Algorithmen und Datenstrukturen
Tutorium V

Michael R. Jung

25. - 30. 05. 2016

1 Sortieralgorithmen

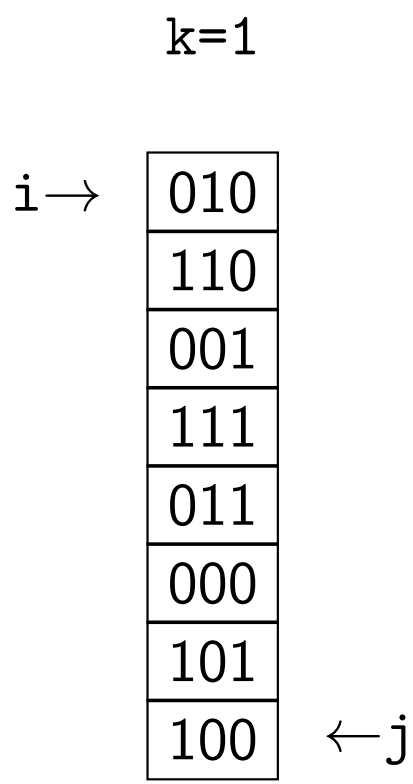
- RadixExchangeSort
- BucketSort

An folgender Beispielinstanz wird RadixExchangeSort veranschaulicht:

010
110
001
111
011
000
101
100

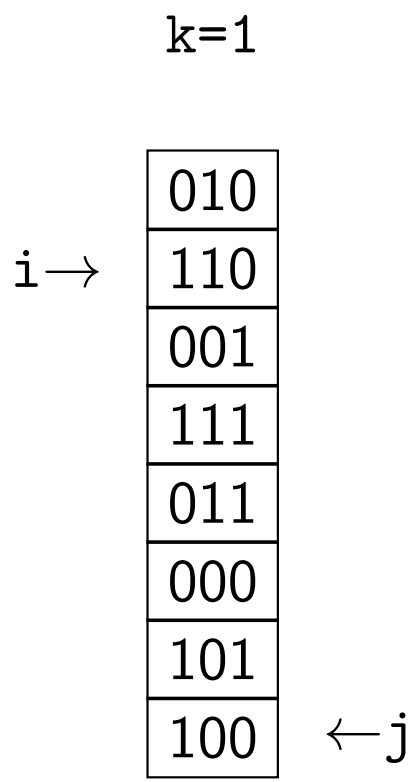
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



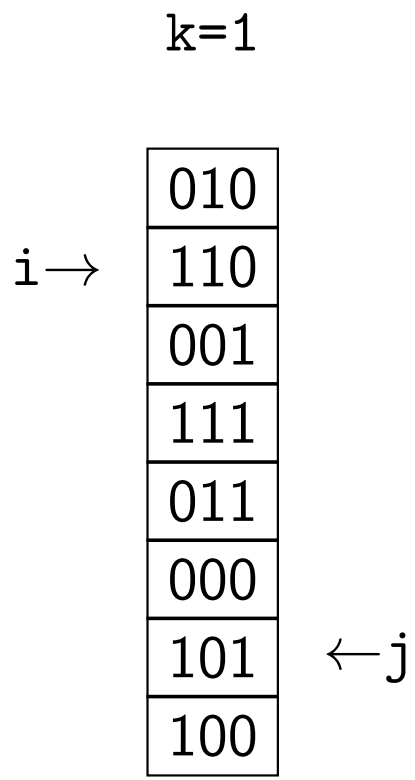
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



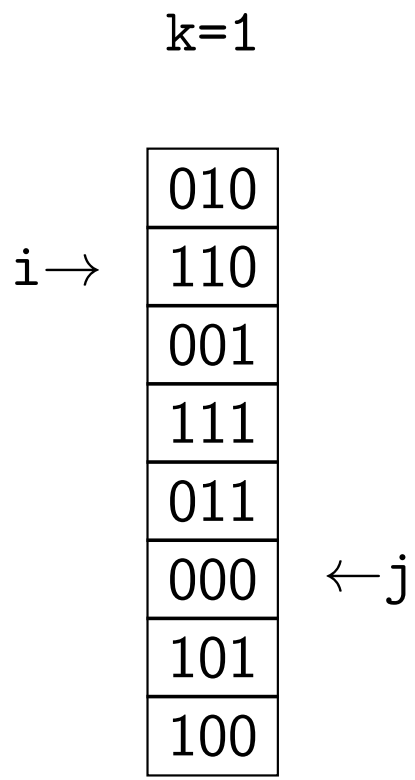
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



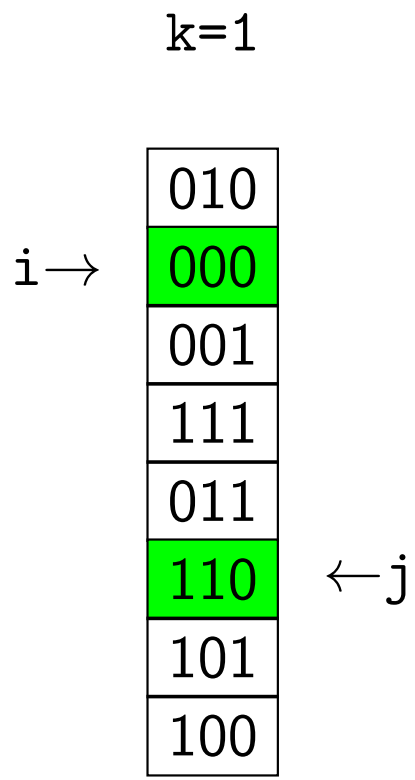
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



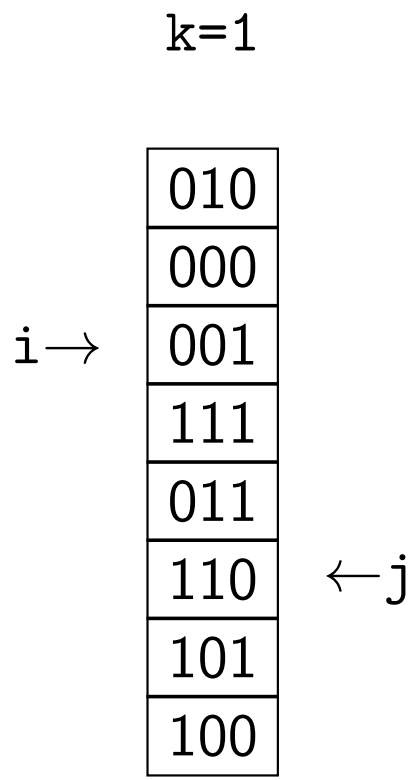
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



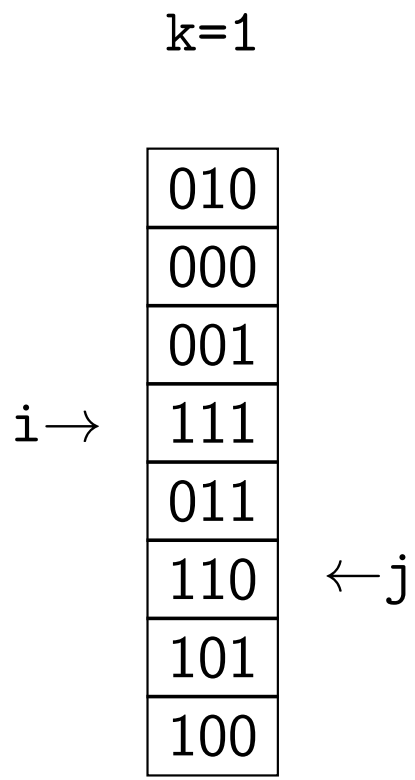
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



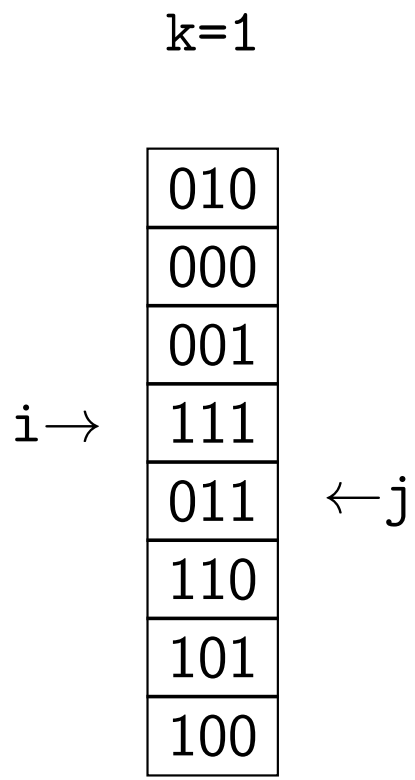
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



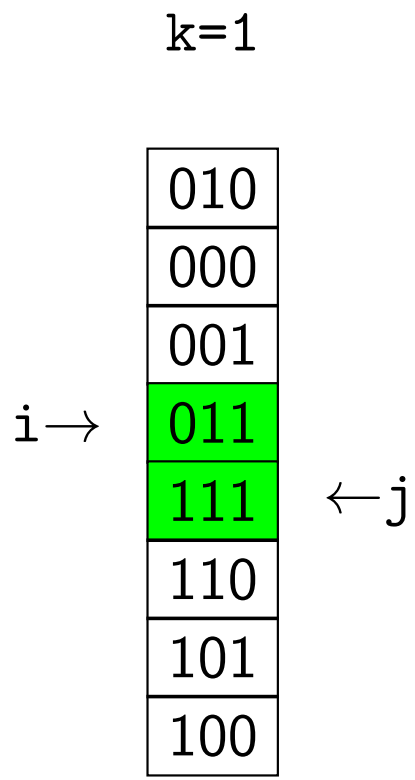
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



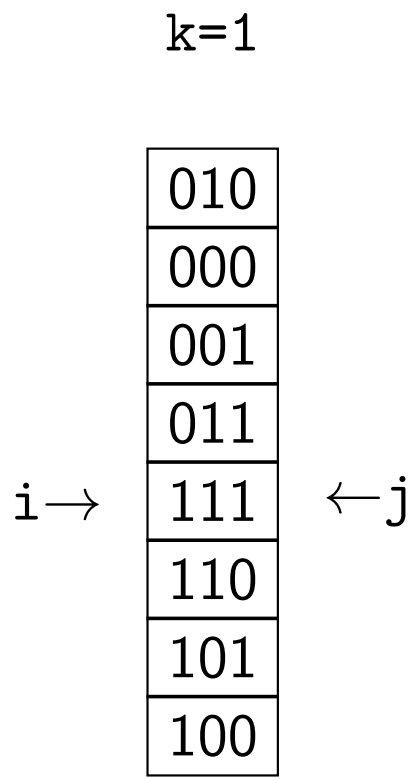
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



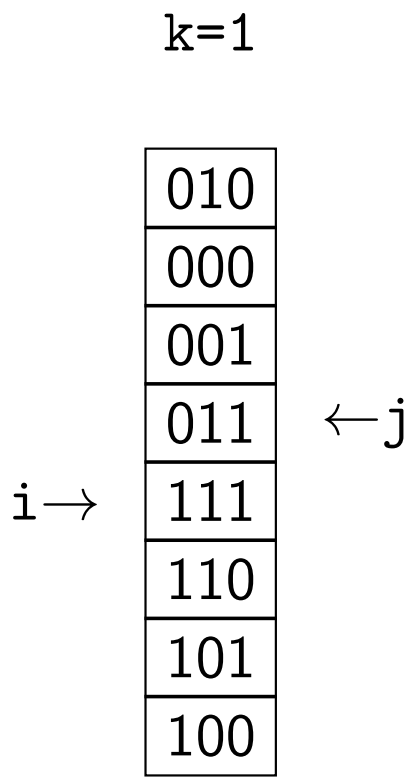
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



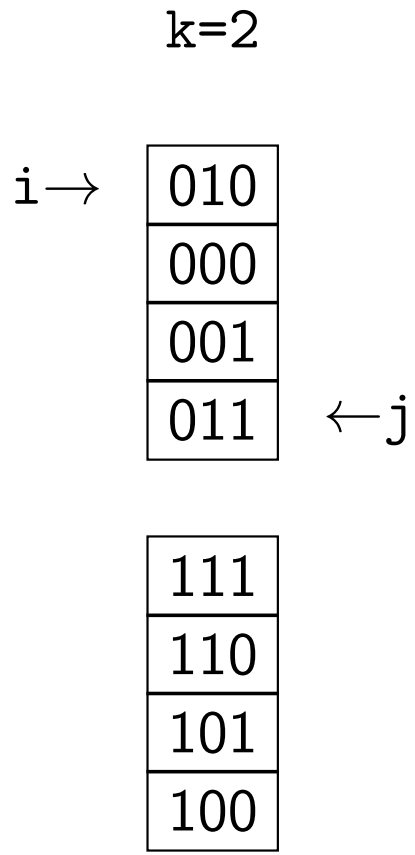
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



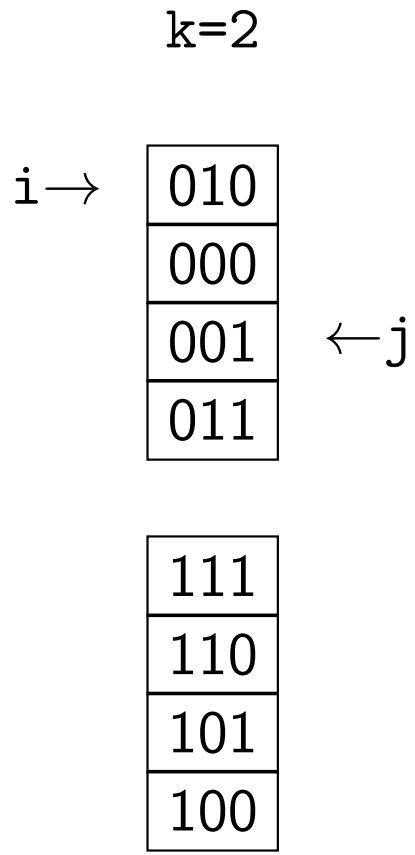
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



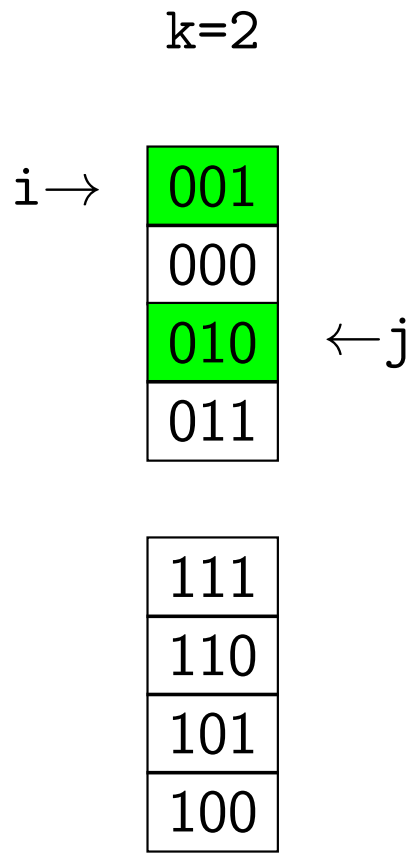
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



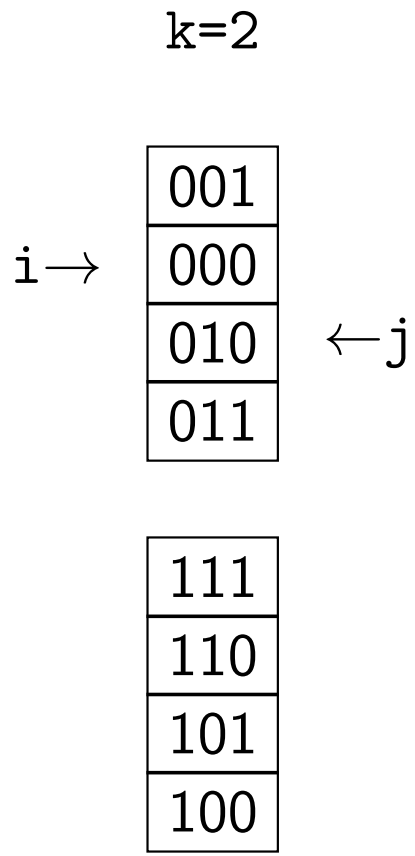
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



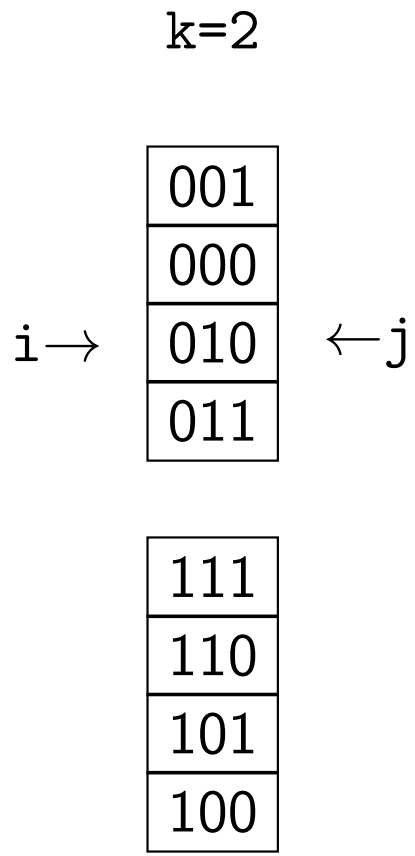
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



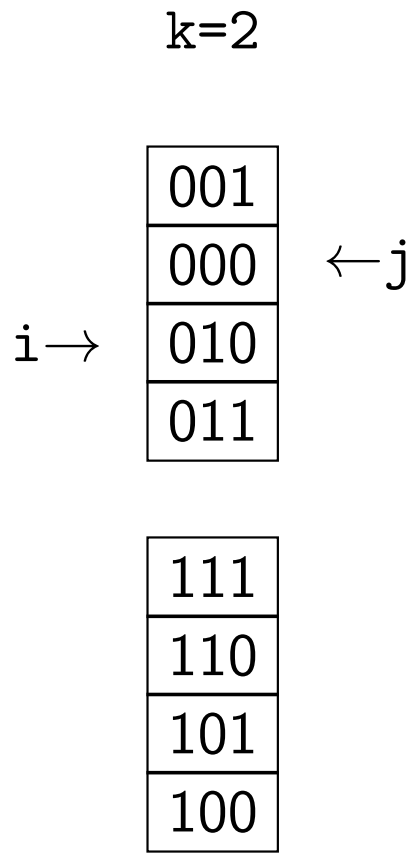
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



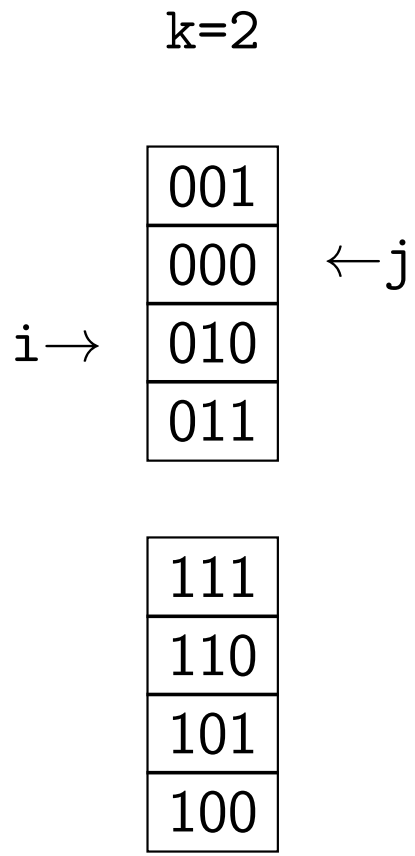
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



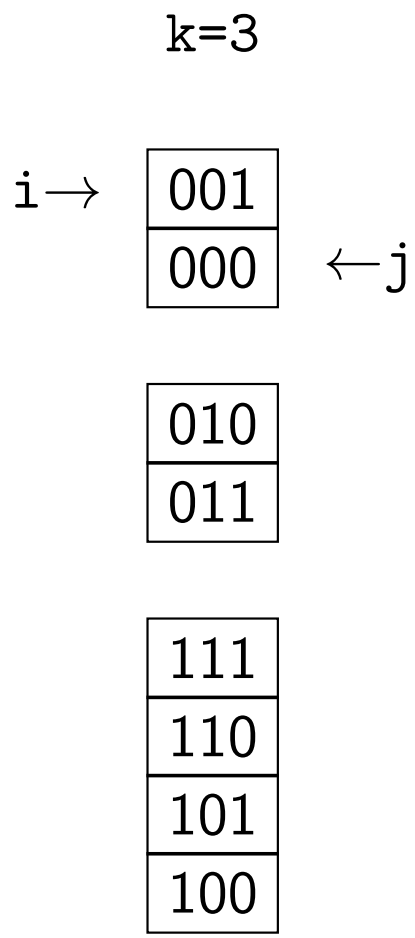
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



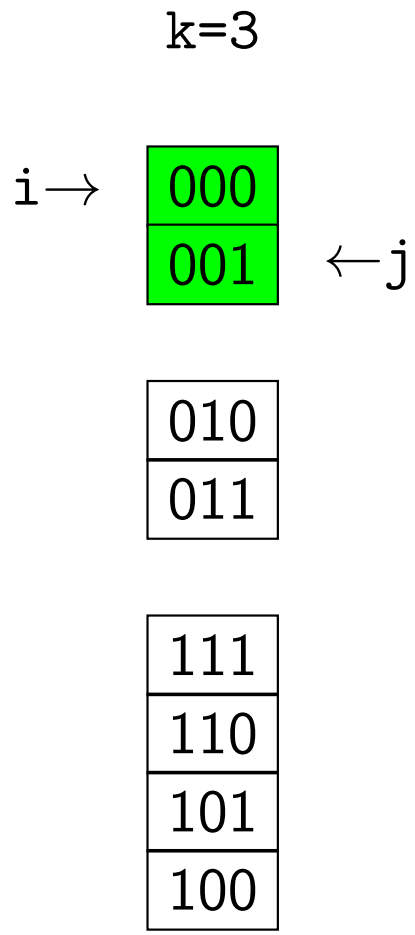
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



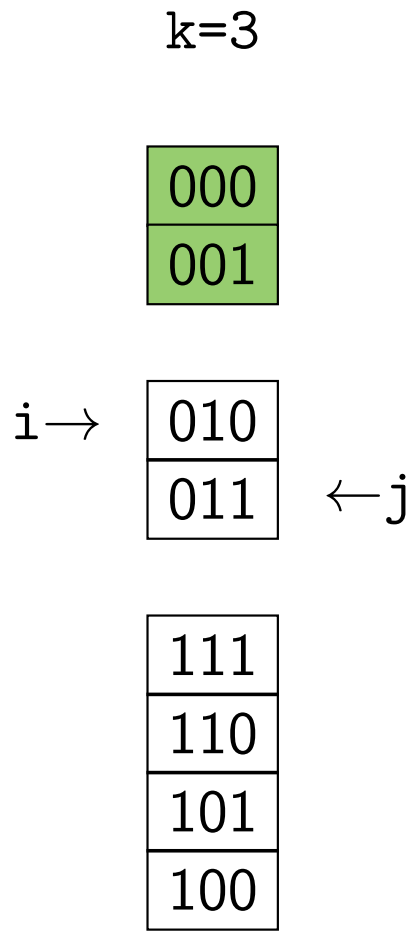
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



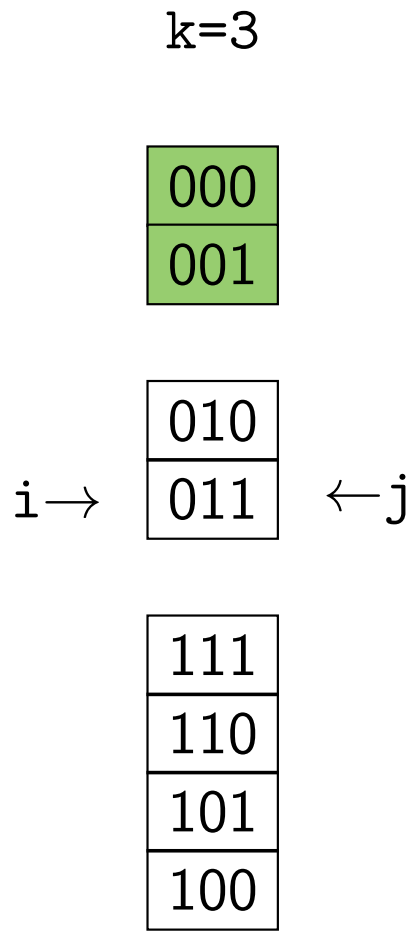
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



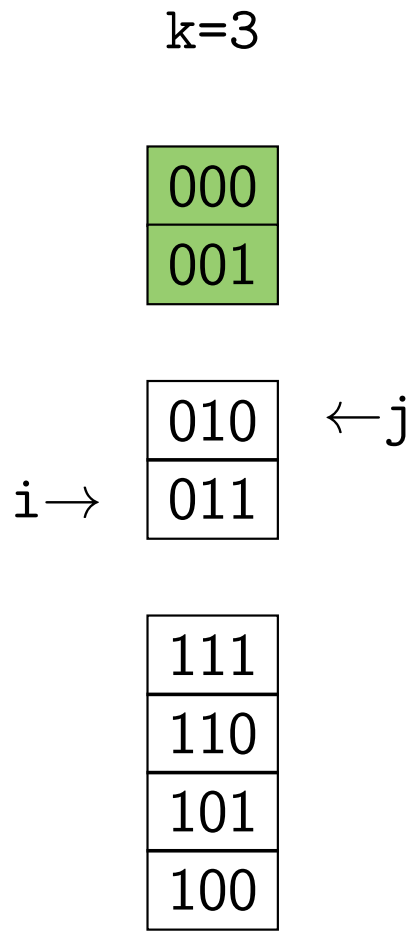
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



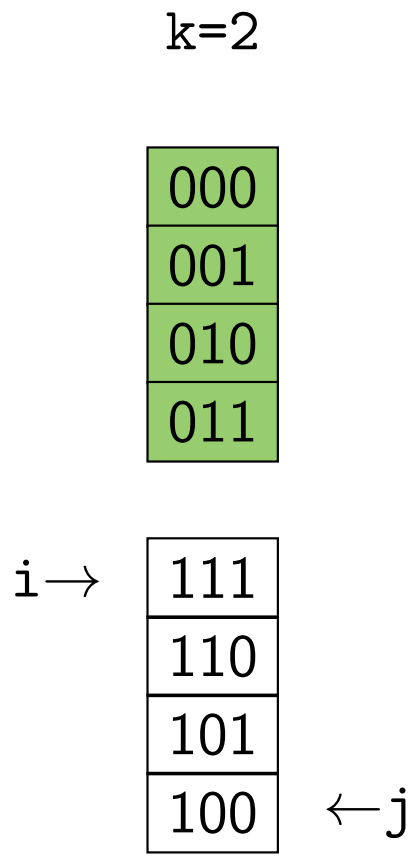
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



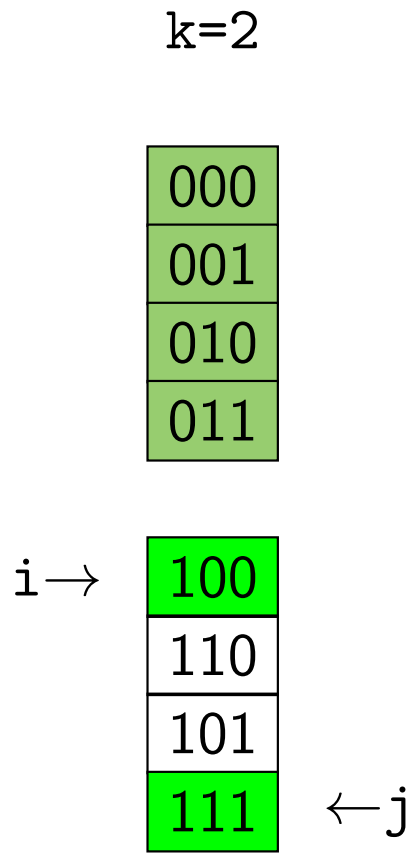
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



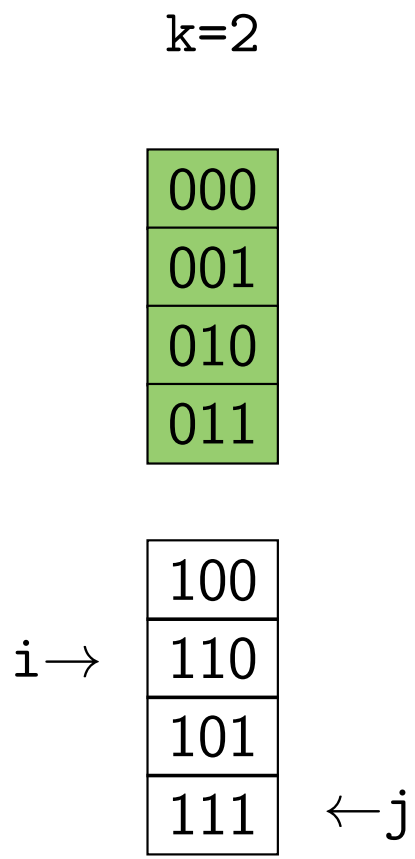
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



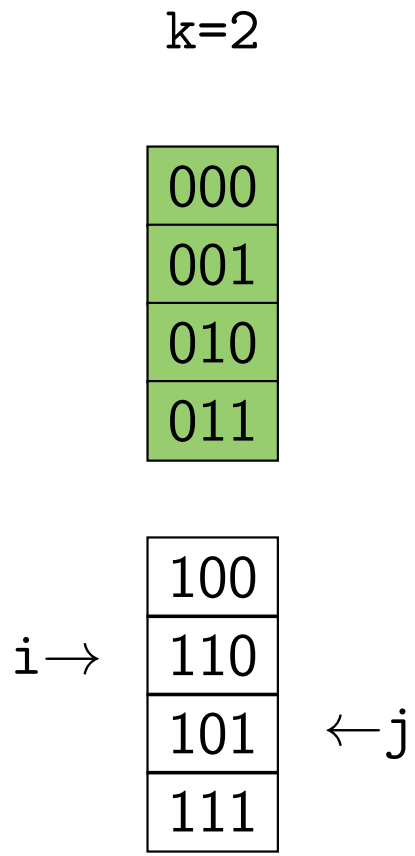
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



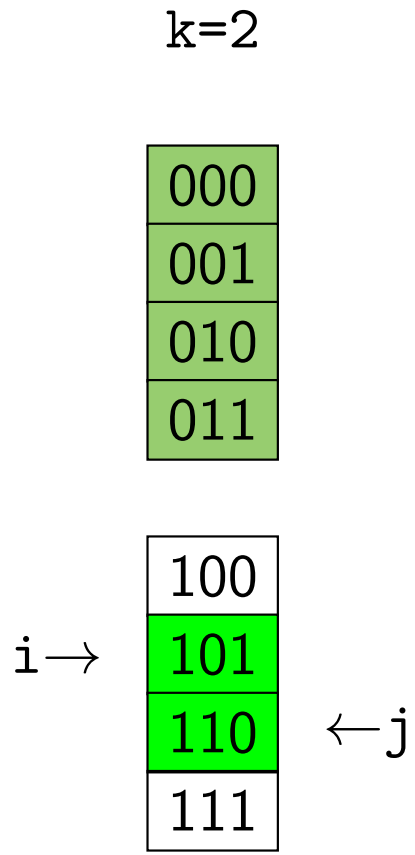
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



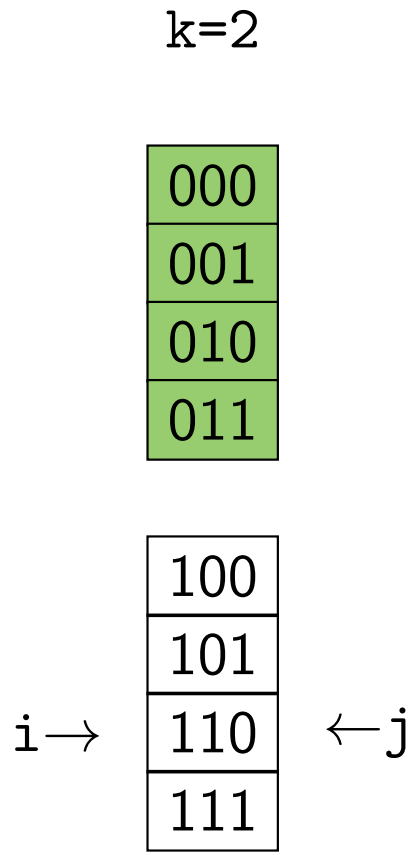
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



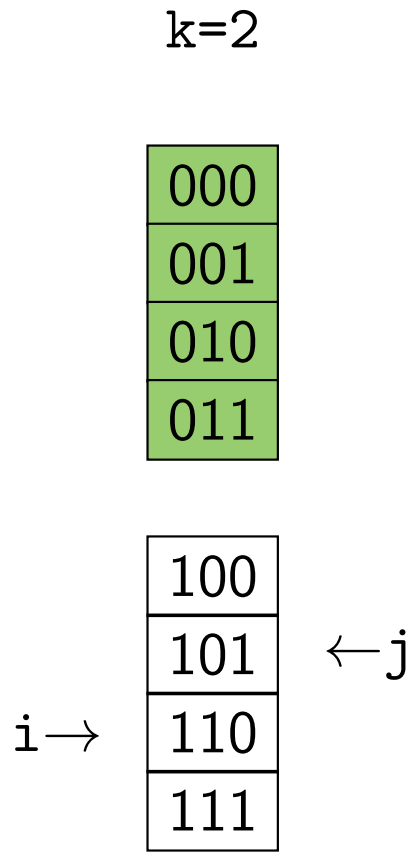
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



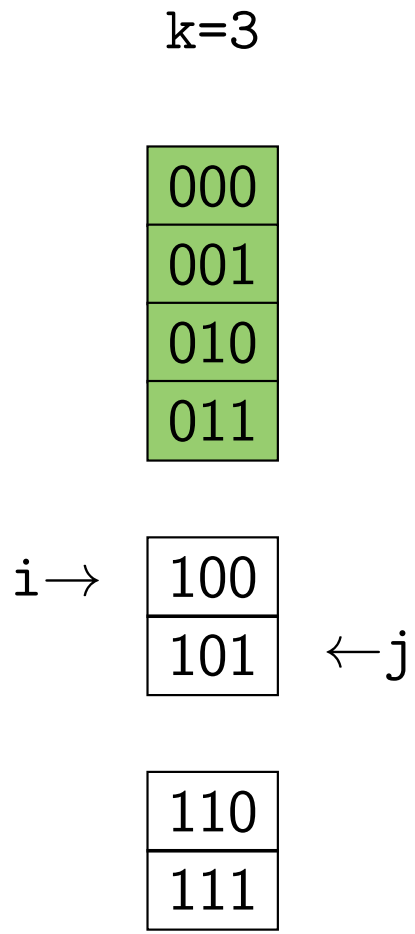
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



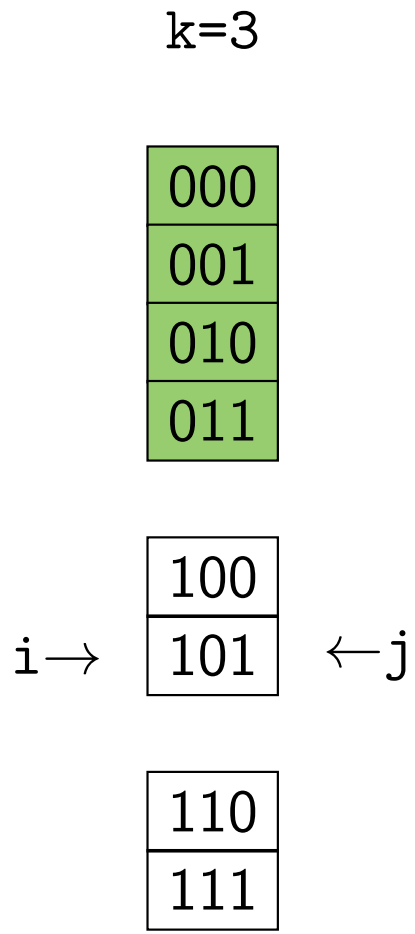
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



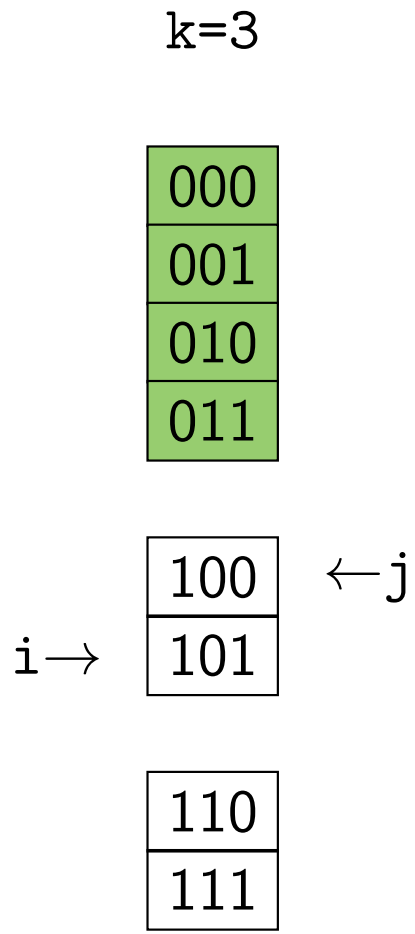
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



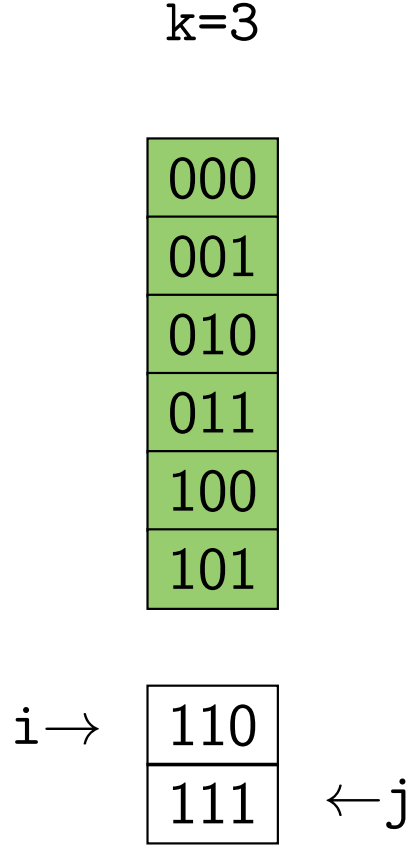
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



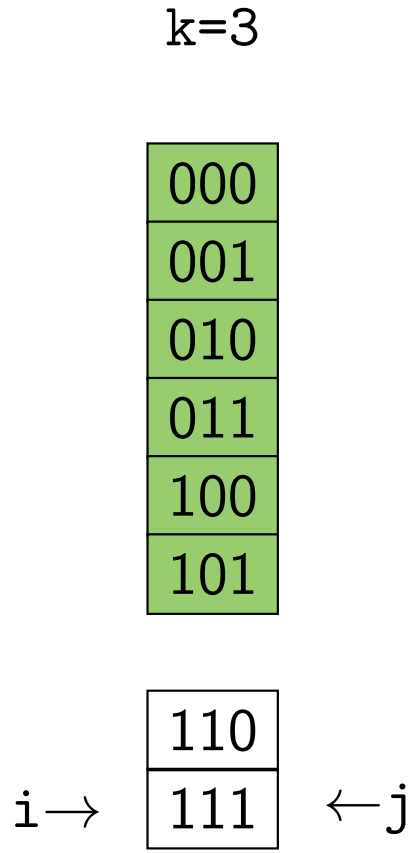
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



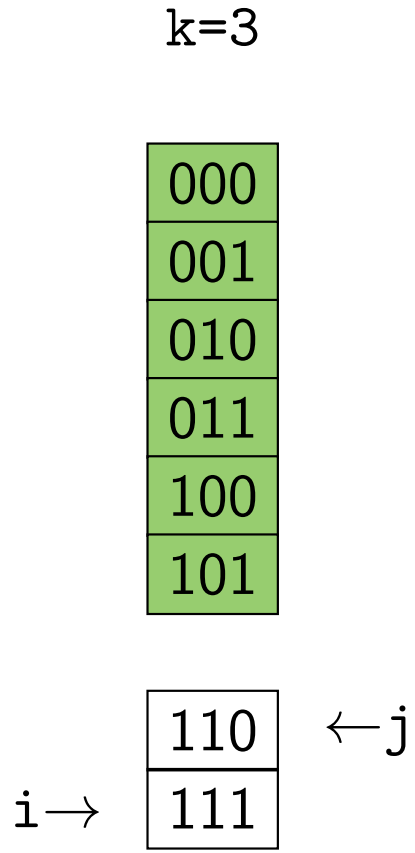
```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```



```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```

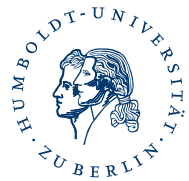


```
1. func radixESort(S array;
2.                 k,l,r: integer) {
3.   if k>m then
4.     return;
5.   end if;
6.   d := divide*(S, k, l, r);
7.   radixESort(S, k+1, l, d);
8.   radixESort(S, k+1, d+1, r);
9. }
```

```
1. func int divide*(S array;
2.                  k,l,r: int) {
3.   i := l-1;
4.   j := r+1;
5.   while true
6.     repeat
7.       i := i+1;
8.     until S[i][k]=1 or i≥j;
9.     repeat
10.      j := j-1;
11.    until S[j][k]=0 or i≥j;
12.    if S[j][k]=1 then j--;
13.    if i≥j then
14.      break while;
15.    end if;
16.    swap( S[i], S[j]);
17.  end while;
18.  return j;
19.}
```

k=3

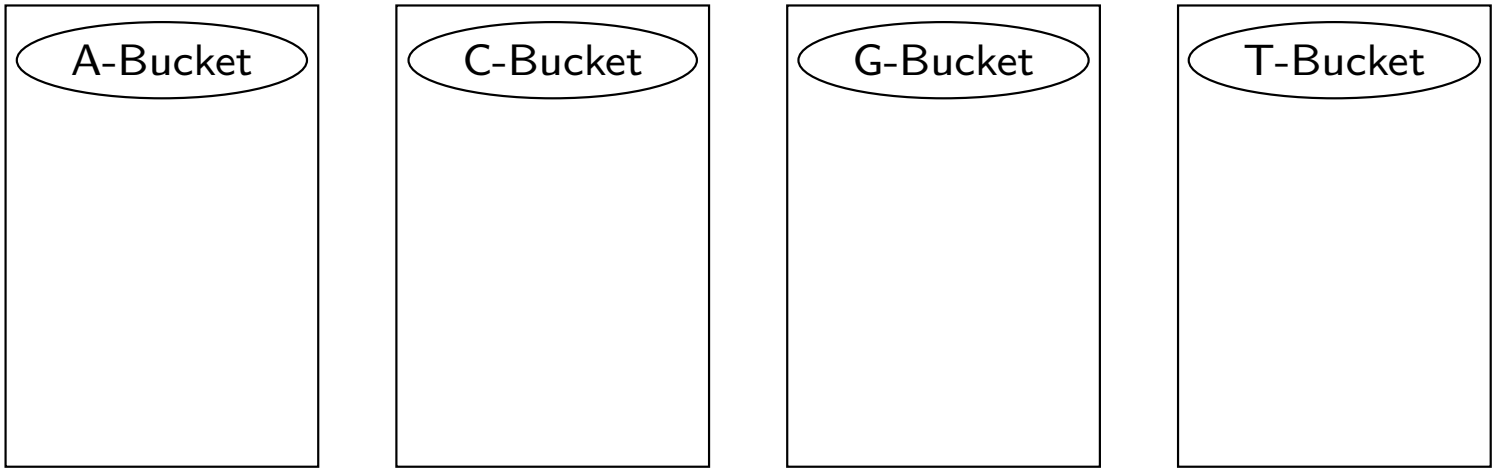
000
001
010
011
100
101
110
111



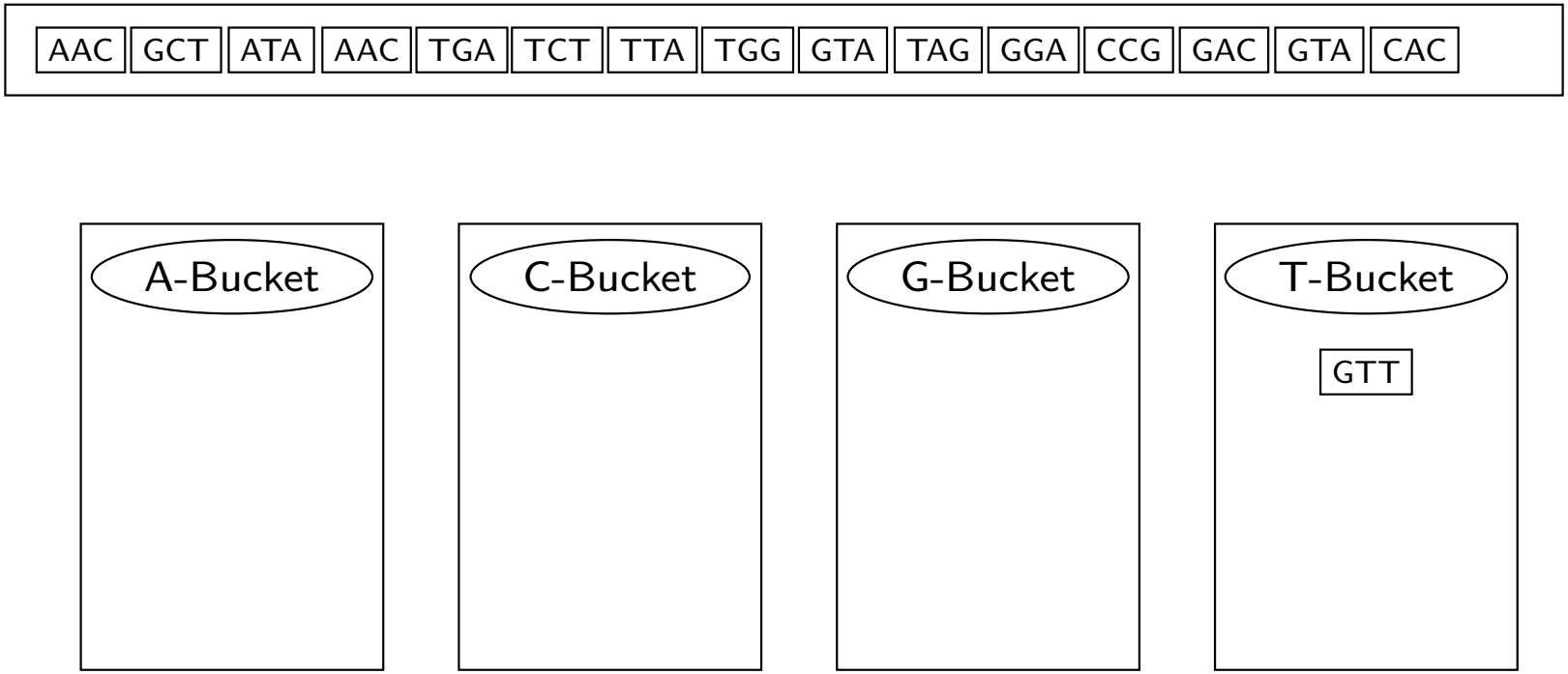
An folgender Beispielinstanz wird Bucketsort veranschaulicht:

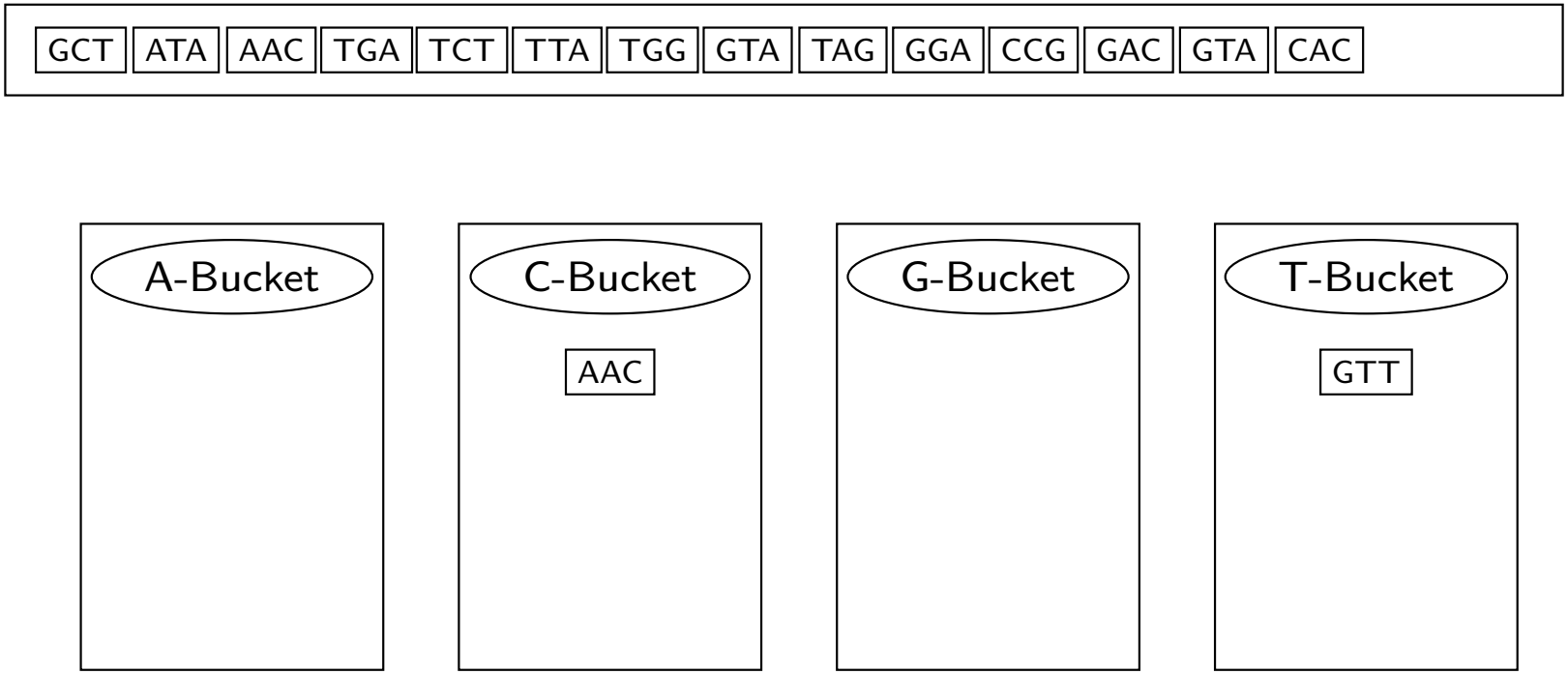
GTT	AAC	GCT	ATA	AAC	TGA	TCT	TTA	TGG	GTA	TAG	GGA	CCG	GAC	GTA	CAC

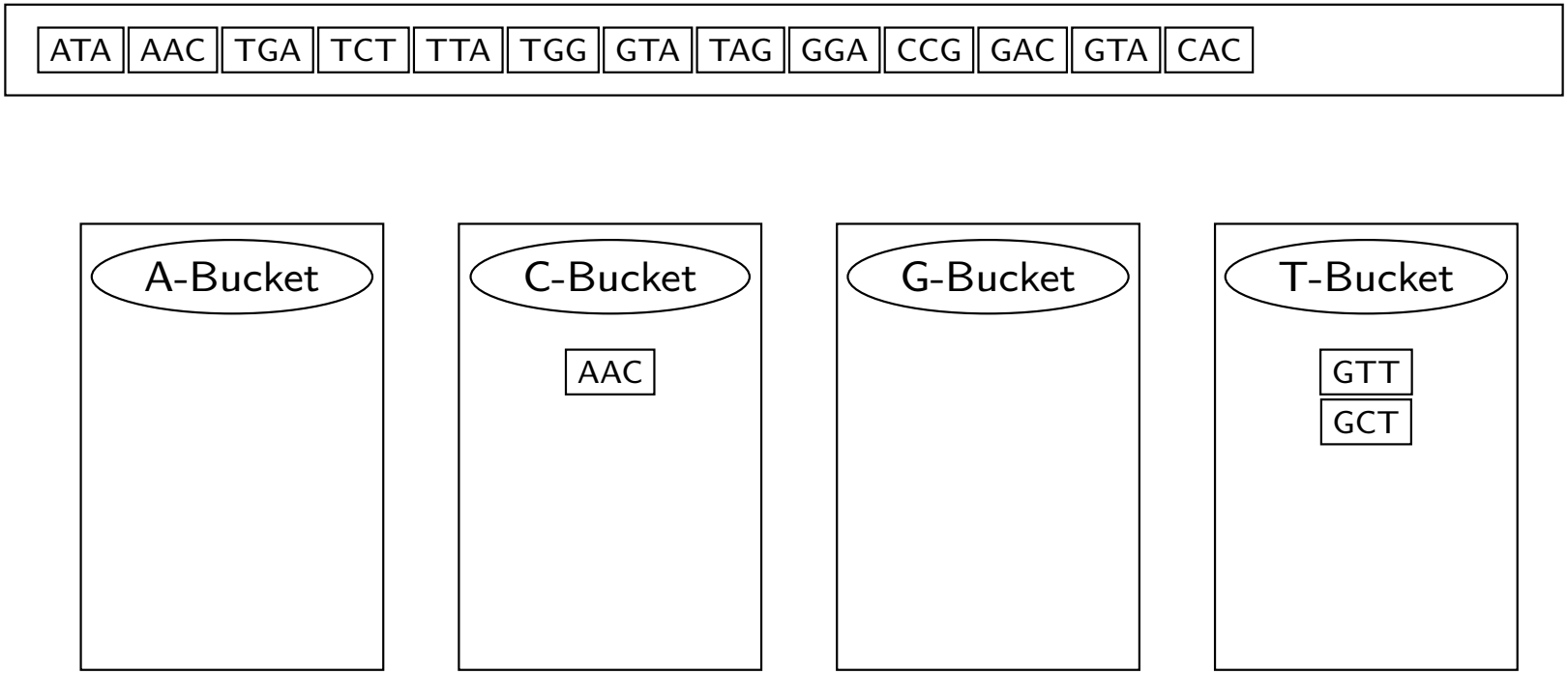

GTT
AAC
GCT
ATA
AAC
TGA
TCT
TTA
TGG
GTA
TAG
GGA
CCG
GAC
GTA
CAC

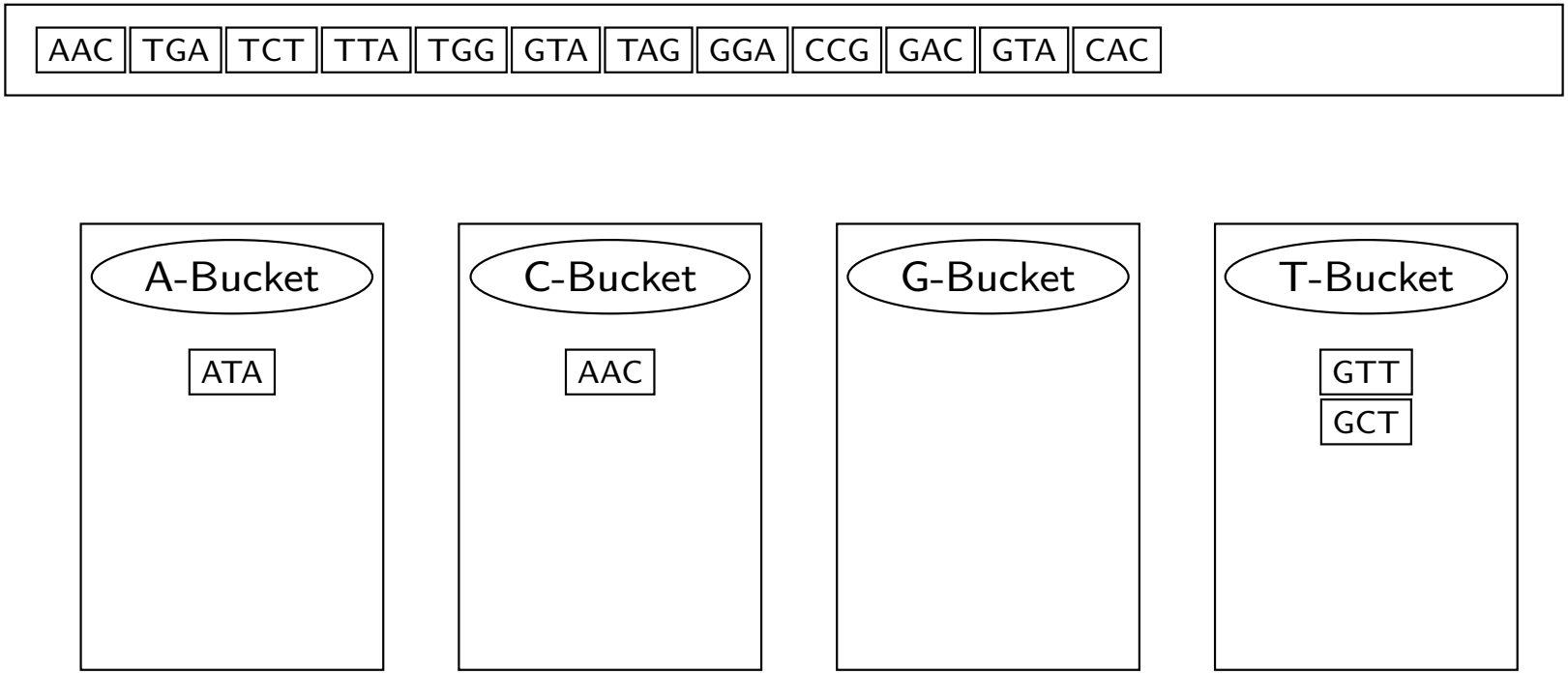

A-Bucket
C-Bucket
G-Bucket
T-Bucket

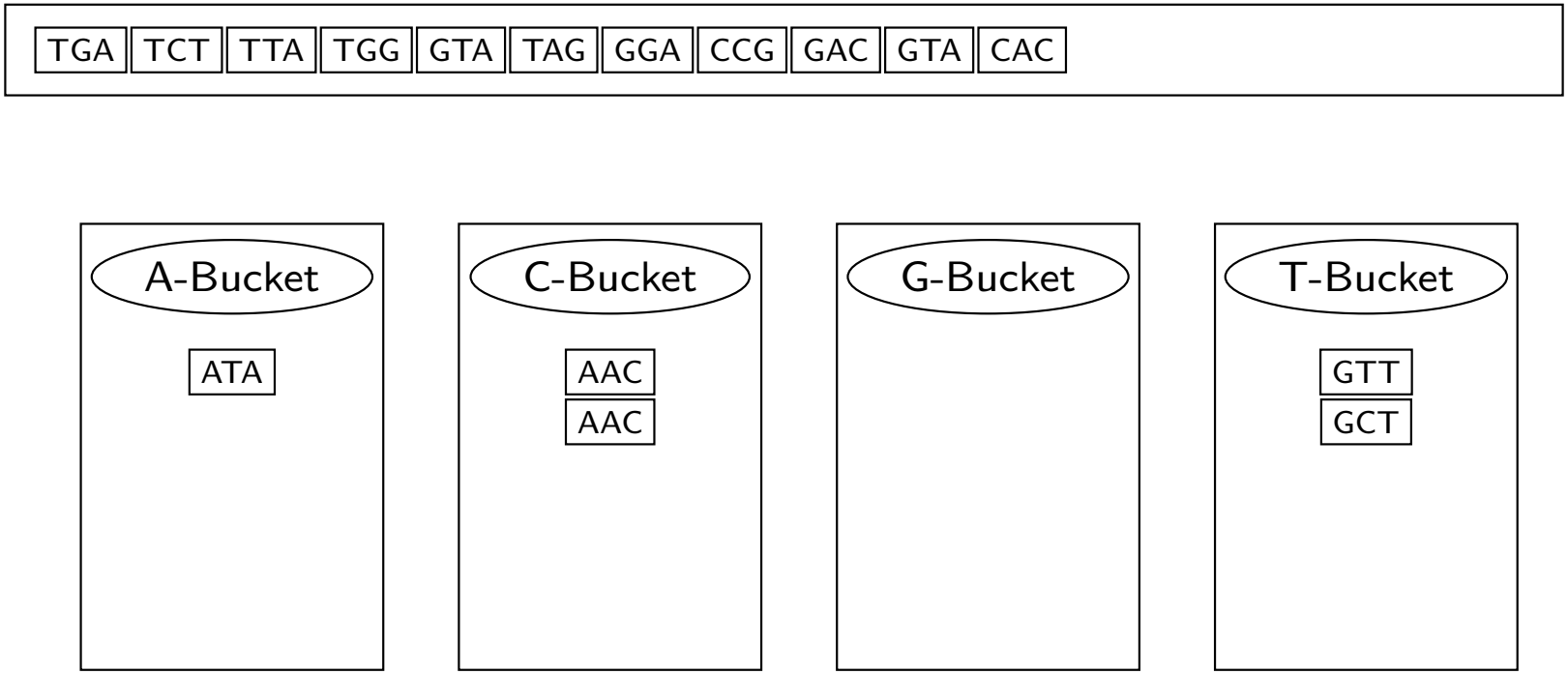

HUMBOLDT-UNIVERSITÄT ZU BERLIN

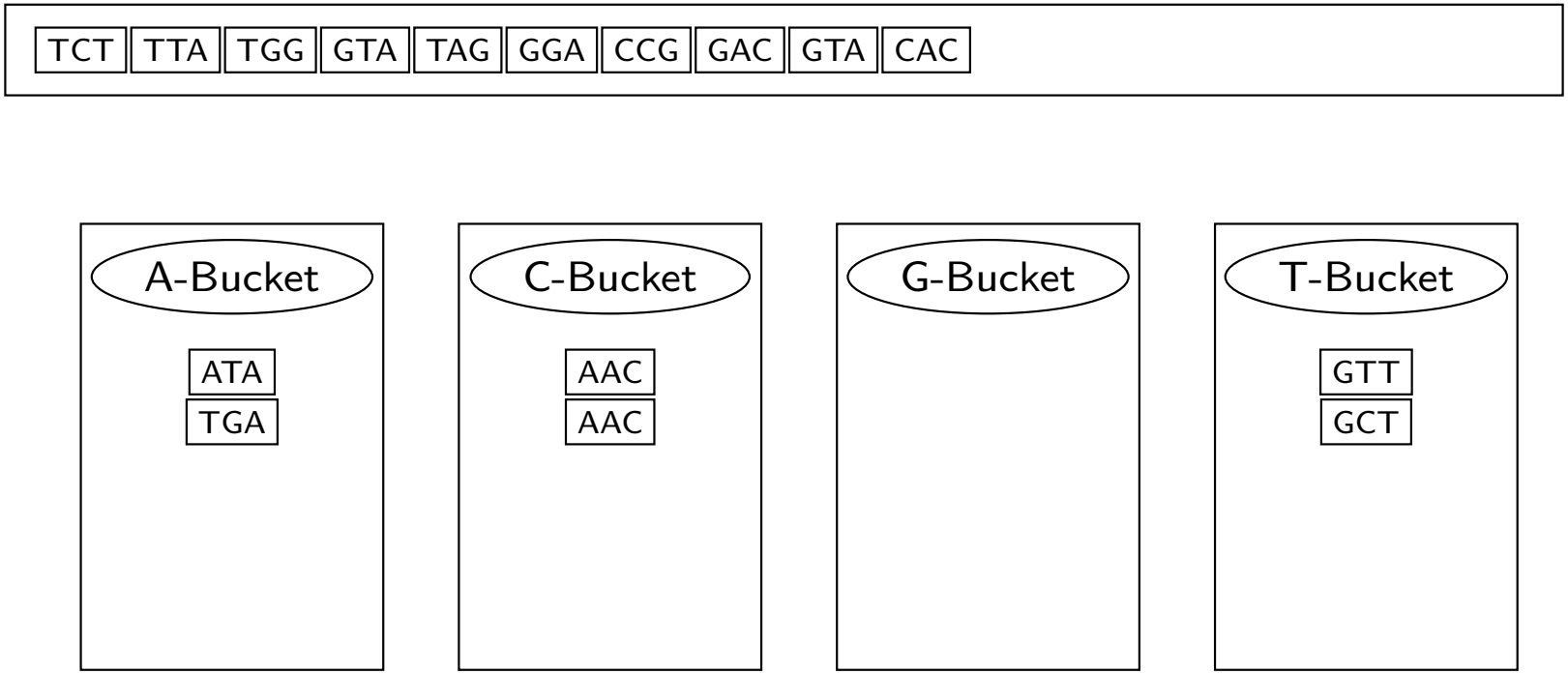

AAC
GCT
ATA
AAC
TGA
TCT
TTA
TGG
GTA
TAG
GGA
CCG
GAC
GTA
CAC
A-Bucket
C-Bucket
G-Bucket
T-Bucket
GTT

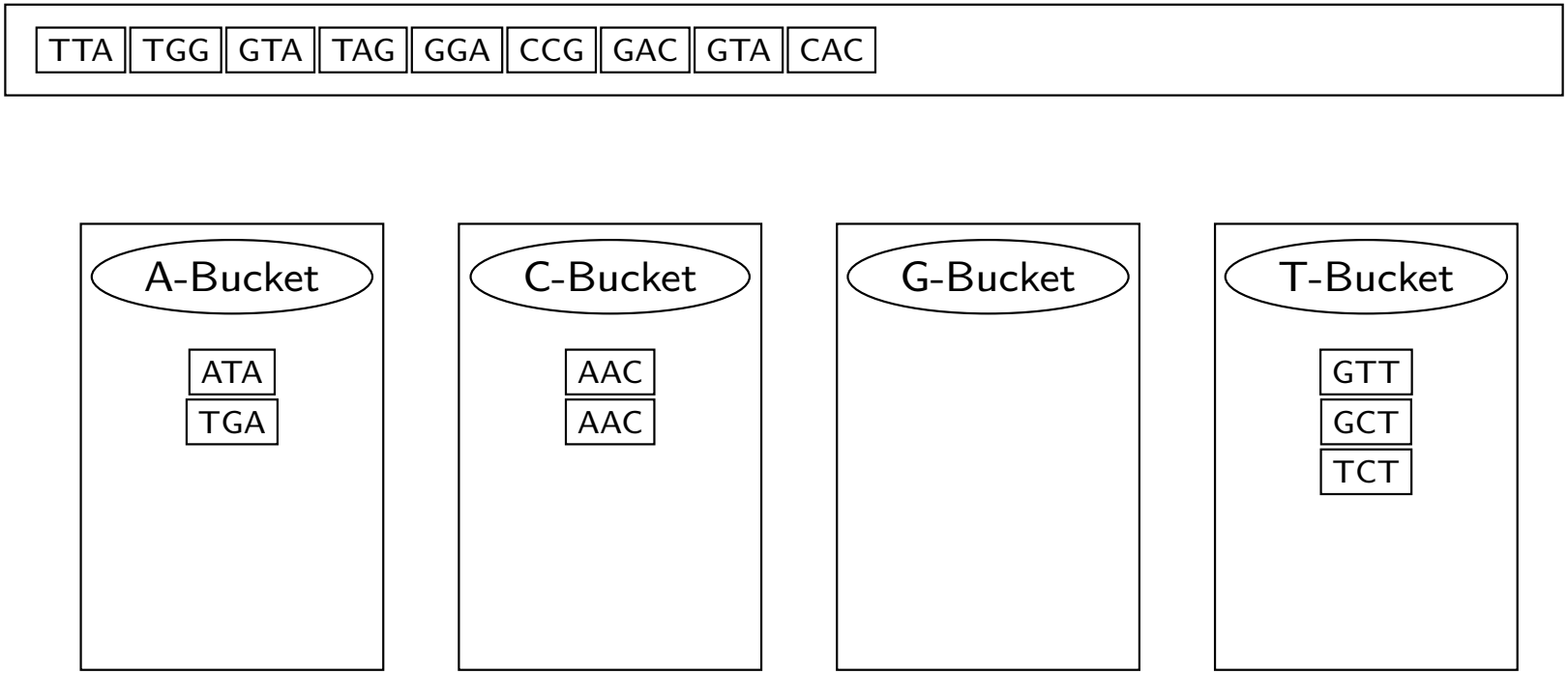

GCT
ATA
AAC
TGA
TCT
TTA
TGG
GTA
TAG
GGA
CCG
GAC
GTA
CAC
A-Bucket
C-Bucket
AAC
G-Bucket
T-Bucket
GTT

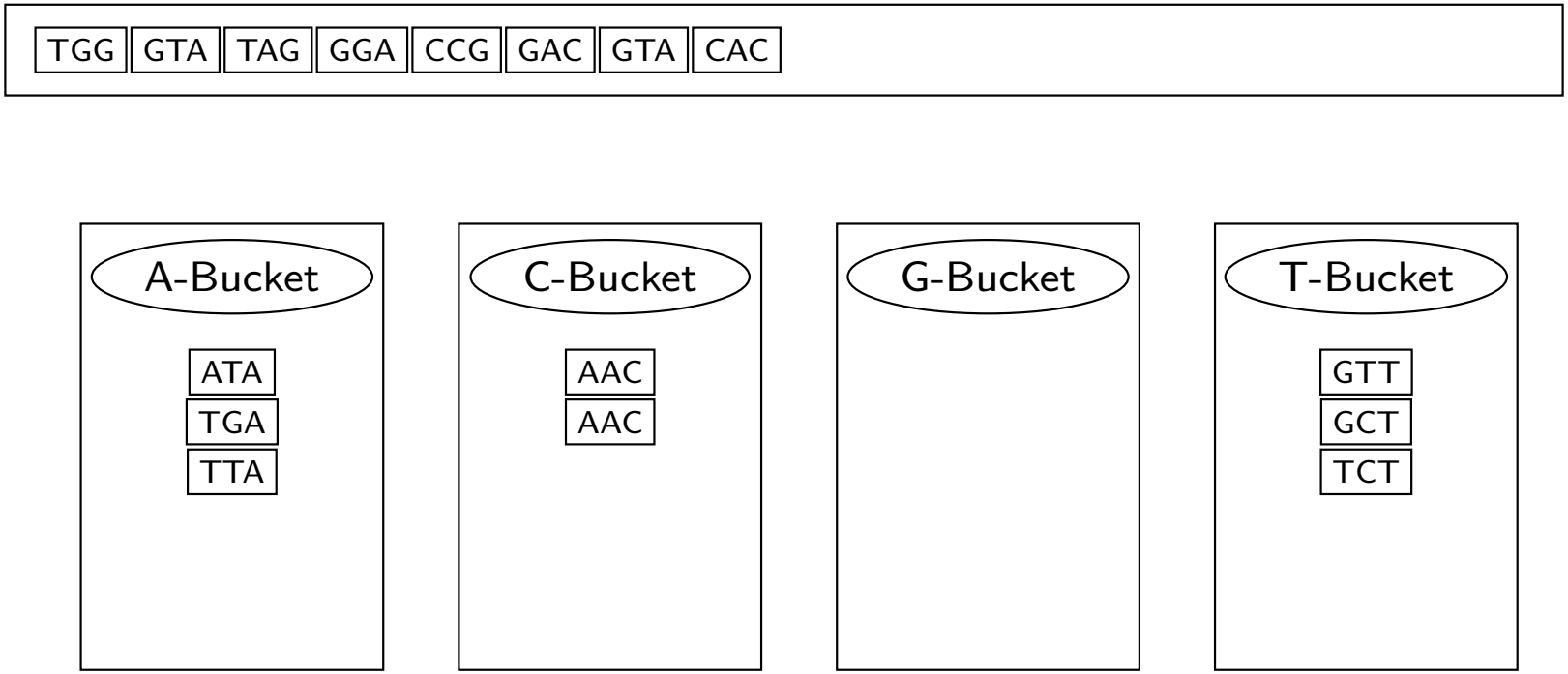

ATA
AAC
TGA
TCT
TTA
TGG
GTA
TAG
GGA
CCG
GAC
GTA
CAC
A-Bucket
C-Bucket
AAC
G-Bucket
T-Bucket
GTT
GCT

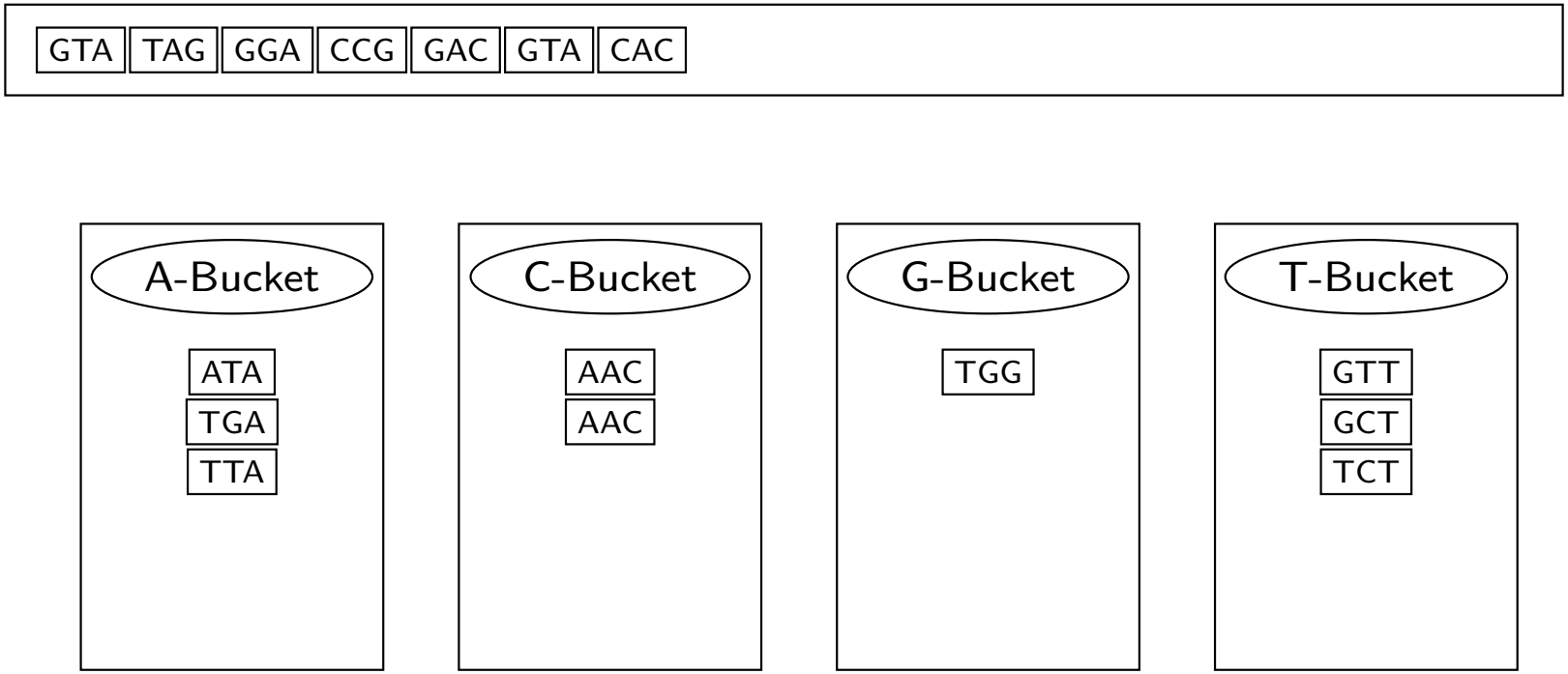

AAC
TGA
TCT
TTA
TGG
GTA
TAG
GGA
CCG
GAC
GTA
CAC
A-Bucket
ATA
C-Bucket
AAC
G-Bucket
T-Bucket
GTT
GCT

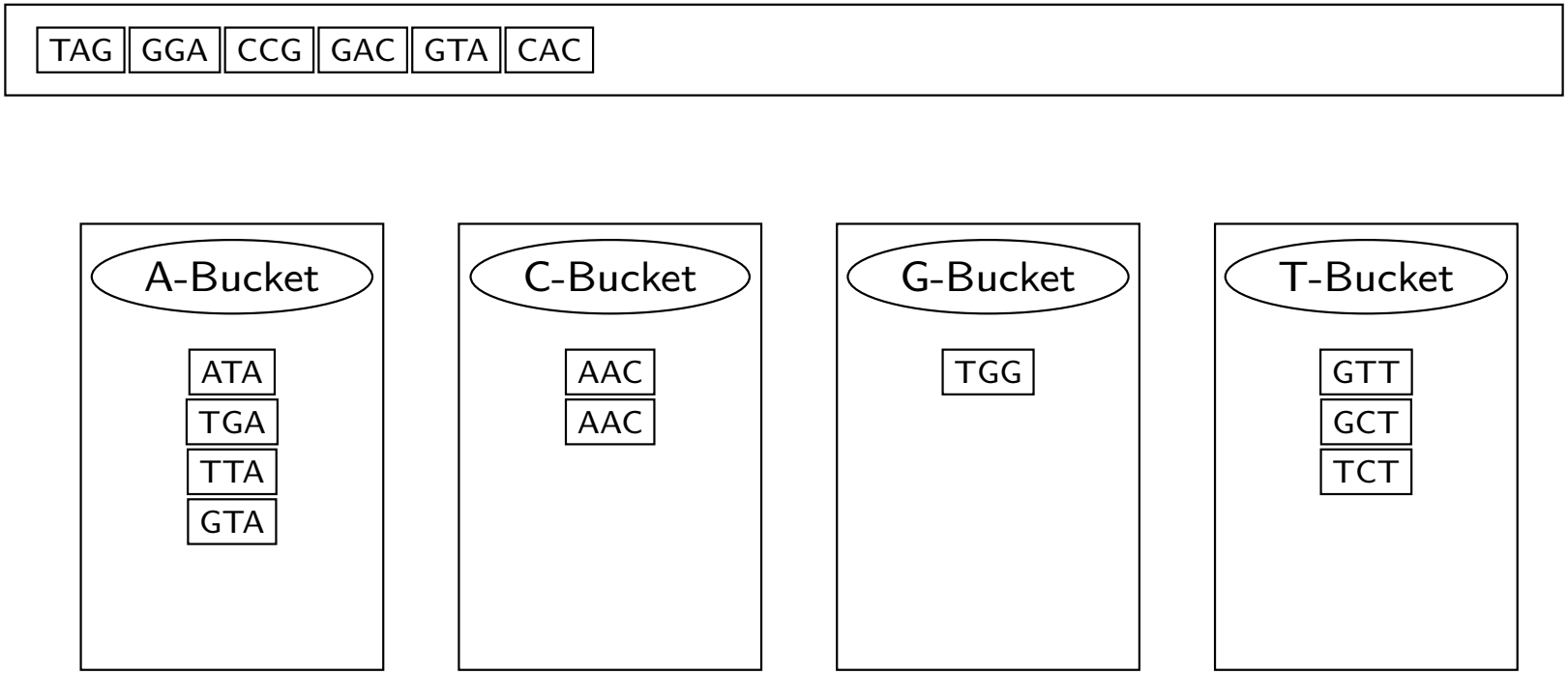

TGA
TCT
TTA
TGG
GTA
TAG
GGA
CCG
GAC
GTA
CAC
A-Bucket
ATA
C-Bucket
AAC
AAC
G-Bucket
T-Bucket
GTT
GCT

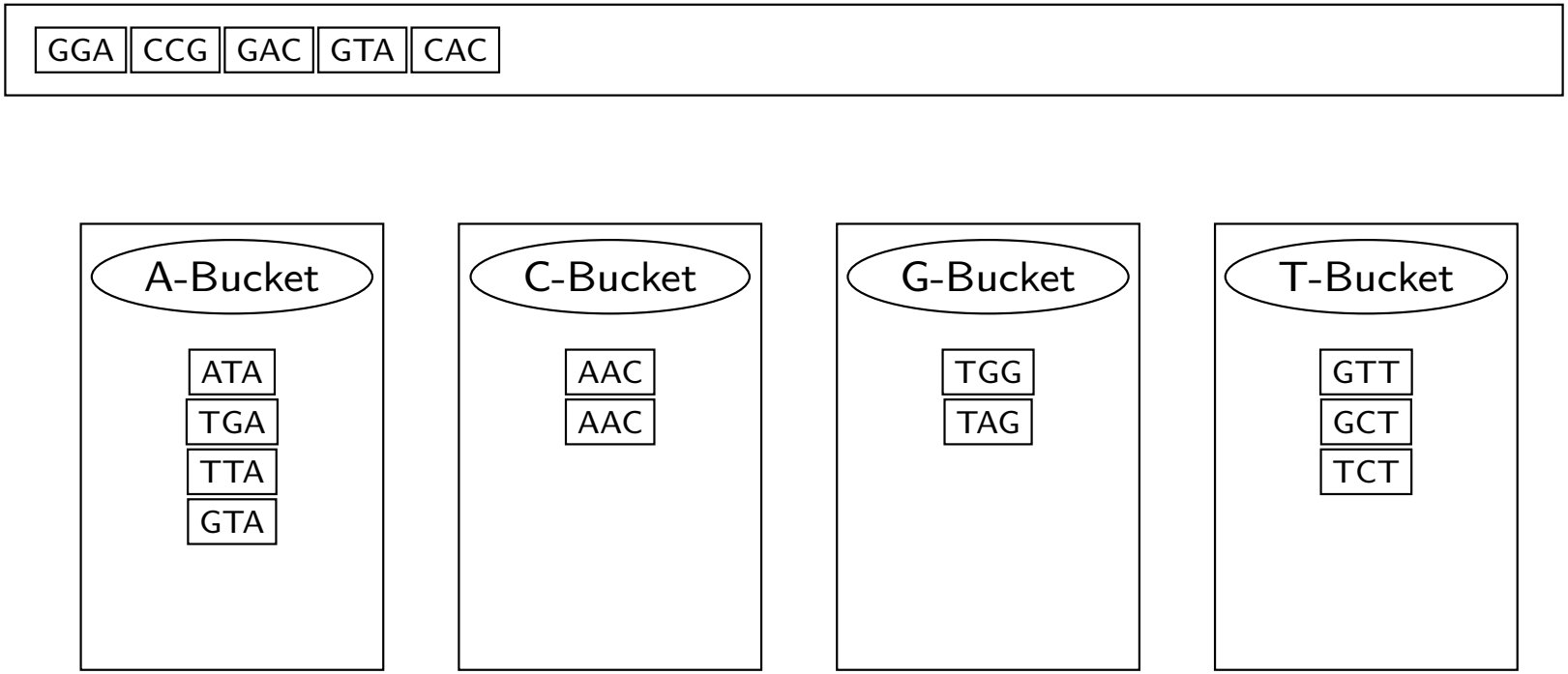

TCT
TTA
TGG
GTA
TAG
GGA
CCG
GAC
GTA
CAC
A-Bucket
ATA
TGA
C-Bucket
AAC
AAC
G-Bucket
T-Bucket
GTT
GCT

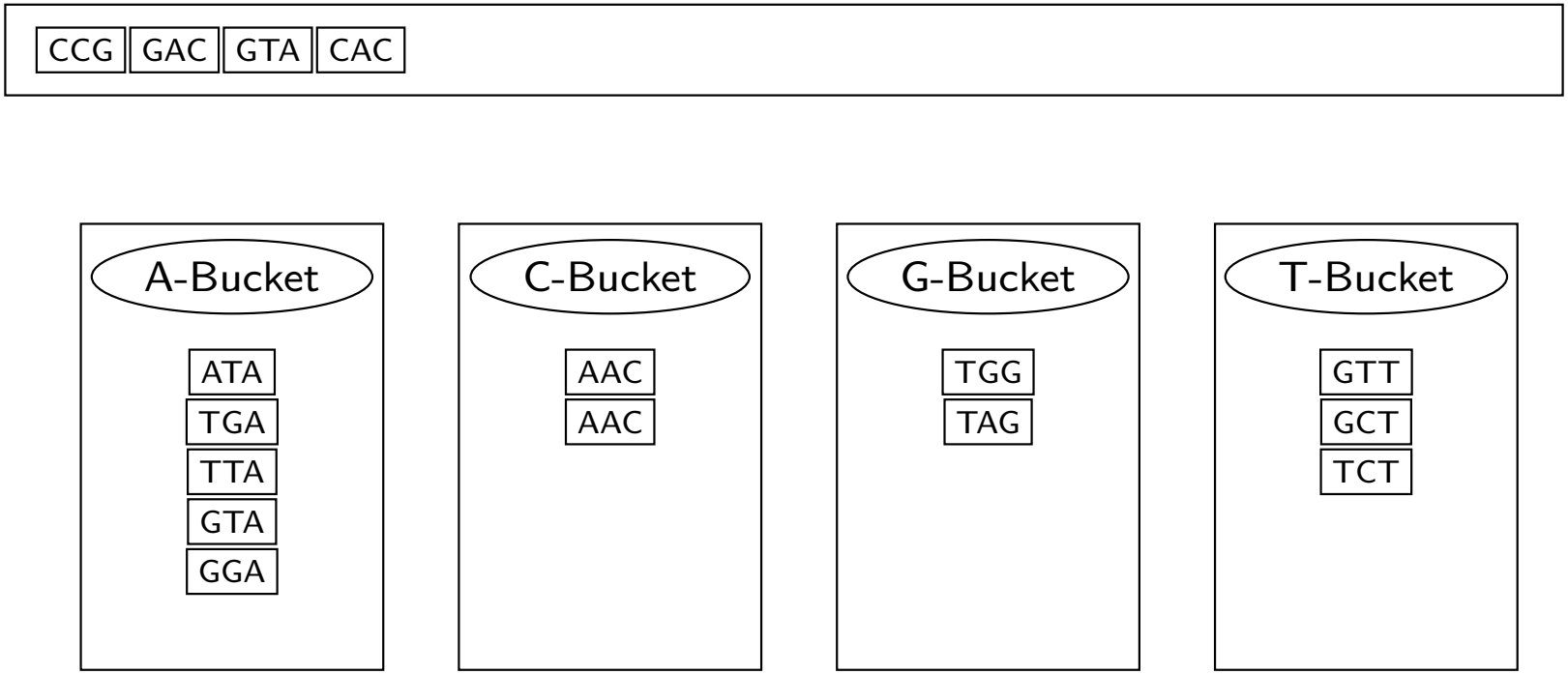

TTA
TGG
GTA
TAG
GGA
CCG
GAC
GTA
CAC
A-Bucket
ATA
TGA
C-Bucket
AAC
AAC
G-Bucket
T-Bucket
GTT
GCT
TCT

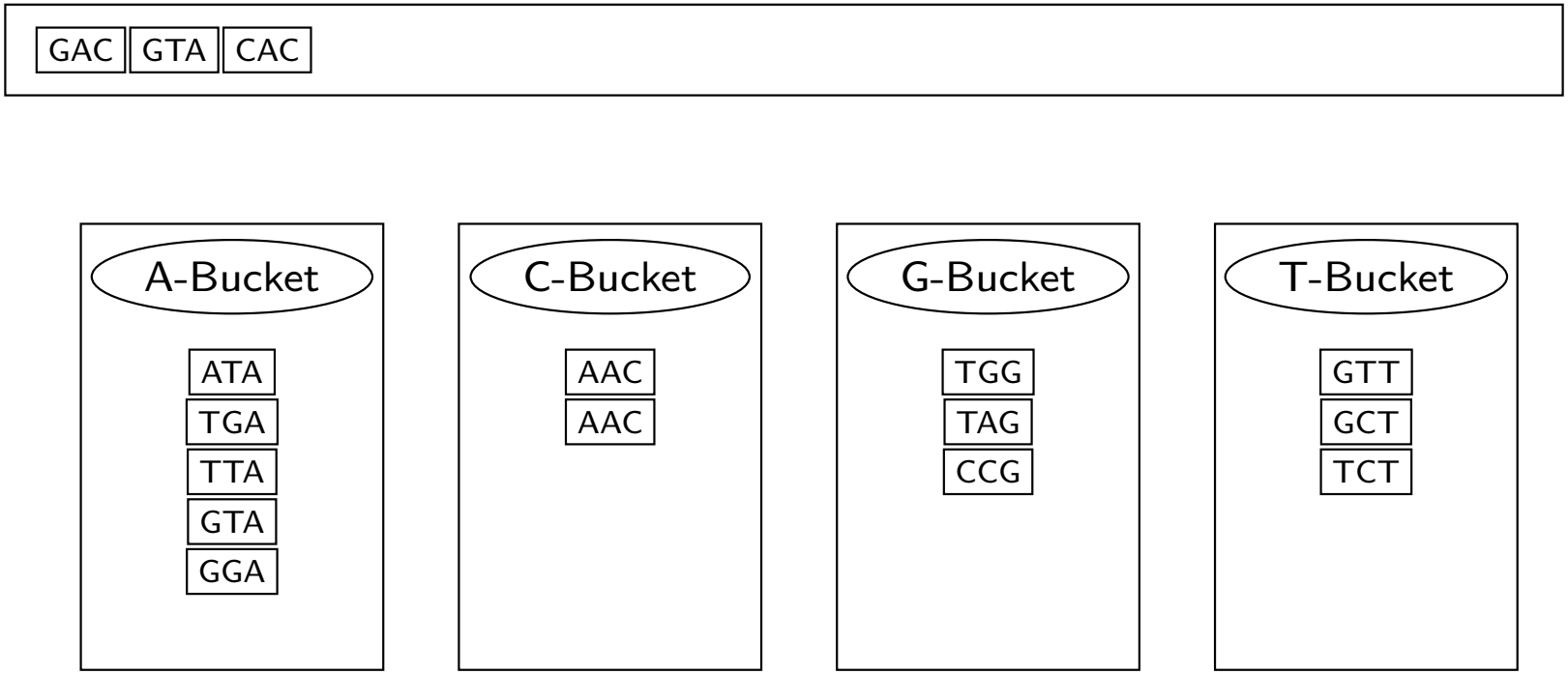

TGG
GTA
TAG
GGA
CCG
GAC
GTA
CAC
A-Bucket
ATA
TGA
TTA
C-Bucket
AAC
AAC
G-Bucket
T-Bucket
GTT
GCT
TCT

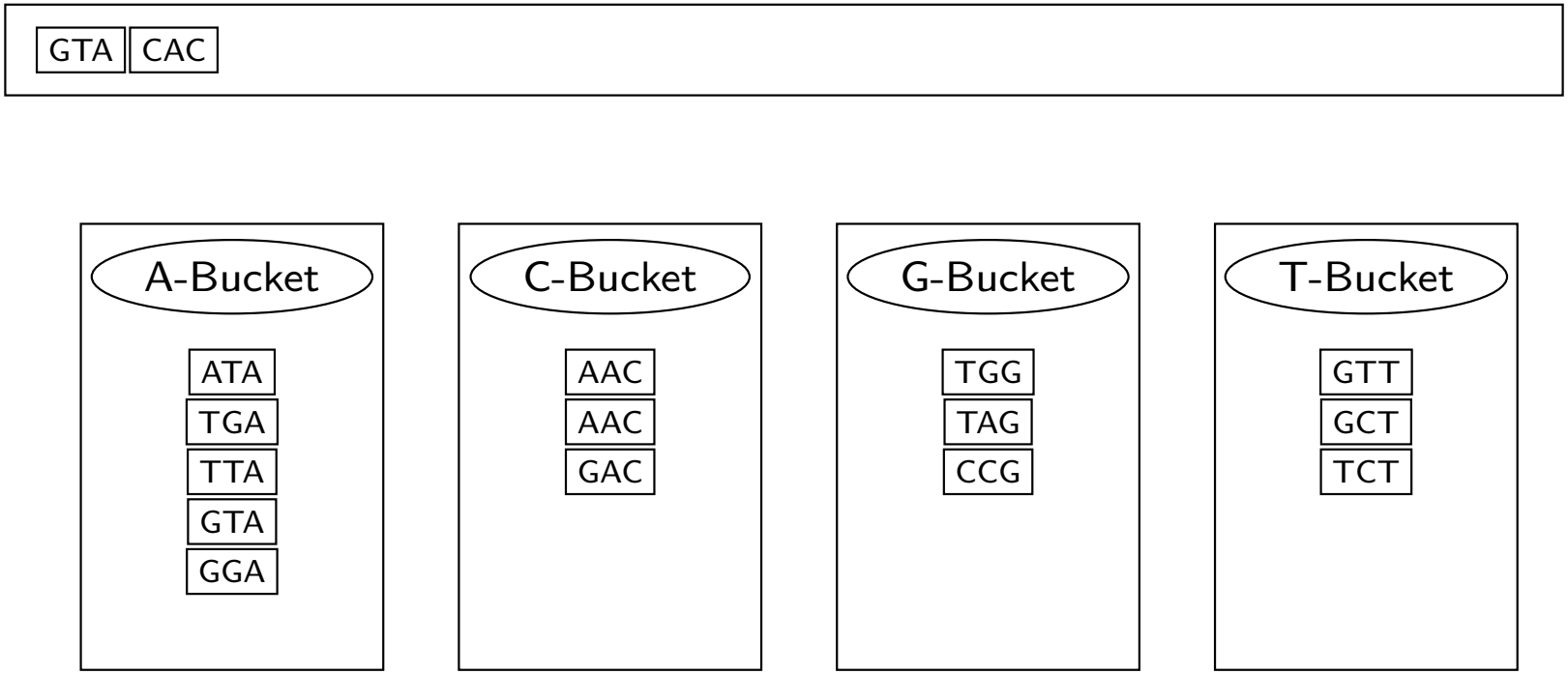

GTA
TAG
GGA
CCG
GAC
GTA
CAC
A-Bucket
ATA
TGA
TTA
C-Bucket
AAC
AAC
G-Bucket
TGG
T-Bucket
GTT
GCT
TCT

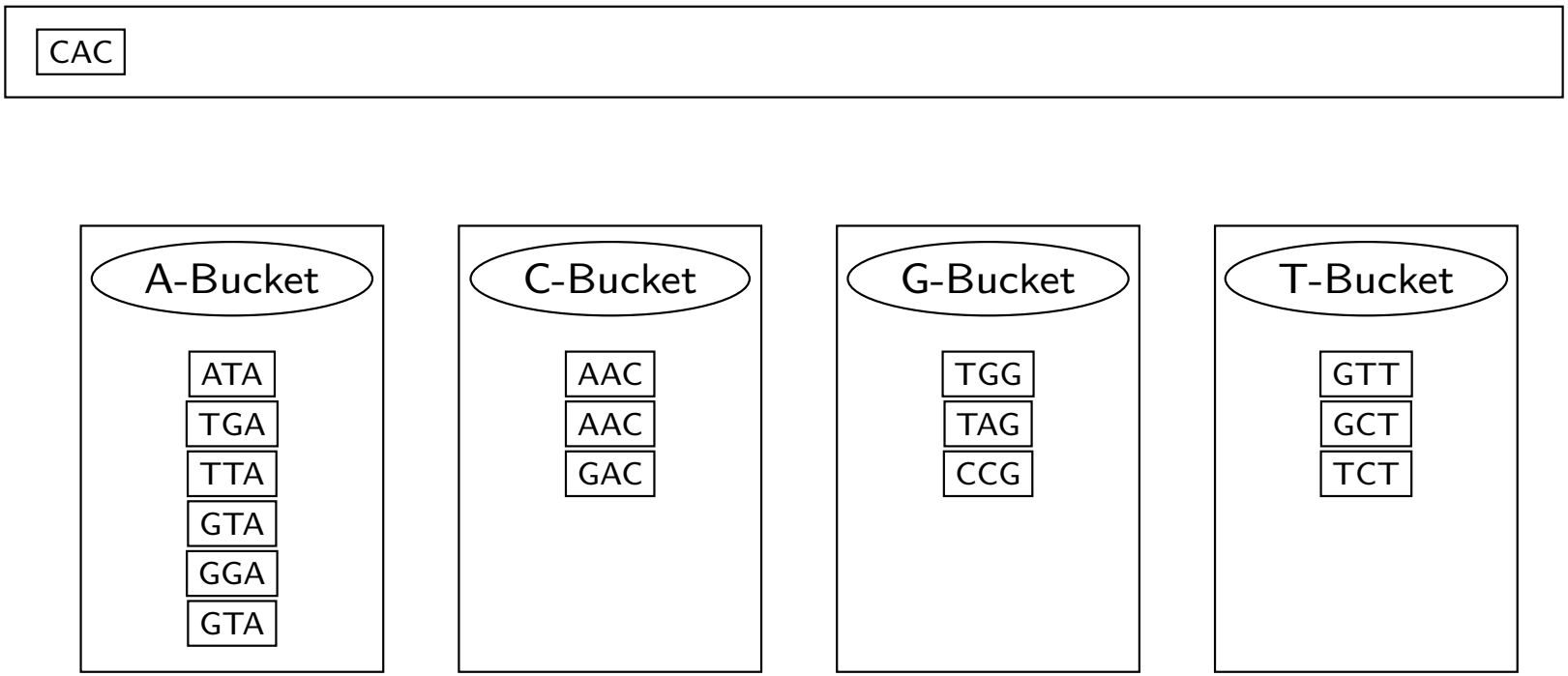

TAG
GGA
CCG
GAC
GTA
CAC
A-Bucket
ATA
TGA
TTA
GTA
C-Bucket
AAC
AAC
G-Bucket
TGG
T-Bucket
GTT
GCT
TCT

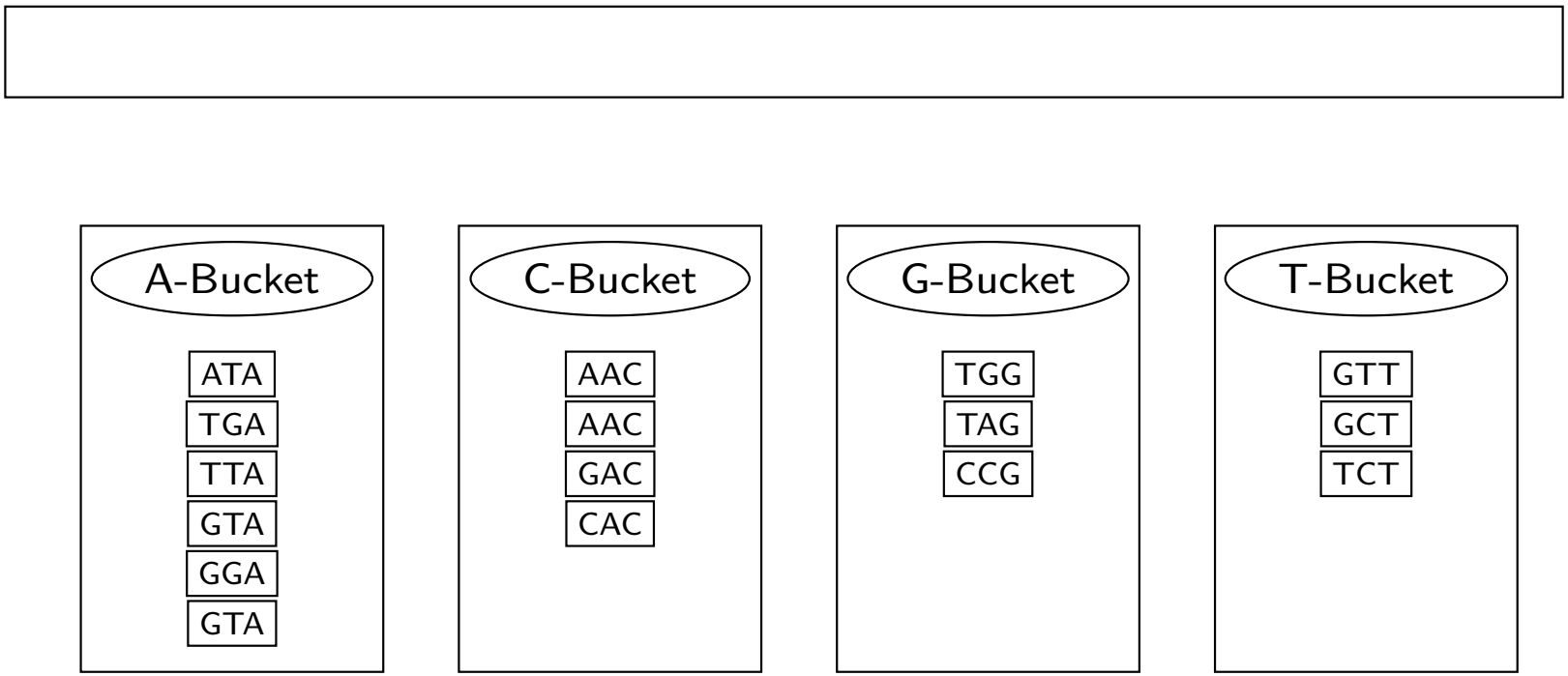

GGA
CCG
GAC
GTA
CAC
A-Bucket
ATA
TGA
TTA
GTA
C-Bucket
AAC
AAC
G-Bucket
TGG
TAG
T-Bucket
GTT
GCT
TCT

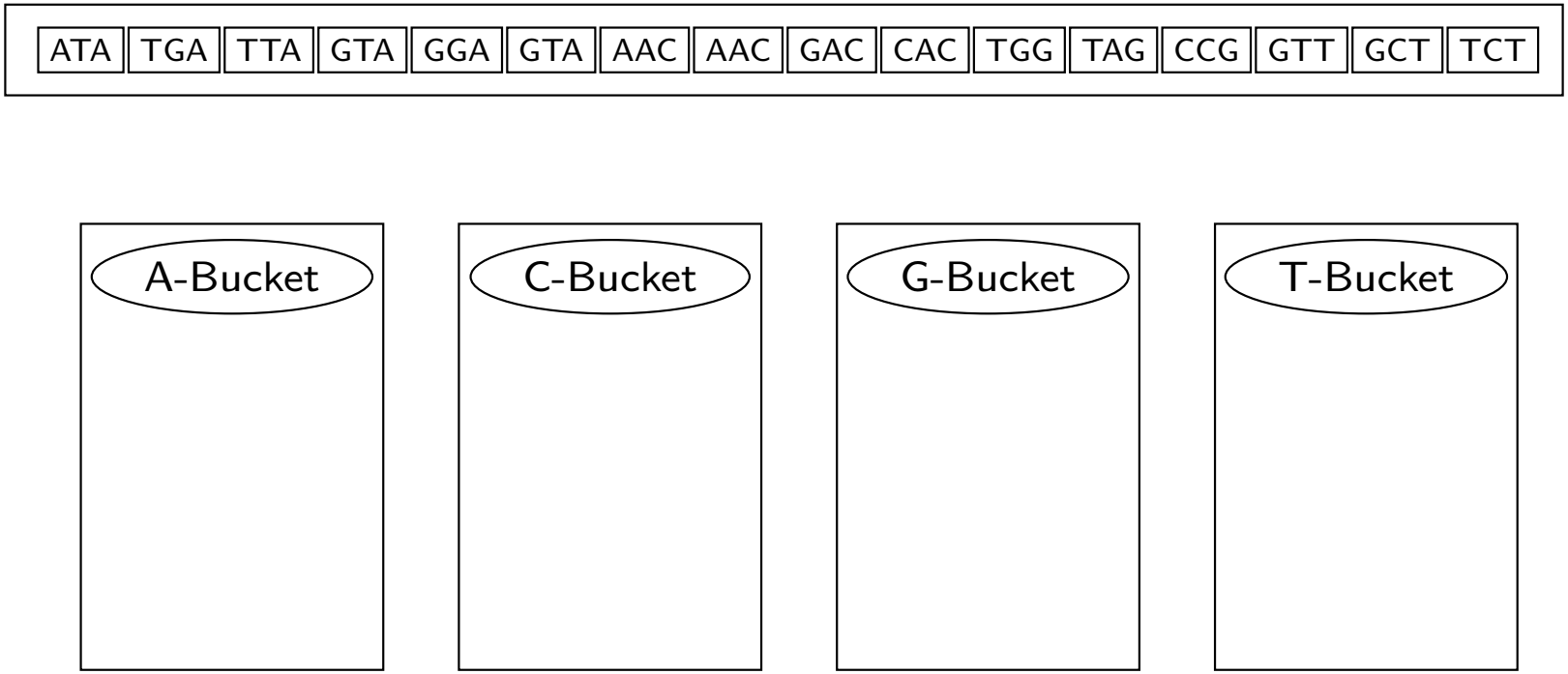

CCG
GAC
GTA
CAC
A-Bucket
ATA
TGA
TTA
GTA
GGA
C-Bucket
AAC
AAC
G-Bucket
TGG
TAG
T-Bucket
GTT
GCT
TCT

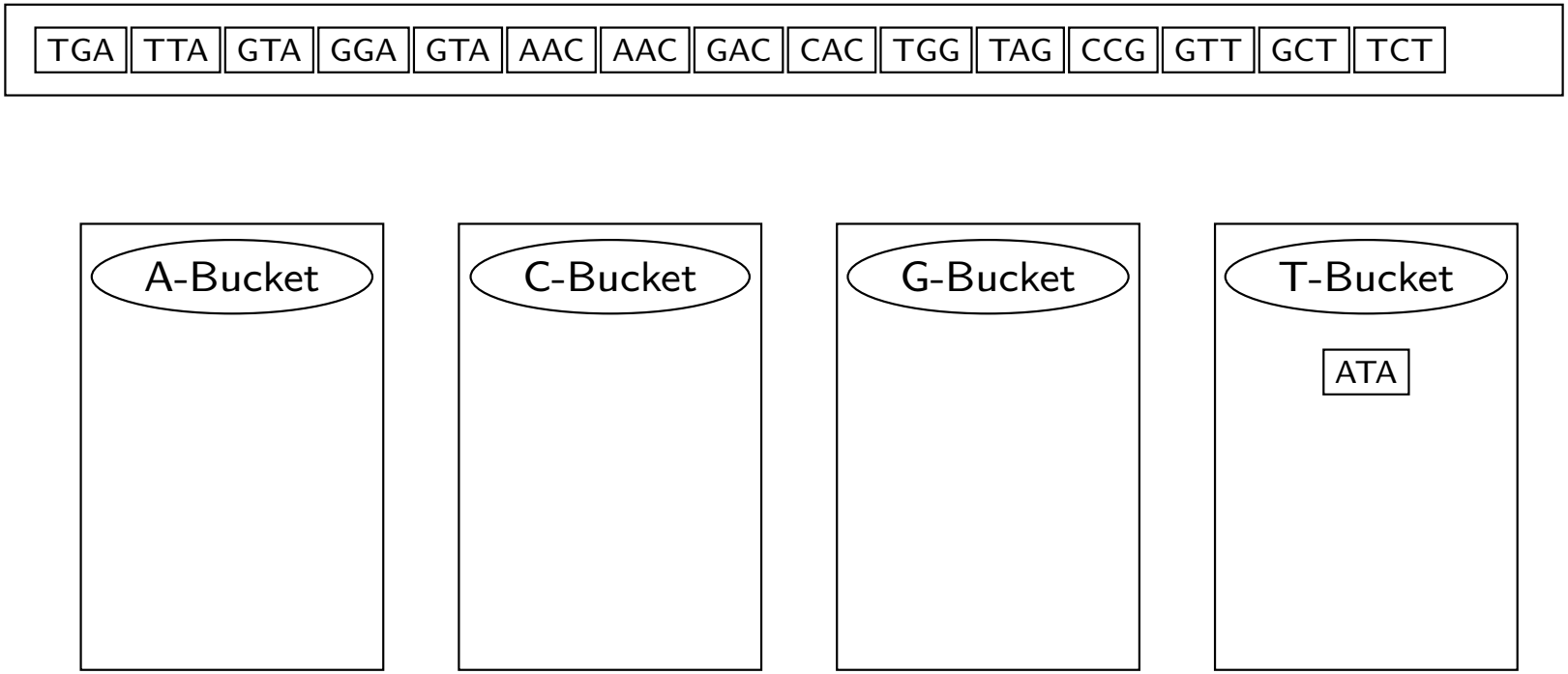

GAC
GTA
CAC
A-Bucket
ATA
TGA
TTA
GTA
GGA
C-Bucket
AAC
AAC
G-Bucket
TGG
TAG
CCG
T-Bucket
GTT
GCT
TCT

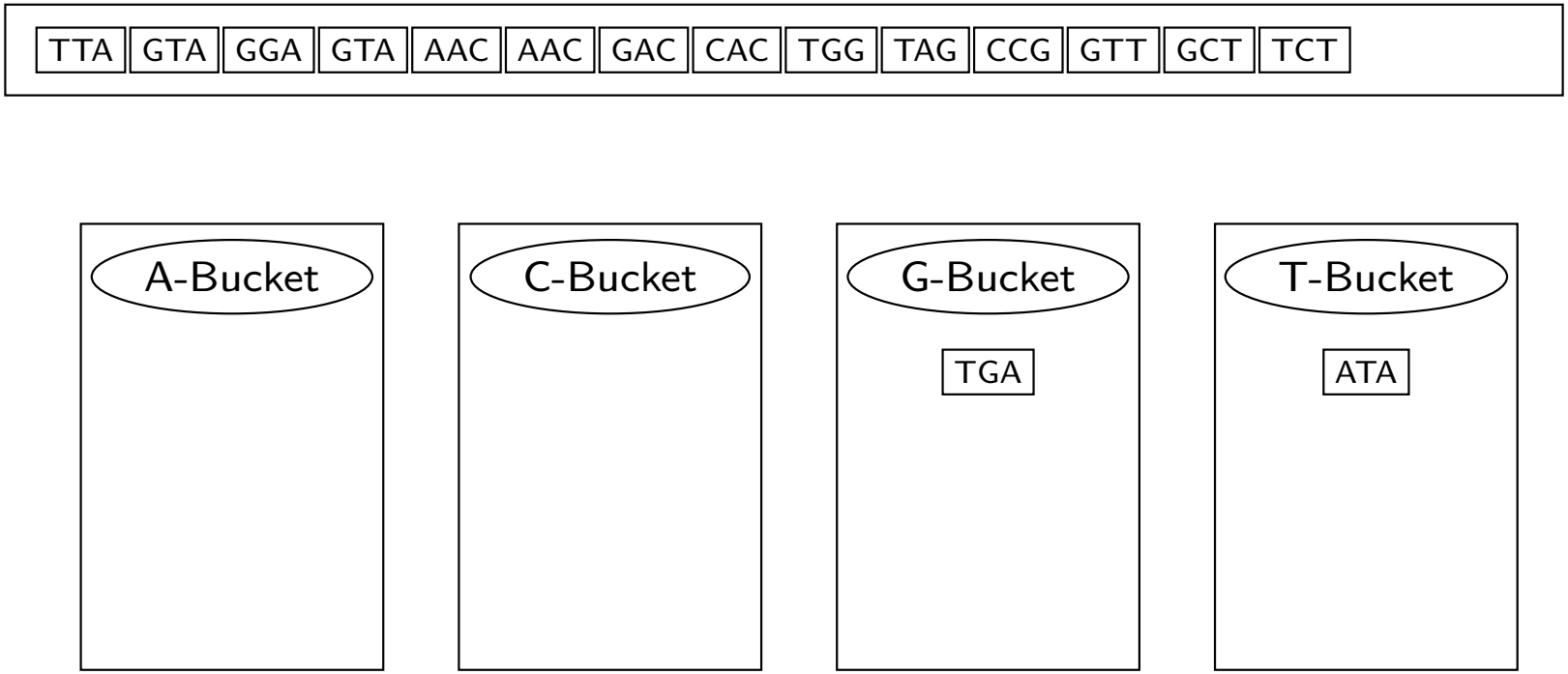

GTA
CAC
A-Bucket
ATA
TGA
TTA
GTA
GGA
C-Bucket
AAC
AAC
GAC
G-Bucket
TGG
TAG
CCG
T-Bucket
GTT
GCT
TCT

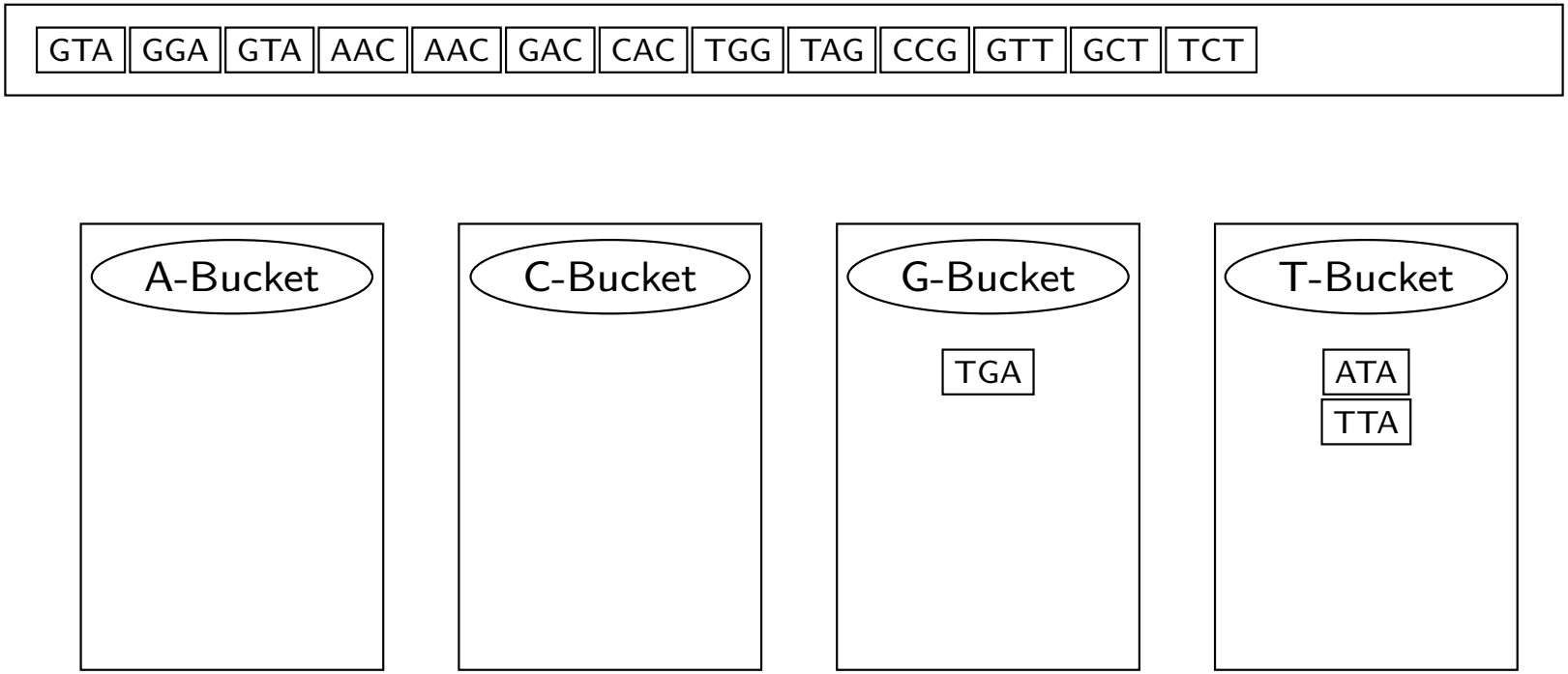

CAC
A-Bucket
ATA
TGA
TTA
GTA
GGA
GTA
C-Bucket
AAC
AAC
GAC
G-Bucket
TGG
TAG
CCG
T-Bucket
GTT
GCT
TCT

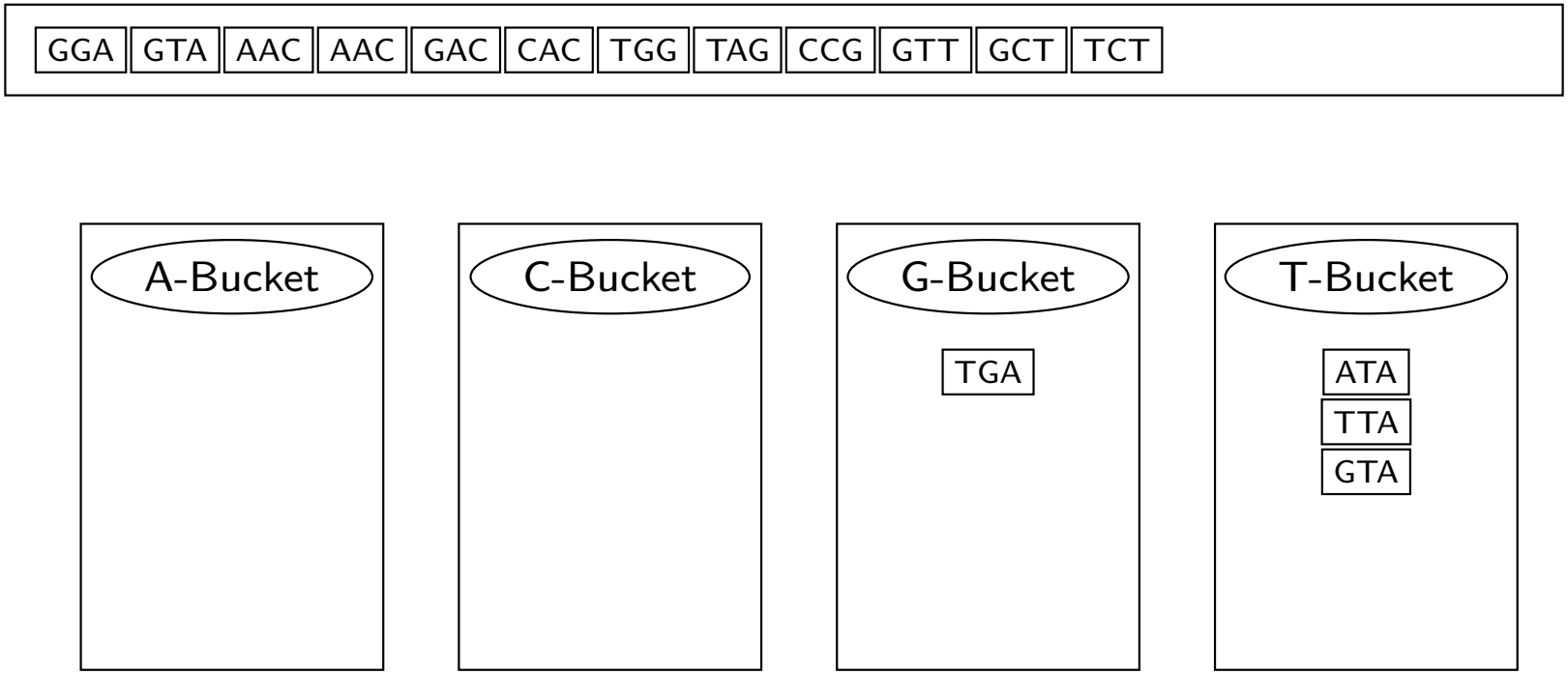

A-Bucket
ATA
TGA
TTA
GTA
GGA
GTA
C-Bucket
AAC
AAC
GAC
CAC
G-Bucket
TGG
TAG
CCG
T-Bucket
GTT
GCT
TCT

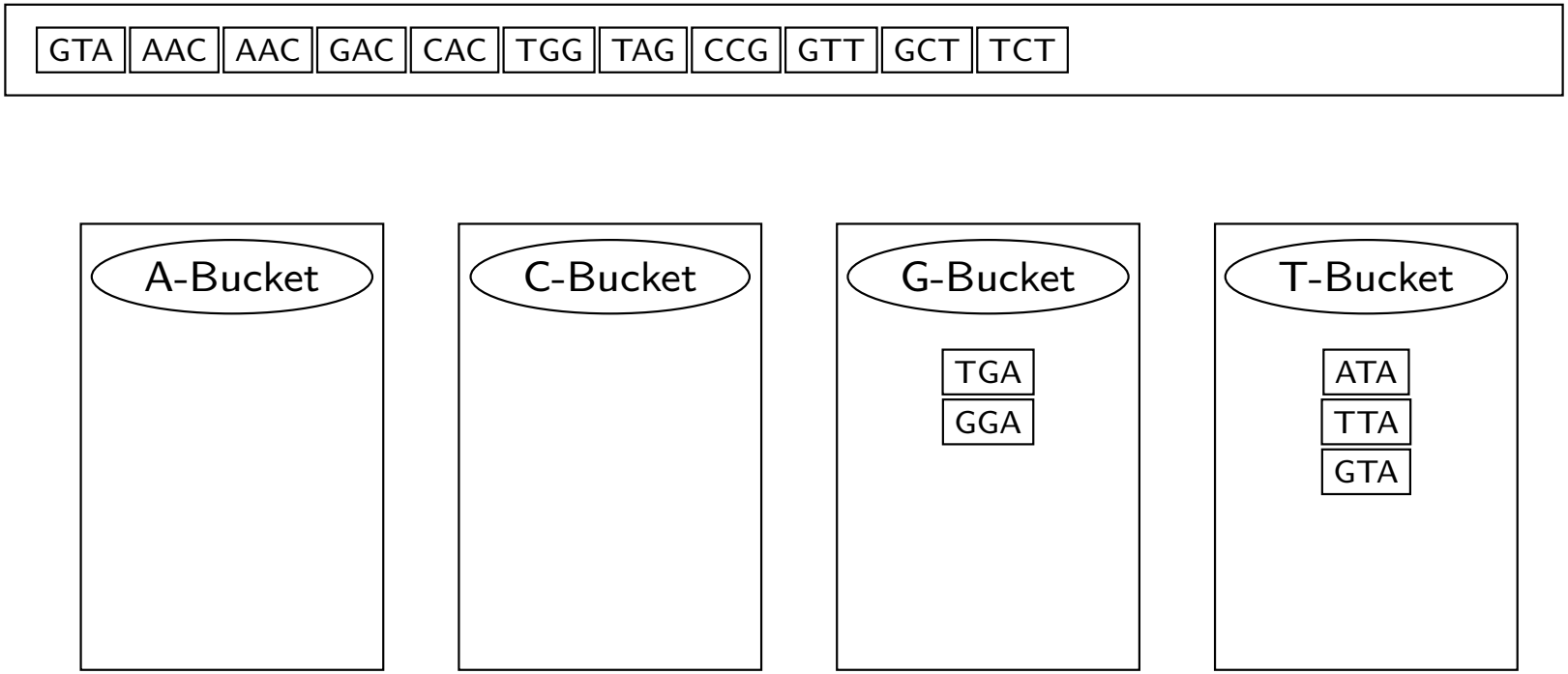

ATA
TGA
TTA
GTA
GGA
GTA
AAC
AAC
GAC
CAC
TGG
TAG
CCG
GTT
GCT
TCT
A-Bucket
C-Bucket
G-Bucket
T-Bucket

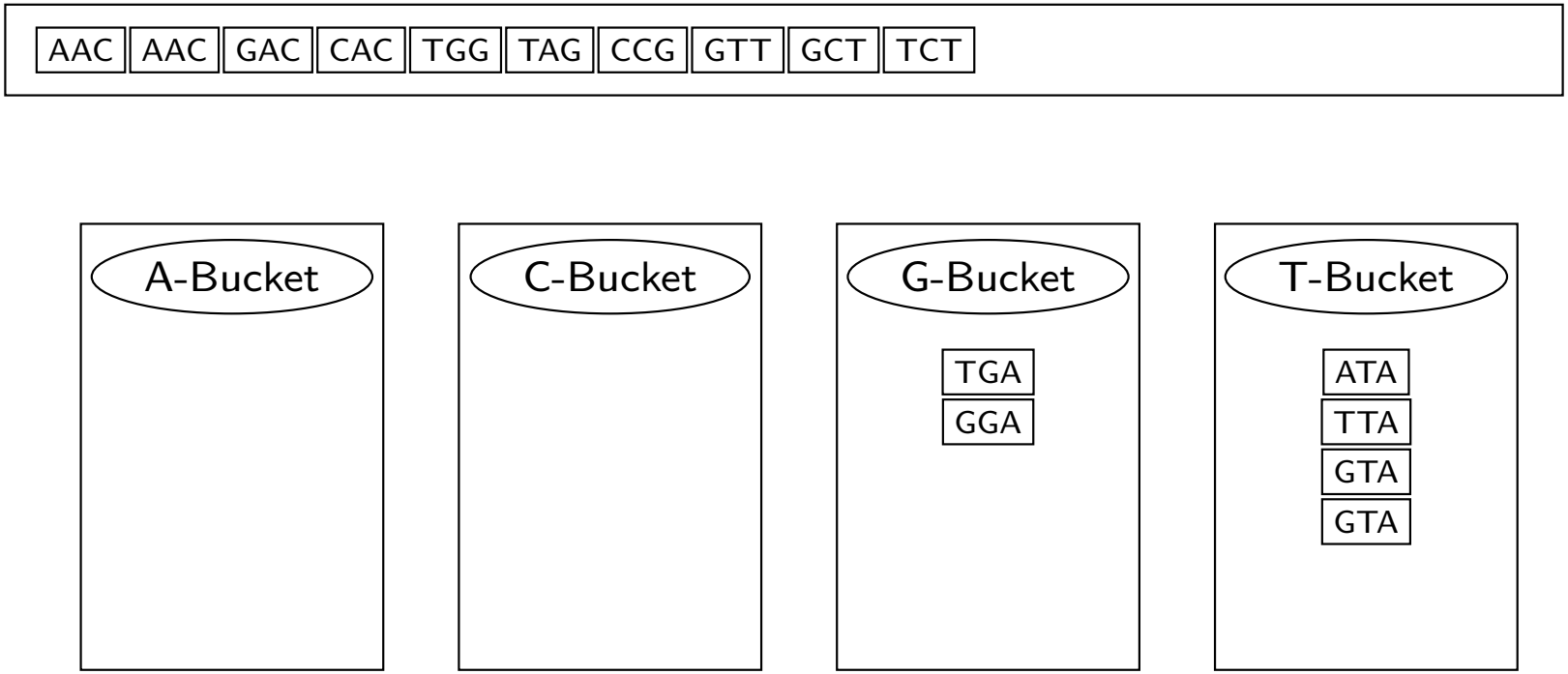

TGA
TTA
GTA
GGA
GTA
AAC
AAC
GAC
CAC
TGG
TAG
CCG
GTT
GCT
TCT
A-Bucket
C-Bucket
G-Bucket
T-Bucket
ATA

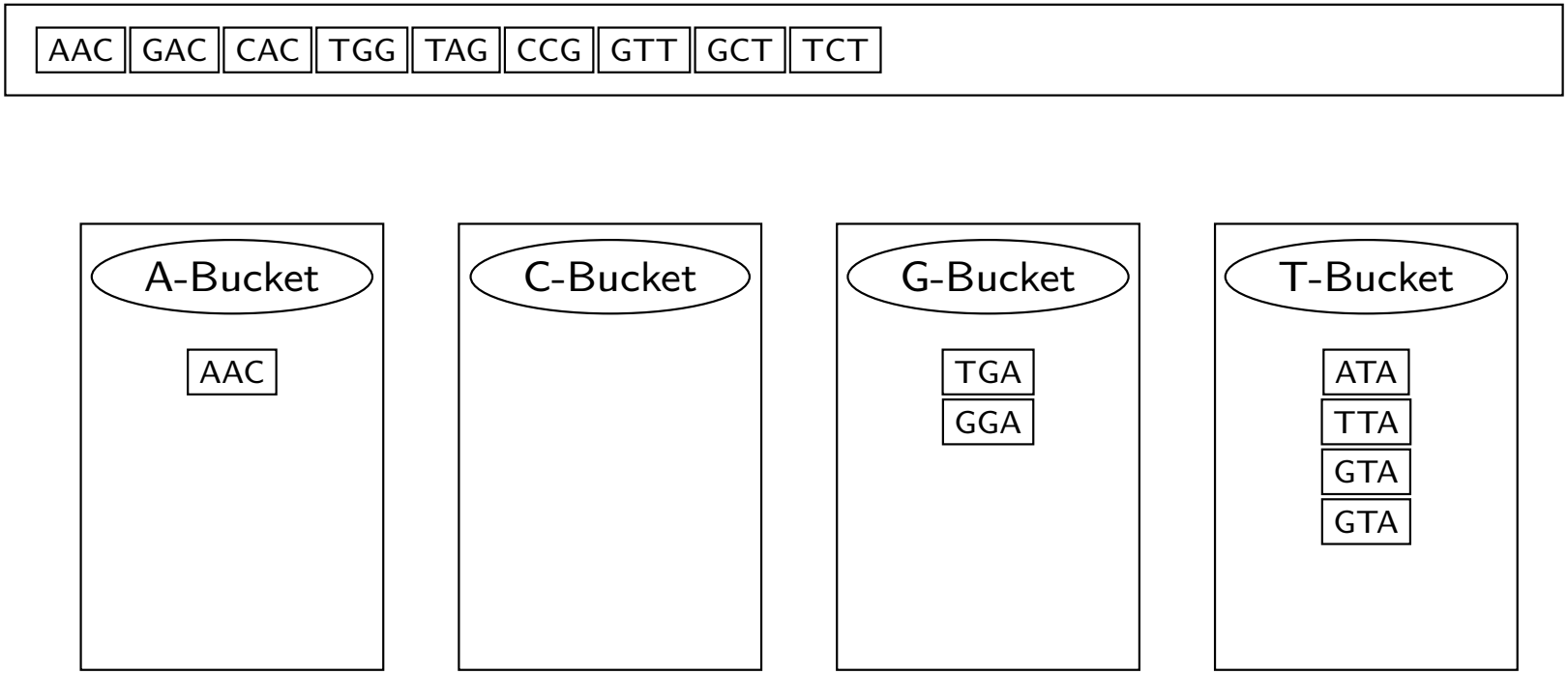

TTA
GTA
GGA
GTA
AAC
AAC
GAC
CAC
TGG
TAG
CCG
GTT
GCT
TCT
A-Bucket
C-Bucket
G-Bucket
TGA
T-Bucket
ATA

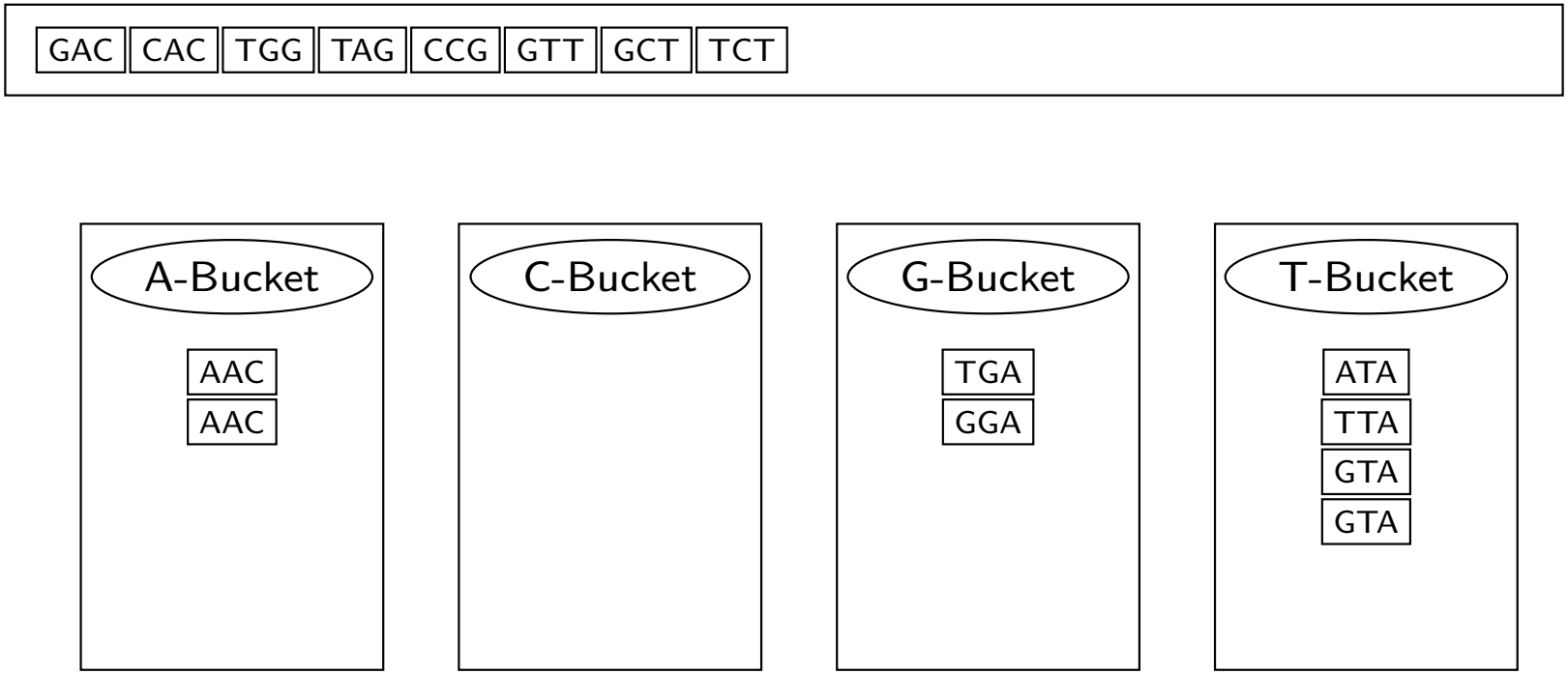

GTA
GGA
GTA
AAC
AAC
GAC
CAC
TGG
TAG
CCG
GTT
GCT
TCT
A-Bucket
C-Bucket
G-Bucket
TGA
T-Bucket
ATA
TTA

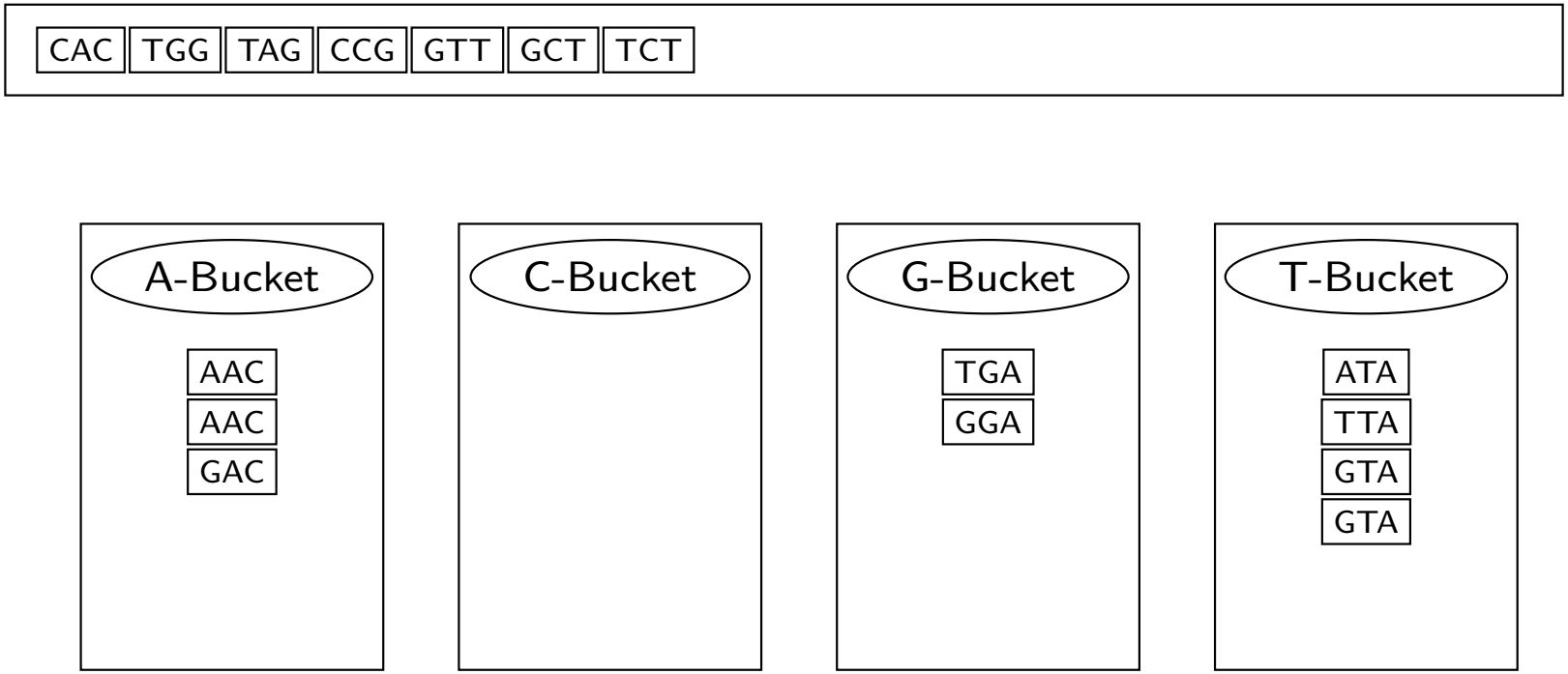

GGA
GTA
AAC
AAC
GAC
CAC
TGG
TAG
CCG
GTT
GCT
TCT
A-Bucket
C-Bucket
G-Bucket
TGA
T-Bucket
ATA
TTA
GTA

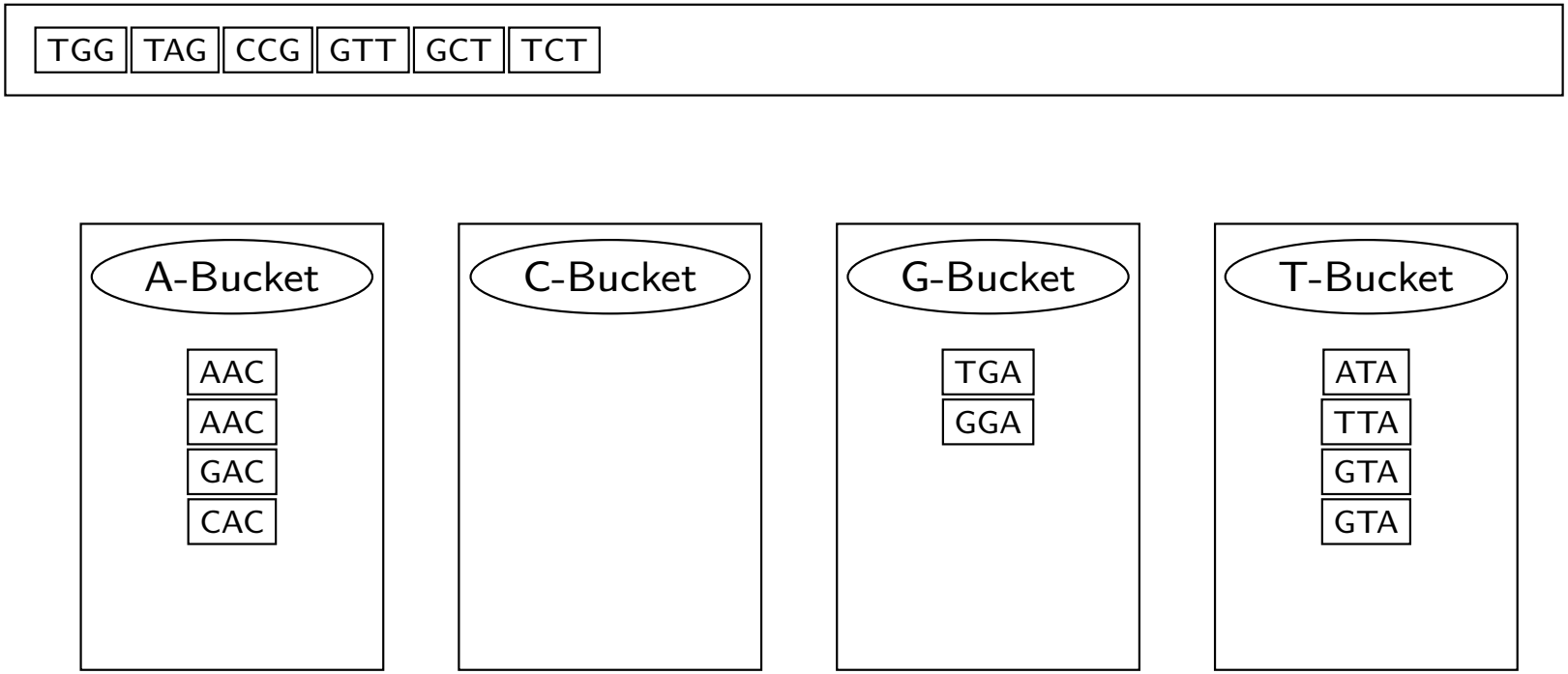

GTA
AAC
AAC
GAC
CAC
TGG
TAG
CCG
GTT
GCT
TCT
A-Bucket
C-Bucket
G-Bucket
TGA
GGA
T-Bucket
ATA
TTA
GTA

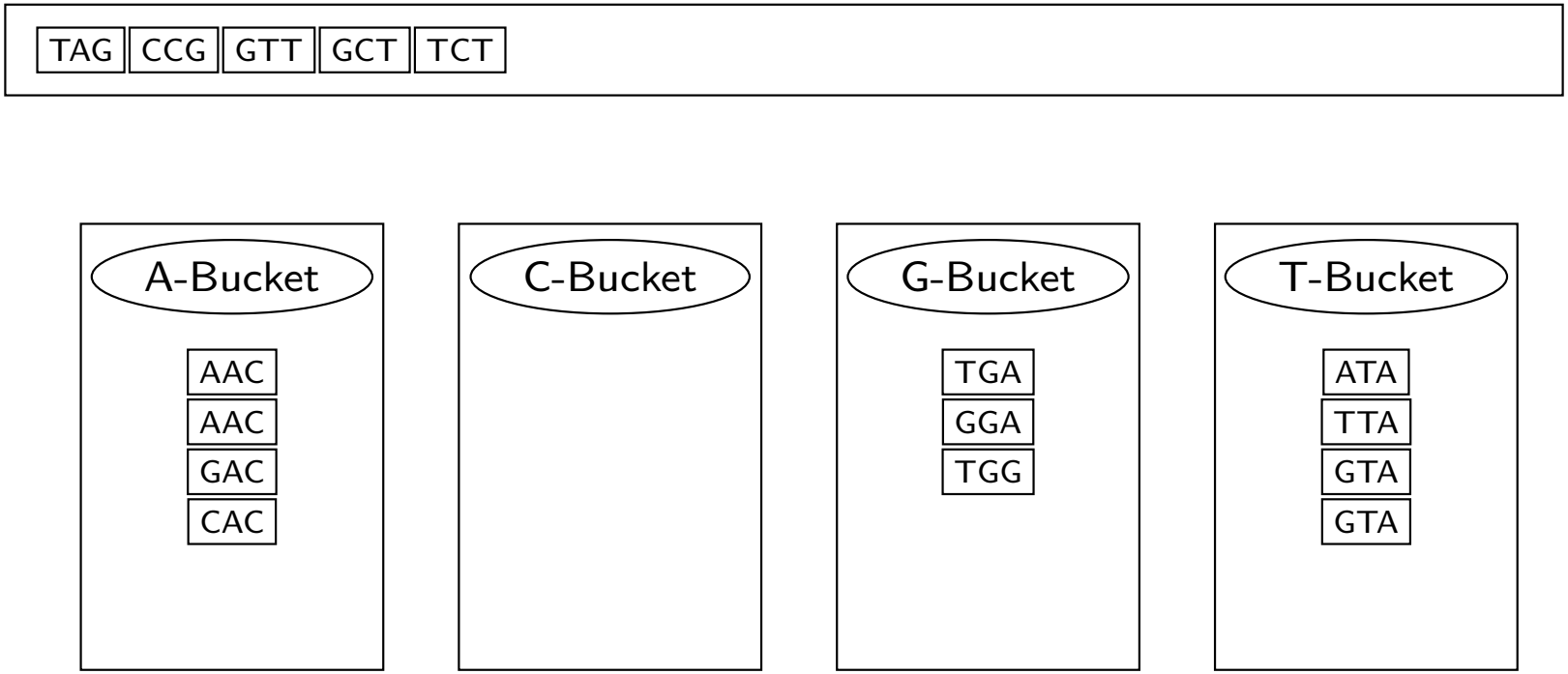

AAC
AAC
GAC
CAC
TGG
TAG
CCG
GTT
GCT
TCT
A-Bucket
C-Bucket
G-Bucket
TGA
GGA
T-Bucket
ATA
TTA
GTA
GTA

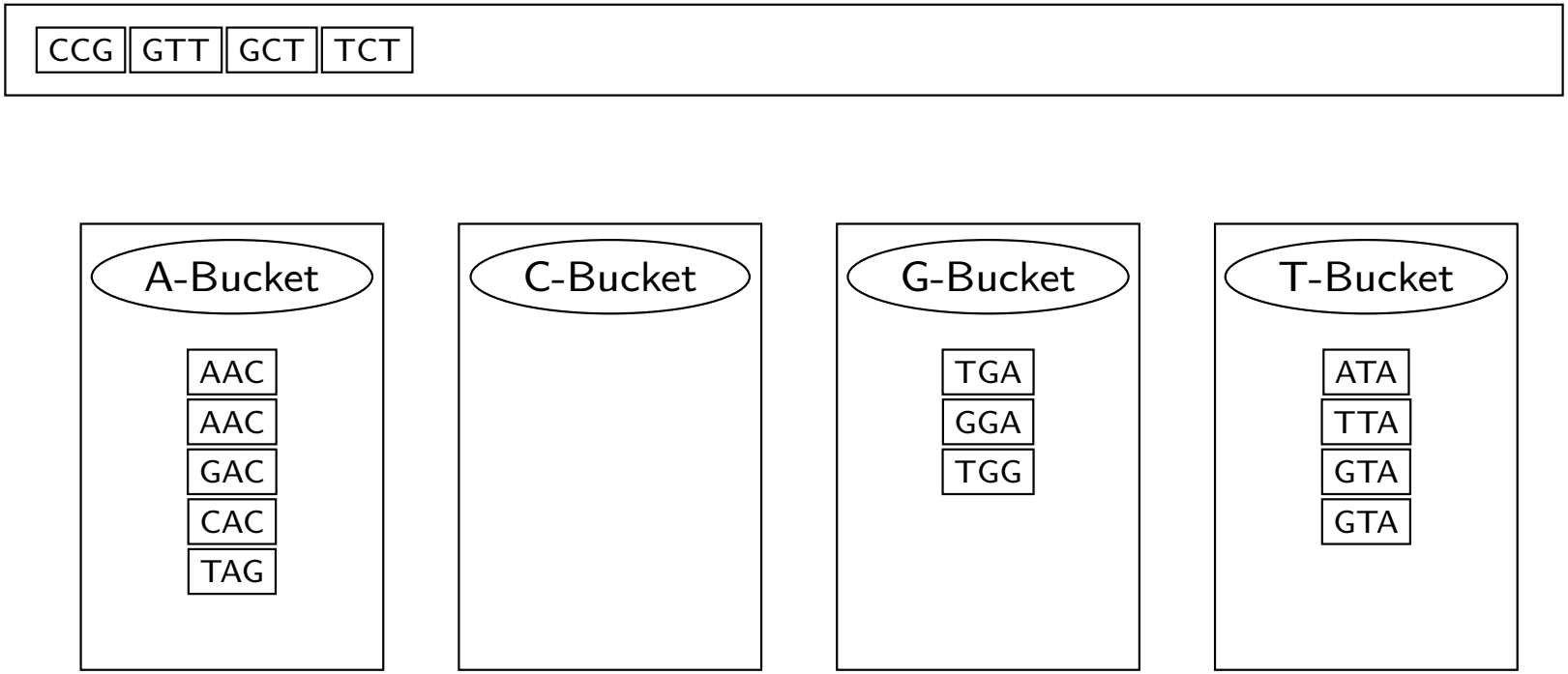

AAC
GAC
CAC
TGG
TAG
CCG
GTT
GCT
TCT
A-Bucket
AAC
C-Bucket
G-Bucket
TGA
GGA
T-Bucket
ATA
TTA
GTA
GTA

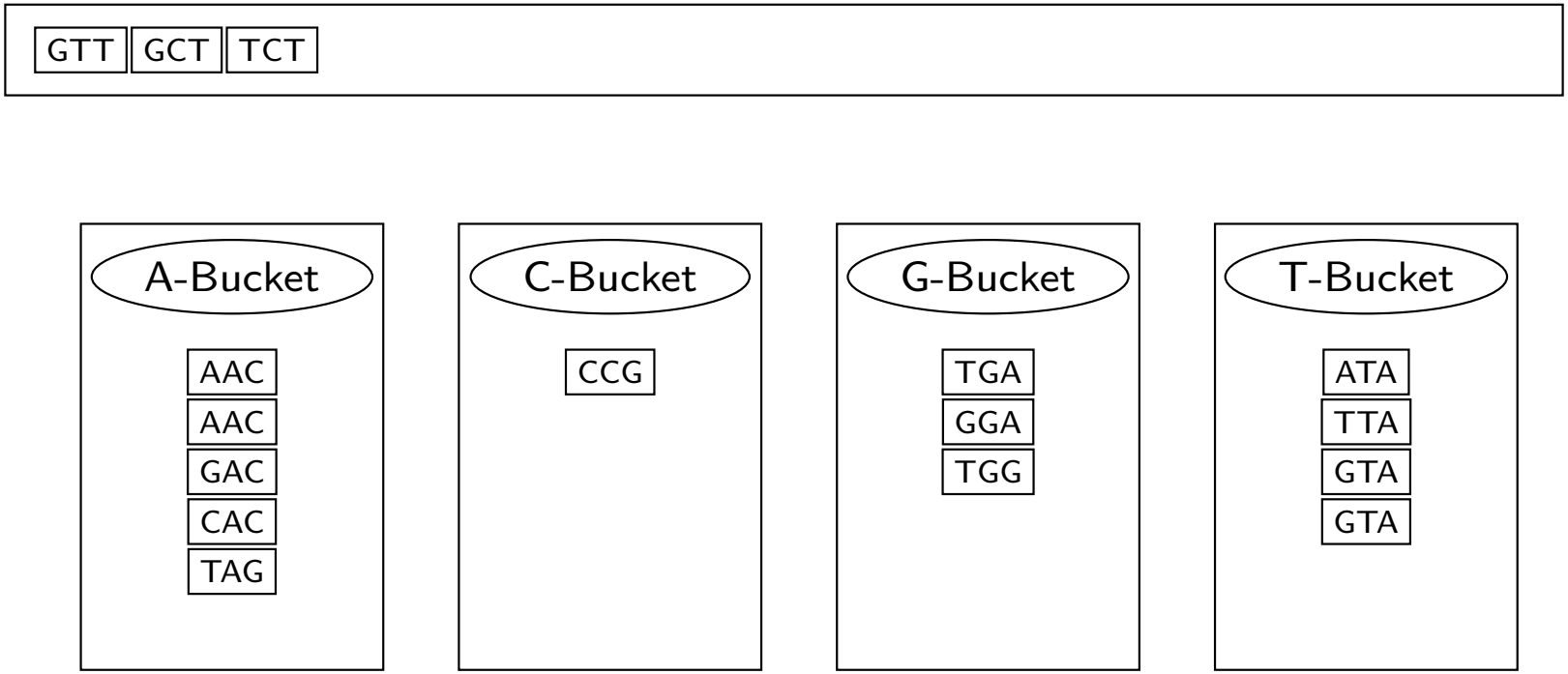

GAC
CAC
TGG
TAG
CCG
GTT
GCT
TCT
A-Bucket
AAC
AAC
C-Bucket
G-Bucket
TGA
GGA
T-Bucket
ATA
TTA
GTA
GTA

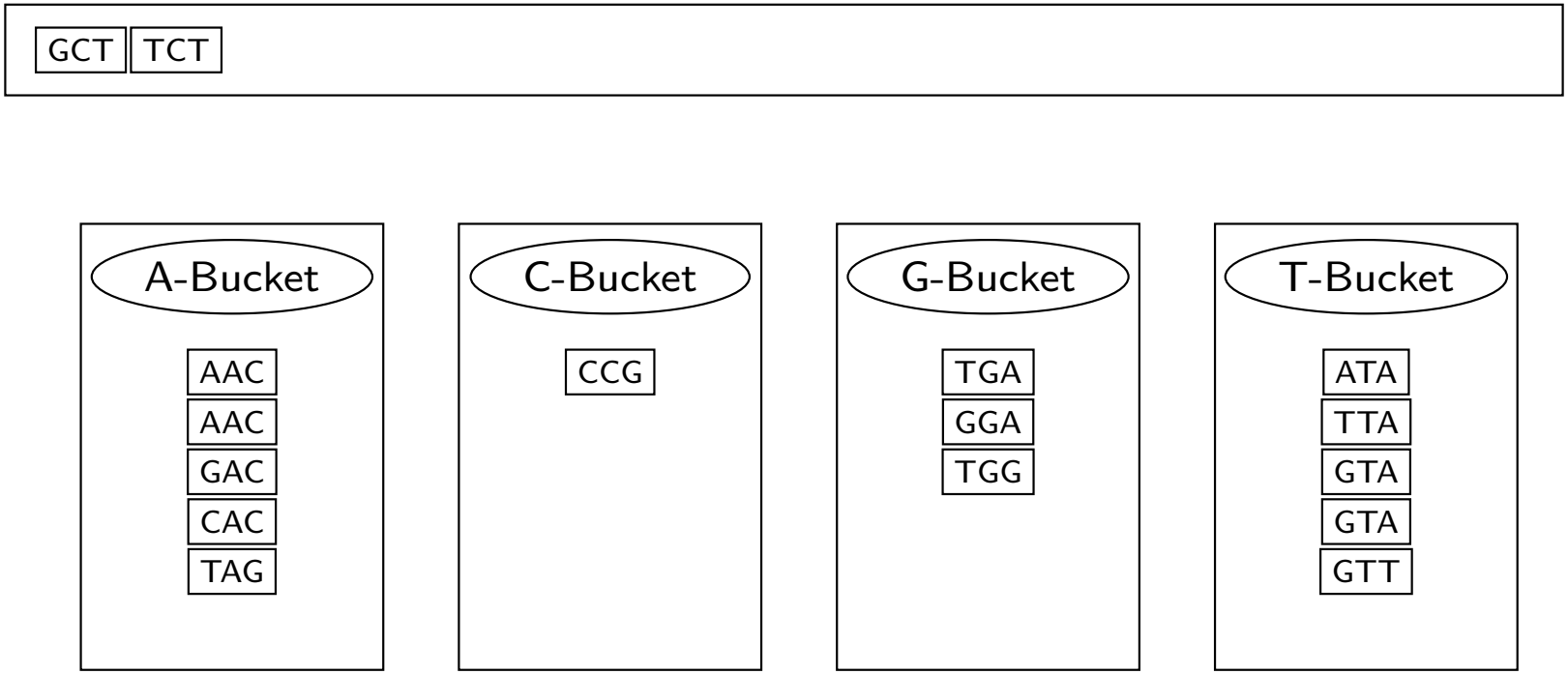

CAC
TGG
TAG
CCG
GTT
GCT
TCT
A-Bucket
AAC
AAC
GAC
C-Bucket
G-Bucket
TGA
GGA
T-Bucket
ATA
TTA
GTA
GTA

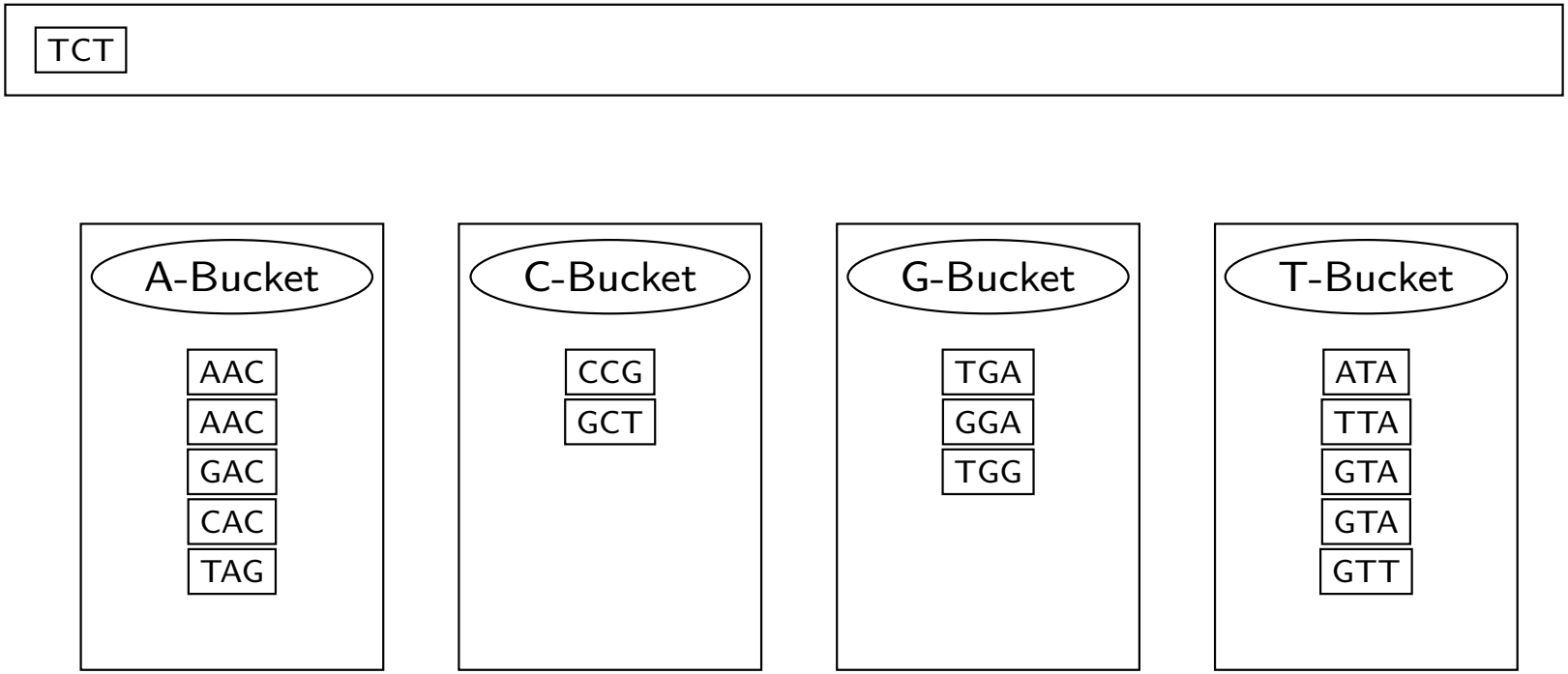

TGG
TAG
CCG
GTT
GCT
TCT
A-Bucket
AAC
AAC
GAC
CAC
C-Bucket
G-Bucket
TGA
GGA
T-Bucket
ATA
TTA
GTA
GTA

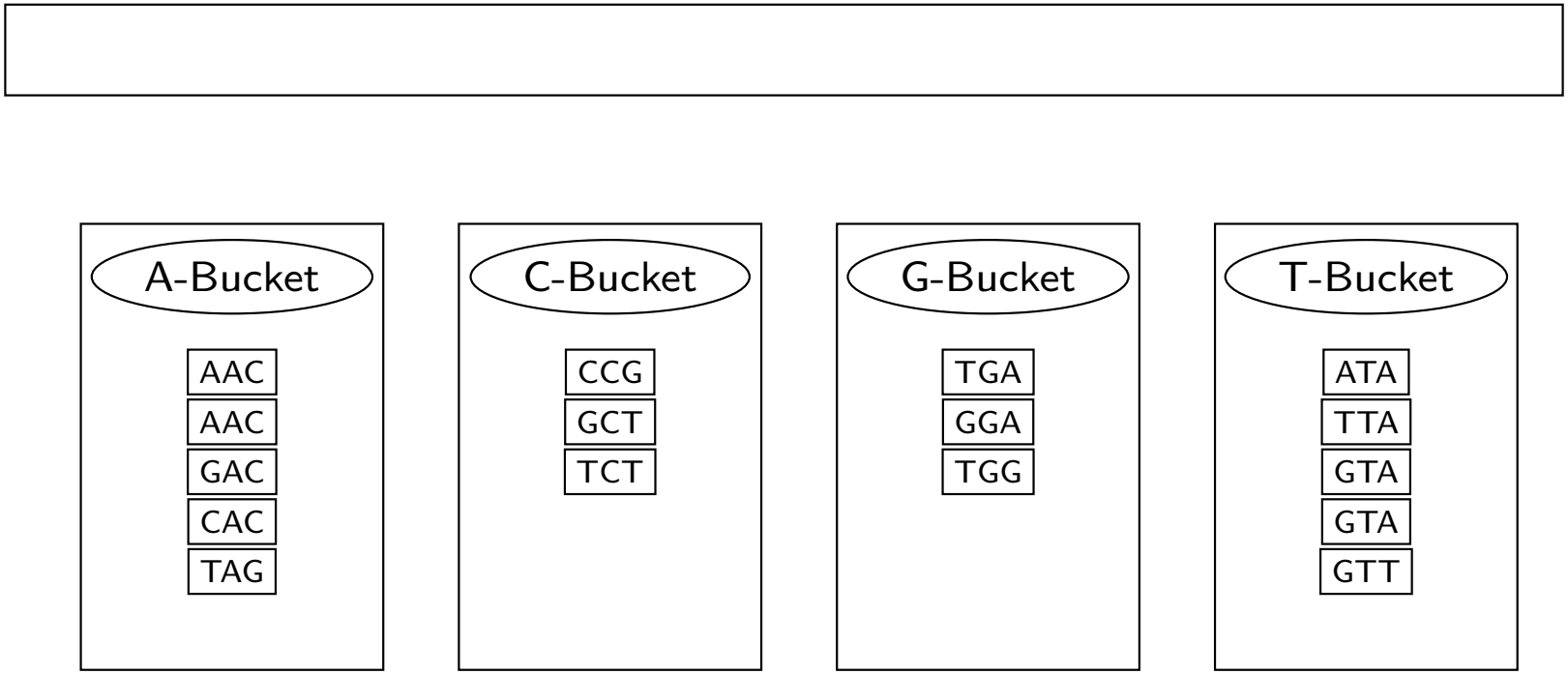

TAG
CCG
GTT
GCT
TCT
A-Bucket
AAC
AAC
GAC
CAC
C-Bucket
G-Bucket
TGA
GGA
TGG
T-Bucket
ATA
TTA
GTA
GTA

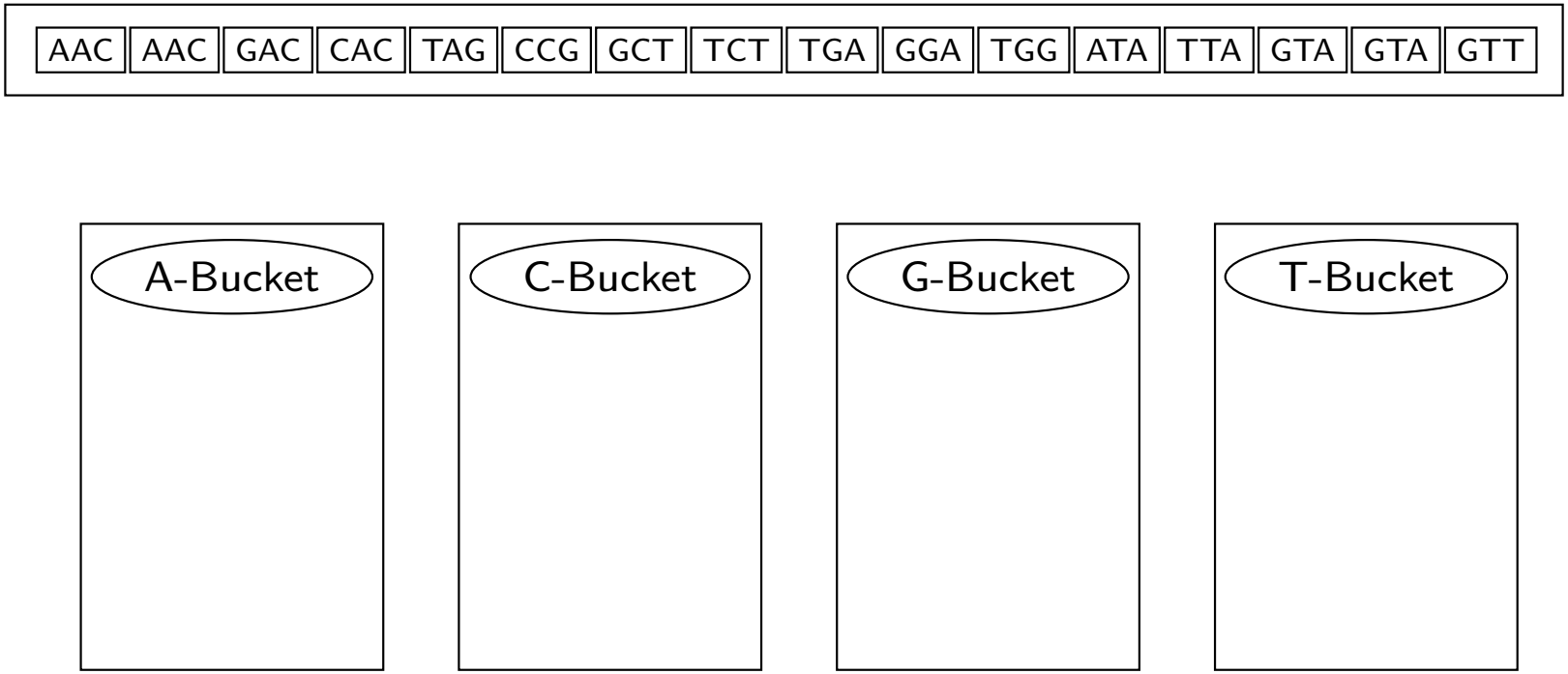

CCG
GTT
GCT
TCT
A-Bucket
AAC
AAC
GAC
CAC
TAG
C-Bucket
G-Bucket
TGA
GGA
TGG
T-Bucket
ATA
TTA
GTA
GTA

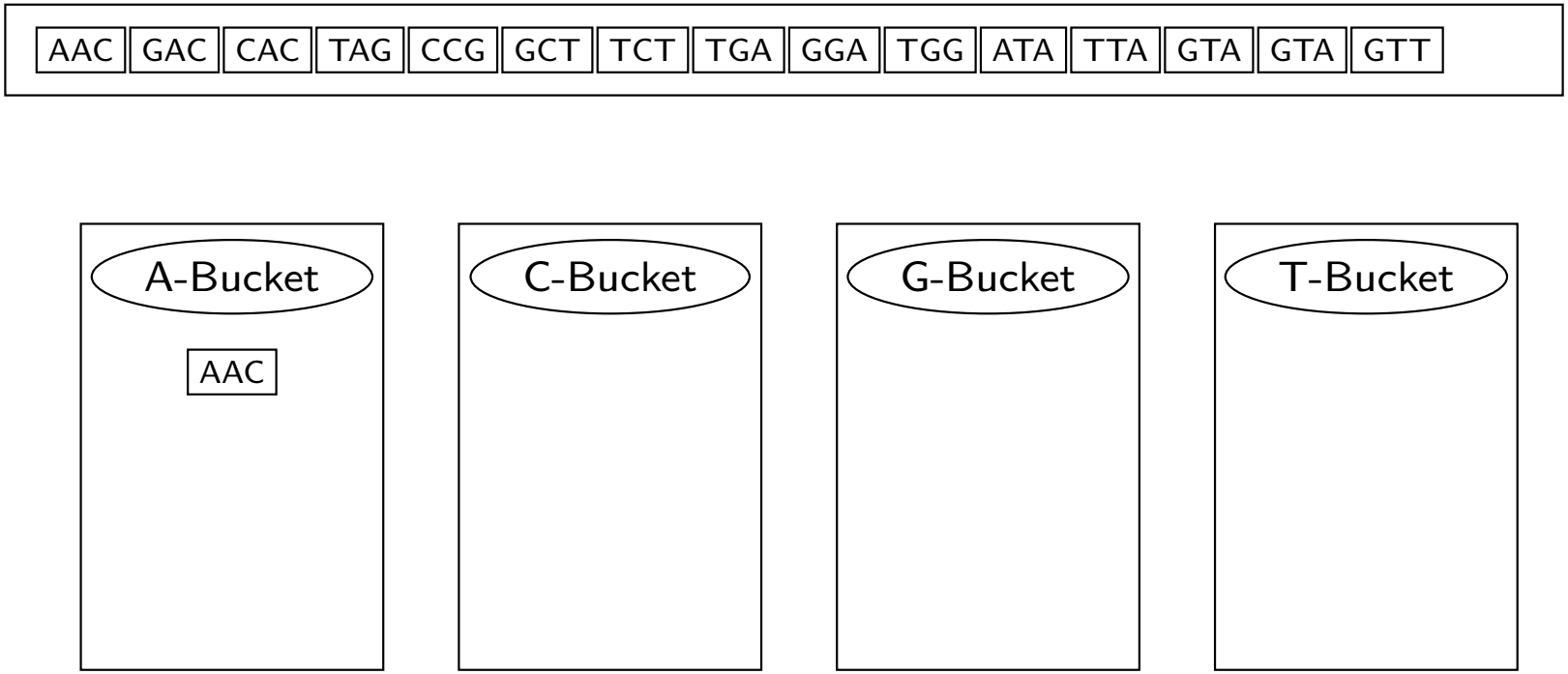

GTT
GCT
TCT
A-Bucket
AAC
AAC
GAC
CAC
TAG
C-Bucket
CCG
G-Bucket
TGA
GGA
TGG
T-Bucket
ATA
TTA
GTA
GTA

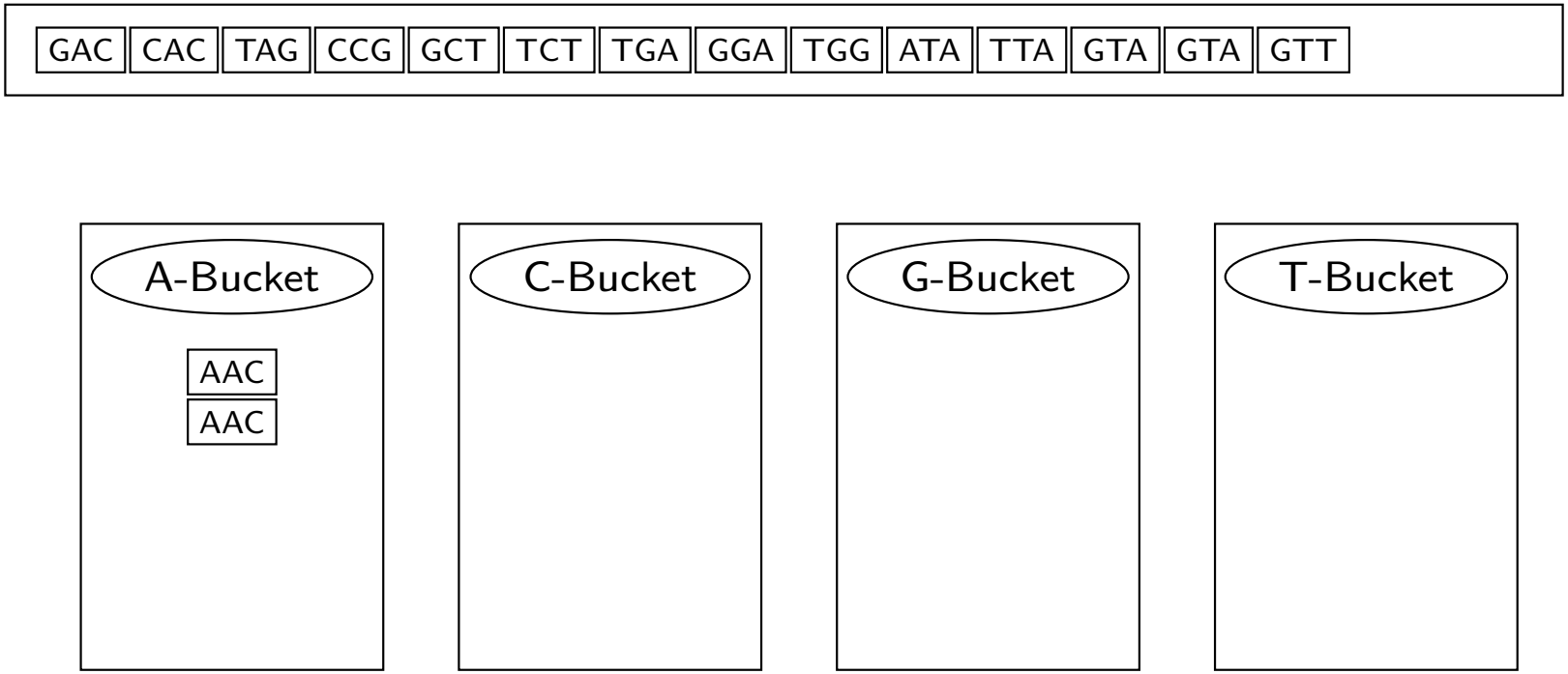

GCT
TCT
A-Bucket
AAC
AAC
GAC
CAC
TAG
C-Bucket
CCG
G-Bucket
TGA
GGA
TGG
T-Bucket
ATA
TTA
GTA
GTA
GTT

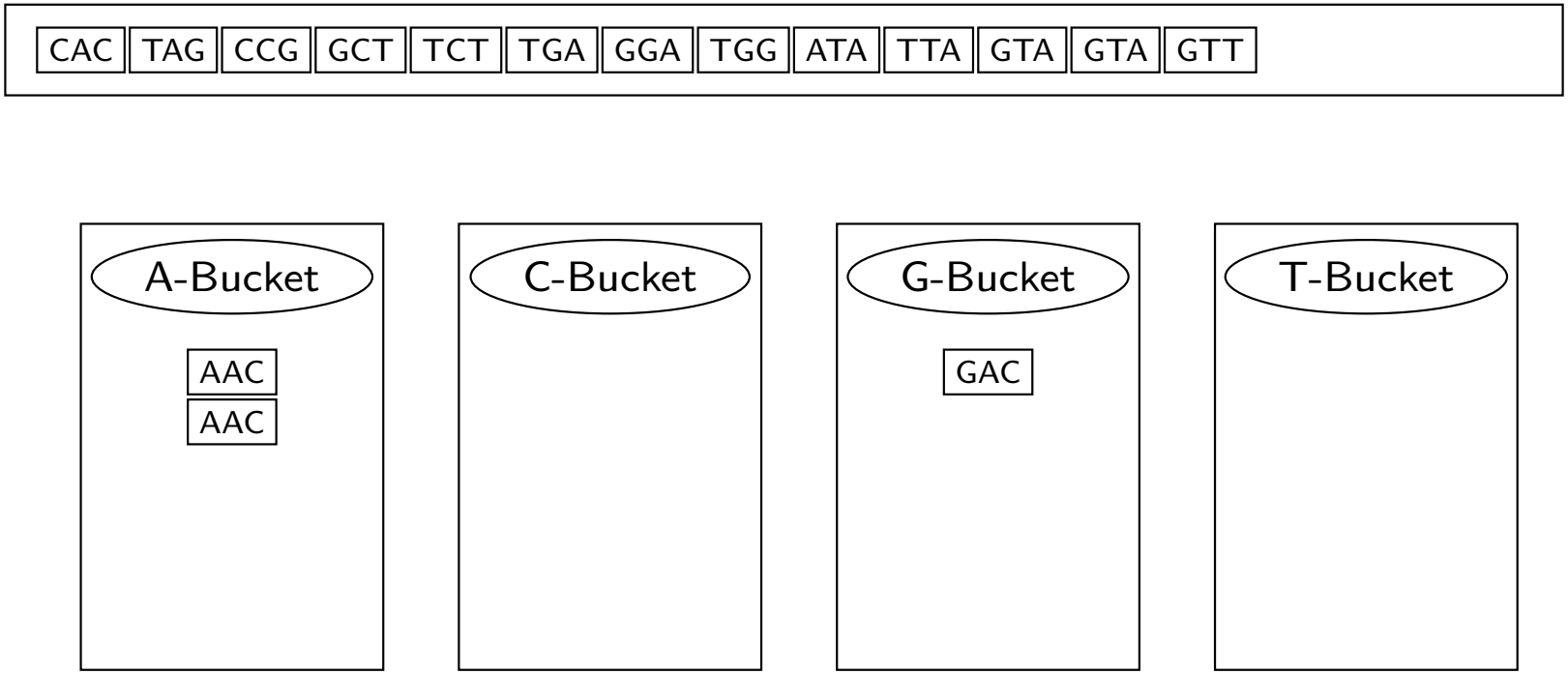

TCT
A-Bucket
AAC
AAC
GAC
CAC
TAG
C-Bucket
CCG
GCT
G-Bucket
TGA
GGA
TGG
T-Bucket
ATA
TTA
GTA
GTA
GTT

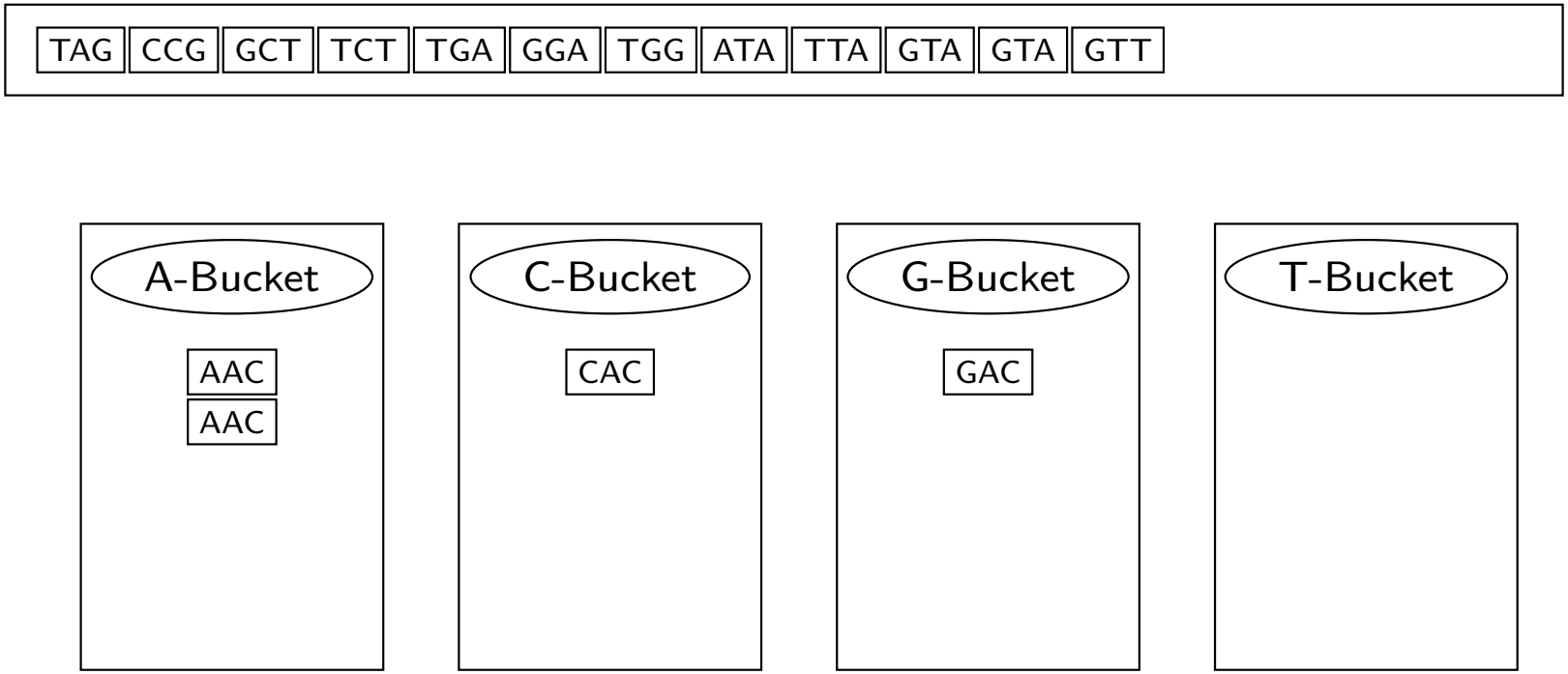

A-Bucket
AAC
AAC
GAC
CAC
TAG
C-Bucket
CCG
GCT
TCT
G-Bucket
TGA
GGA
TGG
T-Bucket
ATA
TTA
GTA
GTA
GTT

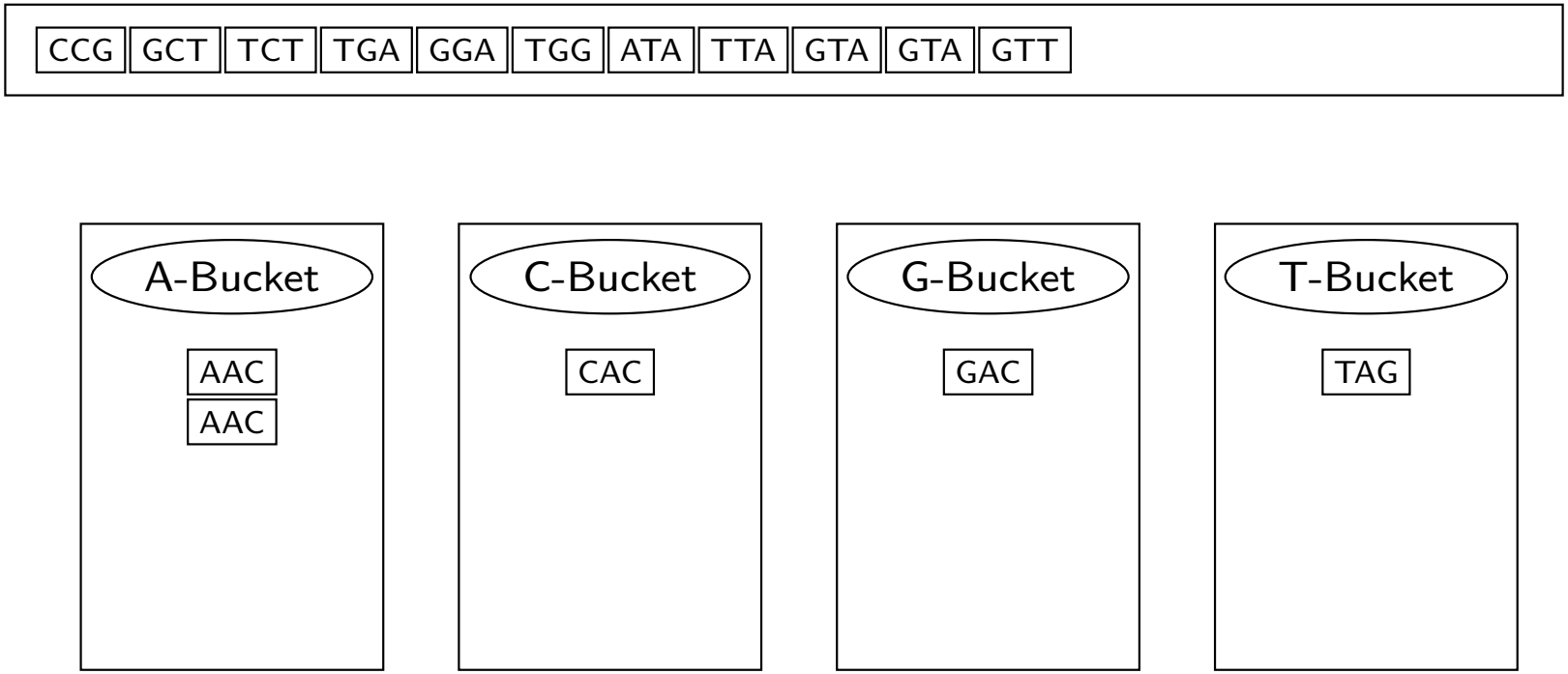

AAC
AAC
GAC
CAC
TAG
CCG
GCT
TCT
TGA
GGA
TGG
ATA
TTA
GTA
GTA
GTT
A-Bucket
C-Bucket
G-Bucket
T-Bucket

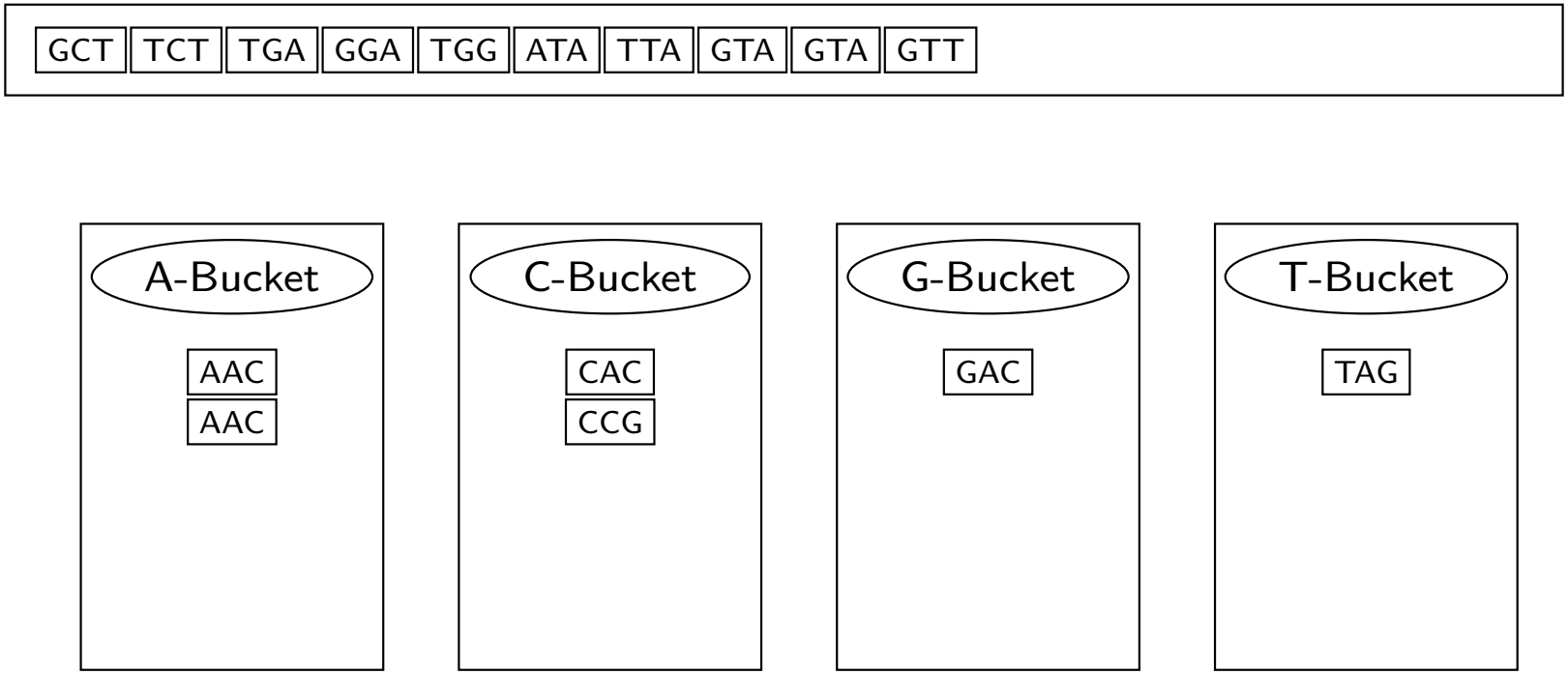

AAC
GAC
CAC
TAG
CCG
GCT
TCT
TGA
GGA
TGG
ATA
TTA
GTA
GTA
GTT
A-Bucket
AAC
C-Bucket
G-Bucket
T-Bucket

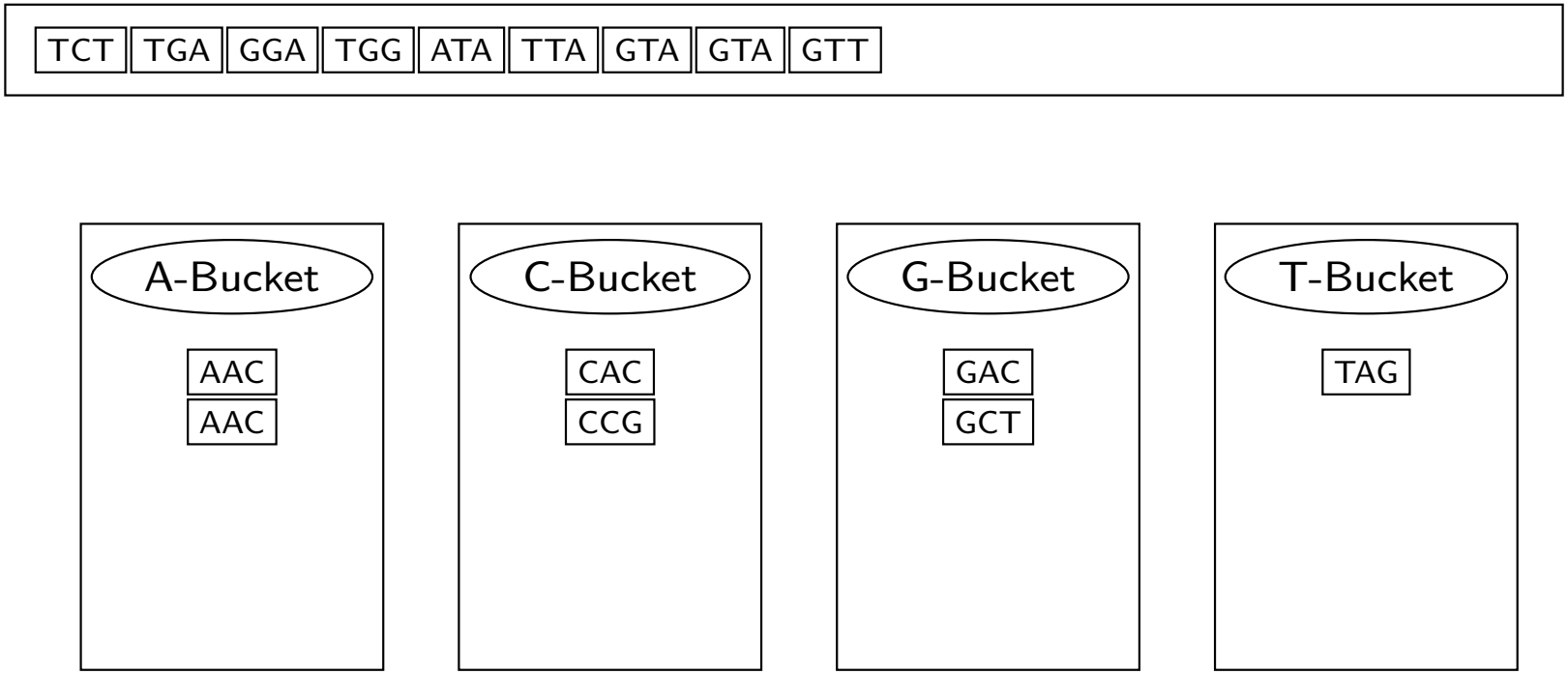

GAC
CAC
TAG
CCG
GCT
TCT
TGA
GGA
TGG
ATA
TTA
GTA
GTA
GTT
A-Bucket
AAC
AAC
C-Bucket
G-Bucket
T-Bucket

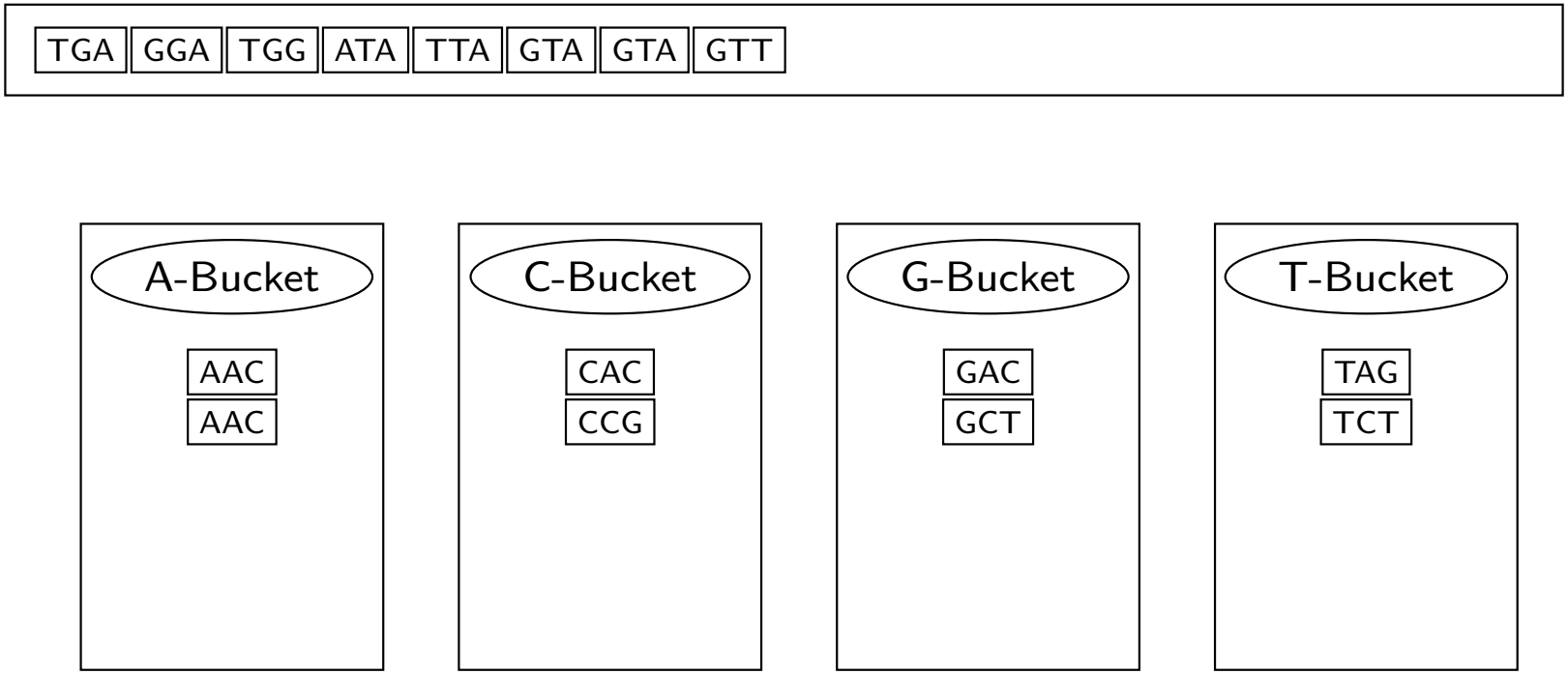

CAC
TAG
CCG
GCT
TCT
TGA
GGA
TGG
ATA
TTA
GTA
GTA
GTT
A-Bucket
AAC
AAC
C-Bucket
G-Bucket
GAC
T-Bucket

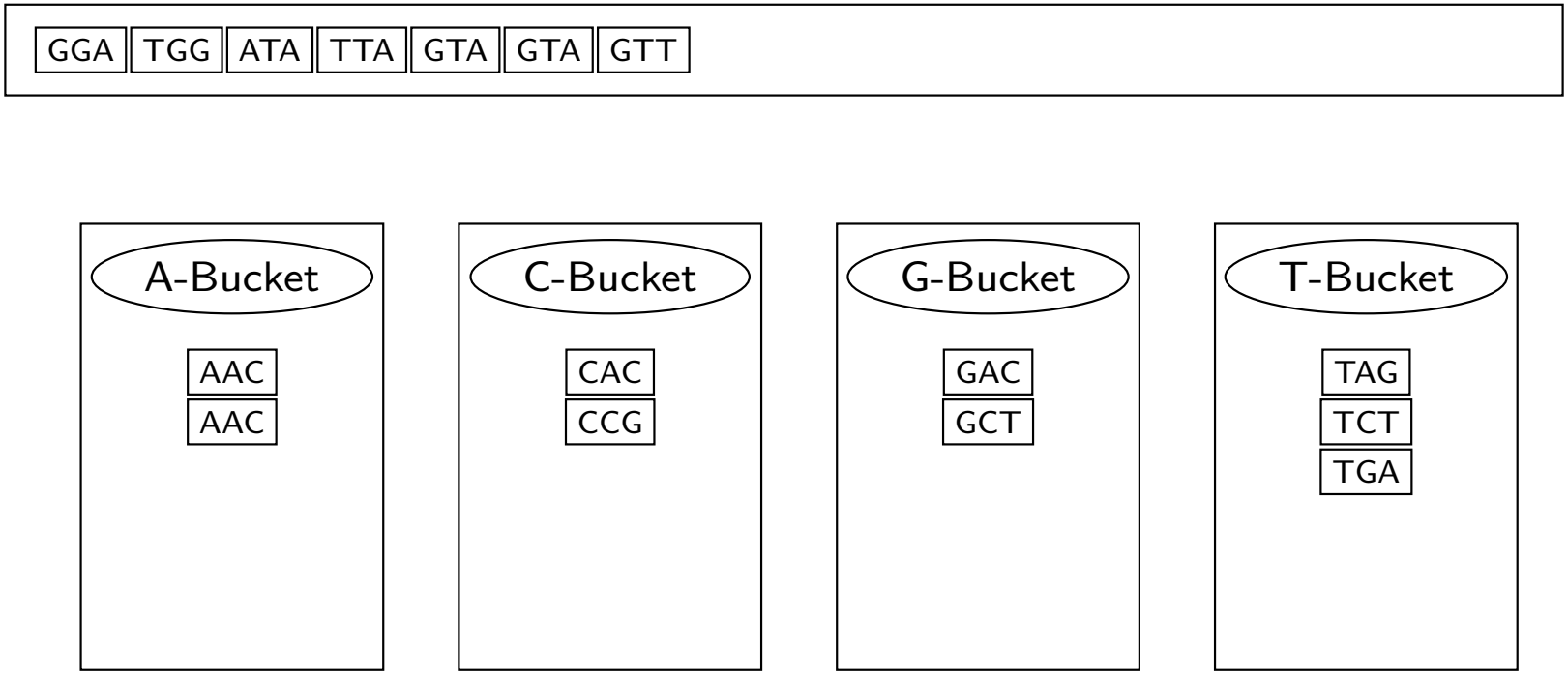

TAG
CCG
GCT
TCT
TGA
GGA
TGG
ATA
TTA
GTA
GTA
GTT
A-Bucket
AAC
AAC
C-Bucket
CAC
G-Bucket
GAC
T-Bucket

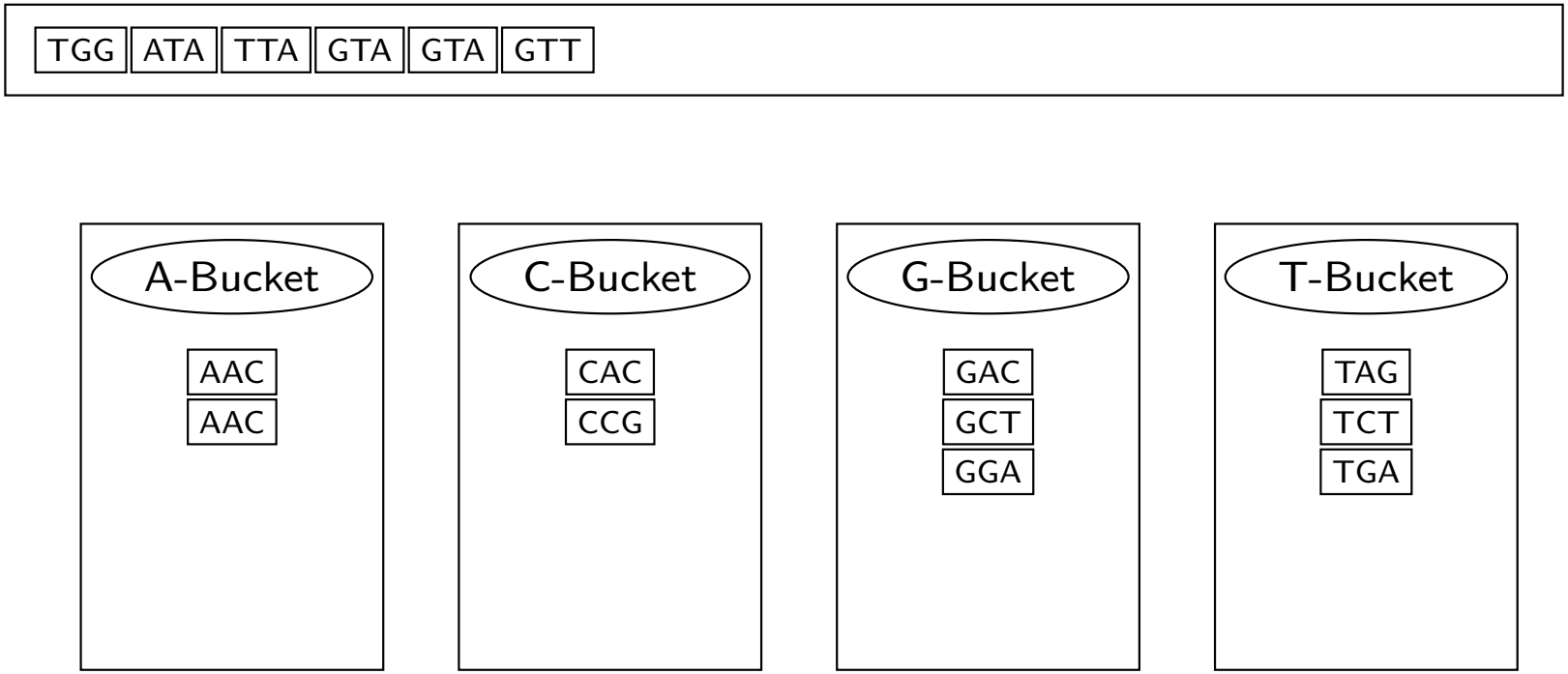

CCG
GCT
TCT
TGA
GGA
TGG
ATA
TTA
GTA
GTA
GTT
A-Bucket
AAC
AAC
C-Bucket
CAC
G-Bucket
GAC
T-Bucket
TAG

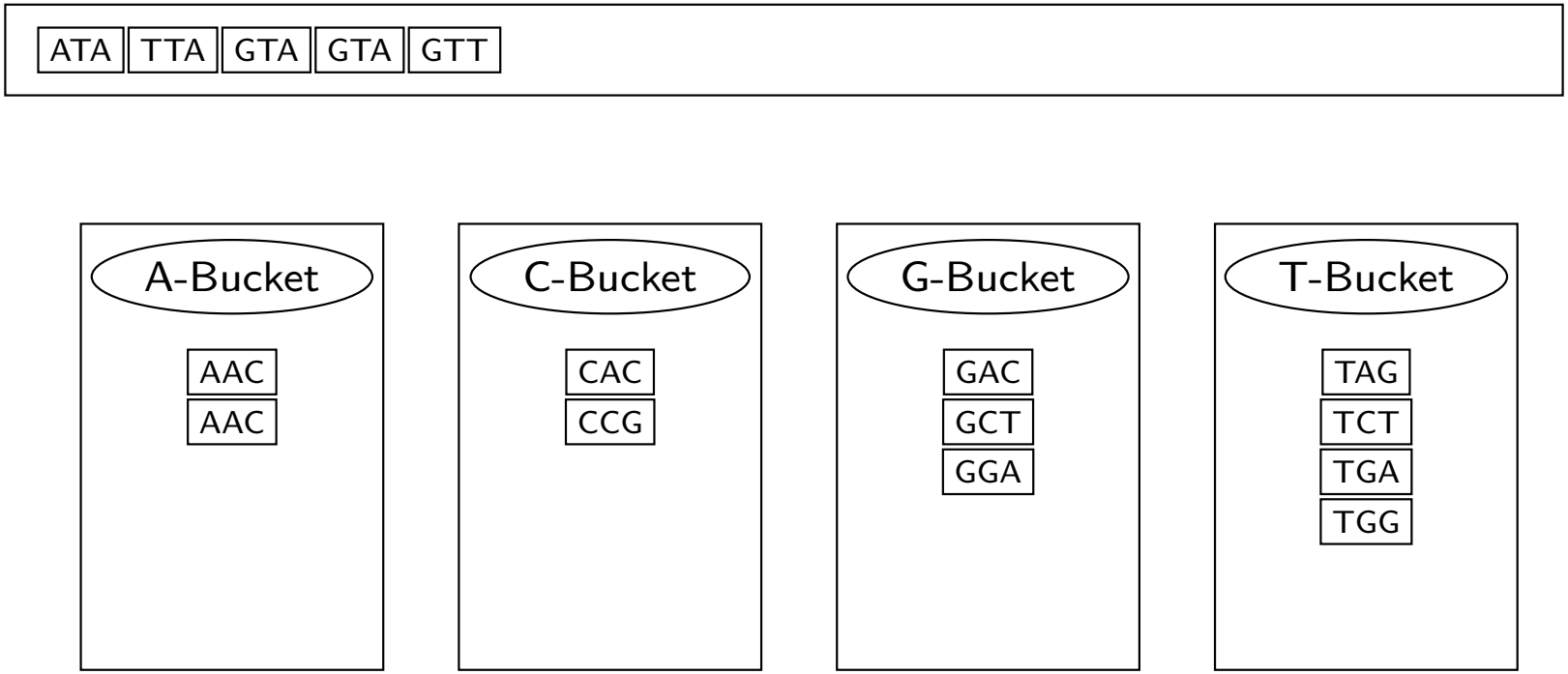

GCT
TCT
TGA
GGA
TGG
ATA
TTA
GTA
GTA
GTT
A-Bucket
AAC
AAC
C-Bucket
CAC
CCG
G-Bucket
GAC
T-Bucket
TAG

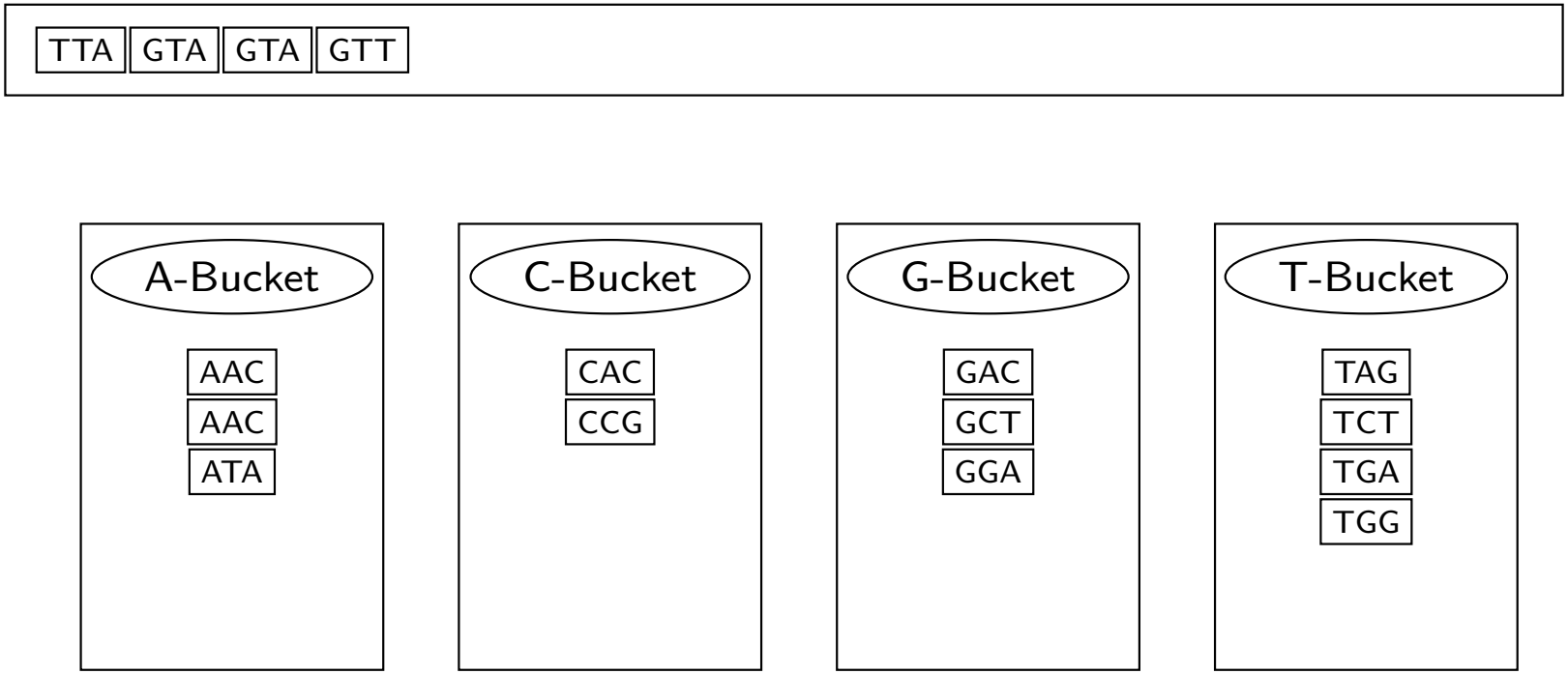

TCT
TGA
GGA
TGG
ATA
TTA
GTA
GTA
GTT
A-Bucket
AAC
AAC
C-Bucket
CAC
CCG
G-Bucket
GAC
GCT
T-Bucket
TAG

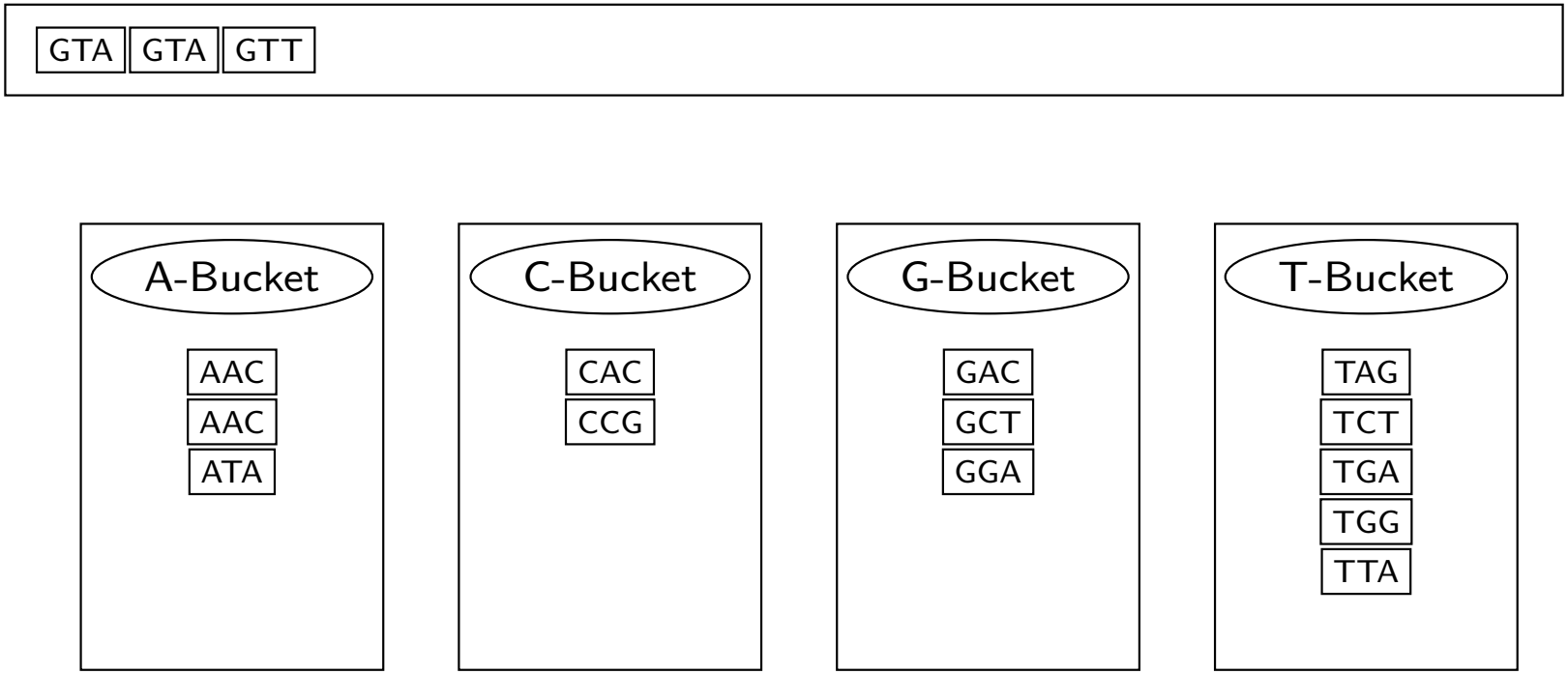

TGA
GGA
TGG
ATA
TTA
GTA
GTA
GTT
A-Bucket
AAC
AAC
C-Bucket
CAC
CCG
G-Bucket
GAC
GCT
T-Bucket
TAG
TCT

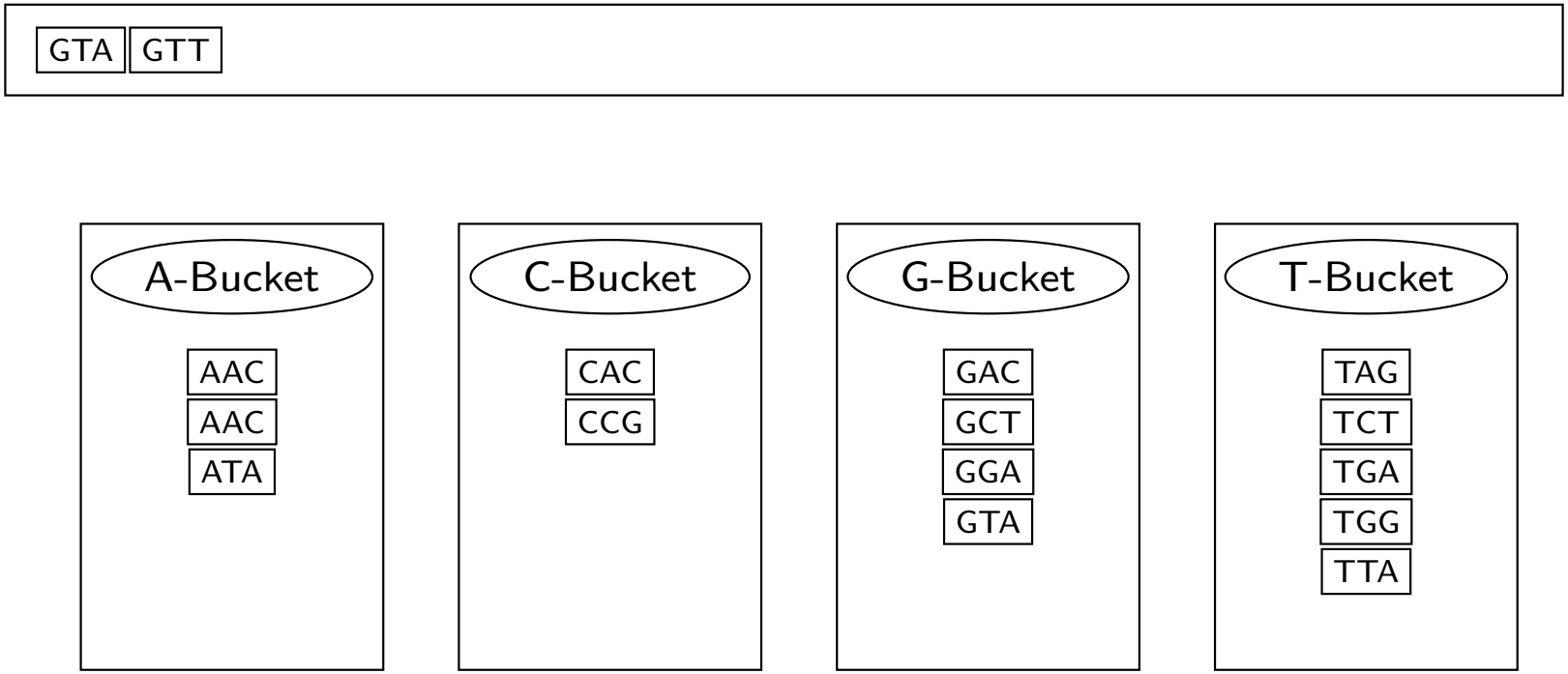

GGA
TGG
ATA
TTA
GTA
GTA
GTT
A-Bucket
AAC
AAC
C-Bucket
CAC
CCG
G-Bucket
GAC
GCT
T-Bucket
TAG
TCT
TGA

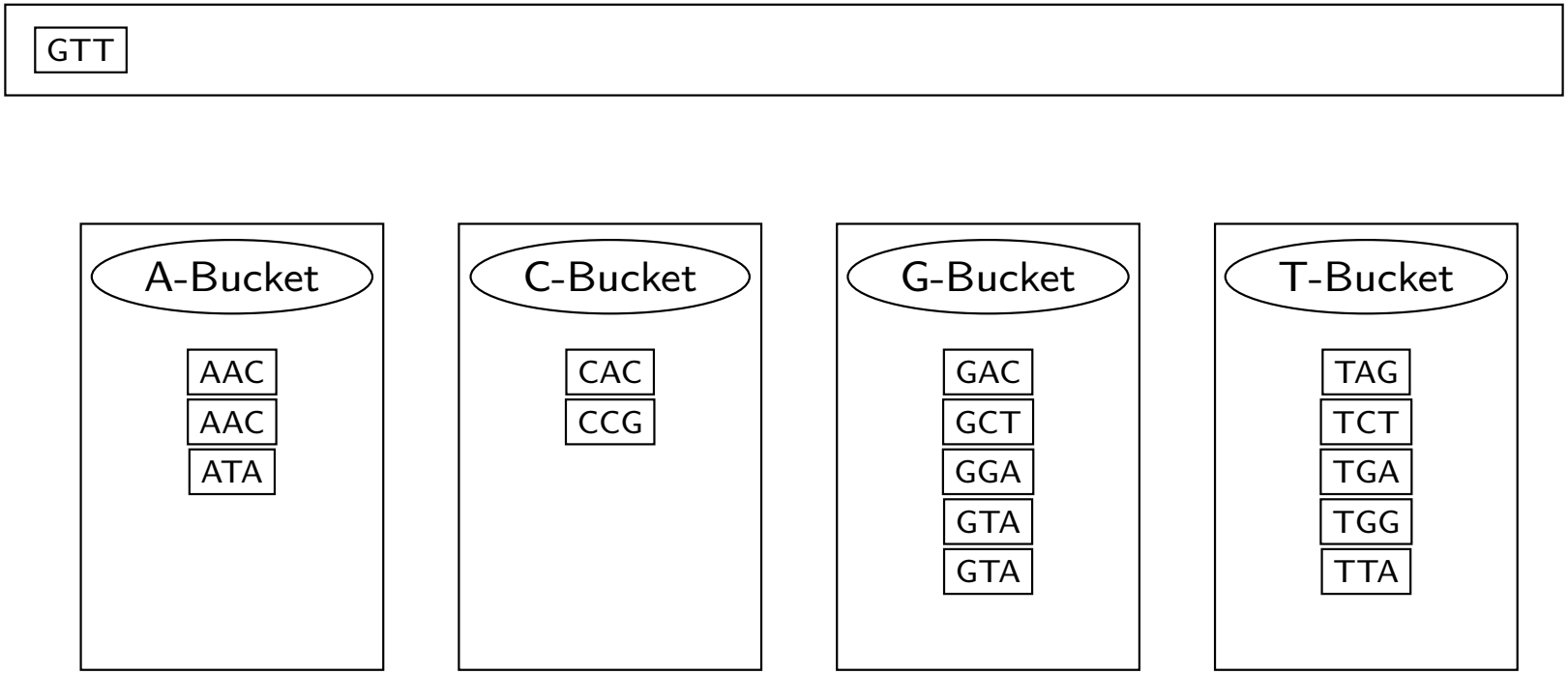

TGG
ATA
TTA
GTA
GTA
GTT
A-Bucket
AAC
AAC
C-Bucket
CAC
CCG
G-Bucket
GAC
GCT
GGA
T-Bucket
TAG
TCT
TGA

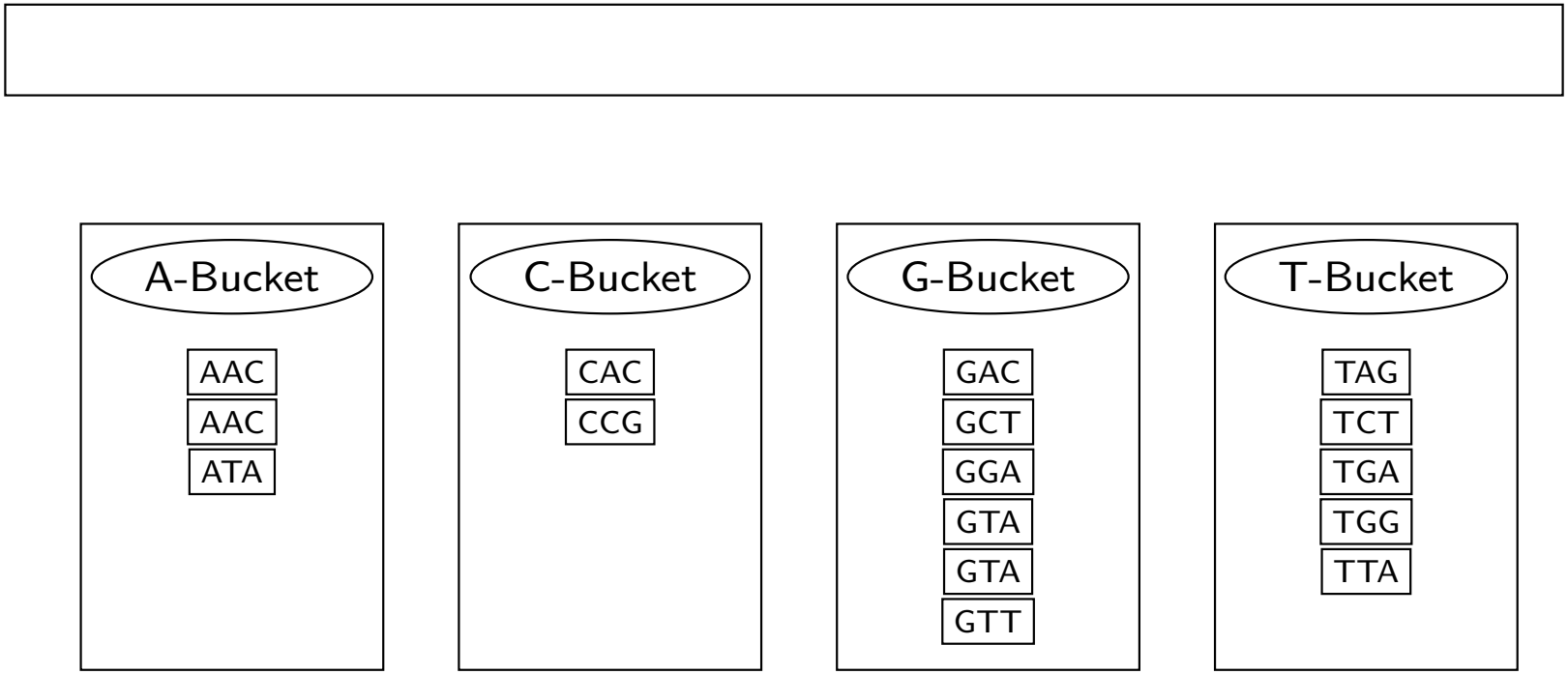

ATA
TTA
GTA
GTA
GTT
A-Bucket
AAC
AAC
C-Bucket
CAC
CCG
G-Bucket
GAC
GCT
GGA
T-Bucket
TAG
TCT
TGA
TGG


TTA
GTA
GTA
GTT
A-Bucket
AAC
AAC
ATA
C-Bucket
CAC
CCG
G-Bucket
GAC
GCT
GGA
T-Bucket
TAG
TCT
TGA
TGG

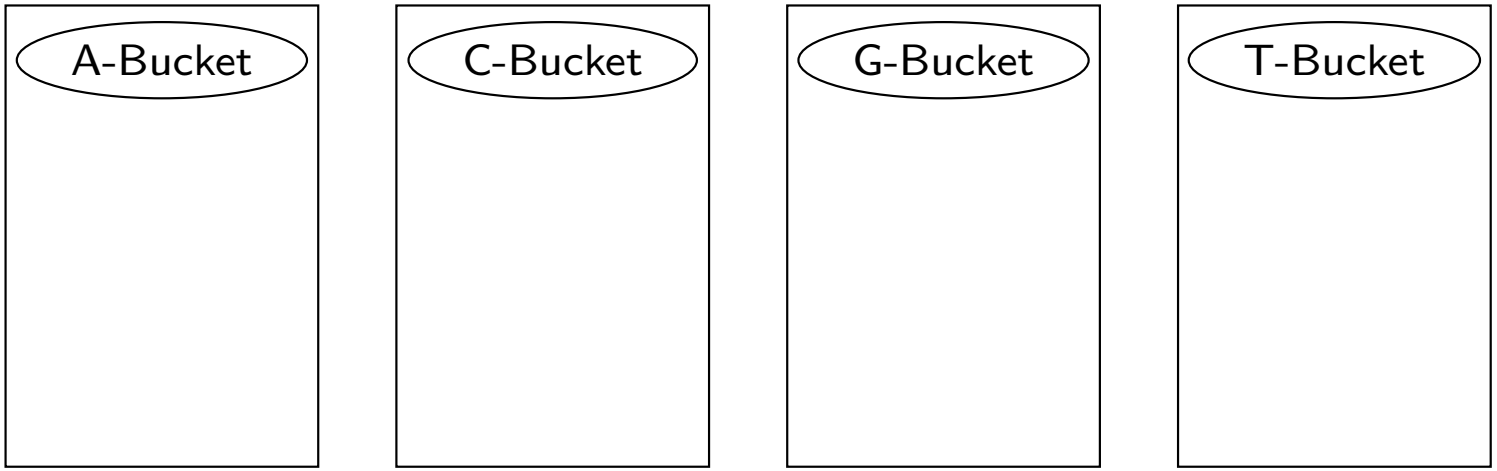

GTA
GTA
GTT
A-Bucket
AAC
AAC
ATA
C-Bucket
CAC
CCG
G-Bucket
GAC
GCT
GGA
T-Bucket
TAG
TCT
TGA
TGG
TTA


GTA
GTT
A-Bucket
AAC
AAC
ATA
C-Bucket
CAC
CCG
G-Bucket
GAC
GCT
GGA
GTA
T-Bucket
TAG
TCT
TGA
TGG
TTA


GTT
A-Bucket
AAC
AAC
ATA
C-Bucket
CAC
CCG
G-Bucket
GAC
GCT
GGA
GTA
GTA
T-Bucket
TAG
TCT
TGA
TGG
TTA


A-Bucket
AAC
AAC
ATA
C-Bucket
CAC
CCG
G-Bucket
GAC
GCT
GGA
GTA
GTA
GTT
T-Bucket
TAG
TCT
TGA
TGG
TTA


AAC
AAC
ATA
CAC
CCG
GAC
GCT
GGA
GTA
GTA
GTT
TAG
TCT
TGA
TGG
TTA


A-Bucket
C-Bucket
G-Bucket
T-Bucket