Chapitre 5

Gestion de la Mémoire

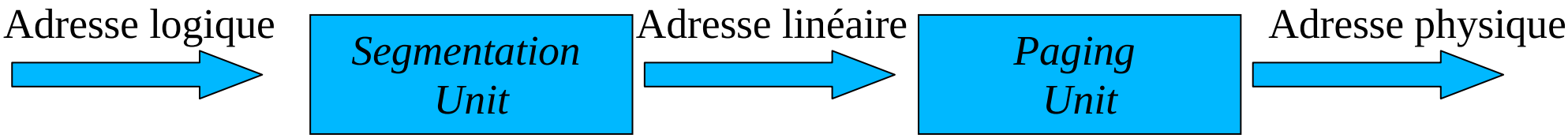
1ère partie: Point de vue processeur

Présentation

- La mémoire peut être adressée de 3 points de vue
- Point de vue processeur
  - Comment le processeur adresse la mémoire physique
- Point de vue kernel
  - Comment le kernel gère la mémoire pour son propre usage
- Point de vue des processus
  - Comment la mémoire est allouée aux processus
  - Allocation différée
  - Gestion des erreurs

Adressage x86

- 3 formes d'adressage dans les CPUs x86
  - Adressage logique
    - Spécifie l'adresse d'un opérande et/ou d'une instruction
    - Chaque adresse logique est constituée d'un segment et d'un offset
  - Adressage linéaire
    - Un entier 32bits non signé permettant d'adresser n'importe quelle zone mémoire
  - Adressage physique
    - Adresse la cellule mémoire
    - Correspond aux signaux électriques



Mode réel, mode protégé

- Mode réel
  - Introduit dans le 80286
  - CPUs précédent n'avaient qu'un unique mode, équivalent à réel
  - Adressage de la mémoire par segments de 20 bits =>1MB max
  - Pas de protection mémoire
  - Mode du processeur au boot pour compatibilité
- Mode protégé
  - Introduit dans le 80286
  - Permet la gestion hardware de la protection mémoire
  - Ajout d'un mécanisme de pagination dans le 80386
  - Introduit les niveaux de privilège ou *rings*
    - 0 a tous les pouvoirs
    - 3 très limité

Segmentation hardware

- Une adresse logique est constituée
  - D'un identificateur de segment
  - D'un offset représentant une adresse relative dans ce segment
- Identificateur (*Segment Selector*): 16 bits
- Offset : 32 bits
- Le processeur contient des registres pour obtenir l'identificateur de segment
  - 6 registres disponibles
  - CS, SS, DS, ES, FS et GS
  - Si plus de segments, sauvegardes des valeurs en mémoire

Registres de segment

- 3 registres ont un rôle spécifique
- CS : Code Segment
  - Pointe vers un segment contenant des instructions
  - Contient un champs de 2 bits qui indique le *Current Privilege Level (CPL)*
- SS : Stack Segment
  - Pointe vers un segment contenant la pile courante du programme
- DS : Data Segment
  - Pointe vers un segment contenant les données statiques du programme
- Les 3 autres registres peuvent faire référence à des segments arbitraires

Segments descriptors

- Chaque segments est décrit par 8 octets
- Champs du *Segment Descriptor*
  - Base : 32 bits contenant l'adresse linéaire de début de segment
  - G : granularité, si 0 le segment est exprimé en octets, sinon en multiples de 4Ko
  - Limit : 20 bits indiquant la longueur du segment en octets
  - S : Si 0, segment contenant des données critiques pour le kernel
  - Type : 4 bits caractérisant le type du segment et ses droits d'accès
  - DPL : Descriptor Privilege Level, 2 bits indiquant le niveau de privilège minimal pour accéder à ce segment
  - Segment-Present : Indique si le segment est en mémoire principale

Descriptor Tables

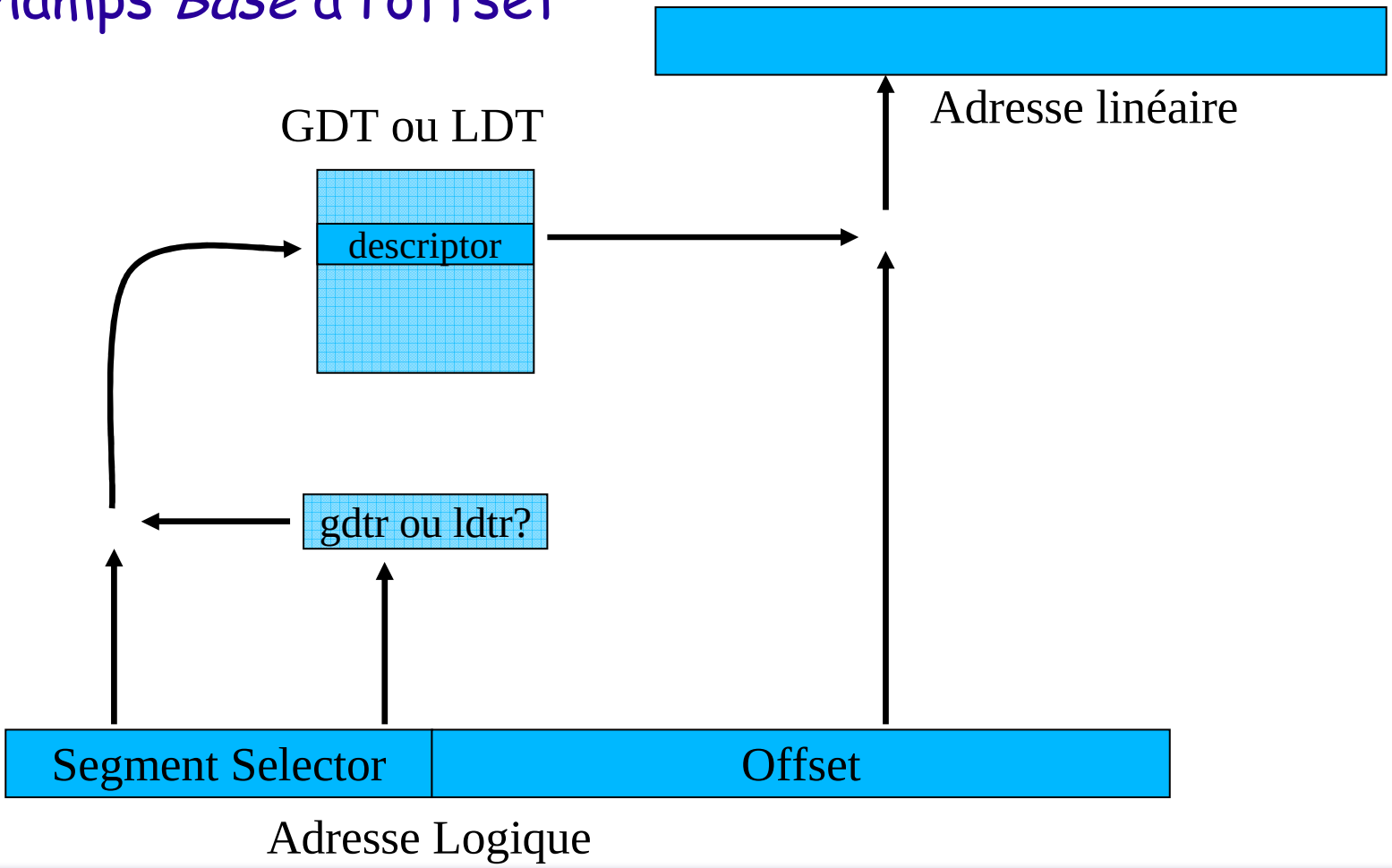
- Les *segments descriptors* sont stockés dans des tables
- Global Descriptor Table
  - Unique
  - Adresse en mémoire contenue dans le registre *gdtr*
- Local Descriptor Table
  - Chaque processus en a une
  - Adresse en mémoire contenue dans le registre *ldtr*
- Pour convertir une adresse logique en adresse linéaire, il faut passer par ces tables
  - Lecture du *Segment Selector* dans l'adresse logique
  - Lire le *Segment Descriptor* dans la table (GTD ou LDT) pour connaître l'adresse de début de segment
  - Vérifier les droits d'accès
  - Calculer l'adresse linéaire

Registres non programmables

- La conversion logique -> linéaire nécessite des accès mémoire
  - Trop coûteux en pratique
- Utilisation de registres CPUs spéciaux
  - Un pour chacun des registres de segmentation
  - Non programmables
  - Contiennent le *Segment Descriptor* correspondant
  - Sont mis à jour quand un *Segment Selector* est chargé
- Mais au fait, que contient un Segment Selector?
  - 13 bits servant à identifier le segment descriptor correspondant dans la GDT ou LDT
  - Un indicateur *TI* (*Table Indicator*) qui indique si le segment descriptor et dans la GDT ou LDT
  - Un *Requestor Privilege Level* (2 bits) qui indique le niveau de privilège du CPU quand le segment descriptor est chargé

Schéma de translation

- Examen du champs *TI* pour savoir dans quelle table regarder
  - Utilisation du registre gdtr ou ldtr
- Trouver le *segment descriptor* correspondant dans la table
- Ajout du champs *Base* à l'offset



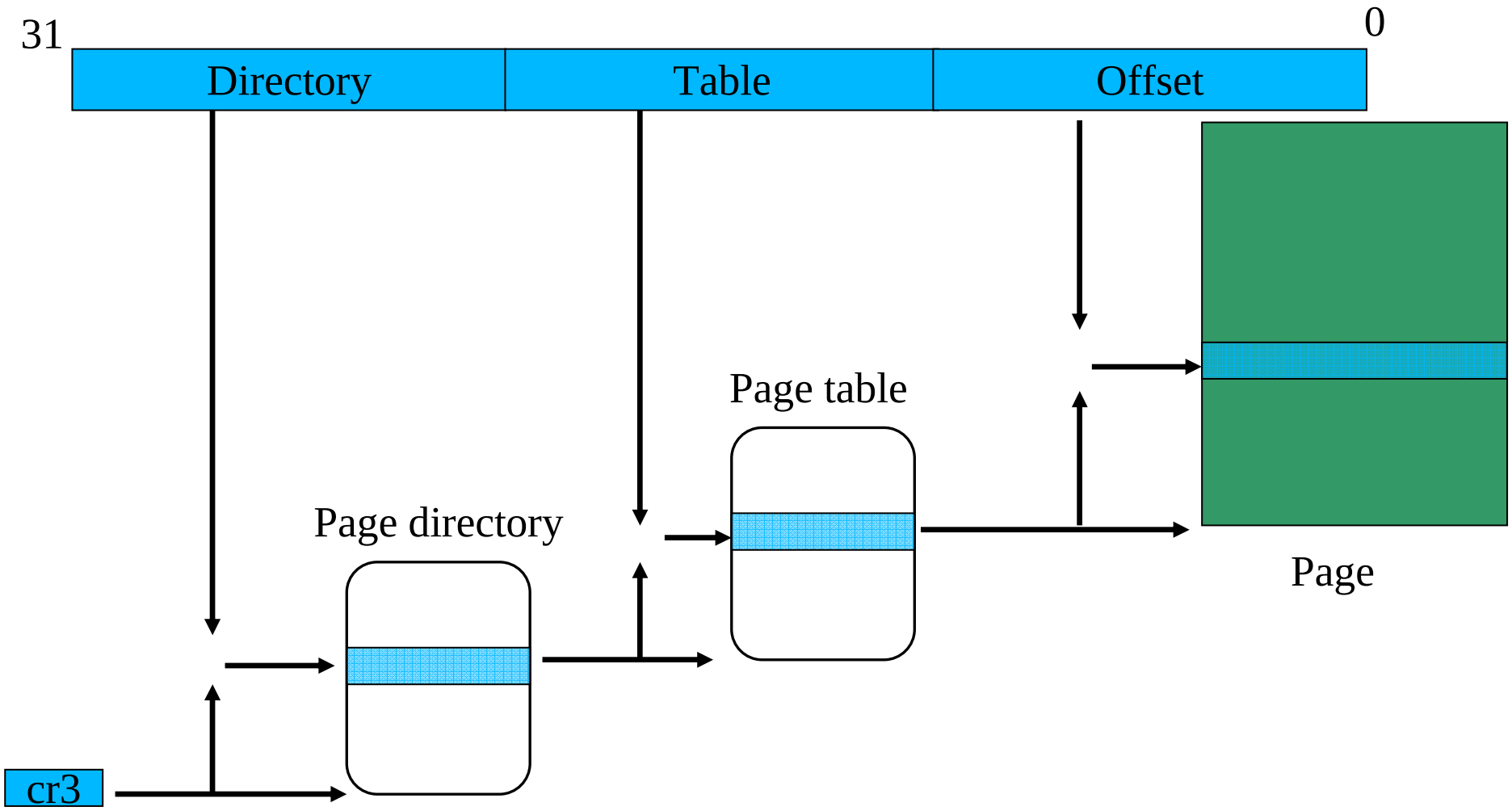
Pagination

- La pagination transforme les adresses linéaires en adresses physiques
- Pour des raisons d'efficacité, les adresses linéaires sont groupées en intervalles de taille fixe: des pages
- Des adresses contigus dans une page sont des adresses physiques contigus
- Différence entre Page manipulée par l'OS et Page physique (*page frame* ou page physique)
  - Une Page peut au cours de l'exécution changer de page physique
- Les structures de données assurant la conversion des adresses sont des tables de page
  - Stockées en mémoire
  - Doivent être initialisées par le kernel
- Les processeurs Intel peuvent fonctionner sans pagination
  - Registre pour activation
  - Si désactivé, adresses linéaire sont interprétées comme physiques

Pagination des 80386

- L'unité de Pagination des 80386 traite des pages physiques de 4KB
- Une adresse linéaire 32 bits est divisée en
  - *Directory* : 10 bits de poids forts
  - *Table* : 10 bits intermédiaires
  - *Offset* : 12 bits de poids faible
- La conversion se fait en 2 étapes, en utilisant le *Page Directory* et le *Page Table*
- Le *Page Directory* contient l'adresse de *Page Table*
- La *Page Table* contient l'adresse des pages physiques
- L'adresse physique du *Page Directory* est stockée dans le registre *cr3*
- Taille maximale adressable:
  - 10 bits pour le Page Directory donc 2^{10} entrées possibles
  - Donc 1024x1024x4096=2^{32} cellules mémoire

Exemple de pagination



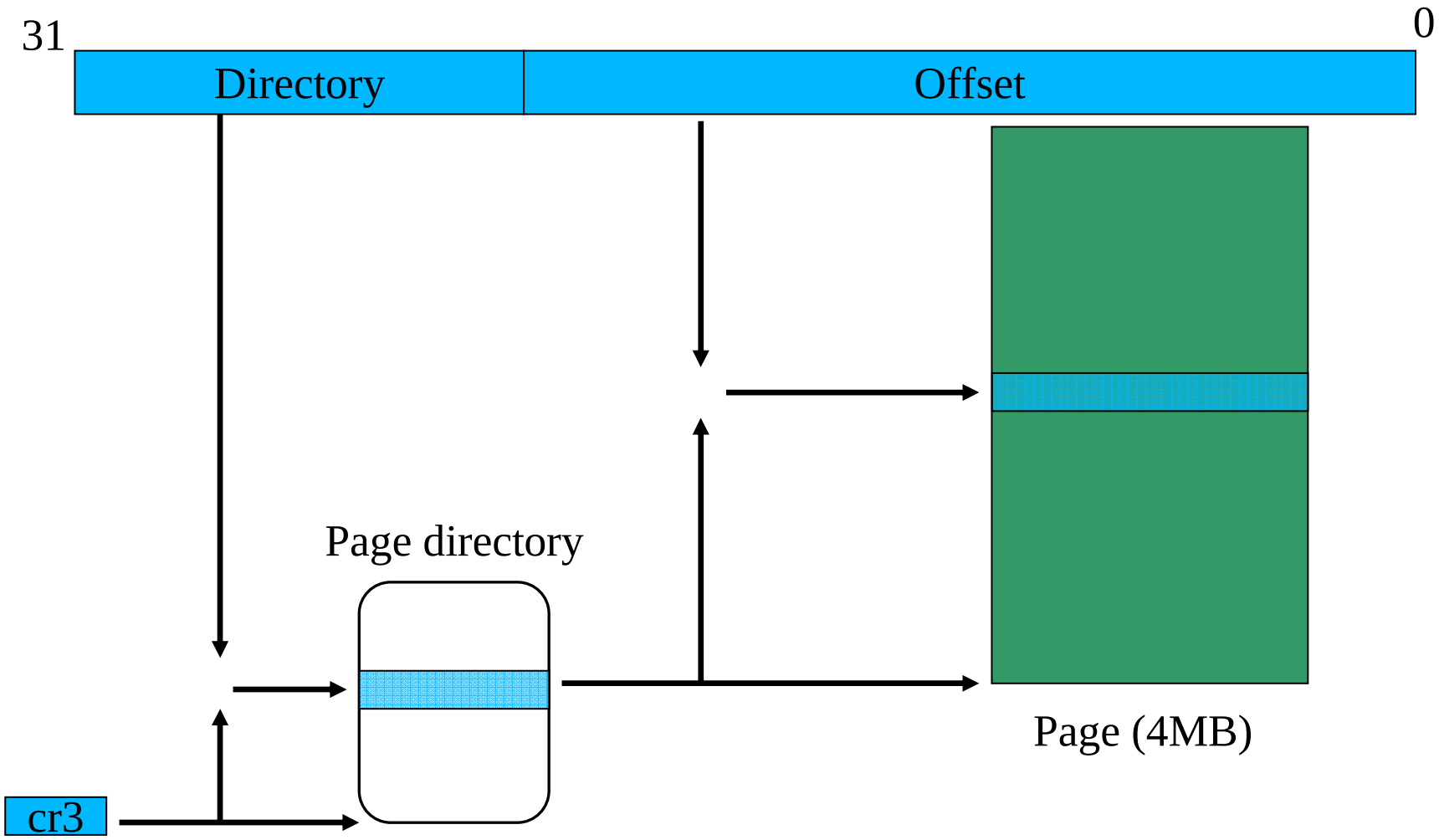
Entrées des tables

- Les entrées des répertoires et tables de pages ont les même entrées
- *Present* : la page ou la table de page sont présents en mémoire
- 20 bits les plus significatifs de l'adresse physique de la page
  - Les pages sont des multiples de 4096
  - Les 12 bits les moins significatifs sont toujours 0
- *Accessed* : Mis quand l'unité d'adressage accède à la page. Remis à 0 par l'OS
- *Dirty* : Applicable seulement aux entrées de la table de pages. Mis à 1 lors de l'écriture dans la page. Remis à 0 par l'OS
- *Read/Write* : Droits de lecture ou lecture/écriture
- *User/Supervisor* : Privilège requis pour accéder à la page ou à la table de pages
- *PCD et PWT* : contrôle le cache hardware
- *Page Size* : applicable seulement aux entrées du répertoire. Si mis, l'entrée correspond à une page physique de 4MB

Pagination étendue

- Les pentiums ont amené la pagination étendue
- Les pages physiques ont une taille de 4KB ou 4MB
- Dans ce cas, l'adresse linéaire est divisée en 2 champs
  - *Directory* : 10 bits les plus significatifs
  - *Offset* : les 22 bits restants
- Entrées dans le répertoire
  - Flag *Page Size* mis à 1
  - Les 10 bits de poids fort seulement sont nécessaires (multiples de 4MB)
- Les deux paginations peuvent cohabiter
- Utilisés pour gérer des gros blocs de mémoire contigus

Exemple de pagination étendue



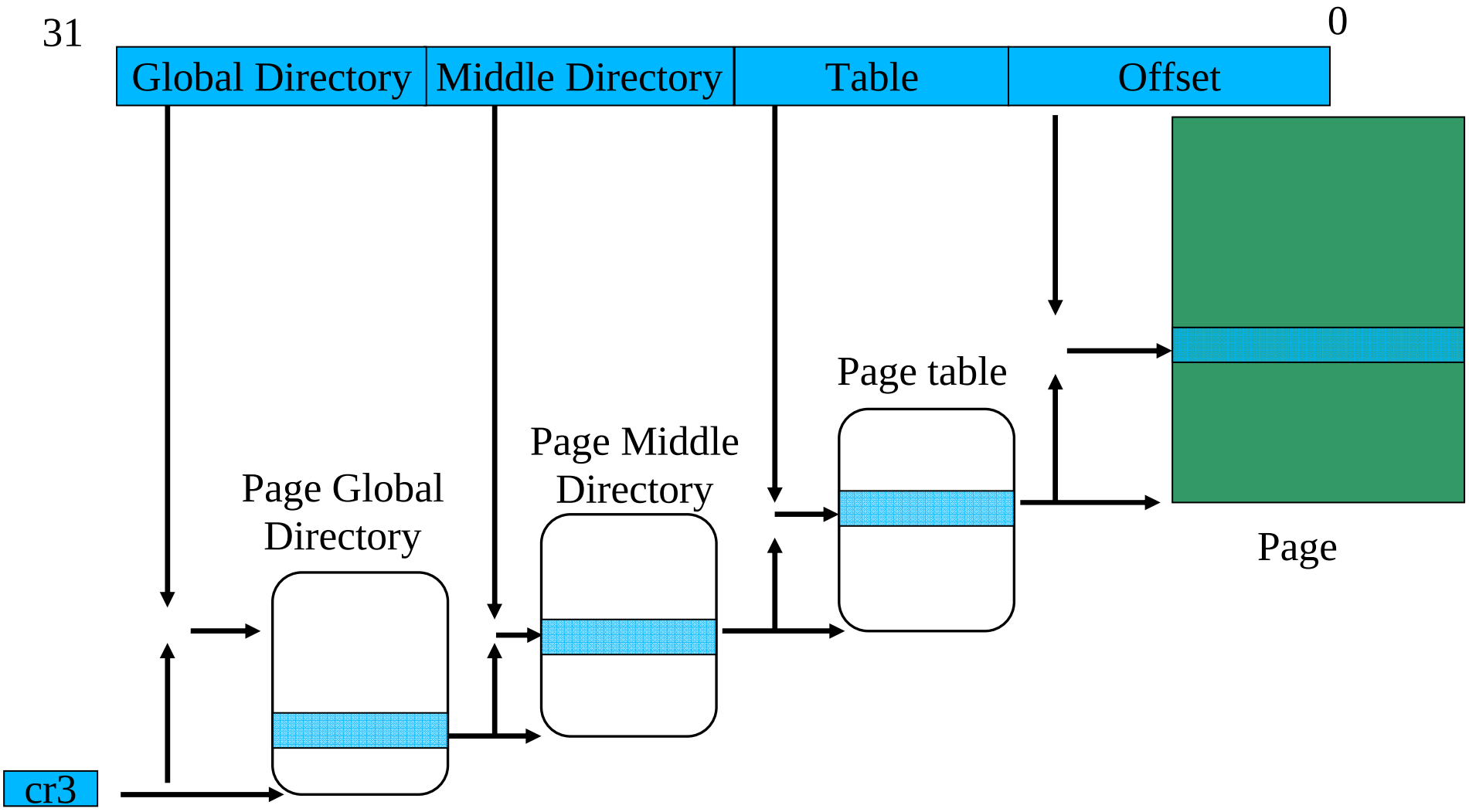
Cache et TLB

- Tous les processeurs modernes ont un ou plusieurs caches
- L'unité de gestion de cache est située entre l'unité de pagination et la mémoire physique
- Sur les x86, le cache peut être globalement désactivé ou controlé
  - Flag *CD* du registre *cr0* pour activation/désactivation
  - Flag *NW* pour Write-through ou Write-back
- Possibilité d'associer une politique de cache pour chaque page
  - *PCD* cache activé/désactivé
  - *PWT* Write-through ou Write-back
- Linux mets tous flags à 0 pour toutes les pages
  - Cache activé avec write-back pour toutes les pages
- Un cache spécifique pour la pagination est disponible
  - *Translation Lookaside Buffer*
  - Garde en cache les adresses linéaires récemment transformées

Pagination sous Linux

- Linux adopte un mécanisme de pagination à 3 niveaux
  - Inspiré des processeurs Alpha
  - Indispensable pour les machines 64 bits
- 3 types de tables
  - *Page Global Directory*
  - *Page Middle Directory*
  - *Page Table*
- L'adresse linéaire est découpée en 4 parties
- Chaque processus a sa propre *Page Global Directory* et ses propres *Page Tables*
- Quand un changement de processus intervient, Linux sauvegarde le contenu de *cr3* et charge la nouvelle valeur
  - L'unité de pagination utilise les bonnes tables
- Si Linux tourne sur un processeur qui n'a pas 3 niveaux de paginations (Pentium…), la Page Middle Directory est mise à 1 entrée

Exemple de pagination à 3 niveaux



Conclusion

- Segmentation et pagination sont 2 mécanismes permettant d'atteindre le même objectif
  - Séparer l'espace d'adressage physique des processus
- La segmentation assigne un espace linéaire différent à chaque processus
- La pagination permet d'associer un espace linéaire à différents espaces physiques
- Linux préfère la pagination
  - La gestion des processus est plus facile sur un unique espace linéaire
  - La segmentation n'est pas disponible sur tous les processeurs