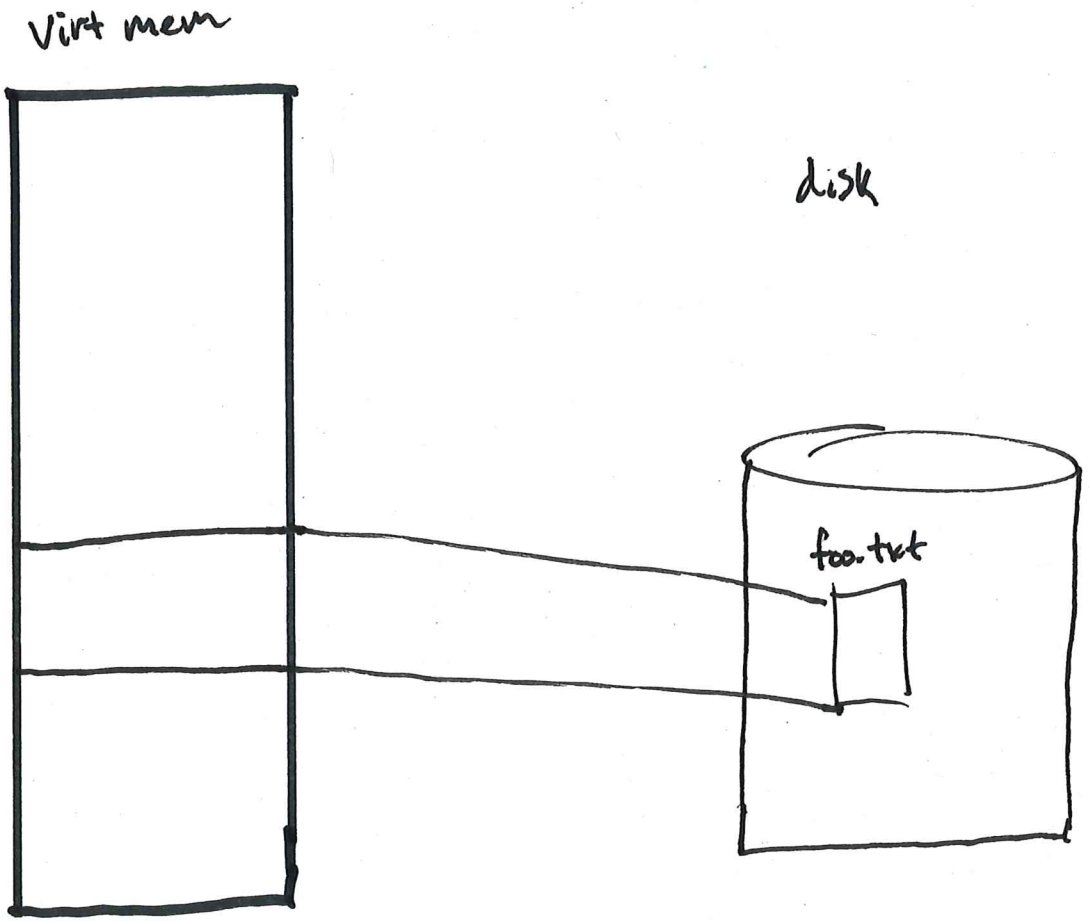
Fill on demand

Run a program without loading all its code

- mark any missing / not-yet-loaded pages "not present" in page table
- instruction fetch will cause page fault
- fault handler detects fault is on fill-on-demand page
- read missing page from ELF file
- update page table entry to mark page present
- resume process, re-execute faulting instruction

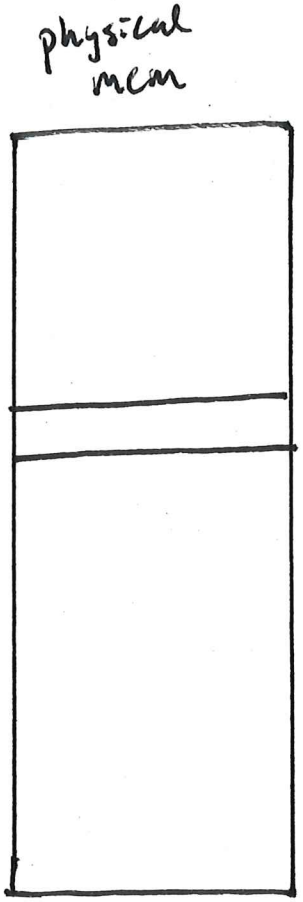
Memory-mapped files (mmap())

- Unix mmap maps a file to a region of virtual memory
- initially, all PTEs marked invalid. load from disk on first reference
- program can edit the file using normal loads/stores
- changes eventually written back to disk
  - changes detected using dirty bit in PTE



page table

pg#	frame	Access
A		
B		



Demand paged virtual memory

- Allow any page of proc's memory to be written out to disk

- Each page is either resident (in phys. mem) or non-resident (on disk)

- Reference to non-res. page causes it to be copied in
  - must find some frame to copy into
  - if no free frames (common), must evict some page