Sistema Operativo - Gestione della Memoria per moderne CPU.

Address Binding Dinamico in esecuzione mediante Segmentazione, Paginazione e Memoria Virtuale (Swap su Disco

Lista Argomenti

- Concetto di Address Binding
- Istruzioni assembly
  - salti assoluti
  - salti relativi
- Linking
  - Rilocazione Statica
  - Istruzioni rilocabili (salti relativi)
  - Istruzioni Non rilocabili (salti assoluti)
  - Indirizzi Logici nel programma.
- Loading (caricamento del programma in memoria per l'esecuzione)
  - Rilocazione Dinamica (collocazione dei segmenti in memoria, creaz. Tabella seg.)
  - Indirizzi (Lineari o Virtuali) nel processo.
- MMU – Memory Management Unit
  - HW dedicato al calcolo degli indirizzi fisici a partire dagli indirizzi logici.
- Segmentazione
  - Tabella dei descrittori di segmento
  - Base e Limite del Segmento
  - Segmentation Fault
- Paginazione e Swap su disco (memoria virtuale)
  - Tabella delle Pagine
  - Indirizzi Fisici su Ram
  - Page Fault

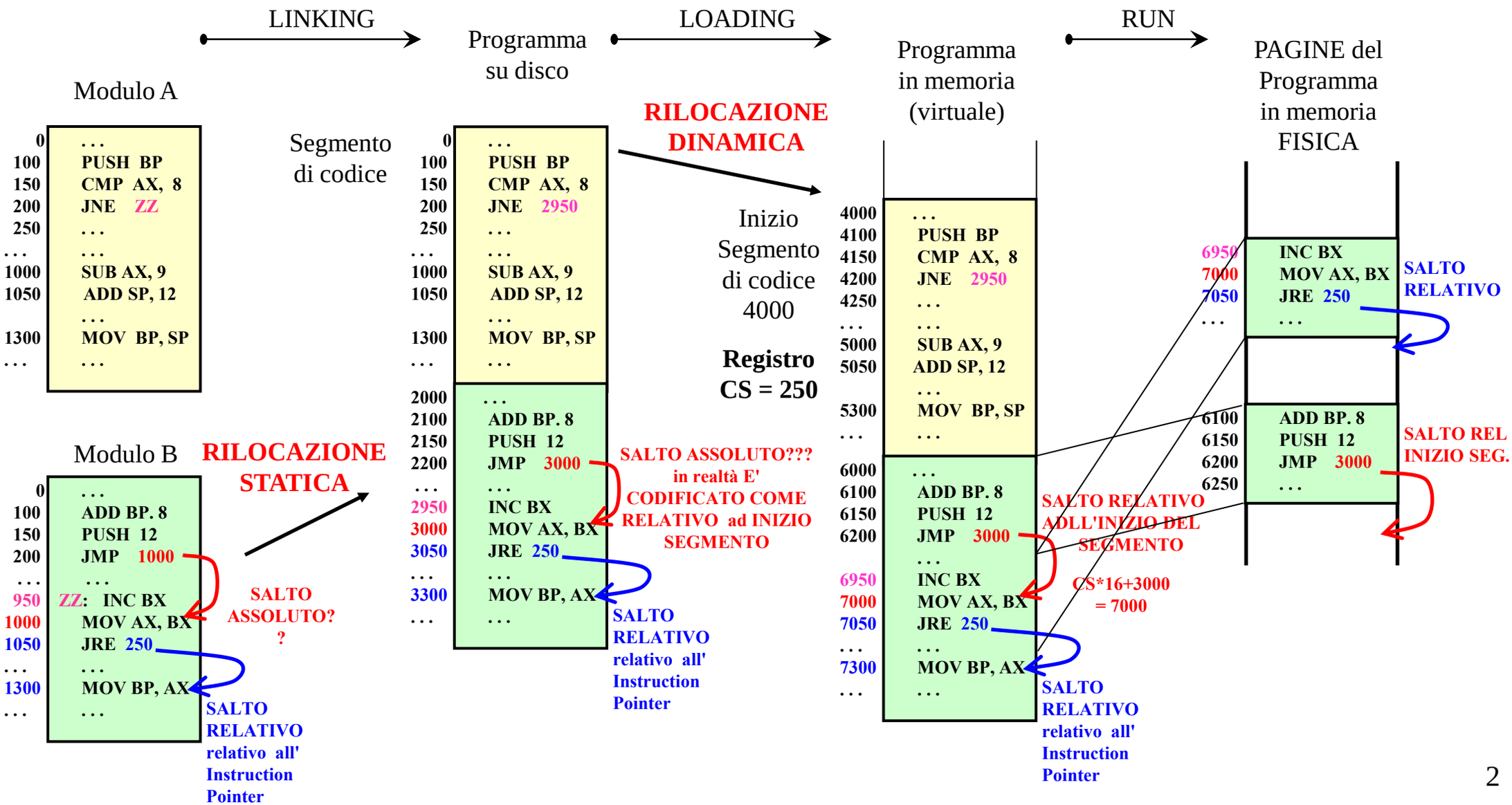
Sistema Operativo - Gestione della Memoria
Tipi di Indirizzi e Rilocazione (Binding)

INDIRIZZI SIMBOLICI (label) o **LOGICI** specificati rispetto all'inizio del modulo

INDIRIZZI LOGICI calcolati rispetto all'inizio del relativo segmento

INDIRIZZI LINEARI (o VIRTUALI) calcolati rispetto all'inizio del relativo segmento in memoria virtuale

INDIRIZZI FISICI Dove si trova realmente in Memoria fisica



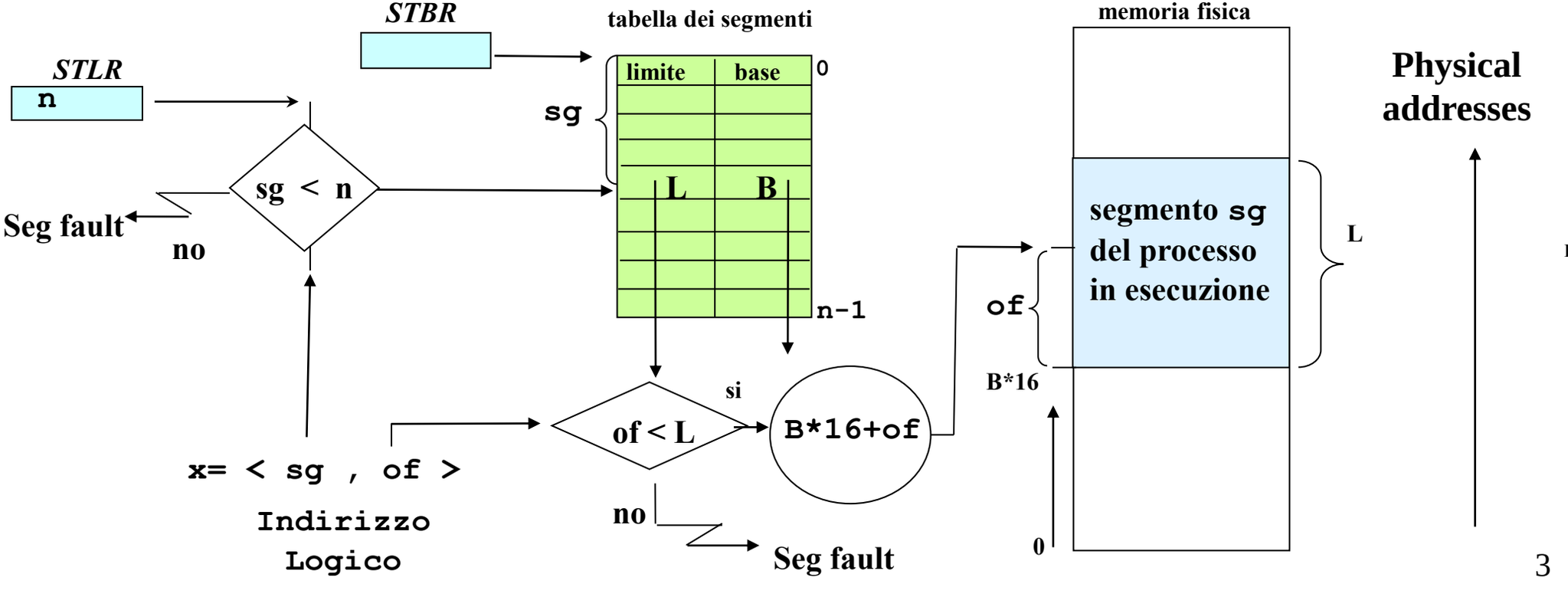
MMU e Calcolo degli indirizzi Fisici

Segmentazione pura e calcolo degli indirizzi fisici nell'8088

- **Indirizzi Logici formati da due parti: Segmento e Offset**
  - **Il Segmento moltiplicato per 16 indica l'inizio del segmento.**
  - **L'Offset e' lo scostamento rispetto all'inizio del segmento**
  - la MMU somma all'indirizzo di inizio di quel segmento l'Offset; **la somma e' l'indirizzo fisico**.

  Indirizzo Fisico = segmento * 16 + offset

Segmentazione pura e calcolo degli indirizzi fisici in IA32 (evoluzione, tabella dei segmenti) (descrizione testuale nella prossima slide)

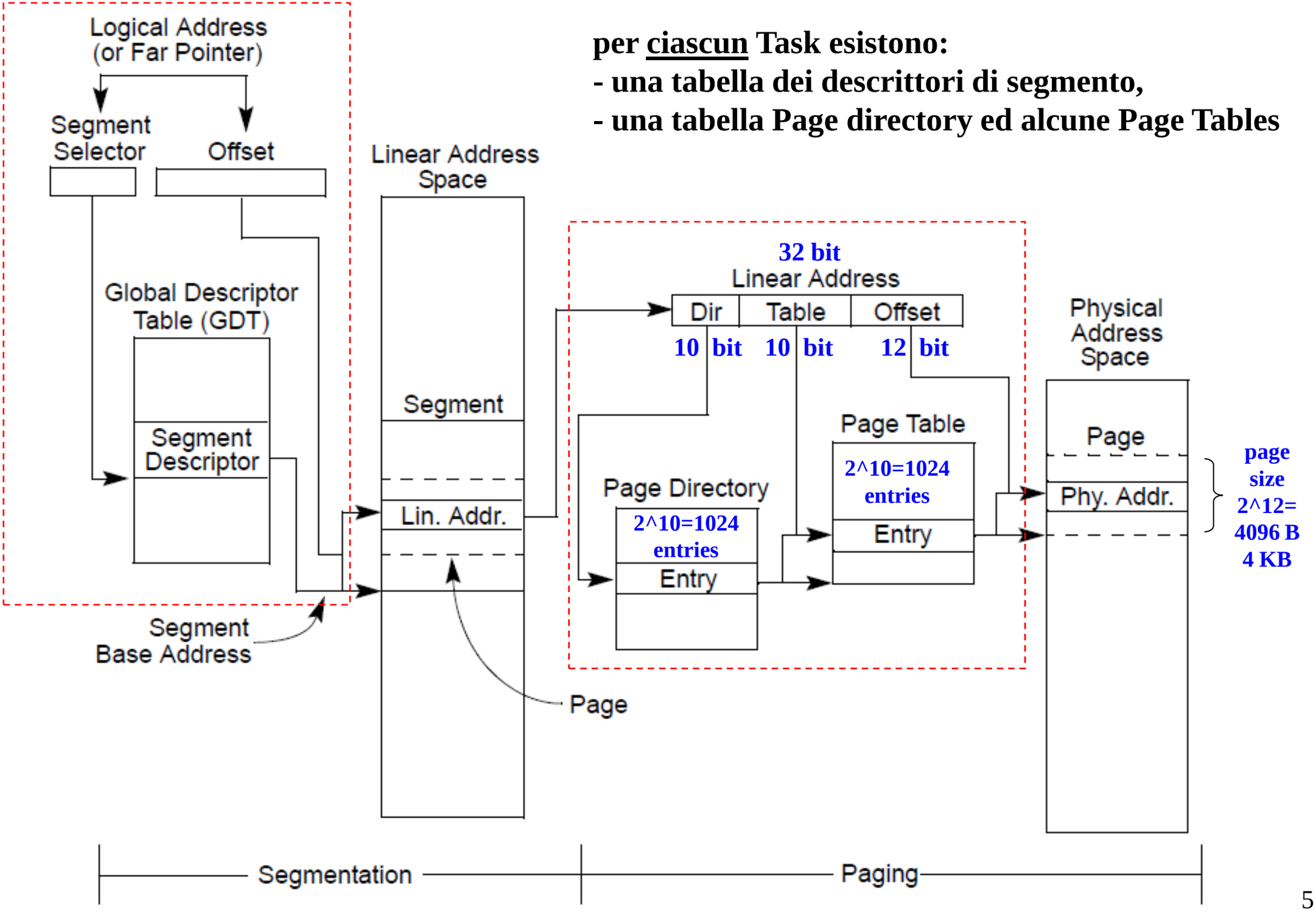


Segmentazione pura (niente paginazione) e calcolo degli indirizzi fisici in IA-32 (Selettore di Segmento e Tabella dei Segmenti – vedi figura precedente)

- **Indirizzi Logici formati da due parti: Selettore di Segmento e Offset**
  - **Il Selettore di Segmento non e' un indirizzo ma e' un Indice di una tabella.**
  - **L'Offset e' lo scostamento rispetto all'inizio del segmento.**
- **MMU: Memory Management Unit - Hardware dedicato al calcolo degli indirizzi fisici a partire da indirizzi logici.**
- Esiste in memoria una Tabella dei Processi
- **Per ciascun Processo** esiste in memoria una **Tabella dei Segmenti** di quel processo.
- Esistono due registri specializzati, STBR e STLR
  - Segment Table Base Register (**STBR**) contiene l'indirizzo di inizio della tabella dei segmenti del processo in esecuzione
  - Segment Table Limit Register (**STLR**) contiene la dimensione della tabella dei segmenti del processo in esecuzione.
- Nella tabella dei Segmenti, per ciascun Segmento viene memorizzato l'indirizzo di inizio (è multiplo di 16) del segmento (Base) (diviso 16) e la sua dimensione (Limite) oltre ai permessi di accesso.
- La MMU riceve un indirizzo formato da Selettore di Segmento e Offset
- Usa il valore del segmento come indice per accedere alla Tabella dei segmenti del processo in esecuzione.
- Controlla se il processo ha il permesso per accedere al segmento.
  - Se non ha il permesso causa Segmentation Fault (processo killato).
- Controlla che l'Offset non superi il limite di quel segmento
  - Se l'Offset cade fuori dal segmento allora causa Segmentation Fault (processo killato).
- Se l'Offset rimane dentro il segmento allora la MMU calcola l'indirizzo Fisico, corrispondente all'indirizzo Logico, nel seguente modo :
  - la MMU preleva dalla Tabella dei Segmenti l'indirizzo Base per quel Segmento.
  - la MMU moltiplica per 16 l'indirizzo Base di quel segmento e somma l'Offset; **la somma e' l'indirizzo fisico**.

Segmentazione Paginata in IA-32

per <u>ciascun</u> Task esistono:
- una tabella dei descrittori di segmento,
- una tabella Page directory ed alcune Page Tables



Segmentazione Paginata in IA-32

In IA-32, quando viene utilizzata la segmentazione paginata, **per ciascun task** (ad es per ciascun processo) vengono utilizzati dal sistema operativo:

- una tabella dei descrittori di segmento,
- una tabella con la directory delle pagine i cui elementi puntano ad alcune tabelle delle pagine.

Perche' fare Paginazione a due (o piu') Livelli?

La paginazione a più livelli è una tecnica introdotta per affrontare il problema relativo a tabelle delle pagine di dimensioni troppo elevate, dovute all'elevato numero di pagine indirizzabili, per spazi logici di indirizzamento estesi.

Ad esempio: usando indirizzi di 32 bit (spazio logico di 4 GB), con dimensione pagina 4KiB (12 bit per dimensione di pagina), la tabella delle pagine dovrebbe contenere 2^32/2^12 elementi-> **2^20 elementi** (**circa 1 Milione di elementi**)!

La paginazione a più livelli permette di risolvere il problema, consentendo una allocazione **non contigua** della tabella delle pagine. In altre parole, si applica ancora la paginazione alla tabella della pagine.

Segmentazione Paginata in IA-32: Precisazioni&Curiosità: Domande

Quali indirizzi vengono specificati dalle istruzioni di un programma in esecuzione? Virtuali o Fisici ?

Quali indirizzi vengono spediti sul bus degli indirizzi? Virtuali o Fisici ?

Se debuggo un programma che usa puntatori, che indirizzi vedo nelle variabili puntatori? Virtuali o Fisici?

Supponiamo di scrivere il seguente programma (stampapuntatore.c) , generare l'eseguibile ed eseguirlo:

```
int main() {  int i;  int *p;  i=17;   p=&i;     printf( " i=%d   p=%p \n", i, p  ); return(0); }
```

Se io eseguo piu' volte lo stesso programma, mi stampa sempre lo stesso output?

Proviamo ora ad aggiungere in fondo al programma uno sleep(1000) che mantiene in memoria il programma per molto tempo.

```
int main() { int i;  int *p;  i=17;  p=&i;   printf( " i=%d   p=%p \n", i, p  ); sleep(1000); return(0); }
```

Lanciare ripetutamente in background il programma, per avere contemporaneamente in esecuzione più copie di uno stesso programma, e vedere se produce ogni volta lo stesso output.
Producono tutti lo stesso output? Perche'?

Segmentazione Paginata in IA-32: Precisazioni&Curiosità: Risposte

Quali indirizzi vengono specificati dalle istruzioni di un programma in esecuzione? **Virtuali**
Quali indirizzi vengono spediti sul bus degli indirizzi? **Fisici**
Se debuggo un programma che usa puntatori, che indirizzi vedo nelle variabili puntatori? **Virtuali**

Gli indirizzi espressi in un programma C sono indirizzi virtuali.
Anche gli indirizzi espressi in un programma in linguaggio macchina sono indirizzi virtuali.
Debuggando il programma vedo indirizzi virtuali, non indirizzi fisici, poiche\ l'opera della MMU e' nascosta alla applicazione.
Gli indirizzi specificati nel codice macchina sono indirizzi virtuali, non fisici: sono trasformati in indirizzi fisici in modo diverso a seconda delle tecniche di gestione della memoria adottate dal sistema operativo ed eseguite dalla MMU.

int main() { int i; int *p; i=17; **p=&i**; printf(" i=%d **p=%p** \n", i, (void*)**p**); return(0); }

Il contenuto della variabile puntatore p , che viene stampato a video, e' un indirizzo virtuale
Se io eseguo piu' volte lo stesso programma, aspettando ogni volta la terminazione del programma precedente, **mi stampa sempre lo stesso output. SI** (*)
Provare ad aggiungere in fondo al programma uno sleep(1000) che mantiene in memoria il programma per molto tempo.
Lanciare ripetutamente in background il programma, per avere contemporaneamente in esecuzione piu\ copie di uno stesso programma, e vedere se produce ogni volta lo stesso output.

int main() { int i; int *p; i=17; **p=&i**; printf(" i=%d **p=%p** \n", i, (void*)**p**); sleep(1000); return(0); }

Lanciando due o più istanze del medesimo processo in esecuzione concorrente, in output si hanno gli stessi indirizzi , poiché ogni programma vede un proprio spazio di indirizzi, separato ma uguale a quello delle altre istanze dello stesso processo.

Segmentazione Paginata in IA-32: Precisazioni&Curiosità: Generiamo Dubbi e Risolviamoli

verifico il tipo di configurazione del kernel

sysctl kernel.randomize_va_space

probabilmente mi stampa a video

kernel.randomize_va_space = 2

configuro il funzionamento del loader del kernel nella maniera primitiva

sudo **sysctl -w kernel.randomize_va_space=0**

eseguo più volte il programma e guardo che succede

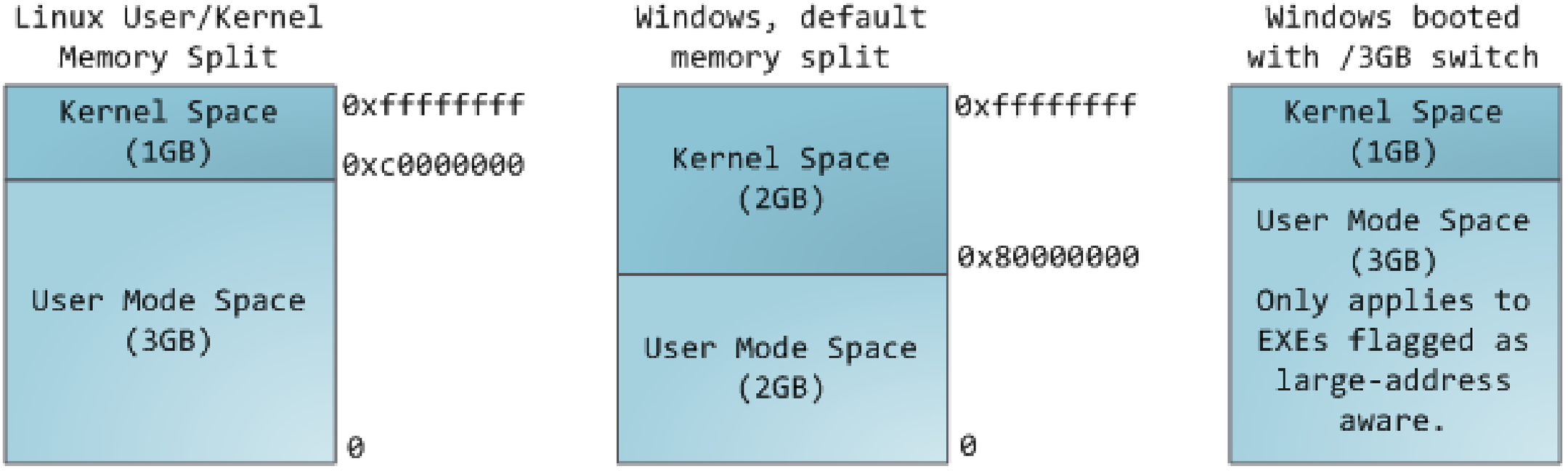
riconfiguro il funzionamento del loader del kernel nella maniera primitiva

sudo **sysctl -w kernel.randomize_va_space=2**

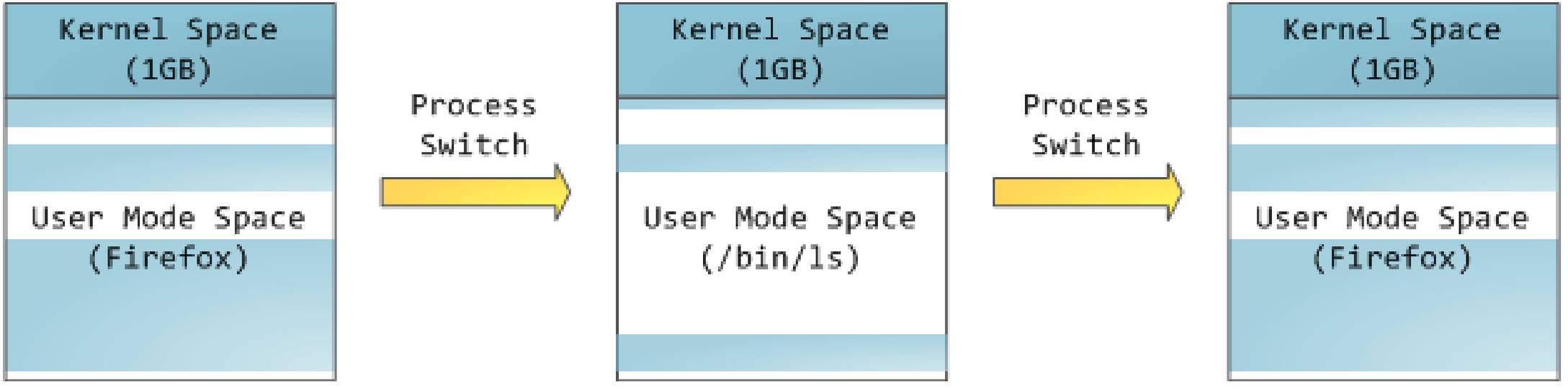
eseguo più volte il programma e guardo che succede.

discussione .....

Separazione tra Spazio Kernel e Utente in Memoria Virtuale

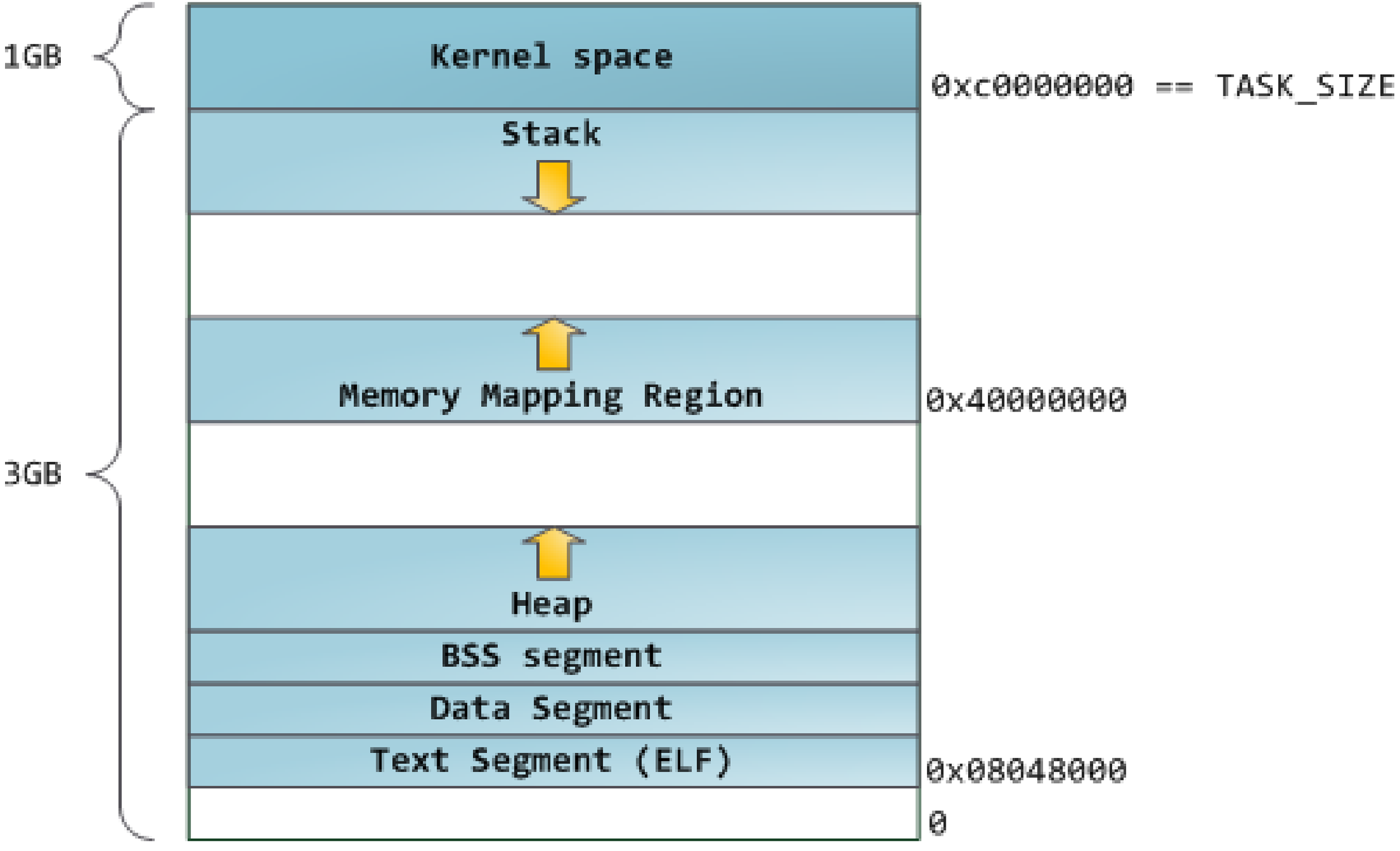


Switch "virtuale" di contesto da un processo all'altro.
Memoria Virtuale



Tratto da : **Anatomy of a Program in Memory , Gustavo Duarte Jan 27th, 2009 .**
http://duartes.org/gustavo/blog/post/anatomy-of-a-program-in-memory/

Separazione tra Spazio Kernel e Immagine Memoria Virtuale



Tratto da : **Anatomy of a Program in Memory , Gustavo Duarte Jan 27th, 2009 .**
http://duartes.org/gustavo/blog/post/anatomy-of-a-program-in-memory/

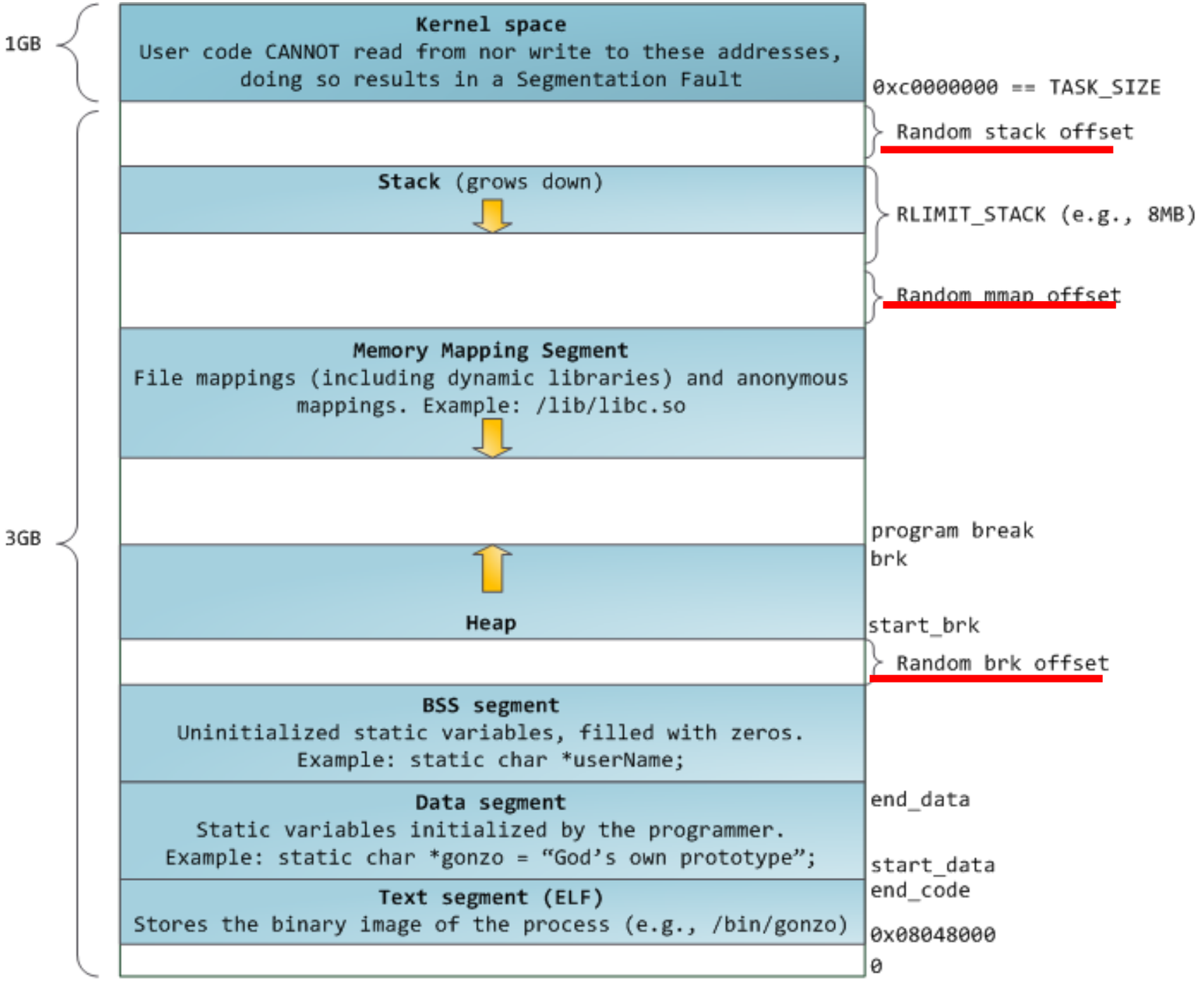
Exploit e Address Space Layout Randomization (1)

Un exploit è un codice che, sfruttando una vulnerabilità di un sistema, porta all'esecuzione di codice non voluto dal sistema, ottenendo privilegi maggiori di quelli a cui si avrebbe diritto.
Alcuni tipi di exploit sfruttano il fatto di conoscere la locazione di memoria (virtuale) in cui viene collocato il codice di alcune funzioni e di alcune DLL.

Per proteggersi da questi tipi di attacchi, i sistemi operativi possono modificare, al caricamento di un programma da eseguire, la collocazione dei segmenti in memoria virtuale aggiungendo una quantita' casuale agli indirizzi base dei segmenti. In tal modo tutti gli indirizzi virtuali specificati in un eseguibile vengono a trovarsi sfalsati rispetto a quanto specificato nell'eseguibile su file.
Si parla percio' di **ASLR (Address Space Layout Randomization).**

Da ormai 7-8 anni, Linux rende causale l'indirizzo di inizio dello stack, dello heap e dei segmenti di memoria allocati dinamicamente (ad es per librerie caricate dinamicamente o per segmenti di memoria condivisa tra processi) aggiungendo un offset all'indirizzo di inizio dei segmenti da randomizzare.

Exploit e Address Space Layout Randomization (2)



Tratto da : **Anatomy of a Program in Memory , Gustavo Duarte Jan 27th, 2009 .**
http://duartes.org/gustavo/blog/post/anatomy-of-a-program-in-memory/

Exploit e Address Space Layout Randomization (3)

Si puo' verificare se il nostro kernel Linux e' impostato per randomizzare la collocazione dei segmenti, sfruttando l'utility di sistema sysctl :

```
sysctl    kernel.randomize_va_space
```

probabilmente mi stampa a video

```
kernel.randomize_va_space = 2
```

che indica che il kernel effettua la randomizzazione nel modo descritto.

Si puo' pero' riconfigurare a run-time il funzionamento del kernel sfruttando l'utility di sistema sysctl cosi' :

```
sudo      sysctl   -w   kernel.randomize_va_space=0
```

Tale ordine imposta il kernel di Linux affinche' il loader si comporti nella maniera primitiva, non effettuando la randomizzazione del layout della memoria.