Recursion

By Dr. Cong WANG

CS Department

City Univ. of Hong Kong

Quick Facts

- Problem = sub-problem + simple problem
- Solve by Divide-and-Conquer
  - Recurrence relation (problem size < n)
  - Base case

Discussion –Factorial (n!)

- Input size: n
- Recurrence relation:

  $f(n) = n*f(n -1)$
- Base Case:

  Definition: 0! = 1

  $\therefore f(0) = 1$

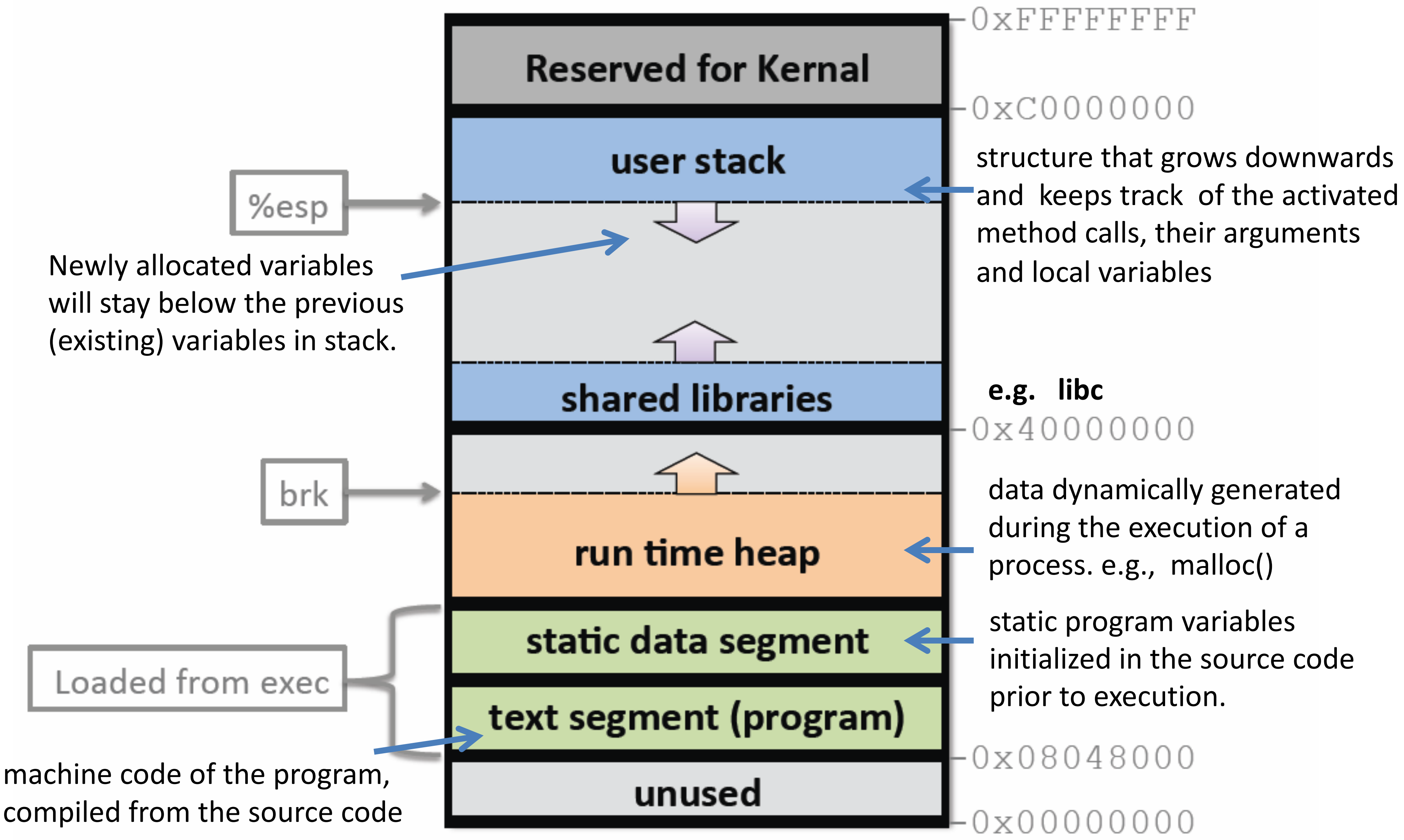
Base case is missing!

```
long factorial (long n){
    return (n * factorial(n-
1));
}
```

Correct version:

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

Linux (32-bit) process memory layout



0xFFFFFFFF

Reserved for Kernal

0xC0000000

user stack

structure that grows downwards and keeps track of the activated method calls, their arguments and local variables

%esp

Newly allocated variables will stay below the previous (existing) variables in stack.

shared libraries

e.g. libc

0x40000000

brk

run time heap

data dynamically generated during the execution of a process. e.g., malloc()

static data segment

static program variables initialized in the source code prior to execution.

Loaded from exec

text segment (program)

machine code of the program, compiled from the source code

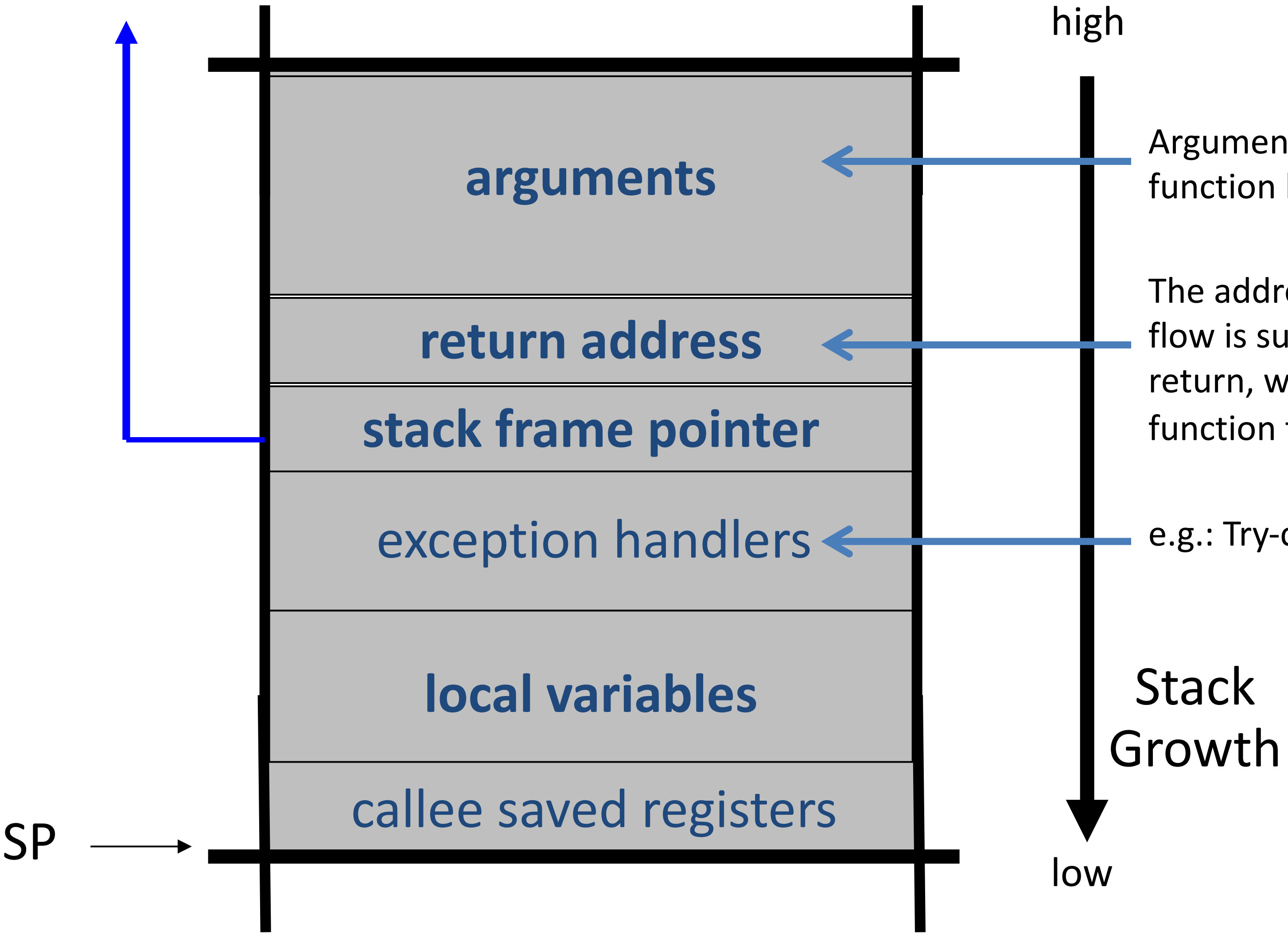
0x08048000

unused

0x00000000

Stack Frame

Every time a function call is executed, a new frame is pushed onto the stack.

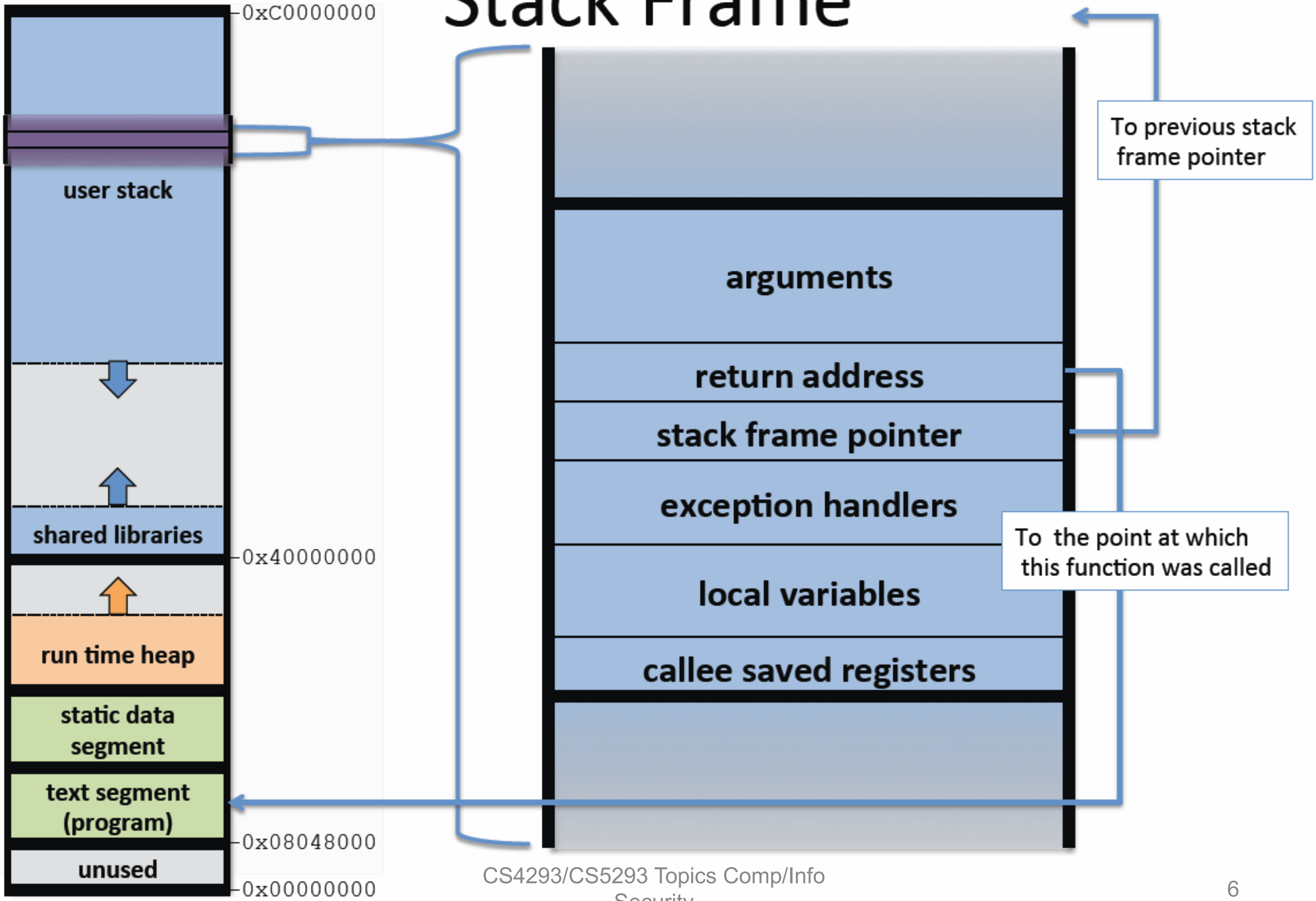


Arguments from the function being called

The address control flow is supposed to return, when the function terminates.

e.g.: Try-catch block

Stack Frame


0xC0000000
user stack
shared libraries
0x40000000
run time heap
static data segment
text segment (program)
0x08048000
unused
0x00000000
arguments
return address
stack frame pointer
exception handlers
local variables
callee saved registers
To previous stack frame pointer
To the point at which this function was called

Discussion –Factorial (n!)

- What if the function has to be called for many times?
  - e.g. List the factorial of 1 to 10?

n	n!
0	1
1	1
2	2
3	6
i	i*i–1 th row

  - What if we need get (50!) ?

Hands-on: nCr

- Given: n characters – a,b,c,d,…
- Want to choose r characters and display in order:
  - e.g. choose 3 characters from "abcde"

```
abc abd abe acd ace ade
bcd bce bde
cde
```

Solution: nCr

- We have r positions.
- Select one char from n chars for position i.
- Select one char from n-1 chars for position i+1.
- Once each position has char, then output.

```
combination(set, subset, r) {

    if (r == 0) {// the subset contains enough elements

        print elements in subset; return;
    }

    while (set is not empty) {
        move the first element from set to subset;
        combination(set, subset, r-1);
        remove the last element in subset;
    }
}
```

Dataflow

set	→	subset	→	r	output
abcde		-		3	
bcde	a	a		2	
cde	b	ab		1	
de	c	abc		0	abc
de		ab	c	1	
e	d	abd		0	abd
e		ab	d	1	
-	e	abe		0	abe
-		ab	e	1	
cde		a	b	2	
de	c	ac		1	
e	d	acd		0	acd
e		ac	d	1	
-	e	ace		0	ace

Dataflow cont.

set	→	subset	→	r	output
-		ac	e	1	
de		a	c	2	
e	d	ad		1	
-	e	ade		0	ade
-		ad	e	1	
e		a	d	2	
-	e	ae		1	
-		a	e	2	
bcde		-	a	3	
cde	b	b		2	
de	c	bc		1	
e	d	bcd		0	bcd
e		bc	d	1	

Dataflow cont.

set	→	subset	→	r	output
-	e	bce		0	bce
-		bc	e	1	
de		b	c	2	
e	d	bd		1	
-	e	bde		0	bde
-		bd	e	1	
e		b	d	2	
-	e	be		1	
-		b	e	2	
cde		-	b	3	
de	c	c		2	
e	d	cd		1	
-	e	cde		0	cde

Dataflow cont.

set	→	subset	→	r	output
-		cd	e	1	
e		c	d	2	
-	e	ce		1	
-		c	e	2	
de		-	c	3	
e	d	d		2	
-	e	de		1	
-		d	e	2	
e		-	d	3	
-	e	e		2	
-		-	e	3	

Explanation

```
char set[5]={'a','b','c','d','e'};
```

We use the index number to represent each character.

a	b	c	d	e
0	1	2	3	4

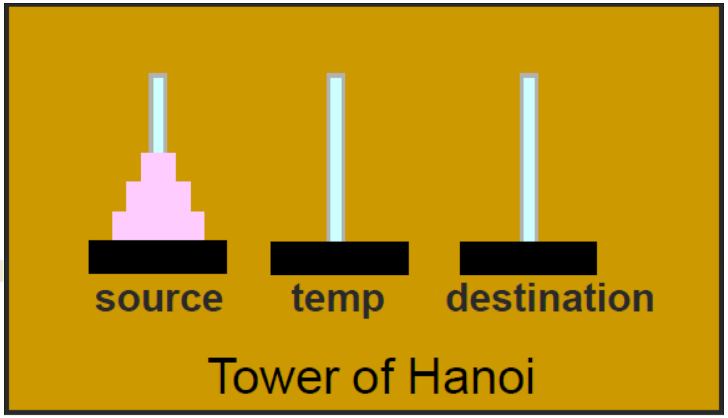
Output:

```
abc abd abe acd ace ade
bcd bce bde
cde
```

```
0
  --1
      --2
      --3
      --4
  --2
      --3
      --4
  --3
      --4
```

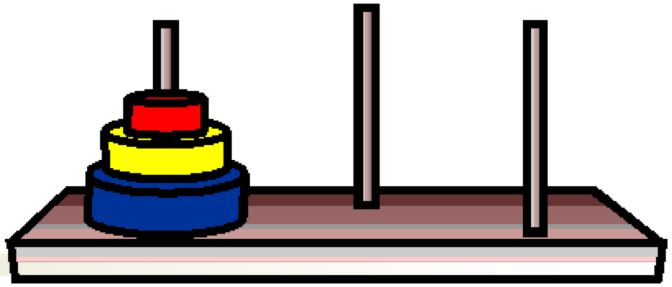
```
1
  --2
      --3
      --4
  --3
      --4
2
  --3
      --4
```

Hands-on



- Objective
  - Move n disks from the source pole to the destination pole (Problem size: n)
- Steps:
  1. Move the smaller n-1 disks from source to temp (Problem size:n-1)
  2. Move the largest disk from source to destination (single step)
  3. Move the smaller n-1 disks from temp to destination (Problem size:n-1)

Implementation



- 3 poles: A (source), B, C (destination)
- Input: number of disks (1..n)
- Output: the sequence of movement for all disks
  - move disk 1 from pole A to pole C
  - move disk 2 from pole A to pole B
  - move disk 1 from pole C to pole B
  - move disk 3 from pole A to pole C
  - move disk 1 from pole B to pole A
  - move disk 2 from pole B to pole C
  - move disk 1 from pole A to pole C

Moving 3 disks from A to C

Pros & Cons

- Pros
  - Once recurrence relation and base case are identified, implementation is straight forward.
  - Easy to handle.
- Cons
  - Debugging can be an issue.
  - Need to consider performance (many duplicated cases, large input).