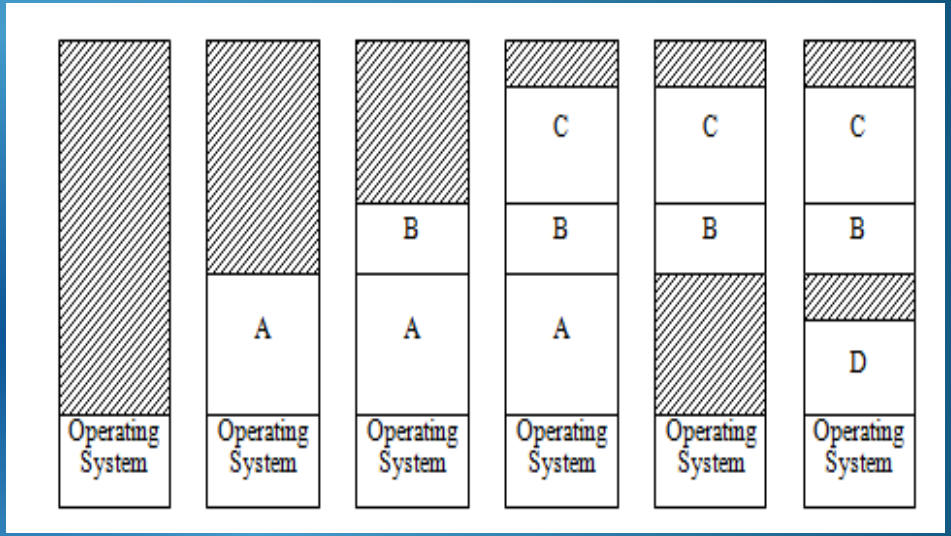
CS240 Operating Systems, Communications and Concurrency

Recall Dynamic Partitioning

Problem of assigning a linear address space area when loading processes.

A similar problem is management of the heap area within a process at run time.



CS240 Operating Systems, Communications and Concurrency

RECALL

A process address space can be divided into three main areas.

Text – Code of the program occupies this area and is generally static.

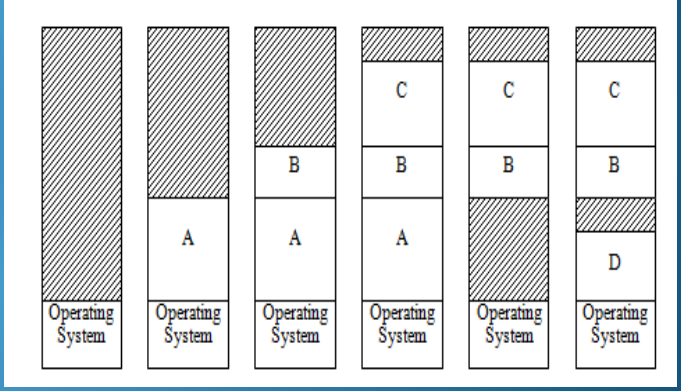
Stack – A stack is generally a fixed sized region of memory which a process uses to create stack frames for holding parameters and local variables and return linkage addresses when calling functions. Frames are always added to and removed from the top of the stack in a LIFO manner which makes allocation and deallocation faster. Each thread must have its own stack but shares a heap.

Heap – A heap is a variable sized region of memory which is used for creating dynamic data structures and objects. Variables created on the heap will still exist after function calls return, unlike stack based variables. Management of the heap's linear space is more complex and takes longer.

CS240 Operating Systems, Communications and Concurrency

Dynamic Partitioning

Implementation of basic operations required to manage a linear address space



Allocation - means memory manager finds and assigns a free space large enough to accommodate the memory request.

Deallocation - means memory manager reclaims the currently allocated memory space back to the free area.

CS240 Operating Systems, Communications and Concurrency

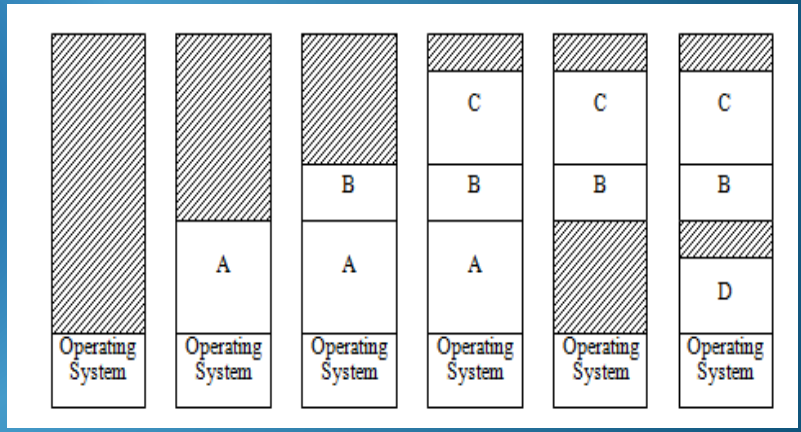
Managing a Linear Memory Space– Key Ideas

Allocation

How to keep track of what parts of the memory space are free?

We look at three approaches:-

- Bit Maps
- Linked List
- Buddy System

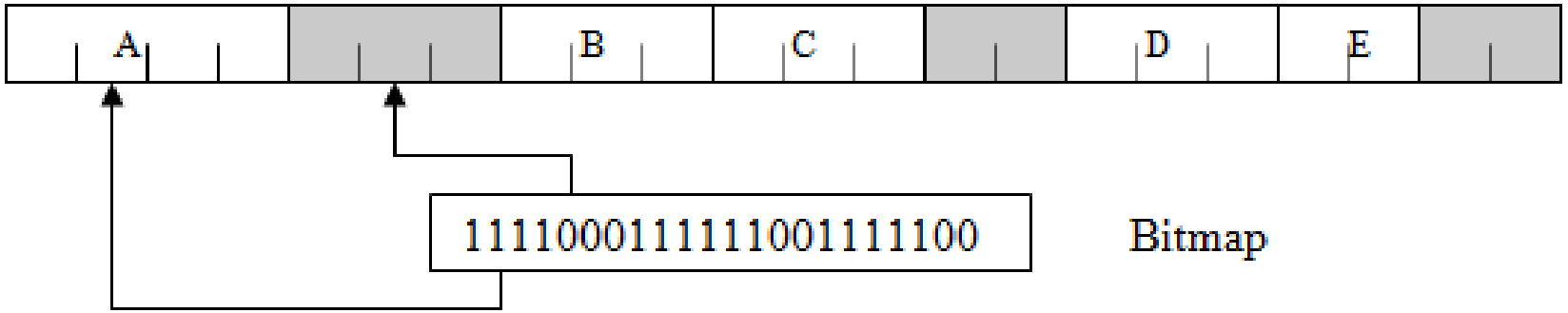


CS240 Operating Systems, Communications and Concurrency

Method 1 - Bit Maps – What are they?

Divide the memory space into small fixed sized allocation units, say `1,024 bytes (1K)` each.

The bitmap contains a single bit for each allocation unit which indicates the status of that unit, e.g. `1` for used, `0` for unused.



CS240 Operating Systems, Communications and Concurrency

Bit Maps – Overhead of structure

The size of memory and the chosen size of allocation units determine the size of the bitmap.

Example

If a memory space was 4Gbytes in size and allocated in 1024 byte allocation units, then there would be 4M allocation units and so the bit map would need to be 512Kbytes in size.

There would be an average of half an allocated unit of internal fragmentation per process, 512 bytes.

CS240 Operating Systems, Communications and Concurrency

Bit Maps – Large Allocation Units

The size of the allocation unit can be an important design consideration.

If the allocation unit is too large, the bit map will be smaller, but we will only be able to allocate memory space in multiples of large allocation units, so it is likely that more of the space within the last allocated unit might not be required by a process.

CS240 Operating Systems, Communications and Concurrency

Bit Maps – Small Allocation Units

The size of the allocation unit can be an important design consideration.

If the memory is large and the allocation units are relatively small, then the bit map will be a bigger data structure and searching the bitmap string may be time consuming.

CS240 Operating Systems, Communications and Concurrency

Bit Maps – Efficiency of expected operations

Allocation

Involves searching for a string of bits which represents a region large enough to accommodate the request.

If a `0` value is used for empty and a `1` value indicates a used allocation unit, then finding a free area means finding a block of zero bytes in the bit map. Some processors have a special instruction for assisting with this.

CS240 Operating Systems, Communications and Concurrency

Bit Maps – Efficiency of expected operations

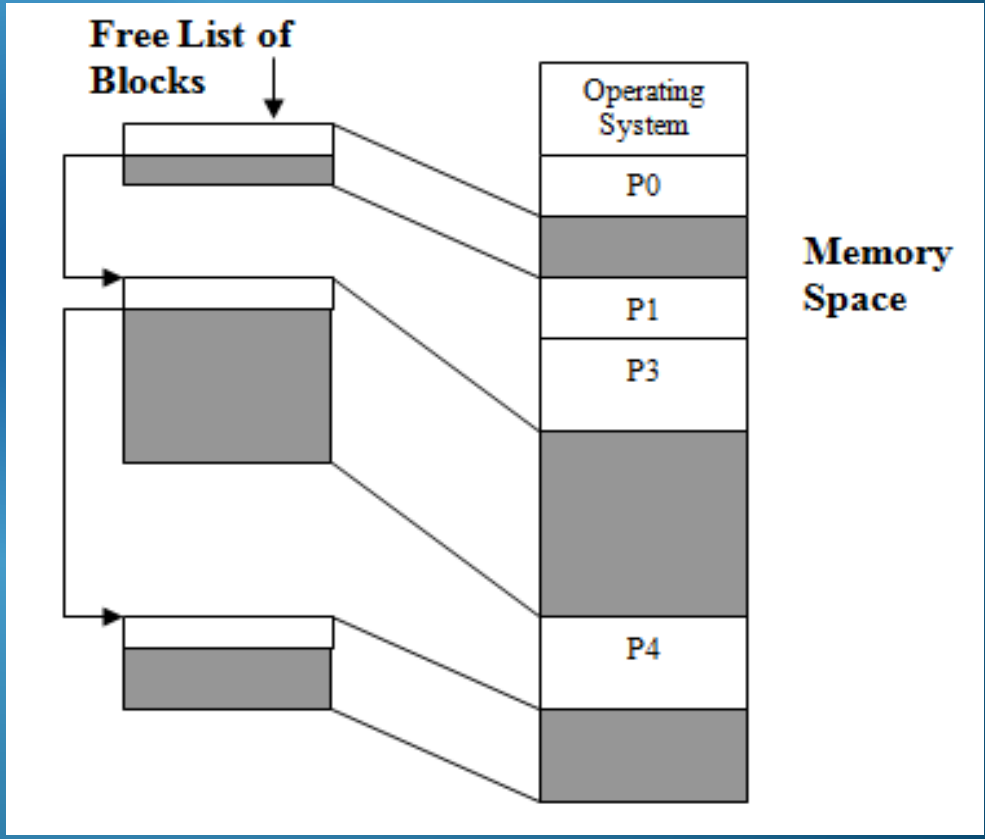
Deallocation

Memory manager simply resets the appropriate bits in the bitmap corresponding to the allocation units being released.

CS240 Operating Systems, Communications and Concurrency

Method 2 - Linked Lists – Tracking Free Memory Space

Maintain an information record describing the size and location of each of the variable sized free blocks and to keep this variable number of records in a dynamic linked list structure.



CS240 Operating Systems, Communications and Concurrency

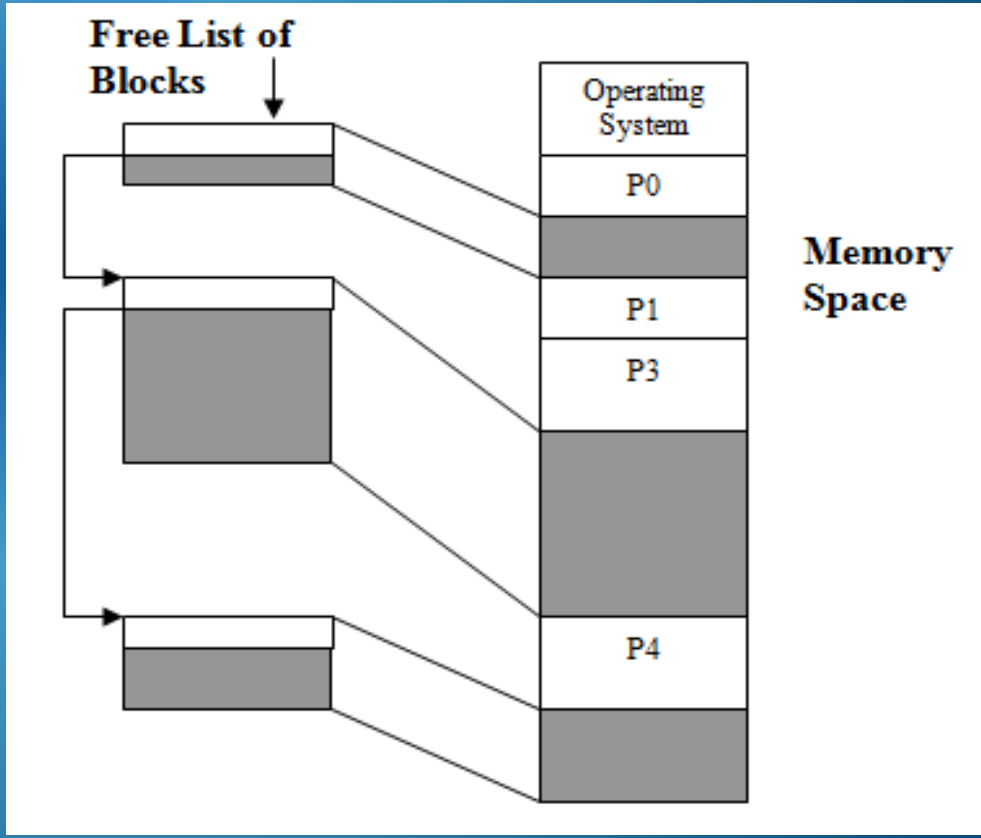
Linked Lists –

Allocation

Memory manager must search the list to find a free block of suitable size for the request.

Deallocation

A new free block record is added to the free list, describing the starting location and size of the released block.

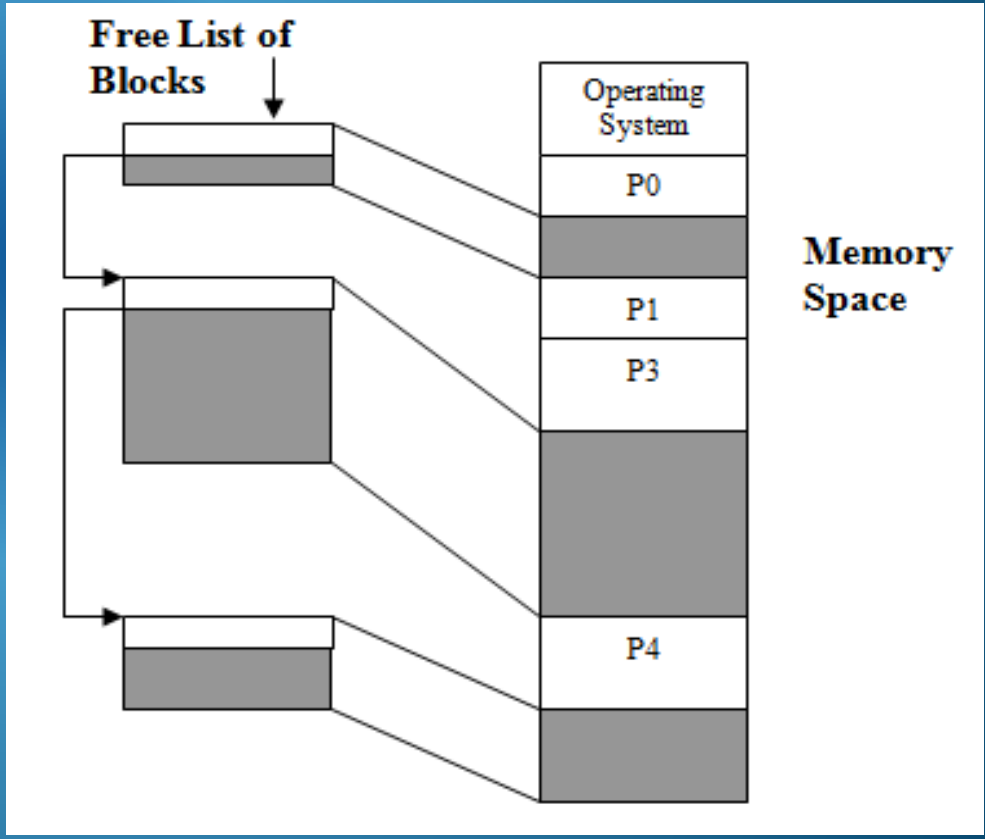


CS240 Operating Systems, Communications and Concurrency

Linked Lists – Overheads

Where to store the linked list?

The records of the free list can be stored within the first few bytes of the unused memory regions which they describe, so that the memory manager simply needs to maintain a pointer to the first record of the list.

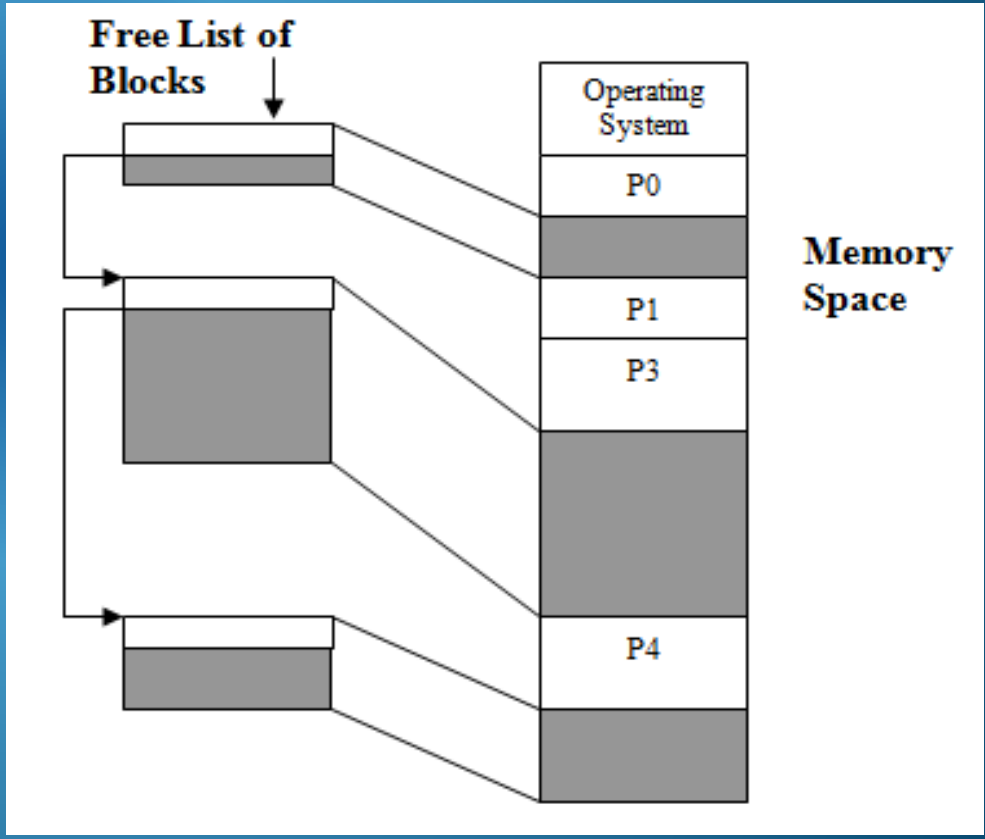


CS240 Operating Systems, Communications and Concurrency

Linked List Search and Allocation Strategies – Placement Algorithms

A number of different strategies for searching the free list can be considered by the memory manager when allocating space to a process.

The goal is to minimise the need for compaction.



CS240 Operating Systems, Communications and Concurrency

Best Fit: Place process in the smallest accommodating block. If the block is slightly too big it leaves only a small empty region.

Worst Fit: The memory manager places the process in the largest free block dividing it to leave a smaller but still useful free block.

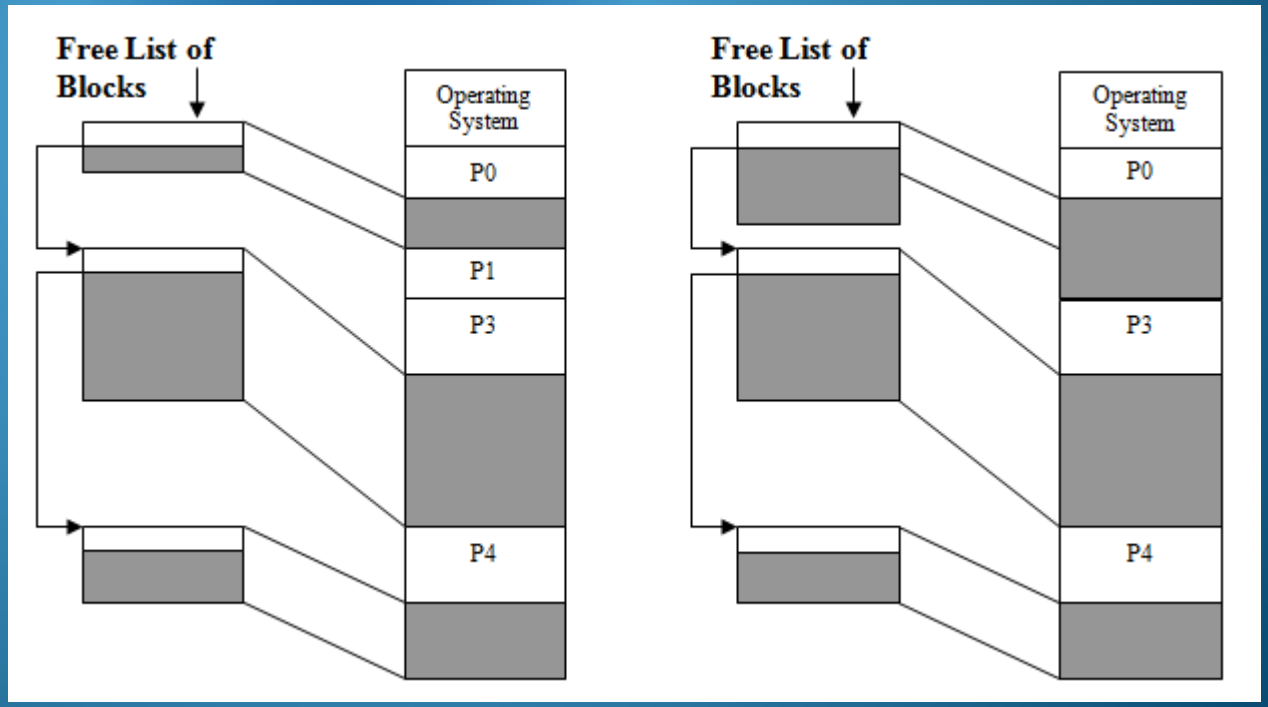
First Fit: The memory manager assigns the process the first block which can accommodate it in its scan of the free list. This saves time analysing the list.

Next Fit: The memory manager remembers the place in the list where it last allocated a free block, and its next search for a free block applies a first fit approach but begins with the record subsequent to the last used

CS240 Operating Systems, Communications and Concurrency

NB: Deallocating a used memory region is a little more complicated than with the bitmap approach.

The memory manager needs to analyse the free list to see if there are any free regions adjacent to the deallocated block so that they can be merged into a single larger block on the list.



CS240 Operating Systems, Communications and Concurrency

Bit Map

Fixed Sized Data Structure regardless of memory usage

Allocation efficiency depends on size of bitmap, byte search

Deallocation very quick

½ page internal fragmentation per process

Linked List

Dynamic Size, depends on number of free segments

Allocation involves possibly long list search if many small segments

Deallocation requires complete list analysis to determine neighbour segment mergers

No internal fragmentation

CS240 Operating Systems, Communications and Concurrency

Method 3 - Buddy System

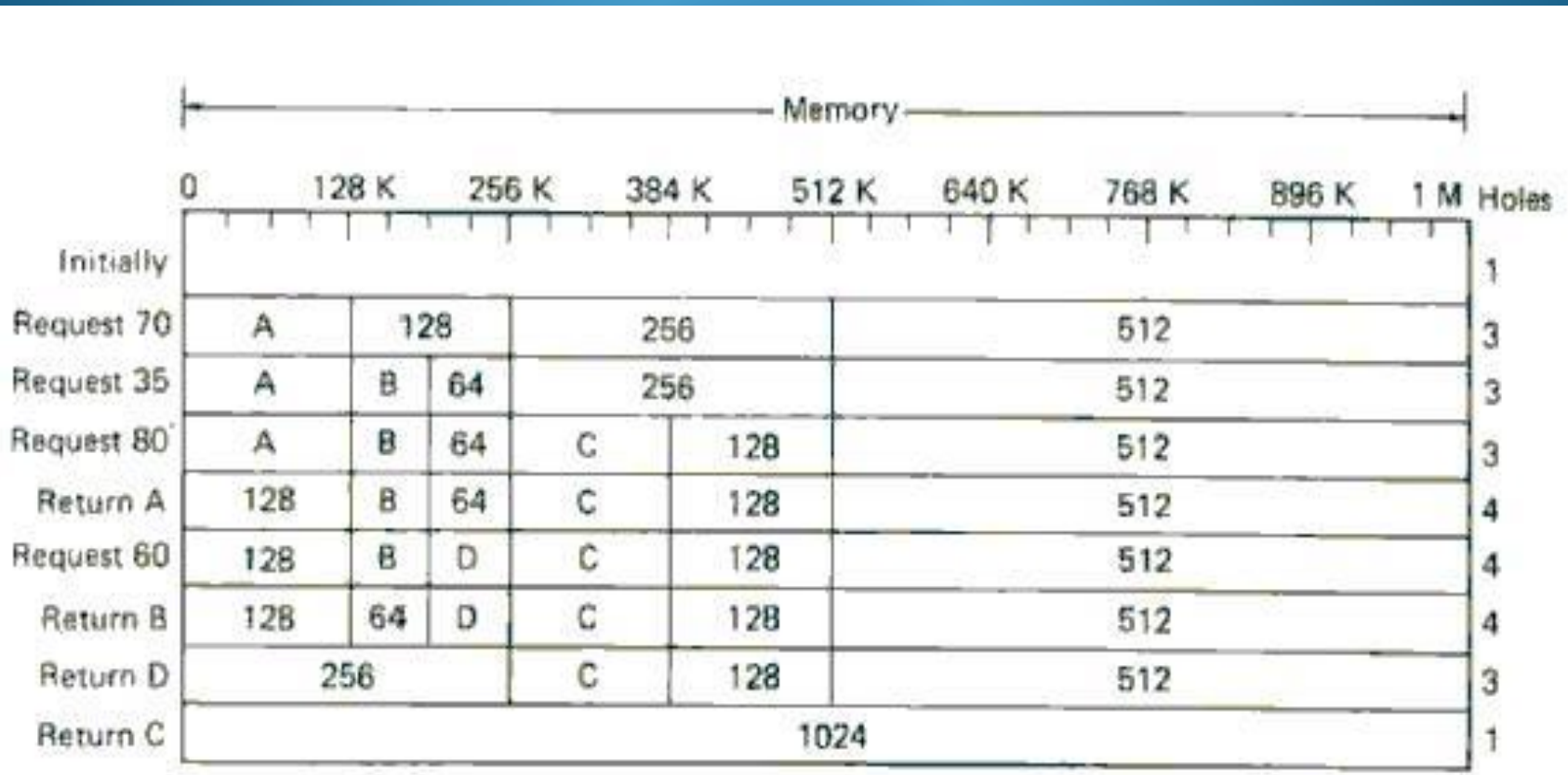
The Buddy System is a compromise system allowing dynamically sized partitions but with restrictions on the sizing of those partitions to make allocation and deallocation simpler.

In the buddy system, lists are maintained of free memory blocks that are powers of 2 bytes in size. Memory allocation requests are mapped to blocks which are the closest power of 2 in size.

For example, a 1Mbyte memory would need 21 lists. The first list would contain free blocks of size 1 byte, the next list free blocks of size 2 bytes, the next list free blocks of size 4 bytes and so on up to the 21st list which would contain free blocks of size 1Mbyte.

CS240 Operating Systems, Communications and Concurrency

Buddy System



CS240 Operating Systems, Communications and Concurrency

Method 4 - Paged Architecture

Need more hardware support

Partitioning with a Base and Limit Register is restrictive as process allocated address space is confined to one contiguous segment.

Memory Partitioning fragments memory into smaller free areas, hard to find one big area to accommodate large contiguous segment without incurring compaction and relocation overhead.

Paged Architectures overcome these problems but require additional hardware support, a mapping mechanism, in place of the Base/Limit register idea.

CS240 Operating Systems, Communications and Concurrency

Paged Architecture

With a paged architecture memory is divided into a number of relatively small (perhaps 512–8Kbytes) fixed sized units known as pages.

A process address space is made up of a number of pages. The physical pages may be allocated to a process from any of the free pages within the memory space.

Key Idea

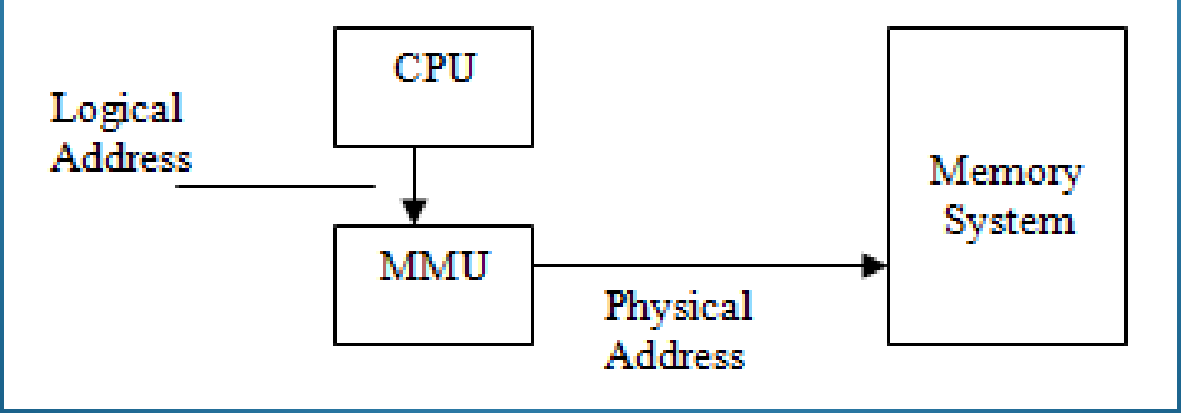
The physical pages do not need to be contiguous (adjacent to one another) – Need a Mapping Mechanism.

CS240 Operating Systems, Communications and Concurrency

Paged Architecture supports logical to physical mapping

The process has a logical view of its memory space as consisting of a number of logical page numbers from 0 to a maximum.

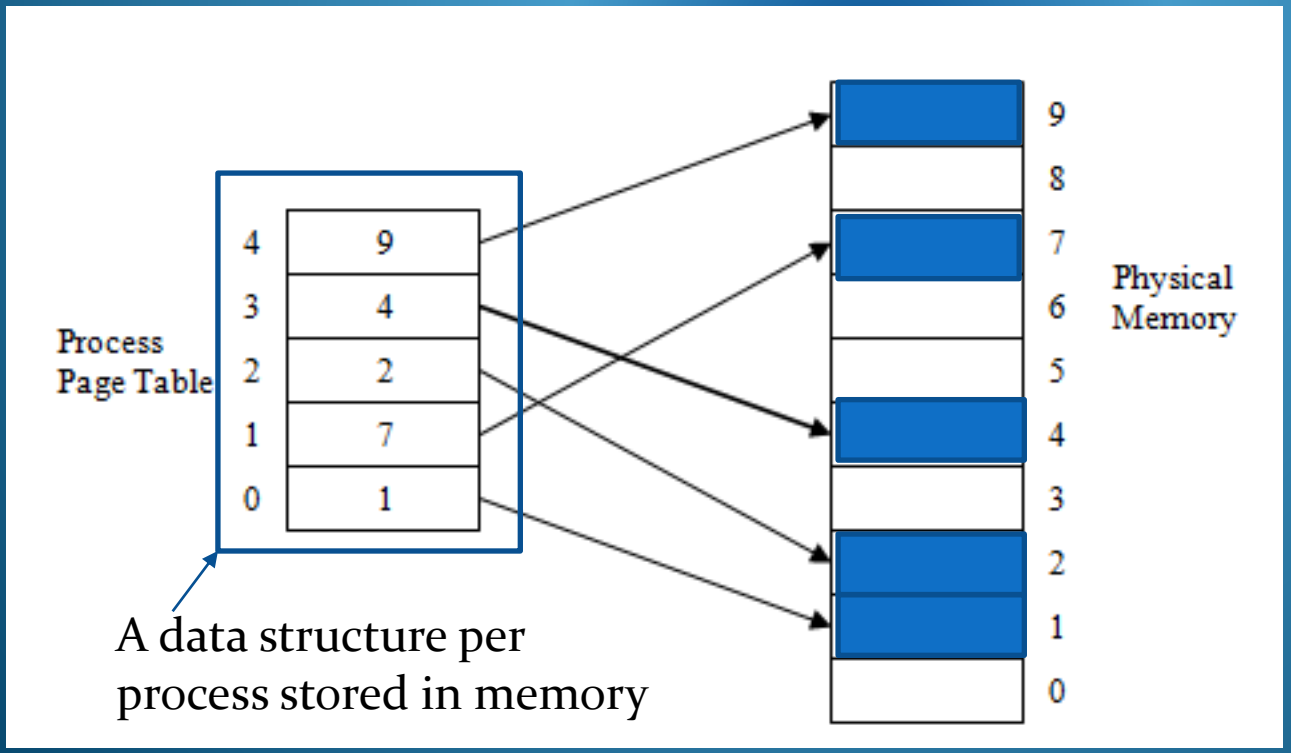
These pages are mapped by the memory management unit of the processor to the corresponding physical pages in the memory.



CS240 Operating Systems, Communications and Concurrency

Paged Architecture

The mapping information for each process is kept in a page table. There is one page table per process. How big is this data structure?

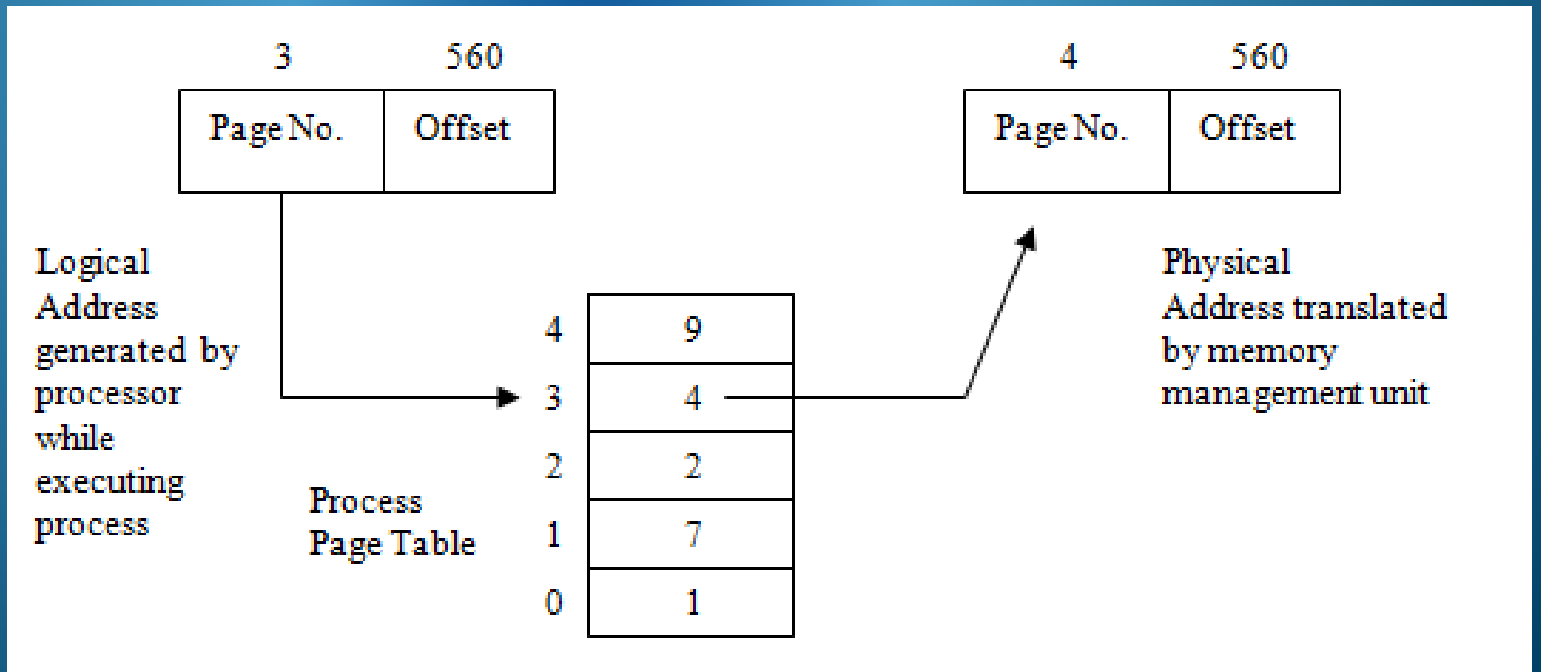


CS240 Operating Systems, Communications and Concurrency

Paged Architecture

A memory address is divided into two parts, a page number and an offset within that page. The MMU exchanges the page number from the logical address generated by the processor, with the corresponding physical page found in the page table.

Say pages are 1000 bytes in size



CS240 Operating Systems, Communications and Concurrency

Paged Architecture - Advantages

Protection - As processes only have a logical view of their address spaces, it is not possible for them to access the address space of other processes because the page table won't contain pointers to pages for any other process.

The memory space assigned to a process can be composed of pages which are scattered randomly throughout the physical memory and do not all have to be adjacent to one another. Easy to keep track of free space and to allocate space of requested size.

It is easy to relocate or expand the address space of a process by altering the mapping of pages into its page table.

CS240 Operating Systems, Communications and Concurrency

Paged Architecture - Disadvantages

Page tables can be large and need to be kept in main memory causing an additional overhead.

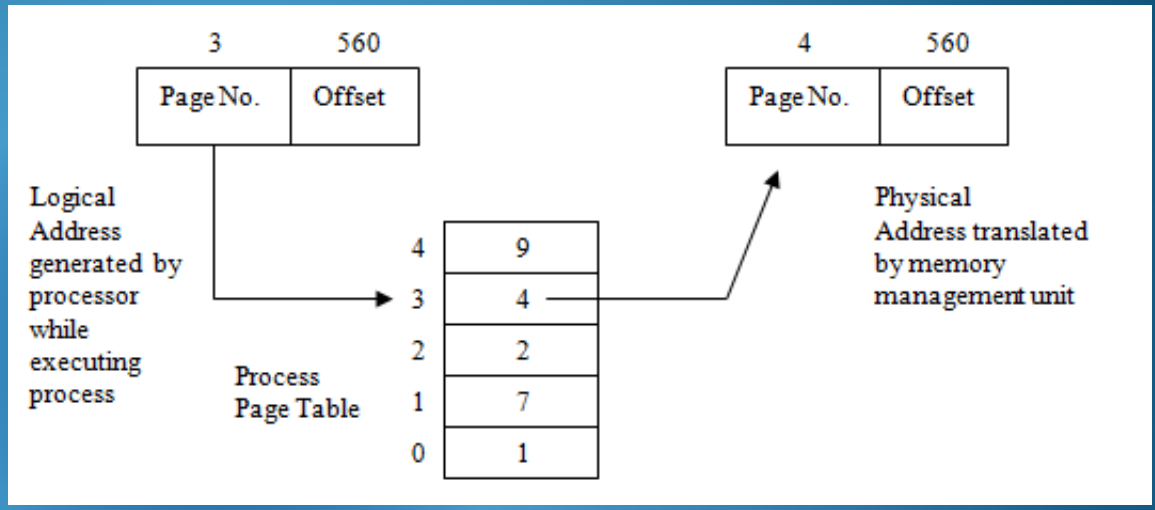
For every logical memory access, two real memory accesses may be needed unless part of the page table can be cached in a translation lookaside buffer by the processor. Page mapping still slows down the access time to memory slightly.

As memory space is allocated to processes in multiples of pages, a process may not require all of the memory space of the last page assigned to it. On average approximately ½ a page per process would be wasted. This phenomenon is known as internal fragmentation.

CS240 Operating Systems, Communications and Concurrency

Demand Paging

To achieve good system performance and device utilisation need to have a good mix of processes resident in memory.



If each process address space is large, we may not have enough real memory space for many.

Observation that process execution behaviour exhibits locality of reference. It might not be necessary to keep the entire process address space in electrical memory at once.

CS240 Operating Systems, Communications and Concurrency

Demand Paging

Key Ideas

Keep resident only portions of each process address space corresponding to the current locality of reference.

Allows us to increase the degree of multiprogramming as less pages are allocated to each process.

Allows process virtual address space to exceed the capacity of real electrical memory.

CS240 Operating Systems, Communications and Concurrency

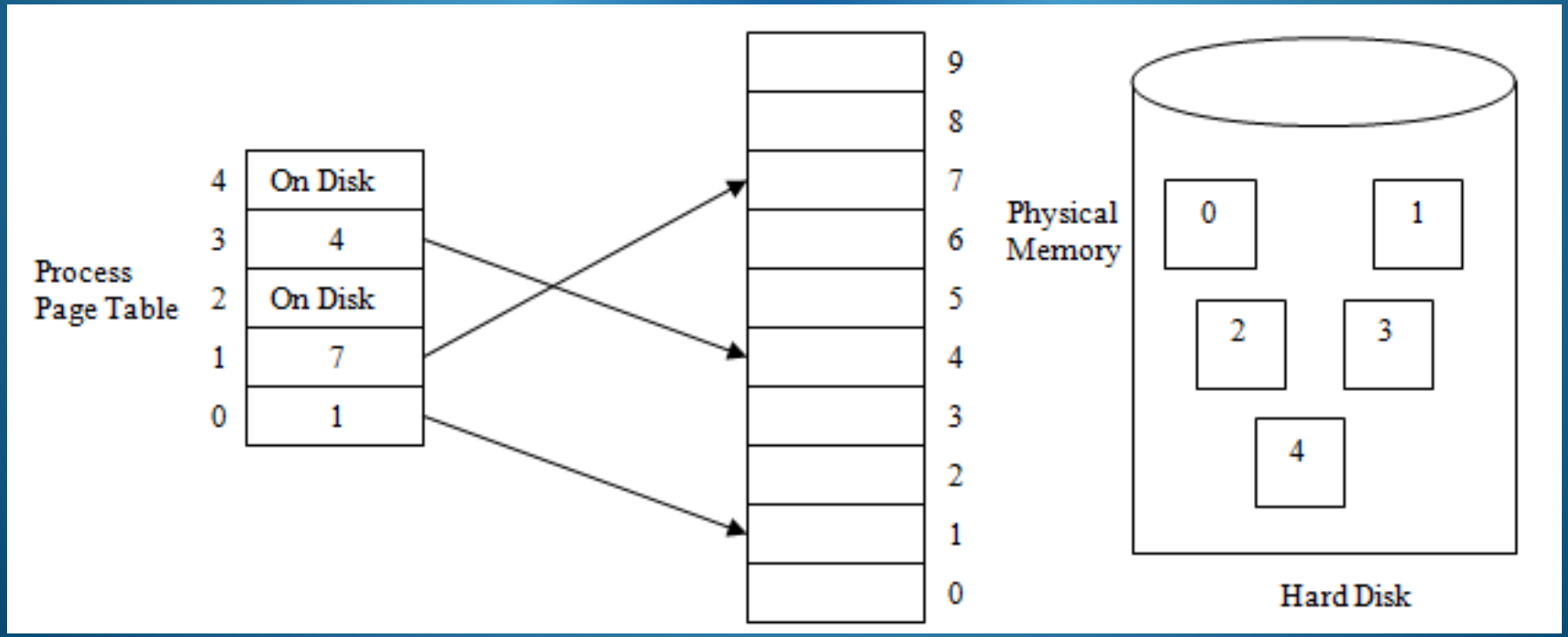
Demand Paging

In order to implement this idea, an area of the hard disk, a swap file, is used by the memory manager to store the address space (all of the logical pages) of each active process in the system.

The memory manager then brings a certain number of these pages into physical memory at a time, the working set, corresponding to the current locality of each process.

CS240 Operating Systems, Communications and Concurrency

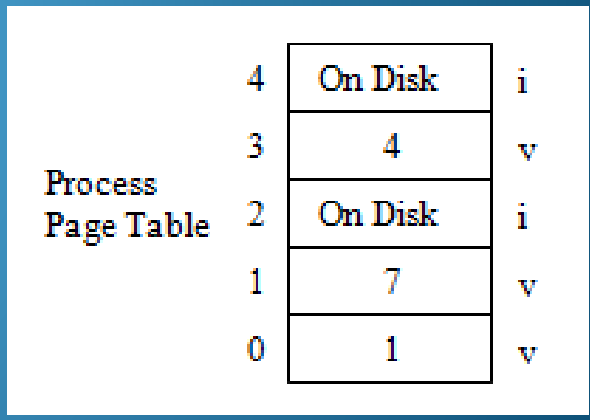
Demand Paged Virtual Memory System Concept



CS240 Operating Systems, Communications and Concurrency

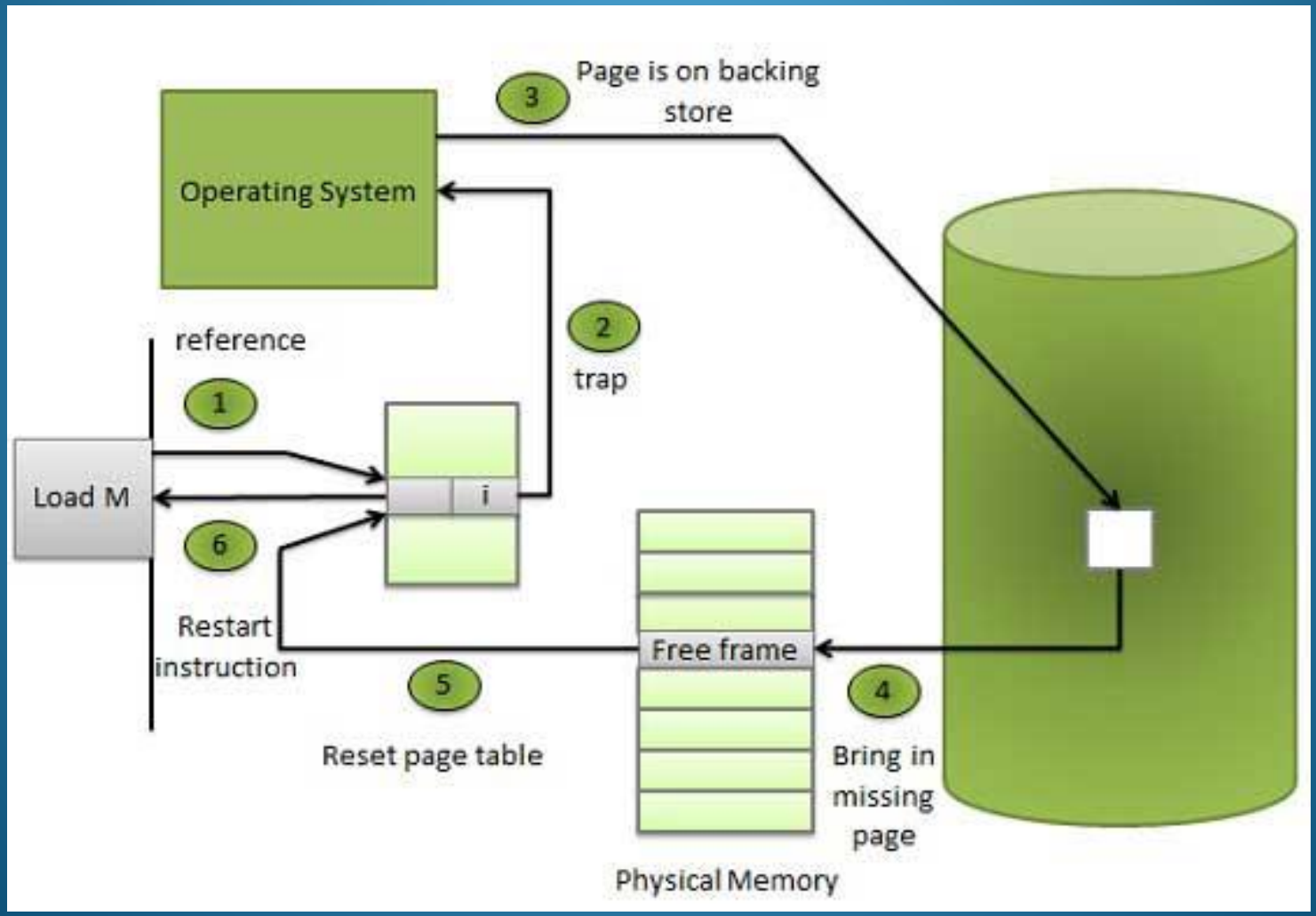
Demand Paging – Need Modifications to Page Table

A single bit field is associated with each page entry, known as a valid/invalid bit. When a reference to an address within a page is made, the MMU consults the relevant page table entry and checks to see if the bit is set to valid or invalid (0 or 1).



If valid, then the entry in the page table is used to access the appropriate physical memory page otherwise the memory management unit triggers an event known as a page fault.

Servicing a Page Fault



CS240 Operating Systems, Communications and Concurrency

Working Set

Each process must be allocated a number of physical memory pages in which the memory manager attempts to store pages from its logical address space corresponding to its current locality of reference. The set of physical pages allocated to a process is known as its working set.

The upper size limit of the working set is determined by the maximum number of physical memory pages available. The lower limit is determined by processor architectural features.

CS240 Operating Systems, Communications and Concurrency

Page Replacement Algorithms

Sometimes, there may be no free pages left in memory to accommodate an incoming page, or perhaps the operating system restricts the size of the working set for a process, in order to be fair to other processes in the allocation of physical memory space.

Both of these scenarios mean that a page currently in memory needs to be swapped out to disk to make room for an incoming page. The page that is selected for replacement is chosen using a page replacement algorithm.

CS240 Operating Systems, Communications and Concurrency

Efficiency Concerns of Page Replacement

It is important not to swap out a page that is likely to be needed again shortly by the process because each page fault results in a disk operation which is significantly slower than access to main memory.

A lot of page faulting activity has the effect of reducing the overall memory access time significantly as well as making the disk system busy.

Sometimes, if the working set is too small, pages of the process will be continually exchanged between memory and disk resulting in poor execution performance. This phenomenon is known as thrashing and can generally be eliminated by increasing the size of the working set.

CS240 Operating Systems, Communications and Concurrency

Efficiency Concerns of Page Replacement

Say that an access to real memory takes 100nsec .

Say that to read or write a page to disk takes 10msec.

Say that the page fault rate is 1%, i.e. 1 out of every 100 memory references is to a page not currently in memory.

Say that the page chosen for replacement is dirty 50% of the time.

What is the effective access time to the memory system?

$0.99 \times 100 \times 10^{-9} + 0.01 \times (0.5 \times (20 \times 10^{-3} + 10 \times 10^{-3}))$

$= 0.000000099 + 0.00015$

$= 0.000150099$ seconds , 1500 times slower

CS240 Operating Systems, Communications and Concurrency

Page Replacement Algorithms

The **Optimal Page Replacement** algorithm, which guarantees a minimum number of page faults is to replace the page which won't be used for the longest time.

Unfortunately this would require future knowledge of which pages a process was going to access and cannot be implemented in practice.

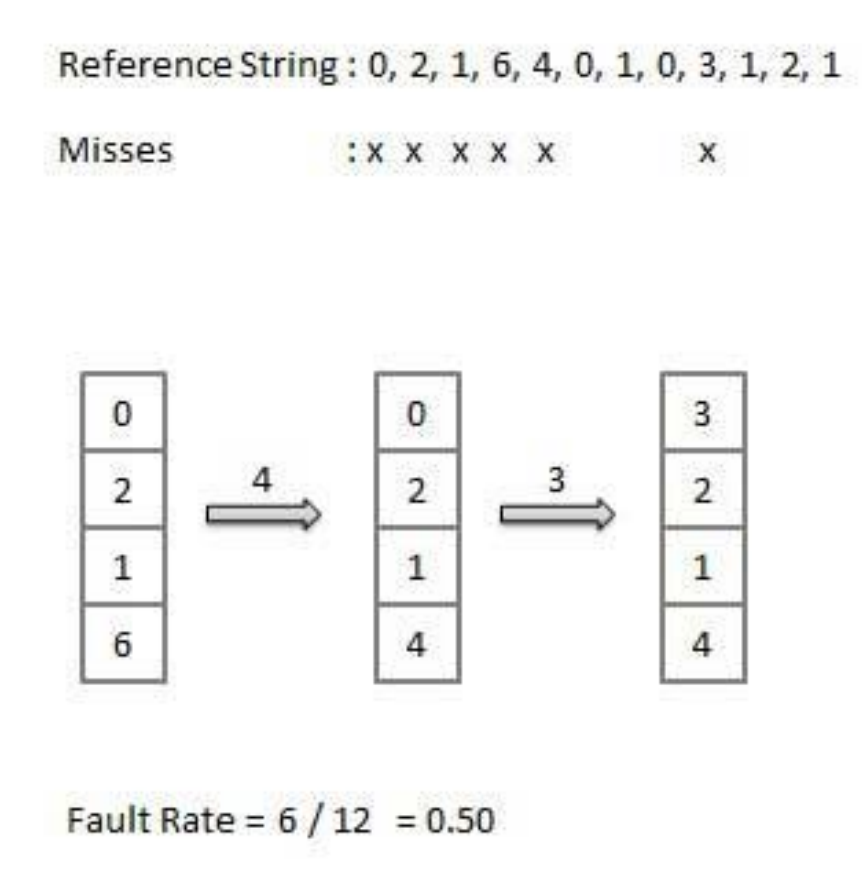
From a theoretical point of view, on a sample series of page references, we can compare the performance of other algorithms to the optimal algorithm.

CS240 Operating Systems, Communications and Concurrency

Page Replacement Algorithms

Say we have a working set of 4 pages, all initially empty and a string of page references generated by a process.

How many page faults result using the optimal page replacement algorithm?



CS240 Operating Systems, Communications and Concurrency

Page Replacement Algorithms

The **First In First Out** Algorithm replaces the page that has been in the memory the longest.

A timestamp is associated with the page when it is first brought into memory, or treat the working set as a circular array.

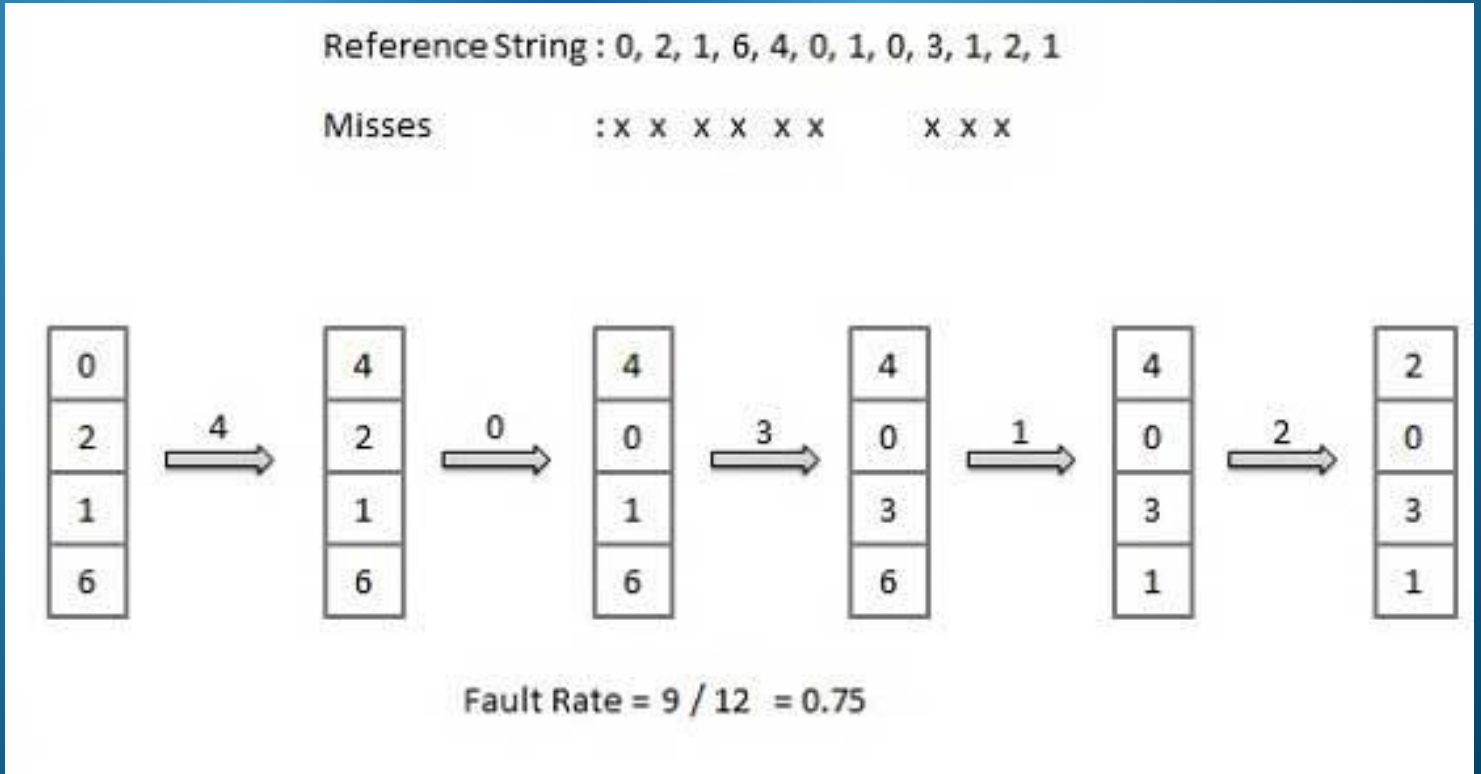
The timestamp does not need to be updated and so the algorithm is very easy and cheap to implement.

However, the behaviour of FIFO is not particularly suited to the behaviour of most programs.

CS240 Operating Systems, Communications and Concurrency

Page Replacement Algorithms

How many page faults result using the FIFO replacement algorithm?



Page Replacement Algorithms – Process behaviour heuristics

The **Least Recently Used algorithm** is based on the idea that if a process has recently used a page then it is likely to need that page again shortly because a lot of our code contains loops.

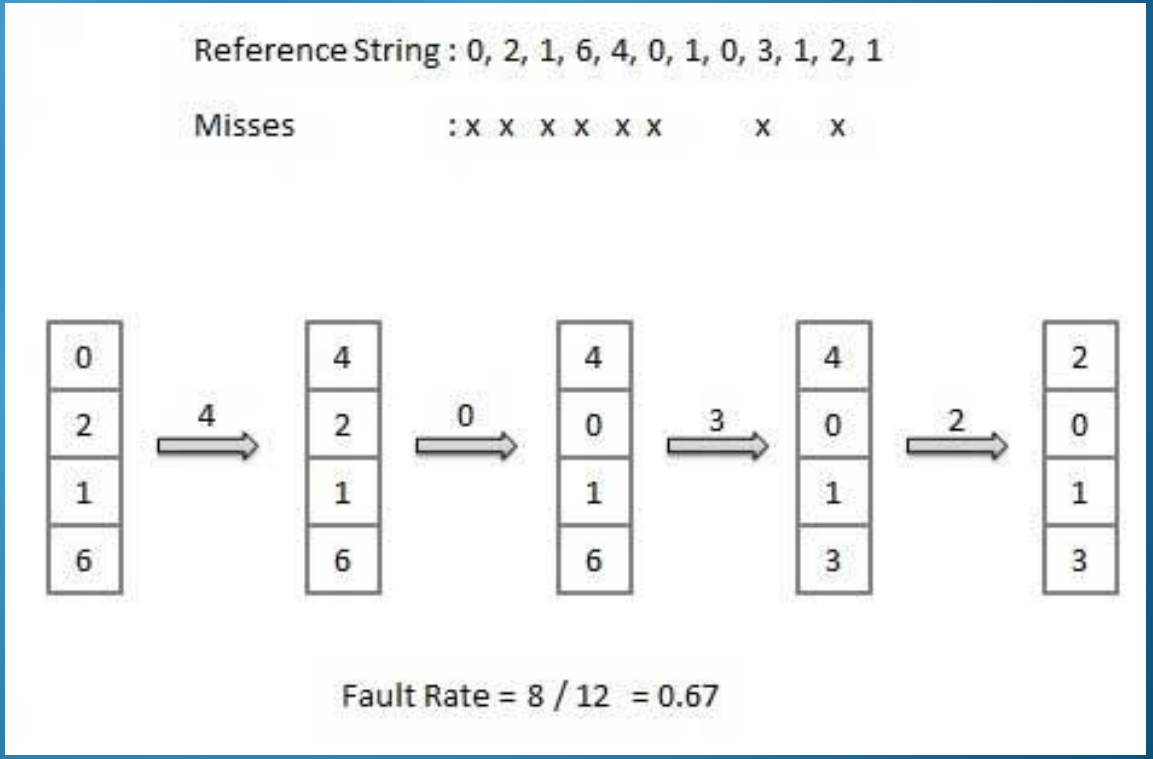
The algorithm therefore selects the page which hasn't been accessed for the longest time for replacement. As our programs tend to move from one locality to another, it is less likely that we need the pages of an old locality to be kept in the working set.

This algorithm would require a timestamp to be associated with each page in the page table and the hardware would have to update this timestamp each time an access to a memory address within the page was made.

CS240 Operating Systems, Communications and Concurrency

Page Replacement Algorithms

How many page faults result using the Least Recently Used algorithm?



Page Replacement Algorithms – Process behaviour heuristics

The **Least Frequently Used** algorithm replaces the page with the fewest references over a past period of time.

The idea is that this page will cause the fewest page faults if it is not often needed.

However, when a process changes its locality of reference, the new pages it requires will initially be low in usage frequency and will be selected for replacement even though they are needed.

Least frequently used requires a counter to be associated with each page table entry which is incremented whenever a memory access to that page is made.

Page Replacement Algorithms

Belady's Anomaly

Intuitively, it might appear that the more page frames available to a process, the fewer page faults will occur.

In general, this is true, however it is not guaranteed as the next example demonstrates.

This anomaly was discovered first in 1970 and is known as Belady's Anomaly.

Belady's Anomaly

Consider a page reference string where pages are referenced in the following order:-

0 1 2 3 0 1 4 0 1 2 3 4

Using a FIFO algorithm with 3 pages in the working set, we get a total of nine page faults as follows:-

	0	1	2	3	0	1	4	4	4	2	3	3
		0	1	2	3	0	1	1	1	4	2	2
			0	1	2	3	0	0	0	1	4	4

	PF	PF	PF	PF	PF	PF	PF			PF	PF	

Belady's Anomaly

Using a FIFO algorithm with 4 pages in the working set, we get a total of ten page faults as follows:-

	0	1	2	3	3	3	4	0	1	2	3	4
		0	1	2	2	2	3	4	0	1	2	3
			0	1	1	1	2	3	4	0	1	2
				0	0	0	1	2	3	4	0	1
	PF	PF	PF	PF			PF	PF	PF	PF	PF	PF

So with a bigger working set we ended up with more page faults.