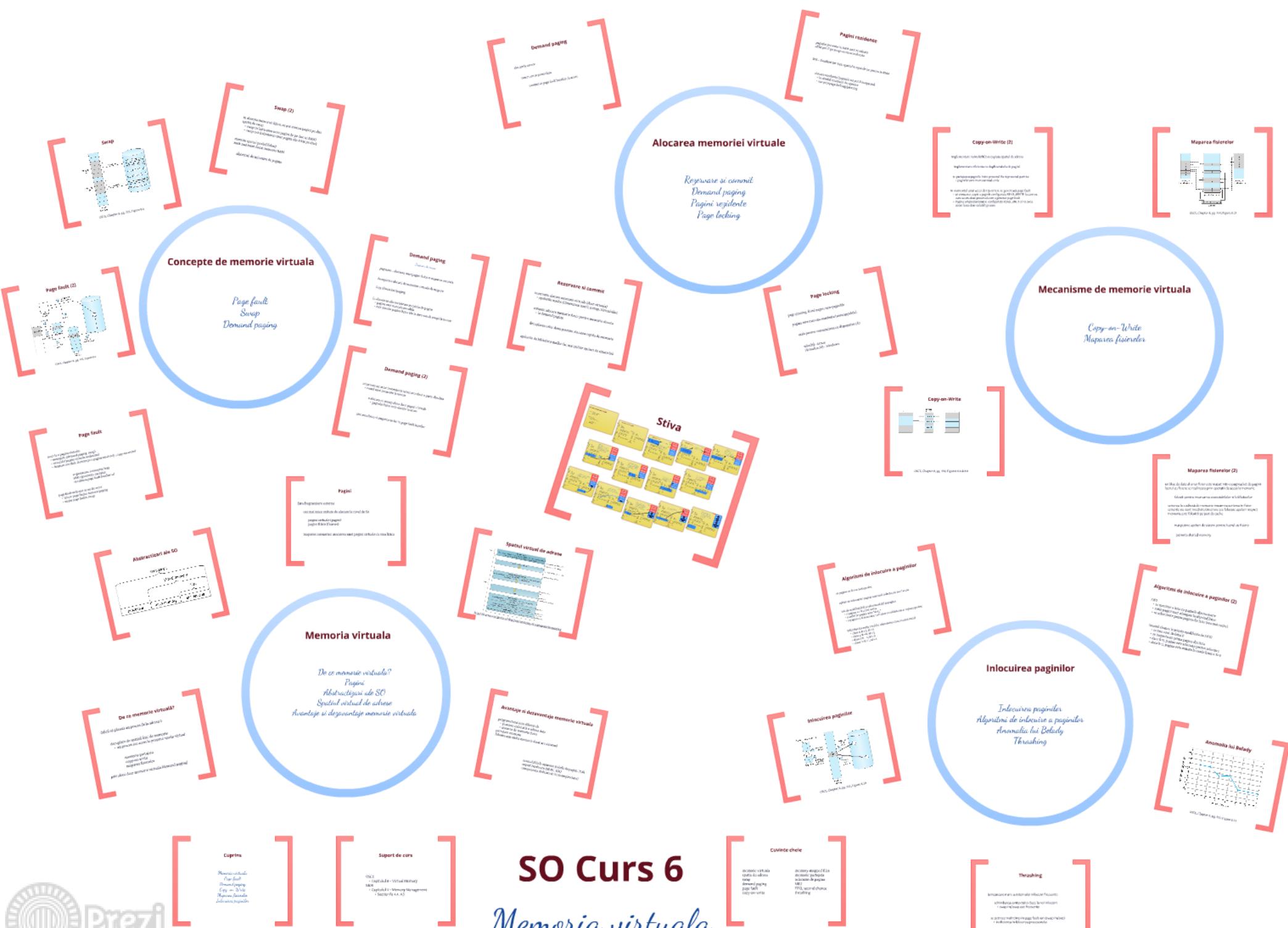


SO Curs 6

Memoria virtuala

 Prezi



SO Curs 6

Memoria virtuala

Memoria virtuala

De ce memorie virtuală?
Păreri
Abstracții ale SO
Spațiu virtual de adresă
Avantaje și dezavantaje memorie virtuală

Cuprins

Memoria virtuală

De ce memorie virtuală?

Pagini

Abstractizare de SO

Spațiu virtual de adrese

Avantaje și dezavantaje memoria virtuală

Support de cours

SO Curs 6

Memoria virtuala

Cuvinte cheie	
memorie virtuală	memorie stoped (față de adresa)
spatiu de adresa	memorie partajată
ramp	memorie de pagină
demand paging	VBU
page fault	PTO, secund chache
copy-on-write	flushing

Thrashing
Almacén de memoria virtual insuficiente
Administración de memoria: Términos



Suport de curs

OSCE

- Capitolul 8 – Virtual Memory

MOS

- Capitolul 4 – Memory Management
 - Secțiunile 4.4, 4.5

Cuprins

Memoria virtuala

Page fault

Demand paging

Copy-on-Write

Maparea fisierelor

Inlocuirea paginilor

Memoria virtuală

De ce memorie virtuală?

Pagini

Abstractizari ale SO

Spatiul virtual de adrese

Avantaje si dezavantaje memorie virtuală

De ce memorie virtuală?

dificil să plasezi un proces de la adresa 0

decuplare de spatiul fizic de memorie

- un proces are acces la propriul spatiu virtual

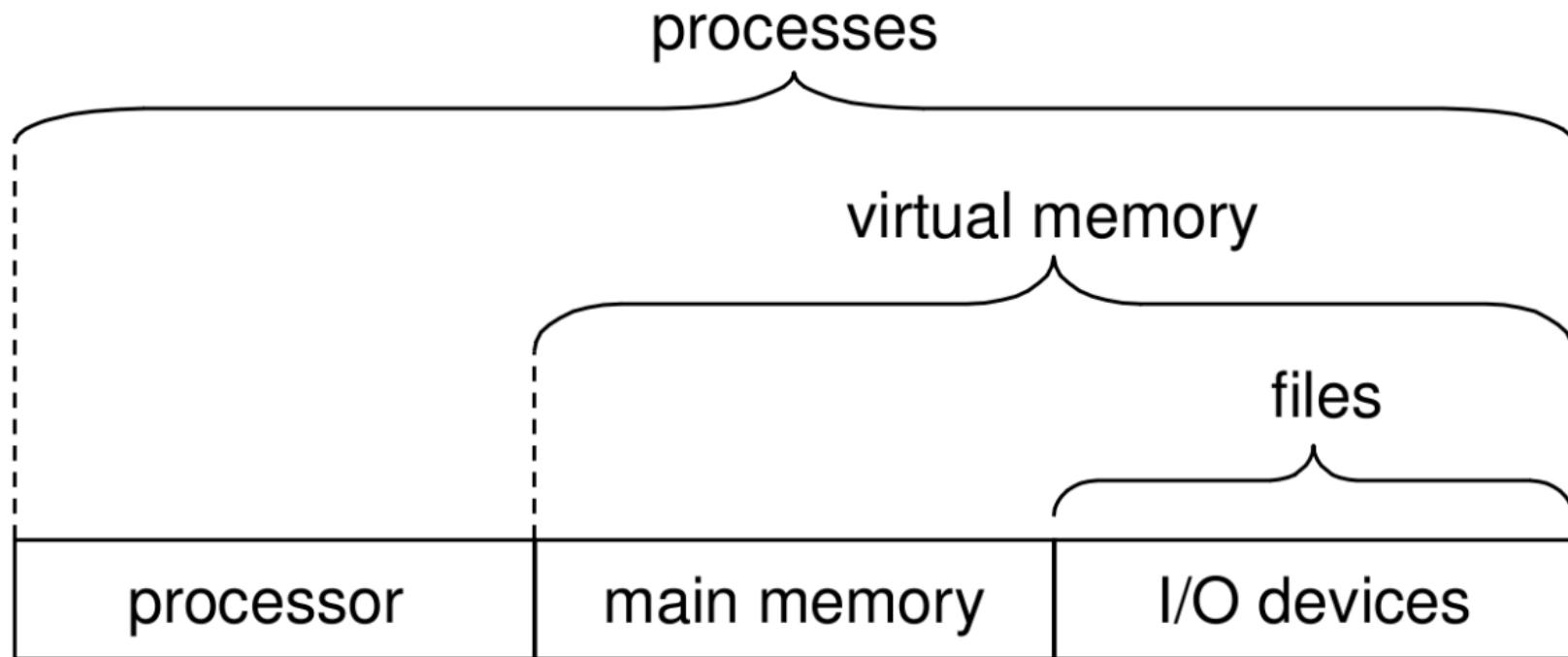
memorie partajata

copy-on-write

maparea fisierelor

poti aloca doar memorie virtuala (demand paging)

Abstractizari ale SO



Pagini

fara fragmentare externa

cea mai mica unitate de alocare la nivel de SO

pagini virtuale (pages)
pagini fizice (frames)

maparea memoriei: asocierea unei pagini virtuale cu una fizica

Avantaje si dezavantaje memorie virtuala

programatorul este eliberat de

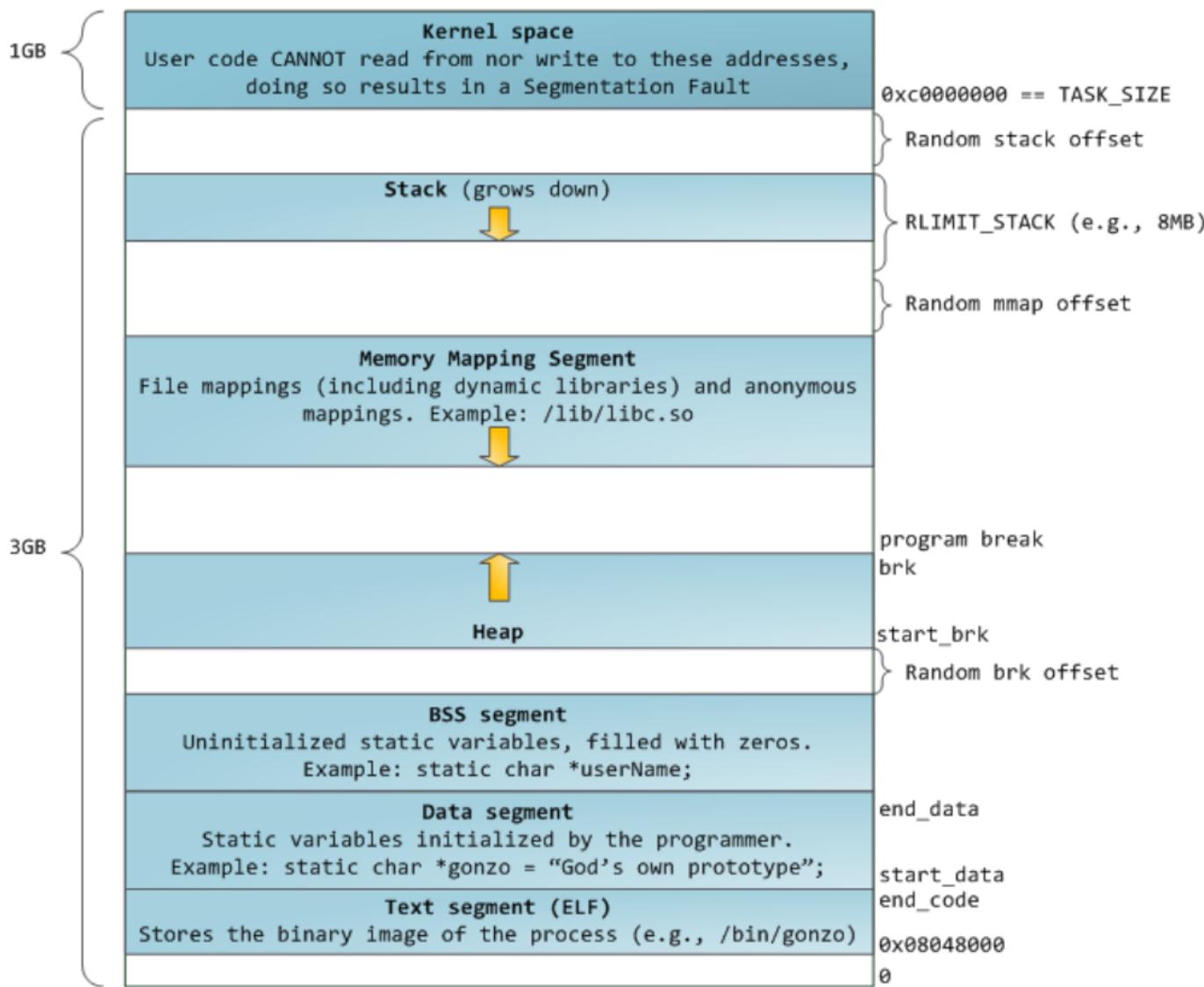
- plasarea codului la o adresa data
- alocarea de memorie fizica

partajare memorie

folosire mai multa memorie decat are sistemul

acces dublu la memorie (tabela de pagini, TLB)
suport hardware (MMU, TLB)
componenta dedicata in SO (complexitate)

Spatiul virtual de adrese



<http://duartes.org/gustavo/blog/post/anatomy-of-a-program-in-memory>

Stiva

Scriem codul la nivel inalt

```
int sum(int a, int b) {  
    return a + b;  
}
```

```
int main() {
    int a = 15, b = -1, c;
    c = sum(a, b);
    return 0;
}
```

		algoritmo pseudocódigo	ebp
main:		In prima faz se adiciona ao vetor valores de saída da função sub alocada na memória.	c = ?
push	ebp	Na etapa de saída o vetor é copiado para o endereço de retorno.	b = -1
mov	ebp, esp	Em uma das etapas o valor é alterado.	a = 15
sub	esp, 0xc	entre inserções e remoções.	b
mov	dword ptr [ebp - 0xc]	15	a
mov	dword ptr [ebp - 0xc], -1		esp = esp + (add esp, 8)
push	dword ptr [ebp - 0xa]	sum proc	
push	dword ptr [ebp - 0xc]	ebp	
call	sum	ebp	
add	ebp, 8	ebp	
mov	dword ptr [ebp - 0x4], eax	mov	
mov	eax, 0	eax, dword ptr [ebp + 0xc]	
leave		add	
ret		eax, edx	
end main		pop	
		ret	
		ebp	
		sum endp	

```

main:
    push    ebp
    mov     ebp, esp
    sub    esp, 0x1C

    mov     dword ptr [ebp - 0x15], 15
    mov     dword ptr [ebp - 0x15], -1

    push    dword ptr [ebp - 0x8]
    push    dword ptr [ebp - 0x8c]
    call    sum
    add    esp, 8
    mov     dword ptr [ebp - 0x4], eax
    mov     eax, 0
    leave
    ret

end main

```

sum endp

```

Traducem în cod assembly

main:
    push    ebp
    mov     ebp, esp
    sub     esp, 0x18

    mov     dword ptr [ebp - 0x10], 15
    mov     dword ptr [ebp - 0x14], -1

    push    dword ptr [ebp - 0xC]
    push    dword ptr [ebp - 0x8]
    call    sum
    add    esp, 8
    mov     dword ptr [ebp - 0x4], eax
    pop    ebp
    ret

    mov     eax, 0
    sum endp

    leave
    ret
end main

```

		Param for above parameter function starts here. Confirm conversion of all parameters on & p to stack in order of the discipline the storage locations in esp to a	ebp
main:			c = ?
push	ebp		b = -1
mov	ebp, esp		a = 15
sub	esp, 0xC		b
mov	dword ptr [ebp - 0xC]		esp
mov	dword ptr [ebp - 0x8]		a
push	dword ptr [ebp - 0x4]	sum proc	
push	dword ptr [ebp - 0x8]	push	ebp
call	sum	mov	ebp, esp
add	esp, 8		
mov	dword ptr [ebp - 0x4], eax	mov	eax, dword ptr [ebp + 0xC]
mov	eax, 0	add	dword ptr [ebp + 0x8]
leave		pop	eax, edx
ret		ret	ebp
end main		sum endp	

		ebp	c = 7
main:			
push	ebp		Functie rezervata calculate si
mov	ebp, esp		după rezultatul în registrul
sub	esp, 0x8C		ace, conform setările
			univertii adăug
mov	dword ptr [ebp - 0x8C], 15		
mov	dword ptr [ebp - 0x8C], proc		
		sum	
push	dword ptr [ebp - 0x8C]	push	ebp
push	dword ptr [ebp - 0x8C]	mov	esp
call	sum		
add	esp, 8	mov	eax, dword ptr [ebp + 0x8C]
mov	dword ptr [ebp - 0x84], eax	add	edx, dword ptr [ebp + 0x84]
		add	eax, edx
mov	eax, 0	pop	ebp
leave		ret	
ret		sum	endf
end main			

			ebp
main:			<i>c = ?</i>
push	ebp		
mov	ebp, esp		
sub	esp, 0xL		
		<i>Fazendo prova de se altera. desloc foince aci passando depois de onde</i>	
mov	dword ptr [ebp - 0x1C], 15		
mov	dword ptr [ebp - 0x1C], 1		
push	dword ptr [ebp - 0x8], sum	proc	
push	dword ptr [ebp - 0x8], push	ebp	
call	sum	ebp	
add	esp, 8		
mov	dword ptr [ebp - 0x1E], eax	eax, dword ptr [ebp + 0x1C]	
mov	eax, 0	mov	
leave		ads	
ret		dword ptr [ebp + 0x1C]	
end main		adu	
		eax, eax	
		pop	
		ret	
		sum endp	
			<i>b = -1</i>
			<i>a = 15</i>
			<i>b</i>
			<i>a</i>
			<i>[ebp, esp]</i>

```

main:
    push    ebp
    mov     ebp, esp
    sub    esp, 0x8
    mov    dword ptr [ebp - 0x8], 15
    mov    dword ptr [ebp - 0x8], -1
    push    dword ptr [ebp - 0x8]
    push    dword ptr [ebp - 0x8]
    call    sum
    add    esp, 8
    mov    dword ptr [ebp - 0x4], eax
    mov    eax, 0
    leave
    ret
end main

        Confusion exercice amusant!
        code utile de certains callers pour
        se détourner des paramètres
        dans un échange rendu possible
        favorisant l'attribution manuelle

    sum proc
        push    ebp
        mov     ebp, esp
        mov    eax, 0
        add    eax, edx
        mov    edx, eax
        mov    eax, edx
        sum endp

```

		esp	ebp
main:			
push	ebp		
mov	ebp, esp		
sub	esp, 0xC		
mov	dword ptr [ebp - 0xC], 15		
mov	dword ptr [ebp - 0xC].1		
push	dword ptr [ebp - 0x8]	sum proc	
push	dword ptr [ebp - 0xC]	push	ebp
call		mov	ebp, esp
add	esp, 8	mov	
mov	dword ptr [ebp - 0xC].4	mov	eax, dword ptr [ebp + 0x8]
mov	eax, 0	add	eax, edx
mov	eax, 0	pop	ebp
leave		ret	
ret			
end main		sum	endp

Scriem codul la nivel înalt

```
int sum(int a, int b) {  
    return a + b;  
}
```

```
int main() {  
    int a = 15, b = -1, c;  
  
    c = sum(a, b);  
    return 0;  
}
```

Traducem în cod assembly

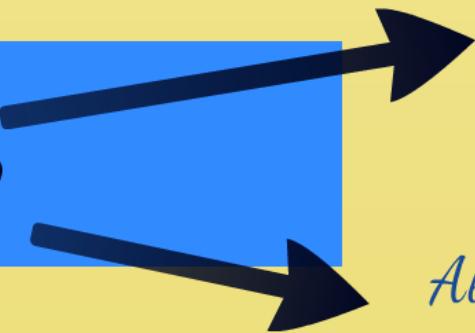
in:

push	ebp		
mov	ebp, esp		
sub	esp, 0x18		
mov	dword ptr [ebp - 0xC], 15	sum proc	push
mov	dword ptr [ebp - 0x8], -1		mov
push	dword ptr [ebp - 0xC]		ebp
push	dword ptr [ebp - 0x8]		ebp, esp
call	sum		
add	esp, 8	mov	eax, dword ptr [ebp + 0xC]
mov	dword ptr [ebp - 0x4], eax	mov	edx, dword ptr [ebp + 0x8]
		add	eax, edx
mov	eax, 0	pop	
leave		ret	ebp
ret		sum endp	

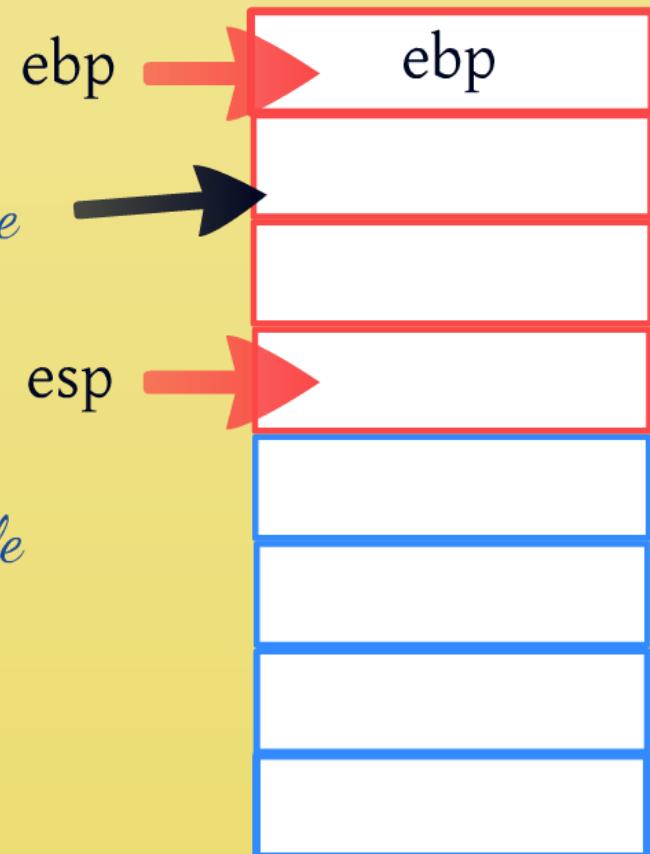
Rulam pas cu pas

in:

```
push    ebp  
mov     ebp, esp  
sub    esp, 0xC
```



Cream stack frame



Alocam spatiu pe stiva
pentru variabilele locale
(a, b, c):
 $3 \times 4 = 0xC$ octeti

```
mov     dword ptr [ebp - 0xC], 15  
mov     dword ptr [ebp - 0x8], -1
```

```
push    dword ptr [ebp - 0xC]  
push    dword ptr [ebp - 0x8]  
call    sum  
add    esp, 8  
mov     dword ptr [ebp - 0x4], eax
```

```
mov     eax, 0  
leave  
ret
```

```
sum proc  
    push    ebp  
    mov     ebp, esp  
  
    mov     eax, dword ptr [ebp + 0xC]  
    mov     edx, dword ptr [ebp + 0x8]  
    add    eax, edx  
    pop    ebp  
    ret  
  
sum endp
```

Rulam pas cu pas

in:

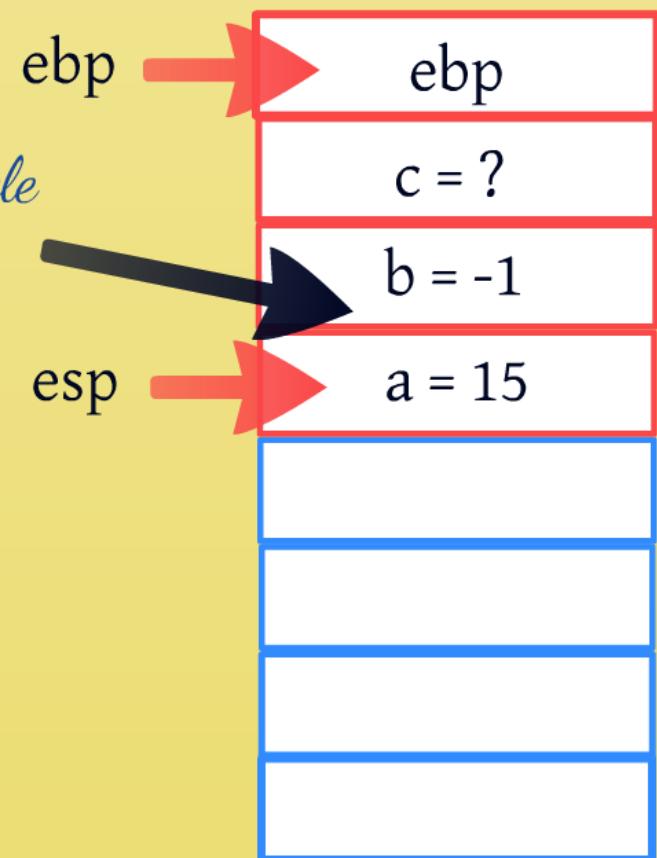
push	ebp
mov	ebp, esp
sub	esp, 0xC

mov	dword ptr [ebp - 0xC], 15
mov	dword ptr [ebp - 0x8], -1

push	dword ptr [ebp - 0x8]
push	dword ptr [ebp - 0xC]
call	sum
add	esp, 8
mov	dword ptr [ebp - 0x4], eax

mov	eax, 0
leave	
ret	

Initializam variabilele locale, de pe stiva.



sum proc	
push	ebp
mov	ebp, esp
mov	dword ptr [ebp + 0xC]
mov	dword ptr [ebp + 0x8]
add	eax, edx
pop	ebp
ret	
sum endp	

in:

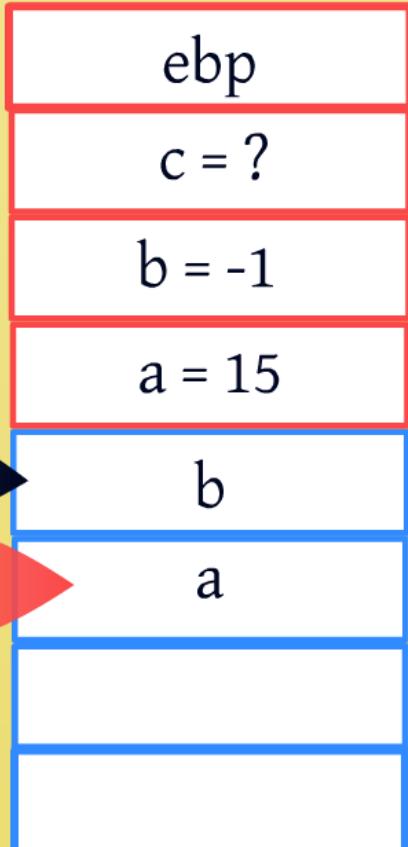
```
push    ebp  
mov     ebp, esp  
sub    esp, 0xC  
  
mov     dword ptr [ebp - 0xC], 15  
mov     dword ptr [ebp - 0x8], -1
```

```
push    dword ptr [ebp - 0x8]  
push    dword ptr [ebp - 0xC]  
call    sum  
add    esp, 8  
mov     dword ptr [ebp - 0x4], eax  
  
mov     eax, 0  
leave  
ret  
sum endp
```

Punem pe stiva parametrii functiei sum(a, b).

Conform conventiei cdecl, parametrii vor fi pusi pe stiva in ordine, de la dreapta la stanga: intai b si apoi a.

esp



```
sum proc  
push    ebp  
mov     ebp, esp  
  
mov     eax, dword ptr [ebp + 0xC]  
mov     edx, dword ptr [ebp + 0x8]  
add    eax, edx  
pop    ebp  
ret  
sum endp
```

in:

push	ebp
mov	ebp, esp
sub	esp, 0xC

mov	dword ptr [ebp - 0xC], 15
mov	dword ptr [ebp - 0x8], -1

push	dword ptr [ebp - 0x8]
push	dword ptr [ebp - 0xC]
call	sum
add	esp, 8
mov	dword ptr [ebp - 0x4], eax

mov	eax, 0
leave	
ret	

Apelam procedura.

In prima fază salvăm pe stivă adresa de întoarcere, adică a următoarei instrucțiuni (add esp, 8).

In a doua etapă efectuam un salt către inceputul funcției sum.

sum proc

push
mov

ebp
ebp, esp

mov	eax, dword ptr [ebp + 0xC]
mov	edx, dword ptr [ebp + 0x8]
add	eax, edx
pop	ebp
ret	

sum endp



in:

```
push    ebp  
mov     ebp, esp  
sub    esp, 0xC
```

```
mov     dword ptr [ebp - 0xC], 15  
mov     dword ptr [ebp - 0x8], -1
```

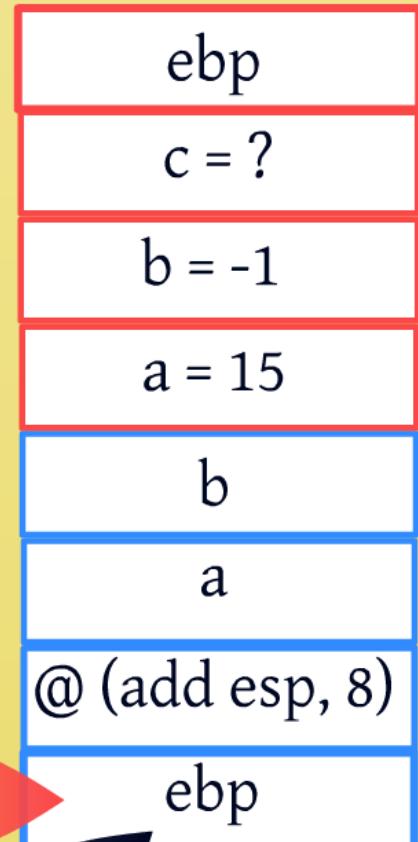
```
push    dword ptr [ebp - 0x8]  
push    dword ptr [ebp - 0xC]  
call    sum
```

```
add    esp, 8  
mov     dword ptr [ebp - 0x4], eax
```

```
mov     eax, 0  
leave  
ret
```

Procedura apelata isi creeaza propriul frame pointer. Parte din contextul ei (partea de parametri din stack frame) a fost creat de catre functia apelanta..

```
sum proc  
    push    ebp  
    mov     ebp, esp  
    add    esp, 8  
    mov     eax, dword ptr [ebp + 0xC]  
    mov     edx, dword ptr [ebp + 0x8]  
    add    eax, edx  
    pop     ebp  
    ret  
sum endp
```



in:

```
push    ebp  
mov     ebp, esp  
sub    esp, 0xC
```

```
mov     dword ptr [ebp - 0xC], 15  
mov     dword ptr [ebp - 0x8], -1
```

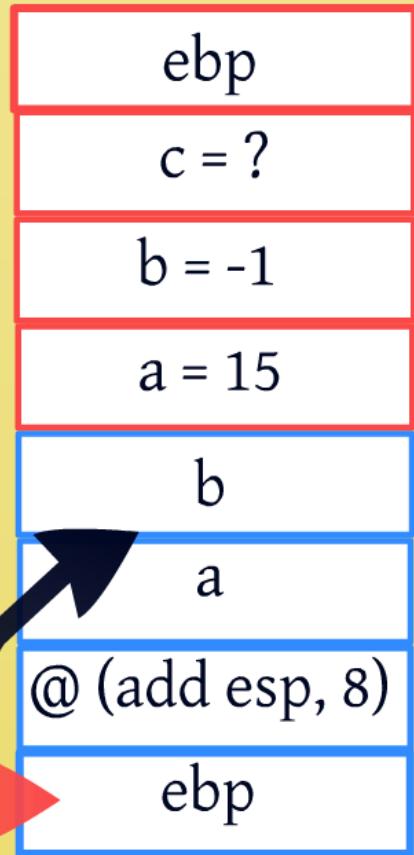
```
push    dword ptr [ebp - 0x8]  
push    dword ptr [ebp - 0xC]  
call    sum  
add    esp, 8  
mov     dword ptr [ebp - 0x4], eax
```

```
mov     eax, 0  
leave  
ret
```

sum proc

push ebp	esp
mov ebp, esp	ebp
mov eax, dword ptr [ebp + 0xC]	eax, edx
mov edx, dword ptr [ebp + 0x8]	add eax, edx
pop ebp	ret
sum endp	

*Functia preia de pe stiva,
din stack frame-ul ei,
parametrii depusi de catre
apelant (caller).*



in:

```
push    ebp  
mov     ebp, esp  
sub    esp, 0xC
```

```
mov     dword ptr [ebp - 0xC], 15  
mov     dword ptr [ebp - 0x8], -1
```

```
push    dword ptr [ebp - 0x8]  
push    dword ptr [ebp - 0xC]  
call    sum  
add    esp, 8
```

```
mov     dword ptr [ebp - 0x4], eax
```

```
mov     eax, 0
```

```
leave  
ret
```

Functia executa calculele, si depune rezultatul in registrul eax, conform aceleiasi conventii cdecl.

sum proc

push ebp
mov ebp, esp

mov eax, dword ptr [ebp + 0xC]
mov edx, dword ptr [ebp + 0x8]

add eax, edx

pop ebp
ret

sum endp

ebp
c = ?
b = -1
a = 15
b
a
@ (add esp, 8)
ebp



in:

push	ebp
mov	ebp, esp
sub	esp, 0xC

Procedura isi distrugе parte din stack frame (partea creata de ea), in pregatirea parasirii executiei.

mov	dword ptr [ebp - 0xC], 15
mov	dword ptr [ebp - 0x8], -1

sum proc

push	dword ptr [ebp - 0x8]
push	dword ptr [ebp - 0xC]
call	sum

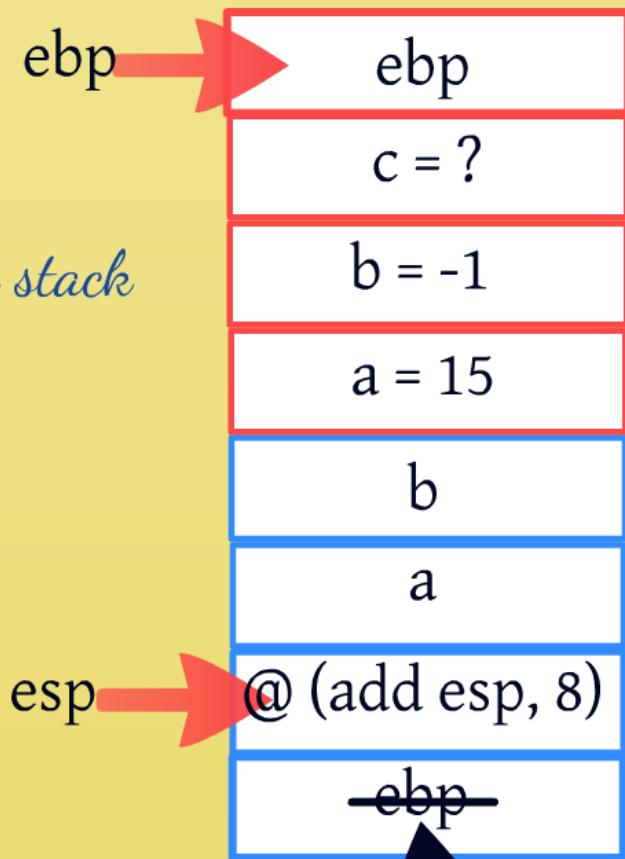
push	ebp
mov	ebp, esp

add	esp, 8
mov	dword ptr [ebp - 0x4], eax

mov	eax, dword ptr [ebp + 0xC]
mov	edx, dword ptr [ebp + 0x8]
add	eax, edx

mov	eax, 0
leave	
ret	

sum endp



in:

push	ebp
mov	ebp, esp
sub	esp, 0xC

mov	dword ptr [ebp - 0xC], 15
mov	dword ptr [ebp - 0x8], -1

push	dword ptr [ebp - 0x8]
push	dword ptr [ebp - 0xC]
call	sum
add	esp, 8

mov	dword ptr [ebp - 0x4], eax
-----	----------------------------

mov	eax, 0
-----	--------

leave	
ret	

*Procedura scoate adresa
instructiunii de revenire din
varful stivei, si executa un salt catre
ea.*

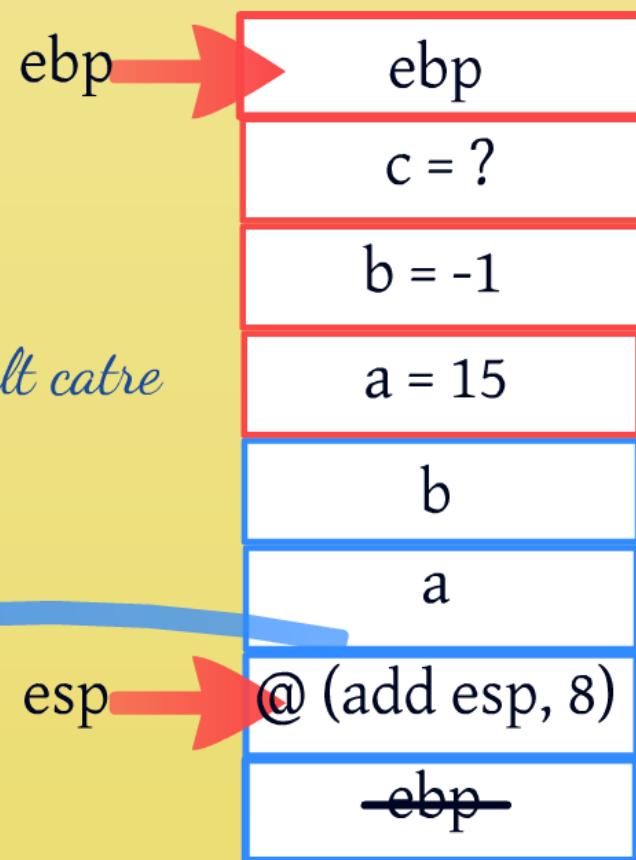
sum proc

push	ebp
mov	ebp, esp

mov	eax, dword ptr [ebp + 0xC]
mov	edx, dword ptr [ebp + 0x8]
add	eax, edx

pop	ebp
ret	

sum endp



in:

push	ebp
mov	ebp, esp
sub	esp, 0xC

mov	dword ptr [ebp - 0xC], 15
mov	dword ptr [ebp - 0x8], -1

push	dword ptr [ebp - 0x8]
push	dword ptr [ebp - 0xC]
call	sum

add	esp, 8
mov	dword ptr [ebp - 0x4], eax

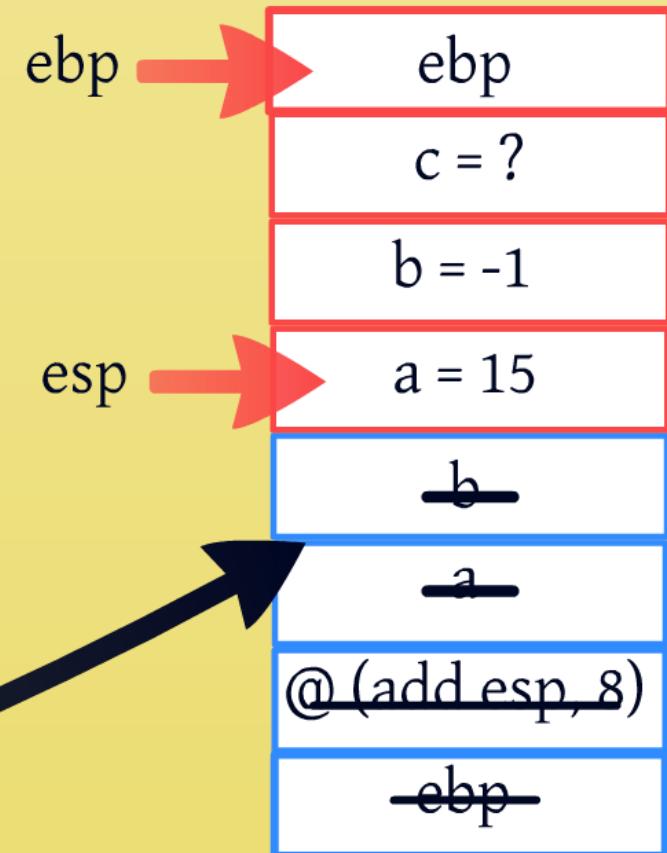
mov	eax, 0
leave	
ret	

*Conform aceleiasi conventii
cdecl, este de datoria caller-ului
sa descarce stiva de parametri.
deci sa distruga restul stack
frame-ului (stack unwinding).*

sum proc
push
ebp
mov
ebp, esp

mov
mov
add
eax, dword ptr [ebp + 0xC]
edx, dword ptr [ebp + 0x8]
eax, edx

pop
ret
sum endp
ebp



in:

push	ebp
mov	ebp, esp
sub	esp, 0xC

mov	dword ptr [ebp - 0xC], 15
mov	dword ptr [ebp - 0x8], -1

push	dword ptr [ebp - 0x8]
push	dword ptr [ebp - 0xC]
call	sum

add	esp, 8
-----	--------

mov	dword ptr [ebp - 0x4], eax
-----	----------------------------

mov	eax, 0
-----	--------

leave	
ret	

Depune rezultatul functiei in variabila locala c.

sum proc

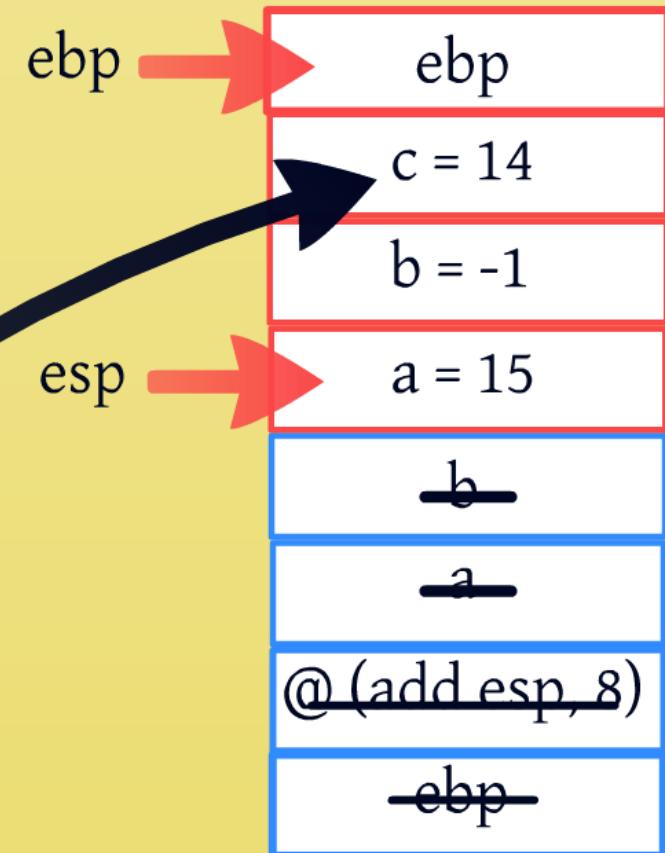
push	ebp
mov	ebp, esp

mov	eax, dword ptr [ebp + 0xC]
mov	edx, dword ptr [ebp + 0x8]
add	eax, edx

da, acel eax

pop	
ret	

sum endp



esp →

ebp
c = 14
b = -1
a = 15
b
a
@ (add esp, 8)
-ebp

in:

push ebp
mov esp, ebp
sub esp, 0xC

mov dword ptr [ebp - 0xC], 15
mov dword ptr [ebp - 0x8], -1

sum proc

push dword ptr [ebp - 0x8]
push dword ptr [ebp - 0xC]

call sum

add esp, 8

mov dword ptr [ebp - 0x4], eax

mov eax, 0

leave

ret

pop ret

sum endp

Codul de return sta in eax.
Leave este o instructiune care
distrugе stack frame-ul functiei,
executand de fapt mov esp, ebp
si apoi pop ebp.

Concepte de memorie virtuala

*Page fault
Swap
Demand paging*

Page fault

acces la o pagina virtuala:

- nemapata (demand paging, swap)
- nevalida (pagina virtuala nealocata)
- drepturi nevalide (scriere pe o pagina read-only, copy-on-write)

se genereaza o exceptie/trap

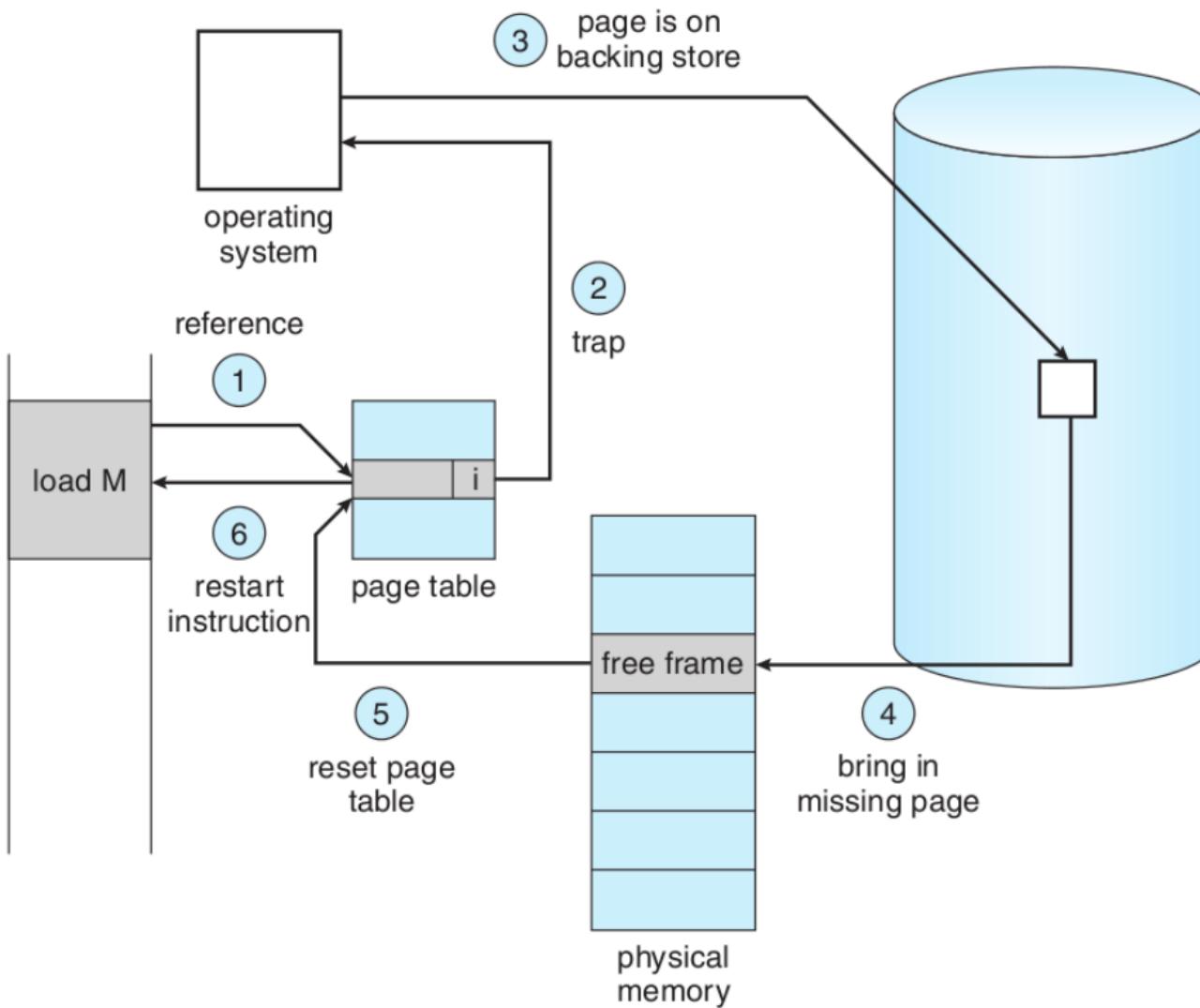
MMU genereaza exceptia

se ruleaza page fault handler-ul

page fault-urile pot sa nu fie erori

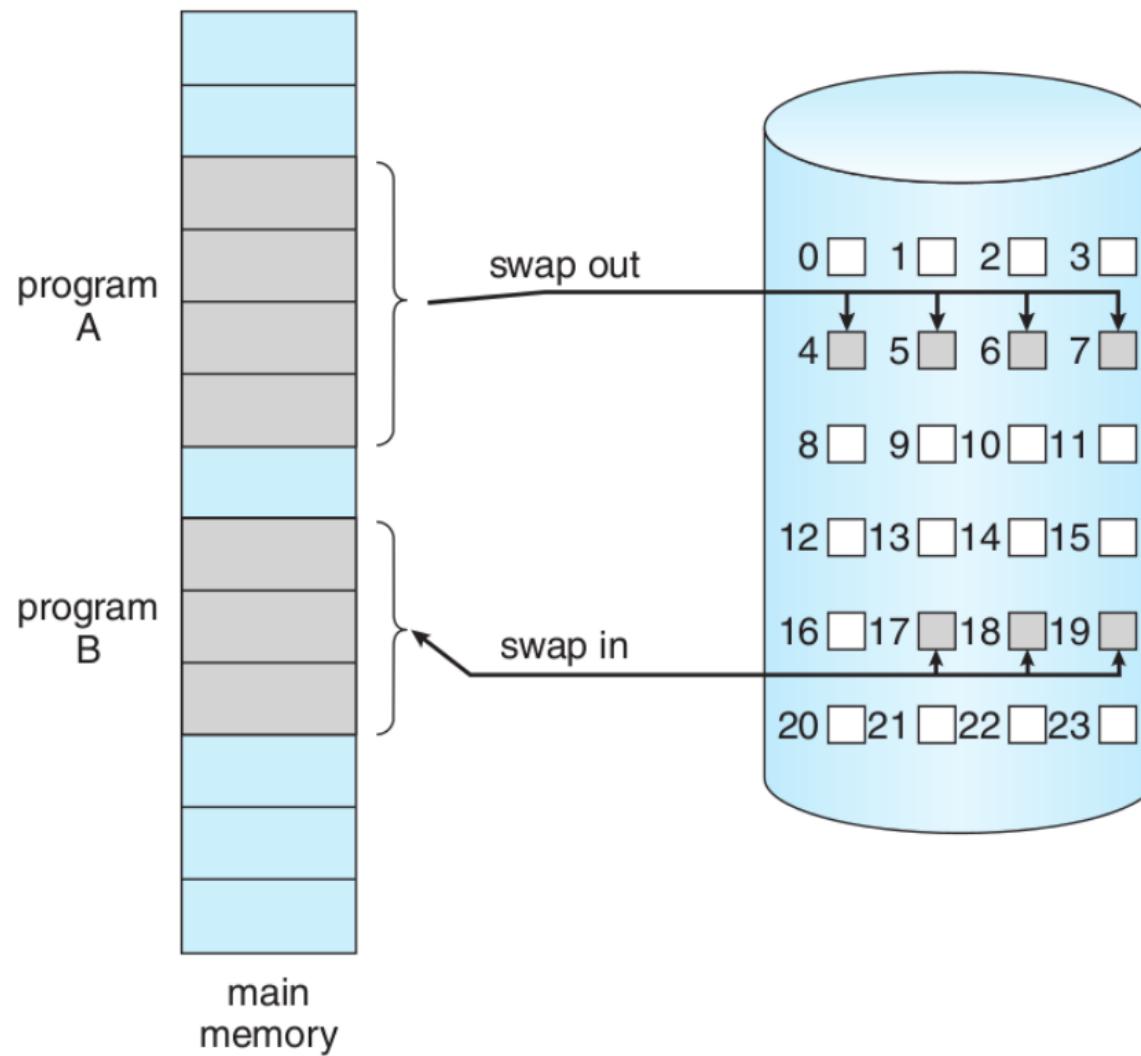
- minor page faults: demand paging
- major page faults: swap

Page fault (2)



OSCE, Chapter 8, pg. 325, Figure 8.6

Swap



OSCE, Chapter 8, pg. 323, Figure 8.4

Swap (2)

in absenta memoriei fizice, se pot evacua pagini pe disc
spatiu de swap

- swap in (aducerea unei pagini de pe disc in RAM)
- swap out (trimiterea unei pagini din RAM pe disc)

maresteste spatiul posibil folosit
mult mai incet decat memoria RAM

algoritmi de inlocuire de pagina

Demand paging

Paginare la cerere

paginare = alocarea unei pagini fizice si maparea acesteia

decuplarea alocarii de memorie virtuala de mapare

lazy allocation/paging

la alocare se aloca o intrare in tabela de pagini

- pagina este marcata nevalida
- este alocata pagina fizica (de la zero sau de swap) la nevoie

Demand paging (2)

un proces incarcat in memorie isi incarca doar o parte din date

- restul sunt incarcate la nevoie

o alocare cu mmap aloca doar pagini virtuale

- paginile fizice sunt alocate la acces

alocarea fizica si maparea se fac in page fault handler

Alocarea memoriei virtuale

*Rezervare si commit
Demand paging
Pagini rezidente
Page locking*

Rezervare si commit

rezervare: alocare memorie virtuala (doar virtuala)

- apelurile malloc (dimensiuni mari), mmap, VirtualAlloc

commit: alocare memorie fizica pentru memoria alocata

- la demand paging

decuplarea celor doua permite alocarea rapida de memorie

apelurile de biblioteca malloc fac mai putine apeluri de sistem brk

Demand paging

alocare la cerere

rezervare in prima faza

commit in page fault handler, la acces

Pagini rezidente

paginiile prezente in RAM sunt rezidente
altfel pot fi pe swap sau inca nealocate

RSS = Resident Set Size, spatiul ocupat de un proces in RAM

alocare rezidenta (paginile nu pot fi swappate)

- la nivelul nucleului de operare
- sau prin page locking/pinning

Page locking

page pinning, fixed pages, non-pageable

pagina este marcata rezidenta (neswappabila)

utila pentru comunicarea cu dispozitive I/O

mlock() - Linux

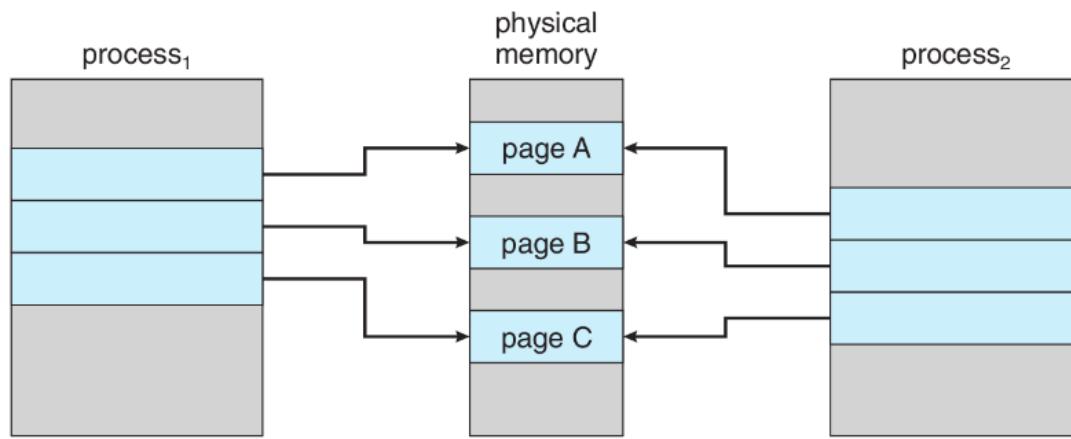
VirtualLock() - Windows

Mecanisme de memorie virtuală

*Copy-on-Write
Maparea fisierelor*



Copy-on-Write



OSCE, Chapter 8, pg. 330, Figures 8.6 & 8.8

Copy-on-Write (2)

implementare naiva fork(): se copiaza spatiul de adresa

implementare eficienta: se duplica tabela de pagini

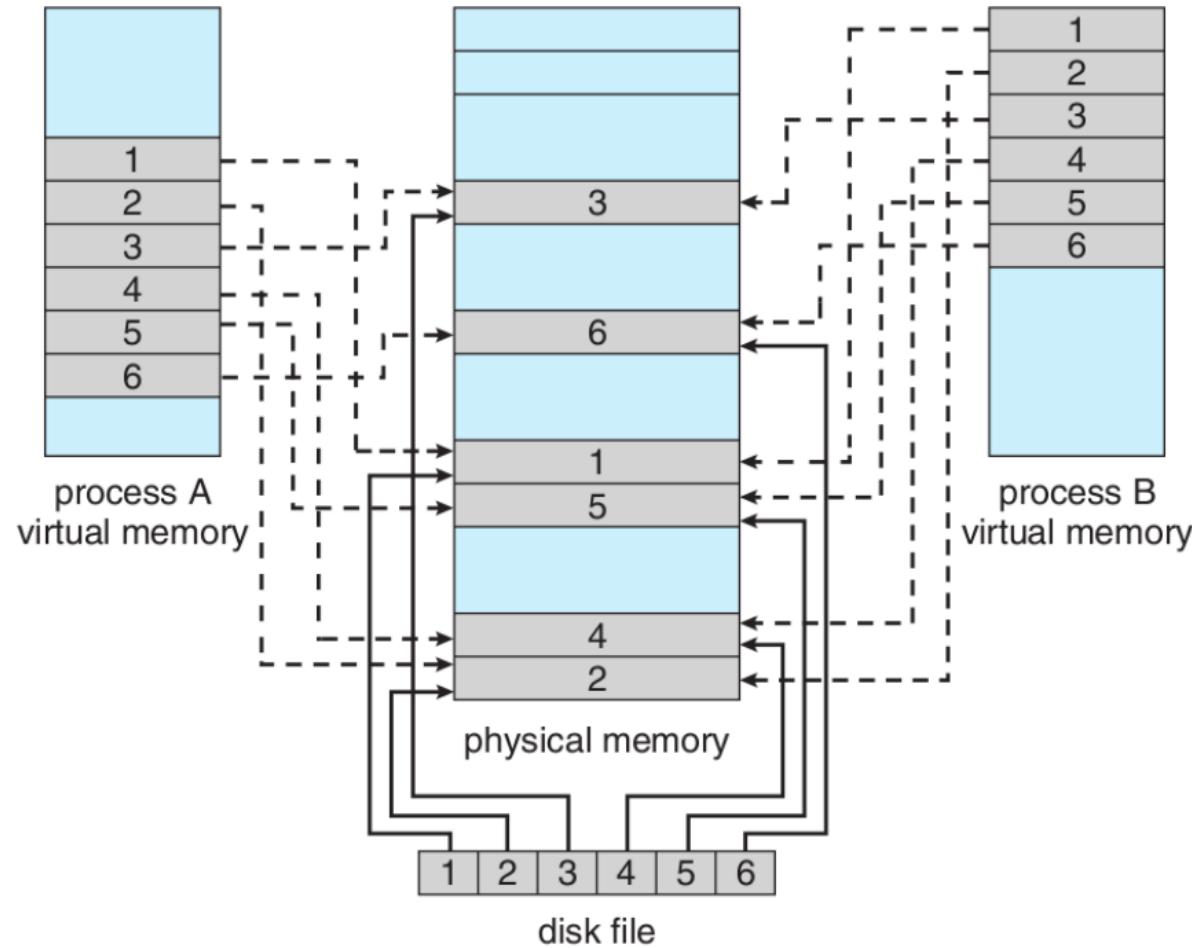
se partajeaza paginile intre procesul fiu si procesul parinte

- paginile sunt marcate read-only

in momentul unui acces de tip scriere se genereaza page fault

- se creeaza o copie a paginii configurata READ_WRITE la care va avea acces doar procesul care a generat page fault
- Pagina originala ramane configurata READ_ONLY si va avea acces la ea doar celalalt proces

Maparea fisierelor



OSCE, Chapter 8, pg. 354, Figure 8.23

Maparea fisierelor (2)

un bloc de date al unui fisier este mapat intr-o pagina/set de pagini
lucrul cu fisiere se realizeaza prin operatii de acces la memorie

folosit pentru incarcarea executabilelor si bibliotecilor

scrierea la o adresă de memorie inseamna scrierea in fisier
scriserile nu sunt imediate/sincrone (se folosesc apele msync)
memoria este folosita pe post de cache

mai putine apele de sistem pentru lucru cu fisiere

permite shared memory

Second chance

- se tine corespunzător
- se inspectează paginile
- dacă $R=0$, pagina este înlocuită
- dacă $R=1$, pagina este lăsată în memoria fizică

Inlocuirea paginilor

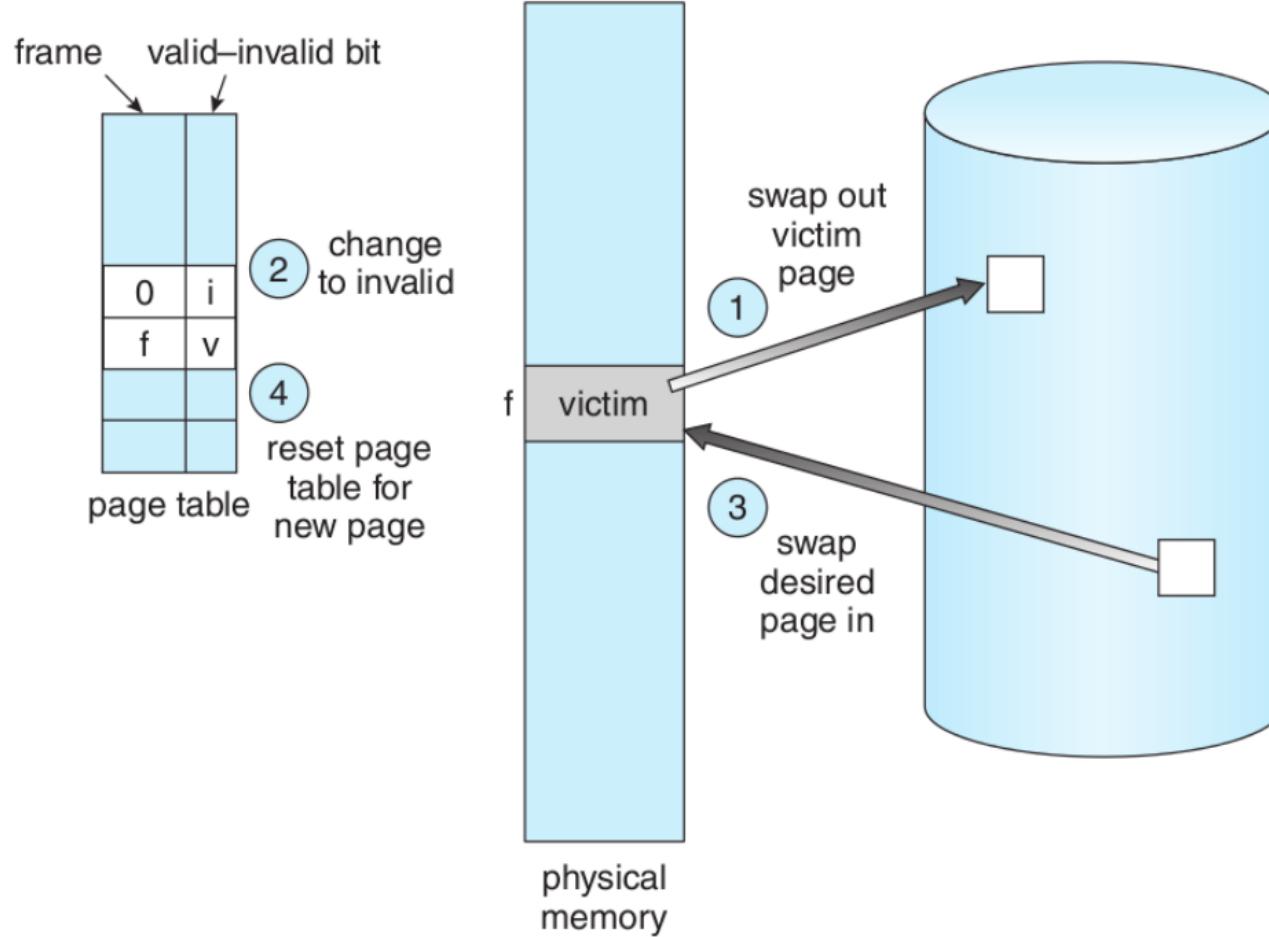
Inlocuirea paginilor

Algoritmi de inlocuire a paginilor

Anomalia lui Belady

Thrashing

Inlocuirea paginilor



OSCE, Chapter 8, pg. 333, Figure 8.10

Algoritmi de inlocuire a paginilor

ce pagina va fi evacuata pe disc

optim: se inlocuieste pagina care va fi referita cel mai tarziu

biti de modified (M) si referenced (R) in pagina

- o pagina cu M a fost scrisa
- o astfel de pagina este "dirty"
- o pagina cu R evacuata, va fi doar invalidata nu si copiata pe disc

NRU: Not Recently Used (se inlocuieste clasa cea mai mica)

- clasa 0: $R = 0, M = 0$
- clasa 1: $R = 0, M = 1$
- clasa 2: $R = 1, M = 0$
- clasa 3: $R = 1, M = 1$

Algoritmi de inlocuire a paginilor (2)

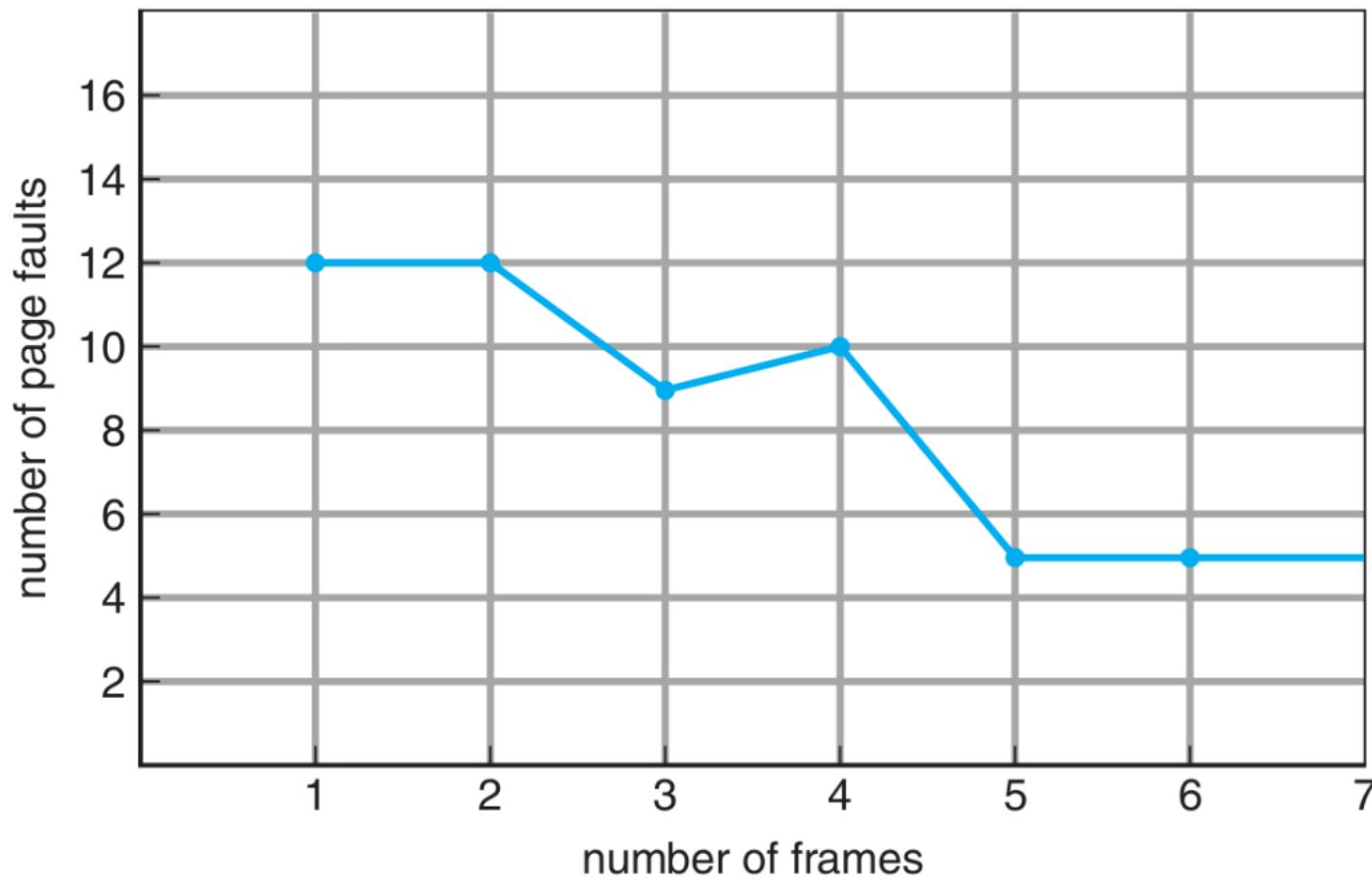
FIFO

- se mentine o lista cu paginile din memorie
- noile pagini sunt adaugate la sfarsitul listei
- se inlocuieste prima pagina din lista (cea mai veche)

Second chance (varianta modificata de FIFO)

- se tine cont de bitul R
- se inspecteaza prima pagina din lista
- daca $R=0$, pagina este selectata pentru inlocuire
- daca $R=1$, pagina este mutata la coada listei si $R=0$

Anomalia lui Belady



OSCE, Chapter 8, pg. 337, Figure 8.13

Thrashing

la incarcare mare a sistemului inlocuiri frecvente

schimbarea contextului duce la noi inlocuiri

- swap in/swap out frecvente

se petrece mult timp in page fault-uri (swap in/out)

- ineficienta in folosirea procesorului

exista si notiunea de cache thrashing

Cuvinte cheie

memorie virtuală
spatiu de adresa
swap
demand paging
page fault
copy-on-write

memory-mapped files
memorie partajată
inlocuire de pagina
NRU
FIFO, second chance
thrashing