Special Course on Computer Architectures

~GPU Programming Contest~

2017/6/16

Email: hunga@am.ics.keio.ac.jp

Graphics Processing Unit (GPU) and CUDA

Overview of GPU (Nvidia GM204)

Streaming Multiprocessor (SM)

CUDA core

Reference: Nvidia GeForce GTX 980 whitepaper

GPU

- Feature
  - Several hundred of cores
  - Several thousands of threads are executed concurrently
  - High memory bandwidth
  - Nvidia GPU can be controlled by CUDA

- Our used GPU is GeForce GTX 970 (Maxwell)

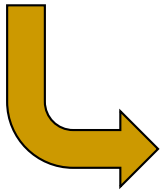
Table: Specification of GeForce GTX 970

CUDA cores	1664
Streaming multiprocessor	13 (Each of them has 128 cores)
Maximum memory bandwidth	224 GB/s

CUDA program

- CUDA: Compute Unified Device Architecture
  - Programming environment for Nvidia GPU

```
// C program
void
vector_add(int *a, int *b, int *c){
  int j;
  for(j = 0; j < N; j++)
    c[j] = a[j] + b[j];
}
```



Transform each iteration to each thread

```
// CUDA program
__global__ void
kernel(int *a, int *b, int *c){
  int j;
  j = blockDim.x +* blockIdx.x + threadIdx.x;
  c[j] = a[j] + b[j];
}
```

This function is a behavior of each thread.

```
void
vector_add(int *a, int *b, int *c){
  dim3 dg(N/BLOCKS, 1, 1);
  dim3 db(BLOCKS, 1, 1);
  kernel<<<dg, db>>>(a, b, c);
}
```

Specify # of thread blocks and threads per thread block

Flow of CUDA program

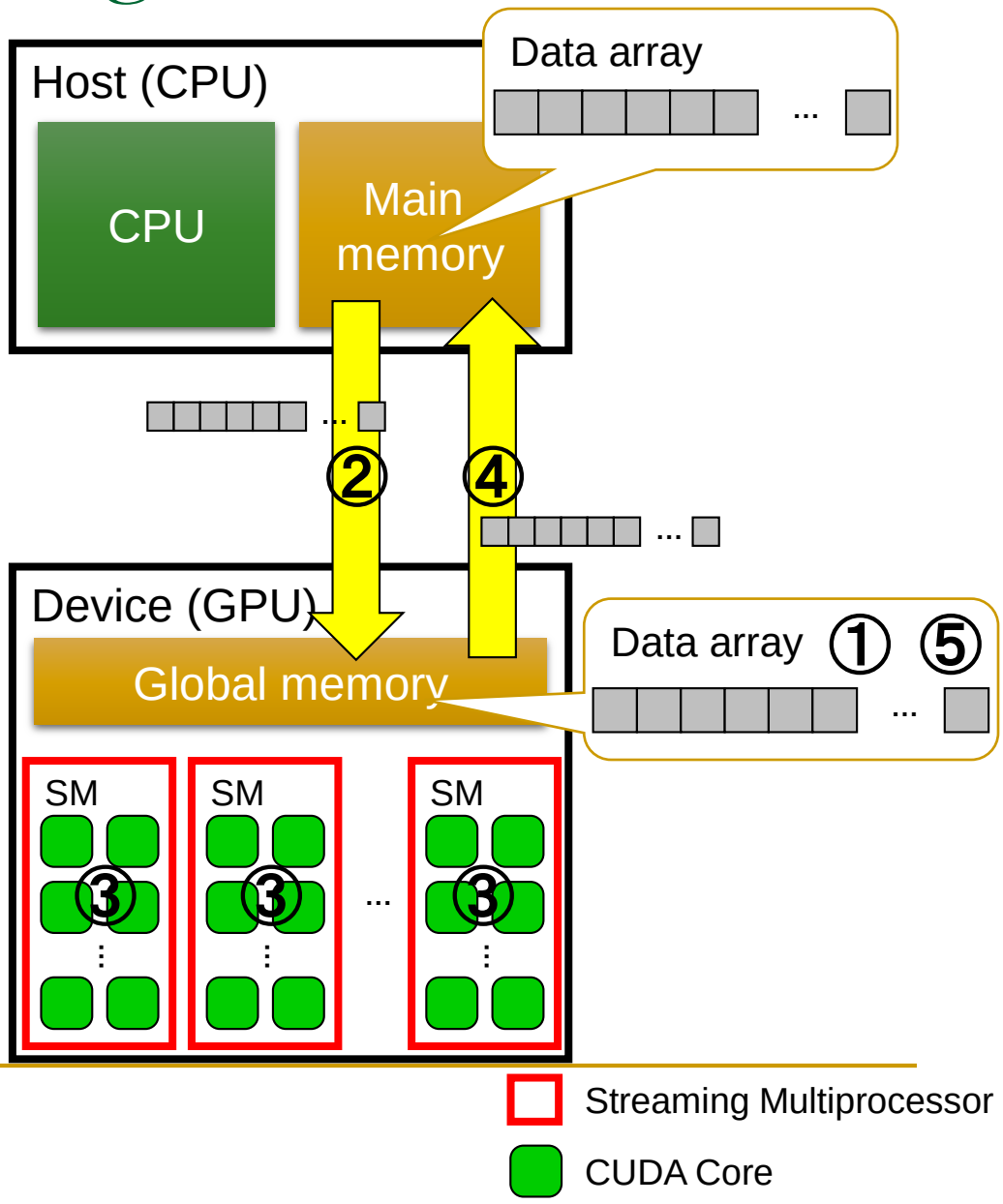
① Allocate GPU memory space
- `cudaMalloc()`

② Send input data from CPU to GPU
- `cudaMemcpy()`

③ Execute kernel on a GPU

④ Receive calculation results from GPU
- `cudaMemcpy()`

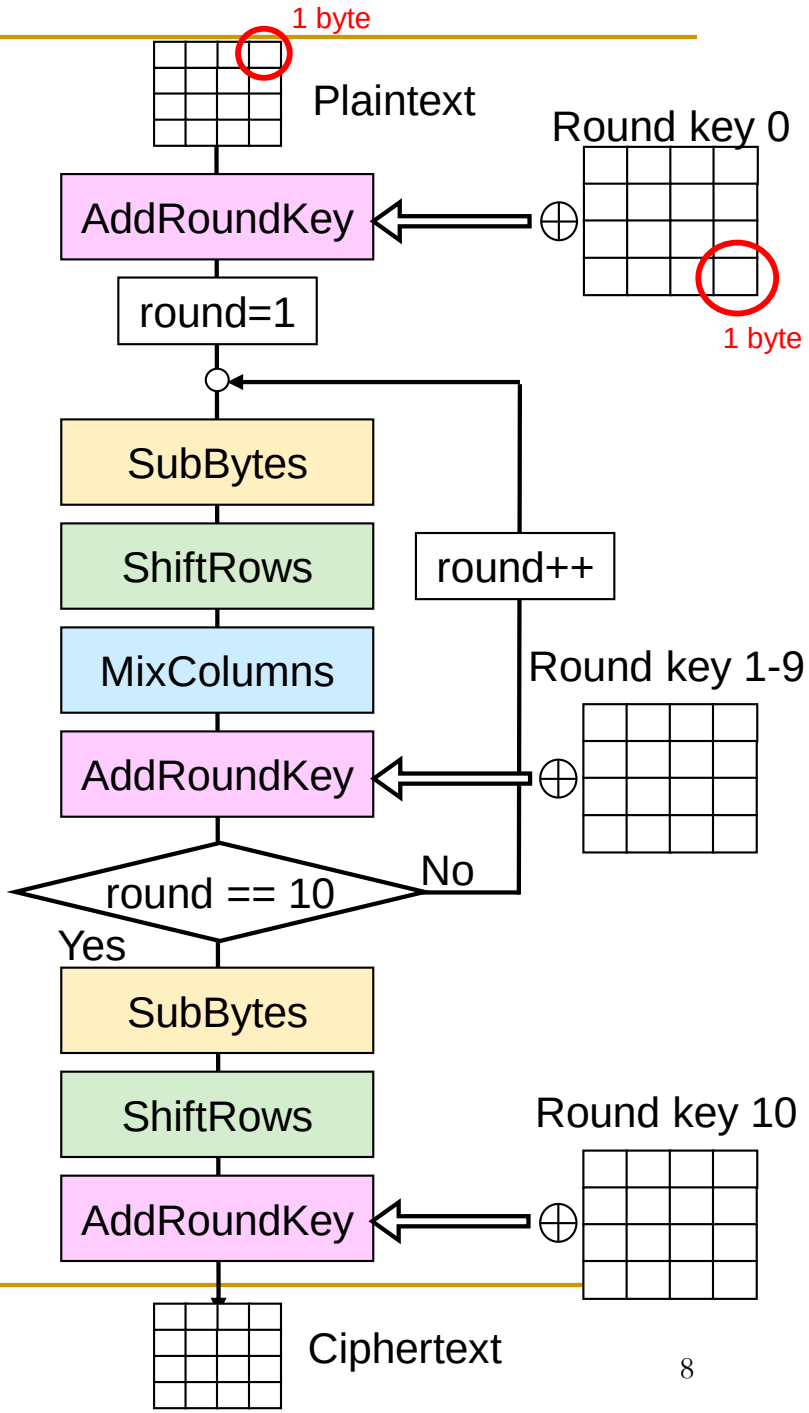
⑤ Free GPU memory space
- `cudaFree()`



Advanced Encryption Standard (AES)

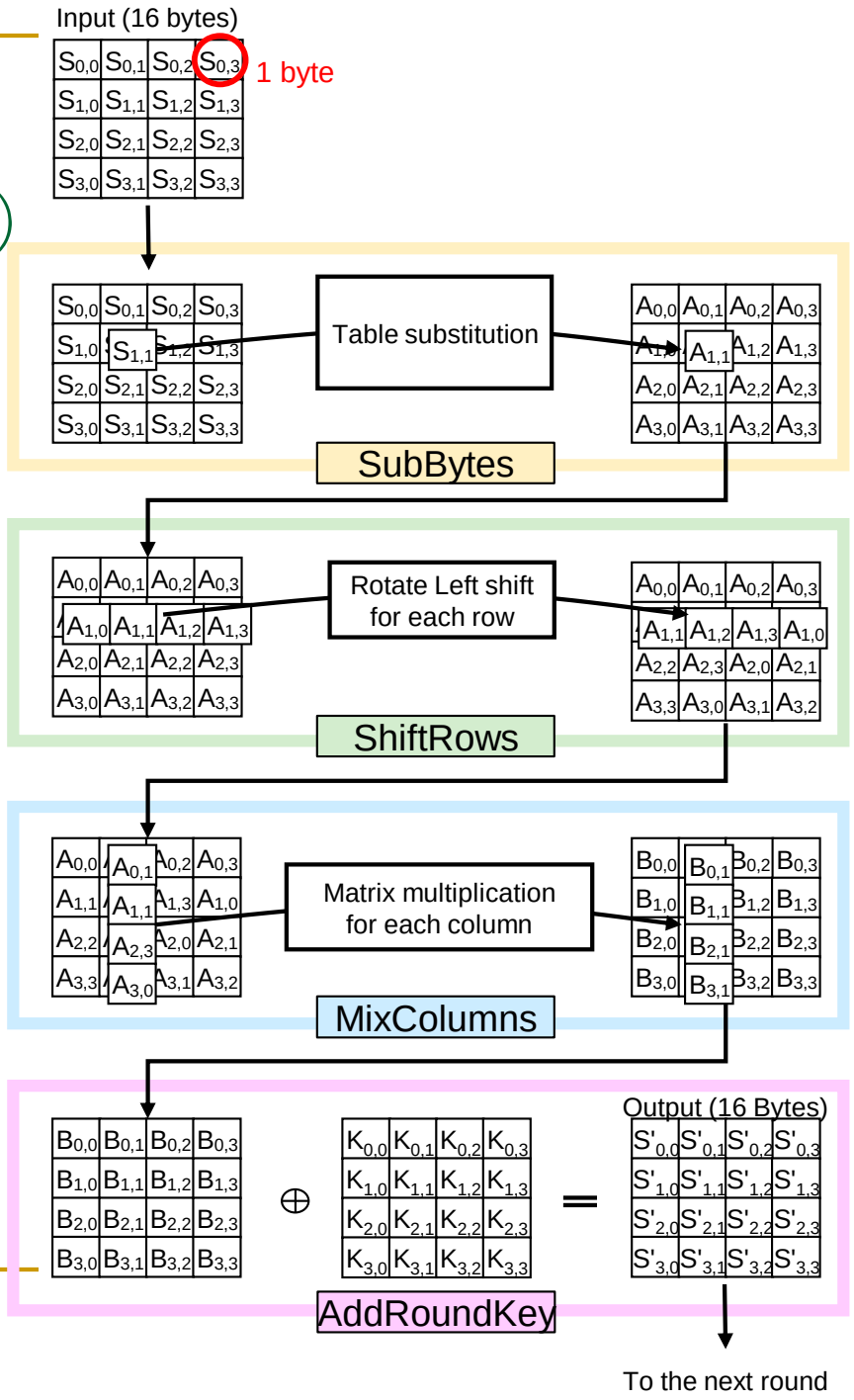
Target: AES (1)
(Advanced Encryption Standard)

- Mainstream of encryption algorithms
  - AES Specification
- One of the symmetric Block Ciphers
  - Plaintext
    - Text size: 128 bits (16 Bytes)
  - Secret key
    - Key size: 128 bits
    - Extends eleven 128-bit round keys through a key-schedule algorithm
  - 10 repetitive round processes



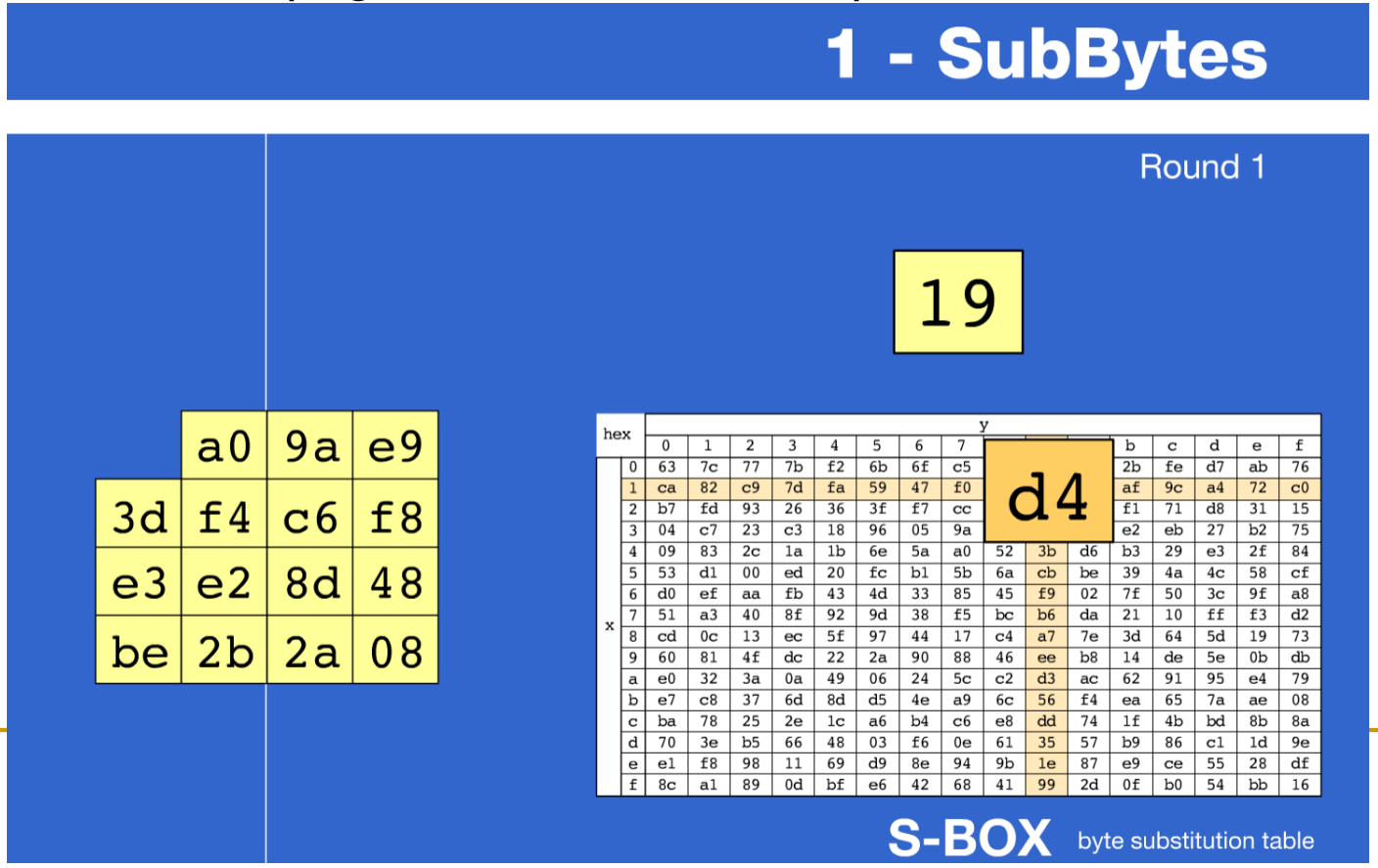
Target: AES (2)
(Advanced Encryption Standard)

- Each round process is the figure to the right
  - Fundamental calculation unit: 1 byte
- Each round: four types of transformation
  - SubBytes
    - Substitution transformation per byte
  - ShiftRows
    - Rotation of left shift per row
  - MixColumns
    - Multiplication and addition with constant matrix value per column
  - AddRoundKey
    - XOR with a round key per byte



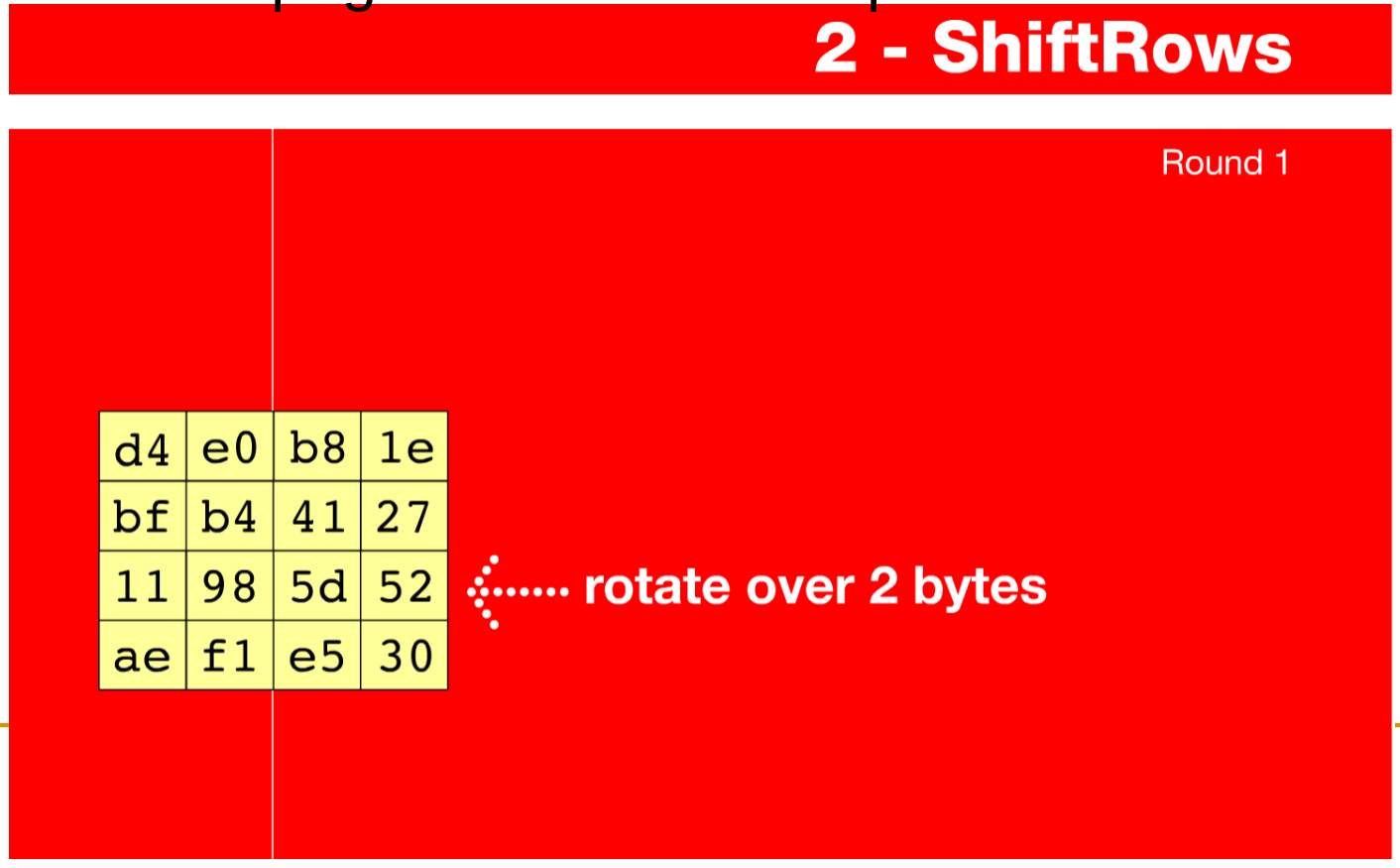
Stage 1 of each round process: SubBytes

- A non-linear byte substitution
  - Operates independently on each byte of the given 16 bytes input using a substitution table (S-box)
    - Refer the page 15-16 of the AES specification in detail



Stage 2 of each round process: ShiftRows

- A transformation that the bytes in the last three rows of the given 16 bytes are cyclically shifted over different numbers of bytes (offsets).
  - Refer the page 17 of the AES specification in detail



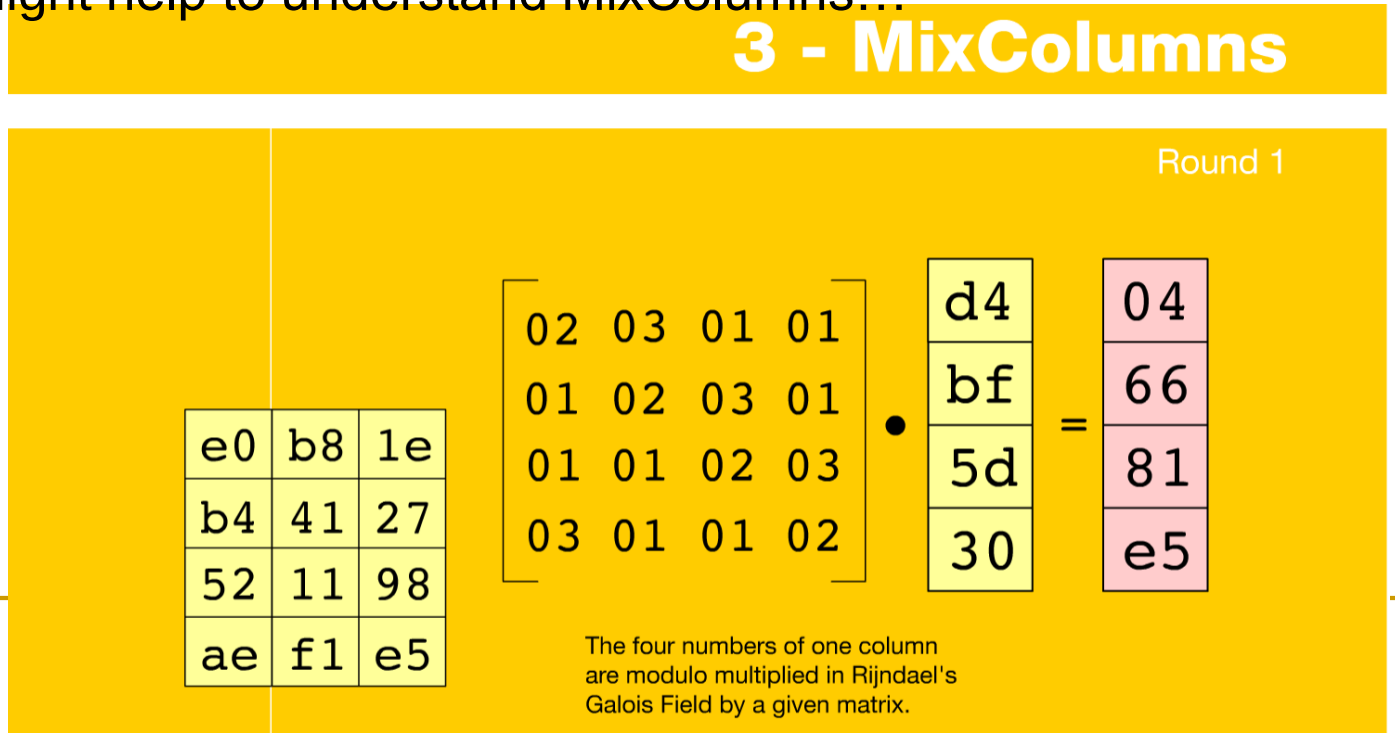
Stage 3 of each round process: MixColumns

- A transformation that operates on the given 16 bytes column-by-column, treating each column as a four-term polynomial over $GF(2^8)$ as follows:

$$a(x)=\{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

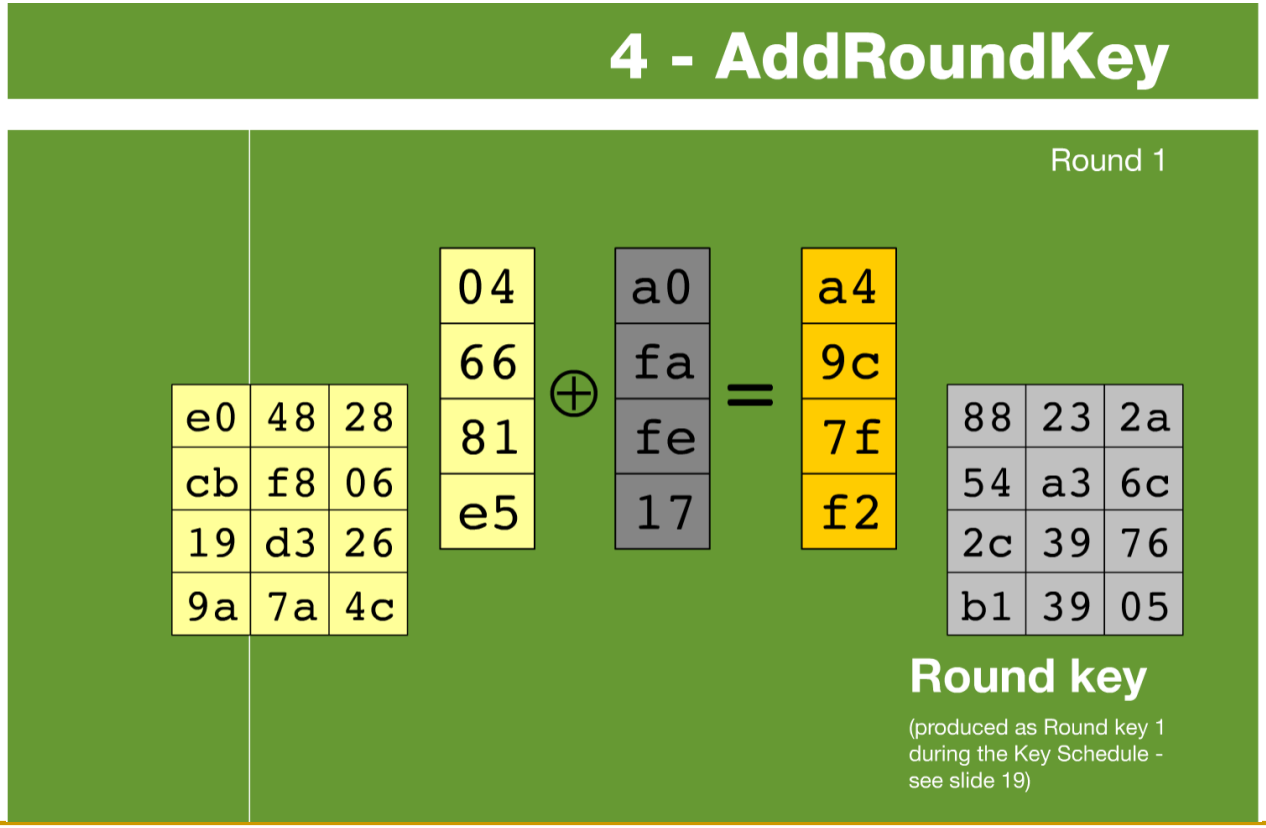
  - Refer the page 17-18 of the AES specification
  - http://www.angelfire.com/biz7/atleast/mix_columns.pdf

might help to understand MixColumns…



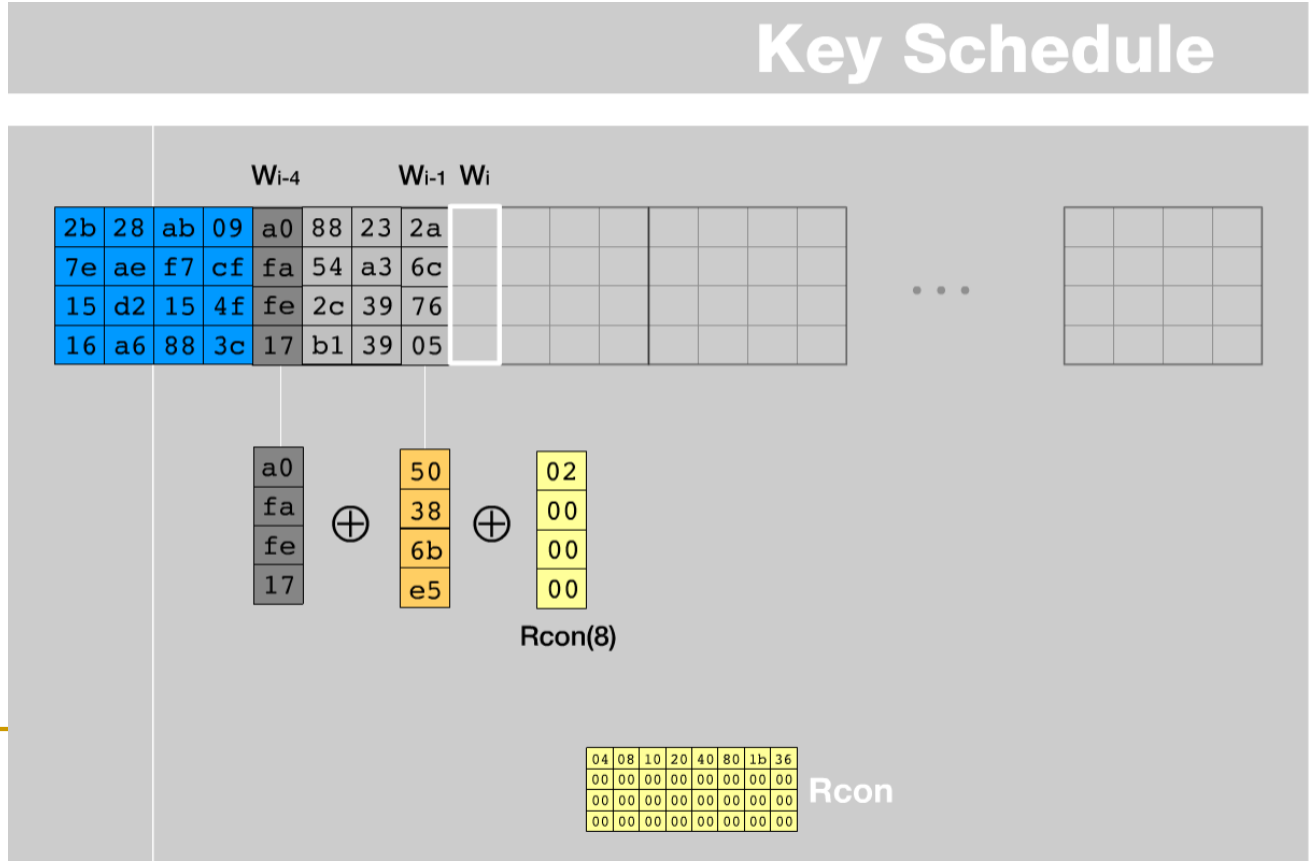
Stage 4 of each round process: AddRoundKey

- A transformation that a round key is added to the given 16 bytes by a simple bitwise XOR operation.



Notice: Key Schedule

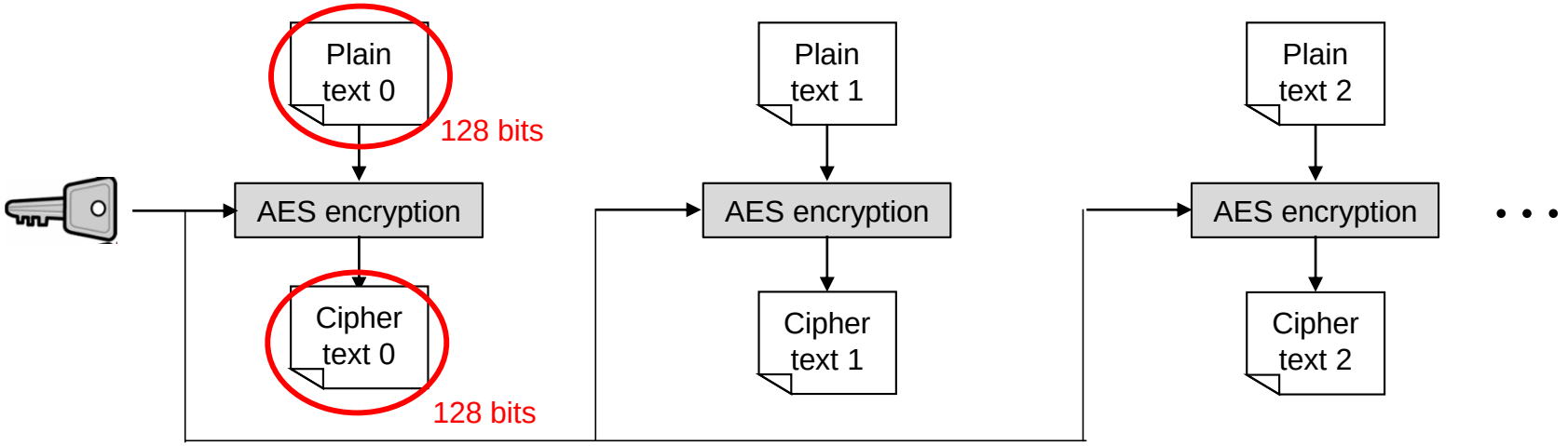
- Key schedule can be ignored to understand in this contest
  - Concentrate on optimization of randomization algorithm of AES



For more understanding AES encryption

- Please see the following flash movie
  - Rijndael cipher ~128-bit version encryption~
    - “Rijndael” is another name of AES

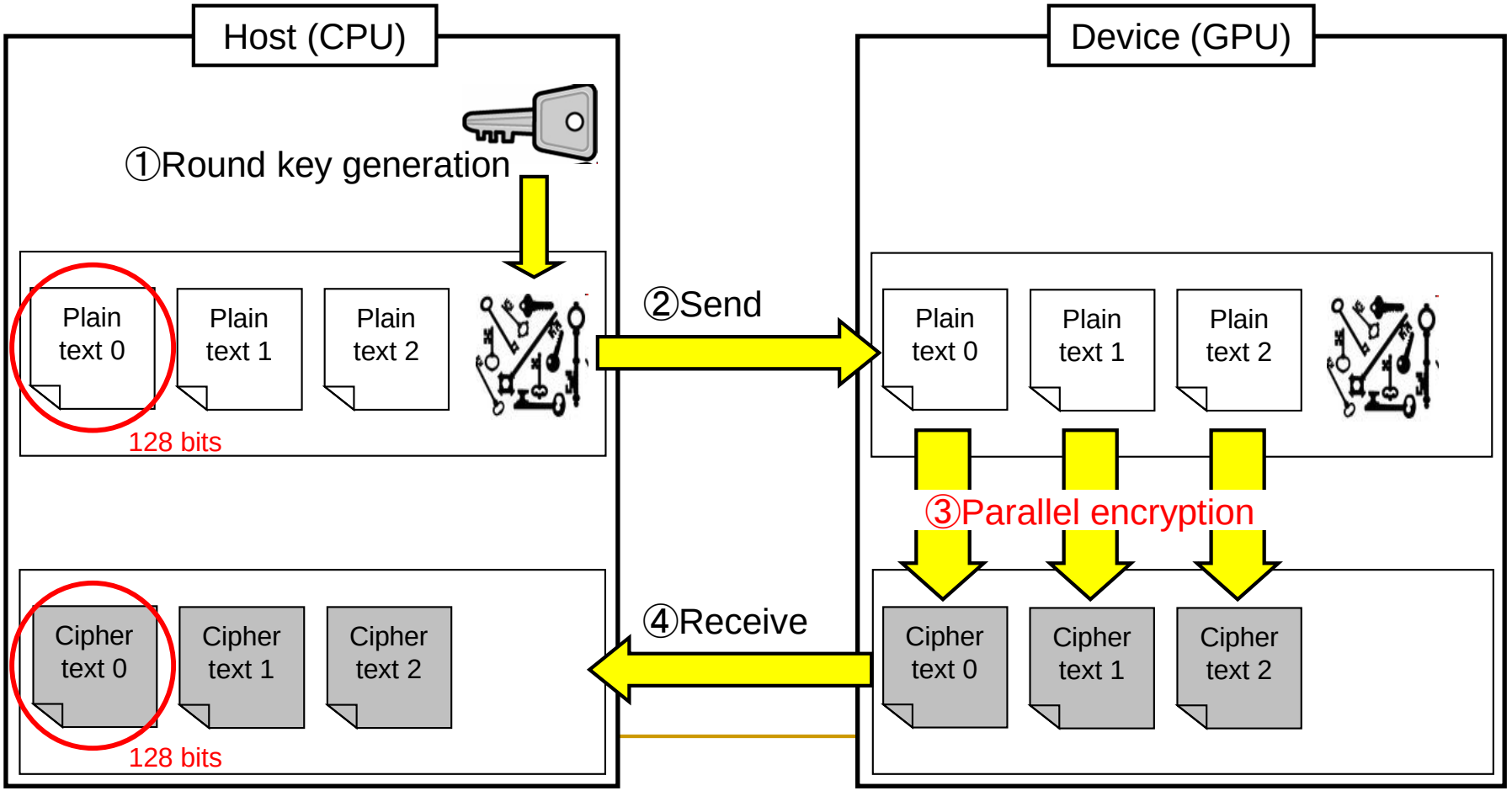
  http://poincare.matf.bg.ac.rs/~ezivkovm/nastava/rijndael_animacija.swf
- Several encryption modes available
  - However, this contest deals with the simplest mode: single key affects all the plaintext data, as described below



GPGPU Contest

AES calculation flow using GPU

- AES calculation flow using GPU
  - 128-bit plaintexts can be encrypted in parallel on GPU.



Contest program

- Example

```
$ ~/cuda/contest18/aes -> Sorry. Please
download from the website
$ make
$ ./aes

initialize..

size = 32
You can use printf function to
eliminate bugs in your kernel.
This thread ID is 0.
Verification finished... Overall
size: 32 bytes

Verification error detected. Pos
cpu 0x69, gpu 0x0
```

・At your submission, please set the FILESIZE parameter in calculation.h to “16*128*13*16*512”

Verify the results…
・Execute AES encryption on both CPU and GPU
・If the error is happen on your AES algorithm on GPU, a verification error will be displayed.

Elapsed time
・If your AES design is correct, then the elapsed time on GPU program will be obtained as follows.

```
Verification finished... Overall data size:
218103808 byte OK
Elapsed time on CPU: 0.015360 [msec]
Elapsed time on GPU: 0.773120 [msec]
Acceleration rate  : 0.019868 times faster than
the CPU
```

Contest

- Minimum requirements
  - Accelerating AES by using GPU
    - Modify only gpu_calc.cu basically
    - Set FILESIZE parameter in calcuration.h to 16*128*13*16*512
    - Not have to execute all parts on GPU
    - Initialization and verification supported by toolkit
- Advance
  - Optimization to achieve higher performance

How to start

- Login server
  - Address: comparc{01/02}.am.ics.keio.ac.jp
  - $ ssh user_name@server_address -XY
  - Your account has been available. If you have not received an account strip, please send mail to shimura@am.ics.keio.ac.jp
- There are useful sample codes in `cuda` directory.
  - Refer to the directories such as `~/cuda/cuda_samples` or `~/cuda/sample1`

Toolkit

- gpu_calc.cu
  - AES program for GPU
  - Not implemented (please modify this file)
- cpu_calc.cpp
  - AES program without GPU
  - Refer to modify gpu_aes.cu
- calculation.h
  - Parameters and prototypes for AES codes
- toolkit[.c/.h], timer[.c/.h]
  - Initialization, verification, timer, and so on.
- main.cpp
  - Call functions
- Makefile
  - To build this toolkit files

Toolkit (data sets)

- Plaintext are prepared as randomized data in main.cpp as follows

```
srand((unsigned)time(NULL));
for(int i = 0; i < FILESIZE; i++){
    plaintext[i] = rand() & 0xff;
}
```

Tips for writing faster code

- How to optimize program
  - Use Shared Memory and Constant memory
  - Coalesced memory access
  - Avoid conditional branch such as if statement, as much as possible
  - More sophisticated encryption algorithm would be better performance

- More information about CUDA architecture
  - CUDA Toolkit Documentation
  - CUDA C Best Practices Guide
  - Aoki et al, “はじめてのCUDAプログラミング (In Japanese)”, 工学社, 2009

Sample code

- Vector addition program
  - Kernel execution without time measurement
    - $ cd sample1
    - $ nvcc sample1.cu sample1_kernel.cu
    - $ ./a.out
  - Kernel execution with time measurement
    - In the same directory above
      - $ nvcc sample1_time.cu sample1_kernel_time.cu

- Points
  - Memory allocation on a GPU
    - `cudaMalloc(), cudaFree()`
  - Data transfer between CPU and GPU
    - `cudaMemcpy()`
  - Format of GPU kernel function

Caution!

- Download aes.tar from the website. Don’t use the one in your account.
- The number of thread must be exactly the same as FILESIZE/16.
- The maximum number of grid is 65535 for each direction.
- The maximum number of threads in a block is 512.

For a beginner as a programmer

- Try OpenMP programming contest instead.
- Optimize the code of dft with OpenMp parallel execution.
- Download and take a look at dft.tar.

make dft

- You can executable code without optimization.
- Insert the pragma in the openmp part in main.cpp.
- Try it by changing the number of threads.

Announcement

- If you have not an account, mail to: hunga4125@gmail.com
  - Don’t mistake the mail address.
  - Your name and which machine did you use (comparc01 or 02) should be included in the mail.
- *Deadline: 8/4 (Mon) 24:00*
  - Your name should be included in the mail.
- Make the directory “contest” in your home directory of comparc01 or comparc02, and copy the follows.
  - Source code and simple report (Text, PDF, etc)
- Please check the website. Additional information will be on it.
- If you have any question about the contest, please contact to shimura@am.ics.keio.ac.jp