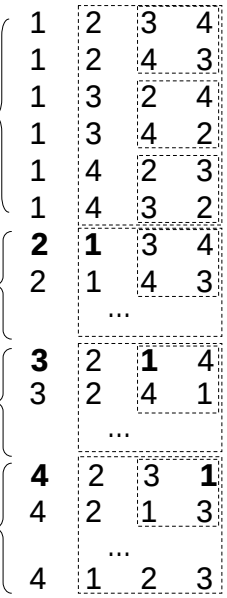
Permutation from Swapping (1/4)

- **Recursive**
  - 1 + permutations of {2, 3, 4}
  - 2 + permutations of {1, 3, 4}
  - 3 + permutations of {2, 1, 4}
  - 4 + permutations of {2, 3, 1}

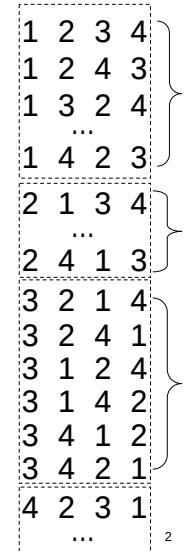
```
for (i=0; i<n; i++) a[i] = i+1;
permutation(a, 0, n-1);
```

```
void permutation(int perm[], int start, int end) {
    if (start == end) printPerm(perm, end+1);
    for (int i=start; i<=end; i++) {
        swap(&perm[start], &perm[i]);
        permutation(perm, start+1, end);
        swap(&perm[start], &perm[i]);
    }
}
```



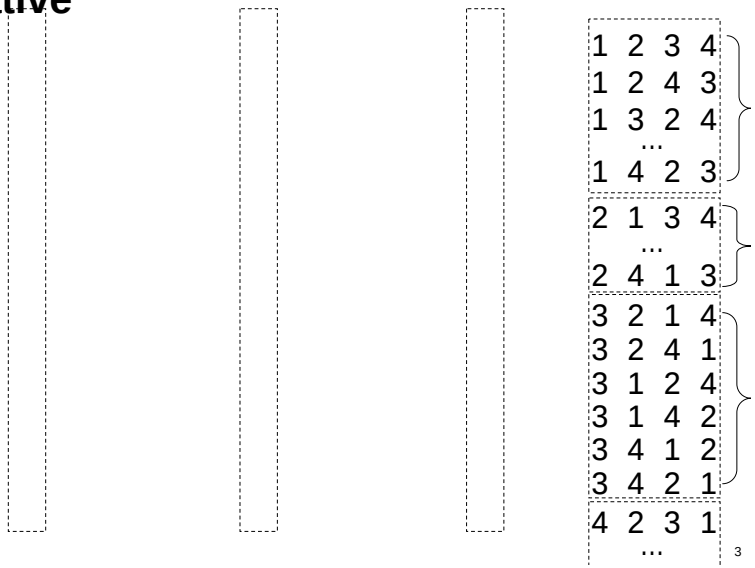
Permutation from Swapping (2/4)

- **Iterative**



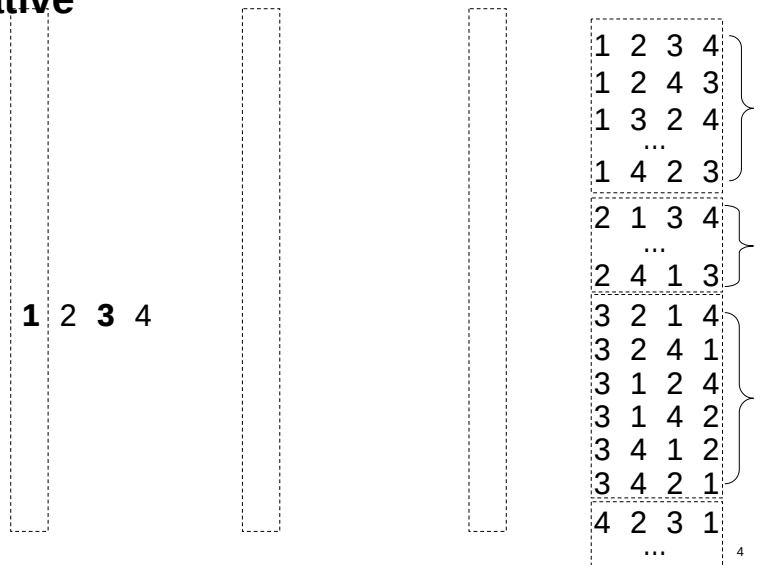
Permutation from Swapping (2/4)

- **Iterative**



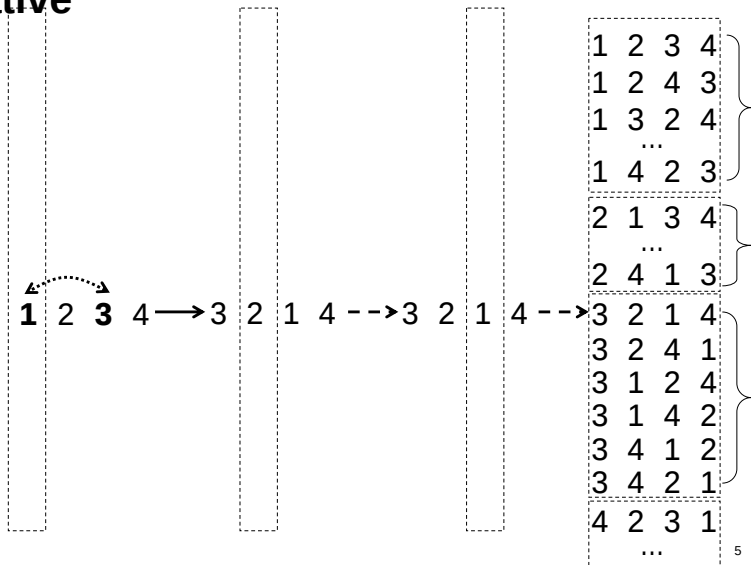
Permutation from Swapping (2/4)

- **Iterative**



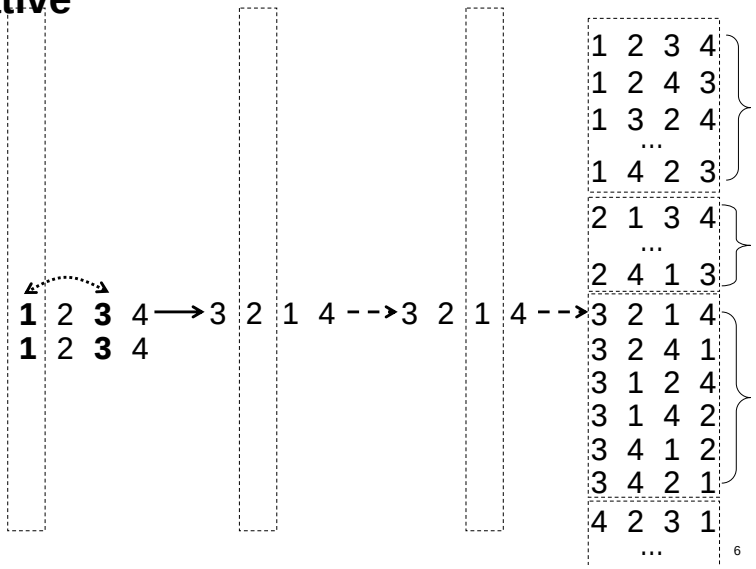
Permutation from Swapping (2/4)

- **Iterative**



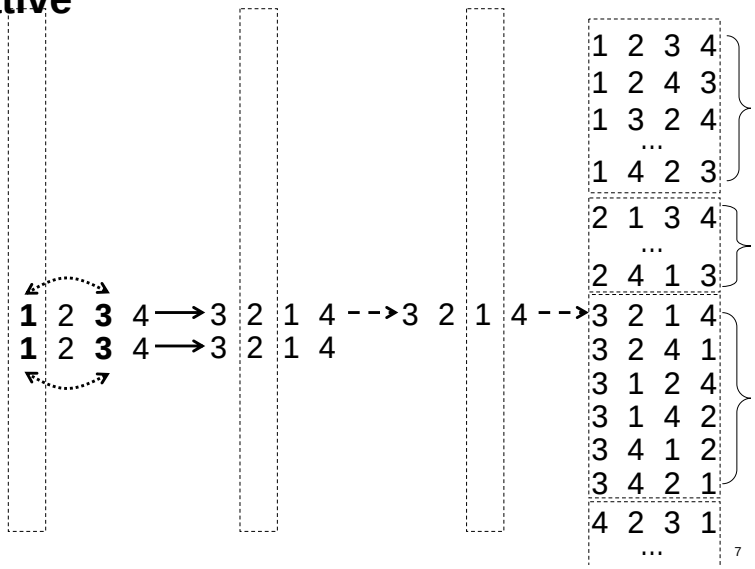
Permutation from Swapping (2/4)

- **Iterative**



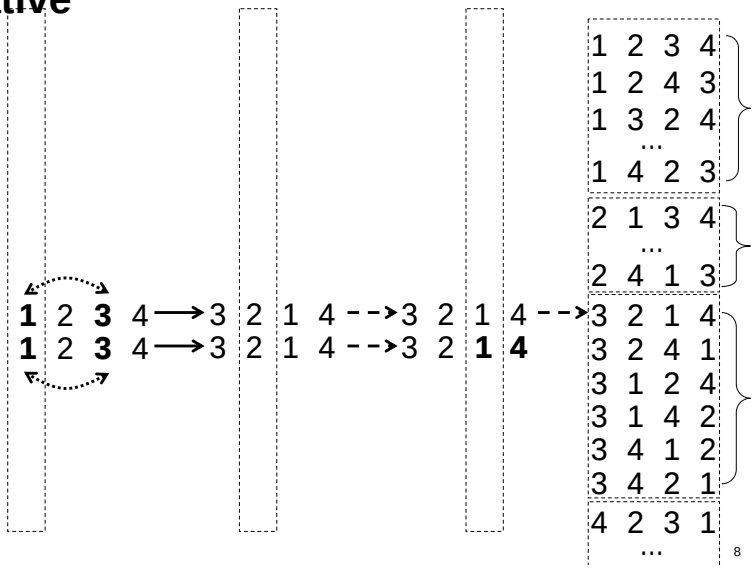
Permutation from Swapping (2/4)

- **Iterative**



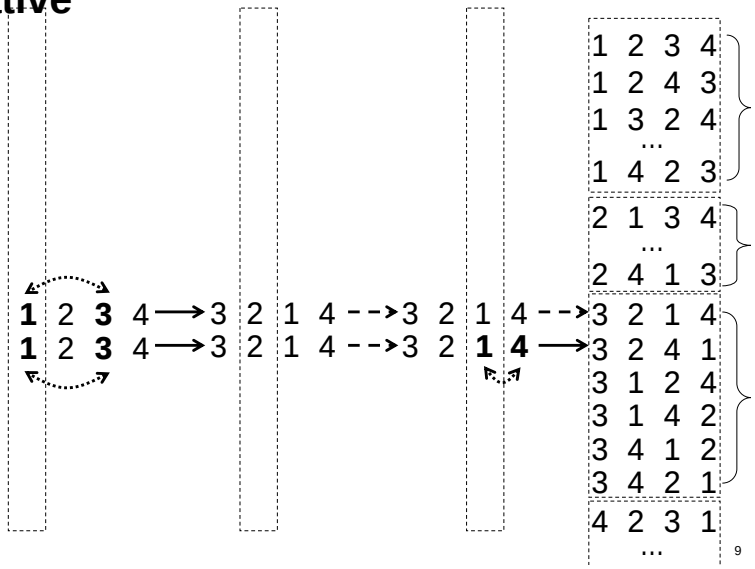
Permutation from Swapping (2/4)

- **Iterative**



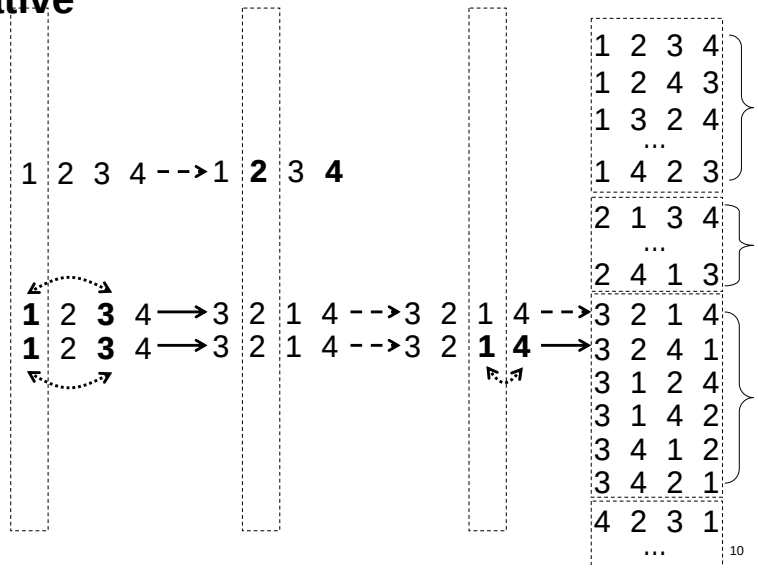
Permutation from Swapping (2/4)

- **Iterative**



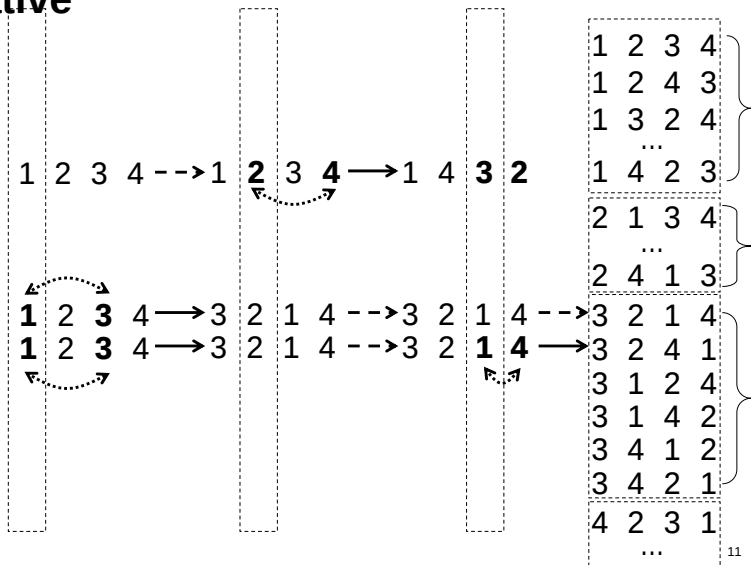
Permutation from Swapping (2/4)

- **Iterative**



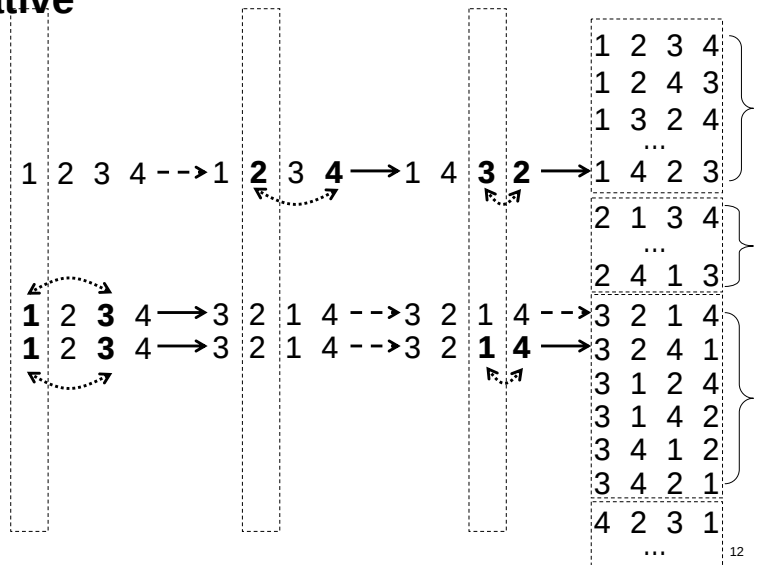
Permutation from Swapping (2/4)

- **Iterative**



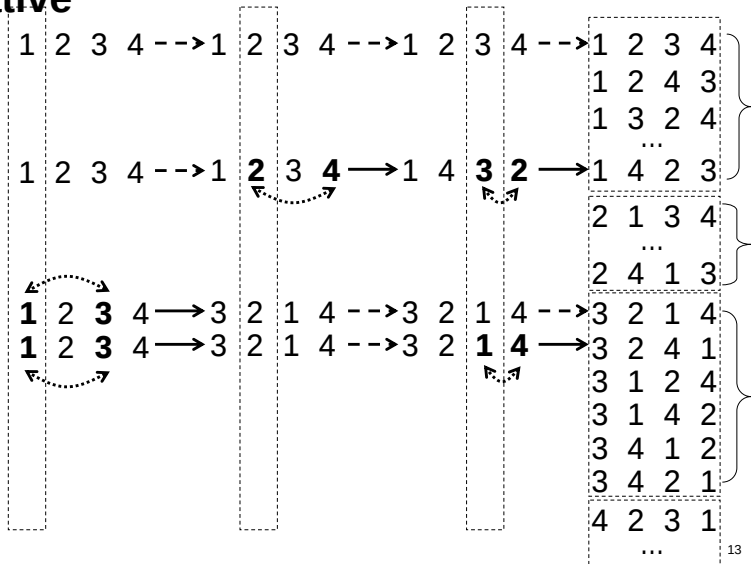
Permutation from Swapping (2/4)

- **Iterative**



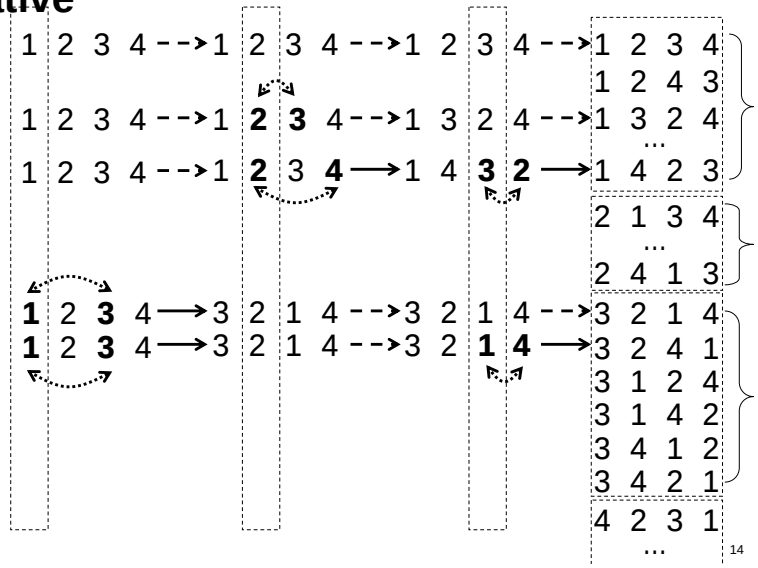
Permutation from Swapping (2/4)

- **Iterative**



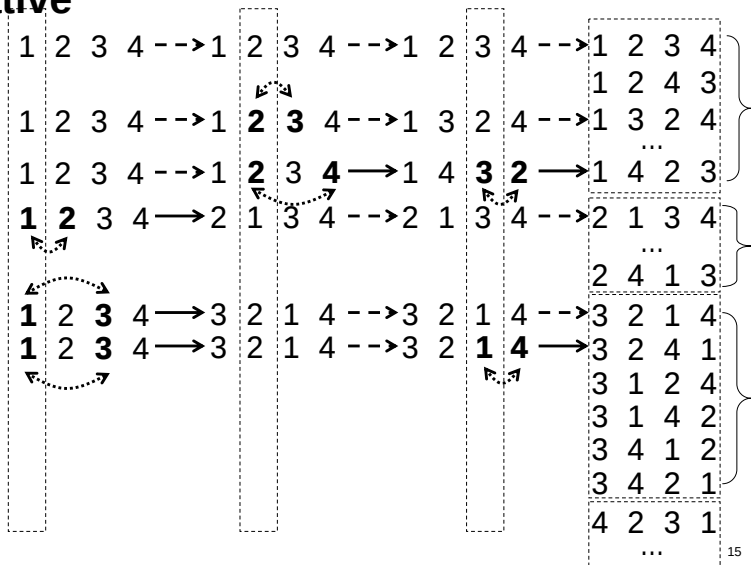
Permutation from Swapping (2/4)

- **Iterative**



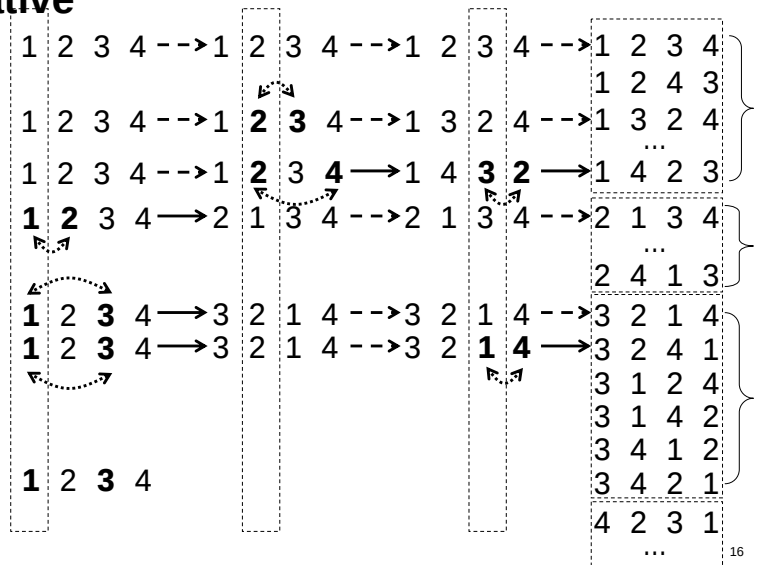
Permutation from Swapping (2/4)

- **Iterative**



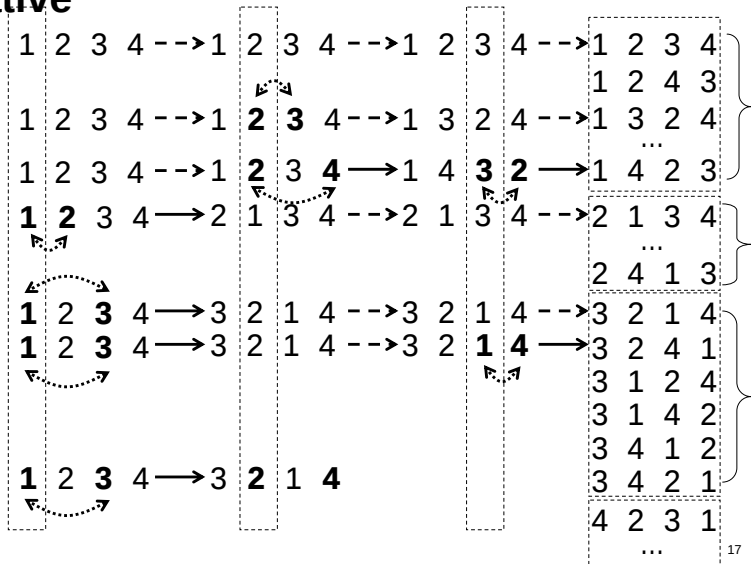
Permutation from Swapping (2/4)

- **Iterative**



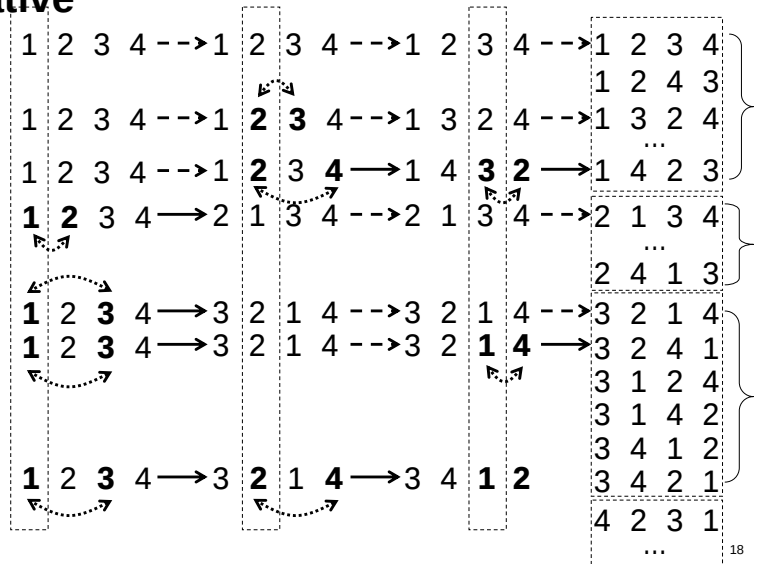
Permutation from Swapping (2/4)

- **Iterative**



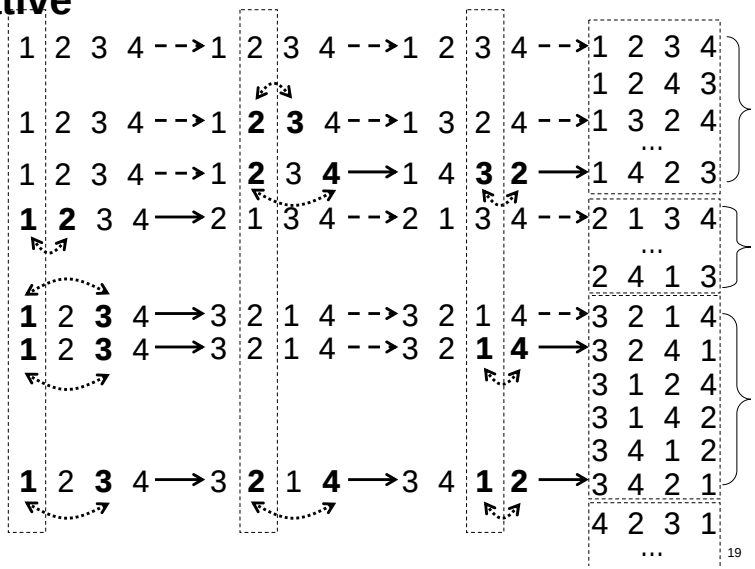
Permutation from Swapping (2/4)

- **Iterative**



Permutation from Swapping (2/4)

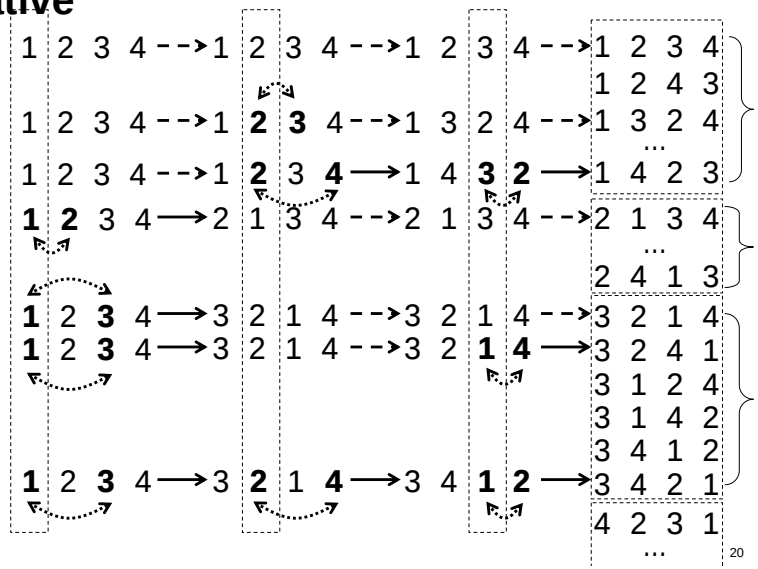
- **Iterative**



Permutation from Swapping (2/4)

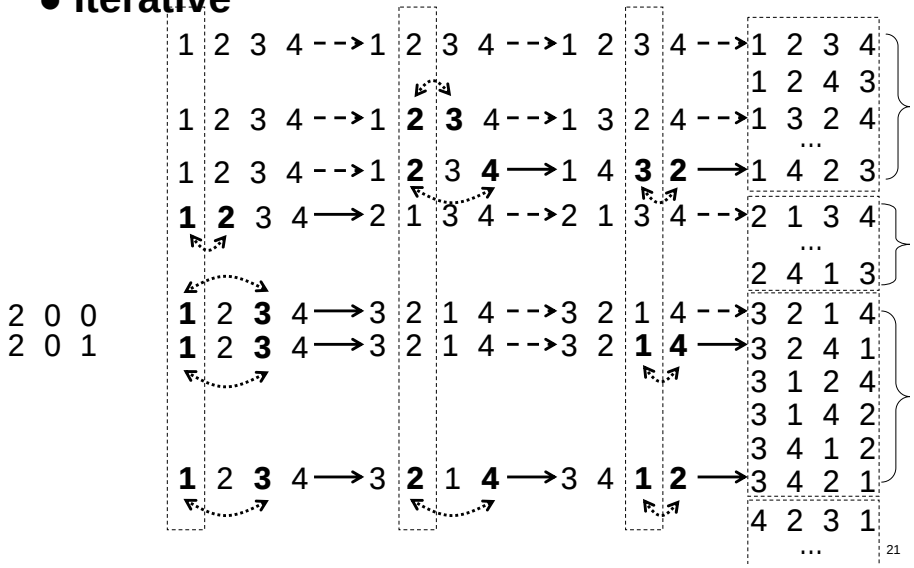
- **Iterative**

2 0 0



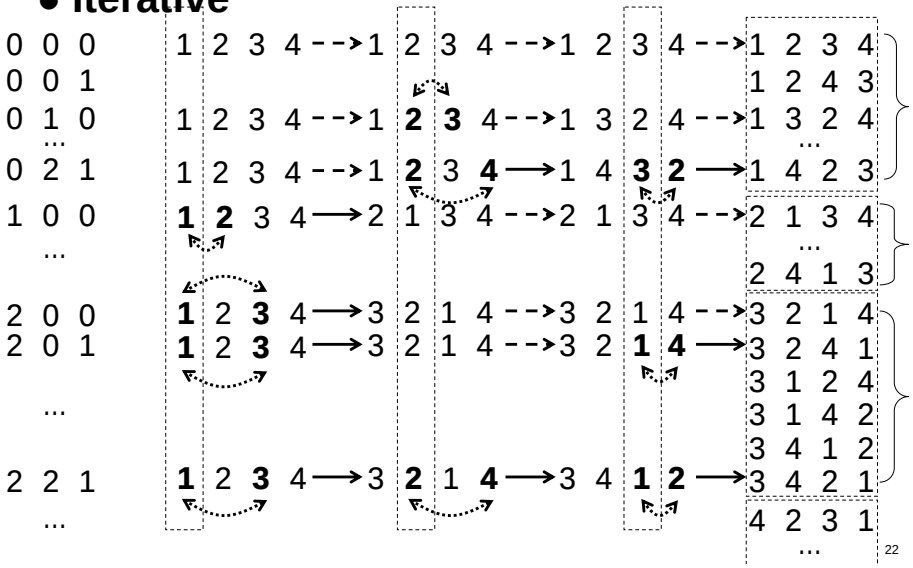
Permutation from Swapping (2/4)

- **Iterative**



Permutation from Swapping (2/4)

- **Iterative**



Permutation from Swapping (3/4)

- counting up

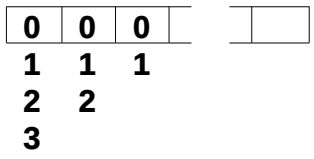
Permutation from Swapping (3/4)

- counting up

0	**0**	**0**			
1	**1**	**1**			
2	**2**				
3					

Permutation from Swapping (3/4)

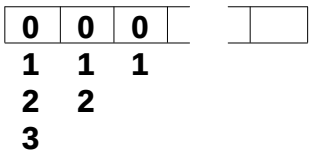
- counting up



0 0 0
0 0 1
0 1 0
0 1 1
0 2 0
0 2 1

Permutation from Swapping (3/4)

- counting up

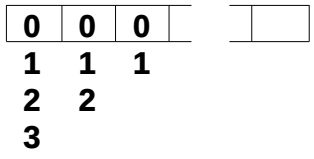


0 0 0
0 0 1
0 1 0
0 1 1
0 2 0
0 2 1

1 0 0
1 0 1
1 1 0
1 1 1
1 2 0
1 2 1

Permutation from Swapping (3/4)

- counting up



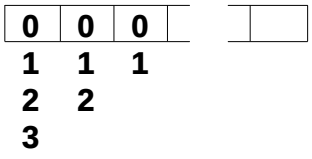
0 0 0
0 0 1
0 1 0
0 1 1
0 2 0
0 2 1

2 0 0
2 0 1
2 1 0
2 1 1
2 2 0
2 2 1

1 0 0
1 0 1
1 1 0
1 1 1
1 2 0
1 2 1

Permutation from Swapping (3/4)

- counting up



0 0 0
0 0 1
0 1 0
0 1 1
0 2 0
0 2 1

2 0 0
2 0 1
2 1 0
2 1 1
2 2 0
2 2 1

1 0 0
1 0 1
1 1 0
1 1 1
1 2 0
1 2 1

3 0 0
3 0 1
3 1 0
3 1 1
3 2 0
3 2 1

n-layer for Loop Implementation

```
int i1, i2, i3, i4;
for (i1=1; i1<=4; i1++) {
   for (i2=1; i2<=4; i2++) {
      if (i2 == i1) continue;
      for (i3=1; i3<=4; i3++) {
         if ((i3==i2)||(i3==i1)) continue;
         for (i4=1; i4<=4; i4++) {
            if ((i4==i3)||(i4==i2)||(i4==i1)) continue;
            printf("%d %d %d %d\n", i1, i2, i3, i4);
         }
      }
   }
}
```

i1	i2	i3	i4

This is a quick implementation but **NOT scalable**.

2-layer for loop Implementation

- Consider this simplified loop without collision constraints

```
int i1, i2, i3, i4;
for (i1=1; i1<=4; i1++) {
   for (i2=1; i2<=4; i2++) {
      for (i3=1; i3<=4; i3++) {
         for (i4=1; i4<=4; i4++) {
            printf("%d %d %d %d\n",
                  i1, i2, i3, i4);
         }
      }
   }
}
```

i1	i2	i3	i4
1	1	1	1
1	1	1	2
1	1	1	3
1	1	1	4
1	1	2	1
1	1	2	2
1	1	2	3
1	1	2	4
1	1	3	1
...			

- Is there other ***scalable*** program structure to generate the same (i1, i2, i3, i4) sequence? yes

Equivalent 2-layer for loop

```
void nextIndex(int index[], int n) {
   int i;
   for (i=n-1; i>0; i--)
      if (index[i]<n)
         { index[i]++; return; }
      else
         index[i] = 1;
   index[0]++;
}
```

index

1	1	1	1
1	1	1	2
1	1	1	3
1	1	1	4
1	1	2	1
1	1	2	2
1	1	2	3
1	1	2	4
1	1	3	1
...			

Scalable

```
int index[4], n=4;
for (i=0; i<n; i++)
   index[i] = 1;
for (; index[0]<=n; nextIndex(index, n))
   printArray(index, n);
```


2-layer for loop for Permutation

index

1	2	3	4
1	2	4	3
1	3	2	4
1	3	4	2
1	4	2	3
1	4	3	2
2	1	3	4
2	1	4	3
...			