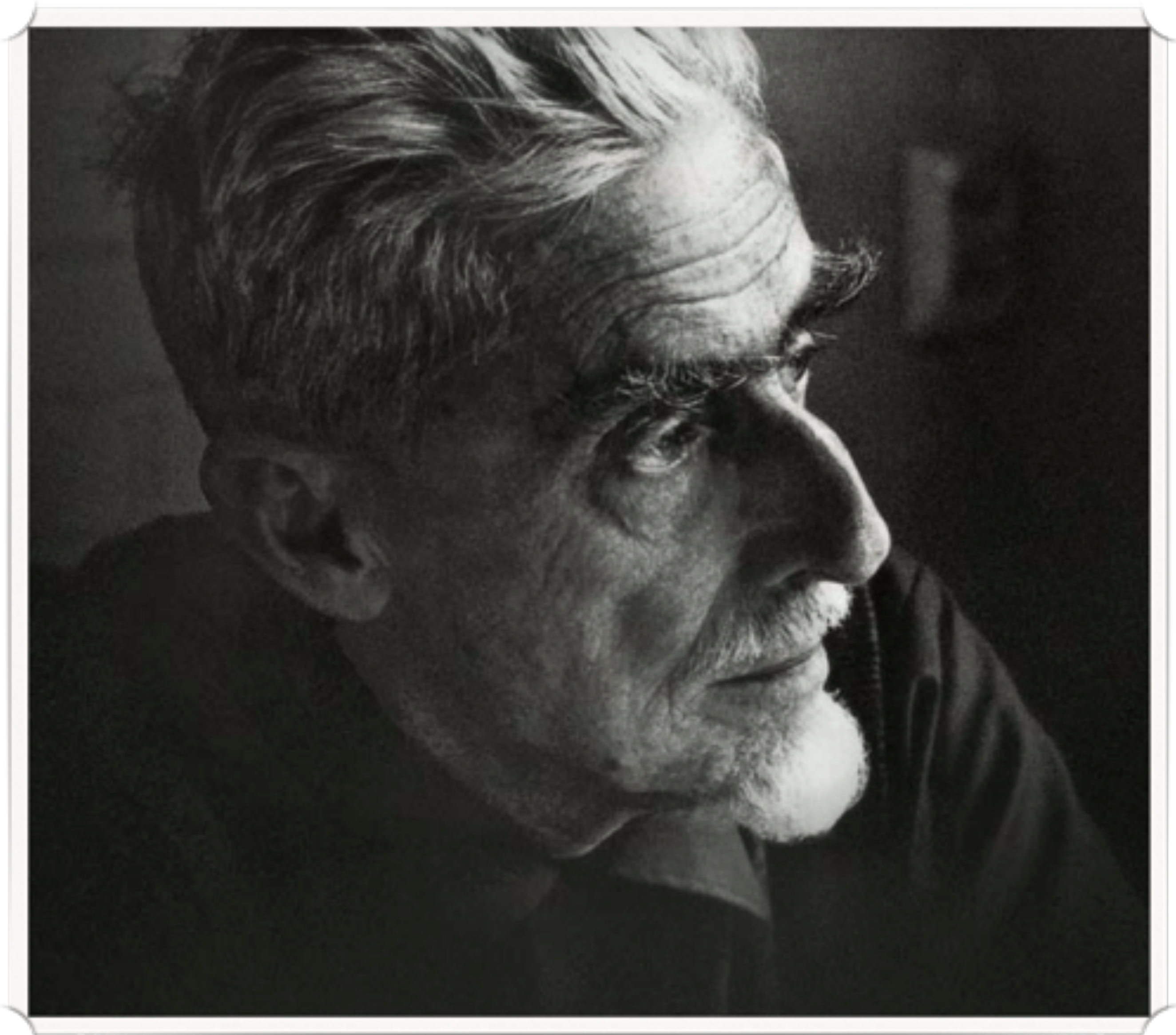
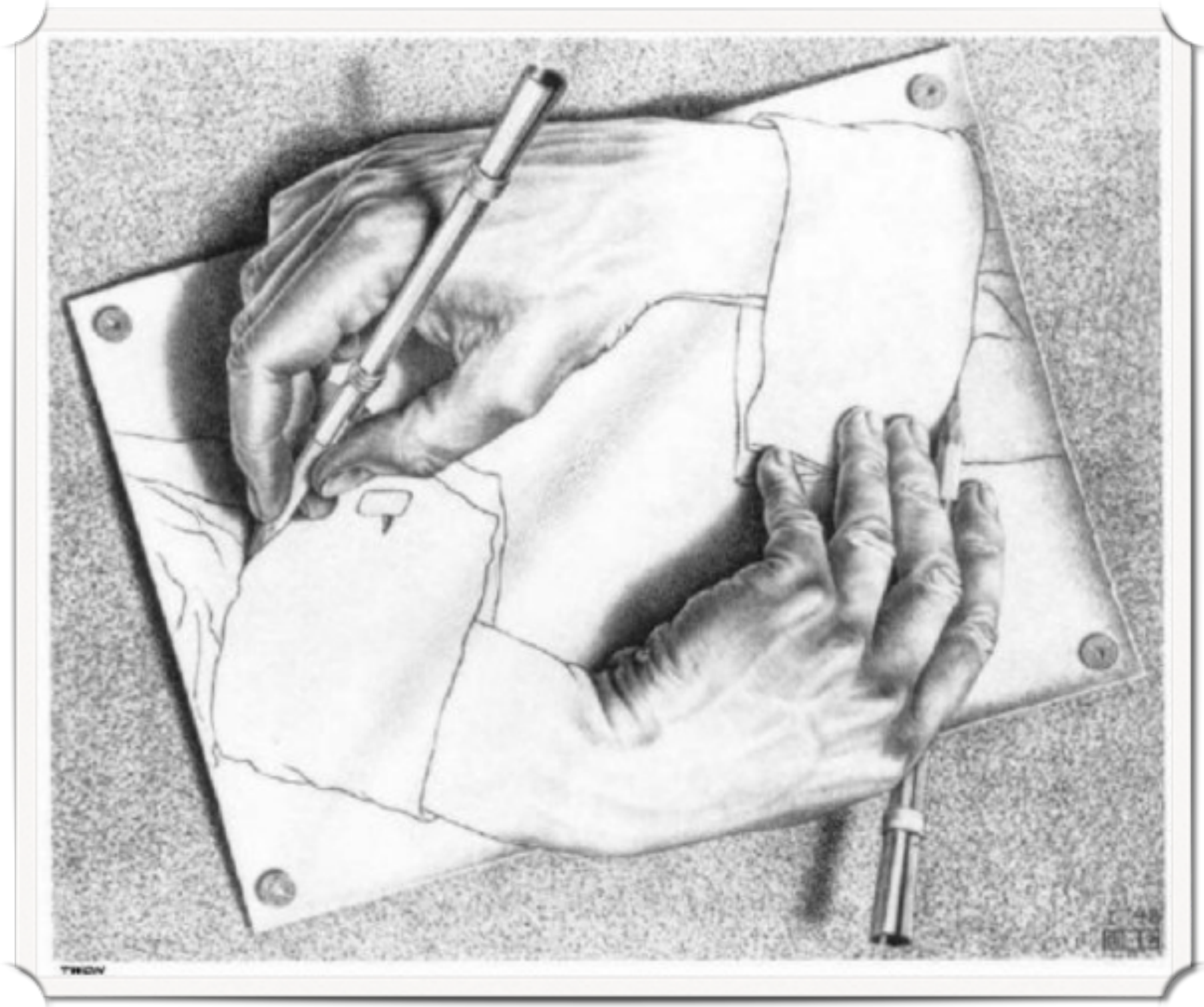
递归

RECURSION

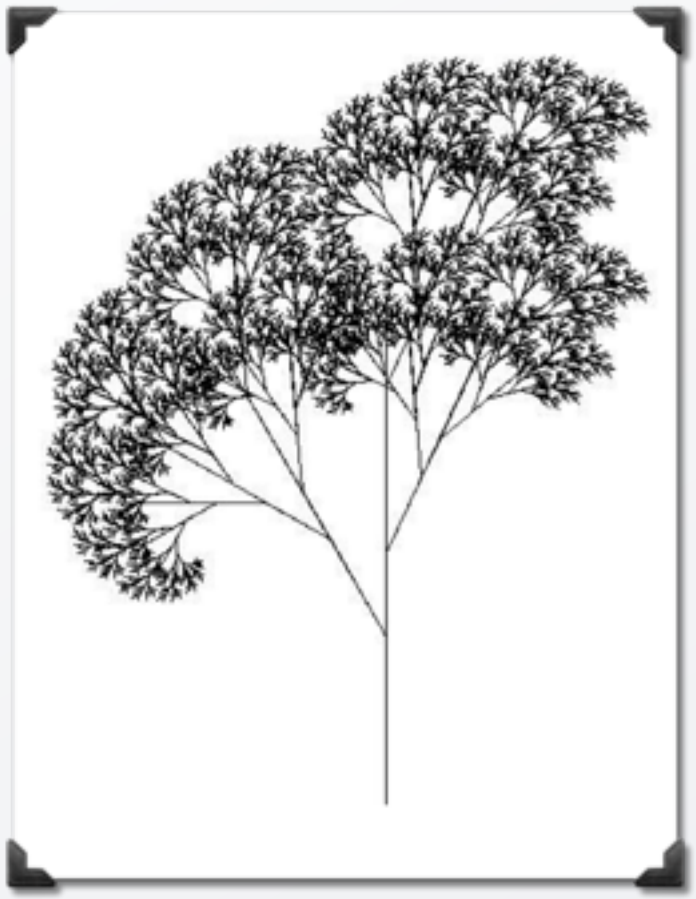
“The power of recursion evidently lies in the possibility of defining an infinite set of objects by a finite statement. In the same manner, an infinite number of computations can be described by a finite recursive program, even if this program contains no explicit repetitions."



Maurits Cornelis Escher (1898-1972)

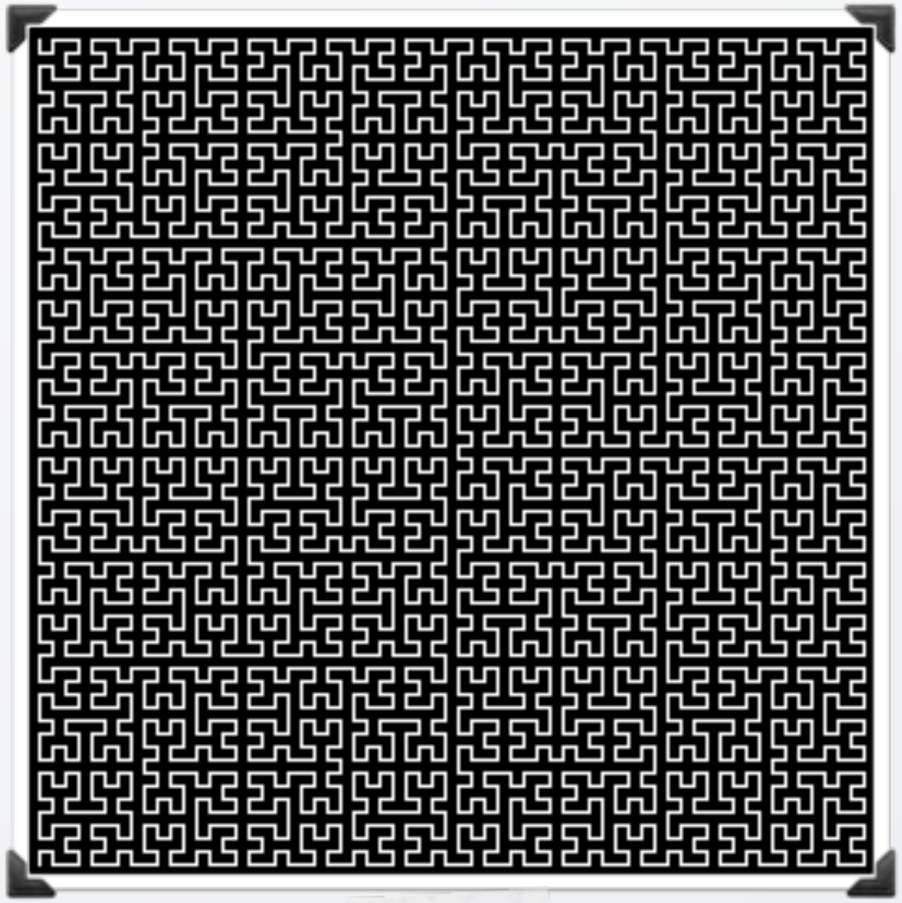


采用递归函数绘制的图形



Sielpinski Triangle

采用递归函数绘制的图形



Hilbert Curve

递归函数

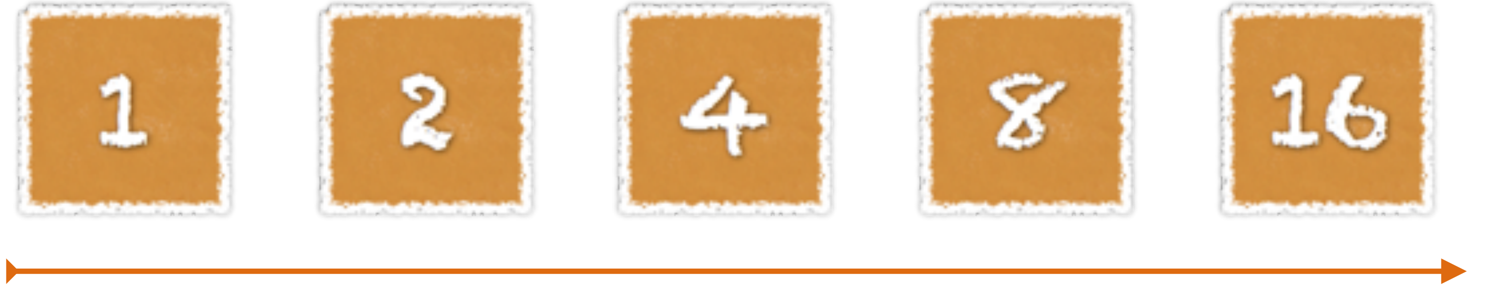
RECURSIVE FUNCTION

回忆曾经学过的数列

如果我们知道数列$\{a_n\}$的首项$a_1=1$，第n项和第$n-1$项有递推公式：

$$a_n = 2a_{n-1}$$

请问a_5应该是？

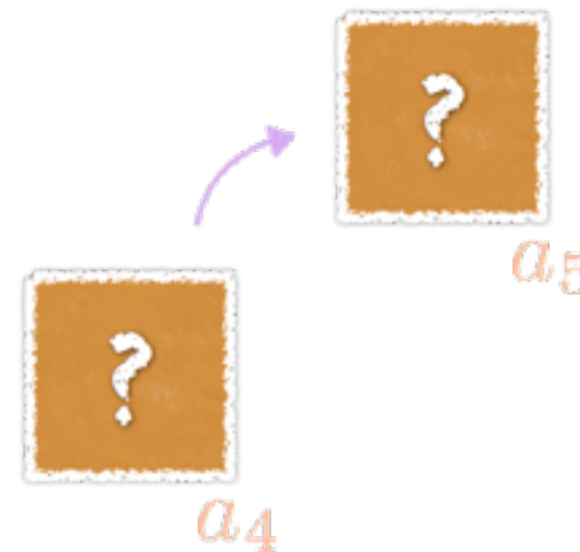

1
2
4
8
16

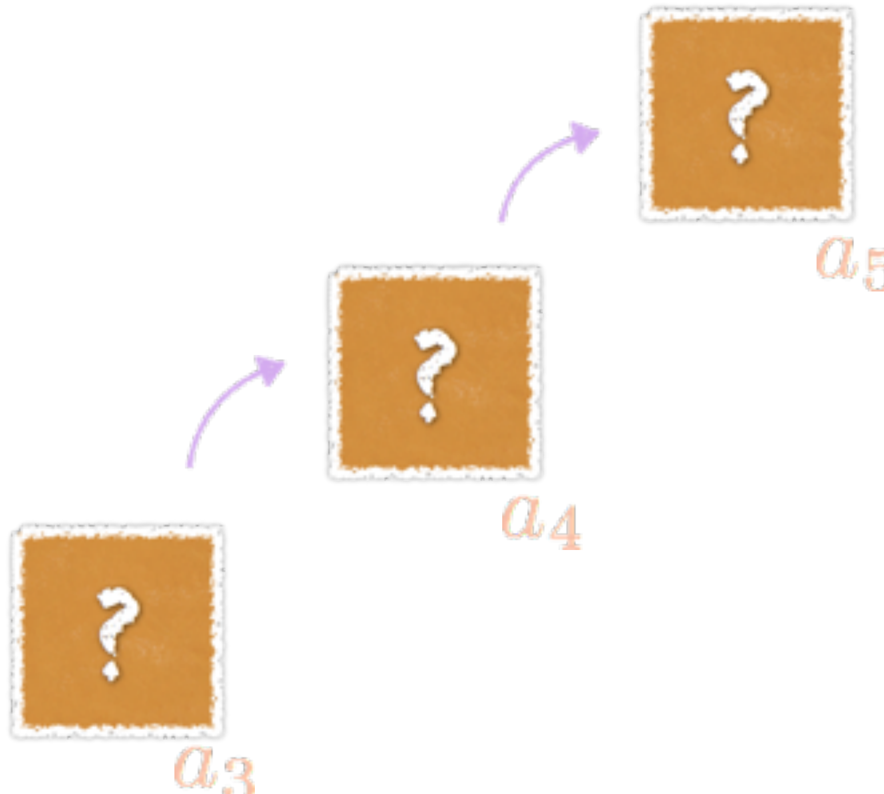
16

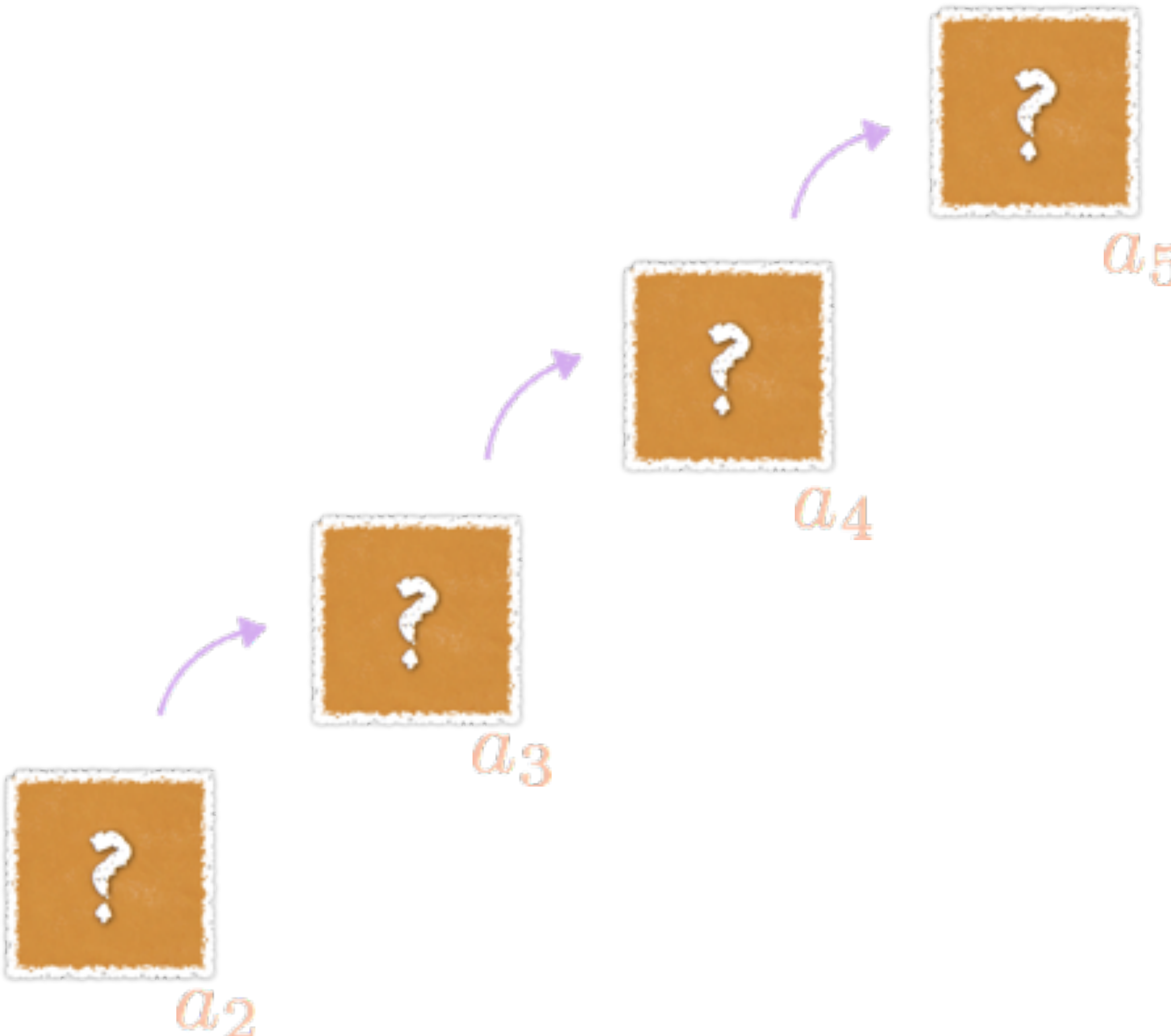
a_5

我们能不能从第5项开始思考呢?

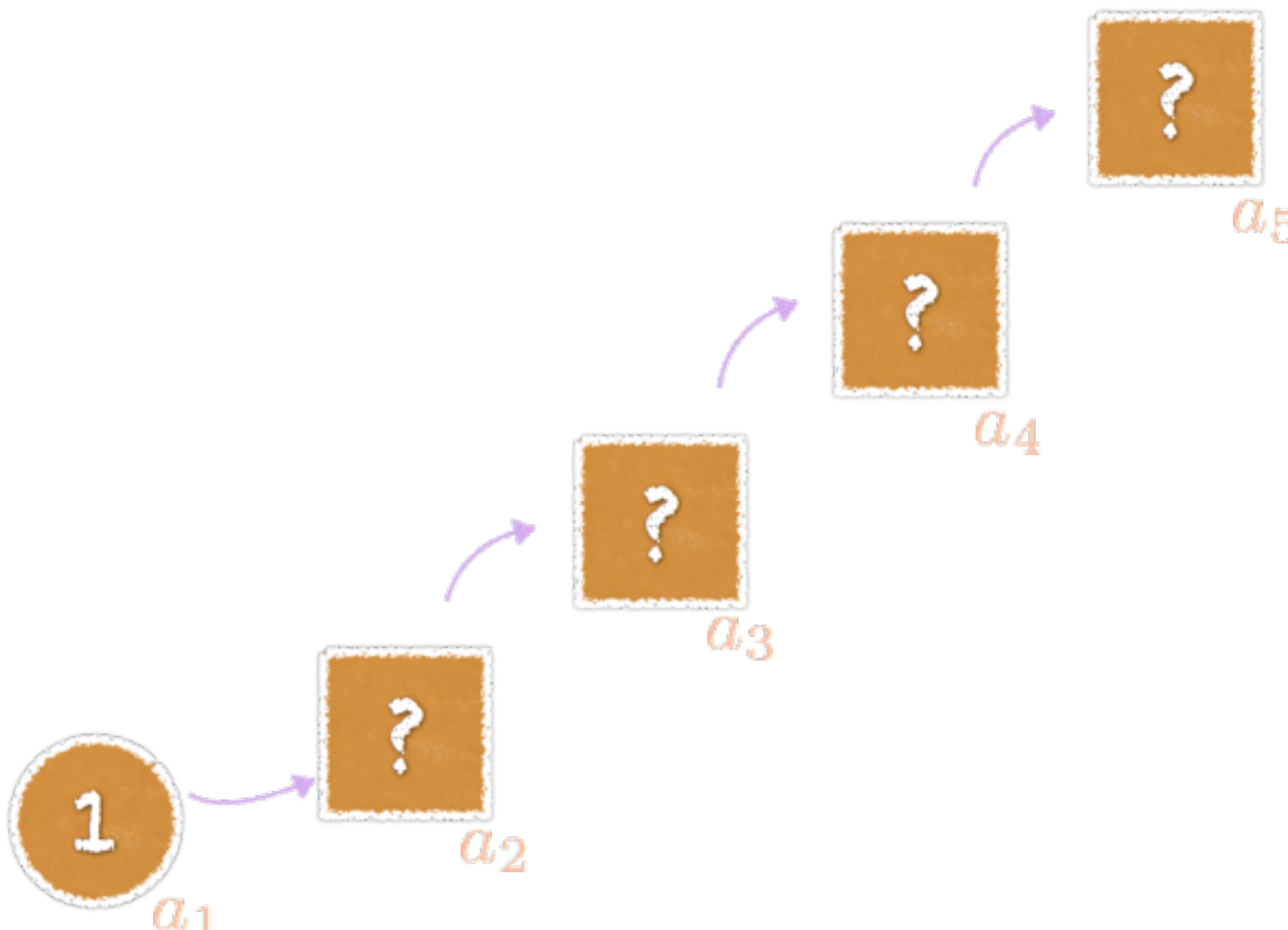
a_5


a_5
a_4

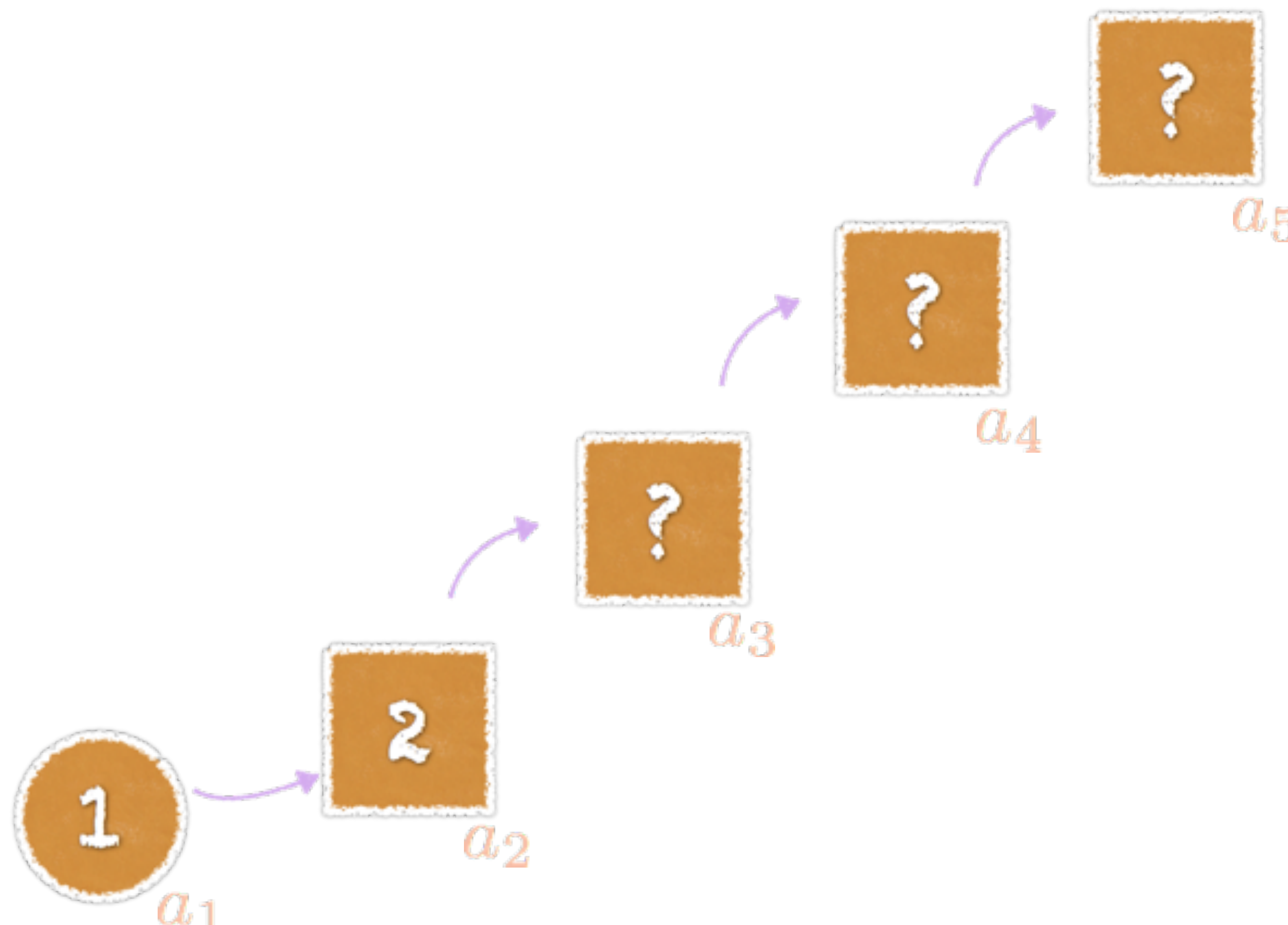

a_3
a_4
a_5

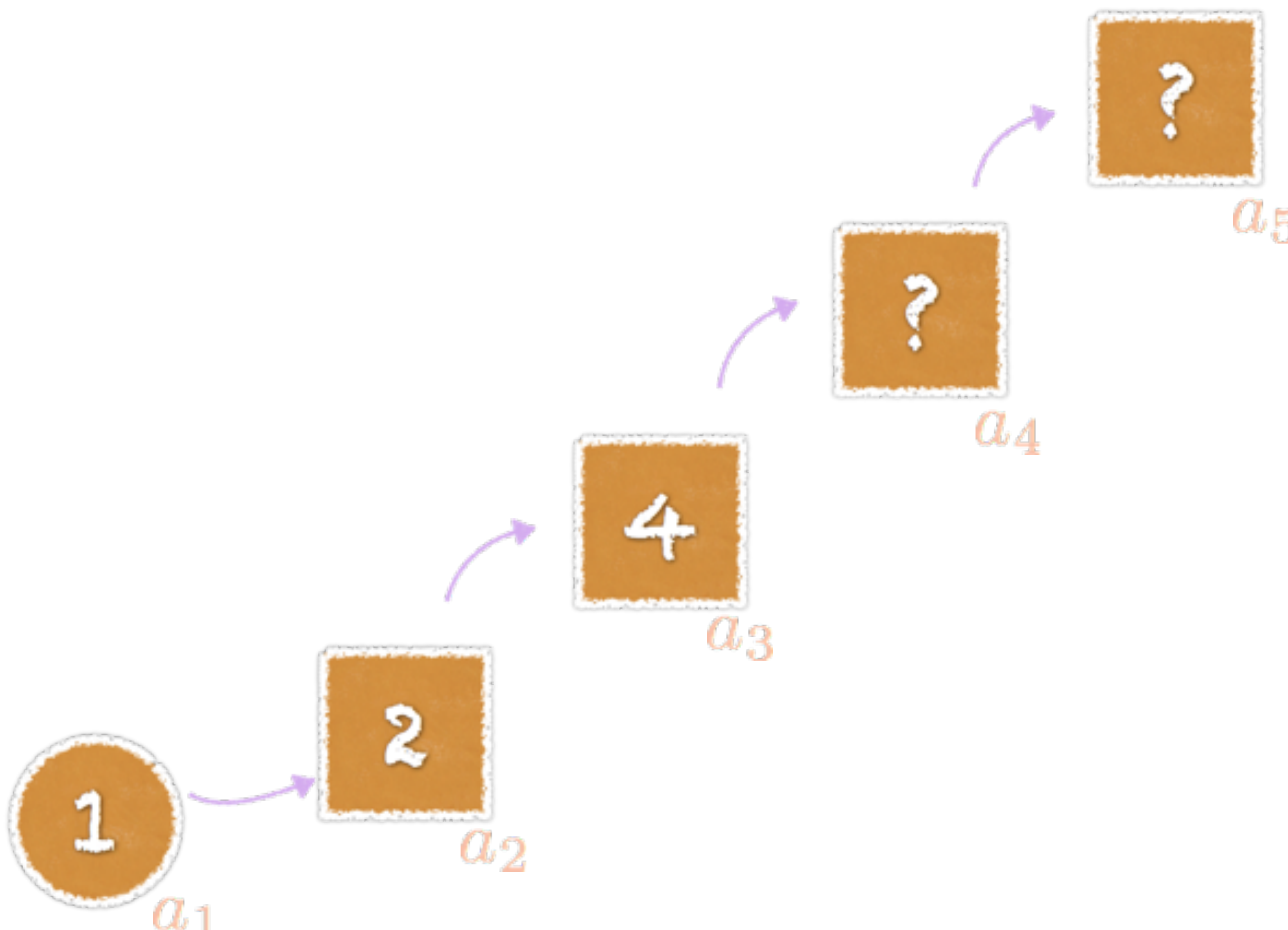


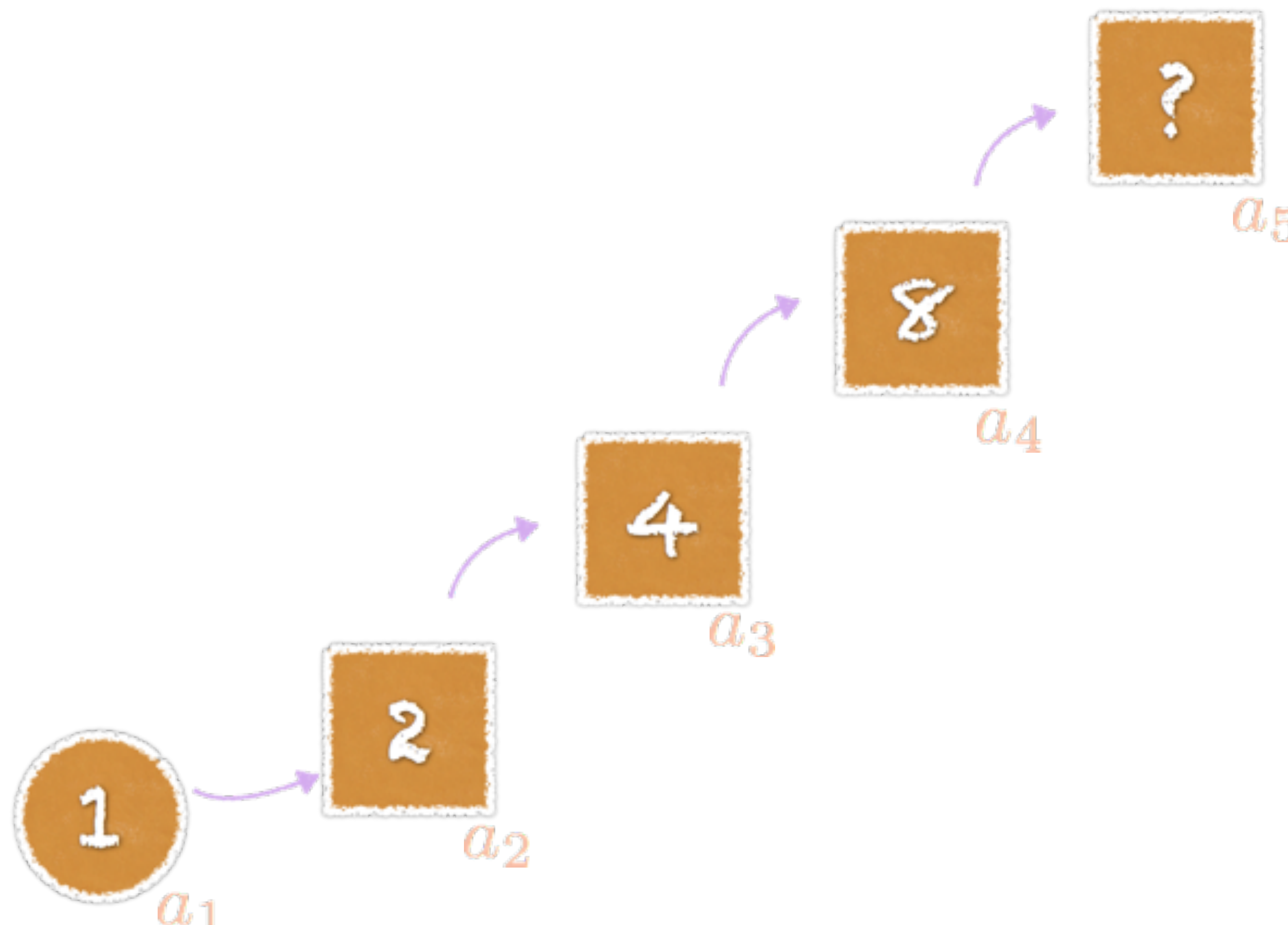
a_2
a_3
a_4
a_5



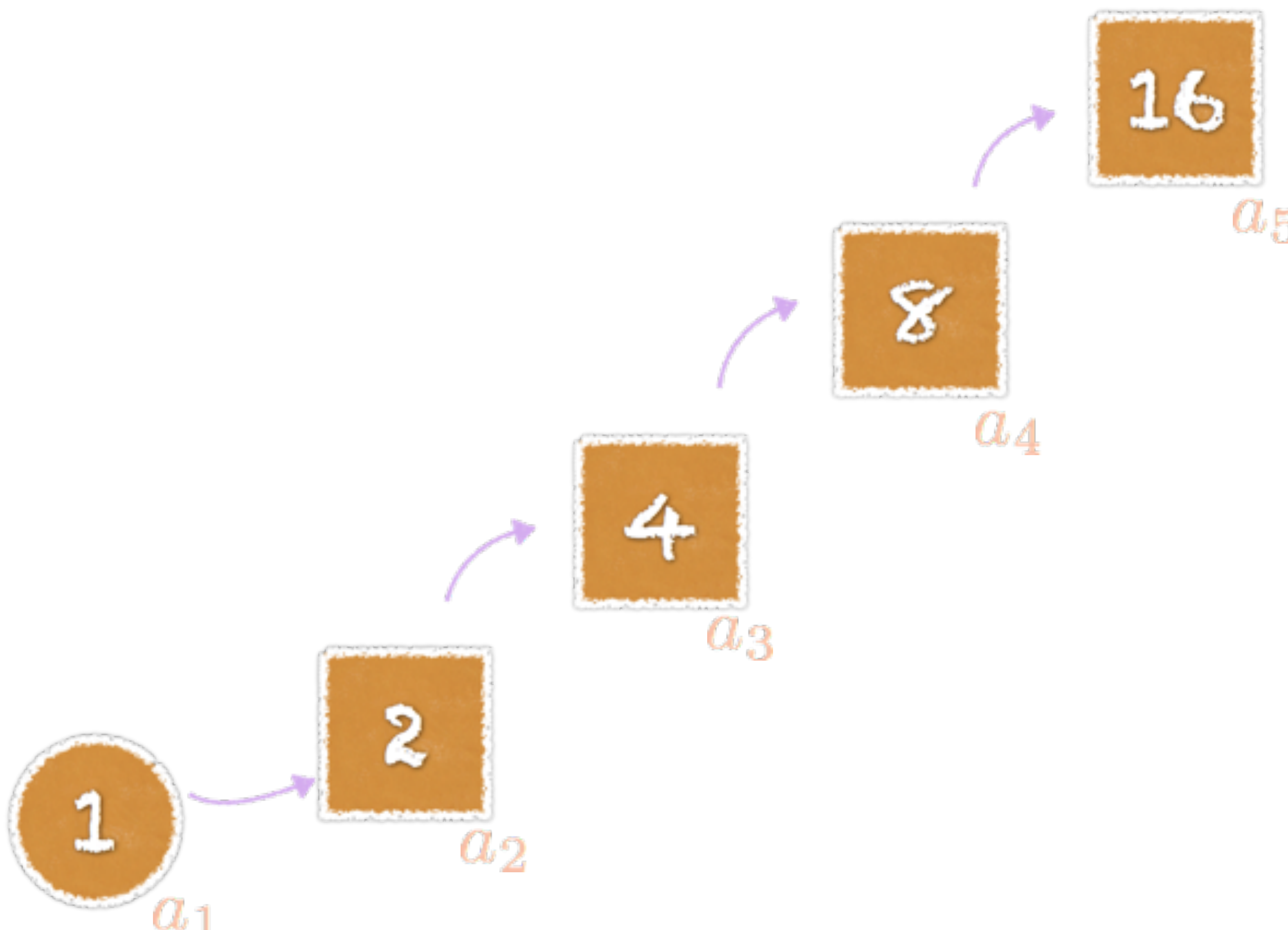
1
?
?
?
?
a_1
a_2
a_3
a_4
a_5


1
2
?
?
?
a_1
a_2
a_3
a_4
a_5


1
2
4
?
?
a1
a2
a3
a4
a5



1
2
4
8
?
a_1
a_2
a_3
a_4
a_5


1
a_1
2
a_2
4
a_3
8
a_4
16
a_5

16

a_5

我们把求解a_5的问题转化成求解a_4的问题，进而又依次转化成求解a_3 a_2 a_1的问题。

问题的规模依次减小，直到我们可以解决。

注意，求解问题a_5和子问题a_4的方法是完全一致的。

使用递归思想把问题分解为子问题，并采用与求解当前问题完全相同的方法去求解子问题。

递归函数

C语言函数定义的代码中，允许出现对当前定义函数的调用语句。这种函数称为递归函数。

算术运算符

add sub mul div mod

加法函数 add

$$f(a,b)=\begin{cases} a & b=0 \\ f(a+1,b-1) & b>0 \end{cases}$$

加法函数 add

```
int
add(int a, int b) {
  assert(b >= 0);
  return b ? add(++a, --b) : a;
}
```

加法函数 add

```
int
add(int a, int b) {
  assert(b >= 0);
  return b ? add(++a, --b) : a;
}
```

算术运算符

add sub mul div mod

阶乘函数 factorial

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

阶乘函数 factorial

```
unsigned int
factorial(unsigned int n) {
  if (n == 0)
    return 1;
  else
    return mul(factorial(sub(n, 1)), n);
}
```

求最大公约数

$$g(a,b) = \begin{cases} a & b = 0 \\ g(b, \quad a \mod b) & b > 0 \end{cases}$$

求最大公约数

```c
unsigned int
gcd(unsigned int a, unsigned int b) {
  return b ? gcd(b, a % b) : a;
}
```

位运算符

~	**&**	**|**
取反	与	或

判别一个正整数是不是一个回文正整数。

例如：

123321	TRUE
23	FALSE
9	TRUE
6547456	TRUE

计算一个整数的位数

$$c(n,b) = \begin{cases} 0 & n = 0 \\ 1 + c(\frac{n}{b}, b) & n > 0 \end{cases}$$

- 对于10进制数， b = 10

计算一个整数的位数

```
unsigned int
count_digits(unsigned int n, unsigned int base) {
  if (n == 0) return 0;
  return add(1, count_digits(div(n, base), base));
}
```

快速乘方计算

$$p(x,n) = \begin{cases} 1 & n = 0 \\ p(x, \frac{n}{2})^2 & n = 2k \\ xp(x, n-1) & n = 2k+1 \end{cases}$$

$$k \in N$$

快速乘方计算

```c
// Fast Exponentiation Algorithms
int
power_recursive(int base, unsigned int exponent) {
  if (exponent == 0) return 1;

  return (exponent & 1) ? mul(power_recursive(base, --exponent), base)
                        : square(power_recursive(base, div(exponent, 2)));
}
```

逆转数位

```
unsigned int
reverse_digits(unsigned int n) {
  unsigned int leftpart, rightpart;
  assert(n >= 0);
  if (n < 10) return n;
  leftpart = div(n, 10);
  rightpart = mod(n, 10);
  return add(mul(rightpart, power_recursive(10, (count_digits(leftpart, 10)))),
             reverse_digits(leftpart));
}
```

潮随暗浪雪山倾，远浦渔舟钓月明。
桥对寺门松径小，槛当泉眼石波清。
迢迢绿树江天晚，霭霭红霞晓日晴。
遥望四边云接水，雪峰千点数鸥轻。

轻鸥数点千峰雪，水接云边四望遥。
晴日晓霞红霭霭，晚天江树绿迢迢。
清波石眼泉当槛，小径松门寺对桥。
明月钓舟渔浦远，倾山雪浪暗随潮。

苏轼

NEVER ODD OR EVEN

判断回文数Palindrome

```
bool
is_palindrome(unsigned int n) {
	return (n == reverse_digits(n));
}
```

小题狂做

设计一个**递归函数**，用来判别一个正整数是不是一个回文正整数。

判断回文数Palindrome

递归版本

```
bool
is_palindrome_r (unsigned int n)
{
  if (count_digits (n, 10) == 0 || count_digits (n, 10) == 1)
    return true;

  unsigned int last_digit = mod (n, 10);
  unsigned int all_digit_but_last = div (n, 10);

  unsigned int all_digit_but_last_reversed = reverse_digits (all_digit_but_last);
  unsigned int first_digit = mod (all_digit_but_last_reversed, 10);

  unsigned int m = div (all_digit_but_last_reversed, 10);

  if (last_digit == first_digit && is_palindrome_r (m))
    return true;

  return false;
}
```

整理一下我们的代码…

```c
#include <assert.h>
#include <stdbool.h>

int
add (int a, int b)
{
  assert (b >= 0);
  return b ? add (++a, --b) : a;
}


int
sub (int a, int b)
{
  return add (-b, a);
}
```

```c
int
mul (int a, int b)
{
  assert (b >= 0);
  if (b == 0)
    return 0;

  return add (mul (a, sub (b, 1)), a);
}

int
div (int a, int b)
{
  assert (b > 0 && a >= 0);
  if (a < b)
    return 0;

  return add (1, div (sub (a, b), b));
}
```

```c
int
mod (int a, int b)
{
  assert (a > 0 && b > 0);
  return a < b ? a : mod (sub (a, b), b);
}


unsigned int
square (int x)
{
  return mul (x, x);
}
```

```c
unsigned int
count_digits (unsigned int n, unsigned int base)
{
  if (n == 0)
    return 0;

  return add (1, count_digits (div (n, base), base));
}


int
power_recursive (int base, unsigned int exponent)
{
  if (exponent == 0)
    return 1;

  return (exponent & 1) ? mul (power_recursive (base, --exponent), base)
    : square (power_recursive (base, div (exponent, 2)));
}
```

```c
unsigned int
reverse_digits (unsigned int n)
{
  unsigned int leftpart, rightpart;

  assert (n >= 0);

  if (n < 10)
    return n;

  leftpart = div (n, 10);
  rightpart = mod (n, 10);

  return
    add (mul (rightpart, power_recursive (10, (count_digits (leftpart, 10)))),
         reverse_digits (leftpart));
}

bool
is_palindrome (unsigned int n)
{
  return (n == reverse_digits (n));
}
```

```c
int
main (int argc, char const *argv[])
{
  assert (is_palindrome (123321));
  return 0;
}
```

注意：编译增加 -O2 选项，以提高执行速度。

小结

1. 使用递归思想，程序设计者可以将复杂问题分解成完全相同、且规模更小的子问题，直到出现可解的终结子问题。

2. 终结子问题必须保证在有限步内获得解决。

WENZHENG COLLEGE OF SOOCHOW UNIVERSITY

2017.3.29