Operações em Vectores

Sistematização

Operações em Vectores

Classe de exemplo

```
public class MyVector {
  private static final int DEFAULT_SIZE=???;
  private type [] v;
  private int count;

  public MyVector() {
    v = new type[DEFAULT_SIZE];
    count = 0;
  }
  public void insertLast(type e);
  public void removeLast();
  public void insertAt(type e, int pos);
  public void removeAt(int pos);
  public boolean searchForward(type e);
  public boolean searchBackward(type e);
  public void bubbleSort();
  public boolean binarySearch(type e);
}
```

Os elementos do vector são de um tipo genérico *type*.

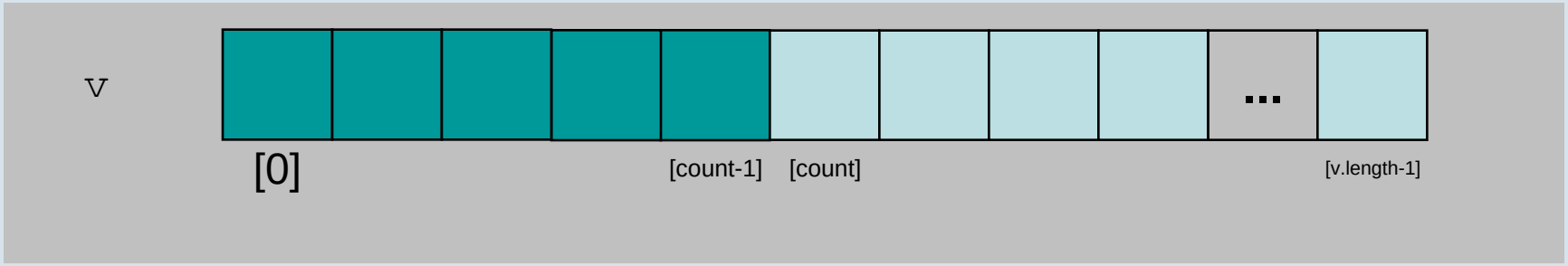
a variável `count` servirá para contar o número de elementos armazenados no vector `v`.

No construtor limitamo-nos a inicializar o vector `v` e o contador de elementos `count`.

Operações em Vectores
(Inserção no final)

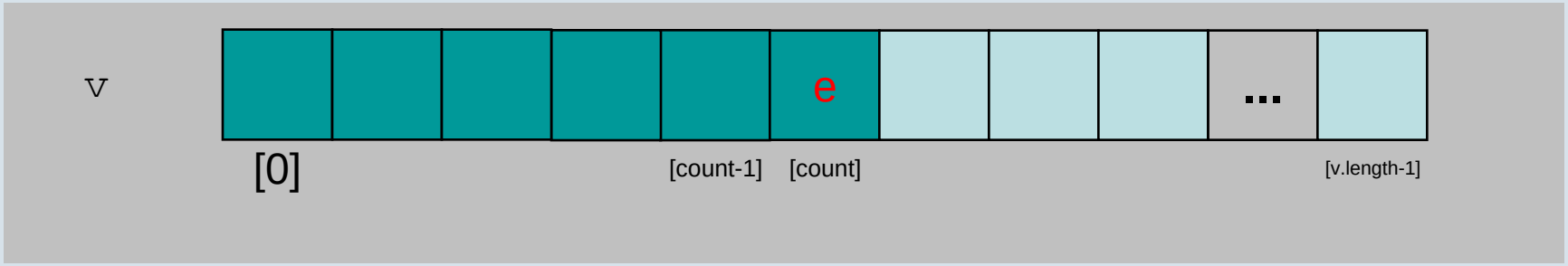
```
// pre: count < v.length
public void insertLast(type e) {
  v[count] = e;
  count++;
}
```

Como pré-condição para a execução da operação é necessário que o vector não se encontre todo preenchido



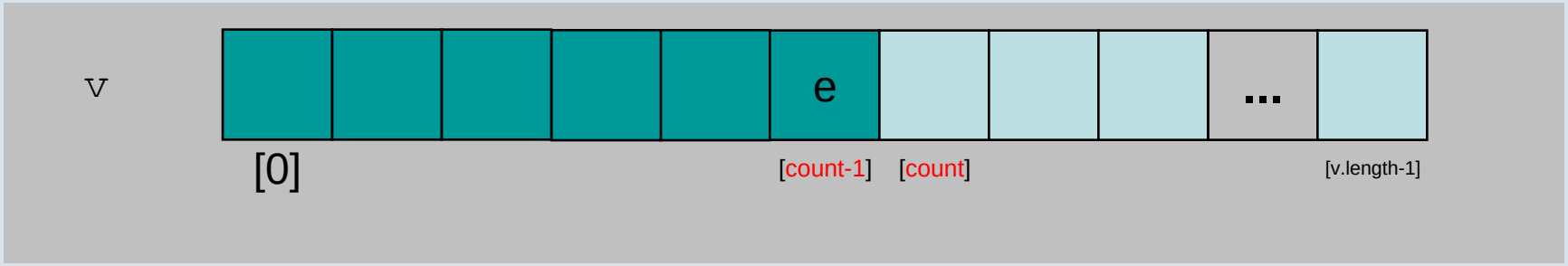
Operações em Vectores
(Inserção no final)

```
// pre: count < v.length
public void insertLast(type e) {
  v[count] = e;
  count++;
}
```



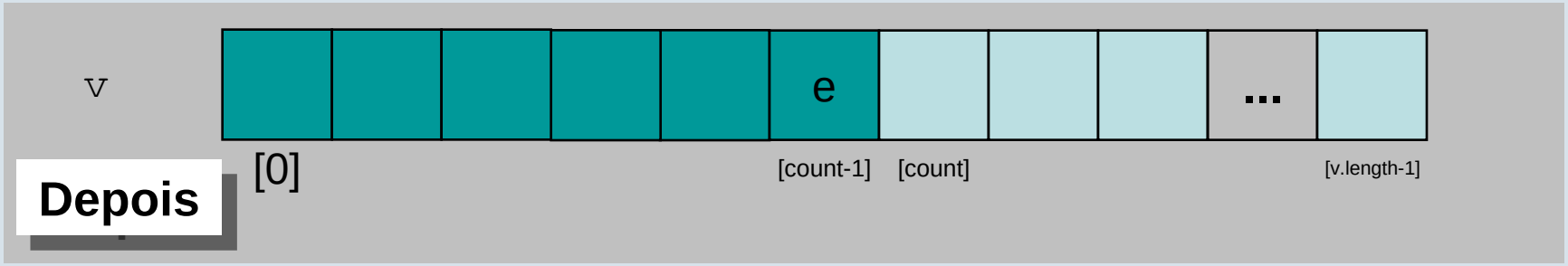
Operações em Vectores
(Inserção no final)

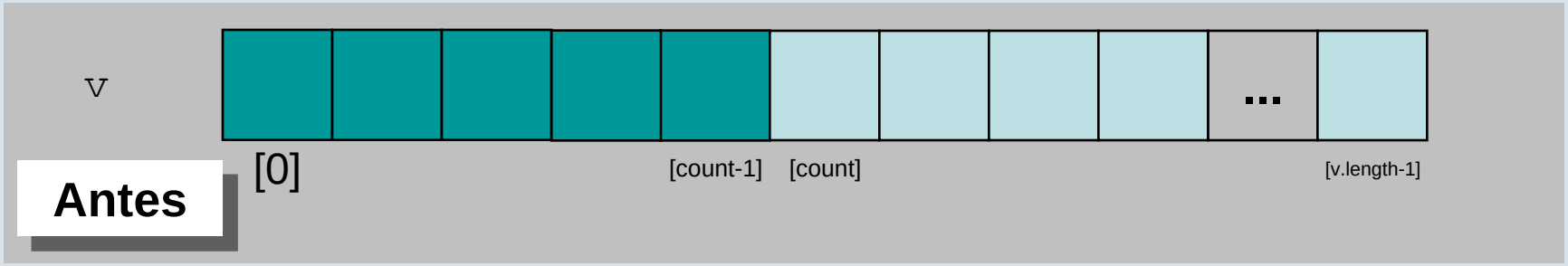
```
// pre: count < v.length
public void insertLast(type e) {
  v[count] = e;
  count++;
}
```



Operações em Vectores
(Inserção no final)

```
// pre: count < v.length
public void insertLast(type e) {
  v[count] = e;
  count++;
}
```

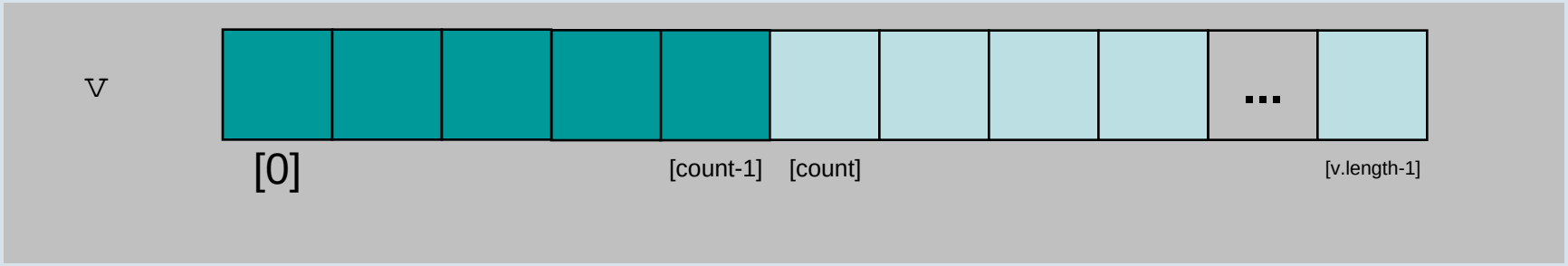




Operações em Vectores
(Remoção do final)

```
// pre: count > 0
public void removeLast() {
  count--;
}
```

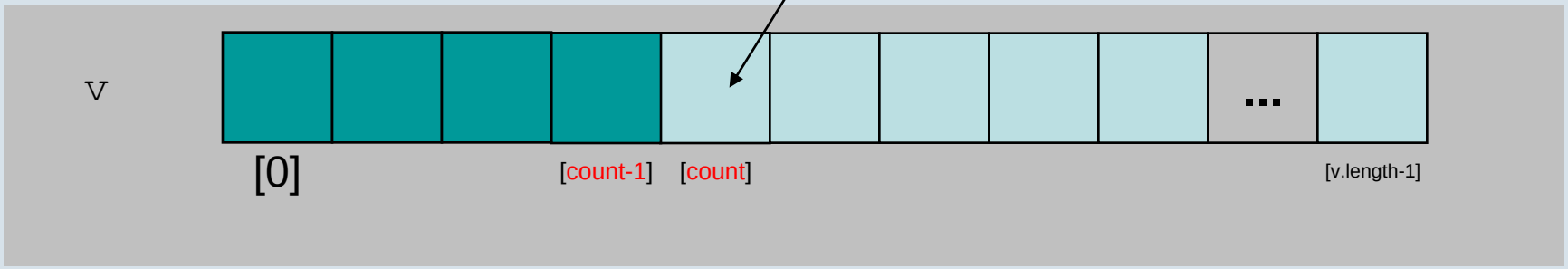
O vector tem que ter pelo menos um elemento...



Operações em Vectores
(Remoção do final)

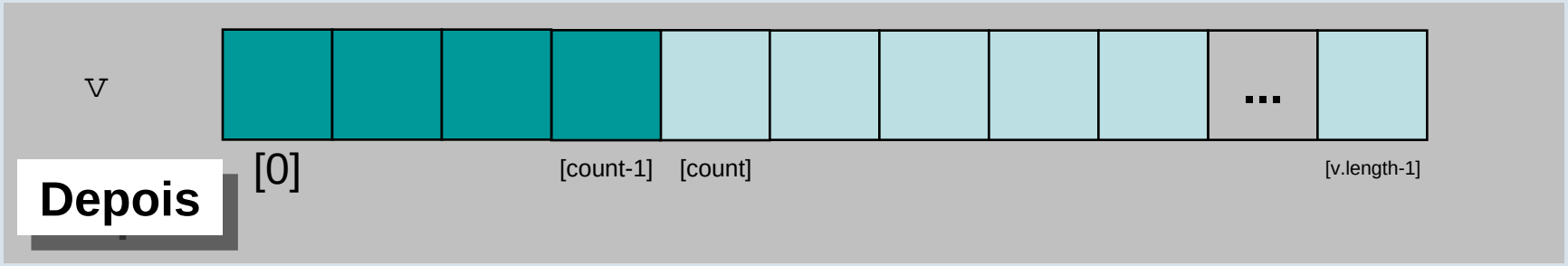
```
// pre: count > 0
public void removeLast() {
   count--;
}
```

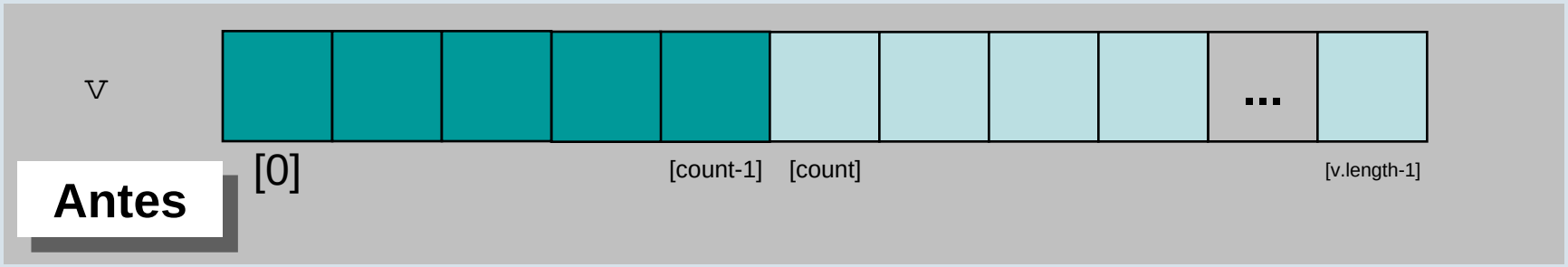
Basta decrementar o contador de elementos! O que antes era o último elemento permanece no vector mas é “esquecido” em futuras operações...



Operações em Vectores
(Remoção do final)

```
// pre: count > 0
public void removeLast() {
  count--;
}
```

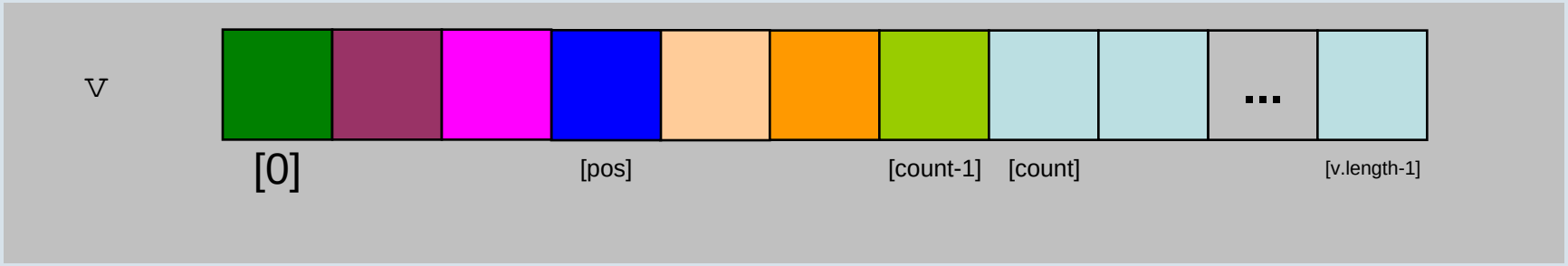




Operações em Vectores
(Inserção)

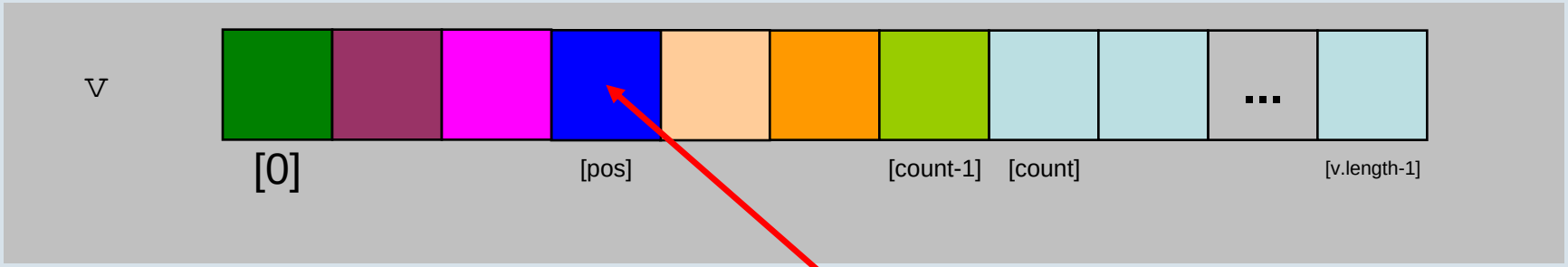
```
// pre: count < v.length && pos <= count
public void insertAt(type e, int pos) {
   for(int i=count-1; i >= pos; i--)
     v[i+1] = v[i];
   v[pos] = e;
   count++;
}
```

O vector não pode estar todo preenchido e a posição dada tem que estar preenchida ou ser a primeira vazia



Operações em Vectores
(Inserção)

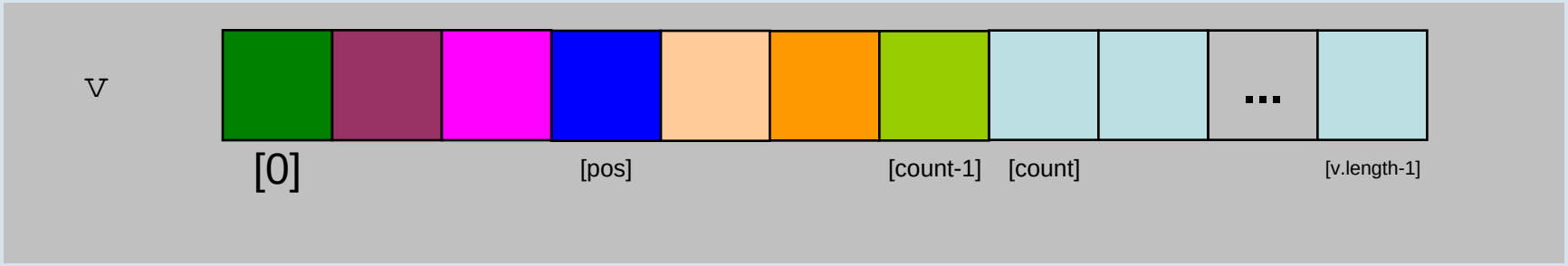
```
// pre: count < v.length
public void insertAt(type e, int pos) {
  for(int i=count-1; i >= pos; i--)
    v[i+1] = v[i];
  v[pos] = e;
  count++;
}
```



o elemento `e` deverá ser inserido na posição de índice `pos`.

Operações em Vectores
(Inserção)

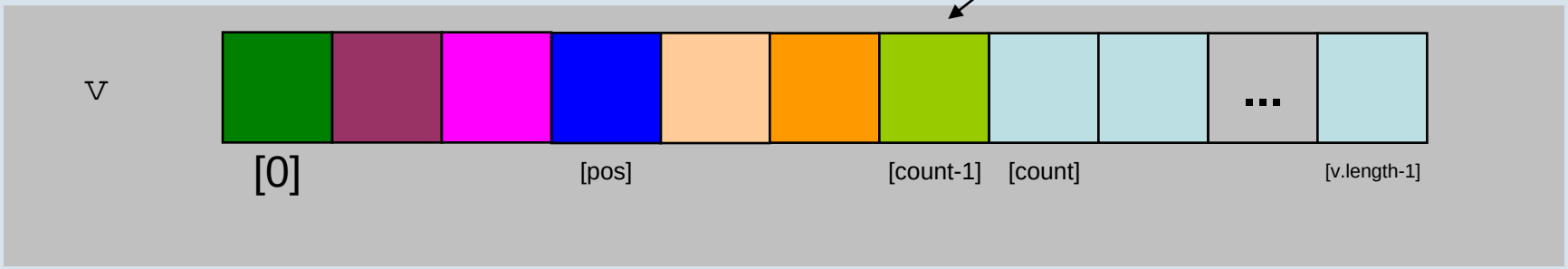
```
// pre: count < v.length
public void insertAt(type e, int pos) {
  for(int i=count-1; i >= pos; i--)
    v[i+1] = v[i];
  v[pos] = e;
  count++;
}
```



Mas antes é necessário abrir espaço para o lá colocar...

Operações em Vectores
(Inserção)

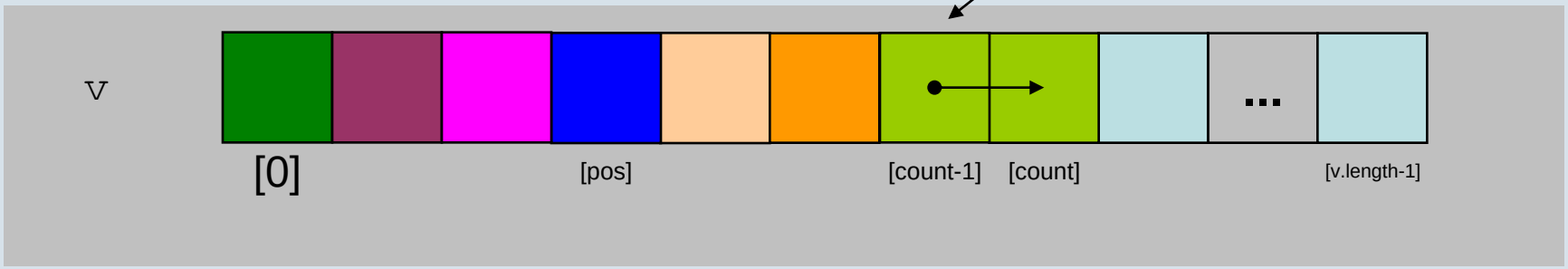
```
// pre: count < v.length
public void insertAt(type e, int pos) {
  for(int i=count-1; i >= pos; i--)
    v[i+1] = v[i];
  v[pos] = e;
  count++;
}
```



Mas antes é necessário abrir espaço para o lá colocar...

Operações em Vectores
(Inserção)

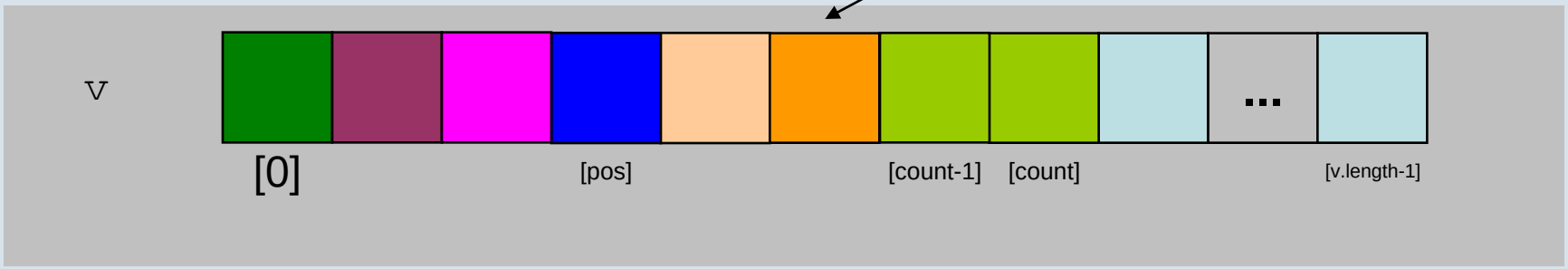
```
// pre: count < v.length
public void insertAt(type e, int pos) {
  for(int i=count-1; i >= pos; i--)
    v[i+1] = v[i];
  v[pos] = e;
  count++;
}
```



Mas antes é necessário abrir espaço para o lá colocar...

Operações em Vectores
(Inserção)

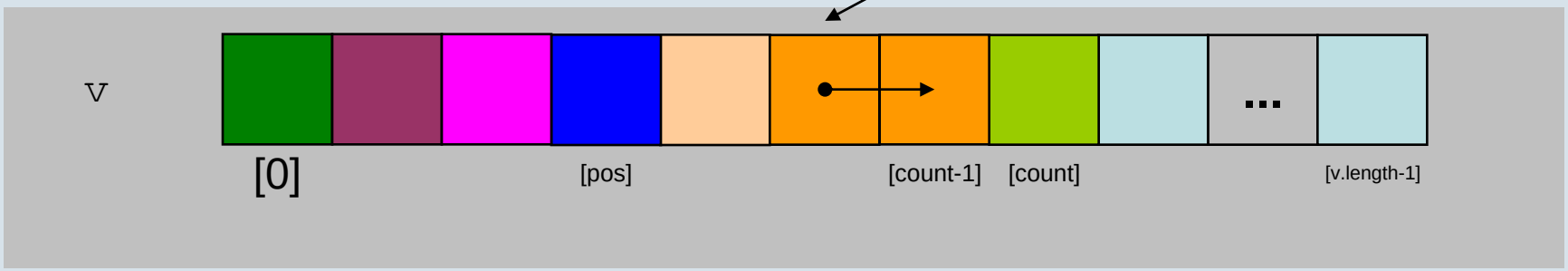
```
// pre: count < v.length
public void insertAt(type e, int pos) {
  for(int i=count-1; i >= pos; i--)
    v[i+1] = v[i];
  v[pos] = e;
  count++;
}
```



Mas antes é necessário abrir espaço para o lá colocar...

Operações em Vectores
(Inserção)

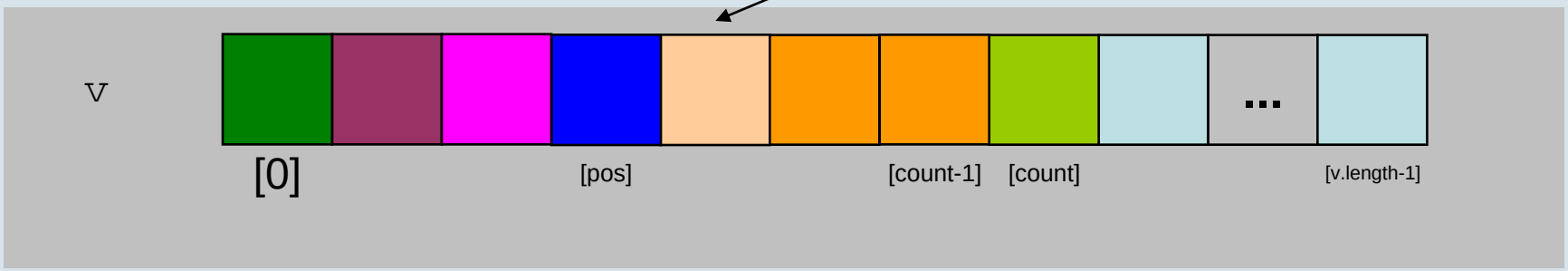
```
// pre: count < v.length
public void insertAt(type e, int pos) {
  for(int i=count-1; i >= pos; i--)
    v[i+1] = v[i];
  v[pos] = e;
  count++;
}
```



Mas antes é necessário abrir espaço para o lá colocar...

Operações em Vectores
(Inserção)

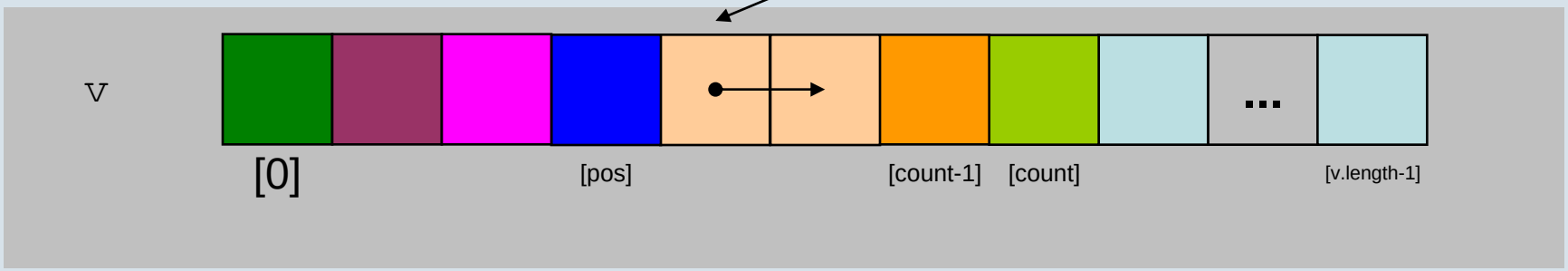
```
// pre: count < v.length
public void insertAt(type e, int pos) {
  for(int i=count-1; i >= pos; i--)
    v[i+1] = v[i];
  v[pos] = e;
  count++;
}
```



Mas antes é necessário abrir espaço para o lá colocar...

Operações em Vectores
(Inserção)

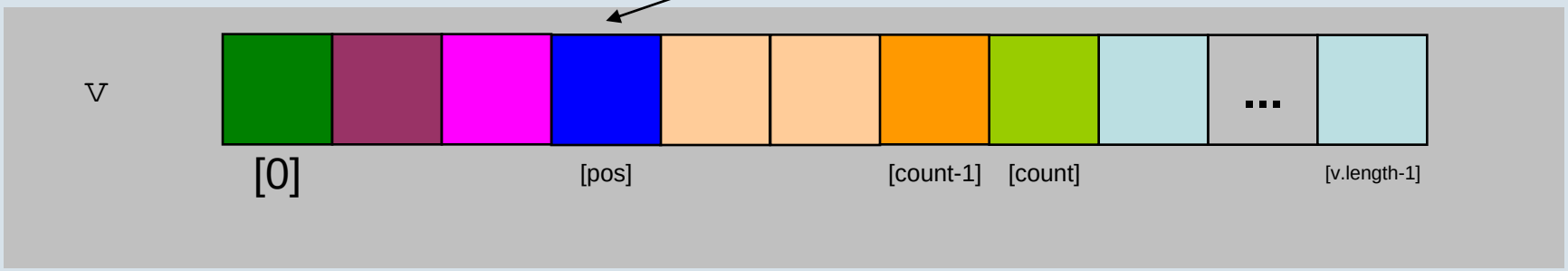
```
// pre: count < v.length
public void insertAt(type e, int pos) {
  for(int i=count-1; i >= pos; i--)
    v[i+1] = v[i];
  v[pos] = e;
  count++;
}
```



Mas antes é necessário abrir espaço para o lá colocar...

Operações em Vectores
(Inserção)

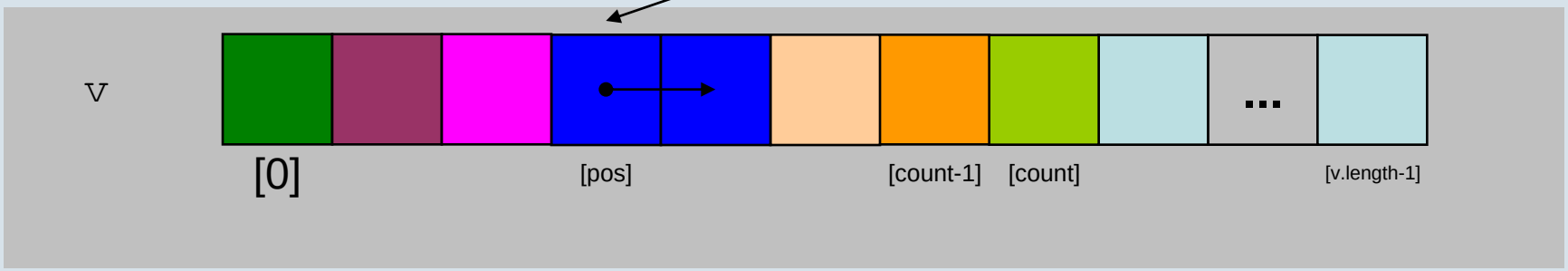
```
// pre: count < v.length
public void insertAt(type e, int pos) {
  for(int i=count-1; i >= pos; i--)
    v[i+1] = v[i];
  v[pos] = e;
  count++;
}
```



Mas antes é necessário abrir espaço para o lá colocar...

Operações em Vectores
(Inserção)

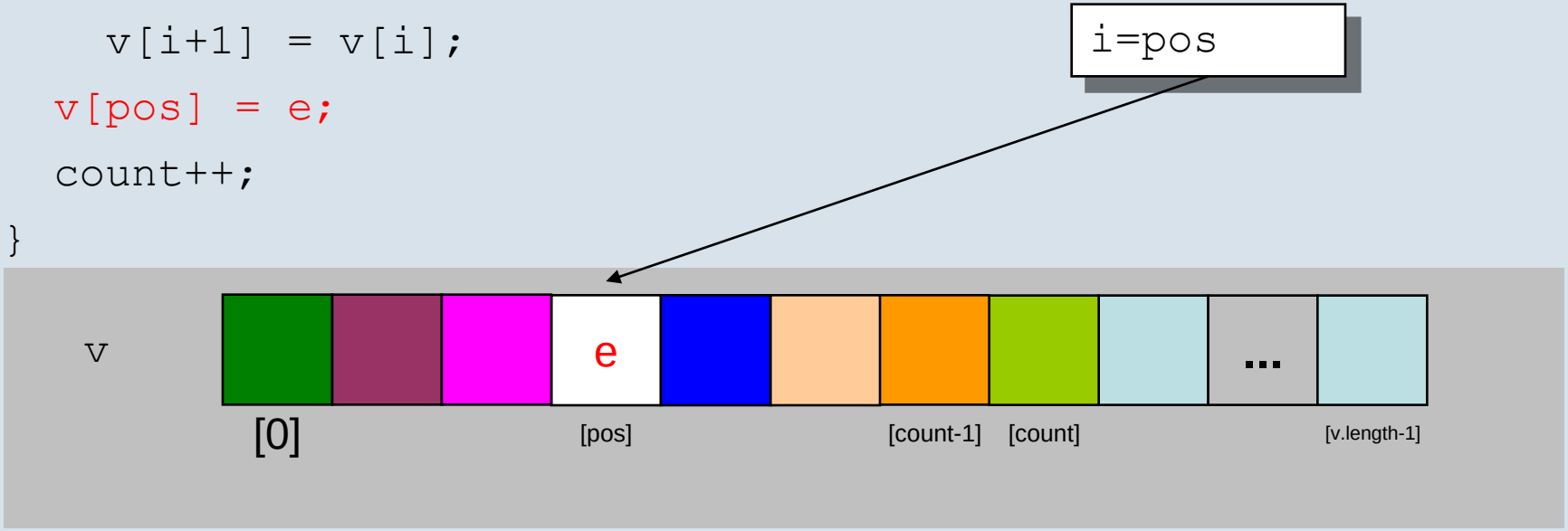
```
// pre: count < v.length
public void insertAt(type e, int pos) {
  for(int i=count-1; i >= pos; i--)
    v[i+1] = v[i];
  v[pos] = e;
  count++;
}
```



Mas antes é necessário abrir espaço para o lá colocar...

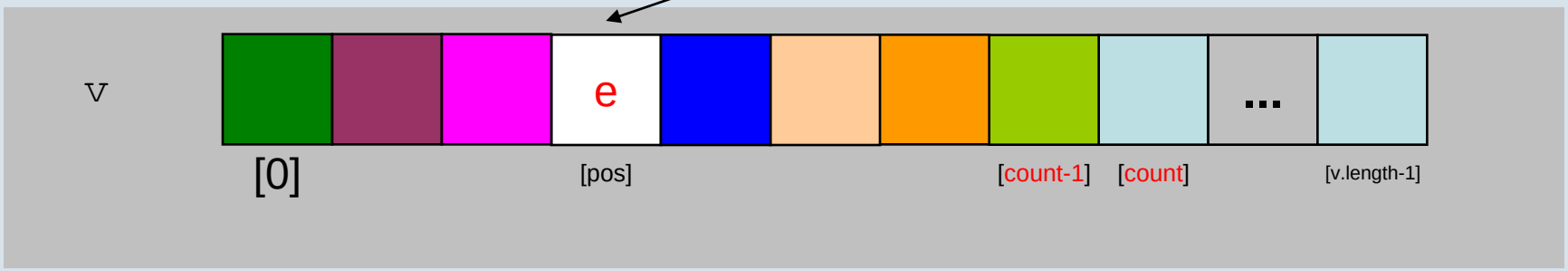
Operações em Vectores
(Inserção)

```
// pre: count < v.length
public void insertAt(type e, int pos) {
  for(int i=count-1; i >= pos; i--)
    v[i+1] = v[i];
  v[pos] = e;
  count++;
}
```



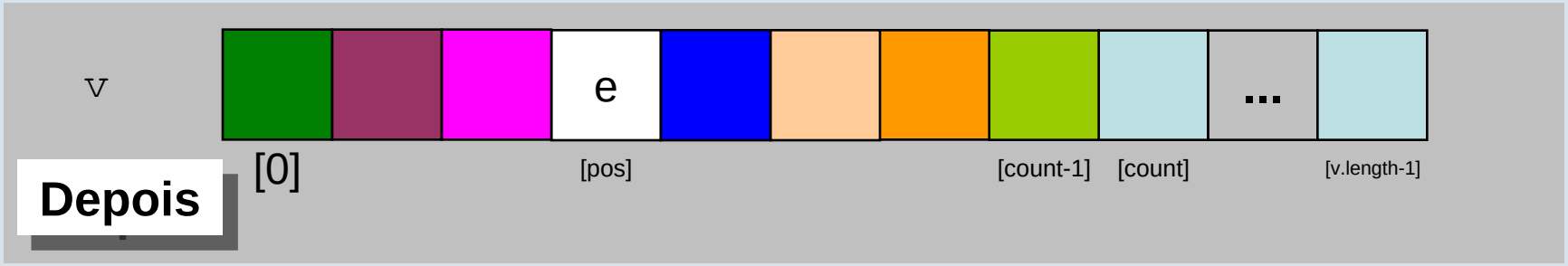
Operações em Vectores
(Inserção)

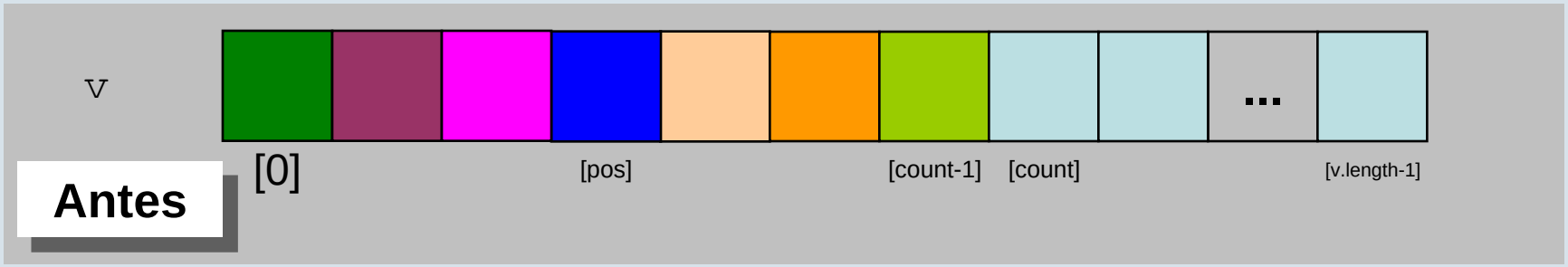
```
// pre: count < v.length
public void insertAt(type e, int pos) {
   for(int i=count-1; i >= pos; i--)
     v[i+1] = v[i];
   v[pos] = e;
   count++;
}
```



Operações em Vectores
(Inserção)

```
// pre: count < v.length
public void insertAt(type e, int pos) {
  for(int i=count-1; i >= pos; i--)
    v[i+1] = v[i];
  v[pos] = e;
  count++;
}
```

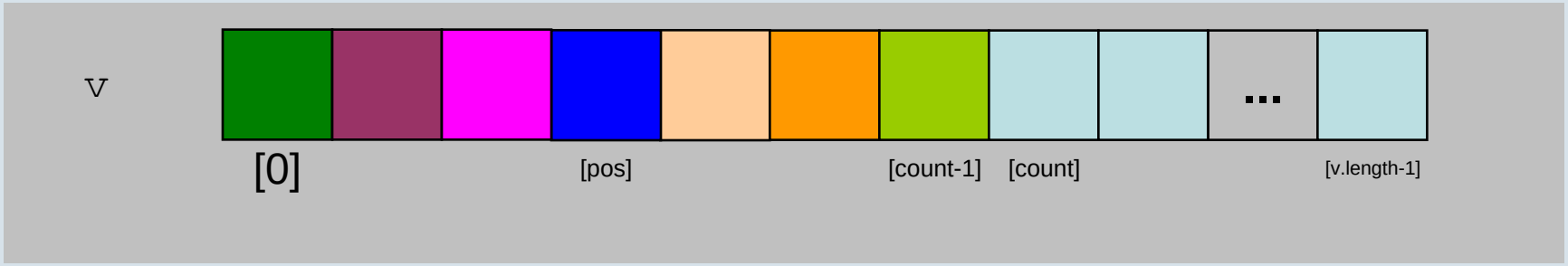




Operações em Vectores
(Remoção)

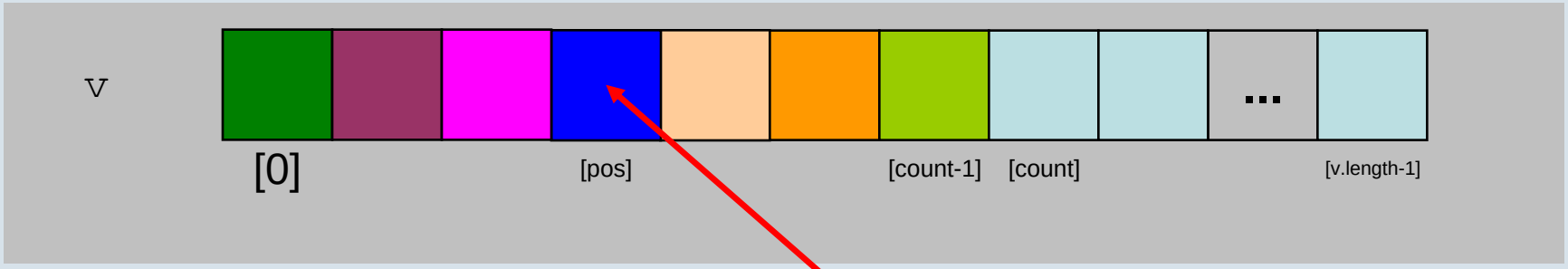
```
pre: count > 0 && pos < count
public void removeAt(int pos) {
   for(int i=pos; i<count-1; i++)
     v[i] = v[i+1];
   count--;
}
```

O vector tem que ter pelo menos um elemento e a posição indicada tem que estar preenchida



Operações em Vectores
(Remoção)

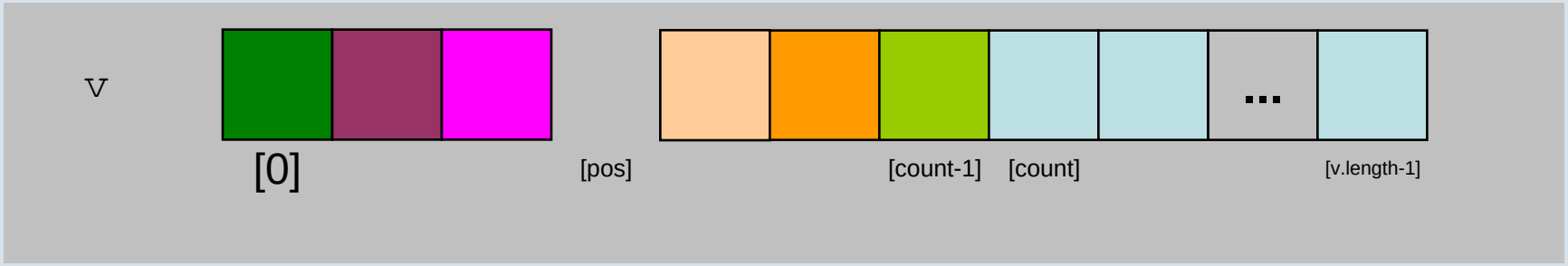
```
pre: count > 0 && pos < count
public void removeAt(int pos) {
  for(int i=pos; i<count-1; i++)
    v[i] = v[i+1];
  count--;
}
```



o elemento deverá ser removido da posição de índice `pos`.

Operações em Vectores
(Remoção)

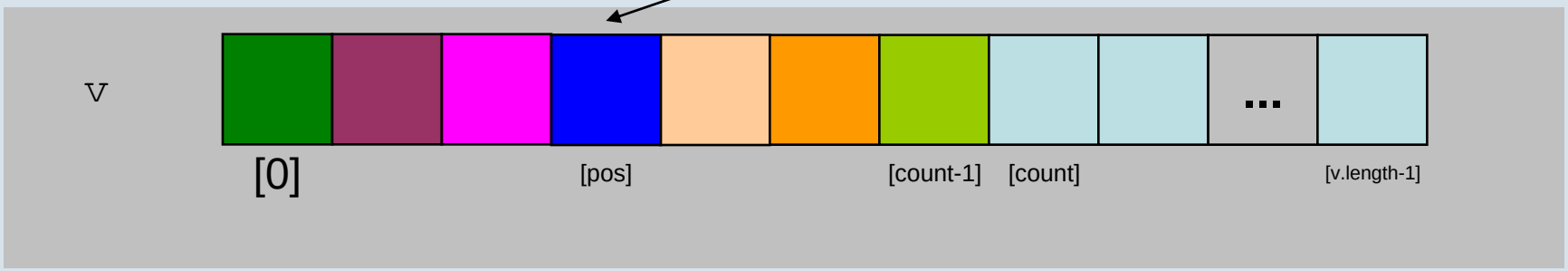
```
pre: count > 0 && pos < count
public void removeAt(int pos) {
  for(int i=pos; i<count-1; i++)
    v[i] = v[i+1];
  count--;
}
```



Mas não podemos deixar “um buraco” no vector...

Operações em Vectores
(Remoção)

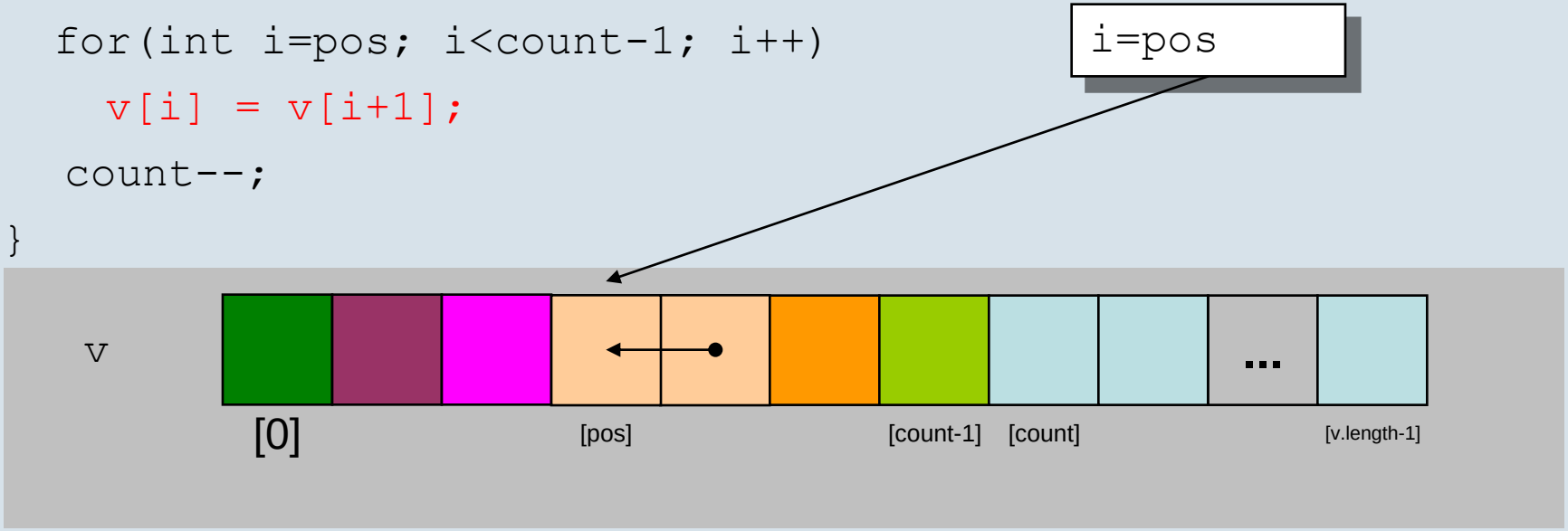
```
pre: count > 0 && pos < count
public void removeAt(int pos) {
   for(int i=pos; i<count-1; i++)
     v[i] = v[i+1];
   count--;
}
```



Mas não podemos deixar “um buraco” no vector...

Operações em Vectores
(Remoção)

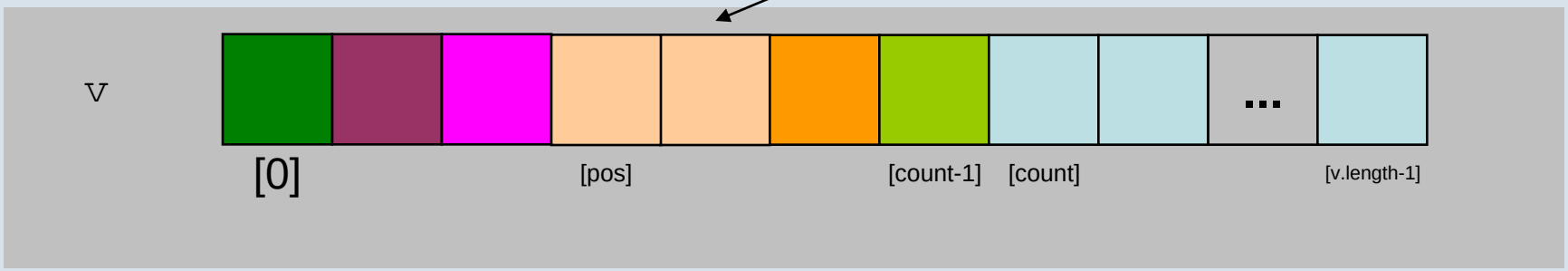
```
pre: count > 0 && pos < count
public void removeAt(int pos) {
  for(int i=pos; i<count-1; i++)
    v[i] = v[i+1];
  count--;
}
```



Mas não podemos deixar “um buraco” no vector...

Operações em Vectores
(Remoção)

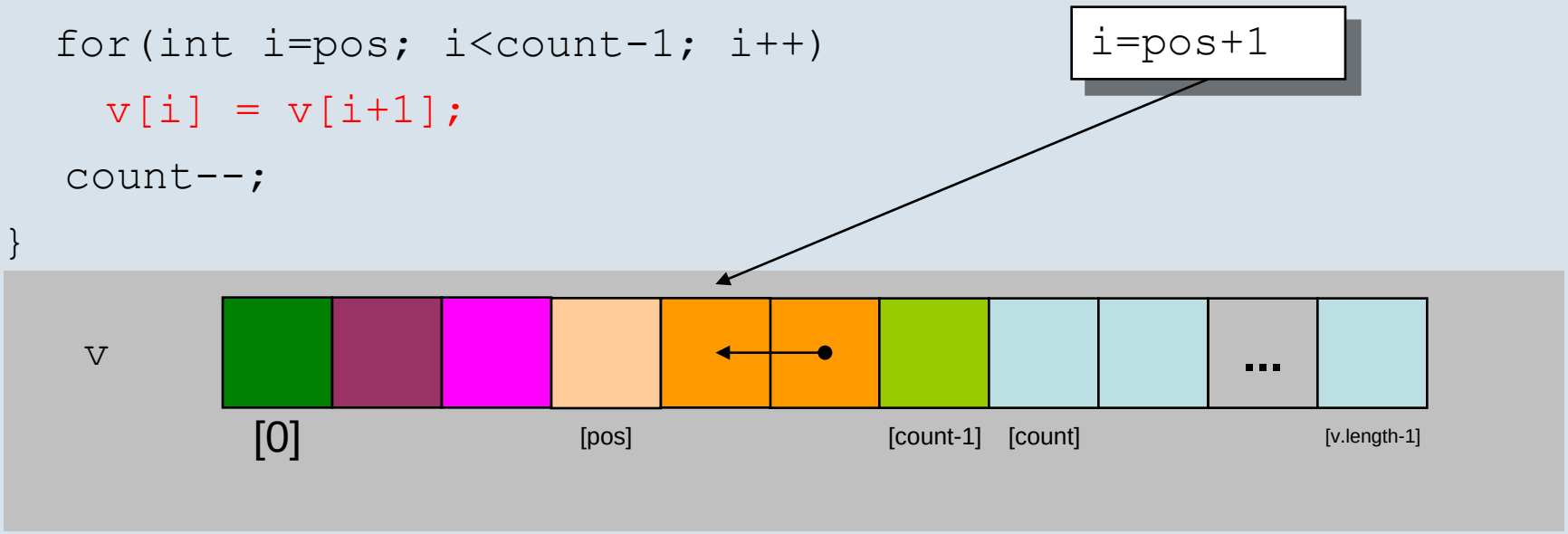
```
pre: count > 0 && pos < count
public void removeAt(int pos) {
  for(int i=pos; i<count-1; i++)
    v[i] = v[i+1];
  count--;
}
```



Mas não podemos deixar “um buraco” no vector...

Operações em Vectores
(Remoção)

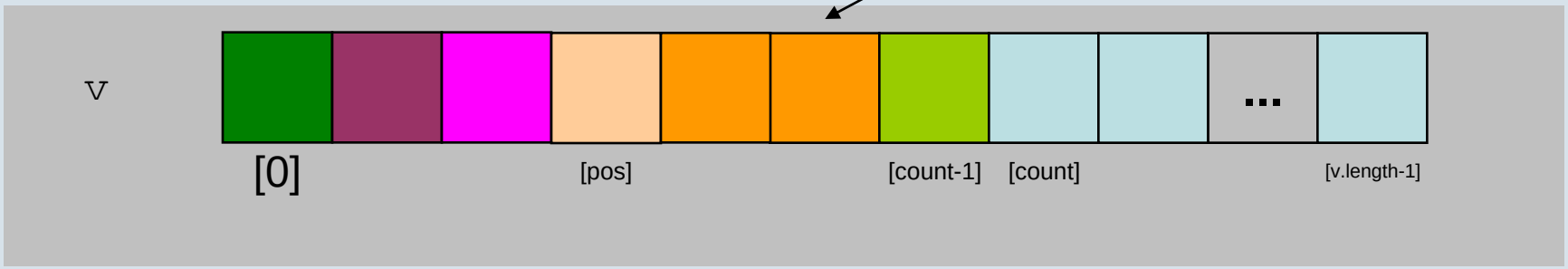
```
pre: count > 0 && pos < count
public void removeAt(int pos) {
  for(int i=pos; i<count-1; i++)
    v[i] = v[i+1];
  count--;
}
```



Mas não podemos deixar “um buraco” no vector...

Operações em Vectores
(Remoção)

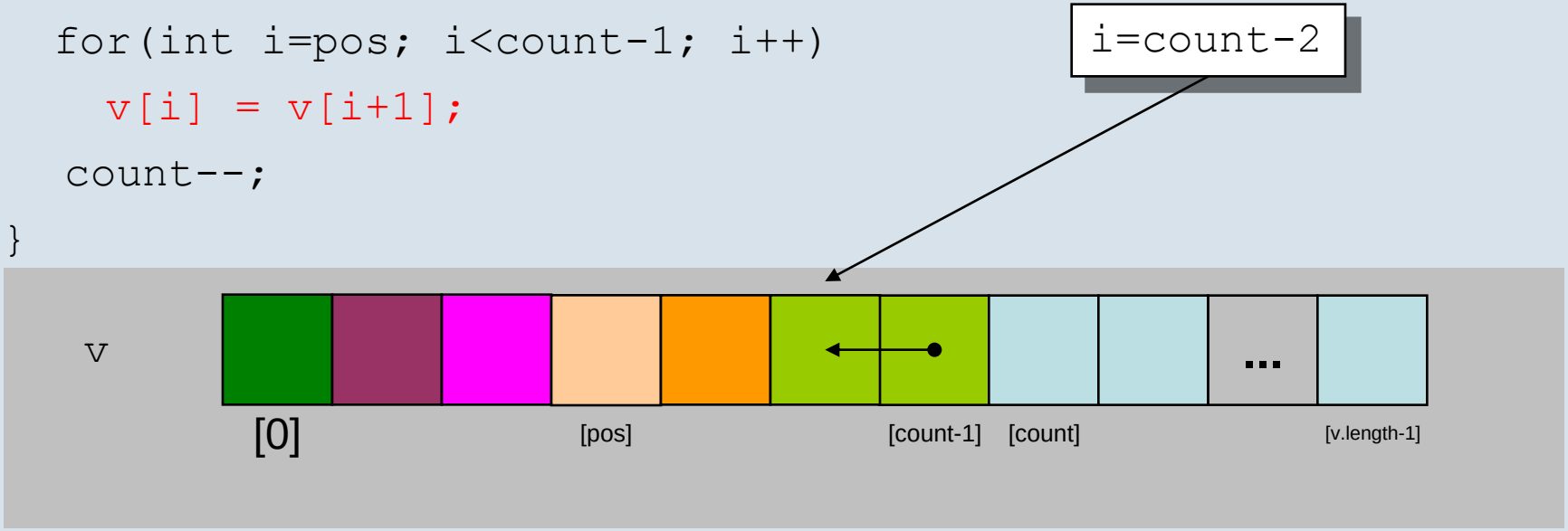
```
pre: count > 0 && pos < count
public void removeAt(int pos) {
  for(int i=pos; i<count-1; i++)
    v[i] = v[i+1];
  count--;
}
```



Mas não podemos deixar “um buraco” no vector...

Operações em Vectores
(Remoção)

```
pre: count > 0 && pos < count
public void removeAt(int pos) {
  for(int i=pos; i<count-1; i++)
    v[i] = v[i+1];
  count--;
}
```

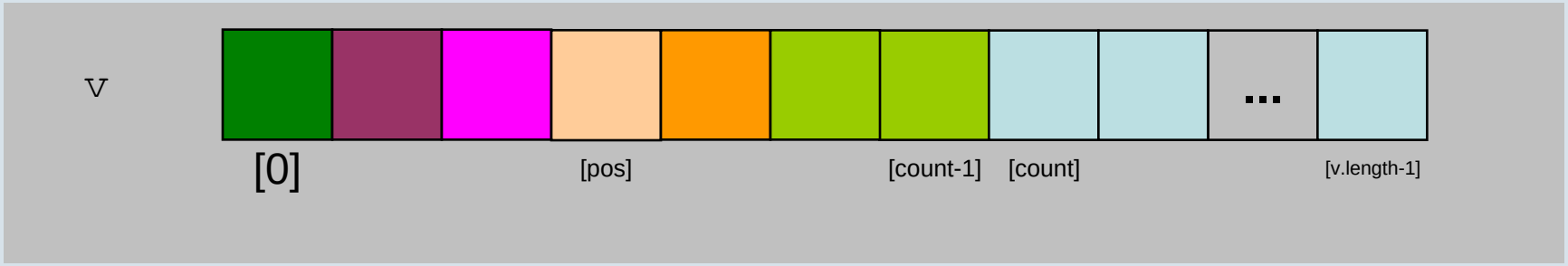


Mas não podemos deixar “um buraco” no vector...

Operações em Vectores
(Remoção)

```
pre: count > 0 && pos < count
public void removeAt(int pos) {
  for(int i=pos; i<count-1; i++)
    v[i] = v[i+1];
  count--;
}
```

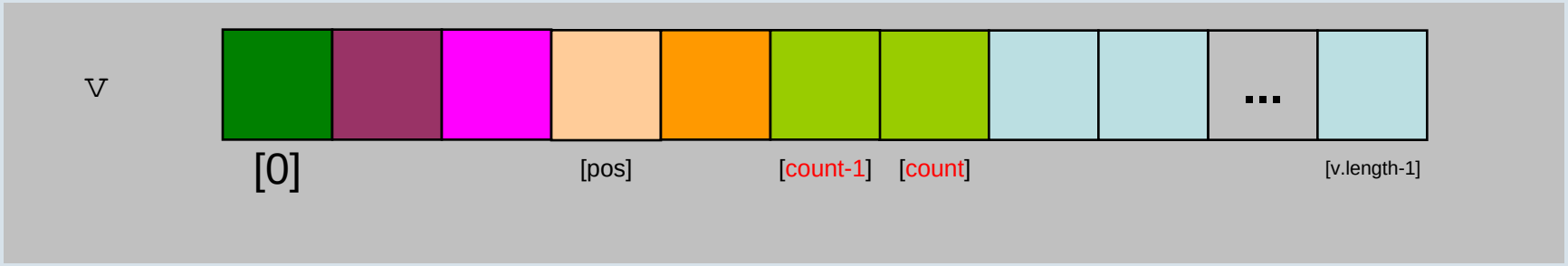
i=count-1



Mas não podemos deixar “um buraco” no vector...

Operações em Vectores
(Remoção)

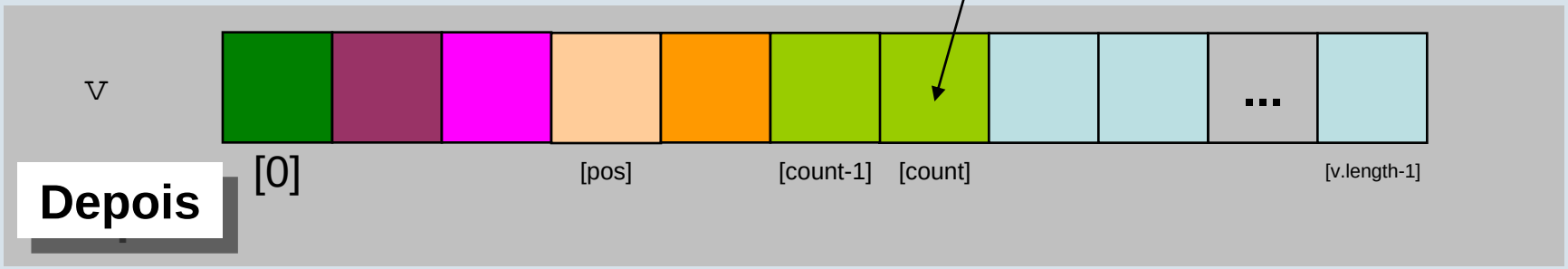
```
pre: count > 0 && pos < count
public void removeAt(int pos) {
  for(int i=pos; i<count-1; i++)
    v[i] = v[i+1];
  count--;
}
```

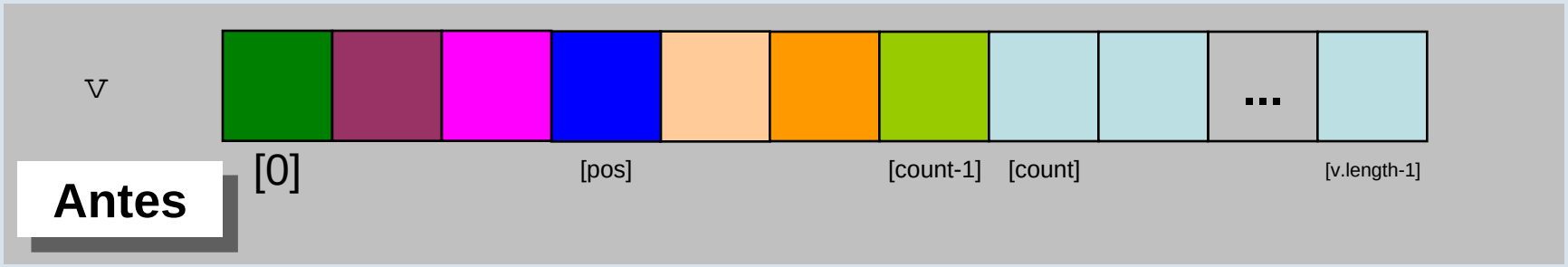


Operações em Vectores
(Remoção)

```
pre: count > 0 && pos < count
public void removeAt(int pos) {
  for(int i=pos; i<count-1; i++)
    v[i] = v[i+1];
  count--;
}
```

Apesar do último elemento estar repetido, nunca será utilizado pois não está contabilizado...

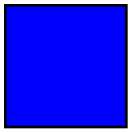


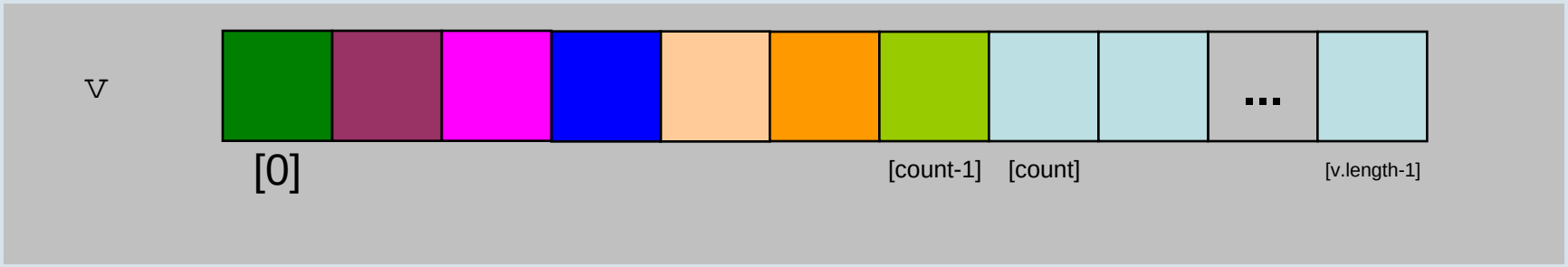


Operações em Vectores
(Pesquisa Linear – índices crescentes)

```
public boolean searchForward(type e) {
  boolean found = false;
  int i=0;
  while(i<count && !found)
    if(v[i] == e)
      found = true;
    else i++;
  return found;
}
```

Existirá no vector o elemento

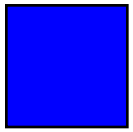
e =  ?



Operações em Vectores
(Pesquisa Linear – índices crescentes)

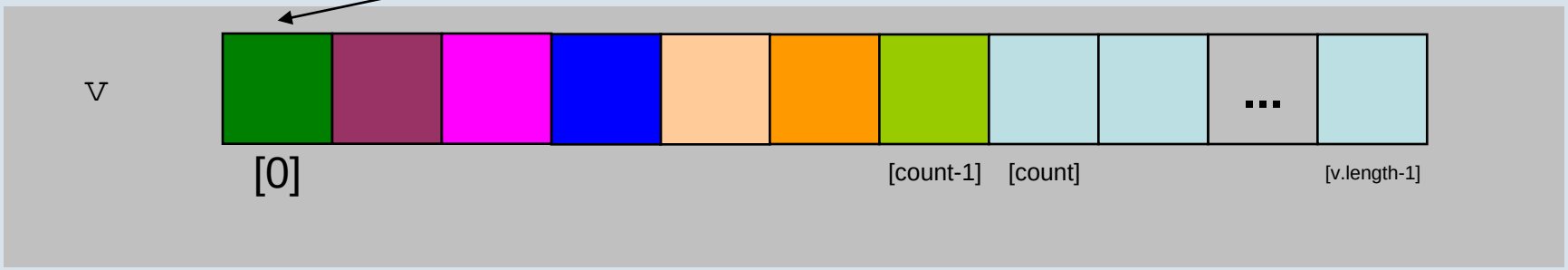
```
public boolean searchForward(type e) {
  boolean found = false;
  int i=0;
  while(i<count && !found)
    if(v[i] == e)
      found = true;
    else i++;
  return found;
}
```

Existirá no vector o elemento

e =  ?

```
found = false
```

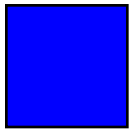
```
i=0
```



Operações em Vectores
(Pesquisa Linear – índices crescentes)

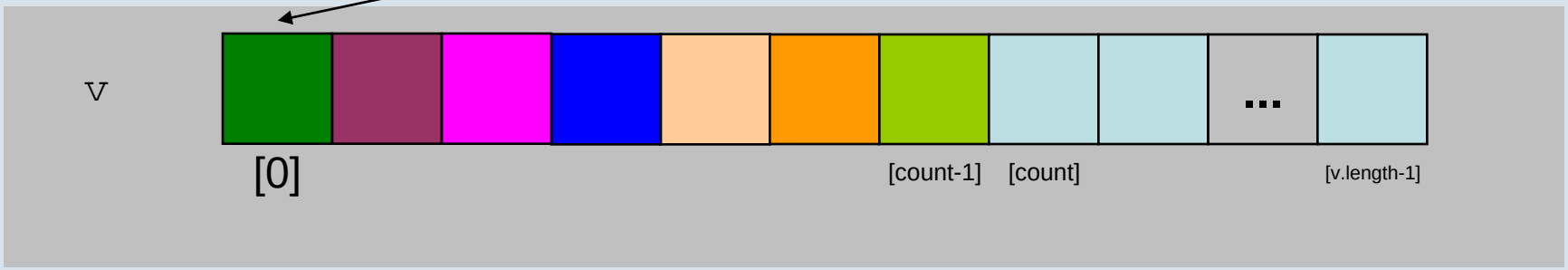
```
public boolean searchForward(type e) {
  boolean found = false;
  int i=0;
  while(i<count && !found)
    if(v[i] == e)
      found = true;
    else i++;
  return found;
}
```

Existirá no vector o elemento

e =  ?

found = false

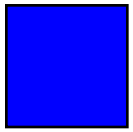
i=0



Operações em Vectores
(Pesquisa Linear – índices crescentes)

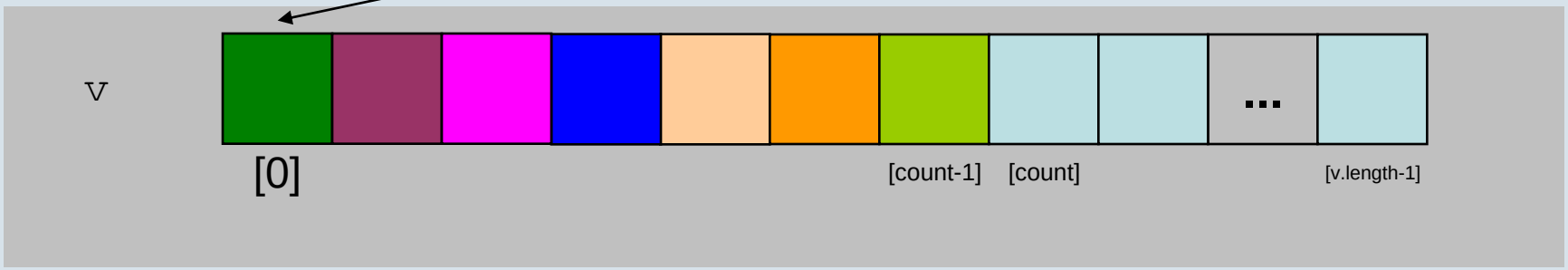
```
public boolean searchForward(type e) {
  boolean found = false;
  int i=0;
  while(i<count && !found)
    if(v[i] == e)
      found = true;
    else i++;
  return found;
}
```

Existirá no vector o elemento

e =  ?

found = false

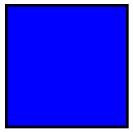
i=0



Operações em Vectores
(Pesquisa Linear – índices crescentes)

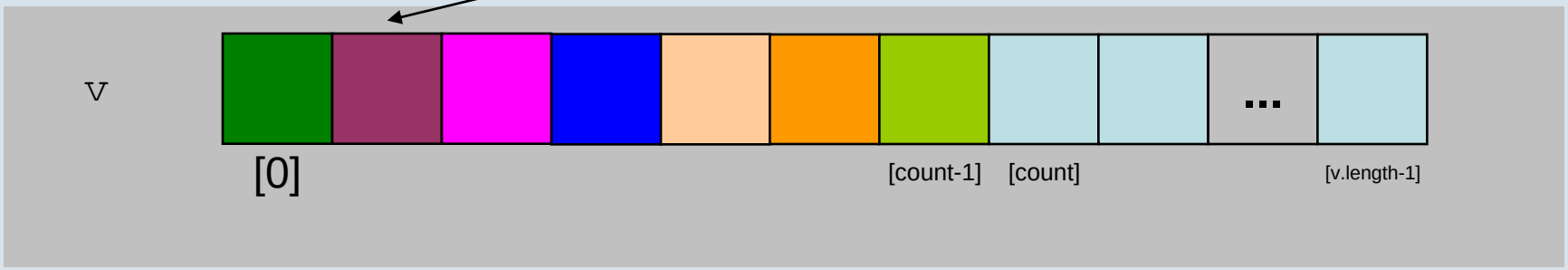
```
public boolean searchForward(type e) {
  boolean found = false;
  int i=0;
  while(i<count && !found)
    if(v[i] == e)
      found = true;
    else i++;
  return found;
}
```

Existirá no vector o elemento

e =  ?

```
found = false
```

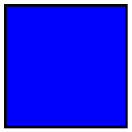
```
i=1
```



Operações em Vectores
(Pesquisa Linear – índices crescentes)

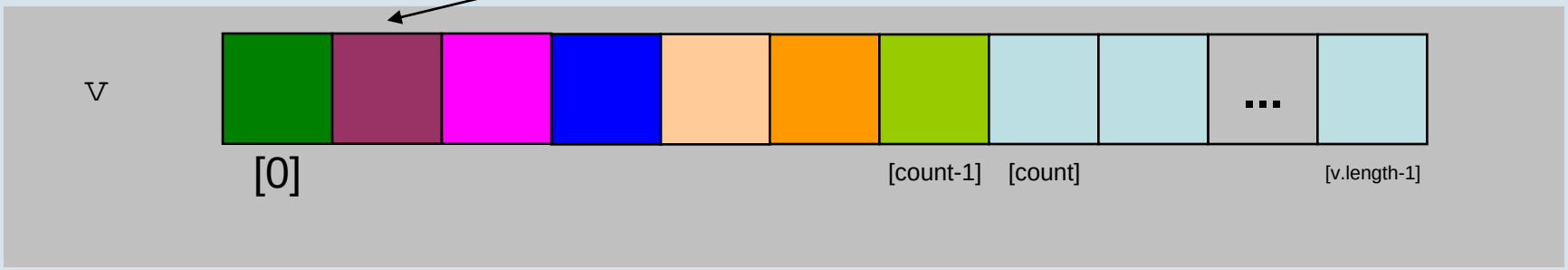
```
public boolean searchForward(type e) {
  boolean found = false;
  int i=0;
  while(i<count && !found)
    if(v[i] == e)
      found = true;
    else i++;
  return found;
}
```

Existirá no vector o elemento

e =  ?

found = false

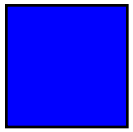
i=1



Operações em Vectores
(Pesquisa Linear – índices crescentes)

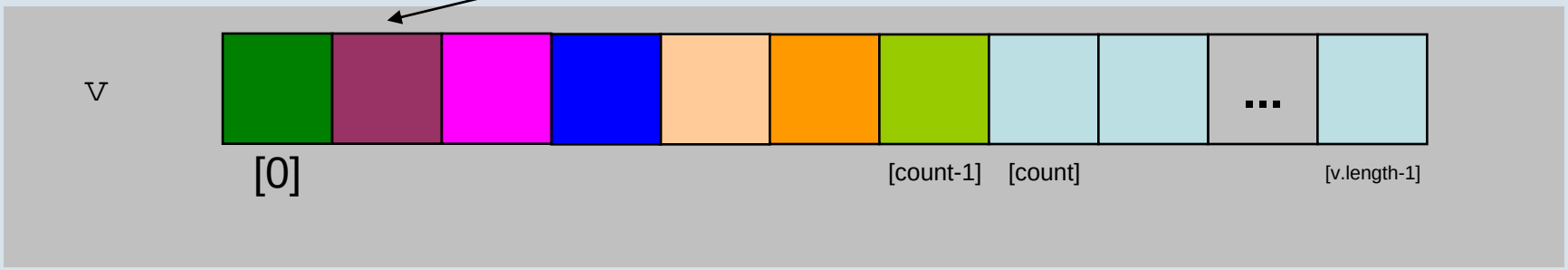
```
public boolean searchForward(type e) {
  boolean found = false;
  int i=0;
  while(i<count && !found)
    if(v[i] == e)
      found = true;
    else i++;
  return found;
}
```

Existirá no vector o elemento

e =  ?

found = false

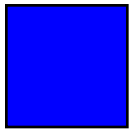
i=1



Operações em Vectores
(Pesquisa Linear – índices crescentes)

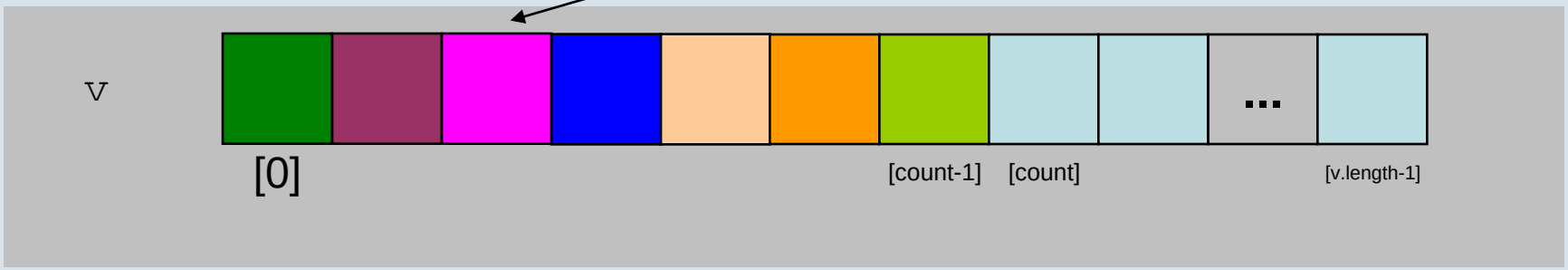
```
public boolean searchForward(type e) {
  boolean found = false;
  int i=0;
  while(i<count && !found)
    if(v[i] == e)
      found = true;
    else i++;
  return found;
}
```

Existirá no vector o elemento

e =  ?

found = false

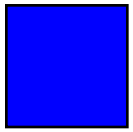
i=2



Operações em Vectores
(Pesquisa Linear – índices crescentes)

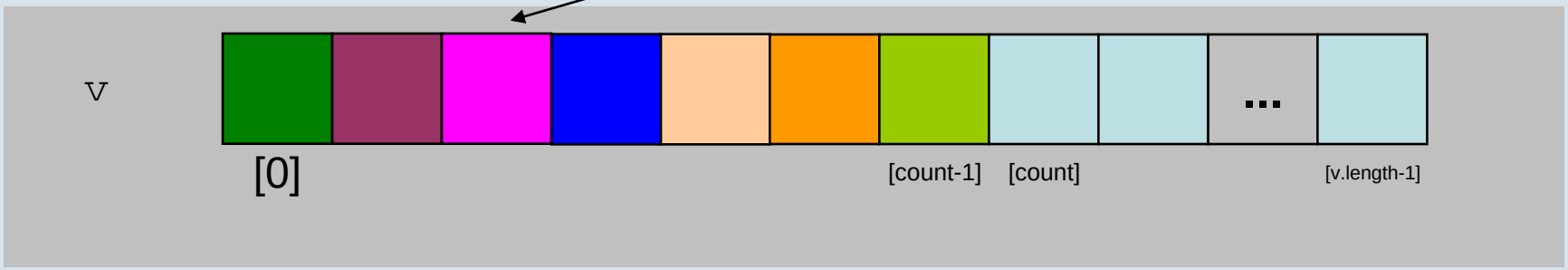
```
public boolean searchForward(type e) {
  boolean found = false;
  int i=0;
  while(i<count && !found)
    if(v[i] == e)
      found = true;
    else i++;
  return found;
}
```

Existirá no vector o elemento

e =  ?

found = false

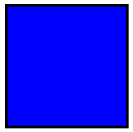
i=2



Operações em Vectores
(Pesquisa Linear – índices crescentes)

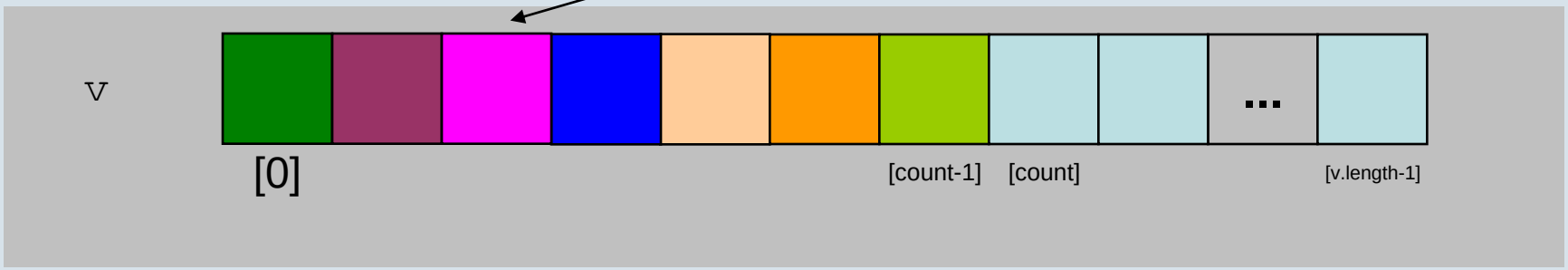
```
public boolean searchForward(type e) {
  boolean found = false;
  int i=0;
  while(i<count && !found)
    if(v[i] == e)
      found = true;
    else i++;
  return found;
}
```

Existirá no vector o elemento

e =  ?

found = false

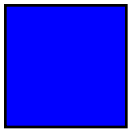
i=2



Operações em Vectores
(Pesquisa Linear – índices crescentes)

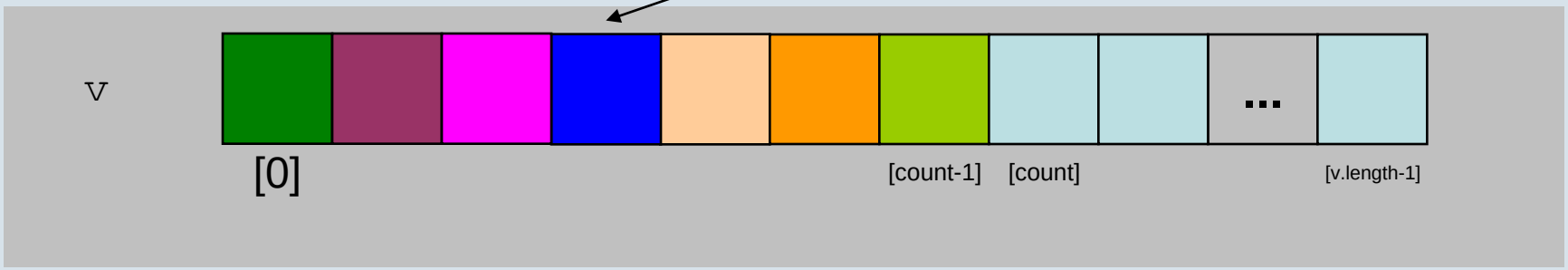
```
public boolean searchForward(type e) {
  boolean found = false;
  int i=0;
  while(i<count && !found)
    if(v[i] == e)
      found = true;
    else i++;
  return found;
}
```

Existirá no vector o elemento

e =  ?

found = false

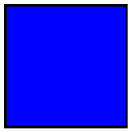
i=3



Operações em Vectores
(Pesquisa Linear – índices crescentes)

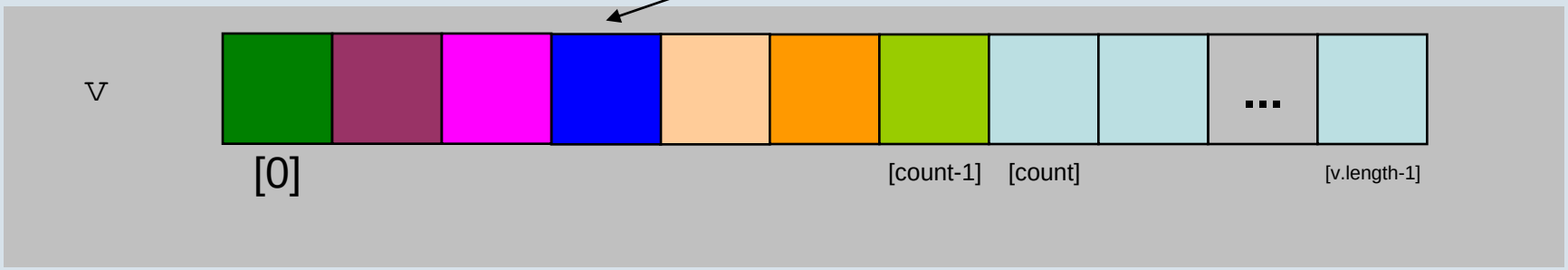
```
public boolean searchForward(type e) {
  boolean found = false;
  int i=0;
  while(i<count && !found)
    if(v[i] == e)
      found = true;
    else i++;
  return found;
}
```

Existirá no vector o elemento

e =  ?

found = false

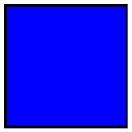
i=3



Operações em Vectores
(Pesquisa Linear – índices crescentes)

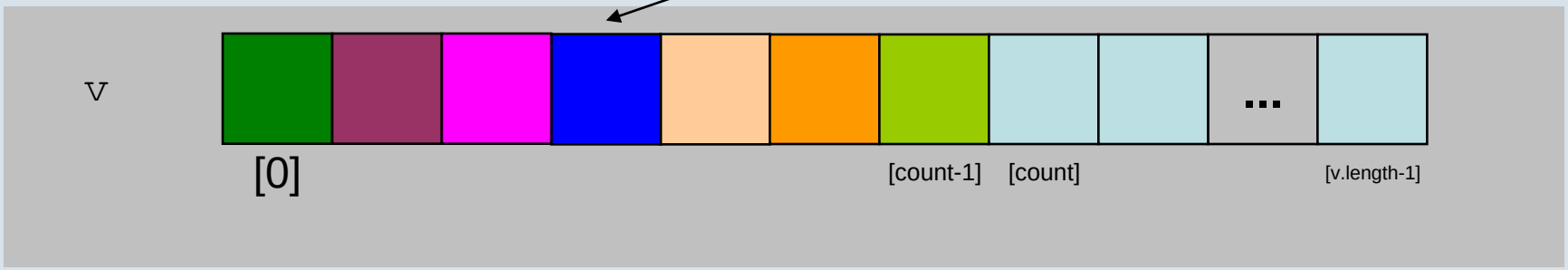
```
public boolean searchForward(type e) {
  boolean found = false;
  int i=0;
  while(i<count && !found)
    if(v[i] == e)
      found = true;
    else i++;
  return found;
}
```

Existirá no vector o elemento

e =  ?

found = false

i=3



Operações em Vectores
(Pesquisa Linear – índices crescentes)

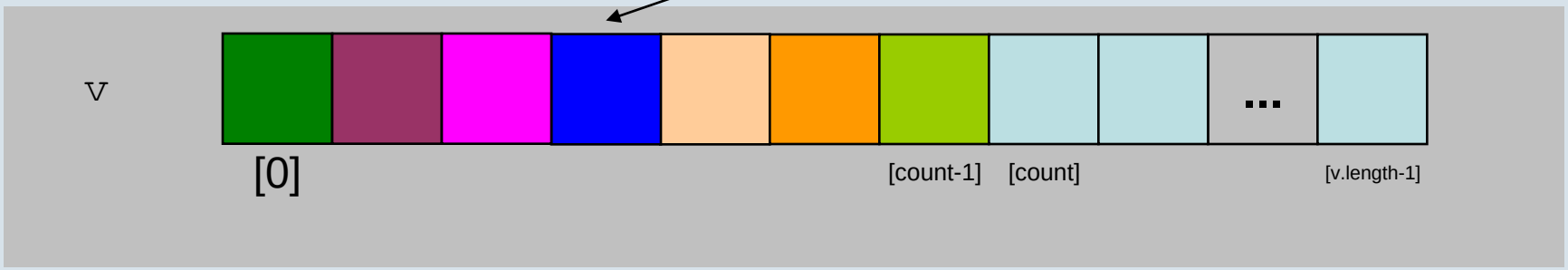
```
public boolean searchForward(type e) {
  boolean found = false;
  int i=0;
  while(i<count && !found)
    if(v[i] == e)
      found = true;
    else i++;
  return found;
}
```

Existirá no vector o elemento

e = ?

found = true

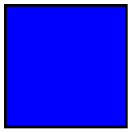
i=3



Operações em Vectores
(Pesquisa Linear – índices crescentes)

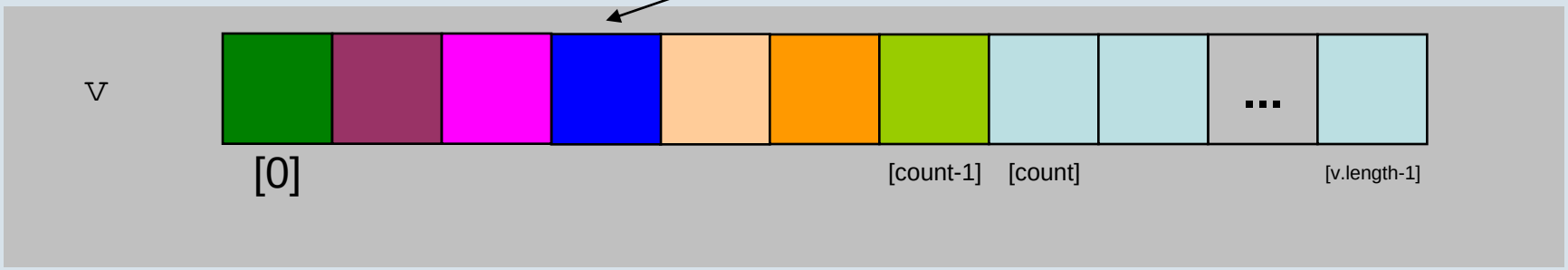
```
public boolean searchForward(type e) {
  boolean found = false;
  int i=0;
  while(i<count && !found)
    if(v[i] == e)
      found = true;
    else i++;
  return found;
}
```

Existirá no vector o elemento

e =  ?

found = true

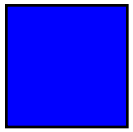
i=3



Operações em Vectores
(Pesquisa Linear – índices crescentes)

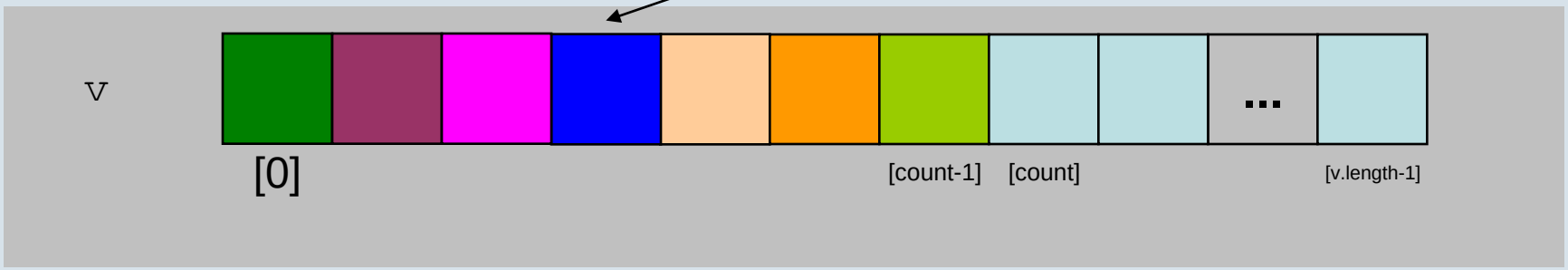
```
public boolean searchForward(type e) {
  boolean found = false;
  int i=0;
  while(i<count && !found)
    if(v[i] == e)
      found = true;
    else i++;
  return found;
}
```

Existirá no vector o elemento

e =  ?

found = true

i=3



Operações em Vectores
(Pesquisa Linear – índices crescentes)

```
public boolean searchForward(...) {
  boolean found = false;
  int i=0;
  while(i<count && !found)
    if(p(v[i])
      found = true;
    else i++;
  return found;
}
```

Por vezes a busca não pretende saber se um elemento específico se encontra no vector mas apenas se existe um elemento que satisfaça uma determinada condição.

p() é uma função booleana e representa o critério que determina se um dado elemento é o elemento procurado.

Exemplos: procurar um elemento que seja um número primo, procurar uma conta com saldo negativo, etc...

Operações em Vectores
(Pesquisa Linear – índices decrescentes)

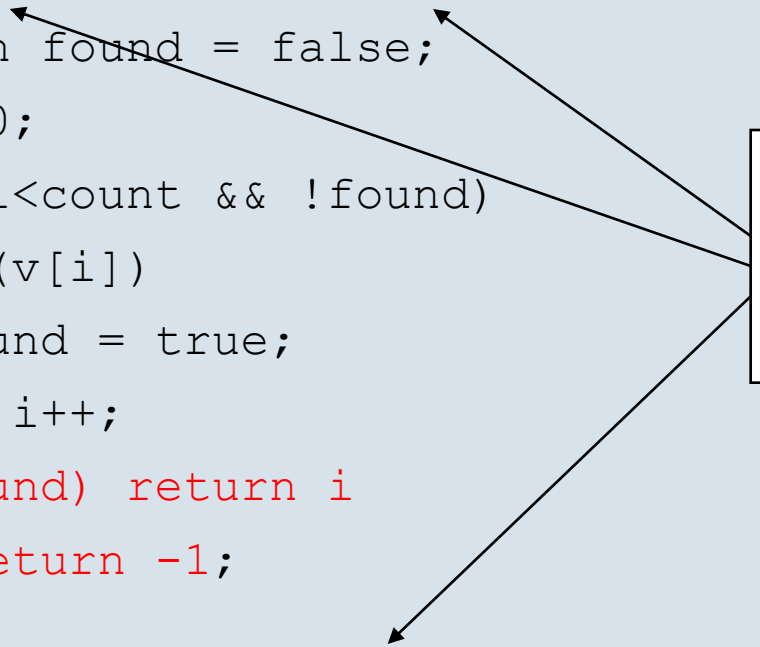
```
public boolean searchBackward(...) {
  boolean found = false;
  int i=count-1;
  while(i>=0 && !found)
    if(p(v[i])
      found = true;
    else i--;
  return found;
}
```

Esta busca é idêntica à anterior mas efectua-se da direita para a esquerda.

Por vezes queremos saber a última ocorrência de um determinado elemento dentro dum vector...

Operações em Vectores
(Pesquisa Linear – índices crescentes)

```
public int searchIndexOf(...) {
  boolean found = false;
  int i=0;
  while(i<count && !found)
    if(p(v[i])
      found = true;
    else i++;
  if (found) return i
  else return -1;
}
```



Se se pretender determinar a localização do elemento a procurar é fácil adaptar a função de pesquisa...

Operações em Vectores
(Pesquisa Linear – índices crescentes)

```
public boolean searchForward(...) {
   return searchIndexOf(...) != -1;
}
```

Mas o melhor será programar a operação mais geral (a que determina a posição do elemento dentro do vector) e programar a operação de busca simples à custa da primeira…