Welcome to CS 132 Computer Science II C++

Lonnie Heinke

Today's Agenda

- About your instructor
- Highlight syllabus
- Recursion
- Homework #1

While you are waiting for class to start:

- Please check with me if you want to add the class
- Log into the computers with your ***name*** from your ***name***@students.everettcc.edu account
- Start up Visual Studio 2017 and create a project called SandBox1

About me…..

- BS and MS in Computer Science
- Worked at HP as a programmer and then IT Specialist for about 5 years

- Taught Computer Programming for over 15+ years
  - 3 years at Cabrillo College in Aptos, California
  - 11 years at Tian Jin University in Tian Jin, China
  - 1 ½ years here at Everett Community College

Syllabus

- Homework: Read over syllabus tonight, and ask any questions next class

- I will gradually talk about the pieces as we hit them in the class

- New Things:
  - No computer usage during lecture time other than note taking or viewing the slides.
  - No printing during lecture time.
  - No cell phone usage during lecture or lab time. If you finish a lab early, then explore beyond the minimum. Break time is the only time for using phones

- To pass the class, you need a programming average and exam average of at least 70%

Non-discrimination

Everett Community College *does not* discriminate on the basis of race, religion, creed, color, national origin, age, sex, sexual orientation, gender identity or gender expression, marital status, disability, genetic information or status as a veteran of war as required by law.

My desire, rather than look at this in what we don’t do, is to instead list it as what my goal is to do:

Respect everyone and promote an environment where **everyone is comfortable** and can flourish in their learning.

???

- Who is responsible for your education?

Owning your Education

- Who is responsible for your education?
  - **You** are responsible for your education.
- You are not alone, You have plenty of help:
  - classmates,
  - student assistants
  - teacher
- But what does it mean to be responsible for your education......
  ......to own your education ?

Activity

- Take a couple minutes to think of what worked and didn’t work for your learning in previous programming class.
- Write down some of the things that didn’t help your learning.
- Write down some of the things that really helped your learning.
- What are some things that you haven’t tried, but you think could help with your learning

Each of us comes to a new learning experience with some baggage…..good and bad

I’m good at programming

I’m a terrible student

Teachers like me

Computers hate me

Growth Mindset

- I’m new to this……….I’ll make mistakes.
- I’m learning ……….there will be times of confusion
- I’m not there yet ……….have fun in the journey.

This mindset reminds you that learning is a process of *exploring, making mistakes, trying new things*….it is more of a journey than a destination.

Struggling is just a part of the journey.....if you aren’t struggling then you may be doing something wrong

My board gaming mantra

- In reference to learning or playing a new game:

“The first three games don’t count”

In other words:
Have fun,
Experiment,
Don’t worry about playing well.

How to be successful with your C++ journey

- Start programs early, and work on them often
- Start programs early, and work on them often
- Start programs early, and work on them often
- Start programs early, and work on them often
- Start programs early, and work on them often
    - Start programs early, and work on them often
    - Start programs early, and work on them often
    - Start programs early, and work on them often

Be actively engaging with the ideas, and start programs early, and work on them often

How to be successful with your C++ journey

- Growth mindset....experiment, willing to make mistakes
- Read the book
- Start programs early, and work on them often
- Experiment with lots of code….beyond assignments and labs
- Ask questions…discuss the idea you are acquiring
    - From reading
    - During class
    - Outside class

Be actively engaging with the ideas, and take responsibility for your education

My Goal

- My personal goal for the class is that you will:
  - Grow more proficient with C++
  - Be comfortable and confident in using it

- I will make every effort that is appropriate as a teacher to help you learn

…but you are still the one who has to do the learning on this journey

Beyond, there be dragons…

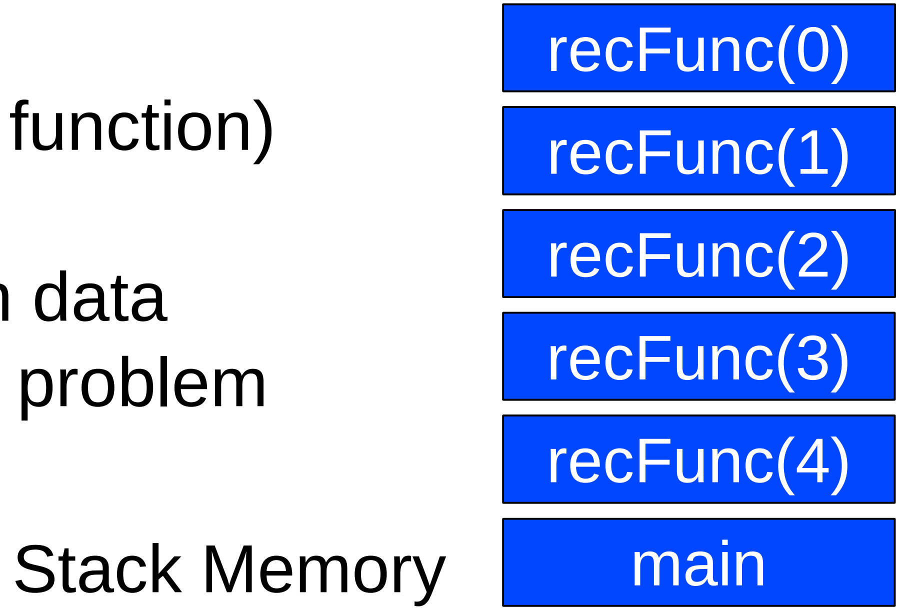
Questions…..

Recursion

- New Topic or Review ???
- What is Recursion ?

Recursion

- What is Recursion ?
  - a programming solution that uses a function that calls itself
  - kind of like looping but using functions instead
  - Picture this, a function has
    - the same algorithm (since it is just one function)
    - but each function call can have it’s own data
      - be working on a different part of the problem



recFunc(0)

recFunc(1)

recFunc(2)

recFunc(3)

recFunc(4)

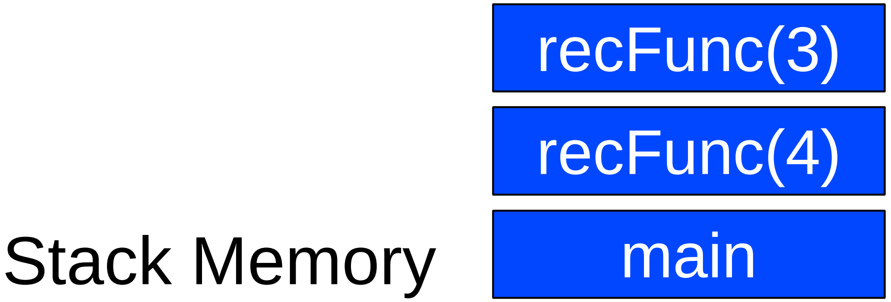
main

Stack Memory

Recursion

- Good for really understanding function calls

```
int recFunc( int num ) {      // recFunc is short of recursive function
    int localVar(num);
    …..
    localVar = recFunc( localVar );
    …..
    return localVar;
}
```



recFunc(3)

recFunc(4)

Stack Memory

main

Setting up a Recursive Solution

- Look for an ending condition

- Look for the repeating pattern

- Remember that when a function finishes, it returns and continues execution from it calling point

Simple Example: Factorial

4! = 4 * 3 * 2 * 1

n! is the **product** of all positive integers less than or equal to n

where 0! = 1

Simple Example: Factorial

Another way to think of this...

possibly a recursively way

n! is the **product** of all positive integers less than or equal to n

where 0! = 1

4! = ??

4! = 4 * 3!

Simple Example: Factorial

4! = ??

4! = 4 * 3!

3! = 3 * 2!   // pattern ?

n! is the **product** of all positive integers less than or equal to n

where 0! = 1

Simple Example: Factorial

4! = ??

n! is the product of all positive integers less than or equal to n

where 0! = 1

4! = 4 * 3!

3! = 3 * 2!   // pattern ?

n! = n * (n-1)!   where 0! = 1 (our end condition)

Converting to code

4! = ??

4! = 4 * 3!

3! = 3 * 2!

n! = n * (n-1)! where 0! = 1 // our end condition

```
int factorial( int num) {
     if ( num == 0 )
          return 1;

}
```

Converting to code

4! = ??

4! = 4 * 3!

3! = 3 * 2!

n! = n * (n-1)! where 0! = 1 // our end condition

the repeating pattern

```
int factorial( int n ) {
     if ( n == 0 )
          return 1;
     else
          return n *  factorial( n - 1);
}
```

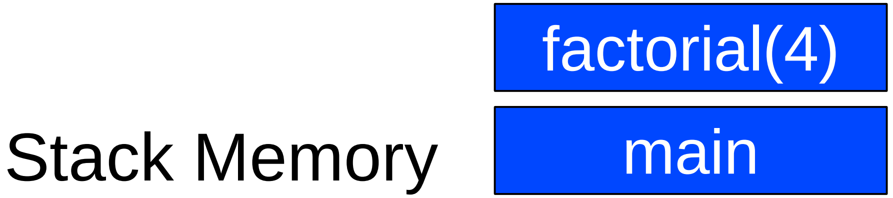
Converting to code

4! = ??

// in main

cout << factorial (4) << endl;

```
int factorial( int n) {
     if ( n == 0 )
         return 1;
     else
         return n *  factorial( n - 1);
}
```



Stack Memory

factorial(4)

main

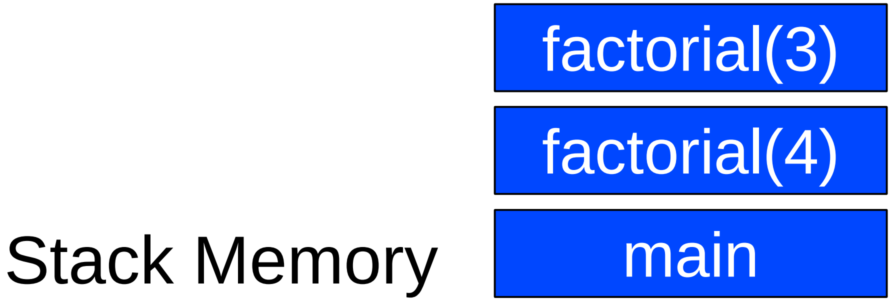
Converting to code

4! = ??

// in main

cout << factorial (4) << endl;

```
int factorial( int n) {
     if ( n == 0 )
         return 1;
     else
         return n *  factorial( n - 1);
}
```



Stack Memory

factorial(3)

factorial(4)

main

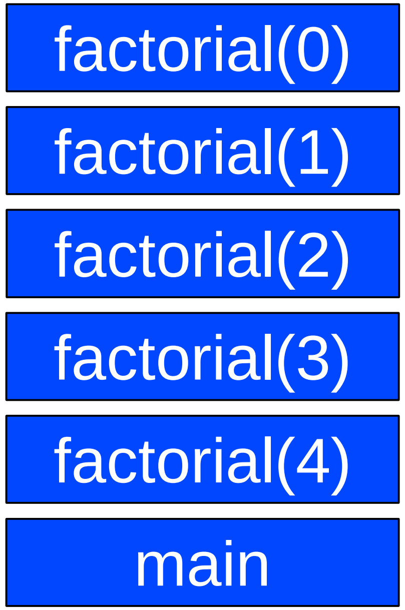
Converting to code

4! = ??

```
// in main


cout << factorial ( 4 ) << endl;
```

```
int factorial( int n) {
    if ( n == 0 )
        return 1;
    else
        return n *  factorial( n - 1);
}
```



Stack Memory

Converting to code

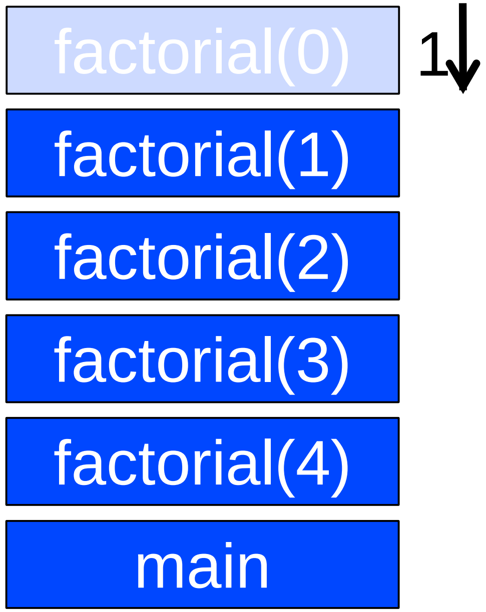
4! = ??

```
// in main


cout << factorial ( 4 ) << endl;
```

```
int factorial( int n) {
     if ( n == 0 )
         return 1;
     else
         return n *  factorial( n - 1);
}
```

Stack Memory



Converting to code

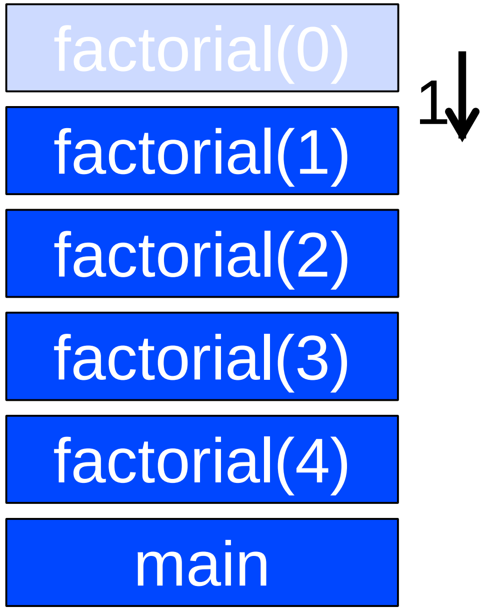
4! = ??

```
// in main


cout << factorial ( 4 ) << endl;
```

```
int factorial( int n) {  \\ n = 1
     if ( n == 0 )
         return 1;
     else
         return n *  1 ~~factorial( n - 1)~~;
}
```

Stack Memory



remember the returned
1 replaces the function
call in that statement

Converting to code

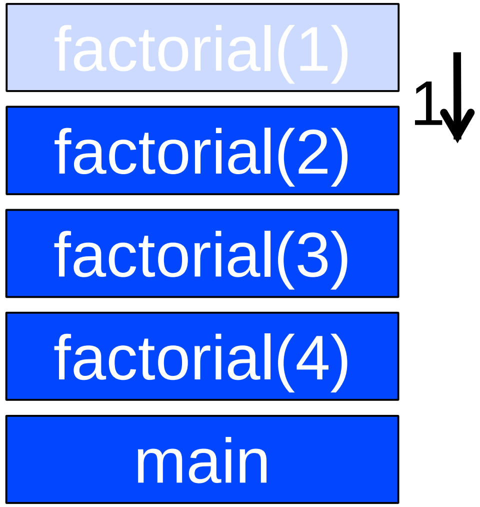
4! = ??

// in main

cout << factorial (4) << endl;

```
int factorial( int n) {  \\ n = 2
     if ( n == 0 )
         return 1;
     else
         return n * 1 ~~factorial( n - 1)~~;
}
```

Stack Memory



Converting to code

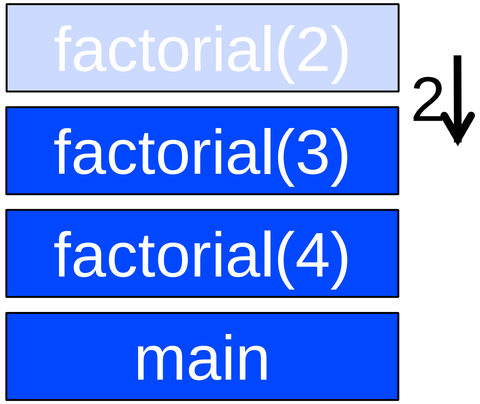
4! = ??

// in main

cout << factorial (4) << endl;

```
int factorial( int n) {  \\ n = 3
    if ( n == 0 )
        return 1;
    else
        return n * 2 ~~factorial( n - 1)~~;
}
```

Stack Memory



Converting to code

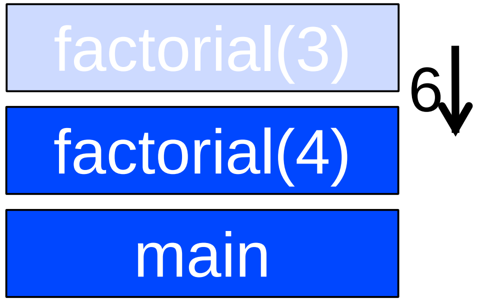
4! = ??

```
// in main


cout << factorial ( 4 ) << endl;
```

```
int factorial( int n) {  \\ n = 4
    if ( n == 0 )
        return 1;
    else
        return n * 6 ~~factorial( n - 1)~~;
}
```

Stack Memory



Converting to code

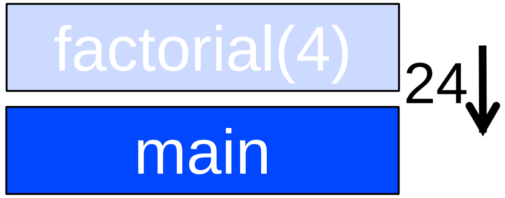
4! = ??

```
// in main


cout << 24 ~~factorial ( 4 )~~ << endl;
```

```
int factorial( int n) {  \\ n = 4
     if ( n == 0 )
         return 1;
     else
         return n * 6 ~~factorial( n - 1)~~;
}
```

Stack Memory



Good example from our book

```
void write_vertical( int num)
```

```
write_vertical( 1234 );    // example function call
```

Output

```
1
2
3
4
```

What cases are easy to solve? **Certain numbers** or **parts of the number**

Good example from the book

```
void write_vertical( int num)
```

```
write_vertical( 1234 );    // example function call
```

Output

```
1
2
3
4
```

What cases are easy to solve? Certain numbers or parts of the number

Single digit number (< 10)

The last digit (num %10)

Good example from the book

Output

```
1
2
3
4
```

```
void write_vertical( int num)


write_vertical( 1234 );    // example function call
```

What cases are easy to solve? Certain numbers or parts of the number

Single digit number (< 10)   The last digit (num %10)

Ending Condition??   Repeating pattern??

Good example from the book

Output

```
1
2
3
4
```

```
void write_vertical( int num)

write_vertical( 1234 );    // example function call
```

What cases are easy to solve? Certain numbers or parts of the number

Single digit number (< 10) The last digit (num %10)

Ending Condition??

Single digit numbers

Repeating pattern??

call function for all digits except last digit

print out the last digit

Good example from the book

```
void write_vertical( int num){
    if (num < 10 ){                              // end condition
          cout << num << endl;
    }
    else {                                       // repeating pattern
          write_vertical( num / 10 );
          cout << num % 10 << endl;
    }
}

write_vertical( 1234 );    // example function call
```

Output

```
1
2
3
4
```

Ending Condition??

Single digit numbers

Repeating pattern??

call function for all digits except last digit

print out the last digit

Good example from the book

```
void write_vertical( int num){
    if (num < 10 ){                    // end condition
          cout << num << endl;
    }
    else {                             // repeating pattern
          write_vertical( num / 10 );
          cout << num % 10 << endl;
    }
}


write_vertical( 1234 );    // example function call
```

Output

```
1
2
3
4
```

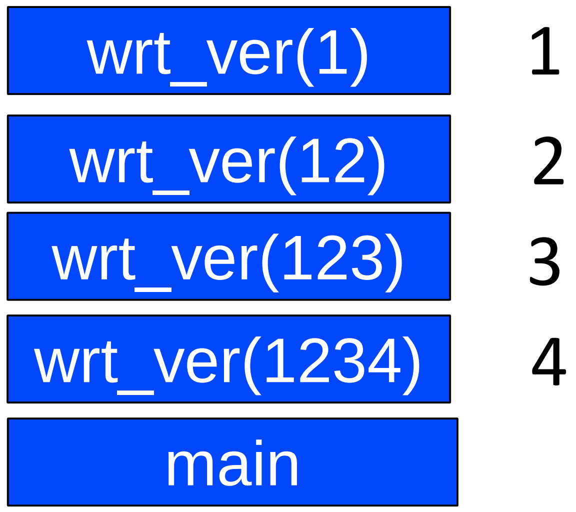
How many times will the function be called and in which function call is each digit output?

Good example from the book

```
void write_vertical( int num){
    if (num < 10 ){                    // end condition
        cout << num << endl;
    }
    else {                             // repeating pattern
        write_vertical( num / 10 );
        cout << num % 10 << endl;
    }
}


write_vertical( 1234 );    // example function call
```

How many times will the function be called and in which function call is each digit output?



wrt_ver(1) 1

wrt_ver(12) 2

wrt_ver(123) 3

wrt_ver(1234) 4

main

Stack Memory

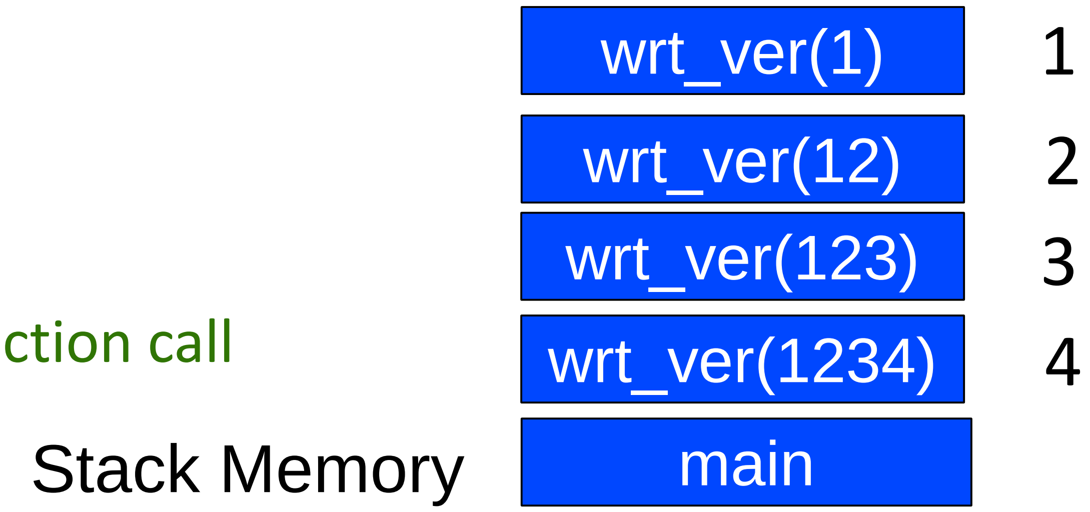
Good example from the book

```
void write_vertical( int num){
    if (num < 10 ){                  // end condition
        cout << num << endl;
    }
    else {                           // repeating pattern
        cout << num % 10 << endl;
        write_vertical( num / 10 );
    }
}

write_vertical( 1234 );    // example function call
```

If we change the order of these two lines

Now what will the output be?



wrt_ver(1)	1
wrt_ver(12)	2
wrt_ver(123)	3
wrt_ver(1234)	4
main	

Stack Memory

Good example from the

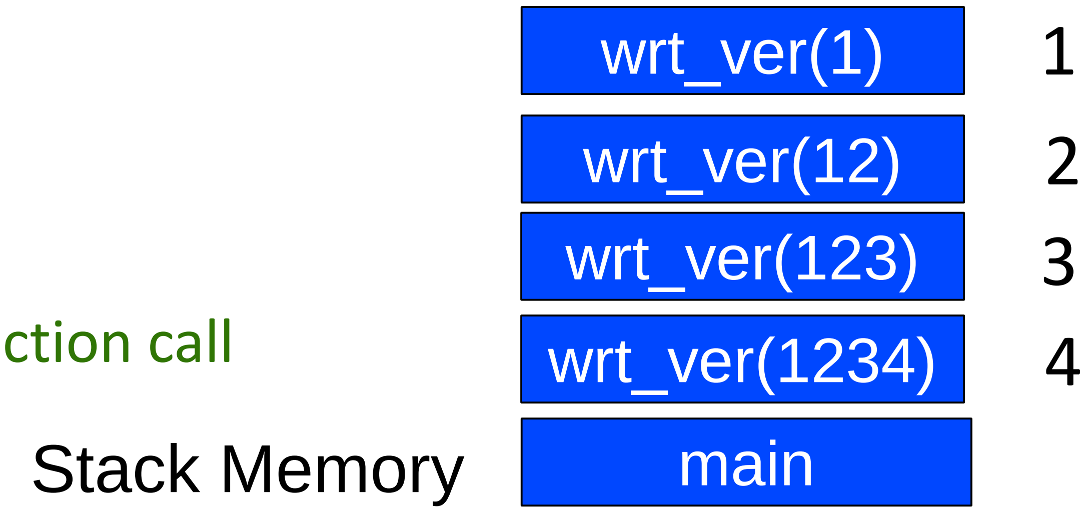
Now what will the output be?

```
void write_vertical( int num){
    if (num < 10 ){                // end c
        cout << num << endl;
    }
    else {                         // repeating pattern
        cout << num % 10 << endl;
        write_vertical( num / 10 );
    }
}


write_vertical( 1234 );   // example function call
```

Output

4
3
2
1

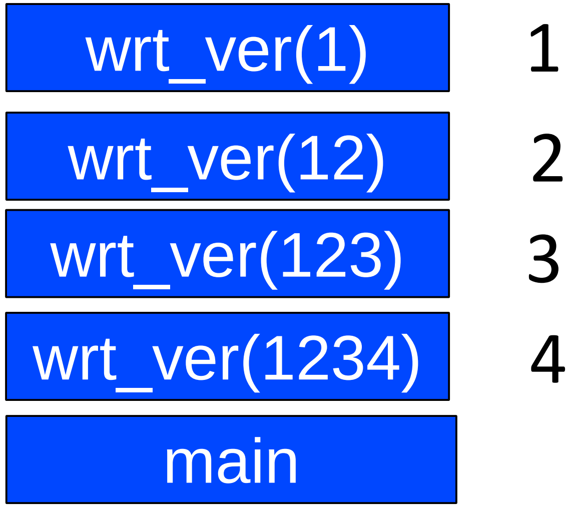


Good example from the book

```
void write_vertical( int num){
    if (num < 10 ){                    // end condition
          cout << num << endl;
    }
    else {                             // repeating pattern
          write_vertical( num / 10 );
          cout << num % 10 << endl;
    }
}


write_vertical( 1234 );    // example function call
```

I like this example because it shows us how each function is “paused” when it calls the next function



Stack Memory

wrt_ver(1) 1

wrt_ver(12) 2

wrt_ver(123) 3

wrt_ver(1234) 4

main

Setting up a Recursive Solution

- Look for an ending condition

- Look for the repeating pattern

- Remember that when a function finishes it returns and continues execution from it calling point

Another Example as a Programming Activity

- Reverse Printing a cstring without using loops

cPtr → “Everett Community College”

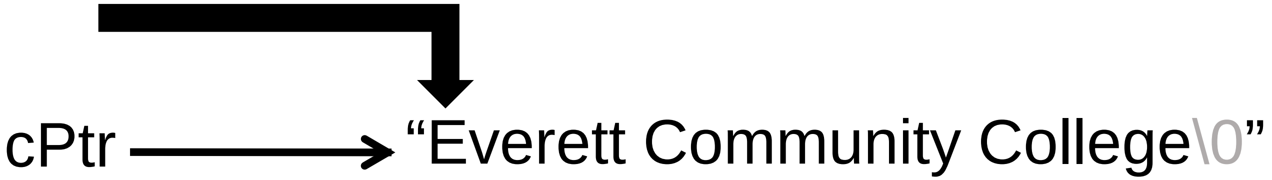
```
void reversePrint( char * ptr);
```

Imagine that we can still move the pointer forward, but for some reason moving backwards doesn’t work   (and indexing [ ] doesn’t work either)

```
cPtr + 1;    // works
cPtr -  1;   //  doesn’t work
```

Programming Activity:

- Reverse Printing a cstring without using loops



cPtr → "Everett Community College\0"

cstring has the null terminator

```
void reversePrint( char * ptr);
```

- What is the end condition?

- What is the repeating pattern?

Remember cPtr is just pointing to one char….the first one

Programming Activity

- Reverse Printing a cstring without using loops

cPtr “Everett Community College”

void reversePrint(char * ptr);

- What is the end condition?
  - null character '\0'
- What is the repeating pattern?

Programming Activity

- Reverse Printing a cstring without using loops

cPtr → “Everett Community College”

```
void reversePrint( char * ptr);
```

- What is the end condition?
  - null character '\0'

- What is the repeating pattern?
  - if not pointing to null, call the function for the next char
  - print one char

Programming Activity

- Reverse Printing a cstring without using loops

cPtr → “Everett Community College”

```
void reversePrint( char * ptr);
```

```
void reversePrint( const char * ptr) {
    if ( *ptr == '\0' )
        return;
    else
        reversePrint( ptr + 1 );
    cout  <<  *ptr;
}
```

- What is the end condition?
  - null character '\0'
- What is the repeating pattern?
  - if not pointing to null, call the function for the next char
  - print one char

Programming Activity

- Reverse Printing a cstring

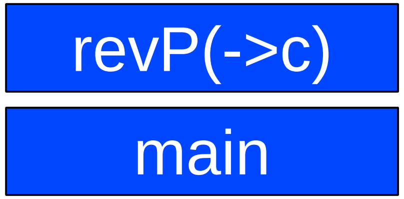
cPtr → “cat\0”

reversePrint(cPtr);

```
void reversePrint( const char * ptr) {
    if ( *ptr == '\0' )
        return;
    else
        reversePrint( ptr + 1 );
    cout  <<  *ptr;
}
```

- What is the end condition?
  - null character '\0'

- What is the repeating pattern?
  - if not pointing to null, call the function for the next char
  - print one char



Stack Memory

Programming Activity

- Reverse Printing a cstring

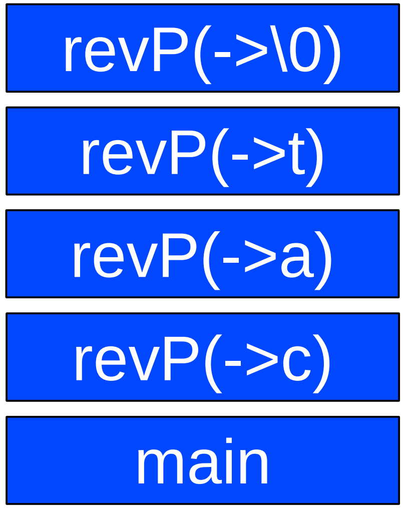
cPtr → “cat\0”

reversePrint(cPtr);

```
void reversePrint( const char * ptr) {
    if ( *ptr == '\0' )
        return;
    else
        reversePrint( ptr + 1 );
    cout  <<  *ptr;
}
```

- What is the end condition?
  - null character '\0'

- What is the repeating pattern?
  - if not pointing to null, call the function for the next char
  - print one char



revP(->\0)

revP(->t)

revP(->a)

revP(->c)

main

Stack Memory

Programming Activity

- Reverse Printing a cstring

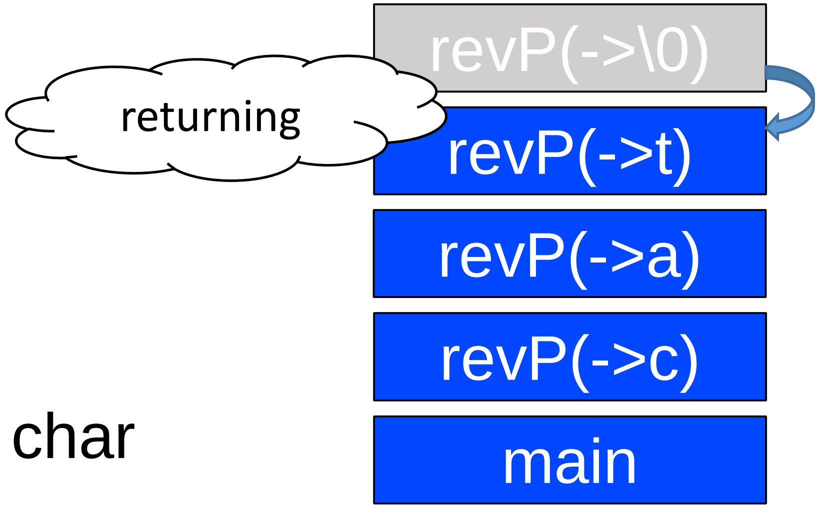
cPtr → “cat\0”

in revP(->t) call

```
void reversePrint( const char * ptr) {
    if ( *ptr == '\0' )
        return;
    else
        reversePrint( ptr + 1 );
    cout  <<  *ptr;
}
```

reversePrint(cPtr);

- What is the end condition?
  - null character '\0'
- What is the repeating pattern?
  - if not pointing to null, call the function for the next char
  - print one char



Stack Memory

Recursion Review

~ A small amount of code can be very powerful

~ It is generally slower than looping because it uses function calls.

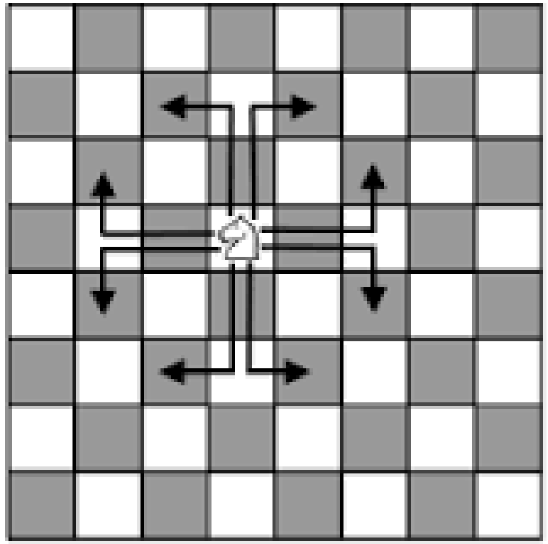
~ It can bomb by running out of memory due to using too many function calls, so be careful

Key: Find **Ending Condition**, and **Repeating Pattern**

Program 1: Knight’s tour

- In Chess there are many pieces with different kinds of movements.

- The knight moves in a bit of an unusual pattern of 2 squares horizontally and then 1 square vertically or 2 squares vertically and then 1 square horizontally. The 8 possible moves for a knight are shown here



- So the question came up ;

  “Is it possible to move a knight to each space on the board without ever traveling to a square twice in a tour?”

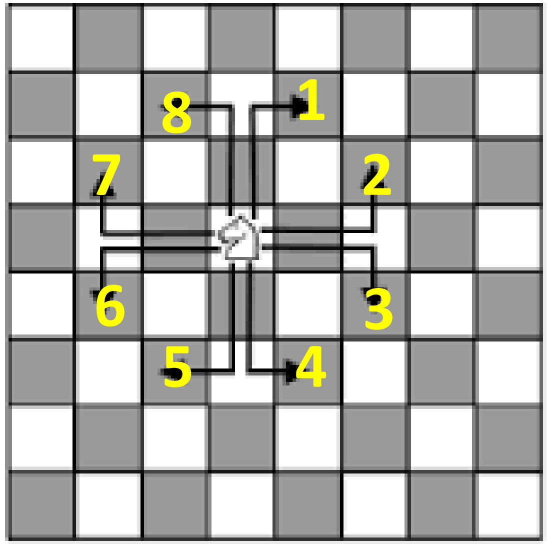
Program 1: Knight’s tour

- So for Program 1, we are going to write a recursive function that will try to discover a knight’s tour.
- As your knight moves to each square it should record the move number in the square. So its starting square would be marked with a 1, and then the next square it moves to would be marked with a 2 and so on.
- When your program is finished it should print out a representative of the chess board with the move number in each square.
- Here is an example of a knight’s tour on a 5 x 5 board

```
Yeehaw!!!
after 41 tries
and 16 bad moves
 1 20 17 12  3
16 11  2  7 18
21 24 19  4 13
10 15  6 23  8
25 22  9 14  5
```

Program 1: Knight’s tour

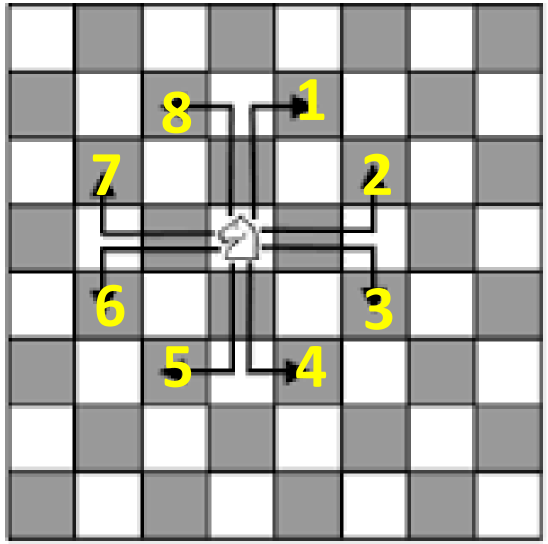
- A chess board is 8 x 8 and so a complete tour will need to travel to all 64 spaces.
- You can start your knight in any square on the board
- Depending on your algorithm some starting squares may take longer than other squares
- Using an algorithm where I had the knight try moves in the order shown to the right, and starting from a certain square, it tried over 62 billion moves and was still trying new paths after running all night…..starting from a different square it solve relatively quickly after 32 thousand moves



Program 1: Knight’s tour

- A chess board is 8 x 8 and so a complete tour will need to travel to all 64 spaces.

- To do this recursively, your recursive function should only be doing one move each call



- To start, you may first want to write a **printBoard** function that will display your board, so that you can view your board during testing and debugging

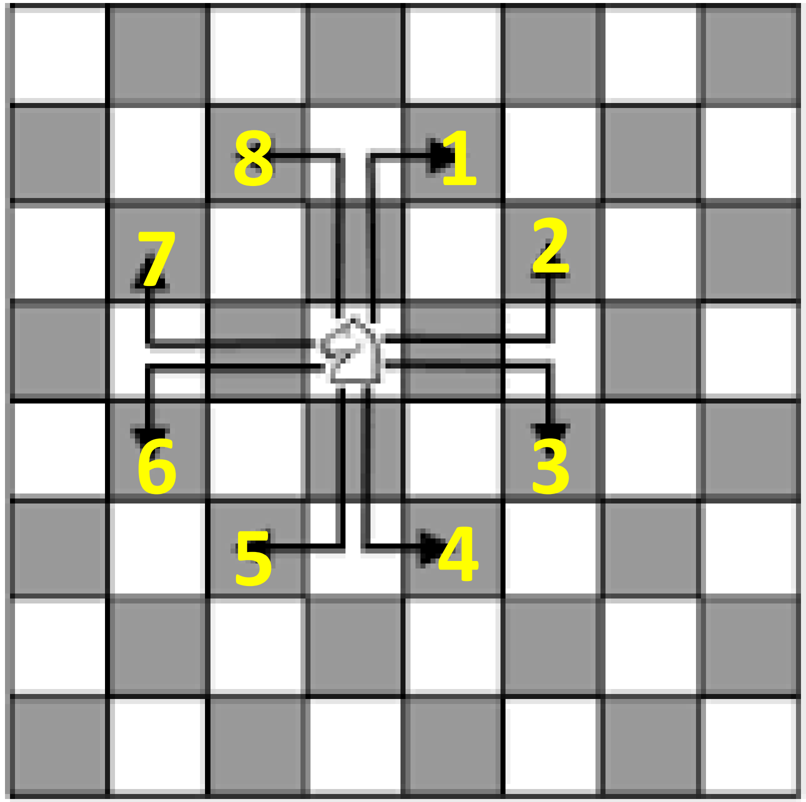
- You also may want to start with a 5 x 5 to watch your algorithm’s movements early on.

Lab Work: Knight’s tour

- Try a tour using a 5x5 board, starting in the upper left corner as your first move.

- Then continue moving using the move preferences shown below.

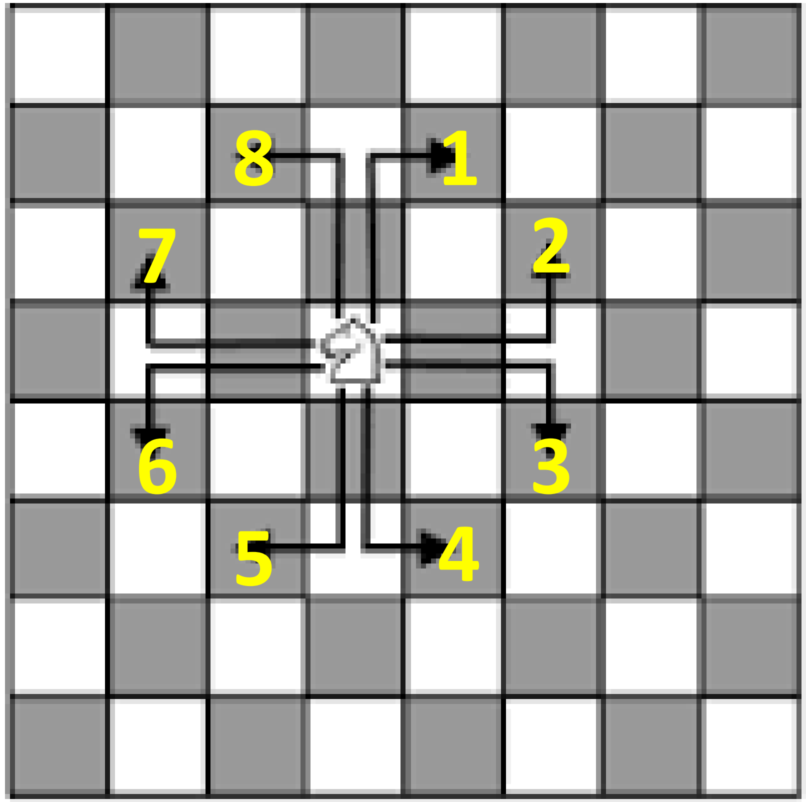
- Continue moving until you hit a dead-end.   What number did you get to?



Lab Work Part 2 : Knight's tour

- Think about the algorithm for the recursive solution to Knight's tour
- Here is a sample of the function prototype:
  bool moveKnight(int row, int col, int movNum);

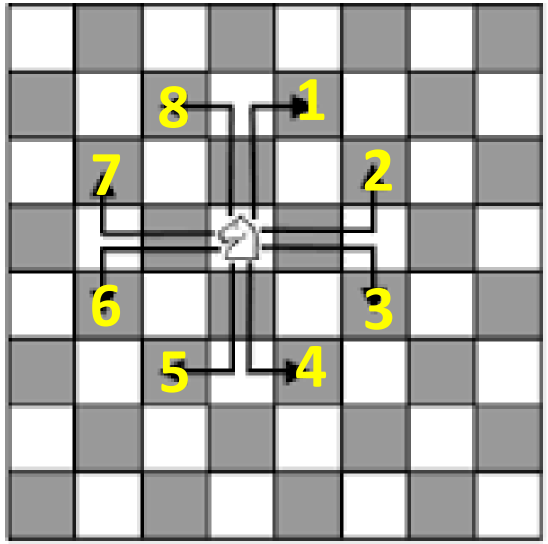
- As a group use pseudo-code to write out your ideas for a recursive solution



- End condition ?? Repeating Pattern ??
- The board and the total number of tries can be **global variables**.

Program 1: Knight's tour

- Before you get started, you may want to create a 5 x 5 board on paper and try to do a knight's tour by hand
- Then think of what the **ending conditions** and **repeating patterns** could be
- Then start to compose your algorithm
- Have fun with the challenge
- WARNING: This is a famous problem, and so there are lots of solutions out there on the internet, so please do not cheat yourself out of the joy, confidence, and learning experience of solving it yourself.



Recursion Review

~ A small amount of code can be very powerful

~ It is generally slower than looping because it uses function calls.

~ It can bomb by running out of memory due to using too many function calls, so be careful

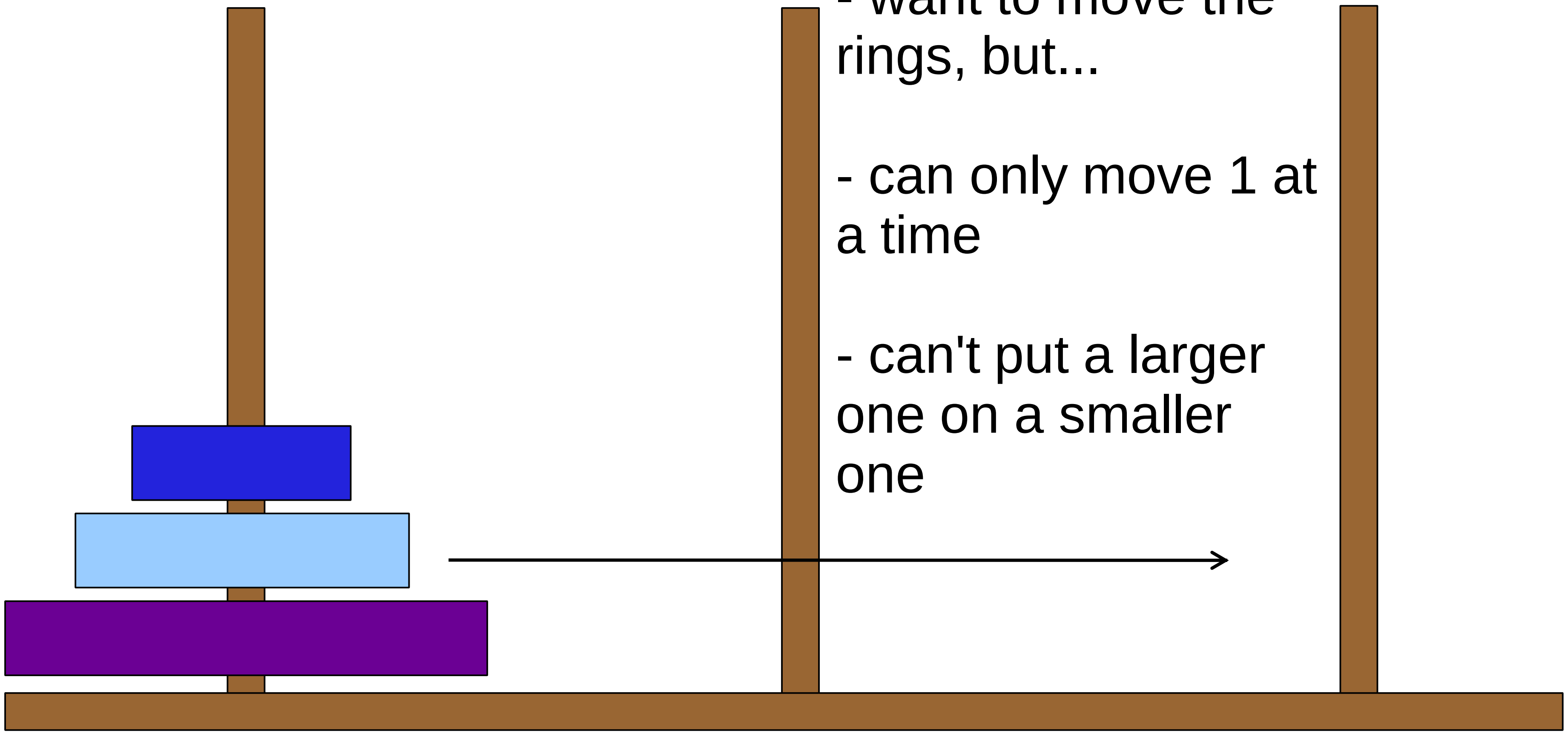
Key: Find **Ending Condition**, and **Repeating Pattern**

Extras: Another famous recursive problem

- Towers of Hanoi Problem

- This is extra (not on quiz or exam), and not required. I have used these slides in the past in lectures, and so included them for your enjoyment

Towers of Hanoi Problem

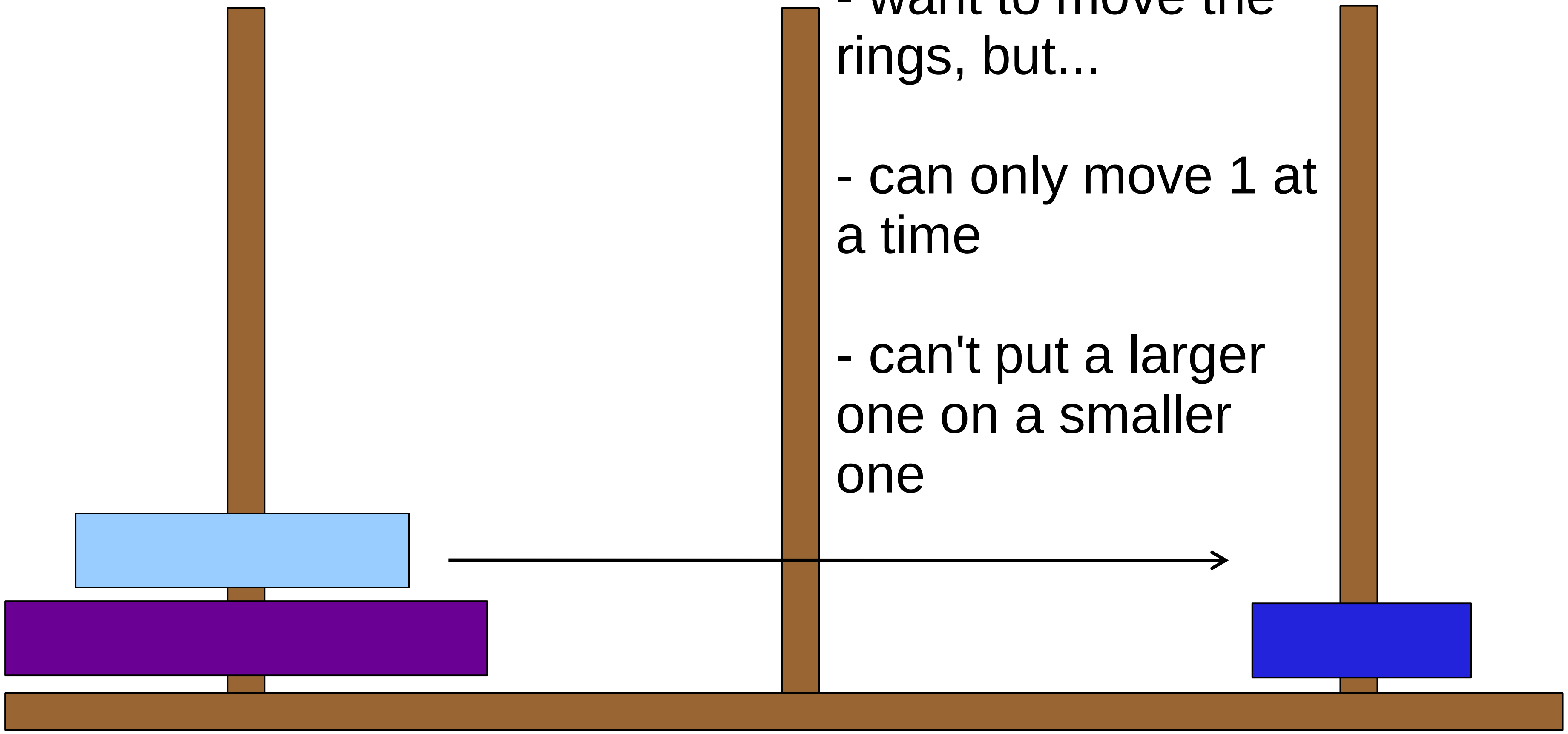


- want to move the rings, but...

- can only move 1 at a time

- can't put a larger one on a smaller one

Towers of Hanoi Problem

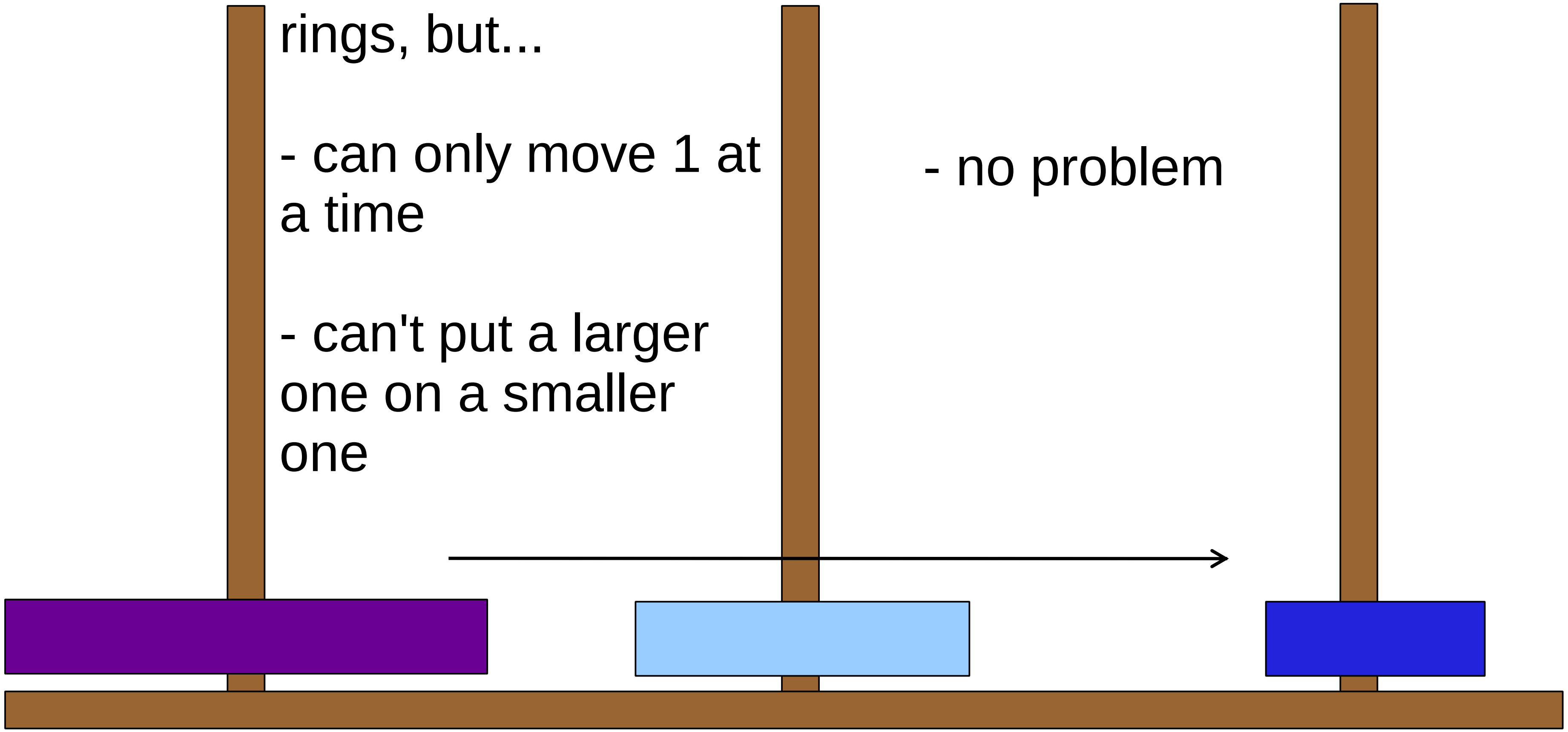


- want to move the rings, but...

- can only move 1 at a time

- can't put a larger one on a smaller one

Towers of Hanoi Problem



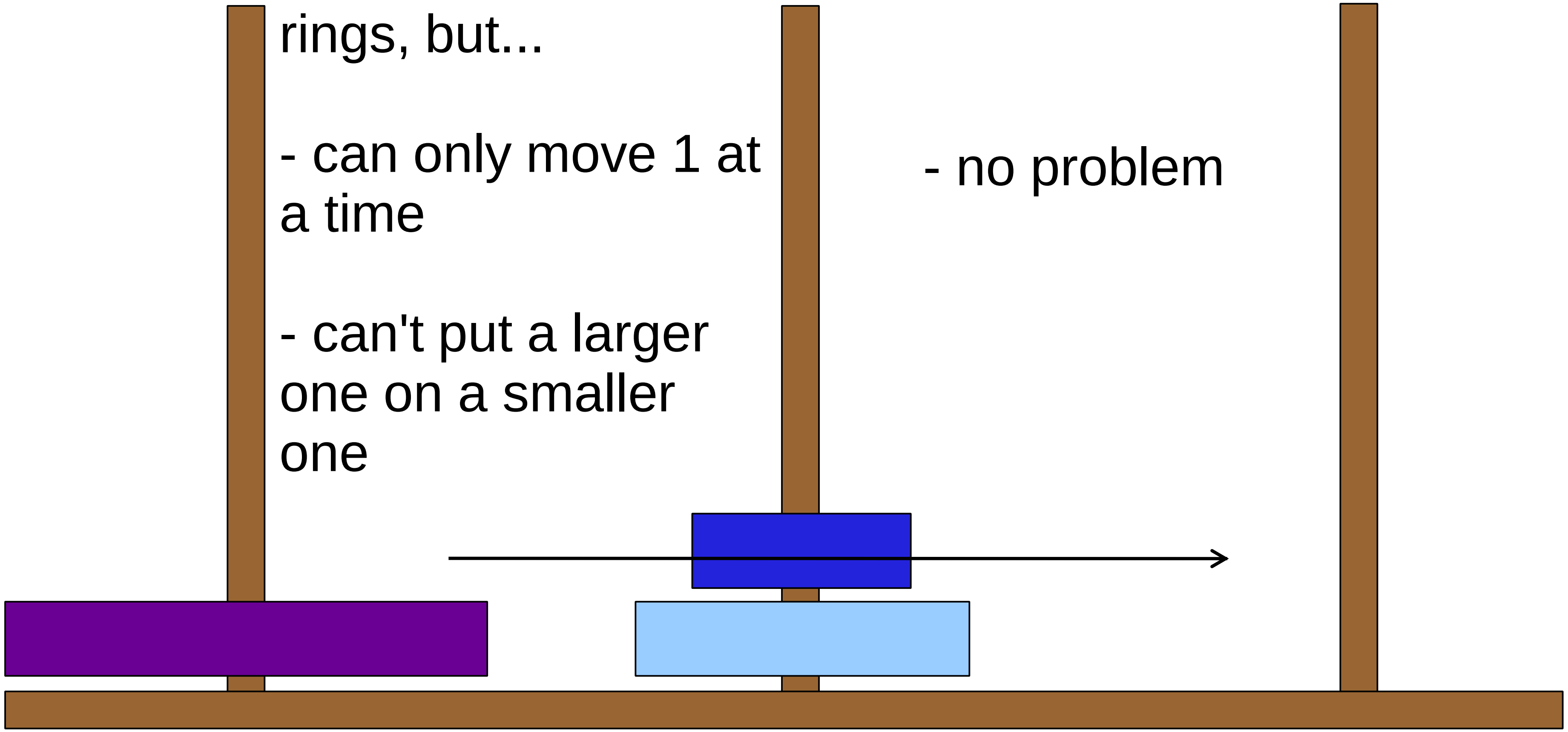
- want to move the rings, but...

- can only move 1 at a time

- can't put a larger one on a smaller one

- no problem

Towers of Hanoi Problem



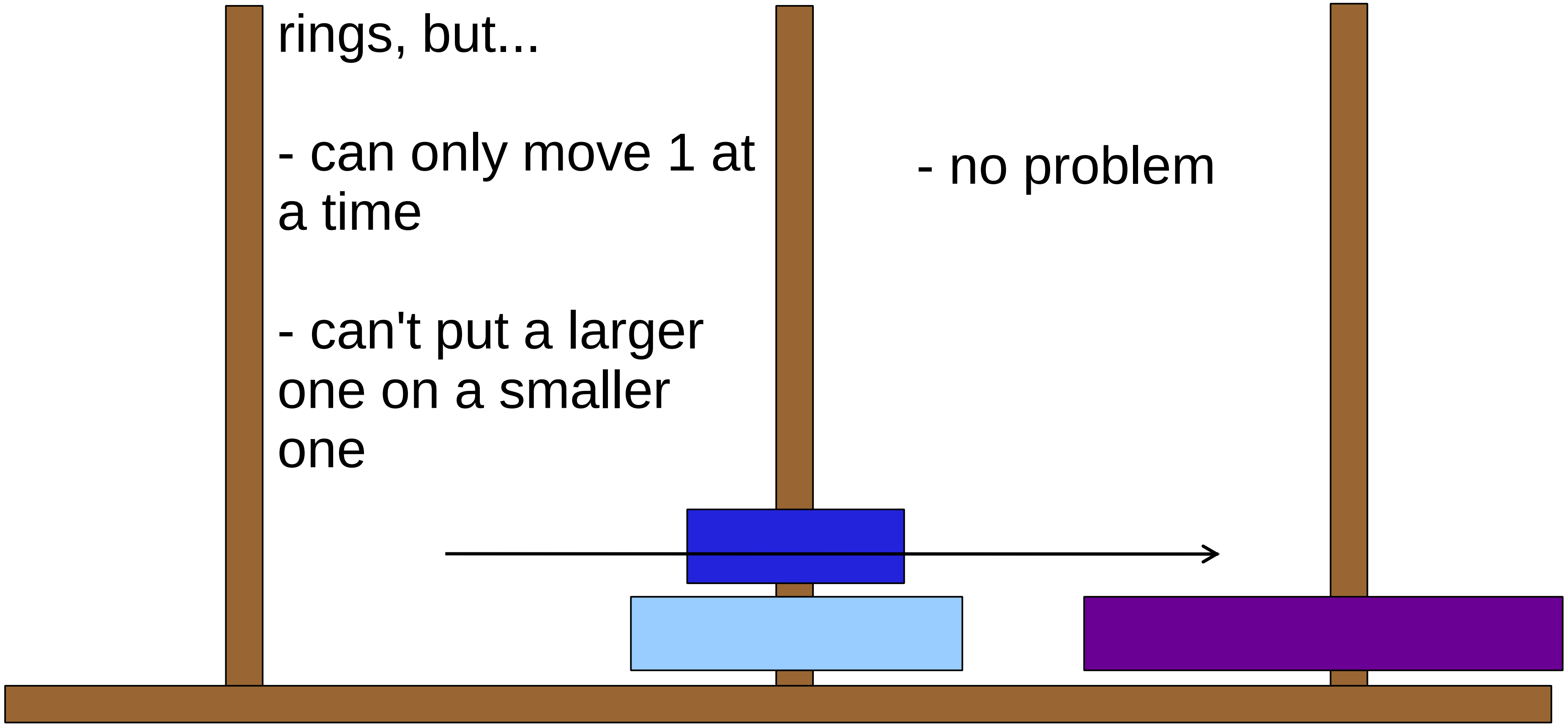
- want to move the rings, but...

- can only move 1 at a time

- can't put a larger one on a smaller one

- no problem

Towers of Hanoi Problem



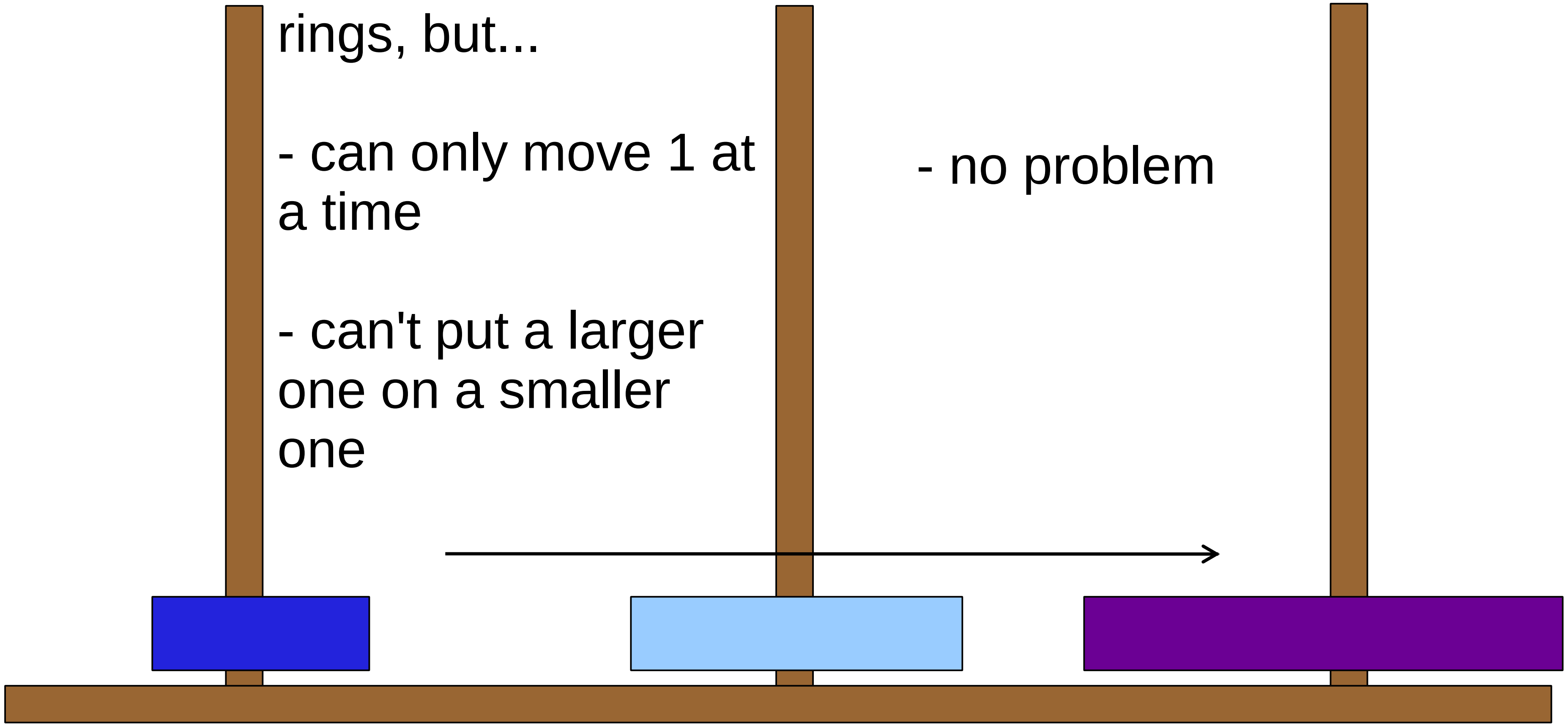
- want to move the rings, but...

- can only move 1 at a time

- can't put a larger one on a smaller one

- no problem

Towers of Hanoi Problem



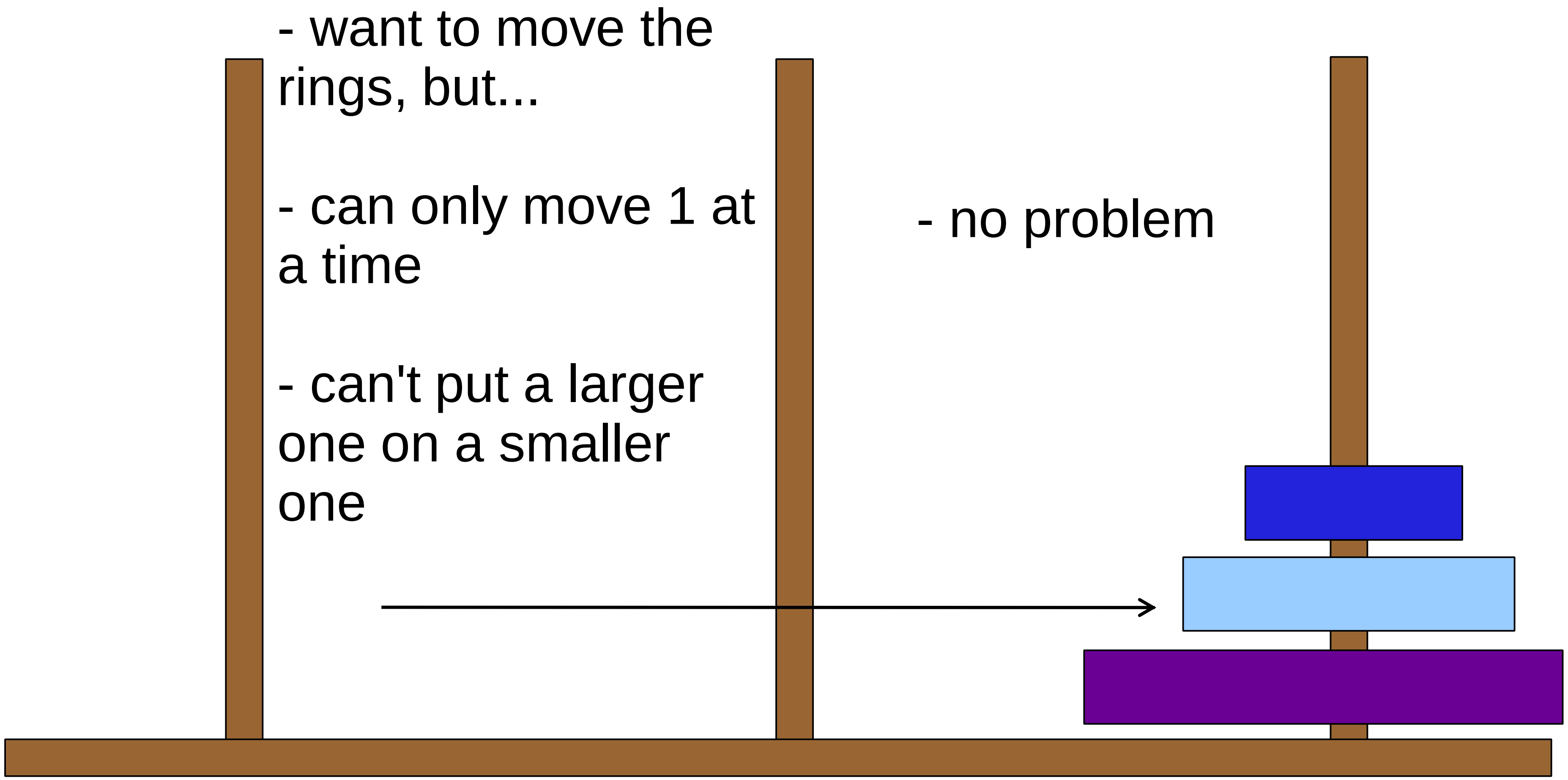
- want to move the rings, but...

- can only move 1 at a time

- can't put a larger one on a smaller one

- no problem

Towers of Hanoi Problem



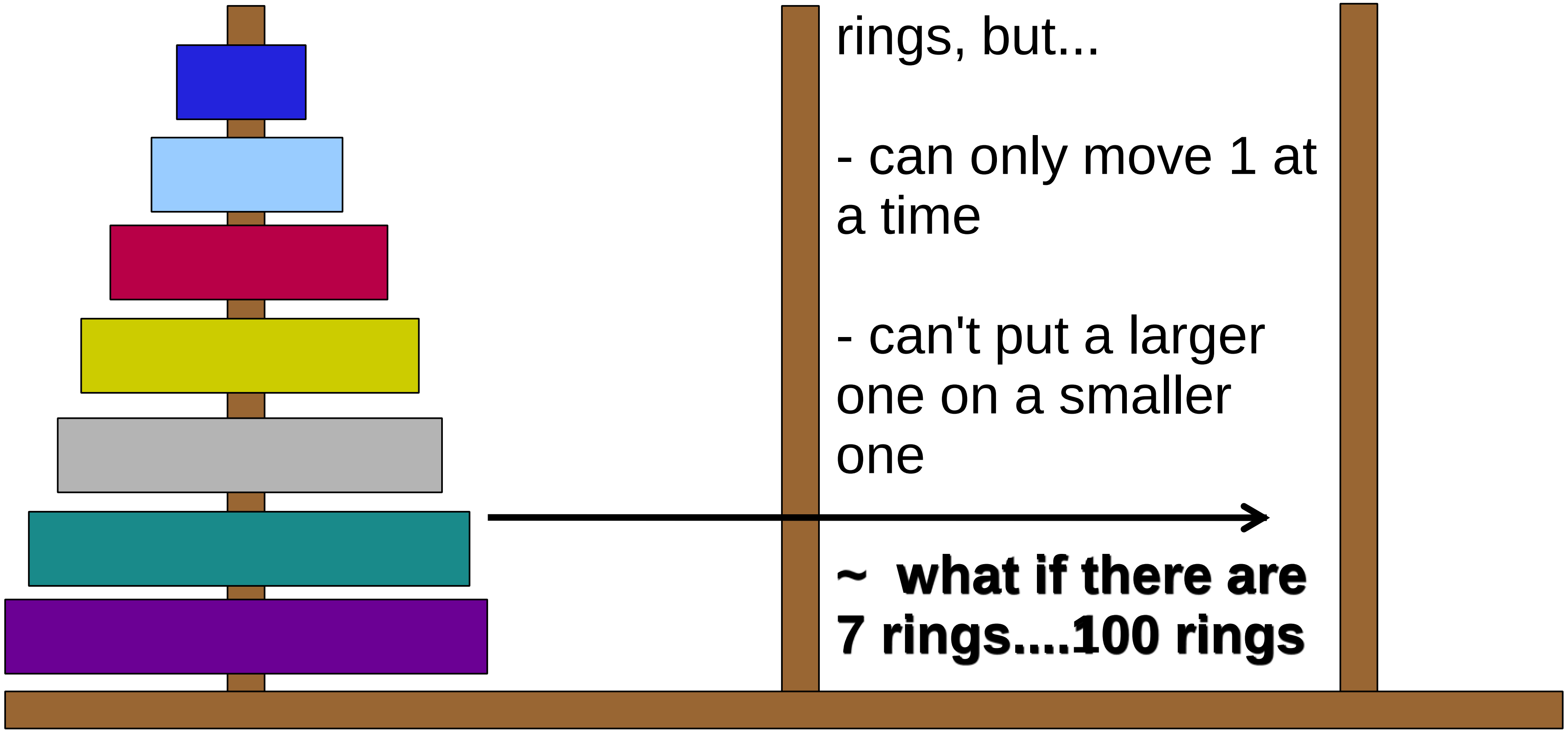
- want to move the rings, but...

- can only move 1 at a time

- can't put a larger one on a smaller one

- no problem

Towers of Hanoi Problem



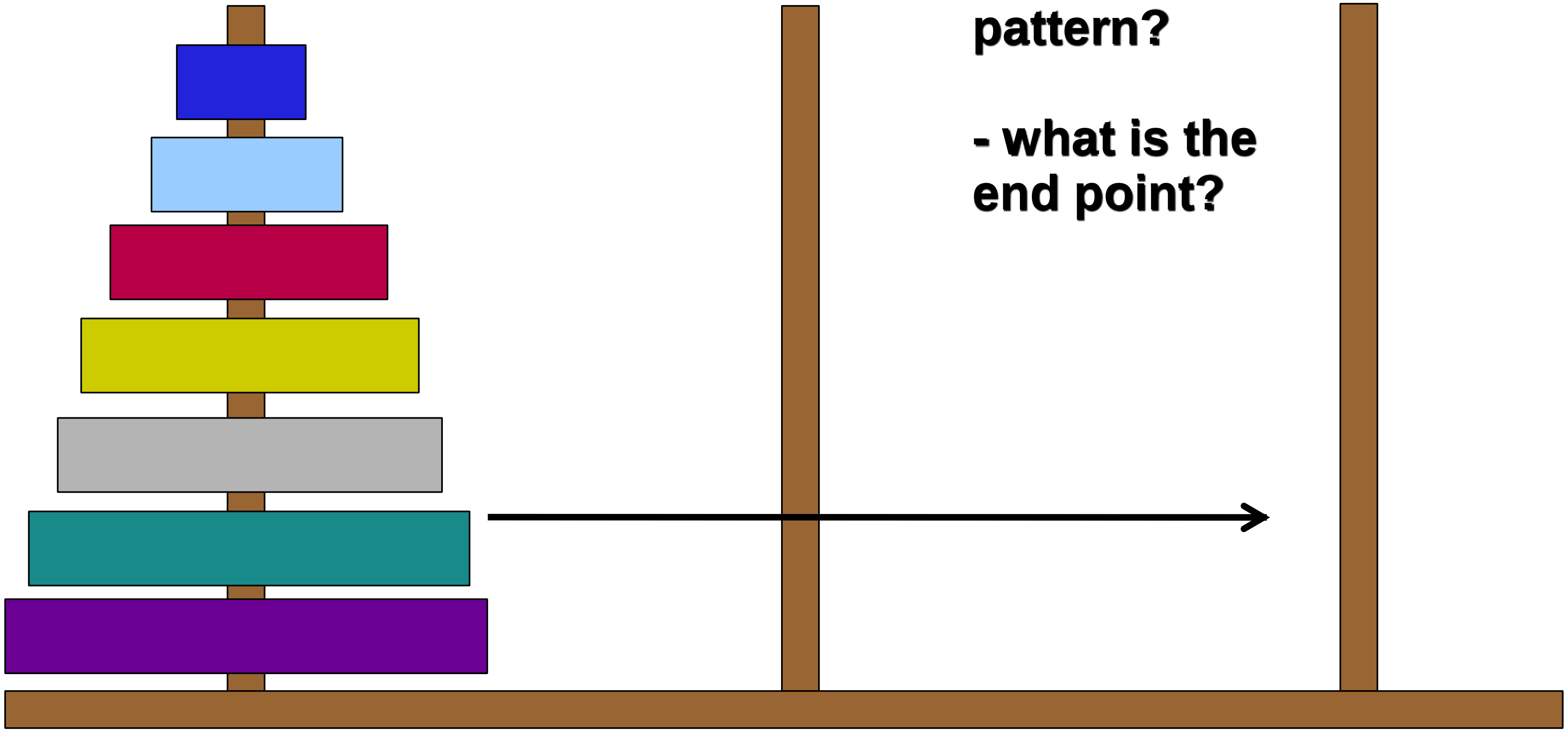
- want to move the rings, but...

- can only move 1 at a time

- can't put a larger one on a smaller one

~ what if there are 7 rings....100 rings

Towers of Hanoi Problem

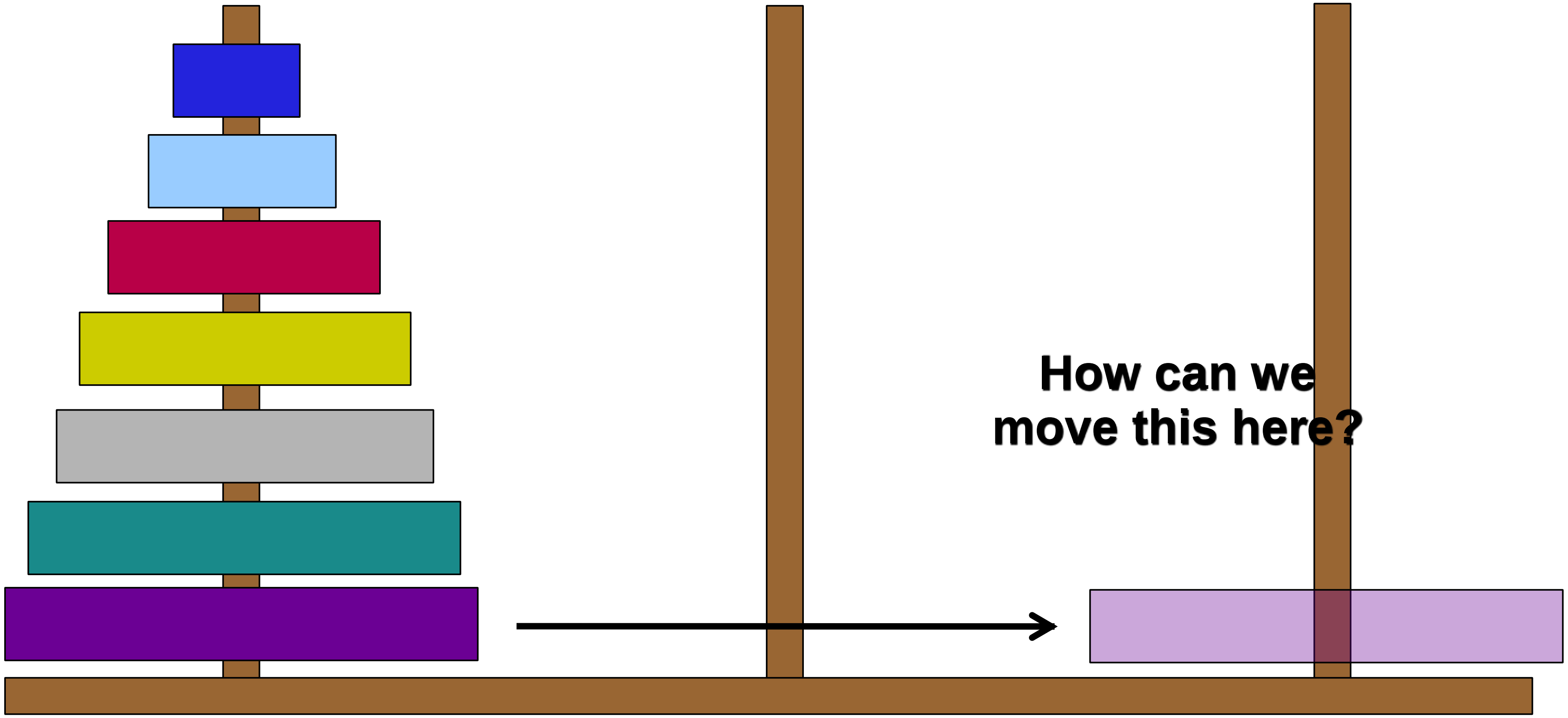


-what is the pattern?

- what is the end point?

-what is the pattern?

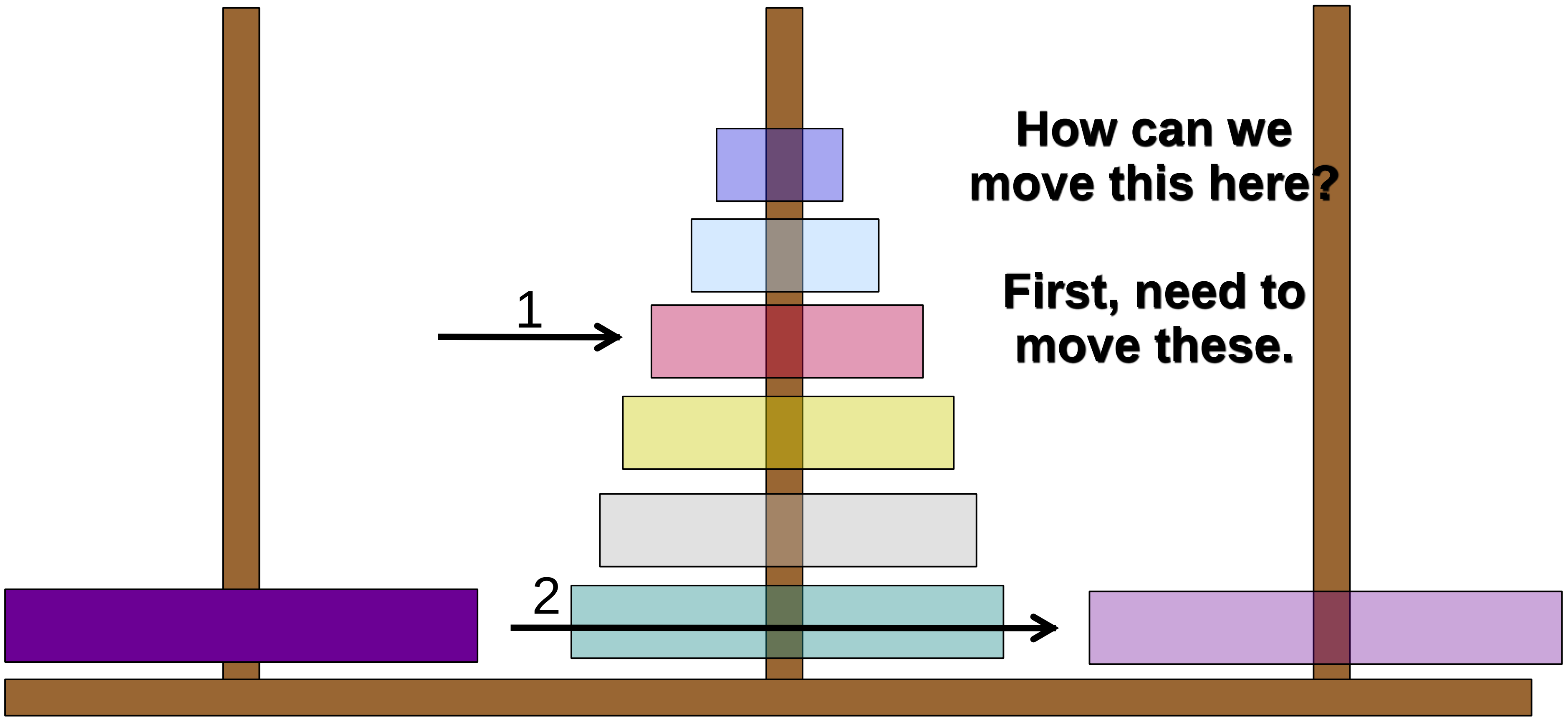
- what is the end point?



How can we move this here?

-what is the pattern?

- what is the end point?



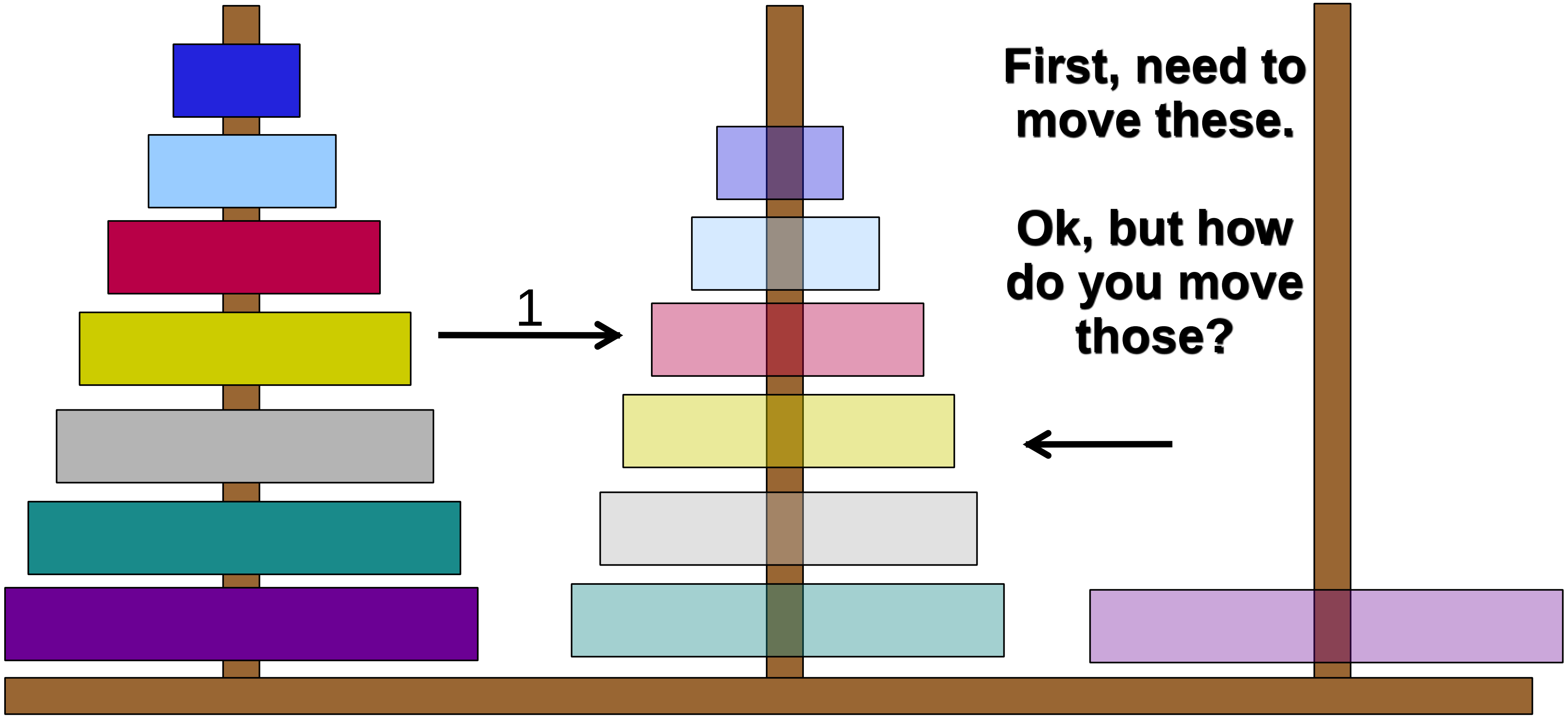
How can we move this here?

First, need to move these.

-what is the pattern?

- what is the end point?

how can we move this here?



First, need to move these.

Ok, but how do you move those?

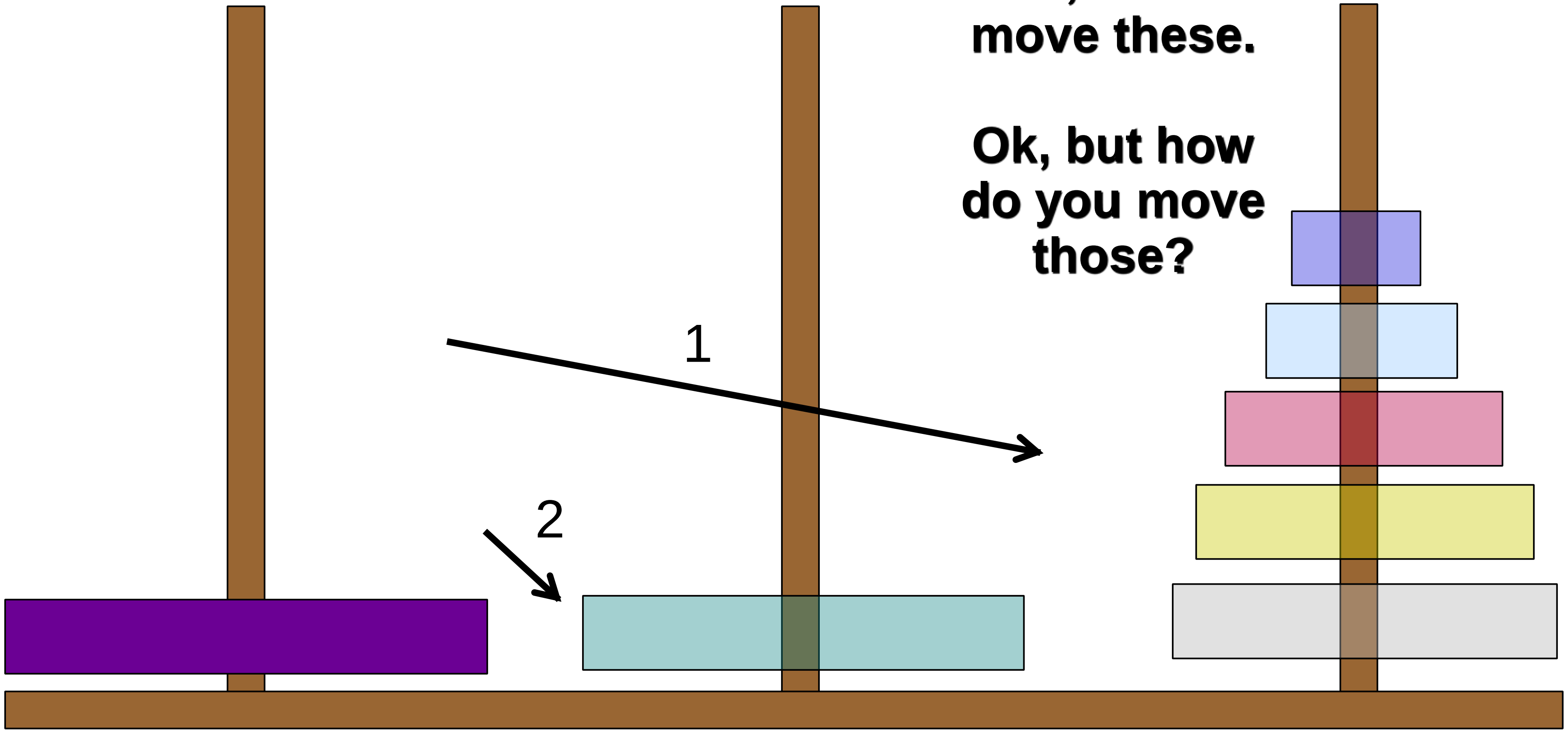
-what is the pattern?

- what is the end point?

how can we move this here?

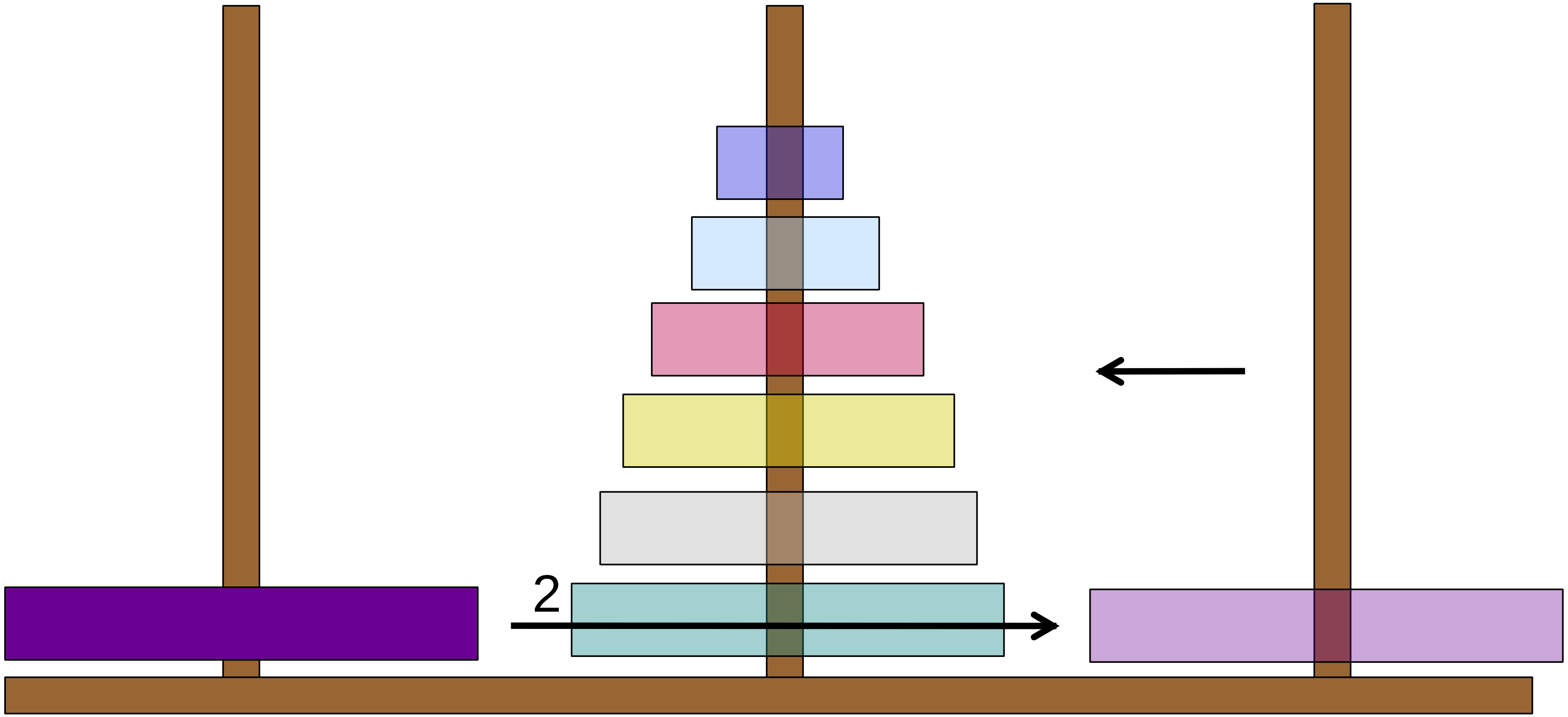
First, need to move these.

Ok, but how do you move those?



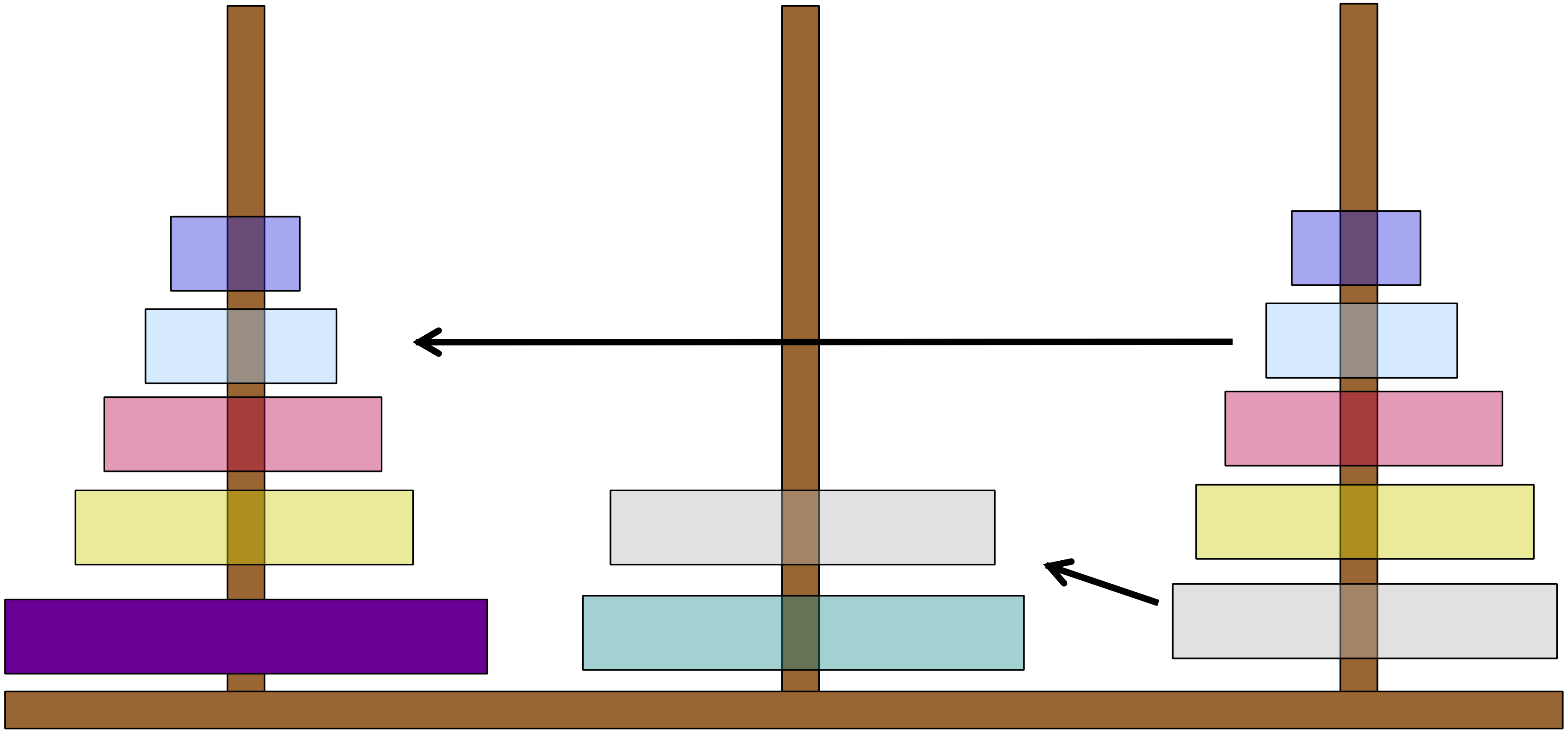
-what is the pattern?

- what is the end point?



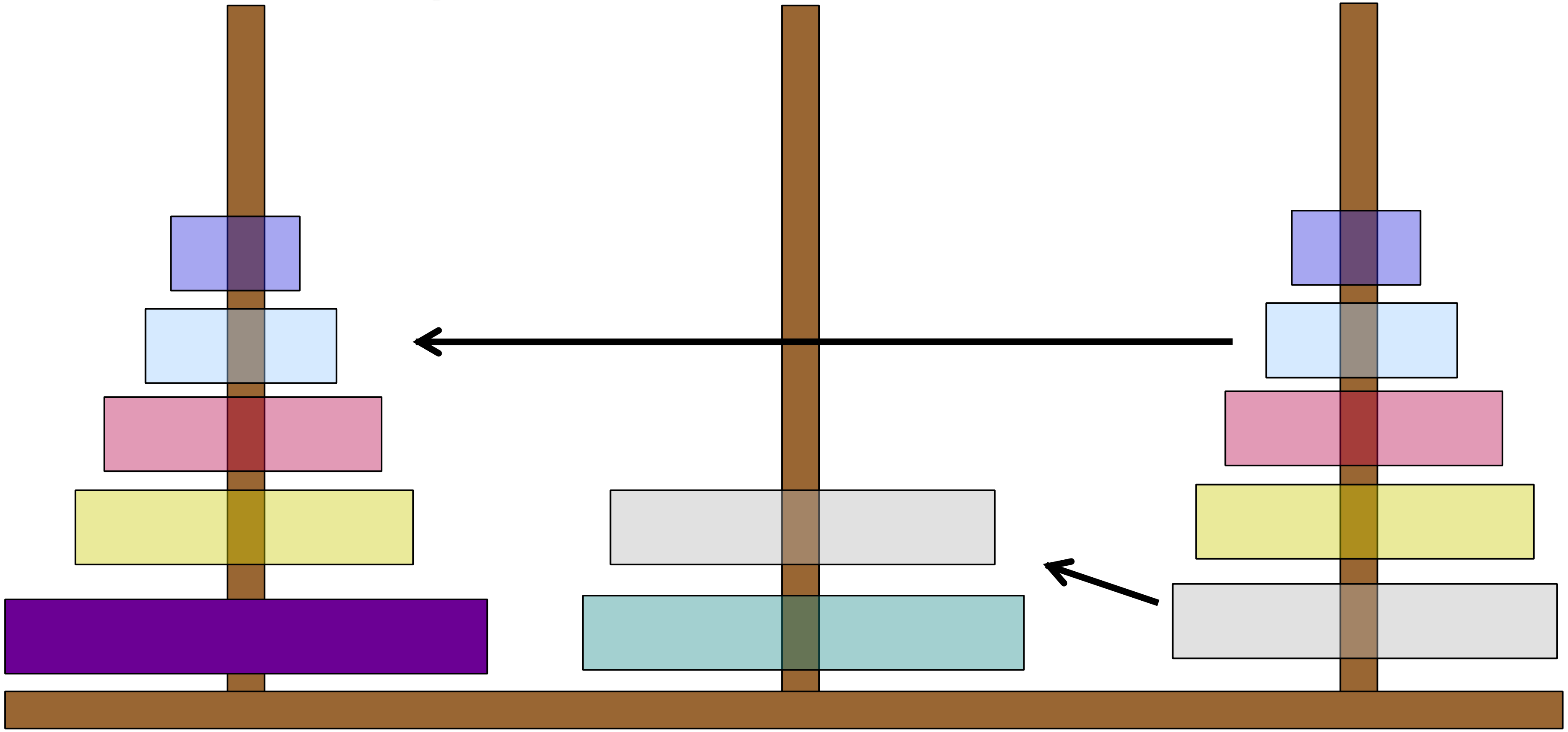
-what is the pattern?

- what is the end point?



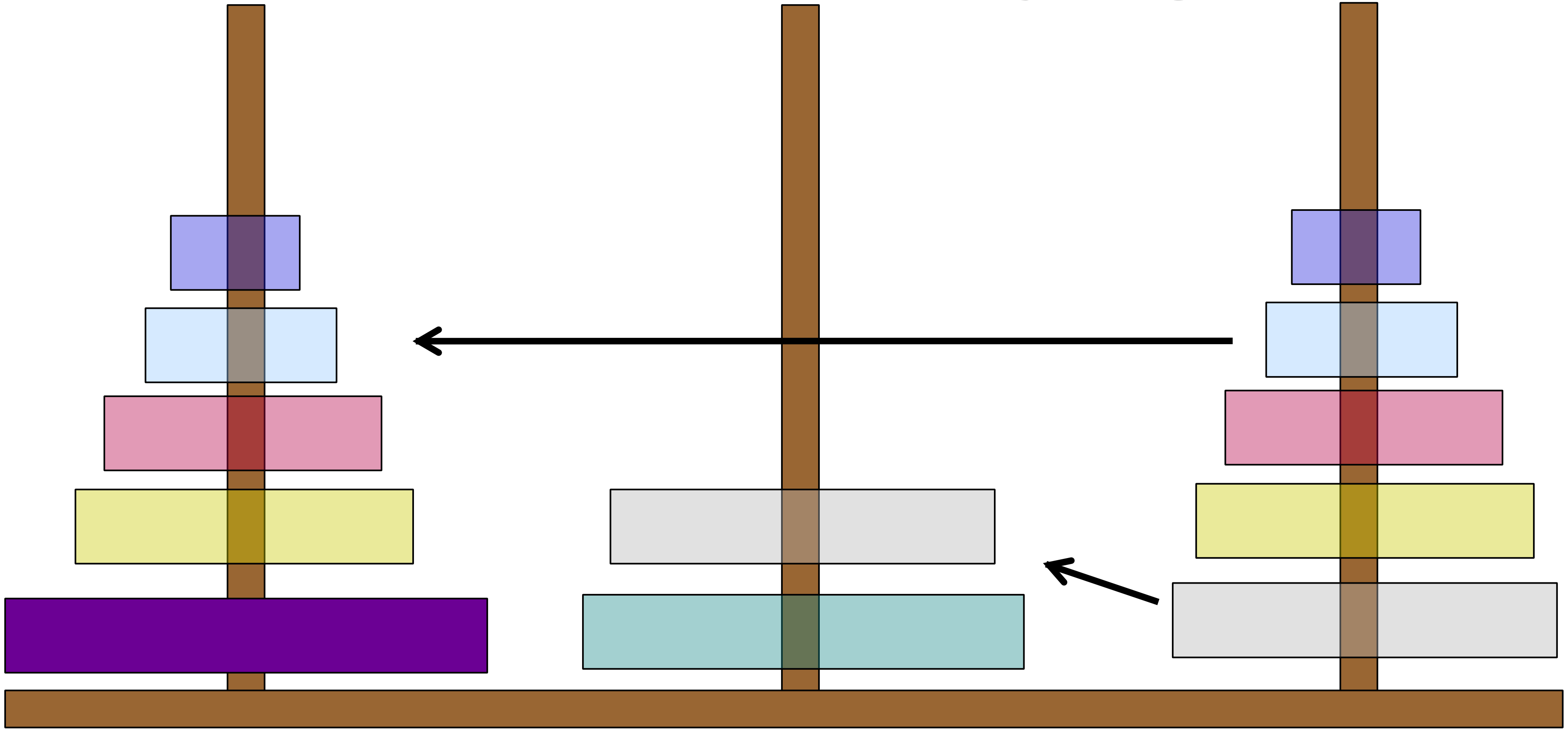
-what is the pattern? To move the nth ring, we must move n-1 smaller rings

- what is the end point?



-what is the pattern? To move the nth ring, we must move n-1 smaller rings

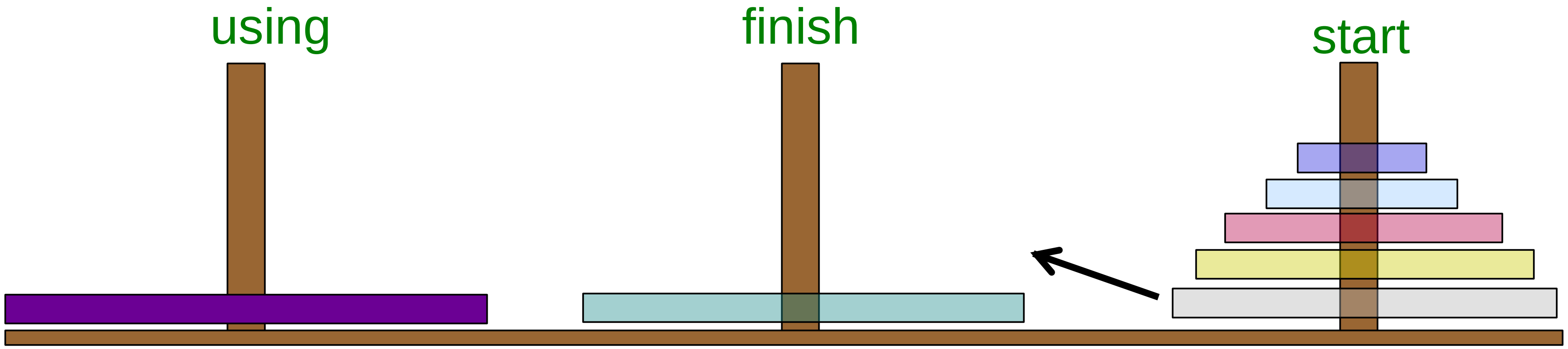
- what is the end point? When there is only 1 ring to move



-**what is the pattern? To move the nth ring, we must move n-1 smaller rings**

- what is the end point? When there is only 1 ring to move

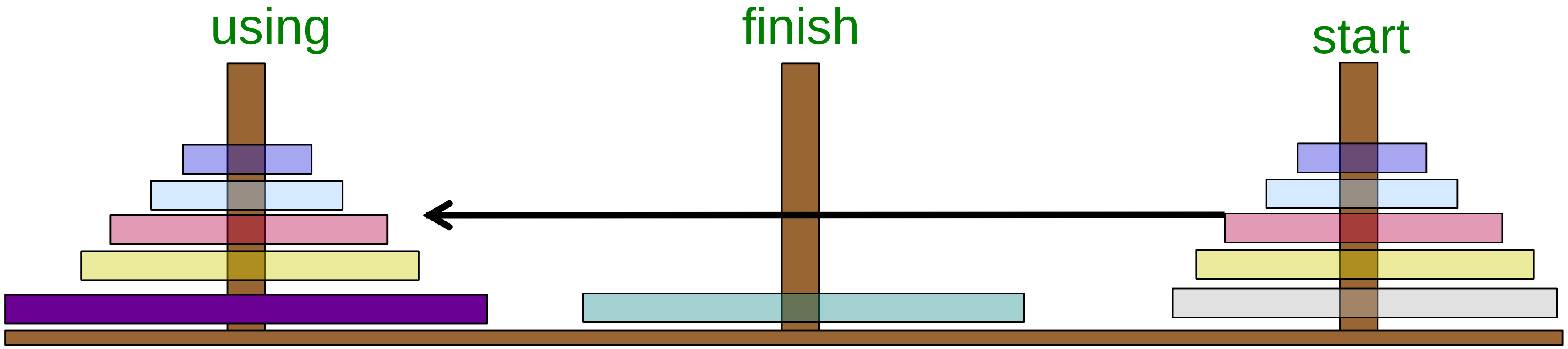
```
moveRings ( int numRs,  char start, char finish, char using)
    if numRs == 1
        then move it from start to finish
    else
        moveRings for all smaller rings (numRs-1) from start to using
        move our ring from start to finish
        moveRings for all smaller rings back from using to finish
```



-what is the pattern? To move the nth ring, we must move n-1 smaller rings

- what is the end point? When there is only 1 ring to move

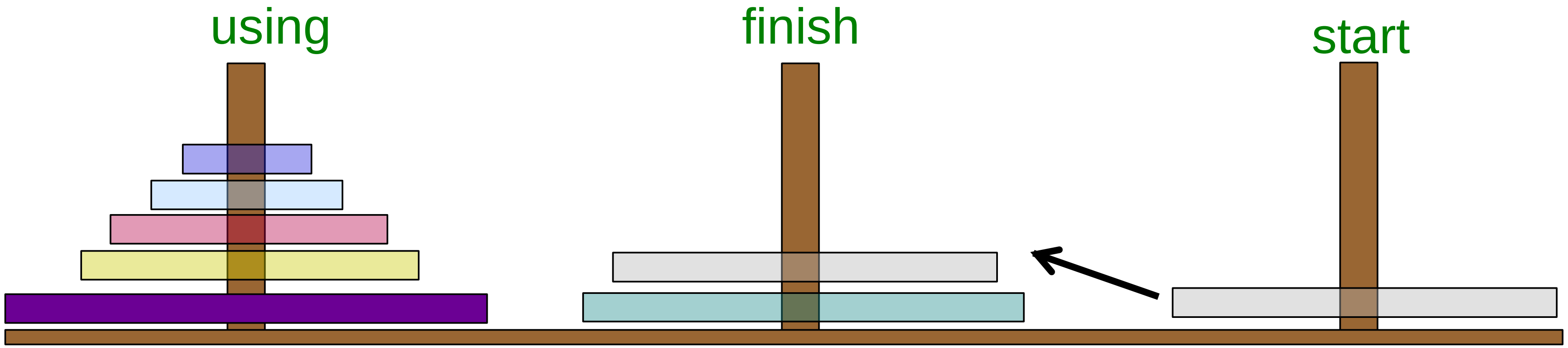
```
moveRings ( int numRs,  char start, char finish, char using)
    if numRs == 1
       then move it from start to finish
    else
       moveRings for all smaller rings (numRs-1) from start to using
       move our ring from start to finish
       moveRings for all smaller rings back from using to finish
```



-what is the pattern? To move the nth ring, we must move n-1 smaller rings

- what is the end point? When there is only 1 ring to move

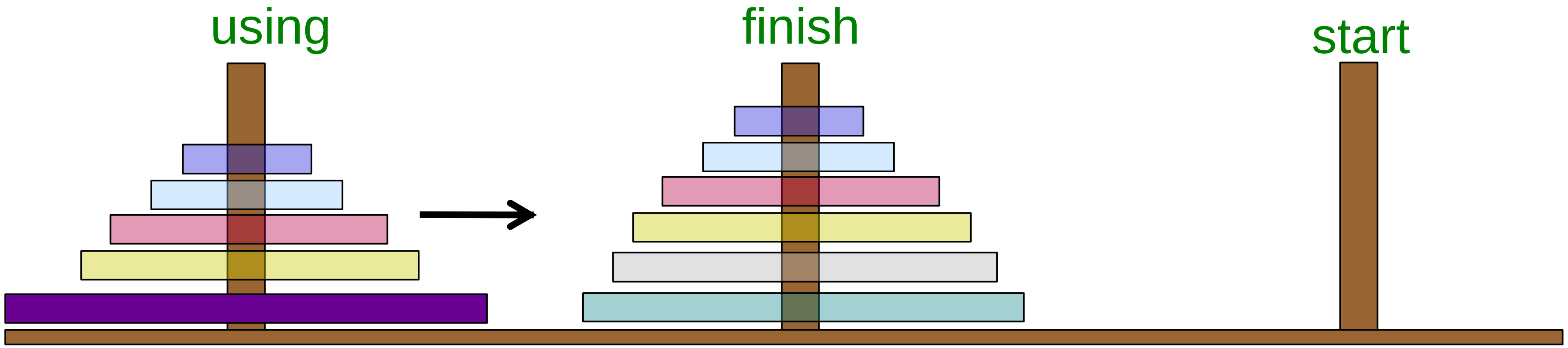
```
moveRings ( int numRs,  char start, char finish, char using)
    if numRs == 1
        then move it from start to finish
    else
        moveRings for all smaller rings (numRs-1) from start to using
        move our ring from start to finish
        moveRings for all smaller rings back from using to finish
```



-what is the pattern? To move the nth ring, we must move n-1 smaller rings

- what is the end point? When there is only 1 ring to move

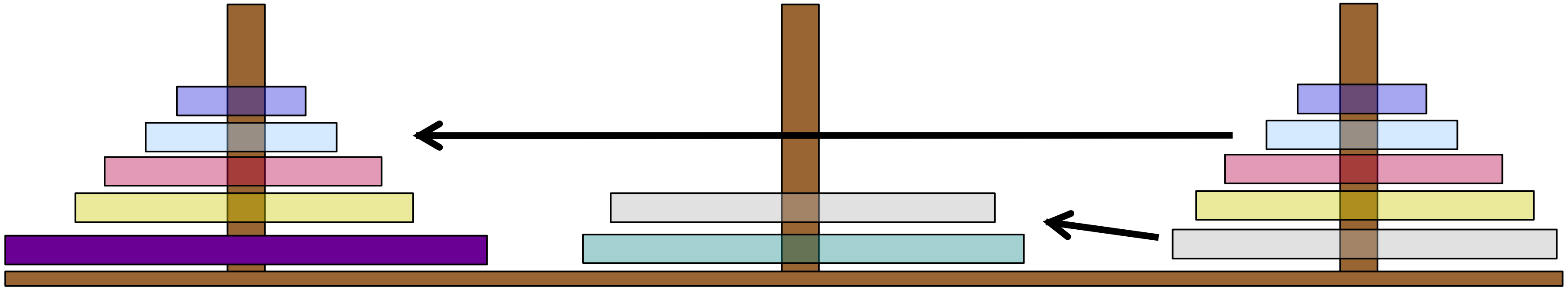
```
moveRings ( int numRs,  char start, char finish, char using)
      if numRs == 1
          then move it from start to finish
      else
          moveRings for all smaller rings (numRs-1) from start to using
          move our ring from start to finish
          moveRings for all smaller rings back from using to finish
```



-**what is the pattern? To move the n^th^ ring, we must move n-1 smaller rings**

- what is the end point? When there is only 1 ring to move

```
moveRings ( int numRs,  char start, char finish, char using)
    if numRs == 1
        then move it from start to finish
    else
        moveRings for all smaller rings (numRs-1) from start to using
        move our ring from start to finish
        moveRings for all smaller rings back from using to finish
```



Some great quotes from Bjarne Stroustrup
~ the creator of C++ ~

“The compiler has no common sense (it isn’t human) and it is very picky about details. Since it has no common sense, you wouldn’t like it to try to guess what you meant by something that “looked OK” but didn’t conform to the definition of C++”

Some great quotes from Bjarne Stroustrup ~ the creator of C++ ~

"The compiler has no common sense (it isn't human) and it is very picky about details. Since it has no common sense, you wouldn't like it to try to guess what you meant by something that "looked OK" but didn't conform to the definition of C++"

"When all is said and done, the compiler saves us from a lot of self-inflicted problems. It saves us from many more problems than it causes. So please remember: the compiler is your friend; possibly the compiler is the best friend you will have when you program"

Questions?