

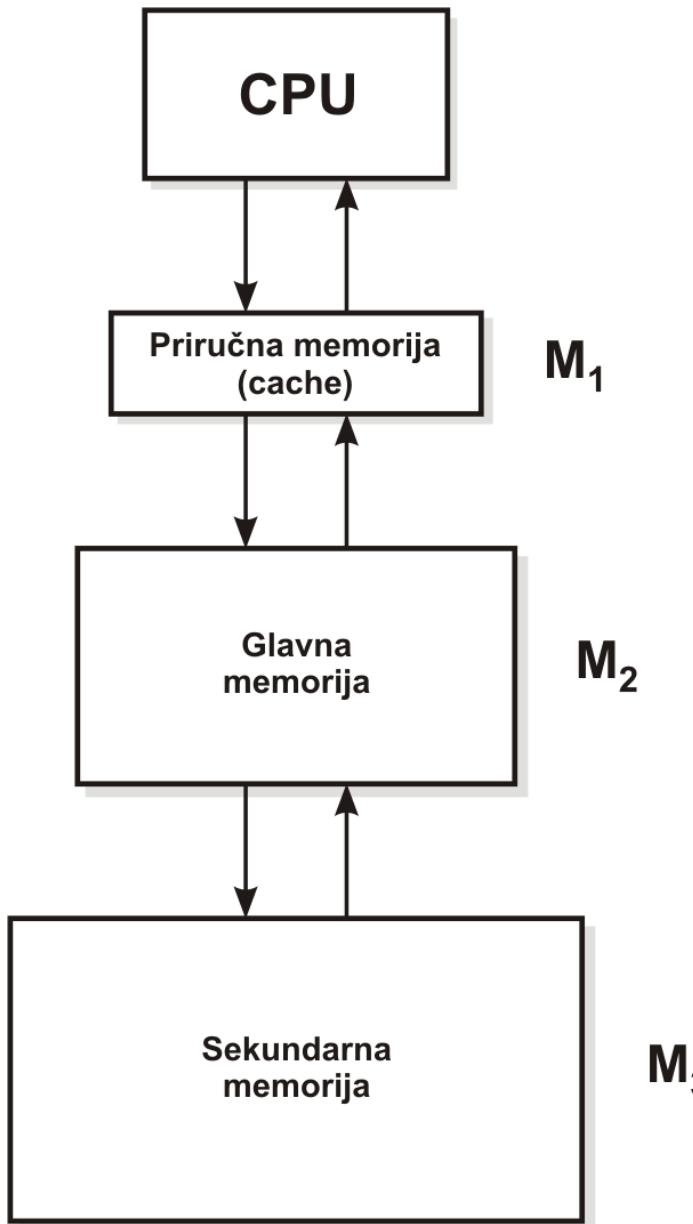
# **VIRTUALNI MEMORIJSKI SUSTAV**

- 1. Memorijска hijerarhija**
2. Fizički i logički adresni prostor
3. Straničenje
4. Višerazinsko straničenje, translacijski spremnik
5. Segmentacija
6. Detalji izvedbe

# Prisjetimo se memorejske hijerarhije

- niže razine imaju **veći kapacitet** i **veću latenciju**
- ideja: smanjiti prosječnu latenciju korištenjem lokalnih kopija “popularnih” podataka u bržoj memoriji
- želimo da, prividno, računalo ima kapacitet diska i brzinu registara

zaporni sklopovi	.05 ns ( $0.2 \times \Delta T_{CPU}$ ), 100B
registri (SRAM)	.25 ns ( $1 \times \Delta T_{CPU}$ ), 500B
cache (SRAM)	1 ns ( $4 \times \Delta T_{CPU}$ ), 64kB
RAM (DRAM)	50 ns, 1 GB
diskovi	10 ms, 1 TB



Memorijska hijerarhija:

- $(M_1, M_2, M_3, \dots M_n)$
- $M_i$  “podređena” memoriji  $M_{i-1}$
- CPU izravno komunicira s  $M_1$

Neka je zadano:

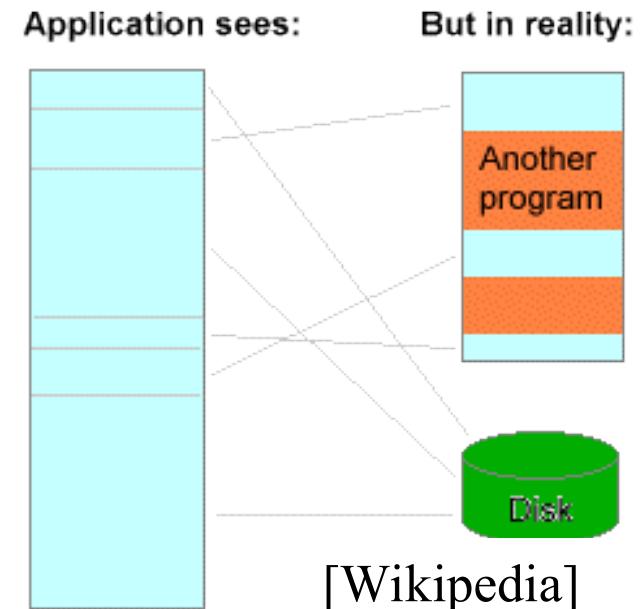
$$\begin{aligned}
 c_i &- \text{cijena po bitu} \\
 t_{ai} &- \text{vrijeme pristupa} \\
 S_i &- \text{kapacitet memorije}
 \end{aligned}$$

Tada vrijedi:

$$\begin{aligned}
 c_i &> c_{i+1} \\
 t_{ai} &< t_{ai+1} \\
 S_i &< S_{i+1}
 \end{aligned}$$

## Virtualna memorija, glavne ideje:

- **Virtualno** proširiti kapacitet radne memorije (DRAM) korištenjem prostora na magnetskom disku
  - **virtualna memorija** brzine DRAM-a, a veličine diska
  - kao i kod **priručnih** memorija, pouzdajemo se u **lokalnost pristupa**
- Dodatne funkcije (ne manje važne):
  - **adresno preslikavanje**: programima (procesima) pružiti **linearan** pogled na diskontinuirani fizički prostor
  - **transparentno** dijeljenje memorije
  - **zaštita pristupa**: onemogućiti neželjenu interakciju među procesima (korisničkim i jezgrenim)
    - posebno važno na višeprogramskim računalima!



D.J. Wheeler: "Any problem in computer science can be solved with another layer of indirection. But that usually will create another problem."

# VIRTUALNI MEMORIJSKI SUSTAV

1. Memorijска hijerarhija
2. **Fizički i logički adresni prostor**
3. Adresno preslikavanje
4. Višerazinsko straničenje, translacijski spremnik
5. Segmentacija
6. Detalji izvedbe

## Fizički adresni prostor

Skup stvarnih memorijskih lokacija oblikuje **fizičku memoriju**.

Funkciju fizičke memorije obavljaju elektronički uređaji priključeni na sabirnicu procesora, odnosno računala.

Adrese koje su jednoznačno dodijeljene fizičkim memorijskim lokacijama čine **fizički adresni prostor FAP**.

Npr, za 32-bitno računalo s 512 MB memorije:  $|FAP|=5.4e8B$ .

## Logički adresni prostor

Logički adresni prostor **LAP** je skup svih adresa koje generira programski model procesora.

Adrese varijabli i potprograma te pokazivači stogova su **logičke adrese**.

Npr, za 32-bitno računalo s 512 MB memorije,  $|LAP| = 4.3e9$  B

## Mogući odnosi između veličina fizičkog i logičkog prostora:

$|LAP| = |FAP|$ : često kod računala s 16-bitnim adresama

$|LAP| < |FAP|$

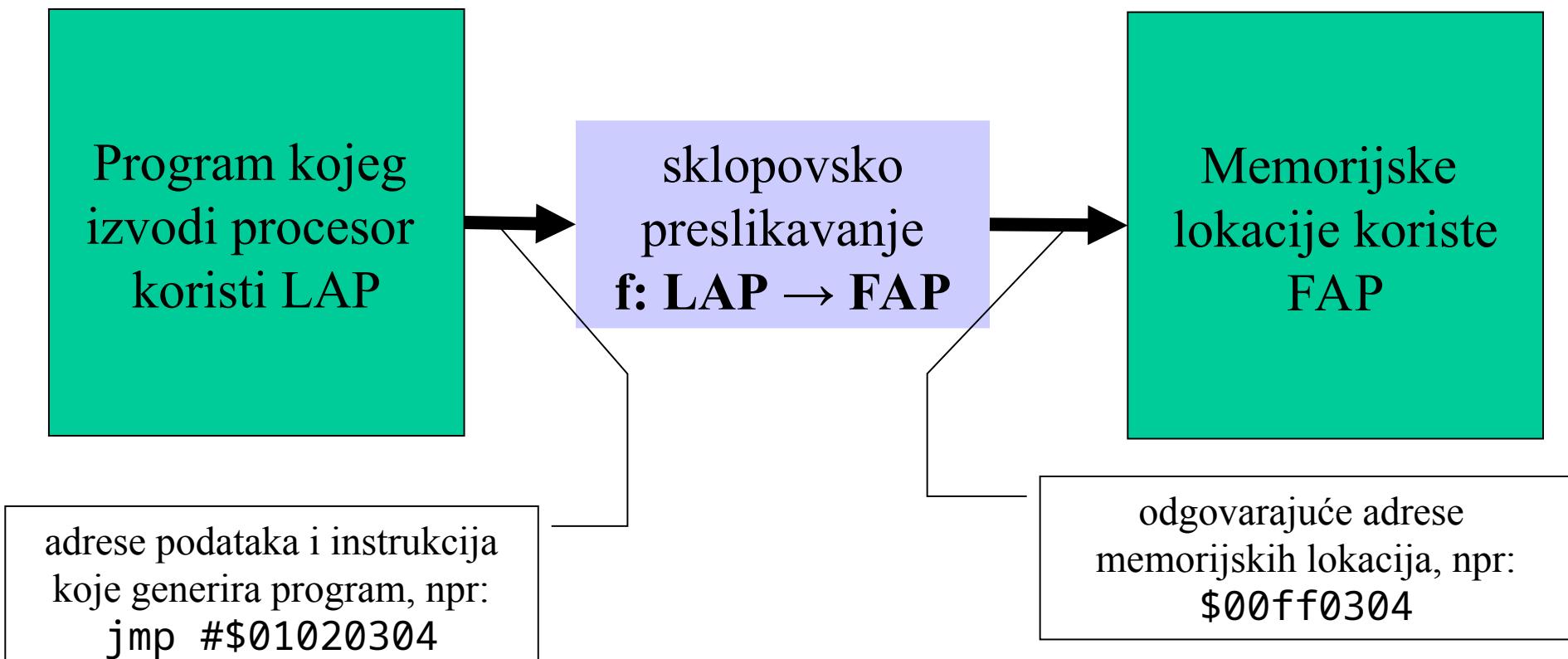
- kod računala bez adresnog preslikavanja:
  - FAP se izvodi s prekrivanjem (overlay)
  - programer odabire "vidljive" memorijske sklopove
- kod nekih 32-bitnih računala (x86: PAE):
  - procesor ima više adresnih linija nego što ISA koristi
    - P4: 36 linija, max  $|FAP| = 64 \text{ GB}$
  - pojedini korisnici transparentno koriste LAP od 4GB
  - ako korisnik želi više od 4GB, mora moliti OS za pomoć

$|LAP| > |FAP|$ : najčešći slučaj (sve vrste računala)

- poželjno zbog mogućnosti lakog proširenja  
(razlog za 64-bitne procesore)

# Adresno preslikavanje

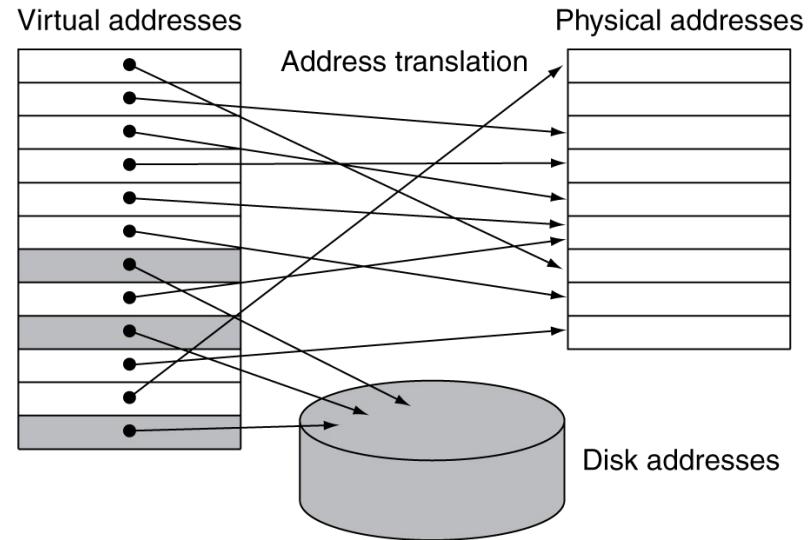
- ideja: svaki proces radi u **privatnom** virtualnom LAP-u
- OS odlučuje kamo se mapira koji LAP (nema fragmentacije, zaštita, ušteda)
- adresno preslikavanje ključni detalj priče, obavlja se uz pomoć **sklopovlja**



# Svojstva funkcije preslikavanja

Za najčešći slučaj  $LAP >> FAP$  funkcija preslikavanja  $f$  je:

$$f: LAP \rightarrow FAP \cup \phi$$



Za svaki  $a \in LAP$ , funkcija  $f$  vraćа:

[Patterson08]

$f(a) = a'$       ako se podatak s virtualnom adresom  $a$  nalazi na adresi  $a'$  u fizičkoj memoriji ( $a' \in FAP$ )

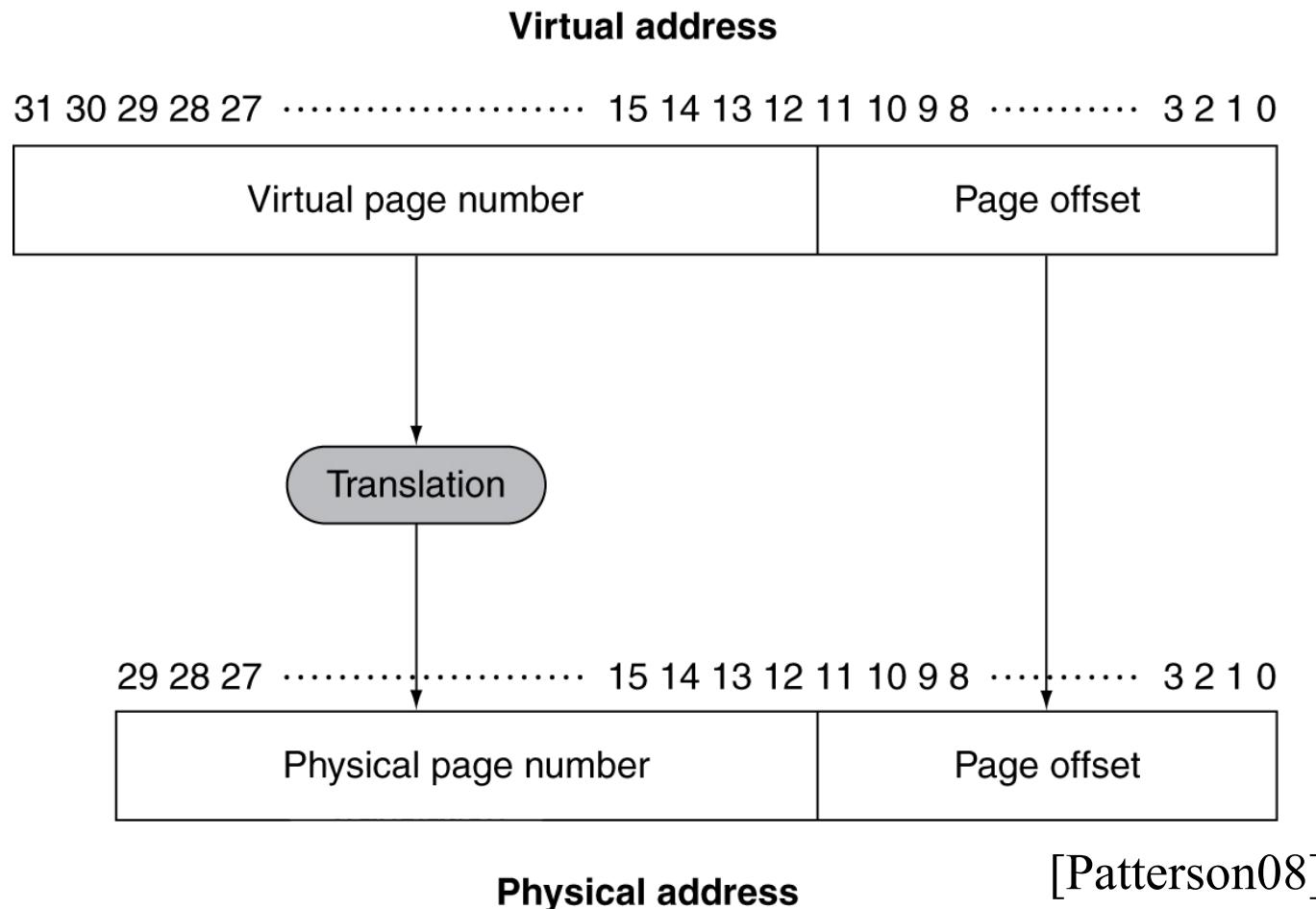
$f(a) = \phi$       označava **promašaj** virtualne memorije

# VIRTUALNI MEMORIJSKI SUSTAV

1. Memorijска hijerarhija
2. Fizički i logički adresni prostor
3. **Straničenje**
4. Višerazinsko straničenje, translacijski spremnik
5. Segmentacija
6. Detalji izvedbe

# Straničenje (engl. Paging)

- najčešće korištena implementacija VM, analogna cachevima:
  - preslikavaju se memorjski **blokovi** fiksne veličine
  - blokove nazivamo stranicama (tipična veličina stranice je 4KB)



## Straničenje se temelji na straničnoj tablici (ST):

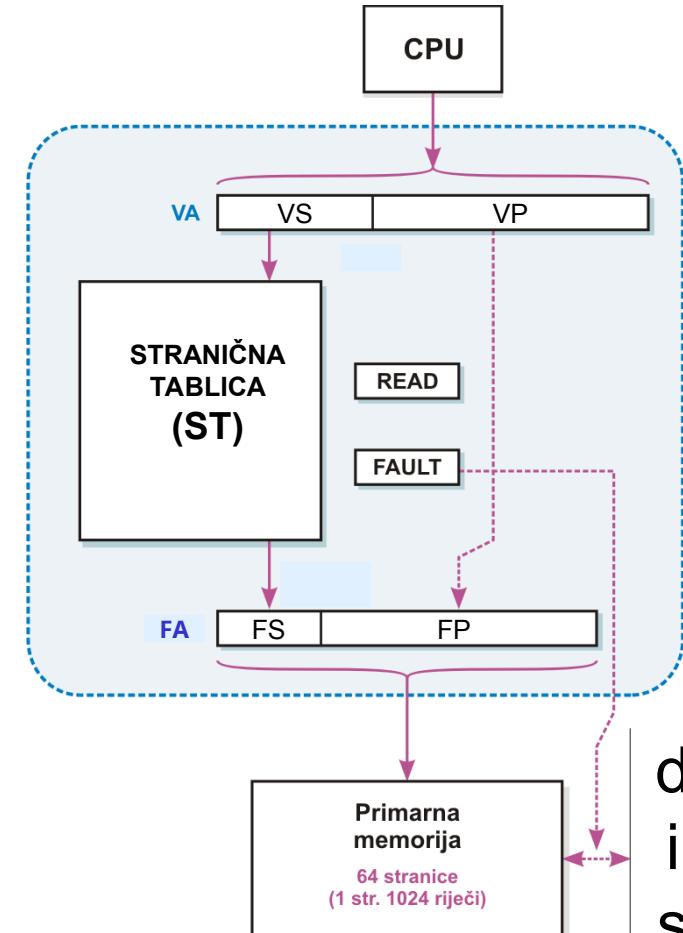
- virtualnu adresu (VA) dijelimo na dva dijela (kao kod cacheva)



- virtualna stranica (VS)** adresira ST
- virtualni pomak (VP)** adresira riječ stranice

## Određivanje fizičke adrese (FA):

- okvir:** FS = ST (VS)
- fizički pomak:** FP = VP!
- analogno priručnoj memoriji!



## Servisiranje promašaja VM (page fault):

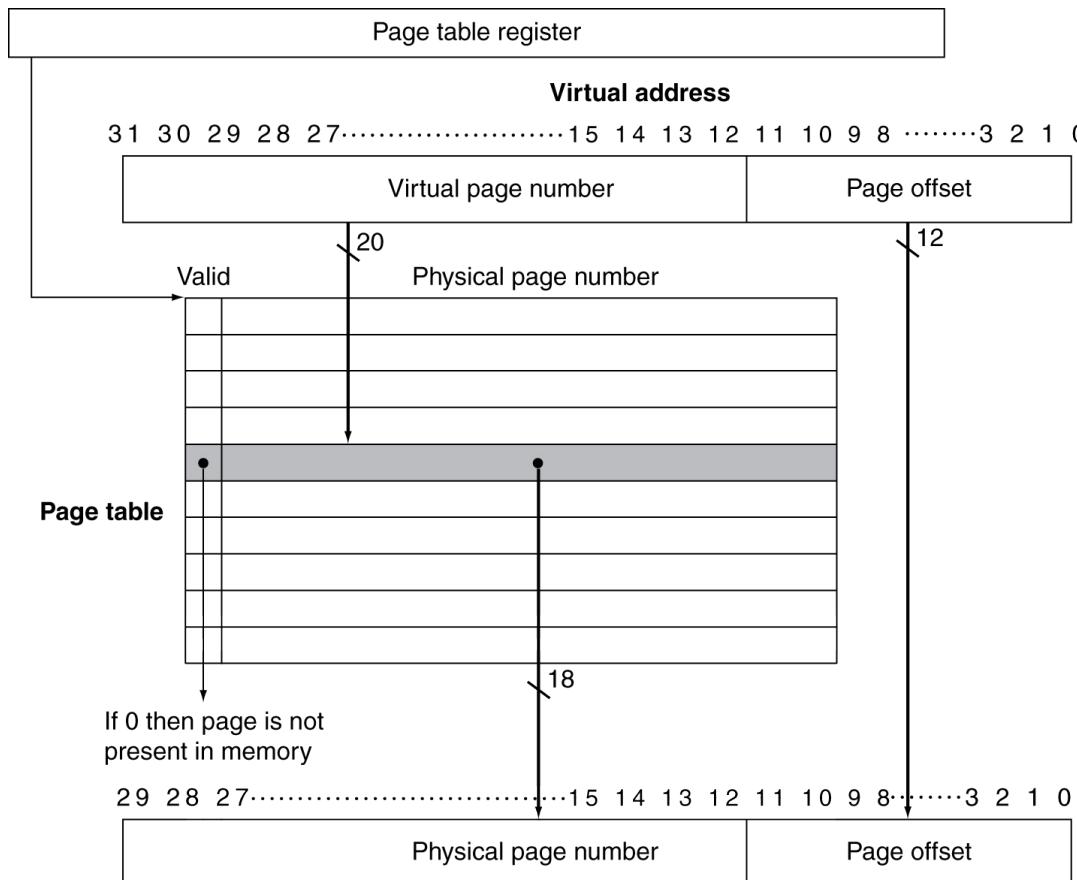
- stranica mora biti učitana s diska ( $\sim 1e6 \Delta T$ )
- za to se brine operacijski sustav
- sofisticirane tehnike minimiziranja promašaja

[Ribarić96]

d  
i  
s  
k

# ST se smješta u radnu memoriju

- element procesa, kao i registri (PC, ...)
- organizacija i održavanje u domeni OS-a
- pokazivač na ST procesa u posebnom registru



# Još o straničenju

- Najčešće korištena implementacija VM
  - stranice su jedinica transfera prema sekundarnoj memoriji
  - **prednosti**: transparentno preslikavanje, nema **vanske** fragmentacije
  - **nedostatci**: **unutrašnja** fragmentacija, složena implementacija
- Elementi ST (**stranični opisnici**) pohranjuju dodatne informacije o stranici:
  - bitovi kontrole pristupa (zaštite):
    - RO (read only)
    - NX (no execute)
    - S (supervisor only)
  - ostale servisne informacije:
    - D (dirty), A (accessed),
    - WT (write through), CD (cache disable), ...
  - omogućava se sofisticirano upravljanje memorijom (OS)

# Elementi ST: stranični opisnici (AMD Opteron)

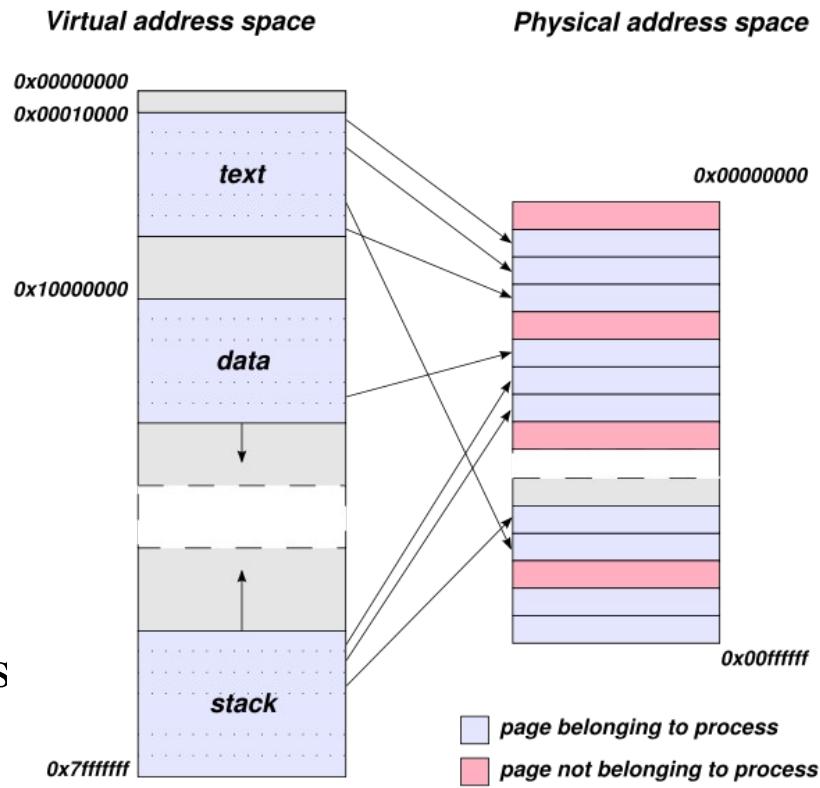


P	RW	US	D	A	NX	CD	WT	okvir

- P (presence): 0 ⇒ podatak je na disku
- RW (read/write): samo čitanje ili čitanje i pisanje
- US (user/supervisor): aplikacije/OS
- D (dirty): sadržaj je mijenjan
- A (accessed): da li je proces pristupao stranici?
- NX (no-execute): zabrana izvršavanja
- CD (cache-disable): zabrana cacheiranja
- WT (write-through): trenutni upis u fizičku memoriju

# Svojstva straničenja

- linearne logičke adrese:
  - straničenje rješava fragmentaciju
  - proces može dinamički **rasti**
    - npr, treba više prostora na stogu:
    - adresiramo nemapirani dio LAP-a
    - dolazi do iznimke (page fault)
    - OS **umeće** nove stranice u LAP
  - programi mogu imati apsolutne adres
- zaštita i privatnost:
  - svaki proces ima **privatnu** ST
  - proces ne može negativno utjecati na druge procese ili jezgru
- povećanje dostupnog prostora
  - ograničenje: veličina rezerviranog prostora na disku (swap file)
- mogućnost **dijeljenja** fizičkih stranica (biblioteke!)



[Wikipedia]

# VM straničenjem: problemi

## 1. značajan prostor potreban za linearu ST

- 32bit LAP, 4KB stranice  $\Rightarrow$  20b oznake  
 $\Rightarrow$  4MB za **svaki proces** (pa i onaj najmanji)!
- a što je s računalima koja imaju 64-bitni LAP???
- rješava se **višerazinskim ST**

## 2. svaki logički pristup $\Rightarrow$ (barem) dva fizička pristupa

- pristup ST (više njih?) + pristup podatku
- a što je s pristupnim bitovima (D,A,R/W, ...)?
- rješava se **cacheiranjem opisnika ST** (translacijski spremnik, TLB)

## 3. što ako **radni skup** (popularne stranice) $>$ FAP?

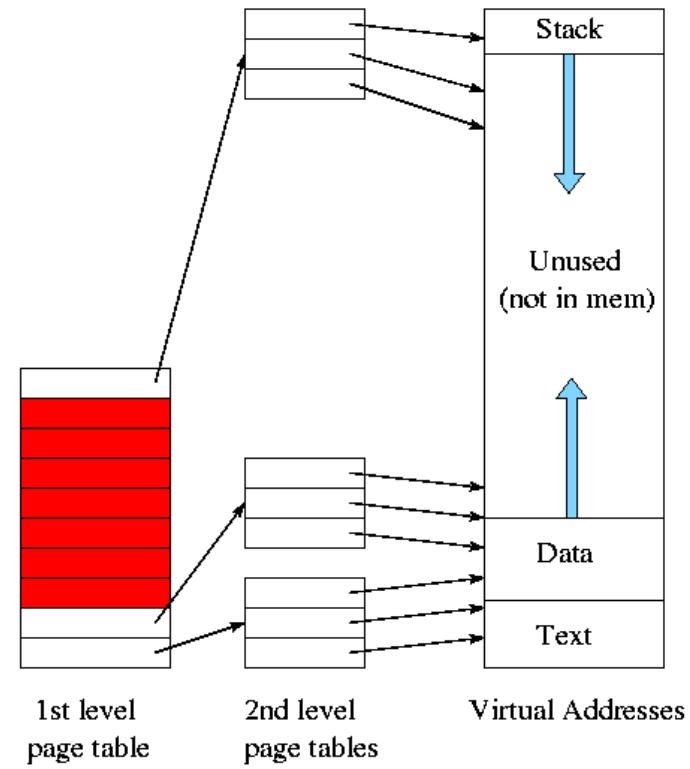
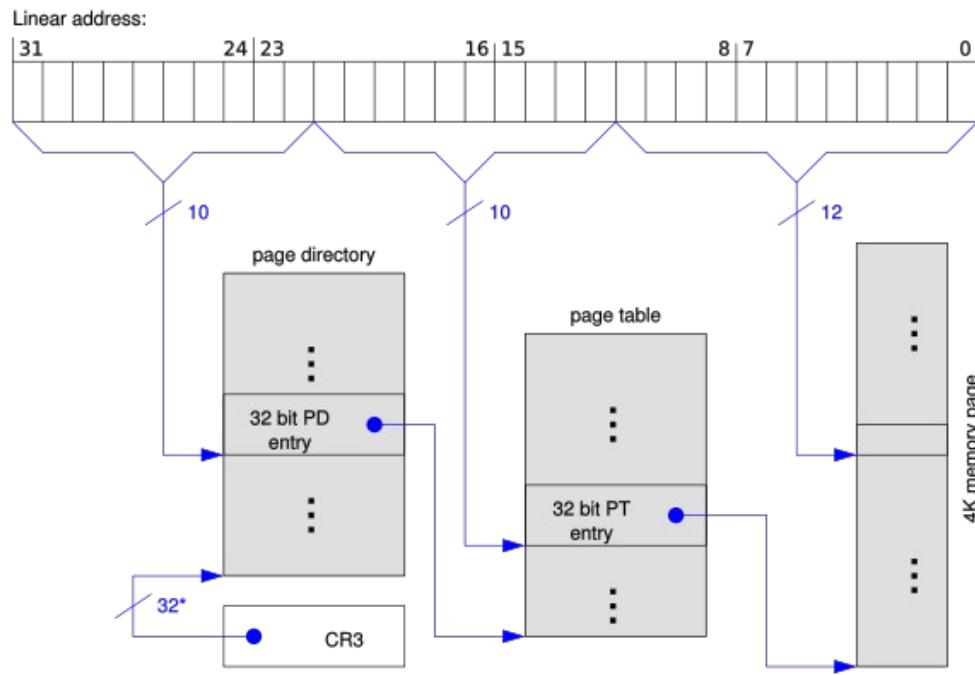
- u najgorem slučaju, svaki podatak dohvaćamo s diska
- gubimo vrijeme u žongliraju stranicama - thrashing (mlaćenje)
- koncept VM **nije prikladan** u takvim slučajevima

# VIRTUALNI MEMORIJSKI SUSTAV

1. Memorijска hijerarhija
2. Fizički i logički adresni prostor
3. Straničenje
4. **Višerazinsko straničenje, translacijski spremnik**
5. Segmentacija
6. Detalji izvedbe

# Višerazinske stranične tablice

- **Problem** linearne ST: mali procesi plaćaju punu cijenu preslikavanja!
- **Višerazinska ST:** manja cijena nekorištenih dijelova LAP-a (prilagodljivo tabeliranje)
- Dvorazinsko straničenje x86 (nakon 80386):

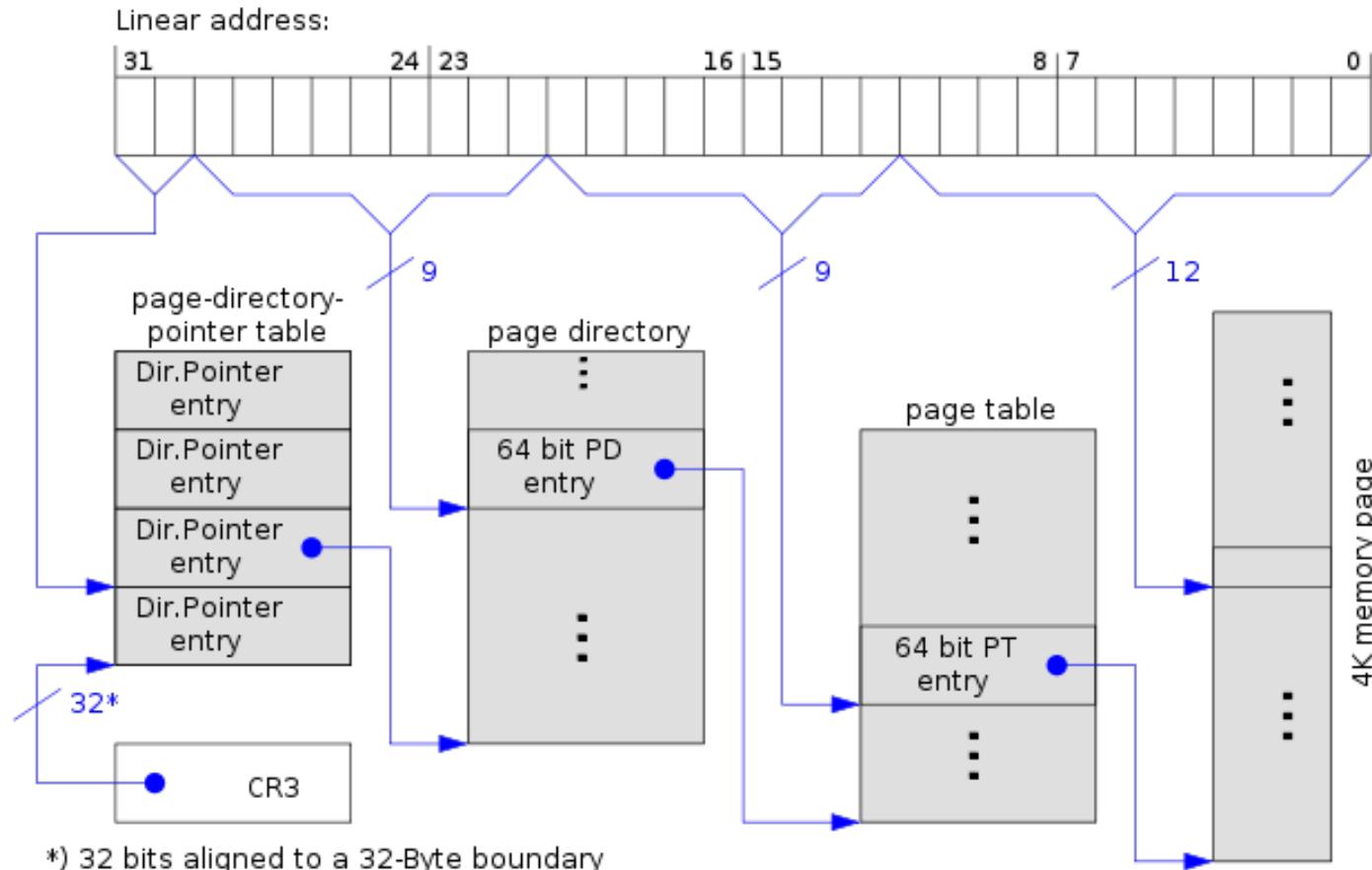


[Gottlieb00]

[Wikipedia]

# Trorazinsko straničenje x86 (Pentium II →)

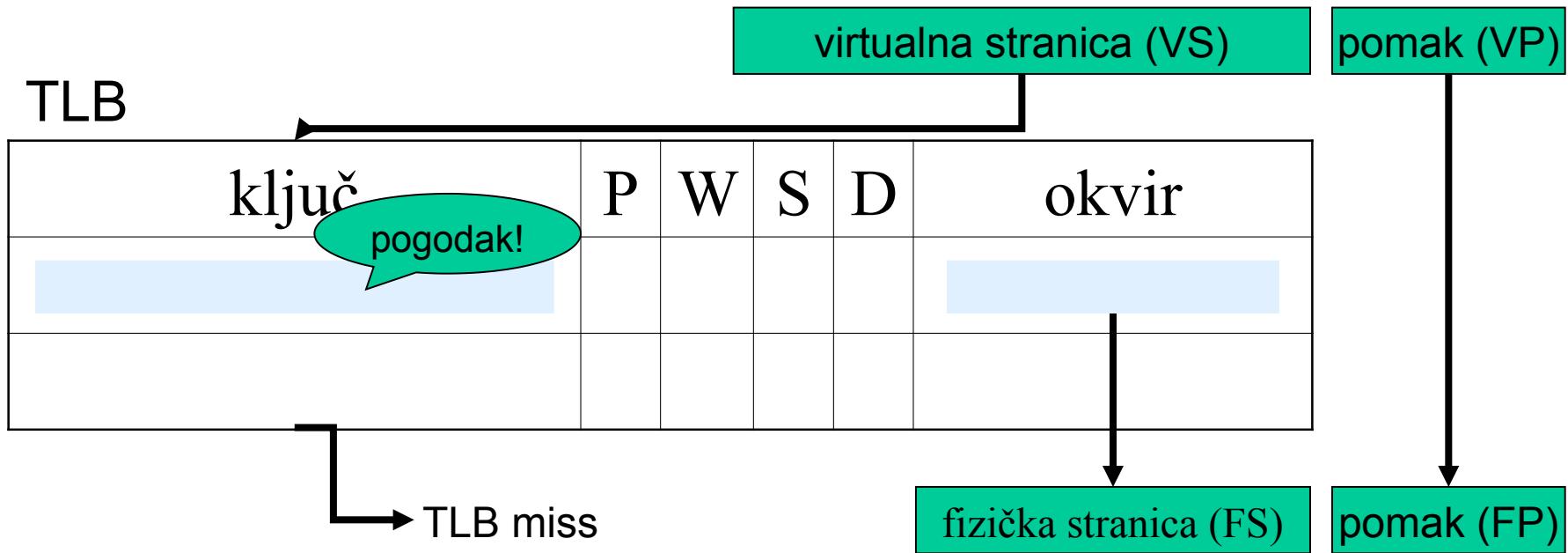
- ekstenzija PAE: physical address extension
- $|FAP|=2^{36} B$  (pojedinačni procesi koriste  $|LAP|=2^{32} B$ )



[Wikipedia]

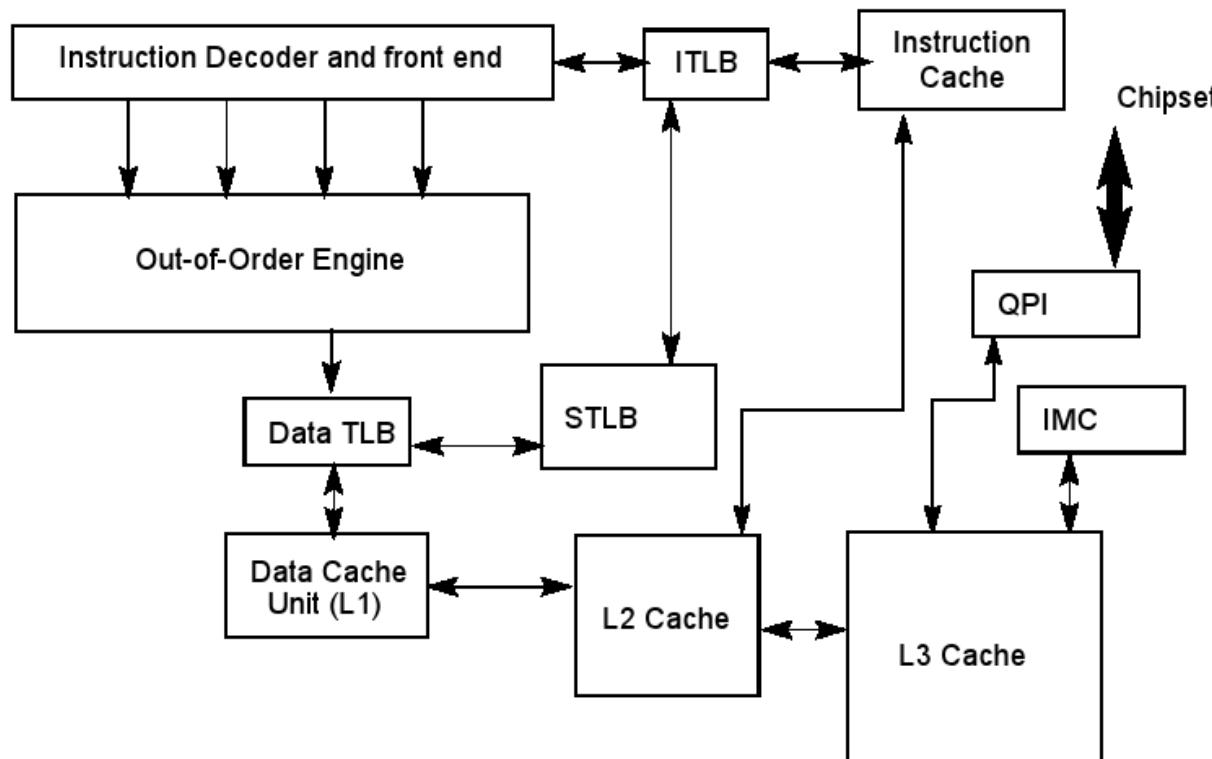
# Korištenje translacijskog spremnika (TLB)

- **Problem:** više fizičkih pristupa memoriji  $\forall$  preslikavanje
- **Ideja:** cacheirati posljednjih nekoliko stotina preslikavanja
  - TLB: **asocira** virtualnu stranicu s odgovarajućim SO (kao kod PM!)
  - u slučaju pogotka, gotovi smo u jednom ciklusu
  - u slučaju promašaja, šetnja straničnom tablicom te upis opisnika u TLB
  - punjenje TLB-a može biti programsko (OS) ili sklopovsko (MMU)



# Odnos TLB-a i priručne memorije

- Najčišće kad je (barem koncepcualno) TLB ispred cachea
  - dobro: u cacheu ne mogu biti dvije LA koje se odnose na istu FA!
  - problem: tada latencija TLB-a produžava latenciju pristupa cacheu
  - to se rješava na razne komplikirane načine
    - ti načini su izvan dosega ovog kolegija :-)



## TLB, izvedbeni detalji

- veličine variraju (8 - 4096 zapisa), kao i broj razina (1 ili 2)
- kao i kod PM, kombinira se nepotpuno indeksiranje i asociranje
- manji TLB-ovi potpuno asocijativni, veći skupno asocijativni
  - ST koristi potpuno indeksiranje:
    - rijetko se proziva, višerazinska struktura štedi prostor
    - želimo iskoristiti sve stranične okvire (tj, fizičku memoriju)
- postotak pogotka vrlo visok, promašaji  $\sim 0.1\%$   
(stranica veća od linije, 4096B vs. 64B)
- cijena promašaja niska, pogotovo ako je ST u cacheu

## Zadano:

- virtualna adresa:  $w(VA) = 40$  b
- fizička adresa:  $w(FA) = 36$  b
- veličina stranice: 16 kB

Odrediti strukturu VA i FA:

$$w(VS), w(FS), w(VP), w(FP) = ?$$

- $w(VP) = w(FP) = 14$  b
- $w(VS) = 26$  b
- $w(FS) = 22$  b

## Zadano:

- virtualna adresa:  $w(VA) = 40$  b
- fizička adresa:  $w(FA) = 36$  b
- veličina stranice: 16 kB
- TLB: 512 zapisa, 2× asocijativan, 4 bita opisa stranice

Odrediti strukturu i ukupan broj bitova zapisa u TLB-u  
(oznaka, bitovi opisa, fizička stranica):  $w(TLBo) + 4 + w(FS) = ?$

## Rješenje:

- VS → TLBO || TLBI (kao i kod PM!)
- $w(TLBindeks) = 8$  b                    ( $= \log_2(512/2)$ )
- $w(TLBoznaka) = 18$  b                    ( $= 40 - 14 - 8$ )
- $w(TLBzapis) = 44$  b                    ( $= 18_{TLBozn} + 4 + 22_{FS}$ )

# VIRTUALNI MEMORIJSKI SUSTAV

1. Memorijska hijerarhija
2. Fizički i logički adresni prostor
3. Straničenje
4. Višerazinsko straničenje, translacijski spremnik
- 5. Segmentacija**
6. Detalji izvedbe

# Segmentacija

- Razlike u odnosu na straničenje:
  - blokovi su veći i promjenljive duljine
  - LAP procesa čini nekoliko segmenata
- prednost: preslikavanje zbrajanjem
  - jednostavno ( $FA = S + LA$ ), jeftino (tablica deskriptora  $\ll ST$ )
  - (početak i duljina segmenta, prava pristupa)
- nedostatak: upravljanje memorijom na pregruboj razini
  - vanjska fragmentacija: neiskorišteni prostor između segmenata  
(teško naći kontinuirani logički interval u kojem preslikati novi segment)
  - netransparentan pristup kad:
    - iscrpimo mogućnosti rasta tekućeg segmenta
    - pristupamo dijeljenim segmentima (potreban poseban tip pokazivača)
- arhitektura x86 podržava i segmentaciju i straničenje  
(moderni OS-ovi koriste samo straničenje)

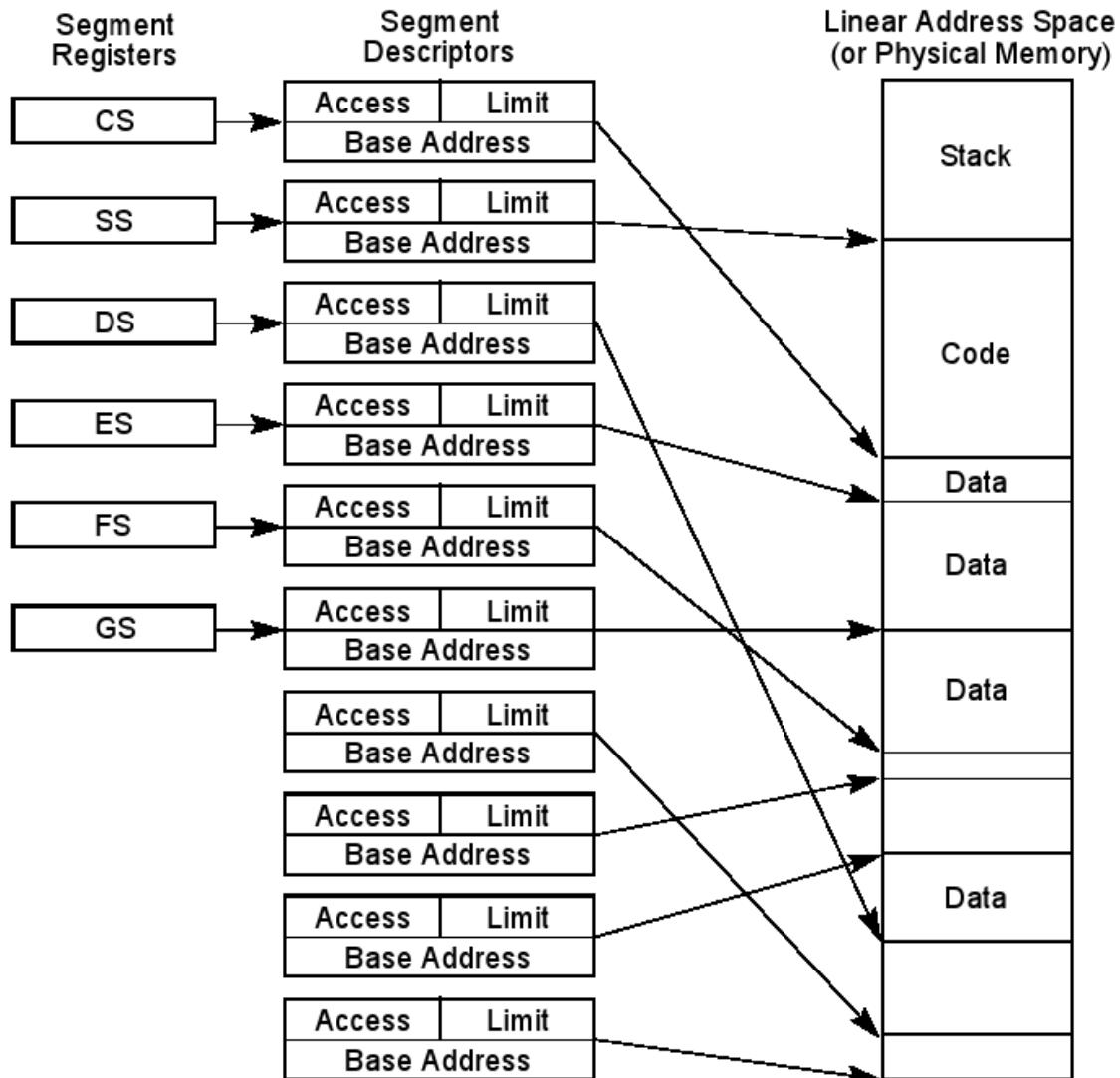
Local Descriptor Table (LDT)

	Linear base address (BASE)	Segment size (LIMIT)
4	0x21430	0xC000
3		
2	0x0CEFO	0xA300
1	0x28C00	0xFC00
0		

Segment 3  
Segment selector 0x000FSegment 2  
Segment selector: 0x0027Segment 1  
Segment selector 0x0017[Wikipedia  
]

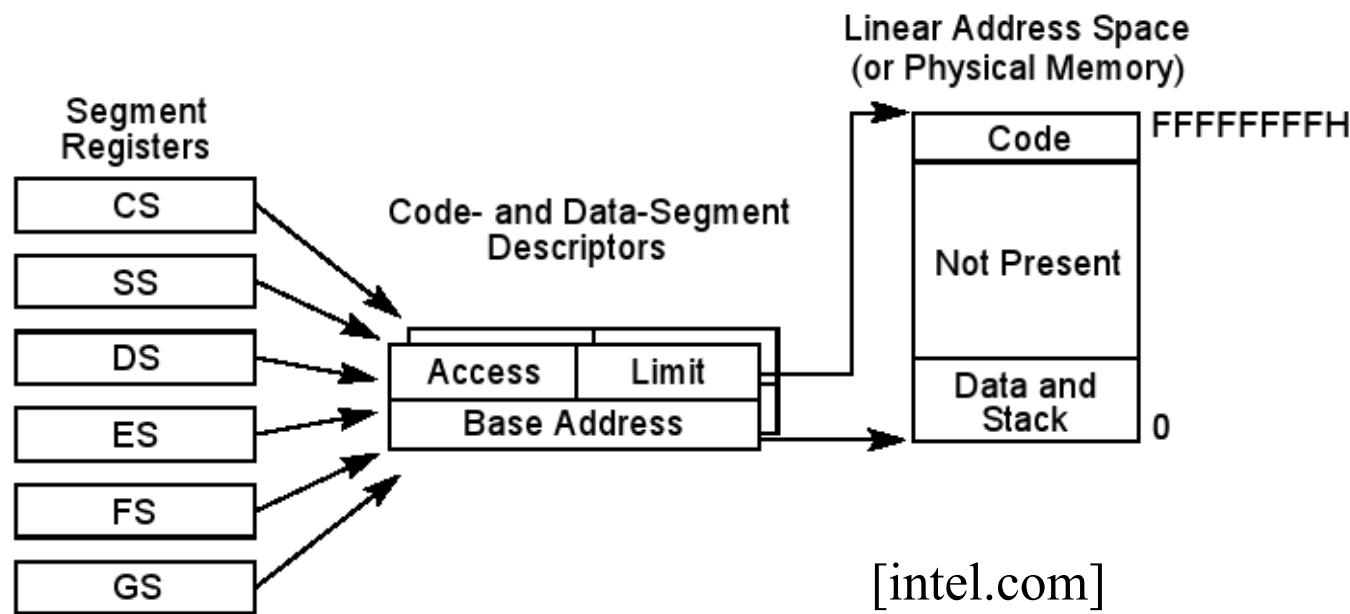
# Segmentacija x86

- pristupom memoriji upravlja 6 segmentnih registara
- svaki segmentni opisnik sadrži bitove zaštite i veličinu
- nedostatak: pokazivač na podatke izvan segmenta ima 48 bitova
- nedostatak: vanjska fragmentacija
- nedostatak: što ako je segment nedovoljno velik?

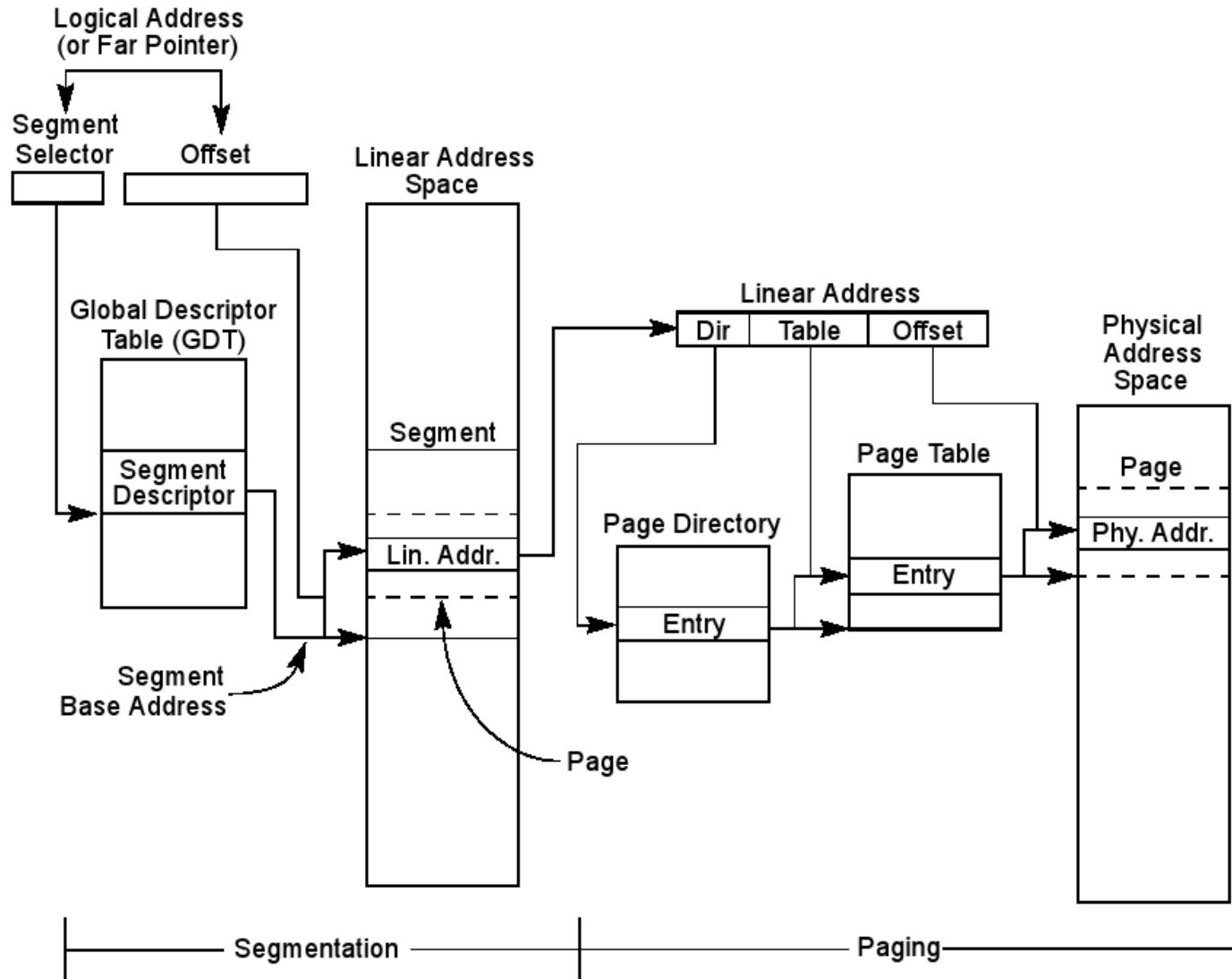


## Segmentacija x86 (2)

- moderni OS-ovi konfiguiraju minimalno korištenje segmentacije
- Intel taj način naziva “basic flat model” (vidi sliku)
- preslikavanje i zaštita se prepušta straničenju
- segmentacija danas postoji prvenstveno zbog kompatibilnosti
- AMD Opteron u 64-bitnom načinu rada nema segmentaciju



# VM x86: kombinacija segmentacije i straničenja



# **VIRTUALNI MEMORIJSKI SUSTAV**

- 1. Memorijska hijerarhija**
- 2. Fizički i logički adresni prostor**
- 3. Straničenje**
- 4. Višerazinsko straničenje, translacijski spremnik**
- 5. Segmentacija**
- 6. Detalji izvedbe**

# Obrada promašaja stranice

Promašaj stranice generira iznimku koju servisira OS:

1. locirati SO na temelju adrese koja je uzrokovala promašaj
  - šetnja višerazinskom ST (ona bi trebala biti u RAM-u)
2. ako SO nije valjan (tj. referenciramo novu fizičku stranicu):
  - ili mapirati novu stranicu (npr. pri širenju stoga)
  - ili likvidirati program zbog ilegalnog pristupa
3. ako je fizička memorija popunjena, odabrati stranicu koju ćemo izbaciti (swap-out, evict)
  - ako smo u stranicu pisali (bit D), upisujemo je na disk (**dugo čekanje**)
4. ako je SO valjan: učitati stranicu s diska (**dugo čekanje**)
5. ažurirati ST, premjestiti proces u red aktivnih procesa
  - nastavak izvođenja od instrukcije koja je generirala promašaj

# Zamjena stranice, upis na disk

- najčešće potpuno asocijativno preslikavanje + LRU
  - potpuna asocijativnost izvediva zbog ogromne ST u RAM-u
  - izvedba LRU se temelji na servisnom bitu pristupa (A):
    - postavlja se na 1 kod svakog pristupa (TLB)
    - operacijski sustav periodički poništava sve bitove pristupa
    - stranica s A = 0 nije korištena u bliskoj prošlosti
- tehnike upisa na disk:
  - promptno upisivanje (write through) uglavnom nepraktično
  - tipično, stranica se upisuje na disk tek nakon zamjene (write back)
  - servisni bit D (dirty) postavlja se u SO nakon svakog upisa

# Performansa memorijskog sustava

- možemo dobiti promašaj u: TLB-u, SO-u, cacheu, DRAM-u
- neki od mogućih ishoda:
  - pogodak TLB-a i cachea (zaobilazimo radnu memoriju): 2 ns
  - pogodak TLB-a, promašaj cachea, pogodak DRAM: 50 ns
  - promašaj TLB-a, SO u cacheu, pogodak cachea: 10 ns
  - promašaj TLB-a, SO u DRAM-u, promašaj cachea, pogodak DRAM: 100 ns
  - promašaj TLB-a, SO u DRAM-u, promašaj cachea, promašaj DRAM: 10 ms  
(tijekom tih 10ms OS pokušava dodijeliti CPU nekom drugom procesu)
- širok spektar mogućnosti i to bez razmatranja PM L2, PM L3, TLB L2!
- vrijeme pojedinog pristupa memoriji podataka vrlo teško predvidjeti  
(prisjetimo se, u prosjeku svaka treća instrukcija pristupa memoriji)
  - pristupi instrukcijskoj memoriji znatno pravilniji!
- u prisustvu nepredvidivih zastoja u vezi s memorijom (i predviđanjem grananja) prednost imaju procesori s dinamičkim raspoređivanjem!

## Usporedba virtualne i priručne memorije [Hennesy07]

<b>parametar</b>	<b>PM (L1)</b>	<b>VM</b>
veličina bloka	16 - 128 B	4 - 64 kB
vrijeme pogotka	1 - 3 $\Delta T$	100 - 200 $\Delta T$
vrijeme promašaja	8 - 200 $\Delta T$	$10^6$ - $10^7 \Delta T$
učestalost promašaja	$10^{-3}$ - $10^{-1}$	$10^{-7}$ - $10^{-5}$
adresno preslikavanje	25 - 45 b $\rightarrow$ 14 - 20 b	32 - 64 b $\rightarrow$ 25 - 45 b

## Virtualna memorija, prednosti:

### 1. transparentno adresno preslikavanje

- ogromni kontinuirani LAP, fragmentirani FAP
- nekorišteni dijelovi programa ostaju na disku

### 2. transparentna zaštita procesa i jezgre OS

- ograničena neželjena interakcija među procesima
- upravljanje pristupom memoriji (RO, NX, S, ...)

### 3. transparentno rukovanje dijeljenom memorijom

- dijeljene biblioteke (libc, libm, ...)
- usporedna obrada više procesa

### 4. prividno proširenje kapaciteta radne memorije

- blokirani procesi smještaju se na disk
- mogućnost hibernacije

Virtualna memorija, izvedbeni detalji:

## Translacijski spremnik

- svaki pristup memoriji zahtjeva adresno preslikavanje
- TLB omogućava dovoljno brzo preslikavanje u najčešćem slučaju

## Višerazinsko preslikavanje

- svaki proces zahtjeva svoju straničnu tablicu
- stranične tablice zahtijevaju velike količine memorije
- višerazinska organizacija omogućava da maleni procesi ne plaćaju veliku memorijsku cijenu