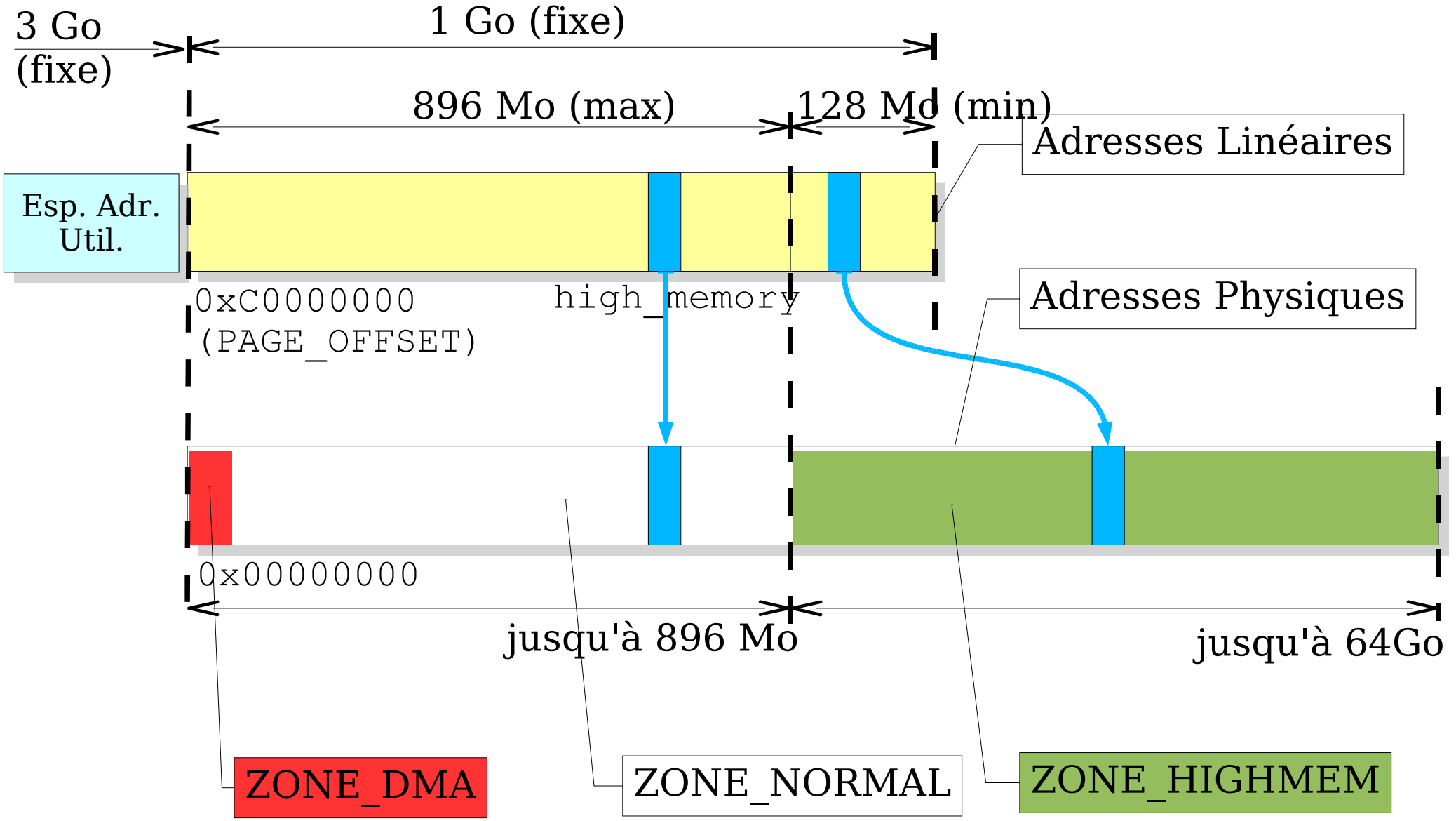
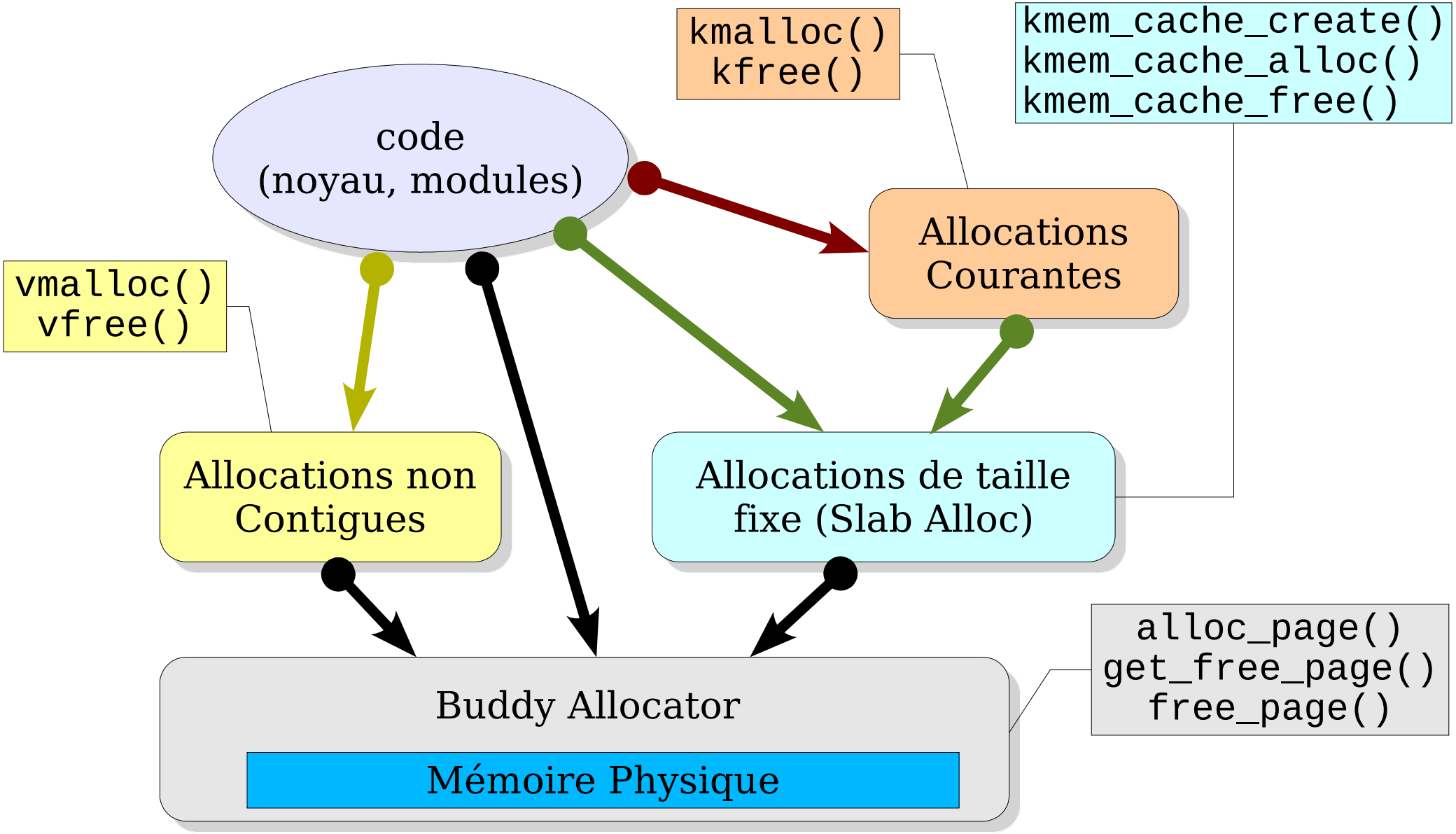
Gestion de la Mémoire Dynamique

Partie 2 : Les Allocateurs de Linux

Rappel chap 4 : Organisation des Espaces d'Adressage



Allocateurs de Linux



Buddy Allocator : Alloc. de Cadre(s Contigus)

- Syntaxe
  - Allocation :
    - `struct page *alloc_page(gfp_flags)`
    - `struct page *alloc_pages(gfp_flags,order)`
    - `u. long get_zeroed_page(gfp_flags)`
    - `u. long __get_free_page(gfp_flags)`
    - `u. long __get_free_pages(gfp_flags, order)`
    - `u. long __get_dma_pages(gfp_flags, order)`
  - Libération:
    - `void free_page(ptr)`
    - `void free_pages(ptr, order)`
    - `void __free_page(page)`
    - `void __free_pages(page,order)`
- Allocation de pages entières
  - quantité = 2^{order}

Paramètres d'Allocation

- Drapeaux élémentaires fixant le mode d'allocation
  - `__GFP_WAIT` : peut bloquer
  - `__GFP_HIGH` : priorité haute (peut utiliser la réserve en zone normale)
  - `__GFP_IO` : autorise les E/S pour libérer des pages en zone normale
  - `__GFP_HIGHIO` : autorise les E/S pour libérer des pages de mémoire haute
  - `__GFP_FS` : peut déclencher des opérations VFS de bas niveau
  - `__GFP_DMA` : page requise en zone DMA
  - `__GFP_HIGHMEM` : les cadres peuvent être trouvés en mémoire haute

Modes d'Allocation Prédéfinis

- Combinaison des drapeaux élémentaires
  - **GFP_ATOMIC** <=> __GFP_HIGH
  - **GFP_NOIO** <=> __GFP_HIGH + __GFP_WAIT
  - **GFP_NOHIGHIO** <=> *GFP_NOIO* + __GFP_IO
  - **GFP_NOFS** <=> *GFP_NOHIGHIO* + __GFP_HIGHIO
  - **GFP_KERNEL** <=> *GFP_NOFS* + __GFP_FS
  - **GFP_NFS** <=> *GFP_KERNEL*
  - **GFP_KSWAPD** <=> __GFP_WAIT + __GFP_IO + __GFP_HIGHIO + __GFP_FS
  - **GFP_HIGHUSER** <=> *GFP_KSWAPD*
- Sélecteur de zone
  - Defaut : ZONE_NORMAL + ZONE_DMA
  - **__GFP_DMA** + X : Uniquement ZONE_DMA
  - **__GFP_HIGHMEM** (sans __GFP_DMA) : toutes les zones

Allocations de Pages en Zone Haute

- Rappel : adressage direct impossible dans le noyau
- Drapeau **__GFP_HIGHMEM** ?!
  - Incompatible avec famille **get_free_page** ( )
    - Ces fonctions retournent l'@ linéaire du début de la page
    - Pas assez d'@ linéaires pour la mémoire haute …
- Pas de problème avec **alloc_page** ( )
  - Retourne l'@ linéaire (en mémoire basse) du **descripteur** du cadre
  - Ce cadre doit ensuite être mappé dans l'espace linéaire du noyau

Mapping Noyau des Pages Allouées en Mem. Haute

- Dans la zone linéaire haute du noyau
  - Adresses > `high_memory`
- `void *`**`kmap`**`(page)`
  - Associee l'@ physique d'une `struct page` à une page dans l'espace linéaire du noyau
  - Nombre limité de mappings : peut bloquer…
  - Appel accumulatif : une page peut être « kmappée » plusieurs fois
    - Elle doit être unmappée autant fois qu'elle a été mappée
- `void` **`kunmap`**`(page)`
- Translation entre adresse de page et adresse « kmappée »
  - `struct page *`**`virt_to_page`**`(void *kaddr)`
  - `void *`**`page_address`**`(page)`

Le 'Slab Allocator'

- Allocation optimisée pour des objets de taille fixe
  - Implémente un cache logiciel
    - Utilise le Buddy Alloc pour accroître son espace
    - Optimise l'occupation des pages
      - Fragmentation minimale
    - Ne restitue pas spontanément les pages
      - Le BA récupère des pages inutilisées en cas de pénurie
  - Optimisation pour les caches matériels
    - Alignements
    - Répartition homogène (algorithme de coloration)

Initialisation d'un Cache du 'Slab Allocator'

- Syntaxe

```
kmem_cache_t *
kmem_cache_create(name,size,offset,flags,
                  constr,destr)
```

  - **char * name** : max 20 carac, identitifiant, utile pour debug (généralement le type)
  - **size_t size** : taille des objets alloués
  - **size_t offset** : décalage du 1er objet par rapport au début de page (généralement 0)
  - **u. long flags** :
    - SLAB_NO_REAP : interdit la réduction du cache par le BA
    - SLAB_HWCACHE_ALIGN : généralement une bonne idée
    - SLAB_CACHE_DMA : allocation dans la zone DMA
  - **constr, destr** : possibilité d'associer des constructeurs/destructeurs (généralement inutilisé)

Utilisation et Destruction d'un Cache

- Utilisation :
  - `void *kmem_cache_alloc(cache, gfp_flags)`
    - `gfp_flags` : identique Buddy Alloc
  - `void kmem_cache_free(cache,obj_addr)`

- Destruction du cache :
  - `int kmem_cache_destroy(cache)`
    - Succès seulement si le cache est totalement vide
      - Les objets doivent être libérés préalablement
    - Indication d'une fuite mémoire en cas d'échec

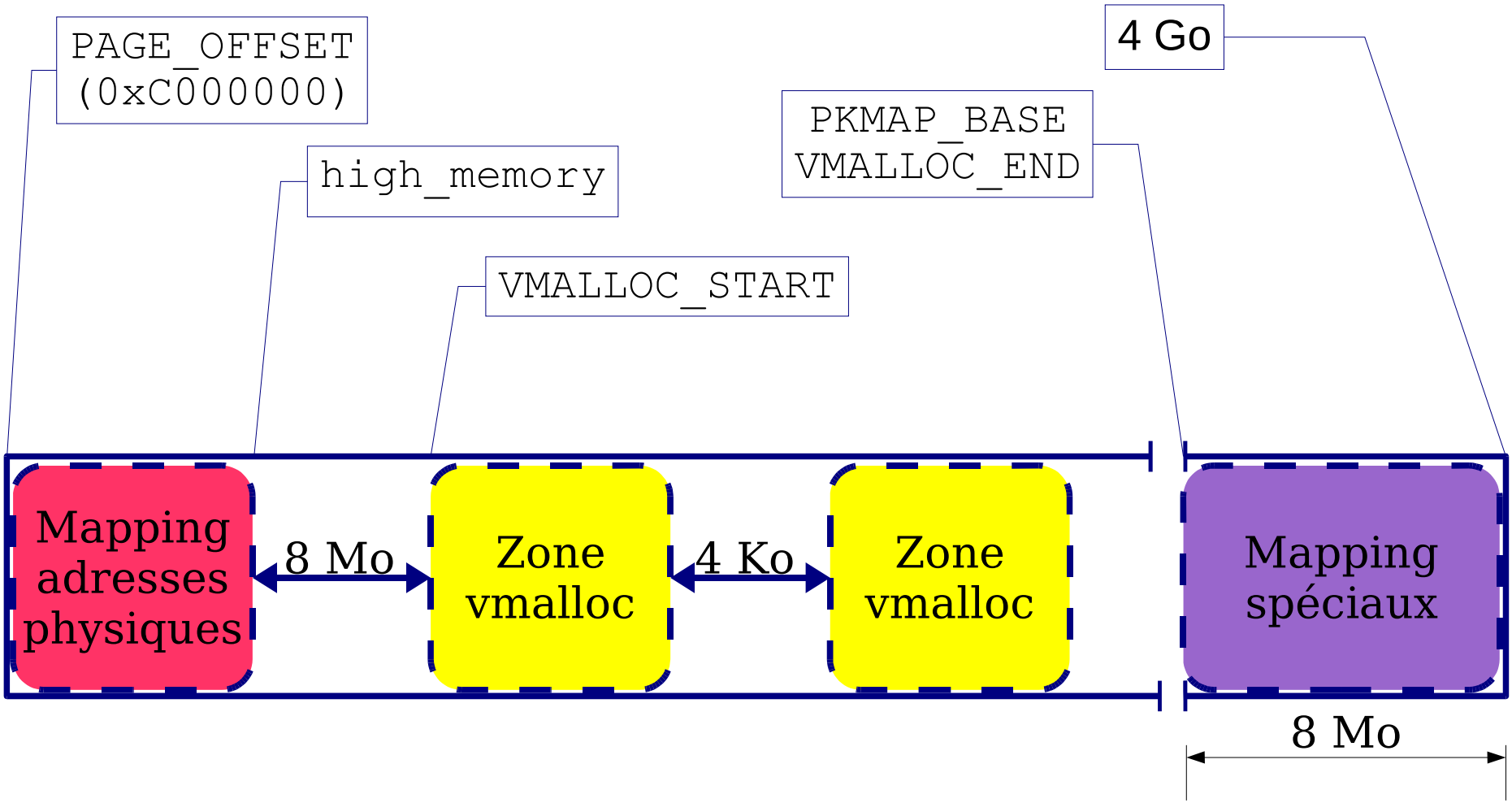
Allocation Courantes

- Syntaxe
  - `ptr = kmalloc(size,gfp_flags)`
    - Allocation d'une zone de mémoire contigue de taille `size`
      - Alignement effectif sur des tailles prédéfinies (dist géométrique)
    - `gfp_flags`
      - `GFP_ATOMIC` : non bloquant
      - `GFP_KERNEL` : peut bloquer
      - `GFP_USER` : peut bloquer, priorité faible
      - `__GFP_DMA, GFP_HIGHUSER, __GFP_HIGHMEM, ...`
  - `void kfree(ptr)`
- Avantage : simplicité
- Inconvénients : pas toujours le plus efficace
  - Fragmentation, alloc. récurrentes, grandes quantités, ...

Allocations Non Contigues

- Allocation de pages mémoire :
  - CONTIGUES dans l'espace d'Adressage Linéaire
  - NON CONTIGUES dans l'espace d'Adressage Physique
    - Appels successifs à **get_free_page()**
- API
  - `void *vmalloc(size)`
  - `void vfree(addr)`

Plan d'Occupation de l'Espace d'Adr. Linéaire



Propriétés Remarquables des Alloc. Non Contigües

- Bon moyen d'allouer de grandes quantités de mémoire
  - Ex: utilisé par insmod() pour allouer la mémoire d'un module
- Gestion plus lourde
  - Allocation : obligation de construire les tables de page correspondantes
  - Tables de page du processus courant inchangées
    - Défaut de page entrainant la recopie des entrées de la table de page du noyau dans celles du processus
- Allocation bloquante (`GFP_KERNEL + __GFP_HIGHMEM`)

Mapping Manuel

- Principe
  - Construit les tables de page de façon à obtenir des adresses linéaires
  - Mais sans allouer de mémoire physique
    - Utile par exemple pour mapper de la mémoire vidéo
  - `void *ioremap(offset,size)`
    - offset : adresse physique début
    - size : taille espace mémoire
    - Note : la mémoire linéaire doit être manipulée avec fonctions tq `bread()`
  - `void iounmap(addr)`