Adressage mémoire et protection

2017 – 2018

Florent Gluck – Florent.Gluck@hesge.ch

Version 0.5

h e p i a
Haute école du paysage, d'ingénierie
et d'architecture de Genève

Hes·so GENÈVE
Haute Ecole Spécialisée
de Suisse occidentale

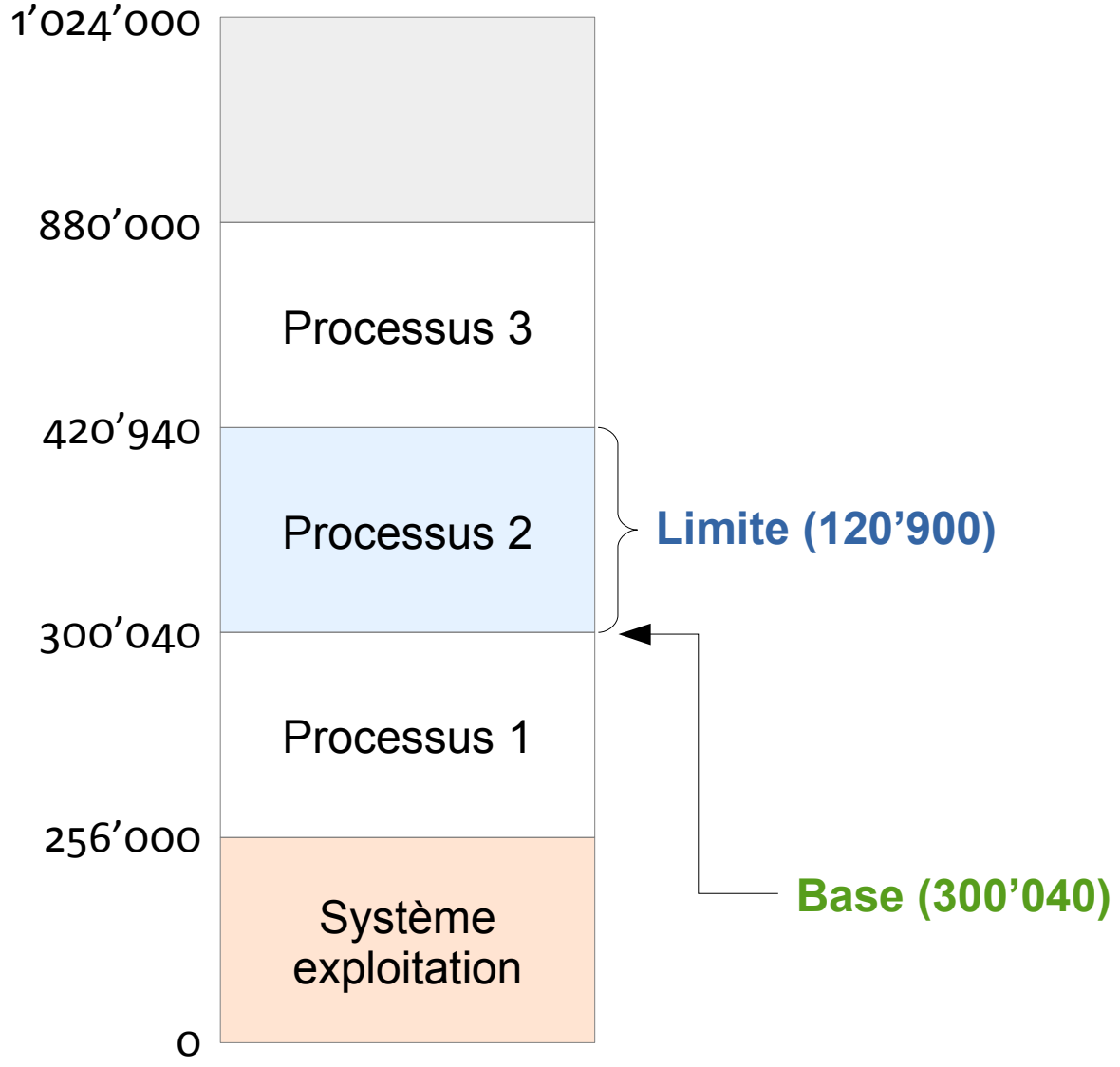
Exécution d'un programme

- Programme doit être lu depuis le disque et chargé en mémoire pour être exécuté.
- CPU peut seulement accéder : mémoire (RAM) et registres.
- Protection mémoire **nécessaire** pour assurer la fiabilité du système !

Concept : base et limite

L'espace mémoire adressable par une processus est défini par la paire :

- **Base**
- **Limite**



Adresses mémoires et relogemement

Adresses des instructions et données d'un processus en mémoire peuvent être établies à trois moments :

- A la **compilation** :
  - Si adresse mémoire connue à l'avance → code à adressage absolu peut être généré.
  - Le code doit être recompilé si l'adresse change.
- Au **chargement** :
  - Code relogeable doit être généré si adresse mémoire pas connue à la compilation.
- A l'**exécution** :
  - Etablissement repoussé à l'exécution si processus peut être déplacé d'un segment à un autre durant son exécution.
  - Nécessite hardware dédié (ex : registres base et limite).

Relogement et protection

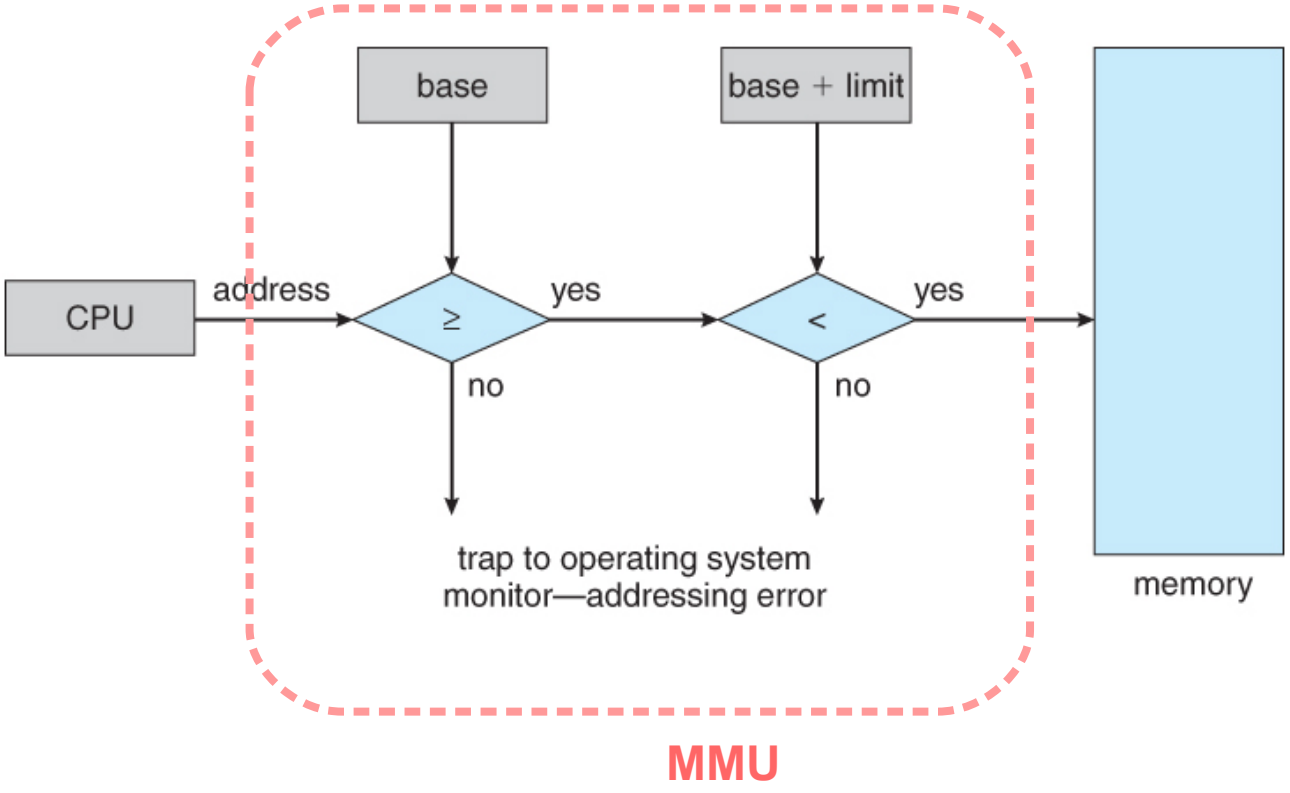
- Aucune certitude où un programme sera chargé en mémoire
  - Les adresses des données et du code ne peuvent pas être absolues.
  - Les espaces d'adressage des processus doivent être **mutuellement exclusifs**.
- Utilisation des valeurs base et limite
  - La valeur de base est ajoutée à toutes les adresses référencées par le CPU → mapping adresses virtuelles en adresses physiques.
  - Si des adresses sont référencées au dessus de la limite → dépassement de limite → **erreur** !

Memory Management Unit (MMU)

- Dispositif matériel réalisant le mapping d'adresses **virtuelles** en adresses **physiques**.
- A chaque référencement mémoire, le MMU s'occupe d'ajouter la valeur de base et de vérifier que la limite n'est pas dépassée.
- Un programme utilisateur ne voit que des adresses virtuelles et ne voit donc jamais des adresses physiques.
  - En réalité, **le CPU ne voit que des adresses virtuelles**.

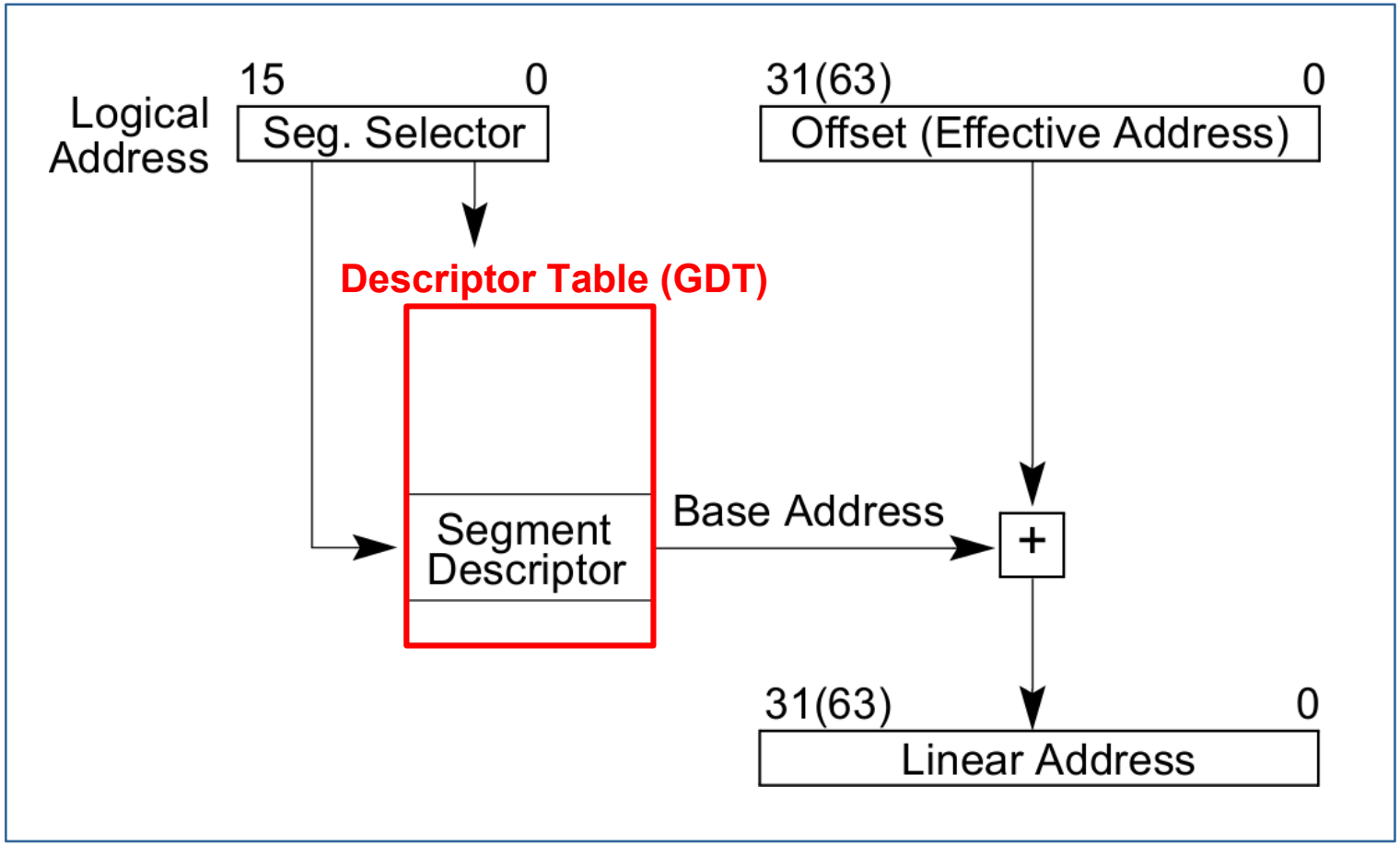
Protection mémoire

Protection mémoire hardware utilisant le concept de base et limite, grâce au MMU :

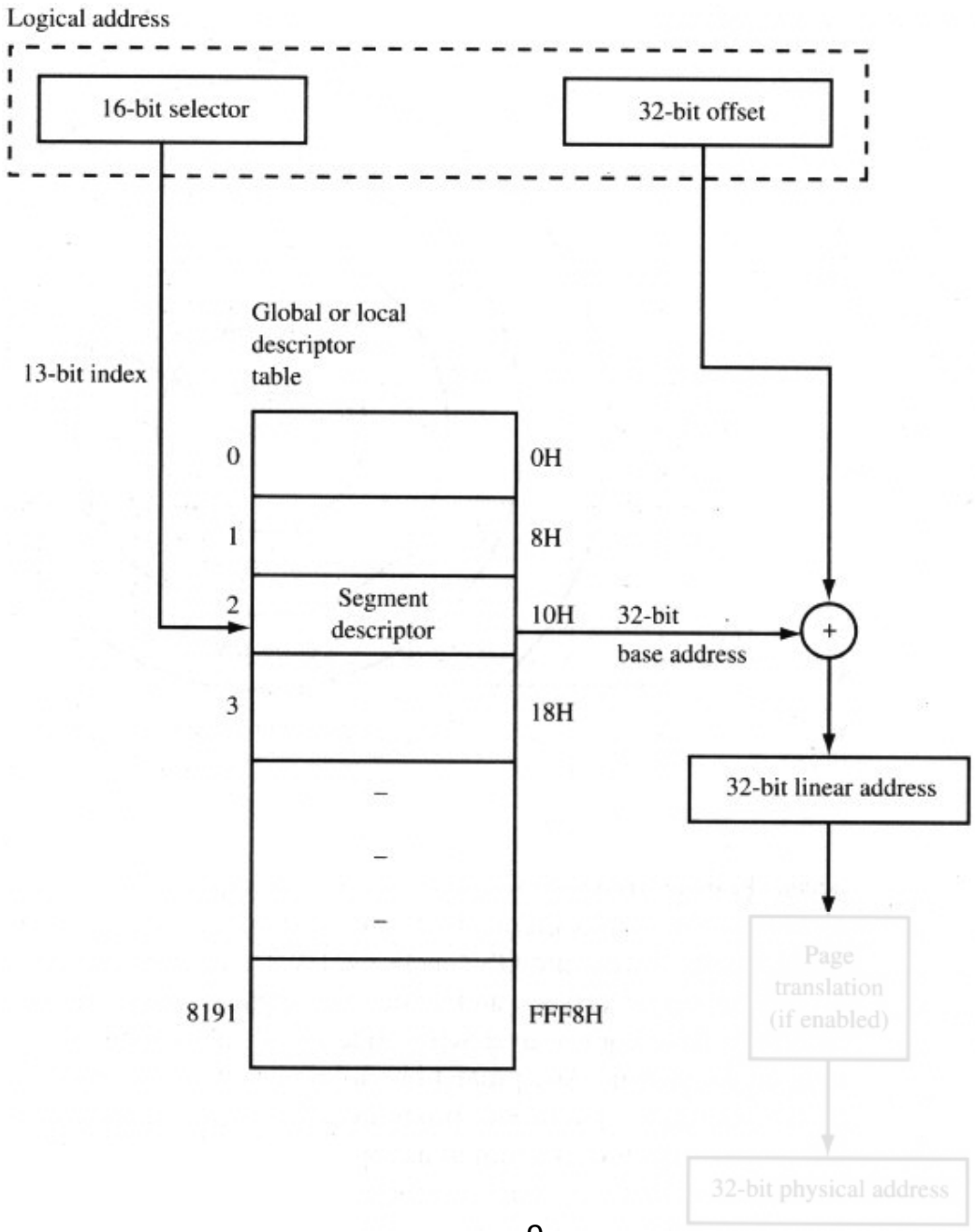


IA-32 : Protection mémoire

Translation d'adresses logiques en adresses physiques sur l'architecture IA-32 :

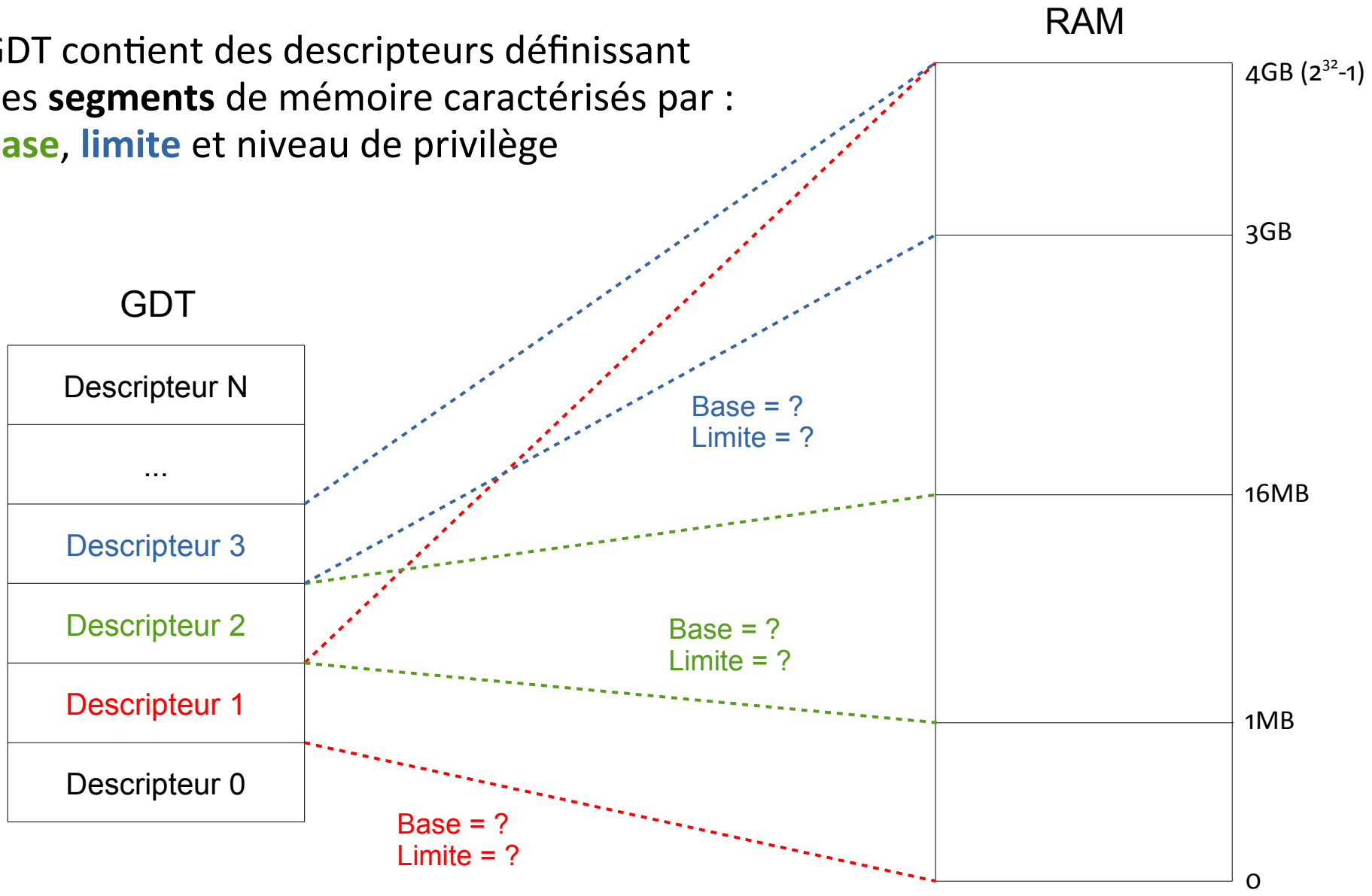


Adressage segmenté avec la GDT



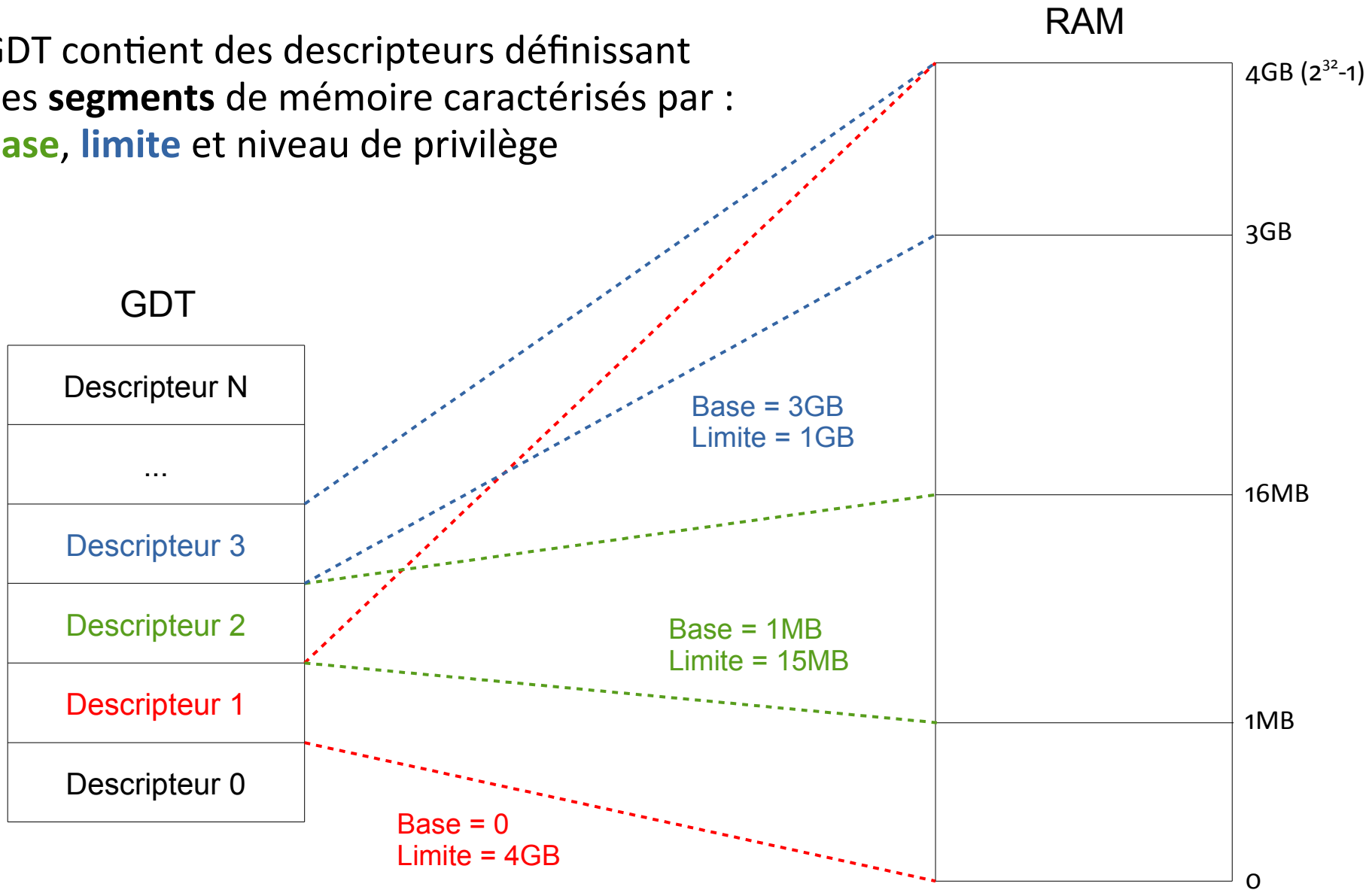
GDT : Global Descriptor Table

GDT contient des descripteurs définissant des **segments** de mémoire caractérisés par : **base**, **limite** et niveau de privilège



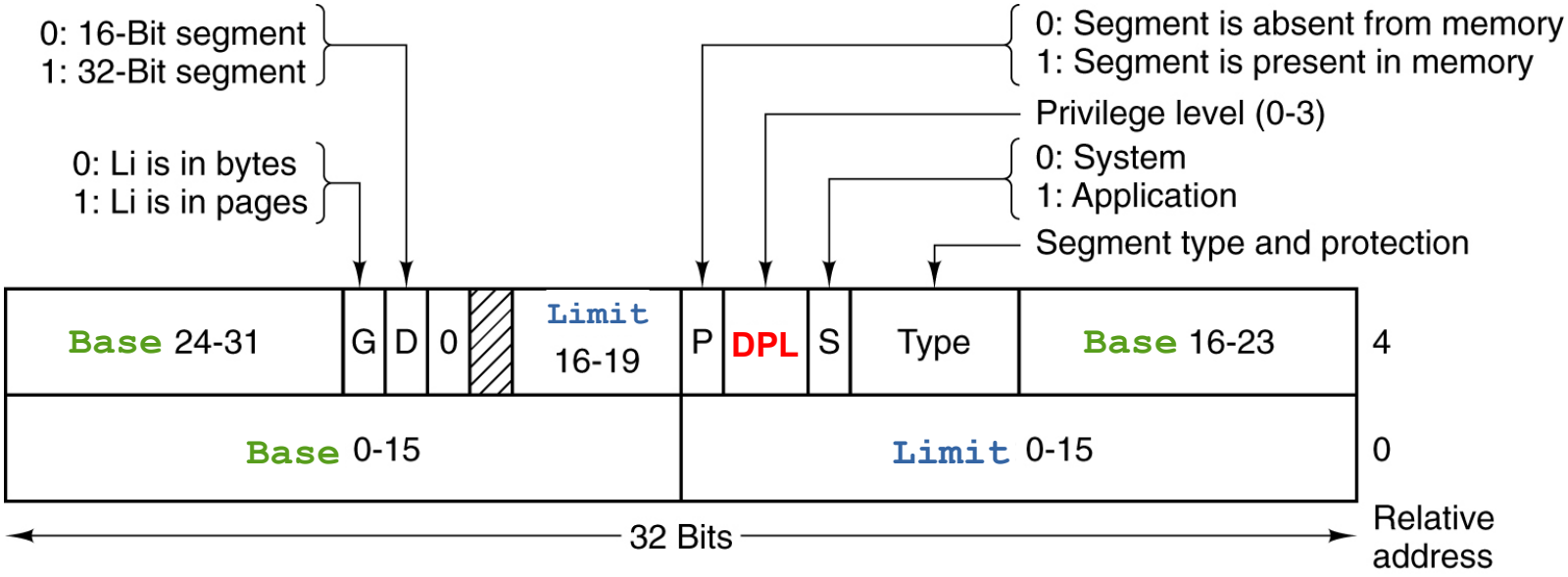
GDT : Global Descriptor Table

GDT contient des descripteurs définissant des **segments** de mémoire caractérisés par : **base**, **limite** et niveau de privilège



GDT et descripteur de segment

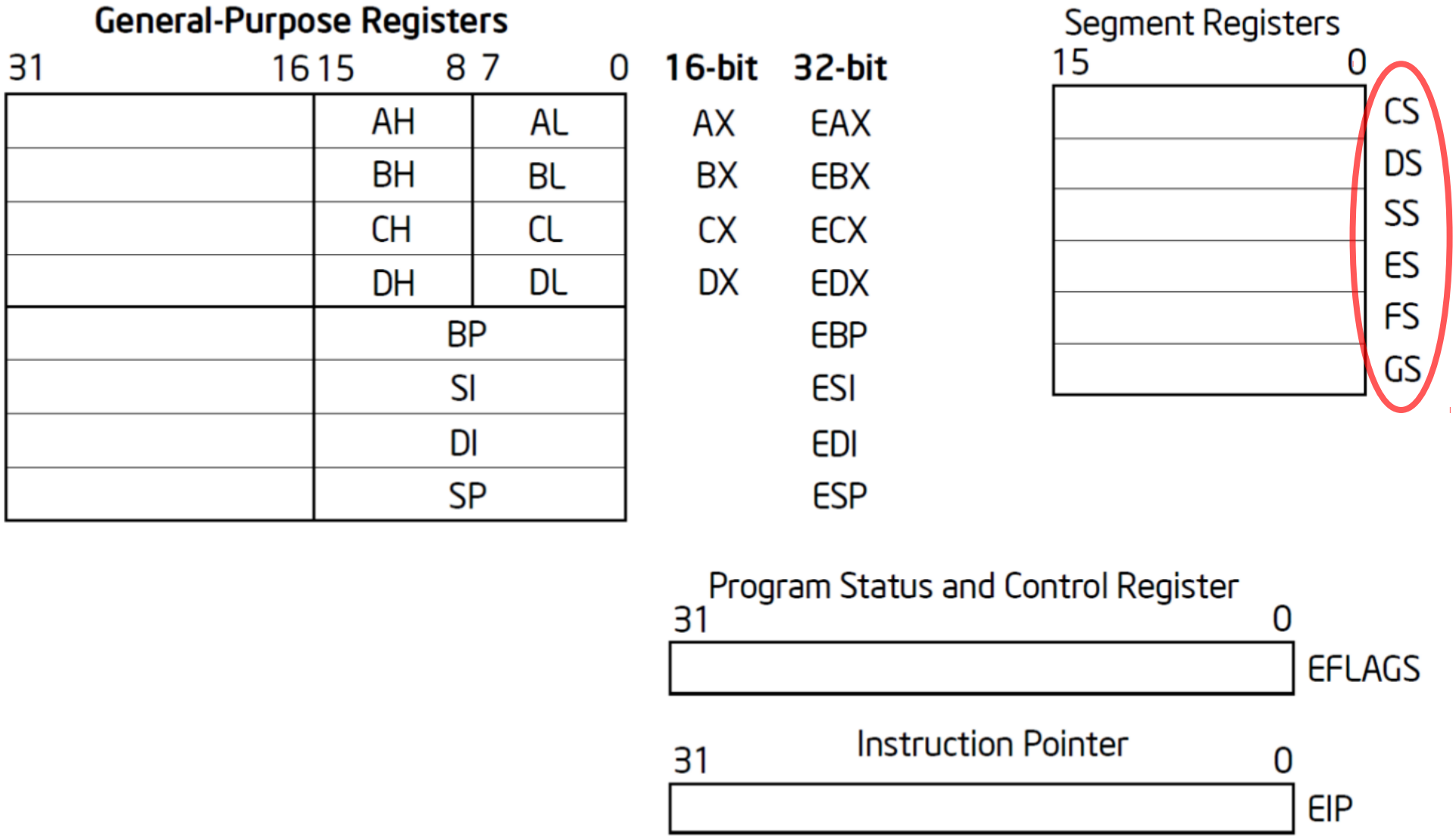
- La GDT (Global Descriptor Table) est une table résidant en mémoire
- Contient des entrées de 64-bits appelées **descripteurs**
- La GDT peut contenir différents types de descripteurs
- Nous présentons ici les descripteurs de segments
- Chaque descripteur de segment défini une zone mémoire contigüe caractérisée par une **base**, **limite** et un niveau de privilège (**DPL**) :



Limite

- Limite du segment encodée sur 20-bits ⟶ limite max de 1MB ?
- Comment indiquer une limite de 4GB (2^{32}) ?
- Le bit G du descripteur indique la granularité du champ limite :
  - Si G = 0 ⟶ granularité de 1 byte ⟶ $limite_{segment}$ = 1 * **Limit**
    ⟶ $limite_{segment}$ max = 2^{20} = 1MB
  - Si G = 1 ⟶ granularité de 4KB ⟶ $limite_{segment}$ = 4096 * **Limit**
    ⟶ $limite_{segment}$ max = 4096 * 2^{20} = 2^{12} * 2^{20} = 2^{32} = 4GB
- Lorsque bit G à 1 ⟶ **Limit** multipliée par 4096 (2^{12}) par le CPU.
- Champ **Limit** codé sur 20-bits, en décalant cette valeur de 12-bits (mult. par 4096) on obtient une valeur 32-bits ⟶ 4GB adressable

Registres


General-Purpose Registers
31
16 15
8 7
0
AH
AL
BH
BL
CH
CL
DH
DL
BP
SI
DI
SP
16-bit
32-bit
AX
EAX
BX
EBX
CX
ECX
DX
EDX
EBP
ESI
EDI
ESP
Segment Registers
15
0
CS
DS
SS
ES
FS
GS
Program Status and Control Register
31
0
EFLAGS
Instruction Pointer
31
0
EIP

Règle de sélection des segments

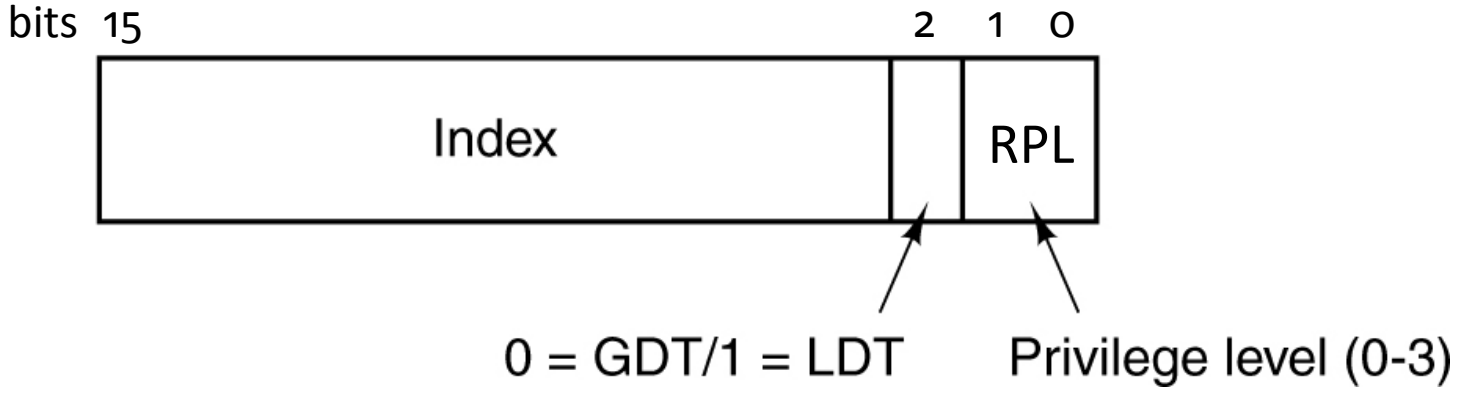
Type d'accès	Registre utilisé	Segment utilisé	Règle de sélection
Instruction	CS	Segment de code	• *Fetch* d'instructions
Pile	SS	Segment de pile	• Empilements (push) et dépilements (pop) • Accès mémoire avec ESP ou EBP
Données	DS	Segment de données	• Accès à des données (variables), sauf sur la pile
Strings (destination)	ES	Segment de données pointé par ES	• Destination d'une opération sur les strings (instructions spéciales)

Sélecteur de segment

- En mode protégé, les registres de segment du CPU doivent “pointer” sur des descripteurs de segment dans la GDT
- Au minimum, les registres CS (code), DS (données) et SS (pile) doivent pointer sur des descripteurs de segments
- Un descripteur de segment fait 64-bits de long alors qu'un registre de segment fait seulement 16-bits… comment faire ?
- Solution → **sélecteurs de segment**

Sélecteur de segment (2)

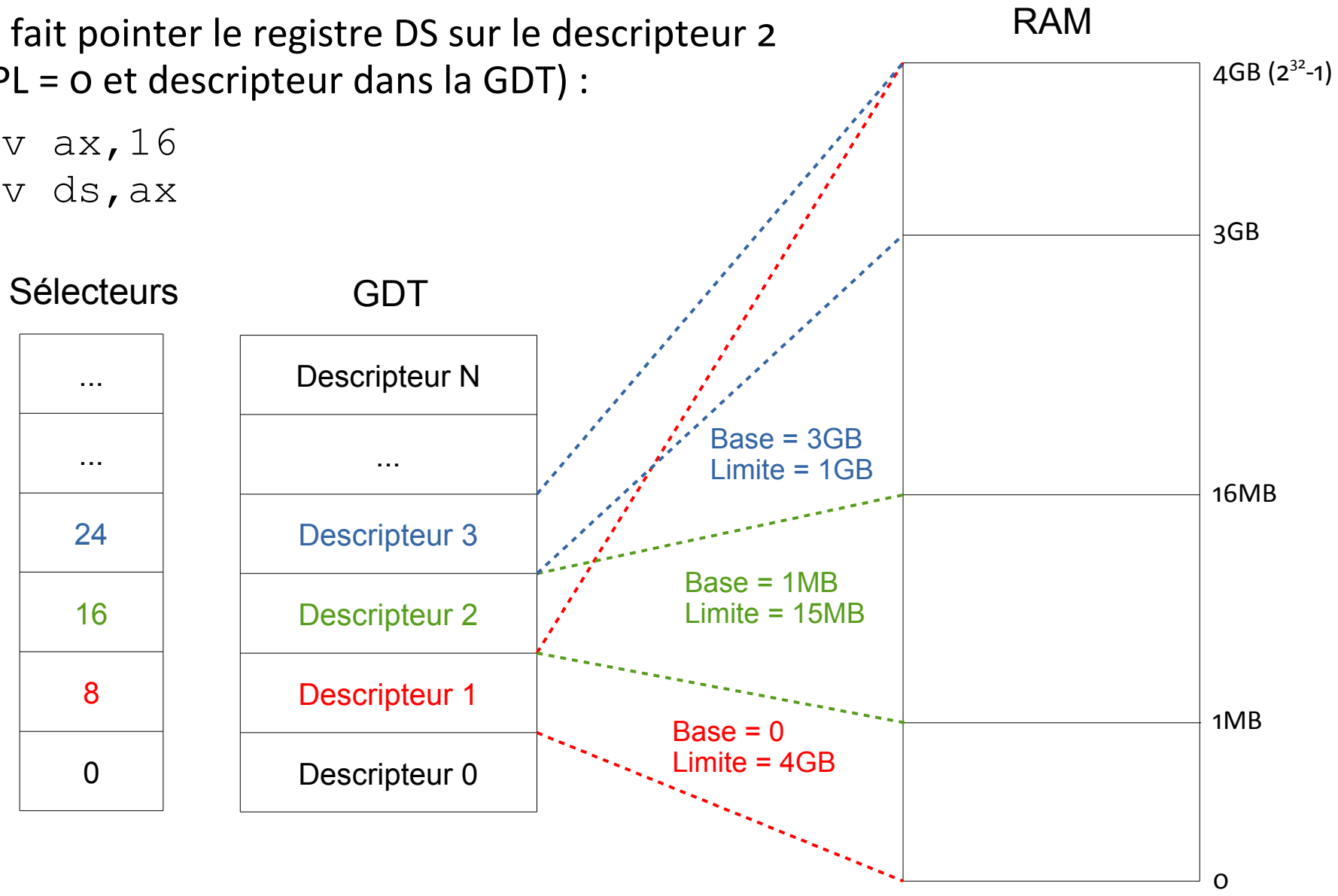
- Un **sélecteur** de segment (16-bits) identifie un descripteur de segment dans la GDT (ou la LDT)
- Il contient :
  - L'index du descripteur de segment dans la GDT
  - Un bit indiquant si l'entrée est dans la GDT (global au système) ou la LDT (local à une tâche)
  - Le niveau de privilège : 0 = system (highest), 3 = user (lowest)



Sélecteur de segment et registres

Ex: fait pointer le registre DS sur le descripteur 2 (DPL = 0 et descripteur dans la GDT) :

```
mov ax,16
mov ds,ax
```



Protection mémoire

- Un descripteur de segment défini un segment contigü de mémoire.
- Au moment d'un accès mémoire, par exemple lors d'un accès à variable `X`, le CPU va référencer le segment mémoire sur lequel pointe DS (car on adresse ici une donnée).
- Le CPU va ensuite vérifier que la mémoire adressée ne dépasse pas la limite définie dans le descripteur de segment :

  (`[X]` + `Base`) ≤ `Limit` ?
- Si ce test est faux, le CPU va générer une exception “General Protection Fault” et l'exécution sera alors stoppée.

Adressage et registres de segments

- Les opérations adressant le **code** (décodage des instructions en mémoire, sauts, etc.) référencent le descripteur de segment sur lequel pointe le registre **CS**

- Les opérations adressant les **données** (adressage de variables ou d'adresses mémoires) référencent le descripteur de segment sur lequel pointe le registre **DS**

- Les opérations adressant la **pile** (push et pop) référencent le descripteur de segment sur lequel pointe le registre **SS**

Registres de segment

Adressage mémoire référençant :

- Des données :
  - MOV [registres], MOV [adresse] → segment DS
- La pile :
  - MOV [ESP], MOV [EBP], push, pop, call, ret → segment SS
- Le code :
  - JMP adresse, JMP label, call, ret → segment CS

Possibilité de forcer l'utilisation d'un registre de segment :

- MOV EAX,[EBX] → [EBX] référence DS par défaut
- MOV EAX,[ES:EBX] → force référencement d'ES au lieu de DS

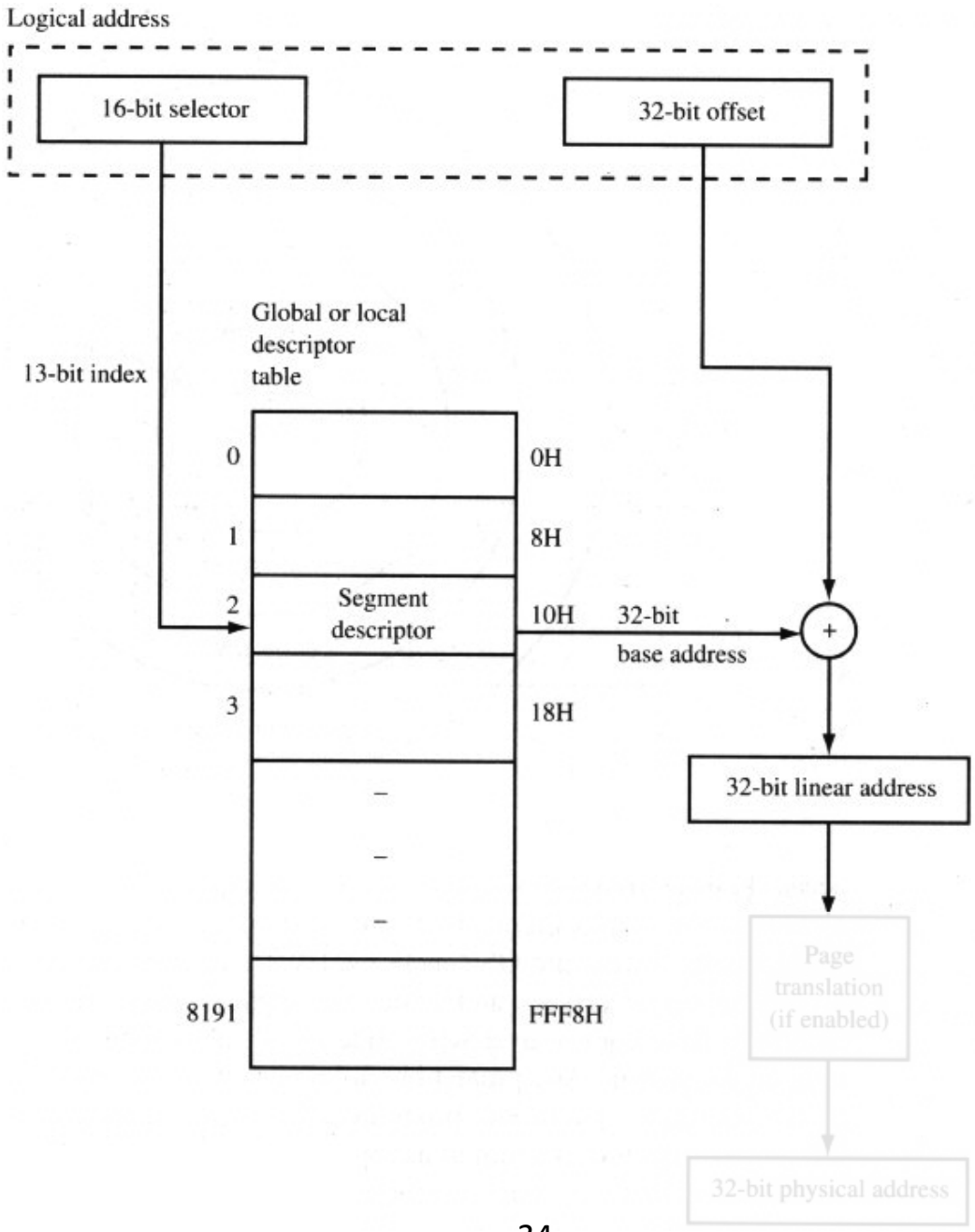
Registres de segment : exemples

- MOV EAX,[EBX]
- MOV [n],77
- MOV [ESP],EBX
- MOV [EBP],EBX
- JMP 1000
- PUSH 50
- MOV [ESI],1234
- PUSH DWORD [ESI]
- JNE my_label
- INC byte [x]
- MOV EAX,4

Registres de segment : exemples

- MOV EAX,[EBX] → DS
- MOV [n],77 → DS
- MOV [ESP],EBX → SS
- MOV [EBP],EBX → SS
- JMP 1000 → CS
- PUSH 50 → SS
- MOV [ESI],1234 → DS
- PUSH DWORD [ESI] → DS et SS
- JNE my_label → CS
- INC byte [x] → DS
- MOV EAX,4 → aucun registre de segment !

Adressage segmenté avec la GDT



Segmentation (1)

- Disons qu'on veut définir les deux segments de mémoire A et B :

- Soit `descA` et `descB`, deux descripteurs de segments définissant les segments de mémoire A et B :
  - `descA.base = 0` `descA.limit = 10000`
  - `descB.base = 50000` `descB.limit = 200001`
- Chaque descripteur est stocké à un index dédié dans la GDT
- Par exemple, soit A est à l'index 3 et B à l'index 5 dans la GDT

Segmentation (2)

- Si A se trouve à l'index 3 et B à l'index 5 dans la GDT, alors :
  - Le sélecteur du segment A vaut 3*8 = 24
  - Le sélecteur du segment B vaut 5*8 = 40
  - On multiplie par 8, car l'index commence au bit 3, ce qui implique un décalage de 3-bits
  - Nous considérons aussi un niveau de privilège (DPL) de 0 (système) et des descripteur dans la GDT (par opposition à la LDT)

- Si `CS` est initialisé au sélecteur pour le segment A (24), alors seulement la zone mémoire de 0 à 9999 sera adressable par le code

- Si `DS` est initialisé au sélecteur pour le segment B (40), alors seulement la zone mémoire de 50000 à 250000 sera adressable par les données

Adressage segmenté (1)

- Que se passe-t-il lorsque la mémoire est adressée ?

```
mov    ebx,1234
mov    [ebx],byte 33
```

Adressage segmenté (2)

- Que se passe-t-il lorsque la mémoire est adressée ?

```
mov    ebx,1234
mov    [ebx],byte 33
```

1) Le CPU extrait la valeur du sélecteur de segment depuis le registre `DS`
2) Le CPU fait un lookup de la GDT avec la valeur de `DS` pour obtenir la base et limite du segment
3) Le CPU ajoute l'adresse spécifiée (1234) à la base et vérifie que l'adresse résultante ne dépasse pas la limite du segment
   - Si c'est le cas, le CPU génère l'exception “general protection fault”

Chargement de la GDT

La GDT possède quelques particularités :

- Le premier segment (sélecteur 0) doit **toujours** être nul (que des zéros)
- Le chargement de la GDT se fait à l'aide de l'instruction spécifique **`lgdt`**
- L'instruction **`lgdt`** prend l'adresse d'une structure de donnée de 48–bits en RAM définissant :
  - La limite (16-bits) de la GDT, c'est-à-dire la taille de la GDT – 1
  - L'adresse (32-bits) de la GDT

Adressage “FLAT” grâce à la GDT

Il peut être souhaitable d'adresser toute la mémoire physique de manière “flat”, c'est-à-dire avoir accès à toute la mémoire de manière linéaire. Cela peut-être fait de la manière suivante :

- Une entrée NULLE pour le descripteur à l’index 0.
- Une entrée pour le segment de code : base de 0 et limite de 2^{20}-1
- Une entrée pour le segment de données : base de 0 et limite de 2^{20}-1
- Les segments de code et de données adressent des zones de mémoires communes ; dans ce cas on dit qu'ils “**overlappent**” (se chevauchent)

Références

Sur la page Cyberlearn du cours :

- "Taming the x86 beast" by Jim Turley
- Intel® 64 and IA-32 Architectures Software Developer's Manual :
  - Volume 3A: System Programming Guide, Part 1