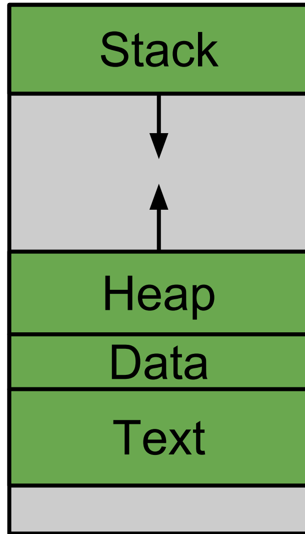
SO: Segmentação/Paginação

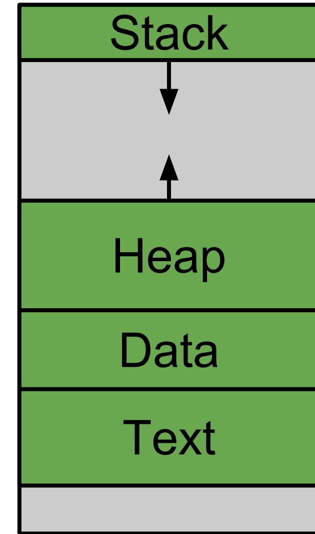
Sistemas Operacionais
2017-1
Flavio Figueiredo (http://flaviovdf.github.io)

Segmentação

Segmentação

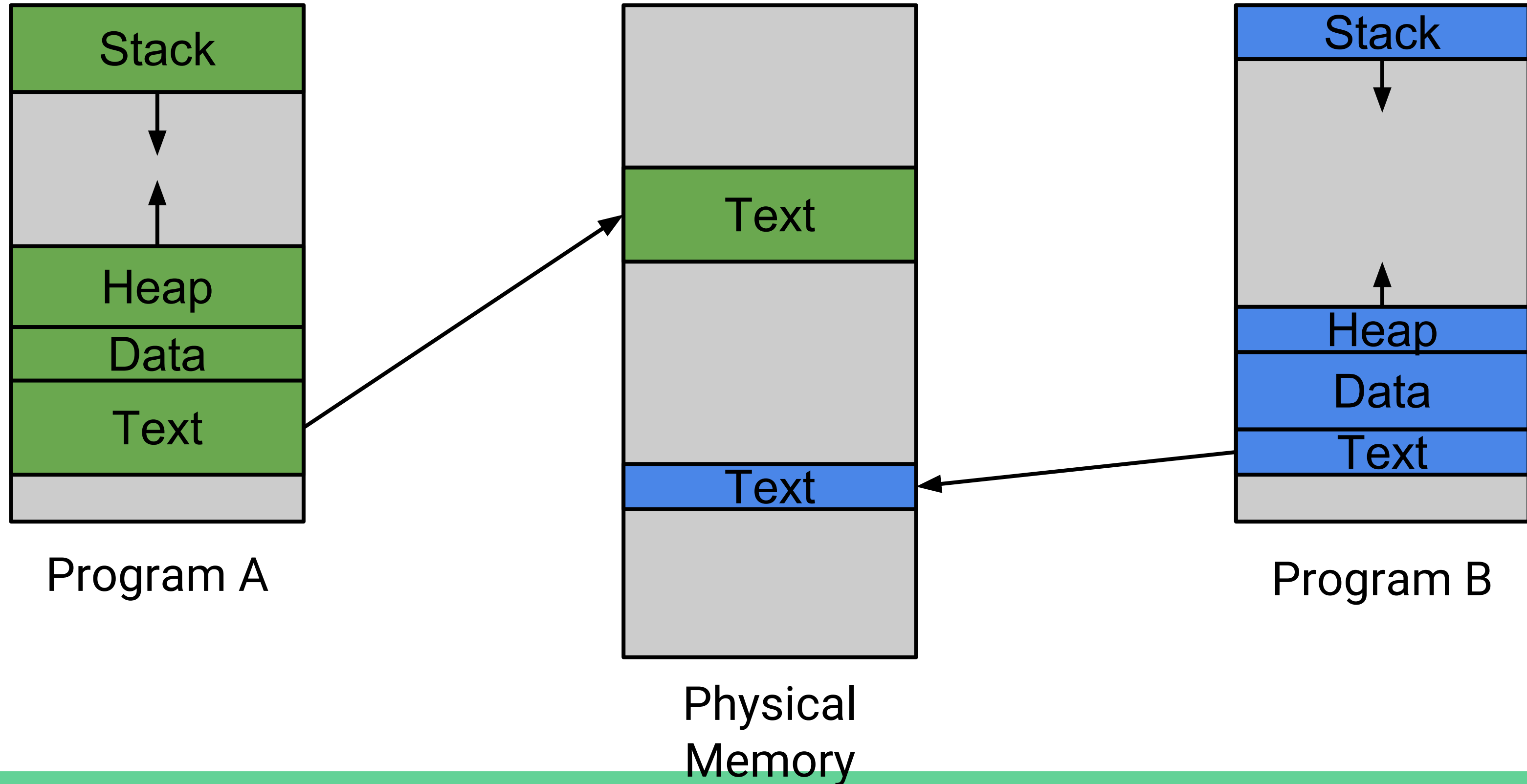


Program A

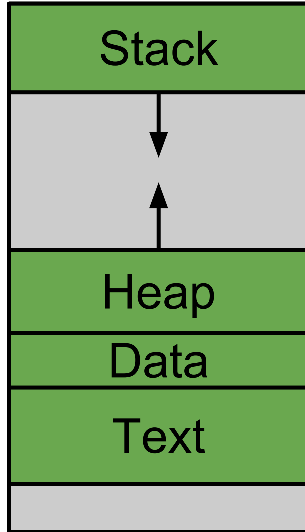


Program B

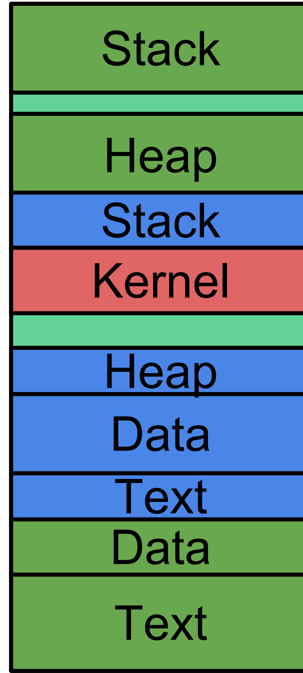
Segmentação



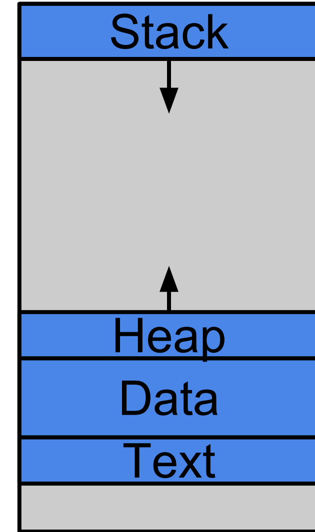
Segmentação



Program A



Physical Memory



Program B

Como implementar?

Como implementar?

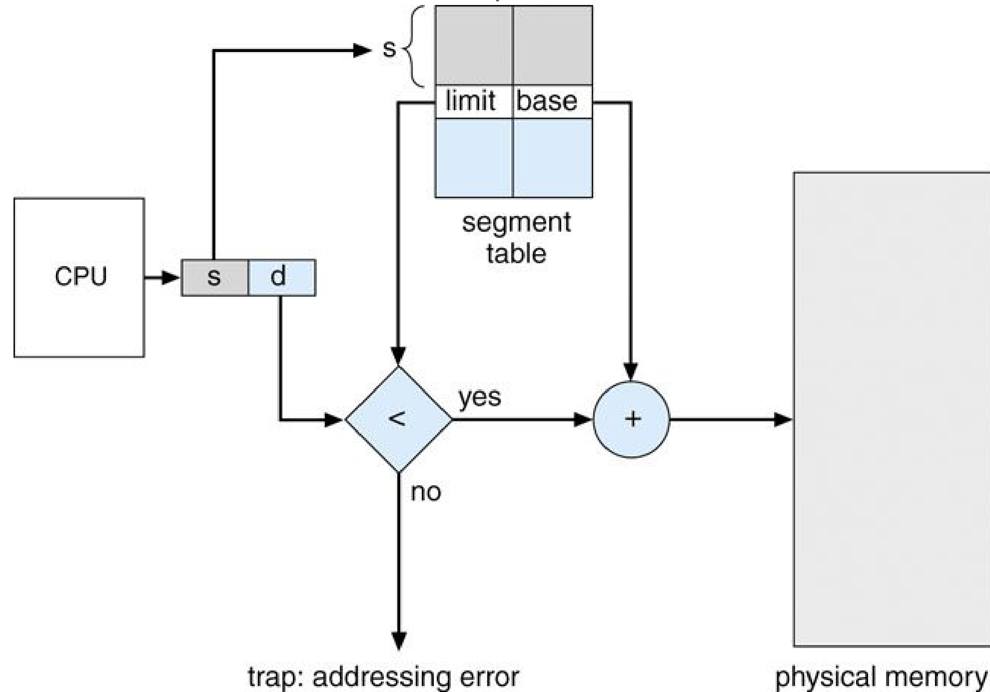
- Particionar a memória em segmentos
  - Code, heap, stack ...
- Lidamos com a fragmentação interna
- Ainda temos a externa

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

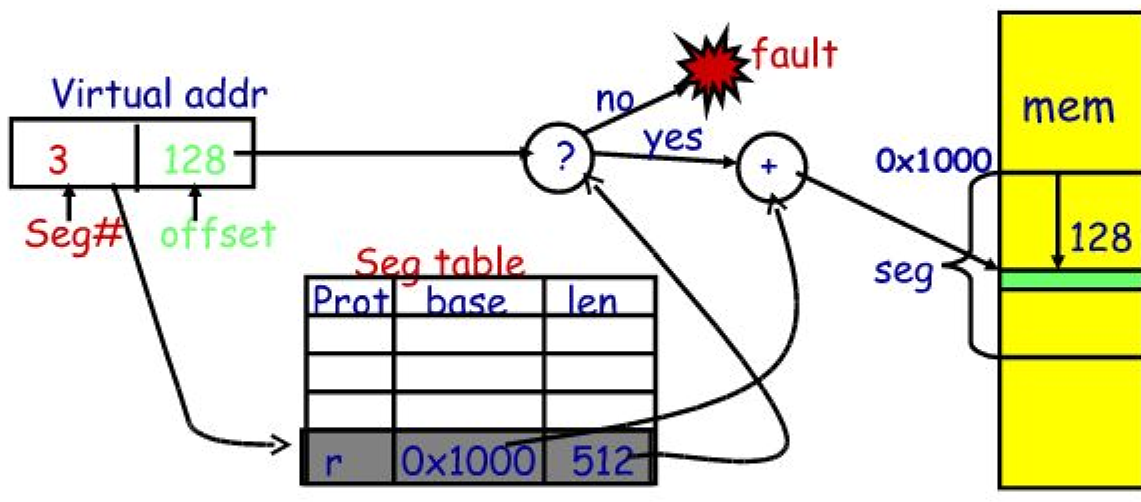
- Proteção
  - Campos de Read/Write/Execute
  - Data
    - Read/Execute
  - Heap
    - Read/Write

- 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



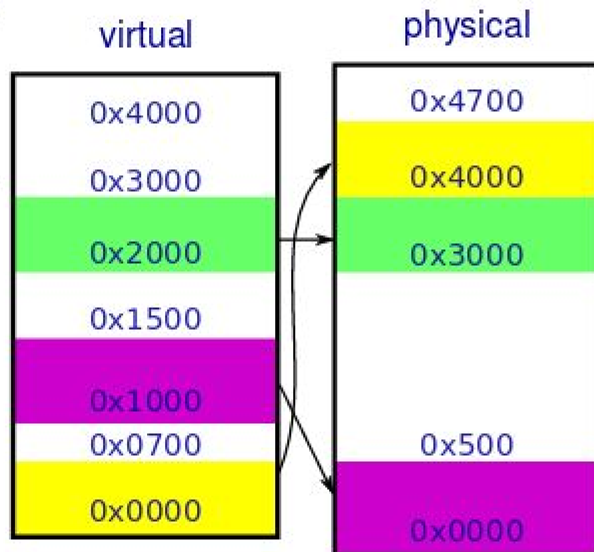
Em Bits

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

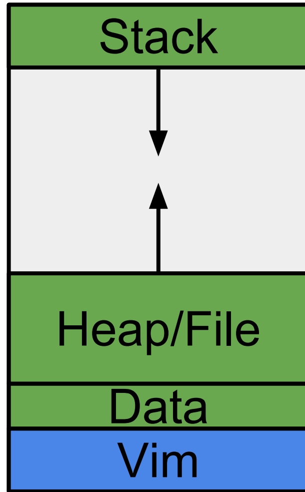
```
1  // get top 2 bits of 14-bit VA
2  Segment = (VirtualAddress & SEG_MASK) >> SEG_SHIFT
3  // now get offset
4  Offset  = VirtualAddress & OFFSET_MASK
5  if (Offset >= Bounds[Segment])
6      RaiseException(PROTECTION_FAULT)
7  else
8      PhysAddr = Base[Segment] + Offset
9      Register = AccessMemory(PhysAddr)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

Em Bits

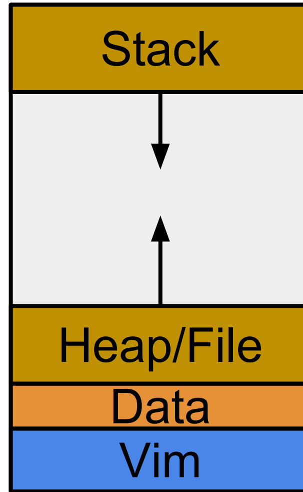
Seg	base	bounds	rw
0	0x4000	0x6ff	10
1	0x0000	0x4ff	11
2	0x3000	0xfff	11
3			00



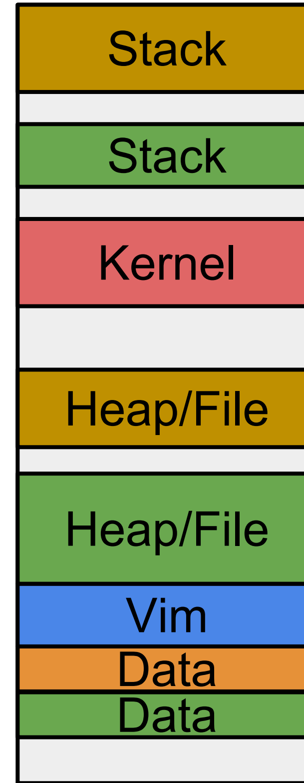
Compartilhando Memória



Vim 1



Vim 2



Physical Memory

Considerações

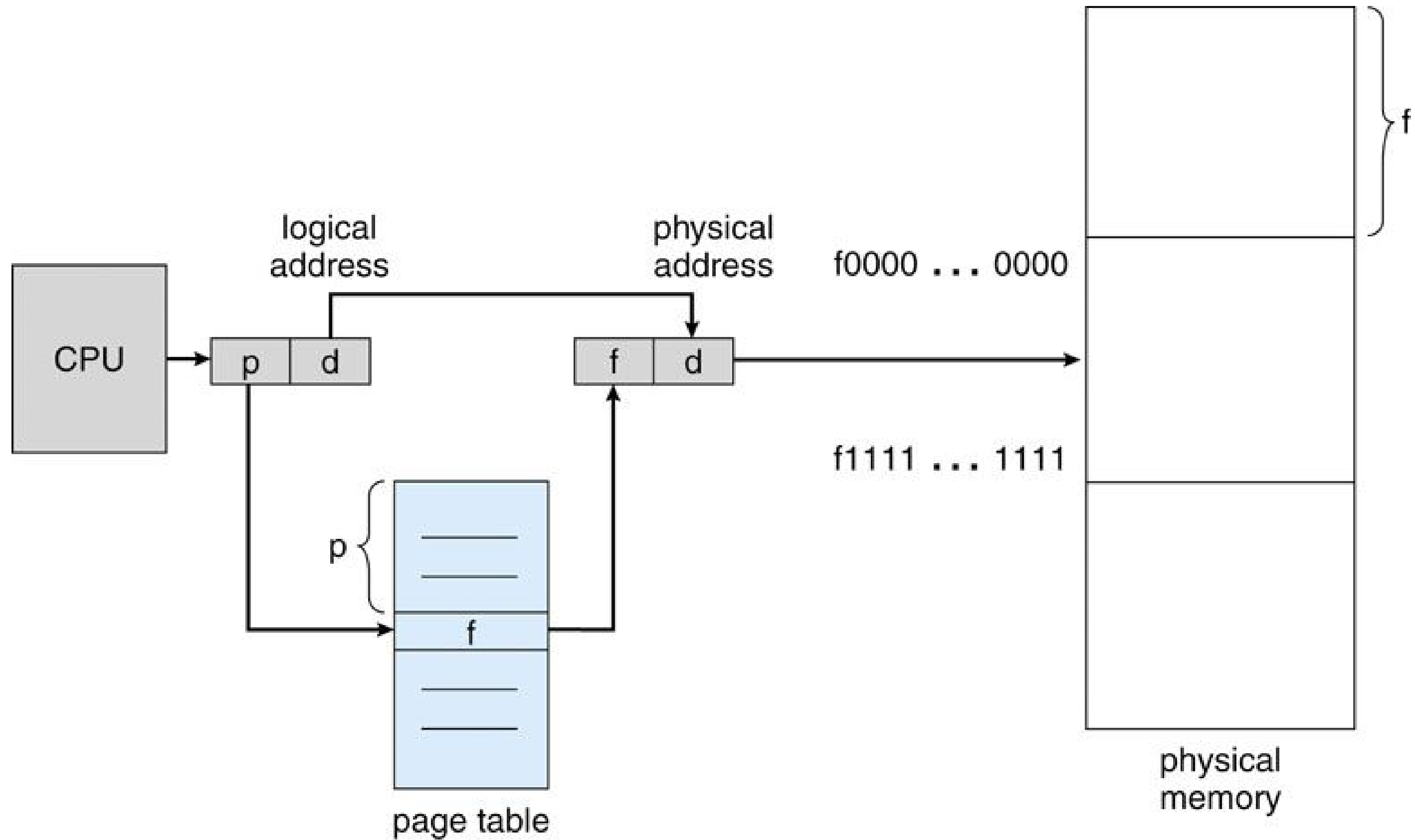
- Segmentação pura é bem simples de implementar
- SO precisa apenas gerenciar o início e fim de cada segmento
    - Hardware faz a tradução
- Ainda temos problemas de fragmentação
- Ainda temos problemas se um segmento não cabe em memória
- Vamos fatiar mais ainda os programas

Paginação

Paginação

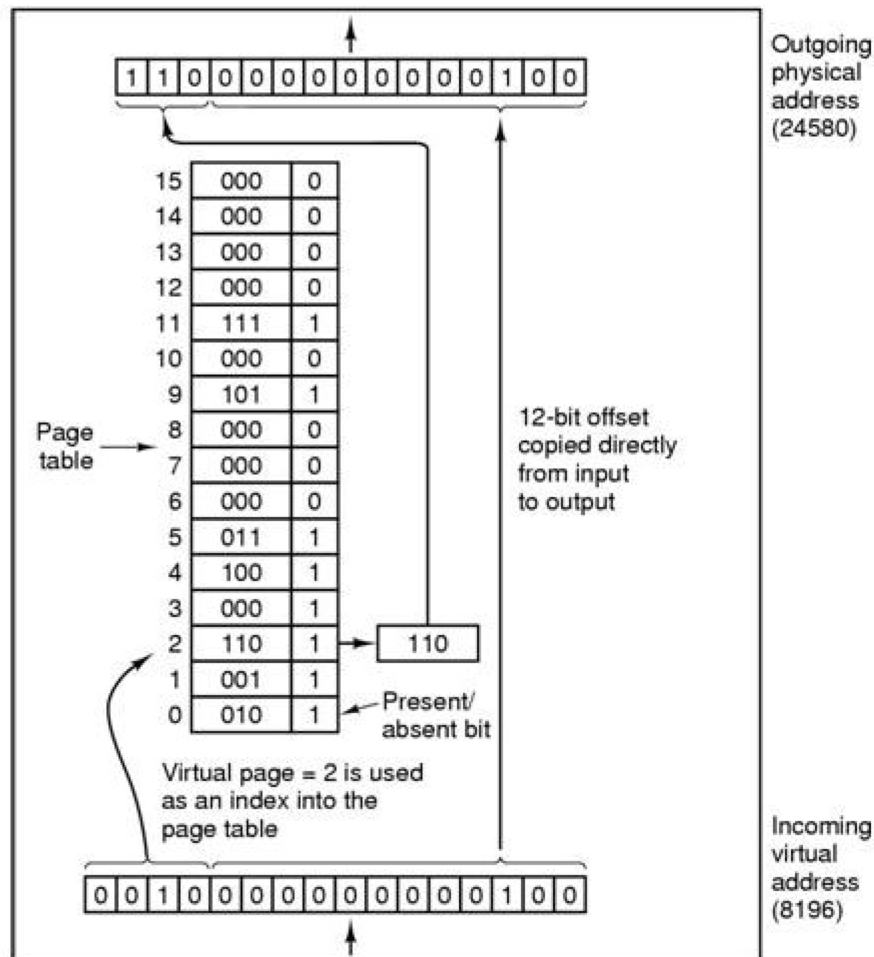
- “Fatiamento” da memória em *frames* de tamanhos iguais
  - 4KB ou 4MB
- Páginas virtuais
  - Um programa com **n** páginas faz uso de **n** frames
- Memória gerencia as páginas não utilizadas

Hardware



Hardware

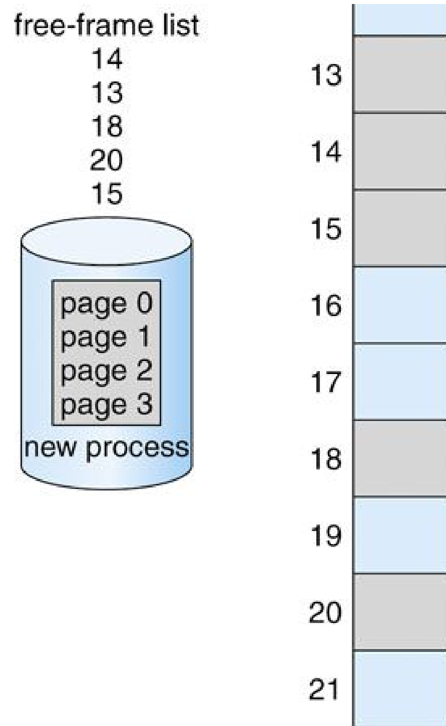
- Primeiros bits para identificar a página
- Seguintes identificam o endereço
- Tradução em hardware
- Tabelas de página na memória
  - Jogar fora alguns bits
  - Substituir por outros

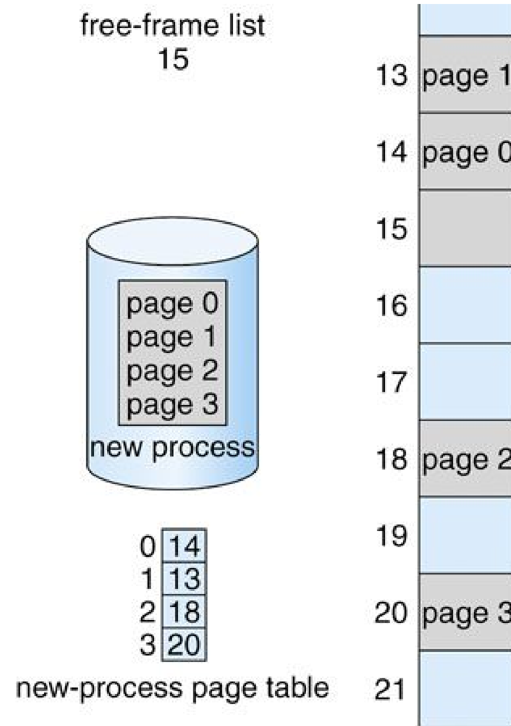


Tabelas de Página

```
1   // Extract the VPN from the virtual address
2   VPN = (VirtualAddress & VPN_MASK) >> SHIFT
3
4   // Form the address of the page-table entry (PTE)
5   PTEAddr = PTBR + (VPN * sizeof(PTE))
6
7   // Fetch the PTE
8   PTE = AccessMemory(PTEAddr)
9
10  // Check if process can access the page
11  if (PTE.Valid == False)
12      RaiseException(SEGMENTATION_FAULT)
13  else if (CanAccess(PTE.ProtectBits) == False)
14      RaiseException(PROTECTION_FAULT)
15  else
16      // Access is OK: form physical address and fetch it
17      offset   = VirtualAddress & OFFSET_MASK
18      PhysAddr = (PTE.PFN << PFN_SHIFT) | offset
19      Register = AccessMemory(PhysAddr)
```

Novos Processos





x86 page entry

31	12	11	10	9	8	7	6	5	4	3	2	1	0
4KiB-aligned Page Address		Avail			G	S	W	A	D	T	U	R	P

- **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

31		12	11	10	9	8	7	6	5	4	3	2	1	0
4KiB-aligned Page Address			Avail			G	S	W	A	D	T	U	R	P

- **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

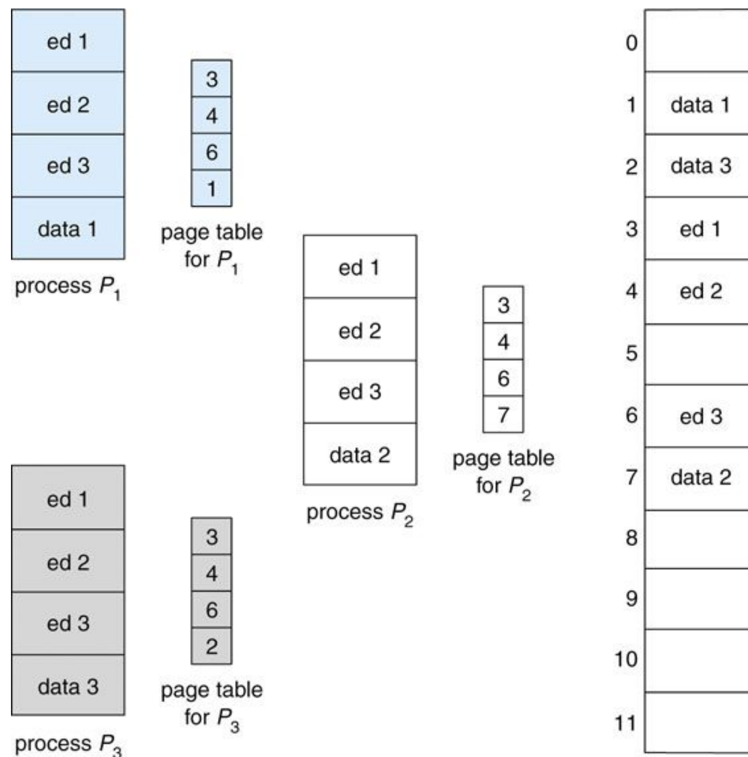
- Similar a segmentação

- Mais controle
  - Pequenos pedaços

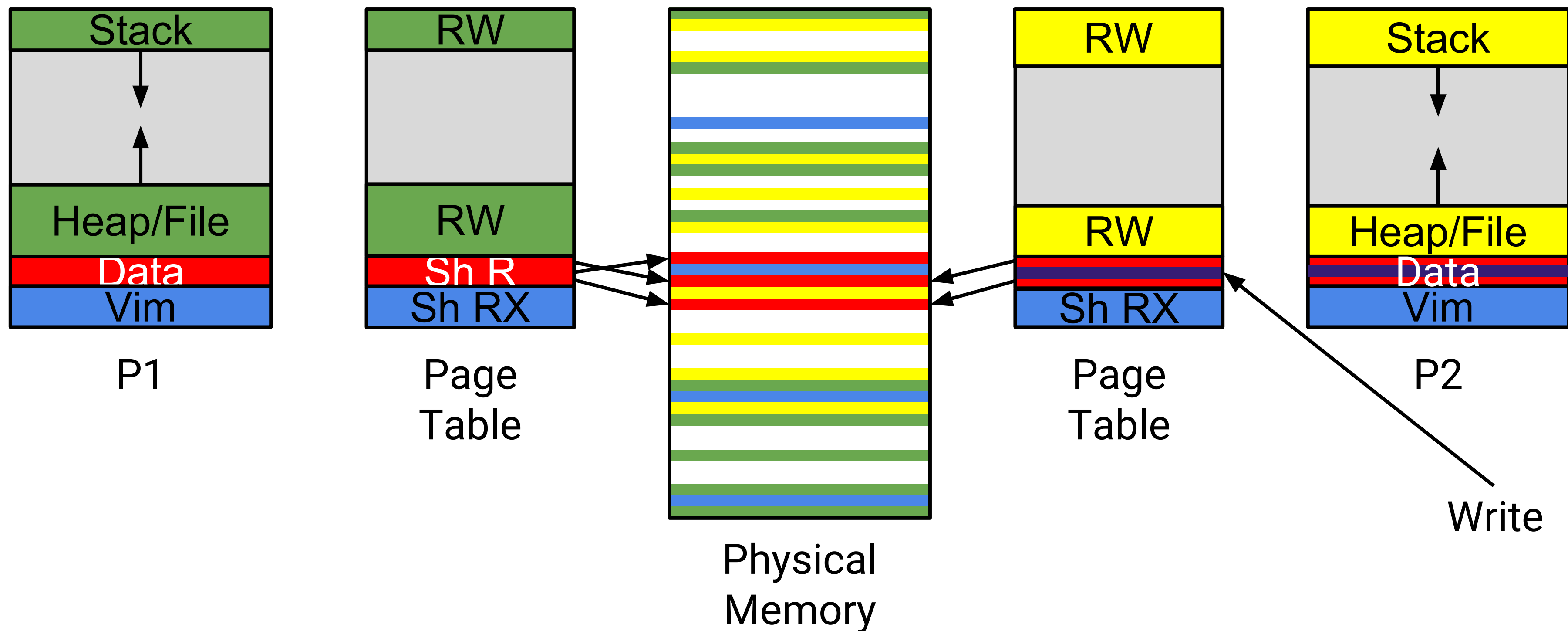
  - Lembre-se do fork/exec

Compartilhamento de Páginas

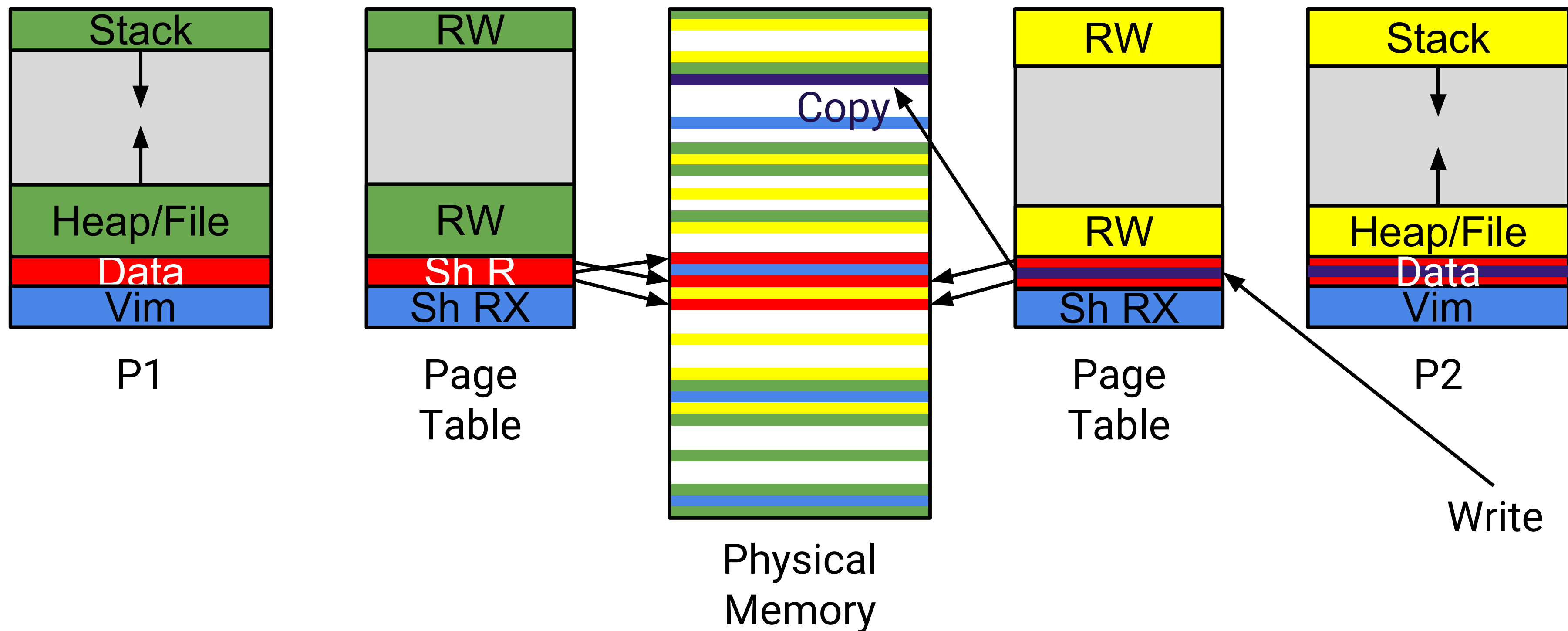
- Cada página com número de referências
  - Processos
- Se zerar
  - A página pode ser liberada
- Páginas compartilhadas iniciam como read-only
  - Viram write apenas na primeira escrita
  - Page-fault



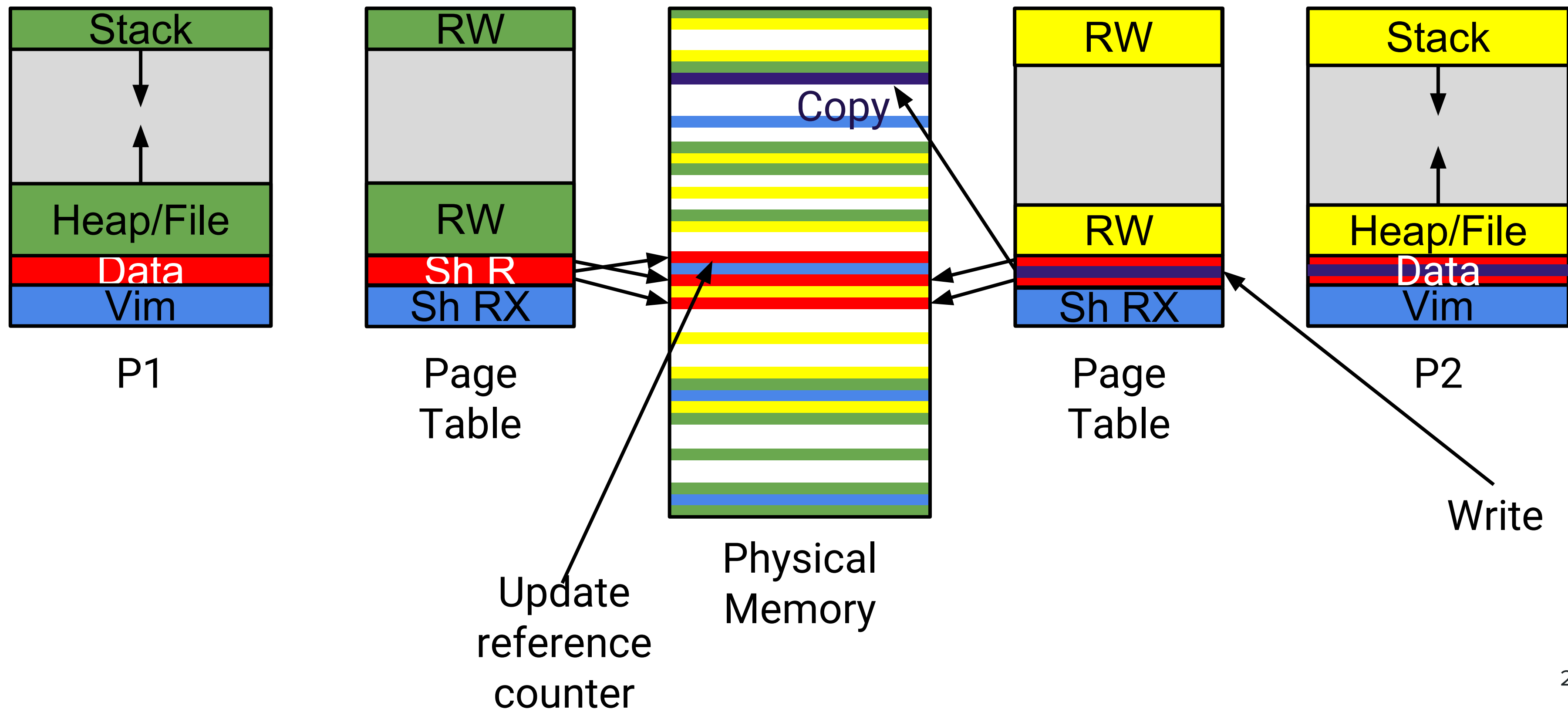
Copy-on-write



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

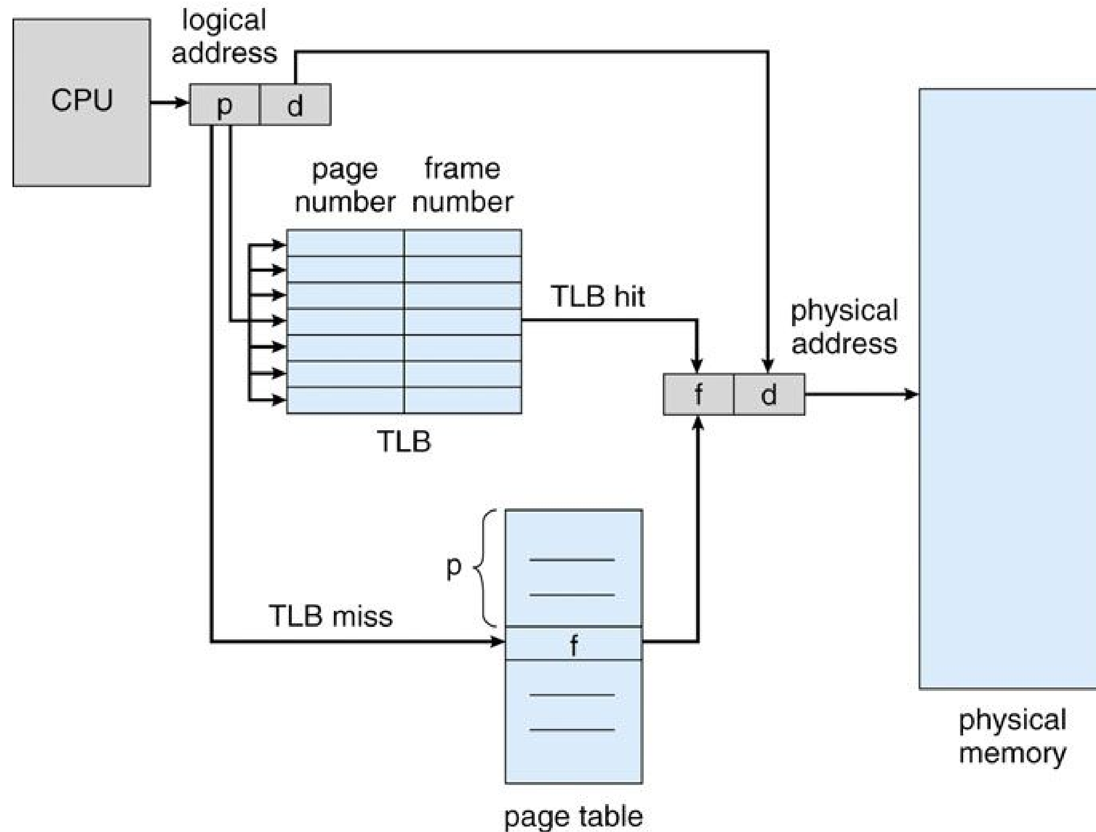
- Paginação é lenta

- Tabelas de página residem na memória
  - Se mal feito, ocupam muito espaço
  - Podemos mitigar com boas estruturas de dados
  - Operações feitas por software

- Tradução final ainda fica no hardware

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):

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

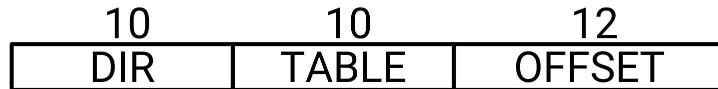
Hierarchical page tables

32 bits

x86:

- Addresses have 32 bits

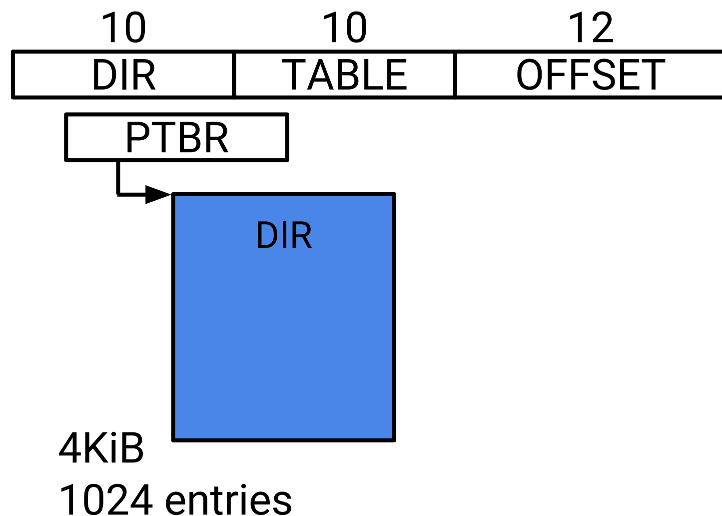
Hierarchical page tables



x86:

- Addresses have 32 bits

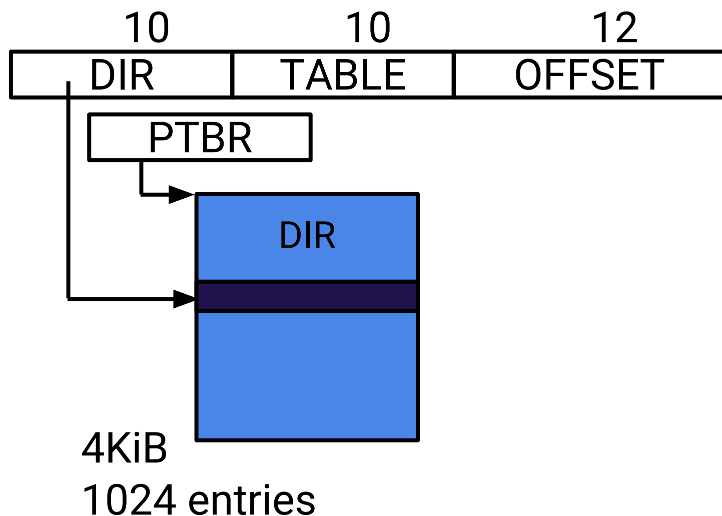
Hierarchical page tables



x86:

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

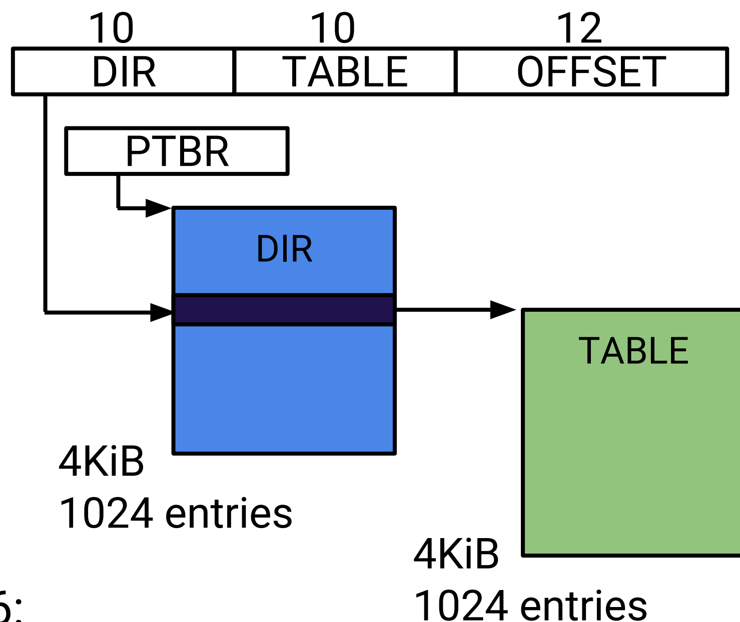
Hierarchical page tables



x86:

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

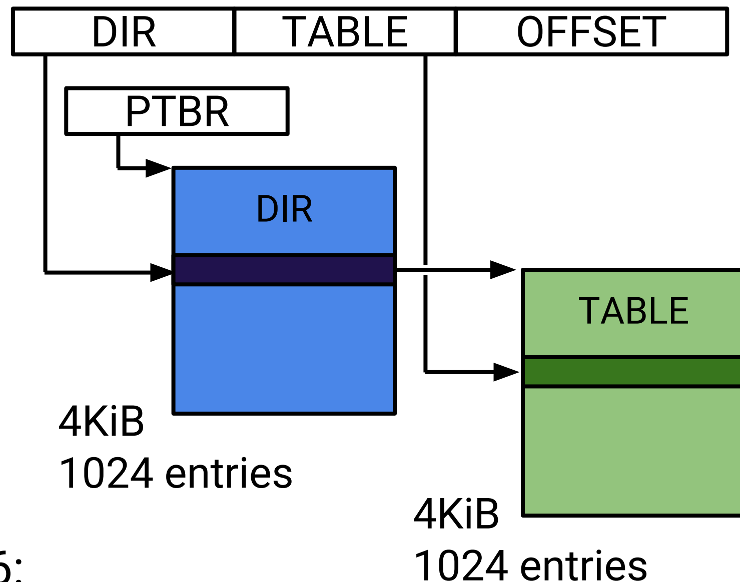
Hierarchical page tables



x86:

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

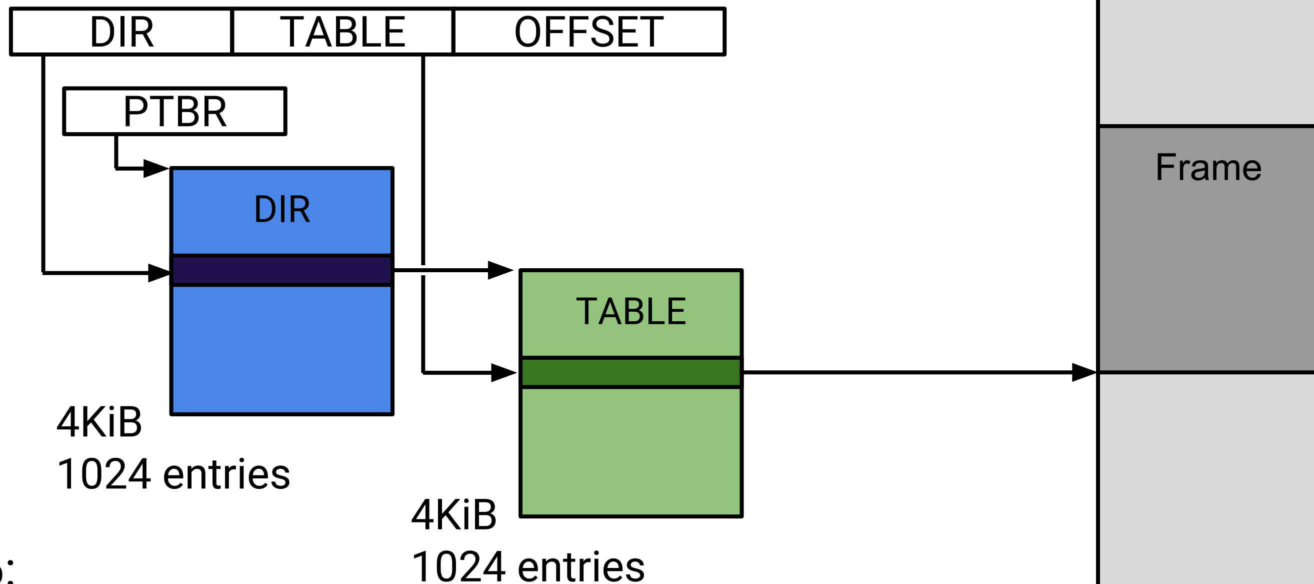
Hierarchical page tables



x86:

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

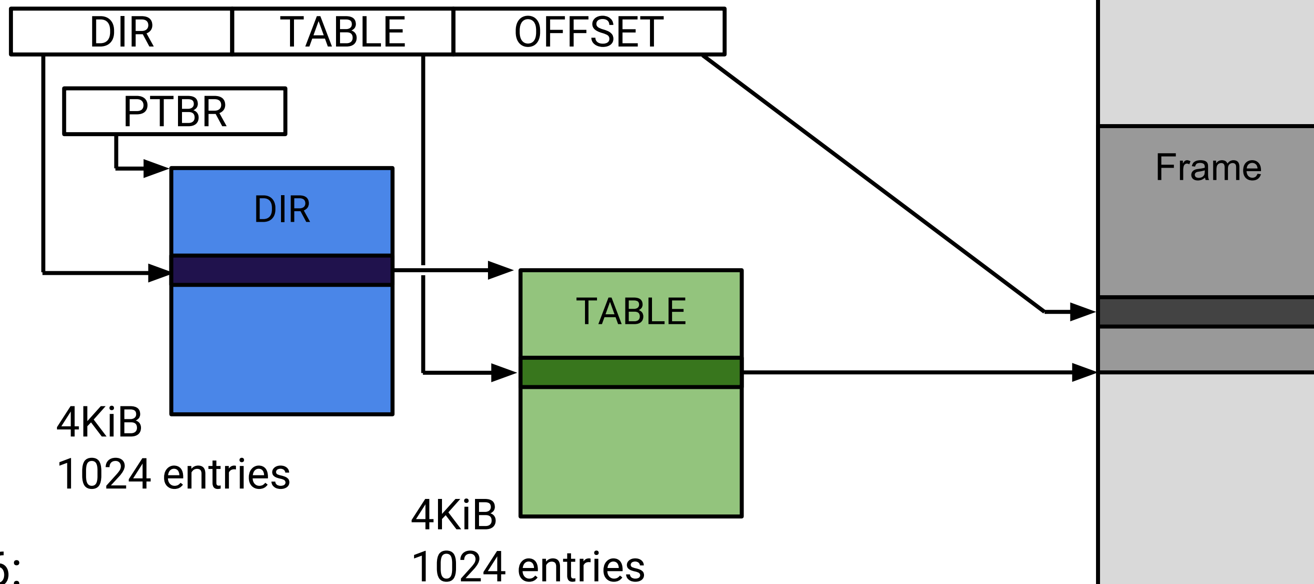
Hierarchical page tables



x86:

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

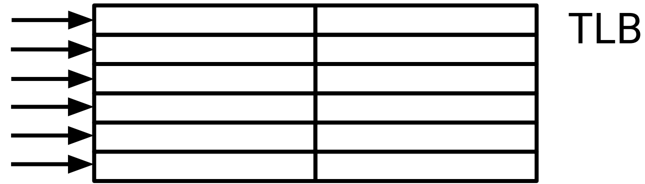
Hierarchical page tables

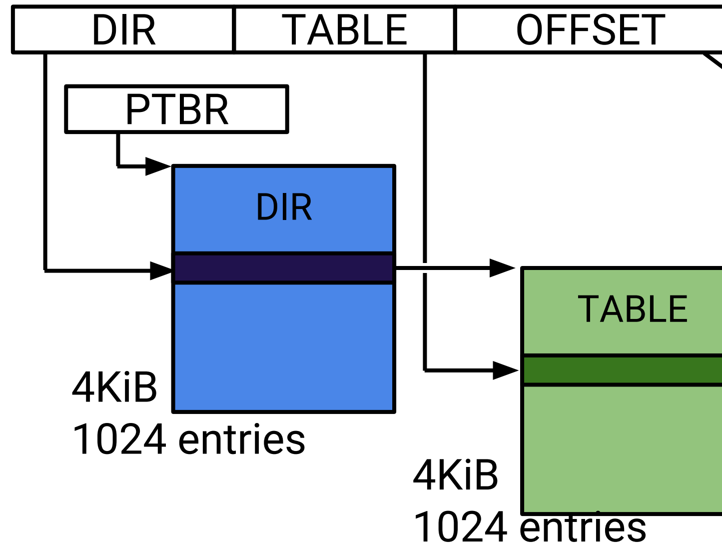


x86:

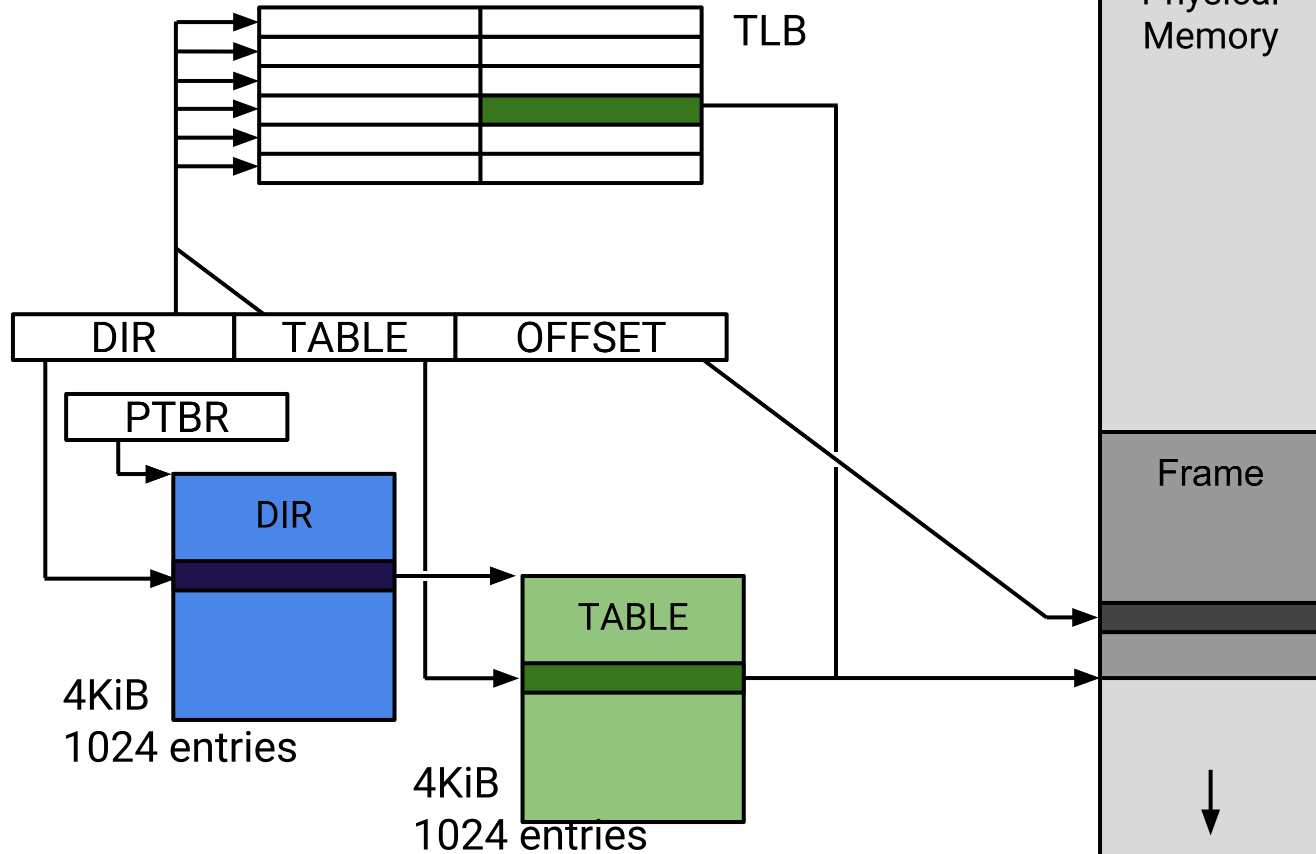
- Addresses have 32 bits
- Frames and pages are 4KiB
- Each page table entry has 32bits

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 endereços menores

- TLB com mais entradas
  - Image um TLB com tamanho fixo, afinal é hardware
  - Quando mais páginas couberem lá melhor
  - Para cabe + páginas, reduzimos o tamanho de endereçamento das mesmas
    - Ao invés de 20 bits, 10 bits
    - Movemos o overhead para o software

- Além disto, podemos alocar tabelas de páginas apenas dos diretórios em uso
  - Poupamos espaço em software

- + 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

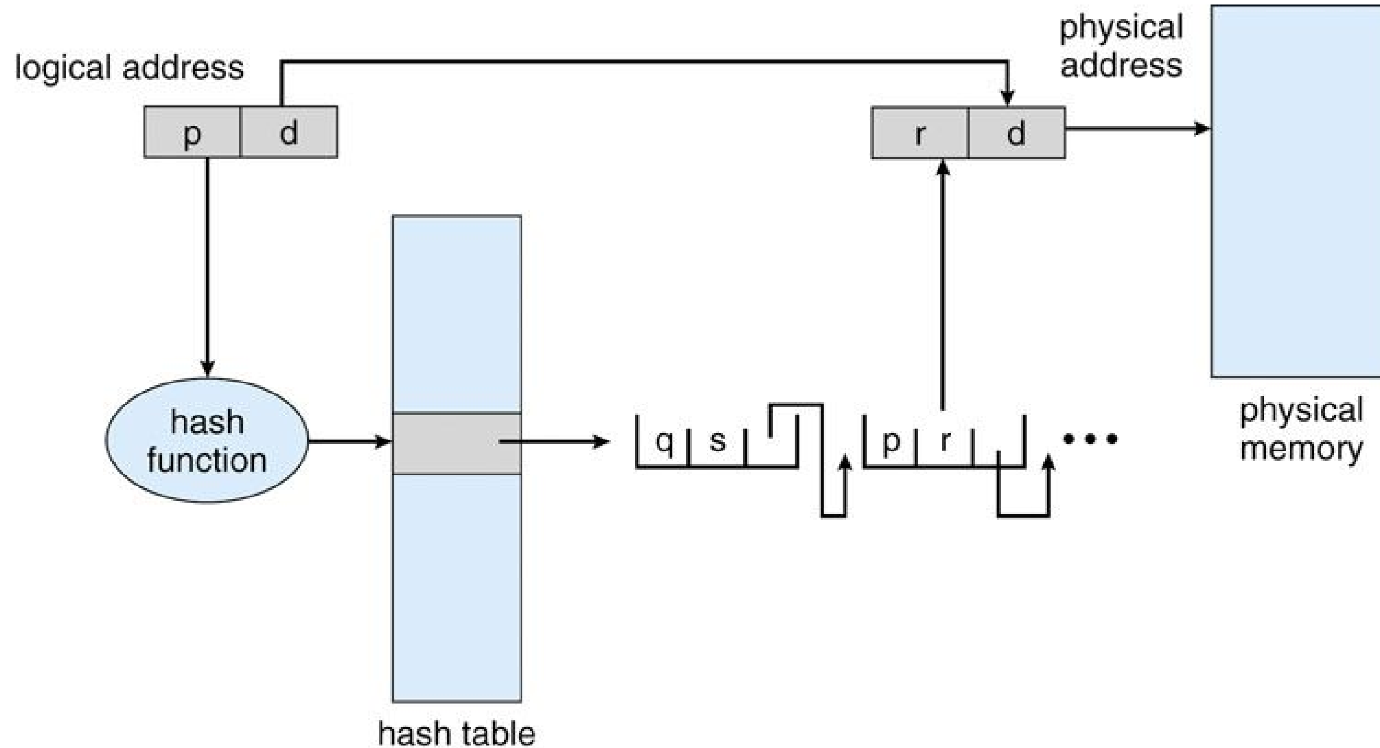
Tabelas de Página Existem em Software

- Antigamente a tabela de página era fixa, sempre usava os 2^20 bits
  - O hardware gerenciava o TLB
  - Precisava atualizar alguns bits da tabela
  - Alterava a memória diretamente, mais rápido
    - Um registrador indicava onde iniciava a tabela na memória
    - Assim o hardware poderia fazer mudanças na mesma

- A ideia acima fica inviável com 64 bits
- Poupar espaço com hash
  - Tudo em software

- Qualquer outra estrutura de dados pode ser usada

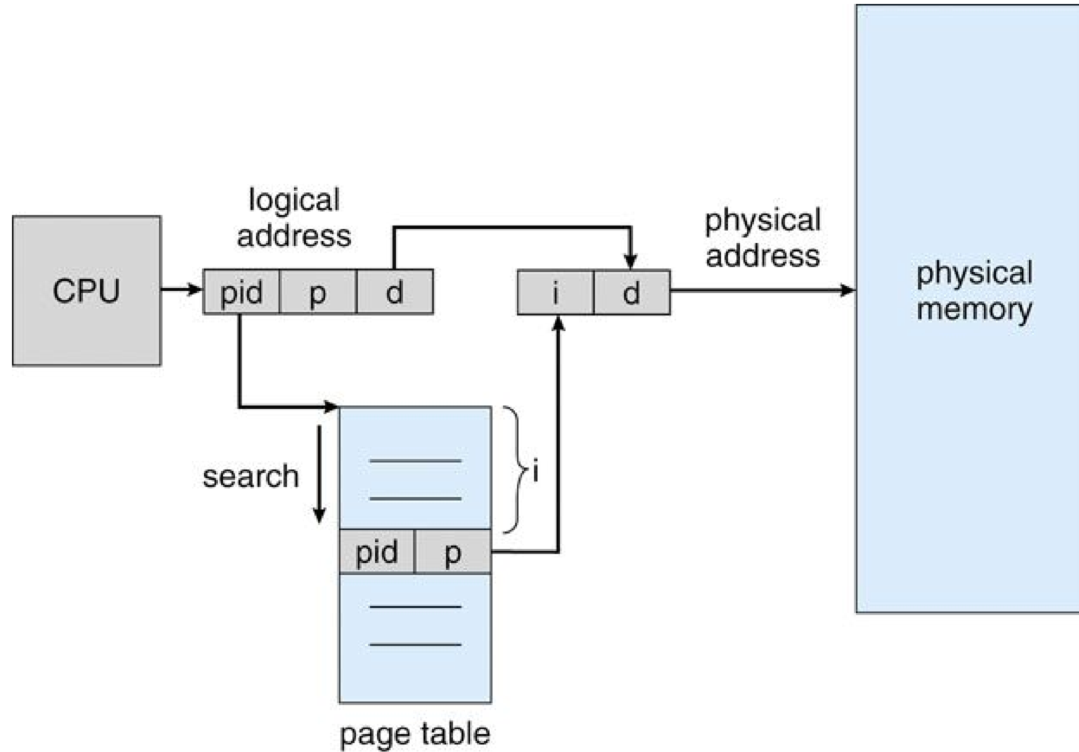
Tabela de Página com Hash



Tabelas de Páginas Invertida

- Tabela Hash
- Chaves são processos + endereço real da página
- Valores equivalem ao endereço virtual
- Não precisamos mais de uma tabela por processo
  - Um mapa para todos os processos do SO

Tabelas de Páginas Invertida



+ Problemas?

+ Problemas?

- Melhoramos a vida do SO em questão de memória
  - Hashes
  - Diretórios de Páginas

- Velocidade com o TLB
  - Ainda temos que gerenciar o mesmo de alguma forma
  - Um *miss gera um trap*
  - Como tratamos?

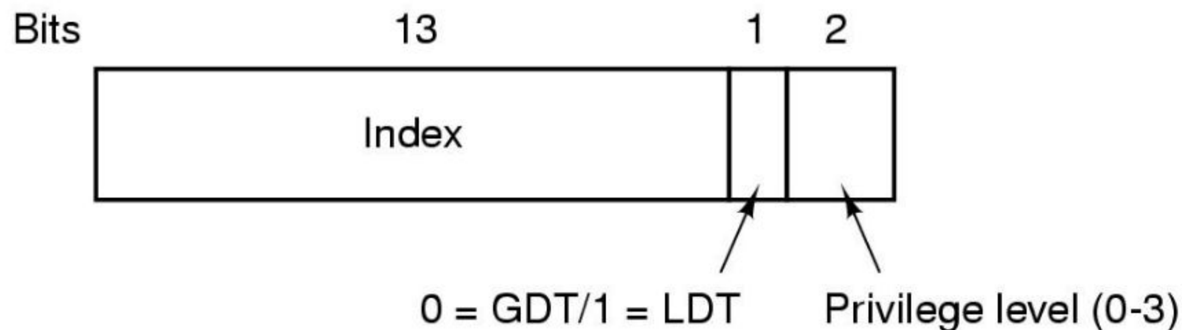
- Ainda estamos assumindo que tudo cabe em memória
  - Mesmo com a ajuda acima não será verdade sempre

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



Proteção

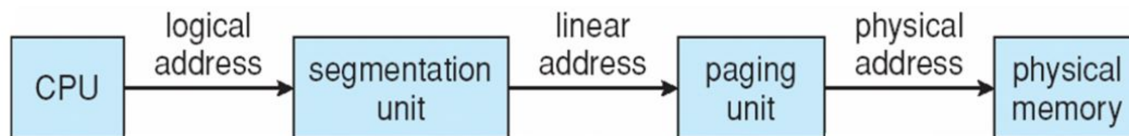
31		12	11	10	9	8	7	6	5	4	3	2	1	0
4KiB-aligned Page Address			Avail			G	S	W	A	D	T	U	R	P

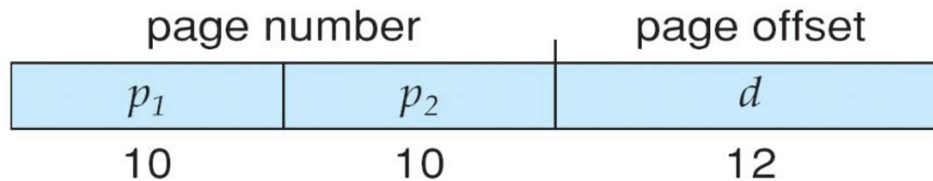
- **Available:** free for the OS to use. If P is unset, all bits are available.
- **G:** If 1, page is global and TLB will not update it.

- **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).

Pentium Intel

- Paginação de 2 níveis por segmento





Aonde estamos...

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