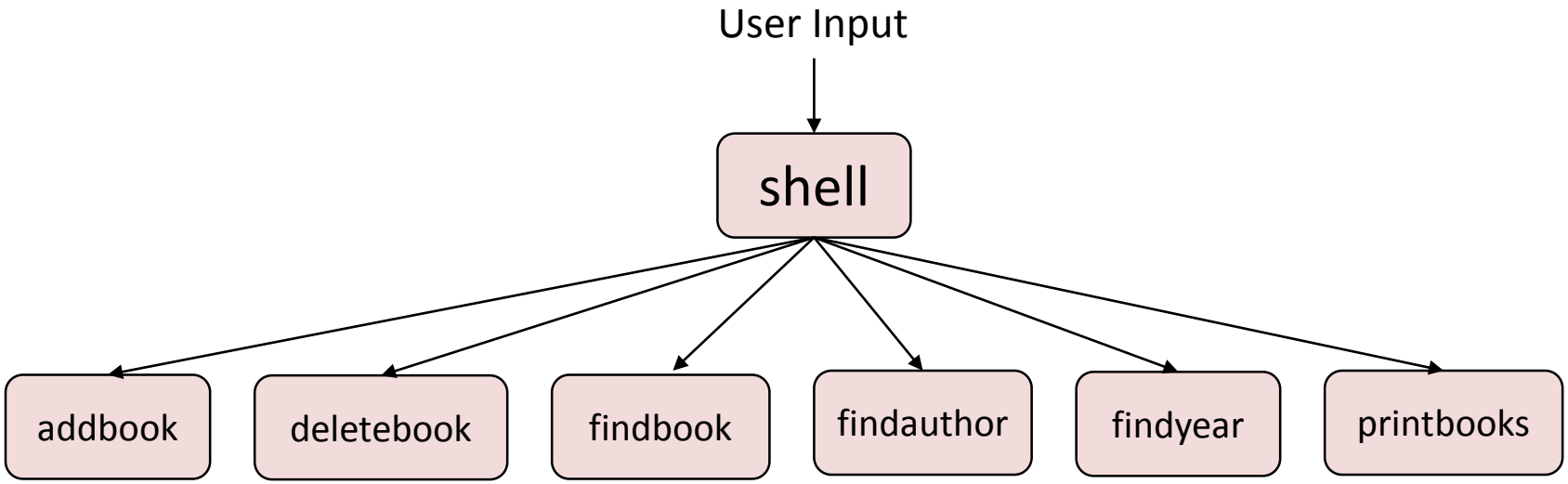
Software Structure

- Real programs have many **features** which interact
- Ex. Web browser
  1. Send requests to servers
  2. Receive responses from servers
  3. Present HTML documents on the screen
  4. Execute Javascript
- Common to implement **different functions** for each feature

Lab 3: Book Database

- Keep track of a set of books
- Each book has a **title**, **year**, and **author**
- Need a **shell** to accept user commands
- Commands are:
  - addbook, printbooks, deletebook, findbook, findtitle, findyear, quit

Functional Decomposition



- Each operation is a function, + shell

Organizing the Data

- Need to store a set of books
- Each book has a **title**, **year**, and **author**
- How should you store information about a book?
- List?
  - [“title1”, 1990, “Smith”] or [“title2”, 2000, “Doe”]
- Maybe tuples?

Storing a Single Book

- Named Tuple is good

```
>>> from collections import namedtuple
>>> Book = namedtuple('Book', 'title year author')
>>> b1 = Book('title1', 1990, 'Smith')
>>> b2 = Book('title2', 2000, 'Doe')
```

- Can refer to fields by name, not just index

```
>>> print(b1.title)
>>> 'title1'
```

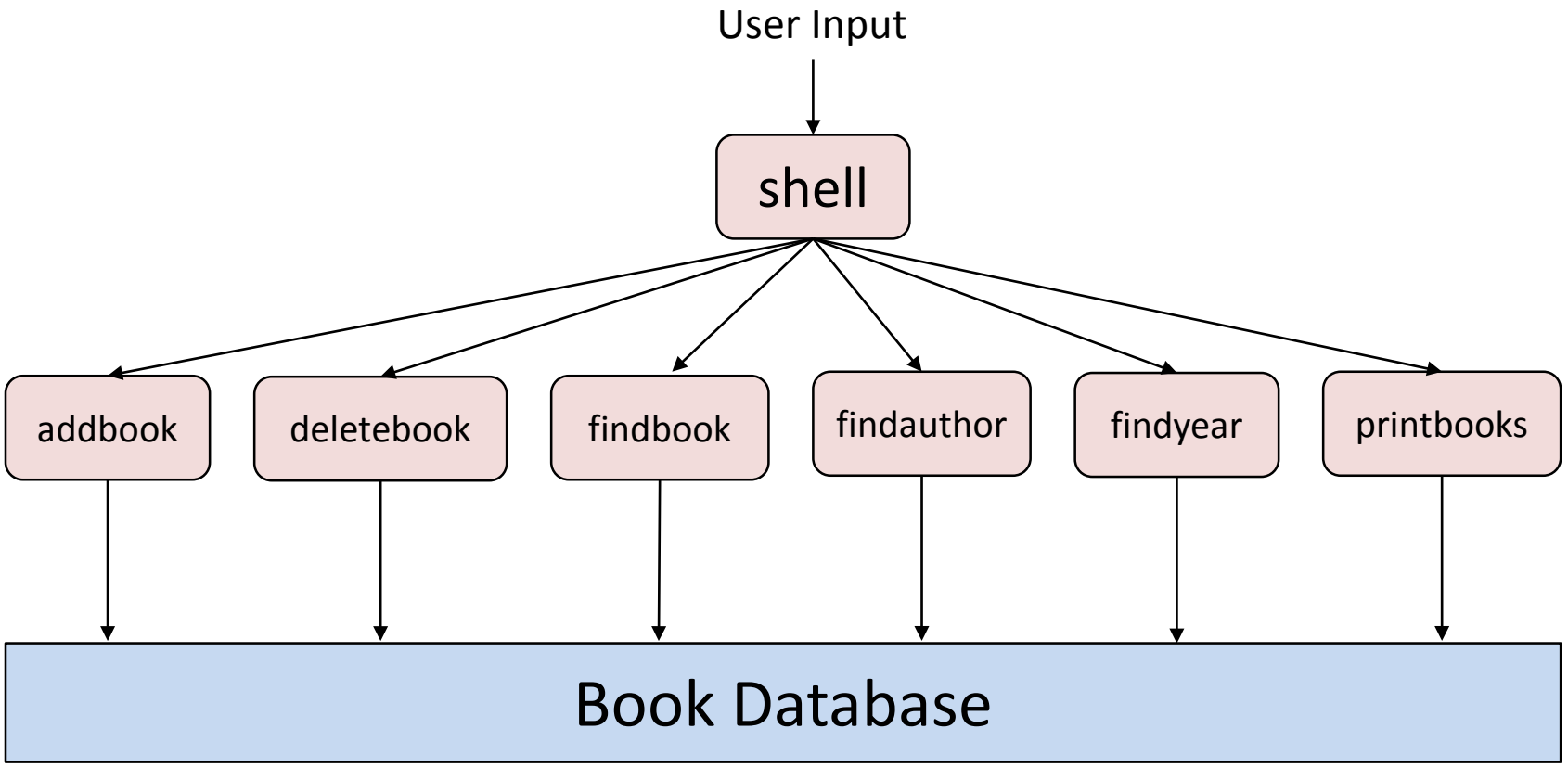
Storing a Book Database

- How do you store all of the books?
- List is the obvious choice

```
>>> dbase = [b1, b2]
```

- Can easily append, delete, search

Functions with Data



Implementation/Test Plan

- Divide and Conquer
  - Divide the software into a set of functions
  - Implement one function at a time
  - Test the function
  - Continue until done

Testing a Function

- Execute the function with appropriate inputs
  - Call it with test arguments
  - Enter test input interactively
- Check that the results are correct
  - View return value
  - View data printed on the screen

Testing the Shell

- Enter test data by entering commands with arguments

```
$ addbook title1 1990 Smith
$ deletebook title1
```

- How do you check the results if the other functions have not been tested yet?
- Design **stub functions** which just print the arguments

Stub Functions

```
def addbook(t, y, a):
    print(“AB: “ + t + “-” + str(y) + “-” + a)

$ addbook title1, 1990, Smith
AB: title1-1990-Smith
$ addbook title2, 2000, Doe
AB: title2-2000-Doe
```

Testing Other Functions

- Once the shell works, you can apply test input to other functions easily
- Need to **observe the results** of other functions
- Most functions affect the book database
- Need to **view the book database**
- Use **`printbooks`**

Function Testing Example

```
$ addbook title1, 1990, Smith
$ addbook title2, 2000, Doe
$ printbooks
title1, 1990, Smith
title2, 2000, Doe
$ deletebook title1
$ printbooks
title1, 1990, Smith
```