CMPS 405: OPERATING SYSTEMS

Memory Management

This material is based on the operating system books by Silberschatz and Stallings and other Linux and system programming books.

Objectives

- To review related material from the course CMPE 263 (Computer Architecture & Organization I). This a required self-study material.
- To introduce the terminology of main and virtual memory.
- To present a number of different methods for memory allocation and their relevant problems and available solutions.

Topics

- Motivation
- Binding of Instructions and Data to Memory
    - Logical vs. Physical Address Space
    - Overlay
    - swapping
- Contiguous Memory Allocation
    - Dynamic Storage-Allocation Problem
    - Internal and External Fragmentation
- Non-Contiguous Memory Allocation
    - Paging
    - Segmentation
- Virtual Memory
    - Demand paging
    - Page fault handling
    - Page Replacement Algorithms

CMPS 405: OPERATING SYSTEMS

Review of related material (CMPE 263)

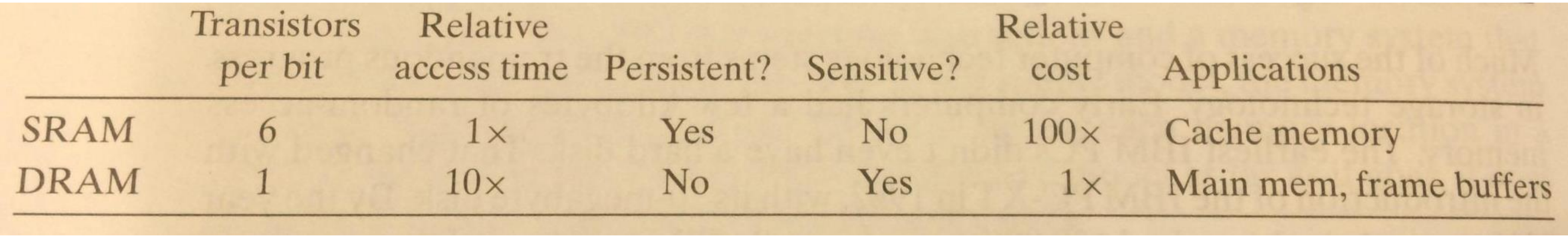
RAM and VRAM

Motivation

This material is based on the operating system books by Silberschatz and Stallings
And other Linux, system programming, and simulation books.

RAM (Random-access Memory)

- **RAM** is volatile in the sense that it looses its data if it is powered off.
- **RAM** comes in two varieties
  - **SRAM** (Static RAM)
  - **DRAM** (Dynamic RAM)



	Transistors per bit	Relative access time	Persistent?	Sensitive?	Relative cost	Applications
SRAM	6	1×	Yes	No	100×	Cache memory
DRAM	1	10×	No	Yes	1×	Main mem, frame buffers

RAM (Random-access Memory)

- **SRAM** (Static RAM)
  - Faster and significantly more expensive than DRAM.
  - Used for cache memories, both on and off the CPU chip.
  - A computer system has few megabytes of SRAM.
  - Stores data in a memory cell implemented with six transistor circuit.
  - The cell can stay indefinitely in either of two voltage configurations, states, as long as it is kept powered.

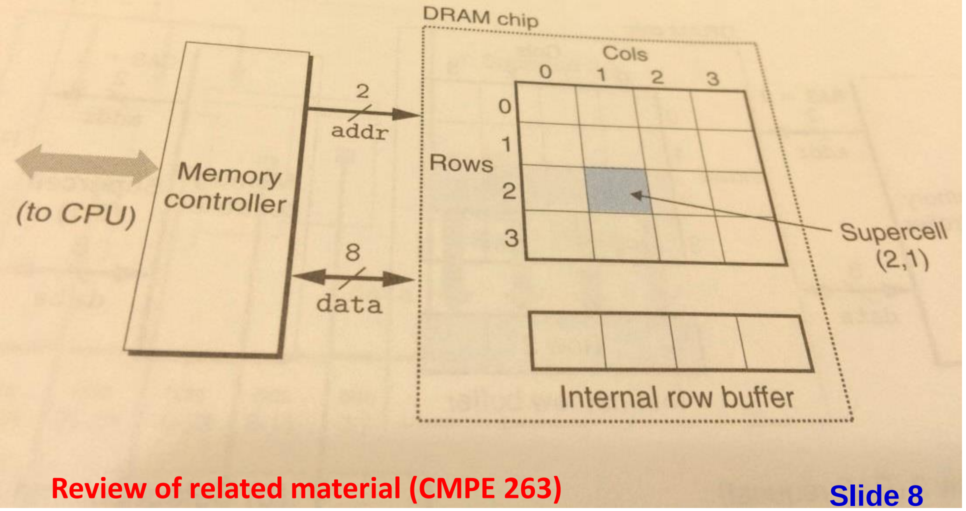
RAM (Random-access Memory)

- **DRAM** (Dynamic RAM)
  - Used for the main memory and frame buffer of a graphics system.
  - A computer system has thousands of megabytes of DRAM.
  - Stores each bit as charge on a capacitor of size around 30 femtofarads, $30\text{x}10^{-15}$ farads.
  - The cell consists of a capacitor and a single access transistor.
  - Very sensitive to light.
  - Current leakage causes a DRAM to lose its charge within 10 to 100 ms.
    - That is not a problem because computers clock cycle times is in nanoseconds. In some implementations, the refresh happens every 64 ns.
    - The memory system must periodically refresh every bit of memory by reading it out and then rewriting it.
    - Words error-correcting codes can be used to detect and correct any single erroneous bit with a word.

RAM (Random-access Memory)

Conventional DRAM

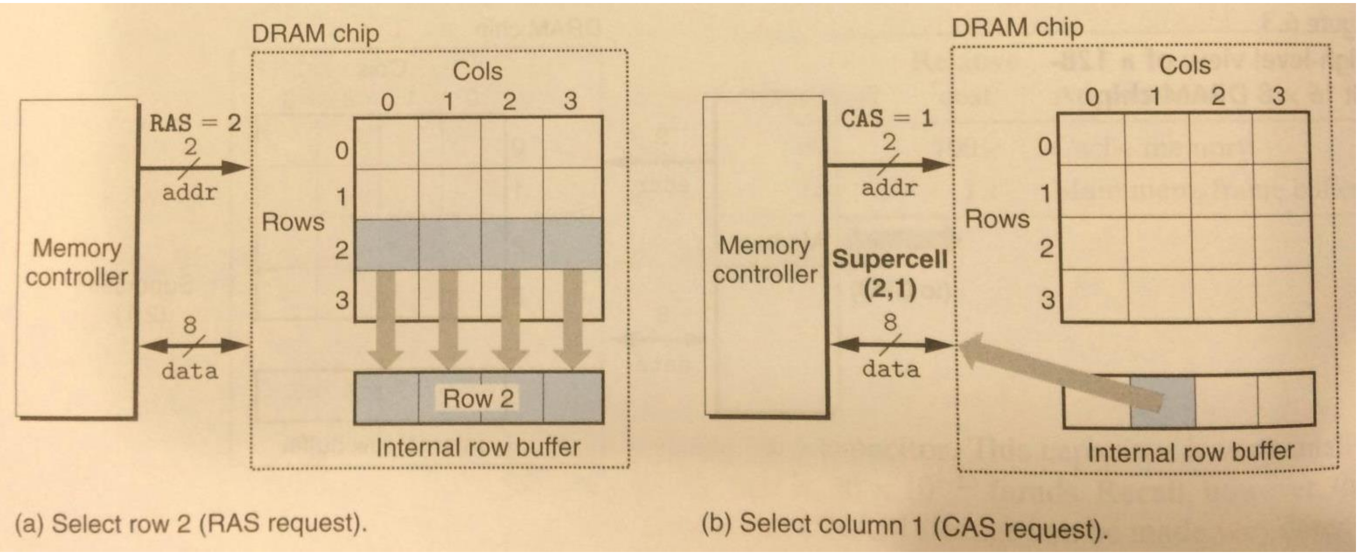
- The cells are partitioned into d supercells each of *w* DRAM cells.
- A *dw* DRAM stores a total of *dw* bits.
- The supercells are organized as an *rc* rectangular array, where *rc=d*.
- Each supercell has an address of (*i,j*).
- Information flows in and out of the chip via external connector called pins.
- Each pin carries one bit signal.
- Address pins and control pins are also connected to the chip.
- Each chip is connected to some circuitry, known as memory controller that can transfer *w* bits at a time from and to each DRAM chip.



Review of related material (CMPE 263)

RAM (Random-access Memory)

- Reading supercell (*2*,*1*) from a *16*x*8* DRAM
  - Memory controller sends address row *2* using the address pins
  - DRAM copies the contents of row *2* into an internal row buffer.
  - Memory controller sends column address *1* using the address pins
  - DRAM copies the *8* bits of cell *1* from the internal buffer and sends it to the memory controller.
- Organizing supercells in rectangular form reduces the needed number of address pins. But, it has the disadvantage that the address has to be sent in two distinct steps.



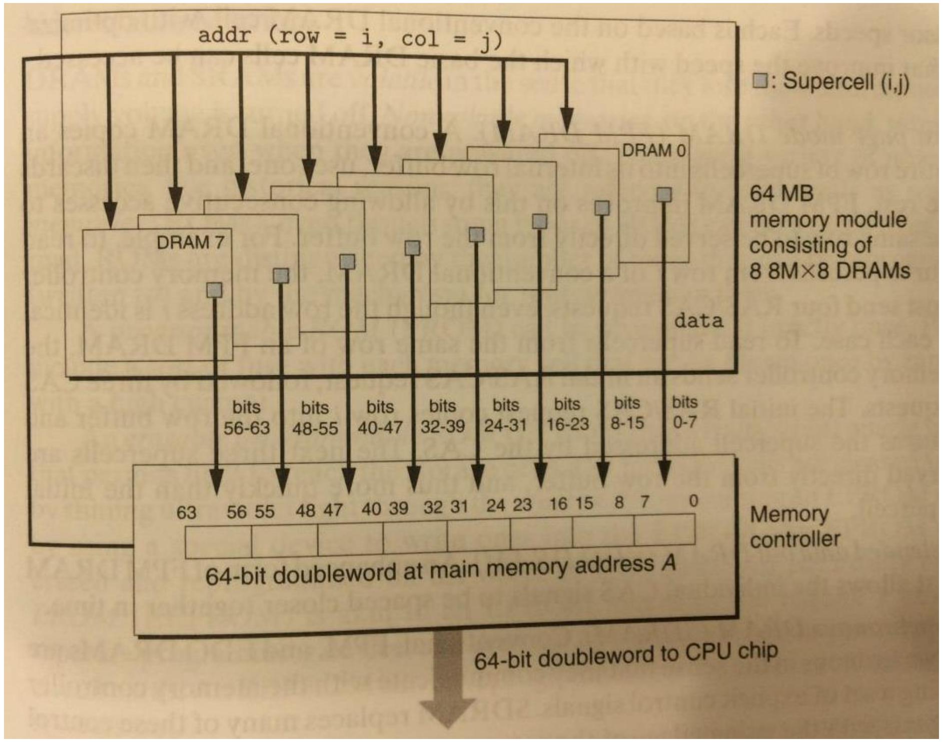
(a) Select row 2 (RAS request).

(b) Select column 1 (CAS request).

RAM (Random-access Memory)

- DRAM chips are packaged in memory modules that plug into expansion slots on the main system board (motherboard).
- Common packages include the *168*-pin dual inline memory module (DIMM), which can transfer data from and to memory controller in *64*-bit chunks. And *72*-pin single inline memory module, which can transfer data in *32*-bit chunks.

- Main memory can be aggregated by connecting multiple memory modules to the memory controller.
  - In this case, when the controller receives a an address *A*, the controller selects the module *k* that contains *A*, converts *A* to its (*i*,*j*) form, and sends (*i*,*j*) to module *k*.



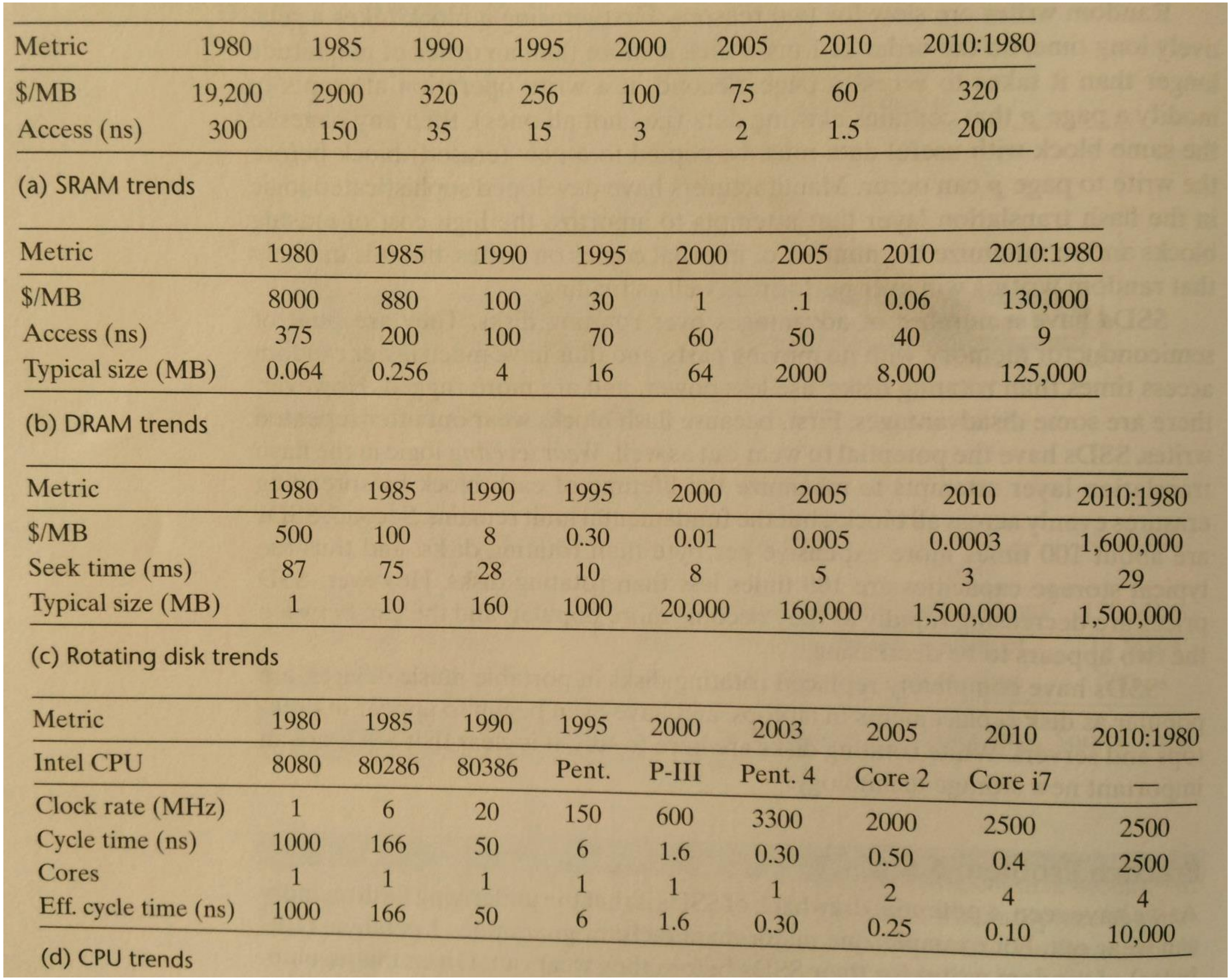
RAM (Random-access Memory)

- **Types of DRAM**
  - Fast page mode DRAM (**FPM DRAM**): reading *k* supercells from row *i* is achieved by sending one RAS/CAS request followed by *k-1* CAS requests.
  - Extended data out DRAM (**EDO DRAM**): individual CAS signals are spaced closer together in time.
  - Synchronous DRAM (**SDRAM**): communicates with the memory controller using the rising edge of the external clock signal the derives the memory controller instead of using a set of explicit control signals. Therefore, is faster.
  - Double Data-rate Synchronous DRAM (**DDR SDRAM**): doubles the speed by using both clock edges and control signals. A small prefetch buffer is used to increase effictive bandwidth: 2 bits (DDR), 4 bits (DDR2), 8 bits (DDR3), … .
  - Rambus DRAM (**RDRAM**): uses an alternative technology with higher maximum bandwidth than DDR SDRAM.
  - Video RAM (**VRAM**): used in the frame buffers of graphics systems and is similar to FPM DRAM.

ROM (Read-only Memory)

- Historically called so even though they can be written.
- **ROM**s are nonvolatile memories, retaining their information even wen they are powered off.
- **There are varieties of ROMs:**
  - Programmable ROM (**PROM**): can be programmed exactly once.
  - Erasable programmable ROM (**EPROM**): can be programmed using a special device to write ones to EPROM. They can be erased and programmed in the order of *1000* times.
  - Electrical erasable PROM (**EEPROM**): does not require a special device instead can be programmed and erased in-place on printed circuit cards. Can be programmed in the order of 10^5 times.
  - **Flash memory**: based on EEPROM. The solid state disk (**SSD**) is a newer form of flash-based disk.
- Programs stored in **ROM** devices are often referred to as ***firmware***.

Comparison



Metric	1980	1985	1990	1995	2000	2005	2010	2010:1980
$/MB	19,200	2900	320	256	100	75	60	320
Access (ns)	300	150	35	15	3	2	1.5	200

(a) SRAM trends

Metric	1980	1985	1990	1995	2000	2005	2010	2010:1980
$/MB	8000	880	100	30	1	.1	0.06	130,000
Access (ns)	375	200	100	70	60	50	40	9
Typical size (MB)	0.064	0.256	4	16	64	2000	8,000	125,000

(b) DRAM trends

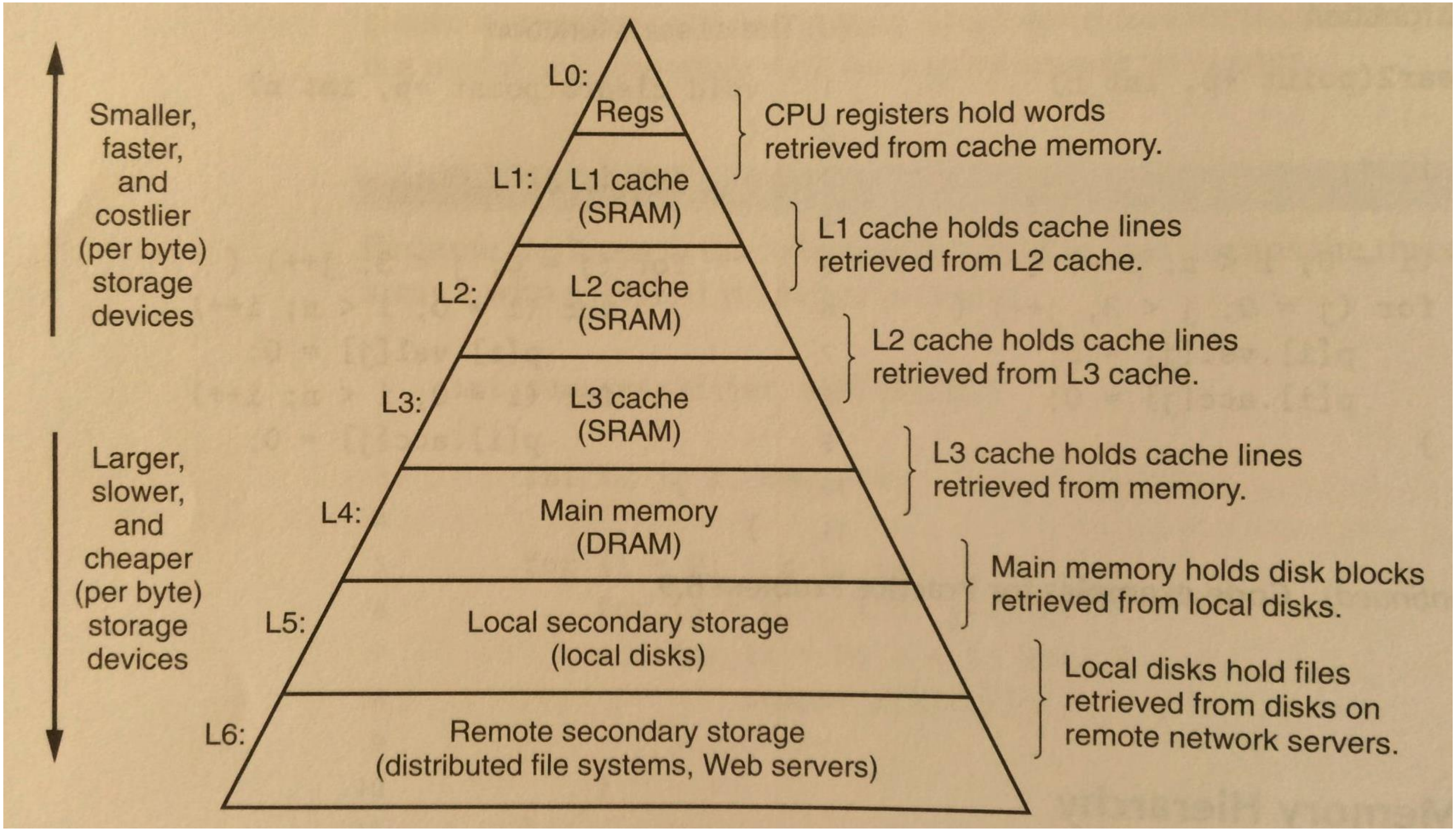
Metric	1980	1985	1990	1995	2000	2005	2010	2010:1980
$/MB	500	100	8	0.30	0.01	0.005	0.0003	1,600,000
Seek time (ms)	87	75	28	10	8	5	3	29
Typical size (MB)	1	10	160	1000	20,000	160,000	1,500,000	1,500,000

(c) Rotating disk trends

Metric	1980	1985	1990	1995	2000	2003	2005	2010	2010:1980
Intel CPU	8080	80286	80386	Pent.	P-III	Pent. 4	Core 2	Core i7	—
Clock rate (MHz)	1	6	20	150	600	3300	2000	2500	2500
Cycle time (ns)	1000	166	50	6	1.6	0.30	0.50	0.4	2500
Cores	1	1	1	1	1	1	2	4	4
Eff. cycle time (ns)	1000	166	50	6	1.6	0.30	0.25	0.10	10,000

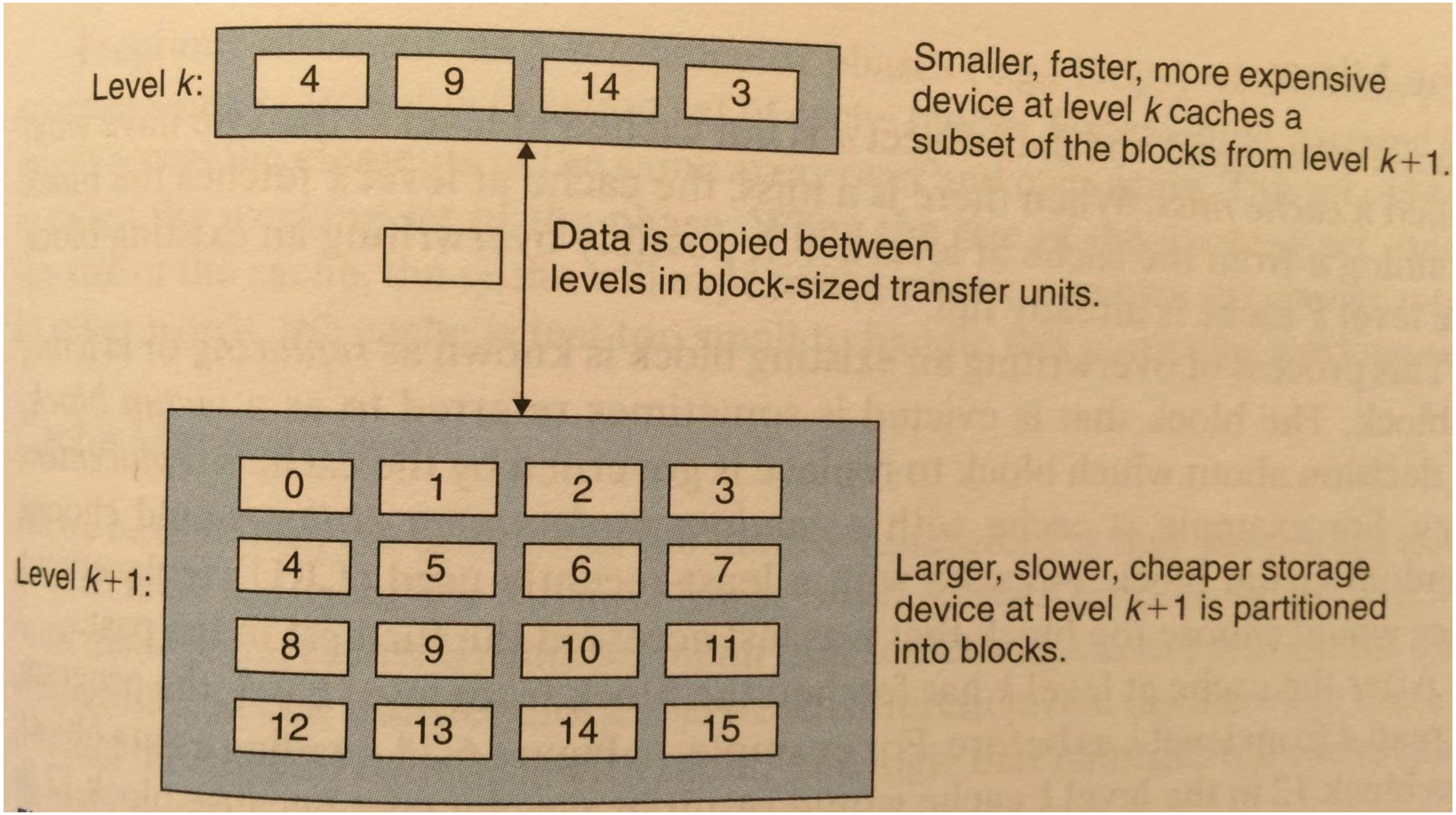
(d) CPU trends

Memory Hierarchies


Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0:
Regs
L1:
L1 cache (SRAM)
L2:
L2 cache (SRAM)
L3:
L3 cache (SRAM)
L4:
Main memory (DRAM)
L5:
Local secondary storage (local disks)
L6:
Remote secondary storage (distributed file systems, Web servers)
CPU registers hold words retrieved from cache memory.
L1 cache holds cache lines retrieved from L2 cache.
L2 cache holds cache lines retrieved from L3 cache.
L3 cache holds cache lines retrieved from memory.
Main memory holds disk blocks retrieved from local disks.
Local disks hold files retrieved from disks on remote network servers.

Review of related material (CMPE 263)

Caching Concept

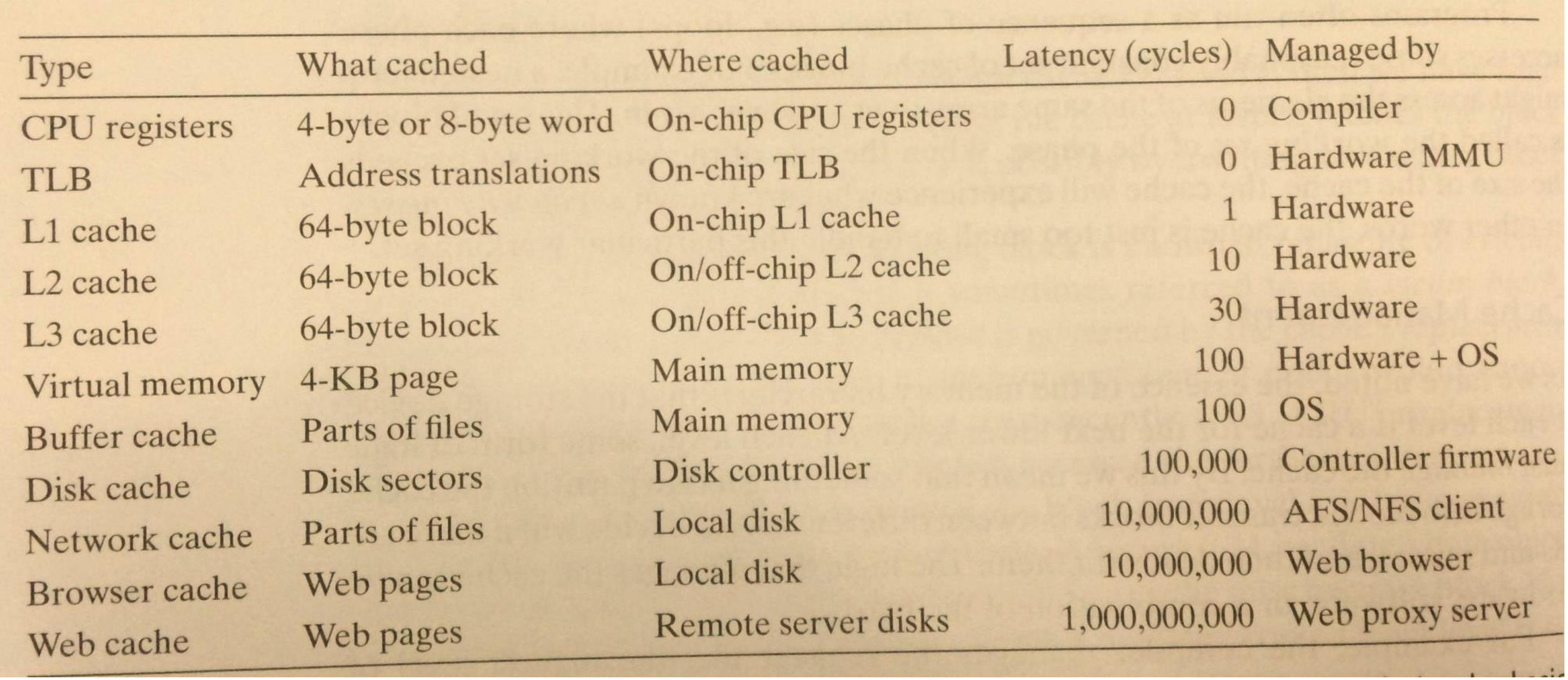


Smaller, faster, more expensive device at level k caches a subset of the blocks from level k+1.

Data is copied between levels in block-sized transfer units.

Larger, slower, cheaper storage device at level k+1 is partitioned into blocks.

Examples of Caching



Type	What cached	Where cached	Latency (cycles)	Managed by
CPU registers	4-byte or 8-byte word	On-chip CPU registers	0	Compiler
TLB	Address translations	On-chip TLB	0	Hardware MMU
L1 cache	64-byte block	On-chip L1 cache	1	Hardware
L2 cache	64-byte block	On/off-chip L2 cache	10	Hardware
L3 cache	64-byte block	On/off-chip L3 cache	30	Hardware
Virtual memory	4-KB page	Main memory	100	Hardware + OS
Buffer cache	Parts of files	Main memory	100	OS
Disk cache	Disk sectors	Disk controller	100,000	Controller firmware
Network cache	Parts of files	Local disk	10,000,000	AFS/NFS client
Browser cache	Web pages	Local disk	10,000,000	Web browser
Web cache	Web pages	Remote server disks	1,000,000,000	Web proxy server

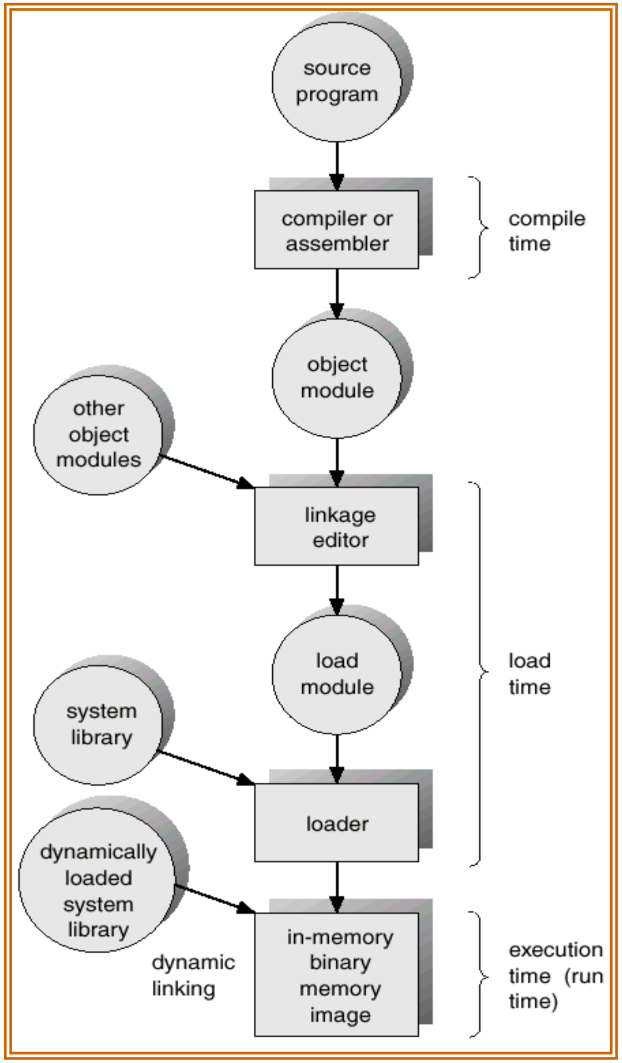
Review of related material (CMPE 263)

End of
review material

Binding of Instructions and Data to Memory

Address binding of instructions and data to memory addresses can happen at three different stages

- **Compile time**: If memory location known a priori, *absolute code* can be generated; must recompile code if starting location changes
- **Load time**: Must generate *relocatable code* if memory location is not known at compile time
- **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., *base* and *limit registers*).



Logical vs. Physical Address Space

- The concept of a logical *address space* that is bound to a separate *physical address space* is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as *virtual address*
  - **Physical address** – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme.

Memory-Management Unit (MMU)

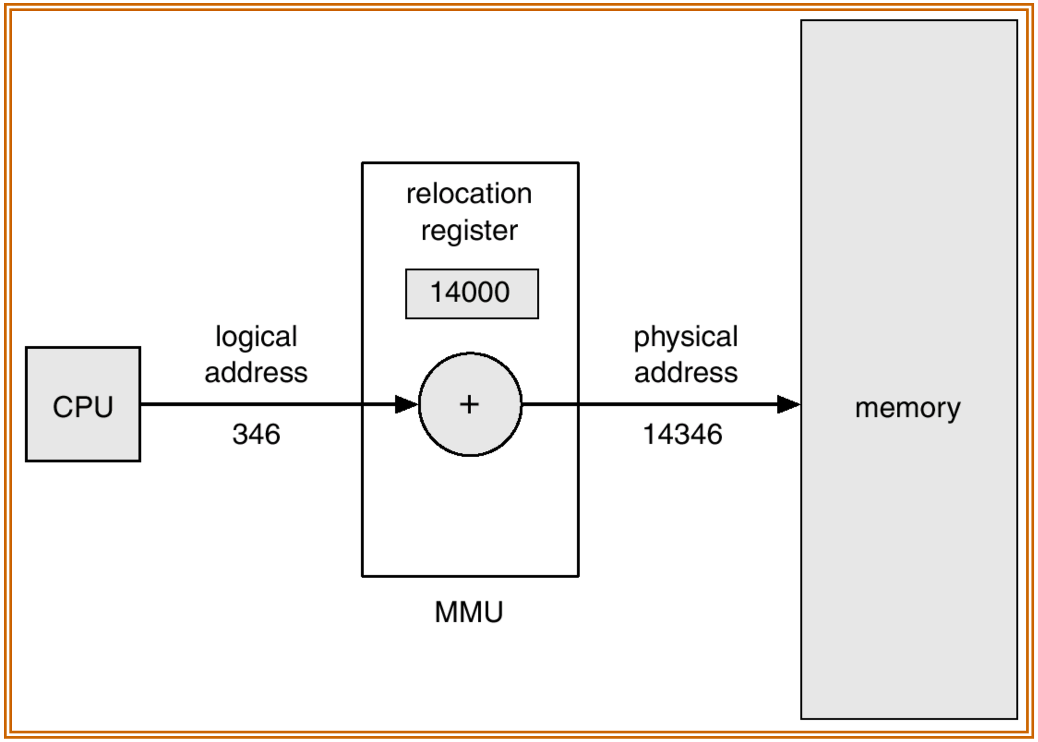
- MMU is a computer hardware component that is usually integrated into the processor, although in some systems it occupies a separate IC (integrated circuit) chip.

- MMU is responsible for all aspects of memory management.

- A processor with a MMU that provides virtual memory has an on-chip cache of "translations". Each "translation record/entry" tells the CPU the mapping of one virtual address to one physical address.

Dynamic relocation using a relocation register

- In this scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.



Dynamic Loading

- Routine is not loaded until it is called
  - Better memory-space utilization; unused routine is never loaded.
  - Useful when large amounts of code are needed to handle infrequently occurring cases.
  - No special support from the operating system is required implemented through program design.

Dynamic Linking

- Linking postponed until execution time.
  - Small piece of code, *stub*, used to locate the appropriate memory-resident library routine.
  - Stub replaces itself with the address of the routine, and executes the routine.
  - Operating system needed to check if routine is in processes’ memory address.
  - Dynamic linking is particularly useful for libraries

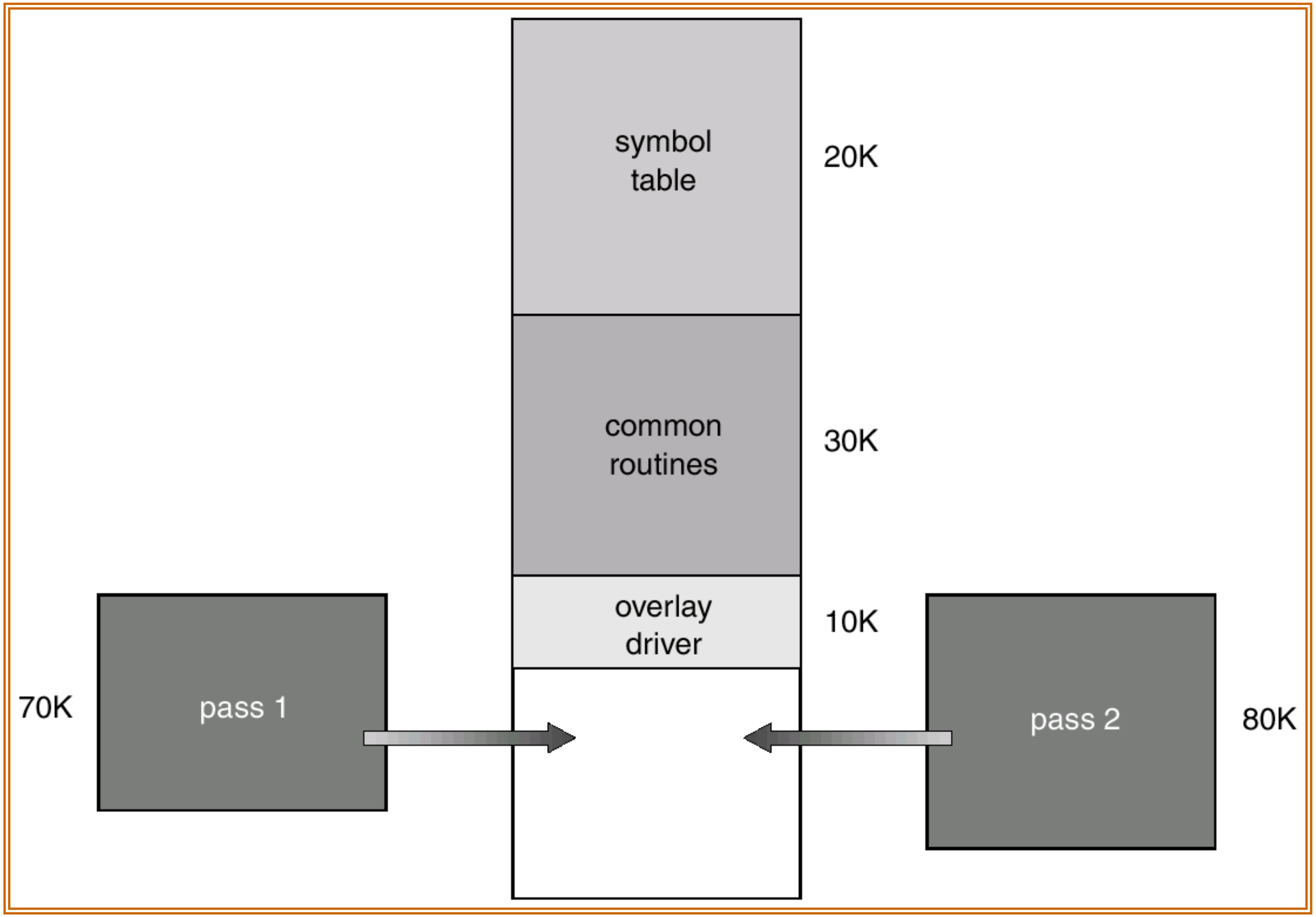
Overlays

- Keep in memory only those instructions and data that are needed at any given time.

  - Needed when process is larger than amount of memory allocated to it.

  - Implemented by user, no special support needed from operating system, programming design of overlay structure is complex.

Overlays for a Two-Pass Assembler



Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution

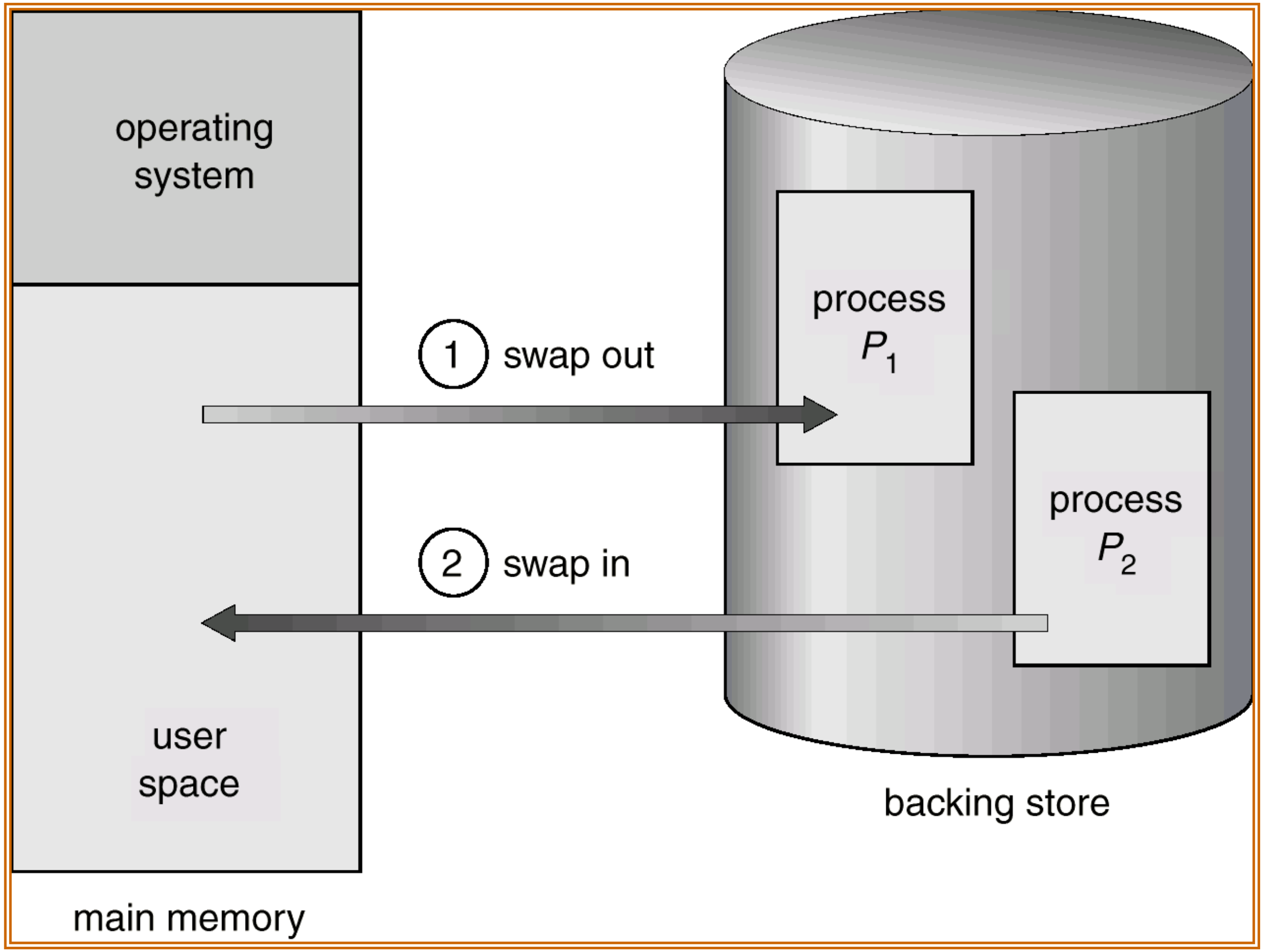
- **Backing store** – fast disk large enough to accommodate copies of all **memory images** for all users; must provide direct access to these memory images

- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed

- Major part of **swap time** is transfer time; total transfer time is directly proportional to the amount of memory swapped

- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)

Schematic View of Swapping

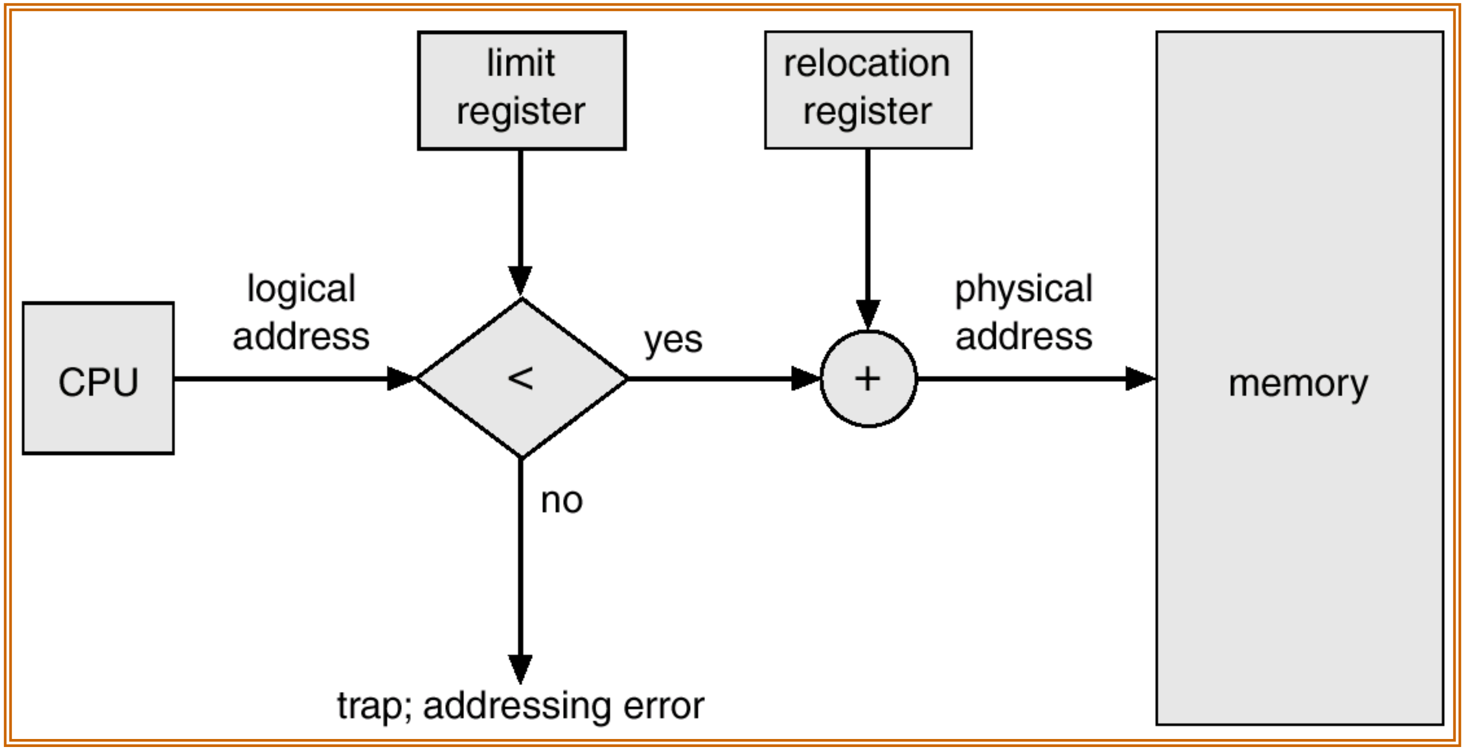


Contiguous Allocation

- Main memory is usually divided into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector.
  - User processes then held in high memory.

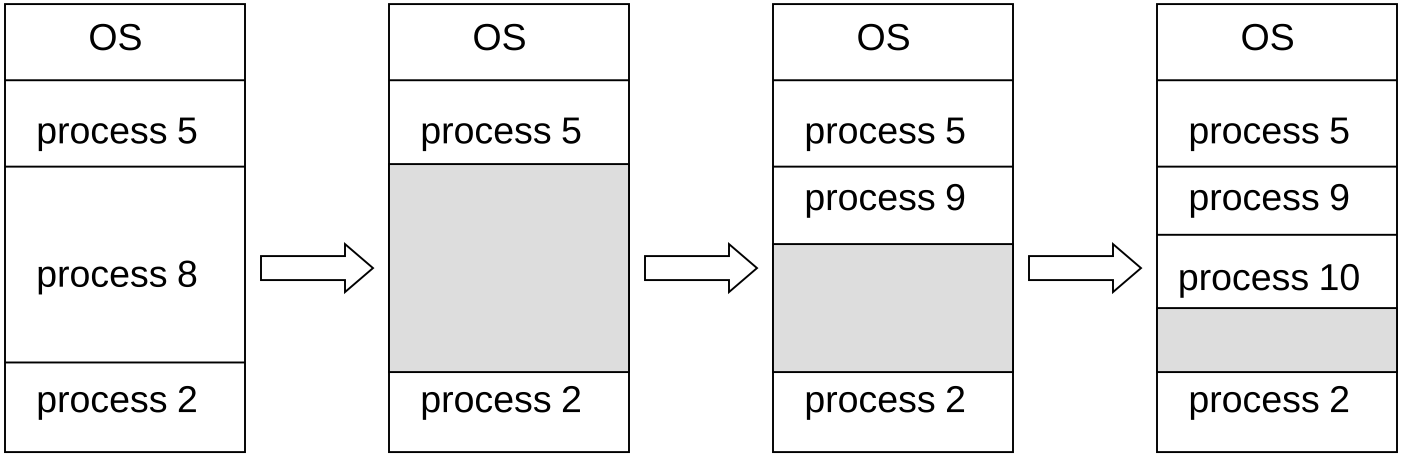
- Single-partition allocation
  - Relocation-register scheme used to protect user processes from each other, and from changing operating-system code and data
  - **Relocation register** contains value of smallest physical address; **limit register** contains range of logical addresses – each logical address must be less than the limit register.

Hardware Support for Relocation and Limit Registers



Contiguous Allocation

- Multiple-partition allocation
  - ***Hole*** – block of available memory; holes of various size are scattered throughout memory
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Operating system maintains information about:
    a) allocated partitions   b) free partitions (hole)



Dynamic Storage-Allocation Problem

How to satisfy a request of size *n* from a list of free holes

- **First-fit**: Allocate the *first* hole that is big enough

- **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.

- **Worst-fit**: Allocate the *largest* hole; must also search entire list. Produces the largest leftover hole.

Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous.

- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.

- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block.
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time.

Motivation to non-contiguous memory allocation

- The aim of this lecture is to present memory-management schemes allowing the physical address space of a process to be non-contiguous.

- This aim is achieved by memory allocation schemes such as paging and segmentation.

- Both schemes require hardware support.

- Paging and segmentation can also be used together forming another memory allocation scheme.

Paging

How does it work?

- Pre-memory allocation:
  - Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, usually between 512 bytes and 8,192 bytes depending on the computer architecture)
  - Divide logical memory into blocks of same size called **pages**
  - Keep track of all free frames
- Memory allocation:
  - To run a program of size ***n*** pages, need to find *n* free frames and load program from backing store to main memory.
  - Set up a page table to translate logical to physical addresses

****Some CPUs and Kernels support:**

- **Large page sizes say 16 MB or**
  - **Multiple page sizes: For example Solaris uses page sizes of 8K and 4 MB. Or**
  - **Variable on-the-fly page size.**

+s vs. -s

- **Drawback**
  - Internal fragmentation: the last frame may not be completely full.
    - Example: if page size is 2,048 bytes, a process of 72,766 bytes would need 35 pages plus 1,086 bytes. The process is allocated 36 frames in which frame number 36 has internal fragmentation of 962 bytes (2048 – 1086).
    - What is the worst case of such internal fragmentation?
    - What is the average case of such internal fragmentation?
  - Separates between the user's logical view of memory and the actual physical memory.

- **Strength**
  - overcomes the problem of fitting memory chunks of different sizes onto the backing store.
    - This problem causes fragmentation problems with the backing store where compaction is impossible because accessing backing store is much slower than main memory.
  - No external fragmentation.

Address Translation Scheme

- Address generated by CPU (virtual address) is divided into two parts:

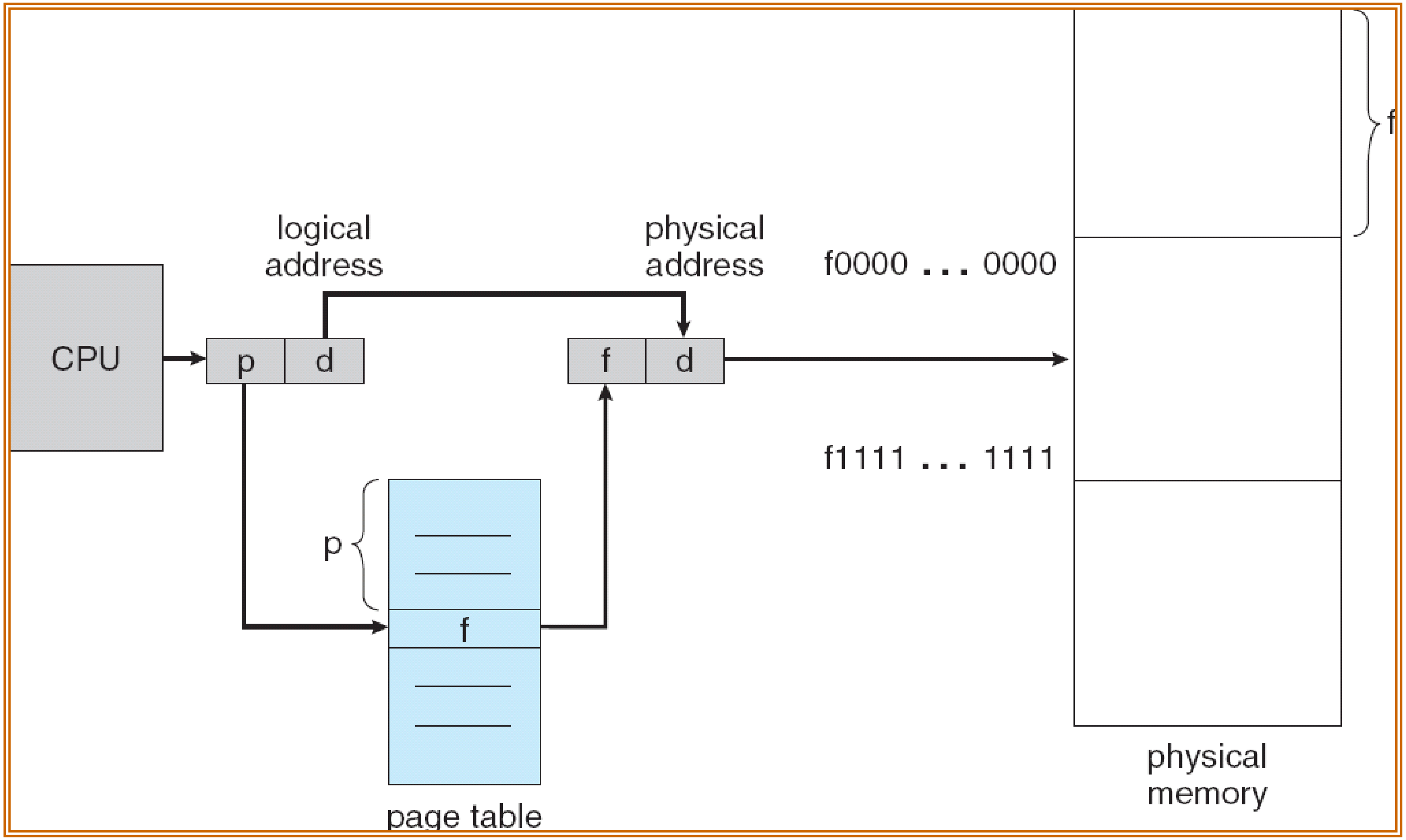
  - **Page number (*p*)** – used as an index into a *page table* which contains base address of each page in physical memory.

  - **Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit.

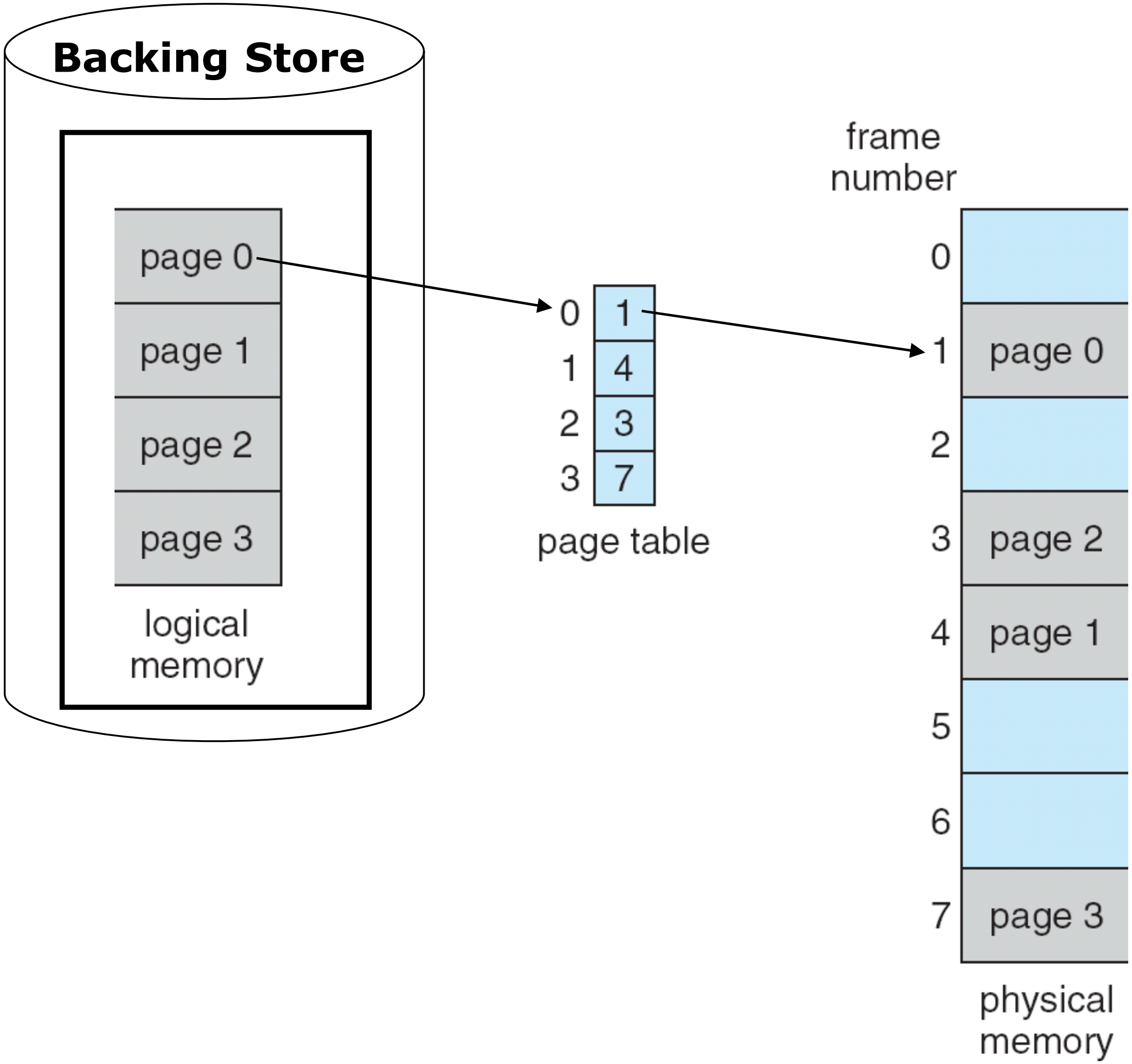
page number	**page offset**
p	***d***
m* – *n *bits*	***n*** *bits*

A virtual address for given logical address space **2^m** *and page size* **2^n**

Paging Hardware



Paging Model of Logical and Physical Memory



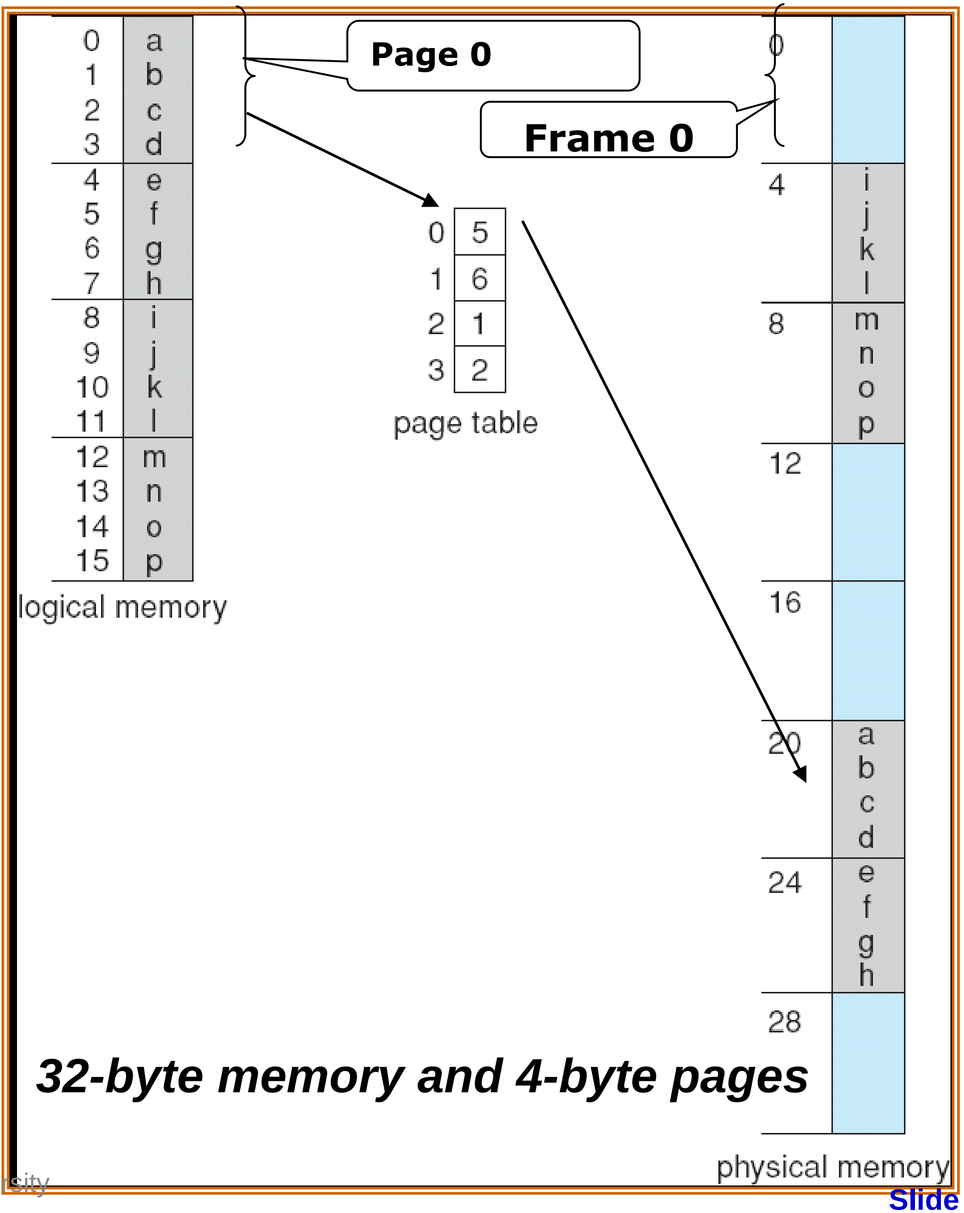
Paging Example

Logical address 0:
page 0 offset 0.
Mapped to physical address:
20 (5X4 + 0)

Logical address 3:
page 0 offset 3.
Mapped to physical address:
23 (5X4 + 3)

Logical address 4:
page 1 offset 0.
Mapped to physical address:
24 (6X4 + 0)

Logical address 13:
page 3 offset 1.
Mapped to physical address:
9 (2X4 + 1)



Page 0

Frame 0

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p

logical memory

0	5
1	6
2	1
3	2

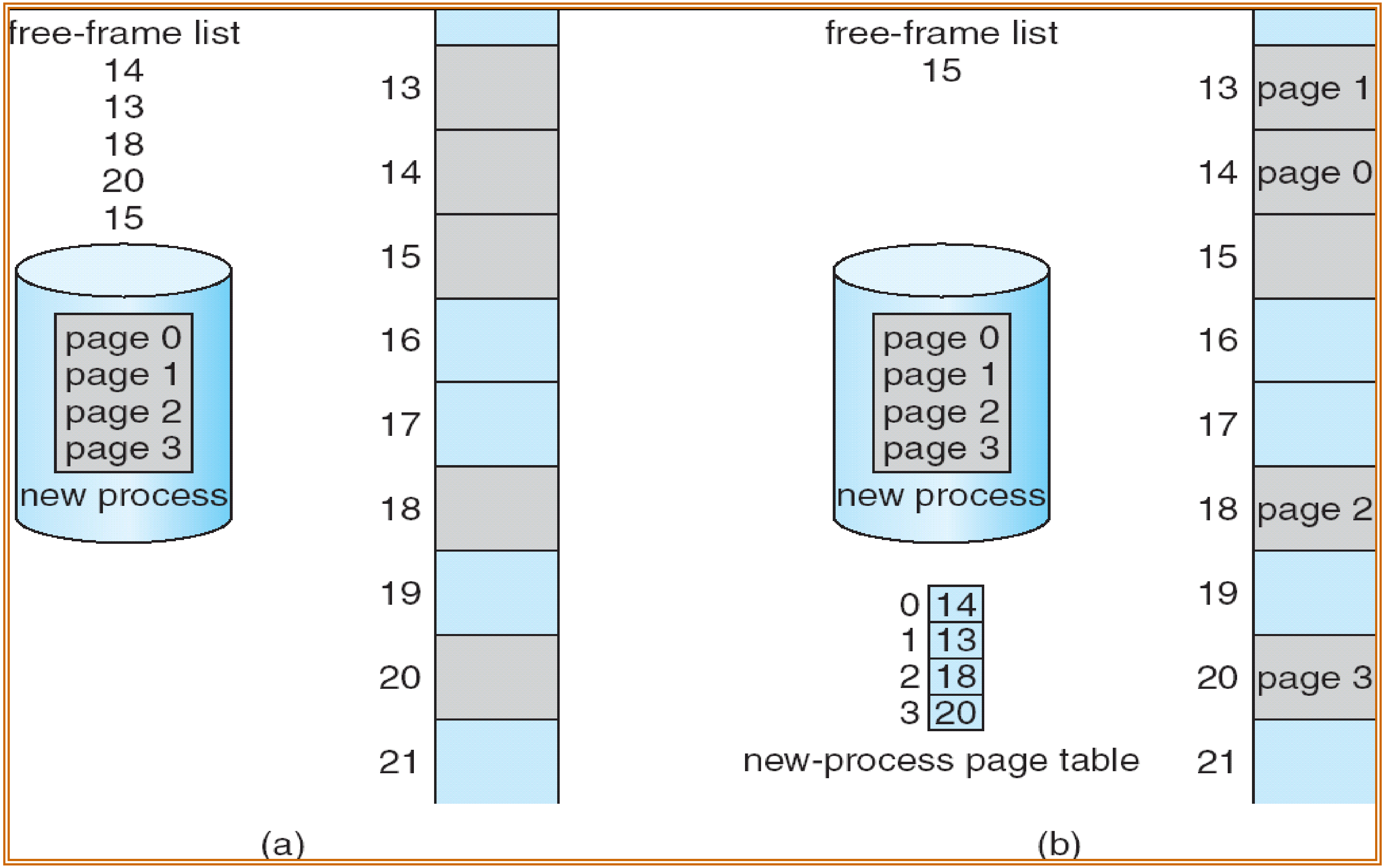
page table

0	
4	i j k l
8	m n o p
12	
16	
20	a b c d
24	e f g h
28	

physical memory

32-byte memory and 4-byte pages

Free Frames



0	14
1	13
2	18
3	20

new-process page table

Before allocation

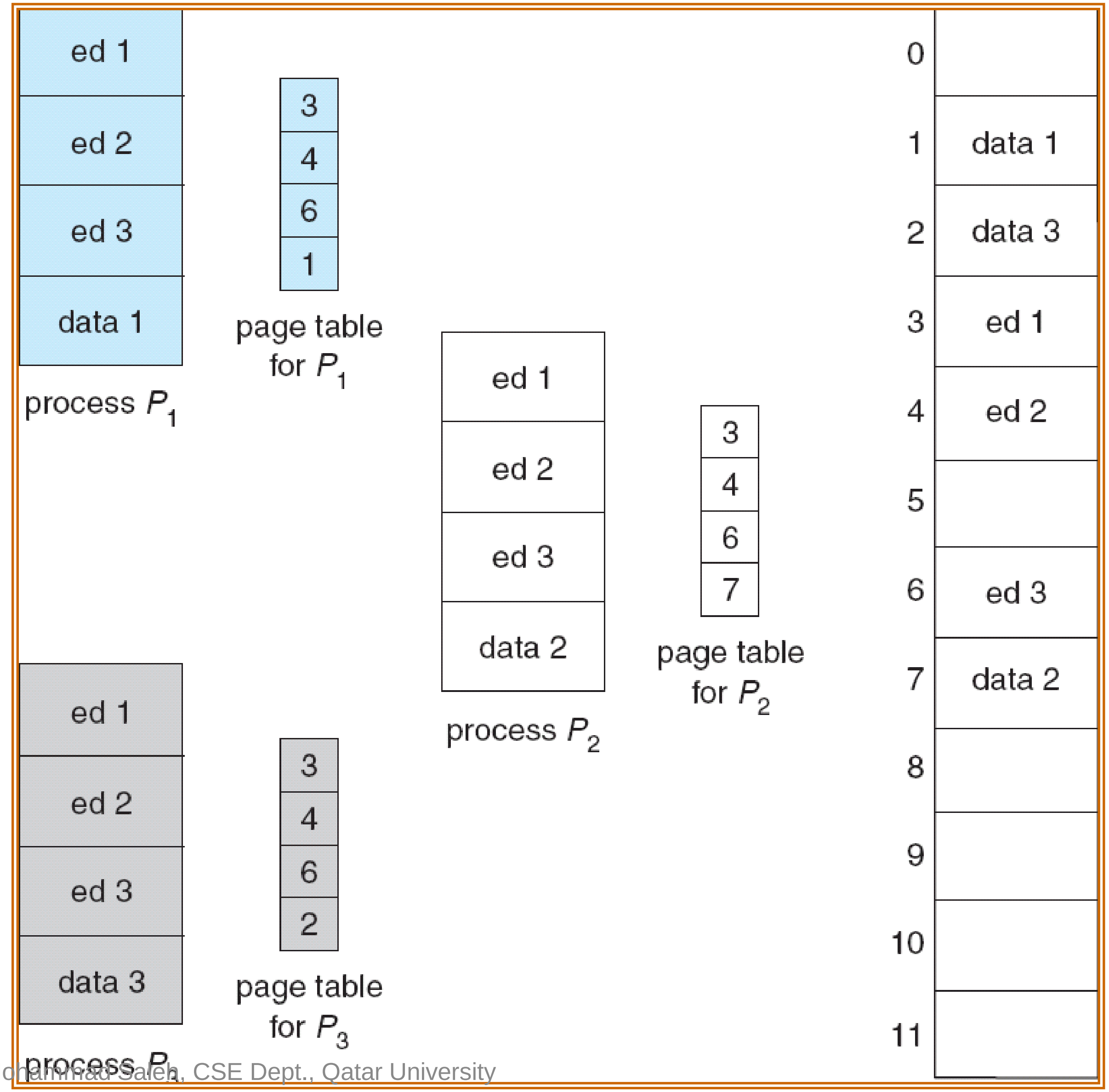
After allocation

Shared Pages

- **Shared code**
  - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
  - Shared code must appear in same location in the logical address space of all processes

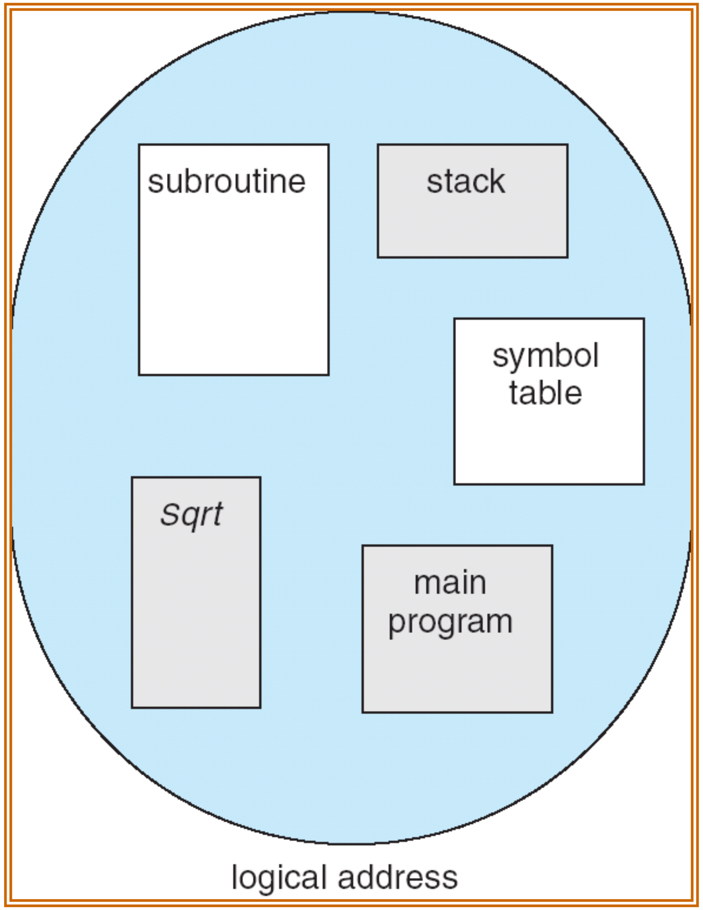
- **Private code and data**
  - Each process keeps a separate copy of the code and data
  - The pages for the private code and data can appear anywhere in the logical address space

Shared Pages Example



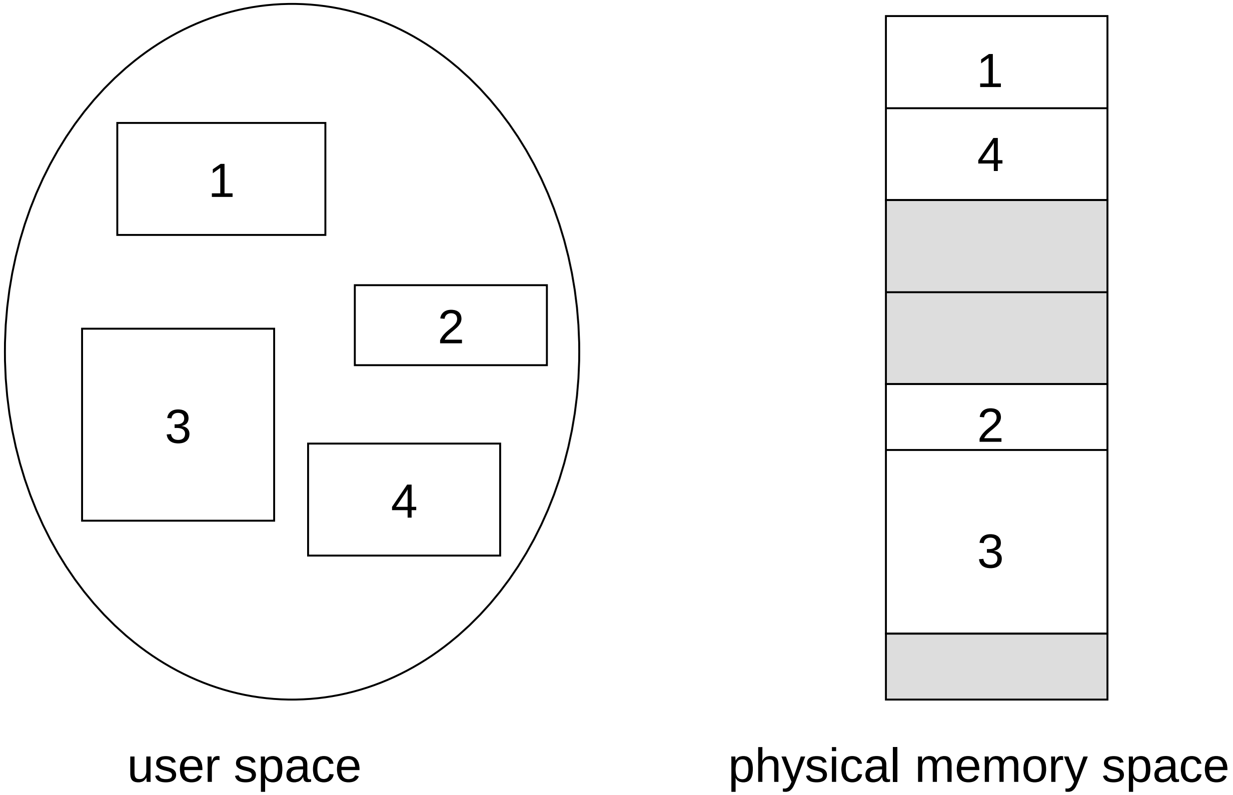
Segmentation

- Memory-management scheme that supports user view of memory
- A program is a collection of segments. A segment is a logical unit such as:
  - main program,
  - procedure,
  - function,
  - method,
  - object,
  - local variables, global variables,
  - common block,
  - stack,
  - symbol table, arrays



User's view of the program

Logical View of Segmentation



Segmentation Architecture

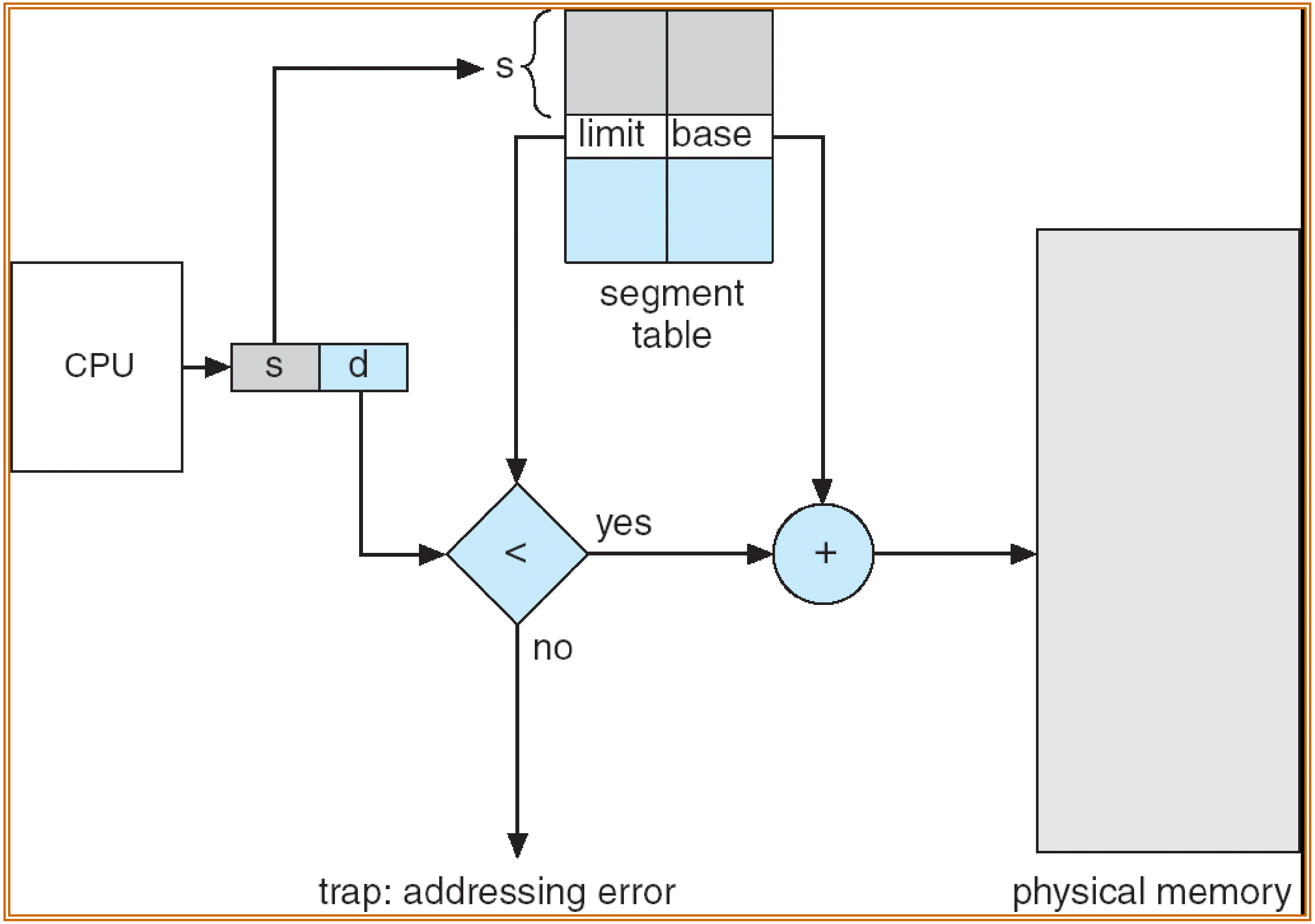
- Logical address consists of a two tuple:

  <segment-number, offset>,

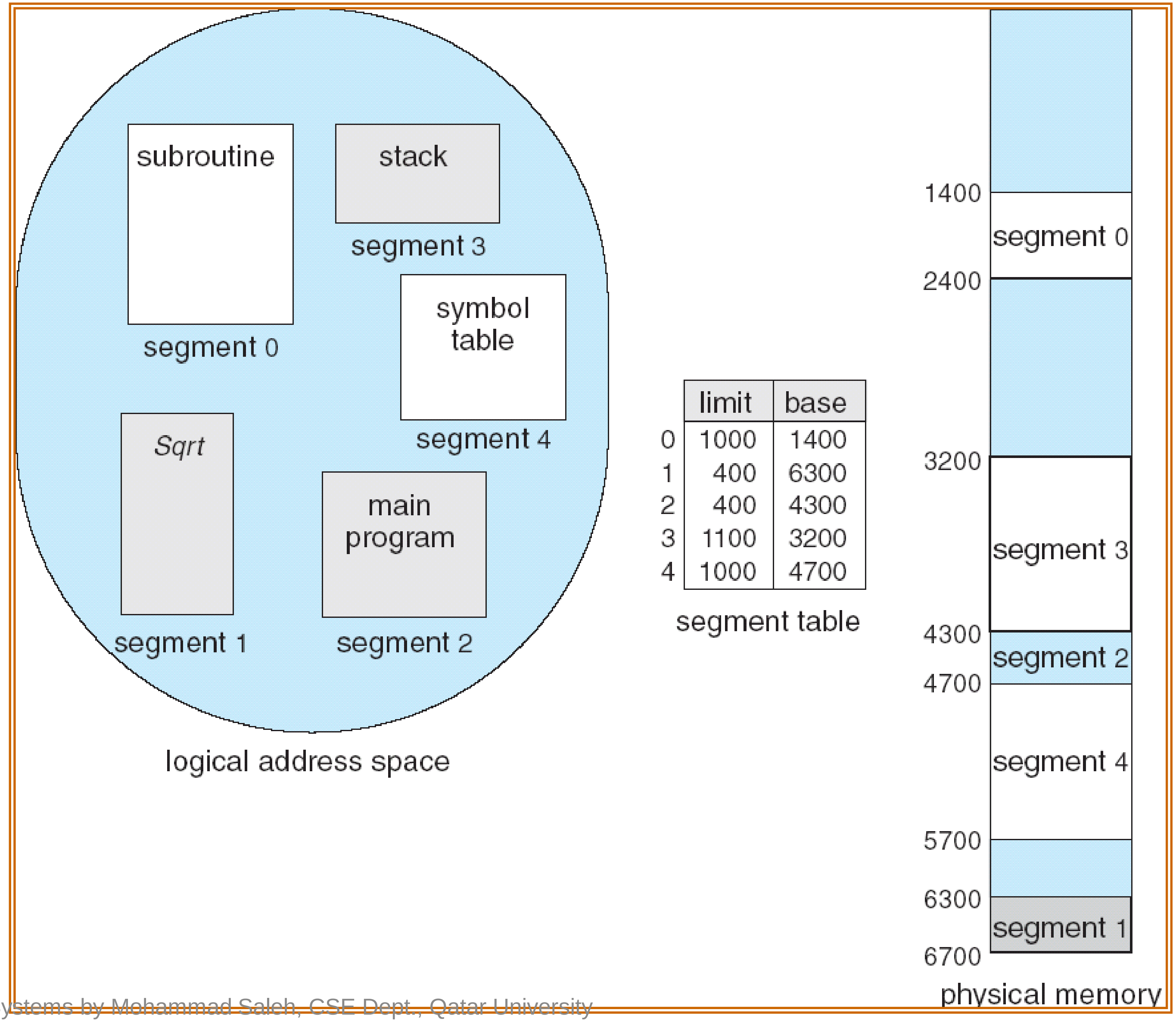
- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program;

  segment number ***s*** is legal if ***s*** < **STLR**

Segmentation Hardware



Example of Segmentation

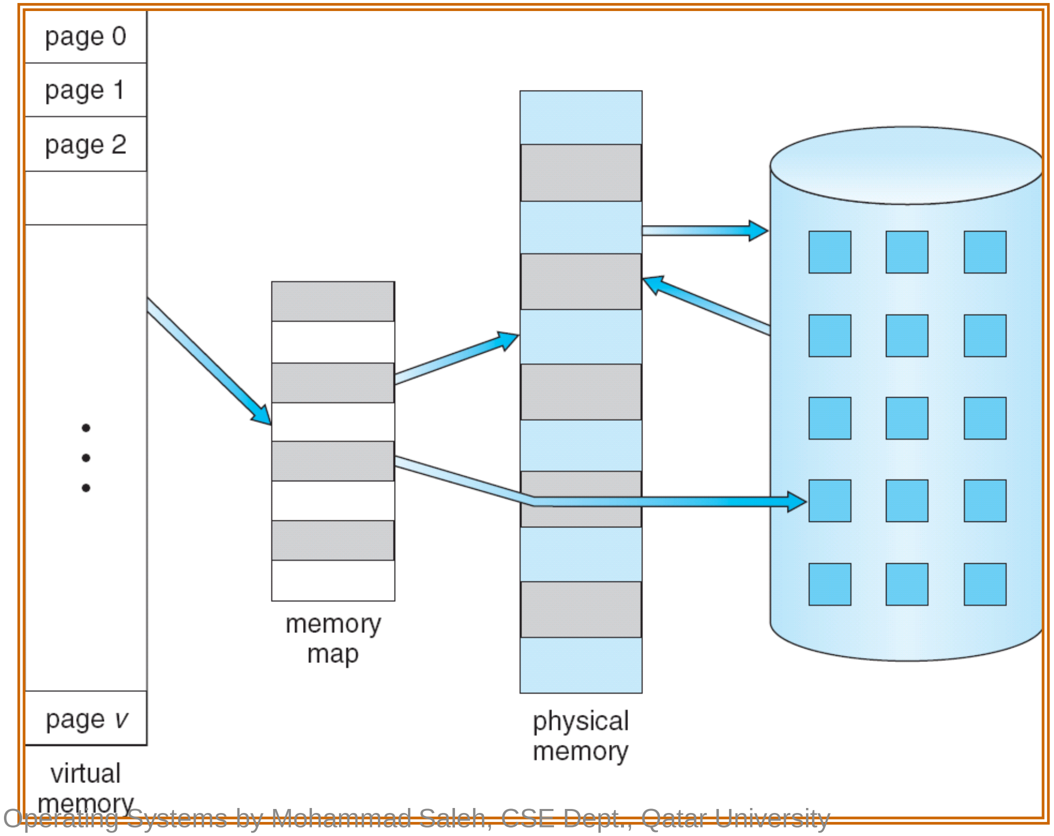


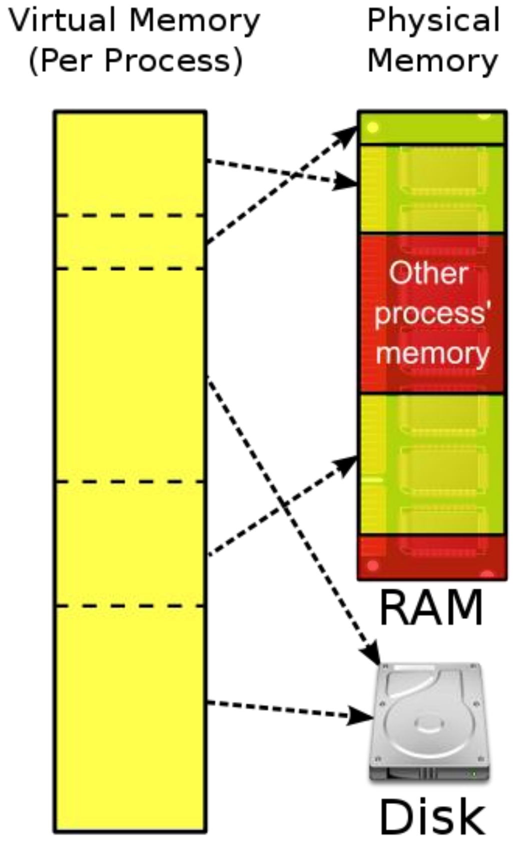
	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table

Virtual Memory on WIKIPEDIA

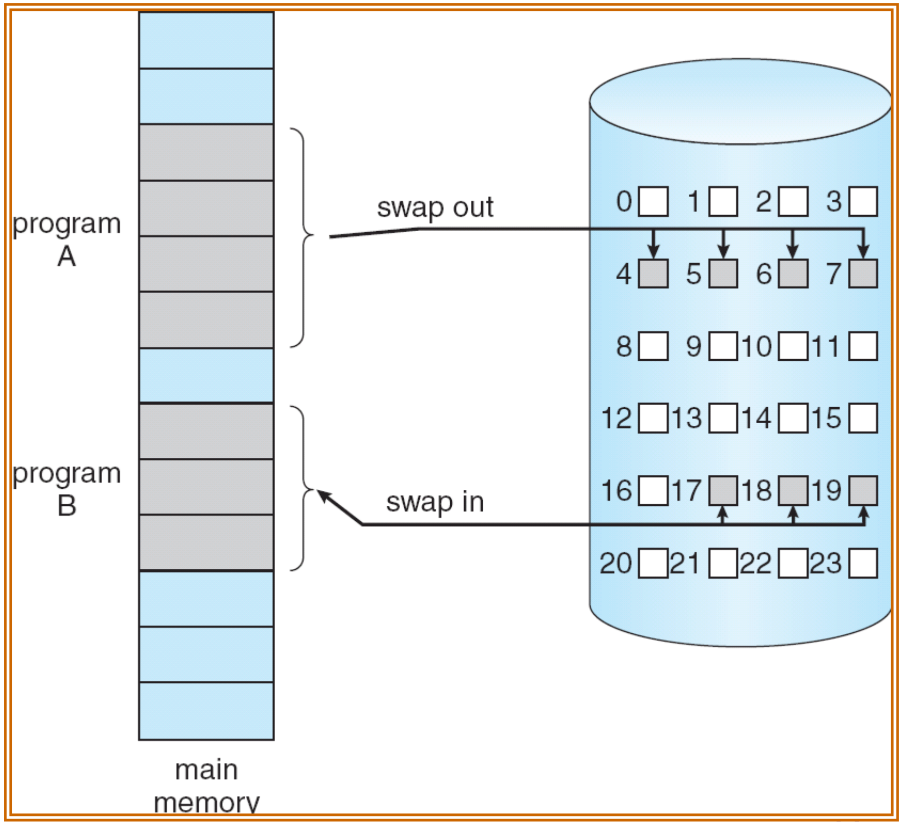
- The program thinks it has a large range of contiguous addresses; but in reality the parts it is currently using are scattered around RAM, and the inactive parts are saved in a disk file





Background

- **Since** only part of the program needs to be in memory for execution:
  - Logical address space can therefore be much larger than physical address space
  - Allows for more efficient process creation
- Virtual memory implementations:
  - Demand paging
  - Demand segmentation

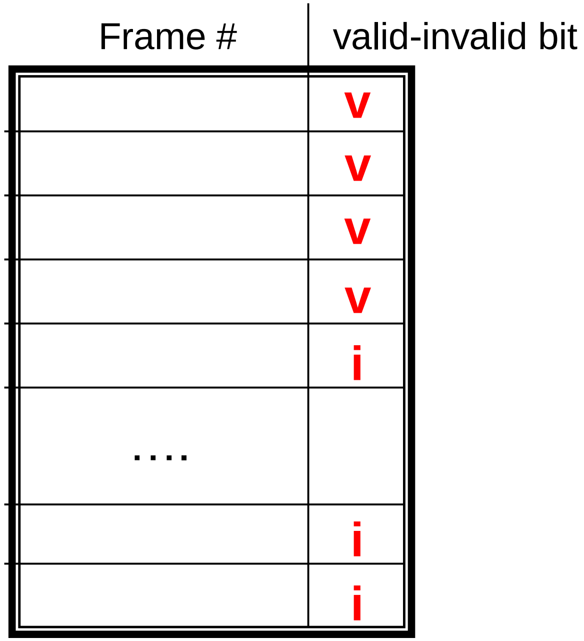


Demand Paging

- Bring a page into memory only when it is needed
  - Less memory needed
  - Faster response
  - More users
- Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory
- **Lazy swapper** – never swaps a page into memory unless page will be needed
  - Swapper that deals with pages is a **pager**

Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated (**v** ⇒ in-memory, **i** ⇒ not-in-memory)
- Initially valid–invalid bit is set to **i** on all entries
- Example of a page table snapshot:

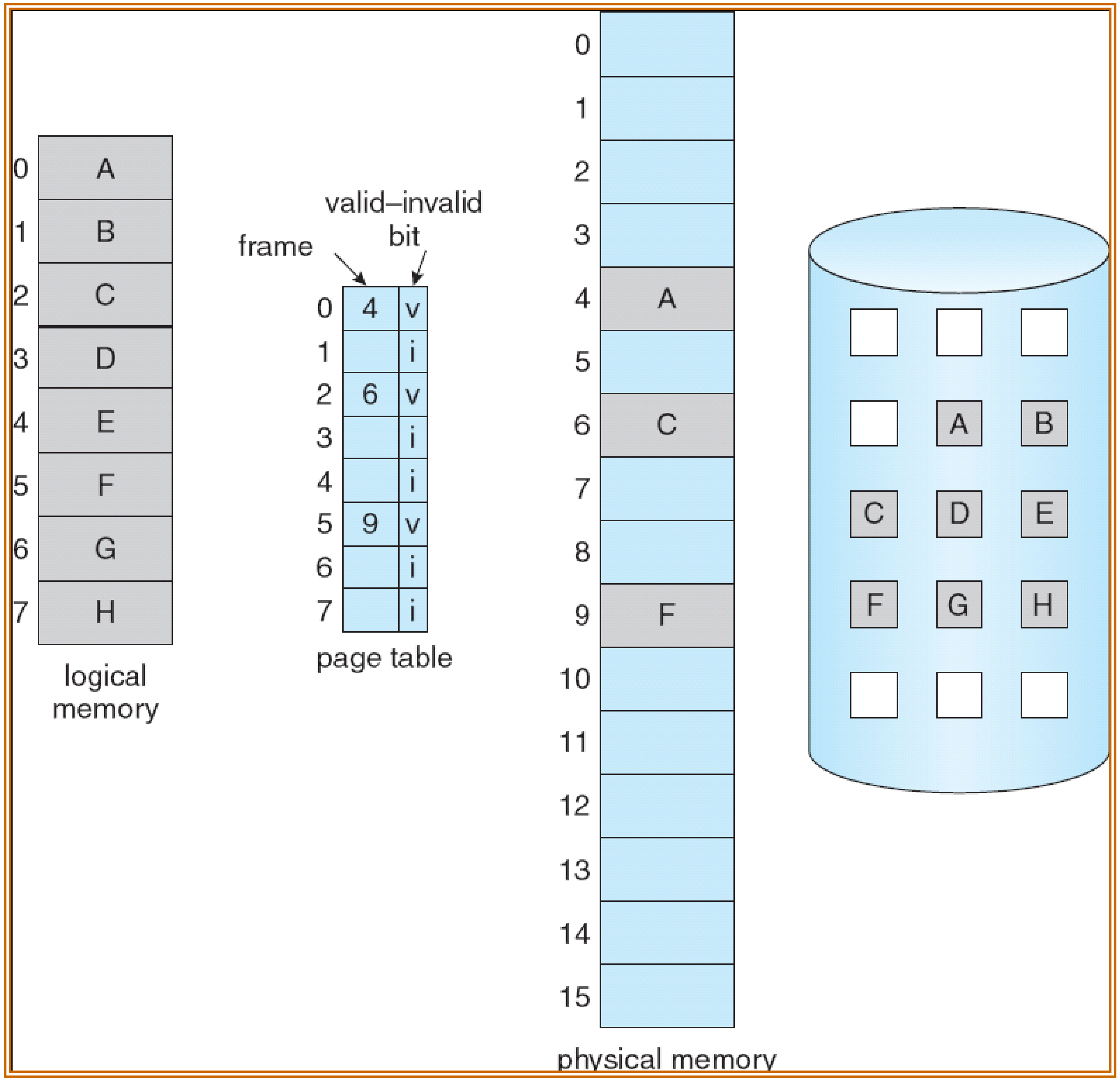


Frame #	valid-invalid bit
	v
	v
	v
	v
	i
….	
	i
	i

page table

- During address translation, if valid–invalid bit in page table entry is **i** ⇒ page fault

Page Table When Some Pages Are Not in Main Memory



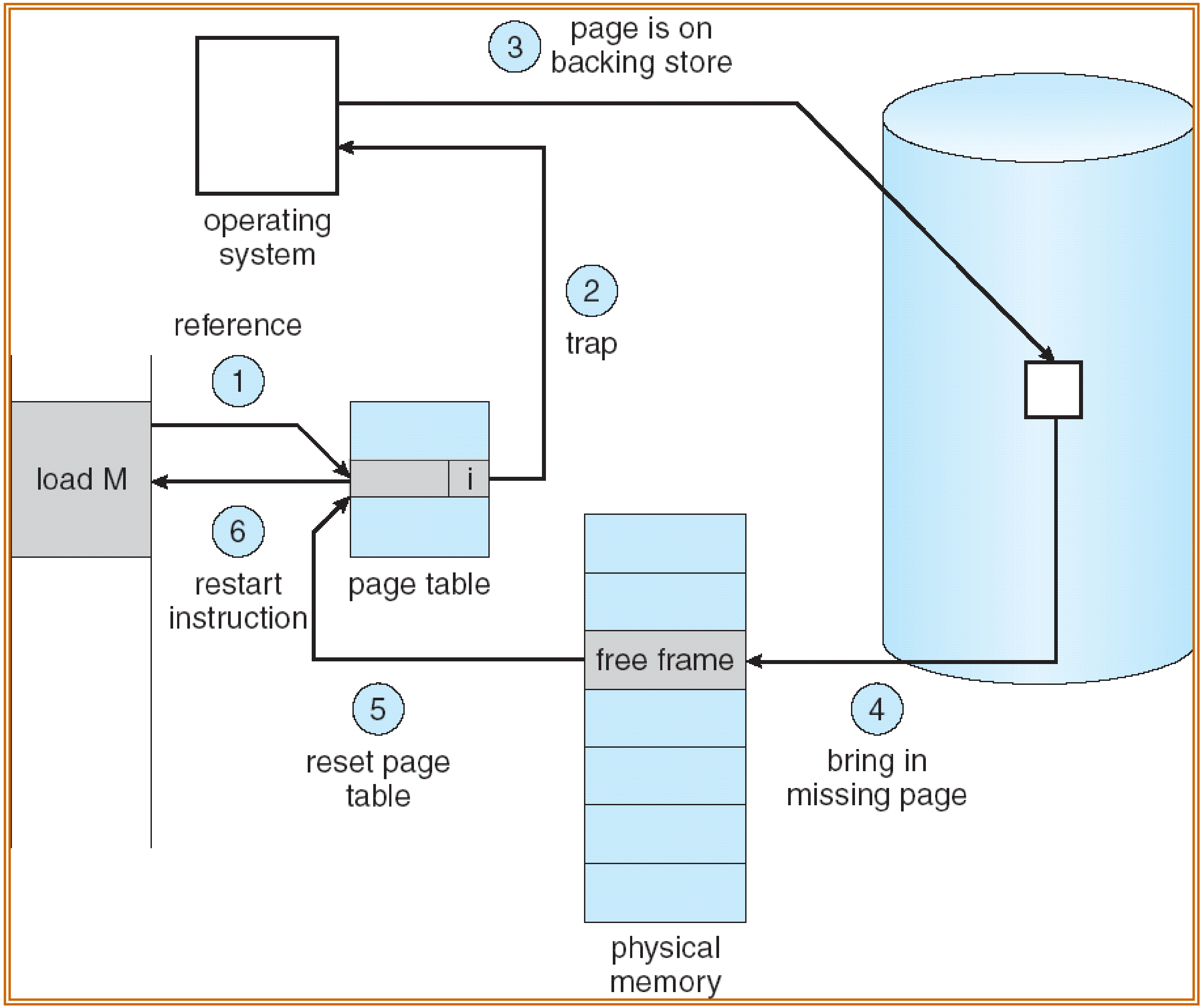
Page Fault

- If there is a reference to a page, first reference to that page will trap to operating system:

  page fault

1. Operating system looks at another table to decide:
   - Invalid reference ⇒ abort
   - Just not in memory
2. Get empty frame
3. Swap page into frame
4. Reset tables
5. Set validation bit = **v**
6. Restart the instruction that caused the page fault

Steps in Handling a Page Fault



Performance of Demand Paging

- Page Fault Rate $0 \le p \le 1.0$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault

- Effective Access Time (EAT)

$$
\begin{aligned}
\text{EAT} = (1 - p) \times \text{memory access} \\
+\, p\ (\text{page fault overhead} \\
+\ \text{swap page out} \\
+\ \text{swap page in} \\
+\ \text{restart overhead}\ )
\end{aligned}
$$

Demand Paging Example

- Memory access time = 200 nanoseconds
- Average page-fault service time = 8 milliseconds
- EAT = $(1 – p) \times 200 + p$ (8 milliseconds)

  $= (1 – p \ \ \times 200 + p \times 8{,}000{,}000$

  $= 200 + p \times 7{,}999{,}800$
- If one access out of 1,000 causes a page fault, then

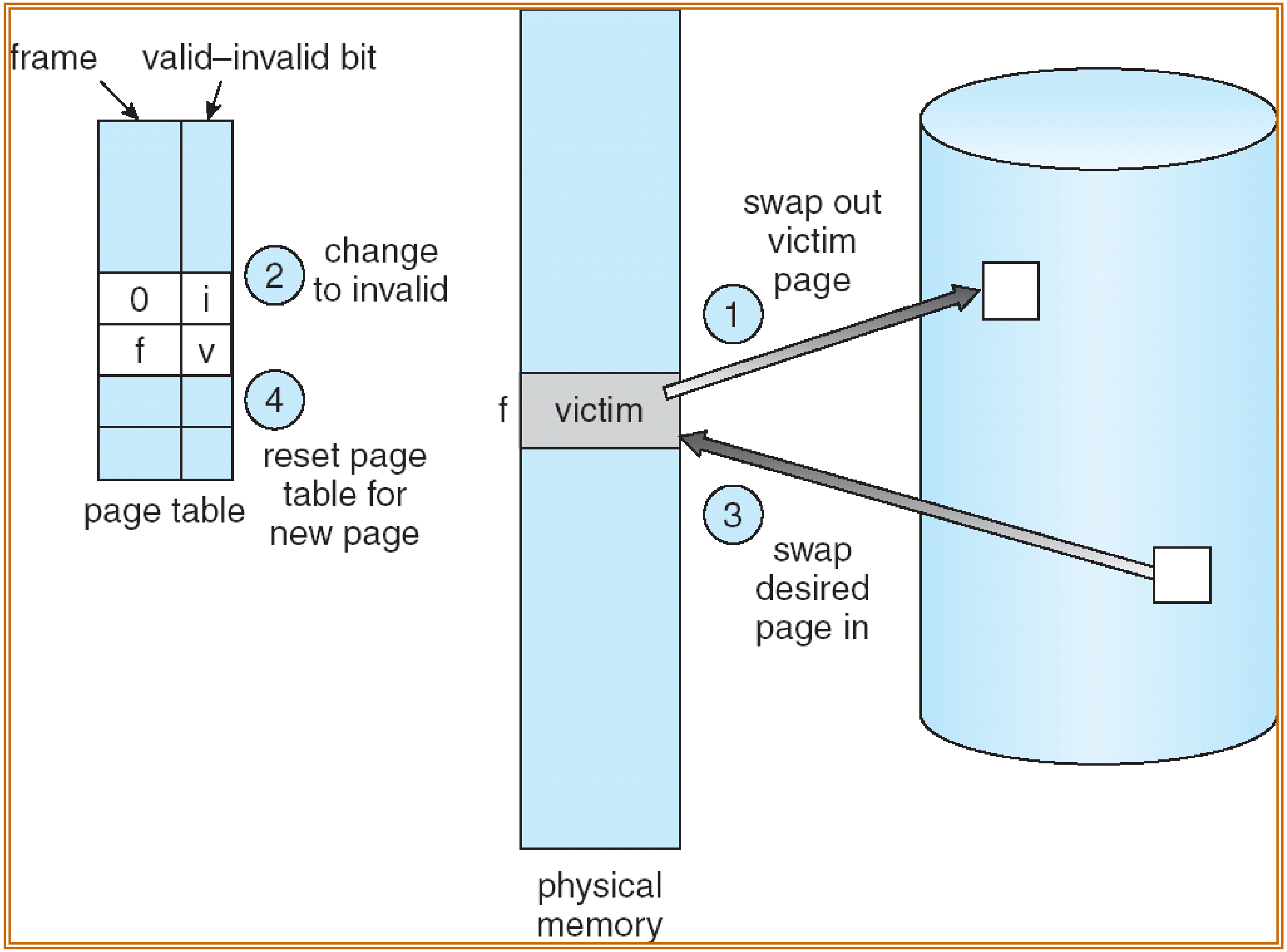
  EAT = 8.2 microseconds.

  This is a slowdown by a factor of 40!!

Basic Page Replacement

1. Find the location of the desired page on disk
2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a **victim** frame
3. Bring the desired page into the (newly) free frame; update the page and frame tables
4. Restart the process

Page Replacement



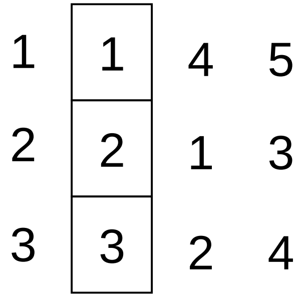
Page Replacement Algorithms

- Want lowest page-fault rate
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
- In all our examples, the reference string is

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)



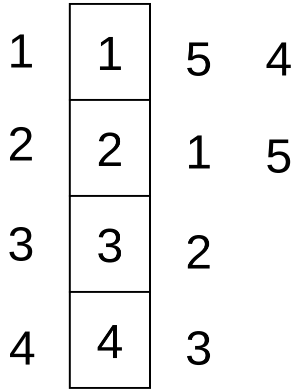
1 1 4 5

2 2 1 3

3 3 2 4

9 page faults

- 4 frames



1 1 5 4

2 2 1 5

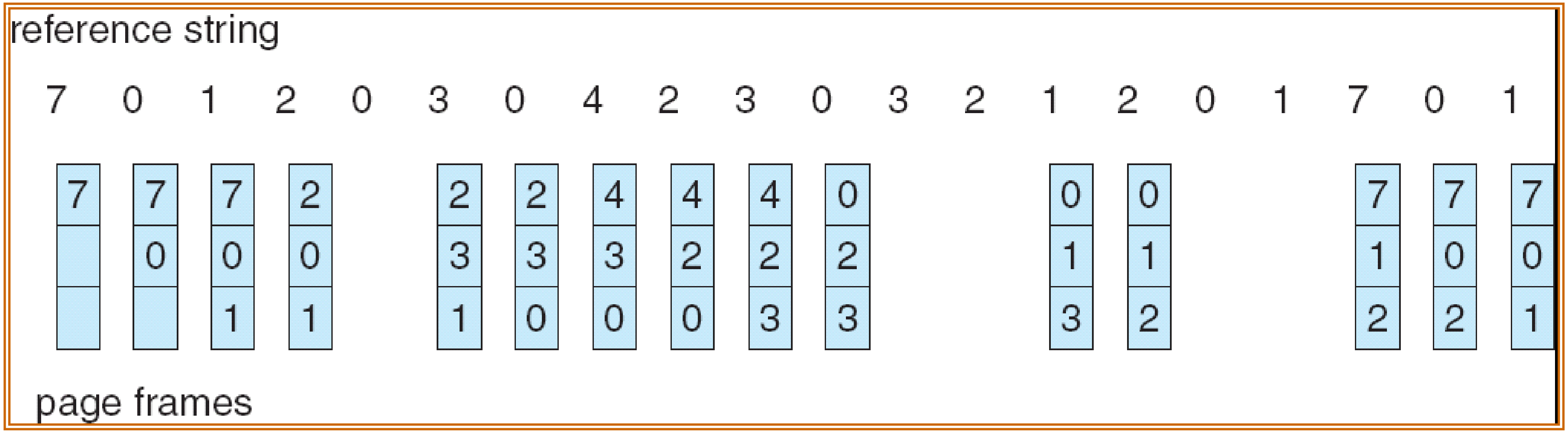
3 3 2

4 4 3

10 page faults

- Belady’s Anomaly: more frames ⇒ more page faults

FIFO Page Replacement



reference string

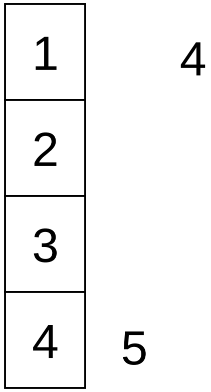
7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2	2	4	4	4	0			0	0			7	7	7
	0	0	0		3	3	3	2	2	2			1	1			1	0	0
		1	1		1	0	0	0	3	3			3	2			2	2	1

page frames

Optimal Algorithm

- Replace page that will not be used for longest period of time
- 4 frames example

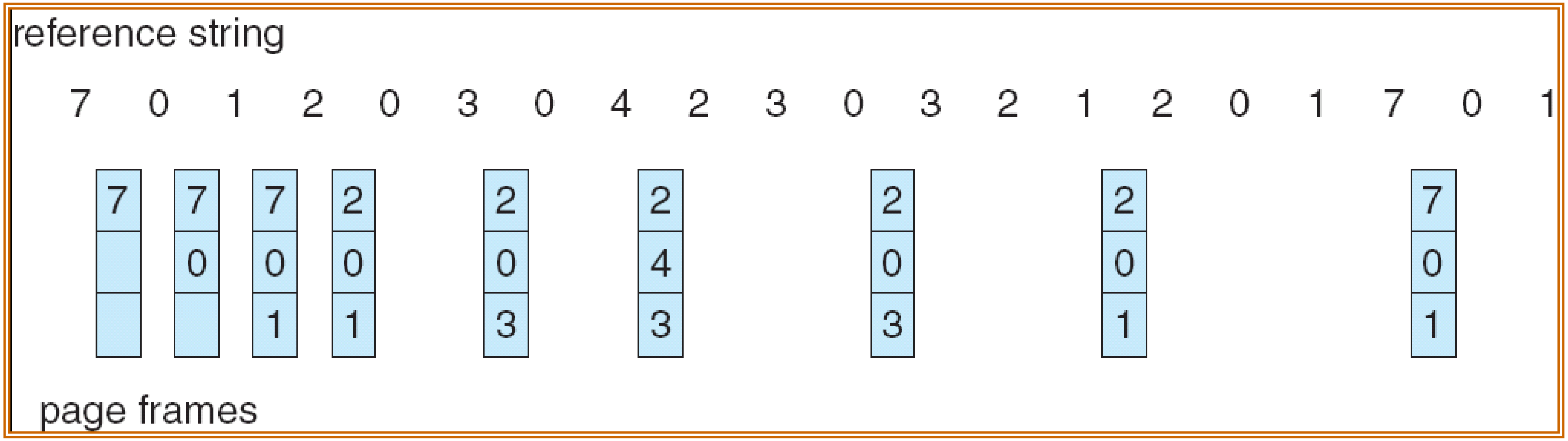
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



6 page faults

- How do you know this?
- Used for measuring how well your algorithm performs

Optimal Page Replacement



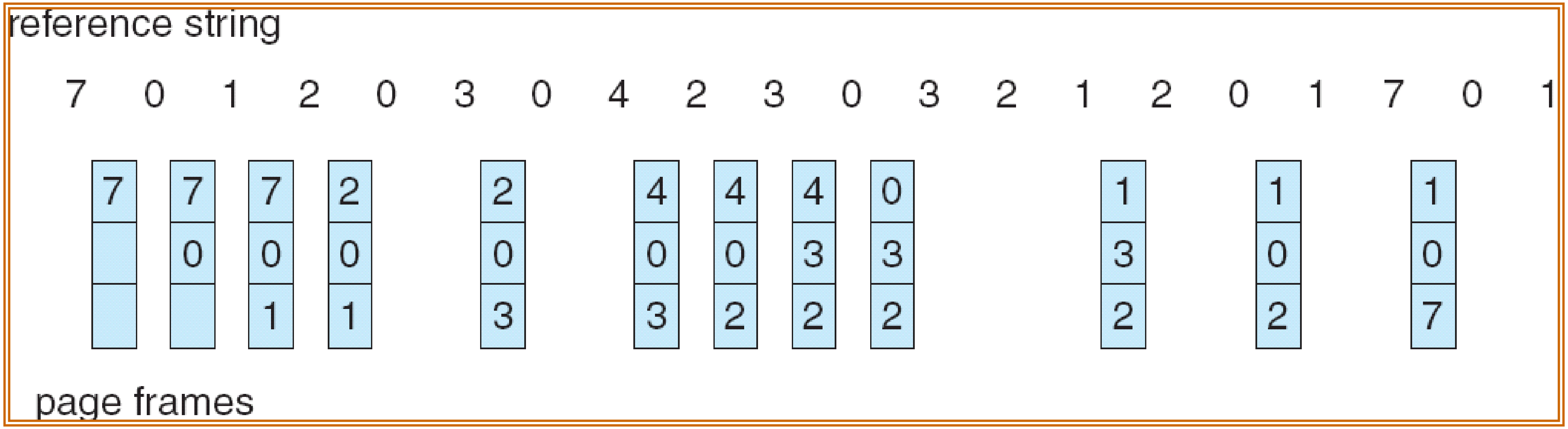
Least Recently Used (LRU) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**

1	1	1	1	**5**
2	2	2	2	2
3	**5**	5	**4**	4
4	4	**3**	3	3

- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
  - When a page needs to be changed, look at the counters to determine which are to change

LRU Page Replacement



reference string

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

page frames

Counting Algorithms

- Keep a counter of the number of references that have been made to each page
- **LFU Algorithm**: replaces page with smallest count
- **MFU Algorithm**: based on the argument that the page with the smallest count was probably just brought in and has yet to be used