Managing Virtual Storage

Read Text chapter 10

Different Strategies

Demand and Anticipatory Paging

The Concept of Locality

Page Sizes

Page Replacement Strategies

Program Behavior With Paging

Thrashing

Management Strategies

1. Fetch Strategies

When should a page be brought from secondary to main memory -- on demand only or should it be anticipated ?

2. Placement Strategies

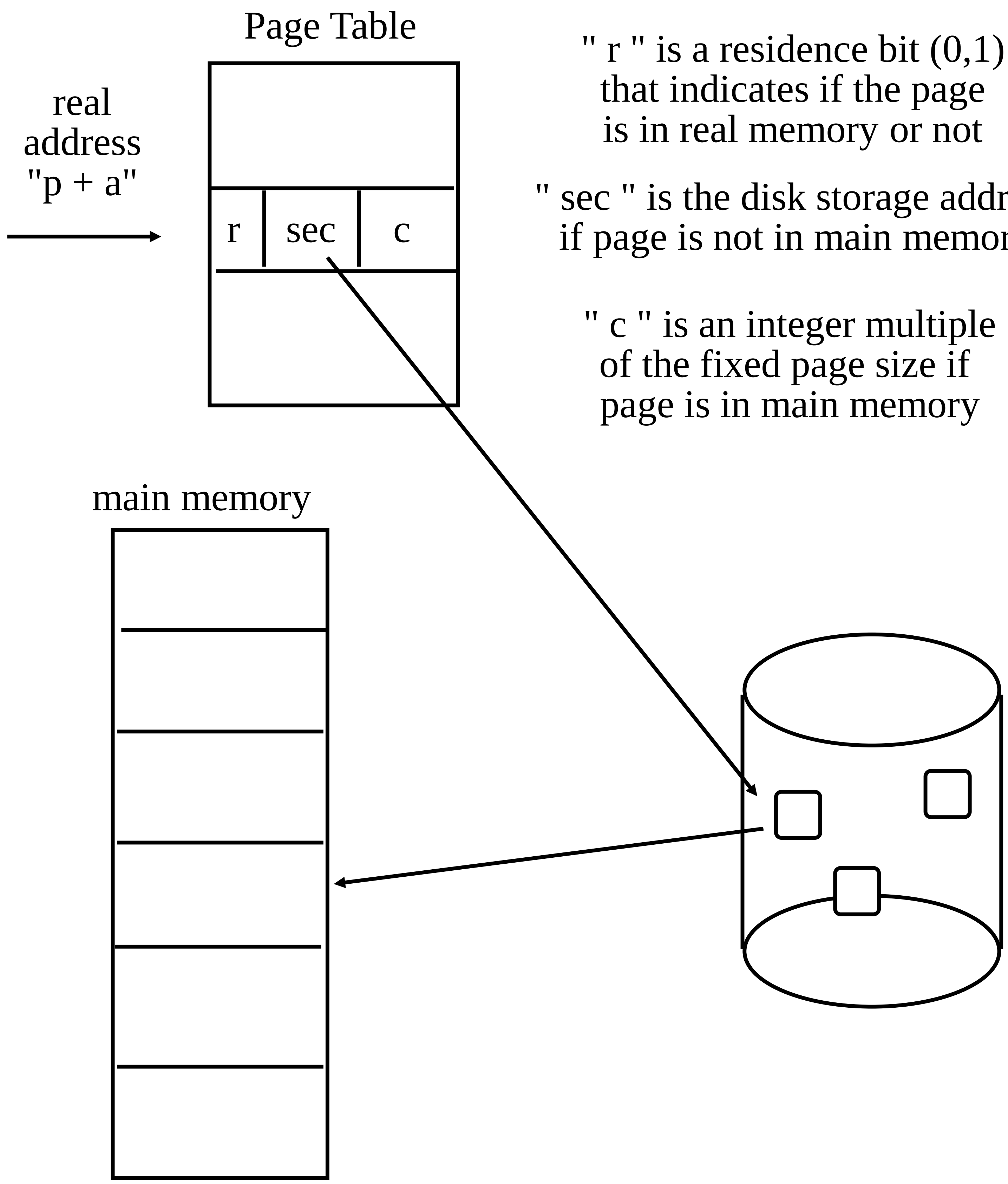
Where should a page be placed in main memory

3. Replacement Strategies

Which page should be displaced when a new one is brought in from secondary storage

Demand Paging

Bring a page from disk into physical memory only when that page is referenced by a running process. A page fault brings the operating system into action



" r " is a residence bit (0,1) that indicates if the page is in real memory or not

" sec " is the disk storage address if page is not in main memory

" c " is an integer multiple of the fixed page size if page is in main memory

Time Spent Loading a Page

EXAMPLE FROM TEXTBOOK – PAGE 398

Notice great discrepancy in speed between

- **Bringing a page from secondary storage – 8 msecs**
- **Accessing main memory location – 200 nanoseconds**

$8X10^{-3}$ vs $200X10^{-9}$ = 40,000 slower

To keep slowdown due to paging to 10% we need to keep page faults to less than 1 per 400,000 memory accesses.

Anticipatory Paging

While a process runs with its present pages, the OS predicts which new pages will be needed by the process in the future and loads those pages into free page frames in main memory

The process will run faster since when it needs the pages they will already be in main memory

For this strategy to work we need to:

Develop accurate prediction mechanisms (spatial locality)

Not pay a high price if a bad prediction is made

Isn't the time required to bring the page into main memory the same now as it would be later on ?

Then, how is it that the process runs faster with anticipatory paging ?

Advantages and Disadvantages of Demand Paging

Advantages

The only pages brought into main memory are those required by the running processes

No overhead associated with predicting which page will be used next

Disadvantages

In terms of resources being tied up, it is more "expensive" to wait for a page transfer when a greater amount of main memory is occupied since other processes are being kept from using that memory space

Locality

Processes tend to reference storage in nonuniform, highly localized patterns

Temporal Locality

Storage locations referenced recently are likely to be referenced in the ***near future***

Reasons for this:

looping
subroutines
variables used for counting and totaling

Spatial Locality

Once a storage location is referenced, it is likely that ***nearby locations*** would be referenced

Reasons for this:

array traversals
sequential code execution
related variables are usually defined near each other

Working Set

It is the collection of pages a process is actively referencing.

Working sets are transient for a a given process. The next working set of a process will include different pages than the present one.

For a program to run efficiently its working set must be maintained in main memory. If not, its page fault rate will be high and will slowdown execution of that process considerable.

Example: Assume memory access time of 100 nanoseconds and page fault rate, p

effective access time = $(1 - p) * 100 * 10^{-9} + p * 25 * 10^{-3}$

effective access time = 100 + 25000000 p in nanoseconds

if we get one page fault out of every 1000 instructions, p = 0.001

effective access time = 100 + 25000 nanoseconds

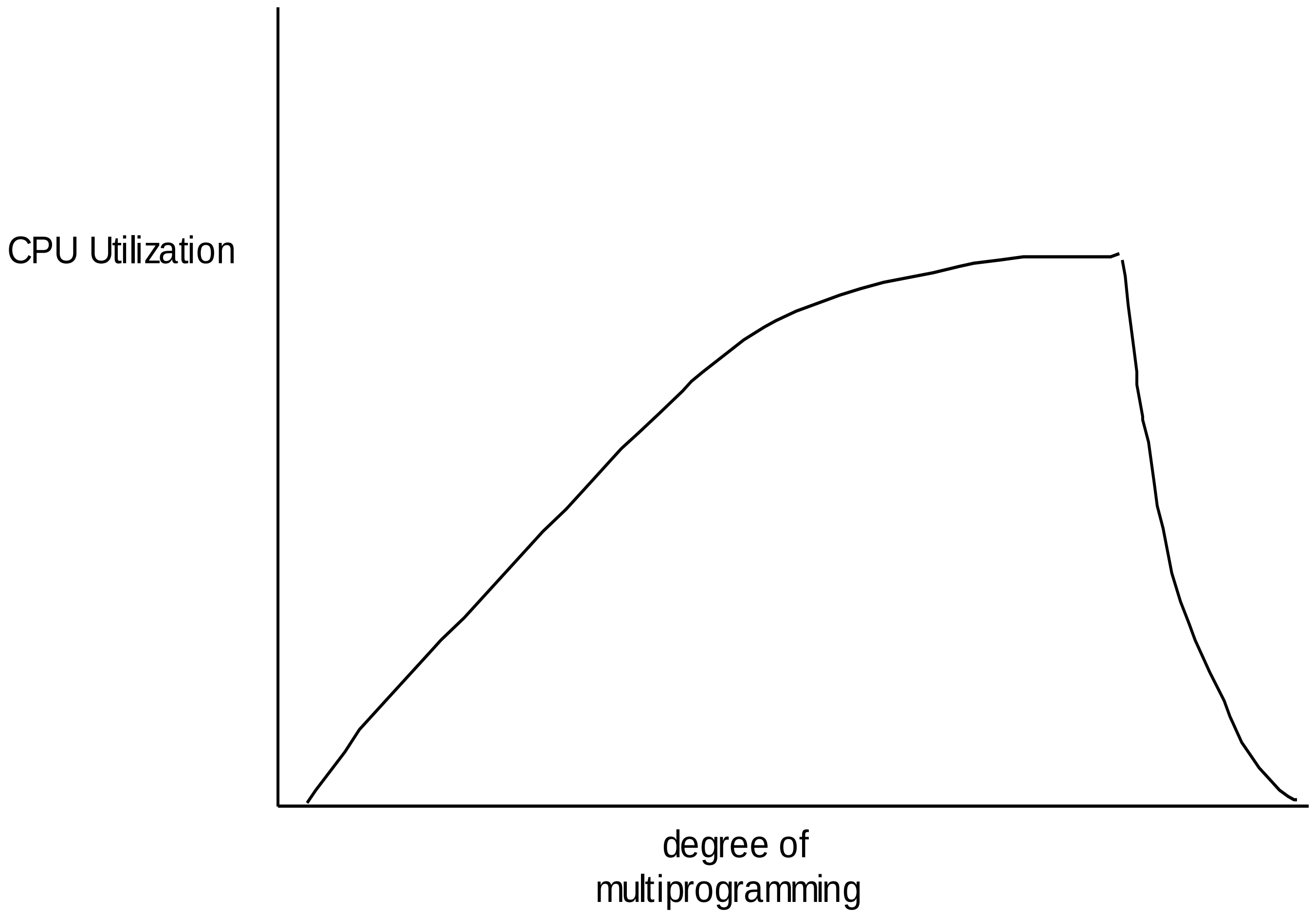
This is 250 times slower than if no paging were being used and would be unacceptable execution speed.

If we want less than 10% degradation is speed then we need

25000000p = 10 or

p = 1 page fault for every 2,500,000 instructions

Thrashing



Thrashing occurs when CPU utilization decreases significantly du
to increase number of page faults.

1. As the # of processes ↑ the # of pages/process ↓

   As long as the # of pages/process is large enough to hold the working set then we actually increase CPU utilization without page faulting

2. As the # of processes ↑↑ then CPU utilization ↓↓

   Because the # of pages/process goes below the number required to hold the working set. Each process generates a page fault every time with almost every instruction

3. As CPU utilization ↓↓ scheduler ↑ # of processes and then CPU utilization ↓↓

Small versus Large Page sizes

Larger page sizes are better because:

Processes page tables will be smaller thus saving main memory dedicated to them

Less number of transfers from secondary memory thus process should be able to run faster

Smaller page sizes are better because:

Smaller amounts of code are brought into main memory thus less amount of code that ultimately may not be referenced

Less main memory fragmentation

Since program locality is small, small pages will contain code that will be referenced more often - thus less allocated total memory

Page Replacement Strategies

If all pages in physical memory are occupied, and a page fault occurs, which page should be displaced from physical memory to make room for the new page ?

Obviously the one that will not be referenced for a long time !

Unfortunately we don't know that information

The following strategies all have advantages and disadvantages

Random Replacement

FIFO

Least Recently Used

Least Frequently Used

Not Used Recently

Optimal Replacement

Choose the page to be replaced in physical memory that will not be referenced again for the longest time.

The information to make this choice is obviously not available, but we can use it here to compare the performance of other strategies

Random Replacement

Choose the page to be replaced in physical memory at random

The advantage here is low overhead in the selection process

The disadvantage is unpredictability - you may select the page that will be referenced in the next instruction

Seldom used

FIFO

Choose the page that has been in physical memory the longest

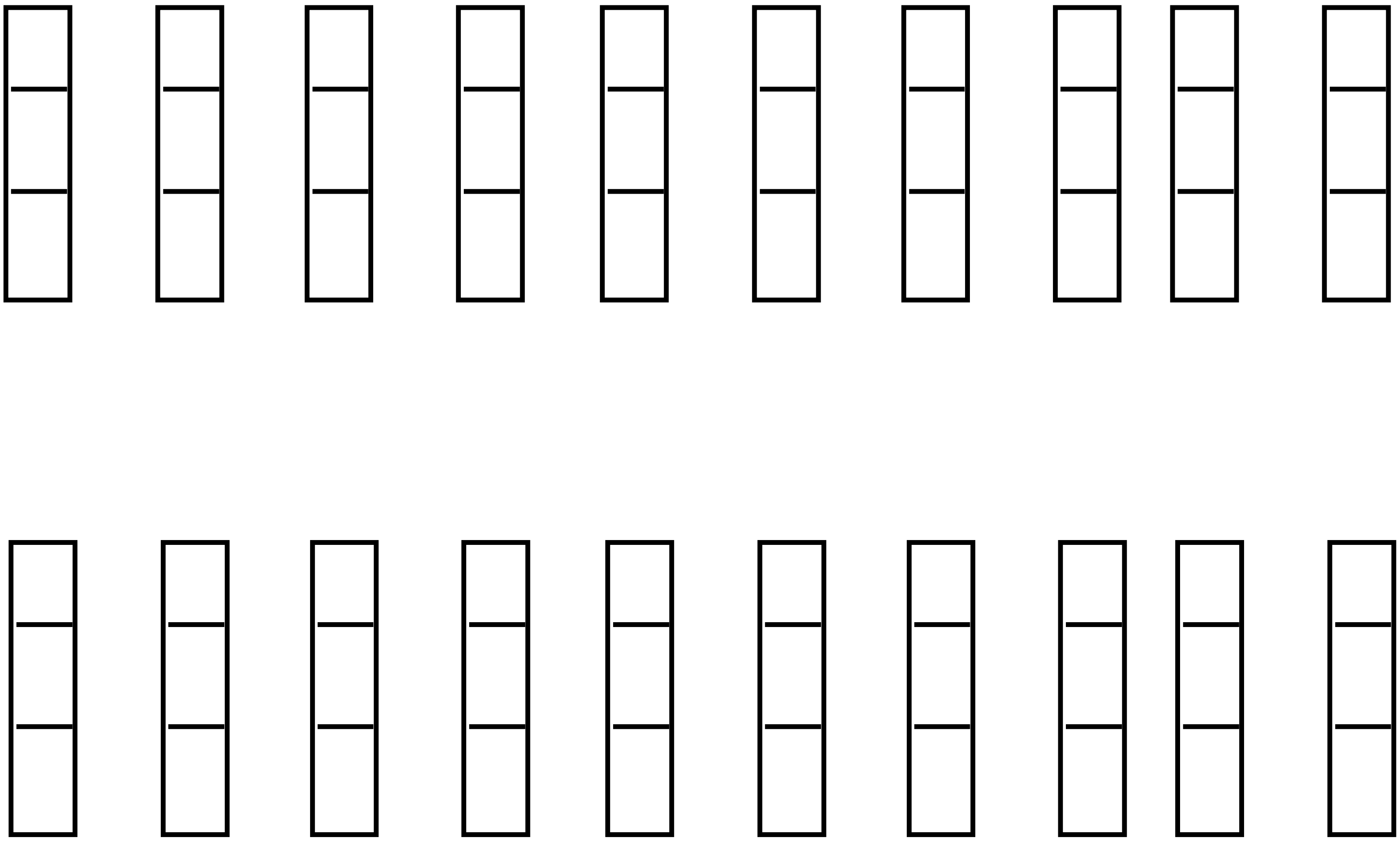
The advantage is ease of implementation through a FIFO queue

The disadvantage is that you may replace a heavily used page

Example

Assume that we have a system with only 3 page frames in main memory and that we have only one process running.
The following string of page references is made by the process as it executes. Determine how the pages will be loaded into main memory if an optimal replacement strategy is used. Also determine the number of page faults

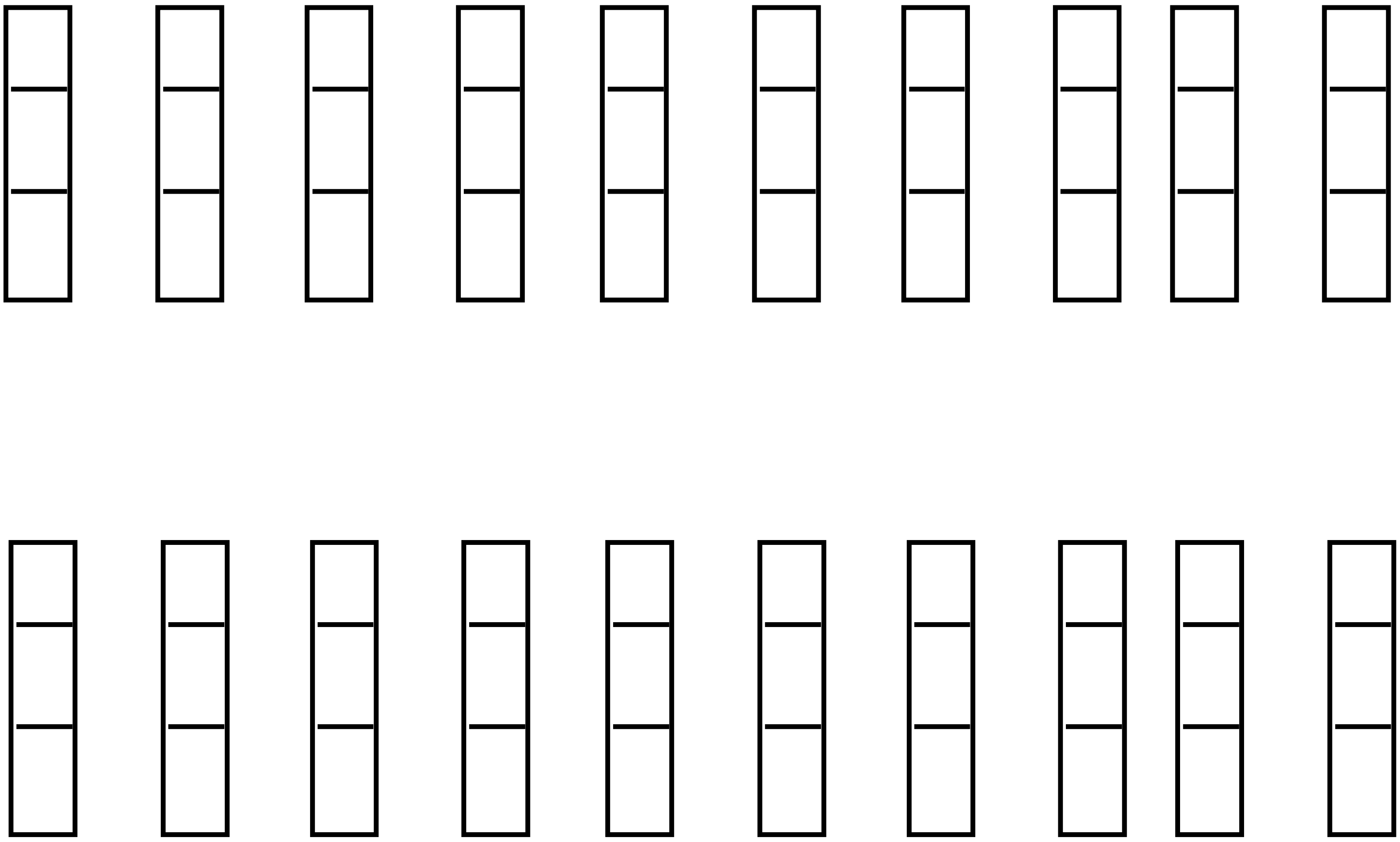
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



Example

For the same example as before, determine how the pages will be loaded into main memory if a FIFO replacement strategy is used. Also determine the number of page faults

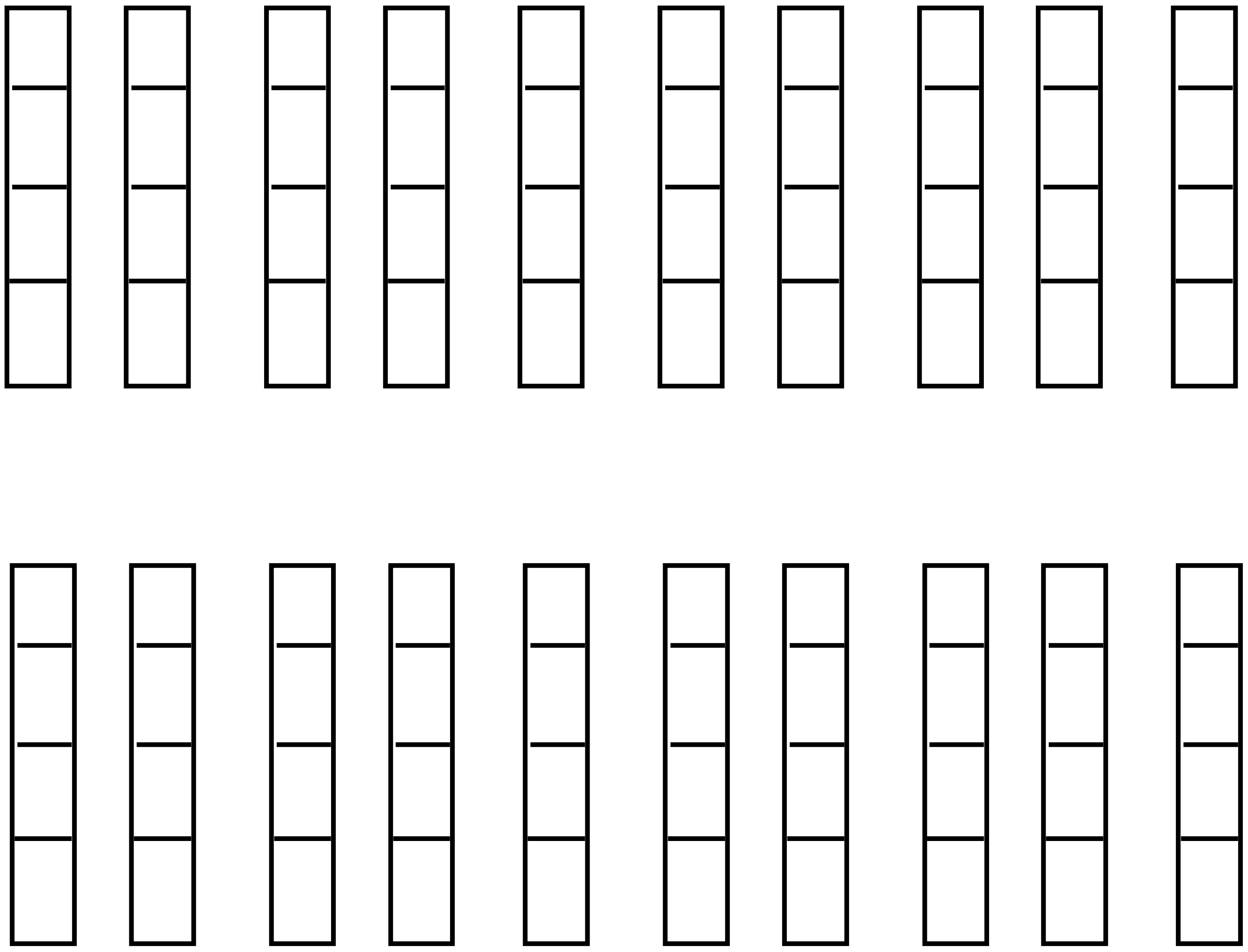
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



Example

Same problem as in previous example but assume that we have 4 page frames available in our system

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



Sometimes the number of page faults increases as the number of page frames increases. This is known as Belady's Anomaly

Least Recently Used

Choose the page from physical memory that has not been used
for the longest time. This may be a good indication that it will not
be used again for the longest time

This tries to implement the optimal replacement strategy

By optimal we mean the one had will result in the minimum number
of page faults

LRU requires that each page be time stamped when it is used.
Large overhead can be experienced with this approach
A software implementation using a stack can effectively time
stamp pages as they are being used

Least Frequently Used

Choose the page from physical memory that has been referenced
the least amount.

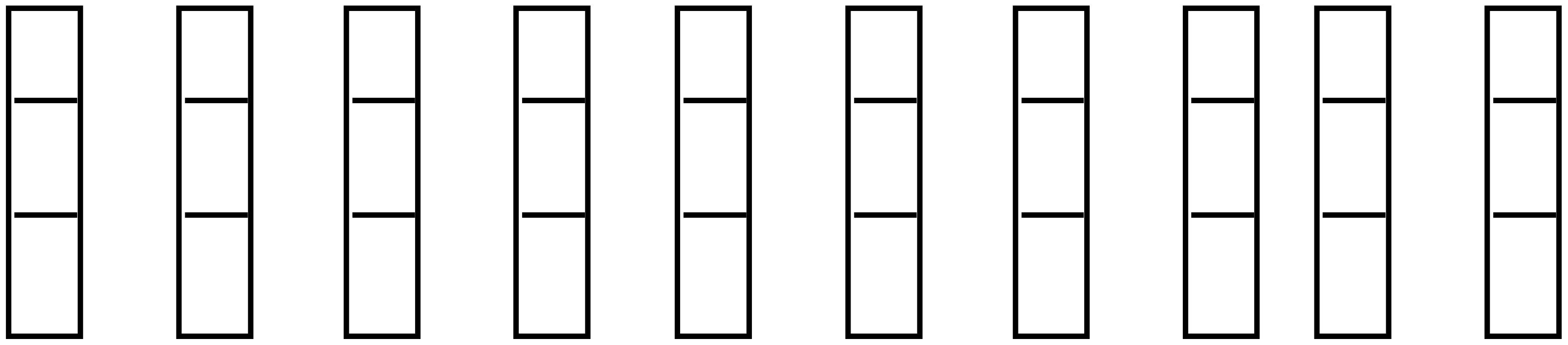
This is used as an approximation to the LRU strategy

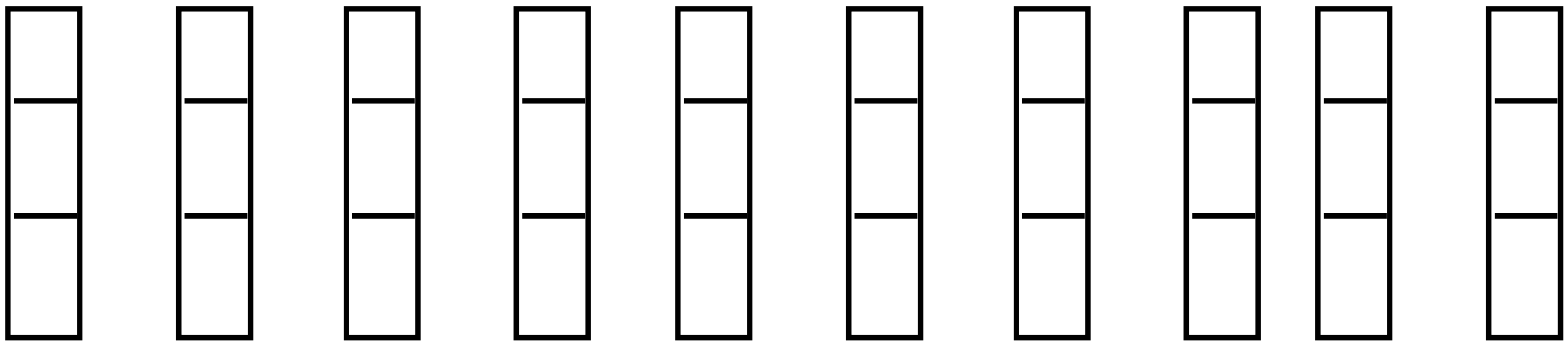
A counter is kept for each page and incremented each time the page
is referenced

Example

Assume that we have a system with only 3 page frames in main memory and that we have only one process running.
The following string of page references is made by the process as it executes. Determine how the pages will be loaded into main memory if a LRU replacement strategy is used. Also determine the number of page faults

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1





Not Used Recently

Choose the page from physical memory that has recently been used the least. This may be a good indication that it will not be used again for the longest time

NUR is implemented using two hardware bits associated with each page

referenced bit = 1 if page has been referenced
= 0 if not

modified bit = 1 if page has been modified
= 0 if not

Periodically all the reference bits are set to 0 so only those pages referenced recently will have their reference bits set to 1

Pages fall into four classes

(0,0) neither used nor modified

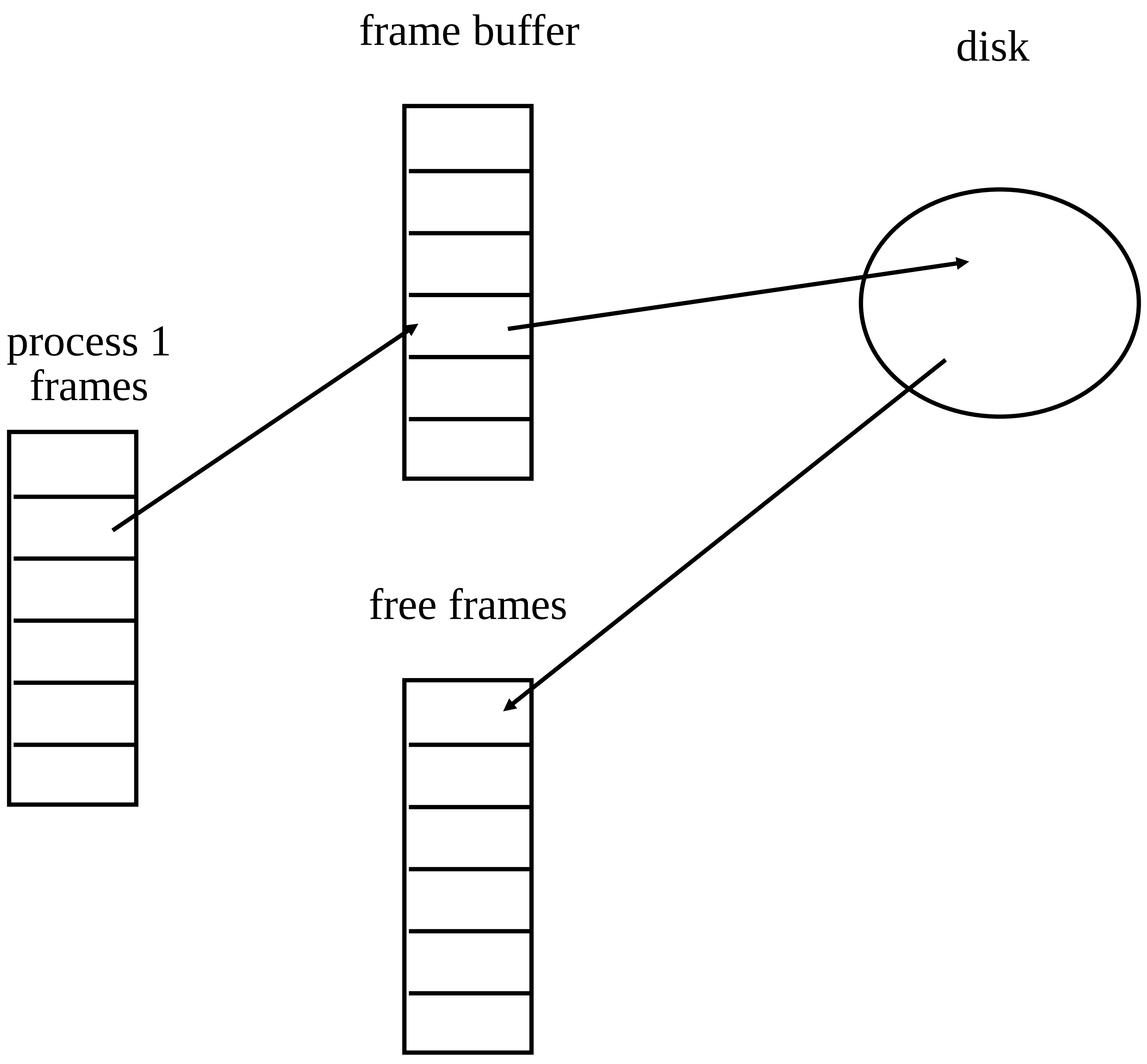
(0,1) not used recently but modified

(1,0) used but not modified

(1,1) used recently and modified

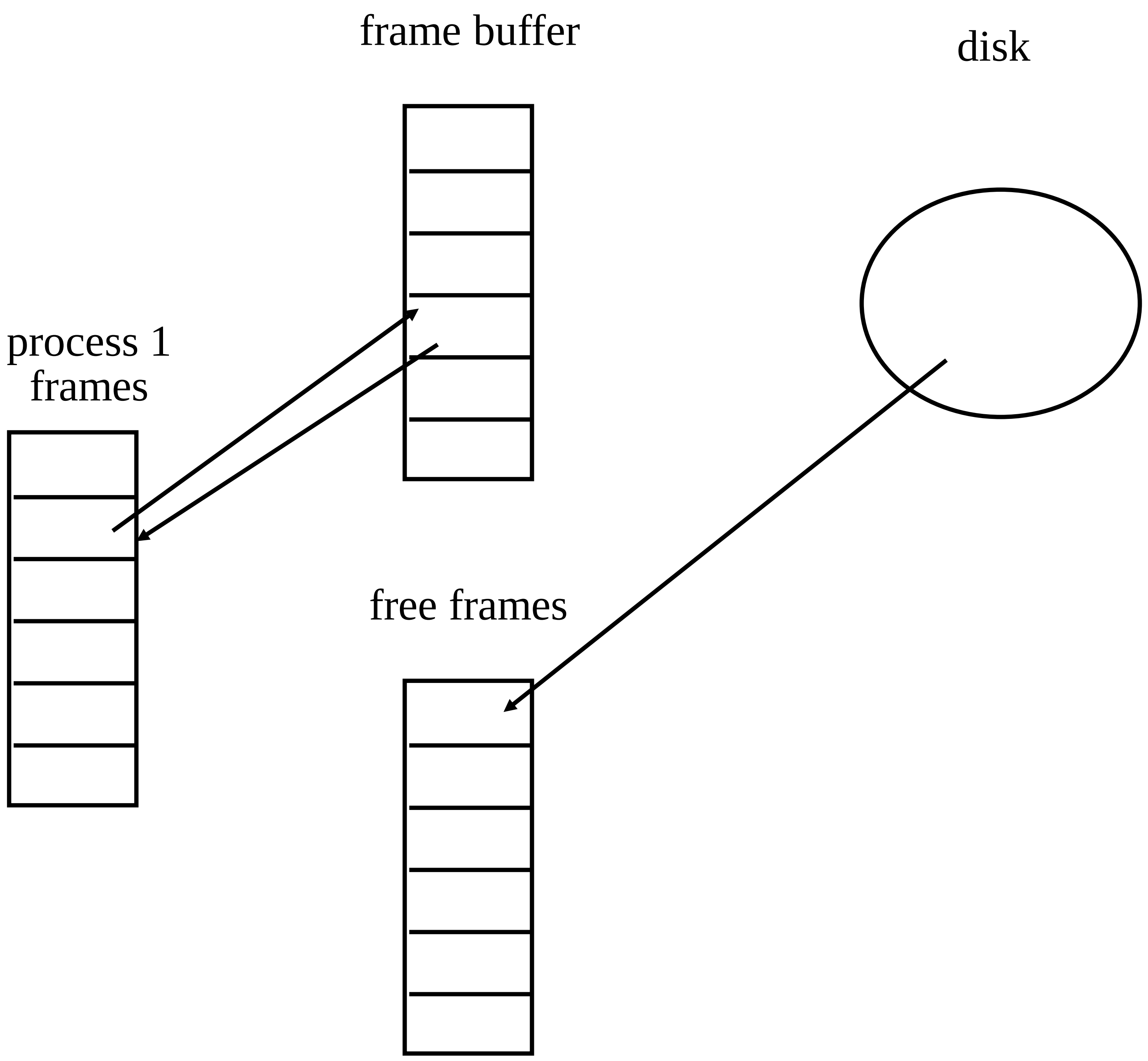
Pages in the top classes [(0,0), (0,1) etc] should be selected first to be replaced. Within a class, random or FIFO selection strategies can be used.

Page Buffering



When a page fault occurs and a new page must be brought in from disk and replace one in main memory, the new page is brought in first into a free frame and the process is ready to run. The old frame, if it had been modified, is written to disk later.

Page Buffering



When the page replaced is needed again right after its replacement, it can brought back from the frame buffer and avoid a disk read. For this the OS needs to remember in which buffer frame the replaced page ended up.

Original Frame Allocation

Minimum number of frames

There is a minimum number of frames that a process needs

There must be enough frames to hold all the different pages that a single instruction can reference

an instruction may be two words long. In that case the first word may be in the last position of one page and the next word of that instruction in the first position of the next page - thus requiring a minimum of two frames

the instruction may reference a memory address that is in another page - thus we need another frame for that. And if indirect addressing is allowed, another frame would be needed for that

Thus, the minimum number of frames depends on the architecture of the computer.

Original Frame Allocation

Equal Allocation

Divide the # of frames by the # of processes and save some frames for a free frame buffer

Proportional Allocation

Allocate frames to each process in proportion to the size of the process.

For example if there are 90 frames available and process 1 is 50 pages long and process 2 is 100 pages long then allocate 30 frames to process 1 and 60 frames to process 2

Proportional Priority Allocation

Allocate frames to each process in proportion to the size and priority of the process. Processes with higher priorities should receive a larger number of frames

Replacement Frame Allocation

Global Replacement

After a page fault, if no free frames are available, obtain a frame from any process in the system

Thus the page fault frequency of a process can depend on the paging behavior of other processes

Local Replacement

After a page fault, if no free frames are available, obtain a frame from the process creating the page fault

Global replacement is preferred because of higher overall throughput