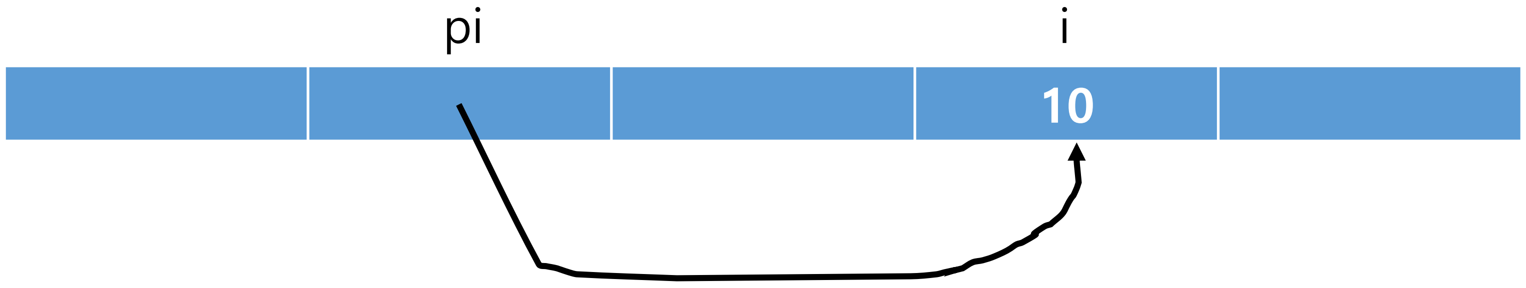
Chap 1. Basic Concepts (Part 1)

C Pointer

- & : address operator
- * : dereferencing (or indirection) operator
- e.g.,

```
int i, *pi;

pi = &i;
i = 10;
*pi = 10;
```

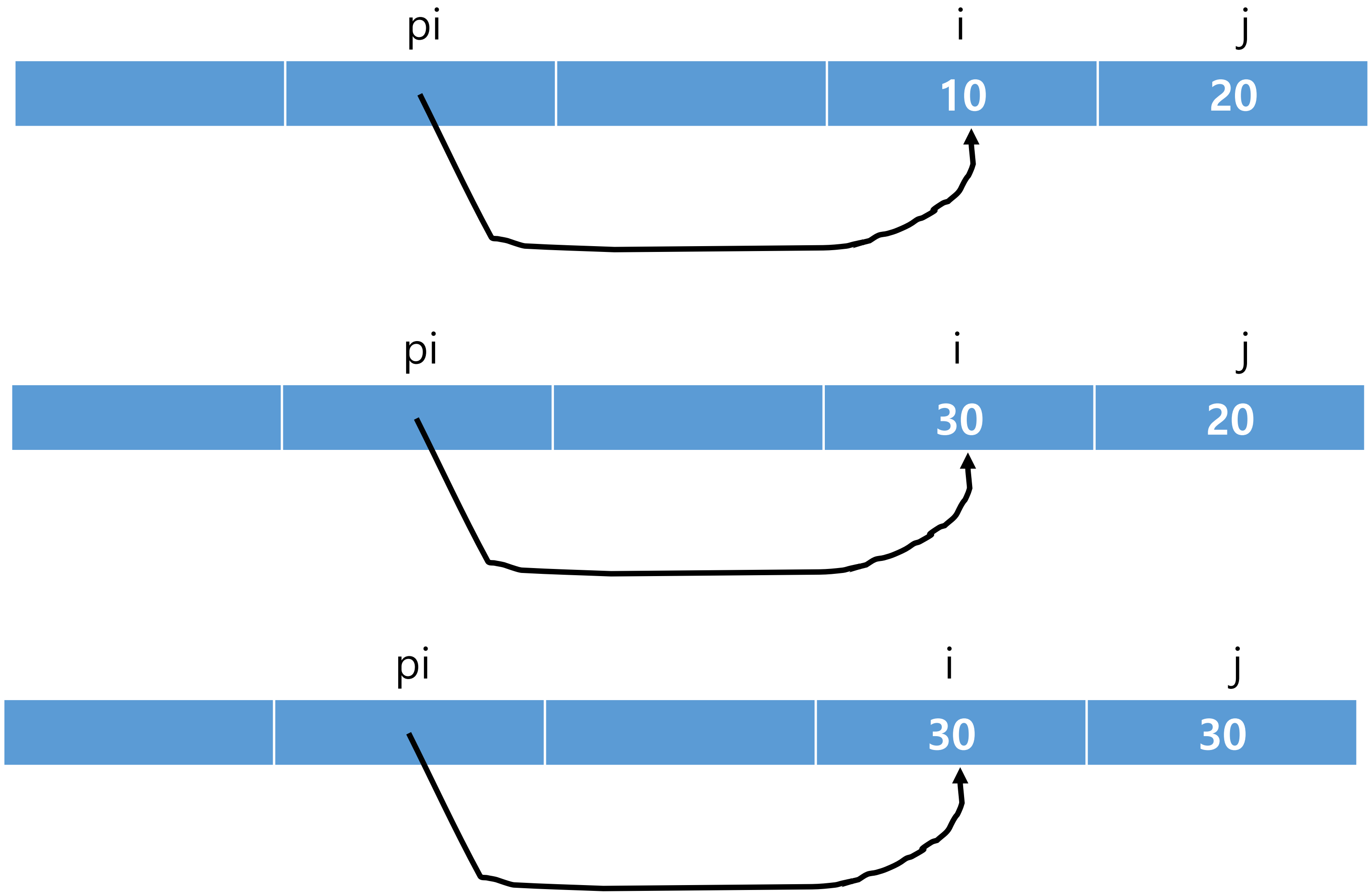


C Pointer

- e.g.,

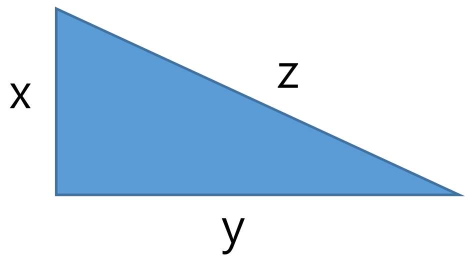
```
int i, *pi, j;

pi = &i;
i = 10;
j = 20;
*pi = 30;
j = *pi;
```



C Pointer

- Parameter passing in function call
- call by value vs. call by reference
- e.g.,




```
int x, y, z;
x = 3;
y = 4;
pythagoras(x, y, z);
printf("z = %d₩n", z);
```

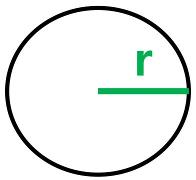
```
void pythagoras(int x, int y, int z) {
z = (int) sqrt((double) (x*x) + (double) (y*y));
}
```

```
int x, y, z;
x = 3;
y = 4;
pythagoras(x, y, &z);
print("z = %d₩n", z);
```

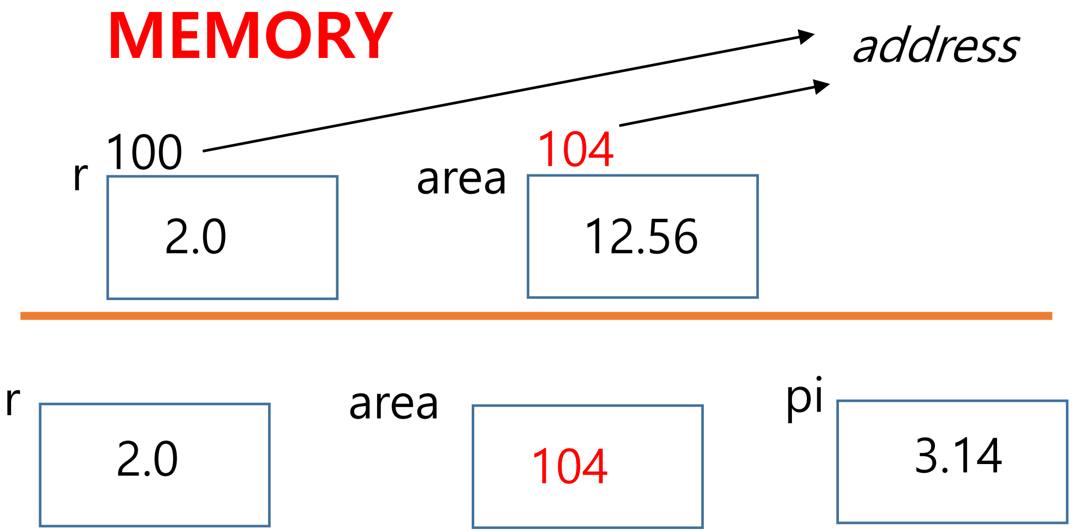
```
void pythagoras(int x, int y, int *z) {
*z = (int) sqrt((double) (x*x) + (double) (y*y));
}
```

C Pointer

- Example:
  - 원의 면적 구하기
  - πr^2




```
float r, area; //r: radius
r = 2.0;
area_of_circle(r, &area);
printf("₩nradius = %f, area = %f", r, area);
```




```
void area_of_circle(float r, float *area) {
        float pi = 3.14;
        *area = pi * (r*r);
}
```

Algorithm

- A finite set of instructions that accomplishes a certain task
- Criteria
  - Input
  - Output
  - Definiteness: clear, unambiguous
  - Finiteness: termination
  - Effectiveness
- Specification
  - 자연어
  - Diagram (e.g., flow chart)
  - Pseudo code
  - Programming language
  - etc.

Pseudo code

- 장점:
  - programming language의 문법(syntax) 준수에서 자유로움
  - 가독성 높은 알고리즘 명세 용이
- 단점: 알고리즘의 criteria 충족하지 못할 수 있는 점 주의 필요
- Example: 배열 A[0..n-1]에 저장된 n개의 수 중 양수의 개수와 그 합 구하기

- C 언어

```
sum = 0;
count = 0;
for(i = 0; i < n; i++) {
     if(A[i] > 0) {
          sum += A[i];
          count++;
     }
}
```

- Pseudo codes

```
sum ← 0
count ← 0
for i ← 0 to n-1 do {
        if(A[i] > 0) then {
                sum ← sum + A[i]
                count ← count + 1
        }
}
```

```
sum := 0
count := 0
for i := 0 to n-1 do
        if(A[i] > 0) then
            begin
                sum := sum + A[i]
                count := count + 1
            end
        endif
endfor
```

```
sum = 0
count = 0
for i = 0 to n-1 do
        if(A[i] > 0)
            sum = sum + A[i]
            count = count + 1
```

들여쓰기(indentation):
- Be careful !!

- 각 명령문: basic & feasible해야 함

Pseudo code 구문

- Assignment: ←, =, := (C 언어: =)
- 제어문
  - if-then-else
  - for i ← 1 to n [by 1] {…} //for(i=1;i<=n;i++) {…}
  - for i ← n to 1 by -1 {…} //for(i=n;i>=1;i--) {…}
  - while(condition) {…}
  - do {…} while(condition)
  - loop {…} endloop //termination condition in the loop body
  - case //C언어 switch문
  - Block of statements: {…}, begin…end, if…endif, for…endfor, while…endwhile, …
  - 기타

Pseudo code 구문

- 두 수 a와 b를 비교:
  - 결과: 3가지 경우 a>b, a=b, a<b
- C 언어

```
if(a<b) {…
}
else if(a==b) {…
}
else {//a>b
…
}
```

- Pseudo codes

```
compare(a, b) {
   case '<' : …
   case '=' : …
   case '>' : …
}
```

```
compare(a, b) {
   case '<' : …
            break
   case '=' : …
            break
   case '>' : …
            break
}
```

```
switch(compare(a,b)){
   case '<' : …
            break
   case '=' : …
            break
   case '>' : …
            break
}
```

```
switch(compare(a,b)){
   case '<' : …
   case '=' : …
   case '>' : …
}
```

- Try to avoid ambiguity

Selection sort

- Example:
  - Sort 7 integers in A[0..6]

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
unsorted	30	40	10	70	20	60	50
sorted	10	20	30	40	50	60	70

- Overview of Algorithm for n integers in A[0..n-1]
  - Array is partitioned into two lists
    - A[0], …, A[i-1]: sorted list //initially empty
    - A[i], …, A[n-1]: unsorted yet //initially i =0
  - Select the minimum from the unsorted list A[i], …, A[n-1]
    - Start assuming that A[i] is the minimum
    - Scan A[i+1] to A[n-1] to find the new minimum if exists
  - Swap A[i] & the minimum
    - A[0], …, A[i]: sorted list
    - A[i+1], …, A[n-1]: unsorted yet
  - Repeat until sorting is completed
    - A[0], …, A[n-1]: sorted list or
    - A[0], …, A[n-2]: sorted list (& A[n-1]: unsorted list)

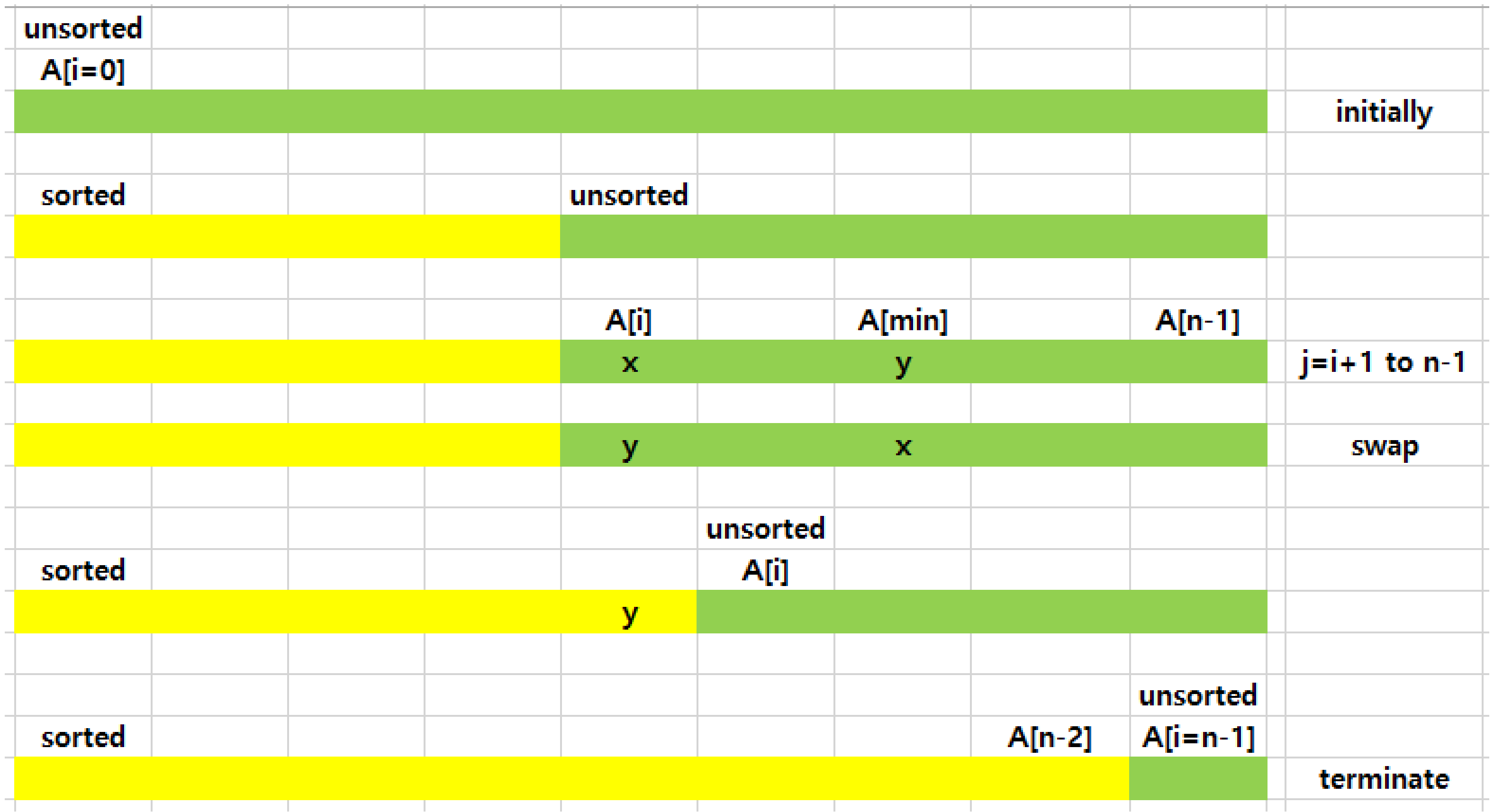
Selection sort

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	
unsorted	30	40	10	70	20	60	50	
			min					
	10	40	30	70	20	60	50	
					min			
	10	20	30	70	40	60	50	
			min					
	10	20	30	70	40	60	50	
					min			
	10	20	30	40	70	60	50	
							min	
	10	20	30	40	50	60	70	
						min		
	10	20	30	40	50	60	70	terminates
sorted	10	20	30	40	50	60	70	

Yellow: sorted list

Selection sort

```
for i ← 0 to n-2 do {
   min ← i
   for j ← i+1 to n-1 do {
      if(A[j] < A[min]) min ← j
   }
   swap(A[i], A[min]) //A[i] ←→ A[min]
}
```



Binary search (이진 탐색)

- Search a *value* in a *sorted* list in A[0..n-1]
- Assuming values in A[] are distinct: A[i] != A[j] (i != j)
- Returns
  - -1: if the value does not exist
  - pos: if A[pos] = *value*
- Binary search
  - Takes advantage of the fact that the list is sorted
  - Compares *value* with A[middle]
  - If A[middle] != *value*, prune either half of the list

Binary search

```
binary_search(A[0..n-1], value) { //A[0], …, A[n-1]
    left ← 0
    right ← n-1
    while ( left <= right ) do {
        mid ← (left + right)/2
        switch(compare(value, A[mid])) {
                case ‘<‘ : right ← mid -1
                case ‘=‘ : return mid
                case ‘>’ : left ← mid +1
        }
    }
    return -1
}
```

Binary search

- Example:

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
10	20	30	40	50	60	70

- Search 10

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
10	20	30	40	50	60	70
			mid			
A[0]	A[1]	A[2]				
10	20	30				
	mid					
A[0]						
10						
mid						
returns 0						

Search 55

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
10	20	30	40	50	60	70
left			mid			right
				A[4]	A[5]	A[6]
				50	60	70
				left	mid	right
				A[4]		
				50		
				L, mid, R		
				right	left	
				does not	exist	(R < L)
				returns -1		

Recursion

- Recursive function (재귀적 함수)
  - A function that calls itself
- Example: n!

```
fact(n) {
    if(n=0) return (1)
    return (n*fact(n-1))
}
```

Recursion

- 작성
  - 함수 정의: input, output
  - 재귀적 호출의 결과 활용
    - application/problem-dependent
    - 함수 작성
  - 종료조건
- Example: n!

```
fact(n) { //input: n (>=0), output: n!
    if(n=0) return (1) //종료 조건: 0! = 1
    return (n*fact(n-1)) //재귀적 호출의 결과로 fact(n) 함수 작성
}
```

- Recursive function
  - 종료 or
  - Recursive calls

Recursion: more examples

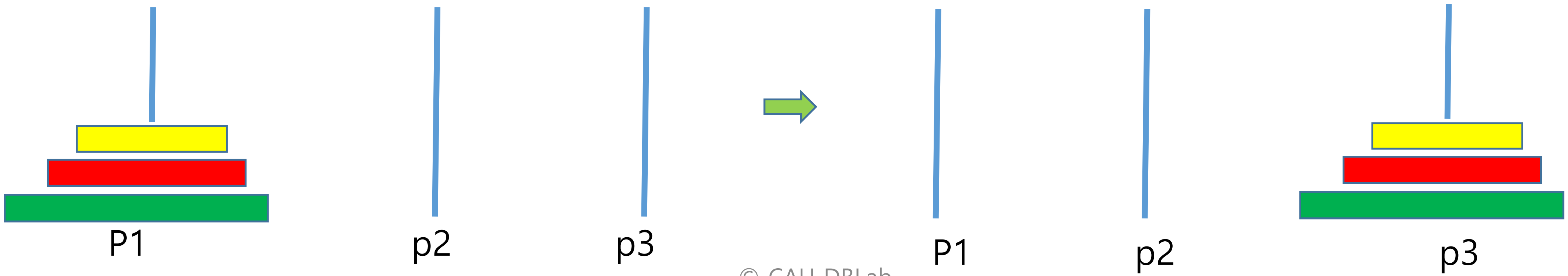
- Binary search
  - R_binary_search(A[], left, right, value)
  - 함수 정의:
    - 정렬된 배열 A[left], …, A[right]에서 value를 탐색
    - If found, (A[pos]=value), return pos
    - Otherwise, return -1
  - 종료 조건: ?
  - 재귀적 호출 및 결과 활용: ?
  - First call: R_binary_search(A[], 0, n-1, value)

Recursion: binary search

```
R_binary_search(A[], left, right, value) {
    if(left > right) return -1 //종료조건
    mid ← (left+right)/2
    switch(compare(value, A[mid])) {
        case ‘<‘ : pos ← R_binary_search(A[], left, mid-1, value)
        case ‘=‘ : pos ← mid
        case ‘>’ : pos ← R_binary_search(A[], mid+1, right, value)
    }
    return pos
}
```

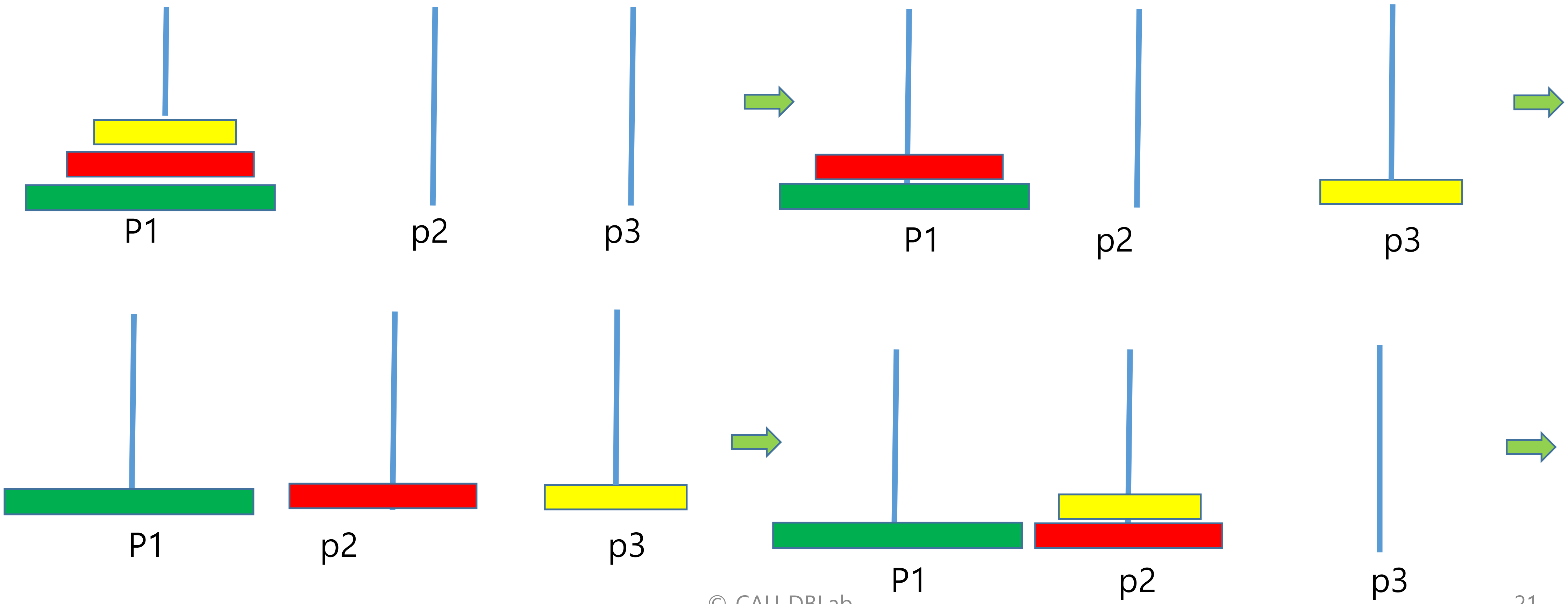
Recursion: Towers of Hanoi

- 3 poles: pole 1, 2, 3 (p1, p2, p3 or just 1,2,3 for short)
- Tower
    - p1: n(=64) disks with different diameters
    - smaller disk on top of bigger ones
- Goal: move the tower from p1 to p3
- Rules
    - One disk at a time
    - Bigger disk cannot be placed on top of smaller one



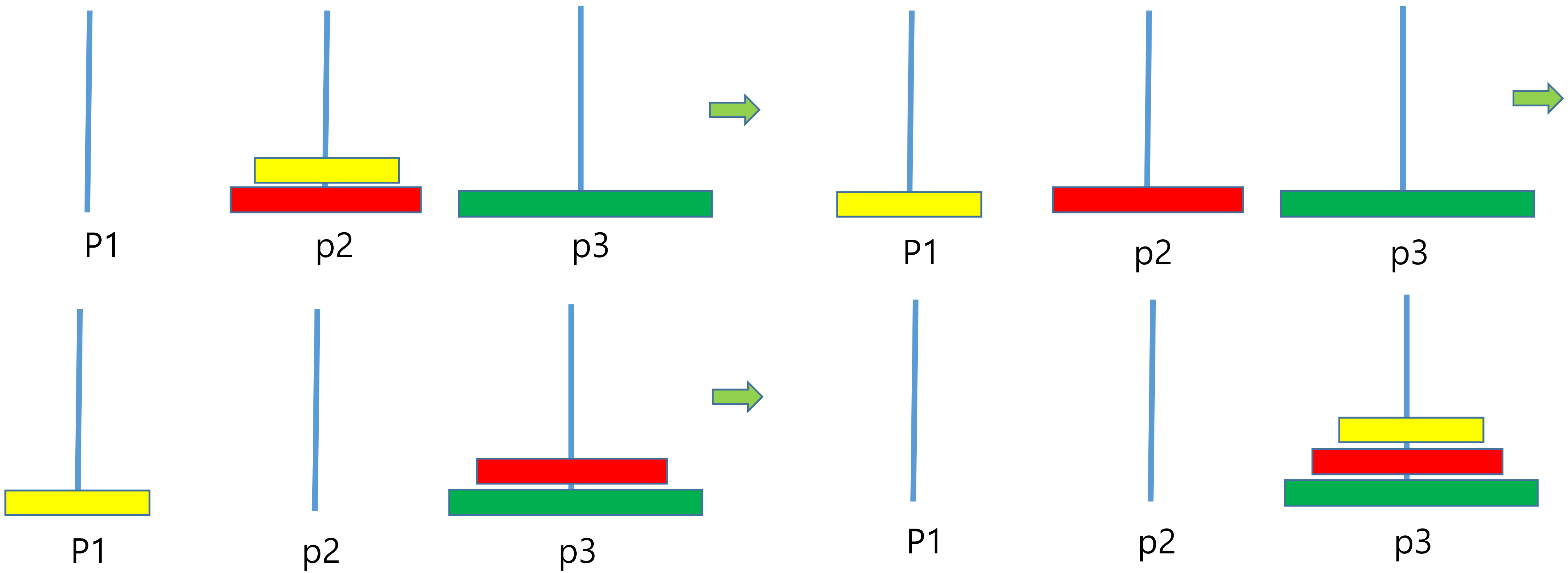
Towers of Hanoi

- Example: n=3



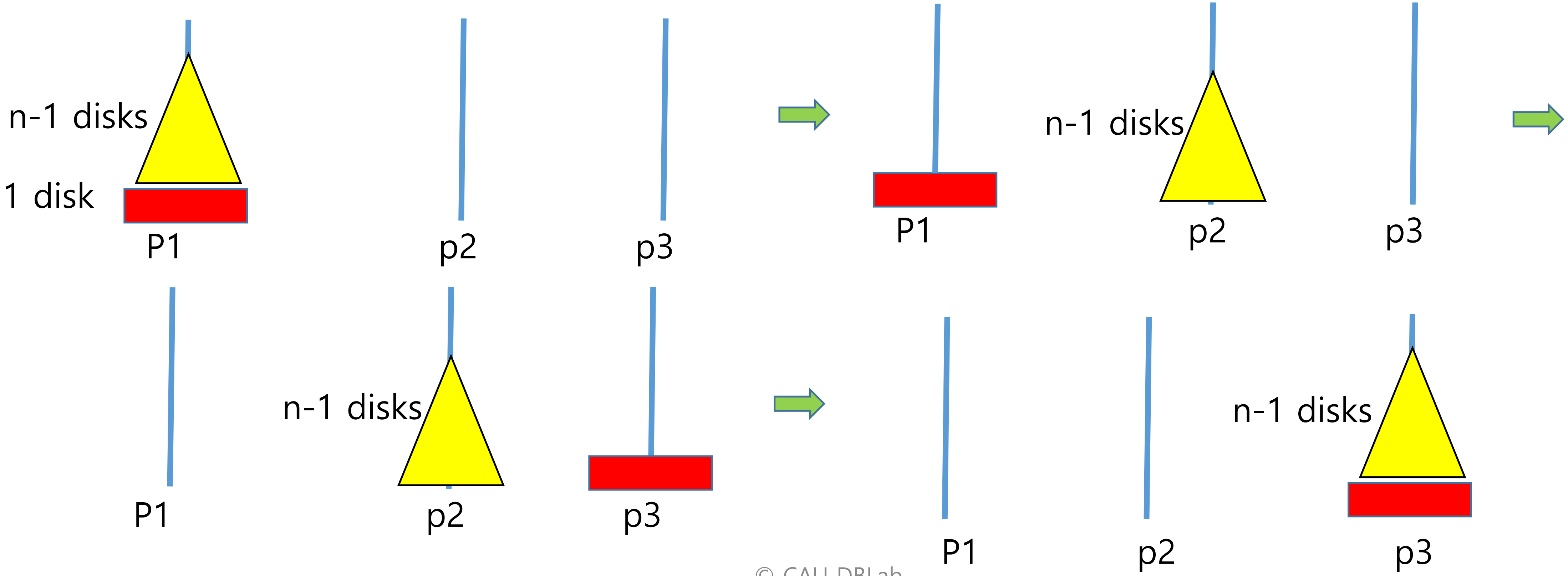
Towers of Hanoi

- Example: n=3



Towers of Hanoi

- n=1, 2: trivial
- n=3: as shown
- n=4, 5, …, 64 ? : recursive solution

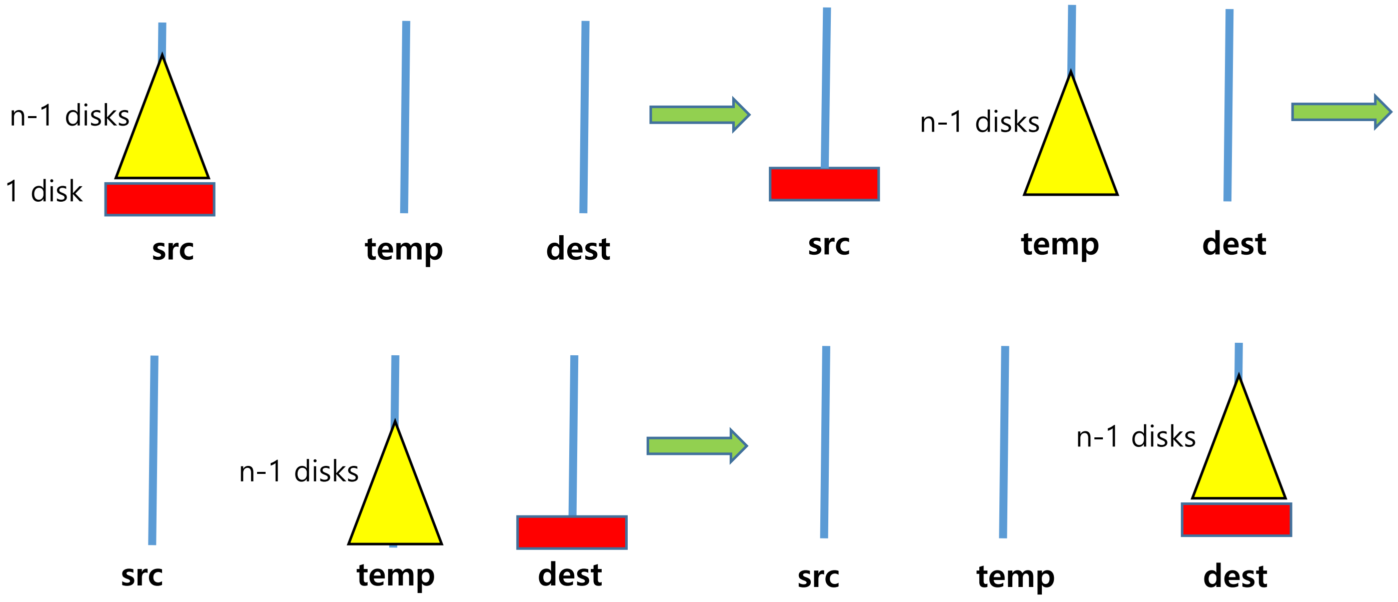


Towers of Hanoi

- 함수 TH(n, src, dest, temp)
  - n개 disk 탑을 pole src에서 pole dest로 move
  - pole temp를 이용
- First call: TH(64, 1, 3, 2)
- Recursive calls

```
TH(n, src, dest, temp) {
    TH(n-1, src, temp, dest)
    TH(1, src, dest, temp)
    TH(n-1, temp, dest, src)
}
```

- 종료조건?



Towers of Hanoi

```
TH(n, src, dest, temp) {
    if(n=1) { //종료조건
        move a disk from pole src to pole dest
        return
    }
    TH(n-1, src, temp, dest)
    TH(1, src, dest, temp)
    TH(n-1, temp, dest, src)
}

TH(n, src, dest, temp) {
    if(n=1) then move a disk from pole src to pole dest //종료조건: without explicit return
    else {
        TH(n-1, src, temp, dest)
        TH(1, src, dest, temp)
        TH(n-1, temp, dest, src)
    }
}
```

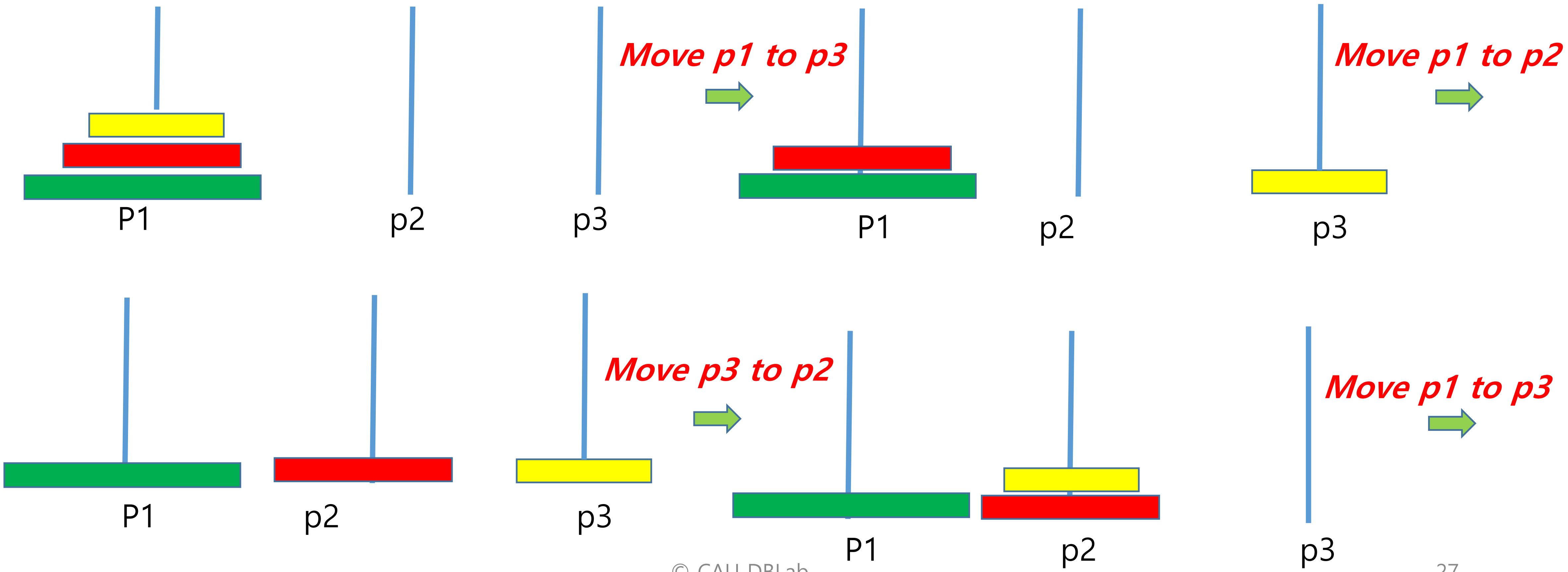
Towers of Hanoi

```
TH(n, src, dest, temp) {
    if(n=0) return //종료조건: if n=0, do nothing & just return
    if(n>=1) {
        TH(n-1, src, temp, dest)
        move a disk from pole 'src' to pole 'dest'
        TH(n-1, temp, dest, src)
    }
}

TH(n, src, dest, temp) {
    if(n>=1) {
        TH(n-1, src, temp, dest)
        move a disk from pole 'src' to pole 'dest'
        TH(n-1, temp, dest, src)
    }
    //종료조건: implicitly given: if n=0, do nothing & just return
}
```

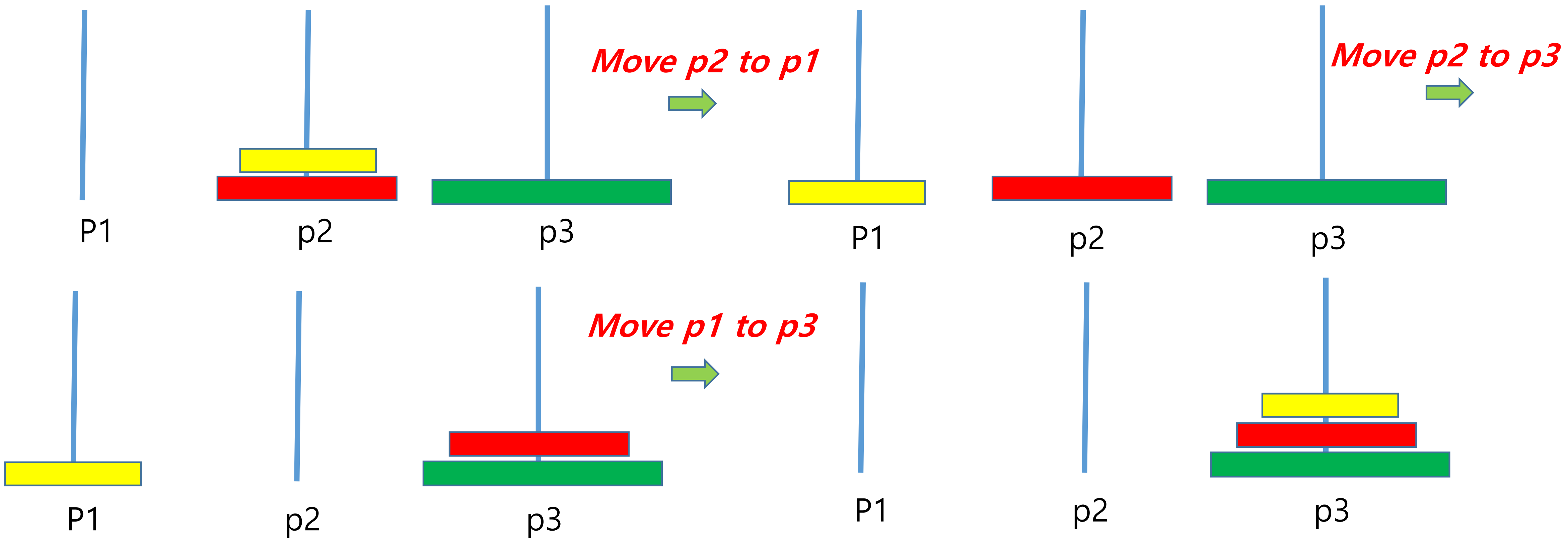
Towers of Hanoi: Print sequence of moves

- Example: n=3



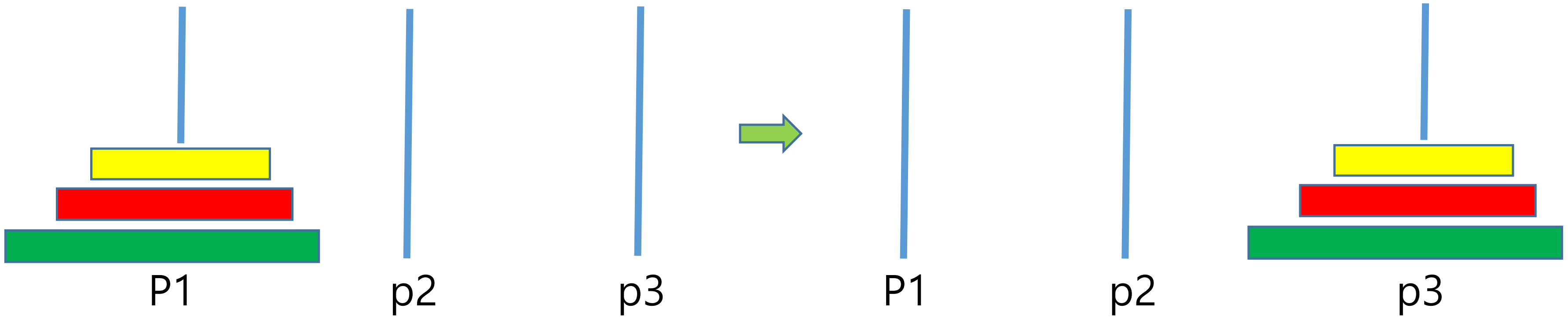
Towers of Hanoi: Print sequence of moves

- Example: n=3



Towers of Hanoi: Print sequence of moves

- Example: n=3
- Sequence
  - 1 to 3
  - 1 to 2
  - 3 to 2
  - 1 to 3
  - 2 to 1
  - 2 to 3
  - 1 to 3



Towers of Hanoi: Print sequence of moves

- 함수 TH_seq(n, src, dest, temp)
  - n개 disk 탑을 pole src에서 pole temp를 이용하여 pole dest로 move하는 sequence를 출력

```
TH_seq(n, src, dest, temp) {
    if(n=1) then print("₩nsrc to dest")
    else {
        TH_seq(n-1, src, temp, dest)
        TH_seq(1, src, dest, temp)
        TH_seq(n-1, temp, dest, src)
    }
}
```

TH_seq(3, 1, 3, 2) 호출 결과:

```
1 to 3
1 to 2
3 to 2
1 to 3
2 to 1
2 to 3
1 to 3
```

Performance Analysis

- Algorithm analysis
- Space complexity and time complexity
- Time complexity
  - Measurement/experiment
    - Run the program on some data instance and measure running time
    - Limitations
      - Limited instances of data
      - Dependent on the system capacity and capability
  - Theoretical approach
    - count execution steps
    - Worst, best, average case analysis
      - Example: number of compare operations in binary search
    - Big “oh” notation

Big “oh” notation

- Worst case analysis
- O(n): order of n
- 대표적 time complexities: $O(\log n)$, $O(n)$, $O(n \log n)$, $O(n^2)$, $O(n^3)$, $O(2^n)$
- O(1)
  - order of one
  - constant time independent of n
- Lower order terms and constant factors are ignored
  - e.g., $n^2 + 3n - 3$: $O(n^2)$
- Limitations: constant factors are ignored
  - n+10000 vs. n+1: both O(n)
  - n+1 vs. 2n: both O(n)
  - 10000n: O(n) vs. $2n^2$: $O(n^2)$
- Polynomial time algorithm
- Polynomial time vs. exponential time
- Logarithmic time vs. polynomial time

Example: Selection sort

	COUNT
for i ← 0 to n-2 do {	n
min ← i	n-1
for j ← i+1 to n-1 do {	X+(n-1)
if(A[j] < A[min]) min ← j	X
}	
swap(A[i], A[min])	n-1
}	
TOTAL	2X+4n-3

$$X = (n-1)+(n-2)+(n-3)+\ldots+2+1$$
$$= (n-1)*n/2$$

$$\text{Total count} = (n-1)*n + 4n - 3$$
$$= n^2 + 3n - 3$$

Time complexity: $O(n^2)$

i	0	1	2	...	n-2
j	1..n-1	2..n-1	3..n-1	...	n-1..n-1
count	n-1	n-2	n-3	...	1

Example: nested loop

			for i ← 1 to n do {		
		$O(1)$	count ← 0	$O(n)$	
			for j ← 1 to n do {		
			if(A[i] = B[j]) {		
$O(n)$	$O(n)$	$O(1)$	print(i, j)	$O(n^2)$	$O(n^2)$
			count ← count + 1		
			}		
			}		
		$O(1)$	print(A[i], count)	$O(n)$	
			}		

Example: n!

- Recursive function

```
fact(n) {
      if(n=0) return (1)
      return (n*fact(n-1))
}
```

- Let C(n) be the total step counts of fact(n).
- C(n) = C(n-1) + c where c is some constant
- $C(n) = \begin{cases} C(n-1) + c, & n > 0 \\ 1, & n = 0 \end{cases}$
- C(n) = C(n-1) + c = C(n-2) + 2*c = … = C(n-k) + k*c
- k=n, C(n) = C(0) + c*n = c*n + 1
- Time complexity: O(n)