Who am I

Prof. Wonjun Kim

Engineering Building C, 424-2

E-mail : wonjkim@konkuk.ac.kr

Research Interest :

Computer vision, image/video processing, pattern recognition, machine learning

Homepage : https://dcvl.konkuk.ac.kr/

(You can download lecture notes in my homepage)

Embedded Data Structure
(Lecture Note 1)

Introduction to Data Structure

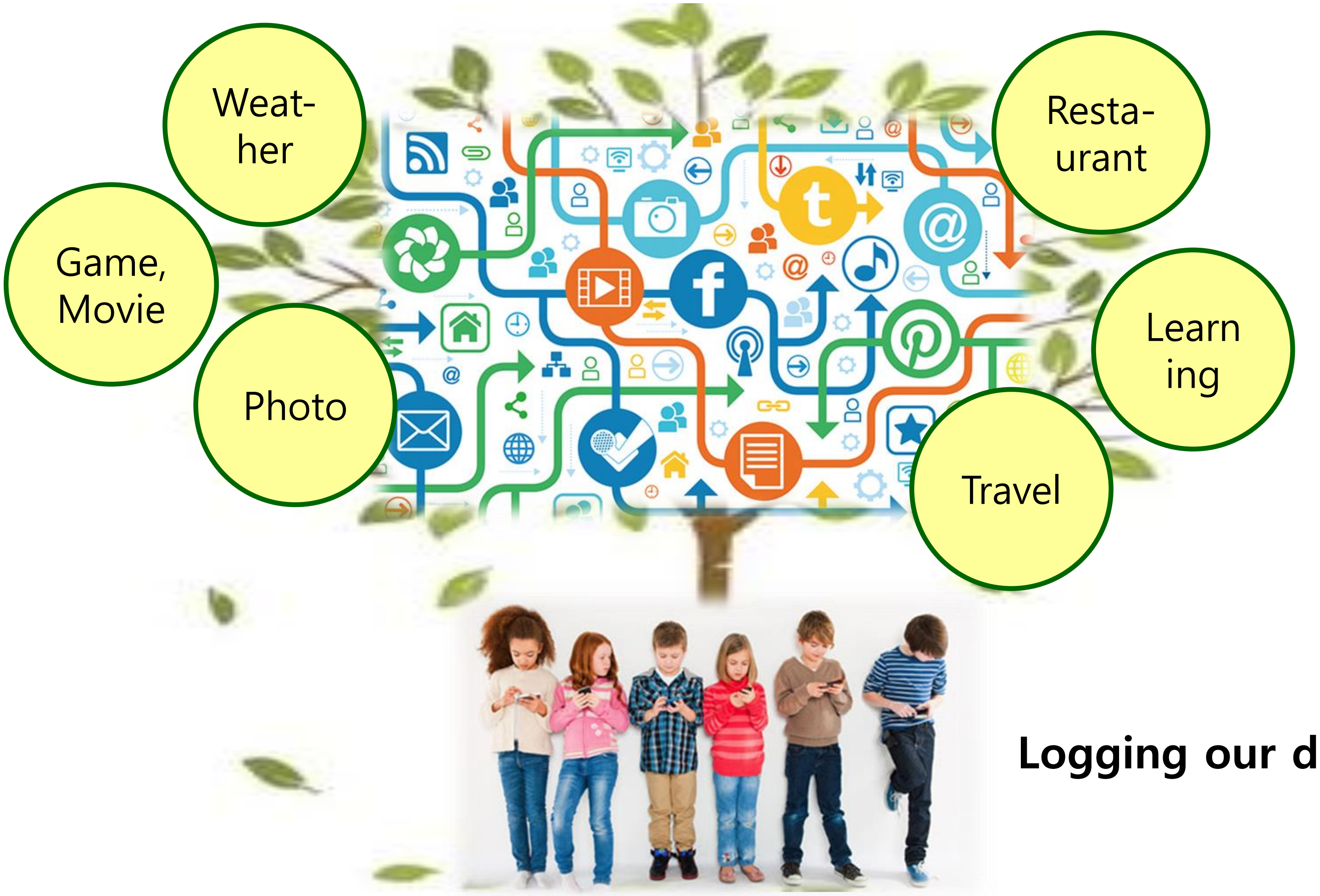
2019. 3. 4

Prof. Wonjun Kim

School of Electrical and Electronics Engineering

Before Starting … (1/2)

- **Revolution of data !**



Weat-her

Game, Movie

Photo

Resta-urant

Learn ing

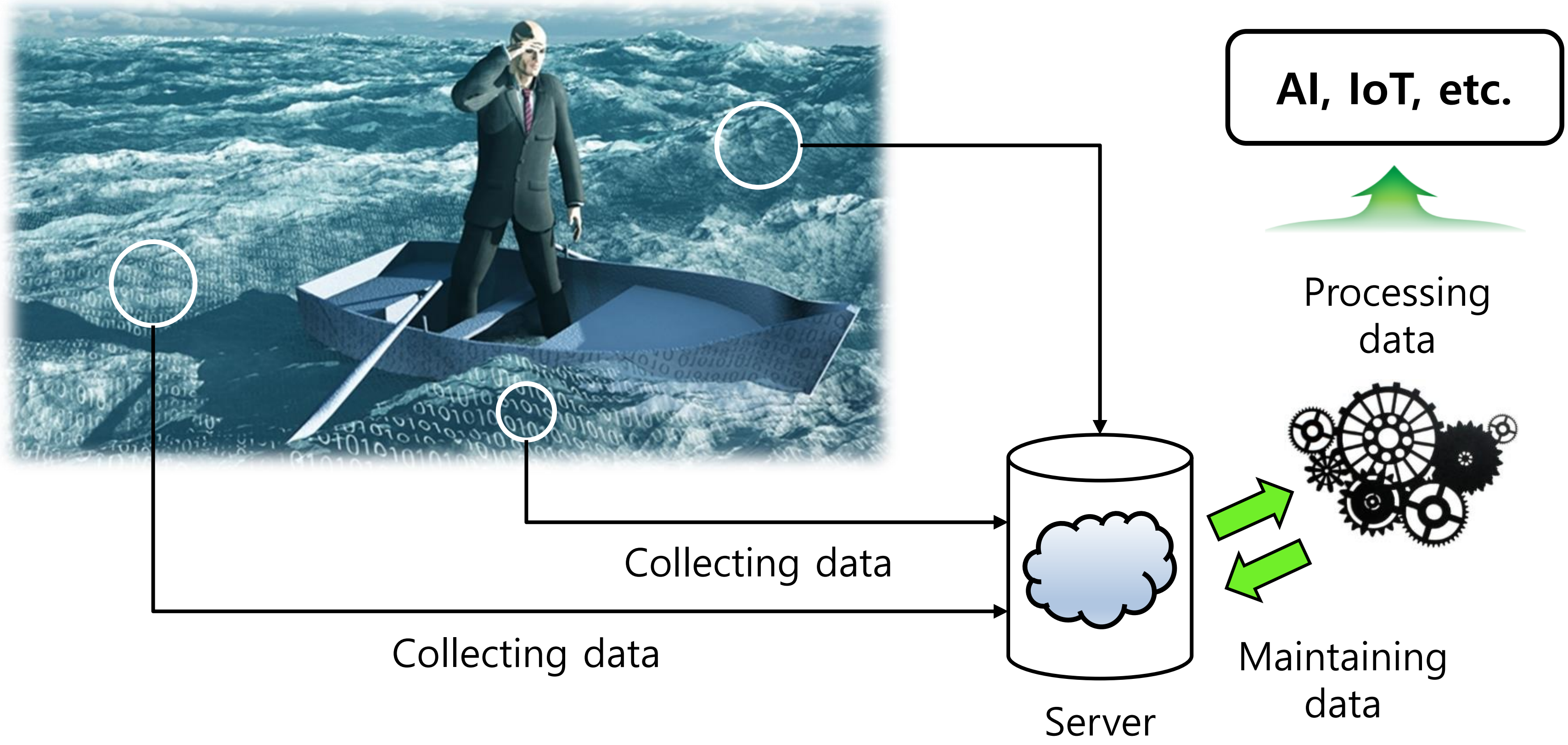
Travel

Logging our daily life

In our daily life

Before Starting … (2/2)

- Key point : finding the useful "meaning" in data
  - To do this, handling and managing data is very important



AI, IoT, etc.

Processing data

Collecting data

Collecting data

Server

Maintaining data

Class Schedule

What we learn in this class …

- Introduction to data structure
- Revisit C programming (array, pointer, memory, etc.)
- Linked list
- Stack and queue
- Data labeling (by using stack)
- Tree structure
- Sorting algorithms
- Searching algorithms
- Practical examples

Lecture Note

※ Language : C or (C++)

Introduction (1/8)

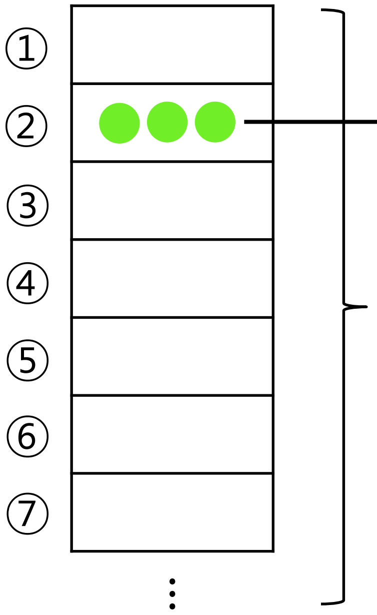
▪ Data by C programming

• Why C (vs. Matlab, Python, etc.) ?

* C is the only one that handles the address of the memory

→ Efficient to access the memory

→ Critical to the embedded applications



How can we copy (or move) this data to ⑤ ?

: In C, ( ) can make it possible !

Memory structure is linear !

※ Difference btw. global and local allocation

Introduction (2/8)

Why C for embedded programming ? (summary)

- **Two reasons : memory and hardware control**

1) Only C performs in an efficient way under limited memory environments

: Other visual languages require quite a lot memory spaces

※ The capacity of program executable binary is very small in C

2) C can efficiently assign values in a specific address using "pointer"

: Data can be easily handled with hardware controls

➡ **That's the reason why we focus on "pointer" in C programming**

➡ **For efficiently managing the memory space,
memory assign/free needs to be conducted with pointers
(except for the global data)**

Introduction (3/8)

▪ How about big data ?

- **Problem of memory allocation (> 10 GB)**

※ Find the memory limit for Visual Studio 2017 (64bit)

Alternatives :

1) Divide the data and use the multi-core processor

: Be careful for handling data seamlessly

2) Via GPU (e.g., Titan Xp (12GB memory))

: With CUDA coding → efficiency ↑ (c.f. deep learning scheme)

- **Monitoring the working memory**

: To prevent the stack overflow, allocation-free needs to be fully conducted

← It will be helpful to handle big data in your systems

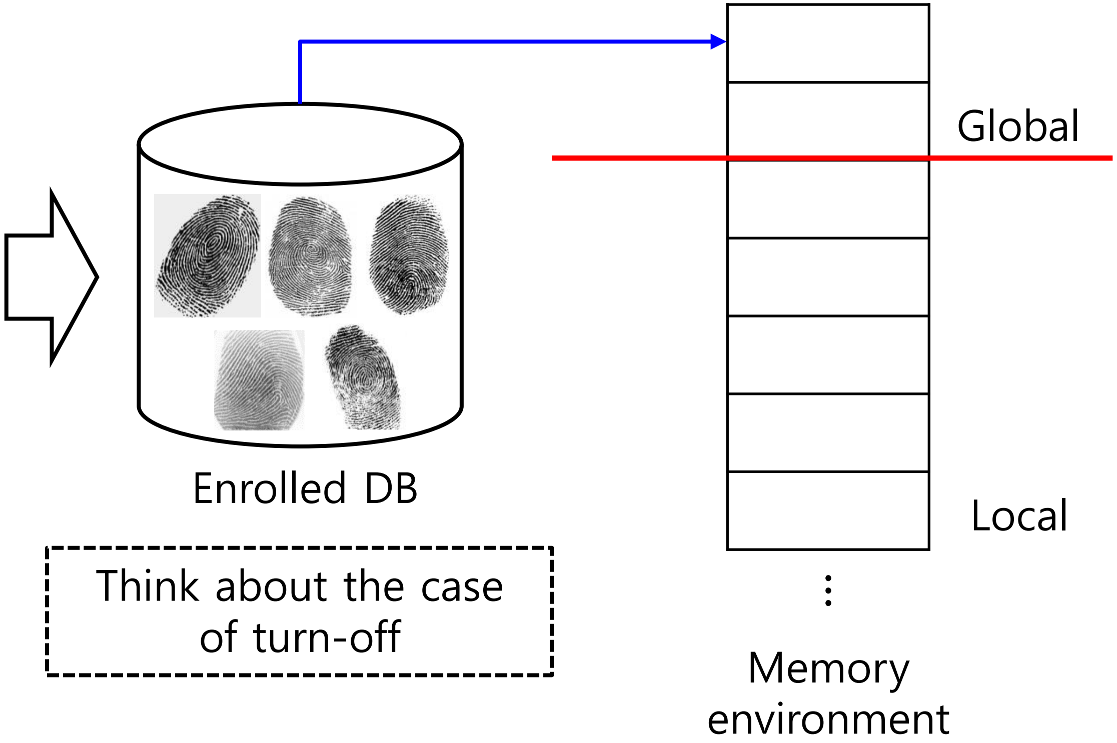
Introduction (4/8)

▪ Example : embedded system

• Example of data handling : fingerprint recognition

※ Difference between global and local allocation (data store)

[Fingerprint recognition (S10)]



Think about the case of turn-off

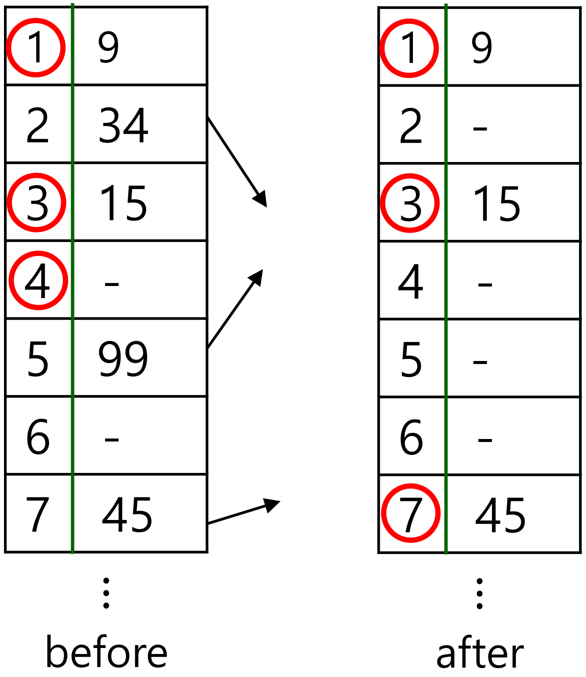
Introduction (5/8)

▪ Example : embedded system – cont'd

- **Efficient use of memory is very important !**

※ Data needs to be clearly structured and maintained for efficiency

[Think about the space]



①	9
2	34
③	15
④	-
5	99
6	-
7	45

⋮

before

①	9
2	-
③	15
4	-
5	-
6	-
⑦	45

⋮

after

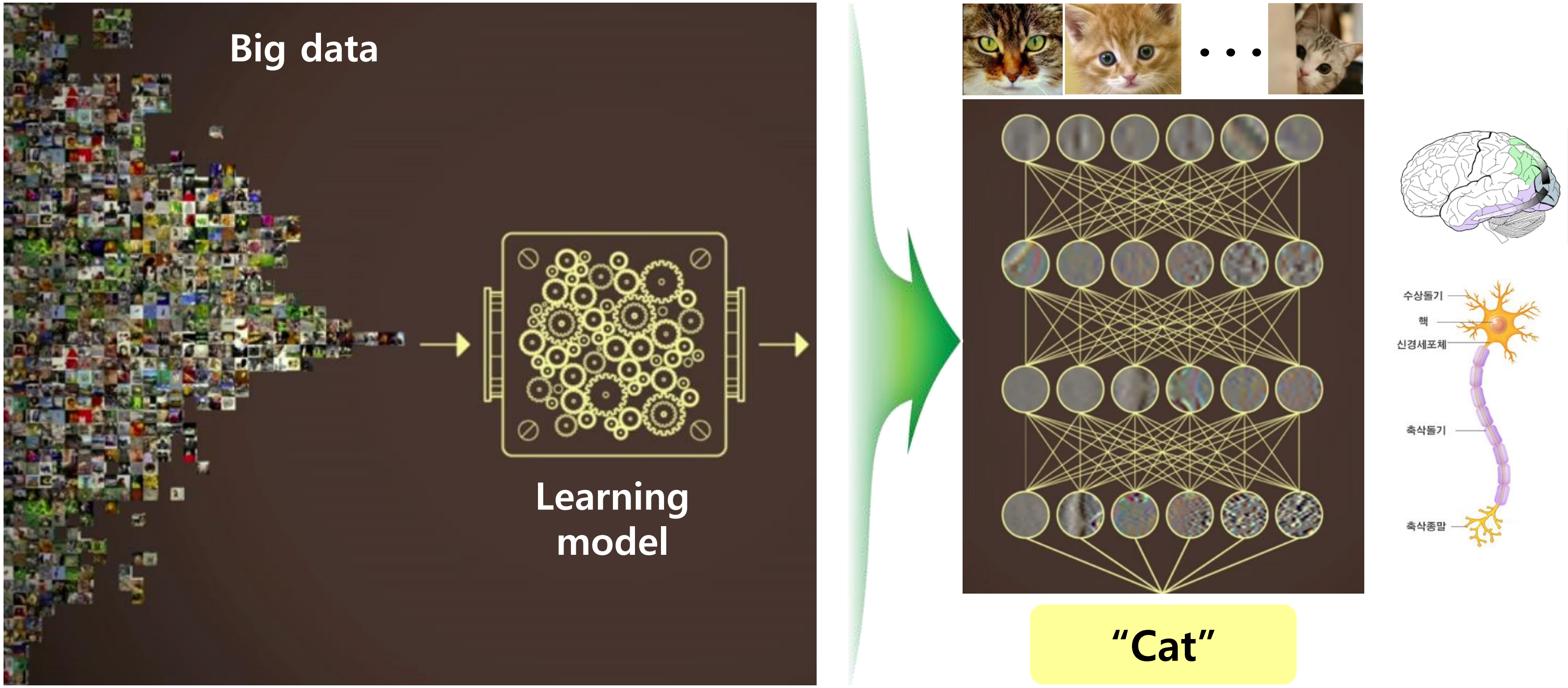
[Think about the memory space]

Introduction (6/8)

▪ Example : deep learning with big data

- **Example of data load and processing in GPU**

  ※ This scheme is the heart of the field of deep learning

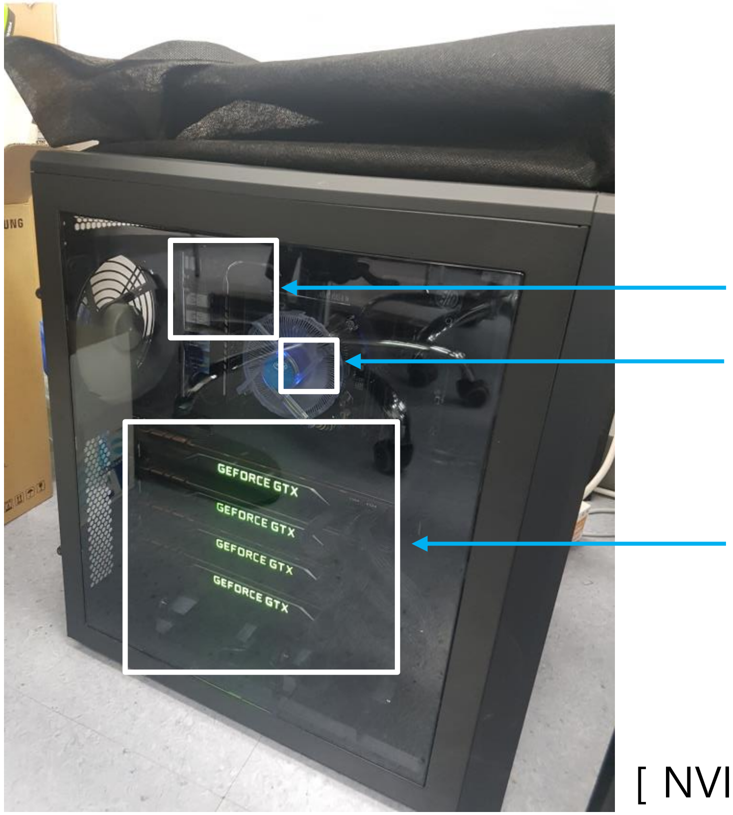


Introduction (7/8)

- **Communication between CPU and GPU**

  - **Design carefully training scheme**

→ This is because data transferring makes a significant bottleneck !



SSD (Data is here)

CPU
(Intel® Core i7 and Xeon® processor E5-2600, E5-1600 v3 v4)

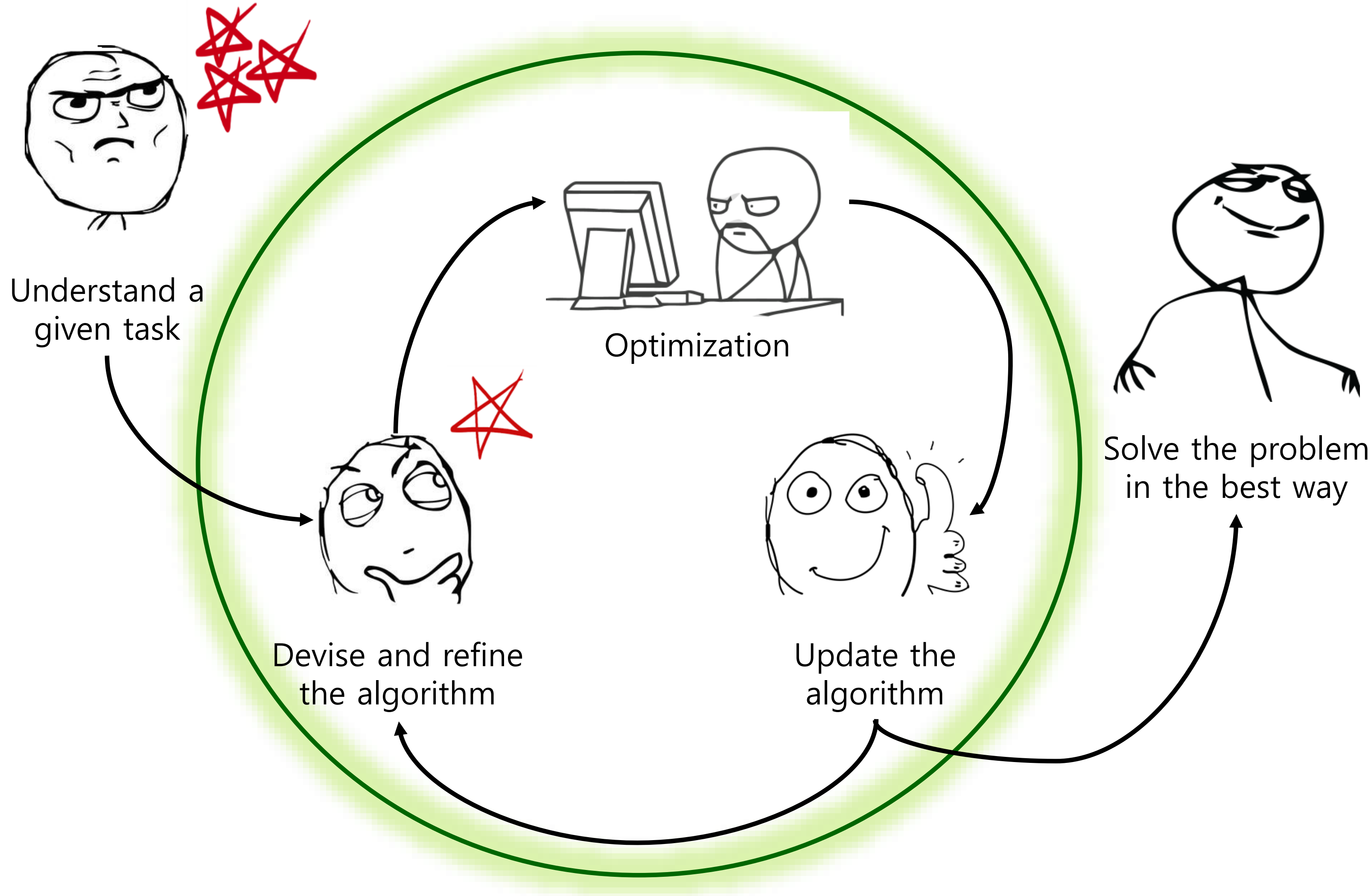
4 GPUs (Model is here)

Read all the data into RAM in GPUs

[NVIDIA SDX DevBox (4 Titan Xp) @ our lab.]

Introduction (8/8)

- **From data to algorithm :**



Understand a given task

Optimization

Devise and refine the algorithm

Update the algorithm

Solve the problem in the best way

Warming-up Example (1/3)

- **Data access (with memory usage)**
  - **Throw a dice 100 times and compute the frequency (빈도수)**

※ Use the rand() for generating the random number

Make your C codes !

Warming-up Example (2/3)

▪ Data access – cont’d

- **Implementation details**

1) By using the “scanf”, input your trial numbers (e.g., 100)
2) By using the “rand()”, you can get the random variable from 1 to 6
3) By using the “for” (or whilie) loop, you can get a number at each iteration
4) Store this number in your memory !
5) Allocate the memory space to store the frequency as follows :

e.g.) 100 trials → freq[1] = 30, freq[2] = 35, freq[3] = 9
freq[4] = 10, freq[5] = 9, freq[6] = 7

※ You need to use “malloc” or “calloc” for allocating the memory (freq)

Warming-up Example (3/3)

Appendix : time checker

- Compute the processing time as follows :

```
#include <Windows.h>

LARGE_INTEGER freq, start, stop;
double diff;

QueryPerformanceFrequency(&freq); // computer frequency
QueryPerformanceCounter(&start); // starting point

Algorithm

QueryPerformanceCounter(&stop); // stopping point
diff = (double)(stop.QuadPart - start.QuadPart)/ freq.QuadPart;
```

Measuring time

Summary

- Data revolution
  - Handling data (with C) is very important for real-world applications
- Enjoy thinking with your programming