

第7章 函数 (2)


S.IST@XMU

复习回顾

➤ 上次课的内容：

◆ 字符串应用举例

◆ 函数基本概念

◆ 如何定义函数

◆ 函数的调用过程

◆ 提醒：今晚恢复上机课

◆ 警告：今晚上机课时出现类似情况不许拔插头！

大一刚学C语言，第四次上机课，当我发现我照着书抄写的程序在运行之后的黑框里跳出一排
烫烫烫烫烫烫烫烫烫烫烫烫烫烫，
当时就震惊了。我真的以为这是电脑过热发出的警告，于是我弯下腰把插头拔了。。。

数组简单编程举例

➤ 如何比较两个数组是否完全相等

```
1. #include <stdio.h>

2. int main()
3. {
4.     int s[5]={1,2,3,4,5},k[5]={1,2,3,4,5};
5.     int match=1,i;
6.     for (i=0; i<5; i++)
7.     {
8.         if (s[i]!=k[i])
9.         {
10.             match = 0;
11.             break;
12.         }
13.     }
14.     printf((match==1)?"same\n":"different\n");
15.     return 0;
16. }
```

数组编程解析 (1)

➤ 2017/11/17 作业题1：统计字符串出现次数

```
1. #include <stdio.h>
2. #include <string.h>

3. int main()
4. {
5.     char s[201],k[201];
6.     int sn,kn,i,j,pos,match,times=0;
7.     gets(s);
8.     gets(k);
9.     sn = strlen(s);
10.    kn = strlen(k);

    .....

29.    printf("The word \"%s\" appears
        %d %s in the text", k, times,
        (times>1)?"times":"time");
30.    return 0;
31.}
```

```
11.    for (i=0; i<sn-kn; i++)
12.    {
13.        pos = i;
14.        match = 1;
15.        for (j=0; j<kn; j++)
16.        {
17.            if (s[pos+j]!=k[j])
18.            {
19.                match = 0;
20.                break;
21.            }
22.        }
23.        if (match == 1)
24.        {
25.            times++;
26.            i+=kn-1;
27.        }
28.    }
```

数组编程解析 (2)

➤ 2017/11/17 作业题2：高精度加法

```
1. #include <stdio.h>
2. #include <string.h>
3. int main()
4. {
5.     char a[101],b[101],ra[101],
        rb[101], rc[101];
6.     int an,bn,i,digit=0,min,max,tmp;
7.     gets(a);      gets(b);
8.     an = strlen(a); bn = strlen(b);
9.     for (i=0; i<an; i++) //把数字倒过来
10.         ra[an-1-i] = a[i]-'0';
11.     for (i=0; i<bn; i++)
12.         rb[bn-1-i] = b[i]-'0';
13.     min = (an>bn)?bn:an;
14.     max = (an>bn)?an:bn;
15.     for (i=0; i<max+1; i++)
16.         rc[i] = 0; //把rc清零
17.
18.     .....
19.     if (rc[max]!=0) //开始输出rc
20.         printf("%d", rc[max]);
21.     for (i=max-1; i>=0; i--)
22.         printf("%d", rc[i]);
23.     printf("\n");
24.     return 0;
25. }
```

```
17.     for (i=0; i<min; i++) { //计算前min位
18.         tmp = ra[i]+rb[i]+digit;
19.         if (tmp >= 10)
20.         {
21.             tmp -= 10;
22.             digit = 1;
23.         }
24.         else
25.             digit = 0;
26.         rc[i]=tmp;
27.     }
28.     if (an>min) //计算第min位到第max位
29.         for (i=min; i<max; i++) {
30.             rc[i] = ra[i]+digit;
31.             digit = (rc[i]>=10)?1:0;
32.             rc[i] %= 10;
33.         }
34.     else if (bn>min)
35.         for (i=min; i<max; i++) {
36.             rc[i] = rb[i]+digit;
37.             digit = (rc[i]>=10)?1:0;
38.             rc[i] %= 10;
39.         }
40.     rc[max] += digit; //计算rc的第max+1位
```

函数声明的例子：求和

- 输入两个实数，用一个函数求出它们之和
- 解题思路：
 - ◆ 用add函数实现。首先要定义add函数，它为float型，它应有两个参数，也应为float型
 - ◆ 分别编写add函数和main函数，它们组成一个源程序文件
 - ◆ main函数的位置在add函数之前
 - ◆ 要对add函数进行声明，声明可在main函数中。

函数声明的例子：求和

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    float add(float x, float y);
```

```
    float a,b,c;
```

```
    printf("Please enter a and b:");
```

```
    scanf("%f,%f",&a,&b);
```

```
    c=add(a,b);
```

```
    printf("sum is %f\n",c);
```

```
    return 0;
```

```
}
```

对add函数声明

求两个实数之和，
函数值也是实型

```
float add(float x,float y)
```

```
{
```

```
    float z;
```

```
    z=x+y;
```

```
    return(z);
```

```
}
```

函数声明的例子：求和

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    float add(float x, float y);
```

```
    float a,b,c;
```

```
    printf("Please enter a and b:");
```

```
    scanf("%f,%f",&a,&b);
```

```
    c=add(a,b);
```

```
    printf("sum is %f\n",c);
```

```
    return 0;
```

```
}
```

只差一个分号

```
float add(float x,float y)
```

```
{
```

```
    float z;
```

```
    z=x+y;
```

```
    return(z);
```

```
}
```


函数声明的例子：求和

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    float add(float x, float y);
```

```
    float a,b,c;
```

```
    printf("Please enter a and b:");
```

```
    scanf("%f,%f",&a,&b);
```

```
    c=add(a,b);
```

```
    printf("sum is %f\n",c);
```

```
    return 0;
```

```
}
```

```
Please enter a and b:3.6,6.5  
sum is 10.100000
```

调用add函数

定义add函数

```
float add(float x,float y)
```

```
{
```

```
    float z;
```

```
    z=x+y;
```

```
    return(z);
```

```
}
```

函数原型

- 函数原型的一般形式有两种：

如 `float add(float x, float y);`

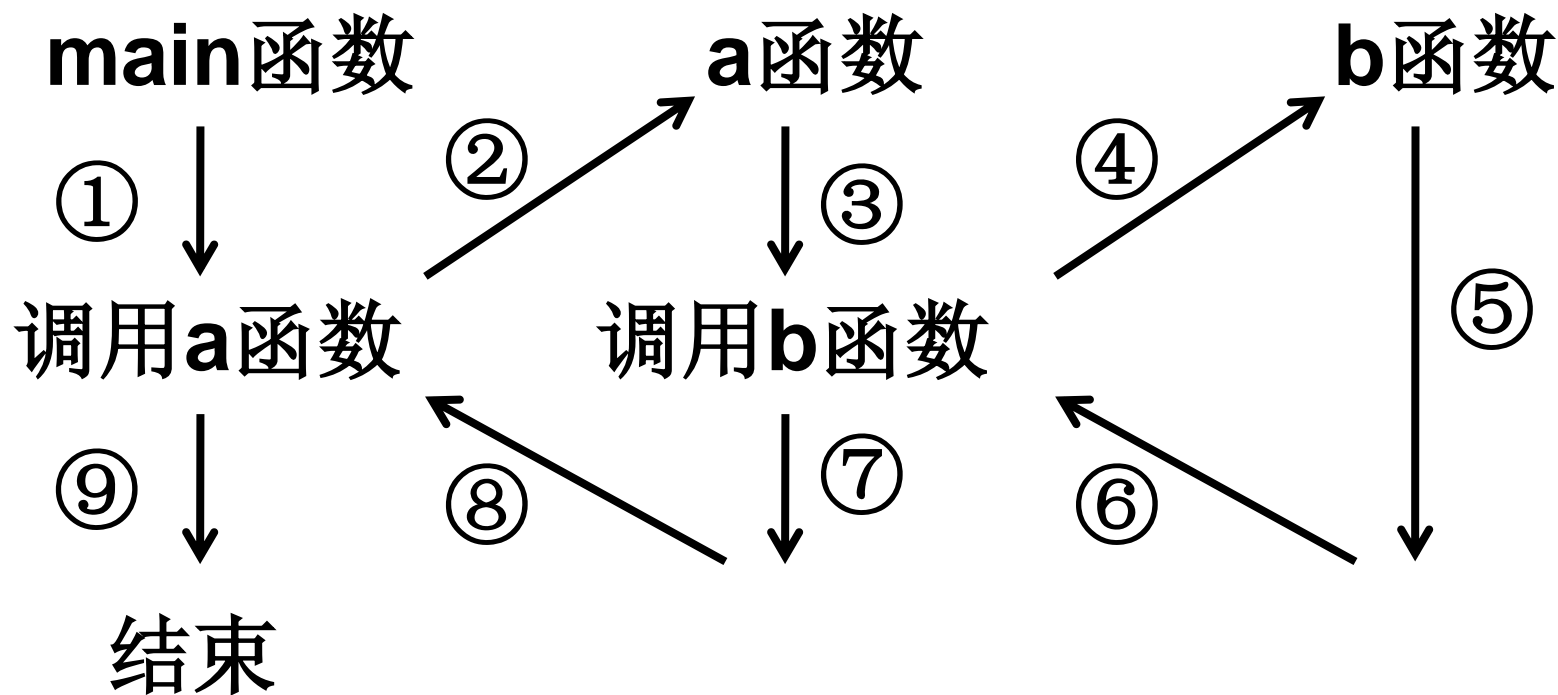
`float add(float, float);`

- **注意：**形式参数声明时前面的数据类型不能省略！
- 原型说明可以放在文件的开头，这时该文件所有函数都可以使用此函数

函数的嵌套调用

- C语言的函数定义是互相平行、独立的，函数不能嵌套定义，即**函数内部不能定义函数**！
- **但可以嵌套调用函数**，即调用一个函数的过程中，又可以调用另一个函数

函数的嵌套调用过程



函数嵌套调用的例子：最大数

输入4个整数，找出其中最大的数。用函数的嵌套调用来处理。

➤ 解题思路：

- ◆ main中调用max4函数，找4个数中最大者
- ◆ max4中再调用max2，找两个数中的大者
- ◆ max4中多次调用max2，可找4个数中的大者，然后把它作为函数值返回main函数
- ◆ main函数中输出结果

函数嵌套调用的例子：最大数

主函数部分

```
#include <stdio.h>
int main()
{
    int max4(int a,int b,int c,int d);
    int a,b,c,d,max;
    printf("4 interger numbers:");
    scanf("%d%d%d%d",&a,&b,&c,&d);
    max = max4(a,b,c,d);
    printf("max=%d \n",max);
    return 0;
}
```

对max4 函数声明

函数嵌套调用的例子：最大数

主函数部分

```
#include <stdio.h>
int main()
{
    int max4(int a,int b,int c,int d);
    int a,b,c,d,max;
    printf("4 interger numbers:");
    scanf("%d%d%d%d",&a,&b,&c,&d);
    max = max4(a,b,c,d);
    printf("max=%d \n",max);
    return 0;
}
```

输入4个整数

函数嵌套调用的例子：最大数

主函数部分

```
#include <stdio.h>
int main()
{
    int max4(int a,int b,int c,int d);
    int a,b,c,d,max;
    printf("4 interger numbers:");
    scanf("%d%d%d%d",&a,&b,&c,&d);
    max = max4(a,b,c,d);
    printf("max=%d \n",max);
    return 0;
}
```

调用后肯定是4
个数中最大者

输出最大者

函数嵌套调用的例子：最大数

max4函数部分

```
int max4(int a,int b,int c,int d)
{
    int max2(int a,int b);
    int m;

    m = max2(a,b);
    m = max2(m,c);
    m = max2(m,d);
    return(m);
}
```

对max2 函数声明

函数嵌套调用的例子：最大数

max4函数部分

```
int max4(int a,int b,int c,int d)
{
    int max2(int a,int b);
    int m;

    m = max2(a,b);
    m = max2(m,c);
    m = max2(m,d);
    return(m);
}
```

a,b中较大者

a,b,c中较大者

a,b,c,d中最大者

函数嵌套调用的例子：最大数

max4函数部分

```
int max4(int a,int b,int c,int d)
{
    int max2(int a,int b);
    int m;

    m = max2(a,b);
    m = max2(m,c);
    m = max2(m,d);
    return(m);
}
```

找a,b中较大者

max2函数

```
int max2(int a,int b)
{
    if (a>=b)
        return a;
    else
        return b;
}
```

函数嵌套调用的例子：最大数

max4函数部分

```
int max4(int a,int b,int c,int d)
{
    int max2(int a,int b);
    int m;

    m = max2(a,b);
    m = max2(m,c);
    m = max2(m,d);
    return(m);
}
```

return(a>b?a:b);

代码可否简化?

max2函数

```
int max2(int a,int b)
{
    if (a>=b)
        return a;
    else
        return b;
}
```

函数嵌套调用的例子：最大数

max4函数部分

```
int max4(int a,int b,int c,int d)
{
    int max2(int a,int b);
    int m;

    m = max2(a,b);
    m = max2(m,c);
    m = max2(m,d);
    return(m);
}
```

m=max2(max2(a,b),c);

max2函数

```
int max2(int a,int b)
{
    return(a>b?a:b);
}
```

函数嵌套调用的例子：最大数

max4函数部分

```
int max4(int a,int b,int c,int d)
{
    int max2(int a,int b);
    int m;
    m = max2(a,b);
    m = max2(m,c);
    m = max2(m,d);
    return(m);
}
```

m=max2(max2(max2(a,b),c),d);

max2函数

```
int max2(int a,int b)
{
    return(a>b?a:b);
}
```

函数嵌套调用的例子：最大数

max4函数部分

```
int max4(int a,int b,int c,int d)
{
    int max2(int a,int b);
    int m;
    return max2(max2(max2(a,b),c),d);
```

```
    m = max2(a,b);
    m = max2(m,c);
    m = max2(m,d);
    return m;
}
```

max2函数

```
int max2(int a,int b)
{
    return (a>b?a:b);
}
```

函数嵌套调用的例子：最大数

```
#include <stdio.h>
```

```
int main()
```

```
{ .....
```

```
max=max4(a,b,c,d);
```

```
.....
```

```
}
```

```
int max4(int a,int b,int c,int d)
```

```
{ int max2(int a,int b);
```

```
return max2(max2(max2(a,b),c),d);
```

```
}
```

```
int max2(int a,int b)
```

```
{
```

```
return(a>b?a:b);
```

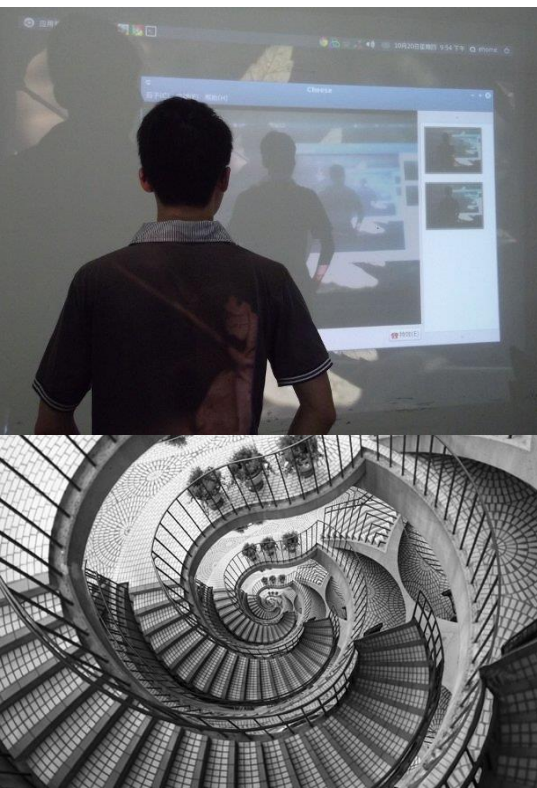
```
}
```

```
4 integer numbers:12 45 -6 89
max=89
```


递归的概念

- **标准定义**：若一个对象部分地**包含它自己**，或用它自己给自己定义，则称这个对象是递归的，若一个过程直接或间接地**调用自己**，则称这个过程是递归的过程。
- **生活中的递归**：
 - ◆ 从前有座山，山里有座庙，庙里有个老和尚，老和尚给小和尚讲了一个古老的故事：从前有座山，山里有座庙。。。
 - ◆ 如果一个屏幕正显示着摄像机拍到的内容，那么用摄像机拍摄这个屏幕我们将看到什么？

递归的图像



递归的无限和有限

➤ 无限递归

- ◆ 没有循环中止条件的递归
- ◆ 迷惑于计算机创造的虚拟空间了怎么办？两片镜子就可以帮你分辨虚拟和现实。

➤ 有限递归

- ◆ 递归到某一步就停止不再递归
- ◆ 偶数的递归定义：“0是偶数，一个偶数加上2还是偶数。”

函数的递归调用

- 在调用一个函数的过程中又出现直接或间接地调用该函数本身，称为函数的**递归调用**。
- 允许函数的递归调用是C语言的特点之一。
- **递归要诀：**

想要理解递归，你首先要理解递归！

直接/间接递归调用实例

```
int f(int x)
```

```
{
```

直接调用本函数

```
int y,z;
```

```
z=f(y);
```

```
return (2*z);
```

```
}
```

间接调用本函数

f函数

调用f函数

f1函数

f2函数

应使用if语句控制结束调用

调用f2函数 调用f1函数

递归的简单应用：年龄

➤ 有5个学生坐在一起

- ◆ 问第5个学生多少岁？他说比第4个学生大2岁
- ◆ 问第4个学生岁数，他说比第3个学生大2岁
- ◆ 问第3个学生，又说比第2个学生大2岁
- ◆ 问第2个学生，说比第1个学生大2岁
- ◆ 最后问第1个学生， he说是10岁
- ◆ 请问第5个学生多大

递归的简单应用：年龄

➤ 解题思路：

$$\text{age}(5) = \text{age}(4) + 2$$

$$\text{age}(4) = \text{age}(3) + 2$$

$$\text{age}(3) = \text{age}(2) + 2$$

$$\text{age}(2) = \text{age}(1) + 2$$

$$\text{age}(1) = 10$$

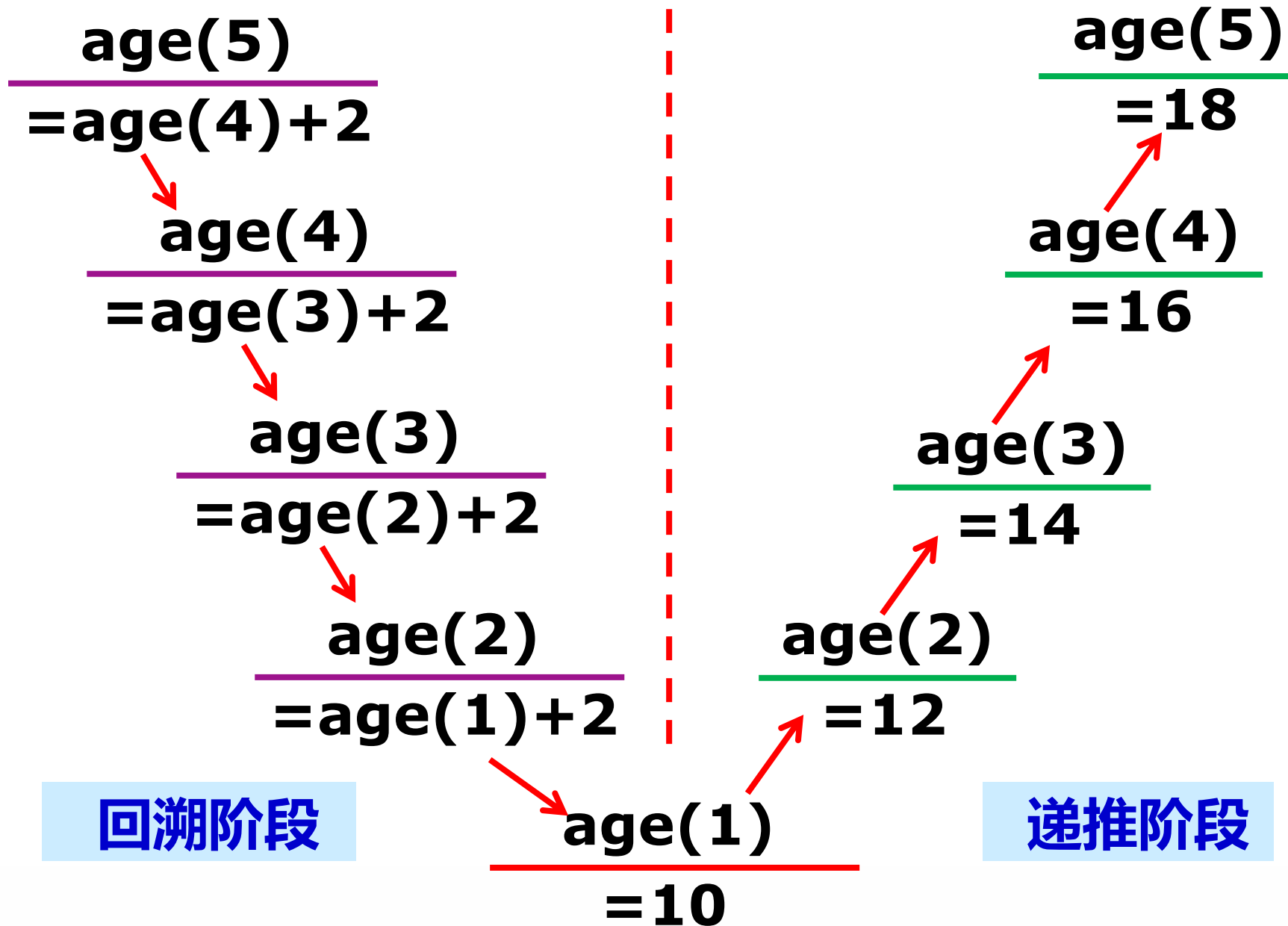
$$\text{age}(n) = 10 \quad (n = 1)$$

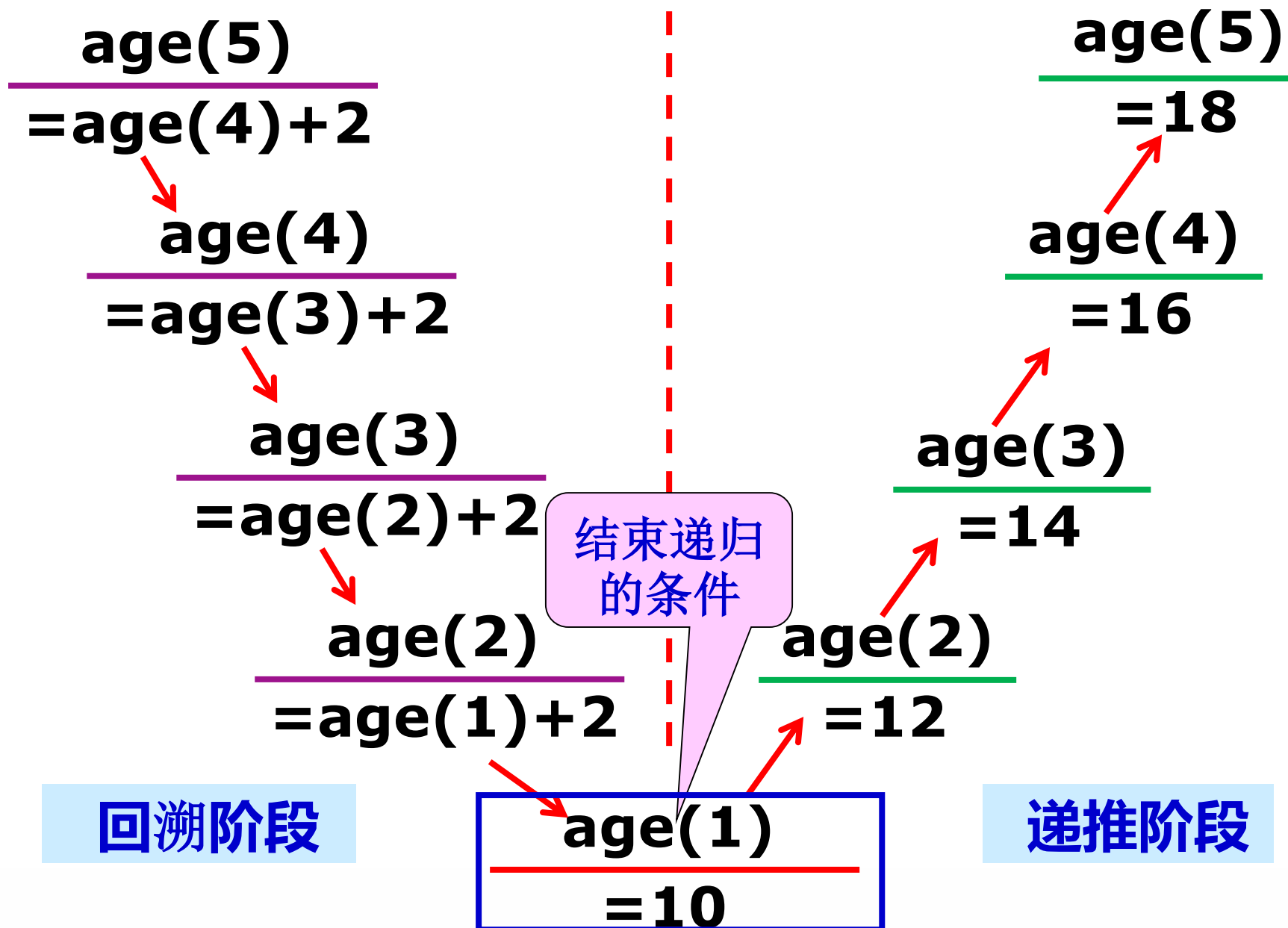
$$\text{age}(n) = \text{age}(n - 1) + 2 \quad (n > 1)$$

递归的简单应用：年龄

➤ 解题思路：

- ◆ 要求第 5 个年龄，就必须先知道第 4 个年龄
- ◆ 要求第 4 个年龄必须先知道第 3 个年龄
- ◆ 第 3 个年龄又取决于第 2 个年龄
- ◆ 第 2 个年龄取决于第 1 个年龄
- ◆ 每个学生年龄都比其前 1 个学生的年龄大 2

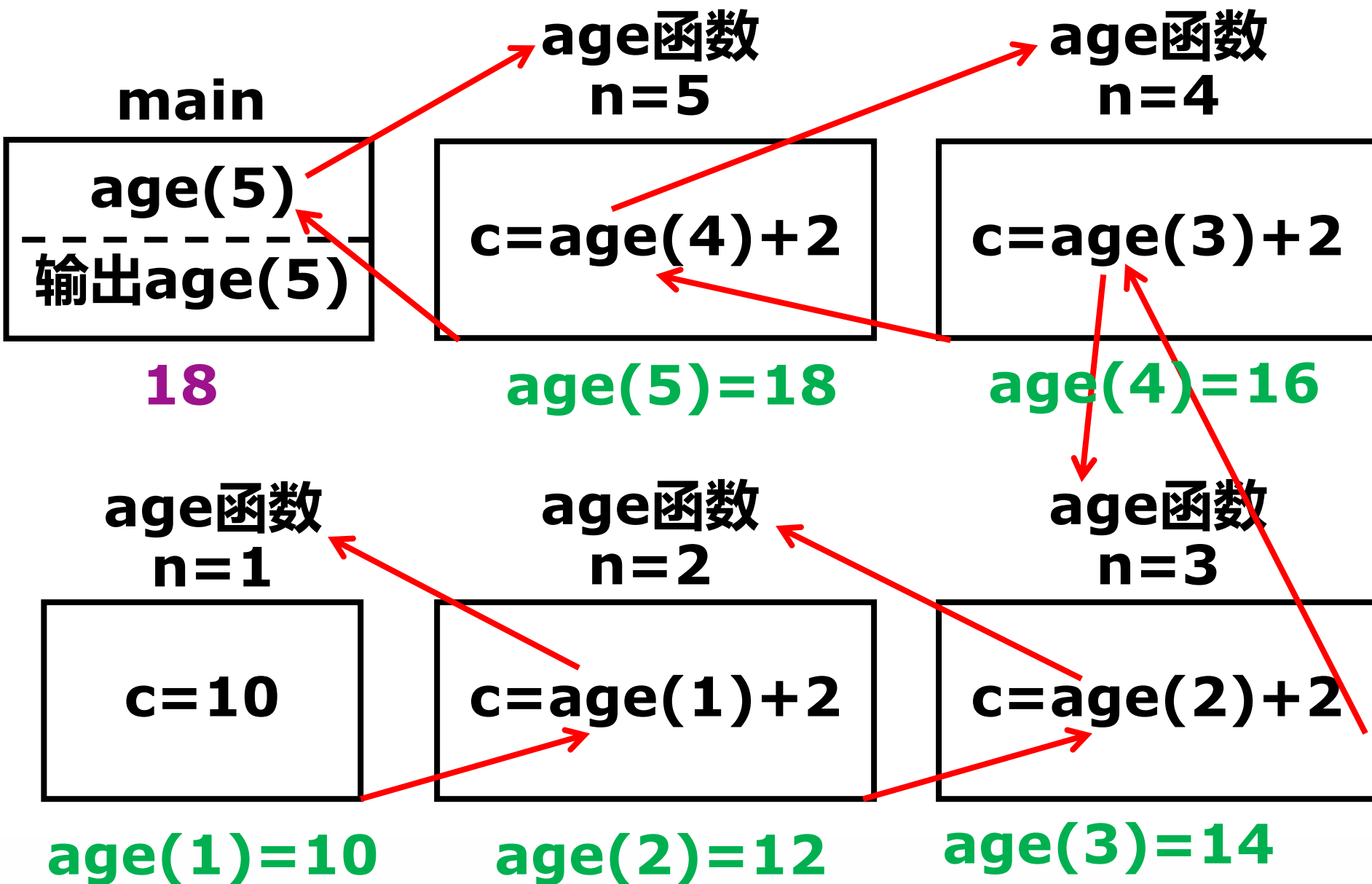




递归的简单应用：年龄

```
1.#include <stdio.h>
2.int main()
3.{
4.    int age(int n);
5.    printf("NO.5,age:%d\n",age(5));
6.    return 0;
7.}
8.int age(int n)
9.{
10.    int c;
11.    if (n==1)
12.        c = 10;
13.    else
14.        c = age(n-1)+2;
15.    return(c);
16.}
```

NO.5,age:18



递归的简单应用：阶乘

➤ 用递归方法求 $n!$

➤ 解题思路：

◆ 求 $n!$ 可以用**递推方法（以前的做法）**：即从 1 开始，乘 2，再乘 3……一直乘到 n 。

◆ 求 $n!$ 也可以用**递归方法**，即 $5!$ 等于 $4! \times 5$ ，而 $4! = 3! \times 4 \dots$ ， $1! = 1$

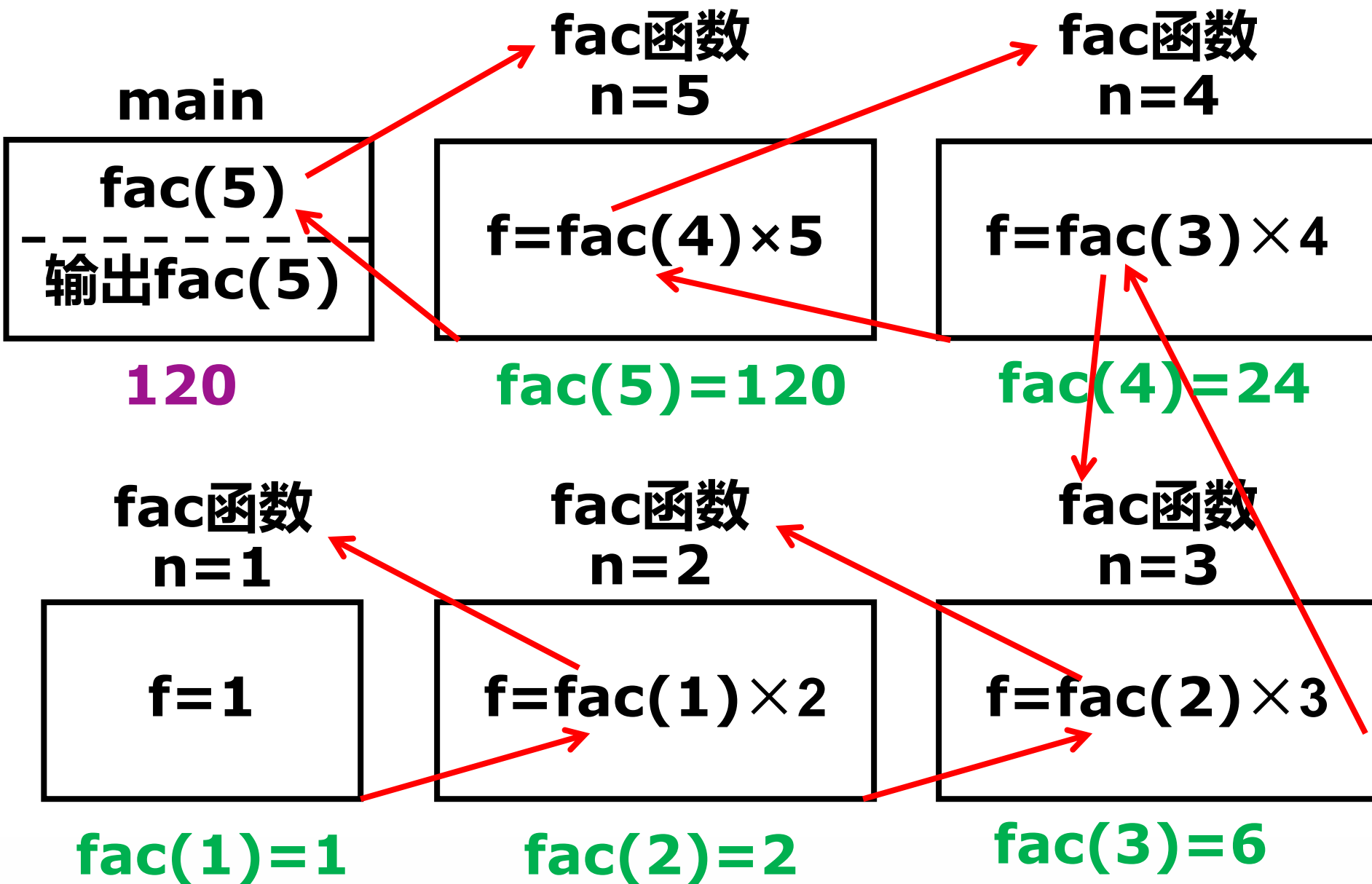
◆ 可用下面的递归公式表示：

$$n! = \begin{cases} n! = 1 & (n = 0, 1) \\ n \bullet (n-1)! & (n > 1) \end{cases}$$

递归的简单应用：阶乘

```
1. #include <stdio.h>
2. int main()
3. {
4.     int fac(int n);
5.     int n, y;
6.     printf("input an integer number:");
7.     scanf("%d",&n);
8.     y = fac(n);
9.     printf("%d!=%d\n",n,y);
10.    return 0;
11.}
12.int fac(int n)
13.{
14.    if (n < 0)
15.        printf("n<0, data error!");
16.    else if (n==0||n==1)
17.        return 1;
18.    else
19.        return fac(n-1)*n;
20.}
```

```
input an integer number:13
13!=1932053504
```



编写和使用递归函数的原则

- 编写递归函数时，首先必须保证函数体内有些“**基本条件**”，当它满足时，能够采用非递归的方式计算得到结果。
- 所谓“**基本条件**”就是当采用递归处理后的子问题**可以直接解决时，就停止继续递归**。
- 为确保递归能够完结，任何递归调用都要**向着“基本条件”的方向**进行。
- 例如：计算阶乘的递归函数，当输入的 n 大于0时，语句“`if (n==0 || n==1) return 1;`”就是基本条件。语句“`return fac(n-1)*n;`”是使递归向着基本条件方向进行的调用。

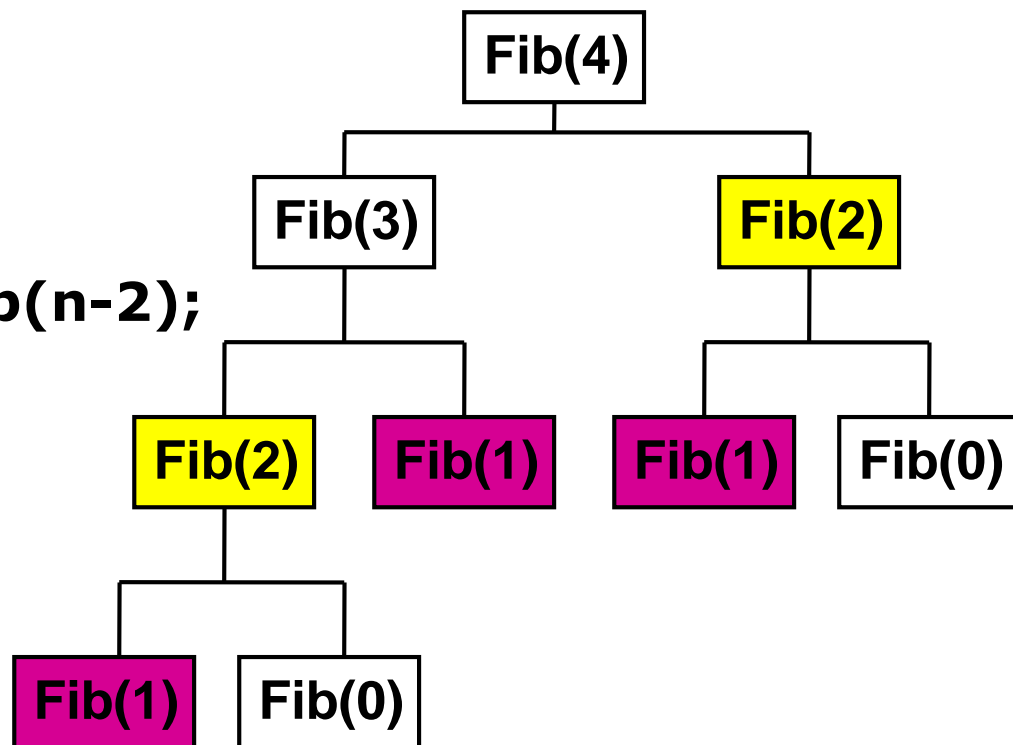
不需要试图验证递归的每一步

- 理解递归，就类似于理解数学归纳法。
- **数学归纳法的基本思想**：首先确定初始条件比如 $f(0)$ 成立，然后试图证明若 $f(n)$ 成立，则 $f(n+1)$ 成立，如此，则对任意大的整数 m ， $f(m)$ 成立。
 - ◆ 注意：这个过程中，我们并不去验证每一个整数 m ， $f(m)$ 是否都成立。
- 同理，我们不必关心每次递归是否能够正确进行，只要保证基本条件存在，且递归方向朝着基本条件进行即可。

应避免过度使用递归

➤ 以下斐波那契数列的递归实现有什么问题

```
int Fib (int n)
{
    if (n > 1)
        return Fib(n-1) + Fib(n-2);
    else
        return n;
}
```



正确理解递归调用的顺序

➤ 变动函数语句的顺序会导致执行结果的巨大变化

```
1.#include <stdio.h>
2.void func(int a)
3.{
4.    if (a < 4)
5.    {
6.        printf("%d\n", a);
7.        func(a+1);
8.    }
9.}
10.int main()
11.{
12.    func(0);
13.    return 0;
14.}
```

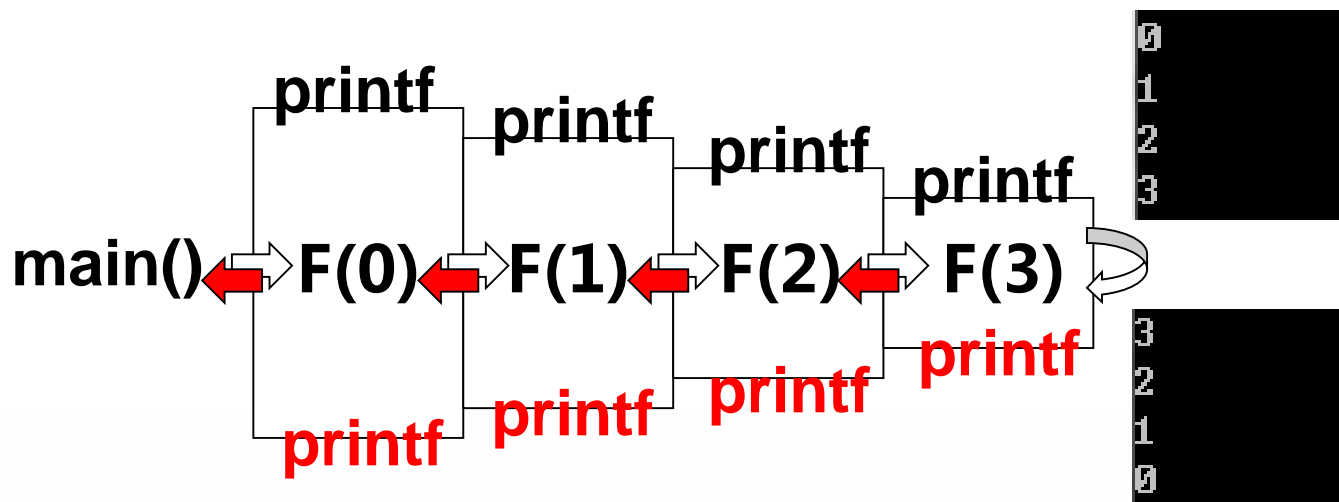
0
1
2
3

```
1.#include <stdio.h>
2.void func(int a)
3.{
4.    if (a < 4)
5.    {
6.        func(a+1);
7.        printf("%d\n", a);
8.    }
9.}
10.int main()
11.{
12.    func(0);
13.    return 0;
14.}
```

3
2
1
0

小结递归的过程

- 递归过程的实现就是**自己调用自己**；
- 递归调用的过程总是**层层向下**，退出时次序正好相反；
- 主程序**首次调用**递归过程为**外部调用**；
- 递归过程**每次递归**调用自己都属于**内部调用**
- 一个递归调用的函数，每次函数**返回**的位置各不相同



汉诺塔的传说：世界末日

- 在世界中心贝拿勒斯（在印度北部）的圣庙里，一块黄铜板上插着三根宝石针。印度教的主神梵天在创造世界的时候，在其中一根针上从下到上地穿好了由大到小的64片金片，这就是所谓的**汉诺塔**。
- 不论白天黑夜，总有一个僧侣在按照下面的法则移动这些金片：**一次只移动一片，不管在哪根针上，小片必须在大片上面。**
- 僧侣们预言，当所有的金片都从梵天穿好的那根针上移到另外一根针上时，世界就将在一声霹雳中消灭，而梵塔、庙宇和众生也都将同归于尽。

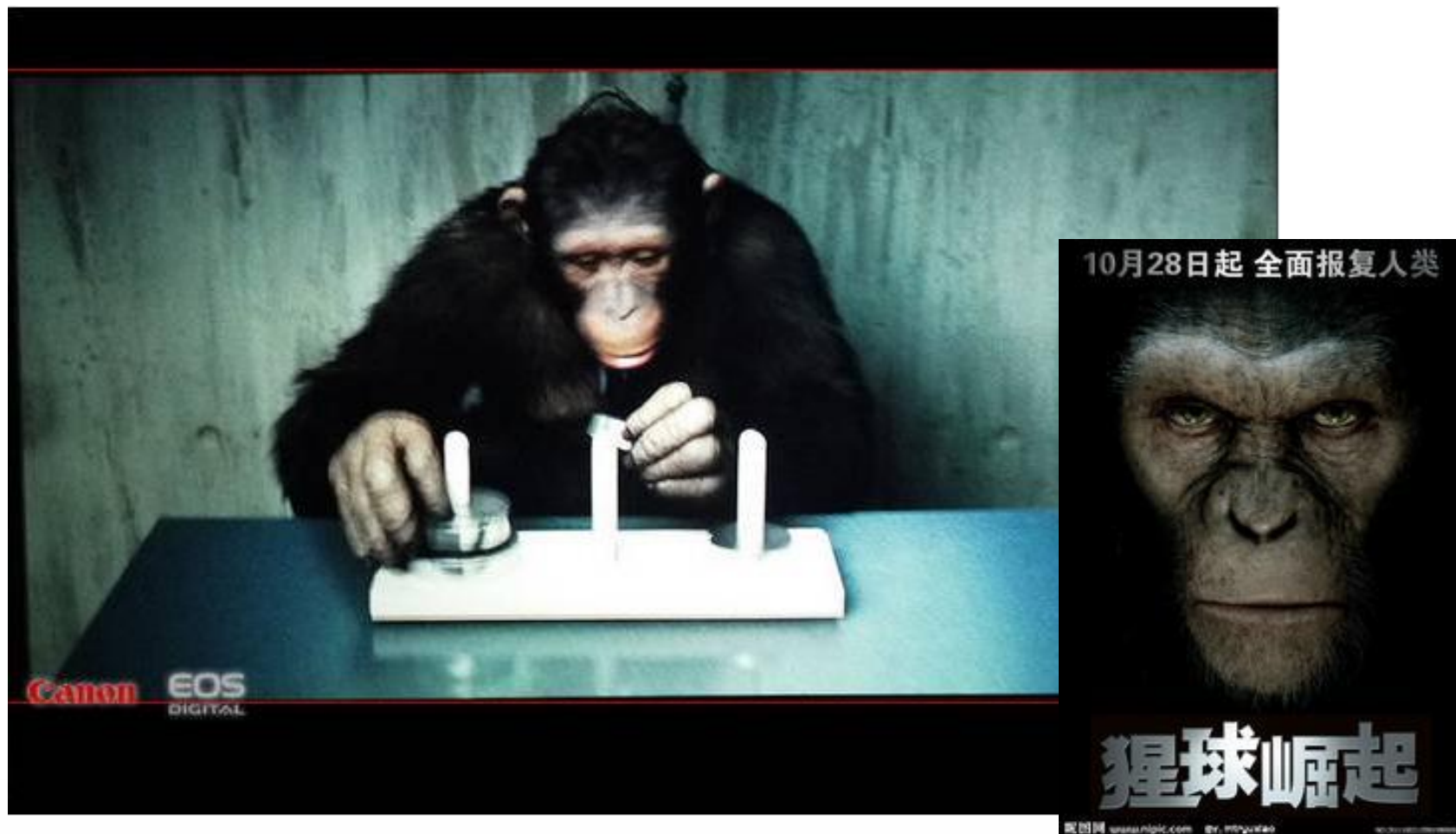


经典递归问题：汉诺塔

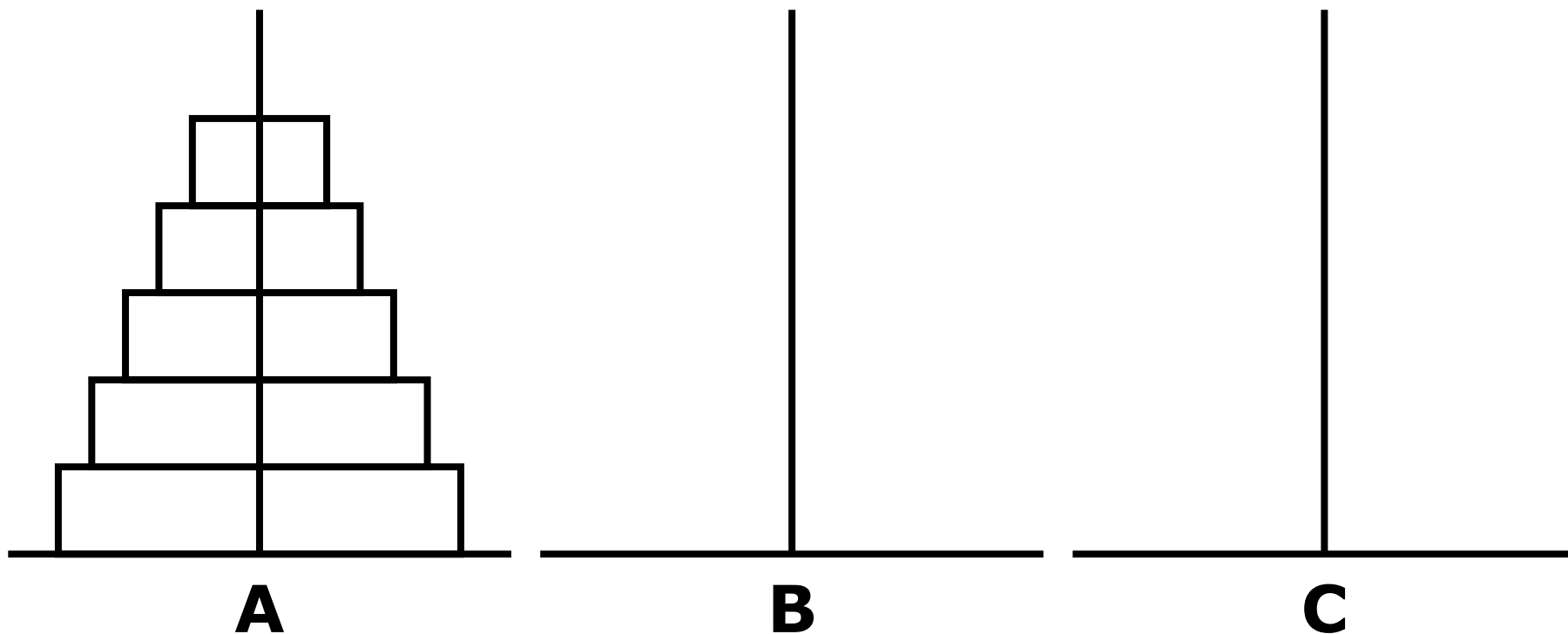


- 古代有一个梵塔，塔内有3个座A、B、C，开始时A座上有64个盘子，盘子大小不等，大的在下，小的在上。有一个老和尚想把这64个盘子从A座移到C座，但规定**每次只允许移动一个盘，且在移动过程中在3个座上都始终保持大盘在下，小盘在上。在移动过程中可以利用B座。**要求编程序输出移动一盘子的步骤。

若理解不了汉诺塔，压力会很大



经典递归问题：汉诺塔



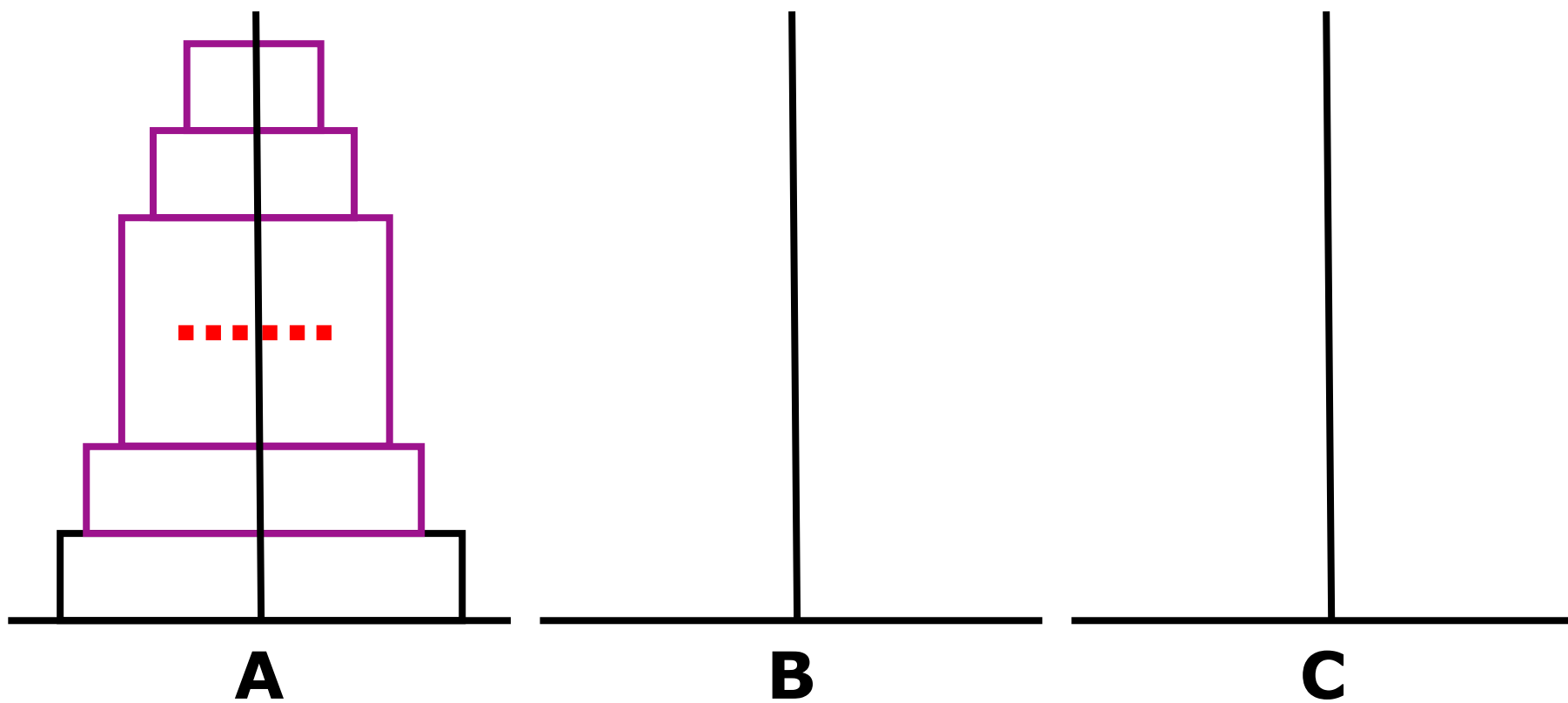
经典递归问题：汉诺塔

➤ 解题思路：

- ◆ 要把64个盘子从A座移动到C座，需要移动大约 2^{64} 次盘子。一般人是不可能直接确定移动盘子的每一个具体步骤的
- ◆ 老和尚会这样想：假如有另外一个和尚能有办法将上面63个盘子从一个座移到另一座。那么，问题就解决了。此时老和尚只需这样做：
 - (1) 命令第2个和尚将63个盘子从A座移到B座
 - (2) 自己将1个盘子（最底下的、最大的盘子）从A座移到C座
 - (3) 再命令第2个和尚将63个盘子从B座移到C座

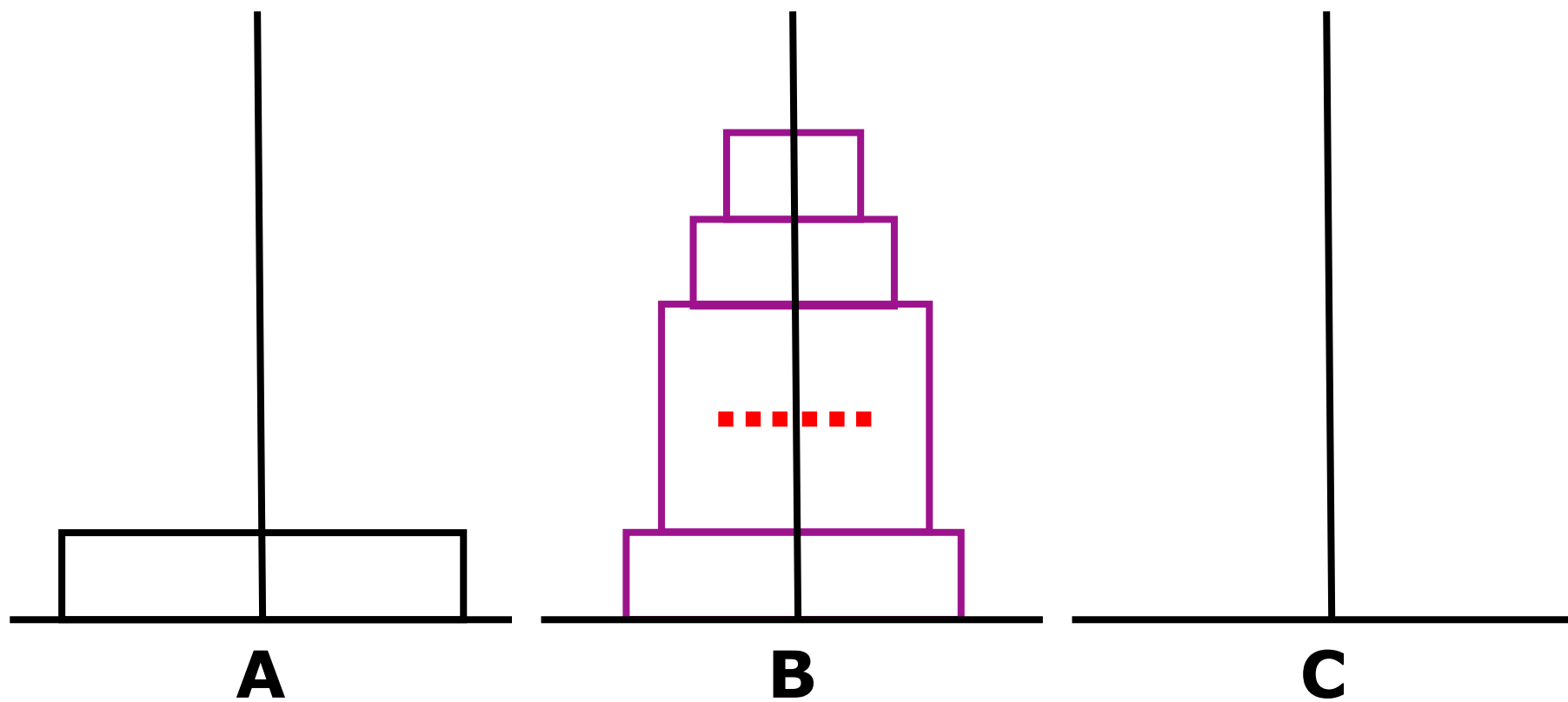
第1个和尚的做法

将63个从A到B



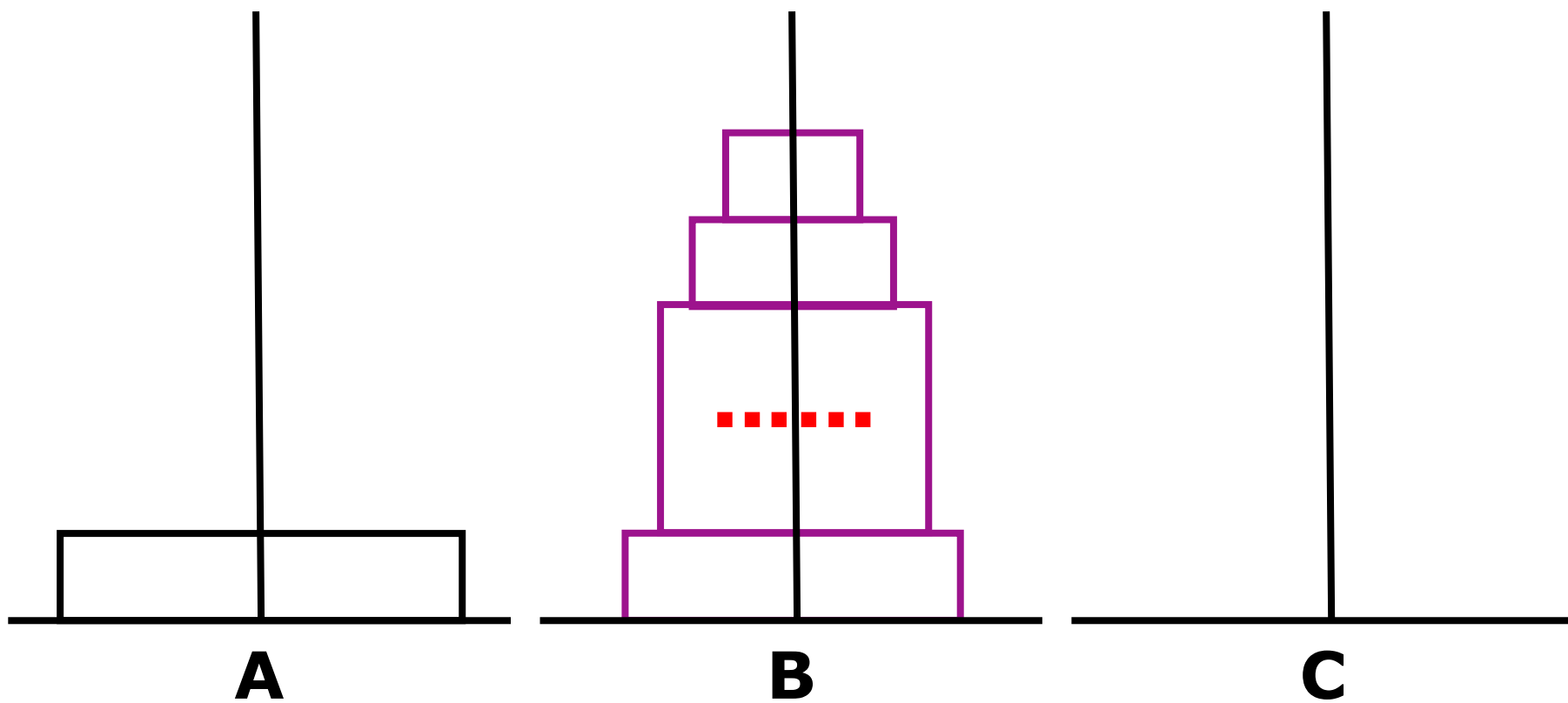
第1个和尚的做法

将63个从A到B



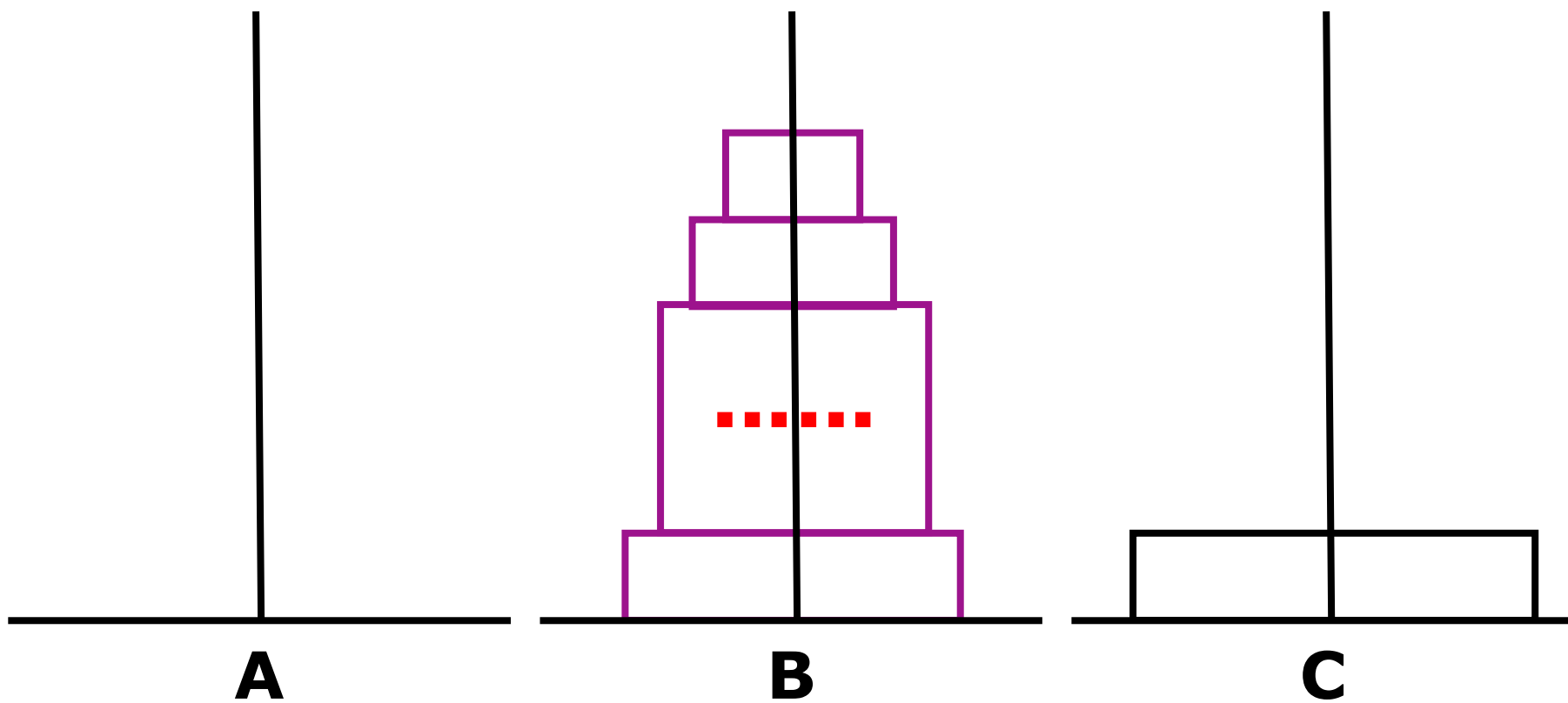
第1个和尚的做法

将1个从A到C



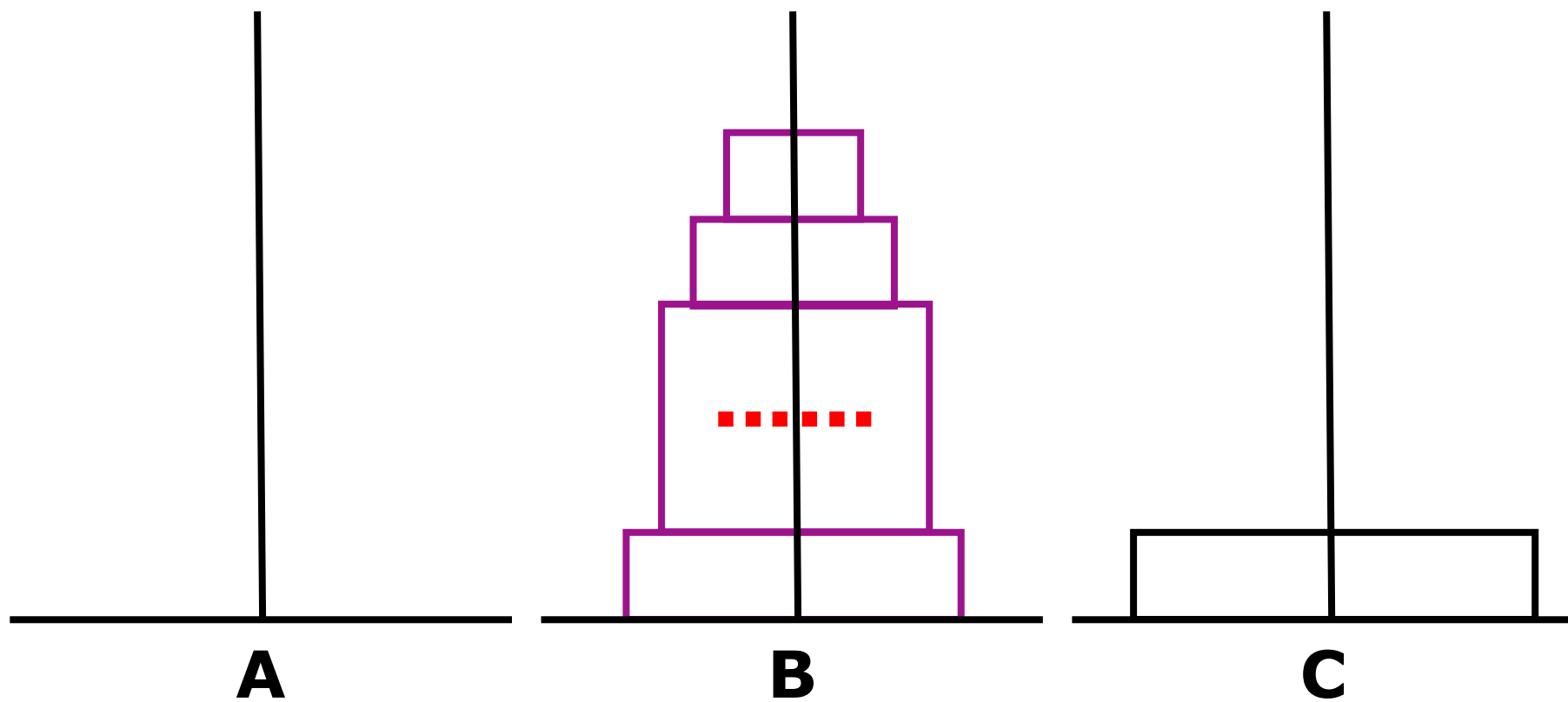
第1个和尚的做法

将1个从A到C



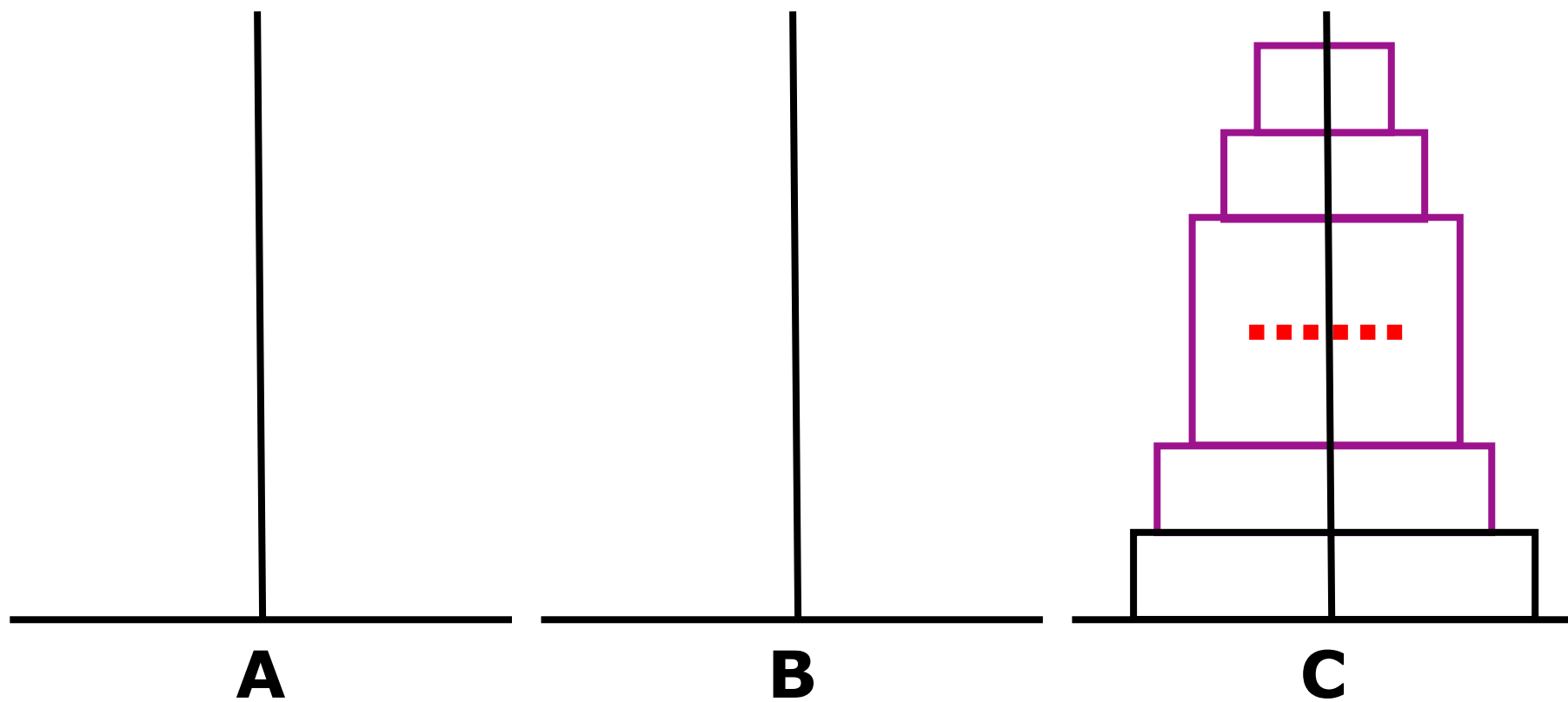
第1个和尚的做法

将63个从B到C



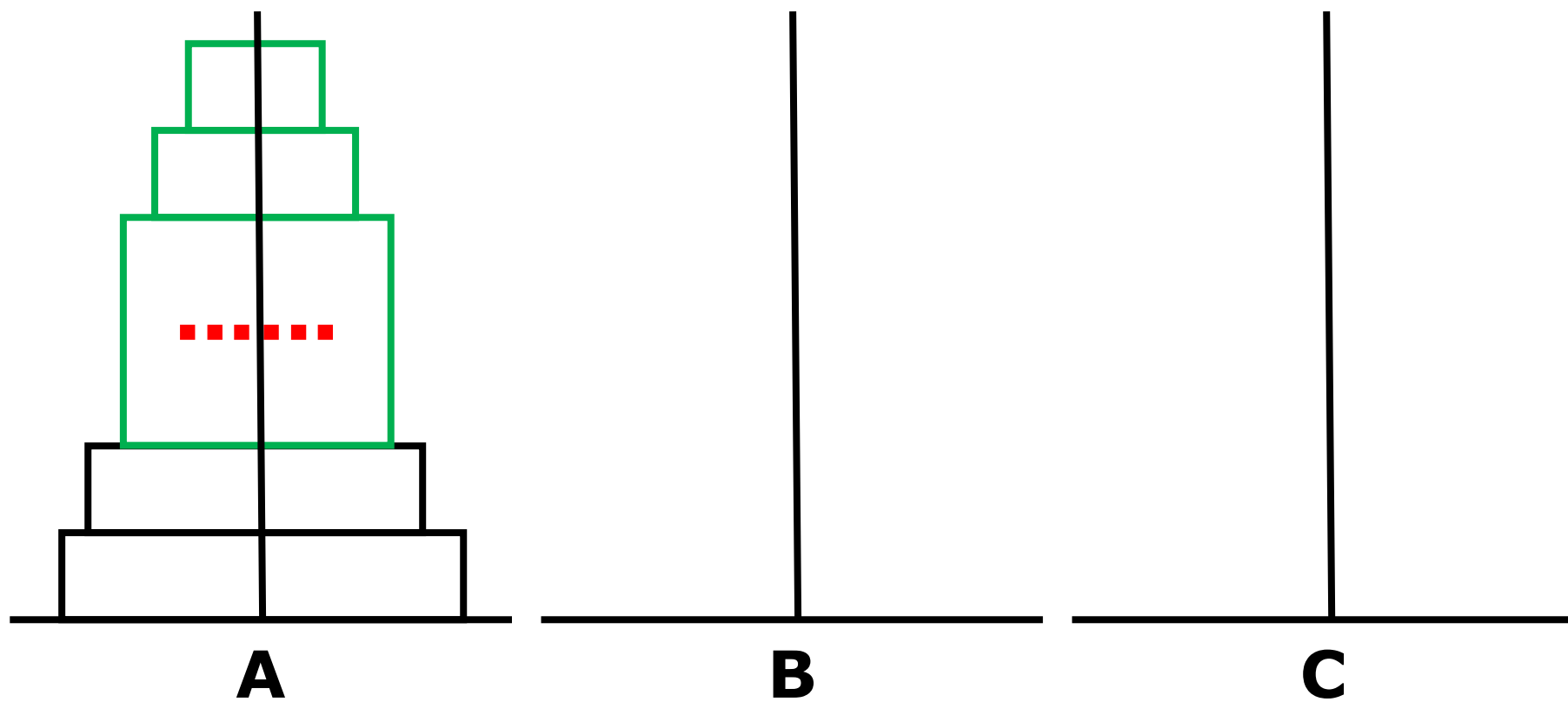
第1个和尚的做法

将63个从B到C



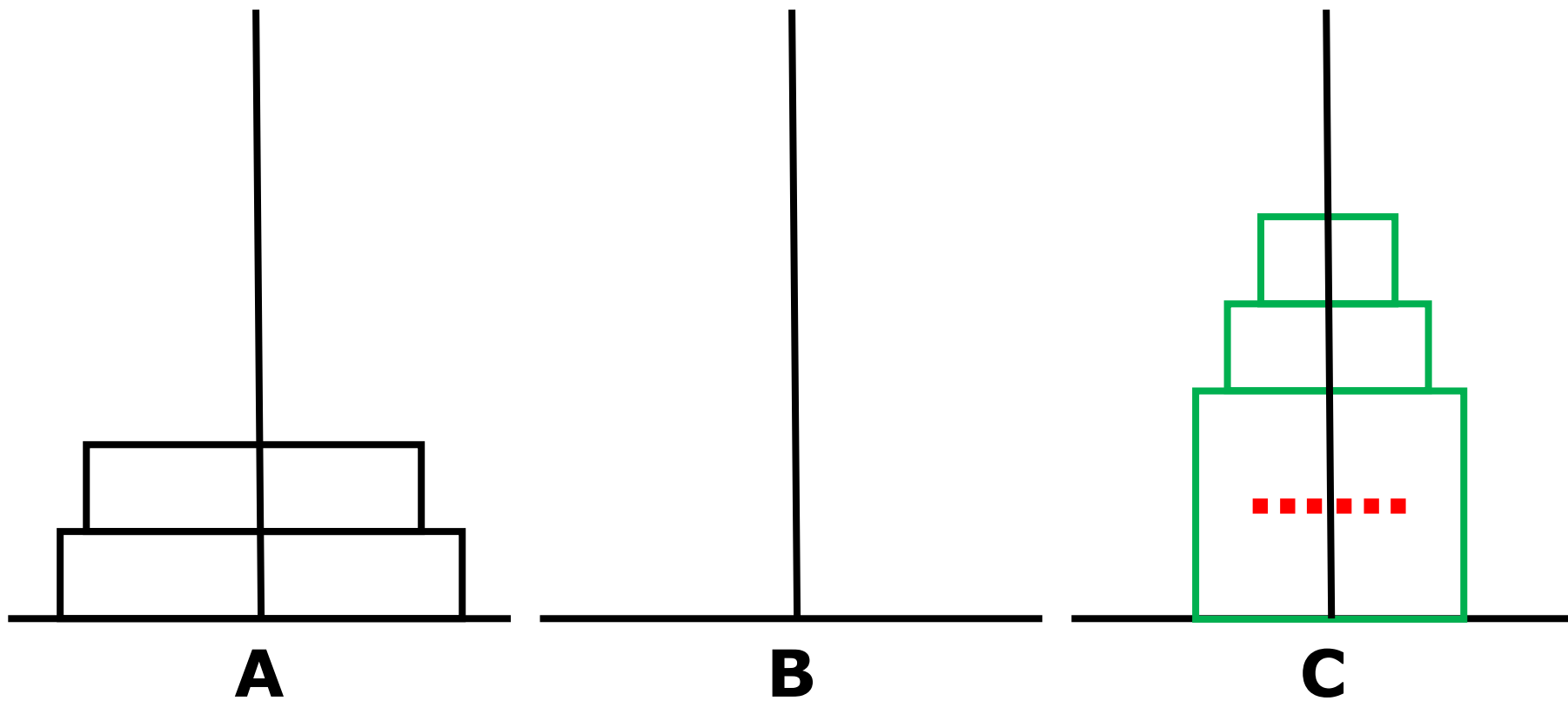
第2个和尚的做法

将62个从A到C



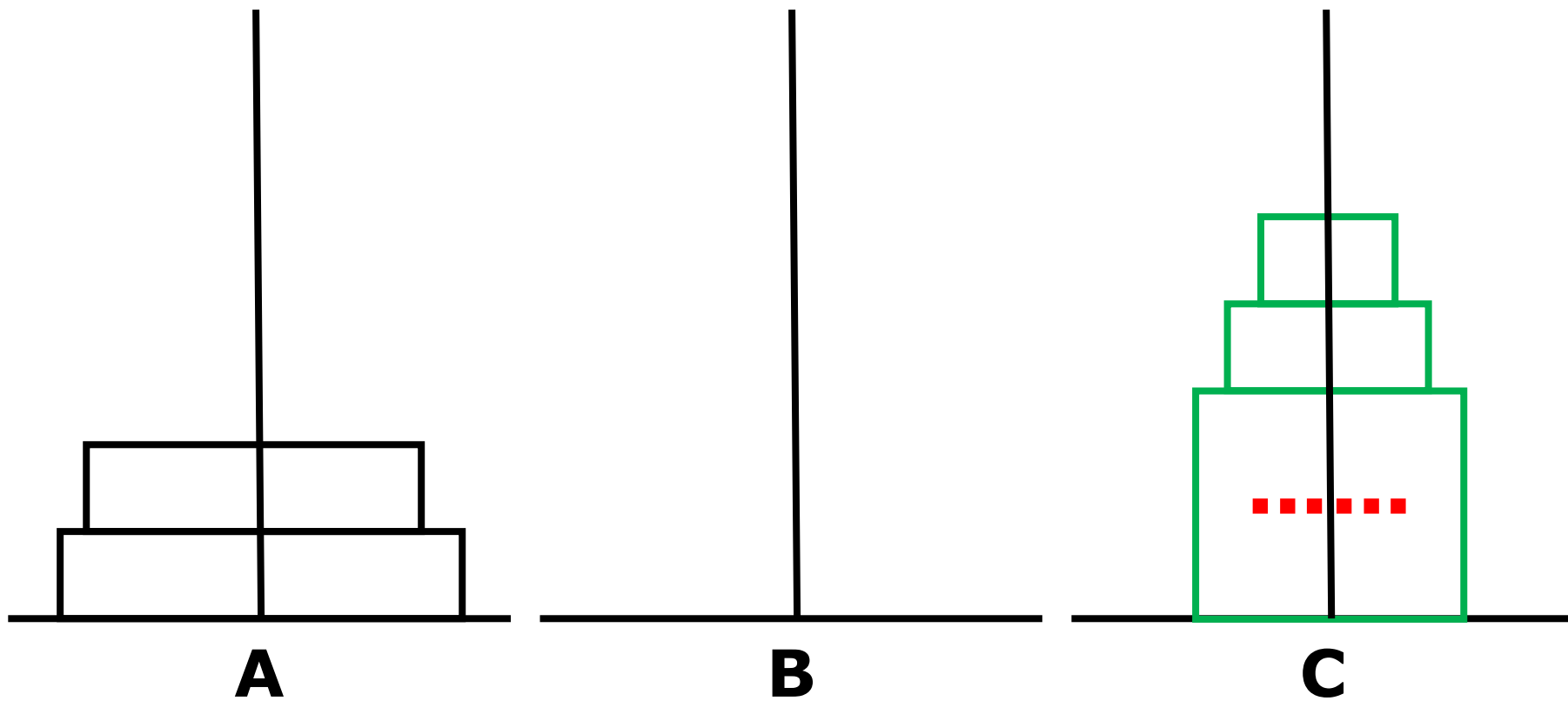
第2个和尚的做法

将62个从A到C



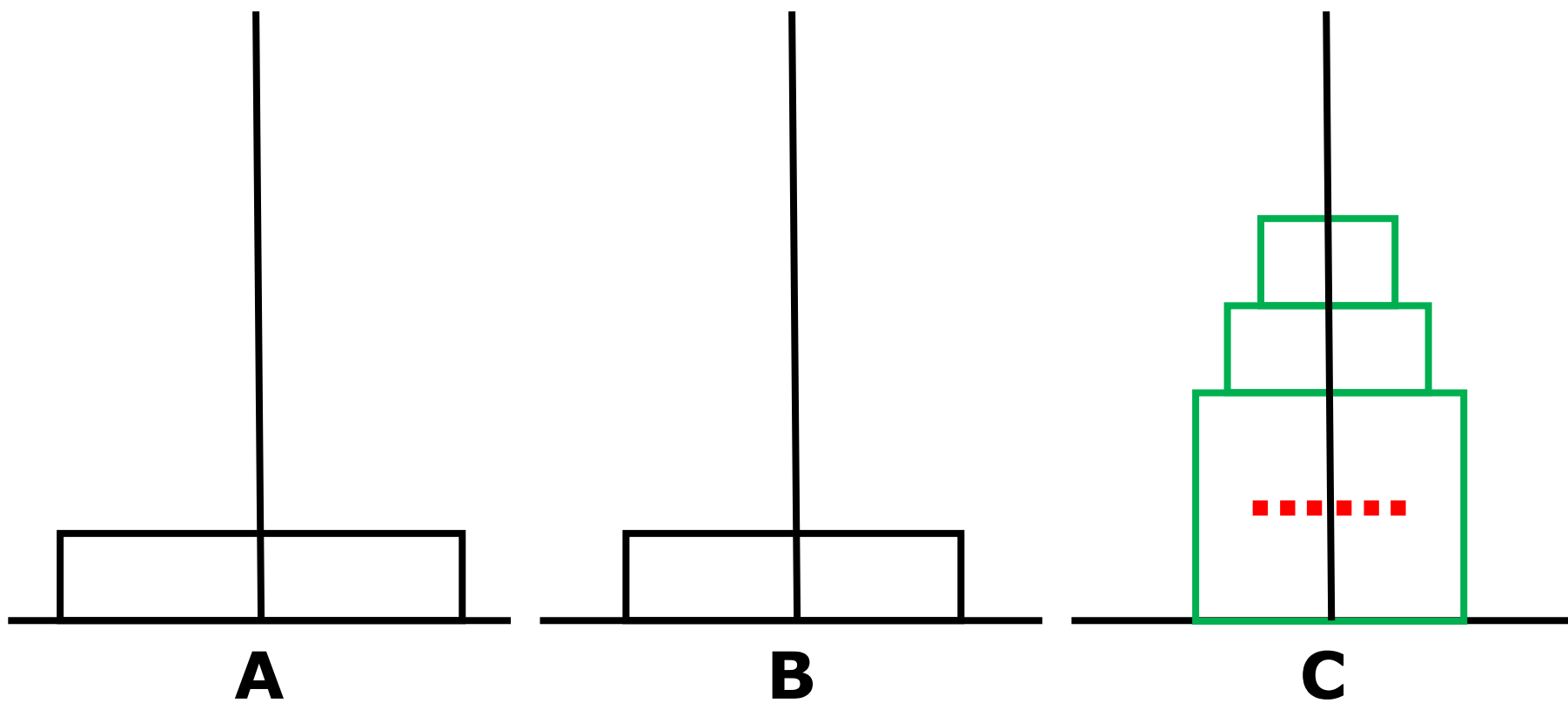
第2个和尚的做法

将1个从A到B



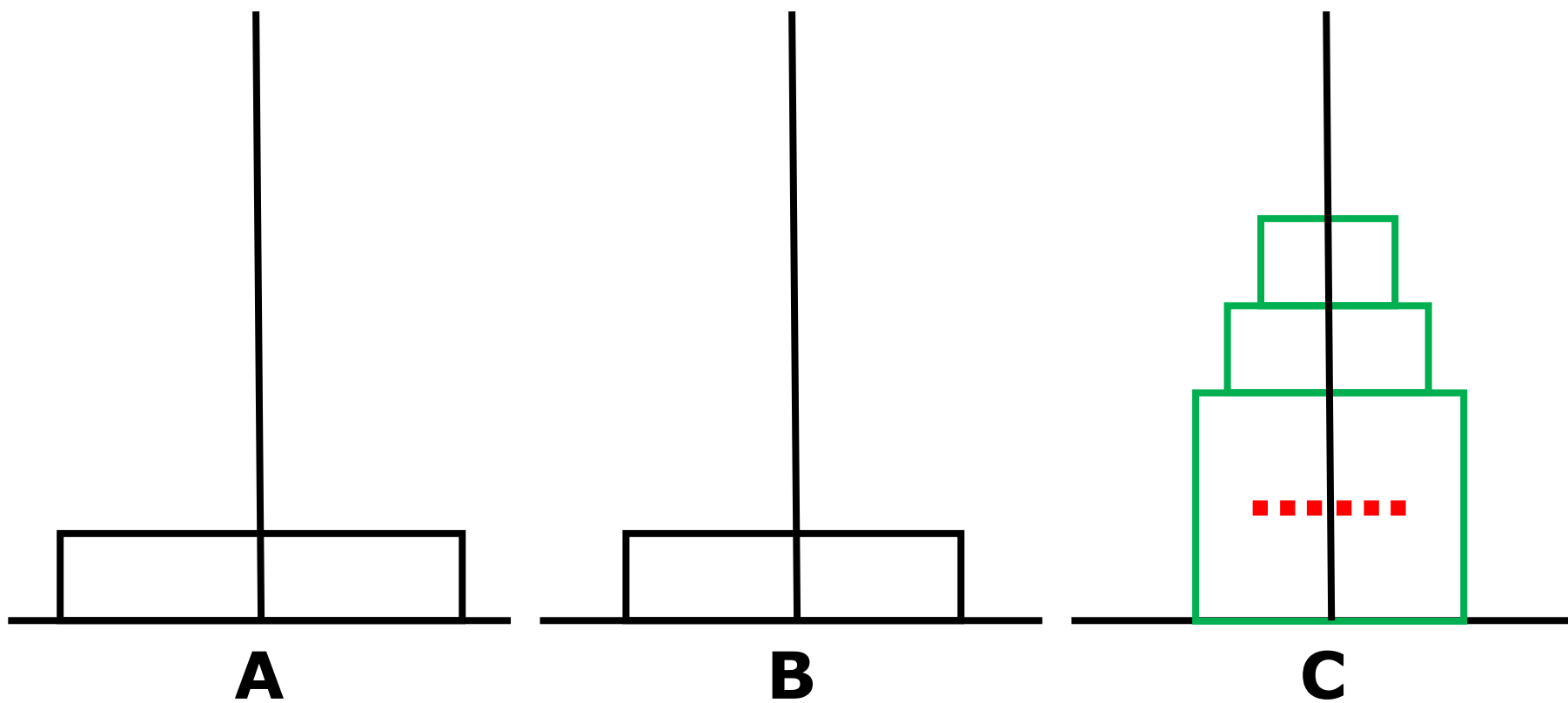
第2个和尚的做法

将1个从A到B



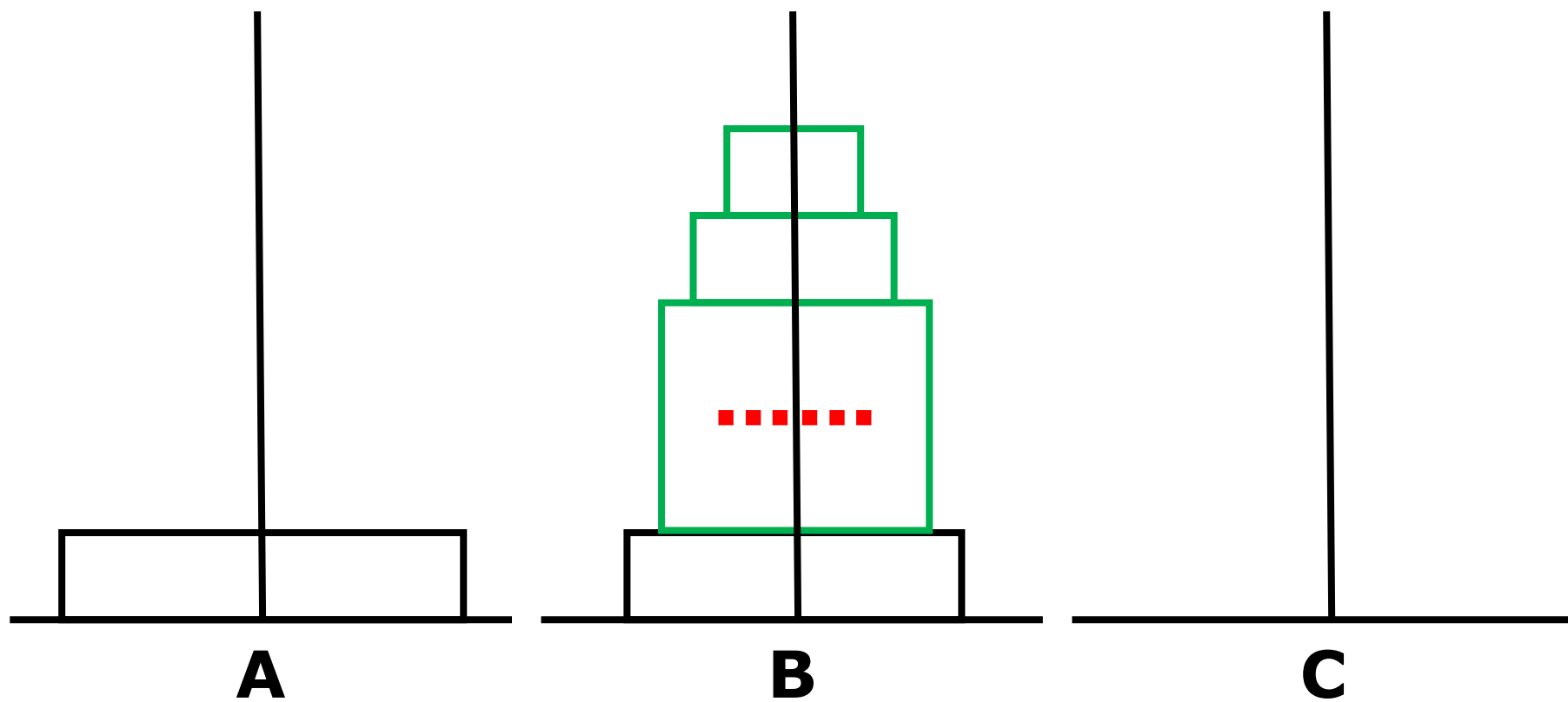
第2个和尚的做法

将62个从C到B



第2个和尚的做法

将62个从C到B



第**3**个和尚的做法

第**4**个和尚的做法

第**5**个和尚的做法

第**6**个和尚的做法

第**7**个和尚的做法

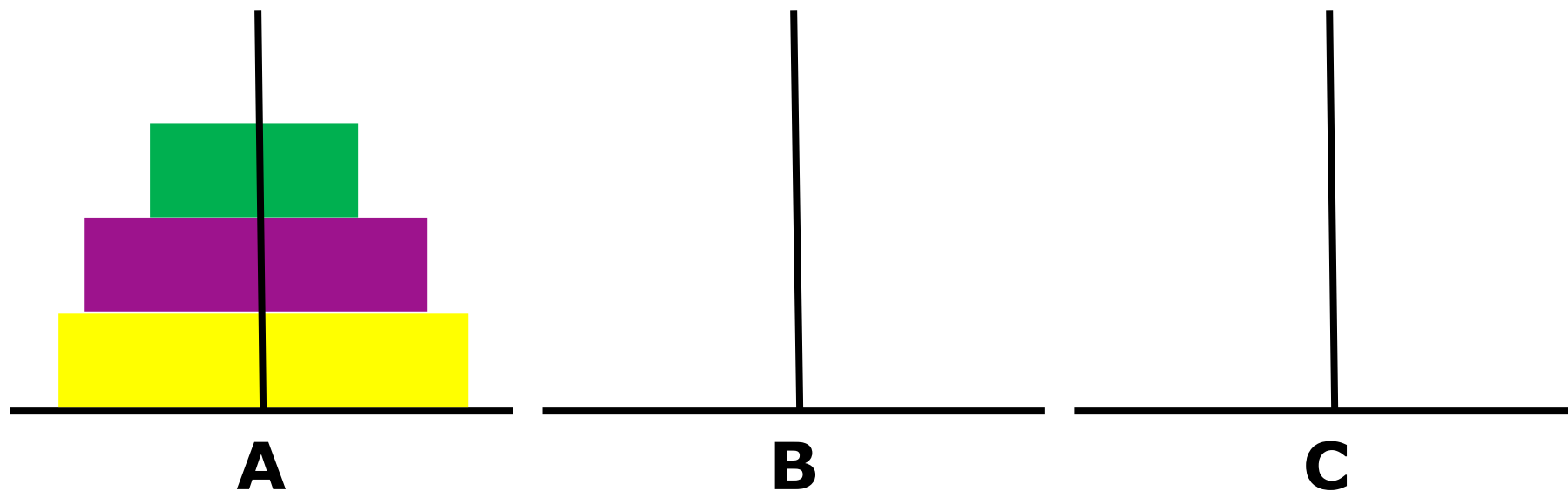
.....

第**63**个和尚的做法

第**64**个和尚仅做：将**1**个从A移到C

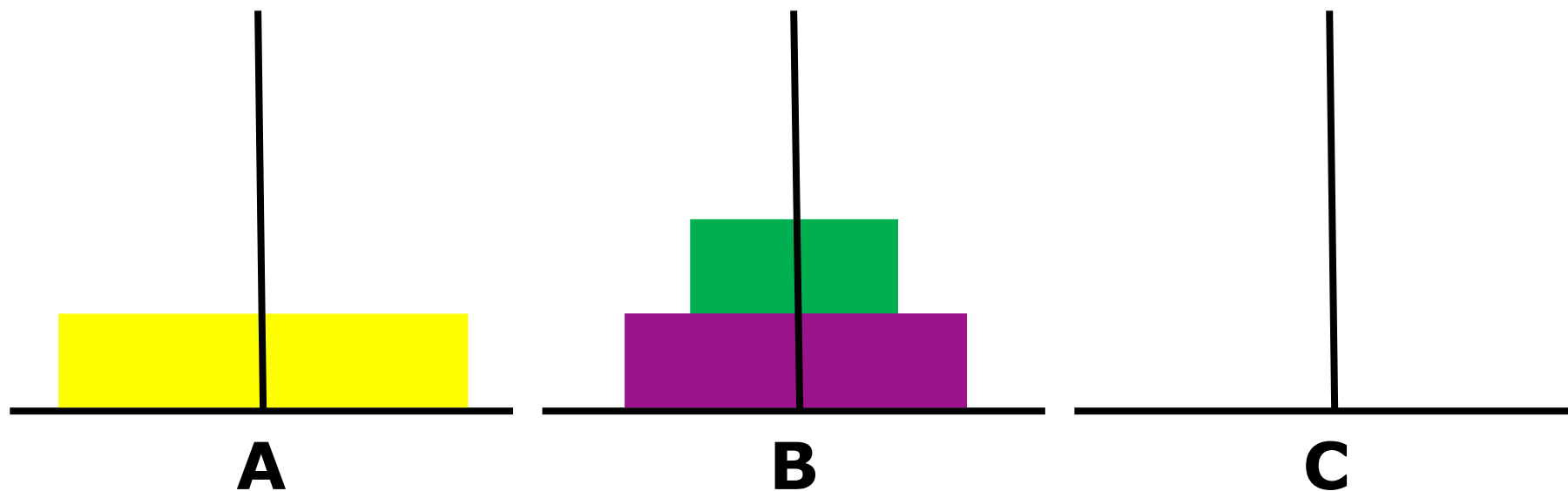
将3个盘子从A移到C的全过程

将2个盘子从A移到B



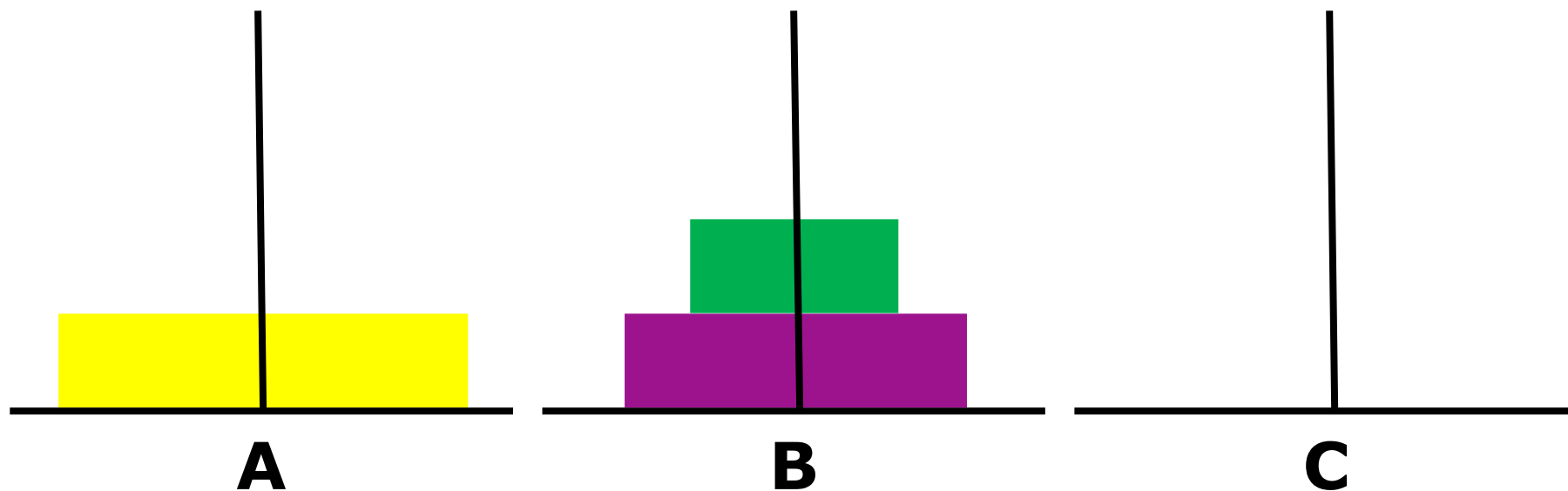
将3个盘子从A移到C的全过程

将2个盘子从A移到B



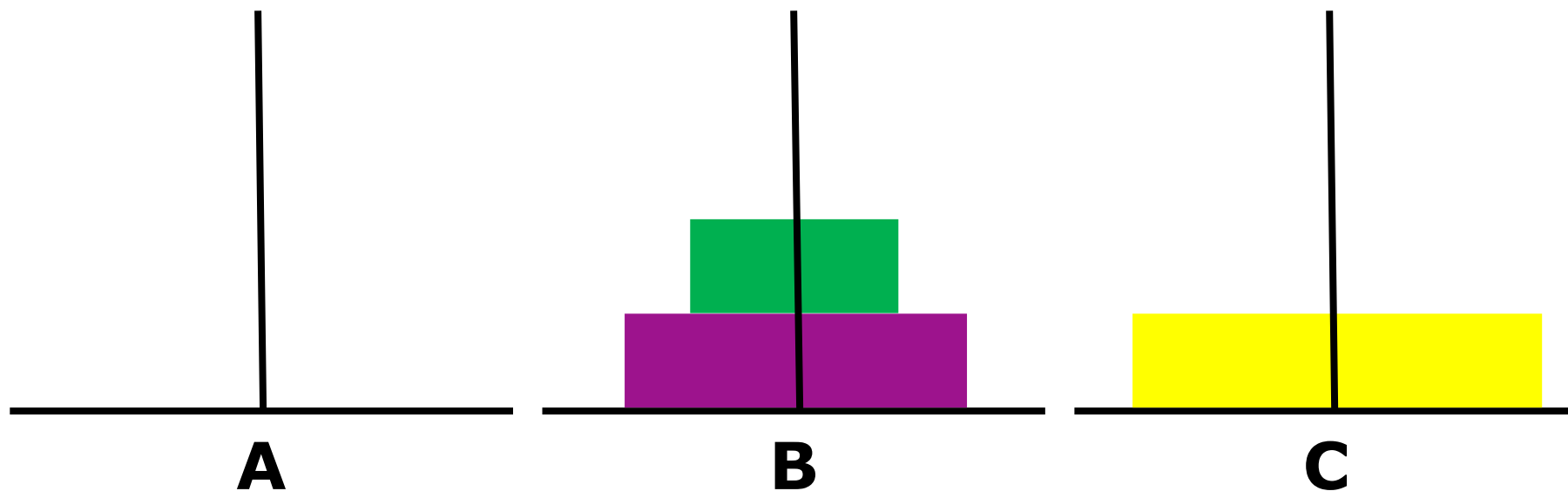
将3个盘子从A移到C的全过程

将1个盘子从A移到C



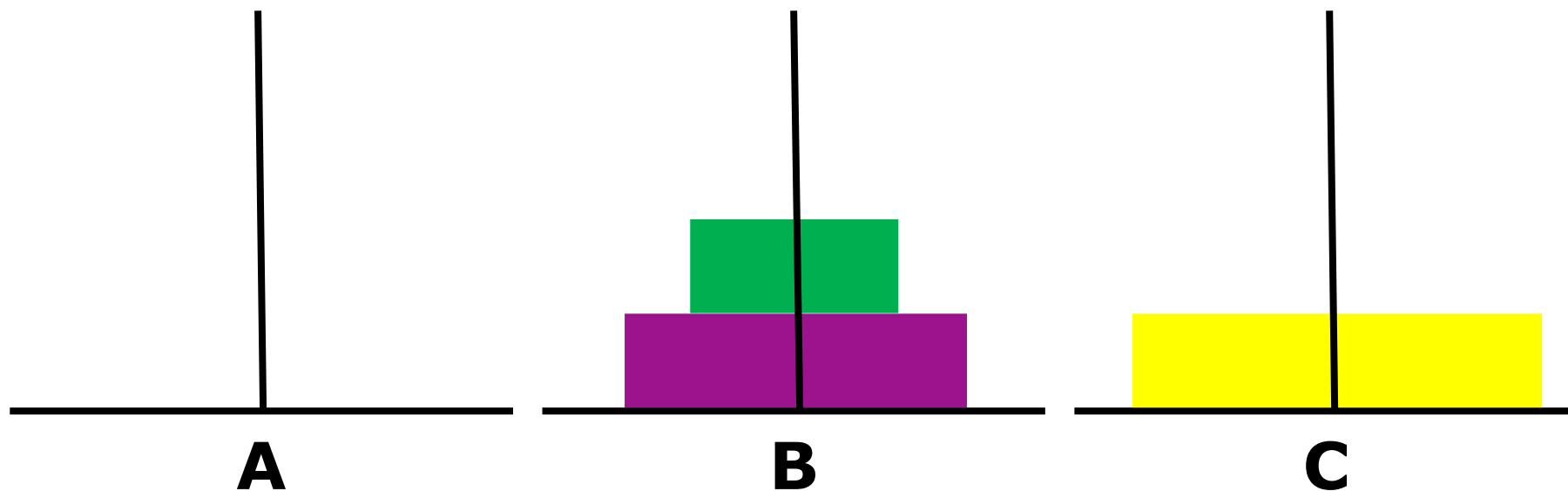
将3个盘子从A移到C的全过程

将1个盘子从A移到C



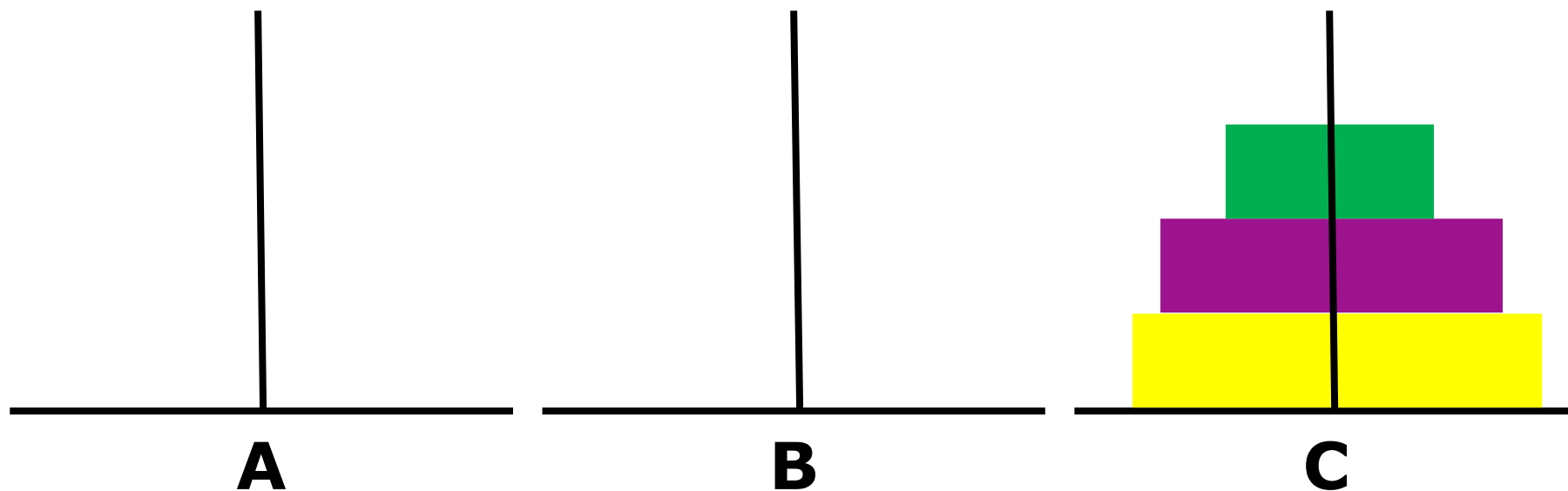
将3个盘子从A移到C的全过程

将2个盘子从B移到C



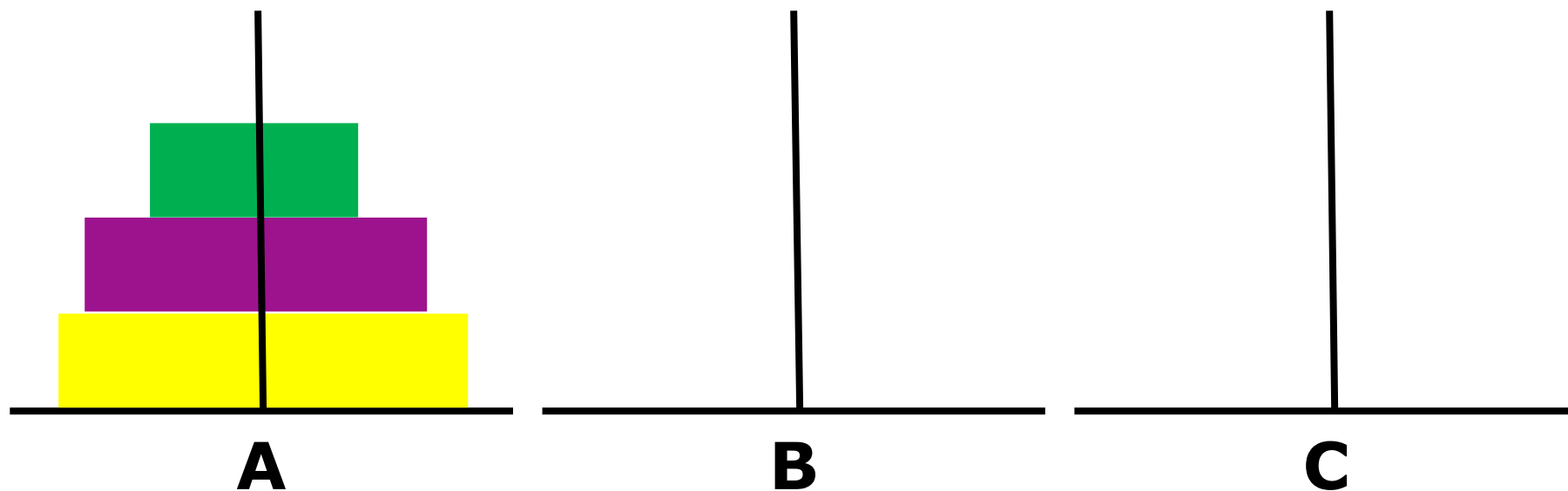
将3个盘子从A移到C的全过程

将2个盘子从B移到C



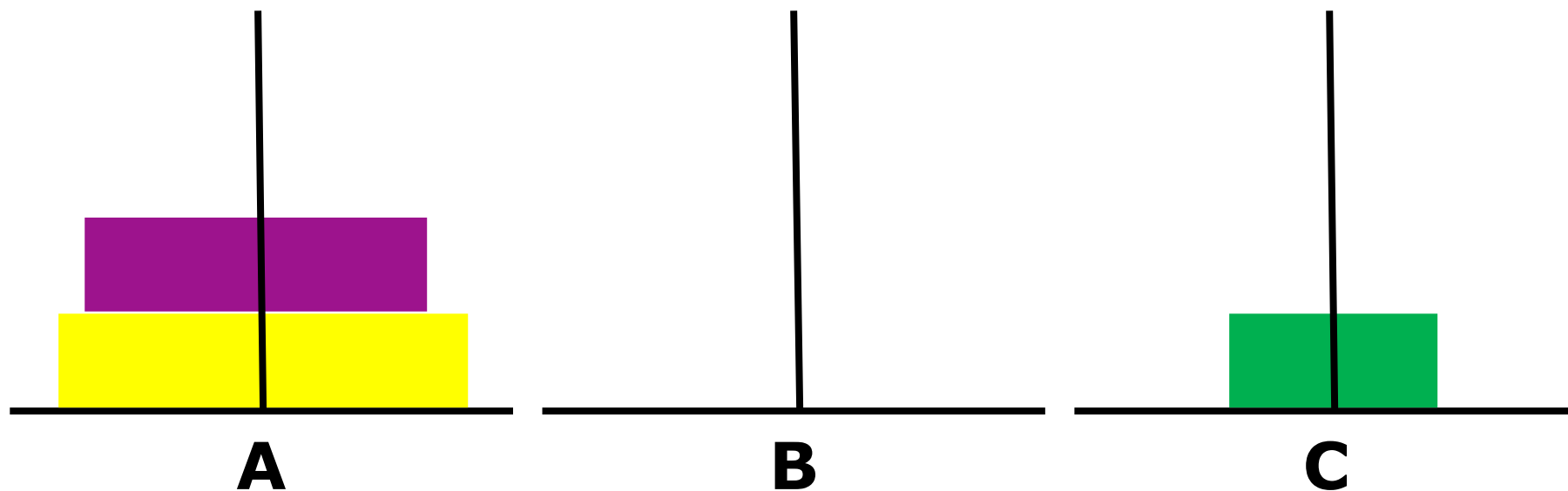
将2个盘子从A移到B的过程

将1个盘子从A移到C



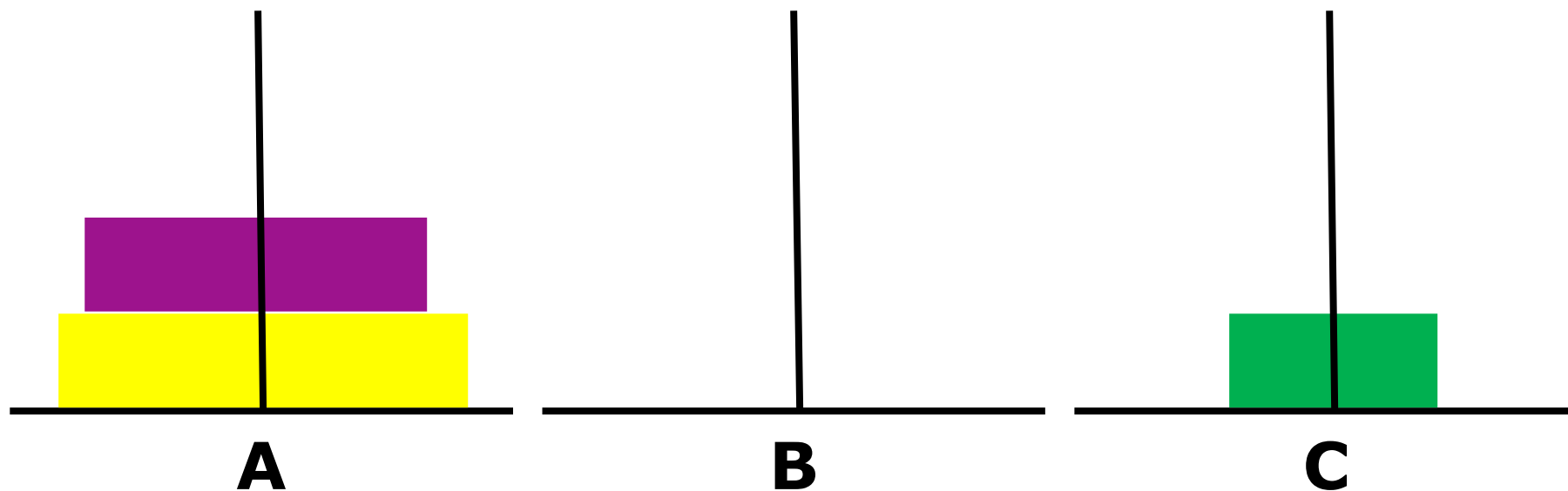
将2个盘子从A移到B的过程

将1个盘子从A移到C



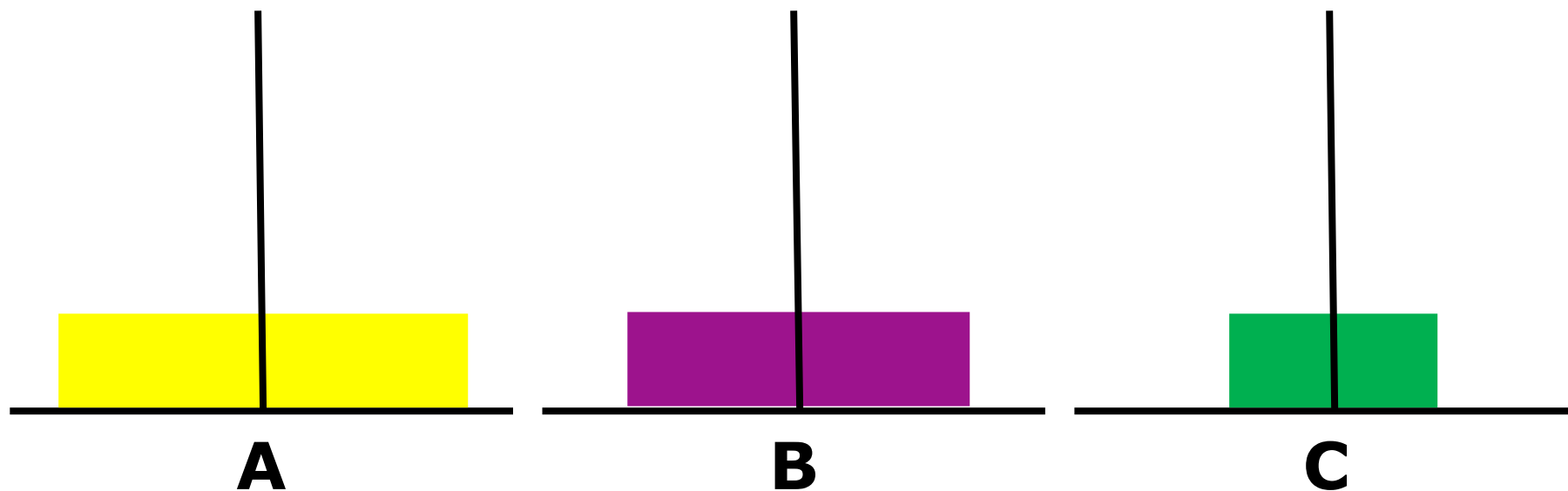
将2个盘子从A移到B的过程

将1个盘子从A移到B



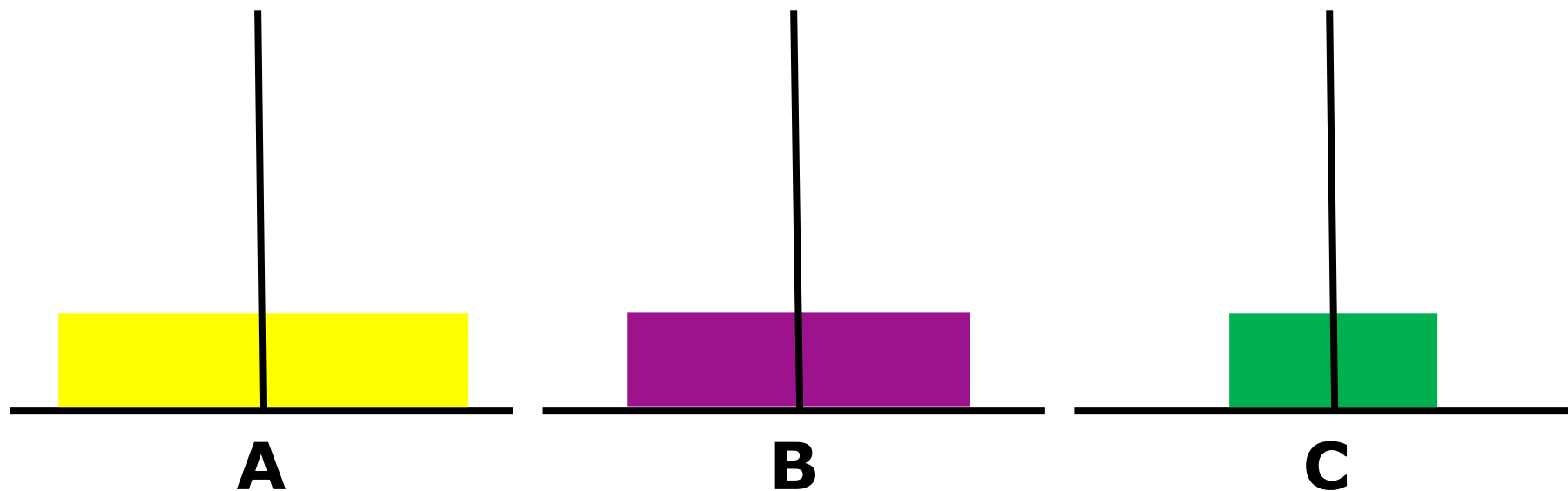
将2个盘子从A移到B的过程

将1个盘子从A移到B



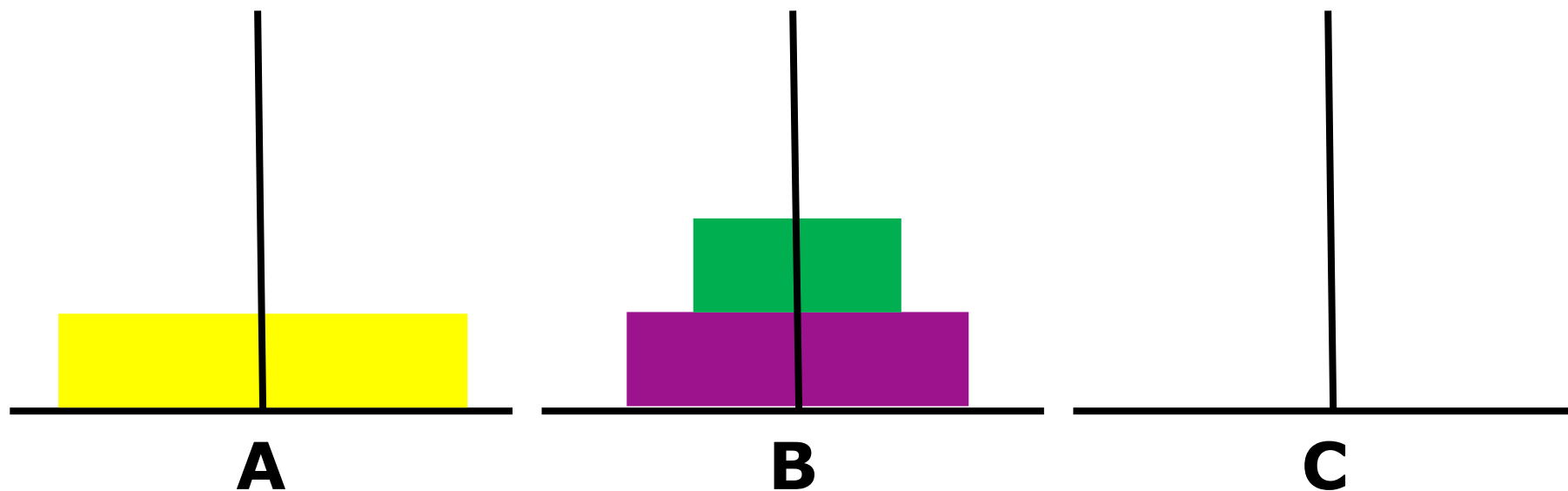
将2个盘子从A移到B的过程

将1个盘子从C移到B

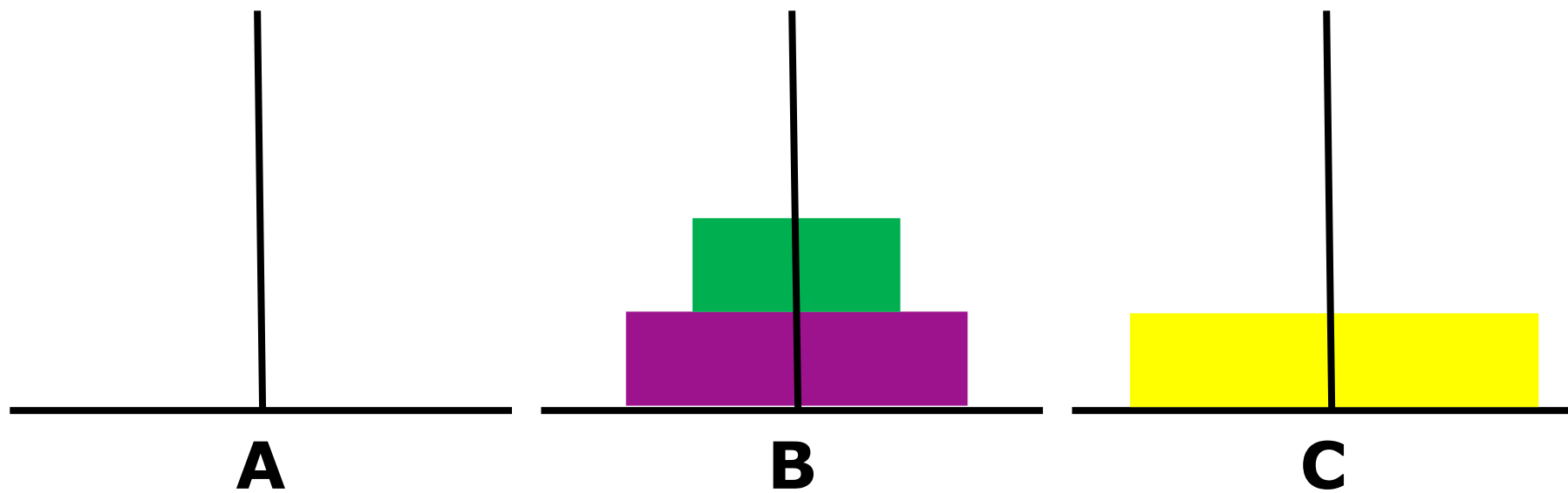


将2个盘子从A移到B的过程

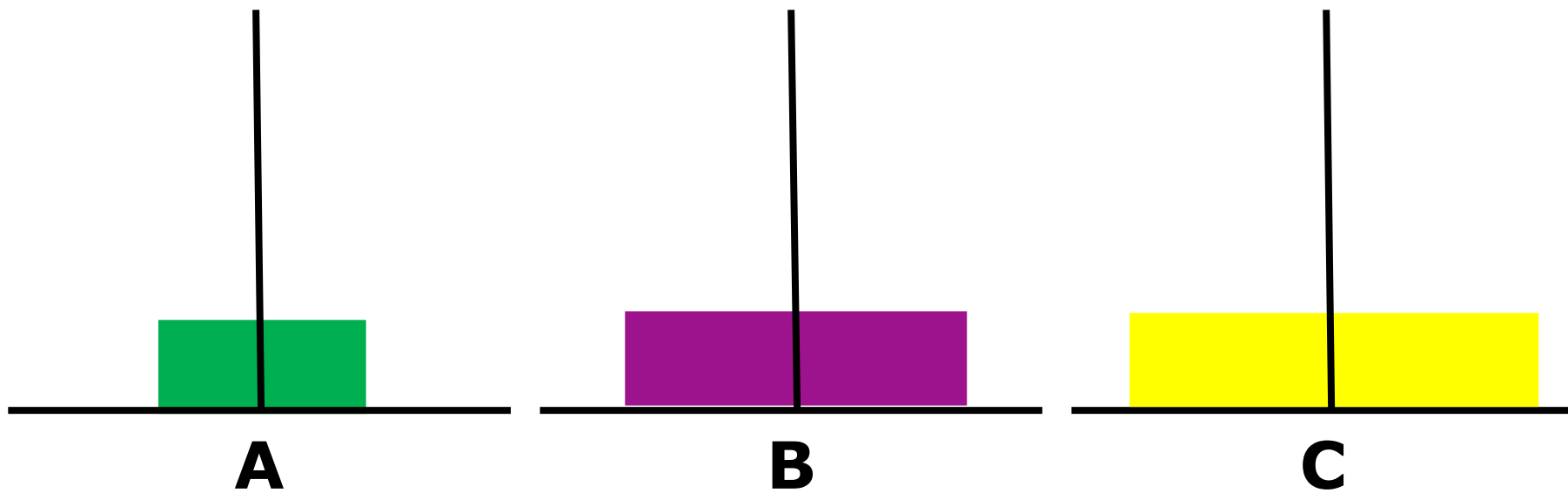
将1个盘子从C移到B



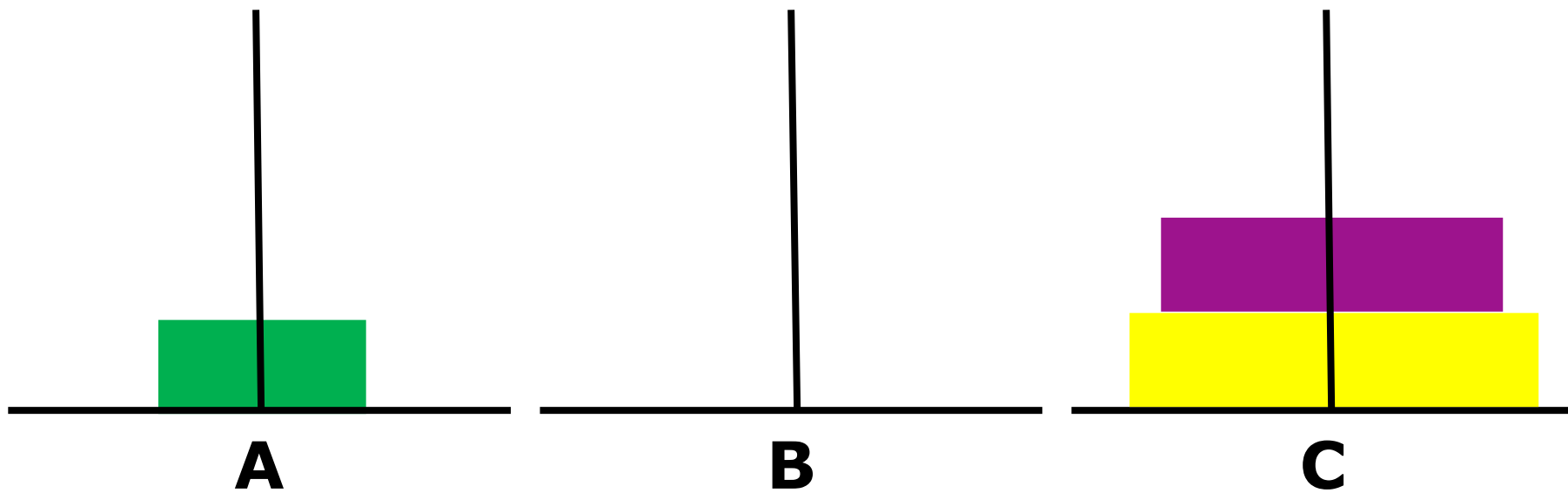
将2个盘子从B移到C的过程



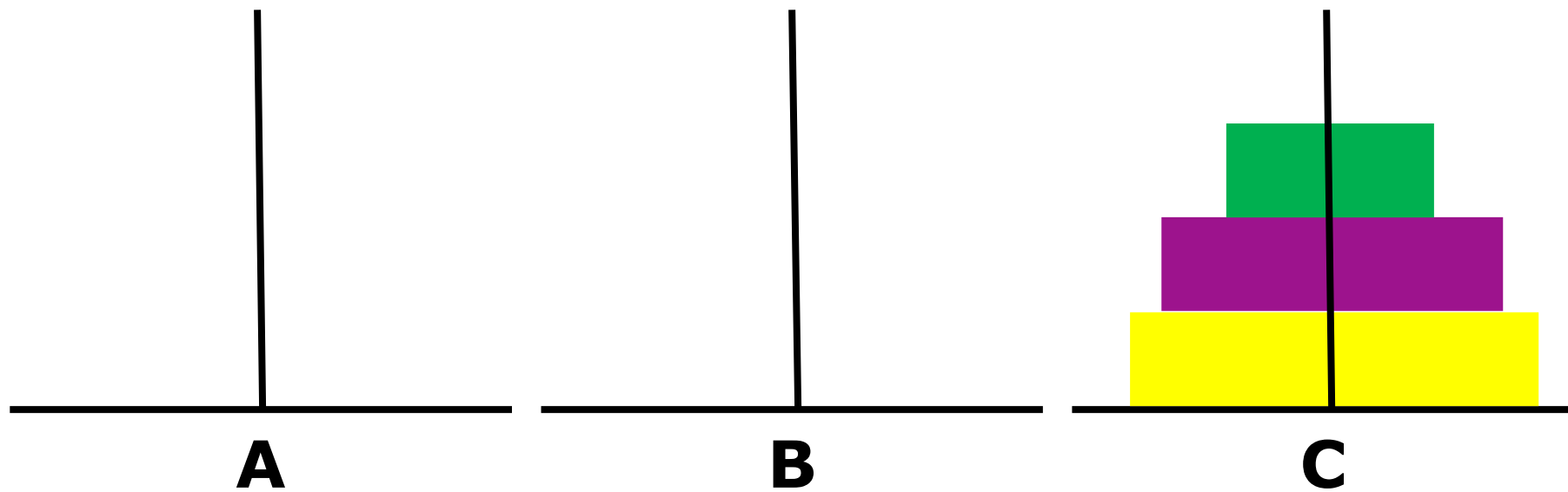
将2个盘子从B移到C的过程



将2个盘子从B移到C的过程



将2个盘子从B移到C的过程



经典递归问题：汉诺塔

- 由上面的分析可知：将 n 个盘子从A座移到C座可以分解为以下3个步骤：
- (1) 将 $n-1$ 个盘从A座借助C座先移到B座上
 - (2) 把A座上剩下的一个盘移到C座上
 - (3) 将 $n-1$ 个盘从B座借助于A座移到C座上

经典递归问题：汉诺塔

- 可以将第(1)步和第(3)步表示为：
 - ◆ 将 “one” 座上 $n-1$ 个盘移到 “two” 座(借助 “three” 座)。
 - ◆ 在第(1)步和第(3)步中，one、two、three和A、B、C的对应关系不同。
 - ◆ 对第(1)步，对应关系是one对应A，two对应B，three对应C。
 - ◆ 对第(3)步，对应关系是one对应B，two对应C，three对应A。

经典递归问题：汉诺塔

➤ 把上面3个步骤分成两类操作：

- (1) 将 $n-1$ 个盘从一个座移到另一个座上 ($n > 1$)。这就是大和尚让小和尚做的工作，它是一个递归的过程，即和尚将任务层层下放，直到第64个和尚为止。
- (2) 将1个盘子从一个座上移到另一座上。这是大和尚自己做的工作。

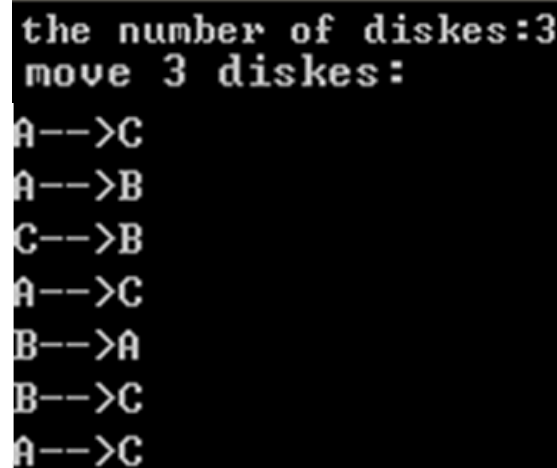
经典递归问题：汉诺塔

➤ 编写程序。

- ◆ 用hanoi函数实现第1类操作（即模拟小和尚的任务）
- ◆ 用move函数实现第2类操作（模拟大和尚自己移盘）
- ◆ 函数调用hanoi(n,one,two,three)表示将n个盘子从“one”座移到“three”座的过程(借助“two”座)
- ◆ 函数调用move(x,y)表示将1个盘子从x座移到y座的过程。x和y是代表A、B、C座之一，根据每次不同情况分别取A、B、C代入

经典递归问题：汉诺塔

```
1. #include <stdio.h>
2. int main()
3. {
4.     void hanoi(int n, char from, char transfer, char to);
5.     int m;
6.     printf("the number of disks:");
7.     scanf("%d",&m);
8.     printf("move %d disks:\n",m);
9.     hanoi(m, 'A', 'B', 'C');
10.    return 0;
11.}
12.void hanoi(int n, char from, char transfer, char to)
13.{
14.    void move(char x,char y);
15.    if (n==1)
16.        move(from,to);
17.    else
18.    {
19.        hanoi(n-1,from,to,transfer);
20.        move(from,to);
21.        hanoi(n-1,transfer,from,to);
22.    }
23.}
24.void move(char from,char to) { printf("%c-->%c\n",from,to); }
```



the number of disks:3
move 3 disks:
A-->C
A-->B
C-->B
A-->C
B-->A
B-->C
A-->C

作业 2017/11/21

➤ 按下列要求编写程序，提交手写源代码

1. 写两个函数，分别求两个整数的最大公约数和最小公倍数，用主函数调用者两个函数，并输出结果。两个整数由键盘输入。
2. 写一个判素数的函数，在主函数输入一个整数，输出是否为素数的信息

➤ 上机练习（不用交）：本讲义例程，教材第七章2，4，8，12.

重要通知：关于期中考预告

➤ 时间：11.26 周日上午（8:00-10:00）

➤ 地点：海韵教学楼307

➤ 范围：到第六章 数组

➤ 形式：笔试

➤ 心情：感觉鸭梨山大吗？

◆ 其实没什么好担心的...

◆ 详见万能治愈图→_→

