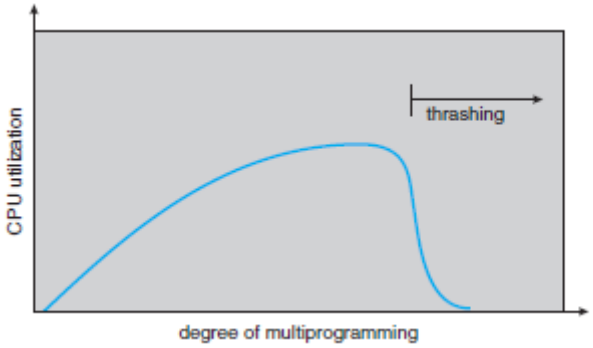
THRASHING:

- **If the process does not have the number of frames it needs to support pages in active use, it will quickly page-fault. At this point, it must replace some page. If all its pages are in active use, it must replace a page that will be needed again right away. So it quickly faults again, and again, and again, replacing pages that it must bring back in immediately. This high paging activity is called thrashing.**
- A process is thrashing if it is spending more time paging than executing.

Causes of Thrashing:

- The operating system monitors CPU utilization If CPU utilization is too low; we increase the degree of multiprogramming by introducing a new Process to the system.
- Now suppose that a process enters a new phase in its execution and needs more frames. It starts faulting and taking frames away from other processes.
- **A global page-replacement algorithm is used**; it replaces pages without regard to the process to which they belong.
- These processes need those pages, however, and so they also fault, taking frames from other processes. These faulting processes must use the paging device to swap pages in and out. As processes wait for the paging device, CPU utilization decreases.
- The CPU scheduler sees the decreasing CPU utilization and ***increases*** the degree of multiprogramming as a result. The new process tries to get started by taking frames from running processes, causing more page faults and a longer queue for the paging device.
- As a result, CPU utilization drops even further, and the CPU scheduler tries to increase the degree of multiprogramming even more. **Thrashing has occurred,** and system throughput plunges.



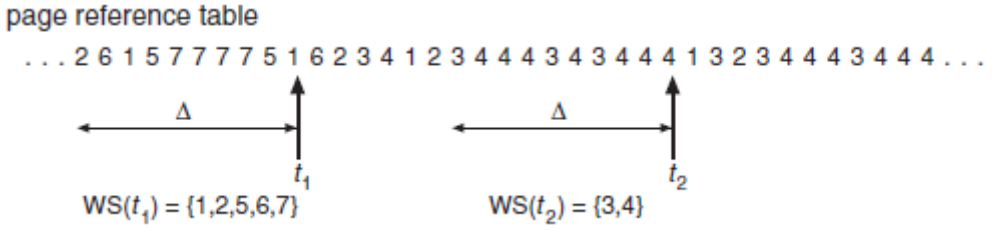
- At this point, to increase CPU utilization and stop thrashing, we must ***decrease*** the degree of Multi programming.
- We can limit the effects of thrashing by using a **local replacement algorithm.** With local replacement, if one process starts thrashing, it cannot steal frames from another process, so the page fault of one process does not affect the other process.
- To prevent thrashing, we must provide a process with as many frames as it needs. The OS need to know how many frames are required by the process.
- The working-set strategy starts by looking at how many frames a process is actually using. This approach defines the **locality model** of process execution.
- A locality is a set of pages that are actively used together. A program is generally composed of several different localities, which may overlap.
- Suppose we allocate enough frames to a process to accommodate its current locality. It will fault for the pages in its locality until all these pages are in memory; then, it will not fault again until it changes localities.
- If we do not allocate enough frames to accommodate the size of the current locality, the process will thrash, since it cannot keep in memory all the pages that it is actively using.

Working-Set Model

- The **working-set model** is based on the assumption of locality.
- This model uses a parameter Δ to define the **working-set window**.
- The idea is to examine the most recent Δ page references.
- The set of pages in the most recent Δ page references is the **working set.**

- If a page is in active use, it will be in the working set.
- If it is no longer being used, it will drop from the working set Δ time units after its last reference.

- **EXAMPLE:** Consider the sequence of memory references shown. If Δ = 10 memory references, then the working set at time *t*1 is *{*1, 2, 5, 6, 7*}*. By time *t*2, the working set has changed to *{*3, 4*}*.



- If Δ is too small, it will not encompass the entire locality;
- If Δ is too large, it may overlap several localities.
- If Δ is infinite, the working set is the set of pages touched during the process execution.
- The most important property of the working set, then, is its size. If we compute the working-set size, *WSSi* , for each process in the system, we can then consider that

$$D = \sum WSS_i,$$

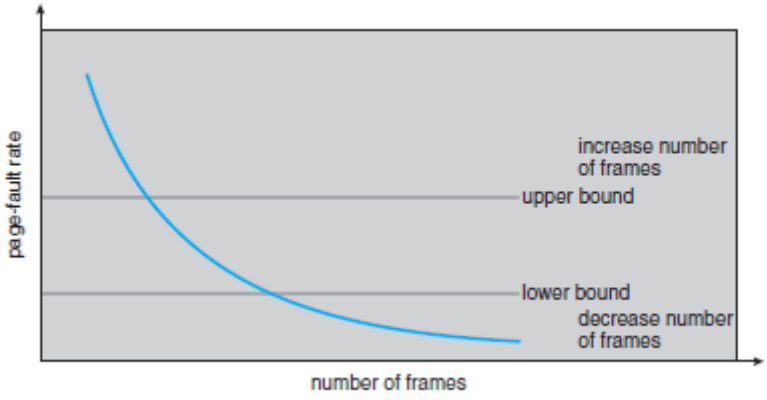
- Here *D* is the total demand for frames.
- Thus, process *i* needs *WSSi* frames. If the total demand is greater than the total number of available frames (*D*> *m*), thrashing will occur, because some processes will not have enough frames.
- If there are enough extra frames, another process can be initiated.
- If the sum of the working-set sizes increases, exceeding the total number of available frames, the operating system selects a process to suspend.
- This working-set strategy prevents thrashing while keeping the degree of multiprogramming as high as possible.

Page-Fault Frequency

- The page fault frequency is calculated by the total number of faults to the total number of references.

Page fault frequency = No. of page Faults / No. of References

- Thrashing has a high page-fault rate. Thus, we want to control the page-fault rate.
- When it is too high, we know that the process needs more frames.
- Conversely, if the page-fault rate is too low, then the process may have too many frames.
- Establish an upper and lower bounds on the desired page-fault rate.
- If the actual page-fault rate exceeds the upper limit, we allocate the process another frame.



- If the page-fault rate falls below the lower limit, we remove a frame from the process.
- Thus, we can directly measure and control the page-fault rate to prevent thrashing.
- If the page-fault rate increases and no free frames are available, we must select some process and swap it out to backing store.
- The freed frames are then distributed to processes with high page-fault rates