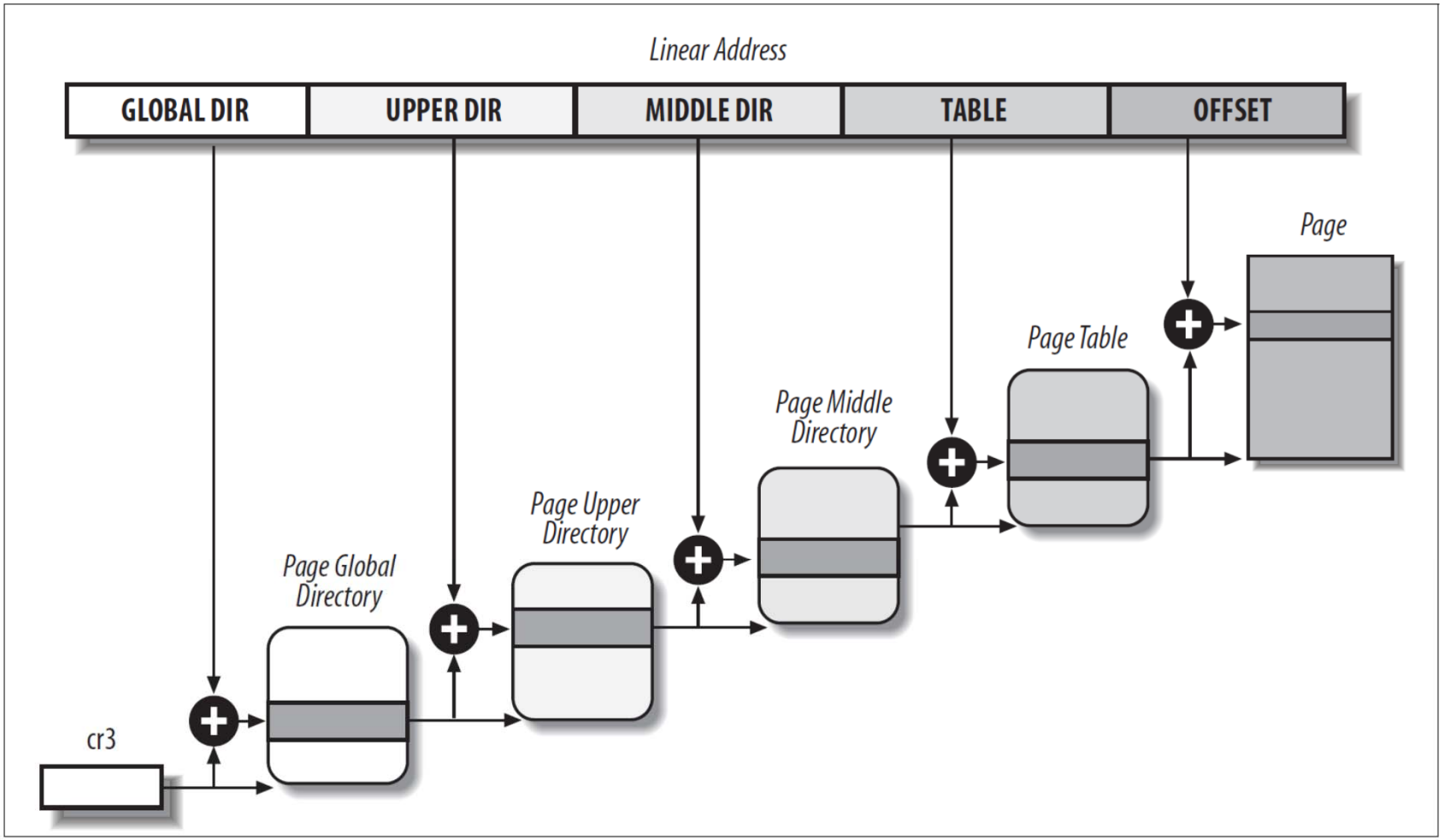
Realizzazione della paginazione in *x64*

la paginazione di *x64*

- l'unità che gestisce la memoria a livello hardware è la *MMU* (*memory management unit*)
- l'indirizzo virtuale (*effective o linear address* in *x64*) si riferisce a uno spazio virtuale di 2^{48} byte, con indirizzo da 48 bit
- il meccanismo di paginazione si basa su pagine di 4 K byte e l'indirizzo si scompone così
  - 12 bit di spiazzamento (*offset*), per i 4 K byte nella pagina
  - 36 bit di *NPV* (*Numero di Pagina Virtuale* – 2^{36} pagine virtuali)
- la Tabella delle Pagine (*TP*) complessiva è organizzata come albero a quattro livelli
  - i 36 bit di *NPV* sono suddivisi in quattro gruppi da 9 bit
  - ciascun gruppo di 9 bit rappresenta lo spiazzamento (*offset*) all'interno di una tabella (*directory*) costituita da 512 righe, ciascuna da 64 bit
  - una riga di queste tabelle si chiama *PTE* – *Page Table Entry*
- dato che ogni *PTE* occupa 64 bit (cioè 8 byte)
  - la dimensione di ogni tabella è di 4 K byte
  - ovvero ogni tabella occupa esattamente una pagina
- l'indirizzo (fisico) della tabella principale è contenuto nel registro *CR3* della *CPU*

struttura degli indirizzi di Linux in *x64*



meccanismo di conversione di *NPV* in *NPF*

- richiede di eseguire quattro accessi aggiuntivi a memoria per ogni accesso utile
  - accedi alla tabella di primo livello tramite *CR3*
  - trova la *PTE* indicata dallo spiazzamento di primo livello di *NPV*
  - leggi a tale spiazzamento l'indirizzo di base della tabella di livello inferiore
  - ripeti la stessa procedura per i livelli successivi
- si noti che gli indirizzi contenuti nei diversi livelli di tabelle sono *indirizzi fisici*, perché la *MMU* li utilizza direttamente per accedere alla memoria fisica
  - il meccanismo che traduce un indirizzo da virtuale a fisico non può a sua volta essere virtuale !
- ciò significa che ciascuna *PTE* (riga di tabella) contiene l'indirizzo fisico (non virtuale) di una tabella a livello inferiore
- dunque la *PTE* di una tabella a livello minimo nell'albero (una foglia) contiene l'indirizzo fisico della pagina finale (codice, dati o pila) a cui si vuole accedere
- l'indirizzo fisico della tabella principale (radice dell'albero) è nel registro *CR3*

bit di protezione (*flag*)

- una *PTE* contiene non solo l'indirizzo di una tabella di livello inferiore, ma anche un certo numero di *bit di protezione* (*flag*) che rappresentano delle proprietà della pagina
- essi sono memorizzati nei 12 bit meno significativi della *PTE* al posto dello spiazzamento, e uno di essi (il 13-esimo) sta nel bit più significativo della *PTE*
- i principali tipi di bit di protezione usati da Linux in una generica *PTE* sono i seguenti

posizione	**sigla**	**nome**	**interpretazione valori**
0	P	present	la PTE ha un contenuto valido
1	R/W	read/write	la pagina è scrivibile: 1=W, 0=ReadOnly
2	U/S	user/supervisor	la pagina appartiene a spazio User: 1=U, 0=S
5	A	accessed	la pagina è stata acceduta (azzerabile da software)
6	D	dirty	la pagina è stata scritta (azzerabile da software)
8	G	global page	vedi sotto (gestione TLB)
63	NX	no execute	la pagina non contiene codice exeguibile

translation Lookaside Buffer (*TLB*)

- quando il processore accede a un indirizzo fisico partendo da un indirizzo virtuale, deve attraversare tutta la gerarchia della *TP* (tutto l'albero) per trovare la *PTE*
- questa operazione, detta *page walk*, richiede ben cinque accessi a memoria per infine leggere o scrivere una sola parola utile (istruzione o dato) di memoria
  - i primi quattro accessi leggono altrettante *PTE* in un percorso radice-foglia nell'albero di tabelle
  - il quinto accesso legge o scrive la parola finale (istruzione o dato) a cui si deve accedere
- per evitare un tale numero di accessi, l'architettura x64 ha un *TLB*, cioè una memoria associativa veloce dove si conservano le corrispondenze *NPV-NPF* più utilizzate di recente
- si può considerare il *TLB* come una sorta di *memoria cache associativa* (*fully associative*) specificamente dedicata alla gestione efficiente della *TP*
- quando si modifica il contenuto del registro *CR3*, la *MMU* invalida tutte le *PTE* del *TLB*, escluse quelle marcate come globali (con il bit di protezione *G* attivo)
- ci sono alcune istruzioni privilegiate per controllare il *TLB*, per esempio per registrare una nuova *PTE* o per invalidare una singola *PTE*

manco di pagina (*page fault*) e *TLB miss*

- il processore emette solo indirizzi virtuali e la *MMU* si incarica di tradurli in fisici
- se una corrispondenza *NPV-NPF* non è registrata nel *TLB,* la *MMU* cerca nella *TP* la *PTE* finale per tale pagina, e poi comunque accede alla pagina *NPF*
- in particolare, dapprima la *MMU* verifica se la corrispondenza richiesta è nel *TLB*
- se la corrispondenza è presente, la *MMU* accede subito alla pagina fisica *NPF*
- se la corrispondenza non è presente, si verifica un *errore di TLB* (*TLB miss*)
- in questo secondo caso la *MMU* svolge le operazioni seguenti:

  esegue un *page walk* (con cinque accessi a memoria) sulla *TP* per recuperare la *PTE* con il numero di pagina fisico *NPF* relativo alla pagina virtuale *NPV*

  il *page walk* è svolto autonomamente dalla *MMU* a livello hardware, senza ricorrere al processore

  se la *PTE* finale è valida, il suo contenuto (cioè il numero di pagina fisico *NPF*) viene registrato nel *TLB* come corrispondenza *NPV-NPF* e subito dopo la pagina fisica viene acceduta

  se la *PTE* non è valida (per vari motivi) si verifica un *page fault*, da gestire come detto prima

esercizio: scomporre l'indirizzo **0x 0000 7fff b0d4 2118**

- lo spiazzamento è 0x 118, corrispondente in decimale a 8 + 16 + 256 = 280
- l'indice in *PT* è 0x 142 costituito dall'ultimo bit della cifra *d* (perché è costituito da 9 bit) seguito da 0x 42
  - 0x 42 in binario è 0100 0010 corrispondente a 64 + 2
  - il primo bit vale 256, quindi 256 + 66 = 322
- l'indice in *PMD* è costituito dagli ultimi 2 bit della cifra *b* (10**11**), seguita da 0, seguita dai primi 3 bit della cifra *d* (**110**1)
  - quindi in decimale è 2 + 4 + 128 + 256 = 390
- procedere in maniera simile per i primi 2 livelli

255:510:390:322

```
offset:          280
indice in PT:    322
indice in PMD:   390
indice in PUD:   510
indice in PGD:   255
```

```
7    F    F    F    B    0    D    4    2    1    1    8
0111 1111 1111 1111 1011 0000 1101 0100 0010 0001 0001 1000
255       510       390       322       280
```

paginazione in Linux

- la gestione della paginazione dipende in grande misura dall'architettura hardware
- Linux utilizza un modello parametrizzato per adattarsi alle diverse architetture
- tale modello caratterizza il comportamento dello *HW* tramite una serie di parametri contenuti nei file di architettura (vedi struttura software più avanti)
- per esempio, in tali file sono descritti
  - la dimensione delle pagine
  - la struttura degli indirizzi
  - il numero di livelli della tabella delle pagine
  - la lunghezza dei diversi spiazzamenti
  - e vari altri aspetti ...

struttura della Tabella delle Pagine di Linux in *x64*

- nel caso di *x64* il modello di Linux è fortemente aderente a quello dello hardware
- si useranno nel seguito i nomi di Linux per designare le diverse tabella delle pagine (*directory*), in forma estesa oppure con gli acronimi *PGD*, *PUD*, *PMD* e *PT*
- la Tabella delle Pagine *TP* è sempre tutta quanta residente in memoria fisica e mappa tutto lo spazio di indirizzamento del processo, sia *user* sia *kernel*

paginazione del Sistema Operativo

- in *x64* tutta la memoria è paginata, indipendentemente dal modo di funzionamento della *CPU*, quindi anche il *SO* e la stessa Tabella delle Pagine sono paginati
- all'avviamento del sistema la Tabella delle Pagine non è ancora inizializzata, quindi deve esistere un meccanismo di avviamento che permetta di arrivare a caricare tale tabella per far partire il sistema:

  all'avviamento del sistema *x64* la paginazione non è attiva

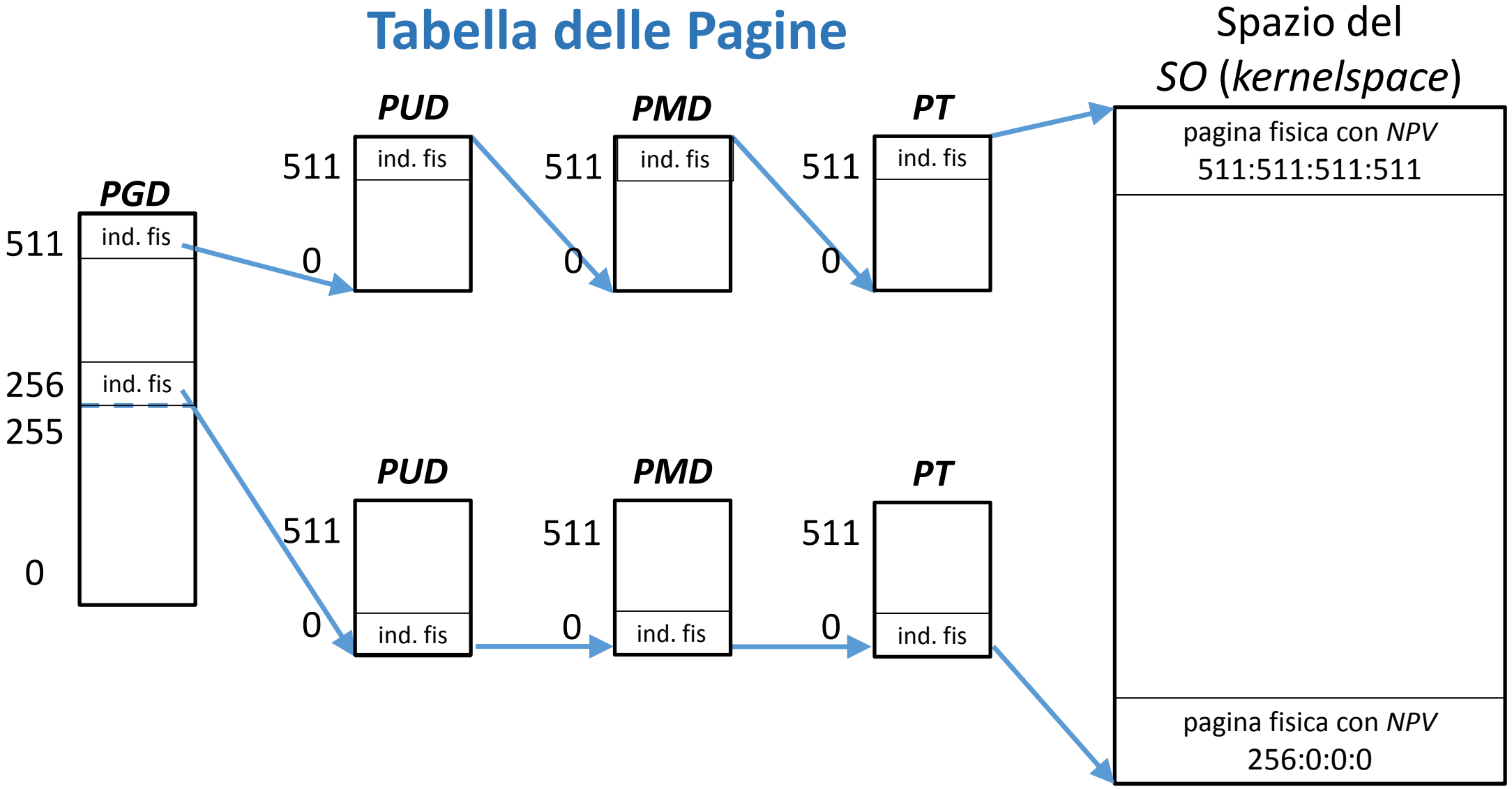
  le funzioni di caricamento iniziale operano accedendo direttamente la memoria fisica, senza rilocazione

  quando è stata caricata una porzione della tabella delle pagine adeguata a fare funzionare almeno una parte del *SO*, il meccanismo di paginazione viene attivato e il caricamento completo del *kernel* viene terminato

Tabella delle Pagine (*TP*) del *kernel*

- la tabella delle pagine attiva è quella puntata dal registro *CR3*, che è unico, quindi non può esistere una *TP* separata per il *SO*
- dato che non esiste una *TP* del *SO*, ma solamente quella dei processi, *il SO viene mappato nella TP di ogni processo*
- lo spazio virtuale è suddiviso esattamente a metà tra modo *U* e modo *S*, la metà superiore della *TP* di ciascun processo è quindi dedicata a mappare il *SO*
- questo fatto non genera ridondanza, perché tutte le metà superiori delle *TP* di ogni processo puntano alla stessa (unica) struttura di sotto-tabelle relativi al *SO*

Tabella delle Pagine



Componenti del *kernel*

Area	Costanti Simboliche per indirizzi iniziali	Indirizzo Iniziale	Dim
Mappatura Mem. Fisica	PAGE_OFFSET	`ffff 8800..`	64 Tb
Memoria dinamica Kernel	VMALLOC_START	`ffff c900..`	32 Tb
Mappatura memoria virtuale	VMEMMAP_START	`ffff ea00..`	1 Tb
Codice e dati	_START_KERNEL_MAP	`ffff ffff 80..`	0,5 Gb
Area per caricare i moduli	MODULES_VADDR	`ffff ffff a0..`	1,5 Gb

Tabella delle Pagine

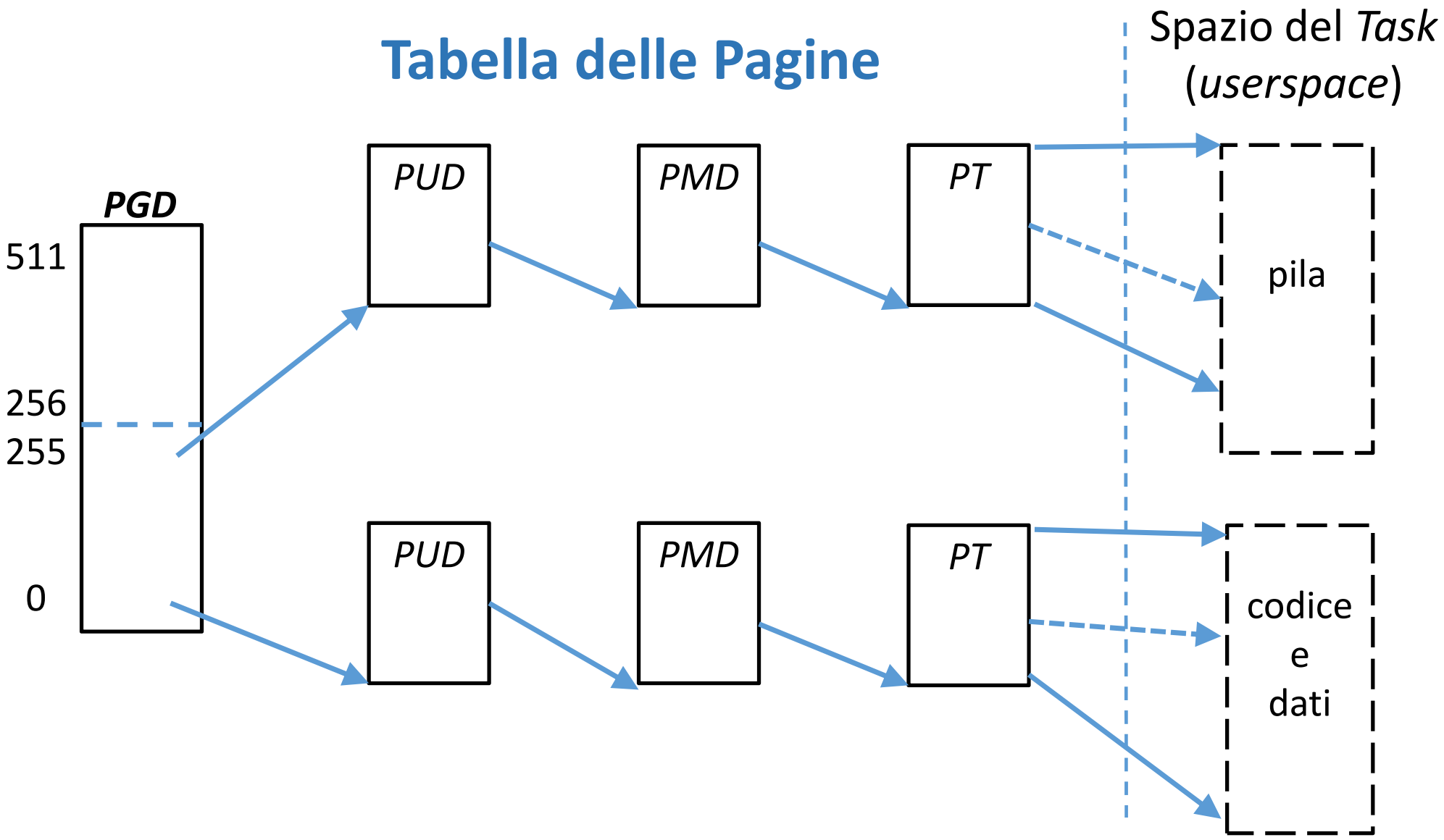
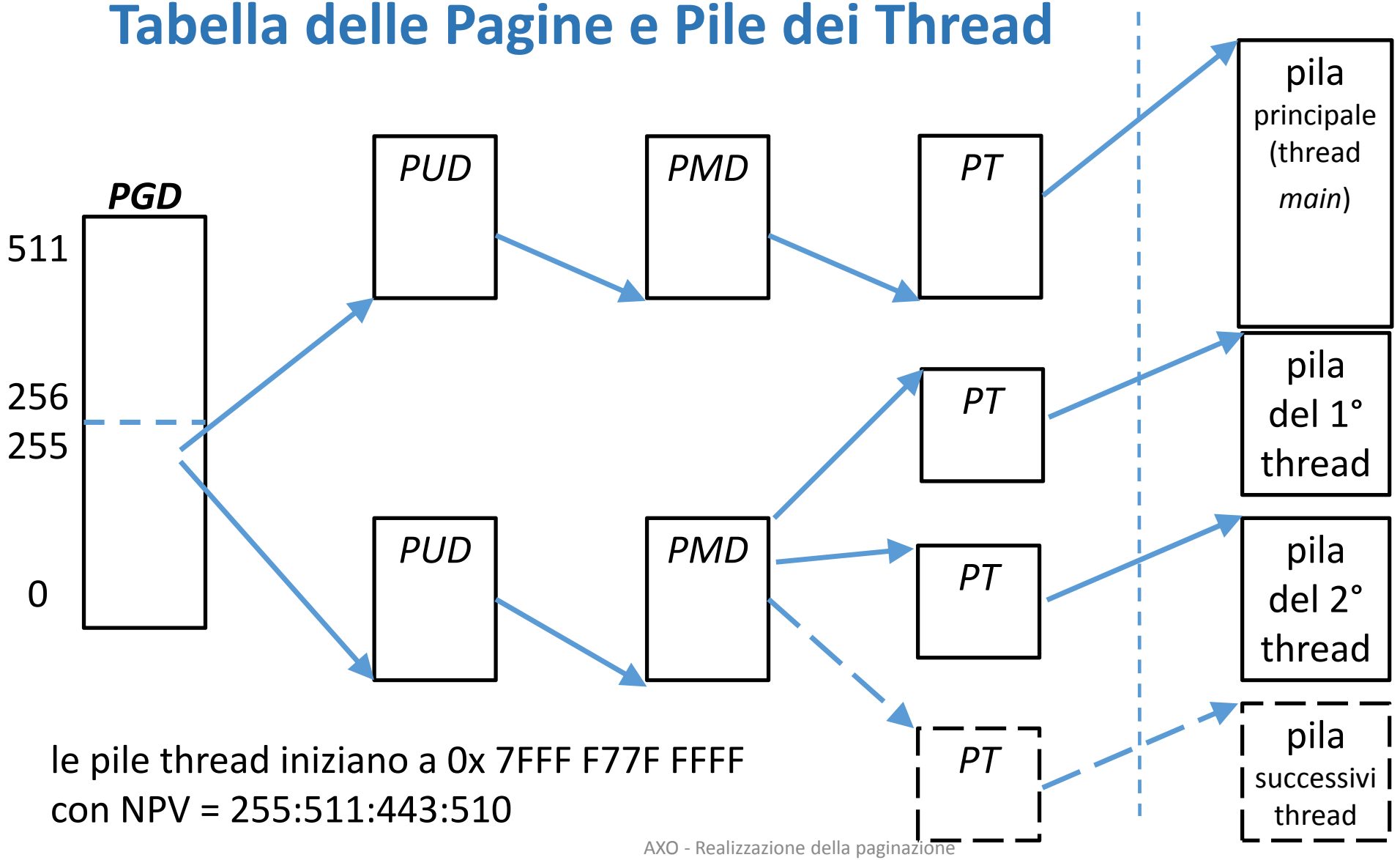


tabella delle pagine e pila dei thread

- indirizzo iniziale delle pile degli eventuali thread è **`0x 7FFF F77F FFFF`** →
- indirizzo decomposto 255:511:443:510
- la loro dimensione è di 8 M byte
- richiedono 4 *PT* per ognuna
- più la pagina di interposizione

Tabella delle Pagine e Pile dei Thread



le pile thread iniziano a 0x 7FFF F77F FFFF
con NPV = 255:511:443:510

dimensione della tabella delle pagine

dimensione di memoria resa accessibile da una singola *PTE* ai diversi livelli della gerarchia

- una *PTE* di *PMD* accede a una pagina di *PT*, quindi $512 \times 4\ \text{K} = 2\ \text{M}$
- una *PTE* di *PUD* accede a una pagina di *PMD*, quindi $512 \times 2\ \text{M} = 1\ \text{G}$
- una *PTE* di *PGD* accede a una pagina di *PUD*, quindi $512 \times 1\ \text{G} = 512\ \text{G} = 0{,}5\ \text{T}$

considerazioni sull'occupazione di memoria della TP

- l'occupazione percentuale di memoria dovuta alla *TP* rispetto al programma diminuisce per programmi grandi fino a scendere nettamente sotto 1 % già per un programma che satura solamente l'ultimo livello di tabelle
- aumentando ancora la dimensione del programma, il peso delle pagine che occupano i livelli superiori di tabelle diventa percentualmente sempre meno rilevante
- il rapporto dimensionale tra *TP* e programma tende a quello fondamentale tra una pagina di PT e la sua area indirizzabile, cioè 1 / 512 = 0,00195, cioè circa 0,2 %
- si osservi che questo rapporto è lo stesso che esiste tra la dimensione di una pagina e quello della *PTE* necessaria a indirizzarla, cioè 4 K byte / 8 byte = 1 / 512

gestione della Tabella delle Pagine

- il *SO* può anche scrivere la *TP*
- le principali funzioni del *SO* per modificare la *TP* sono:
  - funzione per creare una *PTE*: *mk_pte*
    - un numero di pagina e i bit di protezione vengono convertiti in una *PTE* in formato corretto
  - funzioni per allocare e inizializzare pagine della *TP*: *pgd_alloc*, *pud_alloc*, *pmd_alloc*, *pte_alloc*
  - funzioni per liberare la memoria occupata dalla *TP*: *pgd_free*, *pud_free*, *pmd_free*, *pte_free*
  - funzioni per assegnare un valore a una *PTE*: *set_pgd*, *set_pud*, *set_pmd*, *set_pte*
- quando viene richiesto l'accesso a una pagina *NPV* non ancora mappata dalla *TP* il *SO* deve:
  - creare e inserire nella *PT* una nuova *PTE*, se la *PT* esiste già
  - in caso contrario:
    - deve allocare una nuova *PT* (*pte_alloc*)
    - creare e inserire nel *PMD* una nuova *PTE* che punta alla *PT* appena creata, se il *PMD* esiste già
    - in caso contrario, ripetere l'operazione per il livello superiore

esercizio: VMA e Tabella Pagine

Data la struttura di VMA di un processo P che ha creato 4 aree virtuali di tipo M e 2 thread Q ed R, mostrare la struttura della TP a livello delle singole directory

```
PROC: P/Q/R  ************************************
VMA : C  000000400,  3 , R  ,  P  ,  M  , <X,0>
      K  000000600,  1 , R  ,  P  ,  M  , <X,3>
      S  000000601,  4 , W  ,  P  ,  M  , <X,4>
      D  000000605,  2 , W  ,  P  ,  A  , <-1,0>
      M0 000010000,  2 , W  ,  S  ,  M  , <G,2>
      M1 000020000,  1 , R  ,  S  ,  M  , <G,4>
      M2 000030000,  1 , W  ,  P  ,  M  , <F,2>
      M3 000040000,  1 , W  ,  P  ,  A  , <-1,0>
      T1 7FFFF77FB,  2 , W  ,  P  ,  A  , <-1,0>
      T0 7FFFF77FE,  2 , W  ,  P  ,  A  , <-1,0>
      P  7FFFFFFFC,  3 , W  ,  P  ,  A  , <-1,0>
```

```
Struttura della TP
******************
    VMA C   0:  0:  2:  0
            0:  0:  2:  1
            0:  0:  2:  2
    VMA K   0:  0:  3:  0
    VMA S   0:  0:  3:  1
            0:  0:  3:  2
            0:  0:  3:  3
            0:  0:  3:  4
    VMA D   0:  0:  3:  5
            0:  0:  3:  6
    VMA M   0:  0:128:  0
            0:  0:128:  1
    VMA M   0:  0:256:  0
    VMA M   0:  0:384:  0
    VMA M   0:  1:  0:  0
    VMA T 255:511:443:507
          255:511:443:508
    VMA T 255:511:443:510
          255:511:443:511
    VMA P 255:511:511:508
          255:511:511:509
          255:511:511:510
```