Hierarquia de memórias

José C. Cunha –DI- FCT/UNL

Necessidades de memória

suporte para a execução de programas e acesso aos dados pelo CPU, durante a execução dos programas (curtos períodos de tempo)

suporte para arquivo de ficheiros (longos períodos de tempo, memória não volátil)

Memória Virtual

esconder as limitações do espaço de endereços reais

simular uma memória dedicada a cada processo, com dimensão adequada e acesso rápido

Factores condicionantes

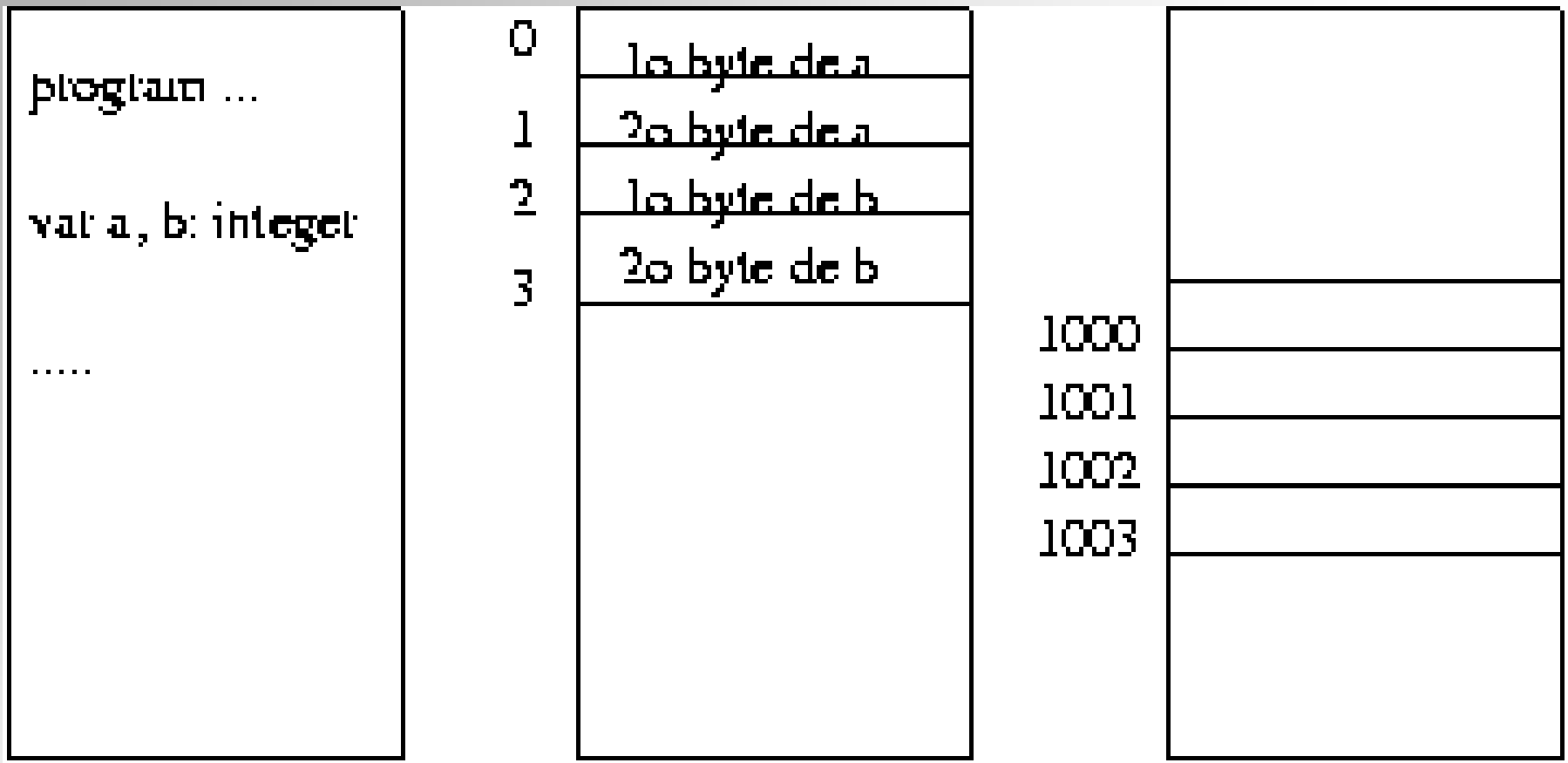
esquemas de endereçamento hardware

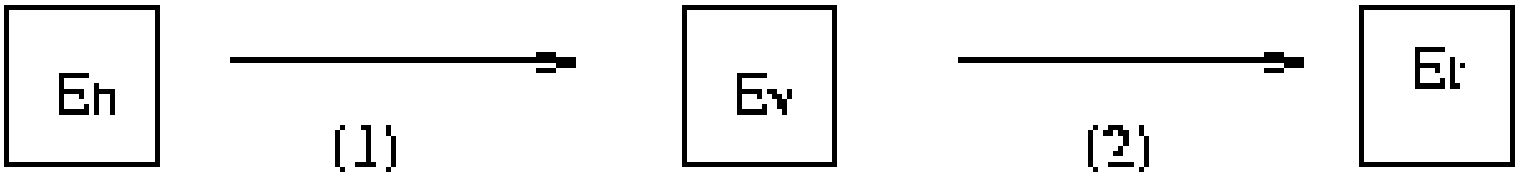
limitações do espaço (real e físico) de memória

necessidade de partilhar a memória central por múltiplos programas utilizadores

Funções do SO

- manter estruturas de dados sobre o estado da memória
- estratégias de atribuição de memória aos processos (carregar em M/ remover para disco)
- protecção entre os mapas de memória de processos diferentes (com suporte do hardware)

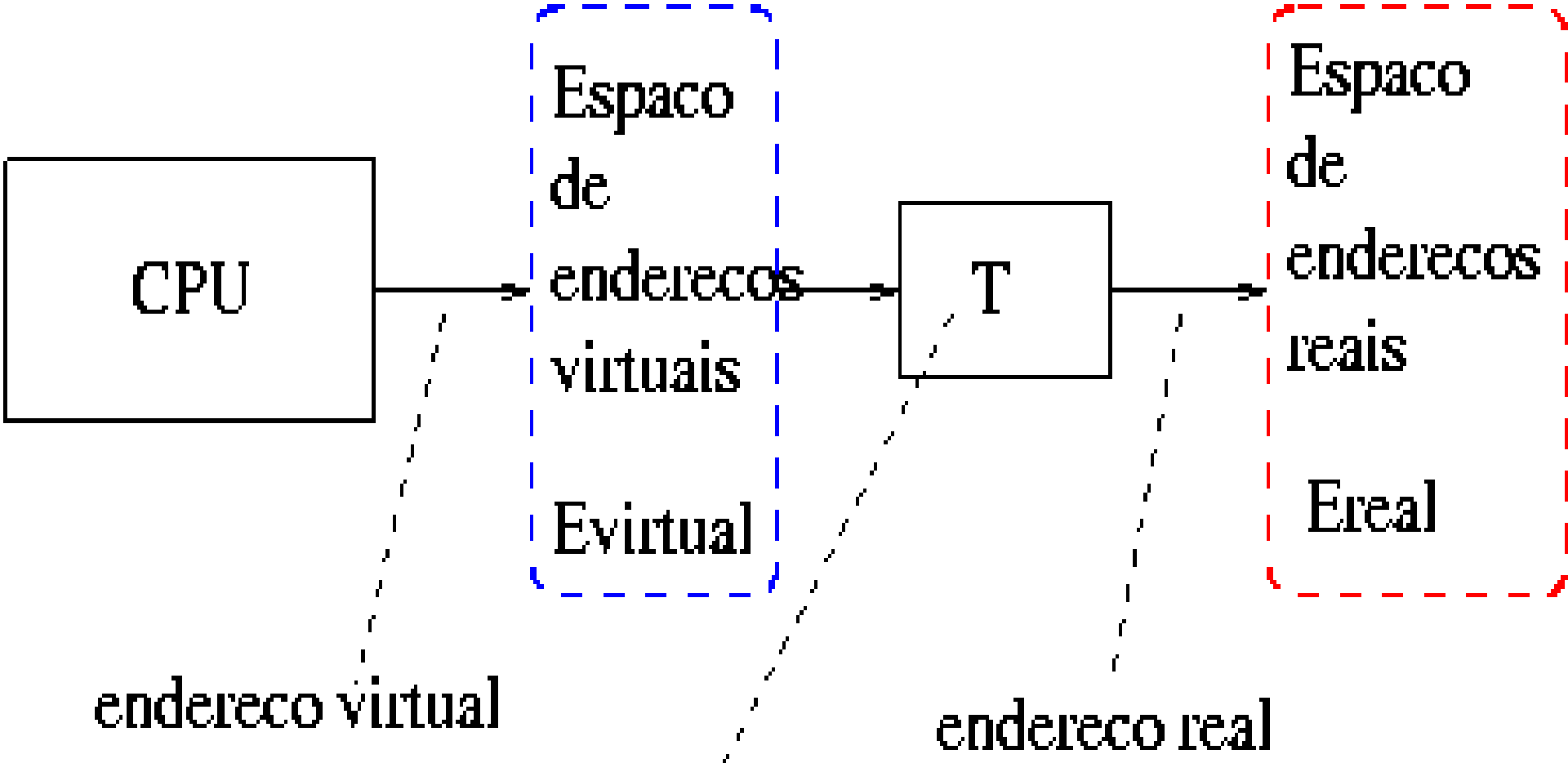

program ...
var a, b: integer
.....
0
1
2
3
1o byte de a
2o byte de a
1o byte de b
2o byte de b
1000
1001
1002
1003
Espaco de nomes do programa Pascal (En)
Espaco de endereços virtuais (Ev)
Espaco de endereços reais (Er)


En
(1)
Ev
(2)
Er

Transformação de endereços

Diversas abordagens:

- (1) os endereços de memória são já os endereços absolutos reais, quando se escreve o programa;
- (2) os endereços de memória são recolocáveis e só receberão valores absolutos reais por acção de um programa Carregador, antes de se iniciar a execução, passando a ser válidos durante toda a execução do programa (Recolocação Estática);
- (3) os endereços de memória são recolocáveis e só serão calculados os valores reais absolutos, a cada referência de memória, pelo CPU, durante a execução (Recolocação Dinâmica).



Unidade de Transformacao
de Enderecos
(MMU–Memory Management Unit)

Endereços virtuais: definidos pelo programa a nível das instruções de referência de memória e dependendo dos modos de endereçamento.

A dimensão máxima e a organização do Espaço de Endereços Virtuais (EV) é determinada pelo endereço efectivo gerado pelas instruções máquina.

Endereços reais: definidos pelas linhas de endereço do Bus que dão acesso às células físicas de Memória Central.

A dimensão máxima do Espaço de Endereços Reais (ER) é definida pelo número de linhas de endereço do Bus.

Espaço de endereços físicos (EF): definidos pela capacidade de memória central instalada em cada computador.

Em geral EV >= ER >= EF

Separação de Espaços EV e ER

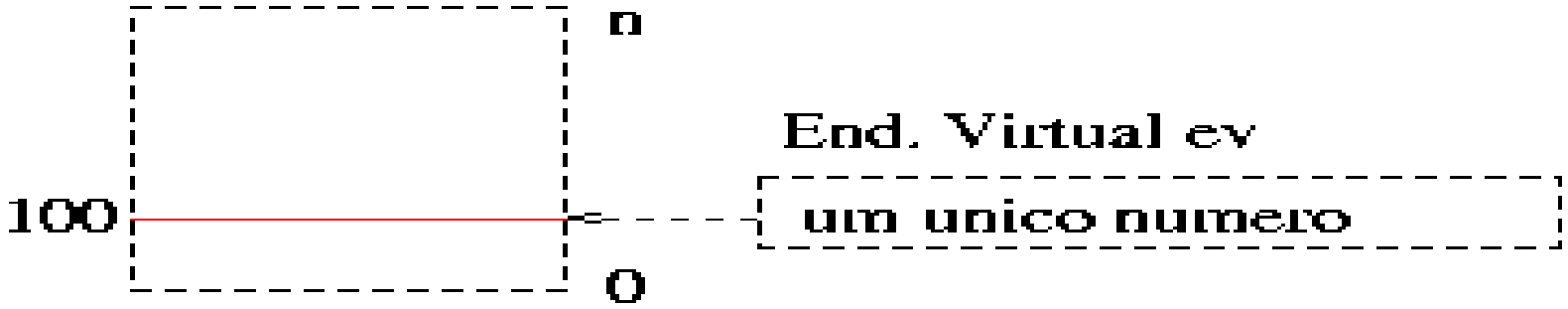
torna independentes as organizações dos dois espaços;

permite que o Espaço Virtual dum processo seja independente do Espaço Real, face a:

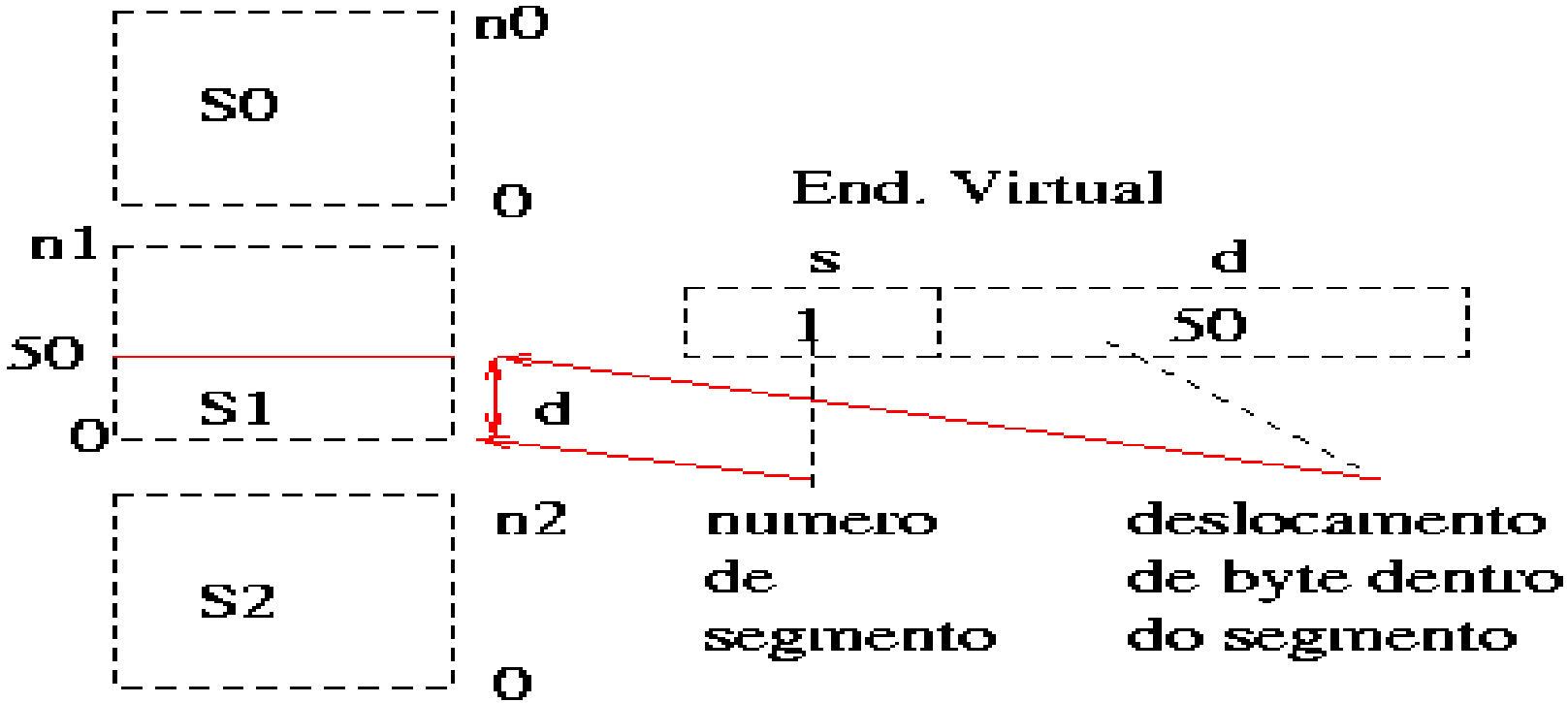
- dimensão máxima → Memória Virtual
- localização de endereços reais de memória que é atribuída a cada processo → Recolocação dinâmica

Organizacao do Espaco de Enderecos Virtuais

a) Espaco Virtual Linear: um unico segmento logico
posicoes logicamente contigu

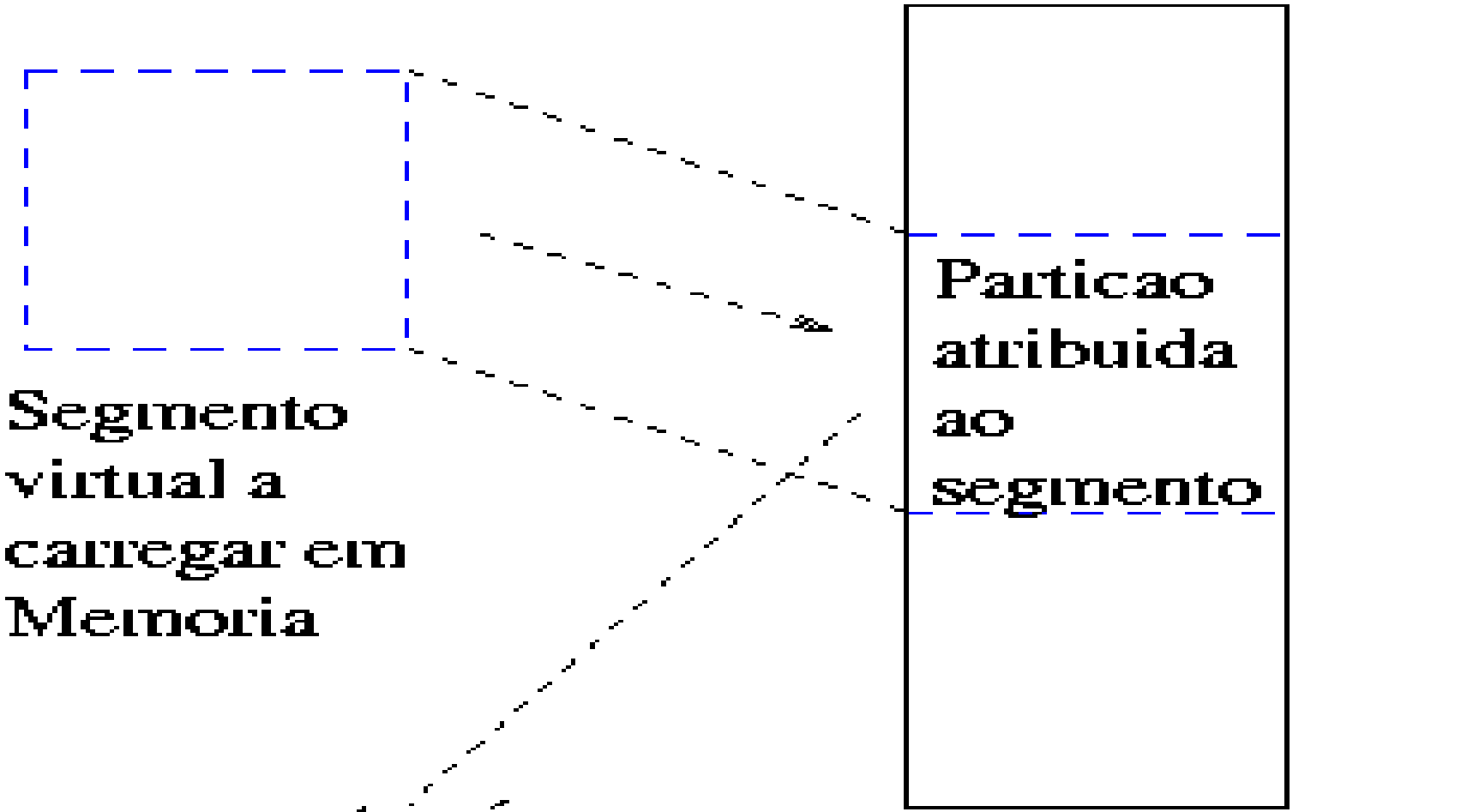


b) Espaco Virtual Segmentado

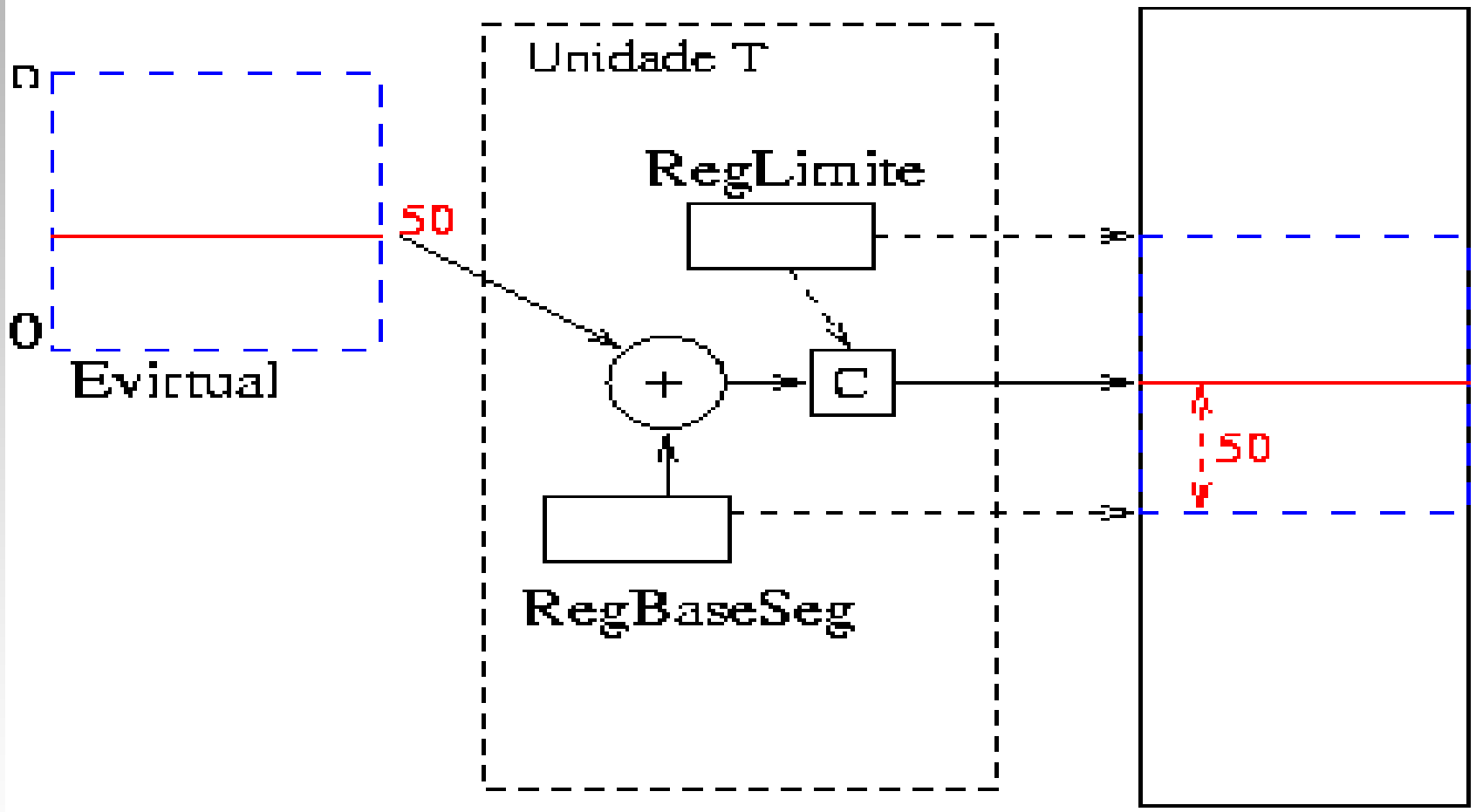


Organizacao do Espaco de Enderecos Reais

a) Particoes de memoria de tamanho variavel



Organizacao do Espaco de Enderecos Reais
a) Particoes de memoria de tamanho variavel



Transformacao de enderecos feita pela unidade T, a cada referencia de memoria gerada pelo CPU

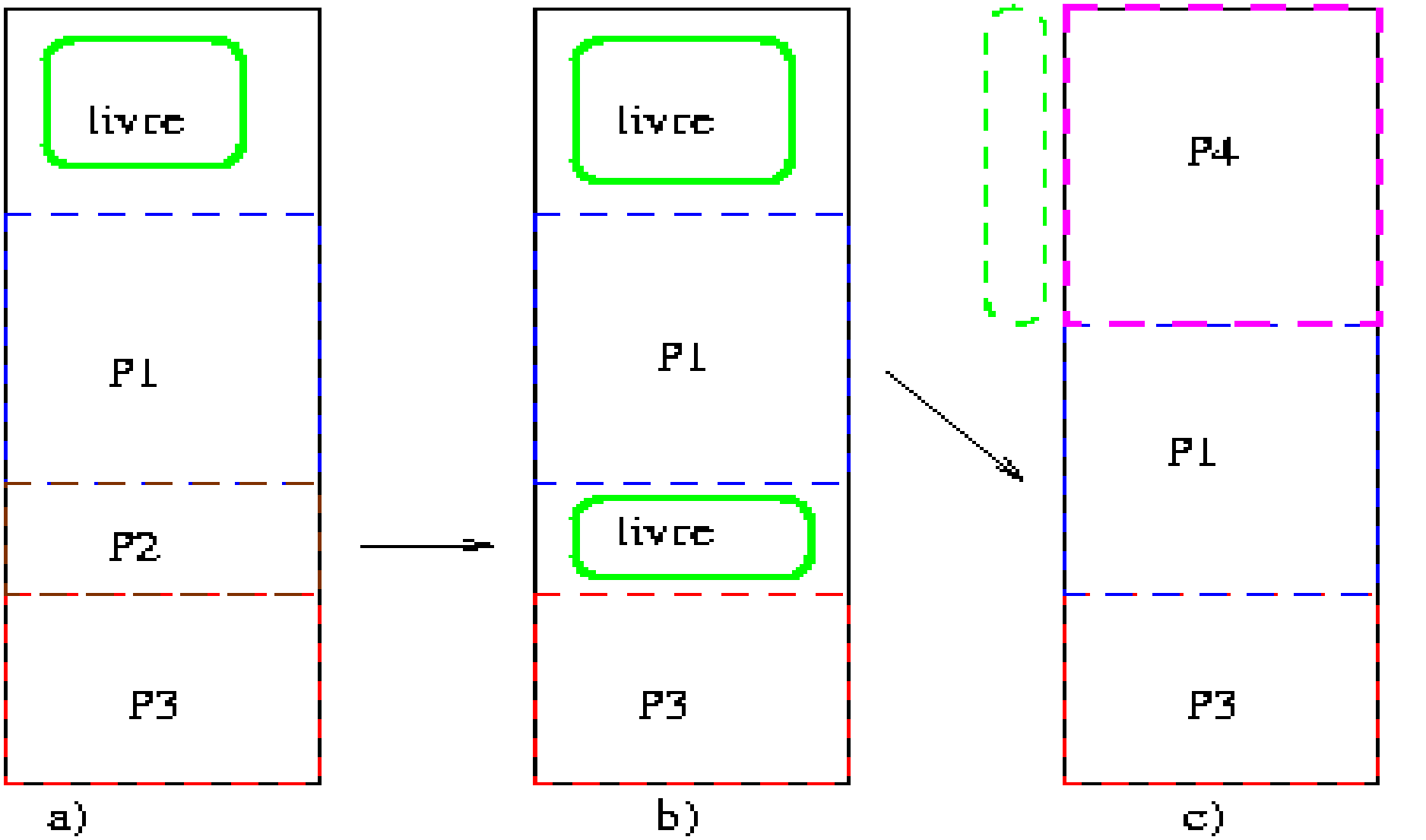
Transformação de endereços T1

EV linear → ER com partições de tamanho variável

- garante fácil recolocação
- garante independência de EV em relação à localização física em memória
- gestão elaborada do espaço de memória central
- limita a dimensão máxima de EV a ser menor ou igual à máxima memória central disponível →exige o carregamento em memória de todo o módulo executável.

Organizacao do Espaco de Enderecos Reais

a) Particoes de memoria de tamanho variavel



Transformação de endereços T2

EV linear → ER com páginas de tamanho fixo por hardware

Paginação: técnica de gestão de memória que subdivide o Espaço de endereços reais ER em zonas iguais:

→ blocos ou **páginas reais** de memória

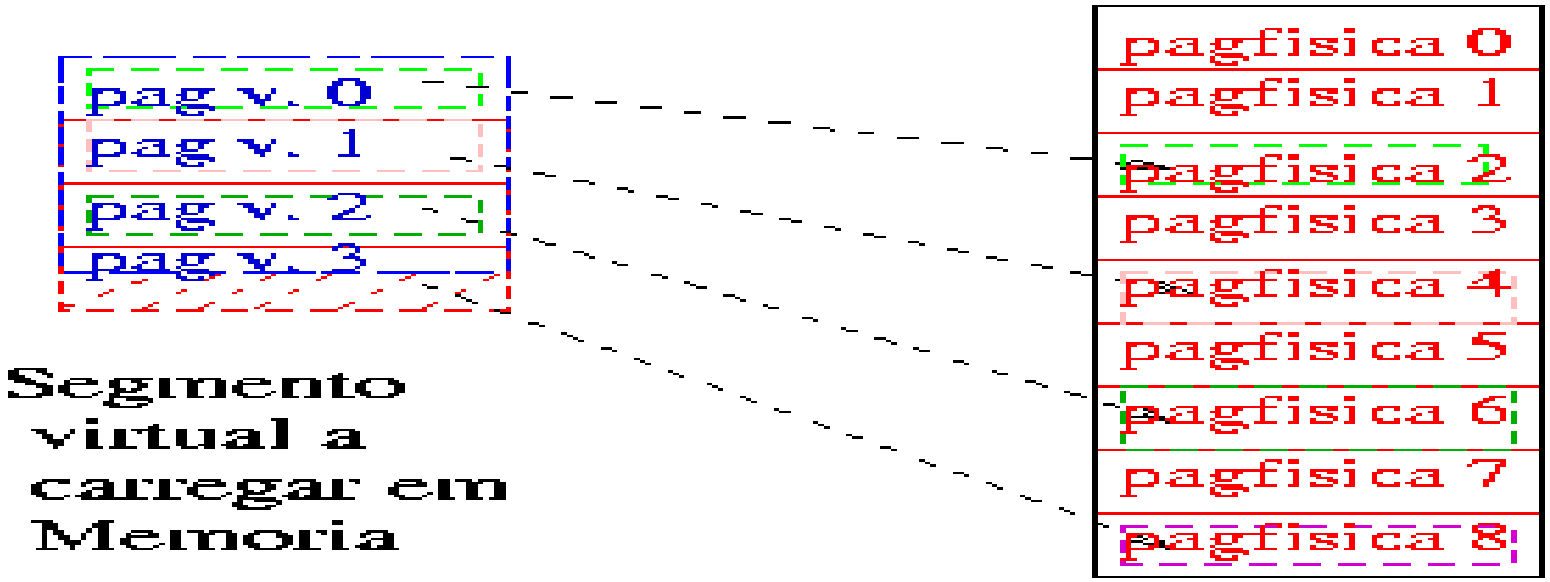
para facilitar a gestão de memória.

O espaço EV, para efeitos do carregamento em Memória, é considerado subdividido, em **páginas virtuais** (ou lógicas) de dimensão igual à das páginas reais de memória.

As páginas virtuais ficam contíguas no espaço virtual EV.

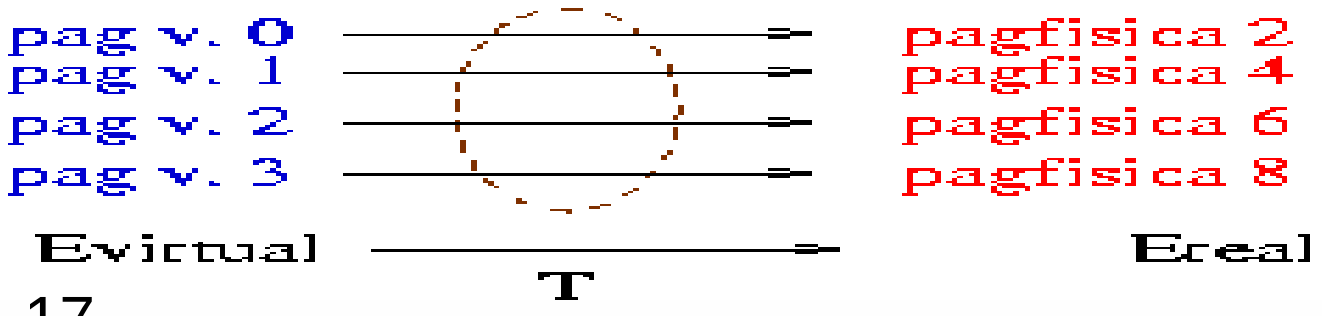
As páginas reais podem ficar espalhadas pela Memória, consoante as zonas de memória livres: não necessariamente contíguas.

Organizacao do Espaco de Enderecos Reais
b) Paginas de memoria com tamanho fixo



pagfisica 0: enderecos desde 0 ate´ 1023

pagfisica1: enderecos desde 1024 ate´ 2047



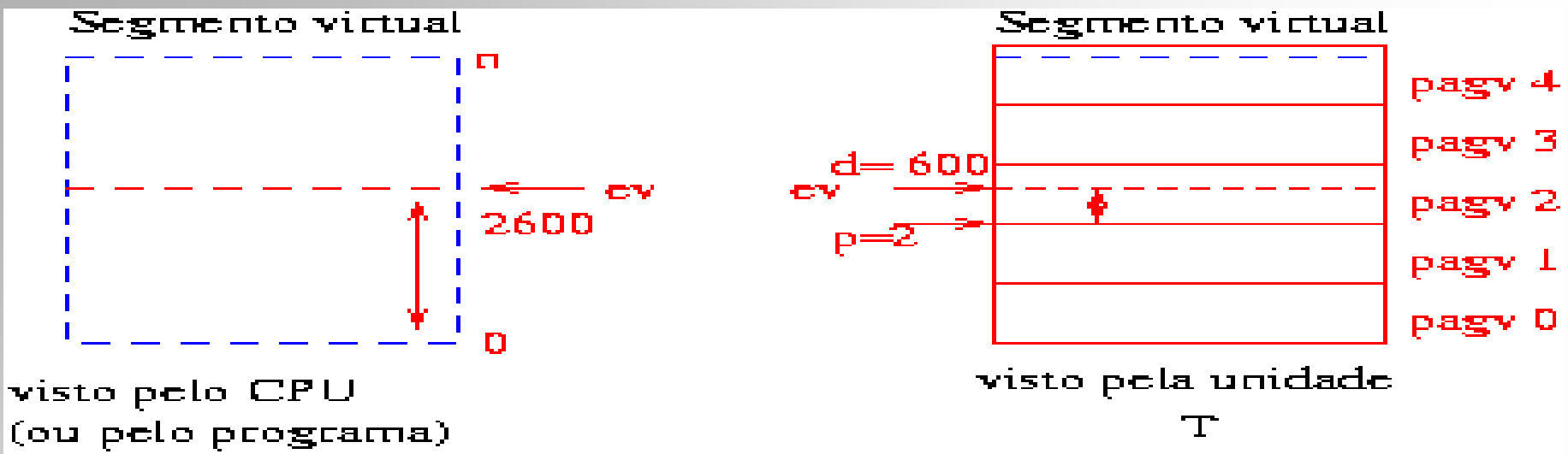
Interpretação do endereço virtual

O endereço virtual ev é gerado pelo CPU.

Antes de ser enviado para a Memória, o endereço ev é interpretado pela unidade T de transformação de endereços.

A unidade T calcula, com base no endereço ev, qual é o número de página virtual correspondente e qual é o deslocamento do byte dentro dessa página.

Esta interpretação é completamente invisível ao CPU (e ao programador).



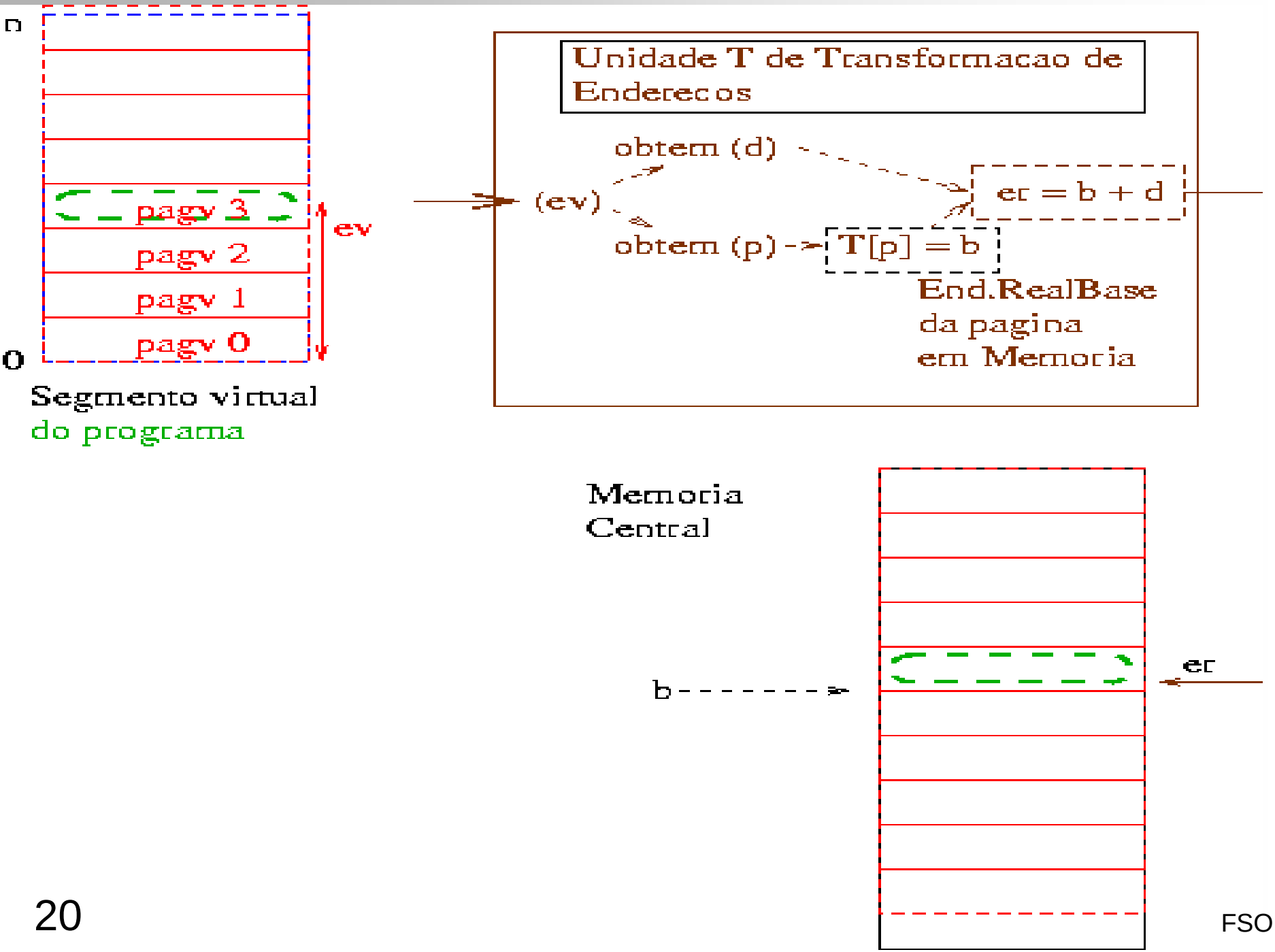
Tamanho da pagina = DIM = 1000 (base 10)

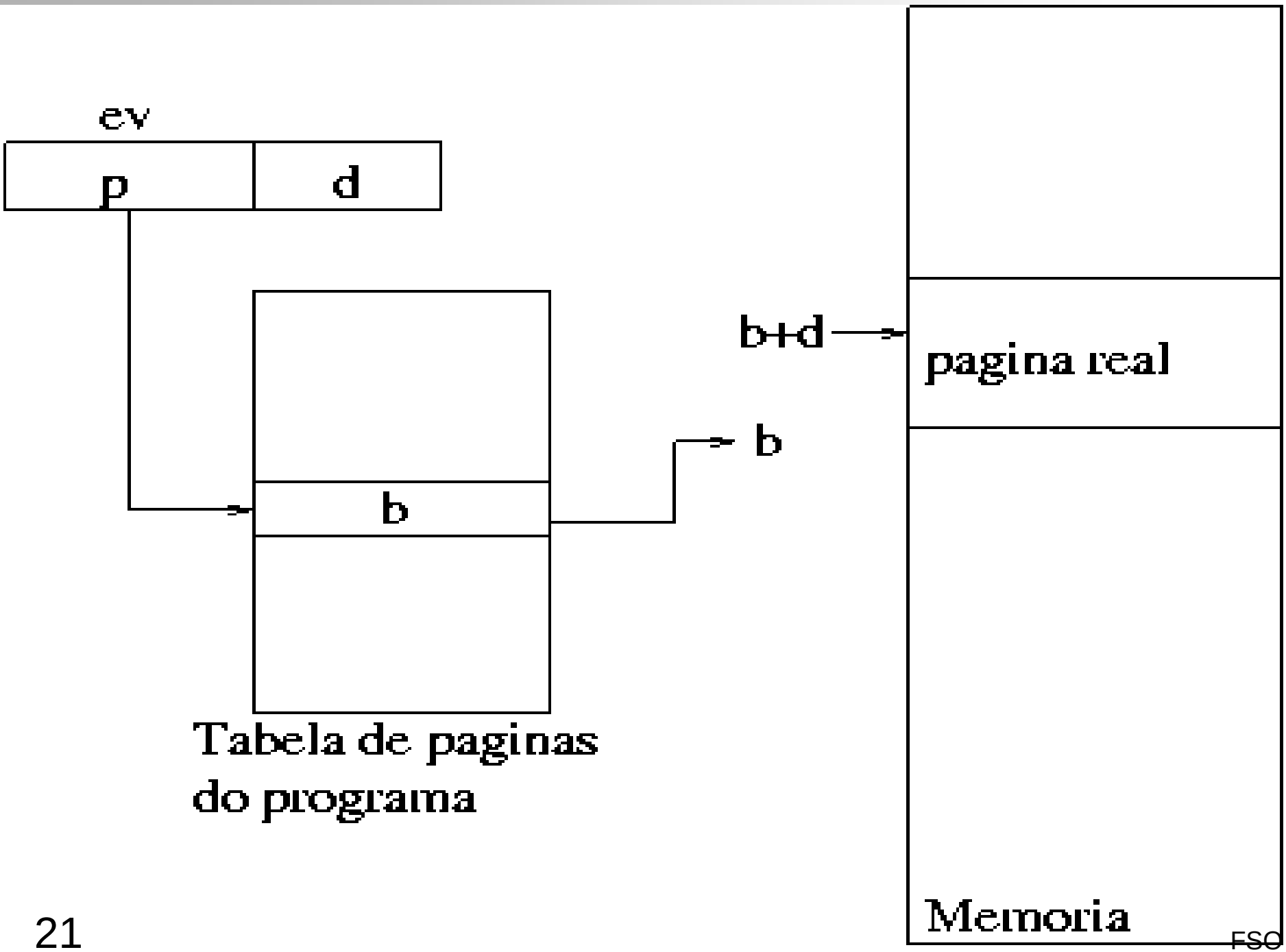
ev = 2600
interpretado como

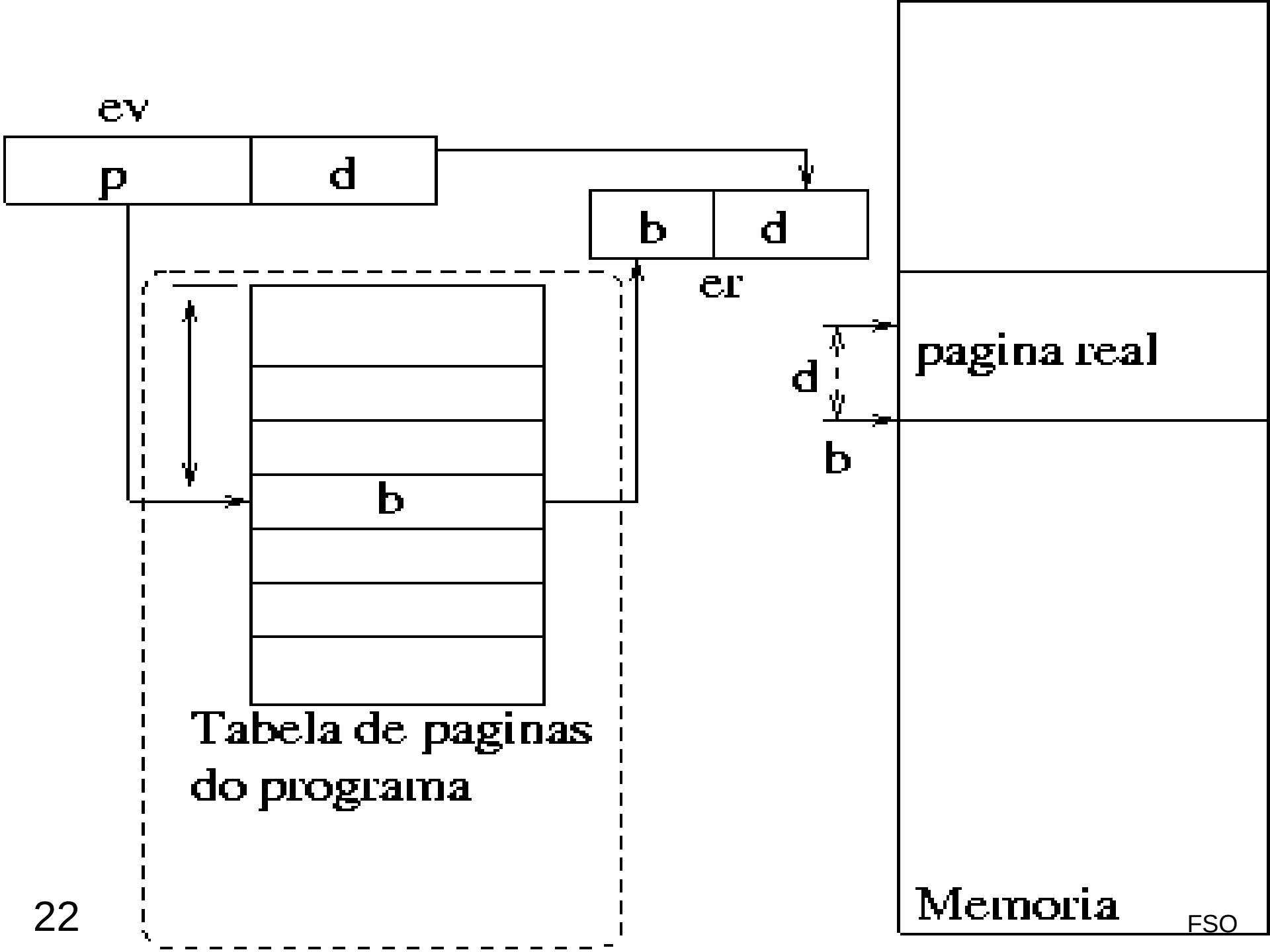
p = ev div 1000 = 2 ——> pagina virtual 2

d = ev − p * 1000 = 600 ——> deslocamento de byte dentr da pagina

A unidade T usa 2 como indice na Tabela de Paginas do Programa, onde foi colocado o Endereco Real de Base da Pagina Fisica em Memoria onde o SO carregou a Pagina Virtual 2.


n
pagv 3
pagv 2
pagv 1
pagv 0
ev
0
Segmento virtual
do programa
Unidade T de Transformacao de Enderecos
obtem (d)
(ev)
obtem (p)
er = b + d
T[p] = b
End.RealBase
da pagina
em Memoria
Memoria
Central
b
er


ev
p
d
b
b
b+d
pagina real
Tabela de paginas
do programa
Memoria


ev
p
d
b
d
er
b
Tabela de paginas
do programa
d
b
pagina real
Memoria

Exemplo: Espaço Virtual EV de 4 GigaBytes

Endereço virtual de 32 bits

31 30 29 ……..12 11 10 ……..0

Páginas de 4 KiloBytes:

O deslocamento do byte é indicado por um nº de 12 bits

O máximo EV contém 2^{32} / 2^{12} páginas = 2^{20} páginas:

O nº de página tem 20 bits

Vantagens da Paginação

Facilita a gestão eficiente de memória: quaisquer páginas reais de memória livres servem para carregar páginas virtuais.

Nâo se exige atribuição em zonas contíguas de memória: não há fragmentação da memória.

Para o carregamento de programas em memória, permite 2 casos

a) Pré-carregamento de todas as páginas virtuais de um programa em Memória, antes de iniciar a execução.

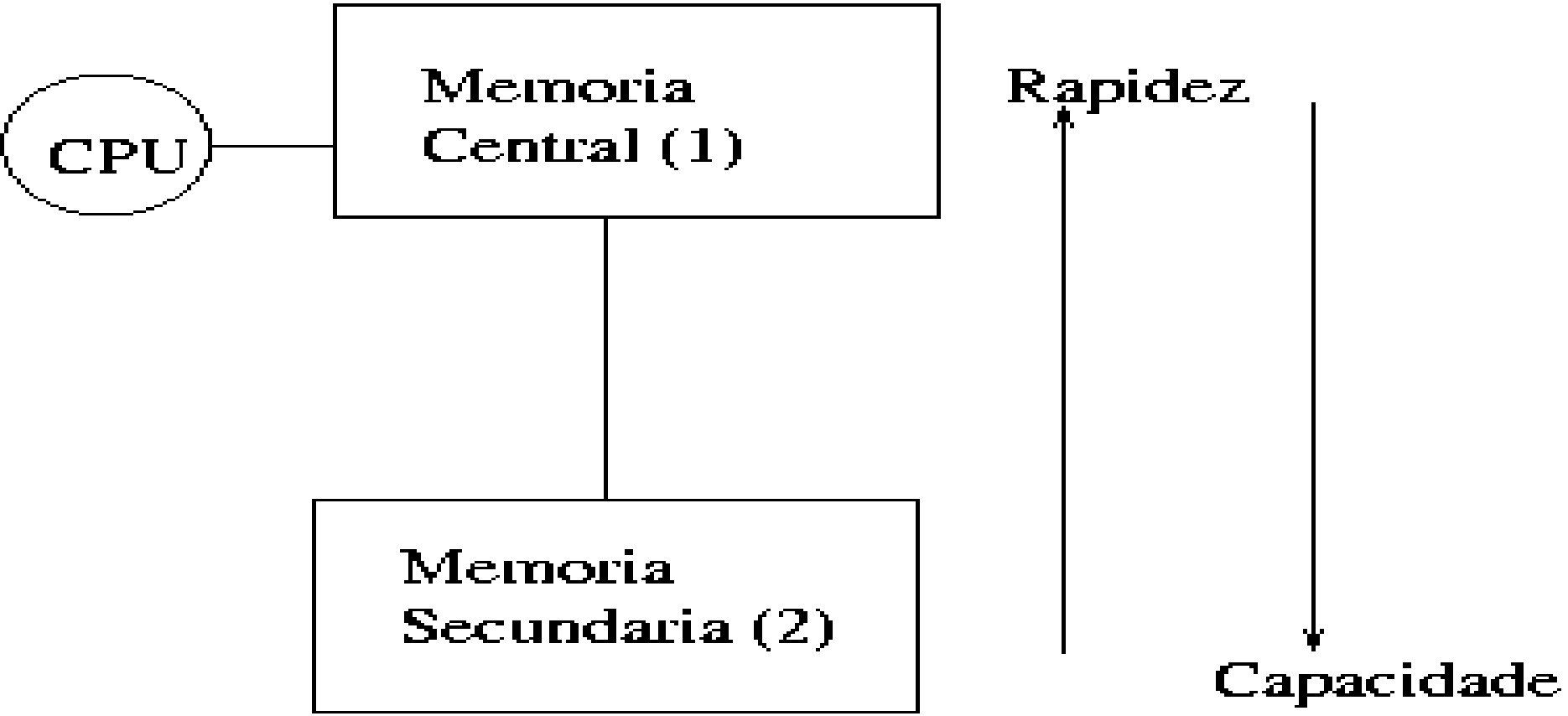
b) Carregamento a pedido (*On-demand paging*): esquema dinâmico de carregamento de páginas:

só carrega uma dada página virtual, no momento da 1ª referência, durante a execução: quando o endereço é gerado pelo CPU, a unidade T de transformação de endereços detecta se a correspondente página virtual já está carregada em memória ou não:

- Se a página virtual referida não estiver ainda em memória, a unidade T gera um pedido de interrupção ao CPU (*Page-fault interrupt*):
  - a rotina de serviço de interrupções deste tipo:
    - Desencadeia o carregamento da página virtual pedida, de disco para memória central

- Consegue-se, assim, simular uma memória virtual cuja capacidade é definida pelo tamanho dos endereços virtuais (por exemplo 4 GigaBytes), que pode ser superior à capacidade do Espaço de endereços reais e à capacidade da Memória central instalada.

Dois niveis na hierarquia de memoria



Um esquema dinamico:

- deteccao de referencias a paginas ausentes de memoria
- gestao de transferencias entre a Memoria e o Disco

Paginação dinâmica, por pedido:

Hardware adicional:

na Tabela de Páginas: um bit indicador de Presença / Ausência de página em Memória

(inicializado pelo SO, por omissão, todos os bits a 0 no início)

suporte para tipo de interrupções por FaltadePágina:

- a unidade T deve poder determinar qual o endereço virtual ev que gerou a situação de ´página ausente´;

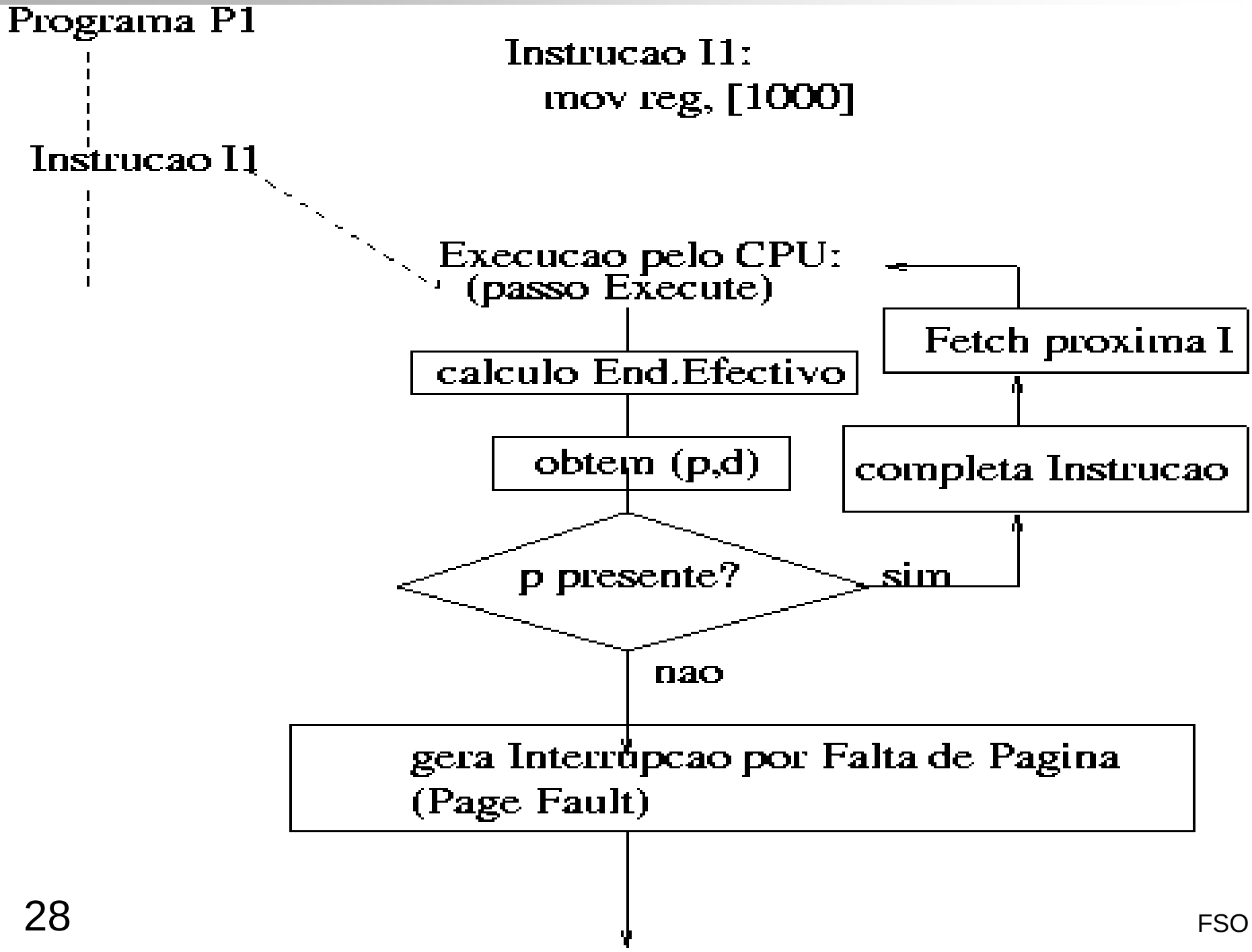
- a instrução que gerou essa referência deve poder ser repetida, desde o seu início.

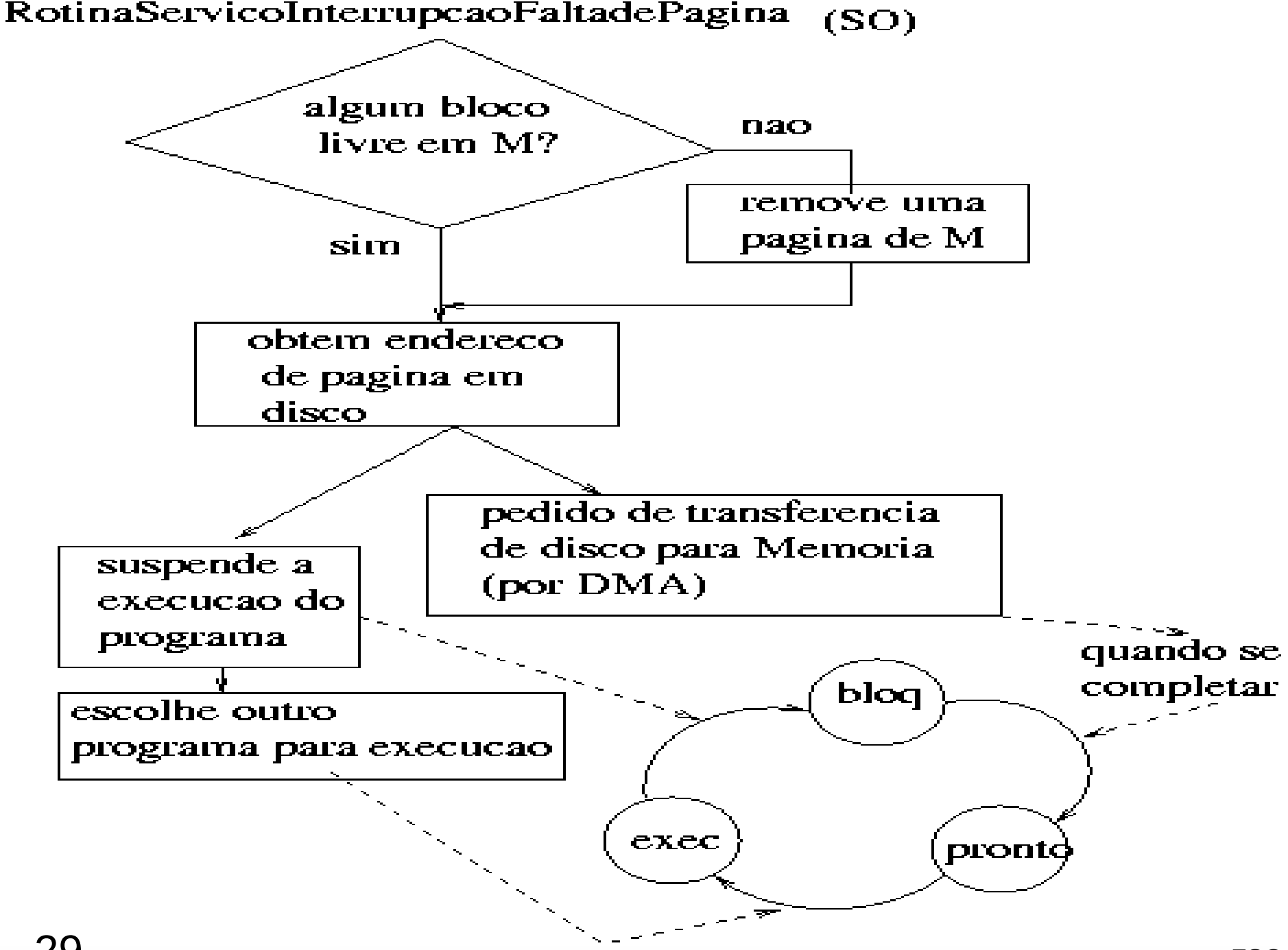
Software adicional (do SO):

rotina de serviço de interrupções por falta de página

rotinas de controlo da transferência de páginas disco-Mem

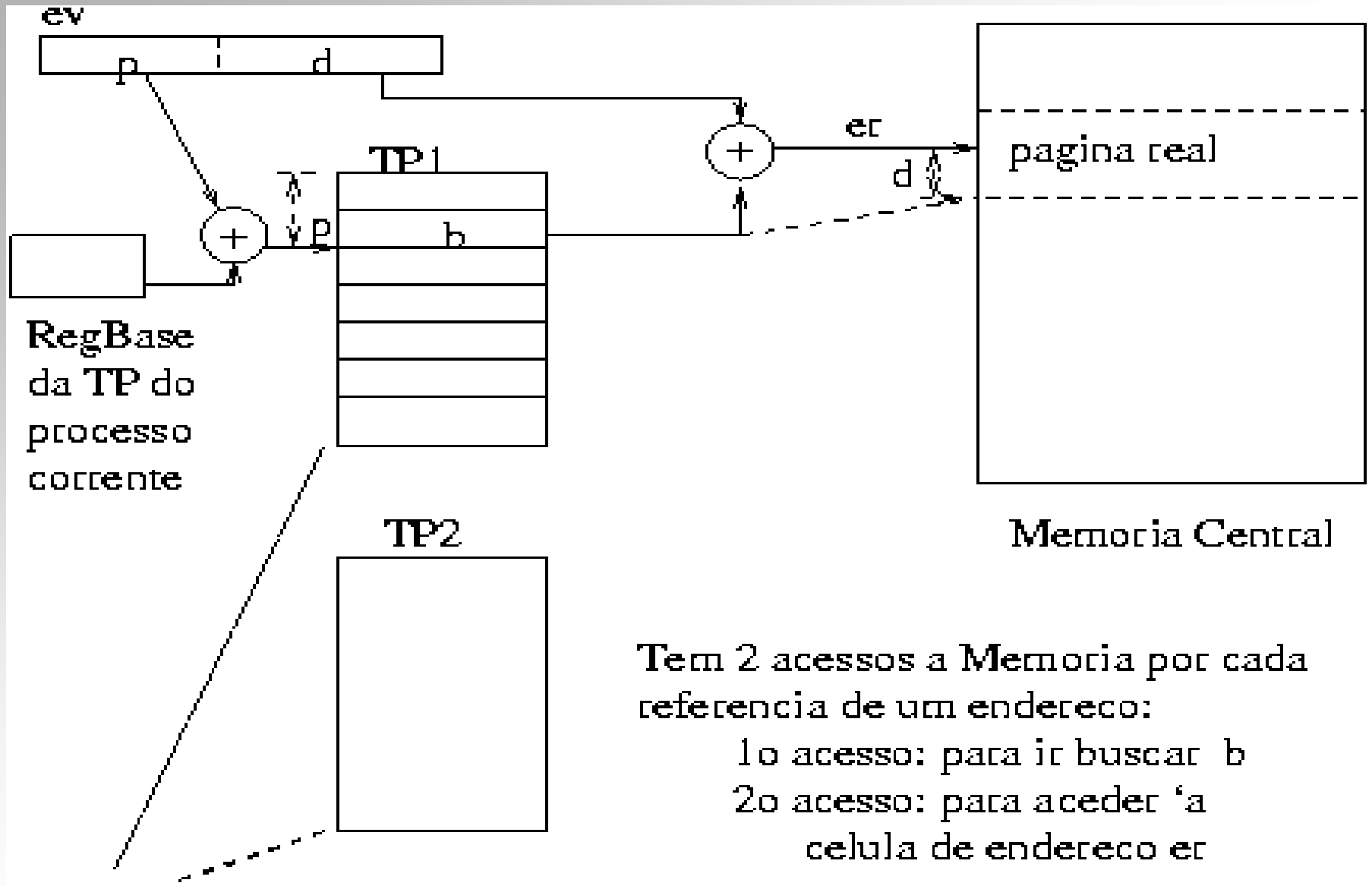
rotinas de gestão do espaço de memória, em termos de páginas

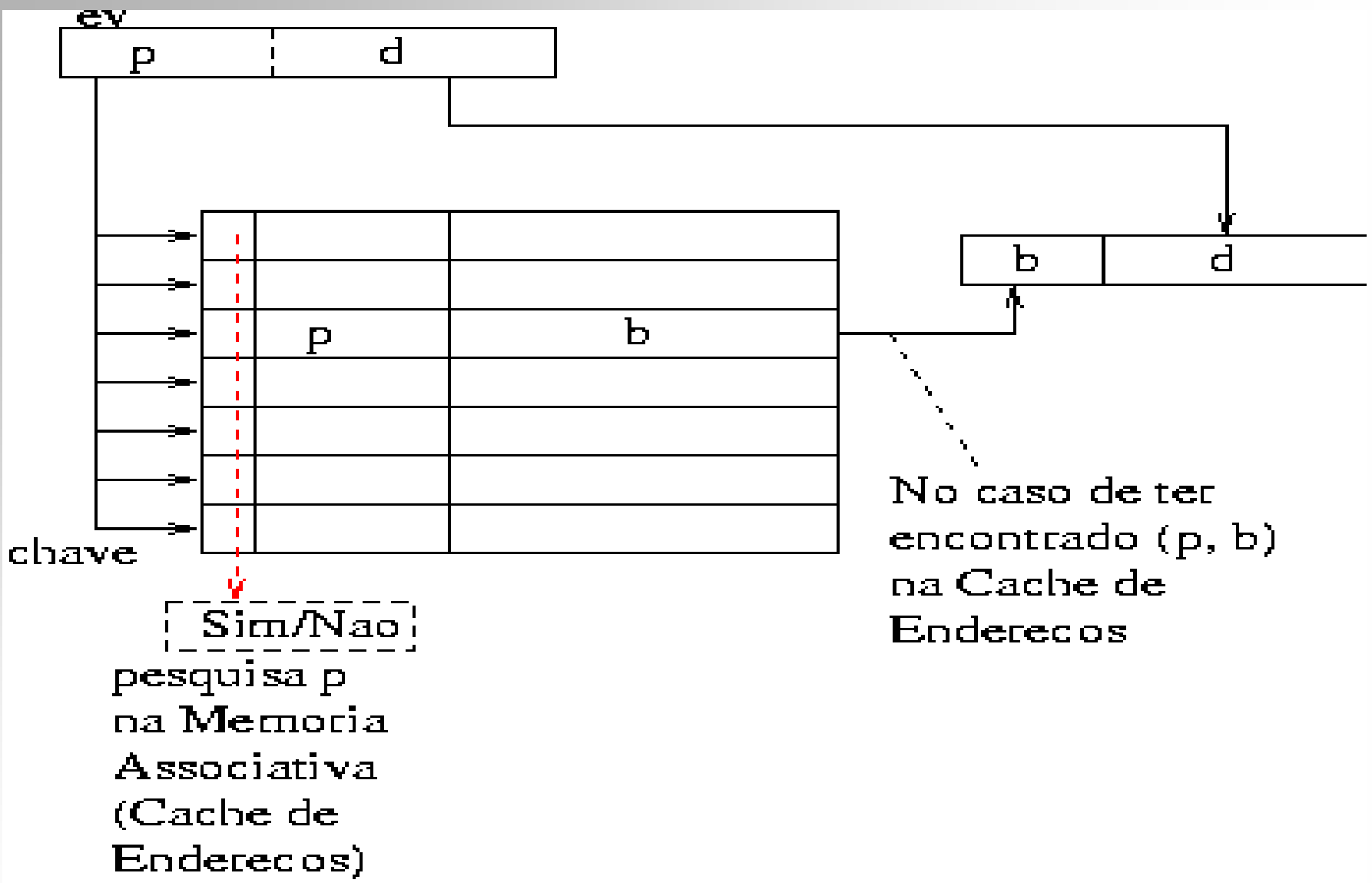

Programa P1
Instrucao I1:
mov reg, [1000]
Instrucao I1
Execucao pelo CPU:
(passo Execute)
Fetch proxima I
calculo End.Efectivo
obtem (p,d)
completa Instrucao
p presente?
sim
nao
gera Interrupcao por Falta de Pagina
(Page Fault)


RotinaServicoInterrupcaoFaltadePagina (SO)
algum bloco livre em M?
nao
remove uma pagina de M
sim
obtem endereco de pagina em disco
suspende a execucao do programa
pedido de transferencia de disco para Memoria (por DMA)
quando se completar
escolhe outro programa para execucao
bloq
exec
pronto

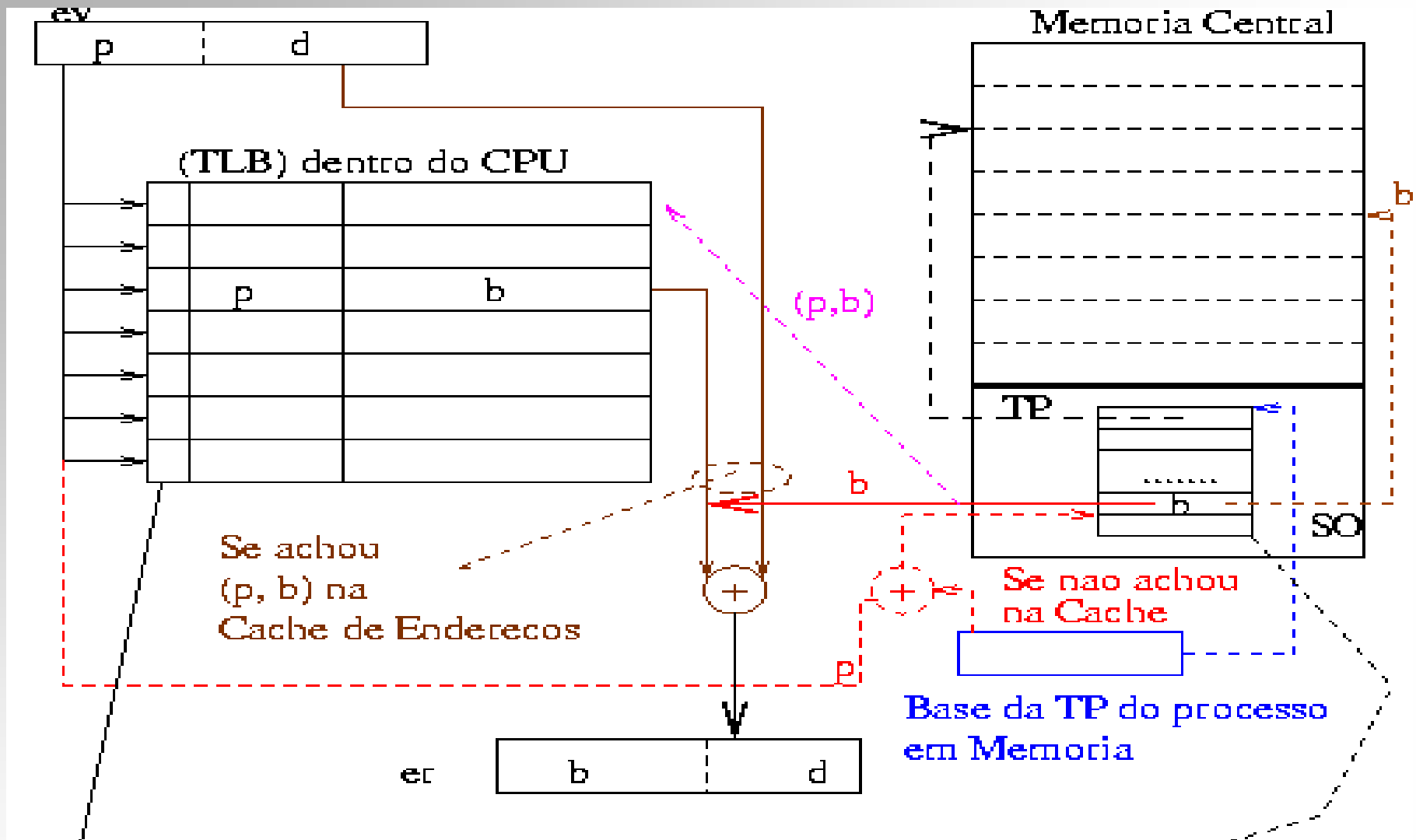
Características da Paginação:

- Elimina fragmentação externa de memória
- Apresenta fragmentação interna do módulo do programa, se este tiver um tamanho que não é múltiplo do tamanho da página
- Com paginação por pedido, suporta Memória Virtual, tal que, a nível do programa, o espaço de endereçamento ´´visto´´ é uma memória linear, de dimensão superior à da memória real
- Exige suporte hardware adicional e eficiência nas transferências entre memória central e secundária
- Exige suporte software, a nível do SO, com algoritmos que tentem optimizar a utilização do espaço de memória e escolher o melhor conjunto de páginas a manter carregadas em memória, a cada momento.


ev
p
d
er
pagina real
d
TP1
p
b
RegBase
da TP do
processo
corrente
TP2
Memoria Central
Tem 2 acessos a Memoria por cada
referencia de um endereco:
1o acesso: para ir buscar b
2o acesso: para aceder 'a
celula de endereco er
Tabelas de Paginas dos
Processos: estao em Memoria,
em zona reservada do SO

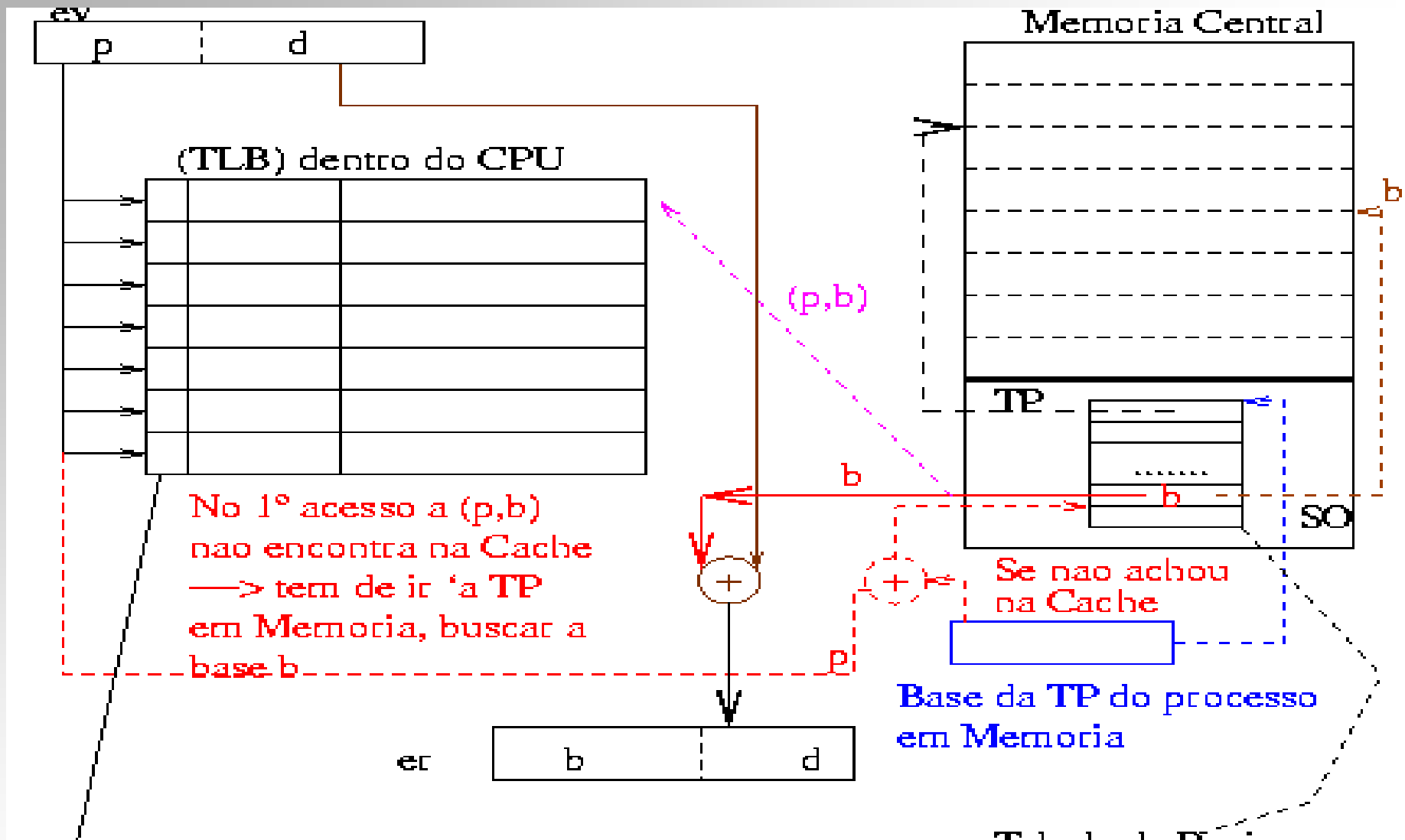


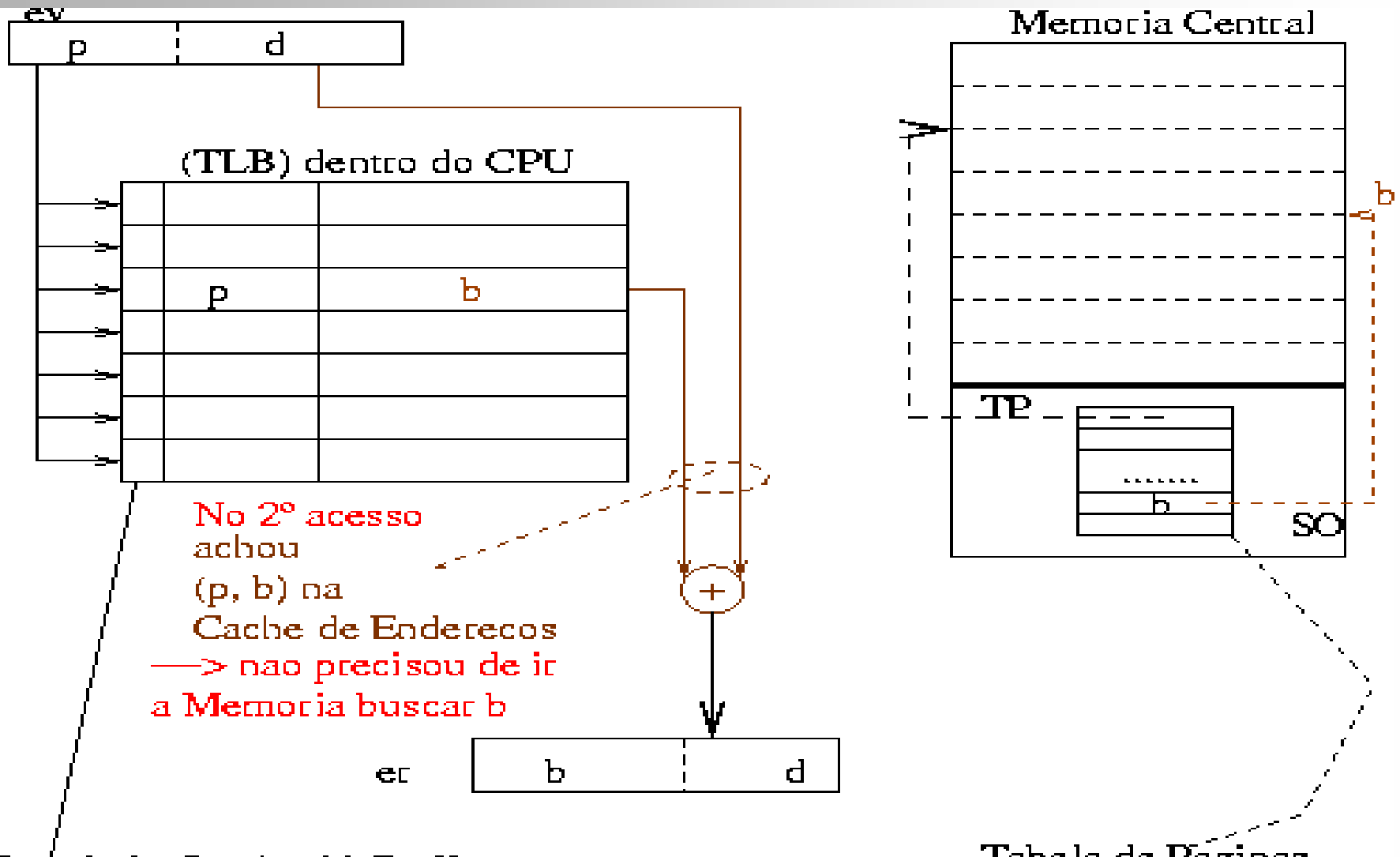
Chave: nº da pagina virtual
Conteudo: base da pagina em Memoria



TranslationLookasideBuffer
(TLB) dentro do CPU
— contem os pares (p, b) das paginas que foram referenciadas nas ultimas vezes pelo programa

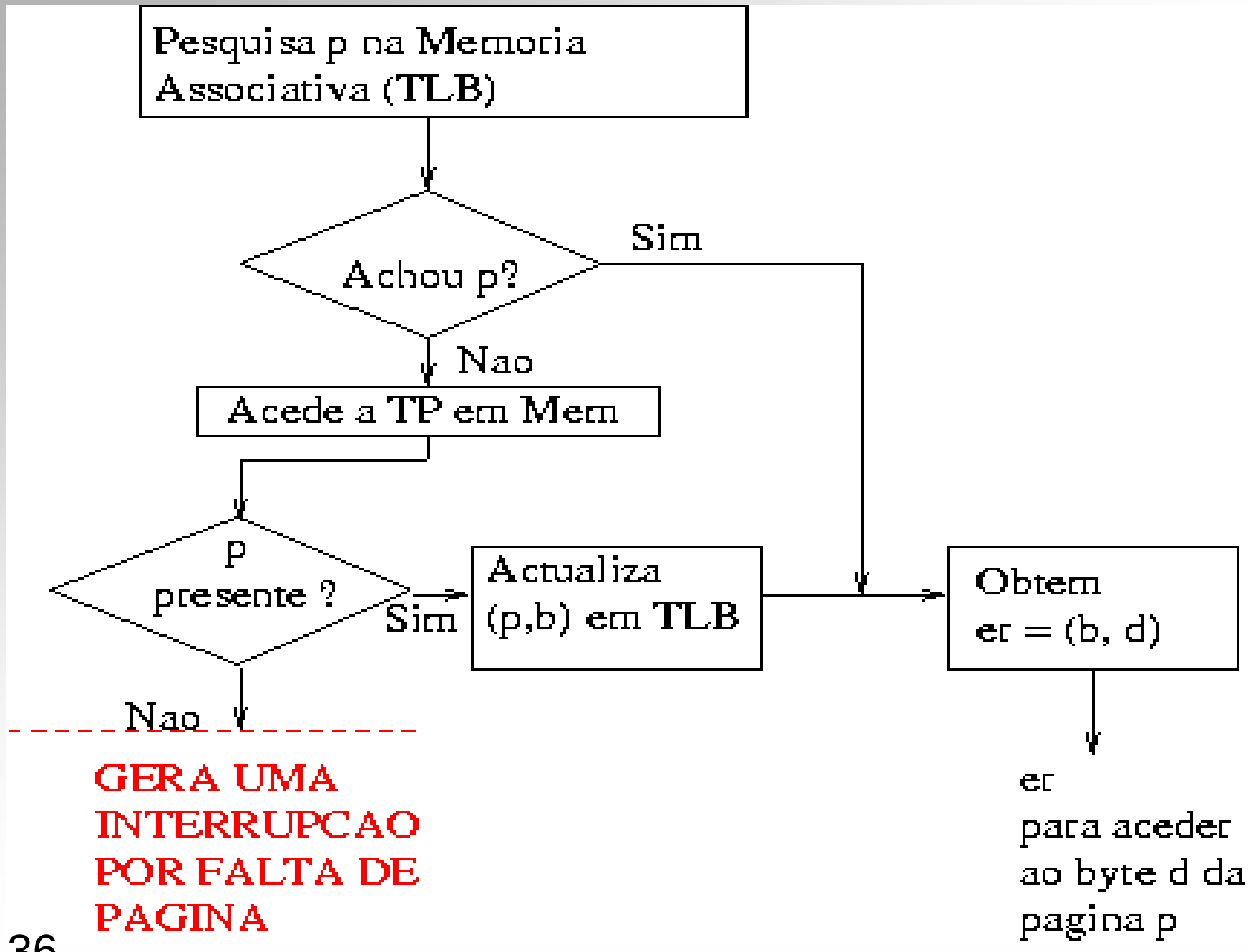
Tabela de Paginas
do Processo corrente
em Memoria

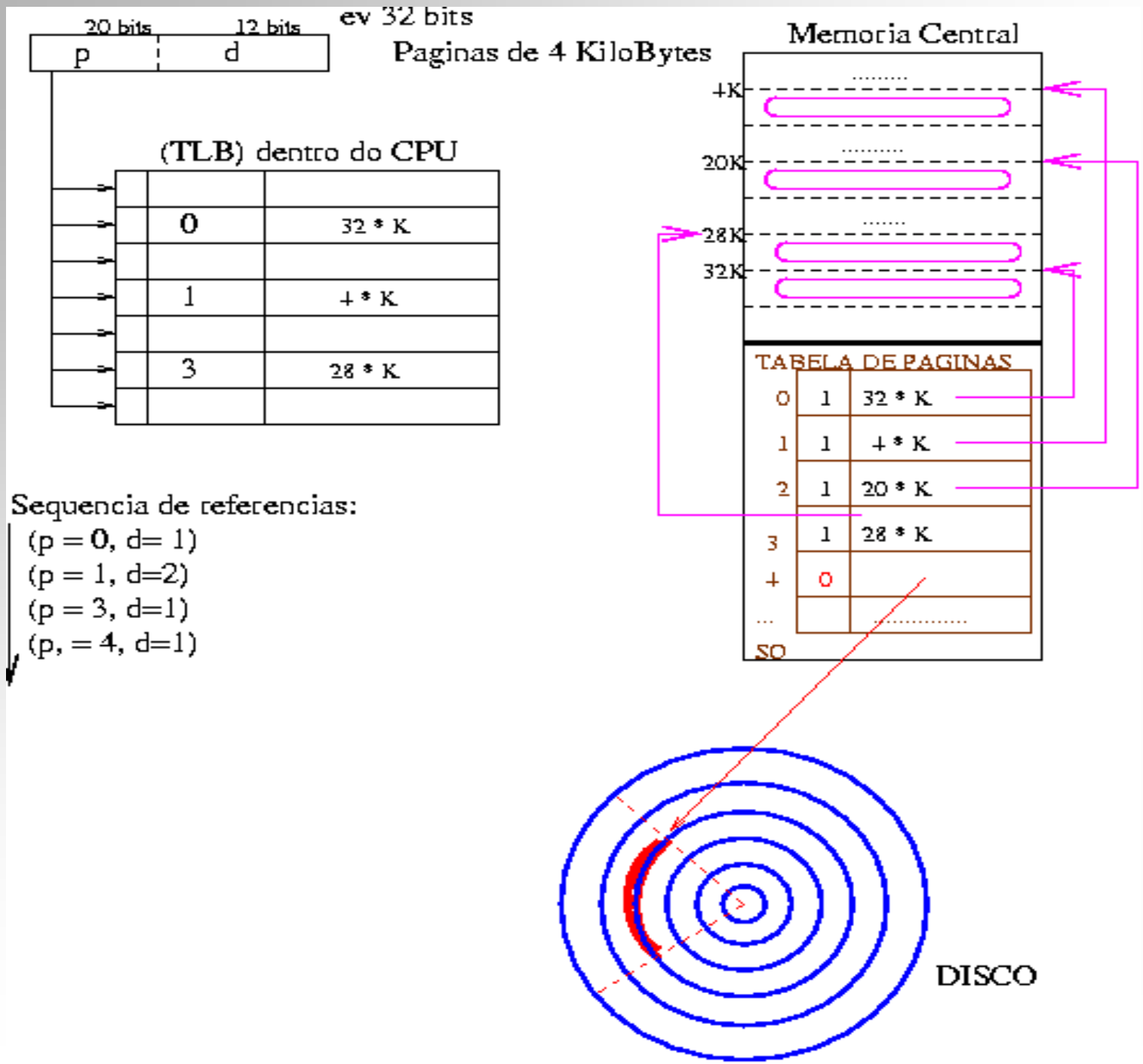

ev
p
d
(TLB) dentro do CPU
Memoria Central
b
(p,b)
TP
.....
b
b
SO
No 1º acesso a (p,b)
nao encontra na Cache
—> tem de ir 'a TP
em Memoria, buscar a
base b
p
Se nao achou
na Cache
Base da TP do processo
em Memoria
er
b
d
TranslationLookasideBuffer
(TLB) dentro do CPU
— contem os pares (p, b) das
paginas que foram referenciadas
nas ultimas vezes pelo programa
Tabela de Paginas
do Processo corrente
em Memoria

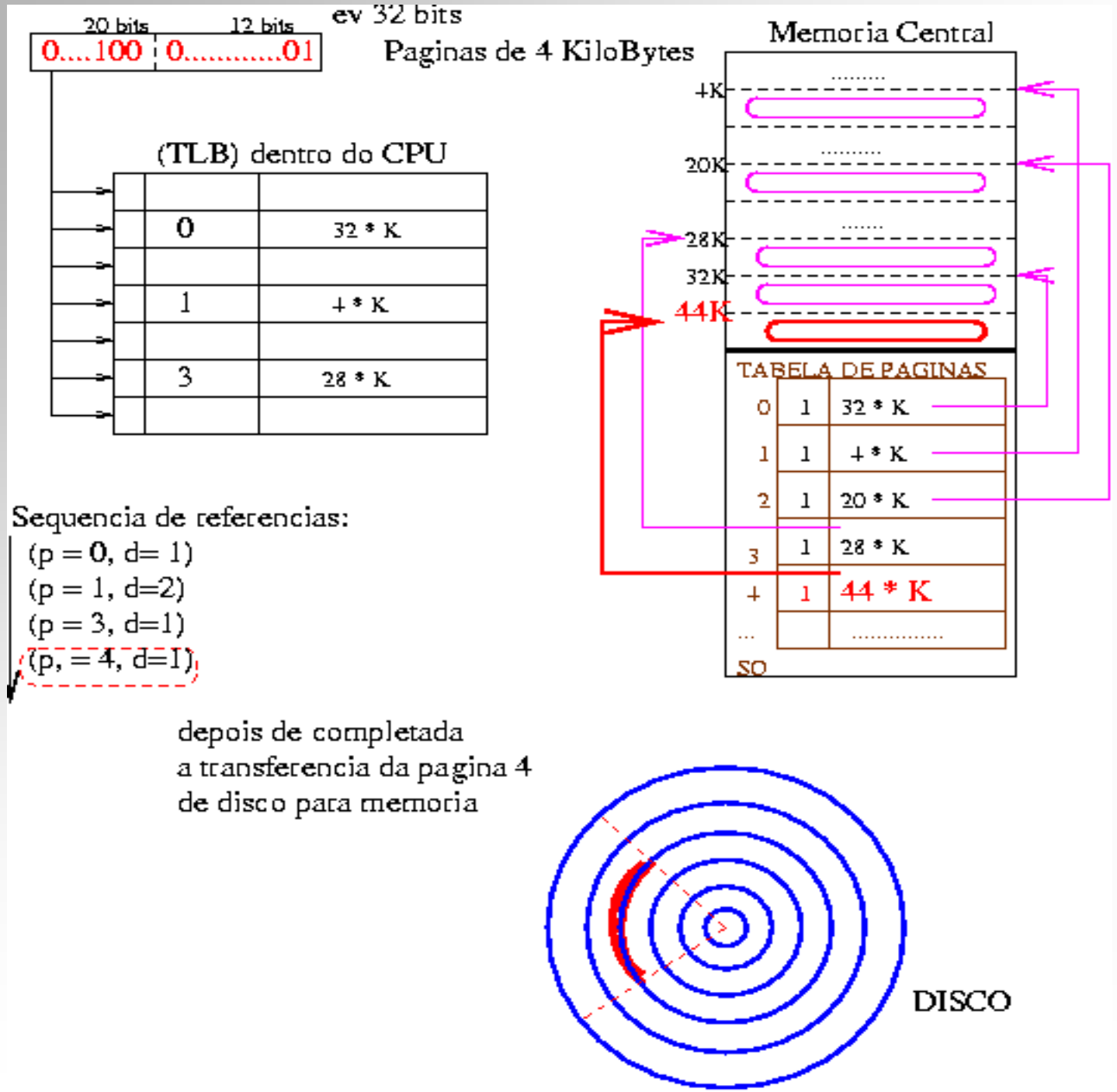


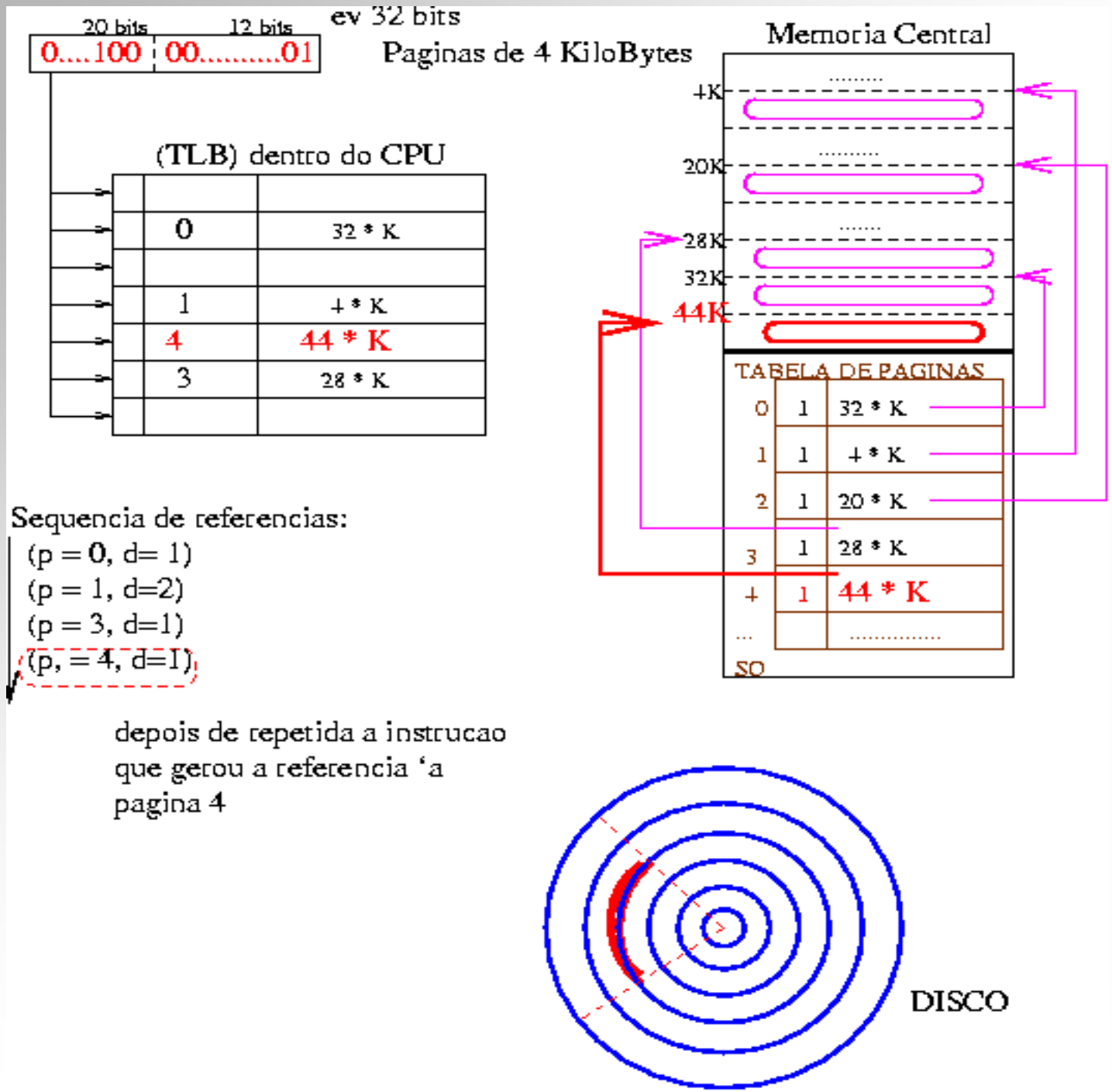
TranslationLookasideBuffer (TLB) dentro do CPU
— contem os pares (p, b) das paginas que foram referenciadas nas ultimas vezes pelo programa

Tabela de Paginas do Processo corrente em Memoria


Pesquisa p na Memoria Associativa (TLB)
Achou p?
Sim
Nao
Acede a TP em Mem
p presente ?
Sim
Nao
Actualiza (p,b) em TLB
Obtem er = (b, d)
GERA UMA INTERRUPCAO POR FALTA DE PAGINA
er para aceder ao byte d da pagina p


20 bits
12 bits
p
d
ev 32 bits
Paginas de 4 KiloBytes
Memoria Central
(TLB) dentro do CPU
0
32 * K
1
4 * K
3
28 * K
4K
20K
28K
32K
TABELA DE PAGINAS
0 1 32 * K
1 1 4 * K
2 1 20 * K
3 1 28 * K
4 0
SO
Sequencia de referencias:
(p = 0, d= 1)
(p = 1, d=2)
(p = 3, d=1)
(p, = 4, d=1)
DISCO


20 bits
12 bits
ev 32 bits
0....100
0.....01
Paginas de 4 KiloBytes
Memoria Central
(TLB) dentro do CPU
0
32 * K
1
4 * K
3
28 * K
4K
20K
28K
32K
44K
TABELA DE PAGINAS
0 1 32 * K
1 1 4 * K
2 1 20 * K
3 1 28 * K
4 1 44 * K
SO
Sequencia de referencias:
(p = 0, d= 1)
(p = 1, d=2)
(p = 3, d=1)
(p, = 4, d=1)
depois de completada
a transferencia da pagina 4
de disco para memoria
DISCO


20 bits
12 bits
ev 32 bits
0....100
00.....01
Paginas de 4 KiloBytes
Memoria Central
(TLB) dentro do CPU
0
32 * K
1
4 * K
4
44 * K
3
28 * K
4K
20K
28K
32K
44K
TABELA DE PAGINAS
0 1 32 * K
1 1 4 * K
2 1 20 * K
3 1 28 * K
4 1 44 * K
SO
Sequencia de referencias:
(p = 0, d= 1)
(p = 1, d=2)
(p = 3, d=1)
(p, = 4, d=1)
depois de repetida a instrucao
que gerou a referencia 'a
pagina 4
DISCO

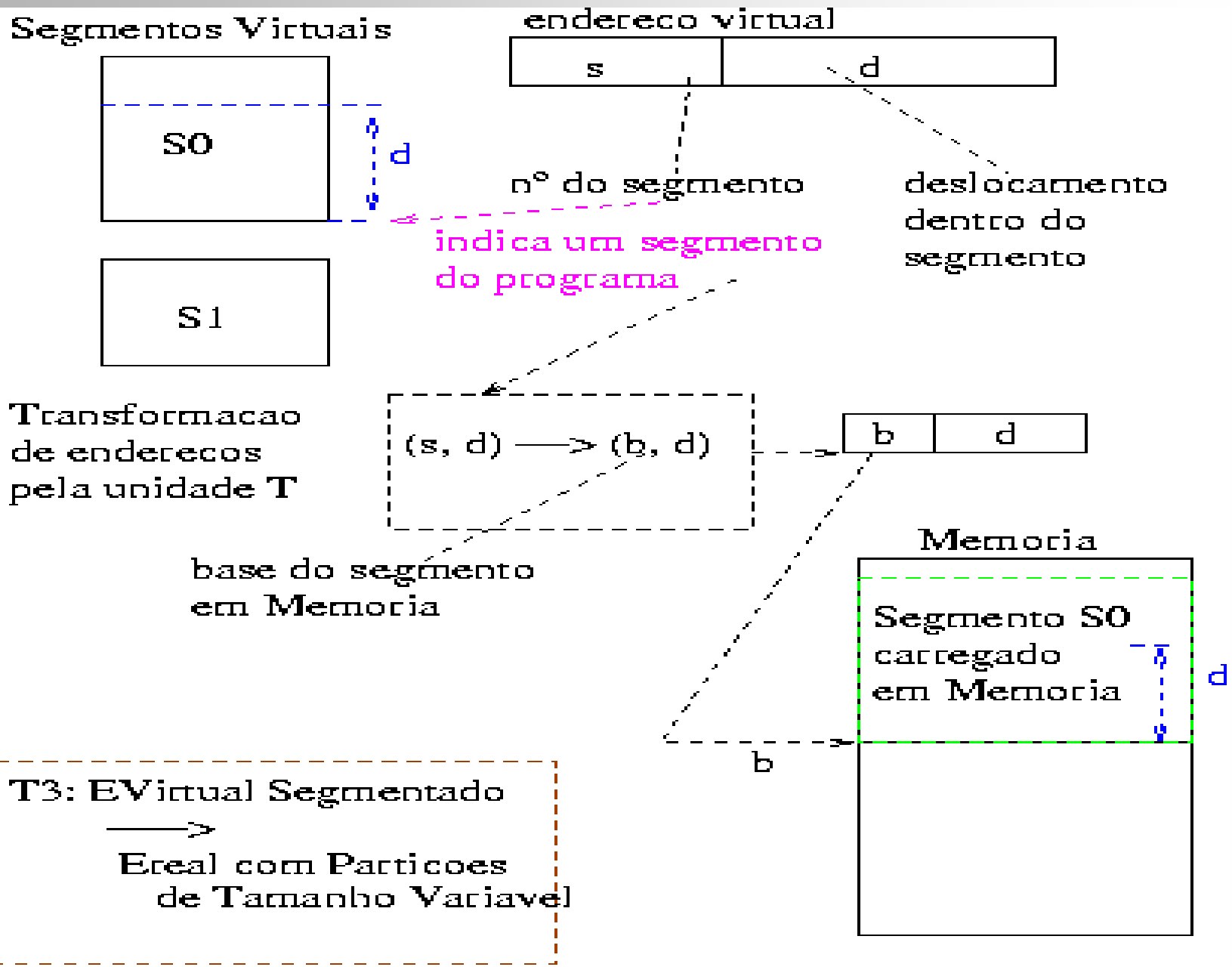
Transformação T3: EV Segmentado → ER: Partições Variáveis

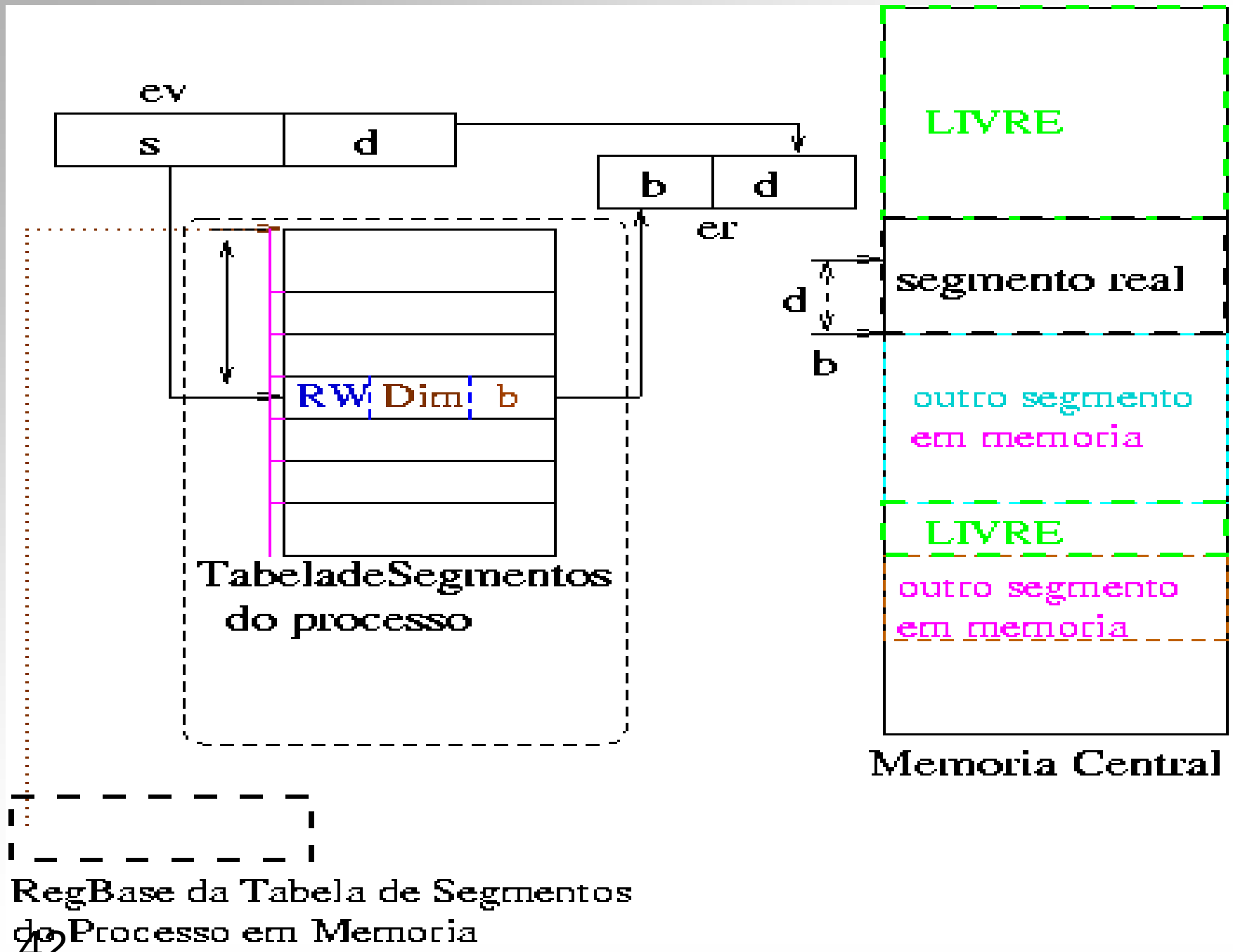
Permite:

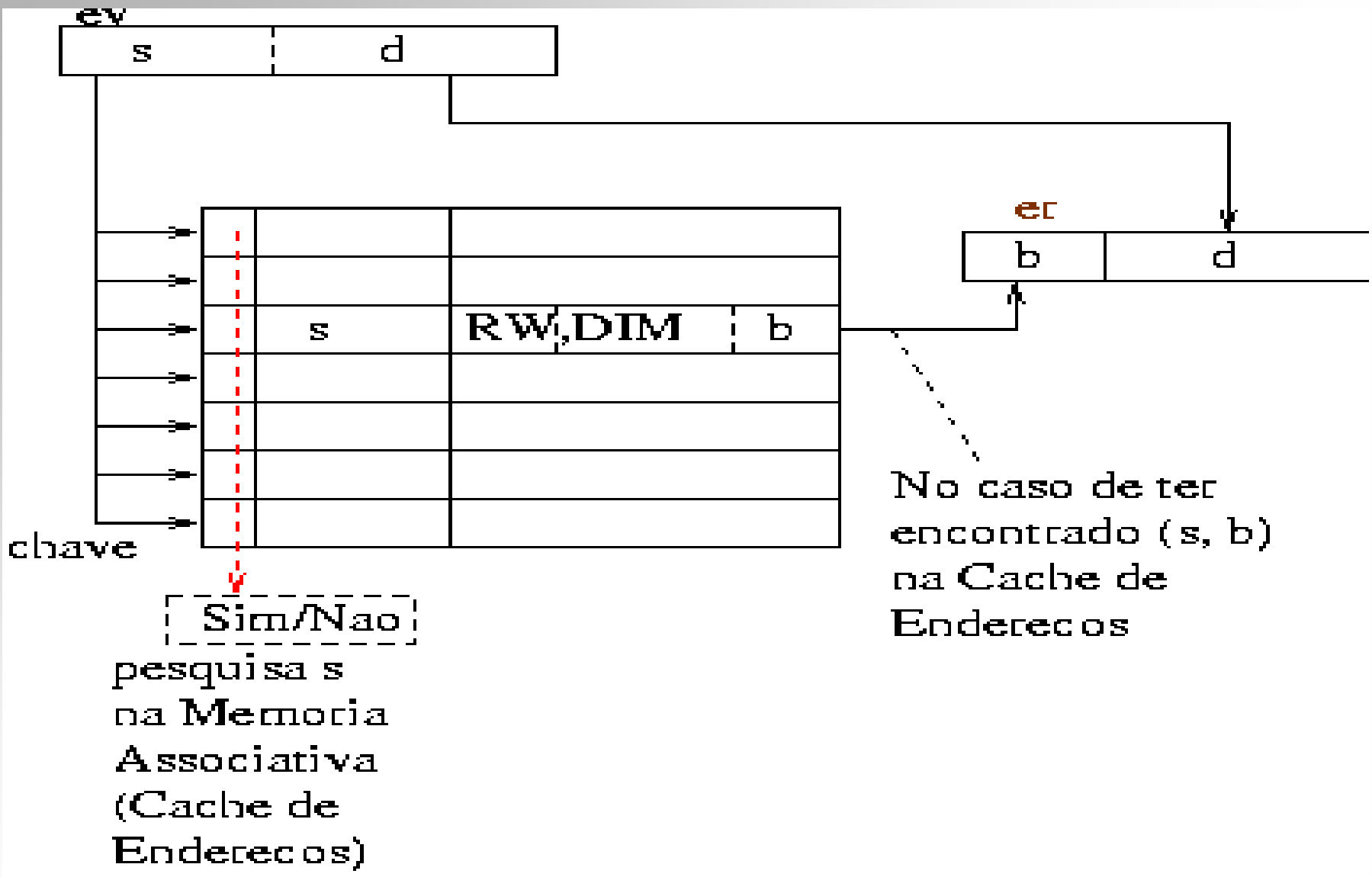
- estruturação lógica do Espaço de Endereços Virtuais
- segmentos de dimensão variável
- possível protecção por hardware, a nível do segmento
- segmentos possivelmente partilhados
- pré-carregamento dos segmentos em memória

ou segmentação dinâmica, por pedido

- gestão de memória dificultada, devido a fragmentação externa (partições de tamanho variável)

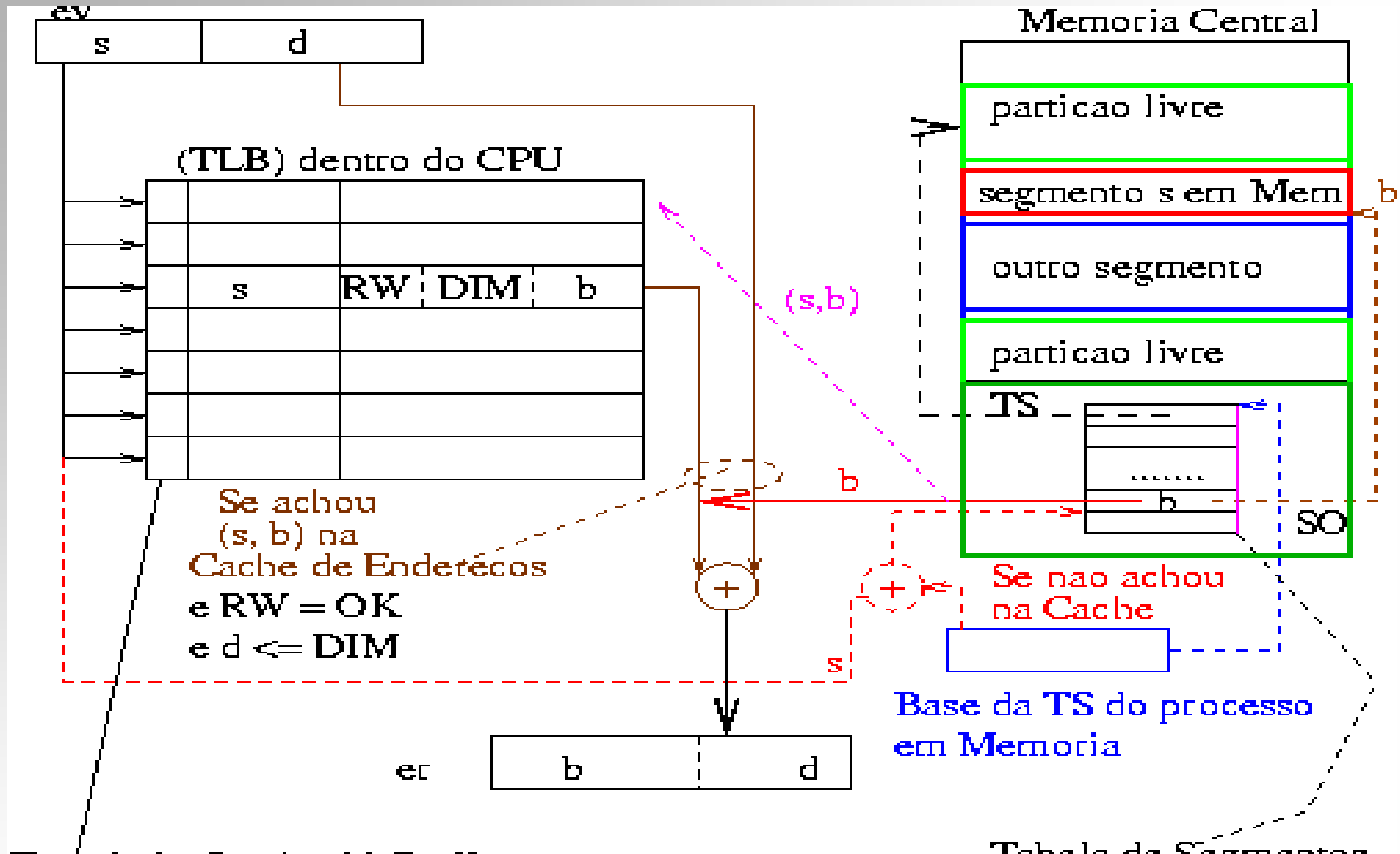

Segmentos Virtuais
S0
d
S1
endereco virtual
s
d
nº do segmento
deslocamento dentro do segmento
indica um segmento do programa
Transformacao de enderecos pela unidade T
(s, d) ——> (b, d)
b
d
base do segmento em Memoria
Memoria
Segmento S0 carregado em Memoria
d
b
T3: EVirtual Segmentado
——>
Ereal com Particoes de Tamanho Variavel

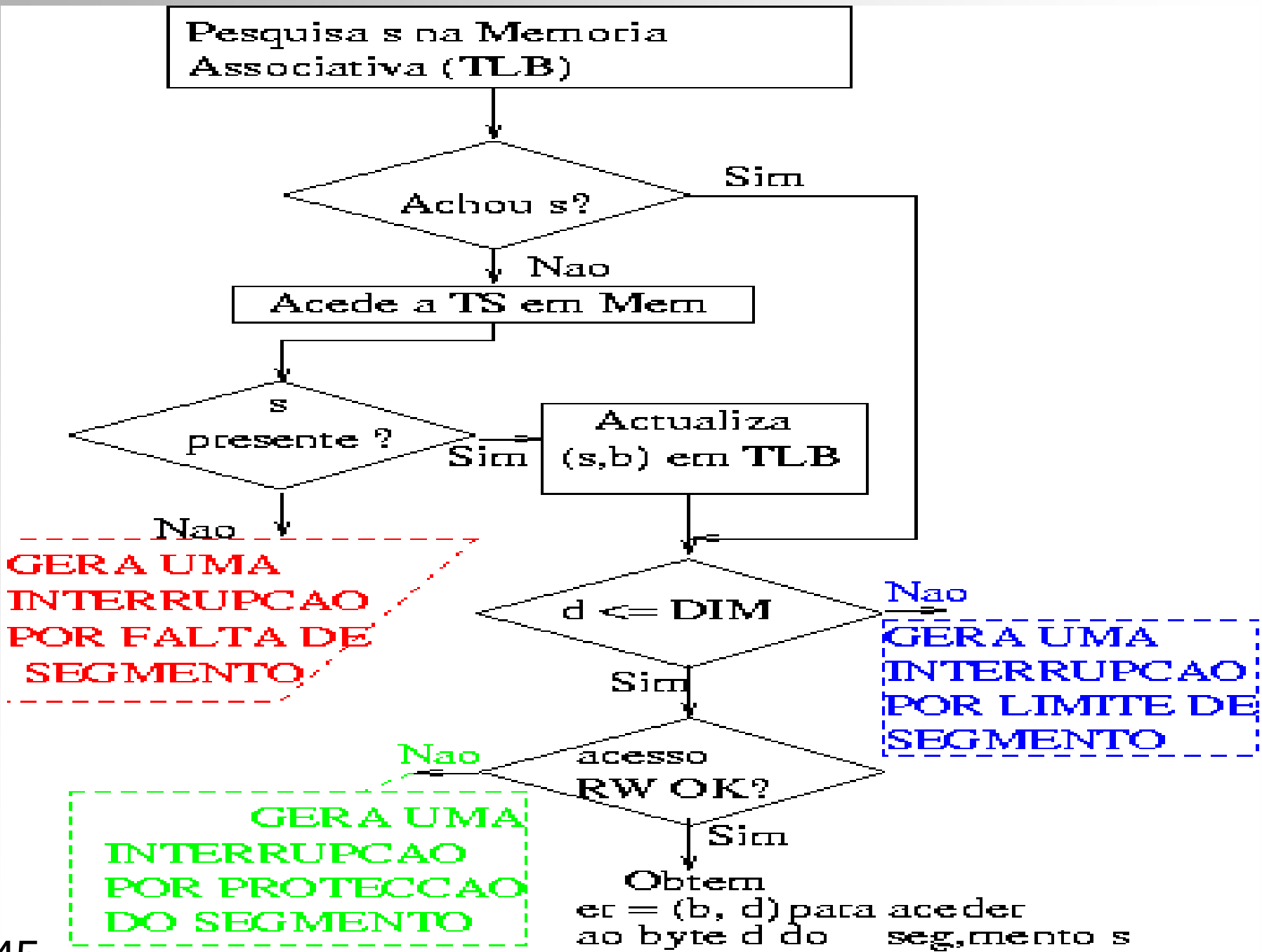

ev
s
d
b
d
er
RW
Dim
b
TabeladeSegmentos
do processo
LIVRE
d
segmento real
b
outro segmento
em memoria
LIVRE
outro segmento
em memoria
Memoria Central
RegBase da Tabela de Segmentos
do Processo em Memoria



Chave: nº do segmento virtual

Conteudo: base do segmento em Memoria


ev
s
d
(TLB) dentro do CPU
s
RW
DIM
b
(s,b)
Memoria Central
particao livre
segmento s em Mem
b
outro segmento
particao livre
TS
.....
b
SO
b
Se achou
(s, b) na
Cache de Endetécos
e RW = OK
e d <= DIM
s
Se nao achou
na Cache
Base da TS do processo
em Memoria
er
b
d
TranslationLookasideBuffer
(TLB) dentro do CPU
— contem os pares (s, b) dos
segmentos que foram referenciados
nas ultimas vezes pelo programa
Tabela de Segmentos
do Processo corrente
em Memoria


Pesquisa s na Memoria Associativa (TLB)
Achou s?
Sim
Nao
Acede a TS em Mem
s presente ?
Sim
Actualiza (s,b) em TLB
Nao
GERA UMA INTERRUPCAO POR FALTA DE SEGMENTO
d <= DIM
Nao
GERA UMA INTERRUPCAO POR LIMITE DE SEGMENTO
Sim
acesso RW OK?
Nao
GERA UMA INTERRUPCAO POR PROTECCAO DO SEGMENTO
Sim
Obtem er = (b, d) para aceder ao byte d do seg,mento s

Transformação T4: EV Segmentado → ER: Paginação

Reune as vantagens de:

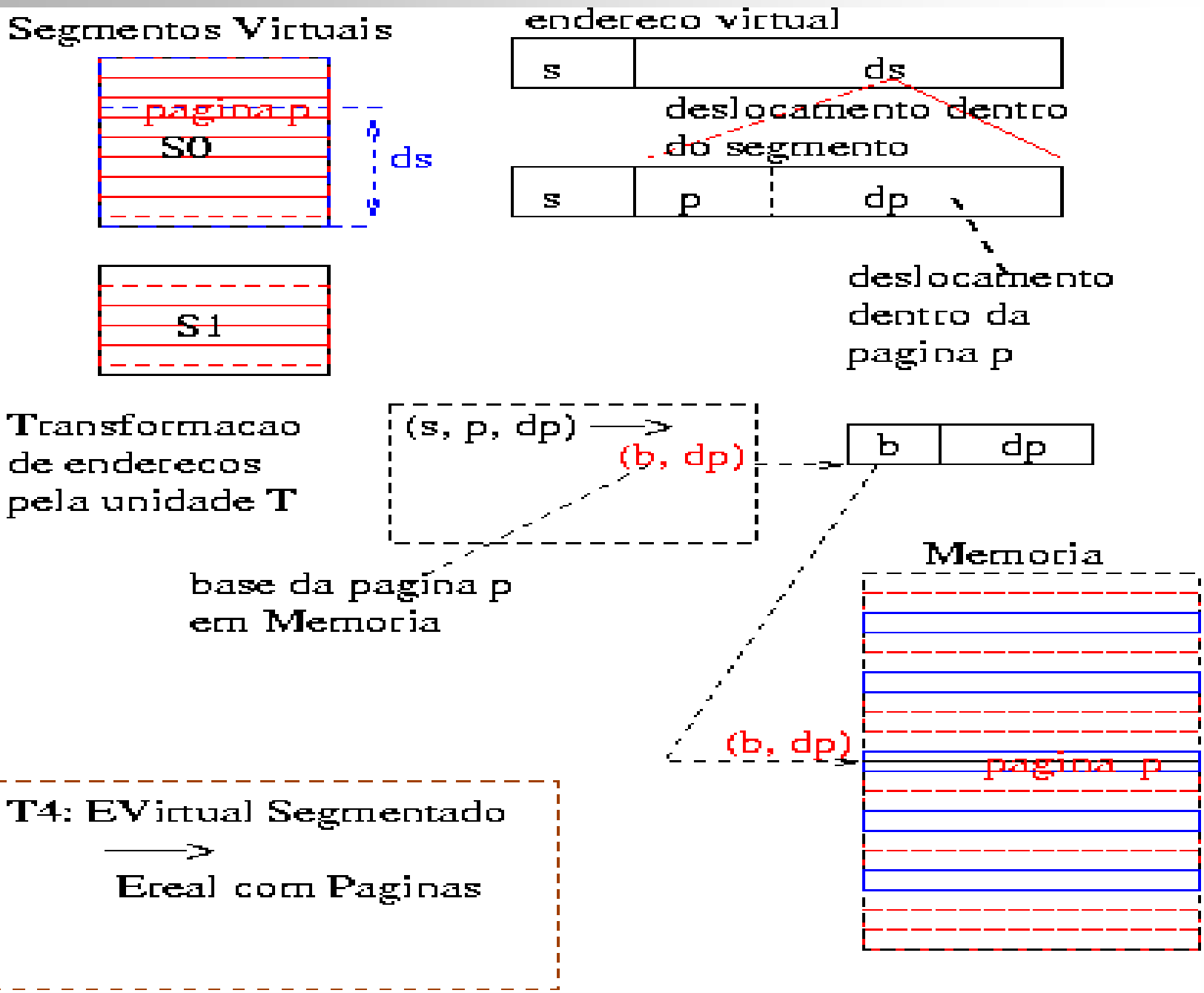
- segmentação: do ponto de vista da estrutura do Espaço de Endereços Virtuais.
- paginação: do ponto de vista da gestão de memória e do suporte de Memória Virtual.

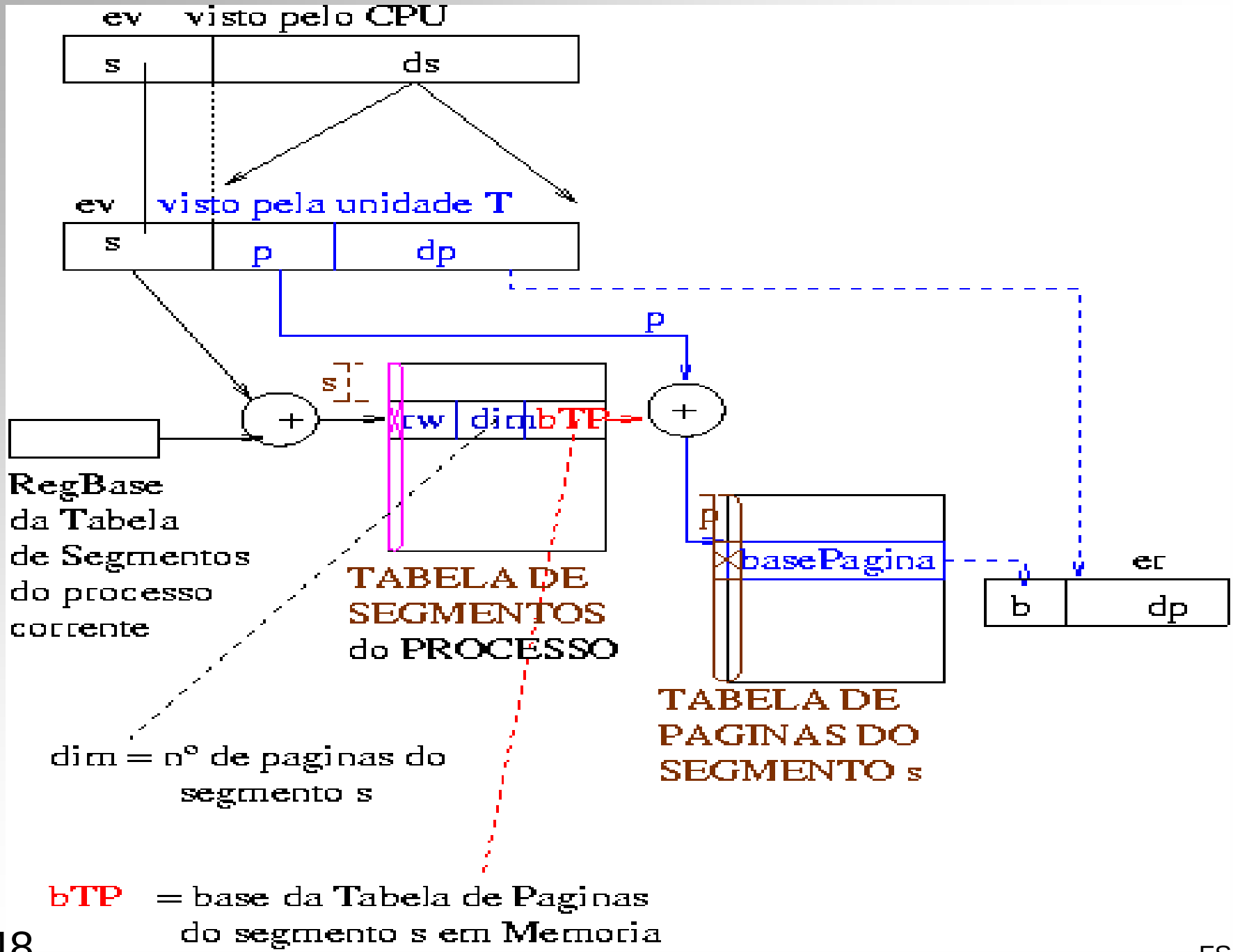
O programa é estruturado em Segmentos.

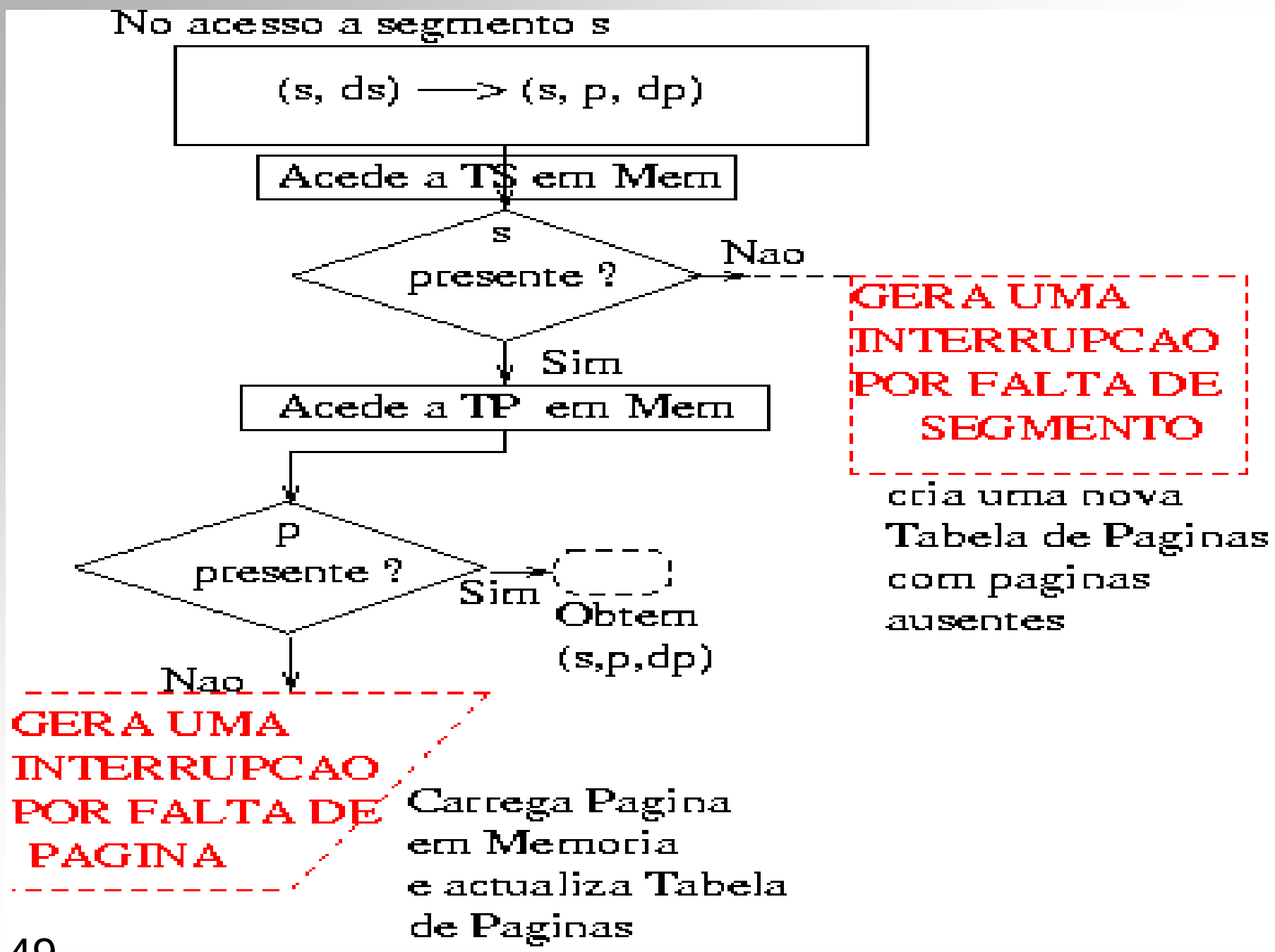
Os Segmentos são considerados subdivididos em Páginas pela unidade T:

→ as páginas de cada segmento não precisam de ser carregadas em posições contíguas de memória central

→ não exige que todo um segmento esteja carregado em memória central: só precisam de se manter em memória, as páginas correntemente em uso.


Segmentos Virtuais
pagina p
S0
ds
S1
endereco virtual
s
ds
deslocamento dentro
do segmento
s
p
dp
deslocamento
dentro da
pagina p
Transformacao
de enderecos
pela unidade T
(s, p, dp) ⟶
(b, dp)
b
dp
base da pagina p
em Memoria
Memoria
(b, dp)
pagina p
T4: EVirtual Segmentado
⟶
Ereal com Paginas


ev visto pelo CPU
s
ds
ev visto pela unidade T
s
p
dp
p
s
rw
dim
bTP
RegBase
da Tabela
de Segmentos
do processo
corrente
TABELA DE
SEGMENTOS
do PROCESSO
p
basePagina
TABELA DE
PAGINAS DO
SEGMENTO s
b
dp
dim = nº de paginas do
segmento s
bTP = base da Tabela de Paginas
do segmento s em Memoria


No acesso a segmento s
(s, ds) ——> (s, p, dp)
Acede a T$ em Mem
s
presente ?
Nao
GERA UMA INTERRUPCAO POR FALTA DE SEGMENTO
cria uma nova Tabela de Paginas com paginas ausentes
Sim
Acede a TP em Mem
p
presente ?
Sim
Obtem (s,p,dp)
Nao
GERA UMA INTERRUPCAO POR FALTA DE PAGINA
Carrega Pagina em Memoria e actualiza Tabela de Paginas

Algoritmos de gestão de memória

Sem paginação

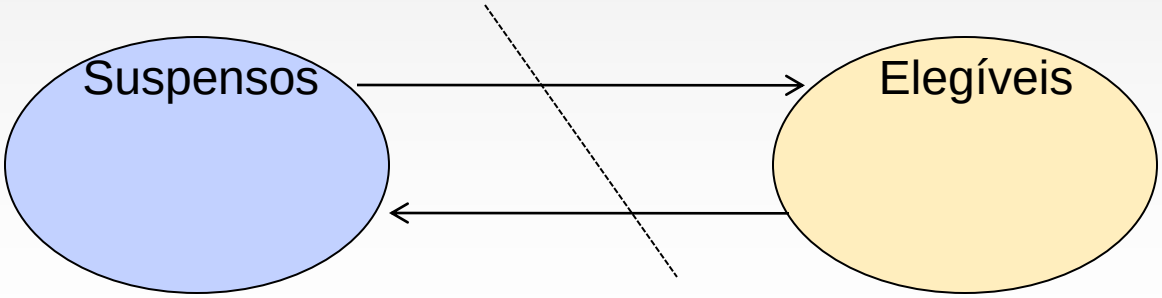
partições de tamanho variável

Com paginação

páginas de tamanho fixo

Sem varrimento para disco (swapping)

Com varrimento de/para disco



Algoritmos de gestão de memória

Multiprogramação

uso da memória – uso do CPU

Ex: se cada processo só utiliza 20% do tempo CPU
enquanto está carregado em memória
se na memória cabem os mapas de 5 processos
e se os processos não se bloqueiam por I/O

Então o CPU estaria utilizado a 100% do tempo

Mas... os processos bloqueiam-se por I/O...

Algoritmos de gestão de memória

modelos probabilísticos (aproximações):

Se um P bloqueado uma fracção p do tempo

se há n processos multiprogramados

Probabilidade de que os n processos estejam simultaneamente bloqueados por I/O: p^n

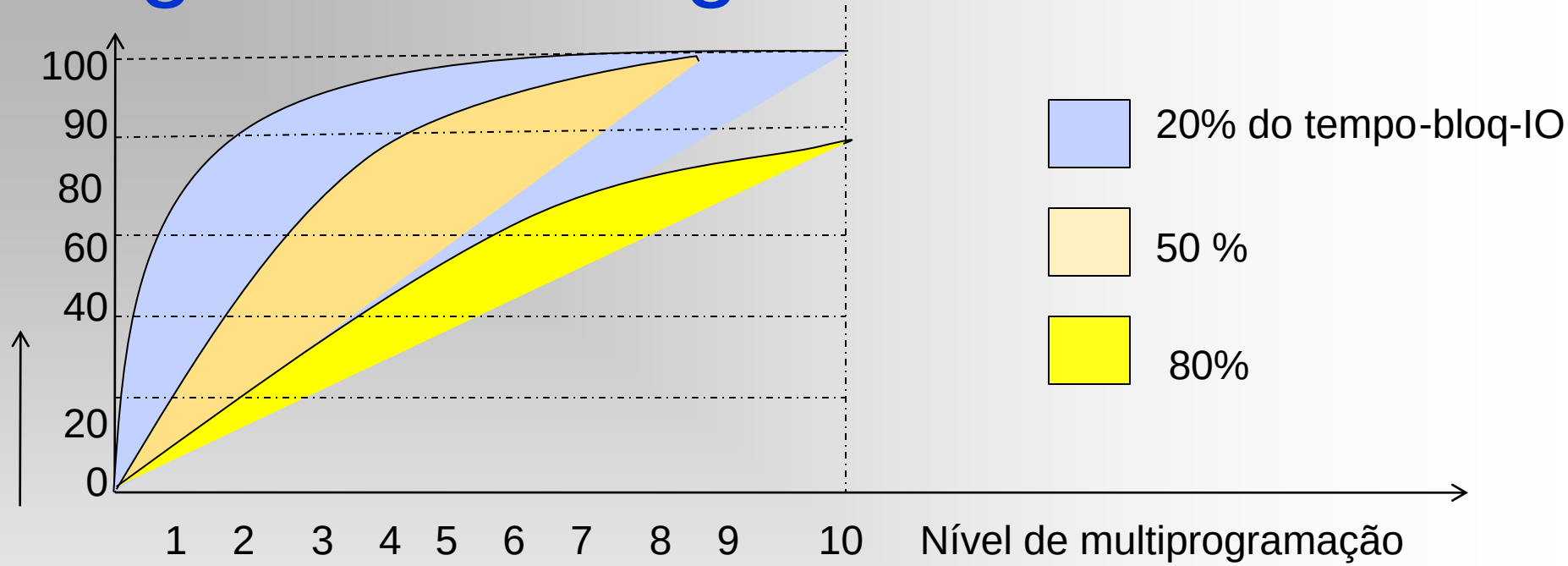
e nesse caso o CPU estará desocupado

A utilização do CPU será $1 – p^n$

(se Ps fossem independentes)

(Consultar: Modern Operating Systems, A. Tanenbaum)

Algoritmos de gestão de memória



Utilização CPU (%)

--- se os processos ficam 80% do tempo bloqueados em IO
um mínimo de 10 processos devem residir em Memória
para que a taxa de utilização do CPU seja pelo menos 90%

Na prática, tempos da ordem de 80% de bloqueio são vulgares
seja em regime batch ou interactivo time-sharing

(ver: Modern Operating Systems, A. Tanenbaum)

Algoritmos de gestão de memória

Na verdade os processos não são independentes:

havendo só um CPU

→ só se pode ter 1 processo Executando!

É preciso um modelo mais que tenha isto em conta:

baseado na Teoria das filas de espera

(Queueing Theory)

Mas o modelo aproximado pode ser usado para obter estimativas ainda que menos exactas.

Algoritmos de gestão de memória

Exemplo: Memória: 1 GB

OS + = 200 MB

1 Processo U = 200 MB

80% do tempo bloqueado IO

Se n = 4 → Utilização do CPU = $1 - 0.8^4$ → cerca de 60%

Adicionar mais 1GB à Memória?

pode aumentar-se 5 ++ n = 9

→ Utilização do CPU = $1 - 0.8^9$ → cerca de 89%

um aumento de cerca de 45% face ao anterior!

Adicionar mais 1GB à Memória?

pode aumentar-se 5 ++ n = 14

→ Utilização do CPU = $1 - 0.8^{14}$ → cerca de 98%

um aumento de cerca de 10% face ao anterior!

Algoritmos de gestão de memória

Ver Livro: Modern Operating Systems

A. Tanenbaum

+ análise de desempenho

Algoritmos de gestão de memória

Multiprogramação com partições fixas

fácil gestão mas ineficiente

sempre que uma partição fica livre:

escolhe o 1º processo que caiba

para evitar fragmentação interna

escolher o maior processo na fila, que caiba

→ prejudica os processos mais pequenos!

que deveriam ser beneficiados!!

Algoritmos de gestão de memória

Multiprogramação com partições variáveis

exige compactação de memória

quanta memória atribuir a cada processo?

i) tamanho fixa, quando se carrega o mapa

ii) tamanho variável, durante a execução

→ os segmentos podem crescer/contrair dinamicamente

pode exigir:

mover o processo para uma partição maior!

varrer para disco um outro processo (menos prio)

para libertar espaço!

se não puder: matar o processo!

Alternativa: atribuir maior partição do que a necessária ao mapa

para antever variações dinâmicas

Algoritmos de gestão de memória

Listas ligadas: de partições reservadas ou livres

como ordenar as listas?

por endereço:

fácil de actualizar quando um P termina

possível fusão de partições vizinhas

como encontrar partição adequada para um novo P?

a) First-Fit: até achar um bloco de tamanho suficiente

se encaixa – OK, senão: divide em 2 partes: reservada + livre

junta a parte livre a uma lista de blocos livres

b) Next-Fit: como a), mas memoriza o último ponto da lista em que ficou, na pesquisa anterior e inicia daí a nova pesquisa

(→ pior desempenho do que a))

c) Best-Fit: pesquisa toda a lista até achar o menor bloco livre adequado!

p/ evitar fragmentar os blocos, mas é mais lento do que a)

tende a produzir muitos blocos livres muito pequenos... Muitas migalhas...

Algoritmos de gestão de memória

Avaliar os algoritmos?

- complexidade
- simulação com cenários concretos

Ver: Modern Operating Systems, A. Tanenbaum

Algoritmos de gestão de memória

Gestão baseada no *Buddy system* *(Knuth 73)*

apressar a fusão de blocos livres vizinhos quando 1 P termina

manter a vantagem da atribuição rápida para listas ordenadas por tamanho dos blocos

listas de blocos livres de tamanhos

1,2,4,8,16, ..., até ao tamanho total da memória

Quando 1 bloco de 2^k é libertado:

pesquisar a lista de 2^k blocos para ver se pode haver fusão

Mas é ineficiente em utilização da memória: arredonda os pedidos para a potência de 2 → fragmentação interna

Algoritmos de gestão de memória

Com paginação:

Algoritmos de substituição de páginas

Óptimo: quando ocorre 1 falta de página ---

das páginas residentes:

1 delas será referida na próxima instrução

outras só o serão daqui a muitas instruções....

→cada página tem uma etiqueta com o número de instruções que faltam executar antes de ser referida!

-- remover a página com a maior etiqueta...

Como implementar isto?

executar 1 vez, registar o traço de referências

e depois aplicar isto a partir da 2ª execução!!!

Algoritmos de gestão de memória

Not recently used (NRU):

cada página tem 2 bits de controlo:

M – posto a 1 por hardware quando é escrita

R – posto a 1 por hardware quando é referida

Os bits só voltam a 0 quando o SO faz reset deles.

→ Ou simular esses bits por misto hardware / software

Quando P inicia: todas entradas da TP marcadas ausentes

página ref → falta de página: o SO põe bit R a 1

carrega a página

põe permissão a READONLY e repete a instrução

página escrita → falta de página; SO põe bit M a 1

muda permissão para READ/WRITE

Algoritmos de gestão de memória

NRU: quando 1 P inicia todos bits R e M na TP a 0

Periodicamente (T) o bit R é posto a 0

(o SO assim distingue as páginas não refs nesse T)

Falta de página → SO analisa as páginas em 4 classes:

classe 0: não R, não M

1: não R, M (R foi posto a 0, no último T)

2: R, não M

3: R, M

-- remove uma página aleatoriamente,

mas a partir das classes inferiores

Algoritmos de gestão de memória

FIFO: -- com uma lista de todas as páginas residentes pela ordem de carregamento em memória

FIFO modificado – evita remover páginas antigas mas muito usadas: analisa as mais antigas mas começando a partir das classes inferiores...

FIFO com Segunda Oportunidade:

procura 1 página antiga que não foi referida no último T

-- analisa o bit R da página mais antiga:

se é 0 – remove-a

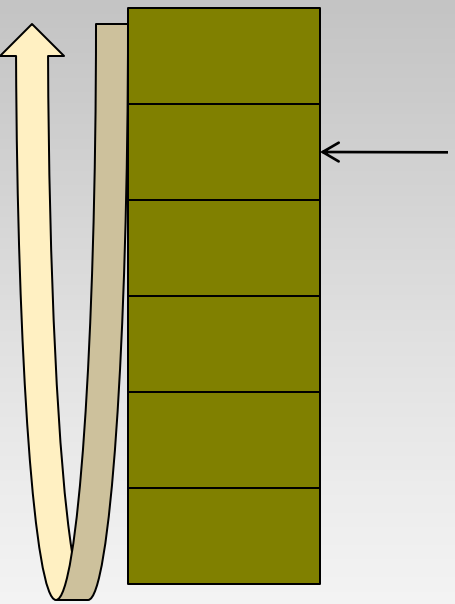
se é 1 – põe o bit R a 0 e põe a página no fim da lista de páginas como se tivesse acabado de ser carregada em memória

- neste caso continua a pesquisa até achar uma página com R = 0

Se todas as páginas foram refs no último T, equivale ao FIFO puro.

Algoritmos de gestão de memória

Pode ser implementado de forma eficiente com uma lista circular e um apontador circulante



Falta de página:

analisa a página apontada:

se R é 0 remove-a

se R é 1 põe-no a 0

e continua a pesquisa

Algoritmos de gestão de memória

LRU – Lest Recently Used

Quando há falta de página:

remove a página que não é usada há mais tempo

É pesado de implementar:

uma lista ligada das páginas residentes

a página mais recente à cabeça, a menos recente na cauda

a lista actualizada a cada ref de memória!

Uso de hardware especial:

1 contador C de 64 bits, automatica/ incrementado após cada instrução

cada descritor de página na TP: contém o valor do contador

após cada ref de memória: valor corrente de C é posto na TP

quando há falta de página: remove a página com menor valor de C

Algoritmos de gestão de memória

Simular LRU por software

→ NFU not frequently used

1 contador C software por cada página, inicial 0

periodicamente: o SO, para cada página em memória,

soma o bit R (0 ou 1) ao C

C regista quantas refs houve àquela página

quando há falta de página:

remove a página com menor C

nunca esquece nada…. Está sempre a somar…

Algoritmos de gestão de memória

NFU modificado: um mecanismo de envelhecimento

antes de somar o valor de R

os C sofrem um desvio de 1 bit à direita,

e o bit R é somado ao bit mais significativo
e não ao menos significativo

quando há falta de página:

remove a página com menor C

Algoritmos de gestão de memória

Conceito de Working Set (WS)

Princípio da Localidade

WS: conjunto de páginas que um processo usa correntemente

Se está todo em M → P não gera muitas faltas

até que se mova para outra localidade do programa

Se a M não chega para conter todos os WS de todos os P

- Cada P gera muitas faltas → pode conduzir a Trashing...

Como avaliar dinamicamente o WS de cada P?

para depois o carregar todo de 1 vez (após prévio swap)

-- pré-paginação

Dimensão de WS: se o tamanho total dos WS de todos os Ps em memória excede a M → pode haver trashing

exige controlo do nível de multiprogramação

Algoritmos de gestão de memória

Estratégias Locais versus Globais

Local : dar uma partição fixa de M a cada P

Global: afectar dinamicamente blocos de um pool global entre os Ps elegíveis

-- operam melhor, quandos os WS podem variar dinamica/

Locais: se o WS cresce → pode ocorrer trashing desse P

(mesmo que M tenha blocos livres)

se o WS decresce → desperdício de M

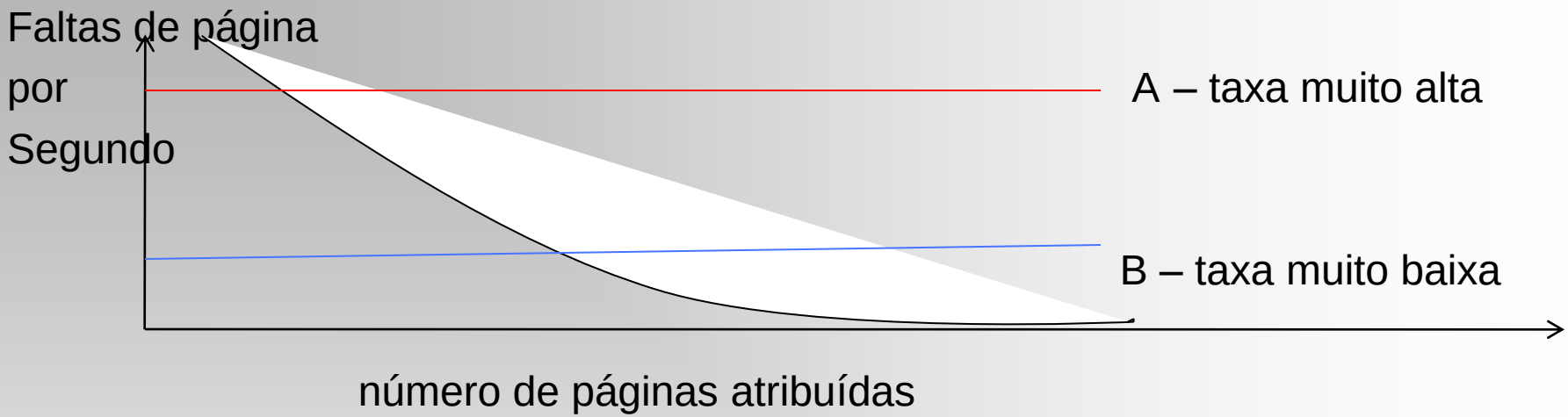
Globais: o SO deve continuamente decidir quantos blocos atribuir a cada WS de cada P

-- via NFU com aging: pouca exactidão: mudanças microseg versus T ~ ms

Ou

.. Via PFF – PAGE FAULT FREQUENCY

Algoritmos de gestão de memória



Em A: deve aumentar-se o WS para reduzir a taxa de faltas

Em B: deve diminuir-se o WS para libertar M não em uso

PFF visa manter a taxa de Faltas dentro dos limiares A e B.

Se há tantos P em M que não se consegue garantir todos com F abaixo de A

→ remover algum WS todo de um P para disco

e libertar suas páginas para o pool global, para uso por outros Ps...

Consultar: Modern Operating Systems, A. Tanenbaum

Referências:

Modern Operating Systems, A. Tanenbaum

Sistemas Operativos, J. Alves Marques et al.

Sistemas de Exploração, J. Cardoso e Cunha