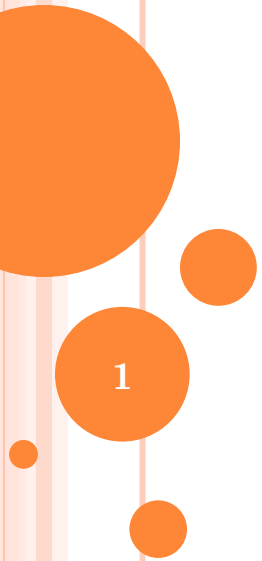
Comp 346



Memory Management (Virtual Memory)

Virtual Memory

- A process may be broken up into pieces (pages or segments) that do not need to be located contiguously in main memory
- Memory references are dynamically translated into physical addresses at run time
  - a process may be swapped in and out of main memory such that it occupies different regions
- These characteristics lead to a breakthrough: *it is not necessary for all pages/segments of a process to be in main memory during execution*
  - Execution can proceed as long as the instruction and data needed next are in main memory

Advantages of Partial Loading

- More processes can be maintained in main memory
  - We only load some of the pieces of each process, so there is room for more processes
  - with more processes in main memory, it is more likely that a process will be in the Ready state at any given time, improving processor utilization
- A process can now execute even if it is larger than the main memory size

Possibility of Thrashing

- To accommodate as many processes as possible, only a few pieces of each process are maintained in main memory
- But main memory may be full: when the OS brings one piece in, it must swap one piece out
- The OS must not swap out a piece of a process just before that piece is needed
- If it does this too often this leads to thrashing:
  - the processor spends most of its time swapping pieces rather than executing user instructions

Principle of Locality

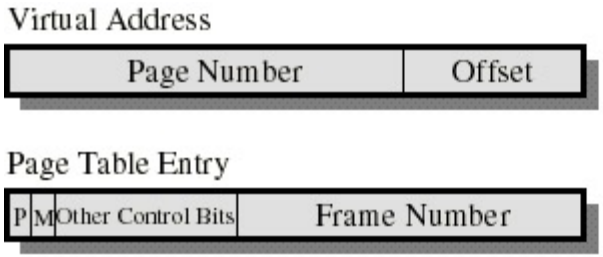
- Principle of locality states that program and data references within a process tend to cluster
- Hence, only a few pieces of a process will be needed over a short period of time
- Use principle of locality to make intelligent guesses about which pieces will be needed in the near future to avoid thrashing

Address Translation in a Paging System

- Translate virtual (logical) address into real (physical) address using page table
- Virtual address: (page #, offset)
- Real address: (frame #, offset)
- Page # is used to index the page table and look up the corresponding frame #
- Frame # combined with the offset produces the real address

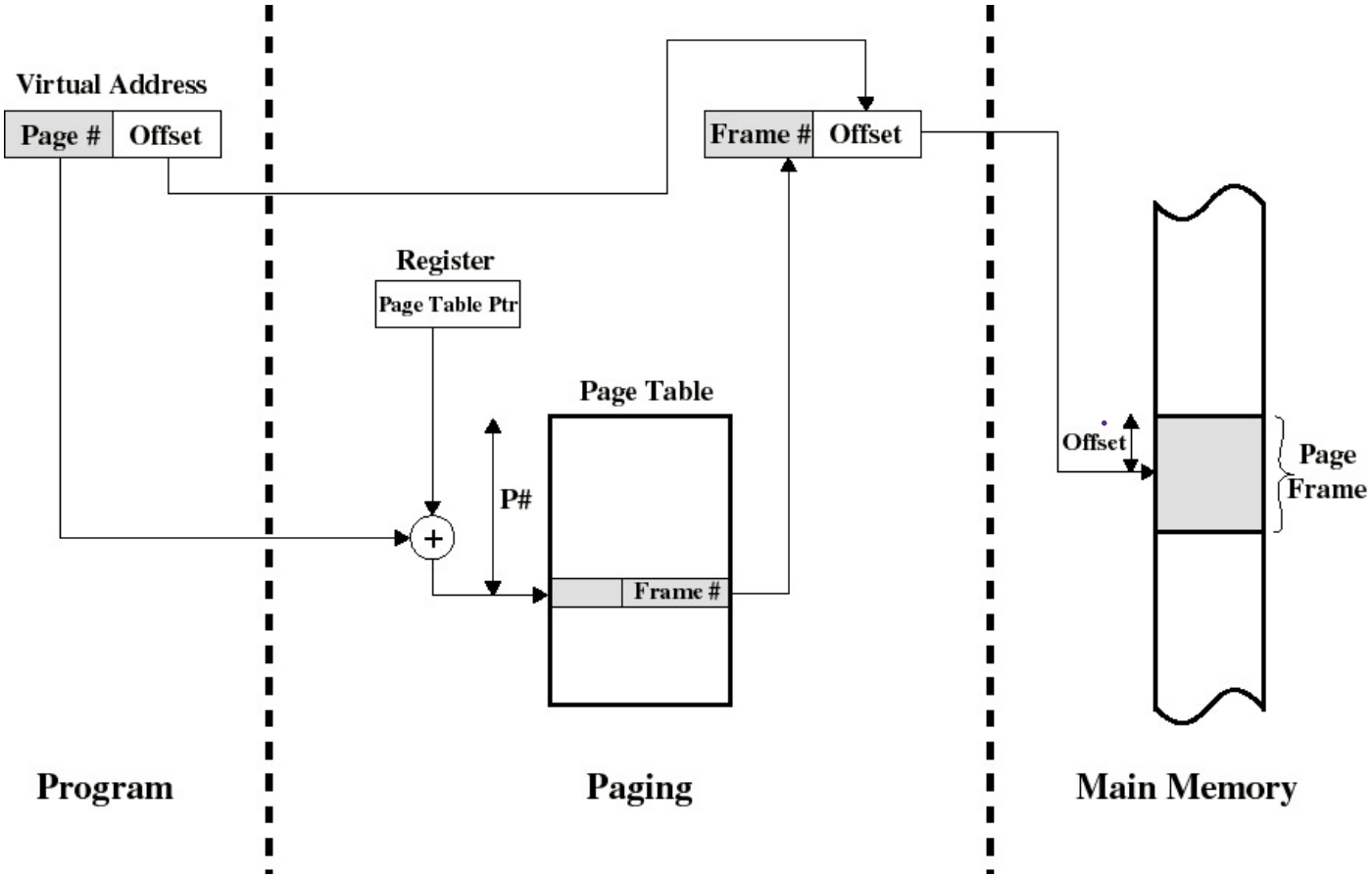
PAGING

- Each page table entry contains a *present* bit to indicate whether the page is in main memory or not.
  - if it is in main memory, the entry contains the frame number of the corresponding page in main memory
  - if it is not in main memory, the entry may contain the address of that page on disk or the page number may be used to index another table (often in the PCB) to obtain the address of that page on disk
- Typically, each process has its own page table



P= present bit
M = Modified bit

Address Translation in a Paging System



Page Tables and Virtual Memory

- Most computer systems support a very large virtual address space
  - 32 to 64 bits are used for logical addresses
  - if (only) 32 bits are used with 4KB pages, a page table may have 2^{20} entries
- The entire page table may take up too much main memory. Hence, page tables are often also stored in virtual memory and subjected to paging
  - when a process is running, part of its page table must be in main memory (including the page table entry of the currently executing page)

Multilevel Page Tables

- Since a page table will generally require several pages to be stored. One solution is to organize page tables into a multilevel hierarchy
  - when 2 levels are used (ex: 386, Pentium), the page number is split into two numbers p1 and p2
  - First level: root page table (outer page table or page directory)
  - Second level: process page table (user page table)
  - The root page table entries point to pages of the process page table
  - p1 indexes the outer page table and p2 indexes the resulting page in the process page table
  - Page directory is always kept in main memory, but part of the process page table may be swapped out.

Two-level Page Tables

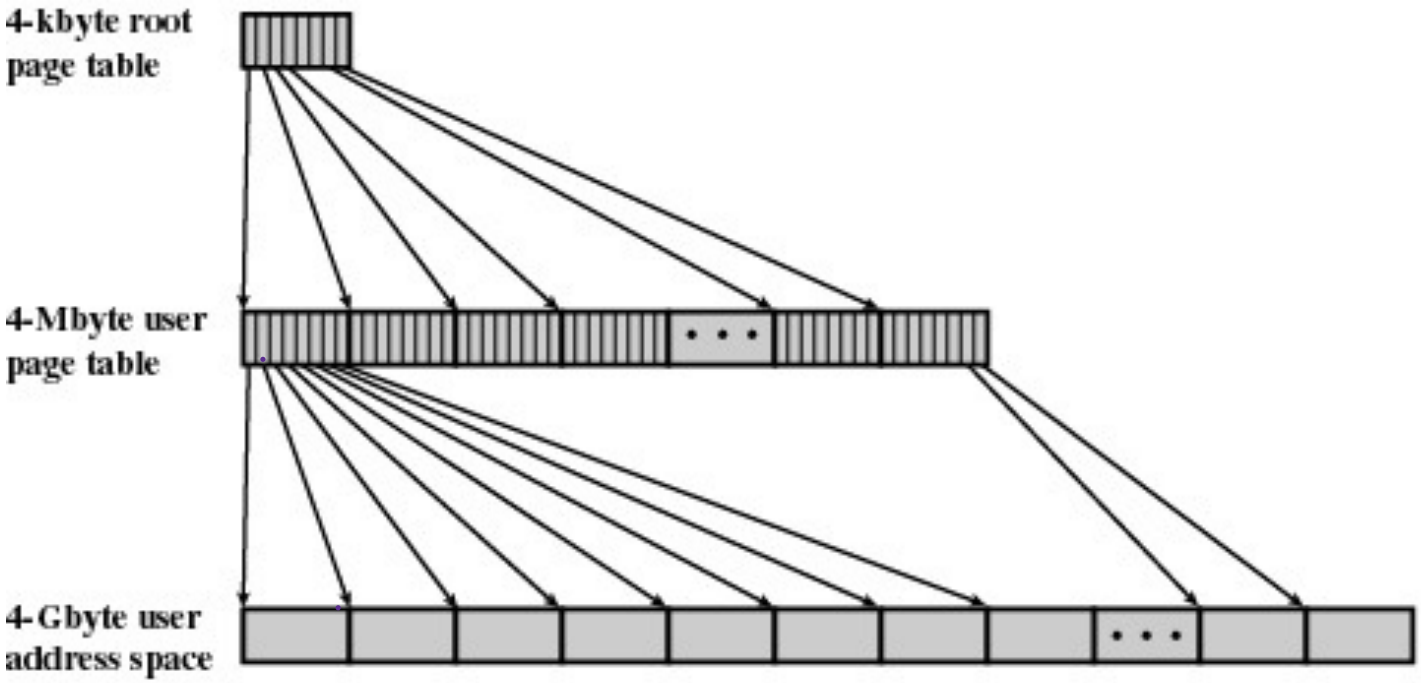
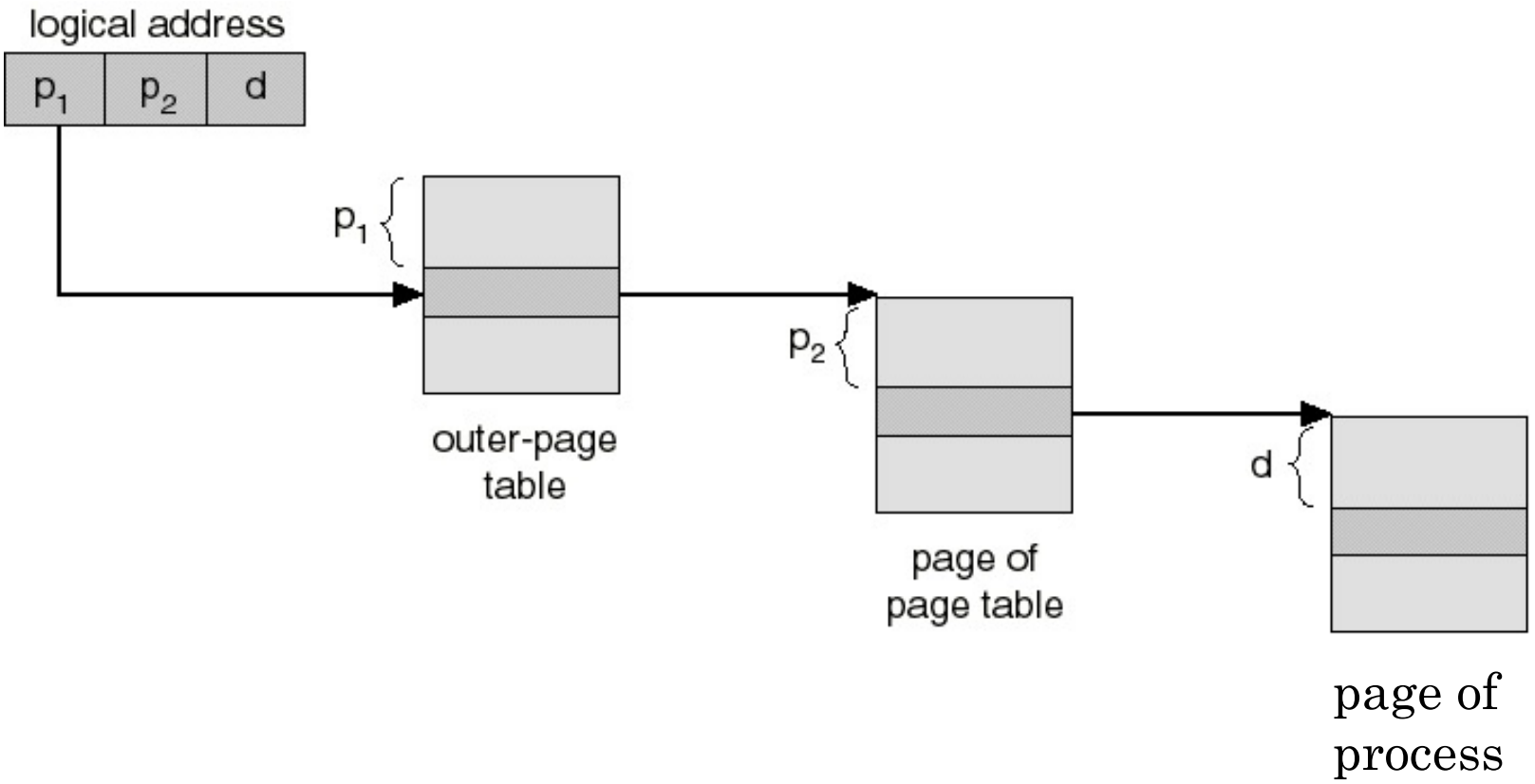


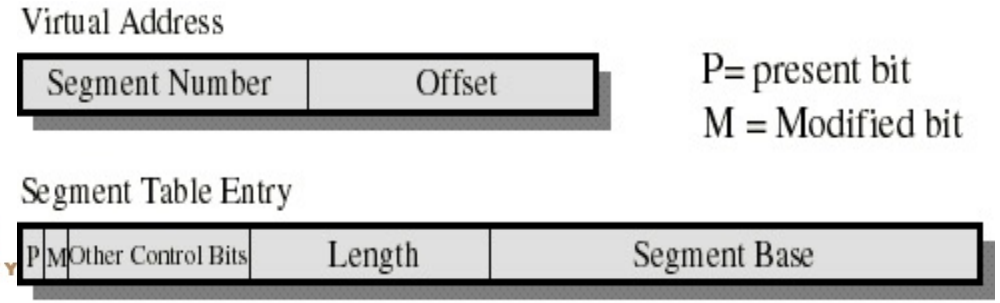
Figure 8.4 A Two-Level Hierarchical Page Table [JACO98a]

Two-level Page Tables

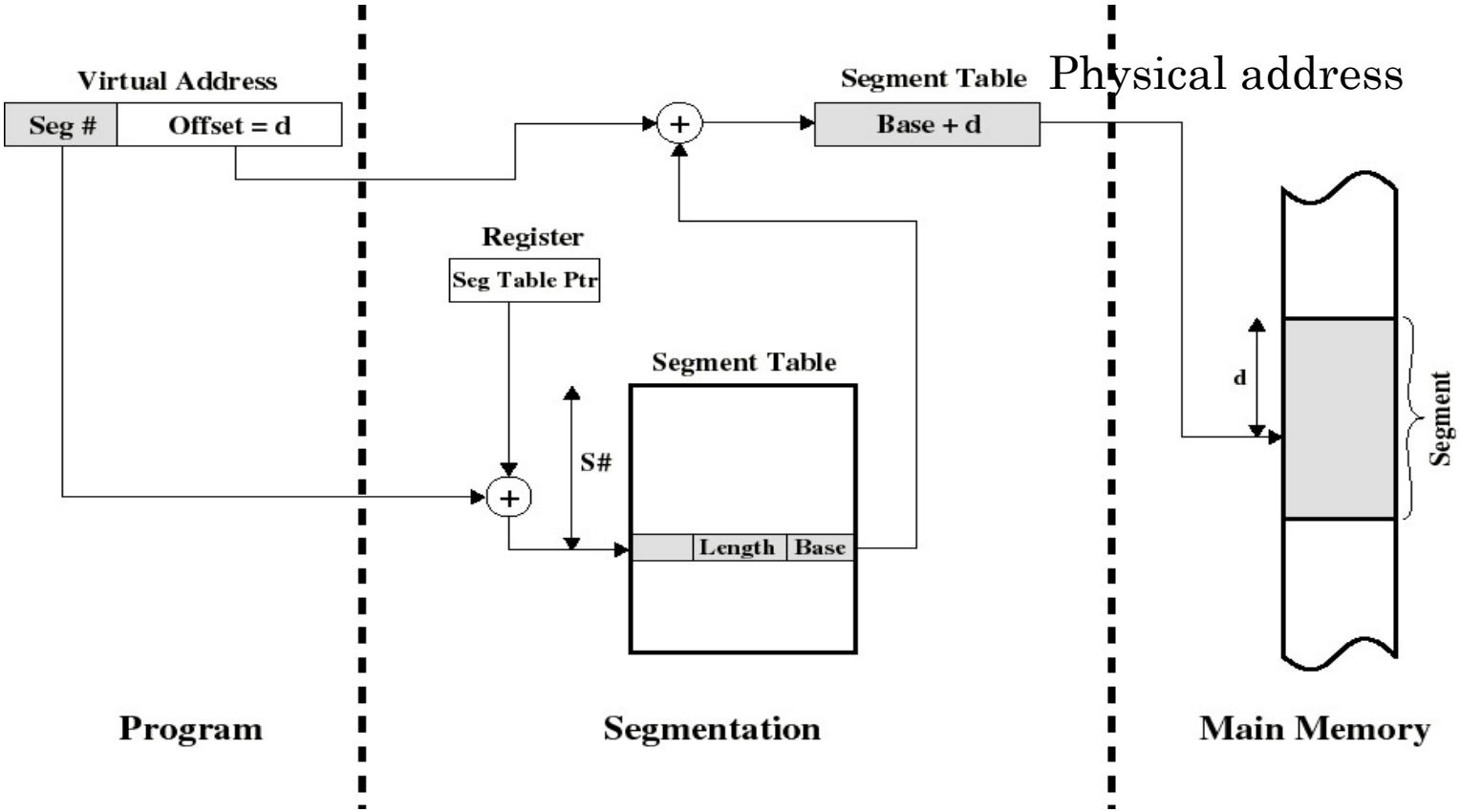


SEGMENTATION

- Typically, each process has its own segment table
- Similarly to paging, each segment table entry contains a present bit and a modified bit
- If the segment is in main memory, the entry contains the starting (base) address and the length of that segment
- Other control bits may be present if protection and sharing is managed at the segment level
- Logical to physical address translation is similar to paging except that the offset is added to the starting address (instead of being appended)



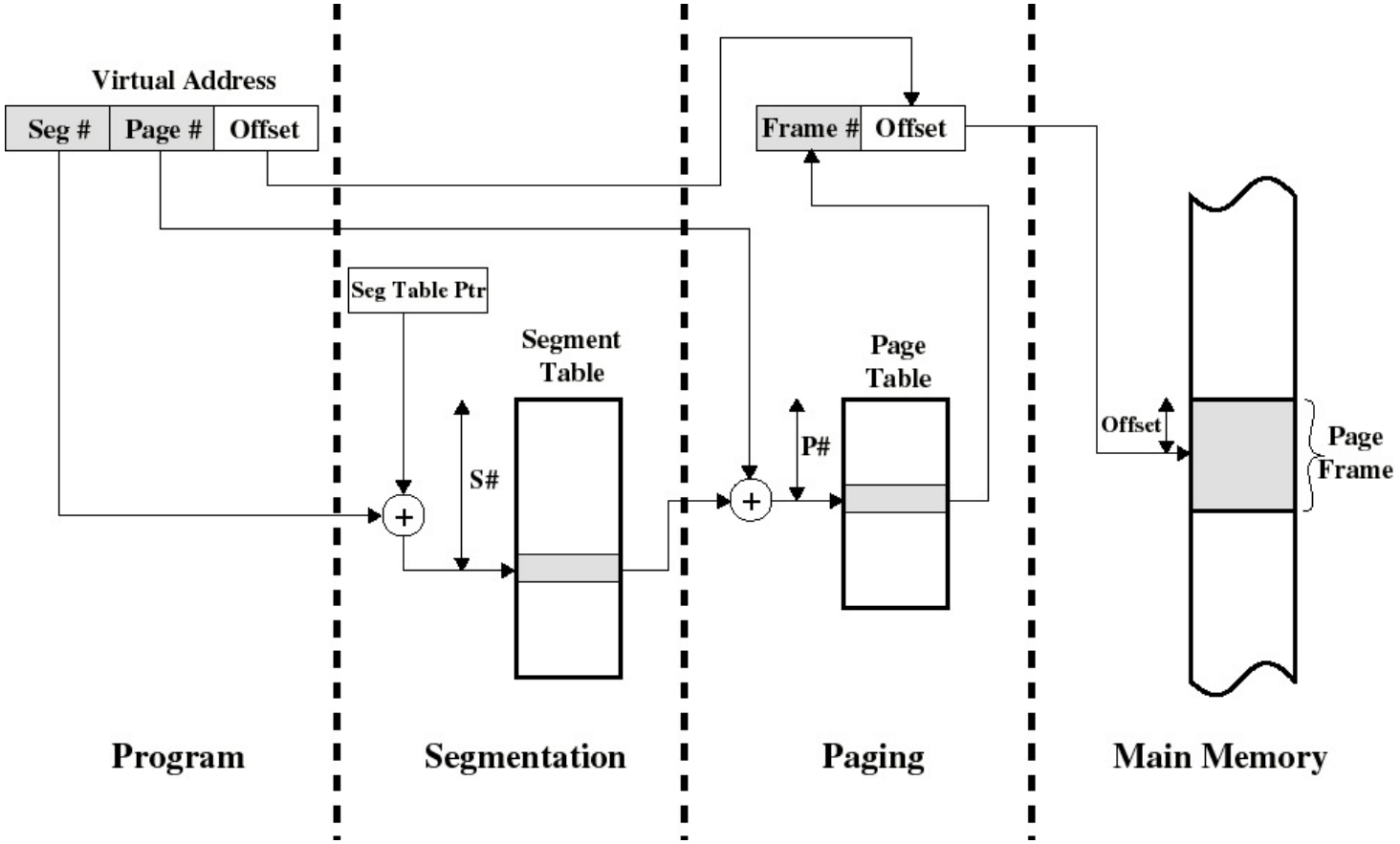
Address Translation in a Segmentation System



Combined Segmentation and Paging

- To combine their advantages some processors and OS page the segments
- Several combinations exists. Here is a simple one
- Each process has:
  - one segment table
  - several page tables: one page table per segment
- The virtual address consists of:
  - a segment number: used to index the segment table whose entry gives the starting address of the page table for that segment
  - a page number: used to index that page table to obtain the corresponding frame number
  - an offset: used to locate the word within the frame

Address Translation in a combined Segmentation/Paging System



Static Paging Algorithms

- When process starts, it is allocated a *fixed* number of frames in RAM.
- Paging Policies: defines how pages will be loaded/unloaded into frames allocated to the process.

Basic Paging Policies

1. **Fetch Policy**: determines ***when*** **a page** should **be *loaded*** into RAM.
2. **Replacement Policy**: if all frames are full, determines ***which page*** should be *replaced*.

Demand Paging

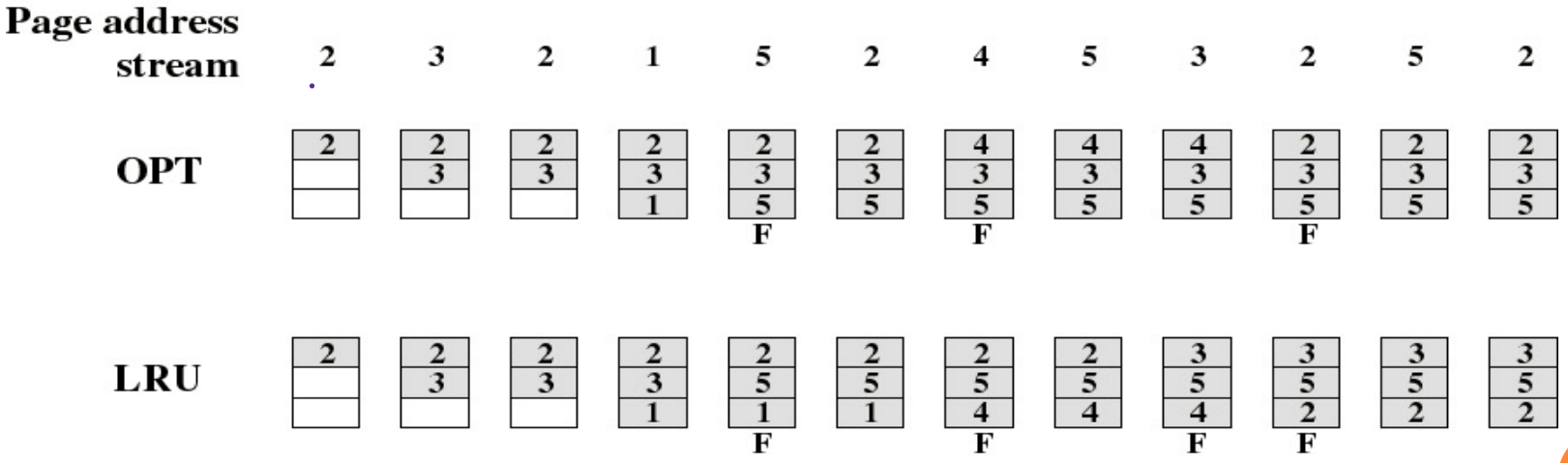
- Since the page reference stream is not known *ahead of time*, we can't "pre-fetch" pages.
- We know which page to load *during run-time* so we load them on "**demand**".
- Result: concentrate on **replacement** policies.

Replacement Algorithms

- Optimal policy
  - selects for replacement the page for which the time to the next reference is the longest
  - produces the fewest number of page faults
  - impossible to implement (need to know the future) but serves as a standard to compare with the other algorithms we shall study
- Least recently used (LRU)
- First-in, first-out (FIFO)
- Clock

LRU Policy

- Replaces the page that has not been referenced for the longest time
  - by the principle of locality, this page is least likely to be referenced in the near future
  - performs nearly as well as the optimal policy
- Example: A process of 5 pages with an OS that fixes the resident set size to 3

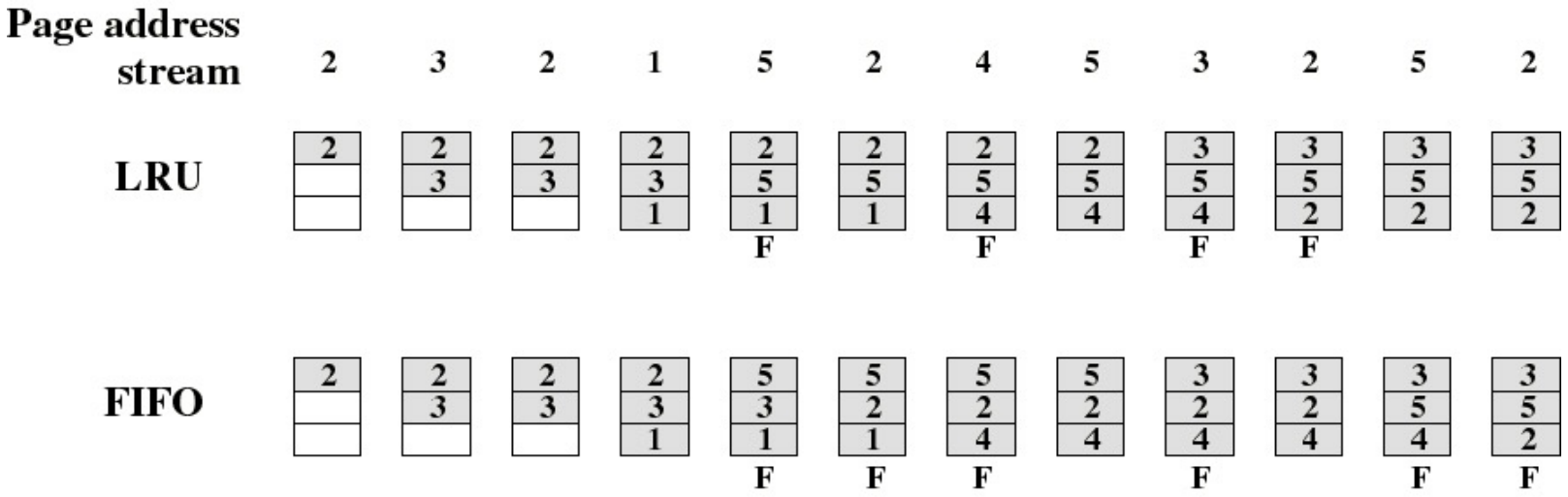


FIFO Policy

- Treats page frames allocated to a process as a circular buffer; pages removed in round-robin style
- When the buffer is full, the page that has been in memory the longest is replaced. Hence, first-in, first-out
  - A page fetched into memory a long time ago may now have fallen out of use
  - But a frequently used page is often the oldest, so it will be repeatedly paged out by FIFO
- Simple to implement
  - requires only a pointer that circles through the page frames of the process

Comparison of FIFO with LRU

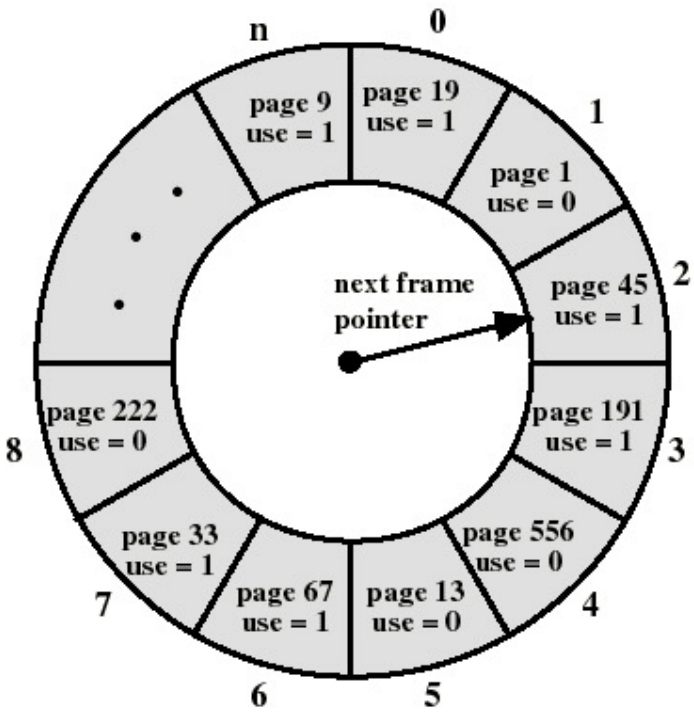
- LRU recognizes that pages 2 and 5 are referenced more frequently than others but FIFO does not
- FIFO performs relatively poorly



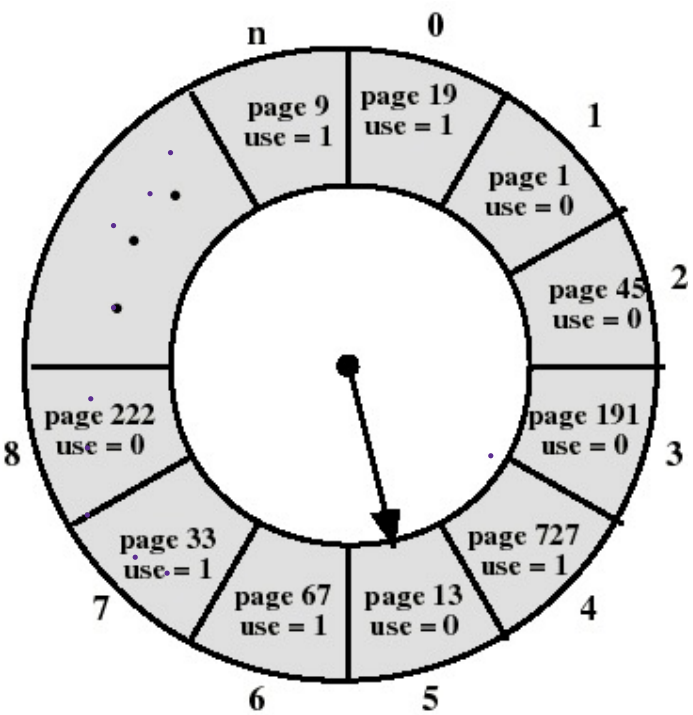
Clock Policy

- The set of frames that are candidates for replacement (local or global scope) is considered as a circular buffer
- When a page is replaced, a pointer is set to point to the next frame in buffer
- A use bit for each frame is set to 1 whenever
  - a page is first loaded into the frame
  - the page is referenced
- When it is time to replace a page, the first frame encountered with the use bit set to 0 is replaced
  - during the search for replacement, each use bit set to 1 is changed to 0

Clock Policy: An Example

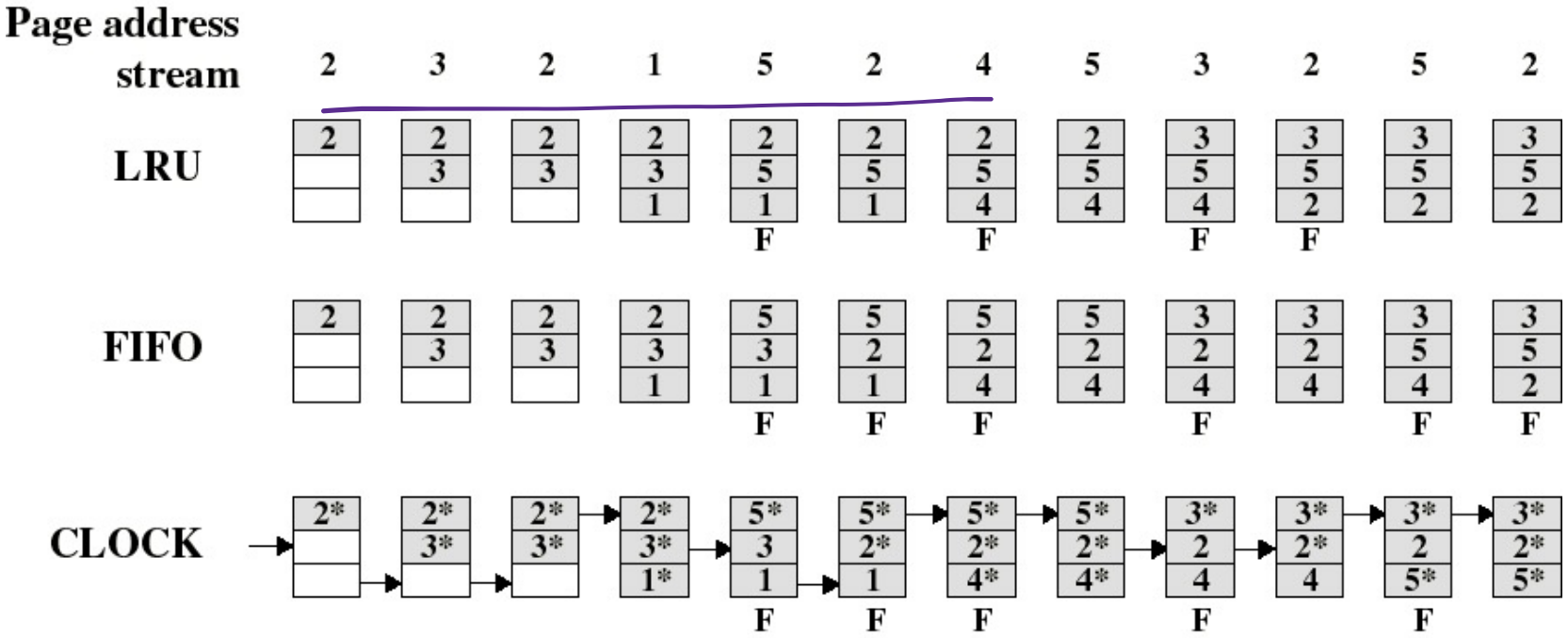


(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

Comparison of Clock with FIFO and LRU



- Asterisk indicates that the corresponding use bit is set to 1
- Clock protects frequently referenced pages by setting the use bit to 1 at each reference

Clock policy: Comments

- If all frames have a use bit of 1, then the pointer will make one full circle setting all use bits to 0. It stops at the starting position because it can now replace the page in that frame.
- Similar to FIFO, except frames with use bit set to 1 are passed over
- Numerical experiments tend to show that performance of Clock is close to that of LRU
- Many OS'es use variations of the clock algorithm. Example: Solaris

Problem solving 1

- Show the memory representation using the LRU algorithms and calculate the page fault rate.
- Consider the reference string R={0, 1, 0,1, 0, 1, 2, 3, 4, 5, 3, 4, 5, 6, 7, 8, 9} and an allocation of 3 frames for a given process.

SOLUTION

(i) Show the memory representation using the LRU algorithms and calculate the page fault rate.

Frame	0	1	0	1	0	1	2	3	4	5	3	4	5	6	7	8	9
0	***0***	0	0	0	0	0	0	***3***	3	3	3	3	3	***6***	6	6	***9***
1		***1***	1	1	1	1	1	1	***4***	4	4	4	4	4	***7***	7	7
2							***2***	2	2	***5***	5	5	5	5	5	***8***	8

- *Page fault rate is 10/17 = 58.82%*