COMP-SCI-431
Intro Operating Systems

Lecture 6 – Memory Management

Adu Baffour, PhD

University of Missouri-Kansas City
Division of Computing, Analytics, and Mathematics
School of Science and Engineering
aabnxq@umkc.edu

UMKC

Lecture Objectives

- To understand the requirements for efficient memory management.
- To explore strategies for managing situations with insufficient memory.
- To gain insights into the concept and implementation of paging in memory management.
- To understand the principles and benefits of combining segmentation and paging.

Outline

6.1 Requirements for efficient memory management

6.2 Managing insufficient memory

6.3 Paging

6.4 Segmentation and paging

Outline

6.1 Requirements for efficient memory management

6.2 Managing insufficient memory

6.3 Paging

6.4 Segmentation and paging

UMKC

6.1 Requirements for efficient memory management

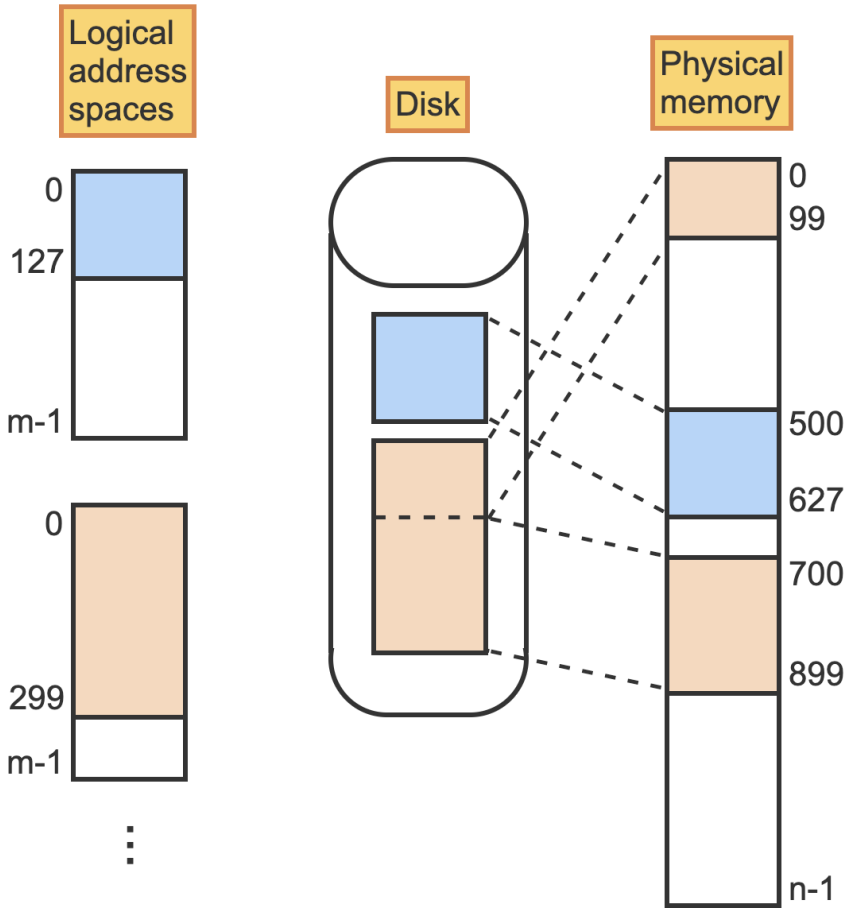
Logical vs physical memory

- A computer's ***physical memory*** (***RAM***) is a hardware structure consisting of a linear sequence of words that hold a program during execution.
- A ***word*** is a fixed-size unit of data. A typical word size is 1, 2, or 4 bytes.
- A ***physical address*** is an integer in the range [0: n-1] that identifies a word in a physical memory of size n.
- A ***logical address space*** is an abstraction of physical memory, consisting of a sequence of imaginary memory locations in a range [0: m-1], where m is the size of the logical address space.
- A ***logical address*** is an integer in the range [0: m-1] that identifies a word in a logical address space.

UMKC

6.1 Requirements for efficient memory management

Logical vs physical memory



6.1 Requirements for efficient memory management

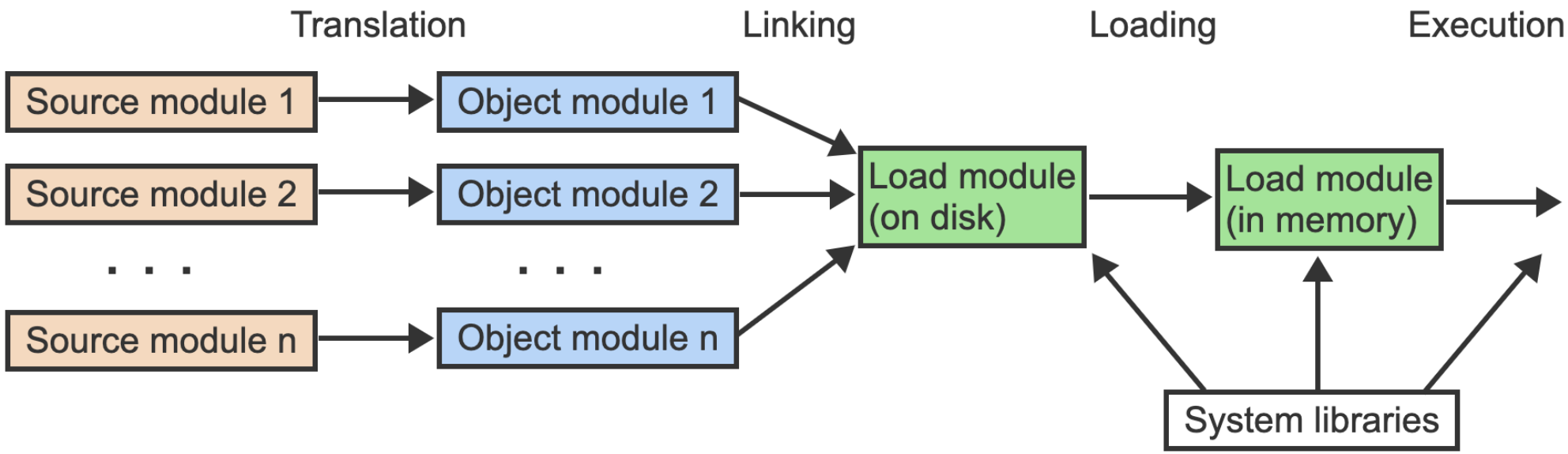
Program transformations

- A ***source module*** is a program or a program component written in a symbolic language, like C, or an assembly language that a compiler or assembler must translate into executable machine code.
- An ***object module*** is the machine-language output of a compiler or assembler generated from a source module.
- A ***load module*** is a program or combination of programs in a form ready to be loaded into the main memory and executed.

UMKC

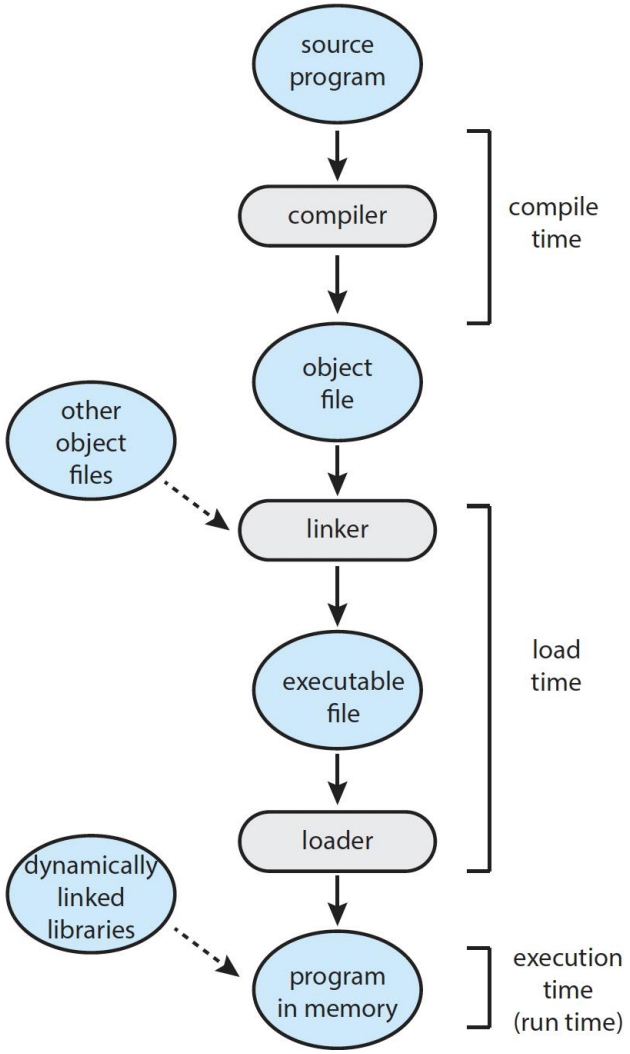
6.1 Requirements for efficient memory management

Program transformations



6.1 Requirements for efficient memory management

Program transformations



6.1 Requirements for efficient memory management

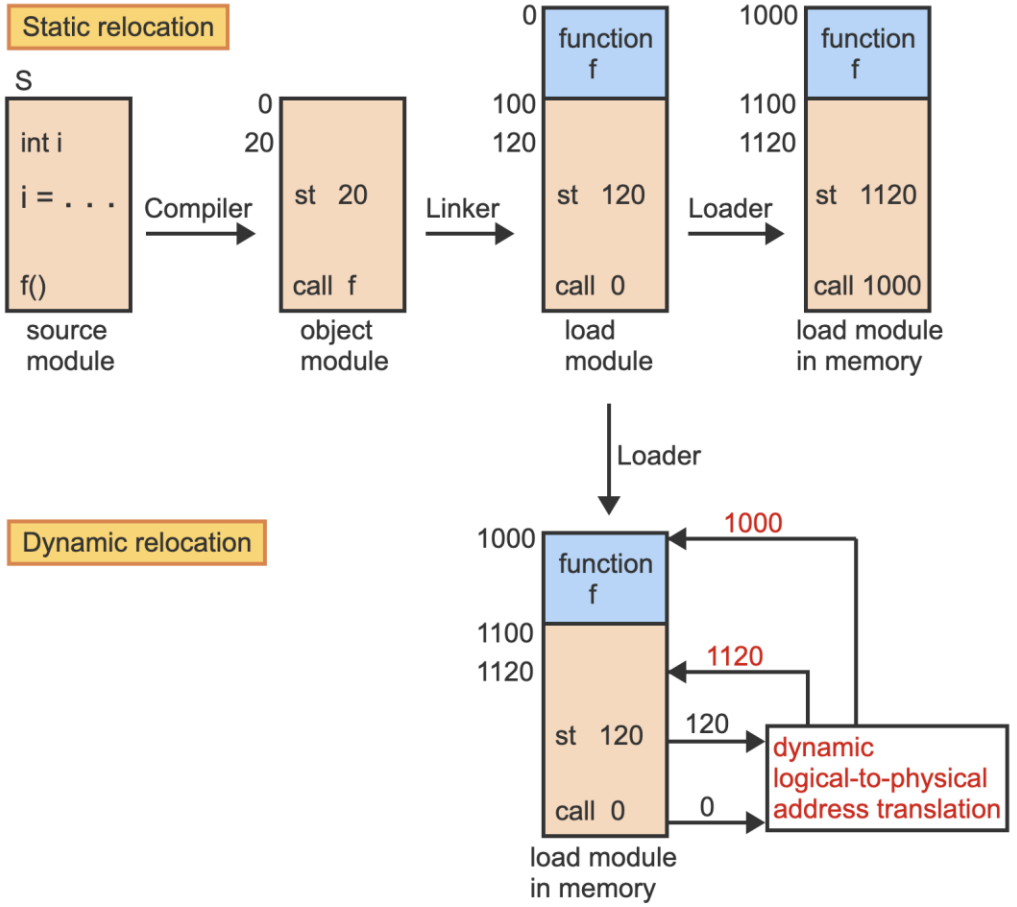
Relocation and address binding

- Program ***relocation*** is moving a program component from one address space to another.
- The relocation may be between two logical address spaces or from a logical address space to a physical address space.
- ***Static relocation*** binds all logical addresses to physical addresses before execution.
- ***Dynamic relocation*** postpones binding a logical address to a physical address until the addressed item is accessed during execution.

UMKC

6.1 Requirements for efficient memory management

Static vs dynamic relocation



6.1 Requirements for efficient memory management

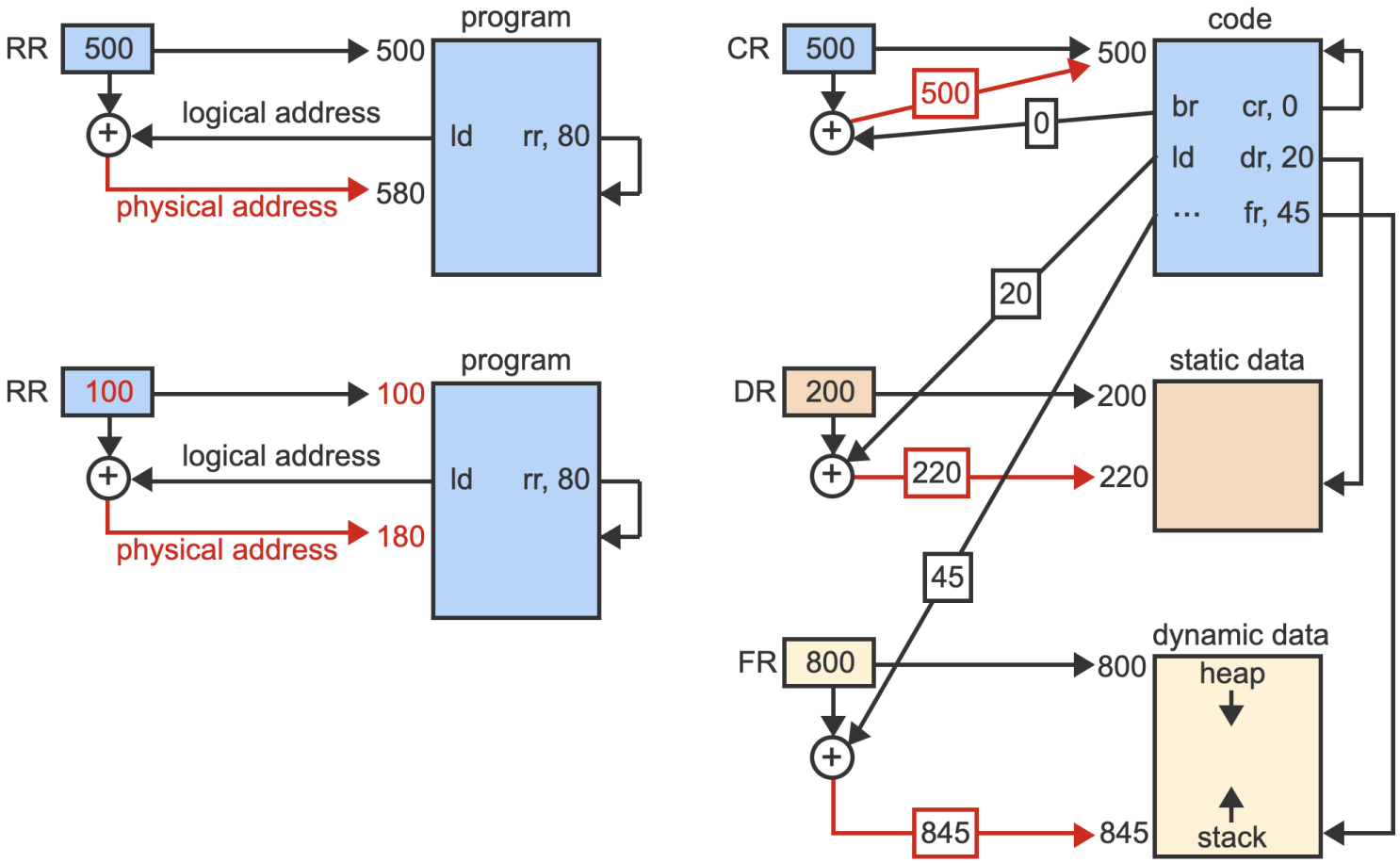
Implementing dynamic relocation using relocation registers

- A ***relocation register*** contains the physical starting address of a program or program component in memory.
- Most programs have 3 main components: code, static data, and dynamic data.
- A simple memory management scheme treats the 3 components as one unit, which must be moved into one contiguous memory area.
- Dynamic relocation can be implemented using a single relocation register loaded with the program's starting address.
- The CPU adds the register content automatically to every logical address of the program to generate the physical address.
- A more flexible management scheme treats the 3 components as separate modules, each of which may reside in a different memory area.
- Three relocation registers, each loaded with the starting address of one of the modules, then accomplish dynamic relocation by being added to all logical addresses at runtime.

UMKC

6.1 Requirements for efficient memory management

Dynamic relocation using relocation registers



6.1 Requirements for efficient memory management

Free space management

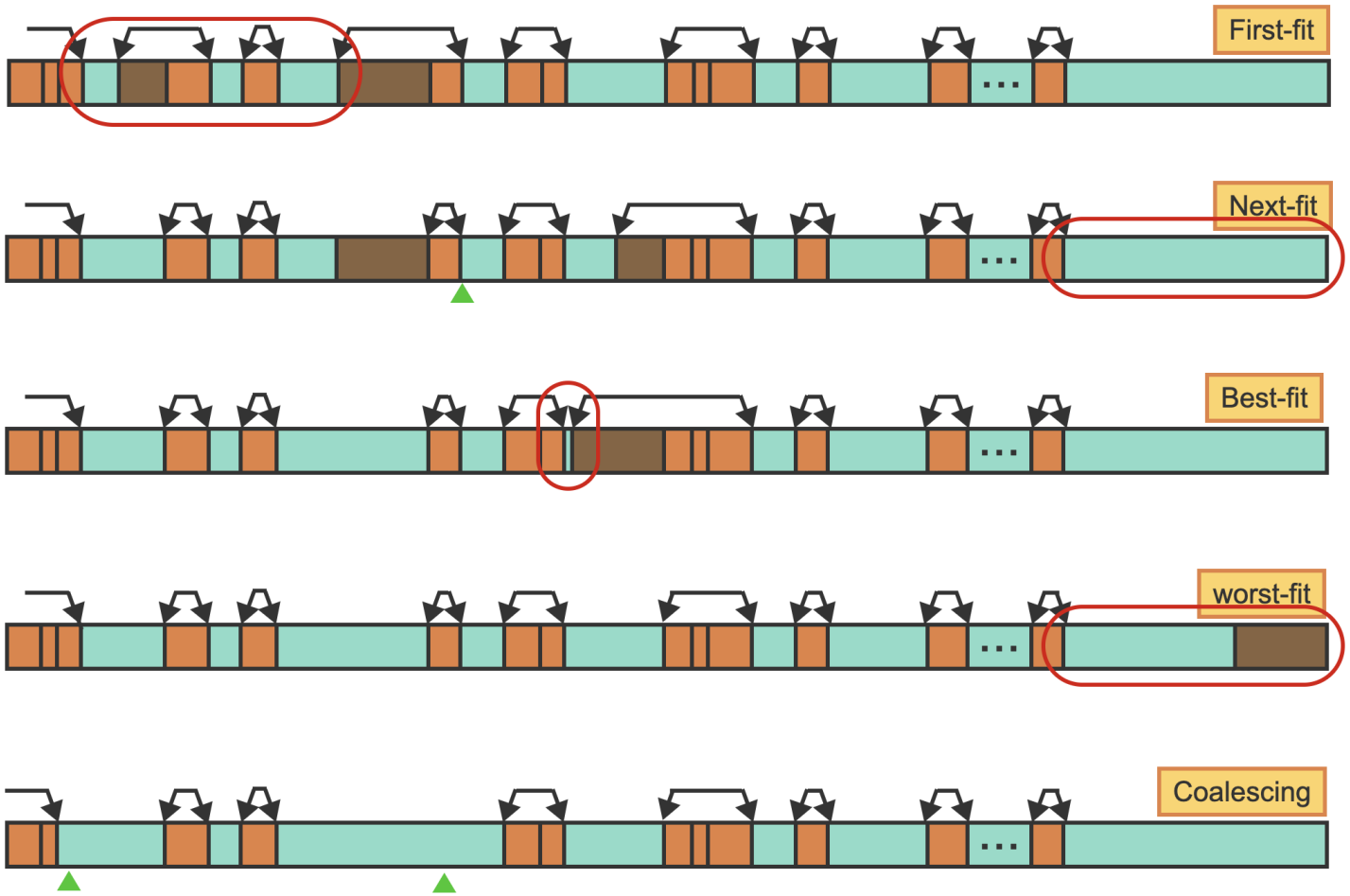
Different search strategies have been explored:

- ***First-fit*** always starts the search from the beginning of the list and allocates the first hole large enough to accommodate the request.
- ***Next-fit*** starts each search at the point of the last allocation.
- ***Best-fit*** searches the entire list and chooses the smallest hole large enough to accommodate the request.
- ***Worst-fit*** takes the opposite approach from best-fit by always choosing the largest available hole for any request.
- First-fit and best-fit are better than worst-fit in terms of speed and storage utilization

UMKC

6.1 Requirements for efficient memory management

Allocation strategies and coalescing



Outline

6.1 Requirements for efficient memory management

6.2 Managing insufficient memory

6.3 Paging

6.4 Segmentation and paging

6.2 Managing insufficient memory

Memory fragmentation

- ***External memory fragmentation*** is the loss of usable memory space due to holes between allocated blocks of variable sizes, such that 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 not being used.
- First fit analysis reveals that 0.5 N blocks lost to fragmentation 1/3 may be unusable -> 50-percent rule given N blocks allocated.
- A smaller average hole size implies better memory utilization since less memory is wasted on holes.

UMKC

6.2 Managing insufficient memory

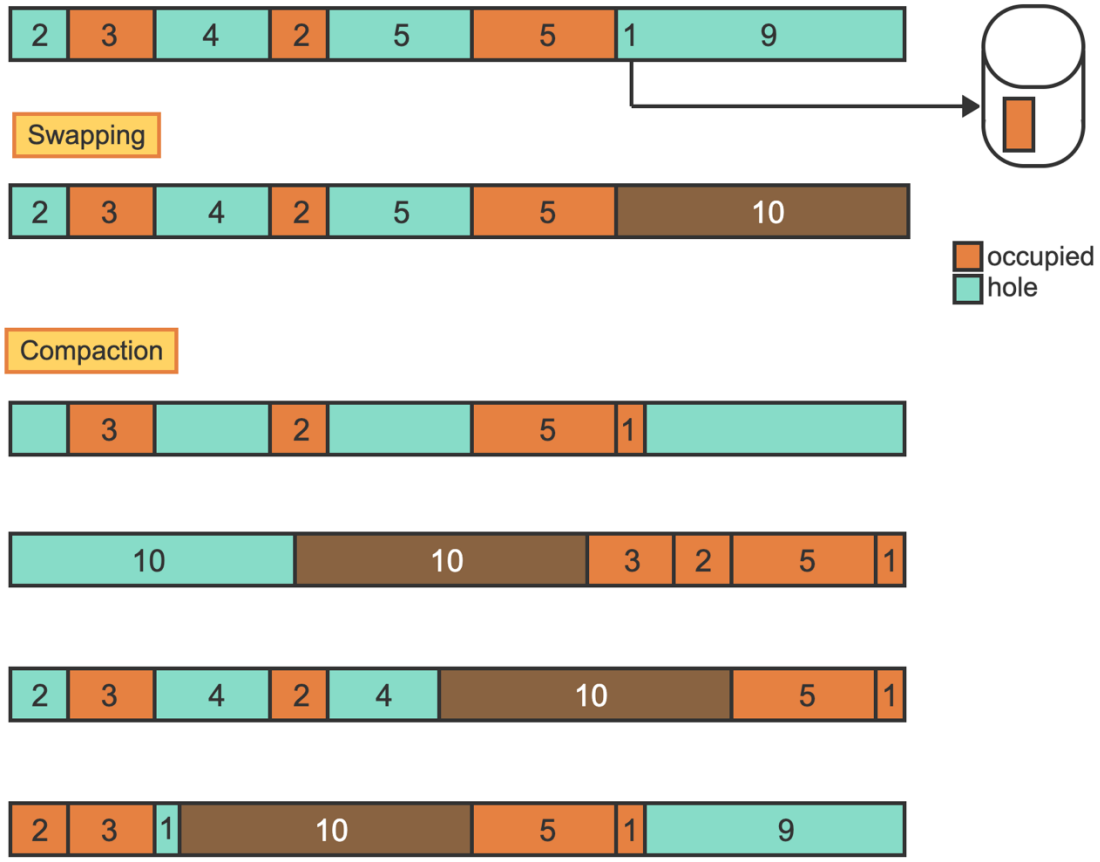
Swapping and memory compaction

- ***Swapping*** is the temporary removal of a module from memory.
- The module is saved on a disk and later moved back to memory.
- ***Memory compaction*** is the systematic shifting of modules in memory, generally in one direction, to consolidate multiple disjoint holes into one more giant hole.

UMKC

6.2 Managing insufficient memory

Swapping and compaction



6.2 Managing insufficient memory

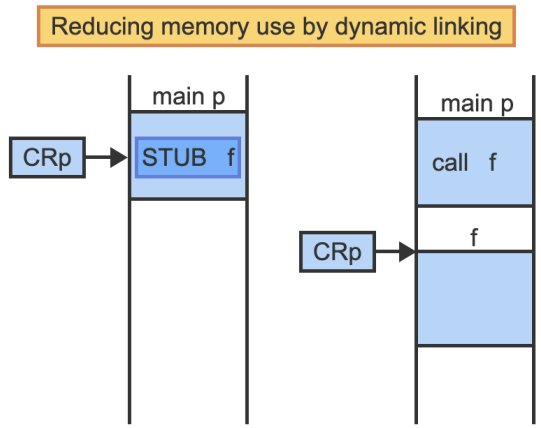
Dynamic linking and sharing

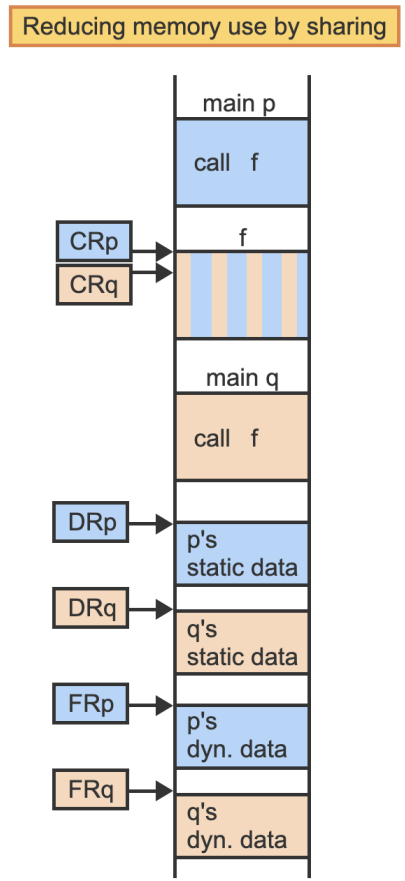
- ***Linking*** resolves external references among object modules and can be done statically, before loading, or dynamically while the program executes.
- Example: Windows DLLs
- ***Sharing*** is linking the same copy of a module to multiple other modules.
- Sharing improves memory utilization by allowing multiple processes to share common routines or services (Ex, compilers, editors, word processors) or common data (Ex, dictionaries).
- Sharing is possible under both static and dynamic linking.

UMKC

6.2 Managing insufficient memory

Dynamic linking and sharing





Outline

6.1 Requirements for efficient memory management

6.2 Managing insufficient memory

6.3 Paging

6.4 Segmentation and paging

6.3 Paging

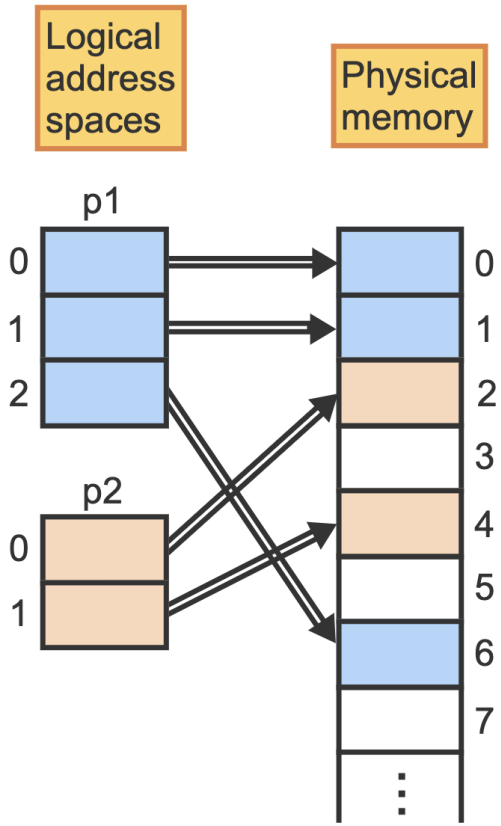
Principles of paging

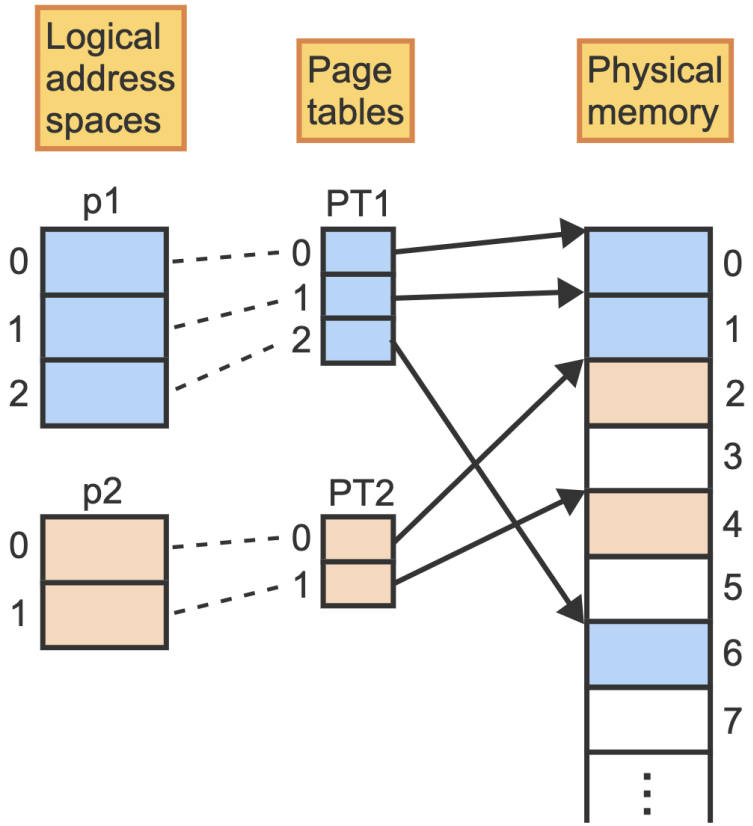
- Paging divides a process's logical address space and the physical memory (RAM) into contiguous, equal-sized partitions such that any logical partition can be mapped into any physical partition.
- A ***page*** is a fixed-size contiguous block of a logical address space identified by a single number, the page number.
- A ***page frame*** is a fixed-size contiguous block of physical memory identified by a single page frame number.
- A ***page table*** is an array that tracks which pages of a given logical address space reside in which page frames.

UMKC

6.3 Paging

Principles of paging





6.3 Paging

Address translation

- A logical address is broken into two components: a page number, p, and an offset w within the page – (p, w).
- Similarly, a physical address is broken into two components: a frame number f and an offset w within the frame – (f, w)
- The OS must translate logical addresses into corresponding physical addresses:

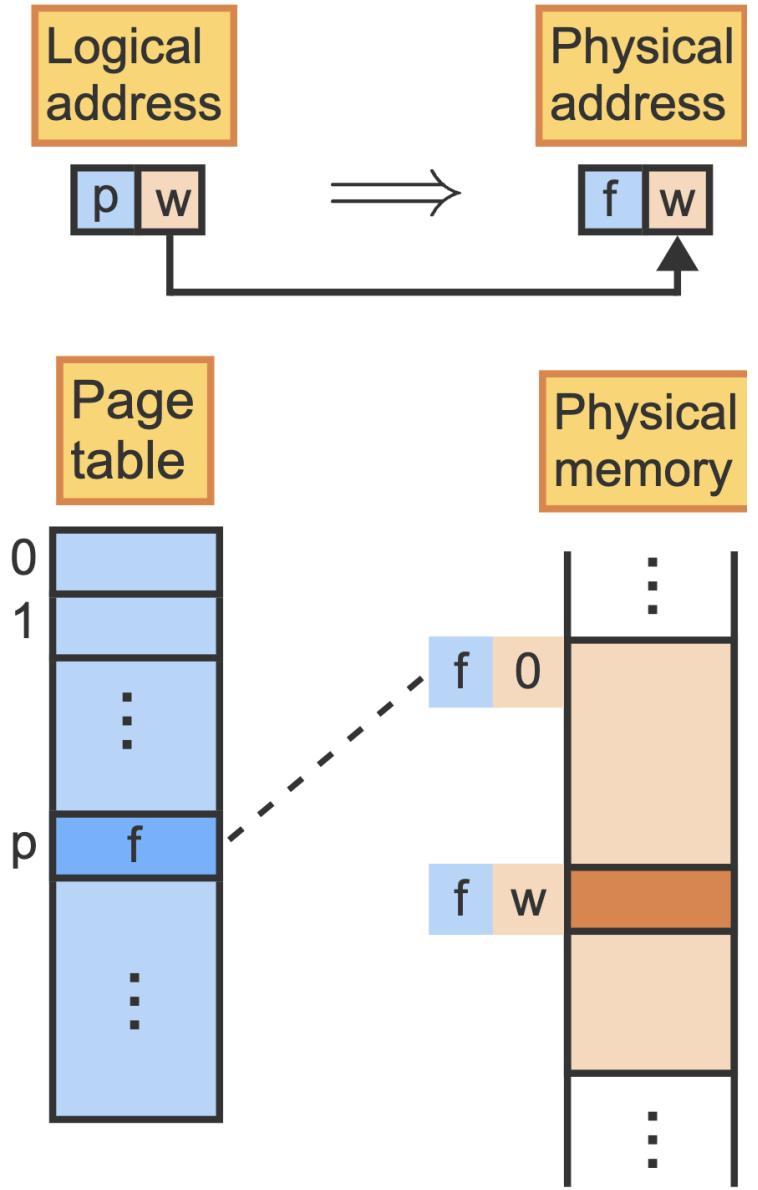
1. Given a logical address (p, w), access the page table entry corresponding to page p.
2. Read the frame number, f, of the frame containing p.
3. Combine f with the offset w to find the physical address (f, w) corresponding to the logical address (p, w).

UMKC

6.3 Paging

Address translation with paging

1. A logical address (p, w) needs to be translated into a physical address.
2. The page number p is used to access the page table at offset w.
3. The page table entry contains the frame number f holding the page p. The address (f, 0) is the starting address of the frame.
4. The frame number f is copied into the physical address being formed.
5. The offset w is copied from the logical address to the physical address unchanged.
6. (f, w) identifies the word in physical memory corresponding to the word in the logical address space at address (p, w).



6.3 Paging

Internal fragmentation and bound check

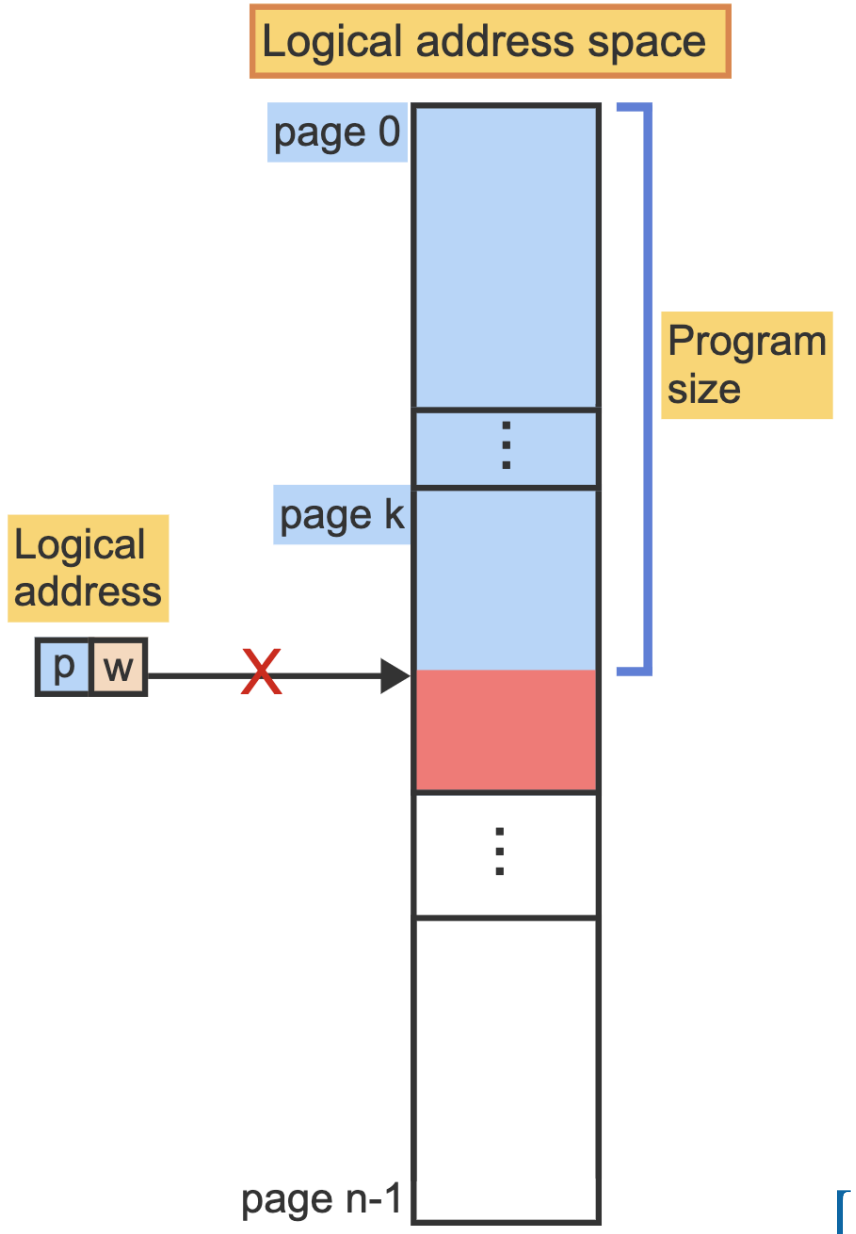
- ***Internal fragmentation*** occurs when we lose usable memory space due to the mismatch between the page size and the program size, which creates a hole at the end of the program's last page.
- ***Bound check***. Most programs also do not need all pages of the entire logical address space and must be prevented from accessing any page not belonging to the program.
- Some systems implement a valid-bit in each page table entry and prevent access to pages not belonging to the program.

UMKC

6.3 Paging

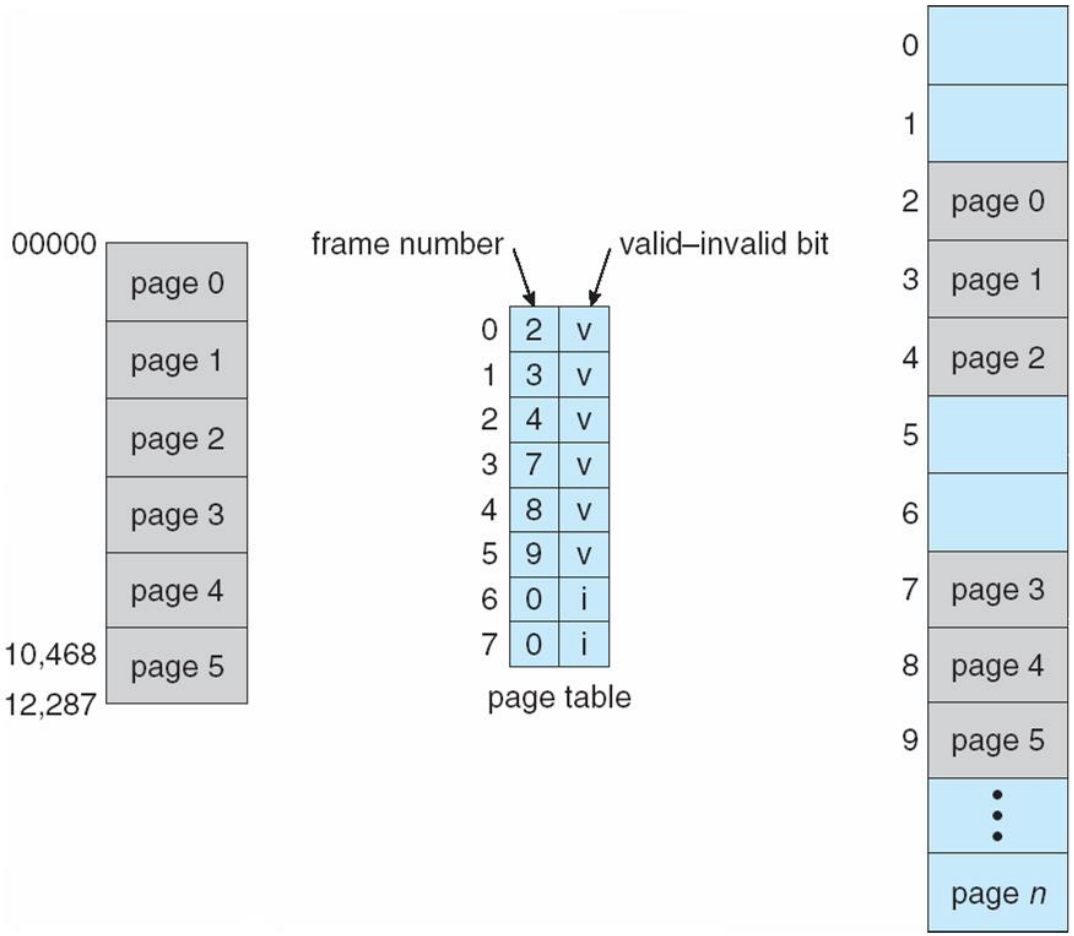
Internal fragmentation

1. A logical address (p, w) needs to be translated into a physical address.
2. The page number p is used to access the page table at offset w.
3. The page table entry contains the frame number f holding the page p. The address (f, 0) is the starting address of the frame.
4. The frame number f is copied into the physical address being formed.
5. The offset w is copied from the logical address to the physical address unchanged.
6. (f, w) identifies the word in physical memory corresponding to the word in the logical address space at address (p, w).



6.3 Paging

Internal fragmentation and bound check



Outline

6.1 Requirements for efficient memory management

6.2 Managing insufficient memory

6.3 Paging

6.4 Segmentation and paging

UMKC

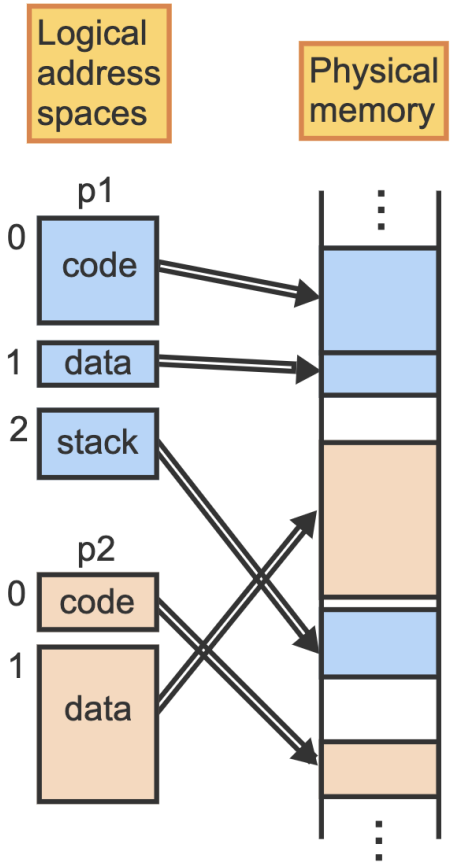
6.4 Segmentation and paging

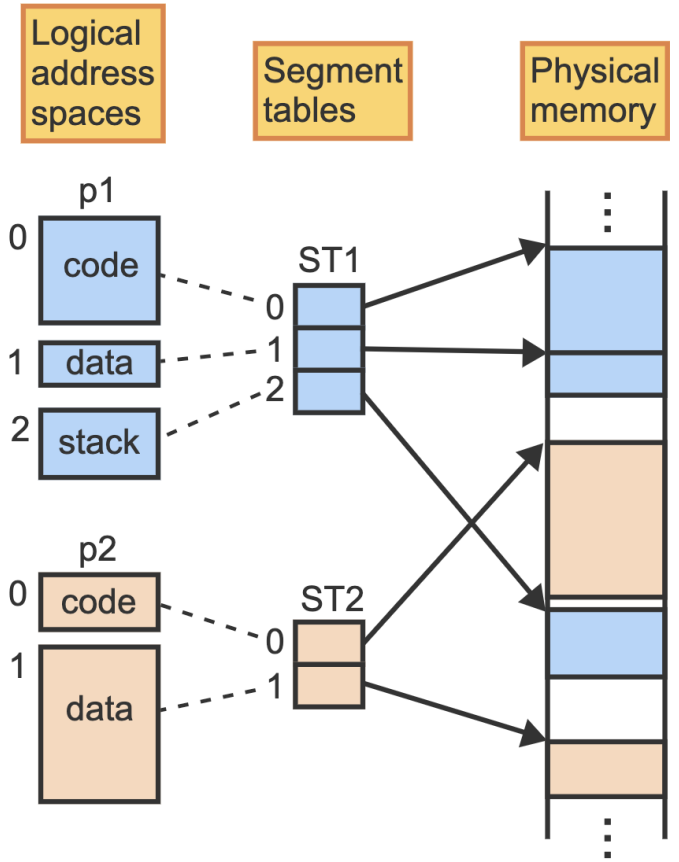
Principles of segmentation

- A ***segment*** is a variable-size block of a logical address space identified by a segment number.
- With pure segmentation (no paging), a segment occupies a contiguous area of physical memory and is the smallest data unit for memory management.
- A ***segment table*** is an array that tracks which segment resides in which area of physical memory.
- Each entry corresponds to one segment and contains the starting address of the segment.

6.4 Segmentation and paging

1. Process p1's logical address space consists of 3 segments, which are mapped to variable-size areas in physical memory based on space availability.
2. Similarly, process p2's segments are mapped to variable-size areas in physical memory.
3. To keep track of the mapping, each process has a segment table (ST1 and ST2), where each entry corresponds to one of the segments.
4. Each segment table entry points to the beginning of the corresponding segment in physical memory.





UMKC

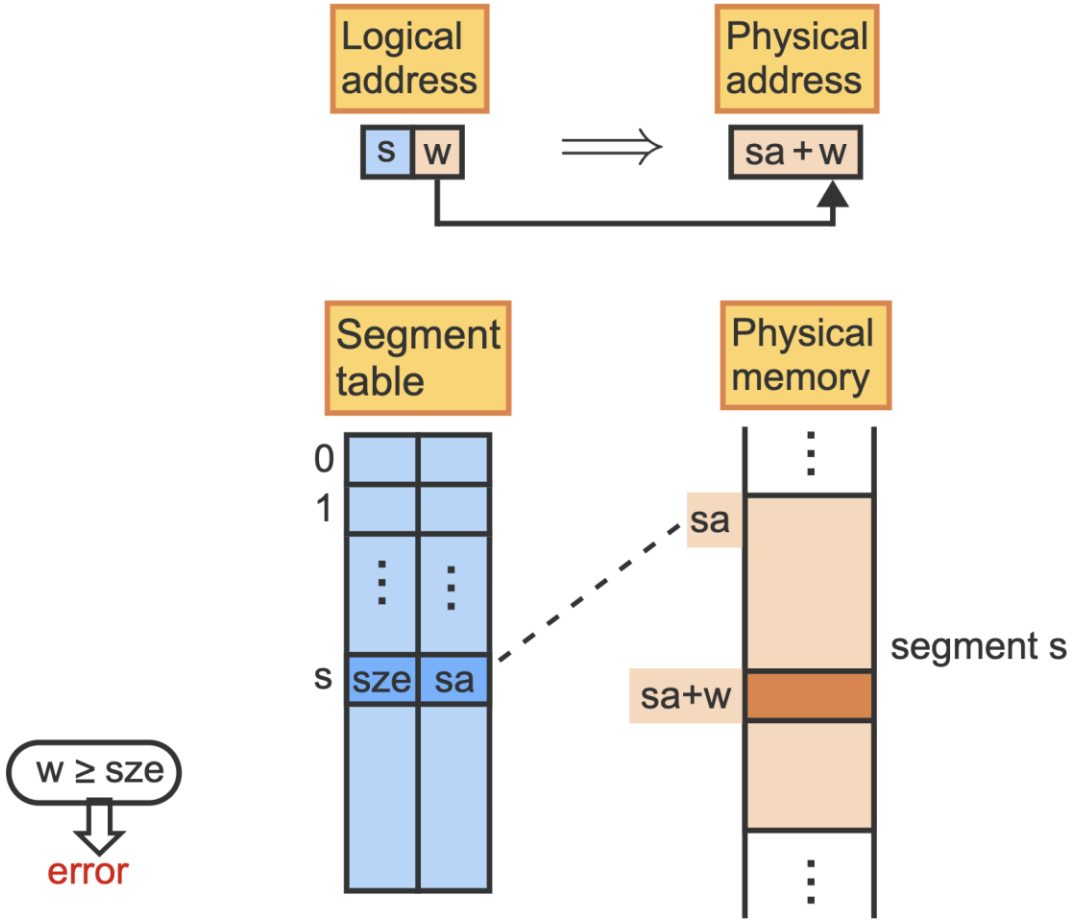
6.4 Segmentation and paging

Address translation

1. Given a logical address (s, w), access the segment table entry at offset s and get the segment's starting address, sa, and size, sze.
2. If w ≥ sze then reject the address as illegal.
3. Otherwise, add w to sa to form the physical address, sa + w, corresponding to the logical address (s, w).

6.4 Segmentation and paging

Address translation with segmentation



6.4 Segmentation and paging

Segmentation with paging

- The logical address space consists of multiple variable-size segments but each segment is divided into fixed-size pages.
- A logical address is then divided into 3 components: a segment number s, a page number p, and an offset w within the page.
- A physical address is divided into 2 components as before: a frame number f and an offset w within the frame.
- Every process has one segment table where each entry points to a page table corresponding to one of the segments and each page table entry points to a page frame.
- The OS then translates logical addresses of the form (s, p, w) into corresponding physical addresses (f, w).

UMKC

6.4 Segmentation and paging

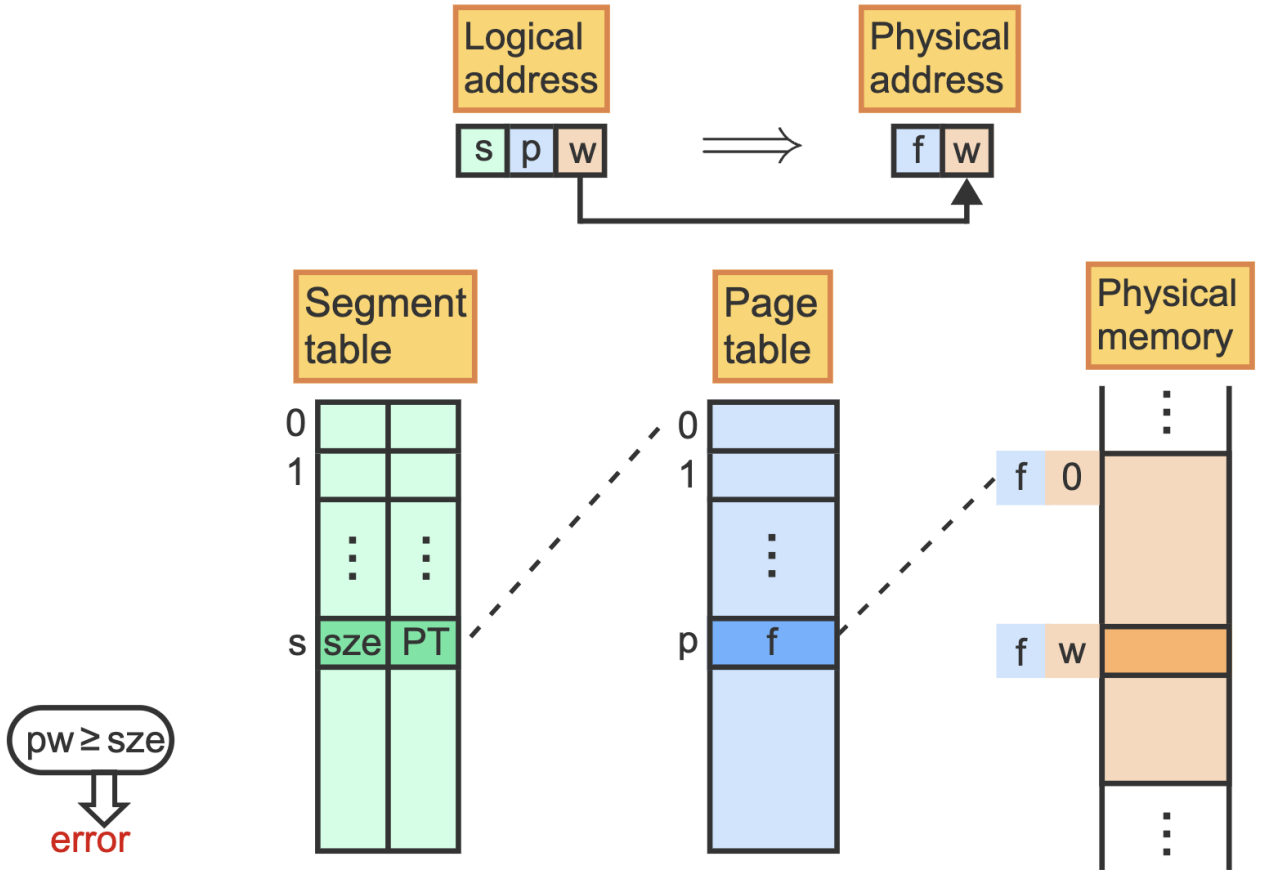
Segmentation with paging

1. Given a logical address (s, p, w), access the segment table at offset s to find the page table and the size, sze, of segment s.
2. If (p, w) ≥ sze then reject the address as illegal. Otherwise, access the page table at offset p and read the frame number, f, of the frame containing page p.
3. Combine f with the offset w to form the physical address, (f, w).

UMKC

6.4 Segmentation and paging

Address translation with segmentation and paging



End of Lecture

Thank you

Any questions?

UMKC