

Structures Linéaires





9. Trier une liste par échange

Exemple: tri par sélection



```
fonction tri_sélection(T) :  
  N ← taille de T  
  pour j de 0 à N-2  
    iMin ← j  
    pour i de j+1 à N-1  
      si T[i] < T[iMin]  
        iMin ← i  
    échanger T[iMin], T[j]
```

Algorithme pas applicable aux listes :
Pas d'accès indexé

Dans la STL, `std::sort` et
`std::stable_sort` ne fonctionnent
pas avec `std::list` et
`std::forward_list`

Tri par sélection sur liste



- Parcours imbriqués avec deux itérateurs
- j de début à fin
- i de j à fin, cherche iMin
- Comment mettre en oeuvre échanger **iMin, *j* ?

```
fonction tri_sélection(L):  
    j ← L.début()  
    tant que j != L.fin()  
        i ← suivant(j)  
        iMin ← j  
        tant que i != L.fin()  
            si *i < *iMin  
                iMin ← i  
            i ← suivant(i)  
        échanger *iMin, *j  
        j ← suivant(j)
```

Echanger les éléments ?

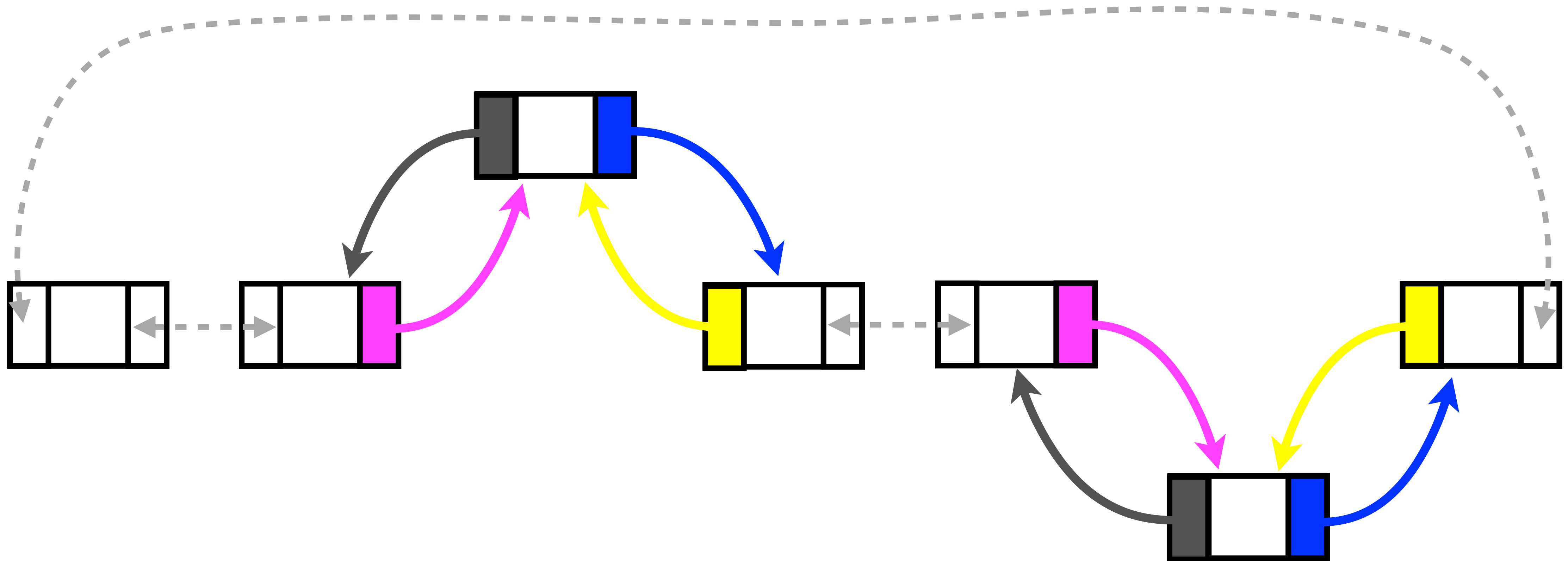


- Echanger les attributs valeur des maillons

```
fonction échanger_valeurs (m1, m2 )  
    tmp          ← m1.valeur  
    m1.valeur ← m2.valeur  
    m2.valeur ← tmp
```

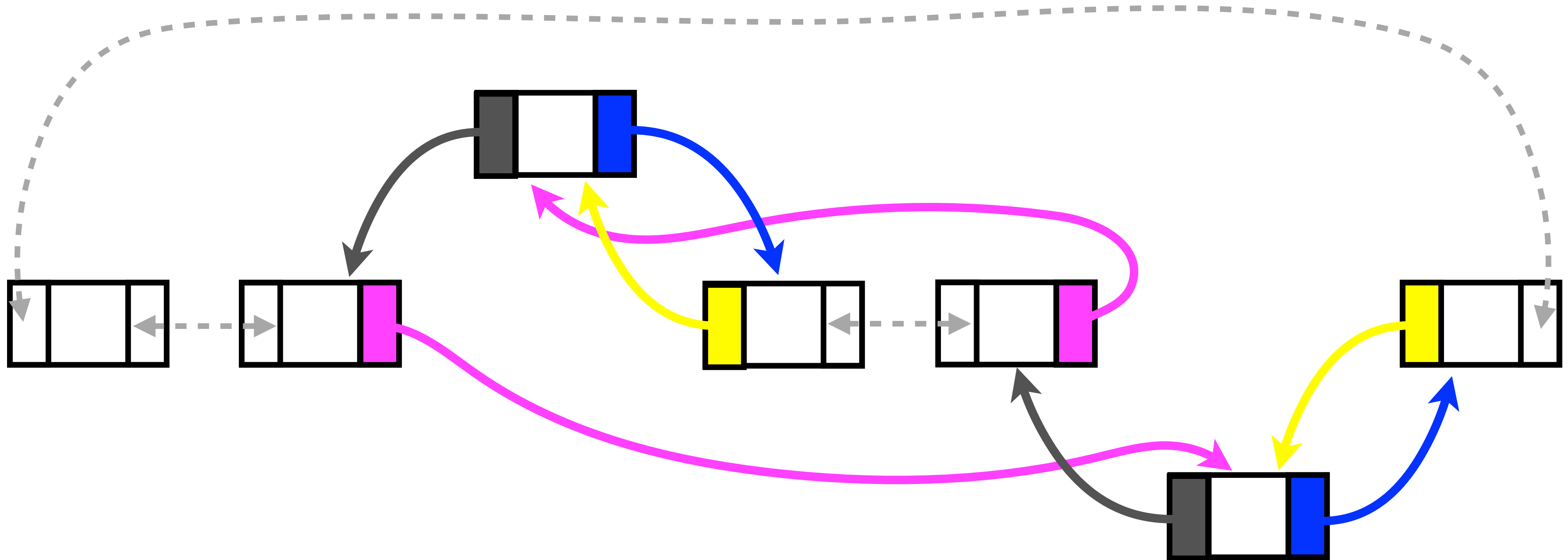
- En général pas approprié si le choix d'une liste est pertinent

Echanger les maillons



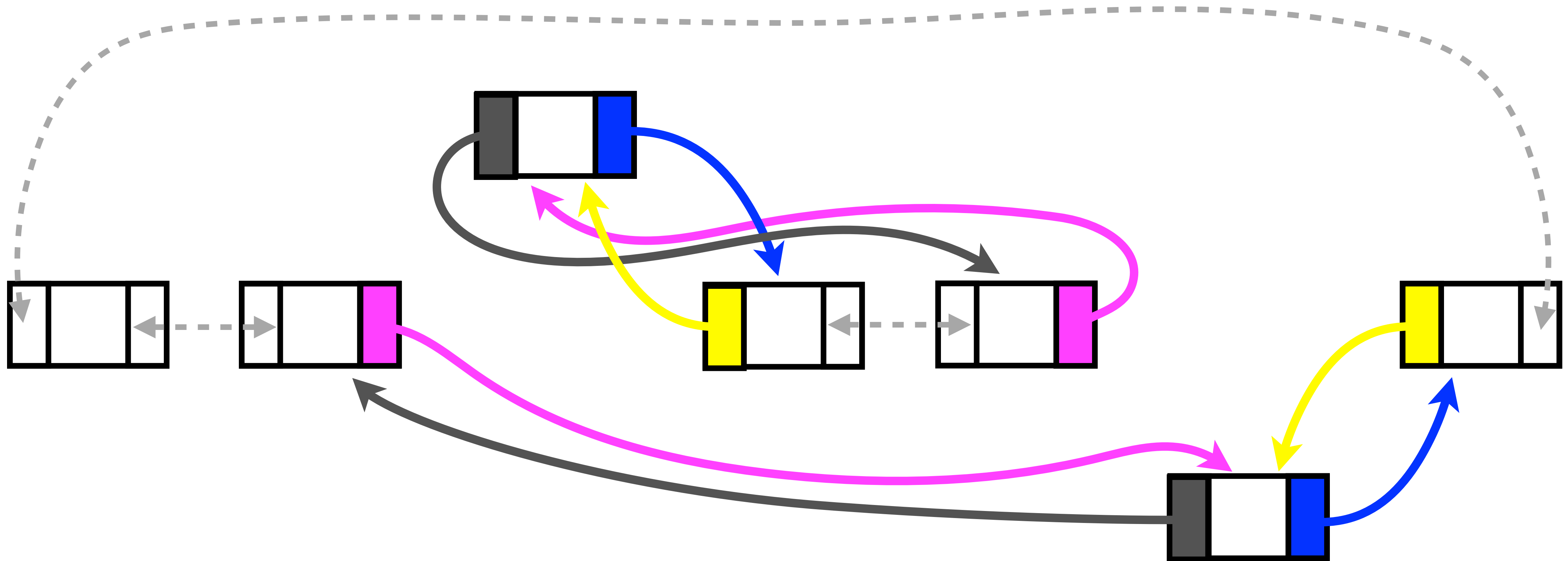
On veut swap entre elles les valeurs de

Echanger les maillons



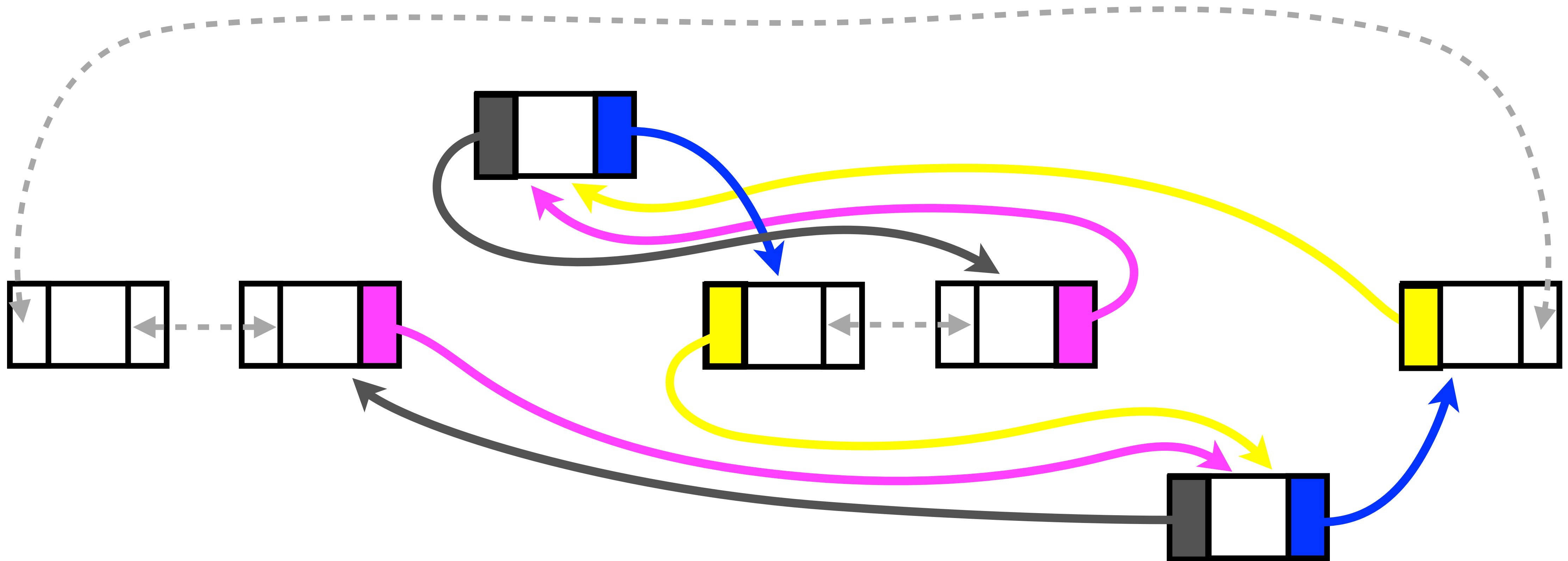
On veut swap entre elles les valeurs de

Echanger les maillons



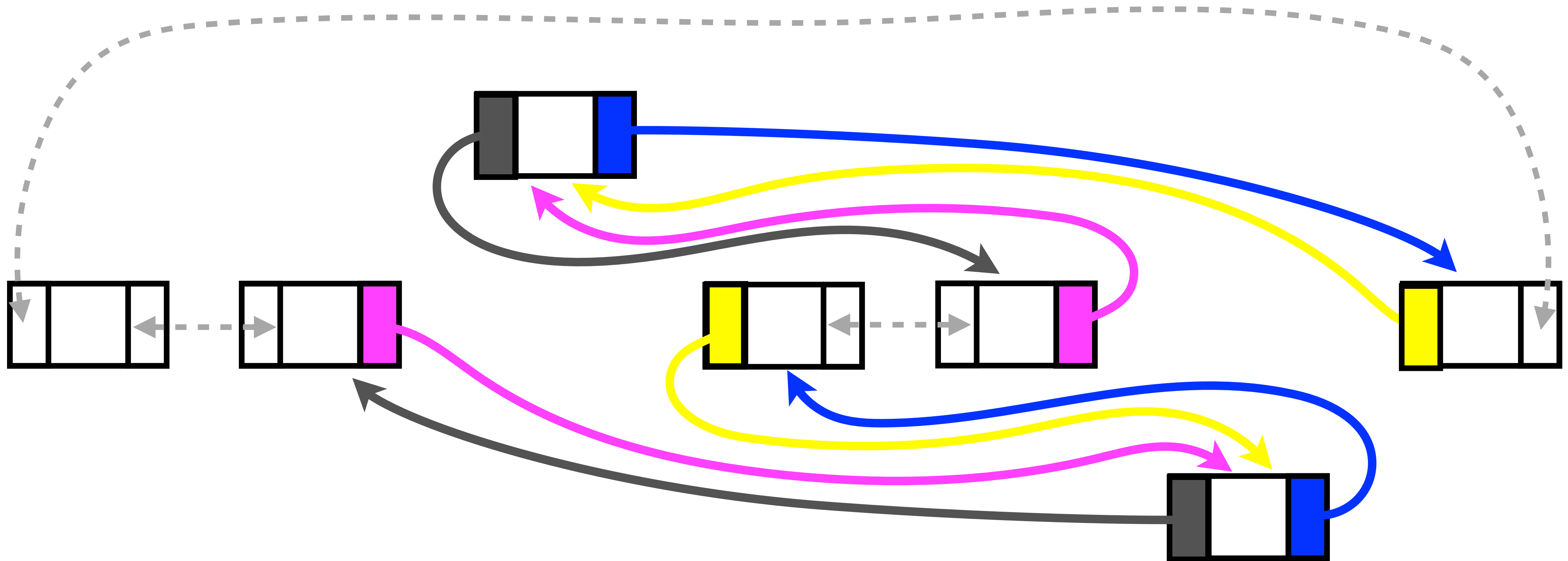
On veut swap entre elles les valeurs de

Echanger les maillons



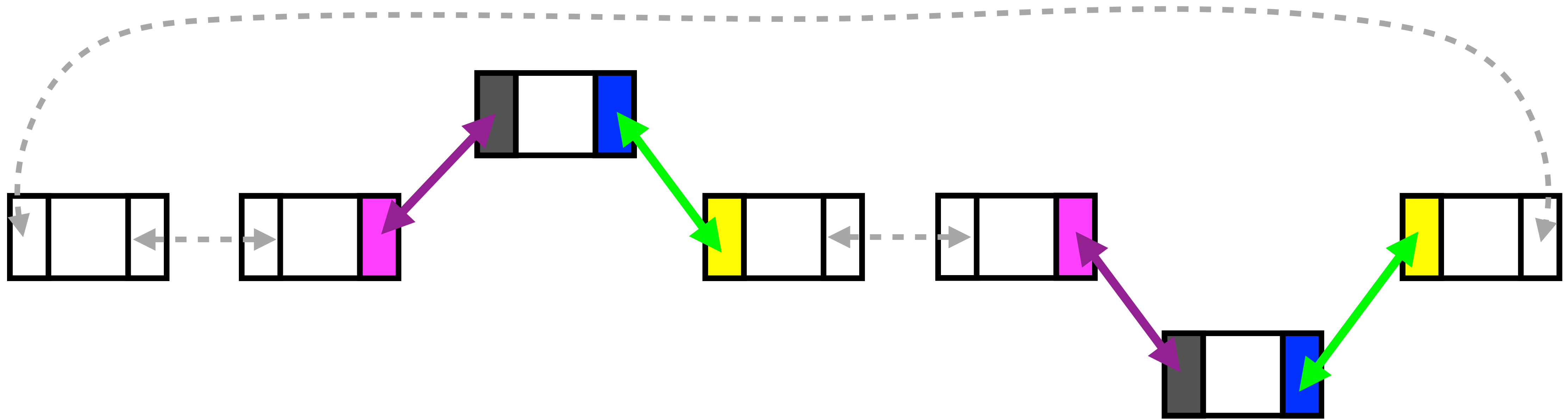
On veut swap entre elles les valeurs de

Echanger les maillons



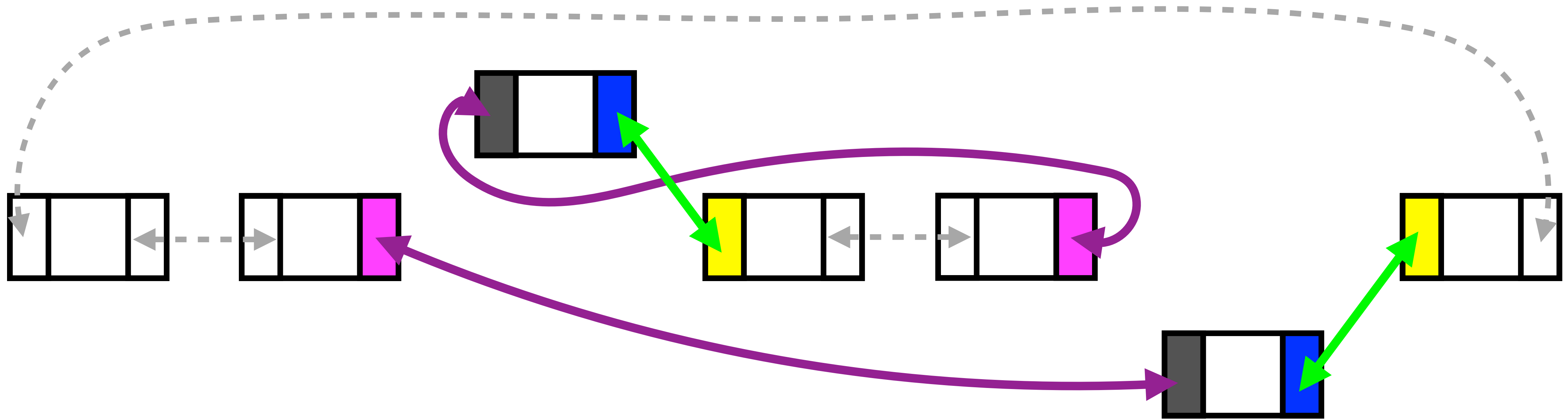
On veut swap entre elles les valeurs de

Echanger les maillons



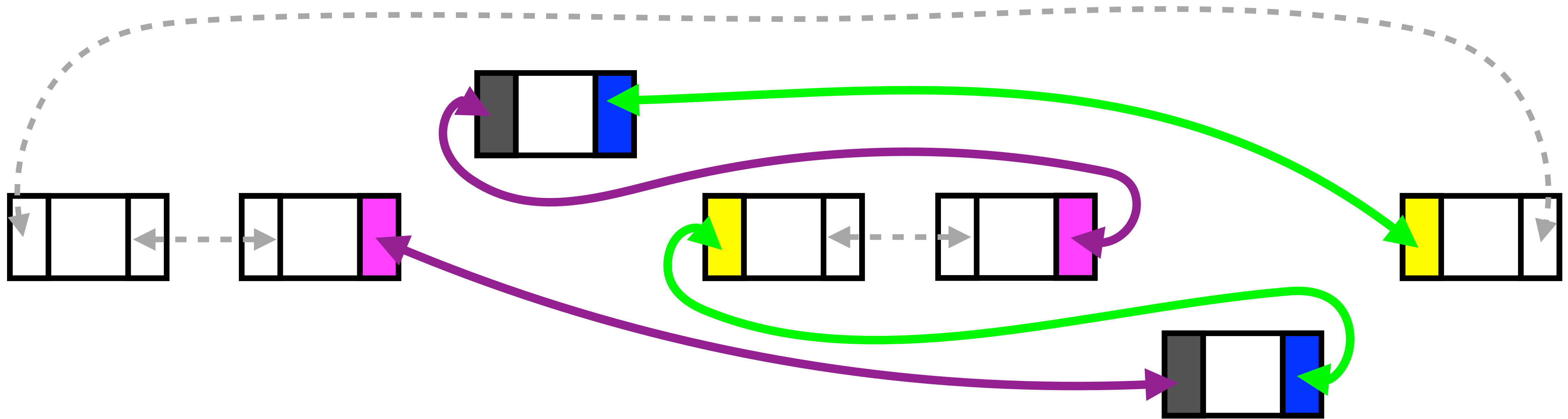
On veut swap entre elles les valeurs de

Echanger les maillons



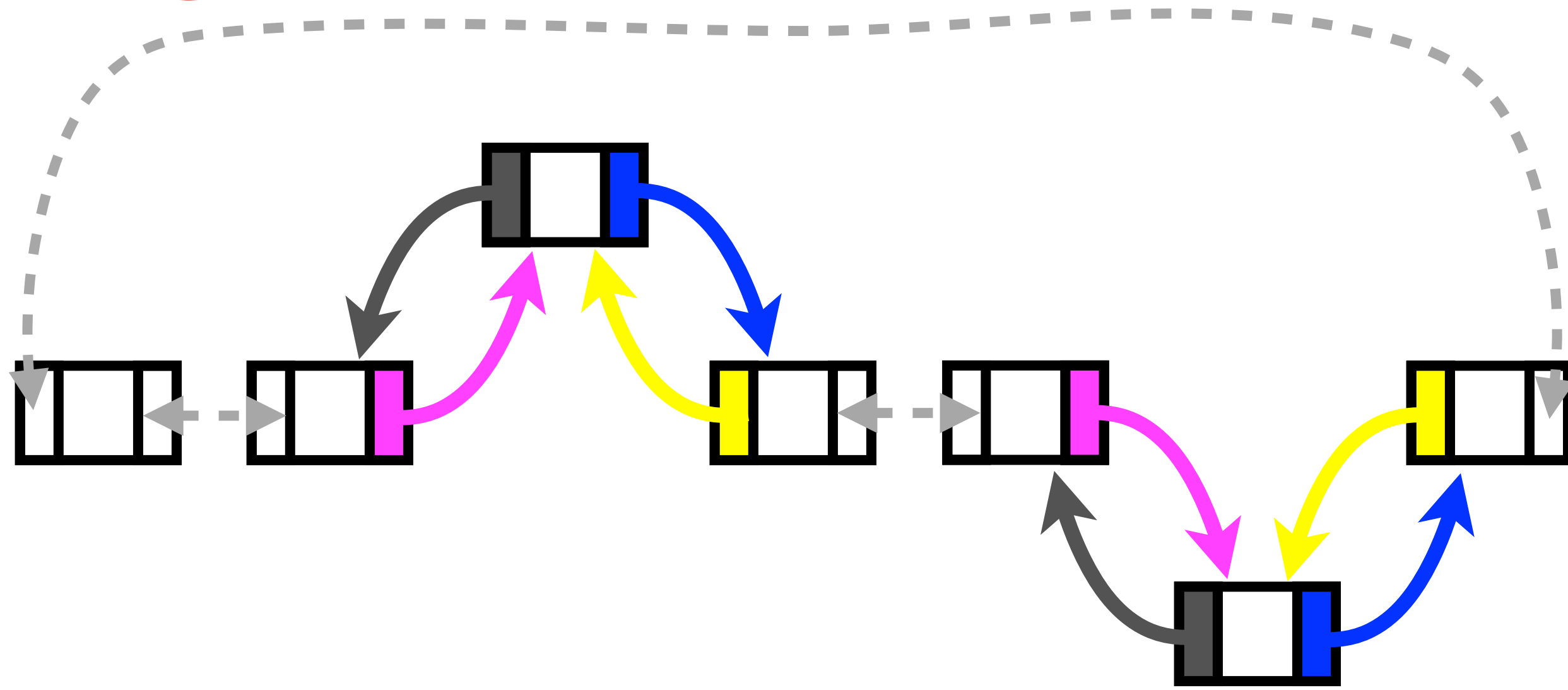
On veut swap entre elles les valeurs de

Echanger les maillons



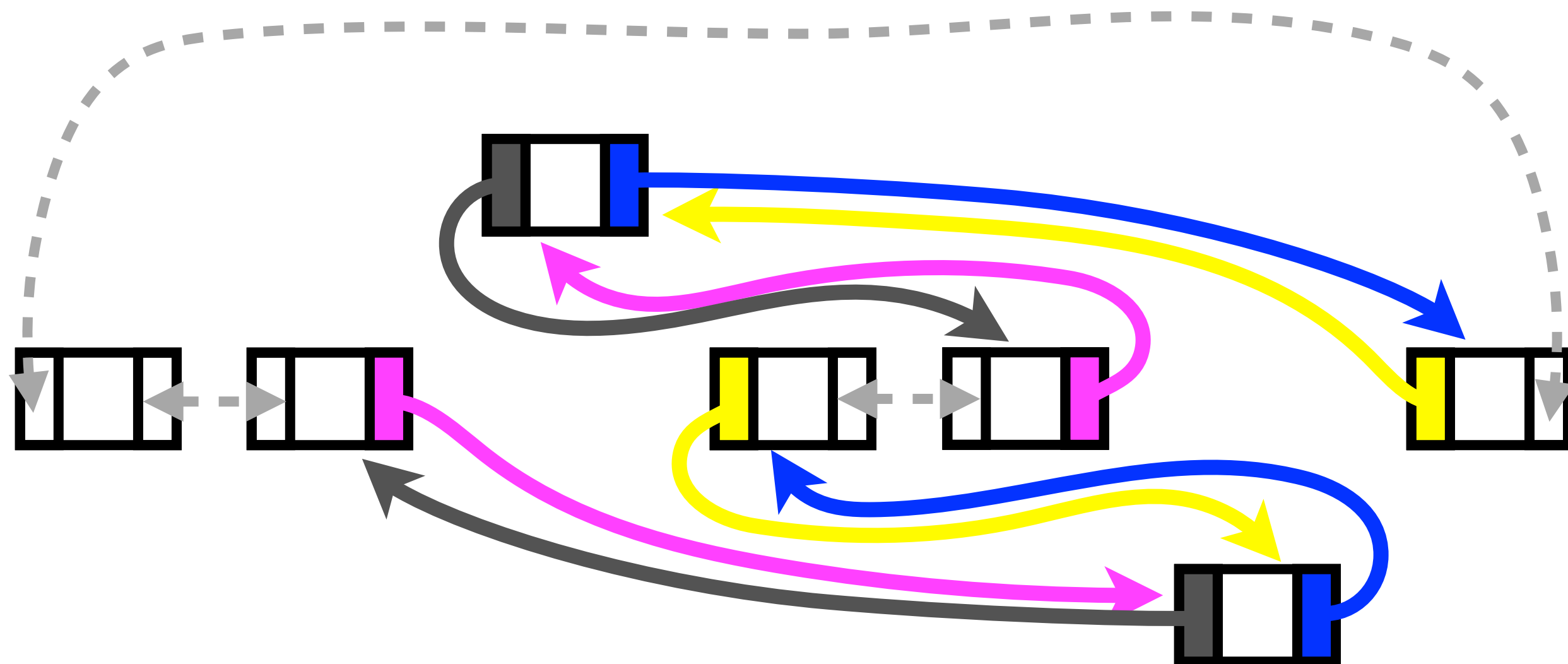
On veut swap entre elles les valeurs de

Echanger les maillons

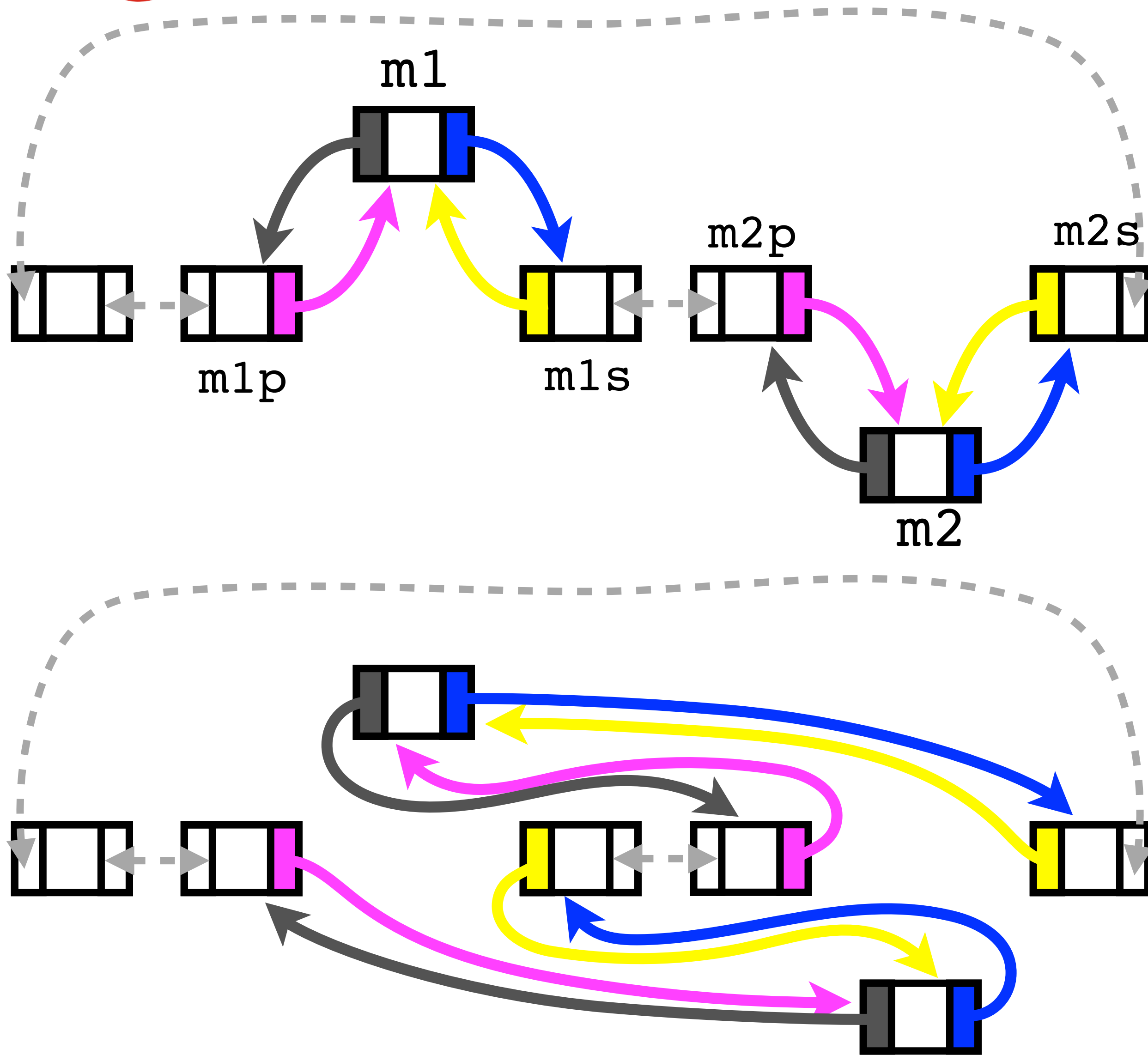


À vous de faire !

Lien sur cyberlearn



Echanger les maillons



```
fonction échanger_maillons(m1, m2)
```

```
    si m1 == m2, retourner
```

```
    m1p ← m1.precedent
```

```
    m1s ← m1.suivant
```

```
    m2p ← m2.precedent
```

```
    m2s ← m2.suivant
```

```
    Éch. m1p.suivant, m2p.suivant
```

```
    Éch. m1.precedent, m2.precedent
```

```
    Éch. m1.suivant, m2.suivant
```

```
    Éch. m1s.precedent, m2s.precedent
```

Tri par sélection sur liste (2)



```
fonction tri_sélection(L):  
    j ← L.début()  
    tant que j != L.fin()  
        iMin ← j  
        i ← suivant(j)  
        tant que i != L.fin()  
            si *i < *iMin  
                iMin ← i  
            i ← suivant(i)  
        échanger_maillons(iMin, j)  
        j ← suivant(iMin)
```

- Utilise la fonction `échanger_maillons`
- Attention, elle change la position des itérateurs `j` et `iMin` dans la liste

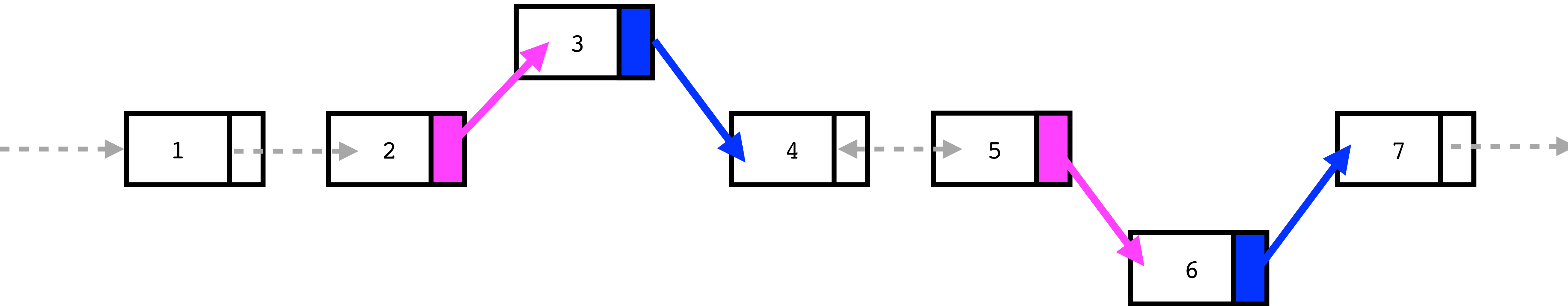
Tri par sélection sur liste (2)



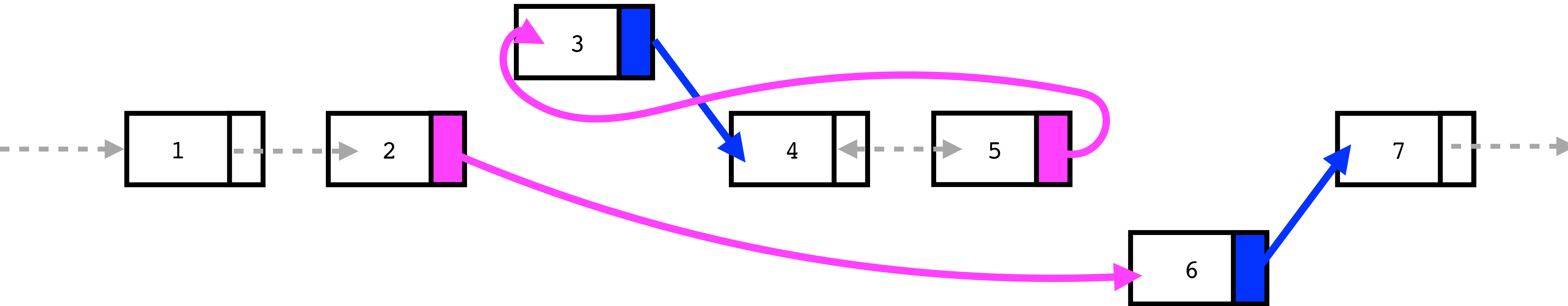
```
fonction tri_sélection(L):  
    j ← L.début()  
    tant que j != L.fin()  
        iMin ← j  
        i ← suivant(j)  
        tant que i != L.fin()  
            si *i < *iMin  
                iMin ← i  
            i ← suivant(i)  
        échanger_maillons(iMin, j)  
        j ← suivant(iMin)
```

- Utilise la fonction `échanger_maillons`
- Attention, elle change la position des itérateurs `j` et `iMin` dans la liste
- Remplacer la ligne
`j ← suivant(j)` par
`j ← suivant(iMin)`

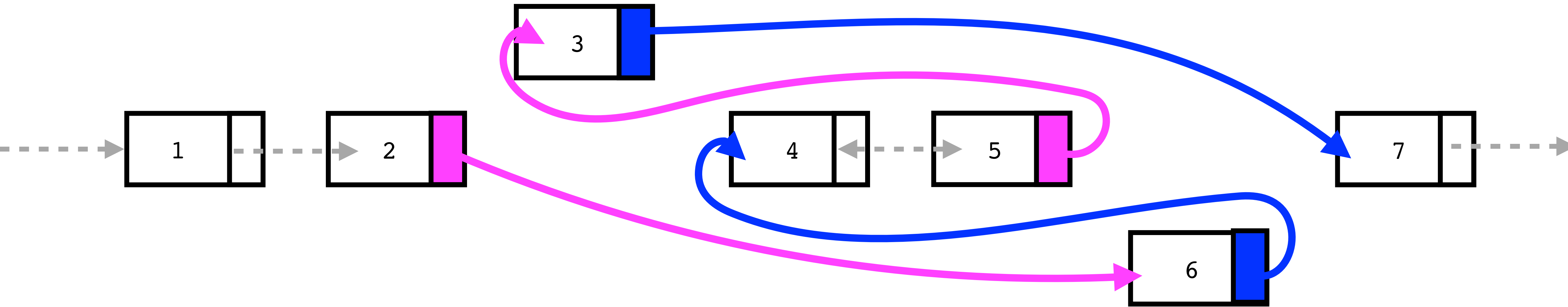
Et pour les listes simples ?



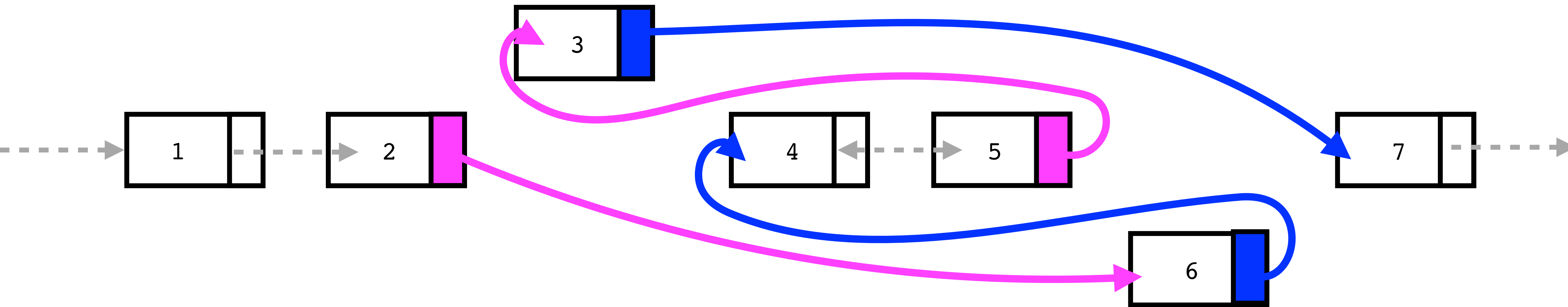
Et pour les listes simples ?



Et pour les listes simples ?

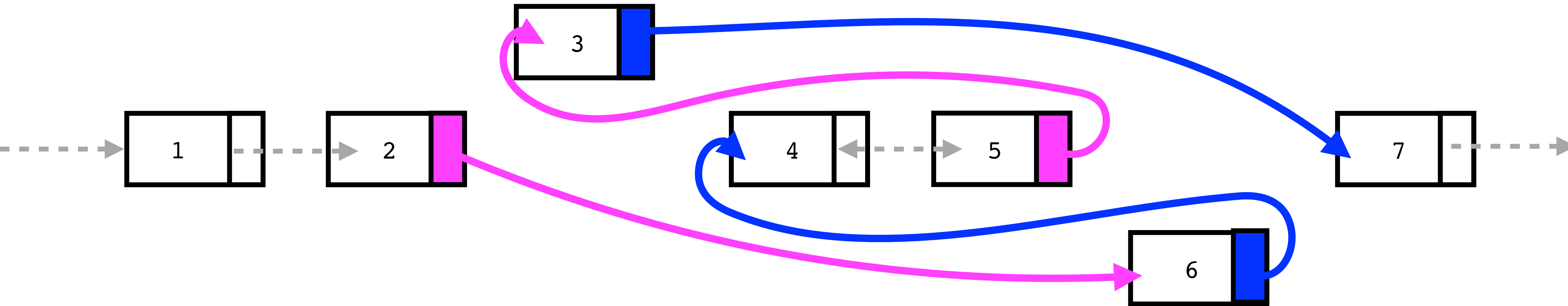


Et pour les listes simples ?



- 2 échanges suffisent
- Interface différent : fonction `échange_après`
- Est-ce que les itérateurs fournis à `échange_après` peuvent changer de position dans la liste ?

Et pour les listes simples ?



- 2 échanges suffisent
- Interface différent : fonction `échange_après`
- Est-ce que les itérateurs fournis à `échange_après` peuvent changer de position dans la liste ?
- Oui, si on échange deux éléments consécutifs ; sinon non.



10. Trier une liste par déplacement

Rappel : tri par insertion



```
fonction tri_par_insertion(T)
    N ← taille de T
    pour k de 1 à N-1
        tmp ← T[k]
        i ← k
        tant que i > 0 et tmp < T[i-1]
            T[i] ← T[i-1]
            i--
        T[i] ← tmp
```

- Algorithme pour un tableau T
- Parcours croissant : k
- Parcours décroissant : i
- Accès indexé : T[i]

Tri par insertion sur liste double



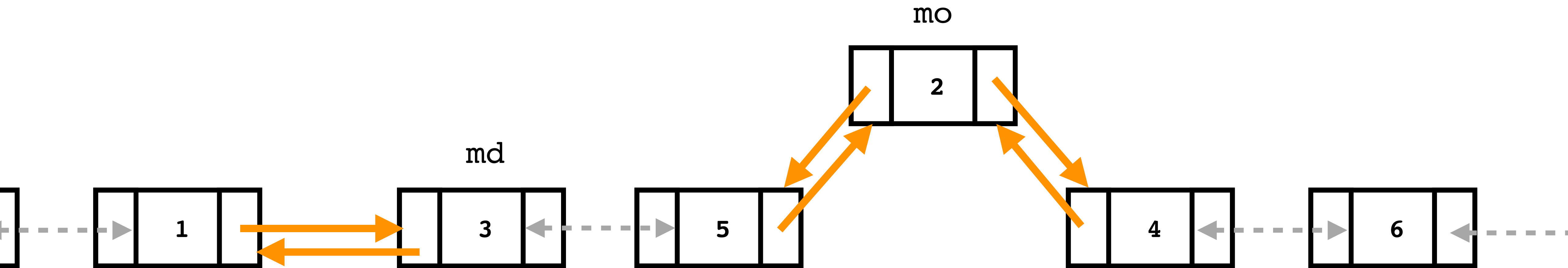
- Parcours croissant : itérateur k
- Parcours décroissant :
itérateurs i et pi au lieu de i
et i-1
- Accès par itérateurs *i et *pi
- La ligne *i ← *pi semble
inutile ; on pourrait déplacer le
maillon !

```
fonction tri_par_insertion(L) :  
    si L.empty()  
        retourner  
    k ← suivant(L.début())  
    tant que k != L.fin()  
        tmp ← *k  
        i ← k  
        pi ← précédent(i)  
        tant que i != L.début() et tmp < *pi  
            *i ← *pi  
            i ← pi  
            pi ← precedent(i)  
        *i ← tmp  
    k ← suivant(k)
```

Déplacer un maillon



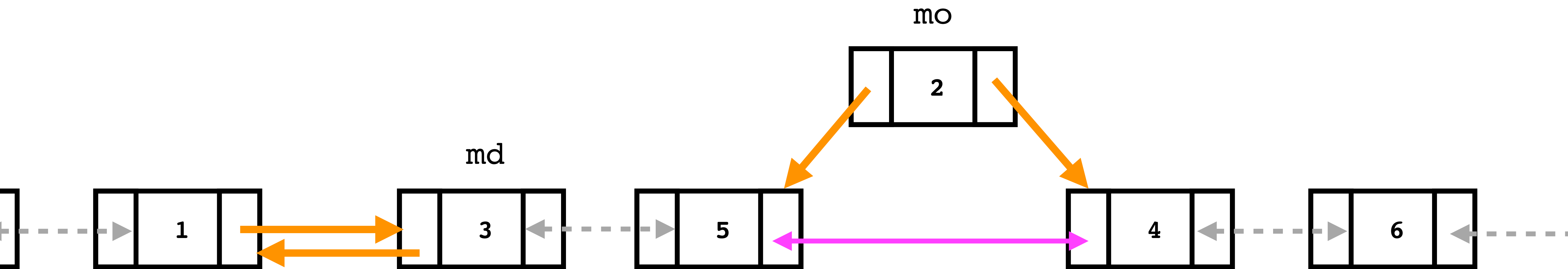
déplacer_maillon(*mo*, *md*)



Déplacer un maillon



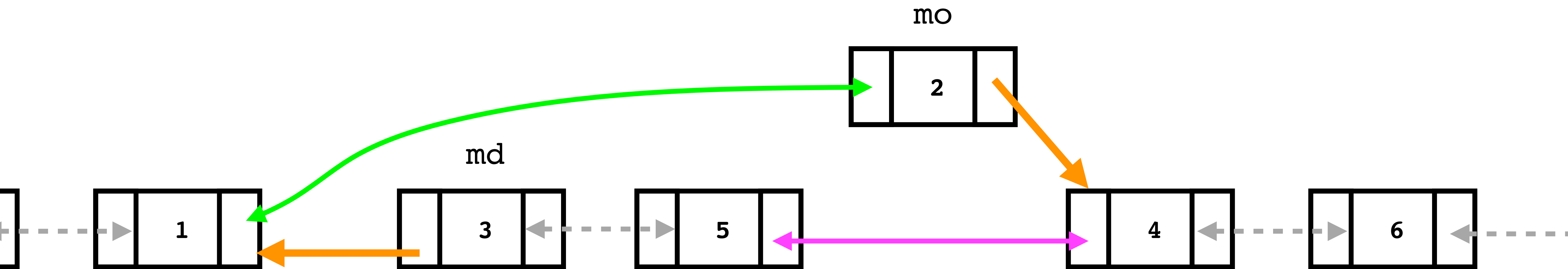
déplacer_maillon(*mo*, *md*)



Déplacer un maillon



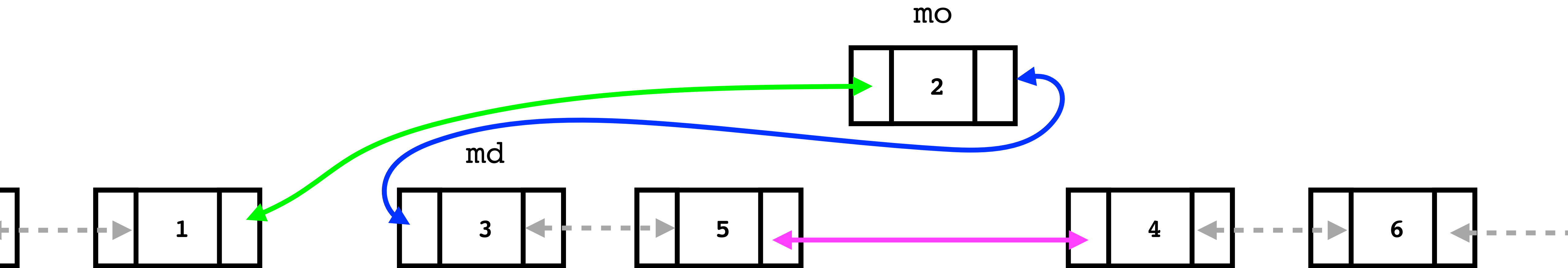
déplacer_maillon(*mo*, *md*)



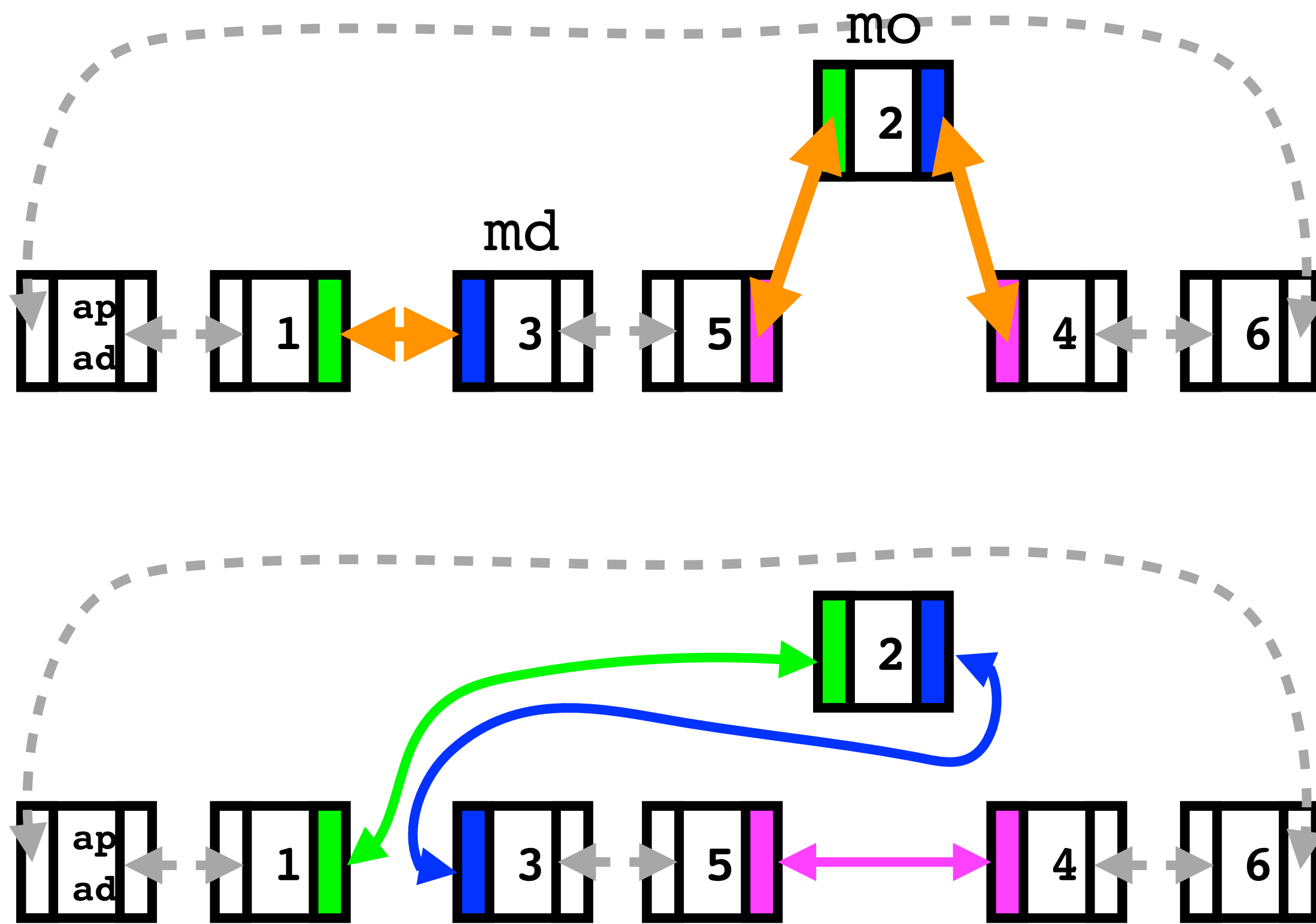
Déplacer un maillon



déplacer_maillon(*mo*, *md*)



Déplacer un maillon



```
fonction déplacer_maillon(mo, md) :
```

```
    si mo == md
```

```
        sortir
```

```
    mo.precedent.suivant ← mo.suivant
```

```
    mo.suivant.precedent ← mo.precedent
```

```
    md.precedent.suivant ← mo
```

```
    mo.precedent ← md.precedent
```

```
    md.precedent ← mo
```

```
    mo.suivant ← md
```

Tri par insertion sur liste double (2)



À vous de faire !

[Lien sur Cyberlearn](#)

Tri par insertion sur liste double (2)



- La boucle sur i ne déplace rien. Elle cherche l'emplacement d'insertion

```
fonction tri_par_insertion(L):  
    si L.empty()  
        sortir  
    k = suivant(L.début())  
    tant que k != L.fin()  
        i ← k  
        tant que i != L.début()  
            et *k < *(précédent(i))  
                i ← précédent(i)  
        t ← suivant(k)  
        déplacer_maillon(k, i)  
        k ← t
```

Tri par insertion sur liste double (2)



- La boucle sur i ne déplace rien. Elle cherche l'emplacement d'insertion
- utilise la fonction `déplacer_maillons`
- Attention, elle change la position de l'itérateur k

```
fonction tri_par_insertion(L) :  
    si L.empty()   
        sortir  
    k = suivant(L.début())  
    tant que k != L.fin()  
        i ← k  
        tant que i != L.début()   
            et *k < *(précédent(i))  
            i ← précédent(i)  
        t ← suivant(k)  
        déplacer_maillon(k, i)  
        k ← t
```

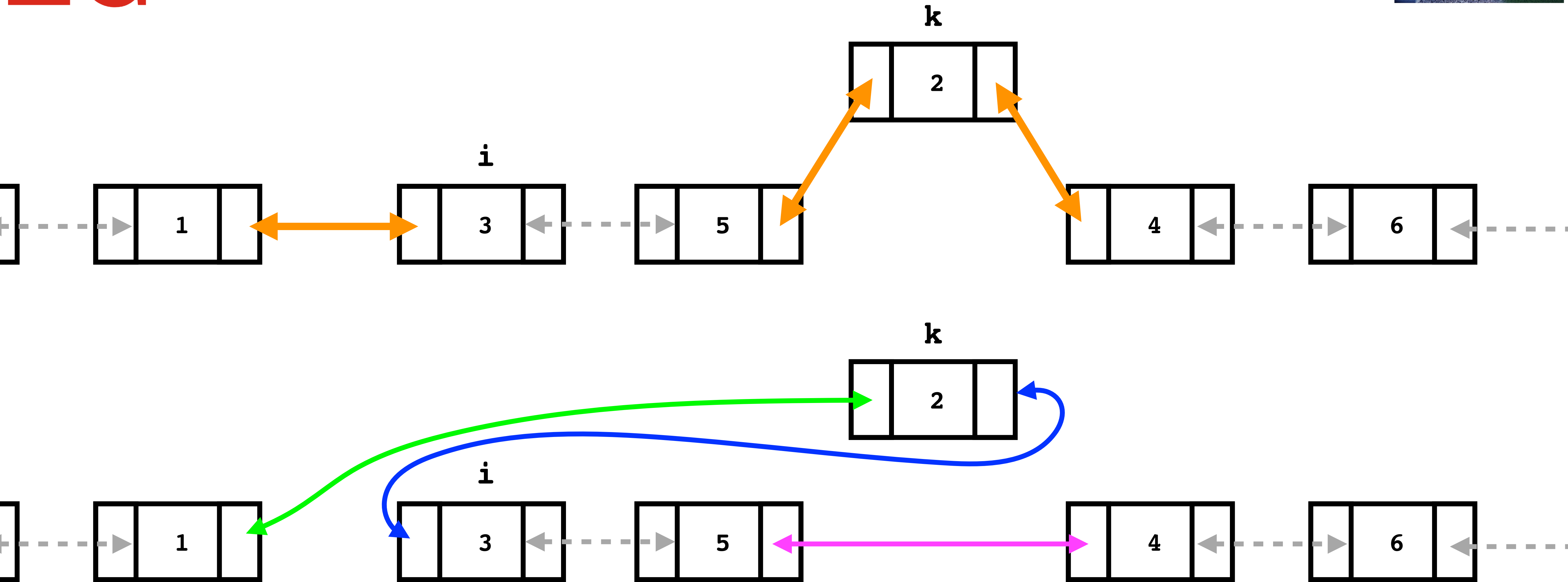
Tri par insertion sur liste double (2)



- La boucle sur i ne déplace rien. Elle cherche l'emplacement d'insertion
- utilise la fonction `déplacer_maillons`
- Attention, elle change la position de l'itérateur k
- Remplacer la ligne
 $k \leftarrow \text{suivant}(k)$ par
 $t \leftarrow \text{suivant}(k)$ avant déplacement, puis $k \leftarrow t$

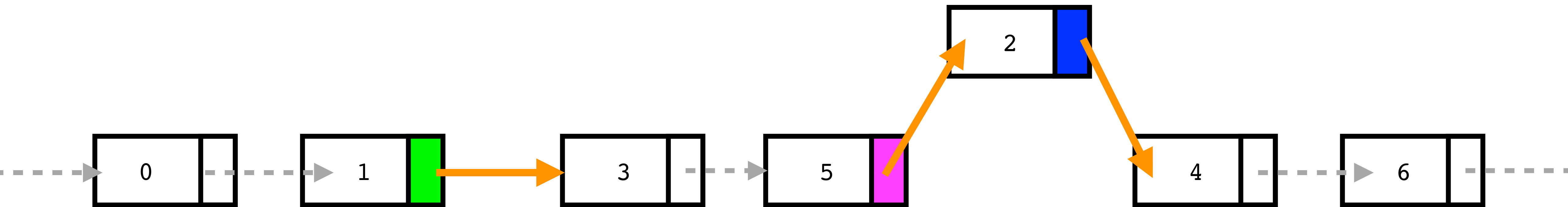
```
fonction tri_par_insertion(L) :  
    si L.empty()  
        sortir  
    k = suivant(L.début())  
    tant que k != L.fin()  
        i ← k  
        tant que i != L.début()  
            et *k < *(précédent(i))  
            i ← précédent(i)  
        t ← suivant(k)  
        déplacer_maillon(k, i)  
        k ← t
```

Déplacer un maillon



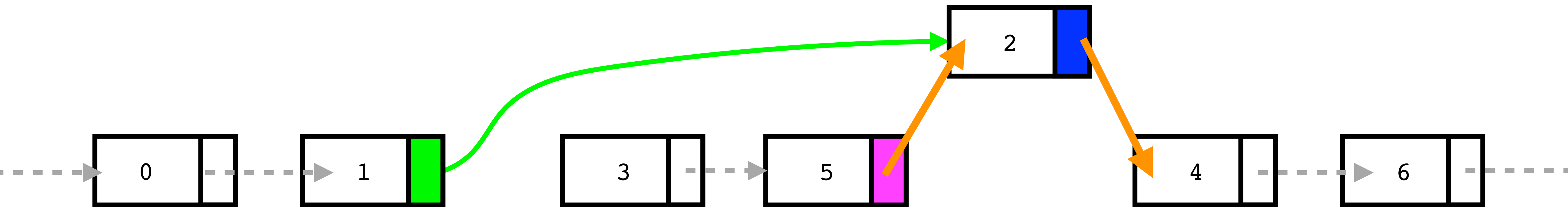
L'itérateur k a changé de position dans la liste.

Et pour une liste simple ?



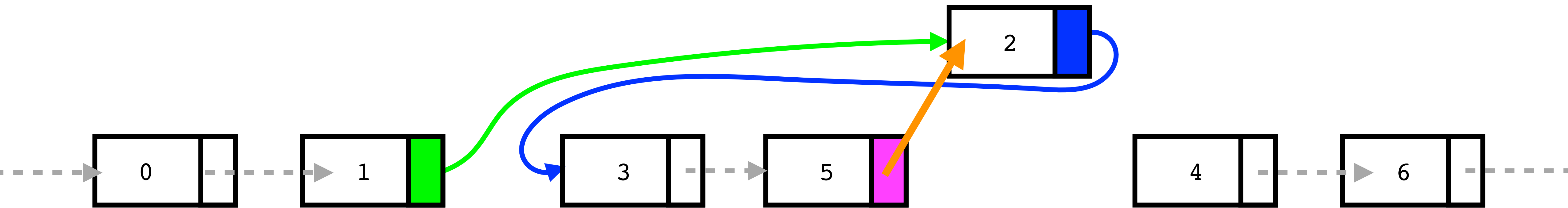
- Fonction `deplacer_après` qui prend les maillons précédents en paramètre

Et pour une liste simple ?



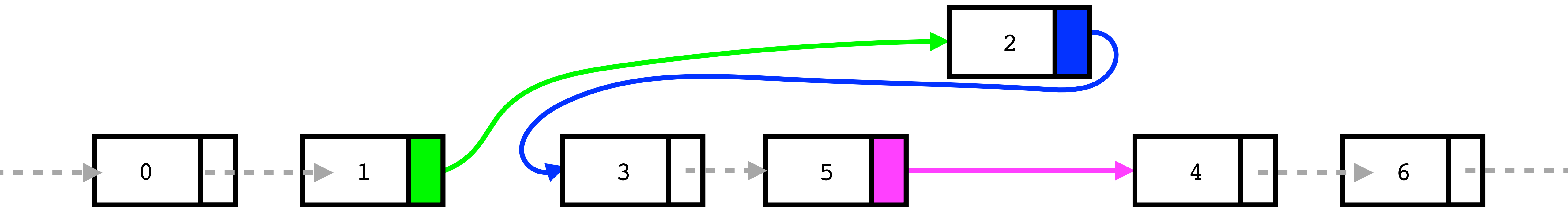
- Fonction `deplacer_après` qui prend les maillons précédents en paramètre

Et pour une liste simple ?



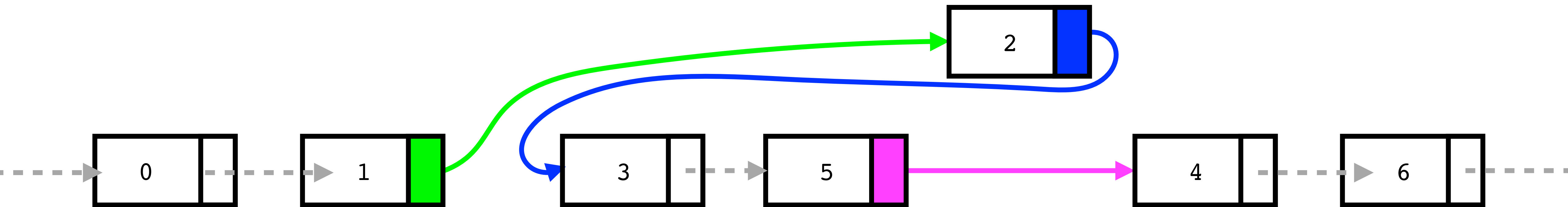
- Fonction `deplacer_après` qui prend les maillons précédents en paramètre

Et pour une liste simple ?



- Fonction `deplacer_après` qui prend les maillons précédents en paramètre

Et pour une liste simple ?



- Fonction `deplacer_après` qui prend les maillons précédents en paramètre
- 3 pointeurs à modifier
- Tri par insertion pas possible en l'absence de parcours inverse