National Yang Ming Chiao Tung University
Computer Architecture & System Lab

Paging

IOC5226 Operating System Capstone

Tsung Tai Yeh
Department of Computer Science
National Yang Ming Chiao Tung University

Acknowledgements and Disclaimer

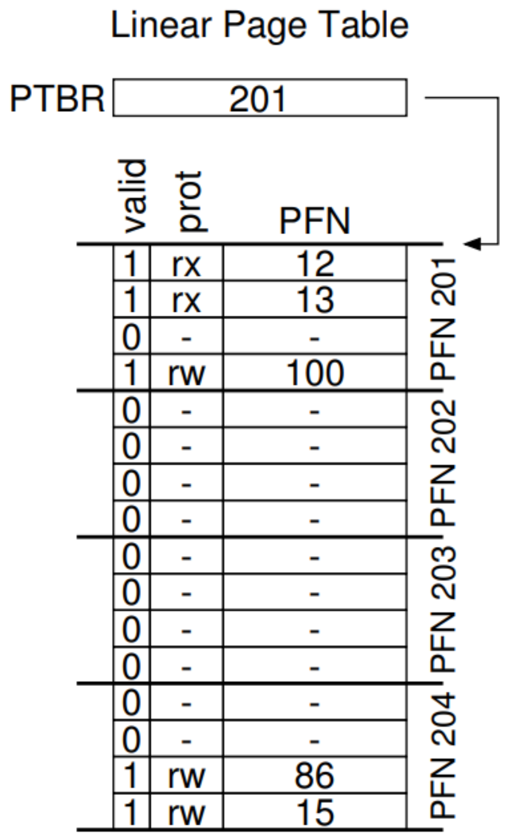
- Slides were developed in the reference with
  - MIT 6.828 Operating system engineering class, 2018
  - MIT 6.004 Operating system, 2018
  - Remzi H. Arpaci-Dusseau etl. , Operating systems: Three easy pieces. WISC

Outline

- Paging
  - Multi-level page table
  - Demand paging
  - The inverted page table
  - Page sharing

Page table (linear structure)

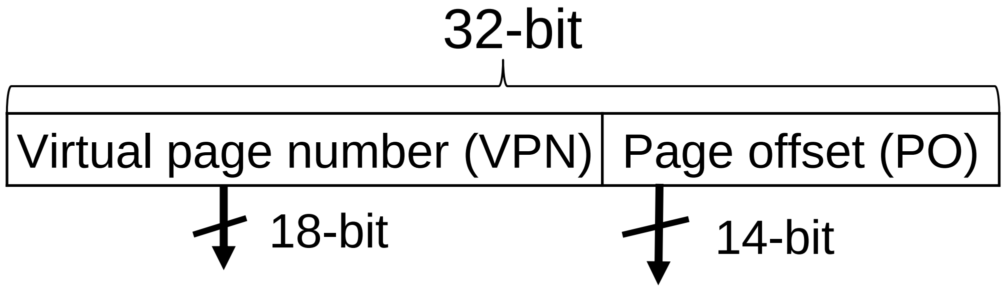
- Page table (linear structure) can be vary large !
    - 32-bit address (2^{32} bytes), 4KB (2^{12} bytes) pages, 4B PT entry
    - The number of page is ($2^{32}/2^{12} = 2^{20}$),
    - One page table size is 2^{20} x 4 bytes = 4MB per process
    - Hundreds of processes -> Hundreds of MB for PT



valid	prot	PFN	
1	rx	12	PFN 201
1	rx	13	
0	-	-	
1	rw	100	
0	-	-	PFN 202
0	-	-	
0	-	-	
0	-	-	
0	-	-	PFN 203
0	-	-	
0	-	-	
0	-	-	
0	-	-	PFN 204
0	-	-	
1	rw	86	
1	rw	15	

Bigger pages

- A 32-bit address, but increase page size from 4KB to 16KB
  - Each PTE is 4 bytes and now we have 2^{18} entries in our page table
  - The total size of a page table is (2^{18} x 4 bytes = 1MB)
  - The page table size (4MB in case 4 KB page size)
- What problems are shown with this approach ?
  - **Internal fragmentation** (big pages lead to waste within each page)



Page size

- **Arguments for larger page size**
  - Leads to a smaller page table
  - May be more efficient for disk access (block size of disk)
  - Larger page size – TLB entries capture more addresses per entry (why?), so there are fewer misses, with the “right” locality
  - X86 page sizes: 4KB, 2MB, 4MB …
- **Arguments for smaller page size**
  - Conserve storage space – less fragmentation (why?)

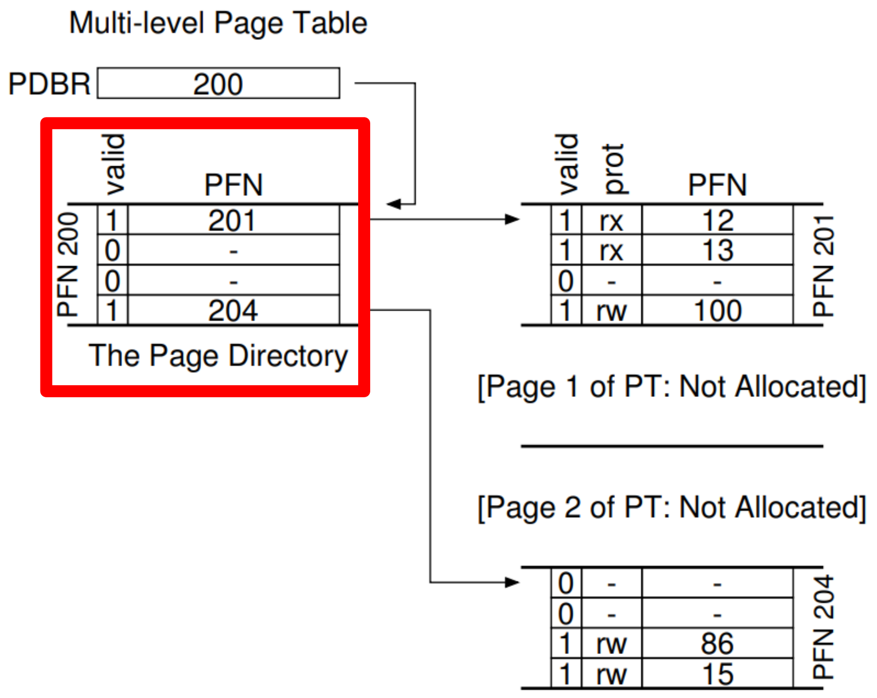
https://people.cs.pitt.edu/~childers/CS2410/slides/lect-virtual-memory.pdf

Multi-level page tables

- Turn page table into a tree (hierarchy) structure
  - Aim to reduce the memory space requirement to store page table
  - <u>Divide page table (PT) into page sized chunks</u>
  - <u>Hold only the part of PT where PT entries are valid</u>
  - Directory points to portions of the PT
  - Directory says where to find PT or that chunk is invalid

Multi-level page table (cont.)

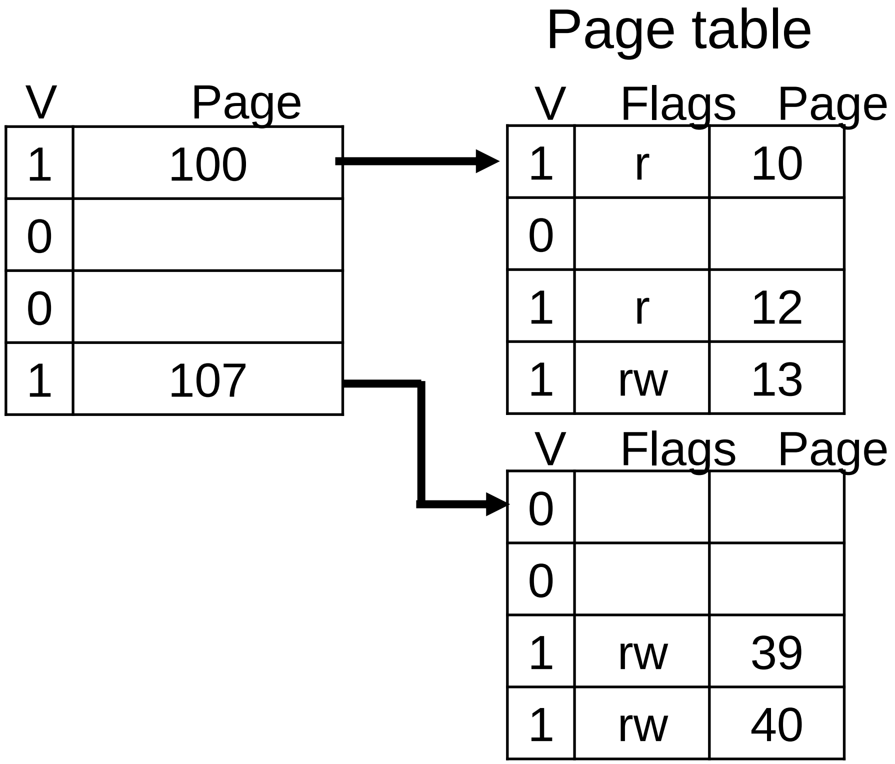
- **Multi-level page table**
  - Chop up the page table into page-sized units
  - **Page directory** tells where a page of the page table is
    - A number of **page directory entries (PDE)**
    - A **page frame number (PFN),** and a valid bit



https://pages.cs.wisc.edu/~remzi/OSTEP/vm-smalltables.pdf

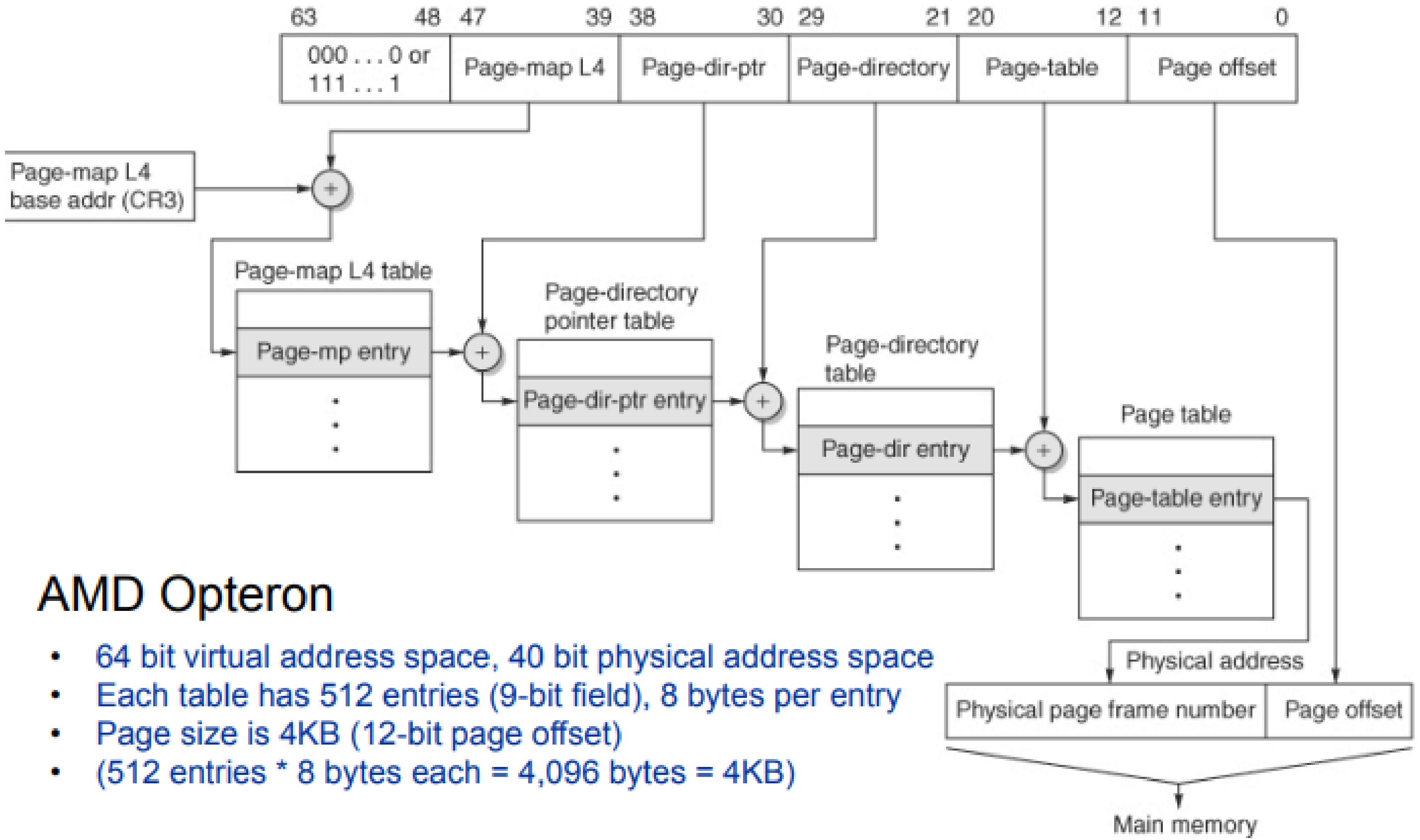
Multi-level page table (cont.)

- What are the advantages of multi-level page table?
  - Only allocate "using" page-table space
  - Compact and supports **sparse** address space



http://people.ee.duke.edu/~sorin/prior-courses/ece152-spring2008/lectures/6.9-memory.pdf

Multi-level paging table (cont.)



AMD Opteron

- 64 bit virtual address space, 40 bit physical address space
- Each table has 512 entries (9-bit field), 8 bytes per entry
- Page size is 4KB (12-bit page offset)
- (512 entries * 8 bytes each = 4,096 bytes = 4KB)

Important formula for paging table

- Number of entries in page table
  - (virtual address space size) / (page size) = number of pages
- Virtual address space size
  - 2^n Bytes
- Size of page table
  - (Number of entries in page table) x (size of PTE)

Case study of multi-level paging table

- How many levels of page tables would be required ?
  - A virtual memory system with physical memory of 8 GB, a page size of 8 KB, 46 bit virtual address, and PTE size is 4 B
- Initially
  - Page size = 8 KB = 2^{13} B
  - Virtual address space size = 2^{46} B
  - PTE = 4 B = 2^{2} B
  - Number of pages or number of entries in page table = 2^{46} B / 2^{13} B = 2^{33}
  - Size of page table = 2^{33} x 2^{2} B = 2^{35} B

Case study of multi-level paging table (cont.)

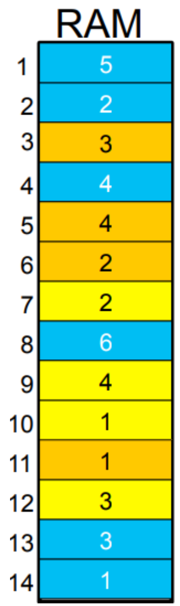
- How many levels of page tables would be required ?
  - A virtual memory system with physical memory of 8 GB, a page size of 8 KB, 46 bit virtual address, and PTE size is 4 B
- Now, size of page table > page size (2^{35} B > 2^{13} B)
  - Create one more level
  - Number of page tables in last level
    2^{35} B / 2^{13} B = 2^{22}
  - Size of page table [second last level]
  - 2^{22} x 2^{2} B = 2^{24} B

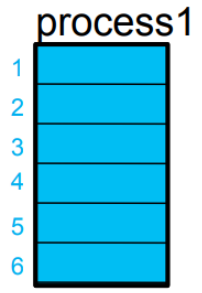
Case study of multi-level paging table (cont.)

- How many levels of page tables would be required ?
  - A virtual memory system with physical memory of 8 GB, a page size of 8 KB, 46 bit virtual address, and PTE size is 4 B
- Now, size of page table > page size (2^{24} B > 2^{13} B)
  - Create one more level [third last level]
  - Number of page tables in second last level
    = 2^{24} B / 2^{13} B = 2^{11}
  - Size of page table [third last level]=
    = 2^{11} x 2^{2} B = 2^{13} B = page size

Virtual memory

- Do we need to load all blocks of the page table into main memory before the process starts executing ?
    - **No !!**
    - Some code may not even be executed
- How to reduce the loading of unnecessary pages ?





process page table

block	page frame
1	14
2	2
3	13
4	4
5	1
6	8

Demand paging

- **Demand paging**
  - Pages are loaded from disk to RAM, **only when needed**
- **Why demand paging ?**
  - Reducing I/O
  - More users and decrease the number of memory requests
- **Pure demand paging**
  - When no pages are loaded into memory initially, pages generates page faults. No prediction !!
- **Pre-paging**
  - Predict which pages will be used and swap them into the RAM

Demand paging

- **How does demand paging work ?**
  - Using **present bit** in process page table to indicate if the block is in RAM or not
  - The process issues a page fault interrupt if an accessed page is not present in RAM
  - The OS loads the page into RAM and mark the present bit to 1
  - The OS removes another block from RAM if no pages on RAM are free

Demand paging implementation

- Keep copy of process's pages on disk, some are in RAM
  - Extend page table to include **"present" and "valid"**
  - Access to "page out" data by using **trap**
- **Inside the trap handler**
  - Access pages that are temporarily paged out to disk
  - Allocate a physical page frame to hold contents (might require another page to be paged out)
  - Copy data from disk to allocated page frame **(slow!!)**
  - Update page table entry
  - OS schedules processes

Effective access times (EAT)

- **EAT** is used to **measure the performance of demand paging**
- **Parameter**
  - p: page fault rate; ma: memory access time; pft: page fault time
  - EAT: $(1 – p) \times ma + p \times pft$
- Discussion
  - <u>The EAT is proportional to page fault time</u>

Effective access times (EAT)

- **What is average access latency ?**
  - L1 cache: 2 cycles
  - L2 cache: 10 cycles
  - Main memory: 150 cycles
  - Disk: 10 ms -> 30, 000, 000 cycles on 3.0 GHz processor
- **Assume access having following characteristics:**
  - 98% handled by L1 cache
  - 1% handled by L2 cache
  - 0.99% handled by DRAM
  - 0.01% cause page fault
  - What’s the average access latency ?

Effective access times (EAT)

- **Assume access having following characteristics:**
  - 98% handled by L1 cache
  - 1% handled by L2 cache
  - 0.99% handled by DRAM
  - 0.01% cause page fault
  - What’s the average access latency ?
- **Average access latency:**
  - (0.98 x 2) + (0.01 x 10) + (0.99 x 150) + (0.0001 x 30,000,000) =
    1.96 + 0.1 + 1.485 + 3000 = about 3000 cycles / access
- Need **LOW** page fault rates to sustain performance !!

More issues

- **Page selection policy**
  - When do we load a page ?
- **Page replacement policy**
  - What pages do we swap to disk to make room for new pages ?
  - When do we swap pages out to disk ?

Page selection policy

- **Demand paging**
  - Load page in response to access (page fault)
- **Pre-paging (prefetching)**
  - Predict what pages will be accesses in near future
  - Prefetch pages in advance of access
  - Problems
    - Hard to predict accurately
    - Mispredictions can cause useful pages to be replaced

Page replacement policies

- **Random**
- **FIFO** (first in, first out)
  - Throw out oldest pages
- **Optimal**
  - Throw out page used farthest in the future
- **LRU** (least recently used)
  - Throw out page not used in the longest time
- **NFU** (not frequently used)
  - Do not throw out recently used pages

Demand paging issues

- **Performance**
  - Need lots of locality -> otherwise run at disk speeds
    - If most accesses are to data already in DRAM -> great !
    - **Spatial locality**: often access “nearby” addresses
    - **Temporal locality:** Often re-access same addresses again and again
  - How to resume a process ?
    - Re-execute instruction? Only if no side effects !
  - Run other processes / threads while serving the page fault

Belay’s Anomaly

- Belay’s anomaly
  - For some replacement algorithms
  - **MORE** pages in main memory can lead to **MORE** page faults !!
- **Example:**
  - FIFO replacement policy
  - Reference string: A B C D A B E A B C D E
  - Three pages -> 9 faults
  - Four pages -> 10 faults
  - Adding more memory might not help for page faults in some replacement algorithms

Thrashing

- **Working set**
  - Collection of memory currently being used by a process
- **Thrashing**
  - If all working sets do not fit in memory
  - One “hot” page replaces another
  - Percentage of accesses that generate page faults skyrockets
- **Typical solutions**
  - “swap out” entire processes
  - Invoked when page fault rate exceeds some bound
  - Linux invokes the out-of-memory (OOM) killer

Inverted page tables

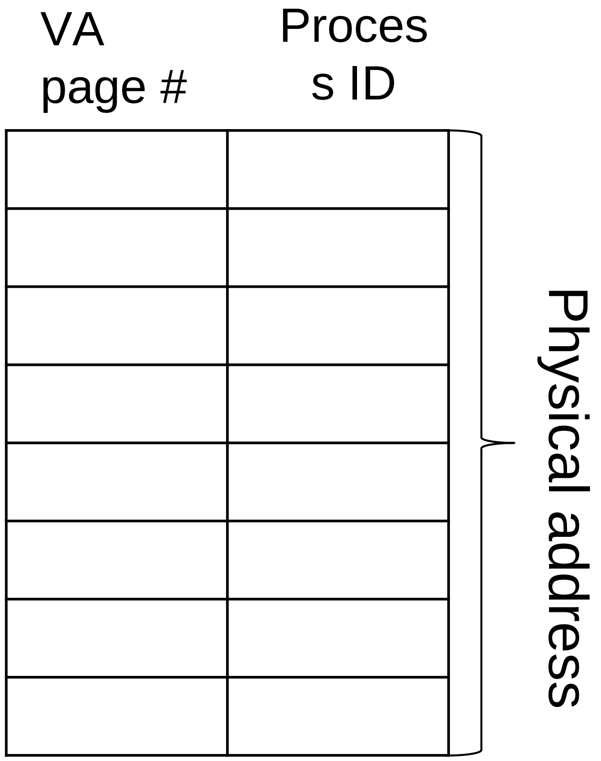
- **Multi-level page table**
  - The number of levels is increased as the size of virtual memory address space grows
  - Given 64-bits address space, 4-KB page size, a PTE of 4 bytes, each page table can store 1024 entries
  - 6 (ceil(52/10)) levels are required
  - 6 memory accesses for each address translation
- Observation of the inverted page table
  - Size of physical memory is much smaller
  - The inverted page table only has entries corresponding to actual physical frames

Inverted page tables

- **Inverted page table**
  - Aim to create a page table for all processes
  - The inverted page table stores physical frame
  - Each page table entry record which used pages for user processes
  - The page table include process ID and page number
  - Search problem in an inverted page table

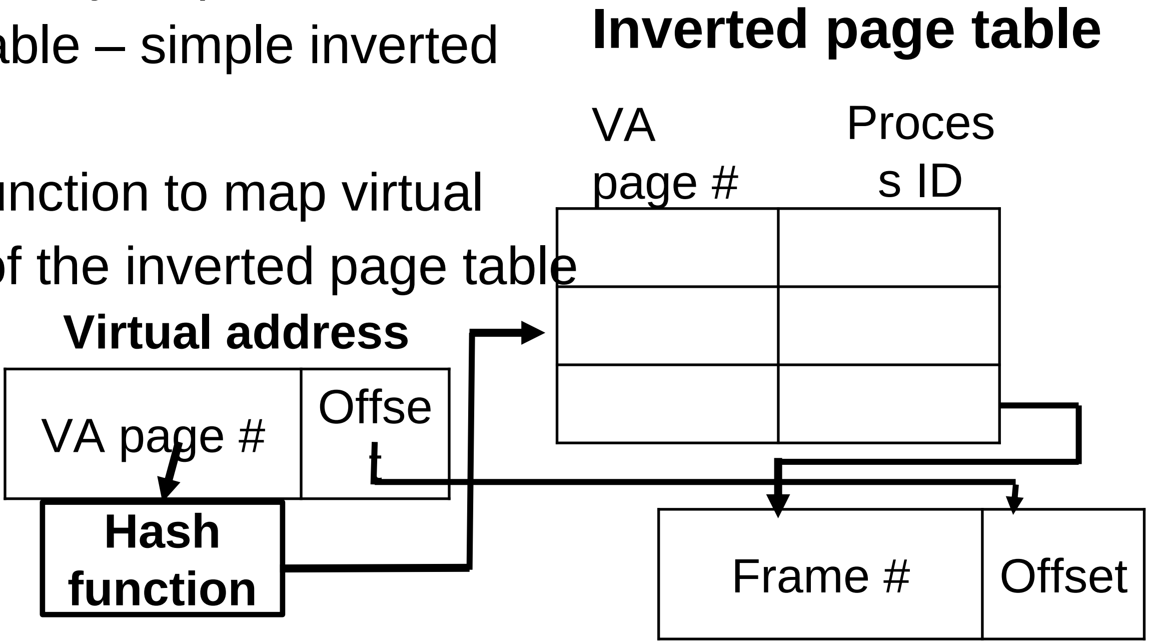
Inverted page table

- **Simple Inverted page table**
  - **The number of page table entries (PTE) = the number of physical frames**
  - Each PTE contain the pair <process ID, virtual page #>
  - Translate a virtual address, compare each <process ID, virtual page #> against each entry
  - If a match is found, the inverted page table index is used to obtain physical address



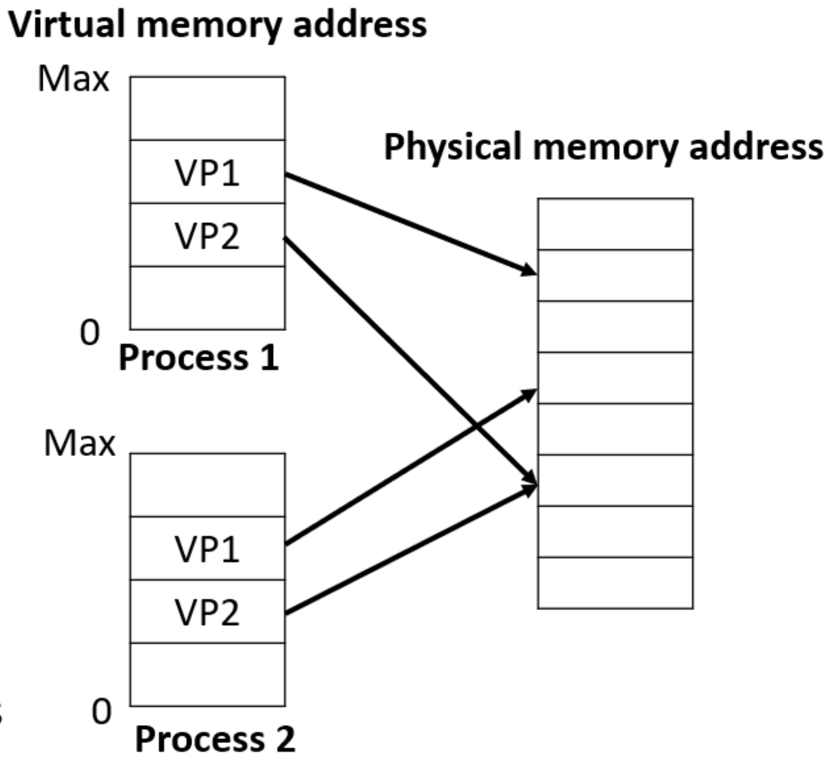
Inverted page table

- **Hashed inverted page table**
  - Finding a match may require to scan through entire table – simple inverted page table
  - Using hashed function to map virtual page # to PTE of the inverted page table



Sharing pages

- How to share memory?
    - Entries in different process page tables map to the same PPN
    - Each process can have its own registers and data
    - There can be only one copy of data kept in physical memory, such as shared library
    - Each process's page table maps onto the same PPN



Conclusion

- Multi-level page table
  - Reduce the page table
- Demand paging
  - Reduce I/O, load data from disk to memory when it is needed
- Effective access time (EAT)
  - Measure the performance of demand paging
- Page replacement
- Inverted page table
- Page sharing

Takeaway Questions

- What are purposes of the multi-level paging?
  - (A) Increase the number of pages can be used
  - (B) Reduce the memory space requirement to store page table
  - (C) Decrease the latency of the page table access
- How to reduce the number of unnecessary page access?
  - (A) Multi-level page table
  - (B) Page sharing
  - (C) Demand paging

Takeaway Questions

- How to reduce effective access time (EAT) ?
  - (A) Reduce the number of page faults
  - (B) Reduce page size
  - (C) Using the multi-level page table
- What are problems of an inverted page table?
  - (A) Increase the virtual page size
  - (B) The physical frame size is large
  - (C) Costly PTE search