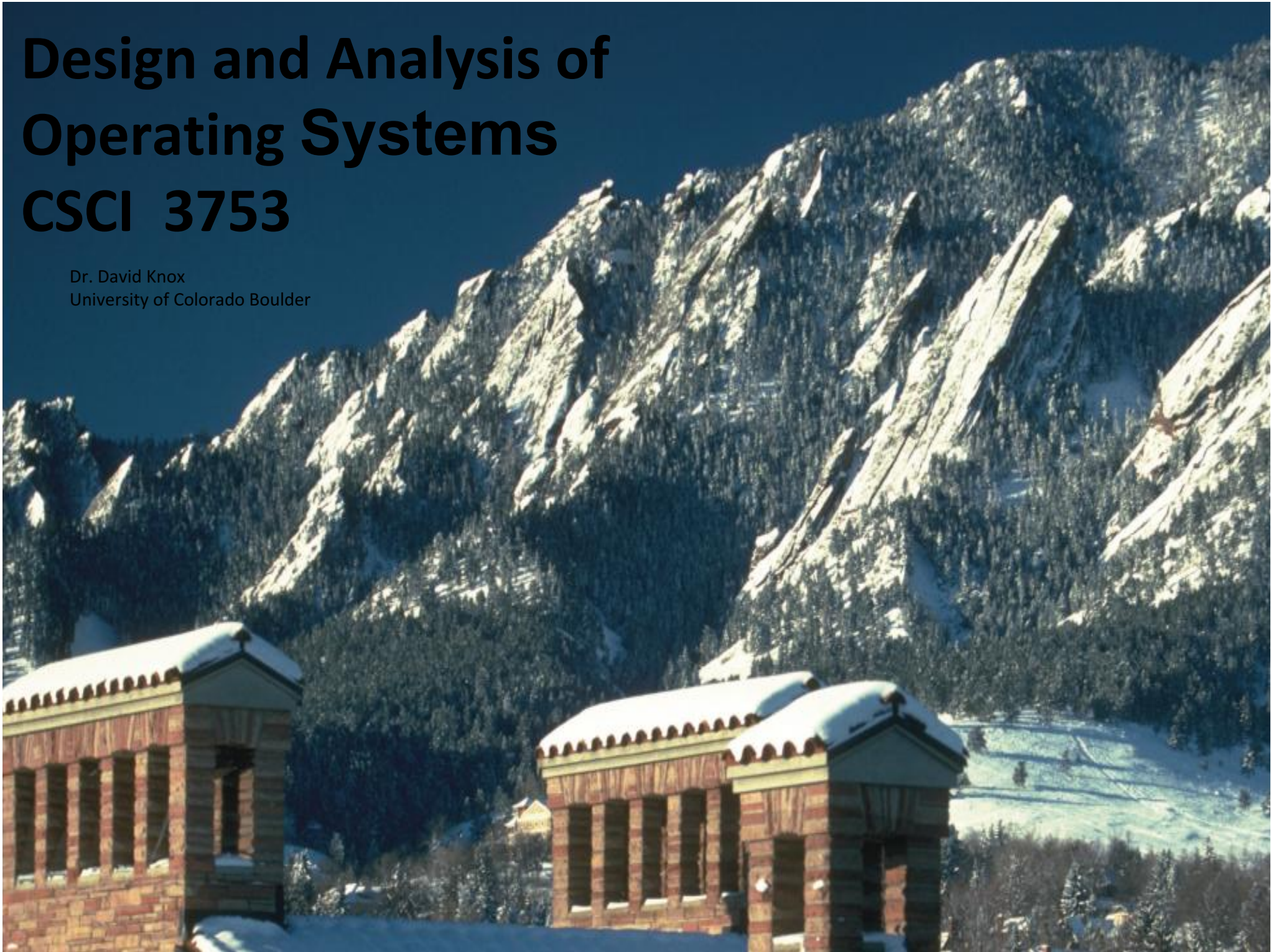


Design and Analysis of Operating Systems CSCI 3753

Dr. David Knox
University of Colorado Boulder





Department of Computer Science
UNIVERSITY OF COLORADO **BOULDER**



Design and Analysis of Operating Systems CSCI 3753

Memory Management Thrashing

Dr. David Knox
University of
Colorado Boulder

Material adapted from: Operating Systems: A Modern Perspective : Copyright © 2004 Pearson Education, Inc.

Memory Management Thrashing

Memory Management Thrashing

FIFO Page Replacement Example

Let page reference stream, $\mathcal{R} = 012301230123$

Frame	$\mathcal{R} = 0$	1	2	3	0	1	2	3	0	1	2	3
0	<u>0</u>	0	0	<u>3</u>	3	3	<u>2</u>	2	2	<u>1</u>	1	1
1		<u>1</u>	1	<u>1</u>	<u>0</u>	0	0	<u>3</u>	3	<u>3</u>	<u>2</u>	2
2			<u>2</u>	2	2	<u>1</u>	1	1	<u>0</u>	0	0	<u>3</u>

- FIFO with $m = 3$ has 12 faults
- Every request is causing a page fault
- This is an extreme example of page thrashing

Thrashing

- **Describes situation of repeated page faulting**
 - this significantly slows down performance of the entire system, and is to be avoided
- **Occurs when a process' allocated # of frames < size of its recently accessed set of frames**
 - each page access causes a page fault (or the page fault rate is high)
 - must replace a page, load a new page from disk
 - but then the next page access also is not in memory, causing another page fault, resulting in a domino effect of page faults - this is called *thrashing*
 - a process spends more time page faulting to disk than executing – is I/O bound, causing a severe performance penalty

Thrashing

- **Thrashing under local replacement: an example**
 - a single process P1 exhibits poor locality in code and/or data but processes P2 through PN exhibit strong locality in code/data.
 - Suppose P1 is not allocated enough memory frames, but P2 through PN have enough frames.
 - P1 will thrash, but local replacement confines the thrashing to P1. P2 through PN will not thrash.
 - The isolation is not perfect. P1's disk reads and writes will slow down other processes that have disk I/O

Thrashing

■ Thrashing under global replacement

- a process needs more frames, page faults and takes frames away from other processes, which then take frames from others, ... - domino effect
- as processes queue up for the disk, CPU utilization drops
- The process isolation is not perfect. A process's disk reads and writes will slow down other processes that have disk I/O
- OS can add fuel to the fire:
 - suppose OS has the policy that if it notices CPU utilization dropping, then it thinks that the CPU is free, so it restarts some processes that have been frozen in swap space
 - these new processes page fault more, causing CPU utilization to drop even further

Thrashing

- **Solution 1: Build a *working set* model.**

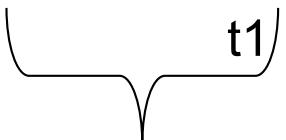
- Programs tend to exhibit *locality* of behavior, i.e. they tend to access/reuse same code/data pages
- Find a set of recently accessed pages that captures this locality
- To measure locality, define a window size Δ of the Δ most recent page references
 - Within the set of pages in Δ , the working set is the set of *unique* pages
 - pages recently referenced in working set will likely be referenced again
- Then allocate to a process the size of the working set

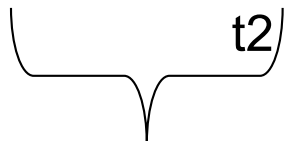
Thrashing

■ Working set solution (continued):

- Example: Assume $\Delta = 4$ and we have a page reference sequence of

2, 7, 3, 2, 0, 1, 2, 1, 2, 1, 0, 5, ...


working set at t1,
WS1 = {0,2,3,7}
working set size = 4


working set at t2, WS2 = {1,2}
working set size = 2

- at time t1, process only needs to be allocated 4 frames; at time t2, process only needs 2 frames
- Is $\Delta = 4$ the right size window to capture the locality?

Thrashing

- **Working set solution (cont.)**

- Choose window Δ carefully

- if Δ is too small, Δ won't capture the locality of a process
 - if Δ is too large, then Δ will capture too many frames that aren't really relevant to the local behavior of the process
 - which will result in too many frames being allocated to the process

Thrashing

■ Working set solution (cont.)

- Once Δ is selected, here is the working set solution:
 1. Periodically compute a working set and working set size WSS_i for each process P_i
 2. Allocate at least WSS_i frames for each process
 3. Let demand $D = \sum WSS_i$. If $D > m$ total # of free frames, then there will be thrashing. So swap out a process, and reallocate its frames to other processes.
- This working set strategy limits thrashing

Thrashing

- **Approximate working set with a timer & a reference bit**
 - set a timer to expire periodically
 - any time a page is accessed, set its reference bit
 - at each expiration of the timer, shift the reference bit into the most significant bit of a record kept with each page,
 - each byte records references in the last 8 timer epochs
 - If any reference bit is set during the last N timer intervals, then the page is considered part of the working set, i.e. if record > 0
 - re-allocate frames based on the newly calculated working sets. Thus, it is not necessary to recalculate the working set on every page reference.

Thrashing

- **Solution 2: Instead of using a working set model, just directly measure the page fault frequency (PFF)**
 - When $PFF > \text{upper threshold}$, then increase the # frames allocated to this process
 - When $PFF < \text{lower threshold}$, then decrease the # frames allocated to this process (it doesn't need so many frames)

Linux Global Page Replacement

- **Doesn't explicitly use working sets to avoid thrashing**
- **Instead, applies Clock-like LRU to entire pool of all process' pages**
 - The evicted page is the least recently used globally over all processes, not just one process
 - This should be the best candidate in the entire system to remove, hence globally optimal by minimizing thrashing – won't need it again soon, as would occur for thrashing
 - As a process needs more pages, its effective working set expands and it takes pages from others
 - But the evicted page is the global LRU – least likely to be used again soon by the entire system, hence minimizing thrashing

Linux Global Page Replacement

- **Hence, Linux's approach:**
 - Effectively and adaptively allocates in an implicit way the working set to each process
- **Thrashing can still occur if the sum of all process' working sets exceeds size of physical memory**
 - In this case, Linux has a mechanism to terminate processes – rarely used



Department of Computer Science
UNIVERSITY OF COLORADO **BOULDER**



Design and Analysis of Operating Systems CSCI 3753



Dr. David Knox
University of
Colorado Boulder

Material adapted from: Operating Systems: A Modern Perspective : Copyright © 2004 Pearson Education, Inc.