2026 CS 332 Subject 8

Alex. Gerbessiotis CS Department

The yellowish/brownish notes are copyright the textbook author.

The bluish notes are copyright the instructor and are add-ons and provide more explanation or additional details

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Module 1

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Operating Systems: Internals and Design Principles

Chapter 8 Virtual Memory

Ninth Edition

William Stallings

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

(Simple)Paging vs Virtual Memory

Subject 7 (Simple) Paging	Subject 8 Demand Paging or Virtual Memory (using paging)
ProgramSize depends on MainMemorySize	ProgramSize independent of MainMemory Size [Depends on Disk size]
(ProgramSize <= MM always)	(ProgramSize > MM is possible)
All of Program IN or All out of MainMemory	Some IN but MOST out of MainMemory
Page Table is an Array	Page Table is a tree-like structure
PTE (Page Table Entry) contains FrameNumber	PTE (Page Table Entry) contains FrameNumber Present Bit (is Page in MM?) Modified Bit (was Page modified in MM?) Control Bit (additional info)

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

(Simple)Paging vs Virtual Memory

Subject 7

Logical Address or
Linear Address

Logical Address

Physical Address or
Absolute Address or
Actual Address

Physical Address

Subject 8

Logical Address or
Virtual Address

Prevailing: Virtual Address

Physical Address or
Real Address

Prevailing: Real or Physical Address

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Virtual memory also known as demand paging	A storage allocation scheme in which secondary memory can be addressed as though it were part of main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program-generated addresses are translated automatically to the corresponding machine addresses.The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of secondary memory available and not by the actual number of main storage locations.
Virtual address	The address assigned to a location in virtual memory to allow that location to be accessed as though it were part of main memory.
Virtual address space	The virtual storage assigned to a process.
Address space	The range of memory addresses available to a process.
Real address	The address of a storage location in main memory.

Table 8.1 Virtual Memory Terminology

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Virtual Memory

Disk

A			
	B		
		C	
		D	E

Logical View

0	A
1	B
2	C
3	D
4	E

Page Table

0	2	P:1
1	4	P:1
2		P:0
3	6	P:1
4		P:0

Memory

0	
1	
2	A
3	
4	B
5	
6	D
7	

!!!Present bit is denoted ... P
it is not a ... page ..!!!

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Why Virtual Memory

Benefits of using virtual memory

(a) PORTABILITY: no matter how much main memory is available, program(s) would work

(b) EFFICIENCY: main memory is not wasted on parts of program not used by CPU

(c) VERSATILITY/SCALABILITY Code can grow larger than main memory

Page Table maintains mapping between
pages (virtual addresses) and frames (physical addresses)

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Hardware and Control Structures

- Two characteristics fundamental to memory management:
  1) All memory references are logical addresses that are dynamically translated into physical addresses at run time
  2) A process may be broken up into a number of pieces that don't need to be contiguously located in main memory during execution
- If these two characteristics are present, it is not necessary that all of the pages or segments of a process be in main memory during execution

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Execution of a Process

- Operating system brings into main memory a few pieces of the program
- Resident set
  - Portion of process that is in main memory
- An interrupt is generated when an address is needed that is not in main memory
- Operating system places the process in a blocking state

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Continued . . .

Execution of a Process

- Piece of process that contains the logical address is brought into main memory
  - Operating system issues a disk I/O Read request
  - Another process is dispatched to run while the disk I/O takes place
  - An interrupt is issued when disk I/O is complete, which causes the operating system to place the affected process in the Ready state
  - **UPDATE ALSO PAGE TABLE !!!**

PAGER (vs swapper that move all of 'pages' in or out)

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Execution of a process: example

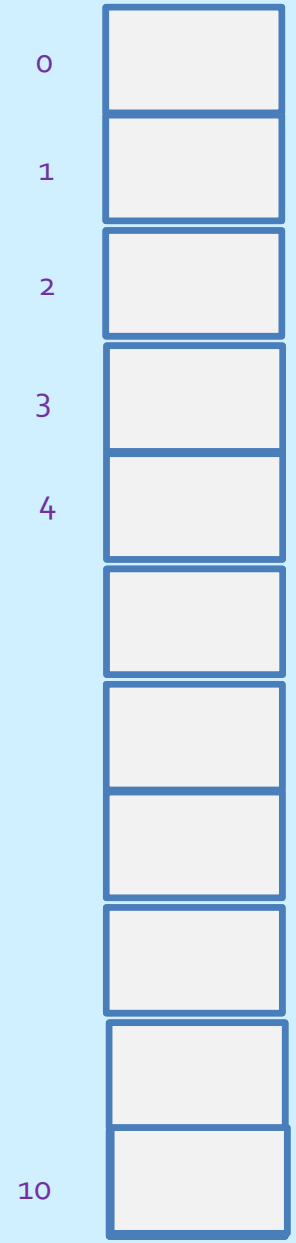
|S|=|R|=100B

```
.text   0      LOAD AC  100
        1      ADD  AC  200
        2      ADD  AC  300
             ... lots of zero bytes ...
      100      51
             ... lots  of zero bytes ...
      200      61
             ... lots of zero bytes
      300      71
             ... lots of zero bytes
```

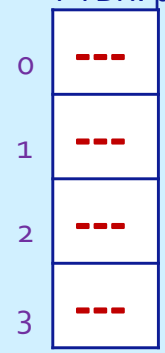
3-instruction program

ProgramSize = 400B

MM:
11 frames



PTBR: page table



PC=0
AC= -
IR = -

```
ProgramSize=400B ,  PageSize=S=100B
#pagesforProgram= ceiling(ProgramSize/S)
                = ceil(400/100)=4
PTBR.length     = 4
PTBR.size       = 4*sizeof(PTBR)=4*?4B?=16B

MM.size = MM.lengthinFrames * R = 11*100=1100B
```

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Execution of a process: example

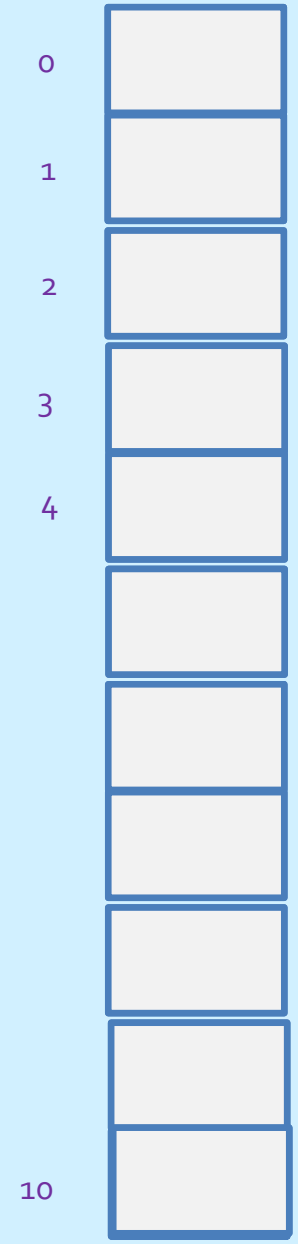
|S|=|R|=100B

.text

```
LOAD AC 100
ADD  AC 200
ADD  AC 300
```

3-instruction program

MM:
11 frames



PTBR: page table

0	---
1	---
2	---
3	---

PC=0
AC= -
IR = -

```
Instruction-1 FETCH STAGE
  PC = 0 => L=PC=0 => L=(P,T)=(0,0)=<0,0>
            F=PTBR[P] = PTBR[0] = --- page fault
            FETCH STAGE STALL …
Instruction-1 fetch stage page fault servicing takes over
<page fault: Pager brings page  0 into MM from SM @20ms
Page 0 gets mapped to Frame 10; PTBR updated (next slide)

Instruction-1 FETCH STAGE to be rexecuted
```

Cost this slide:

```
 1instruction stalled  +
80ns   (PTBR[0])       +
20ms   (pager)         ~  20ms
```

We thought this slide would cost

```
1instruction +
2 mem accesses         ~ 160ns
```

20ms/160ns = 20,000,000/160 > 100,000 slowdown

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Execution of a process: example

|S|=|R|=100B

.text
LOAD AC 100
ADD AC 200
ADD AC 300

3-instruction program

MM:
11 frames

PTBR: page table

0	10
1	---
2	---
3	---

PC=0
AC=-

IR=LOAD AC 100

```
Instruction-1 FETCH STAGE
  PC = 0 => L=PC=0 => L=(P,T)=(0,0)=<0,0>
            F=PTBR[P] = PTBR[0] = 10
            A=(F,T)=(10,0)=10*100+0=1000
            IR=M[A]       // IR = LOAD AC 100
Instruction-1 FETCH STAGE completed
```

Resident set (set of pages in MM) : page 0 ->frame 10
Resident set size is 1

Cost this slide:
1 instruction stalled
3*80ns +
20ms (pager) ~ 20ms

PTBR: page table

0	10
1	---
2	---
3	---

PC=1
AC=-

IR=LOAD AC 100

```
Instruction-1 EXECUTE STAGE
IR=LOAD AC 100 => L=100 => L=(P,T)=(1,0)=<1,0>
            F=PTBR[P] = PTBR[1] = --- page fault
            EXECUTE STAGE STALL …
Instruction-1 execute stage page fault servicer takes over
<page fault: Pager brings page  1 into MM from SM @20ms
Page 1 gets mapped to Frame 3; PTBR updated (next slide)

Instruction-1 EXECUTE STAGE to be rexecuted!
```

0	
1	
2	
3	
4	
10	LOAD AC 100 ADD AC 200 ADD AC 300

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Execution of a process: example

|S|=|R|=100B

.text

```
LOAD AC 100
ADD  AC 200
ADD  AC 300
```

3-instruction program

MM: 11 frames

Resident set (set of pages in MM) : 0,1

Working set (set of pages in related to execution of current instruction) : 0,1

Cost this slide:
1 instruction
2*80ns ~ 160ns

1st instruction : execute stage rexecuted

PTBR: page table

0	10
1	3
2	---
3	---

PC=1

AC=51

IR=LOAD AC 100

```
Instruction-1 EXECUTE STAGE reexecution
IR=LOAD AC 100 => L=100 => L=(P,T)=(1,0)=<1,0>
            F=PTBR[P] = PTBR[1] = 3
            A=(F,T)= (3,0) = 300
            AC = M[A] = M[300] = 51 (denary)
Instruction-1 EXECUTE STAGE completed
Instruction-1 completed (finally!)
```

0	
1	
2	
3	Denary 51
4	
10	LOAD AC 100 ADD AC 200 ADD AC 300

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Execution of a process: example

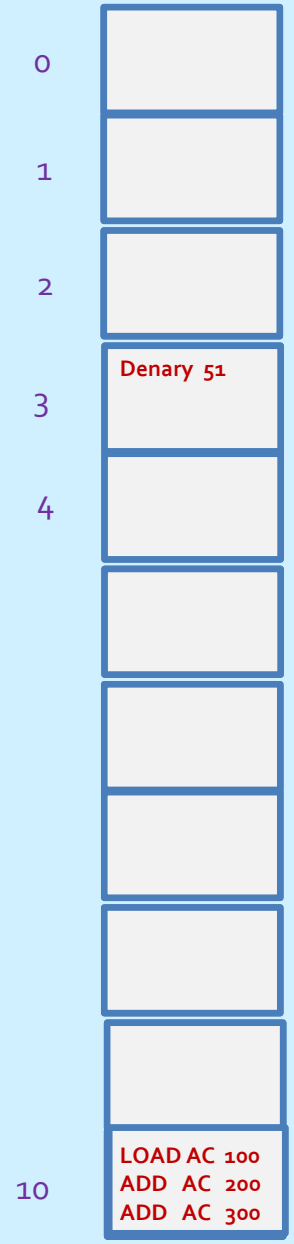
|S|=|R|=100B

.text

```
LOAD AC  100
ADD  AC  200
ADD  AC  300
```

3-instruction program

MM:
11 frames



Theory

1 instruction ~
2x16ons=32ons
(fetch:16ons;execute:16ons)

SLOWDOWN = 40ms / 32ons
=40x 10**-3 / 320x 10**-9
= 1,000,000 / 8
=125,000
~ 100,000

3,000,000 instruction
roughly
1s execution

could become
due to
page faults

Practice

Two page faults+ …
2x20ms ~ 40ms

100,000s execution
which is roughly
1+ days
(THRASHING!!!)

A. Gerbessiotis

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

not Sub1! : Locality of reference

BAD PROGRAM

```
LOAD AC  100
ADD  AC  200
ADD  AC  300
ADD  AC  400
  ... and so on...
ADD  AC   9900
ADD  AC  10000
```

BETTER PROGRAM

```
LOAD  AC 100
ADD   AC 101
ADD   AC 102
ADD   AC 103
 ... and so on
ADD   AC 198
ADD   AC 199
```

Observation 1 : 100, 200, 300, ..., 10000 in 100 pages

Observation 2 : 100, 101, 102, ..., 199 in 1page ... page with page number 1!

100page faults vs 1 page fault!

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Implications

- More processes may be maintained in main memory
  - Because only some of the pieces of any particular process are loaded, there is room for more processes
  - This leads to more efficient utilization of the processor because it is more likely that at least one of the more numerous processes will be in a Ready state at any particular time
- A process may be larger than all of main memory
  - If the program being written is too large, the programmer must devise ways to structure the program into pieces that can be loaded separately in some sort of overlay strategy
  - With virtual memory based on paging or segmentation, that job is left to the OS and the hardware
  - The OS automatically loads pieces of a process into main memory as required

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Real and Virtual Memory

Real memory

- Main memory, the actual RAM

Virtual memory

- Memory on disk
- Allows for effective multiprogramming and relieves the user of tight constraints of main memory

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Simple Paging	Virtual Memory Paging	Simple Segmentation	Virtual Memory Segmentation
Main memory partitioned into small fixed-size chunks called frames		Main memory not partitioned	
Program broken into pages by the compiler or memory management system		Program segments specified by the programmer to the compiler (i.e., the decision is made by the programmer)	
Internal fragmentation within frames		No internal fragmentation	
No external fragmentation		External fragmentation	
Operating system must maintain a page table for each process showing which frame each page occupies		Operating system must maintain a segment table for each process showing the load address and length of each segment	
Operating system must maintain a free frame list		Operating system must maintain a list of free holes in main memory	
Processor uses page number, offset to calculate absolute address		Processor uses segment number, offset to calculate absolute address	
All the pages of a process must be in main memory for process to run, unless overlays are used	Not all pages of a process need be in main memory frames for the process to run. Pages may be read in as needed	All the segments of a process must be in main memory for process to run, unless overlays are used	Not all segments of a process need be in main memory for the process to run. Segments may be read in as needed
	Reading a page into main memory may require writing a page out to disk		Reading a segment into main memory may require writing one or more segments out to disk

Table 8.2

Characteristics of Paging and Segmentation

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Thrashing

We want
Disk size memory (eg 10TiB) at
RAM/MainMemory speed (eg 80ns)

A state in which the system spends most of its time swapping process pieces rather than executing instructions

To avoid this, the operating system tries to guess, based on recent history, which pieces are least likely to be used in the near future

We get
RAM/Main Memory size (eg 8GiB)
at Disk speed (eg 20ms)

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Principle of Locality

- Program and data references within a process tend to cluster
- Only a few pieces of a process will be needed over a short period of time
- Therefore it is possible to make intelligent guesses about which pieces will be needed in the future
- Avoids thrashing

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Support Needed for Virtual Memory

For virtual memory to be practical and effective:

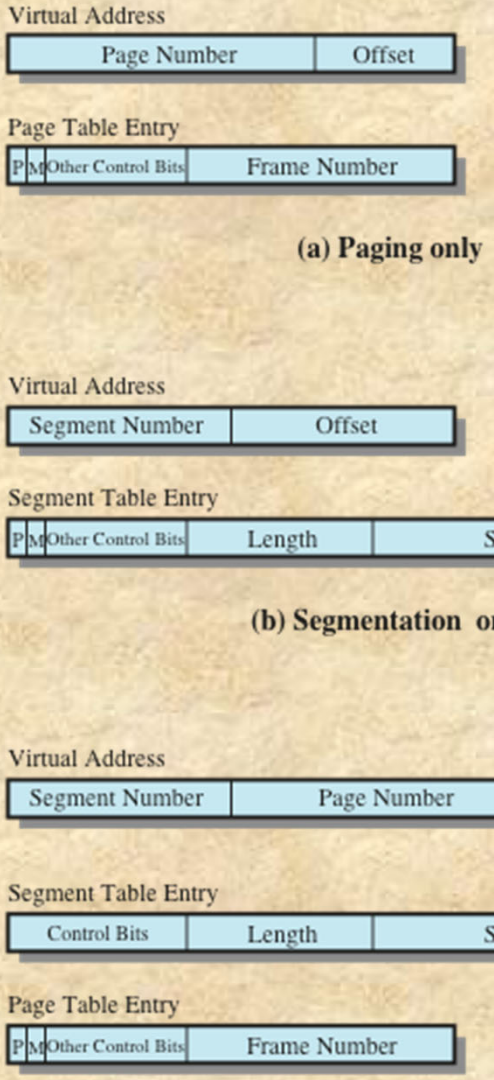
- Hardware must support paging and segmentation
- Operating system must include software for managing the movement of pages and/or segments between secondary memory and main memory

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Paging

- The term *virtual memory* is usually associated with systems that employ paging
- Use of paging to achieve virtual memory was first reported for the Atlas computer
- Each process has its own page table
  - Each page table entry (PTE) contains the frame number of the corresponding page in main memory
  - A page table is also needed for a virtual memory scheme based on paging

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.



How many bit or B for a PTE (Page Table Entry)?

Hint: A PTE must accommodate a frame number plus more. A frame number (just like a page number) is usually offset bit smaller than the bit of a physical address (32-bit, or 64-bit, or the 48-bit low bit of a 64-bit address who high 16 bit are set to 0)

Figure 8.1 Typical Memory Management Formats

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

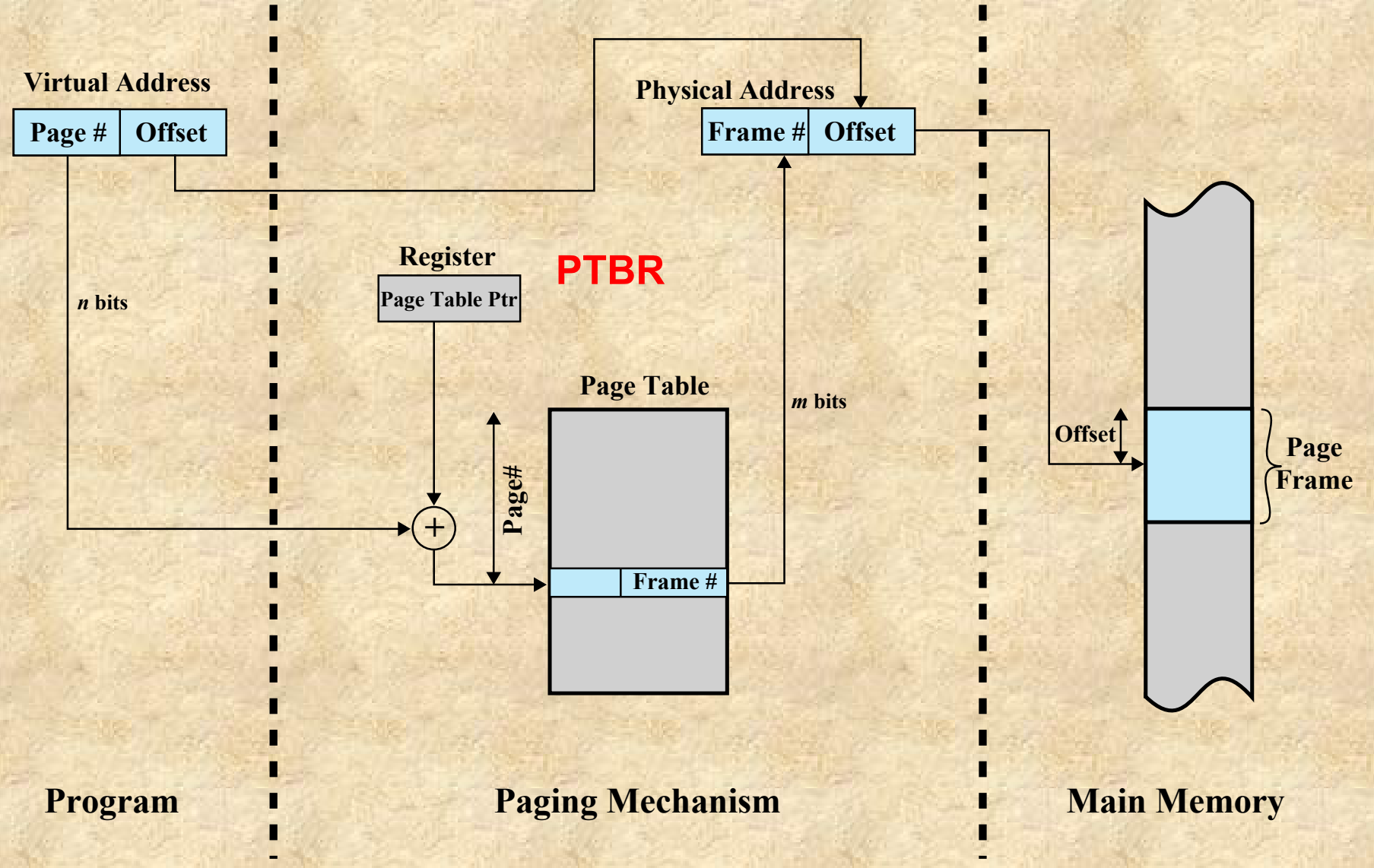


Figure 8.2 Address Translation in a Paging System

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Memory Addressing

Alex. Gerbessiotis
COMPUTER SCIENCE

1940-1960s
Simple Addressing
(Subject 1)

L=1001

A=L=1001

M[A] mem access

Time: 1x8ons=8ons

1950s-1970s
Segment Addressing
(Subject 7)

SBR = 100

L=1001

A=SBR+L=1101

M[A] mem access

Time:1x8ons+1ADD~8ons

1970s-1990s
(Simple)Paging
(Subject 7 for details)

P=S=500B

L=1001 of Process X

L=<P,T>=<P/S,P%S>=<2,1>

F=PT[P]=M[PT+P*sizeof(P)]

A=<F,T>=<4,1>

A=4*500+1=2001

M[A] mem access

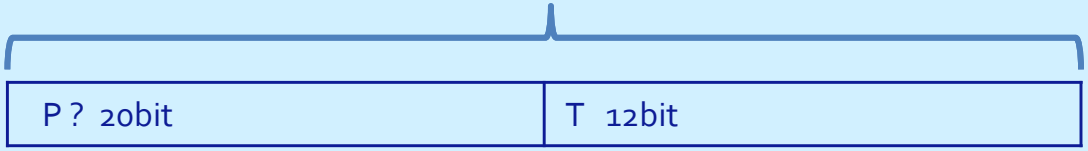
Time: 2x8ons = 16ons

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Sub7 paging: 1-level PT (array)

Alex. Gerbessiotis
COMPUTER SCIENCE

Example 1: Linux 32-bit (S=R=4KiB=4096B) SimplePagingSub7



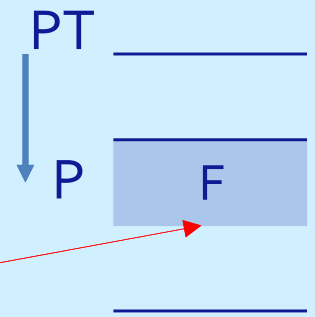
L = <P,T> L, A is 32-bit; P,F,T at most 32bit i.e. at most 4B

A = <F,T> where F=PT[P] MM<=4GiB, ProgramSize<=MM<=4GiB

1. #bitT = lg(S) =lg(4096)=12bit
2. #bitP = 32bit -#bitT= 20bit ; moreover #bitF = 32bit-12bit=20bit
3. 20bitF are to be stored in an entry of the page table; PTE = PageTableEntry ; PTE of size 4B (32bit)
   20bitF require >= 2.5B >= 3B; for bit manipulation a power of two # B implies a PTE of size 4B
4. How many P# ? 2**(#bitP) = 2**20 ; How many F# ? 2**(#bitT) = 2**20
5. A P is an index to the PT ; i.e. PT[P] ; How many indexes? The number of P# i.e. 2**20
6. Size of page table in B : LengthofPT x SizeOfPTE = 2**20 x 4B = 4MiB

Simple Paging; All in or All out of MM; ProgramSize <=MM

PT is stored in MM ; from L to M[A] we need two mem accesses.



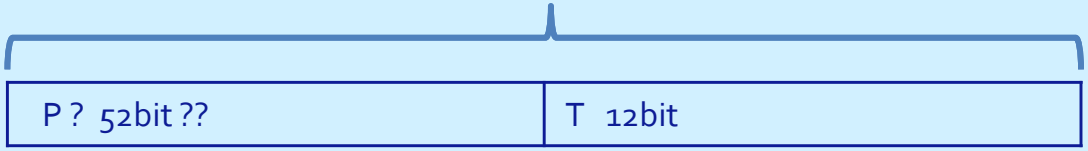
A PTE (PageTableEntry) is one 'row' or 'one element' of the PT tat stores here an F (framenumber)

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Sub7 paging: 1-level PT (array)

Alex. Gerbessiotis
COMPUTER SCIENCE

Example 2: Linux 64-bit (S=R=4KiB=4096B) Simple Paging



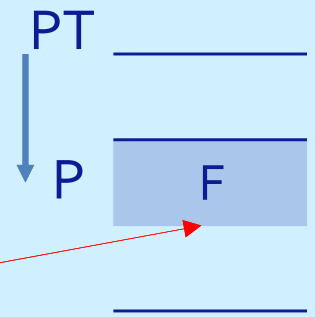
L = <P,T> L, A is 64-bit; P,F,T at most 52bit i.e. at most 8B but...

A = <F,T> where F=PT[P] MM=64GiB, ProgramSize<=MM

0. ProgramSize <= MM =2**36B ➔ #bit for an address no more than 36bit of the 64bit (64-36=28bit of zeroes)
1. #bitT = lg(S) =lg(4096)=12bit
2. #bitP = min{ 64bit -#bitT, 36-#bitT}= 24bit ; moreover #bitF = 36bit-12bit=24bit
3. 24bitF are to be stored in an entry of the page table; PTE = PageTableEntry ; PTE of size 4B (32bit)
   24bitF require >= 3B; for bit manipulation a power of two # B implies a PTE of size 4B
4. How many P# ? 2**(#bitP) = 2**24 ; How many F# ? 2**(#bitT) = 2**24
5. A P is an index to the PT ; i.e. PT[P] ; How many indexes? The number of P# i.e. 2**24
6. Size of page table in B : LengthofPT x SizeOfPTE = 2**24 x 4B = 64MiB ; it is getting too large ...

Simple Paging; All in or All out of MM; ProgramSize <=MM

PT is stored in MM ; from L to M[A] we need two mem accesses.



A PTE (PageTableEntry) is one 'row' or 'one element' of the PT tat stores here an F (framenumber)

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Multi-level Page Tables

Idea:

Instead of having one long index P pointing into one long page table (PTBR), we split index P into smaller P1, P2 indexes each one pointing into a smaller page table (ONE page/frame long)

P : 20 bit => P1:10bit , P2: 10bit

Example

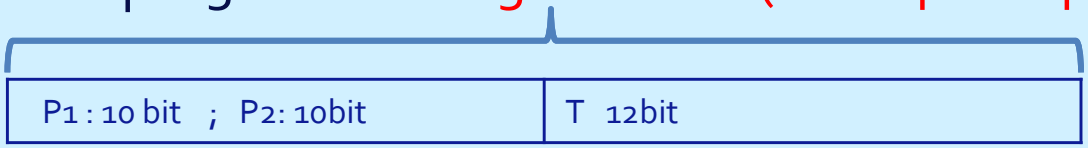
P=<11110000111100001111> => P1=<1111000011> and P2=<1100001111>

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Virtual Memory : 2-level PT

Alex. Gerbessiotis
COMPUTER SCIENCE

Example 3: Linux 32-bit (S=R=4KiB=4096B) VirtualMemorySub8



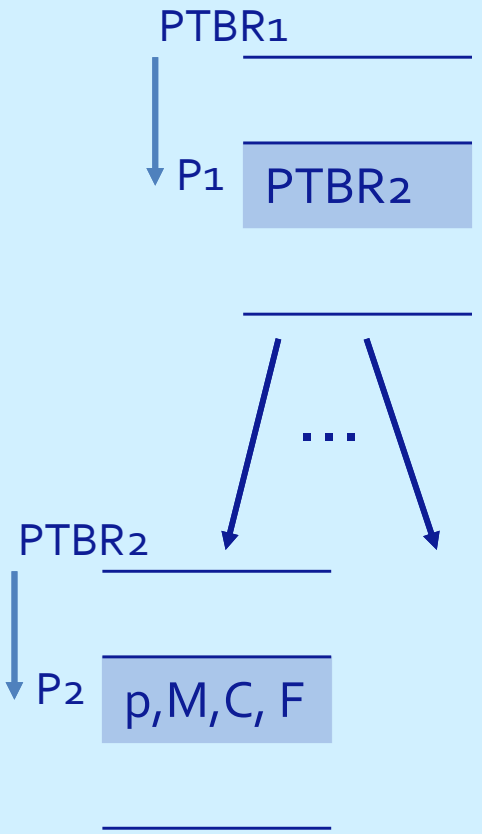
L = <P,T>=<P1 P2,T> L, A is 32-bit; F,T at most 20bit i.e. at least 3B
A = <F,T> where F=PT[P] MM<=4GiB, ProgramSize<=MM<=4GiB

Virtual Memory with Tree-like Page Table structure Page Table PTBR1

L=<P1, P2, T> #bitP1=#bitP2=#bitP/2 =20/2=10bit

1. L=<P,T> =>
1'. L=<P1 P2 T>
2. Page Table Structure stored at address : PTBR1 of MainMemory
3. (1st Mem. Access) PTBR2 = PTBR1 [P1]
4. (2nd Mem. Access) F = PTBR2[P2]
5. A = <F,T>
6. (3nd Mem. Access) M[A]
7. Time needed = 3 x MemoryAccesses= 3x80ns =240ns
8. continued next page

PTBR1 stored in MM (kernel) memory



PTBR1 and PTBR2 structures in Main Memory; PTRB1 stores 32-bit addresses(eg PTBR2)
It is PTRB2 that stores the FrameNumber F, the present bit p, the modified bit M and control bit C

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Virtual Memory : 2-level PT

Example 3 continued Linux 32-bit (S=R=4KiB=4096B) VirtualMemorySub8
Virtual Memory with Tree-like Page Table structure Page Table PTBR1

L = <P,T> L, A is 32-bit; P,F,T at most 32bit i.e. at most 4B
L=<P1, P2, T> #bitP1=#bitP2=#bitP/2 =10bit

(use also information from Example 1 as applicable)

9. How large is the root node or its child (children)? PageSize i.e. S long
10. How many entries in root node? S / PTEsize = 4KiB/4B = 1024
11. How big is an index P1 or P2 in bit? lg(Answer10)=lg(1024)=10bit
    (Answer of 11 reconciles and confirms #bitP1=bitP2=#bitP/2 shown earlier but not explained why!)
12. How many children does root node have? Answer10 says 1024!
13. What do we store in root node? Addresses ~~or Page/FrameNumbers~~ of PTBR2 child structures

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Virtual Memory : 4-level PT

Example 4: Linux 48-bit x86-64 (S=R=4KiB=4096B) VirtualMemorySub8

Virtual Memory with Tree-like Page Table structure Page Table PTBR1

L = <P,T> L, A is 48-bit; P,F 36 bit i.e. at least 5B

L=<P1, P2,P3,P4, T> P1,P2,P3,P4, is 9bit, T is 12 bit : 4x9+12=36+12=48bit

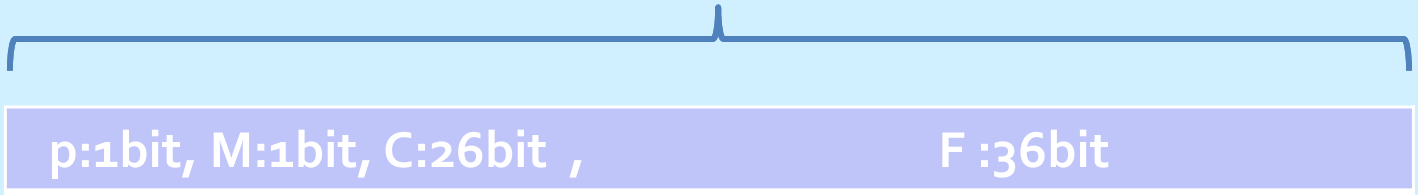
1.	L=<P,T> =>
1'.	L=<P1 P2 P3 P4 T>
2.	PageTableStructure: PTBR1
3. (1st Memory Access)	PTBR2 = PTBR1 [P1]
4. (2nd Memory Access)	PTBR3 = PTBR2 [P2]
5. (3rd Memory Access)	PTRB4 = PTBR3 [P3]
6. (4th Memory Access)	F = PTBR4[P2]
7.	A = <F,T> PTBR2
8. (5th Memory Access)	M[A]
9. Time needed =	5 x MemoryAccesses= 5x80ns =400ns
10. continued next page	

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Virtual Memory : 4-level PT

Example: Linux 48-bit x86-64 (S=R=4KiB=4096B) VirtualMemorySub8
Virtual Memory with Tree-like Page Table structure Page Table PTBR1

11. How large is a PTE (PageTableEntry i.e. row)? 8B



(just the F requires 36bit i.e. 5B and thus if PTE cannot be 4B the next power of two beyond 4 (of 4B) is 8 i.e. 8B for a PTE)

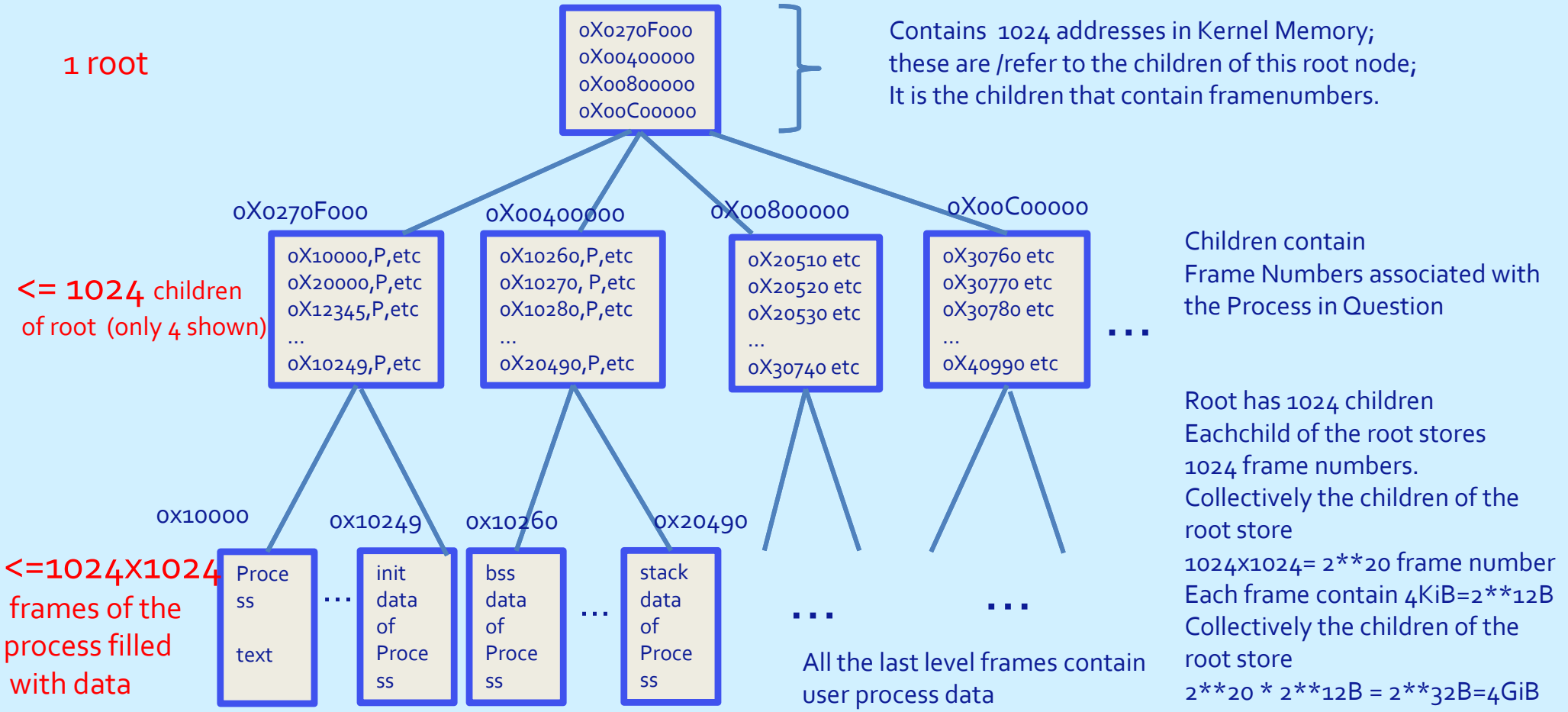
12. How large is the root node or its child (children)? PageSize S long
13. How many entries in root and children node(s)? S / PTEsize = 4KiB/8B = 512
(4096B /page) / (8B /PTE) = 512PTE / page.
14. How big is an index P1 … P4 in bit? lg(Answer13)=lg(512)=9bit
(this reconciles and explains the previous page choice of 9bit!)
15. How many children does root node have? Answer13 implies 512!
16. What do we store in root node? Addresses or ~~Page/FrameNumbers~~ of PTBRi child structures, i>1

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

2-level Page Table of Ex. 3

Alex. Gerbessiotis
COMPUTER SCIENCE

For Page Size=Frame Size = 4096B =4KiB=> #bit offset=lg(4096)=12bit



© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Example 3

Example3. PageTable of slide 36; 32-bit addresses , S=R=4KiB . You are given

L=<00000000 00000000 00100000 00001111>

Q. For the given L what is the A for the PageTable of the previous page?

1. Page size 4KiB. #bit for a page/frame offset T is lg(4096)=12bit
   L=<P,T>=<00000000 00000000 0010,000000001111>
2. Address is 32-bit; # bit for pagenumber P is 32-12=20bit
   L=<P,T>=<00000000 00000000 0010,000000001111>
3. Split P into P1 and P2 (10bit each; see Case 3 as well)
   L=<P,T>=<*0000000000*,*0000000010*,000000001111>=<P1 P2 T>
4. Use <*P1*>=<*0000000000*> as an index to the root of PT (previous page)
   Index <0000000000> stores a 0X0270F000 address.
   Go to address 0X0270F000 (left-most child of the root, level 2 of tree).
5. Use <*P2*>=<*0000000010*> as an index to that node (letmost child)
   Index <0000000010> stores a 0X12345 (a framenumber). That is F.
   Form now A=<F,T>=<00010010001101000101 000000001111>
   A=<000100100011010001010000000001111>

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

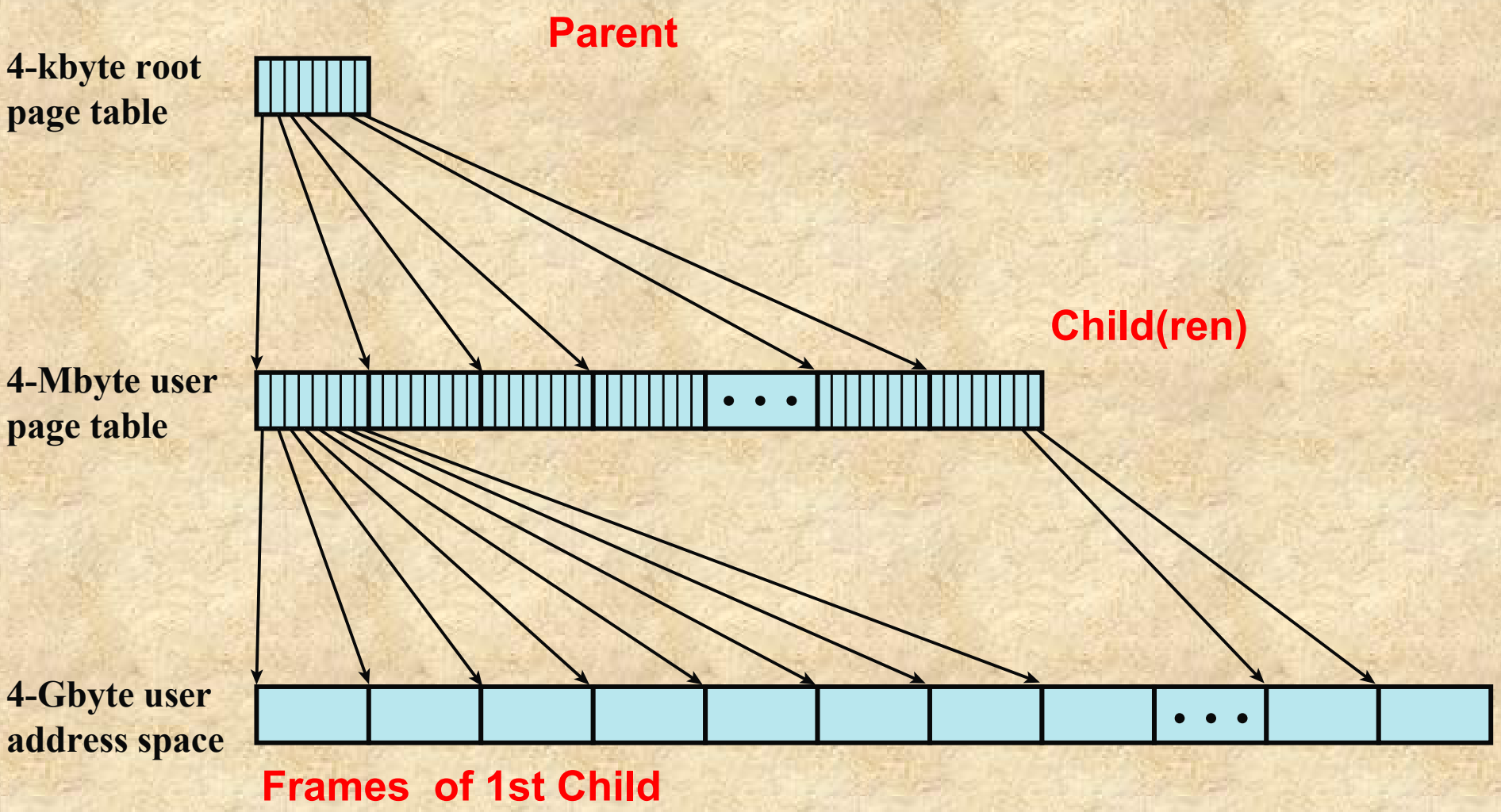


Figure 8.3 A Two-Level Hierarchical Page Table

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

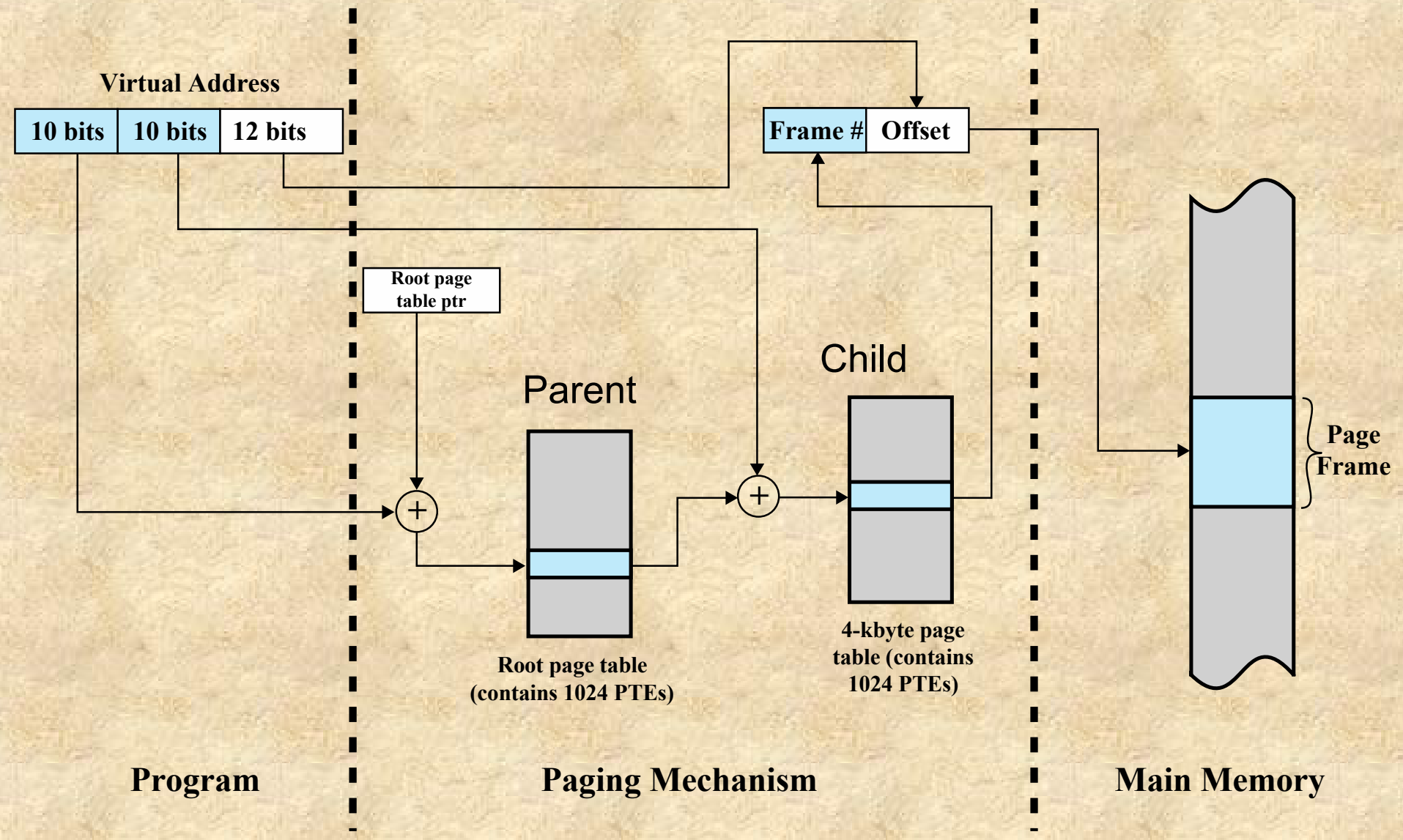


Figure 8.4 Address Translation in a Two-Level Paging System

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Example 3 duplicate slide

Example1. PageTable of slide 36; 32-bit addresses , S=R=4KiB . You are given

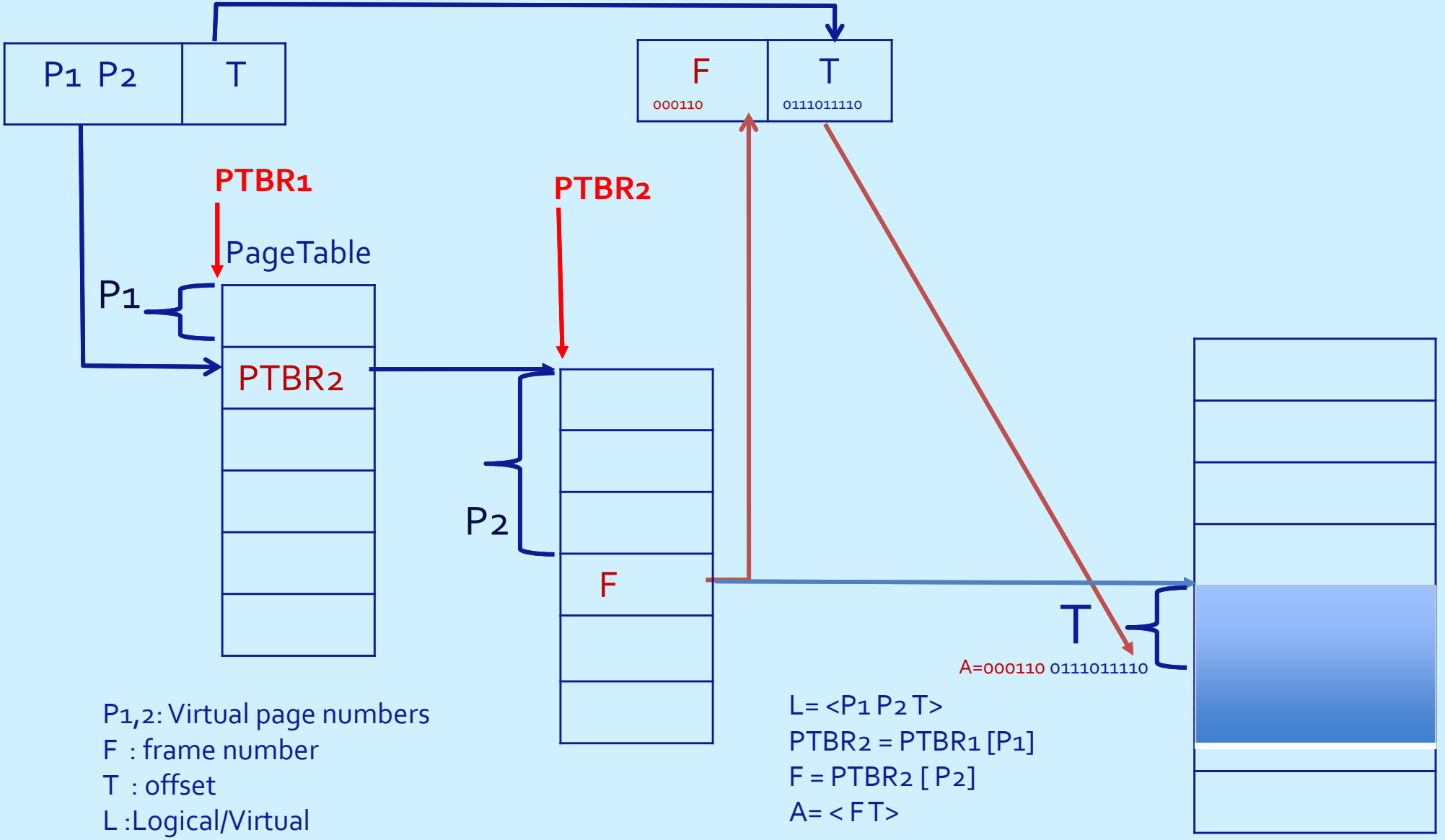
L=<00000000 00000000 00100000 00001111>

Q. For the given L what is the A for the PageTable of the previous page?

1. Page size 4KiB. #bit for a page/frame offset T is lg(4096)=12bit
   L=<P,T>=<00000000 00000000 0010,000000001111>
2. Address is 32-bit; # bit for pagenumber P is 32-12=20bit
   L=<P,T>=<00000000 00000000 0010,000000001111>
3. Split P into P1 and P2 (10bit each; see Case 3 as well)
   L=<P,T>=<*0000000000*,*0000000010*,000000001111>=<P1 P2 T>
4. Use <*P1*>=<*0000000000*> as an index to the root of PT (previous page)
   Index <0000000000> stores a 0X0270F000 address.
   Go to address 0X0270F000 (left-most child of the root, level 2 of tree).
5. Use <*P2*>=<*0000000010*> as an index to that node (letmost child)
   Index <0000000010> stores a 0X12345 (a framenumber). That is F.
   Form now A=<F,T>=<00010010001101000101 000000001111>
   A=<00010010001101000101000000001111>

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Two-levels: P1 , P2



P1,2: Virtual page numbers
F : frame number
T : offset
L :Logical/Virtual
A: Physical/Real

L= <P1 P2 T>
PTBR2 = PTBR1 [P1]
F = PTBR2 [P2]
A= < F T>

Memory access : M[A]

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Multi-level: 64-bit?

From 48bit to 64bit headaches...

P1	P2	T
42bit	10bit	12bit

X
Is Pi <= 10 bit ?

P1	P2	P3	T
32bit	10bit	10bit	12bit

X
Is Pi <= 10 bit ?

P1	P2	P3	P4	P5	P6	T
7bit	9bit	9bit	9bit	9bit	9bit	12bit

Result: Slow down by a factor of 7? ... unless ....
Can we do 6 levels instead ? How about 5 levels?

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Practice 1

1.1. Work out the 64-bit case of the previous ... slide.

1.2 Having done this. 64-bit addressing only use 48-bit for an address. The remaining 16 bit are always zero. Why ? Because we do not have computers with more than 2**48B of main memory (that is 256TiB). Workout the details for 48-bit logical address space.

1.3 Same as in 1.2 but now you know that any machine that exists only has 1TiB of main memory max. Will this change your 1.2 answers?

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Module 2

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Alternatives

Alternatives to CS 114 / CS 116 aka

Multi-level Page Tables

CS 435: Hash Table (collision resolution by …)

CPU supported inverted Page Tables

CS 332 Caches … not … Hashes…

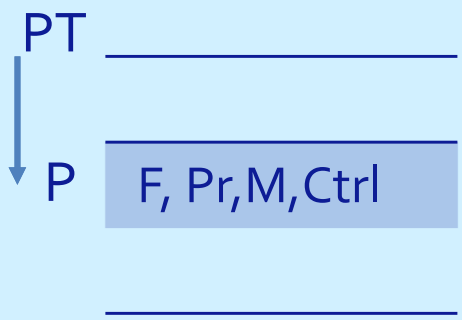
TLB

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

PT vs Inverted PT

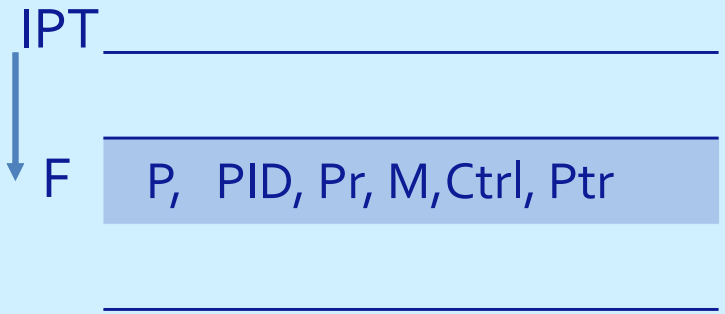
Alex. Gerbessiotis
COMPUTER SCIENCE

Page Table PT of process PID
F=PT[P]
Virtual Memory



1 page table per process

Inverted Page Table of System
P=IPT[F]
Virtual Memory



1 inverted page table per system

Present bit is denoted Pr;
P is the PageNumber
M is Modified bit
PID is PID of process
F frame Number

Multiple processes can have a Page with Page Number 0 (all of the active processes)!

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

Inverted Page Table

- Page number portion of a virtual address is mapped into a hash value
  - Hash value points to inverted page table
- Fixed proportion of real memory is required for the tables regardless of the number of processes or virtual pages supported
- Structure is called *inverted* because it indexes page table entries by frame number rather than by virtual page number

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

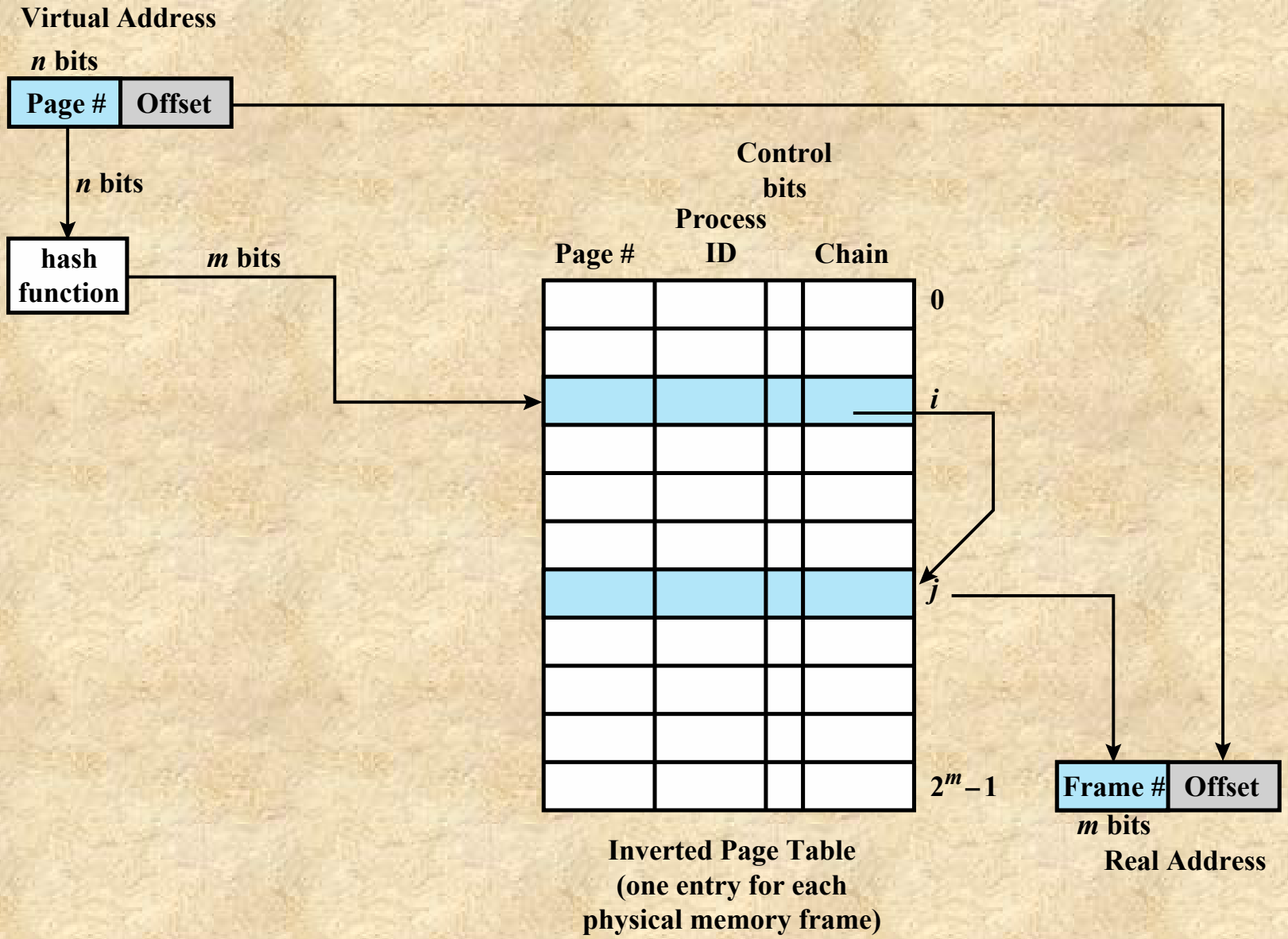


Figure 8.5 Inverted Page Table Structure

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Inverted Page Table

Each entry in the page table includes:

Page number	Process identifier	Control bits	Chain pointer
• This is the page number portion of the virtual address	• The process that owns this page	• Includes flags and protection and locking information	• The index value of the next entry in the chain

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Translation Lookaside Buffer (TLB)

- Each virtual memory reference can cause two physical memory accesses:
  - One to fetch the page table entry **ie frame number**
  - One to fetch the data
- To overcome the effect of doubling the memory access time, most virtual memory schemes make use of a special high-speed cache called a ***translation lookaside buffer*** (TLB)
  - This cache functions in the same way as a memory cache and contains those page table entities that have been most recently used

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

TLB vs Inverted PT

System TLB (part of CPU)

Virtual Memory

TLB

P, PID, Pr, M,Ctrl, F

1 TLB per CPU

Inverted Page Table of System

P=IPT[F]

Virtual Memory

IPT

F

P, PID, Pr, M,Ctrl, Ptr

1 inverted page table per system

Present bit is denoted Pr;
P is the PageNumber
M is Modified bit
PID is PID of process
F frame Number
Pr is Present bit
Ctrl is Control bit
Ptr is pointer to next entry (IPT only)
Multiple processes can have a Page with Page Number o (all of the active processes)!

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

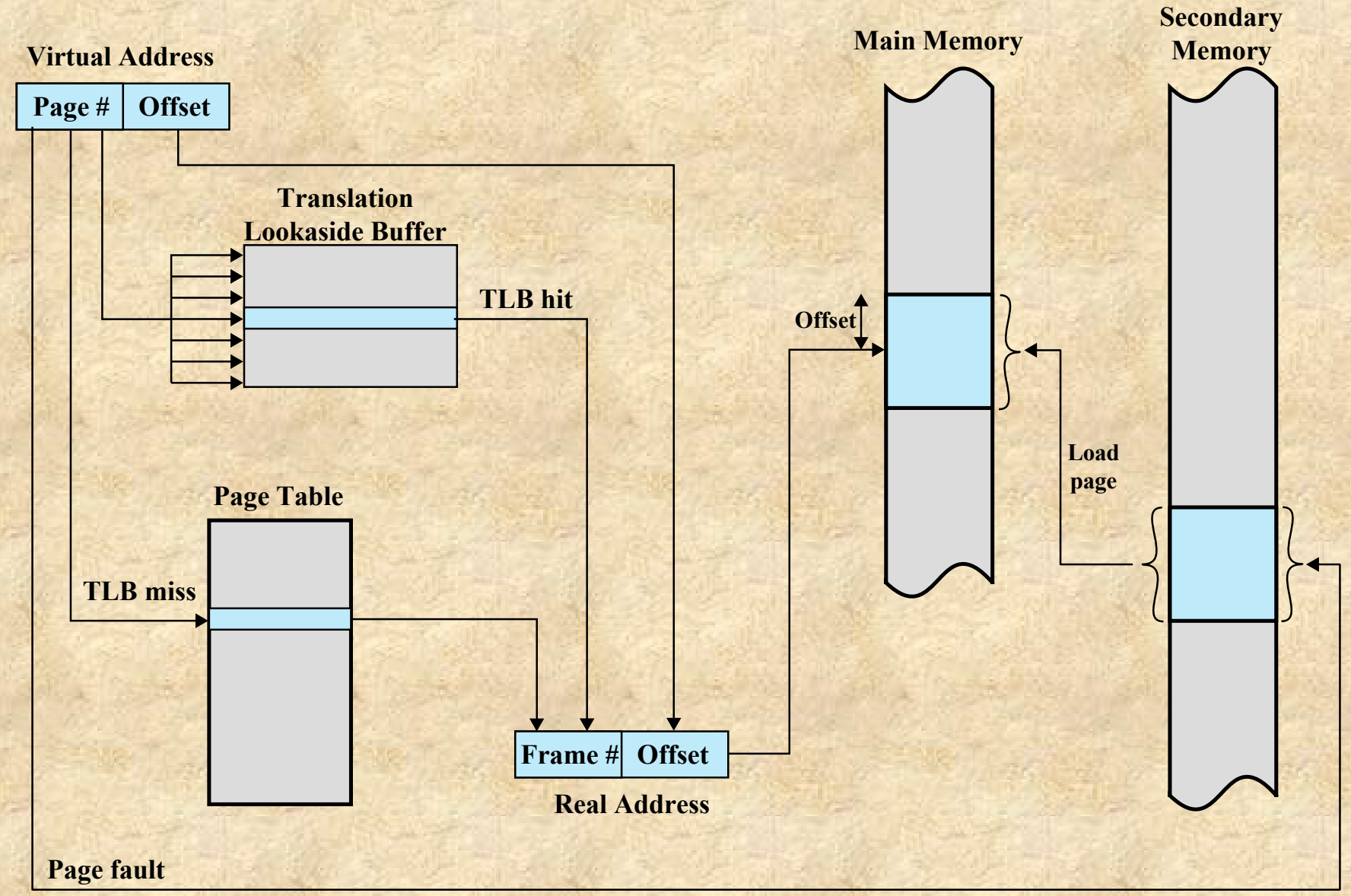


Figure 8.6 Use of a Translation Lookaside Buffer

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

More on TLBs

TLB hardware expensive and fast 64 to max of 1024 entries
TLB uses cache technology
PageTables are in (Kernel) RAM
SETUP: 1 TLB per system + 1 PT per process!

TLB hit: Lookup in TLB: 20ns and HIT! go to next step
Access Memory: 80ns Total: 100ns

TLB miss: Lookup TLB : 20ns and a MISS ; goto page TABLE
Lookup in PageT : 80+80+80=240ns
Access Memory: 80ns Total:340ns

TLB hit = 90% , TLB miss = 10%

Eff. Mem. Access Time:= 90% x 100 + 10% x 340 =
0.9 x 100 + 0.10 x 340 = 124ns
close to 100ns

© 2019-2027. Alex. Gerbessiotis. All rights reserved.

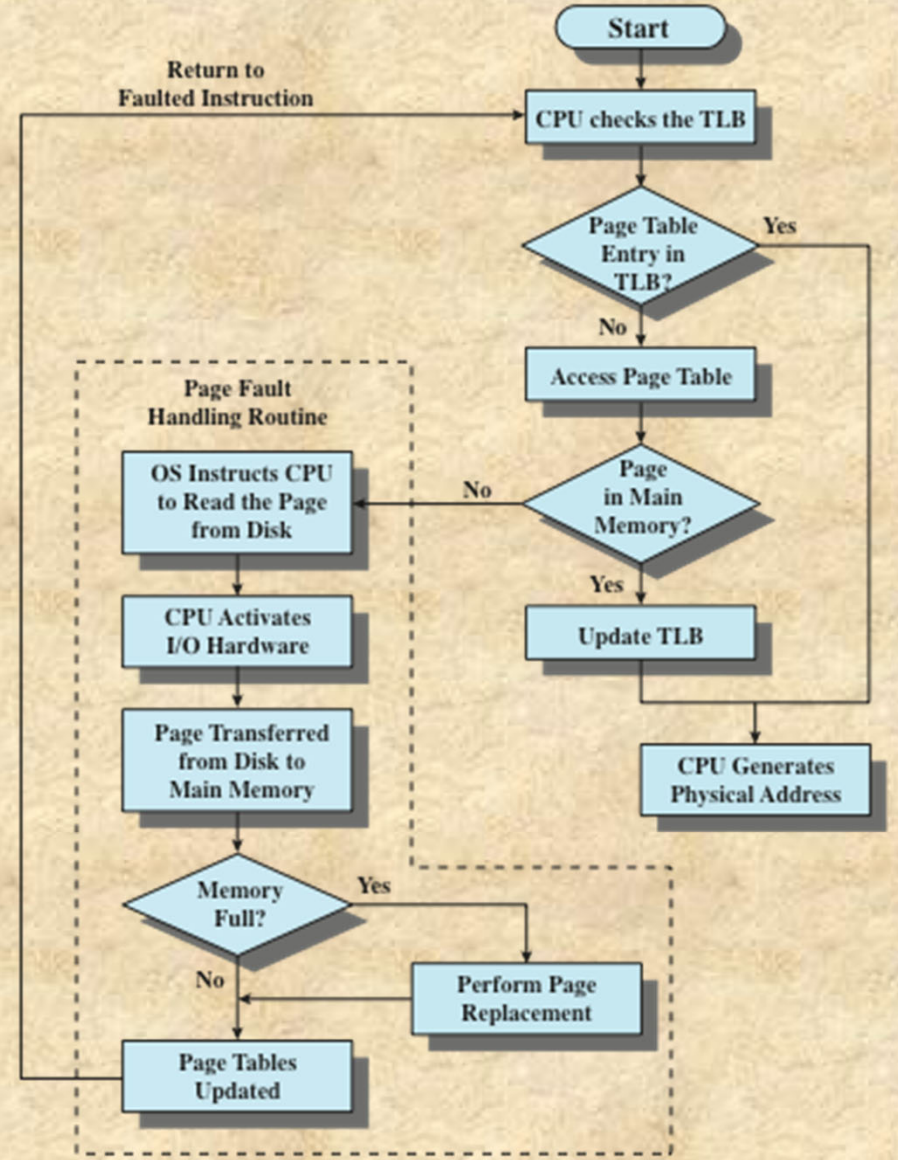


Figure 8.7 Operation of Paging and Translation Lookaside Buffer (TLB)

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Operating System Software

The design of the memory management portion of an operating system depends on three fundamental areas of choice:

- Whether or not to use virtual memory techniques
- The use of paging or segmentation or both
- The algorithms employed for various aspects of memory management

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

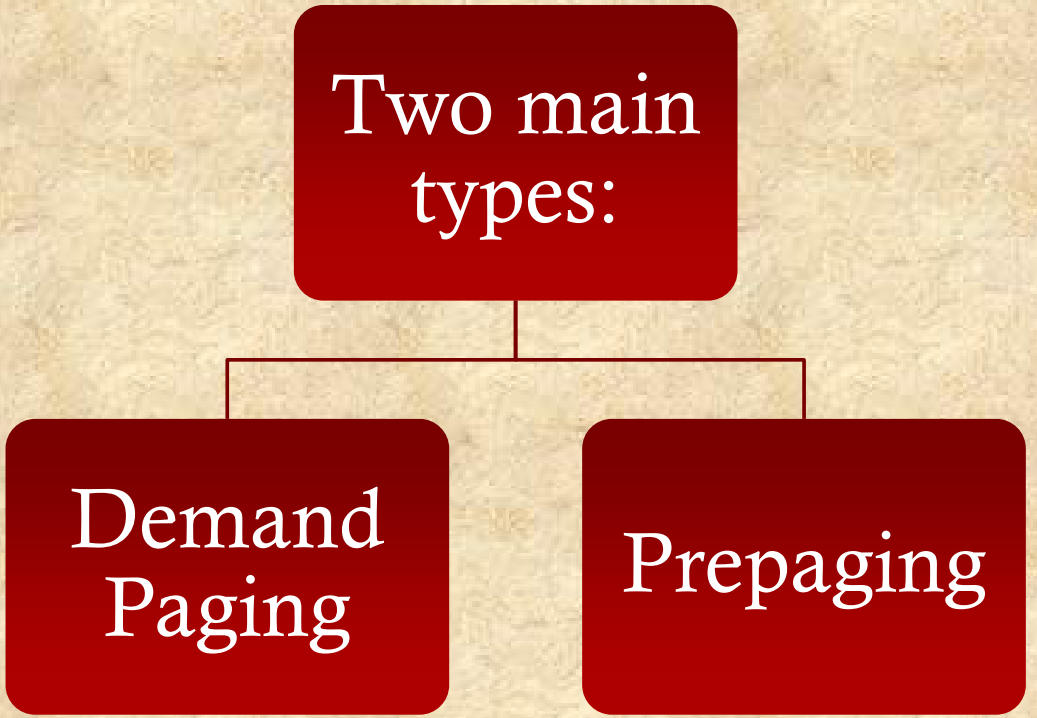
Fetch Policy Demand paging Prepaging **Placement Policy** **Replacement Policy** Basic Algorithms Optimal Least recently used (LRU) First-in-first-out (FIFO) Clock Page Buffering	**Resident Set Management** Resident set size Fixed Variable Replacement Scope Global Local **Cleaning Policy** Demand Precleaning **Load Control** Degree of multiprogramming

Table 8.4 Operating System Policies for Virtual Memory

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Fetch Policy

- Determines when a page should be brought into memory



© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Demand Paging

- **Demand Paging**
  - Only brings pages into main memory when a reference is made to a location on the page
  - Many page faults when process is first started
  - Principle of locality suggests that as more and more pages are brought in, most future references will be to pages that have recently been brought in, and page faults should drop to a very low level

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Prepaging

- **Prepaging**
  - Pages other than the one demanded by a page fault are brought in
  - Exploits the characteristics of most secondary memory devices
  - If pages of a process are stored contiguously in secondary memory it is more efficient to bring in a number of pages at one time
  - Ineffective if extra pages are not referenced
  - Should not be confused with "swapping"

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Placement Policy

- Determines where in real memory a process piece is to reside
- Important design issue in a segmentation system
- Paging or combined paging with segmentation placing is irrelevant because hardware performs functions with equal efficiency
- For NUMA systems an automatic placement strategy is desirable

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Replacement Policy

- Deals with the selection of a page in main memory to be replaced when a new page must be brought in
  - Objective is that the page that is removed be the page least likely to be referenced in the near future
- The more elaborate the replacement policy the greater the hardware and software overhead to implement it

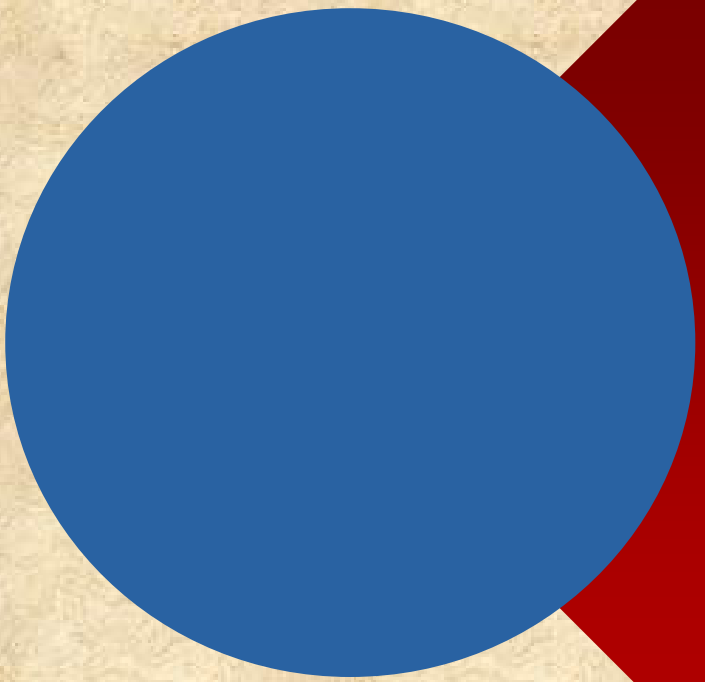
© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Frame Locking

- When a frame is locked the page currently stored in that frame may not be replaced
  - Kernel of the OS as well as key control structures are held in locked frames
  - I/O buffers and time-critical areas may be locked into main memory frames
  - Locking is achieved by associating a lock bit with each frame

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

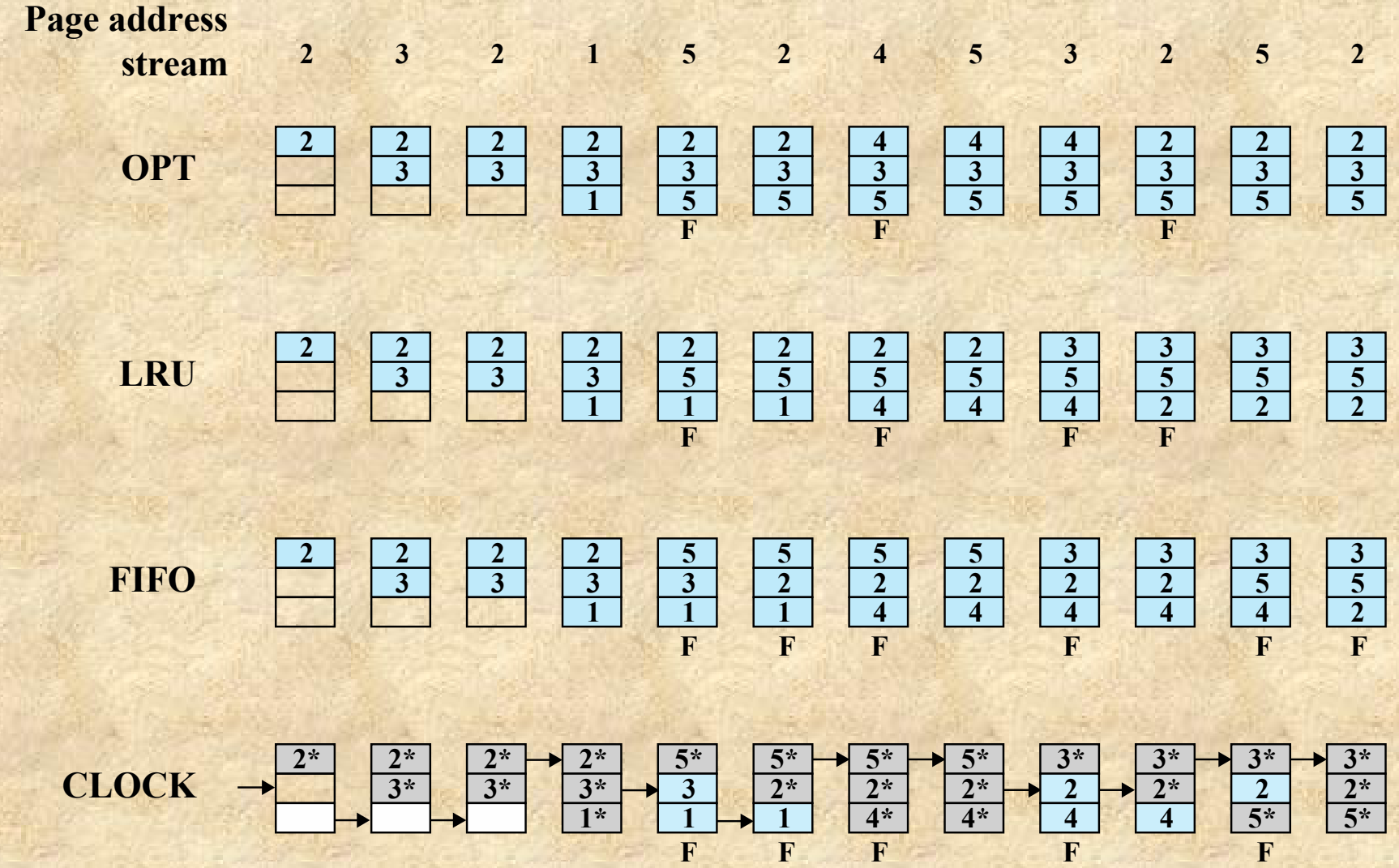
Basic Algorithms



Algorithms used for the selection of a page to replace:

- Optimal
- Least recently used (LRU)
- First-in-first-out (FIFO)
- Clock

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.



F = page fault occurring after the frame allocation is initially filled

Figure 8.14 Behavior of Four Page-Replacement Algorithms

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Least Recently Used (LRU)

- Replaces the page that has not been referenced for the longest time
- By the principle of locality, this should be the page least likely to be referenced in the near future
- Difficult to implement
  - One approach is to tag each page with the time of last reference
    - This requires a great deal of overhead

Another one : LFU : Least Frequently Used ...

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

First-in-First-out (FIFO)

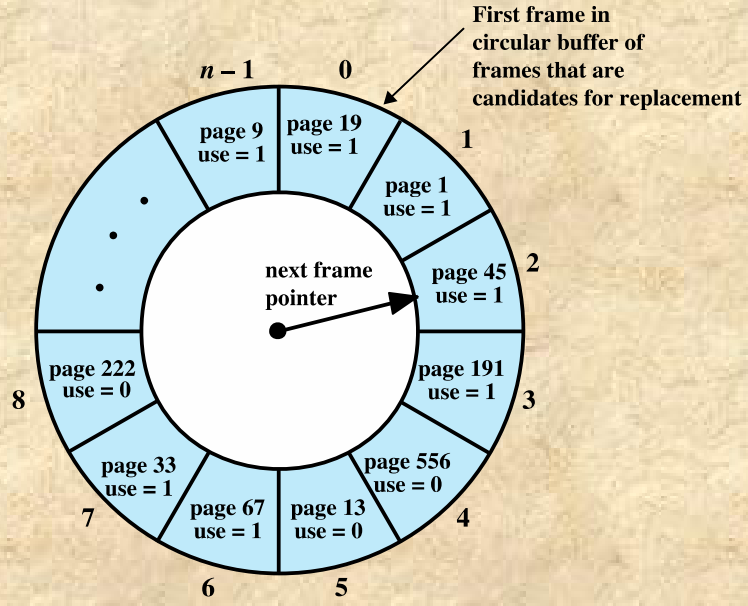
- Treats page frames allocated to a process as a circular buffer
- Pages are removed in round-robin style
  - Simple replacement policy to implement
- Page that has been in memory the longest is replaced

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

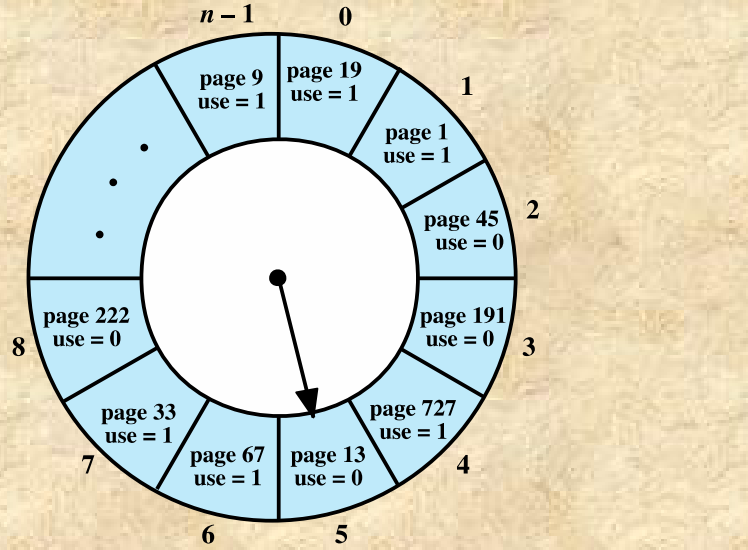
Clock Policy

- Requires the association of an additional bit with each frame
  - Referred to as the *use* bit
- When a page is first loaded in memory or referenced, the use bit is set to 1
- The set of frames is considered to be a circular buffer
- Any frame with a use bit of 1 is passed over by the algorithm
- Page frames visualized as laid out in a circle

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.



(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

Figure 8.15 Example of Clock Policy Operation

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

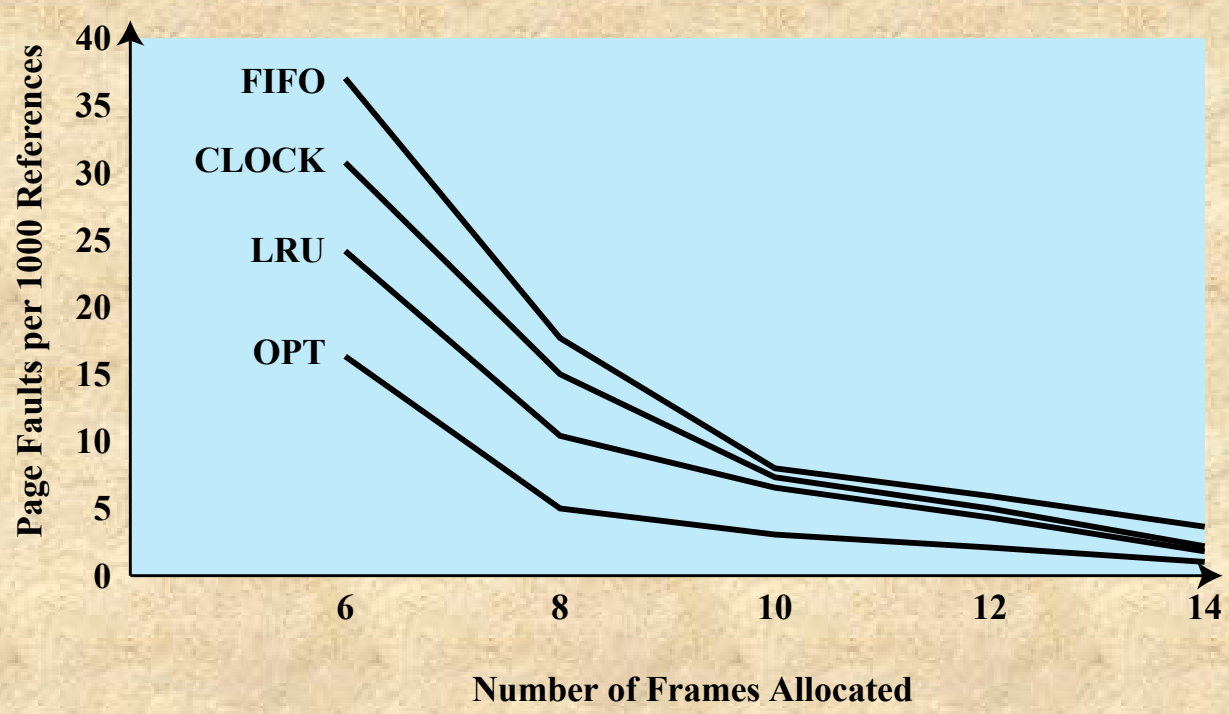
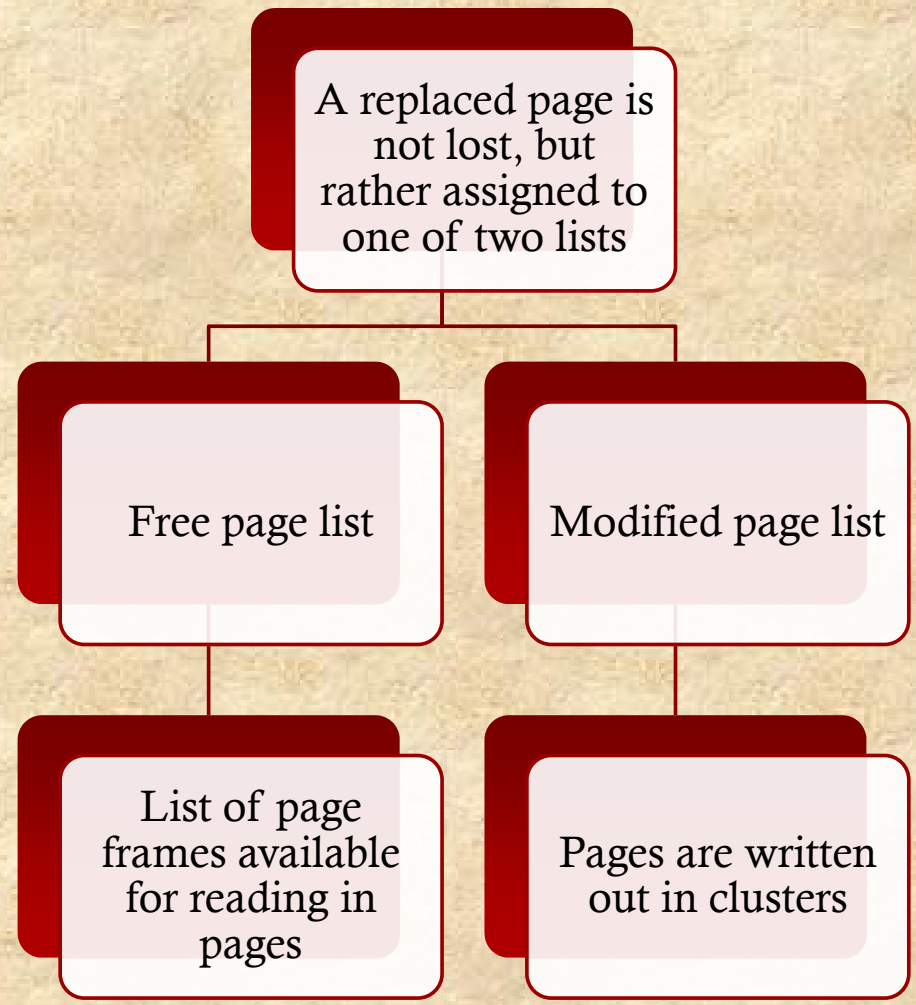


Figure 8.16 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Page Buffering

- Improves paging performance and allows the use of a simpler page replacement policy



© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Resident Set Management

- The OS must decide how many pages to bring into main memory
  - The smaller the amount of memory allocated to each process, the more processes can reside in memory
  - Small number of pages loaded increases page faults
  - Beyond a certain size, further allocations of pages will not effect the page fault rate

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Resident Set Size

Fixed-allocation

- Gives a process a fixed number of frames in main memory within which to execute
  - When a page fault occurs, one of the pages of that process must be replaced

Variable-allocation

- Allows the number of page frames allocated to a process to be varied over the lifetime of the process

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Replacement Scope

- The scope of a replacement strategy can be categorized as ***global*** or ***local***
  - Both types are activated by a page fault when there are no free page frames

Local

- Chooses only among the resident pages of the process that generated the page fault

Global

- Considers all unlocked pages in main memory

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

	Local Replacement	Global Replacement
Fixed Allocation	•Number of frames allocated to a process is fixed. •Page to be replaced is chosen from among the frames allocated to that process.	•Not possible.
Variable Allocation	•The number of frames allocated to a process may be changed from time to time to maintain the working set of the process. •Page to be replaced is chosen from among the frames allocated to that process.	•Page to be replaced is chosen from all available frames in main memory; this causes the size of the resident set of processes to vary.

Table 8.5 Resident Set Management

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Fixed Allocation, Local Scope

- Necessary to decide ahead of time the amount of allocation to give a process
- If allocation is too small, there will be a high page fault rate

If allocation is too large, there will be too few programs in main memory

- Increased processor idle time
- Increased time spent in swapping

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Variable Allocation Global Scope

- Easiest to implement
  - Adopted in a number of operating systems
- OS maintains a list of free frames
- Free frame is added to resident set of process when a page fault occurs
- If no frames are available the OS must choose a page currently in memory
- One way to counter potential problems is to use page buffering

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Variable Allocation Local Scope

- When a new process is loaded into main memory, allocate to it a certain number of page frames as its resident set
- When a page fault occurs, select the page to replace from among the resident set of the process that suffers the fault
- Reevaluate the allocation provided to the process and increase or decrease it to improve overall performance

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Variable Allocation Local Scope

- Decision to increase or decrease a resident set size is based on the assessment of the likely future demands of active processes

Key elements:

- Criteria used to determine resident set size
- The timing of changes

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Page Fault Frequency (PFF)

- Requires a use bit to be associated with each page in memory
- Bit is set to 1 when that page is accessed
- When a page fault occurs, the OS notes the virtual time since the last page fault for that process
- Does not perform well during the transient periods when there is a shift to a new locality

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Variable-Interval Sampled Working Set (VSWS)

- Evaluates the working set of a process at sampling instances based on elapsed virtual time
- Driven by three parameters:

The minimum duration of the sampling interval	The maximum duration of the sampling interval	The number of page faults that are allowed to occur between sampling instances

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Cleaning Policy

- Concerned with determining when a modified page should be written out to secondary memory

Demand Cleaning

A page is written out to secondary memory only when it has been selected for replacement

Precleaning

Allows the writing of pages in batches

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Load Control

- Determines the number of processes that will be resident in main memory
  - *Multiprogramming* level
- Critical in effective memory management
- Too few processes, many occasions when all processes will be blocked and much time will be spent in swapping
- Too many processes will lead to thrashing

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Process Suspension

- If the degree of multiprogramming is to be reduced, one or more of the currently resident processes must be swapped out

Six possibilities exist:

- Lowest-priority process
- Faulting process
- Last process activated
- Process with the smallest resident set
- Largest process
- Process with the largest remaining execution window

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

UNIX

- Intended to be machine independent so its memory management schemes will vary
  - Early UNIX: variable partitioning with no virtual memory scheme
  - [illegible] make use of paged

SVR4 and Solaris use two separate schemes:

- Paging system
- Kernel memory allocator

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Paging System and Kernel Memory Allocator

Paging System

Provides a virtual memory capability that allocates page frames in main memory to processes

Allocates page frames to disk block buffers

Kernel Memory Allocator

Allocates memory for the kernel

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

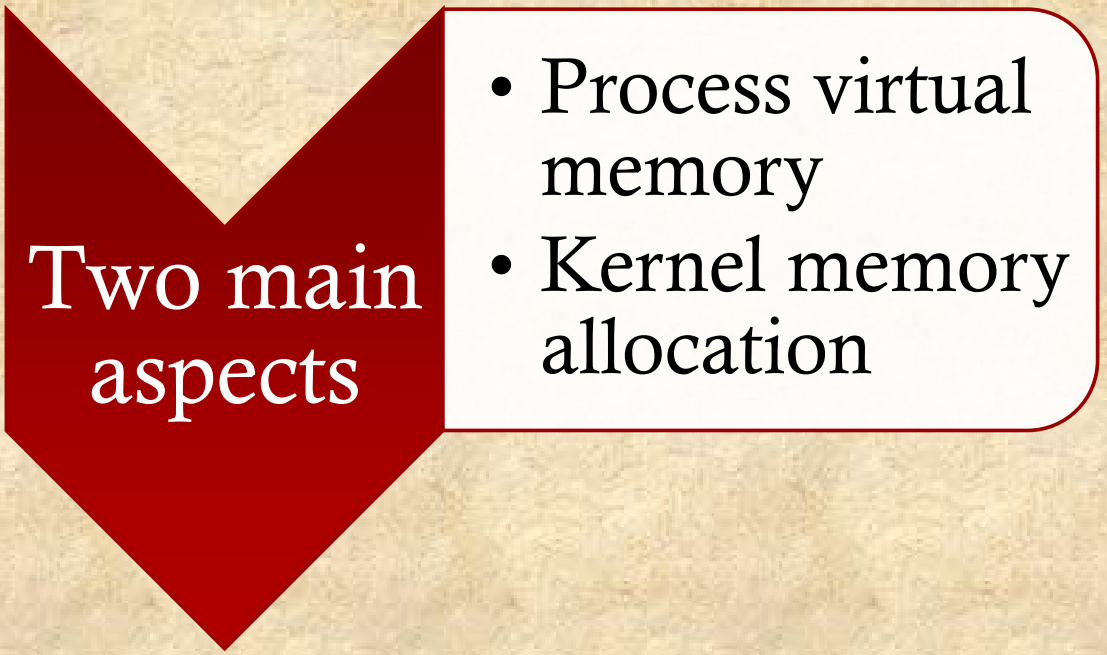
Kernel Memory Allocator

- The kernel generates and destroys small tables and buffers frequently during the course of execution, each of which requires dynamic memory allocation.
- Most of these blocks are significantly smaller than typical pages (therefore paging would be inefficient)
- Allocations and free operations must be made as fast as possible

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Linux Memory Management

- Shares many characteristics with UNIX
- Is quite complex



© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Linux Virtual Memory

- Three level page table structure:

Page directory	Page middle directory	Page table
Process has a single page directory	May span multiple pages	May also span multiple pages
Each entry points to one page of the page middle directory	Each entry points to one page in the page table	Each entry refers to one virtual page of the process
Must be in main memory for an active process		

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

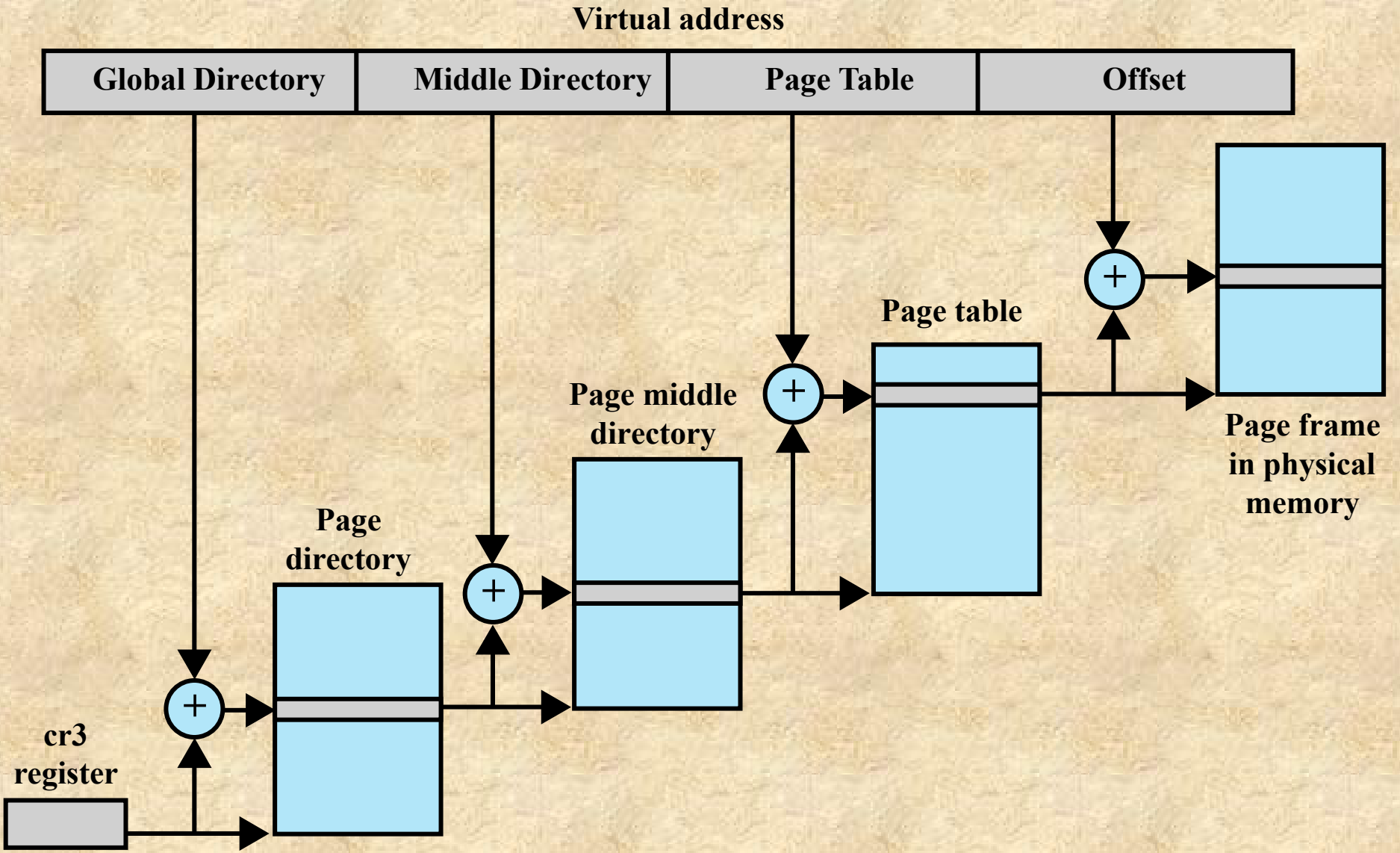


Figure 8.23 Address Translation in Linux Virtual Memory Scheme

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Linux Page Replacement

- Based on the clock algorithm
- The use bit is replaced with an 8-bit age variable
  - Incremented each time the page is accessed
- Periodically decrements the age bits
  - A page with an age of 0 is an "old" page that has not been referenced in some time and is the best candidate for replacement
- A form of least frequently used policy

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Windows Memory Management

- Virtual memory manager controls how memory is allocated and how paging is performed
- Designed to operate over a variety of platforms
- Uses page sizes ranging from 4 Kbytes to 64 Kbytes

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

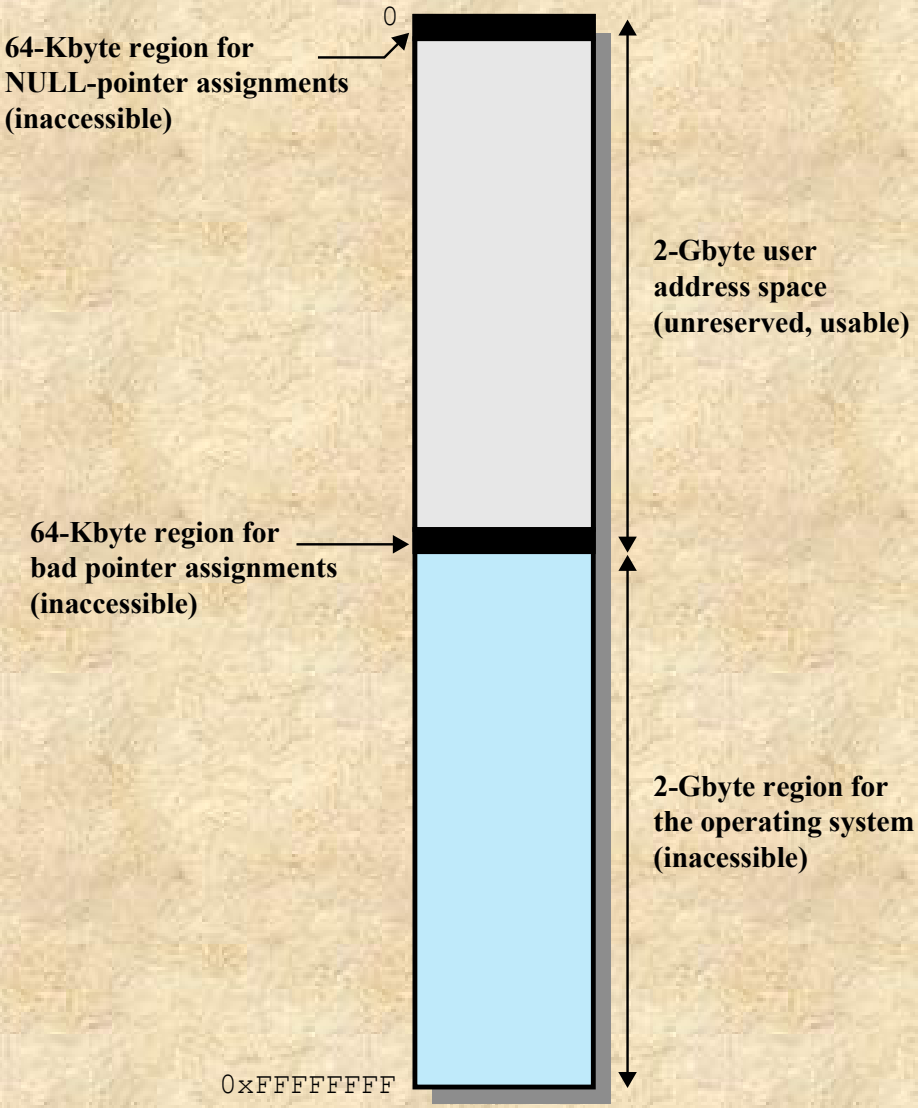


Figure 8.25 Windows Default 32-bit Virtual Address Space

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Android Memory Management

- Android includes a number of extensions to the normal Linux kernel memory management facility
- These include:
  - ASHMem
    - This feature provides anonymous shared memory, which abstracts memory as file descriptors
    - A file descriptor can be passed to another process to share memory
  - Pmem
    - This feature allocates virtual memory so that it is physically contiguous
    - This feature is useful for hardware that does not support virtual memory
  - Low Memory Killer
    - This feature enables the system to notify an app or apps that they need to free up memory
    - If an app does not cooperate, it is terminated

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Practice 2

2.1 Solve the puzzle with the MOV instruction (next page) if you are interested (read Intel manuals!)

2.2. Practice with page replacement policies.

2.3 With reference to the Linux Figure 8.23 (slide 86) how many levels does the page table structure have and how many Memory Accesses do we need to perform to resolve an M[A] starting with L? (Eccentric Hint: one extra level indicated is to accommodate Dec Alpha CPUs and is not used for Intel CPUs.)

How many pages per process?

Find most complicated instruction (working set size):

MOV [source-addr] [destination-addr]

#B for MOV :>1 [source-addr, dest-addr] i.e 2 pages

(first byte of MOV is last byte of a page; remaining bytes are the first few bytes of second page]

#B for source : 32-bit i.e. 4B 2 pages

#B for dest: 32-bit i.e. 4B 2 pages

Puzzle: Indirect Addressing

MOV [[src-iaddr]] [[dest-iaddr]]

i.e. Memory location [src-iaddr] contains src-addr

i.e. Memory location [dest-iaddr] contains dest-addr

New Jersey's Science &
Technology University

THE EDGE IN KNOWLEDGE

Alex. Gerbessiotis

CS Department