Que abstrações são providas por um SO moderno?

- Processos — programa em execução, estado
- memória 'infinita' — $|\,pointers\,| \geq 32$ bits
- multiprogramação — ≥ 1 processos/usuários no sistema
- sistema de arquivos unificado — tudo são sequências de bytes
- recursos compartilhados — competição e cooperação no uso
  - ▷ processador
  - ▷ memória principal (DRAM)
  - ▷ memória secundária, discos
  - ▷ interf. de rede, periféricos
- segurança — um processo não perturba outros

A mãe dos sistemas operacionais modernos é o Unix, escrito em 2-3 semanas em assembly do PDP-8 por Ken Thompson, em 1969.

Processos

Processo é um programa em execução

O estado de execução de um processo é mantido
(i) nos registradores (PC, r1-r31, HI,LO),
(ii) nas variáveis em memória, e
(iii) nos arquivos abertos e dispositivos alocados ao processo.

Um processo pode estar
executando no processador
suspenso enquanto espera por um evento externo (E/S)
pronto para executar, enquanto espera sua vez de usar o processador

Várias cópias de um mesmo programa podem executar simultaneamente (`bash`, `ls`)

Multiprogramação

O processador é multiplexado no tempo entre os processos ativos

Cada processo executa durante um *quantum* de tempo; trocas rápidas ($\geq$100x/s) dão a impressão de que todos os processos executam em paralelo

Cada processo executa durante seu *quantum* e então é removido do processador e inserido na fila de processos prontos para executar
ou
um processo devolve o processador ao SO quando necessita uma operação de E/S demorada

Numa troca de contexto, o estado do processo que deixa o processador é salvado em memória, e o estado do processo na cabeça da fila de prontos é carregado no processador

Segurança

Um processo não pode ler nem escrever no espaço de endereçamento de outro processo

Um processo não pode ler, escrever, ou alterar o estado de um periférico alocado a outro processo

Espaço de endereçamento é o conjunto de endereços alocados a um processo para a sua execução e consiste de:
– segmento de texto (código)
– segmento de dados (variáveis inicializadas, BSS, *heap*)
– pilha
– área do SO protegida para armazenar estado do processo

Uma parte dos mecanismos de segurança é implementada sobre o espaço de endereçamento

Memória 'infinita'

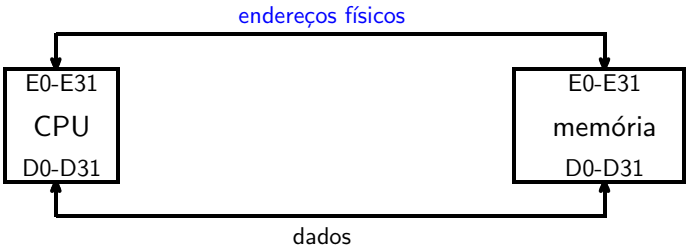
- Uma máquina multiprogramada suporta a execução concorrente de muitos processos;
- potencialmente, cada um deles usa todo o seu espaço de endereçamento 4 Gbytes em processadores com *pointers* de 32 bits
- memória física da máquina (DRAM) é, em geral, menor que a demanda agregada de todos os processos
- como gerenciar a memória física para que todos os processos terminem, ou ao menos, que haja *forward progress*?
- memória física também deve ser multiplexada no tempo

Estes sistemas foram desenvolvidos quando a memória física disponível era da ordem de uns poucos KILO-bytes

Espaço de Endereçamento – três modelos

- Primeiro Modelo (até ≈1960)
  - Programador supõe que espaço de endereçamento é contínuo espaço se extende de `0x0000.0000` a `0xffff.ffff`
  - programa deve ser carregado sempre no endereço físico inicial
  - memória pode conter somente um programa em execução
- Segundo Modelo (até 1972 - IBM 370, ideia de 1962 [Kilburn, 1962])
  - EdE contínuo, > que memória física → overlays
  - Registrador de tradução para fazer relocação do programa → programa pode ser carregado em qualquer endereço físico
  - mais de um programa carregado em memória para execução
- Terceiro Modelo (após 1972 - IBM 370)
  - EdE > que memória física, tradução de ender transparente
  - muitos programas carregados na memória

Limitações do Endereçamento Físico

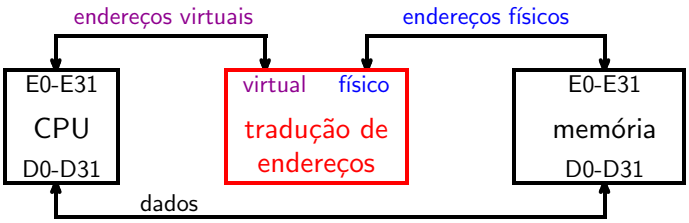


Todos programas compartilham um espaço de endereçamento físico

Programas em linguagem de máquina "conhecem"
organização do computador

Não há maneira de evitar que um programa acesse
qualquer recurso da máquina

Nível de indireção para obter proteção



Programas de usuário executam em espaço de endereçamento virtual

Unidade de gerenciamento de memória (MMU) traduz endereços
é gerenciada pelo Sistema Operacional (SO),
mapeia endereços virtuais em endereços físicos em memória

MMU suporta as funcionalidades "modernas" de SOs: [Kilburn, 1962]
proteção, tradução (de endereços), compartilhamento

Memória Virtual

Processador "enxerga" espaço de endereçamento de 4 Gbytes
$\rightleftharpoons$ pointers de 32 bits como definidos na linguagem **C**

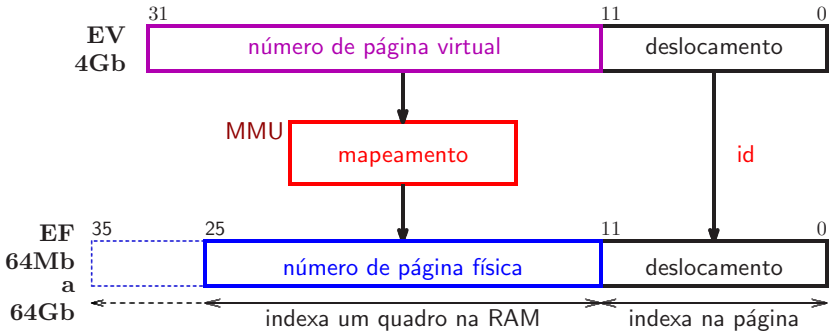
Memória física (RAM) tem de 256Mbytes a 16Gbytes
endereços físicos com 28 a 34 bits

MMU + sistema de memória virtual mapeiam
endereços virtuais (4 Gbytes)
em endereços físicos (256M-16Gbytes)

Tradução de endereço virtual $\rightarrow$ endereço físico
tem como efeito principal a separação de espaços de endereçamento
e como efeito colateral principal a proteção entre EdEs $\neq$s

Paginação (i)

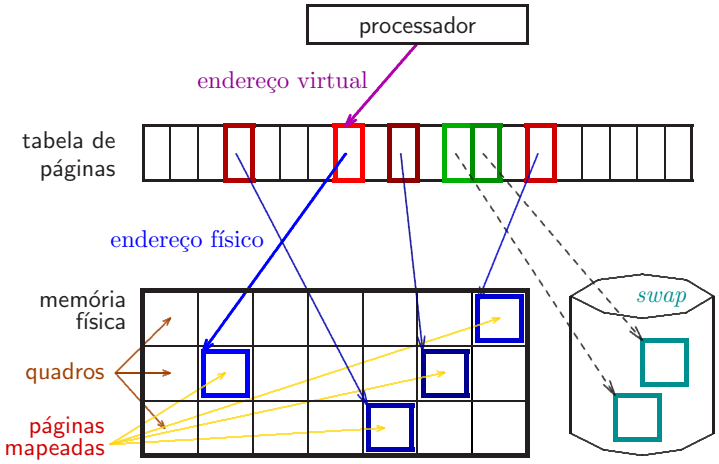
Espaço de endereçamento dividido em páginas de 4K ou 8Kbytes
Tabela de Páginas mantém mapeamento entre
endereços virtuais e endereços físicos



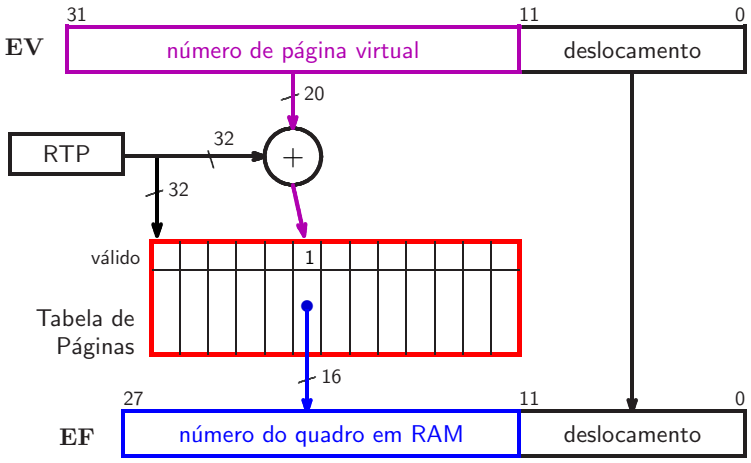
nestes slides, mostraremos somente páginas de 4Kbytes

Paginação (ii)

Tabela de Páginas faz mapeamento associativo entre
páginas virtuais em quadros (*frames*) de memória física

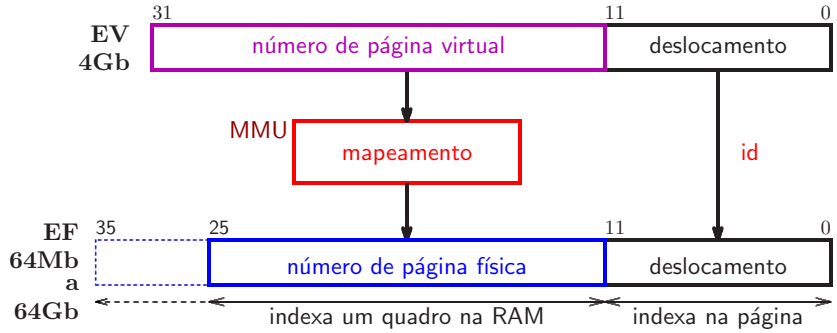


Paginação (iii)



Registrador de Tabela de Páginas (RTP) aponta
início da Tabela de Páginas do processo corrente

Paginação (iv)



Qual o tamanho da tabela de páginas?

Quantas tabelas de páginas são necessárias?

Hierarquia de Memória

Tamanho dos componentes da hierarquia de memória

nível	capacidade [bytes]	$\mathcal{T}_{\text{acesso}}$ [ciclos]
cache primária L1	16–64 K	1-2
cache secundária L2	256–4096 K	10-20
memória DRAM	1-16 G	100-300
disco	$\gg$ 4 G	$\gg$100.000

Memória física é usada como cache para o disco

Custo de uma falta de página condiciona projeto da hierarquia:
- páginas grandes para amortizar penalidade (custo da carga)
- reduzir taxa de faltas é importante $\rightarrow$ mapeamento associativo
- tratamento de faltas em software $\rightarrow$ algoritmos melhores
- escrita preguiçosa

DRAM como Cache de Disco

Se página virtual não está em memória física,
$\rightarrow$ deve ser copiada do disco
$\rightarrow$ alguma página deve ser ejetada para abrir espaço
princípio da localidade recomenda:
vítima deve ser "usada no passado mais distante" (LRU)

Exemplo: 11, 10, 12, 9, 11, 7, 11, 13 LRU c.r.a 13? 11?

Escrita é preguiçosa:
páginas somente de leitura (código) são substituídas
páginas com atualizações (escritas) são marcadas `modificadas` e,
antes de sobre-escritas/substituídas, o disco deve ser atualizado

Tabela de Páginas Linear

Registrador de Tabela de Páginas (RTP) aponta para
início da Tabela de Páginas do processo

Cada elemento da tabela de páginas contém:

nPF número da página física residente em memória
nPD núm da página em disco (se página foi movida para disco)
status bits de status e proteção/uso: Used, Modif, RO, RW, EX
D/M/U página em ***D***isco / mapeada em ***M***emória / ***U***nmapped

31 7	6 2	1 0
nPF/nPD	**U M RO WR EX**	**D/M/U**

Detalhes em ci215, muitos detalhes mais...

Tabela de Páginas Linear – elemento da TP

31 7	6 2	1 0
nPF/nPD	**U M RO WR EX**	**D/M/U**

```
typedef struct etp {
   int v :  2,      // 11 - em memória,
                    // 00 - unmapped, 01 - em disco
   int EX : 1,      // executável
   int WR : 1,      // writable
   int RO : 1,      // read-only
   int U : 1,       // usado
   int M : 1,       // modificado
   int nPF : 25;    // número da pág física + padding
} etpType;
```


Tamanho da Tabela de Páginas Linear

Com endereços de 32 bits, páginas de 4 Kbytes, 4 bytes/elemento:
→ 2^{20} elementos ⇝ | TP | = 4 Mbytes por processo
→ 4 Gbytes de swap para conter espaço de endereçam. completo

Páginas maiores!
⇝ mais fragmentação interna última página meio vazia
⇝ penalidade por falta maior mais tempo para ler do disco

Processadores de 64 bits
→ mesmo páginas de 1 Mbyte implicam em tabelas com
2^{44} elementos de 8 bytes (≈ 35 TBytes)

Há salvação? Como?

Tabela de Páginas Hierárquica

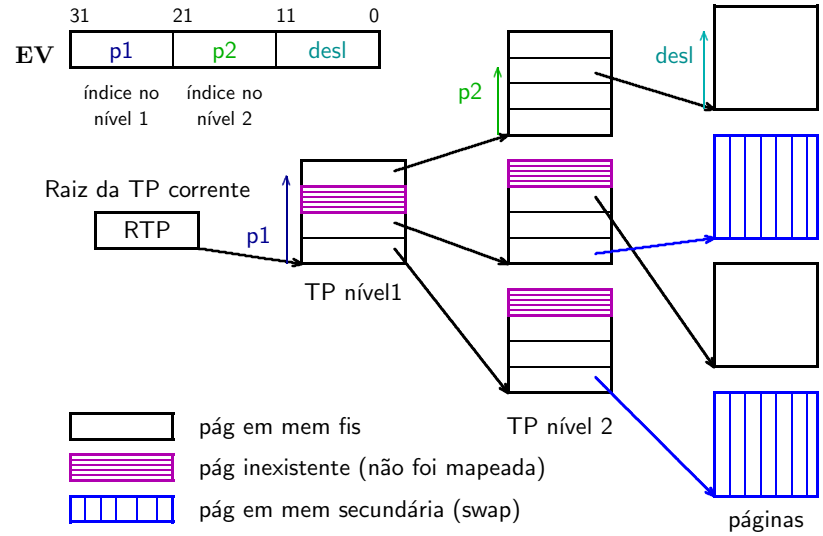
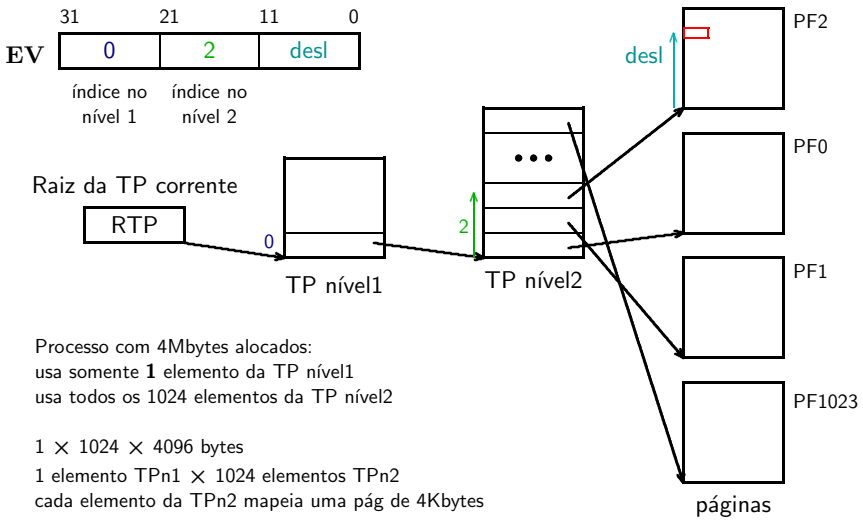


Tabela de Páginas Hierárquica – exemplo



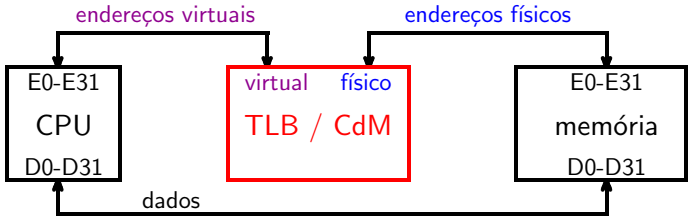
Processo com 4Mbytes alocados:
usa somente **1** elemento da TP nível1
usa todos os 1024 elementos da TP nível2

1 × 1024 × 4096 bytes
1 elemento TPn1 × 1024 elementos TPn2
cada elemento da TPn2 mapeia uma pág de 4Kbytes

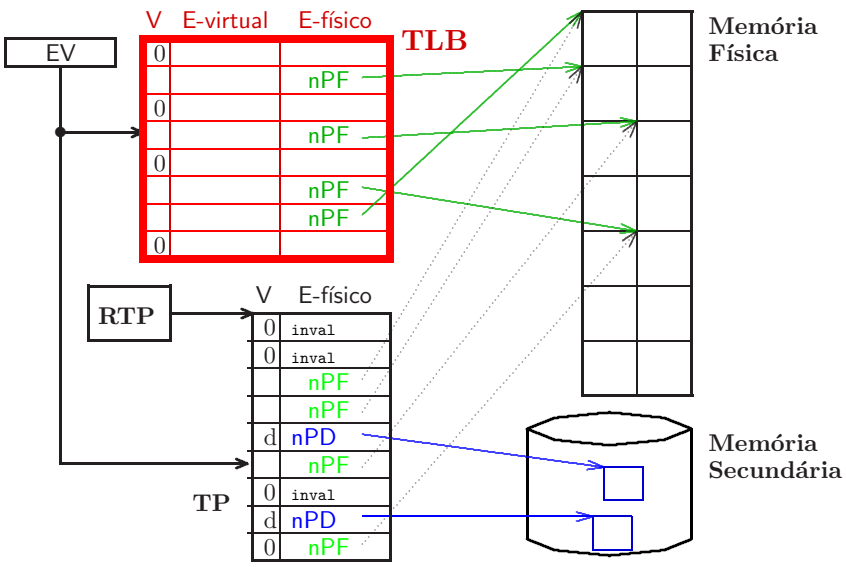
Cache de Mapeamentos (i)

A cada referência, processador consulta TP
para descobrir endereço físico da palavra
→ faz uma referência à Tabela de Páginas para obter endereço
e então faz referência à palavra...
→ para cada referência, DOIS acessos à memória...

Solução:
Cache de Mapeamentos (CdM) ou Translation Lookaside Buffer (TLB)



Cache de Mapeamentos (ii)



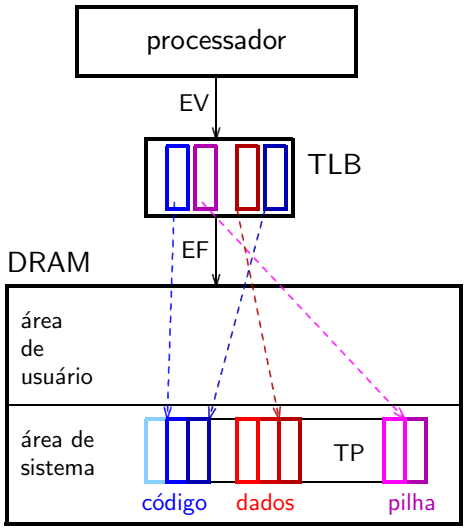
Cache de Mapeamentos (iii)

Processador procura mapeamento na TLB
se encontra, completa referência;
senão, busca mapeamento da Tabela de Páginas (em mem física)
e guarda na TLB para uso futuro;

Parâmetros de projeto:

- tamanho de bloco: 1 ou 2 mapeamentos
- tempo de acerto: 1/2 ciclo
- penalidade por falta: acesso à DRAM (100-300 ciclos)
- taxa de faltas: 0.01% a 1%
- tamanho: 32 a 128 blocos
- assocatividade: alta (4,8,total)

Cache de Mapeamentos (iv)



A TLB contém cópias dos mapeamentos de EV→EF na Tabela de Páginas.

A TP fica em DRAM, na área reservada ao SO.

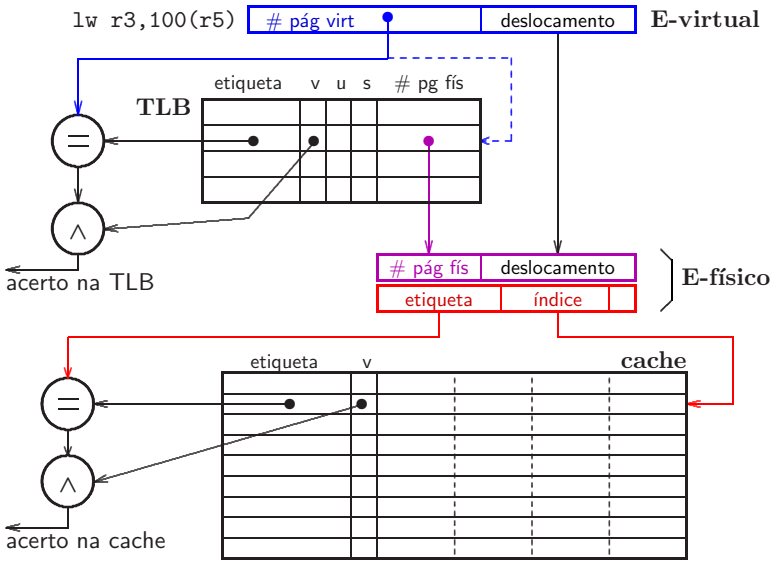
Campos da Cache de Mapeamentos

53 33	32 8	7 3	2 1	0
etiqueta	**nPF**	**U M RO WR EX**	**M/U**	**V**

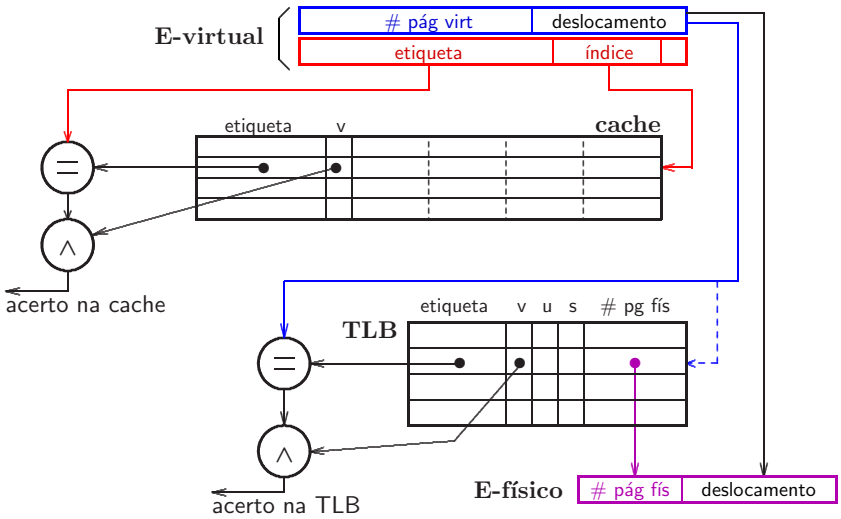
```
typedef struct etlb {
   int v :  1,       // 1 - em memória, 0 - inválido
   int EX : 1,       // executável
   int WR : 1,       // writable
   int RO : 1,       // read-only
   int U : 1,        // usado
   int M : 1,        // modificado
   int nPF : 25;     // número da pág física + padding
} etlbType;
```

Se página virtual não está em memória,
então não pode haver um mapeamento da página na TLB

TLB & Cache – cache indexada com E-físico



TLB & Cache – cache indexada com E-virtual



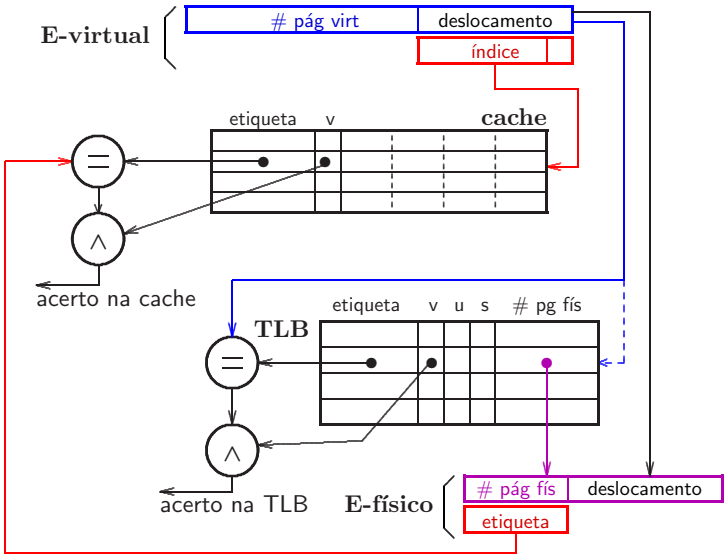
A TLB é acessada somente nas faltas na L1

TLB & Cache – indexação da cache

- **cache indexada com endereço físico**
  - endereço é traduzido (EV⇝ EF) e então cache é indexada
  - TLB no caminho crítico: 1 ciclo para TLB⟶ cache
  - troca de contexto: etiquetas do processo que entra são sempre distintas das do processo que sai: etiquetas são do EF

- **cache indexada com endereço virtual**
  - endereço somente é traduzido (EV⇝ EF) nas faltas na cache
  - TLB fora do caminho crítico
  - troca de contexto: etiquetas do processo que entra podem ser as mesmas que as do processo que sai: etiquetas são do EV
    espaços de endereçamento virtuais não são disjuntos, físicos são
  - cache deve ser expurgada numa troca de contexto
    $\Longrightarrow$ todos os blocos devem ser invalidados

TLB & Cache – índice virtual, etiqueta física



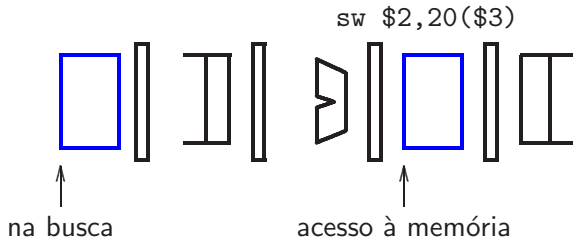
TLB & Cache – índice virtual, etiqueta física (cont.)

- **cache indexada com endereço virtual**
  - índice não pode ser traduzido $\Longrightarrow$ | índice | $\leq$ | deslocamento |
  - TLB fora do caminho crítico

- **etiquetas são endereço físico**
  - tradução (EV⇝ EF) em paralelo com indexação da cache
  - endereço é traduzido (EV⇝ EF) e etiquetas são comparadas

- **trocas de contexto**
  - etiquetas do processo que entra são sempre distintas das do processo que sai: etiquetas são do EF
  - espaços de endereçamento físicos são disjuntos
  - cache não precisa ser expurgada numa troca de contexto
    etiquetas separam espaços de endereçamento disjuntos

TLB & Cache (i)



Faltas & Excessões

Causas: endereço ilegal fora do espaço de endereçamento
ou desalinhado end%4 $\neq 0$
ou falta na TLB
ou falta na tabela de páginas

TLB & Cache (ii)

Faltas na TLB

- na busca de instrução
  → processador esvazia "pipeline"
  re-carrega TLB
  e busca instrução novamente
  custo: número de segmentos + carga da TLB
- referência a dados (`lw sw`)
  → instruções à frente completam
  instrução não pode alterar estado `lw $5, 0($5)`
  processador re-carrega TLB
  e busca instrução novamente
  custo: 1-2 segmentos + carga da TLB + 4-3 segmentos
- por que a instrução causadora é buscada novamente?

Reposição de Páginas

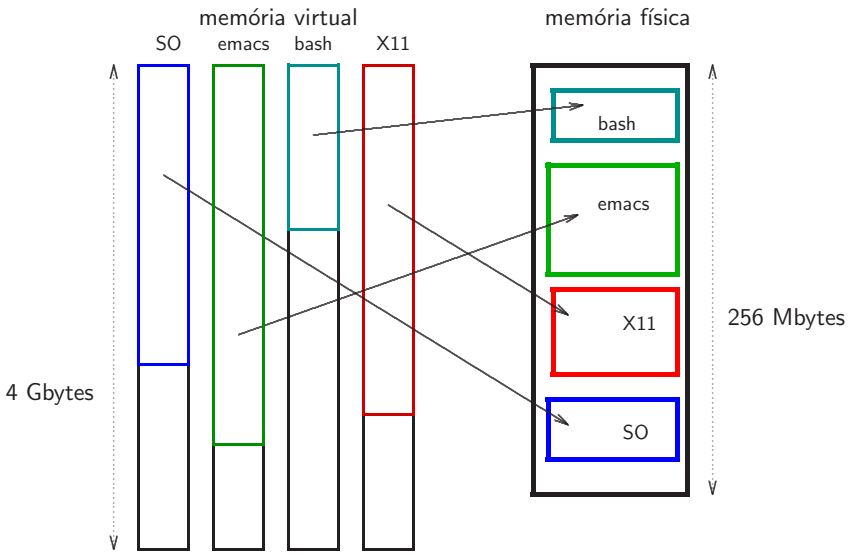
Falta na Tabela de Páginas:

Se página virtual não está em memória *page fault*
SO assume controle e requisita cópia ao controlador de disco
SO escolhe vítima para ser substituída pela nova página;
→ se vítima foi modificada, SO deve atualizar disco

Localidade implica em vítima ser
página usada no passado mais distante (LRU)

Tempo de carga é da ordem de 10^5 a 10^6 ciclos...

Processos e Proteção



Processos e Proteção (i)

Processo é um programa em execução

Estado de um processo:

- conteúdo dos registradores do processador + PC, RTP, status
- conteúdo da TLB
- conteúdo da Tabela de Páginas
- conteúdo da memória (variáveis e pilha)
- estado dos periféricos e arquivos abertos

Processos e Proteção (ii)

A cada processo corresponde um **espaço de endereçamento**

A cada espaço de endereçamento corresponde um
domínio de proteção inclui registradores, arquivos abertos, etc

Proteção:
um processo não pode interferir no
espaço de endereçamento de outro processo

Processador detecta quebra de proteção ao acessar TLB
⇝ excessão de violação de proteção
⇝ como detecta a violação?

Processos e Proteção (iii)

Mecanismos para implementar proteção:

- dois modos de operação: supervisor e usuário
  $\rightsquigarrow$ instruções reservadas ao modo supervisor
- parte do estado do processador não pode ser alterada em modo usuário
- instruções especiais permitem mudança no estado
  $\rightsquigarrow$ chamadas de sistema (*syscall*s)
- `syscall` $\rightarrow$ *trata evento* $\rightarrow$ `eret` $\rightarrow$ *segue execução*

Aluno de primeiro ano não pode formatar os discos...

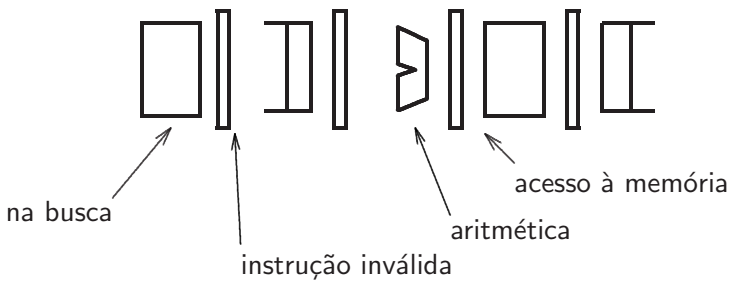
Processos e Proteção (iv)

Tratamento de exceções:

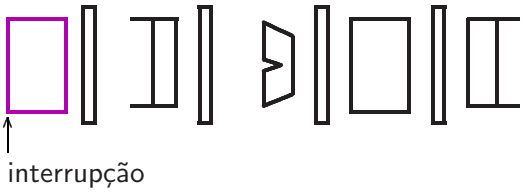
- muda para modo supervisor e desabilita interrupções
- salva EPC (Exception PC) preenchido pelo proc ao detectar evento
- salva registrador de status (Cause Register)
- salva registradores r1-r31 HI,LO
- habilita interrupções
- trata excessão $\rightsquigarrow$ tratamento depende do tipo de evento
- SO faz troca de contexto e escolhe outro processo para executar
- recompõe estado do novo processo
- carrega PC com instrução após última interrupção do novo processo
- e volta a executar em modo usuário

Excessões em processador segmentado

Excessões ocorrem durante a execução ('no meio') das instruções



Interrupções em processador segmentado



Interrupções sinalizam eventos externos ao processador

Atendimento implica em processamento de código para tratar evento

Vetor de Interrupções (e de excessões)
contém endereços das funções associadas aos eventos

Interrupções são detectadas entre duas instruções

Tratamento de Interrupções (i)

Vetor de Interrupções/Exceções (hipotético)

ender. físico	ender. tratador	evento
0x0000	*hardReset()	reset a frio
0x0004	*softReset()	reset a quente
0x0008	*TLBinstr()	falta na TLB de código
0x000c	*TLBdado()	falta na TLB de dados
0x0010	*underflow()	underflow em PF
0x0014	*overflow()	overflow em PF
0x0018	*interrDisco()	tratador interr disco
0x001c	*interrRede()	tratador interr rede
0x0020	*interrDMA()	tratador interr DMA
0x0024	*divZero()	divisão por zero
0x0028	*instrInval()	instrução inválida
0x002c	...	...

Tratamento de interrupções (ii)

- desabilita interrupções (hw)
- salva registrador de status (Cause Register)
- salva registradores
- talvez habilita interrupções (sw)
- salta para endereço do código que trata da interrupção
- se habilitou antes, desabilita interrupções
- recompõe registradores
- habilita interrupções
- retorna para instrução na qual a interrupção foi detectada