Physical memory management

Noël de palma
Jacques Mossiere
Luc Bellissard

Plan

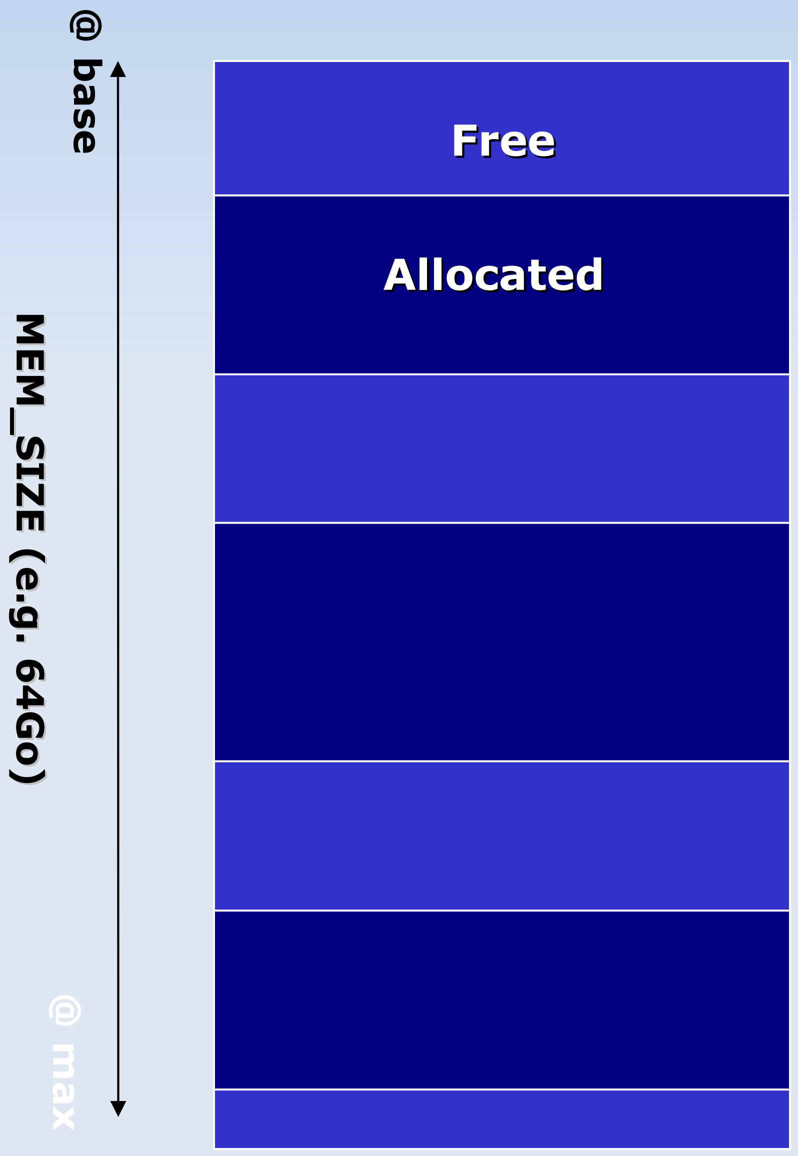
1- Physical memory management

2- Algorithms

Physical Memory management

- Goals
  - Provide memory zone to programs
  - Manage available memory zone
- Two kinds of memory
  - Physical memory
    - Management of the hardware memory (e.g. RAM)
  - Virtual memory
    - Provide a memory space larger than available for user processes

Physical memory management



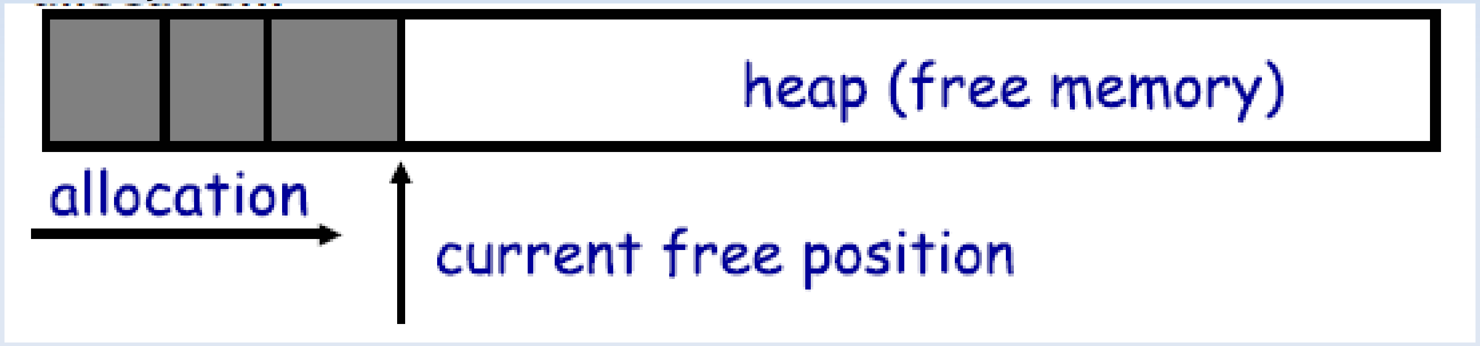
- Know the physical memory
- Track free memory zone, used zone
- Provide free memory upon program request
  - malloc
- Free the zone upon program request
  - free

User

- Operating system
  - Use memory for itself
  - Virtual memory management
- User Processes
  - Dynamic memory allocation requested by processes

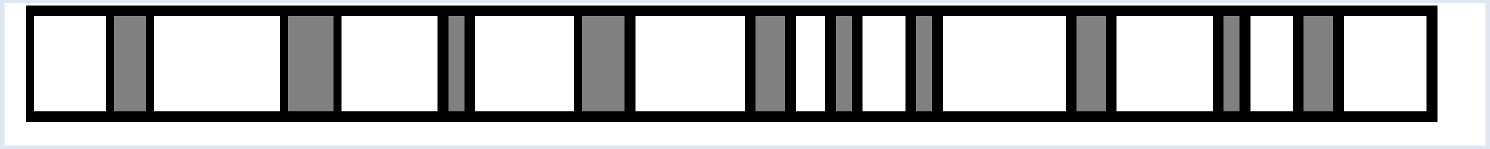
Why it is hard

- Satisfy arbitrary set of allocation and free's.
- Easy without free: set a pointer to the beginning of free memory



Problem: free creates holes (fragmentation") Result?

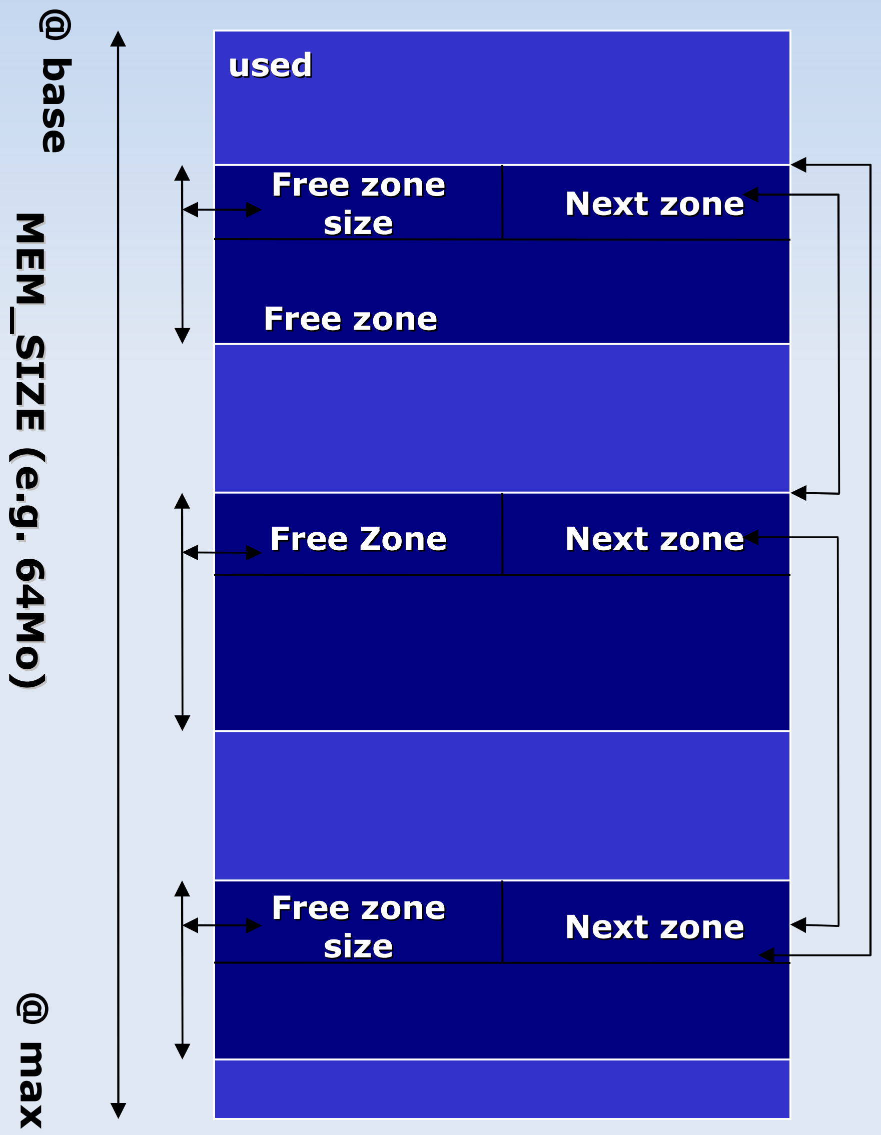
Lots of free space but cannot satisfy big request!



Structure and user API

- Data structure to describe the memory and its usage
  - Zone description (base@, size, …)
  - Free and/or used zone table
- Initialization
  - Initialize the data structure
- Memory allocation
  - Retour the @ of a free zone of contiguous memory of the given size
  - malloc(size) => pointer to the free zone
- Memory deallocation
  - Release a previous allocated zone
  - free(@ zone, size) => error code

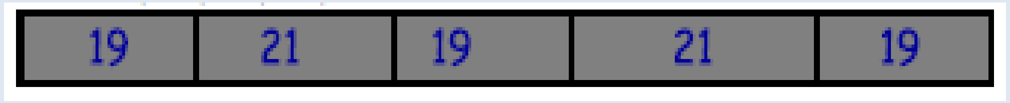
Algorithm : linked list of free block



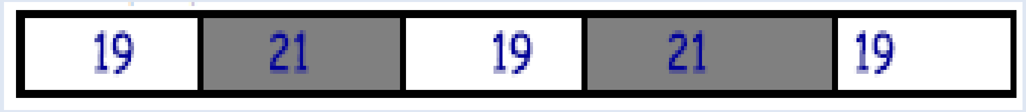
- Free zone descriptions are stored in the free zones
  - Free zone size
  - @ next free zone
- Simple or circular linked list
- Allocation
  - Inspect the free zone list
  - Choose a free zone that fits the requested size
- Various criteria: best fit, first fit, worst fit, …
- Deallocation
  - Merge neighbour free zones

Best fit goes wrong

- Simple bad case: allocate n, m (n < m) in alternating orders
- free all the ns, then try to allocate an n + 1
- Example: start with 100 bytes of memory
- Alloc 19, 21, 19, 21, 19



- Free 19, 19, 19:



- alloc 20? Fails! (wasted space = 57 bytes)

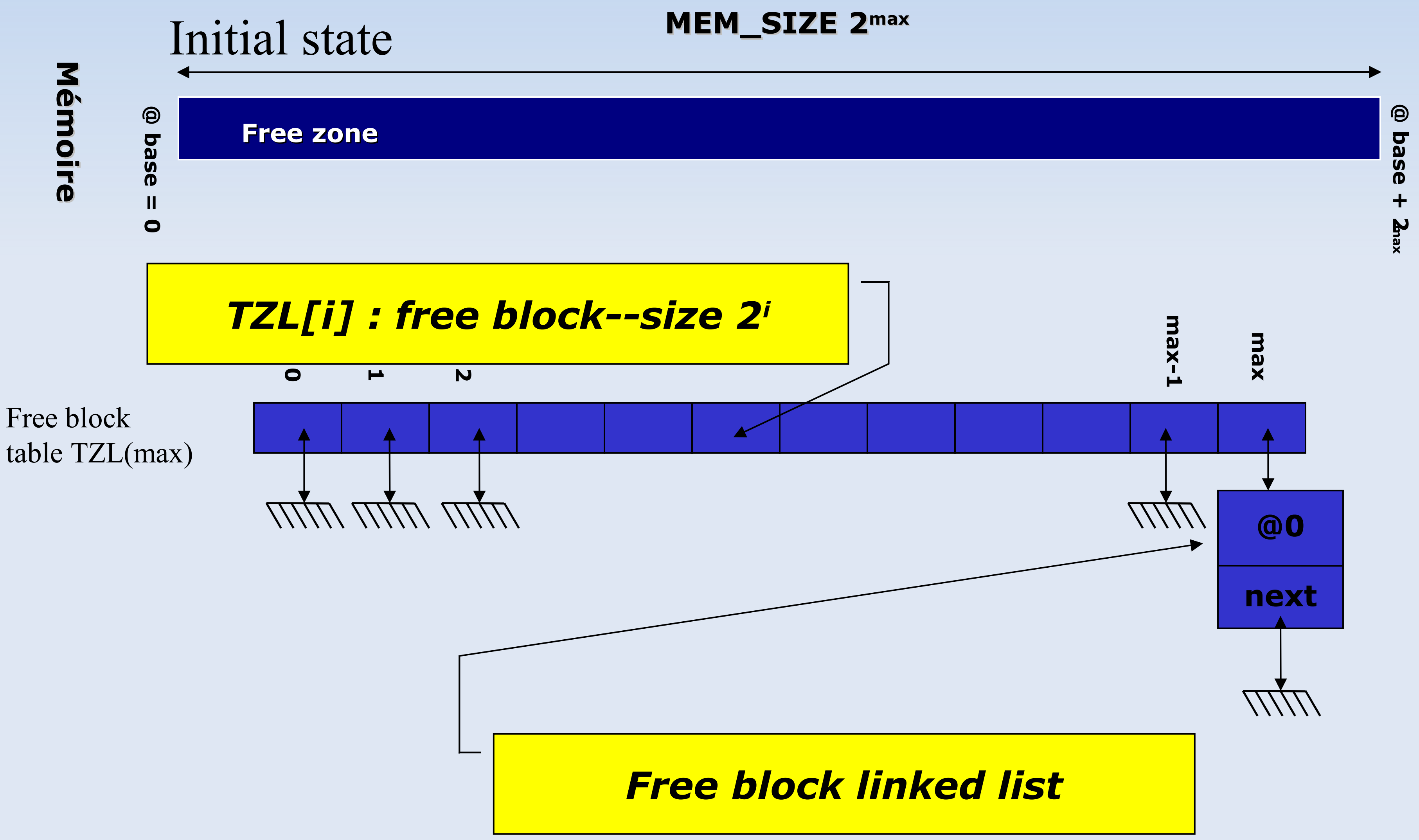
Avantages - Disadvantage

- Avantages
  - No extra memory (free zone descriptions stored in the free zones)
  - Simple Algorithm
- Disadvantage
  - Performances
  - Fragmentation

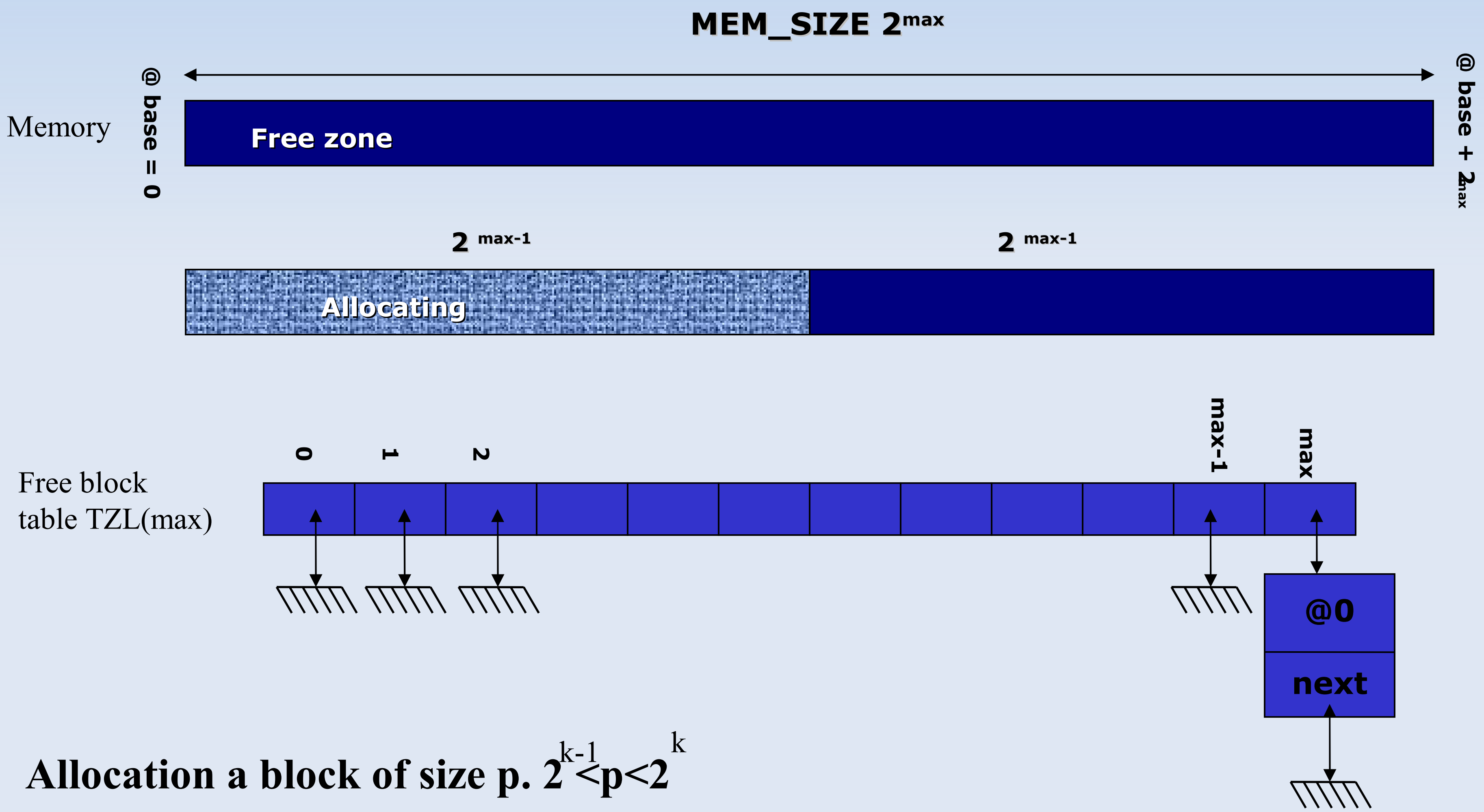
Algorithm : buddy system

- Allocate block of predefined size
- 2^k block for a memory size of de 2^{max}
- Allocation principle
  - Table of free block
  - Look for a block of size 2^k
  - Split recursively free block in two blocks of size 2^{k-1} (buddy) until the block get the right size
- De-allocation principle
  - Look for the buddy of the free block
  - Merge the buddy if possible to provide a bigger one

Allocation with the buddy system

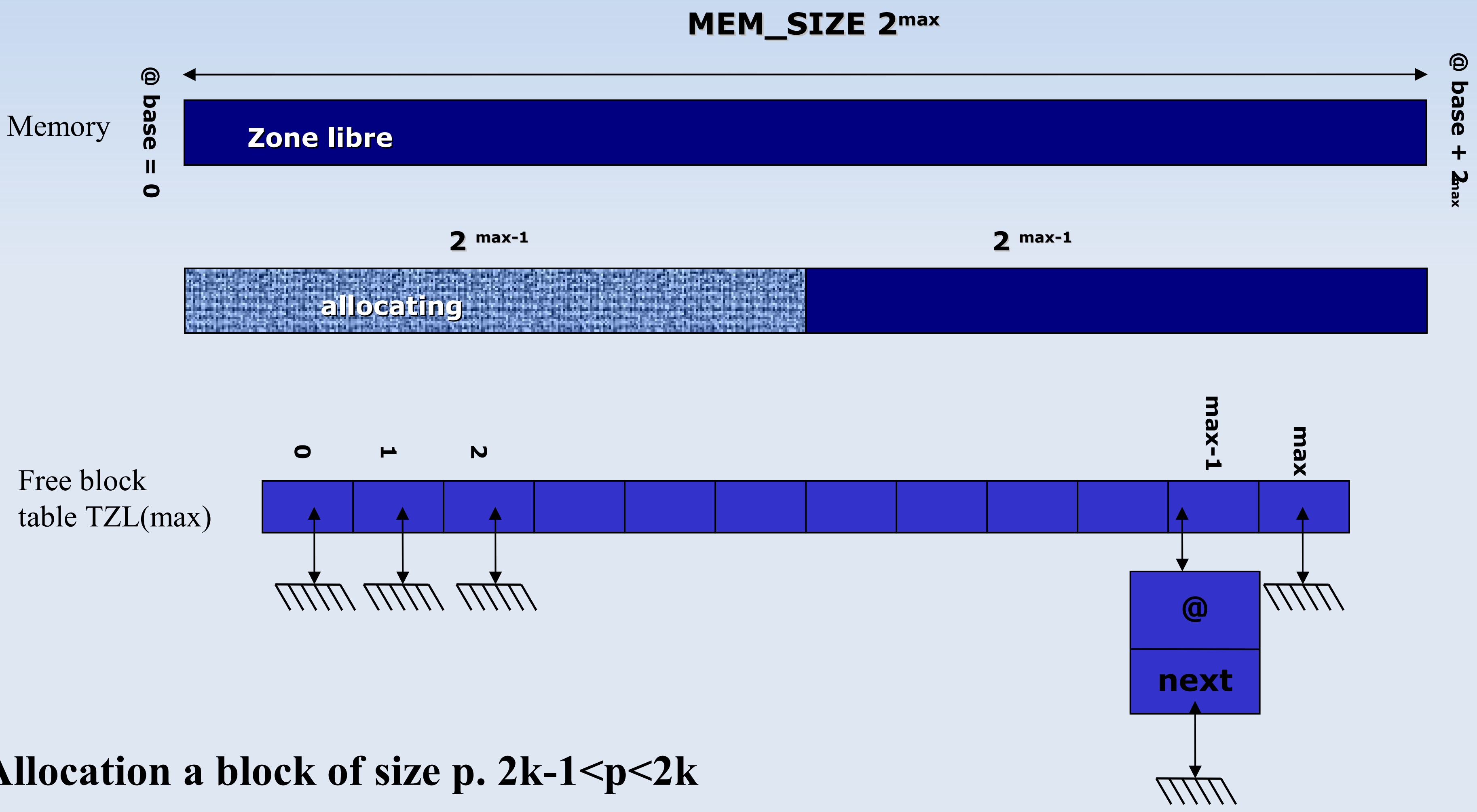


Allocation with the buddy system



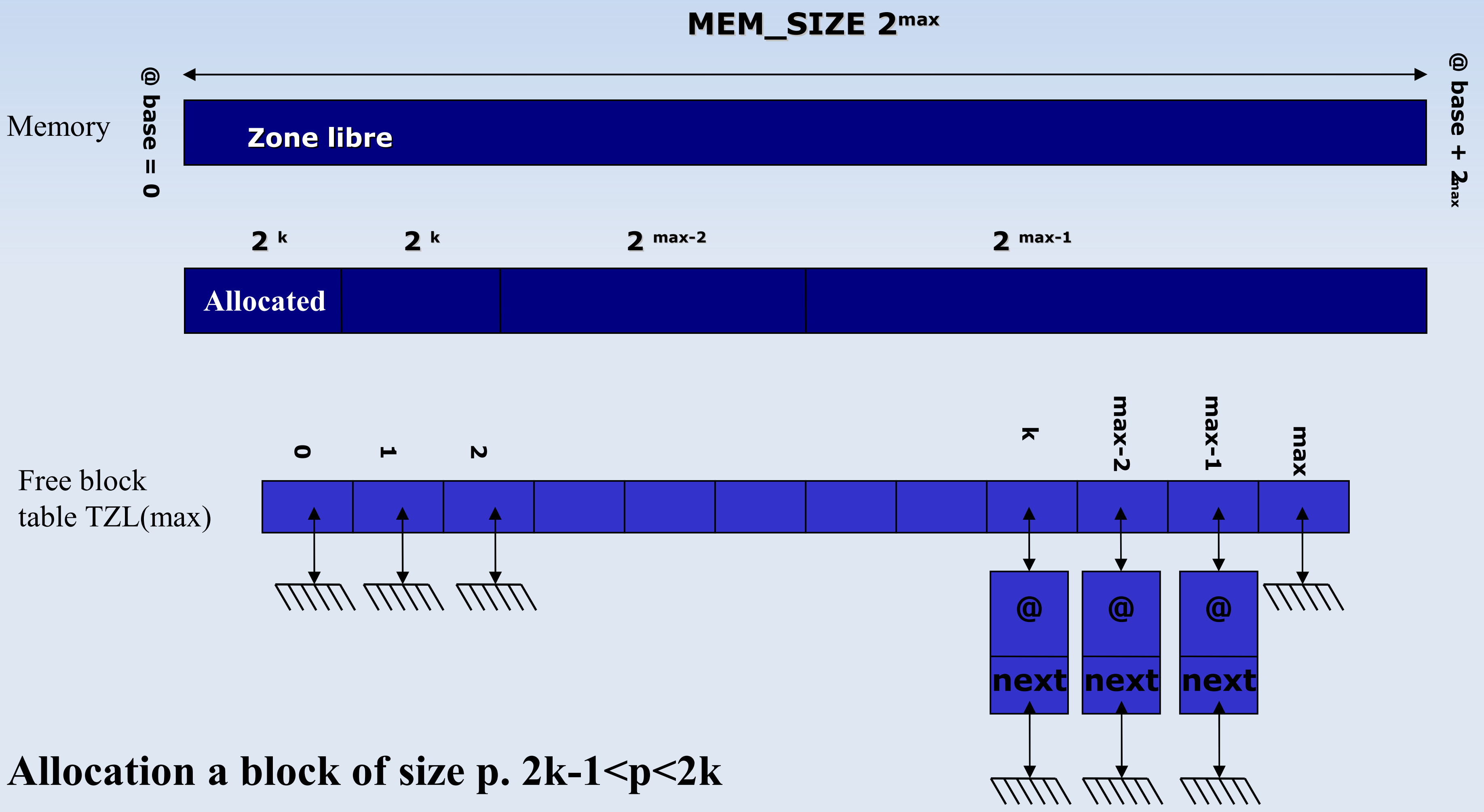
Allocation a block of size p. $2^{k-1} < p < 2^{k}$

Allocation with the buddy system



Allocation a block of size p. 2k-1<p<2k

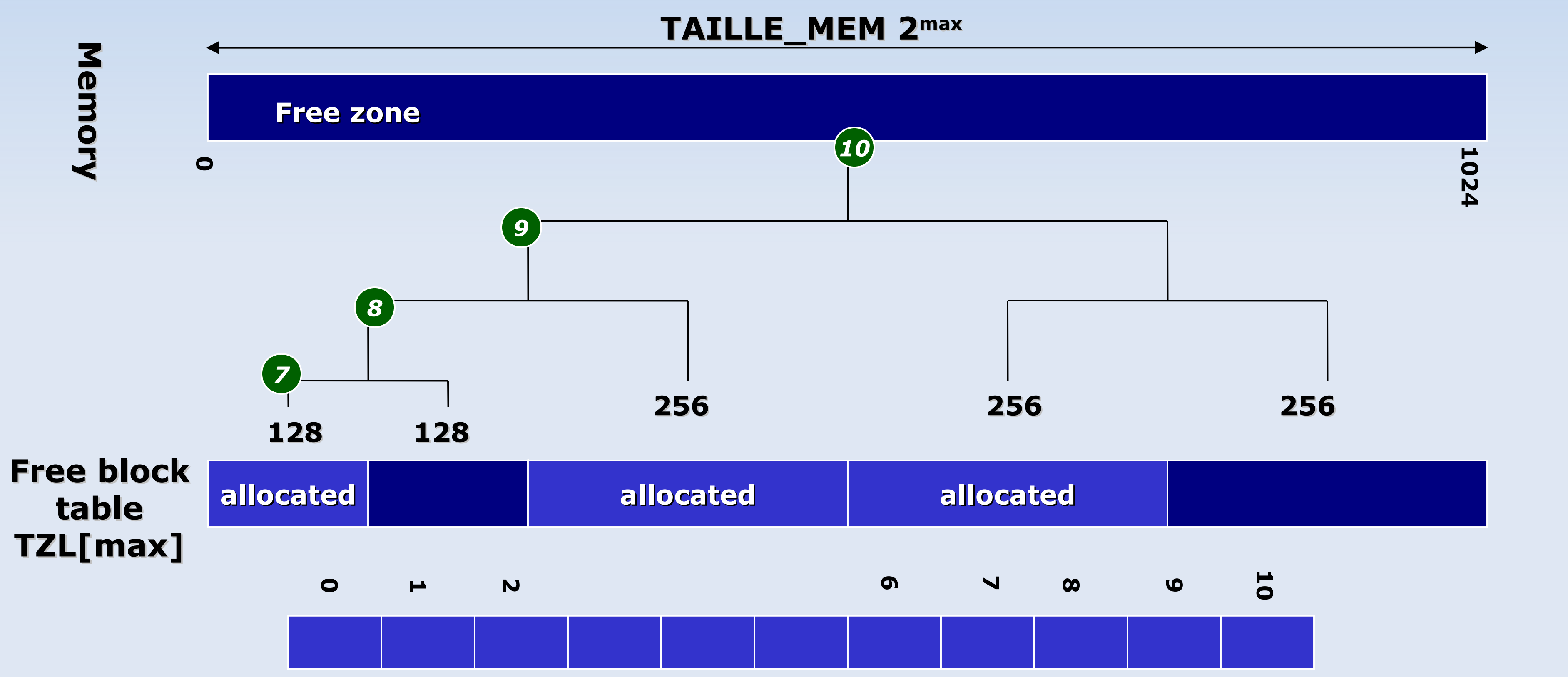
Allocation with the buddy system



Allocation a block of size p. 2k-1<p<2k

Allocation with the buddy system

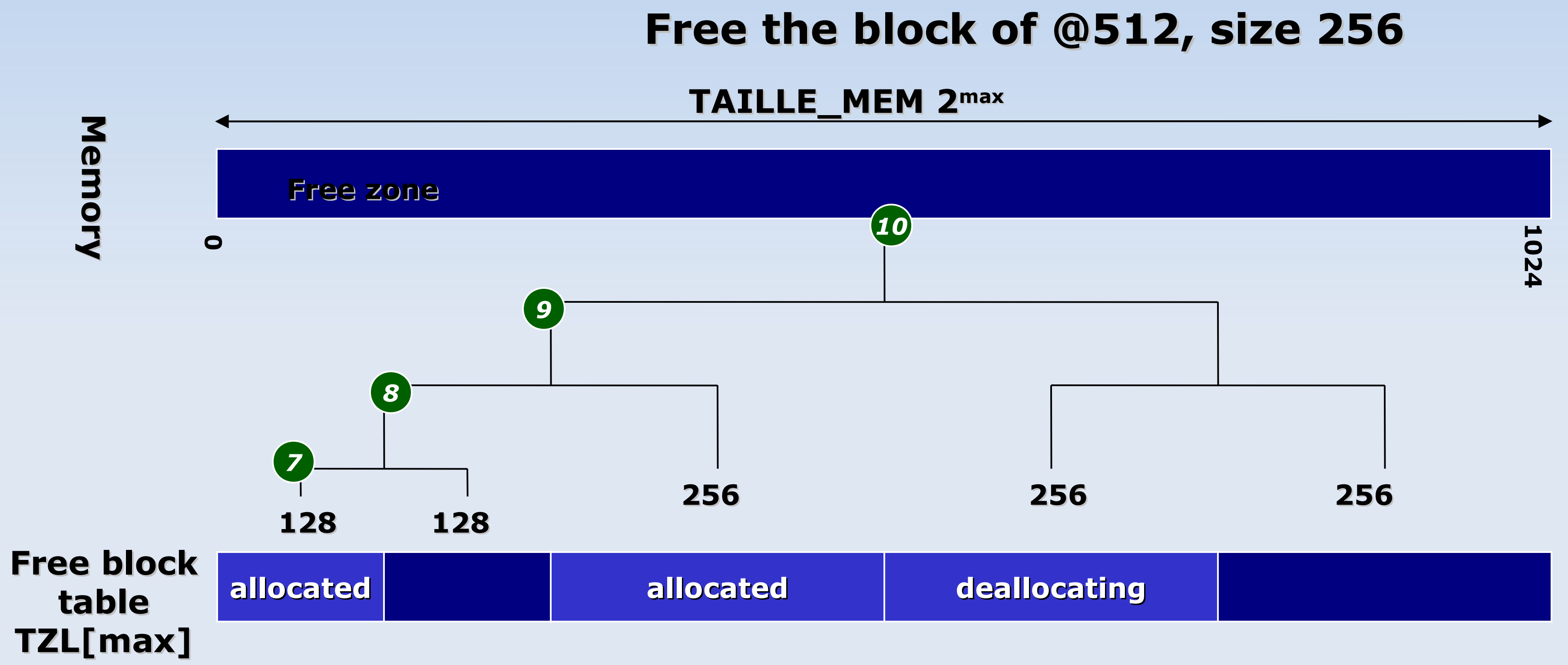
Example : max size 1024



Questions

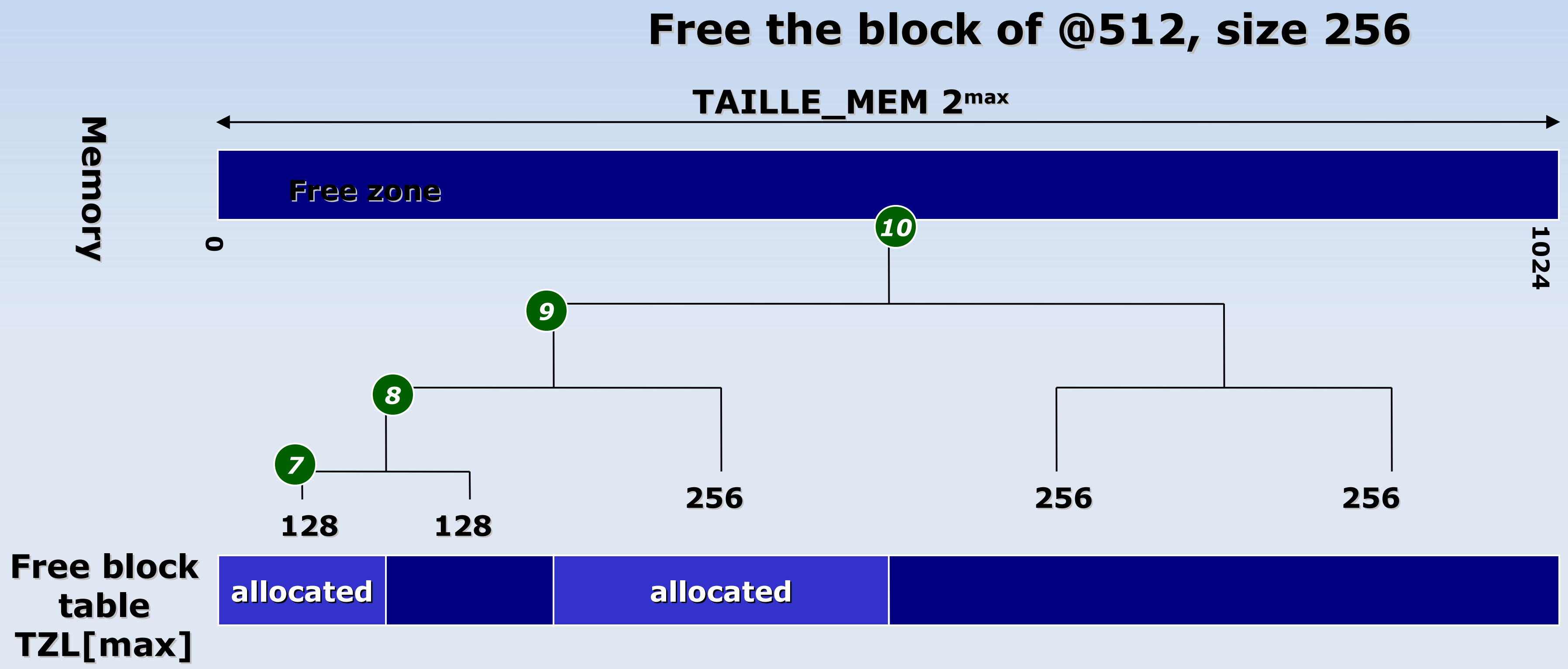
Fill the free block table

De-Allocation with the buddy



Look for the block to de-allocate
Look for the buddy
Merge with the buddy if it is free
Update the free block table

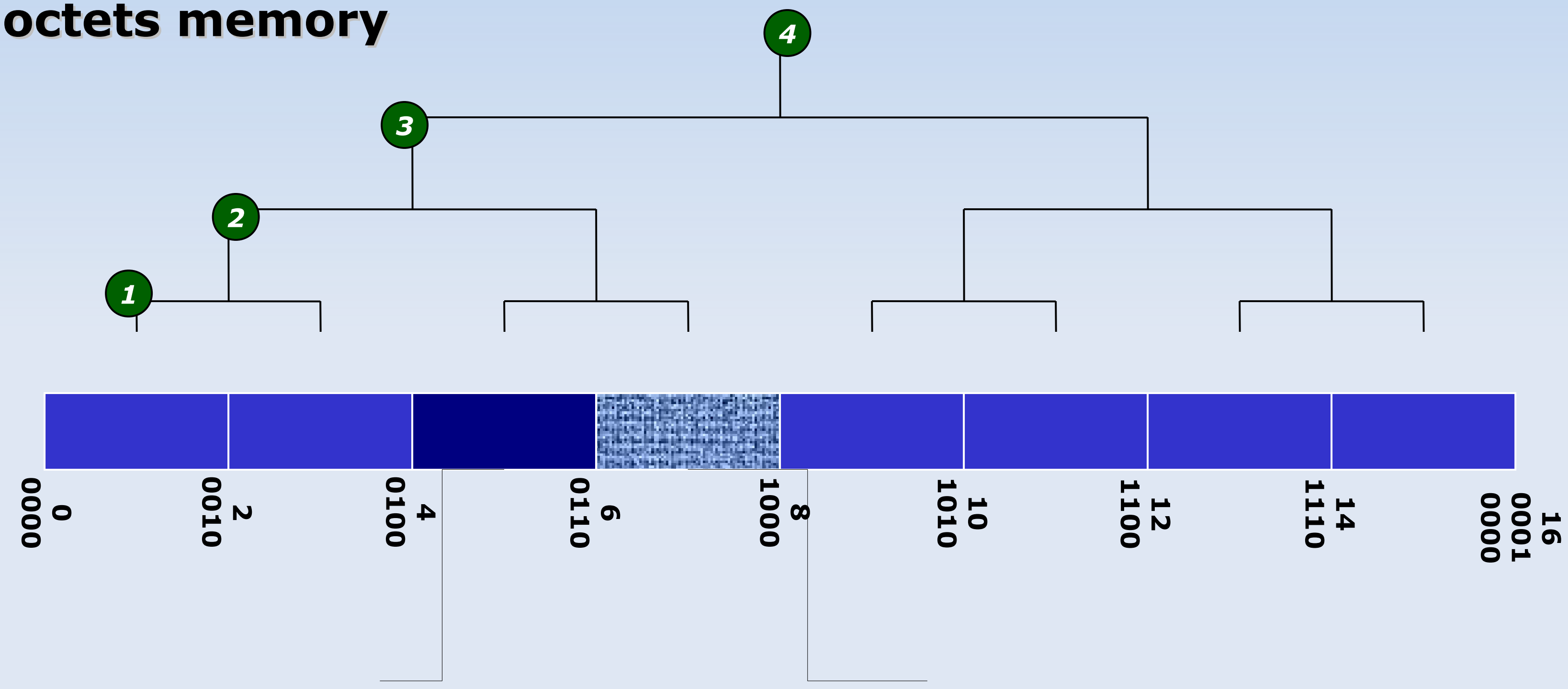
De-Allocation with the buddy



Look for the block to de-allocate
Look for the buddy
Merge with the buddy if it is free
Update the free block table

Looking for the buddy

16 octets memory



buddy : @4 ***Block : @6, size 2^1***

Question

Efficient solution to compute the buddy @ ?