Review

- Arrays are a type just like any other
- ...therefore, they can be attributes of classes.
- Movie database: Movie array as an attribute.
  - Finding a movie: traverse the array until a match is found
  - Adding a movie: find available spot and storing there
  - Deleting a movie:
    * Finding the movie and setting the cell to null (creates fragmentation.)
    * Finding the movie, replacing its reference with the last reference in the array, resetting the last reference to null

Growing arrays

- An array has a finite and fixed amount of memory.
- In some applications we don't know a priori how much memory we need.
- C/C++ allow to grow arrays at will: big data-safety problem.
- Java does not allow to grow arrays directly, but we can simulate it indirectly:
- Growing arrays:
  - Whenever the array of interest fills up, a new, bigger array is created,
  - ...and the values of the old array are copied (shallowly) into the new array.
- Or, use class `ArrayList` or `Vector` from the standard library.

Growing arrays

- Change algorithm for adding a movie m:

1. Find first available cell

2. If an available cell is found:

   (a) Store m in that cell

3. Otherwise:

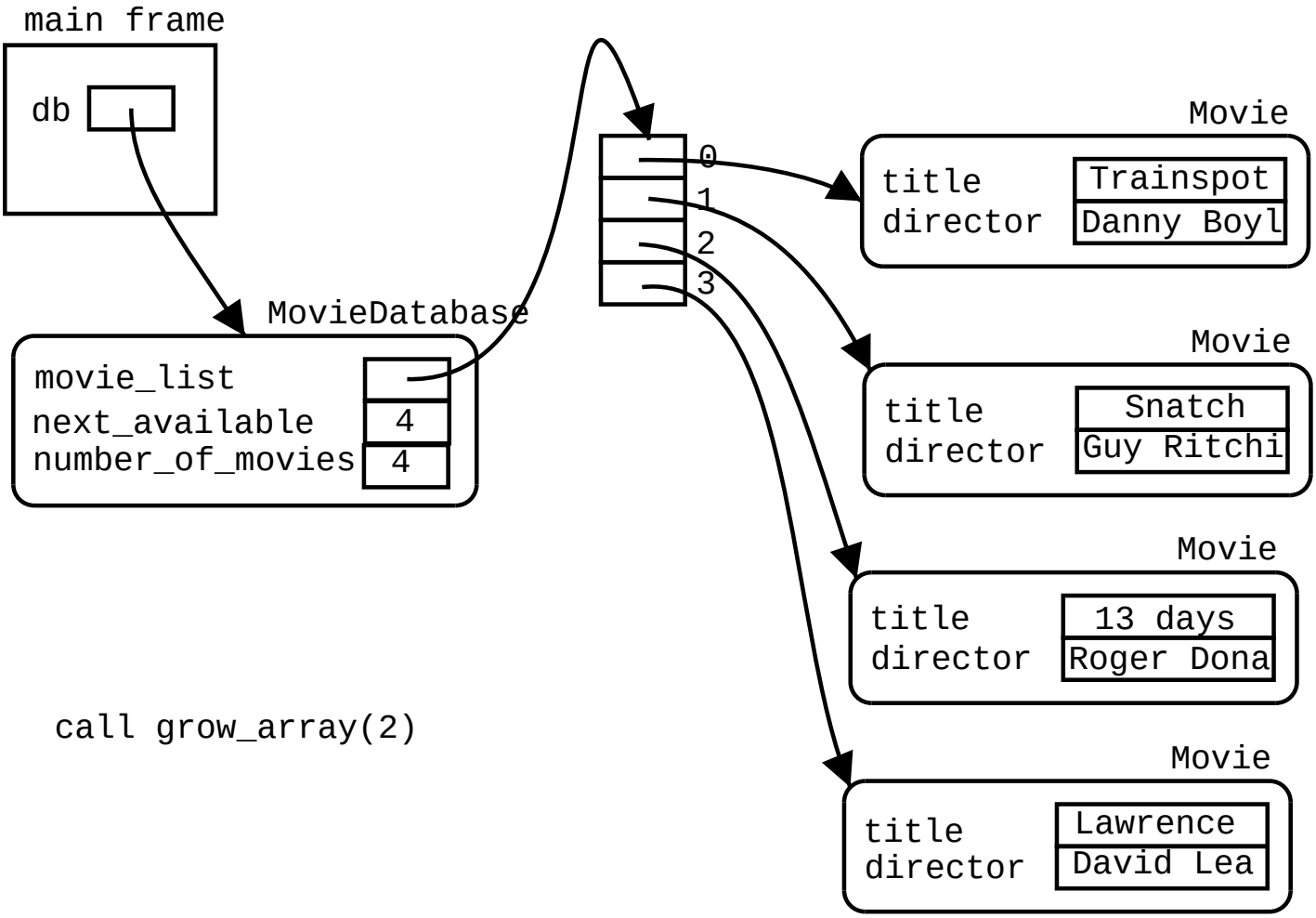
   (a) Grow the array (copying contents of the old to the new)
   (b) Find the first available cell in the new array (guarranteed to exist.)
   (c) Store m in that cell

Growing arrays

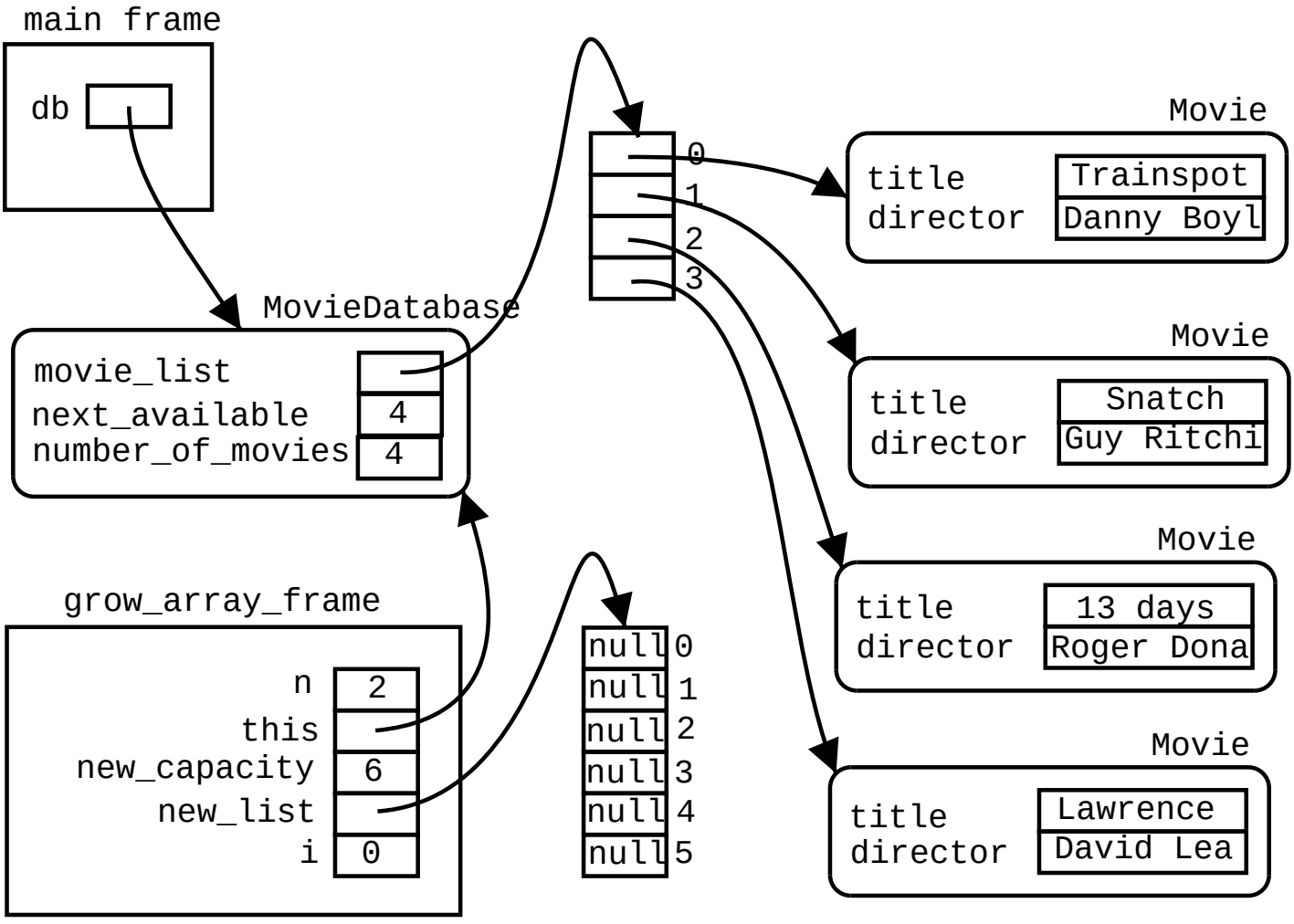
```
// In class MovieDatabase
private void grow_array(int n)
{
  int new_capacity = movie_list.length + n;
  Movie[] new_list = new Movie[new_capacity];
  int i = 0;
  while (i < movie_list.length) {
    new_list[i] = movie_list[i]; // shallow copy
    i++;
  }
  movie_list = new_list; // Update list reference
}
```

The method is private to ensure encapsulation so that only MovieDatabase objects can grow the movie lists.

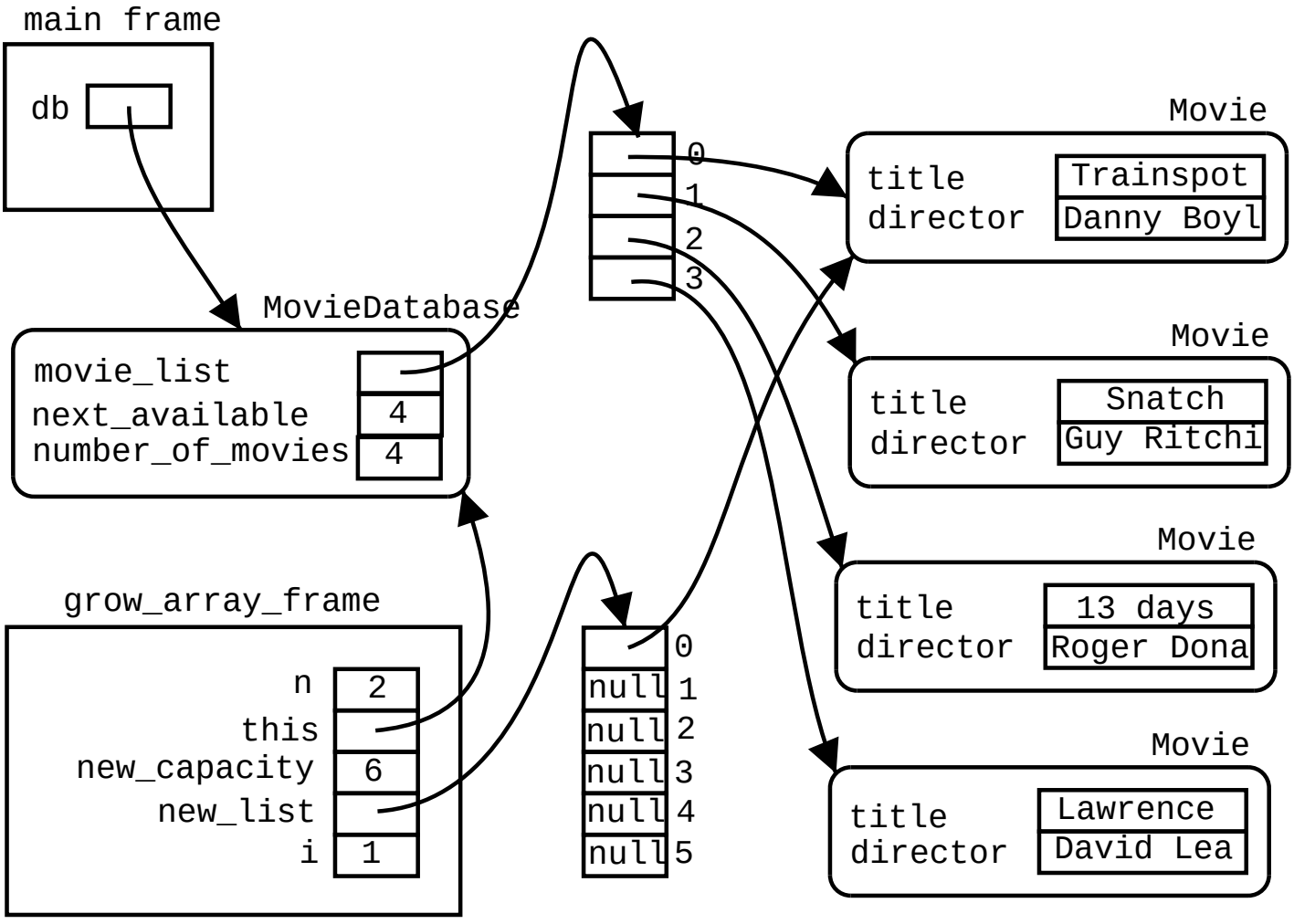
Growing arrays



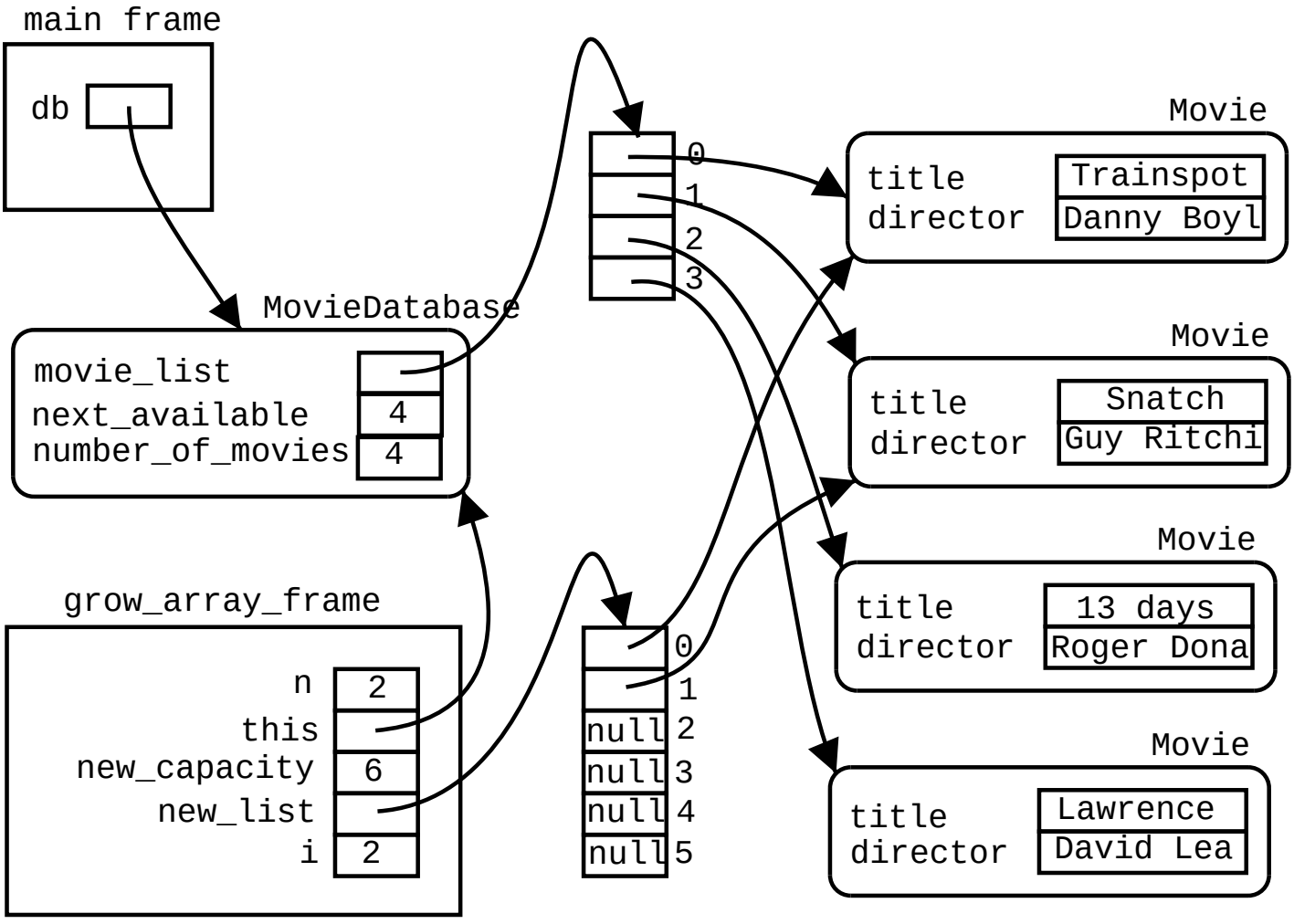
Growing arrays



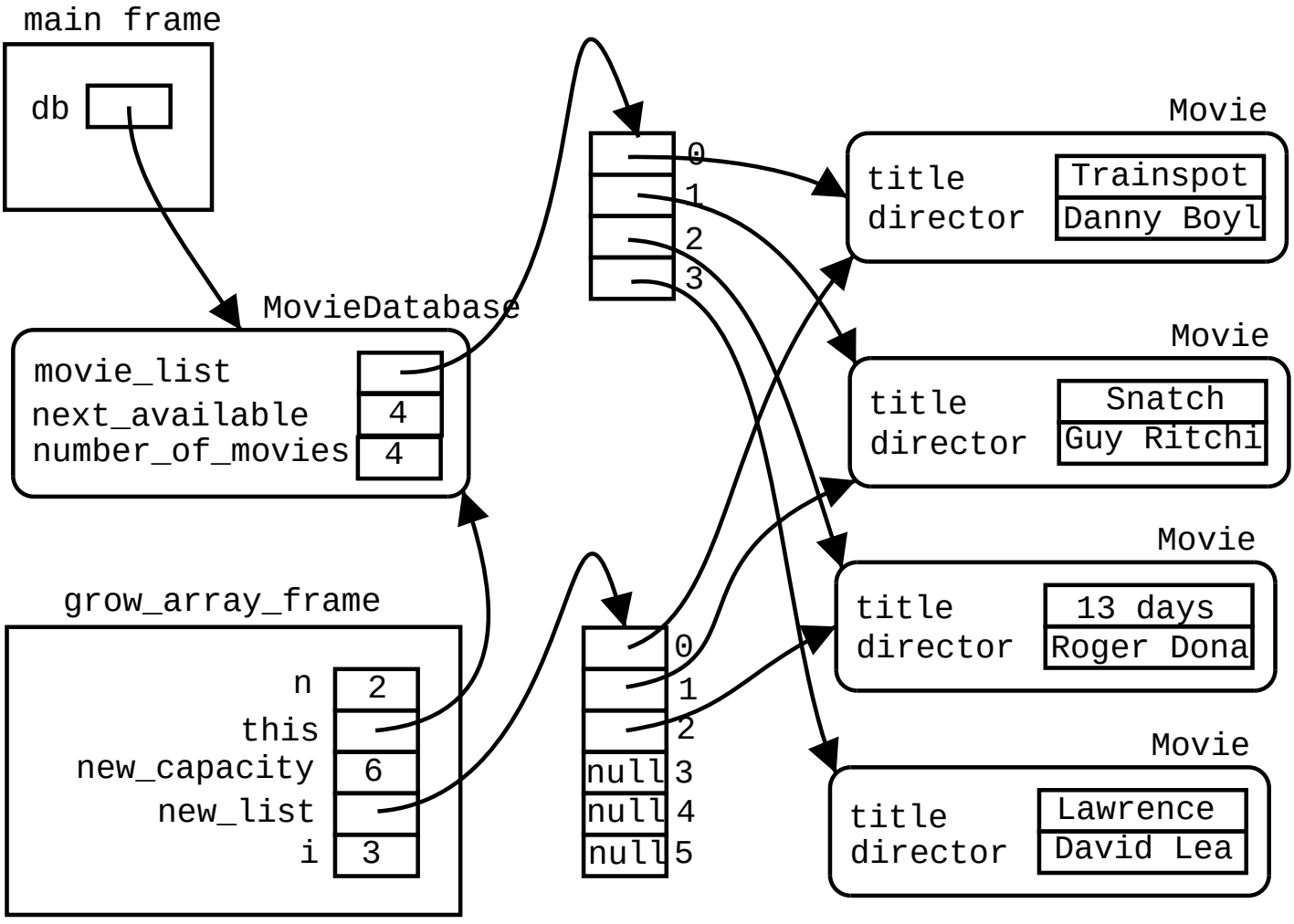
Growing arrays



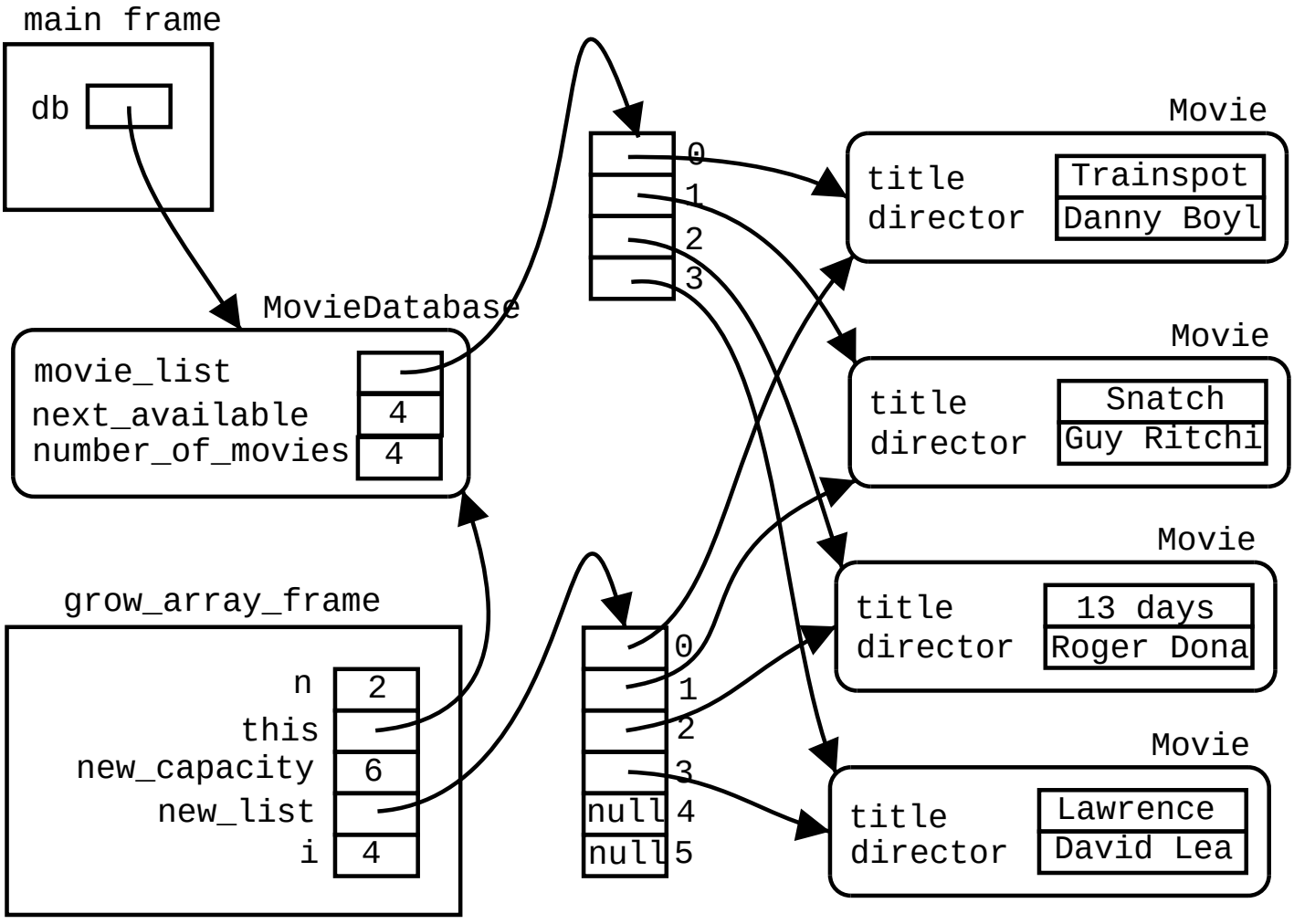
Growing arrays



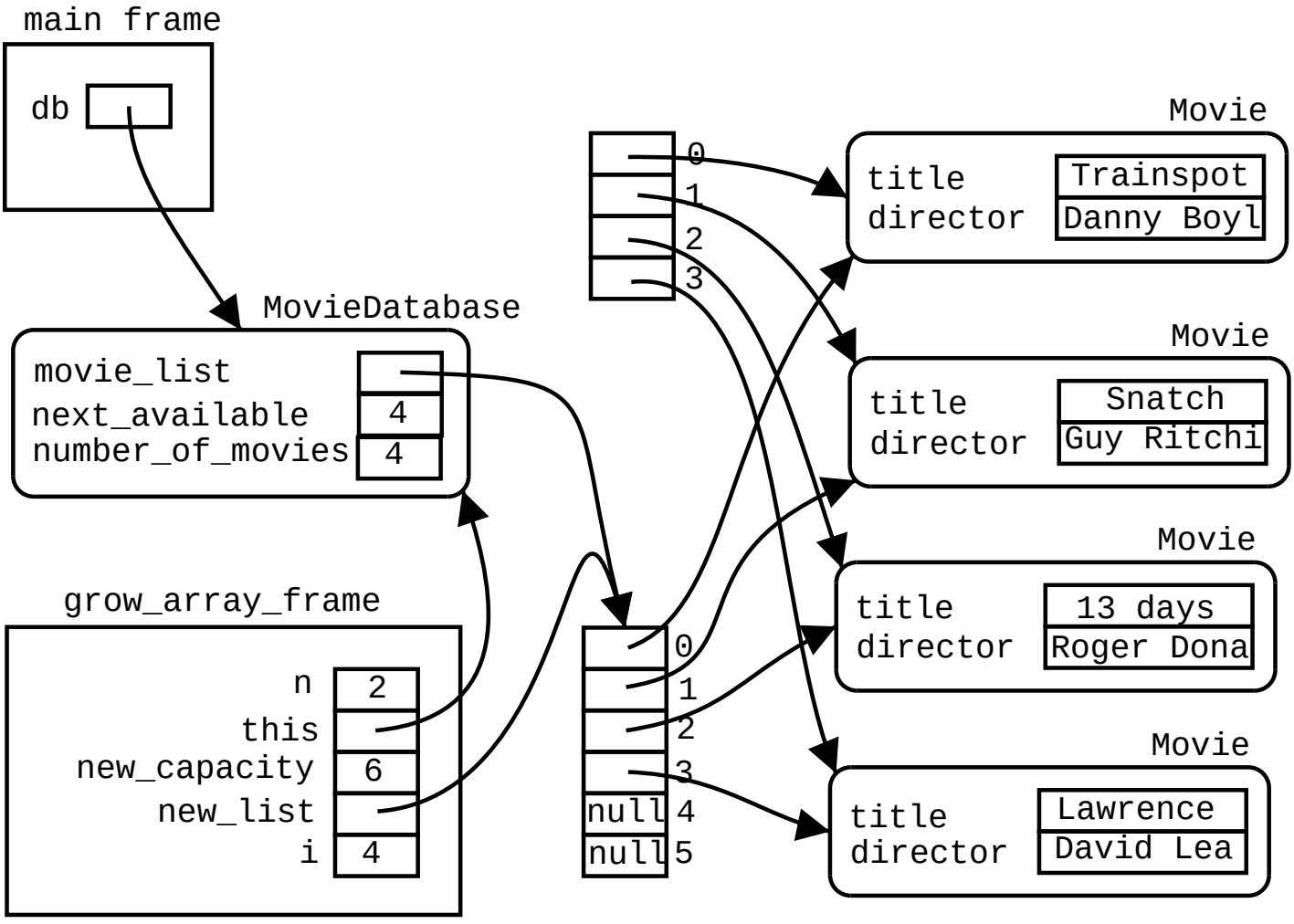
Growing arrays



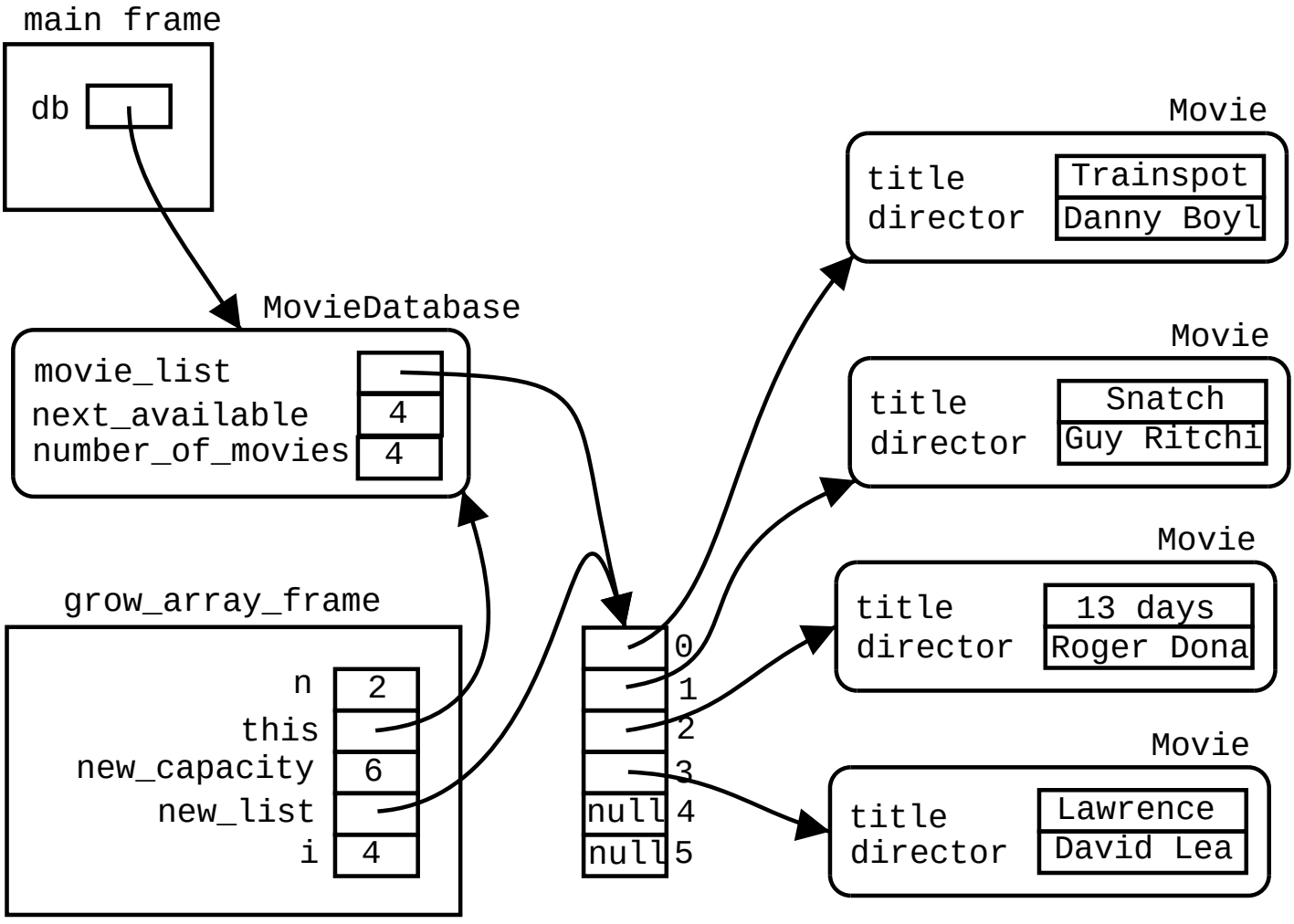
Growing arrays



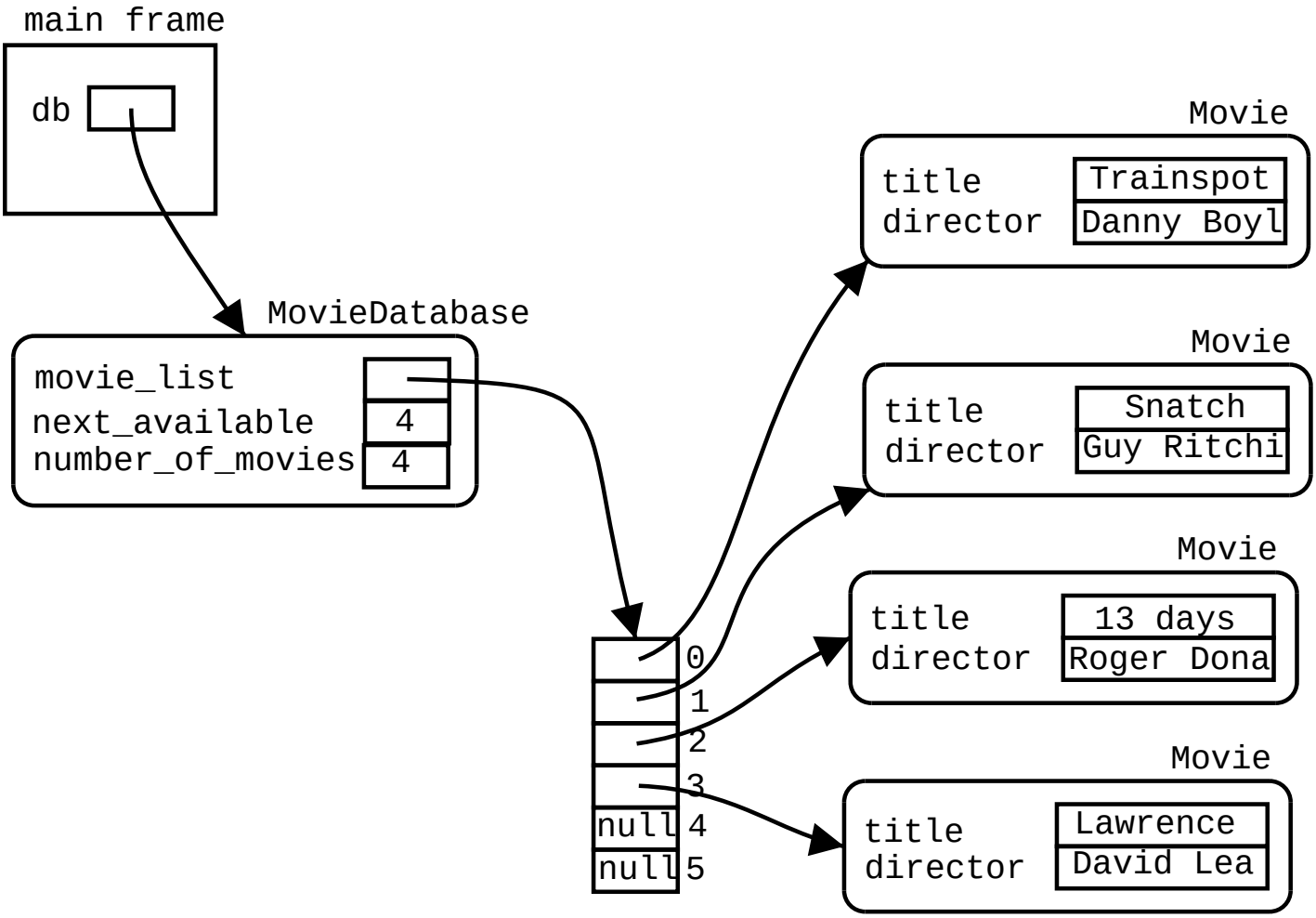
Growing arrays



Growing arrays



Growing arrays



Growing arrays

```
// Version 1: explicit search for available slot
public void add_movie(Movie m)
{
  // Find available slot
  int index = 0;
  while (index < movie_list.length
      && movie_list[index] != null) {
    index++;
  }
  // If available slot found, store it
  if (index < movie_list.length) {
    movie_list[index] = m;
  }
  // Otherwise
  else {
    int l = movie_list.length;
    grow_array((int)(l * 0.10));
    movie_list[l] = m;
  }
  number_of_movies++;
}
```

Growing arrays

```
// Version 2: Optimized (with non-fragmented array
public void add_movie(Movie m)
{
  // If available slot found, store it
  if (next_available < movie_list.length) {
    movie_list[next_available] = m;
  }
  // Otherwise
  else {
    int l = movie_list.length;
    grow_array((int)(l * 0.10));
    movie_list[l] = m;
  }
  next_available++;
}
```

The Vector and ArrayList classes

- Two classes which encapsulate growing arrays
- The two provide essentially the same functionality, but have a slightly different underlying implementation.
- Vector has methods

```
void setElementAt(Object o, int index)
Object elementAt(int index)
int size()
boolean contains(Object o)
int index_of(Object o)
// ... etc
```

- ArrayList has methods

```
Object get(int index)
void set(int index, Object o)
void add(Object o)
int size()
// ... etc
```

Sorting

- Classical problem in Computer Science
- Problem: Given an array of objects, sort the array by some *key*.
- For example: Sort an array of students by name, or sort an array of products by price.
- Solution for small arrays using only conditionals is not *scalable*.

Sorting

- Analysis:
  - Objects:
    - An array of objects
  - Relationships:
    - Each object *has a* key (and maybe other attributes.)
    - For example, if the objects are of class Student, the key can be the name, to sort by name, or the id, to sort by id.
    - Each pair of keys can be compared: there is a (total) order relation between the keys.
  - Input: the array
  - Output: the array, or a copy, where the objects are placed in order (ascending) with respect the the key of interest.

- Small variation of the problem: sort an array of numbers: the order relation between keys is simply <=.

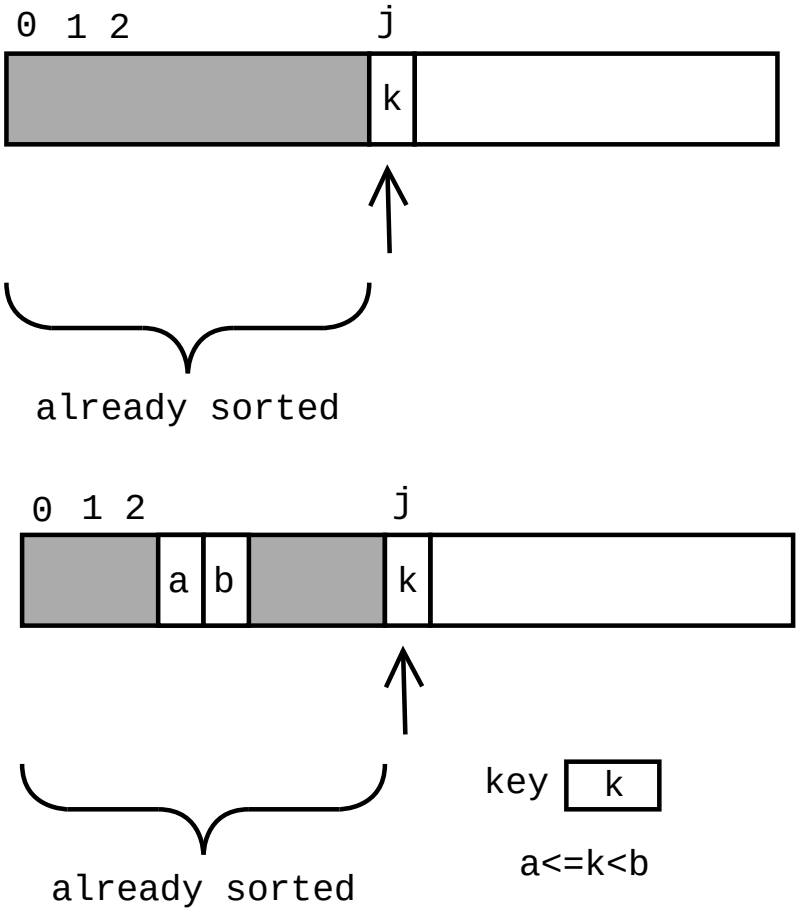
Sorting algorithms

- Insertion sort
- Selection sort
- Bubble sort
- Heap sort
- Merge sort
- Quick sort
- Bucket sort
- Counting sort
- Radix sort
- Sorting networks

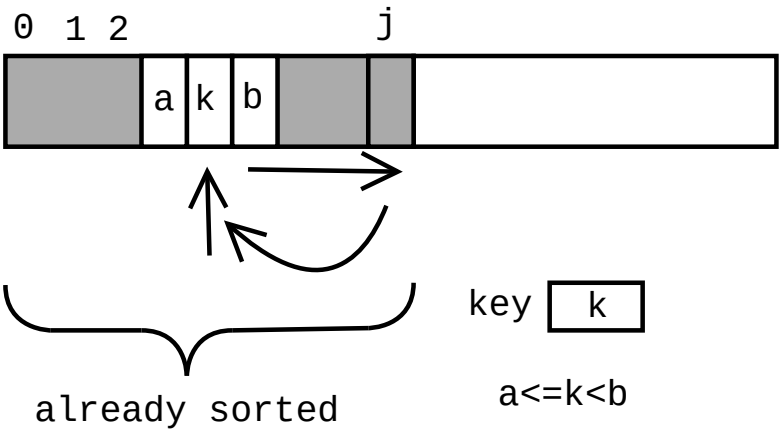
Insertion sort

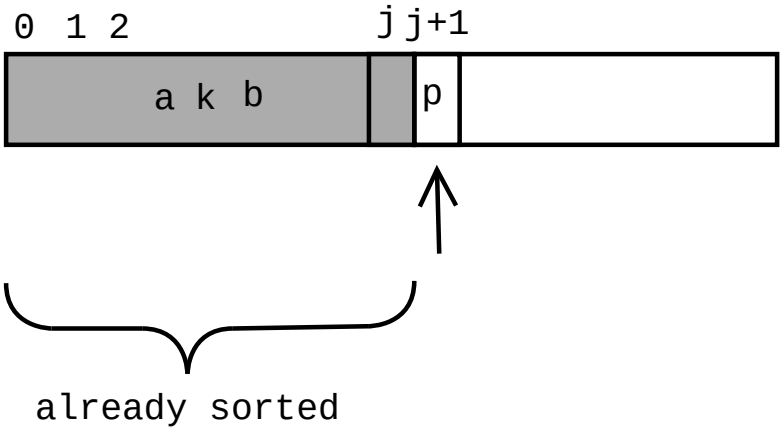
- Notation (not Java!): `a[i..j]` is the part of the array from the `i`-th index to the `j`-th index.

- Idea: sorting a set of cards can be done by inserting a card in the subset of the cards which are already sorted.



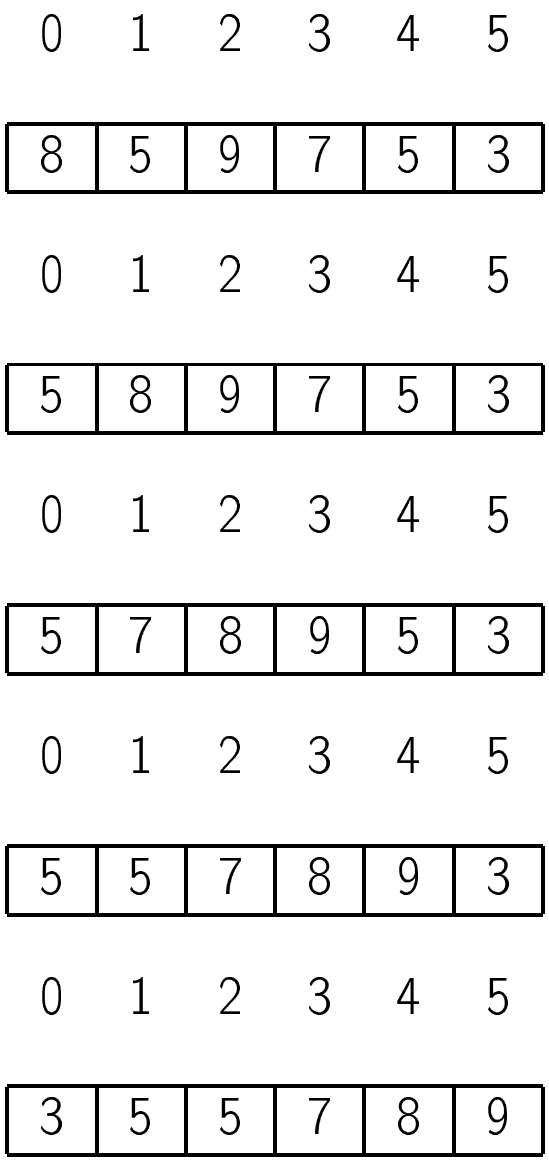
Insertion sort





Insertion sort

- Example:



Insertion sort

- Algorithm:
  - Input: an array of numbers a

1. If a[1]<a[0] swap them.
2. Insert a[2] into a[0..1]
3. Insert a[3] into a[0..2]
4. Insert a[4] into a[0..3]
5. ...
6. Insert a[length of a-1] into a[0..length of a-2]

Insertion sort

- Algorithm refined:

1. For each `j` from 1 to the length of `a-1`
   (a) Insert `a[j]` into the sorted subarray `a[0..j-1]`

- Algorithm refined: (Full algorithm)

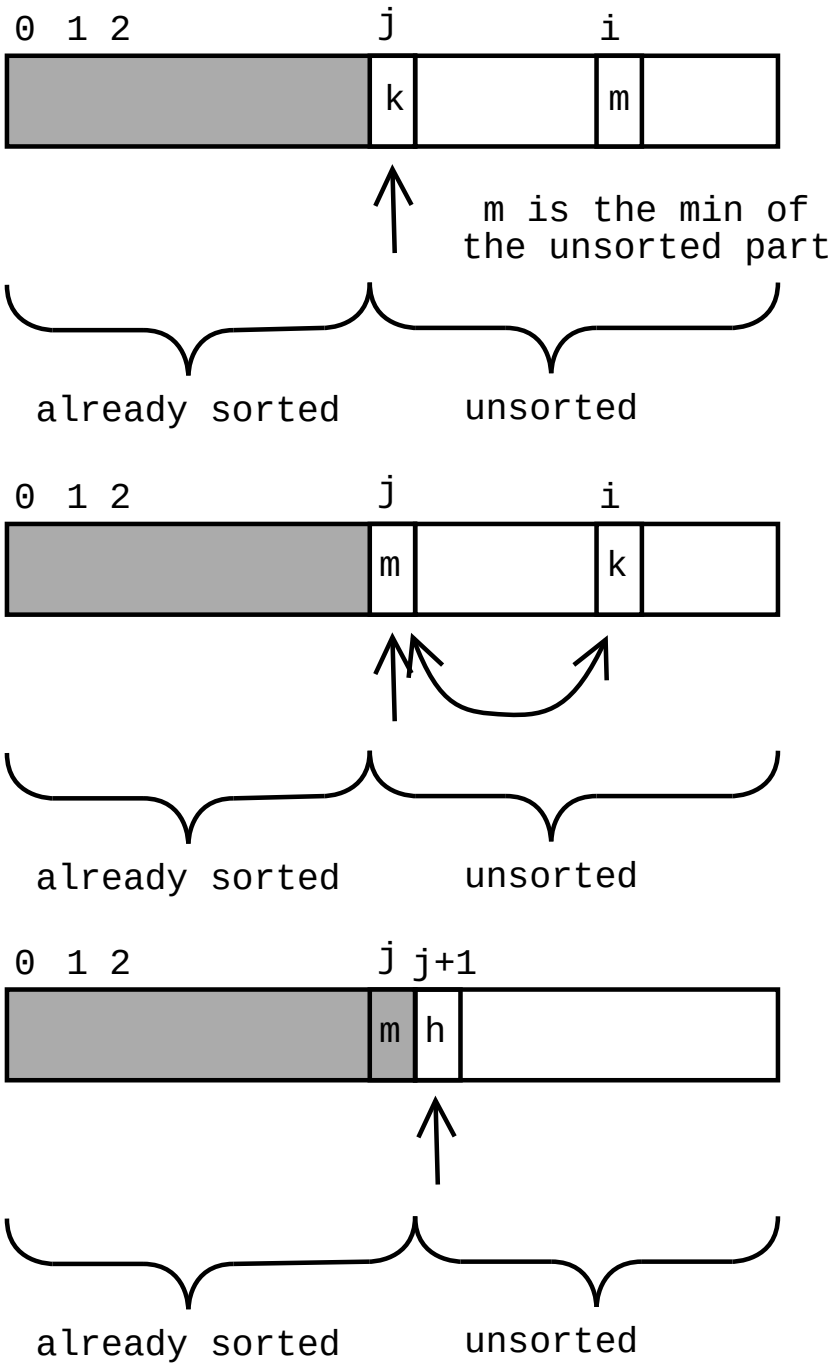
1. For each `j` from 1 to the length of `a-1`
   (a) Set `key` to `a[j]`
   (b) Set `i` to `j - 1`
   (c) While `i >= 0` and `a[i] > key` do
      i. Set `a[i+1]` to `a[i]`
      ii. Decrement `i` by 1
   (d) Set `a[i+1]` to `key`

Insertion sort

- Implementation

```
void insertion_sort(int[] a)
{
  int i, j, key;
  for (j = 1; j < a.length; j++) {
    key = a[j];
    i = j - 1;
    while (i >= 0 && a[i] > key) {
      a[i+1] = a[i];
      i--;
    }
    a[i+1] = key;
  }
}
```

Selection sort



Selection sort

- Idea:

1. Look for the minimum m0 in a[1..length a-1].
2. Swap the minimum and a[0].
3. Look for the minimum m1 in a[2..length a-1]
4. Swap m1 with a[1]
5. Look for the minimum m2 in a[3..length a-1]
6. Swap m2 with a[2]
7. Look for the minimum m3 in a[4..length a-1]
8. Swap m3 with a[3]
9. ...

Selection sort

- Algorithm

1. For each j from 0 to length a - 2 do
   (a) Let min_index to be the index of the minimum in a[j+1..length a-1]
   (b) Swap a[min_index] and a[j]

- Algorithm refined

1. For each j from 0 to length a - 2 do
   (a) Let minimum be a[j]
   (b) Set min_index to j
   (c) For each i from j+1 to the length a - 1 do
      i. If a[i] < minimum then
         A. Set minimum to a[i]
         B. Set min_index to i
   (d) Swap a[min_index] and a[j]

Selection sort

- Implementation

```
void selection_sort(int[] a)
{
  int minimum, min_index, temp;
  for (int j = 0; j <= a.length - 2; j++) {
    minimum = a[j];
    min_index = j;
    for (int i = j + 1; i <= a.length - 1; i++) {
      if (a[i] < minimum) {
        minimum = a[i];
        min_index = i;
      }
    }
    temp = a[j];
    a[j] = a[min_index];
    a[min_index] = temp;
  }
```