==========

- Virtual memory is a technique that allows the execution of processes that may not be completely in memory.
- Implemented via
    - Demand Paging
    - Demand Segmentation

// ADVANTAGES OF VIRTUAL MEMORY
==========

- Copy-on-Write
    - Copy-on-Write (COW) allows both parent and child processes to initially share the same pages in memory.
    - Only when the page is modified, then child creates a new copy
- Memory Mapped Files
    - A file is initially read using demand paging. A page-sized portion of the file is read from the file system into a physical page. Subsequent reads/writes to/from the file are treated as ordinary memory accesses

// DEMAND PAGING
==========

- For demand paging, we use a lazy swapper.
    - Bring a page into memory only when it is needed.
        - Less I/O needed
        - Less memory needed
        - Faster response
        - More users
- **Mechanism**
    - **If Page is needed =====> reference to it**
    - **If invalid reference ====> abort**
    - **If not-in-memory ====> bring to memory**
        - **With each page table entry a valid–invalid bit is associated**
        - **(1 in-memory, 0 page fault)**
        - **Get an empty frame in Main Memory.**
        - **Swap page into frame.**
        - **valid bit = 1.**
        - **Restart instruction**
        - **What happens if there is no free frame in Main Memory?**

- **find some used frame in Main Memory, but not really in use, swap it out(victim frame)(page replacement)**
- **Note: Dirty bit**
    - **Use modify (dirty) bit to reduce overhead of page transfers – only modified pages are written to disk.**

- Overhead of page fault
    - Effective Access Time (EAT)
        - EAT = (1 – p) x memory access+ p (page fault overhead)
- Page Replacement Algorithms
    - FIFO
    - OPTIMAL
    - LRU
    -

// LRU IMPLEMENTATION

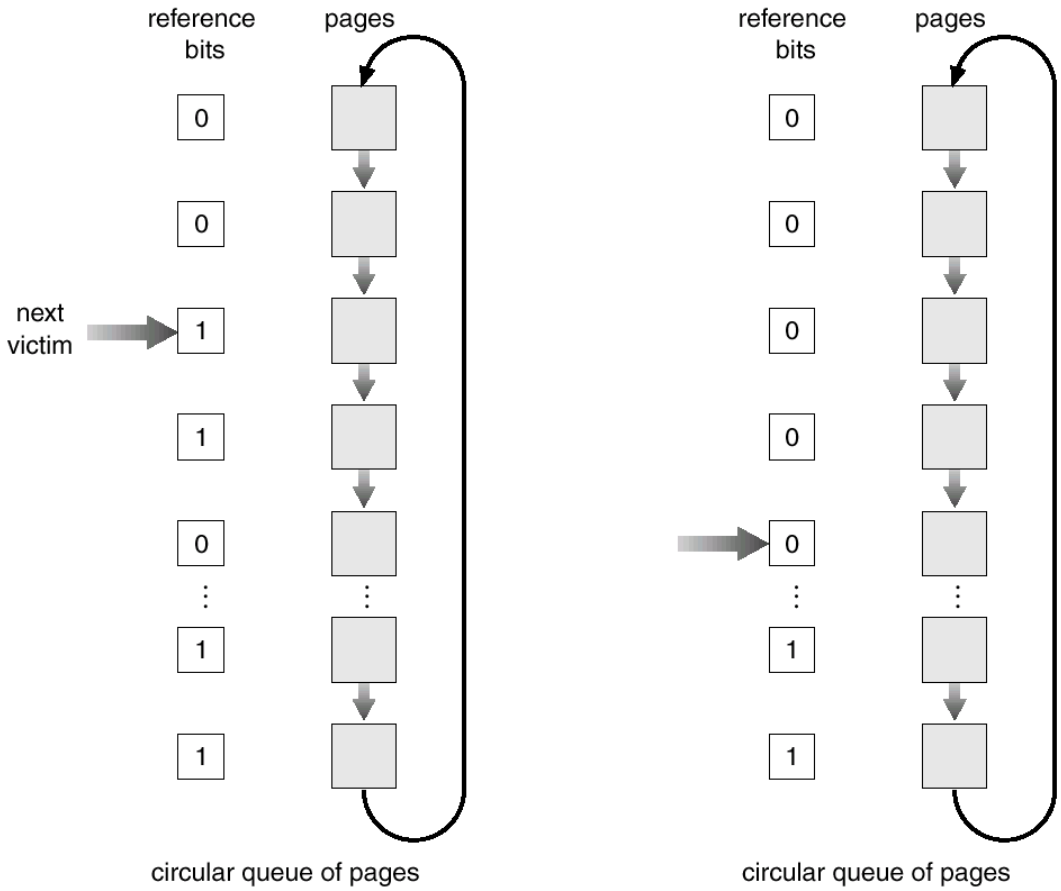
==========

- Counter implementation
    - Every page entry has a counter; every time a page is referenced through this entry, copy the clock into the counter.
    - When a page needs to be changed, look at the counters to determine which are to change.

- Stack implementation – keep a stack of page numbers:
    - IF a Page is referenced:
        - move it to the top
    - Look at the stack if a page needs to be changed

- Issues
    - The updating of stack or clock must be done on every memory reference.
    - If we use interrupt for every reference, to allow software to update data structures, it would slow every reference by a factor of 10
- Solutions
    - Reference bit
        - With each page associate a bit, initially = 0
        - When the page is referenced, bit is set to 1.
        - Replace the one which is 0 (if one exists). We do not know the order, however.
    - Second Chance Algorithm

- Page Reference Bit:
  - Each page in memory is associated with a reference bit, which is initially set to 0.
- Page Access:
  - When a page is accessed (read or written), its reference bit is set to 1, indicating that the page has been used recently.
- Page Replacement:
  - When a page needs to be replaced, the algorithm starts scanning the pages in a circular manner. It looks for a page with a reference bit of 0 (not recently used).
  - If it finds a page with a reference bit of 0, it replaces that page.
  - If it finds a page with a reference bit of 1, it gives that page a "second chance" by setting its reference bit to 0 and continues scanning.



//FRAME ALLOCATION SCHEMES - CONTEXT 1
==========

- fixed allocation
    - Equal allocation – e.g., if 100 frames and 5 processes, give each 20 pages.
    - Proportional allocation – Allocate according to the size of the process.

- priority allocation
    - Use a proportional allocation scheme using priorities rather than size.

//FRAME ALLOCATION SCHEMES - CONTEXT 2
==========

- Global replacement – process selects a replacement frame from the set of all frames; one process can take a frame from another.
- Local replacement – each process selects from only its own set of allocated frames

//THRASHING
==========

- Thrashing a process is spending more time swapping pages in and out.
    - If the process does not have # of frames equivalent to # of active pages, it will very quickly page fault.
    - Since all the pages are in active use it will page fault again

//WORKING SET MODEL
==========

- Working Set Window:
    - The working set model defines a "working set window" that represents a period of time. It's a sliding time interval during which the system monitors the pages accessed by a process.
- Working Set Size:
    - The working set size of a process is the number of distinct pages referenced within the working set window. It reflects the temporal locality of the process.
- Working Set Policy:
    - The working set policy involves adjusting the amount of memory allocated to a process based on its working set size. If the working

set size exceeds a certain threshold, the process may be given more memory to improve performance.

//LOCALITY MODEL

==========

Definition:

- The locality model is based on the observation that programs tend to access a small, localized portion of their address space at any given time. This observation is captured by the principle of locality.

Principle of Locality:

Temporal Locality:

- The principle of temporal locality states that if a program accesses a memory location, it is likely to access the same location again in the near future.

Spatial Locality:

- The principle of spatial locality states that if a program accesses a memory location, it is likely to access nearby memory locations in the near future.