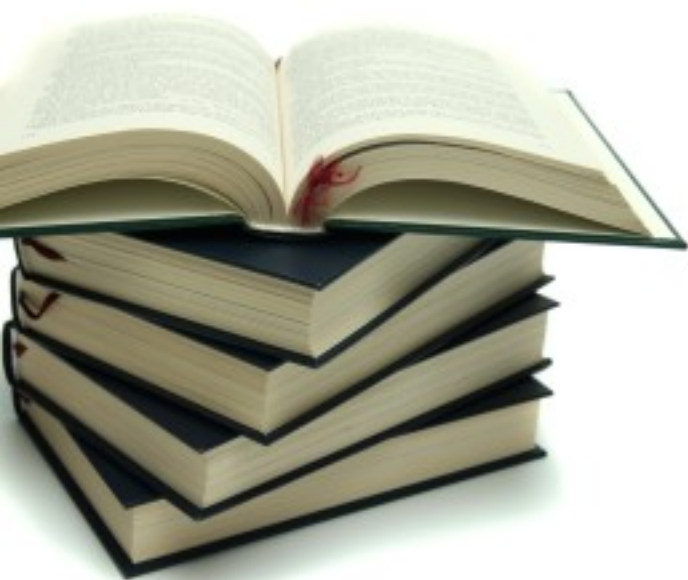


- Descrieți scenariul în care instrucțiunea x86 `INC [ADDR]`, rulând de pe două procesoare diferite, simultan, determină modificarea variabilei de la adresa `ADDR` doar cu o unitate (i.e. ajunge la `X` la `X+1` în loc de `X+2`)
- Ce primitive de sincronizare trebuie folosite pentru a proteja accesul la date comune între rutina de tratare a unui timer și operația `write` a unui device driver?
- Fie următoarea secvență de cod, executată simultan de pe mai multe procesoare:

```
struct a v[100000000];

for(i=0; i<100000000; i++)
    atomic_inc(&v[i].counter);
```

Știm că `sizeof(struct a)` este mult mai mic decât dimensiunea liniei de cache. Ce se poate face pentru a reduce fenomenul de cache trashing?



6

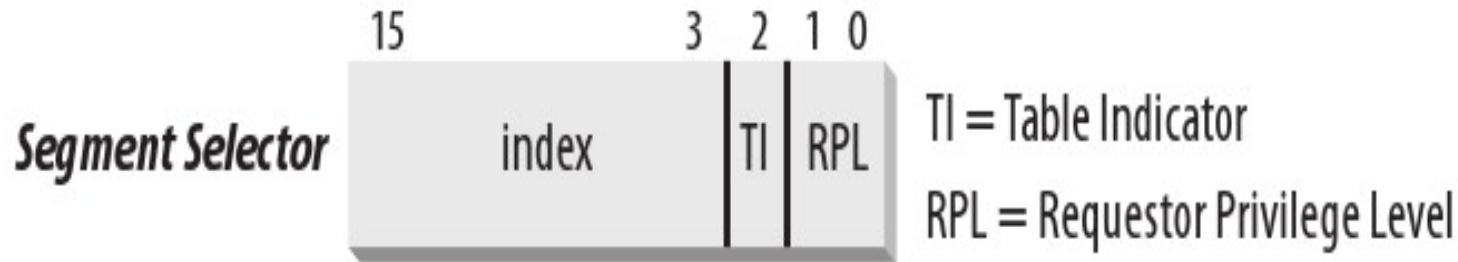
Gestiunea memoriei (I)

1 aprilie 2010

- Adresare memoriei x86
 - Segmentare
 - Paginare
 - TLB
- Organizarea spațiului de adresă
 - User
 - Kernel
- High memory
- Bibliografie
 - UTLK: capitolul 2

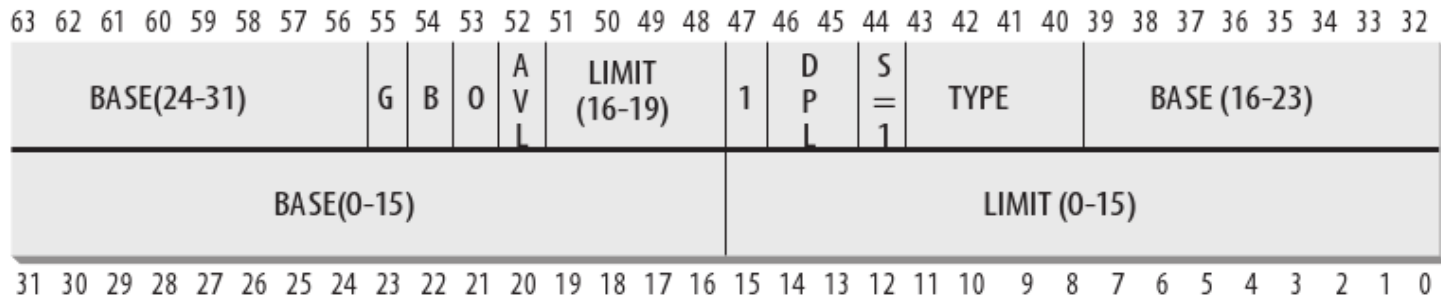
- Segmentare + paginare
- Adrese x86:
 - Logice: adresele pe care le folosește procesorul
 - Lineare: adresele ce sunt generate de unitate de segmentare
 - Fizice: adresele folosite pentru accesarea memoriei fizice; generate de unitate de paginare



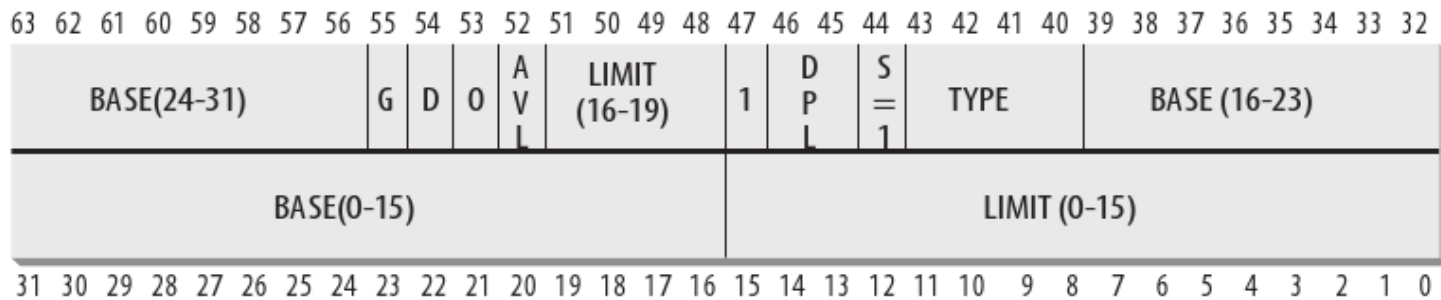


- Selectori = Regiștri: cs, ds, ss, es, fs, gs
- Index: Indexează tabela de descriptori
- TI: indică tabela: GDT sau LDT
- Tabelele sunt ținute în memorie la adrese specificate în regiștri: gdtr și ldtr
- RPL: dacă selectorul este încărcat în CS, specifică nivelul curent de privilegiu (CPL)

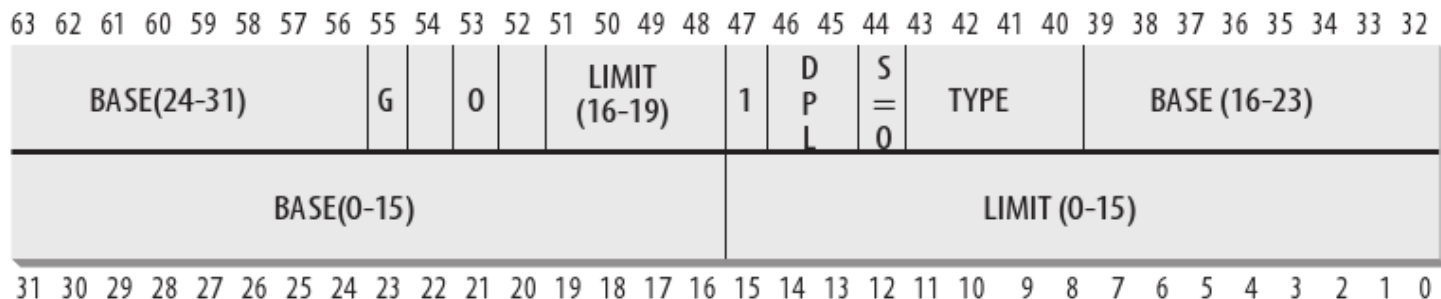
Data Segment Descriptor



Code Segment Descriptor



System Segment Descriptor



- Baza: adresa (liniară) pentru începutul segmentului
- Limita: dimensiunea segmentului
- G: bitul de granularitate: dacă nu este setat dimensiunile se exprimă în octeți, altfel în pagini de 4K
- B/D: data/code
- Tipul: segment de cod, date/stivă, TSS, LDT, GTD
- Protecție: Nivelul minim de privilegiu necesar accesării segmentului (se compara DPL cu CPL)

```
DEFINE_PER_CPU(struct gdt_page, gdt_page) = { .gdt = {  
    /* kernel 4GB code at 0x00000000 */  
    [GDT_ENTRY_KERNEL_CS] = { { { 0x0000ffff, 0x00cf9a00 } } },  
    /* kernel 4GB data at 0x00000000 */  
    [GDT_ENTRY_KERNEL_DS] = { { { 0x0000ffff, 0x00cf9200 } } },  
    /* user 4GB code at 0x00000000 */  
    [GDT_ENTRY_DEFAULT_USER_CS] = { { { 0x0000ffff, 0x00cffa00 } } },  
    /* user 4GB data at 0x00000000 */  
    [GDT_ENTRY_DEFAULT_USER_DS] = { { { 0x0000ffff, 0x00cff200 } } },  
    ...  
}
```

- + segment pentru TSS – pentru adresa stivei kernel: folosită la schimbarea contextul din user mode în kernel mode
- + segment pentru LDT: Aplicațiile ce au nevoie de segmentare (WINE) folosesc apelul de sistem `sys_modify_ldt` pentru a crea noi segmente (în LDT)


```
c:\so\x\>w2k_mem +c
```

```
[...]
```

```
CPU information:
```

```
-----
```

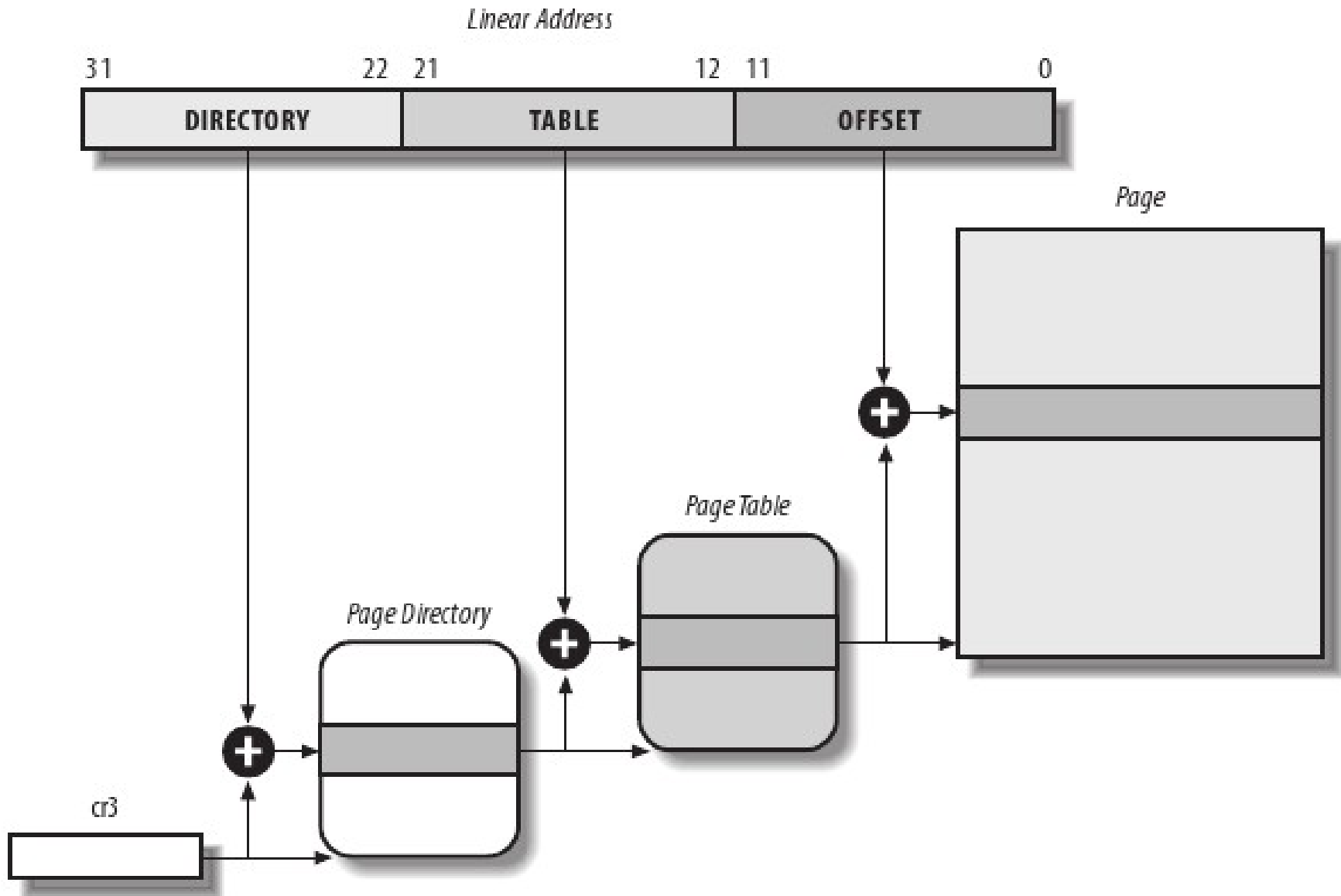
```
User mode segments:
```

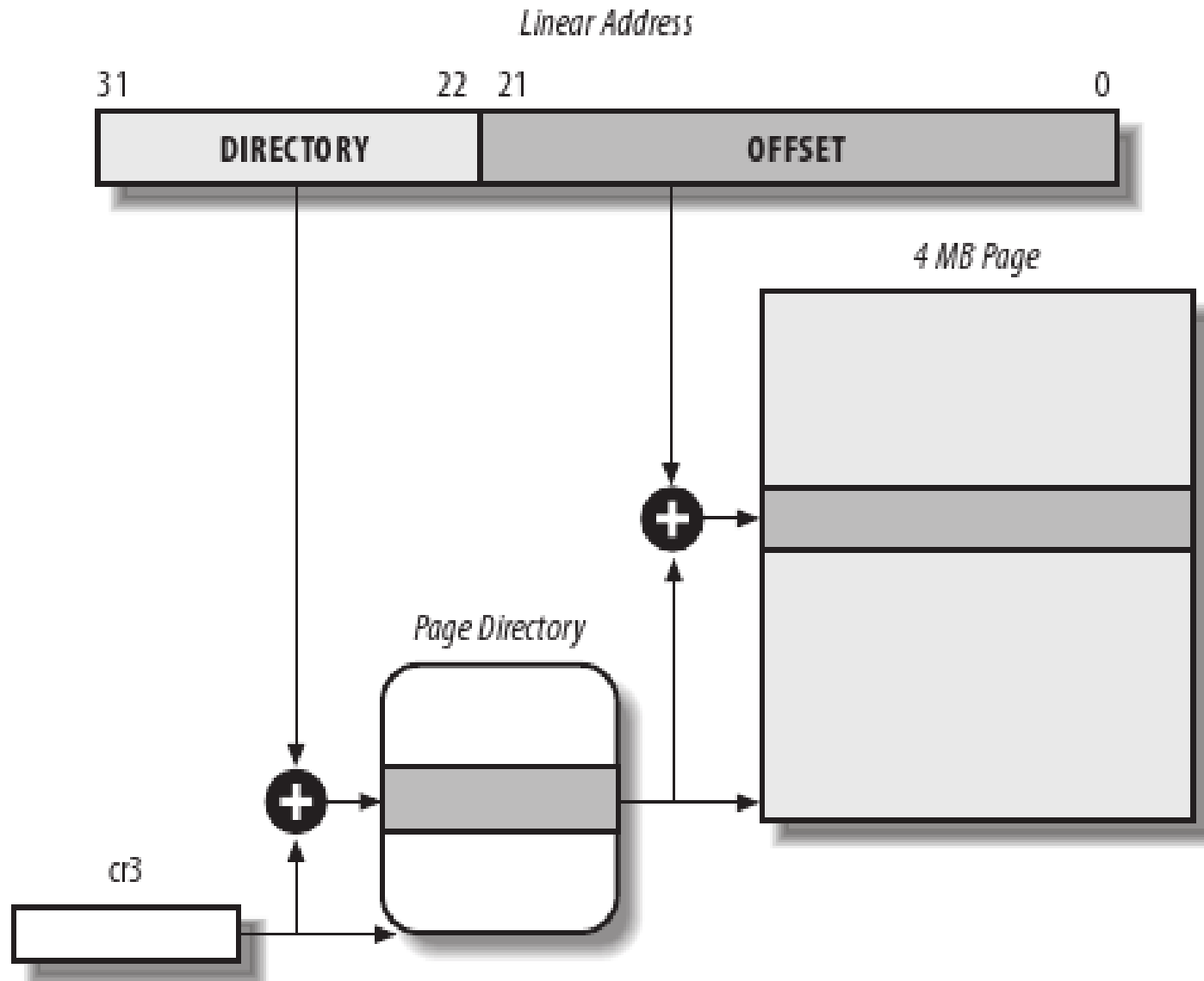
```
CS : Selector = 001B, Base = 00000000, Limit = FFFFFFFF, DPL3, Type = CODE -ra
DS : Selector = 0023, Base = 00000000, Limit = FFFFFFFF, DPL3, Type = DATA -wa
ES : Selector = 0023, Base = 00000000, Limit = FFFFFFFF, DPL3, Type = DATA -wa
FS : Selector = 0038, Base = 7FFDE000, Limit = 00000FFF, DPL3, Type = DATA -wa
SS : Selector = 0023, Base = 00000000, Limit = FFFFFFFF, DPL3, Type = DATA -wa
TSS : Selector = 0028, Base = 80244000, Limit = 000020AB, DPL0, Type = TSS32 b
```

```
Kernel mode segments:
```

```
CS : Selector = 0008, Base = 00000000, Limit = FFFFFFFF, DPL0, Type = CODE -ra
DS : Selector = 0023, Base = 00000000, Limit = FFFFFFFF, DPL3, Type = DATA -wa
ES : Selector = 0023, Base = 00000000, Limit = FFFFFFFF, DPL3, Type = DATA -wa
FS : Selector = 0030, Base = FFDF000, Limit = 00001FFF, DPL0, Type = DATA -wa
SS : Selector = 0010, Base = 00000000, Limit = FFFFFFFF, DPL0, Type = DATA -wa
TSS : Selector = 0028, Base = 80244000, Limit = 000020AB, DPL0, Type = TSS32 b
```

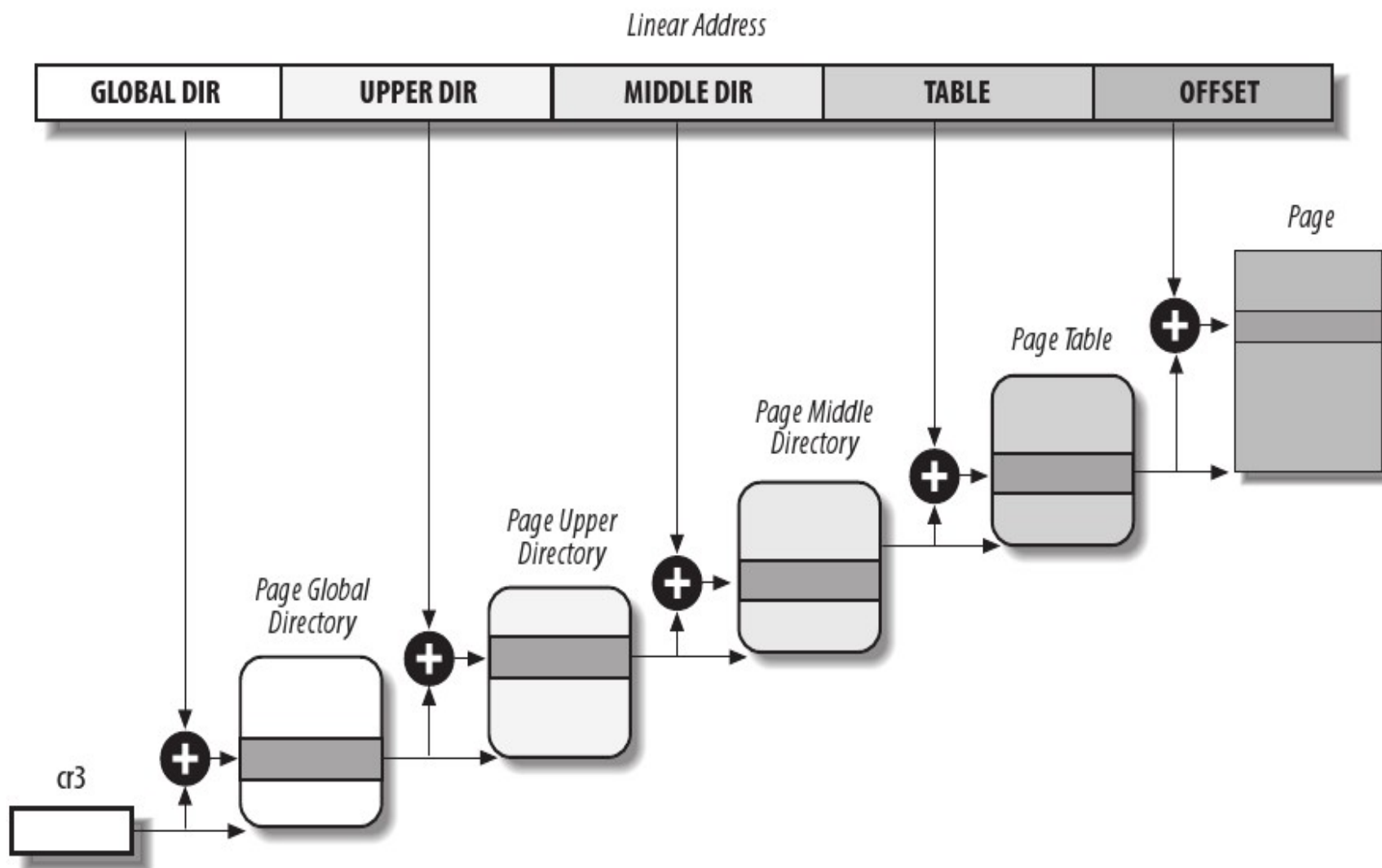
- Paginare normală
 - 2 nivele, pagină de 4K
 - Adresa (lineară) împărțită în 3 câmpuri
 - Directory (cei mai semnificativi 10 biți)
 - Table (următorii cei mai semnificativi 10 biți)
 - Offset (cei mai puțin semnificativi 12 biți)
- Paginare extinsă
 - Un singur nivel, pagină de 4M
 - Adresa lineară împărțită în 2 câmpuri
 - Directory (cei mai semnificativi 10 biți)
 - Offset (cei mai puțin semnificativi 22 biți)





- Fiecare tabelă (atât page directory cât și page table) poate avea maxim 1024 intrari și este ținută în memorie
- Fiecare intrare are 4 octeți
- Adresa Page Directory se setează prin registrul CR3 – adresă fizică
- Page directory conține pointeri către adresa de bază a page table-ului pe care îl referă – adrese fizice

- Present/Absent
- PFN: Cei mai semnificativi 20 de biți ai adresei fizice
- Accessed - flagul NU este actualizat de procesor
- Dirty - flagul NU este actualizat de procesor
- Drepturi de acces: Read/Write
- Privilegiu: User/Supervisor
- Page size - numai pentru intrările din Page Directory; dacă este setat se folosește paginare extinsă
- PCD (page cache disable), PWT (page write through)



```
struct * page;
pgd_t pgd;
pmd_t pmd;
pud_t pud;
pte_t pte;
void *laddr, *paddr;

pgd = pgd_offset(mm, vaddr);
pud = pud_offset(pgd, vaddr);
pmd = pmd_offset(pud, vaddr);
pte = pte_offset(pmd, vaddr);
page = pte_page(pte);
laddr = page_address(page);
paddr = virt_to_phys(laddr);
```



```
static inline pud_t * pud_offset(pgd_t * pgd,  
                                unsigned long address)  
{  
    return (pud_t *)pgd;  
}
```

```
static inline pmd_t * pmd_offset(pud_t * pud,  
                                unsigned long address)  
{  
    return (pmd_t *)pud;  
}
```

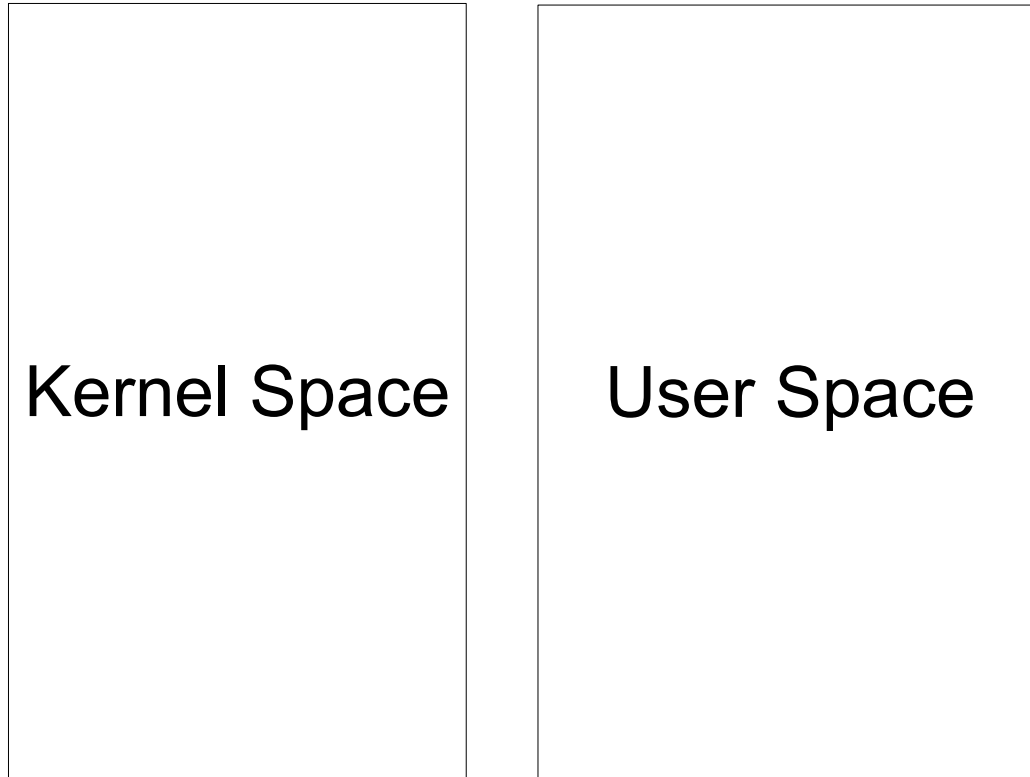
- Păstrează într-o tabelă informațiile despre o anumită pagină (PFN, drepturi, privilegii)
- Tabelă de dimensiune mică (64-128)
- Memorie asociativă (căutare paralelă)
- Două TLB-uri: i-TLB (pentru cod) și d-TLB (pentru date)
- Hit time: un ceas
- Miss penalty: 10 – 30 ceasuri
- Nu menține informații despre spațiul de adresă

- La modificarea mapării paginilor este necesară invalidarea unor intrari din TLB

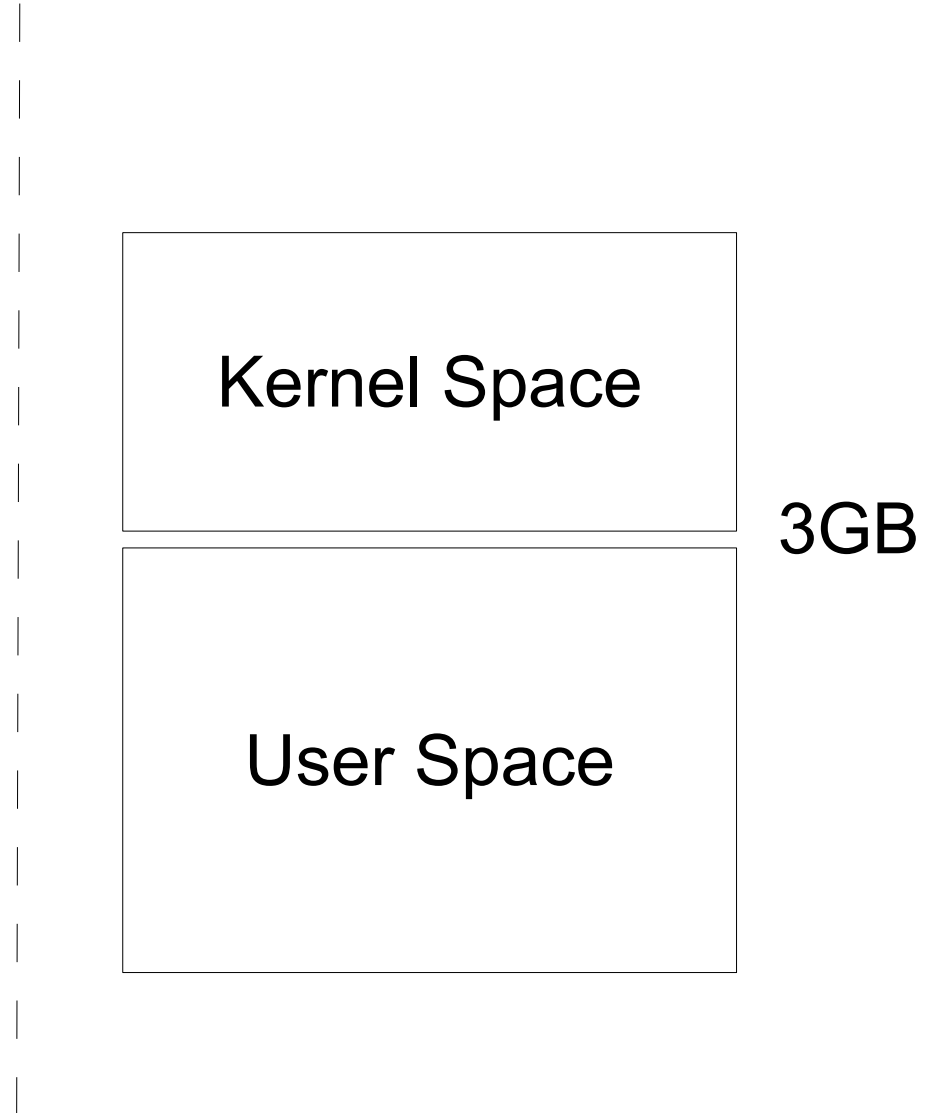
```
mov $addr, %eax  
invlpg %(eax)
```

- La modificarea registrului cr3 este invalidat automat tot TLB-ul

```
mov %cr3, %eax  
move %eax, %cr3
```



(a) 4/4 split



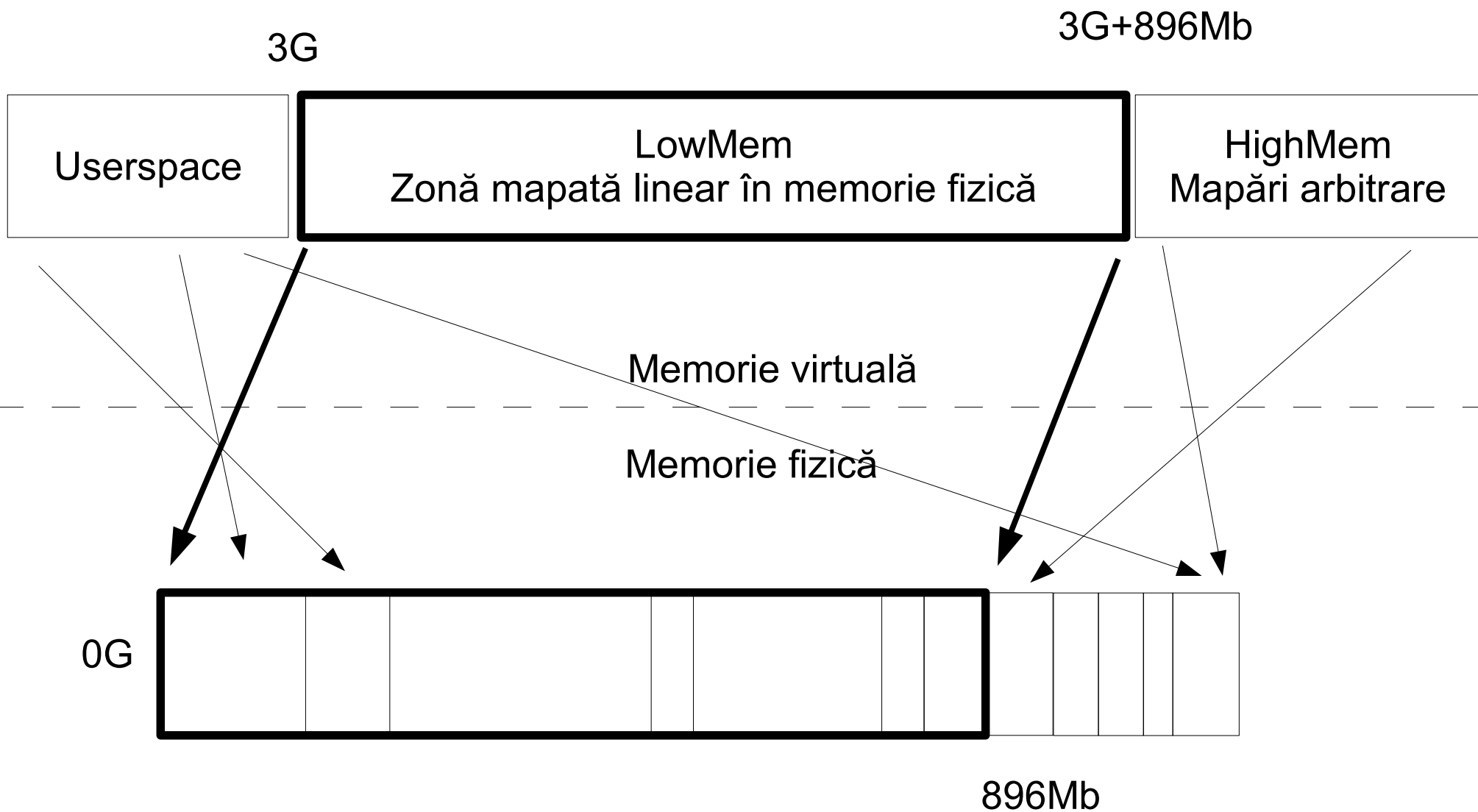
(b) 1/3 sau 2/2 split

(a) Kernel-ul are un spațiu de adresă propriu

- Dezavantaje: la fiecare apel de sistem trebuie să înlocuim spațiul de adresă, trebuie să golim TLB-ul

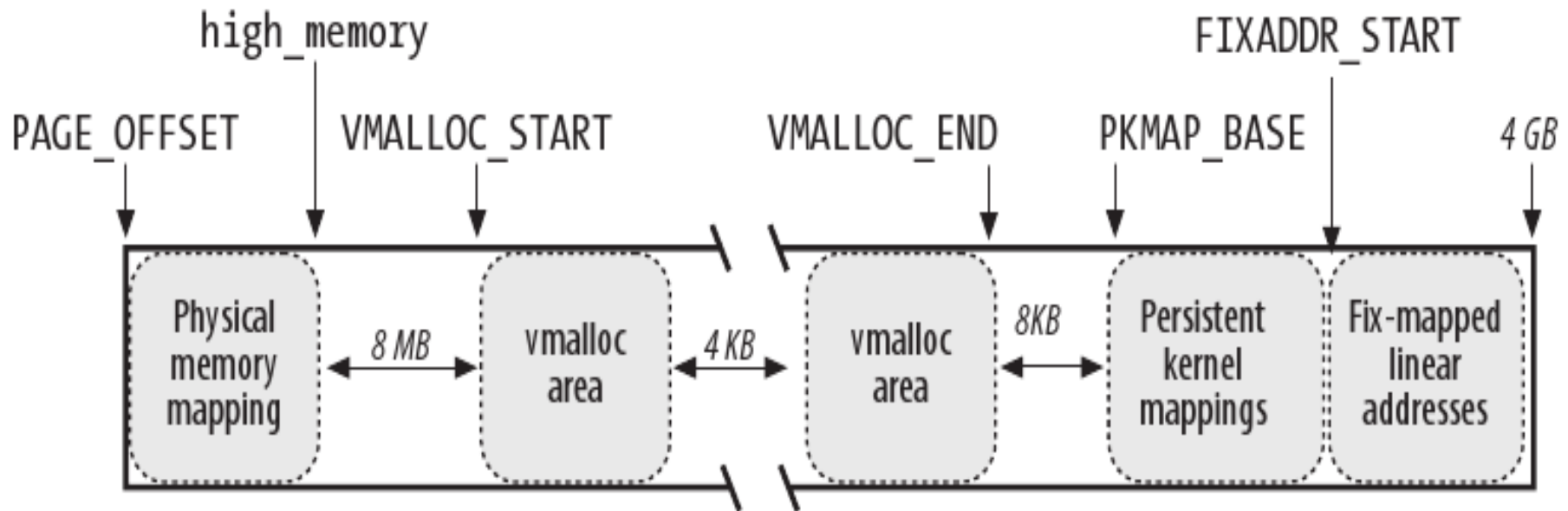
(b) Kernel-ul partajează spațiul de adresă cu procesele utilizator

- Dezavantaje:
 - Procesele au mai puțin spațiu de adresă
 - Kernel-ul are mai puțină memorie fizică mapată direct



- Anumite operații necesită lucrul atât cu adresa fizică cât și cu cea virtuală
- Exemplu: apelul de sistem `write()` are nevoie de
 - Adresa virtuală a buffer-ului kernel: pentru a copia datele din bufer-ul utilizatorului în buffer-ul (cache-ul) kernel
 - Adresa fizică a buffer-ului kernel: pentru a porni un transfer DMA
- Aflarea adresei fizice pentru o adresă virtuală este relativ costisitoare (trebuie parcurse tabelele de pagini)

- Alternativă: maparea lineară în spațiul kernel a memoriei fizice – traslatarea adresă virtuală adresă fizică este redusă la o adunare
- Bonus: se pot folosi pagini de dimensiuni mari (4MB pe x86)
 - Mai puține pagini folosite pentru tabela de pagini
 - O intrare în TLB acoperă o zonă de memorie mai mare
- Bonus 2: se poate afla adresa virtuală pentru o adresă fizică



- Memoria peste 896MB va fi mapată la cerere în spațiul de adrese kernel în zona de 128MB special rezervată
- Trei abordari posibile
 - Mapări permanente
 - Mapări temporare
 - Mapare prin alocare de memorie fizică necontiguă

- Se mapeaza în cei 128MB rezervați, începând de la VMALLOC_START (128MB + 8MB)
- Pentru a “prinde” accesul la zone invalide:
 - Se lasa o “gaura” de 8Mb nemapati in spatiul de adresa
 - Fiecare alocare apoi este separata de catre pagini nemapate

```
void* vmalloc(unsigned long size);  
void vfree(void * addr);  
void *ioremap(unsigned long offset, unsigned  
size);  
void iounmap(void * addr);
```

- Adrese virtuale rezervate la sfârșitul spațiului de adresă kernel (constante)

- Acestor adrese virtuale li se pot asocia adrese fizice cu

```
set_fixmap(idx, phys_addr)
```

```
set_fixmap_nocache(idx, phys_addr)
```

- Setează intrarea din tabela de pagini `idx` către `phys_addr` și setează și flag-ul PCD (page cache disabled) în cazul celei de a doua funcții

```
enum fixed_addresses {
    FIX_HOLE,
    FIX_VDSO,
    FIXDBGP_BASE,
    FIX_EARLYCON_MEM_BASE,
#ifdef CONFIG_X86_LOCAL_APIC
    FIX_APIC_BASE,
#endif
    ...
#ifdef CONFIG_HIGHMEM
    FIX_KMAP_BEGIN,
    FIX_KMAP_END = FIX_KMAP_BEGIN +
        (KM_TYPE_NR * NR_CPUS) - 1,
#endif
}
```

```
inline long fix_to_virt(const unsigned int idx)
{
    if (idx >= __end_of_fixed_addresses)
        __this_fixmap_does_not_exist();
    return (0xffffe000UL - (idx << PAGE_SHIFT));
}
```

- Compilatorul va înlocui acest apel de funcție cu o constantă dacă idx este valid
- Dacă idx este invalid, simbolul `__this_fixmap_does_not_exist()` nefiind definit, se va genera o eroare la link-editarea imaginii

- `kmap_atomic()`, `kunmap_atomic()`
- Folosesc adrese lineare mapate fix
- Pentru fiecare procesor sunt rezervate `KM_TYPE_NR` intrări în tabela de pagini
- Tipuri predefinite:
 - `KM_SKB_SUNRPC_DATA`
 - `KM_SKB_DATA_SOFTIRQ`
 - `KM_BH_IRQ`
 - `KM_USER0`, `KM_USER1`

- Mapările sunt temporare
 - Pot fi folosite doar atât cât contextul în care se execută nu este întrerupt
 - Dacă contextul cedează procesorul se poate întâmpla ca alt context să utilizeze această mapare
- Avantaje:
 - ușor de implementat
 - pot fi folosite în handlerele de întreruperi
 - pot fi folosite de către funcțiile defferable
- Dezavantaje: sunt temporare

- Atunci când se dorește accesarea unei pagini din zona highmem se mapează pe unul din slot-urile dedicate mapărilor temporare

```
void * kmap_atomic(struct page * page, enum km_type type)
{
    enum fixed_addresses idx;
    if (page < highmem_start_page)
        return page->virtual;
    idx = type + KM_TYPE_NR * smp_processor_id( );
    set_pte(kmap_pte-idx, mk_pte(page, 0x063));
    __flush_tlb_one(fix_to_virt(FIX_KMAP_BEGIN+idx));
}
```


- `kmap()`, `kunmap()`
- Mapează pagini din highmem în ultimii 128Mb din spațiul de adresă kernel
- Avantaje: mapările sunt permanente astfel încât contextul care le folosește poate să fie întrerupt
- Dezavantaje: mapările nu se pot face din handler de întrerupere sau funcții defferable
- Este rezervată o întreagă tabelă de pagini pentru asemenea mapări
- Fiecare pagină are asociat un contor
 - 0 - pagina nu este mapată și poate fi folosită
 - 1 - pagina nu este mapată dar nu poate fi folosită pentru că e posibil să fie prezentă în TLB (cu o valoare veche, invalidă)
 - N - pagina este mapată de exact N-1 ori

