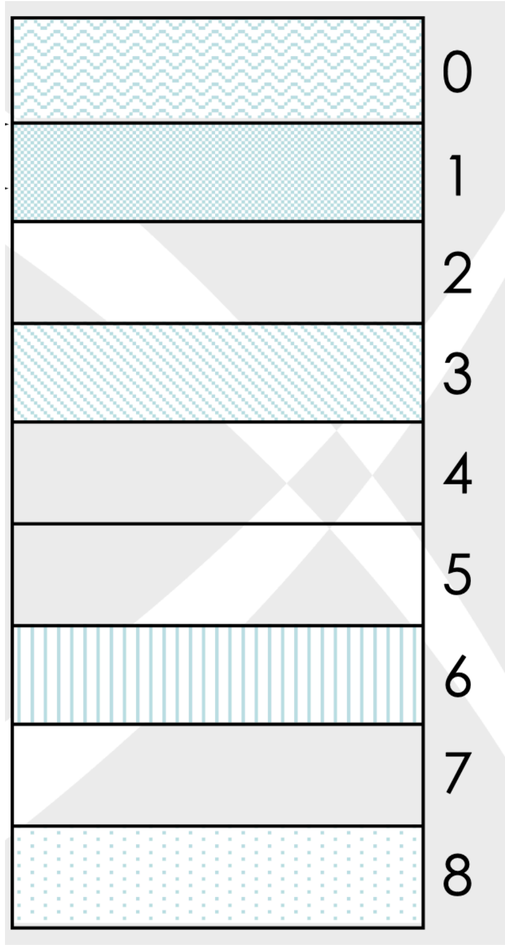
Memory Management (2)

Paging has internal fragmentation

Paging

Paging avoids external fragmentation

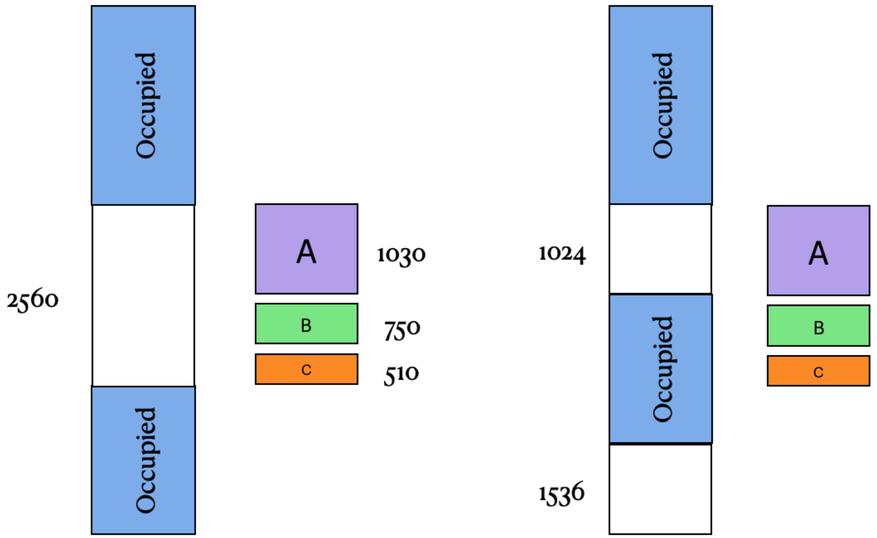
- With contiguous allocations, we allocated memory is variable sized blocks - big enough to fit the requests
- Paging: always allocate memory in fixed sized blocks — called pages
- Request is rounded up to the nearest page multiples
- Page size is 1024 bytes
- Request of any size less than 1024 bytes is a single page! Request for 10 bytes is given a page (1024 bytes)
- A request of 1500 bytes would get 2 pages - 2048 bytes
- Free memory is also multiples of pages



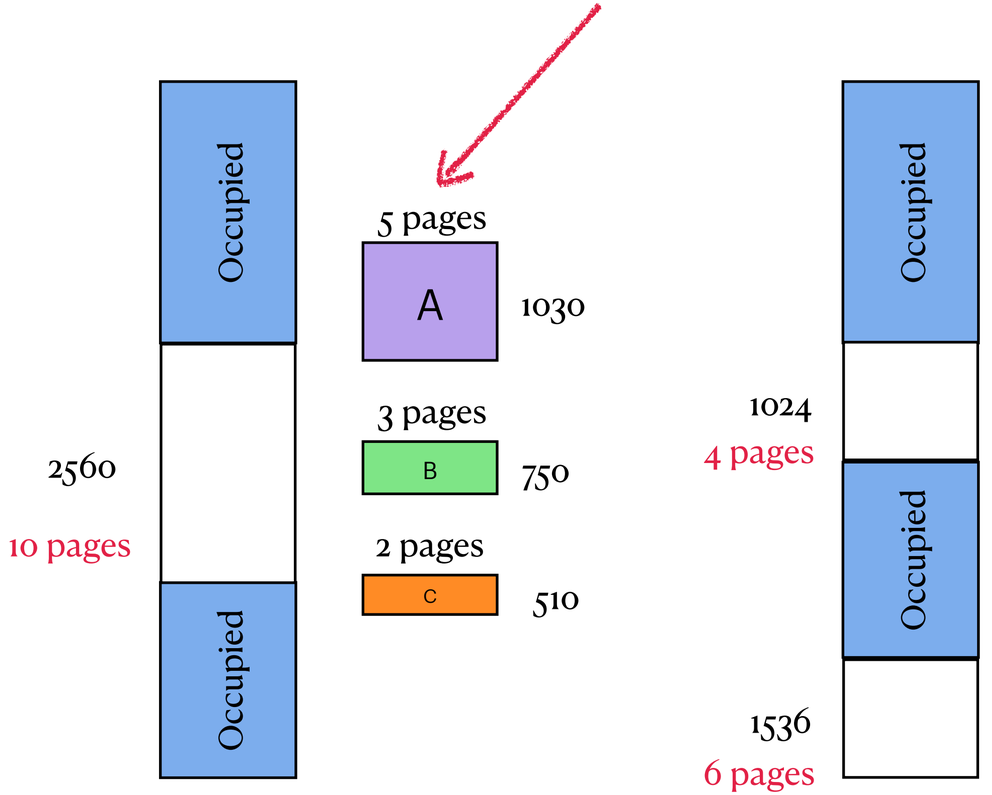
Memory split into pages

Memory Allocation Example Again

- Lets say we have paging - page size is 256 bytes
- Lets consider the old example where we had three processes: A, B, and C



What is the internal fragmentation?

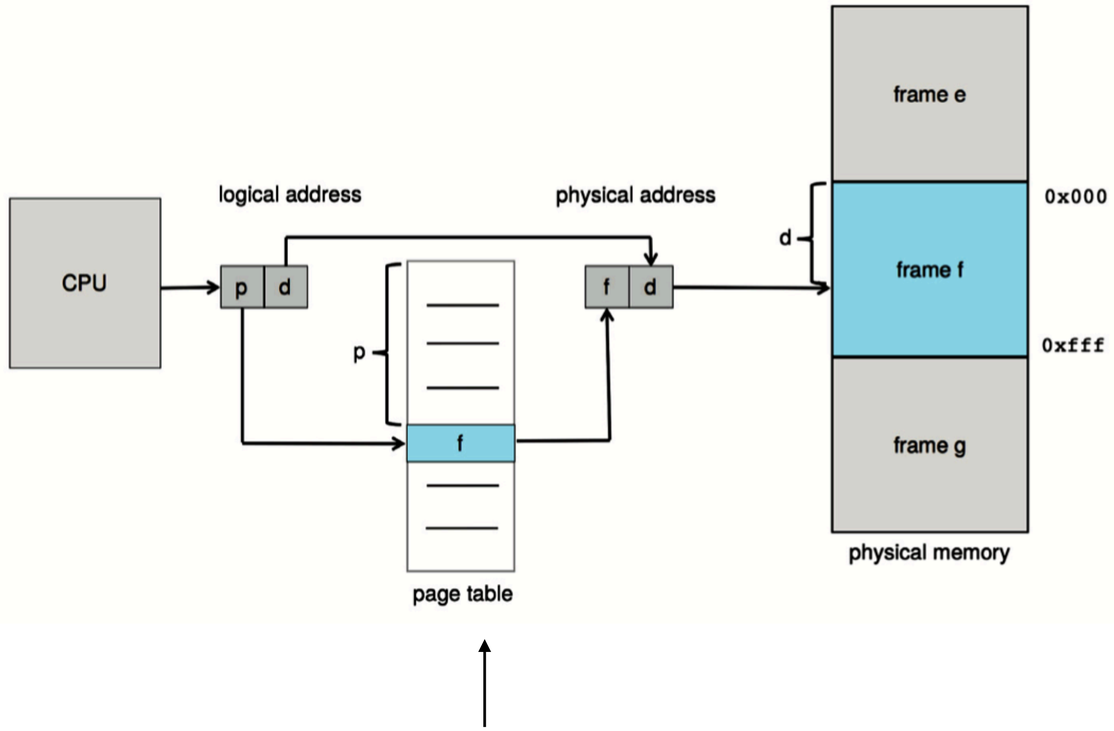


All we need is to allocate 10 pages - no need to be contiguous! All processes can be allocated unlike contiguous allocation.

Paging as a Mapping

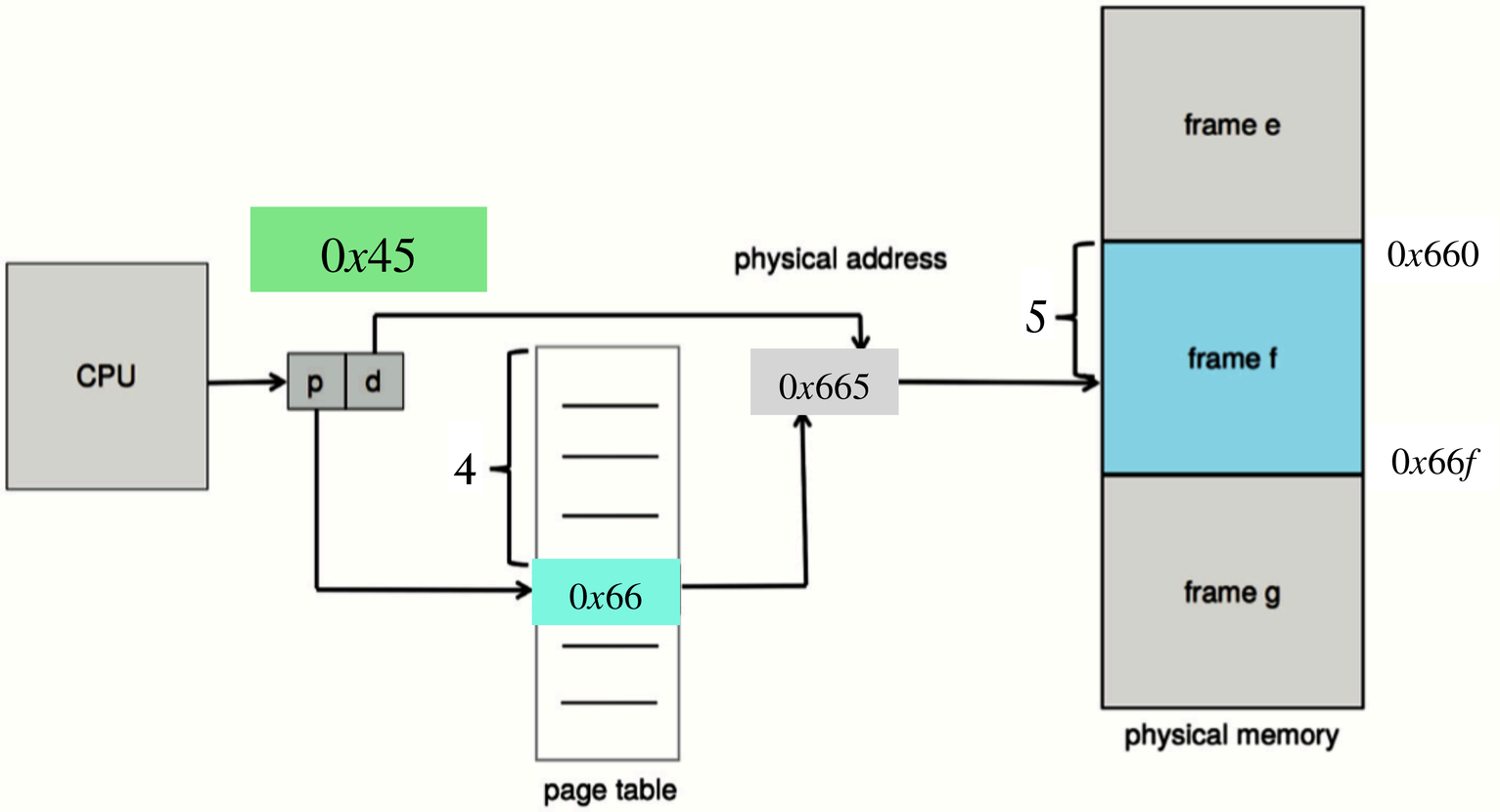
Page size defined by hardware, some hardware provide multiple page sizes

- Paging is mapping logical addresses into physical addresses
- Paging mechanism needs to give the physical address corresponding to a logical address
- Break the physical memory to fixed sized chunks called frames
- Break the logical memory into same fixed sized chunks called pages
- Map pages to frames



Per-process page table

Paging Through An Example

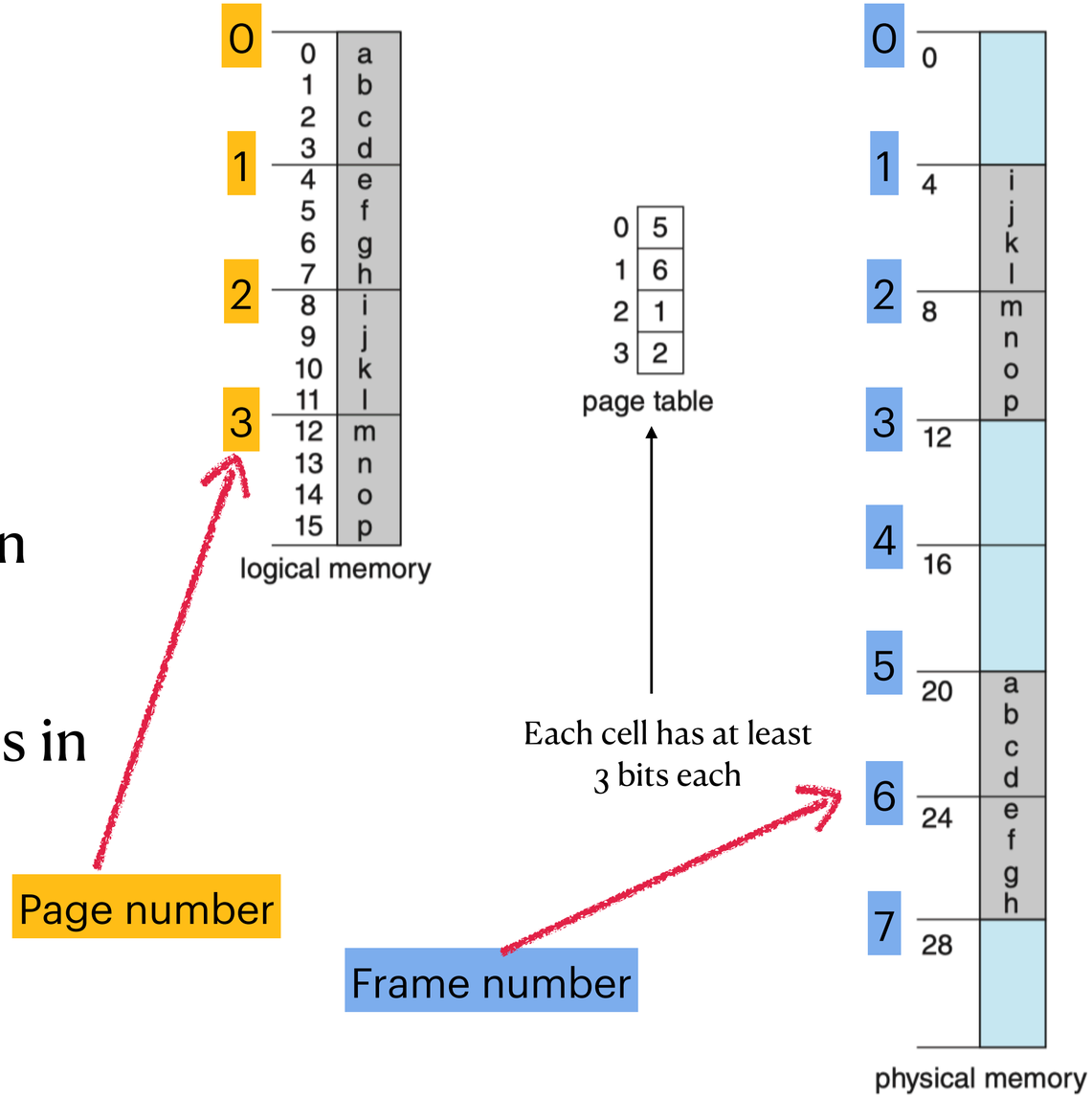


Frame size is 16 (determined by the number of bits in the page offset field)

Per-process page table

More Paging Examples

- Logical address has 4 bits ⇒ 16 addresses in logical address space
- Page offset is 2 bits ⇒ a page has 4 cells
- Page number has 2 bits ⇒ there are 4 pages in the logical address space
- Frame number has 3 bits ⇒ there are 8 frames in the physical memory



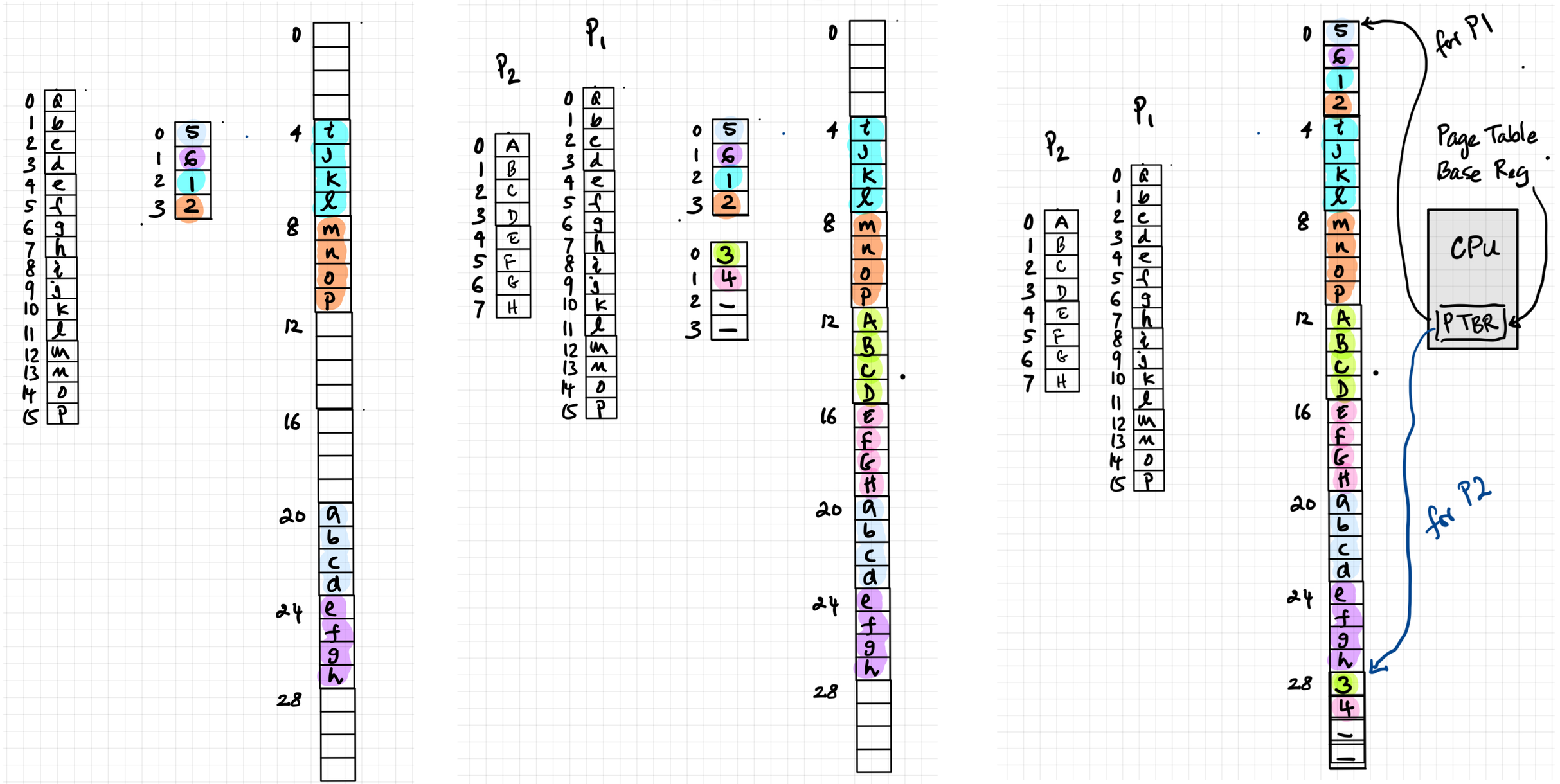
More Paging

- A 32-bit CPU can have 4 byte long page table entries, but they can vary.
- A 32-bit entry in the PTE can point to 2^{32} page frames. If a frame is 4KB (2^{12}), we have support for 2^{44} bytes (16 TB) of physical memory
- A page table entry will typically have other information besides frame number - so actual memory for the above config will be lower

Questions?

- Logical ⇒physical translation is done through page tables
- Where do we hold the page tables? In the memory of course!
- Problem: page table as we saw are contiguous - otherwise offset addressing would not work
- Memory can be allocated in page sized chunks (a page is contiguous) ⇒ Page table itself must be split into pages!
- This gives us hierarchical page tables

Simple Paging Example



Another Paging Example

Process has 20 logical addresses
Need a page table with at least 5 entries given pages of 4 bytes each

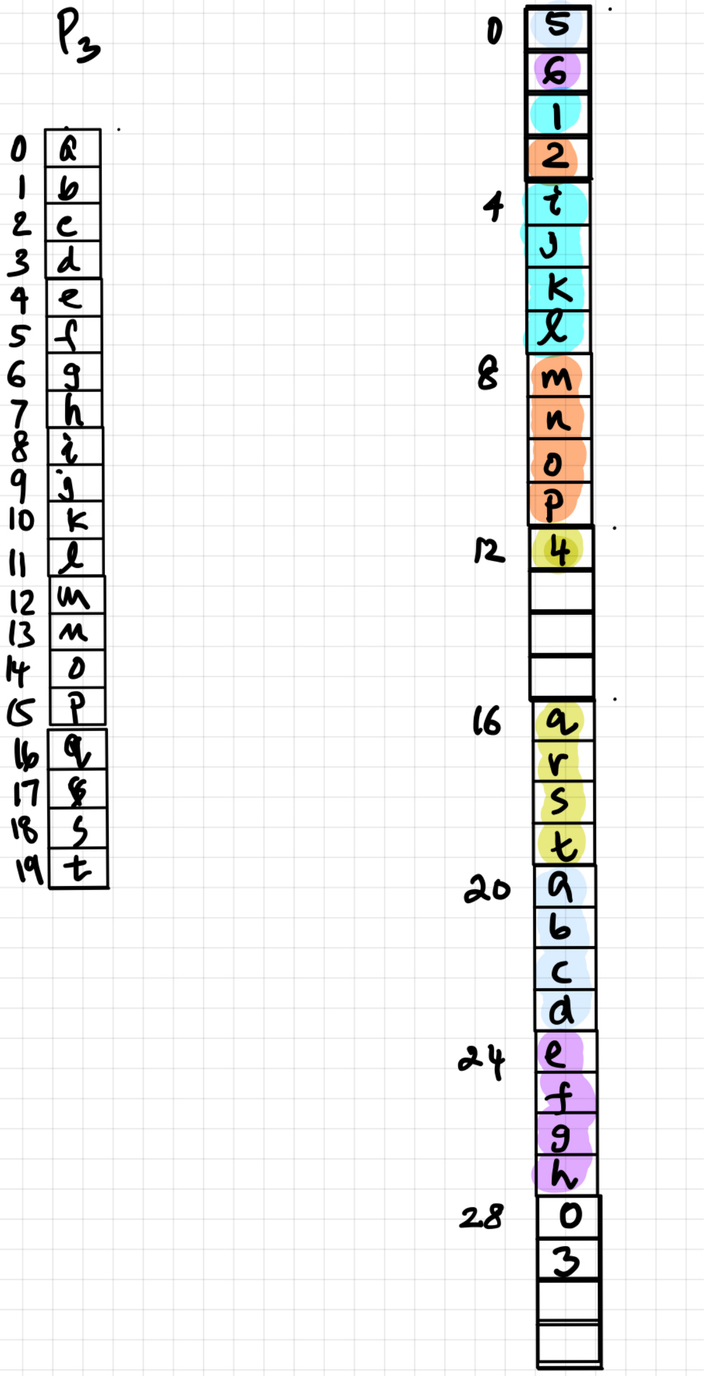
A page table with 5 entries need 2 pages.
We need a page directory (outer page table)

20 address in the logical address space

We have 5 bits in the logical address

Lets consider LA = 15 ⇒ 0 1 1 1 1 (in binary)

Top bit (0) used to access the outer page table (page directory)
Next two bits (11) used to access the inner page table
Last two bits (11) used to access the data within the frame (page in physical memory)



Some Observations About Paging

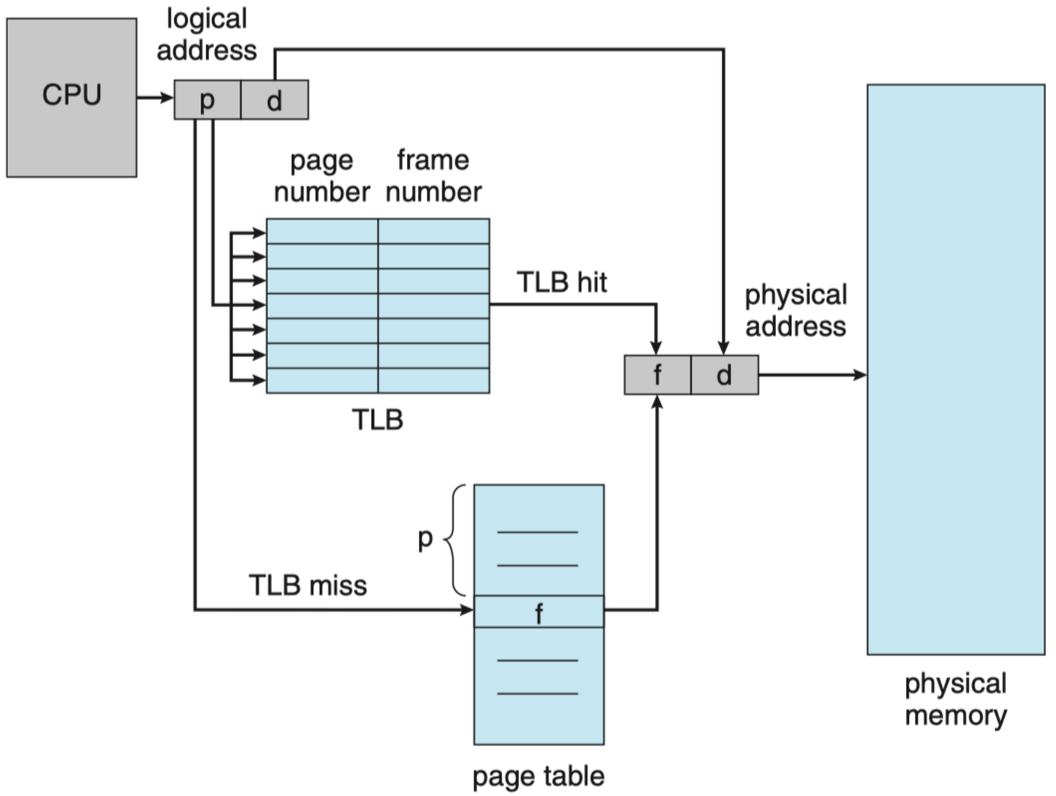
- Page tables are stored in the physical memory along with the data
- Page tables grow as the logical address space becomes large
- Modern CPUs have large logical addresses ⇒we need large amount of memory for page tables
- We can reduce the page table memory usage by having large page sizes ⇒ waste memory due to internal fragmentation
- Each process has its own page tables which are activated when we context switch to that process

Paging Design

- We have a CPU with LA of 32 bits ⇒ total addressable logical address space is 2^{32}
- Lets say we use 8 KB pages ⇒ need 13 bits in the page offset 2^{13}
- Lets say the page table entry (PTE) is 4 bytes ⇒ a page can hold 2048 entries ⇒ page number can have up to 11 bits
- How many levels of page tables do we have?

Translation Look-aside Buffer

- Paging is doing a translation from page number to frame number
- TLB is a way to cache known translations (previously done) and safe on accessing the page tables and actually doing the translation
- TLB performance benefit is determined by the TLB hit rate - fraction of the time we can find the mapping in the TLB cache
- TLB hit rate can be quite high

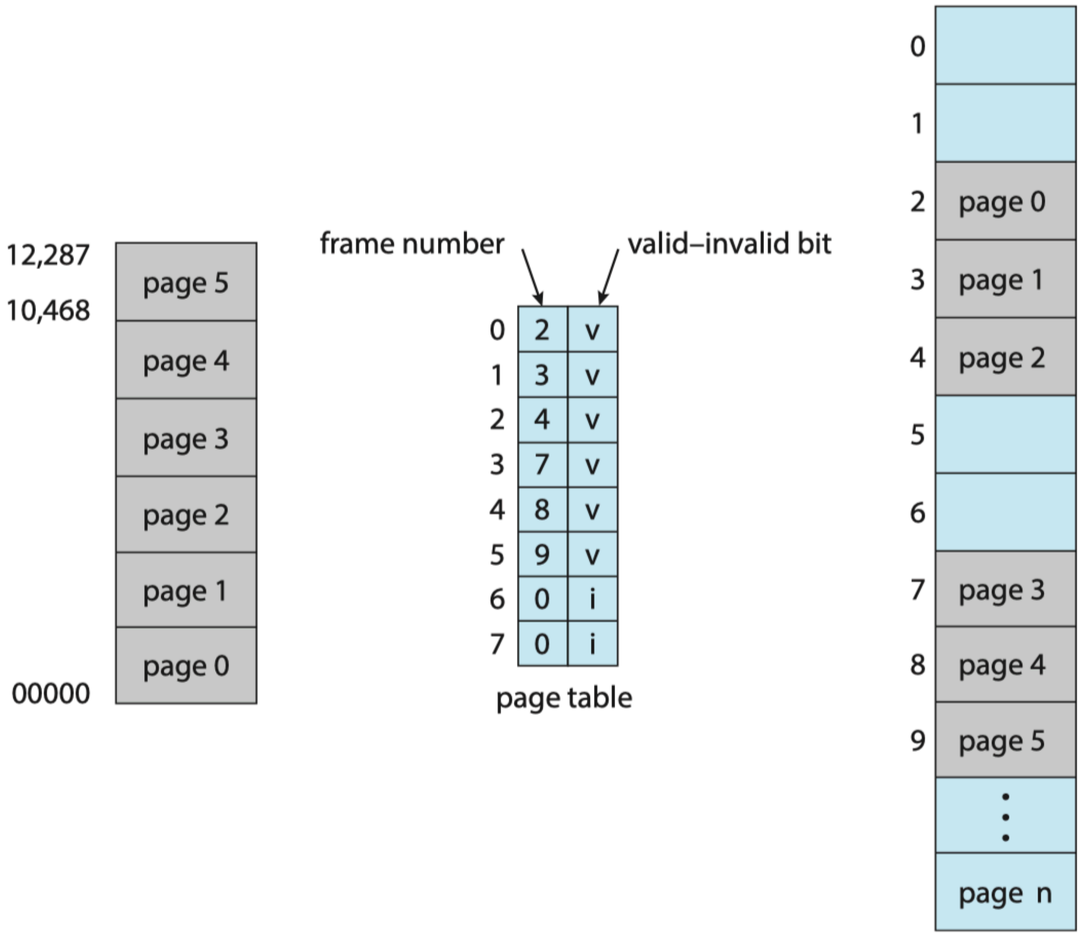


TLB Performance

- Lets say it take 10 ns to access memory
- Lets say we have a 3-level page table ⇒ we need 3 accesses to find the page number to frame number mapping
- Total time = 40 ns for a single memory access ⇒ 4 times slower
- TLB can be used to fix this problem, lets say TLB hit rate is 0.9
- Effective access time = $0.9 * 10 + 0.1(3 * 10 + 10) = 13$ ns!
- TLB hit rates are typically quite high ⇒ slow down due to paging is substantially reduced using TLB
- Intel Core i7 has a 128-entry L1 instruction TLB and 64-entry L1 data TLB

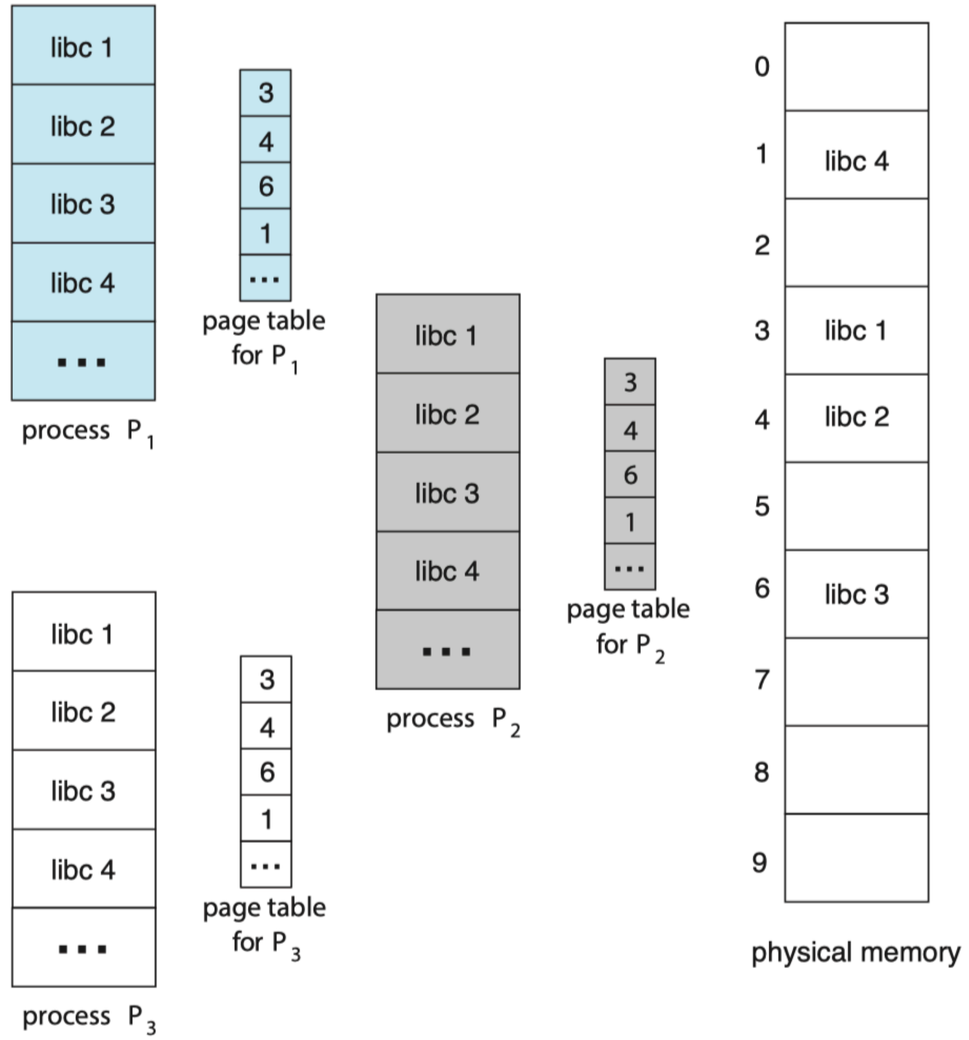
Protection in Paged Systems

- We have logical address system with 14 bit address ⇒ addresses from 0 to 16383
- Our process wants to access from 0 to 10468 (5 pages and another half a page)
- So the process gets access beyond 10468, up until 12287
- This is due to internal fragmentation - no other process is in that memory so excess access is not going to cause harm



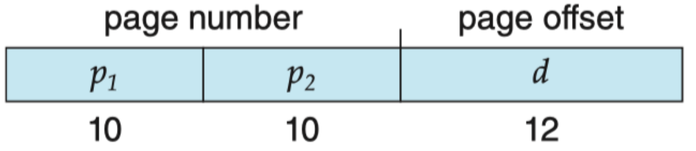
Sharing Pages

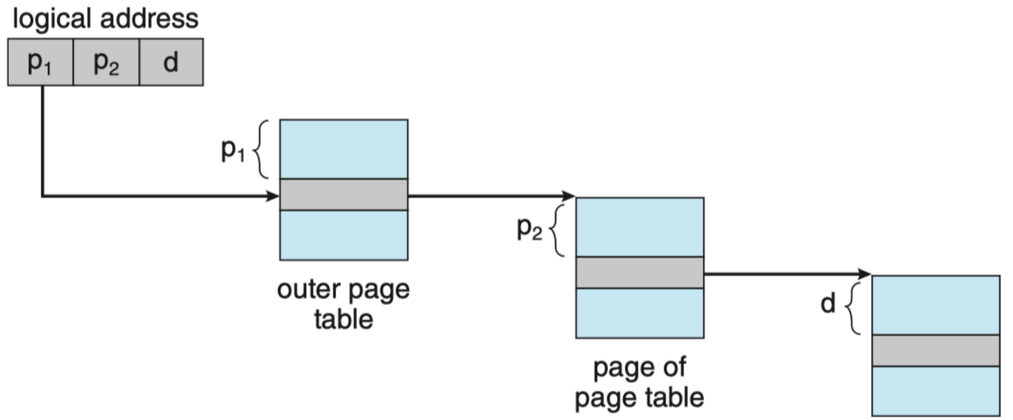
- Advantage of paging is sharing common code ⇒ libc can be loaded into different processes without being duplicated in memory
- Reentrant (non self modifying) code can be shared using page
- Map the same physical pages to the different processes

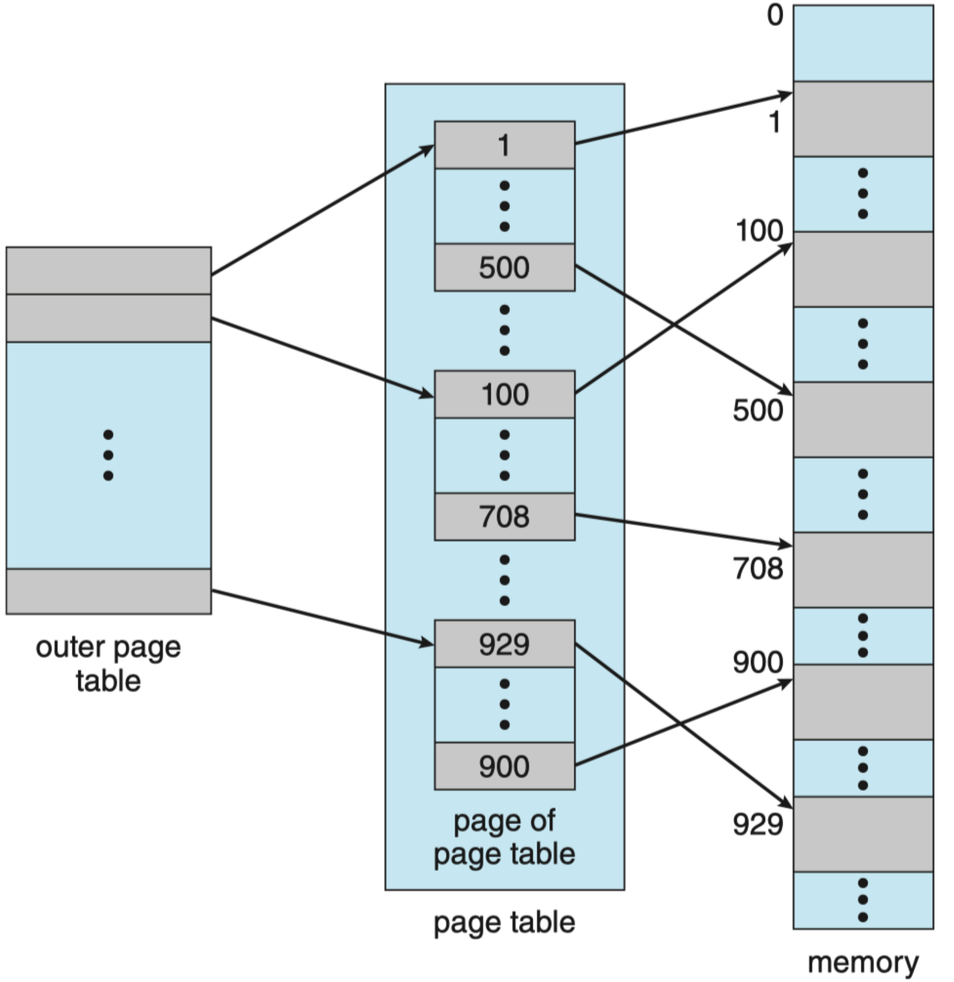


Hierarchical Paging

- We already saw this earlier - lets recap!
- 32-bit logical addresses, 4KB page size
- logical address is divided like shown below

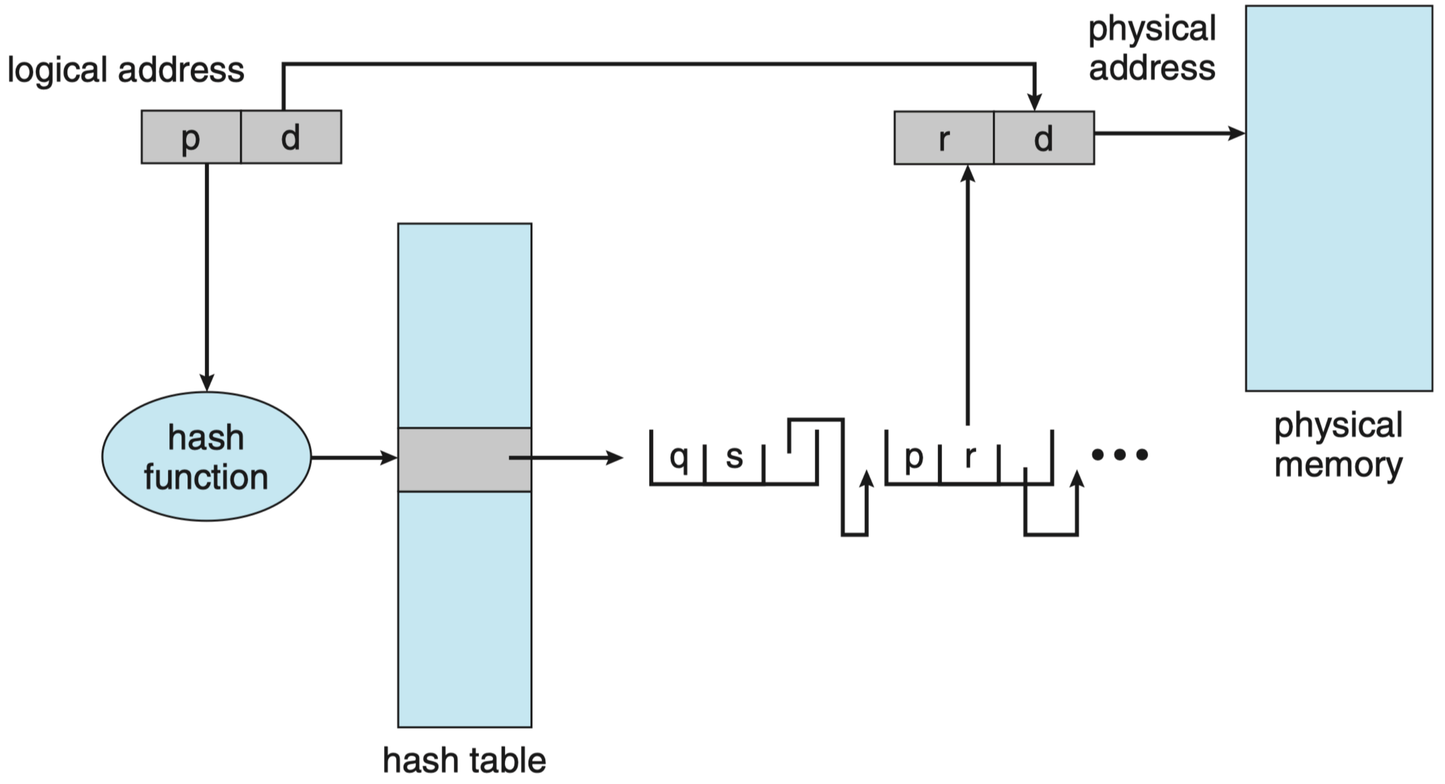






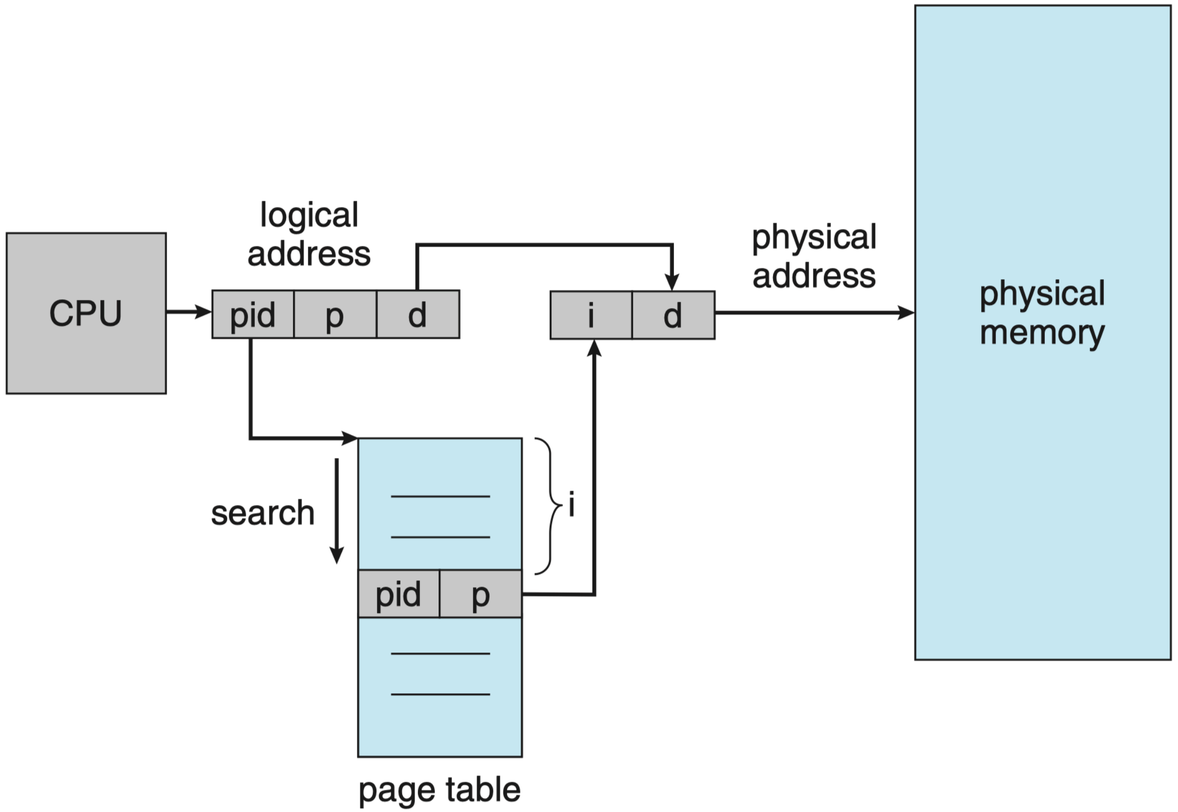
Hashed Page Tables

- Virtual page number (p) is hashed into the hash table
- Hash table points to a linked list of mappings
- If p ⇒ r mapping is found, r is used as the frame number
- Physical address is (r, d) for the logical address (p, d), where d is the page offset



Inverted Page Tables

- In forward page tables, each process has its own page table
- Page table has an entry for each page the process is using
- Page tables can have millions of entries and consume large amount of memory
- Inverted page tables maintain an entry for each real page frame in memory
- Searching the inverted page tables is hard - time consuming



Segmentation and Paging

- Intel 32- and 64-bit architectures have segmentation
- You can have segmentation working with paging