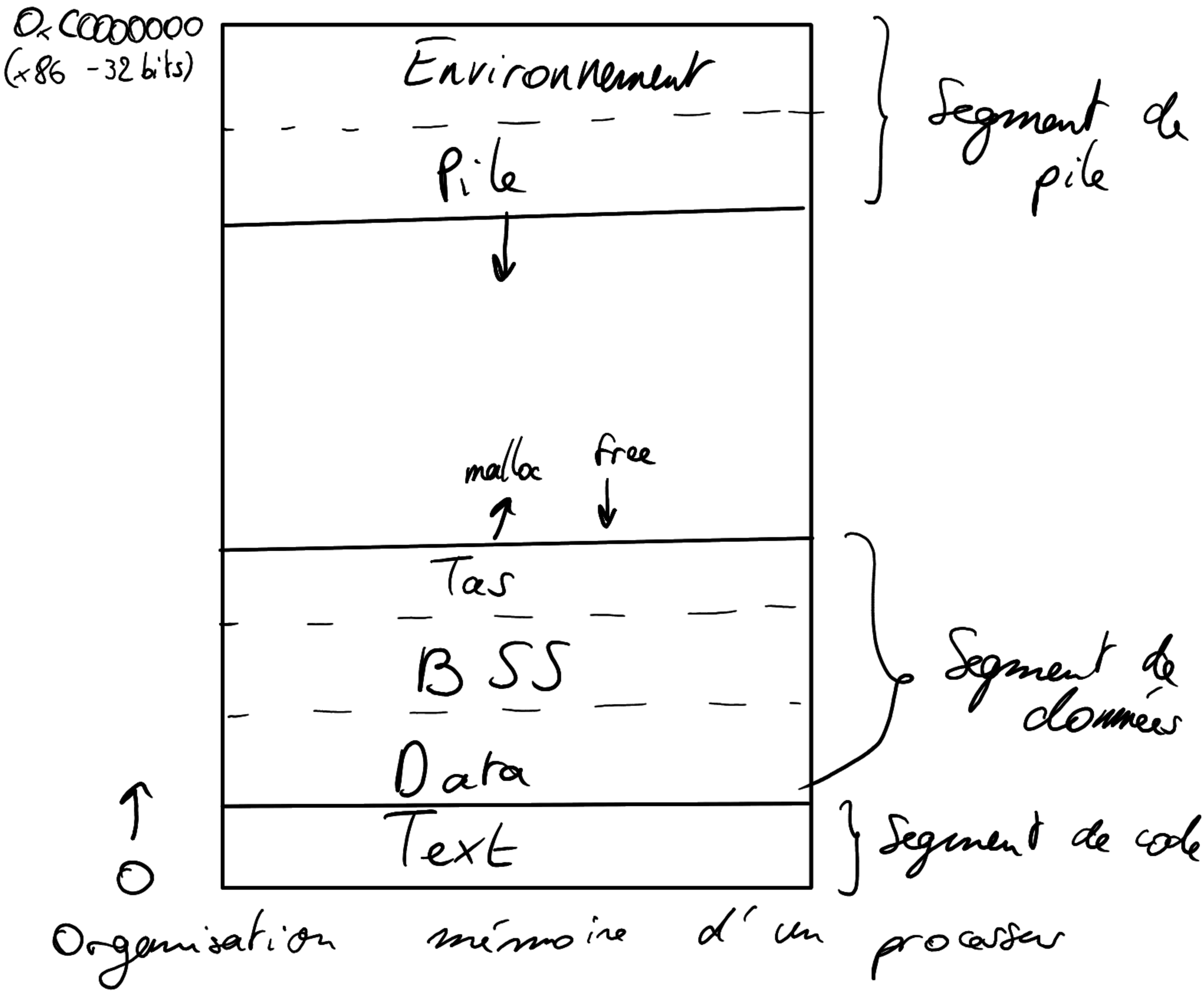
DEV : Organisation d'un processus en mémoire

Inspiration : Systèmes d'exploitation, Tanenbaum 10.4

On se place ici dans le cadre de Linux.

Idée : Modèle simple pour avoir la plus grande compatibilité possible



Organisation mémoire d'un processus

Exemple : Programme C.

```
int   foo;                    → BSS
int   bar = 3;                → Data

int   main(int argc, char **argv){
    static int x = 4;         → Data
    static int *y;            → BSS
    int z = 2;                → Pile
    foo = x + z;
    y = malloc(sizeof(int));  → Tas
    baz(y);                   → Empilement/Dépilement de l'appel
    return foo + y;
}
void baz(int *y){
    *y = 2;
}
```

Segment de données

Segment de code

Segment de données

Segment de code.

Segment de code : Contient toutes les instructions du code exécutable

Segment de données : Contient tout le stockage du programme : constantes, certaines variables, variables statiques...

↳ Séparé en 3 parties : Data / BSS / Tas

Data: Contient les données initialisées au début du programme
↳ Variables globales et variables statiques locales

BSS : Identique au Data pour les données non initialisées au début du programme
↳ Block Started by Symbol (raisons historiques)
↳ Optimisation du segment de données
↳ Toutes les données sont initialisées à 0 au début du programme.
↳ Permet de ne stocker que la taille du segment BSS dans le fichier exécutable et d'allouer des pages de 0 à l'exécution du programme.

Tas : Plage des données allouées dynamiquement
Libc → malloc/free
↳ Allocation gérée par un allocateur mémoire.
↳ Bump allocator (le plus simple)
↳ Free lists (listes chaînées)
↳ Buddy algorithm (pour Linux)

Segment de pile : Contient la pile d'appel des fonctions, les variables locales non statiques, des valeurs temporaires, ...

Pile : Initialisée avec les variables d'environnement et la ligne de commande exécutée → getenv / argc / argv.

Exemple :

```
int y = 3;
int main() {
    int x = y;
    x = x + 1;
    return x;
}
```

```
    .data
y:  .word 3

    .text
main:
...
l:  a0, y
addi a0, a0, 1
ret
...
```