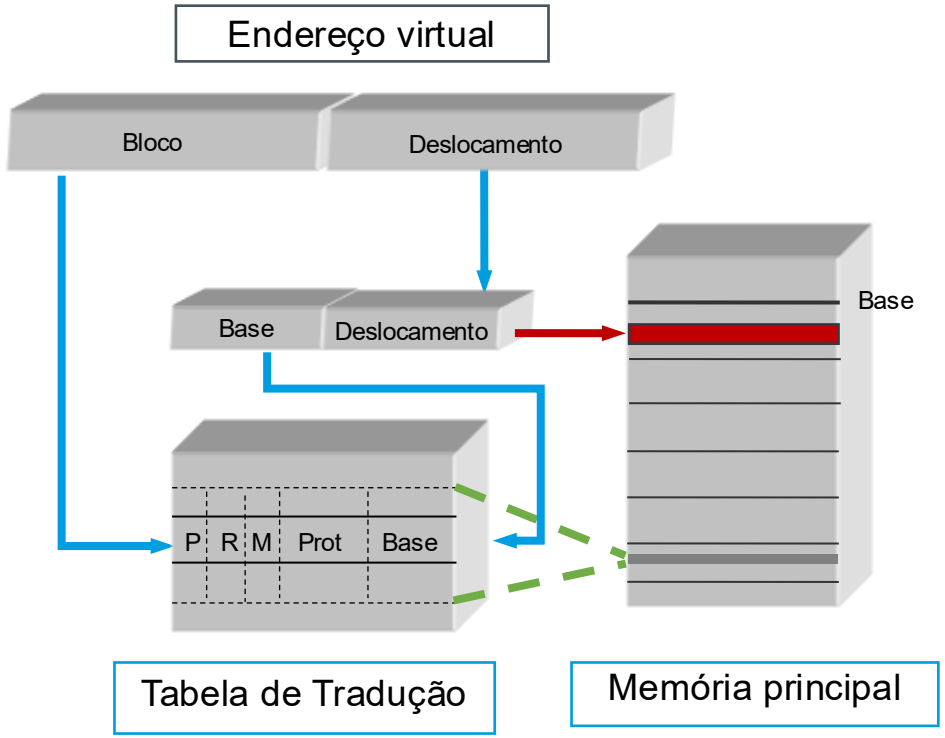
TÉCNICO LISBOA

Memória Virtual Eficiente

Memória virtual duas vezes mais lenta?



- A tabela de tradução está em memória primária, se a MMU tiver de lhe aceder de cada vez que o programa gera um endereço, o custo do acesso a uma posição de memória duplica

- Grosso modo **torna os programas duas vezes mais lentos** do que se fossem executado com endereçamento real.

TÉCNICO LISBOA

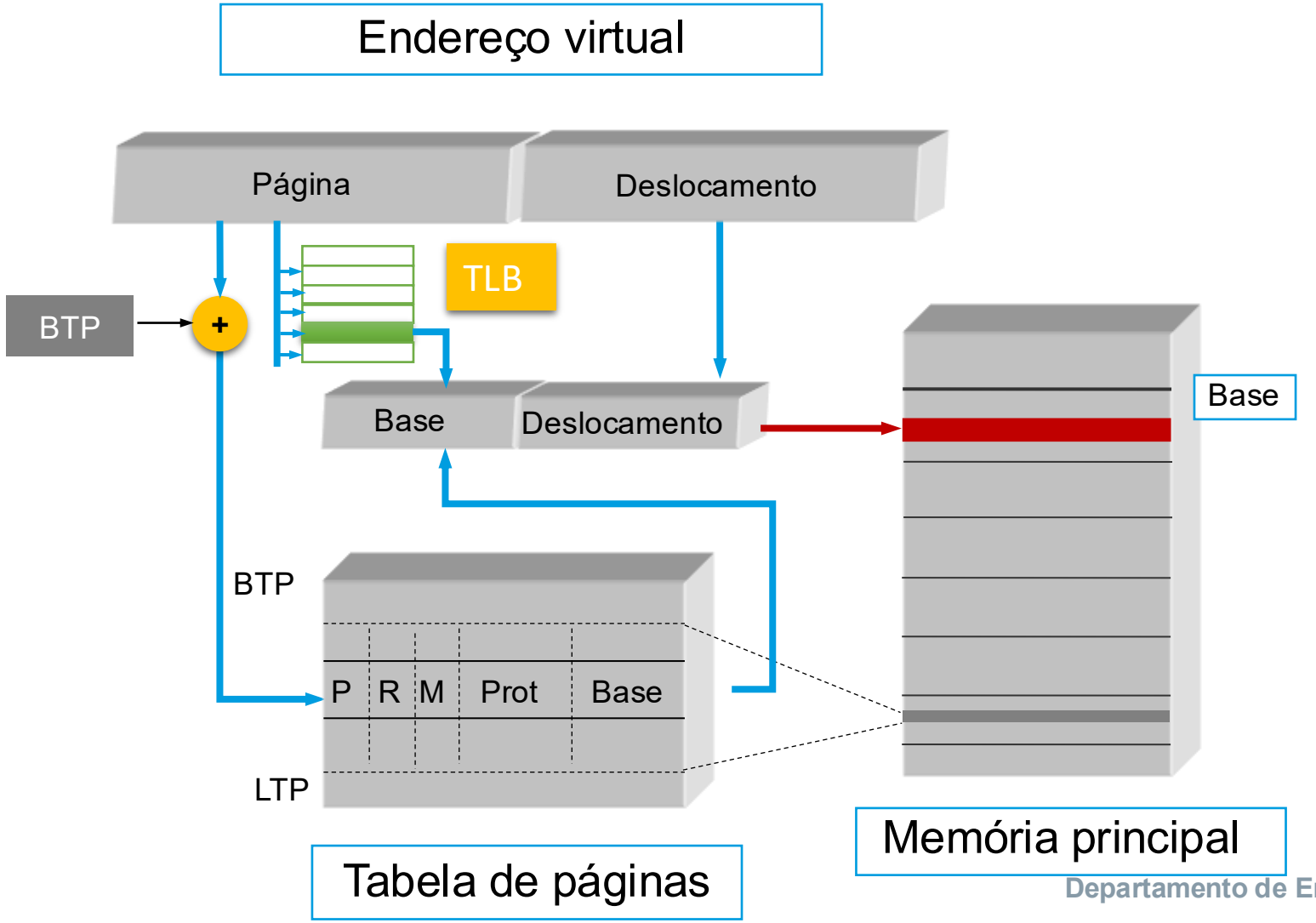
Como otimizar a tradução de endereços?

Cache de endereços base dos blocos!

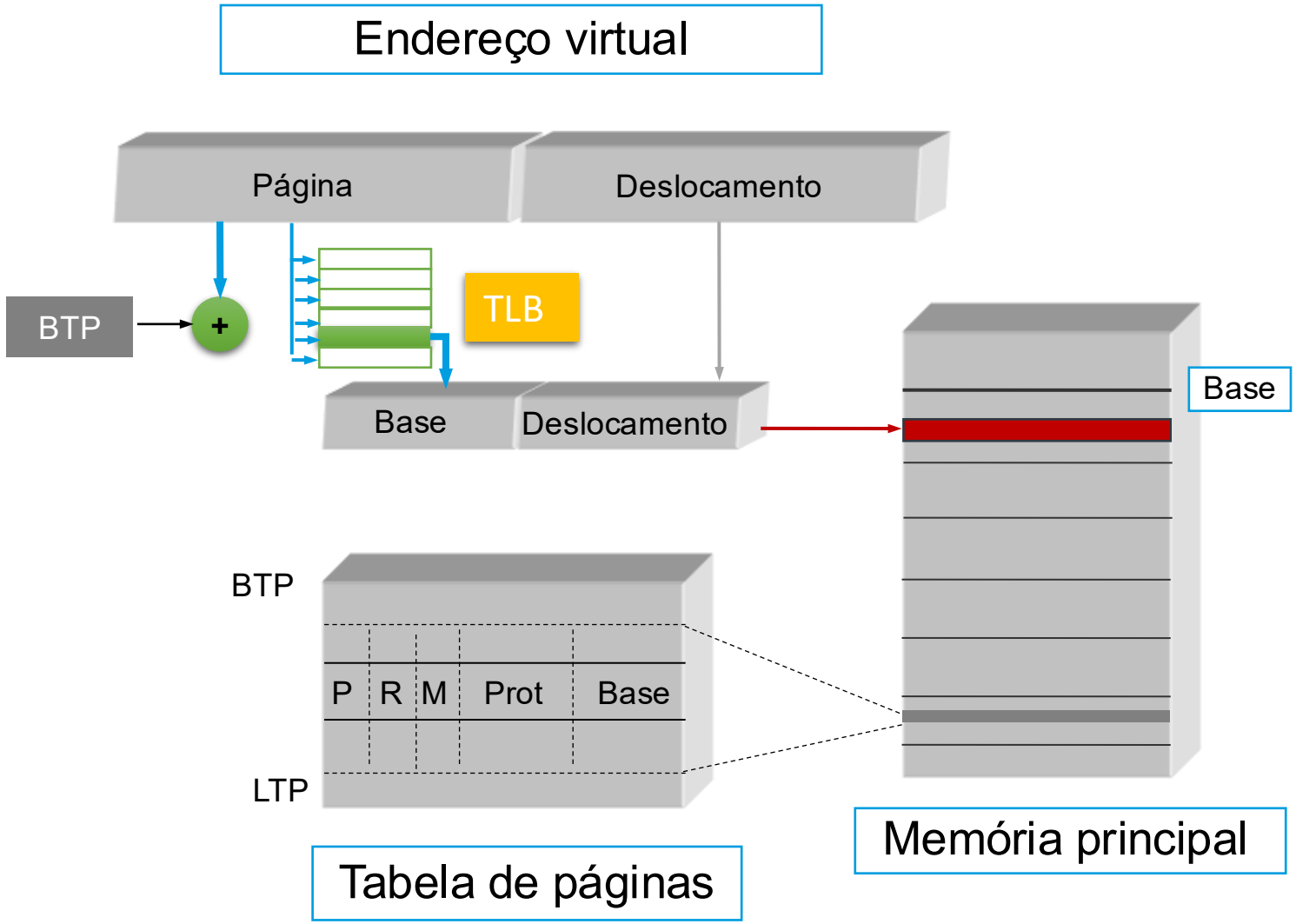
TLB - Translation Lookaside Buffer

- O TLB é uma memória associativa com N linhas, cada uma com:
  1. o endereço da pagina virtual
  2. a base da página na memória física
  3. os bits de proteção.
- Associado a cada endereço de página existe um comparador hardware:
  - se detetar uma igualdade(*TLB hit*) retorna a base da página e os bits de proteção
    - Evita-se o acesso à tabela das páginas em RAM
  - Em caso de TLB-*miss:*
    - Se a página estiver em RAM, o TLB é atualizado pelo MMU
    - Se houver um page fault, cabe ao SO atualizar a TLB
- A TLB faz parte da MMU:
  - É consultada automaticamente pelo *hardware*

Tradução dos endereços virtuais

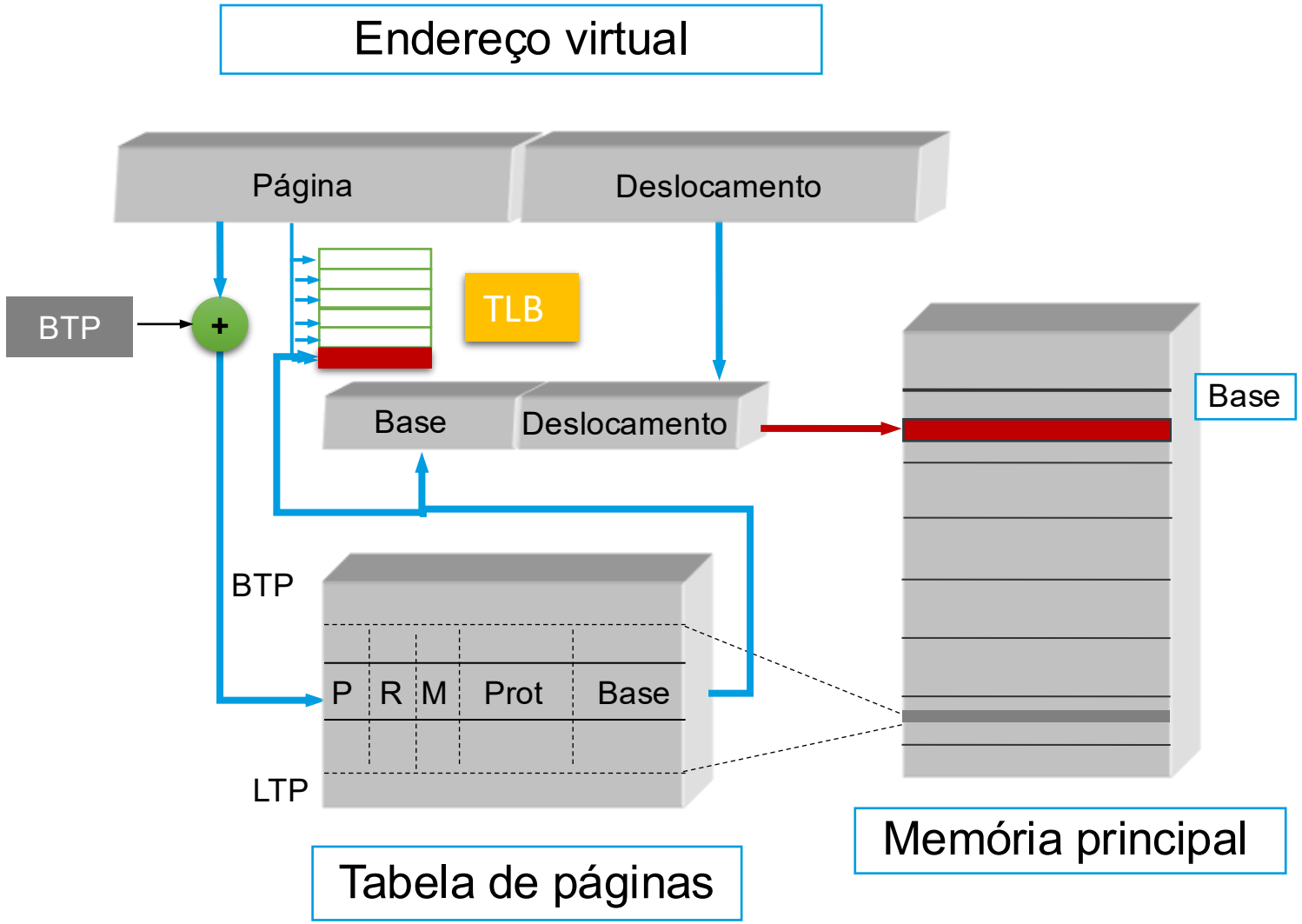


Paginação: TLB *hit*



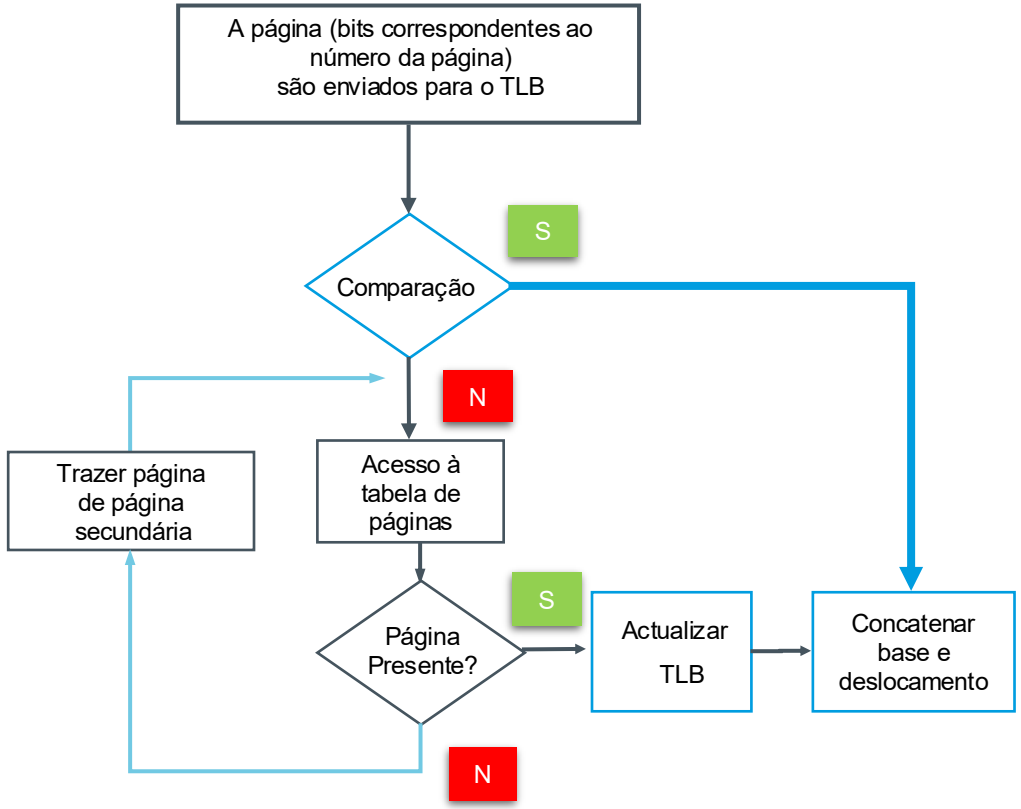
Funcionamento do TLB por hardware muito rápido 0,5 a 1 ciclo de relógio

Paginação: *TLB miss*



Sequência de resolução do endereço

- A pesquisa na TLB é ***lançada em paralelo*** com o acesso à tabela de páginas:
  - Se o descritor for encontrado na TLB, é interrompido o acesso à tabela de páginas.
  - Se não for encontrado, a tabela de páginas é acedida e o descritor é introduzido na TLB, geralmente segundo uma disciplina FIFO



TLB (1/2)

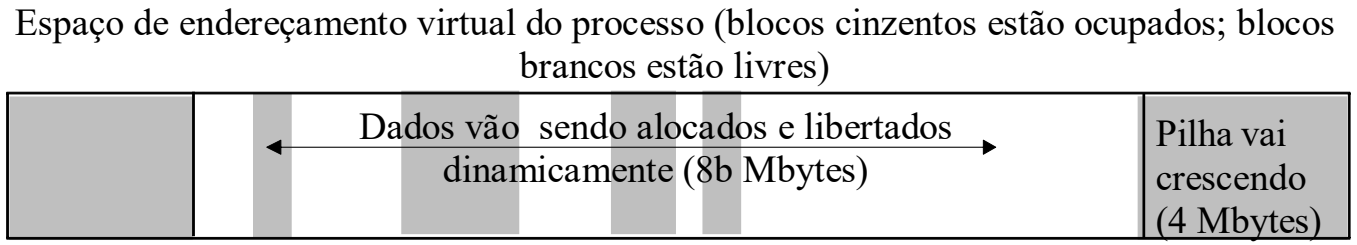
- O ideal seria guardar na TLB não as últimas, mas as próximas páginas a que o programa irá aceder.

- Como isso é impossível de prever:
  - toma-se o funcionamento recente do programa como uma boa previsão para o que ele fará no futuro próximo.
  - se um programa acedeu a uma página, é expectável que os próximos acessos sejam dentro da mesma página.

TLB (2/2)

- A dimensão destas tabelas é pequena, em geral (64, 128 entradas), pois o seu custo é elevado:
  - a sua dimensão é cuidadosamente testada de forma a obter percentagens de sucesso muito elevadas (90-95%)
  - um factor que também influencia a dimensão da TLB é o *quantum* dos processos.
    - a TLB é limpa em cada comutação de processos
    - quanto maior for o *quantum*, maior é o número de páginas acedidas, o que leva à necessidade de ter mais entradas na TLB.

Quanto ocupa a tabela de páginas?



- Assumindo endereços de 64 bits e páginas de 4 Kbytes, quantas páginas pode ter o espaço de endereçamento de um processo?
  - $2^{64}/2^{12}=2^{52}$ páginas
- Assumindo que cada entrada na tabela de páginas (PTE) ocupa 4bytes, qual a dimensão da tabela de páginas?
  - $2^{2}*2^{52}=2^{54}$ bytes=16 Petabytes
- Irrealista assumir que tabela de páginas caberá sempre em memória primária!

Dimensão das Tabelas de Páginas

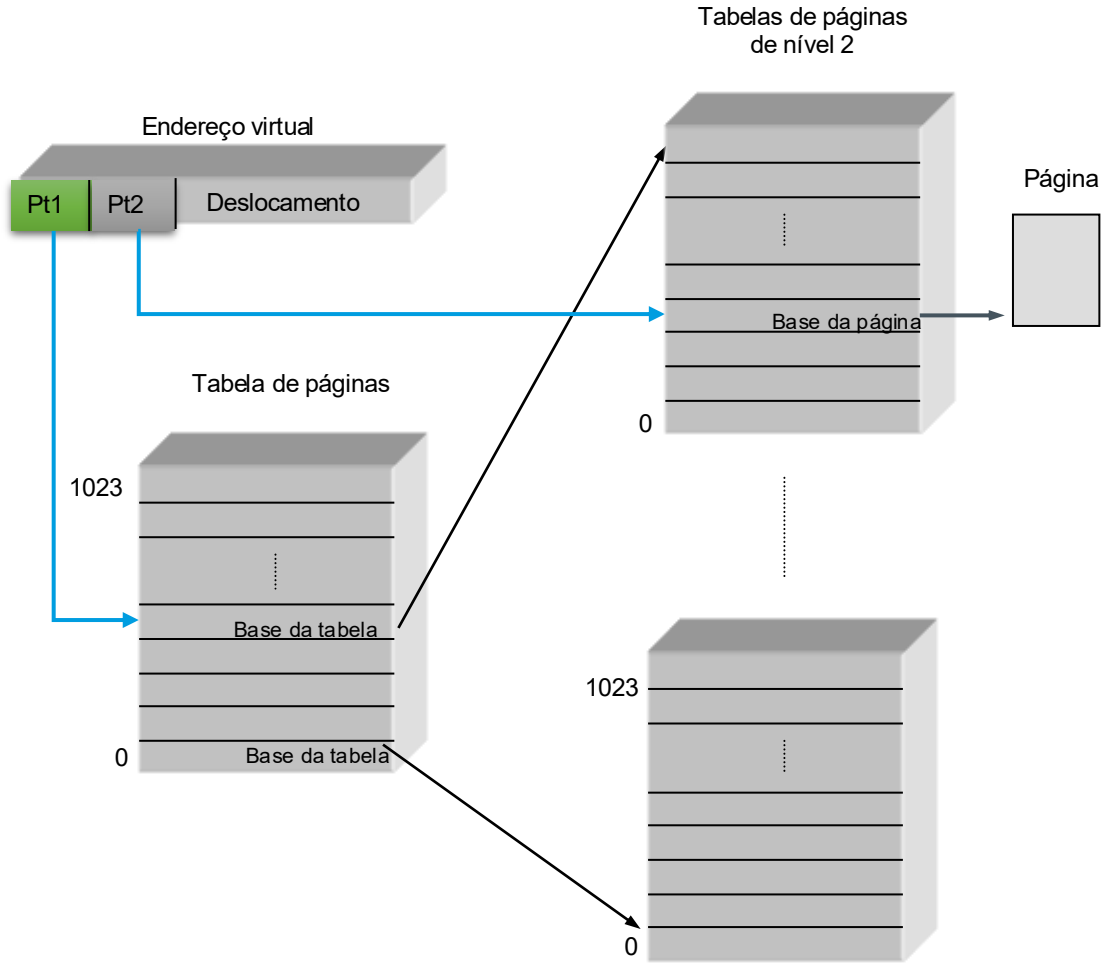
Subdividir as Tabelas de Páginas

- Solução típica para o problema da dimensão

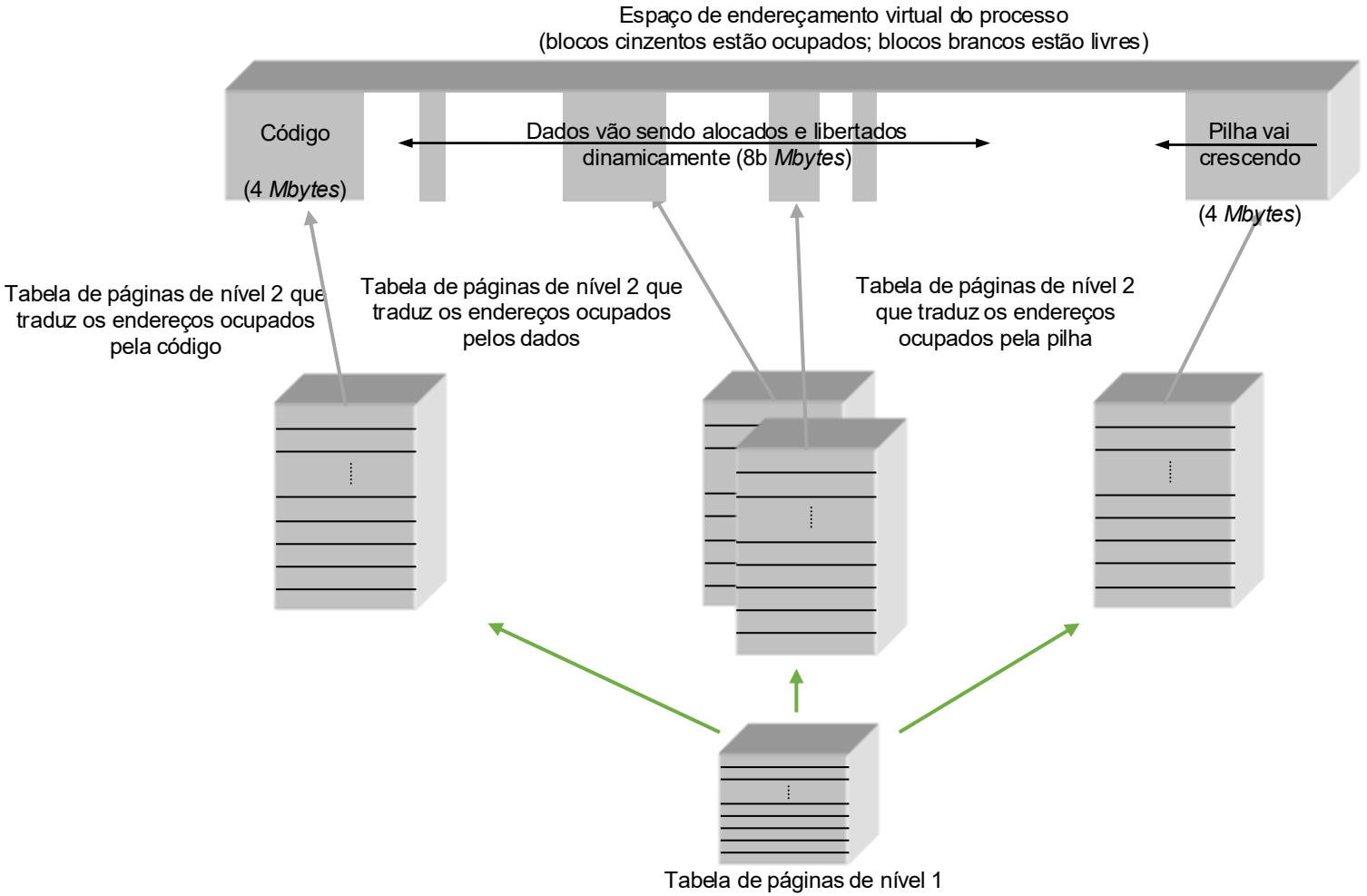
- Exemplo para endereço virtual de 32 bits
  - 10 bits para Pt1
  - 10 bits para Pt2
  - 12 bits para deslocamento
  - Cada entrada na tabela ocupa 2^2 bytes
    - Cada tabela (PT1 e PT2) ocupa $2^{10} \times 2^2 = 2^{12}$ B ou seja 4KB

- Vantagens:
  - apenas têm de estar em memória primária, as tabelas de páginas correspondentes às páginas que estão de facto a ser utilizadas pelo processo

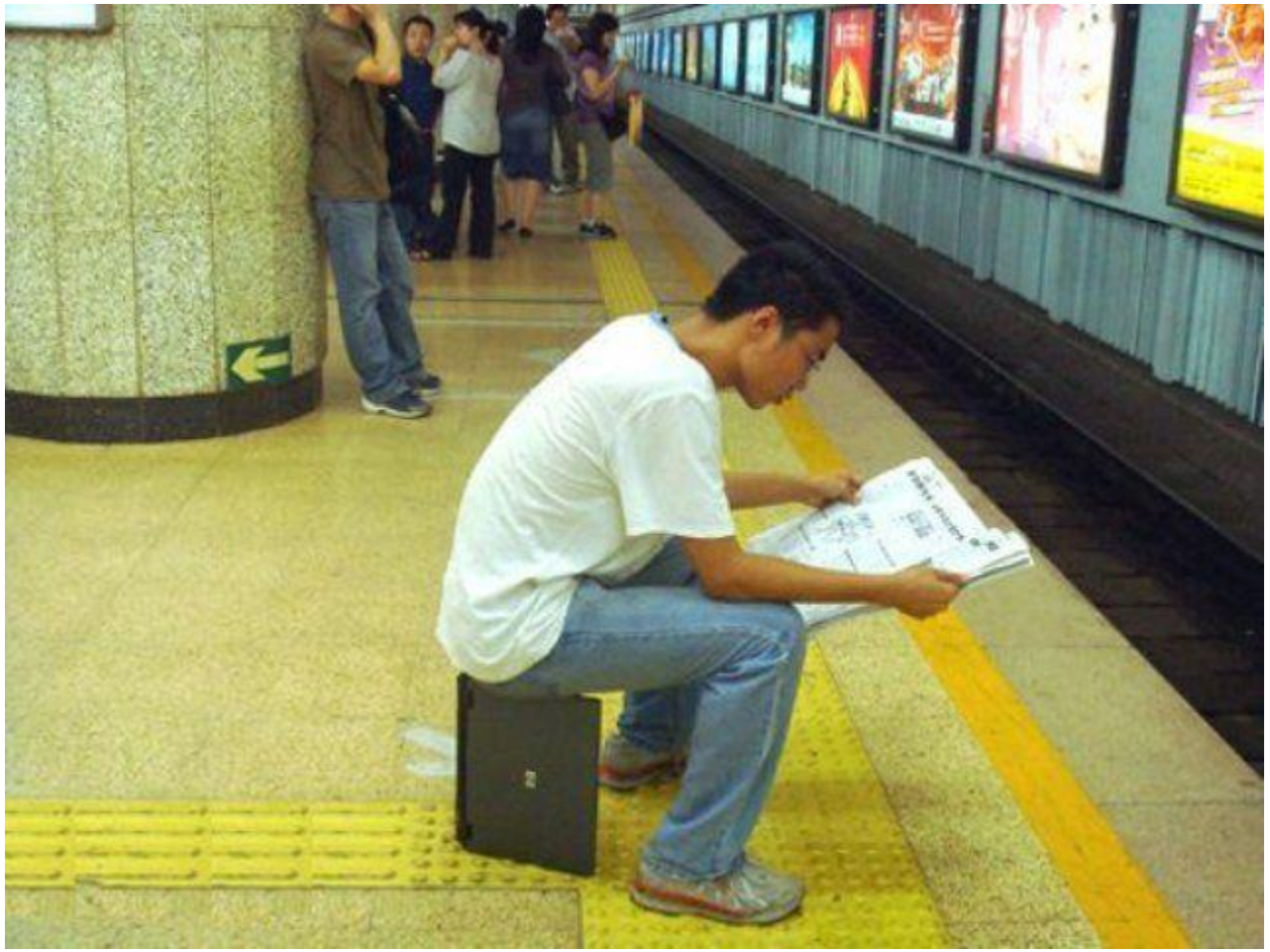
  - Uma tabela ocupa uma pagina física de 4 KB facilitando a gestão



Tabelas de Páginas Multinível



- A solução ocupa mais espaço que uma tabela compactada, mas torna mais simples a criação dinâmica de novas zonas no espaço de endereçamento
- Simplifica muito a gestão da memória primária onde as tabelas de páginas são armazenadas
- Para o funcionamento do TLB é transparente se o endereço tem ou não subdivisões
- Obviamente torna a tradução do endereço mais complexa quando não está no TLB (pode necessitar de 2 acessos a memória)
- Tipicamente é executado pela MMU

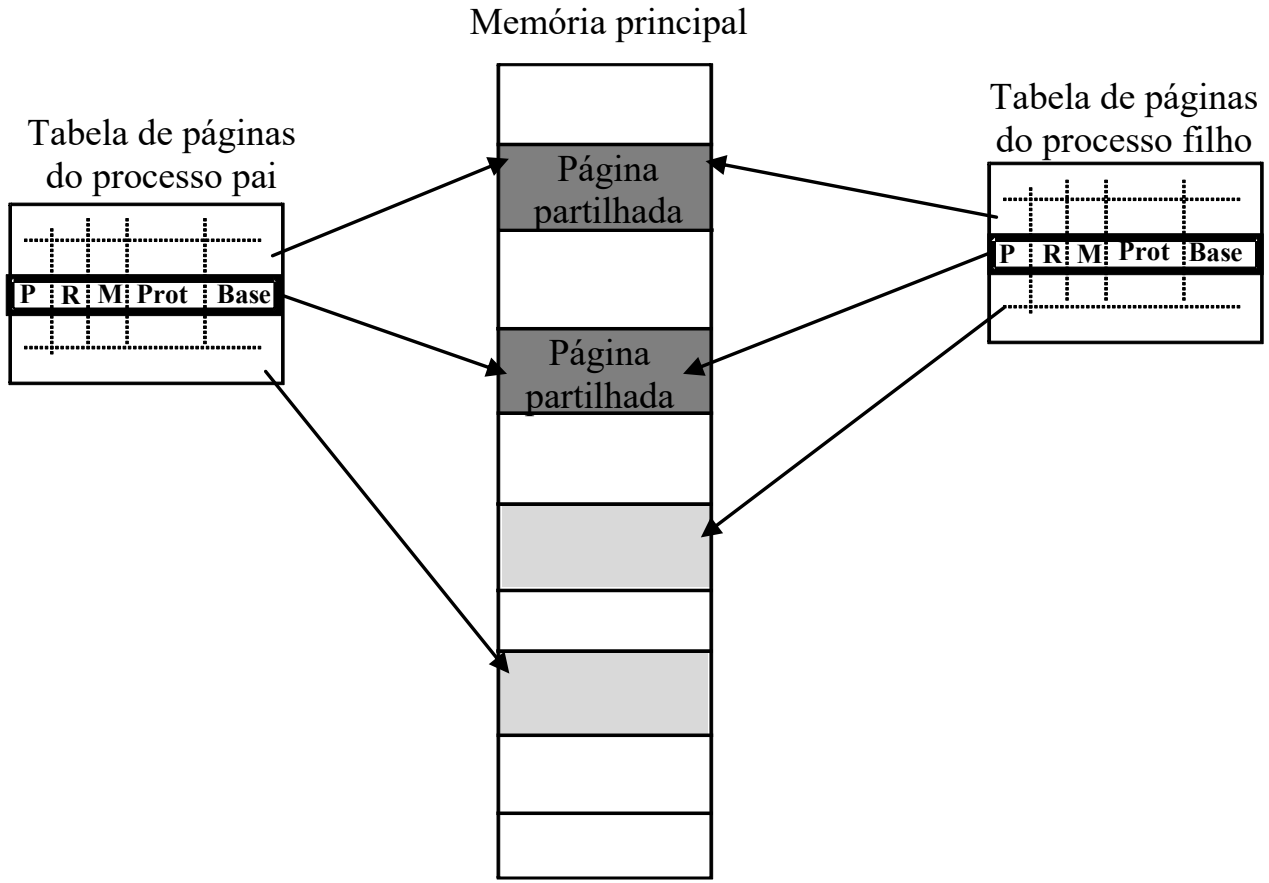


Aproveitar a memória virtual para outros fins desejáveis

Partilhar memória entre processos

- Basta ter, nas tabelas de páginas dos processos em causa, PTEs com a mesma base
- Os segmentos/páginas (virtuais) partilhados não precisam ser mapeados nos mesmos endereços virtuais em ambos os processos
  - Consequências?

Otimizar a cópia de memória na criação de processo



Criação de um Processo

- fork: duplica os segmentos de código, dados e pilha do pai
- Mas não é feita nenhuma cópia física de memória no momento do fork!
- As páginas só são copiadas se e quando tal for necessário
  - *Copy on write*

Copy on write (I)

- Quando ocorre o fork:
  - Aloca nova tabela de páginas para o processo filho e copia o conteúdo da tabela do pai
  - Nas PTEs com permissão de escrita (dados e pilha), retira permissão de escrita e ativa bit CoW
    - Na tabela do pai e do filho

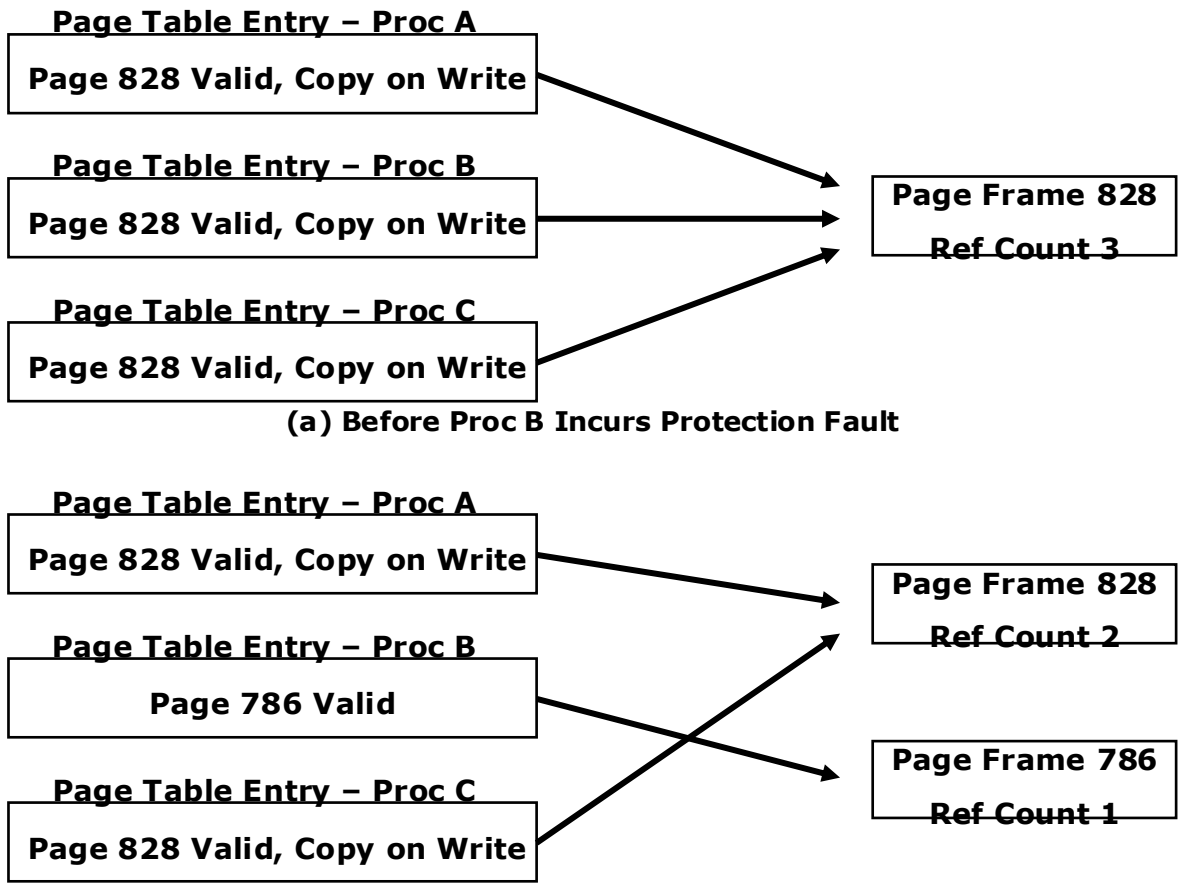
Isto chega para suportar acessos às páginas imutáveis (como as páginas de código)…

…E então as páginas modificáveis?

Copy on write (II)

- Quando pai ou filho tentam escrever numa página partilhada por CoW, ocorre excepção
  - Pois não há permissão de escrita na PTE
- Núcleo acorda e:
  - Aloca nova página e copia para lá o conteúdo da página partilhada
  - Atualiza a PTE do processo onde ocorreu a excepção com:
    - A base (endereço físico) da nova página
    - Permissão de escrita ativada, CoW desativado
  - Caso a página original já só seja referenciada por um processo, atualiza a sua PTE também:
    - Permissão de escrita ativada, CoW desativado

Tratamento do Copy on Write



Conclusão

- O endereçamento virtual paginado é uma forma de organizar a gestão de memória que permite atingir os objetivos
- A solução tem, contudo, muitos problemas quando se pretende conciliar aspetos antagónicos como enormes espaços de endereçamento virtuais com um desempenho otimizado do hardware
- O tema tem sido explorado pela arquitetura dos processadores para tentarem resolver os problemas por hardware – *caches*, TLB, *huge pages*, MMU que gerem hierarquias de paginas e tabelas invertidas de páginas
- A gestão de memória do sistema operativo tem de tirar partido destes mecanismos, mas ter algoritmos eficientes para gerir a memória física, a memória de massa e a dinâmica dos processos