Memory Management for Main Memory

- Spring 2022
- Chi-Sheng Shih/Chung-Wei Lin
- National Taiwan University

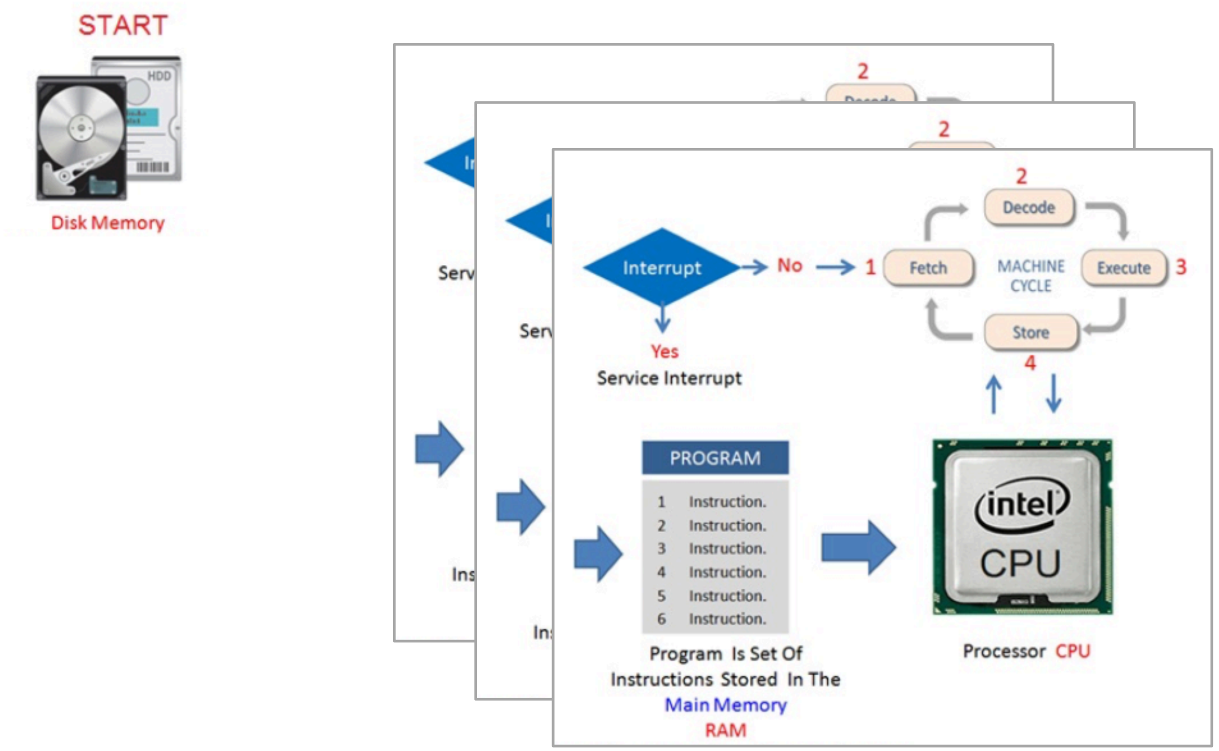
施吉昇

國立臺灣大學

NEWS Lab 嵌入式系統暨無線網路實驗室

Overview of Memory Management

- How to manage memory for limited number of processes on bare metal (physical) ***main memory***? (ch. 9)



Agenda for Main Memory

Ch.9 (OSC)

Agenda for Main Memory

Ch.9 (OSC)

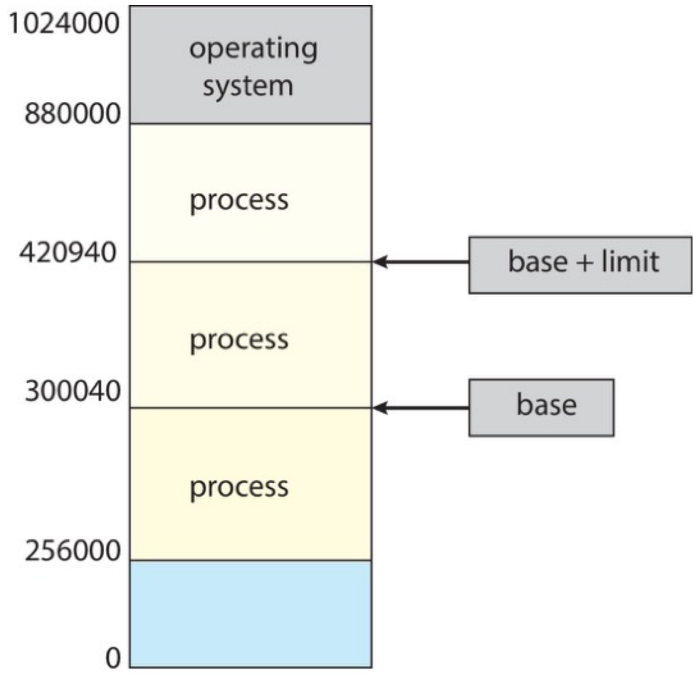
Background

Background

- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage that CPU can access directly
- Memory unit only sees a stream of:
  - addresses + read requests, or
  - address + data and write requests
- Register access is done in one CPU clock (or less)
- Main memory can take many cycles, causing a stall
- Cache sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

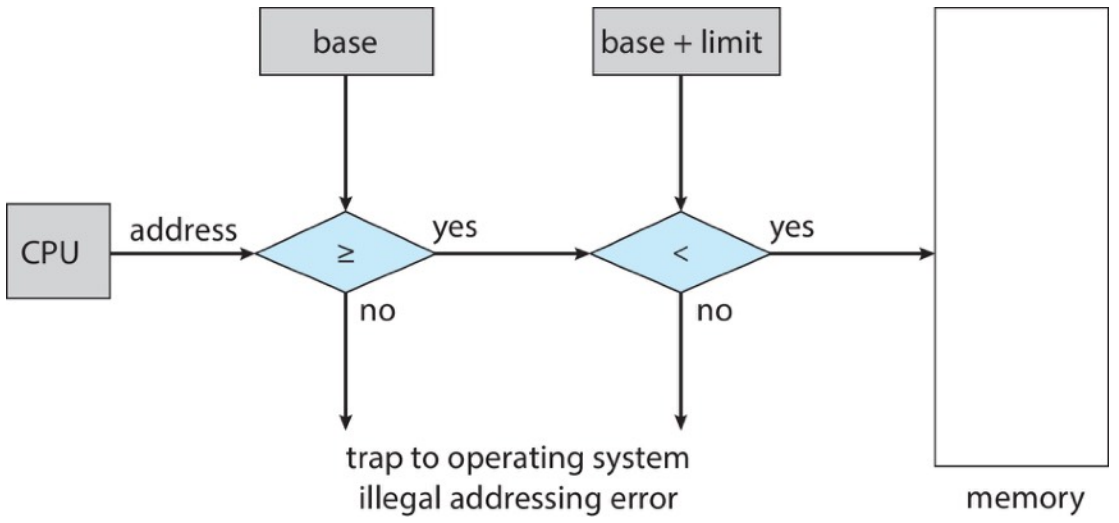
Memory Protection

- Need to censure that a process can access only those addresses in its address space.
- We can provide this protection by using a pair of **base** and **limit registers** define the logical address space of a process



Hardware Address Protection

- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user



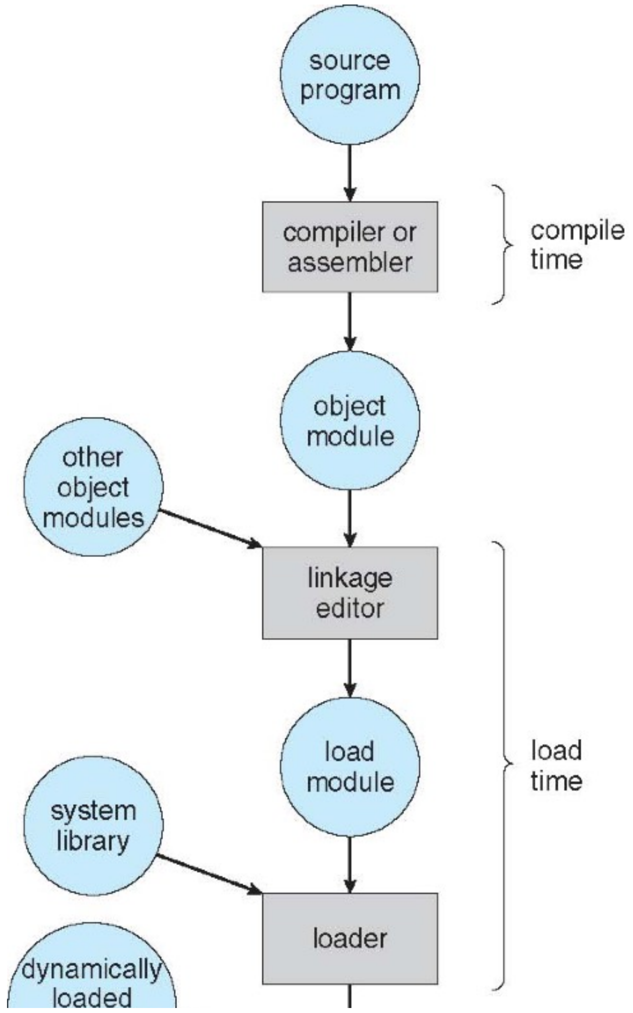
- the instructions to loading the base and limit registers are privileged

Address Binding

- Programs on disk, ready to be brought into memory to execute form an input queue
  - Without support, must be loaded into address 0000
- Inconvenient to have first user process' physical address always at 0000
  - How can it not be?
- Addresses represented in different ways at different stages of a program's life
  - Source code addresses usually symbolic
  - Compiled code addresses bind to relocatable addresses, i.e., "14 bytes from beginning of this module"
  - Linker or loader will bind relocatable addresses to absolute addresses, i.e., 74014
  - Each binding maps one address space to another

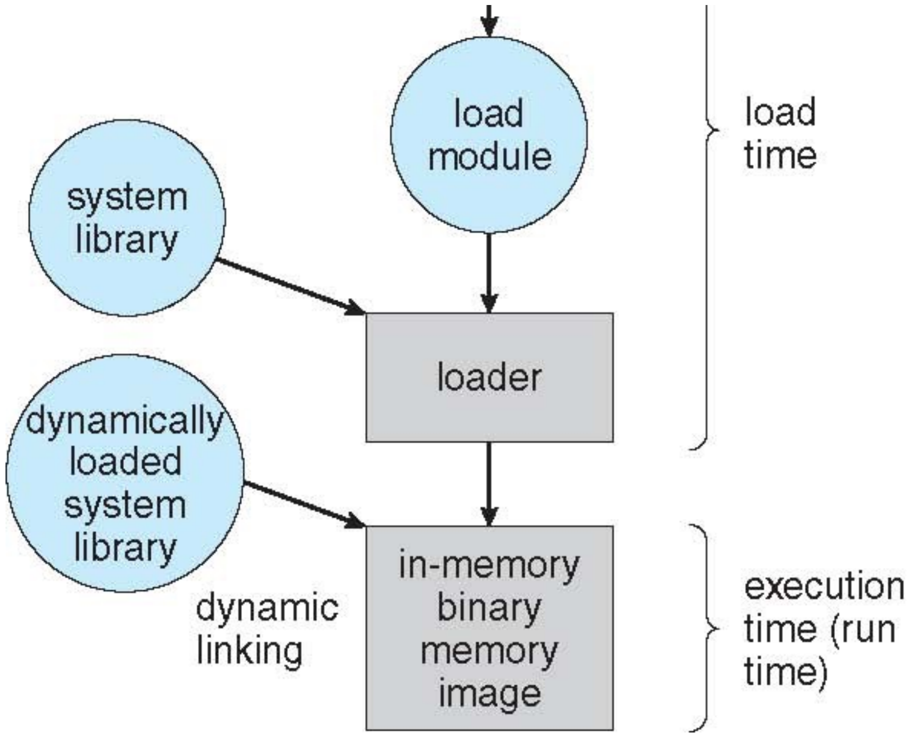
Binding of Instructions and Data to Memory

- 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.

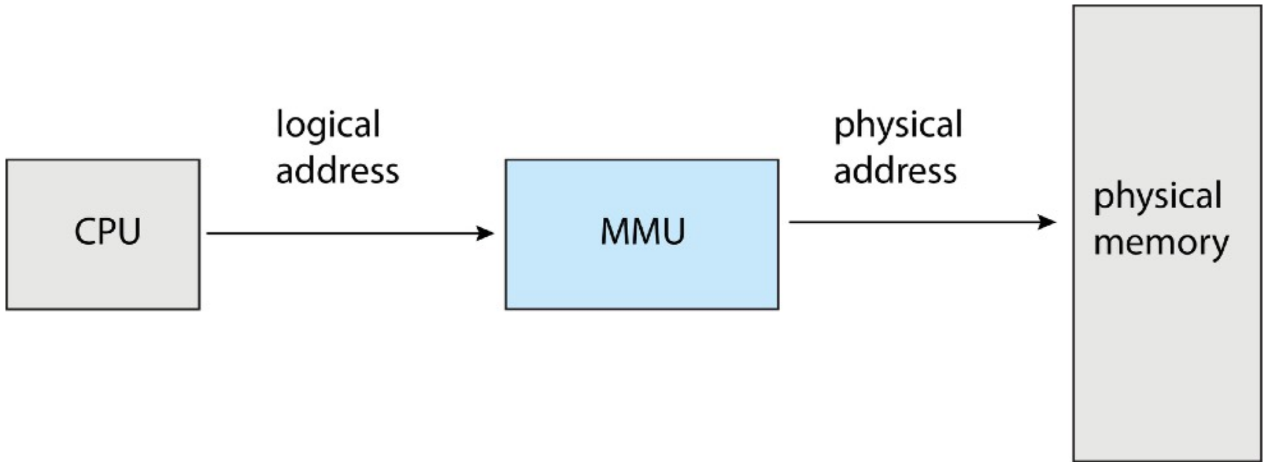


Binding of Instructions and Data to Memory

- **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another, which needs hardware support for address maps (e.g., base and limit registers).


load module
system library
loader
dynamically loaded system library
dynamic linking
in-memory binary memory image
load time
execution time (run time)

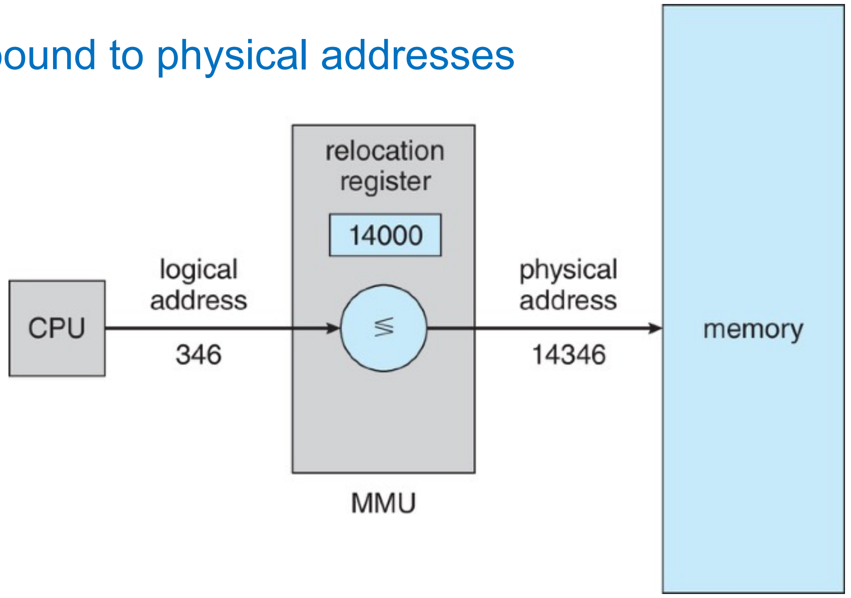
Logical vs. Physical Address Space



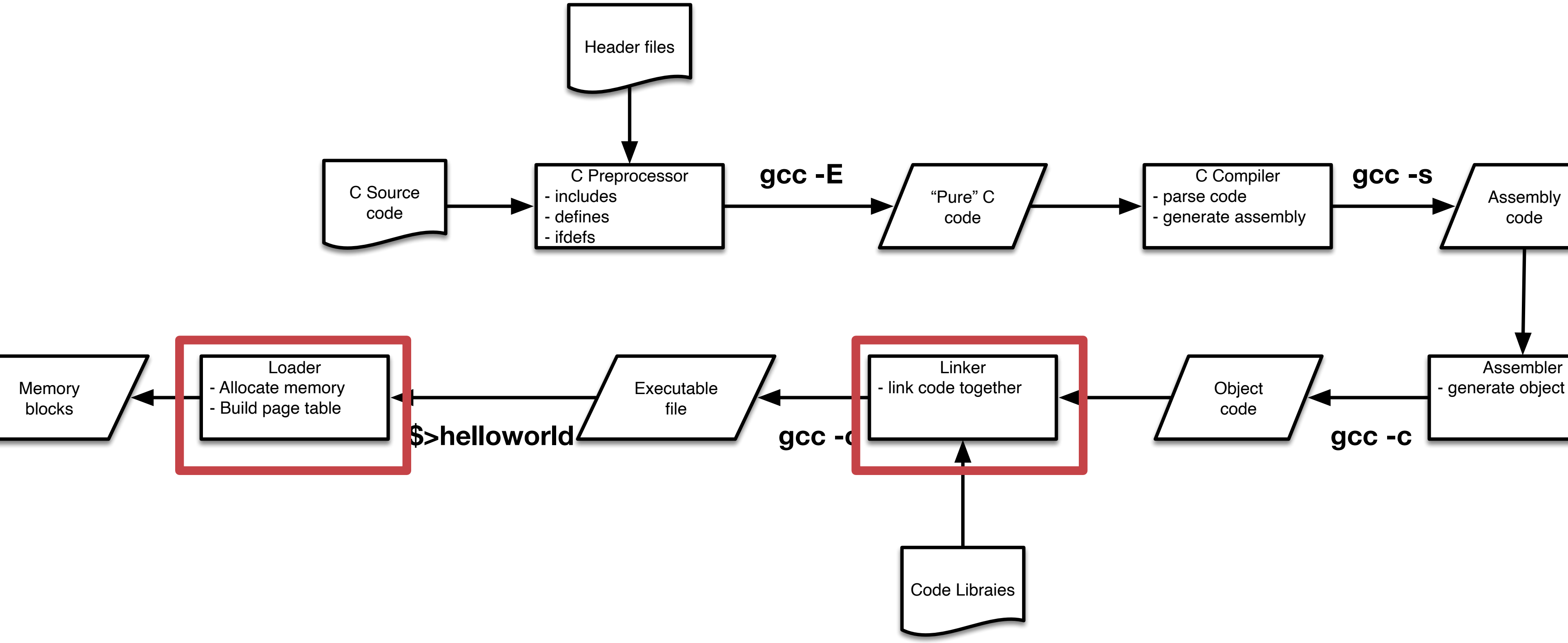
- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as virtual address
  - **Physical address** – address seen by the memory unit
- Logical and physical addresses 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 (Cont.)

- Consider simple scheme, which is a generalization of the base-register scheme.
  - The base register now called **relocation register**
- The value in the relocation register is added to every address generated by a user process at the time it is sent to memory
- The user program deals with logical addresses; it never sees the real physical addresses
  - Execution-time binding occurs when reference is made to location in memory
  - Logical address bound to physical addresses



Linking and Loading

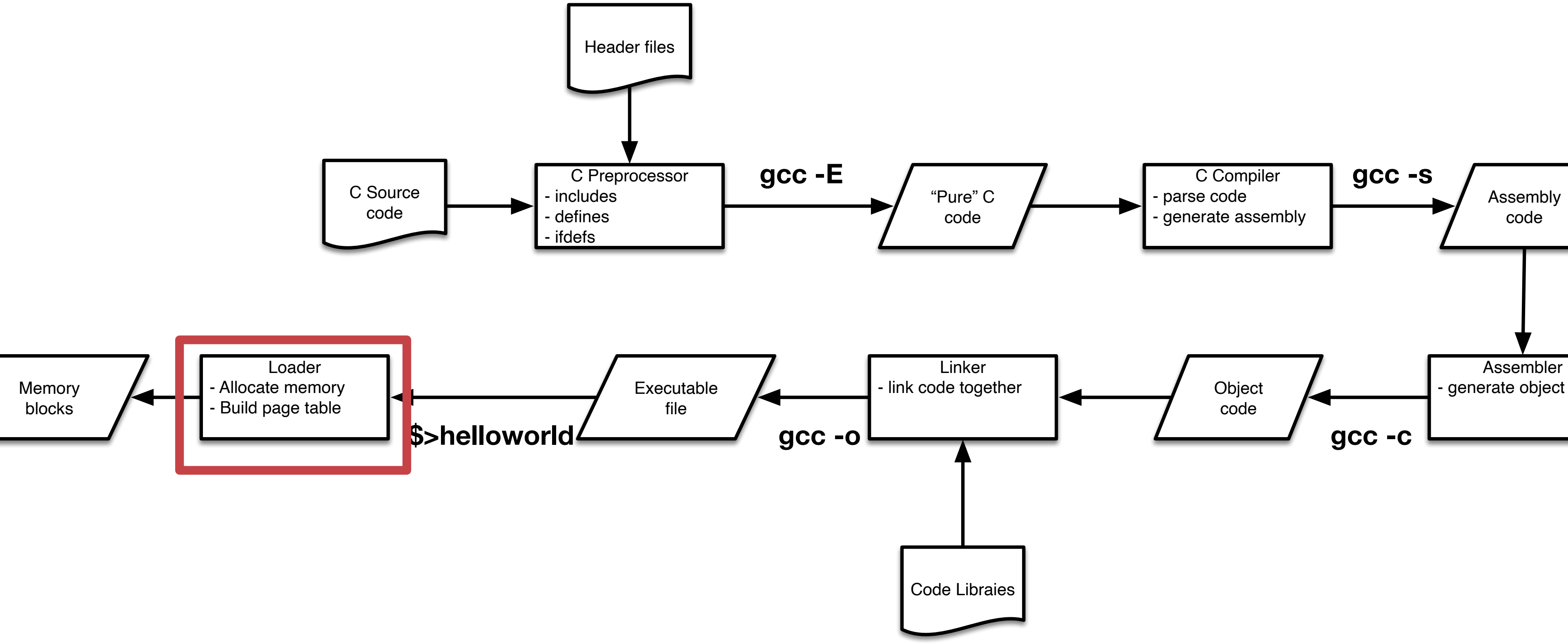


- Linking: Link multiple object files, and libraries into executable files.
- Loading: Open the program file, store them into memory, and notify the OS.

Linking

- Static linking – system libraries and program code combined by the loader into the binary program image
- 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 checks if routine is in processes' memory address
  - If not in address space, add to address space
- Dynamic linking is particularly useful for libraries, also known as shared libraries
- Consider applicability to patching system libraries: Versioning may be needed

Linking and Loading



- Linking: Link multiple object files, and libraries into executable files.
- Loading: Open the program file, store them into memory, and notify the OS.

Loading

- **Static Loading**:
  - The entire program does need to be loaded in memory to execute.
  - The size of memory will be limited by the size of physical memory.
- **Dynamic loading**:
  - Main program is loaded but routine is not loaded until it is called.
  - Better memory-space utilization; unused routine is never loaded
  - All routines kept on disk in relocatable load format.
  - 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
    - OS can help by providing libraries to implement dynamic loading

NEWS Lab
嵌入式系統暨無線網路實驗室

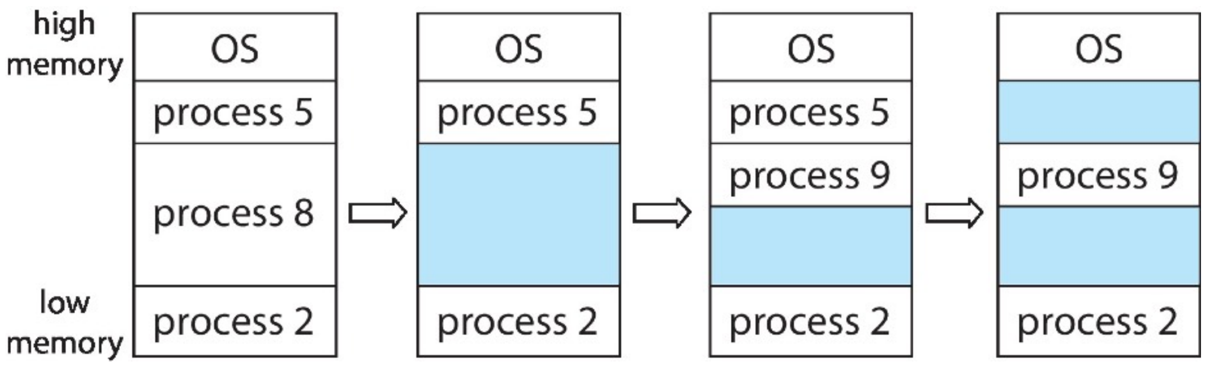
Contiguous Memory Allocation

Contiguous Memory Allocation

- Main memory must store both OS and user processes.
  - Because of limited resource, must allocate efficiently.
  - Contiguous allocation is one (early) method.
- Main memory usually is divided into two partitions for:
  - **Operating system**: usually held in low memory with interrupt vector
  - **User processes**: usually held in high memory
    - Each process contained in single contiguous section of memory

Variable-size Partition

- Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions.
  - Variable-partition sizes for efficiency (sized to a given process' needs).
  - When a process **arrives**, it is allocated memory from a partition large enough to accommodate it
  - When a process **terminates**,
    - the memory is released as a free partition.
    - Adjacent free partitions can be combined to enlarge the partition.
  - **Hole** – block of available memory; holes of various size are scattered throughout memory.
  - Operating system maintains information for a) allocated partitions b) free partitions(hole)



NEWS Lab 嵌入式系統暨無線網路實驗室

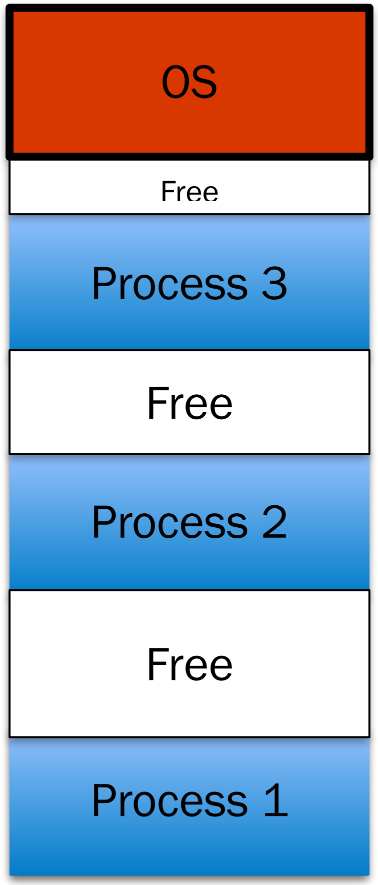
Dynamic Storage-Allocation Problem

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
  - First-fit and best-fit better than worst-fit in terms of speed and storage utilization

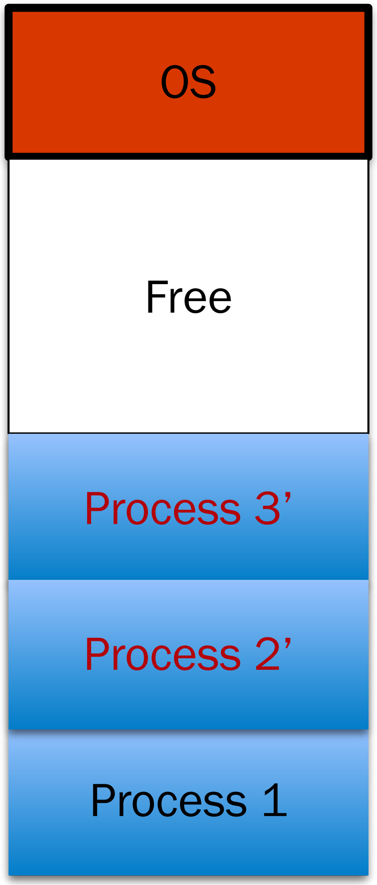
Fragmentation

- After the memory are allocated and freed, the memory is very likely to be partitioned into numbers of small holes, each of which cannot fit a single process.
- However, the sum of the free partitions are greater than the required memory size of the new process.
- This scenario is called ***fragmentation***.
  - To avoid fragmentation, the operating system can compact number of small free partitions into bigger ones.
  - How to do it?



Fragmentation

- After the memory are allocated and freed, the memory is very likely to be partitioned into numbers of small holes, each of which cannot fit a single process.
- However, the sum of the free partitions are greater than the required memory size of the new process.
- This scenario is called ***fragmentation***.
  - To avoid fragmentation, the operating system can compact number of small free partitions into bigger ones.
  - How to do it?
    - In order to merge non-adjacent holes into one, OS needs to move occupied partitions. In addition,
    - the physical address of data and instructions, which are generated by loader, must be changed, i.e., relocated.

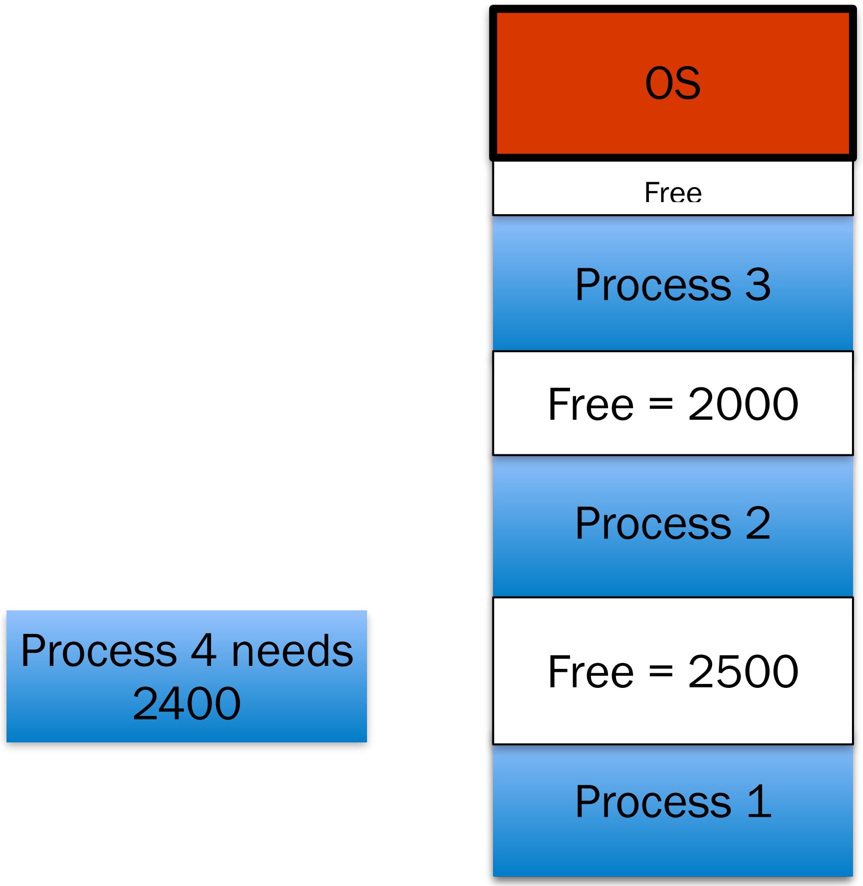


Fragmentation (Cont.)

- 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
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers
- Now consider that backing store has same fragmentation problems

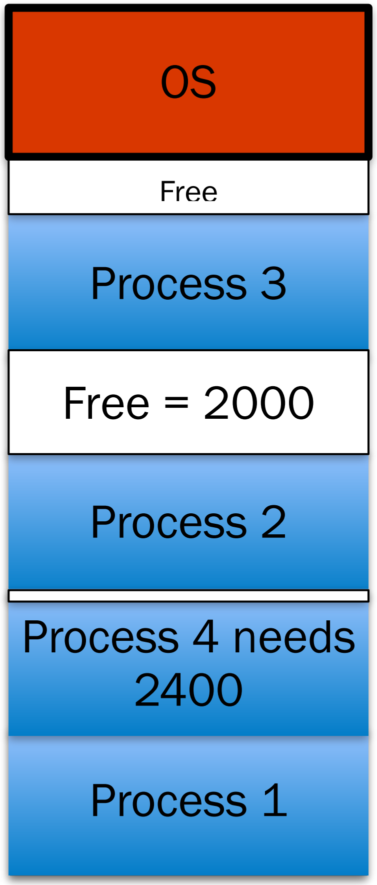
Fragmentation

- **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
- First fit analysis reveals that given *N* blocks allocated, 0.5*N* blocks lost to fragmentation
  - 1/3 may be unusable -> 50-percent rule



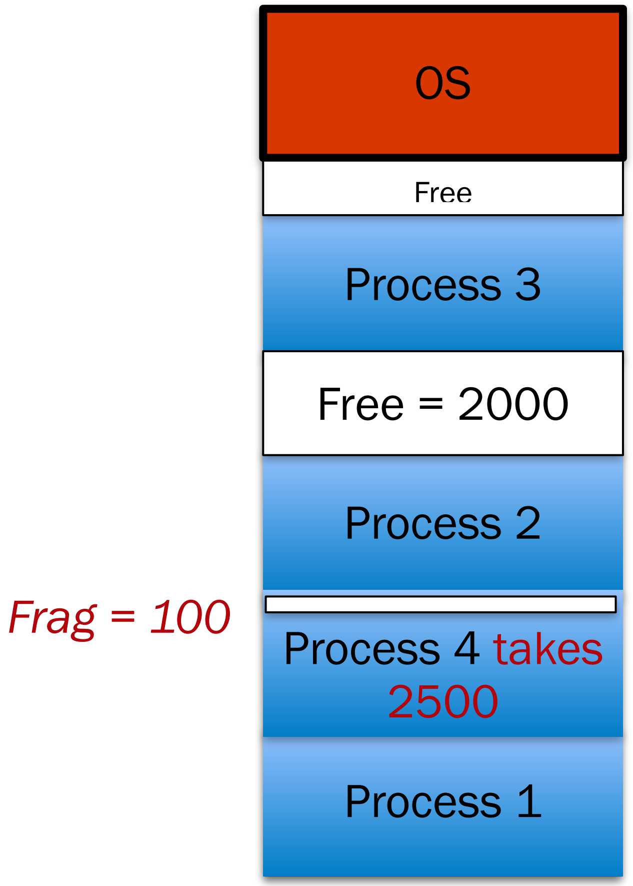
Fragmentation

- **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
- First fit analysis reveals that given *N* blocks allocated, 0.5*N* blocks lost to fragmentation
  - 1/3 may be unusable -> 50-percent rule



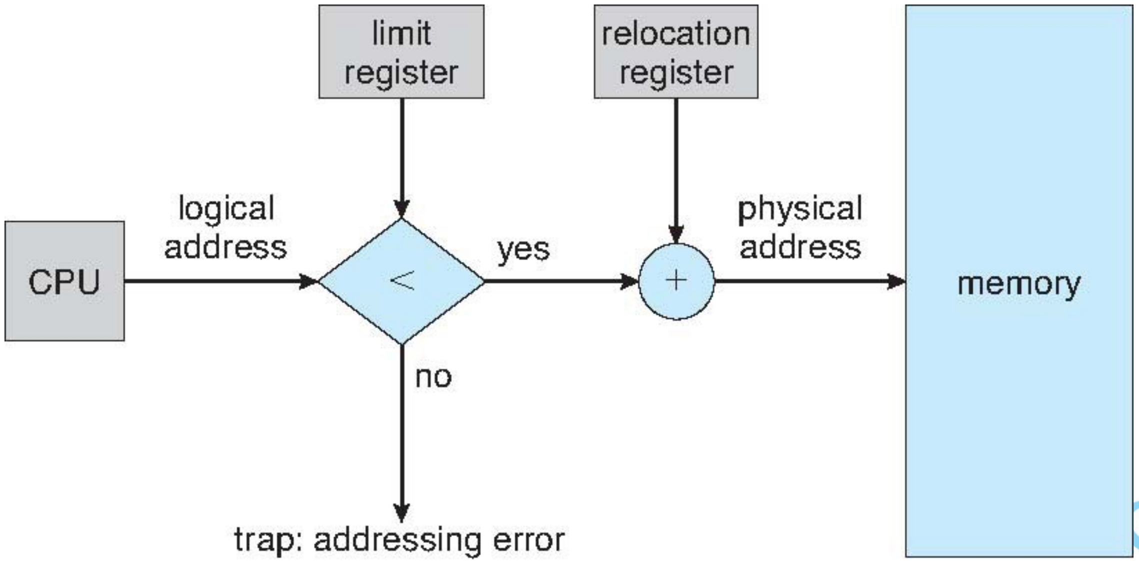
Fragmentation

- **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
- First fit analysis reveals that given *N* blocks allocated, 0.5*N* blocks lost to fragmentation
  - 1/3 may be unusable -> 50-percent rule



Memory Re-location

- Relocation registers used to protect user processes from each other, and from changing operating-system code and data.
  - Base register contains value of smallest physical address.
  - Limit register contains range of logical addresses – each logical address must be less than the limit register.
  - MMU maps logical address dynamically.
  - Can then allow actions such as kernel code being transient and kernel changing size.



Paging

How to eliminate fragmentation or reduce its impact?

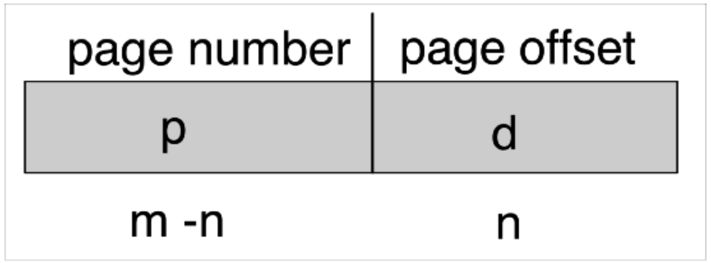
NEWS Lab
嵌入式系統暨無線網路實驗室

Paging

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
  - Avoids external fragmentation
  - Avoids problem of varying sized memory chunks
- Divide <u>physical memory</u> into fixed-sized blocks called ***frames***
  - Size is power of 2, between 512 bytes and 16 Mbytes
- Divide <u>logical memory</u> into blocks of same size called ***pages***
- Keep track of all free frames, rather than free holes.
- To run a program of *N* pages, need to find N free frames in order to load program
- Set up a page table to translate logical to physical addresses

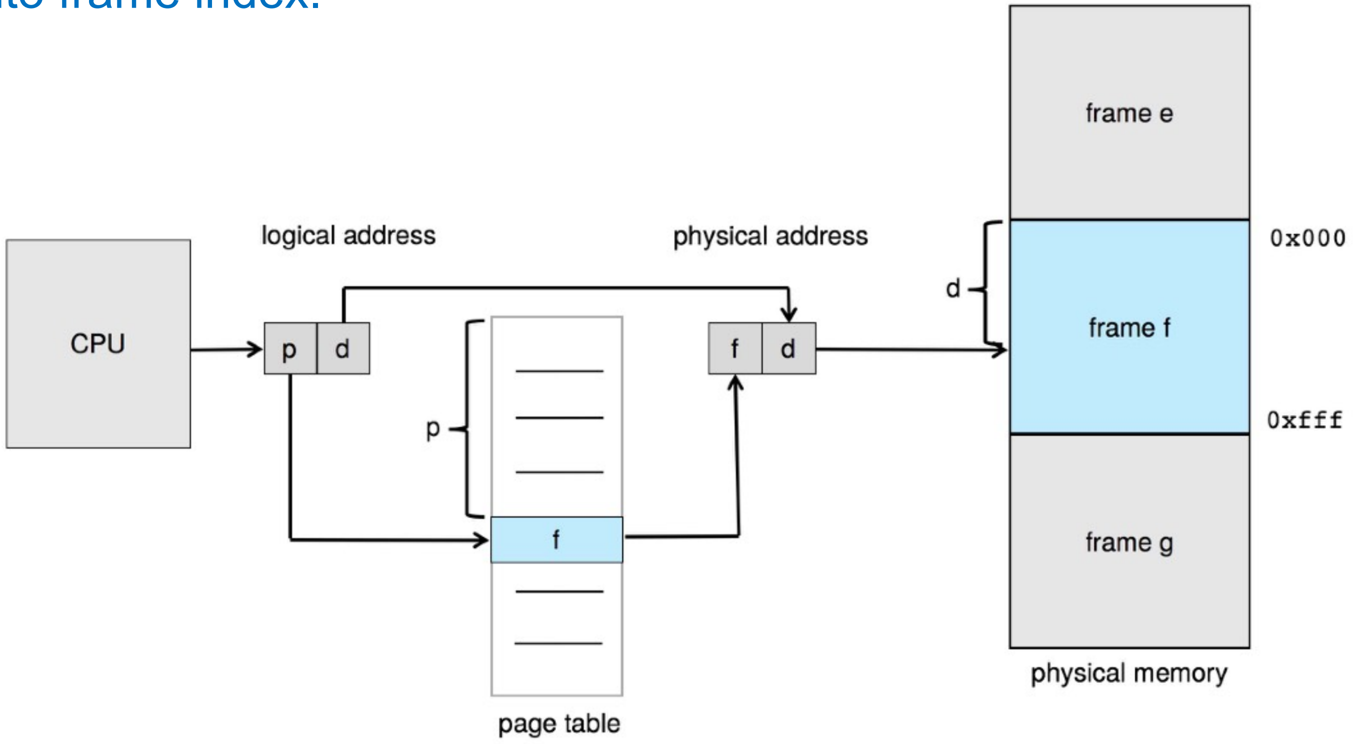
Address Translation Scheme

- Address generated by compiler/loader is divided into:
  - **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.
- For given the size of logical address space is 2^m and page size is 2^n bytes.

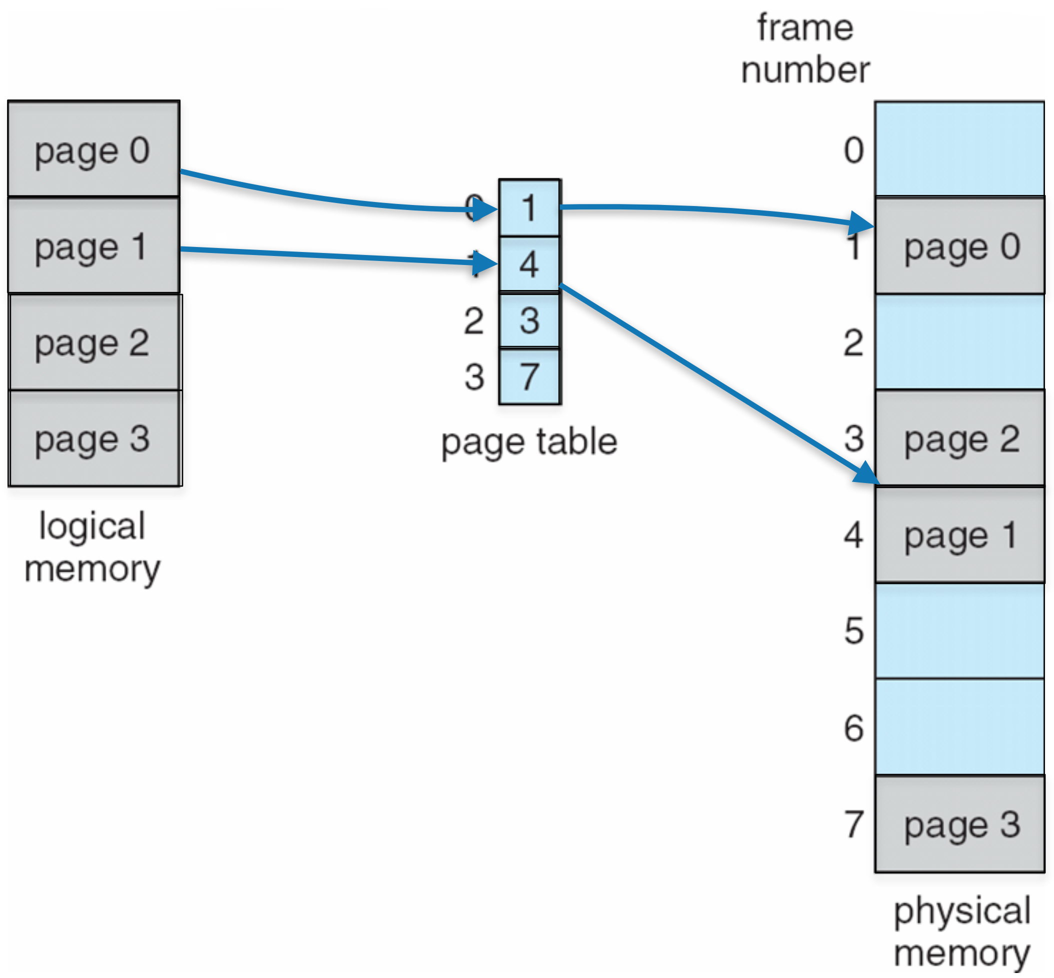


Paging Hardware

- Assume that page size on virtual memory is equal to frame size on physical memory.
- To convert virtual address to physical address, the page table maps the page index into frame index.



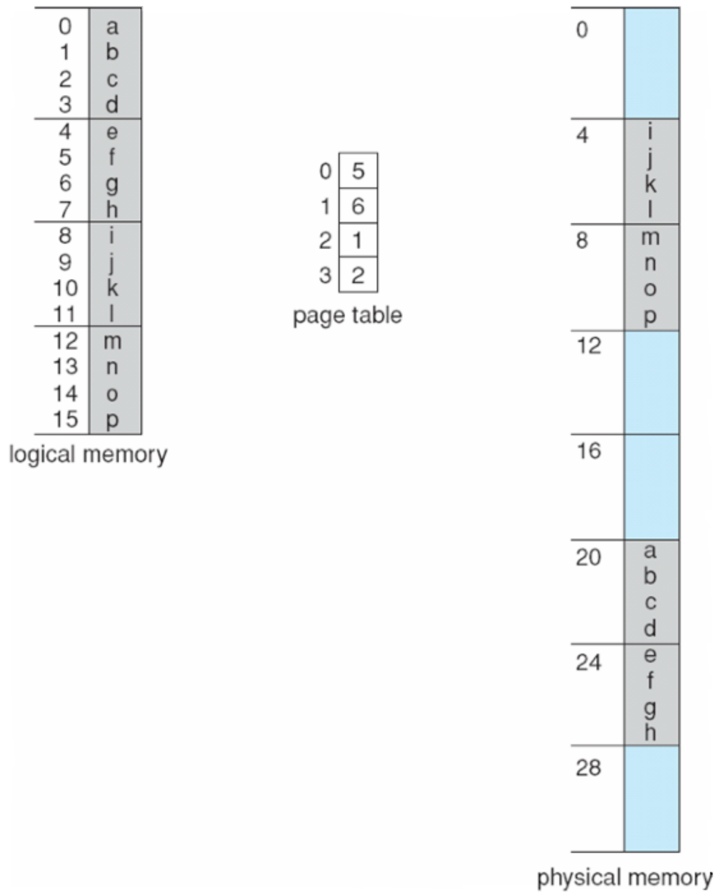
Example of Paging Model of Logical and Physical Memory



Paging Example

- Logical address: n = 2 and m = 4. Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages)

Logical address 0 maps to physical address 20 [= (5 * 4) + 0].

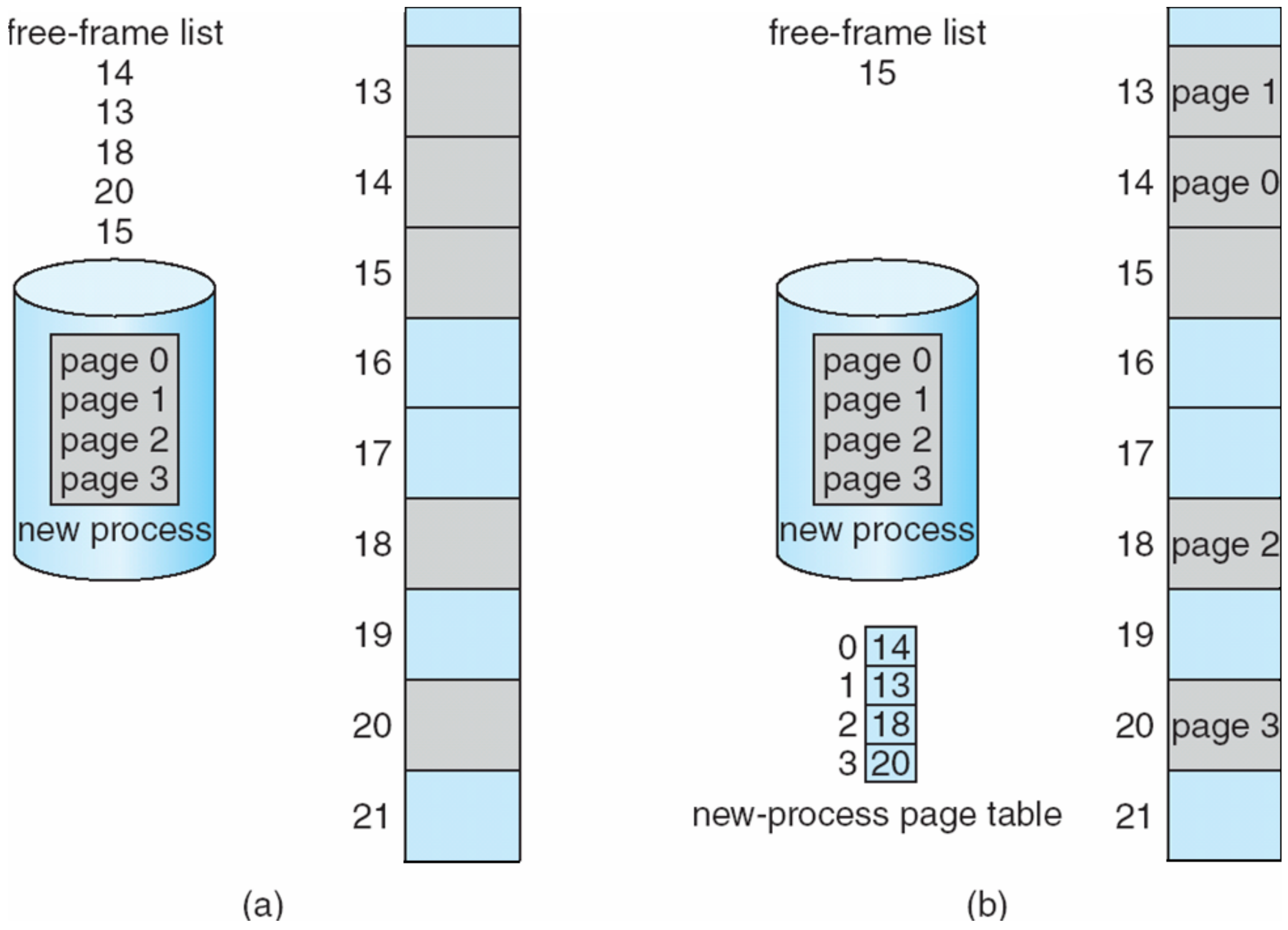


Logical Address	Physical Address
0 = 4 * 0 + 0	4 * 5 + 0 = 20
1	21
2	22
3	23
4 = 4 * 1 + 0	4 * 6 + 0 = 24
5	25
6	26
7	27
8 = 4 * 2 + 0	4 * 1 + 0 = 4
9	5
10	6

Calculating internal fragmentation

- Page size = 2,048 bytes and process memory size = 72,766 bytes
  - 35 pages + 1,086 bytes are required.
  - Internal fragmentation of 2,048 - 1,086 = 962 bytes
- Worst case fragmentation = 1 frame – 1 byte
- On average fragmentation = 1 / 2 frame size
- Is small frame sizes desirable? Good for easy internal fragmentation.
  - However, each page table entry takes memory to track
  - Page sizes growing over time
    - Solaris supports two page sizes – 8 KB and 4 MB

Free Frames



Before allocation

After allocation

Implementation of Page Table

- Page table is kept in **main memory**
  - Page-table base register (PTBR) points to the page table.
  - Page-table length register (PTLR) indicates size of the page table.
- In this scheme every data/instruction access requires two memory accesses
  - One for the page table and one for the data/instruction.
  - When one memory fetch takes 10ns and one instruction takes 0.28 ~ 0.84 ns (on 3.5GHz single core processor).
    - The time to execute one instruction will be 0.28 + 10*2 = 2.28, in which only 12% of time is used for computation.
- The two-memory access problem can be solved by the use of a special fast-lookup hardware cache called translation look-aside buffers (TLBs) (also called associative memory).

Estimate Page Table Size

- Consider a system with a 32-bit logical address space. Assuming that each entry consists of 4 bytes and the page size in such a system is 4 KB (2^{12}), each process may need up to 4 MB of physical address space for the page table alone.

  - Logical Address = 32 bits, Logical Address space = 2^{32} bytes,
    Page size = 4 KB = 2^{12} Bytes
  - Page offset = 12
  - Number of bits in a page = Logical Address - Page Offset = 32 - 12 = 20 bits
  - Number of pages = 2^{20} = 1 M
  - Page table entry = 4 Byte
  - Therefore, the size of the page table = 1M X 4 Byte = 4 MB

NEWS Lab
嵌入式系統暨無線網路實驗室

Estimate Page Table Size

- Consider a system with a **64**-bit logical address space. Assuming that each entry consists of 4 bytes and the page size in such a system is 4 KB (2^{12}), each process may need up to ??? of physical address space for the page table alone.

Translation Look-Aside Buffer

- Some TLBs store address-space identifiers (ASIDs) in each TLB entry – uniquely identifies each process to provide address-space protection for that process
    - Otherwise need to flush at every context switch
- TLBs typically small (64 to 1,024 entries)
- On a TLB miss, value is loaded into the TLB for faster access next time
    - Replacement policies must be considered.
    - Some entries can be wired down for permanent fast access.

TLB Hardware

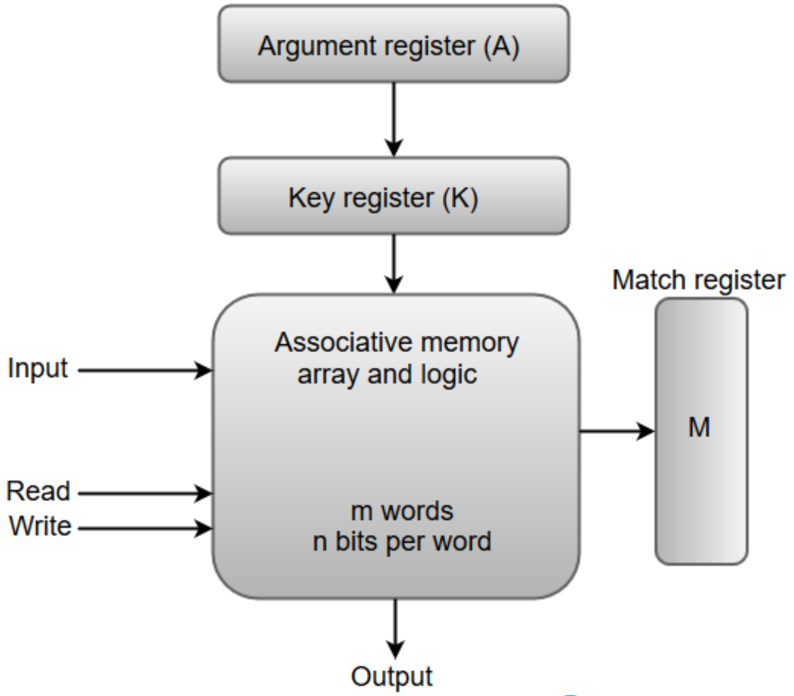
- Associative memory – parallel search for (key, value) pairs, where Key is Page# in TLB.

Page #	Frame #

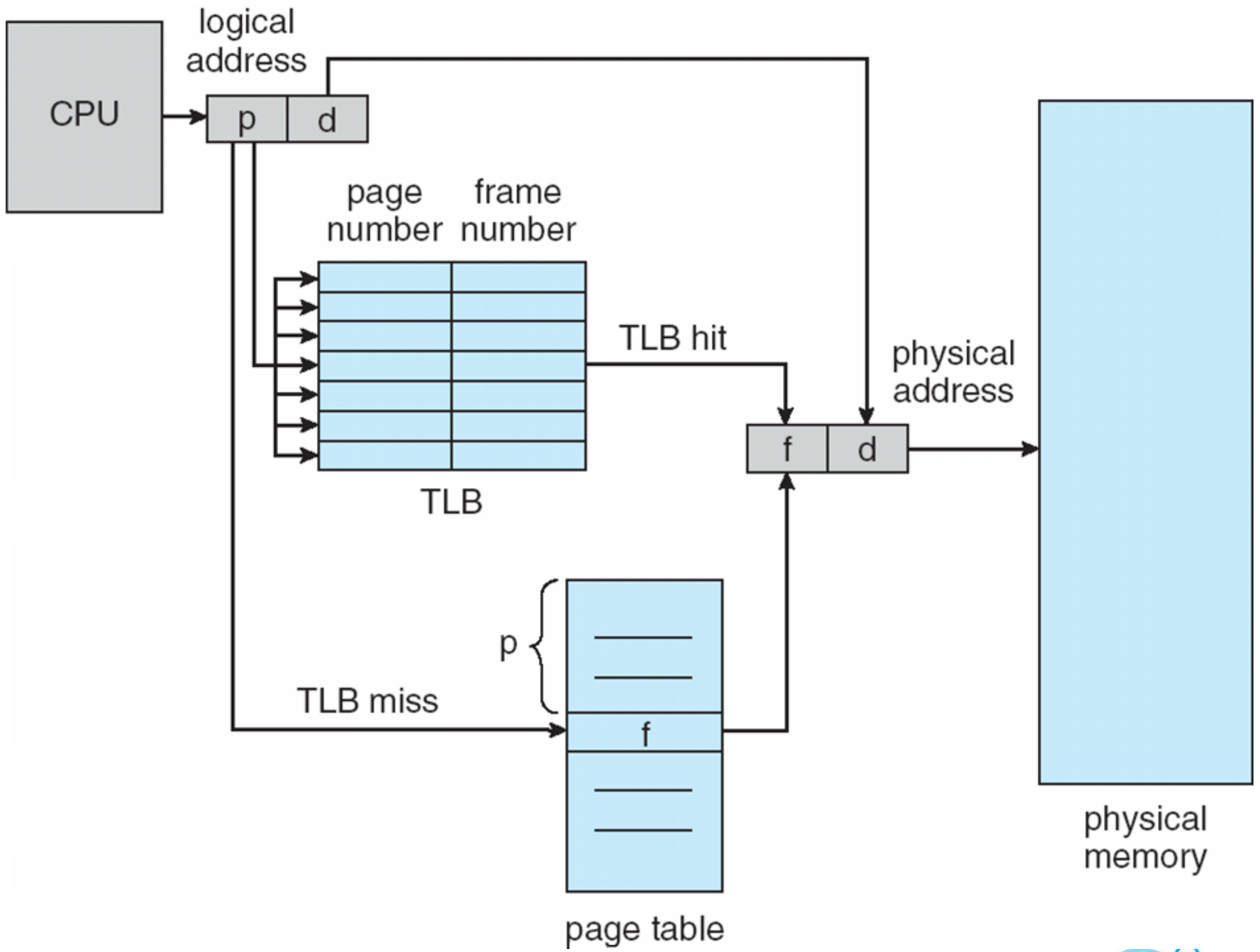
- Address translation (p, d)
  - If p is in associative register, get frame # out
  - Otherwise get frame # from page table in memory

An *associative memory* can be considered as a memory unit whose stored data can be identified for access by the *content* of the data itself rather than by an address or memory location.

Unlike standard computer memory, random access memory (RAM), in which the user supplies a *memory address* and the RAM returns the data word stored at that address, a CAM is designed such that the user supplies a data word and the CAM searches its entire memory to see if that data word is stored anywhere in it.



Paging Hardware With TLB



Effective Access Time

- Hit ratio – percentage of times that a page number is found in the TLB
  - An 80% hit ratio means that we find the desired page number in the TLB 80% of the time.
- Suppose that 10 nanoseconds to access memory.
  - If we find the desired page in TLB, then a mapped-memory access take 10 ns.
  - Otherwise we need two memory access so it is 20 ns
- Effective Access Time (EAT)

  EAT = 0.80 x 10 + 0.20 x 20 = 12 nanoseconds

  implying 20% slowdown in access time
- Consider a more realistic hit ratio of 99%,

  EAT = 0.99 x 10 + 0.01 x 20 = 10.1ns

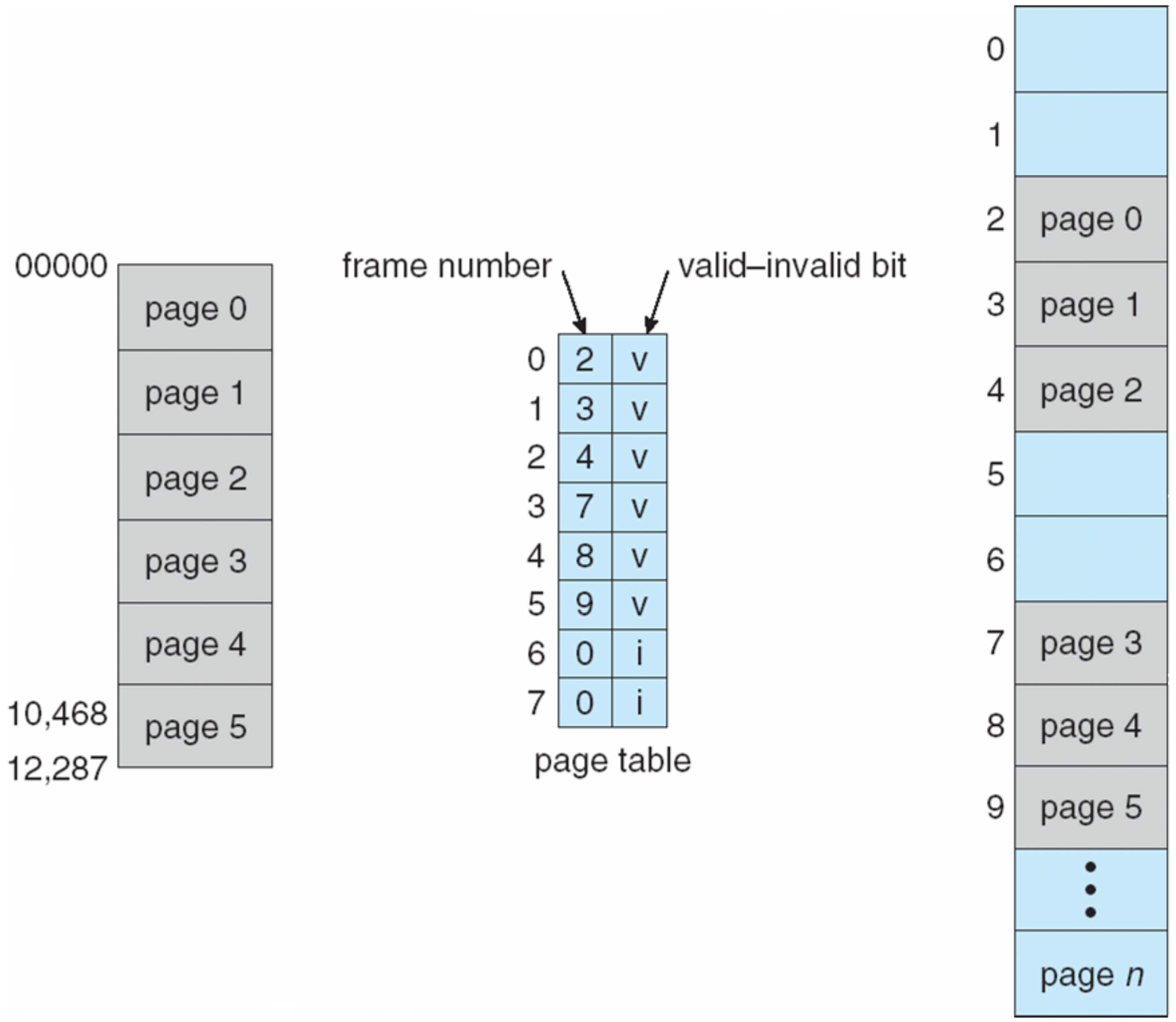
  implying only 1% slowdown in access time.

NEWS Lab 嵌入式系統暨無線網路實驗室

Memory Protection

- Memory protection implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed
  - Can also add more bits to indicate page execute-only, and so on
- Valid-invalid bit attached to each entry in the page table:
  - "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page
  - "invalid" indicates that the page is not in the process' logical address space
  - Or use page-table length register (PTLR)
- Any violations result in a trap to the kernel

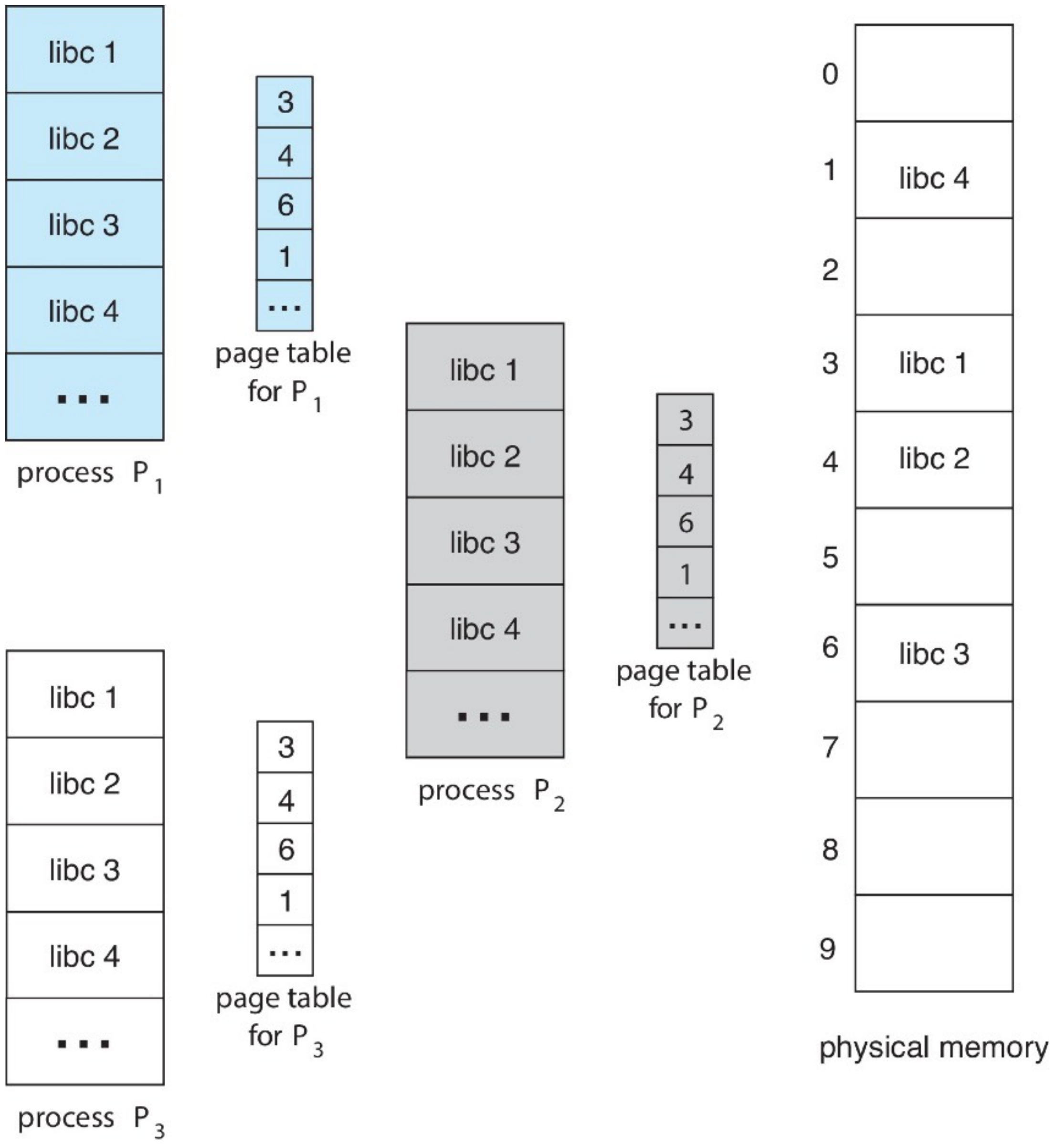
Valid (v) or Invalid (i) Bit In A Page Table



Shared Page

- **Shared code**
  - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems)
  - Similar to multiple threads sharing the same process space
  - Also useful for interprocess communication if sharing of read-write pages is allowed
- **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



Structure of Paging Table

Structure of the Page Table

- Memory structures for paging can get huge using straight-forward methods
  - Consider a 32-bit logical address space as on modern computers and page size of 4 KB (2^{12}).
  - Each process requires 4 MB of physical address space for the page table alone.
  - The simple solution is to divide the page table into smaller units
    - Hierarchical Paging
    - Hashed Page Tables
    - Inverted Page Tables

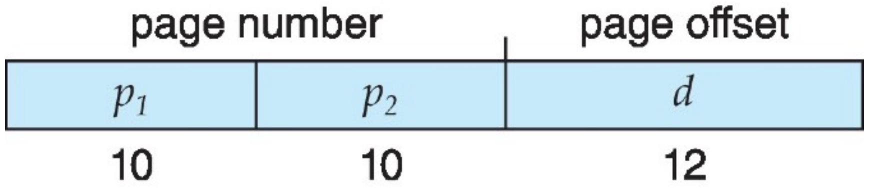
How much memory we need to store page table for 64bits computer?

Hierarchical Page Tables

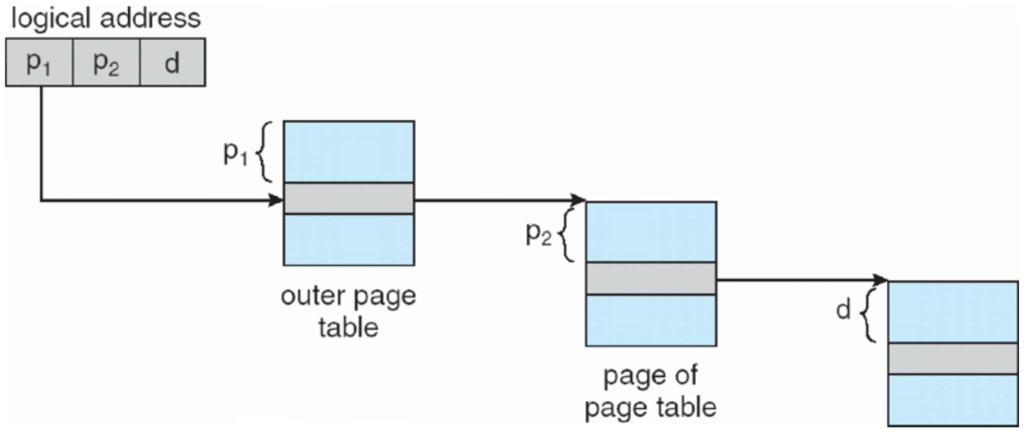
- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table to page the page table

Two-Level Paging Example

- Since the page table is paged, the page number is further divided into:
  - a 20-bit page number
  - a 12-bit page offset (4K page size)
- Thus, a logical address is as follows:

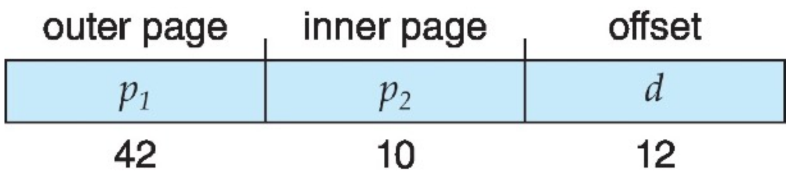


where p_1 is an index into the ***outer page table***, and p_2 is the displacement within the page of the ***inner page table***, known as **forward-mapped page table**



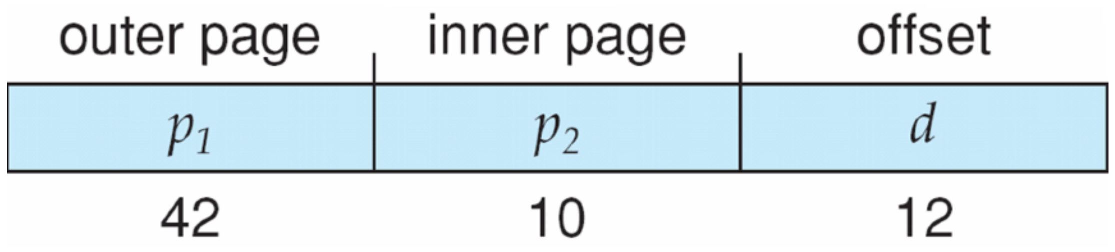
64-bit Logical Address Space

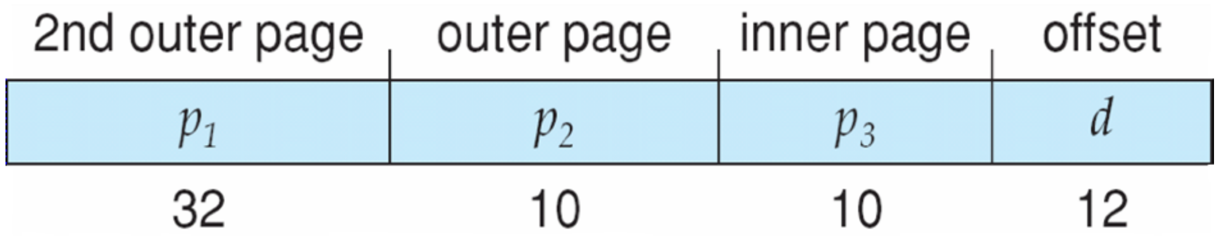
- Even two-level paging scheme is not sufficient
- If page size is 4 KB (2^{12}), the page table has 2^{52} entries
  - In two level scheme, inner page tables, whose sizes are 4KB, should have 2^{10} 4-byte entries.
  - Address would look like



  - Outer page table has 2^{42} entries or 2^{44} bytes
  - One solution is to add a 2nd outer page table
  - But in the following example the 2nd outer page table is still 2^{34} bytes in size
    - And possibly 4 memory access to get to one physical memory location

Multiple-level Paging Scheme



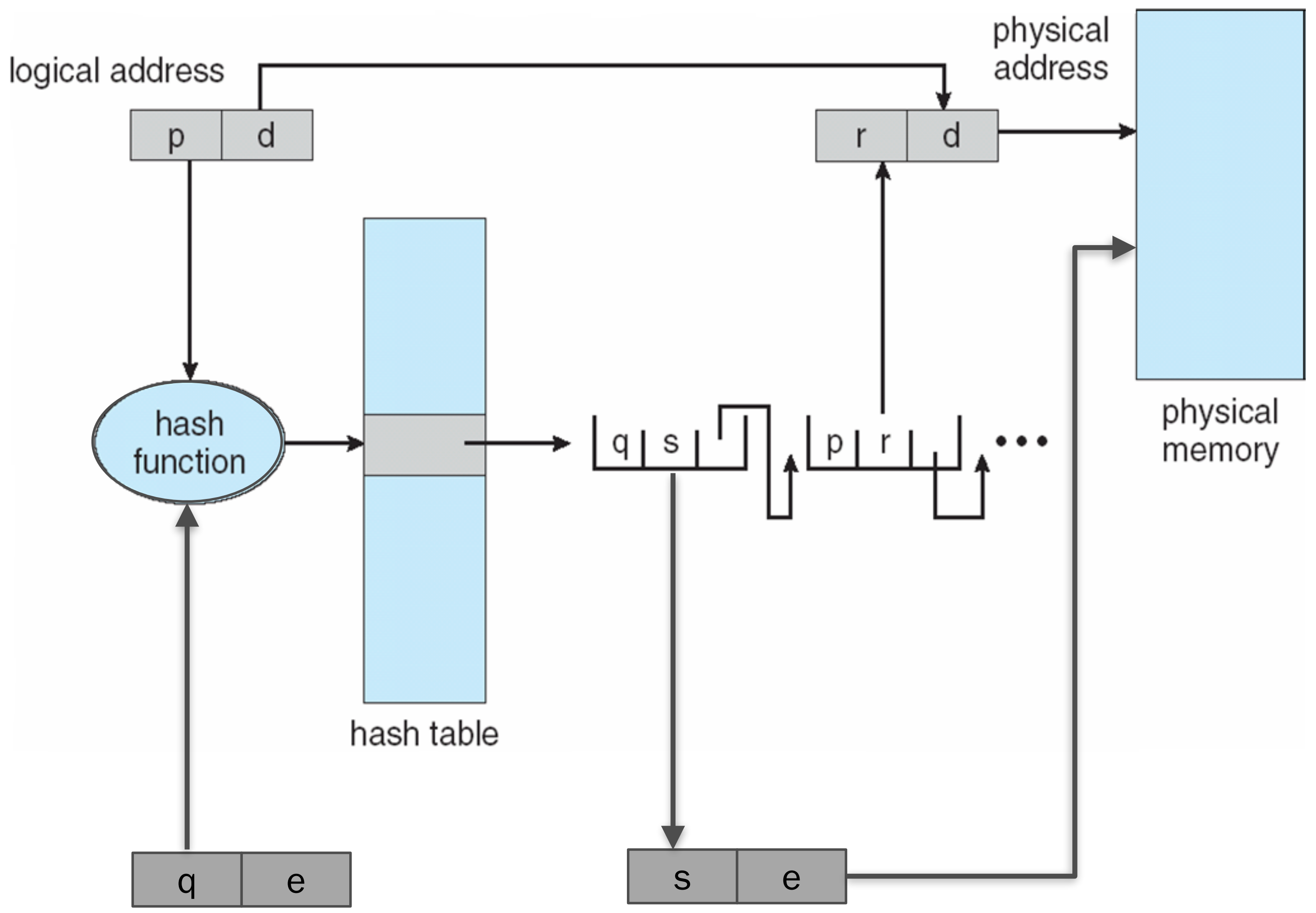


- The 64-bit UltraSPARC would require **seven** levels of paging—a prohibitive number of memory accesses to translate each logical address.
- In summary, for 64-bit architectures, hierarchical page tables are generally considered ***inappropriate***.

Hashed Page Tables

- Common when the memory spaces are addressed by more than 32 bits.
- The virtual page number is hashed into a page table to avoid ***huge*** table
  - This page table contains a chain of elements hashing to the same location
  - Each element contains
    - the virtual page number (for matching),
    - the value of the mapped page frame (for mapping), and
    - a pointer to the next element
- Virtual page numbers are compared in this chain searching for a match
  - If a match is found, the corresponding physical frame is extracted

Hashed Page Table

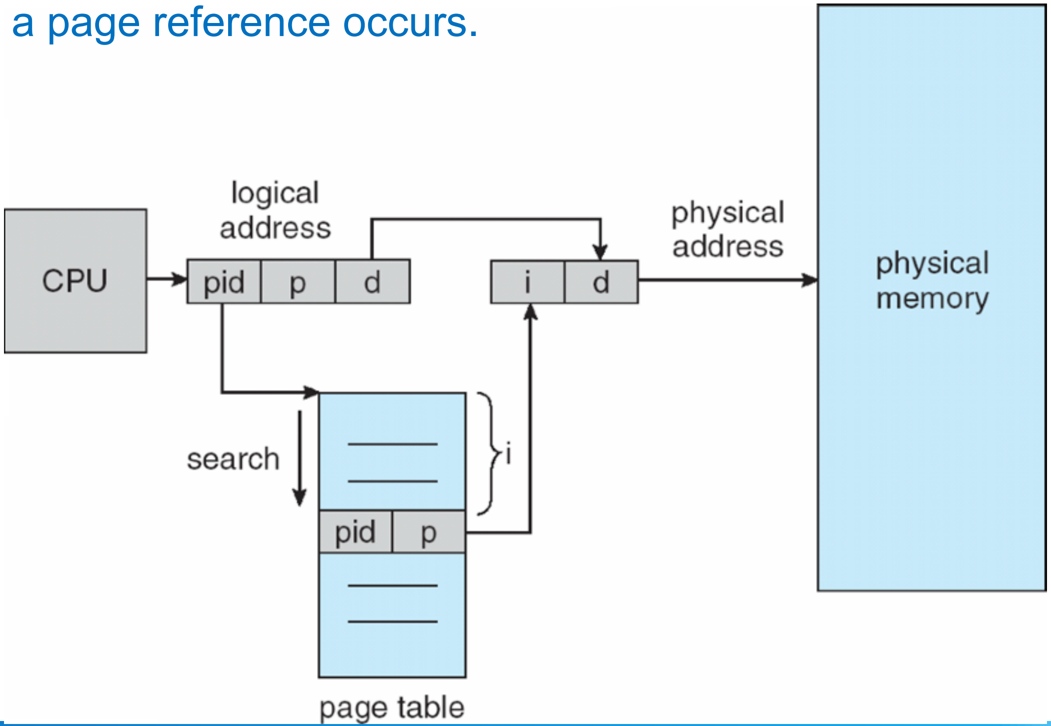


Hashed Page Tables

- Variation for 64-bit addresses is clustered page tables
  - Similar to hashed but each entry refers to several pages (such as 16) rather than 1
  - Especially useful for sparse address spaces (where memory references are non-contiguous and scattered)

Inverted Page Table

- Rather than each process having a page table and keeping track of all possible logical pages, **track all physical pages.**
  - One entry for each real page of memory
  - Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs.



Inverted Page Table

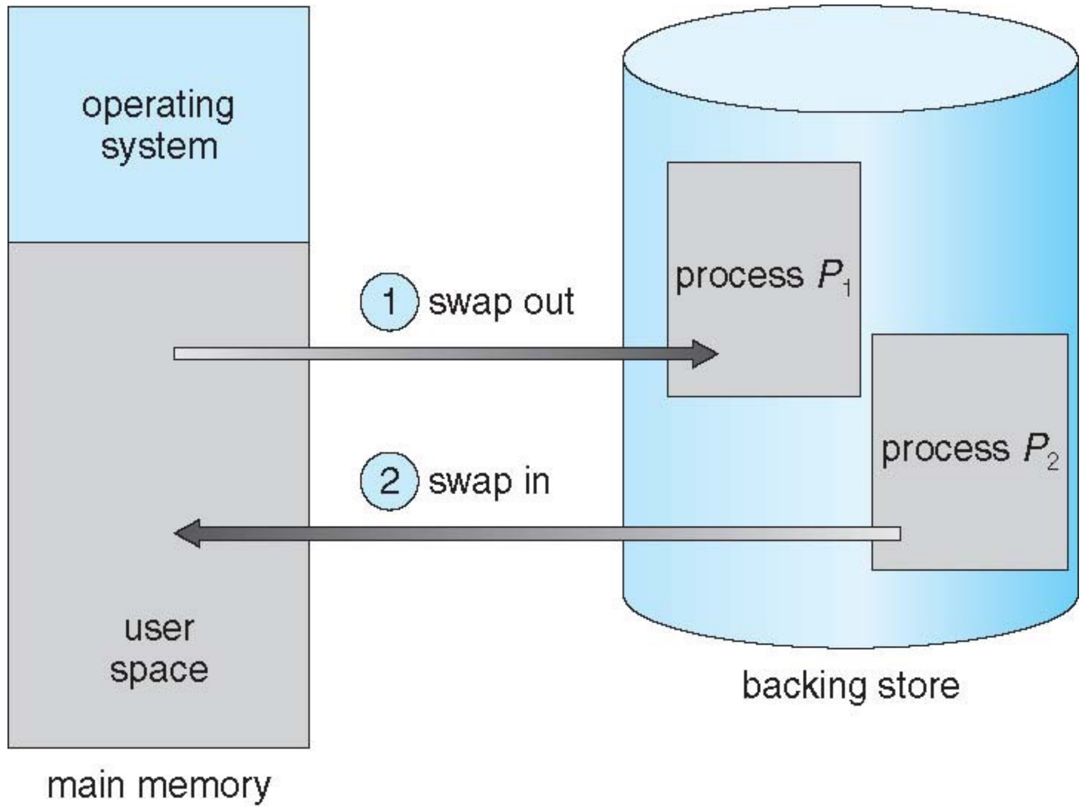
- Use hash table to limit the search to one — or at most a few — page-table entries
  - TLB can accelerate access
- But how to implement shared memory?
  - One mapping of a virtual address to the shared physical address

Swapping

Swapping

- When the total memory space requirements of processes exceeds physical memory, 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.
- System maintains a ready queue of ready-to-run processes which have memory images on disk.

Schematic View of Standard Swapping



- The process memory is stored in a continuous address space, which is reserved by the operating systems for fast and ***sequential*** access.
- The method is ***not*** used in modern operating systems to support ***memory paging***.

Swapping (Cont.)

- Does the swapped out process need to swap back in to same physical addresses?
- Depends on address binding method
  - Plus consider pending I/O to / from process memory space
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
  - (Standard) Swapping normally disabled
  - Started if more than threshold amount of memory allocated
  - Disabled again once memory demand reduced below threshold

NEWS Lab 嵌入式系統暨無線網路實驗室

Context Switch Time including Swapping

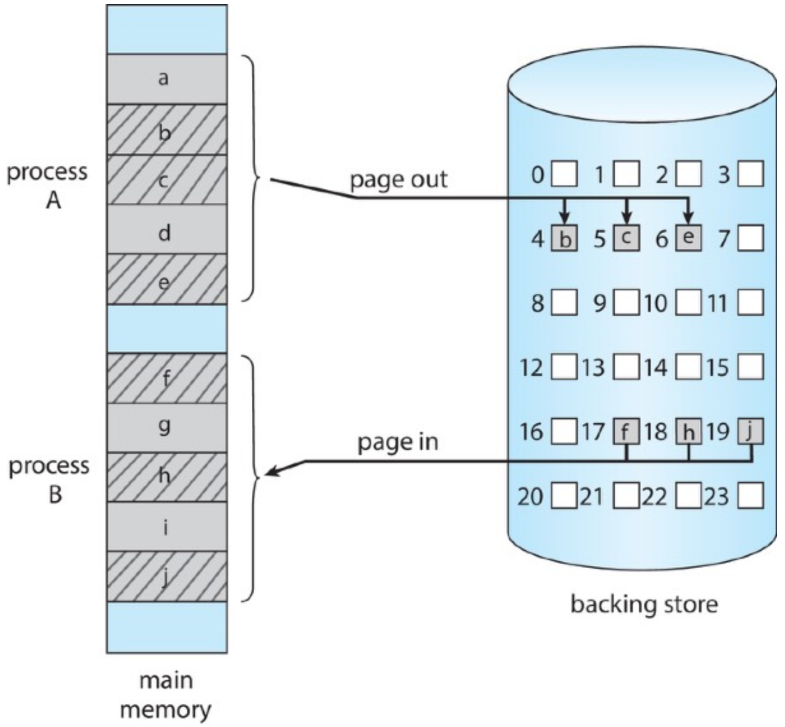
- If next processes to be put on CPU is not in memory, need to swap out a process and swap in target process.
- Context switch time can then be very high
- 100MB process swapping to hard disk with transfer rate of 50MB/sec
  - Swap out time of 2000 ms
  - Plus swap in of same sized process
  - Total context switch swapping component time of 4000ms (4 seconds)
- Can reduce if reduce size of memory swapped – by knowing how much memory really being used
  - System calls to inform OS of memory use via request_memory() and release_memory()

Context Switch Time and Swapping (Cont.)

- Other constraints as well on swapping
  - Pending I/O – can’t swap out as I/O would occur to wrong process
  - Or always transfer I/O to kernel space, then to I/O device
    - Known as double buffering, adds overhead
- Standard swapping not used in modern operating systems
  - But modified version common
    - Swap only when free memory extremely low

Swapping with Paging

- Swapping contiguous memory for processes are challenging.
  - Need to find the victim, which uses similar memory, has lower priority, and not ready to run.
  - Otherwise, the fragmentation will become worse.
- Swapping with paging can avoid the aforementioned issues.



NEWS Lab 嵌入式系統暨無線網路實驗室

Example: The Intel 32 and 64-bit Architectures

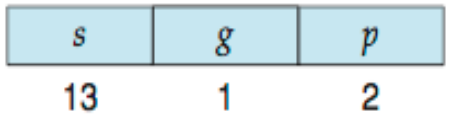
- Dominant industry chips
- Pentium CPUs are 32-bit and called IA-32 architecture
- Current Intel CPUs are 64-bit and called IA-64 architecture
- Many variations in the chips, cover the main ideas here

Example: The Intel IA-32 Architecture

- Supports both segmentation and segmentation with paging
  - Each segment can be 4 GB
  - Up to 16 K segments per process
  - Divided into two partitions
    - First partition of up to 8 K segments are private to process (kept in local descriptor table (LDT))
    - Second partition of up to 8K segments shared among all processes (kept in global descriptor table (GDT))

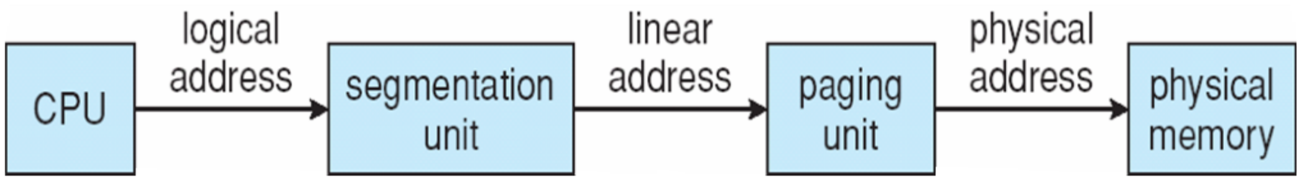
Example: The Intel IA-32 Architecture (Cont.)

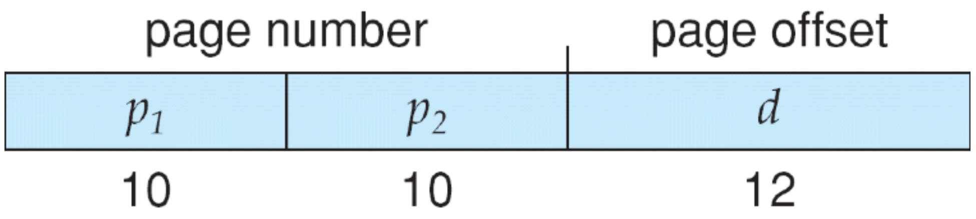
- CPU generates logical address
  - Selector given to segmentation unit which produces linear addresses



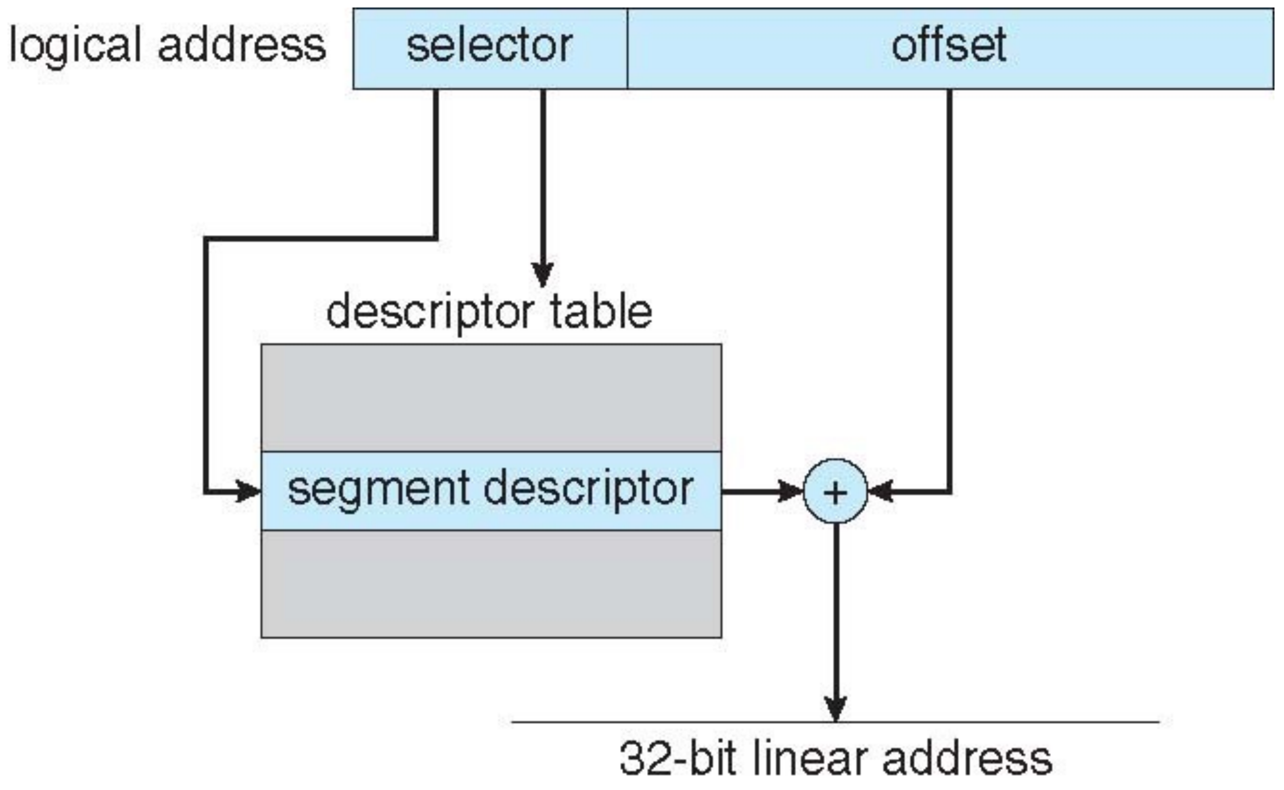
  - Linear address given to paging unit
    - Which generates physical address in main memory
    - Paging units form equivalent of MMU
    - Pages sizes can be 4 KB or 4 MB

Logical to Physical Address Translation in IA-32

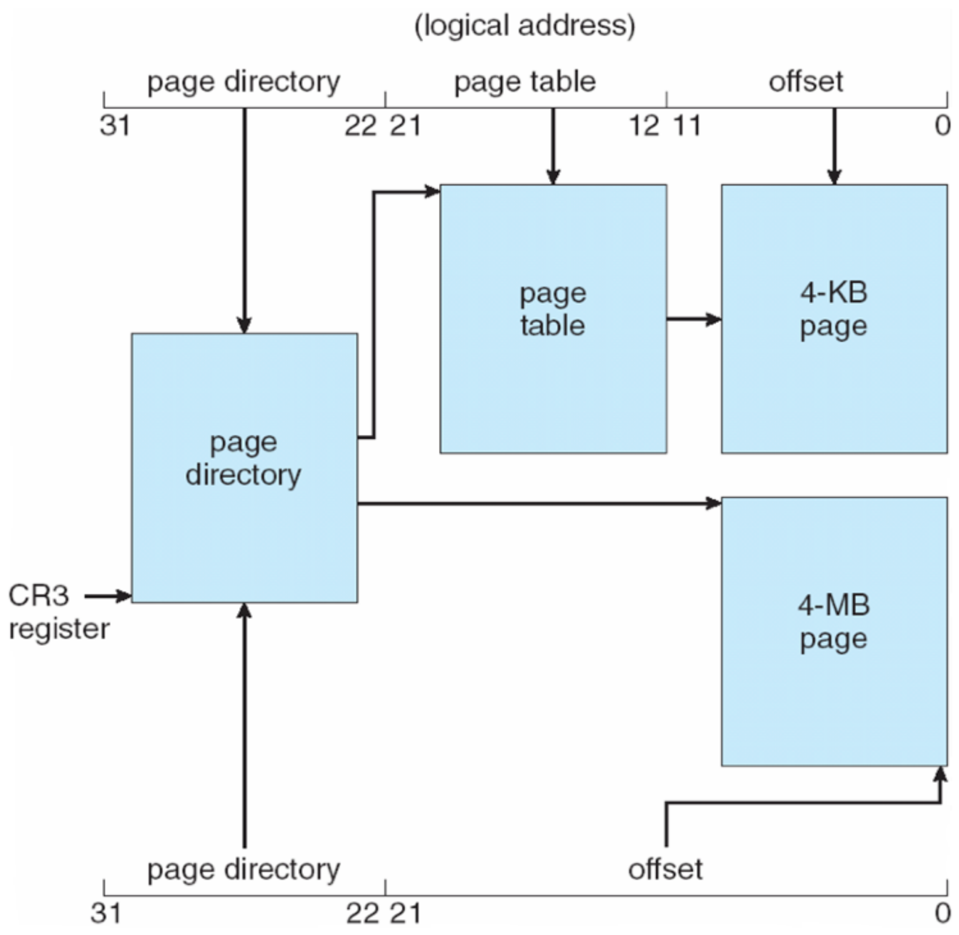




Intel IA-32 Segmentation

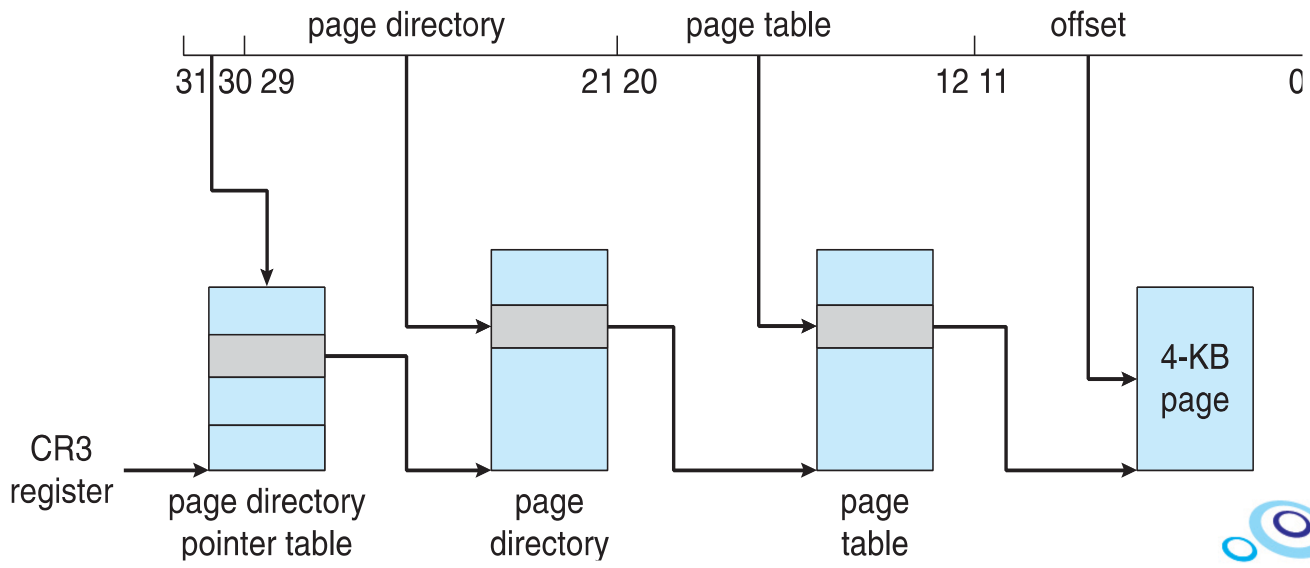


Intel IA-32 Paging Architecture



Intel IA-32 Page Address Extensions

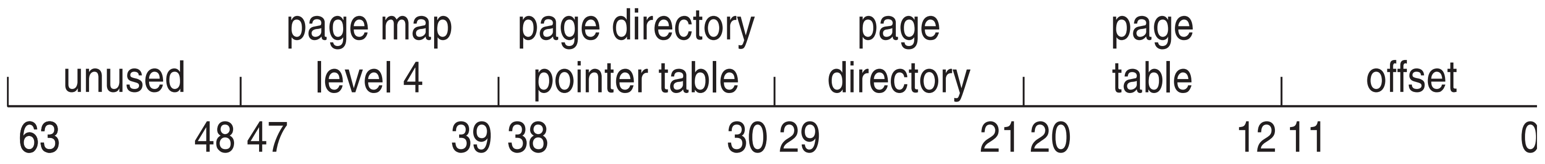
- 32-bit address limits led Intel to create page address extension (PAE), allowing 32-bit apps access to more than 4GB of memory space
  - Paging went to a 3-level scheme
  - Top two bits refer to a page directory pointer table
  - Page-directory and page-table entries moved to 64-bits in size
  - Net effect is increasing address space to 36 bits – 64GB of physical memory



NEWS Lab 嵌入式系統暨無線網路實驗室

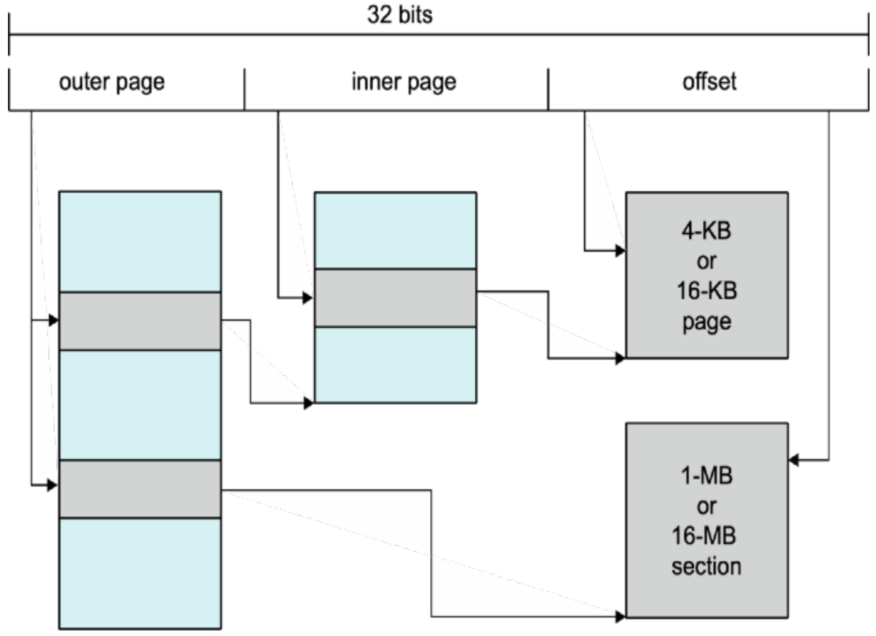
Intel x86-64

- Current generation Intel x86 architecture
- 64 bits is ginormous (> 16 exabytes)
- In practice only implement 48 bit addressing
  - Page sizes of 4 KB, 2 MB, 1 GB
  - Four levels of paging hierarchy
- Can also use PAE so virtual addresses are 48 bits and physical addresses are 52 bits



Example: ARM Architecture

- Dominant mobile platform chip (Apple iOS and Google Android devices for example)
- Modern, energy efficient, 32-bit CPU
- 4 KB and 16 KB pages
- 1 MB and 16 MB pages (termed sections)
- One-level paging for sections, two-level for smaller pages
- Two levels of TLBs
  - Outer level has two micro TLBs (one data, one instruction)
  - Inner is single main TLB
  - First inner is checked, on miss outers are checked, and on miss page table walk performed by CPU



Any Questions?

NEWS Lab
嵌入式系統暨無線網路實驗室

See You

Next Class

NEWS Lab
嵌入式系統暨無線網路實驗室

End of Chapter 9

NEWS Lab
嵌入式系統暨無線網路實驗室