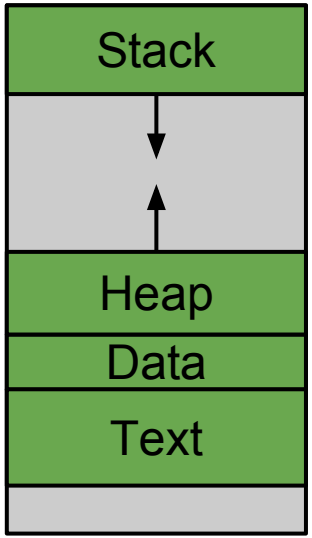
SO: Segmentação/Paginação

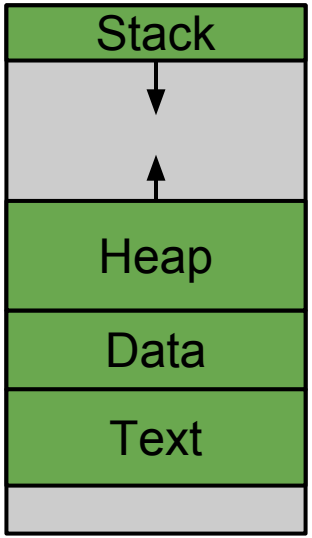
Sistemas Operacionais
2017-1
Flavio Figueiredo (http://flaviovdf.github.io)

Segmentação

Segmentação

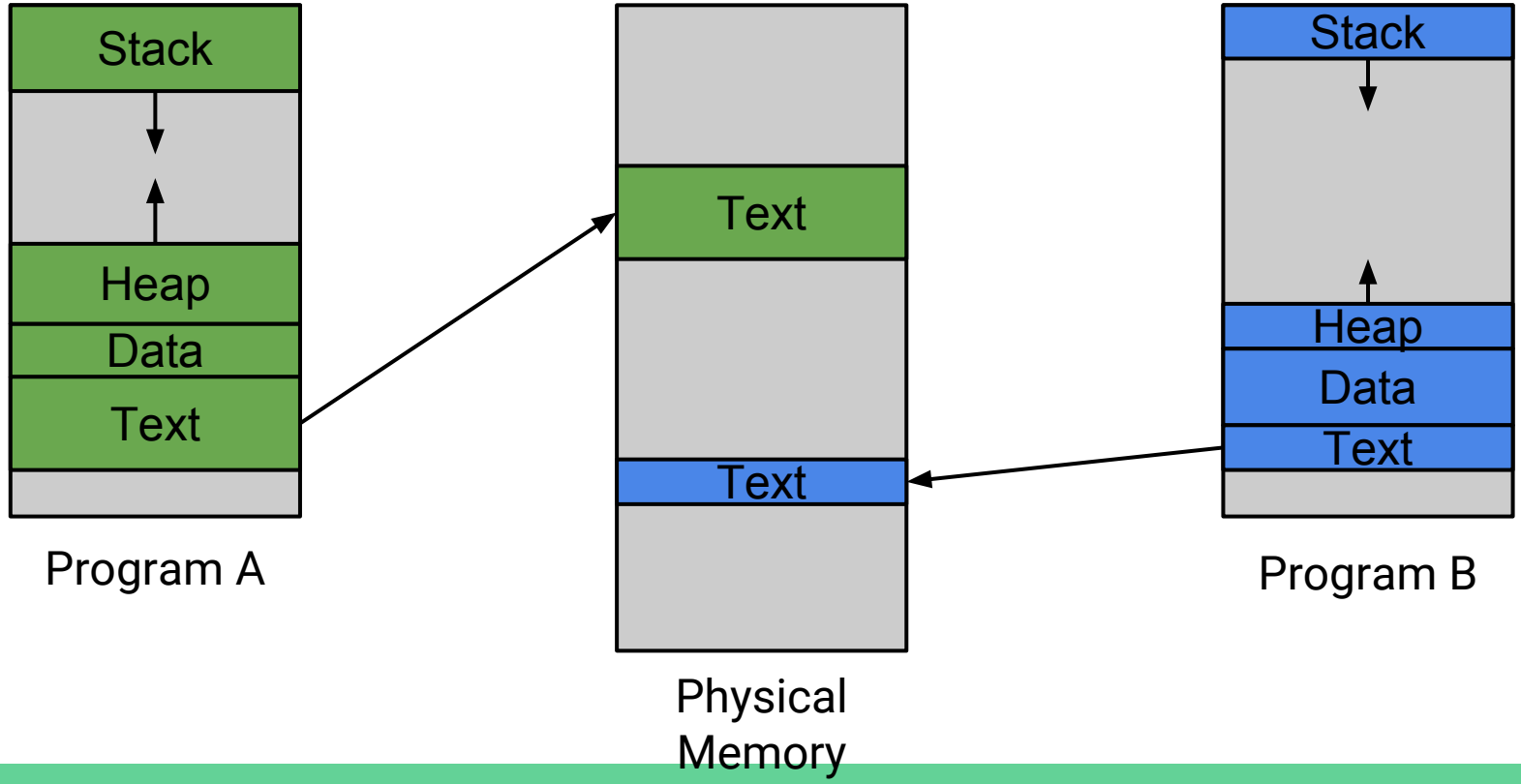


Program A

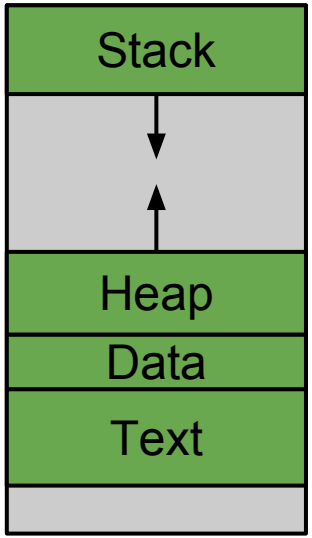


Program B

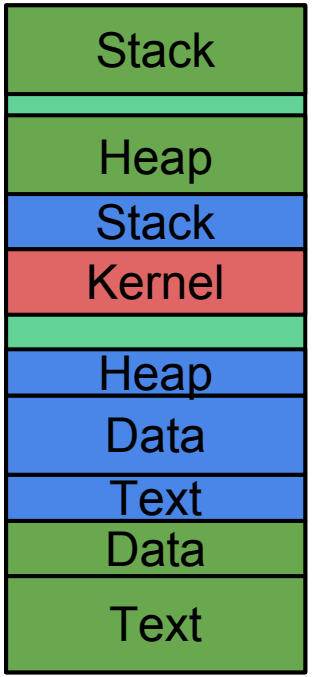
Segmentação



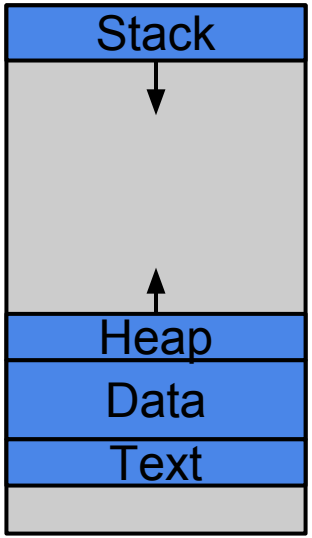
Segmentação



Program A



Physical Memory



Program B

Como implementar?

Como implementar?

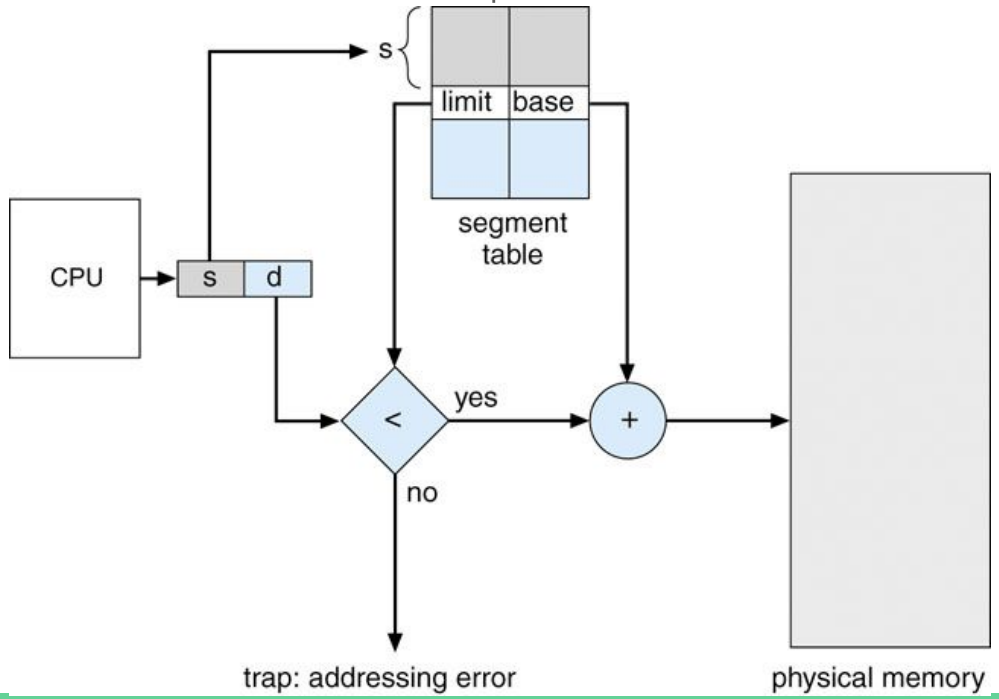
- Particionar a memória em segmentos
  - Code, heap, stack ...
- Lidamos com a fragmentação interna
- Ainda temos a externa

Como Implementar?

- Cada segmento faz um início e um tamanho
  - Similar ao base e limite da MMU mais simples

Baixo Nível

- Cada segmento faz um início e um tamanho
  - Similar ao base e limite da MMU mais simples



Lembrando dos Requisitos

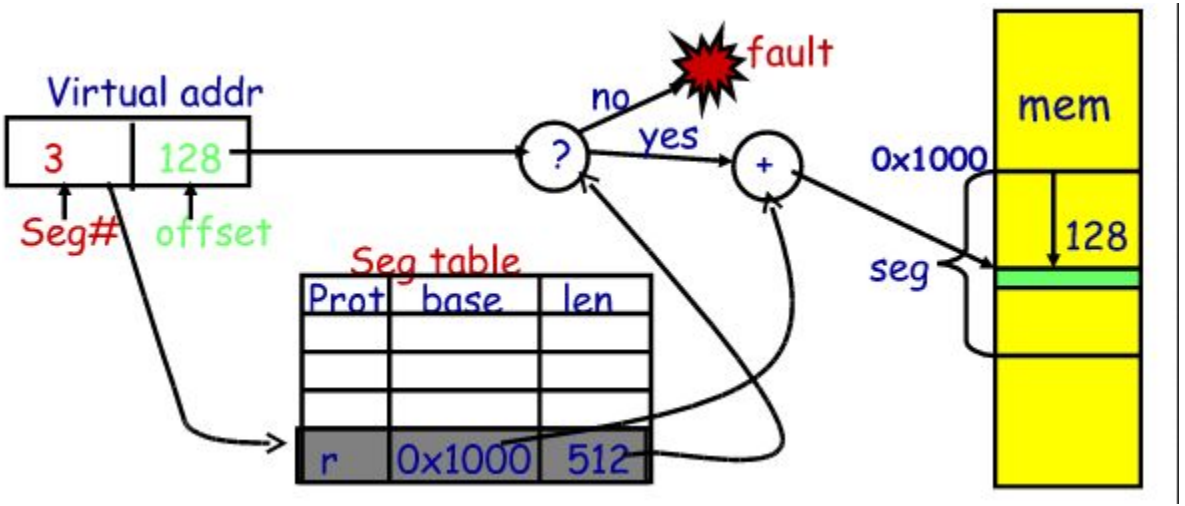
- Proteção
- Transparência
- Recursos suficientes

Lembrando dos Requisitos

- Proteção
  - Campos de Read/Write/Execute
  - Data
    - Read/Execute
  - Heap
    - Read/Write
- Transparência
  - Base e Limite por Segmento
- Recursos suficientes
  - Ainda é um problema
  - Compartilhamento de segmentos ajuda
  - Ou jogar segmentos no disco (mas temos uma ideia melhor para isto, mais a frente)

Em Bits

- Alguns poucos bits para o segmento (2 ou 3)
- Restante dos bits para o endereço
- Shifts, adds e compares para fazer a tradução



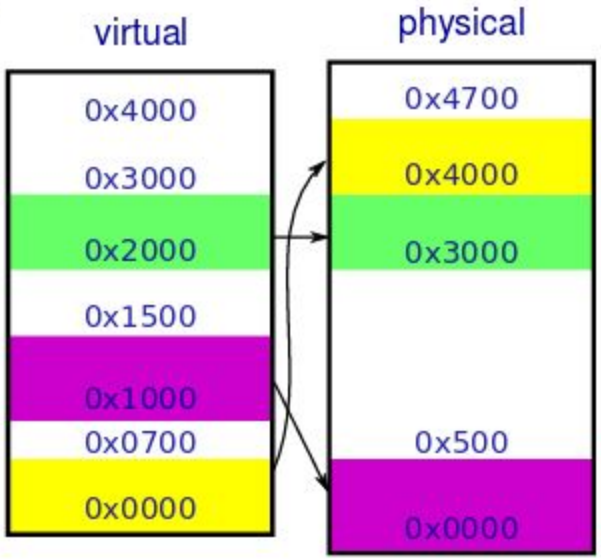
Em Bits

- Alguns poucos bits para o segmento (2 ou 3)
- Restante dos bits para o endereço
- Shifts, adds e compares para fazer a tradução

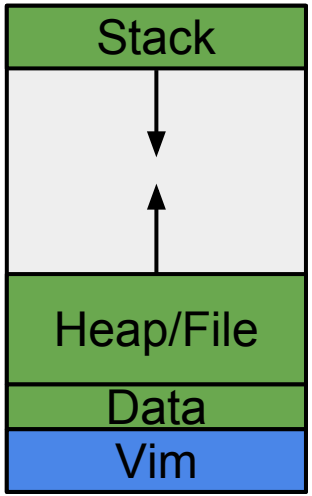
```
1   // get top 2 bits of 14-bit VA
2   Segment = (VirtualAddress & SEG_MASK) >> SEG_SHIFT
3   // now get offset
4   Offset  = VirtualAddress & OFFSET_MASK
5   if (Offset >= Bounds[Segment])
6       RaiseException(PROTECTION_FAULT)
7   else
8       PhysAddr = Base[Segment] + Offset
9       Register = AccessMemory(PhysAddr)
```

Em Bits

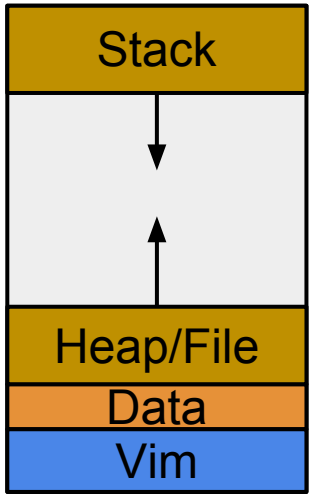
Seg	base	bounds	rw
0	0x4000	0x6ff	10
1	0x0000	0x4ff	11
2	0x3000	0xfff	11
3			00



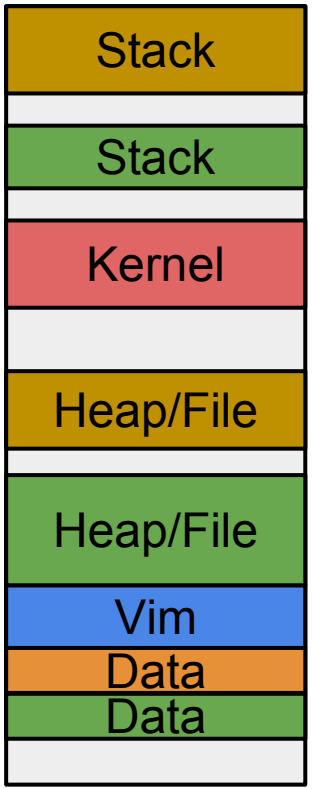
Compartilhando Memória



Vim 1



Vim 2



Physical Memory

Considerações

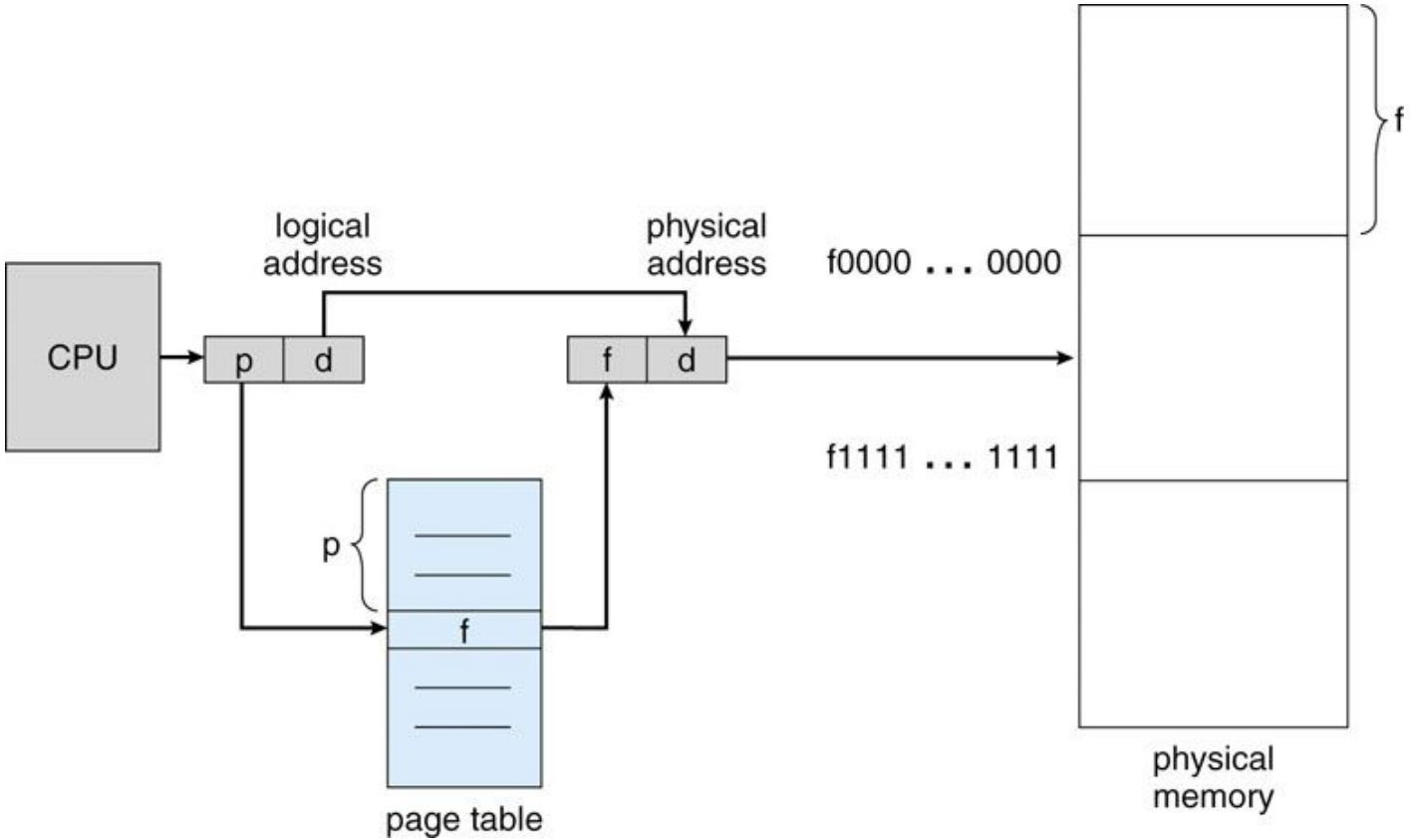
- Segmentação pura é bem simples de implementar
- Ainda temos problemas de fragmentação
- Ainda temos problemas se um segmento não cabe em memória
- Vamos fatiar mais ainda os programas

Paginação

Paginação

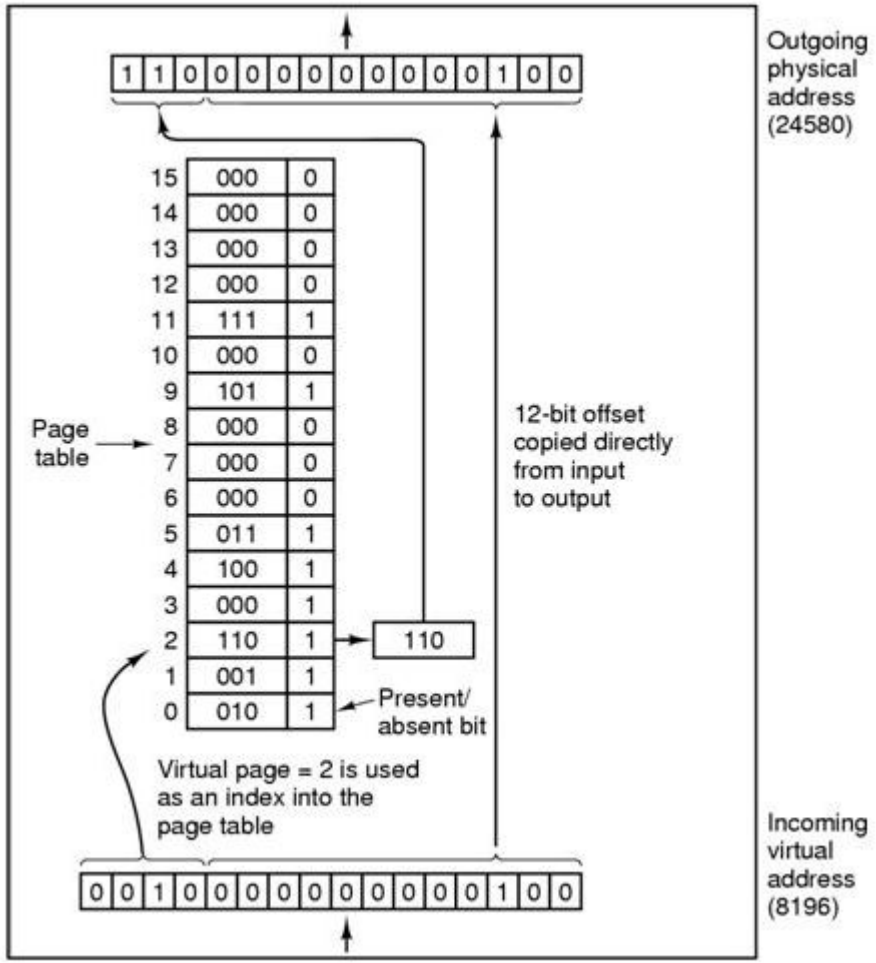
- “Fatiamento” da memória em *frames* de tamanhos iguais
  - 4KiB ou 4MiB
- Páginas virtuais
  - Um programa com **n** páginas faz uso de **n** frames
- Memória gerencia as páginas não utilizadas

Hardware

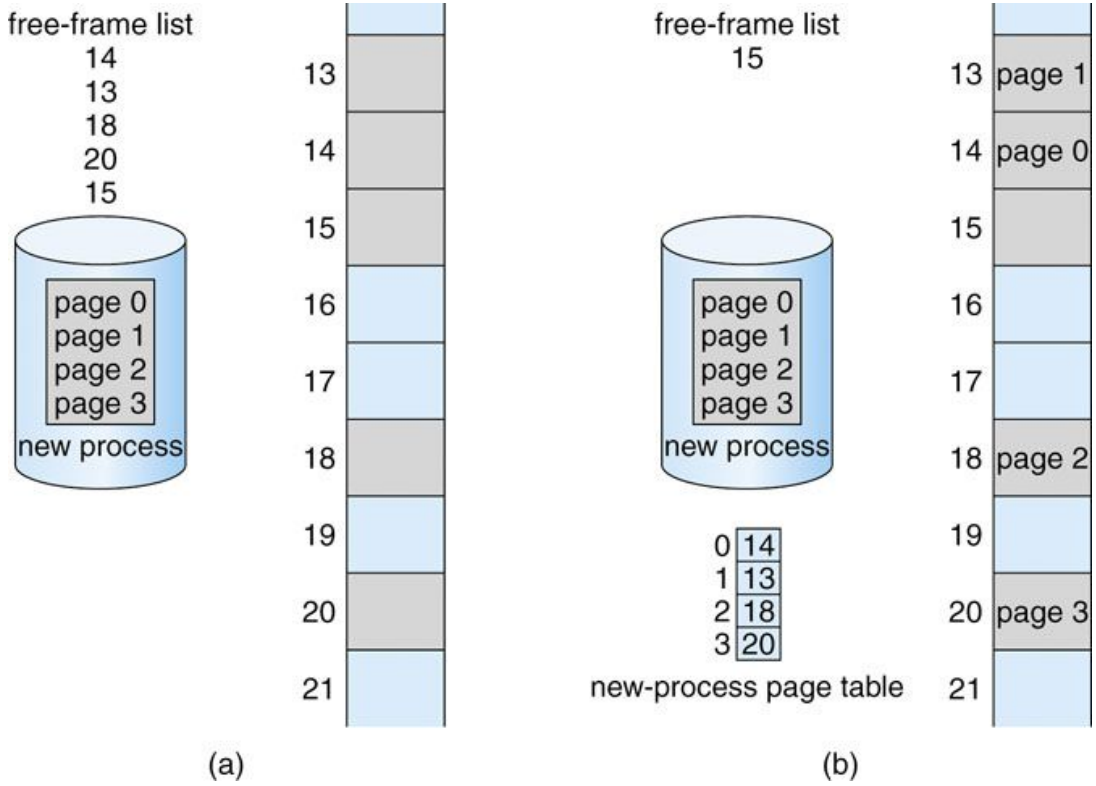

CPU
logical address
p
d
physical address
f
d
f0000 ... 0000
f1111 ... 1111
f
p
f
page table
physical memory

Hardware

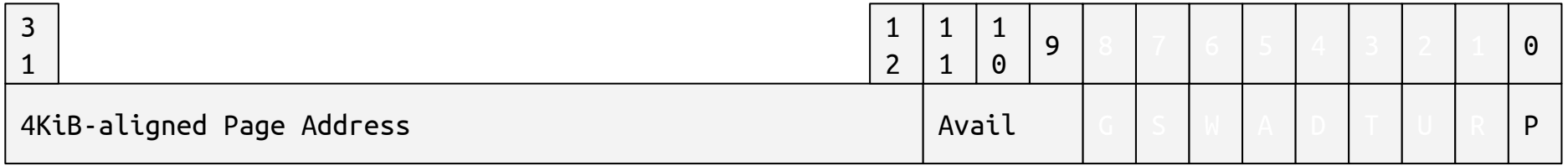
- Primeiros bits para identificar a página
- Seguintes identificam o endereço
- Tradução em hardware
- Tabelas de página na memória



Novos Processos



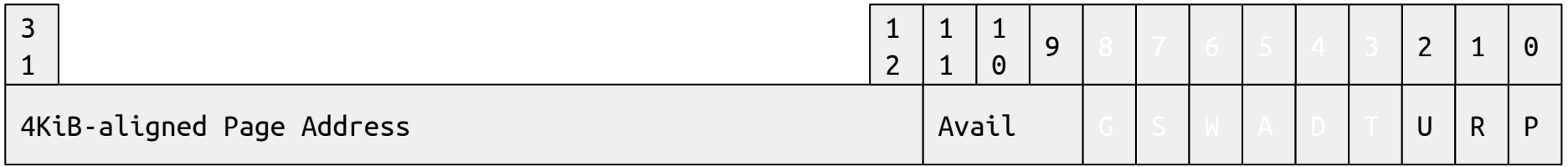
x86 page entry



- **Available:** Free for the OS to use. If P is unset, all bits are available.

- **P:** If 1, page is in memory, otherwise it is not (page fault).

Proteção



- **Available:** free for the OS to use. If P is unset, all bits are available.

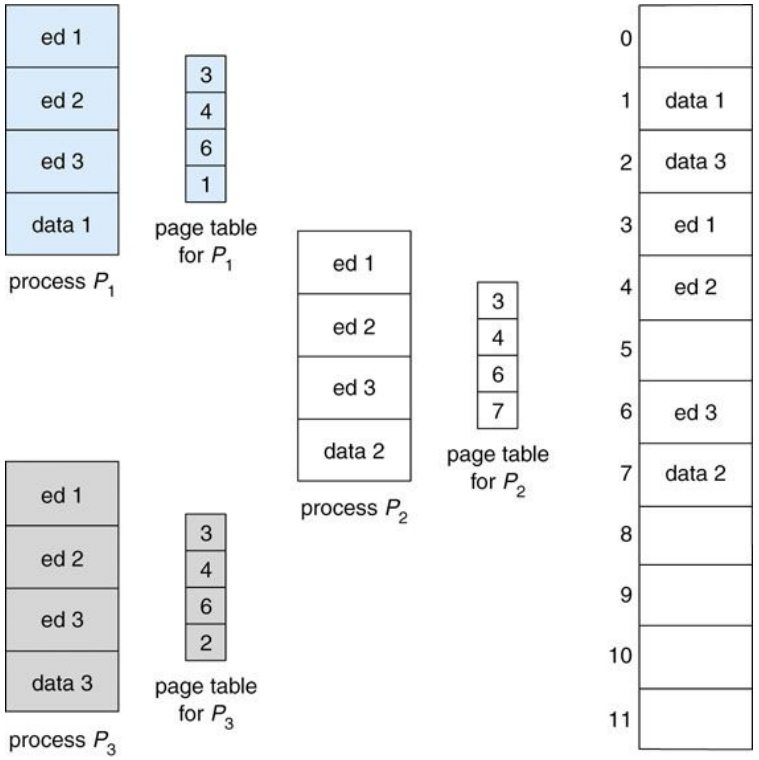
- **U:** If 1, page is user-accessible, otherwise only supervisor-accessible.
- **R:** If 1, page is read-write, otherwise it is read-only.
- **P:** If 1, page is in memory, otherwise it is not (page fault).

Compartilhamento de Páginas

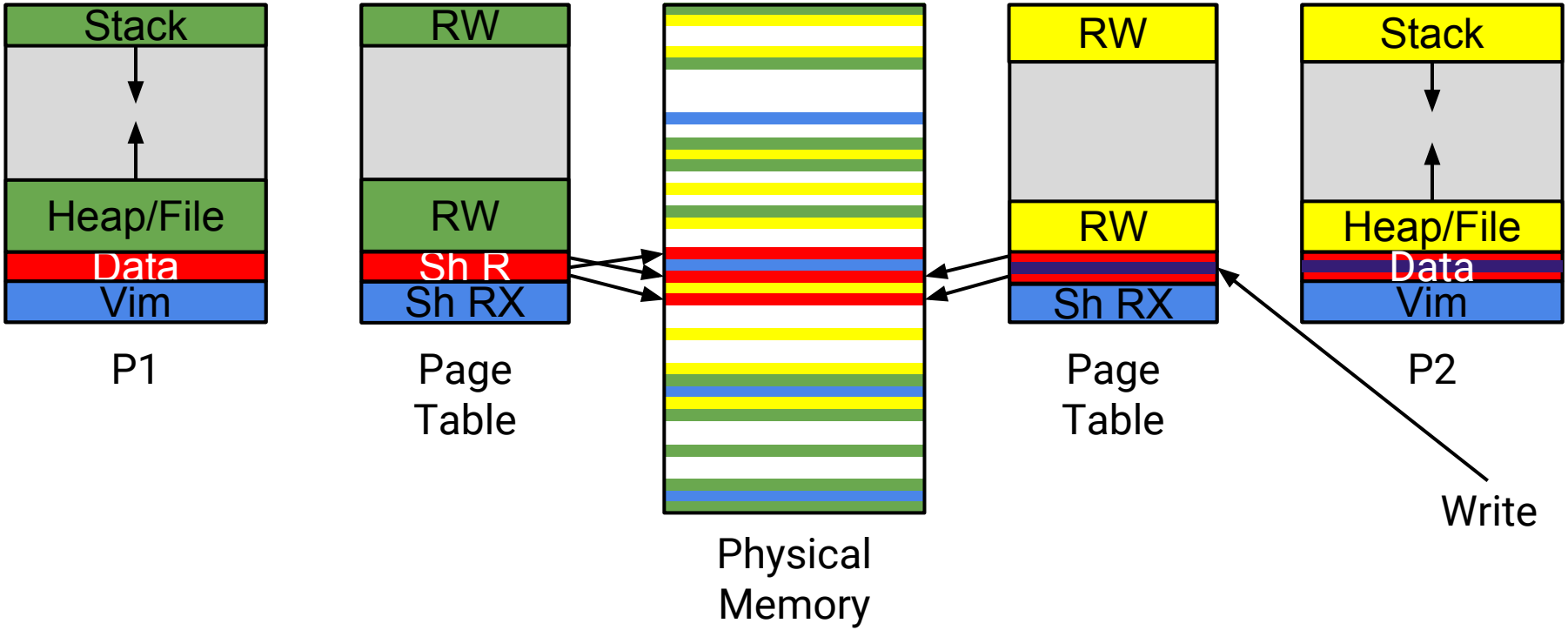
- Similar a segmentação
- Mais controle
  - Pequenos pedaços
  - Lembre-se do fork/exec

Compartilhamento de Páginas

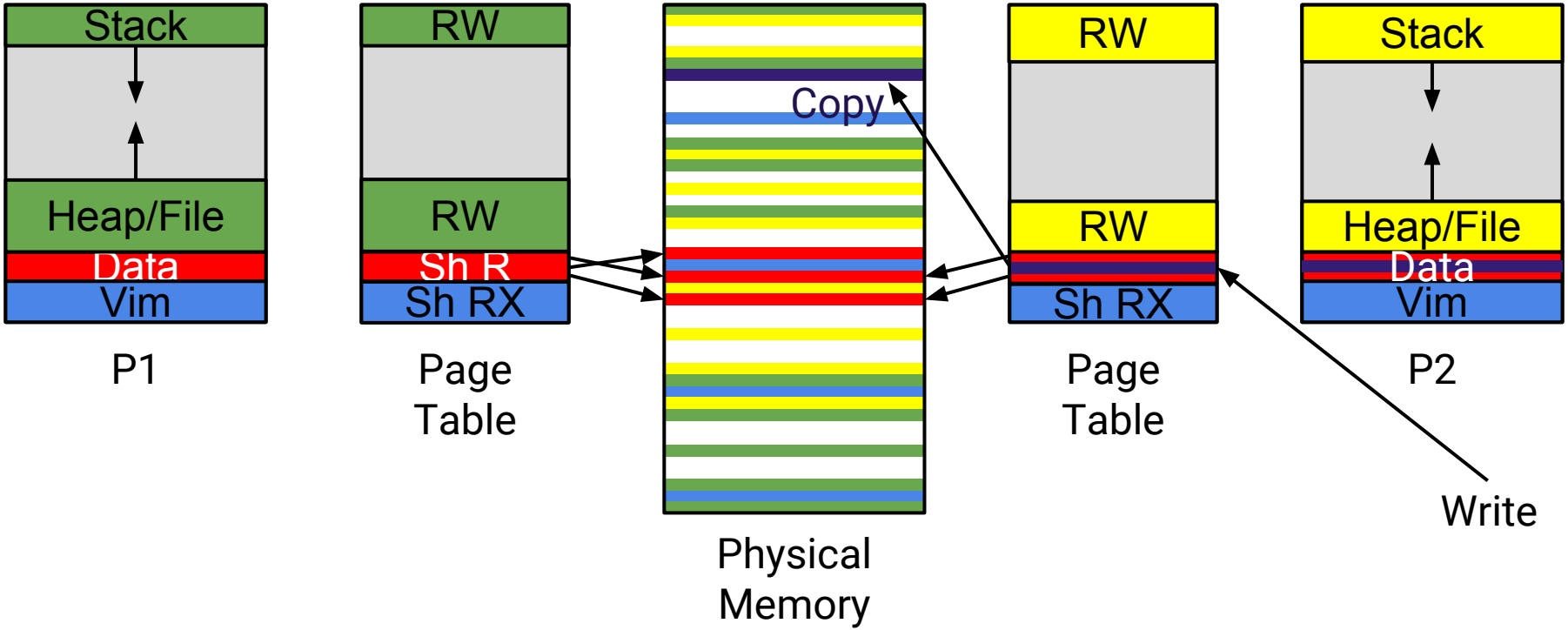
- Cada página com número de referências
  - Processos
- Se zerar
  - A página pode ser liberada
- Páginas compartilhadas iniciam como read-only
  - Viram write apenas na primeira escrita
  - Page-fault



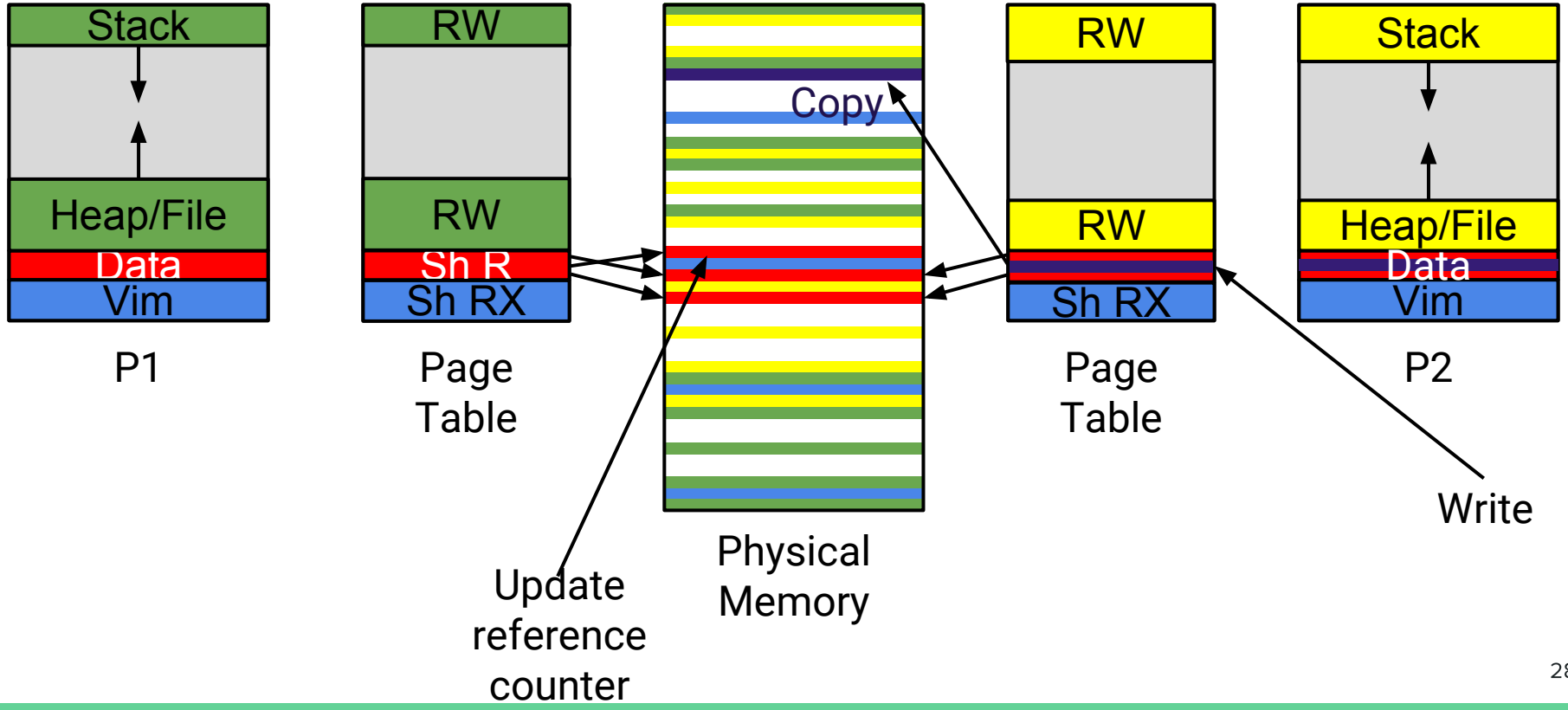
Copy-on-write



Copy-on-write



Copy-on-write



Lembrando de OAC

- A memória armazena em unidades de bytes
- Cada endereço de 32 bits
  - 1 byte de dados

Tamanho de Páginas (registradores de 32 bits)

- 4KB de de tamanho geralmente
  - Offset de tamanho 2 ** 12 bits

- Quanto resta?

Tamanho de Páginas (registradores de 32 bits)

- 4KB de de tamanho geralmente
  - Offset de tamanho 2 ** 12 bits

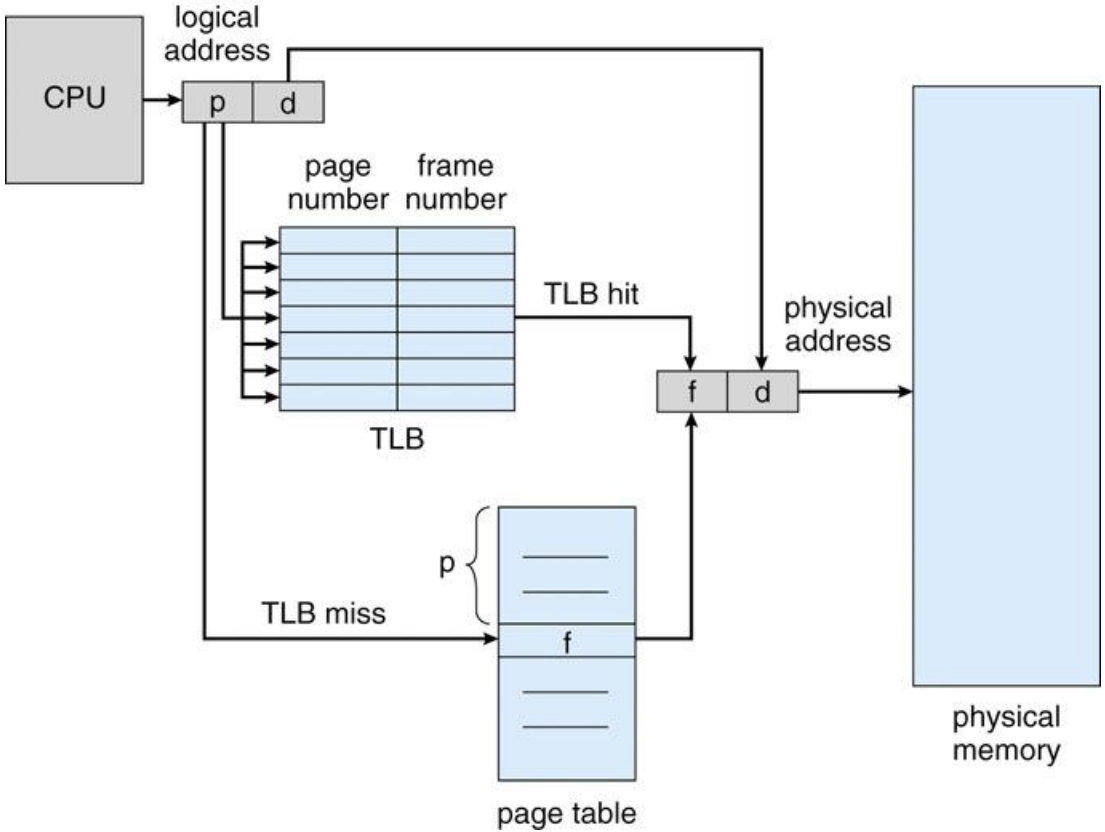
- Quanto resta?
  - 2 ** 20
  - Assumindo 4 bytes de dados na tabela de página
  - 4MB por processo
  - Para 200 processos (800MB)
  - Não é uma boa ideia

+ Problemas

- Paginação é lenta
- Reside na memória
- Operações feitas por software

Vamos focar apenas no problema de velocidade.
Como resolver?

Caching the page table: Translation look-aside buffer (TLB)



Effective memory access time

- TLB lookup: ε
- Memory cycle duration: t
- TLB hit ratio: α
- Effective access time (EAT):

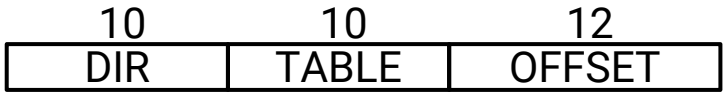
$$EAT = (t+\varepsilon)\alpha + (2t+\varepsilon)(1-\alpha) = (2 - \alpha)t + \varepsilon$$

Hierarchical page tables

x86:

- Addresses have 32 bits

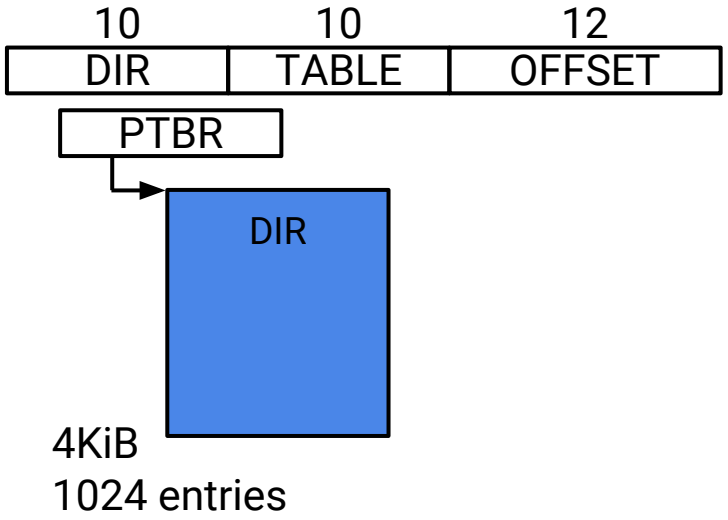
Hierarchical page tables



x86:

- Addresses have 32 bits

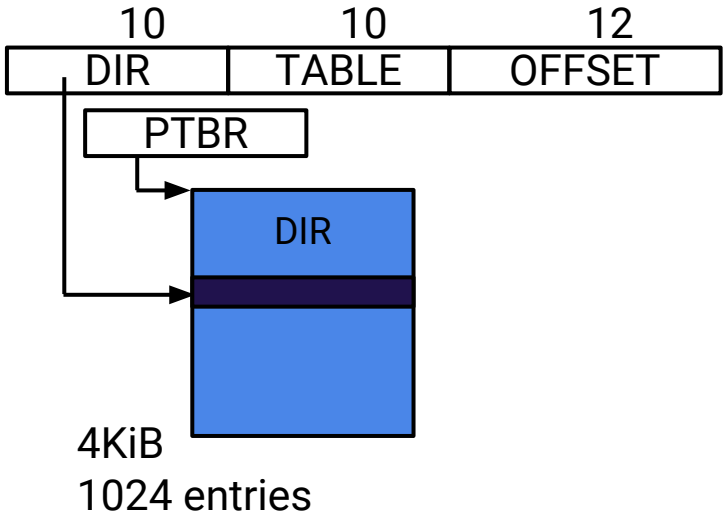
Hierarchical page tables



x86:

- Addresses have 32 bits
- Frames and pages are 4KiB

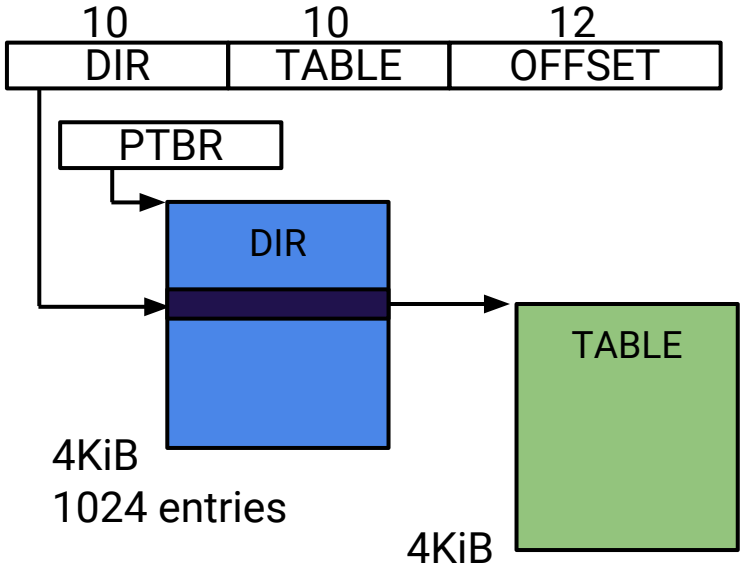
Hierarchical page tables



x86:

- Addresses have 32 bits
- Frames and pages are 4KiB

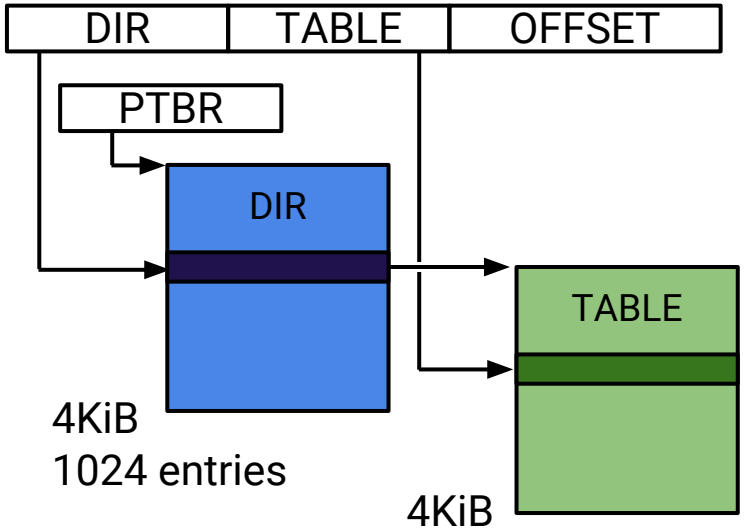
Hierarchical page tables



x86:

- Addresses have 32 bits
- Frames and pages are 4KiB
- Each page table entry has 32bits

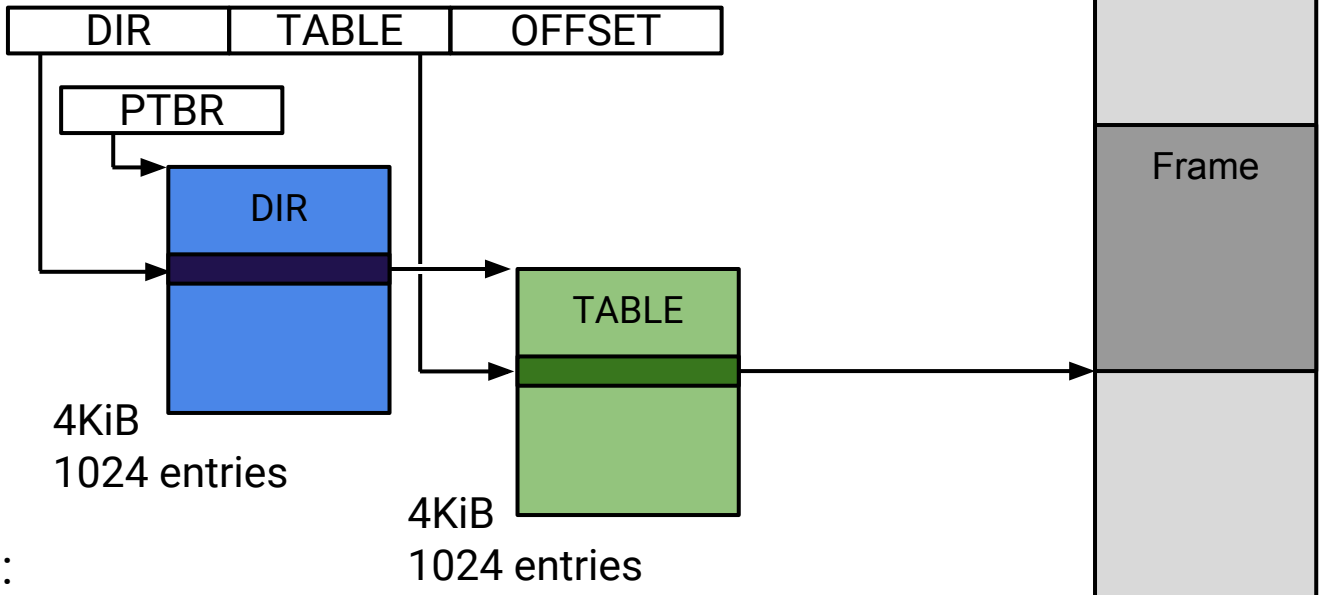
Hierarchical page tables



x86:

- Addresses have 32 bits
- Frames and pages are 4KiB
- Each page table entry has 32bits

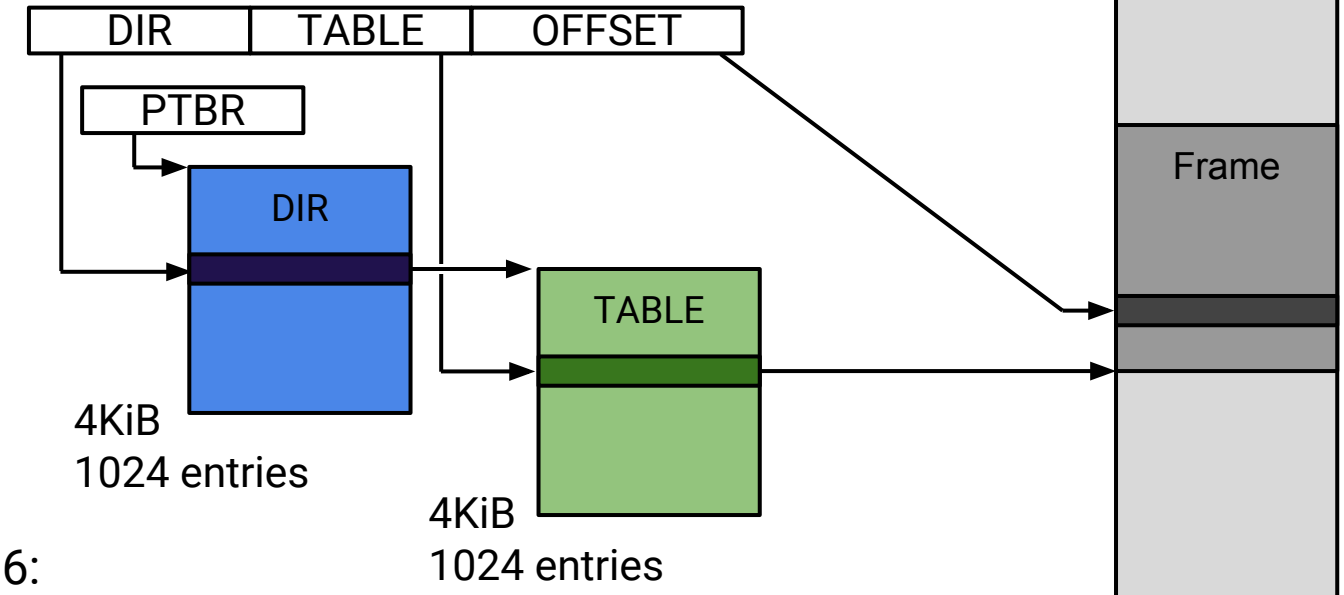
Hierarchical page tables



x86:

- Addresses have 32 bits
- Frames and pages are 4KiB
- Each page table entry has 32bits

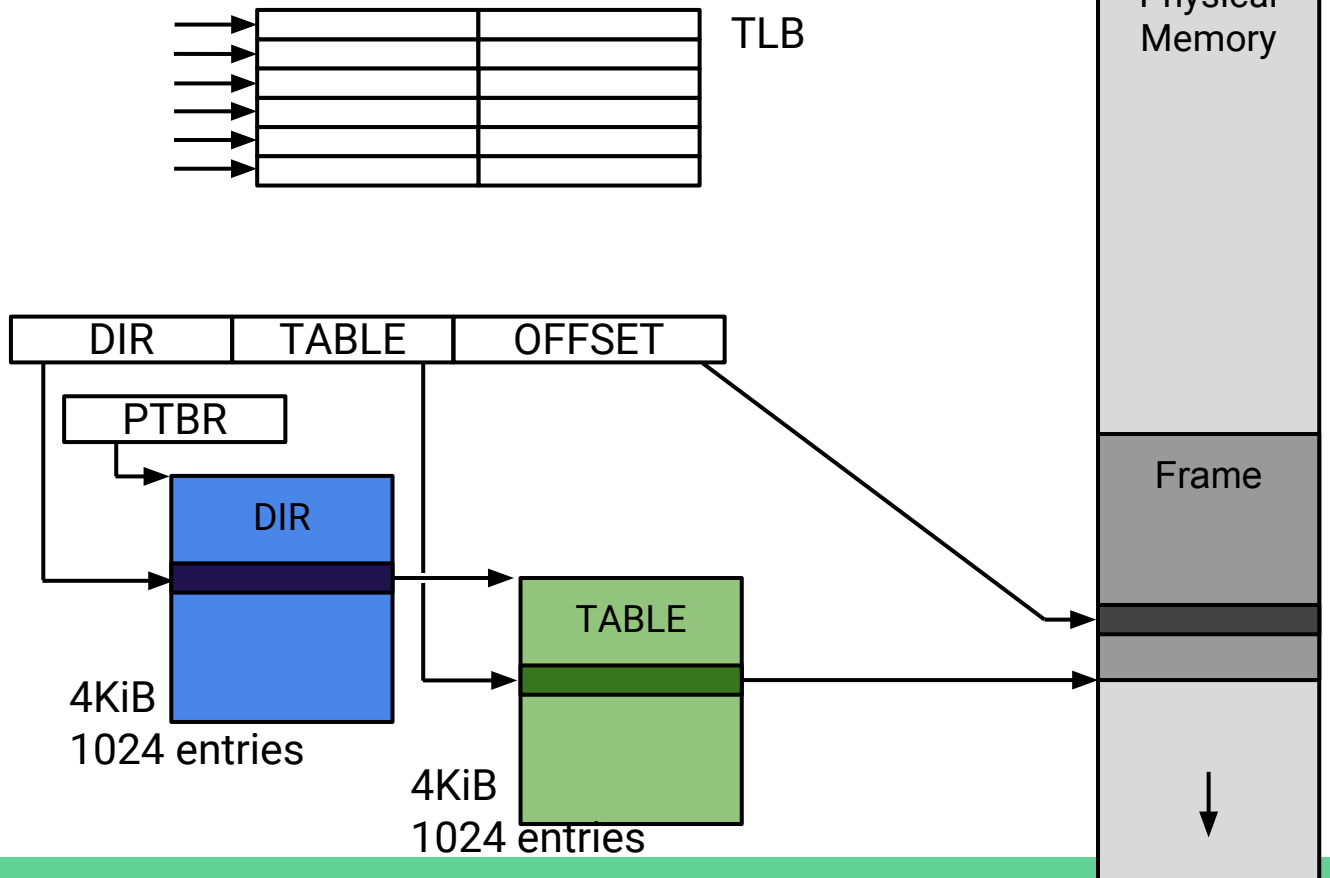
Hierarchical page tables



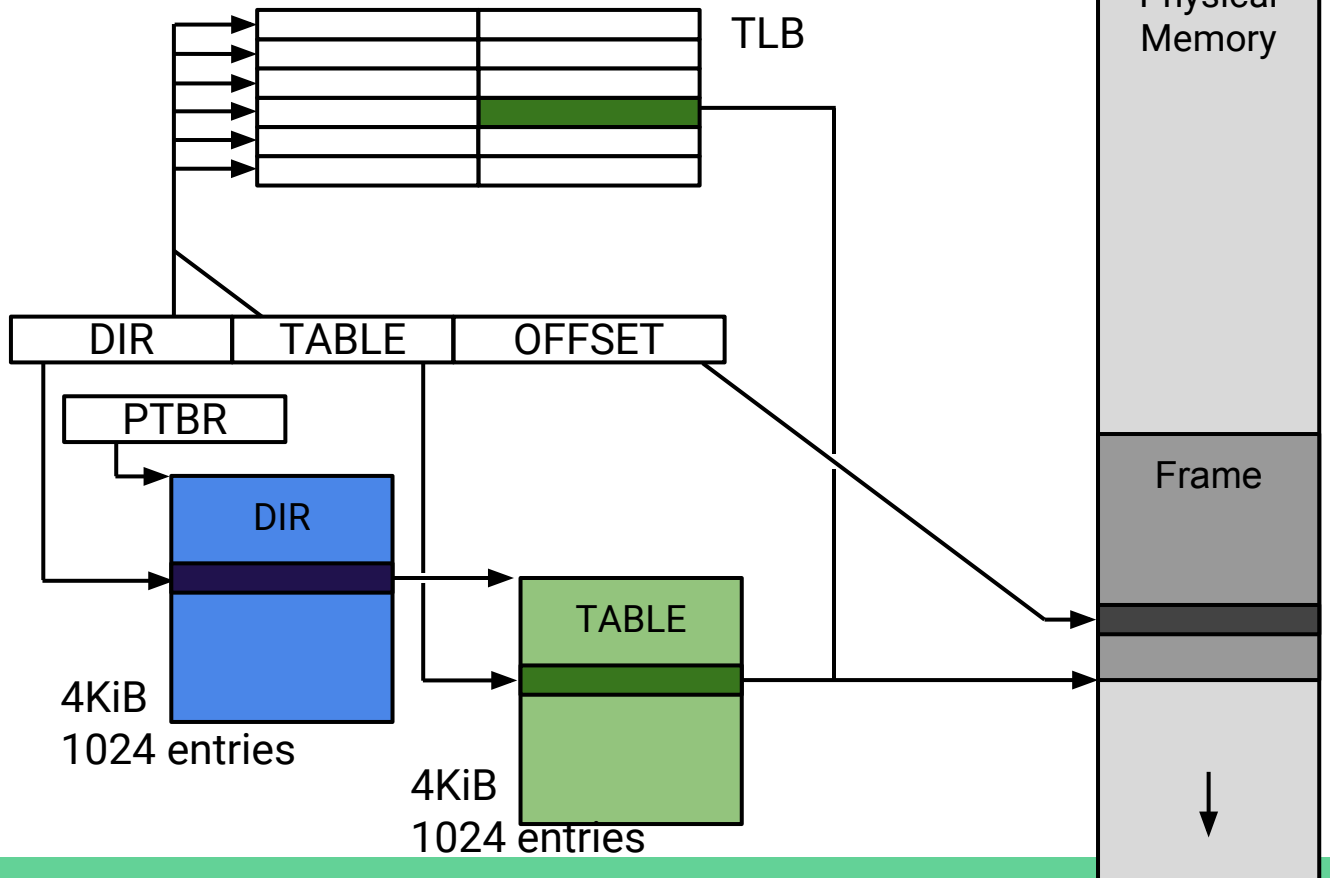
x86:

- Addresses have 32 bits
- Frames and pages are 4KiB
- Each page table entry has 32bits

Hierarchical page tables with TLB



Hierarchical page tables with TLB



Como que tudo isso ajuda a velocidade?

Como que tudo isso ajuda a velocidade?

- Mais hierarquia leva para páginas menores
- TLB com mais entradas
- + Cache Hits

Tudo muito bacana para máquinas de 32 bits

Tamanho de Páginas (registradores de 32 bits)

- 4KB de de tamanho geralmente
  - Offset de tamanho 2 ** 12 bits

- Quanto resta?

Tamanho de Páginas (registradores de 32 bits)

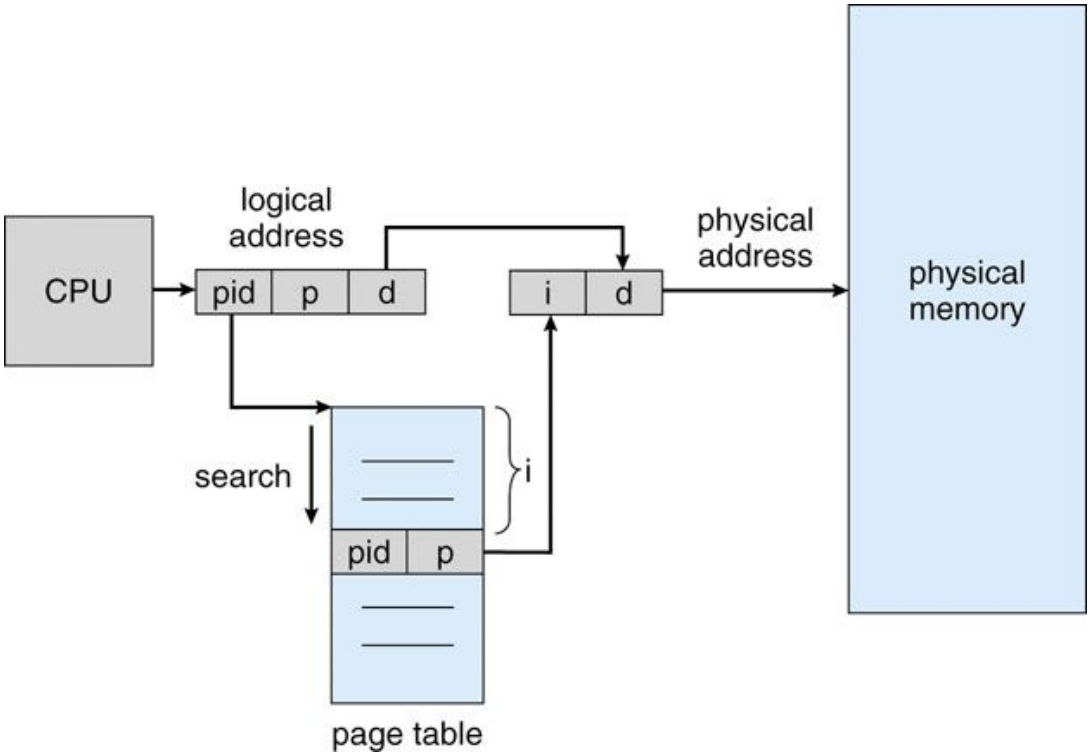
- 4KB de de tamanho geralmente
  - Offset de tamanho 2 ** 12 bits

- Quanto resta?
  - 2 ** 52
  - Assumindo 4 bytes de dados na tabela de página
  - + de 20 Peta Bytes por processo

Inverted page tables

- Maps frames (physical) to pages (logical)
- One entry in the table for each frame
  - Table size depends on physical memory size
- Inverted page table needs to identify the process
  - Two processes can have the same virtual addresses

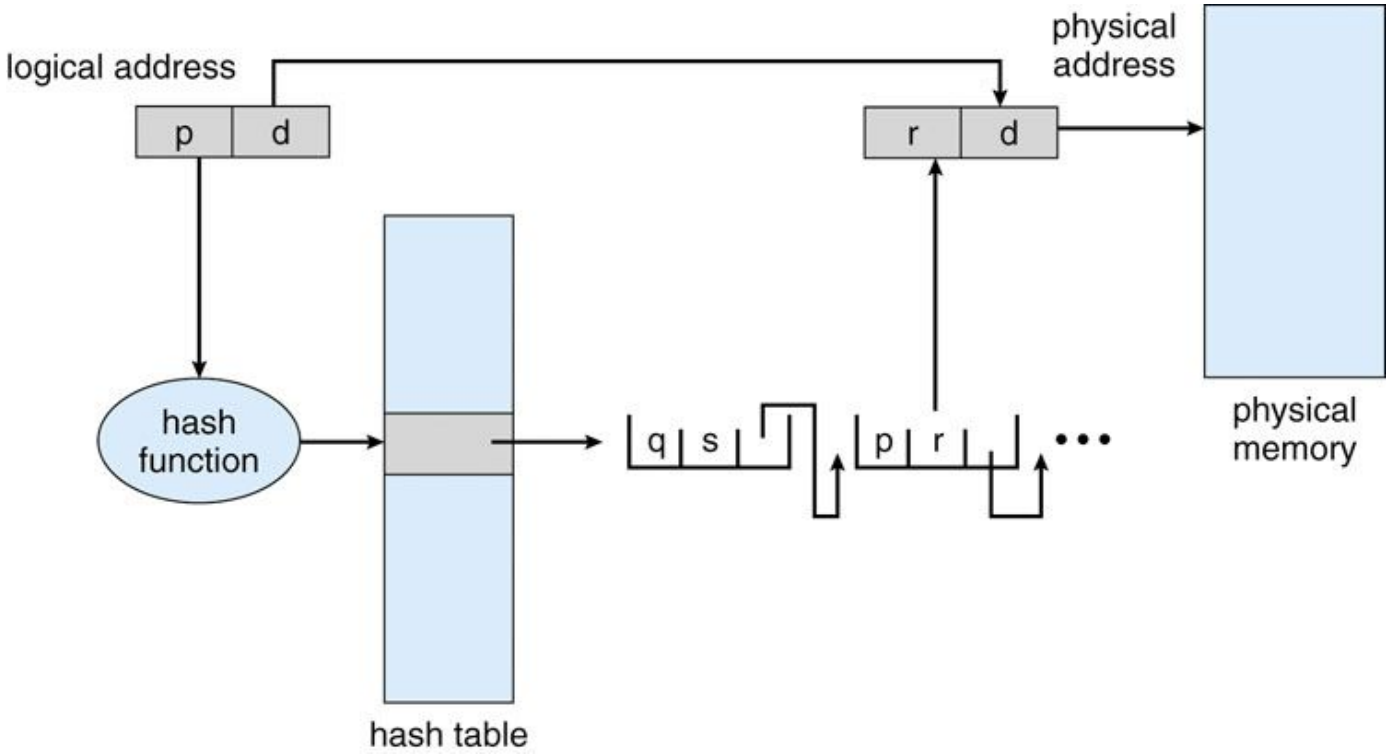
Inverted page tables



Inverted page tables

- Maps frames (physical) to pages (logical)
- One entry in the table for each frame
  - Table size depends on physical memory size
- Inverted page table needs to identify the process
  - Two processes can have the same virtual addresses
- Needs to search the table
  - Usually implemented with a hash
  - Need mechanism to handle collisions

Hash Page Table



+ Problemas?

+ Problemas?

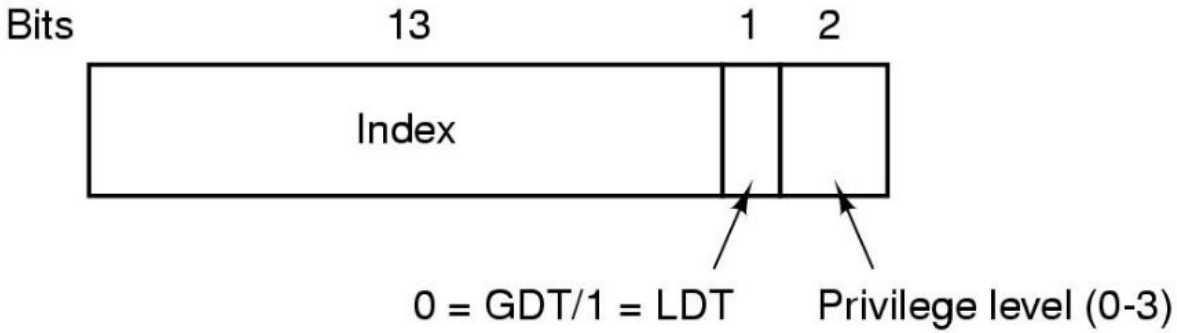
- Só ajudamos o TLB
- Ainda estamos assumindo que tudo cabe em memória

Outras Ideias Interessante

- Combinar Segmentação com Paginação
- Fatiar mais ainda a memória

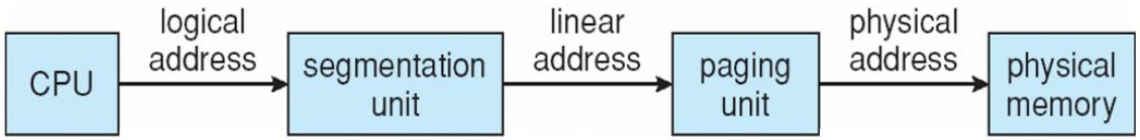
Pentium Intel

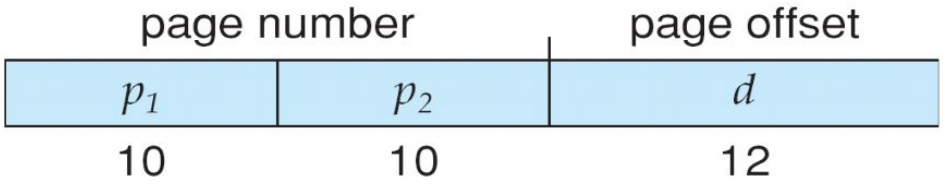
- TLB bem perto da CPU
  - Maior velocidade
- GDT: Global Descriptor Table
  - Segmentos Globals
- LDT: Local Descriptor Table
  - Segmentos Locais



Pentium Intel

- Paginação de 2 níveis por segmento





Aonde estamos...

- Silberschatz
  - Chapt 7
- Tanembaum
  - Chapt 3
- OSTEP
  - Segunda parte de virtualização