Data Structures and Algorithms (3)

Instructor: Ming Zhang
Textbook Authors: Ming Zhang, Tengjiao Wang and Haiyan Zhao
Higher Education Press, 2008.6 (the "Eleventh Five-Year" national planning textbook)
https://courses.edx.org/courses/PekingX/04830050x/2T2014/

Chapter 3 Stacks and Queues

- Stacks
- **Application of stacks**
  - **Implementation of Recursion using Stacks**
- Queues

Transformation from recursion to non-recursion

- The principle of recursive function
- Transformation of recursion
- The non recursive function after optimization

The principle of recursive function

Another study of recursion

- Factorial $f(n) = \begin{cases} n \times f(n-1) & n \geq 1 \\ 1 & n = 0 \end{cases}$
- **Exit of recursion**
  - End condition of recursion is when the minimal problem is solved
  - More than one exits are permitted
- **Rule of recursion**

  (Recursive body + bounded function)
  - Divide the original problem into sub problems
  - Ensure that the scale of recursion is more and more closer to the end condition

The principle of recursive function

Non recursive implementation of recursive algorithm

$$f(n) = \begin{cases} n \times f(n-1) & n \geq 1 \\ 1 & n = 0 \end{cases}$$

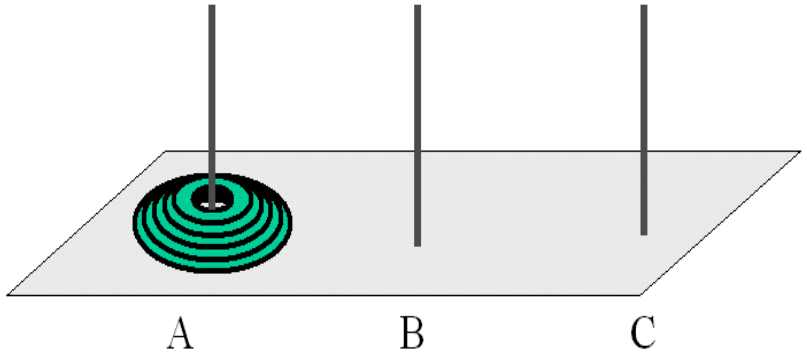
- Non recursive implementation of factorial
  - Establish iteration
  - Transformation from recursion to non-recursion

- How about the problem of Hanoi Tower?

The principle of recursive function

Recursion program for Hanoi tower problem

http://www.17yy.com/f/play/89425.html

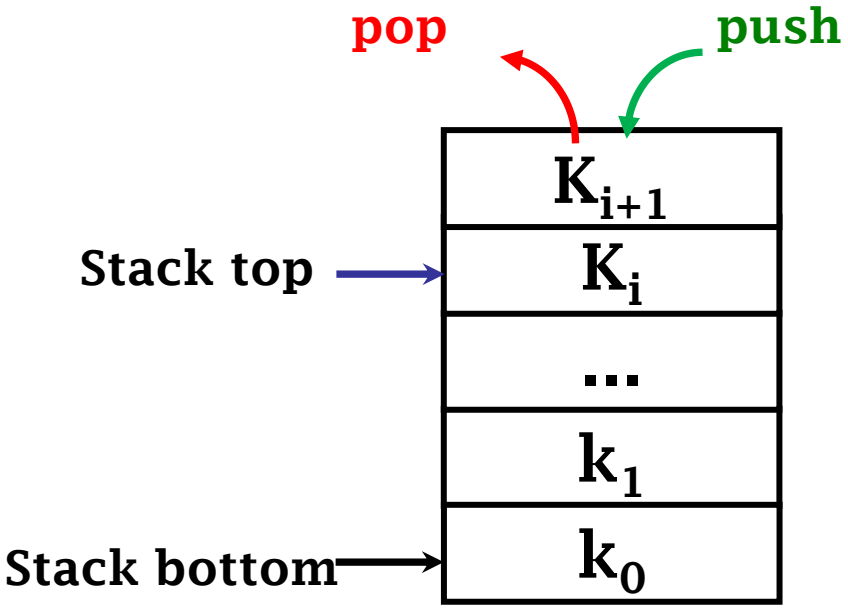


- hanoi(n,X,Y,Z)
  - Move n disk
  - Move the disk from pillar X to pillar Z
  - X、Y、Z can be used to place disks temporarily
    - Big disks cannot be put on small disks
- Such as hanoi(2, 'B', 'C' , 'A')
  - Move 2 disks from pillar B to pillar A

3.1.3 Transformation from recursion to non-recursion

```
void hanoi(int n, char X, char Y, char Z) {
    if (n <= 1)
        move(X,Z);
    else  {
// don’t move the largest disk on X and move the left n-1 disk to Y
        hanoi(n-1,X,Z,Y);
        move(X,Z);        //move the largest disk on X to Z
        hanoi(n-1,Y,X,Z); // move the n-1 disk on Y to Z
    }
}
void move(char X, char Y)
// move the disk on the top of pillar x to pillar Y
{
  cout  << "move"  <<  X  << "to" << Y  << endl;
}
```

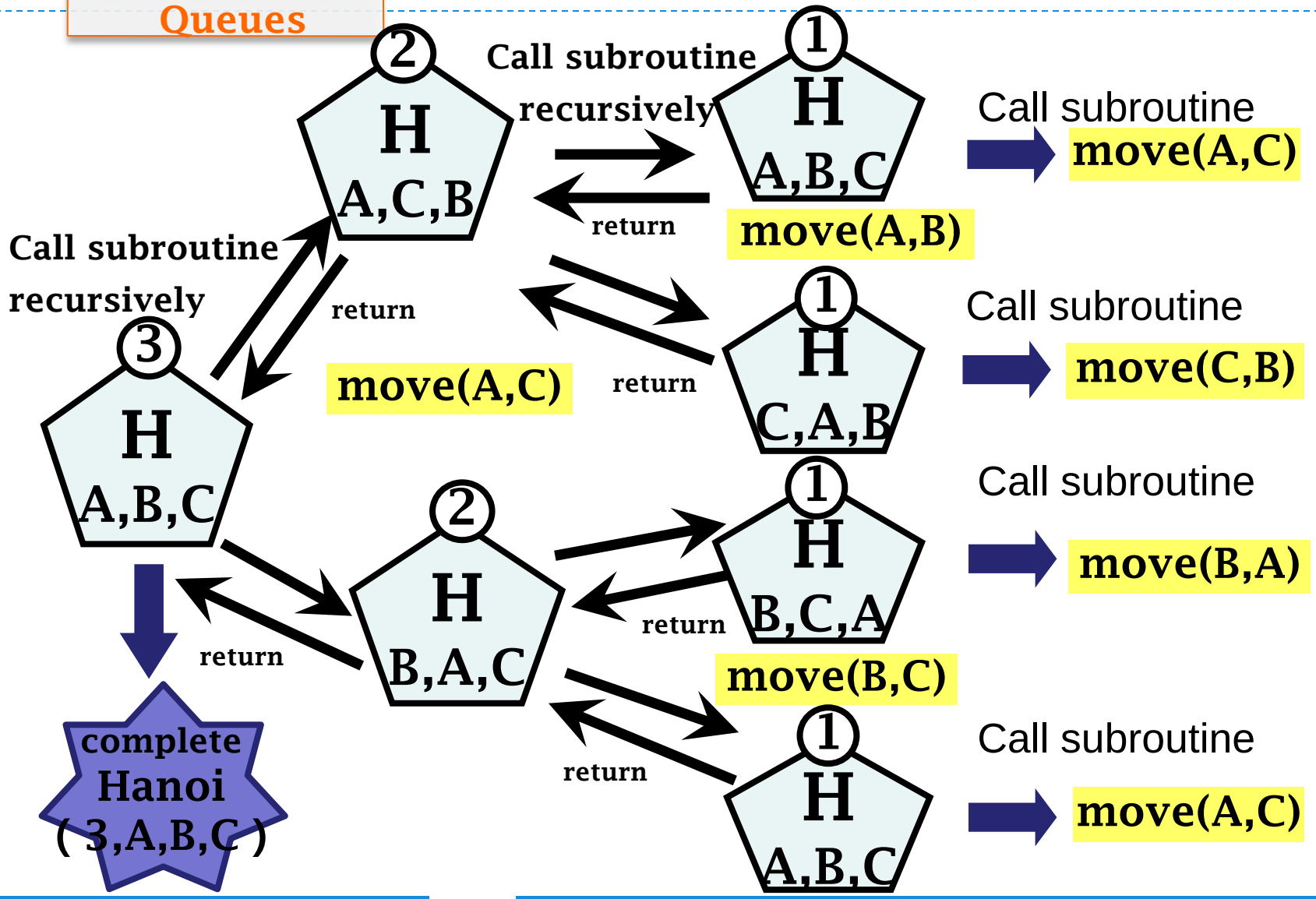
Operating diagram of Hanoi recursive subroutine

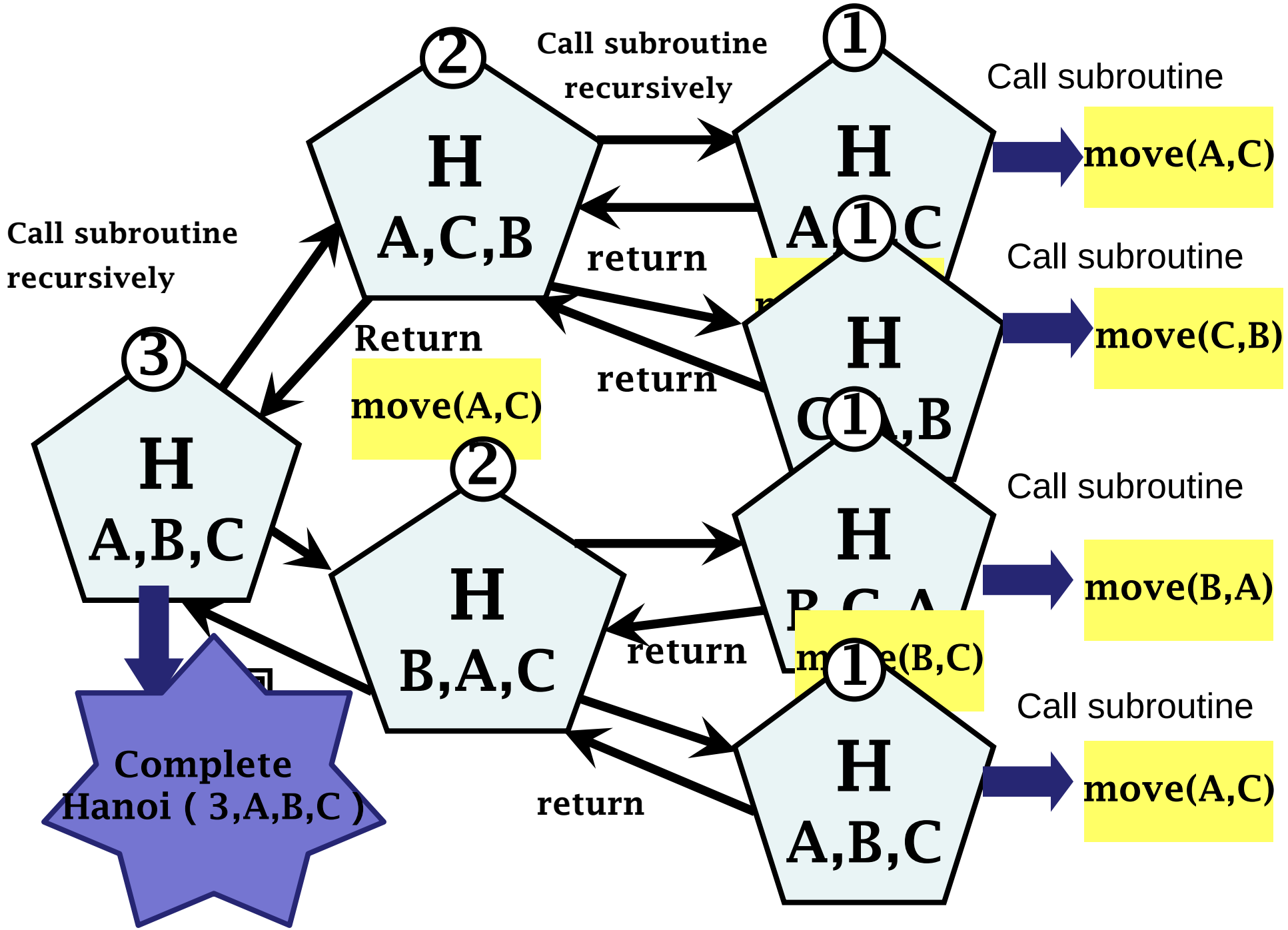


Execute the instructions of Hanoi program

Exchange information with subroutine via stack

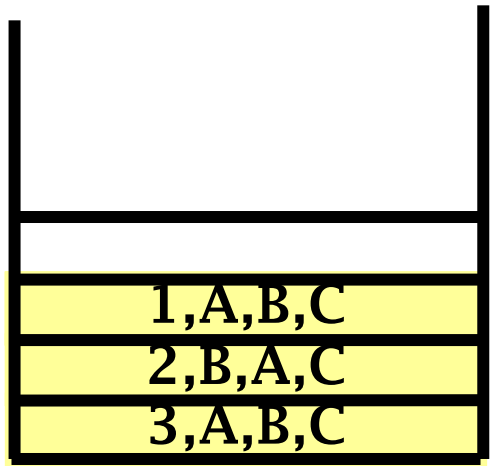
3.1.3 Transformation from recursion to non-recursion




Call subroutine recursively
Call subroutine recursively
Call subroutine
move(A,C)
Call subroutine
move(C,B)
Call subroutine
move(B,A)
Call subroutine
move(A,C)
Return
move(A,C)
return
return
return
return
return
Complete Hanoi (3,A,B,C)
H A,B,C
H A,C,B
H A,C,B
H C,A,B
H B,A,C
H B,C,A
H A,B,C
1
2
3
1
2
1
1

The status of stack when the recursion is executed

hanoi(1,A,B,C)
hanoi(1,B,C,A)
hanoi(2,B,A,C)
hanoi(1,C,A,B)
hanoi(1,A,B,C)
hanoi(2,A,C,B)
hanoi(3,A,B,C)



Perform move(A,C)

A recursive mathematical formula

$$fu(n) = \begin{cases} n+1 & when \quad n<2 \\ fu(\lfloor n/2 \rfloor) * fu(\lfloor n/4 \rfloor) & n \geq 2 \end{cases}$$

Example for recursive function

```
int f(int n) {
  if (n<2)
      return n+1;
  else
      return f(n/2) * f(n/4);
}
```

$$fu(n) = \begin{cases} n+1 & when \quad n<2 \\ fu(\lfloor n/2 \rfloor) * fu(\lfloor n/4 \rfloor) & n \geq 2 \end{cases}$$

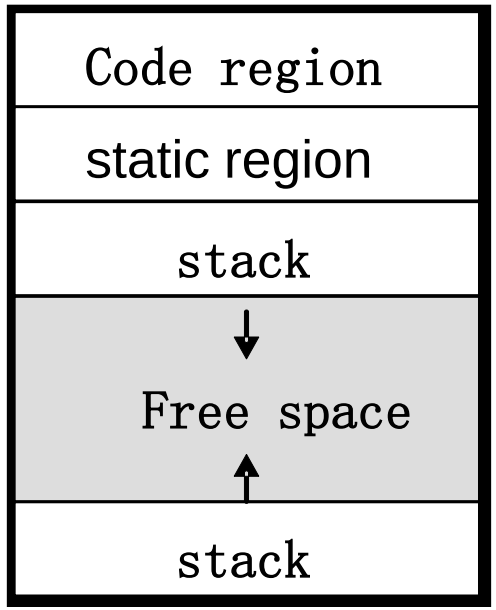
Example for recursive function(change a little)

```
void exmp(int n, int& f) {
    int u1, u2;
    if (n<2)
        f = n+1;
    else {
        exmp((int)(n/2), u1);
        exmp((int)(n/4), u2);
        f = u1*u2;
    }
}
```

$$fu(n) = \begin{cases} n+1 & when \quad n<2 \\ fu(\lfloor n/2 \rfloor) * fu(\lfloor n/4 \rfloor) & n \geq 2 \end{cases}$$

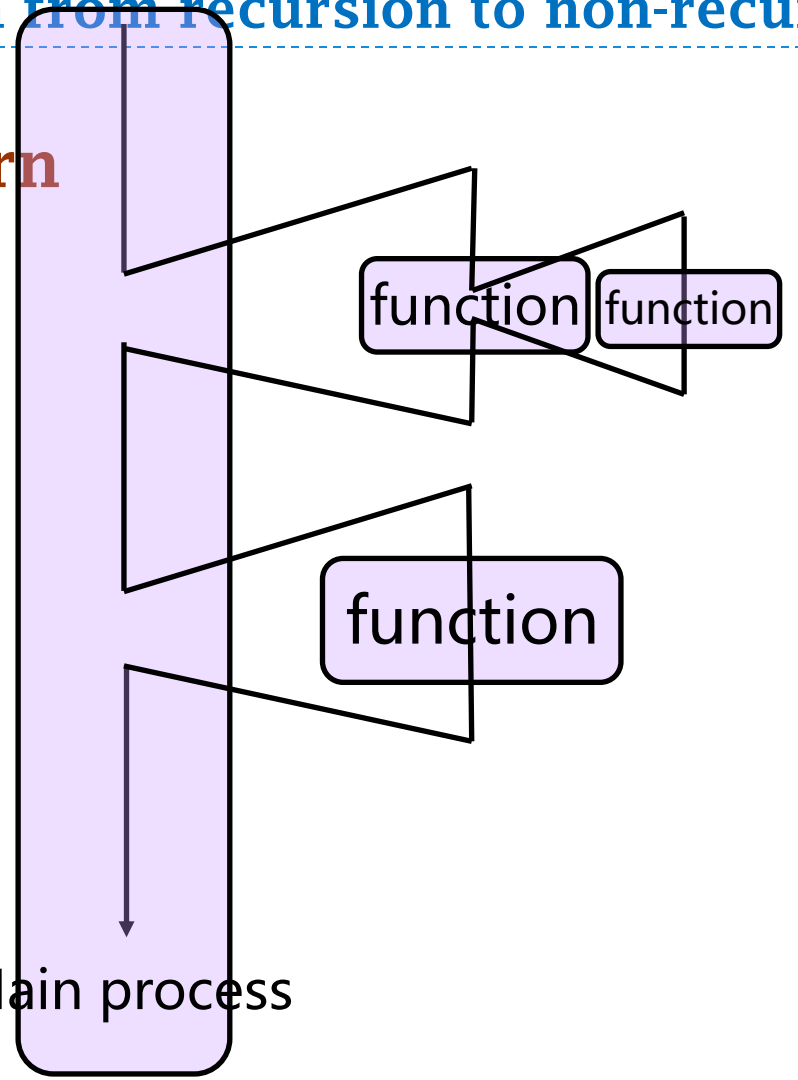
Dynamic memory allocation when the function is executed

- **Stack** is used for data that match last-in and first-out after allocated
  - Such as call function
- **Heap** is used for data which doesn't match LIFO
  - Such as the distribution of the space that the pointer points to



Function call and the steps of return

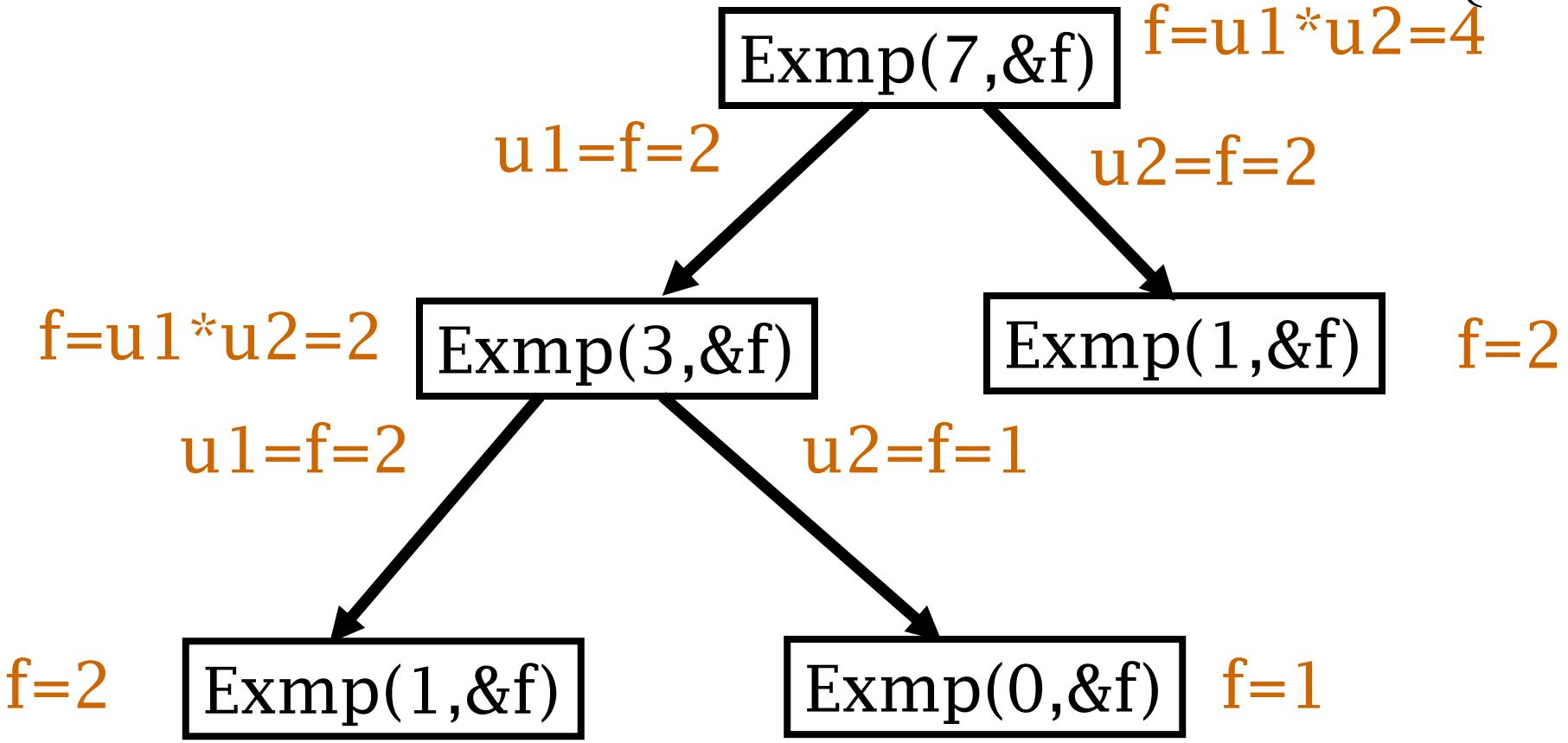
- **Function recall**
  - Save call information (parameter , return address)
  - Distribute data area (Local variable)
  - Control transfers to the exit of the function called
- **Return**
  - Save return information
  - Release data area
  - Control transfers to a superior function (the main call function)



3.1.3 Transformation from recursion to non-recursion

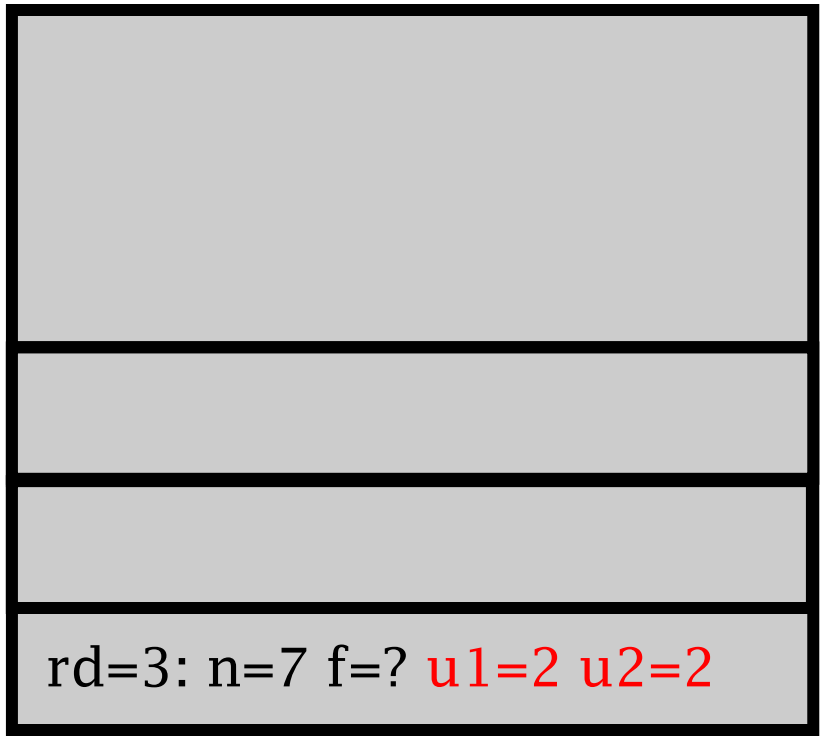
Diagram for the process of executing function

$$fu(n) = \begin{cases} n+1 & when \quad n<2 \\ fu(\lfloor n/2 \rfloor) * fu(\lfloor n/4 \rfloor) & n \geq 2 \end{cases}$$



Simulate the process of recursion call by stack

- Last call , first return (LIFO) , so stack is used




```
void exmp(int n, int& f) {
    int u1, u2;
    if (n<2) f = n+1;
    else {
        exmp((int)(n/2), u1);
        exmp((int)(n/4), u2);
        f = u1*u2;
    }
}
```

Question

- For following function , please draw the recursive tree when n=4 case, and use stack to simulate the process of recursive calls with the stack
  - The factorial function

    $f_0=1$, $f_1=1$, $f_n = n\ f_{n-1}$
  - 2 order Fibonacci function

    $f_0=0$, $f_1=1$, $f_n = f_{n-1}+ f_{n-2}$

Data Structures and Algorithms

Thanks

the National Elaborate Course (Only available for IPs in China)
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

Ming Zhang, Tengjiao Wang and Haiyan Zhao
Higher Education Press, 2008.6 (awarded as the "Eleventh Five-Year" national planning textbook)