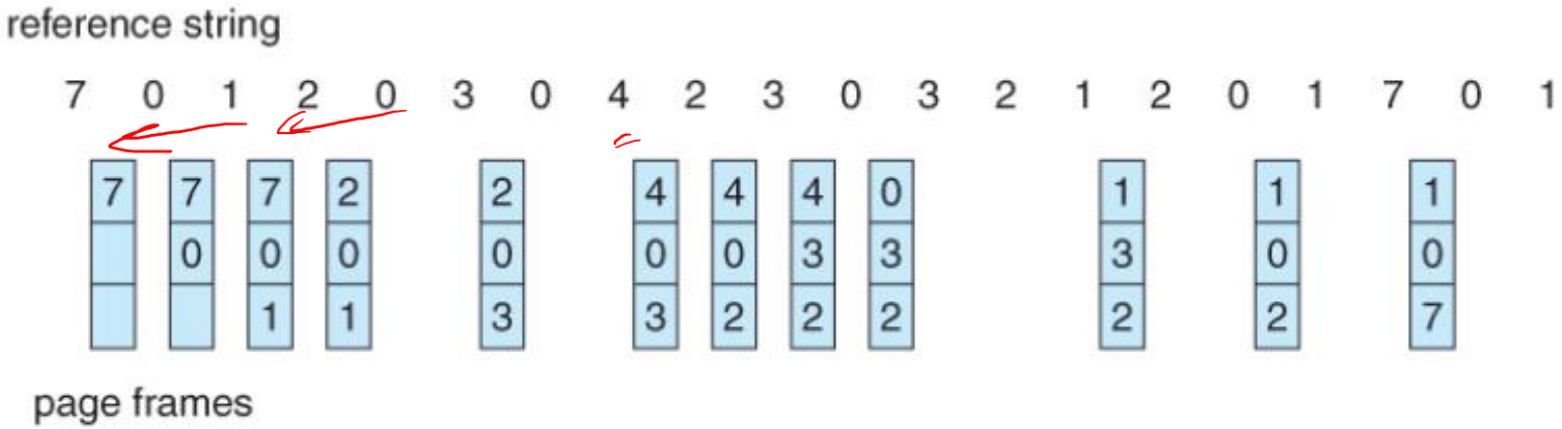
OPERATING SYSTEM: CSET209

CONTENT

- Page Replacement algorithms – LRU, counting algorithm
- Frame Allocation algorithms
- Thrashing

LRU PAGE REPLACEMENT

- **Least Recently Used (LRU)** algorithm replaces the page that has not been used in the longest time considering that it will not be used again in the near future.
- **Note the distinction between FIFO and LRU: The former looks at the oldest load time, and the latter looks at the oldest use time**.
- LRU is analogous to OPT, in the sense that OPT looks forward in time while LRU looks backward
- **EXAMPLE:**



- LRU yields 12 page faults

LRU PAGE REPLACEMENT

- LRU is considered a good replacement policy and is often used.
- Two simple approaches for its implementation.

1. **Counters:** Every memory access increments a counter, and the current value of this counter is stored in the page table entry for that page. Then finding the LRU page involves simply searching the table for the page with the smallest counter value.

2. **Stack:** Another approach is to use a stack, and whenever a page is accessed, pull that page from the middle of the stack and place it on the top. The LRU page will always be at the bottom of the stack.

NOTE:

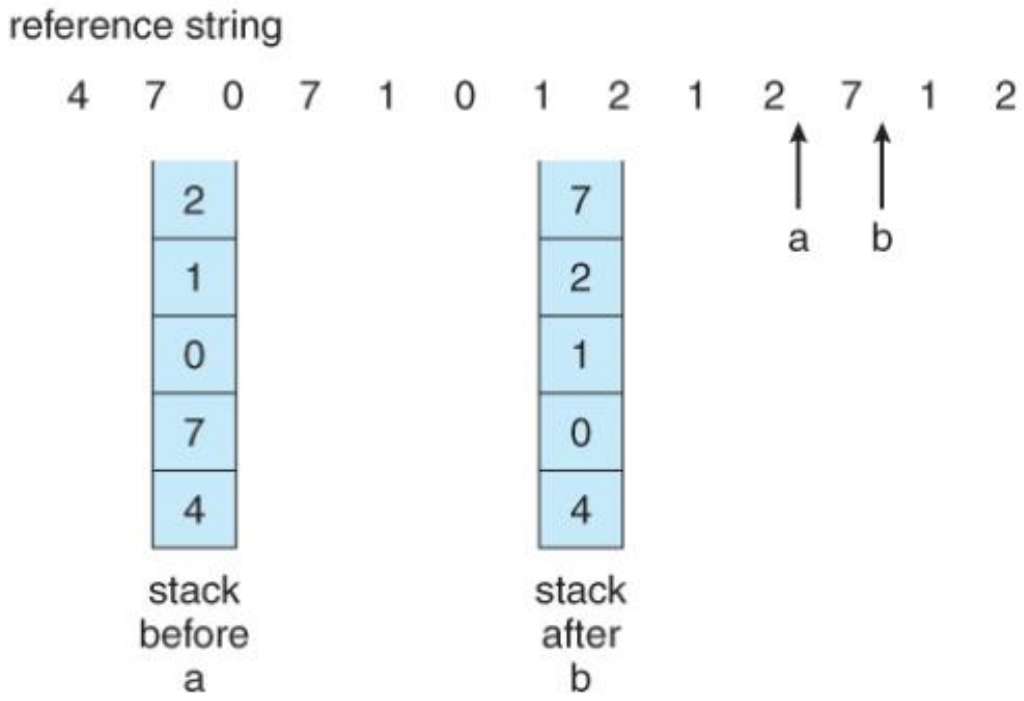
Neither LRU or OPT exhibit Belady's anomaly. Both belong to a class of page-replacement algorithms called

Stack algorithms, which can never exhibit Belady's anomaly.

A stack algorithm is one in which the pages kept in memory for a frame set of size N will always be a subset of the pages kept for a frame size of N + 1.

LRU PAGE REPLACEMENT

- In the case of stack implementation of LRU, the top N pages of the stack will be the same for all frame set sizes of N or anything larger. Example:



COUNTING-BASED PAGE REPLACEMENT

Counting-Based Page Replacement

Least Frequently Used, LFU:

Replace the page with the lowest reference count. A problem can occur if a page is used frequently initially and then not used anymore, as the reference count remains high. A solution to this problem is to right-shift the counters periodically, yielding a time-decaying average reference count.

Most Frequently Used, MFU:

Replace the page with the highest reference count. The logic behind this idea is that pages that have already been referenced a lot have been in the system a long time, and we are probably done with them, whereas pages referenced only a few times have only recently been loaded, and we still need them.

In general counting-based algorithms are not commonly used, as their implementation is expensive and they do not approximate OPT well.

EXAMPLE

- 7 0 2 4 3 1 4 7 2 0 4 3 0 3 2 7

EXAMPLE

7 0 2 4 3 1 4 7 2 0 4 3 0 3 2 7

String	7	0	2	4	3	1	4	7	2	0	4	3	0	3	2	7
Frame 3			2	2	2	1	1	1	2	2	2	3	3	3	3	3
Frame 2		0	0	0	3	3	3	7	7	0	0	0	0	0	2	7
Frame 1	7	7	7	4	4	4	4	4	4	4	4	4	4	4	4	4
Miss/Hit	M	M	M	M	M	M	H	M	M	M	H	M	H	H	M	M

Total number of reference strings = 16

Total number of page faults or page misses = 12

FRAME ALLOCATION

- The absolute minimum number of frames that a process must be allocated is dependent on system architecture and corresponds to the worst-case scenario of the number of pages that could be touched by a single (machine) instruction.
- If an instruction (and its operands) spans a page boundary, then multiple pages could be needed just for the instruction fetch.
- System architectures place a limit (say 16) on the number of levels allowed for an instruction

FRAME ALLOCATION ALGORITHMS

Allocation Algorithms

1. **Equal Allocation** – If there are m frames available and n processes to share them, each process gets m / n frames, and the leftovers are kept in a free-frame buffer pool.

2. **Proportional Allocation -** Allocate the frames proportionally to the size of the process, relative to the total size of all processes. So if the size of process *i* is S_i, and *S* is the sum of all S_i, then the allocation for process P_i is $a_i = m * S_i / S$

   - *E.g. With proportional allocation, we would split 62 frames between two processes, one of 10 pages and one of 127 pages, by allocating 4 frames and 57 frames, respectively, since **10/137 x 62 ~ 4, and 127/137 X 62 ~ 57**. In this way, both processes share the available frames according to their "needs," rather than equally.*

All allocations fluctuate over time as the number of available free frames, m, fluctuates, and all are also subject to the constraints of minimum allocation

GLOBAL VERSUS LOCAL ALLOCATION

Global versus Local Allocation

Frame allocation can be local or global:

- In local allocation, the number of pages allocated to a process is fixed, and page replacement occurs only amongst the pages allocated to this process.
- In global allocation, any page may be a potential victim, whether it currently belongs to the process of seeking a free frame or not.

- Local page replacement allows processes to better control their own page fault rates and leads to more consistent performance of a given process.
- Global page replacement is overall more efficient and is the more commonly used approach.

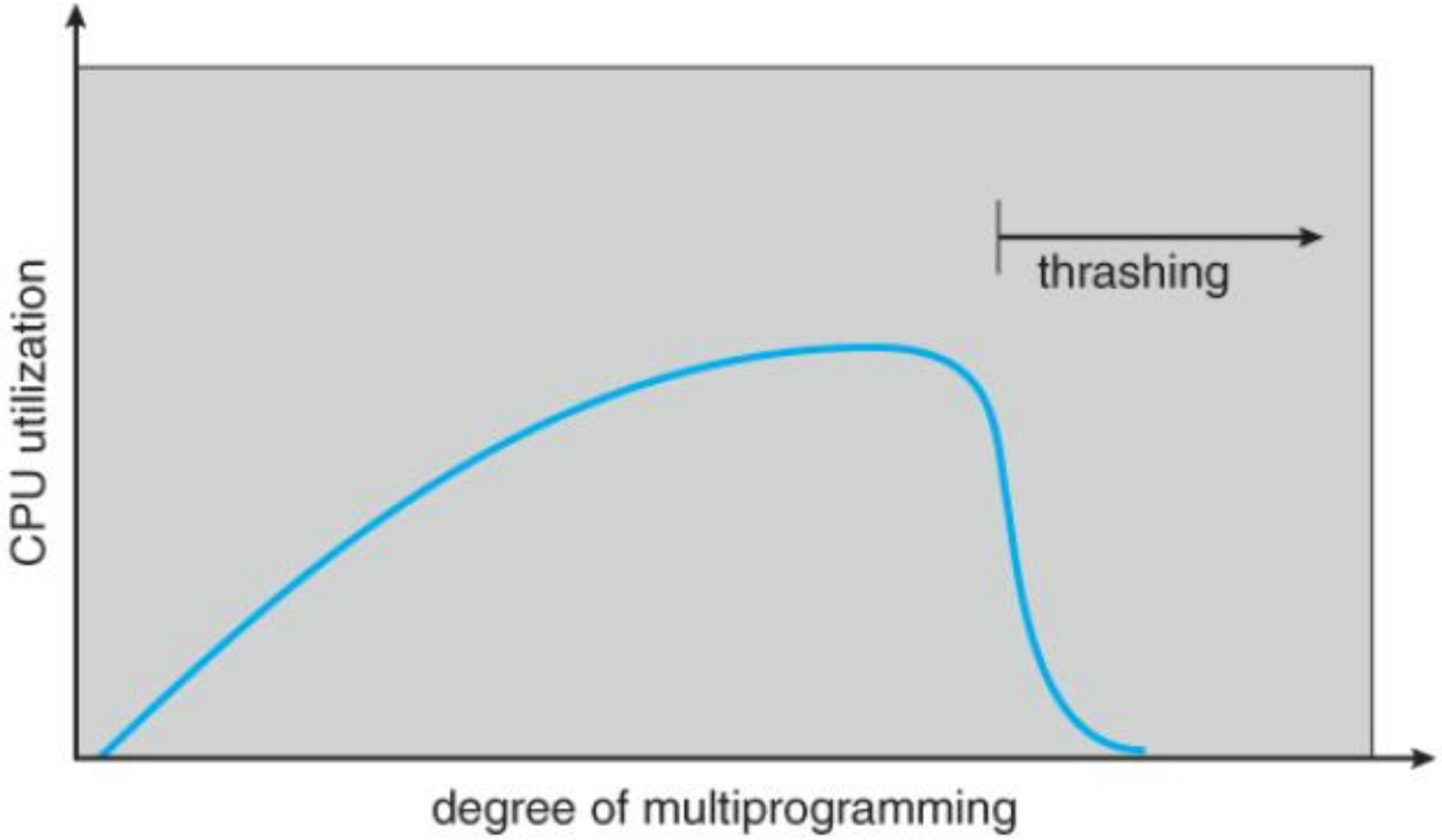
THRASHING

When a process cannot keep all of the frames that it is currently using, it is forced to page out pages that it will need again in the very near future, leading to large numbers of page faults.

A process that is spending more time paging than executing is said to be thrashing.

The problem is that when memory filled up and processes started spending lots of time waiting for their pages to page in, then CPU utilization would lower, causing the schedule to add in even more processes and exacerbating the problem! Eventually the system would essentially grind to a halt.

THRASHING



THRASHING

To prevent thrashing we must provide processes with as many frames as they really need "right now", but how do we know what that is?

The locality model notes that processes typically access memory references in a given

locality, making lots of references to the same general area of memory before moving periodically to a new locality.

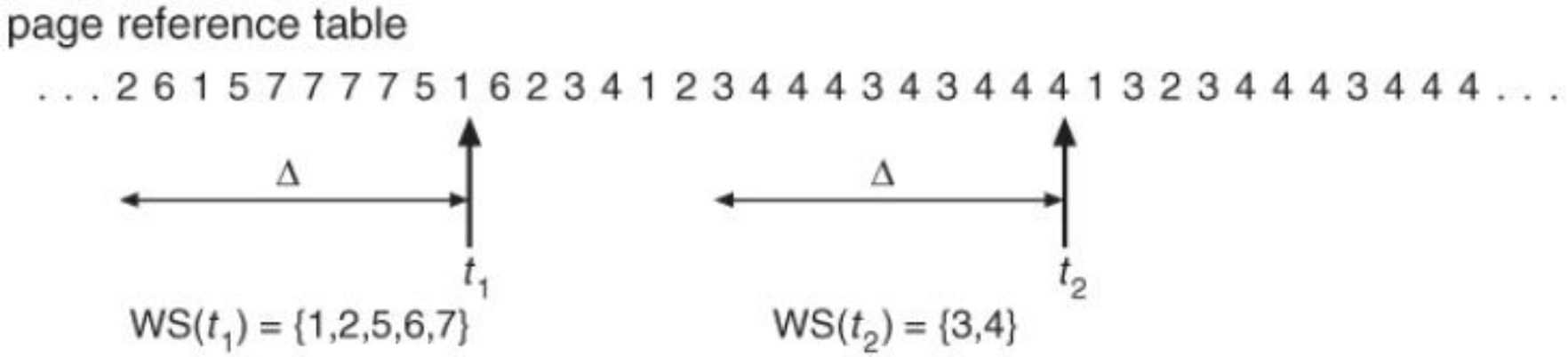
Locality is a set of pages that are actively used together.

If we could just keep as many frames as are involved in the current locality, then page faulting would occur primarily on switches from one locality to another.

THRASHING

Working-Set Model

The working set model is based on the concept of locality, and defines a working set window, of length delta. Whatever pages are included in the most recent delta page references are said to be in the processes working set window, and comprise its current working set



THRASHING

Working-Set Model

If delta is too small then it does not encompass all of the pages of the current locality,

If it is too large, then it encompasses pages that are no longer being frequently accessed.

The total demand, D, is the sum of the sizes of the working sets for all processes. If D exceeds the total number of available frames, then at least one process is thrashing, because there are not enough frames available to satisfy its minimum working set. If D is significantly less than the currently available frames, then additional processes can be launched.

THANK YOU