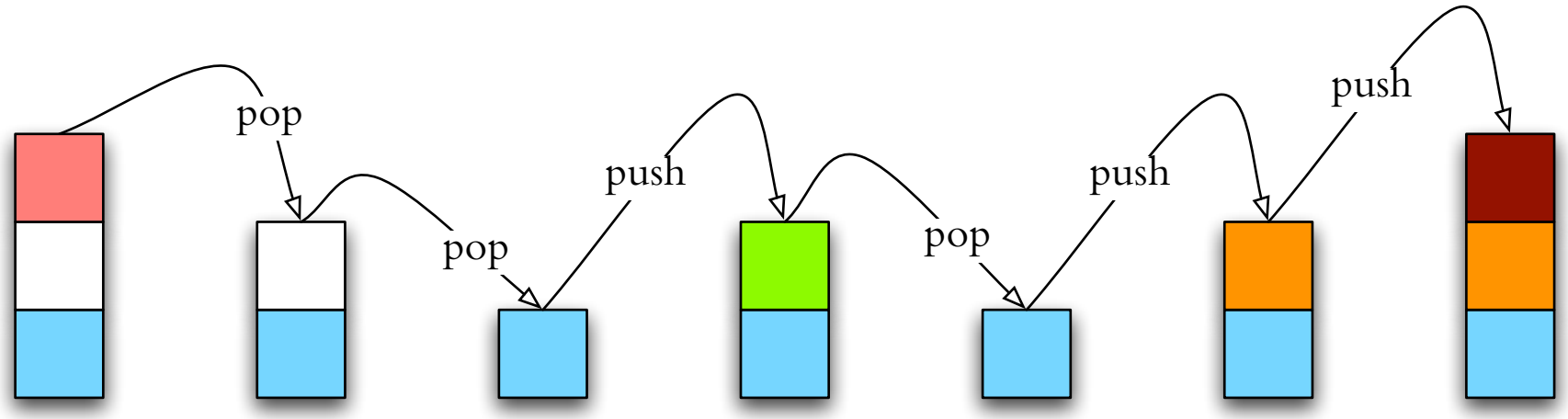
Pile — idée

Idée de pile : objets empilés un sur l'autre, on ne peut accéder qu'à l'élément supérieur

opérations : «empiler» (push) et «dépiler» (pop)



Pile

Objets : listes de style $A_1, A_2, \ldots, A_n$ et la liste nulle

Opérations :
$\mathsf{push}: P \times O \mapsto P$; p.e., $\mathsf{push}(P, x)$
$\mathsf{pop}: P \mapsto P \times O$; p.e., $\mathsf{pop}(P)$
$\mathsf{isEmpty}: P \mapsto \{\mathsf{vrai}, \mathsf{faux}\}$

En fait, `push` et `pop` sont implantés en modifiant leur argument de pile (au lieu de retourner une nouvelle pile)

Pile — implantation avec un tableau

Idée : on utilise un tableau avec un indice pour le sommet de la pile

(Un petit problème : la taille de la pile est bornée dans cette implantation)

```
public class Pile {
   private int sommet;
   private Object[] P;
   private static final int MAX_TAILLE = 100;

   public Pile(){
      P = new Object[MAX_TAILLE];
      sommet = 0;
   }
   public boolean isEmpty(){return (sommet==0);}
   ...
}
```

`sommet` indique où mettre le prochain objet de `push`

Pile — cont.

```
public void push(Object O){
    P[sommet] = O;
    sommet++;
}
```

Qu'est-ce qui se passe si on a trop d'éléments sur la pile ?
→ la pile déborde (*overflow*)

dans cette implantation : `ArrayIndexOutOfBoundsException`
c'est OK

Pile — cont.

```
public Object pop(){
    sommet --;
    return P[sommet];
}
```

Qu'est-ce qui se passe si on essaie de dépiler d'une pile vide ?
→ la pile déborde négativement (*underflow*)

dans cette implantation : `ArrayIndexOutOfBoundsException`
n'est pas trop élégant

Pile — cont.

Introduisons une exception spécifique à notre pile : `StackUnderflowException`

```
public class Pile {
   ...
   static class StackUnderflowException extends RuntimeException {
      private StackUnderflowException(String message){
         super(message);
      }
   }
   public Object pop(){
      if (sommet == 0)
         throw new StackUnderflowException("Rien ici.");
      sommet --;
      return P[sommet];
   }
}
```

(comme c'est une sous-classe de `RuntimeException`, on ne doit pas la déclarer par `throws`)

Pile — efficacité

Temps de calul par opération : $O(1)$

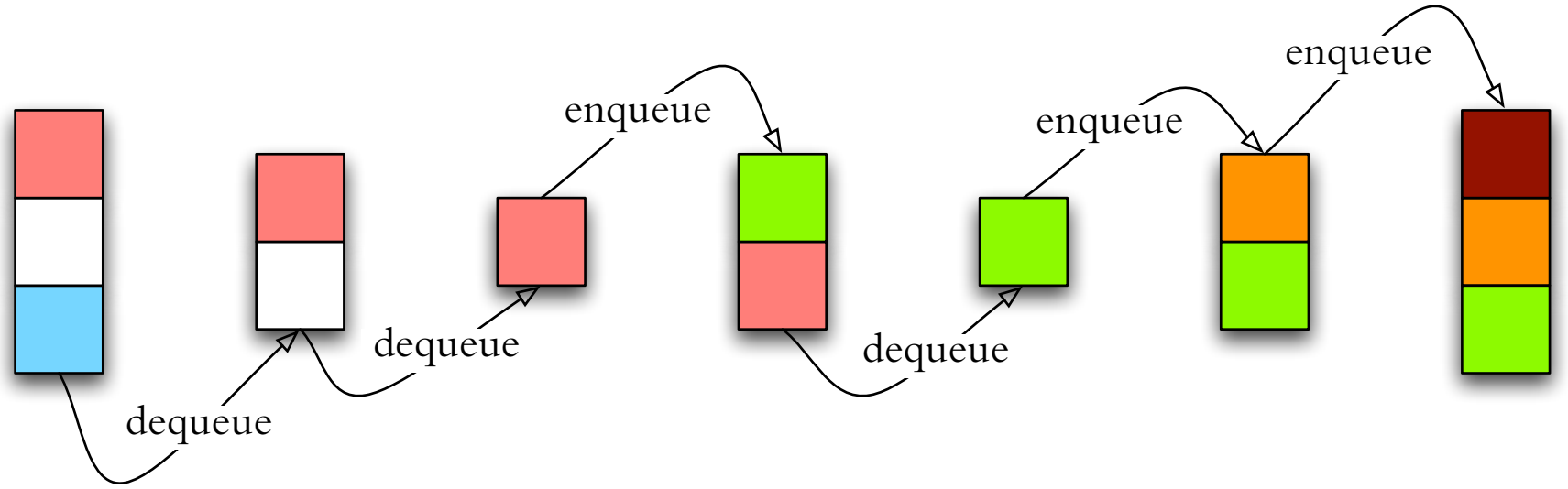
Accès à un élément quelconque : n'est pas possible directement, il faut utiliser une autre liste pour dépiler tous les éléments sur l'élément cherché ça prend $O(n)$ temps et une espace de travail $O(n)$ pour l'élément en position n

Queue — idée

(queue ou file, *queue* en anglais)

Idée de queue : comme une file d'attente, objets rangés un après l'autre, on peut enfiler à la fin ou défiler au début

opérations : «enfiler» (enqueue) et «défiler» (dequeue)



Queue

Objets : listes de style $A_1, A_2, \ldots, A_n$ et la liste nulle

Opérations :
$\mathsf{enqueue}: Q \times O \mapsto Q$; p.e., $\mathsf{enqueue}(Q, x)$
$\mathsf{dequeue}: Q \mapsto Q \times O$; p.e., $\mathsf{dequeue}(Q)$
$\mathsf{isEmpty}: Q \mapsto \{\mathsf{vrai}, \mathsf{faux}\}$

En fait, `enqueue` et `dequeue` sont implantés en modifiant leur argument de queue (au lieu de retourner une nouvelle queue)