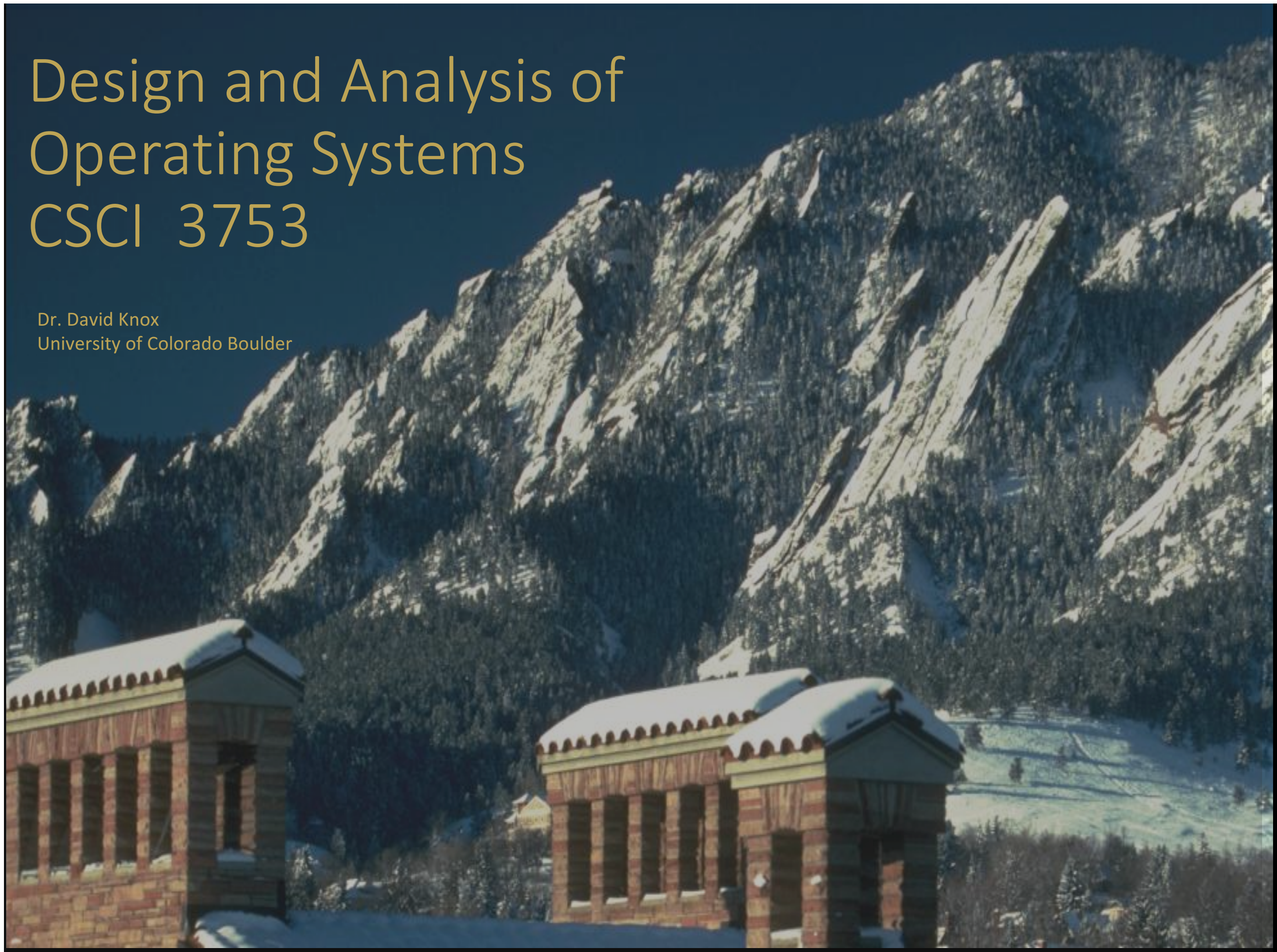
Department of Computer Science
UNIVERSITY OF COLORADO BOULDER

Design and Analysis of Operating Systems CSCI 3753

Dr. David Knox
University of Colorado Boulder

These slides adapted from materials provided by the textbook authors.



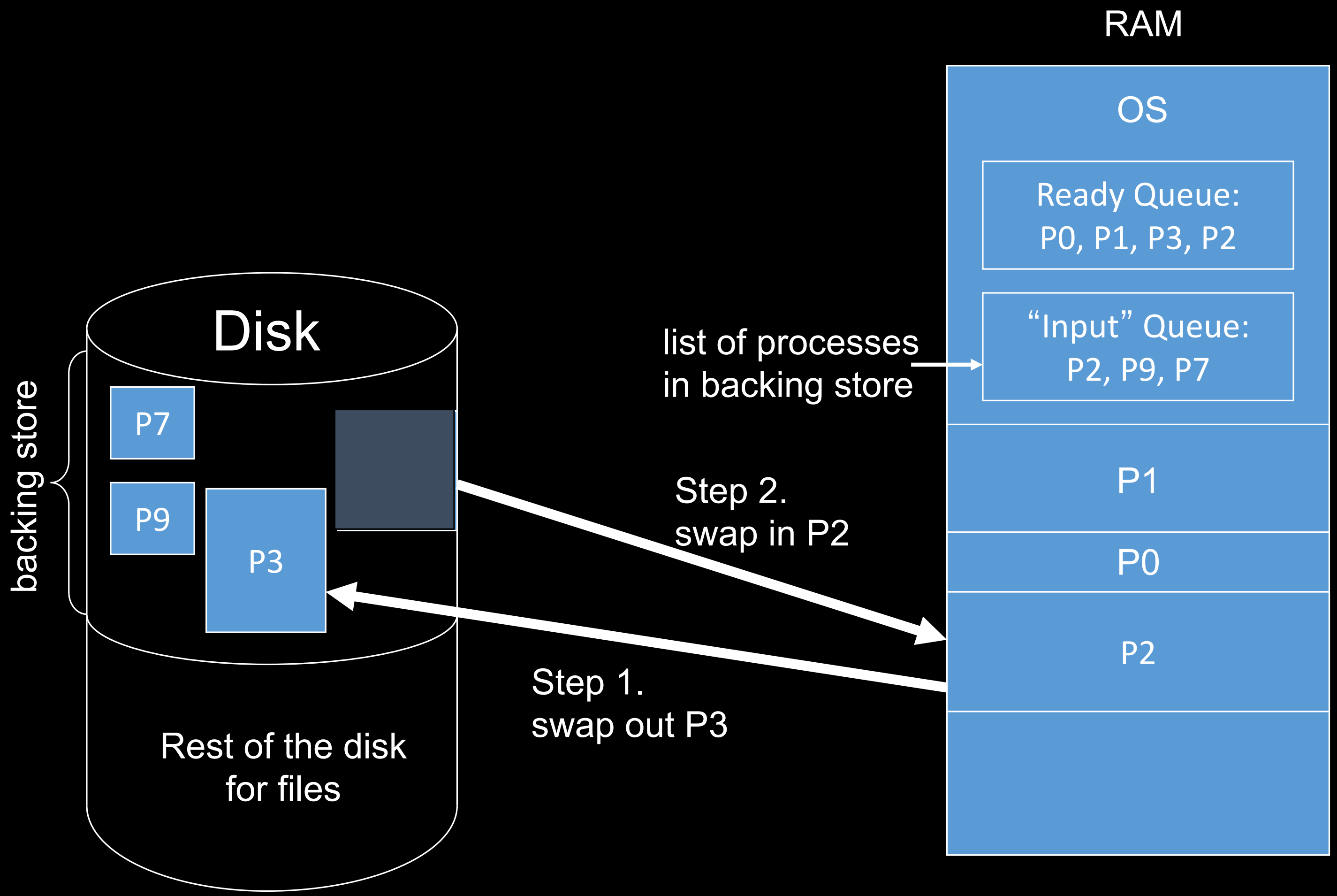
Design and Analysis of Operating Systems CSCI 3753

Dr. David Knox
University of Colorado Boulder

Memory Management
Swapping, Allocation, Fragmentation, Segmentation

Swapping

- When OS scheduler selects process P2, dispatcher checks if P2 is in memory.
- If not, there is not enough free memory, then swap out some process P_k, to make room
- If run time binding is used, then a process can be easily swapped back into a different area of memory.
- If compile time or load time binding is used, then process swapping will become very complicated and slow - basically undesirable



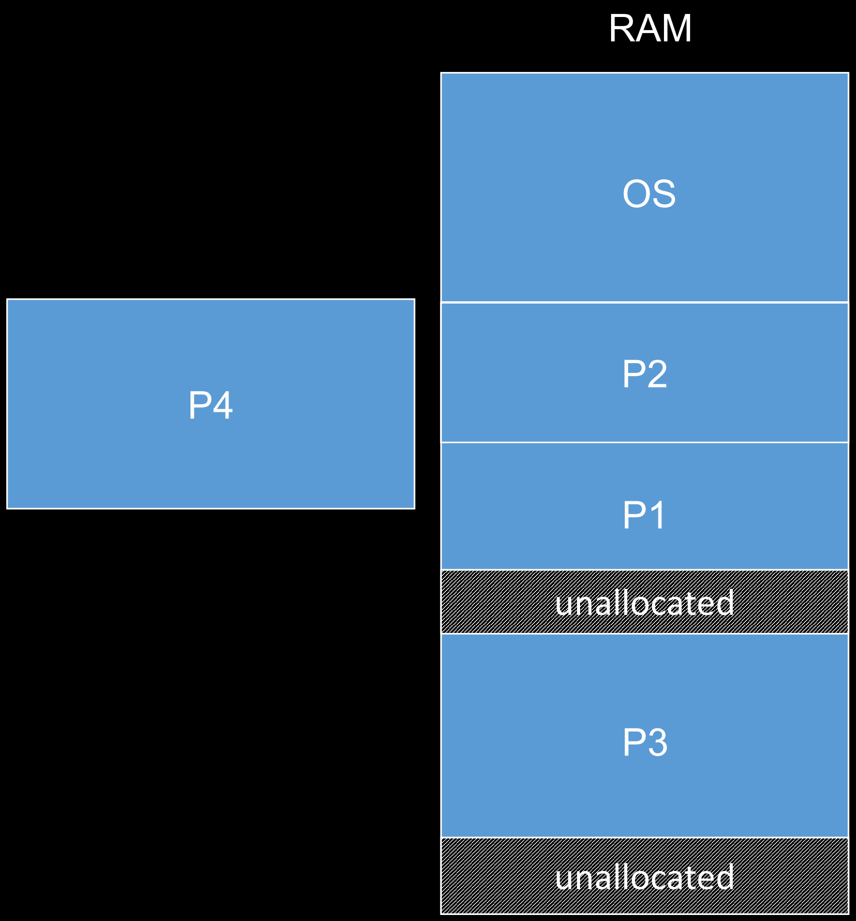
Swapping

- Context-switch time of swapping is very slow
  - Disks take on the order of 10s-100s of ms
  - When adding the size of the process to transfer, then transfer time can take seconds
  - Ideally hide this latency by having other processes to run while swap is taking place behind the scenes,
    - e.g. in RR, swap out the just-run process, and have enough processes in round robin to run before swap-in completes & newly swapped-in process is ready to run
  - can't always hide this latency if in-memory processes are blocked on I/O
  - UNIX avoids swapping unless the memory usage exceeds a threshold
- Swapping of processes that are blocked or waiting on I/O becomes complicated
  - one rule is to simply avoid swapping processes with pending I/O
- Fragmentation of main memory becomes a big issue
  - can also get fragmentation of backing store disk
- Modern OS's swap portions of processes in conjunction with virtual memory and demand paging

Memory Allocation

Allocation over time:

- as processes arrive, they're allocated a space in main memory
- processes leave and memory is deallocated
- results in *fragmentation* of main memory
  - There are many small chunks of non-contiguous unallocated memory between allocated processes in memory
- for next process,
  - OS must find a large enough unallocated chunk in fragmented memory
  - May have enough memory, but not contiguous to allow a process to load

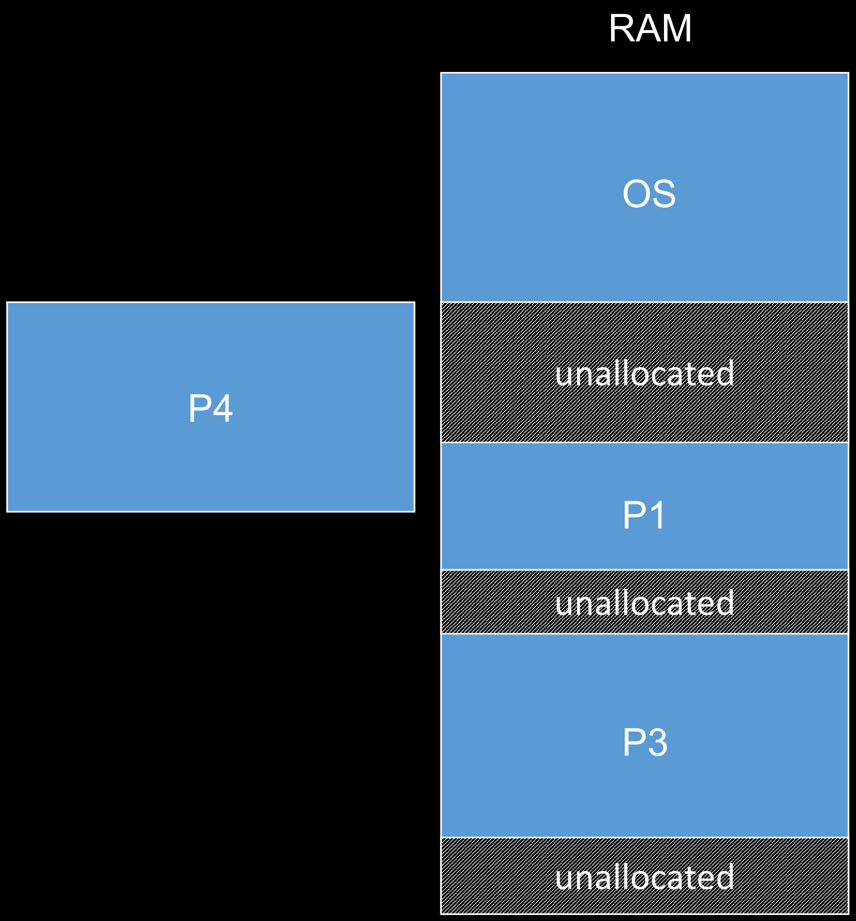


Fragmentation

- This results in ***external fragmentation*** of main memory
    - There are many small chunks of non-contiguous unallocated memory between allocated processes in memory

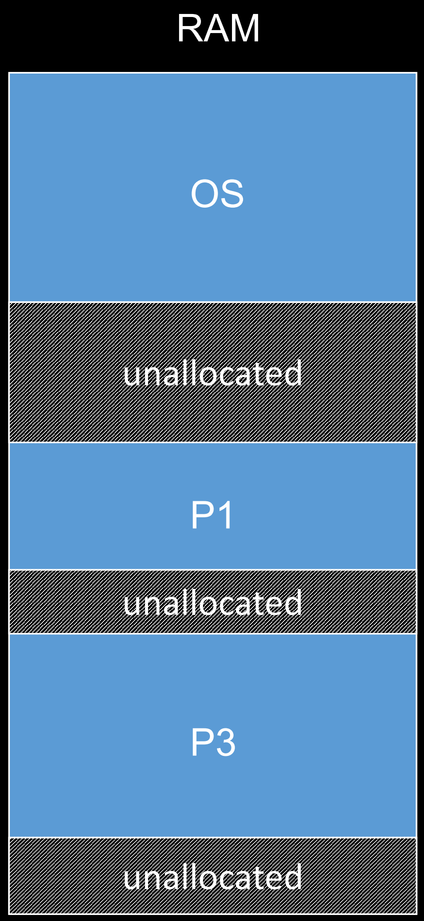
- OS must find a large enough unallocated chunk in fragmented memory that a process will fit into

- De-fragmentation
    - Algorithms for recombining contiguous segments is used, but memory still gets fragmented
    - Moving memory is expensive

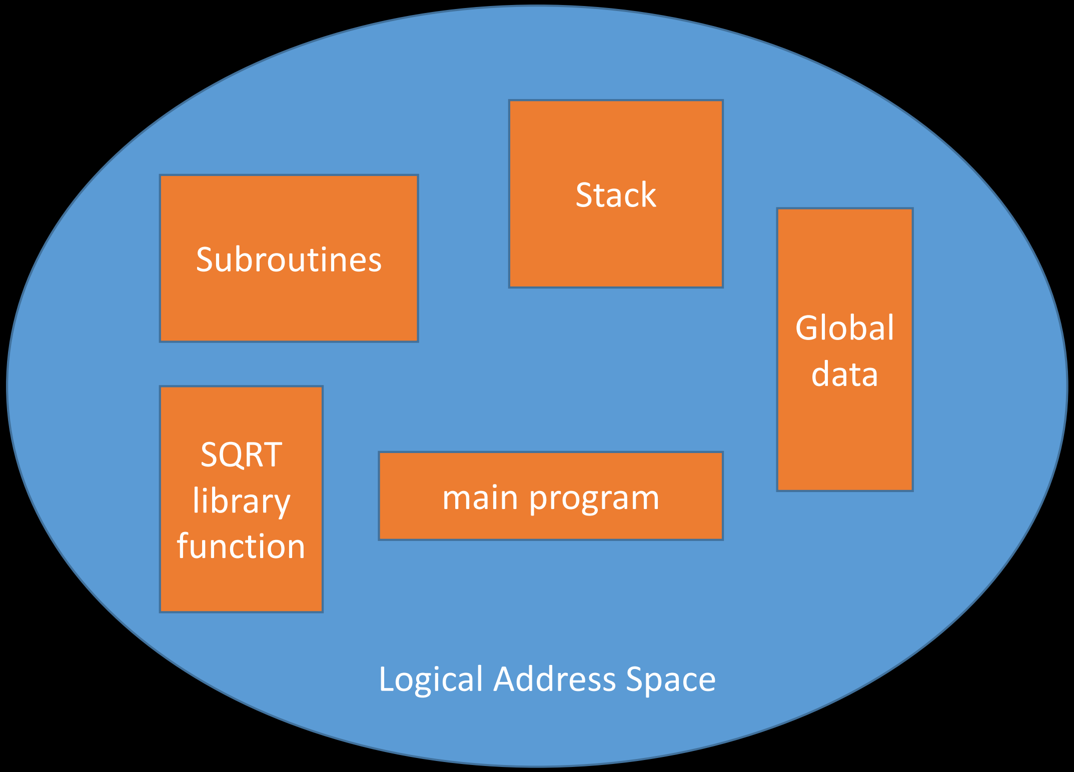


Memory Allocation

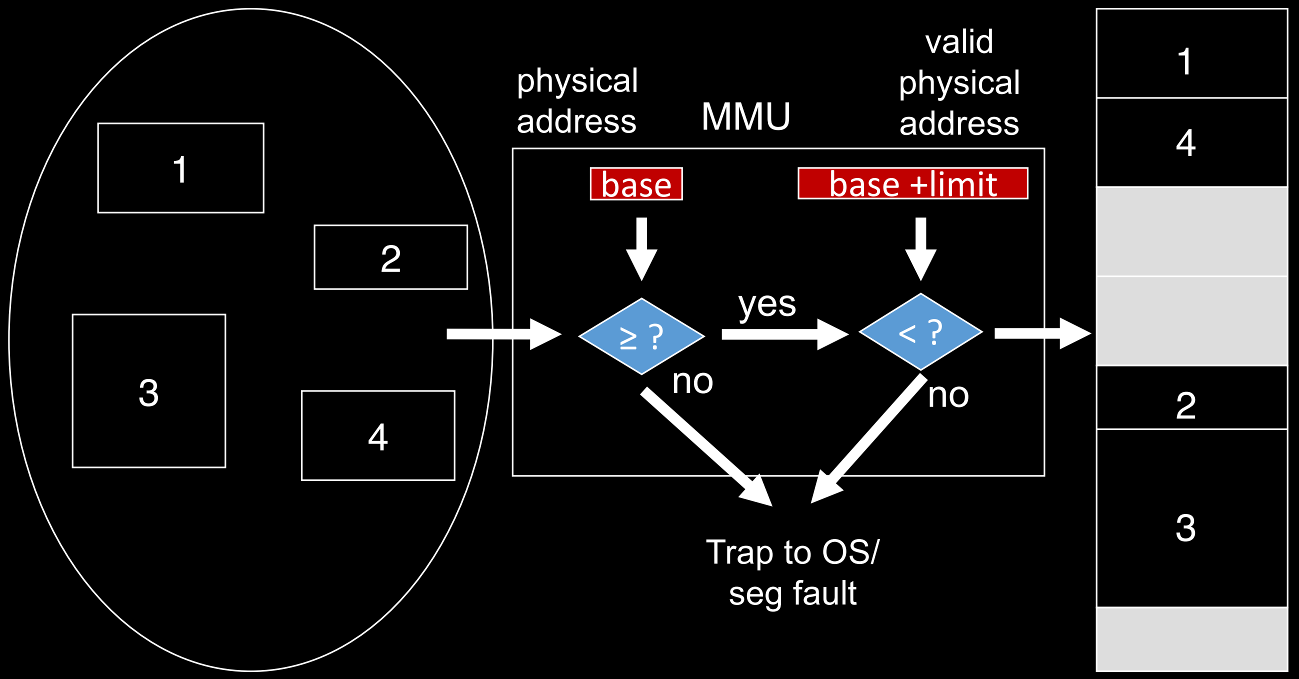
- Multiple strategies:
  - *best fit* - find the smallest chunk that is big enough
    - This results in more and more fragmentation
  - *worst fit* - find the largest chunk that is big enough
    - this leaves the largest contiguous unallocated chunk for the next process
  - *first fit* - find the 1st chunk that is big enough
    - This tends to fragment memory near the beginning of the list
  - *next fit* – view fragments as forming a circular buffer
    - find the 1st chunk that is big enough after the most recently chosen fragment



Instead of one big address space break it up into smaller segments



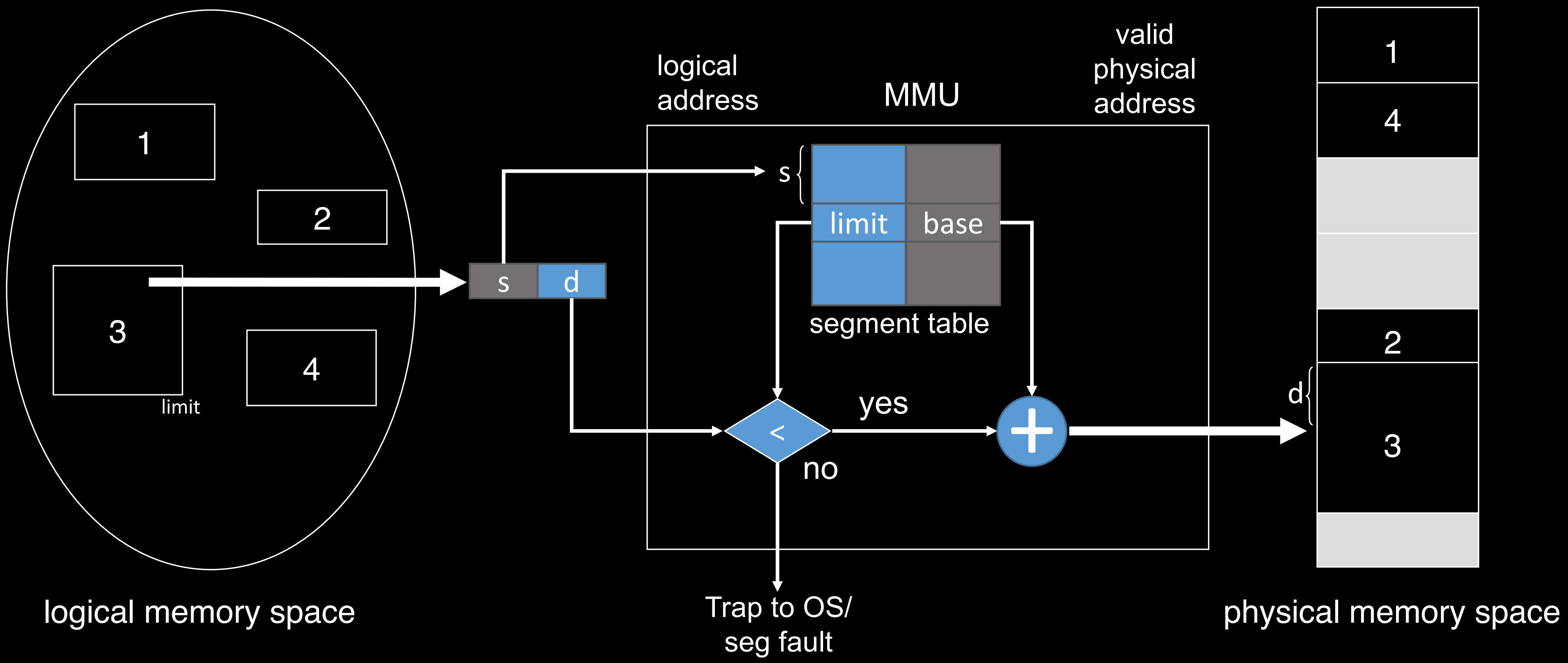
Segmentation of Memory



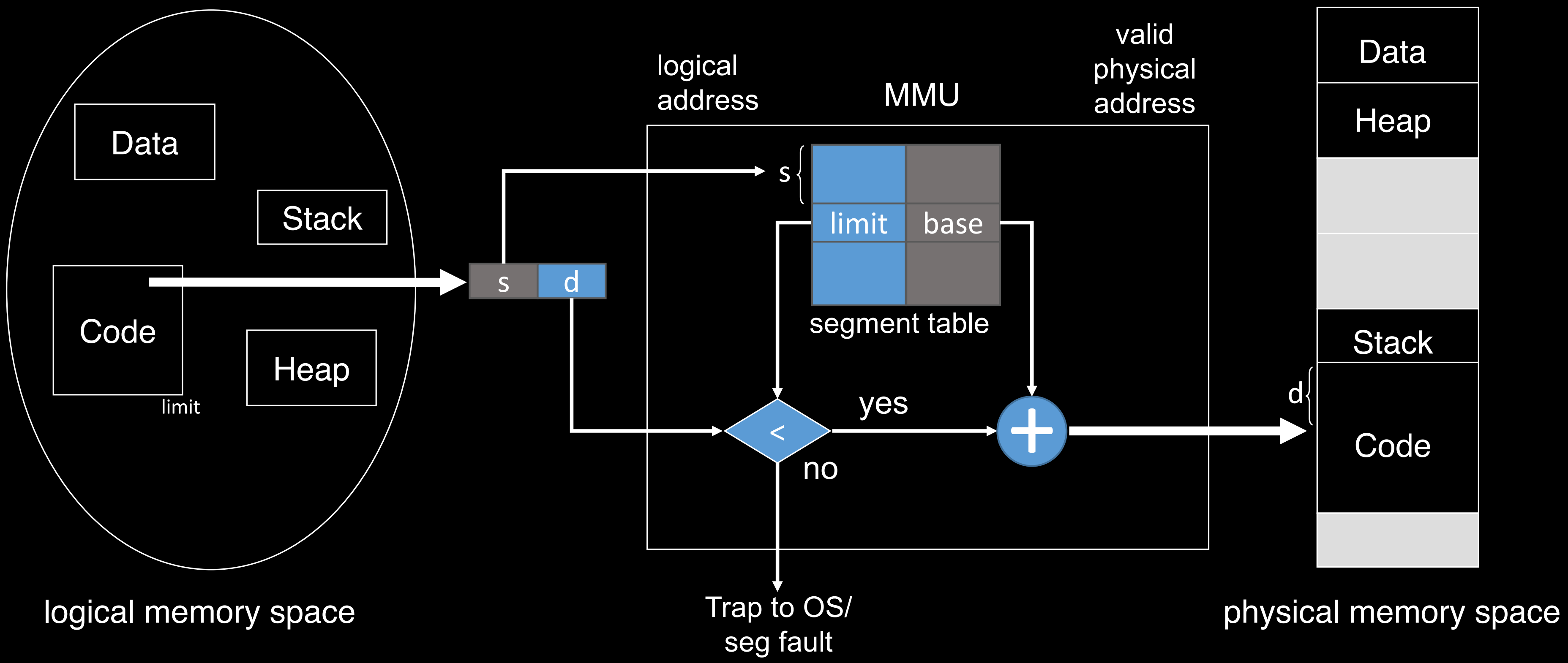
logical memory space

physical memory space

Segmentation of Memory



Segmentation of Memory



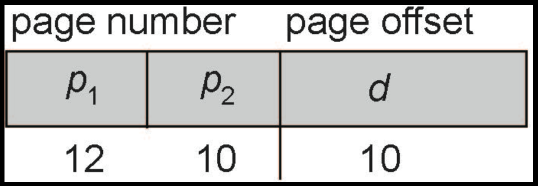
Better Solution to Reduce Fragmentation

Hierarchical Paging

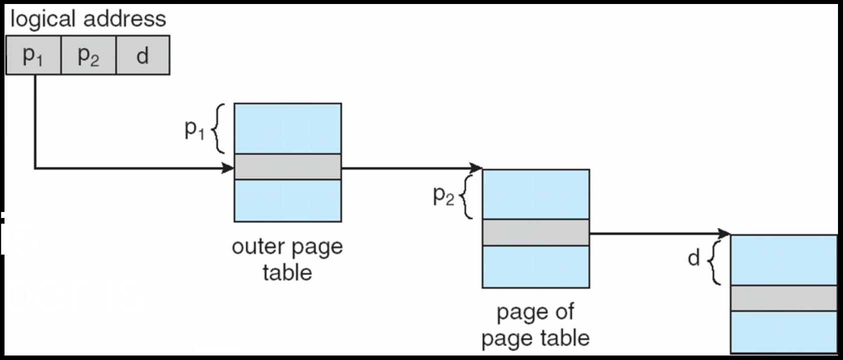
- Problem with page tables: they can get very large
  - example: 32-bit address space with 4 KB/page (2^{12}) implies that there are $2^{32}/2^{12}$ = 1 million entries in page table per process. if each entry is 4 bytes that is 4 MB.
  - it's hard to find contiguous allocation of at least 4 MB for each page table
- Solution: page the page table! this is an example of hierarchical paging.
  - Just as processes are split into pages to fit into a memory, split page table into pages to fit into memory
  - This is an example of 2-level paging. In general, we can apply N-level paging.

Hierarchical Paging

- A logical address (on 32-bit machine with 1K page size) is divided into:
  - a page number consisting of 22 bits
  - a page offset consisting of 10 bits



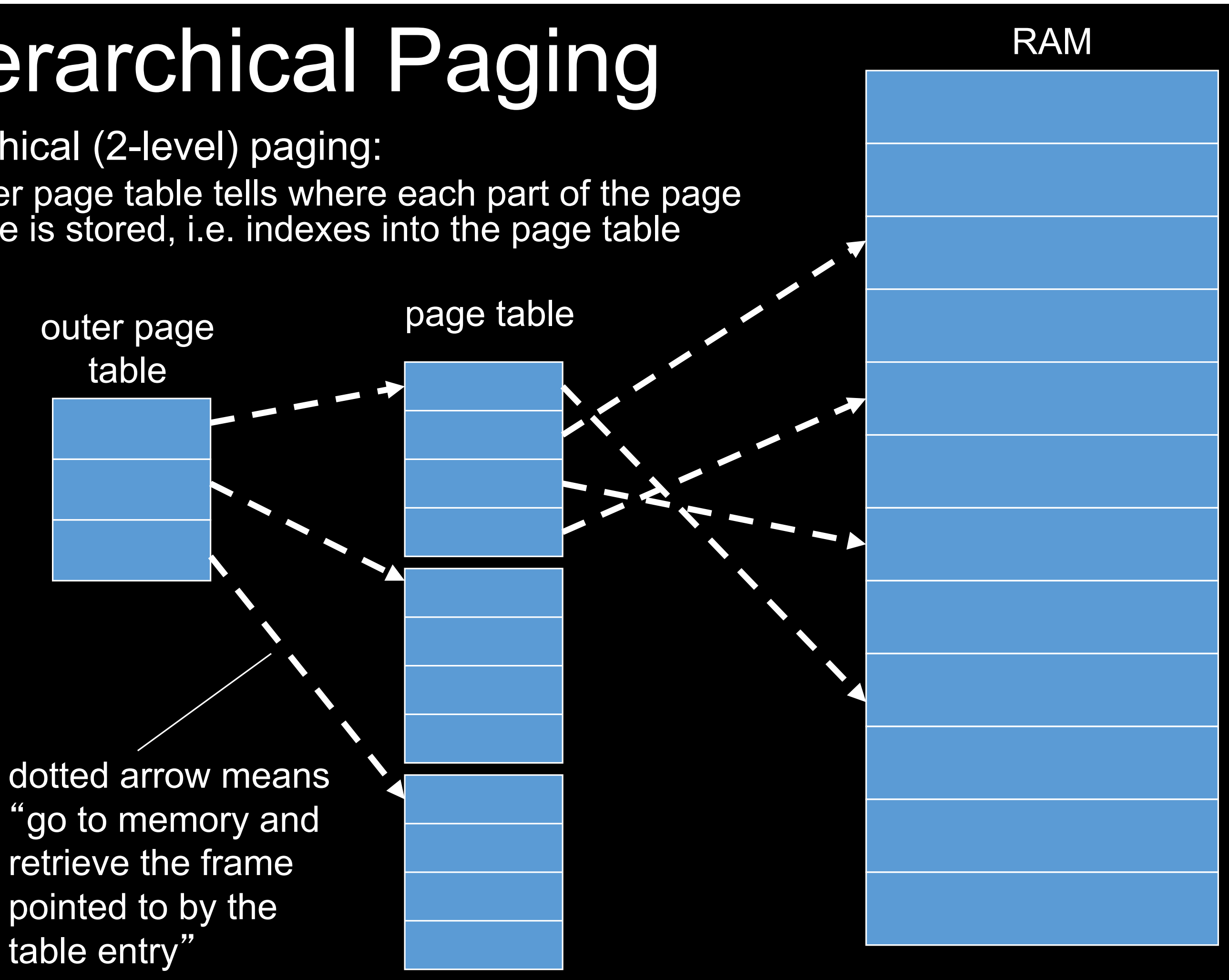
page number		page offset
p_1	p_2	d
12	10	10



- Since the page table i
  paged, the page num
  further divided into:
  - a 12-bit page table number (outer page)
  - a 10-bit page table offset

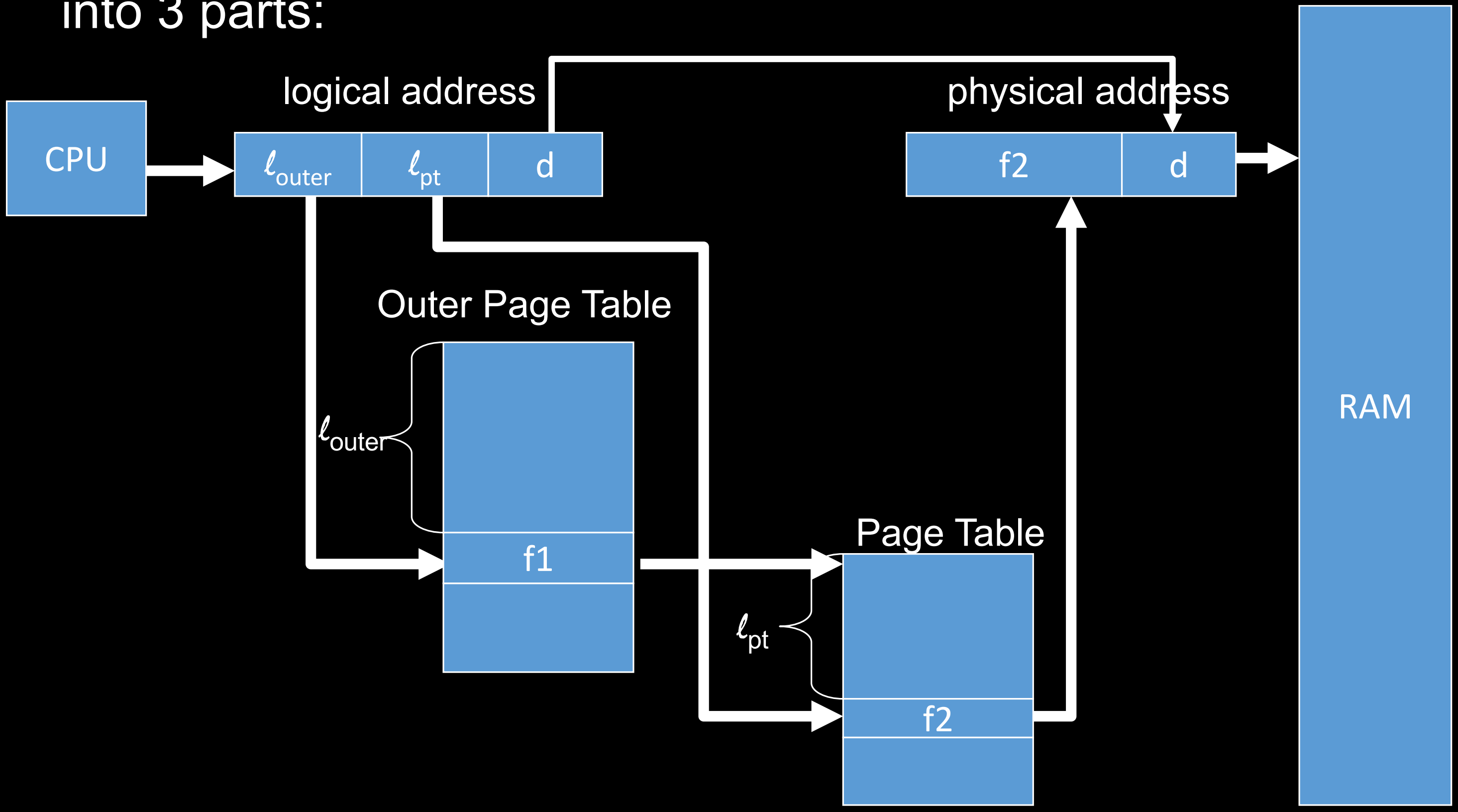
Hierarchical Paging

- Hierarchical (2-level) paging:
  - outer page table tells where each part of the page table is stored, i.e. indexes into the page table


RAM
outer page table
page table
dotted arrow means "go to memory and retrieve the frame pointed to by the table entry"

Hierarchical Paging

- Hierarchical (2-level) paging divides the logical address into 3 parts:



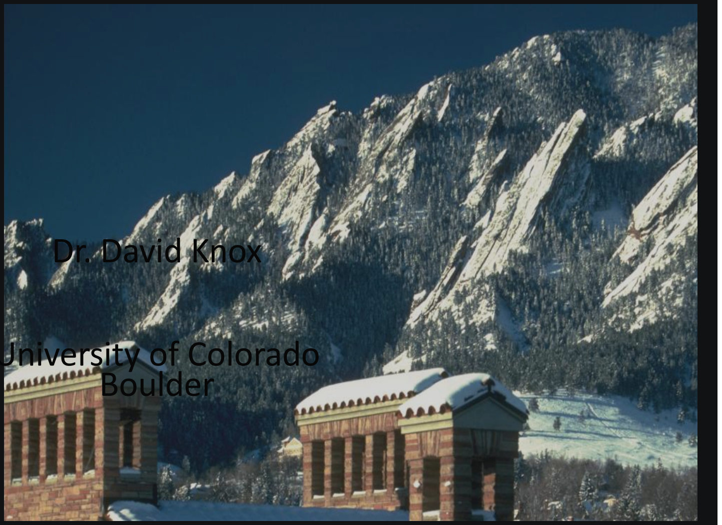
Hierarchical Paging

- While hierarchical page tables enable a large page table to be split to fit into memory, they do not solve the problem of the large size of the page table
  - if several processes have a page table that contains a million entries, then much of RAM may become devoted to storing/managing multiple page tables
  - The page tables may be *sparse* - many of the entries in the page table are just empty placeholders for logical pages that are not in memory, and may never be in memory
    - e.g. stack and heap may never grow large enough to use all of allocated memory

Department of Computer Science
UNIVERSITY OF COLORADO BOULDER

Design and Analysis of Operating Systems CSCI 3753

Dr. David Knox
University of Colorado Boulder



These slides adapted from materials provided by the textbook authors.