



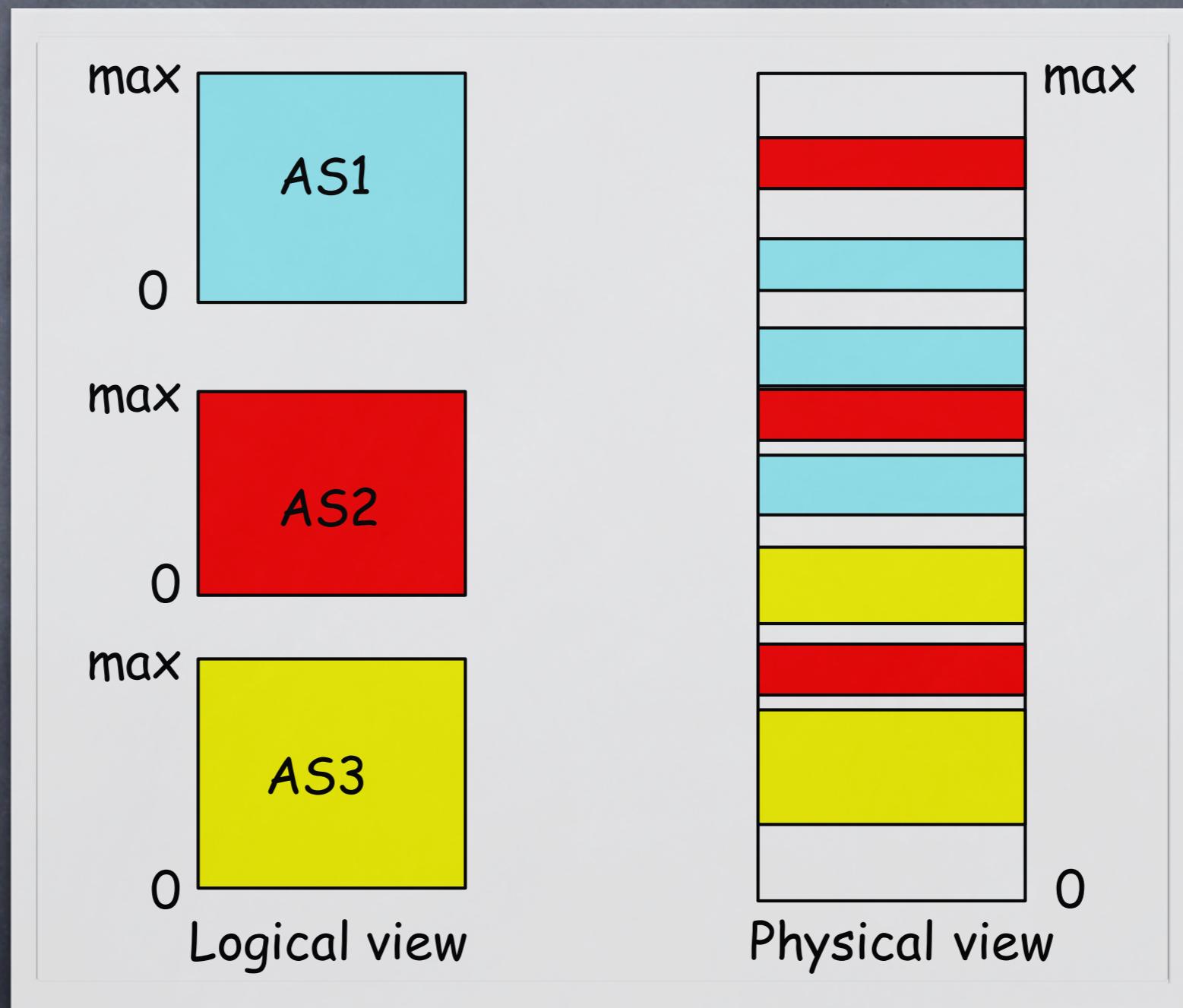
MAC422 Sistemas Operacionais

Prof. Marcel P. Jackowski

Aula #7

Gerenciamento de memória: segmentação e paginação

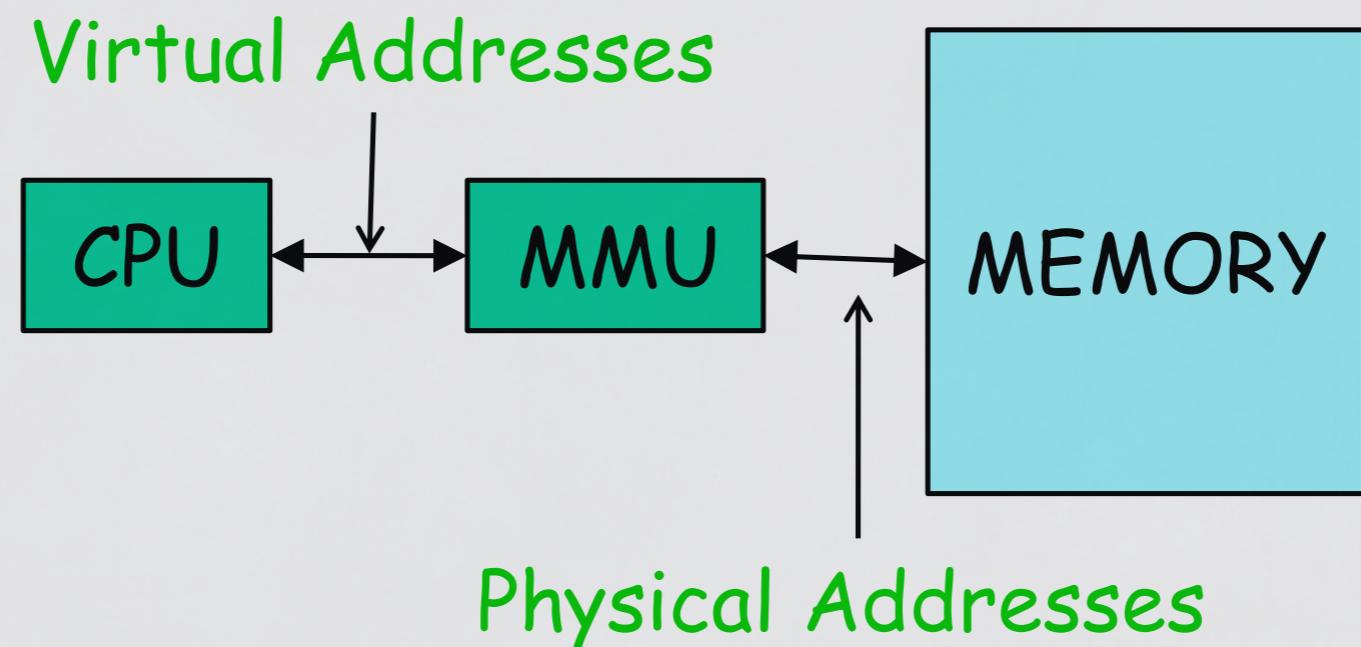
Coexistência de múltiplos espaços de endereçamento



Gerenciamento de memória

- A CPU não realiza nenhuma tarefa de gerenciamento de memória, exceto
 - Mapeamento de endereços lógicos em endereços físicos através utilizando a MMU
- Consequentemente, o SO então deve controlar:
 - Alocação e desalocação de memória
 - Compartilhamento
 - Proteção
 - “Swapping”

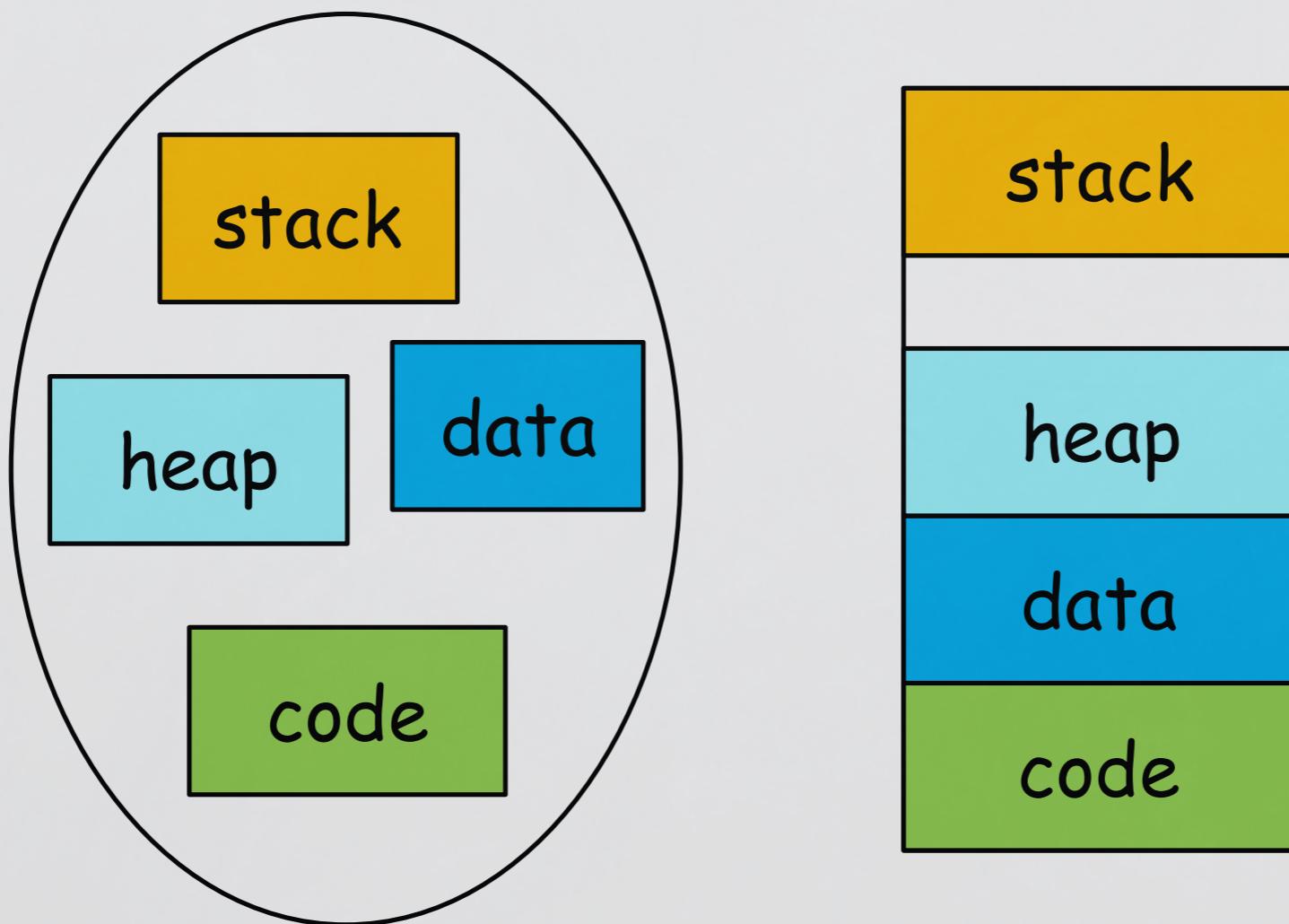
MMU



- Map program-generated address (**virtual address**) to hardware address (**physical address**) dynamically at every reference
- Check range and permissions
- Programmed by OS

Segmentação

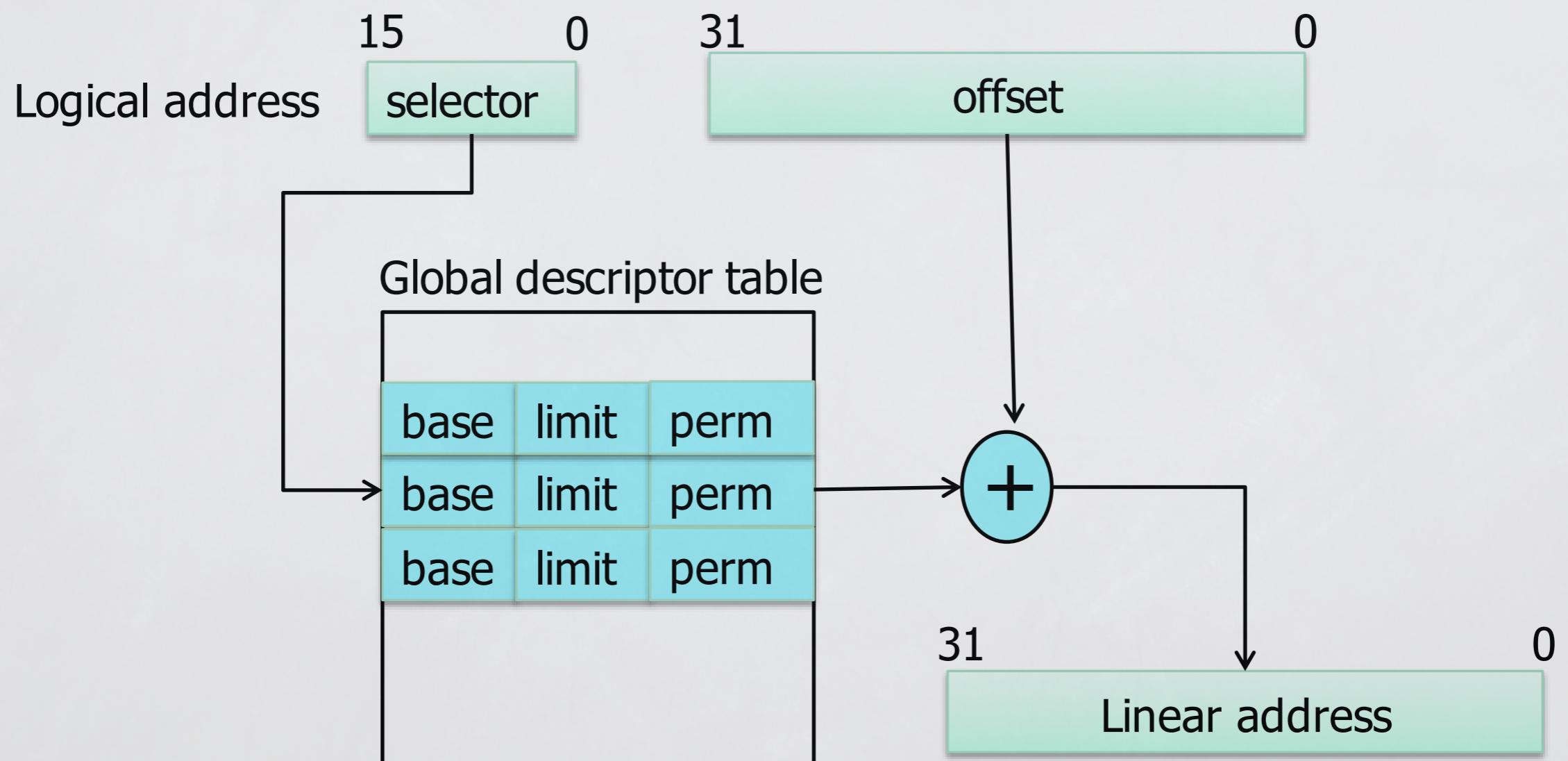
- Divide virtual address space into separate logical segments; each is part of physical mem



Tradução de segmentos

- Virtual address: <segment-number, offset>
- Segment table maps segment number to segment information
 - Base: starting address of the segment in physical memory
 - Limit: length of the segment
 - Addition metadata includes protection bits
- Limit & protection checked on each access

Hardware de segmentação



Vantagens e desvantagens

□ Advantages

- Segment sharing
- Easier to relocate segment than entire program
- Avoids allocating unused memory
- Flexible protection
- Efficient translation
 - Segment table small → fit in MMU

□ Disadvantages

- Segments have variable lengths → dynamic allocation (best fit? first fit?)
- External fragmentation: wasted memory

Partições de que tamanho ?

- Partições de tamanho fixo ou variável são ineficientes em relação ao uso da memória
- Compactação disperdiça recursos do sistema
- Solução ?
 - Dividir a RAM em pequenos pedaços
 - Não precisam ser contíguos
 - Elimina fragmentação externa que ocorre na técnica de segmentação de memória.

Paginação

- Memória física é dividida em pequenos blocos chamados de “frames”
 - Normalmente 4 Kb (2^{12}) (ou 4 Mb)
 - Tamanho é ditado pelo hardware
- Espaço lógico é também dividido em blocos chamados de “páginas”
 - Na maioria das vezes do mesmo tamanho que os “frames”
- Endereços lógicos são mapeados em endereços físicos pela MMU
 - Tabela de páginas

Frames e páginas

Frame number	Main memory
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

(a) Fifteen available frames

Main memory
A.0
A.1
A.2
A.3

(b) Load process A

Main memory
A.0
A.1
A.2
A.3
B.0
B.1
B.2

(c) Load process B

Frames e páginas

Main memory	
0	A.0
1	A.1
2	A.2
3	A.3
4	B.0
5	B.1
6	B.2
7	C.0
8	C.1
9	C.2
10	C.3
11	
12	
13	
14	

(d) Load process C

Main memory	
0	A.0
1	A.1
2	A.2
3	A.3
4	
5	
6	
7	C.0
8	C.1
9	C.2
10	C.3
11	
12	
13	
14	

(e) Swap out B

Main memory	
0	A.0
1	A.1
2	A.2
3	A.3
4	D.0
5	D.1
6	D.2
7	C.0
8	C.1
9	C.2
10	C.3
11	D.3
12	D.4
13	
14	

(f) Load process D

0	0
1	1
2	2
3	3

Process A
page table

0	—
1	—
2	—

Process B
page table

0	7
1	8
2	9
3	10

Process C
page table

0	4
1	5
2	6
3	11
4	12

Process D
page table

13
14

Free frame
list

Main memory	
0	A.0
1	A.1
2	A.2
3	A.3
4	B.0
5	B.1
6	B.2
7	C.0
8	C.1
9	C.2
10	C.3
11	
12	
13	
14	

(d) Load process C

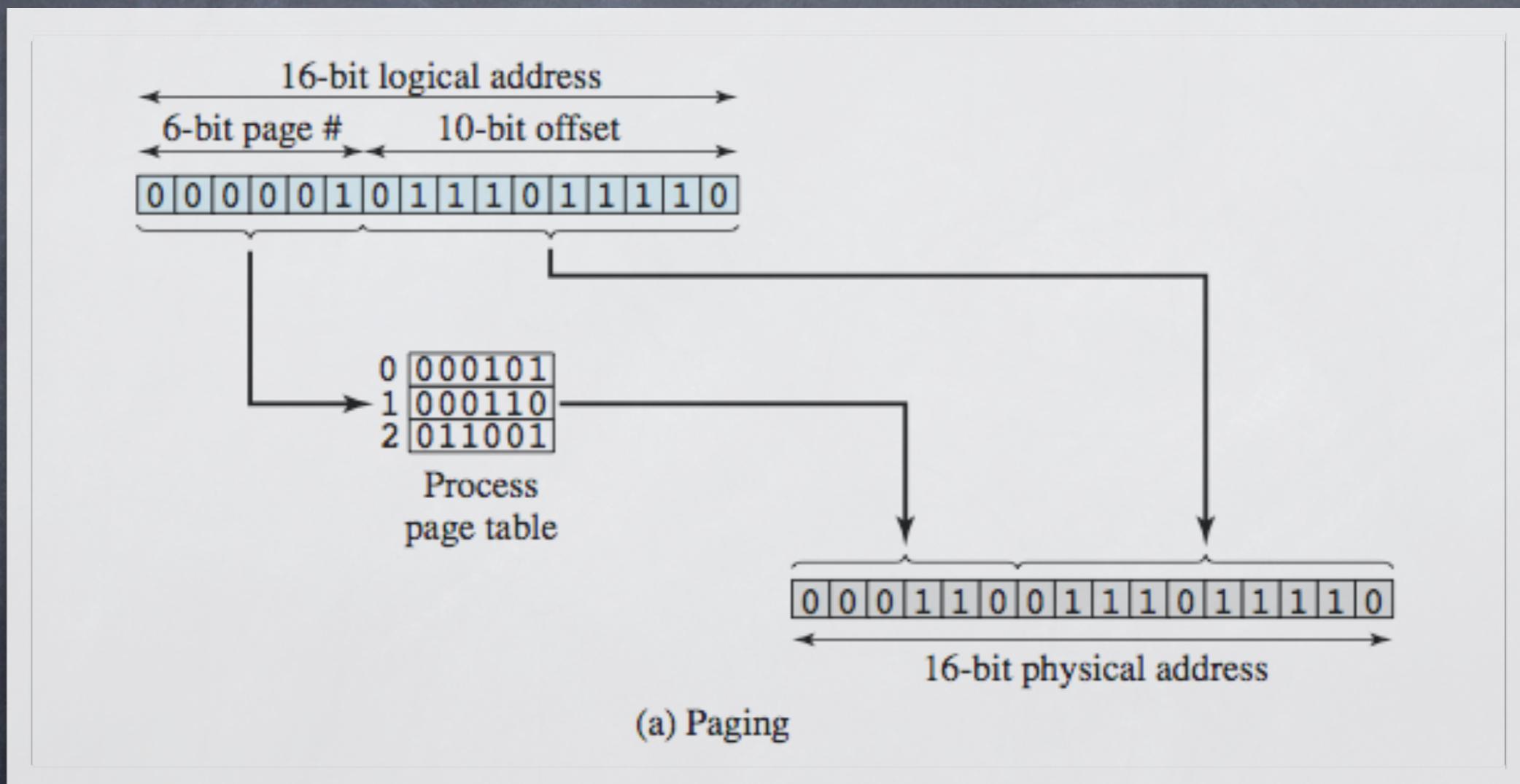
Main memory	
0	A.0
1	A.1
2	A.2
3	A.3
4	
5	
6	
7	C.0
8	C.1
9	C.2
10	C.3
11	
12	
13	
14	

(e) Swap out B

Main memory	
0	A.0
1	A.1
2	A.2
3	A.3
4	D.0
5	D.1
6	D.2
7	C.0
8	C.1
9	C.2
10	C.3
11	D.3
12	D.4
13	
14	

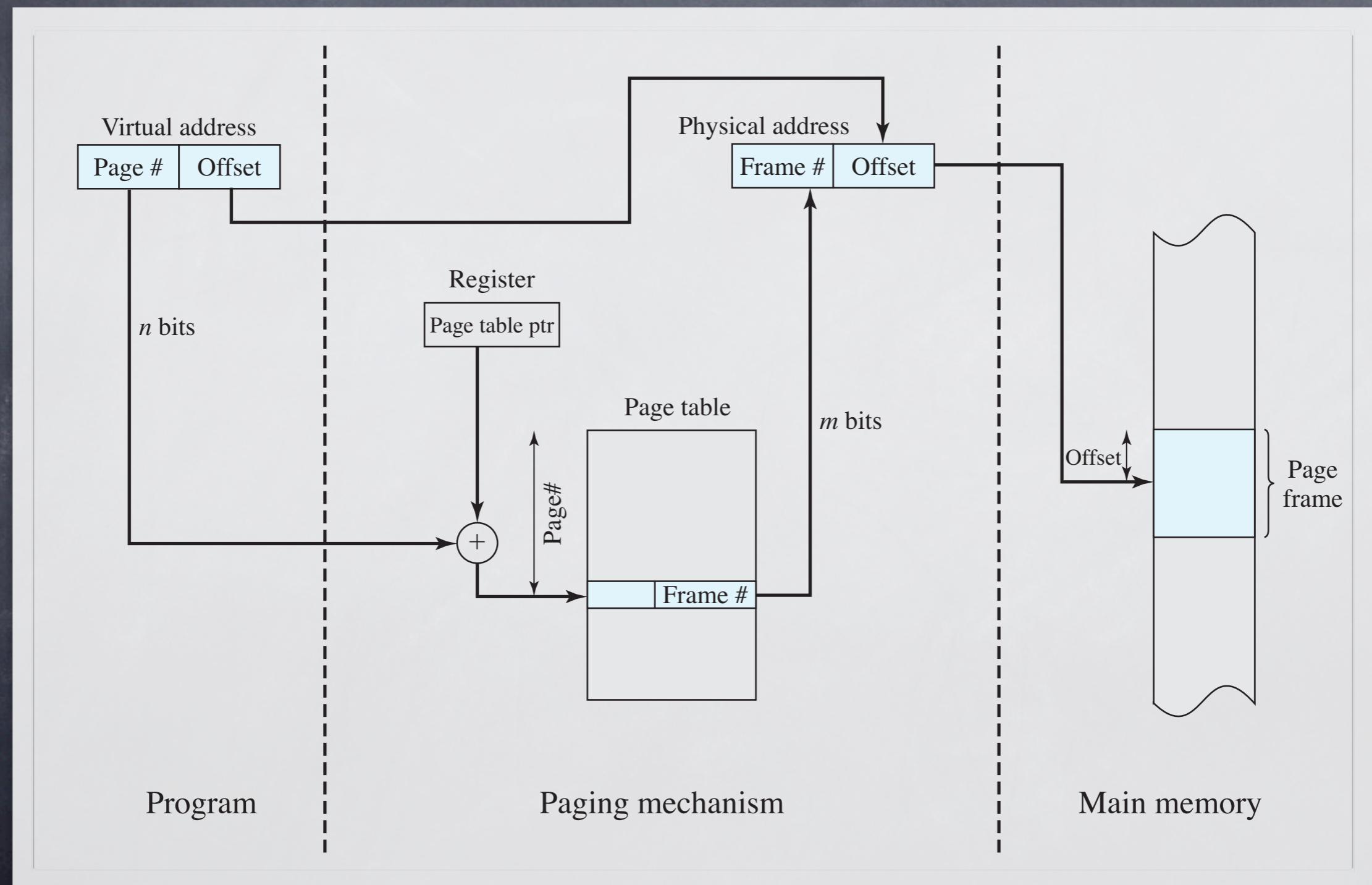
(f) Load process D

Exemplo de mapeamento



Quantas páginas ? Qual o tamanho de cada página ?

Mapeamento por paginação



Exercício

- 8-bit virtual address, 10-bit physical address, and each page is 64 bytes
 - How many virtual pages?
 - How many physical pages?
 - How many entries in page table?
 - Given page table = [2, 5, 1, 8], what's the physical address for virtual address 241?
- m-bit virtual address, n-bit physical address, k-bit page size
 - What are the answers to the above questions?

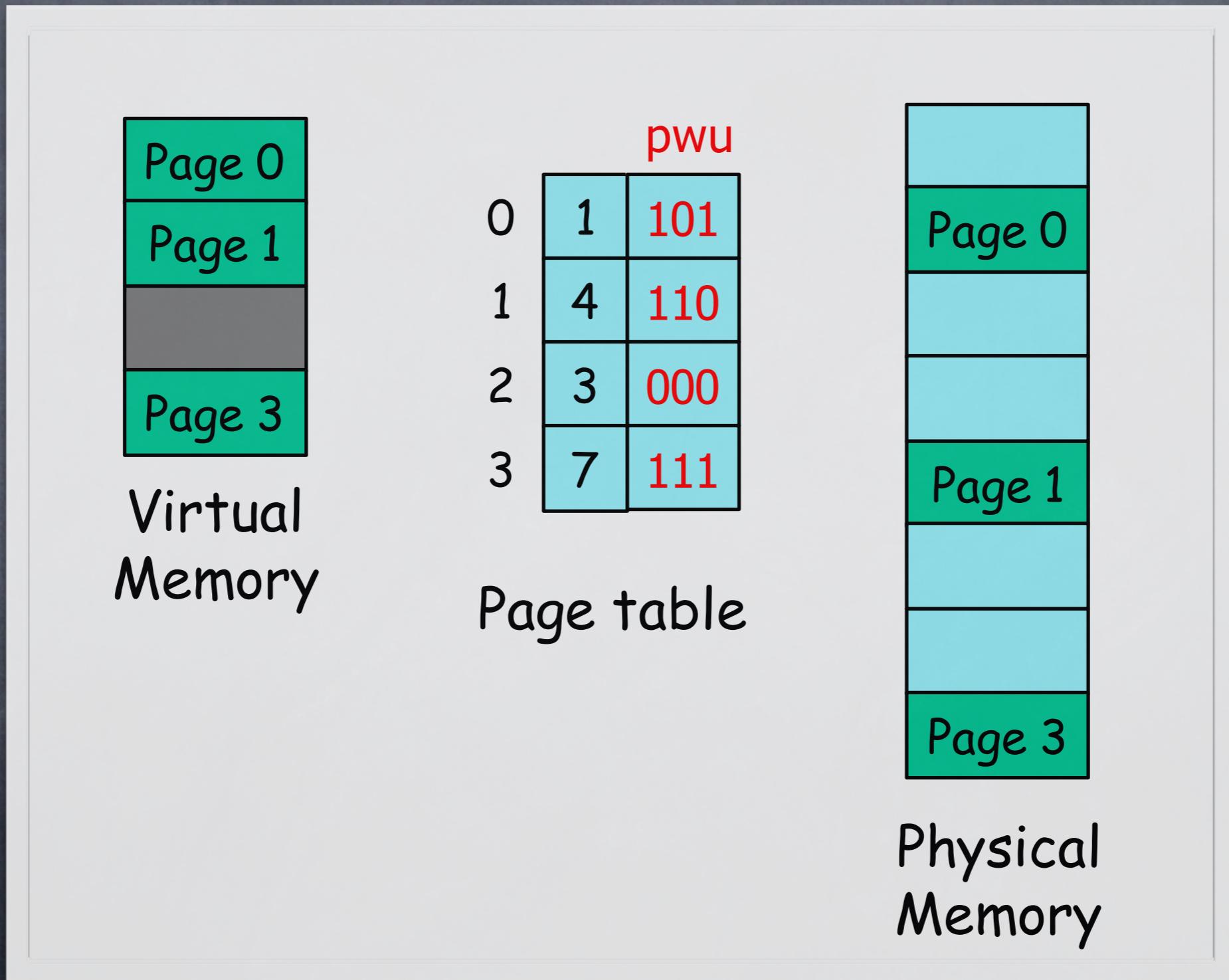
Tamanhos tipos de páginas

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit word
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 Kbytes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
IBM POWER	4 Kbytes
Itanium	4 Kbytes to 256 Mbytes

Proteção de páginas

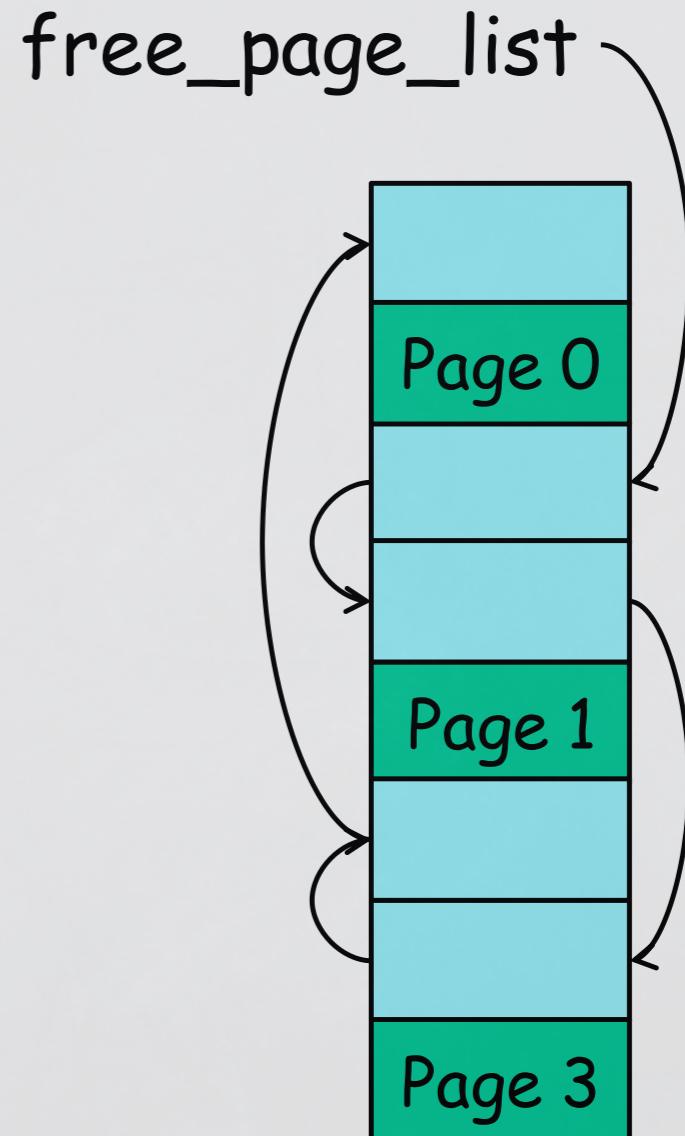
- Implemented by associating protection bits with each virtual page in page table
- Protection bits
 - present bit: map to a valid physical page?
 - read/write/execute bits: can read/write/execute?
 - user bit: can access in user mode?
 - x86: PTE_P, PTE_W, PTE_U
- Checked by MMU on each memory access

Exemplo de proteção



Alocação de páginas

- Free page management
 - E.g., can put page on a free list
- Allocation policy
 - E.g., one page at a time, from head of free list
- xv6: [kalloc.c](#)

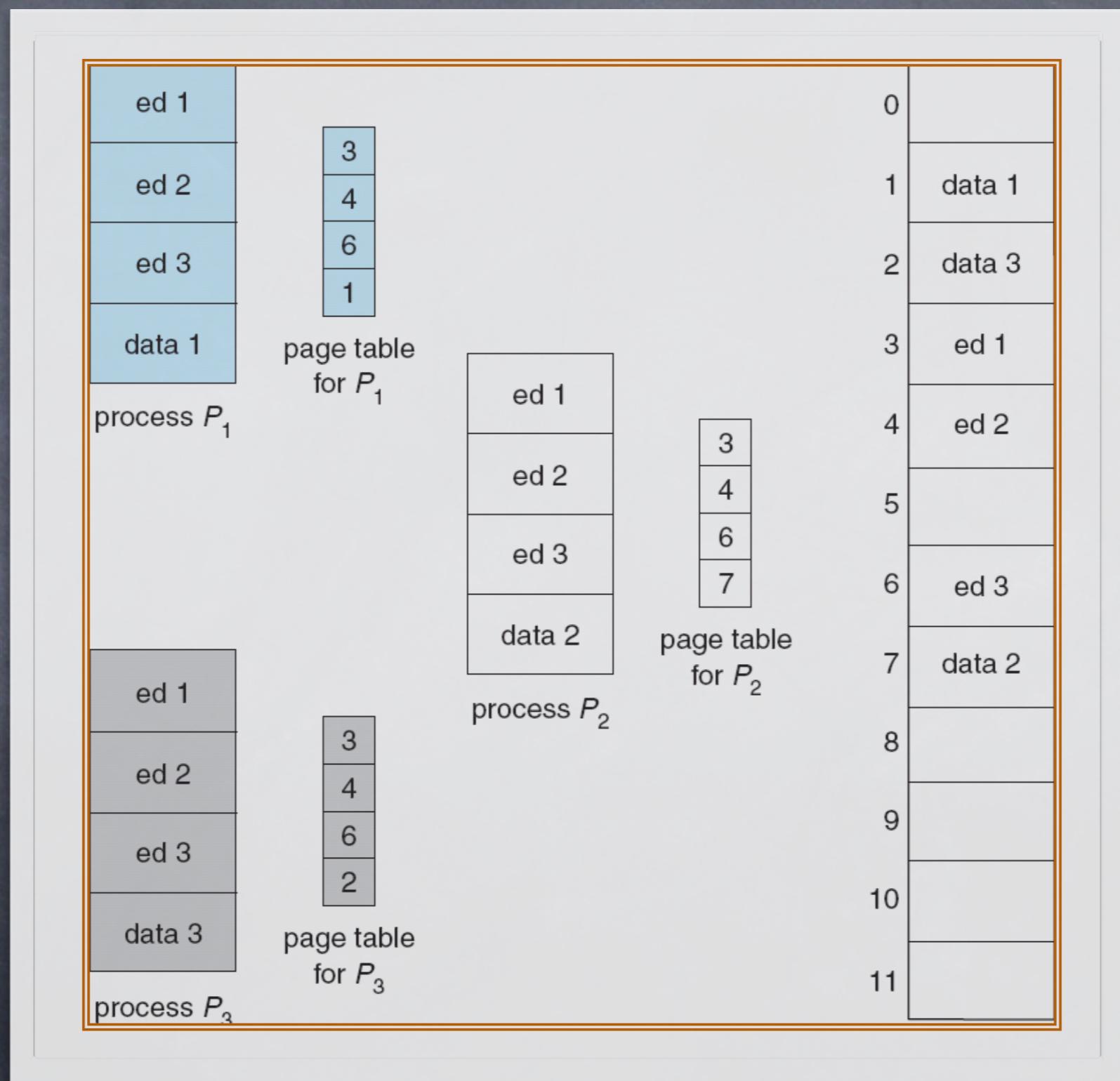


2, 3, 6, 5, 0

Compartilhamento de páginas

- ❑ Efficient communication. Processes communicate by write to shared pages
- ❑ Memory efficiency. One copy of read-only code/data shared among processes
 - Example 1: multiple instances of the shell program
 - Example 2: **copy-on-write fork**. Parent and child processes share pages right after fork; copy only when either writes to a page

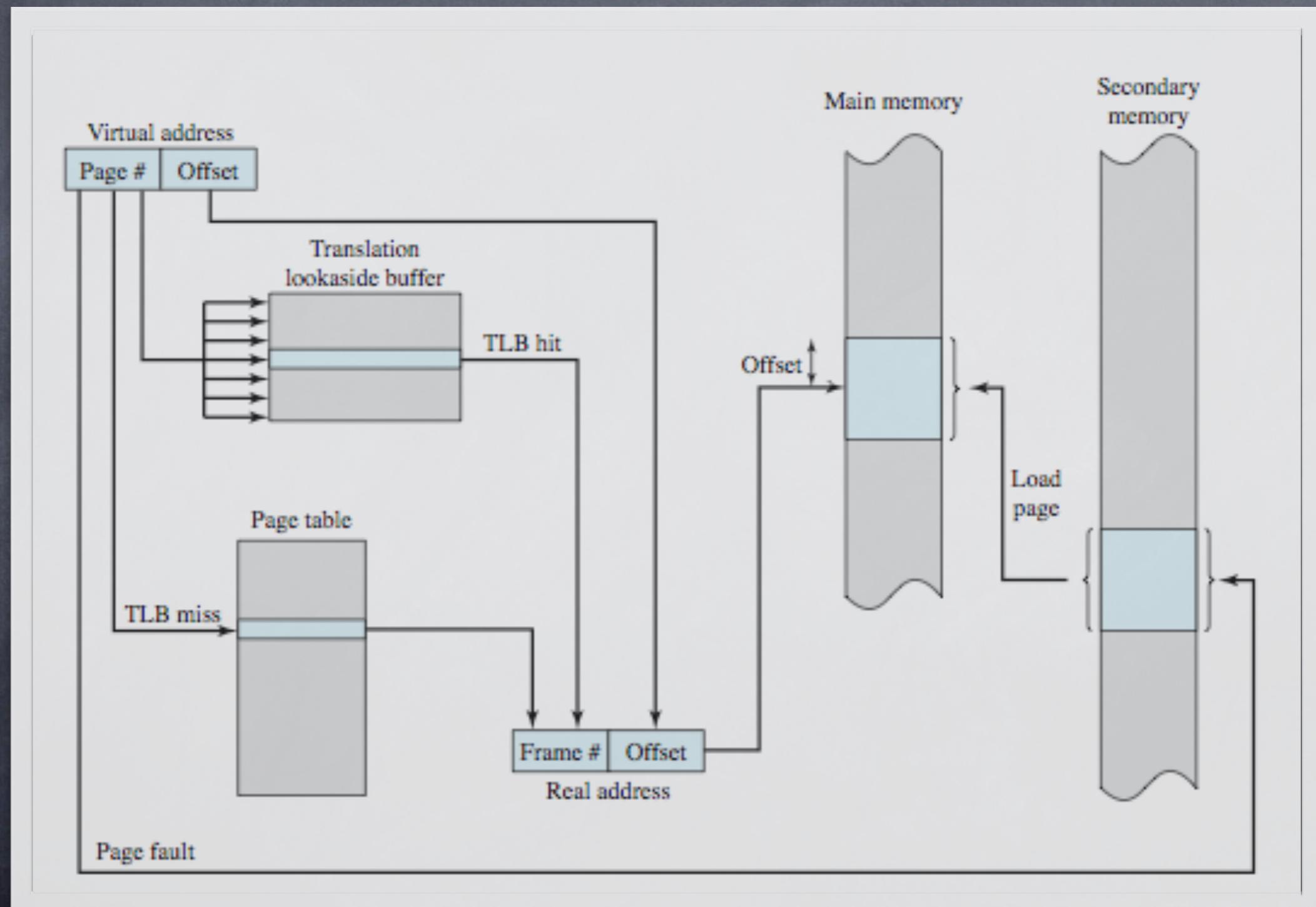
Exemplo de compartilhamento



Implementação da tabela páginas

- Page table is stored in memory
 - Page table base register (PTBR) points to the base of page table
 - x86: cr3
 - OS stores base in process control block (PCB)
 - OS switches PTBR on each context switch
- Problem: each data/instruction access requires two memory accesses
 - Extra memory access for page table

Translation Lookaside Buffer (TLB)



TLB “miss”

- Depending on the architecture, TLB misses are handled in either hardware or software
- Hardware (CISC: x86)
 - Pros: hardware doesn't have to trust OS !
 - Cons: complex hardware, inflexible
- Software (RISC: MIPS, SPARC)
 - Pros: simple, flexible
 - Cons: code may have bug!
 - Question: what can't a TLB miss handler do?

TLB e trocas de contexto

- What happens to TLB on context switches?
- Option 1: flush entire TLB
 - x86
 - load cr3 flushes TLB
 - INVLPG addr: invalidates a single TLB entry
- Option 2: attach process ID to TLB entries
 - ASID: Address Space Identifier
 - MIPS, SPARC

Tempo Efetivo de Acesso

- Associative Lookup = ε time unit
- Assume memory cycle time is 1 ms
- Hit ratio – α
 - Percentage of times that a page number is found in the associative registers; ratio related to number of associative registers
- Effective Access Time (EAT)
$$\begin{aligned} EAT &= (1 + \varepsilon) \alpha + (2 + \varepsilon)(1 - \alpha) \\ &= \alpha + \varepsilon\alpha + 2 + \varepsilon - \varepsilon\alpha - 2\alpha \\ &= 2 + \varepsilon - \alpha \end{aligned}$$

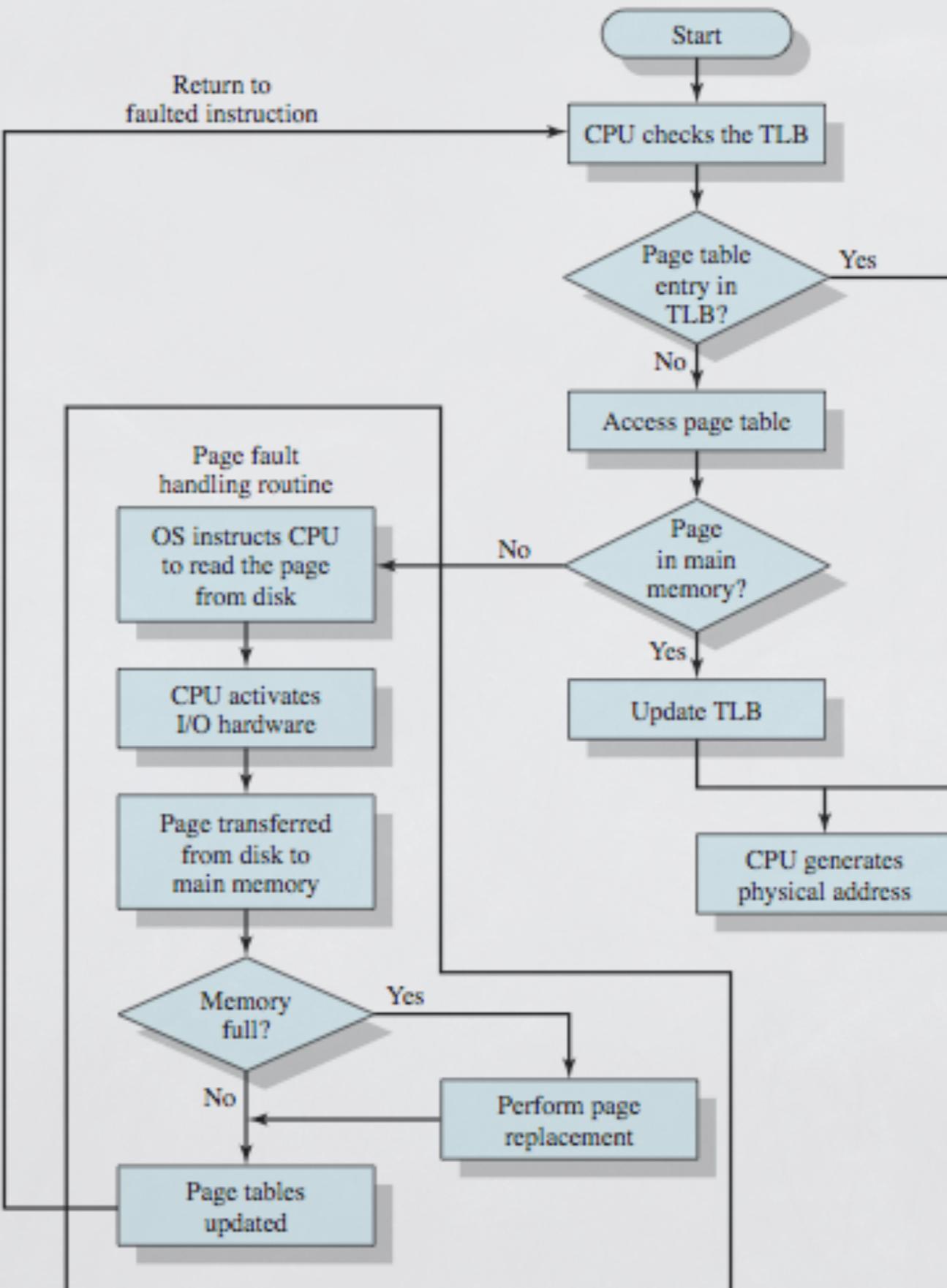


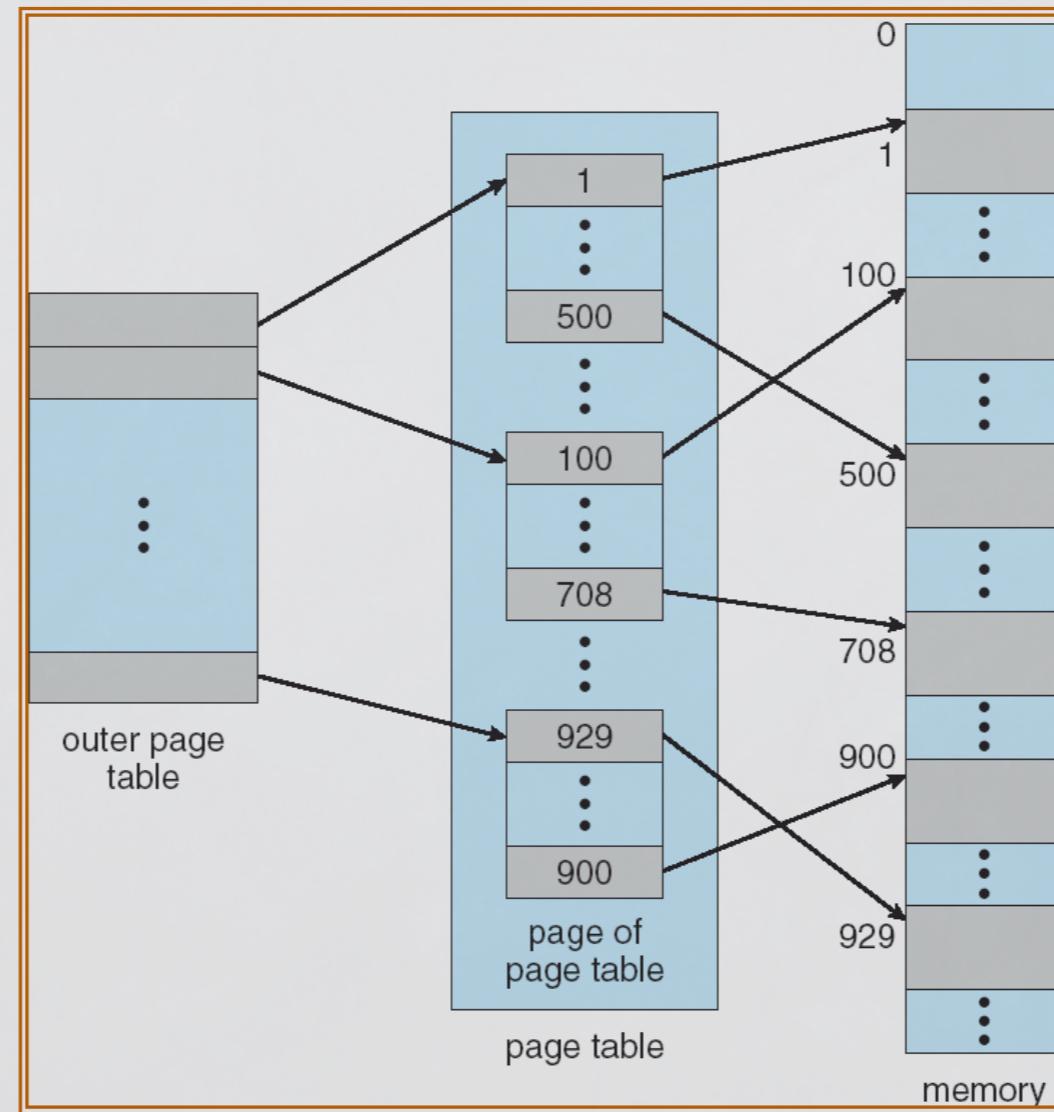
Figure 8.8 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

Tamanho da tabela de páginas

- Given:
 - A 32 bit address space (4 GB)
 - 4 KB pages
 - A page table entry of 4 bytes
- Implication: page table is 4 MB per process!
- Observation: address space are often sparse
 - Few programs use all of 2^{32} bytes
- Change page table structures to save memory
 - Trade translation time for page table space

Tabela de páginas multinível

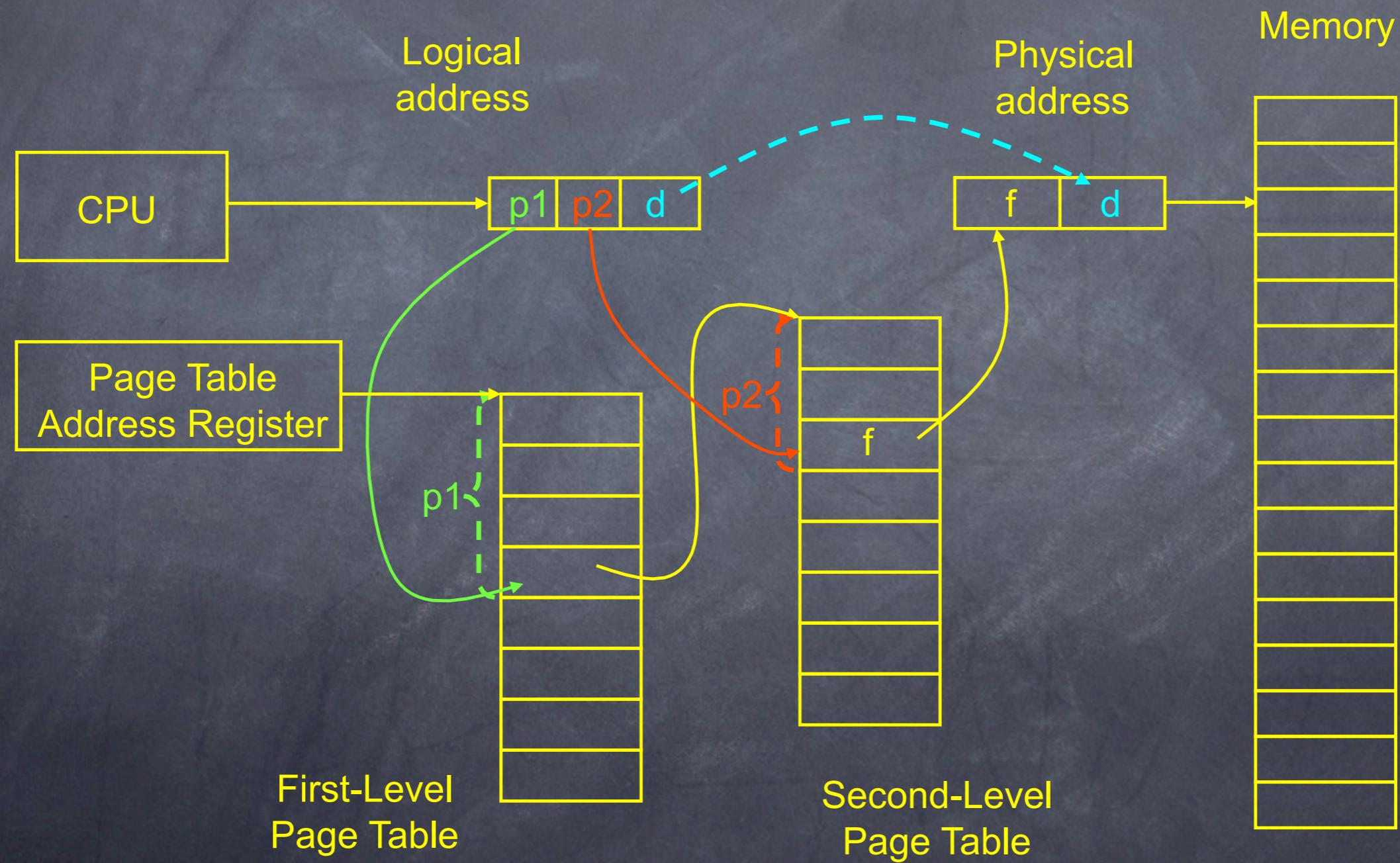
- Break up virtual address space into multiple page tables at different levels



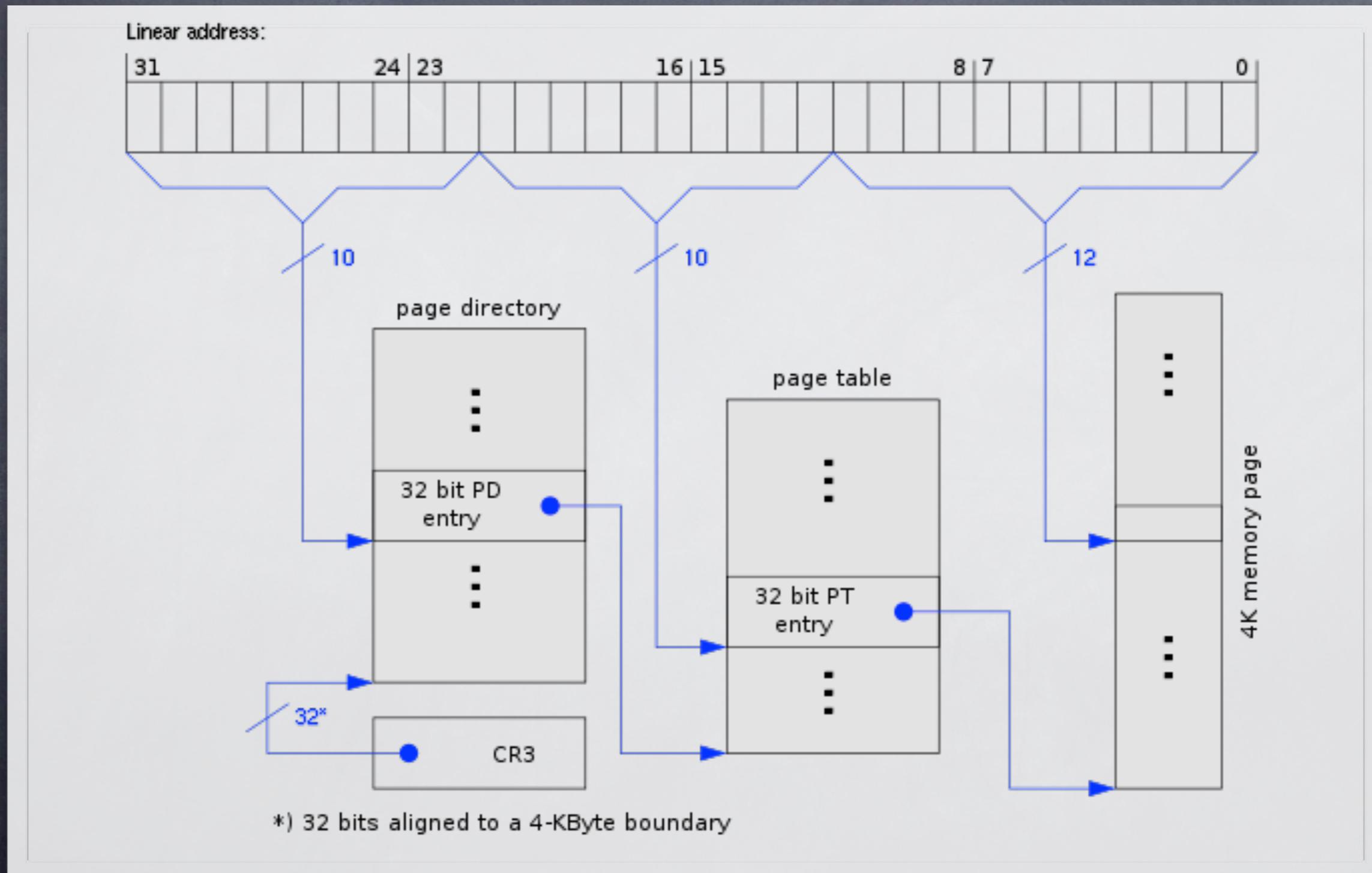
Tabelas multiníveis

- Normalmente divididas em até 4 níveis
- Podem requerer até 5 referências à memória!
- Para nossa sorte, a TLB reflete a coerência espacial ou temporal dos processos
- Rápido acesso

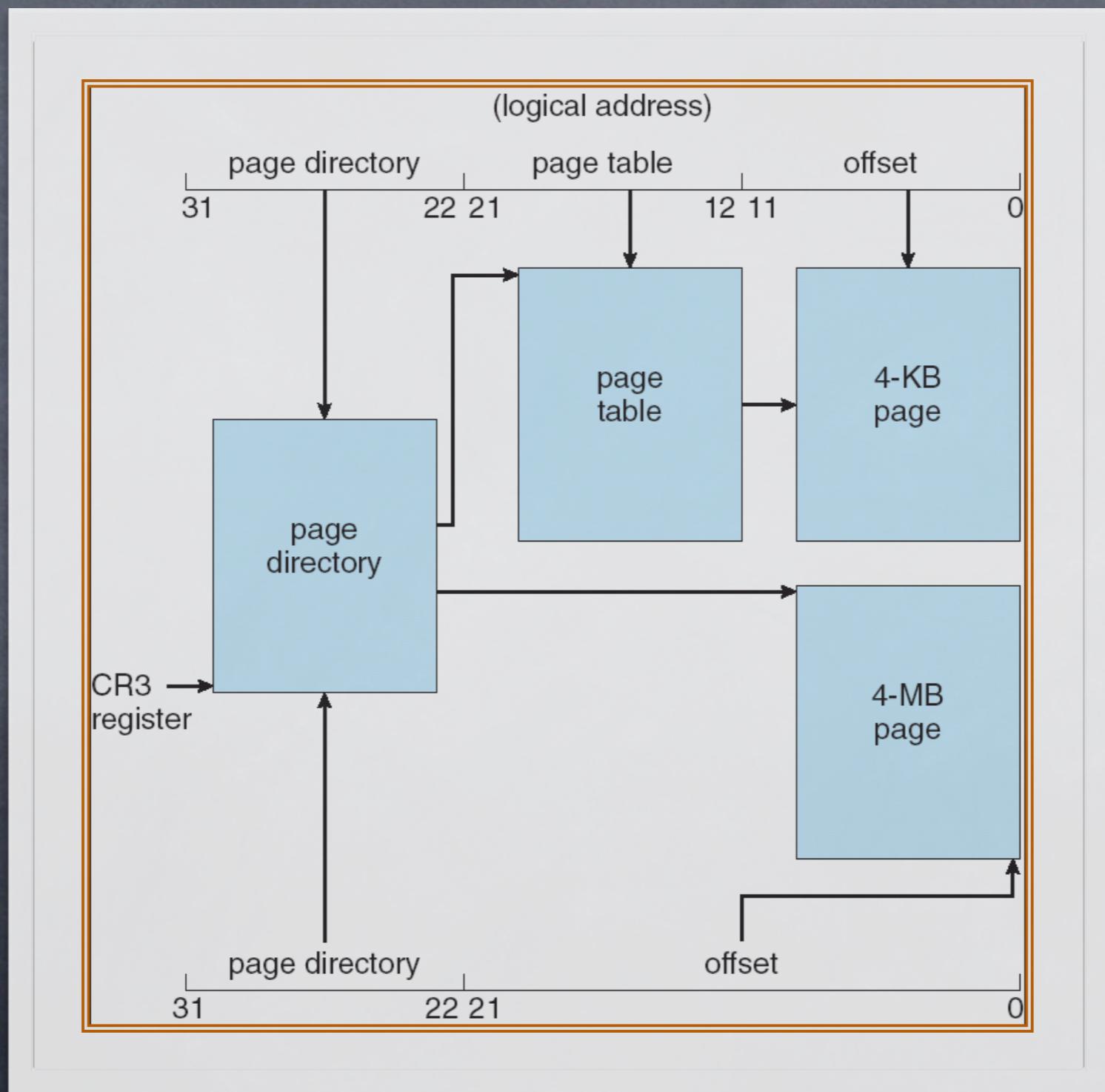
Paginação multinível



Tradução de páginas no xv6

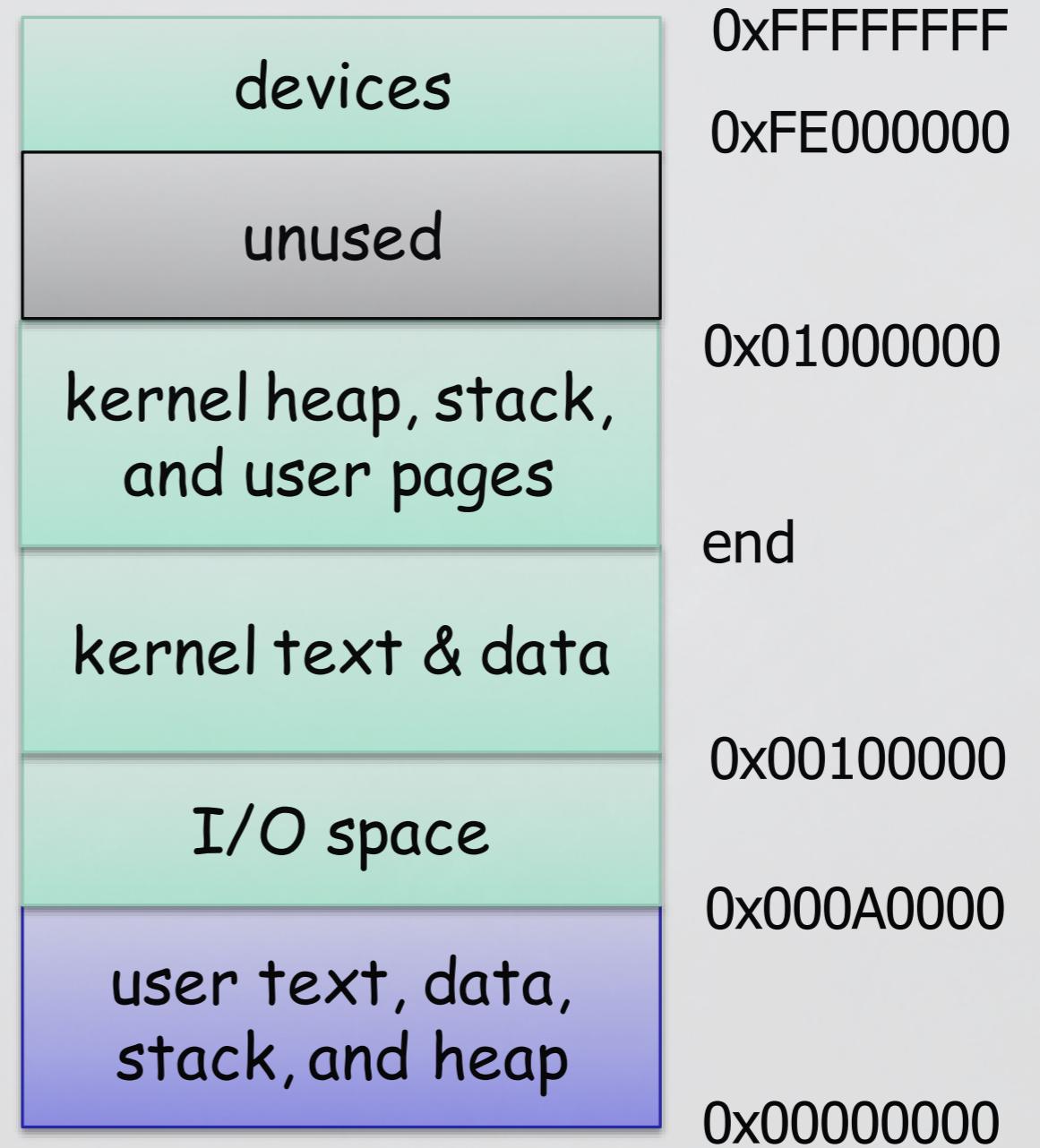


Arquitetura de paginação no x86



Implementação no xv6

- ❑ Split into kernel space and user space
- ❑ User: 0-640KB
 - Map to end-16MB
- ❑ Kernel: 640KB - 4GB
 - Direct (virtual = physical)
- ❑ Kernel: `vm.c, setupkvm()`
- ❑ User: `vm.c, inituvm()` and `exec.c, exec()`



Segmentação com paginação

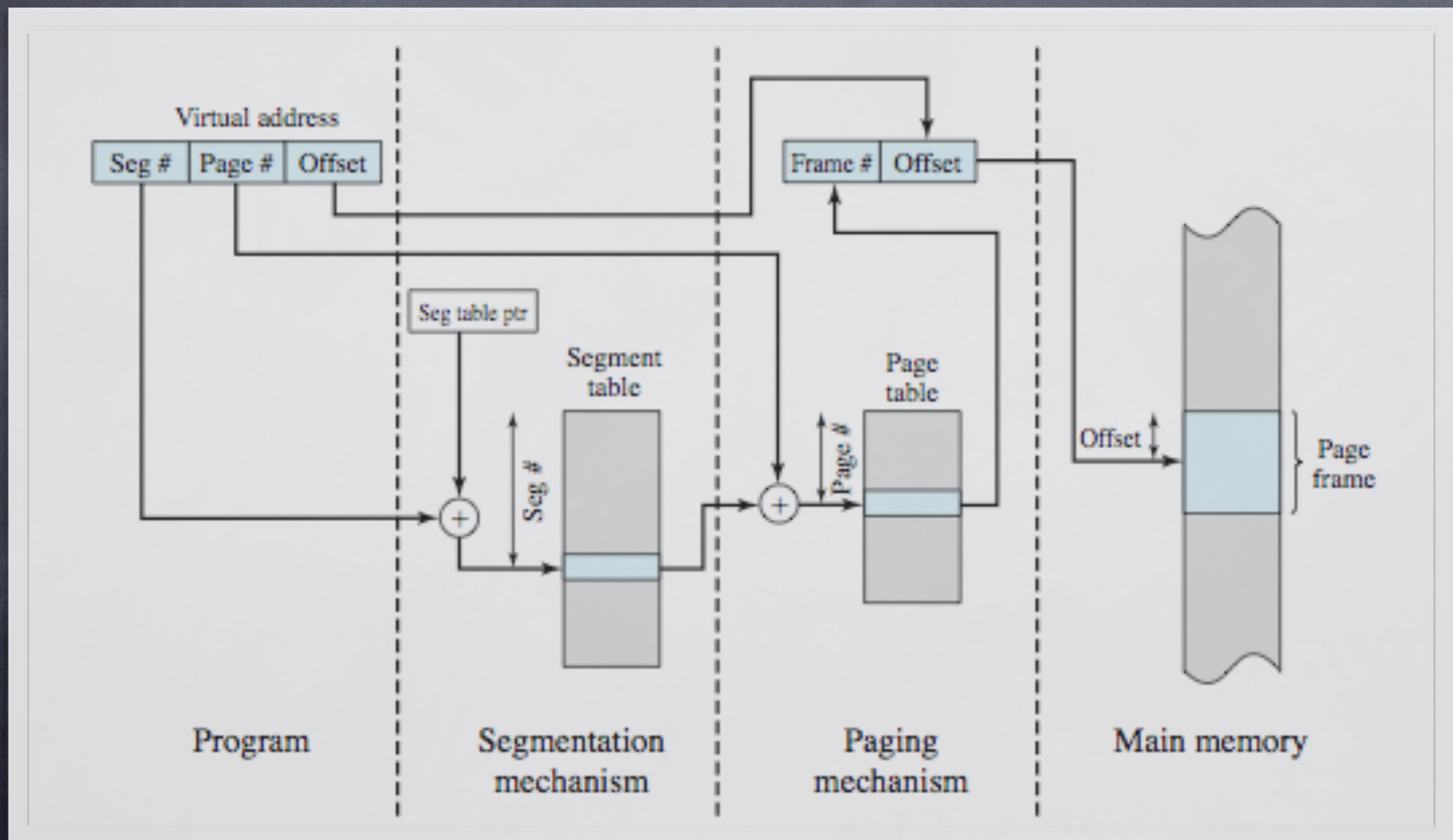
- Remoção da fragmentação externa
- Idéia: paginar os segmentos
- Deixar o compilador/montador/ligador se preocupar em separar os segmentos
- Cada segmento é então quebrado em páginas de forma transparente ao programador e ao compilador/montador/ligador.

Segmentação com paginação

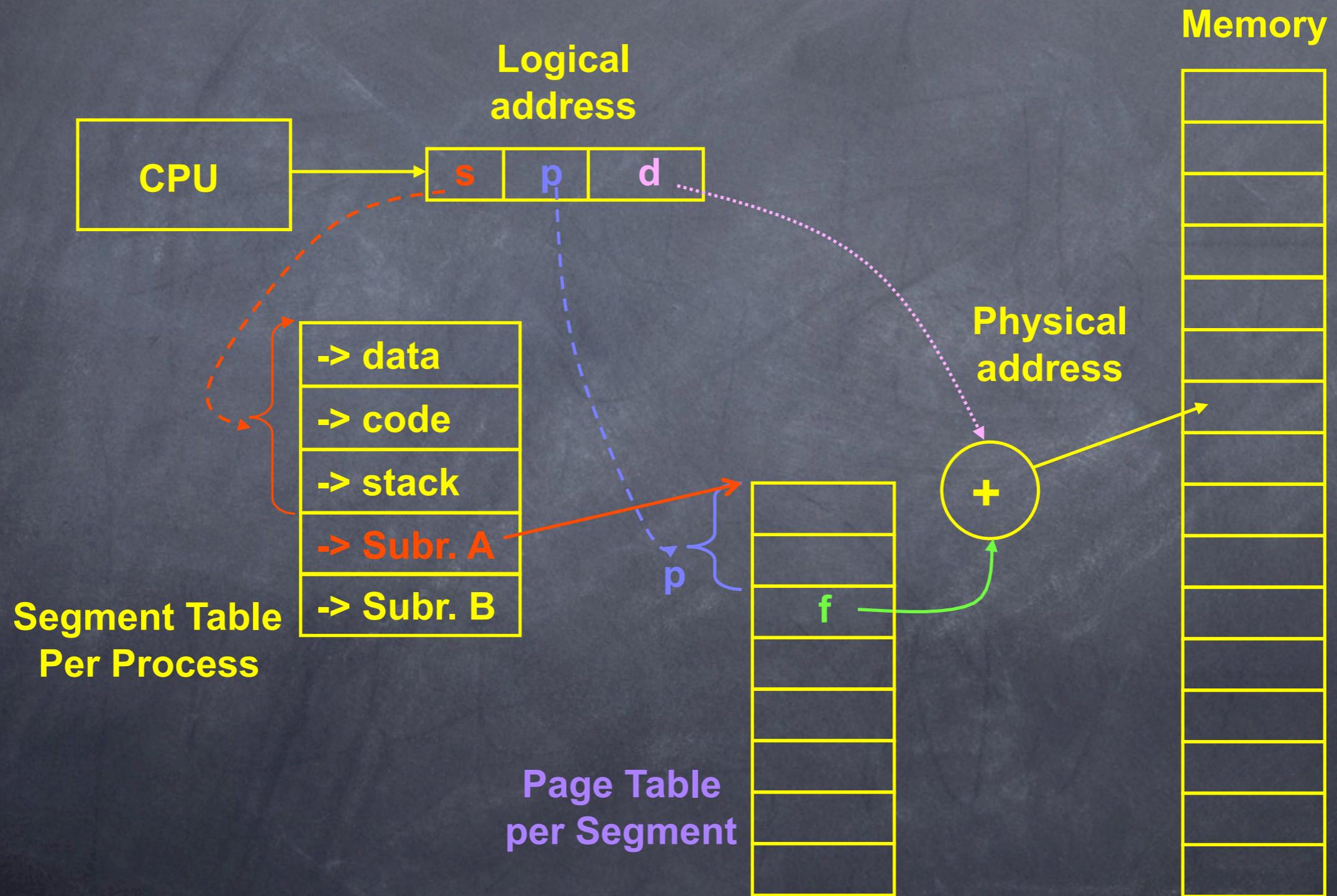
- Structure
 - Segments: logical units in program, such as code, data, and stack
 - Size varies; can be large
 - Each segment contains one or more pages
 - Pages have fixed size
- Two levels of mapping to reduce page table size
 - Page table for each segment
 - Base and limit for each page table
 - Similar to multi-level page table
- Logical address divided into three portions

seg #	page #	offset
-------	--------	--------

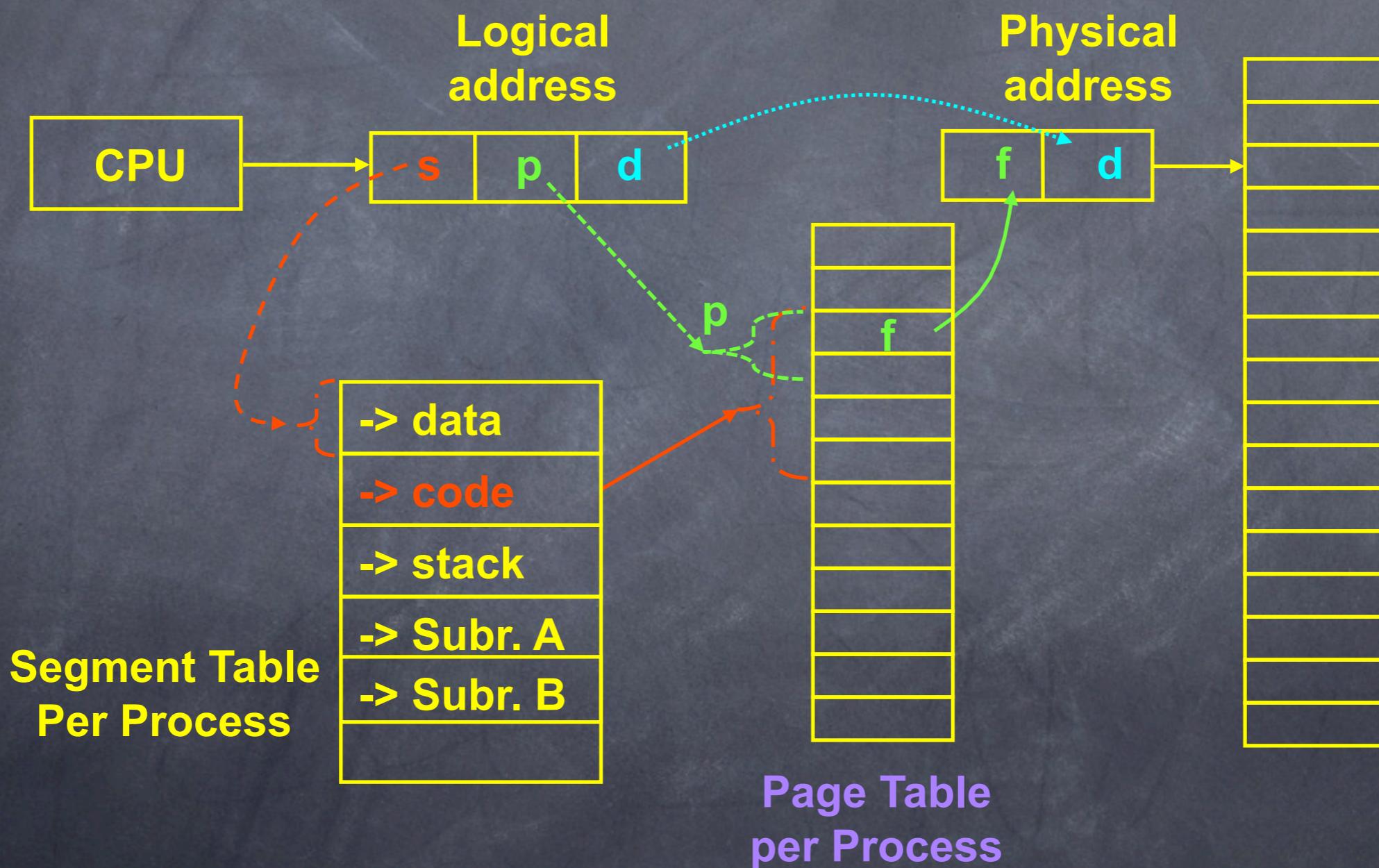
Segmentação com paginação



Segmentação com paginação

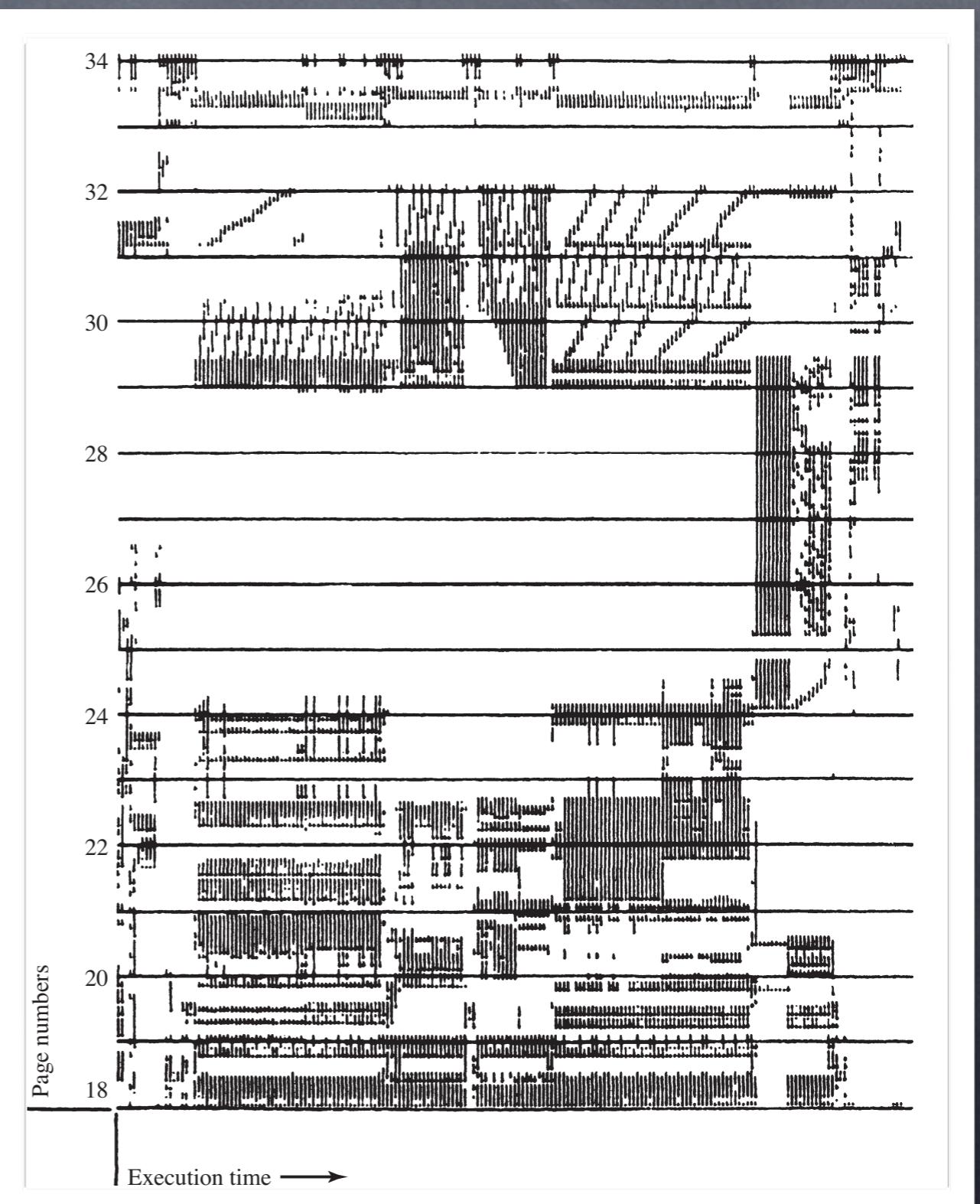


Segmentação com endereçamento linear



Localidade de referência

- Processos não utilizam RAM de forma aleatória:
 - Coerência espacial e temporal
- Regra 80/20
 - 20% do programa realiza 80% do trabalho



Paginação por demanda

- Não carregar páginas até que elas sejam referenciadas
- Podemos executar programas cujo tamanho seja maior que a própria RAM
- Mudar o significado do bit de validade
 - Indica se página está ou não em RAM
 - Interrupção de falta de página (“page fault”)
 - Primeiro faz com que o número da página seja válido
 - Caso positivo, carrega a página e atualiza tabela
- Chamado de “Memória Virtual”

Tarefa de Casa

- EP #2
 - Implementação de syscalls
- Leitura sobre o xv6:
 - trap.pdf
 - mem.pdf