Replacement Algorithms

- **Goal:** Select a page to be replaced when a new page must be swapped into memory

- **Basic Algorithms**
  - **Optimal (Impractical)**
    - Idea: Replace the P for which the time to the next reference is the longest.
    - Impossible to implement, but serves as a reference point
  - **First-In, First-Out (FIFO)**
    - Idea: Replace the oldest P
    - One of the simplest algorithms, but performs poorly
    - Treat page frames of a process as a circular buffer
    - Pages are removed in RR order
    - Implement as a pointer that cycles through the page frames of a process
  - **Least Recently Used (LRU)**
    - Idea: Select the P that hasn’t been referenced for the longest time in the past
    - Clock
      - Approximates LRU using a clock structure

- **Example Page Reference Stream:**
  - 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2
**LRU Page Replacement**

- **Least Recently Used (LRU)**
  - **Idea**: Replace the page that hasn’t been referenced for the longest time in the past
  - Does almost as well as optimal algorithm on some reference sequences
  - Difficult to implement in hardware
    - Time stamp each page and replace the oldest one
    - Use a stack with the most recently referenced page on top

---

**Clock Page Replacement**

- **Clock**
  - **Idea**: Faster approximation of LRU
    - Select an unmarked frame in RR order
    - Cursor cycles through page frames looking for an unmarked page as the replacement page
  - When to unmark the page underneath the cursor:
    - If the page is marked, then go to next page
    - If all pages are marked, the cursor will return to the first one it unmarked
  - When to mark a page:
    - When loaded into main memory
    - When already in main memory and it has been referenced

---

**Matrix Multiply (1 page per row)**

- **Reference strings for** $C[i][j] = A[i][k]*B[k][j]$
  - Let $i$, $a$, and $c$ be base page numbers and $x(n)$ be the nth page relative to the base page number $x$
  - Instructions: $i(0), i(0), i(0), \ldots$
  - $A$ array: $a(0), a(0), \ldots, a(1), \ldots, a(2), \ldots, a(3)$
  - $B$ array: $b(0), b(1), b(2), \ldots b(3), b(0), b(1), \ldots$
  - $C$ array: $c(0), c(0), \ldots, c(1), \ldots, c(2), \ldots, c(3)$
- **Composite reference string**
  - $a(0), b(0), a(i), b(1), \ldots, a(i), b(N-1), c(i), a(i), b(0), \ldots$
  - 1 page of $A$ is accessed $N$ times; $N$ pages of $B$ are accessed 1 time each
### Stress Case

- Example: 3 data pages; lock in 1 instruction page
- **Optimal**
  - *a(0), *b(0), a(0), *b(1), a(0), *b(2), a(0), *b(3), ...
  - Always replace a page of B array
- **FIFO**
  - *a(0), *b(0), a(0), *b(1), a(0), *b(2), *a(0), *b(3), ...
  - Sometimes replace a page of A array
- **LRU**
  - *a(0), *b(0), a(0), *b(1), a(0), *b(2), a(0), *b(3), ...
  - Same as optimal since A array is referenced frequently!
- **Clock**
  - *a(0), *b(0), a(0), *b(1), a(0), *b(2), a(0), *b(3), ...
  - A array is referenced frequently \( \Rightarrow \) A array is unmarked

### Replacement Algorithm Variations

- **Counting Algorithms**
  - Least Frequently Used (LFU): Select page with smallest reference count
  - Most Frequently Used (MFU): Select page with largest reference count
- **Use Bit (U) and Modified Bit (M)**
  - \( U = 1 \) \( \Rightarrow \) Page has been referenced recently
  - \( M = 1 \) \( \Rightarrow \) Page has been modified and needs to be written to disk
- **Page Buffering**
  - Replaced page is assigned to either the free page list (if \( M = 0 \)) or the modified list (otherwise)
- Page remains in memory
- Cluster page writes of modified pages

### The Working Set Model

- \( W(t, \Delta) \) is the working set at virtual time \( t \) with a window size of Delta and is:
  - Defined over the page reference string for each process
  - The set of pages that have been referenced in the time interval \([t - \Delta, t] \)
- **Example:**
  - \[ 9 \ 0 \ 3 \ 8 \ 9 \ *2 \ 3 \ 9 \ 3 \ 2 \ 2 \ 0 \ 9 \ 2 \ 9 \]
  - \( W(4, 4) = \{0, 3, 8, 9\} \)
  - \( W(2, 2) = \{0, 9\} \)
  - \( W(15, 5) = \{0, 2, 9\} \)
- \( W(t, \Delta) \) varies over time \( t \) even with a fixed window size \( \Delta \)

### The Working Set Strategy

- **The Strategy**
  - Monitor \( W(t, \Delta) \) for each process
  - Periodically remove pages from the resident set of a process that are not in its \( W(t, \Delta) \)
  - Schedule a process only if its working set is in main memory
- **Problems**
  - The past doesn’t always predict the future
  - An exact measurement of \( W(t, \Delta) \) is impractical because it requires a time-ordered queue of pages.
  - The optimal value of \( \Delta \) is unknown
Matrix Multiply

- What is the optimal working set size?
  - N+2?
    - If large enough Δ
    - N rows of B and 1 row each for A and C arrays
    - Number of page faults: 3N
  - 3?
    - Number of page faults: 2N + N^3
    - Every element of C will cause N page faults because of B array
  - 3?
    - Programmer transposes B first
    - Number of page faults: 3N

Page-Fault Frequency (PFF) Algorithm

- Idea: Adjust resident set size (RSS) according to page fault rate
- Basic Algorithm
  - Select a threshold F, the minimum time between page faults
  - Mark each page that is referenced with a use-bit (U) of 1
  - When a page fault occurs, compute the time F' since the last page fault and adjust the resident set size:
    - F' < F: Add a page to the resident set
    - F' ≥ F: Discard all pages with a use-bit (U) of 0, and shrink the resident set size
  - Reset all use-bits after a page fault
- Flaw: Poor performance during expanding transition periods

Variable-Interval Sampled WS

- Idea
  - Deal with transition periods by sampling more frequently and discarding unused pages when there are more page faults
- Algorithm Parameters
  - I−: Minimum duration of the sampling interval
  - I+: Maximum duration of the sampling interval
  - F: Number of page faults that are allowed to occur between sampling intervals
- Algorithm
  - Suspend process and scan the U-bits when:
    1) T = I−
    2) I− ≤ T < I+ and #Faults = F
  - RSS can only shrink at end of sampling interval; it either remains the same or increases within each interval