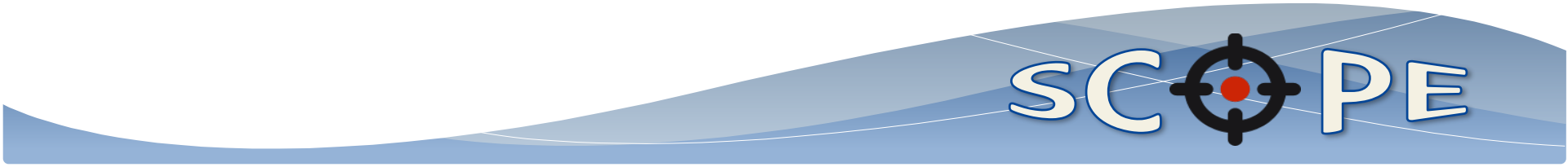
Introduction to Programming(II)

Data Structure and Algorithm

李哲榮



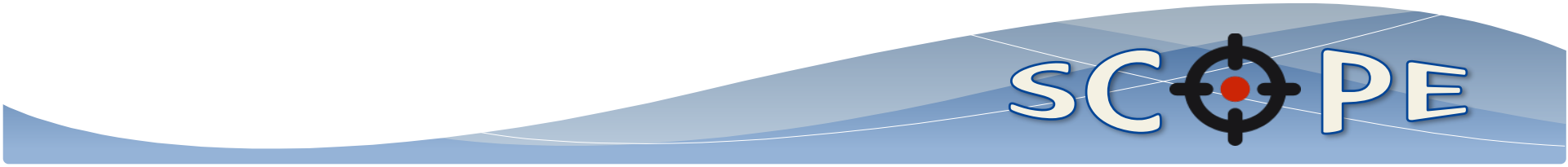
Josephus problem

Flavius Josephus is a Jewish historian living in the 1st century. According to his account, he and his 40 comrade soldiers were trapped in a cave, surrounded by Romans. They chose suicide over capture and decided that they would form a circle and start killing themselves using a step of three. As Josephus did not want to die, he was able to find the safe place, and stayed alive with his comrade, later joining the Romans who captured them.

How to find the safe place

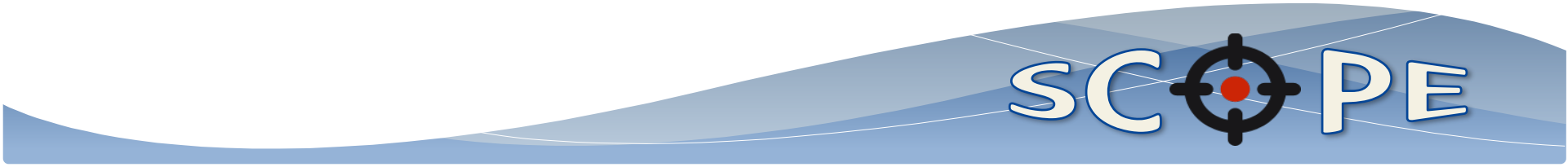
Problem description

- n people form a circle, numbered from 1 to n. Starting from the number 1 person, killing every m^{th} person, who will be the last one?
- This problem has two parameters: n and m, and the output is an integer between 1 and n.
- Problem: write a program to calculate the output.



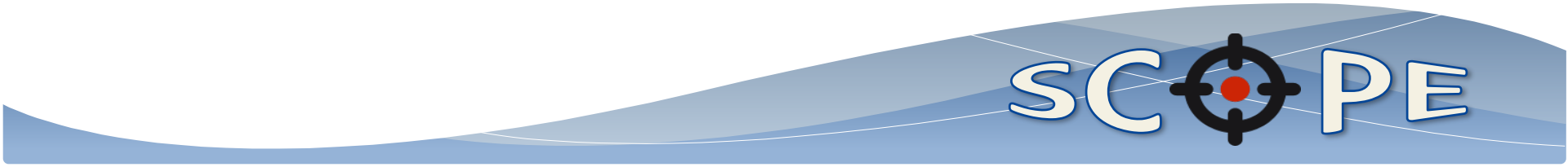
Algorithm

- An effective method for solving a problem using a finite sequence of instructions.
  - It need be able to solve the problem. (correctness)
  - It can be represented by a finite number of (computer) instructions.
    - Each instruction must be achievable (by computer)
  - The more effective, the better algorithm is.
    - How to measure the “efficiency”?

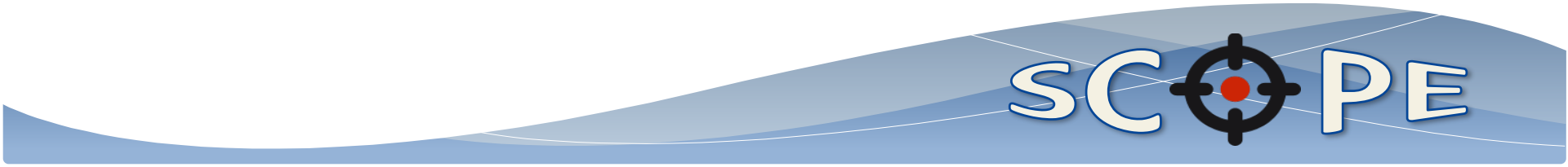


Outline

- The first algorithm
- Time complexity
- The second algorithm
- The third algorithm
- The fourth algorithm
- The fifth algorithm

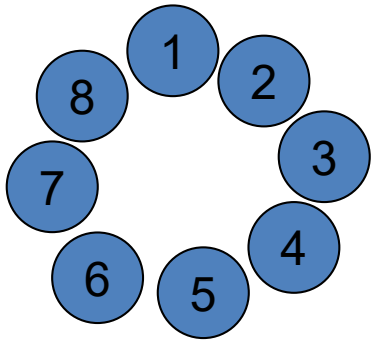


THE FIRST ALGORITHM



A simpler version

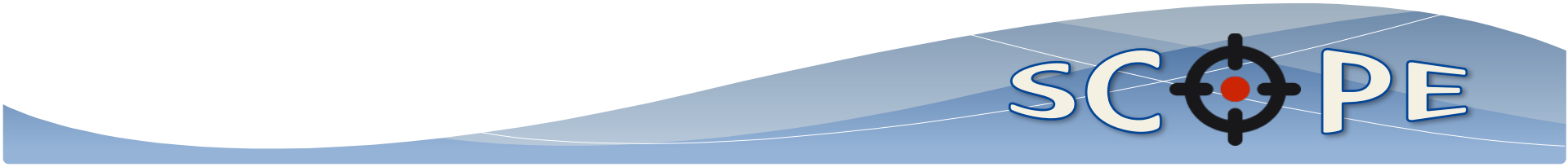
- Let's consider a similar problem
  - There are *n* person in a circle, numbered from 1 to *n* sequentially.
  - Starting from the number 1 person, every 2nd person will be killed.
  - What is the safe place?
- The input is *n*, the output *f*(*n*) is a number between 1 and *n*.
  - Ex: f(8) = ?



The first algorithm: simulation

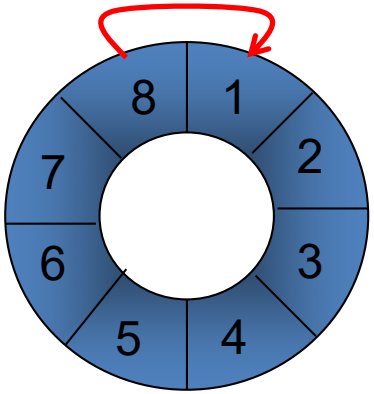
- We can find $f(n)$ using simulation.
  - Simulation is a process to imitate the real objects, states of affairs, or process.
  - We do not need to "kill" anyone to find $f(n)$.
- A nature way to represent people is using an integer array of size n

```
PEOPLE *people;
people = (PEOPLE*)malloc(n*sizeof(PEOPLE));
```

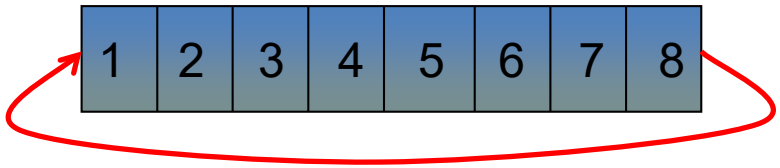


People in a circle

- Array is a linear structure. How to use it to present people in a circle.
- Suppose the variable `current` is used to point the current person.

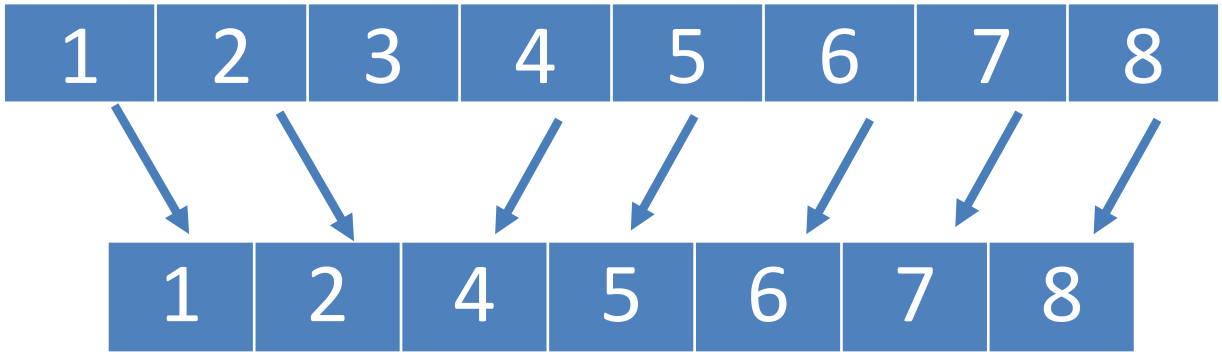



```
if (current == n)
     current = 1;
else
     current++;
```

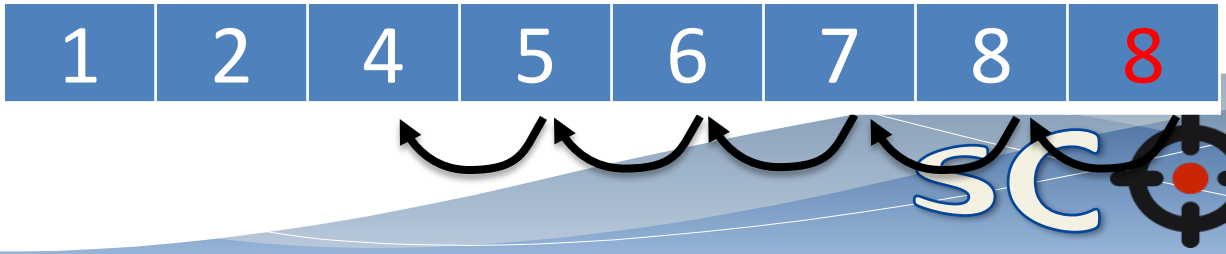


How to kill the next person?

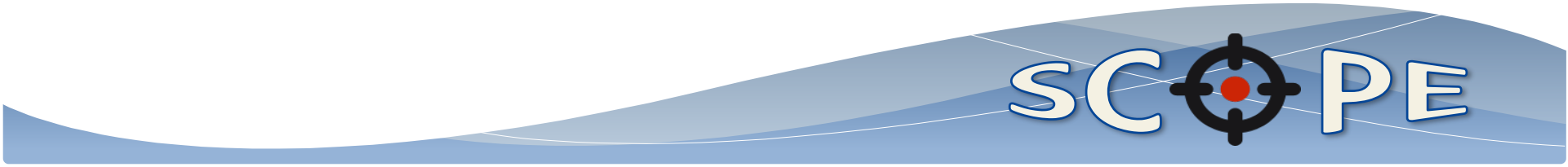
- Keep the ALIVE people in the array and remove the DEAD person.
- Ex: if n = 8, m = 3, after killing the 3rd person,



Or



TIME COMPLEXITY

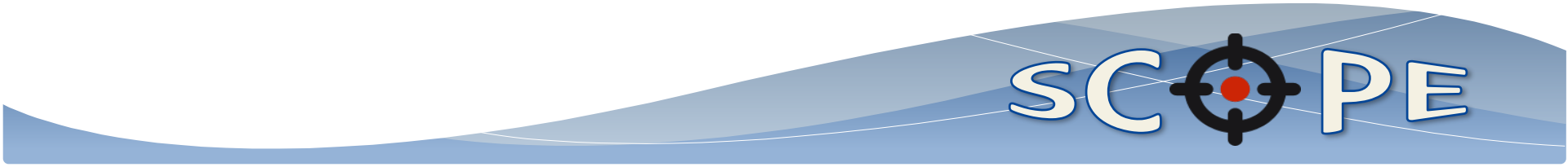


Efficiency of an algorithm

- The efficiency of an algorithm is usually measured by the number of operations
  - assignment, comparison, arithmetic operation
- Operation count need be expressed as a function of problem size *n*.
  - For example, $5n^3+2n^2+10n\log n$ (1)
- Computer scientists are interested in the operation count for large n, determined by the order of the dominant term.
  - n^3 is the order of the dominant term of (1)

Big-O Notation

- f(x) = O(g(x)) if and only if there is a positive constant M such that for all sufficiently large *x*, the absolute value of *f*(*x*) is at most M multiplied by the absolute value of *g*(*x*).
- i.e. $|f(x)| \leq M|g(x)| \text{ for all } x \geq x_0.$
- Example: $5n^3+20000\ n^2 = O(n^3)$
  - Why?
  - But $5n^3+2^n$!= $O(n^3)$, why?



Operation count of the 1st alg

1. **Allocate memory**: It only performs once.
2. **Count people**:
   - You can think this as another problem. Suppose there are infinity number of people in a list, and you need to kill one person for every m persons until n – 1 people are killed. In that case, you need to count m*(n–1) people to finish the job.
   - Big-O notation O(mn)
   - If we keep m as a constant, it is O(n).

3. Remove DEAD people

- After killing the first person, the program moves n-m elements in the array
- After killing the second person, the program moves n-2m elements in the array
- Let's call the walking of the array from the first element to the last element "a **round**".
  - In the first round, the number of movements is

$$(n-m)+(n-2m)+\ldots(n-km)$$

where k is the largest integer <= n/m.

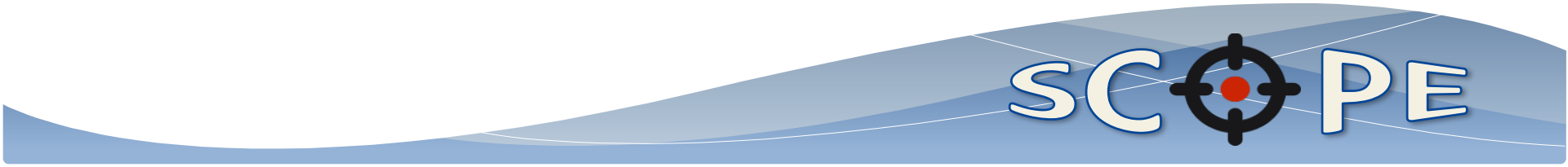
3. Remove DEAD people--cont

- Let k=n/m. The summation equals to $n^2/2m$.
- In the second round, because the total number of people becomes n-k = n(m-1)/m, the number of movements is $n^2(m-1)^2/2m^3$.
- If we let $a=n^2/2m$ and $r=(m-1)^2/m^2$, the movements in the first round and in the second round are a and ar respectively.
- in general, the number of movements in the ith round is ar^{i-1}.

3. Remove DEAD people--cont

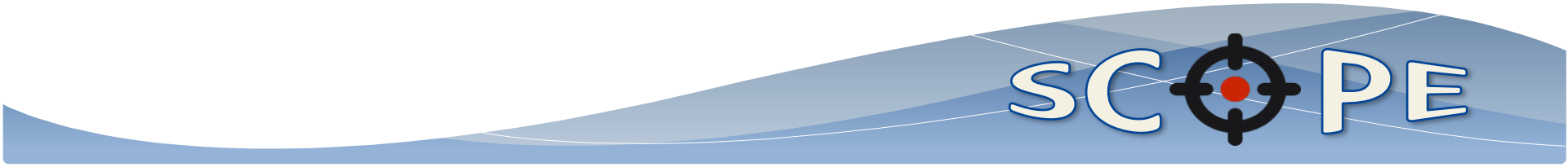
- The summation $a + ar + ar^2 + \ldots$ is roughly equal to $a/(1-r) = n^2/4$.
- In the Big-O notation, it equals to $O(n^2)$.

- Can you get an estimation of the time complexity without carefully analysis?



Time complexity of the 1st algo

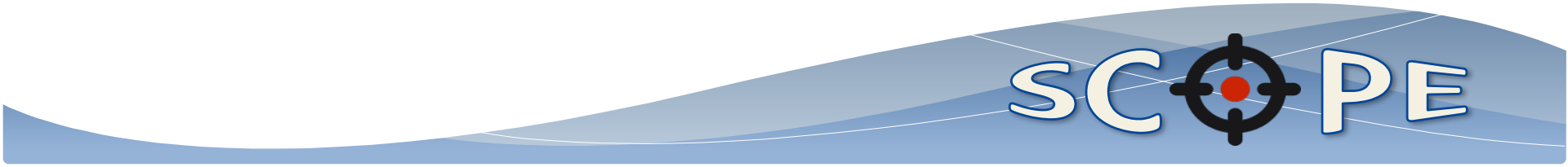
- The time complexity of the 1st algorithm is the summation of all three steps
  - Memory allocation: O(1)
  - Count people: O(n)
  - Remove people: $O(n^2)$
- The summation O(1) + O(n) + $O(n^2)$ equals to $O(n^2)$. Why?



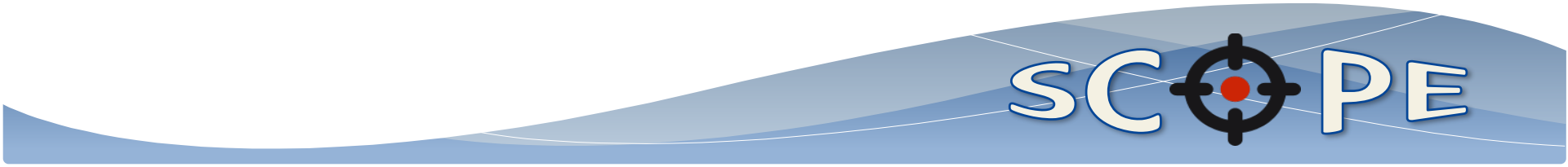
Trend of some functions

- In the big-O notation, we can only keep the function that grows fastest with n.

	log(n)	n	n log n	n^2	2^n	n!
n = 1	0	1	0	1	2	1
n = 10	1	10	10	100	1024	$\sim 10^6$
n = 100	2	100	200	10000	$\sim 10^{30}$	$\sim 10^{157}$



THE SECOND ALGORITHM

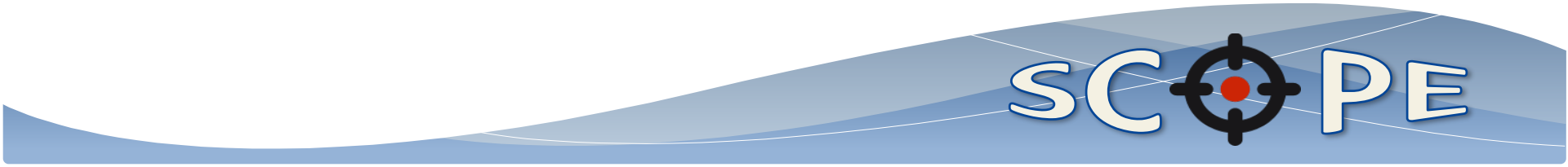


Mark the “dead” one

- We do not need to actually “remove” the dead ones, but just mark them dead.
- In the implementation, you can use an array to represent the status of a person.
- Ex: n=8, m=2.

Time complexity of the 2nd algo

- In the first round, the program can take one step to find the next ALIVE person.
- After the first round, there are roughly n/m people are marked DEAD. So in the second round, we need to skip those DEAD people.
- In the second round, the number of DEAD people becomes $n/m + (n-n/m)/m = 2n/m - n/m^2$ roughly.

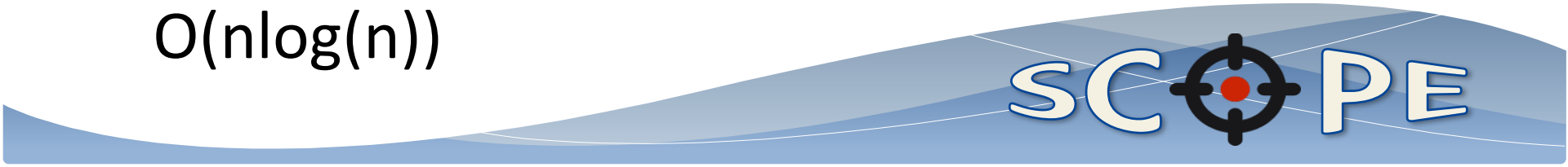


Time complexity of the 2nd algo

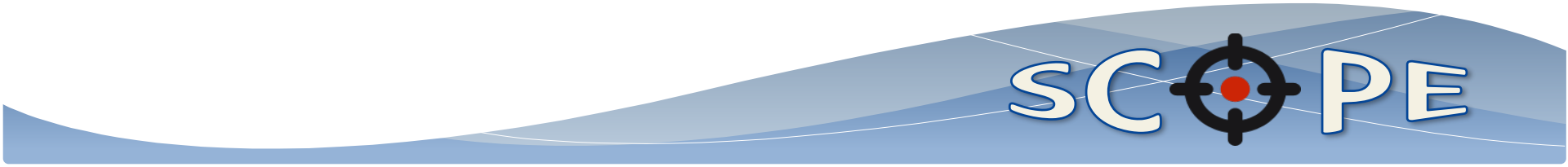
- Let a=n/m and r=(m – 1)/m. The number of dead people increased are a, ar, ar^2,…
- How many rounds to kill n – 1 people?
  - Suppose it needs k rounds.

$$a + ar + \ldots + ar^{k-1} = n - 1$$

  - The solution is $k = \log_{1/r}(n)$.
  - For m=2, 1/r = 2. $k = \log_2(n)$.
- The time complexity of the 2nd algorithm is O(nlog(n))

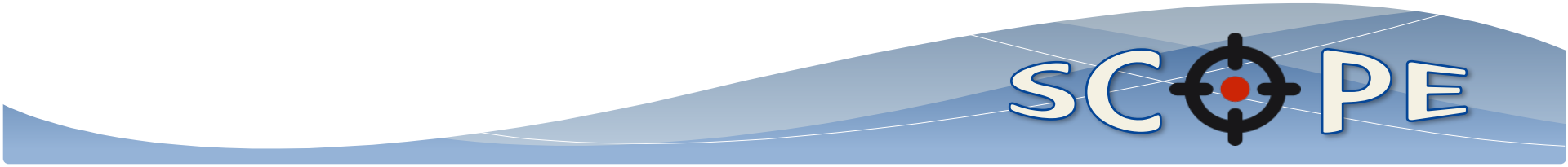


THE THIRD ALGORITHM



How to make it run faster?

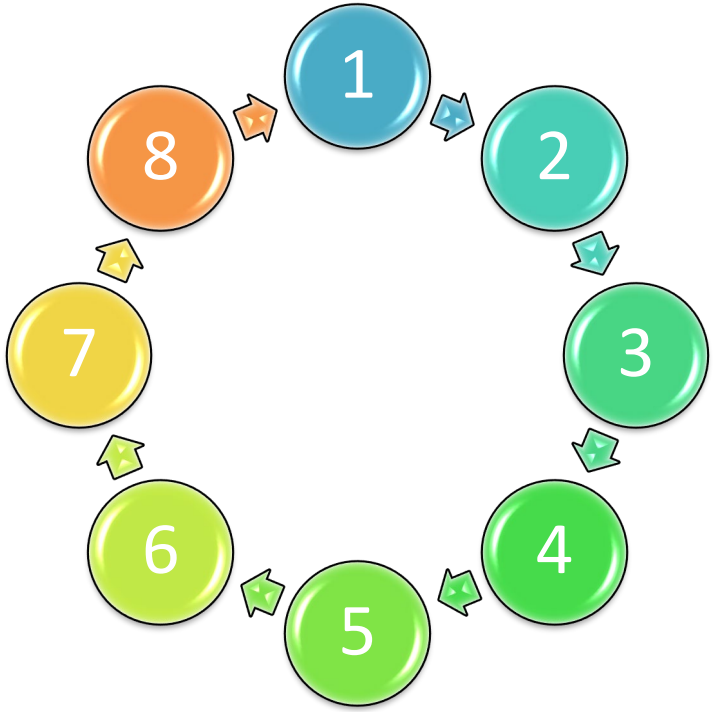
- Two key steps: (1) find the next and (2) kill the next
- If we can make each step O(1) time, we can make the overall time O(n)
- Using array cannot achieve both ➔
- We need a new data structure ➔
- Circular linked list (struct + pointer)



Circular linked list

- The structure of a node

```
typedef struct node {
    int id;
    struct node *next;
} Node;
```

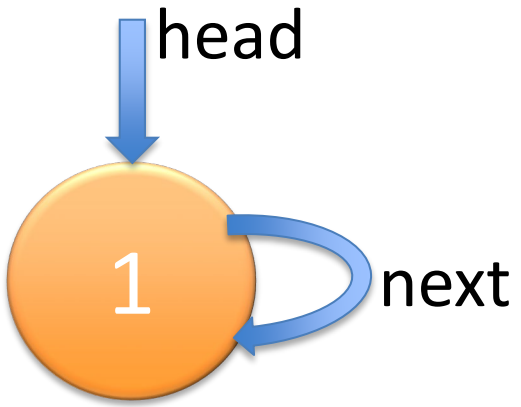


- Operations:
  - Initialization, add a node, delete a node, check the number of nodes

Initialization

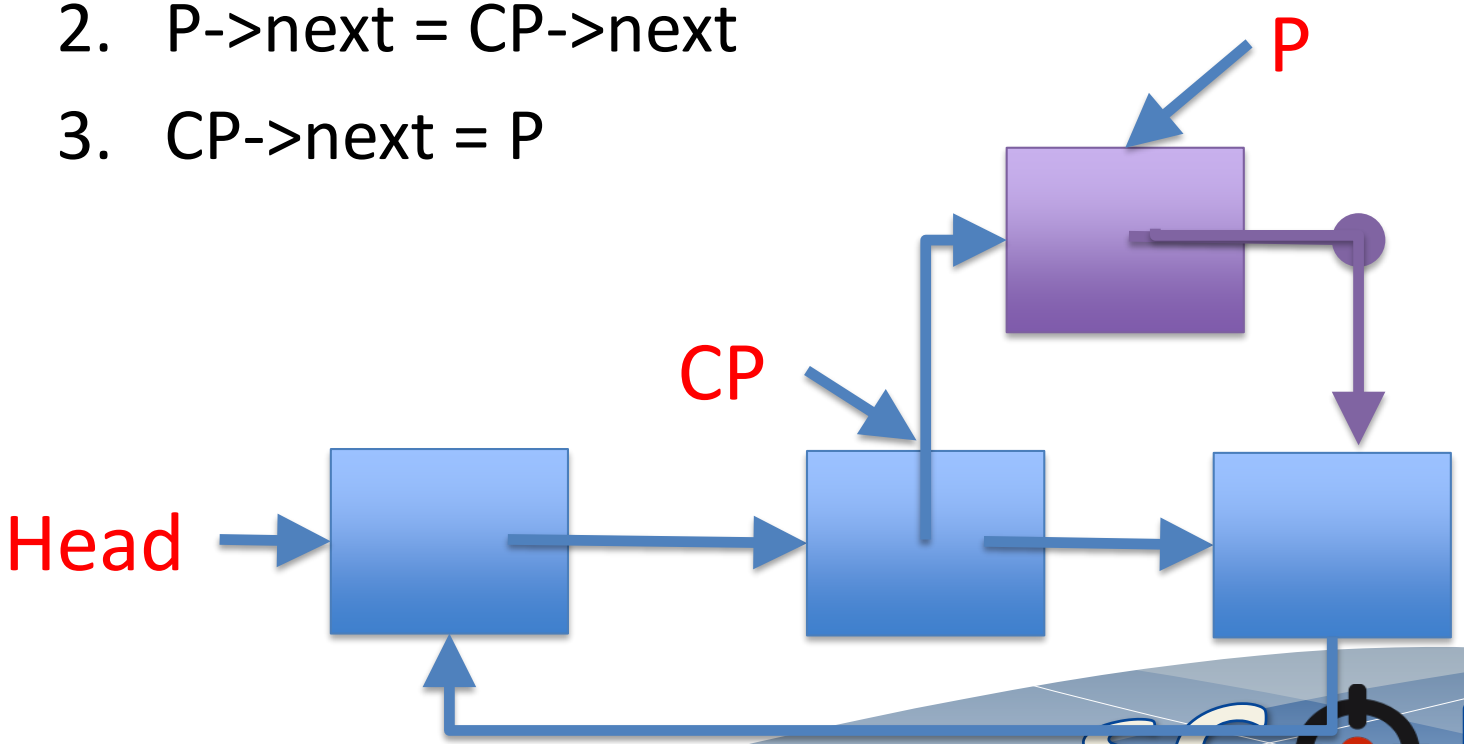
```
Node *head;
head = (Node*) malloc(sizeof(Node));
head->id = 1;
Head->next = head;
```

```
Node *CP = head;
```



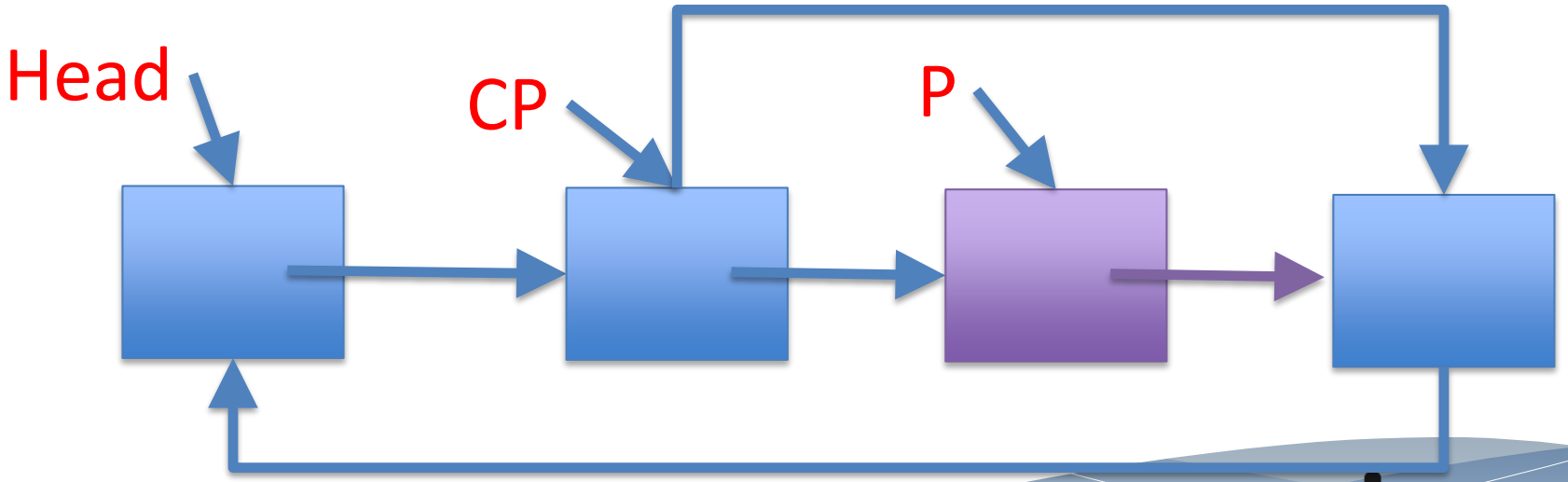
Insert a node

- To insert a node after the node pointed by CP
  1. Allocate a node pointed by P
  2. P->next = CP->next
  3. CP->next = P



Delete a node

- To delete a node after the node pointed by CP
  1. Let a pointer P point to that node, P=CP->next
  2. CP->next = P->next
  3. free(P)



When to stop?

- When there is only one node left, we need to stop?
- How to implement?
  1. Using a counter to count how many nodes are still alive.
  2. Check if CP->next ==CP