Operating Systems: Internals and Design Principles

Memory Management

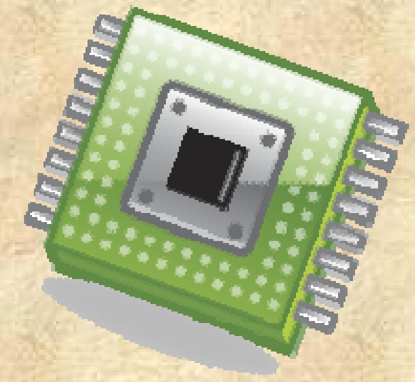
Requirements
Fixed and dynamic partitioning
Partition placement in memory
Paging
Segmentation
Security: buffer overflow, attack and defense
(Note: Virtual memory in Ch 8)

Operating Systems: Internals and Design Principles

I cannot guarantee that I carry all the facts in my mind. Intense mental concentration has a curious way of blotting out what has passed. Each of my cases displaces the last, and Mlle. Carère has blurred my recollection of Baskerville Hall. Tomorrow some other little problem may be submitted to my notice which will in turn dispossess the fair French lady and the infamous Upwood.

— *THE HOUND OF THE BASKERVILLES,*
Arthur Conan Doyle

Memory Management Terms



kehys, sivukehys

Frame	A fixed-length block of main memory.
Page	A fixed-length block of data that resides in secondary memory (such as disk). A page of data may temporarily be copied into a frame of main memory.
Segment	A variable-length block of data that resides in secondary memory. An entire segment may temporarily be copied into an available region of main memory (segmentation) or the segment may be divided into pages which can be individually copied into main memory (combined segmentation and paging).

sivu

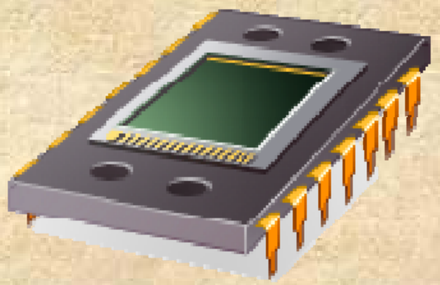
segmentti

Table 7.1

Memory Management Requirements

- Memory management is intended to satisfy the following requirements:

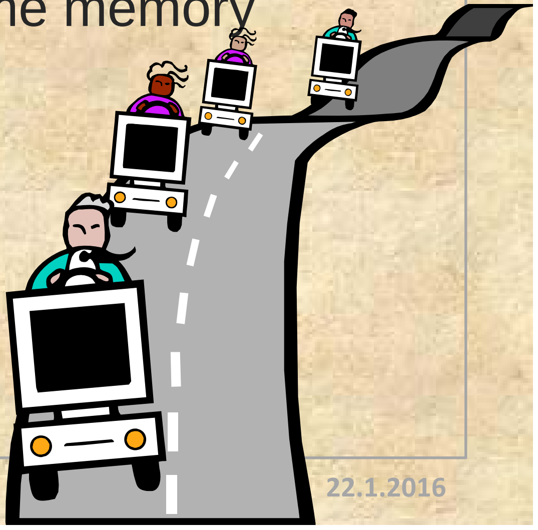
Relocation	uudelleensijoitus
Protection	suojaus
Sharing	yhteiskäyttö
Logical organization	looginen muistin organisointi
Physical organization	fyysinen muistin organisointi



Relocation

uudelleensijoitus

- Programmers typically do not know in advance which other programs will be resident in main memory at the time of execution of their program
- Active processes need to be able to be swapped in and out of main memory in order to maximize processor utilization
- Specifying that a process must be placed in the same memory region when it is swapped back in would be limiting
  - May need to *relocate* the process to a different area of memory
  - What is needed to make relocation possible?



Addressing Requirements

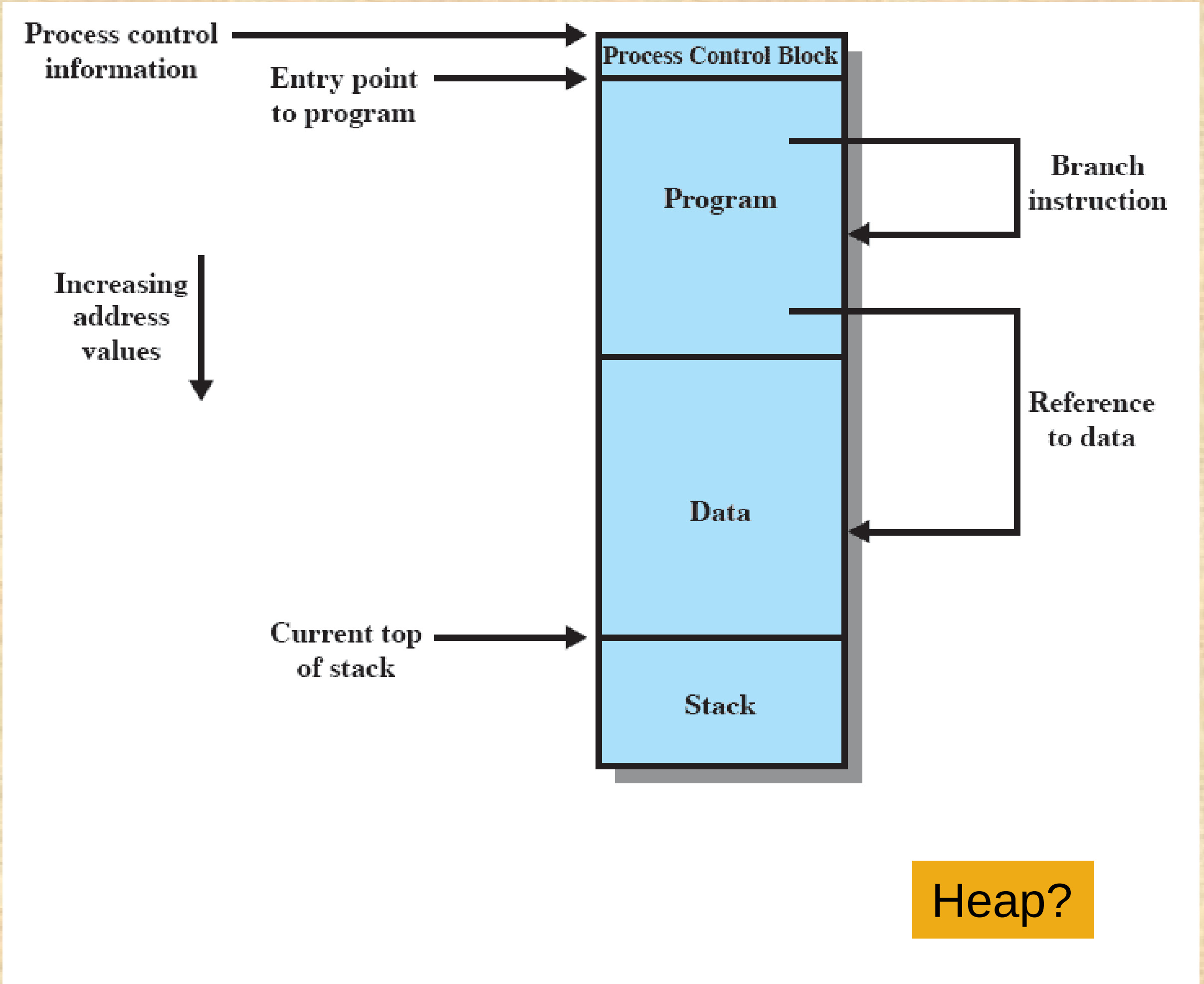


Figure 7.1 Addressing Requirements for a Process

Protection

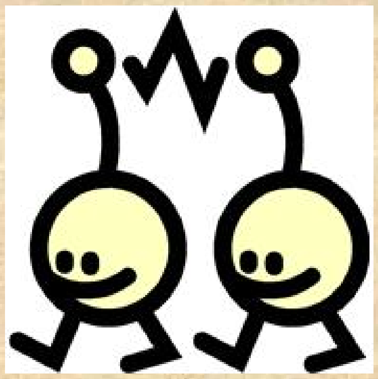
suojaus

- Processes need to acquire permission to reference memory locations for reading or writing purposes
- Location of a program in main memory is unpredictable
- Memory references generated by a process must be checked at run time
  - Memory references known only at run time
  - Must have HW support for protection
- Mechanisms that support relocation also support protection

Sharing

Yhteiskäyttö
(ei ”jakaminen”)

- Advantageous to allow each process access to the same copy of the (module of the) program rather than have their own separate copy
- Memory management must allow controlled access to shared areas of memory without compromising protection
- Mechanisms used to support relocation also support sharing capabilities



Logical Organization

- Memory is organized as linear

Programs are written in modules

- modules can be written and compiled independently
- different degrees of protection given to modules (read-only, execute-only)
- sharing on a module level corresponds to the user's way of viewing the problem

- Segmentation is the tool that most readily satisfies requirements

Physical Organization

Cannot leave the programmer with the responsibility to manage memory

Memory available for a program plus its data may be insufficient

Programmer does not know how much space will be available

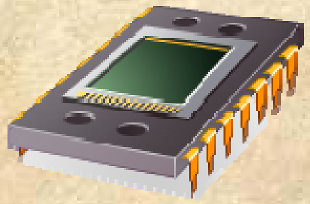
overlaying allows various modules to be assigned the same region of memory but is time consuming to program

kerrostus

partitiointi, ositus

Memory Partitioning

- Memory management brings processes into main memory for execution by the processor
  - Involves virtual memory
  - Based on segmentation and paging
- Partitioning
  - Used in several variations in some now-obsolete operating systems
  - Still used in many real time operating systems
  - Does not involve virtual memory



kiinteät partitiot

Fixed Partitioning

- Equal-size partitions
  - any process whose size is less than or equal to the partition size can be loaded into an available partition
- The operating system can swap out a process if all partitions are full and no process is in the Ready or Running state

Discuss

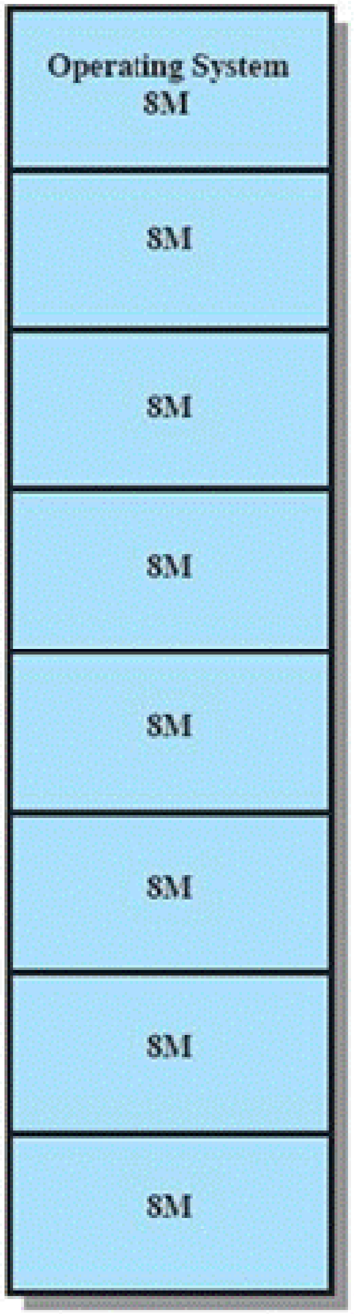
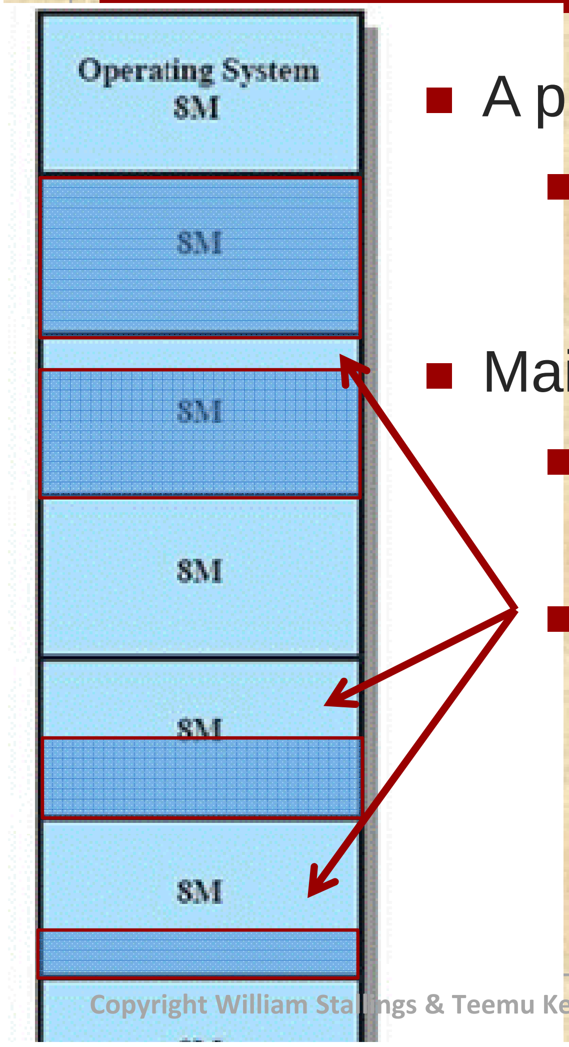


Fig. 7.2
(a) Equal-size partitions

Disadvantages



- A program may be too big to fit in a partition
  - program needs to be designed with the use of overlays
- Main memory utilization is inefficient
  - any program, regardless of size, occupies an entire partition
  - ***internal fragmentation*** sisäinen pirstoutuminen
    - wasted space due to the block of data loaded being smaller than the partition

Unequal Size Partitions

eri kokoiset partitiot

- Using unequal size partitions helps lessen the problems
  - programs up to 16M can be accommodated without overlays
  - partitions smaller than 8M allow smaller programs to be accommodated with less internal fragmentation

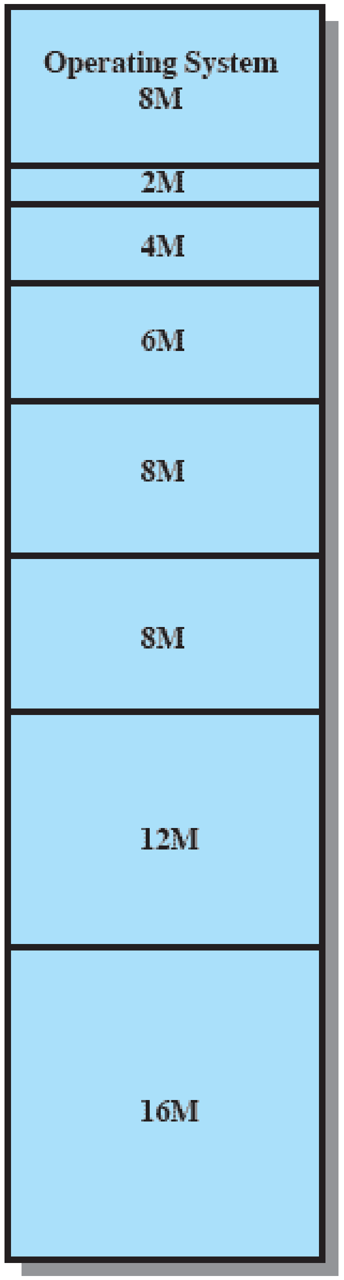


Fig. 7.2
(b) Unequal-size partitions

muistin varaus

Fixed Partitioning

Memory Assignment

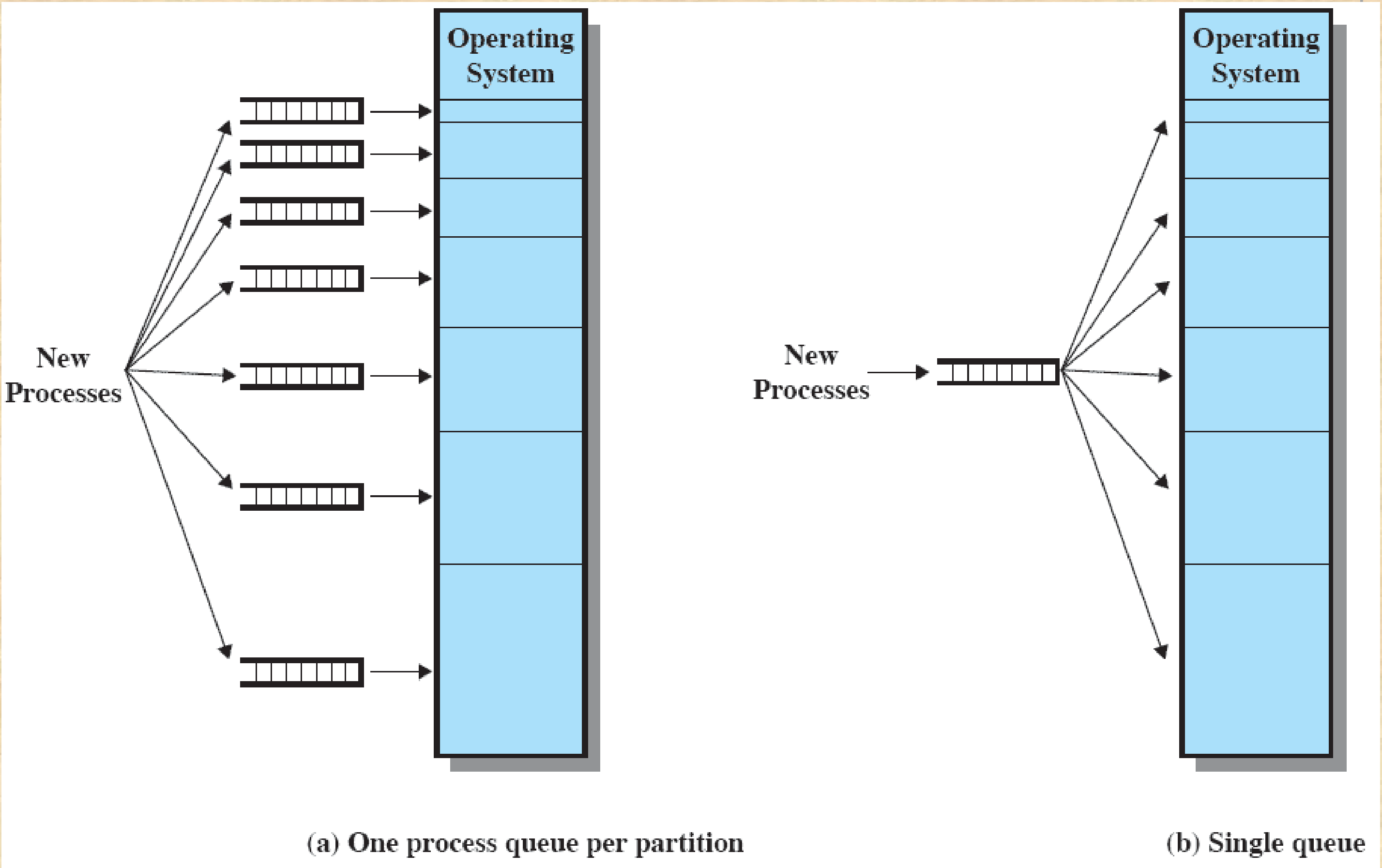


Figure 7.3 Memory Assignment for Fixed Partitioning

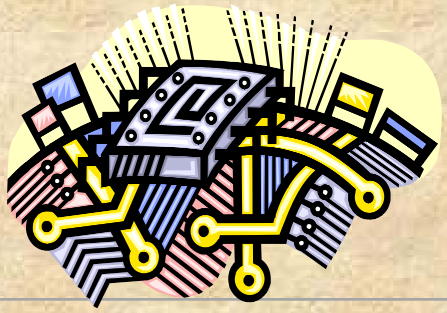
Unequal Size Partition Disadvantages

- The number of partitions specified at system generation time limits the number of active processes in the system
- Small jobs will not utilize partition space efficiently

dynaaminen partitionti

Dynamic Partitioning

- Partitions are of variable length and number
  - Partition allocation when program starts
- Process is allocated exactly as much memory as it requires
- This technique was used by IBM's mainframe operating system, OS/MVT



External Fragmentation of Dynamic Partitioning

ulkoinen pirstoutuminen

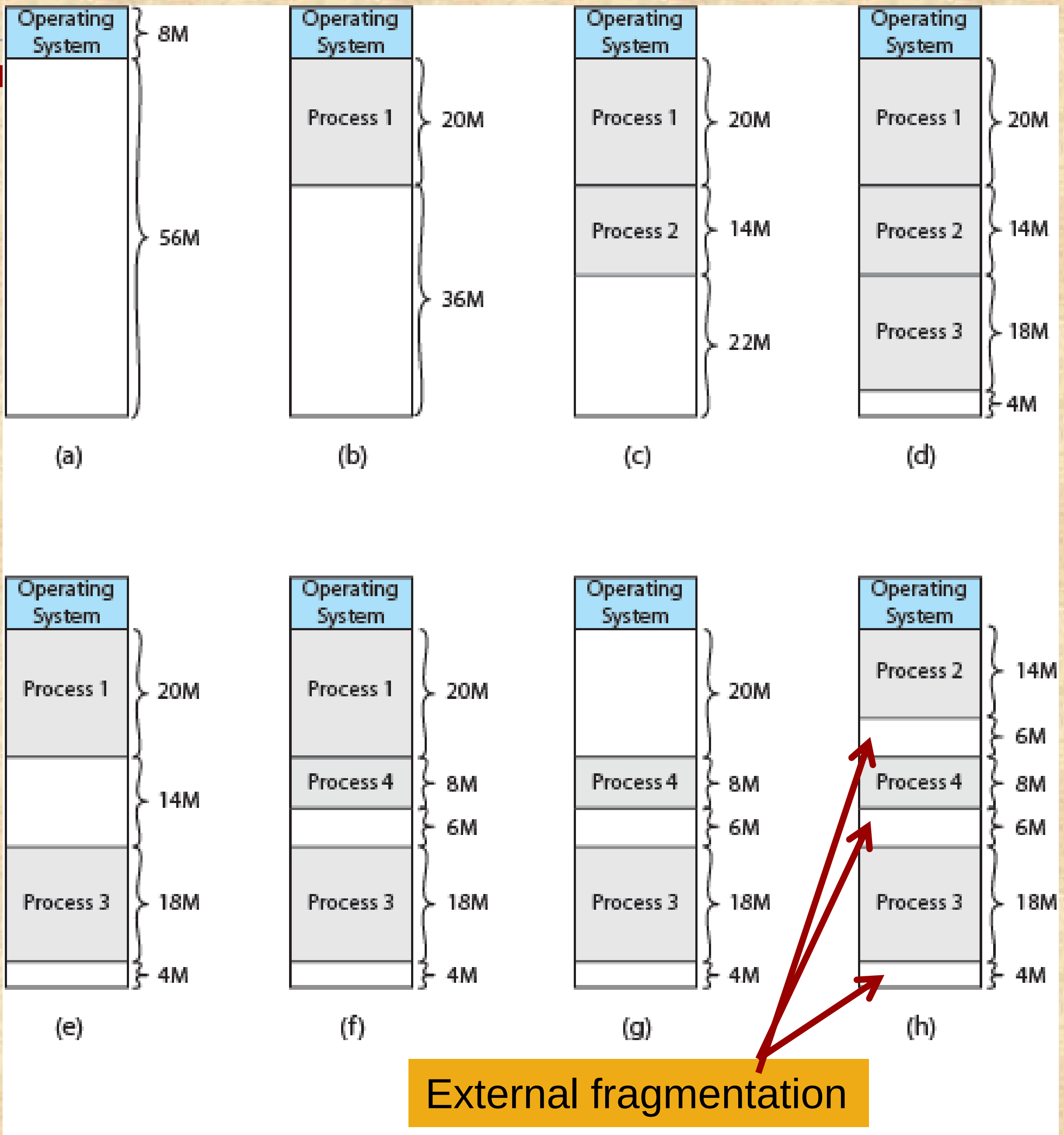


Figure 7.4 The Effect of Dynamic Partitioning

Dynamic Partitioning

External Fragmentation

ulkoinen pirstoutuminen

- memory becomes more and more fragmented
- memory utilization declines

Compaction

tiivistys

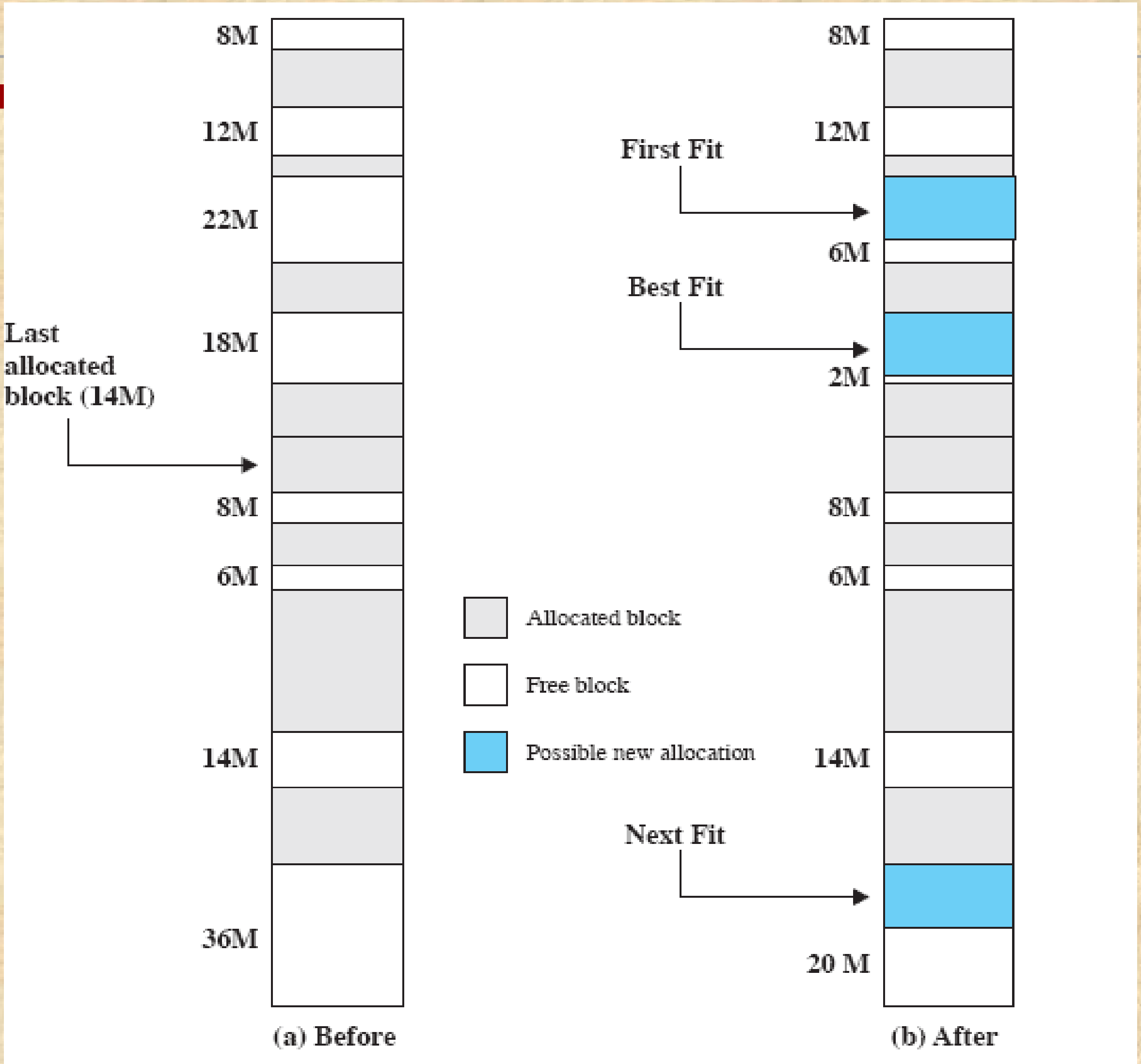
- technique for overcoming external fragmentation
- OS shifts processes so that they are contiguous
- free memory is together in one block
- time consuming and wastes CPU time

sijoitusalgoritmit

Placement Algorithms

Best-fit	First-fit	Next-fit
• chooses the block that is closest in size to the request	• begins to scan memory from the beginning and chooses the first available block that is large enough	• begins to scan memory from the location of the last placement and chooses the next available block that is large enough

Memory Configuration Example



Discuss

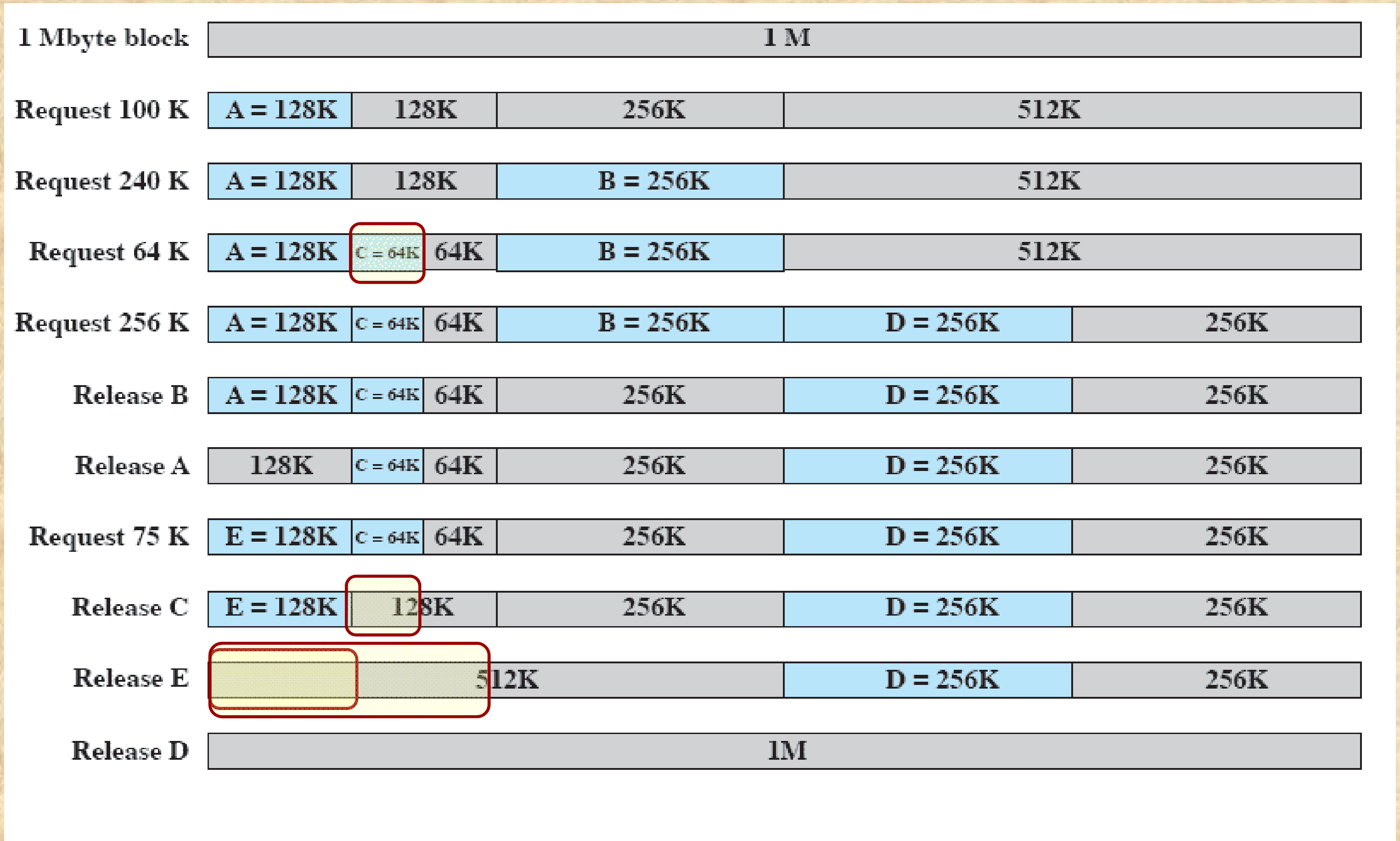
Figure 7.5 Example Memory Configuration before and after Allocation of 16-Mbyte Block

Buddy System

Buddy-systeemi
vieruskaveri

- Comprised of fixed and dynamic partitioning schemes
- Space available for allocation is treated as a single block
- Memory blocks are available of size 2^K *words*, $L \le K \le U$, where
  - 2^L = *smallest size block that is allocated*
  - 2^U = largest size block that is allocated; generally 2^U is the size of the entire memory available for allocation

Buddy System Example



1 Mbyte block | 1 M
Request 100 K | A = 128K | 128K | 256K | 512K
Request 240 K | A = 128K | 128K | B = 256K | 512K
Request 64 K | A = 128K | C = 64K | 64K | B = 256K | 512K
Request 256 K | A = 128K | C = 64K | 64K | B = 256K | D = 256K | 256K
Release B | A = 128K | C = 64K | 64K | 256K | D = 256K | 256K
Release A | 128K | C = 64K | 64K | 256K | D = 256K | 256K
Request 75 K | E = 128K | C = 64K | 64K | 256K | D = 256K | 256K
Release C | E = 128K | 128K | 256K | D = 256K | 256K
Release E | 512K | D = 256K | 256K
Release D | 1M

Figure 7.6 Example of Buddy System

Discuss

Tree Representation

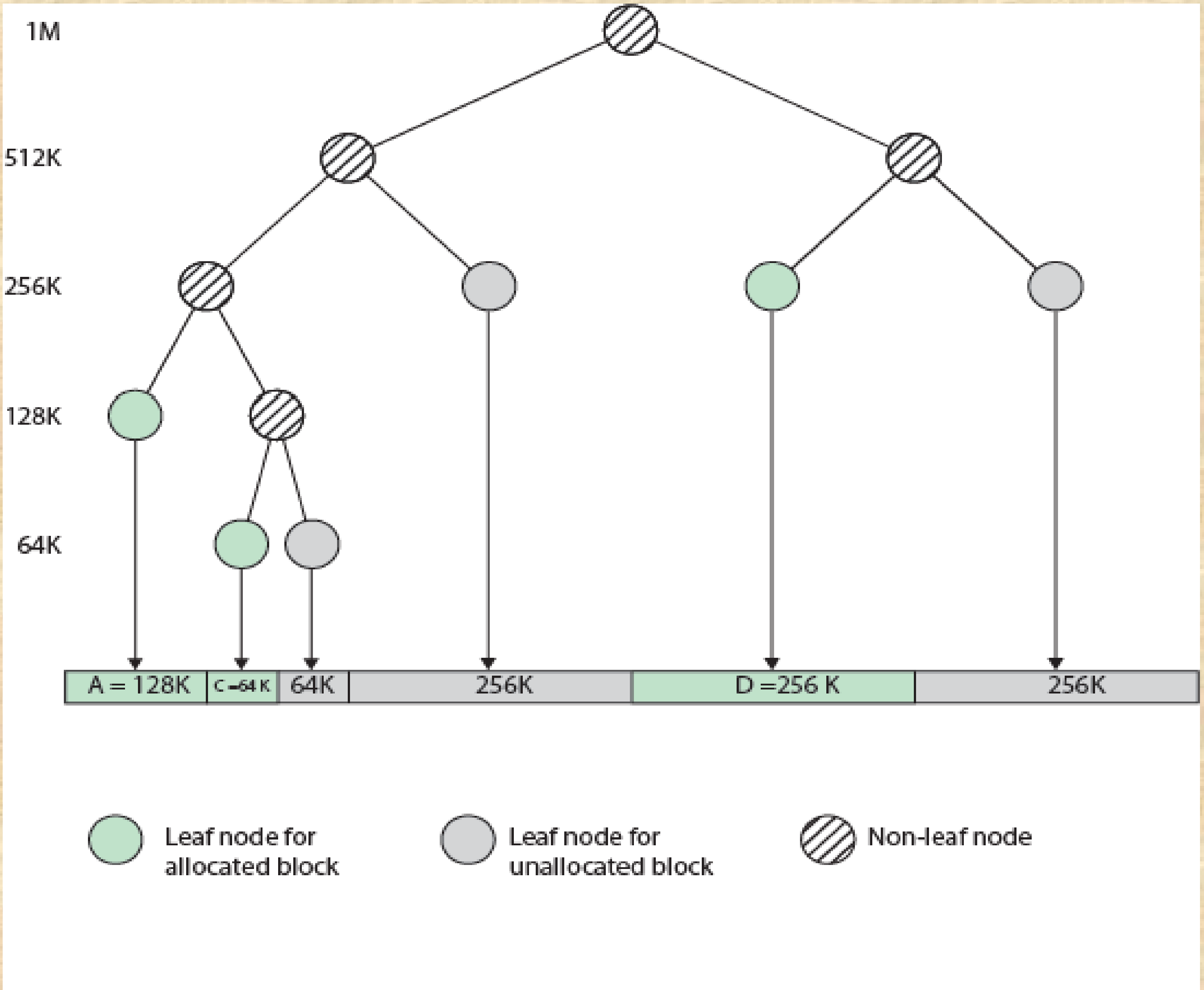


Figure 7.7 Tree Representation of Buddy System

Addresses

Logical looginen

- reference to a memory location <u>independent of the current assignment</u> of data to memory

Relative suhteellinen

- address is expressed as a location <u>relative to some known point</u>

PC, SP, HP, Base-register, …

Physical or Absolute fyysinen, absoluuttinen

- <u>actual location</u> in main memory

Address Calculation with Relocation

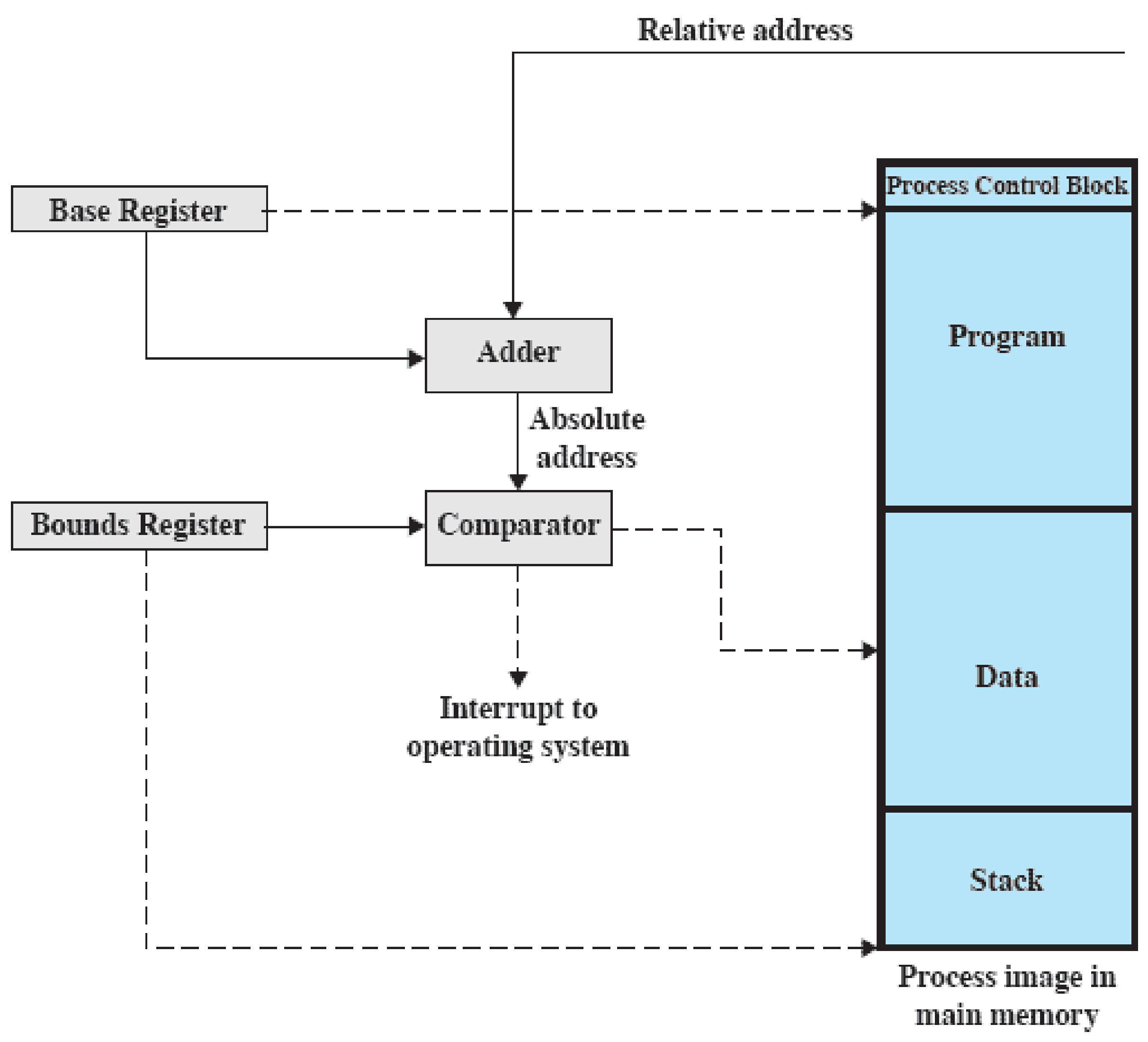


Figure 7.8 Hardware Support for Relocation

Discuss

Paging

sivutus

- Partition memory into equal <u>fixed-size chunks</u> that are relatively small
- Process is also divided into small <u>fixed-size chunks</u> of the <u>same size</u>

sivu **_Pages_**	**_Frames_** kehys, sivukehys
• chunks of a process (address space)	• available chunks of memory

Assignment of Process to Free Frames

frame
sivukehys,
kehys

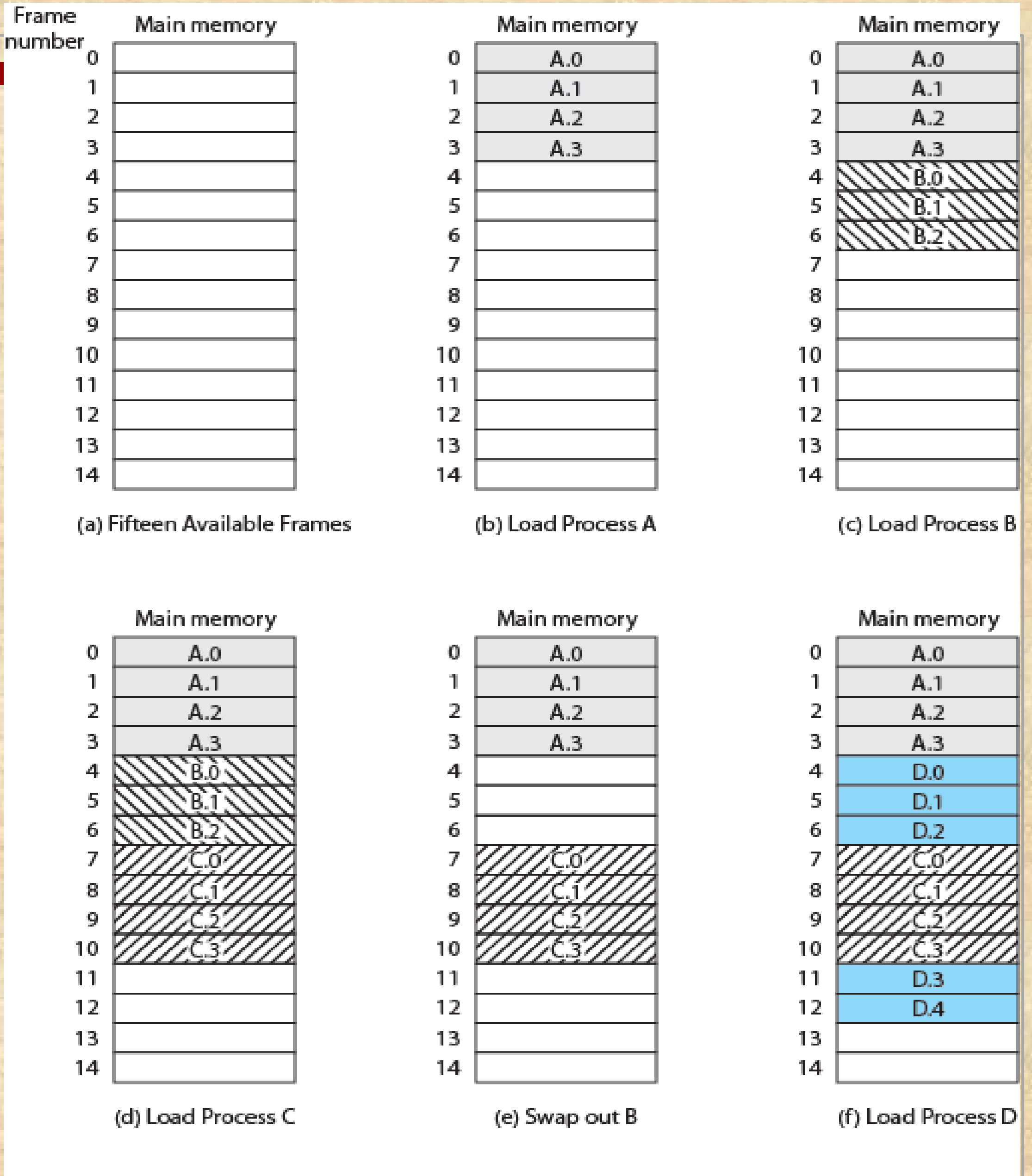
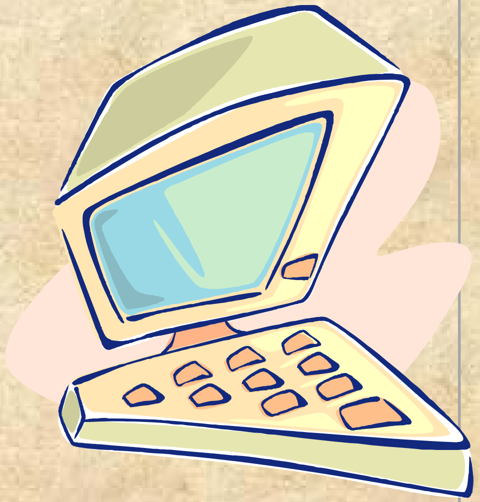


Figure 7.9 Assignment of Process Pages to Free Frames

Page Table

sivutaulu

- Maintained by operating system for each process
- Contains the frame location for each page in the process
  - No virtual memory ⟶ All pages are in main memory!
- Processor must know how to access for the current process
- Used by processor to produce a physical address



Data Structures

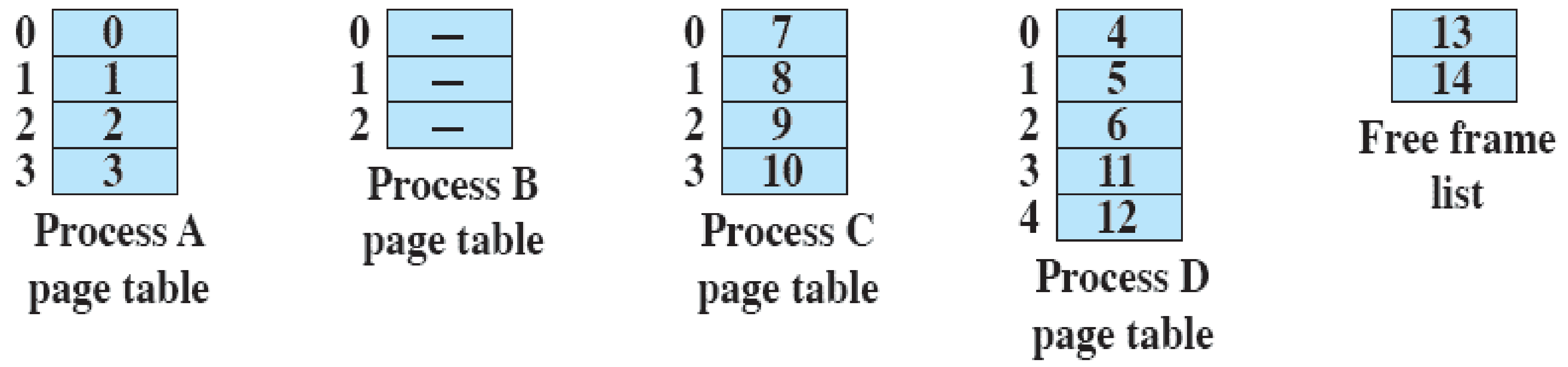


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

Segmentation

segmentointi

- A program can be subdivided into segments
  - may vary in length
  - there is a maximum length
- Addressing consists of two parts:
  - segment number
  - an offset
- Similar to dynamic partitioning
- Eliminates internal fragmentation

sisäinen pirstoutuminen

Logical Addresses

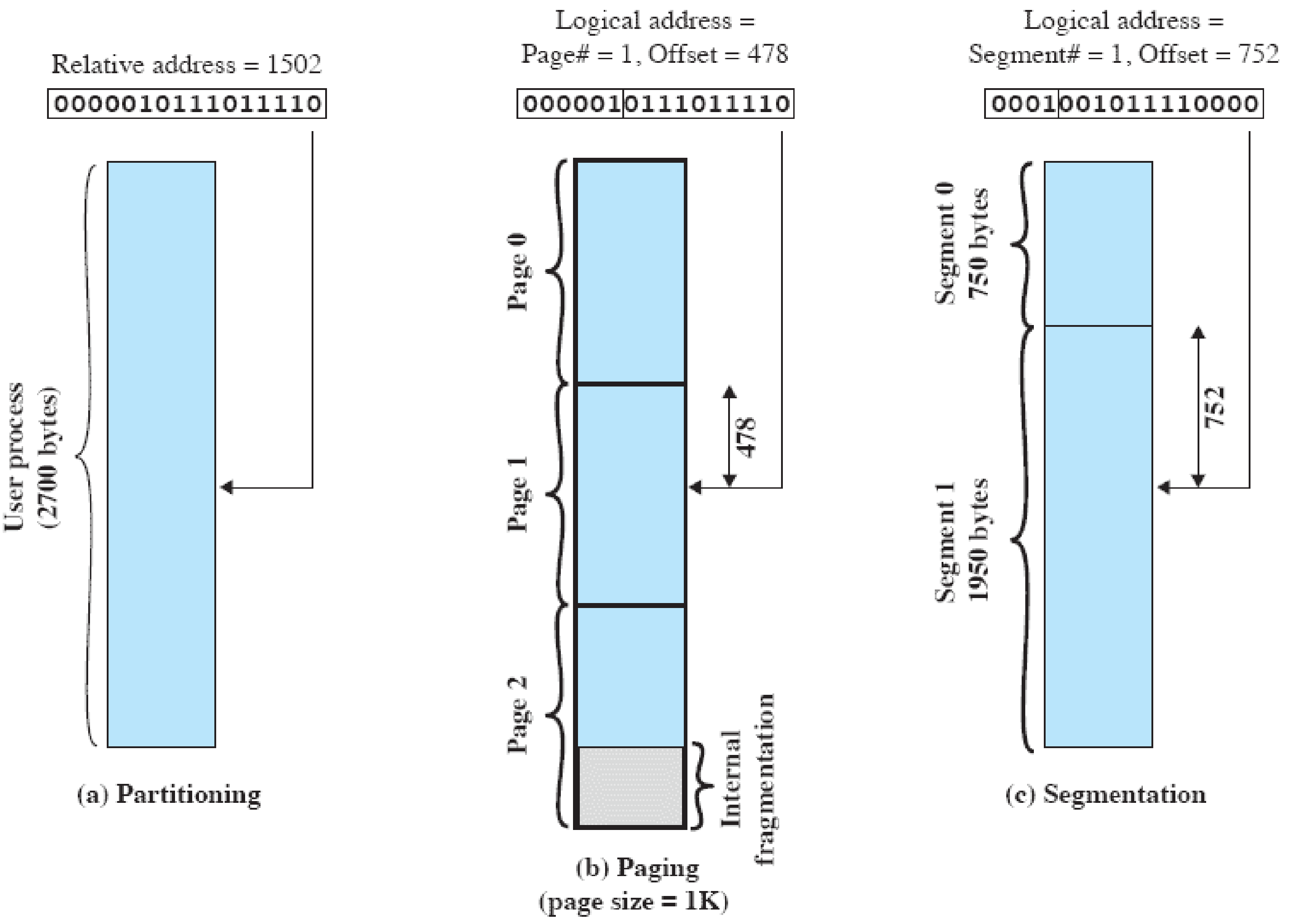


Figure 7.11 Logical Addresses

Logical-to-Physical Address Translation - Paging

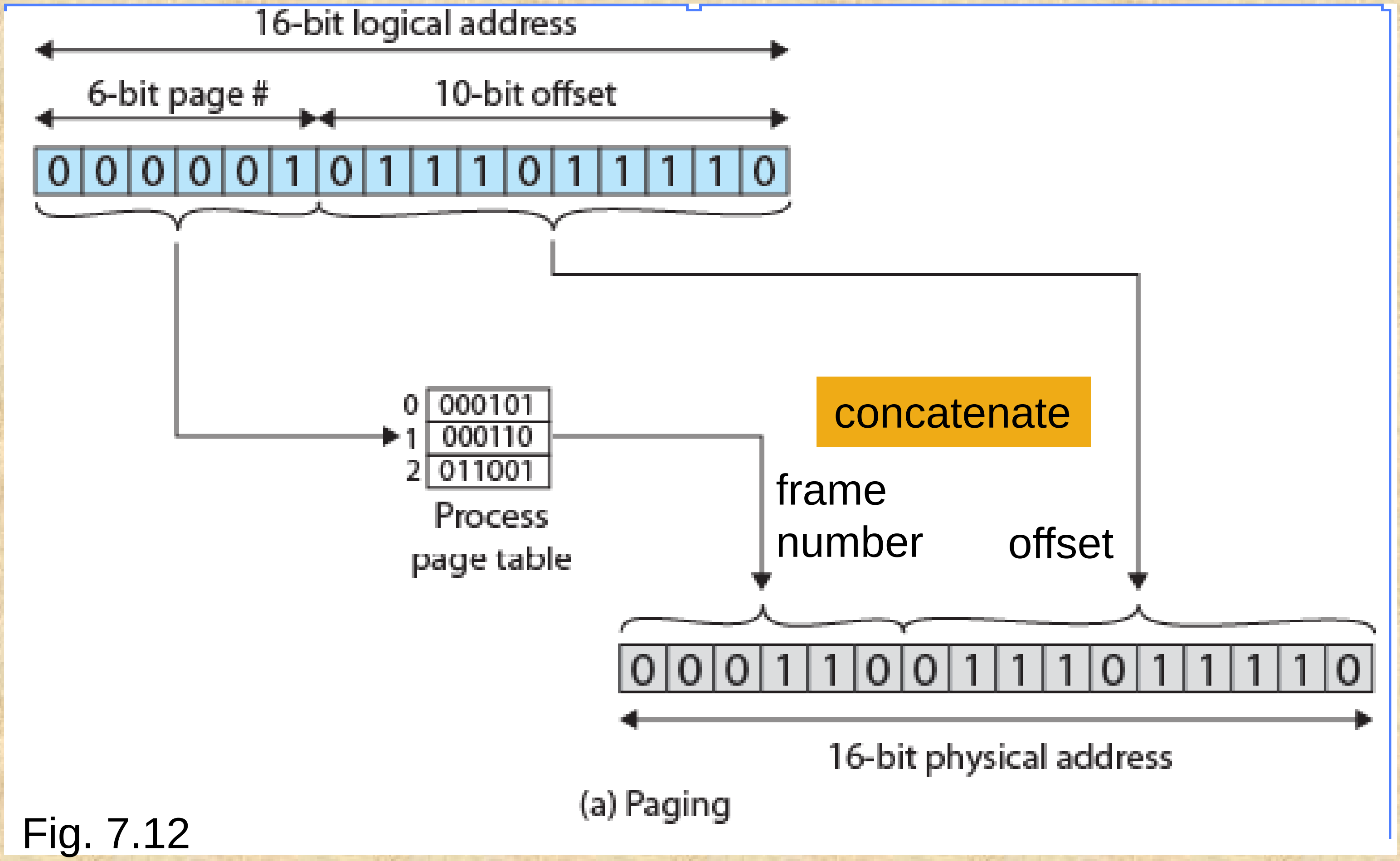


Fig. 7.12

Logical-to-Physical Address Translation - Segmentation

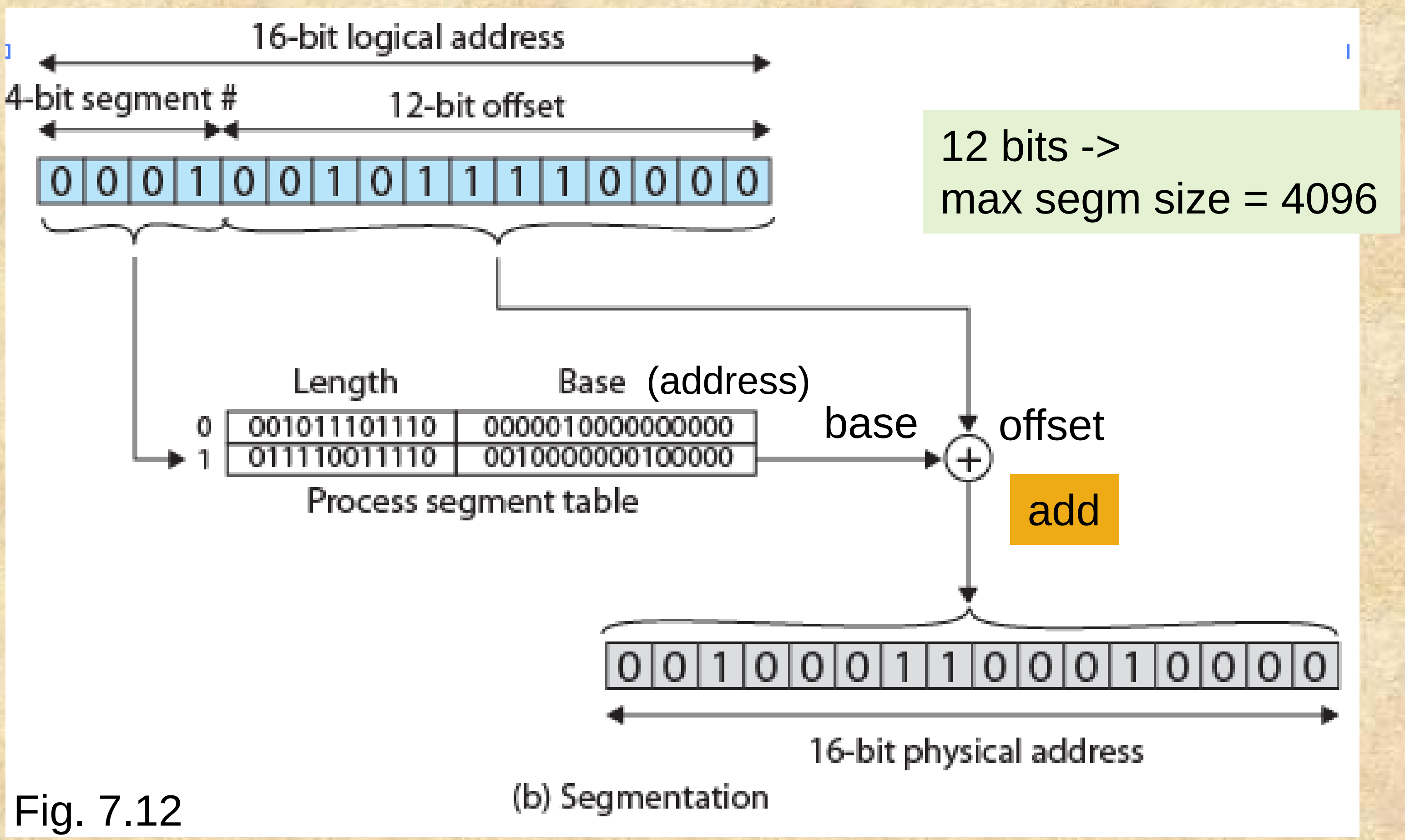


Fig. 7.12

Table 7.2 Memory Management Techniques

Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.
Simple Paging	Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames.	No external fragmentation.	A small amount of internal fragmentation.
Simple Segmentation	Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.	No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.	External fragmentation.
Virtual Memory Paging	As with simple paging, except that it is not necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically.	No external fragmentation; higher degree of multiprogramming; large virtual address space.	Overhead of complex memory management.
Virtual Memory Segmentation	As with simple segmentation, except that it is not necessary to load all of the segments of a process. Nonresident segments that are needed are brought in later automatically.	No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support.	Overhead of complex memory management.

Security Issues

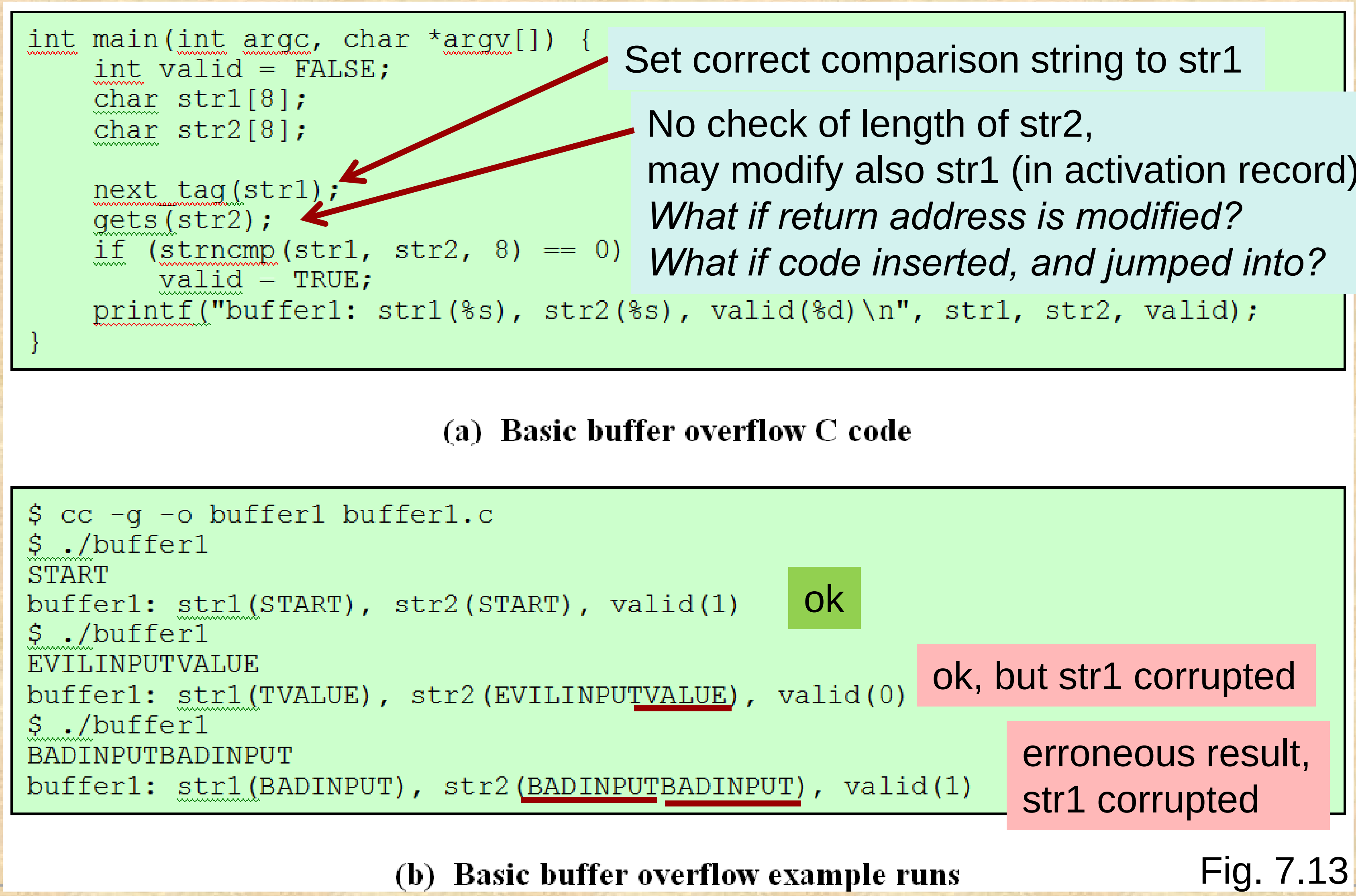
If a process has not declared a portion of its memory to be sharable, then no other process should have access to the contents of that portion of memory

If a process declares that a portion of memory may be shared by other designated processes then the security service of the OS must ensure that only the designated processes have access

Buffer Overflow Attacks

- Security threat related to memory management
- Also known as a buffer overrun
- Can occur when a process attempts to store data beyond the limits of a fixed-sized buffer
- One of the most prevalent and dangerous types of security attacks

Buffer Overflow Example



```
int main(int argc, char *argv[]) {
    int valid = FALSE;
    char str1[8];
    char str2[8];

    next_tag(str1);
    gets(str2);
    if (strncmp(str1, str2, 8) == 0)
        valid = TRUE;
    printf("buffer1: str1(%s), str2(%s), valid(%d)\n", str1, str2, valid);
}
```

Set correct comparison string to str1

No check of length of str2,
may modify also str1 (in activation record).
What if return address is modified?
What if code inserted, and jumped into?

(a) Basic buffer overflow C code

```
$ cc -g -o buffer1 buffer1.c
$ ./buffer1
START
buffer1: str1(START), str2(START), valid(1)
$ ./buffer1
EVILINPUTVALUE
buffer1: str1(TVALUE), str2(EVILINPUTVALUE), valid(0)
$ ./buffer1
BADINPUTBADINPUT
buffer1: str1(BADINPUT), str2(BADINPUTBADINPUT), valid(1)
```

ok

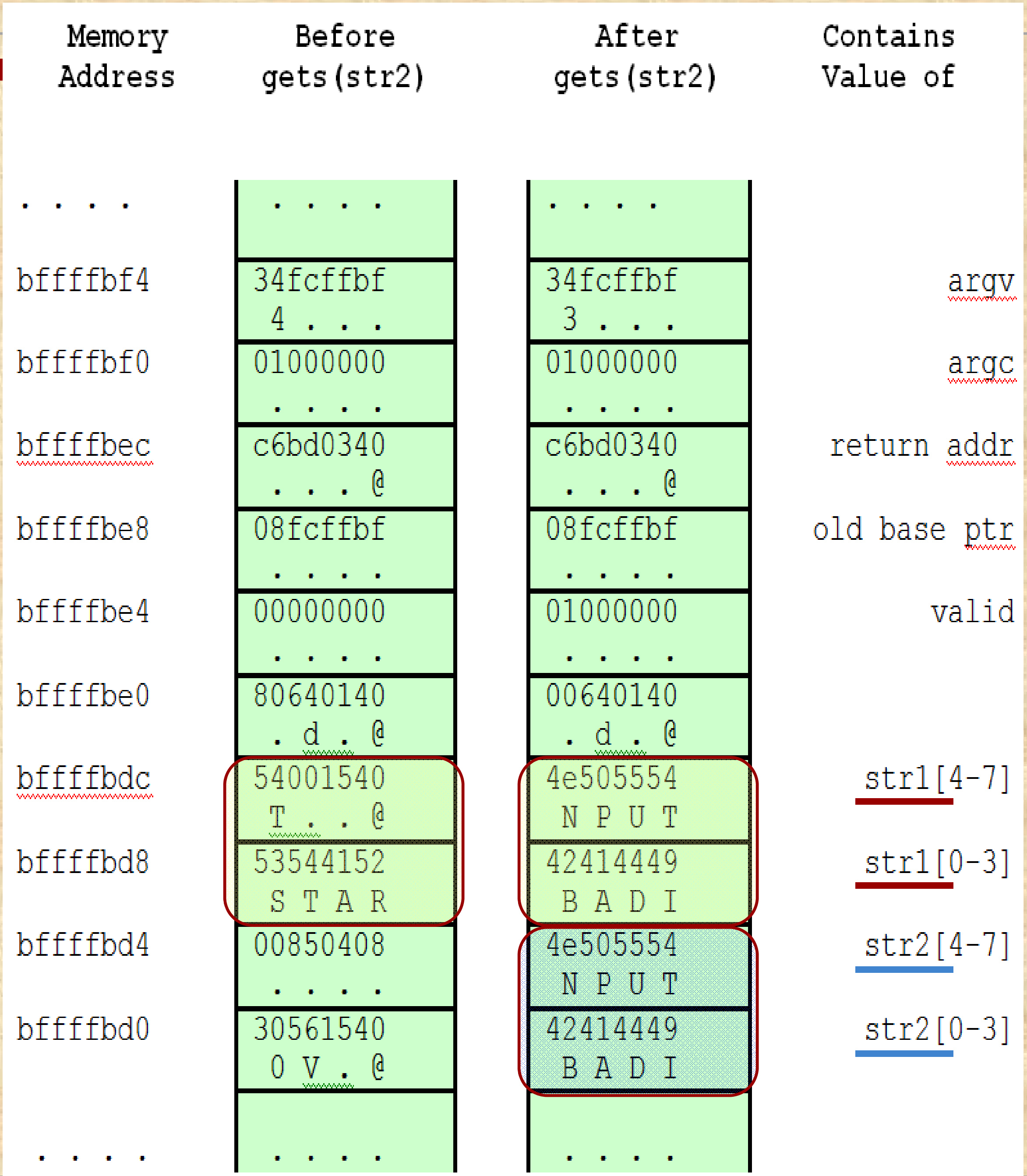
ok, but str1 corrupted

erroneous result,
str1 corrupted

(b) Basic buffer overflow example runs

Fig. 7.13

Buffer Overflow Stack Values



Memory Address	Before gets(str2)	After gets(str2)	Contains Value of
. . . .	. . . .	. . . .	
bffffbf4	34fcffbf 4 . . .	34fcffbf 3 . . .	argv
bffffbf0	01000000	01000000	argc
bffffbec	c6bd0340 . . . @	c6bd0340 . . . @	return addr
bffffbe8	08fcffbf	08fcffbf	old base ptr
bffffbe4	00000000	01000000	valid
bffffbe0	80640140 . d . @	00640140 . d . @	
bffffbdc	54001540 T . . @	4e505554 N P U T	str1[4-7]
bffffbd8	53544152 S T A R	42414449 B A D I	str1[0-3]
bffffbd4	00850408	4e505554 N P U T	str2[4-7]
bffffbd0	30561540 0 V . @	42414449 B A D I	str2[0-3]
. . . .	. . . .	. . . .	

How would you fix the problem?

Defending Against Buffer Overflows

- Prevention

E.g., do not use gets()

fgets(str2, 8, stdin)

- Detecting and aborting

E.g., save token names as read only constants

- Countermeasure categories:

Compile-time Defenses

Automatic check for index overflow code, etc.

See Ch 15

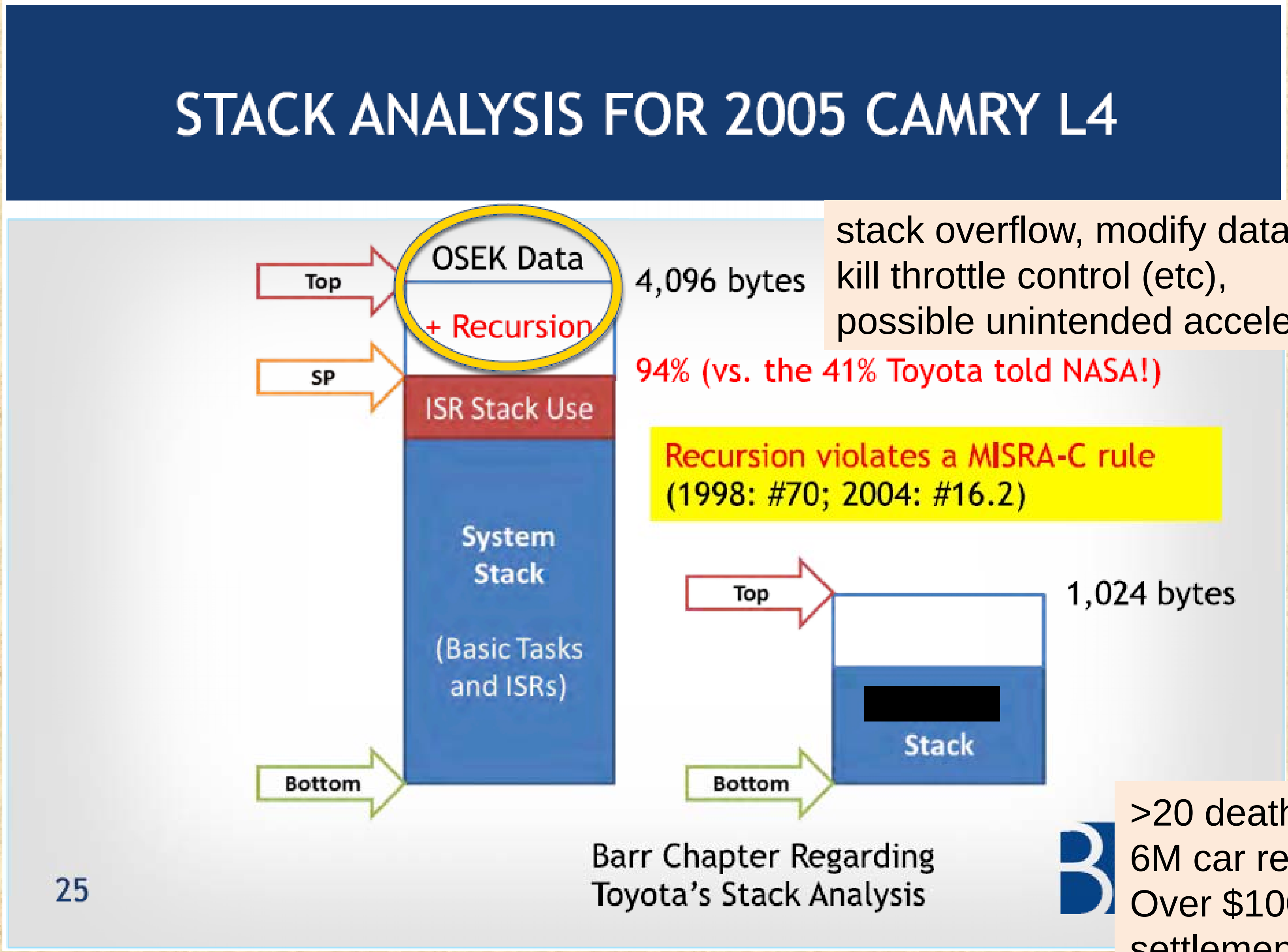
- aim to harden programs to resist attacks in new programs

Run-time Defenses

- aim to detect and abort attacks in existing programs

Keep literals in read-only segment.
Disallow code execution in stack or heap.
Mark all code non-executable by default. Etc.

Bad end result of accidental buffer (stack) overflow:



STACK ANALYSIS FOR 2005 CAMRY L4

94% (vs. the 41% Toyota told NASA!)

Recursion violates a MISRA-C rule
(1998: #70; 2004: #16.2)

25

Barr Chapter Regarding
Toyota's Stack Analysis

stack overflow, modify data,
kill throttle control (etc),
possible unintended acceleration

>20 deaths.
6M car recalled.
Over $1000M
settlement.

http://www.safetyresearch.net/Library/BarrSlides_FINAL_SCRUBBED.pdf

Michel Barr

Summary

- Memory Management
  - one of the most important and complex tasks of an operating system
  - needs to be treated as a resource to be allocated to and shared among a number of active processes
  - desirable to maintain as many processes in main memory as possible
  - desirable to free programmers from size restriction in program development
  - basic tools are paging and segmentation (possible to combine)
    - paging – small fixed-sized pages
    - segmentation – pieces of varying size