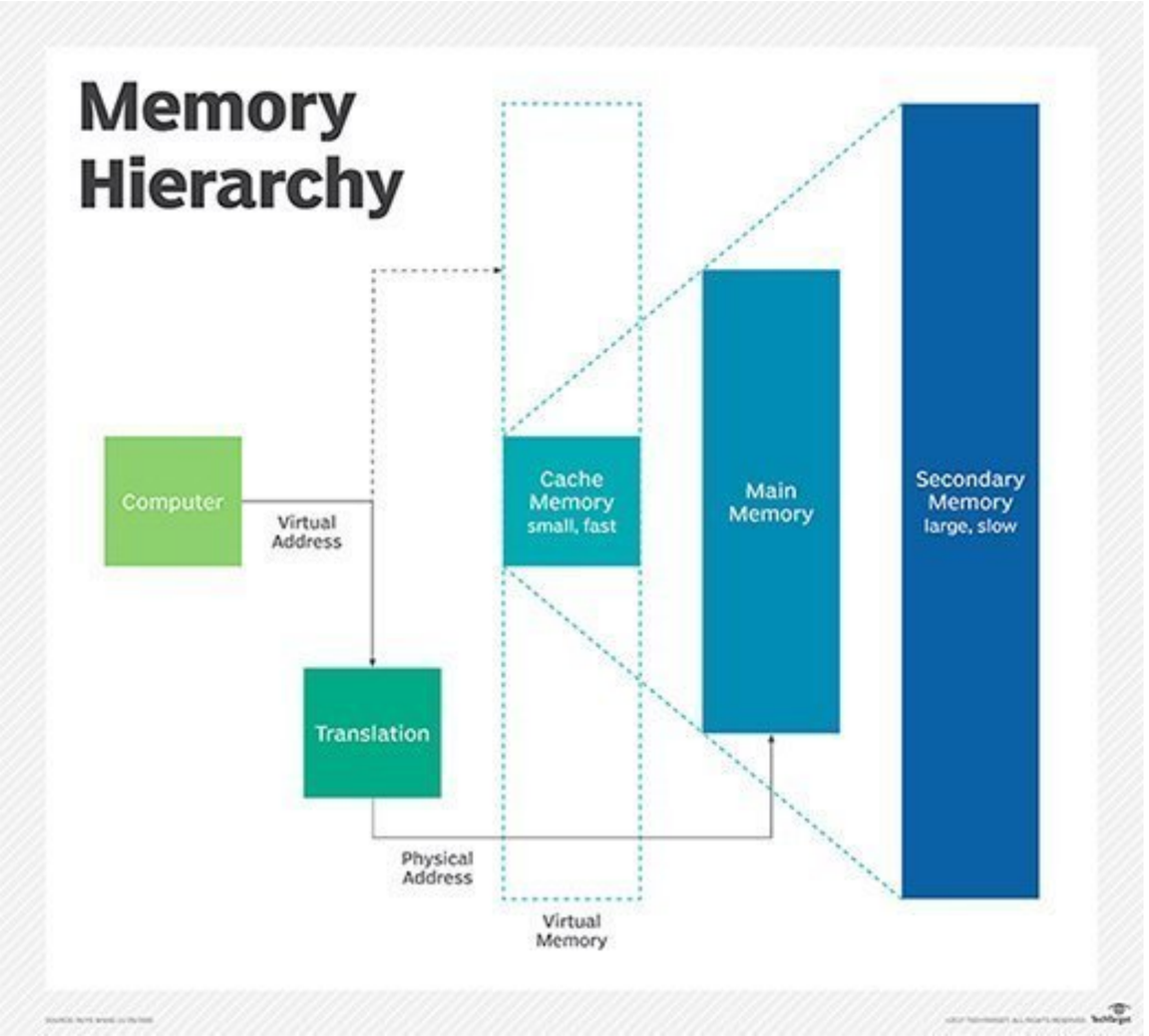
CPSC 5042: Week 3

Understanding Virtual Memory

Last Week: Understanding Main Memory

- Single User Contiguous Scheme
    - Program was loaded into memory contiguously
- Fixed Partitions
    - Lot of work trying to perfect partition sizes
- Dynamic Partitions
    - Defragmentation Management 😥
    - Allocation Strategies: First Fit, Best Fit, Worst Fit
    - Deallocation: Merge blocks
- Swapping vs Compaction


Memory Hierarchy
Computer
Virtual Address
Translation
Physical Address
Cache Memory
small, fast
Main Memory
Secondary Memory
large, slow
Virtual Memory

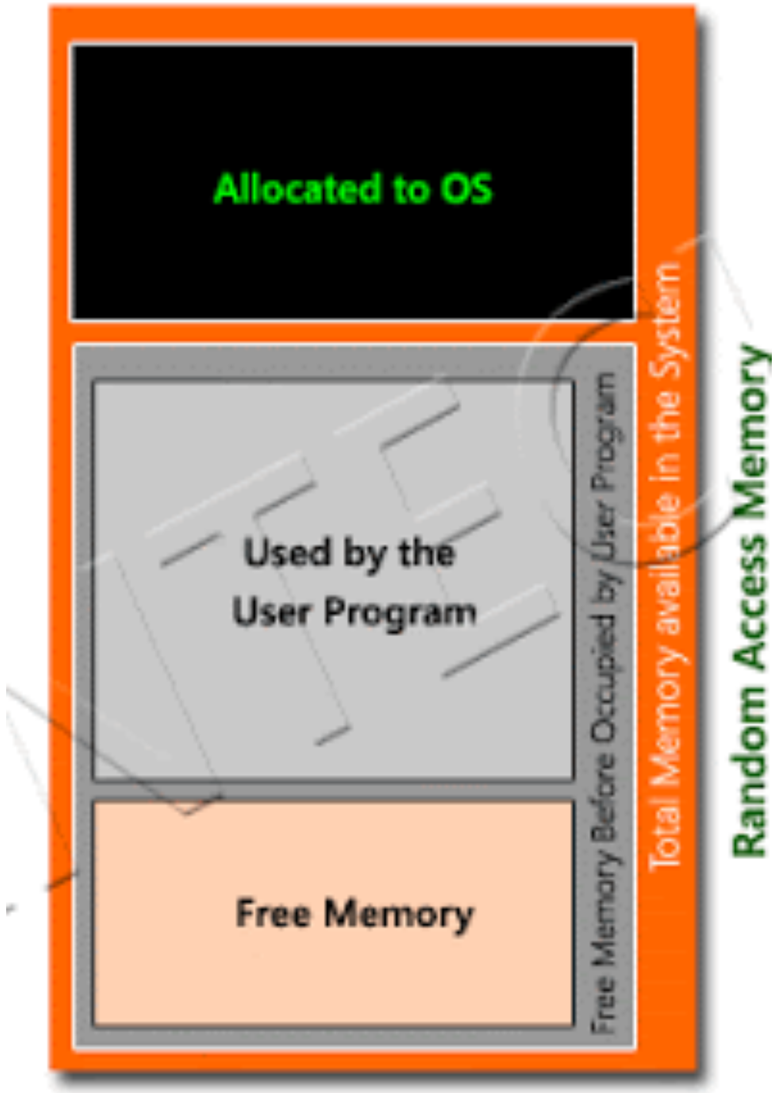
Schemes we looked at last week

- Single User CONTIGUOUS Scheme
- Fixed Partition CONTIGIOUS Scheme
- Dynamic Partition CONTIGIOUS Scheme
- Relocatable Dynamic Partitions CONTIGIOUS Scheme
-

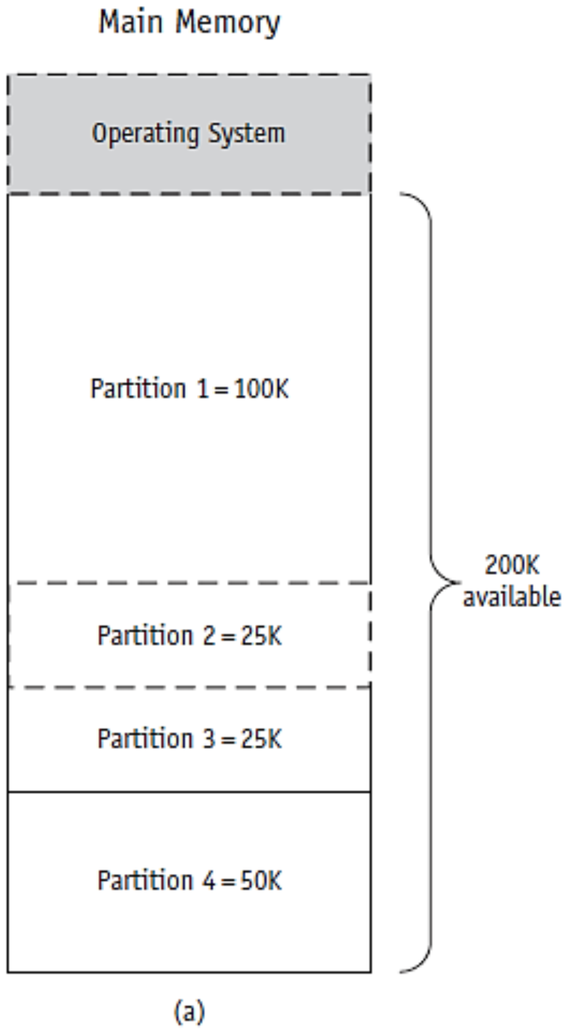
Schemes we looked at last week

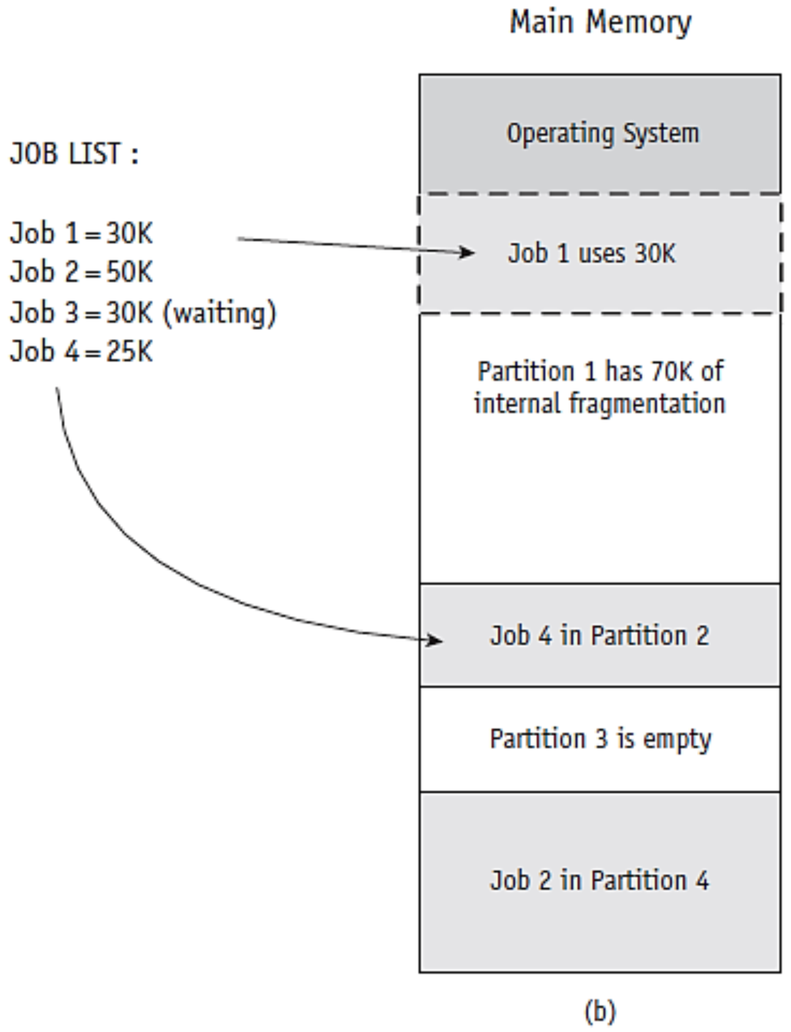
- Single User CONTIGUOUS Scheme
- Fixed Partition CONTIGIOUS Scheme
- Dynamic Partition CONTIGIOUS Scheme
- Relocatable Dynamic Partitions CONTIGIOUS Scheme
- Is there a word that is repeating itself?

Quick Review: Single User

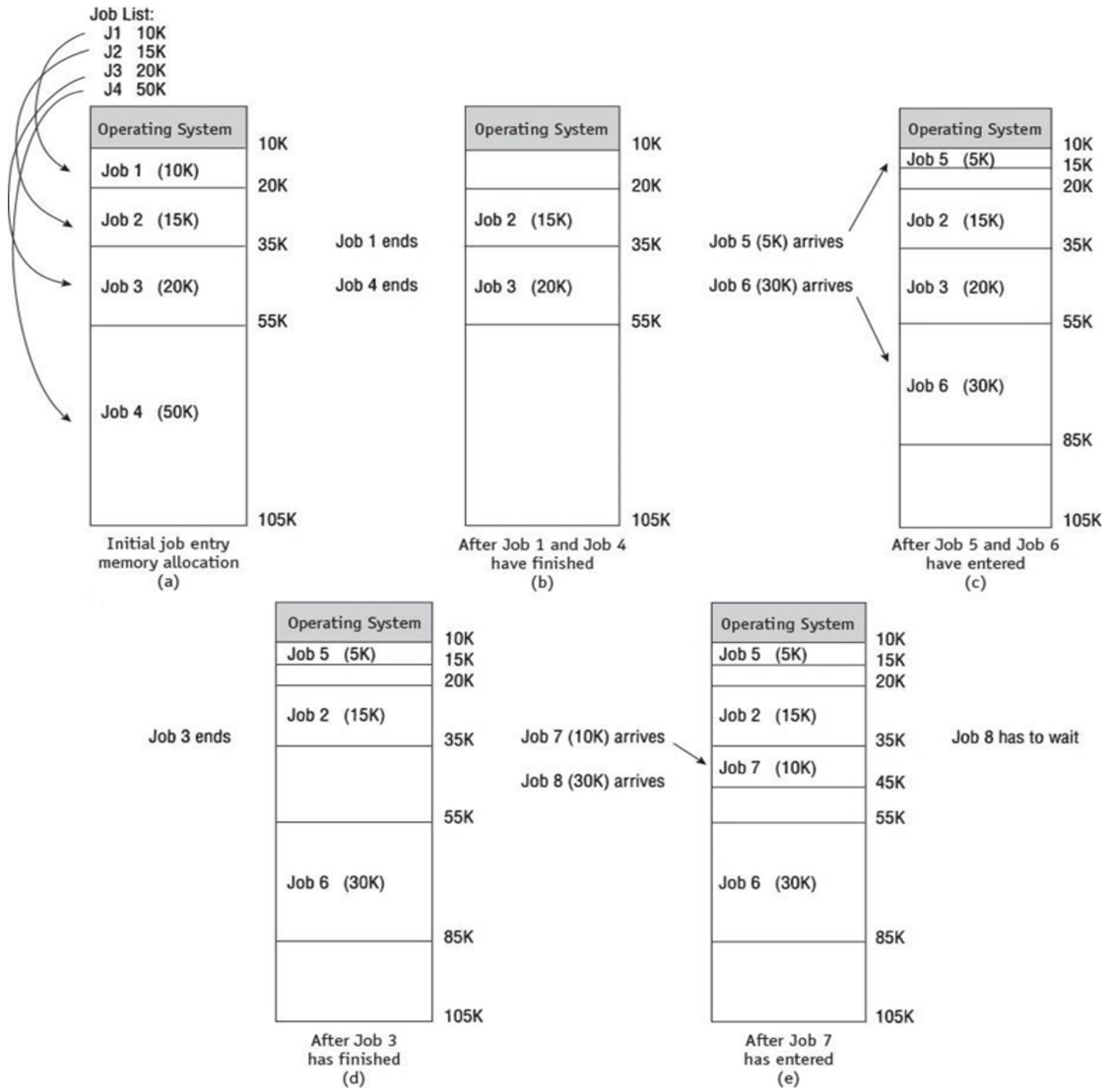


Quick Review: Fixed Partitions





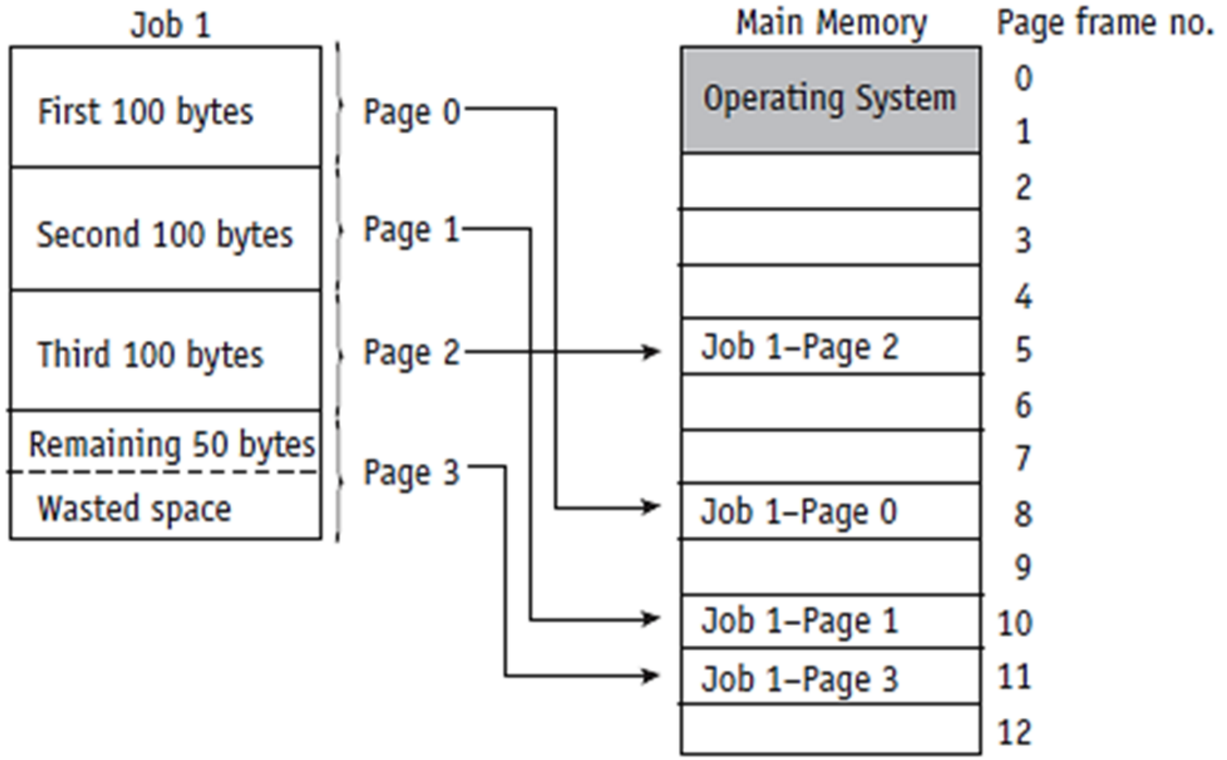
Quick Review: Dynamic Partitions



Paged Memory Allocation

Paged Memory Allocation

- Entire Job is stored in Main Memory
- Page Frame refers to Main Memory and in this example is 100 bytes



Paged Memory Allocation

- Internal fragmentation: job's **last page frame only**
- Fits entire program: required in memory during its execution
- Three tables for tracking pages:
  - Job Table (JT)
  - Page Map Table (PMT)
  - Memory Map Table (MMT)
- Stored in main memory: operating system area

Paged Memory Allocation

- **Job Table: information for each active job**
  - Job size
  - Memory location: job's PMT
- Example:
  - A -> Three jobs are loaded,
  - B -> one exits
  - C -> another takes it place

Job Table

Job Size	PMT Location
400	3096
200	3100
500	3150

(a)

Job Table

Job Size	PMT Location
400	3096
500	3150

(b)

Job Table

Job Size	PMT Location
400	3096
700	3100
500	3150

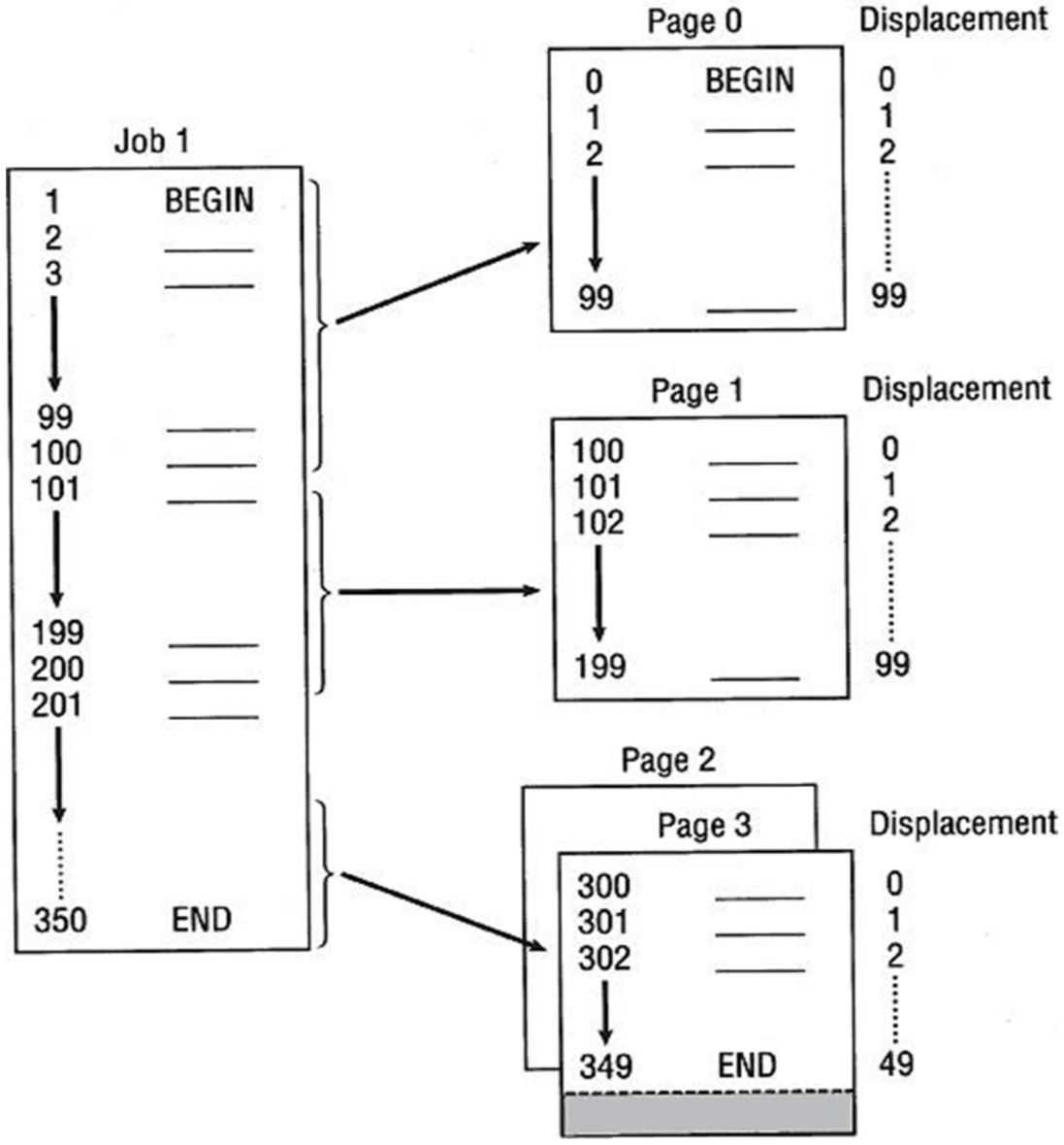
(c)

Paged Memory Allocation

- Process Map Table (PMT)
  - Each Process will have a list of Page Frame Numbers (which refer to the “Real” Pages in main memory)
  - This is how things are found
- This Process Map Table refers to a job that was in the Job table (e.g. the job of size 500)

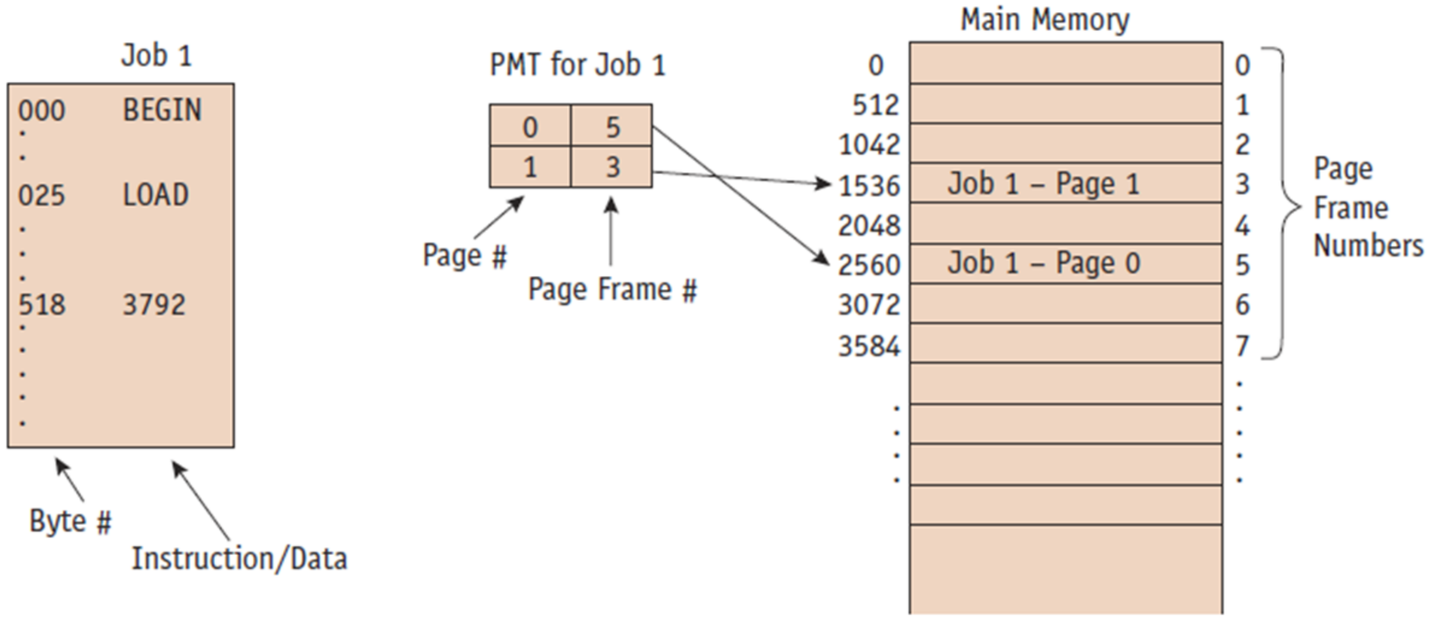
Page Number	Page Frame Number
0	8
1	10
2	5
3	11

Paged Memory Allocation



Paged Memory Allocation

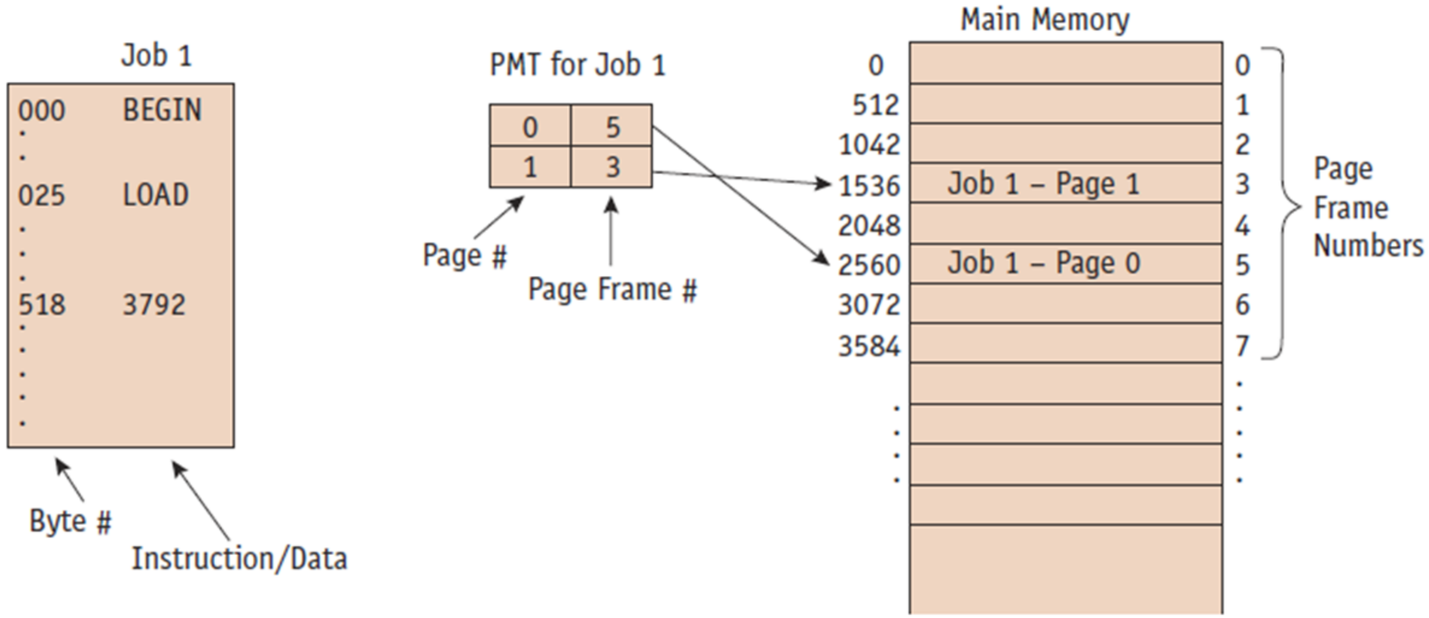
- Each page is 512 bytes here:



- logical_address/page_size = logical page_number
- displacement = logical_address % page_size
- physical_page = pmt[logical_page_number].page_frame
- physical_address = physical_page * page_size + displacement

Paged Memory Allocation

- Each page is 512 bytes here:

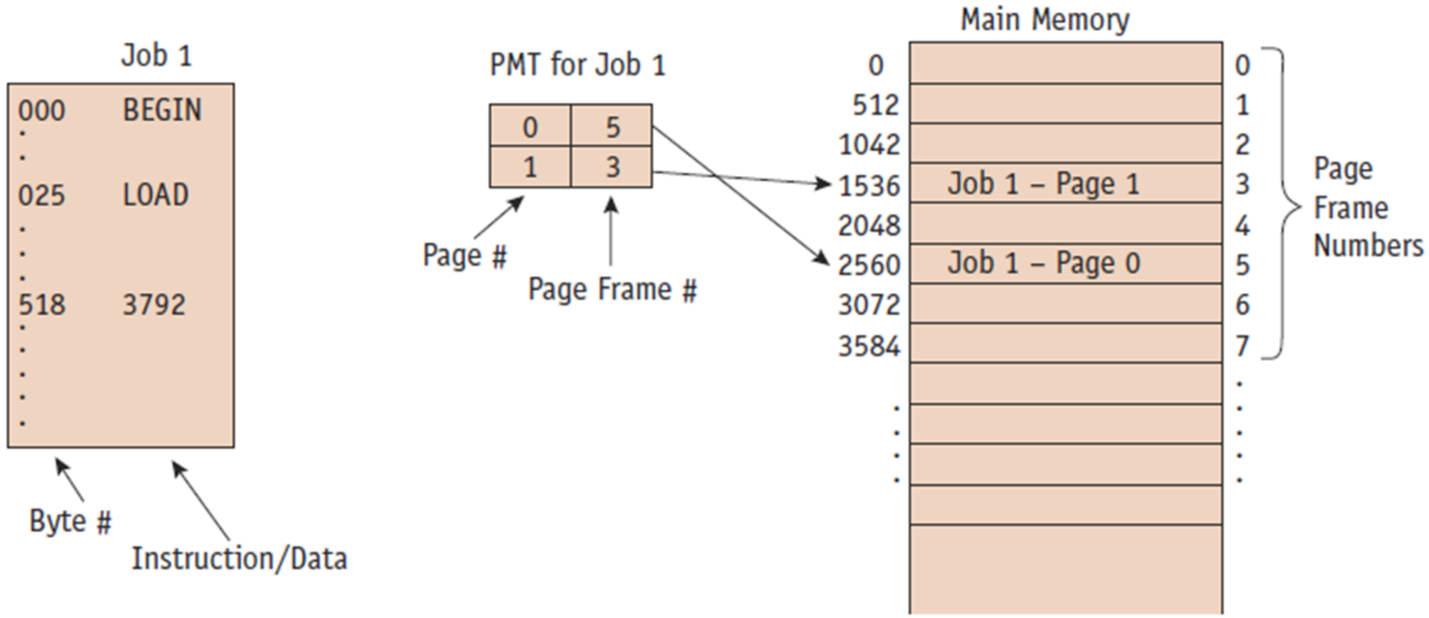


- logical_address/page_size = logical page_number
- displacement = logical_address % page_size
- physical_page = pmt[logical_page_number].page_frame
- physical_address = physical_page * page_size + displacement

- **Which address do you think the value 3792 ends up being?**

Paged Memory Allocation

- Each page is 512 bytes here:



- logical_address/page_size = logical page_number
- displacement = logical_address % page_size
- physical_page = pmt[logical_page_number].page_frame
- physical_address = physical_page * page_size + displacement

- **Which address do you think the value 3792 ends up being?**
- **Ans = 1542**

Paged Memory Allocation

- Advantages
  - Efficient memory use: job allocation in non-contiguous memory
- Disadvantages
  - Increased overhead: address resolution
  - Internal fragmentation: last page
- Page size: crucial
  - Too small: very long PMTs
  - Too large: excessive internal fragmentation

Questions?

Toilet Paper Analogy

- This picture likely describes why we had a Toilet Paper Shortage.
- If people simply bought what they could use (Demand-based) there would likely be enough for everybody.
-
-

Toilet Paper Analogy

- This picture likely describes why we had a Toilet Paper Shortage.
- If people simply bought what they could use (Demand-based) there would likely be enough for everybody.
- To get more throughput only the Pages that are needed are loaded into memory.
- We don’t try to load the entire program into memory!

Probably only use 10% of program?

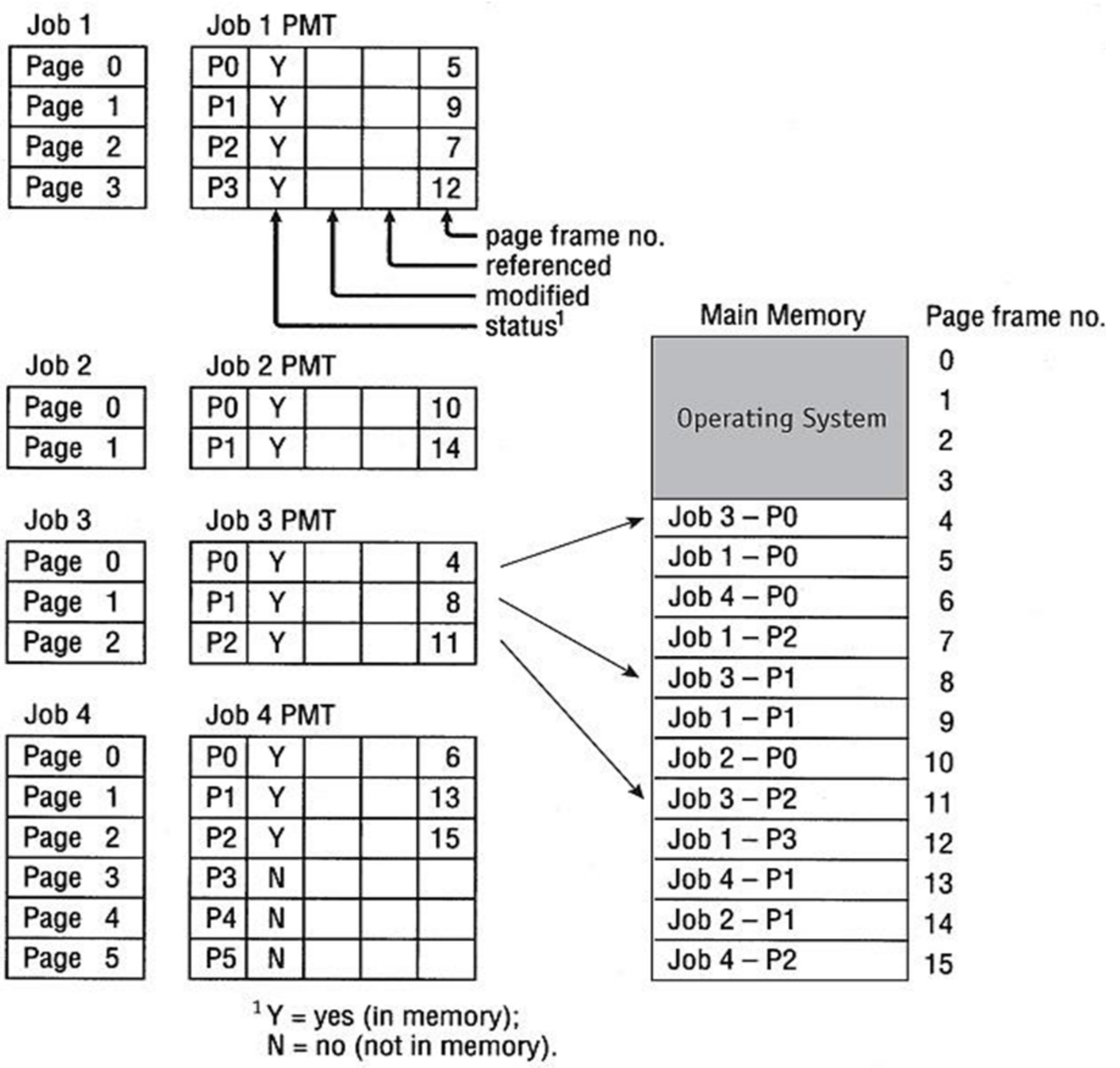
- When you run a program, you might be accessing only a subset of program
  - Static Libraries linked in never used
  - Data that you allocated never used
  - Certain subroutines/functions in other modules never referenced

Probably only use 10% of program?

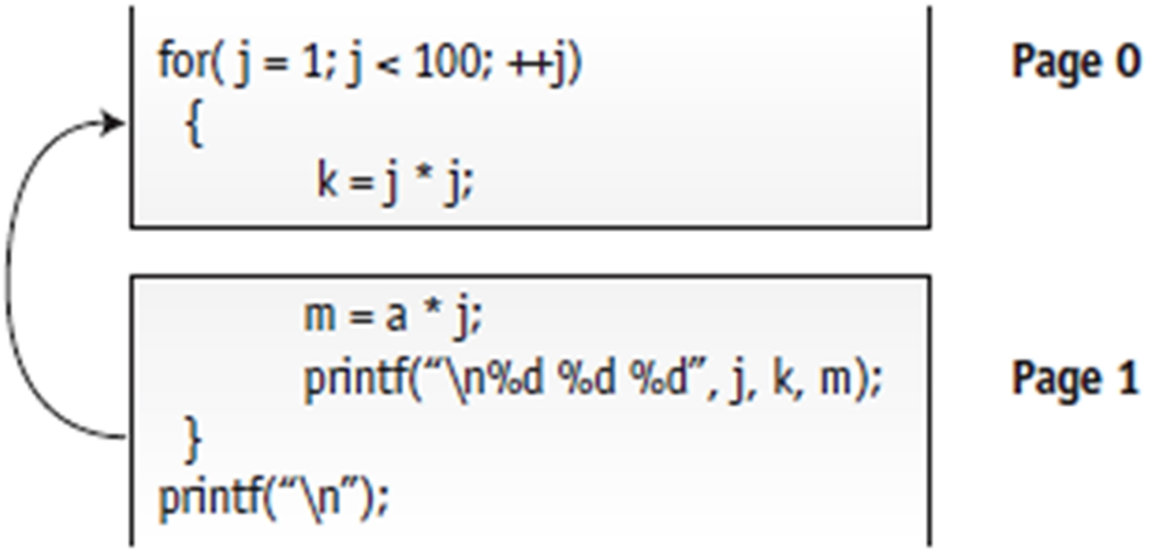
- Knowing this, you can see why demand paging would be a great way to conserve memory.
- It requires additional management to implement properly,
  - but the savings in memory have made this popular with operating systems.
- You can have large programs
  - (but not necessarily all in main memory at same time).
- Typically needs faster drives, as likely there will be more swapping,
  - because pages will be swapped out as you run the program

Demand Paging Memory Allocation

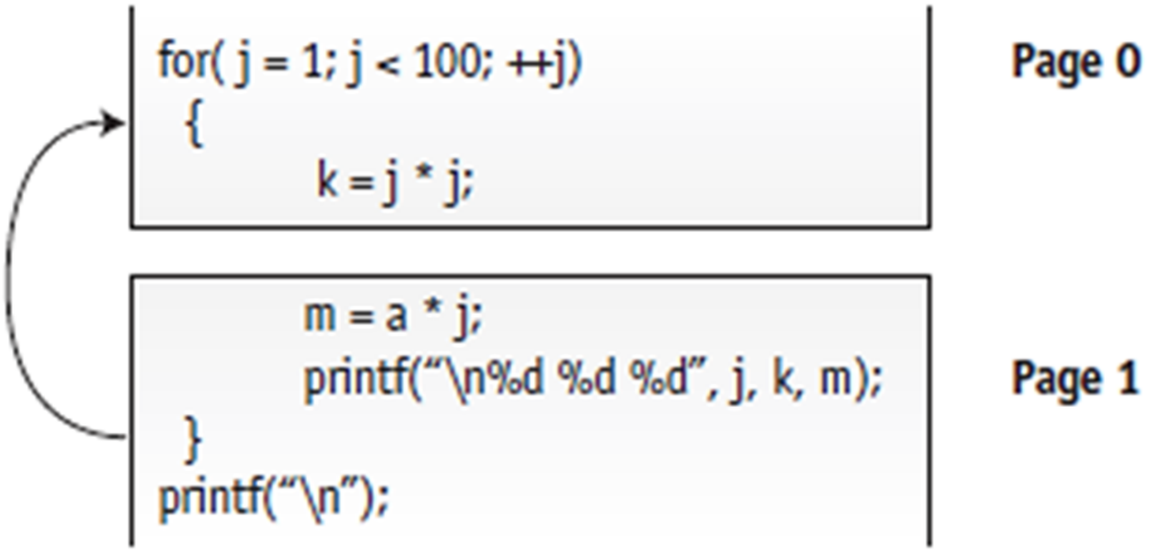
- Demand Paging only loads what it needs
- When it finds it doesn’t have the page in memory, it will load it (but maybe kick somebody else out)
- More Jobs/Less Memory, but ….
    - Excessive Thrashing could occur with Page Faults leading to slowness
- Have you ever had 120 tabs open? 🤭
- Requires various Algorithm’s and tables such as:
    - Job Table
    - Page Map Table
    - Memory Map Table


Job 1
Page 0
Page 1
Page 2
Page 3
Job 1 PMT
P0 Y 5
P1 Y 9
P2 Y 7
P3 Y 12
page frame no.
referenced
modified
status[1]
Job 2
Page 0
Page 1
Job 2 PMT
P0 Y 10
P1 Y 14
Job 3
Page 0
Page 1
Page 2
Job 3 PMT
P0 Y 4
P1 Y 8
P2 Y 11
Job 4
Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Job 4 PMT
P0 Y 6
P1 Y 13
P2 Y 15
P3 N
P4 N
P5 N
Main Memory
Page frame no.
Operating System
0
1
2
3
Job 3 – P0 4
Job 1 – P0 5
Job 4 – P0 6
Job 1 – P2 7
Job 3 – P1 8
Job 1 – P1 9
Job 2 – P0 10
Job 3 – P2 11
Job 1 – P3 12
Job 4 – P1 13
Job 2 – P1 14
Job 4 – P2 15
[1] Y = yes (in memory);
N = no (not in memory).

What happens here?



What happens here?



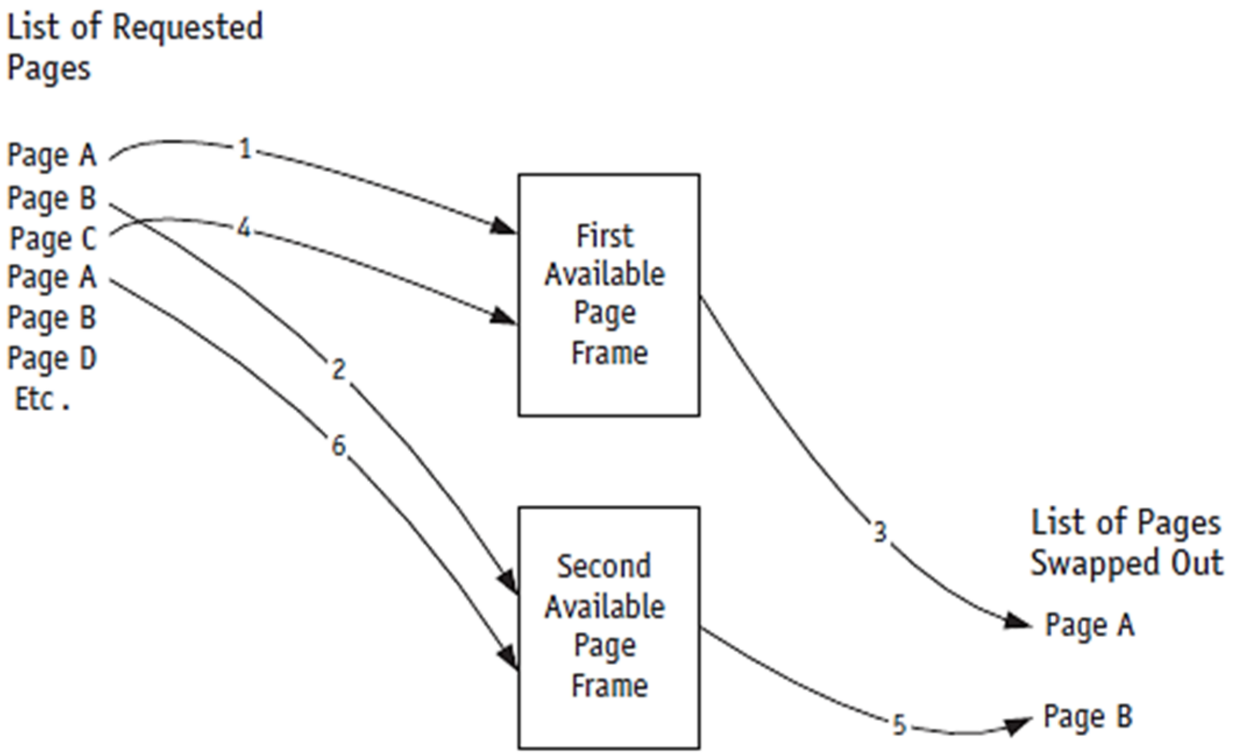
- You still need both pages in the memory
- If there's absolutely no space, you bring in 0 first, then swap for 1
- This is a **Page Fault**
- Again and again - 100 times!
- Its so fast, you'll probably never notice
- If you do, you are experiencing **thrashing**
  - Any disk I/O is super expensive & slow
  - Cost of pretending "unlimited memory"

Who decides which page leaves?

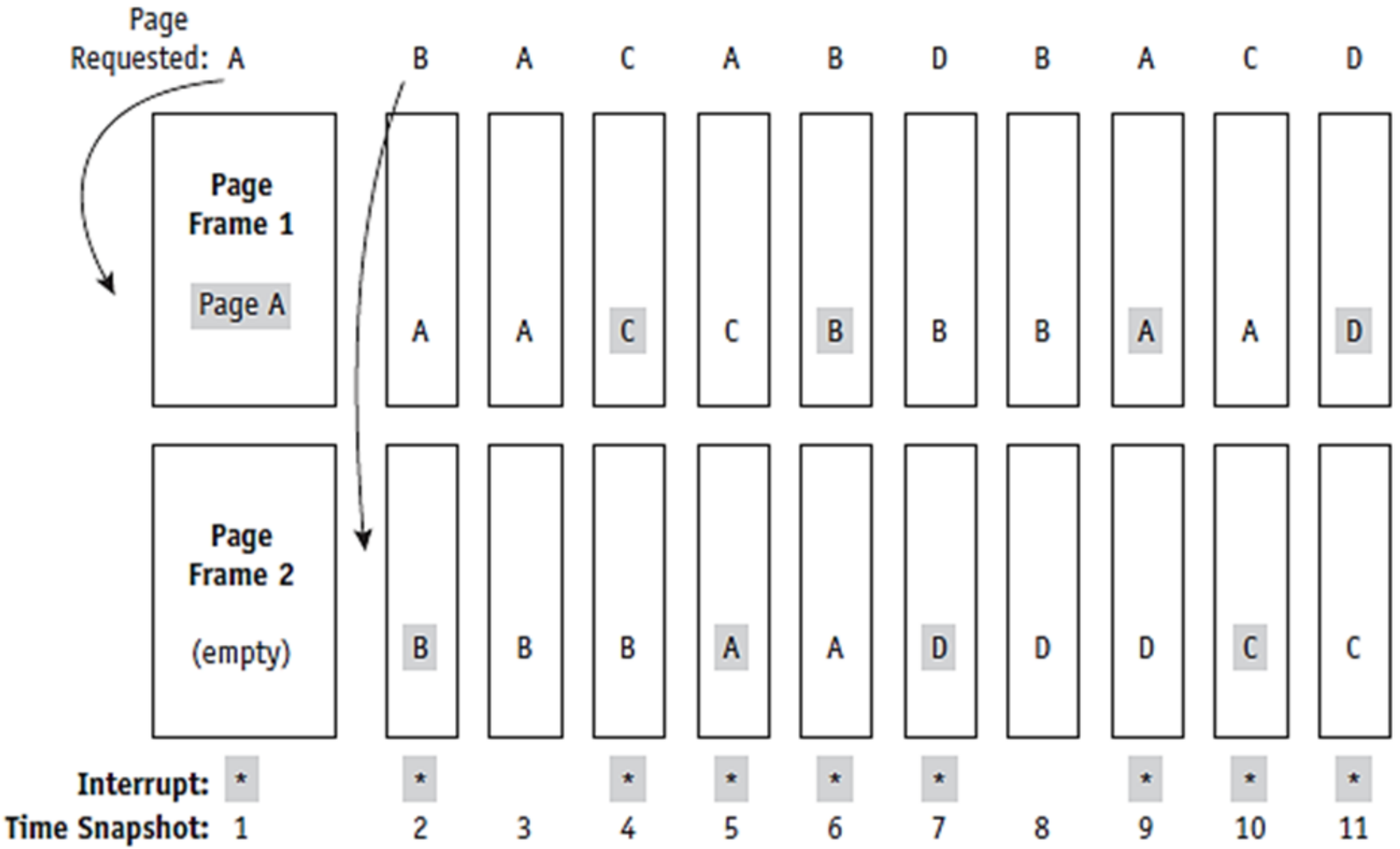
- Two well-known algorithms
  - First-in first-out (F I F O) policy
    - page in memory longest removed
  - Least Recently Used (L R U) policy
    - page least recently accessed removed

FIFO with 2 Frames

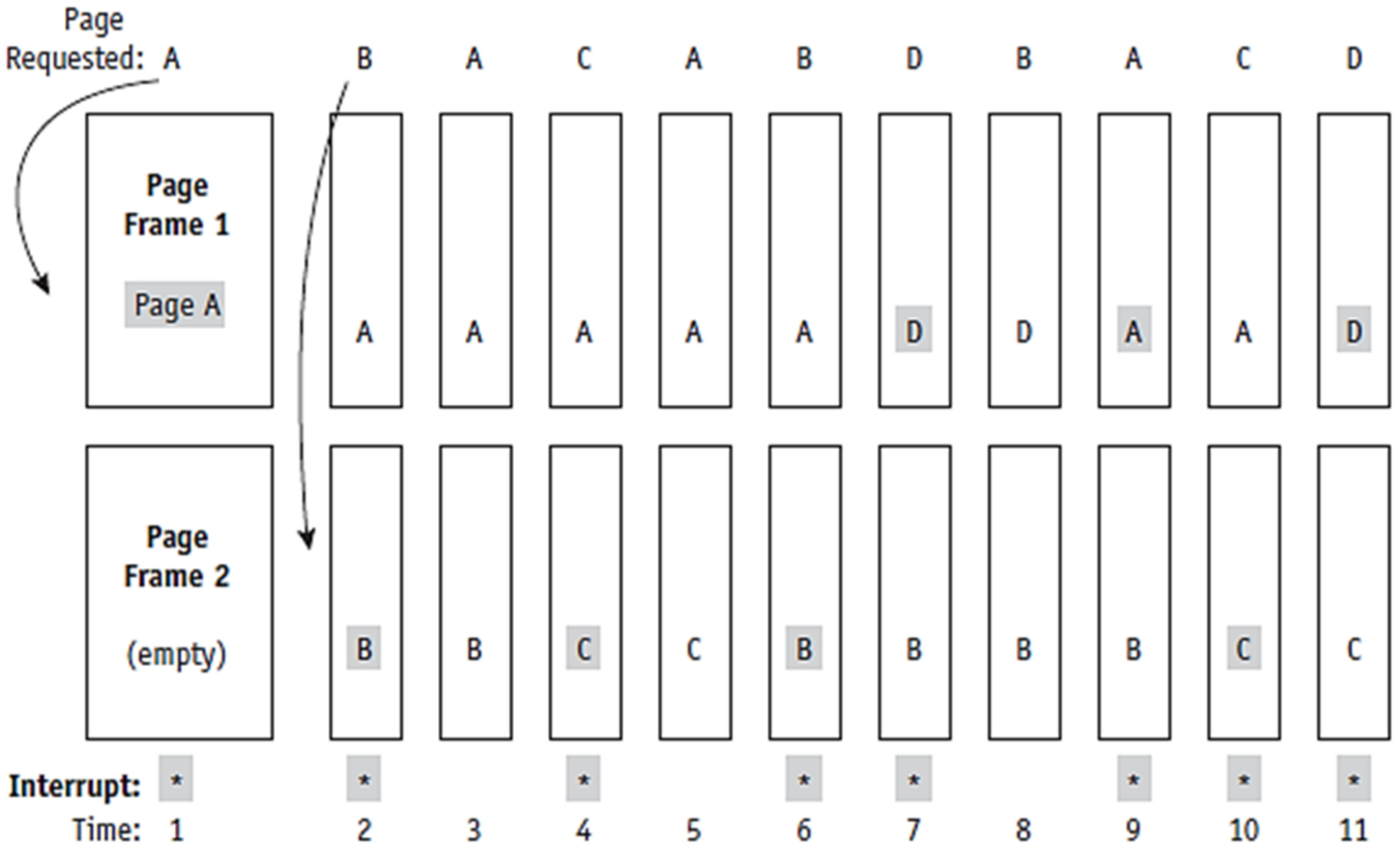
- Likely won’t happen with just 2 frames, but you get the idea:



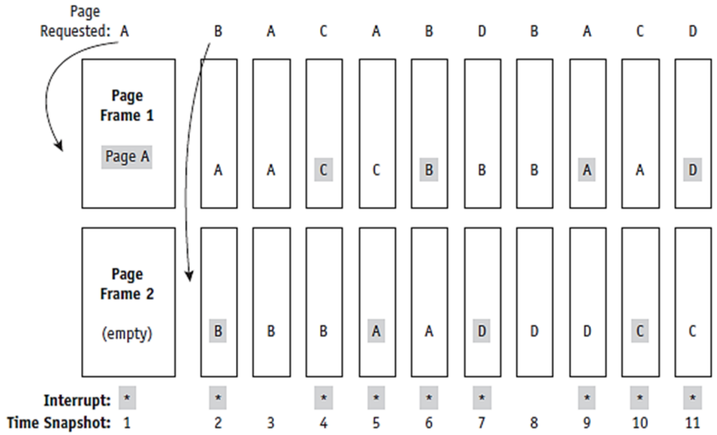
FIFO with 2 Frames

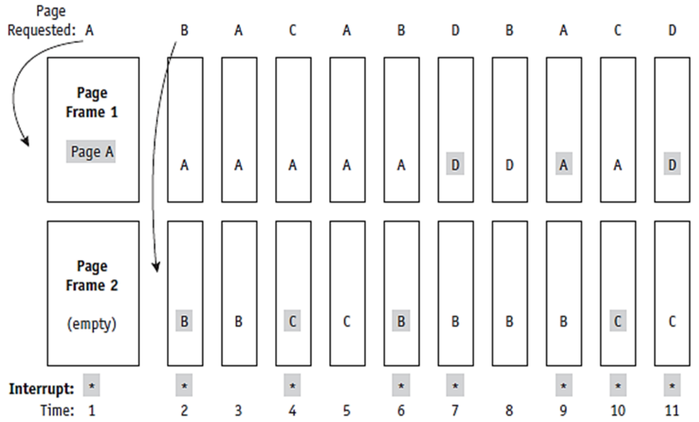


LRU with 2 Frames



Lets compare





- FIFO
- Page Faults: 9

- LRU
- Page Faults: 8

Will increasing the number of page frames help?

Solve:

- 3 Frames
- Order of Requests:
  - **1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**

Frame 2												
Frame 1												
Frame 0												
Fault?												

- Faults = ?

Answer:

- 3 Frames
- Order of Requests:
  - **1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**

Frame 2			3	3	3	2	2	2	2	2	4	4
Frame 1		2	2	2	1	1	1	1	1	3	3	3
Frame 0	1	1	1	4	4	4	5	5	5	5	5	5
Fault?	*	*	*	*	*	*	*			*	*	

- Faults = 9 (out of 12)

Solve:

- 4 Frames
- Order of Requests:
  - **1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**

Frame 3												
Frame 2												
Frame 1												
Frame 0												
Fault?												

- Faults = ?

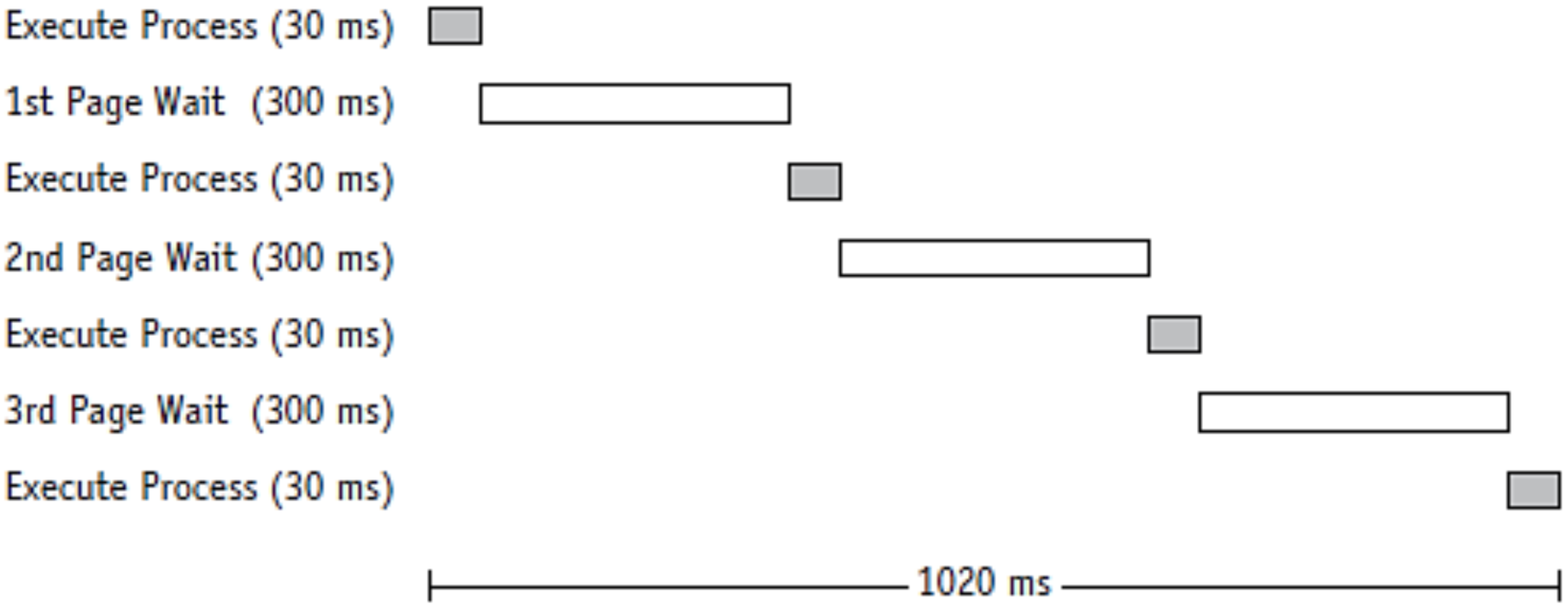
Answer:

- 4 Frames
- Order of Requests:
  - **1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**

Frame 3				4	4	4	4	4	4	3	3	3
Frame 2			3	3	3	3	3	3	2	2	2	2
Frame 1		2	2	2	2	2	2	1	1	1	1	5
Frame 0	1	1	1	1	1	1	5	5	5	5	4	4
Fault?	*	*	*	*			*	*	*	*	*	*

- Faults = 10 (out of 12)

Page Faults affect perfomance



Other LRUs

- Clock Replacement
  - Where it uses a system clock and the concept of LRU to denote which one to replace
  - e.g. 2 milliseconds, which pages were not used? Replace them.
- Bit Shifting Variation
  - It is an offshoot of LRU that use a 8 byte reference byte, where each time it is referenced it will toggle a bit, and then move to the next bit. The system will decide which byte is more least referenced and replace

PMT for Demand Paging

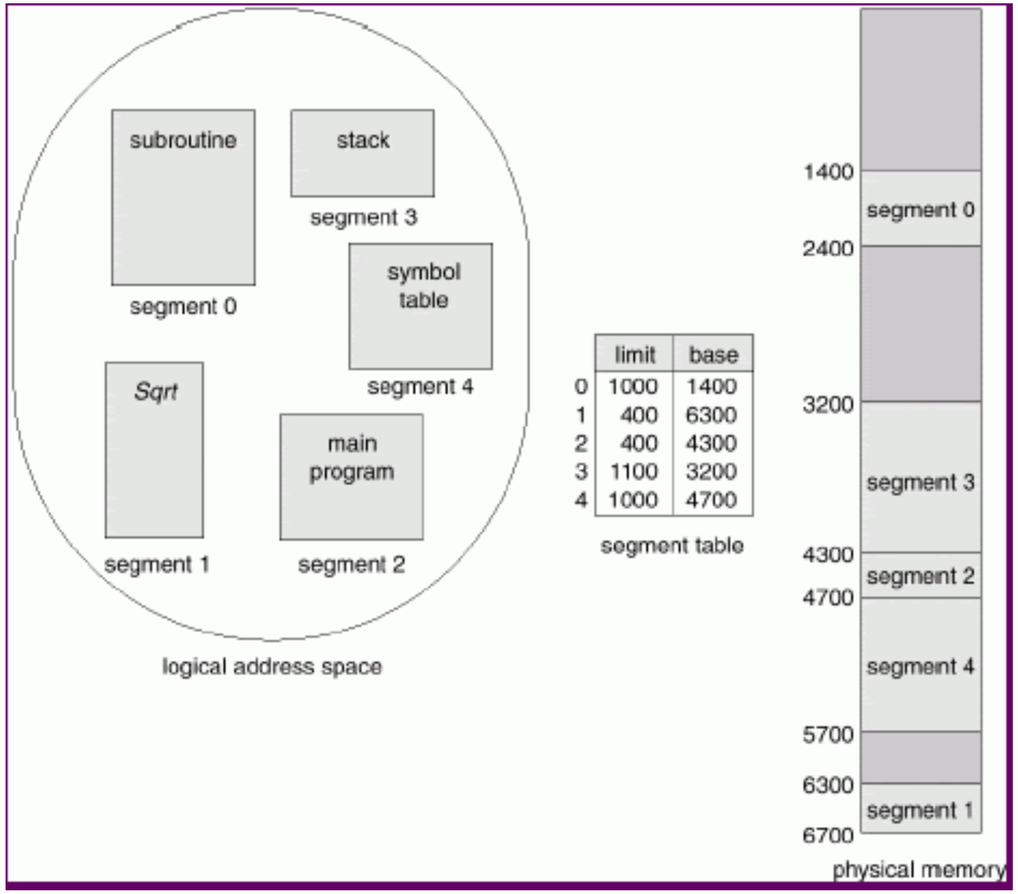
Page No.	Status Bit	Modified Bit	Referenced Bit	Page Frame No.
0	1	1	1	5
1	1	0	0	9
2	1	0	0	7
3	1	0	1	12

- PMT for Demand Page needs extra bits
- Status bit says whether it is in memory
- Modified means it was adjusted (probably data)
- Referenced means it was used (for LRU)

Segmented Memory Allocation

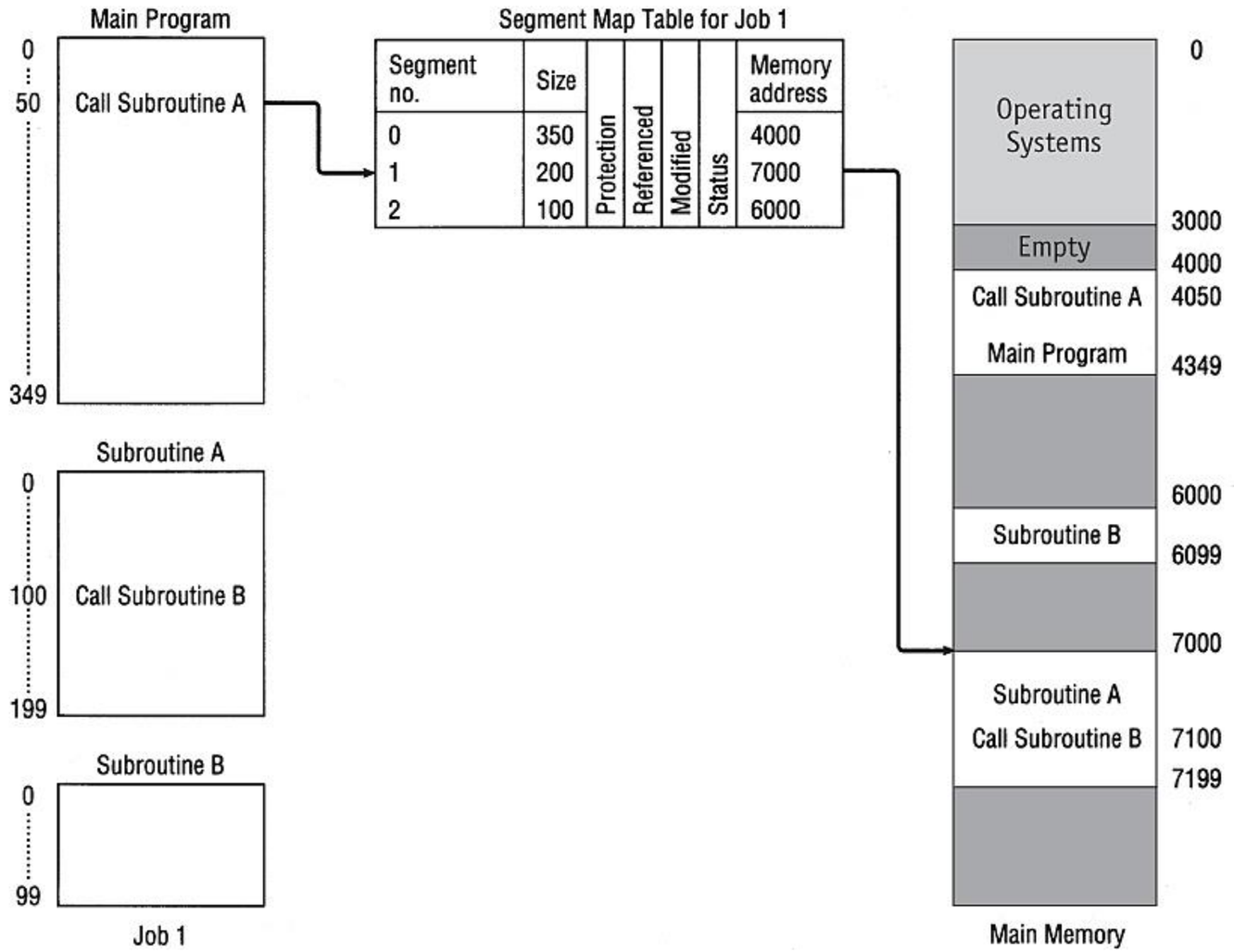
- Programs are physically split into segments (not pages), and the thought was to use this Allocation technique in order to prevent “internal fragmentation”
- Programmers have some control over putting things in segments
    - Its described in the way you code (so you can modularize things now)

Segmented Memory Allocation



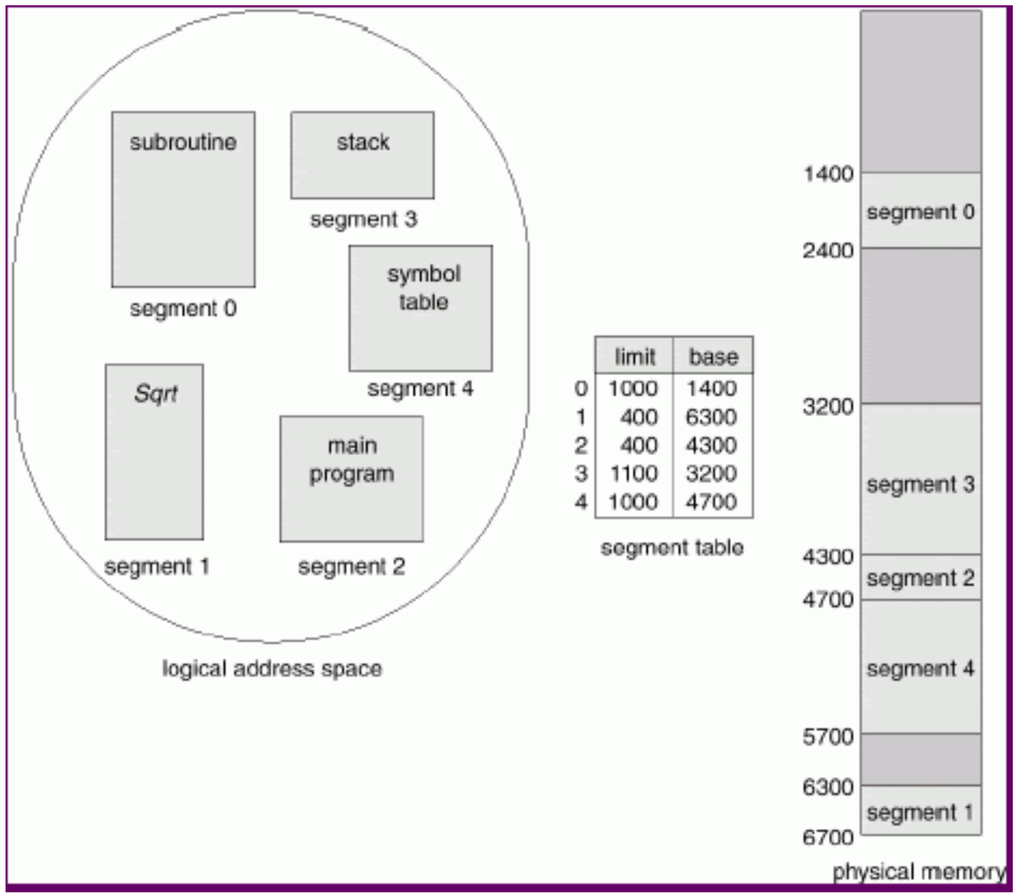
- A program consists of many segments that make up the Program
- They will eventually end up in Physical memory as a “unit” similar to page but not constrained to a certain page size
- Some of the advantages was like that for loop program we saw earlier. A fault would not happen, as that code would be together.

Calling a subroutine in a different segment



- You can see that calling various subroutines will generate calls that require lookups and potentially page faults to load new segments
- This technique is interesting in you won't generate faults, but comes at a disadvantage of external fragmentation. As segments/processes get released it creates holes

Segmented Memory Allocation



- Two-dimensional addressing scheme
  - Segment number and displacement
- Disadvantage
  - External fragmentation
- Major difference between paging and segmentation
  - Pages: physical units; invisible to program
  - Segments: logical units; visible to program; variable sizes

Comparing

Virtual Memory with Paging	Virtual Memory with Segmentation
Allows internal fragmentation within page frames	Doesn’t allow internal fragmentation
Doesn’t allow external fragmentation	Allows external fragmentation
Programs are divided into equal-sized pages	Programs are divided into unequal-sized segments that contain logical groupings of code
The absolute address is calculates using the page number and displacement	The absolute address is calculated using the segment number and displacement
Requires Page Map Table (PMT)	Requires Segment Map Table (SMT)

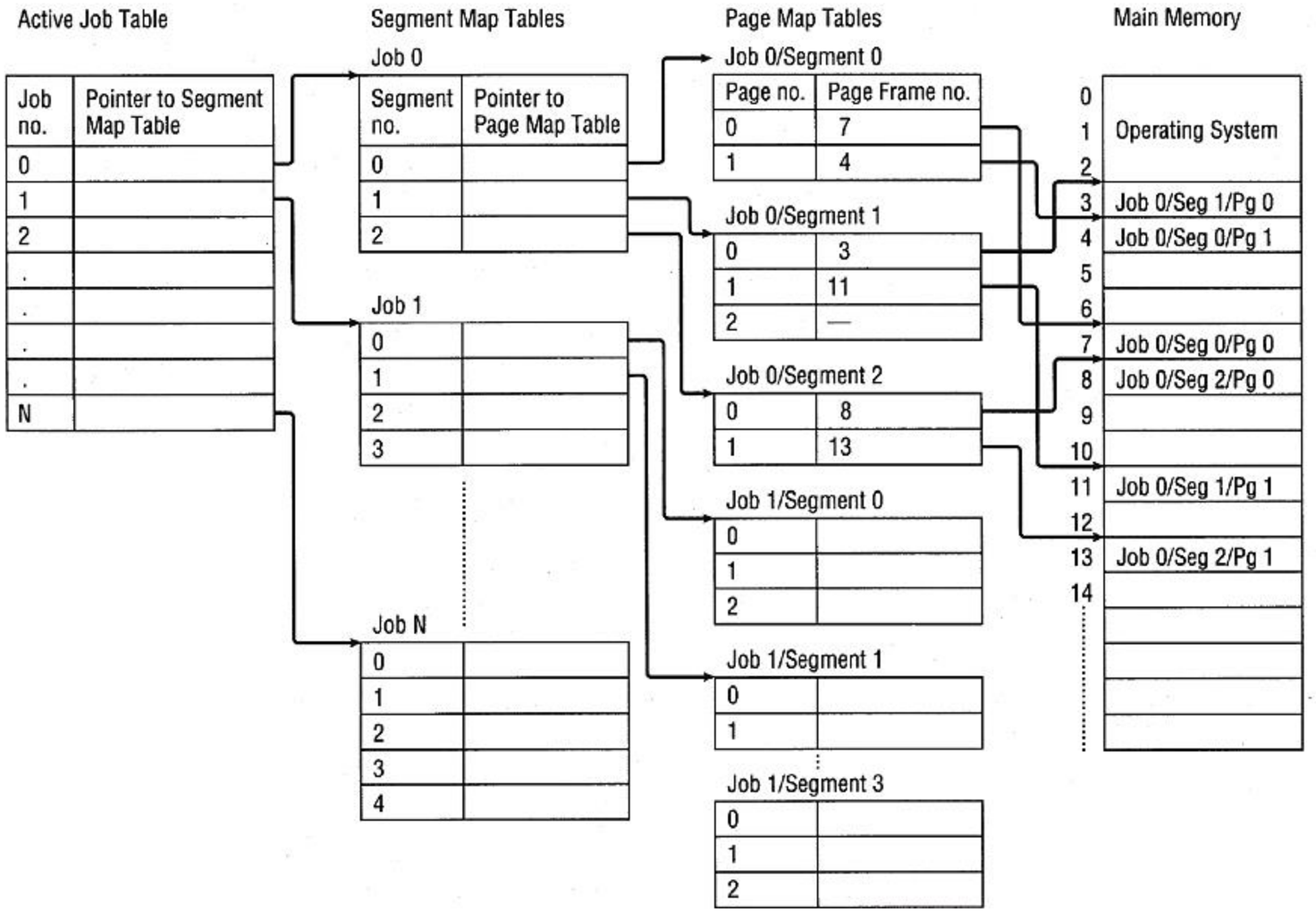
What if we combined Paging & Segmentation?

- This is the approach most OS’ s have
- Segments get put into equal-size pages
- External fragmentation of segments removed
- Three-dimensional addressing:
  - Segment, Page number (within segment) and displacement

Requirements of this system

- Scheme requires four tables
  - Job Table: one for the whole system
    - Every job in process
  - Segment Map Table: one for each job
    - Details about each segment
  - Page Map Table: one for each segment
    - Details about every page
  - Memory Map Table: one for the whole system
    - Monitors main memory allocation: page frames

Paging + Segmentation

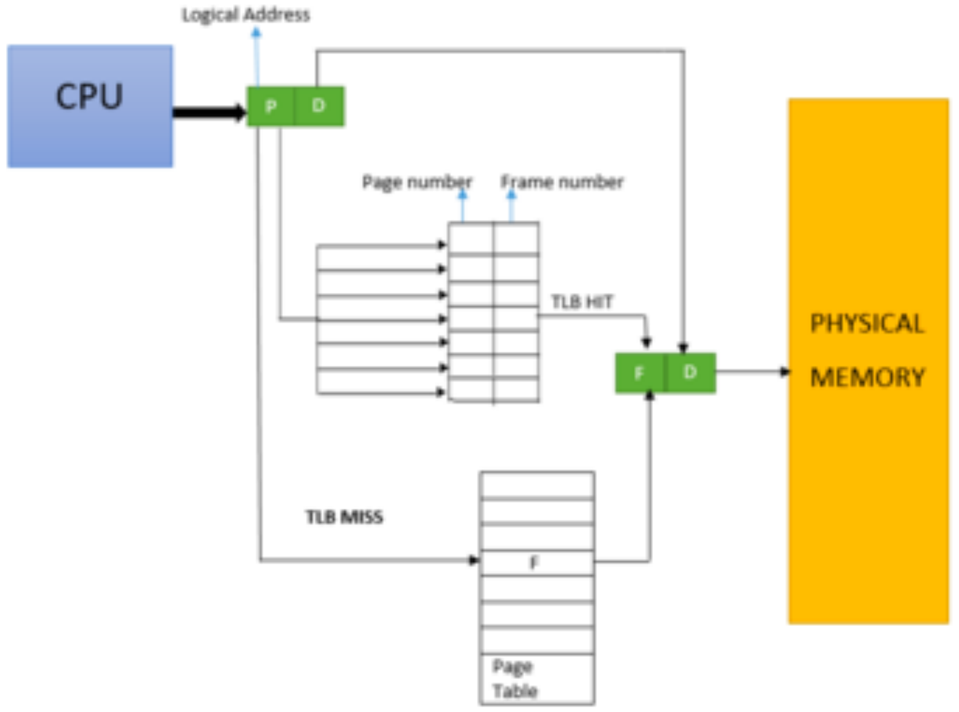


Paging & Segmentation Combined

- Disadvantages
  - Overhead managing tables
  - Time required: referencing tables
- Associative Memory/TLB (Transaction Lookup buffer)
  - Several hardware registers are allocated to job to help with the noted disadvantage
  - The concept to understand is that they might have LRU segments in a cache in which they can translate quickly

Hardware rescuing software

- TLB (Translation Lookaside Buffer) chip that translates Logical Address's quickly to Physical address, thus avoid look up calls to the Page Map table
- Inside the Memory Management Unit
- https://en.wikipedia.org/wiki/Translation_lookaside_buffer



Lets run some commands..

- ulimit -a (user limits, all current limits reported)
- vmstat (report virtual memory statistics)
- top (display processes)
- memory_pressure

- And finally, check the Activity Monitor

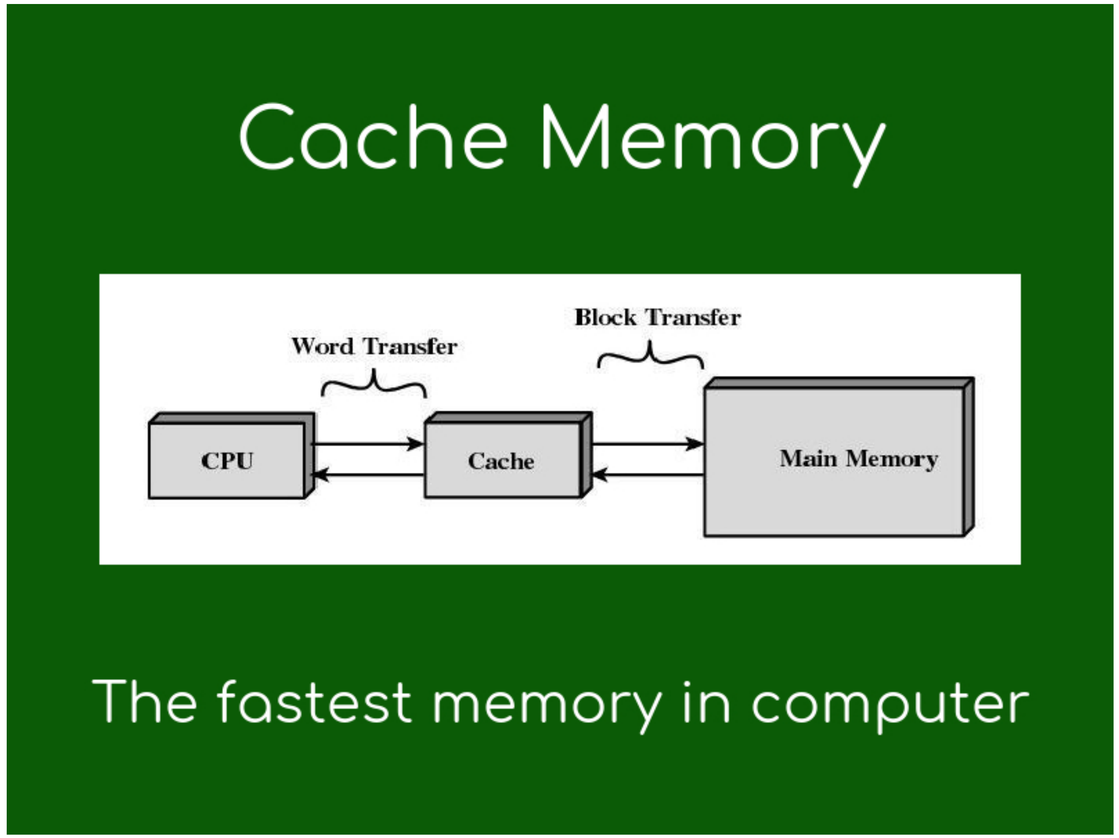
Virtual Memory

- By having more memory than just main memory we have what we call Virtual Memory
- Made possible by swapping pages in/out of memory
- Only a portion of program in memory at given moment
- Requires cooperation between Memory manager and Processor hardware

Virtual Memory

- Advantages
  - Job size: not restricted to size of main memory
  - More efficient memory use
  - Unlimited amount of multiprogramming possible
  - Code and data sharing allowed
  - Dynamic linking of program segments facilitated
- Disadvantages
  - Higher processor hardware costs
  - More overhead: handling paging interrupts
  - Increased software complexity: prevent thrashing

Cache Memory



- It stores LRU and enables having to fetch to main memory (much slower) if it can find it in cache
- The techniques of who gets to be in cache are similar to our prior discussions on the various paging algorithm’s
- Cache hit ratio is a metric for determining success
- It is thought that 80-90% of your demand for pages will actually be found in cache

Final Comparisons

Scheme	Problem Solved	Problem Created	Key Software Changes
Single-User contiguous		Job size limited to physical memory size; CPU often idle	
Fixed partitions	Idle CPU time	Internal fragmentation; job size limited to partition size	Add Processor Scheduler; add protection handler
Dynamic partitions	Internal fragmentation	External fragmentation	Algorithms to manage partitions
Relocatable dynamic partitions	External fragmentation	Compaction overhead; Job size limited to physical memory size	Algorithms for compaction
Paged	Need for compaction	Memory needed for tables; Job size limited to physical memory size; internal fragmentation returns	Algorithms to manage tables
Demand paged	Job size limited to memory size; inefficient memory use	Large number of tables; possibility of thrashing; overhead required by page interrupts; paging hardware added	Algorithm to replace pages; algorithm to search for pages in secondary storage
Segmented	Internal fragmentation	Difficulty managing variable-length segments in secondary storage; external fragmentation	Dynamic linking package; two-dimensional addressing scheme
Segmented/demand paged	Segments not loaded on demand	Table handling overhead; memory needed for page and segment tables	Three-dimensional addressing scheme

Conclusion

- Memory Manager’s main function is to:
  - Allocate memory storage, main memory, cache memory and registers
  - Deallocated memory