

# Calculatoare Numerice (2)

– Cursul 5 –

Memoria virtuală

Facultatea de Automatică și Calculatoare  
Universitatea Politehnica București

**CN2 DE LA 8AM**

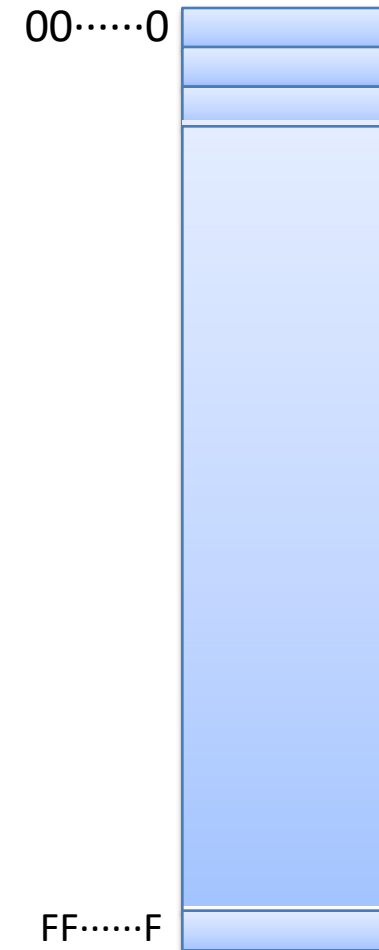
*I am inevitable.*

imgflip.com



# Memoria Virtuală

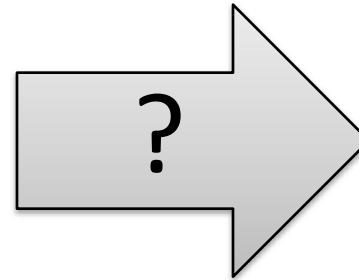
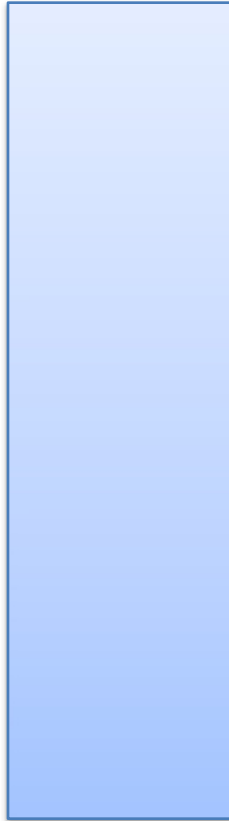
- Programele fac referințe la adrese din memoria virtuală
  - `movl (%ecx),%eax`
  - Organizată conceptual ca un vector foarte mare de octeți
  - Fiecare octet are propria adresă
- De fapt, implementat ca o ierarhie de diferite tipuri de memorii
- Sistemul furnizează spații de adresă pentru fiecare proces
- Alocare: La compilare și run-time
  - Unde trebuie stocate diferitele părți ale programului
  - Toată alocarea se face într-un singur spațiu virtual de adresă
- *Dar de ce avem memorie virtuală?*
- *De ce nu avem direct memorie fizică?*



# Problema 1: Unde încape totul?

---

Adrese pe 64 de biți:  
16 ExaByte  
(1EB = 1.000.000.000GB)



Memorie fizică:  
Câțiva GigaBytes



Și sunt multe procese...



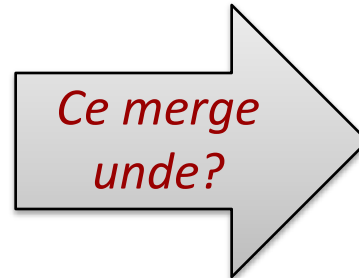
# Problema 2: Memory Management

---

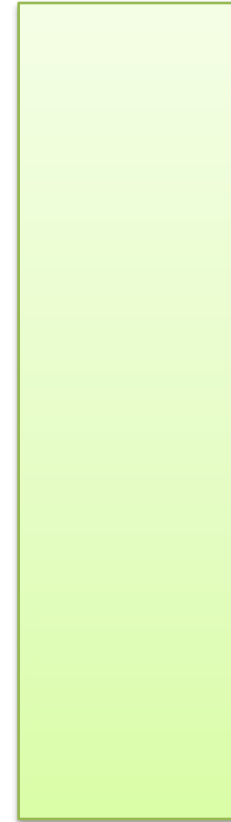
Process 1  
Process 2  
Process 3  
...  
Process n

**X**

stack  
heap  
.text  
.data  
...

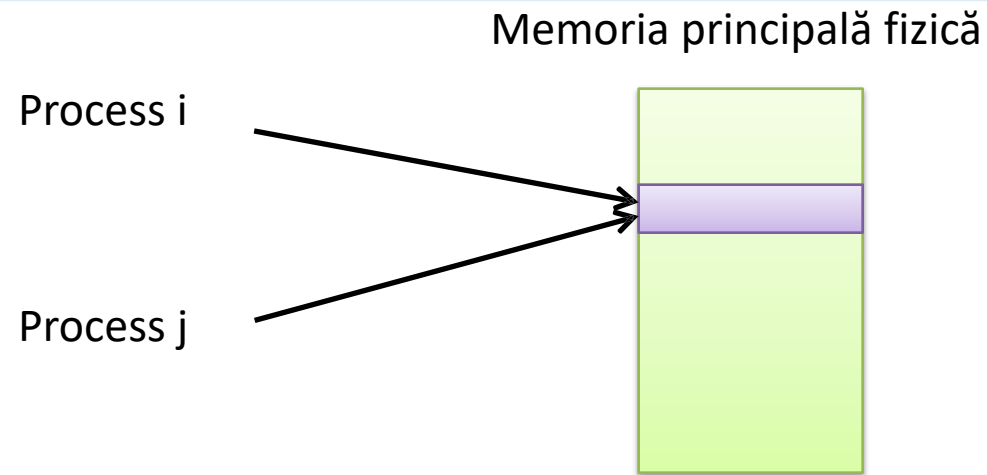


Memoria fizică (RAM)

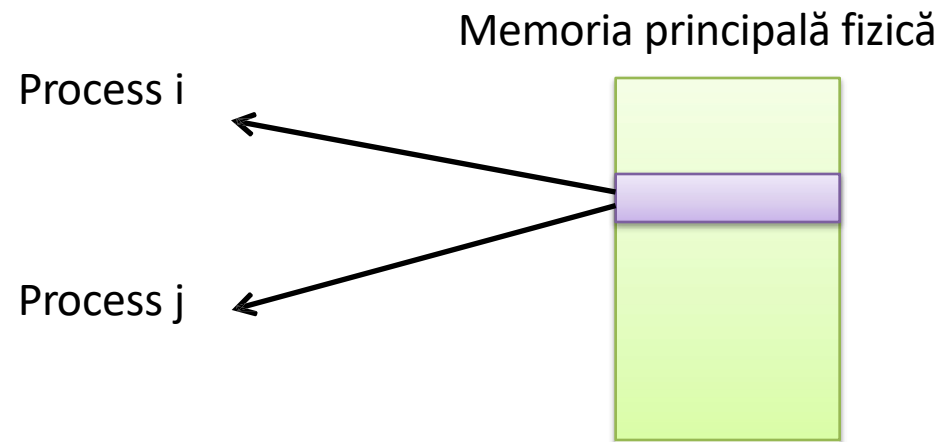


## Problema 3: Protecție

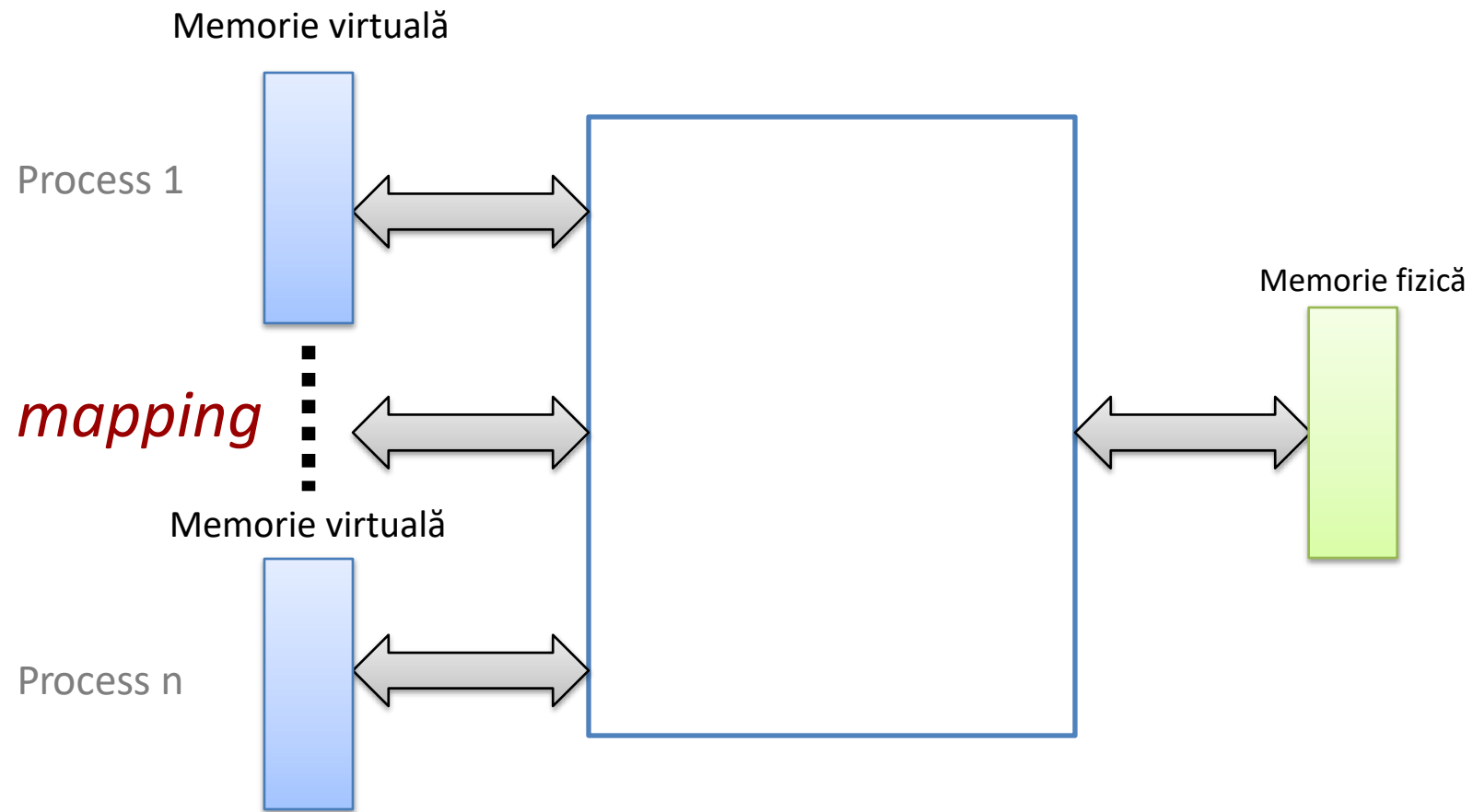
---



## Problema 4: Partajarea memoriei



# Soluție: Mapare



- Fiecare proces primește un spațiu propriu de memorie
- Rezolvă toate problemele anterioare

# Spații de adresă

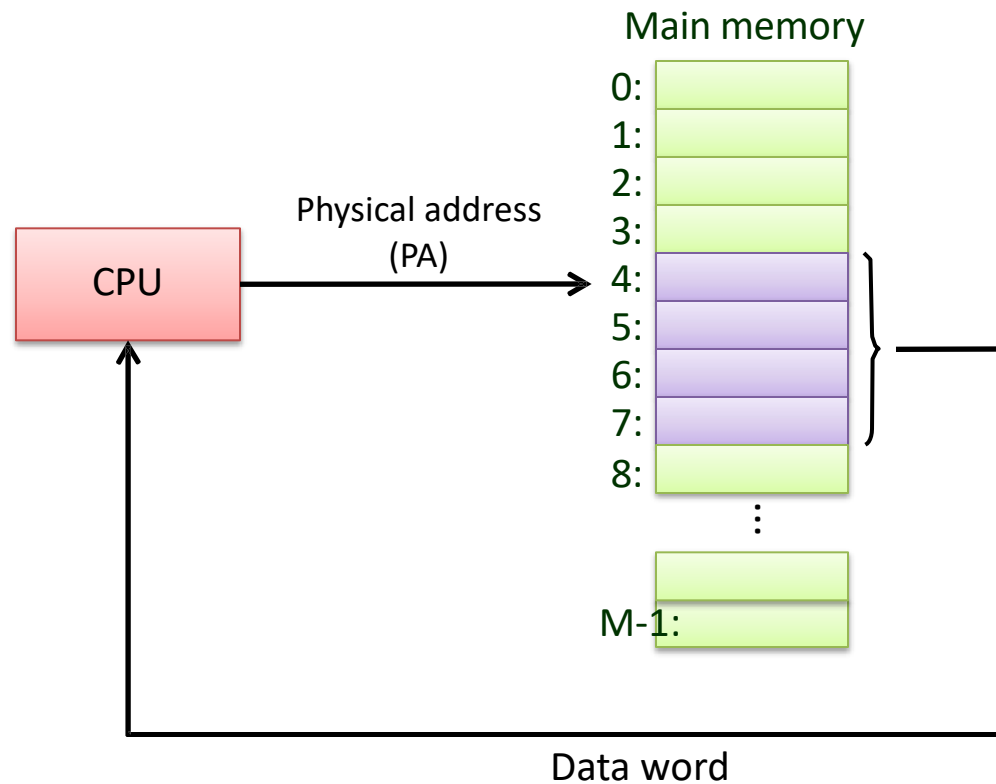
---

- **Spațiu liniar de adresă:** Set ordonat și contiguu de adrese întregi ne-negative:  
{0, 1, 2, 3 ... }
- **Spațiu virtual de adresă:** Set de  $N = 2^n$  adrese virtuale  
{0, 1, 2, 3, ..., N-1}
- **Spațiu fizic de adresă:** Set de  $M = 2^m$  adrese fizice  
{0, 1, 2, 3, ..., M-1}
- Distincție clară între date (bytes) și atributele acestora (adrese)
- Fiecare obiect poate avea acum adrese multiple
- Fiecare octet din memoria principală:  
o adresă fizică, una (sau mai multe) adrese virtuale



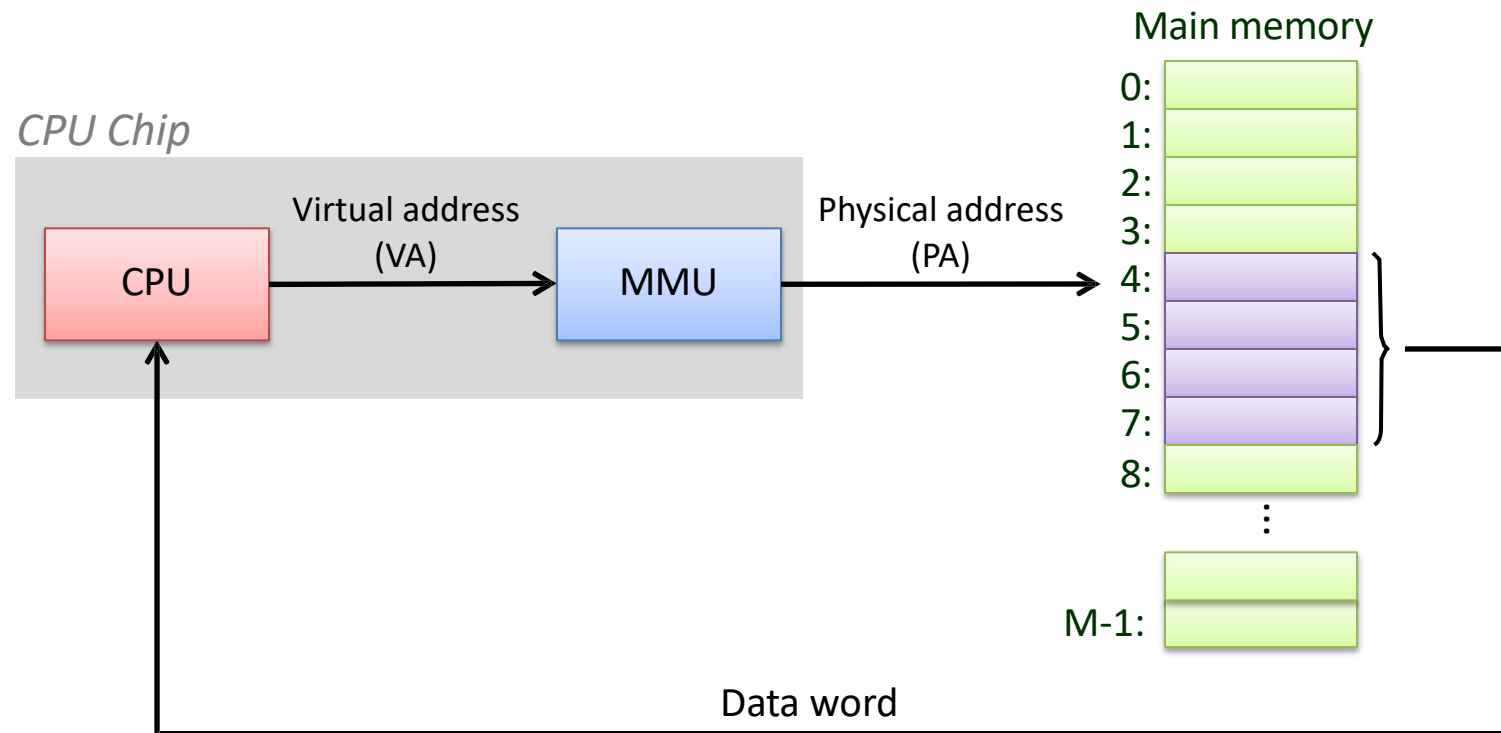


# Sistem ce folosește adresarea fizică



- [Încă] folosite în sisteme "simple" cum sunt microcontrollele (din mașini, lifturi, cuptoare cu microunde, telefoane mobile, rame foto digitale...)

# Sistem cu adresare virtuală



- Folosite în toate calculatoarele moderne (desktop, laptop, server)
- Una din marile "invenții" ale computer science
- MMU verifică cache-ul*

# De ce avem Virtual Memory (VM)?

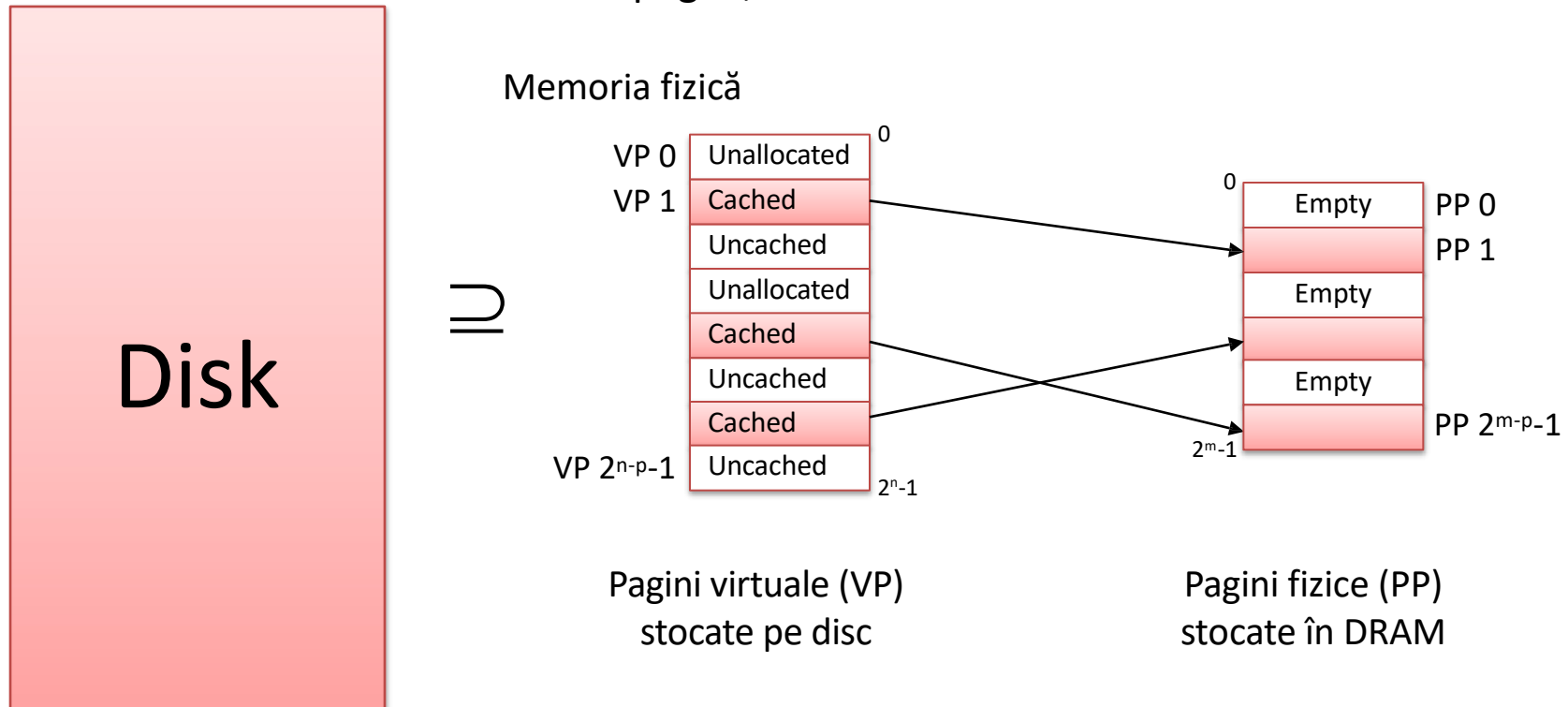
---

- Utilizare eficientă a memoriei principale(RAM)
  - Folosește RAM-ul ca un cache pentru părți dintr-un spațiu virtual de adrese
    - Unele părți care nu sunt în cache sunt stocate pe disc
    - Restul de părți nealocate nu sunt stocate nicăieri
  - Ține în memorie doar zonele active ale spațiului virtual de adresă
    - Transferă datele înainte și înapoi după necesități
- Simplifică managementul memoriei pentru programatori
  - Fiecare proces primește același spațiu privat și liniar de adrese
- Izolează spațiile de adrese
  - Un proces nu poate modifica zona de memorie a altui proces
    - Pentru că operează în spații de memorie diferite
  - Utilizatorii nu pot accesa informații privilegiate
    - Spații diferite de adresă au diferite permisiuni

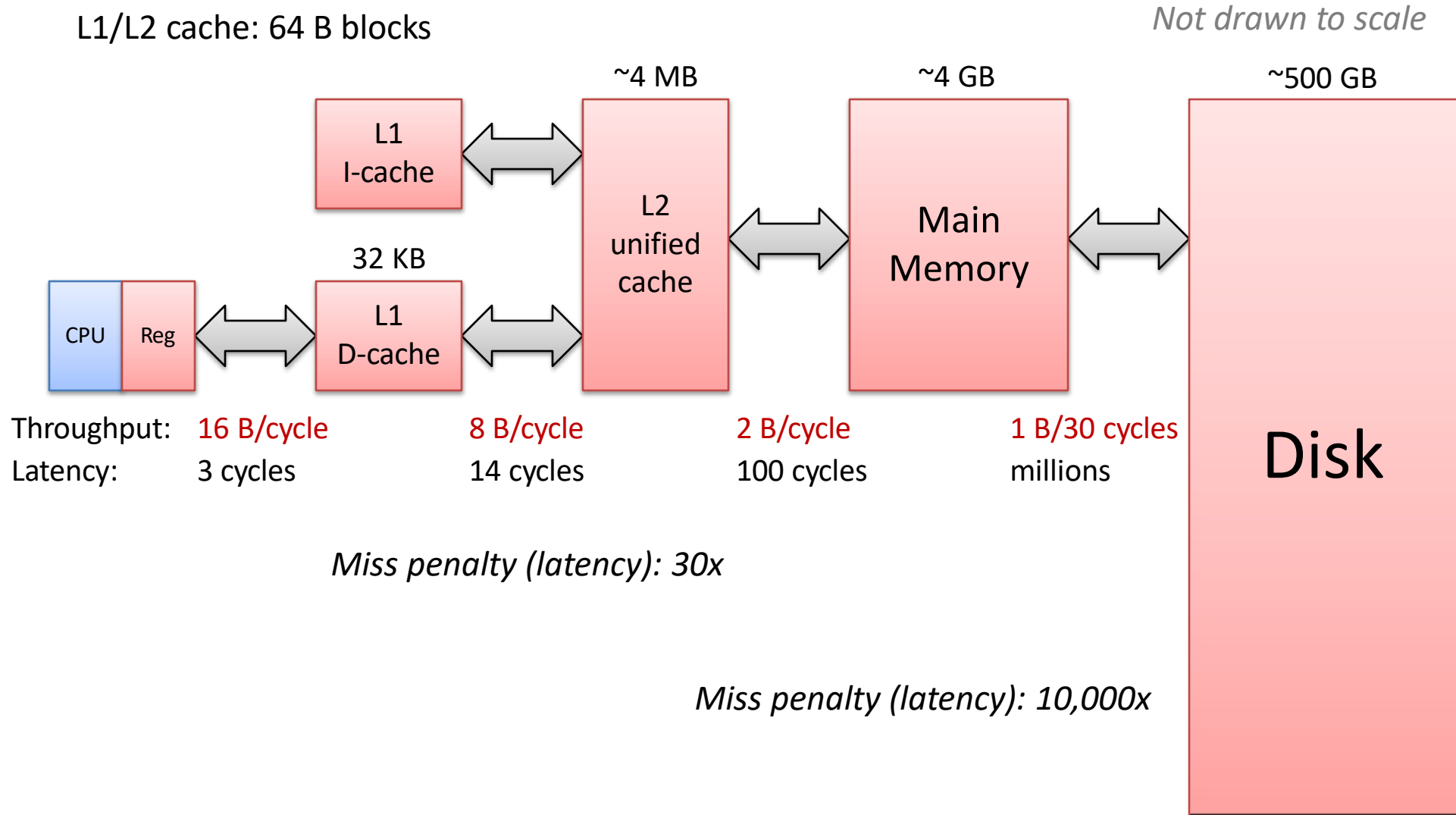


# VM ca utilitar pentru caching

- Memorie virtuală: vector de  $N = 2^n$  octeți
- gândiți-vă că vectorul (partea alocată din el) este stocat pe disc
- Memorie fizică principală (DRAM) = cache pentru memoria virtuală alocată
- Blocurile sunt denumite pagini; dimensiune =  $2^p$



# Ierarhia de memorii: Core 2 Duo



# Organizarea cache DRAM

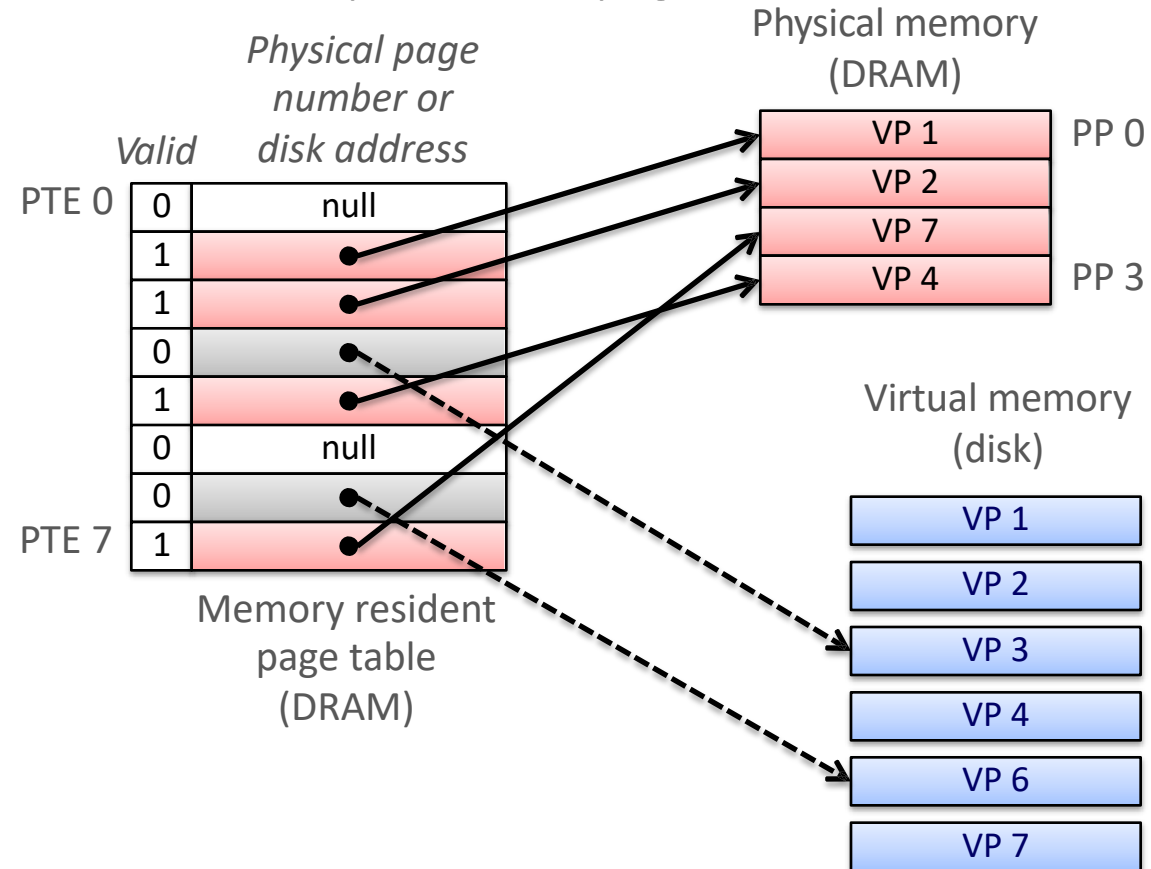
---

- Organizarea DRAM ca un cache este provocată de penalizările enorme pentru miss
  - DRAM este de aprox **10x** mai lentă decât SRAM
  - Hard-disk este de **10,000x** mai lent decât DRAM
    - Pentru primul octet, mai rapid pentru următorii
- Consecințe
  - Dimensiuni mari ale paginilor: de obicei 4-8 KB, câteodată 4 MB
  - Complet-asociativ
    - Orice VP poate fi plasată în orice PP
    - Necesită o funcție "mare" de mapare – diferit față de cache CPU
  - Algoritmi de înlocuire foarte sofisticăți și costisitori
    - Prea complicat de implementat în hardware
  - Write-back în loc de write-through

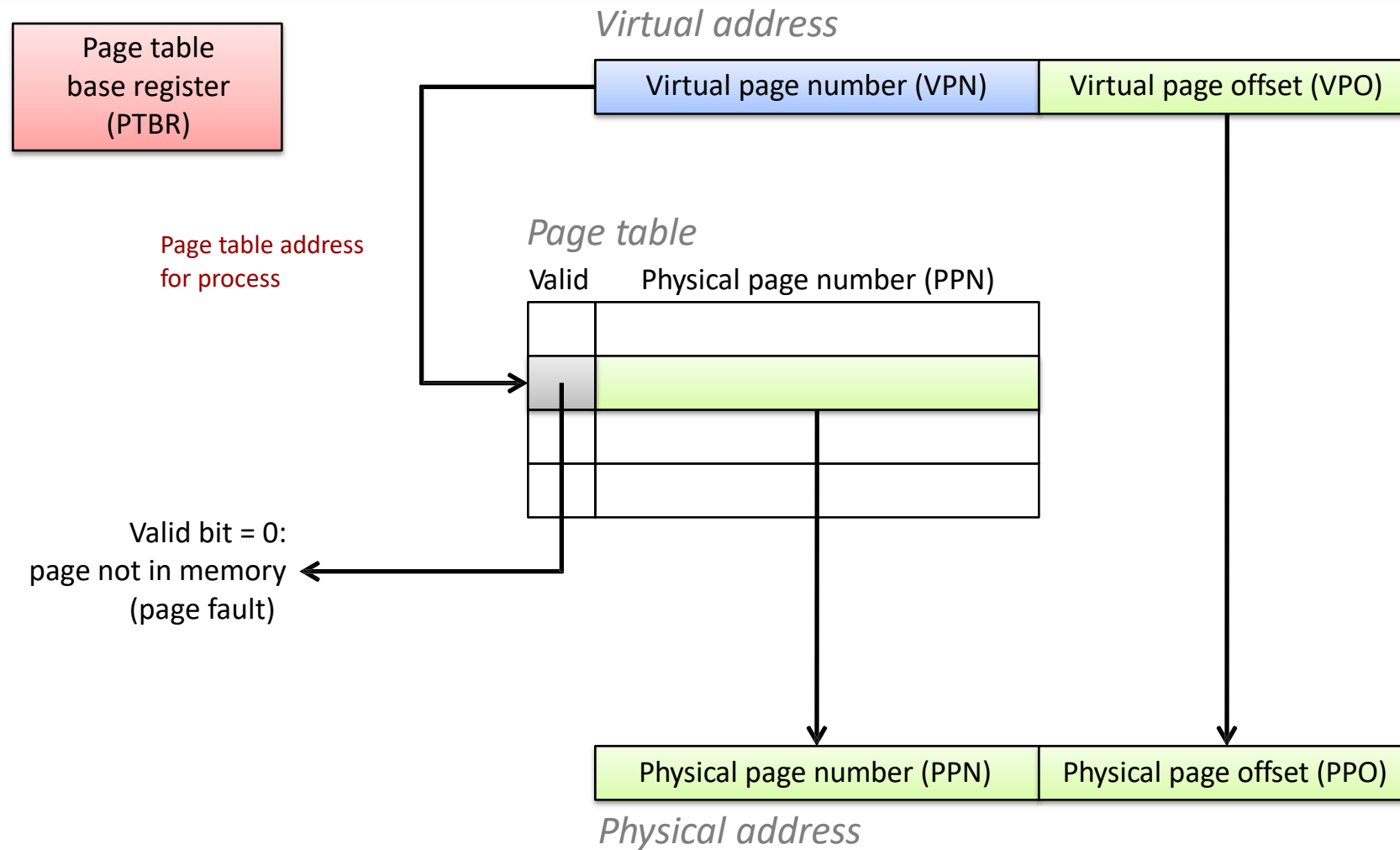


# Translatarea adreselor: Tabela de pagini

- Tabela de pagini este un vector de adrese ale paginilor virtuale (page table entries sau PTE) care stabilește corespondența cu paginile fizice. În acest exemplu avem 8 pagini virtuale.



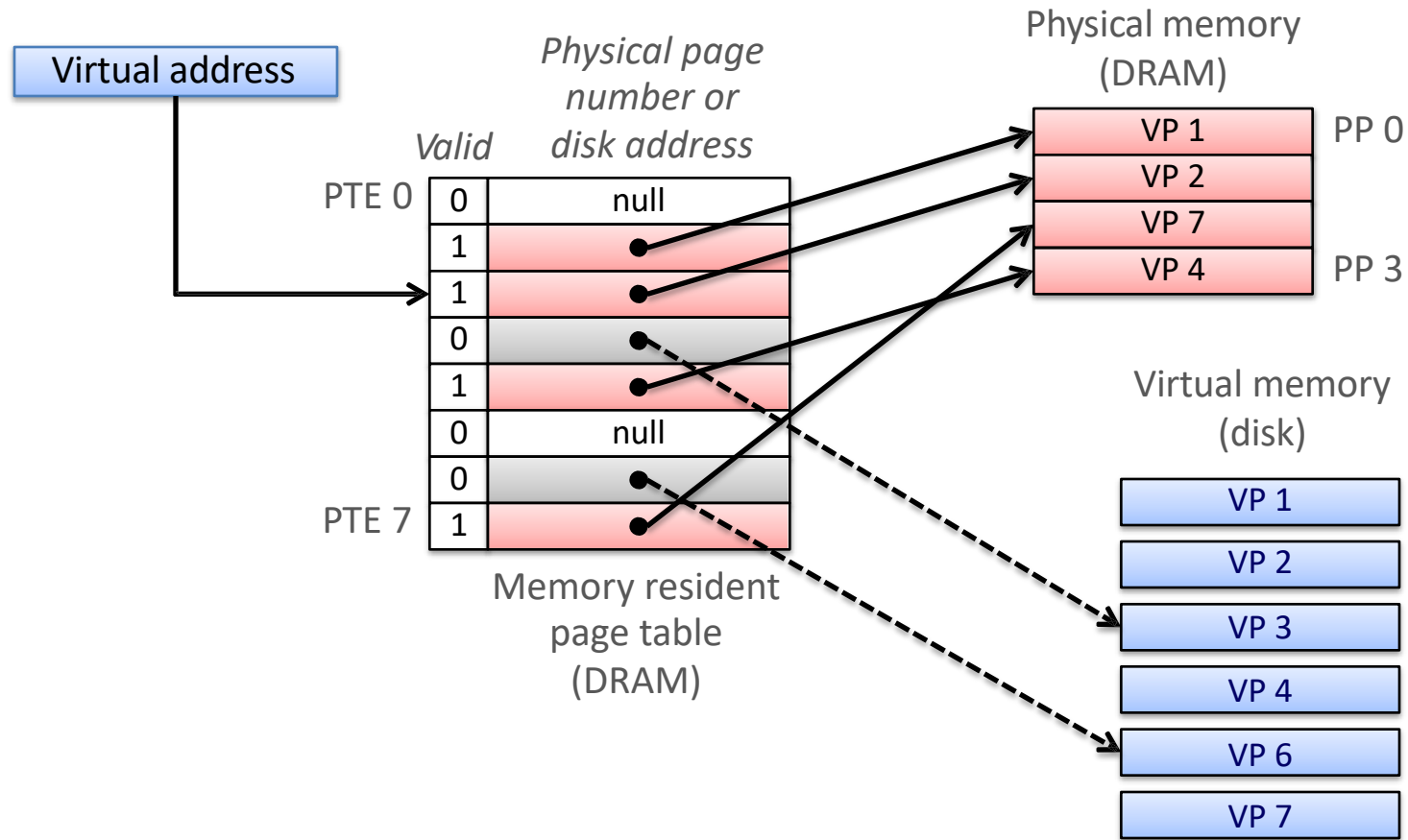
# Translatarea adreselor cu Tabela de Pagini





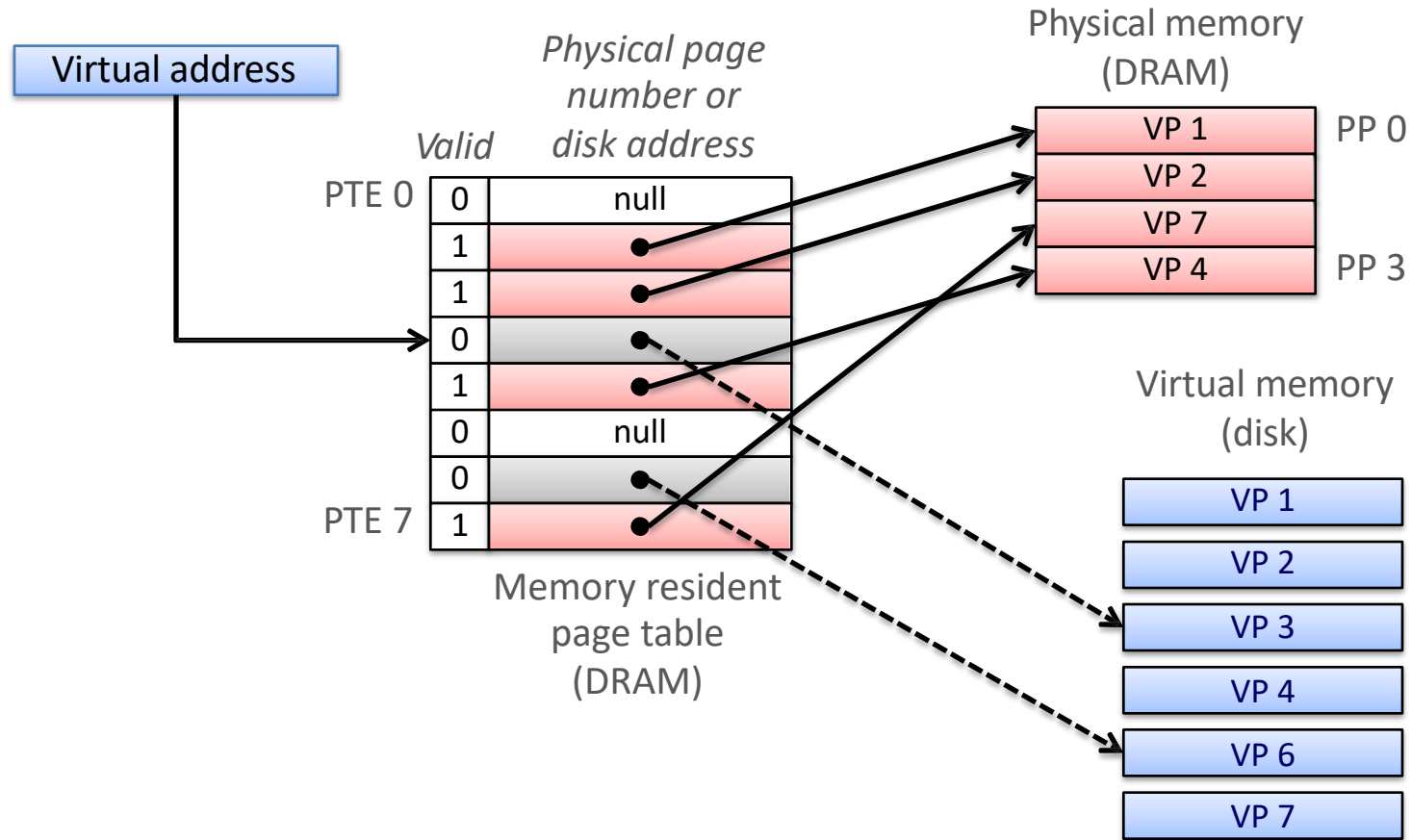
# Page Hit

- *Page hit*: referință la un cuvânt din VM care există în memoria fizică



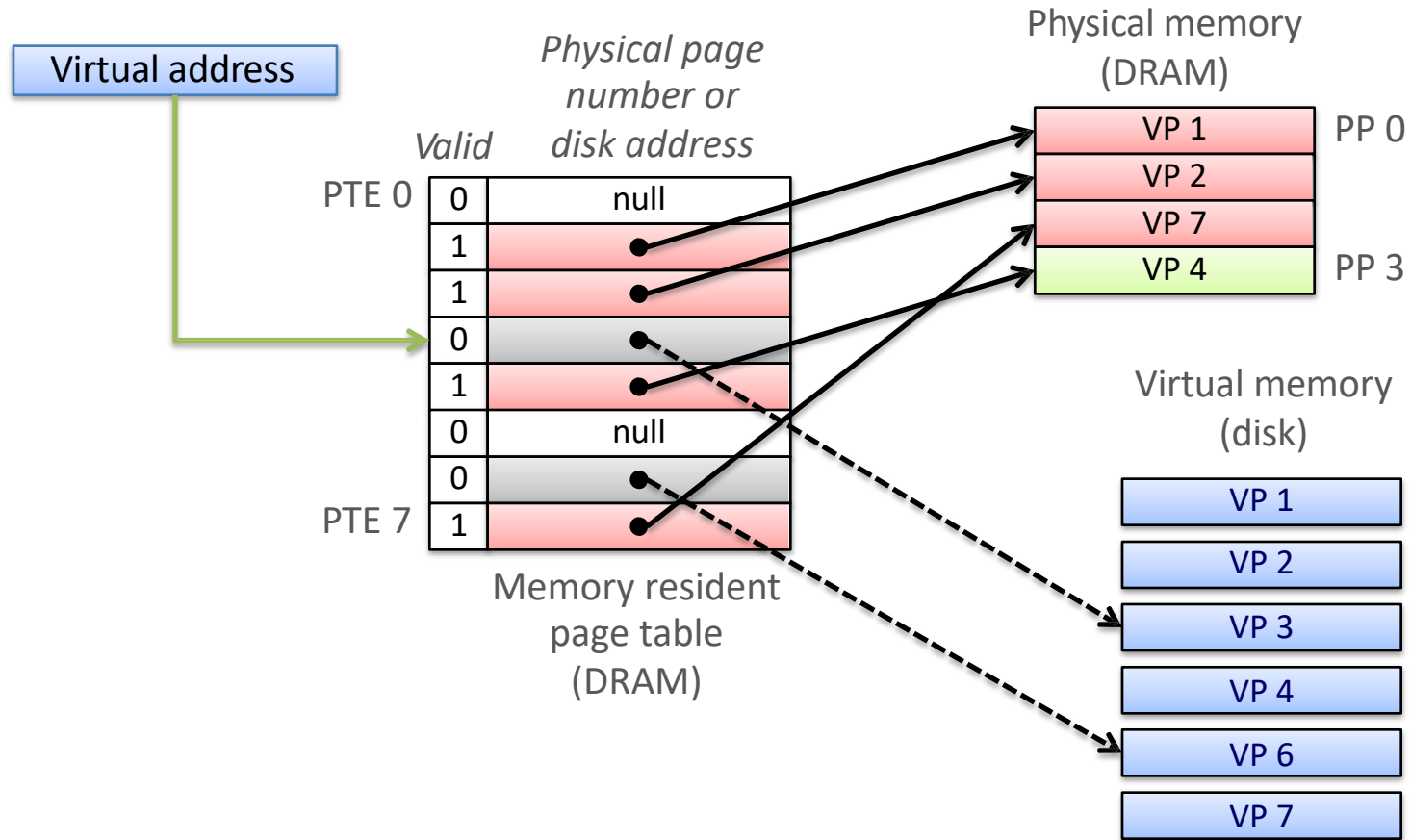
# Page Miss

- *Page miss*: referință la un cuvânt din VM care nu există în memoria fizică



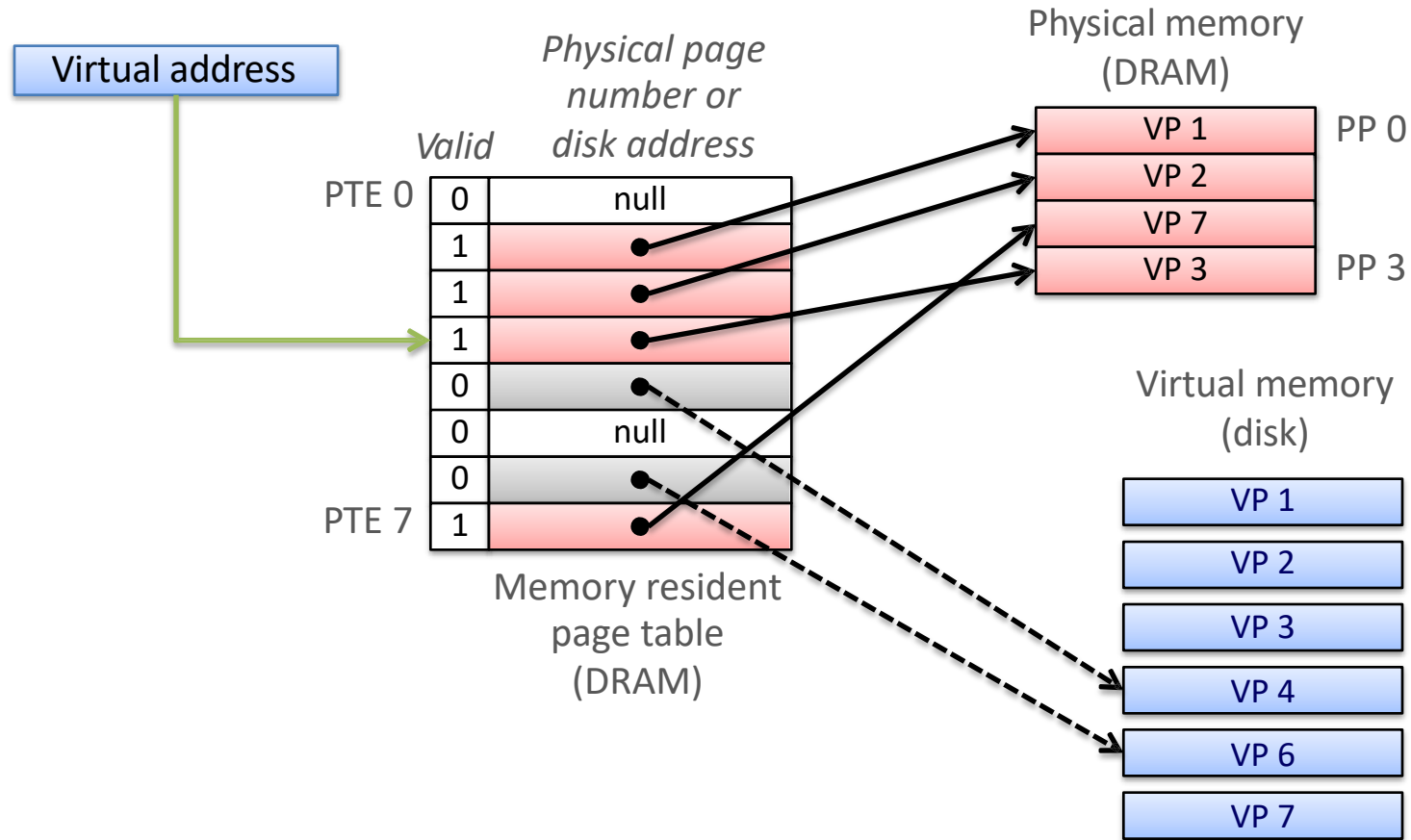
# Page Fault Handling

- Page miss cauzează un page fault (o excepție)
- Handler-ul de Page Fault selectează o victimă pentru a fi evacuată (aici VP 4)



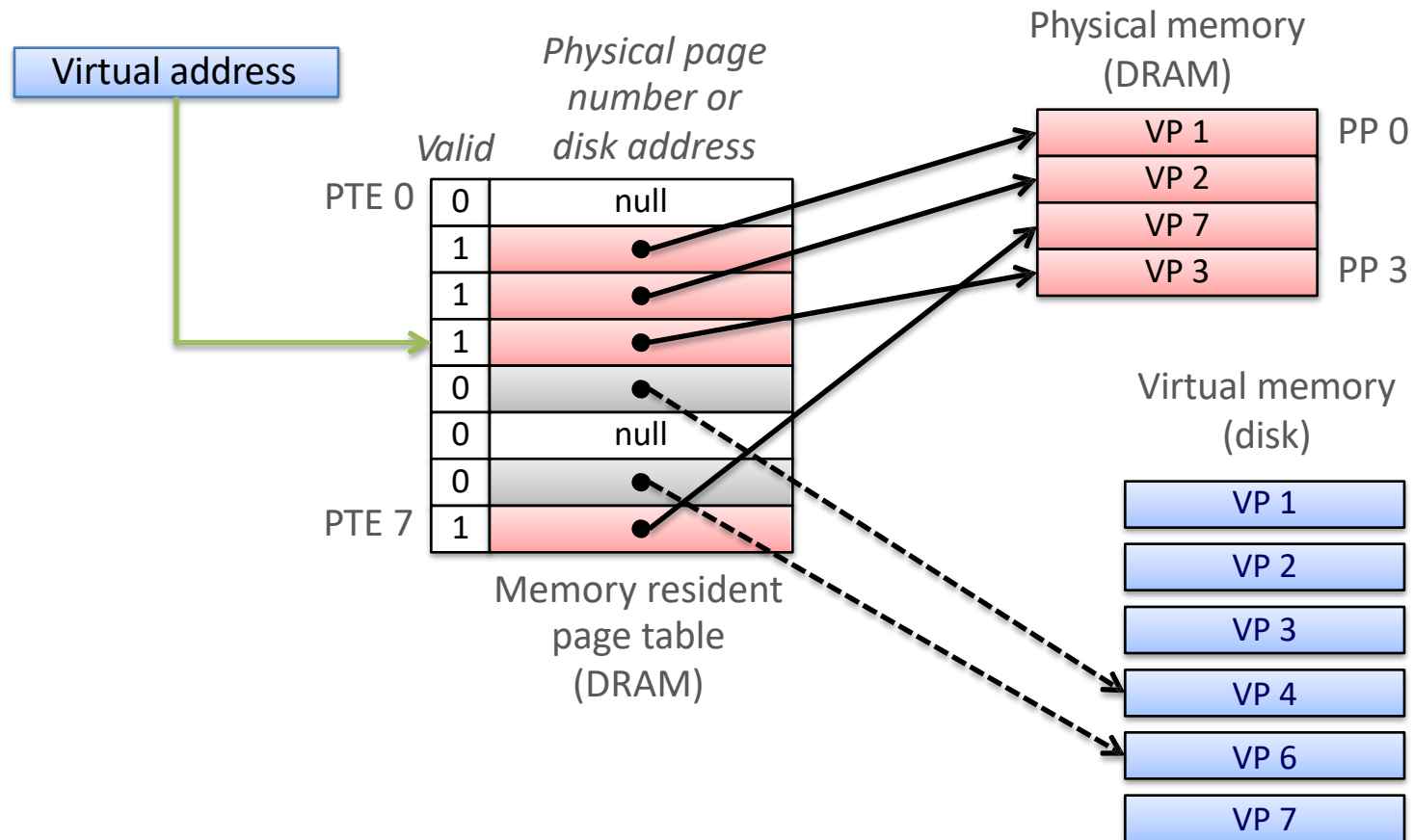
# Page Fault Handling

- Page miss cauzează un page fault (o excepție)
- Handler-ul de Page Fault selectează o victimă pentru a fi evacuată (aici VP 4)



# Page Fault Handling

- Handler-ul de Page Fault selectează o victimă pentru a fi evacuată (aici VP 4)
- Instrucțiunea care a cauzat excepție este restartată: page hit!



# De ce funcționează? Localitatea datelor

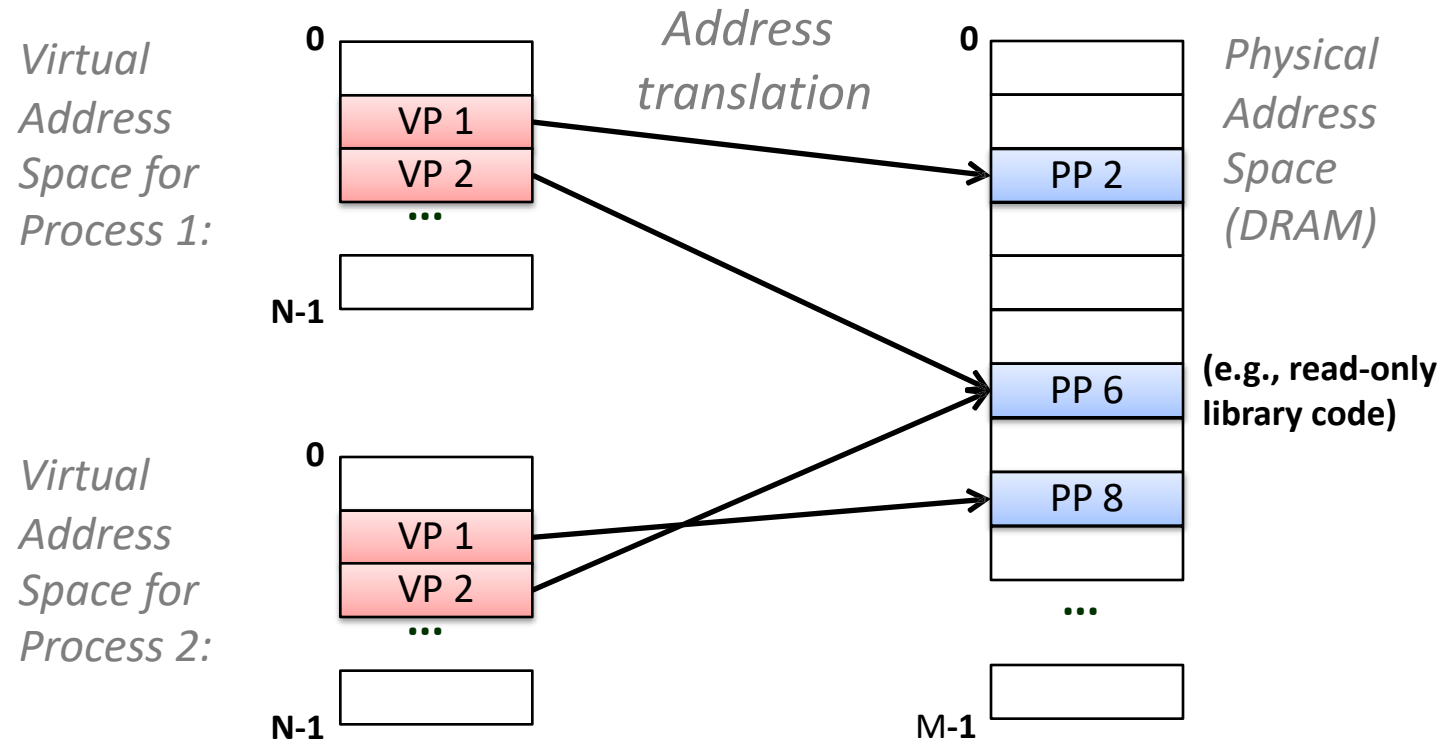
---

- Memoria virtuală funcționează din cauza localității
- În orice moment de timp, programele tind să acceseze un set de pagini virtuale active, numit și *setul de lucru* (*working set*)
  - Programele care au localitate temporală bună, vor avea și seturi de lucru mai mici
- Dacă ( $\text{working set size} < \text{main memory size}$ )
  - Performanță bună pentru un proces după compulsory miss
- Dacă ( $\text{SUM}(\text{working set sizes}) > \text{main memory size}$ )
  - *Thrashing*: Degradarea performanței când facem swap la pagini (le copiem) în continuu din memoria virtuală în fizică și invers



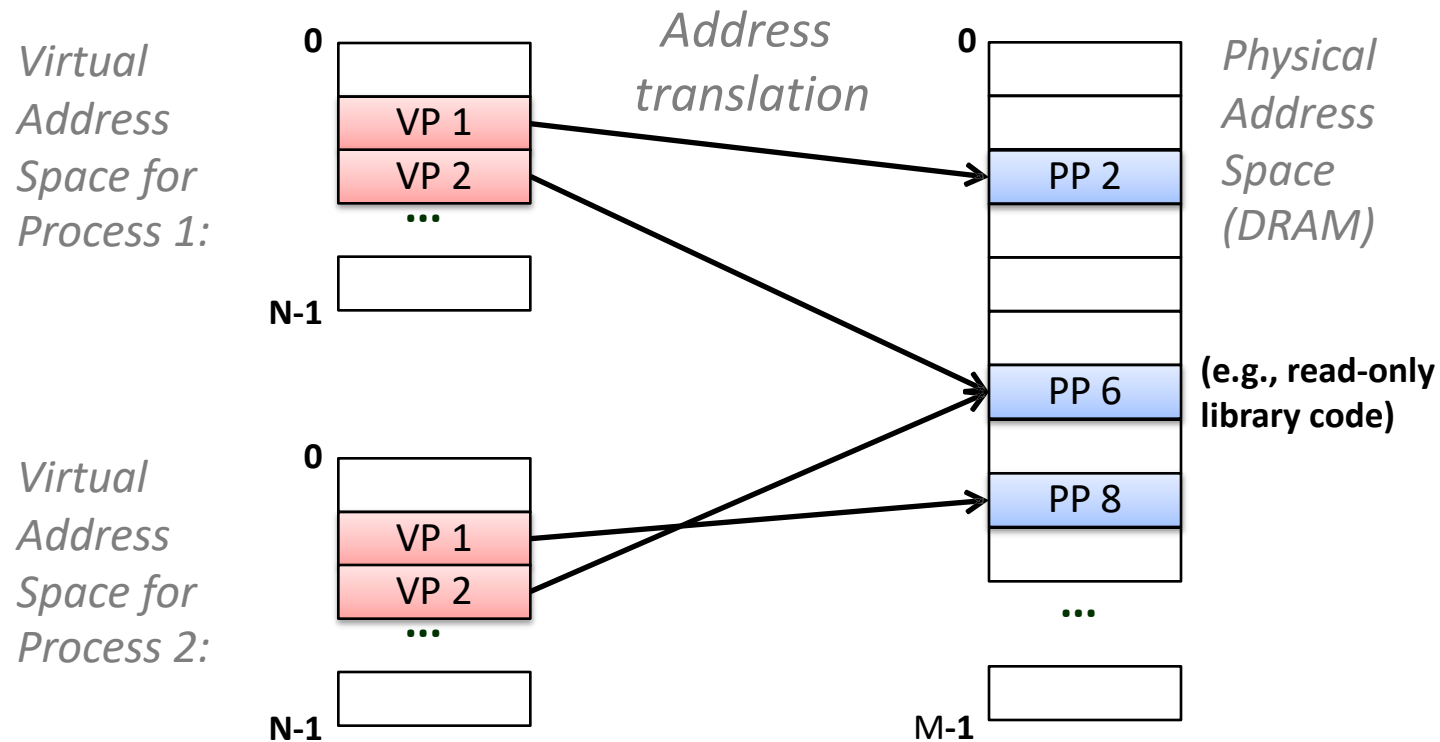
# VM ca suport pentru Memory Management

- Idee de bază: fiecare proces are propriul spațiu de adrese virtuale
  - Poate să "vadă" memoria ca un spațiu liniar
  - Funcția de mapare împrăștie adresele prin memoria fizică
    - O mapare bine aleasă simplifică alocarea și managementul memoriei



# VM ca suport pentru Memory Management

- Alocarea memoriei
  - Fiecare pagină virtuală poate fi mapată în orice pagină fizică
  - O pagină virtuală poate fi stocată în pagini fizice diferite la momente diferite de timp
- Partajarea de cod și de date între procese
  - Maparea paginilor virtuale la aceeași pagină fizică (aici: PP 6)





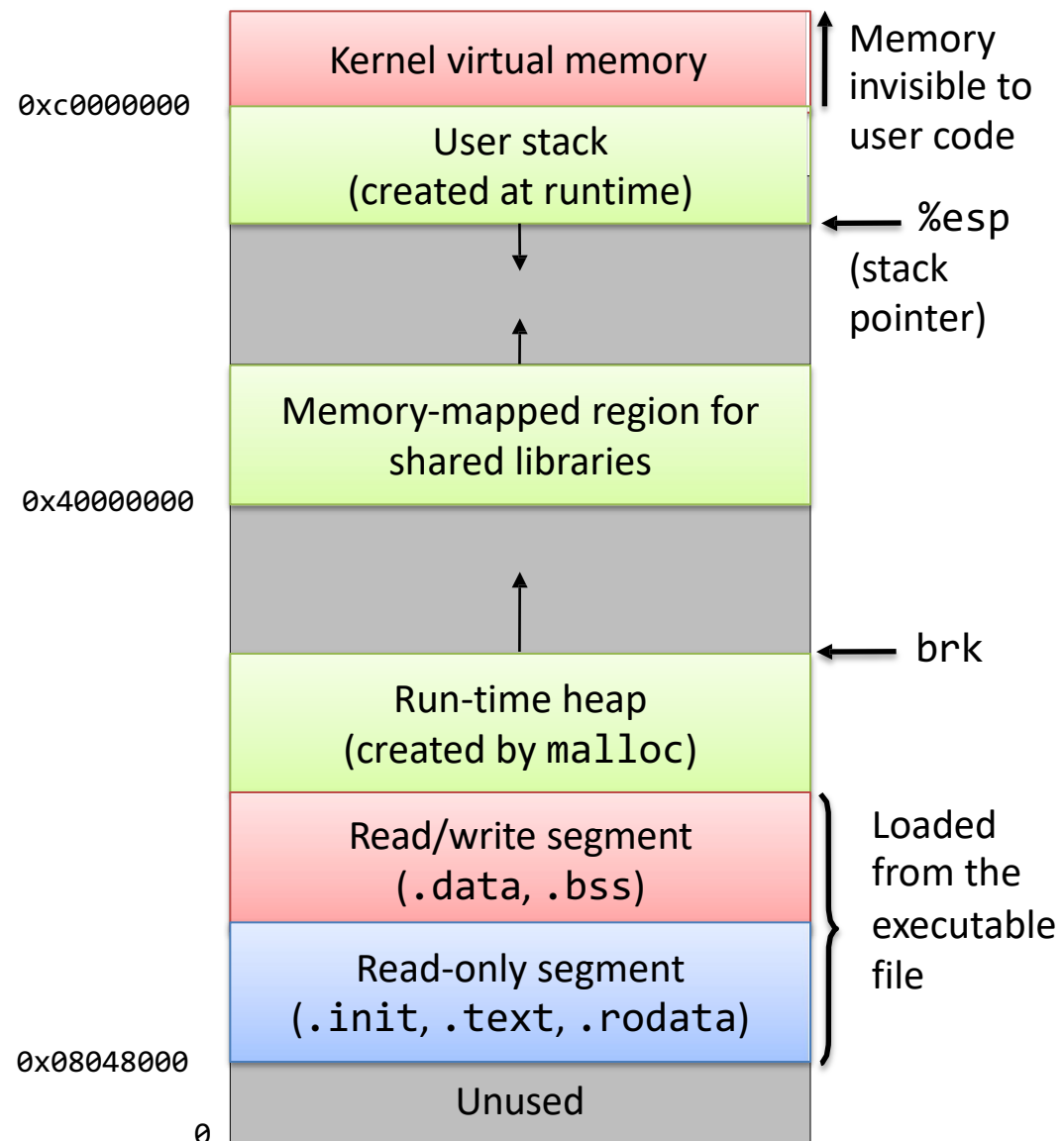
# Simplifică Linking și Loading

- Linking

- Fiecare program are un spațiu virtual de adrese similar
- Codul, stiva și bibliotecile partajate încep întotdeauna de la aceleași adrese

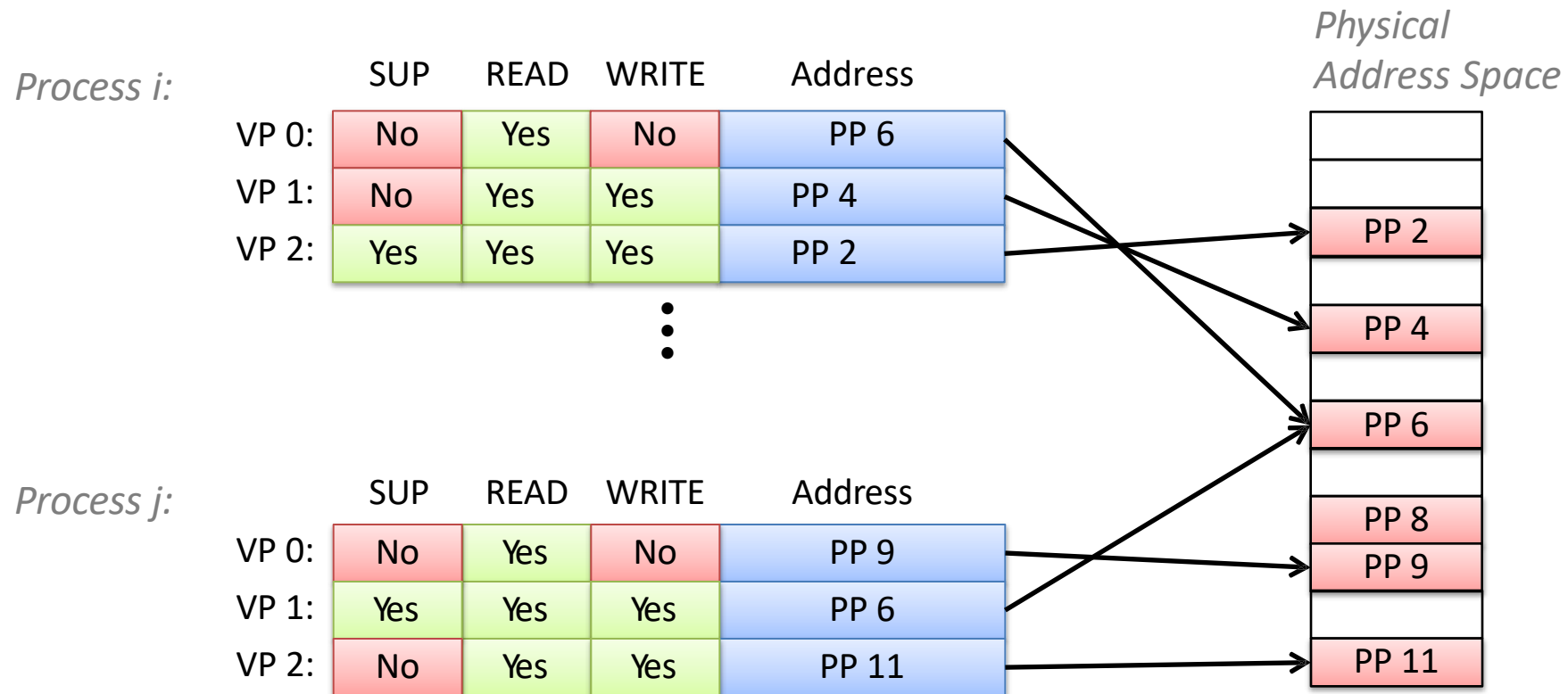
- Loading

- `execve()` aloacă pagini virtuale pentru secțiunile `.text` și `.data` = creează PTE-uri marcate invalid
- Secțiunile `.text` și `.data` sunt copiate, pagină cu pagină, la cererea sistemului de memorie virtuală

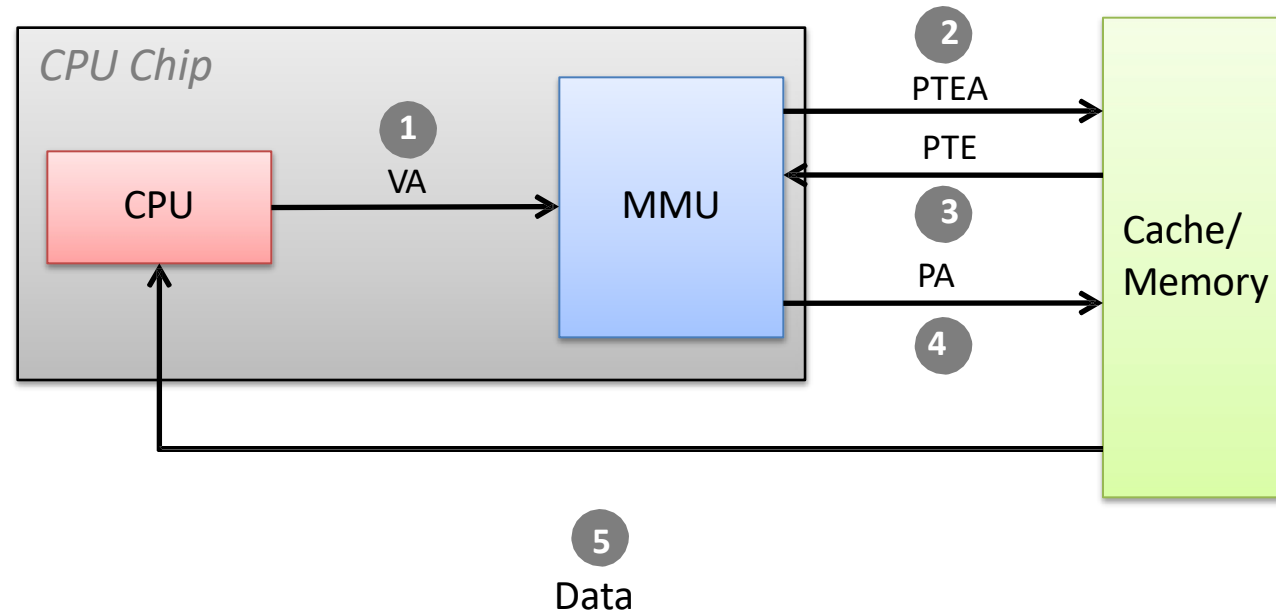


# VM ca utilitar pentru protecția memoriei

- Extinde PTE cu biți pentru permisiuni
- Page fault handler verifică acești biți înainte de remapare
  - Dacă sunt corupți, trimite SIGSEGV (segmentation fault)

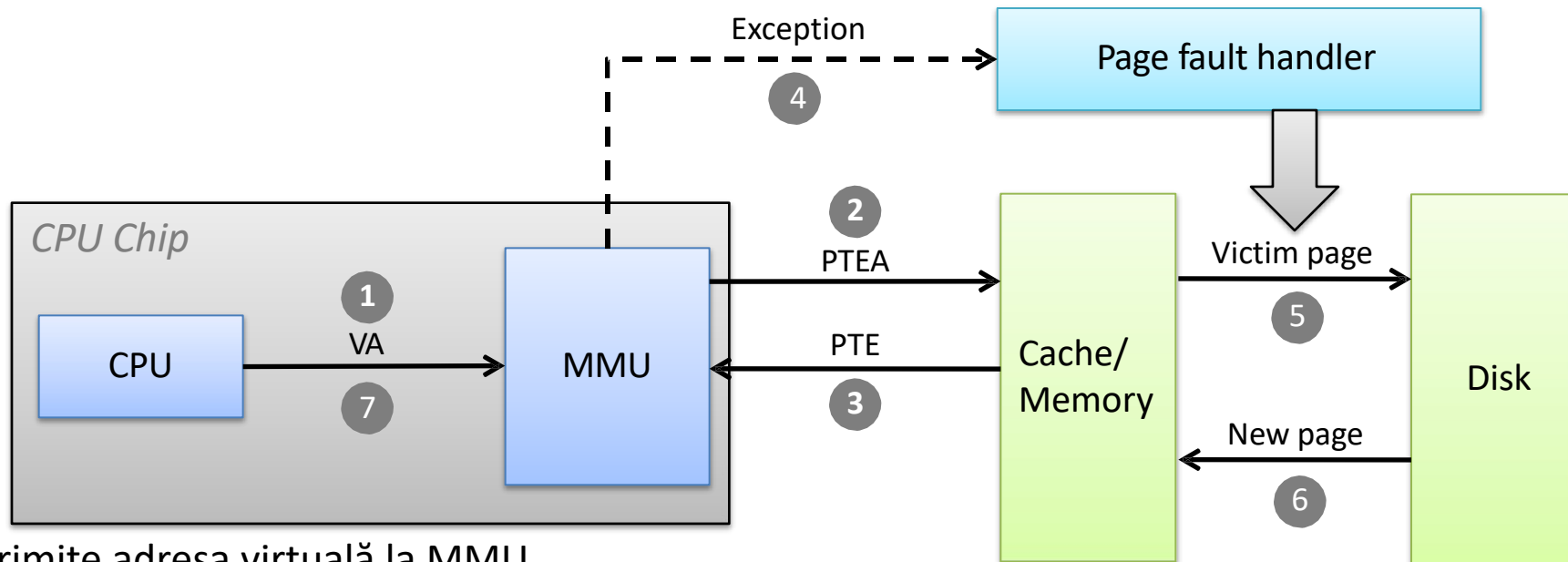


# Translatarea adreselor: Page Hit



- 1) Procesorul trimite adresa virtuală la MMU
- 2-3) MMU face fetch la PTE din tabela de pagini în memorie
- 4) MMU trimite adresa fizică la cache/memorie
- 5) Cache/memoria trimite cuvântul de date la procesor

# Translatarea adreselor: Page Fault



- 1) Procesorul trimite adresa virtuală la MMU
- 2-3) MMU face fetch la PTE din tabela de pagini în memorie
- 4) Valid bit este zero, MMU declanșează page fault exception
- 5) Handler-ul identifică victima (și, dacă e "murdară", o pagează pe disc)
- 6) Handler-ul aduce o nouă pagină și actualizează PTE în memorie
- 7) Handler-ul face return la procesul original, restartând instrucțiunea care a generat excepția

**PROFUL  
A ZIS CĂ NE  
TRECE PE TOȚI**

---

**...PRIN  
TOATĂ MATERIA  
PENTRU EXAMEN**



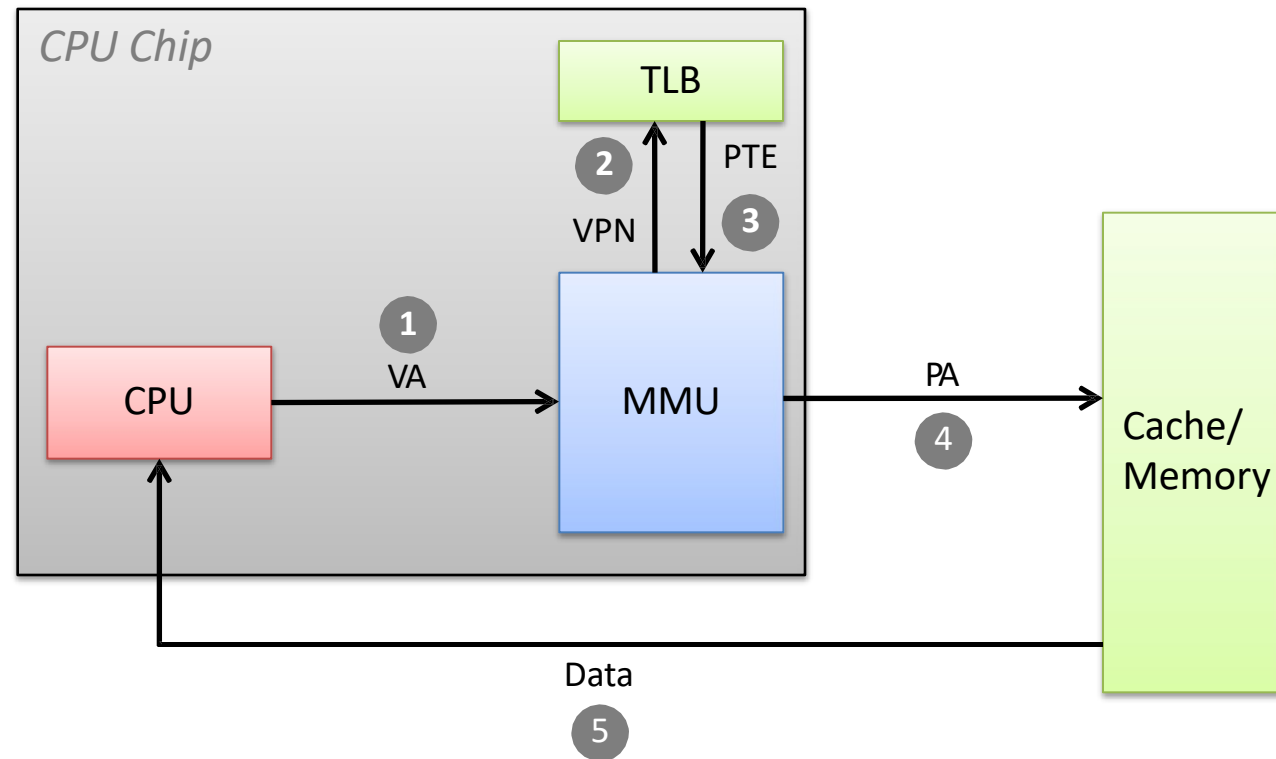
# Facilitarea tranzacțiilor prin TLB

---

- Page table entries (PTEs) sunt stocate în cache L1 ca orice alt cuvânt din memorie
  - PTE-urile pot fi invalidate din cache ca orice alte referințe la date
  - PTE hit în L1 tot necesită o întârziere de 1 (sau 2) cicli
- Soluție: *Translation Lookaside Buffer* (TLB)
  - Cache hardware de mici dimensiuni în MMU
  - Mapează numerele de pagini virtuale la numere corespunzătoare de pagini fizice
  - Conține tabele de pagini întregi pentru un număr mic de pagini

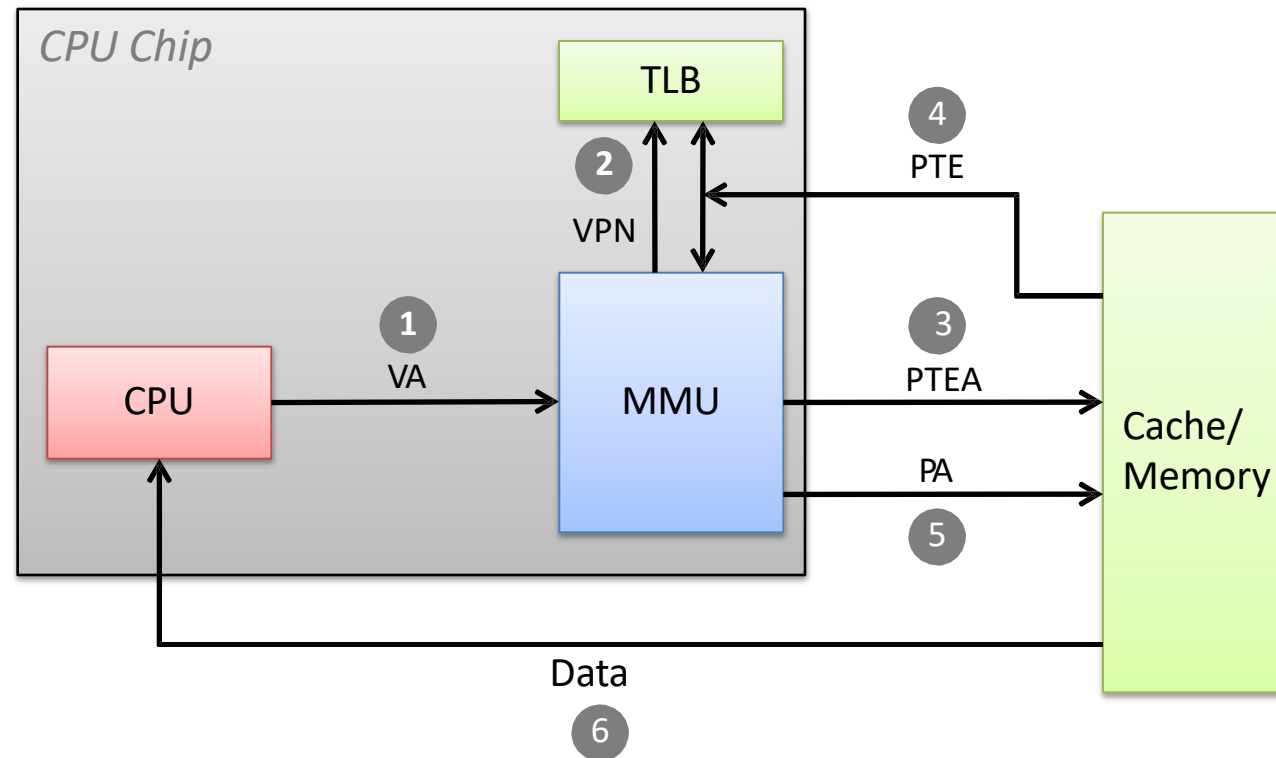


# TLB Hit



**TLB hit elimină necesitatea unui acces la memorie**

# TLB Miss



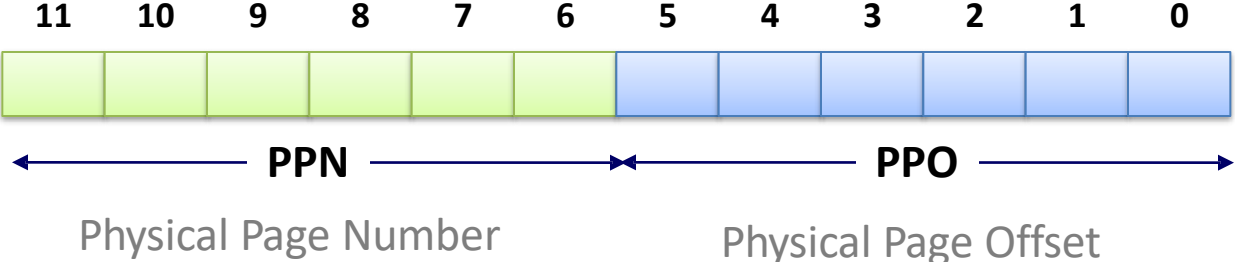
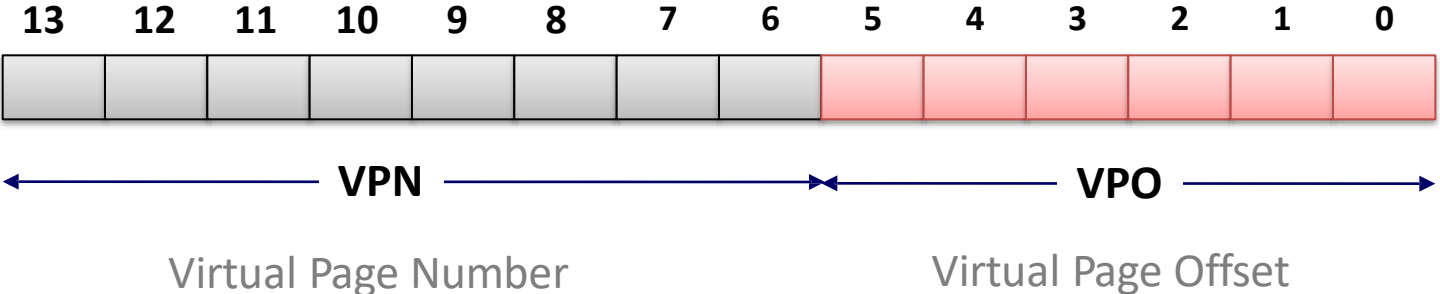
**TLB miss produce un acces suplimentar la memorie  
(pentru a aduce un PTE)**

Din fericire, TLB miss sunt rare



# Exemplu simplu

- Adresare
  - Adrese virtuale pe 14-biți
  - Adrese fizice pe 12-biți
  - Page size = 64 bytes



# Exemplu de tabelă de pagini

---

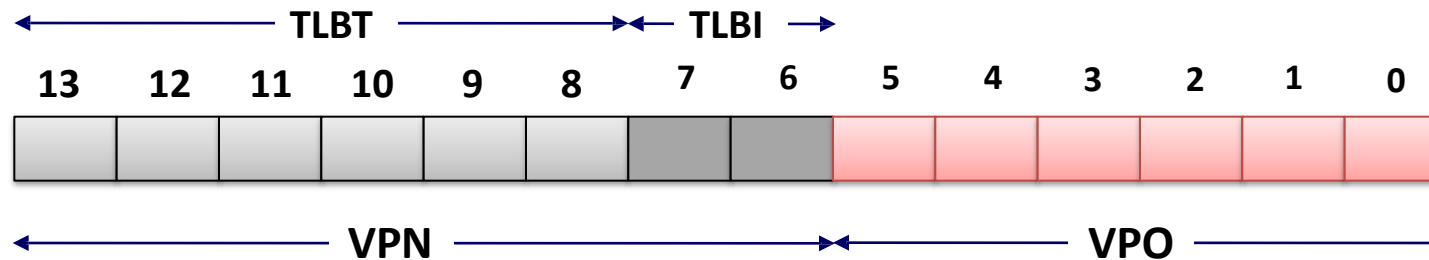
Doar primele 16 intrări (din 256)

<i>VPN</i>	<i>PPN</i>	<i>Valid</i>
00	28	1
01	–	0
02	33	1
03	02	1
04	–	0
05	16	1
06	–	0
07	–	0

<i>VPN</i>	<i>PPN</i>	<i>Valid</i>
08	13	1
09	17	1
0A	09	1
0B	–	0
0C	–	0
0D	2D	1
0E	11	1
0F	0D	1

# Sistem simplu de memorie cu TLB

- 16 intrări
- 4-way asociative

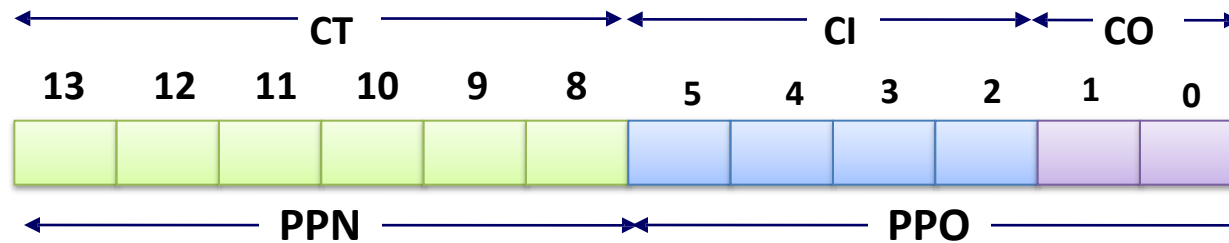


Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	–	0	09	0D	1	00	–	0	07	02	1
1	03	2D	1	02	–	0	04	–	0	0A	–	0
2	02	–	0	08	–	0	06	–	0	03	–	0
3	07	–	0	03	0D	1	0A	34	1	02	–	0



# Cache pentru sistemul simplu de memorie

- 16 linii, 4 octeți per bloc
- Adresare fizică
- Mapat direct



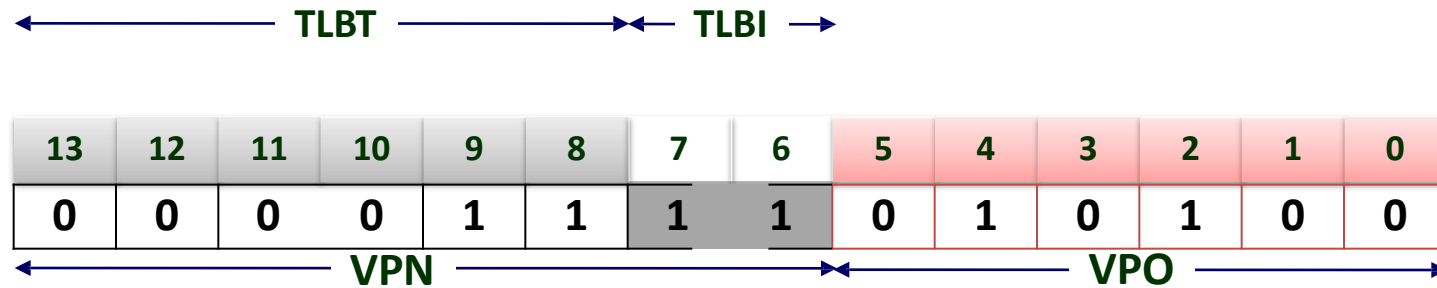
<i>Idx</i>	<i>Tag</i>	<i>Valid</i>	<i>B0</i>	<i>B1</i>	<i>B2</i>	<i>B3</i>
0	19	1	99	11	23	11
1	15	0	–	–	–	–
2	1B	1	00	02	04	08
3	36	0	–	–	–	–
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	–	–	–	–
7	16	1	11	C2	DF	03

<i>Idx</i>	<i>Tag</i>	<i>Valid</i>	<i>B0</i>	<i>B1</i>	<i>B2</i>	<i>B3</i>
8	24	1	3A	00	51	89
9	2D	0	–	–	–	–
A	2D	1	93	15	DA	3B
B	0B	0	–	–	–	–
C	12	0	–	–	–	–
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	–	–	–	–



# Exemplu de translatare de adrese

Adresă virtuală: **0x03D4**



VPN **0x0F**

TLBI **3**

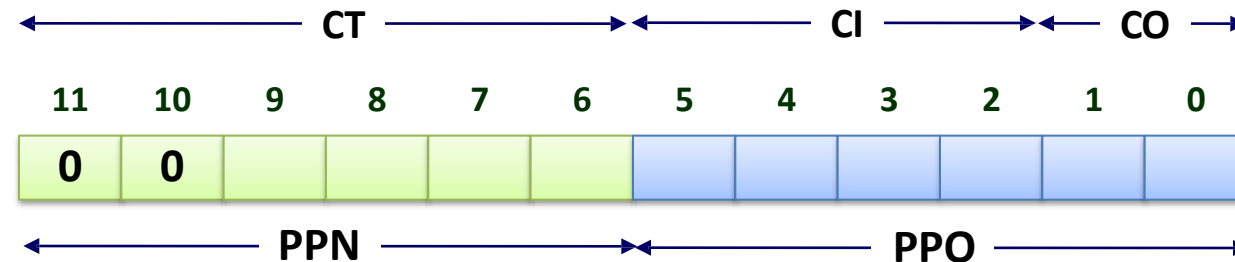
TLBT **0x03**

TLB Hit? **Y**

Page Fault? **N**

PPN: **0x0D**

Adresă fizică



CO **0**

CI **0x5**

CT **0x0D**

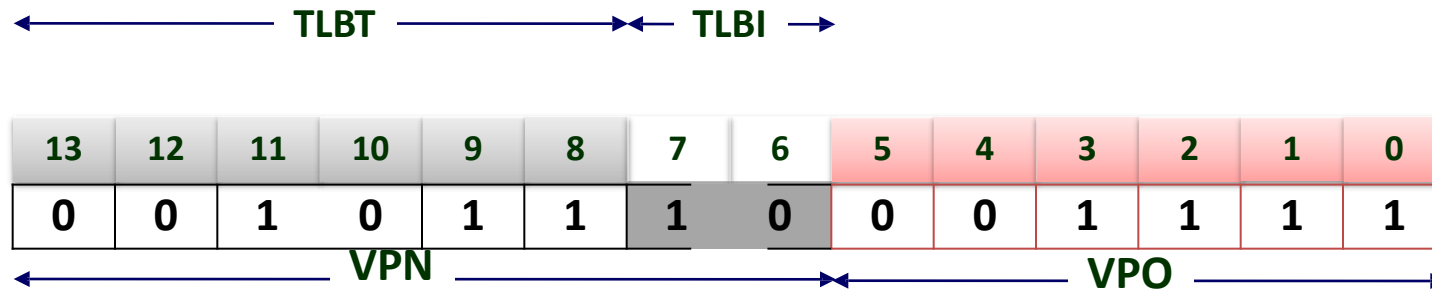
Hit? **Y**

Byte: **0x36**



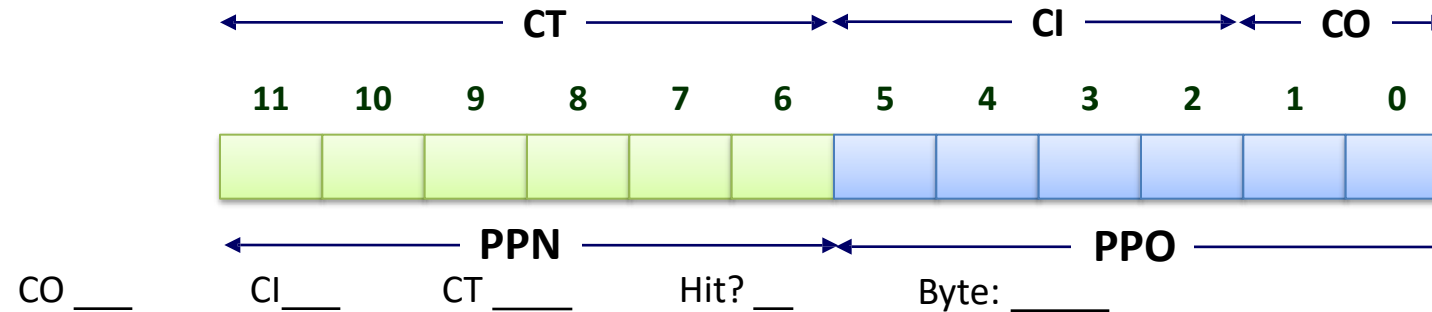
# Alt exemplu

Adresă virtuală: **0x0B8F**



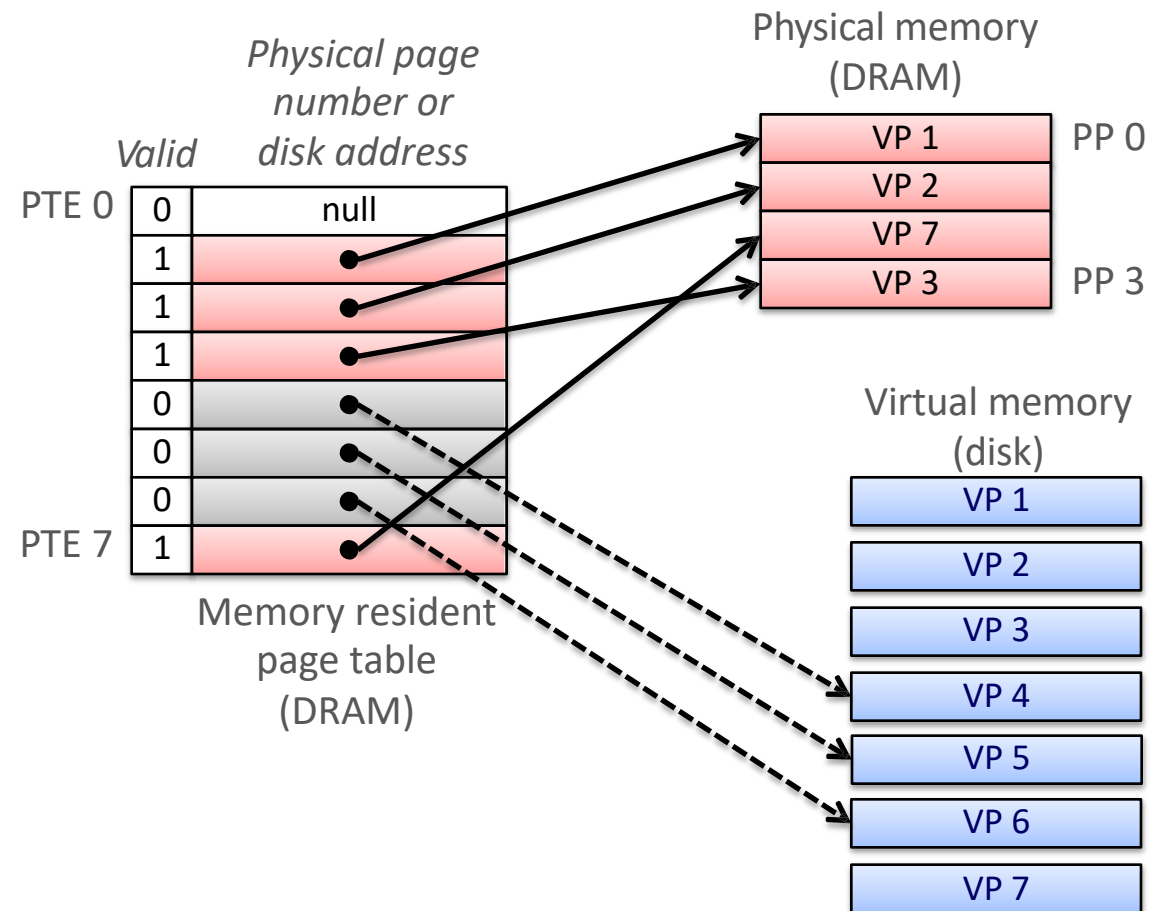
VPN **0x2E**    TLBI **2**    TLBT **0x0B**    TLB Hit? **N**    Page Fault? **Y**    PPN: **TBD**

Adresa fizică

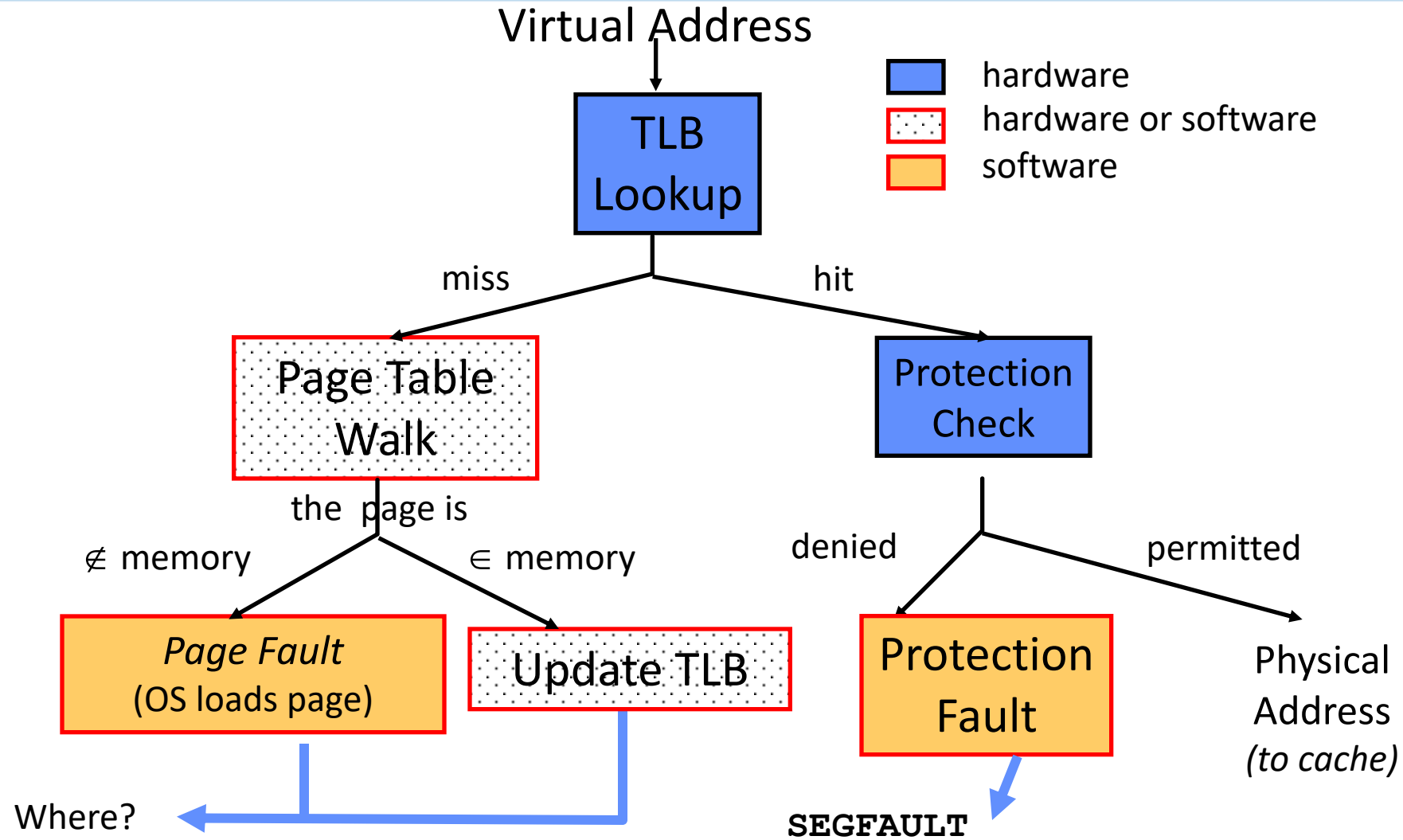


# Alocarea paginilor virtuale

- Exemplu: alocarea VP 5
- Kernel-ul alocă VP 5 pe disc și pointează PTE 5 spre ea



# Translatarea adreselor: *putting it all together*





# Politici de înlocuire a paginilor

---

Least-recently used (LRU)

Implementată prin menținerea unei stive

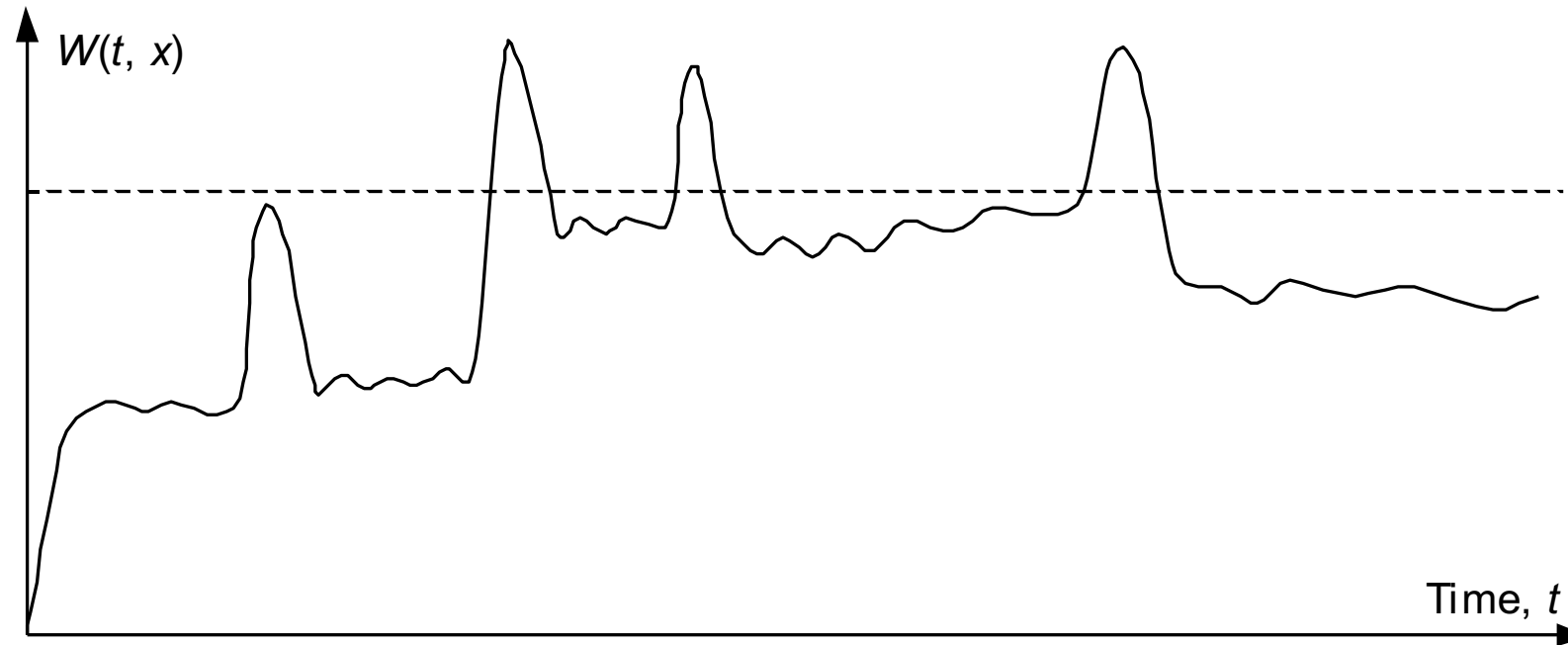
Pagini →		A	B	A	F	B	E	A
LRU stack								
MRU	D	A	B	A	F	B	E	A
	B	D	A	B	A	F	B	E
	E	B	D	D	B	A	F	B
LRU	C	E	E	E	D	D	A	F



# Memoria principală și memoria de masă

Definim setul de lucru al unui proces (working set),  $W(t, x)$ : Setul de pagini accesat de ultimele  $x$  instrucțiuni la timpul  $t$

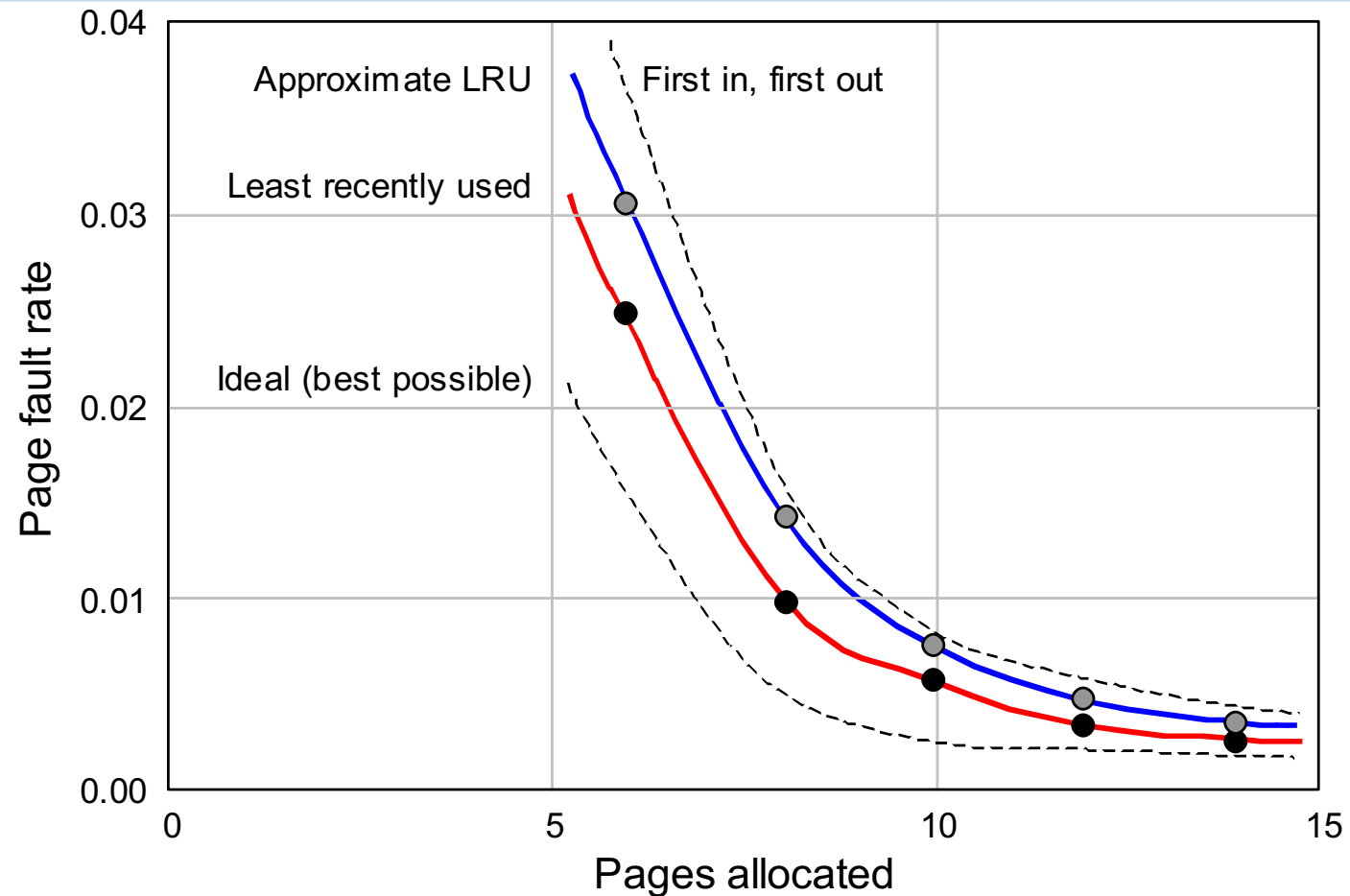
Localitatea datelor asigură faptul că setul de lucru se modifică lent



Variația în dimensiune a setului de lucru pentru un program



# Impactul politicii de înlocuire asupra performanței



Rata de page fault în funcție de numărul de pagini locat și de politica de înlocuire a paginilor



# Memoria virtuală - concluzii

---

- Memoria virtuală mărește **capacitatea**
- Avem un subset de pagini virtuale în memoria fizică
- **Tabela de pagini** mapează paginile virtuale pe pagini fizice – traducere de adrese
- **TLB** mărește viteza cu care se fac translațiile adreselor
- Tabele de pagini diferite pentru programe diferite asigură **protecția memoriei**

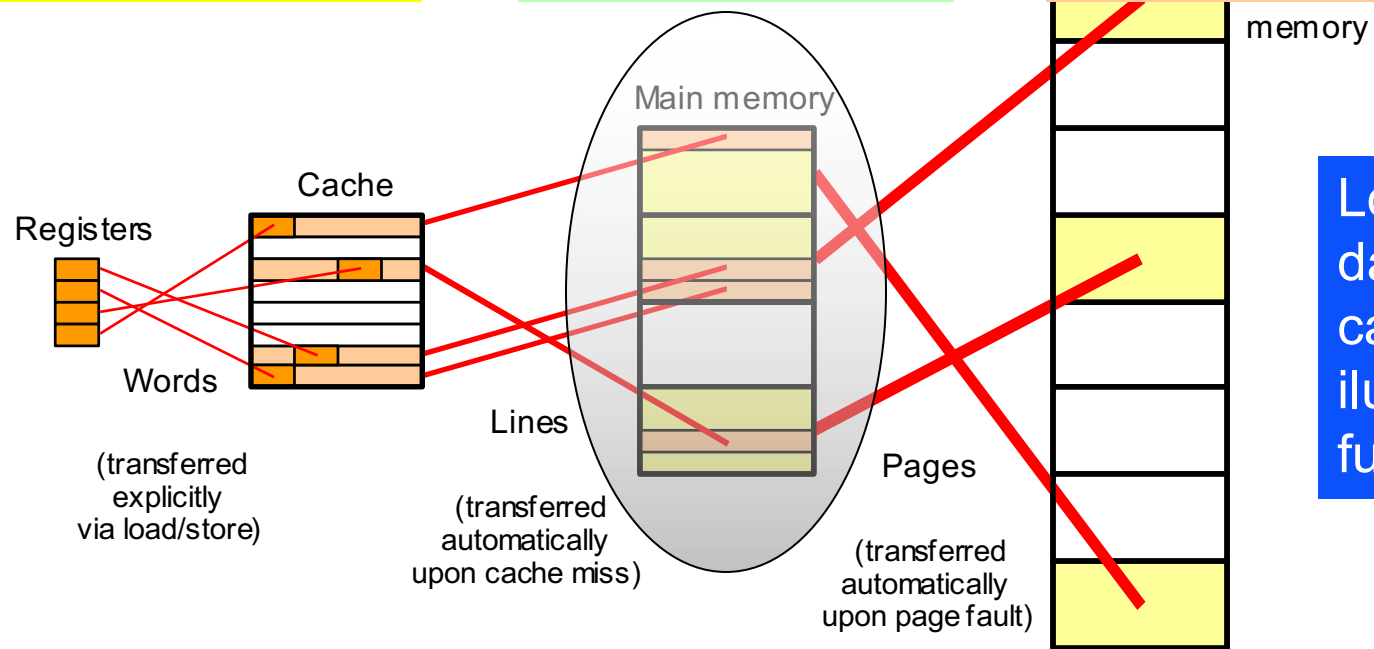


# Ierarhia memoriilor - concluzii

Memoria cache:  
dă iluzia unei  
viteze mari de  
execuție

Memoria  
principală: cost  
rezonabil, dar  
lentă și mică

Memoria virtuală:  
dă iluzia unei  
dimensiuni foarte  
mari a memoriei



Localitatea  
datelor face  
ca toate  
iluziile să  
funcționeze

Fluxul de date într-o ierarhie de memorii



# Sistemul de memorii - rezumat

---

- L1/L2 Cache
  - Există doar pentru a accelera execuția
  - Comportamentul este invizibil programatorului de aplicații și (unei mari părți a) sistemului de operare
  - Implementate în întregime în hardware
- Memoria virtuală
  - Este fundamentul multor funcții ale SO
    - Process creation, task switching, protection
  - Software
    - Alocă/partajează memoria fizică între procese
    - Construiește tabele care urmăresc tipul memoriei, sursele, partajarea
    - Exception handling. Completează tabelele de mapare
  - Hardware
    - Translatează adresele virtuale via tabelele de mapare, impune permisiuni
    - Acelerează maparea prin intermediul TLB



**9:45AM**



**STUDENTII ÎNCEP TESTUL**

# Acknowledgements

---

- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)
- MIT material derived from course 6.823
- UCB material derived from course CS252

