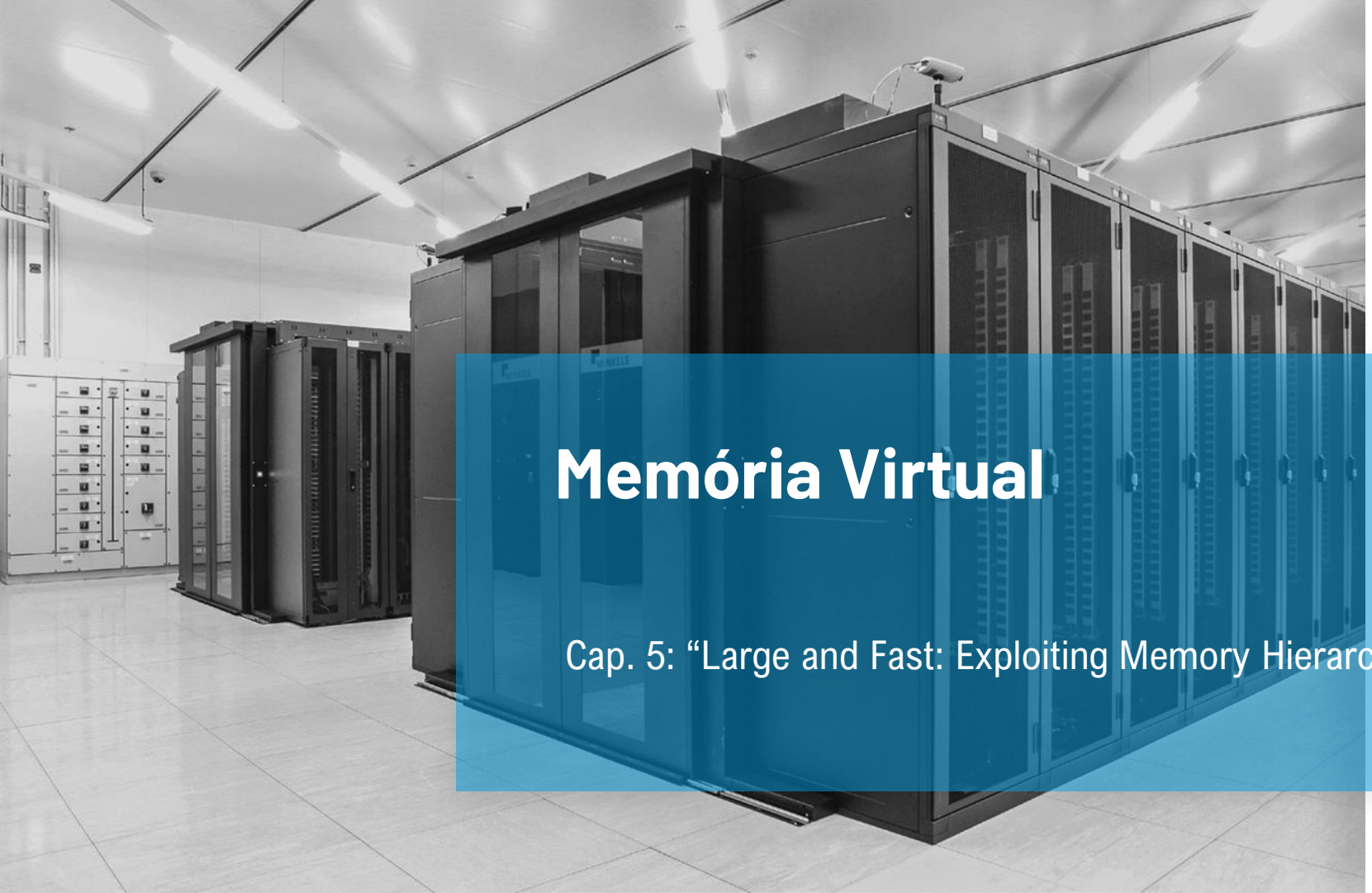


Memória Virtual

Cap. 5: “Large and Fast: Exploiting Memory Hierarchy”

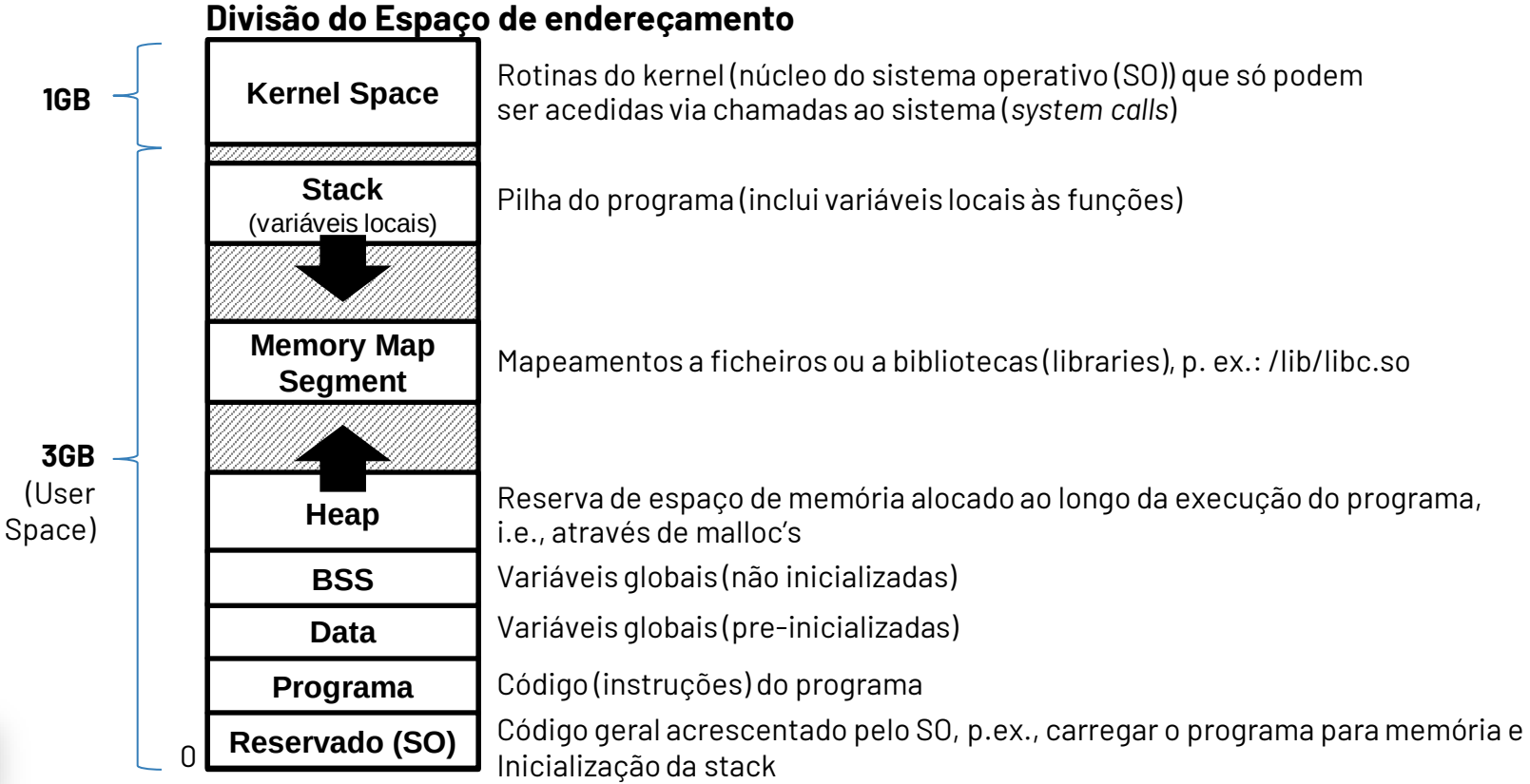
14 June, 2022

Mapa de memória

Revisão do compêndio de slides 3: RISC-V ISA

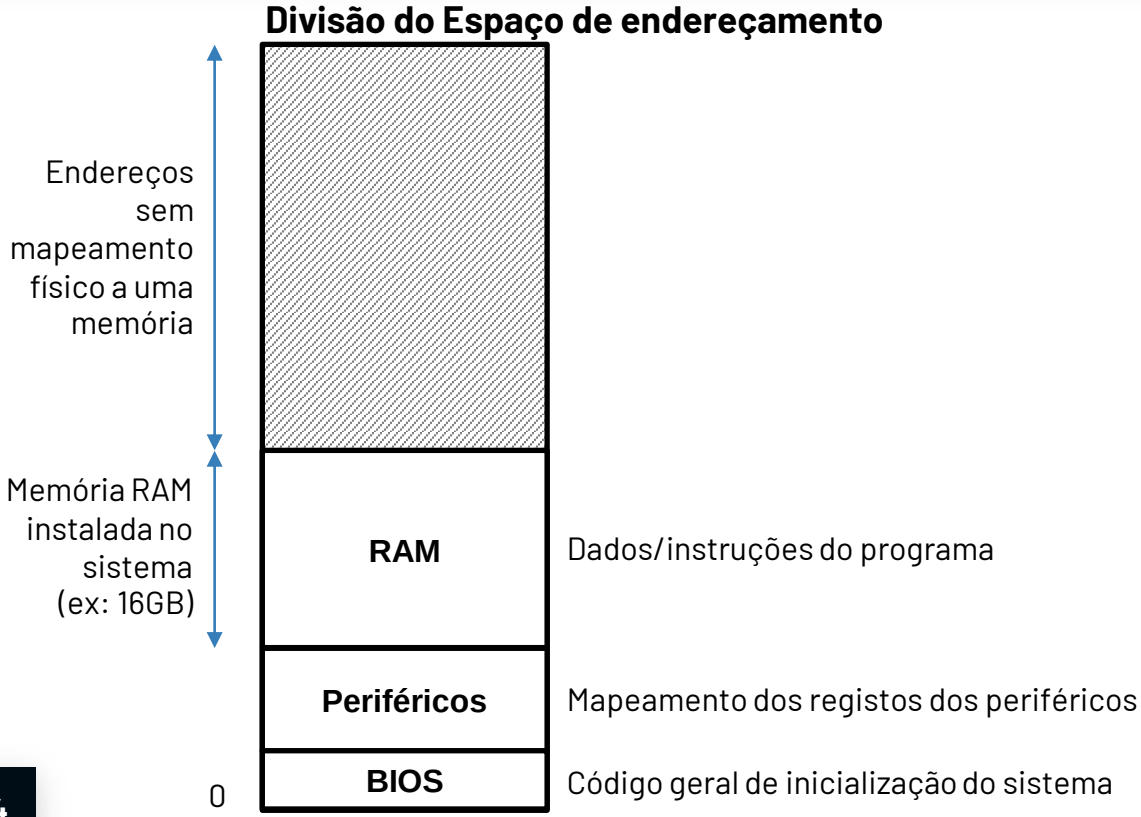
Mapa de memória

Visão da aplicação



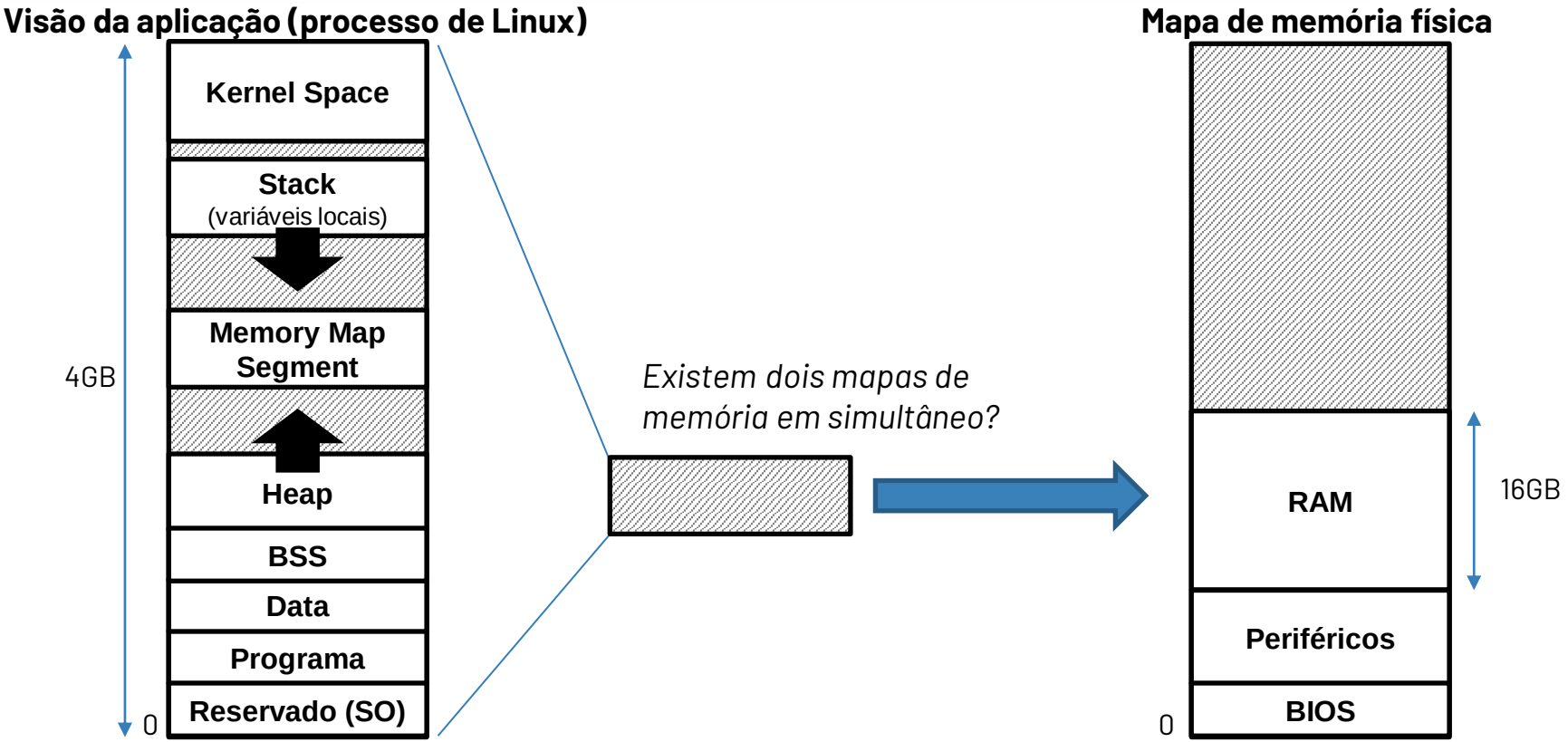
Mapa de memória

Visão do Sistema Operativo (SO) / Bare Metal



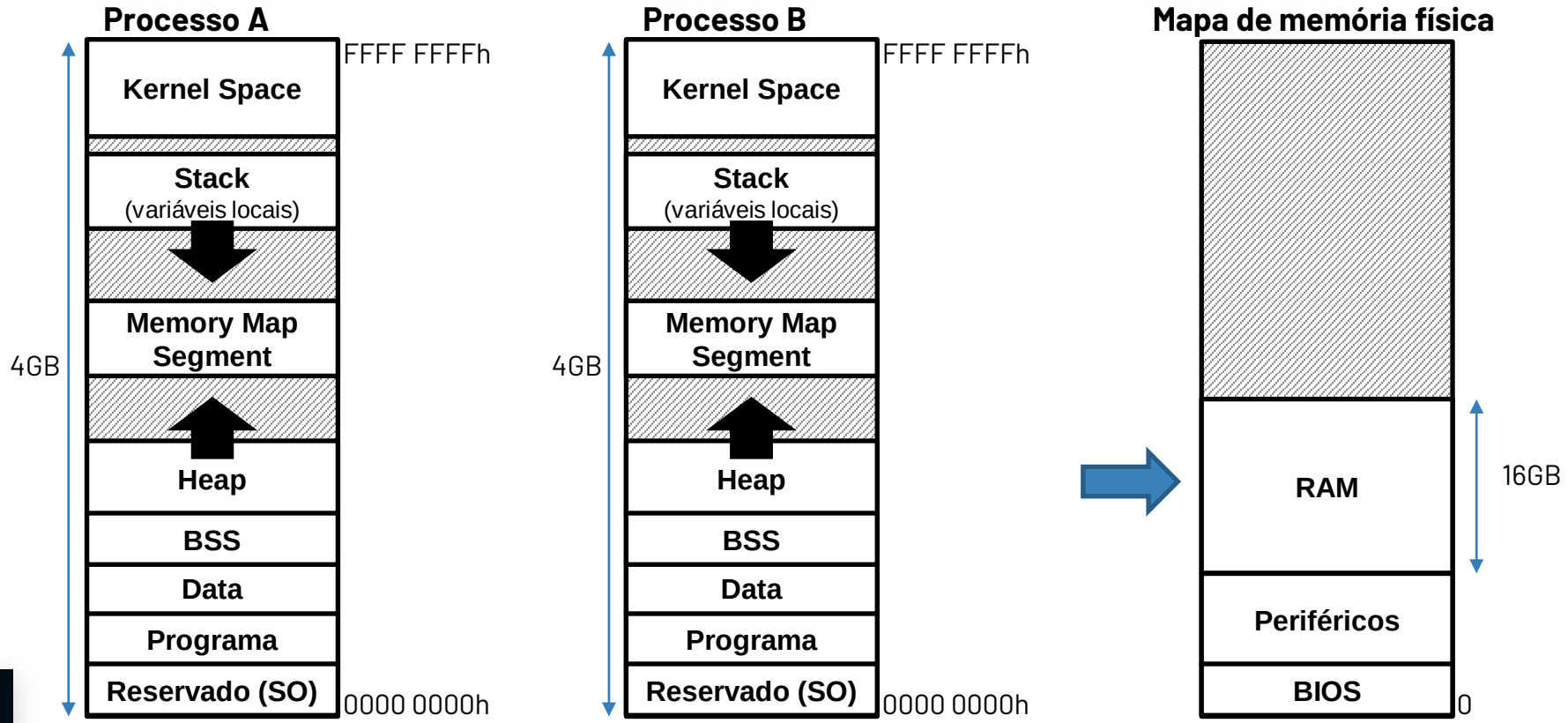
Mapa de memória

Dois mapas de memória diferentes?



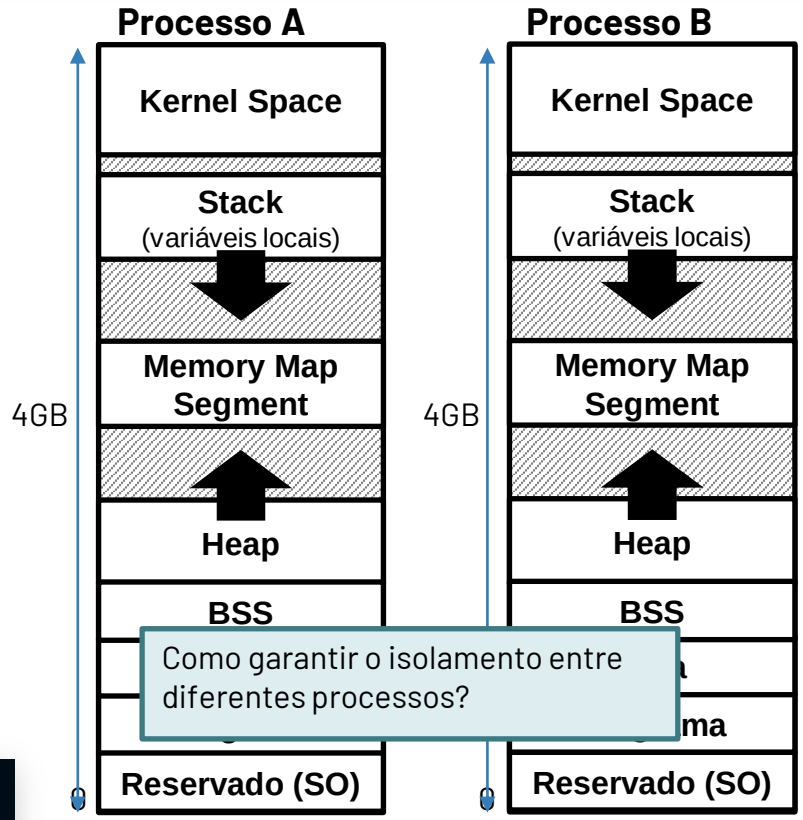
Mapa de memória

Muitos mapas de memória diferentes?



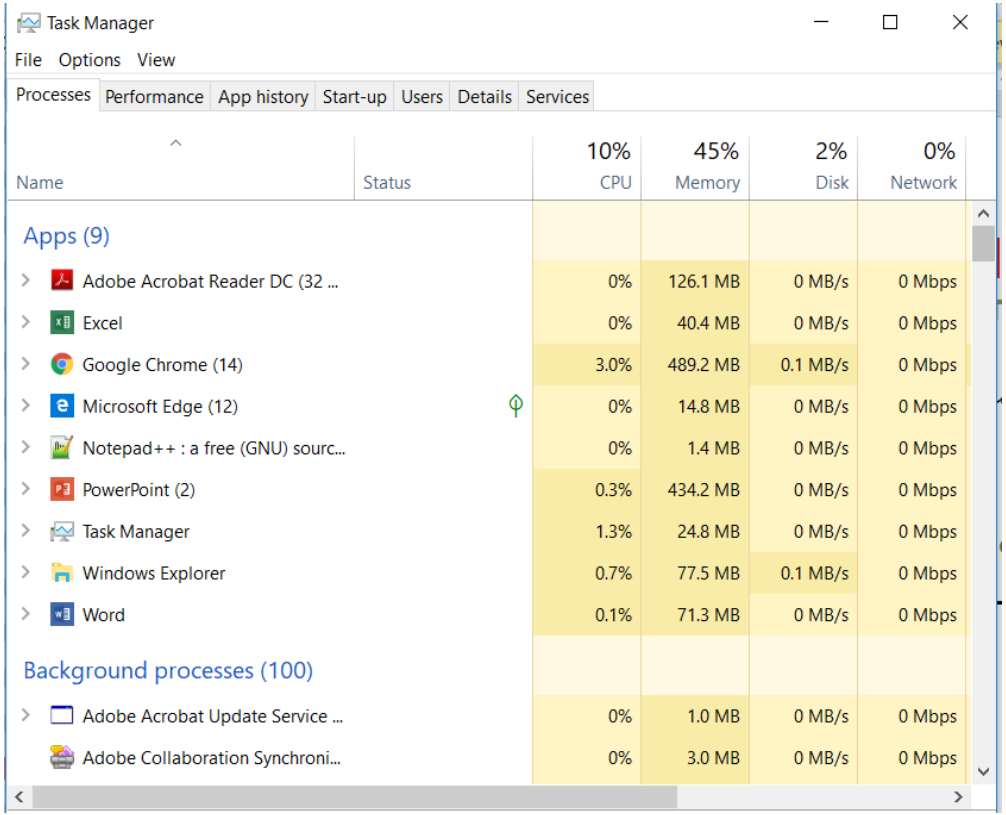
Mapa de memória

Muitos mapas de memória diferentes?



Como garantir o isolamento entre diferentes processos?

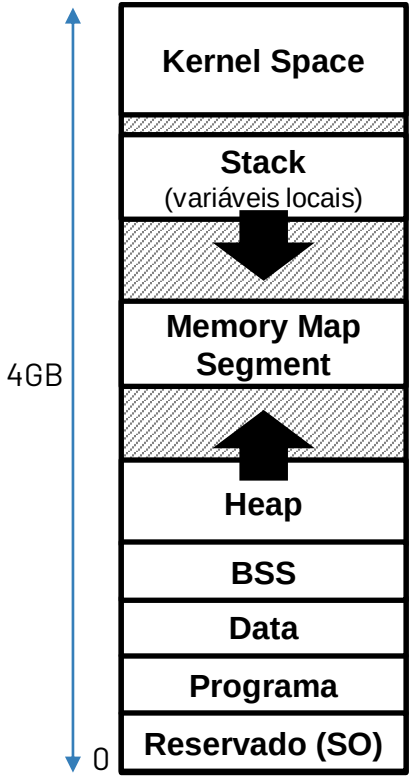
Todos os processos ocupam 4GB?



Mapa de memória

Dois mapas de memória diferentes?

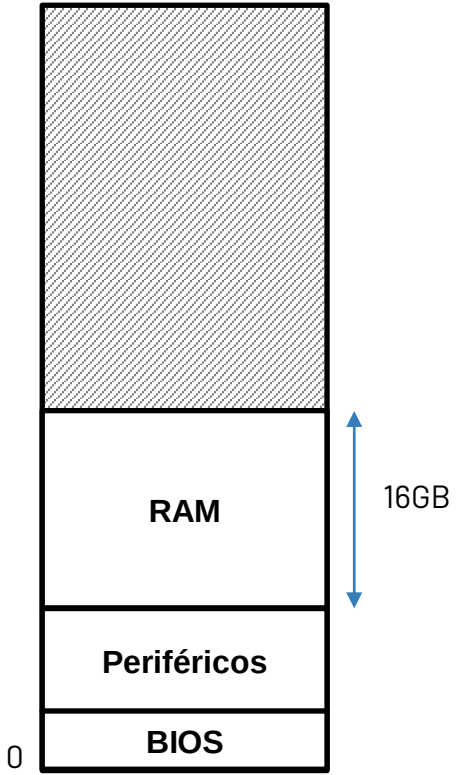
Visão da aplicação (processo de Linux) – Memória Virtual



*Cada aplicação tem uma visão "própria" do espaço de endereçamento – **Memória Virtual***

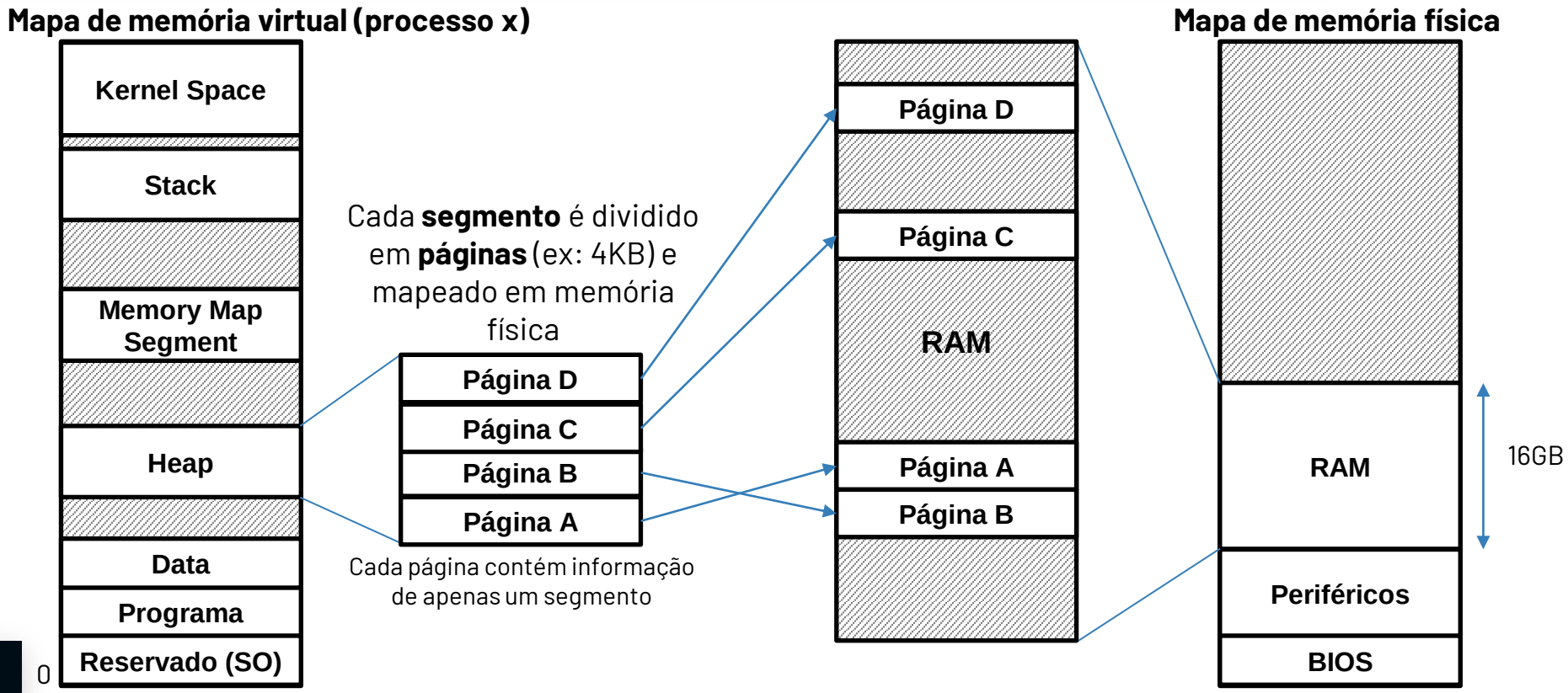
*O espaço de endereçamento **Virtual**, visto por cada aplicação, tem de ser mapeado no espaço de endereçamento **Físico***

Mapa de Memória Física



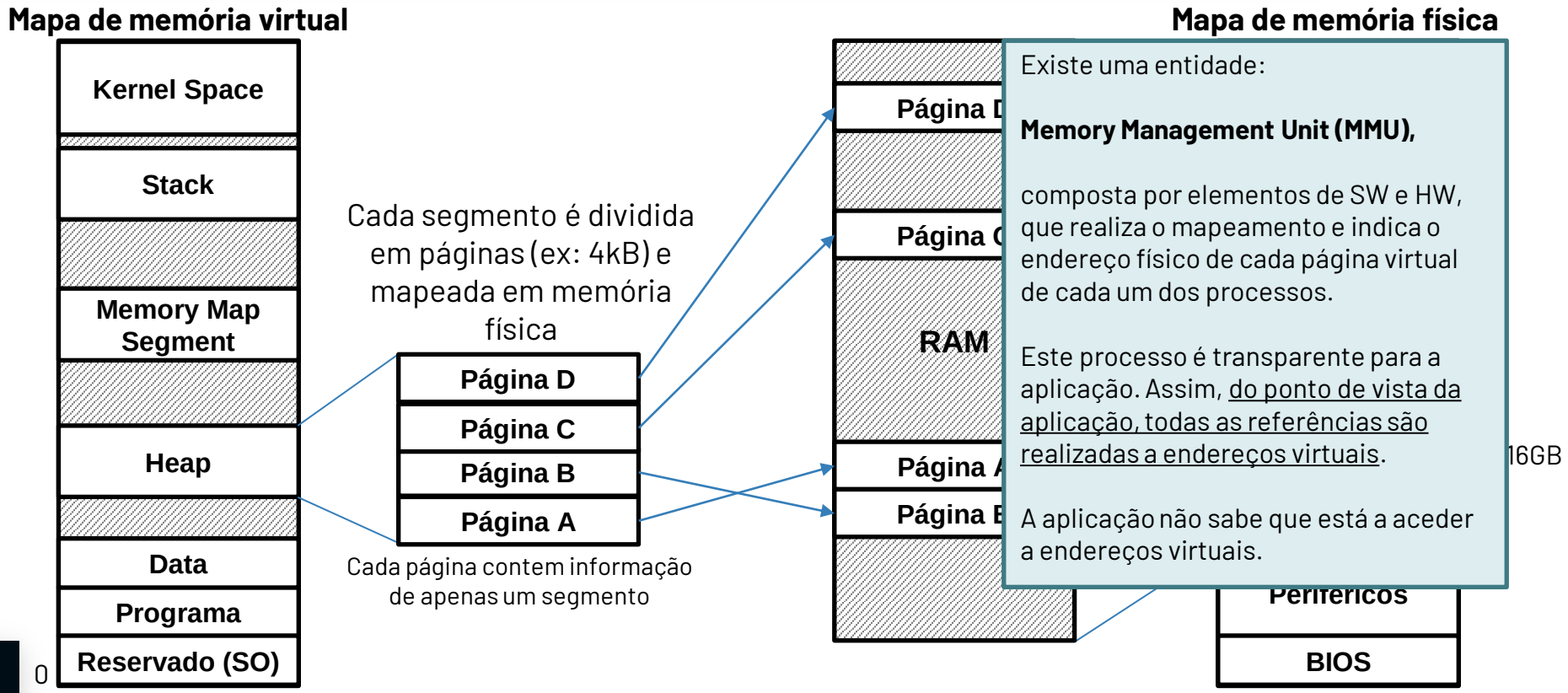
Mapa de memória

Divisão do mapa em páginas



Mapa de memória

Divisão do mapa em páginas



Existe uma entidade:

Memory Management Unit (MMU),

composta por elementos de SW e HW, que realiza o mapeamento e indica o endereço físico de cada página virtual de cada um dos processos.

Este processo é transparente para a aplicação. Assim, do ponto de vista da aplicação, todas as referências são realizadas a endereços virtuais.

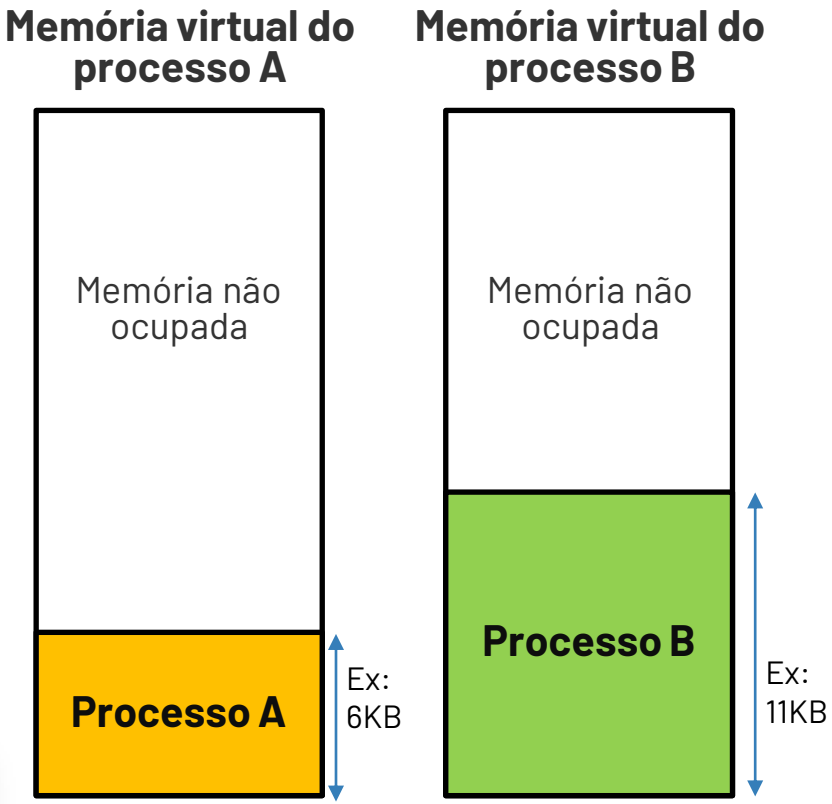
A aplicação não sabe que está a aceder a endereços virtuais.

Memória Virtual

Princípio de tradução (ideia chave)

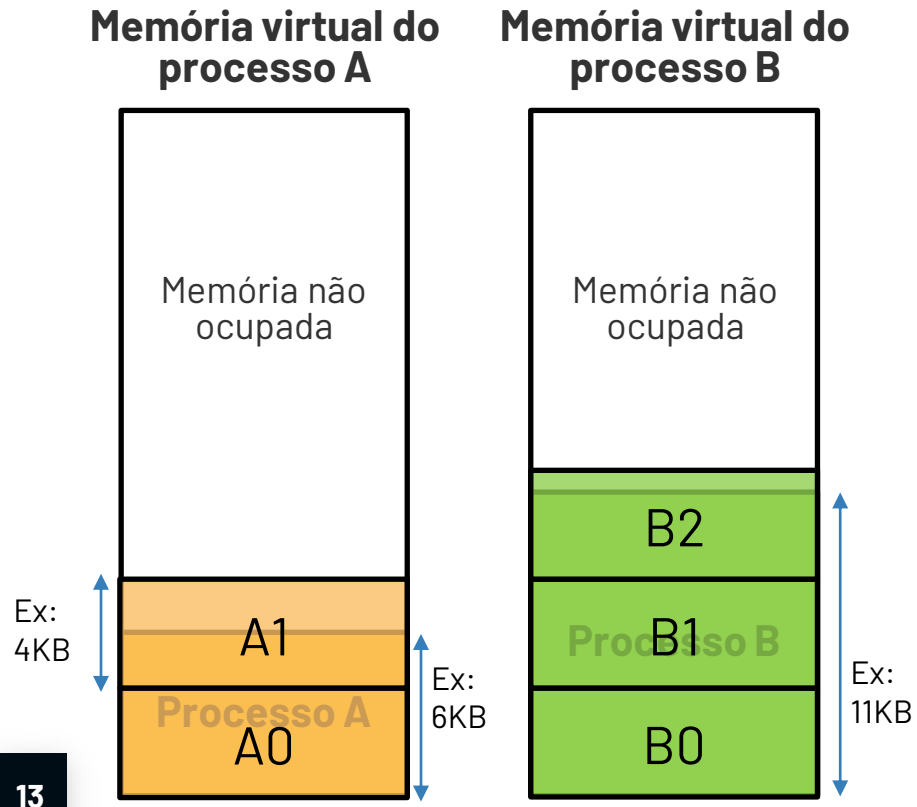
Paginação do mapa de memória

Exemplo considerando apenas o programa (segmento de instruções)



Paginação do mapa de memória

Paginação do segmento de instruções

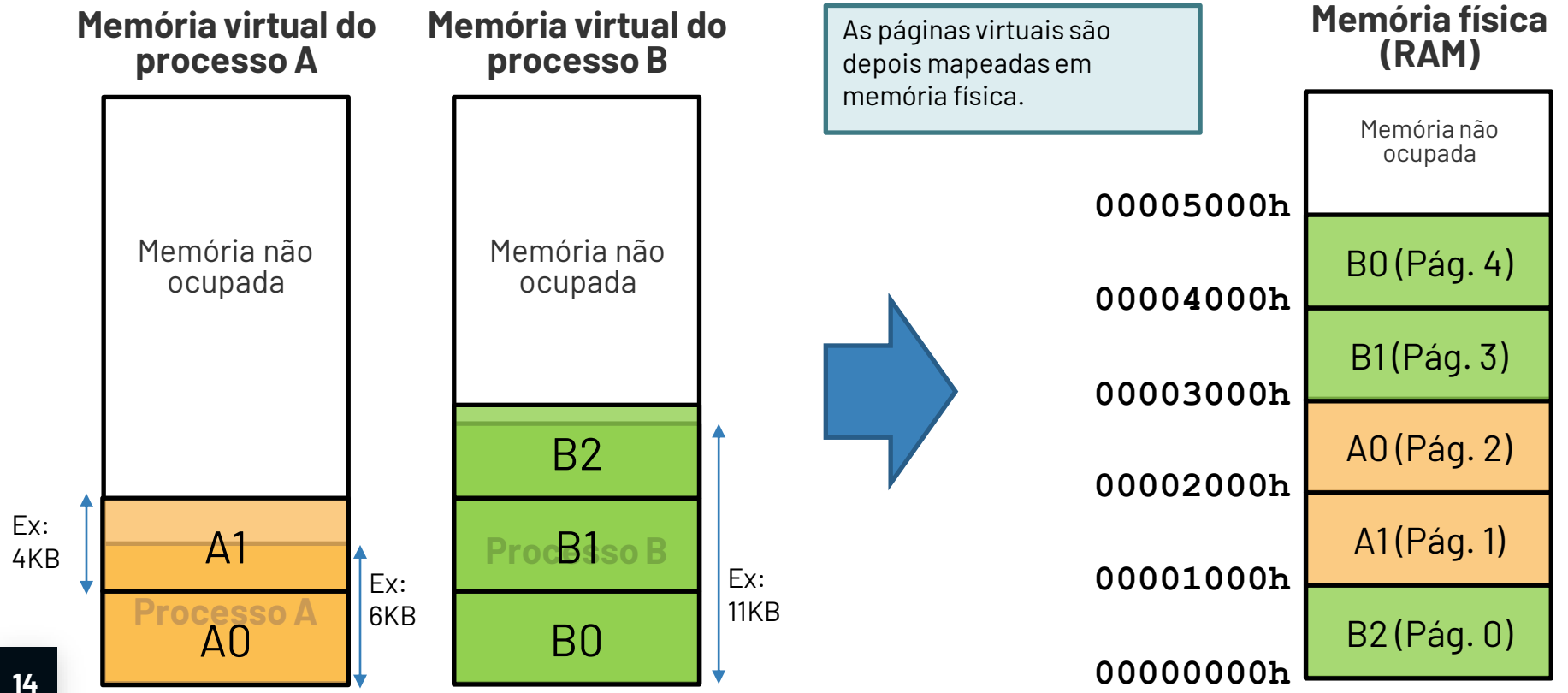


Divide-se cada segmento de cada um dos processos (ex: program, data, BSS, stack) em páginas.

Cada página contém apenas um segmento. Assim, é possível existirem "buracos" (endereços livres) entre segmentos que advêm da dimensão do bloco e do segmento.

Paginação do mapa de memória

Mapeamento das páginas virtuais na memória física



Paginação do mapa de memória

Tabela de tradução entre endereços físicos e virtuais

TÉCNICO LISBOA

Cada processo tem um tabela de páginas para realizar a tradução entre memória virtual e memória física

Memória virtual do processo B

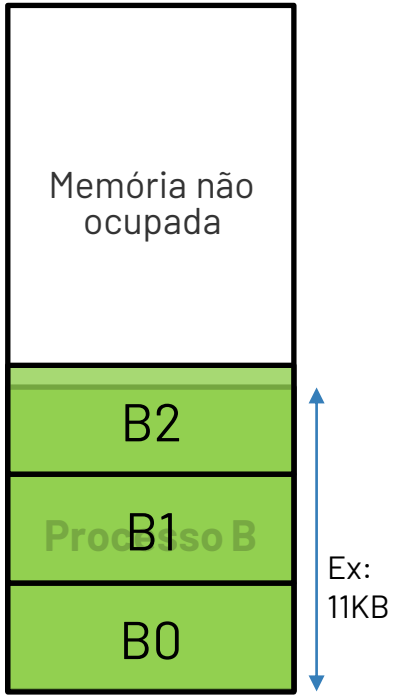
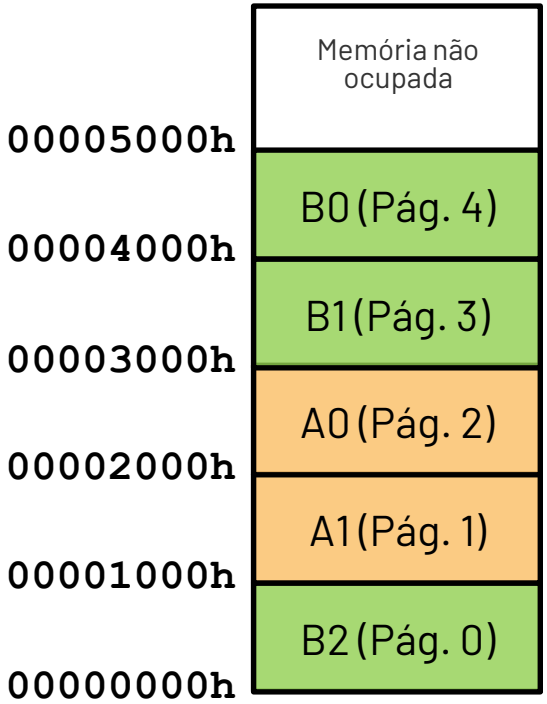


Tabela de páginas (page table)

	. . .
Pág. 2	00000000h
Pág. 1	00003000h
Pág. 0	00004000h

Memória física (RAM)

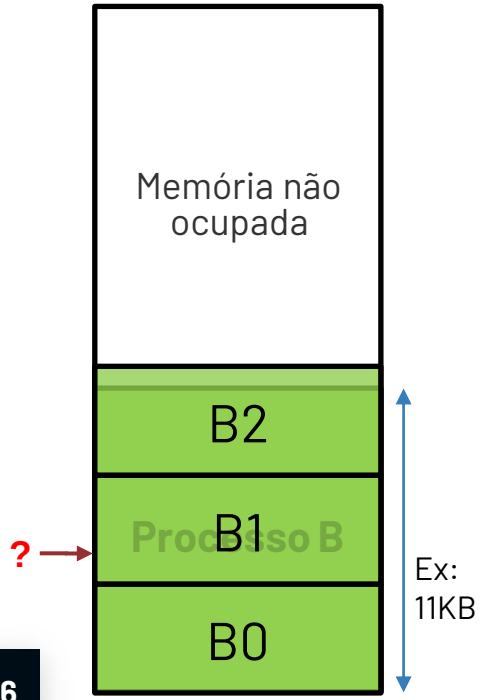


Paginação do mapa de memória

Decomposição do endereço virtual

TÉCNICO LISBOA

Memória virtual do processo B



Endereço virtual

Virtual page number	offset

Todas as referências do programa são feitas a endereços virtuais, os quais podem ser decompostos em:

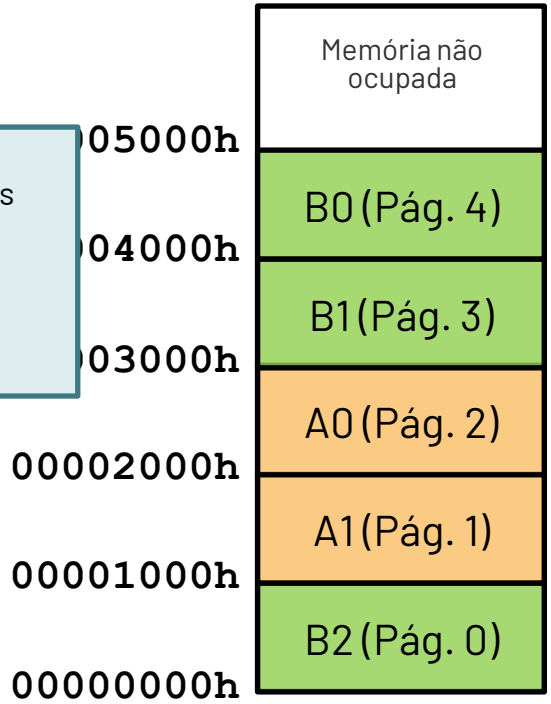
- **Número da página virtual**

+

- **Offset dentro da página**

	. . .
Pág. 2	00000000h
Pág. 1	00003000h
Pág. 0	00004000h

Memória física (RAM)



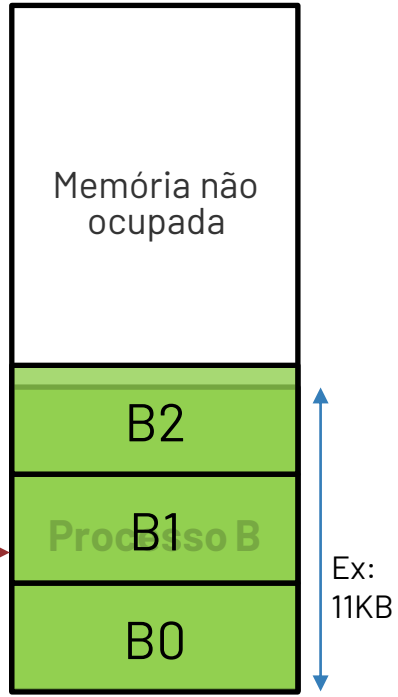
Paginação do mapa de memória

Decomposição do endereço virtual

RV64 ➔ Registos inteiros de 64 bits, endereços de 64 bits
RV32 ➔ Registos inteiros de 32 bits, endereços de 32 bits

Vamos assumir RV64...

Memória virtual do processo B



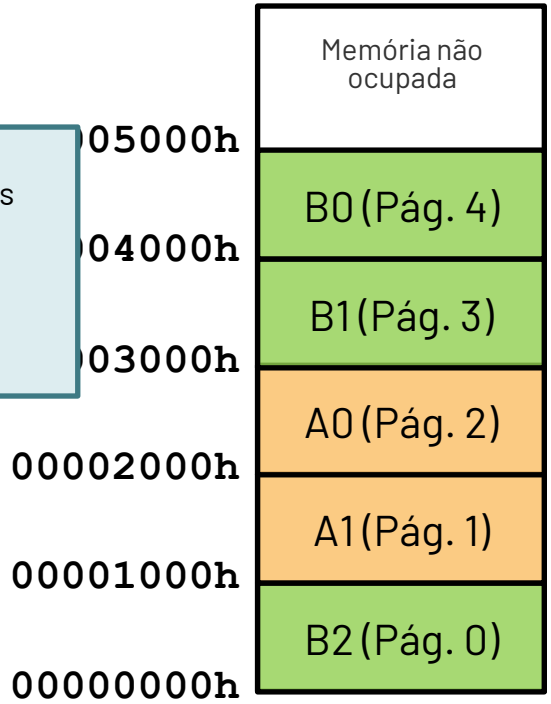
Endereço virtual (64 bits)

Virtual page number	offset

Todas as referências do programa são feitas a endereços virtuais, os quais podem ser decompostos em:
- **Número da página virtual**
+
- **Offset dentro da página**

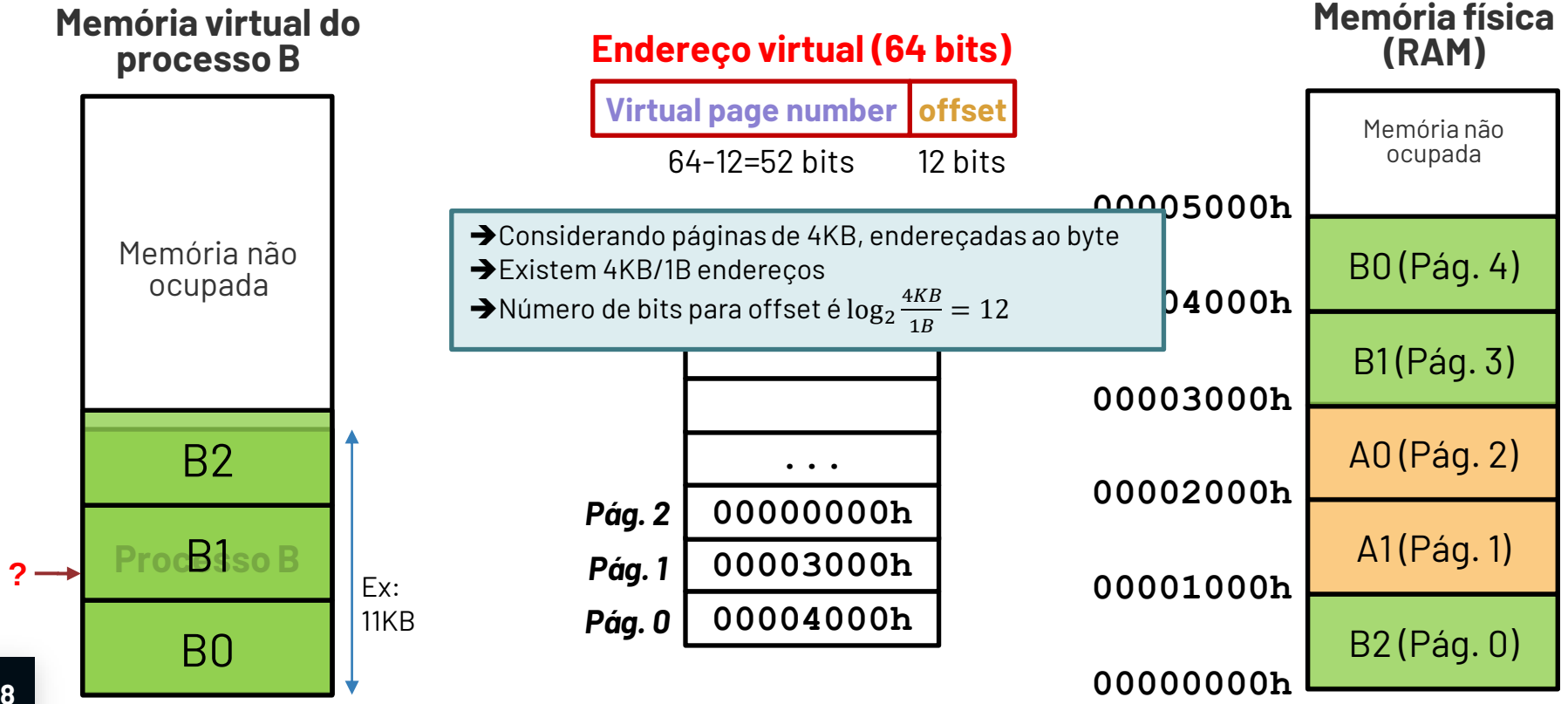
	. . .
Pág. 2	00000000h
Pág. 1	00003000h
Pág. 0	00004000h

Memória física (RAM)



Paginação do mapa de memória

Cálculo do número de bits para cada campo do endereço virtual

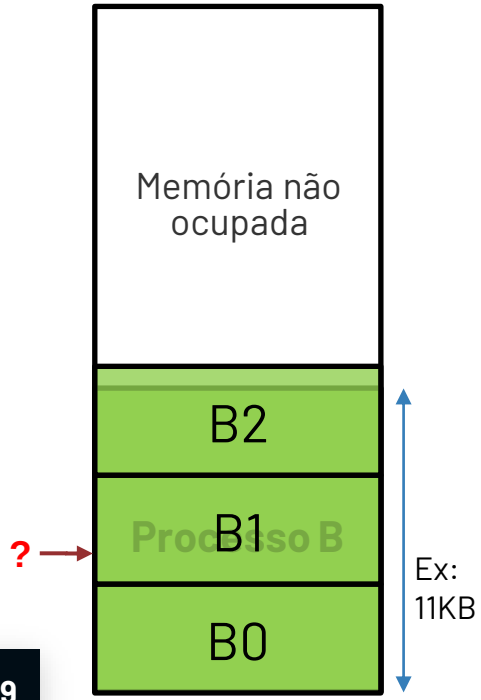


Paginação do mapa de memória

Exemplo de tradução

TÉCNICO LISBOA

Memória virtual do processo B



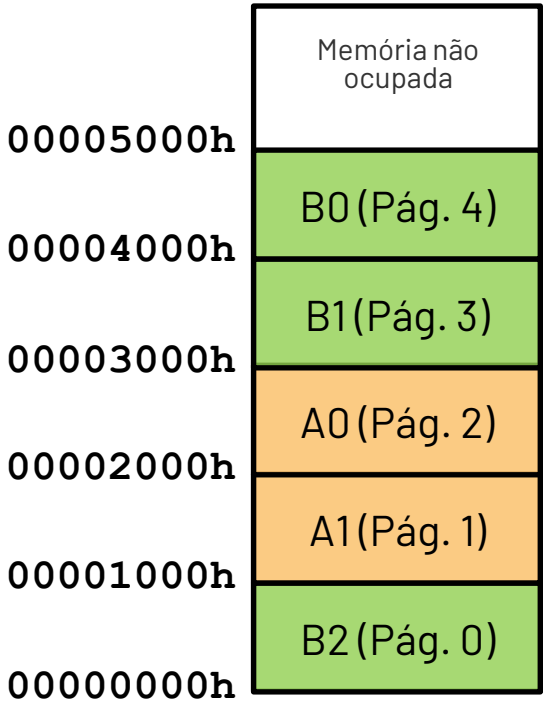
Endereço virtual (64 bits)

0...0 0000 0000 0001	024h
64-12=52 bits	12 bits

Tabela de páginas (page table)

	. . .
Pág. 2	00000000h
Pág. 1	00003000h
Pág. 0	00004000h

Memória física (RAM)



Paginação do mapa de memória

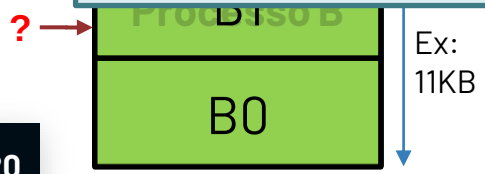
Exemplo de tradução

TÉCNICO LISBOA

Memória virtual do processo B

O campo "***Virtual Page Number***" indica a entrada da tabela de páginas que contém o endereço base (na memória física) da página pretendida.

Ao endereço base é necessário somar o deslocamento (**offset**)



Endereço virtual (64 bits)

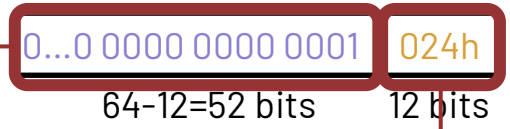
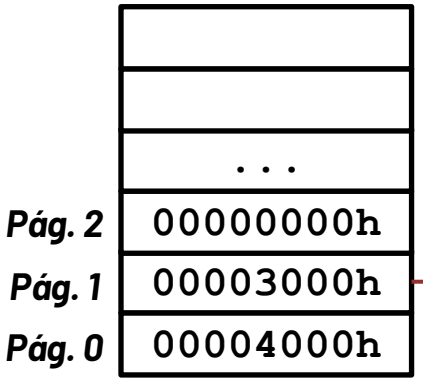
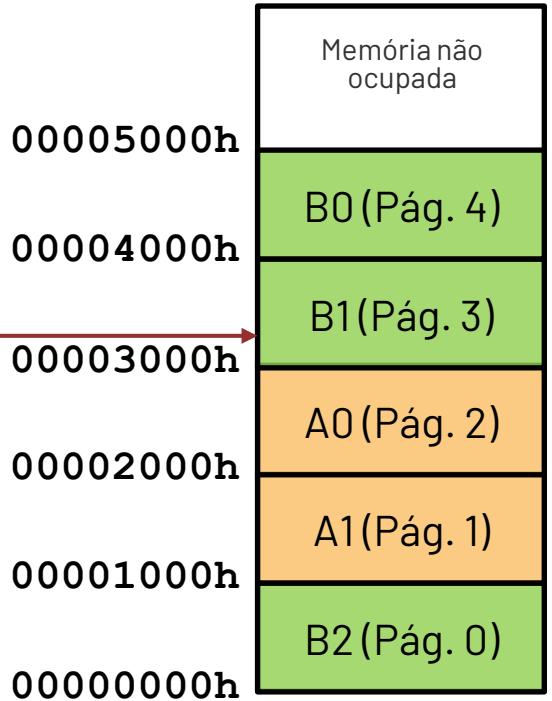


Tabela de páginas (page table)

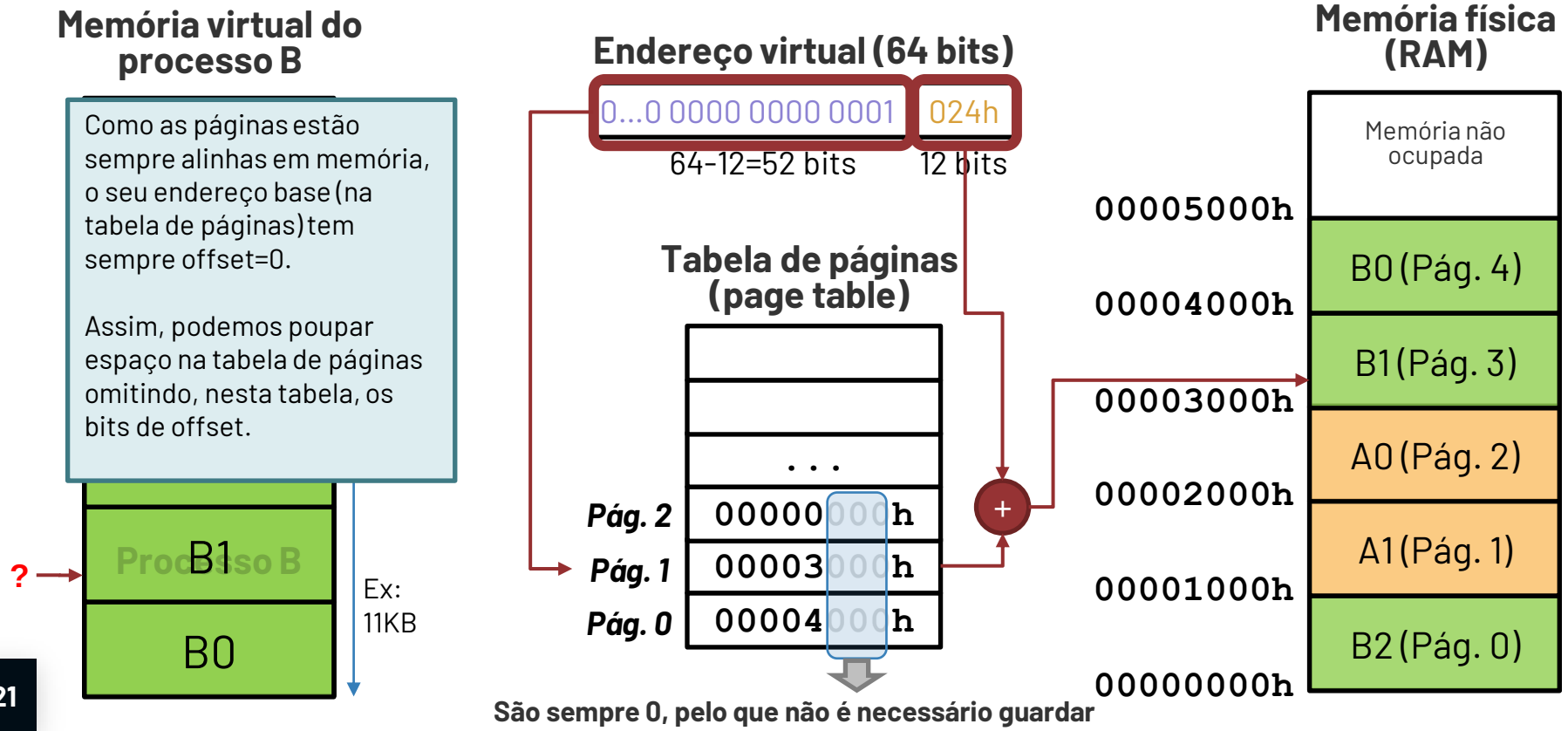


Memória física (RAM)



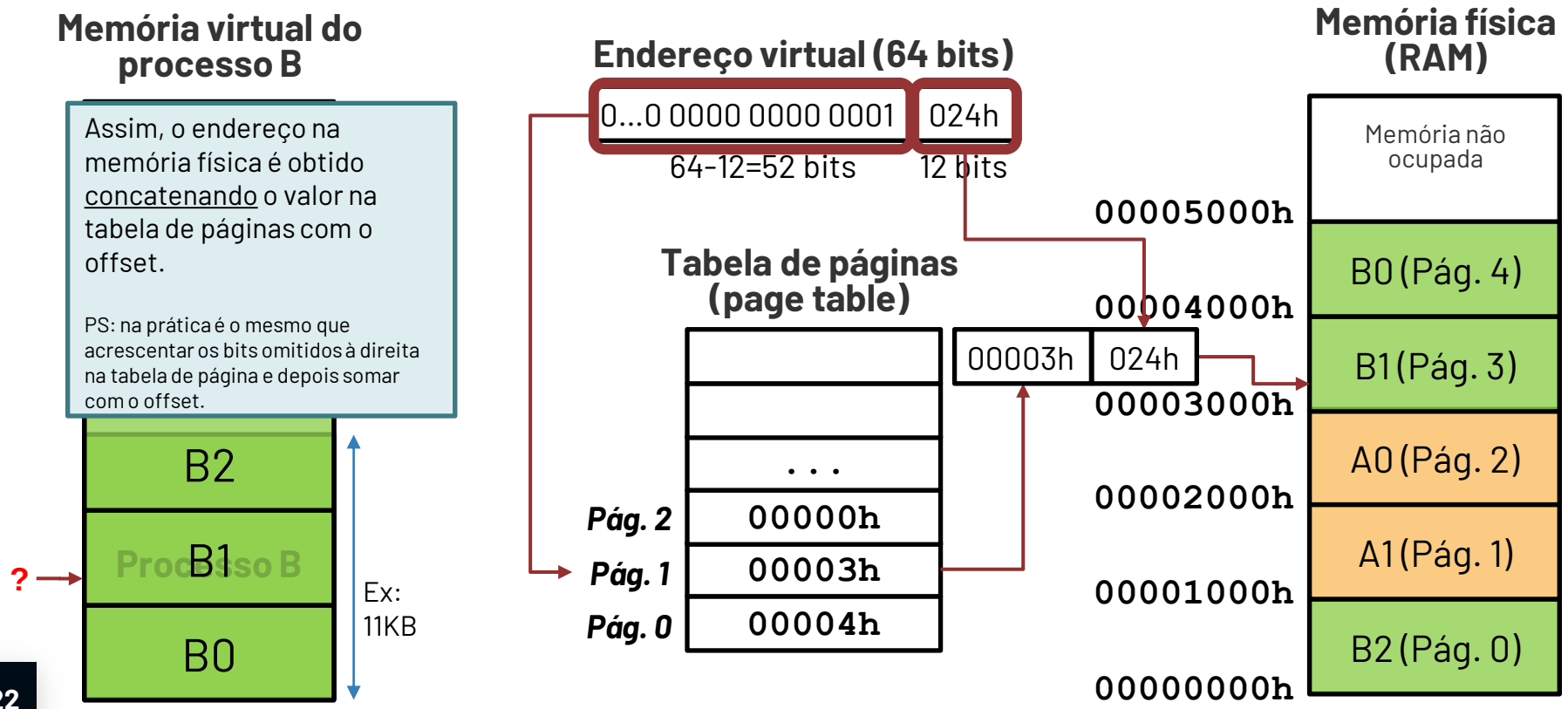
Paginação do mapa de memória

Simplificação dos campos (*page table entries - PTE*) da tabela de páginas



Paginação do mapa de memória

Tradução de endereço virtual em físico após simplificação das PTEs



Paginação do mapa de memória

Problemas

Onde guardar a tabela de páginas?
Na memória!

Qual a sua dimensão?
Se a memória virtual tiver endereços de 64 bits (RISC-V), e cada página tiver 4kB (=2^{12}), vão existir 2^{52} páginas virtuais.

Cada entrada na tabela de páginas tem de conter (pelo menos) o número de bits em falta para endereçar a memória física.

Se a memória física tiver endereços de 64 bits (RISC-V), e sabendo que os 12 bits menos significativos correspondem ao offset, restam 52 bits.

Arredondando para 64 bits (8B), significa que a tabela de páginas ocupa 2^{52}x8B = 32 PB?!!!!

Endereço virtual (64 bits)

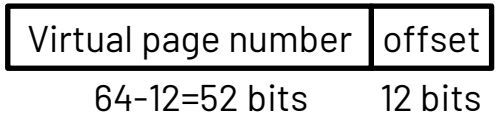
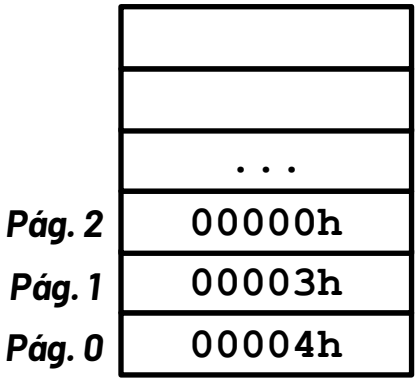
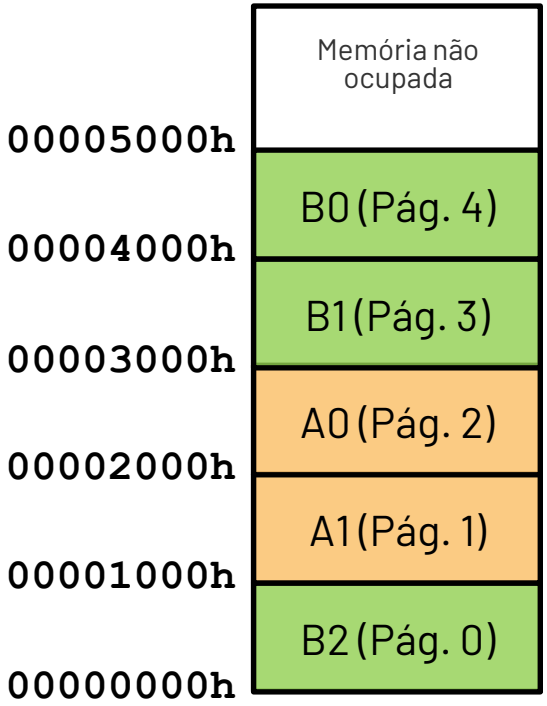


Tabela de páginas (page table)



Memória física (RAM)



Paginação do mapa de memória

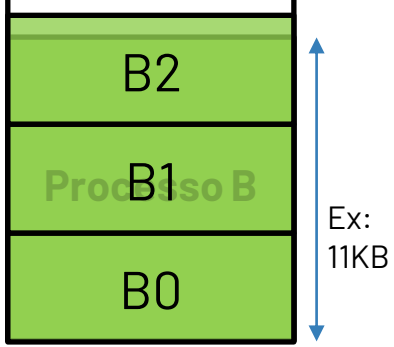
Solução

TÉCNICO LISBOA

Memória virtual do processo B

Solução:

Paginar a tabela de páginas....



Endereço virtual (64 bits)

Virtual page number	offset
64-12=52 bits	12 bits

Tabela de páginas (page table)

	. . .
Pág. 2	**00000h**
Pág. 1	**00003h**
Pág. 0	**00004h**

Memória física (RAM)

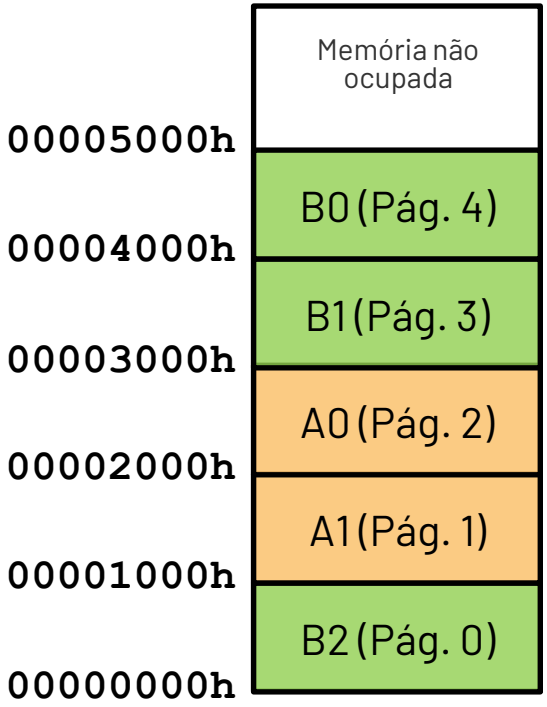
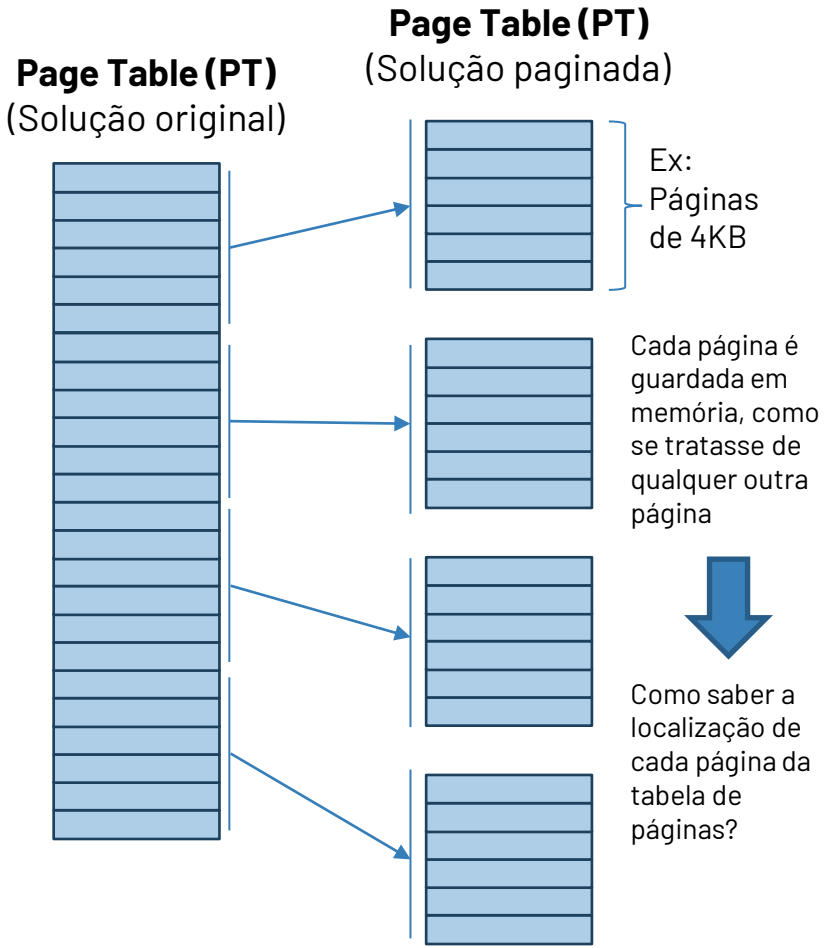


Tabela de páginas hierarquica



Memória RAM

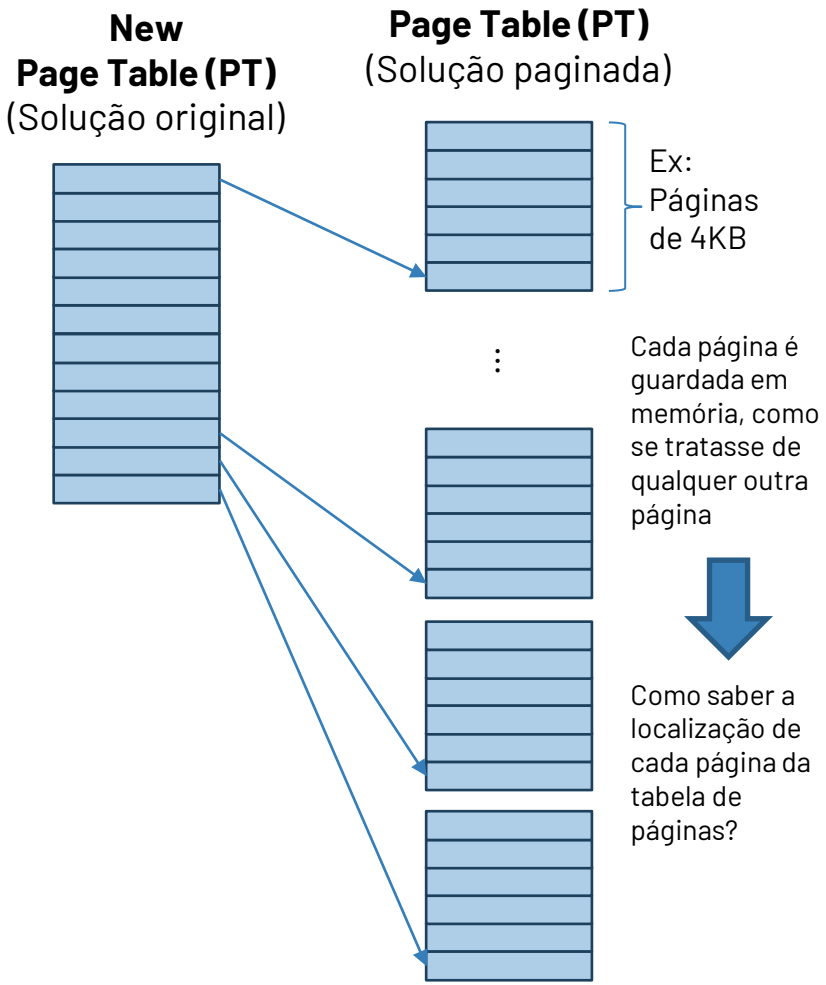
Memória não ocupada
PT1 – Page 0 (process A)
PT0 (process B)
Virtual Page 2 (process B)
Virtual Page 12 (process A)
PT2 – Page 0 (process A)
PT2- Page 0 (process B)
PT0 (process A)
Virtual Page 5 (process B)
Virtual Page 0 (process B)

Tabela de páginas hierarquica

Solução:

Construir uma tabela de páginas que indica a localização da tabela de páginas

Cada entrada da tabela de páginas indica a localização física de uma página em memória (excluindo bits de offset, já que são zero)



Memória RAM

Memória não ocupada
PT1 – Page 0 (process A)
PT0 (process B)
Virtual Page 2 (process B)
Virtual Page 12 (process A)
PT2 – Page 0 (process A)
PT2- Page 0 (process B)
PT0 (process A)
Virtual Page 5 (process B)
Virtual Page 0 (process B)

TÉCNICO LISBOA

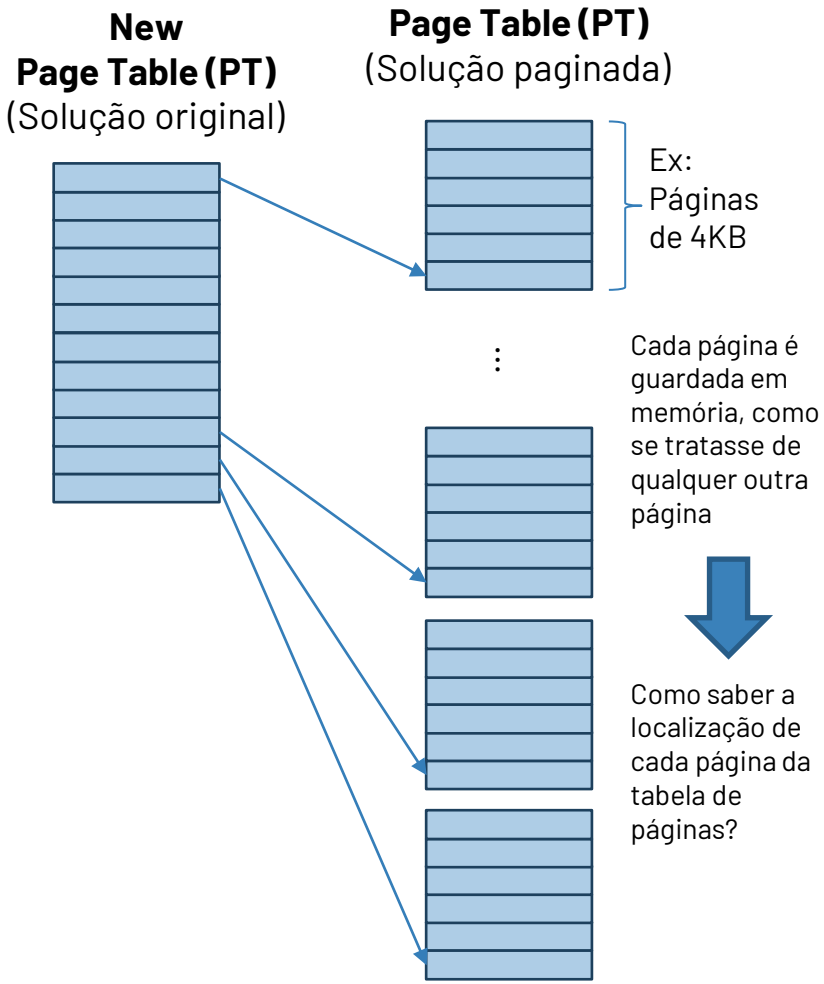
Tabela de páginas hierarquica

Problema:

E se a nova tabela de páginas ainda for muito grande?

Solução:

Repetir o processo até que a tabela de páginas de nível superior couber numa página (ex: 4KB)



Memória RAM

Memória não ocupada
PT1 – Page 0 (process A)
PT0 (process B)
Virtual Page 2 (process B)
Virtual Page 12 (process A)
PT2 – Page 0 (process A)
PT2- Page 0 (process B)
PT0 (process A)
Virtual Page 5 (process B)
Virtual Page 0 (process B)

TÉCNICO LISBOA

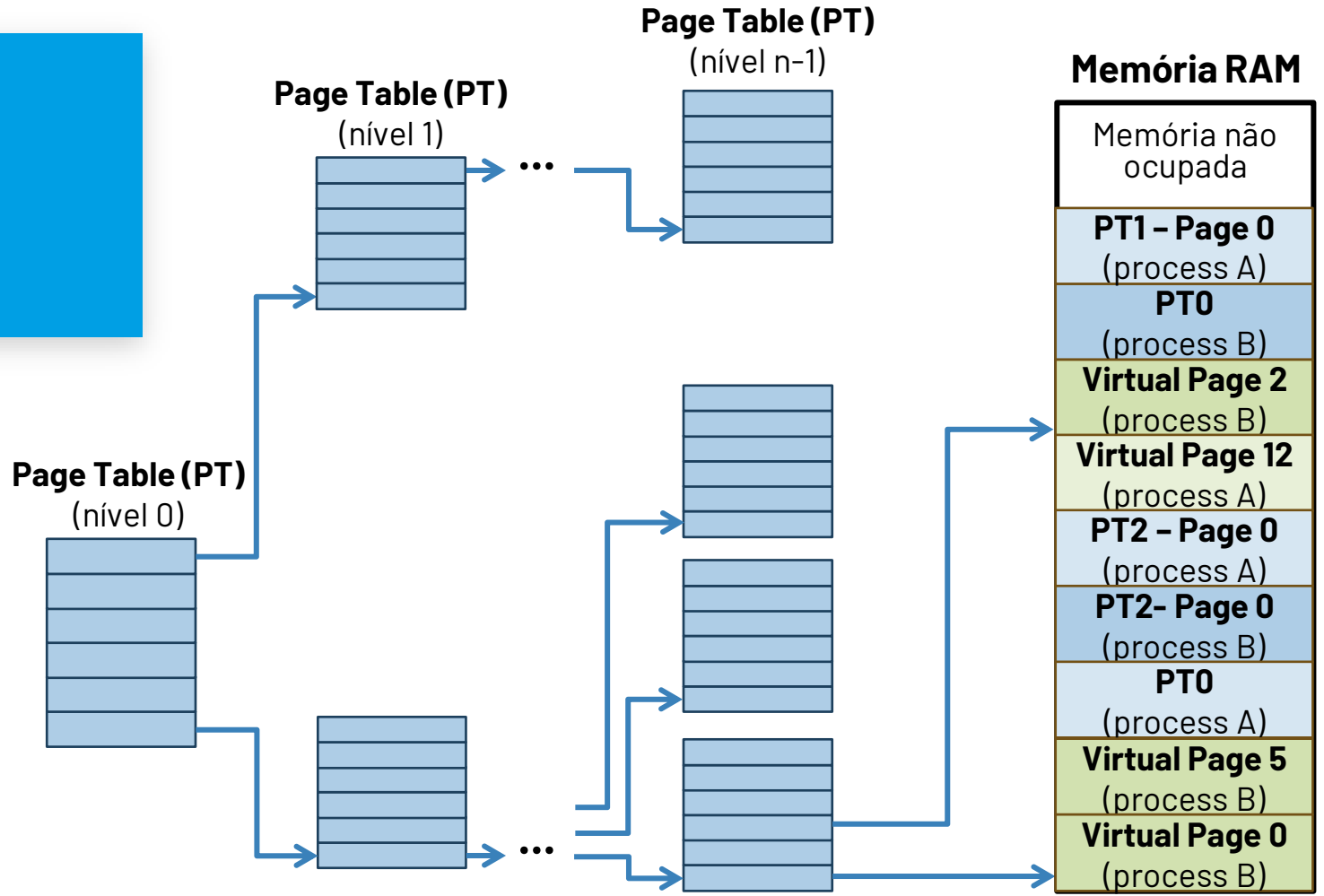
Tabela de páginas hierarquica

Problema:

Como saber a localização da tabela de páginas de nível superior?

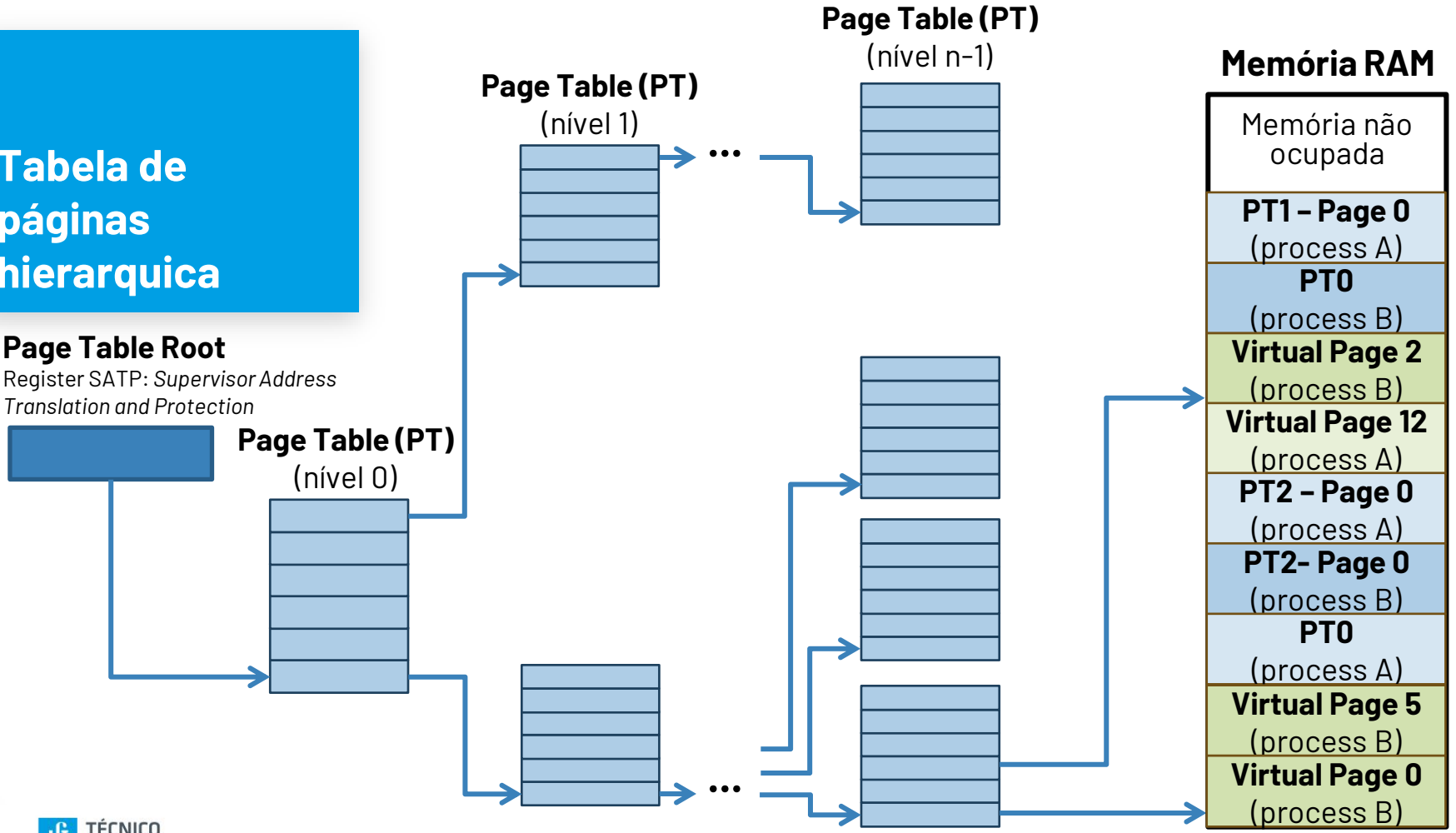
Solução:

Adicionar um registo especial no processador com a localização.



TÉCNICO LISBOA

Tabela de páginas hierarquica



TÉCNICO LISBOA

Tabela de páginas hierarquica

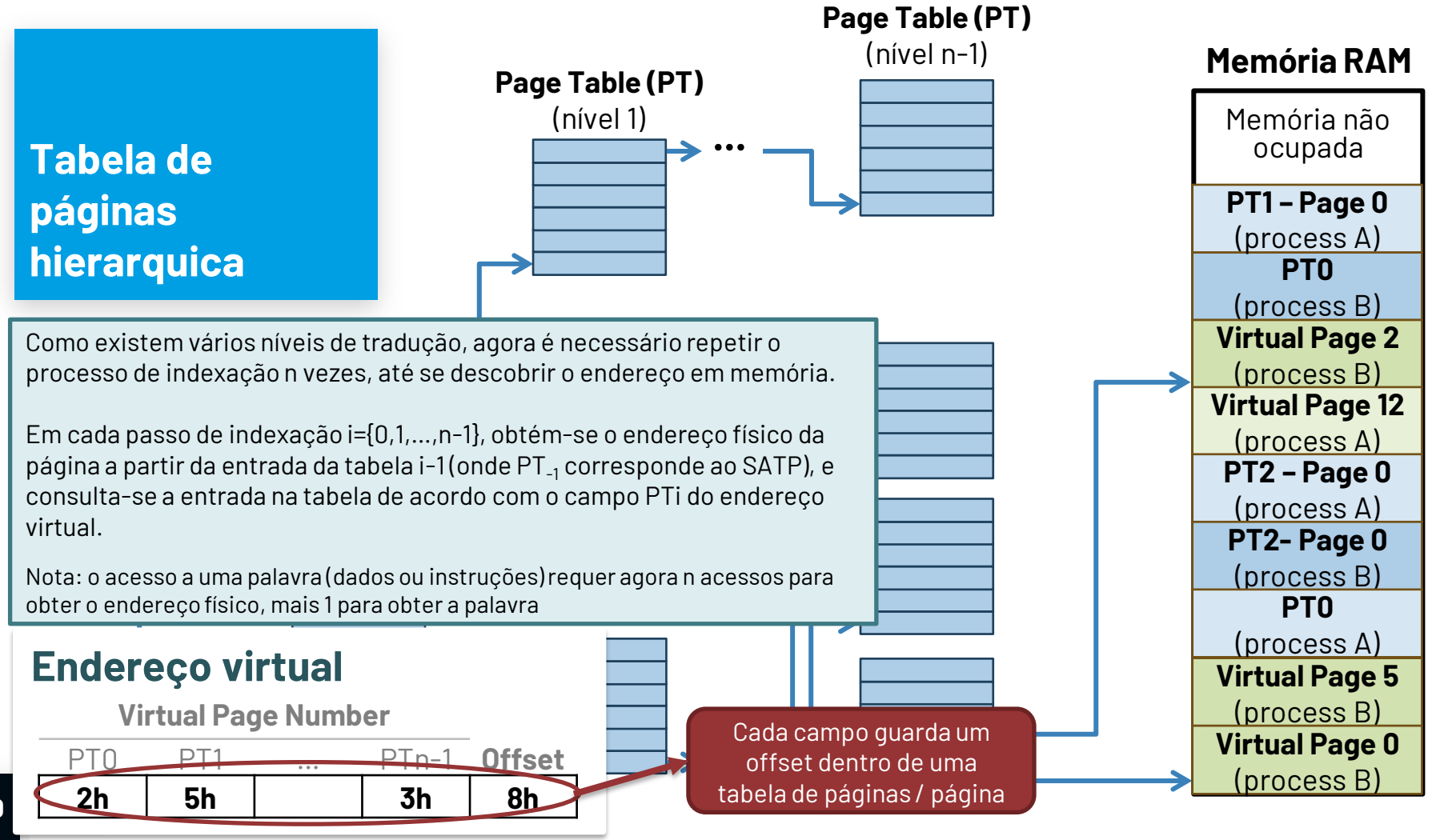
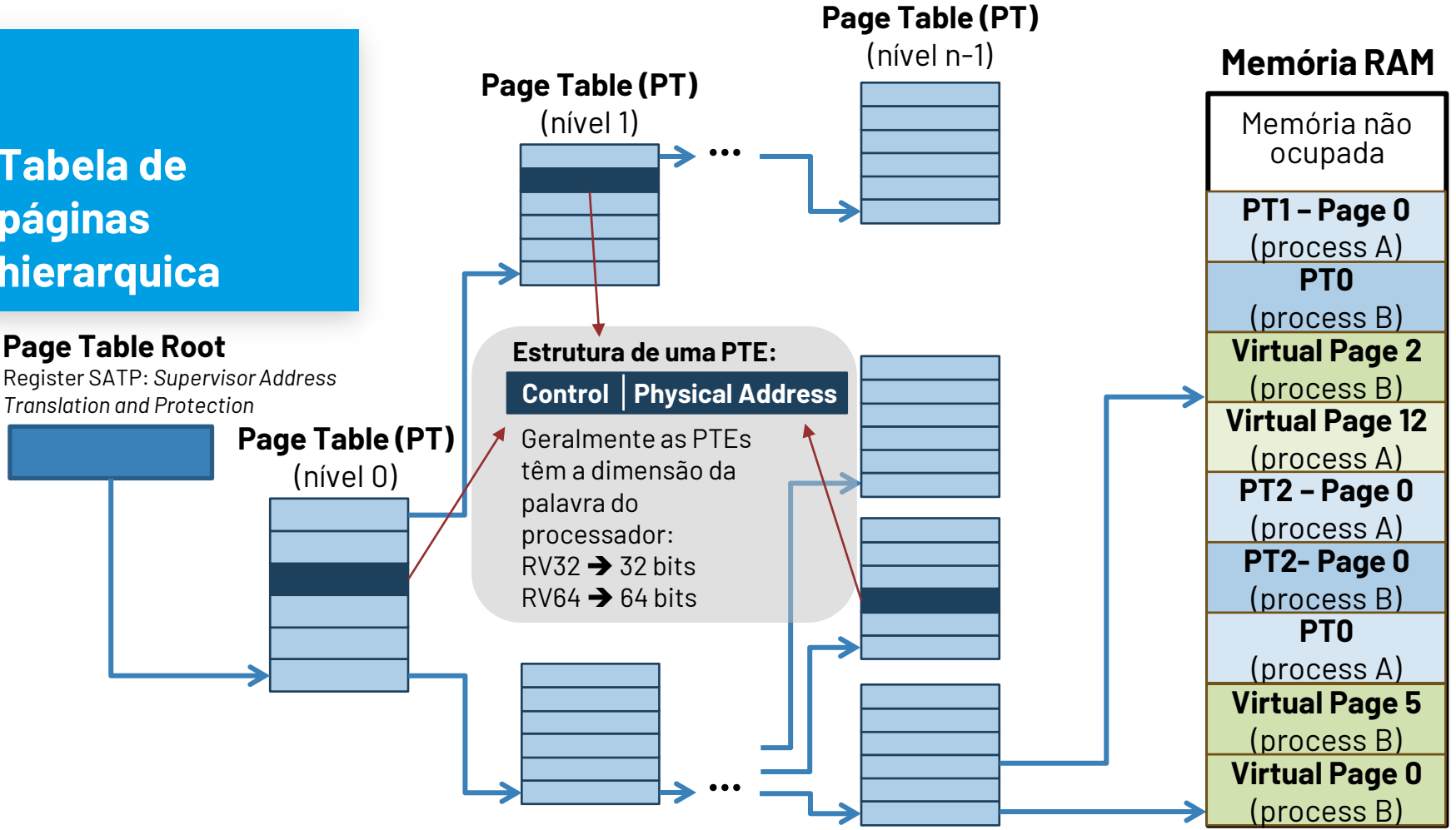
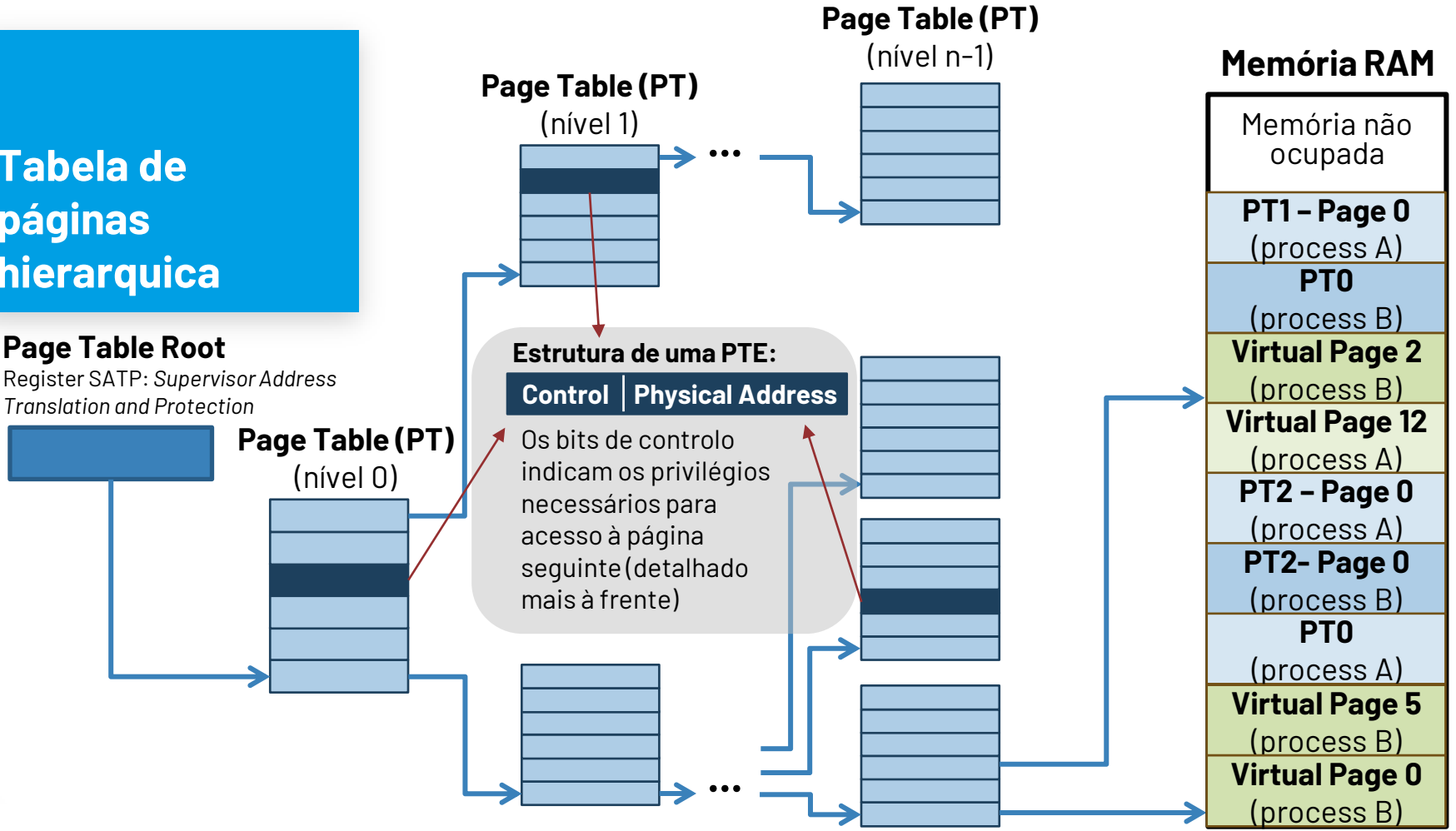


Tabela de páginas hierarquica



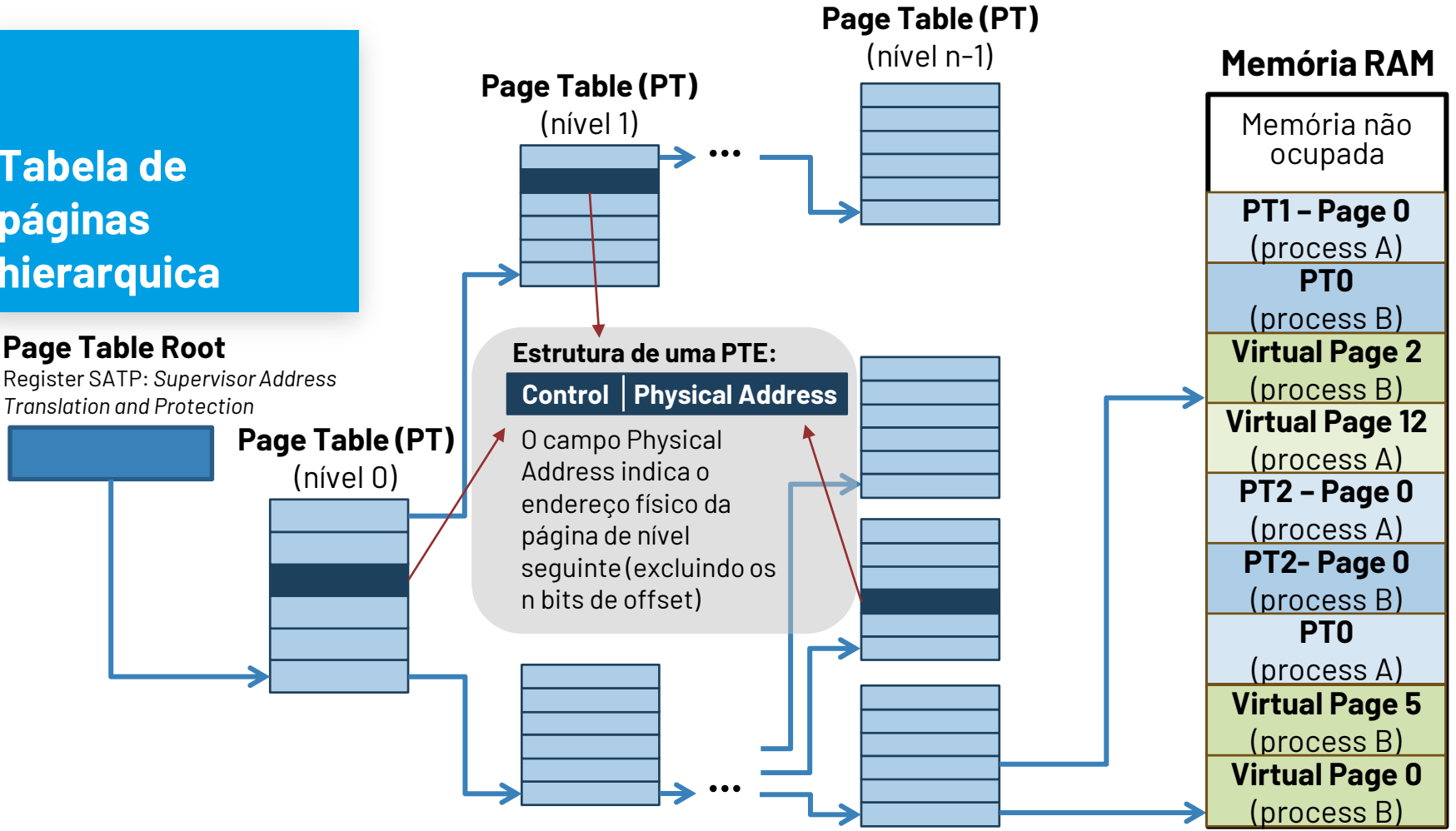
PTE – Page Table Entry

Tabela de páginas hierarquica



PTE – Page Table Entry

Tabela de páginas hierarquica

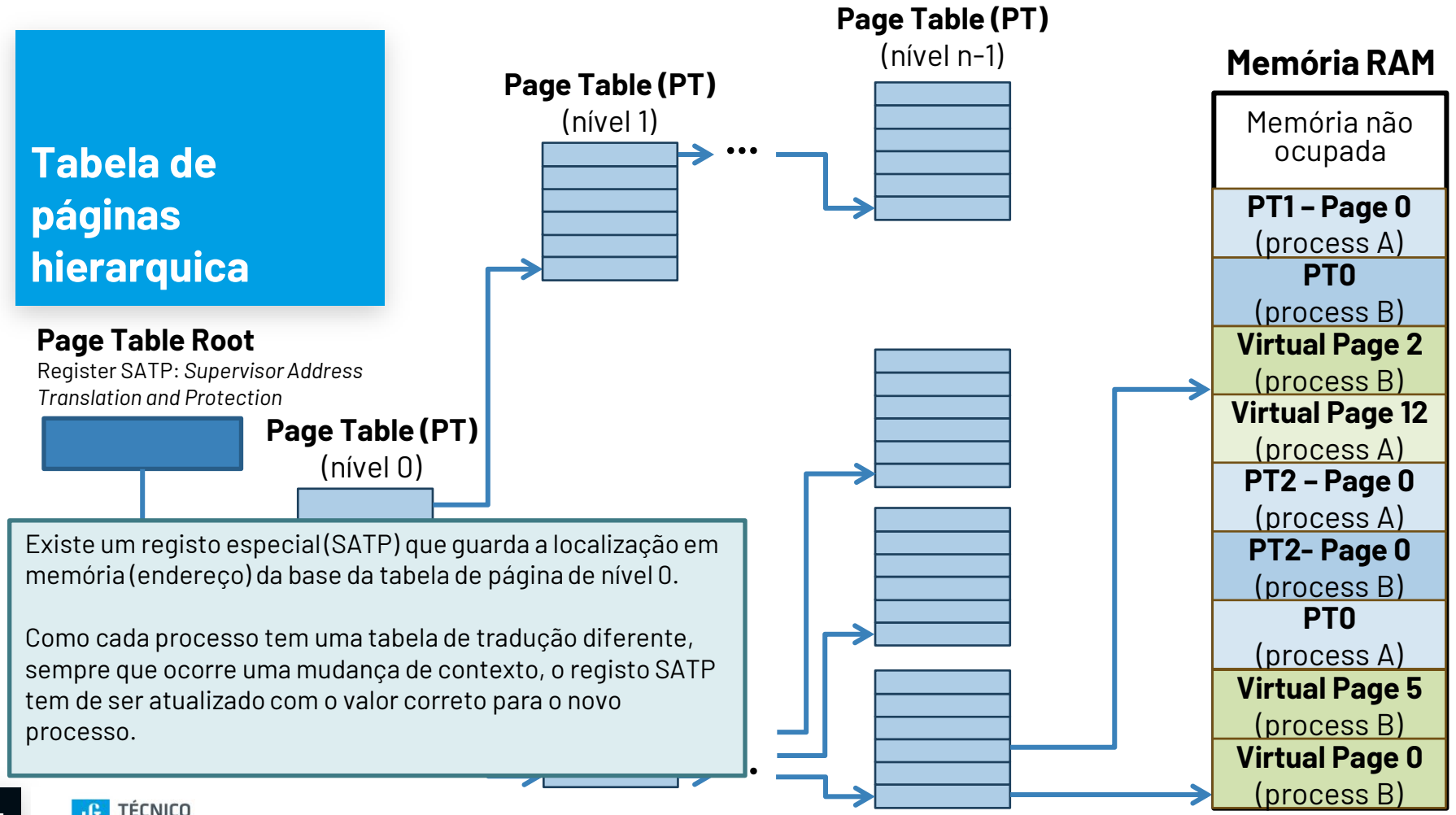


PTE – Page Table Entry

Memória Virtual

Funcionamento

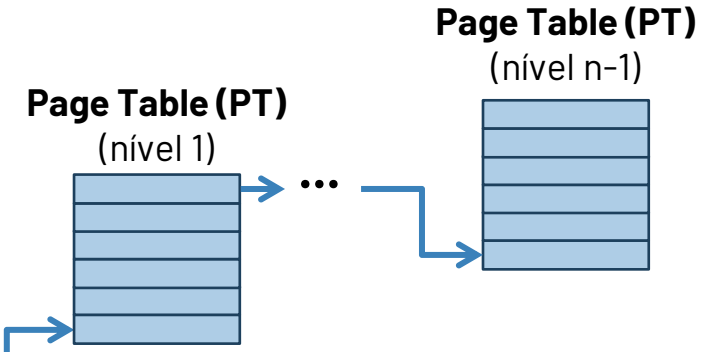
Tabela de páginas hierarquica



Existe um registo especial (SATP) que guarda a localização em memória (endereço) da base da tabela de página de nível 0.

Como cada processo tem uma tabela de tradução diferente, sempre que ocorre uma mudança de contexto, o registo SATP tem de ser atualizado com o valor correto para o novo processo.

Tabela de páginas hierarquica



Como existem vários níveis de tradução, agora é necessário repetir o processo de indexação n vezes, até se descobrir o endereço em memória.

Em cada passo de indexação i={0,1,...,n-1}, obtém-se o endereço físico da página a partir da entrada da tabela i-1 (onde PT_{-1} corresponde ao SATP), e consulta-se a entrada na tabela de acordo com o campo PTi do endereço virtual.

Nota: o acesso a uma palavra (dados ou instruções) requer agora n acessos para obter o endereço físico, mais 1 para obter a palavra

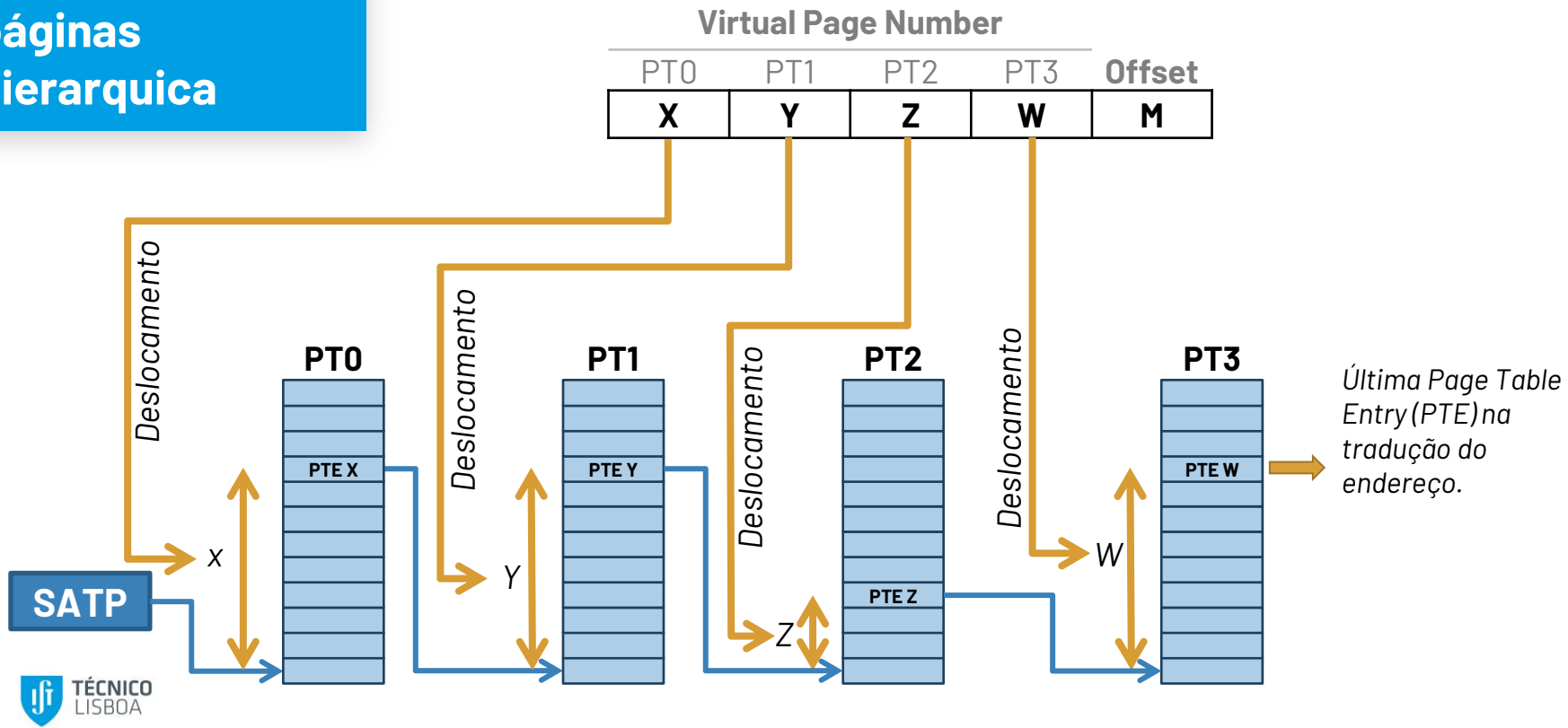
Endereço virtual

Virtual Page Number

PT0	PT1	...	PTn-1	Offset
2h	5h		3h	8h

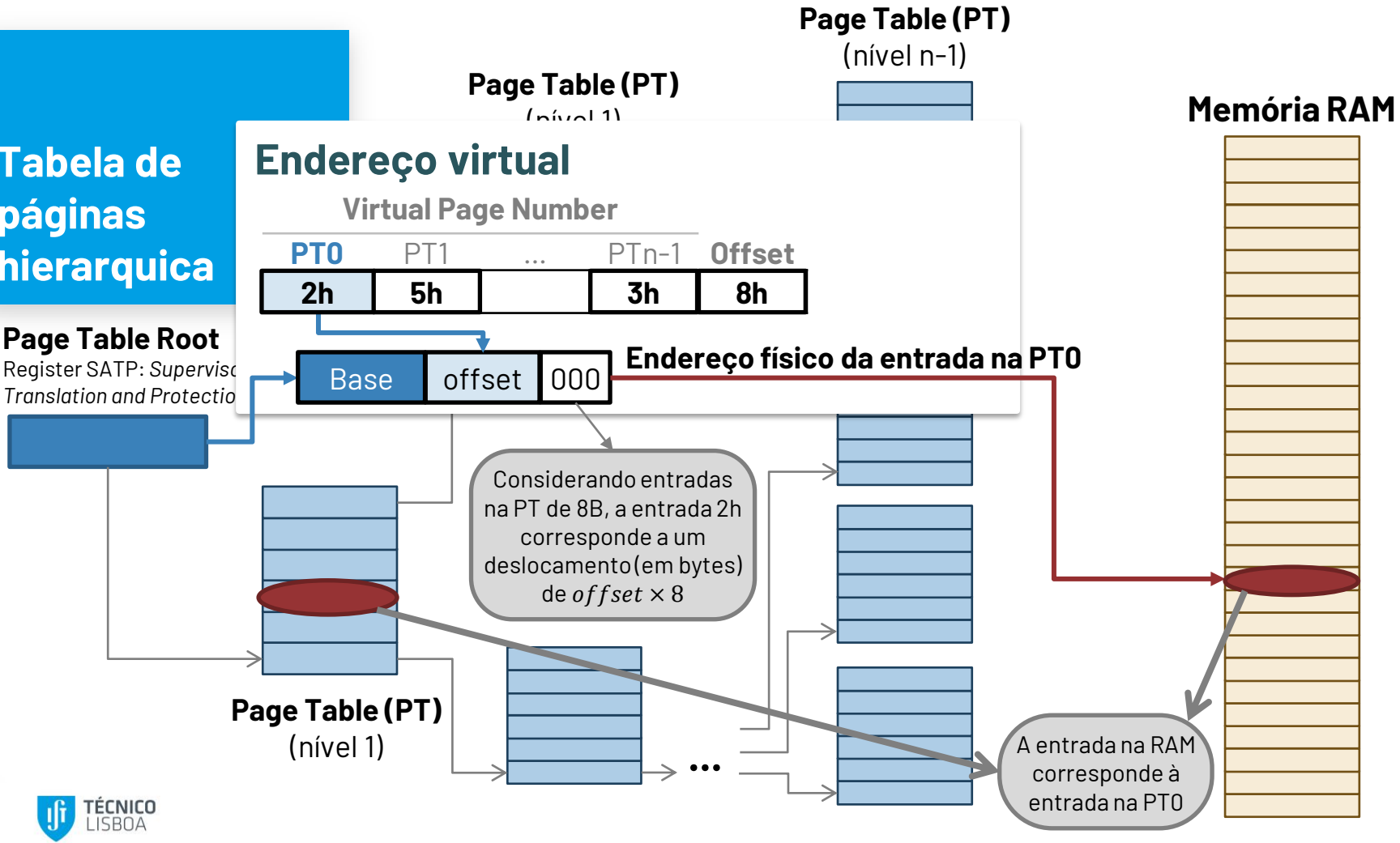
Tabela de páginas hierarquica

Exemplo: 4 níveis de tradução



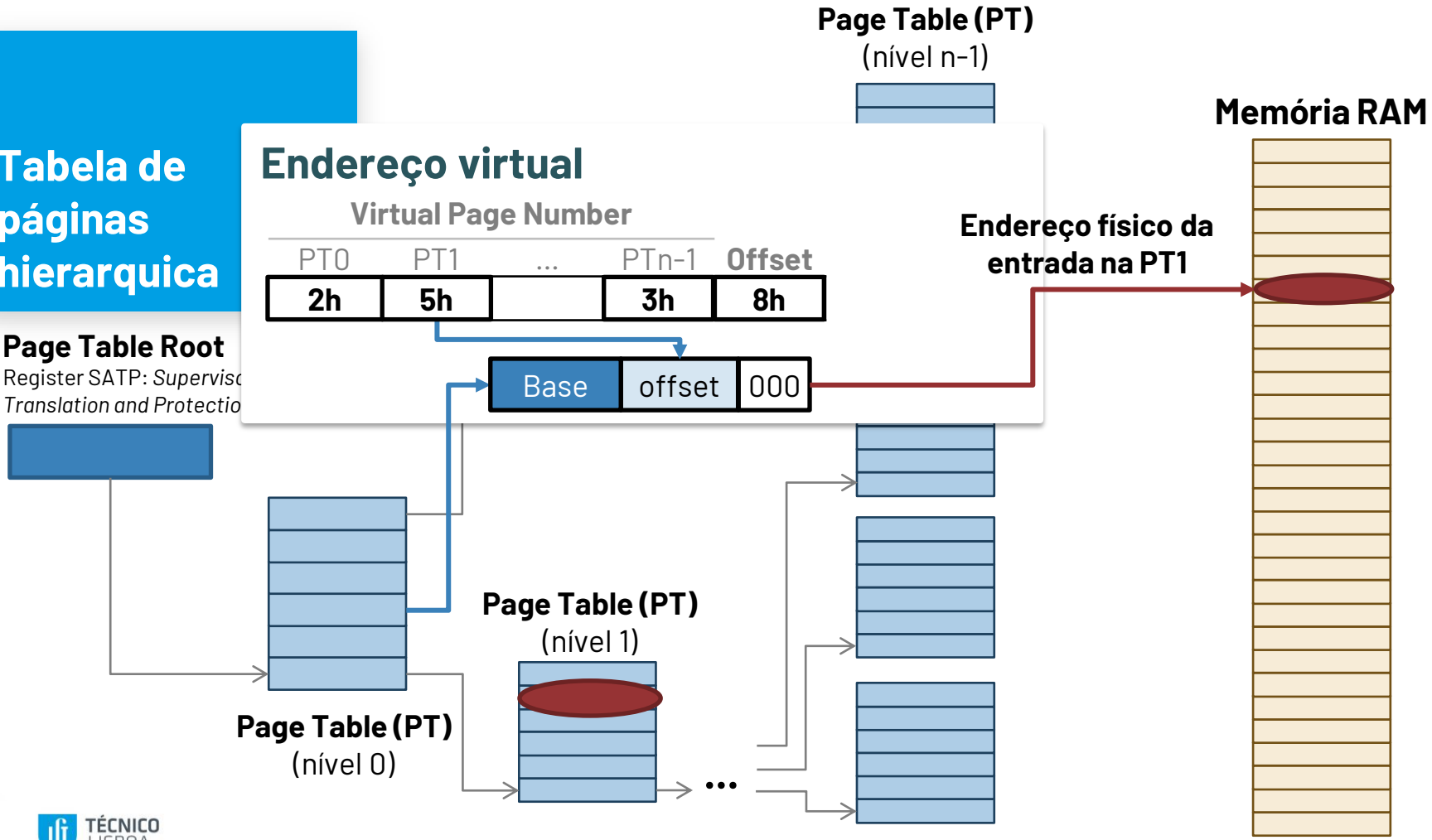
TÉCNICO LISBOA

Tabela de páginas hierarquica



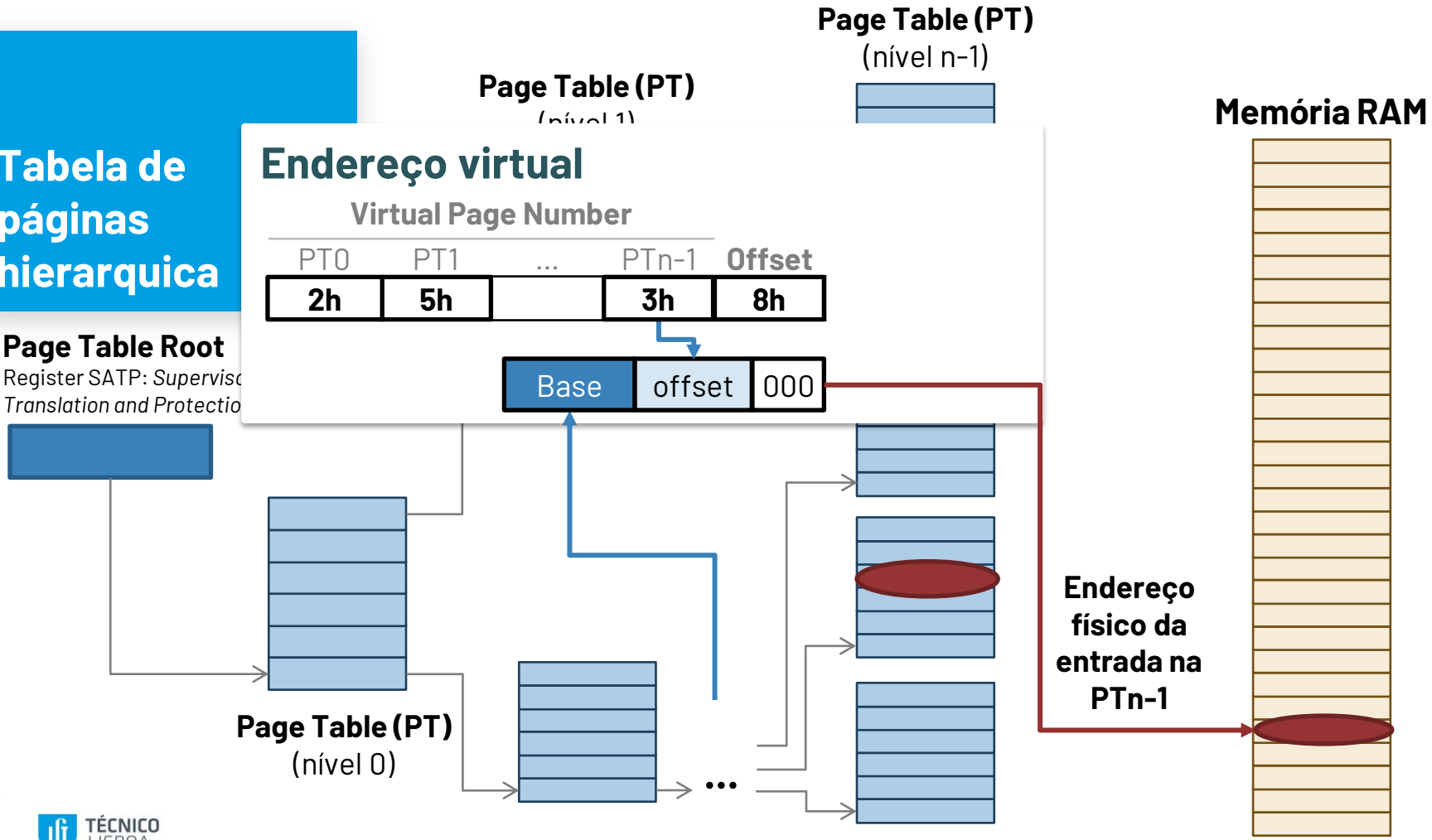
TÉCNICO LISBOA

Tabela de páginas hierarquica



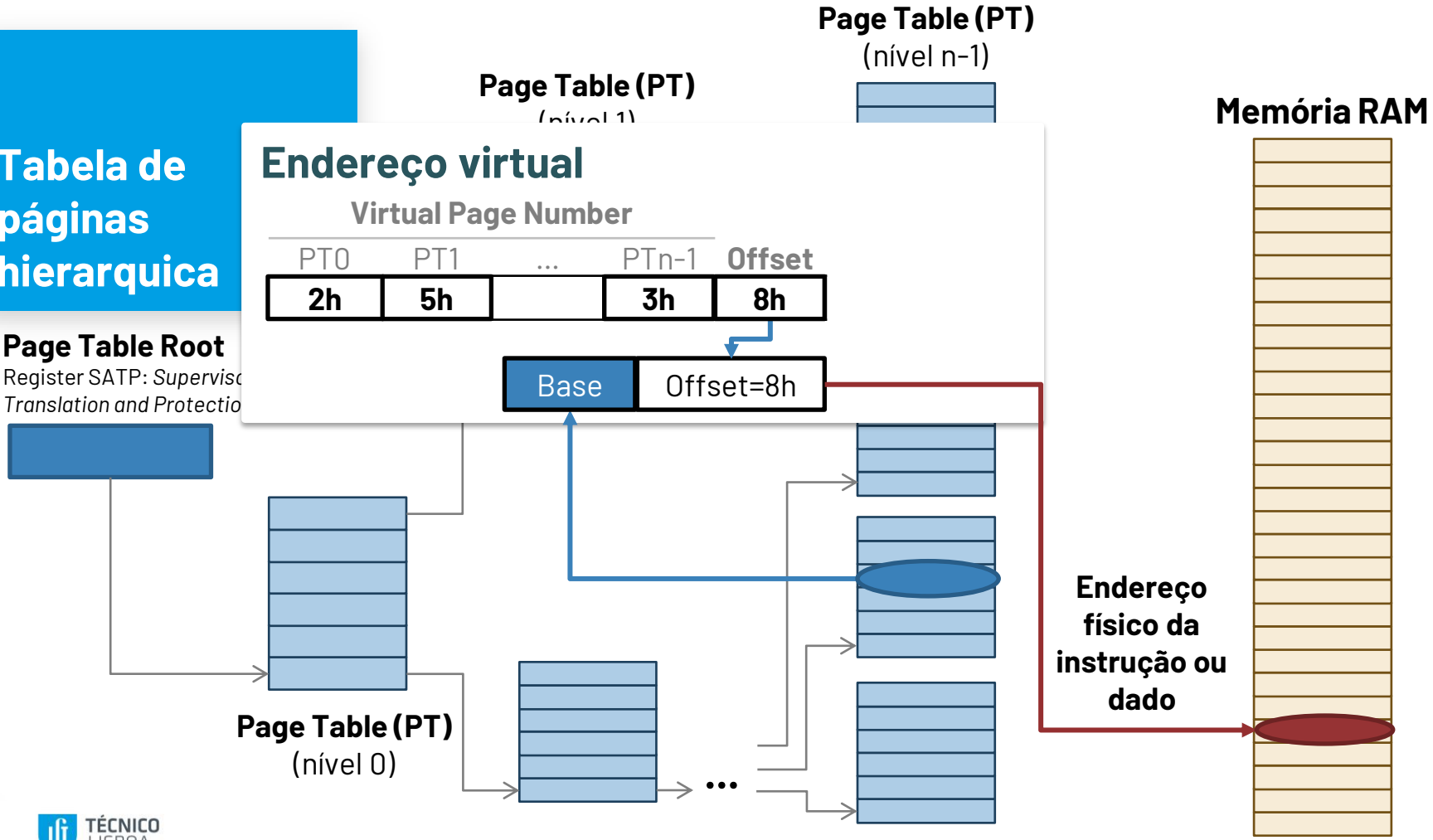
TÉCNICO LISBOA

Tabela de páginas hierarquica



TÉCNICO LISBOA

Tabela de páginas hierarquica



TÉCNICO LISBOA

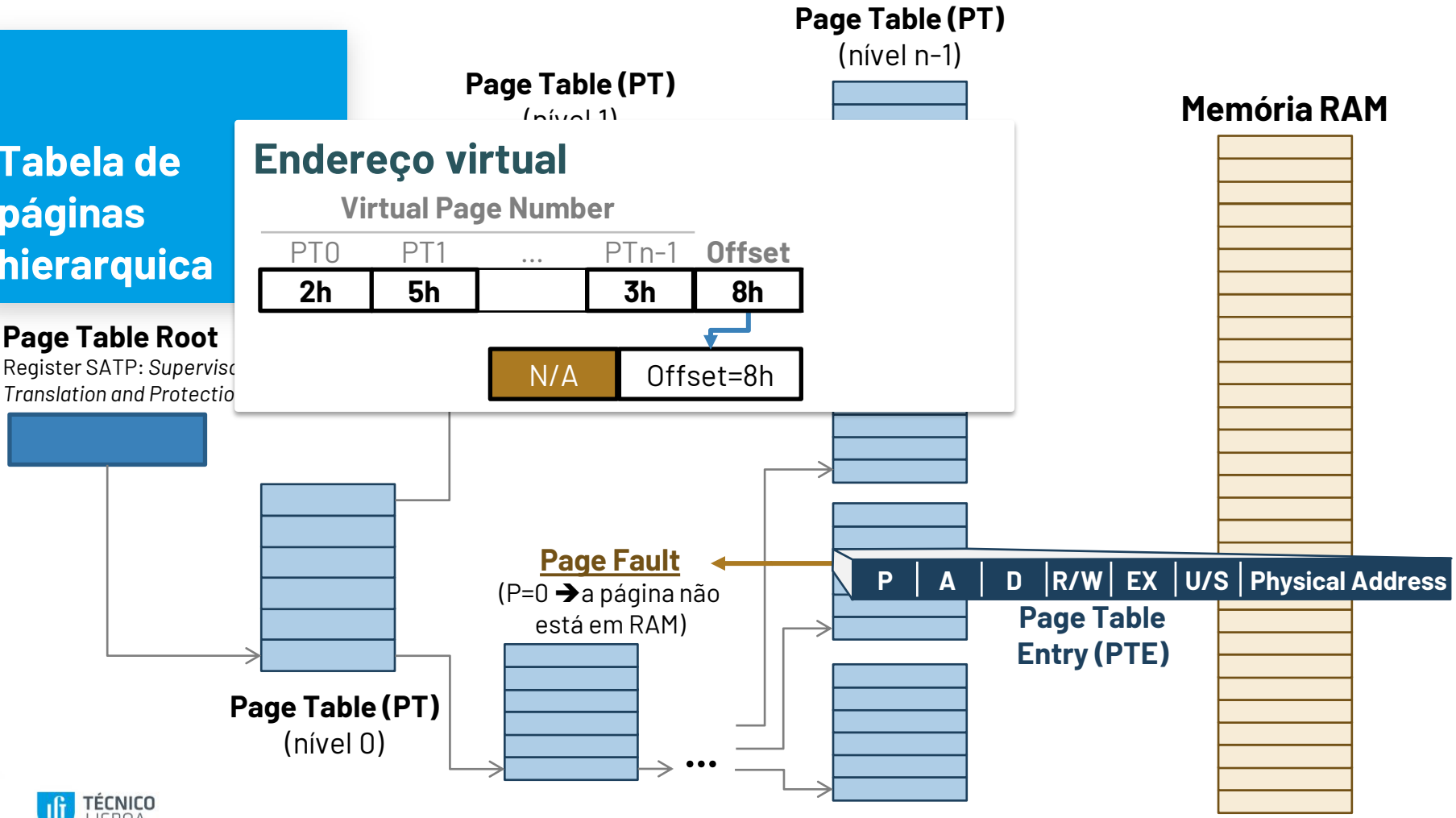
Memória Virtual

Vantagens

- A memória virtual foi inicialmente criada (anos 1960) como forma de mitigar as restrições de memória RAM

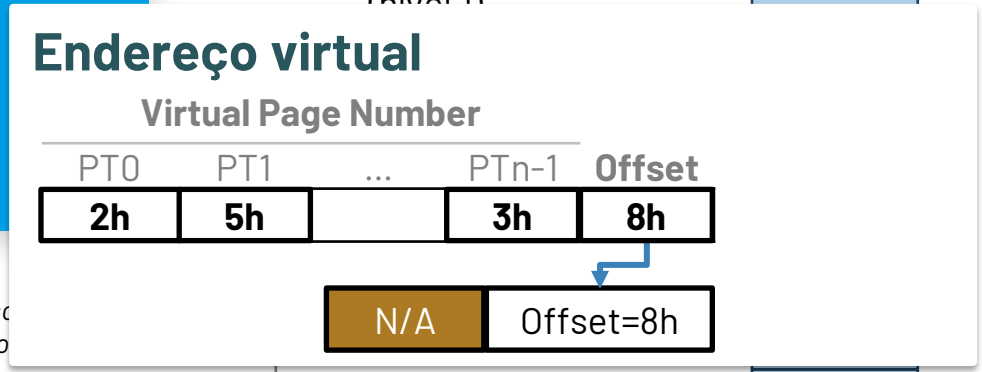
- Por exemplo (a valores actuais):
  - Um sistema com apenas 4GB de memória RAM instalada, pode funcionar como se tivesse 8GB.
  - Para que isto funcione, algumas páginas virtuais têm de ser guardadas em disco
  - Se um processo necessita de uma página em disco, o SO tem de ir buscar ao disco e coloca-la em RAM

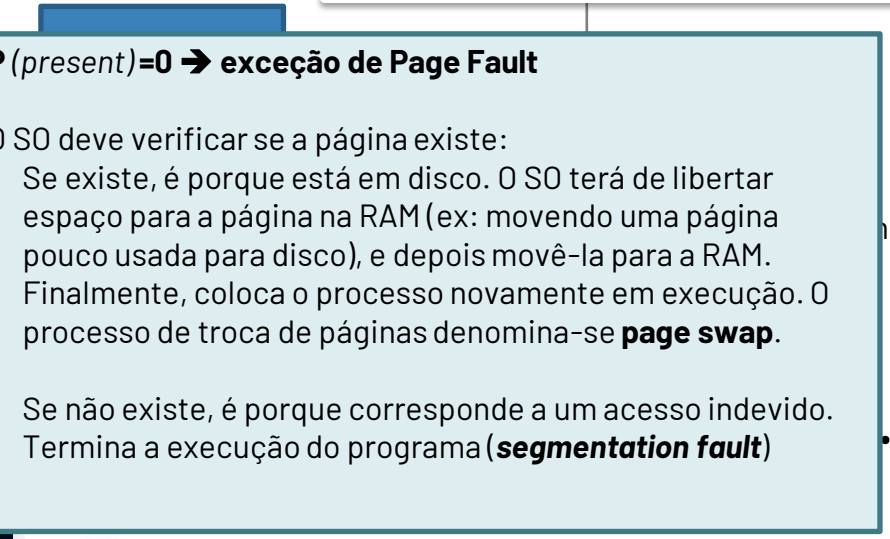
Tabela de páginas hierarquica



TÉCNICO LISBOA

Tabela de páginas hierarquica





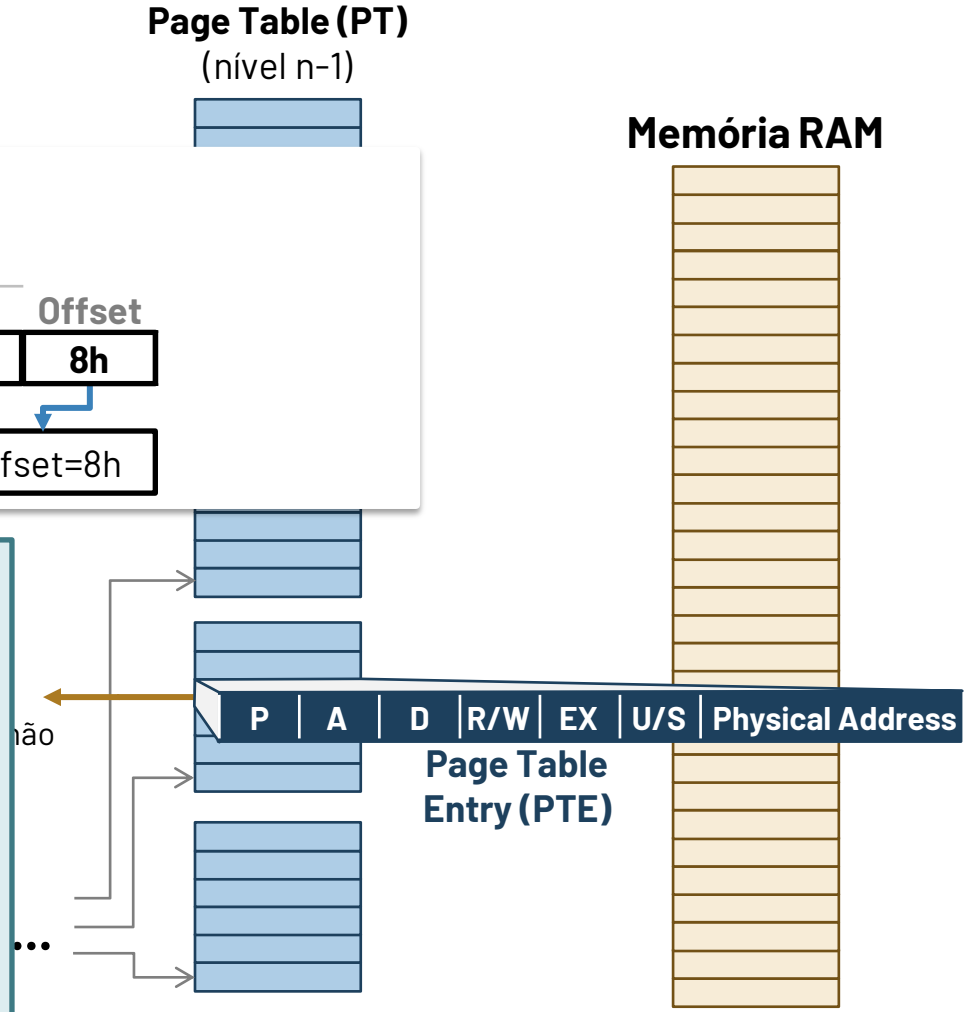
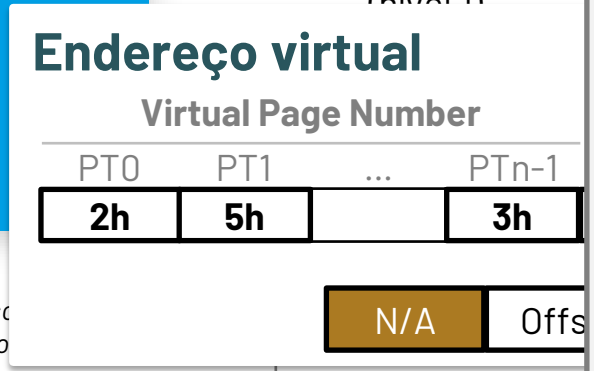


Tabela de páginas hierarquica



Page Table Root
Register SATP: *Supervisc*
Translation and Protectio

Page Table (PT)
(nível 1)

Page Table (PT)
(nível n-1)

P *(present)*=**0 ➔ exceção de Page Fault**

O SO deve verificar se a página existe:

- **EXISTE**/está em disco: o SO terá de libertar espaço para a página na RAM (ex: movendo uma página pouco usada para disco), e depois movê-la para a RAM. Finalmente, deverá colocar o processo novamente em execução. O processo de troca de páginas denomina-se **page swap**.

- **NÃO EXISTE:**
  - **OPÇÃO A**: a página ainda não foi alocada (p. ex.: o Heap/Stack cresceram para além do espaço inicialmente alocado, pelo que o SO terá de alocar mais páginas físicas em RAM.
  - **OPÇÃO B:** existe um acesso indevido a uma página que não pertence a esse processo, pelo que o SO termina a execução do programa com um ***segmentation fault***

Limitação do espaço alocado para tradução de endereços

Este processo baseado em tabelas de páginas hierárquicas permite reduzir o espaço ocupado pelas tabelas de tradução de endereço.

Numa situação original, todas as páginas, em cada um dos níveis, teriam de ser alocadas. Contudo, com a utilização do bit P, o SO apenas necessita de alocar espaço à medida que é necessário.

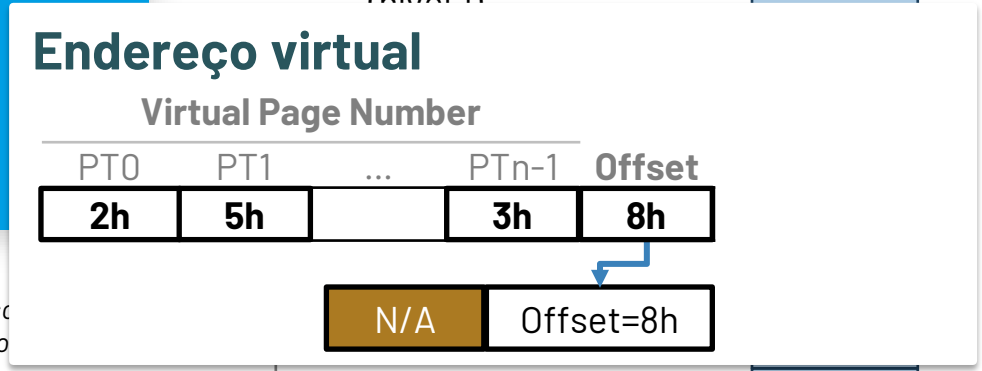
Por exemplo, na paginação (páginas de 4KB) de um programa com 7KB de código e 1KB de stack, assumindo que a stack se encontra nos endereços mais elevados e o programa nos endereços mais baixos, são necessárias:

- 2 tabelas de último nível, uma para endereçar duas páginas virtuais de código, outra para 1 página virtual de stack
- 2 tabelas em cada um dos níveis 1 a n-2, cada uma para endereçar uma entrada para o nível seguinte (2 a n-1)
- A tabela de nível 0.

Assim o espaço total para tradução de endereços é de:
$2x(n-1) \times 4KB + 4KB = 4 \times (2n-1)\,KB$

Para um sistema com 4 níveis (0...3), corresponde a 28KB

Tabela de páginas hierarquica



Page Table (PT)
(nível n-1)

Page Table (PT)
(nível 1)

Memória RAM

Page Table Root
Register SATP: *Supervisc*
Translation and Protectio

Page Fault
0 ➔ a página não está em RAM)

Page Table Entry (PTE)

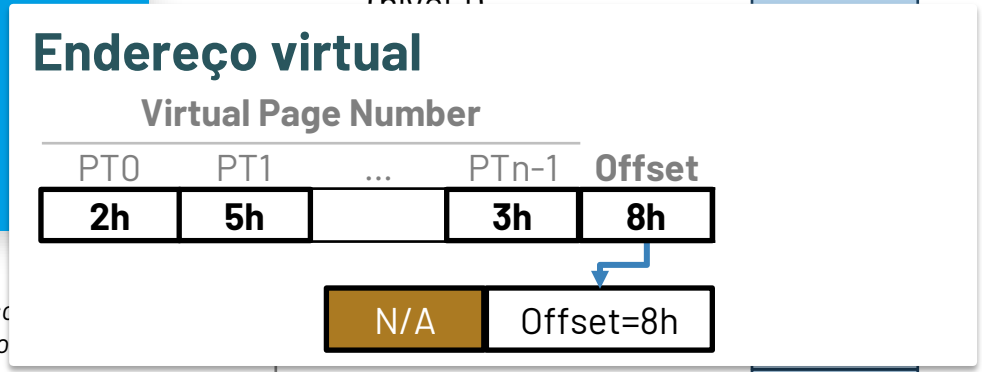
A *(accessed)* **=0 ➔ página não usada recentemente**

Página que pode ser movida para disco caso seja necessário realizar um page swap.

D *(dirty)* **=1 ➔ página que foi escrita.**

Esta página não pode sair da memória RAM sem ser re-escrita (atualizada) em disco. Na prática a RAM funciona com uma política Write-Back, Write-Allocate

Tabela de páginas hierarquica



Page Table (PT)
(nível n-1)

Page Table (PT)

Memória RAM

Page Table Root
Register SATP: *Supervisc*
Translation and Protectio

A *(accessed)* **=0 ➔ página não usada recentemente**

Página que pode ser movida para disco caso seja necessário realizar um page swap.

D *(dirty)* **=1 ➔ página que foi escrita.**

Esta página não pode sair da memória RAM sem ser re-escrita (atualizada) em disco. Na prática a RAM funciona com uma política Write-Back, Write-Allocate

Uma política de escrita do tipo Write-Through é impraticável!

Num sistema de memória cache, usa-se um ***Write Buffer*** para esconder a latência da escrita no nível de memória seguinte. Isto é realizável porque a latência de escrita no nível seguinte demora entre alguns ciclos e algumas centenas de ciclos.

Contudo, a escrita em disco geralmente demora milhões de ciclos do processador, o que torna impraticável a utilização de um write-buffer. Assim, a utilização de uma política do tipo Write-Through é totalmente impraticável.

Memória Virtual

Vantagens

- A memória virtual foi inicialmente criada (anos 1960) como forma de mitigar as restrições de memória RAM:
  - Um sistema com apenas 4GB de memória RAM instalada, pode funcionar como se tivesse 8GB.
  - Para que isto funcione, algumas páginas virtuais têm de ser guardadas em disco
  - Se um processo necessita de uma página em disco, o SO tem de ir buscar ao disco e coloca-la em RAM

Memória Virtual

Vantagens

TÉCNICO LISBOA

- A memória virtual foi inicialmente criada (anos 1960) como forma de mitigar as restrições de memória RAM

- Assim, na prática, a memória RAM funciona como uma cache:
  - **Linhas** (páginas virtuais) **de grande dimensão**: ex: 4KB, 8KB, 64KB, 2MB, 1GB
  - Organização **completamente associativa**: uma página virtual pode ser colocada em qualquer zona da memória
  - Política de escrita: Write-Back, Write Allocate
  - Política de substituição: variante de Not Recently Used (NRU)

Memória Virtual

Vantagens

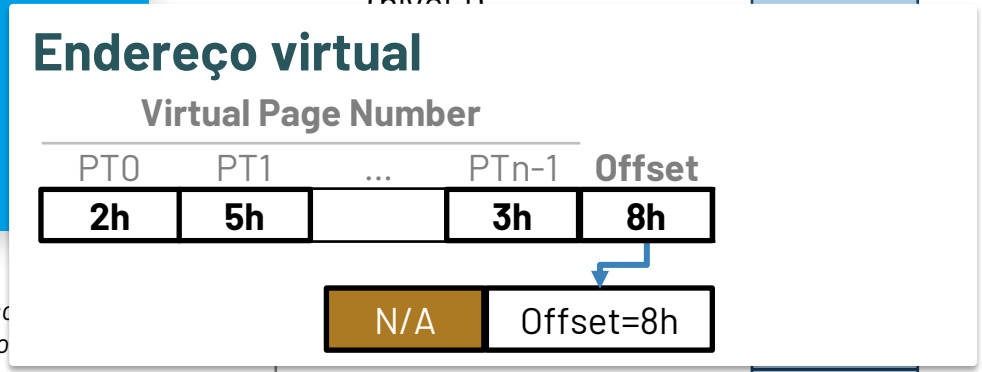
TÉCNICO LISBOA

- A memória virtual foi inicialmente criada (anos 1960) como forma de mitigar as restrições de memória RAM

- Assim, na prática, a memória RAM funciona como uma cache:
  - **Linhas** (páginas virtuais) **de grande dimensão**: ex: 4KB, 8KB, 64KB, 2MB, 1GB
  - Organização **completamente associativa**: uma página virtual pode ser colocada em qualquer zona da memória
  - Política de escrita: Write-Back, Write Allocate
  - Política de substituição: variante de Not Recently Used (NRU)

- Hoje em dia, a grande vantagem da memória virtual é que fornece isolamento entre processos e proteção de acesso a dados críticos

Tabela de páginas hierarquica



Page Table (PT)
(nível n-1)

Page Table (PT)
(nível 1)

Memória RAM

Page Table Root
Register SATP: *Supervisc*
Translation and Protectio

R/W *(read/Write)* ➔ **Permissões de escrita (load/store) na memória.**

EX *(execute)* ➔ **Permissões de execução** (se EX=1, a página contém instruções; se EX=0 a página contém dados)

U/S *(User/Supervisor privileges)* ➔ **Permissões de de acesso** (identifica se a página pode ser acedida pela aplicação ou só em modo SO/kernel)

Page Fault
➔a página não
stá em RAM)

P	A	D	R/W	EX	U/S	Physical Address

Page Table Entry (PTE)

TÉCNICO LISBOA

TLB

Translation Lookaside Buffer

Como reduzir o tempo de tradução de endereços?

Problema

- Na prática o sistema de memória virtual (sem as modificações descritas de seguida) aumenta a latência no acesso aos dados e instruções:
  - Sem memória virtual, para um sistema de memória cache com 2 níveis (L1-I/L1-D e L2), o tempo de acesso aos dados ou instruções era determinado por:

$$T_{acesso\ x} = T_{L1-x} + MR_{L1-x} \times (T_{L2} + MR_{L2} \times T_{DRAM}), \qquad x \in \{I, D\}$$

Como reduzir o tempo de tradução de endereços?

Problema

TÉCNICO LISBOA

- Na prática o sistema de memória virtual (sem as modificações descritas de seguida) aumenta a latência no acesso aos dados e instruções:
  - Sem memória virtual, para um sistema de memória cache com 2 níveis (L1-I/L1-D e L2), o tempo de acesso aos dados ou instruções era determinado por:

$$T_{acesso\ x} = T_{L1-x} + MR_{L1-x} \times (T_{L2} + MR_{L2} \times T_{DRAM}), \qquad x \in \{I, D\}$$

  - Mas com memória virtual (usando uma hierarquia de 4 níveis para tradução de endereços):

$$T_{acesso\ x} = T_{Tradução} + T_{L1-x} + MR_{L1-x} \times (T_{L2} + MR_{L2} \times T_{DRAM}), \qquad x \in \{I, D\}$$
$$T_{tradução} = 4 \times (T_{L1-x} + MR_{L1-x} \times (T_{L2} + MR_{L2} \times T_{DRAM}))$$

Assumindo que o acesso à tabela de tradução de endereços passa pela cache de dados (LI-D) ou instruções (LI-I), conforme o caso, e que o miss rate em cada um dos níveis é igual na tradução e na leitura/escrita dos dados/instruções.

Como reduzir o tempo de tradução de endereços?

Problema

- Na prática
  - Sem m
    tempo
  - Mas co

Na prática, o miss-rate de tradução é bastante diferente do miss-rate de acesso aos dados/instruções

Porquê?

Como cada página tem uma dimensão razoável (ex: 4kB), todos os endereços virtuais (acedidos dentro de uma determinada página) vão ter a mesma tradução

Ou seja:
A localidade temporal e espacial da tradução é válida numa gama de 4K endereços

$$T_{acesso\ x} = T_{Tradução} + T_{L1-x} + MR_{L1-x} \times (T_{L2} + MR_{L2} \times T_{DRAM}), \qquad x \in \{I, D\}$$
$$T_{tradução} = 4 \times (T_{L1-x} + MR_{L1-x} \times (T_{L2} + MR_{L2} \times T_{DRAM}))$$

Assumindo que o acesso à tabela de tradução de endereços passa pela cache de dados (LI-D) ou instruções (LI-I), conforme o caso, e que o miss rate em cada um dos níveis é igual na tradução e na leitura/escrita dos dados/instruções.

Como reduzir o tempo de tradução de endereços?

Solução: introduzir uma cache para tradução de endereços

TÉCNICO LISBOA

- Para reduzir o tempo de tradução, geralmente utiliza-se uma cache (TLB – *Translation Lookaside Buffer*) para acelerar a tradução

- A TLB tradicionalmente corresponde a uma cache endereçada pela virtual page number:
  - Linhas de 1 ou 2 entradas, onde cada entrada corresponde a uma **page table entry (PTE)**
  - Associatividade elevada (tradicionalmente completamente associativa, hoje com uma estrutura *set-associative*)

Considerando páginas de 4KB (12 bits para offset, 52 para virtual page number) a *Virtual Page Number* do endereço:

1234 A318h

corresponde a

1234Ah=0001 0010 0011 0100 1010b

Considerando:

- linhas na TLB para 2 entradas (1 bit de offset)
- uma cache completamente associativa (0 bits para índice)
- A TAG contém 51 bits

Como reduzir o tempo de tradução de endereços?

Solução: introduzir uma cache para tradução de endereços

TÉCNICO LISBOA

- Para reduzir o tempo de tradução, geralmente utiliza-se uma cache (TLB – *Translation Lookaside Buffer*) para acelerar a tradução

- A TLB tradicionalmente corresponde a uma cache endereçada pela virtual page number:
  - Linhas de 1 ou 2 entradas, onde cada entrada corresponde a uma **page table entry (PTE)**
  - Associatividade elevada (tradicionalmente completamente associativa, hoje com uma estrutura *set-associative*)

- De forma a possibilitar mudanças de contexto, a TLB necessita ainda de uma entrada com o identificador do processo (Process ID – PID)

Considerando páginas de 4KB (12 bits para offset, 52 para virtual page number) a *Virtual Page Number* do endereço:

1234 A318h

corresponde a

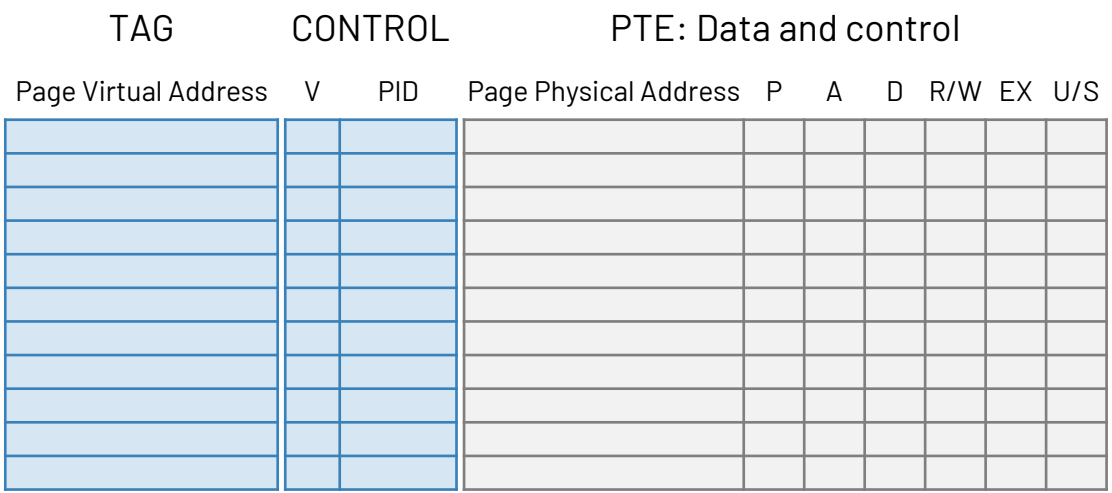
1234Ah=0001 0010 0011 0100 1010b

Considerando:

- linhas na TLB para 2 entradas (1 bit de offset)
- uma cache completamente associativa (0 bits para índice)
- A TAG contém 51 bits

Estrutura de uma TLB

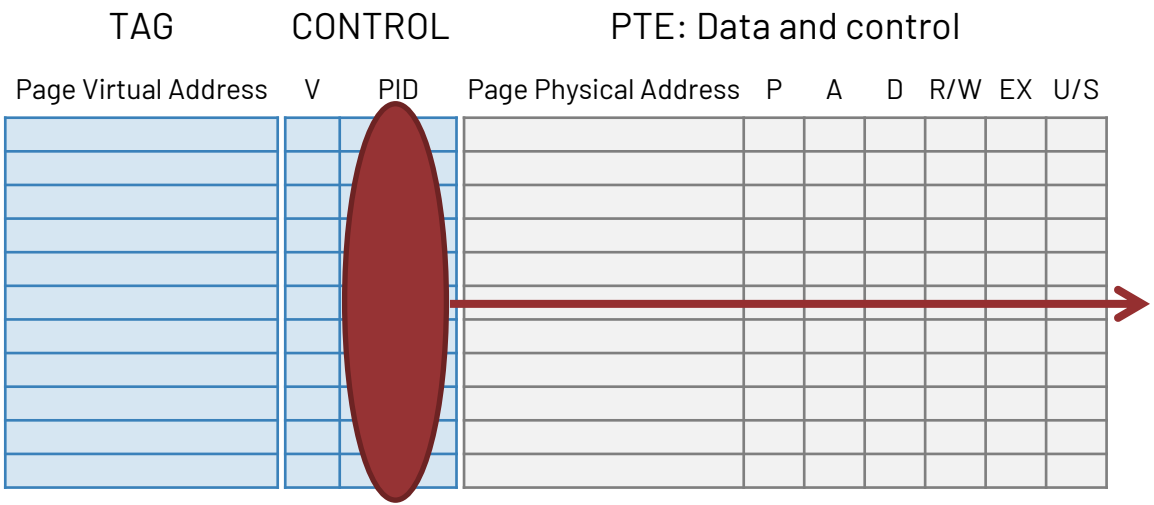
Considerando uma TLB com uma PTE por linha, completamente associativa



Como a cache é totalmente associativa, não são usados quaisquer bits para índice. Sempre que a TLB é acedida, são verificadas todas as entradas (i.e., cada entrada corresponde a uma via diferente).

Estrutura de uma TLB

Considerando uma TLB com uma PTE por linha, completamente associativa



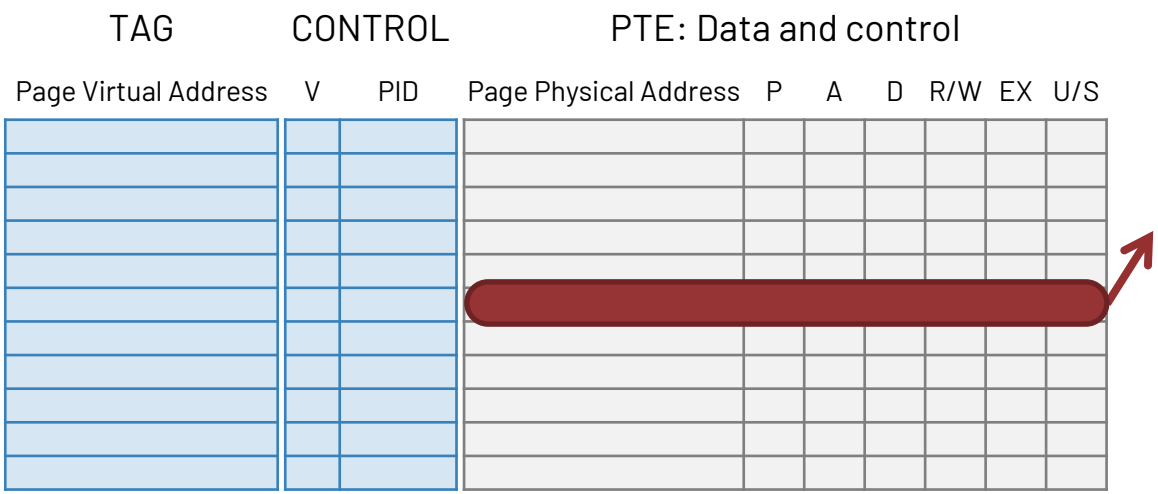
Como a cache é totalmente associativa, não são usados quaisquer bits para índice. Sempre que a TLB é acedida, são verificadas todas as entradas (i.e., cada entrada corresponde a uma via diferente).

Dado que a TLB é endereçada virtualmente, e como todos os processos têm o mesmo mapa de memória virtual, é necessário distinguir se a entrada na TLB corresponde a um processo A ou a um outro processo B. Isso é realizado com o campo PID (Process ID). Assim, existe um HIT na TLB se:

- V=1
- $TAG_{TLB}=TAG_{END.\ VIRTUAL}$
- $PID_{TLB}=PID_{PROCESSO}$

Estrutura de uma TLB

Considerando uma TLB com uma PTE por linha, completamente associativa



Como a cache é totalmente associativa, não são usados quaisquer bits para índice. Sempre que a TLB é acedida, são verificadas todas as entradas (i.e., cada entrada corresponde a uma via diferente).

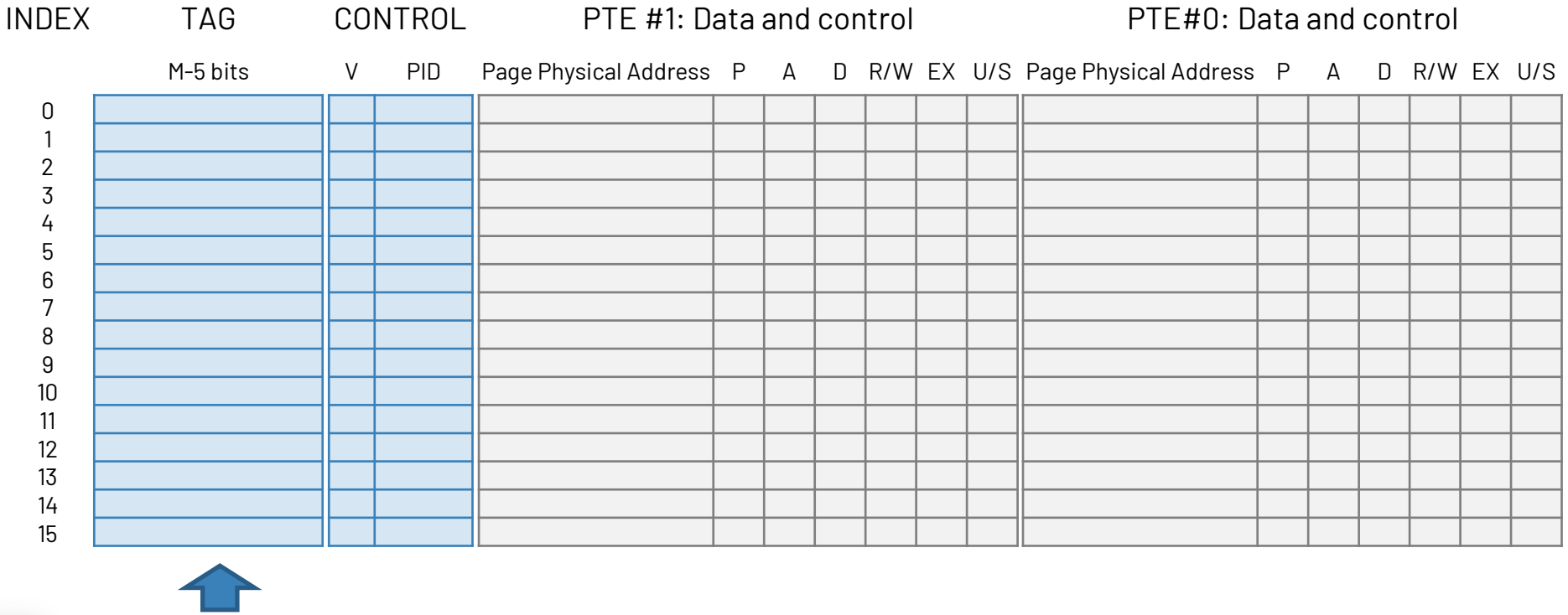
Sempre que a TLB tem um MISS, o processador vai realizar o processo de tradução manual percorrendo os N níveis da tabela de tradução hierárquica (este mecanismo é realizado por um módulo de hardware geralmente denominado de *Page-Table Walker*).

Ao realizar o último nível de tradução, o módulo de hardware (*Page-Table Walker*) guarda a entrada da última Page Table na TLB.

Da próxima vez que existir um acesso à mesma página virtual, a TLB indica logo a PTE do último nível de tradução

Estrutura de uma TLB

Considerando uma TLB set-associative, com duas PTEs por linha

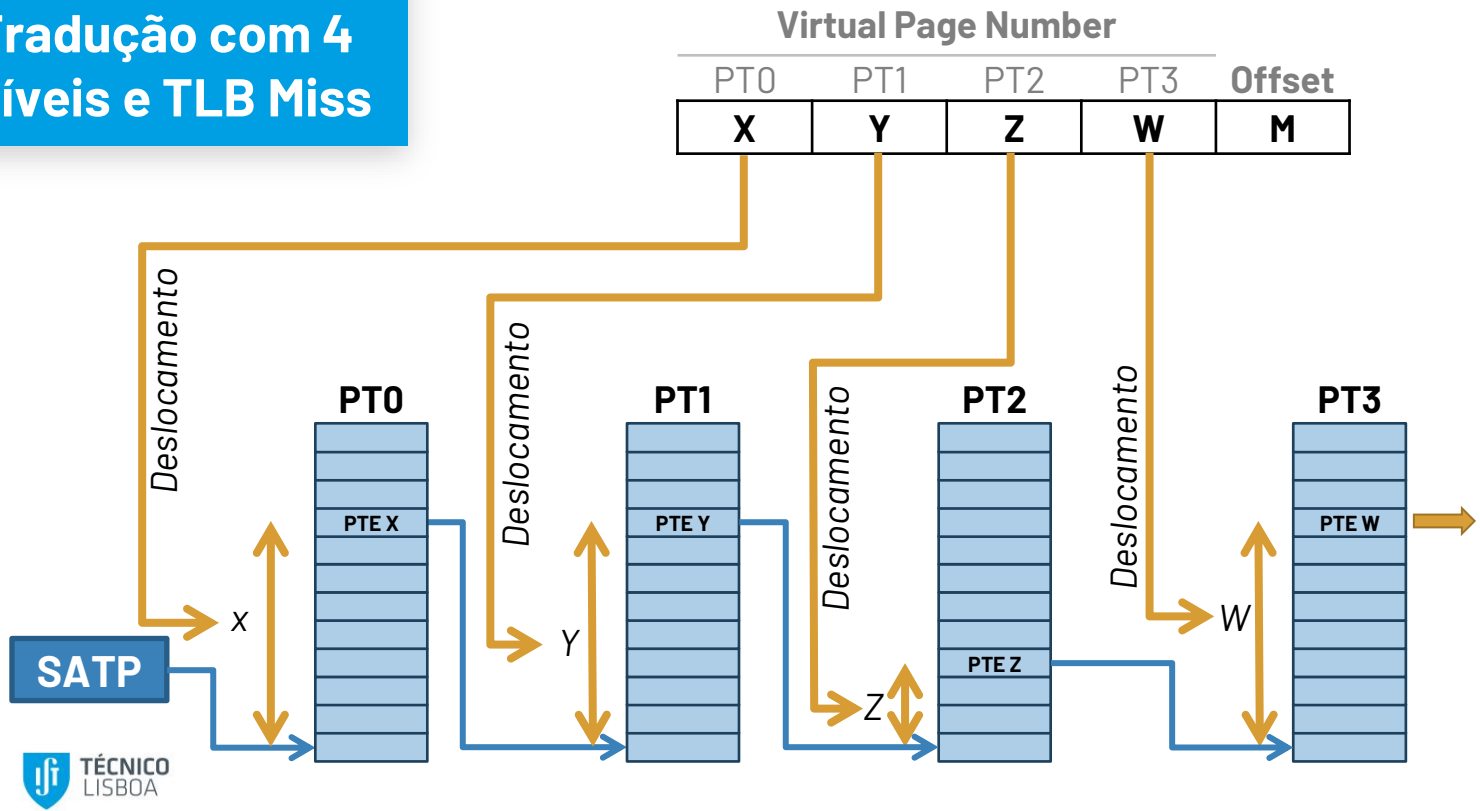


Os M bits do *Virtual Page Number* dividem-se em: 1 bit para offset, 4 bits para índice, restantes para TAG

Tradução com 4 níveis e TLB Miss

Estrutura da PTE em cada um dos níveis

P	A	D	R/W	EX	U/S	Physical Address



Última Page Table Entry (PTE) na tradução do endereço.

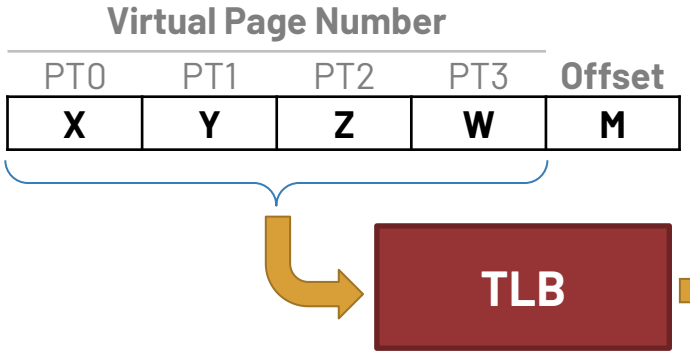
Coloca a entrada (PTE) na TLB e usa o campo Physical Address para obter o endereço físico

TÉCNICO LISBOA

Tradução com TLB Hit

Estrutura da PTE em cada um dos níveis

P	A	D	R/W	EX	U/S	Physical Address



Última Page Table Entry (PTE W) na tradução do endereço.

Usa o campo Physical Address para obter o endereço físico

TÉCNICO LISBOA

Como reduzir o tempo de tradução de endereços?

Solução

TÉCNICO LISBOA

- Na prática o sistema de memória virtual (sem as modificações descritas de seguida) aumenta a latência no acesso aos dados e instruções:
  - Sem memória virtual, para um sistema de memória cache com 2 níveis (L1-I/L1-D e L2), o tempo de acesso aos dados ou instruções era determinado por:

$$T_{acesso\ x} = T_{L1-x} + MR_{L1-x} \times (T_{L2} + MR_{L2} \times T_{DRAM}), \qquad x \in \{I, D\}$$

  - Com memória virtual (usando uma hierarquia de 4 níveis e 1 nível de TLB):

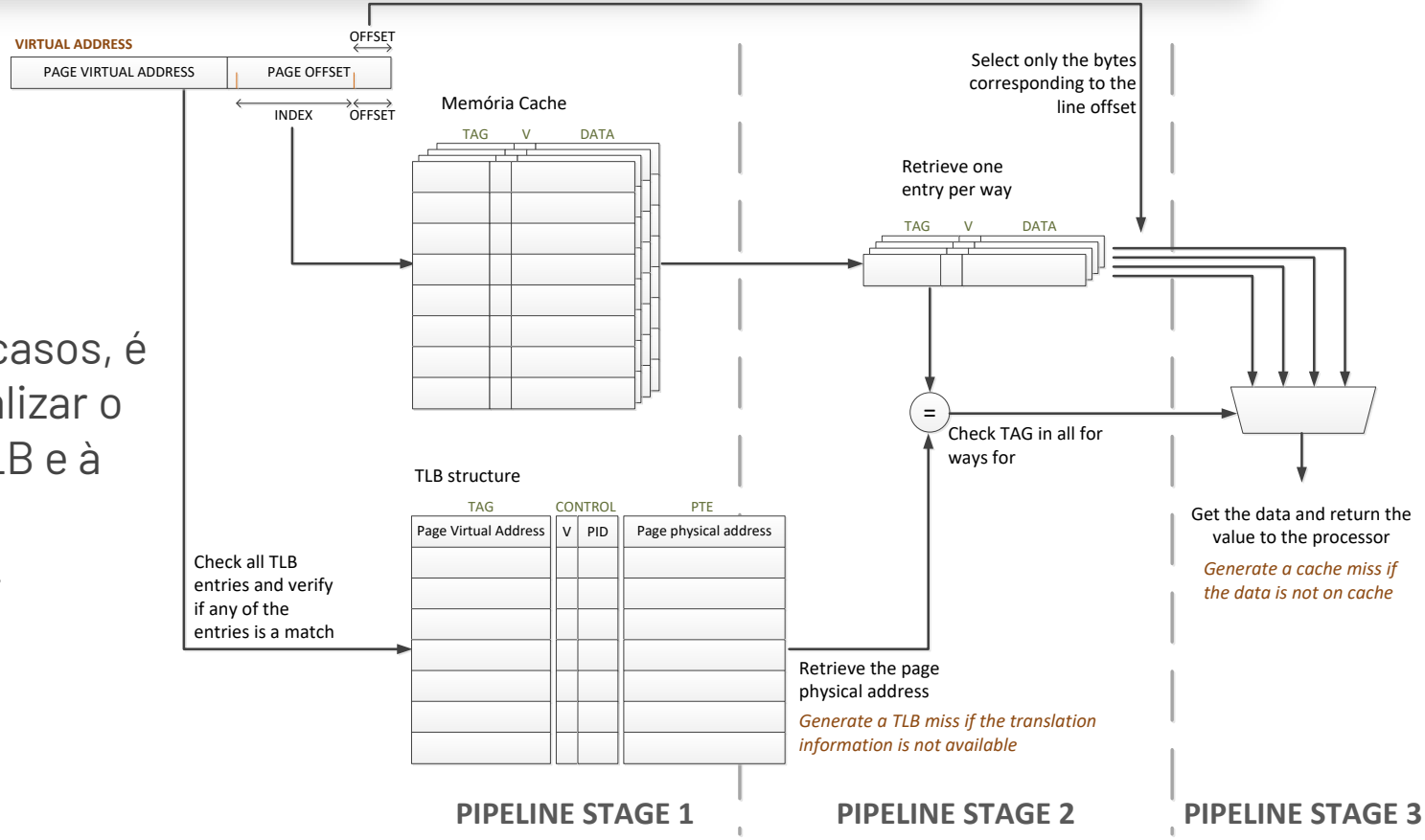
$$T_{acesso\ x} = T_{Tradução} + T_{L1-x} + MR_{L1-x} \times (T_{L2} + MR_{L2} \times T_{DRAM}), \qquad x \in \{I, D\}$$
$$T_{tradução} = T_{TLB} + MR_{TLB} \times [4 \times (T_{L1-x} + MR_{L1-x} \times (T_{L2} + MR_{L2} \times T_{DRAM}))]$$

Como o miss-rate na TLB é geralmente muito baixo (próximo de zero), o custo da tradução de endereços é baixo.

Como reduzir o tempo de tradução de endereços?

Solução

Em alguns casos, é possível realizar o acesso à TLB e à cache em simultâneo.



Como reduzir o tempo de tradução de endereços?

Solução

- Em alguns casos, é possível realizar o acesso à TLB e à cache em simultâneo.

- Nesse caso o tempo de acesso à TLB fica escondido, obtendo-se:

  - Se existe um HIT na TLB:

$$T_{acesso\ x} = T_{L1-x} + MR_{L1-x} \times (T_{L2} + MR_{L2} \times T_{DRAM}), \qquad x \in \{I, D\}$$

  - Se existe um MISS na TLB, o tempo de acesso (incluindo tradução) é:

$$T_{acesso\ x} = 4 \times \left(T_{L1-x} + MR_{Trad.L1-x} \times (T_{L2} + MR_{Trad.L2} \times T_{DRAM})\right) + T_{L1-x} + MR_{L1-x} \times (T_{L2} + MR_{L2} \times T_{DRAM}), \qquad x \in \{I, D\}$$

  - Assim, em média obtém-se:

$$T_{acesso\ x} = MR_{TLB} \times 4 \times \left(T_{L1-x} + MR_{Trad.L1-x} \times (T_{L2} + MR_{Trad.L2} \times T_{DRAM})\right) + T_{L1-x} + MR_{L1-x} \times (T_{L2} + MR_{L2} \times T_{DRAM}), \qquad x \in \{I, D\}$$

Tabela de páginas hierarquica

EXERCÍCIO #1

Tabela de páginas hierarquica

Exercício

Considere um sistema de memória virtual com páginas de 4KB, 40 bits de endereço físico e 48 bits de endereço virtual.

1) Decomponha o endereço virtual nos vários campos e determine o número de níveis para tradução de um endereço virtual em endereço físico.

2) Esboce o esquema que permite a tradução do endereço virtual 0100 7FE0 2001h em endereço físico.

3) Determine o espaço mínimo requirido pelo sistema de tradução (tabela de páginas hierárquica) admitindo que o programa se divide em:
   - Código: 13KB
   - Dados: 1.5MB
   - Heap: 27MB
   - Stack: 14KB

Tabela de páginas hierarquica

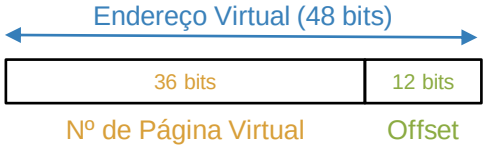
Páginas de 4kB
End. Físico: 40 bits
End. Virtual: 48 bits

Considere um sistema de memória virtual com páginas de 4KB, 40 bits de endereço físico e 48 bits de endereço virtual.

1) Decomponha o endereço virtual nos vários campos e determine o número de níveis para tradução de um endereço virtual em endereço físico.

Página de 4kB
Cada endereço ≡ 1 Byte
$\frac{4kB}{1B} = 4k$ endereços $= 2^{12}$ → 12 bits de offset

Endereço virtual de 48 bits → nº página virtual ≡ 48-12=36 → 36 bits para o nº de página virtual

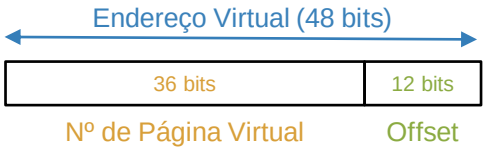


Quantos níveis?

TÉCNICO LISBOA

Tabela de páginas hierarquica

Páginas de 4kB
End. Físico: 40 bits
End. Virtual: 48 bits



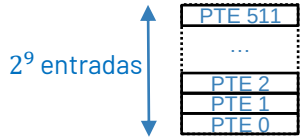
Considere um sistema de memória virtual com páginas de 4KB, 40 bits de endereço físico e 48 bits de endereço virtual.

1) Decomponha o endereço virtual nos vários campos e determine o número de níveis para tradução de um endereço virtual em endereço físico.

Endereço Físico ≡ 40 bits → Assumo que cada PTE ocupa 64 bits = 8 Bytes

Então... como cada tabela parcial de tradução ocupa uma página, cada página/tabela parcial terá:

$\frac{4kB}{8B}$ entradas $= \frac{2^{12}}{2^3} = 2^9$ → cada página/tabela parcial terá 2^9 entradas (PTE)



$$\#\text{níveis} = \frac{\#\text{bits(num.pág. virtual)}}{9} = \left\lceil \frac{36}{9} \right\rceil = 4 \text{ níveis}$$

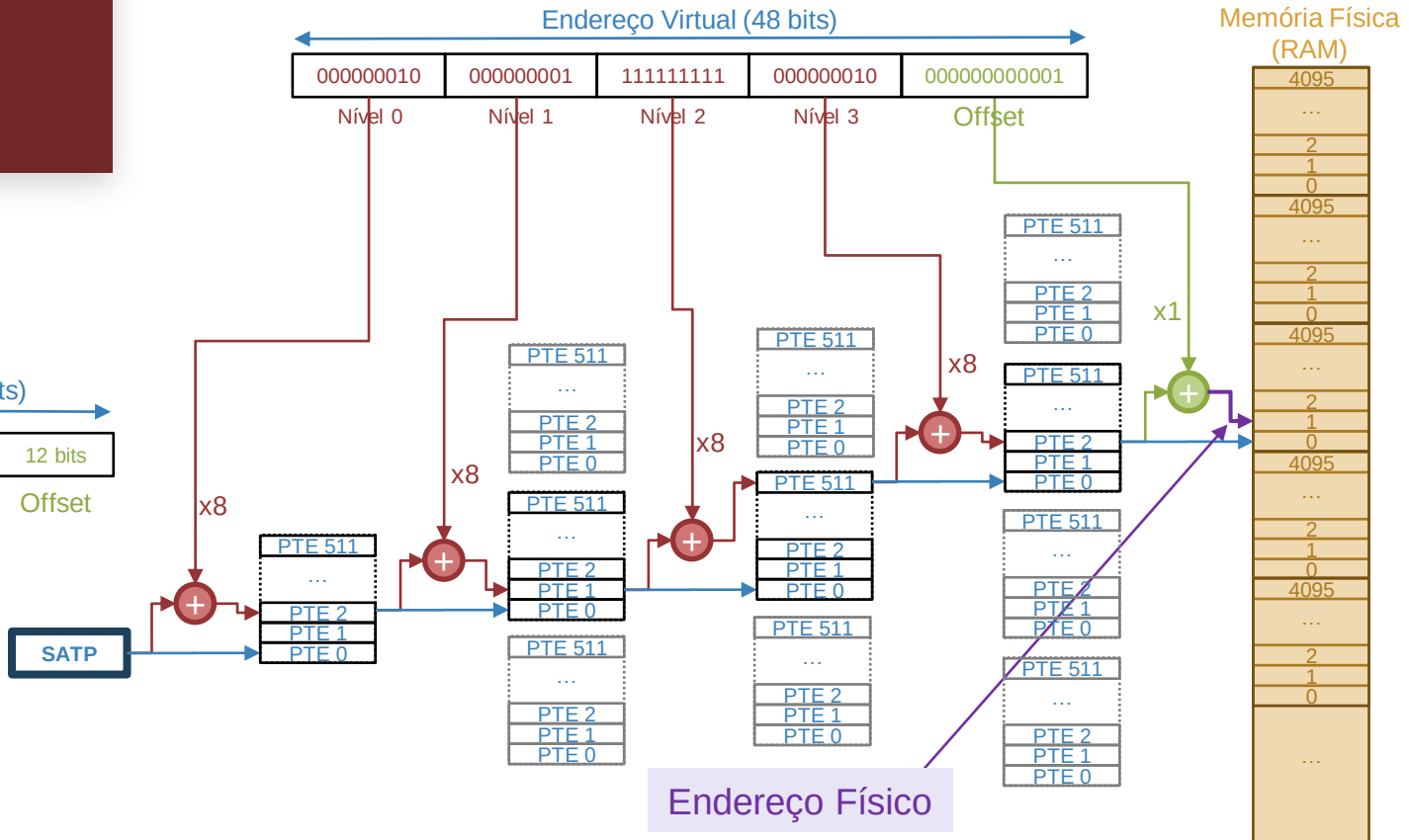
Arredondamento para cima

TÉCNICO LISBOA

Tabela de páginas hierarquica

Páginas de 4kB
End. Físico: 40 bits
End. Virtual: 48 bits

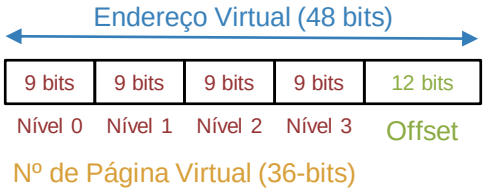
2) Esboce o esquema que permite a tradução do endereço virtual 0100 7FE0 2001h em endereço físico.



TÉCNICO LISBOA

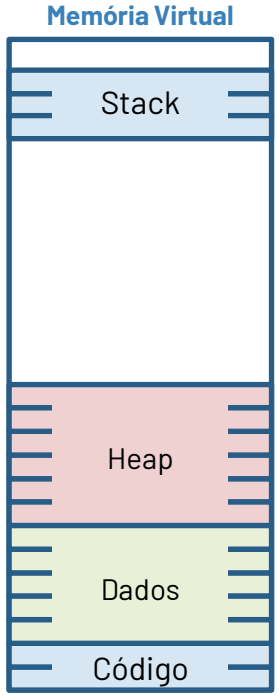
Tabela de páginas hierarquica

Páginas de 4kB
End. Físico: 40 bits
End. Virtual: 48 bits



3) Determine o espaço mínimo requirido pelo sistema de tradução (tabela de páginas hierárquica) admitindo que o programa se divide em: Código: 13KB; Dados: 1,5MB; Heap: 27MB; Stack: 14KB

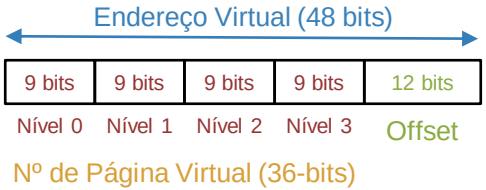
NOTA PRÉVIA: Como se tratam de segmentos distintos, admitimos que: (1) não partilham páginas físicas; mas (2) estão mapeados em zonas contíguas no espaço de endereçamento, excepto (3) a stack que ocupa uma posição superior no espaço de endereçamento virtual.



TÉCNICO LISBOA

Tabela de páginas hierarquica

Páginas de 4kB
End. Físico: 40 bits
End. Virtual: 48 bits



3) Determine o espaço mínimo requirido pelo sistema de tradução (tabela de páginas hierárquica) admitindo que o programa se divide em: Código: 13KB; Dados: 1,5MB; Heap: 27MB; Stack: 14KB

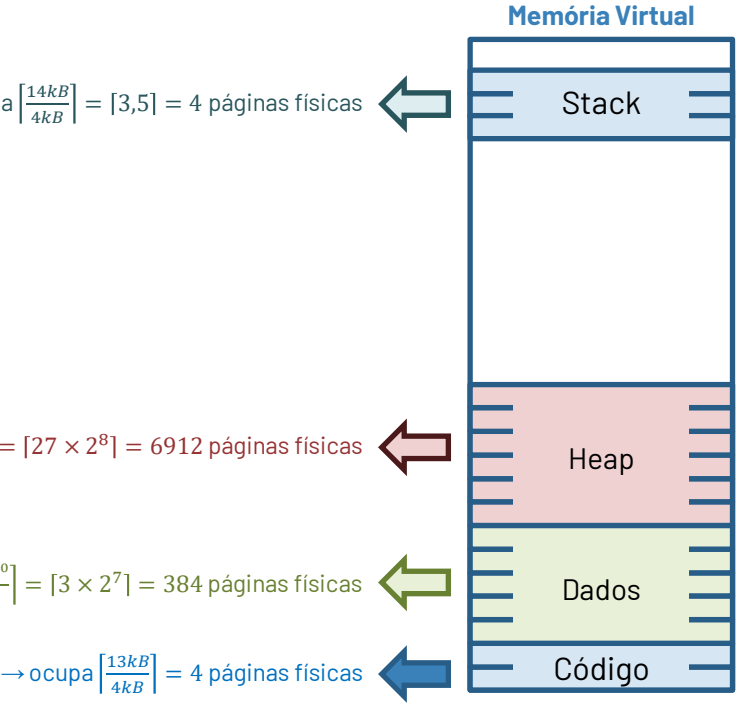
NOTA PRÉVIA: Como se tratam de segmentos distintos, admitimos que: (1) não partilham páginas físicas; mas (2) estão mapeados em zonas contíguas no espaço de endereçamento, excepto (3) a stack que ocupa uma posição superior no espaço de endereçamento virtual.

Stack: 14kB → ocupa $\left\lceil \frac{14kB}{4kB} \right\rceil = \lceil 3{,}5 \rceil = 4$ páginas físicas

Heap: 27MB → ocupa $\left\lceil \frac{27 \times 1024kB}{4kB} \right\rceil = \lceil 27 \times 2^8 \rceil = 6912$ páginas físicas

Dados: 1,5MB → ocupa $\left\lceil \frac{1.5 \times 1024kB}{4kB} \right\rceil = \left\lceil \frac{1.5 \times 2^{10}}{2^2} \right\rceil = \left\lceil \frac{3 \times 2^{10}}{2^3} \right\rceil = \lceil 3 \times 2^7 \rceil = 384$ páginas físicas

Código: 13KB → ocupa $\left\lceil \frac{13kB}{4kB} \right\rceil = 4$ páginas físicas



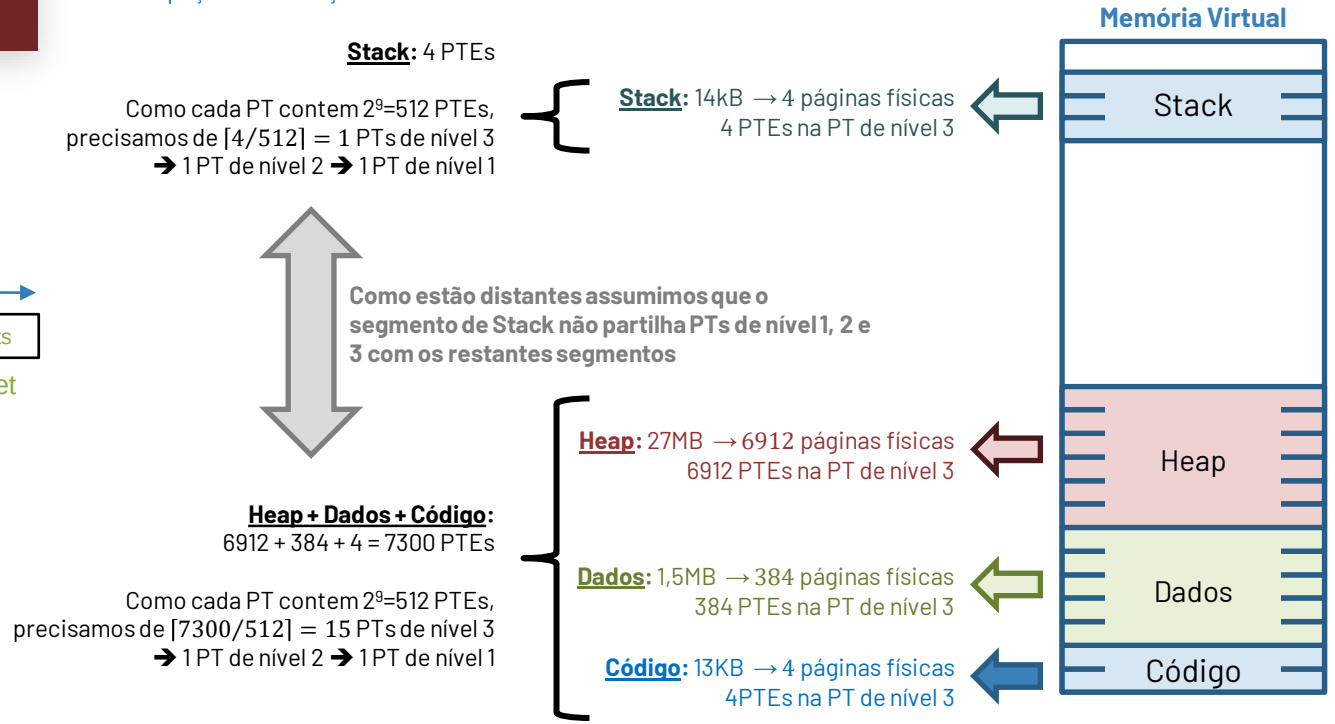
TÉCNICO LISBOA

Tabela de páginas hierarquica

Páginas de 4kB
End. Físico: 40 bits
End. Virtual: 48 bits

3) Determine o espaço mínimo requirido pelo sistema de tradução (tabela de páginas hierárquica) admitindo que o programa se divide em: Código: 13KB; Dados: 1,5MB; Heap: 27MB; Stack: 14KB

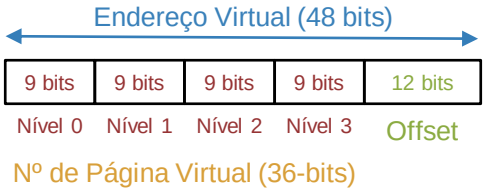
NOTA PRÉVIA: Como se tratam de segmentos distintos, admitimos que: (1) não partilham páginas físicas; mas (2) estão mapeados em zonas contíguas no espaço de endereçamento, excepto (3) a stack que ocupa uma posição superior no espaço de endereçamento virtual.



TÉCNICO LISBOA

Tabela de páginas hierarquica

Páginas de 4kB
End. Físico: 40 bits
End. Virtual: 48 bits



3) Determine o espaço mínimo requirido pelo sistema de tradução (tabela de páginas hierárquica) admitindo que o programa se divide em: Código: 13KB; Dados: 1,5MB; Heap: 27MB; Stack: 14KB

NOTA PRÉVIA: Como se tratam de segmentos distintos, admitimos que: (1) não partilham páginas físicas; mas (2) estão mapeados em zonas contíguas no espaço de endereçamento, excepto (3) a stack que ocupa uma posição superior no espaço de endereçamento virtual.

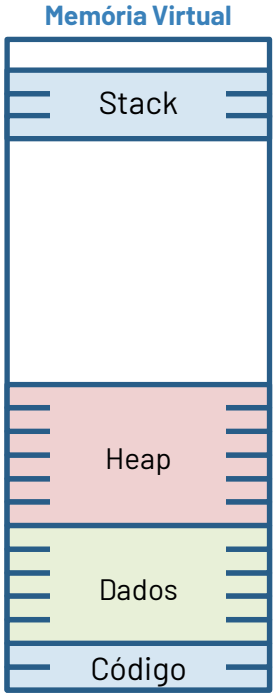
Conclusão:

Para executar este programa, o sistema de tradução ocupa
1+2+2+16 = 21 páginas de 4KB = 84KB

A memória utilizada corresponde a 7304 páginas (28,53125 MB)

O overhead (de memória) do sistema de tradução é de $\frac{21\times 4KB}{7304\times 4KB} = \frac{21}{7304} \approx 0.29\%$

	Nível 0	Nível 1	Nível 2	Nível 3	Físico
Stack	1	1	1	1	4
Heap		1	1	15	6912
Dados					384
Código					4
TOTAL	1	2	2	16	7304



TÉCNICO LISBOA

Tabela de páginas hierarquica

EXERCÍCIO #2

Tabela de páginas hierarquica

Exercício

Considere um sistema de memória virtual anexo a um processador de 64 bits, com páginas de 8KB, e 40 bits de endereço físico.

1) Determine o espaço máximo disponível para cada programa considerando que o sistema de memória virtual tem 3 níveis.

2) Indique quais as razões para a ocorrência de uma exceção de *Page Fault*, e explique como se processa o tratamento da exceção.

3) Explique porque motivo as seções de código (instruções) e dados não podem ficar alocadas na mesma página.

4) Explique para que servem os bits de A (accessed) e D (dirty) nas page table entries.

Tabela de páginas hierarquica

Processador 64-bits
Páginas de 8kB
End. Físico: 40 bits
3 níveis de tradução

1) Determine o espaço máximo disponível para cada programa considerando que o sistema de memória virtual tem 3 níveis.

Como o endereço físico ocupa 40-bits, vamos admitir que cada PTE ocupa 64-bits = 8Bytes (dimensão da palavra do processador)

Cada página/tabela parcial de tradução terá $\left\lceil \frac{8KB}{8} \right\rceil = 2^{10}$ = 1024 entradas (PTE)

- O **nível 0** terá 1 única tabela, com 2^{10} entradas (PTE). Cada uma destas entradas vai apontar para 1 tabela do **nível 1**. Assim, o **nível 0** aponta para 2^{10} tabelas do **nível 1**.

- O **nível 1** terá 2^{10} tabelas, cada uma com 2^{10} entradas (PTE). Cada uma destas entradas vai apontar para 1 tabela do **nível 2**. Assim, o **nível 1** aponta para $2^{10} \times 2^{10}=2^{20}$ tabelas do **nível 2**.

- O **nível 2** terá 2^{20} tabelas, cada uma com 2^{10} entradas (PTE). Cada uma destas entradas vai apontar para 1 **página física**. Assim, o **nível 2** aponta para $2^{20} \times 2^{10}=2^{30}$ **páginas físicas**.

Logo, o espaço máximo disponível para cada programa será de:

$2^{30} \times 8kB = 2^{30} \times 8 \times 2^{10} = 8 \times 2^{40} = 8$ Tera Bytes

Tabela de páginas hierarquica

Processador 64-bits
Páginas de 8kB
End. Físico: 40 bits
3 níveis de tradução

2) Indique quais as razões para a ocorrência de uma exceção de *Page Fault*, e explique como se processa o tratamento da exceção.

A ocorrência de uma excepção de **Page Fault** indica que a página pretendida não se encontra na memória primária (RAM). Este facto é detectado através do bit de control na PTE: **P (Present)**.

PTE:

P	A	D	R/W	EX	U/S	Physical Address

Quando existe um page fault, cabe ao Sistema Operativo verificar a origem do Page Fault. **Se o acesso for válido**, existem duas razões potenciais para o page fault:

1. A página encontra-se em disco. Para tratar esta situação, o sistema operativo deverá mover a página em causa do disco para a RAM, antes de retomar a execução do processo. Se for necessário, isso poderá implicar a prévia libertação de espaço em RAM (ex: movendo uma página pouco usada para disco).

2. A página ainda não foi alocada (ex: as páginas originalmente alocadas para stack ou heap são insuficientes, pelo que terá de alocar mais páginas). Tal como no caso anterior, este processo poderá implicar a prévia libertação de espaço em RAM. De notar que, embora o acesso seja válido, o SO pode abortar a execução do processo porque este esgotou o limite de espaço na stack (recursividade demasiado grande) ou na heap (malloc's excessivos).

Tabela de páginas hierarquica

Processador 64-bits
Páginas de 8kB
End. Físico: 40 bits
3 níveis de tradução

2) Indique quais as razões para a ocorrência de uma exceção de *Page Fault*, e explique como se processa o tratamento da exceção.

A ocorrência de uma excepção de **Page Fault** indica que a página pretendida não se encontra na memória primária (RAM). Este facto é detectado através do bit de control na PTE: **P (Present)**.

PTE:

P	A	D	R/W	EX	U/S	Physical Address

Quando existe um page fault, cabe ao Sistema Operativo verificar a origem do Page Fault. **Se o acesso for inválido**:

> 3. O processo tentou aceder a uma posição de memória que não lhe pertence ou para o qual não tem permissões (escrita numa página read-only, leitura de uma página de instruções, tentativa de execução a partir de uma página que não contem instruções).

Neste caso, o SO termina a execução do processo, geralmente com a indicação de "segmentation fault".

Tabela de páginas hierarquica

Processador 64-bits
Páginas de 8kB
End. Físico: 40 bits
3 níveis de tradução

3) Explique porque motivo as seções de código (instruções) e dados não podem ficar alocadas na mesma página.

Porque só as páginas que contêm instruções têm o bit de controlo **EX (execute)** activo.

As páginas de dados não deverão ter este bit activo e poderão ter acesso de Read/Write (**R/W**), ao passo que as páginas de instruções terão permissões de Read-Only.

PTE:

P	A	D	R/W	EX	U/S	Physical Address

Para além disso, os padrões de acesso no espaço de endereçamento do código são, em geral, muito mais regulares. Daí que não seja conveniente misturar instruções e dados, de modo a tirar melhor partido da localidade espacial (e temporal).

Tabela de páginas hierarquica

Processador 64-bits
Páginas de 8kB
End. Físico: 40 bits
3 níveis de tradução

4) Explique para que servem os bits de A (accessed) e D (dirty) nas page table entries.

PTE:

P	A	D	R/W	EX	U/S	Physical Address

Os bits **A (accessed)** e **D (dirty)** servem para permitir a implementação do mecanismo de substituição de páginas entre memória primária (RAM) e memória secundária (disco).

Quando o bit **A (accessed) =0** significa que a página não foi usada recentemente. Por conseguinte, é uma boa candidata a ser transferia para o disco, quando for necessário libertar espaço em RAM. Se A=1, significa que a página foi acedida há pouco tempo, pelo que é prudente mantê-la em RAM.

Quando o bit **D (dirty) =1** significa que a página foi modificada (na RAM) desde essa mesma página foi copiada do disco para a RAM. Significa isso que, caso seja necessário voltar a passar essa página para o disco, a informação em disco terá de ser actualizada com os dados que entretanto foram escritos enquanto essa página esteve em RAM.

Tabela de páginas hierarquica

EXERCÍCIO #3

Tabela de páginas hierarquica

Exercício

Considere um processador a executar com um relógio de 400 MHz. Assuma que este processador inclui uma MMU para traduzir os endereços do espaço de endereçamento virtual, constituído por 2^{32} endereços, nos endereços do espaço de endereçamento físico, também com 2^{32} endereços, utilizando uma tabela de tradução hierárquica. Cada página ocupa 8 kBytes e cada PTE ocupa uma palavra de 32-bits.

Por opção do fabricante, este modelo do processador adopta um endereçamento da memória com a granularidade da palavra de dados (i.e., cada endereço corresponde a uma palavra de 32-bits).

Considere também que este processador incorpora uma cache de mapeamento directo com 2 palavras em cada linha e com um tempo de acesso (em hit) correspondente a um período de relógio e um hit-rate de 80%.

O tempo de acesso à memória primária é de 33 ns.

a) Represente a estrutura de tradução do endereço virtual.

b) Determine a quantidade (máxima) de memória que é necessária para alojar as estruturas de tradução.

c) Calcule o tempo médio de acesso, considerando um sistema sem TLB.

d) Ao avaliar a execução de uma determinada aplicação numa versão mais evoluída deste sistema equipado com uma TLB (tempo de acesso igual a um período de relógio) foi observado um hit-rate (na TLB) de 95%. Calcule a aceleração média que pode ser obtida no acesso à memória com a inclusão desta TLB.

e) Indique algumas razões que justifiquem porque a taxa de sucesso na TLB é maior do que a taxa de sucesso na cache.

Tabela de páginas hierarquica

F=400MHz
Processador: 32-bits
Endereço virtual: 32-bits
Endereço físico: 32-bits
Página: 8kB

Hit-Rate(cache)=80%
t_{CACHE}=1 clock cycle
t_{RAM}=33ns
Hit-Rate(TLB)=95%
t_{TLB}=1 clock cycle

Considere um processador a executar com um relógio de 400 MHz. Assuma que este processador inclui uma MMU para traduzir os endereços do espaço de endereçamento virtual, constituído por 2^{32} endereços, nos endereços do espaço de endereçamento físico, também com 2^{32} endereços, utilizando uma tabela de tradução hierárquica. Cada página ocupa 8 kBytes e cada PTE ocupa uma palavra de 32-bits.

Por opção do fabricante, este modelo do processador adopta um endereçamento da memória com a granularidade da palavra de dados (i.e., cada endereço corresponde a uma palavra de 32-bits).

Considere peamento directo com 2 pala espondente a um período de

O tempo d

a) Represe

b) Determi a alojar as estruturas

c) Calcule LB.

d) Ao avaliar a execução de uma determinada aplicação numa versão mais evoluída deste sistema equipado com uma TLB (tempo de acesso igual a um período de relógio) foi observado um hit-rate (na TLB) de 95%. Calcule a aceleração média que pode ser obtida no acesso à memória com a inclusão desta TLB.

e) Indique algumas razões que justifiquem porque a taxa de sucesso na TLB é maior do que a taxa de sucesso na cache.

NOTA: Este exemplo retrata uma situação em que o endereçamento virtual não é usado para alargar o espaço de endereçamento físico (têm ambos a mesma dimensão). Contudo, as grandes vantagens em usar o endereçamento virtual mantêm-se:

- Cada aplicação "vê" o espaço de endereçamento inteiramente alocado a si (quando na prática o espaço de endereçamento físico está a ser partilhado pelas várias aplicações em execução);
- O sistema de endereçamento virtual confere mecanismos de segurança que impedem uma aplicação de tentar aceder a posições de memória atribuídas a outra aplicação.

Tabela de páginas hierarquica

F=400MHz
Processador: 32-bits
Endereço virtual: 32-bits
Endereço físico: 32-bits
Página: 8kB

Hit-Rate(cache)=80%
t_{CACHE}=1 clock cycle
t_{RAM}=33ns
Hit-Rate(TLB)=95%
t_{TLB}=1 clock cycle

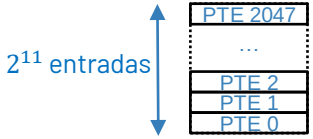
a) Represente a estrutura de tradução do endereço virtual.

Página de 8kB
Cada endereço ≡ 32-bits
$\frac{8kB}{4B} = 2k$ endereços $= 2^{11} \rightarrow$ 11 bits de offset

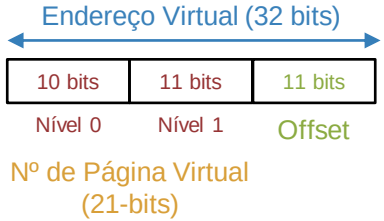
Endereço Virtual de 32 bits → nº página virtual ≡ 32-11=21 → 21 bits para o nº de página virtual

Como cada PTE é constituída por 32 bits (4 Bytes) e como cada tabela parcial de tradução ocupa uma página, cada página/tabela parcial terá:

$\frac{8kB}{4B}$ entradas $= \frac{2^{13}}{2^2} = 2^{11} \rightarrow$ cada página/tabela parcial terá 2^{11} entradas (PTE)



$$\#\text{níveis} = \frac{\#\text{bits (num.pág. virtual)}}{11} = \left\lceil \frac{21}{11} \right\rceil = 2 \text{ níveis}$$



NOTA: O número de bits utilizados para indexar cada um dos níveis de tradução não tem de ser homogéneo. Quando tal não for possível, podemos atribuir menos bits para a indexação do nível 0, fazendo com que nem todas as entradas PTE da tabela deste nível sejam utilizadas, estreitando assim a "árvore" hierárquica de tradução.

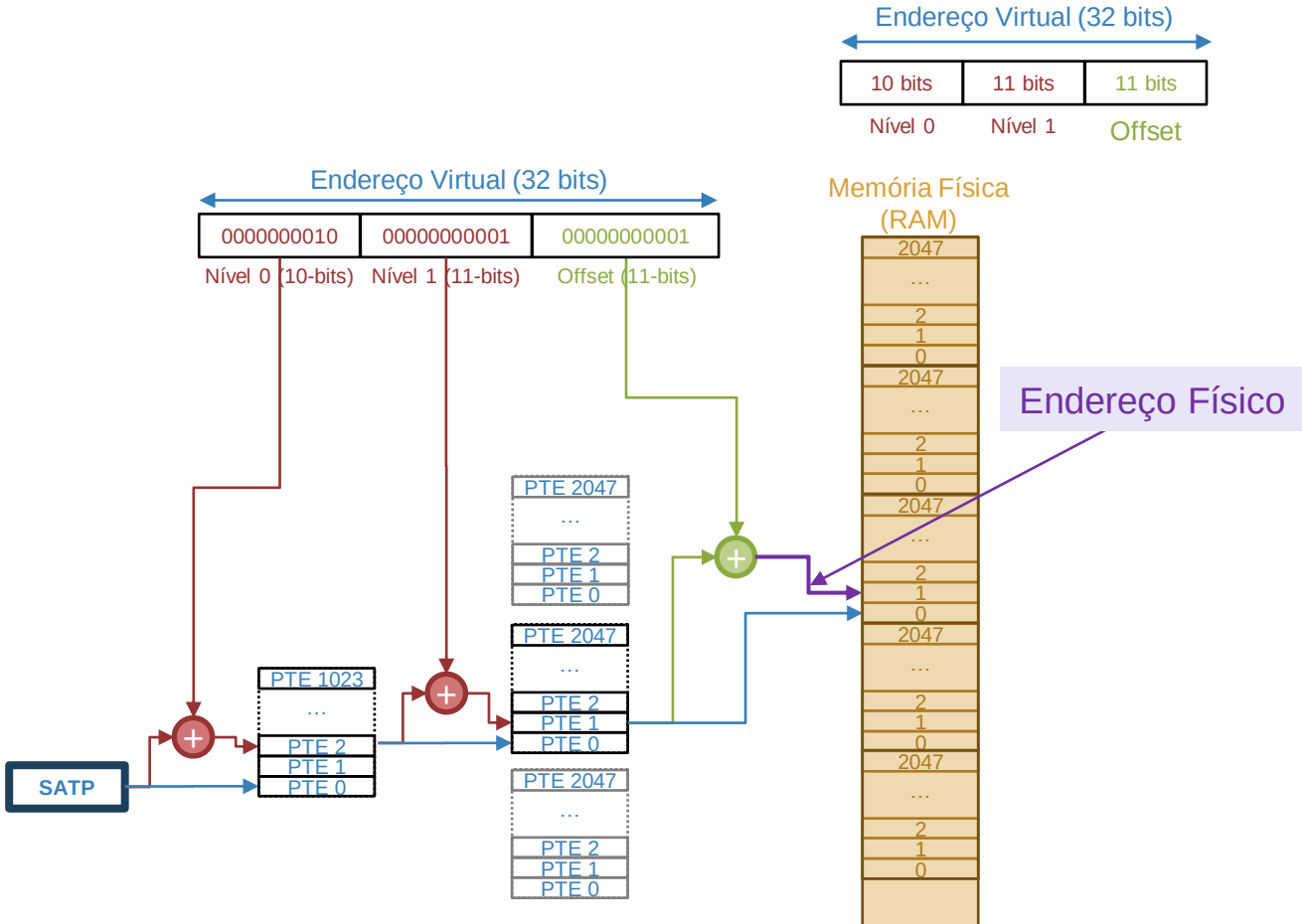
TÉCNICO LISBOA

Tabela de páginas hierarquica

F=400MHz
Processador: 32-bits
Endereço virtual: 32-bits
Endereço físico: 32-bits
Página: 8kB

Hit-Rate(cache)=80%
t_{CACHE}=1 clock cycle
$linha_{CACHE}$=2 palavras
t_{RAM}=33ns
Hit-Rate(TLB)=95%
t_{TLB}=1 clock cycle

a) Represente a estrutura de tradução do endereço virtual.



TÉCNICO LISBOA

Tabela de páginas hierarquica

F=400MHz
Processador: 32-bits
Endereço virtual: 32-bits
Endereço físico: 32-bits
Página: 8kB

Hit-Rate(cache)=80%
t_{CACHE}=1 clock cycle
$linha_{CACHE}$=2 palavras
t_{RAM}=33ns
Hit-Rate(TLB)=95%
t_{TLB}=1 clock cycle

b) Determine a quantidade (máxima) de memória que é necessária para alojar as estruturas de tradução.

Espaço ocupado = tabela Nível 0 + tabelas Nível 1

- No **nível 0**, a tabela ocupa metade de uma página, já que apenas são usados 10 bits (dos 11 possíveis) para indexer a tabela. Contudo, essa metade livre não pode ser utilizada para outro fim, pelo que vamos admitir que o nível 0 ocupa 1 página completa.

- No **nível 1** haverão 2^{10} tabelas parciais, cada uma ocupando uma página completa. Logo, este nível ocupa 2^{10} páginas.

Espaço ocupado = 1 página (Nível 0) + 2^{10} páginas (Nível 1) = 1025 páginas ≈ 8 MByte

Tabela de páginas hierarquica

F=400MHz
Processador: 32-bits
Endereço virtual: 32-bits
Endereço físico: 32-bits
Página: 8kB

Hit-Rate(cache)=80%
t_{CACHE}=1 clock cycle
$linha_{CACHE}$=2 palavras
t_{RAM}=33ns
Hit-Rate(TLB)=95%
t_{TLB}=1 clock cycle

c) Calcule o tempo médio de acesso, considerando um sistema sem TLB.

$$t_{CLK} = \frac{1}{400\ MHz} = 2{,}5ns$$

$$t_{CACHE} = 1\ t_{CLK}$$

$$t_{RAM} = \left\lceil \frac{33ns}{2{,}5ns} \right\rceil = 14\ t_{CLK}$$

$$t_{ACESSO}^{S/TLB} = t_{TRADUÇÃO}^{S/TLB} + t_{ACESSO\_DADOS}$$

$$t_{ACESSO\_DADOS} = t_{CACHE} + MR \times t_{RAM} = t_{CLK} + 0{,}2 \times 14 t_{CLK} = 3{,}8 t_{CLK}$$

$$t_{TRADUÇÃO}^{S/TLB} = t_{NÍVEL0} + t_{NÍVEL1} = \left[t_{CACHE} + MR \times t_{RAM}\right] + \left[t_{CACHE} + MR \times t_{RAM}\right]$$
$$= 2 \times \left[t_{CACHE} + MR \times t_{RAM}\right]$$

$$t_{ACESSO}^{S/TLB} = 2 \times \left[t_{CACHE} + MR \times t_{RAM}\right] + \left[t_{CACHE} + MR \times t_{RAM}\right]$$
$$= 3 \times \left[t_{CACHE} + MR \times t_{RAM}\right] = 3 \times \left[t_{CLK} + 0{,}2 \times 14 t_{CLK}\right]$$
$$= 3 \times 3{,}8 t_{CLK} = 11{,}4 t_{CLK} = 28{,}5\ ns$$

Tabela de páginas hierarquica

F=400MHz
Processador: 32-bits
Endereço virtual: 32-bits
Endereço físico: 32-bits
Página: 8kB

Hit-Rate(cache)=80%
t_{CACHE}=1 clock cycle
$linha_{CACHE}$=2 palavras
t_{RAM}=33ns
Hit-Rate(TLB)=95%
t_{TLB}=1 clock cycle

d) Ao avaliar a execução de uma determinada aplicação numa versão mais evoluída deste sistema equipado com uma TLB (tempo de acesso igual a um período de relógio) foi observado um hit-rate (na TLB) de 95%. Calcule a aceleração média que pode ser obtida no acesso à memória com a inclusão desta TLB.

$$t_{ACESSO}^{C/TLB} = t_{TRADUÇÃO}^{C/TLB} + t_{ACESSO\_DADOS}$$

$$t_{ACESSO\_DADOS} = t_{CACHE} + MR \times t_{RAM} = t_{CLK} + 0{,}2 \times 14t_{CLK} = 3{,}8t_{CLK}$$

$$t_{TRADUÇÃO}^{C/TLB} = t_{TLB} + MR_{TLB} \times t_{TRADUÇÃO}^{S/TLB} = t_{TLB} + MR_{TLB} \times 2 \times \left[t_{CACHE} + MR \times t_{RAM}\right]$$

$$\begin{aligned} t_{ACESSO}^{C/TLB} &= t_{TLB} + 0{,}05 \times 2 \times \left[t_{CACHE} + MR \times t_{RAM}\right] + \left[t_{CACHE} + MR \times t_{RAM}\right] \\ &= t_{TLB} + 1{,}1 \times \left[t_{CACHE} + MR \times t_{RAM}\right] = t_{CLK} + 1{,}1 \times \left[t_{CLK} + 0{,}2 \times 14t_{CLK}\right] \\ &= t_{CLK} + 1{,}1 \times 3{,}8t_{CLK} = 5{,}18t_{CLK} = 12{,}95\ ns \end{aligned}$$

$$Speedup = \frac{t_{ACESSO}^{S/TLB}}{t_{ACESSO}^{C/TLB}} = \frac{11{,}4t_{CLK}}{5{,}18t_{CLK}} = 2{,}2$$

Tabela de páginas hierarquica

F=400MHz
Processador: 32-bits
Endereço virtual: 32-bits
Endereço físico: 32-bits
Página: 8kB

Hit-Rate(cache)=80%
t_{CACHE}=1 clock cycle
$linha_{CACHE}$=2 palavras
t_{RAM}=33ns
Hit-Rate(TLB)=95%
t_{TLB}=1 clock cycle

e) Indique algumas razões que justifiquem porque a taxa de sucesso na TLB é maior do que a taxa de sucesso na cache.

- Cada entrada na TLB corresponde a uma página de código/dados com 8k Bytes.
- Este espaço de memória (8k Bytes) é substancialmente superior ao espaço guardado numa linha da cache (i.e., 2 palavras por linha=8 Bytes).
- Por conseguinte, é expectável que se tire muito mais partido da localidade espacial na na página do que numa linha de cache, pelo que o Hit-Rate na TLB será, em geral, muito maior do que o Hit-Rate na cache.
- Para além disso, a TLB usa, em geral, várias vias de associatividade (ou são mesmo completamente associativas), o que mitiga o número de colisões, preservando mais tempo as traduções na memória da TLB.