
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.

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
```

The Vector and ArrayList classes

```
public class Library {
    private ArrayList book_list;

    public Library()
    {
        book_list = new ArrayList();
    }
    public void add_book(Book m)
    {
        book_list.add(m);
    }
    // ...
}
```

Growing arrays

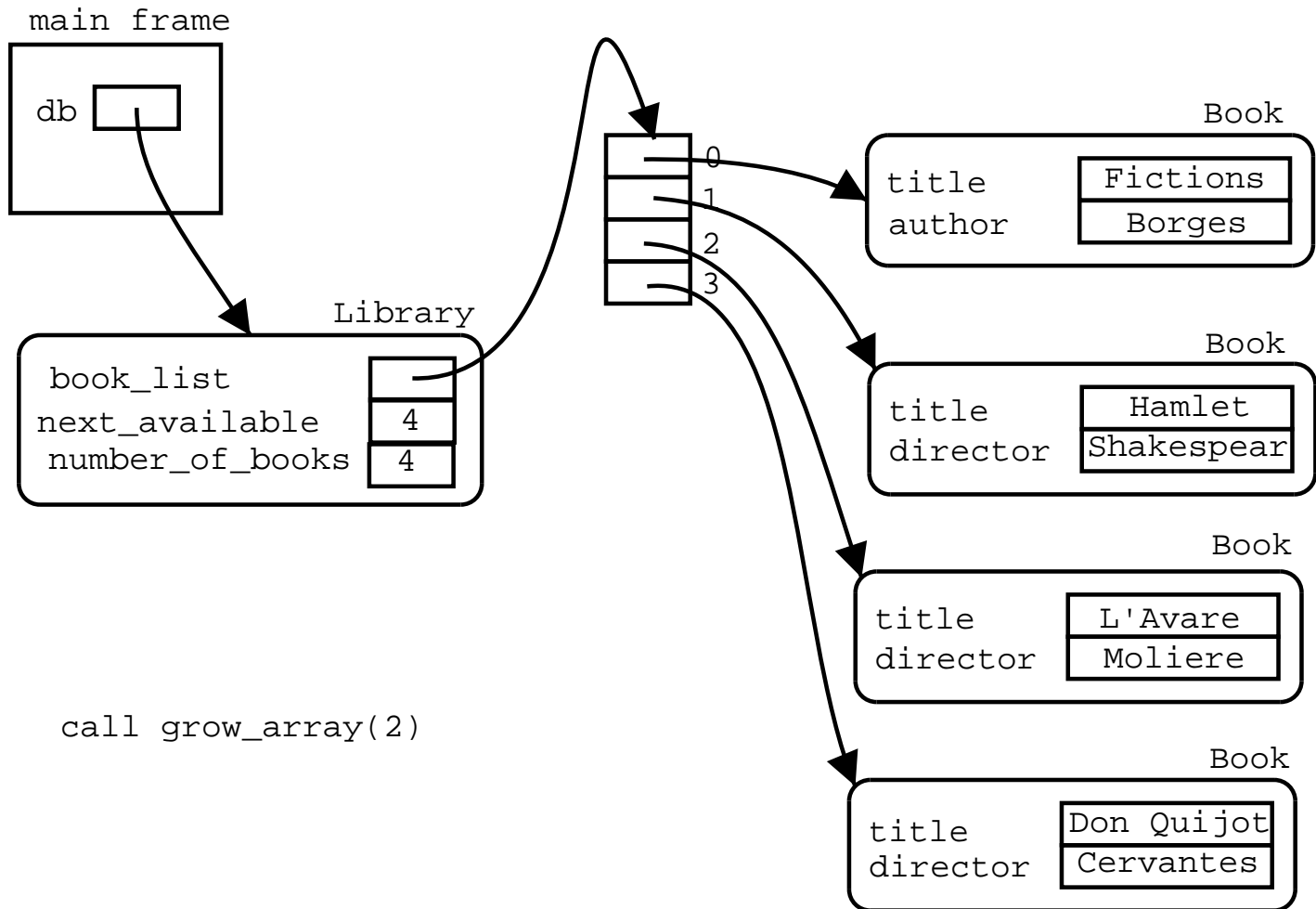
- Change algorithm for adding a book m :
 1. Find first available cell
 2. If an available cell is found:
 - (a) Store m in that cell
 3. Otherwise:
 - (a) Grow the array (copying contents of the old to the new)
 - (b) Find the first available cell in the new array (guaranteed to exist.)
 - (c) Store m in that cell

Growing arrays

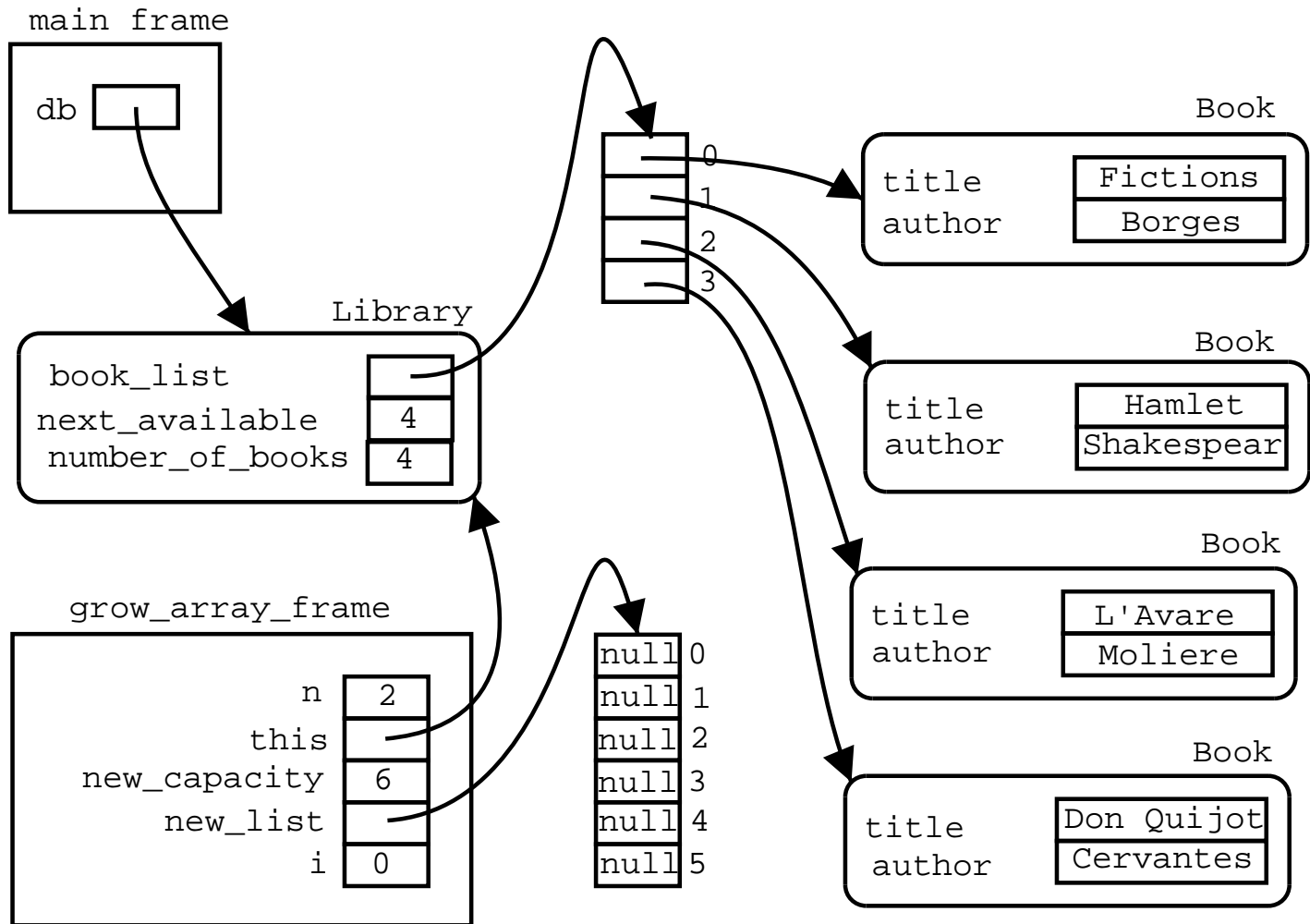
```
// In class Library
private void grow_array(int n)
{
    int new_capacity = book_list.length + n;
    Book[] new_list = new Book[new_capacity];
    int i = 0;
    while (i < Book_list.length) {
        new_list[i] = book_list[i]; // shallow copy
        i++;
    }
    book_list = new_list; // Update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

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

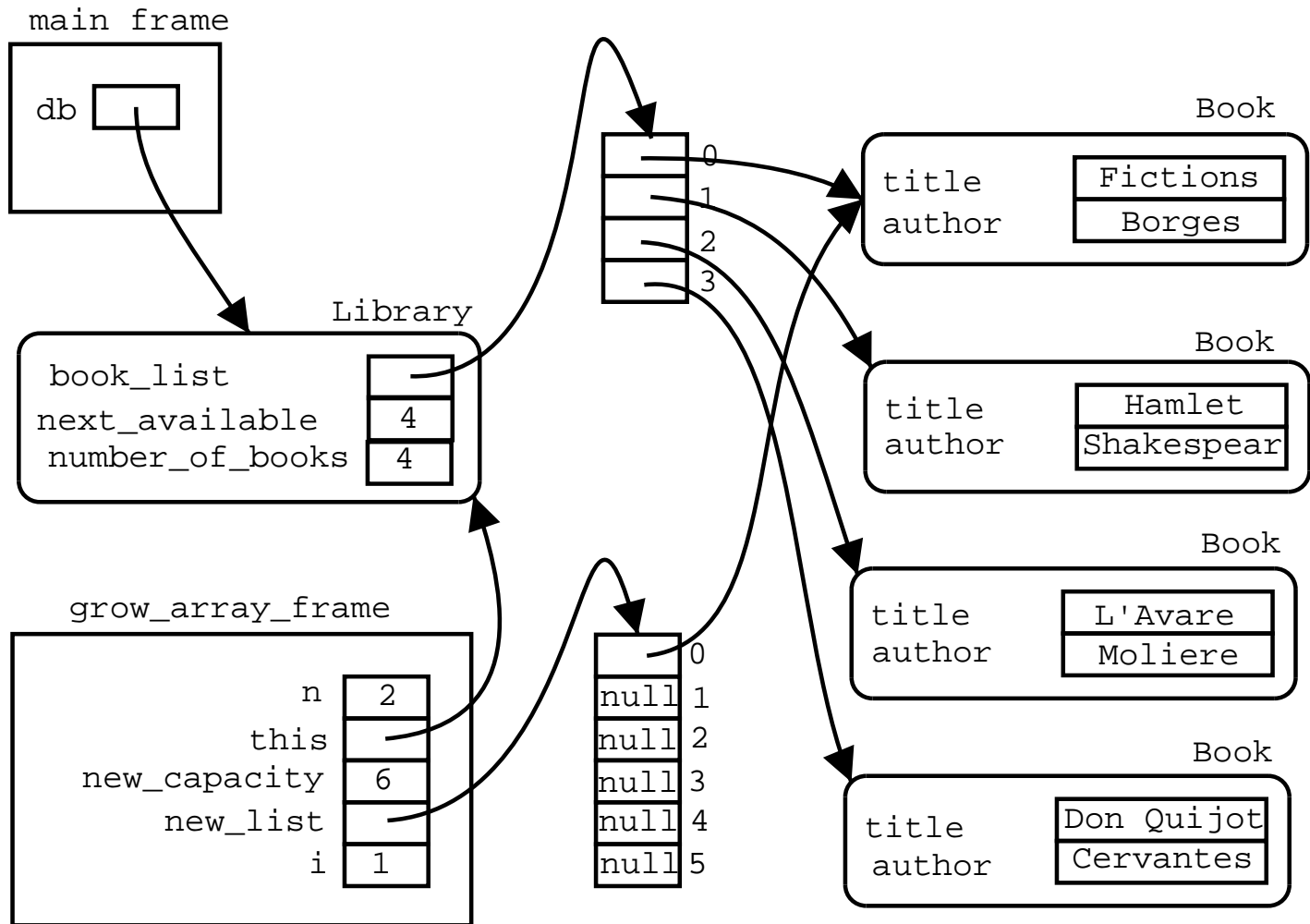
Growing arrays



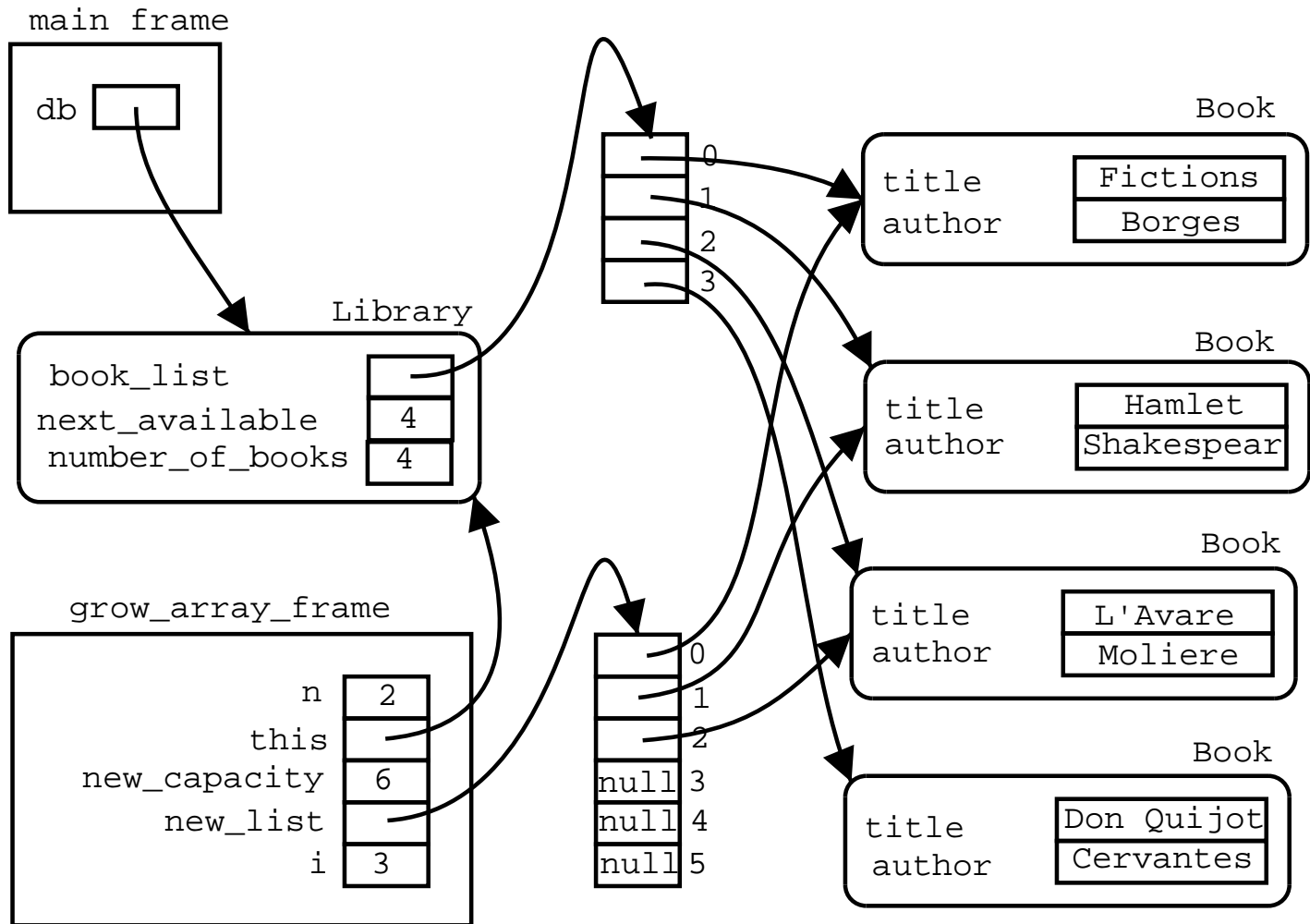
Growing arrays



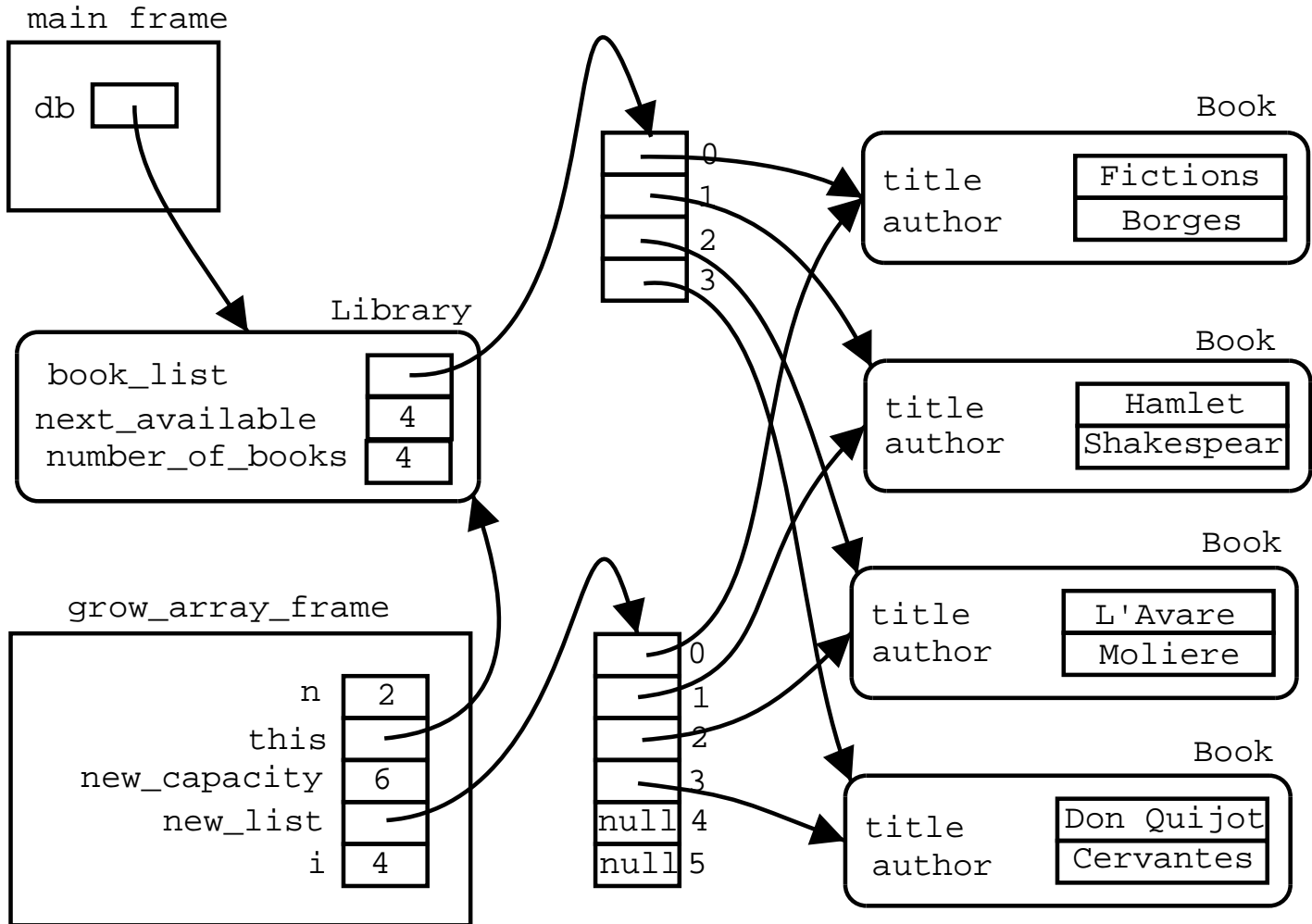
Growing arrays



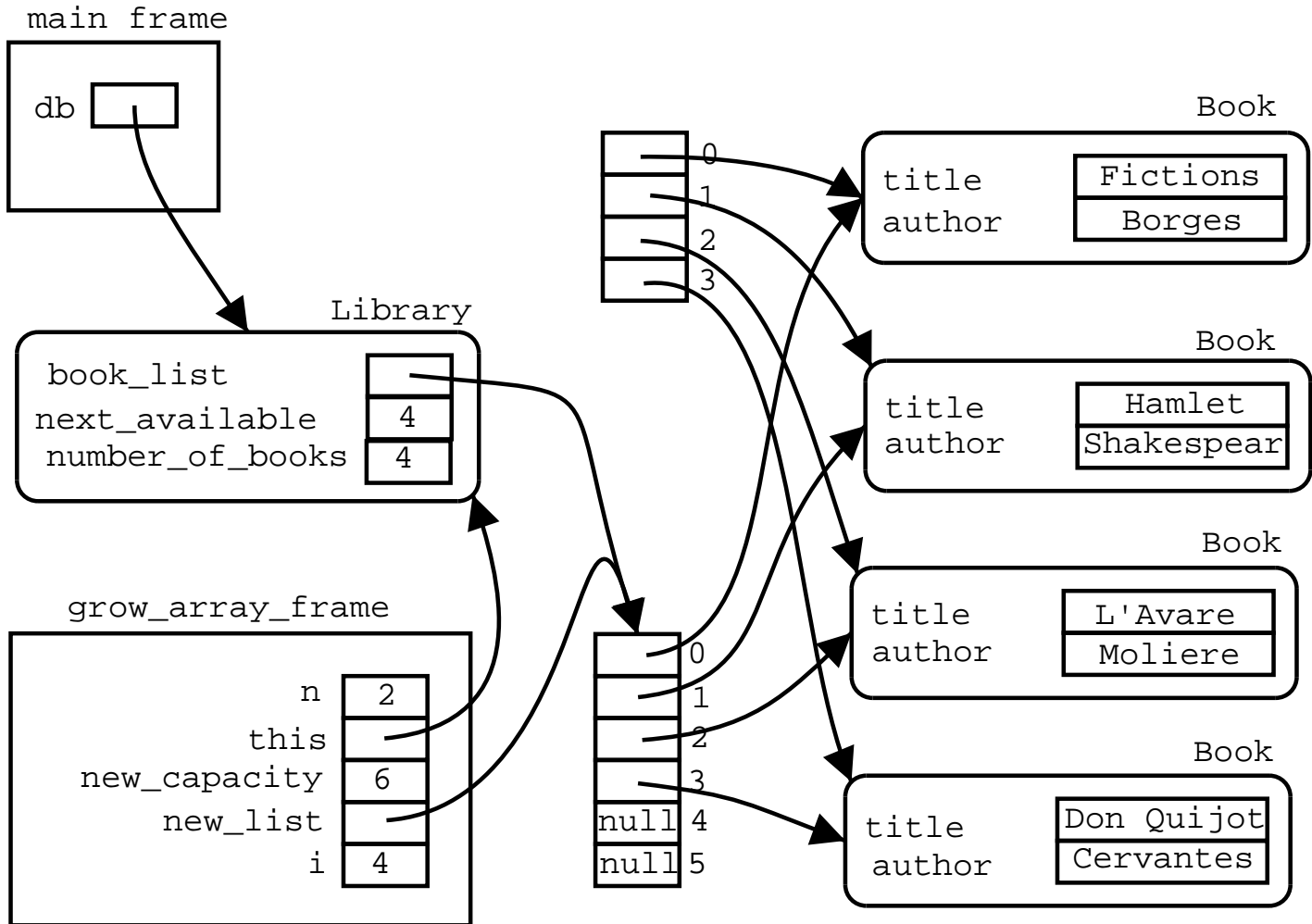
Growing arrays



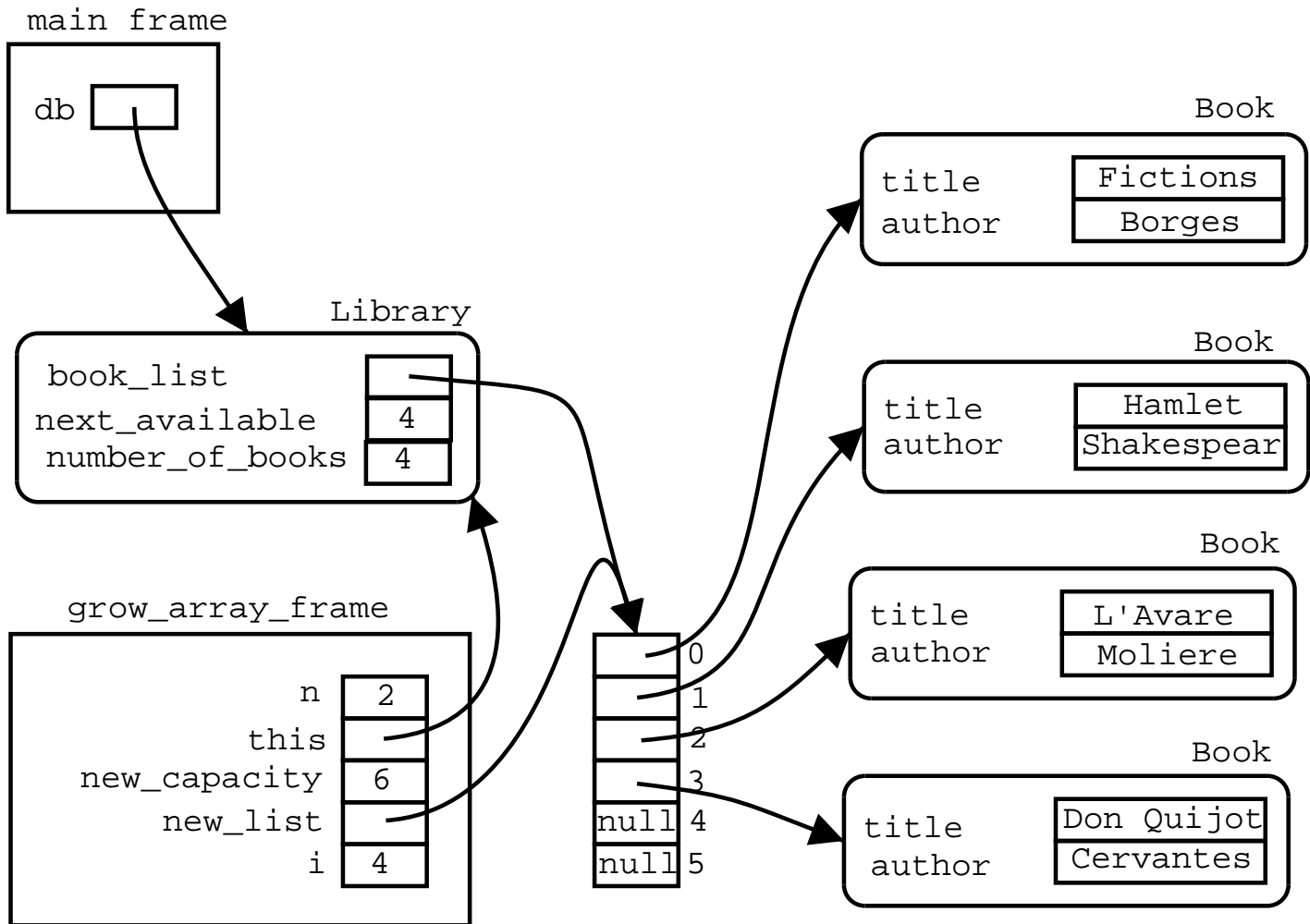
Growing arrays



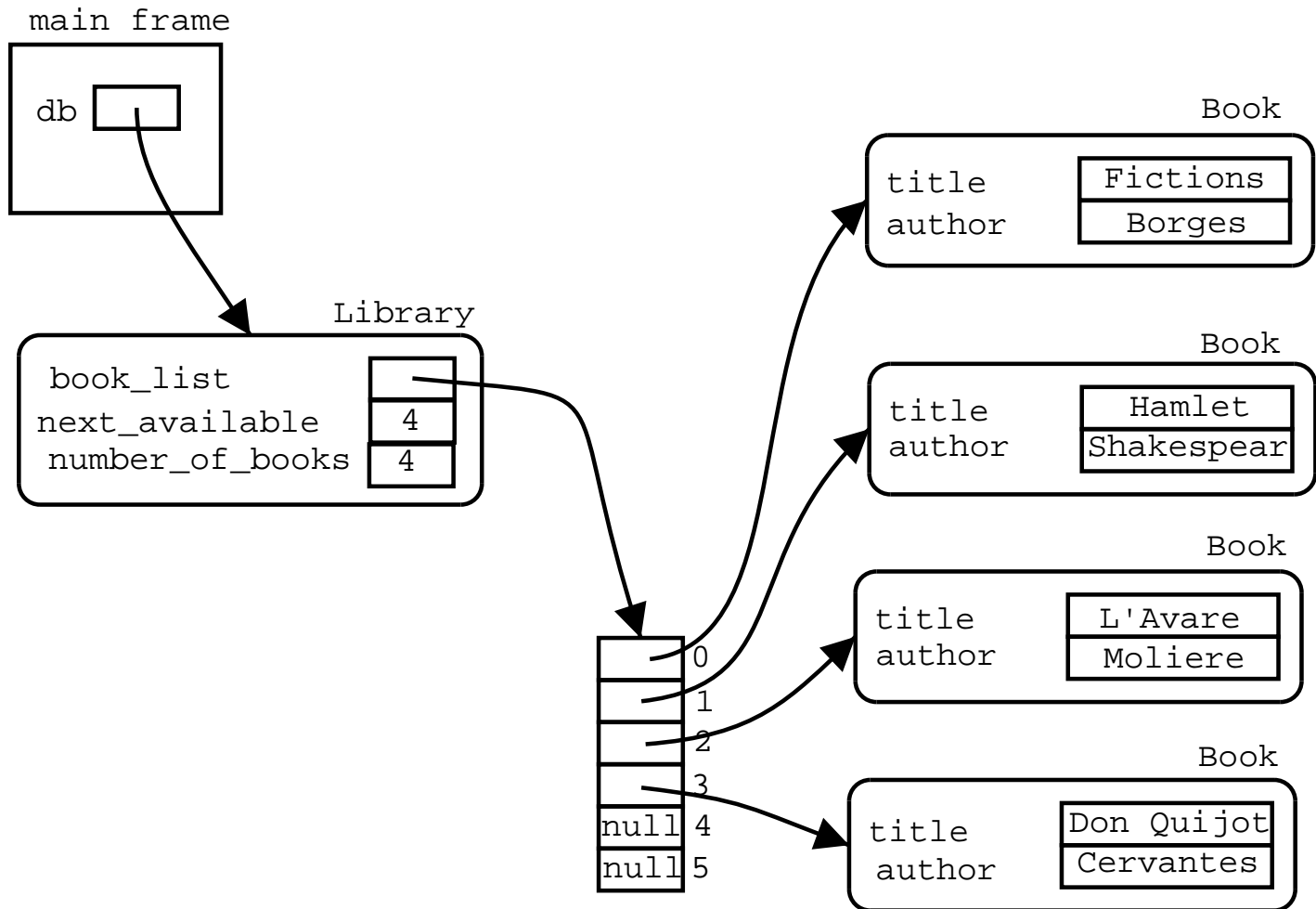
Growing arrays



Growing arrays



Growing arrays



Growing arrays

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

Growing arrays

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

Array operations

- Adding elements
- Removing/deleting elements
- Finding elements
- Increasing the size of an array

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

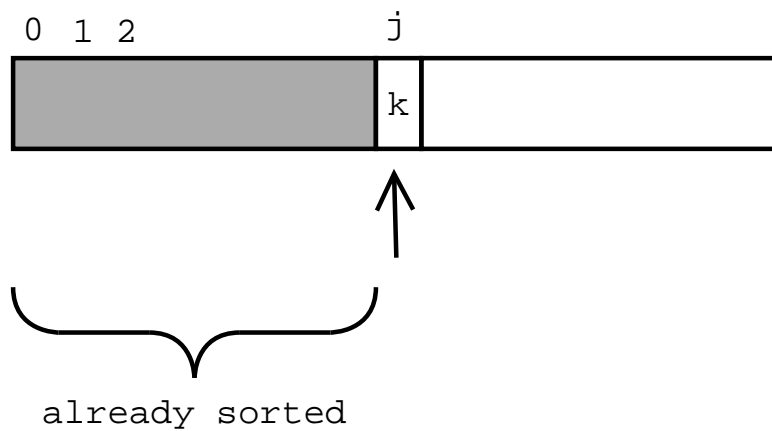
- Analysis:
 - Objects:
 - * An array of objects
 - Relationships:
 - * Each object *has a* key (and maybe other attributes.)
 - * For example, if the objects are of class Student, the key can be the name, to sort by name, or the id, to sort by id.
 - * Each pair of keys can be compared: there is a (total)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

Sorting algorithms

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

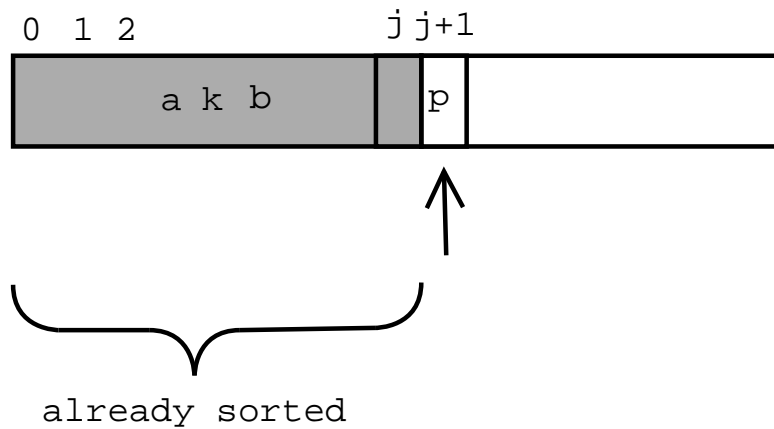
Insertion sort

- 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

- 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

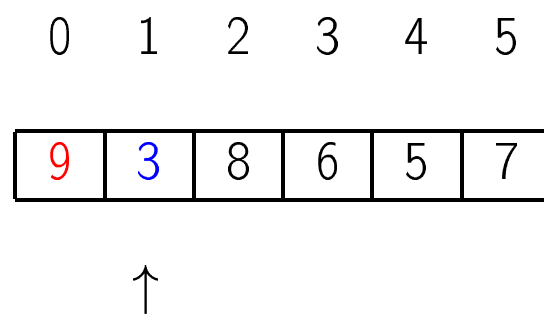
- Example:

0 1 2 3 4 5

9	3	8	6	5	7
---	---	---	---	---	---

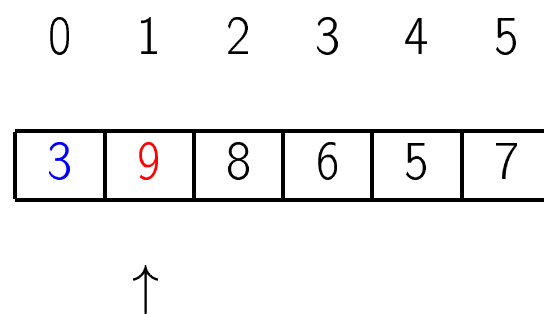
Insertion sort

- Example:



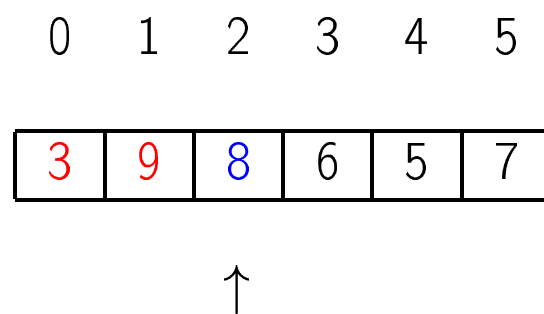
Insertion sort

- Example:



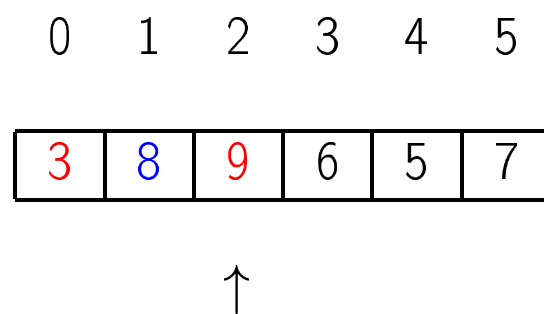
Insertion sort

- Example:



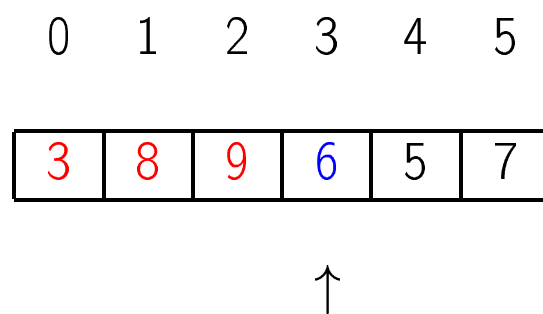
Insertion sort

- Example:



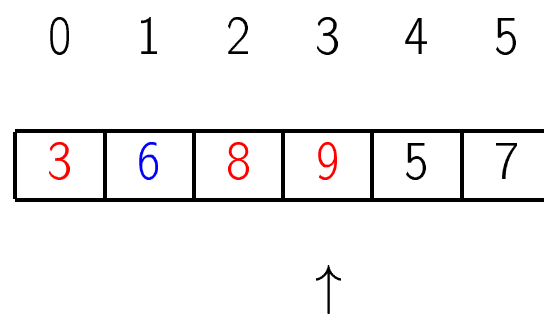
Insertion sort

- Example:



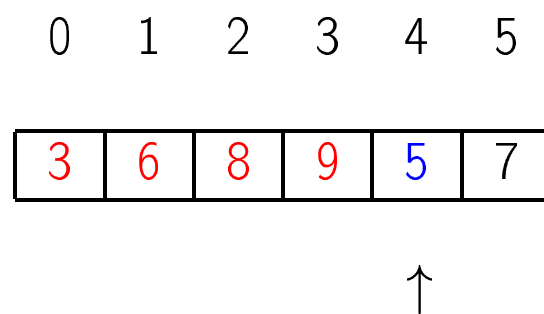
Insertion sort

- Example:



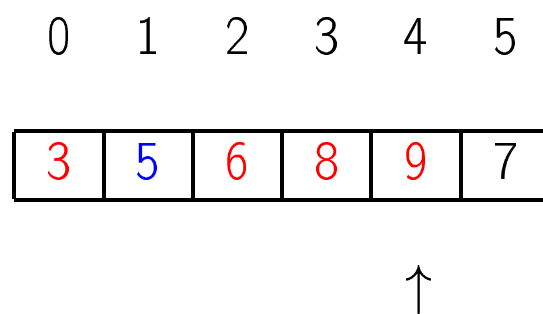
Insertion sort

- Example:



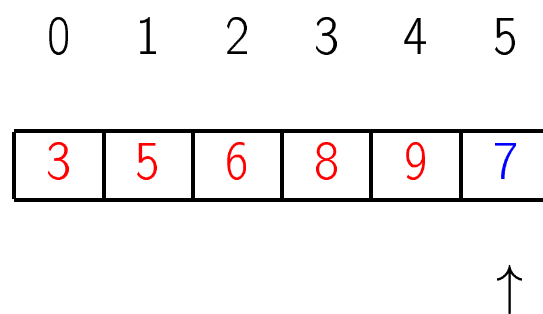
Insertion sort

- Example:



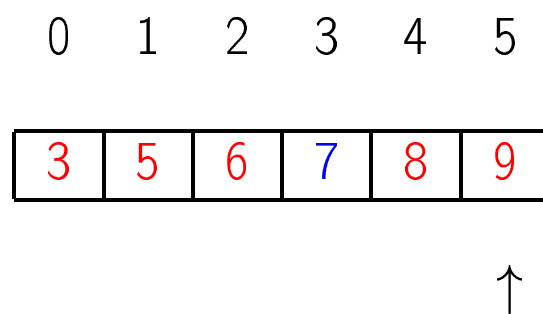
Insertion sort

- Example:



Insertion sort

- Example:



Insertion sort

- Example:

0 1 2 3 4 5

3	5	6	7	8	9
---	---	---	---	---	---

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 $a[\text{length of } a-1]$ into $a[0..\text{length of } a-2]$

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 $i \geq 0$ and $a[i] > key$ do

i. Set $a[i+1]$ to $a[i]$

ii. Decrement i by 1

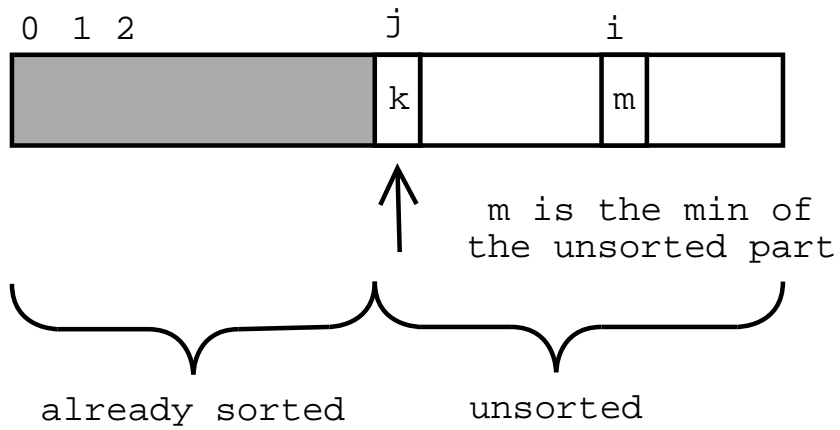
(d) Set $a[i+1]$ to key

Insertion sort

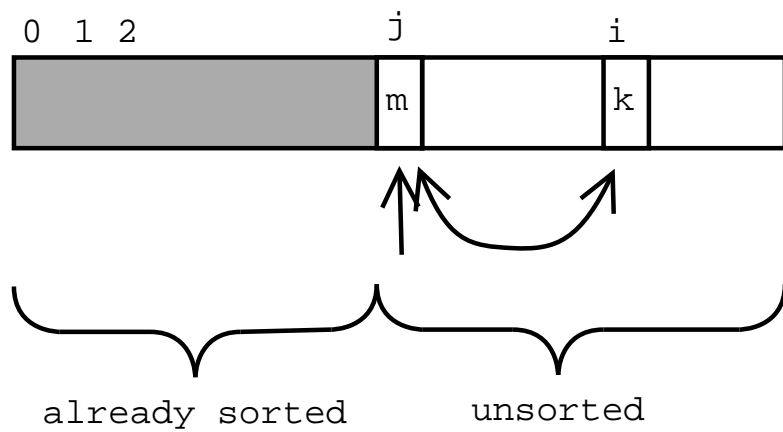
- Implementation

```
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;
    }
}
```

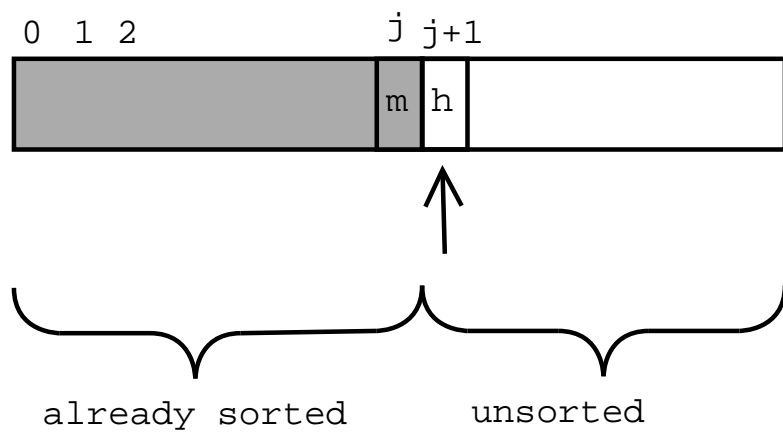
Selection sort



Selection sort



Selection sort



Selection sort

- Example:

0 1 2 3 4 5

9	3	8	6	5	7
---	---	---	---	---	---

Selection sort

- Example:

0	1	2	3	4	5
9	3	8	6	5	7

↑

Selection sort

- Example:

0	1	2	3	4	5
9	3	8	6	5	7
↑	↑				

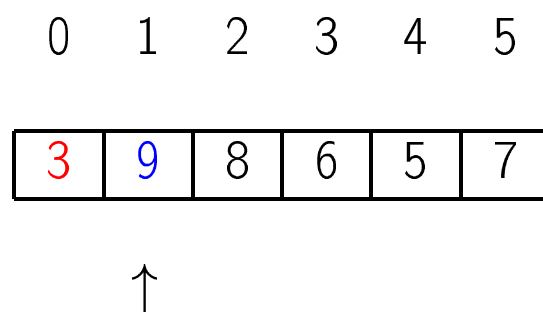
Selection sort

- Example:

0	1	2	3	4	5
3	9	8	6	5	7
↑	↑				

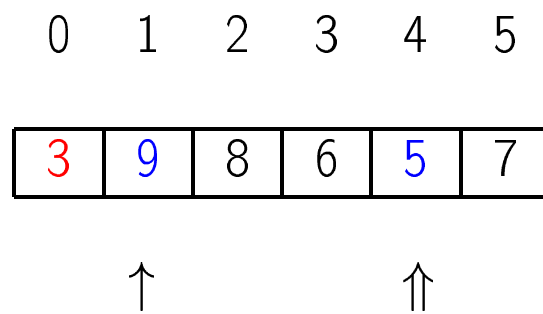
Selection sort

- Example:



Selection sort

- Example:



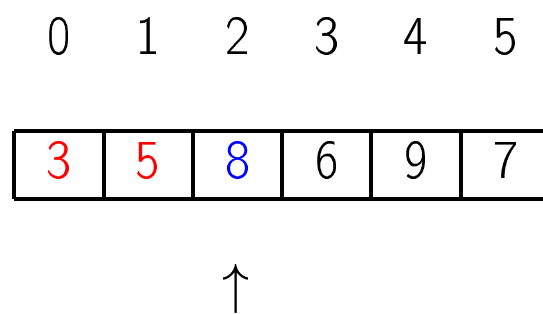
Selection sort

- Example:

0	1	2	3	4	5
3	5	8	6	9	7
	↑			↑	

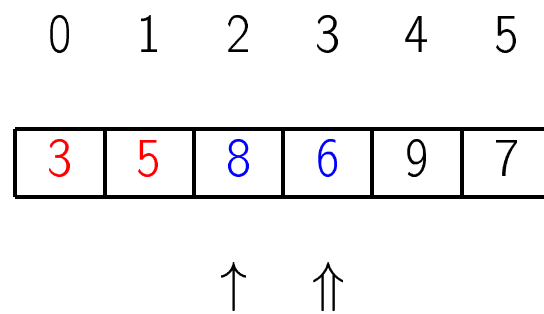
Selection sort

- Example:



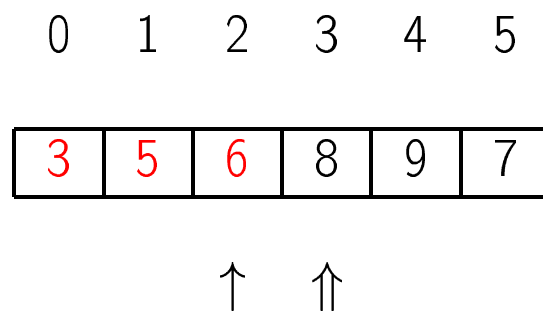
Selection sort

- Example:



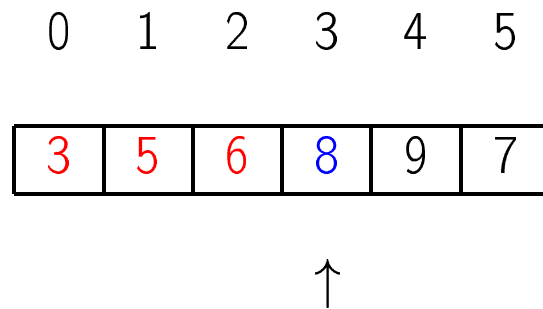
Selection sort

- Example:



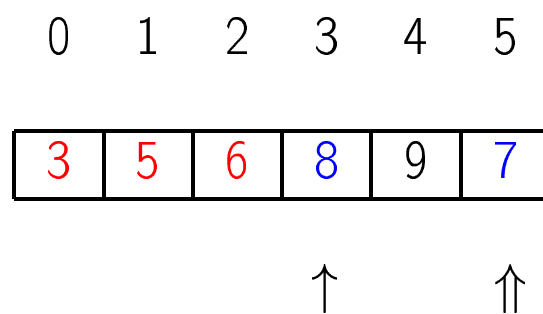
Selection sort

- Example:



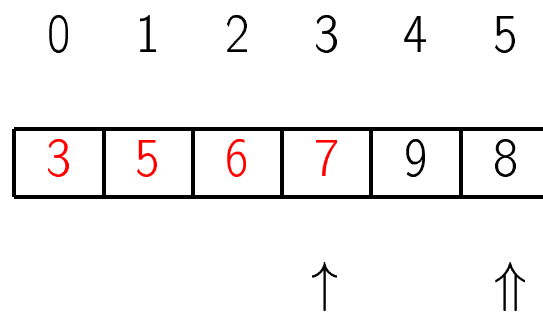
Selection sort

- Example:



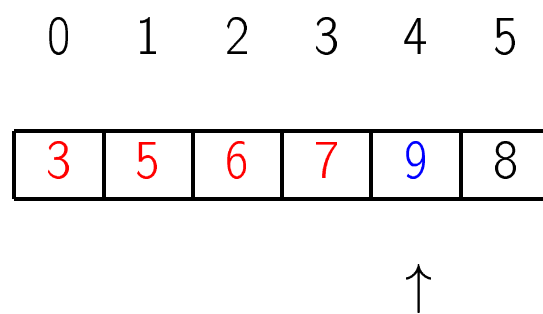
Selection sort

- Example:



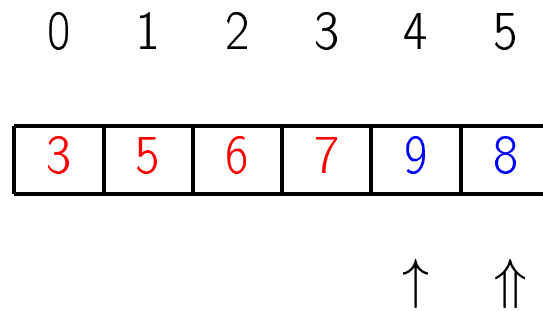
Selection sort

- Example:



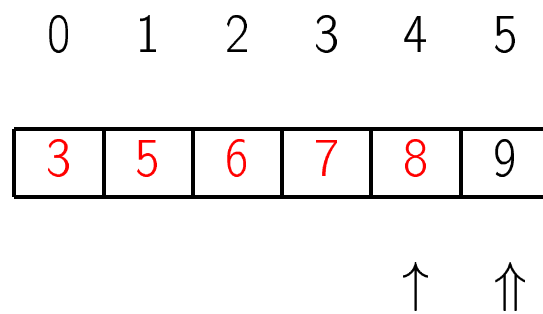
Selection sort

- Example:



Selection sort

- Example:



Selection sort

- Example:

0 1 2 3 4 5

3	5	6	7	8	9
---	---	---	---	---	---

Selection sort

- Idea:
 1. Look for the minimum m_0 in $a[1..length\ a-1]$.
 2. Swap the minimum and $a[0]$.
 3. Look for the minimum m_1 in $a[2..length\ a-1]$
 4. Swap m_1 with $a[1]$
 5. Look for the minimum m_2 in $a[3..length\ a-1]$
 6. Swap m_2 with $a[2]$
 7. Look for the minimum m_3 in $a[4..length\ a-1]$
 8. Swap m_3 with $a[3]$
 9. ...

Selection sort

- Algorithm

1. For each j from 0 to $\text{length } a - 2$ do

- (a) Let min_index to be the index of the minimum in $a[j+1..\text{length } a-1]$

- (b) Swap $a[\text{min_index}]$ and $a[j]$

- Algorithm refined

1. For each j from 0 to $\text{length } a - 2$ do

- (a) Let minimum be $a[j]$

- (b) Set min_index to j

- (c) For each i from $j+1$ to the $\text{length } a - 1$ do

- i. If $a[i] < \text{minimum}$ then

- A. Set minimum to $a[i]$

- B. Set min_index to i

- (d) Swap $a[\text{min_index}]$ 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;
    }
}
```

Sorting arrays of Objects

Choose a key, which can be compared.

```
class Book {
    private String title, author;
    //...
    public String get_title() { return title; }
    public String get_author() { return author; }
    //...
}
```

Sorting arrays of Objects

- Comparing strings: Lexicographical order
- The `compareTo` method from the `String` class
- `s1.compareTo(s2)` returns a negative integer if `s1` is lexicographically before `s2`, 0 if they are equal, and a positive integer if `s1` is lexicographically after `s2`.

```
String s1 = "aacb", s2 = "aafa";  
int n = s1.compareTo(s2); // n = -3;  
String s3 = "aacbgg";  
int m = s3.compareTo(s2); // n = -3  
int k = s3.compareTo(s1); // n = 2
```

Sorting arrays of Objects

```
void insertion_sort(Book[] a)
{
    int i, j;
    String key;
    Book focus;
    for (j = 1; j < a.length; j++) {
        focus = a[j];
        key = focus.get_title();
        i = j - 1;
        while (i >= 0
            && key.compareTo(a[i].get_title()) < 0 ) {
            a[i+1] = a[i]; // copy the reference
            i--;
        }
        a[i+1] = focus;
    }
}
```

Sorting arrays of Objects

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

The end