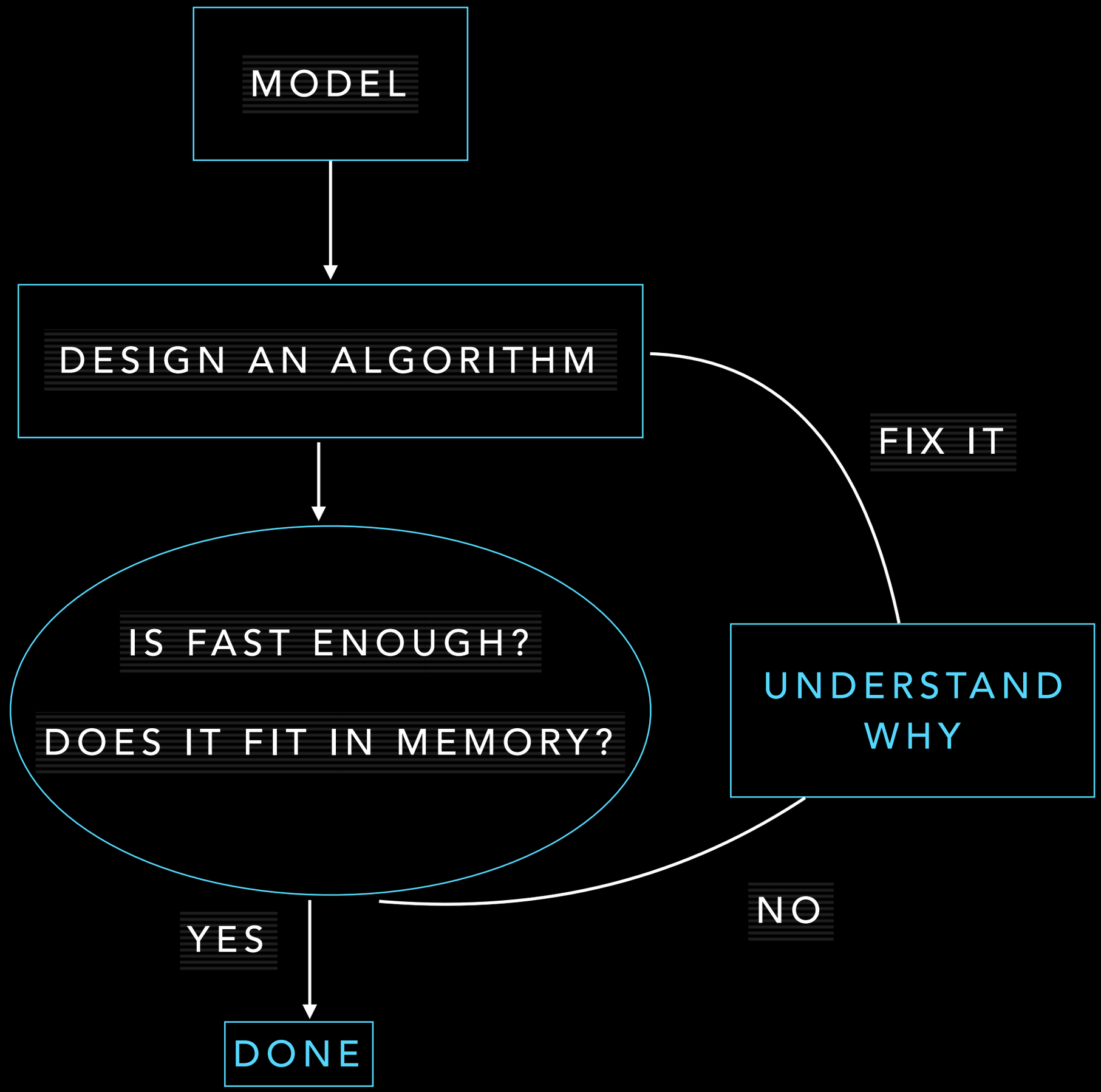
ELEMENTARY SORTS


MODEL
DESIGN AN ALGORITHM
IS FAST ENOUGH?
DOES IT FIT IN MEMORY?
FIX IT
UNDERSTAND WHY
NO
YES
DONE

SORTING

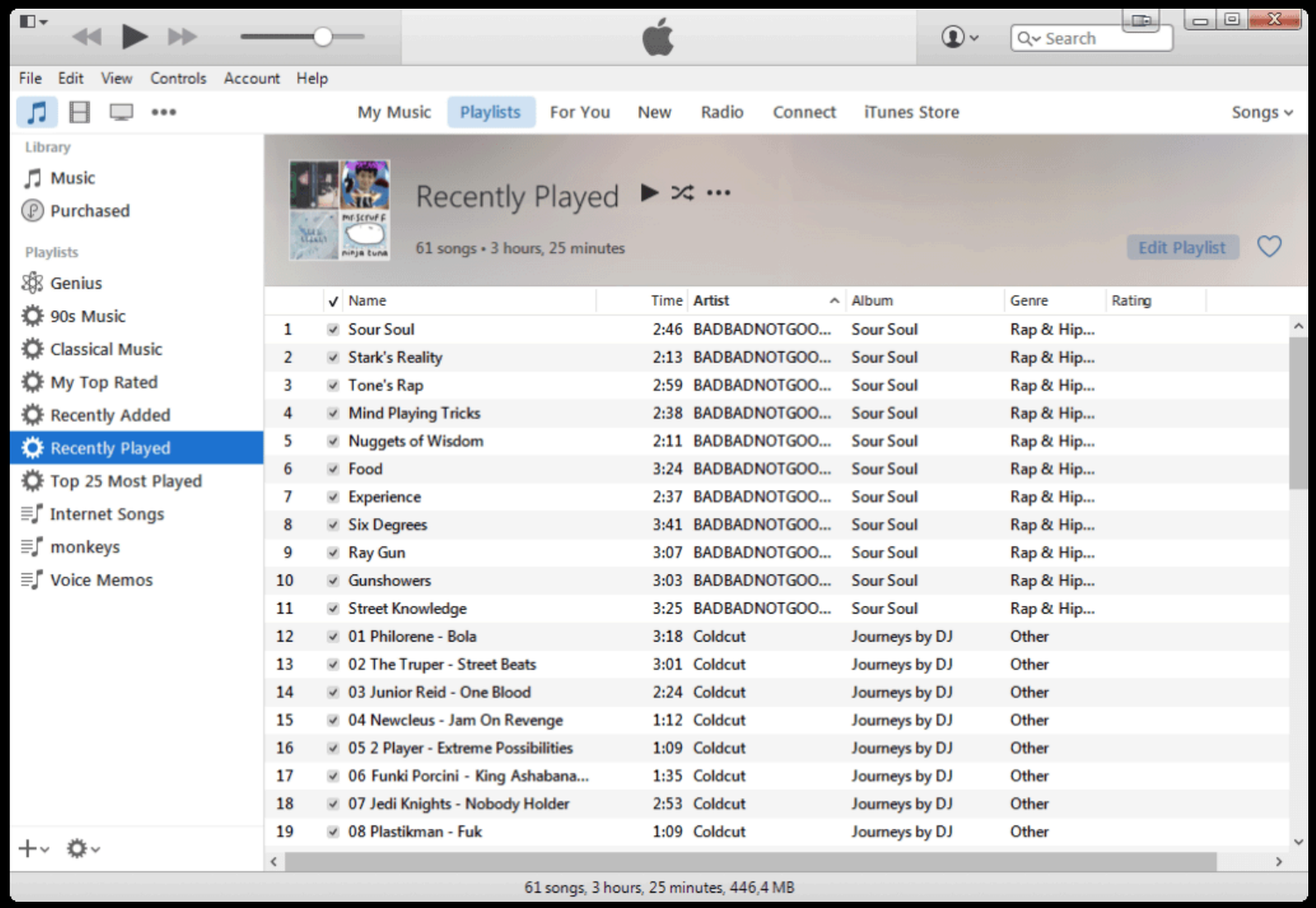
Ex. Student records in a university.

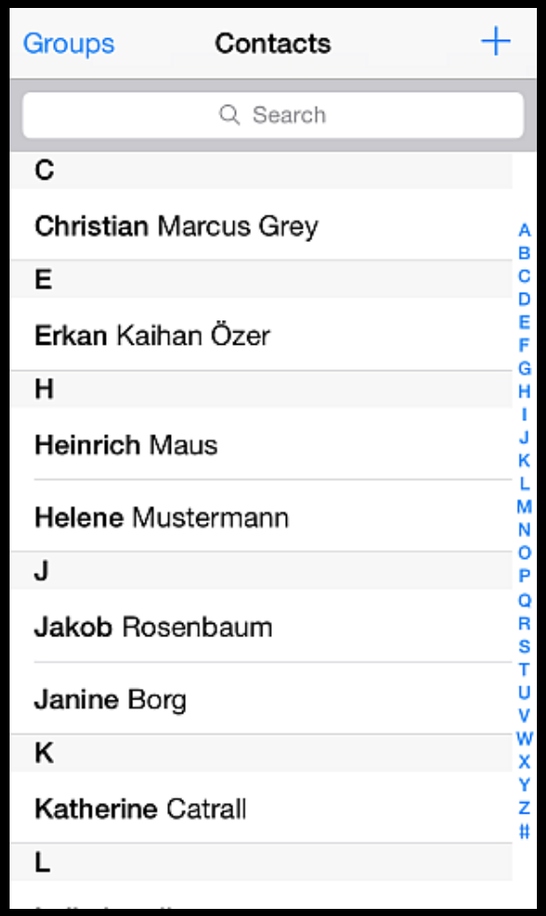
	Chen	3	A	991-878-4944
	Rohde	2	A	232-343-5555
	Gazsi	4	B	766-093-9873
item →	Furia	1	A	766-093-9873
	Kanaga	3	B	898-122-9643
	Andrews	3	A	664-480-0023
key →	Battle	4	C	874-088-1212

Sorted Collection:

Andrews	3	A	664-480-0023
Battle	4	C	874-088-1212
Chen	3	A	991-878-4944
Furia	1	A	766-093-9873
Gazsi	4	B	766-093-9873
Kanaga	3	B	898-122-9643
Rohde	2	A	232-343-5555

APPLICATIONS OF SORTING





WE WANT OUR INTERFACE TO BE GENERIC

Goal. Sort any type of data

SORTING INTEGERS

```
public class Experiment
{
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      Double[] a = new Double[N];
      for (int i = 0; i < N; i++)
         a[i] = StdRandom.uniform();
      Insertion.sort(a);
      for (int i = 0; i < N; i++)
         StdOut.println(a[i]);
   }
}
```

```
% java Experiment 10
0.08614716385210452
0.09054270895414829
0.10708746304898642
0.21166190071646818
0.363292849257276
0.460954145685913
0.5340026311350087
0.7216129793703496
0.9003500354411443
0.9293994908845686
```

SORTING STRINGS

```
public class StringSorter
{
   public static void main(String[] args)
   {
      String[] a = StdIn.readAllStrings();
      Insertion.sort(a);
      for (int i = 0; i < a.length; i++)
         StdOut.println(a[i]);
   }
}
```

```
% more words3.txt
bed bug dad yet zoo ... all bad yes

% java StringSorter < words3.txt
all bad bed bug dad ... yes yet zoo
[suppressing newlines]
```

SORTING FILES

```java
import java.io.File;

public class FileSorter
{
   public static void main(String[] args)
   {
      File directory = new File(args[0]);
      File[] files = directory.listFiles();
      Insertion.sort(files);
      for (int i = 0; i < files.length; i++)
         StdOut.println(files[i].getName());
   }
}
```

```
% java FileSorter .
Insertion.class
Insertion.java
InsertionX.class
InsertionX.java
Selection.class
Selection.java
Shell.class
Shell.java
ShellX.class
ShellX.java
```

HOW CAN WE ACHIEVE A COMMON INTERFACE?

CALLBACKS: ROADMAP

client

```
public class StringSorter
{
   public static void main(String[] args)
   {
      String[] a = StdIn.readAllStrings();
      Insertion.sort(a);
      for (int i = 0; i < a.length; i++)
         StdOut.println(a[i]);
   }
}
```

data-type implementation

```
public class String
implements Comparable<String>
{
   ...
   public int compareTo(String b)
   {
      ...
      return -1;
      ...
      return +1;
      ...
      return 0;
   }
}
```

GENERICS

Comparable interface (built in to Java)

```
public interface Comparable<Item>
{
   public int compareTo(Item that);
}
```

sort implementation

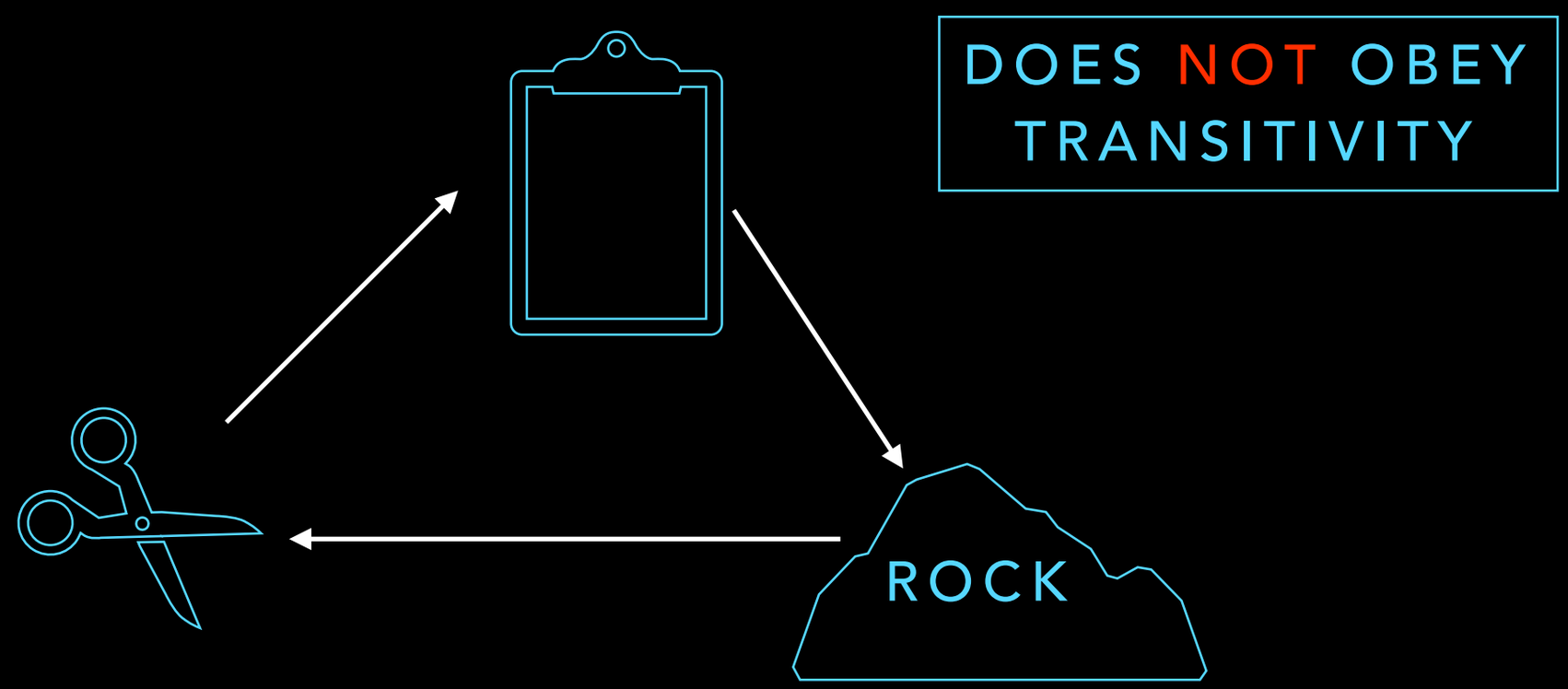
```
public static void sort(Comparable[] a)
{
   int N = a.length;
   for (int i = 0; i < N; i++)
      for (int j = i; j > 0; j--)
         if (a[j].compareTo(a[j-1]) < 0)
            exch(a, j, j-1);
         else break;
}
```

key point: no dependence on String data type

THE COMPARETO METHOD SHOULD IMPLEMENT TOTAL ORDER

Total order is a binary relation $\leq$ that satisfies:

- Antisymmetry: if both $v \leq w$ and $w \leq v$, then $v = w$.
- Transitivity: if both $v \leq w$ and $w \leq x$, then $v \leq x$.
- Totality: either $v \leq w$ or $w \leq v$ or both.



IMPLEMENTING COMPARETO()

Implement `compareTo()` so that `v.compareTo(w)`

v < w	RETURN -1
v > w	RETURN 1
v == w	RETURN 0

EX. IMPLEMENTING COMPARIABLE

Date data type. Simplified version of java.util.Date.

```
public class Date implements Comparable<Date>
{
   private final int month, day, year;

   public Date(int m, int d, int y)
   {
      month = m;
      day   = d;
      year  = y;
   }

   public int compareTo(Date that)
   {
      if (this.year  < that.year ) return -1;
      if (this.year  > that.year ) return +1;
      if (this.month < that.month) return -1;
      if (this.month > that.month) return +1;
      if (this.day   < that.day  ) return -1;
      if (this.day   > that.day  ) return +1;
      return 0;
   }
}
```

ONLY COMPARE DATES TO OTHER DATES

AN INTRANSITIVE ORDER

- Q. Does the following implement total order ?

```
public class Double implements Comparable<Double>
{
   private double x;


   …

   public int compareTo(Double that) {
       if      (this.x < that.x) return -1;
       else if (this.x > that.x) return +1;
   }
}
```

- A. Not totality (the zero case)

SORTING ALGORITHMS

- Why are there so many sorting algorithms
- Why hasn't just one won?
  - Insertion Sort
  - Selection Sort
  - Shell Sort
  - Merge Sort
  - Quick Sort
  - etc

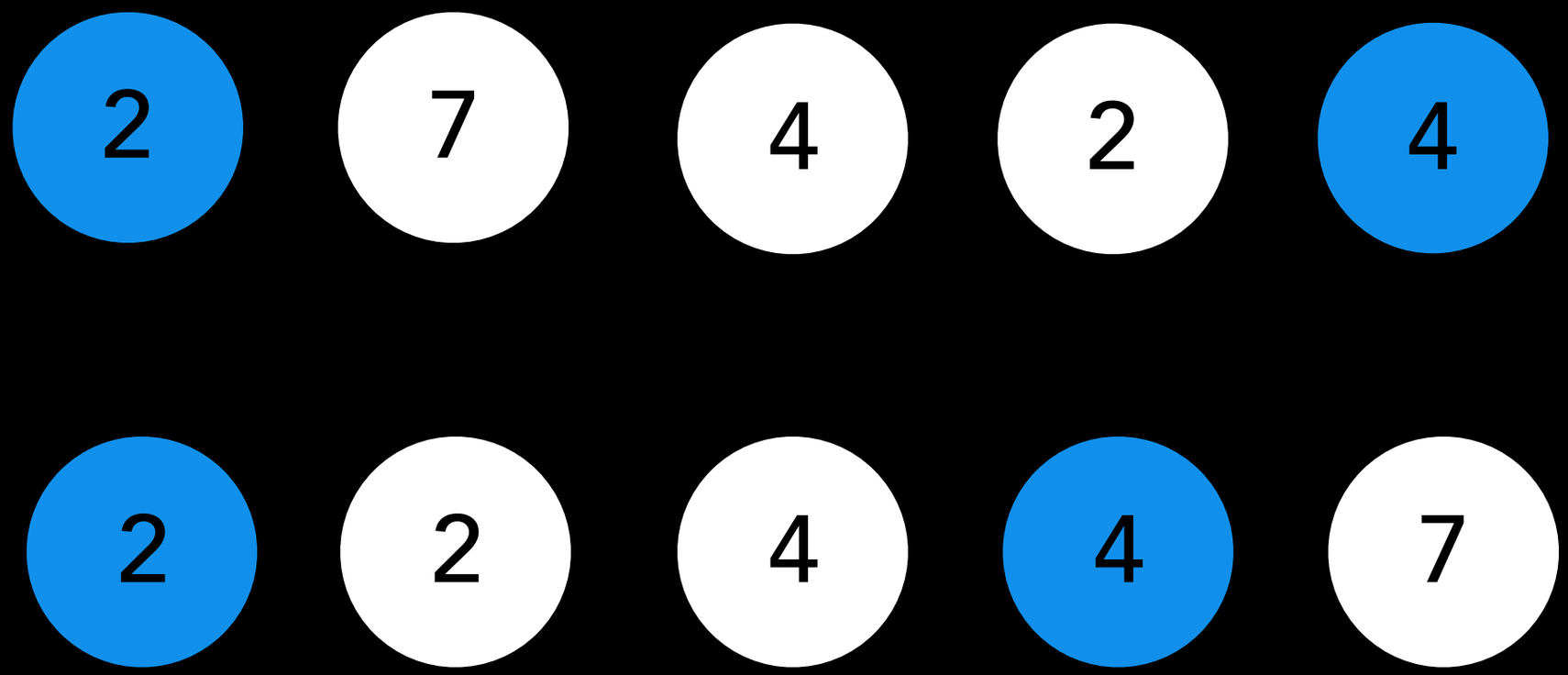
IT IS ABOUT TRADE OFF

ONE IMPORTANT TRADE OFF IS STABILITY

WHAT DOES IT MEAN FOR A SORTING ALGORITHM TO BE STABLE?

A SORTING ALGORITHM IS STABLE IF THE ORDER OF THE ELEMENTS WITH SAME KEY IS THE SAME AFTER SORTING

A SORTING ALGORITHM IS STABLE IF THE ORDER OF THE ELEMENTS WITH SAME KEY IS THE SAME AFTER SORTING



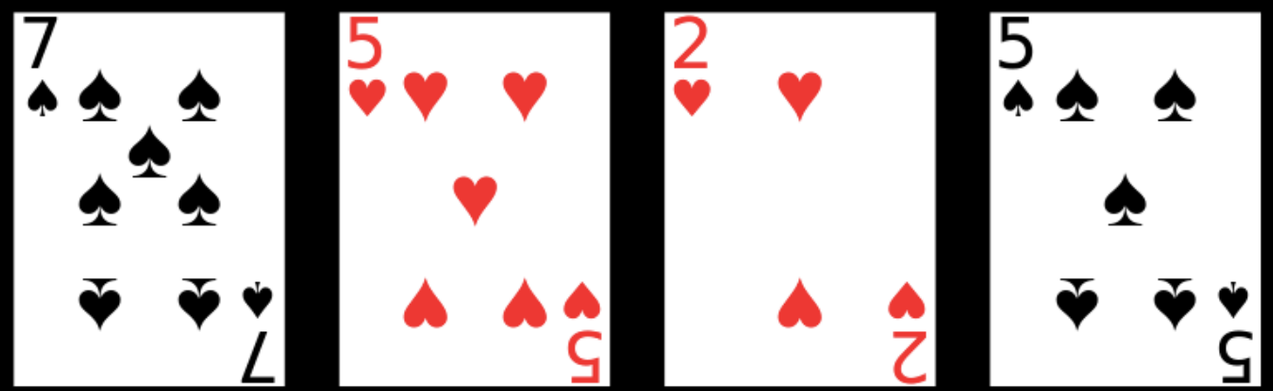
THE BLUE 2 IS STILL BEFORE THE WHITE 2

THE WHITE 4 IS STILL BEFORE THE BLUE 4

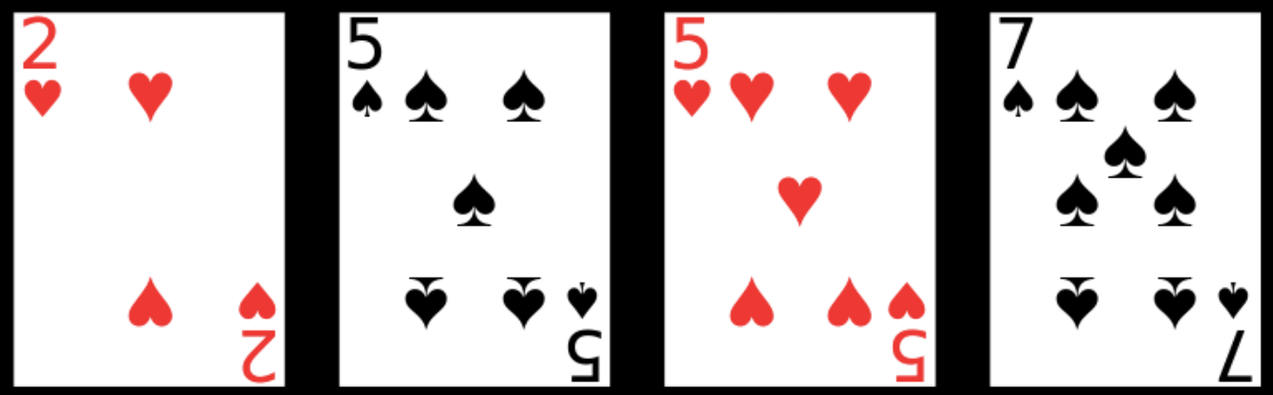
Stable

QUIZ: STABLE OR NOT?

Before Sort



After Sort



WHY IS STABILITY IMPORTANT?

QUIZ: STABLE OR NOT?

Was the sorting algorithm used to sort grades stable?

SORT BY NAME

Andrews	3	A	664-480-0023
Battle	4	C	874-088-1212
Chen	3	A	991-878-4944
Furia	1	A	766-093-9873
Gazsi	4	B	766-093-9873
Kanaga	3	B	898-122-9643
Rohde	2	A	232-343-5555

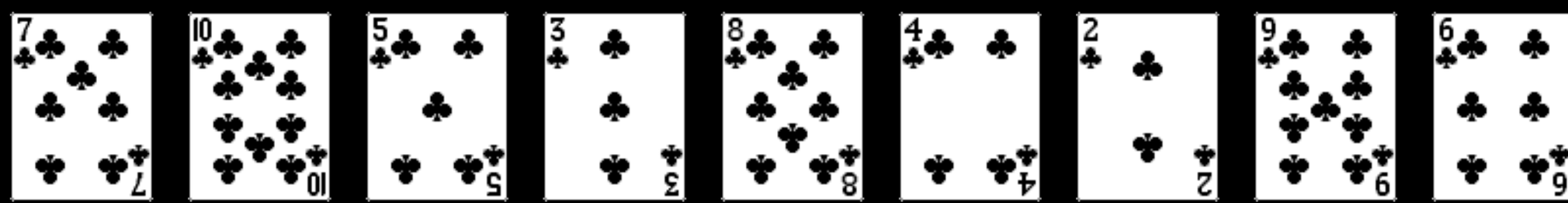
SORT BY GRADE

Andrews	3	A	664-480-0023
Chen	3	A	991-878-4944
Furia	1	A	766-093-9873
Rohde	2	A	232-343-5555
Gazsi	4	B	766-093-9873
Kanaga	3	B	898-122-9643
Battle	4	C	874-088-1212

SELECTION SORT

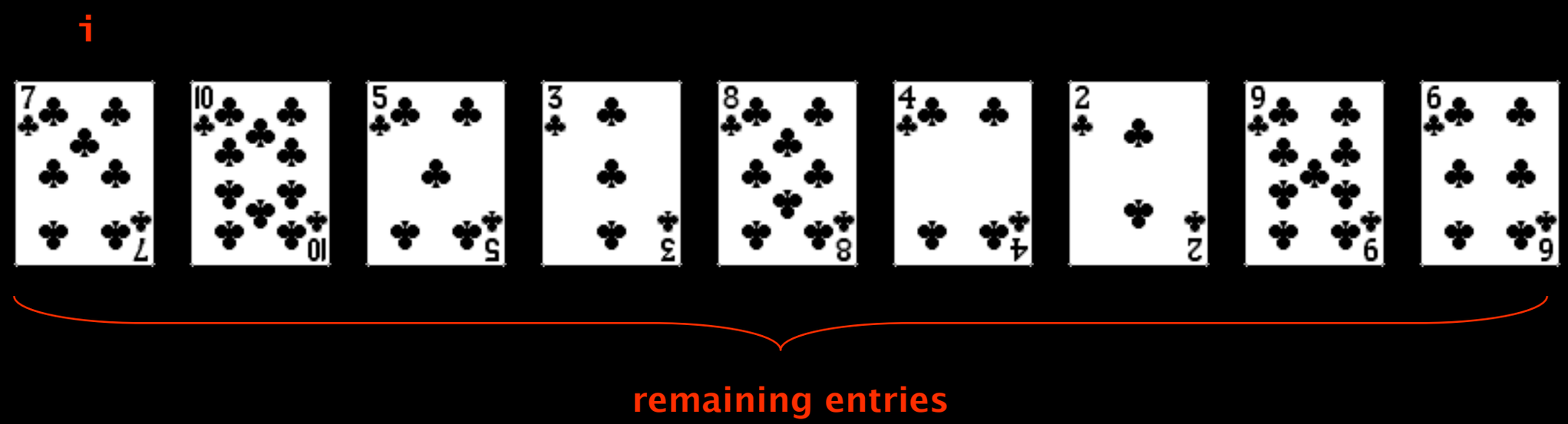
SELECTION SORT

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].

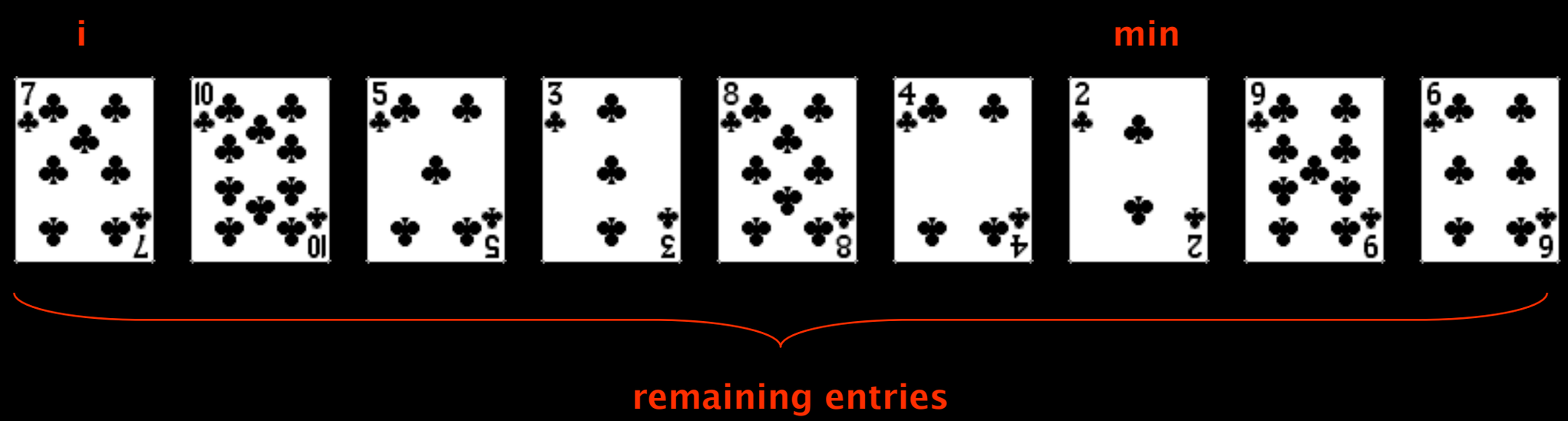


initial

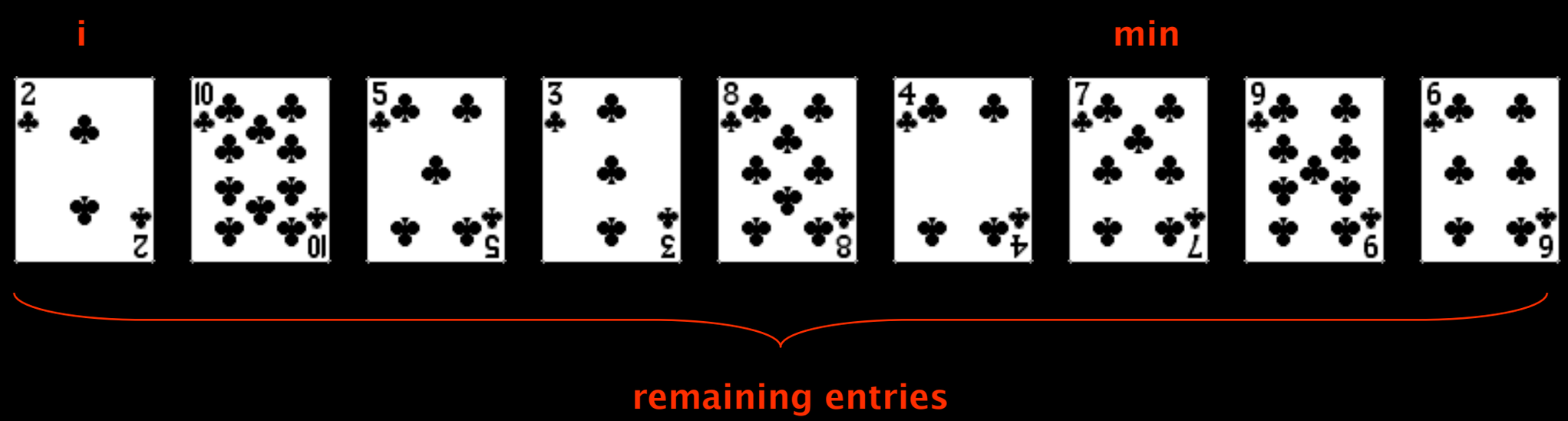
- In iteration `i`, find index `min` of smallest remaining entry.
- Swap `a[i]` and `a[min]`.



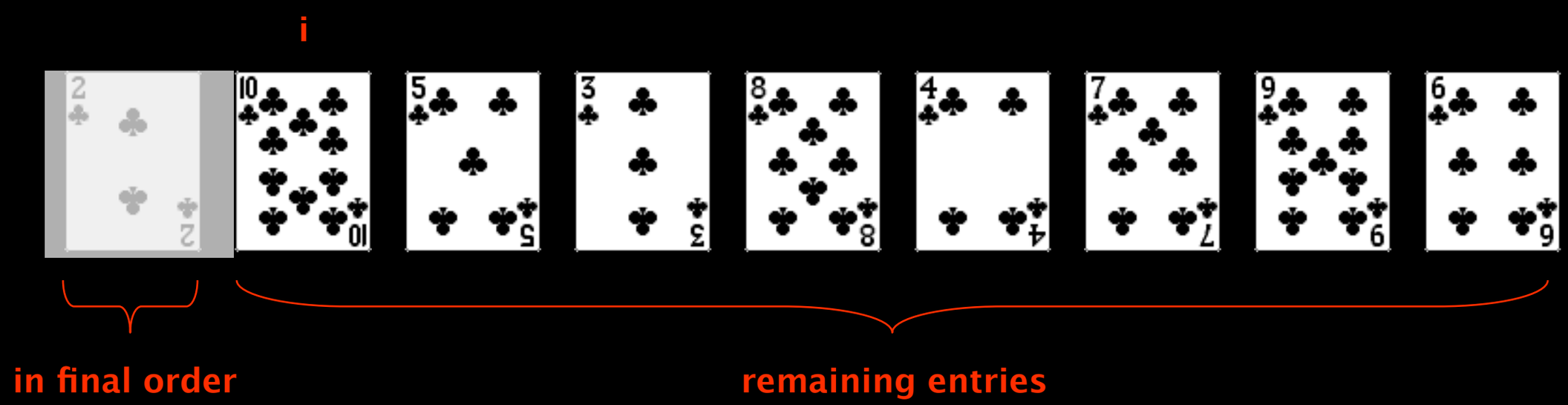
- In iteration `i`, find index `min` of smallest remaining entry.
- Swap `a[i]` and `a[min]`.

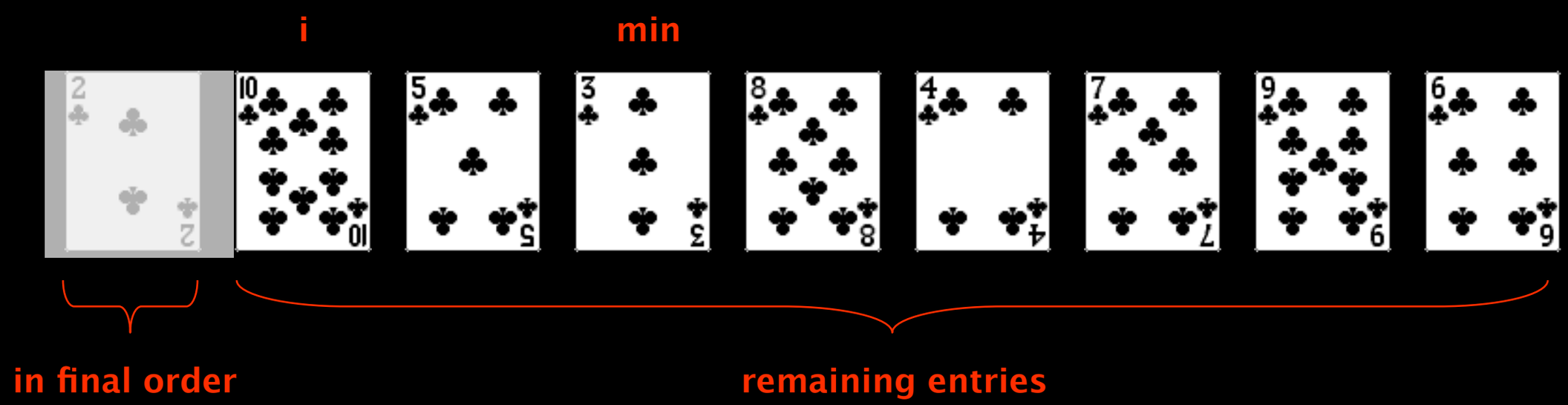


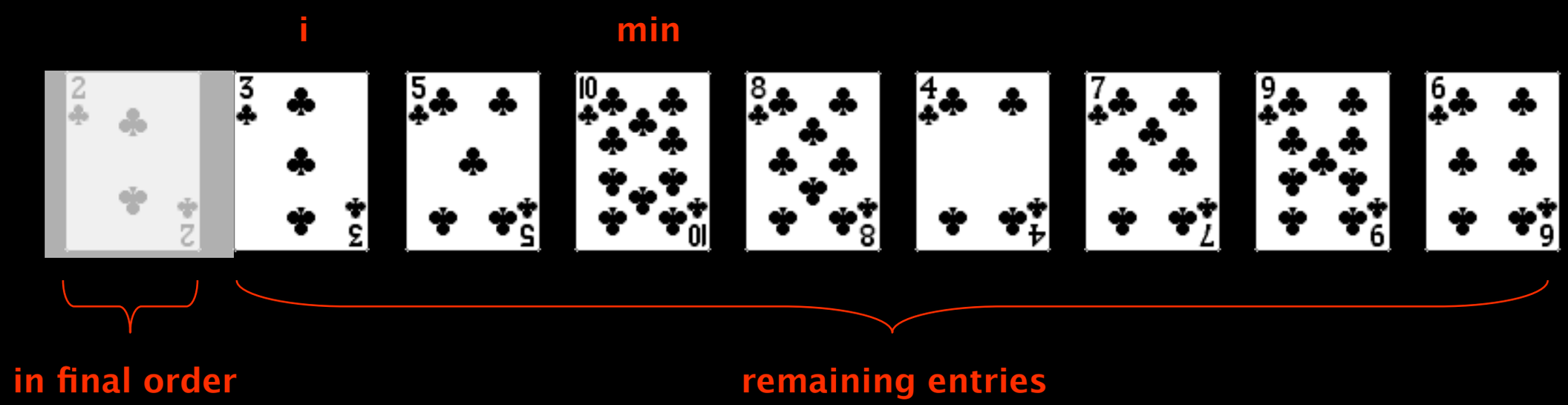
- In iteration `i`, find index `min` of smallest remaining entry.
- Swap `a[i]` and `a[min]`.

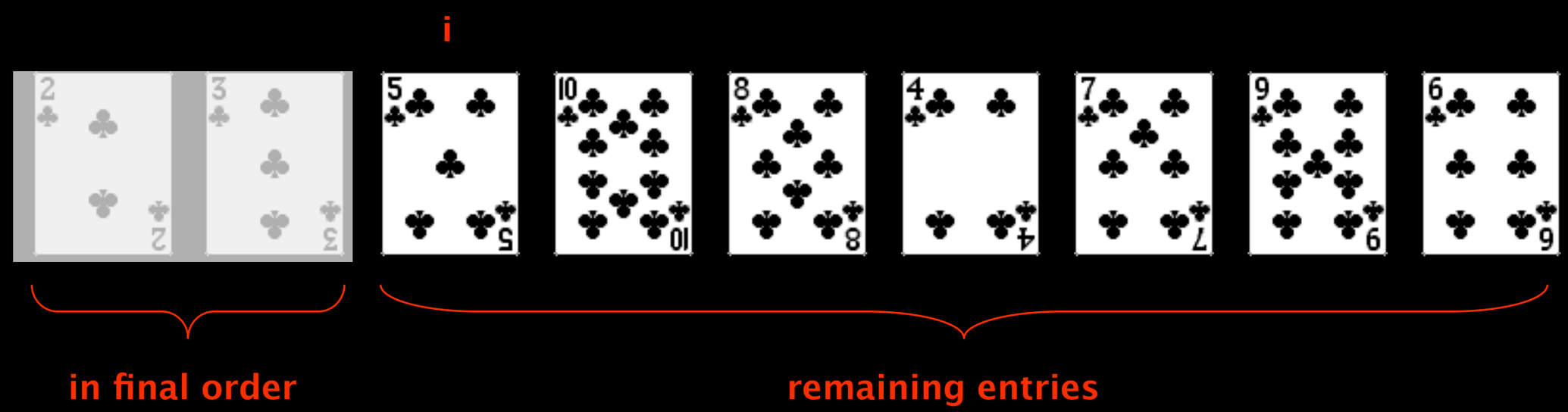


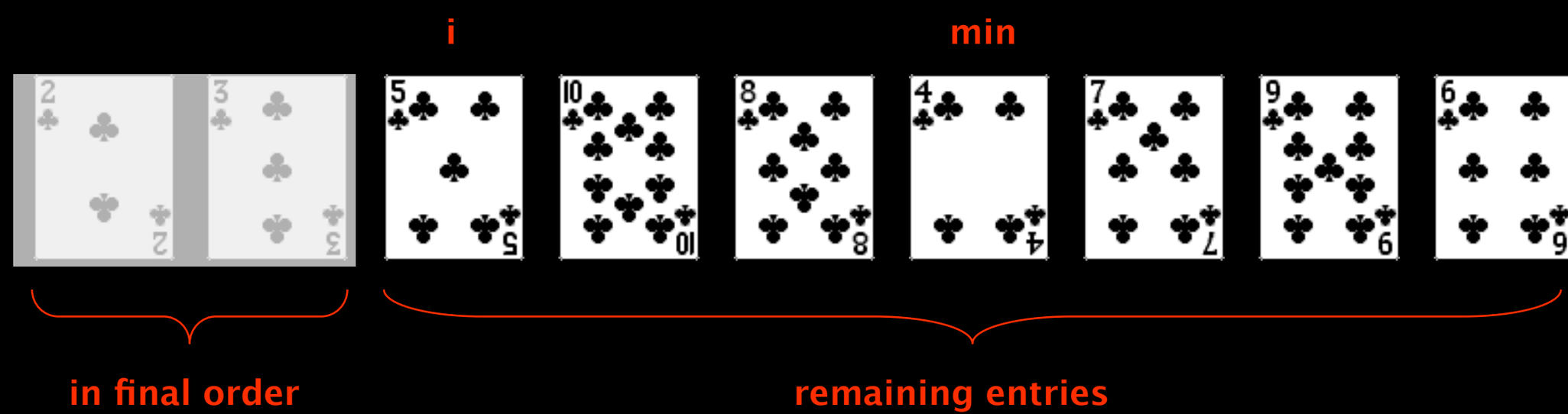
- In iteration `i`, find index `min` of smallest remaining entry.
- Swap `a[i]` and `a[min]`.

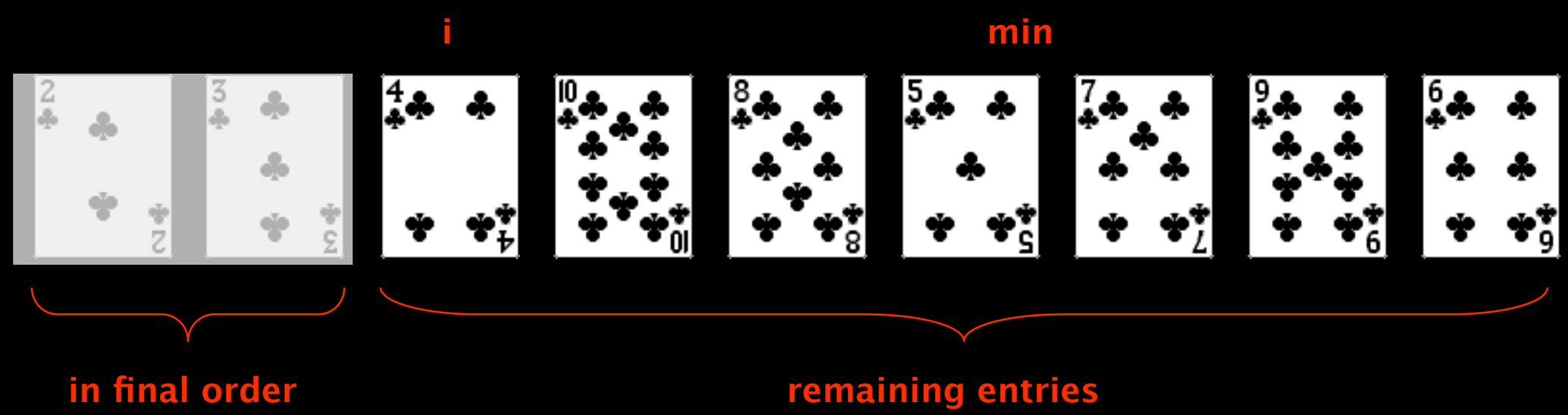


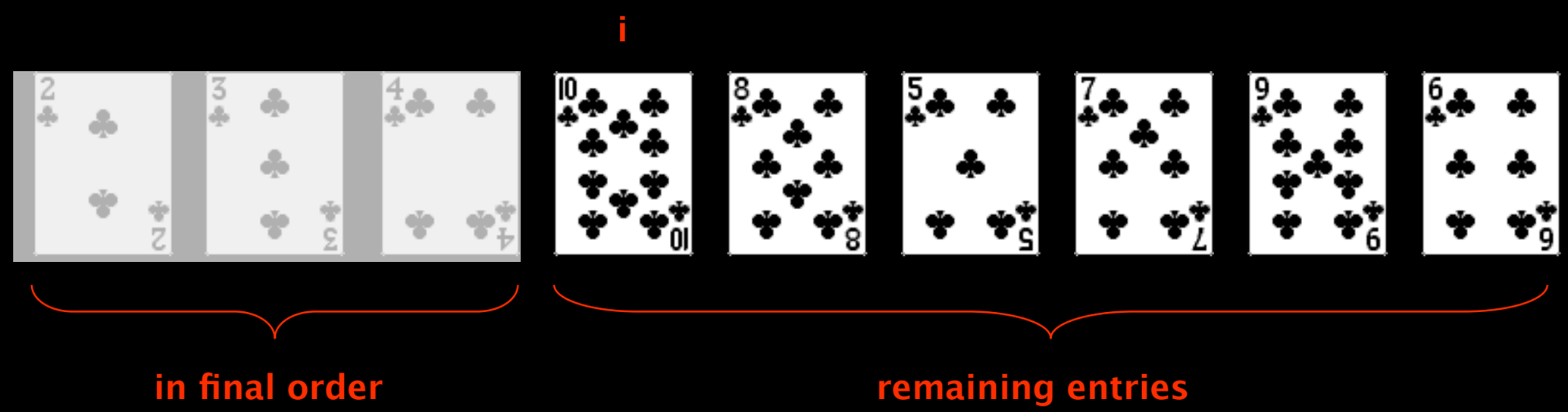

i
min
in final order
remaining entries

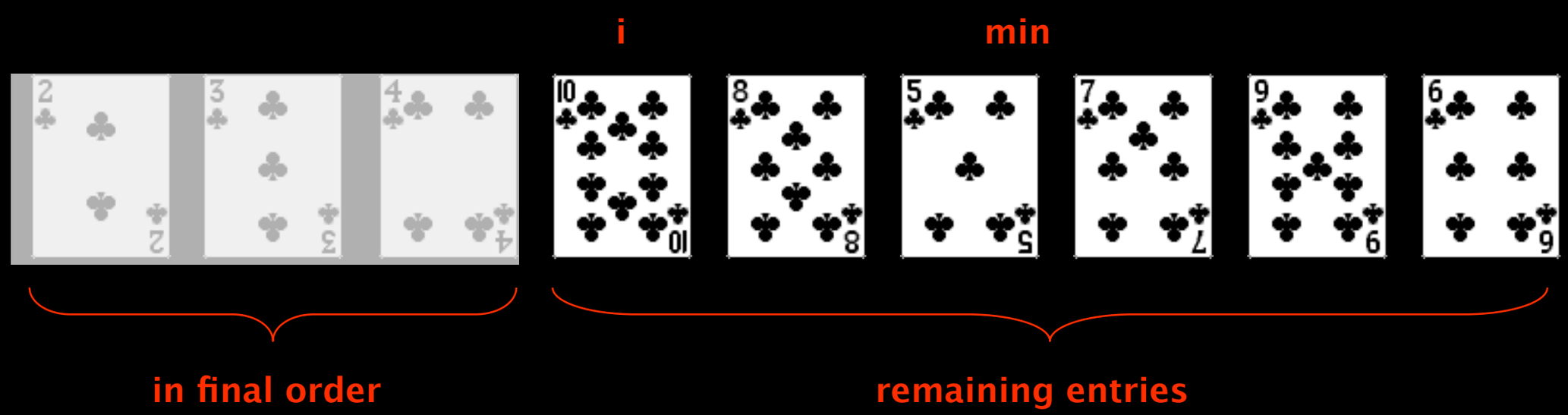

i
min
in final order
remaining entries

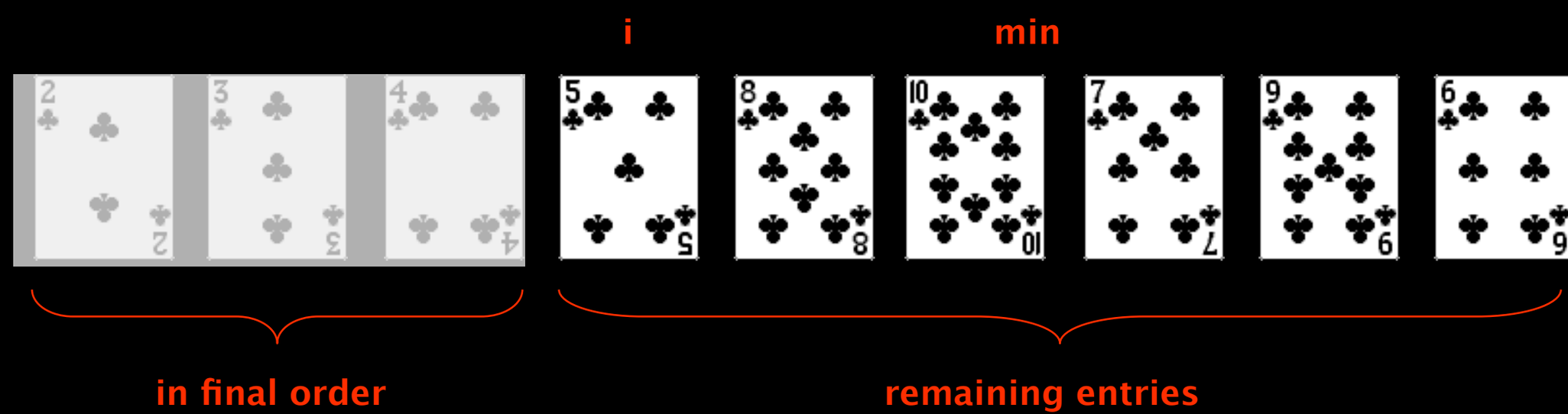

i
in final order
remaining entries

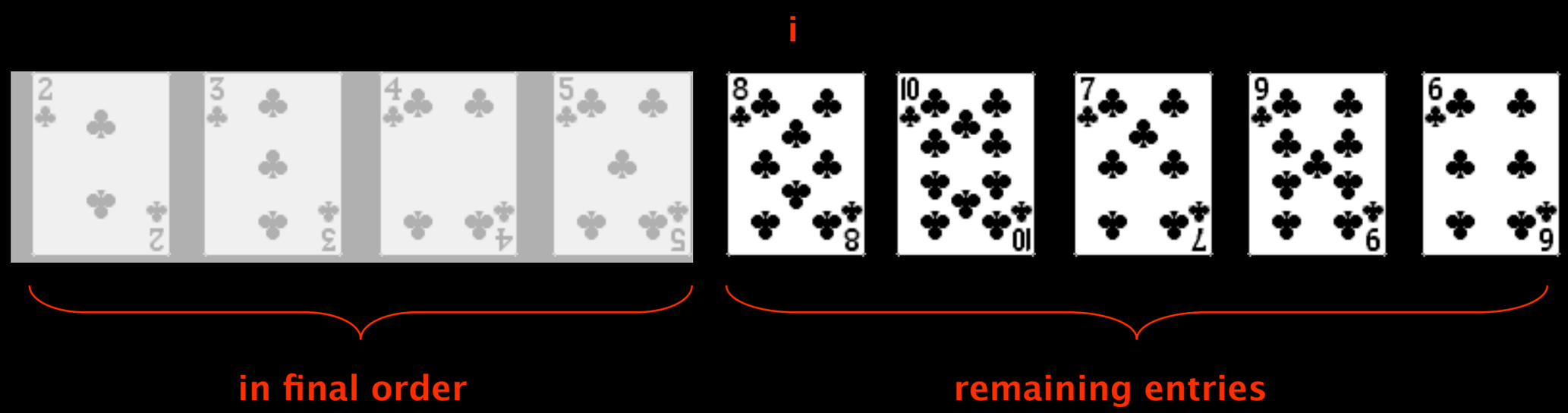

i
min
in final order
remaining entries

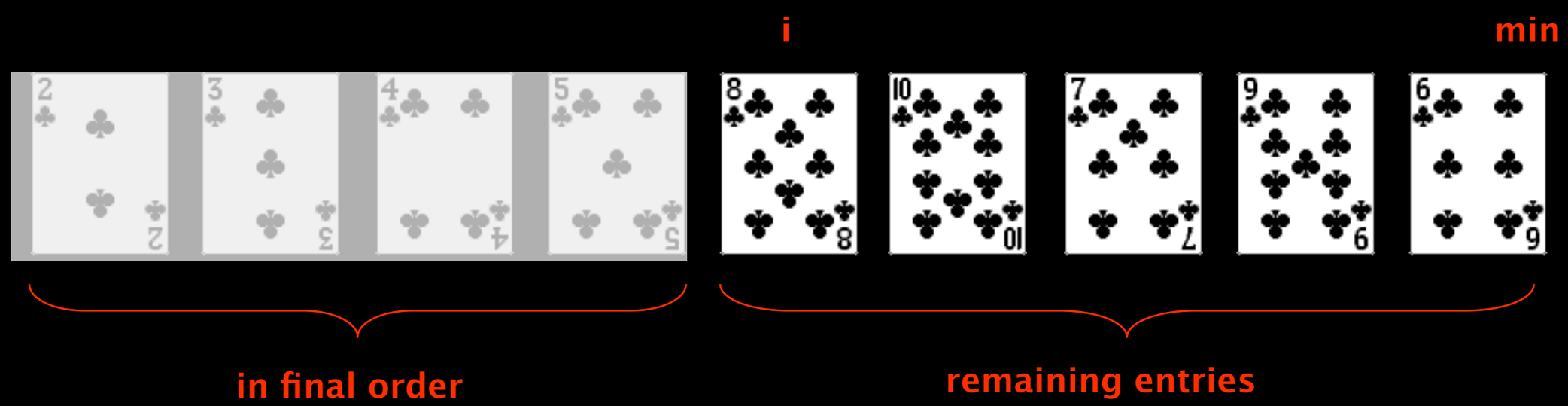

i
min
in final order
remaining entries

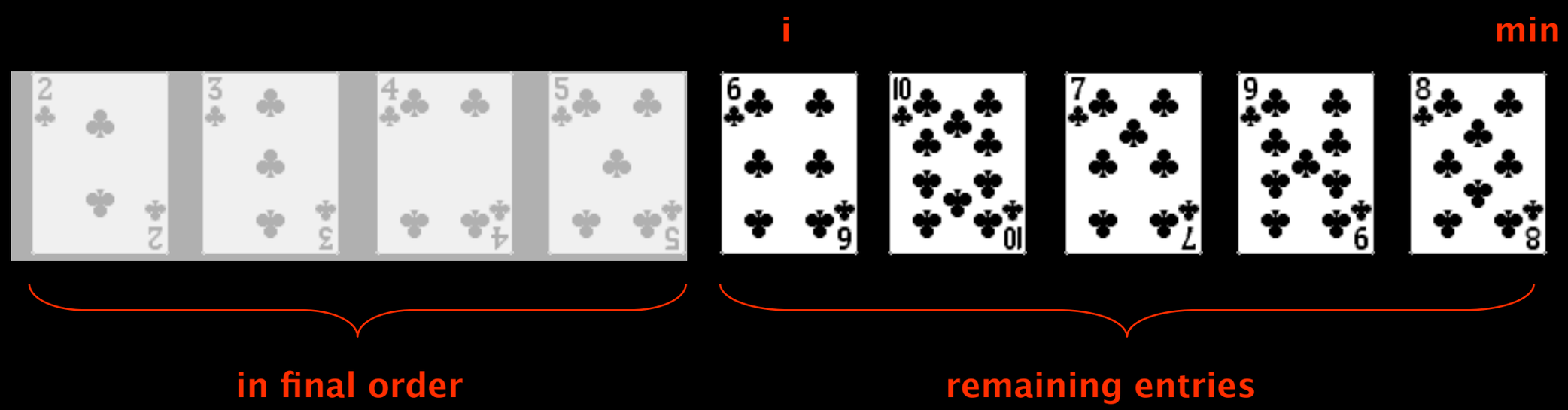

i
in final order
remaining entries

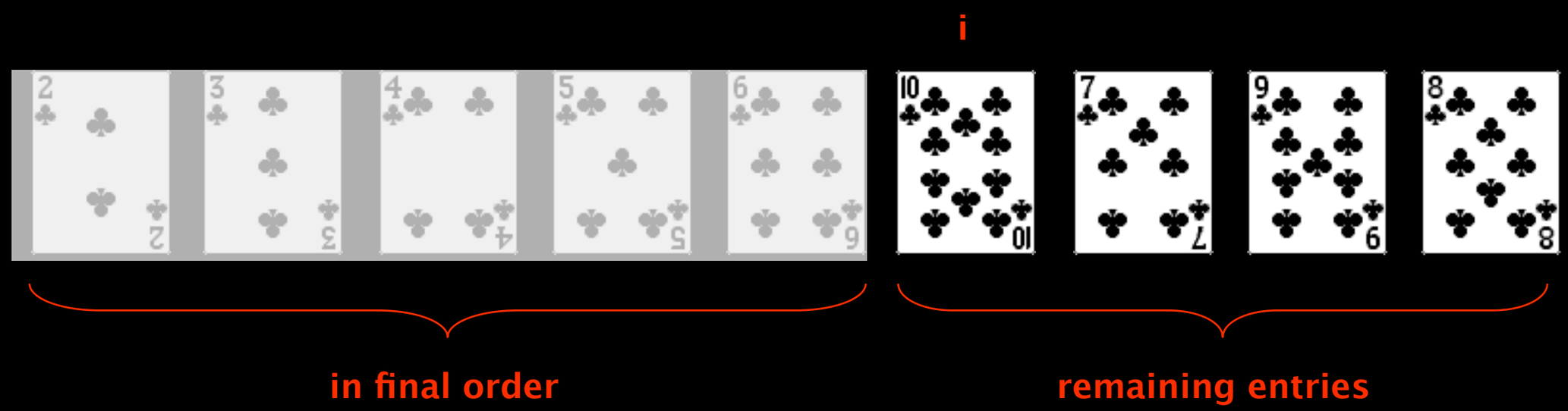

i
min
in final order
remaining entries

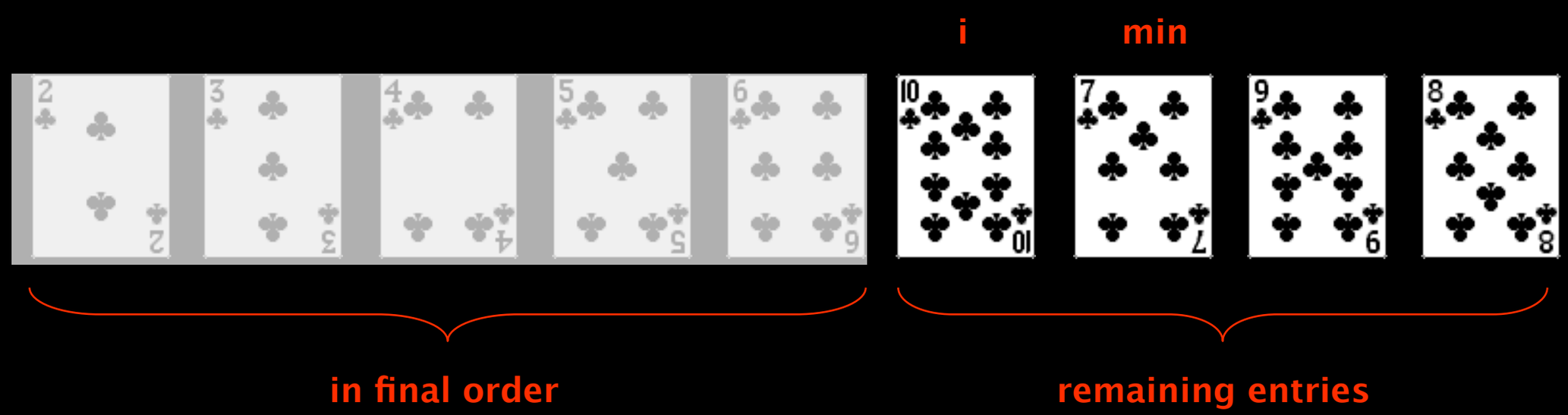

i
min
in final order
remaining entries

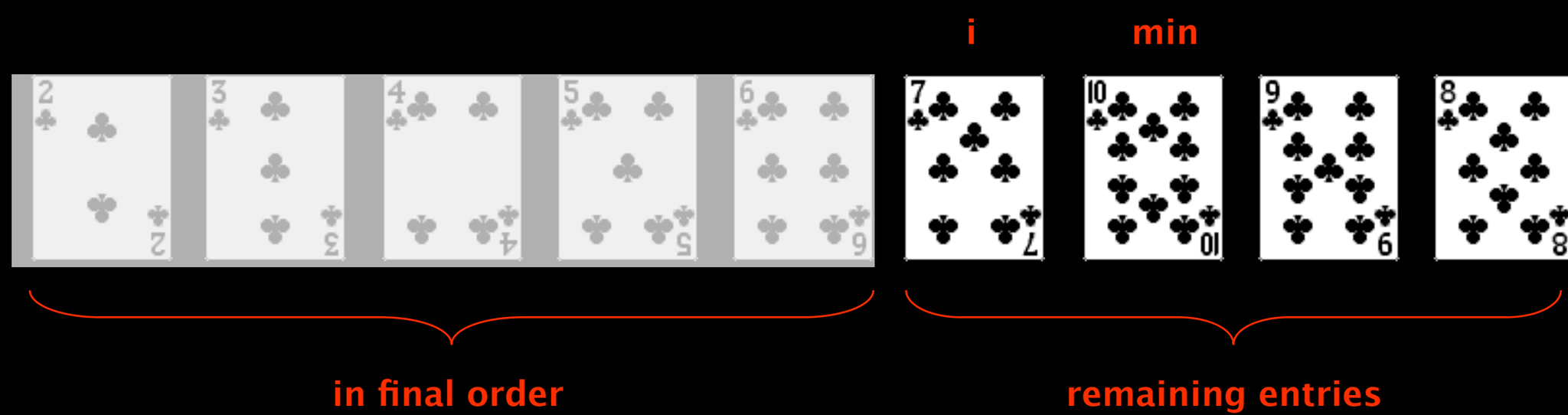

i
in final order
remaining entries

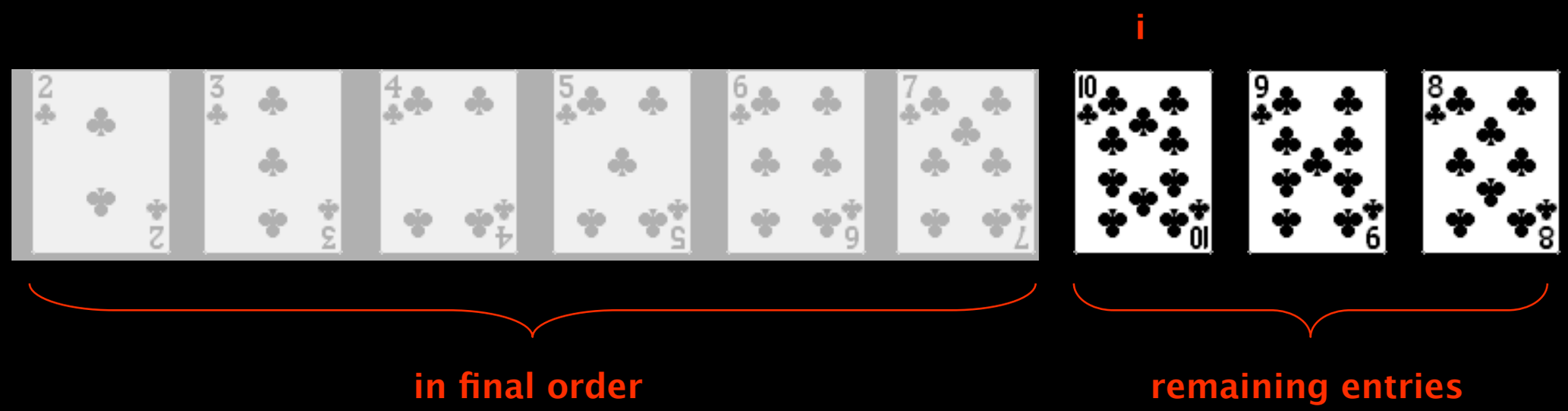

i
min
in final order
remaining entries

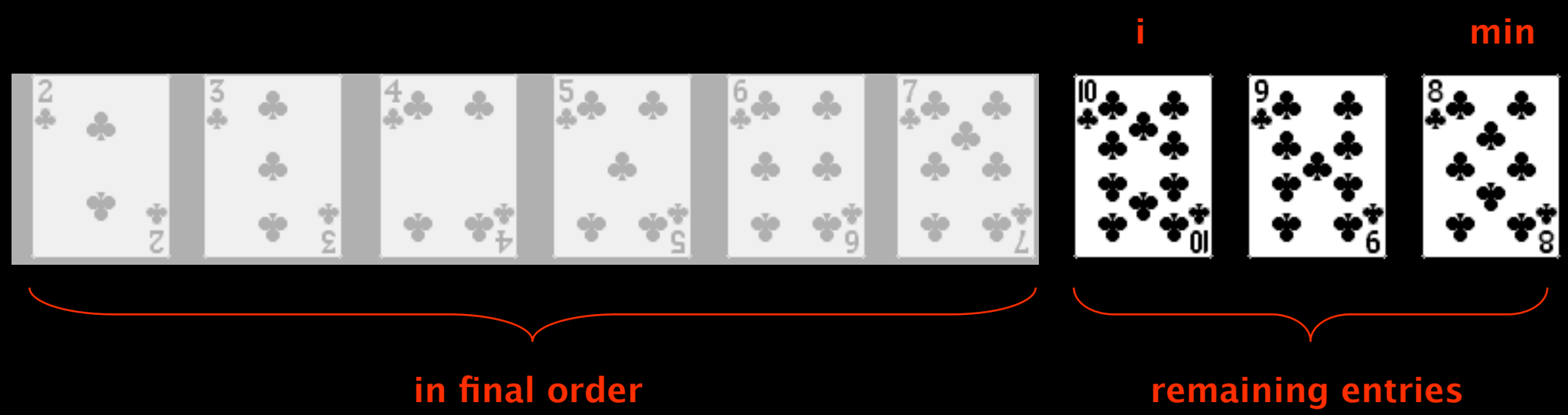

i
min
in final order
remaining entries

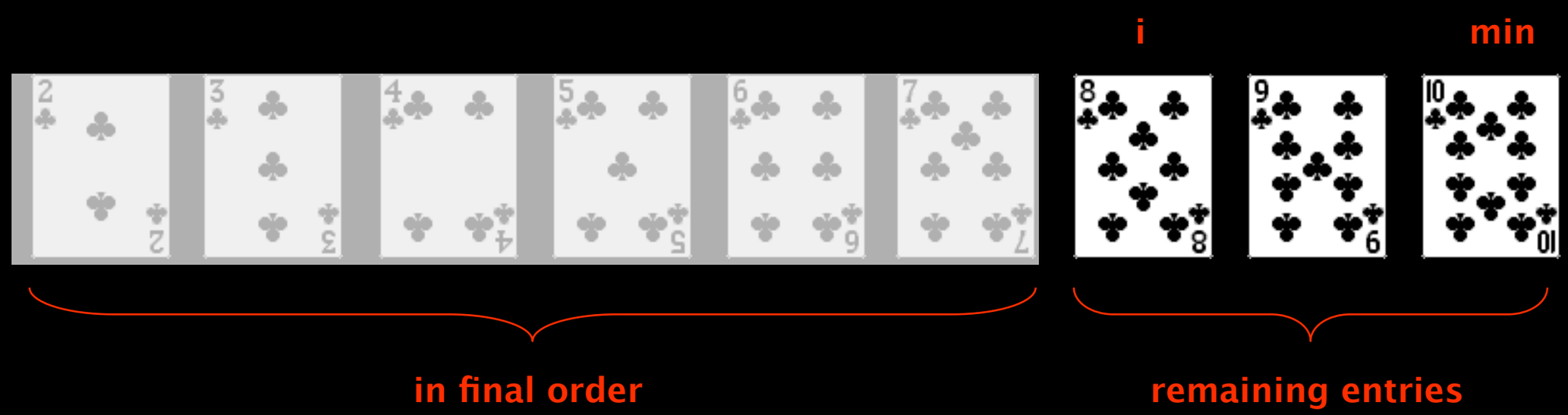

i
in final order
remaining entries

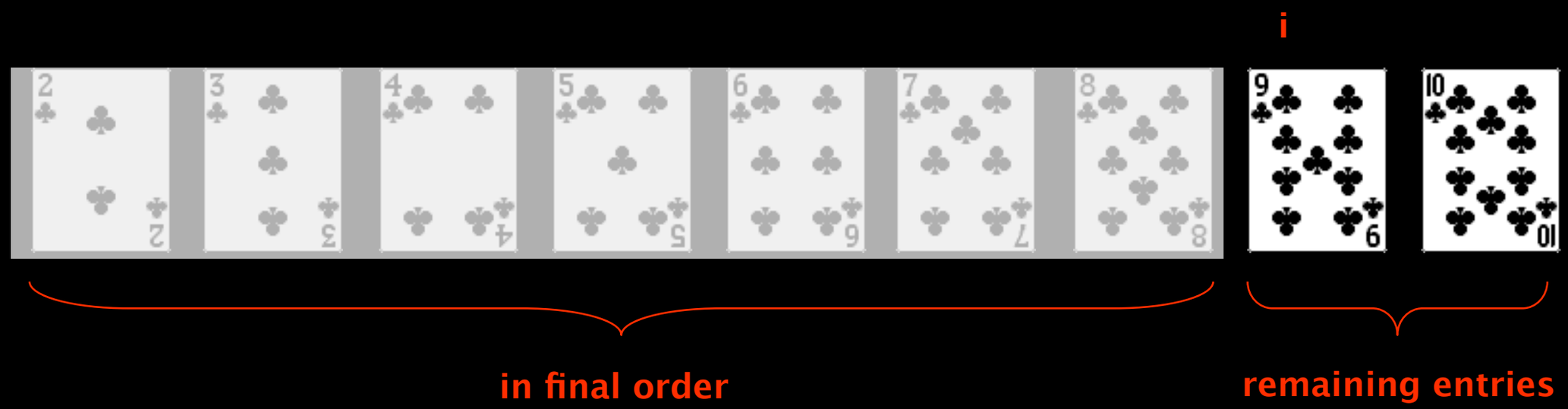

i
min
in final order
remaining entries

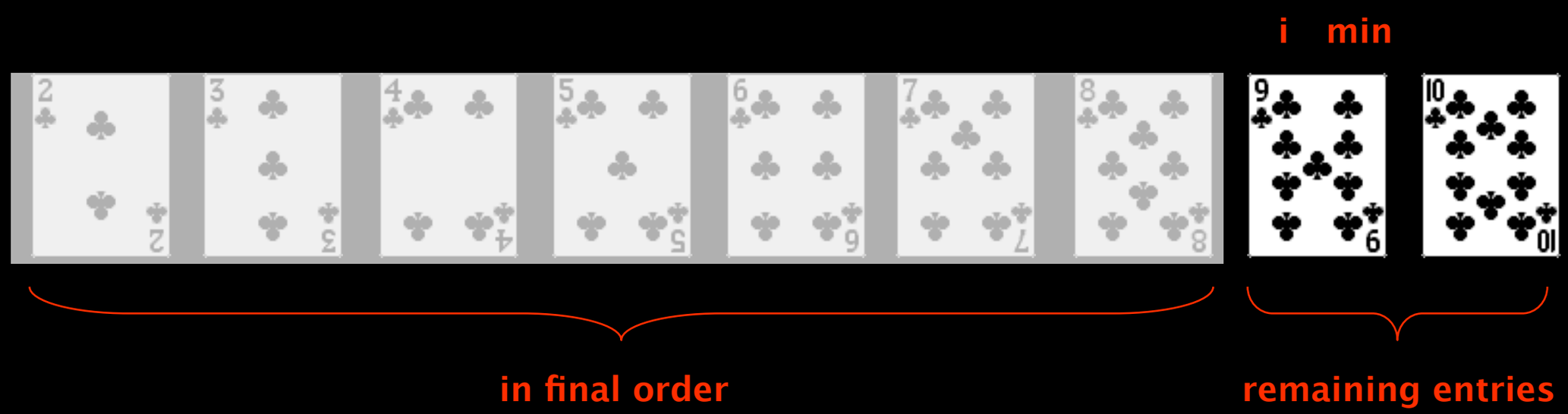

i
min
in final order
remaining entries

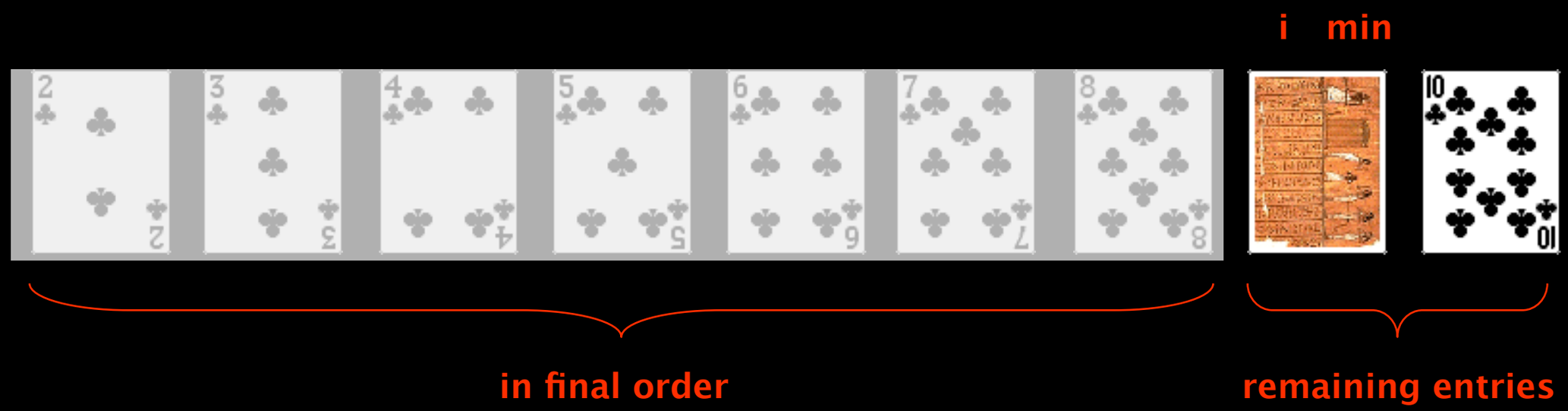

i
in final order
remaining entries

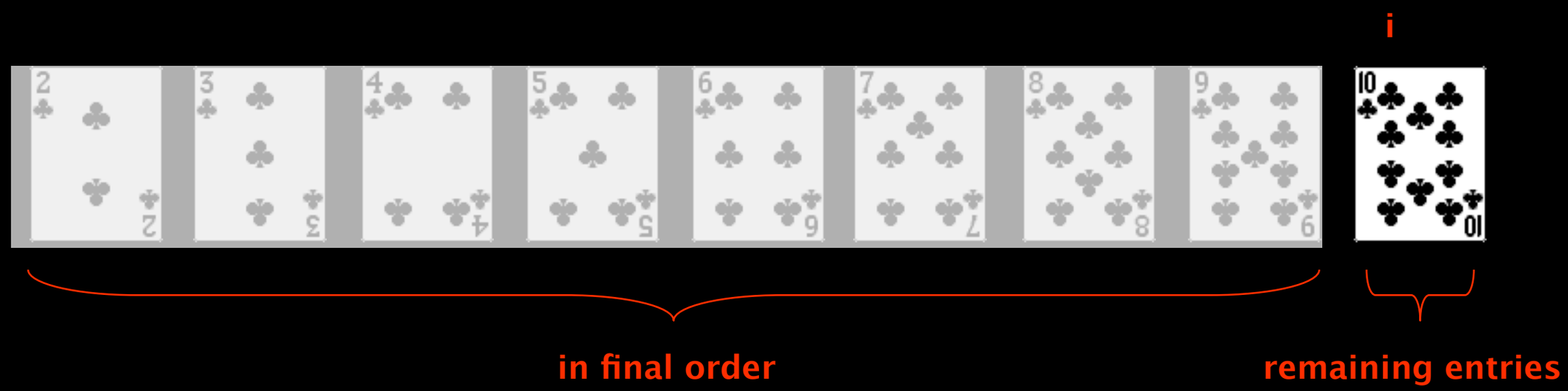

i
min
in final order
remaining entries

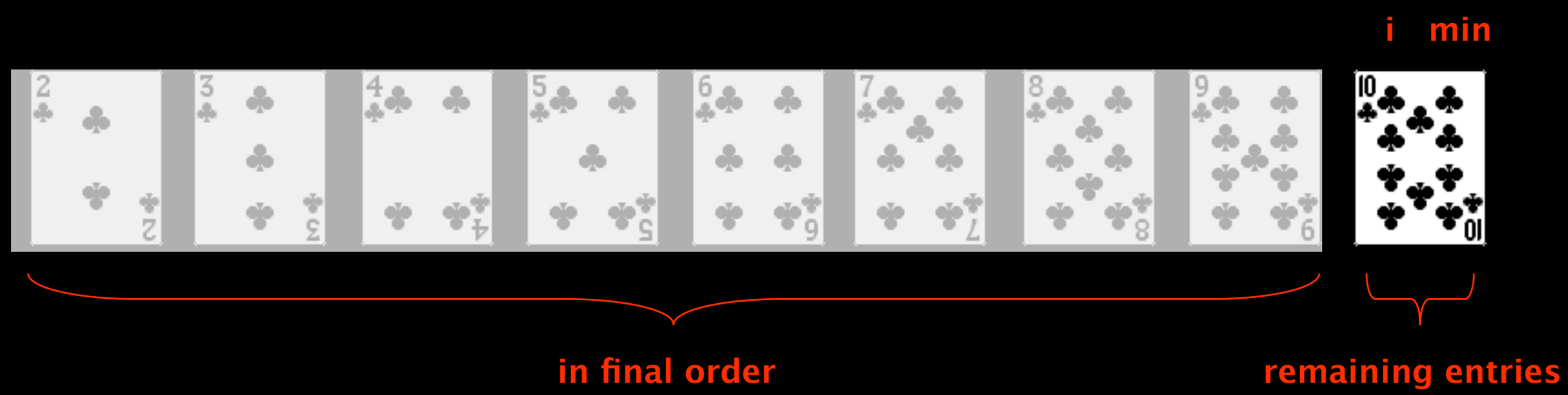

i
min
in final order
remaining entries

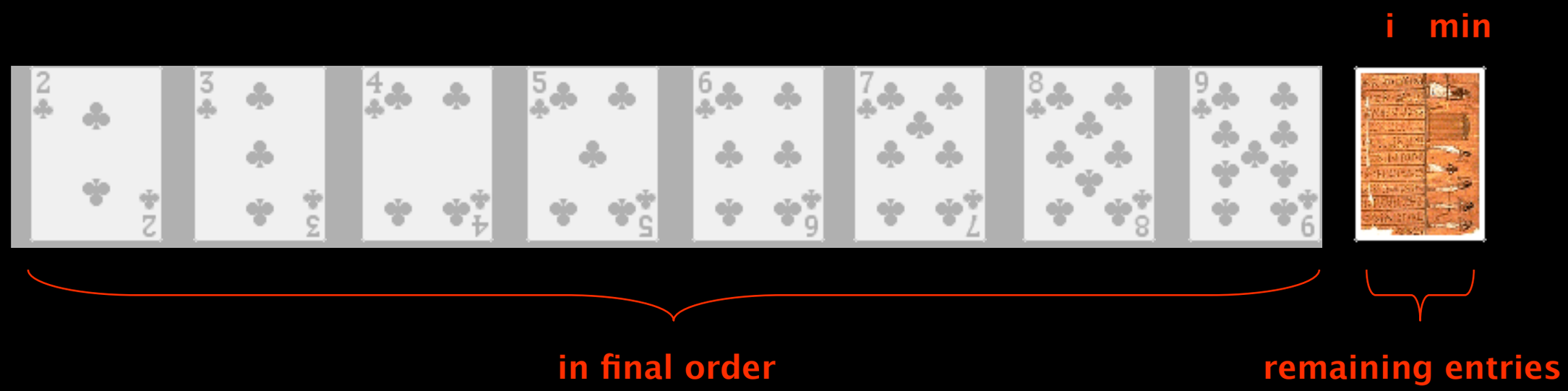

i
in final order
remaining entries

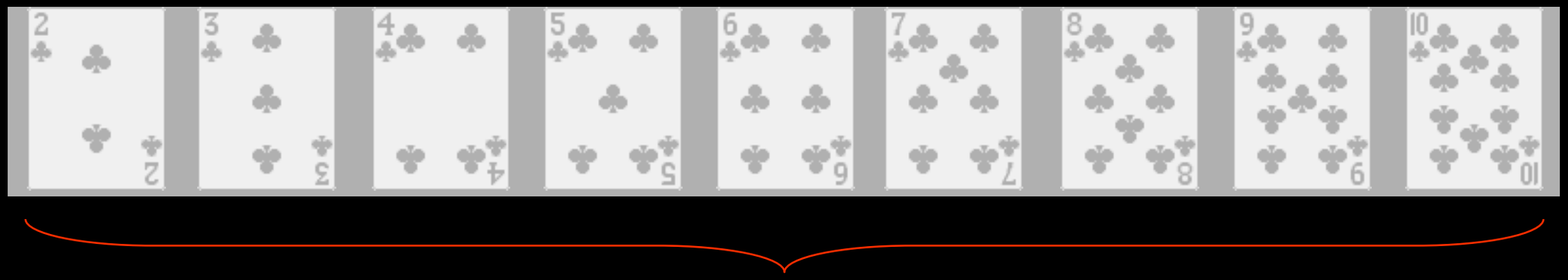

i min
in final order
remaining entries

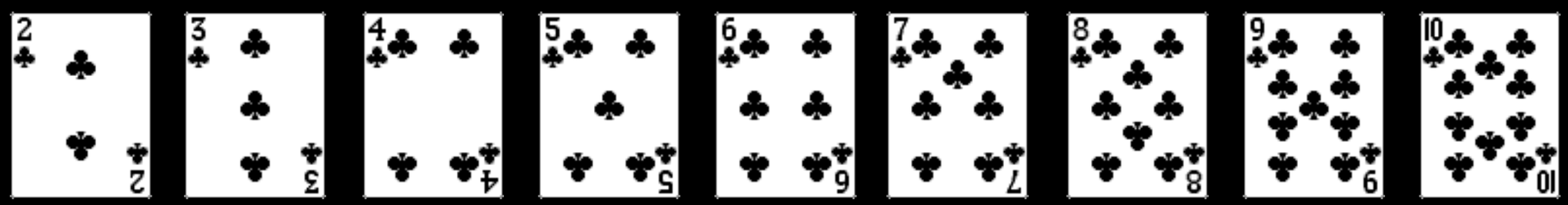

i min
in final order
remaining entries


i
in final order
remaining entries


i min
in final order
remaining entries


i min
in final order
remaining entries

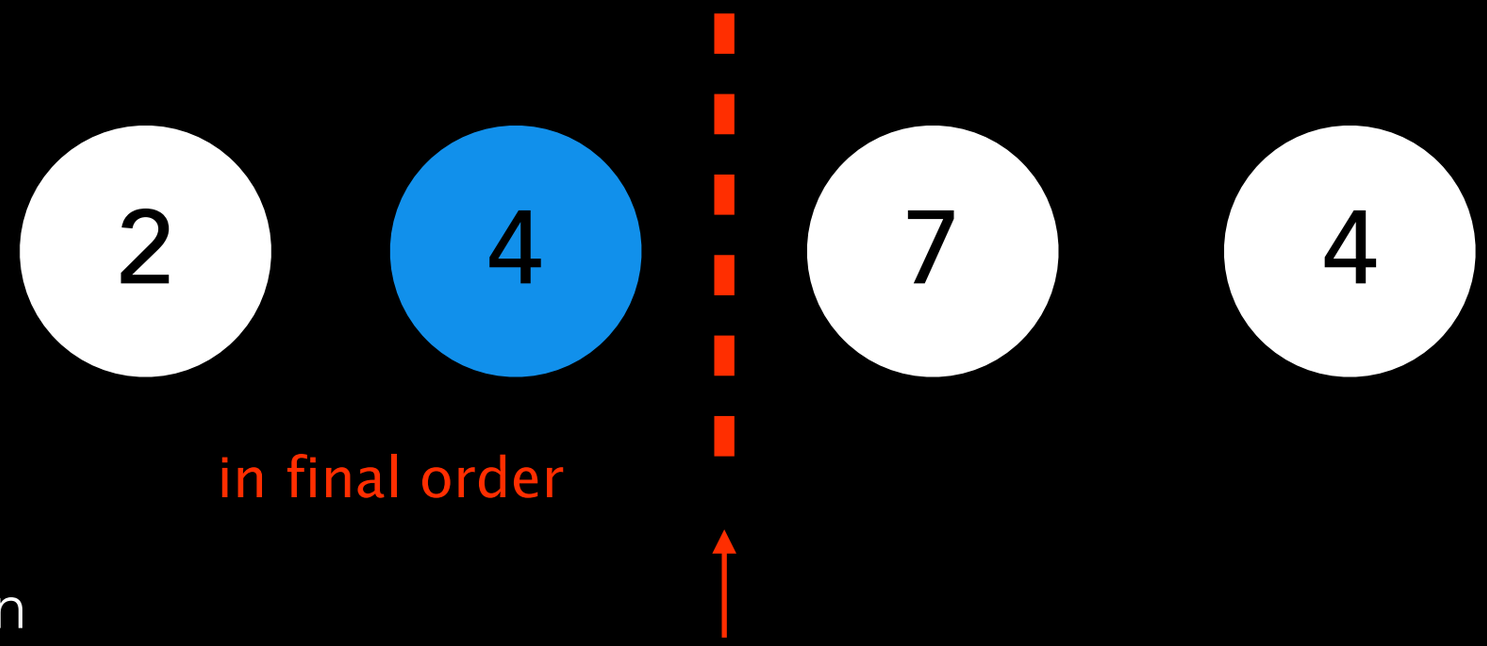

in final order



sorted

SELECTION SORT

- Algorithm. ↑ scans from left to right.



- Invariants.
  - Entries the left of ↑ (including ↑) fixed and in ascending order.
  - No entry to right of ↑ is smaller than any entry to the left of ↑.

LET'S IMPLEMENT SELECTION SORT

Two useful sorting abstractions(functions)

Helper functions. Refer to data through compares and exchanges.

Less. Is item v less than w ?

```
private static boolean less(Comparable v, Comparable w)
{  return v.compareTo(w) < 0;  }
```

Exchange. Swap item in array a[] at index i with the one at index j.

```
private static void exch(Comparable[] a, int i, int j)
{
   Comparable swap = a[i];
   a[i] = a[j];
   a[j] = swap;
}
```

Selection sort inner loop

- Identify index of minimum entry on right.

```
int min = i;
for (int j = i+1; j < N; j++)
   if (less(a[j], a[min]))
      min = j;
```

- Exchange into position.

```
exch(a, i, min);
```

- Move the pointer to the right.

```
i++;
```

SELECTION SORT: JAVA IMPLEMENTATION

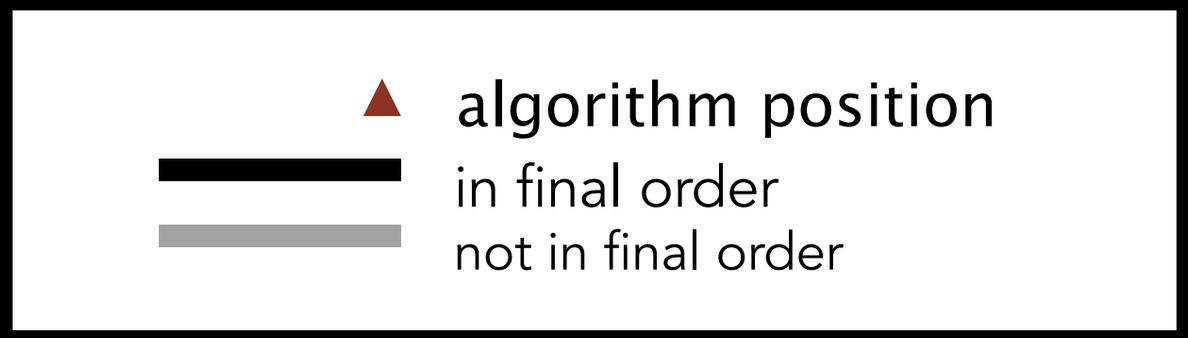
```
public class Selection
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;
      for (int i = 0; i < N; i++)
      {
         int min = i;
         for (int j = i+1; j < N; j++)
            if (less(a[j], a[min]))
               min = j;
         exch(a, i, min);
      }
   }

   private static boolean less(Comparable v, Comparable w)
   {  /* as before */  }

   private static void exch(Comparable[] a, int i, int j)
   {  /* as before */  }
}
```

SELECTION SORT: ANIMATIONS

20 random items



http://www.sorting-algorithms.com/selection-sort

Selection sort: mathematical analysis

Proposition. Selection sort uses $(N-1)+(N-2)+\ \ldots+1+0\ \sim\ N^2/2$ compares and N exchanges.

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
		S	O	R	T	E	X	A	M	P	L	E
0	6	S	O	R	T	E	X	A	M	P	L	E
1	4	A	O	R	T	E	X	S	M	P	L	E
2	10	A	E	R	T	O	X	S	M	P	L	E
3	9	A	E	E	T	O	X	S	M	P	L	R
4	7	A	E	E	L	O	X	S	M	P	T	R
5	7	A	E	E	L	M	X	S	O	P	T	R
6	8	A	E	E	L	M	O	S	X	P	T	R
7	10	A	E	E	L	M	O	P	X	S	T	R
8	8	A	E	E	L	M	O	P	R	S	T	X
9	9	A	E	E	L	M	O	P	R	S	T	X
10	10	A	E	E	L	M	O	P	R	S	T	X
		A	E	E	L	M	O	P	R	S	T	X

entries in black are examined to find the minimum

entries in red are a[min]

entries in gray are in final position

Trace of selection sort (array contents just after each exchange)

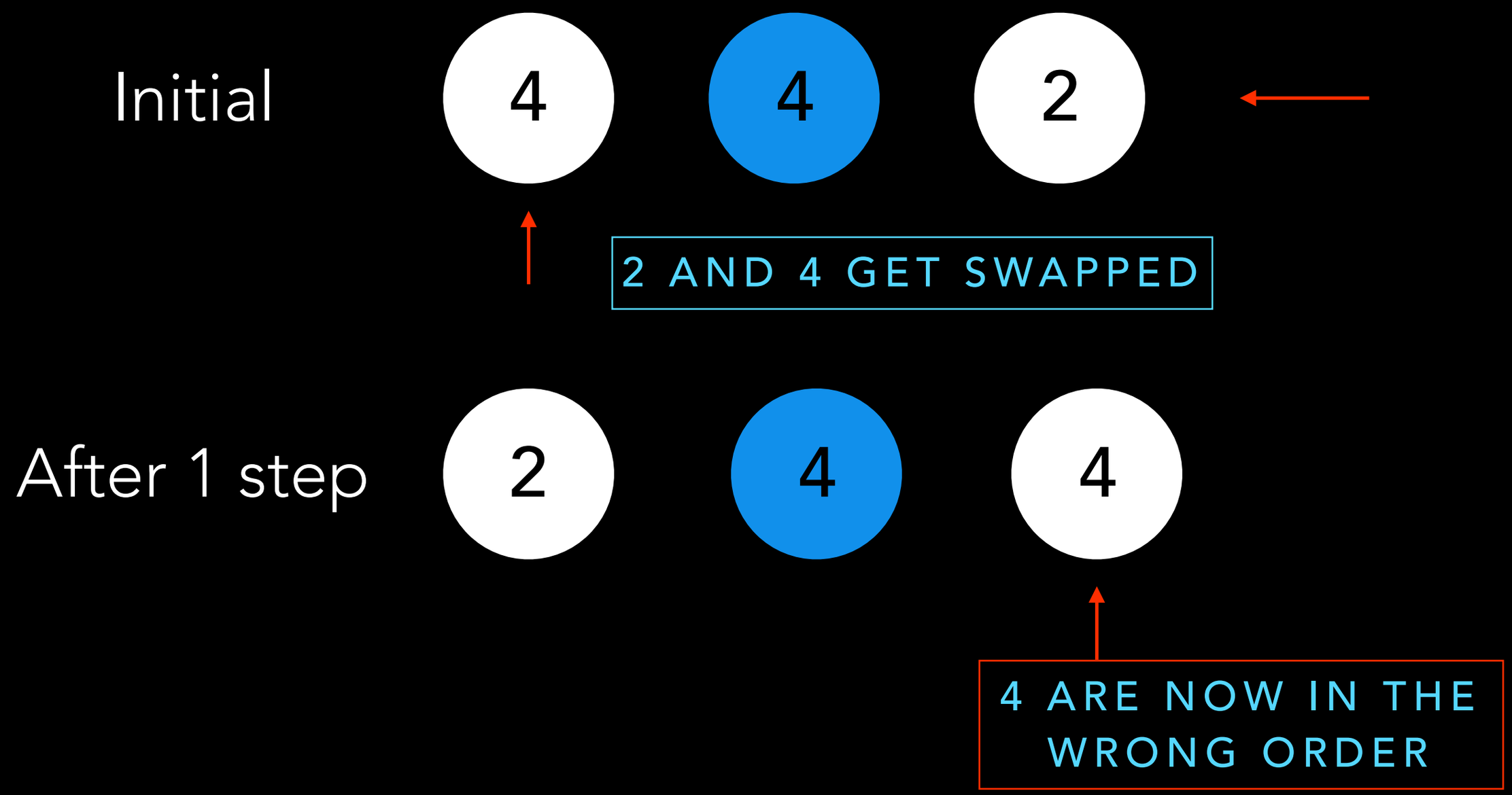
Running time insensitive to input. Quadratic time, even if input is sorted.

Data movement is minimal. Linear number of exchanges.

IS SELECTION SORT STABLE

SELECTION SORT IS NOT STABLE

SELECTION SORT IS NOT STABLE



STOP & THINK

Q: Given a list of non negative integers, arrange them such that they form the largest number.

Example 1:

Input: [10,2]

Output: “210"

Example 2:

Input: [3,30,34,5,9]

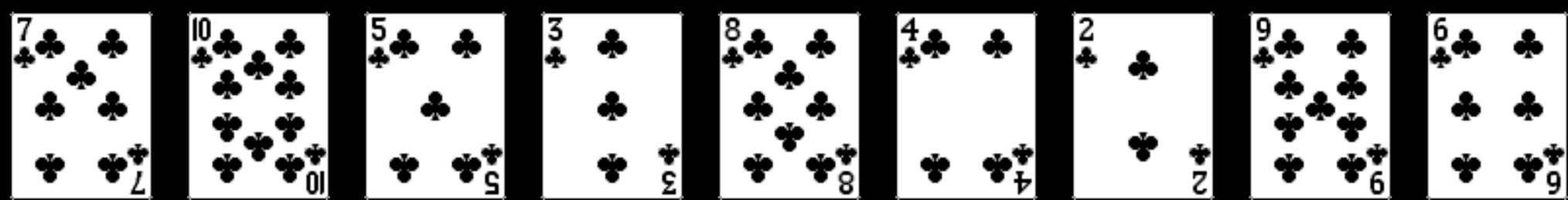
Output: "9534330"

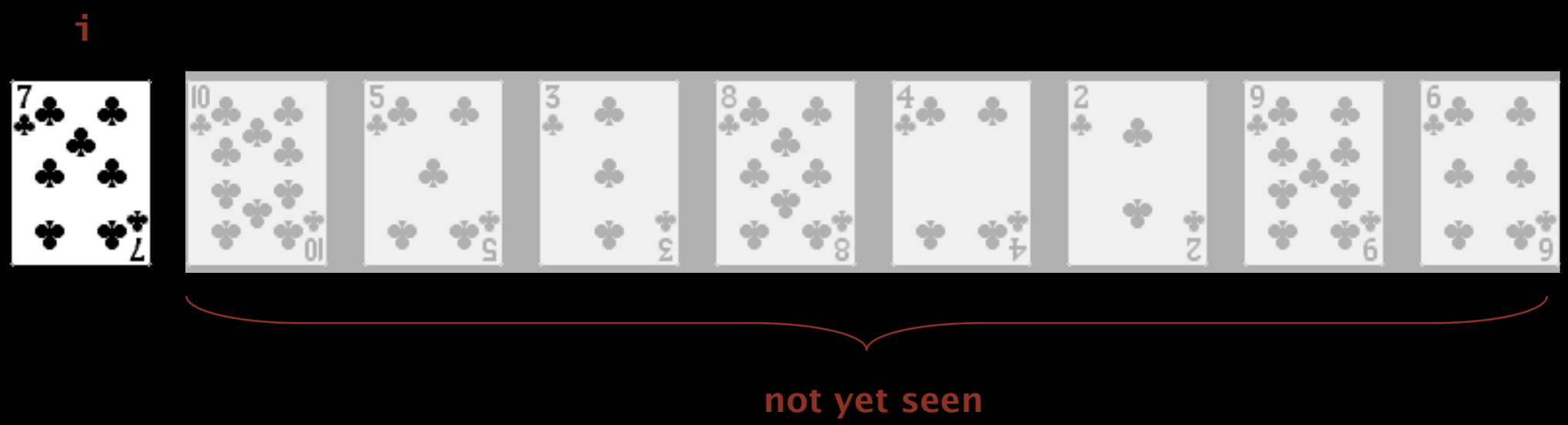
SO ARE THERE ANY SORTS THAT ARE STABLE?

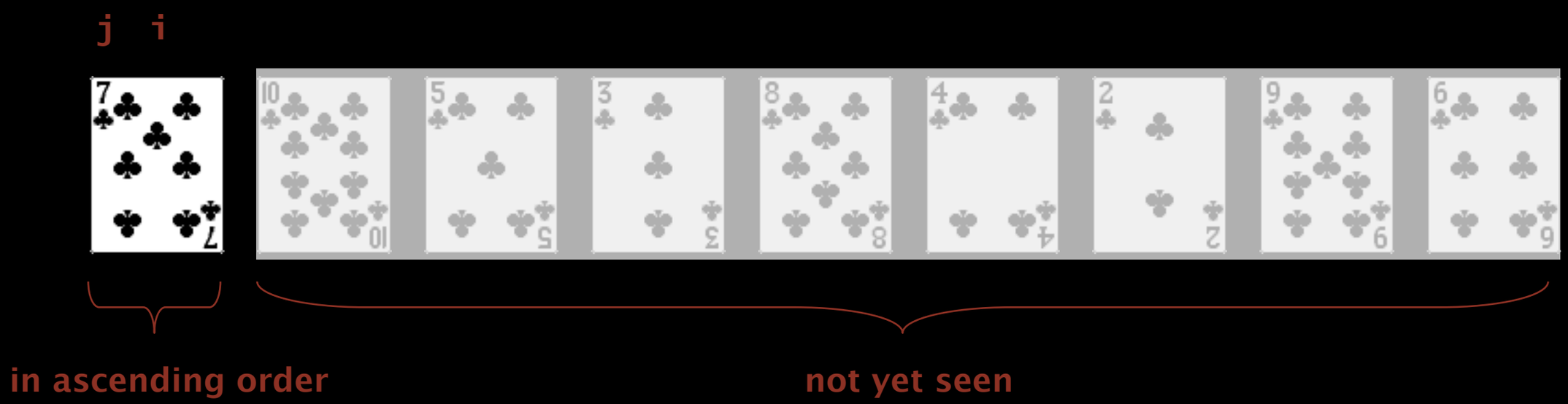
INSERTION SORT DEMO

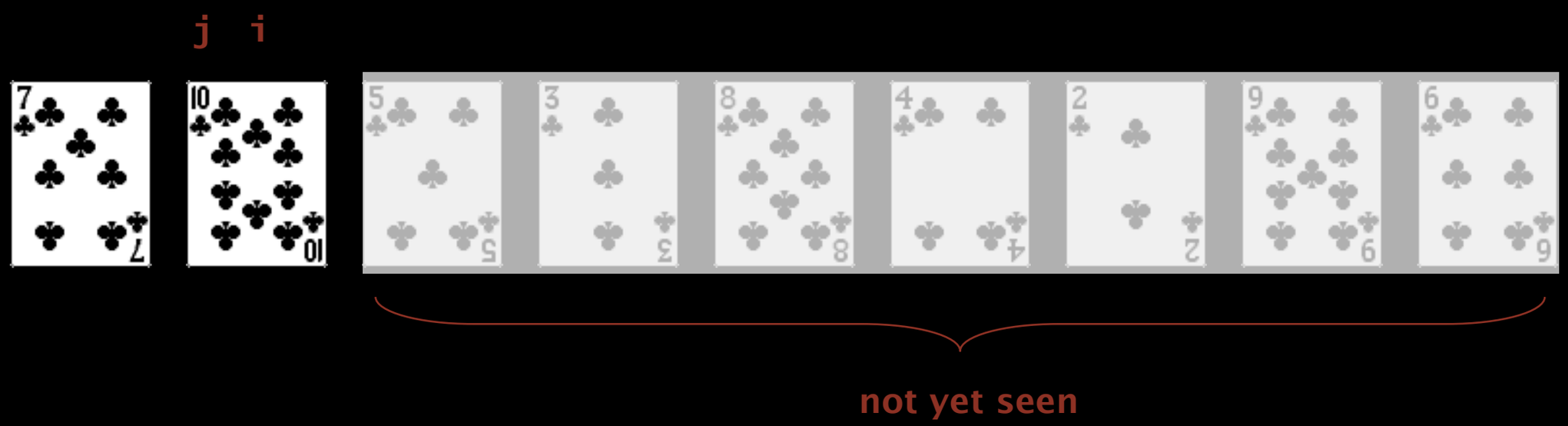
INSERTION SORT DEMO

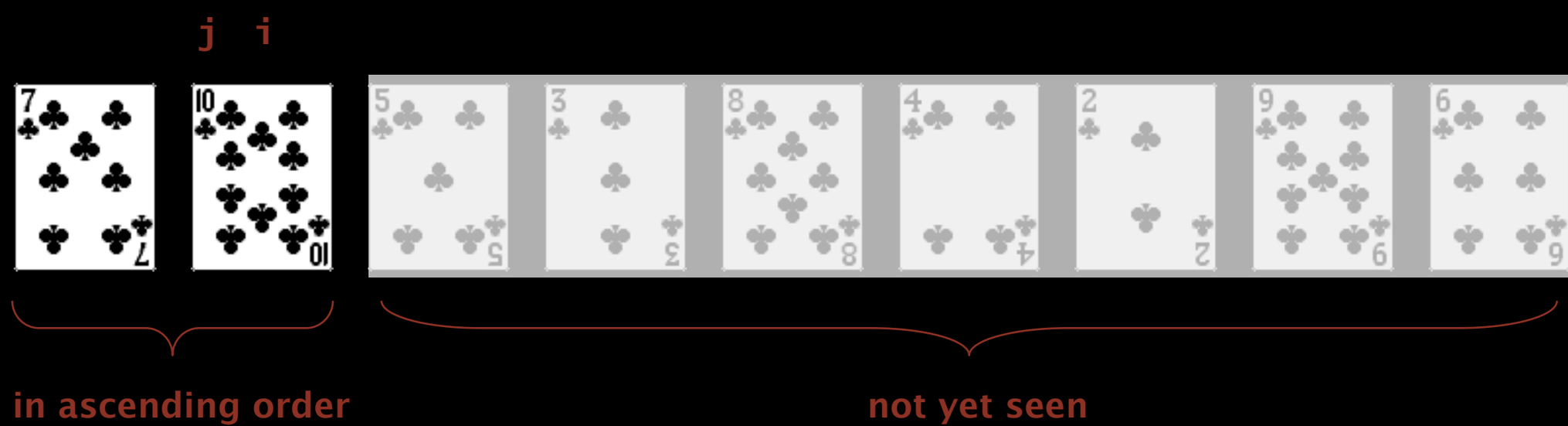
- In iteration `i`, swap `a[i]` with the value that larger than it to its left

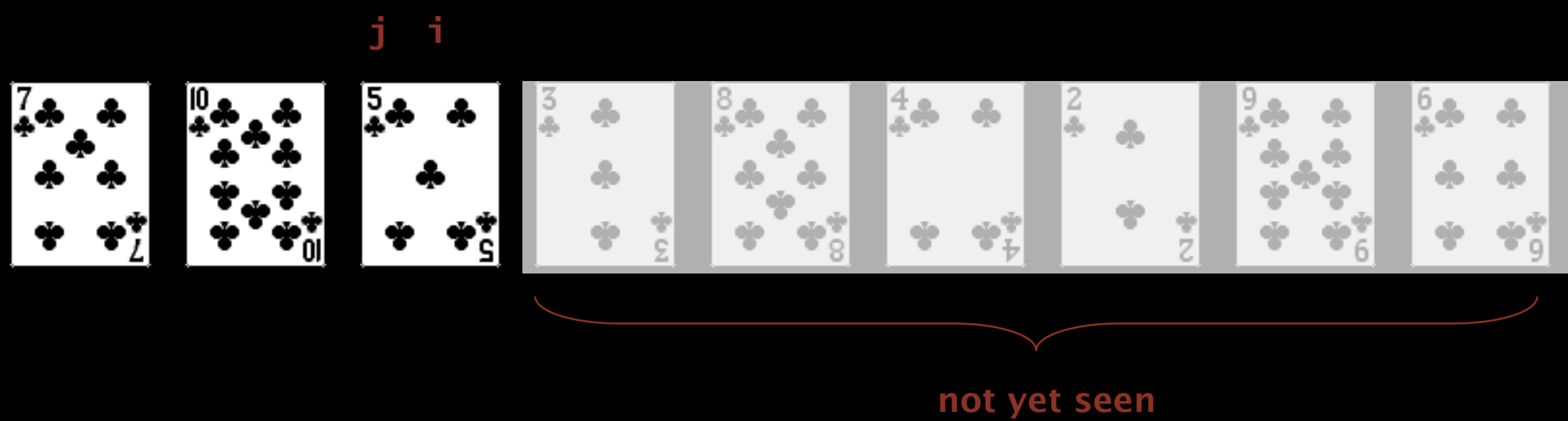


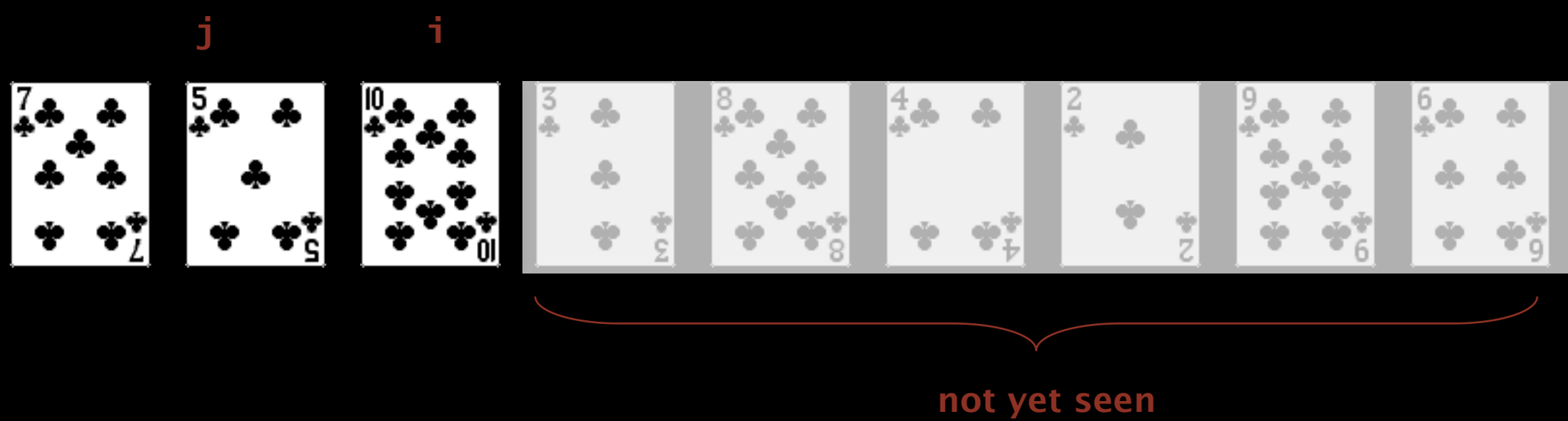

i
not yet seen

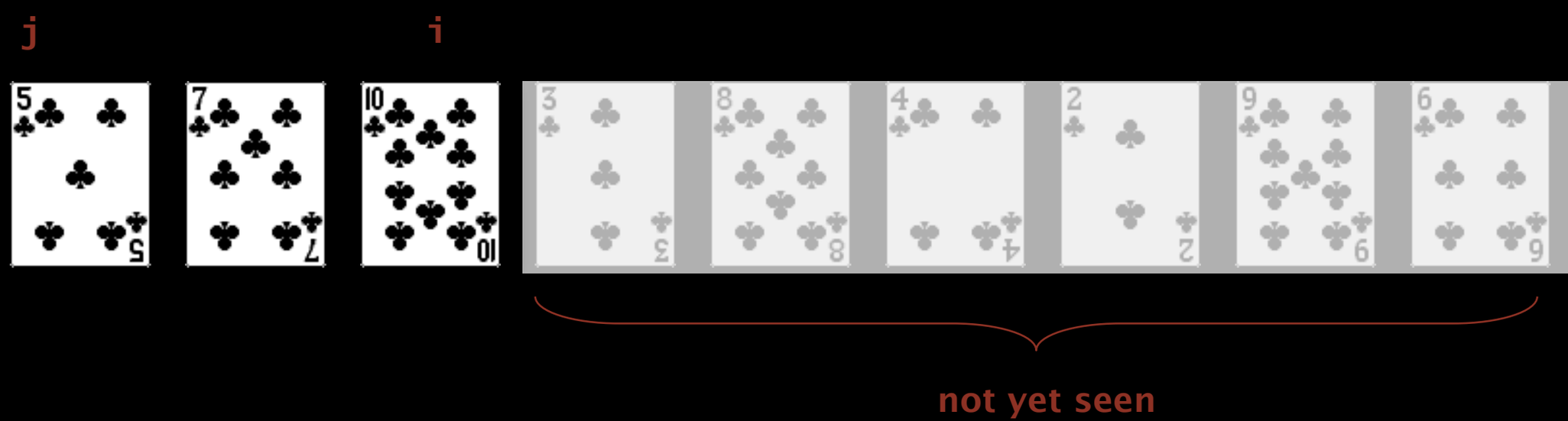

j
i
in ascending order
not yet seen

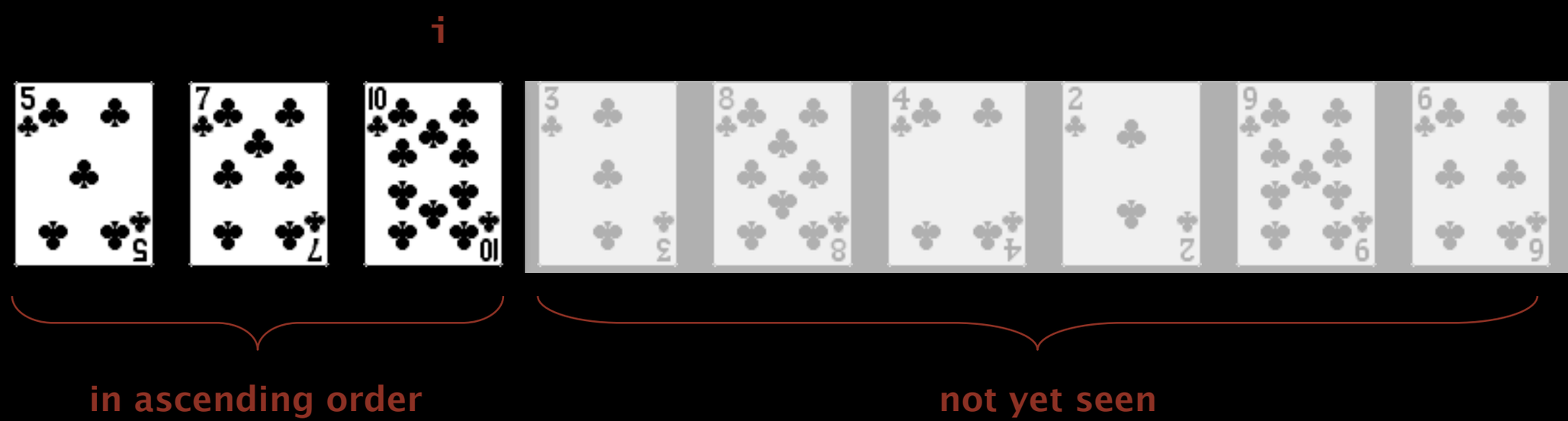

j i
not yet seen

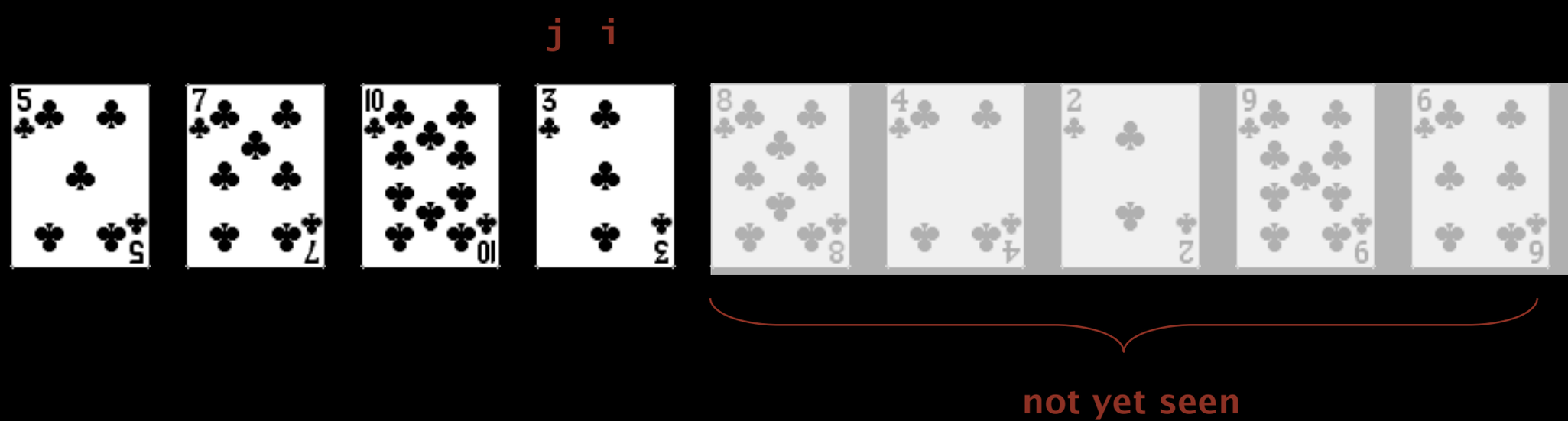

j i
in ascending order
not yet seen

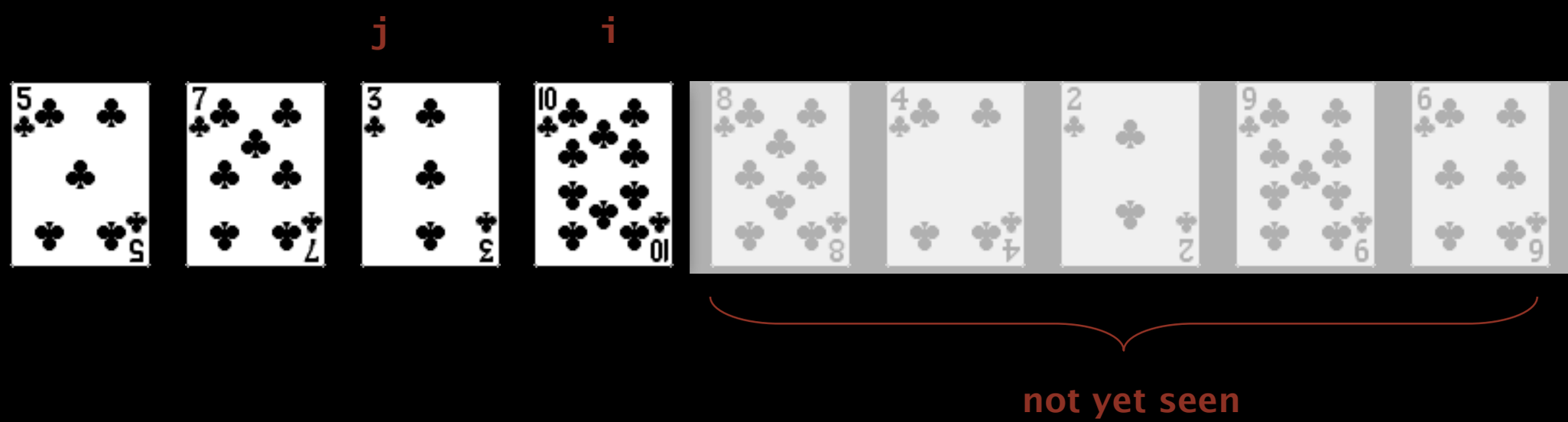

j
i
not yet seen

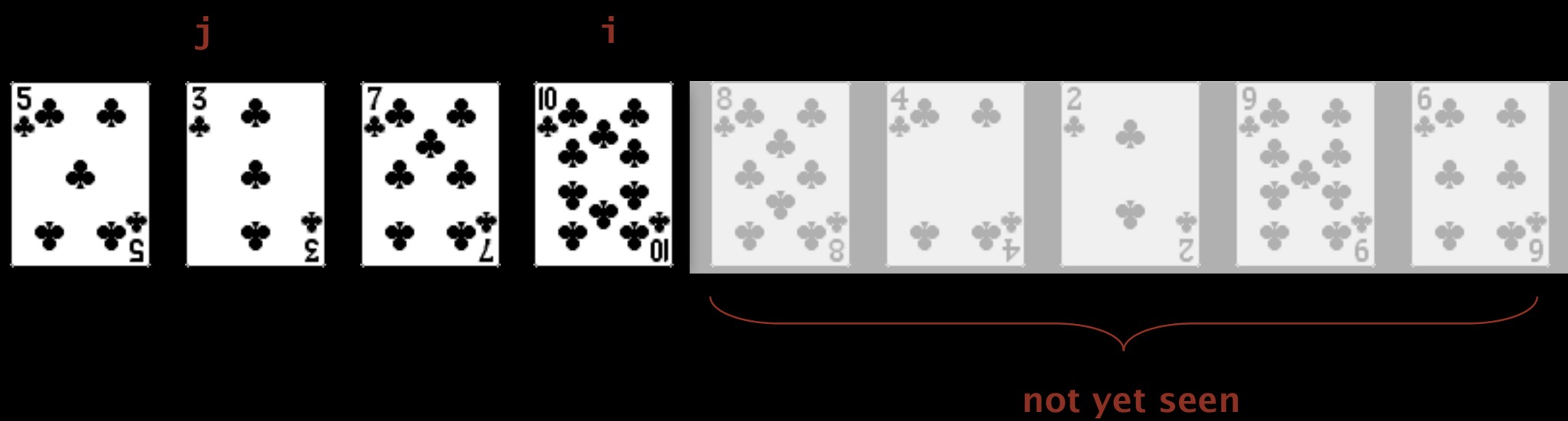

j
i
not yet seen

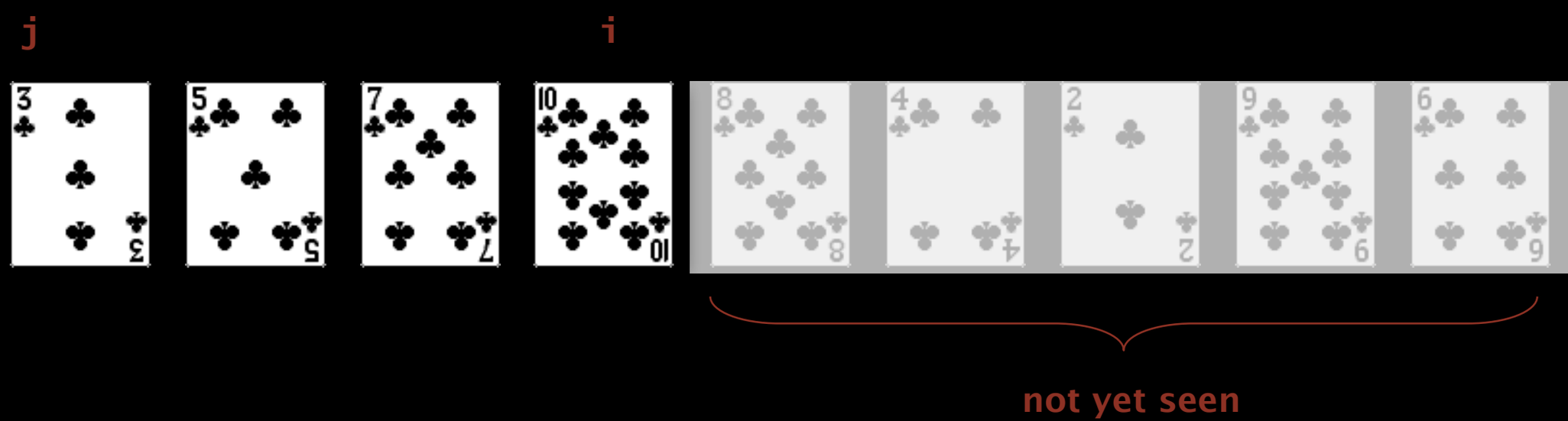

j
i
not yet seen

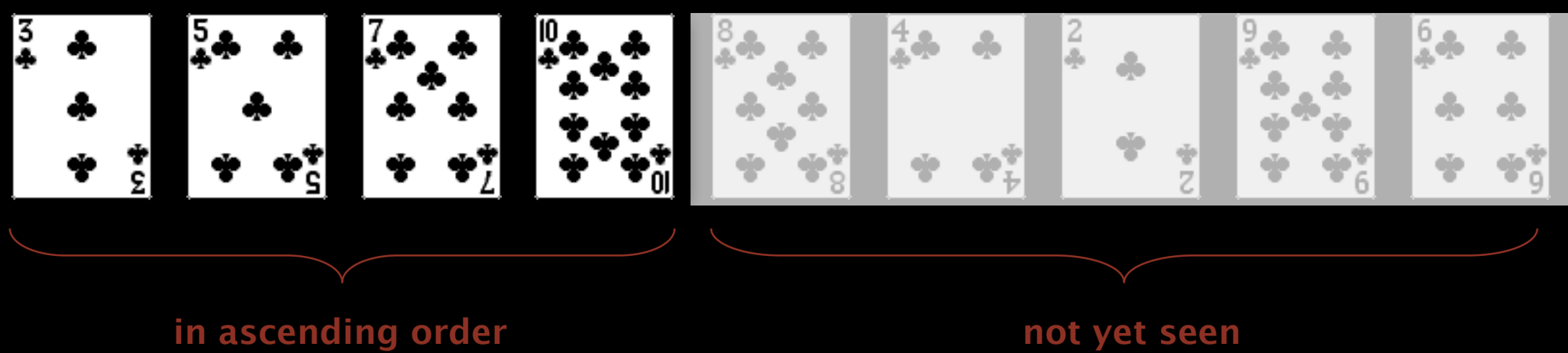

i
in ascending order
not yet seen

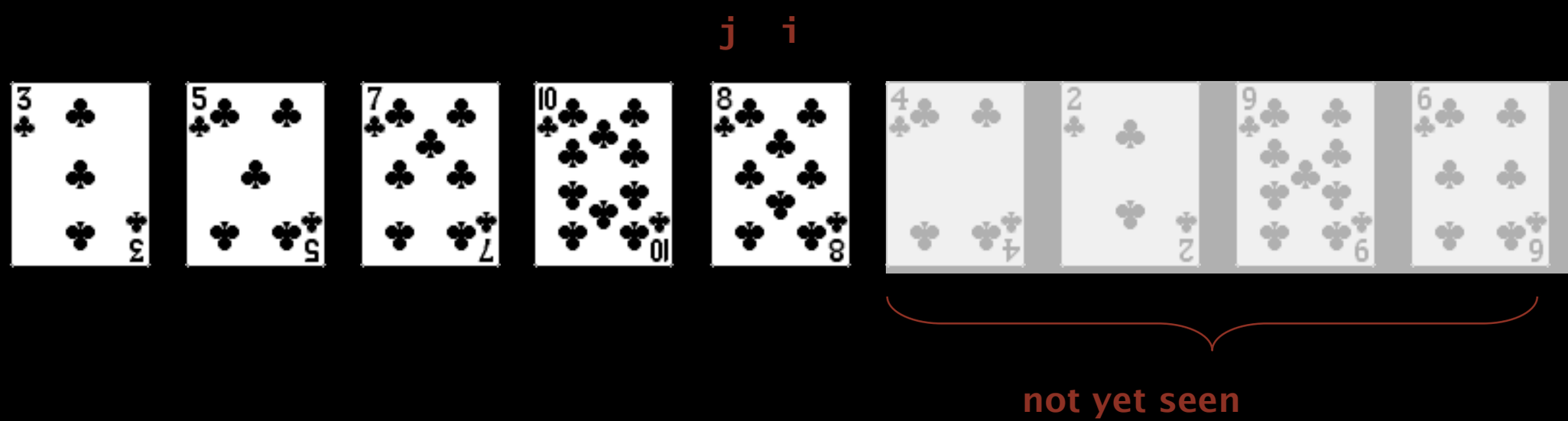

j i
not yet seen

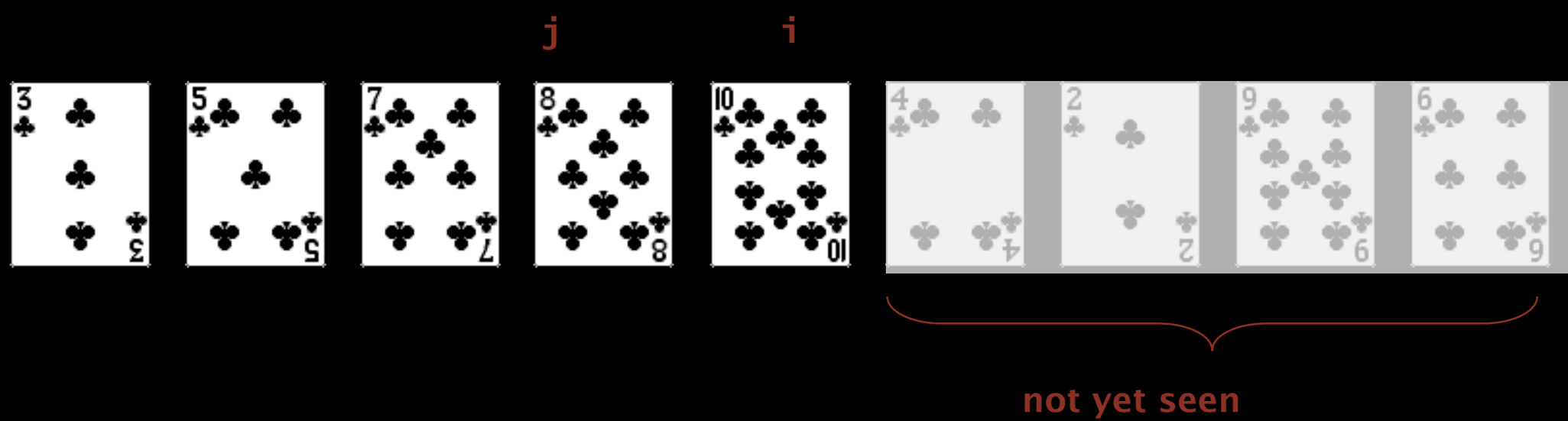

j
i
not yet seen

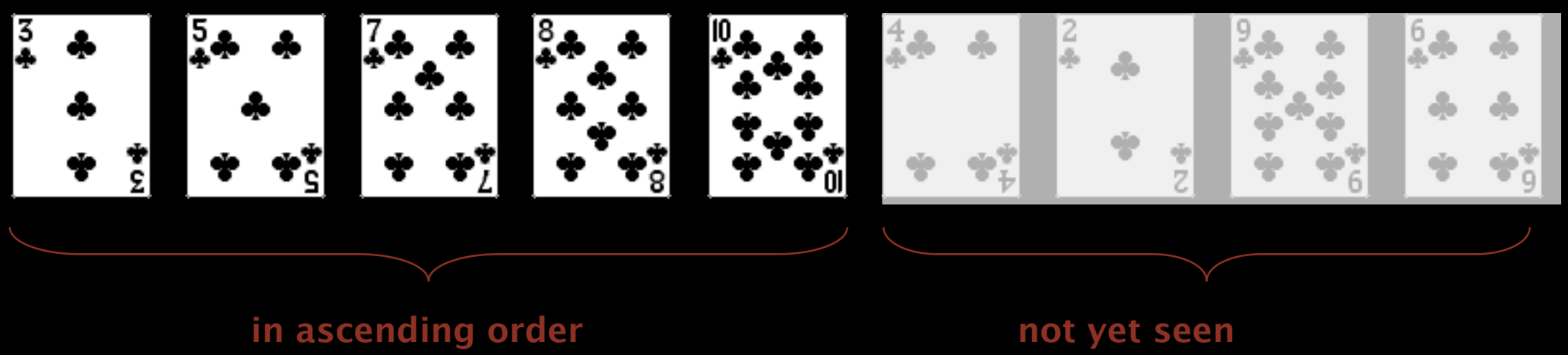

j
i
not yet seen

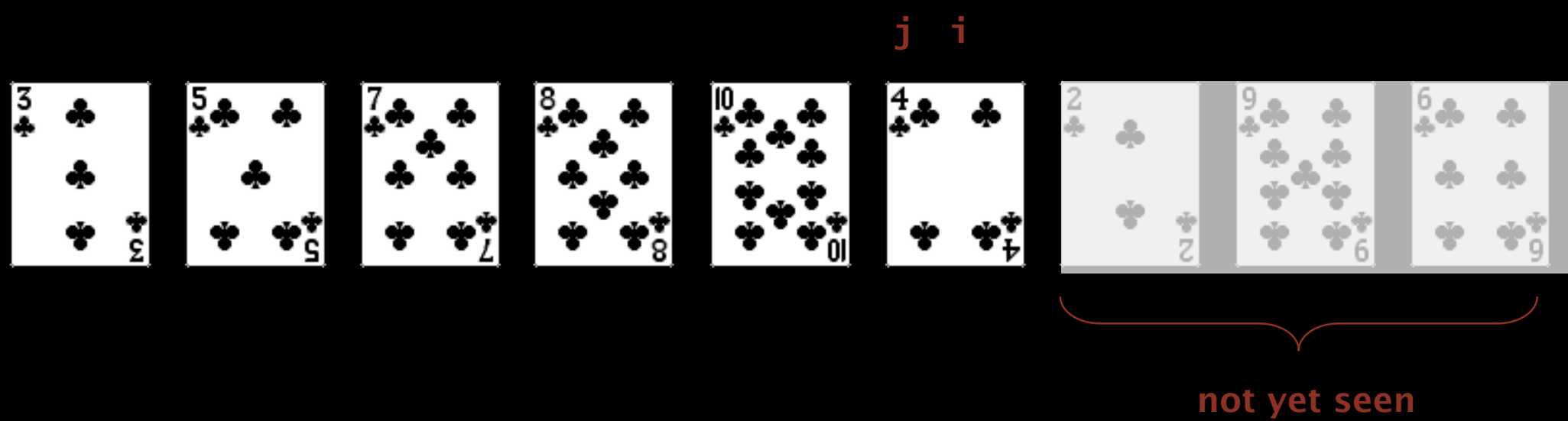

j
i
not yet seen

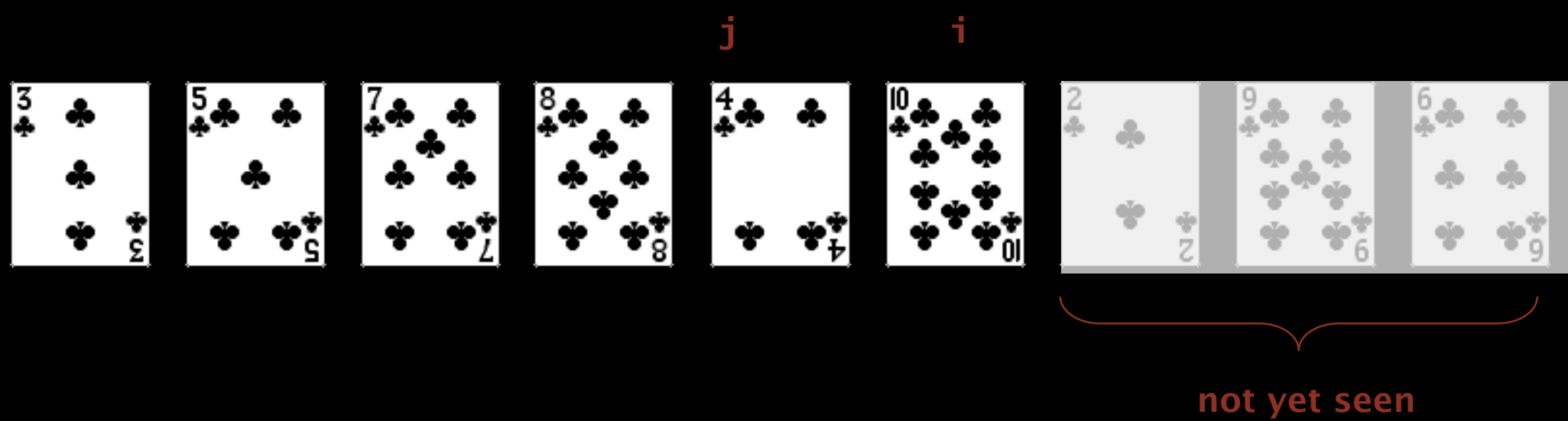

i
in ascending order
not yet seen

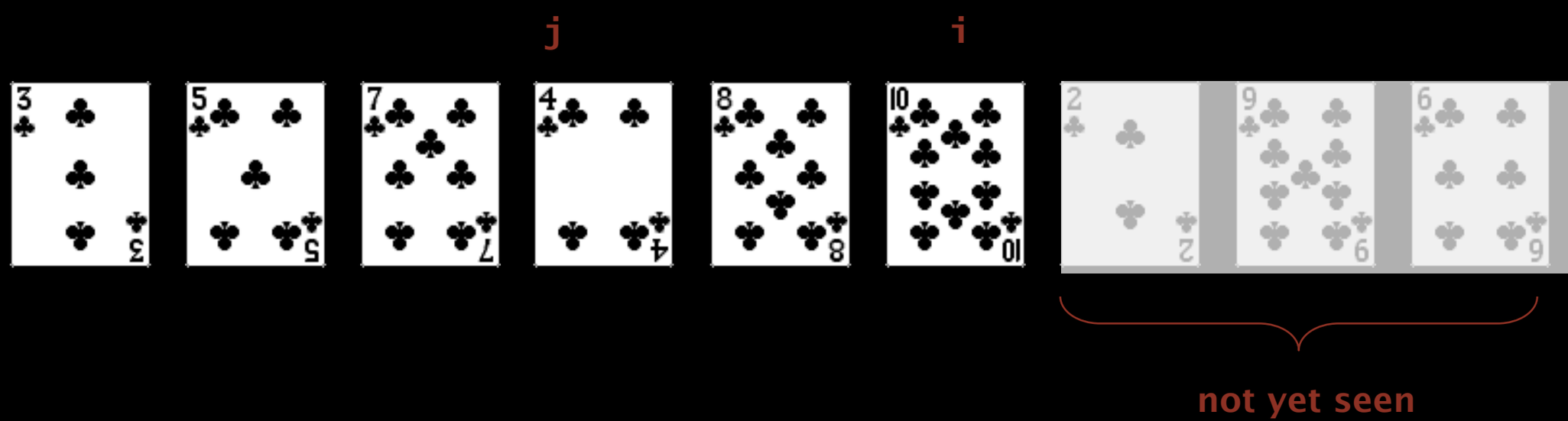

j i
not yet seen

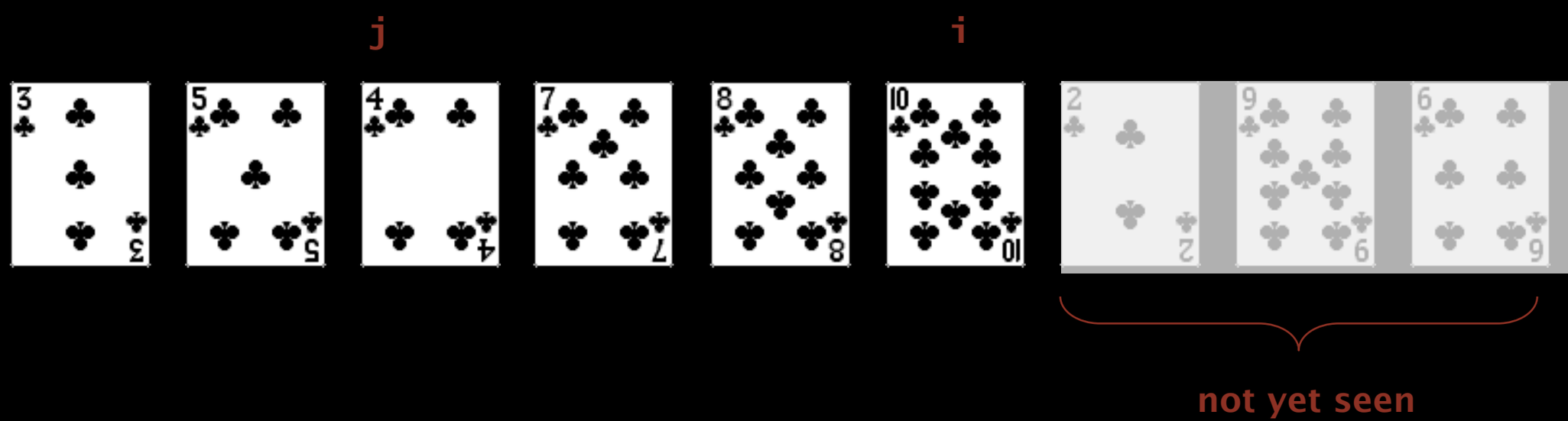

j
i
not yet seen

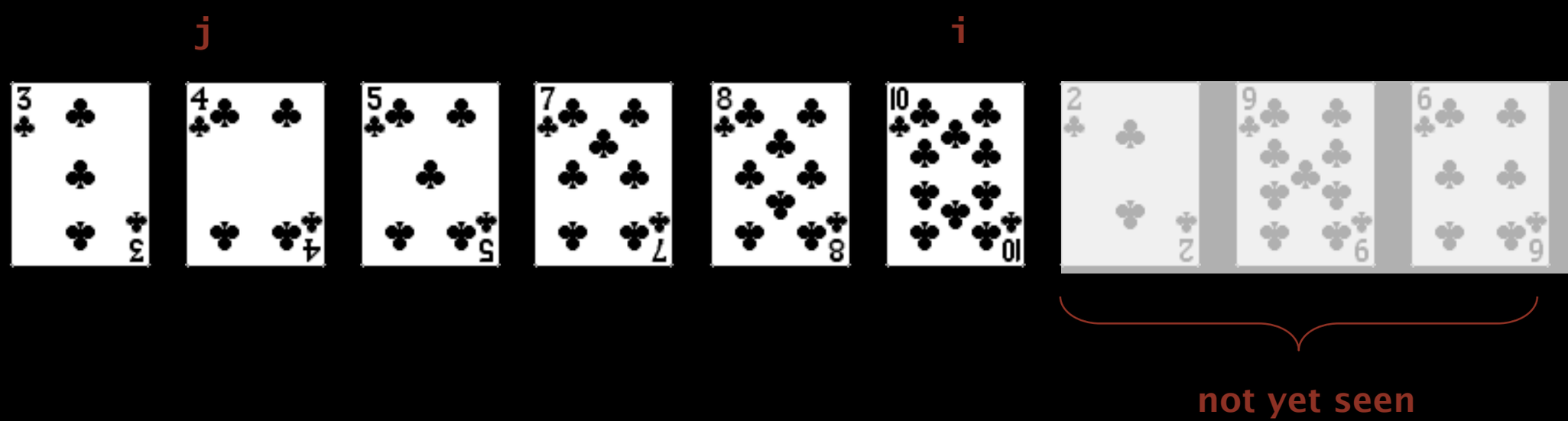

i
in ascending order
not yet seen

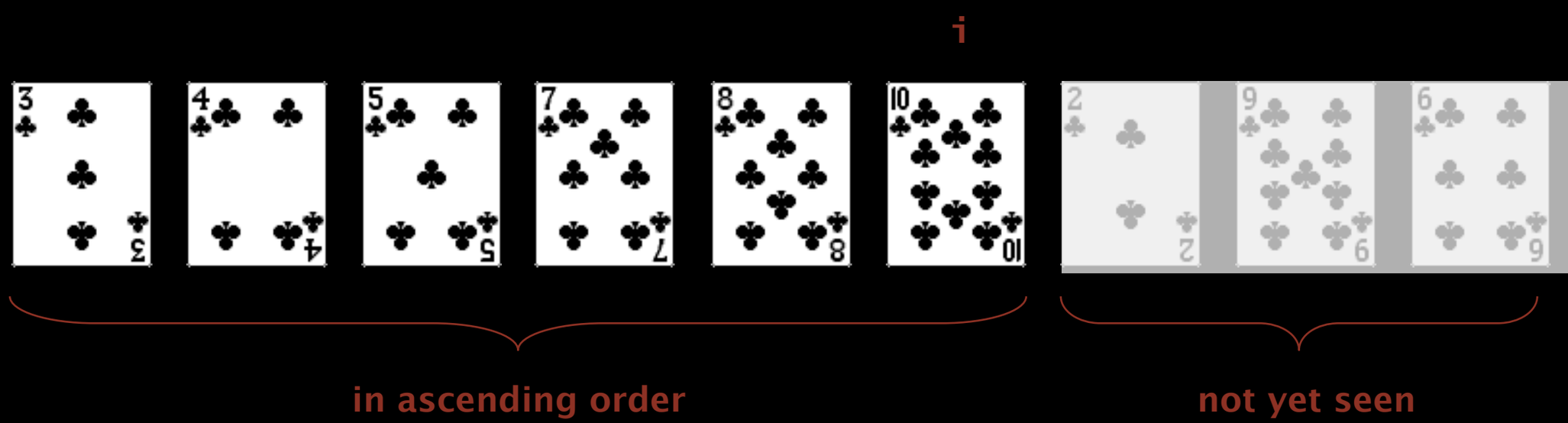

j
i
not yet seen

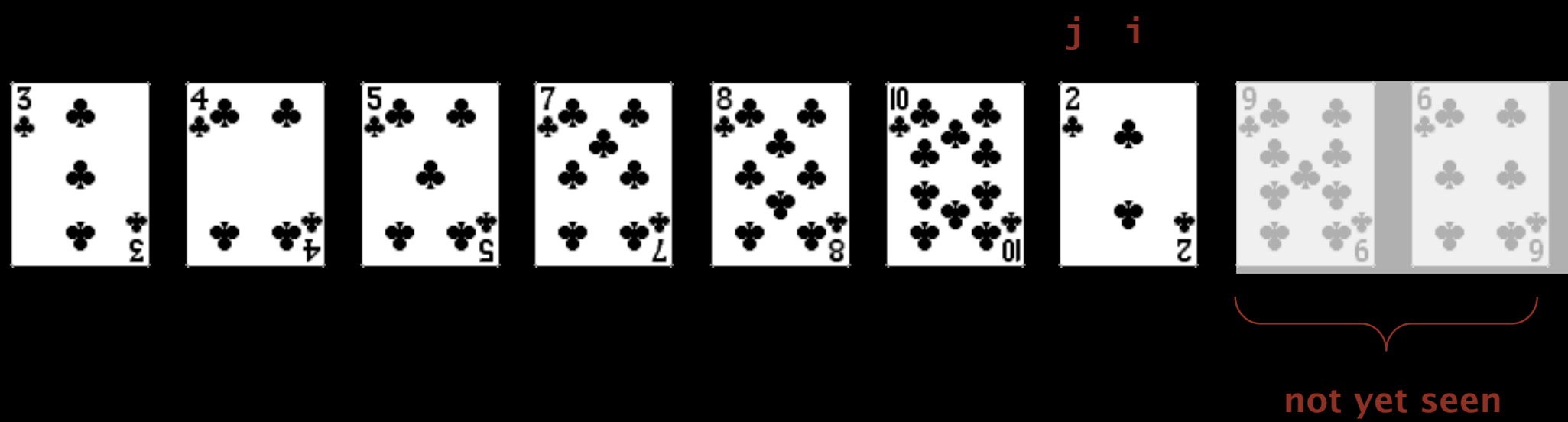

j
i
not yet seen

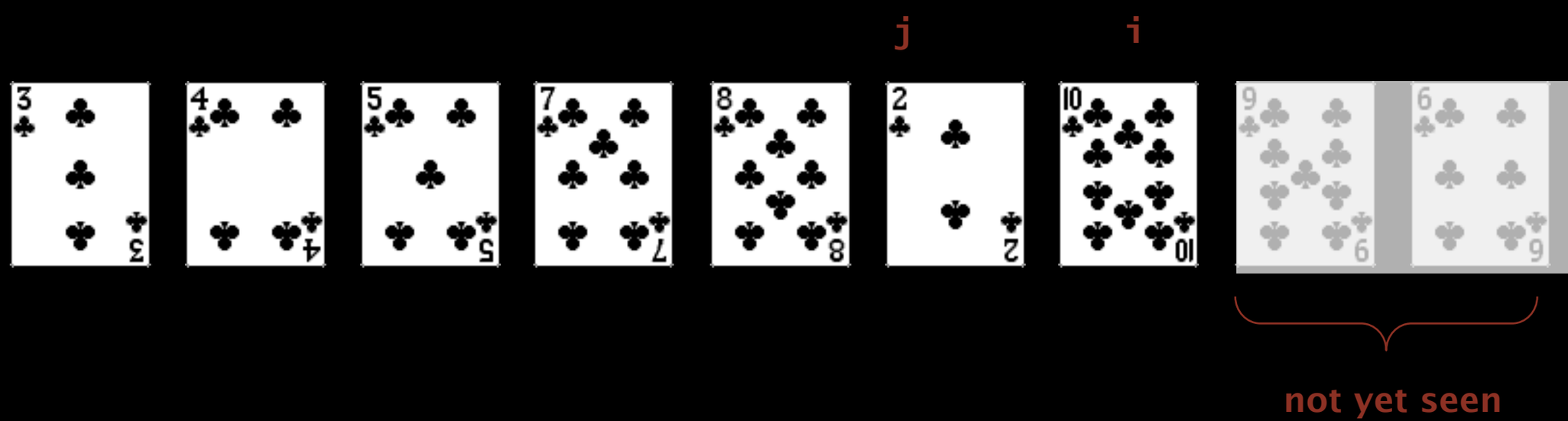

j
i
not yet seen

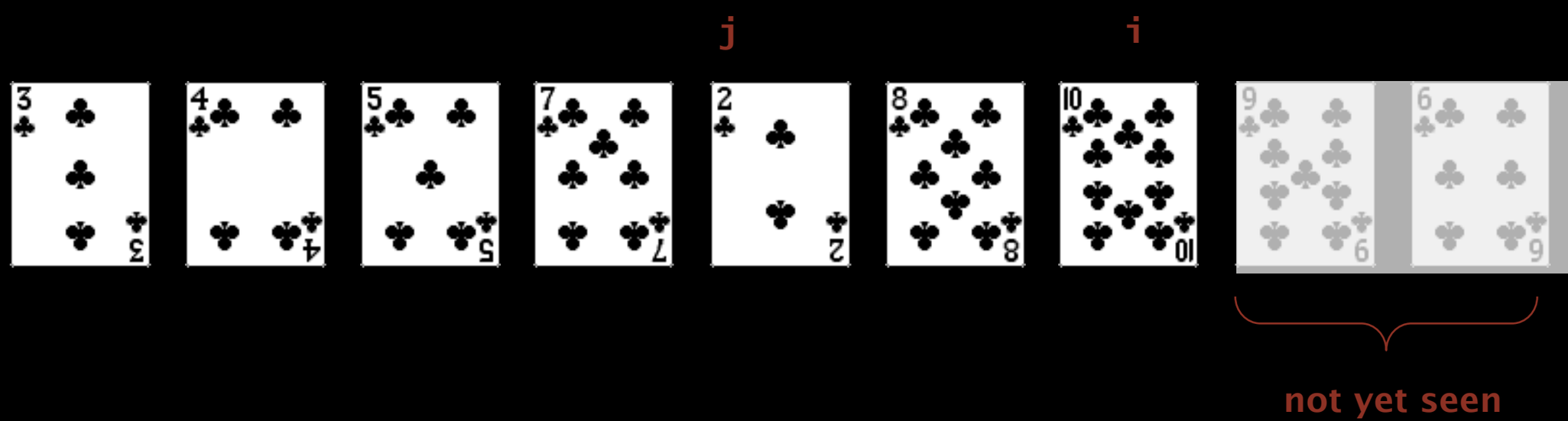

j
i
not yet seen

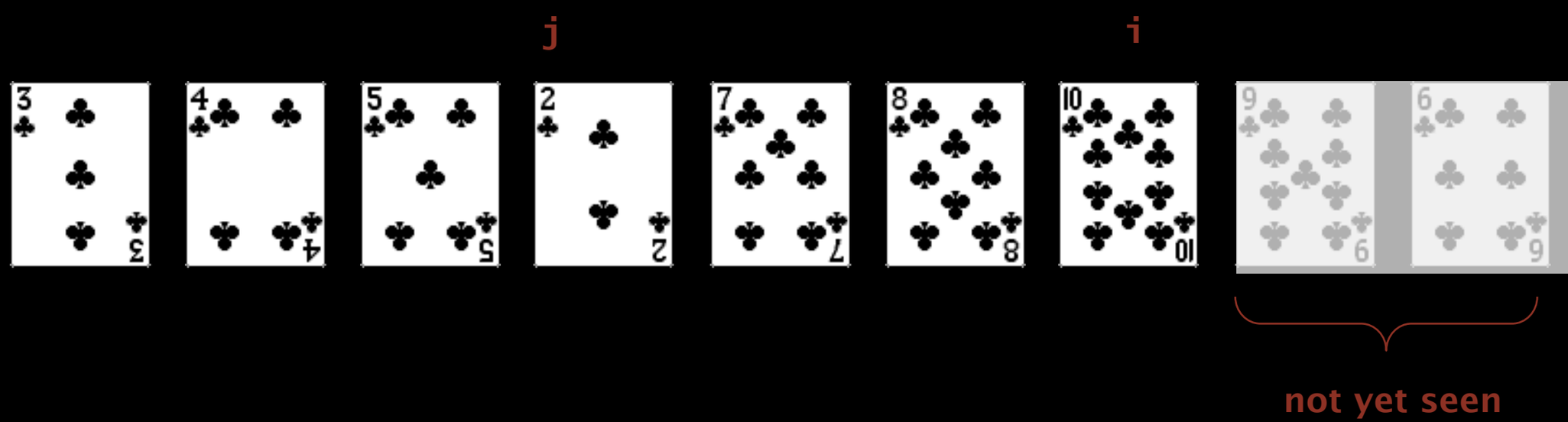

j
i
not yet seen

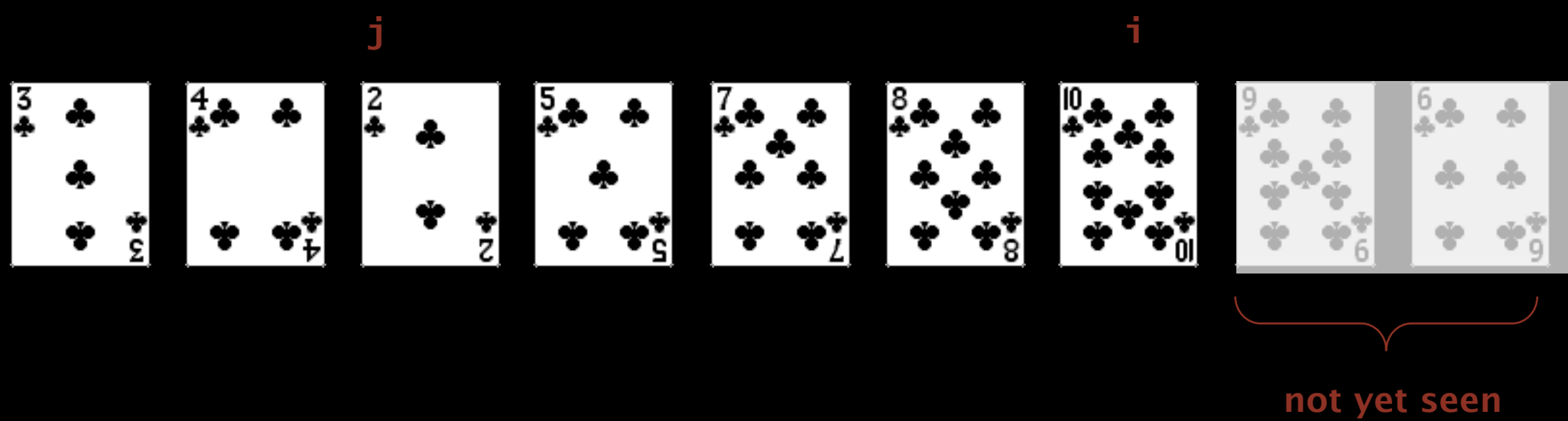

i
in ascending order
not yet seen

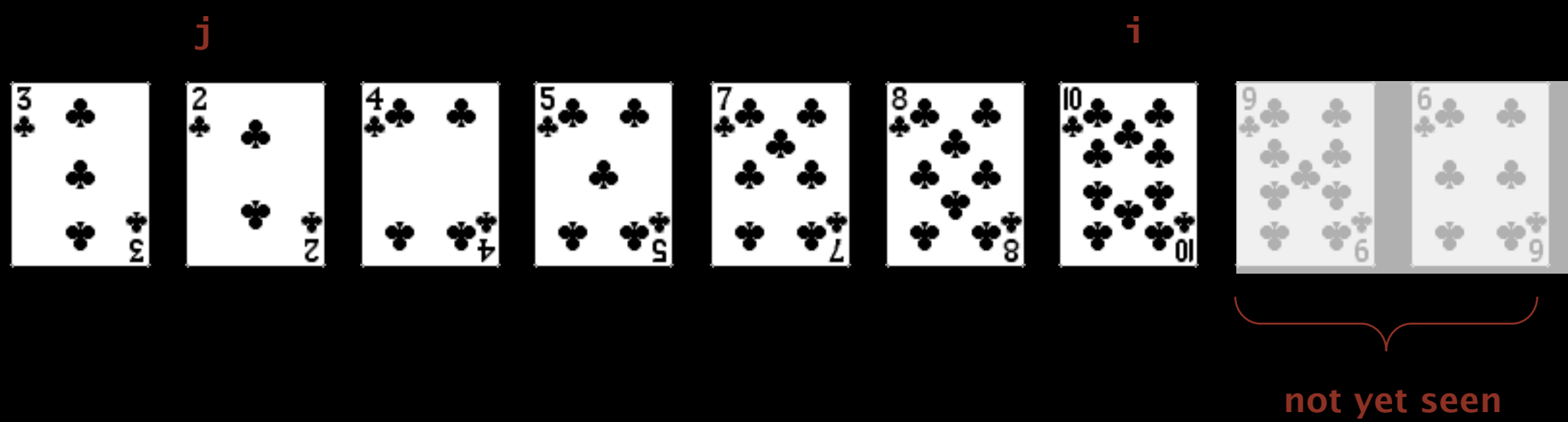

j i
not yet seen

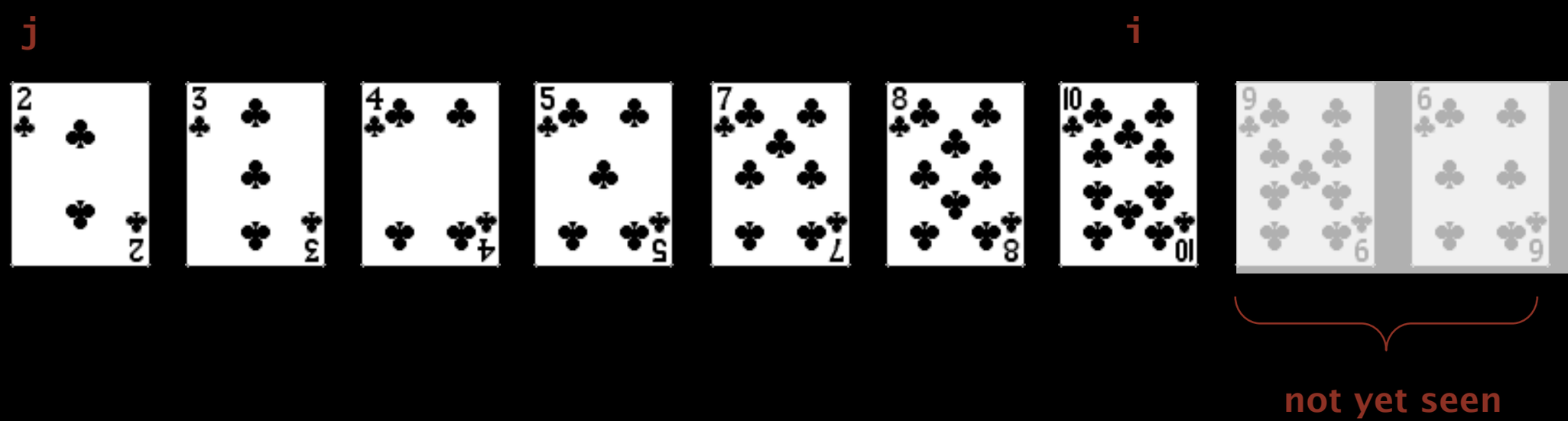

j
i
not yet seen

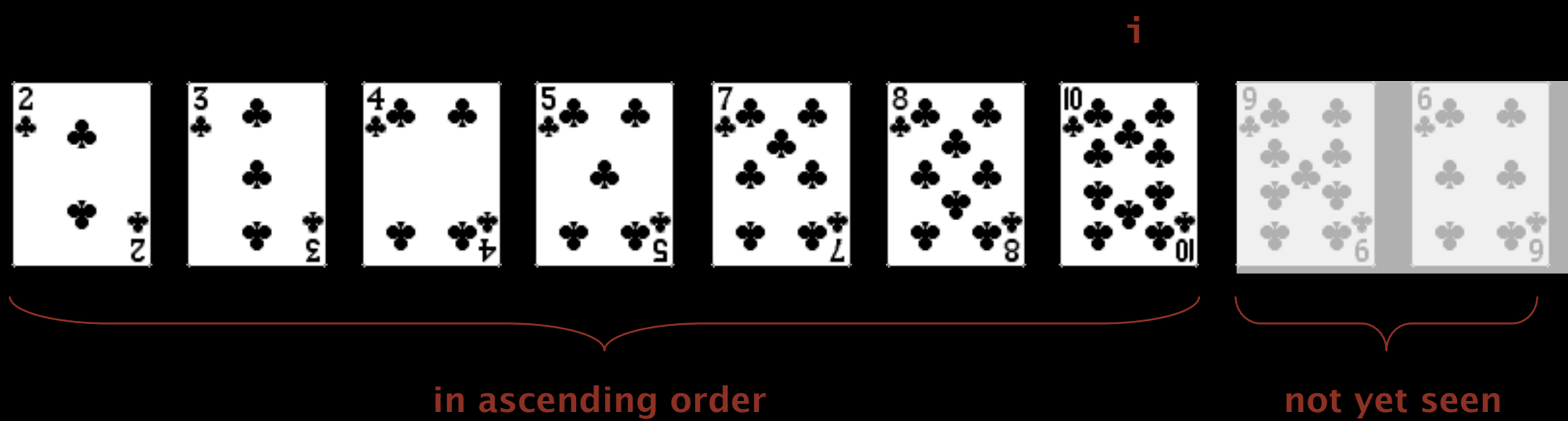

j
i
not yet seen

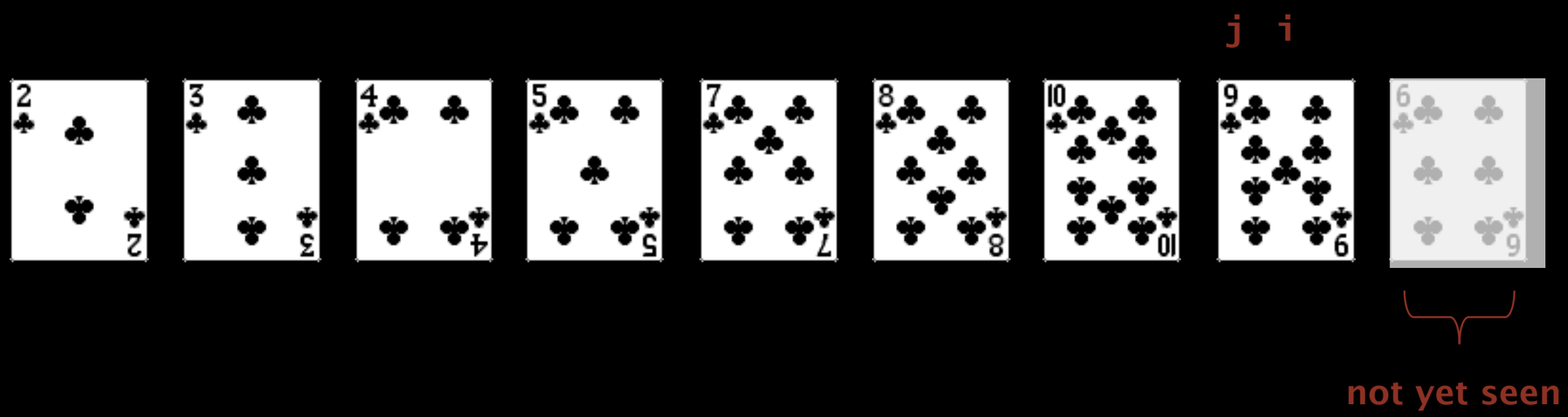

j
i
not yet seen

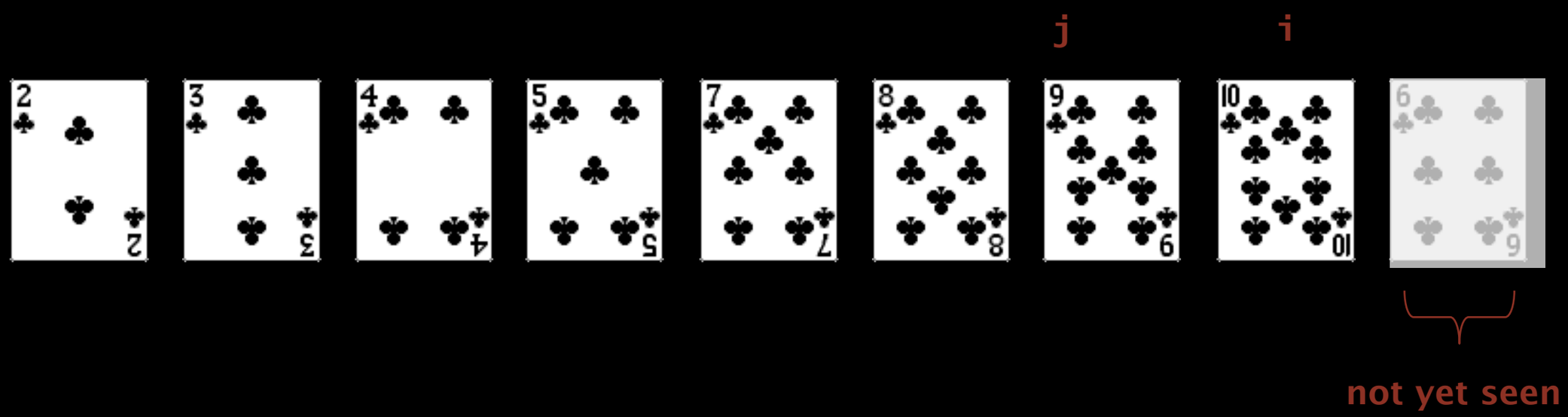

j
i
not yet seen

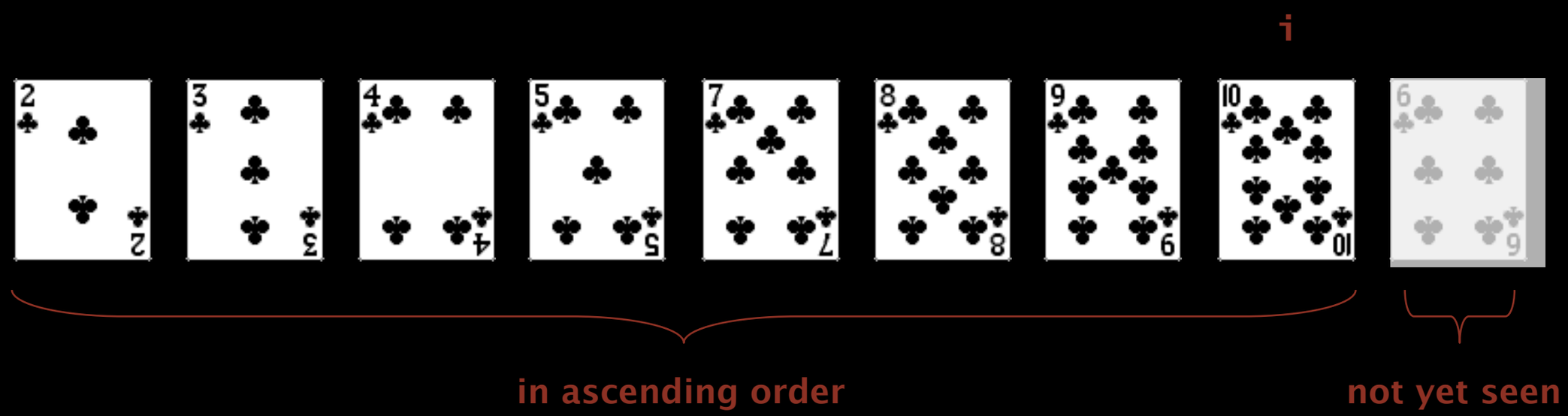

j
i
not yet seen


j
i
not yet seen

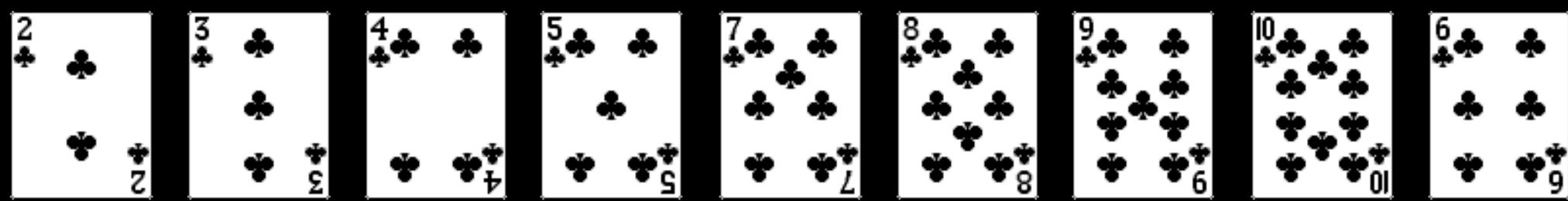

i
in ascending order
not yet seen


j i
not yet seen

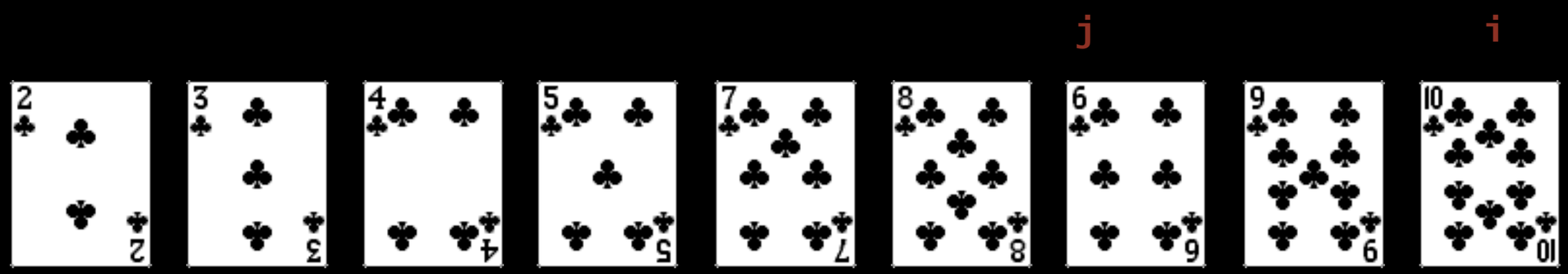

j
i
not yet seen


i
in ascending order
not yet seen

j i

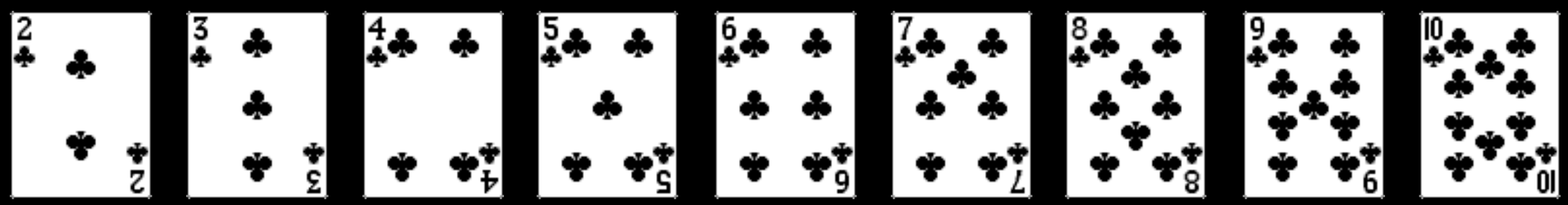


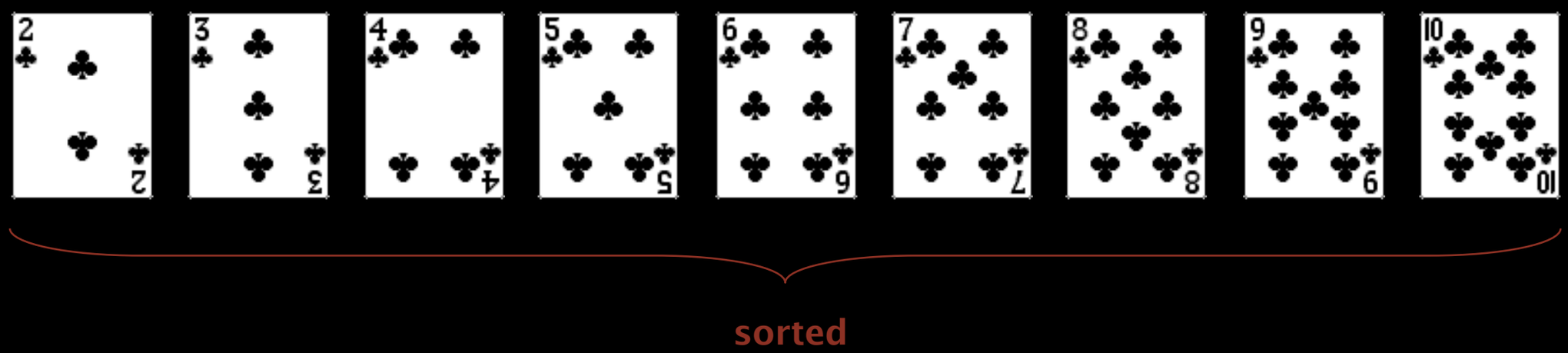
j i


j
i

j i

j i



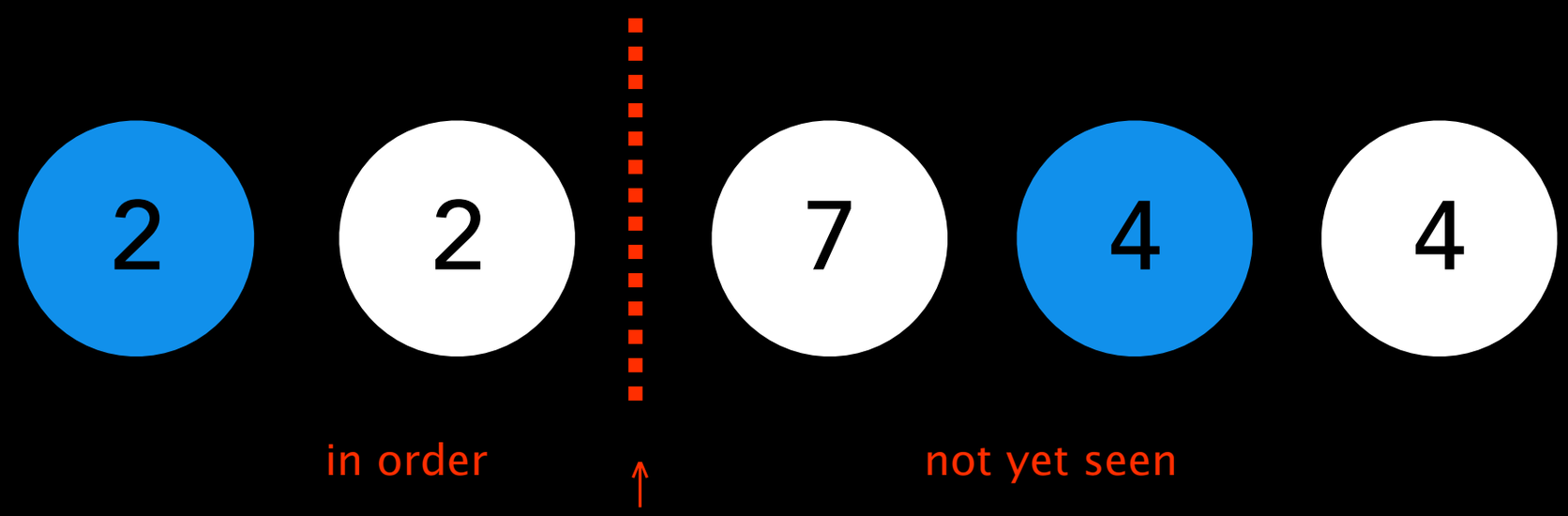

sorted

Insertion sort

Algorithm. ↑ scans from left to right.

Invariants.

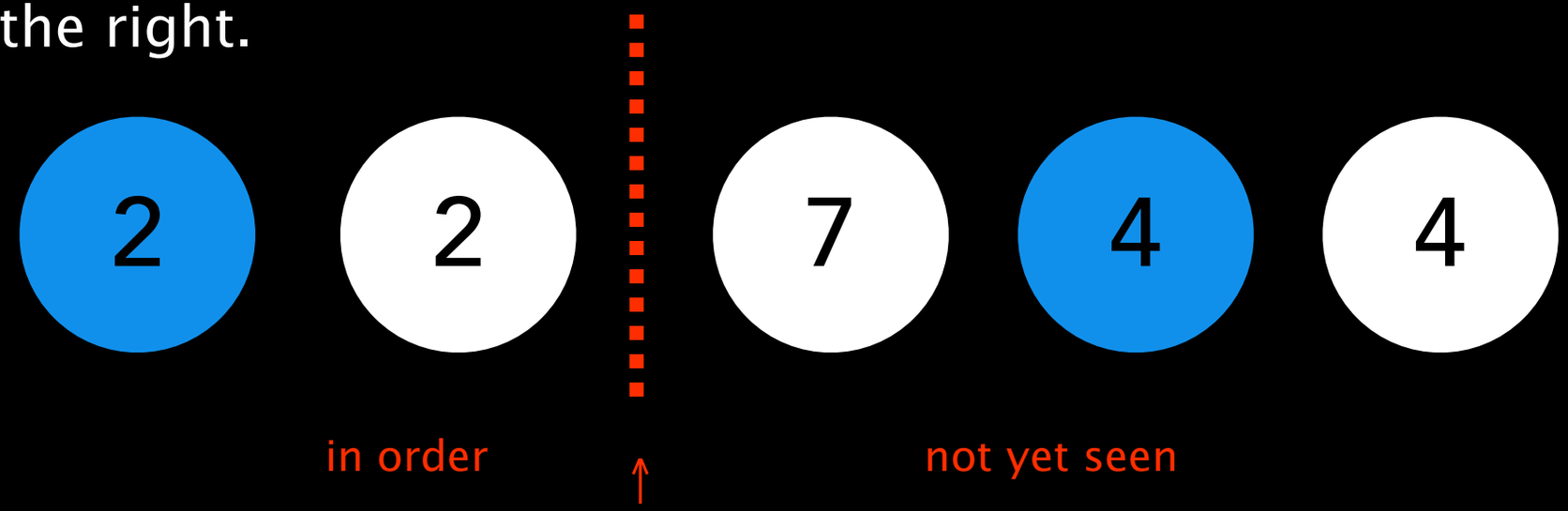
- Entries to the left of ↑ (including ↑) are in ascending order.
- Entries to the right of ↑ have not yet been seen.



Insertion sort inner loop

- Move the pointer to the right.

```
i++;
```



- Moving from right to left, exchange `a[i]` with each larger entry to its left.

```
for (int j = i; j > 0; j--)
   if (less(a[j], a[j-1]))
        exch(a, j, j-1);
   else break;
```

Insertion sort: Java implementation

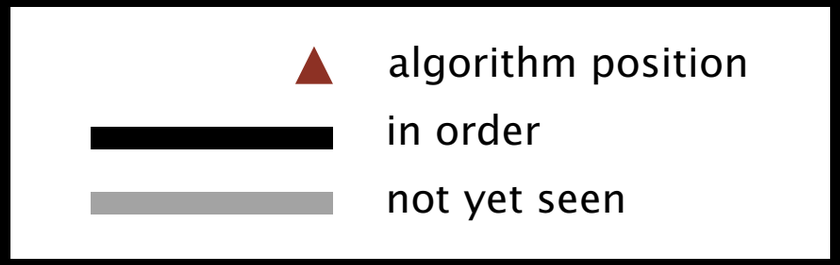
```
public class Insertion
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;
      for (int i = 0; i < N; i++)
         for (int j = i; j > 0; j--)
            if (less(a[j], a[j-1]))
               exch(a, j, j-1);
            else break;
   }

   private static boolean less(Comparable v, Comparable w)
   {  /* as before */  }

   private static void exch(Comparable[] a, int i, int j)
   {  /* as before */  }
}
```

Insertion sort: animation

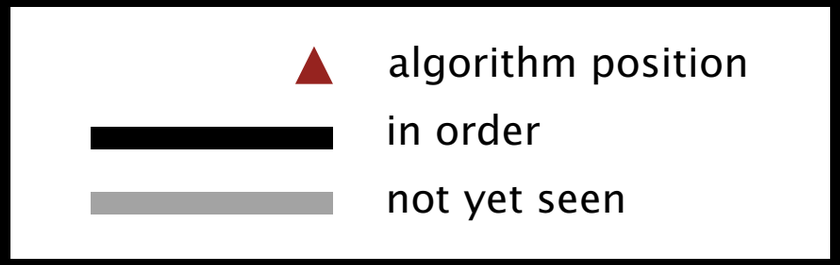
40 random items



http://www.sorting-algorithms.com/insertion-sort

Insertion sort: animation

40 reverse-sorted items



http://www.sorting-algorithms.com/insertion-sort

Insertion sort: analysis

Best case. If the array is in ascending order, insertion sort makes $N-1$ compares and 0 exchanges.

A E E L M O P R S T X

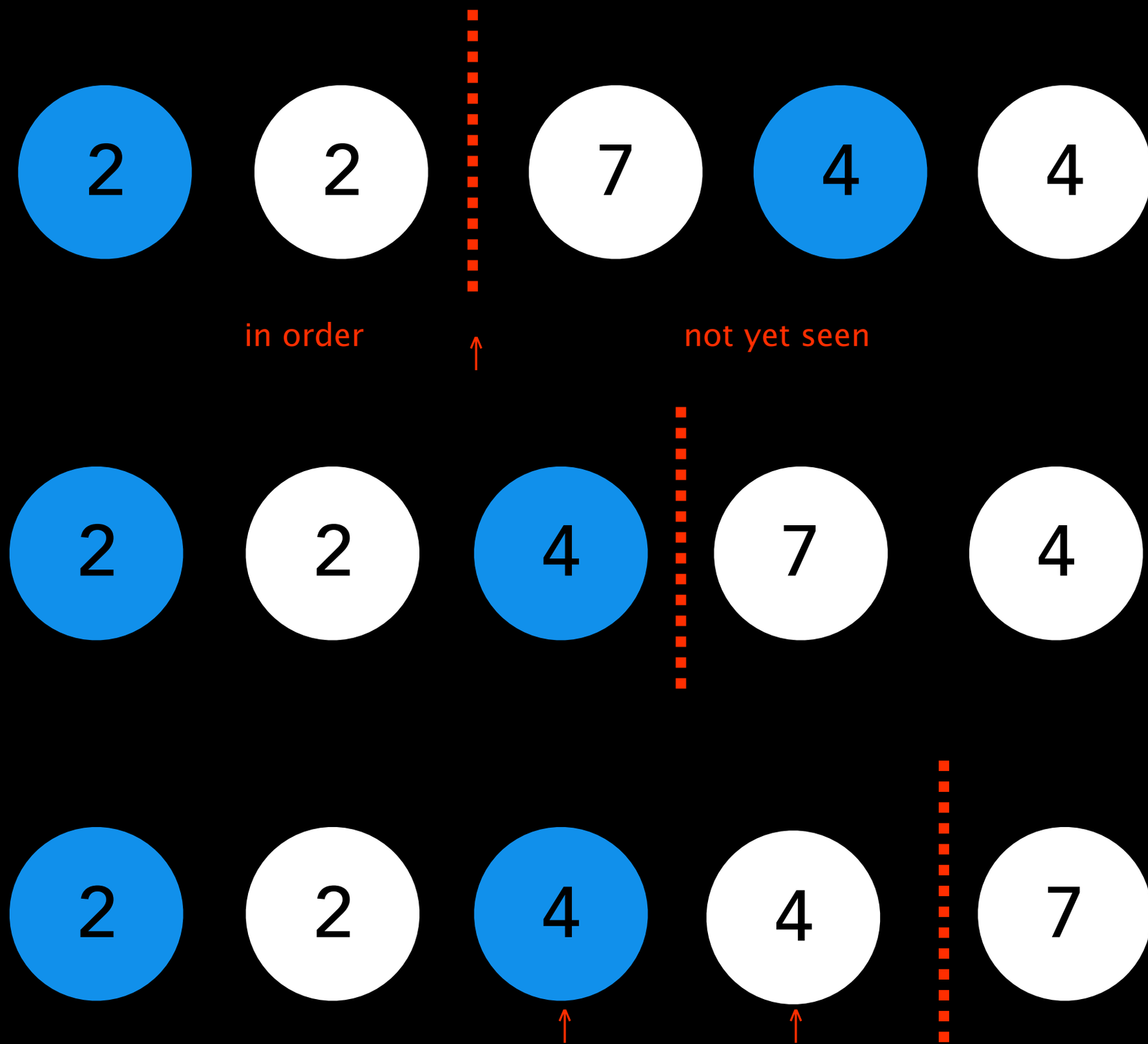
Worst case. If the array is in descending order (and no duplicates), insertion sort makes $\sim \frac{1}{2} N^2$ compares and $\sim \frac{1}{2} N^2$ exchanges.

X T S R P O M L F E A

IS INSERTION SORT STABLE?

YES IT IS

ORDER IS PRESERVED



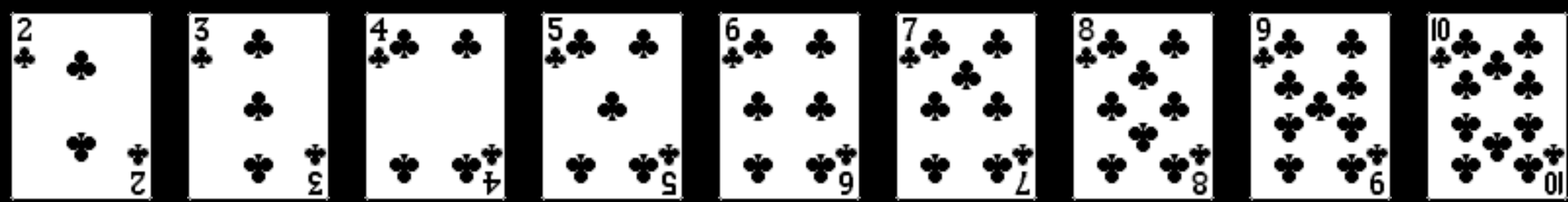
THE WILL NOT GET EXCHANGE

LET'S LOOK AT SOME APPLICATIONS OF SORTING

How to shuffle an array

Goal. Rearrange array so that result is a uniformly random permutation.

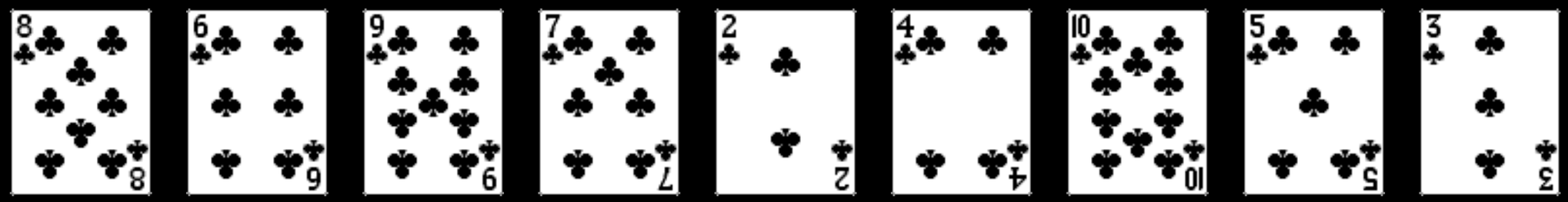
all permutations
equally likely



How to shuffle an array

Goal. Rearrange array so that result is a uniformly random permutation.

all permutations
equally likely

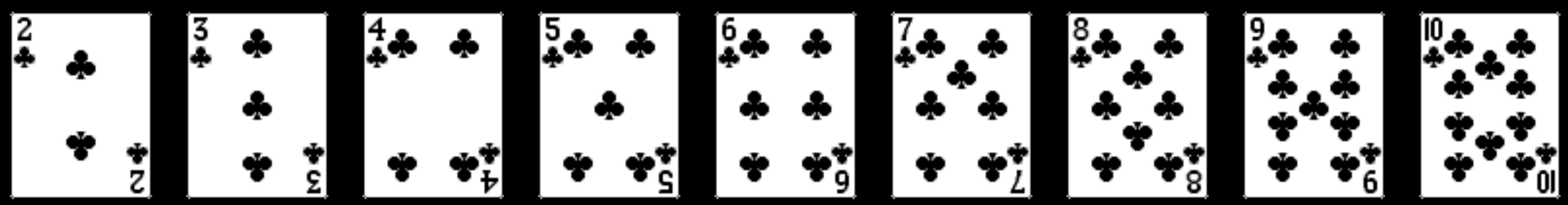


ANY IDEAS ON HOW WE COULD USE SORTING TO SOLVE THIS PROBLEM?

Shuffle sort

- Generate a random real number for each array entry.
- Sort the array.

useful for shuffling columns in a spreadsheet

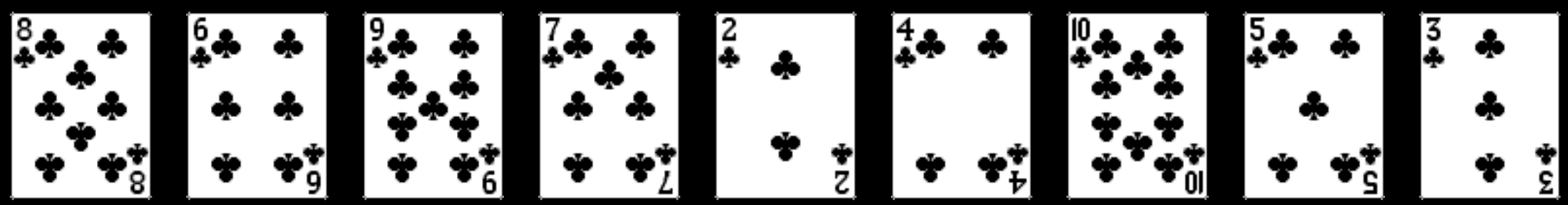


0.8003 0.9706 0.9157 0.9649 0.1576 0.4854 0.1419 0.4218 0.9572

Shuffle sort

- Generate a random real number for each array entry.
- Sort the array.

useful for shuffling columns in a spreadsheet

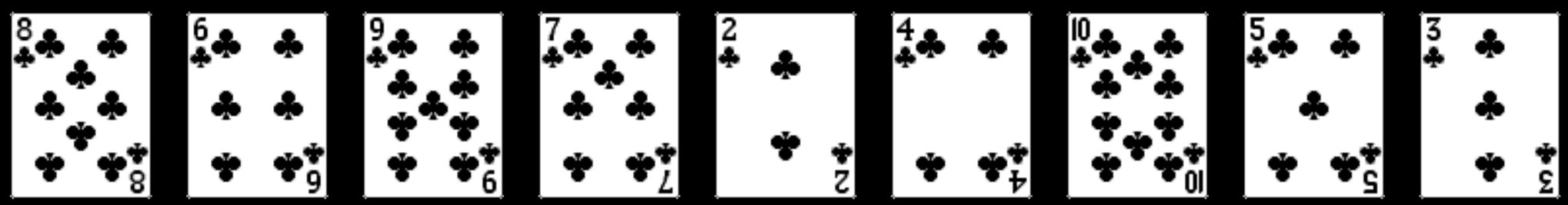


0.1419	0.1576	0.4218	0.4854	0.8003	0.9157	0.9572	0.9649	0.9706

Shuffle sort

- Generate a random real number for each array entry.
- Sort the array.

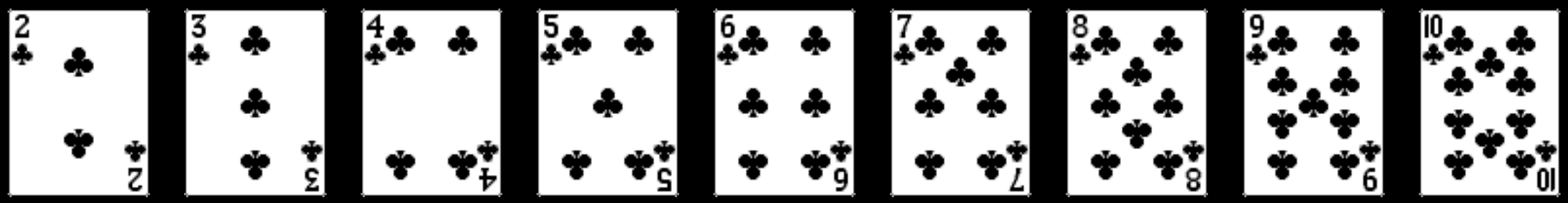
useful for shuffling columns in a spreadsheet

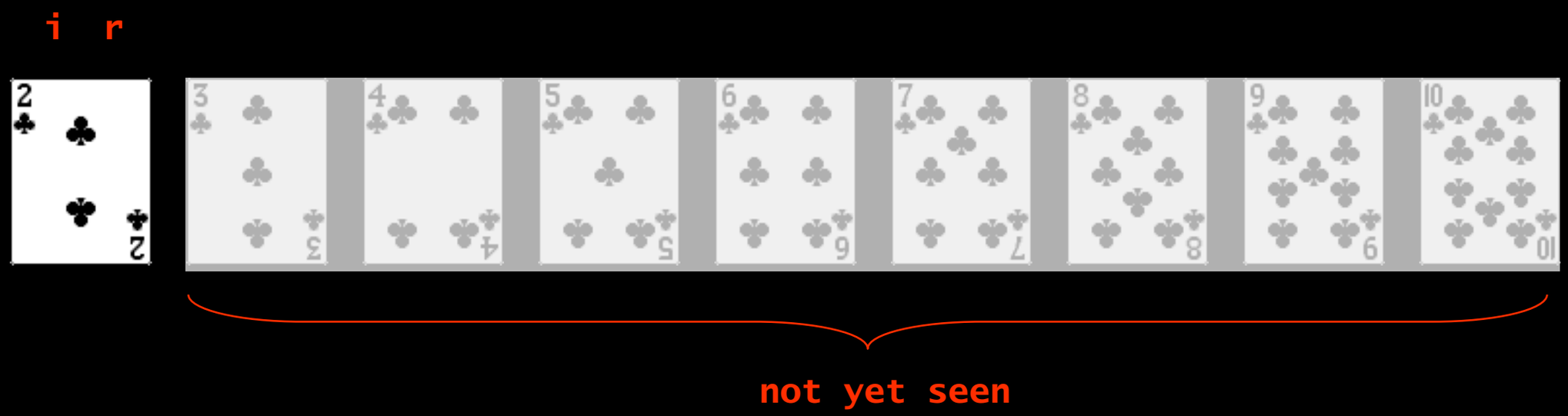


0.1419	0.1576	0.4218	0.4854	0.8003	0.9157	0.9572	0.9649	0.9706

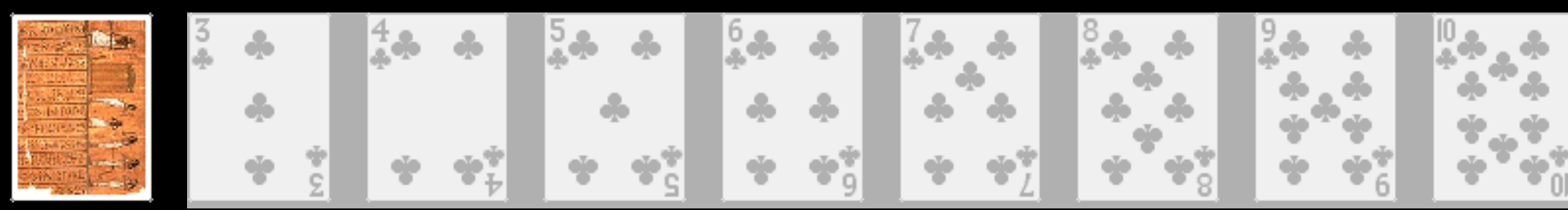
Proposition. Shuffle sort produces a uniformly random permutation.

CAN WE THINK ABOUT HOW TO IMPLEMENT SHUFFLE IN LINEAR TIME

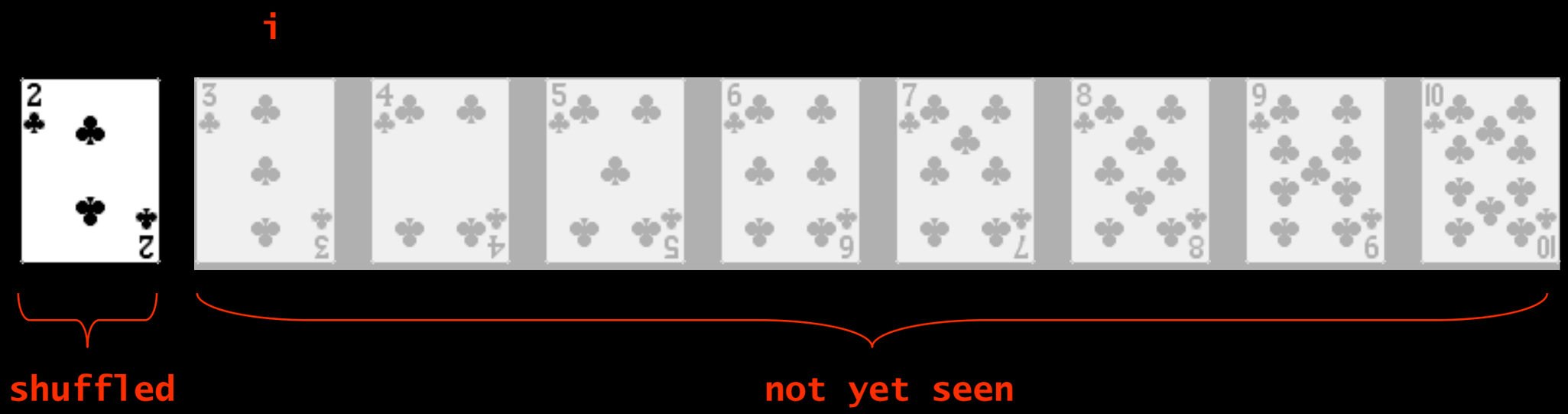


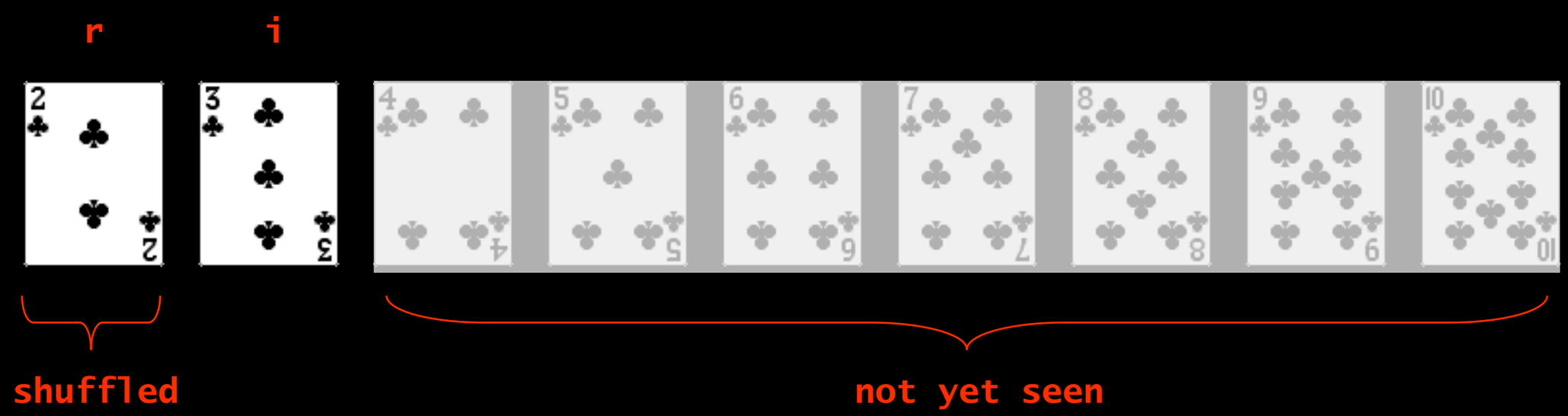

i
r
not yet seen

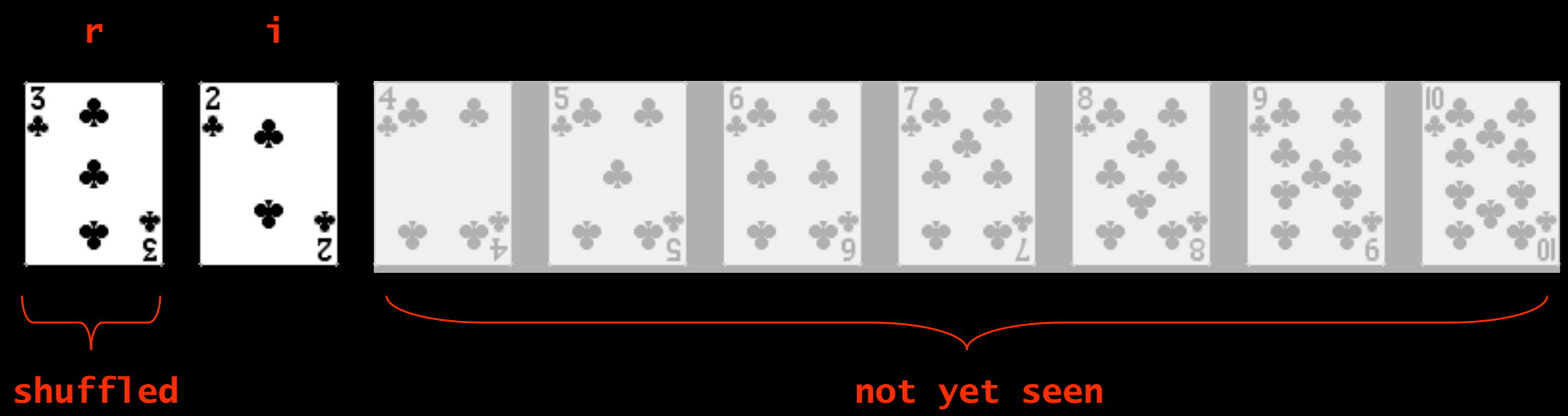
i r

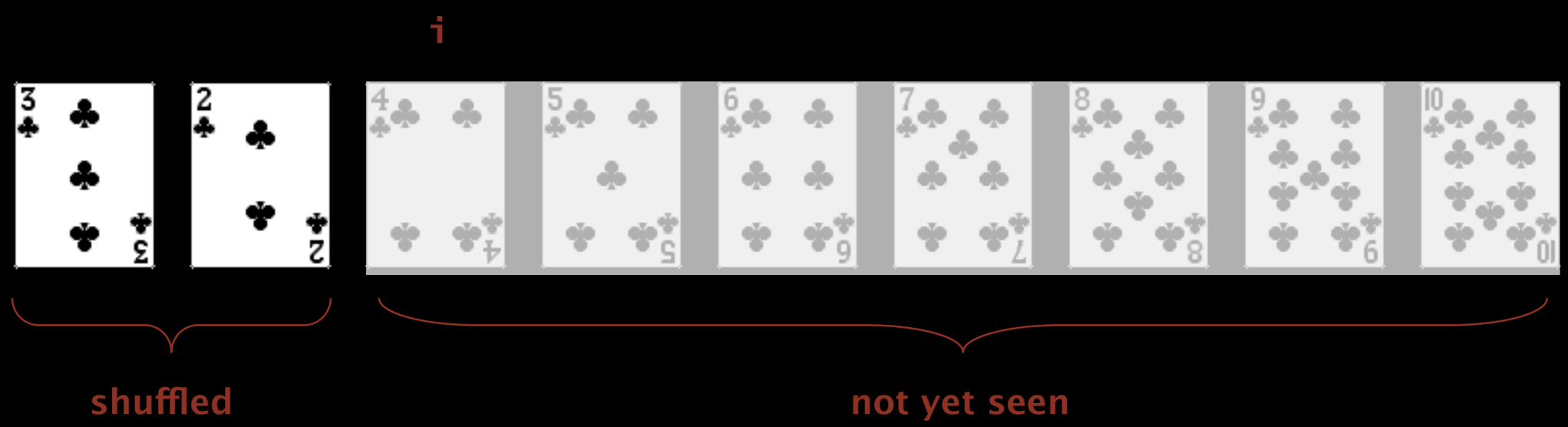


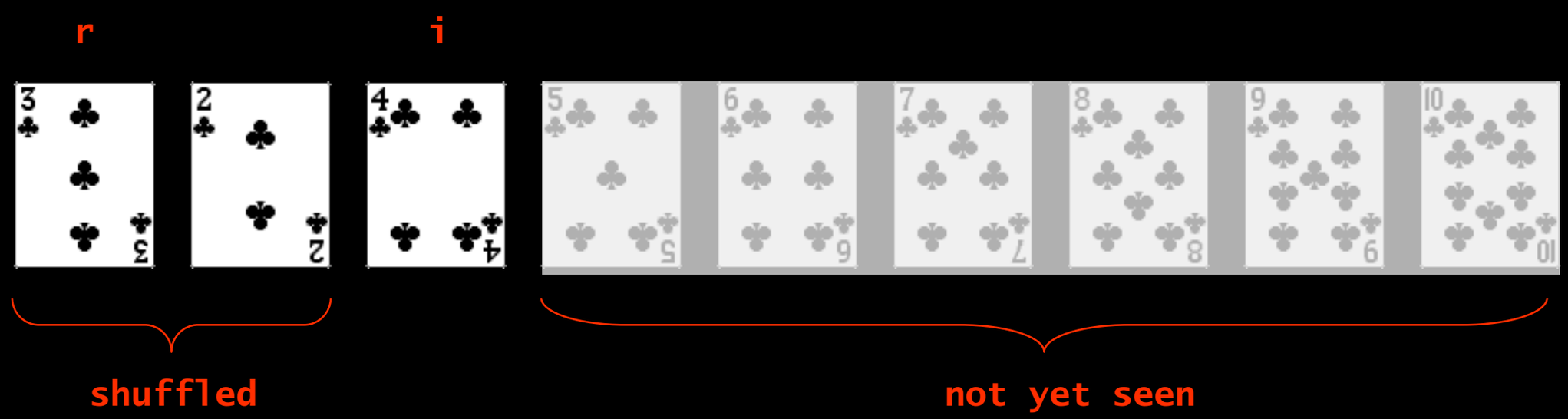
not yet seen

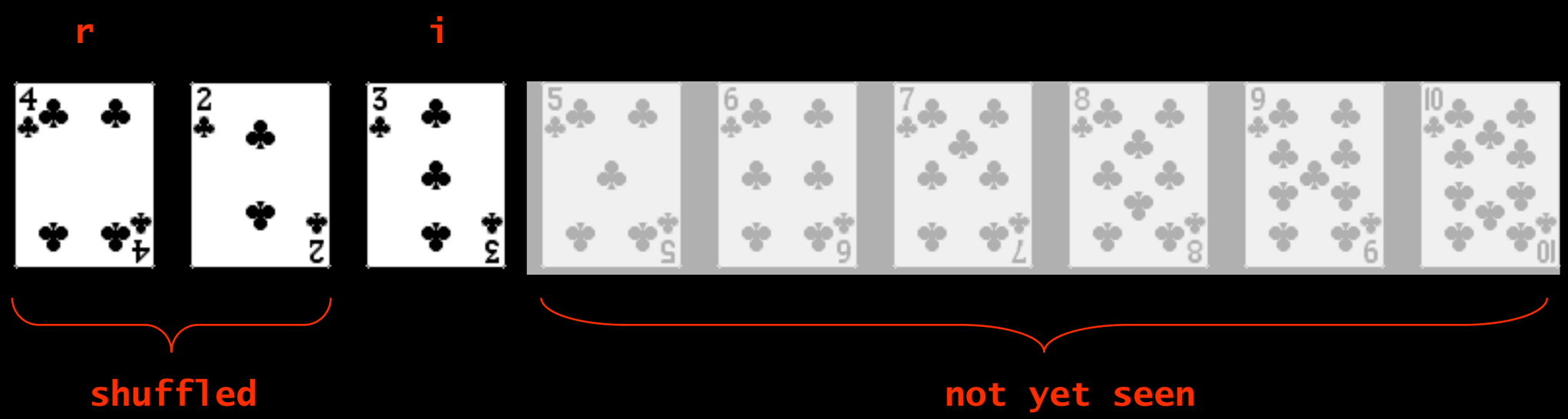

i
shuffled
not yet seen

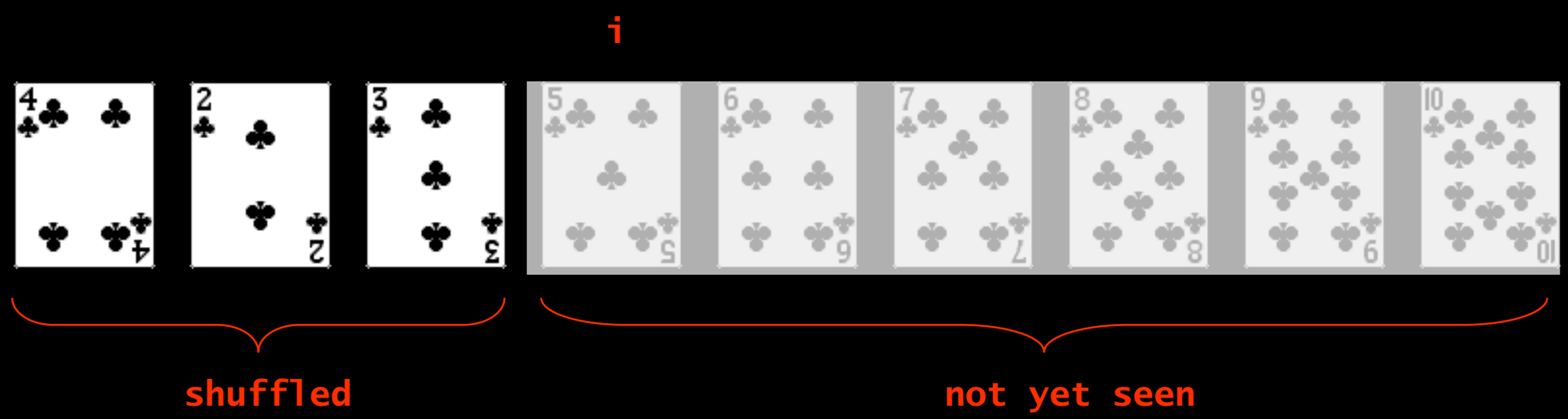

r
i
shuffled
not yet seen

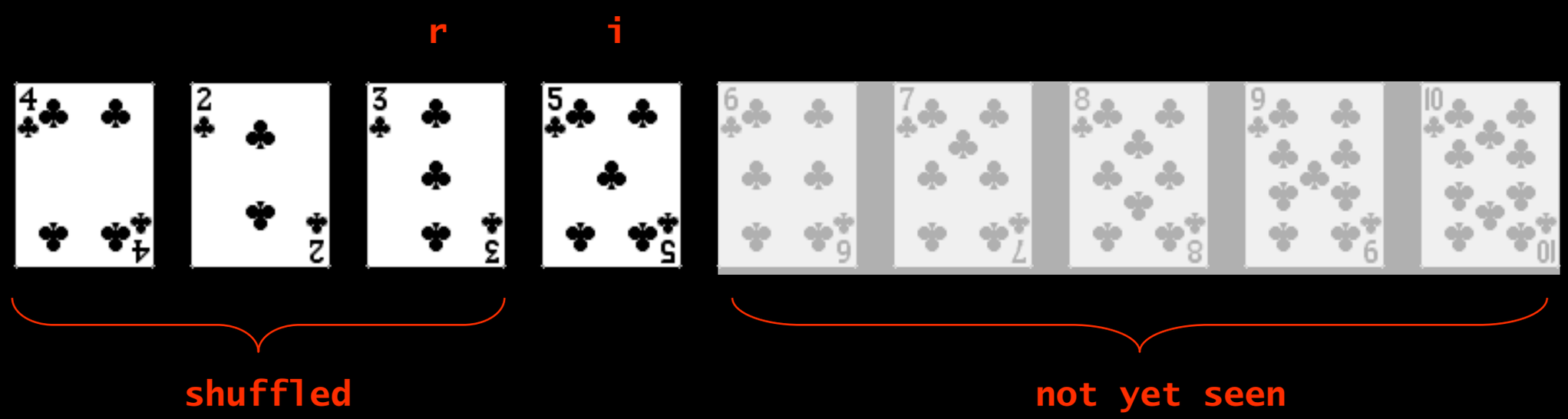

r
i
shuffled
not yet seen

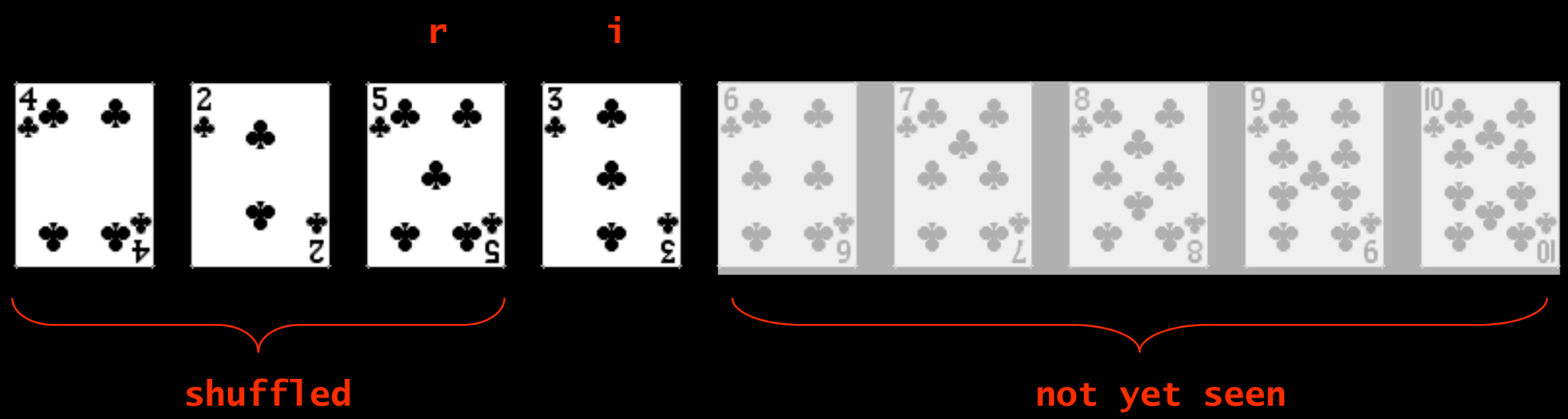

i
shuffled
not yet seen

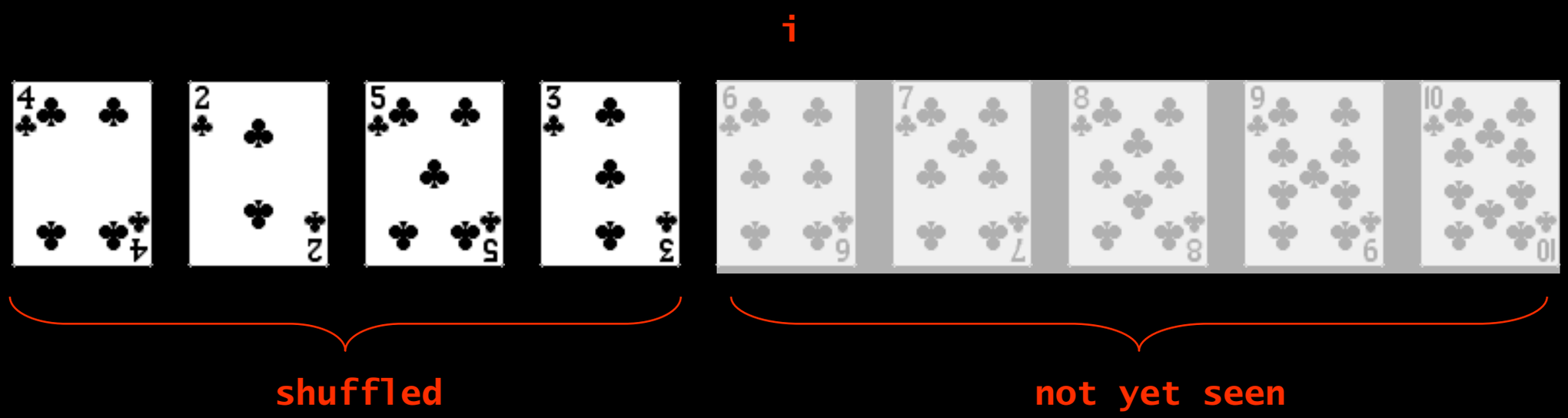

r
i
shuffled
not yet seen

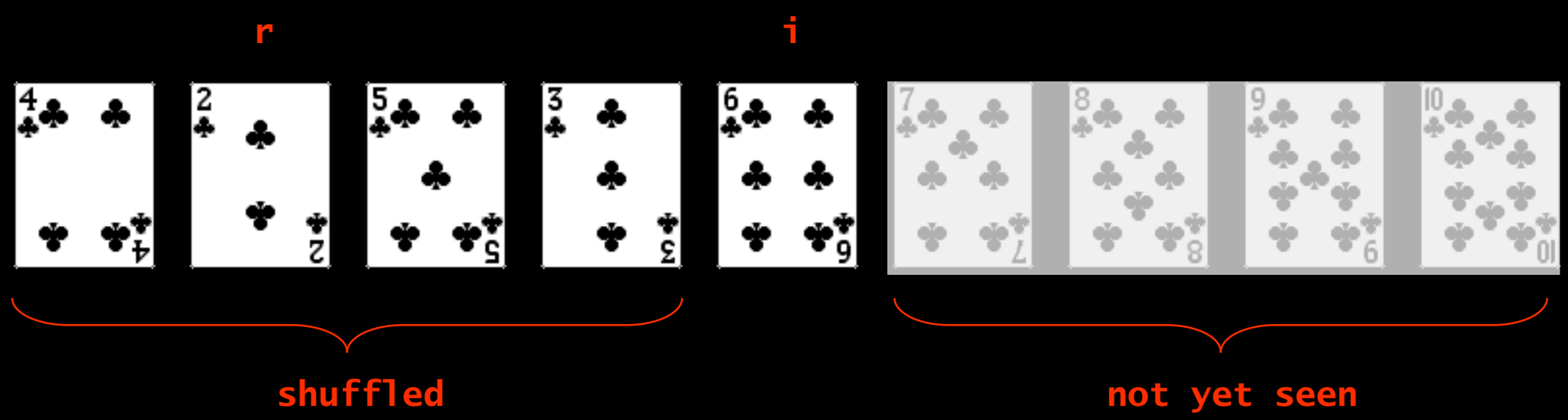

r
i
shuffled
not yet seen

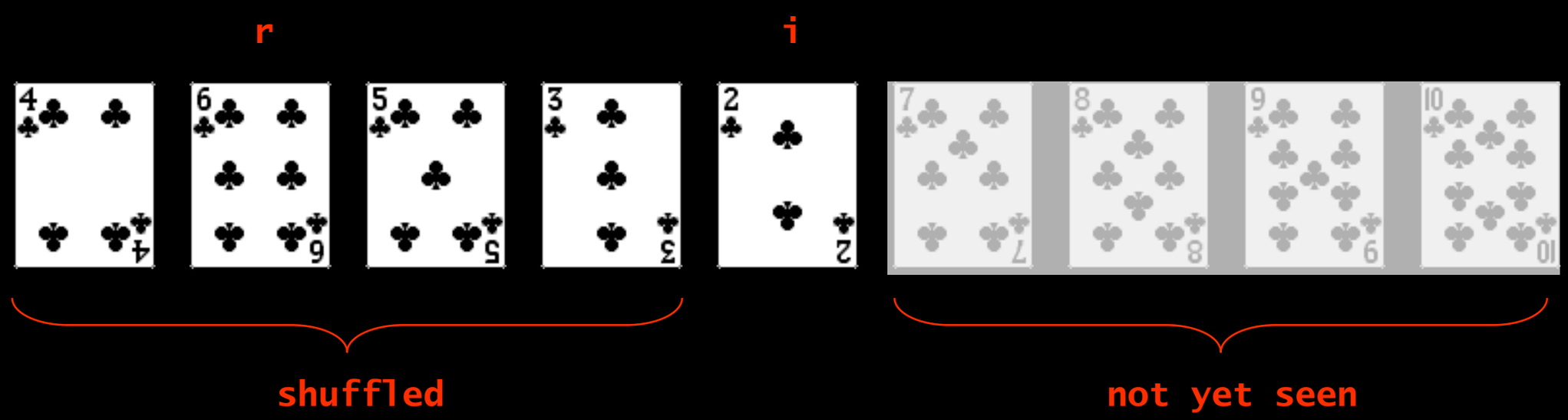

i
shuffled
not yet seen

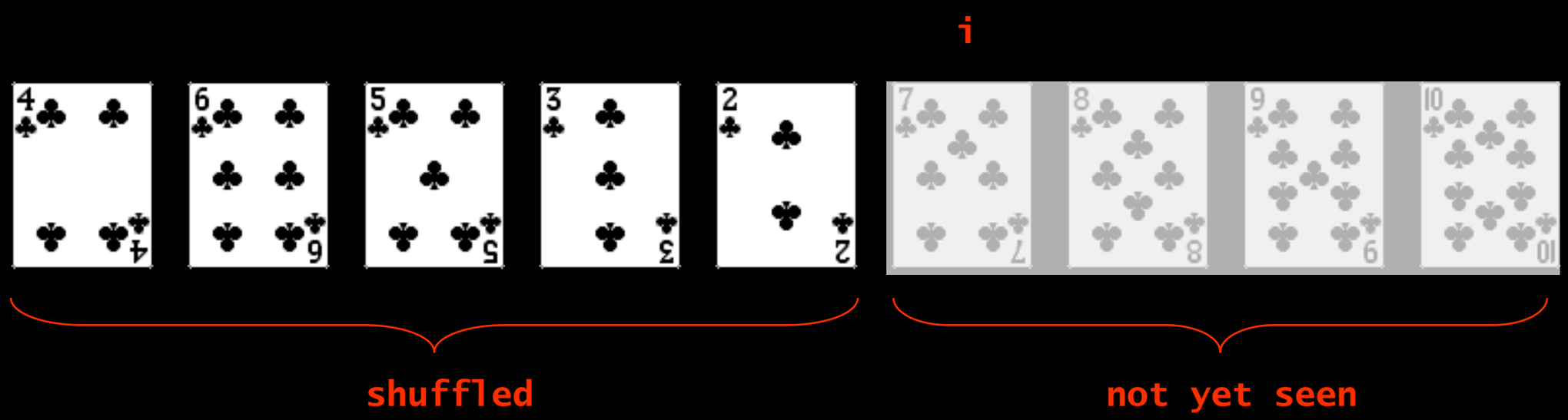

r
i
shuffled
not yet seen

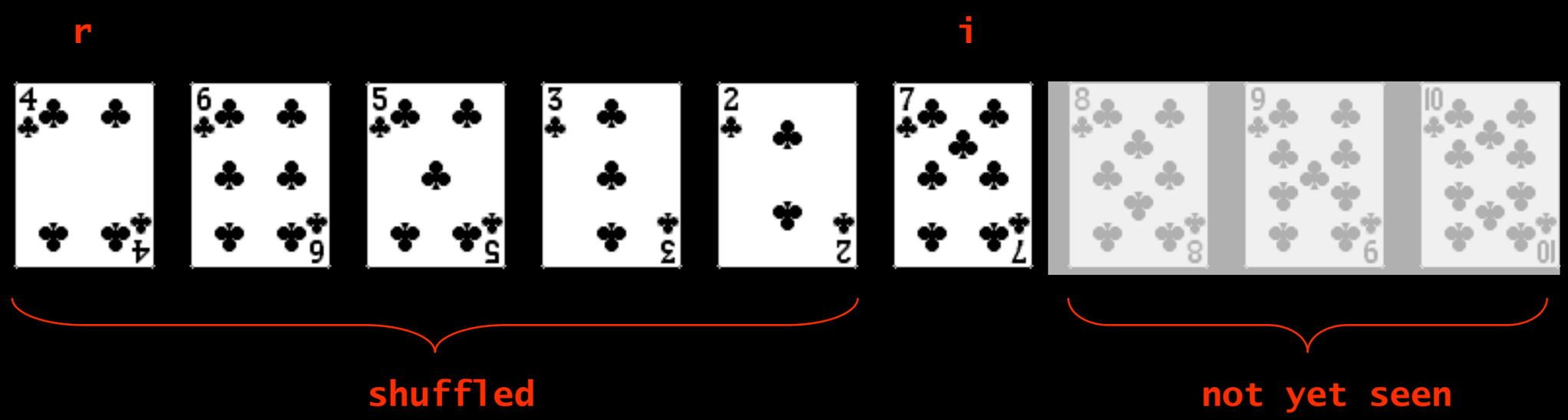

r
i
shuffled
not yet seen

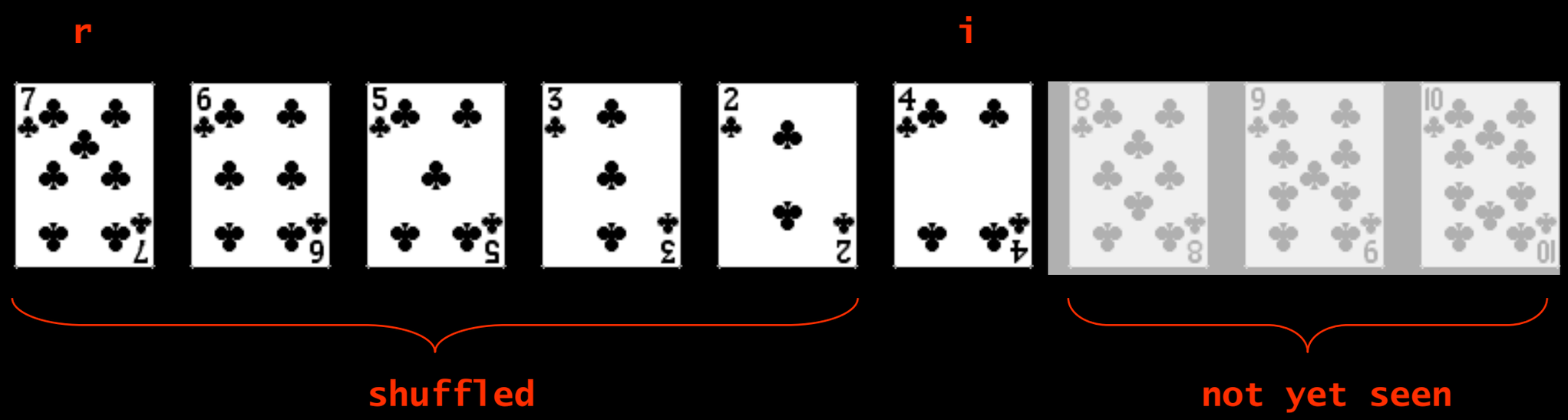

i
shuffled
not yet seen

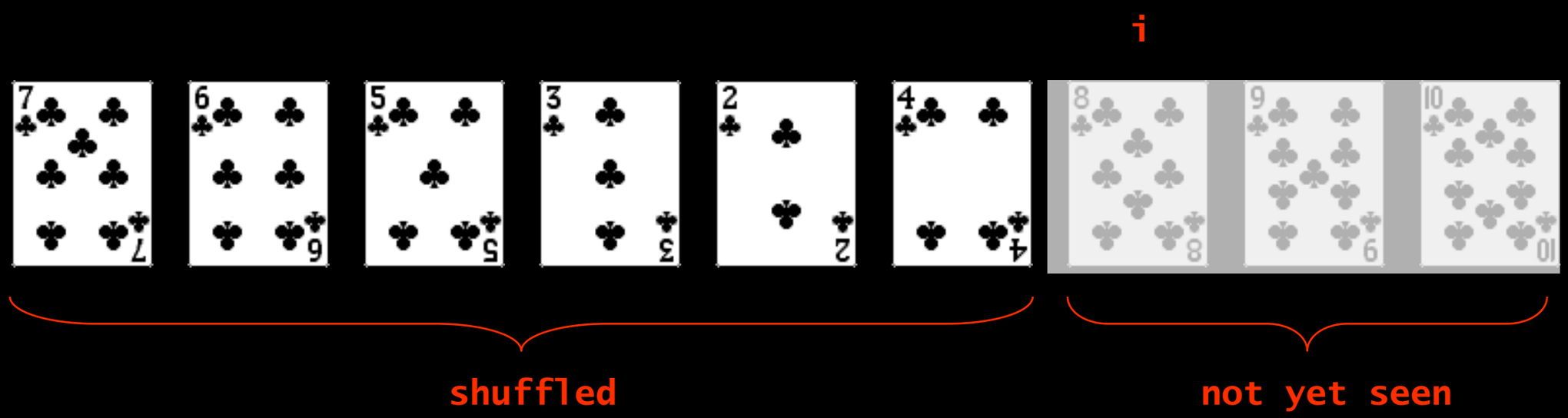

r
i
shuffled
not yet seen

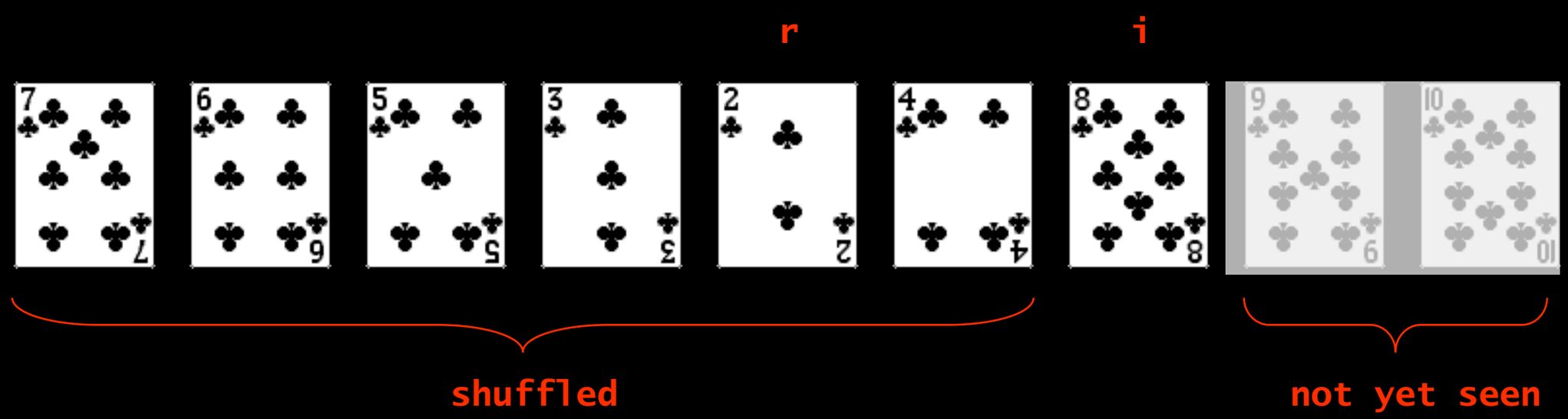

r
i
shuffled
not yet seen

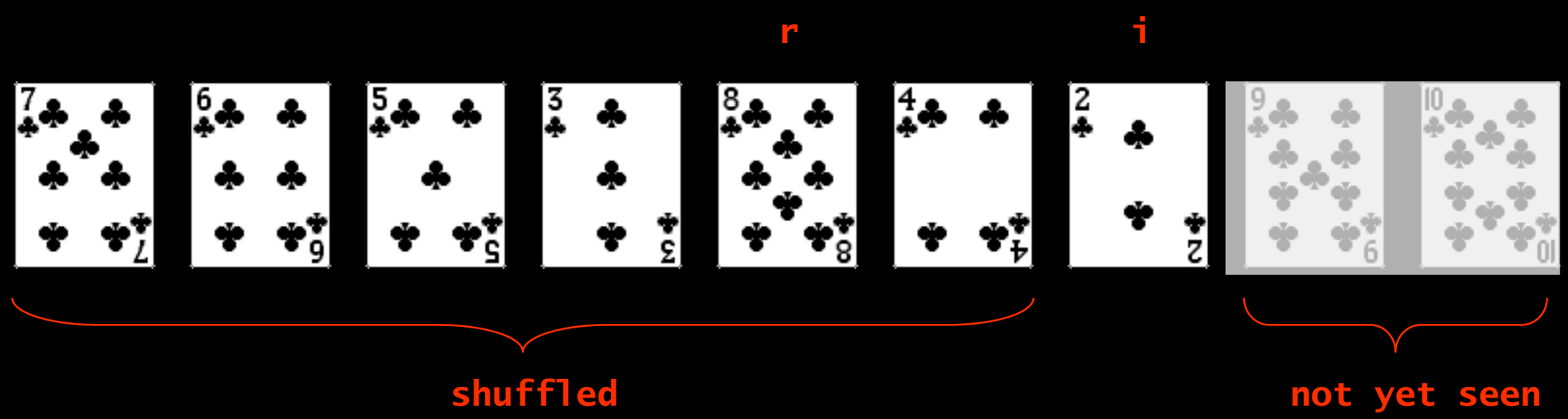

i
shuffled
not yet seen

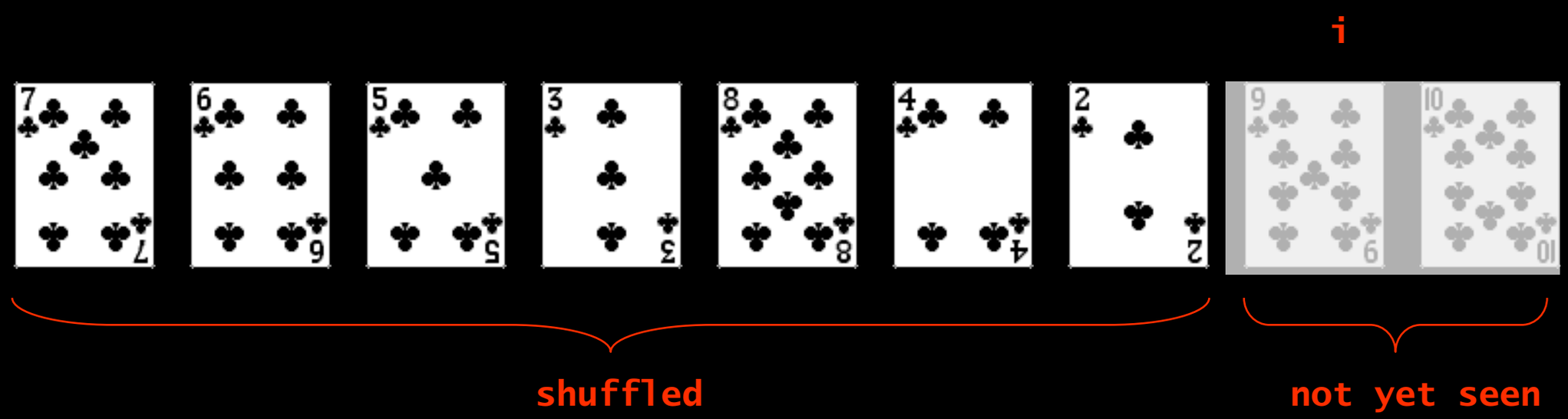

r
i
shuffled
not yet seen

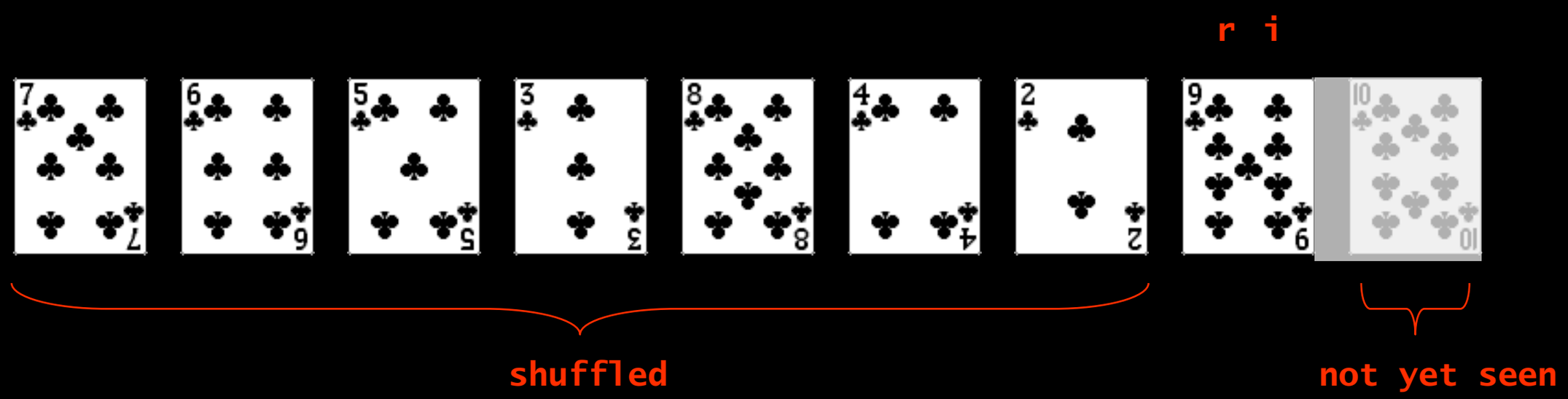

r
i
shuffled
not yet seen

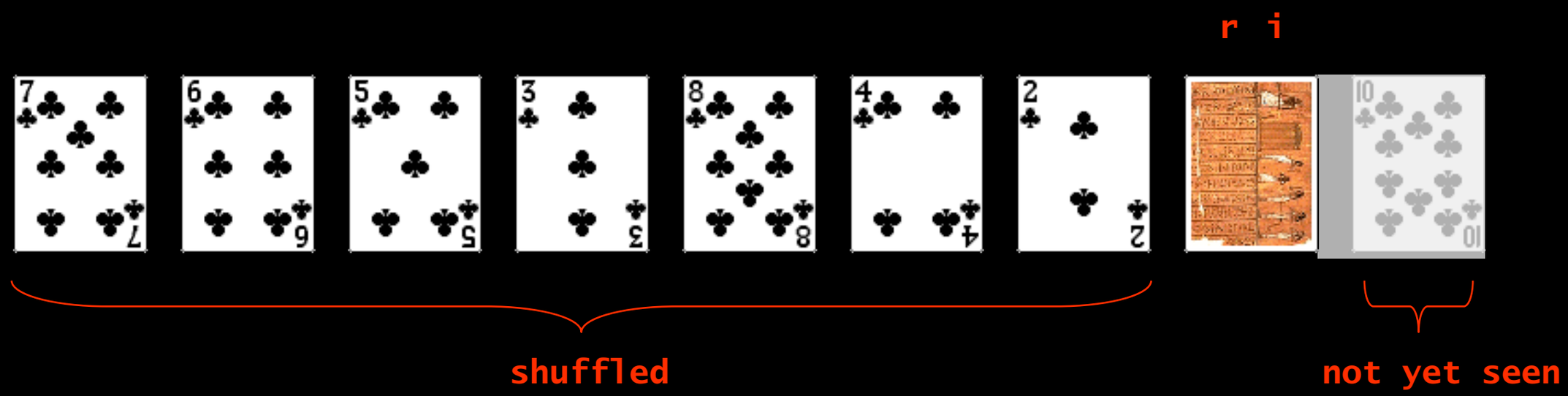

i
shuffled
not yet seen

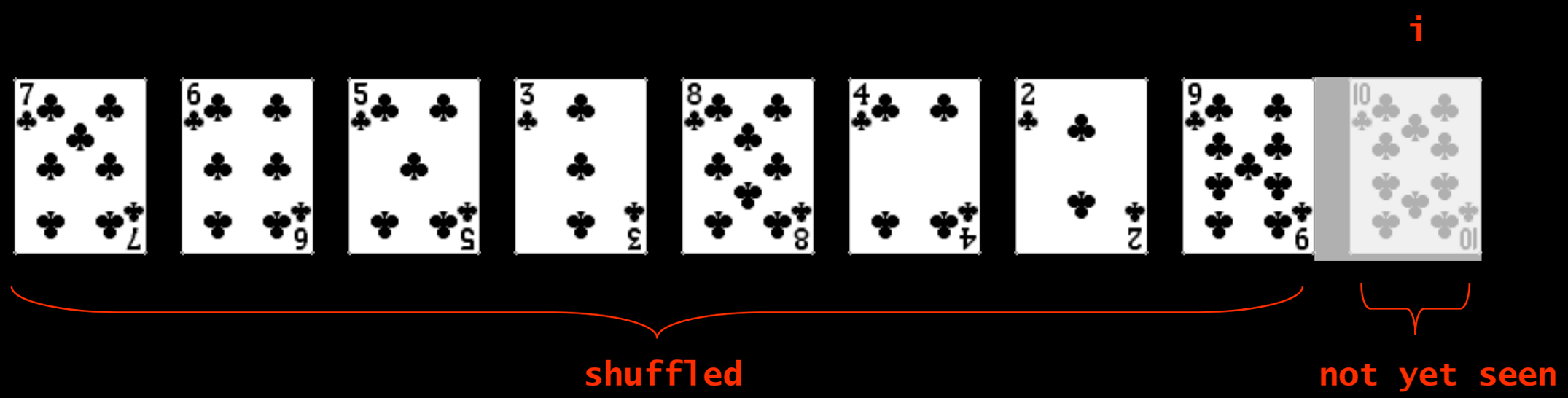

r
i
shuffled
not yet seen

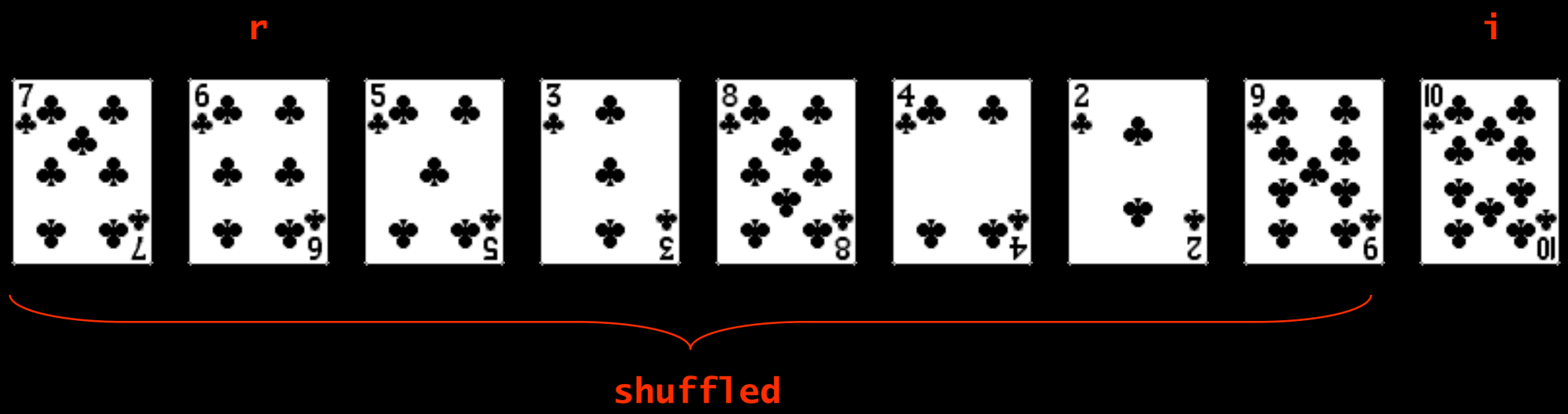

r
i
shuffled
not yet seen

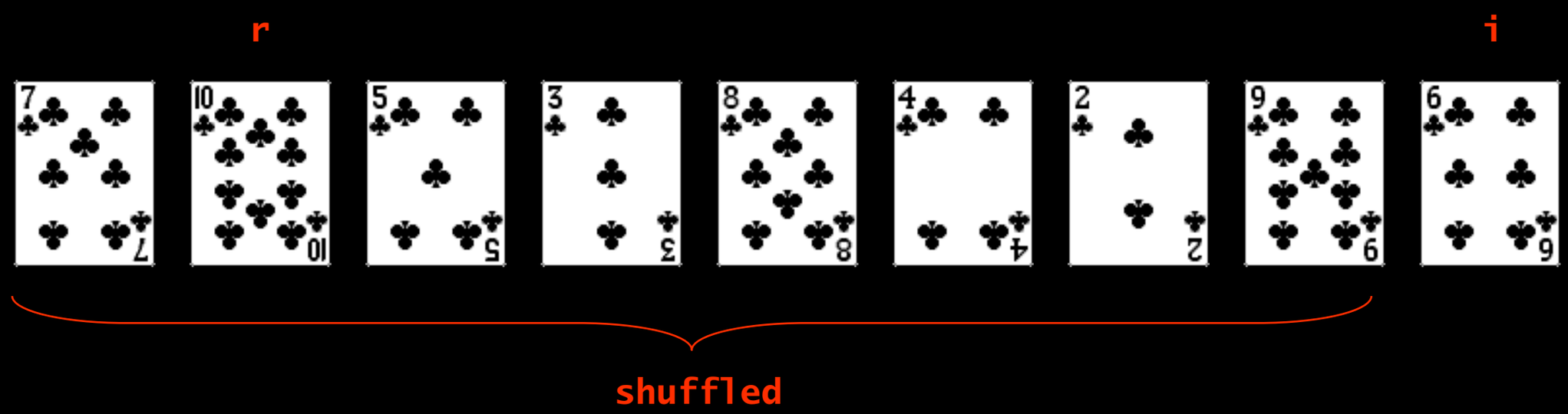

i
shuffled
not yet seen

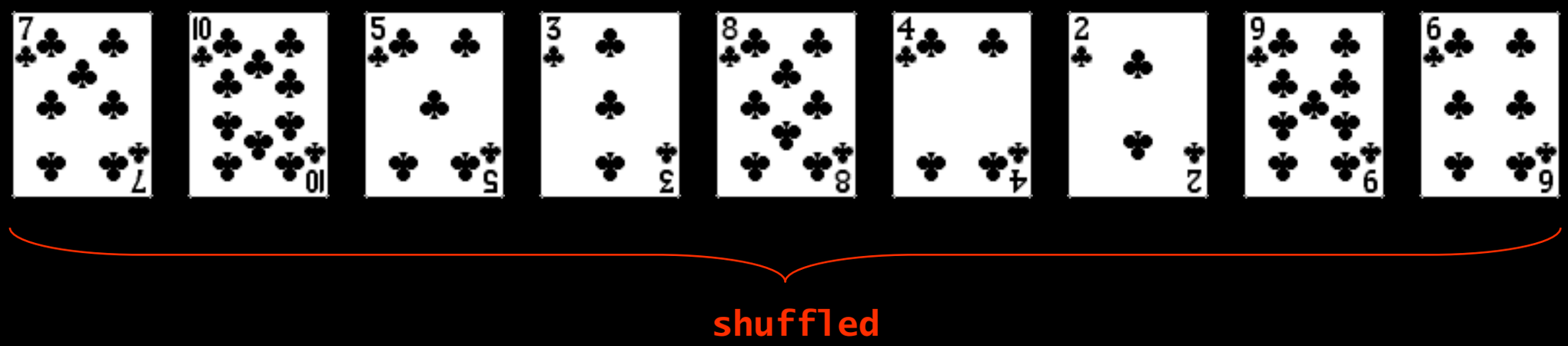

r i
shuffled
not yet seen


r i
shuffled
not yet seen


i
shuffled
not yet seen

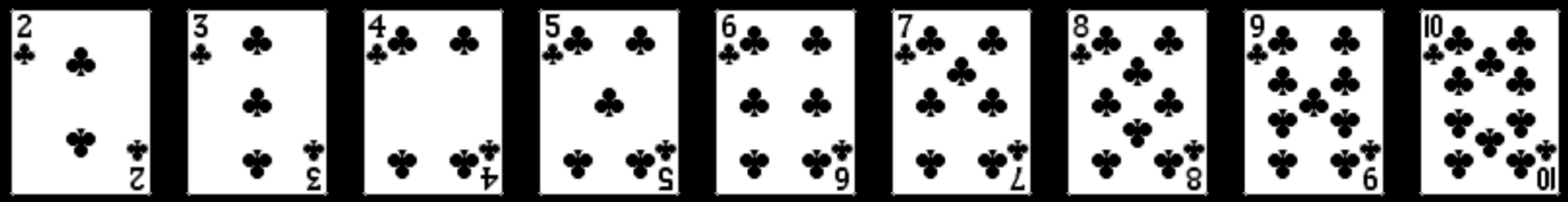

r
i
shuffled


r
i
shuffled


shuffled

Knuth shuffle

- In iteration `i`, pick integer `r` between `0` and `i` uniformly at random.
- Swap `a[i]` and `a[r]`.



Knuth shuffle

- In iteration `i`, pick integer `r` between `0` and `i` uniformly at random.
- Swap `a[i]` and `a[r]`.

common bug: between 0 and N – 1
correct variant: between i and N – 1

```
public class StdRandom
{
   ...
   public static void shuffle(Object[] a)
   {
      int N = a.length;
      for (int i = 0; i < N; i++)
      {
         int r = StdRandom.uniform(i + 1);
         exch(a, i, r);
      }
   }
}
```

between 0 and i

Broken Knuth shuffle

Q. What happens if integer is chosen between `0` and `N-1` ?

A. Not uniformly random!

instead of 0 and i

permutation	Knuth shuffle	broken shuffle
A B C	1/6	4/27
A C B	1/6	5/27
B A C	1/6	5/27
B C A	1/6	5/27
C A B	1/6	4/27
C B A	1/6	4/27

probability of each result when shuffling { A, B, C }

"The generation of random numbers is too important to be left to chance."

– ROBERT R. COVEYOU

REFERENCES

- Robert Sedgewich & Kevin Wayne

  Section 2.1