Department of Physics,
Computer Science & Engineering

CPSC 410 – Operating Systems I

Chapter 7: Memory Management

Memory Management

- Intro
- Requirements
  - Relocation, Protection, Sharing, Logical & Physical organization
- Partitioning
  - Fixed & Dynamic partitioning
- Paging
  - Frames & pages, Addressing
- Security issues

Intro

- Memory Management
  - one part of memory is used by the OS
  - the other is used by processes
    - Memory management deals with the use and control of this memory among processes.

- Terminology
  - Frame : a fixed-size block of main memory
  - Page : a fixed-size block of virtual memory
  - Segment: a variable-size block of…

…a process stored on disk

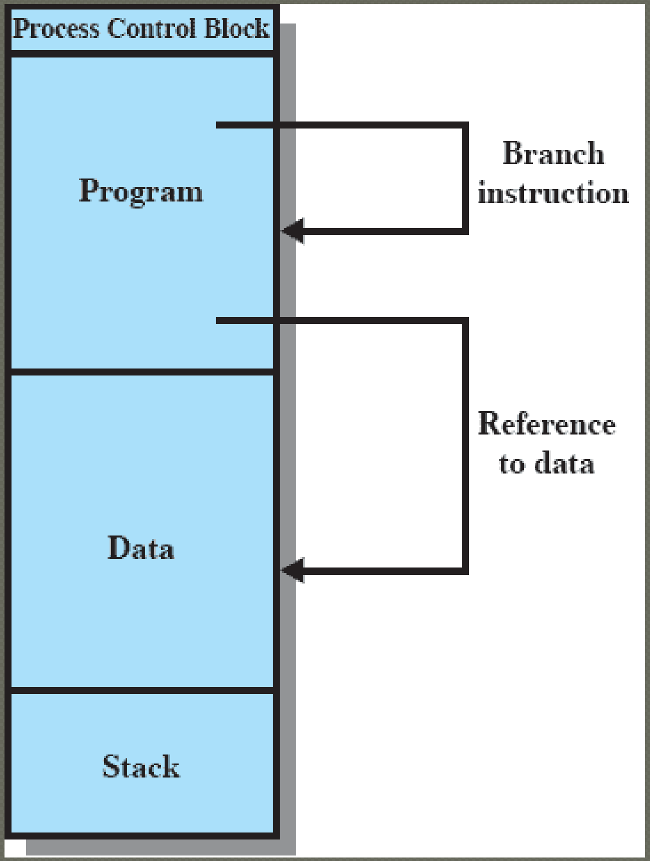
Memory Management

- Intro
- Requirements
  - Relocation, Protection, Sharing, Logical & Physical organization
- Partitioning
  - Fixed & Dynamic partitioning
- Paging
  - Frames & pages, Addressing
- Security issues

Requirements

…that Memory Management is meant to satisfy

- Relocation
  - processes are loaded to main memory to run.
  - eventually, they are swapped in and out of main memory to maximize CPU utilization.
  - Relocation implies that processes may get loaded into different memory spaces between swapping.
  - This has implications for addresses within processes.



Requirements

…that Memory Management is meant to satisfy

- Protection
  - Are processes referencing correct memory locations?
    - locations may change between swaps
  - …memory references must be checked at runtime
    - relocation must also support protection
- Sharing
  - Processes using the same modules could use one copy rather than having their own
    - protection must not be compromised when sharing memory
    - relocation must also support sharing

Requirements

…that Memory Management is meant to satisfy

- Logical organization
  - Memory (main & secondary) are linear
  - Programs are not! They use libraries (code abstraction)
    - written & compiled independently, can be shared
- Physical organization
  - flow of information between main & secondary memory
    - loading/unloading modules & data
  - Should programmers manage this flow?
    - What if a program + data does not fit into memory?
    - What if there are other programs running concurrently?
      - How much memory is available? Where/when will it become available?

This is getting too complicated!

Requirements

…that Memory Management is meant to satisfy

- Logical organization
  - Memory (main & secondary) are linear
  - Programs are not! They use libraries (code abstraction)
    - written & compiled independently, can be shared
- Physical organization
  - flow of information between main & secondary memory
    - loading/unloading modules & data
  - Should programmers manage this flow?
    - What if a program + data does not fit into memory?
    - What if there are other programs running concurrently?
      - How much memory is available? Where/when will it become

That’s why memory management is needed

This is getting too complicated!

Memory Management

- Intro
- Requirements
  - Relocation, Protection, Sharing, Logical & Physical organization

OK so…

memory management is all about

bringing processes into main memory for execution

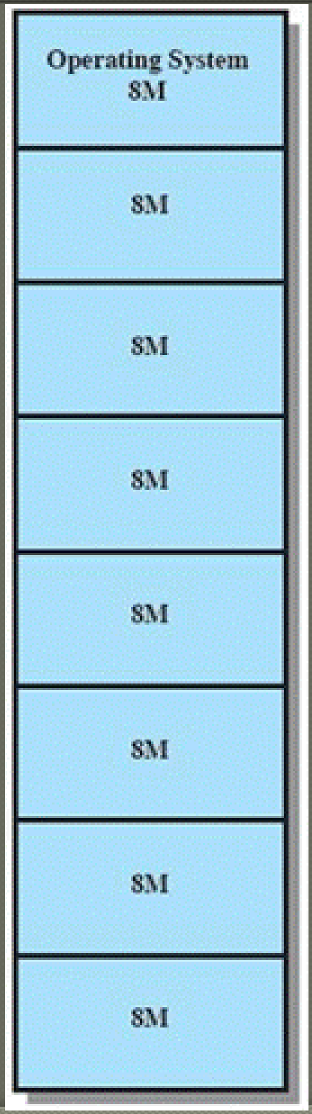
- involves partitioning, paging & segmentation
  - (although obsolete they help contrasting other concepts)
- involves virtual memory
  - (currently in use, see next chapter)

Memory Management

- Intro
- Requirements
  - Relocation, Protection, Sharing, Logical & Physical organization
- Partitioning
  - Fixed & Dynamic partitioning
- Paging
  - Frames & pages, Addressing
- Security issues

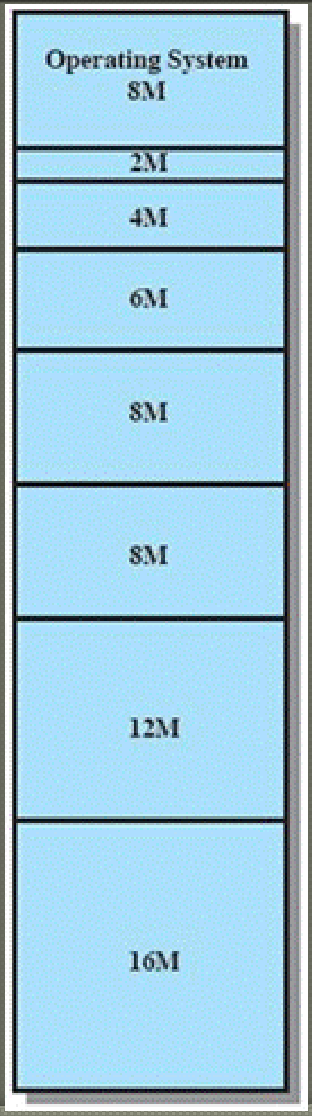
Partitioning

- Fixed Partitioning
  - Equal-size partitions
    - a process is loaded into a partition
    - OS swaps processes in & out as needed
  - Disadvantages
    - What if a process is larger than a partition?
      - code must be designed with overlays
    - What if a process is smaller than a partition?
      - leftover memory is not used
      - aka internal fragmentation
        - wasted space due to the process loaded being smaller than the partition



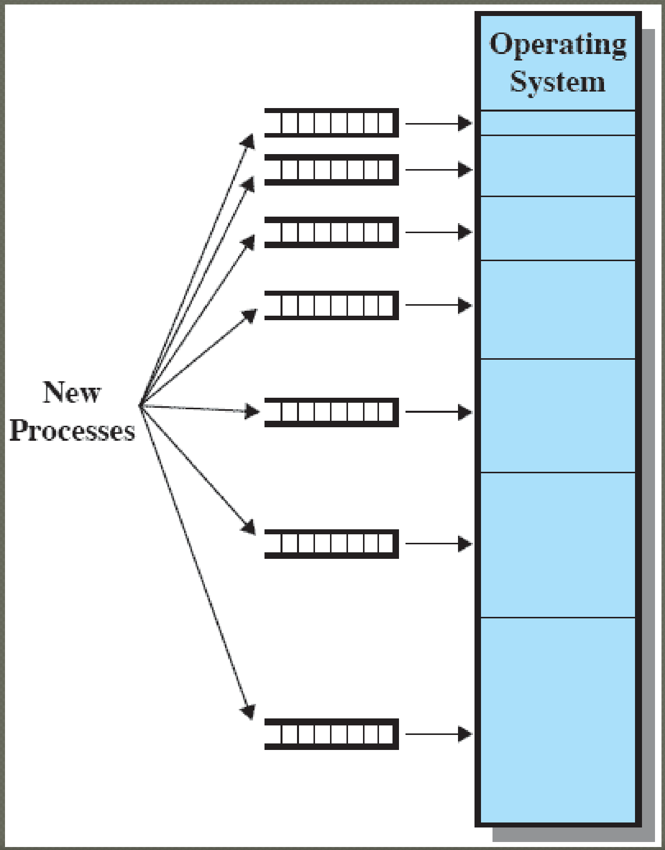
Partitioning

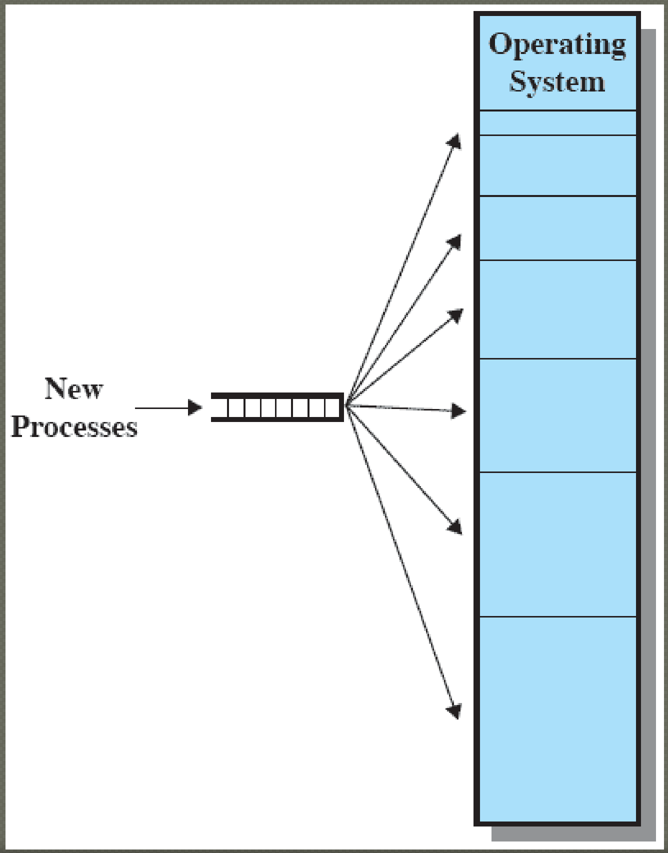
- Fixed Partitioning (II)
  - Unequal-size partitions
    - Larger processes can be accommodated without the need of overlays
    - There is less internal fragmentation by using best fit partition
  - Disadvantages (ditto for Fixed)
    - Number of partitions (set at startup) limits the number of active processes
    - Prone to memory waste in cases when there are many small jobs



Partitioning

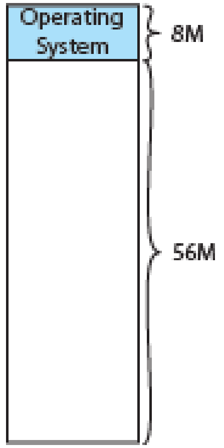
- Fixed Partitioning (II)
  - Unequal-size partitions (placement algorithm)





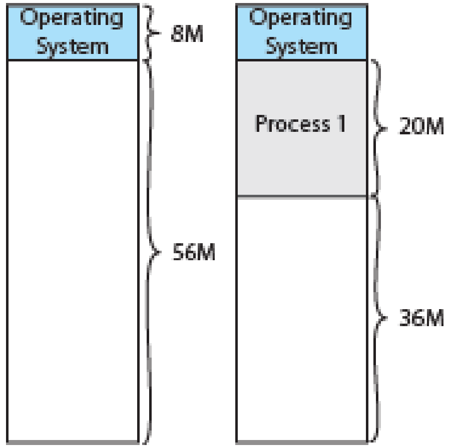
Partitioning

- Dynamic Partitioning
  - Partitions vary in length & number
  - Processes are given the exact memory they require
    - Example (RAM 64M)
      - P1 starts (20M), P2 starts (14M), P3 starts (18M)
      - P2 ends, P4 starts (8M), P1 ends, P2 restarts (14M)



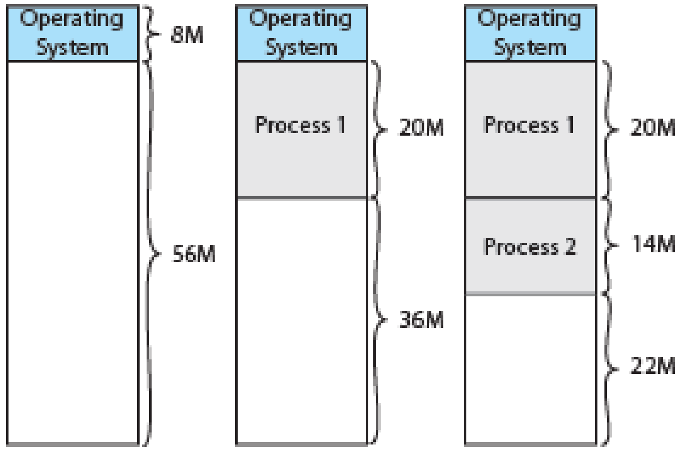
Partitioning

- Dynamic Partitioning
  - Partitions vary in length & number
  - Processes are given the exact memory they require
    - Example (RAM 64M)
      - P1 starts (20M), P2 starts (14M), P3 starts (18M)
      - P2 ends, P4 starts (8M), P1 ends, P2 restarts (14M)



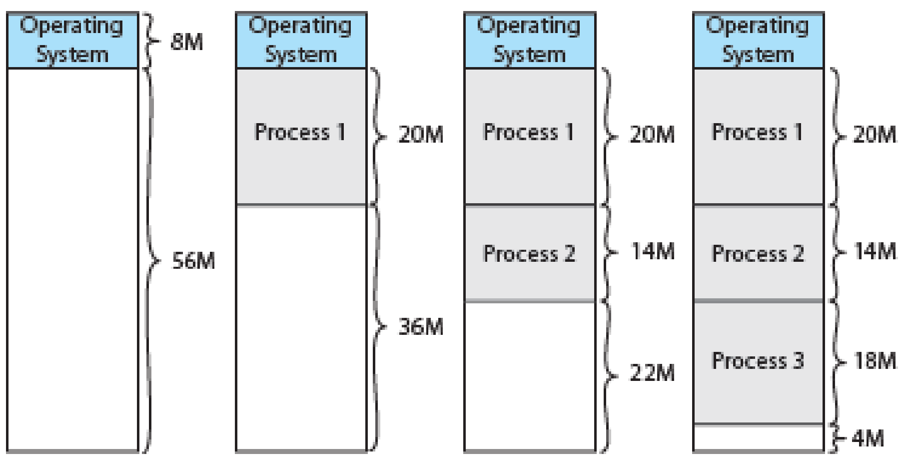
Partitioning

- Dynamic Partitioning
  - Partitions vary in length & number
  - Processes are given the exact memory they require
    - Example (RAM 64M)
      - P1 starts (20M), P2 starts (14M), P3 starts (18M)
      - P2 ends, P4 starts (8M), P1 ends, P2 restarts (14M)



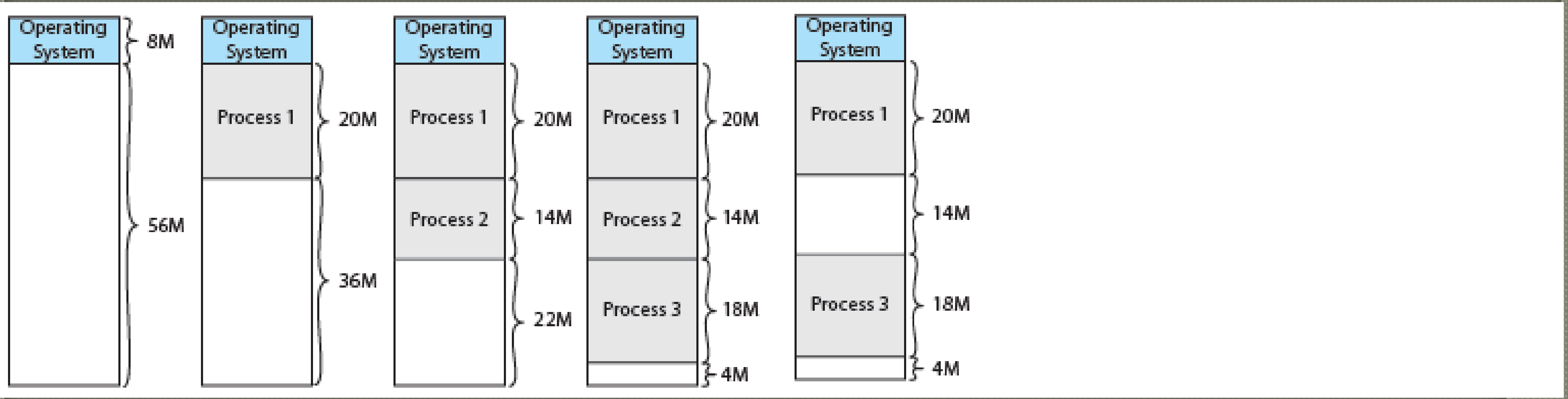
Partitioning

- Dynamic Partitioning
  - Partitions vary in length & number
  - Processes are given the exact memory they require
    - Example (RAM 64M)
      - P1 starts (20M), P2 starts (14M), P3 starts (18M)
      - P2 ends, P4 starts (8M), P1 ends, P2 restarts (14M)



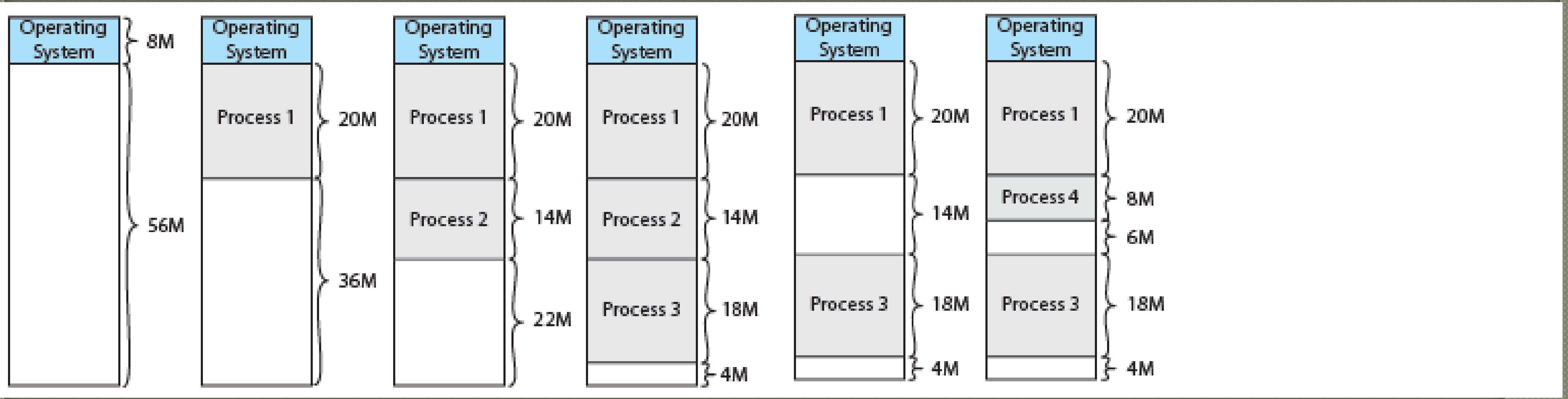
Partitioning

- Dynamic Partitioning
  - Partitions vary in length & number
  - Processes are given the exact memory they require
    - Example (RAM 64M)
      - P1 starts (20M), P2 starts (14M), P3 starts (18M)
      - P2 ends, P4 starts (8M), P1 ends, P2 restarts (14M)



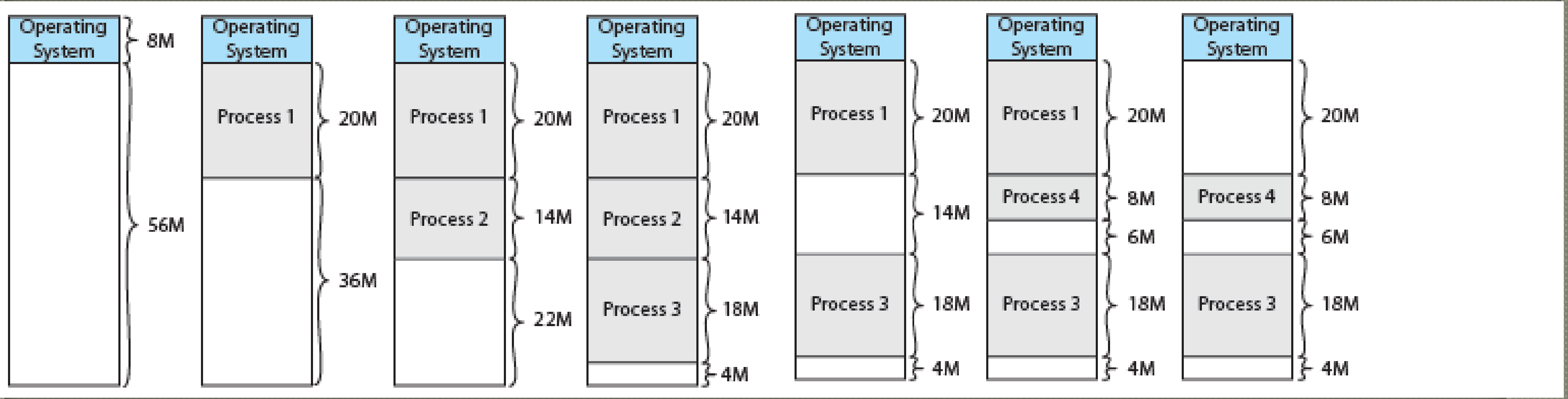
Partitioning

- Dynamic Partitioning
  - Partitions vary in length & number
  - Processes are given the exact memory they require
    - Example (RAM 64M)
      - P1 starts (20M), P2 starts (14M), P3 starts (18M)
      - P2 ends, P4 starts (8M), P1 ends, P2 restarts (14M)



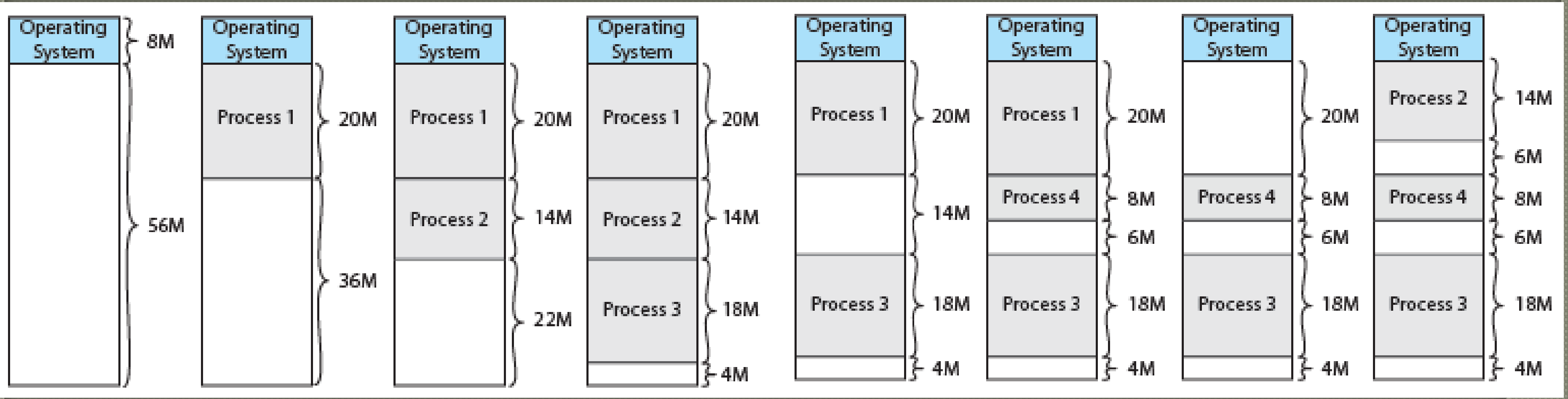
Partitioning

- Dynamic Partitioning
  - Partitions vary in length & number
  - Processes are given the exact memory they require
    - Example (RAM 64M)
      - P1 starts (20M), P2 starts (14M), P3 starts (18M)
      - P2 ends, P4 starts (8M), P1 ends, P2 restarts (14M)



Partitioning

- Dynamic Partitioning
  - Partitions vary in length & number
  - Processes are given the exact memory they require
    - Example (RAM 64M)
      - P1 starts (20M), P2 starts (14M), P3 starts (18M)
      - P2 ends, P4 starts (8M), P1 ends, P2 restarts (14M)

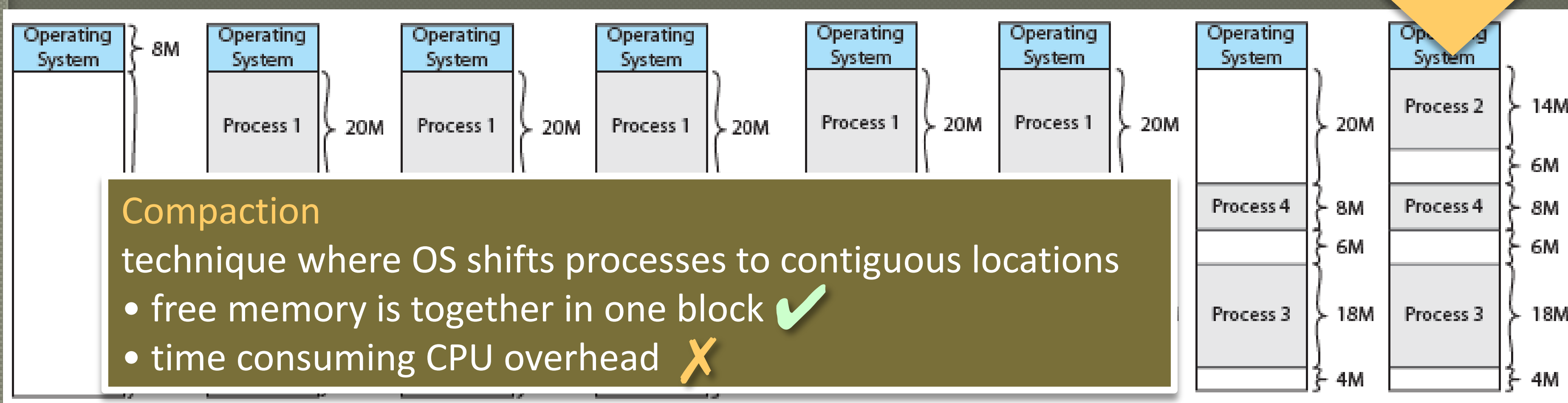


Partitioning

- Dynamic Partitioning
  - Partitions vary in length & number
  - Processes are given the exact memory they require
    - Example (RAM 64M)
      - P1 starts (20M), P2 sta
      - P2 ends, P4 starts (8M

External Fragmentation

- memory becomes more and more fragmented
- memory utilization declines



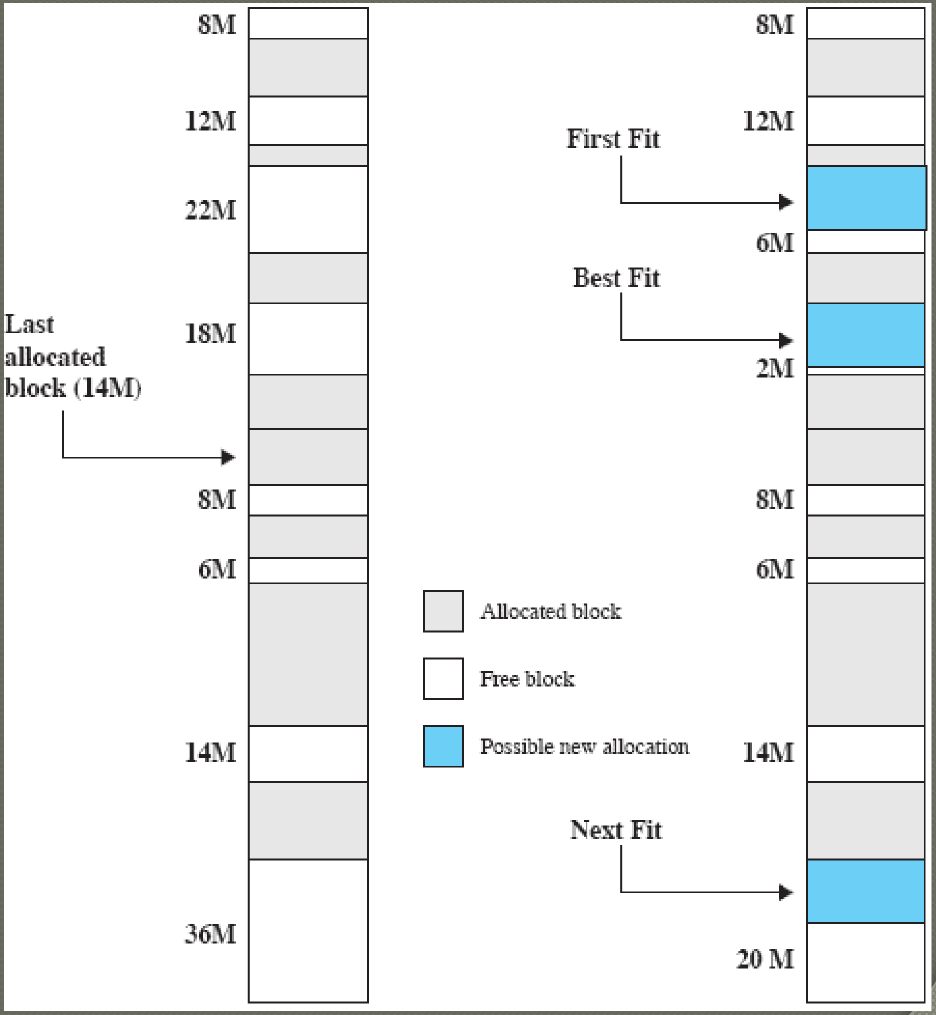
Compaction

technique where OS shifts processes to contiguous locations

- free memory is together in one block ✔
- time consuming CPU overhead ✗

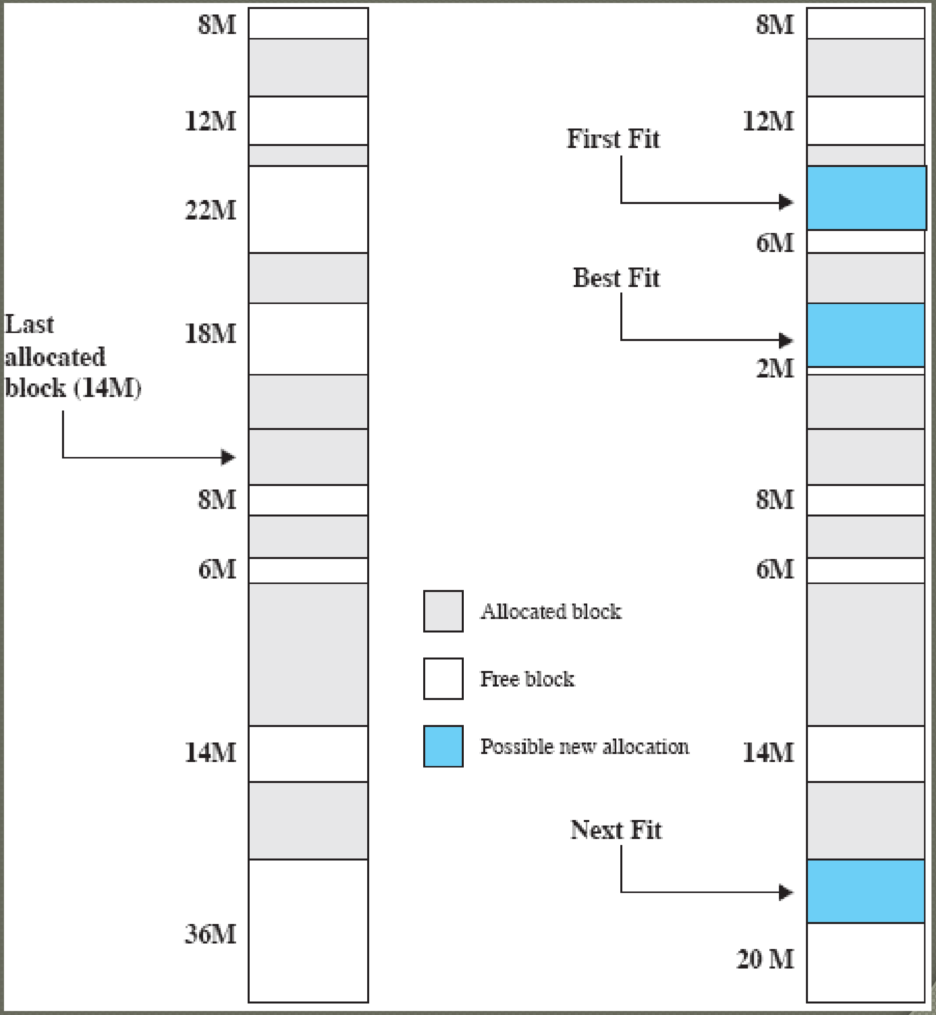
Partitioning

- Dynamic Partitioning
  - Placement algorithms
  - Best-fit
    - chooses block closest in size to fit the request.
  - First-fit
    - scanning from top.
    - chooses first block large enough to fit request
  - Next-fit
    - scanning from place of last allocation.
    - chooses next block large enough to fit request



Partitioning

- Dynamic Partitioning
  - Entire process has to be loaded in contiguous memory block
  - What if you have enough memory but its fragmented?

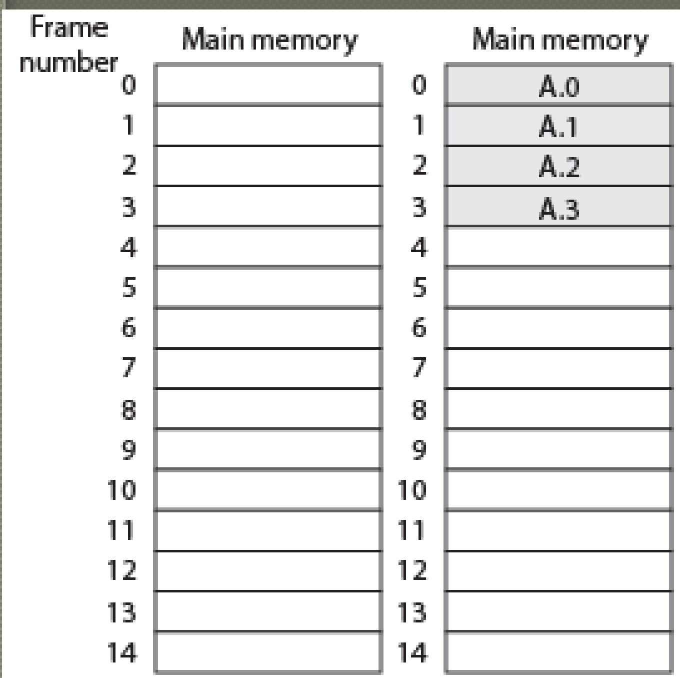


Memory Management

- Intro
- Requirements
  - Relocation, Protection, Sharing, Logical & Physical organization
- Partitioning
  - Fixed & Dynamic partitioning
- Paging
  - Frames & pages, Addressing
- Security issues

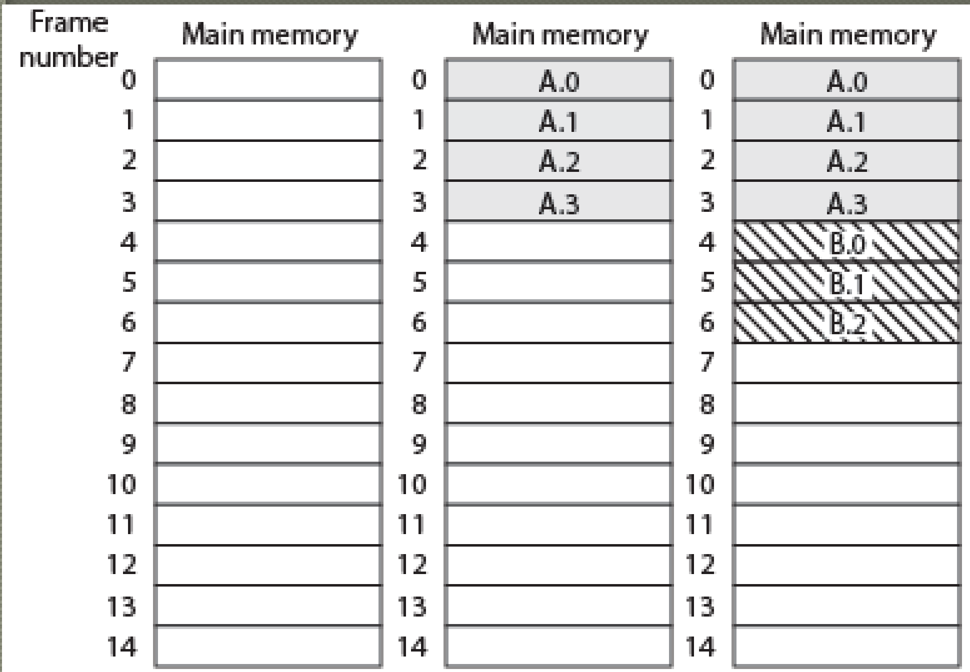
Paging

- Partition memory into frames…
- Partition processes into pages…
  - …which are equal fixed-size chunks relatively small
    - e.g., A(4) runs, B(3) runs, C(4) runs, B ends, D(5) runs



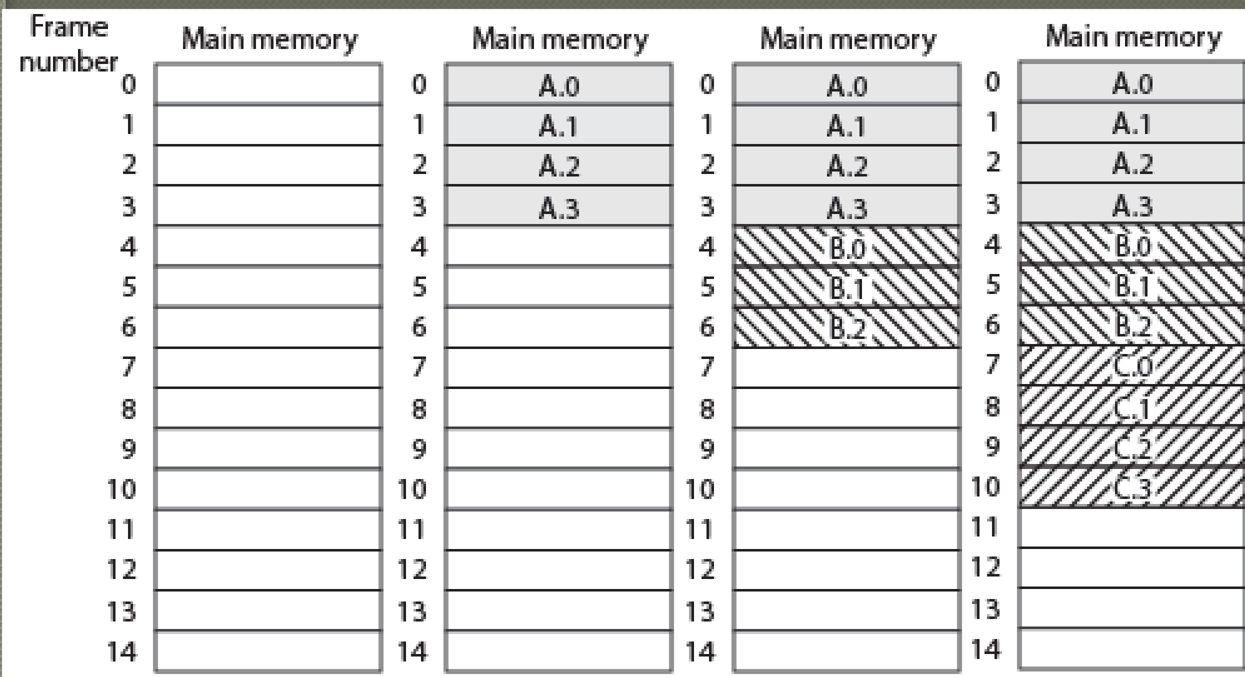
Paging

- Partition memory into frames…
- Partition processes into pages…
  - …which are equal fixed-size chunks relatively small
    - e.g., A(4) runs, B(3) runs, C(4) runs, B ends, D(5) runs



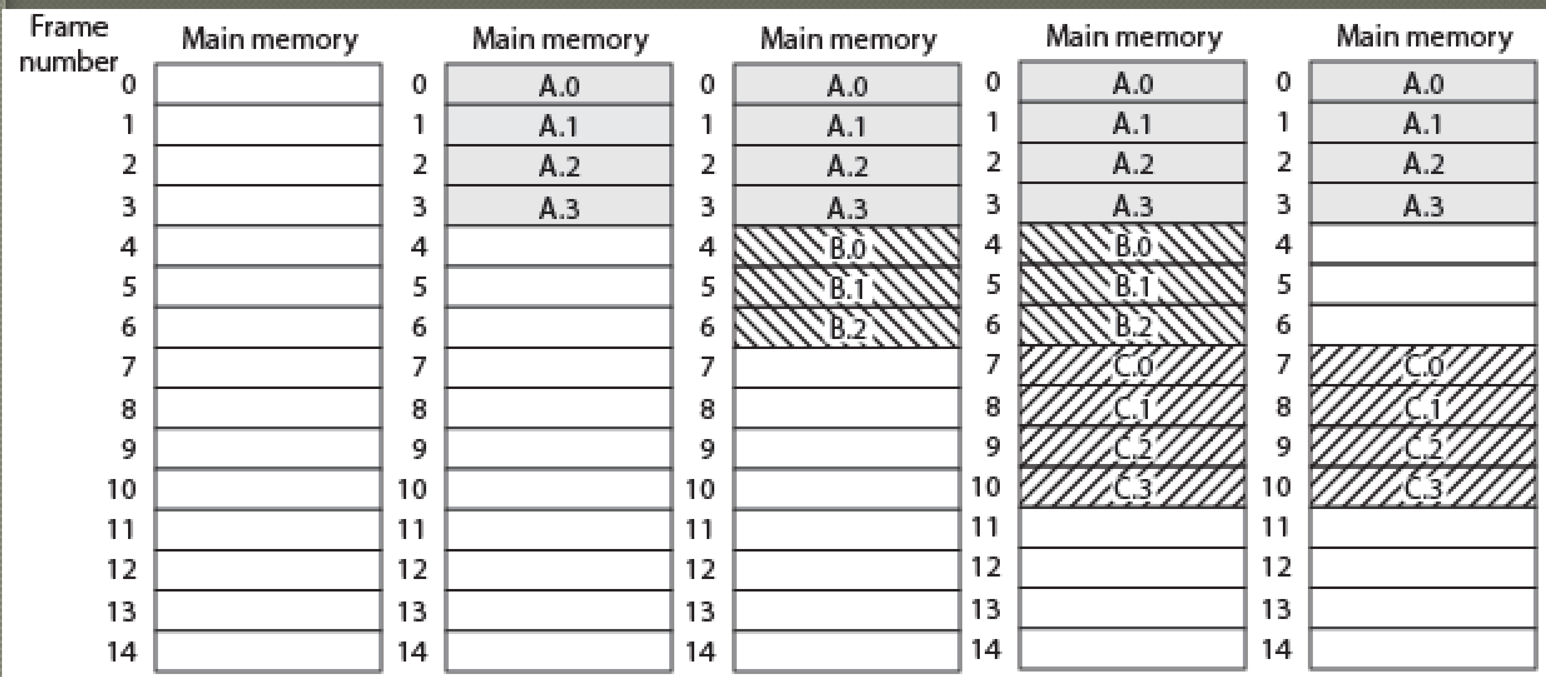
Paging

- Partition memory into frames…
- Partition processes into pages…
  - …which are equal fixed-size chunks relatively small
    - e.g., A(4) runs, B(3) runs, C(4) runs, B ends, D(5) runs



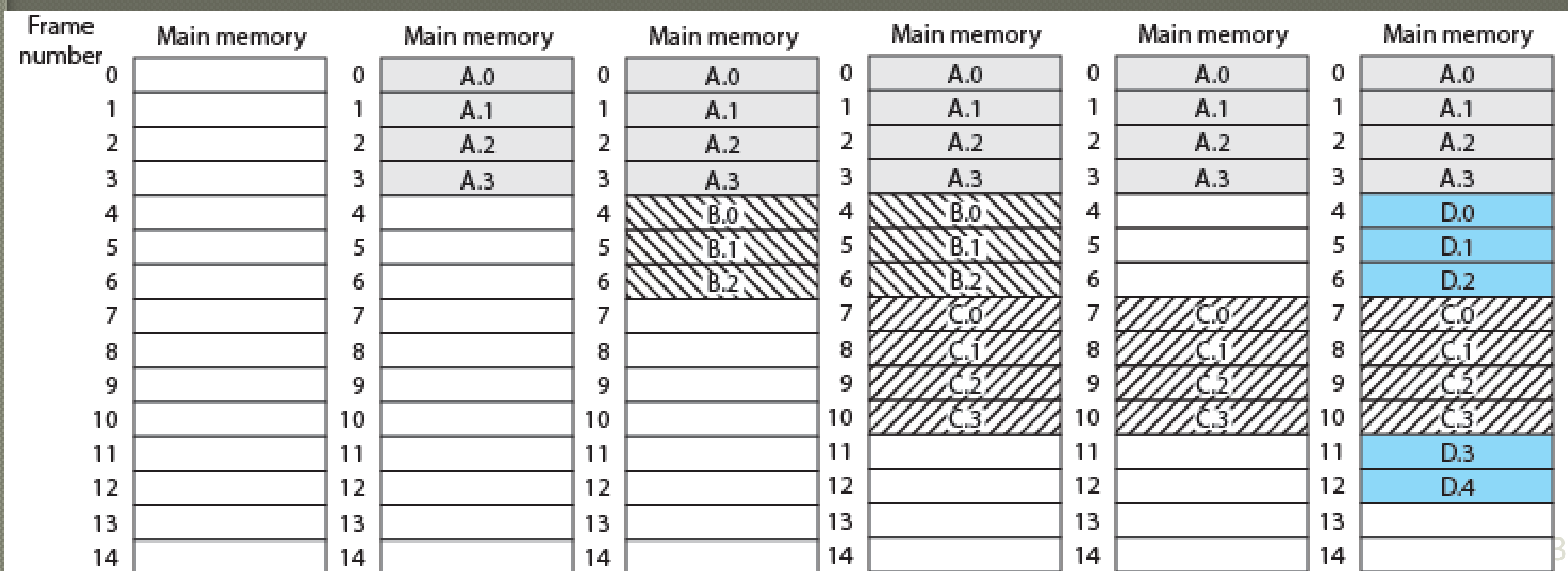
Paging

- Partition memory into frames…
- Partition processes into pages…
  - …which are equal fixed-size chunks relatively small
    - e.g., A(4) runs, B(3) runs, C(4) runs, B ends, D(5) runs



Paging

- Partition memory into frames…
- Partition processes into pages…
  - …which are equal fixed-size chunks relatively small
    - e.g., A(4) runs, B(3) runs, C(4) runs, B ends, D(5) runs



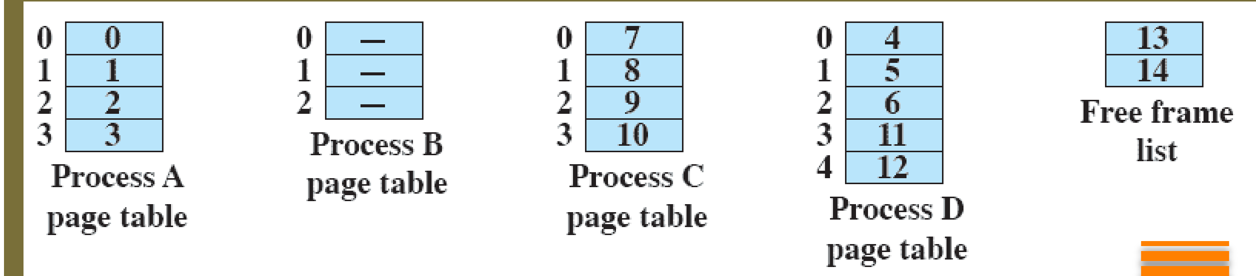
Paging

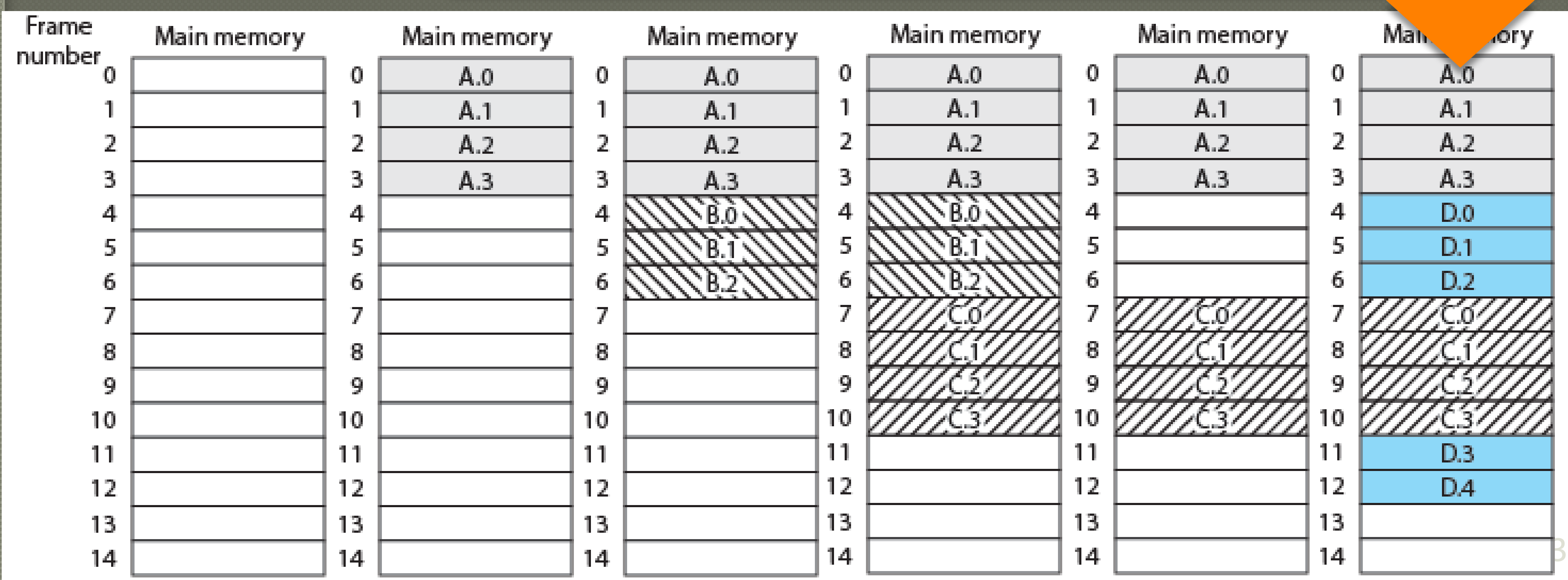
- Partition m
- Partition pr
  - …which are
    - e.g., A(4) r

Page Table

- Table where OS keeps frame location of each process page
- Used by processor to produce a physical address

How?

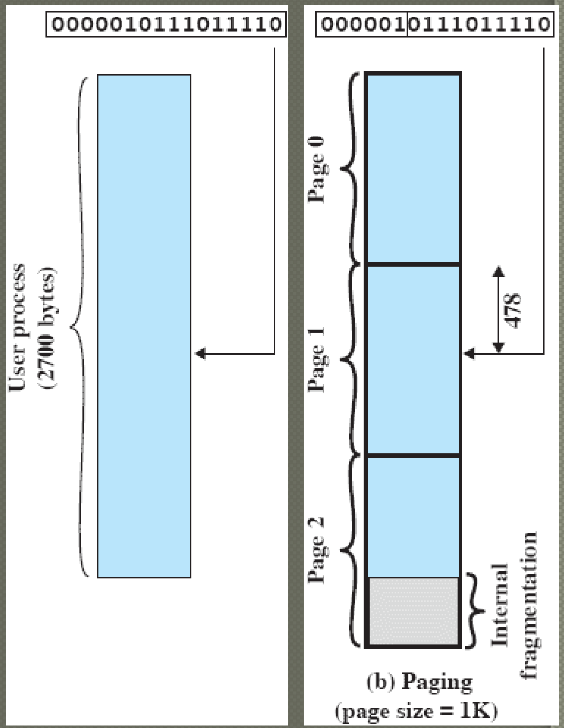




Board work

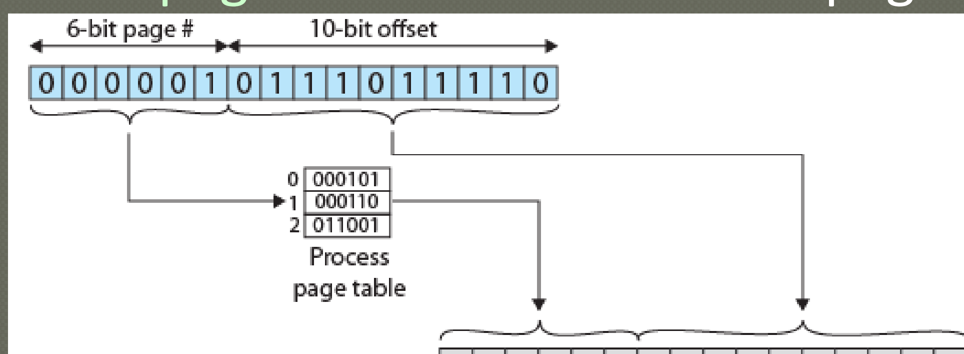
Paging – (fixed page size)

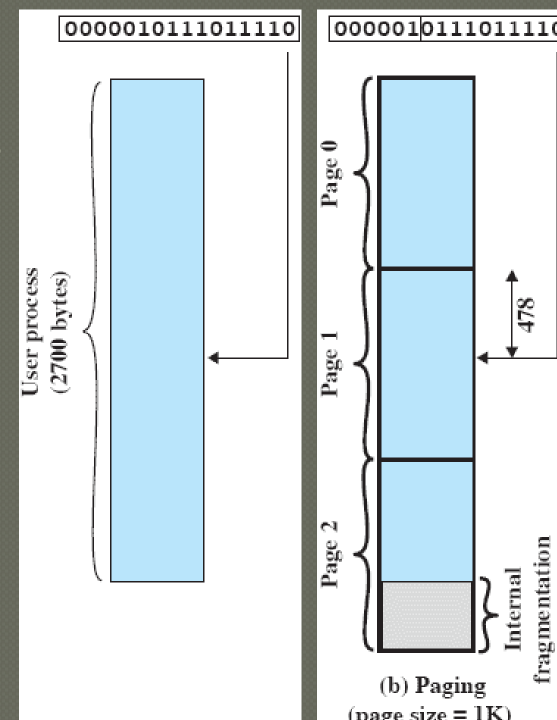
- Addressing: break up into frames
  - Divide address into page bits and address bits
  - Length of each page? 2** address bits
  - Number pages? 2** page bits
    - Ex. if 16-bit addressing && 1K page size
      10 bits page length => 2**10 =1024
      6 bits page number => 2**6 =64 pages



Paging – (fixed page size)

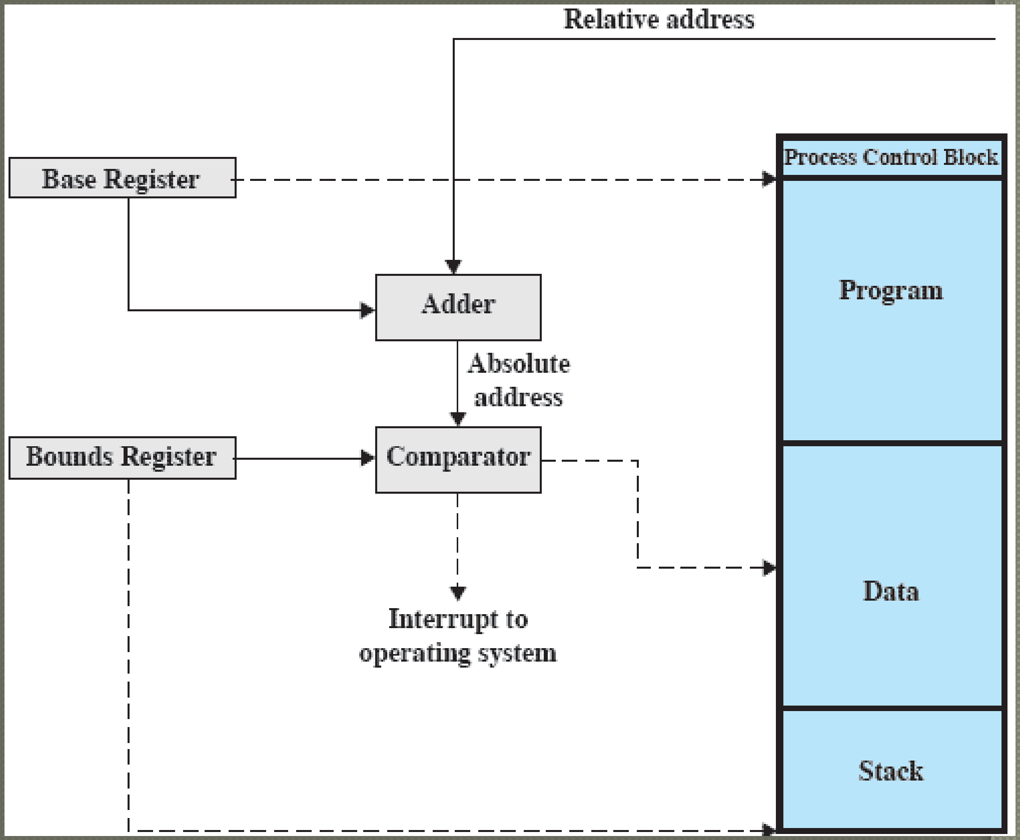
- Addressing: from relative to physical
  - Divide address into page bits and address bits
  - Length of each page? 2** address bits
  - Number pages? 2** page bits
    - Ex. if 16-bit addressing && 1K page size
      10 bits page length => 2**10 =1024
      6 bits page number => 2**6 =64 pages





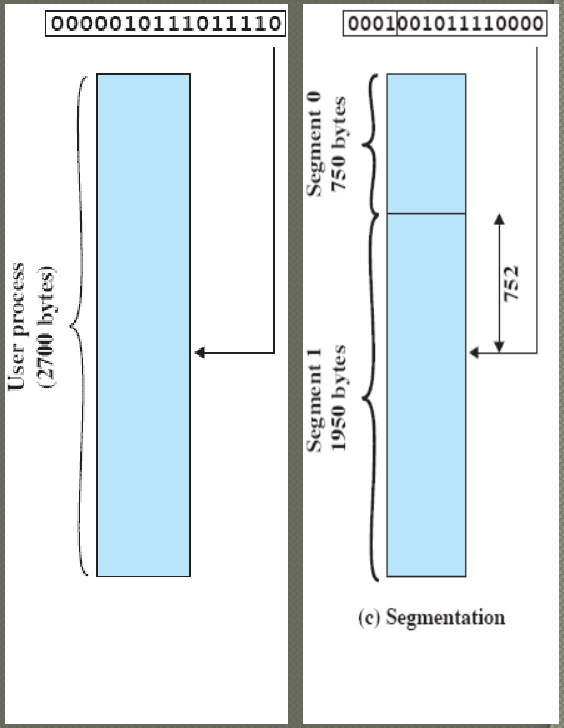
Partitioning

- Addresses
  - Logical
    - reference to a memory location independent of the current assignment of data to memory
  - Relative
    - (special case of logical) reference to a memory location relative to a known point
  - Physical (aka absolute)
    - reference to the actual location in main memory

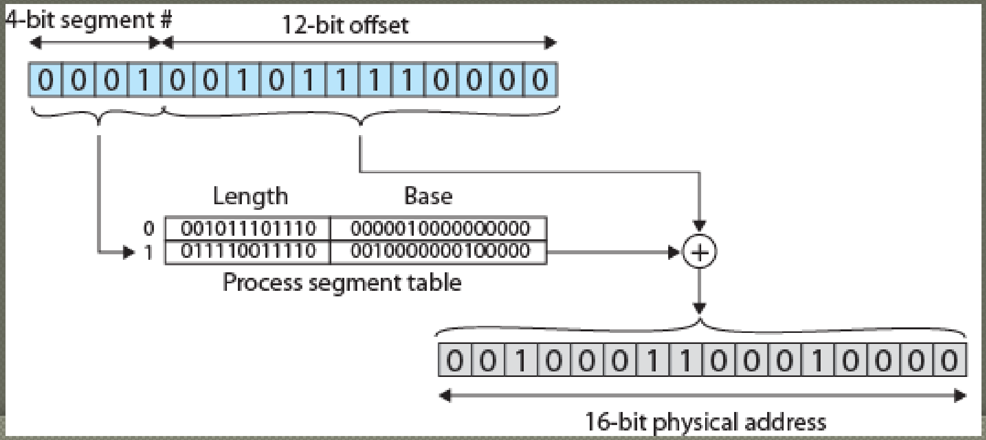


Partitioning

- Partition processes into segments…
  - chunks of varying but limited length
- Same addressing layout applies:
  - segment number + offset
    - e.g., if 16-bit addressing && 12 bits for reference anywhere then there can be 32 segments of up to 4K max in length



(c) Segmentation



Memory Management

- Intro
- Requirements
  - Relocation, Protection, Sharing, Logical & Physical organization
- Partitioning
  - Fixed & Dynamic partitioning
- Paging
  - Frames & pages, Addressing
- Security issues

Security Issues

- Buffer Overflow (aka buffer overrun)
  - security threat related to memory management
  - a process stores data beyond limits of a buffer
  - oldest & more prevalent type of security attacks

```
int main() {
	int	valid = 0;
	char expected[ 8 ];
	char actual[ 8 ];
	strcpy( expected, "START" ); // get expected password
	gets( actual );
	if (strncmp( expected, actual, 8) == 0) {
		valid = 1;
	}
	printf("expected(%s) actual(%s) valid(%d)\n", expected, actual, valid);
	return 0;
}
```

```
bash-3.2$ c++ getPassword.cpp
/tmp/ccSkV7wJ.o: In function `main':
getPassword.cpp:(.text+0x24): warning: the `gets' function is dangerous and should not be used.
bash-3.2$ ./a.out
START
expected(START) actual(START) valid(1)
bash-3.2$ ./a.out
EVILINPUTevilinputEvIlInPuT
expected(utEvIlInPuT) actual(EVILINPUTevilinputEvIlInPuT) valid(0)
bash-3.2$ ./a.out
BADINPUTbadinputBADINPUT
expected(BADINPUT) actual(BADINPUTbadinputBADINPUT) valid(1)
bash-3.2$
```

```
int main() {
	int	valid = 0;
	char expected[ 8 ];
	char actual[ 8 ];
	strcpy( expected, "START" ); // get expected password
	gets( actual );
	if (strncmp( expected, actual, 8) == 0) {
		valid = 1;
	}
	printf("expected(%s) actual(%s) valid(%d)\n", expected, actual, valid);
	return 0;
}
```

Memory Management

- Intro
- Requirements
  - Relocation, Protection, Sha gical & Physical organization
- Partitioning
  - Fixed & Dynam ioning, Buddy system
- Paging
  - Frames & pages, Addressing
- Security issues

Done!