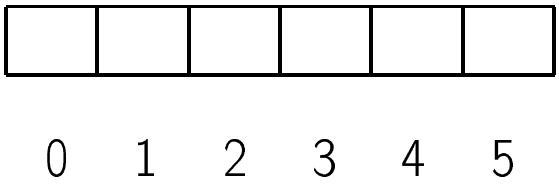
Arrays

- 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 are declared as follows:

  type `[]` *name* `;`

- Where *type* is any data type (primitive or user-defined), called the *base type* of the array.

Review

- 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:

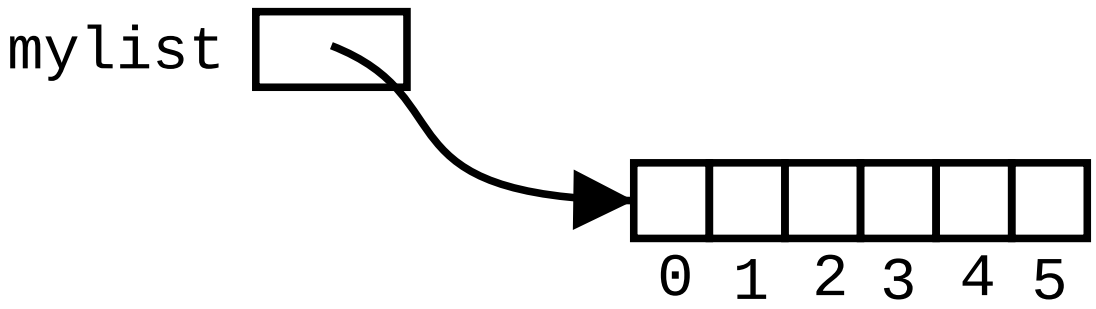
  variable`[`*integer-expression*`]` `=` *expression*`;`

Arrays (contd.)

- But declaring an array does not create the array itself, only a reference.
- To create an array we use the new keyword.

```
mylist = new int[6];
```

- Where the variable mylist is actually a reference to the aray itself



Array access

- To access individual elements of an array we use the indexing operator [.]: If variable is a reference to an array, and *number* is a positive integer, or 0, then the position *number* can be accessed by

  variable[*number*]

- For example `mylist[0]` refers to the first position of mylist, `mylist[1]` to the second, `mylist[2]` to the third, and so on.

- To write a value in the array, we can use the assignment operator:

  variable[*number*] = *expression*;

- Where *expression* must be of the same type as the base type of the array.

Example 1

- Finding the minimum number in an array

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

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:

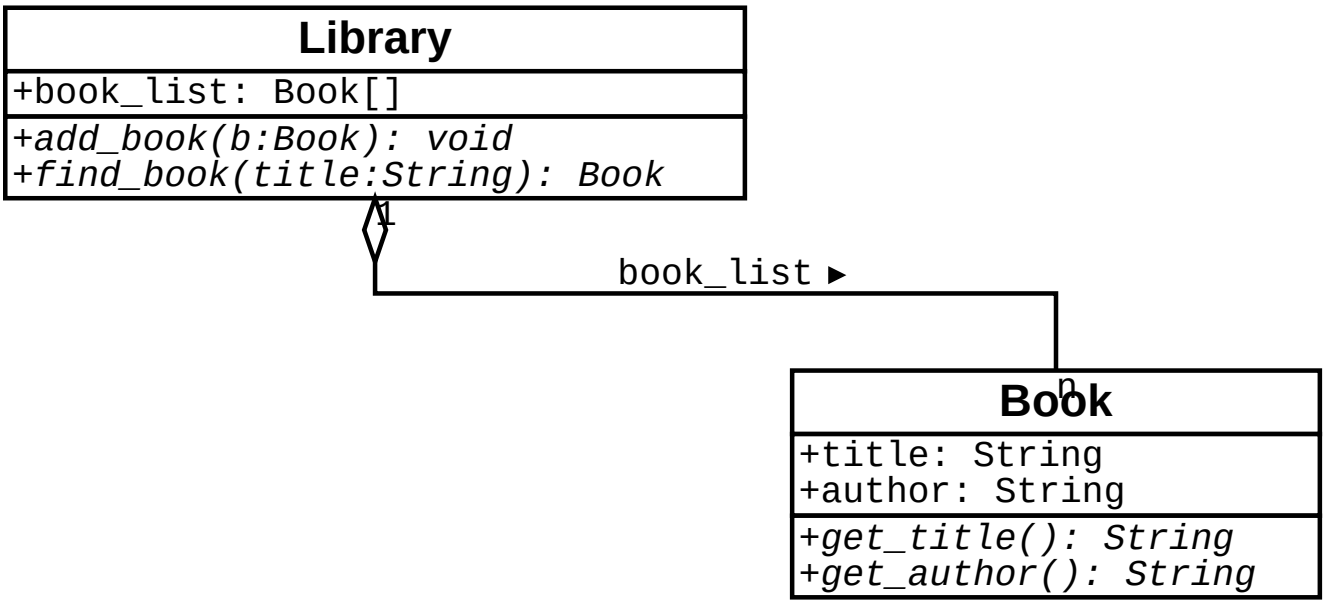
```
for (int i=0; i < list.length; i++)
  list[i] = new B(3);
```

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

Array applications (contd.)

- Using the database

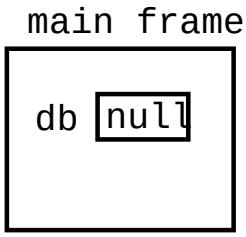
```
public class Test {
  public static void main(String[] args)
  {
    Library db = new Library(10000);
    Book m;
    m = new Book(“Fictions”,“Borges”);
    db.add_book(m);
    m = new Book(“Hamlet”,“Shakespeare”);
    db.add_book(m);
    m = new Book(“L\’Avare”,“Moliere”);
    db.add_book(m);
 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

Array applications (contd.)

- 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.)

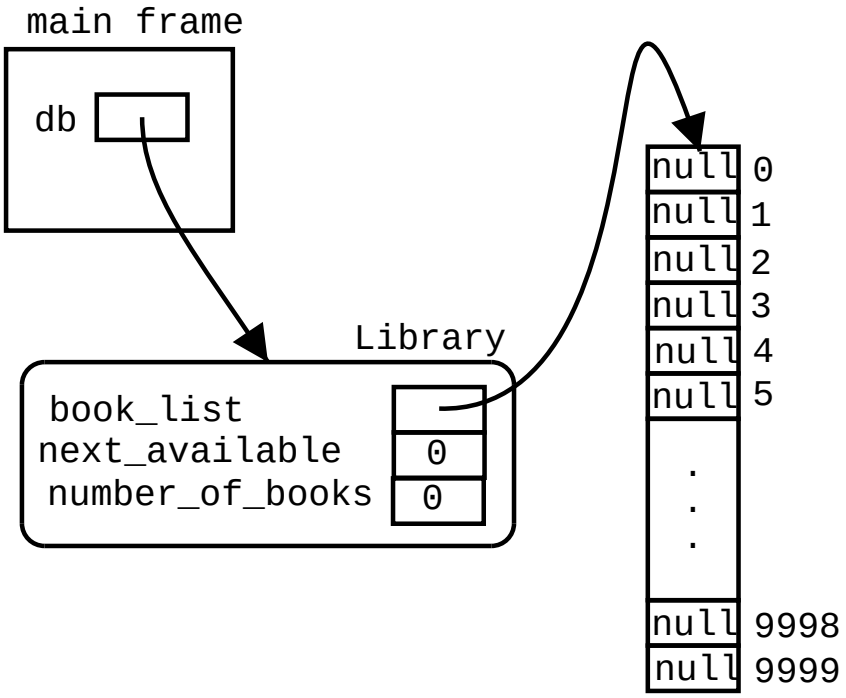


Array applications (contd.)

- 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.)

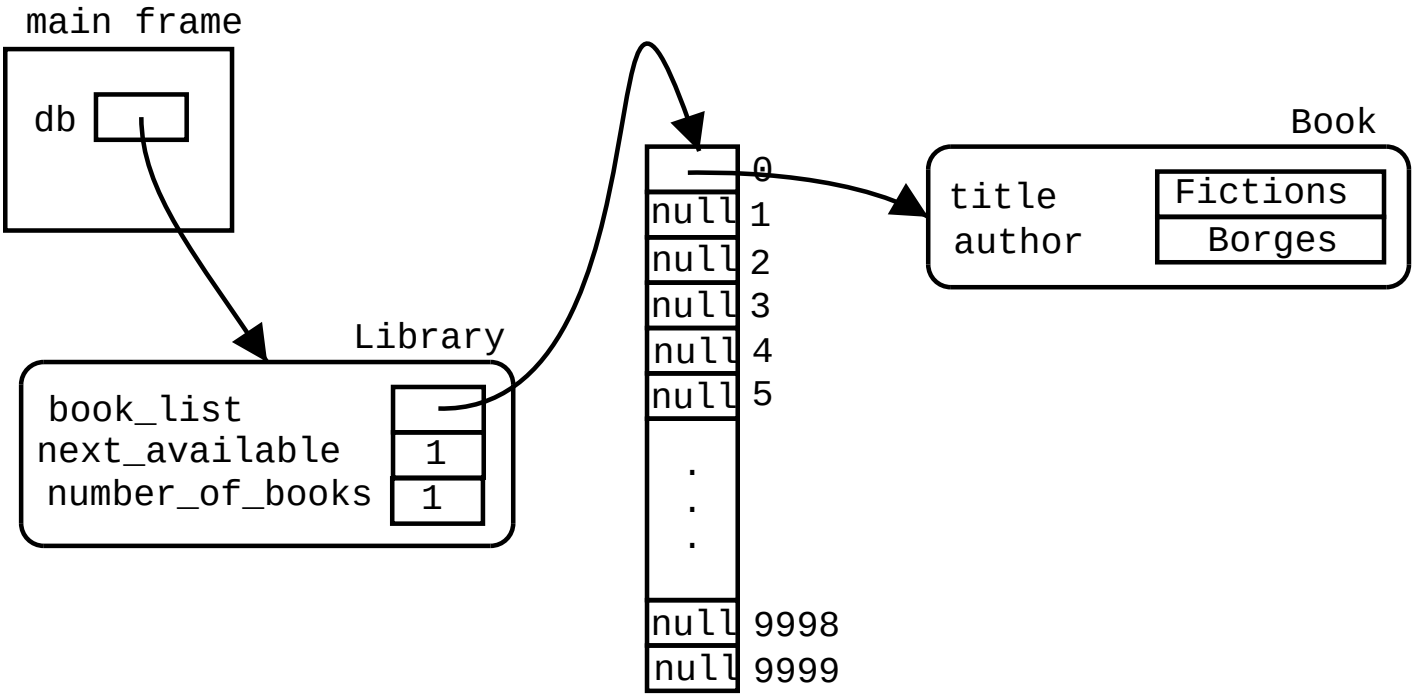


Array applications (contd.)

- 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.)

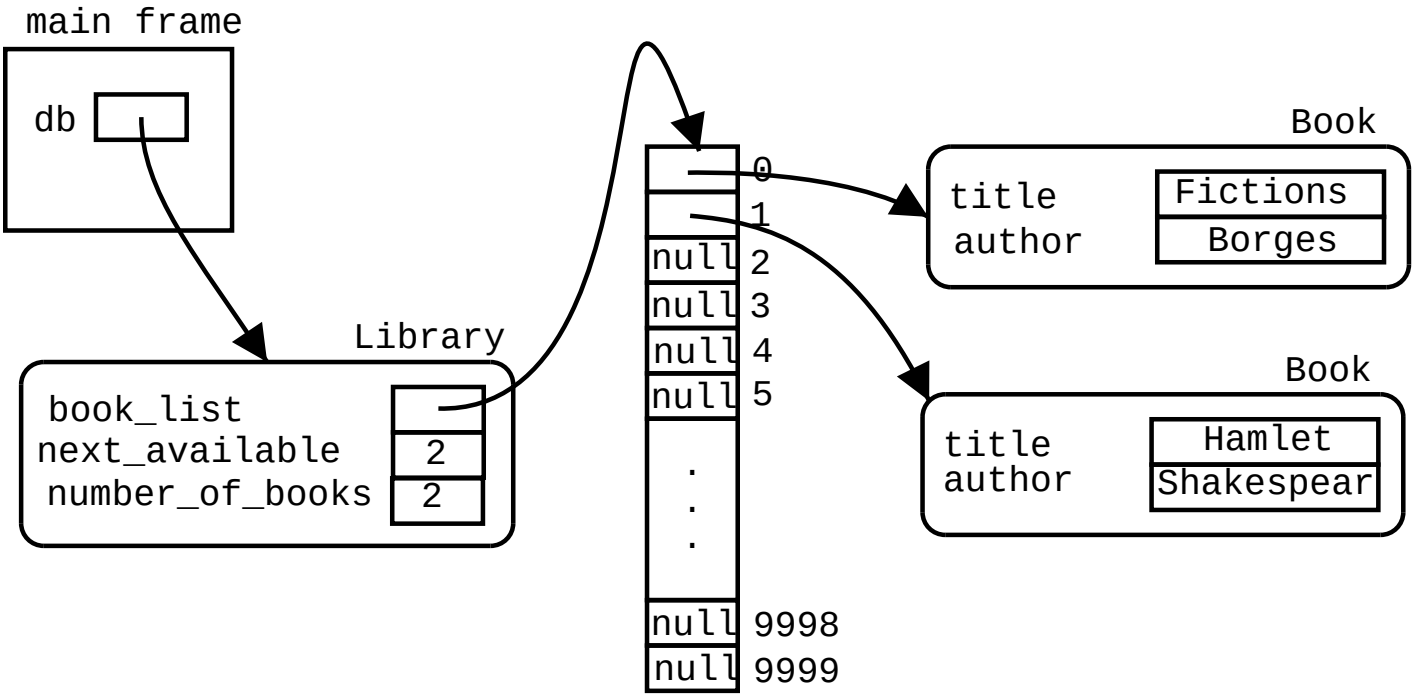


Array applications (contd.)

- 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.)

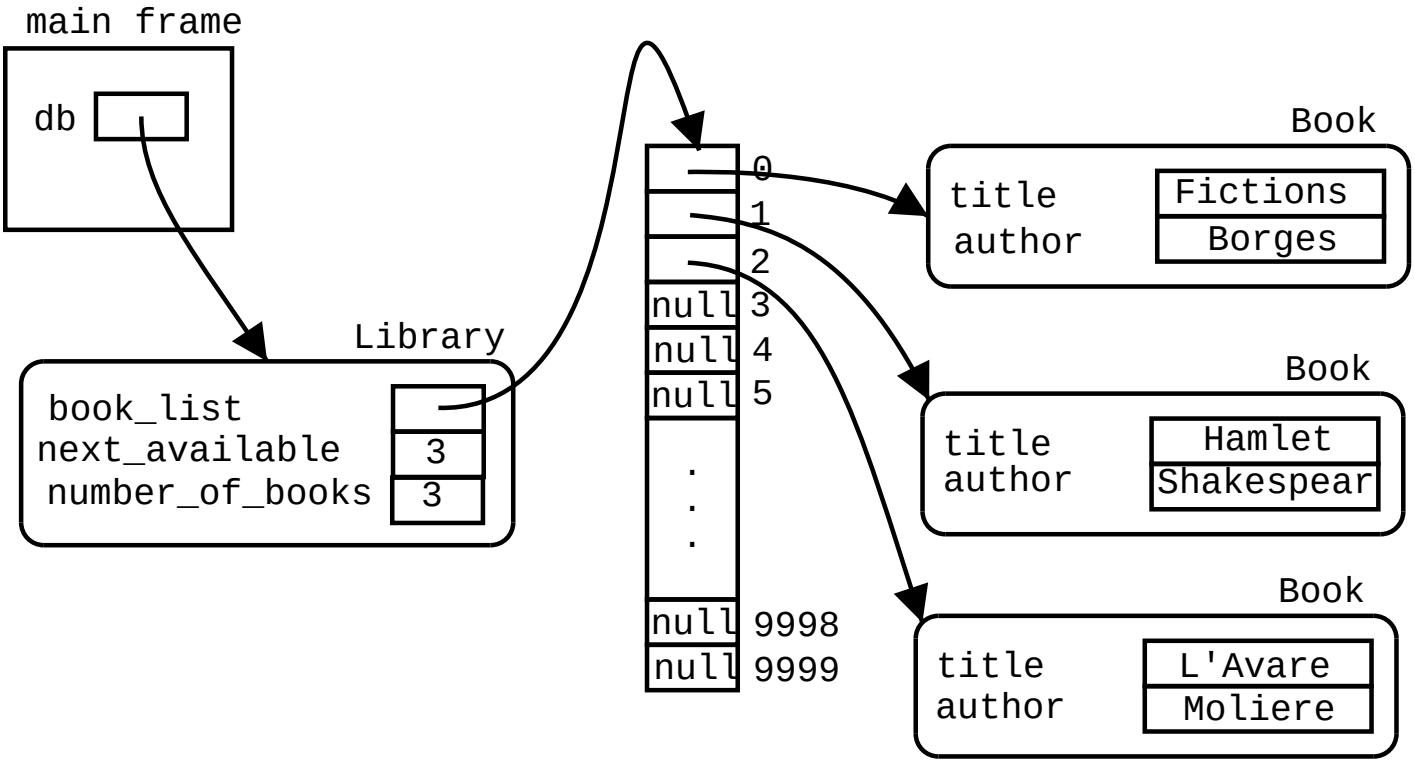


Array applications (contd.)

- 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.)

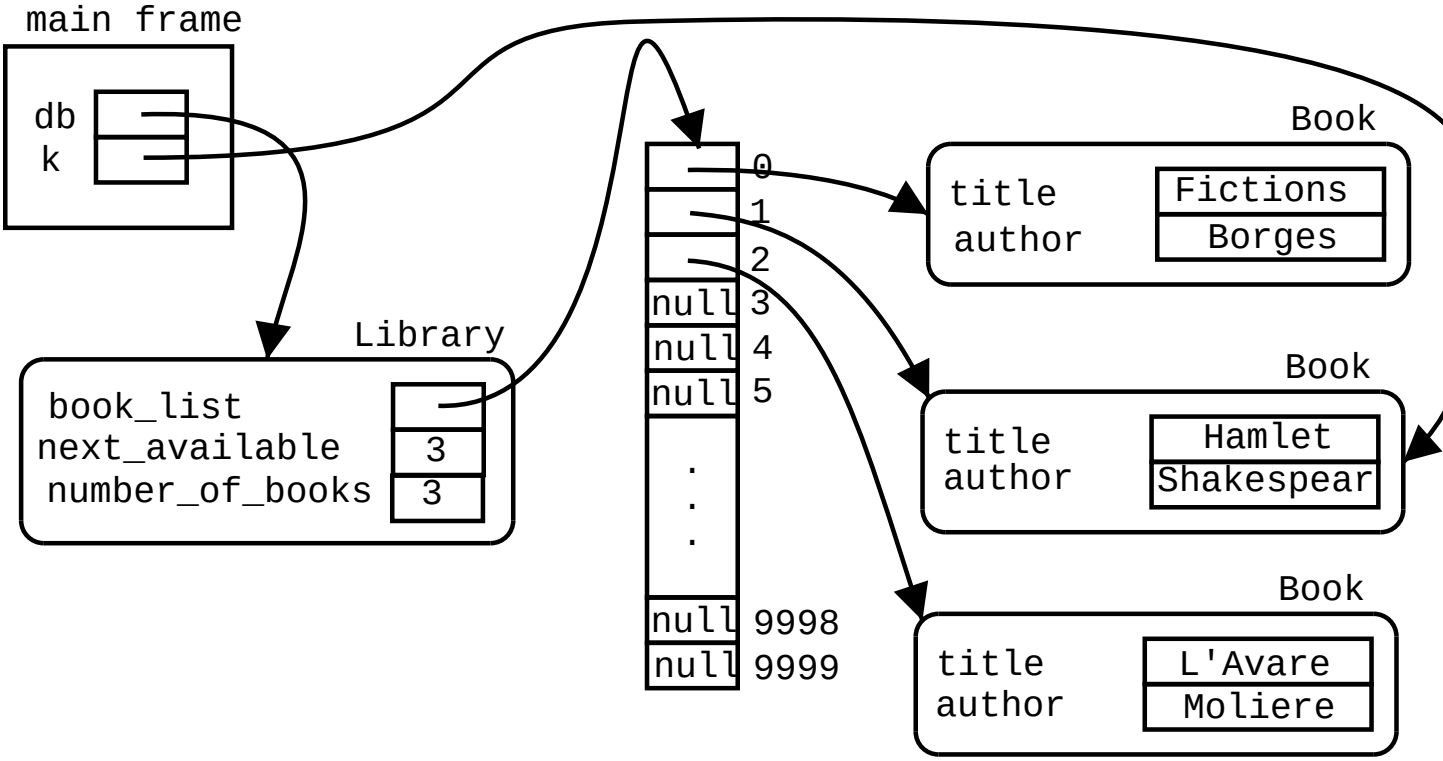


Array applications (contd.)

- 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.)

- 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.)

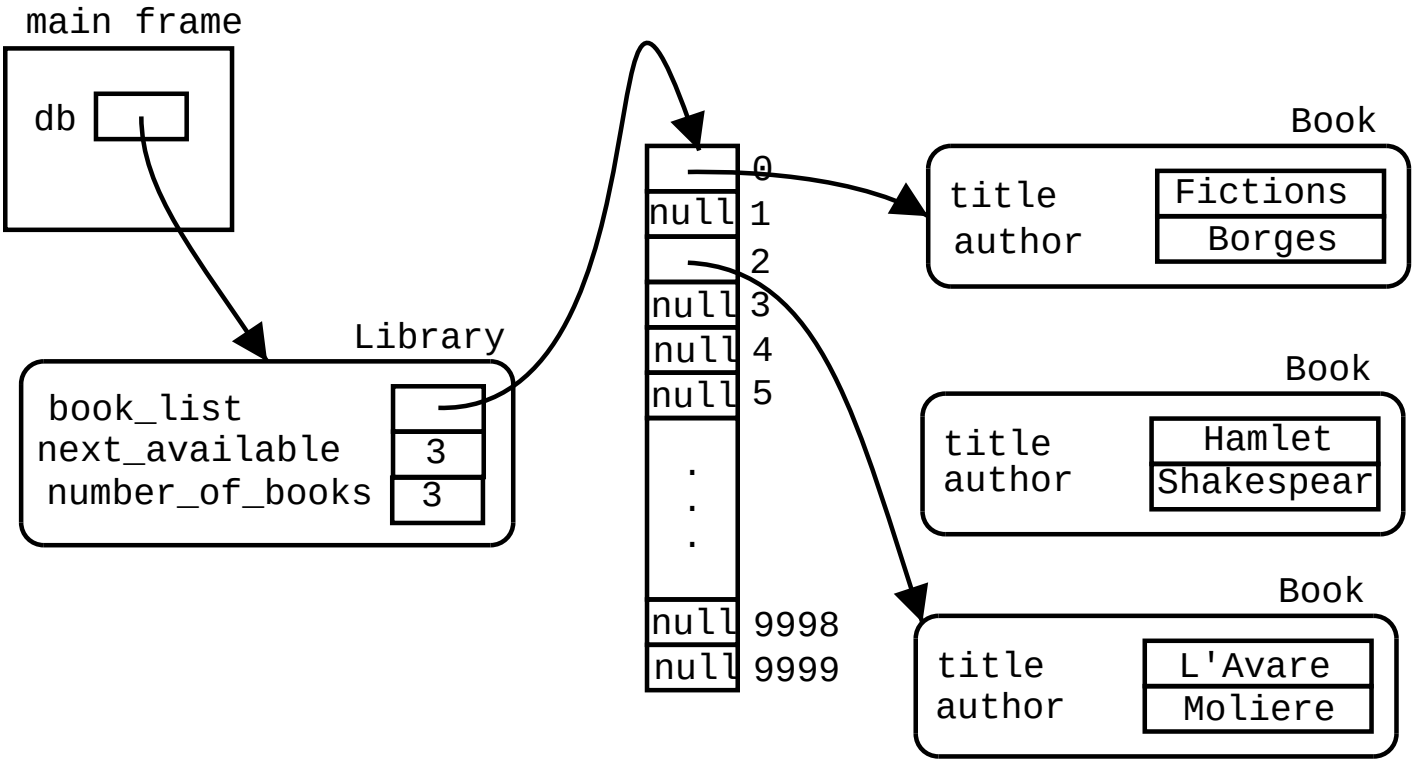
```
public int book_index(String title)
{
  for (int i=0; i < book_list.length; i++) {
    Book m = book_list[i];
    if (m != null) {
      String s = m.title();
      if (s.equals(title))
        return i;
    }
  }
  return -1;
}
public void delete_book(String title)
{
  int i = book_index(title);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

Array applications (contd.)

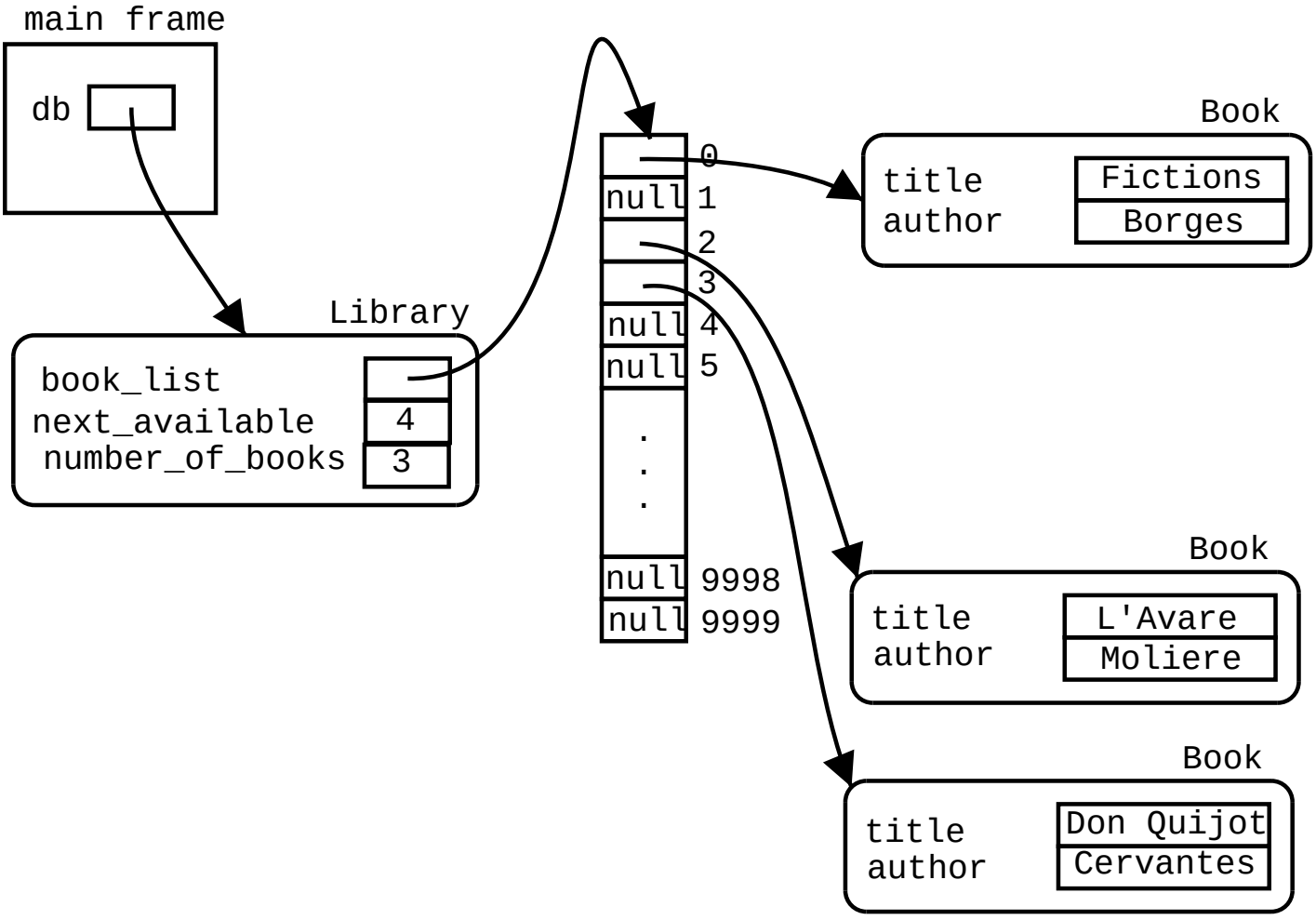
- But there's a problem: holes!

```
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);
  }
}
```

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.)

```
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)
  {
    int index = 0;
    while (index < book_list.length
        && book_list[index] != null) {
      index++;
    }
    if (index < book_list.length) {
      book_list[index] = m;
      number_of_books++;
    }
  }
```

Array applications (contd.)

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

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

Array applications (contd.)

```
  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.

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

  public Library(int max_capacity)
  {
    book_list = new Book[max_capacity];
    next_available = 0;
  }
  // Continues below...
```

Optimized Book database

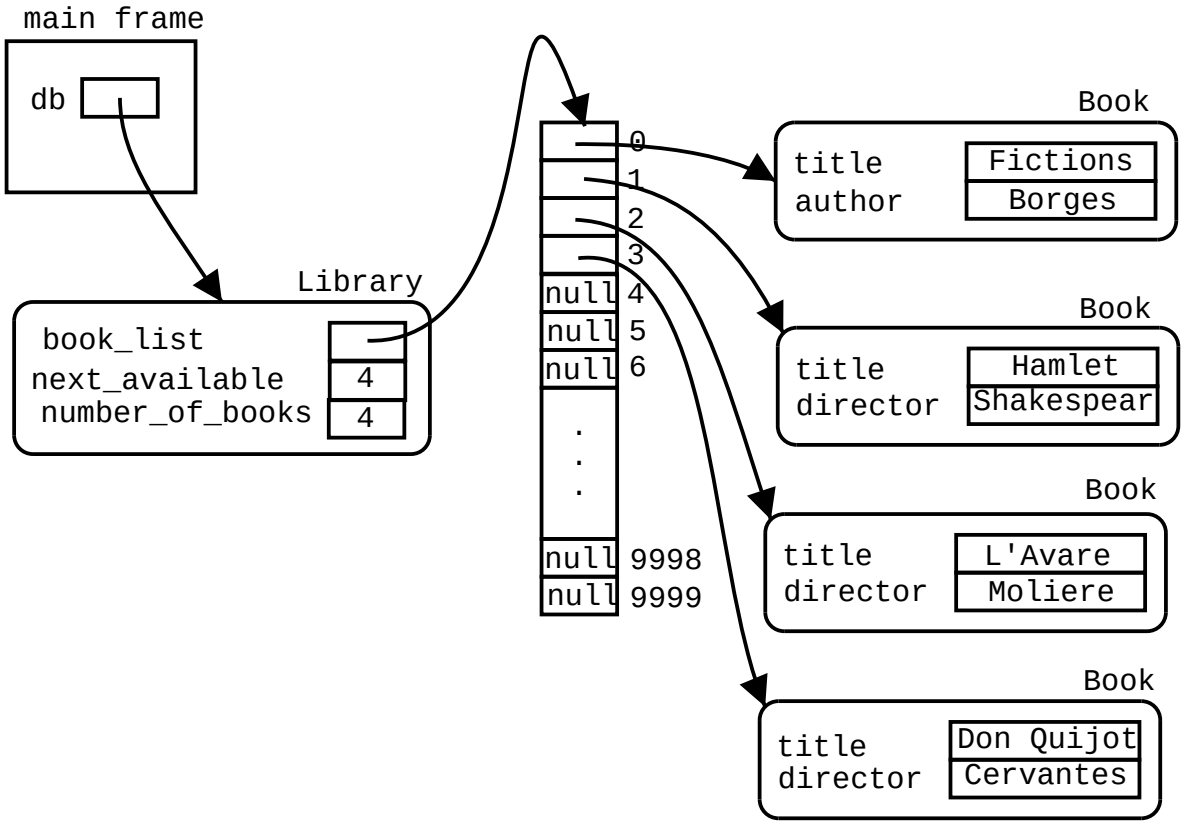
```
public void add_book(Book m)
{
  if (next_available < book_list.length) {
    book_list[next_available] = m;
    next_available++;
  }
}

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

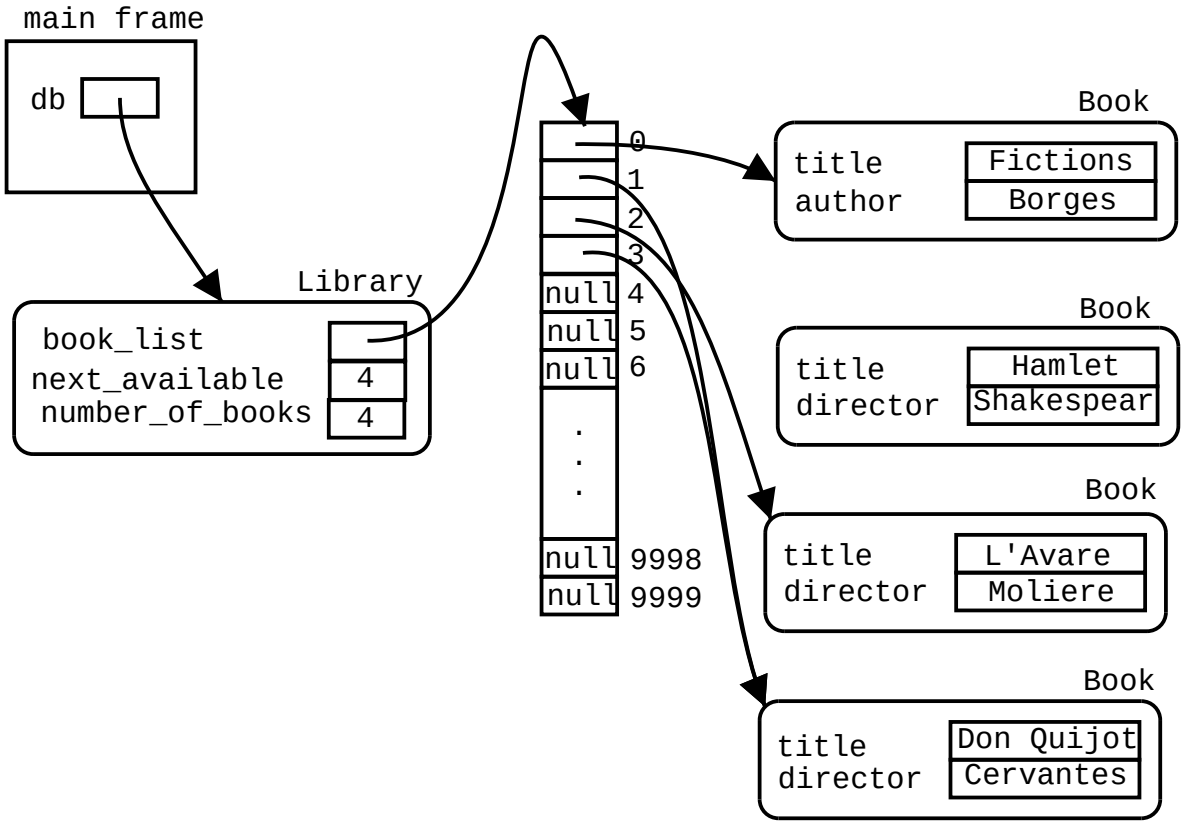
Optimized Book database

```
  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)
  {
    int i = book_index(t);
    if (i != -1) return book_list[i];
    return null;
  }
  public int number_of_books()
  {
    return next_available;
  }
}
```

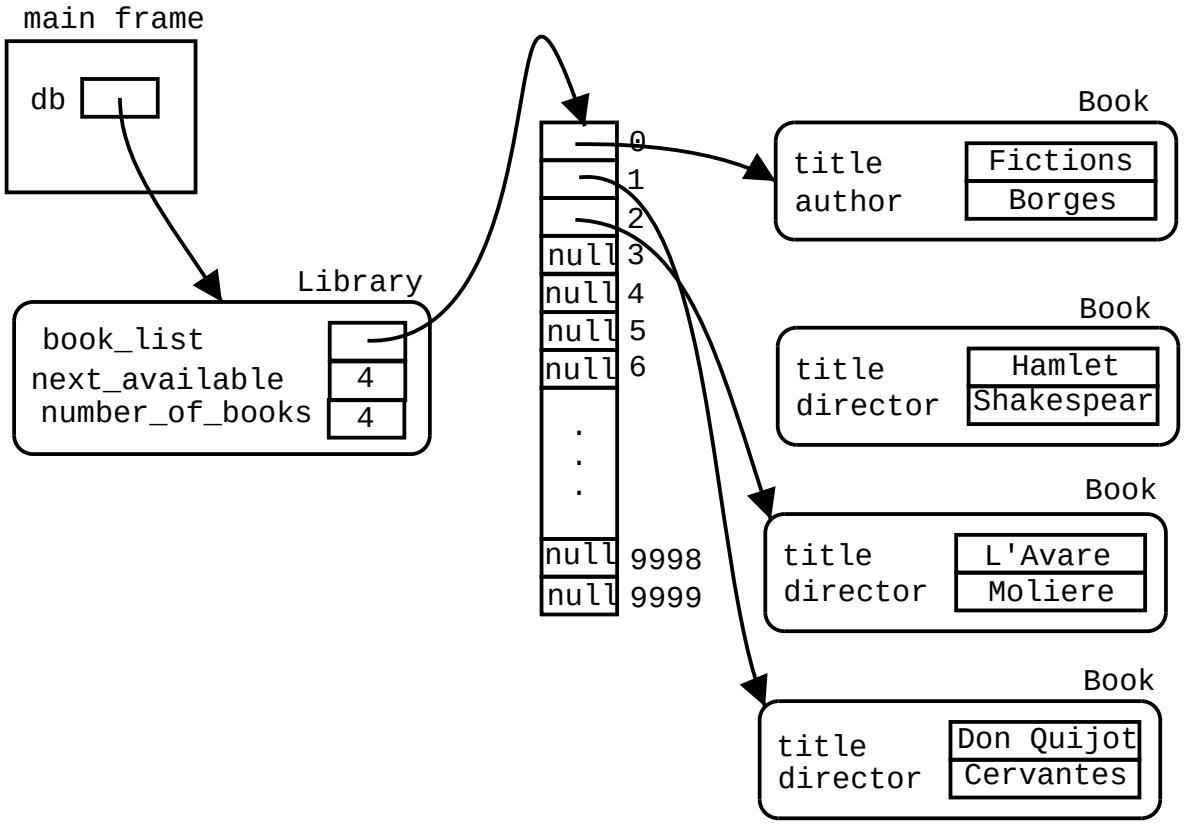
Optimized Book database



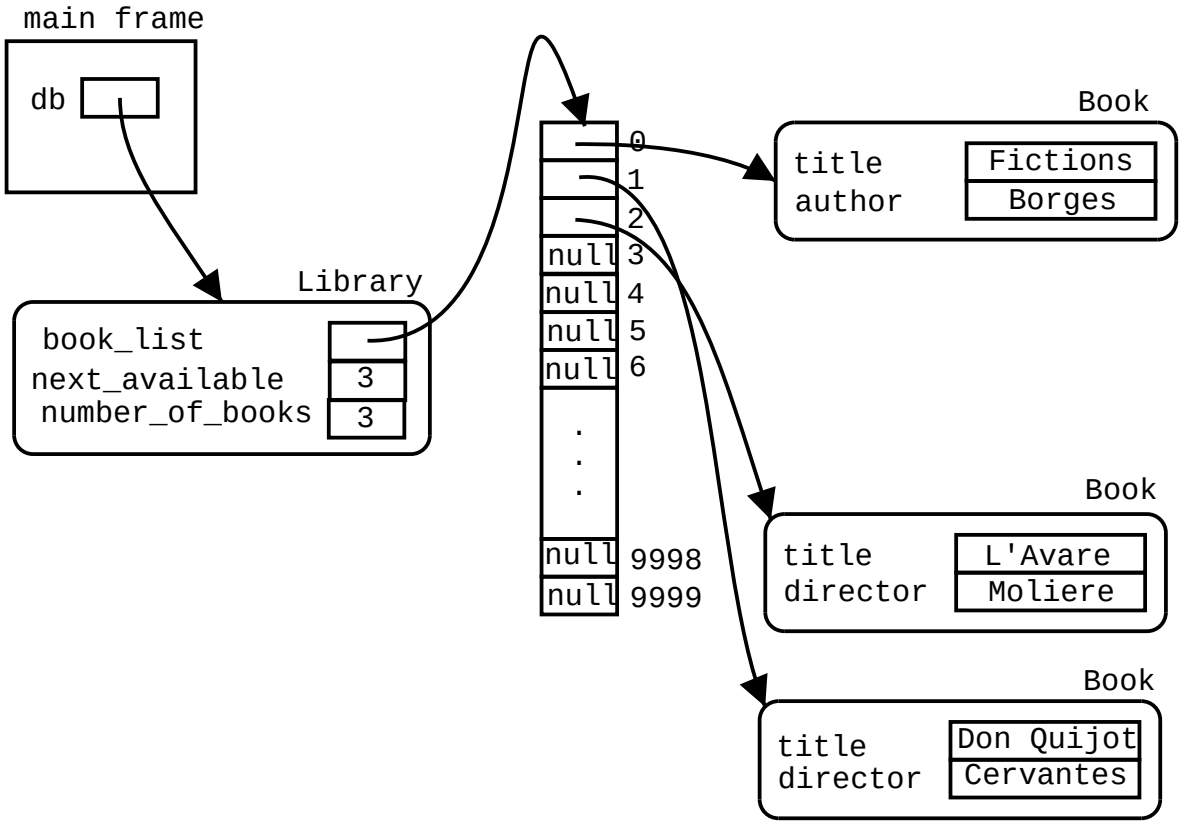
Optimized Book database



Optimized Book database



Optimized Book database



Growing arrays

- An array has a finite and fixed amount of memory.
- 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 Vector();
  }
  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:

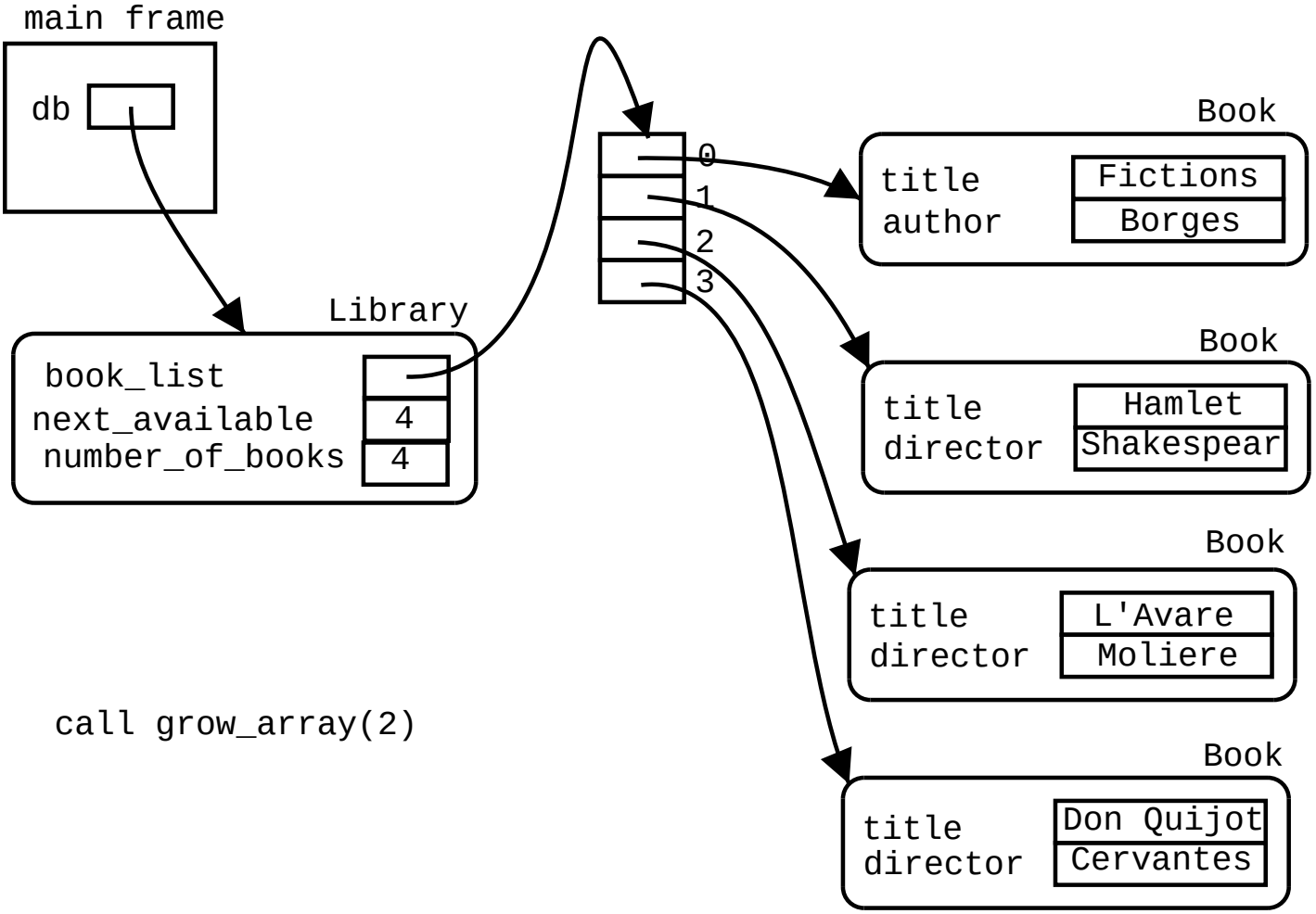
   (a) Grow the array (copying contents of the old to the new)
   (b) Find the first available cell in the new array (guarranteed to exist.)
   (c) Store m in that cell

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]; // shallow copy
    i++;
  }
  book_list = new_list; // Update list reference
}
```

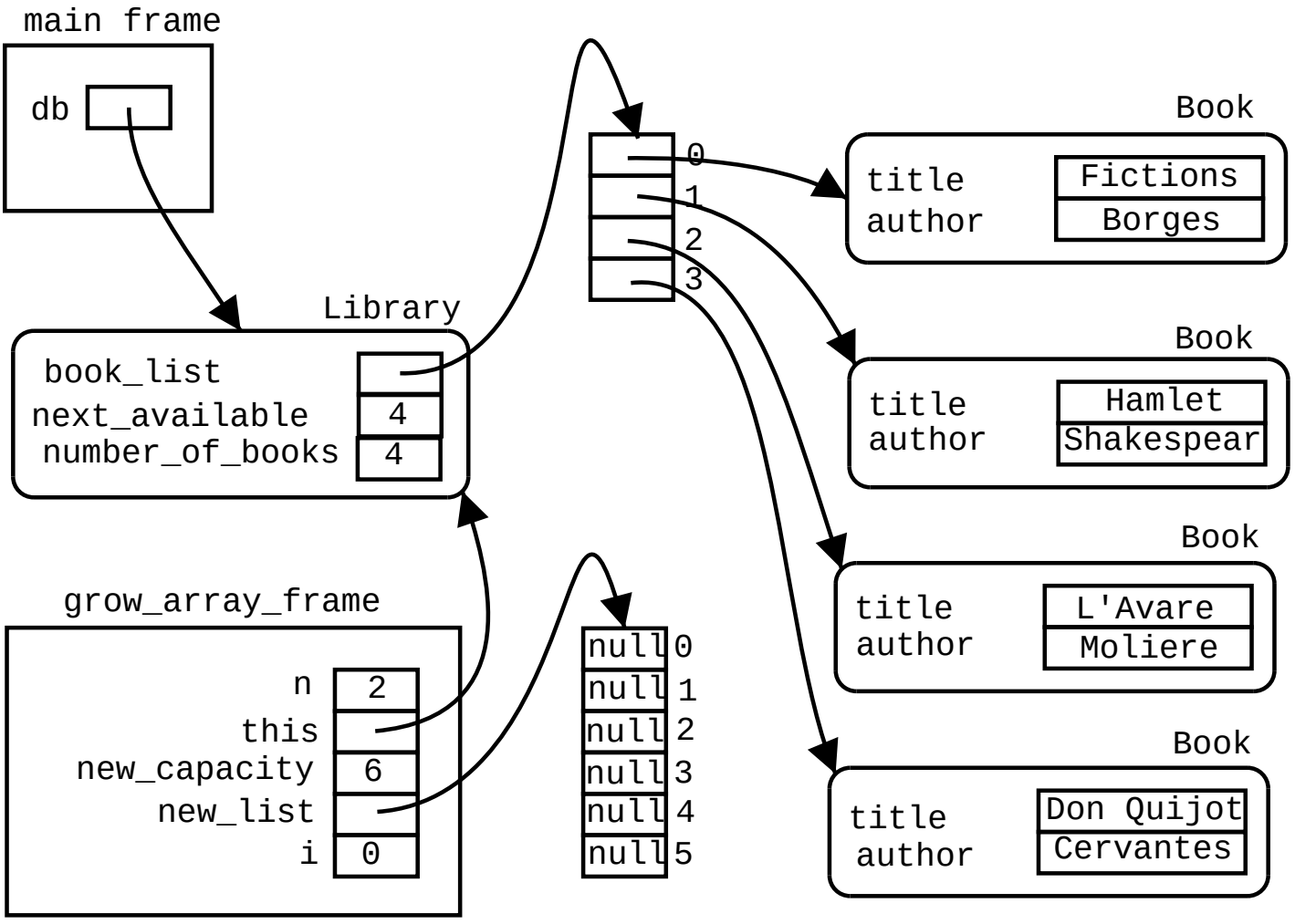
The method is private to ensure encapsulation so that only BookDatabase objects can grow the book lists.

Growing arrays

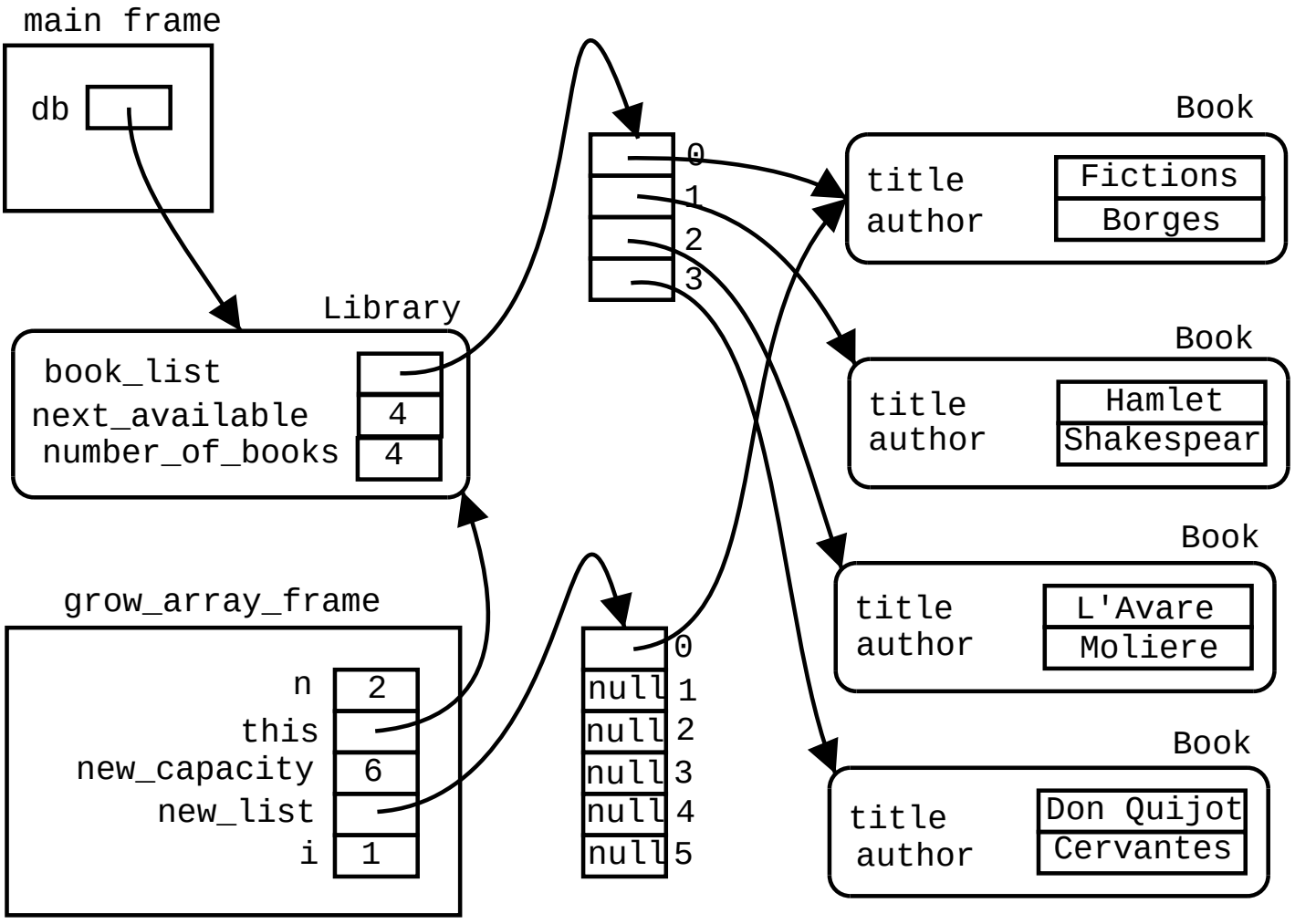


call grow_array(2)

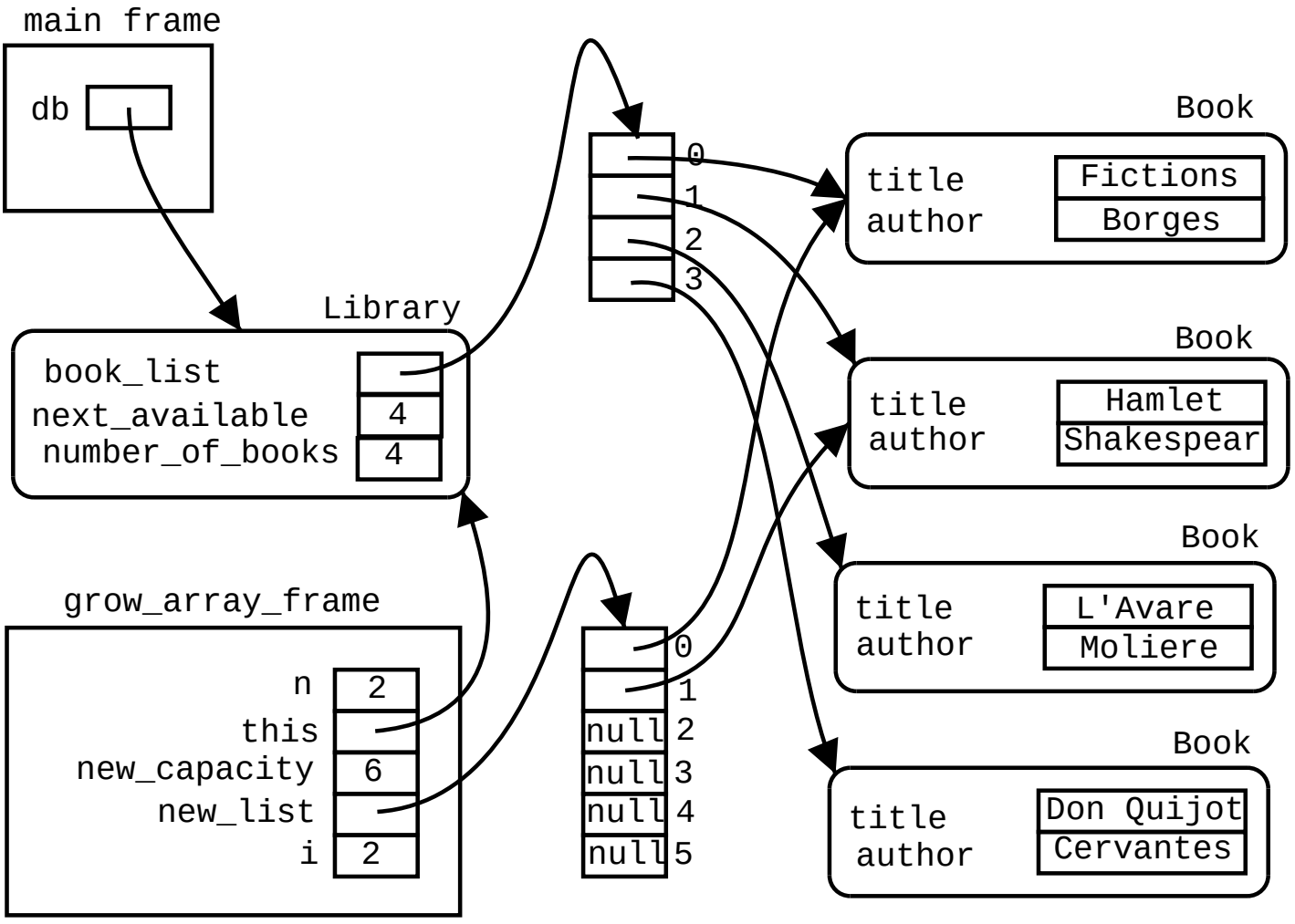
Growing arrays



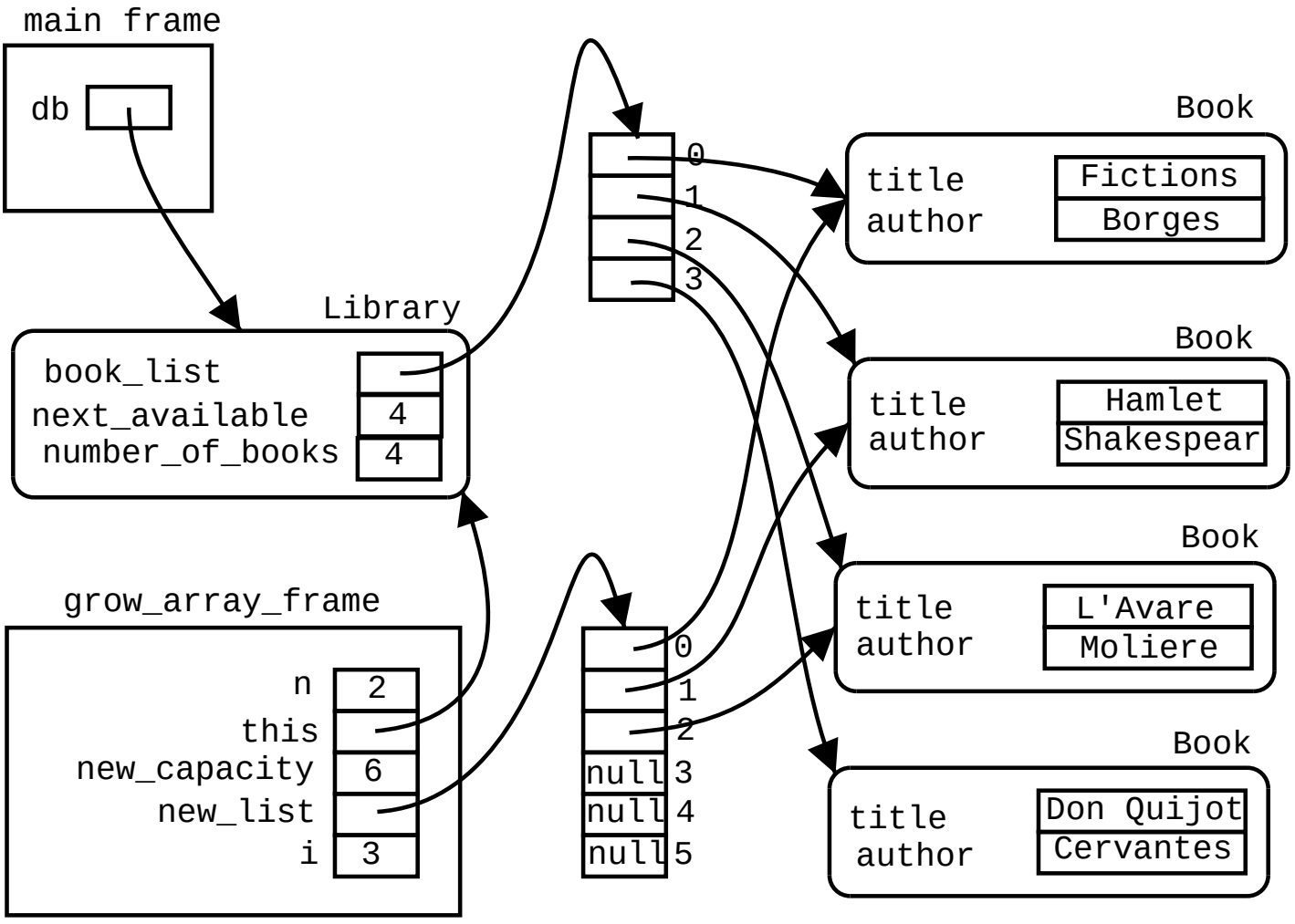
Growing arrays



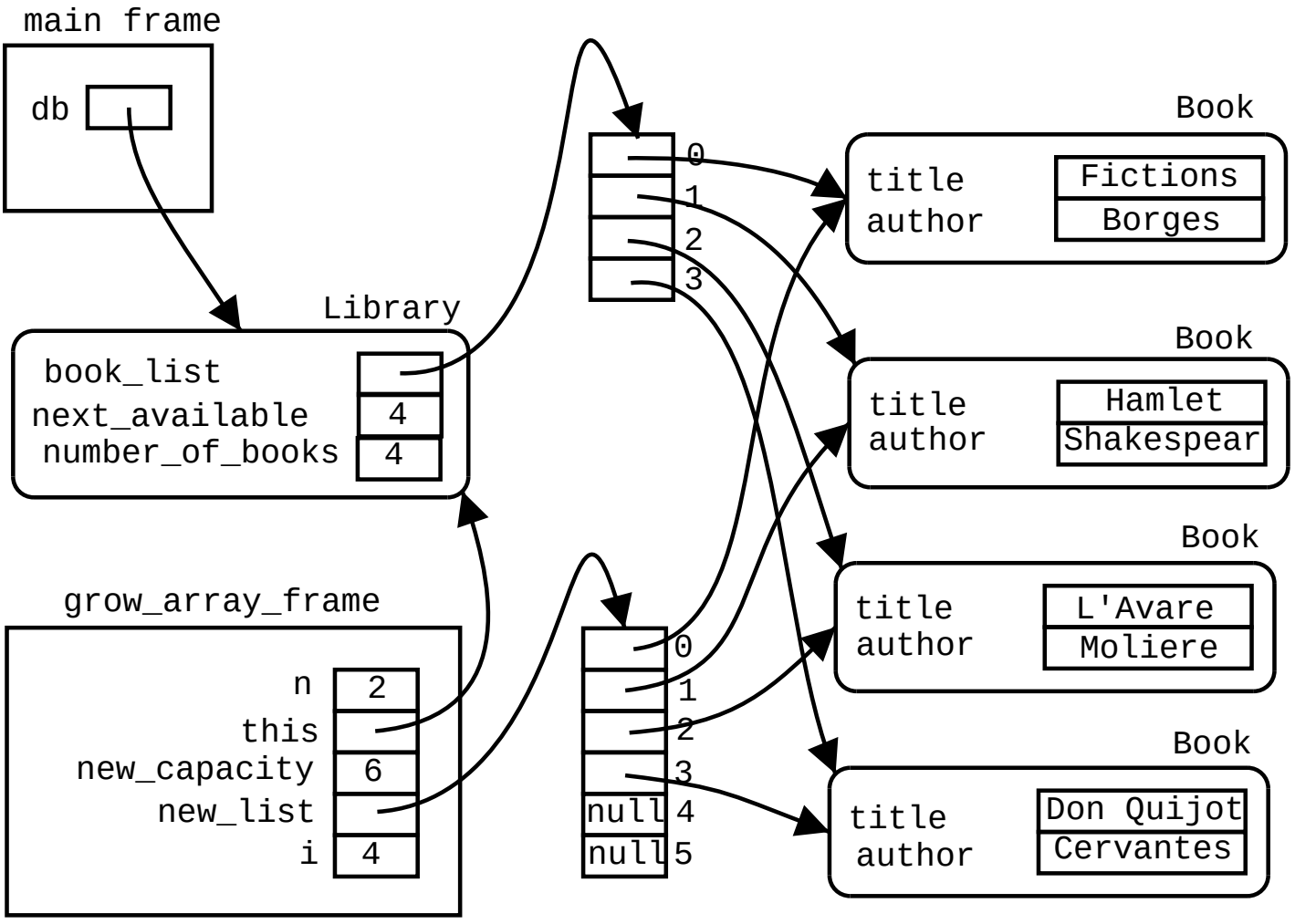
Growing arrays



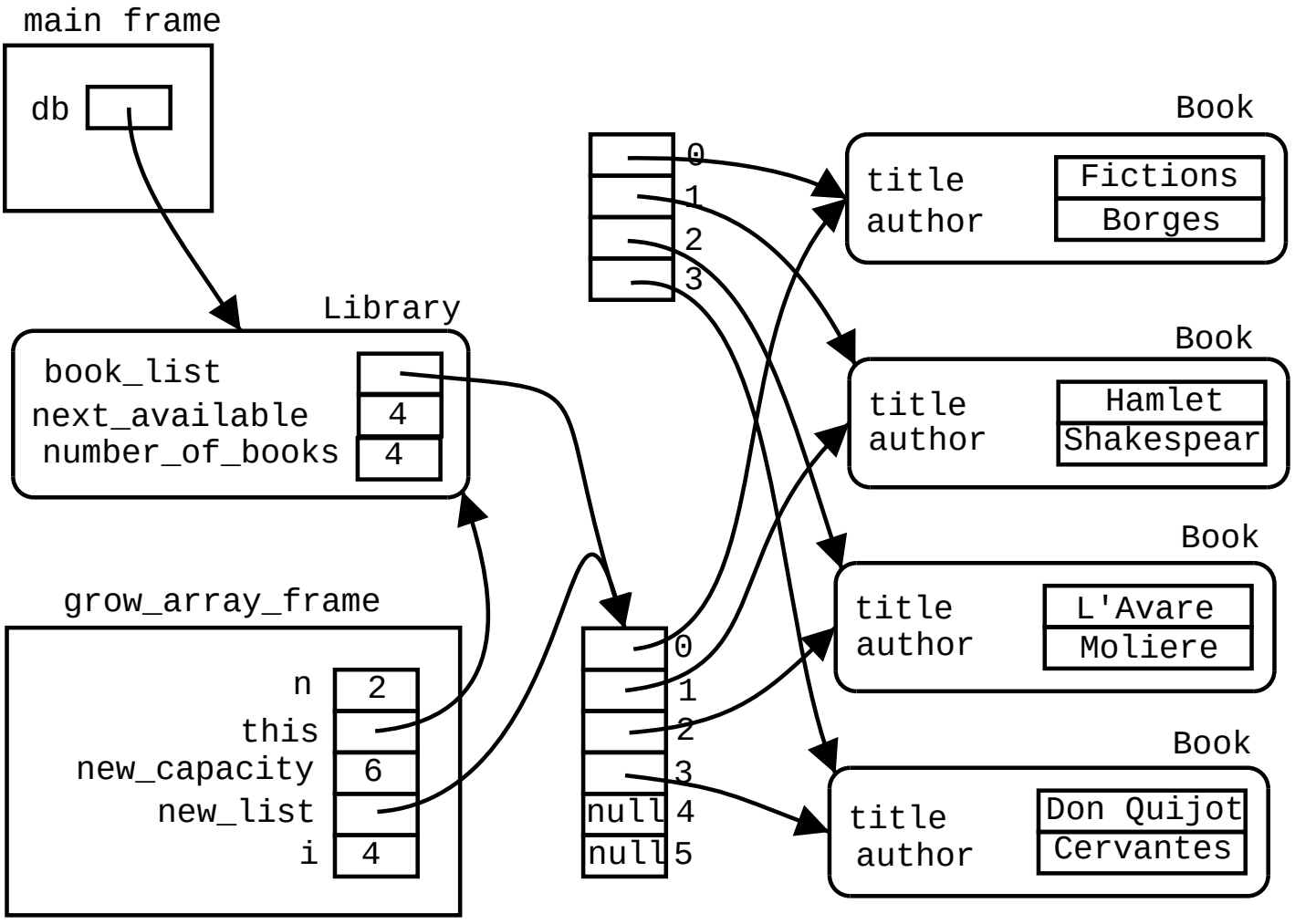
Growing arrays



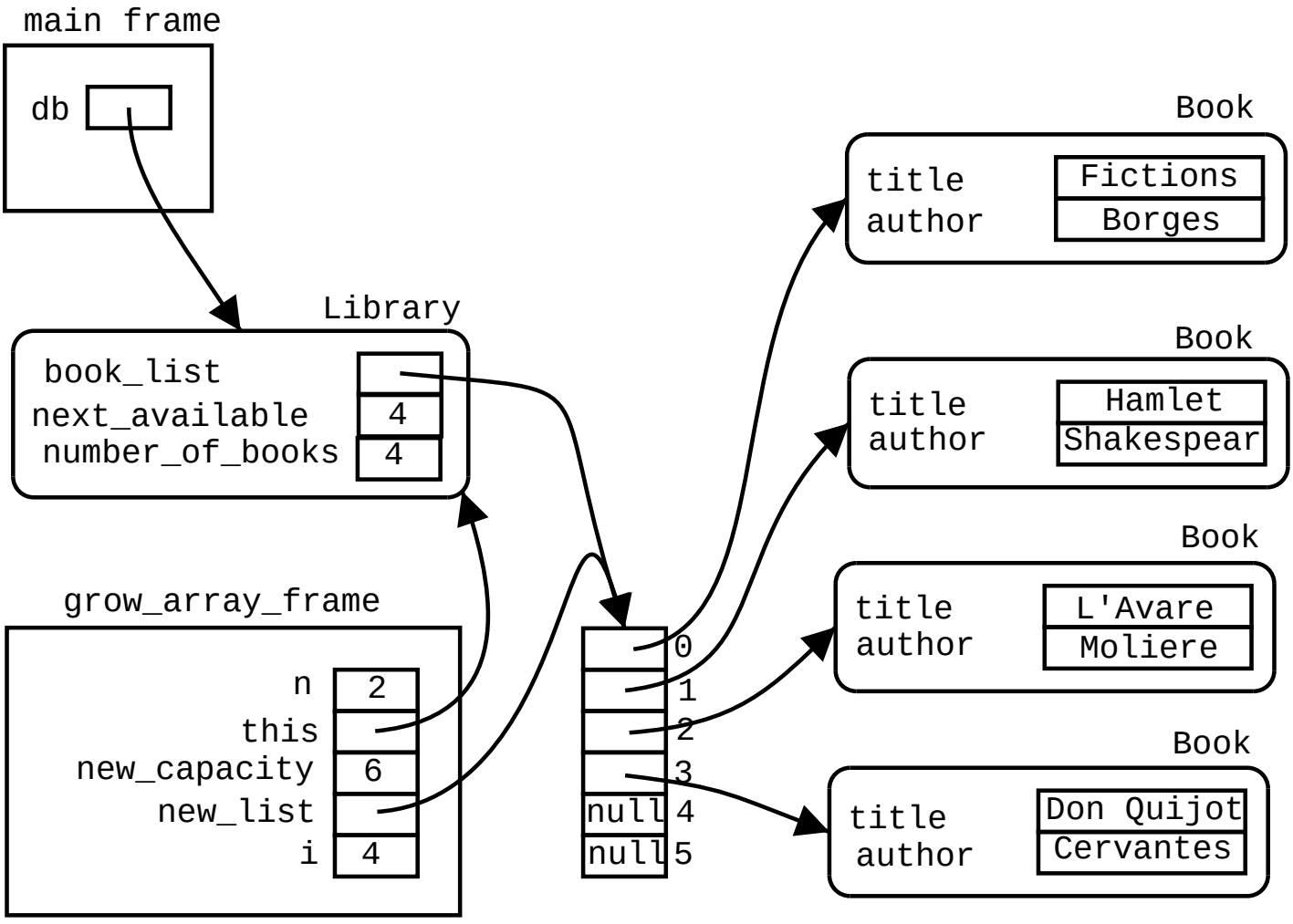
Growing arrays



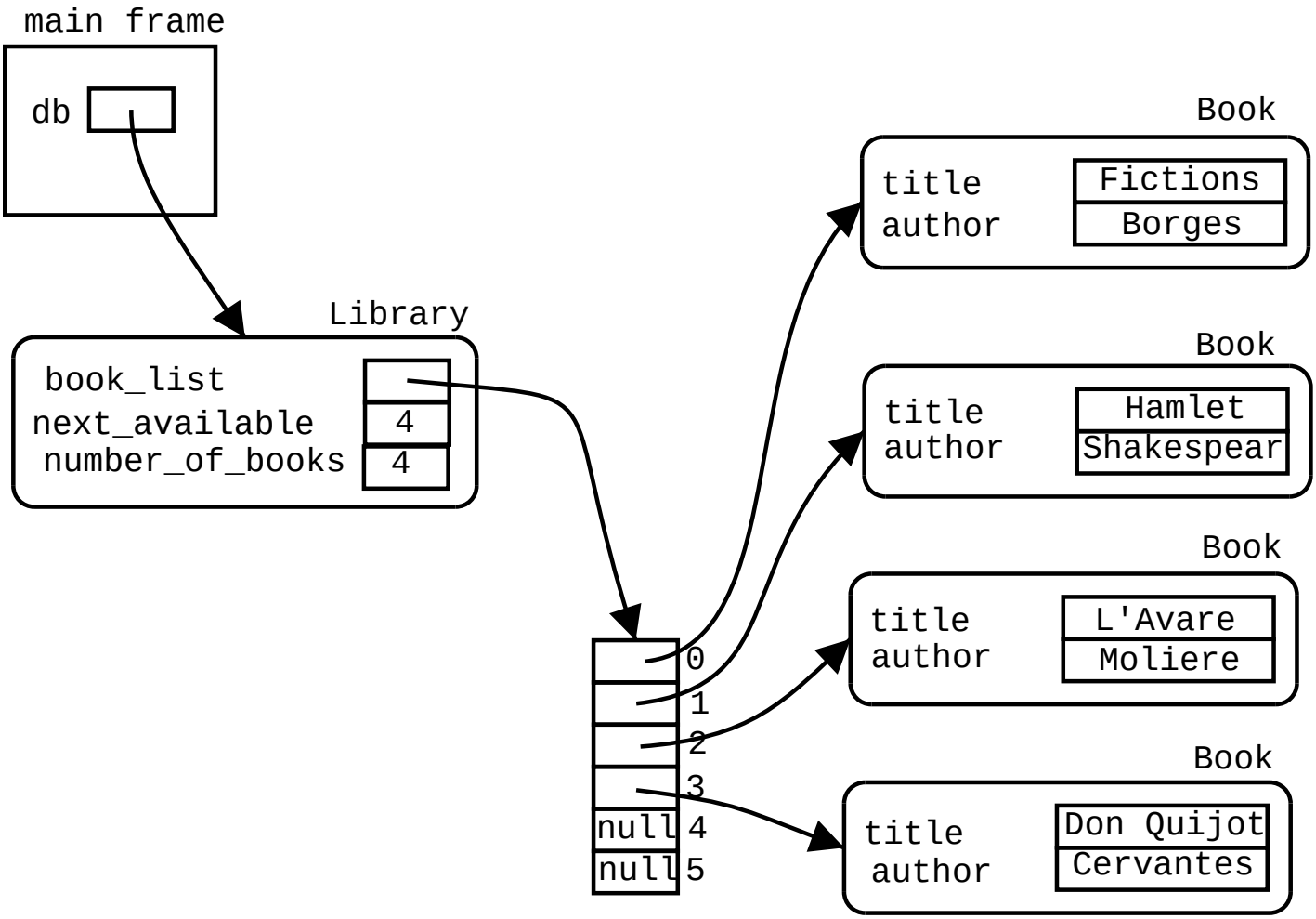
Growing arrays



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++;
}
```

The end