Memory Management

Memory Addressing

- main()
```
{  static int a, b, c;
   a = 5;  b = 3;   c = a + b;
   printf ("addresses: %d, %d, %d\n", &a, &b, &c);
}
```
- Question: Run two instances of the same program, will the addresses of a be the same?

System Operations

- Compiler uses offset for addressing
  - Need to do this because program may reside in different memory locations even during execution
- Linker combines all object files into an executable - unifies the address offset
- Loader loads executable into memory, the base address is determined after loading

Memory Allocation

- Important elements in memory allocation
- Allocation Policy
  - Where in the memory to allocate a program
- Relocation
  - Allow the program being swapped out and swapped back in at a different memory location
- Addressing
  - Has to be offset, otherwise, the instructions has to be changed continuously
  - How to compute physical addresses from offsets
    - Computation needs to be very efficient

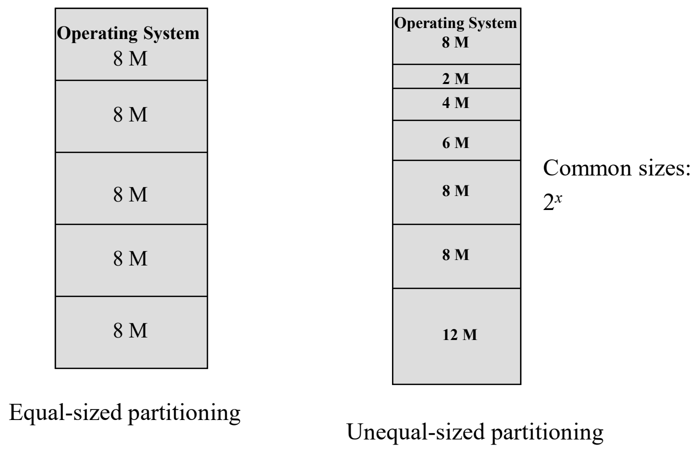
Memory Allocation

- Important elements in memory allocation
- Protection
  - A process should only access its designated memory locations
- Performance Metrics of memory allocation algorithms
  - Fragmentation
    - External fragmentation
    - Internal fragmentation
  - Allocation time
    - Time for executing the allocation algorithm

Allocation Policies

- Fixed Partitioning
- Dynamic Partitioning
- Simple Paging
- Simple Segmentation
- Virtual Memory and Demand Paging
- Virtual Memory and Demand Segmentation

Fixed Partitioning



Fixed Partitioning

- Partitions are set up at system initialization time
  - Programs are still allocated dynamically
  - A program can only be in one partition
- Advantages
  - Simple
- Disadvantages
  - Internal fragmentation
  - Cannot handle processes larger than the biggest partition but smaller than the entire memory

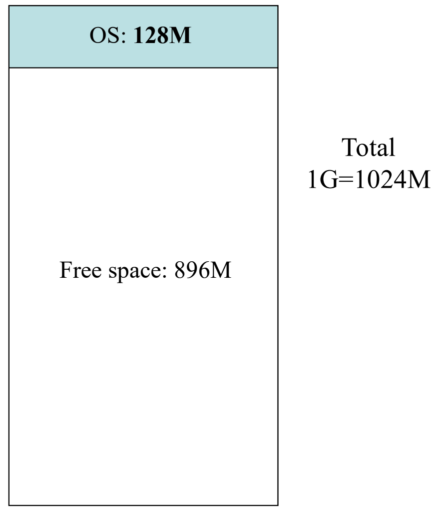
Fixed Partitioning

- Allocation policy
  - For equal-sized partitioning $\rightarrow$ just allocate any available partition
    - Very simple
    - Small process $\Rightarrow$ big fragmentation, Big process $\Rightarrow$ No slot can fit
  - For unequal-sized partitioning $\rightarrow$
    - A job goes to the best-fitted partition that is currently available
      - Possible that the best-fit partition is in use
      - Internal fragmentation
    - Each job wait for the best-fitted partition
      - Less fragmentation
      - If many jobs fit one particular partition, these jobs have to wait even though there are many other available partitions

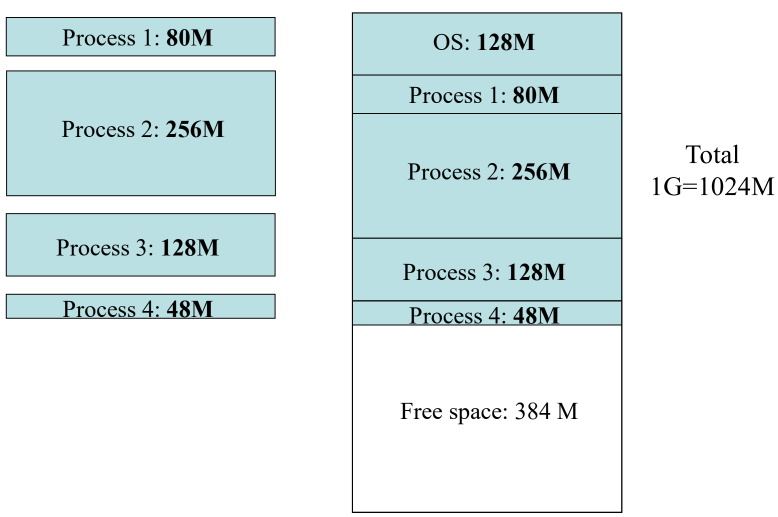
Dynamic Partitioning

- Fixed partitioning is not good
  - Internal fragmentation
  - Cannot handle large processes
- Dynamic partitioning
  - Memory is not partitioned in the beginning
  - One whole partition at initialization time
- Allocation policy
  - Allocation size = Process size
  - No internal fragmentation

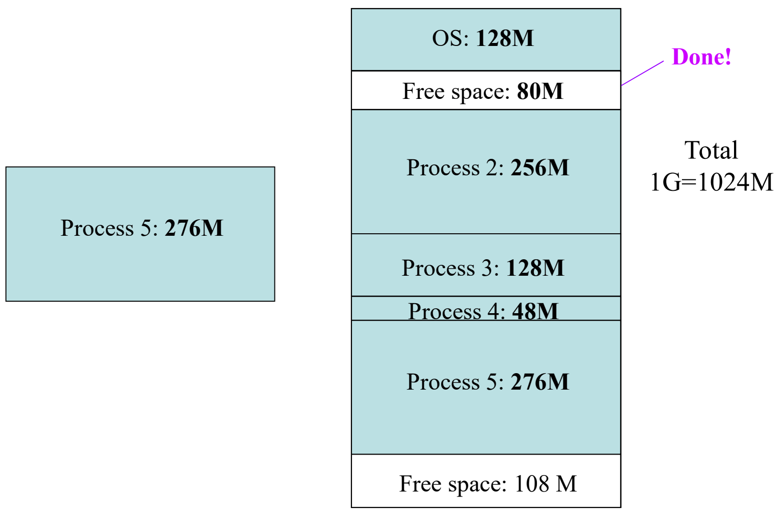
Dynamic Partitioning



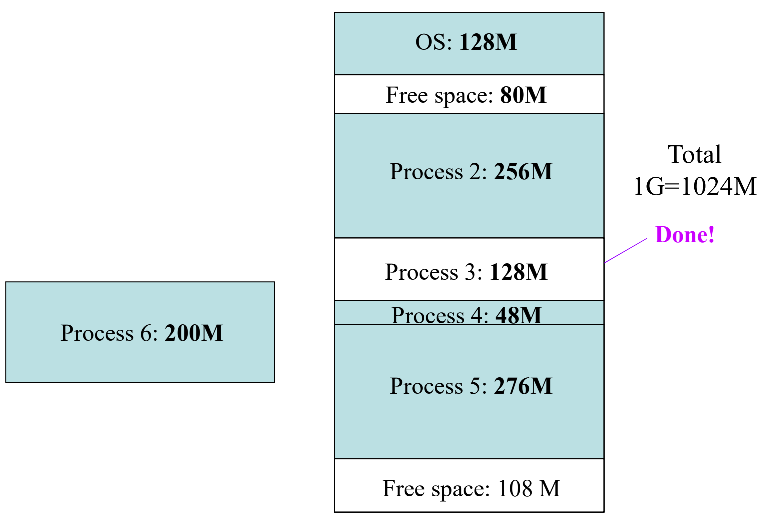
Dynamic Partitioning



Dynamic Partitioning

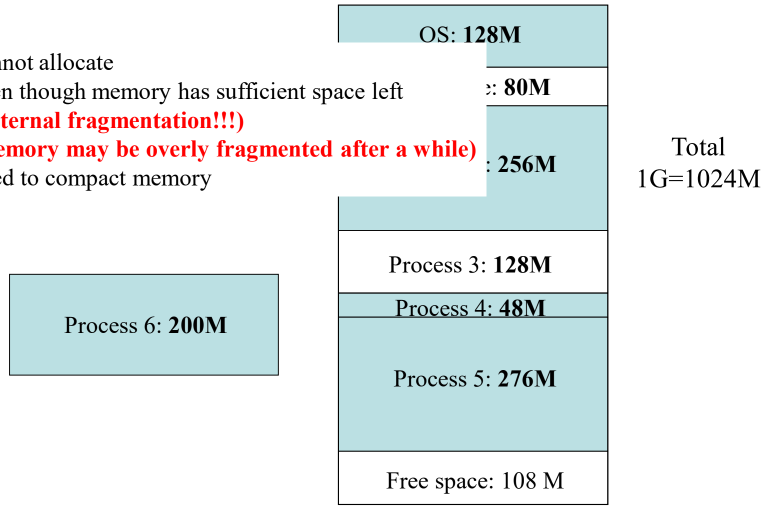


Dynamic Partitioning



Dynamic Partitioning

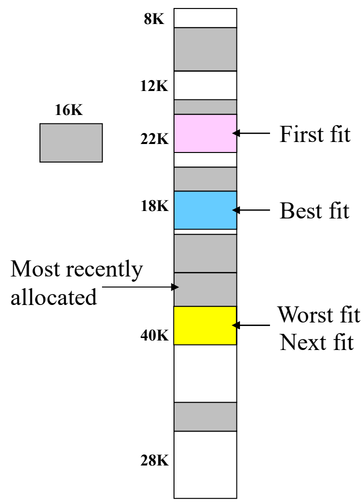
Cannot allocate
Even though memory has sufficient space left
(External fragmentation!!!)
(Memory may be overly fragmented after a while)
Need to compact memory



Dynamic Partitioning

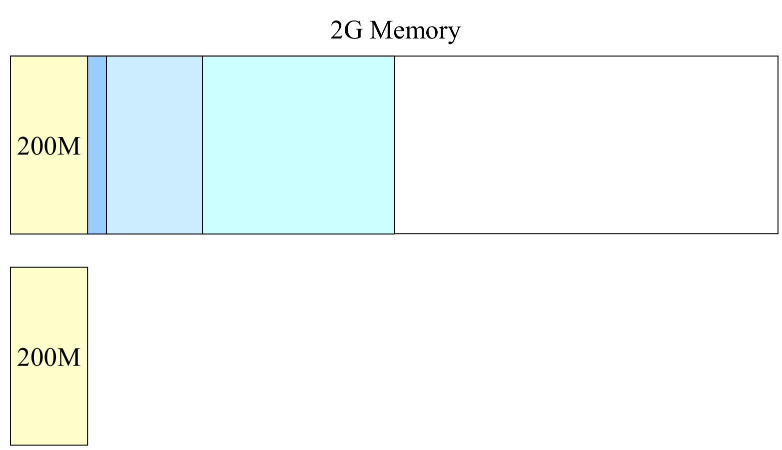
- Allocation Algorithms
  - Where should a new process be allocated?
  - First fit
    - Find first slot which is large enough for the new process
  - Best fit
    - Select the slot with size closest to the requested size
  - Worst fit
    - Select the slot with the largest size
  - Next fit
    - Start to search from the most recently allocated slot
    - Find the first fitting slot
  - Buddy Scheme
    - Split a partition into sizes of 2^x and allocate
    - Merge partitions (if possible) after a process leaves

Allocation Algorithms - DP



Allocation Algorithms - DP

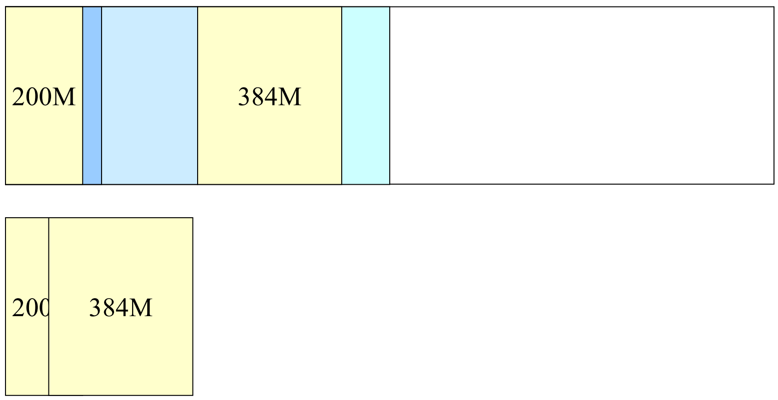
Buddy



Allocation Algorithms - DP

Buddy

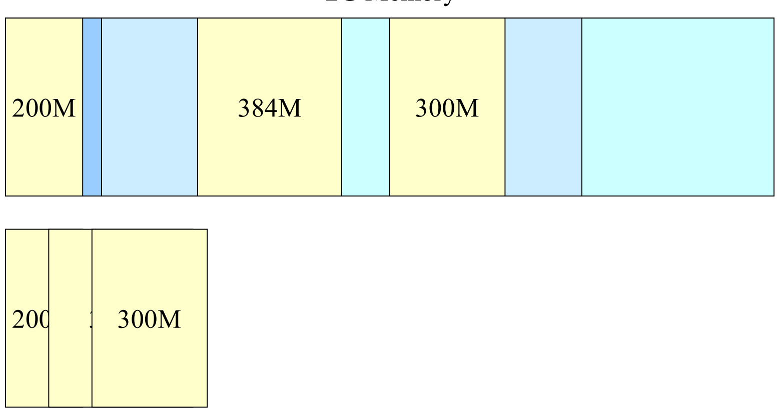
2G Memory



Allocation Algorithms - DP

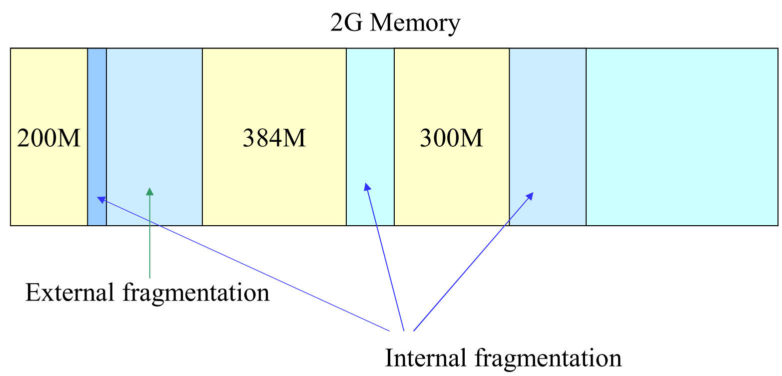
Buddy

2G Memory



Allocation Algorithms - DP

Buddy



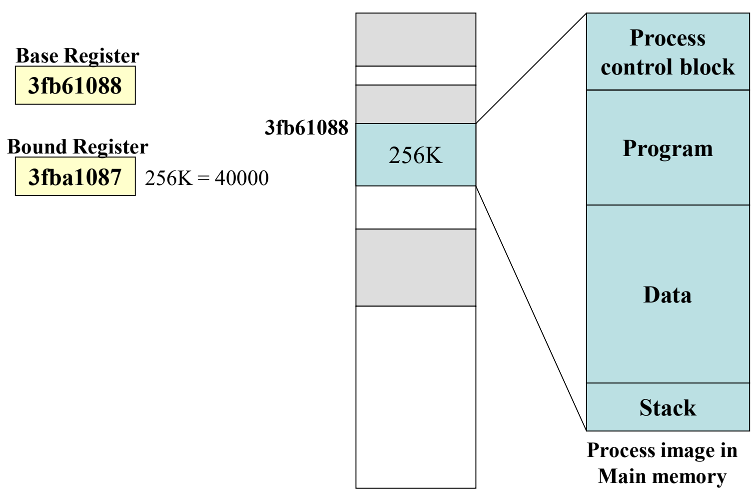
Allocation Algorithms - DP

- Buddy scheme
  - Has both internal and external fragmentation problems
  - Easy table look up (maintain a tree)
- First, best, worst, next fit
  - All has external fragmentation problem
  - Performance studies show they perform similarly
  - Need to keep a free list
    - Best fit and worst fit: list sorted in the order of free partition size
    - First fit and next fit: list sorted in the order of starting address
  - When a job leaves $\rightarrow$ return to free list $\rightarrow$ merge free slots
    - Memory manager checks the neighboring slots
    - Found a pair of free neighboring partitions $\Rightarrow$ Merge them
    - If free list is sorted by size $\Rightarrow$ Hard to locate neighbors

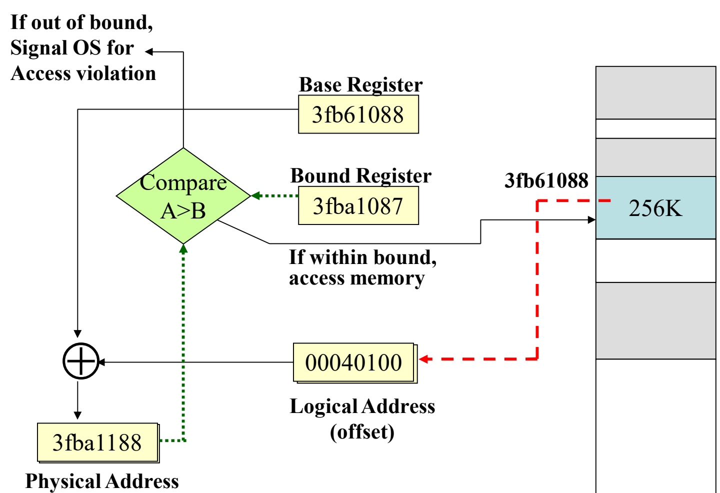
Addressing/Protection - DP

- Addressing
  - physical address = base address + relative address
  - (relative address, offset, logical address) are the same
  - (physical address, absolute address) are the same
  - Use a base address register to store the base address
- Protection
  - physical address limit = base address + program size
  - Use a bound register to protect against access violations

Addressing/Protection - DP



Addressing/Protection - DP



Simple Paging

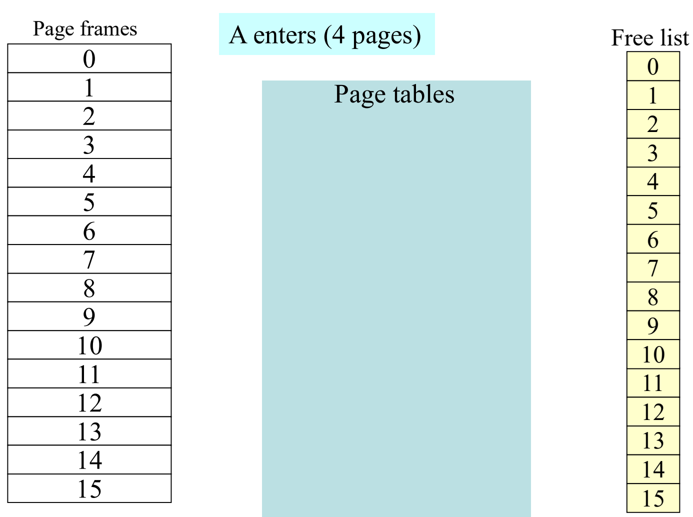
- Dynamic partitioning
  - Major fragmentation problem
    - Have sufficient space but do not have sufficient consecutive space
  - Better solution
    - Let a program to occupy non-consecutive regions
    - Use a table to keep track of the regions used for a program
    - But table may be too long
    - Use page as a unit → reduce number of table entries

⇒ Paging scheme
  - Memory is divided into fixed-size blocks
  - Physical memory block → **page frames**
  - Process address space block → **pages**

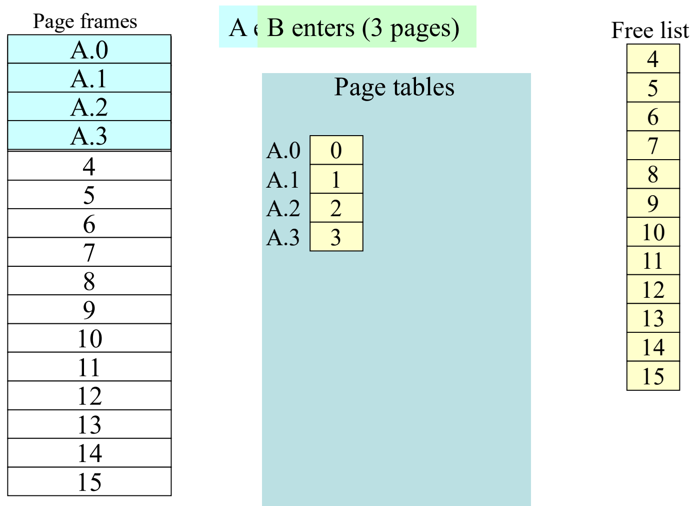
Simple Paging

- OS maintains a list of free frames
- Each process maintains a page table in PCB
- Process does not need to have contiguous frames
- Page size
  - Typically, 1KB to 8KB
  - Too big → internal fragmentation
  - Too small → large page table
- Performance
  - little internal fragmentation (in the last page)
  - Table space and table maintenance overhead

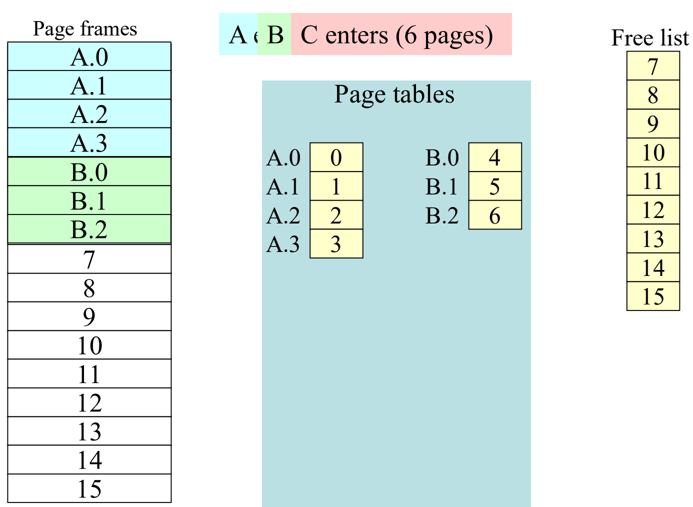
Simple Paging



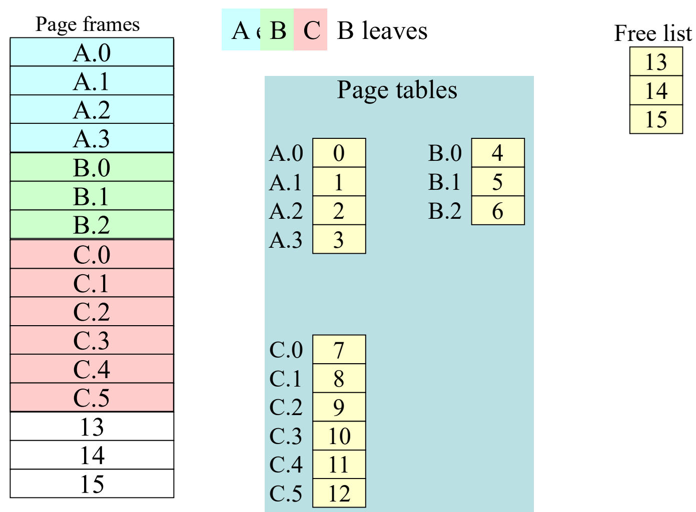
Simple Paging



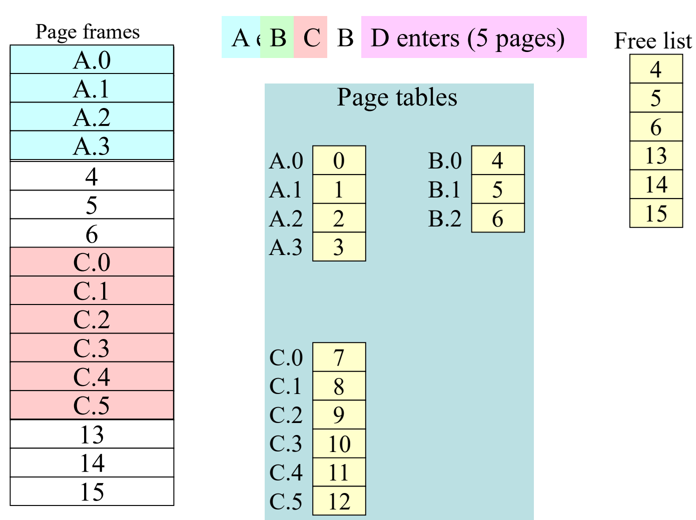
Simple Paging



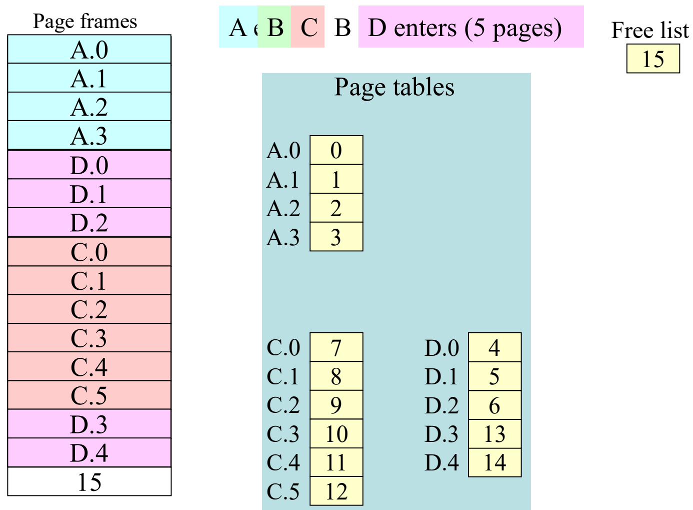
Simple Paging



Simple Paging



Simple Paging



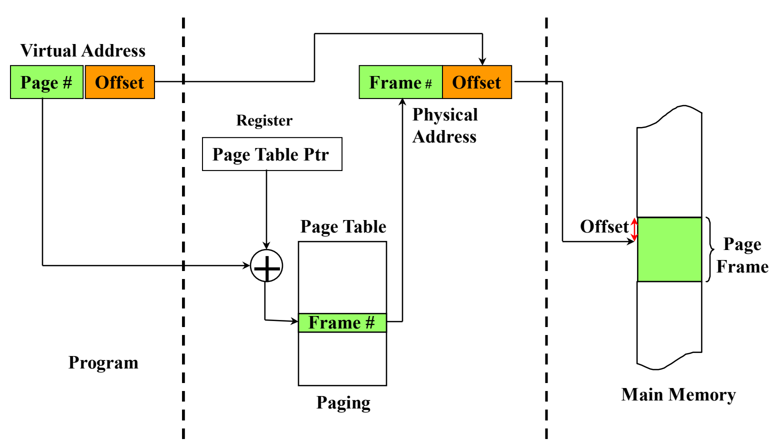
Addressing in Simple Paging (prelim.)

- Address computation?
  - Logical address x
  - Page #: $p = x$ div PS
    - PS: page size
  - Page offset: $t = x$ mod PS
  - Frame # in physical memory: f
    - Need a mapping table, maps p to f
  - Memory address = $f * PS + t$
    - (x div PS) * PS + (x mod PS)
  - If $PS = 2^n$?
    - n bits for offset and beyond n bits for page number

Addressing in Simple Paging (prelim.)

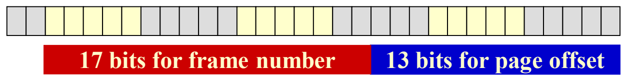
- How many bits are required to address the memory
  - 1KB memory = 2^{10} Bytes → 10 bits
  - 1GB memory = 2^{30} Bytes → 30 bits
  - 32 bits → can address 2^{32} Bytes = 4GB
- Page size and number of pages
  - 1GB memory, 8KB per page ⇒ 128 K pages
- Address (offset) within a page
  - 8KB per page ⇒ 13 bits to address the offset
- Address for pages
  - 128 K page ⇒ 17 bits

Addressing in Simple Paging (prelim.)



Addressing in Simple Paging (prelim.)

- Addressing
  - 1GB memory, 8KB per page ⇒ 128K pages

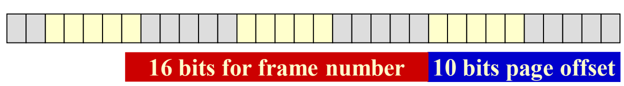


  - 00000000000000001010000000101011
    - Logical address in hexdecimal: 0x0001402b
    - Page number = 1010 = 10
    - Page offset = 101011 = 43
  - 00000000000000010101000000101011
    - Physical address in hexdecimal: 0x0002a02b
    - Frame number = 21 = 10101

P.0	7
P.1	40
P.2	1
	…
P.9	15
P.10	21

Addressing in Simple Paging (prelim.)

- Addressing
  - 64MB memory, 1KB per page ⇒ 64K pages

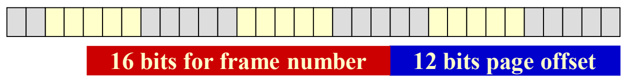


  - 00000000000000000000101011101011
    - Logical address in hexdecimal: 0x00000aeb
    - Page number = 10 = 2
    - Page offset = 1011101011
  - 00000000000000000000011011101011
    - Physical address in hexdecimal: 0x000006eb
    - Frame number = 1

P.0	7
P.1	40
P.2	1
	…
P.9	15
P.10	21

Addressing in Simple Paging (prelim.)

- Addressing
  - 256MB memory, 4KB per page ⇒ 64K pages



  - 00000000000000010010101011101011
    - Logical address in hexdecimal: 0x00012aeb
    - Page number = 10010 = 18
    - Exceed page table → access violation

P.0	7
P.1	40
P.2	1
	…
P.9	15
P.10	21

Readings

- Section 7.1-7.3