Gestion de la mémoire centrale

Plan du chapitre

- Introduction
  - Définitions
  - Différentes stratégies
- Allocation par zones
- Mémoire virtuelle
- Pagination

Le contexte

- Système multiprocessus
- Objectif : permettre aux processus de s'exécuter
- Utilisation efficace de la mémoire et de l ’UC
- Bonnes performances (temps de réponse) pour les utilisateurs

Contraintes

- On suppose que la somme des tailles des processus est supérieure à la taille de la mémoire allouable
- La taille d'un processus est en général beaucoup plus petite que celle de la mémoire disponible
- Avant exécution, le code d'un processus est conservé dans un fichier sur disque

Un schéma simpliste : la monoprogrammation

- Un processus en mémoire à un instant donné
- Chargement à la création du processus
- Mémoire libérée à la fin du processus
- Inconvénients
  - UC oisive pendant les chargements et les entrées sorties
  - Mémoire mal utilisée

Utilisation des temps morts dus aux entrées- sorties lentes

- Ne laisser en mémoire centrale qu'un processus prêt à s'exécuter (éligible)
- Que faire des processus bloqués ?
- Les stocker temporairement sur disque pour les recharger ultérieurement
- On dit qu'on réquisitionne la mémoire centrale

La monoprogrammation avec va et vient (swapping)

- Un processus en mémoire à un instant donné : le processus élu
- Chargement au passage d'éligible à élu
- Vidage à la sortie de l'état éligible
- Inconvénients
  - UC oisive pendant les chargements et vidages
  - Mémoire mal utilisée

Multiprogrammation

- La mémoire est partagée en zones
- Chaque zone contient un processus
- L'UC est allouée à un processus chargé et éligible

Multiprogrammation sans réquisition

- Un processus est chargé une seule fois
- Il reste en mémoire jusqu'à sa terminaison
- L'UC peut être active pendant le chargements d'un nouveau processus

Multiprogrammation avec réquisition

- Un processus dont l'exécution a commencé peut être vidé sur disque
  - Entrée sortie longue
  - Exécution d'un processus prioritaire
- Peut-on recharger un processus vidé dans n 'importe quelle zone ?
- En général, un programme est « attaché » à une zone
  - À cause des adresses absolues
- Comment assurer l'isolation des zones

Mémoire virtuelle

- Objectif : rendre un processus indépendant de la zone de mémoire physique où il s'exécute
- Conventions de programmation
- Mécanisme câblé de traduction dynamique d'adresses
  - Unité de gestion de la mémoire (MMU)

Doit on charger en mémoire la totalité d ’un processus?

- C’est en général inutile
- Comment déterminer les portions utiles ?
- Comment charger dynamiquement les portions qui deviennent indispensables ?
- Les mémoires virtuelles modernes permettent de charger ou de vider des sous-ensembles des programmes appelés des pages

Les problèmes de l'allocation de mémoire

- Correspondance entre adresses virtuelles et adresses physiques
  - En général, les contraintes imposées par le matériel l'emportent
- Gestion de la mémoire physique
- Protection
  - Éviter qu'un processus puisse modifier une zone qui ne lui est pas affectée
- Partage

Implantation statique

- Un processus est chargé une fois pour toutes
- Il réside en mémoire jusqu'à sa terminaison
- Choix
  - Mono ou multiprogrammation
  - Dans ce dernier cas, comment sont définies et gérées les différentes zones

Réimplantation dynamique

- Où charger ?
  - Existe-t-il des emplacements disponibles ou faut il libérer de la place
- Comment charger ?
  - En totalité ou par morceaux (par pages)
- Quand charger ?
  - Statiquement avant exécution
  - Dynamiquement au fur et à mesure des besoins

Gestion de la mémoire par zones

Plan

- Rappels TP1
- Utilisation pour allouer de la mémoire à un processus (gestion du tas)
- Utilisation pour charger des processus et les exécuter
- Zones de tailles quelconques
- Zones de taille fixe
- Zones de tailles prédéfinies

Représentation de l'espace libre

- Liste chaînée des zones libres
- Dans chaque zone, on trouve un descripteur
  - taille de la zone
  - l'adresse de la zone libre suivante
- La liste peut être ordonnée par taille croissante ou par adresses

Allocation d ’une nouvelle zone

- Recherche d’une zone libre de taille suffisante
- Partage de cette zone entre la zone allouée et un résidu qui reste dans la liste libre
- Choix de la zone libre
  - La première trouvée (« first-fit »)
  - La meilleure possible (« best-fit »)
  - La plus mauvaise (« worst-fit »)

Libération d'une zone

- Fusion des zones libres contiguës
- Recherche des zones libres voisines
- Plus efficace si la liste libre est classée par adresses
- Conséquence sur l'allocation
  - Ou bien first fit
  - Ou bien deux listes différentes

La fragmentation externe

- Sa cause : les allocations créent des zones libres trop petites pour être utilisables
- Il faut périodiquement réorganiser l'ensemble de la mémoire

Cas des zones de taille constante

- Un vecteur de bits suffit à représenter l'espace libre, ou une liste de zones dans un ordre quelconque
- Fragmentation interne
  - Différence entre taille allouée et taille réellement utile

Cas des zones de taille prédéfinie

- Une liste libre par taille
- Allocations et libérations simples

Problèmes

- le choix des tailles
- Que faire quand on n'a plus de blocs d'une taille donnée, mais des blocs de taille supérieure
- Solution
  - Technique de subdivision

La méthode du compagnon

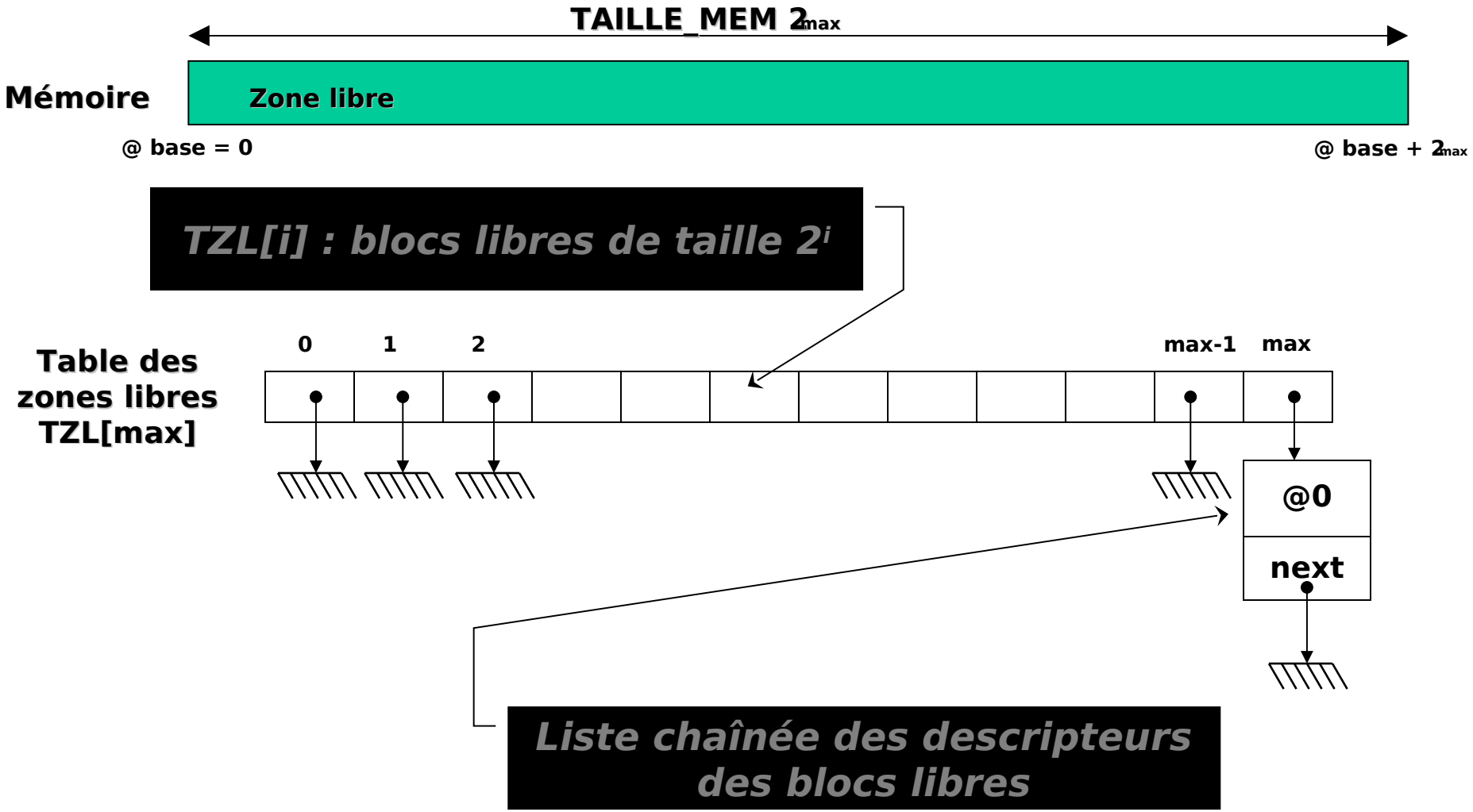
- « buddy system »
- Blocs de taille 2^i m
- Fractionnement à l'allocation
- Fusion à la libération

Algorithme du 'compagnon' (buddy system)

- Allocation par blocs de tailles prédéfinies
  - Blocs de 2^k, pour une taille mémoire de 2^{max}
- Principe d'allocation
  - Table des zones libres
  - Recherche d'un bloc de taille 2^k
  - Découpage des blocs libres en 2 blocs de taille inférieure (2 'compagnons') si possible
- Principe de libération
  - Recherche du 'compagnon' du bloc libéré
  - Fusion des 'compagnons' si possible pour rendre un seul bloc libre de taille supérieur

Allocation dans le 'buddy system'

État initial



Allocation dans le 'buddy system'

Boucle d'allocation : bloc de taille *p* avec $2^{k-1} < p < 2^k$

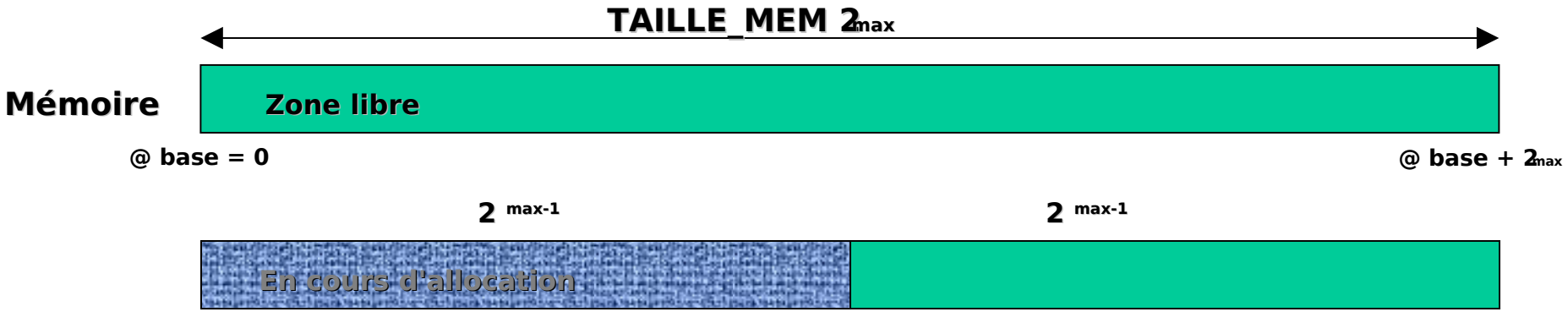
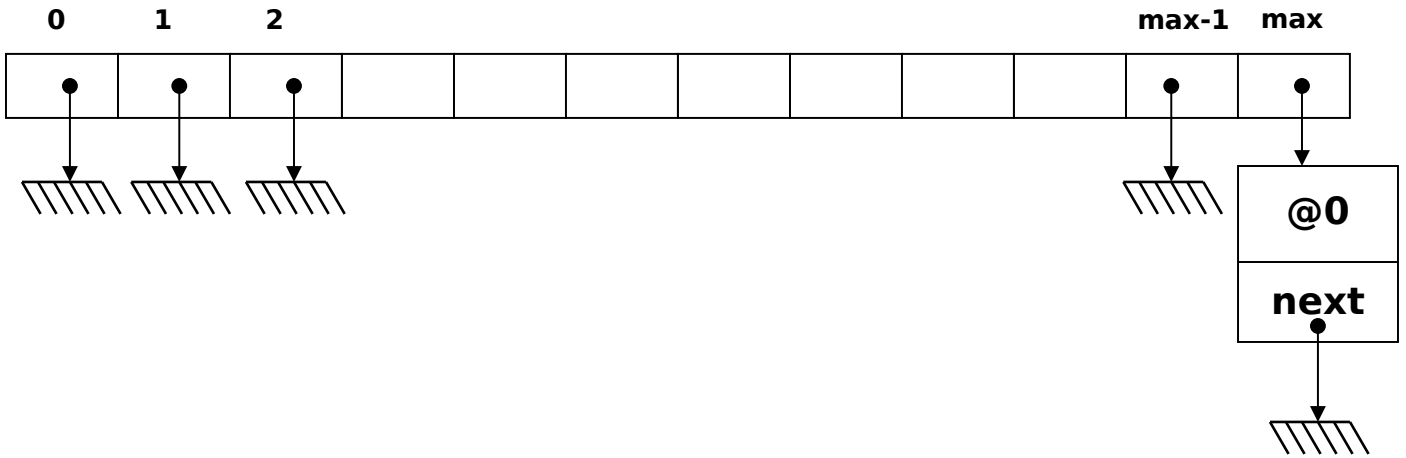
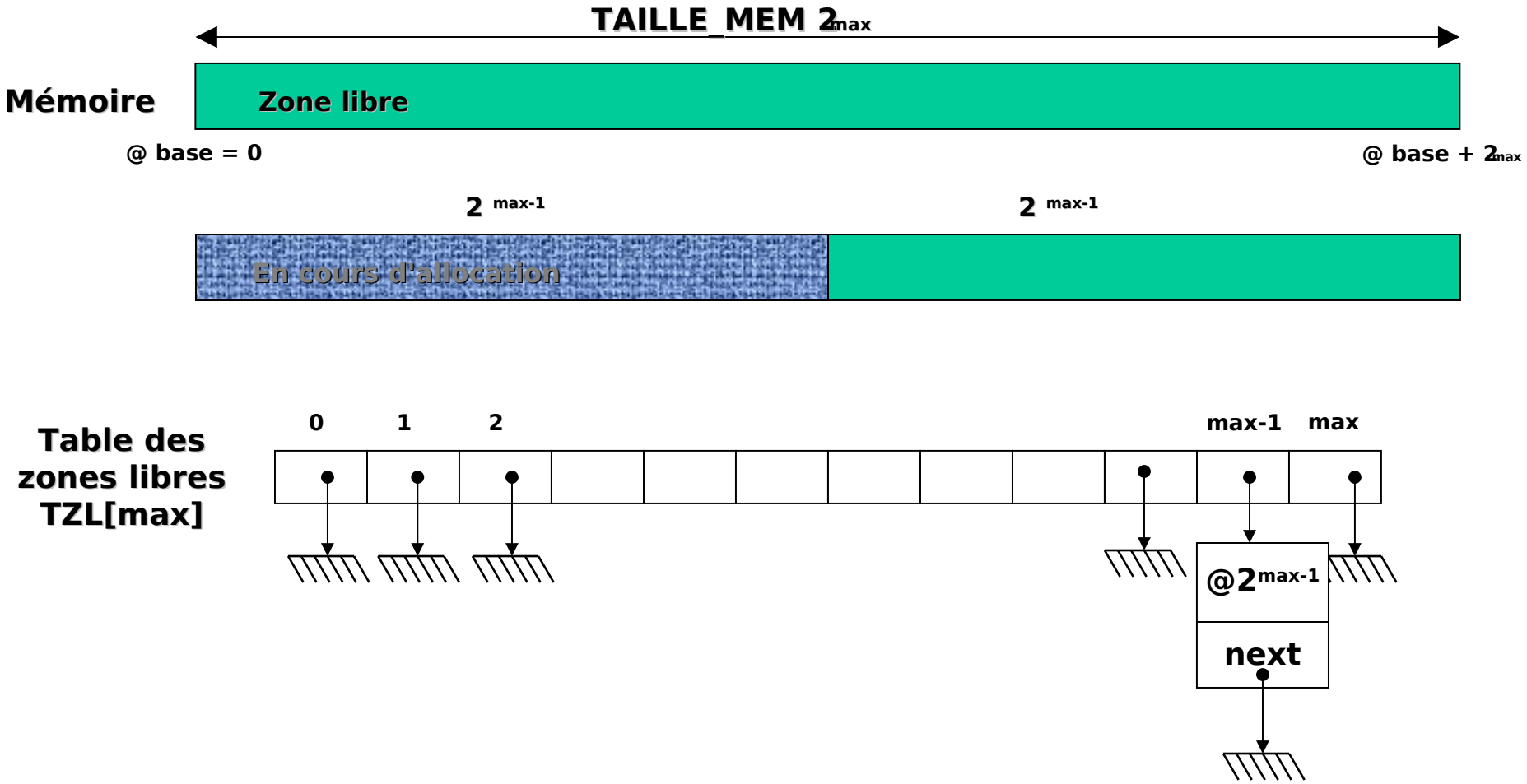


Table des zones libres TZL[max]



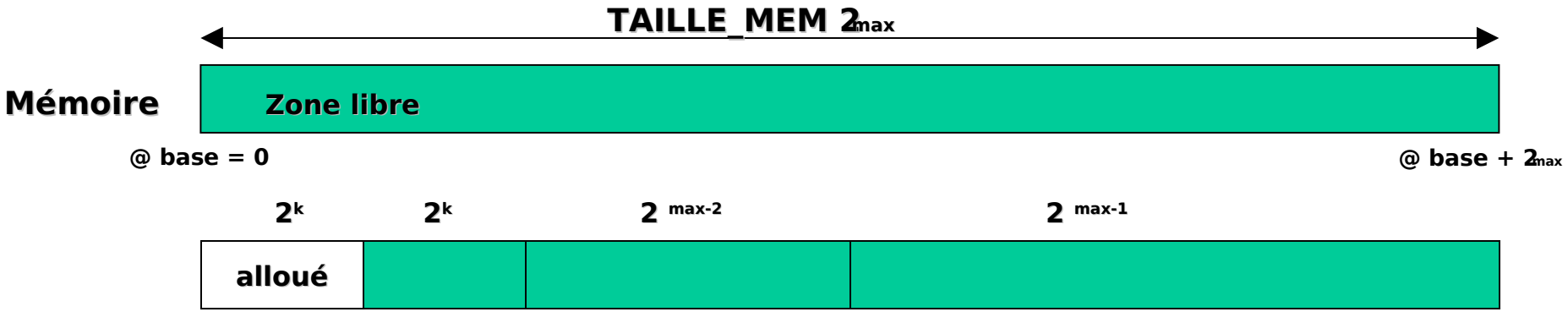
Allocation dans le 'buddy system'

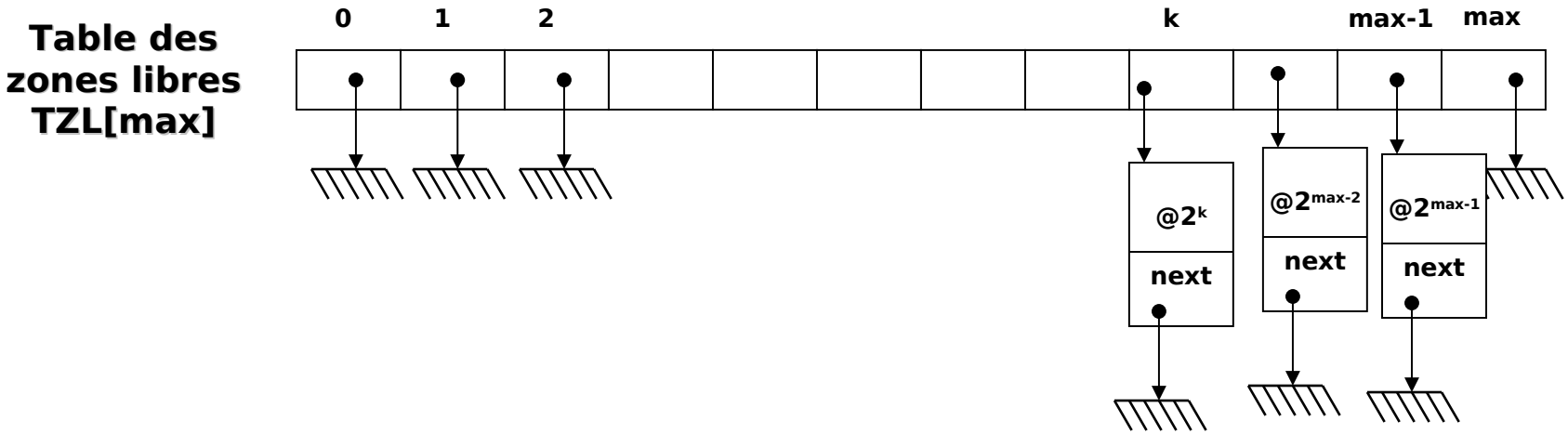
Boucle d'allocation : bloc de taille *p* avec $2^{k-1} < p < 2^k$



Allocation dans le 'buddy system'

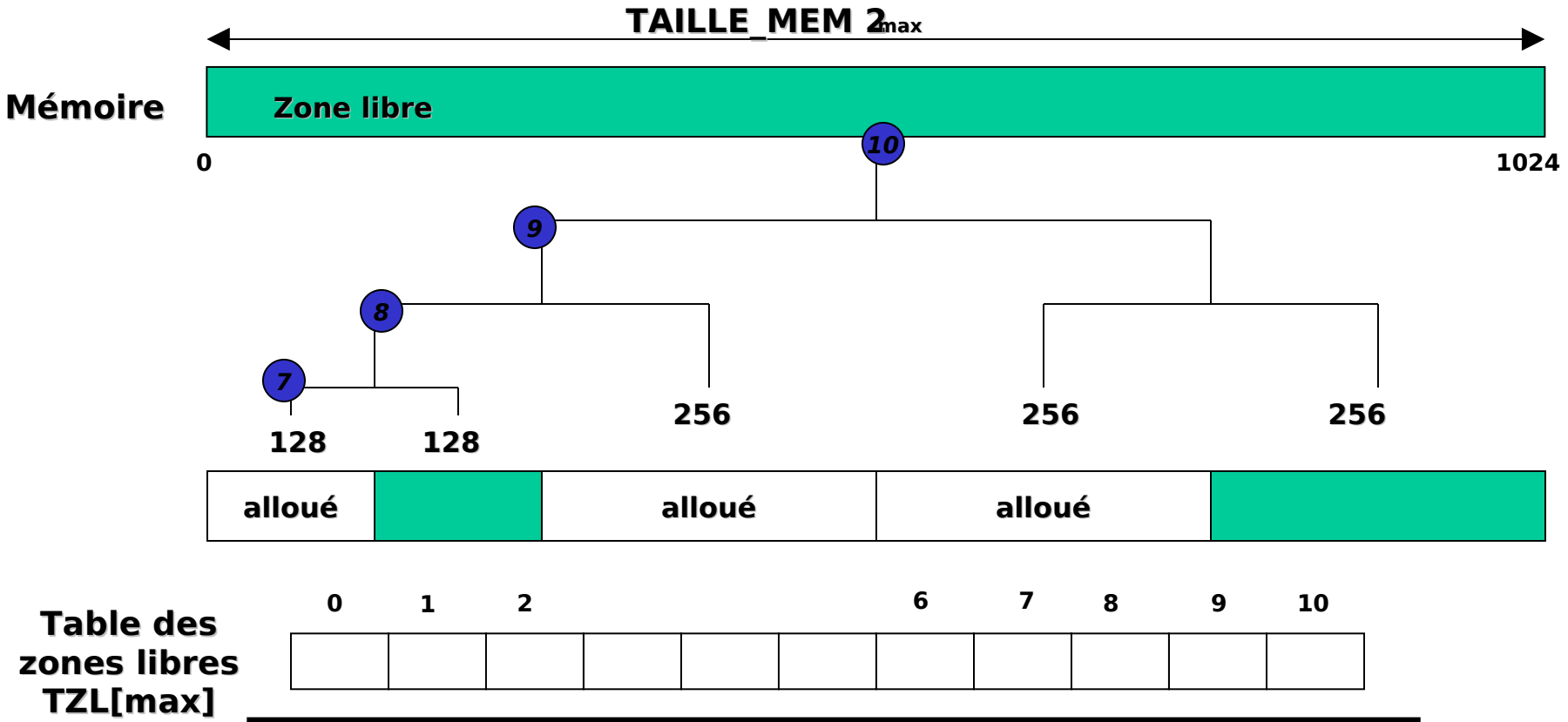
Boucle d'allocation : bloc de taille *p* avec $2^{k-1} < p < 2^k$





Allocation dans le 'buddy system'

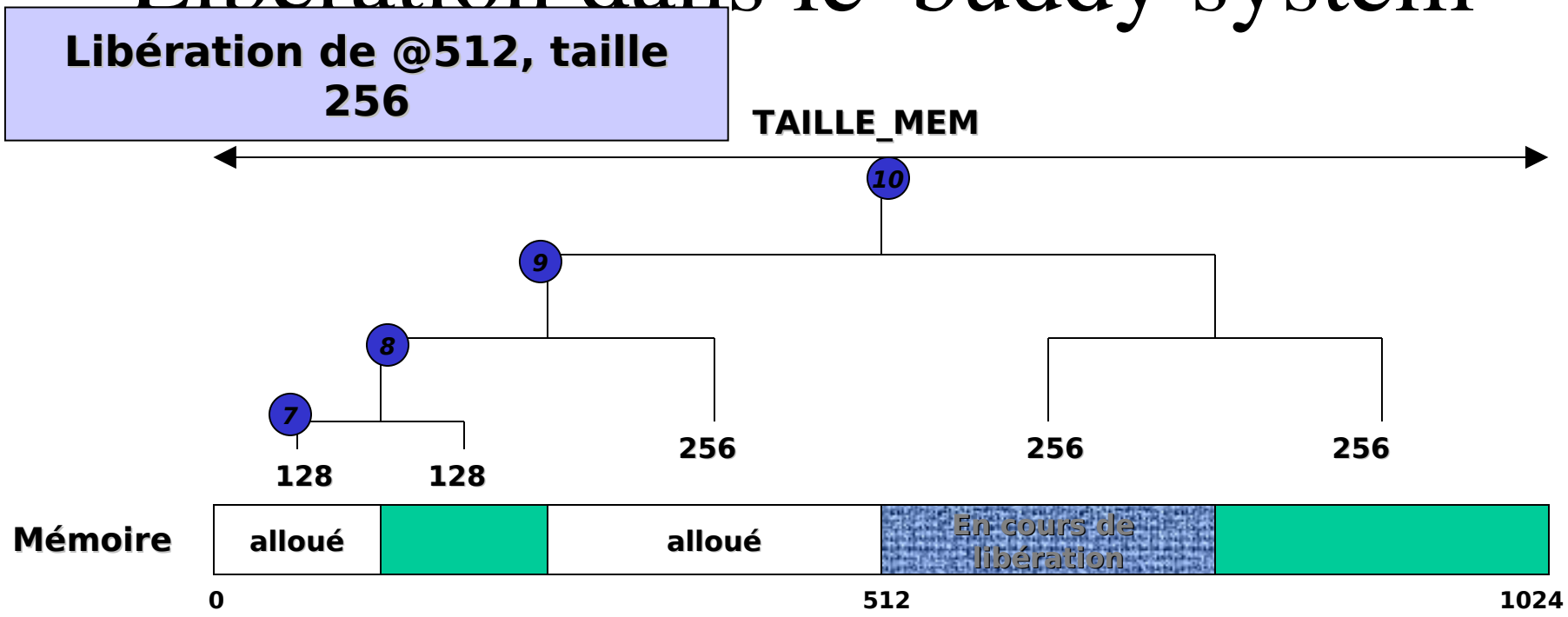
Exemple : taille 1024, blocs alloués



Questions

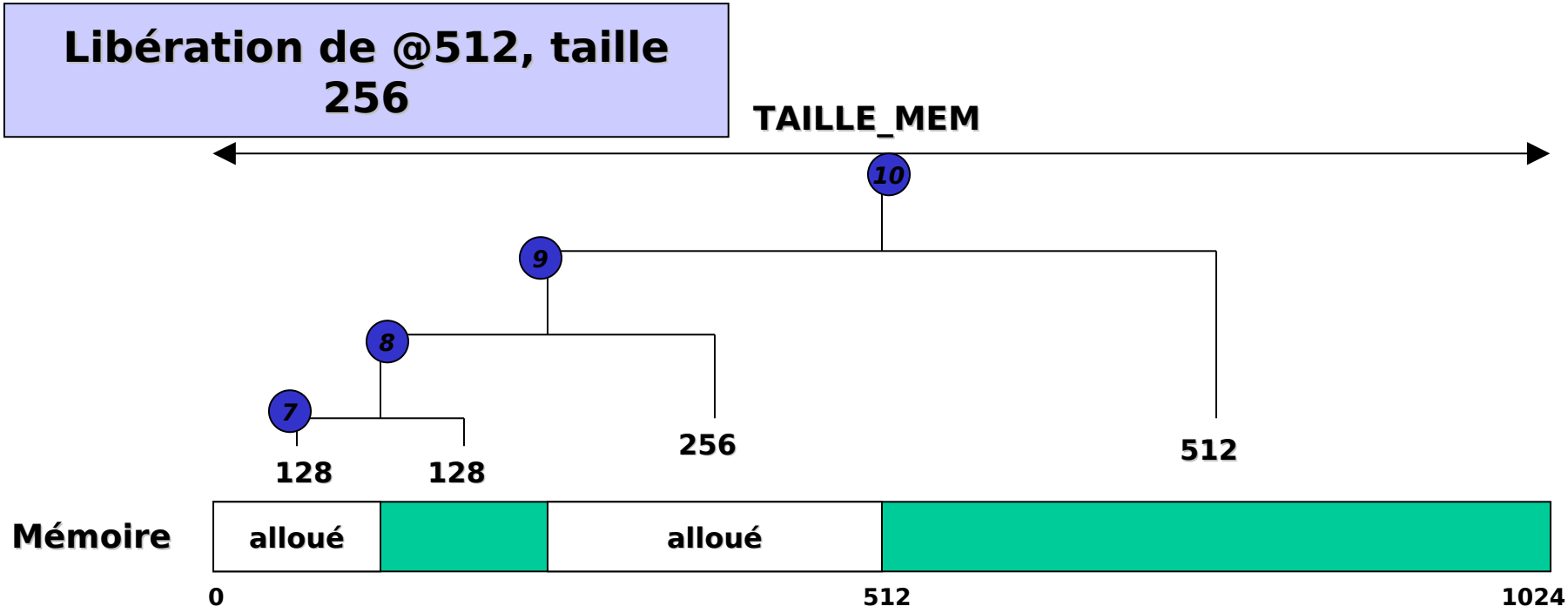
1. remplir la table des zones libres
2. Avec quelle(s) séquence(s) de primitive(s) est-on arrivé à cet état ?

Libération dans le 'buddy system'



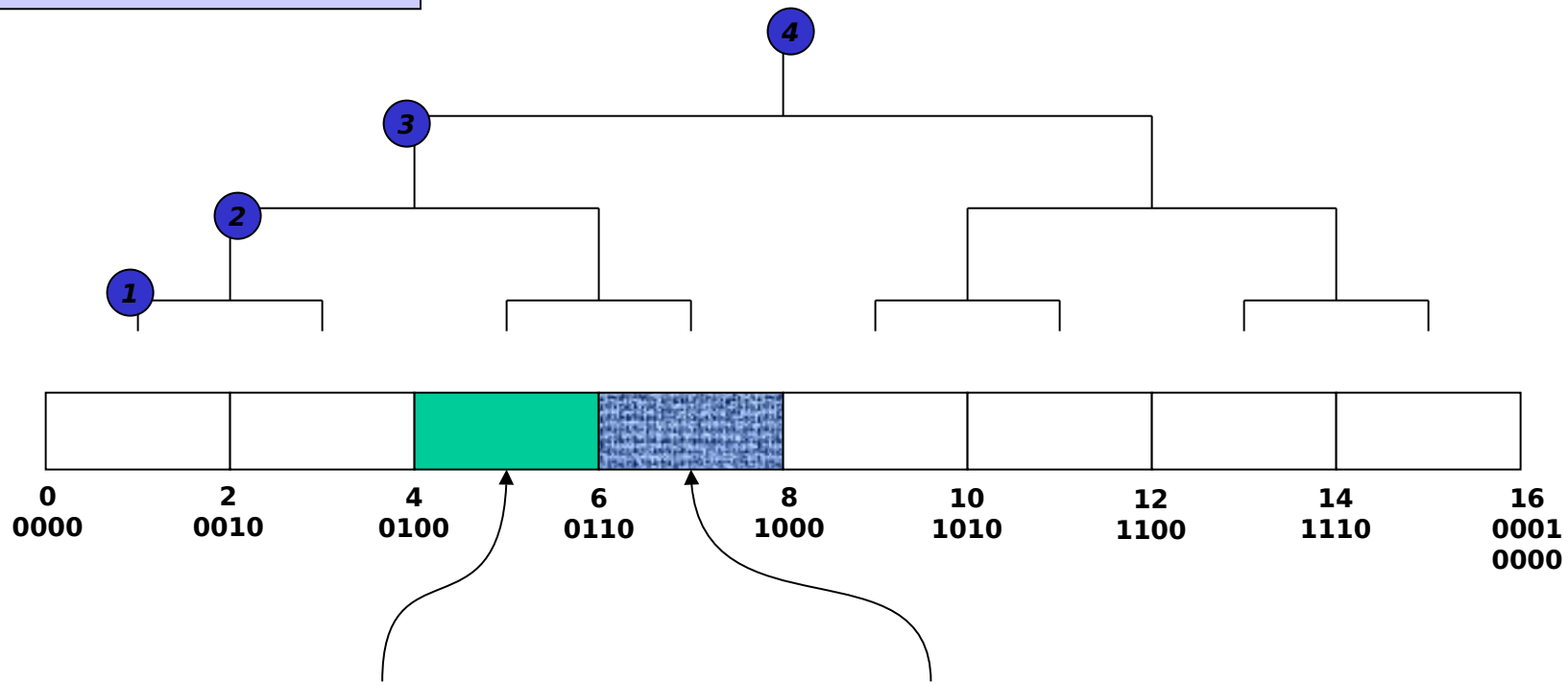
- Recherche de la zone à libérer
- Recherche du compagnon
- Fusion avec le compagnon s'il est libre
- Mise à jour de la table des zones libres

Libération dans le 'buddy system'



Recherche du compagnon

Mémoire de 16o



Compagnon : @4 Bloc : @6, taille 2^1

Questions

1. **Algorithme efficace de calcul du compagnon**

Libération par ramasse-miettes

- Allocations explicites
- Pas de libérations
  - La mémoire est grande, on verra bien
- En cas d'épuisement de la mémoire libre
  - Détermination des zones libres et occupées
  - Création de la liste libre