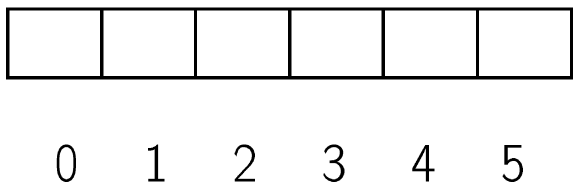
Arrays

- An *array* is an indexed sequence of variables of the same type. By indexed we mean that the variables are consecutive in memory and each of them has an index, with 0 being the first, 1 the second, and so on.



- Each variable in the array is called a *position*, a *cell* or a *slot*, and as any variable, it can contain a value.

Arrays

- An array is an ordered/indexed sequence of elements of the same type.
- Array declaration

```
type[] variable;
```

- Array creation:

```
variable = new type[integer-expression];
```

- Array reading access:

```
variable[integer-expression]
```

- Array writing access:

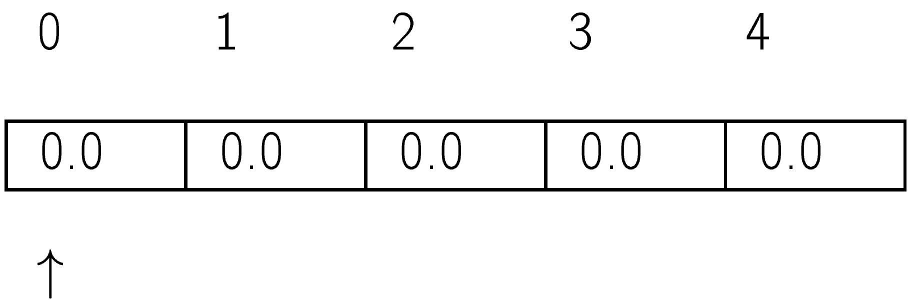
```
variable[integer-expression] = expression;
```

Example 1

- Filling an array

```
static void fill(double[] a)
{
  int index;
  index = 0;
  while (index < a.length)
  {
    a[index] = 100.0;
    index++;
  }
}
```

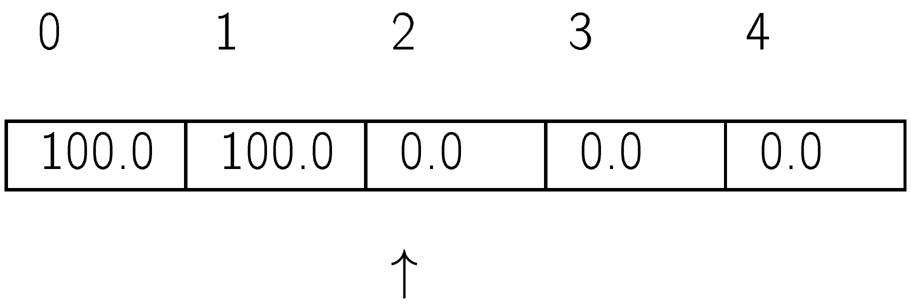
Example 1



Example 1

0	1	2	3	4
100.0	0.0	0.0	0.0	0.0
	↑			

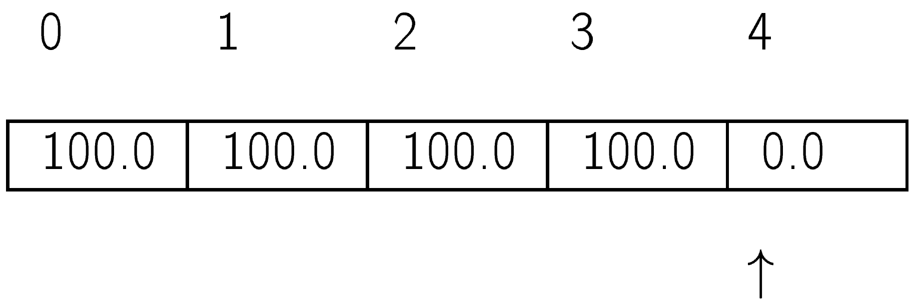
Example 1



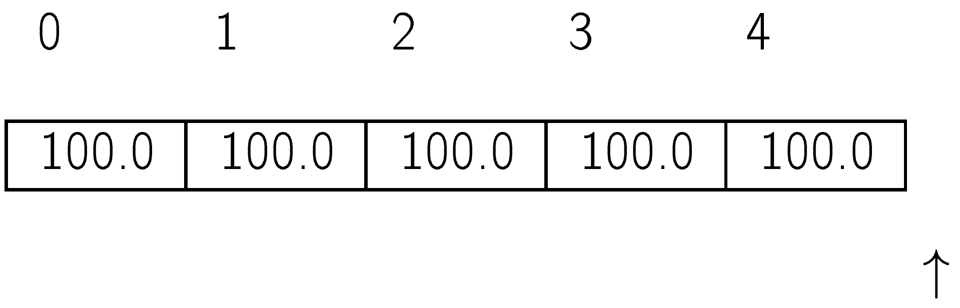
Example 1

0	1	2	3	4
100.0	100.0	100.0	0.0	0.0
			↑	

Example 1



Example 1

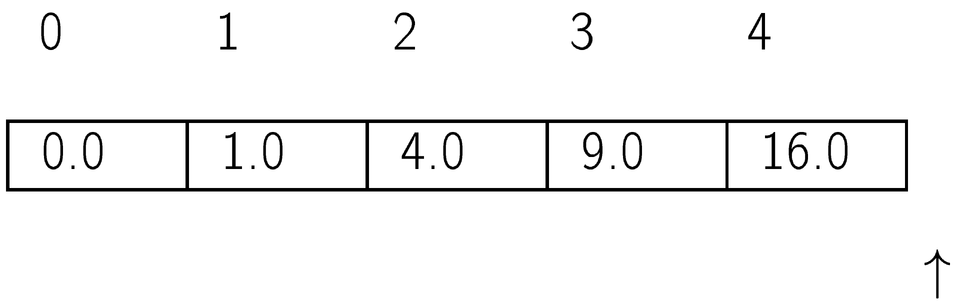


Example 2

- Filling an array

```
static void fillSquares(double[] a)
{
  int index;
  index = 0;
  while (index < a.length)
  {
    a[index] = index * index;
    index++;
  }
}
```

Example 1

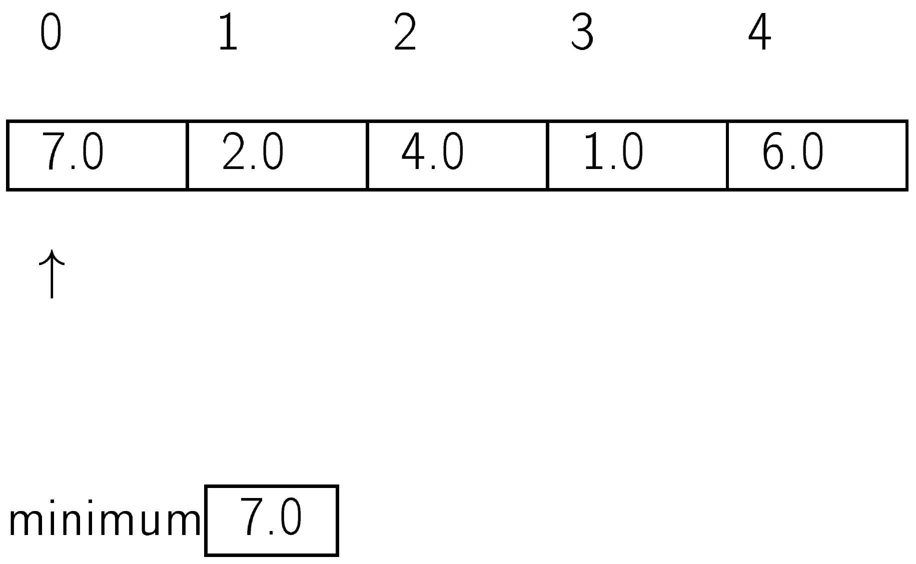


Example 3

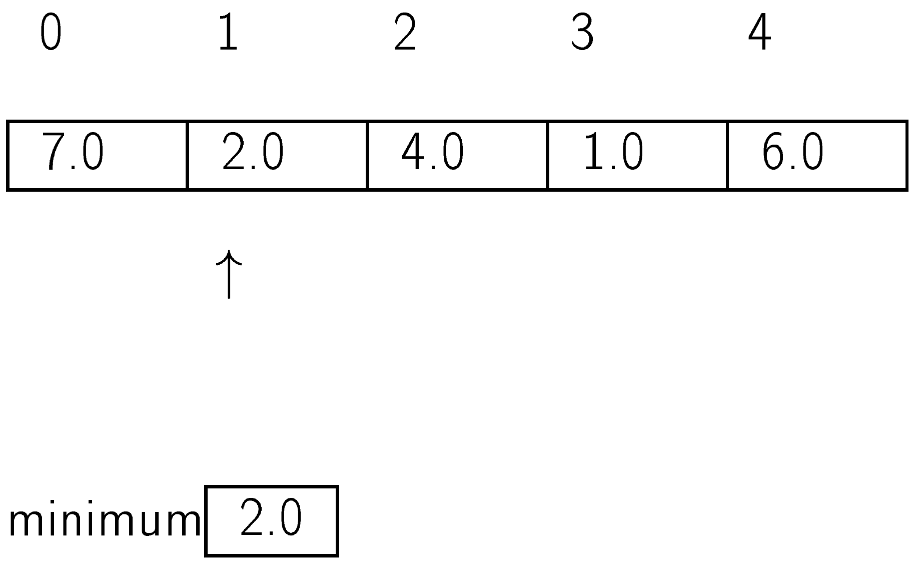
- Finding the minimum number in an array

```
static double find_min(double[] a)
{
  int index;
  double minimum;
  index = 0;
  minimum = a[0];
  while (index < a.length)
  {
    if (a[index] < minimum)
    {
      minimum = a[index];
    }
    index++;
  }
  return minimum;
}
```

Example 1



Example 1

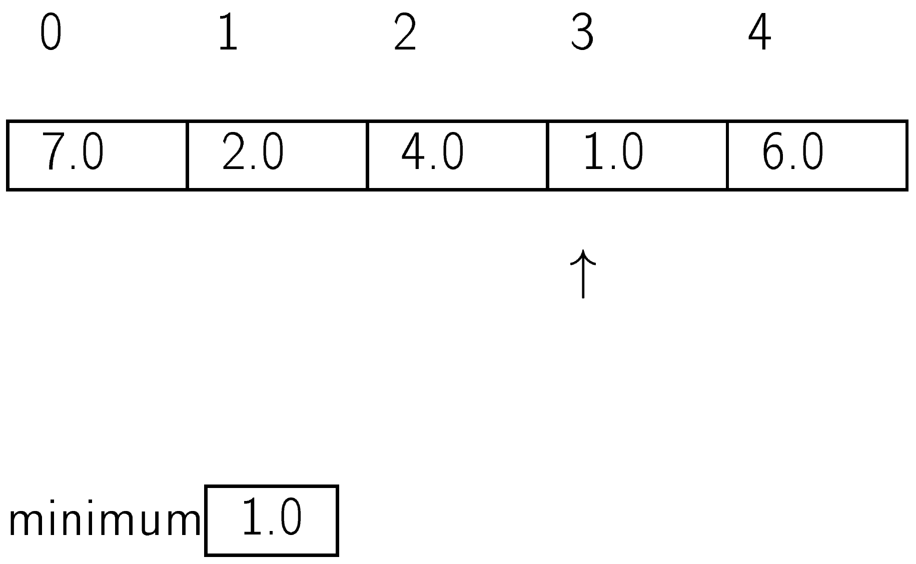


Example 1

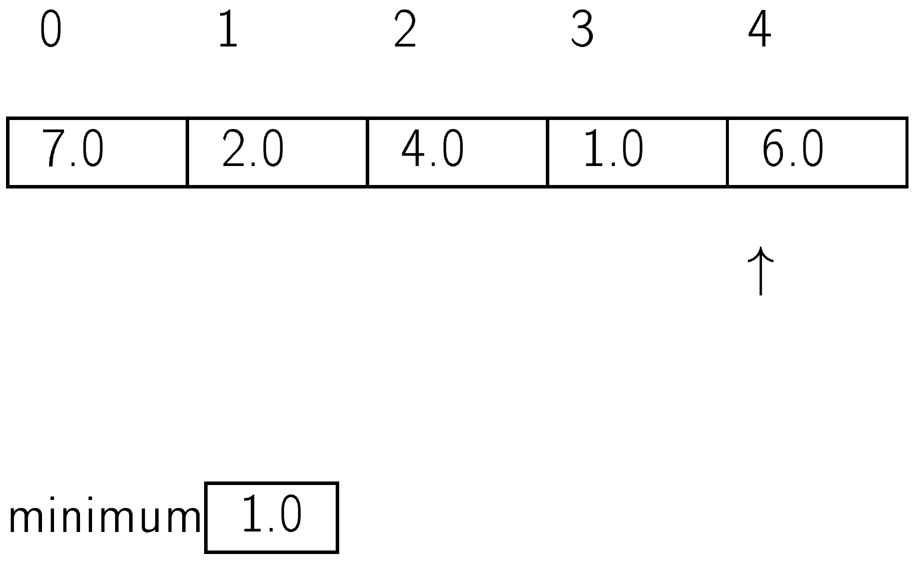
0	1	2	3	4
7.0	2.0	4.0	1.0	6.0
		↑		

minimum $\boxed{2.0}$

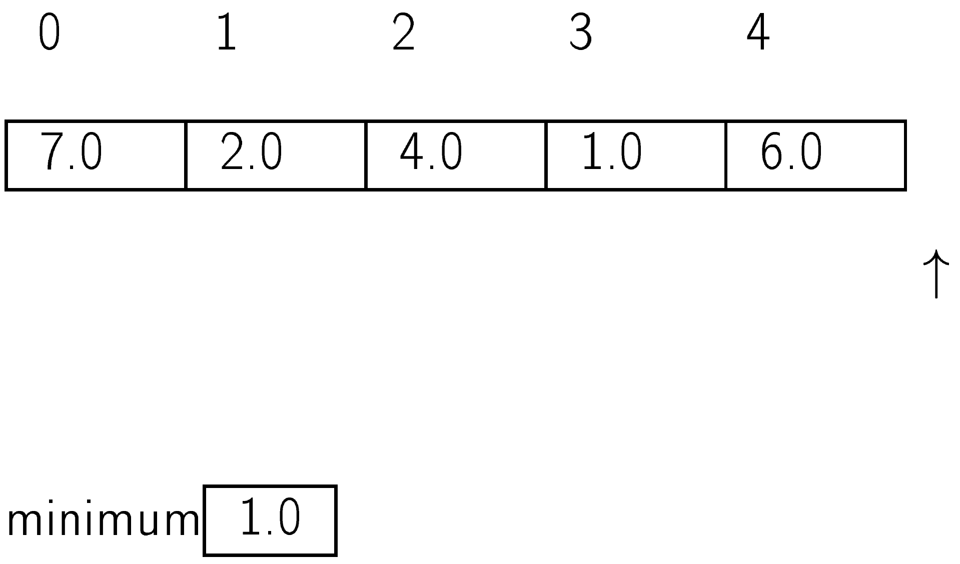
Example 1



Example 1



Example 1



Example 4

- Returning the index where the minimum is located

```
static int find_min(double[] a)
{
  int index, min_index;
  double minimum;
  index = 0;
  min_index = 0;
  minimum = a[0];
  while (index < a.length)
  {
    if (a[index] < minimum)
    {
      minimum = a[index];
      min_index = index;
    }
    index++;
  }
  return min_index;
}
```

Processing arrays: safety

- Since arrays are references, it is often useful to check whether they are null or not before using them, to avoid null-pointer exceptions.

- If the array has as base type a class, it is also useful to check that each slot which will be processed or accessed is not null.

- For example:

```
class A { int x; }
class B {
  static void m(A[] list)
  {
    if (list != null) {
      for (int i = 0; i < list.length; i++) {
        if (list[i] != null) {
          list[i] = 2 * i;
        }
      }
    }
  }
```

Initializing arrays

- If we have a class

```
class B
{
  int n;
  B(int x) { n = x; }
}
```

- and somewhere else we declare and create an array

```
B[] list = new B[7];
```

- Then all the slots in the array will be initialized to `null`. This is, the constructor for B will not be called. If we want an object created in each slot, we have to do it explicitely:

```
for (int i = 0; i < list.length; i++)
{
  list[i] = new B(3);
}
```

McGill

Initializing arrays

- Arrays can be initialized with default values using the syntax:

```
type[] var = { expr1, expr2, ..., exprn };
```

Where each *expri* is of type *type*.

- For example:

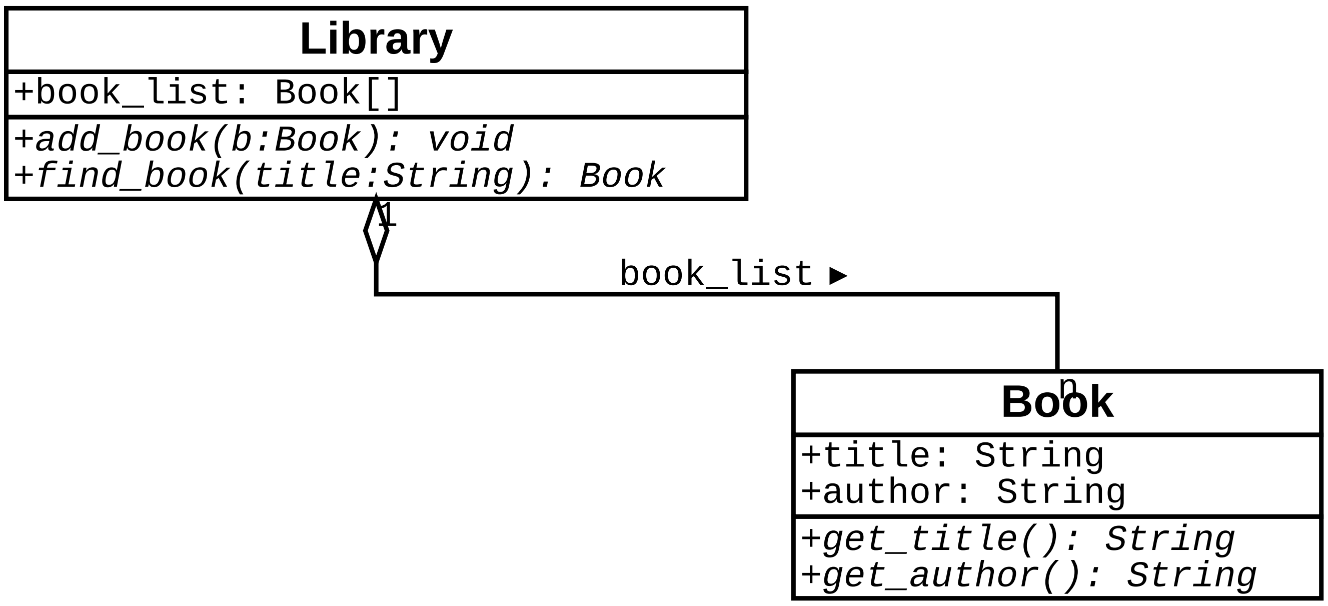
```
int[] a = { 1, 1, 2, 3, 5 };
Z[] u = { new Z(), new Z() };
```

Array applications

- Library: Book database
- Problem: Create a database of books, which supports the operations of adding a new book, and searching for a book by title.
- Analysis:
  - Identify objects and classes:
    - Individual books
    - A library: book database
  - Relationships
    - Each book *has a* title and an author
    - A book database *has a* list of books
  - Operations/Interactions/Behaviour
    - Adding books to a database
    - Searching for a book in a database

Array applications (contd.)

- Design
  - Class diagram



Array applications (contd.)

```
class Book
{
  private String title, author;
  public Book(String t, String d)
  {
    title = t;
    author = d;
  }
  public String title() { return title; }
  public String author() { return author; }

}
```

Array applications (contd.)

```
class Library
{
  private Book[] book_list;
  private int next_available;
  public int number_of_books;

  public Library(int max_capacity)
  {
    book_list = new Book[max_capacity];
    next_available = 0;
    number_of_books = 0;
  }

  // Continues below...
```

```
public void add_book(Book m)
{
  if (next_available < book_list.length)
  {
    book_list[next_available] = m;
    next_available++;
    number_of_books++;
  }
}
```

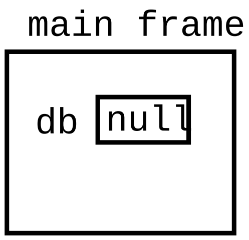
```
  public Book find_book(String title)
  {
    Book b;
    String t;
    int index = 0;
    while (index < number_of_books)
    {
      b = book_list[index];
      t = b.title();
      if (t.equals(title))
      {
        return b;
      }
      index++;
    }
    return null;
  }
} // End of Library
```

Array applications (contd.)

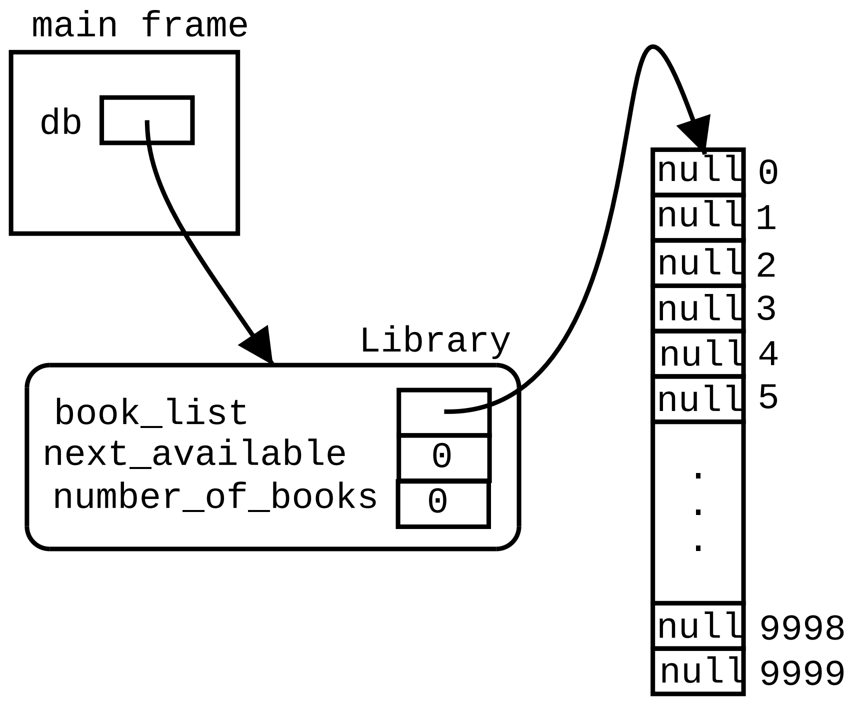
- Using the database

```
public class Test
{
  public static void main(String[] args)
  {
    Library db = new Library(10000);
    Book m;
    m = new Book(‘‘Fictions’’,‘‘Borges’’);
    db.add_book(m);
    m = new Book(‘‘Hamlet’’,‘‘Shakespeare’’);
    db.add_book(m);
    m = new Book(‘‘L\’Avare’’,‘‘Moliere’’);
    db.add_book(m);
    Book k = db.find_book(‘‘Hamlet’’);
    System.out.println( k.author() );
  }
}
```

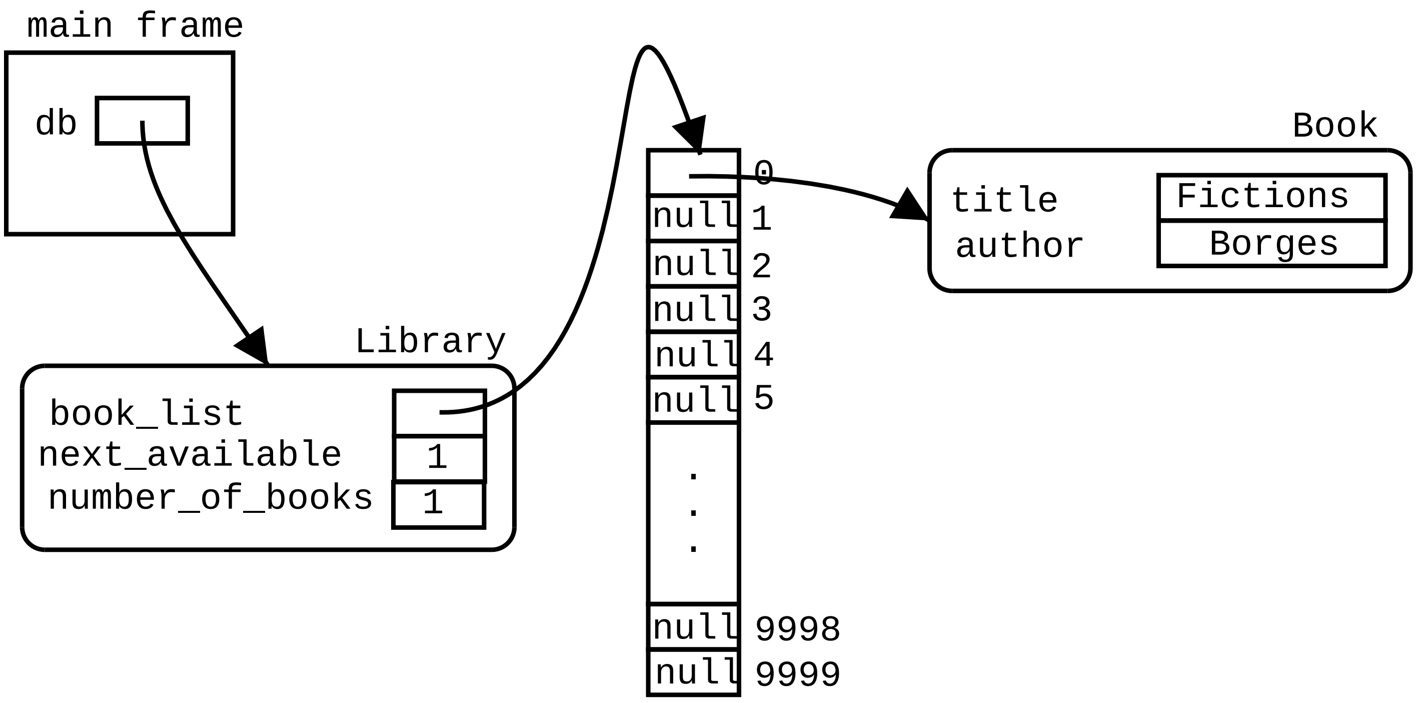
Array applications (contd.)



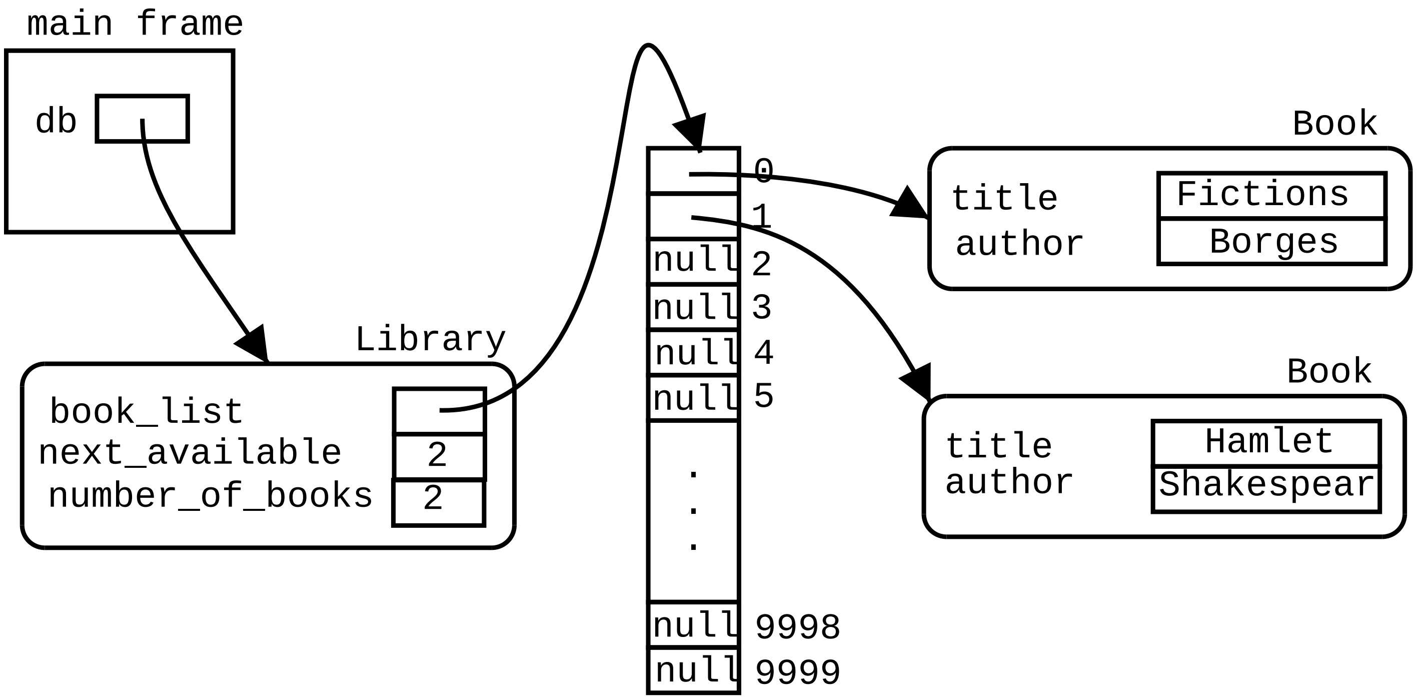
Array applications (contd.)



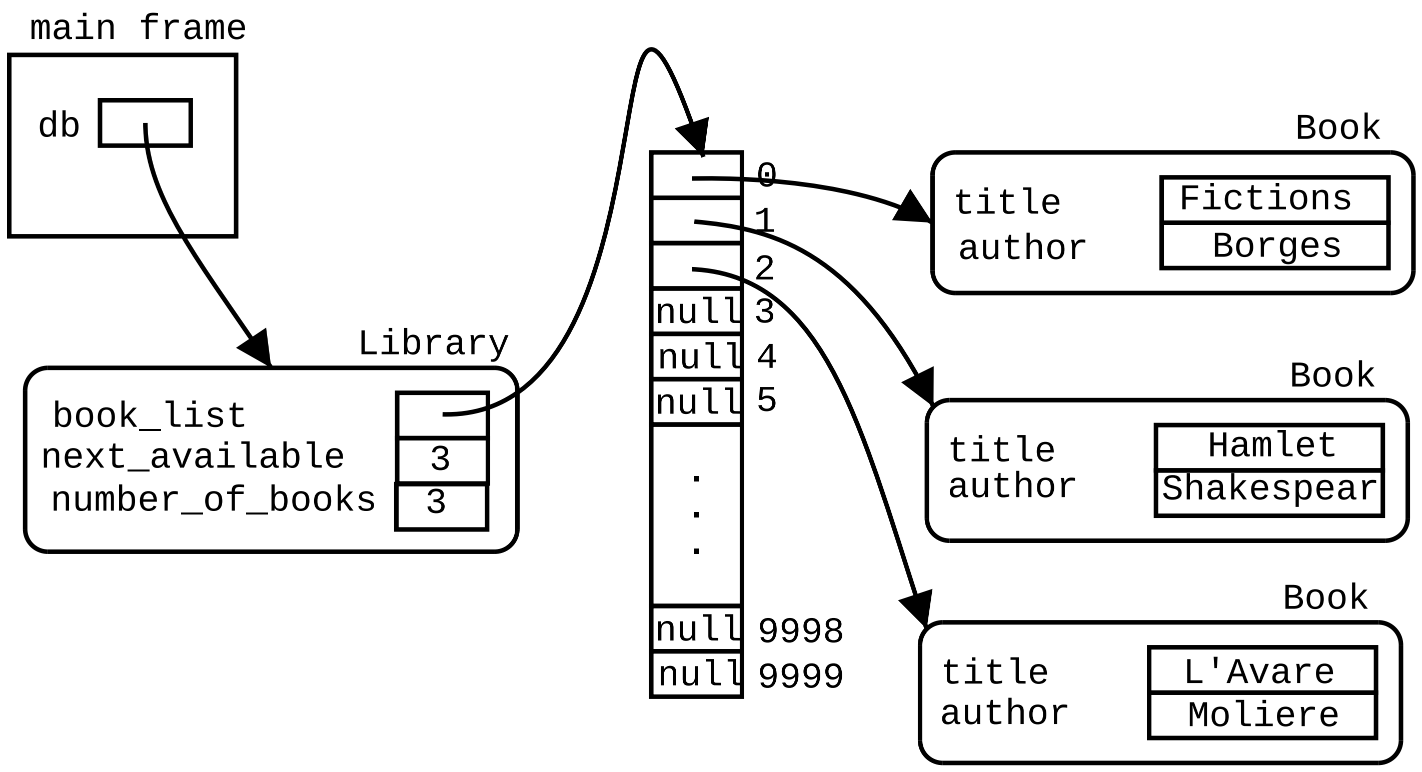
Array applications (contd.)



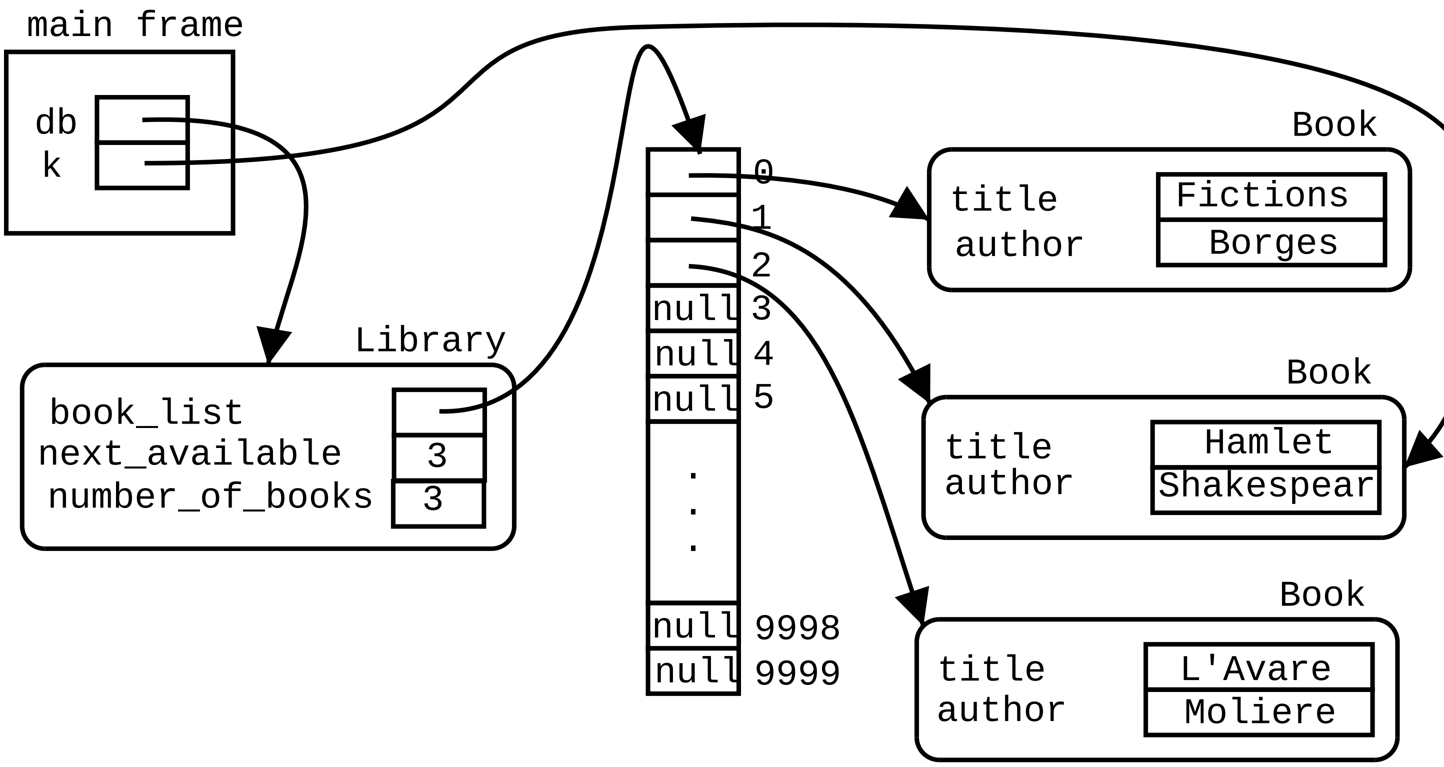
Array applications (contd.)



Array applications (contd.)



Array applications (contd.)



Array applications (contd.)

- Deleting elements from the database:
- To delete a book with title t:

1. Find the index i where the book with title t is located
2. Set book_list[i] to null

Array applications (contd.)

- Deleting elements from the database:
- To delete a book with title t:

1. Find the index i where the book with title t is located
2. If i is a legal index:
   (a) Set book_list[i] to null

Array applications (contd.)

```
public int book_index(String title)
{
  int i;
  i = 0;
  while (i < book_list.length)
  {
    Book m = book_list[i];
    if (m != null)
    {
      String s = m.title();
      if (s.equals(title))
      {
        return i;
      }
    }
    i++;
  }
  return -1;
}
```

Array applications (contd.)

```
public void delete_book(String title)
{
  int i = book_index(title);
  if (i != -1)
  {
    book_list[i] = null;
    number_of_books--;
  }
}
```

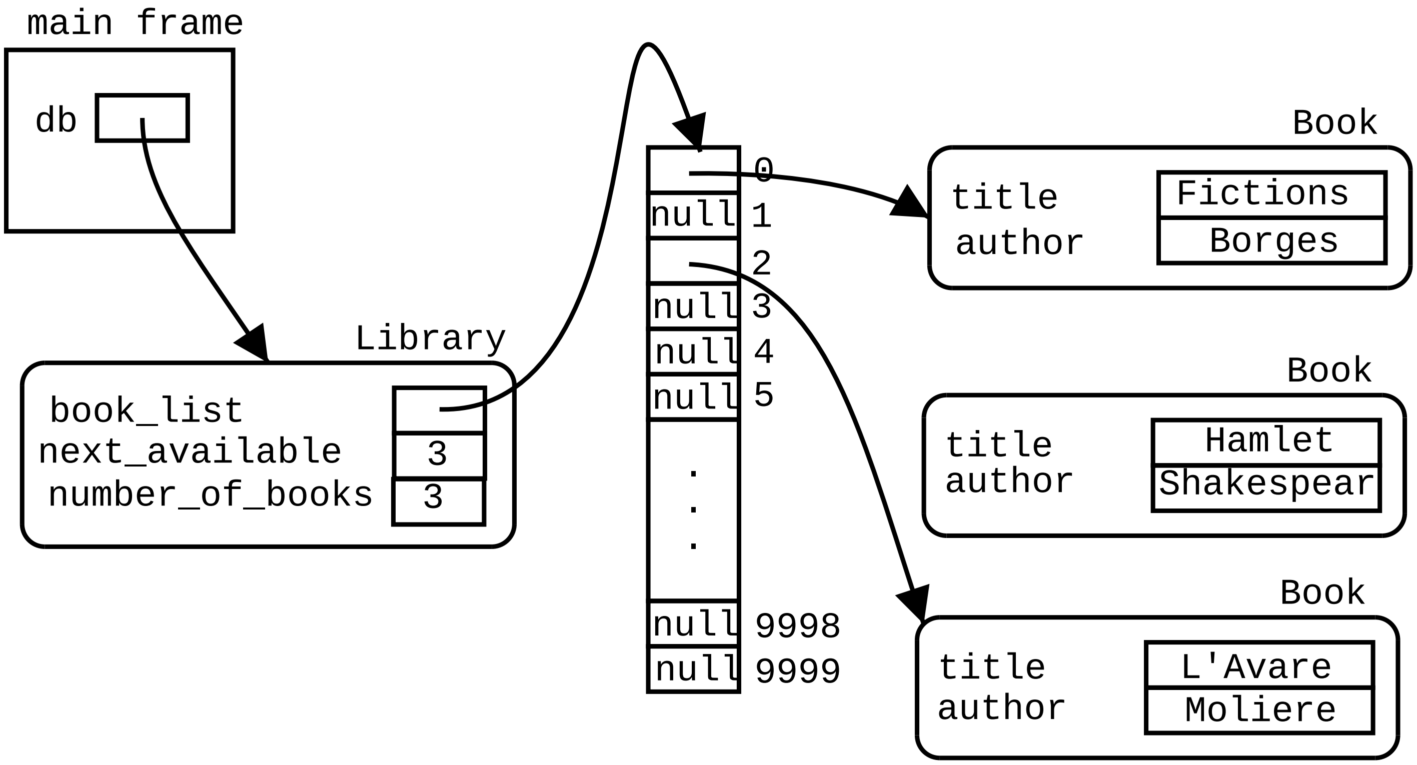
Array applications (contd.)

- But there's a problem: holes!

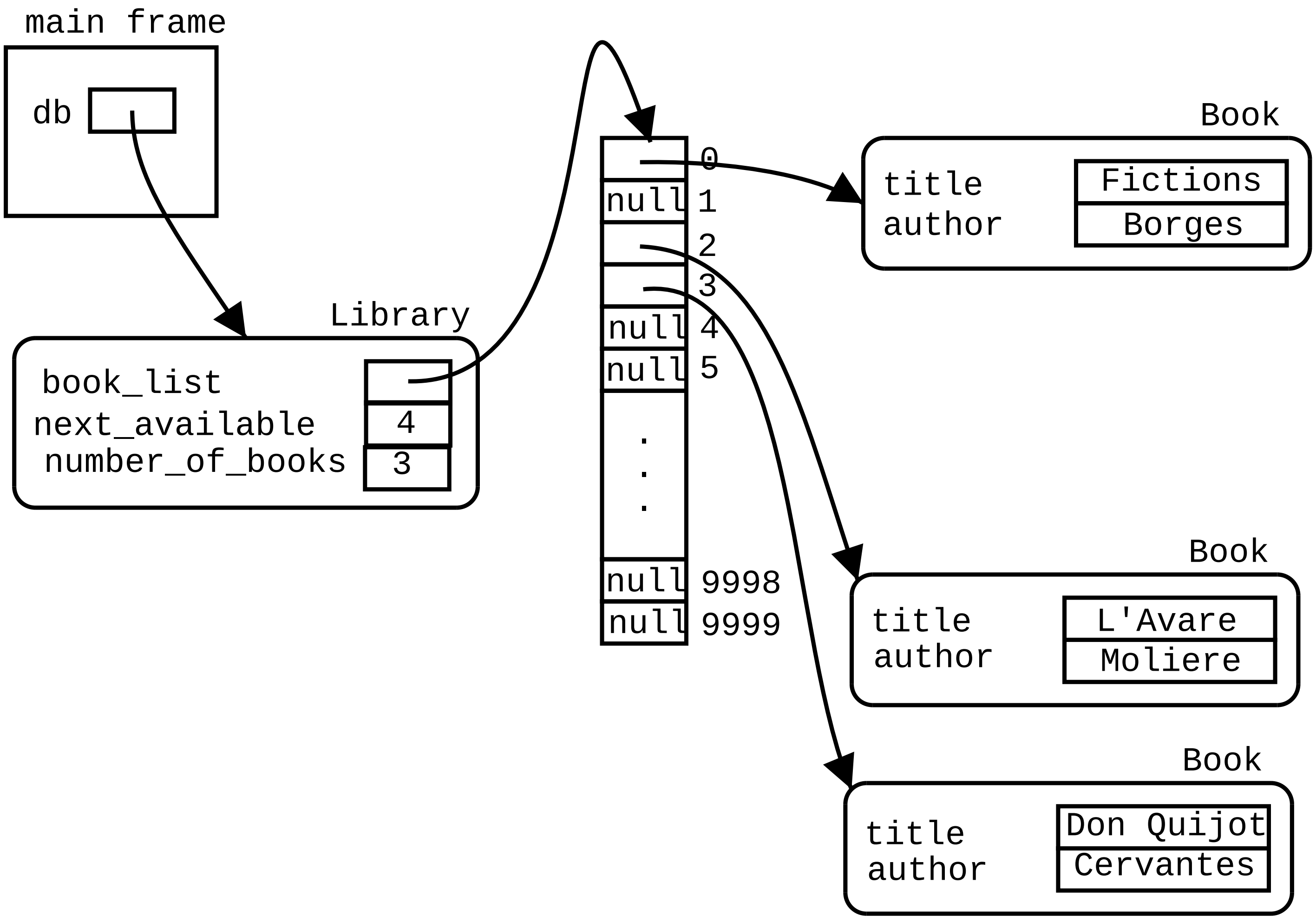
```
public class Test {
  public static void main(String[] args)
  {
    Library db = new Library(10000);
    Book m;
    m = new Book(‘‘Fictions’’,‘‘Borges’’);
    db.add_book(m);
    m = new Book(‘‘Hamlet’’,‘‘Shakespeare’’);
    db.add_book(m);
    m = new Book(‘‘L\’Avare’’,‘‘Moliere’’);
    db.add_book(m);
    db.delete_book(‘‘Hamlet’’);
    m = new Book(‘‘Don Quijote’’,‘‘Cervantes’’);
    db.add_book(m);
  }
}
```

McGill

Array applications (contd.)



Array applications (contd.)



Array applications (contd.)

- New algorithm for adding a book m:

1. Find an available slot i in book_list
2. Set book_list[i] to the book m

Array applications (contd.)

- Implementation

```
public void add_book(Book m)
{
  // Find an empty slot
  int index = 0;
  while (index < book_list.length
      && book_list[index] != null)
  {
    index++;
  }
  // Store the book
  if (index < book_list.length)
  {
    book_list[index] = m;
    number_of_books++;
  }
}
```

Array applications (contd.)

Putting it all together:

```
class Library
{
  private Book[] book_list;
  public int number_of_books;

  public Library(int max_capacity) { ... }

  public void add_book(Book m) { ... }

  public int book_index(String title) { ... }

  public void delete_book(String title) { ... }

  public Book find_book(String title) { ... }
} // End of Library
```

Array applications (contd.)

```
class Library
{
  private Book[] book_list;
  public int number_of_books;

  public Library(int max_capacity)
  {
    book_list = new Book[max_capacity];
    number_of_books = 0;
  }

  public void add_book(Book m)
  {
    // Find an empty slot
    int index = 0;
    while (index < book_list.length
        && book_list[index] != null)
    {
      index++;
    }
    // Store the book
    if (index < book_list.length)
    {
      book_list[index] = m;
      number_of_books++;
    }
  }

  public int book_index(String title)
  {
    int i;
    i = 0;
    while (i < book_list.length)
    {
      Book m = book_list[i];
      if (m != null)
      {
        String s = m.title();
        if (s.equals(title))
        {
          return i;
        }
        }
        i++;
      }
      return -1;
    }

    public void delete_book(String title)
    {
      int i = book_index(title);
      if (i != -1)
      {
        book_list[i] = null;
        number_of_books--;
      }
    }

    public Book find_book(String title)
    {
      int i = book_index(title);
      if (i != -1) return book_list[i];
      return null;
    }
  } // End of Library
```

Optimized Book database

Idea: instead of looking for an available cell each time we add a book, modify the delete method so that when we delete a book, move the last book of the list to the cell which just openned. This way, the array is not fragmented. This is, there are no holes, and all books are all grouped toghether at the beginning of the array.

```
class Library
{
  private Book[] book_list;
  public int next_available;

  public Library(int max_capacity)
  {
    book_list = new Book[max_capacity];
    next_available = 0;
  }

  // Continues below...
```

Optimized Book database

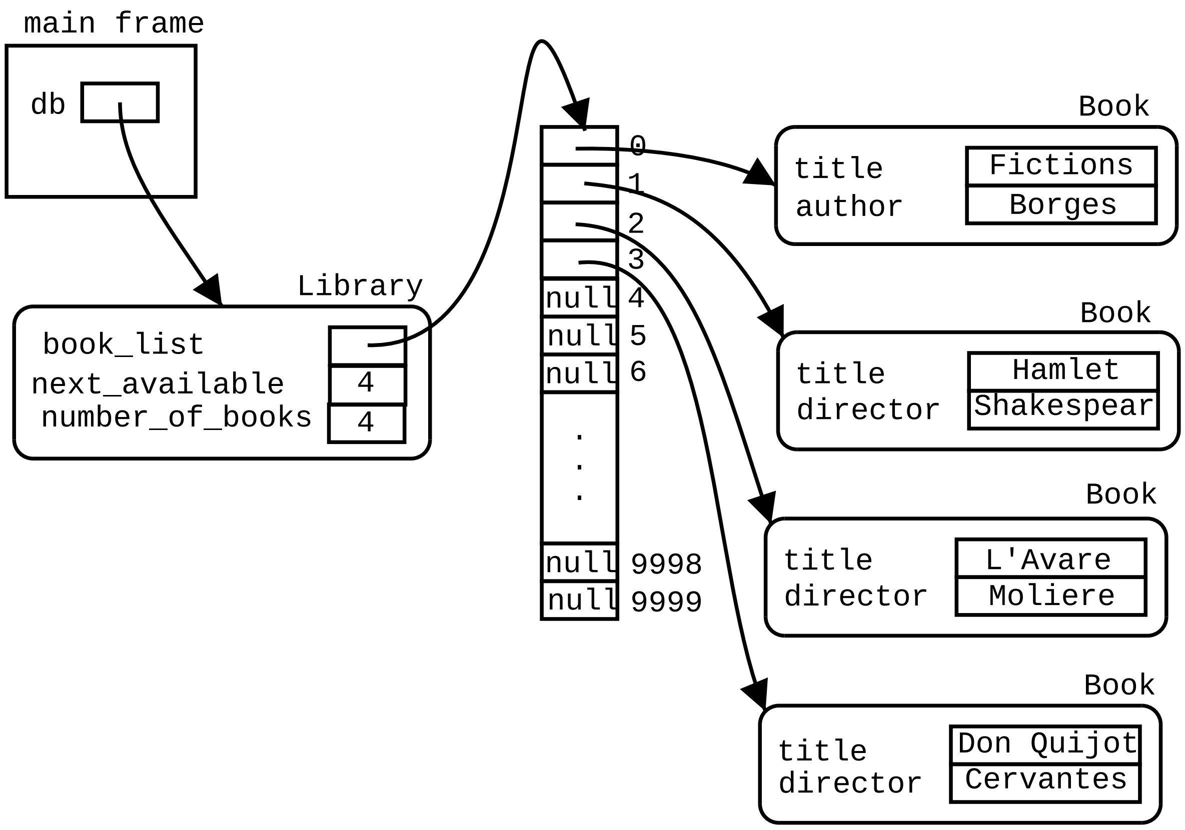
```
public void add_book(Book m)
{
  if (next_available < book_list.length)
  {
    book_list[next_available] = m;
    next_available++;
  }
}

public int book_index(String title)
{
  // The same as before
}
```

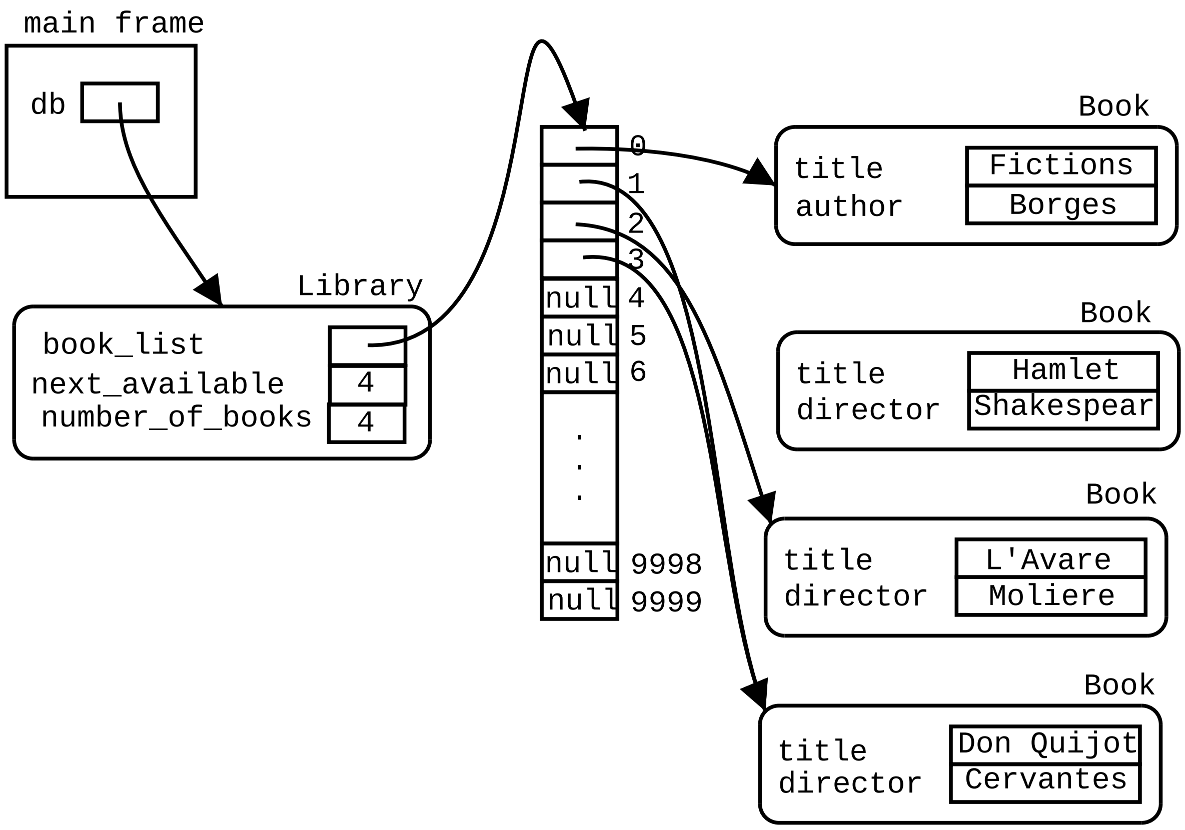
Optimized Book database

```
  public void delete_book(String title)
  {
    int i = book_index(title);
    if (i != -1)
    {
      book_list[i]=book_list[next_available-1];
      book_list[next_available - 1] = null;
      next_available--;
    }
  }
  public Book find_book(String t)
  {
    // The same as before
  }
  public int number_of_books()
  {
    return next_available;
  }
}
```

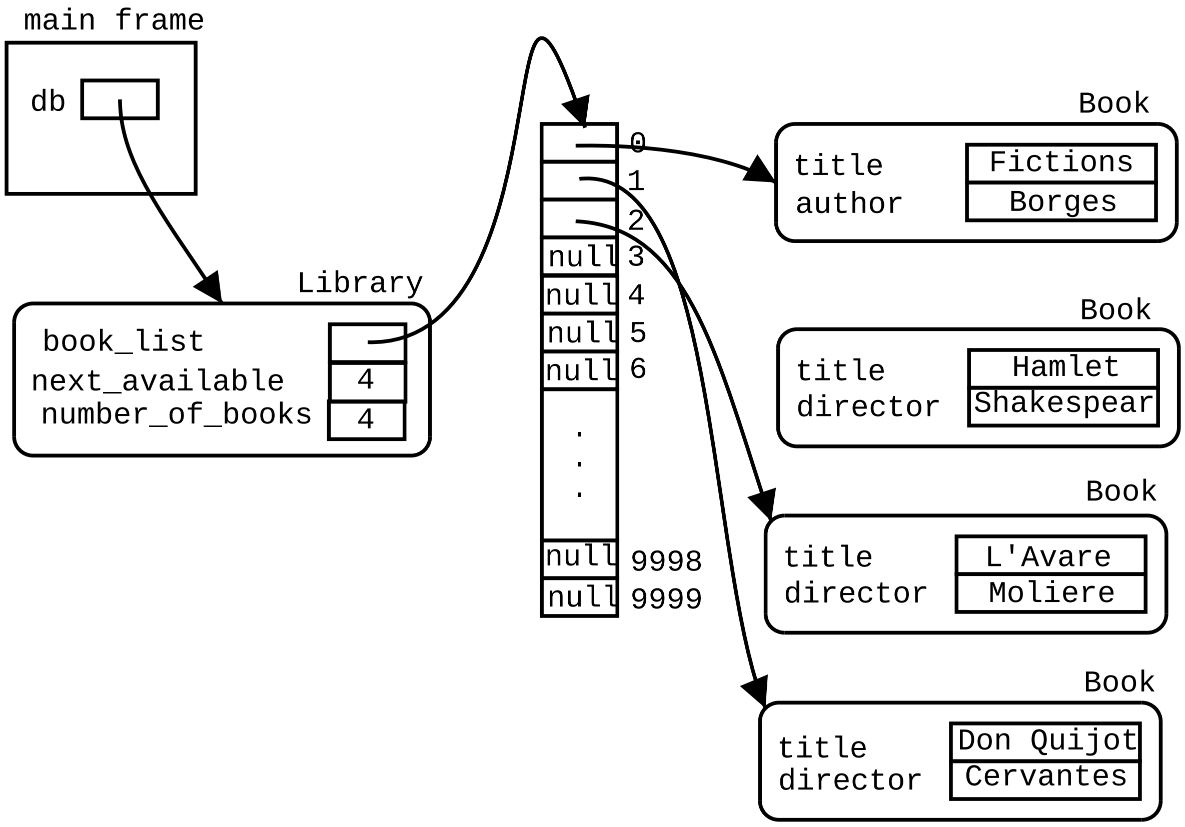
Optimized Book database



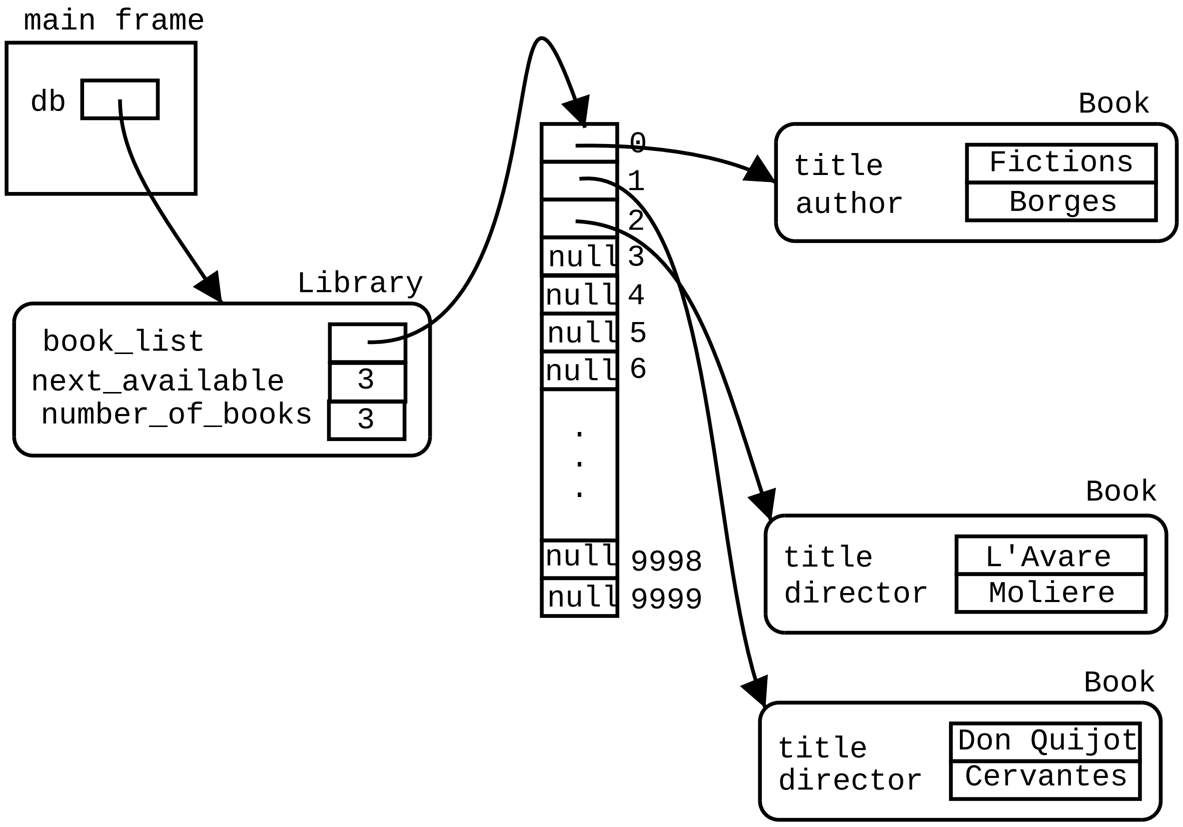
Optimized Book database



Optimized Book database



Optimized Book database



Growing arrays

- An array has a finite and fixed amount of memory.
- What do we do if we run out of space?
- 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

- How to make an array grow: (general idea)
  - Whenever the array of interest fills up, a new, create a bigger array
  - ...and copy the values of the old array into the new array.

Growing arrays

- Change algorithm for adding a book m:

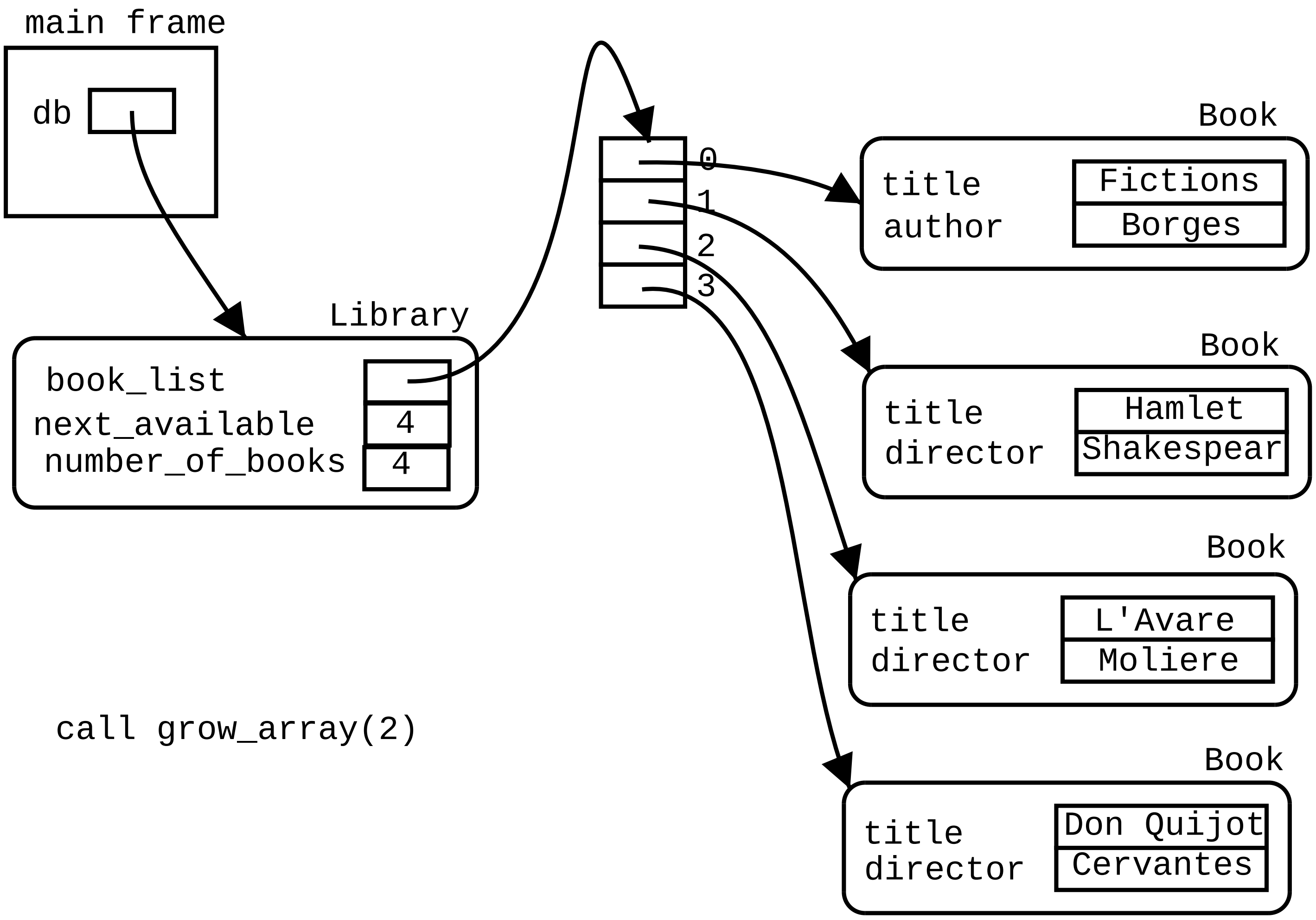
1. Find first available cell
2. If an available cell is found:
   (a) Store m in that cell
3. Otherwise: (This is the new part)
   (a) Grow the array (copying contents of the old to the new)
   (b) Find the first available cell in the new array (guarranteed to exist.)
   (c) Store m in that cell

Growing arrays

```
// In class 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];
    i++;
  }
  book_list = new_list; // Update list reference
}
```

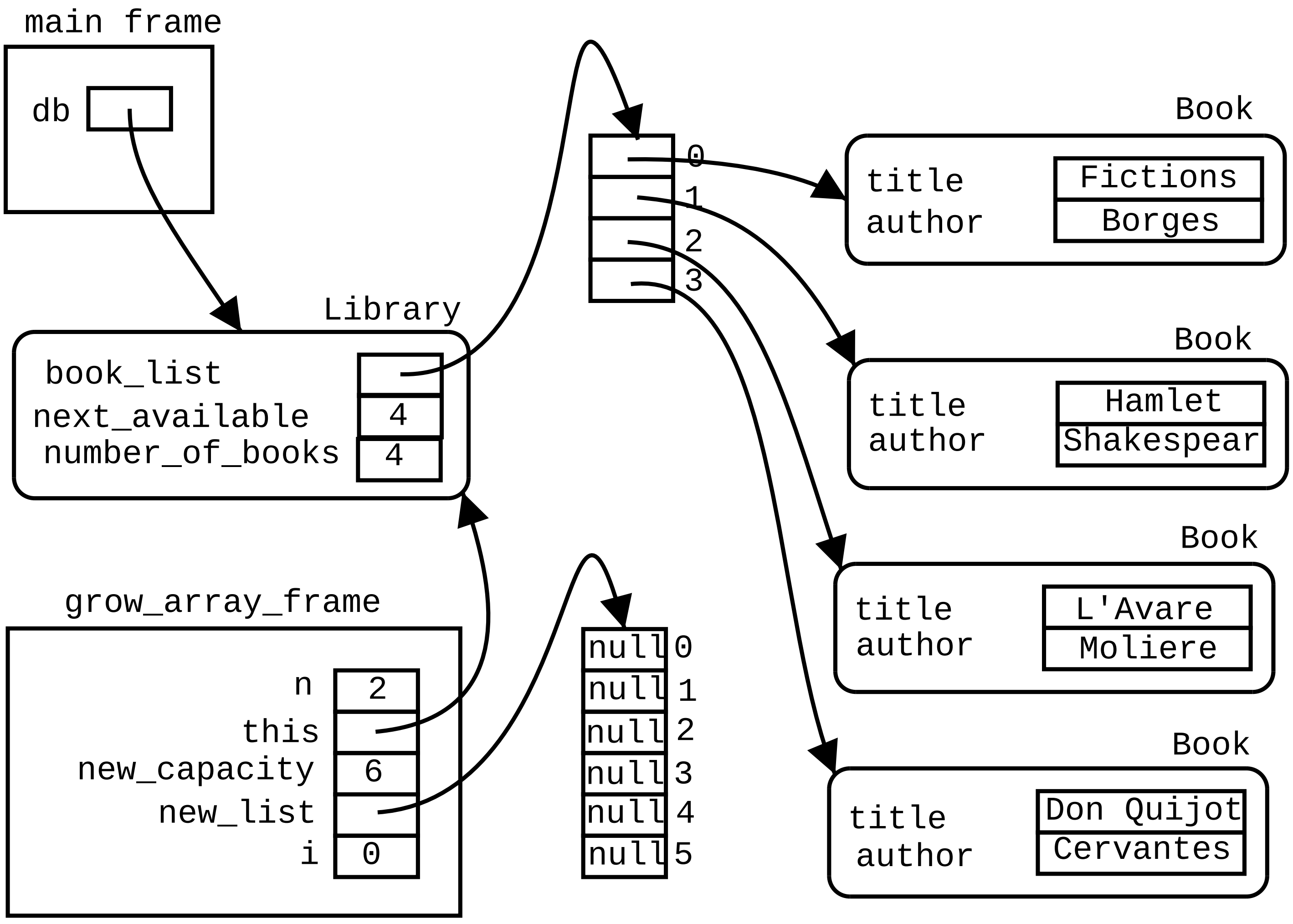
The method is private to ensure encapsulation so that only BookDatabase objects can grow the book lists.

Growing arrays

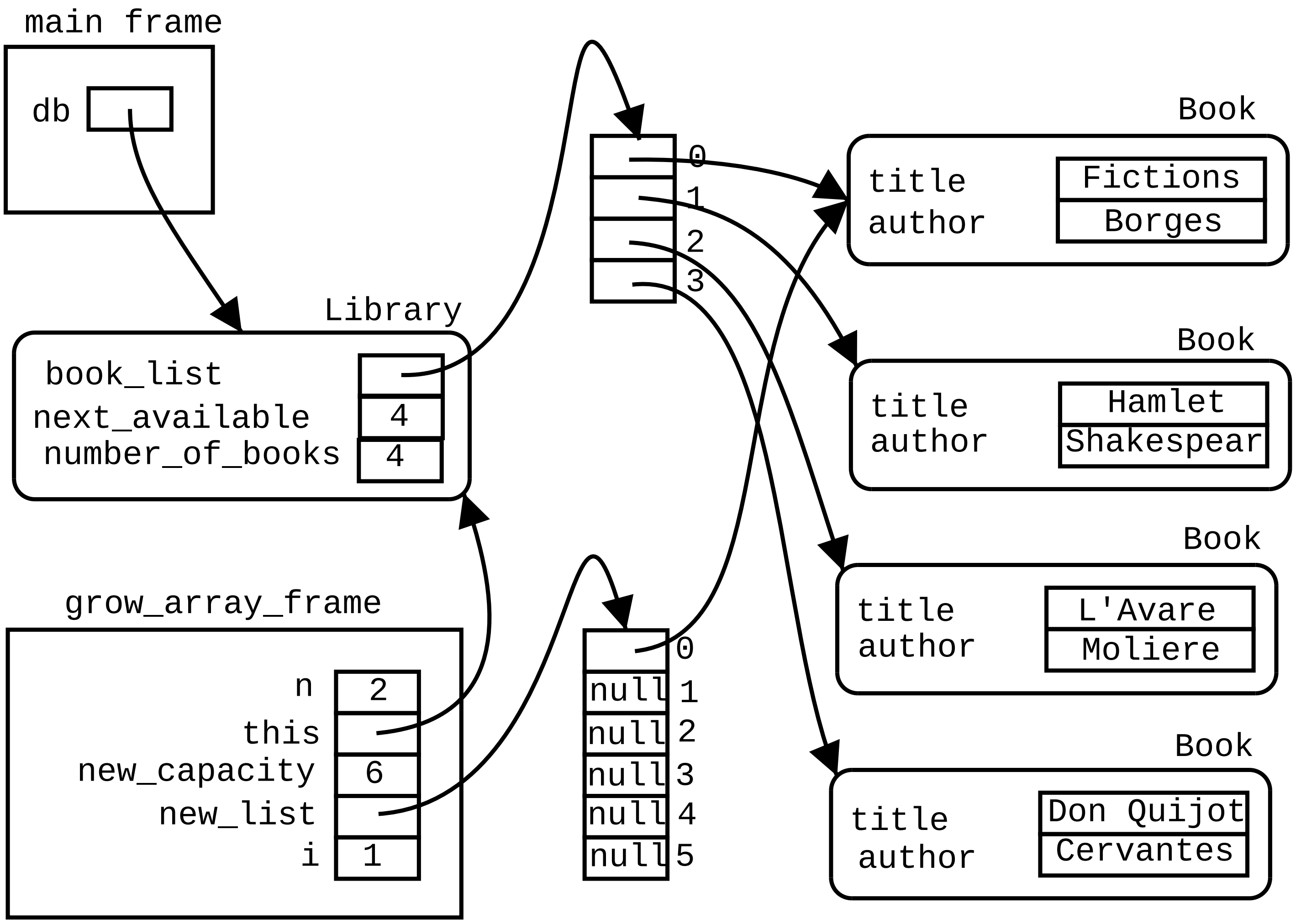


```
call grow_array(2)
```

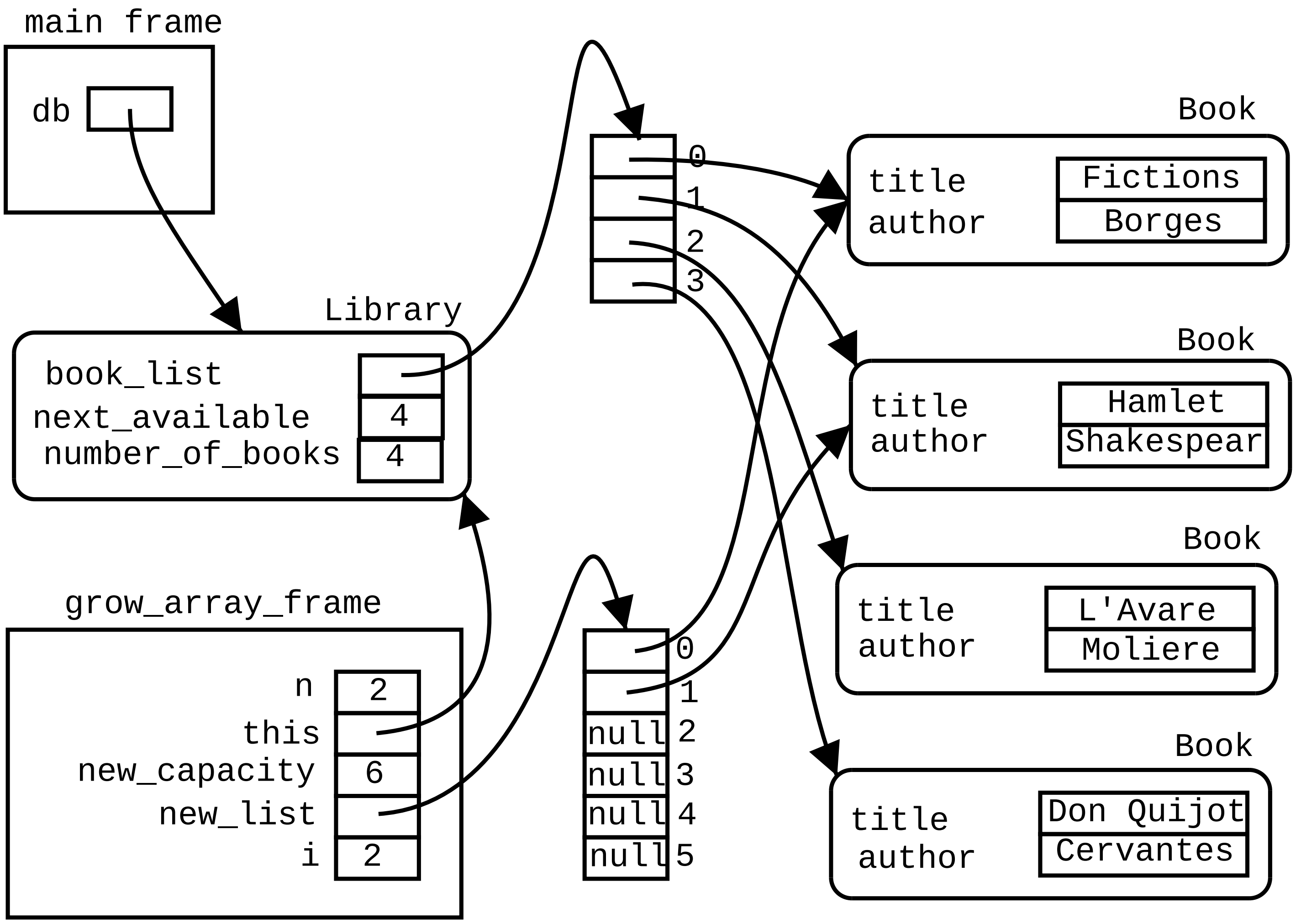
Growing arrays



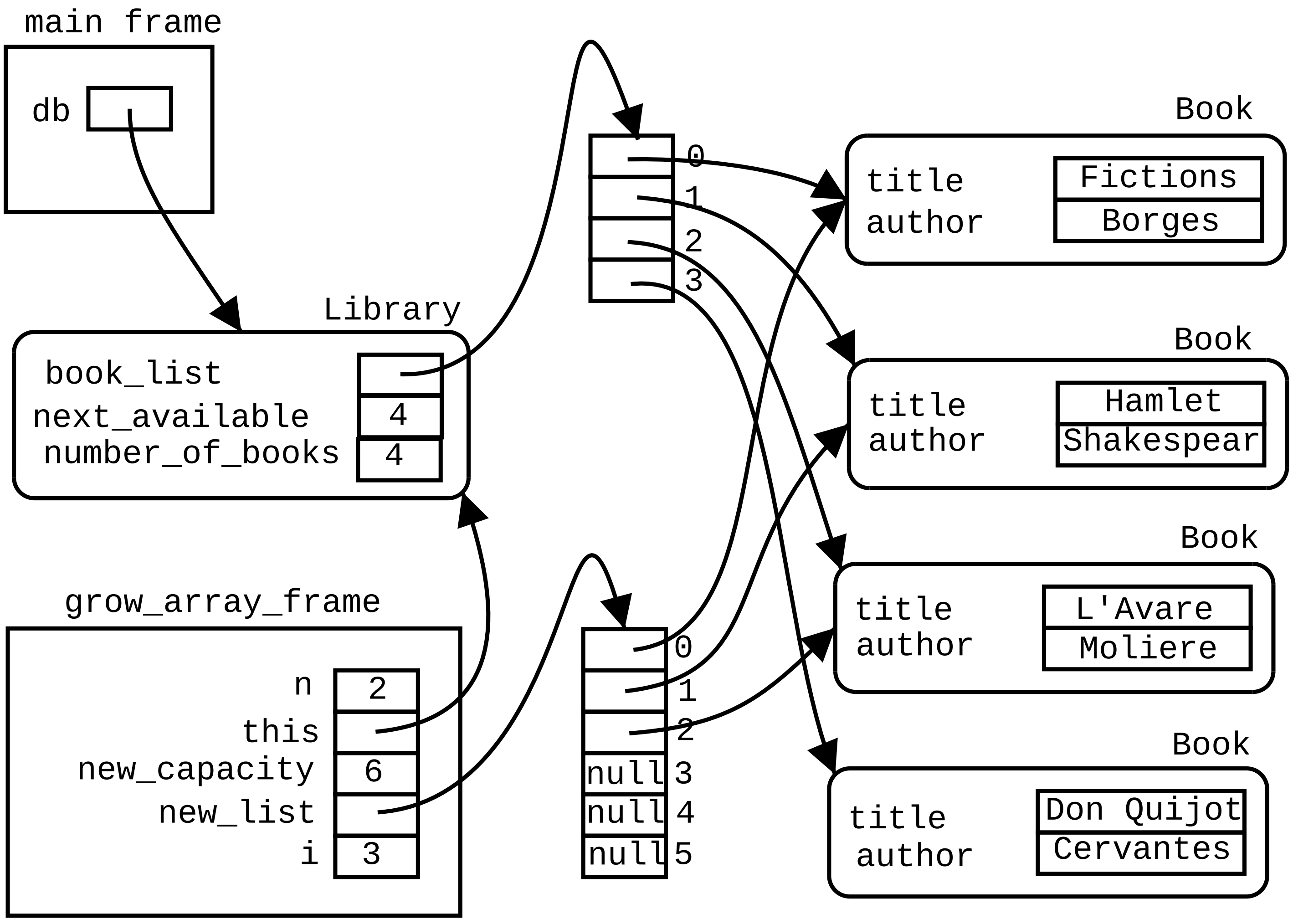
Growing arrays



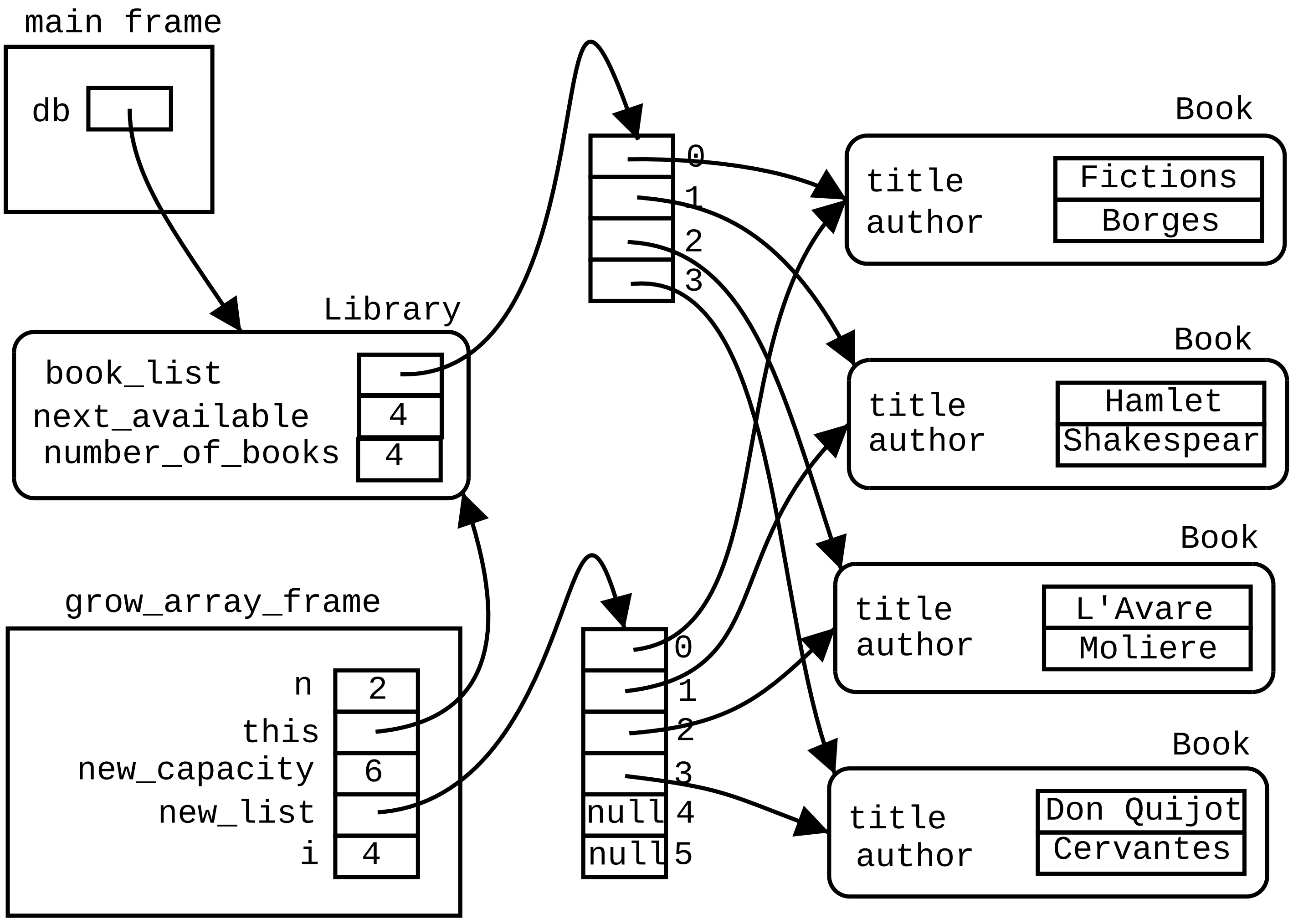
Growing arrays



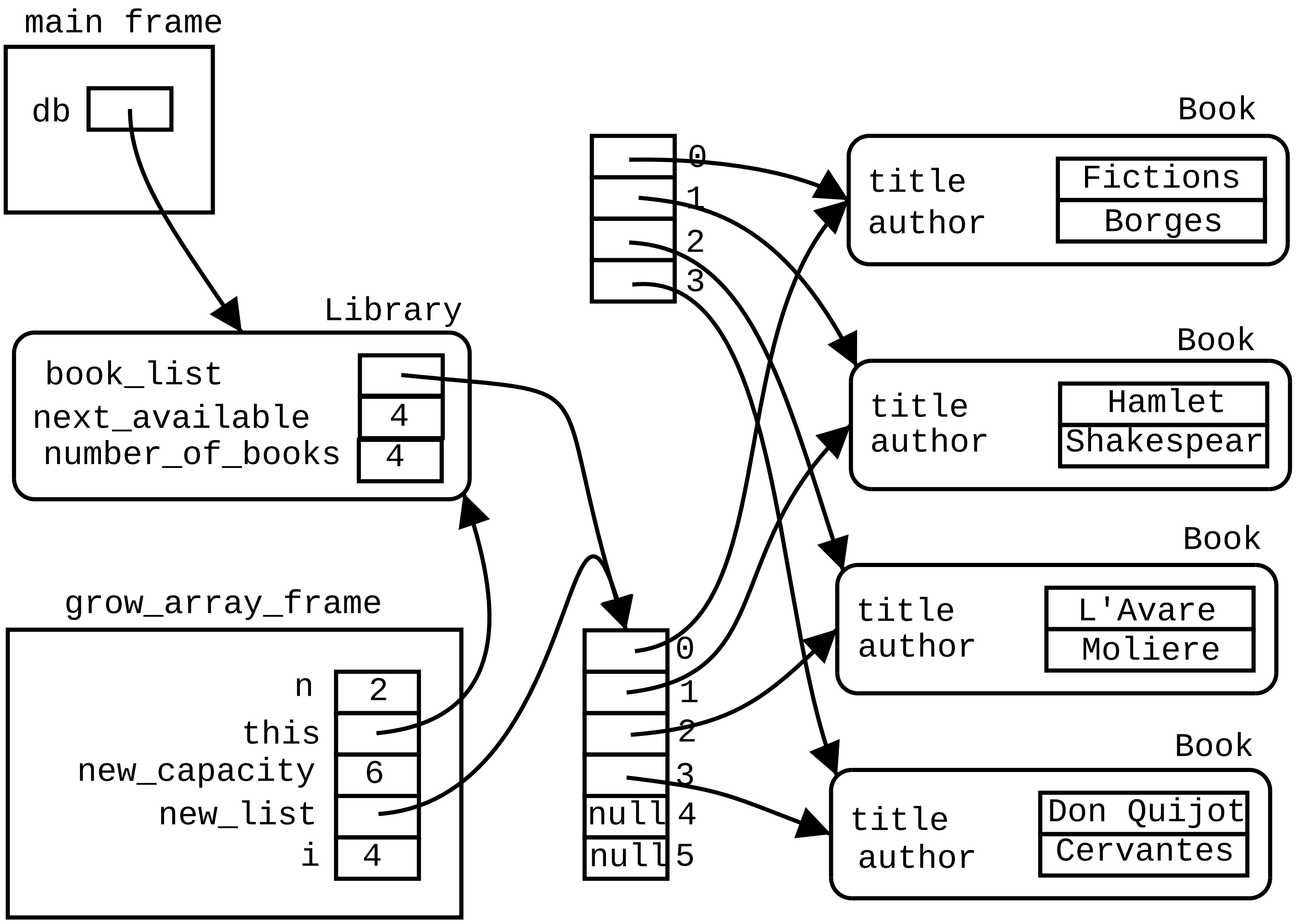
Growing arrays



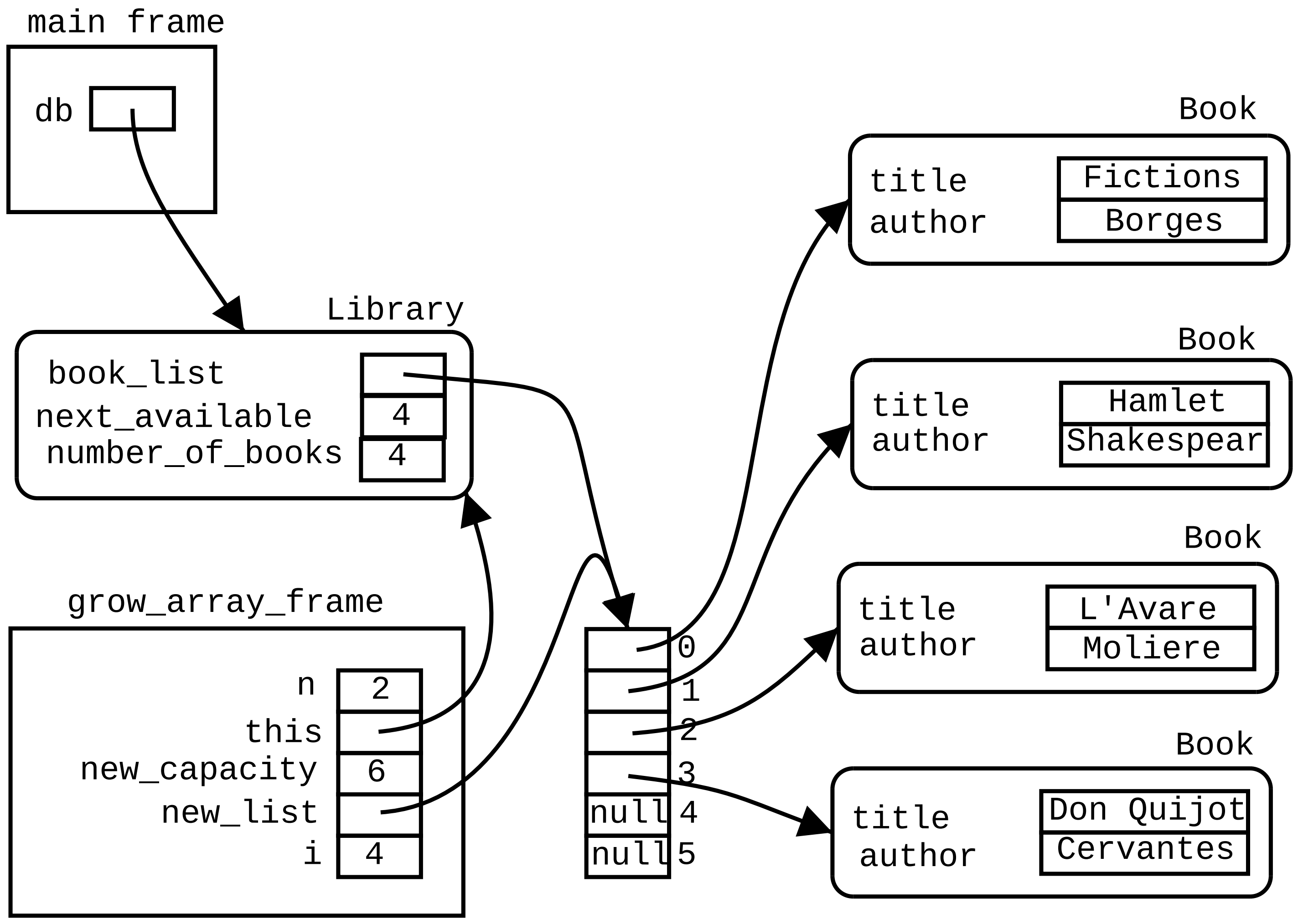
Growing arrays



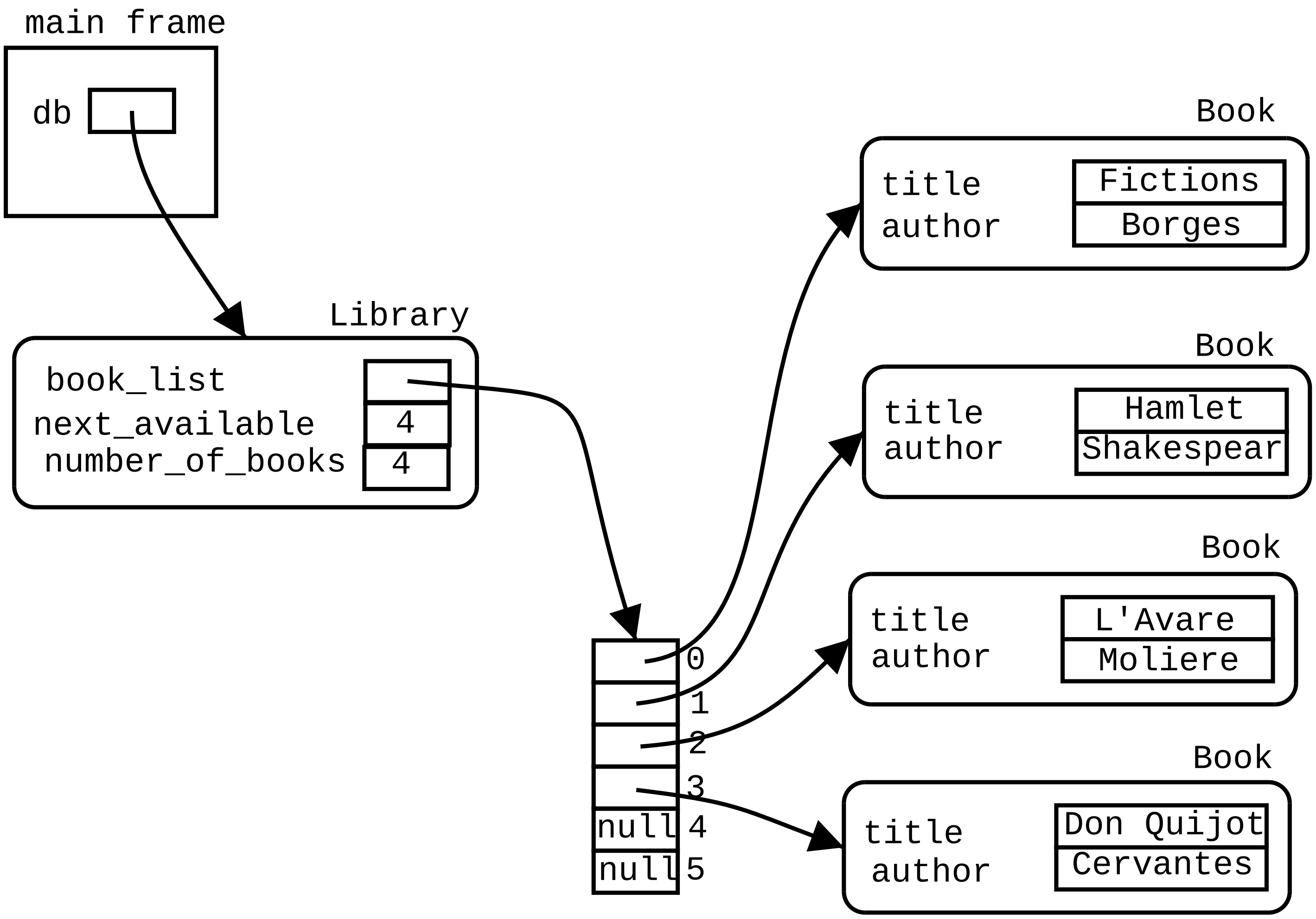
Growing arrays



Growing arrays



Growing arrays



Growing arrays

```
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 the the key of interest.

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

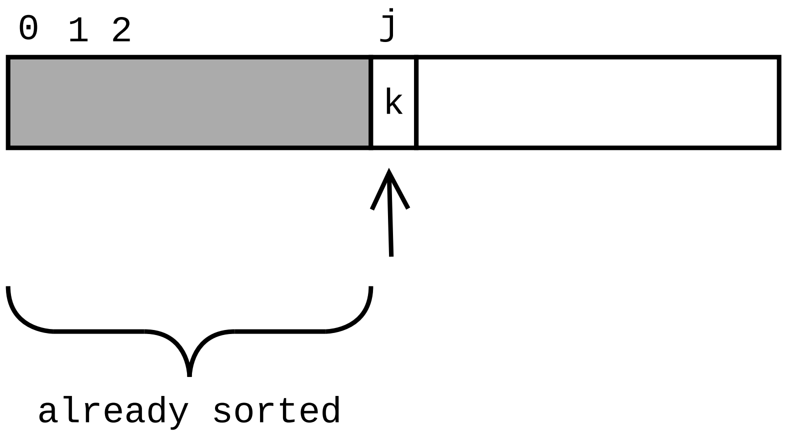
Sorting algorithms

- 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.

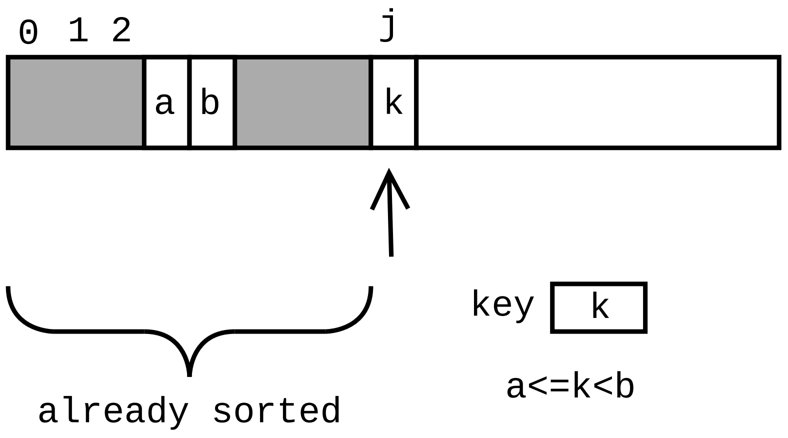
- Idea: sorting a set of cards can be done by inserting a card in the subset of the cards which are already sorted.



Insertion sort

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

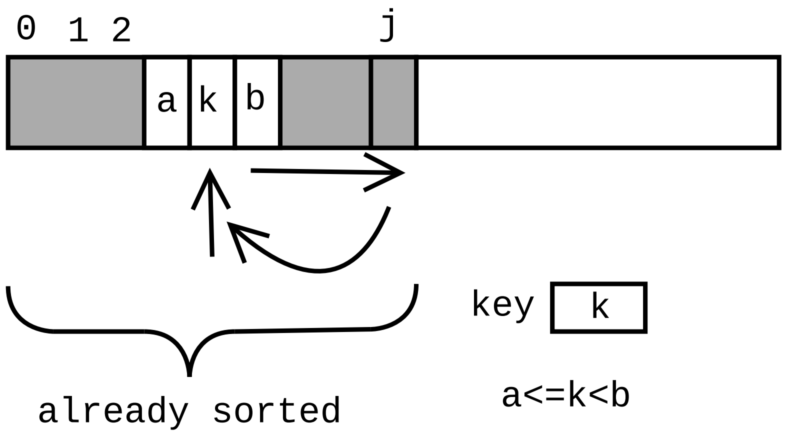
- Idea: sorting a set of cards can be done by inserting a card in the subset of the cards which are already sorted.



Insertion sort

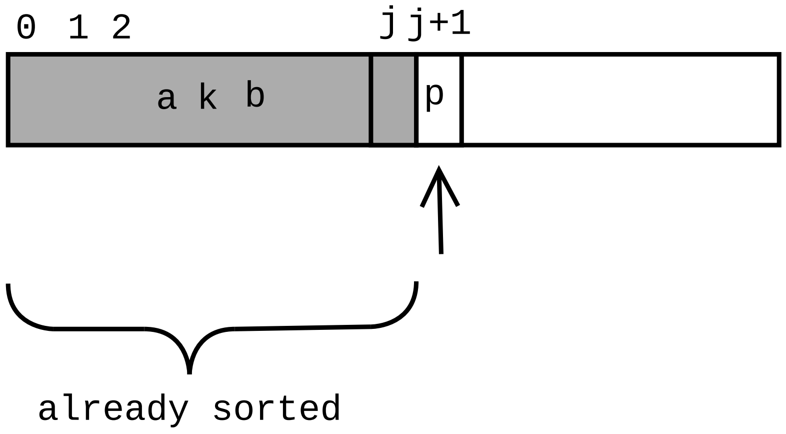
- Notation (not Java!): `a[i..j]` is the part of the array from the `i`-th index to the `j`-th index.

- 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

- 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.



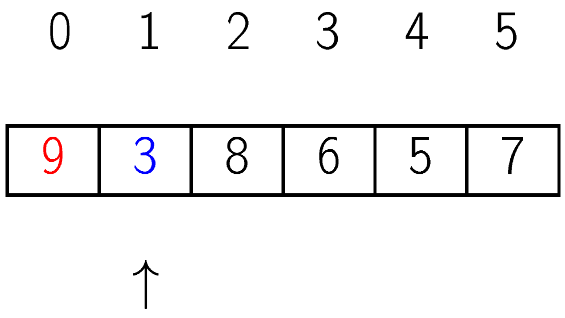
Insertion sort

- Example:

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

Insertion sort

- Example:



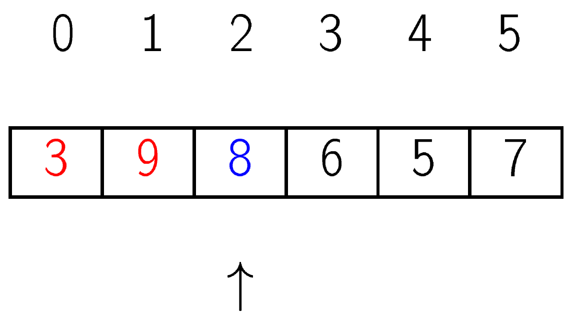
Insertion sort

- Example:

0	1	2	3	4	5
3	9	8	6	5	7
	↑				

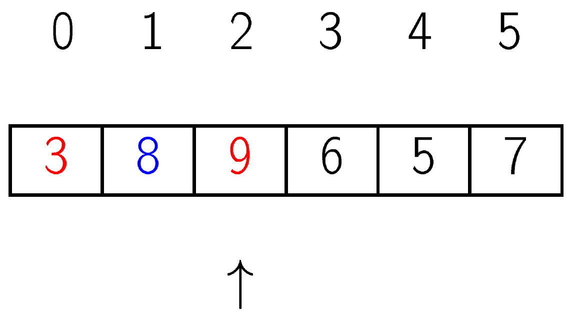
Insertion sort

- Example:



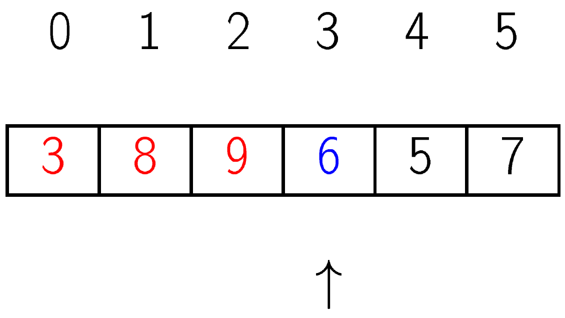
Insertion sort

- Example:



Insertion sort

- Example:



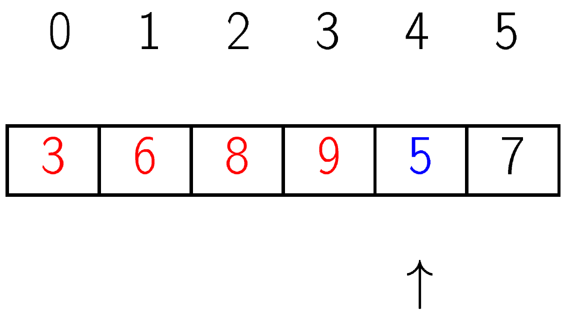
Insertion sort

- Example:

0	1	2	3	4	5
3	6	8	9	5	7
			↑		

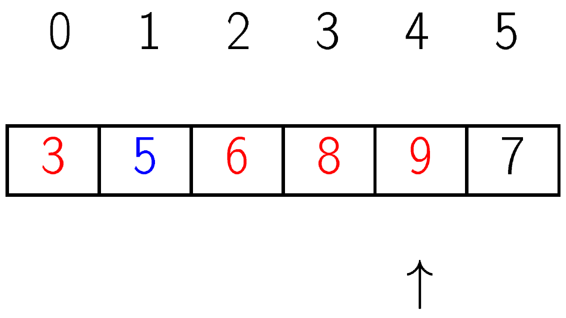
Insertion sort

- Example:



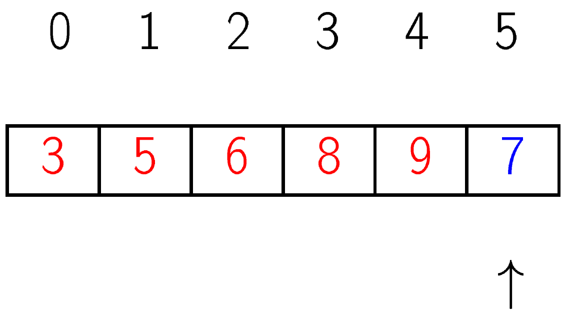
Insertion sort

- Example:



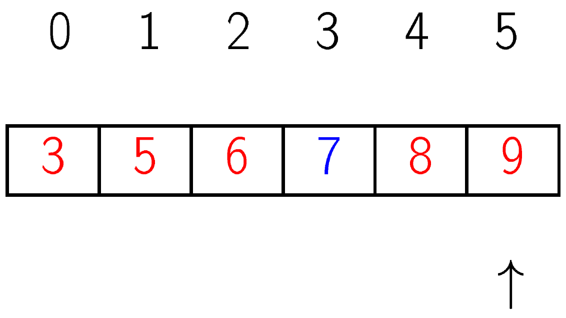
Insertion sort

- 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[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)

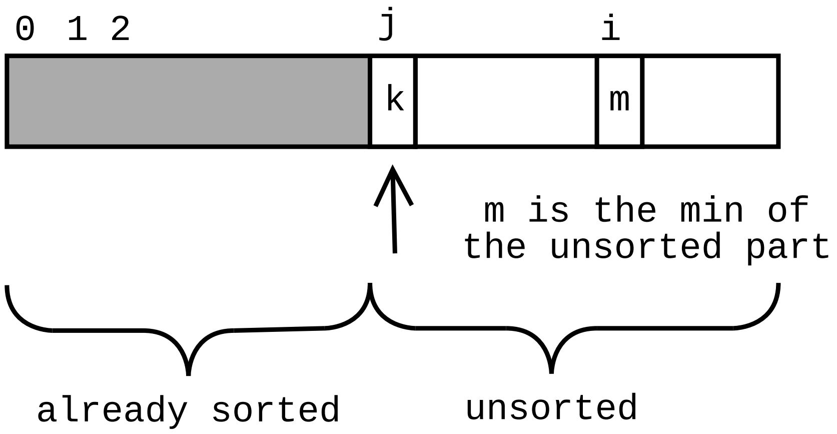
```
1. For each j from 1 to the length of a-1
   (a) Set key to a[j]
   (b) Set i to j - 1
   (c) While i >= 0 and a[i] > key do
         i. Set a[i+1] to a[i]
        ii. Decrement i by 1
   (d) Set a[i+1] to key
```

Insertion sort

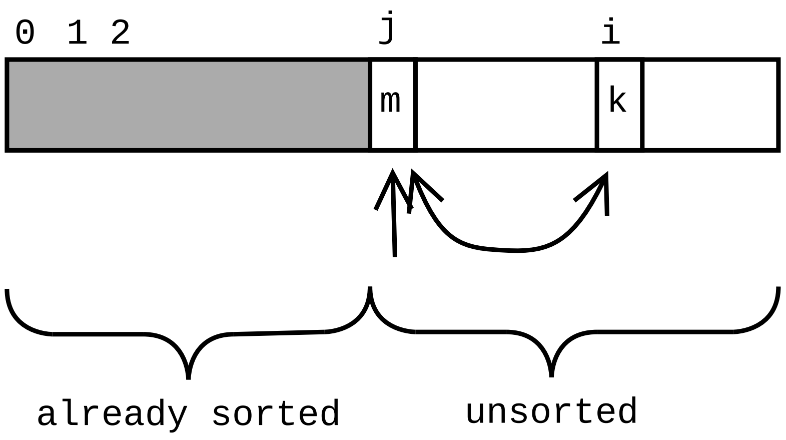
- Implementation

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

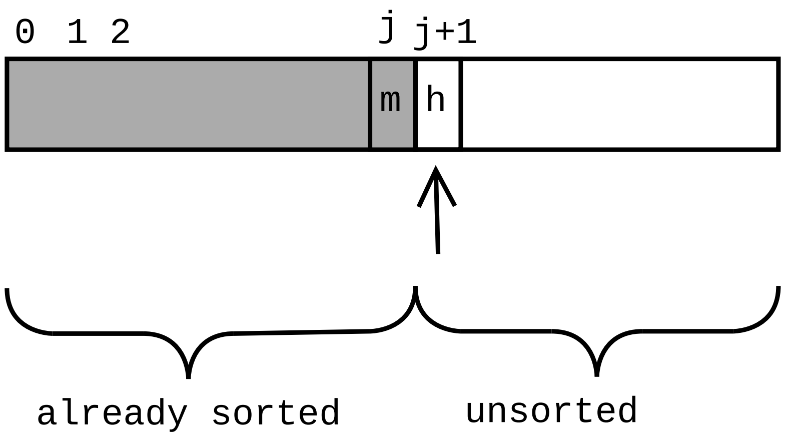
Selection sort



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
$\uparrow$	$\Uparrow$				

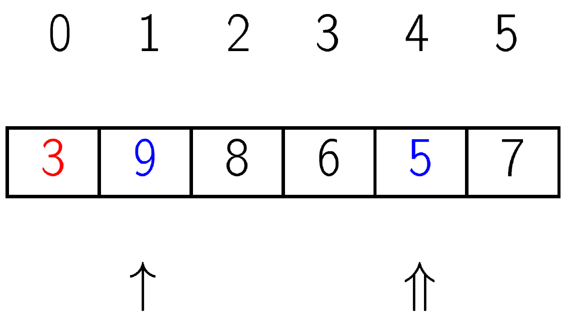
Selection sort

- Example:

0	1	2	3	4	5
3	9	8	6	5	7
	↑				

Selection sort

- Example:



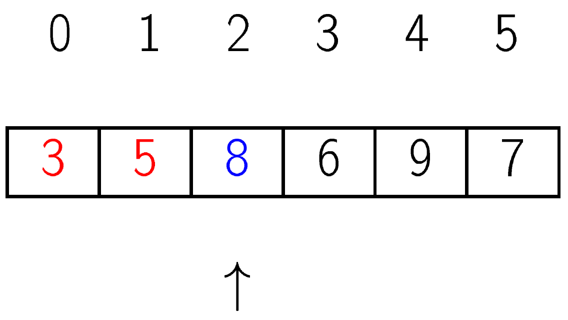
Selection sort

- Example:

0	1	2	3	4	5
3	5	8	6	9	7
	$\uparrow$			$\Uparrow$	

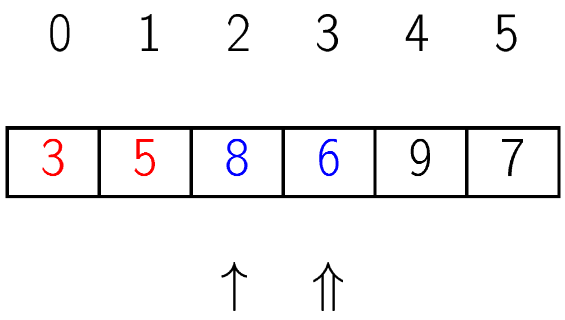
Selection sort

- Example:



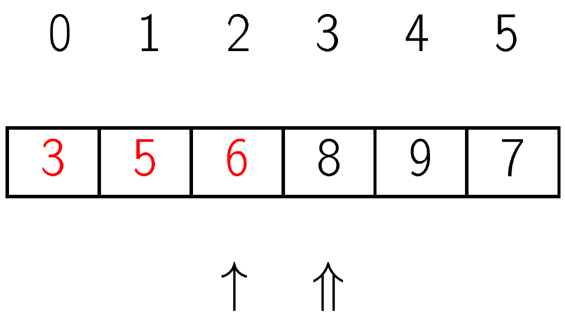
Selection sort

- Example:



Selection sort

- Example:



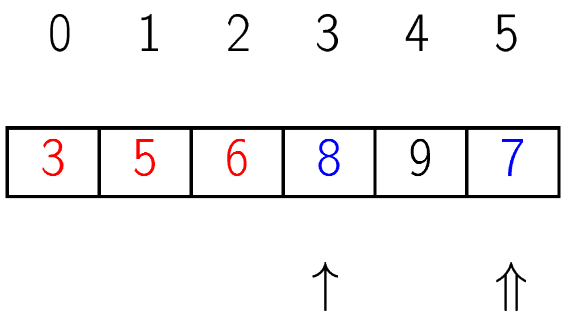
Selection sort

- Example:

0	1	2	3	4	5
3	5	6	8	9	7
			↑		

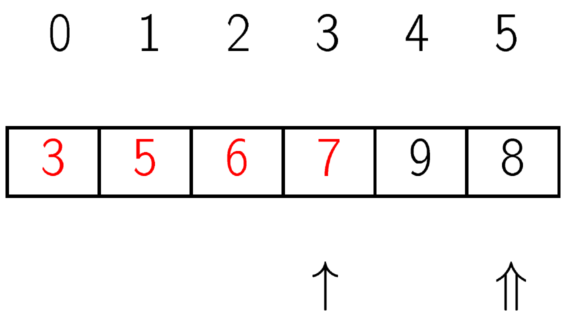
Selection sort

- Example:



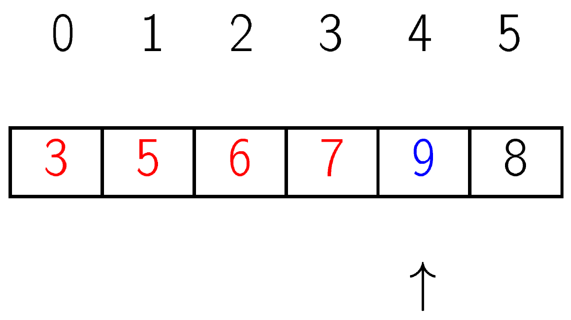
Selection sort

- Example:



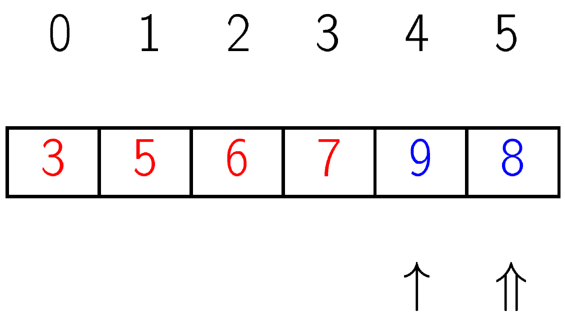
Selection sort

- Example:



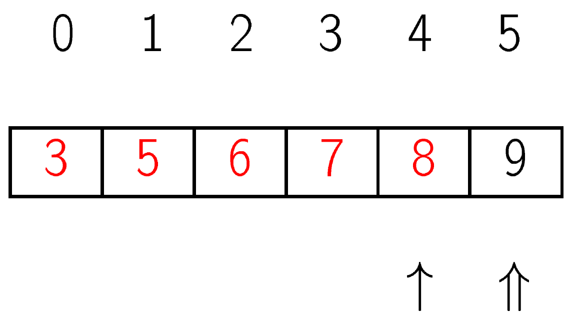
Selection sort

- 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 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;
  }
}
```

The end