Caching

- Optimizing access times to data stored in memory (*physcial*, *hard-disk*, or *other* caches).
- Only useful in certain instances i.e. when Temporal and Spatial Locality is exhibited.
    - **Temporal Locality:** Recently accessed memory is likely to be accesed again.
    - **Spatial Locality:** Addresses around recently accessed memory likely to be accessed.

Types of Caching Algorithms

- **Local Algorithms:** Each process has a fixed-number of pages in cache.
- **Global Algorithms:** Each process has a different-number of pages in cache dynamically calculated based on need.
    - *'need'* is based on **PFF** (Page Fault Frequency), which needs to be close to a defined value for performance reasons.
        - *NOT* true for **FIFO cache-management** algorithm.

Caching Algorithms

NOTE: If a cache entry was *modified* before being evicted, it *needs* to be written back to the main memory.

- **Not Recently Used:**
    - **Operation:**
        - Cache lines/entries have *R* (read) and *M* (Modified) bits.
        - *R* bit periodically cleared; *M* bit cleared after main-memory value updated.
    - **Cache hit:** *R* or *M* bits are modified appropriately
    - **Cache Miss:**
        - Cache entries are sorted into *'buckets'* based on R, M bit values.
        - **R, M** = 0, 0 < 0, 1 < 1, 0, < 1, 1 is the order of importance for cache entries to be evicted.
- **FIFO:**
    - **Operation:**
        - Cache lines/entries are elements of an array-implemented **FIFO queue.**
        - A *pointer* points to some cache-entry (the oldest one) at any given moment.
    - **Cache hit:** Nothing happens.
    - **Cache miss:** Currently pointed-to entry is evicted to be replaced; Pointer is **incremented** (*Pointer* will wrap around the array if required)

- **Second Chance:**
    - **Operation:**
        - Like **FIFO** technique.
        - Each entry has a *R* bit.
        - A *pointer* points to some cache-entry (the oldest one) at any given moment.
    - **Cache hit:** set the *R* bit.
    - **Cache miss:**
        - If *R* = 0**,** then we replace entry with new element, other-wise clear *R*;
        - Pointer is incremented (*Pointer* will wrap around the array if required.)

- **Least Rencently Used:**
    - **Operation:**
        - Like **Circular Double Queue** technique.
        - *Head* points to the oldest element.

    - **Cache hit:** move the element to the end of the list.
    - **Cache miss:**
        - Delete *Head* node.
        - Add new cache-entry to end of list.
- **Not Frequently Used:**
    - **Operation:**
        - Implemented using array.
        - Each entry has a byte-sized *counter.*
    - **Cache hit:**
        - *Counter* is incremented.
    - **Cache miss:**
        - Delete the element with lowest *counter*.
        - Add new entry with counter value set to '1'.
- **Not Frequently Used + Aging:**
    - **Operation:**
        - Implemented using array.
        - Each entry has a byte-sized *counter.*
    - **Cache hit:**
        - *Every* entry's *counter* is right-shifted.
        - *counter* of accessed entry has left-most bit changed to '1'.
    - **Cache miss:**
        - Delete lowest-value element.
        - Right-shift *all* counters
        - Set the left-most bit to '1'

Best Algorithm: LRU (with Hardware Support)**,** or **NFU + Aging**