

memory management

Vaibhav Bajpai
OS 2013

motivation

virtualize resources:

- multiplex CPU (CPU scheduling)
- multiplex memory (memory management)

why manage memory?

- controlled overlap
 - processes should NOT overlap in physical memory
- translation
 - allows to give uniform view of memory to programs



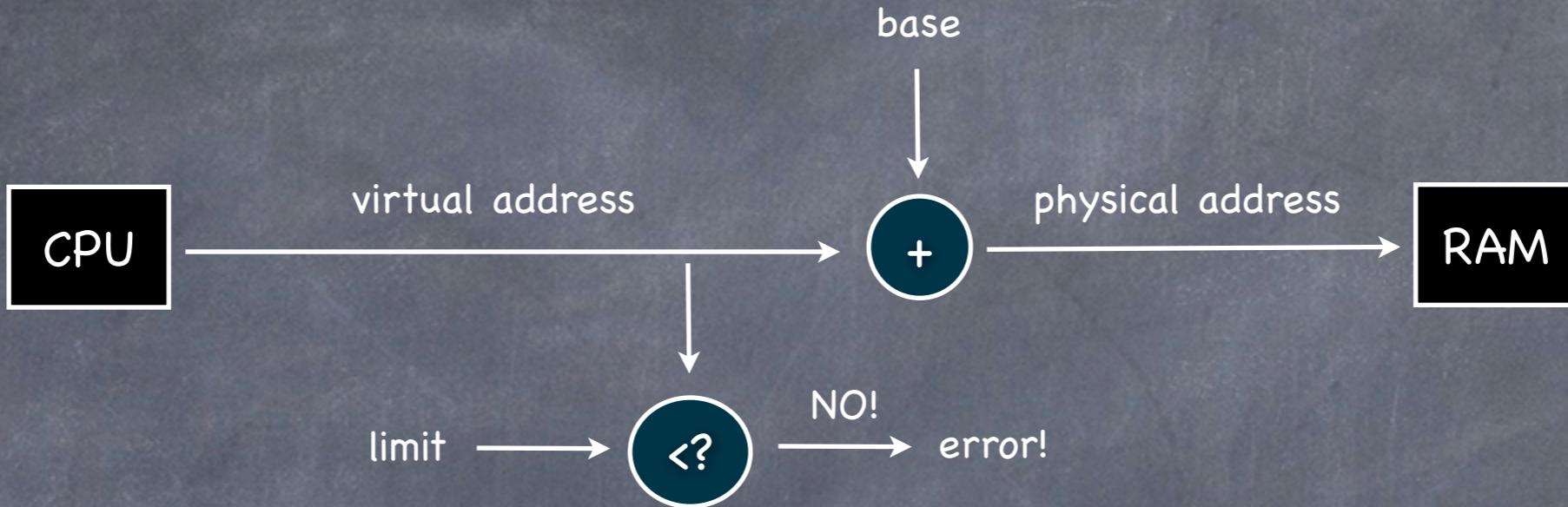
protection

- a process should NOT access other process' memory
- read-only memory locations:
(kernel, process code, et al.)

memory management

- ⦿ base-bound registers
- ⦿ segmentation
- ⦿ paging
- ⦿ segmentation + paging
- ⦿ multi-level page tables
- ⦿ inverted page tables

base and bound registers



- dynamic address translation:

- issues:
 - no inter-process sharing
 - cannot load a process bigger than an available hole
 - maybe load only a part of the process in memory?
 - heap and stack cannot grow within a process
 - not all processes are of same size: leads to external fragmentation

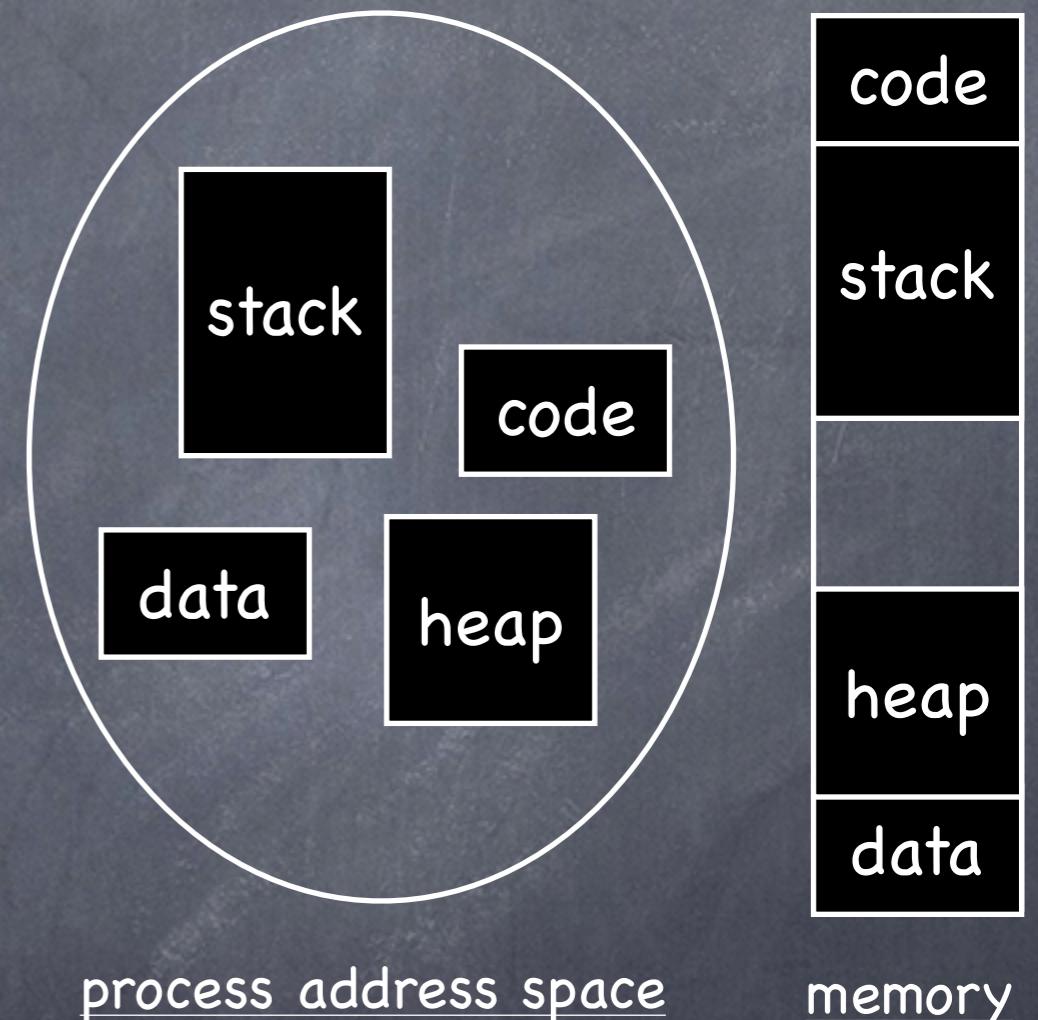
segmentation

- idea:

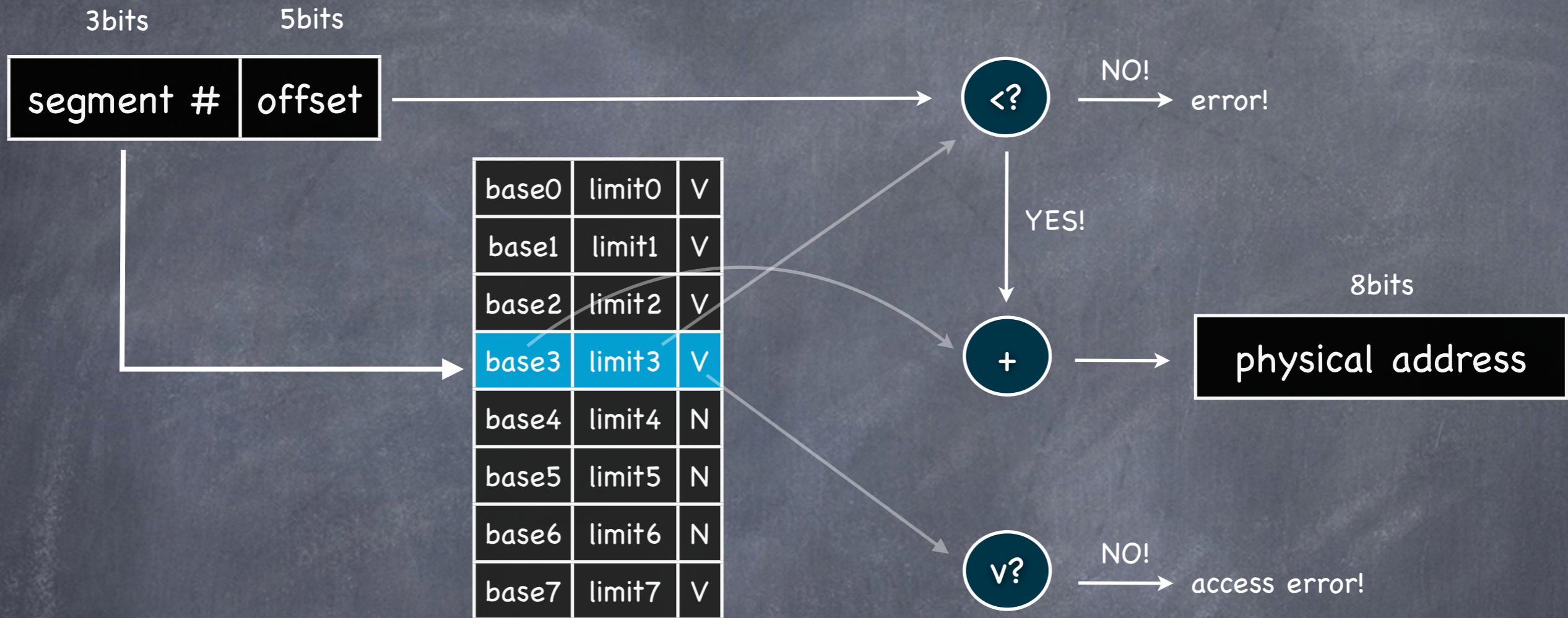
- divide each process into multiple segments each with their own BASE and LIMIT registers

- advantages:

- segments can be shared!
- a part of a process can also be in memory
- heap and stack can grow within a process



segmentation



- the segment table is stored in processor; not memory
- the segment table should be protected from processes
- issues:
 - each segment differs in size, leads to external fragmentation
 - swapping requires a complete segment to be written to disk

paging

- ⦿ idea:

- ⦿ divide the physical memory into fixed size chunks (pages)
- ⦿ use a bit vector to handle allocation

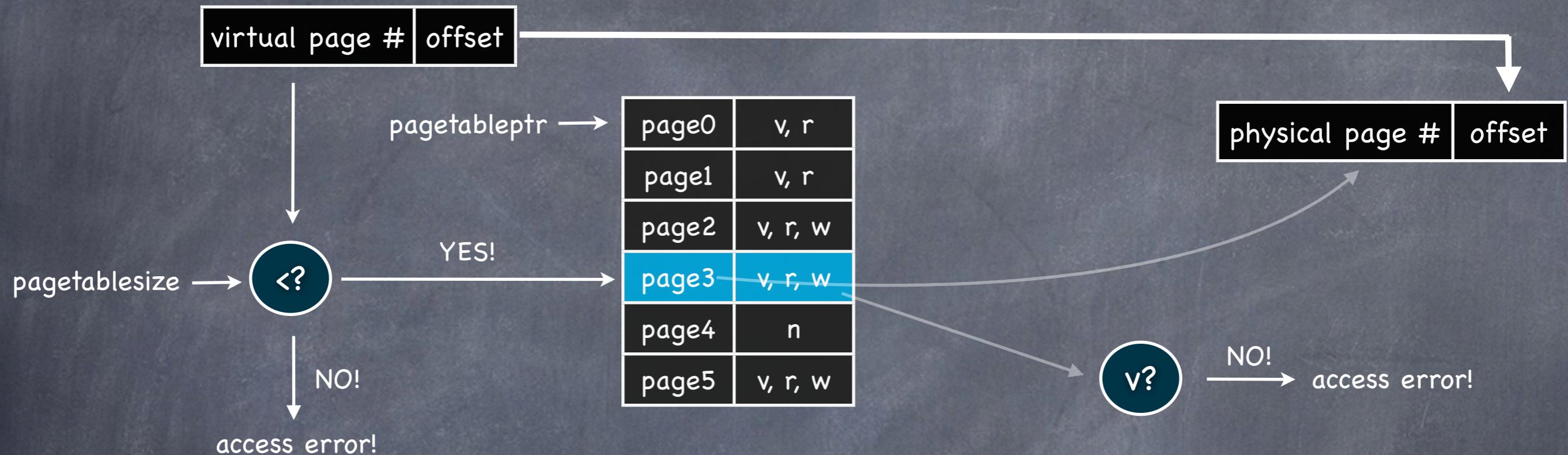
- ⦿ perspectives:

- ⦿ with 4GiB of physical memory, and 4KiB of pages, what should be the size of the bit vector? 220KiB
- ⦿ use small page size to avoid huge internal fragmentation

- ⦿ advantages:

- ⦿ no external fragmentation!
- ⦿ only small-sized pages need to be swapped to disk

paging



- there is a page table for each process
- page table resides in physical memory
- offset represents the size of a page;
 - for a 4K page, how many offset bits do we need? 12 bits
 - for a 4K page, for a 32 bit address space, how many page numbers are possible? 1m
 - why are we not checking limit for the offset? all pages are of same size!
 - can data structures be allocated that span more than a page? yes!
 - would it be contiguous in physical memory? not guaranteed!
 - why do we have a page table size? the page table grows ...
 - can stack/heap grow in this scheme? no! requires allocation of each page table entry

paging

process A

virtual page #	offset
----------------	--------

pagetableptr →

page0	v, r
page1	v, r
page2	v, r, w
page3	v, r, w
page4	n
page5	v, r, w

shared
page

virtual page #	offset
----------------	--------

process B

pagetableptr →

page0	v, r
page1	v, r
page2	v, r, w
page3	v, r, w

- sharing pages:

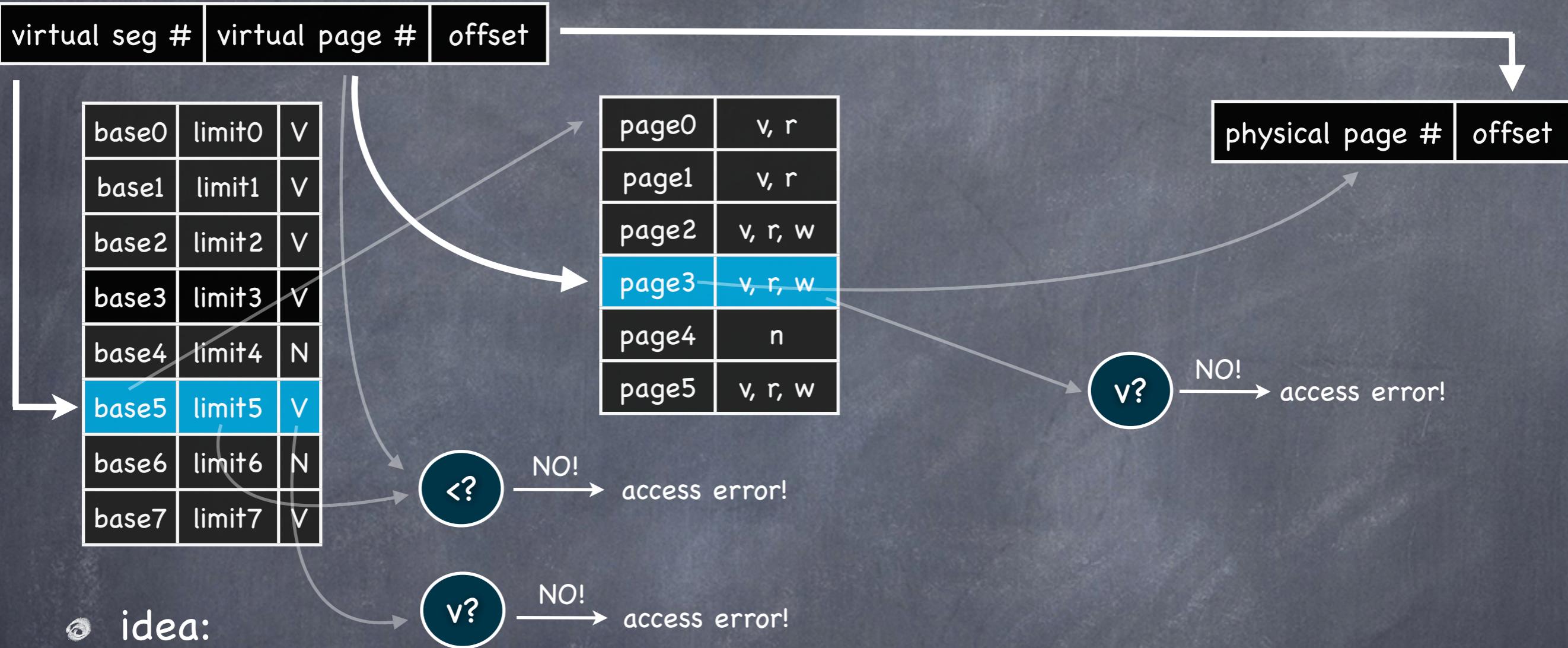
- process B can only read; however process A can read/write
- note: virtual address of a physical address can be different across processes
- what do we need on a process context switch?
 - page table pointer
 - page table size

paging

- ⦿ issues:

- ⦿ cannot handle sparse address space
 - ⦿ segmentation had this feature!
- ⦿ page tables reside in memory and they can get REALLY big!
 - ⦿ it's possible to swap out individual pages to disk
 - ⦿ would be nice to swap parts of page table to disk!

segmentation + paging



idea:

- create a multi-level translation
- lowest-level is a page-table
- higher-levels are segmented

advantages:

- whole segments can be shared!
- can handle sparse address space

- what do we need on a process context switch?
- highest level segment table

segmentation + paging

process A

virtual seg #	virtual page #	offset
---------------	----------------	--------

base0	limit0	V
base1	limit1	V
base2	limit2	V
base3	limit3	V
base4	limit4	N
base5	limit5	V
base6	limit6	N
base7	limit7	V

page0	v, r
page1	v, r
page2	v, r, w
page3	v, r, w
page4	n
page5	v, r, w

shared segment

base0	limit0	V
base1	limit1	V
base2	limit2	V
base3	limit3	V
base4	limit4	N
base5	limit5	V
base6	limit6	N
base7	limit7	V

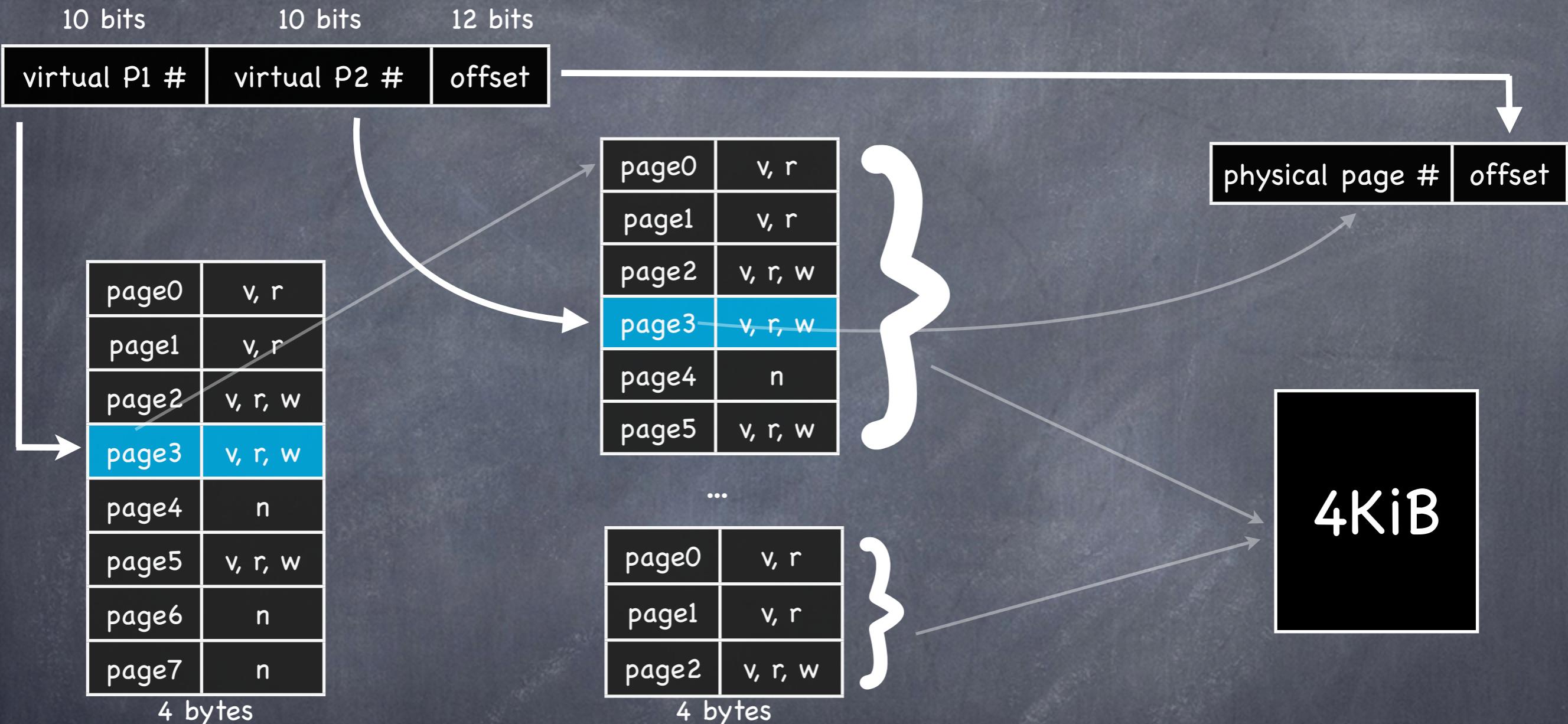
virtual seg #	virtual page #	offset
---------------	----------------	--------

process B

- sharing:

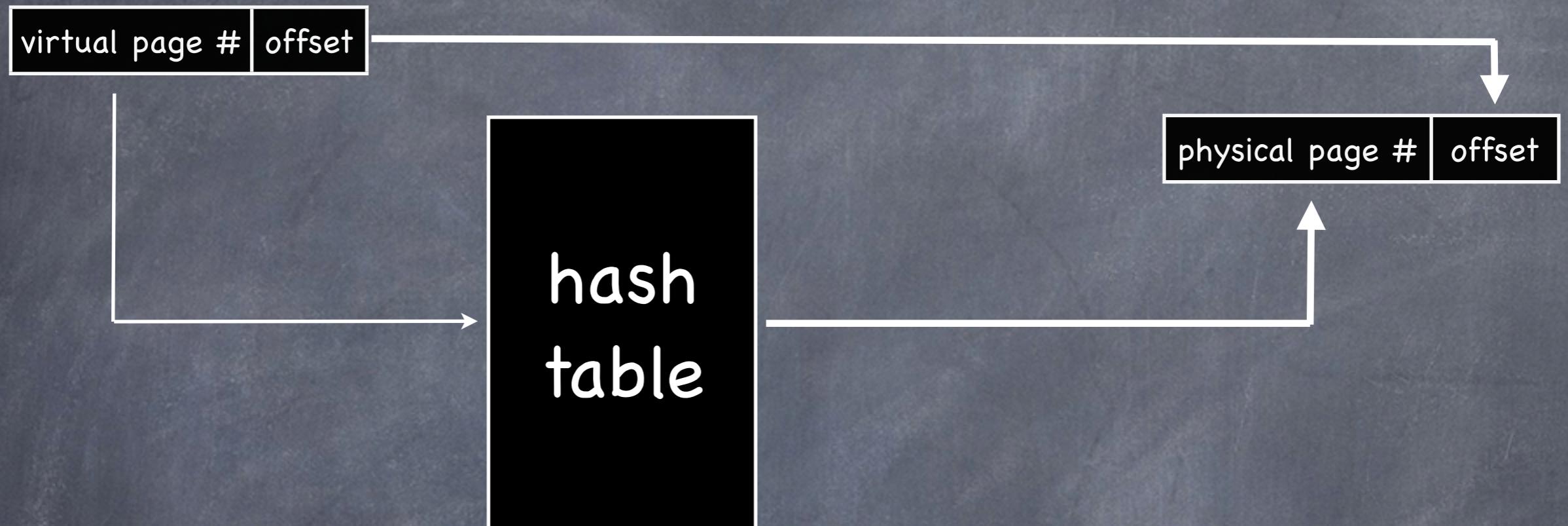
- whole segments can be shared!
- individual pages can be shared as well!

two-level page table



- all the levels of page table and pages themselves are 4K!
- allows to transparently swap out parts of the page table!
- issues:
 - two (or more) level lookups become expensive (use a TLB)

inverted page table



- page table size becomes independent of virtual address space
 - becomes directly related to the amount of physical memory
 - very attractive for 64-bit virtual address spaces
-
- issues:
 - complexity of managing a hash table in hardware

summary

⦿ base-bound registers

- ⦿ complete process contiguously loaded in physical memory
- ⦿ external fragmentation, no inter-process sharing

⦿ segmentation

- ⦿ part of a process can be loaded into memory
- ⦿ external fragmentation, large segments swapped to disk

⦿ paging

- ⦿ no external fragmentation, flexible sharing
- ⦿ cannot handle sparse addresses, page tables can grow BIG!

⦿ segmentation + paging

- ⦿ can handle sparse addresses
- ⦿ complete segments or individual pages can be shared

⦿ two-level page tables

- ⦿ portion of the page-table can be swapped to disk

⦿ inverted page tables

- ⦿ page-table size directly related to size of physical memory
- ⦿ complexity to manage hash table in hardware