ChunDoong & Colab

Represented by 이준열

Hanyang University
Deep learning & Big Data System Lab

Contents

Clusters for running CUDA

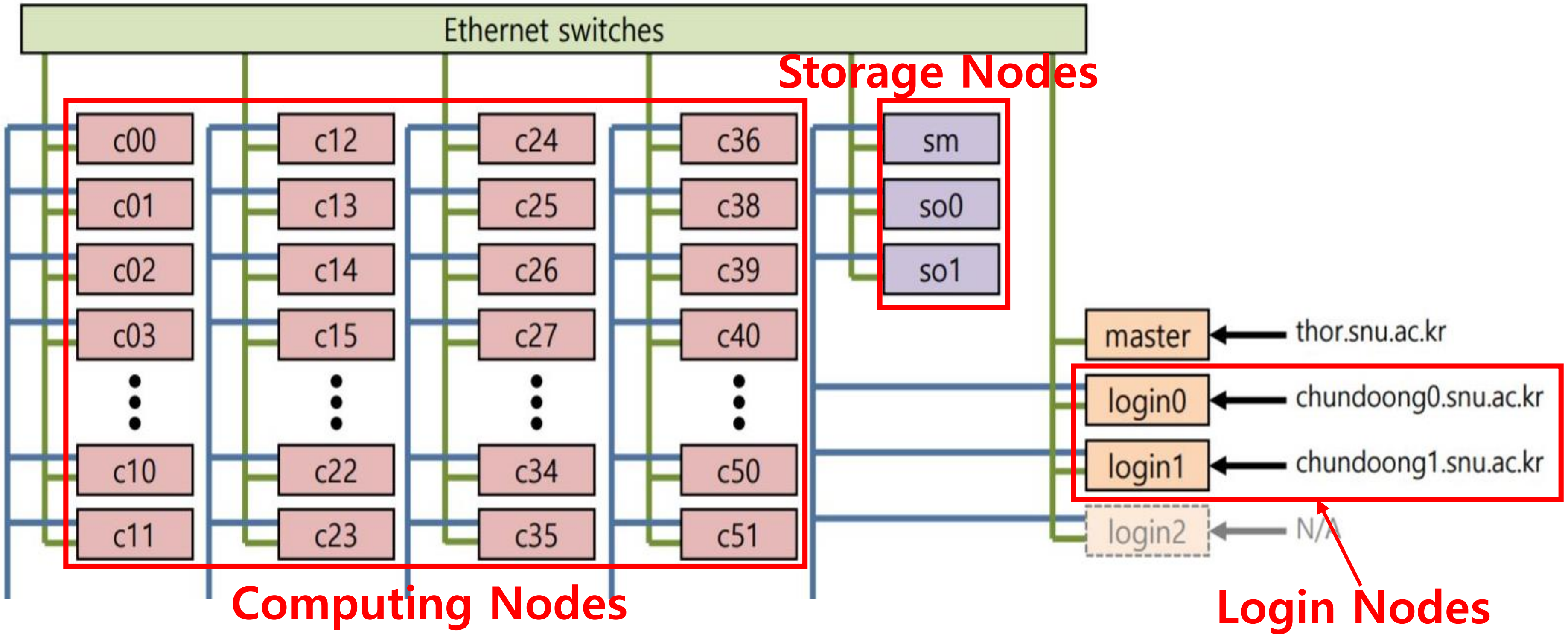
1. ChunDoong(천둥)

2. Google Colab

3. Your machine

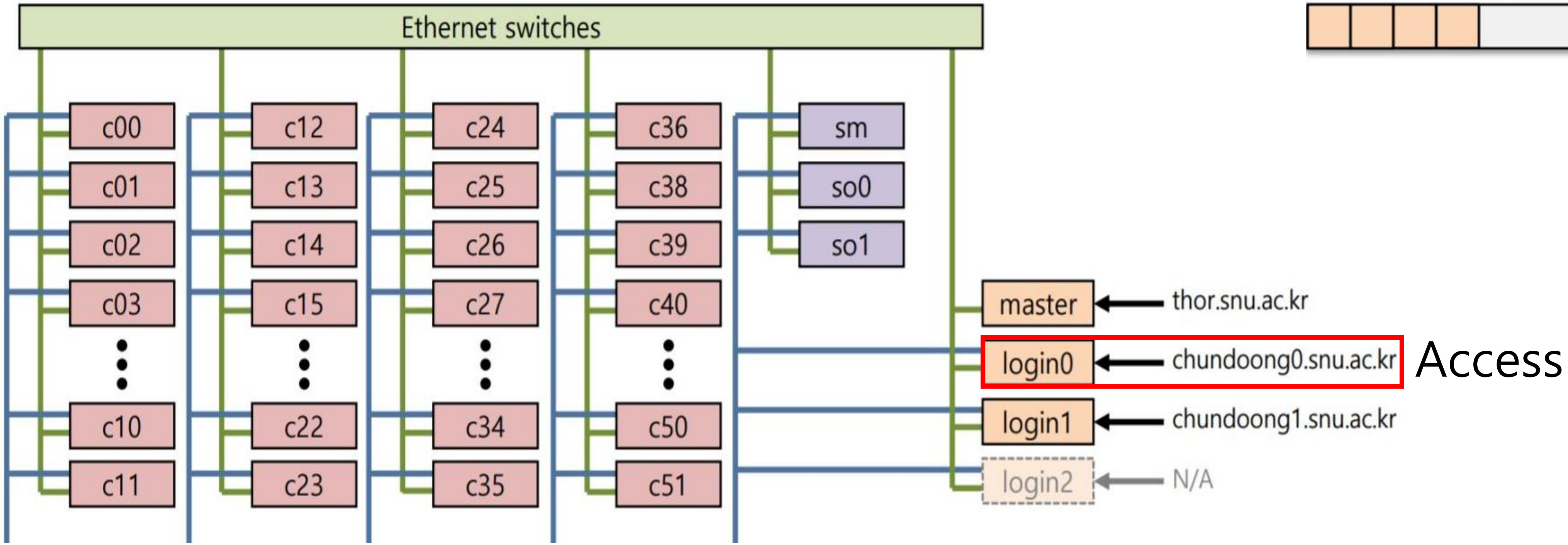
ChunDoong(천둥)

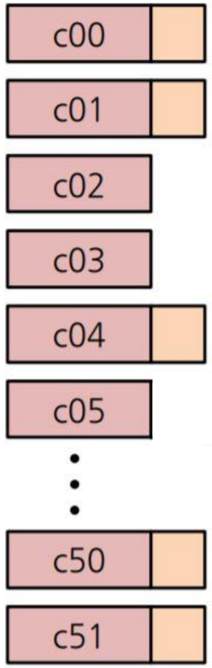
- ChunDoong is super computer at Seoul National University
- ChunDoong consists of 52 computing nodes, 3 storage nodes, and 2 login nodes.



ChunDoong(천둥) - Usage

- Login

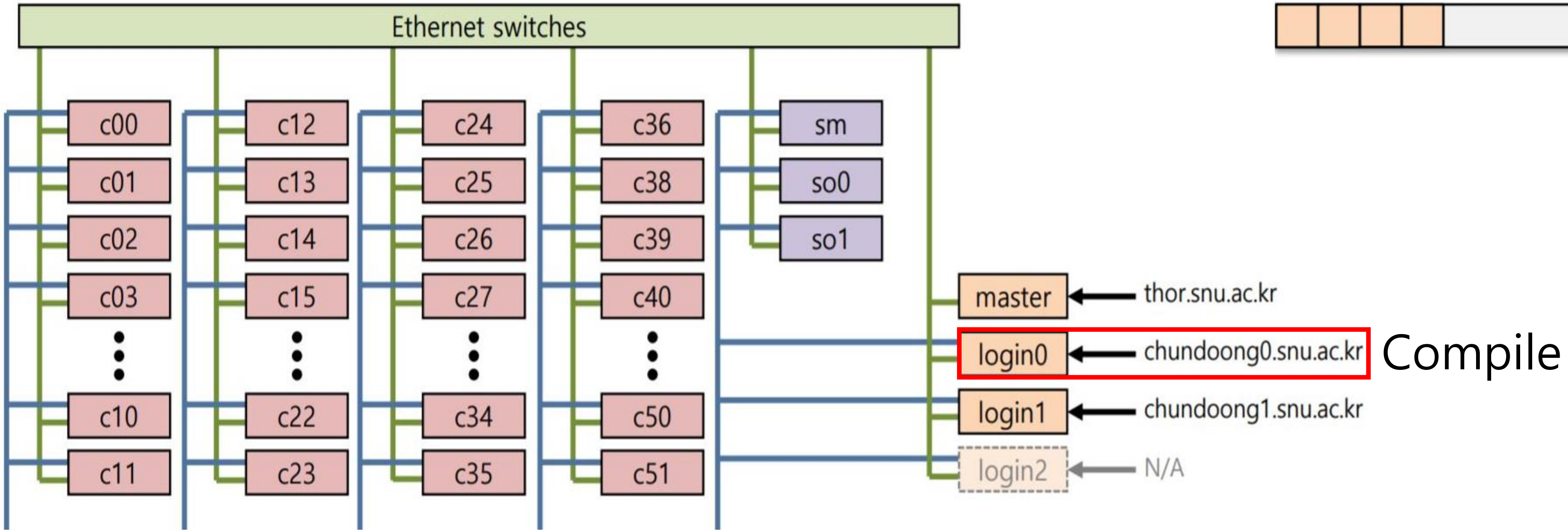


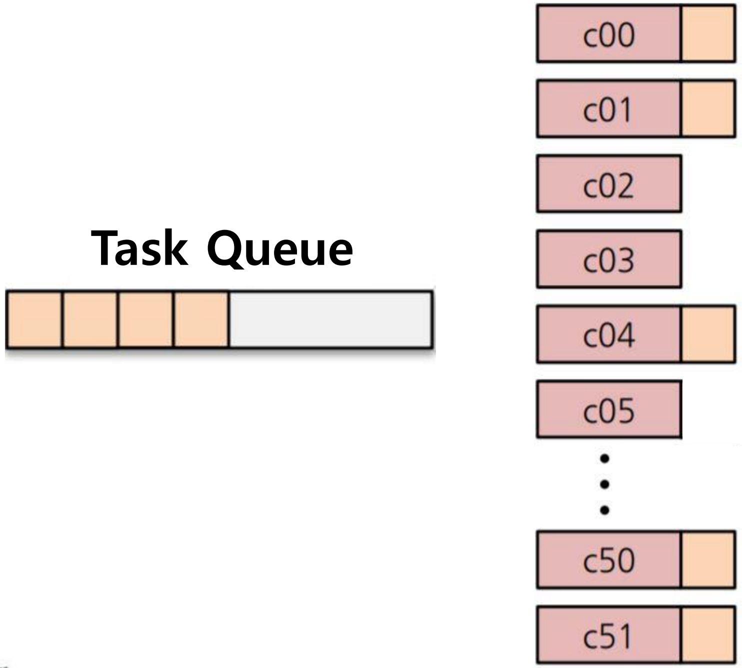



```
[~]$ ssh hyu99@chundoong0.sun.ac.kr
```

ChunDoong(천둥) - Usage

- Compile CUDA code

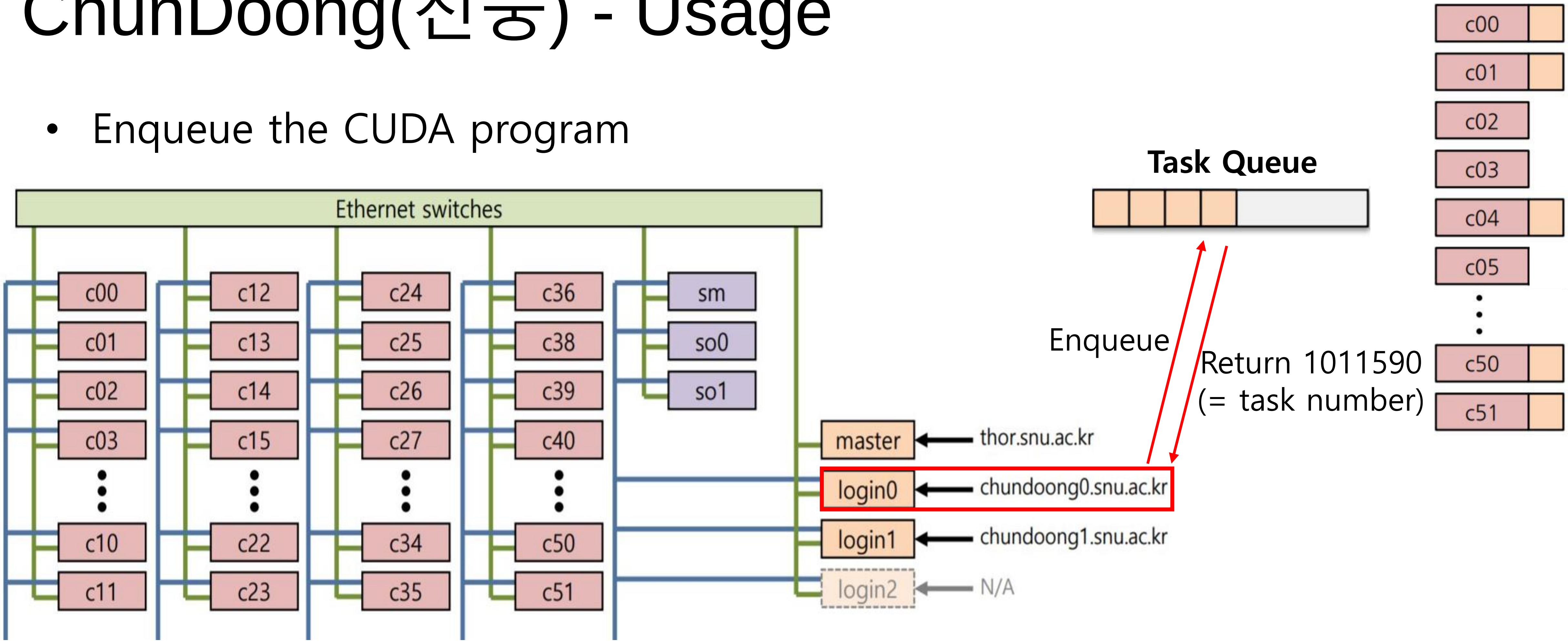





```
[hyu99@login0 ~]$ nvcc vecAdd.cu -o vecAdd
```

ChunDoong(천둥) - Usage

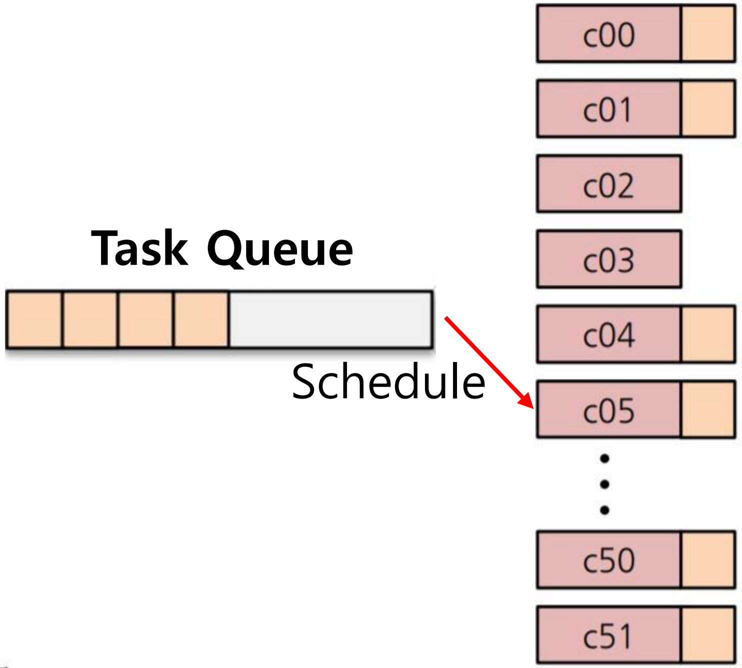
- Enqueue the CUDA program

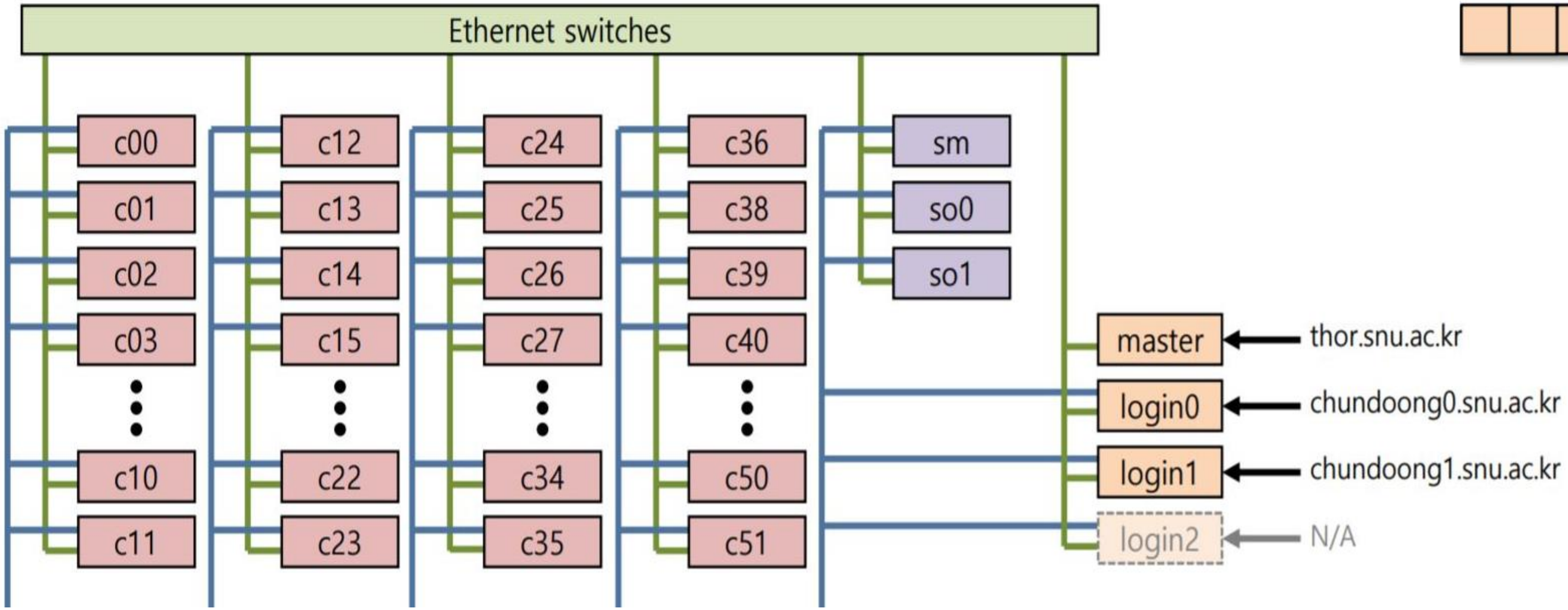



```
[hyu99@login0 ~]$ thorq --add --mode single --device gpu/1080 vecAdd
```

ChunDoong(천둥) - Usage

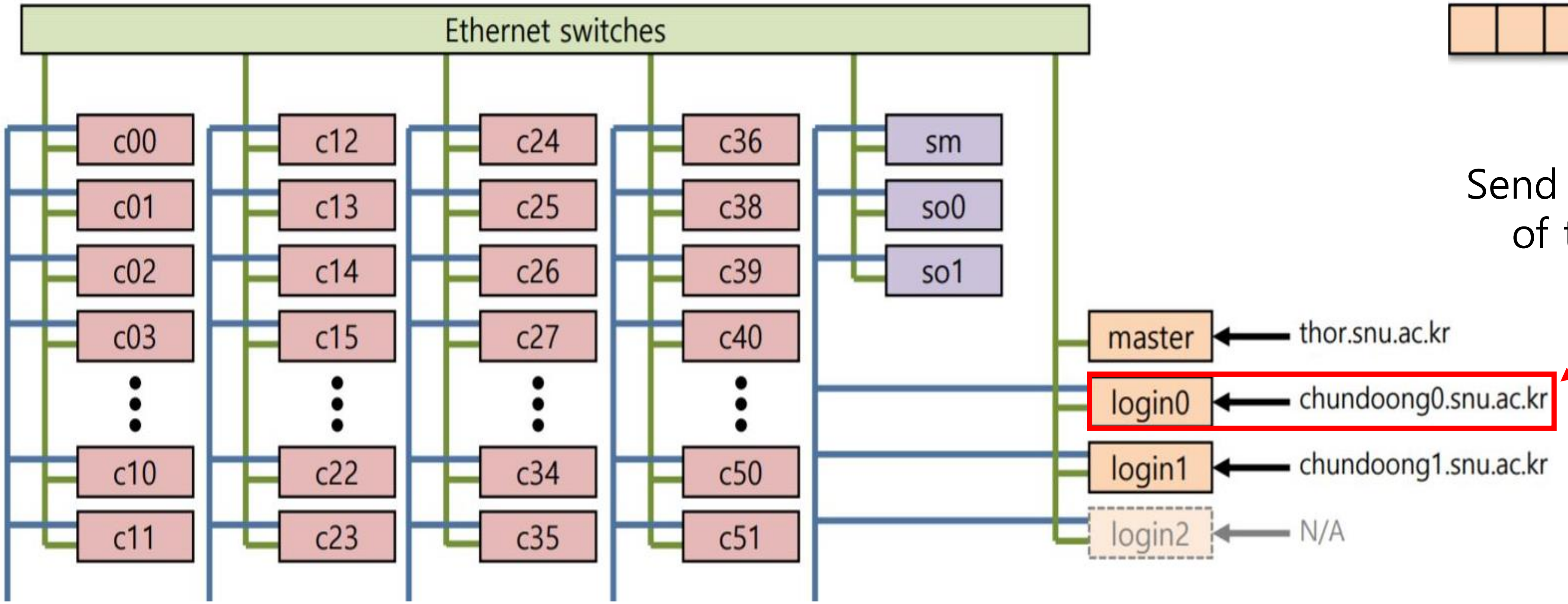
- Wait for scheduling and running

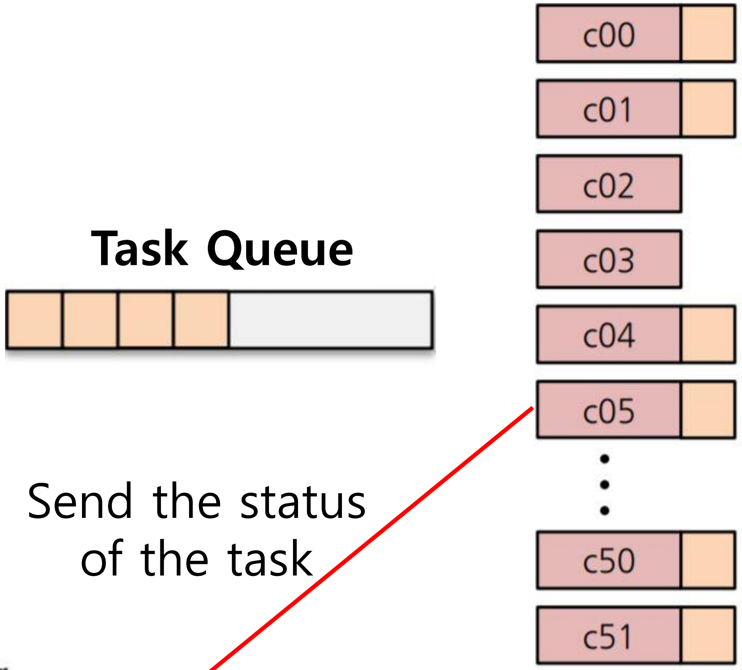




ChunDoong(천둥) - Usage

- Wait for scheduling and running






```
[hyu99@login0 ~]$ thorq --stat 1011590
```

ChunDoong(천둥) - Usage

- Wait for scheduling and running

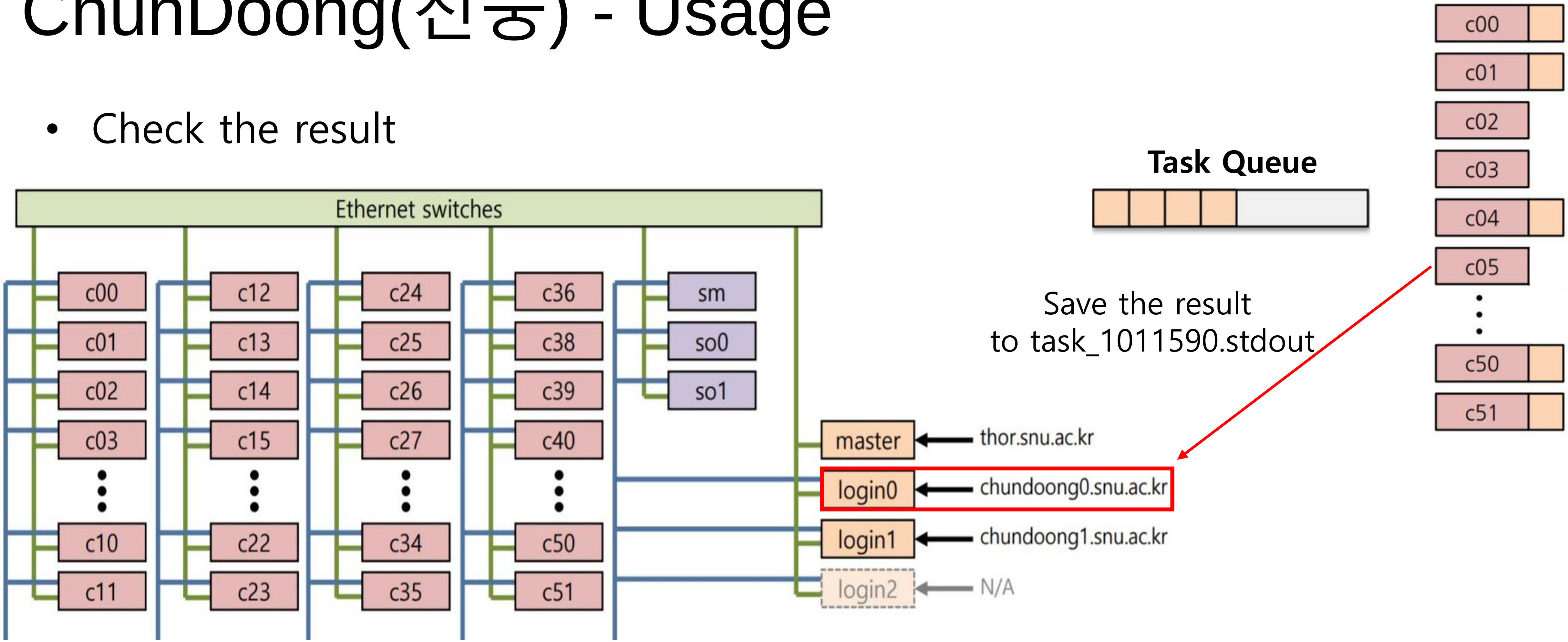
```
[hyu42@login0 ~]$ thorq --stat 1011590
===============================================================
ID            : 1011590
Name          : task_1011590
Status        : Running
Enqueued      : 2019-05-27 11:00:30
Executed      : 2019-05-27 11:00:31
Finished      :
Assigned nodes: c36
# of nodes    : 1
# of slots    : 1
Timeout       : 259200 s
Device        : CPU & GPU (NVIDIA GeForce GTX 1080)
Command string: thorq --add --mode single --device gpu
===============================================================
```

Task Number

```
[hyu42@login0 ~]$ thorq --stat 1011590
===============================================================
ID            : 1011590
Name          : task_1011590
Status        : Finished (success)
Enqueued      : 2019-05-27 11:00:30
Executed      : 2019-05-27 11:00:31
Finished      : 2019-05-27 11:01:09
Elapsed time  : 38.421191 s
Assigned nodes: c36
# of nodes    : 1
# of slots    : 1
Timeout       : 259200 s
Device        : CPU & GPU (NVIDIA GeForce GTX 1080)
Command string: thorq --add --mode single --device gpu/1080 vecAdd
===============================================================
```

ChunDoong(천둥) - Usage

- Check the result




```
[hyu99@login0 ~]$ cat task_1011590.stdout
```

ChunDoong(천둥) - Usage

- ‘task_1011590.stdout’ will be saved at where ‘thorq --add’ was executed

```
[hyu42@login0 ~]$ ls
vecAdd   vecadd.cu
```

```
[hyu42@login0 ~]$ ls
task_1011590.stderr  task_1011590.stdout  vecAdd  vecadd.cu
```

ChunDoong – Login(SSH)

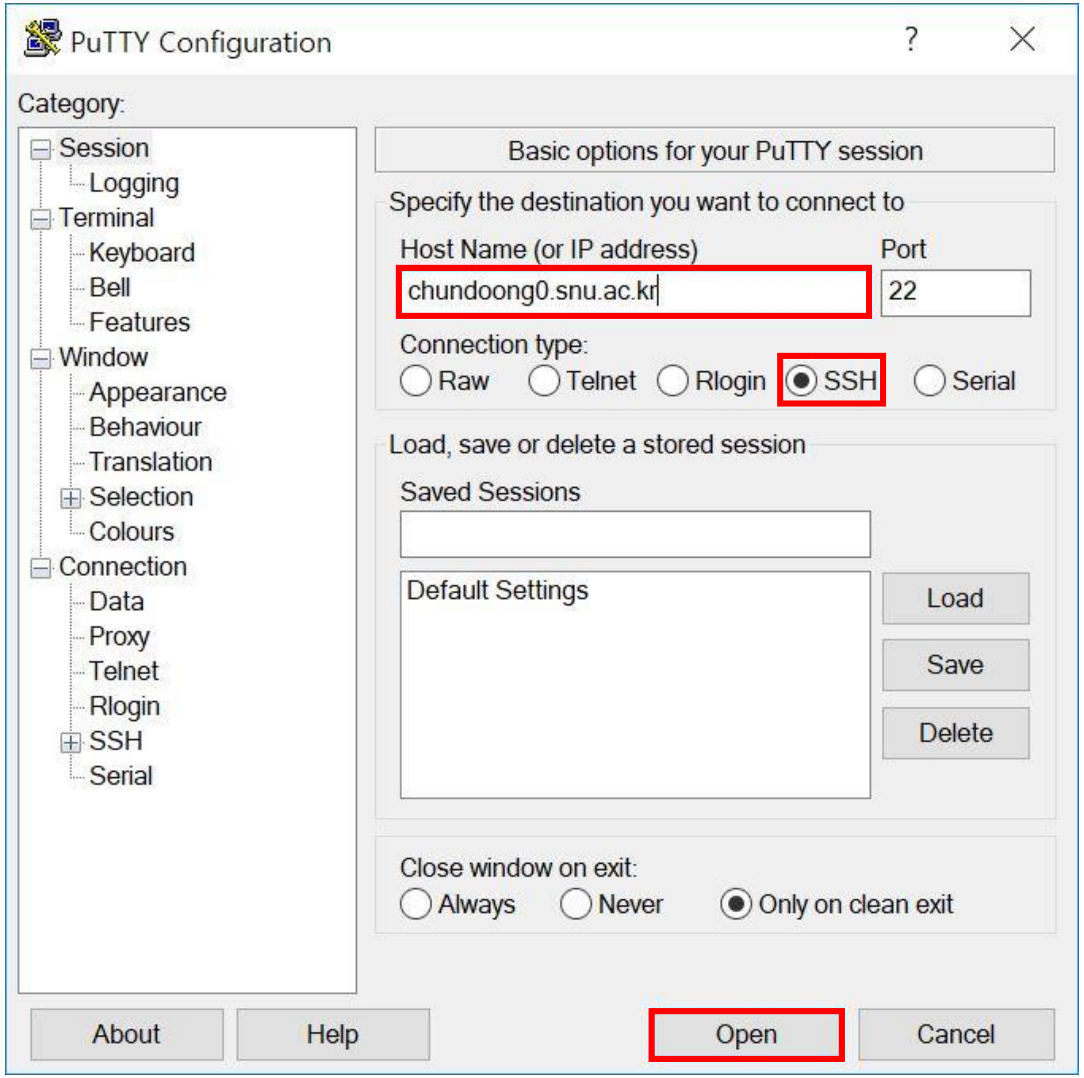
1. Open the Terminal or CMD(Windows 10 supports OpenSSH)
2. Type ‘ssh <ID>@chundoong0.snu.ac.kr’ or ‘ssh <ID>@chundoong1.snu.ac.kr’
3. Type ‘yes’ when an authentication message is appeared and enter the password

```
C:₩Users₩ints>ssh hyu01@chundoong1.snu.ac.kr
The authenticity of host 'chundoong1.snu.ac.kr (147.46.219.242)' can't be established.
ECDSA key fingerprint is SHA256:TKUfAA8yW4FgyWg4egxArMMzLVfTINyhvDCVOZFwJHU.
Are you sure you want to continue connecting (yes/no)? yes
hyu01@chundoong1.snu.ac.kr's password:
Last login: Fri Jun  8 01:57:57 2018 from 124.111.230.138
[hyu01@login1 ~]$
```

ChunDoong – Login(PUTTY)

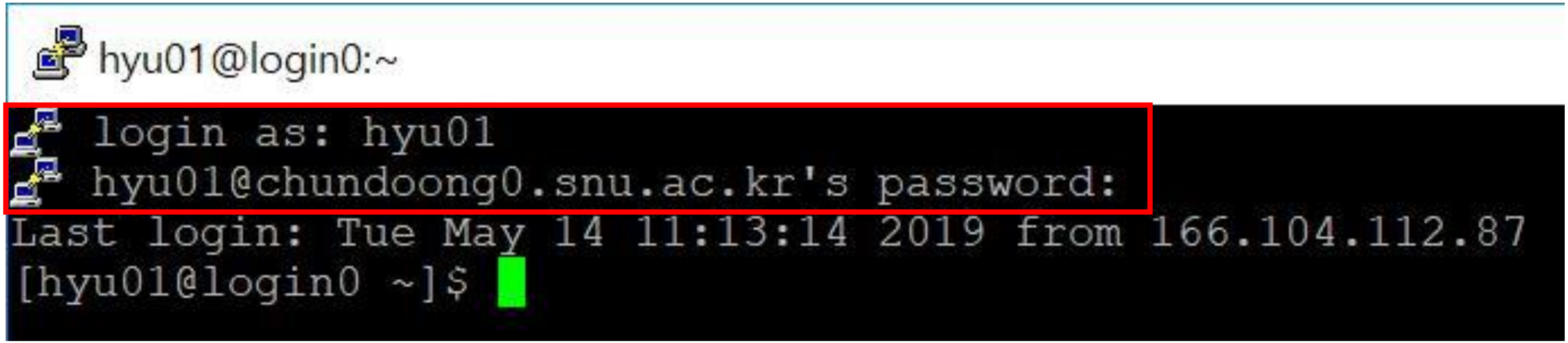
1. Open PUTTY

2. Type 'chundoong0.snu.ac.kr' or 'chundoong1.snu.ac.kr' in 'Host Name (or IP Address)' and Click 'Open' button



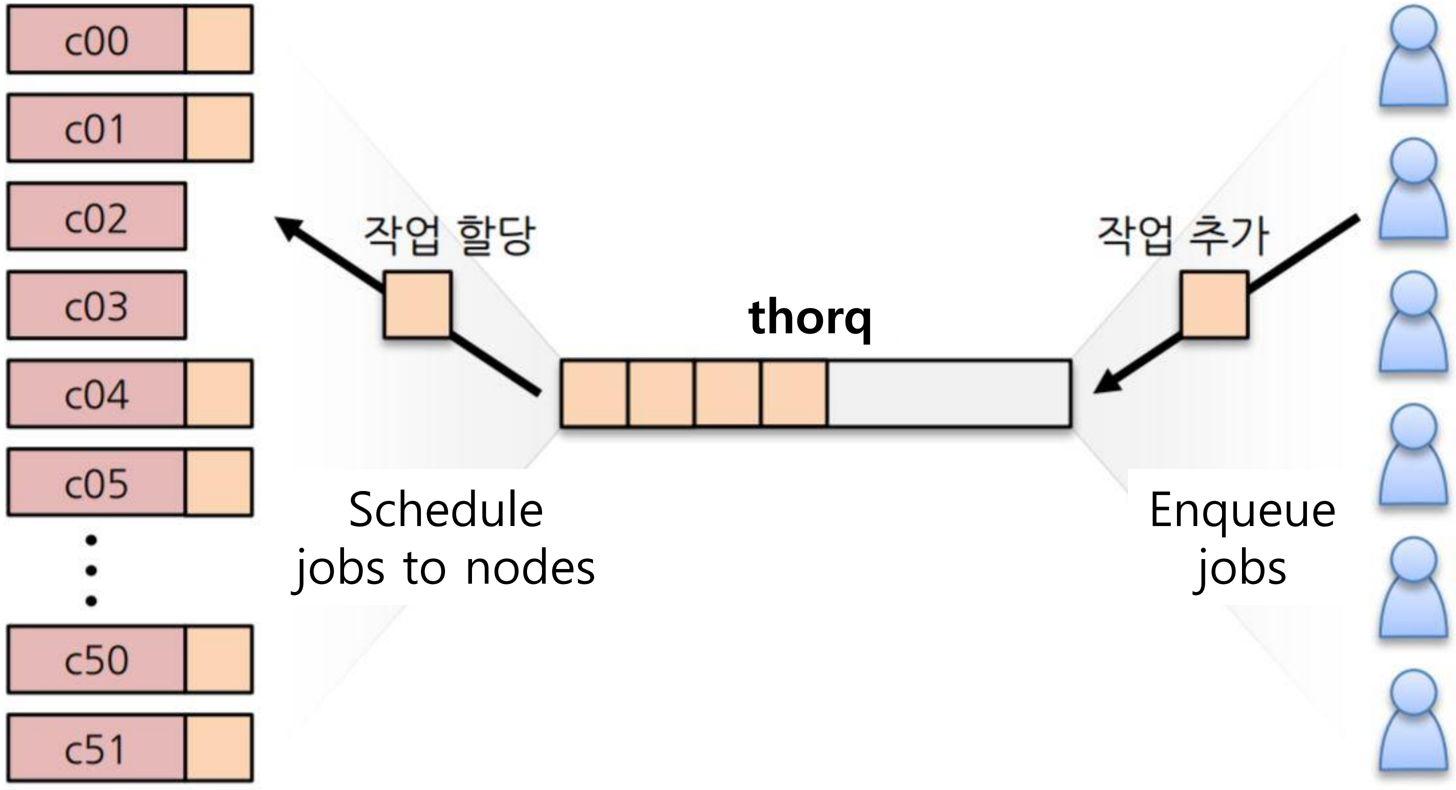
ChunDoong – Login(PUTTY)

3. Enter your ID and Password



ChunDoong – Enqueuing

- ChunDoong uses ‘**thorq**’ for task schduling



ChunDoong – Enqueuing

- Using 'thorq --add', you can enqueue the CUDA program.

thorq --add --mode single --device gpu/1080 [options] exec_file [arg1, arg2, ...]

After the program enqueued, the task number will be displayed

- [options]
  - --timeout [# seconds] : Kill if the program run after a certain time. (Default : 3 days)
  - --name [job_name] : set the name of task to [job_name]

- For example, if the name of the program is 'vecAdd'

thorq --add --mode single --device gpu/1080 vecAdd

ChunDoong – Checking task status

- **thorq --stat <task_num>** shows the task by its task number

```
[user_id@login0 ~]$ thorq --stat 229
==================================================================
ID              : 229
Name            : thorq_test
Status          : Finished (success)
Enqueued        : 2013-03-22 17:25:25
Executed        : 2013-03-22 17:25:35
Finished        : 2013-03-22 17:25:50
Executed time   : 15.090000 s
Assigned nodes: c16, c17, c18, c19, c20, c21, c22, c23, c24, c25
Device          : CPU & GPU (AMD Radeon HD 7970)
# of nodes      : 10
Command string: thorq --add --mode mpi --nodes 10 --device
   gpu/7970
--timeout 100 --name thorq_test bin/a.out 10 20 30
==================================================================
```

ChunDoong – Other commands

- **thorq --kill <task_num>** kills the task by its task number
- **thorq --kill-all** kills all tasks are Enqueued or Running status

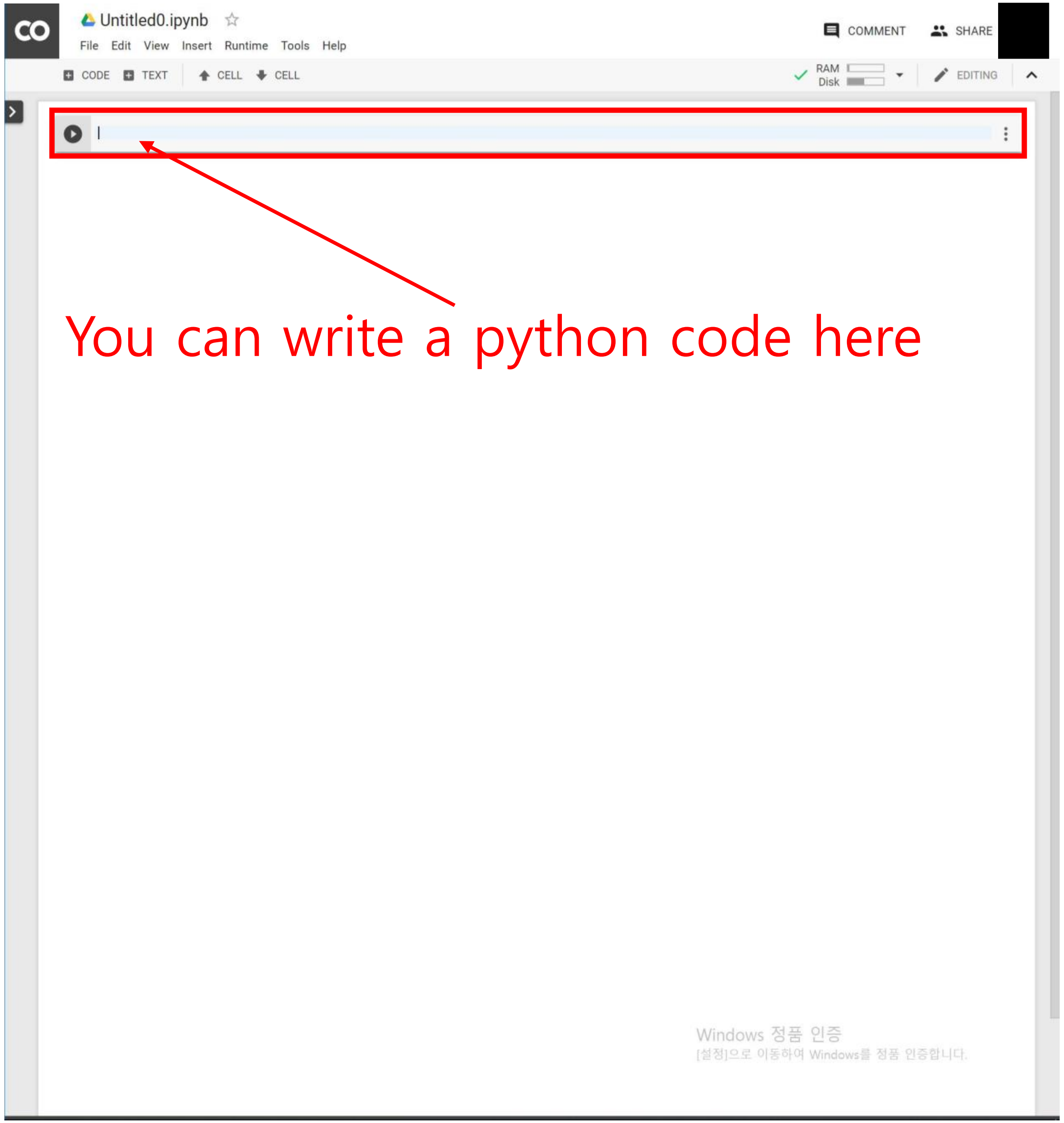
ChunDoong – Quota

- All accounts have 300,000 quotas, worth 27 hours GPU computation
- A running CUDA program consumes 3 quotas per second (only when running)
- If you want to see remaining quotas, Type **thorq --quota**
- More Informations (Written in Korean) :
  http://chundoong.snu.ac.kr/files/chundoong_user_manual_1.5.pdf

ChunDoong – DEMO

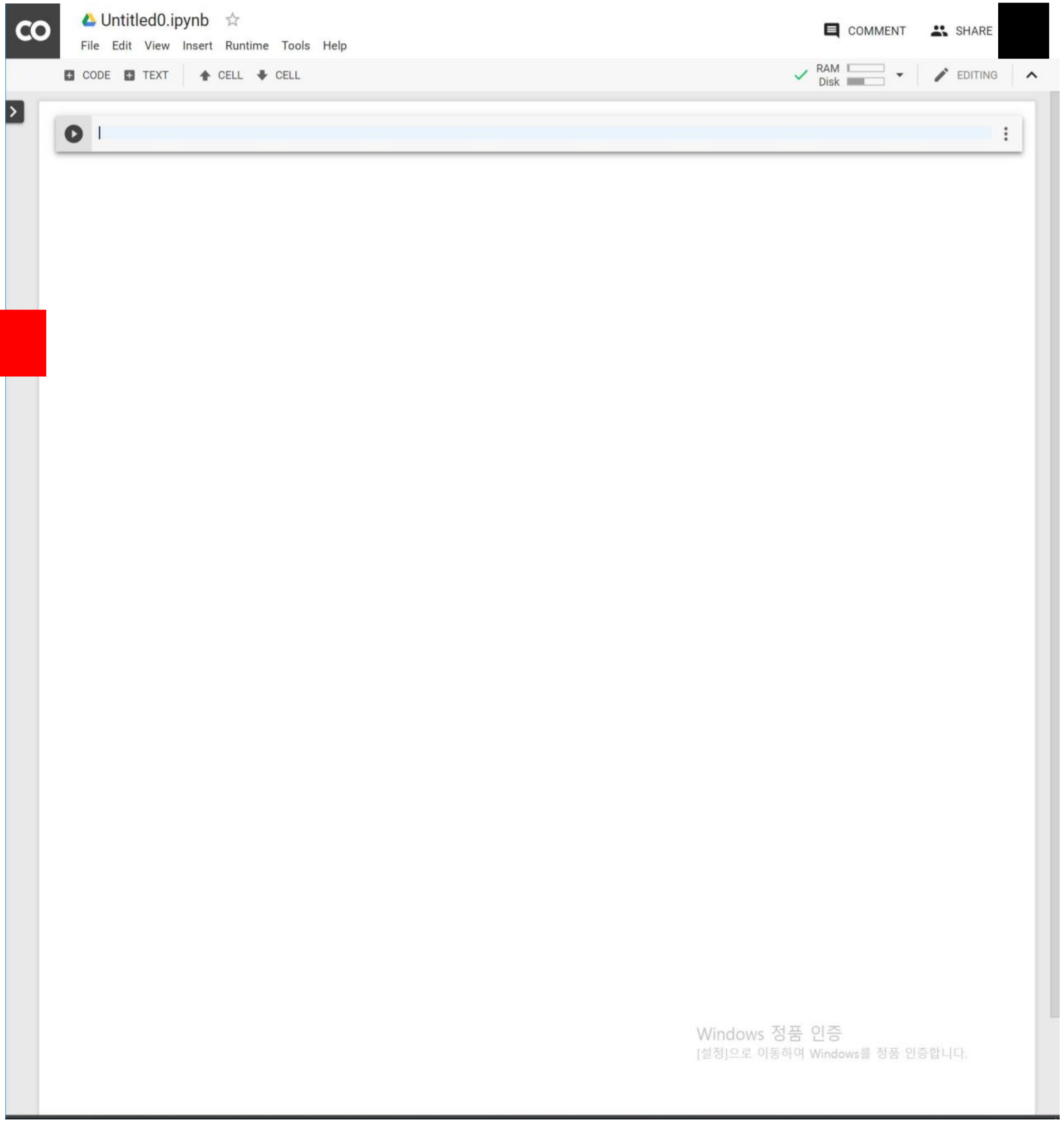
Google Colab

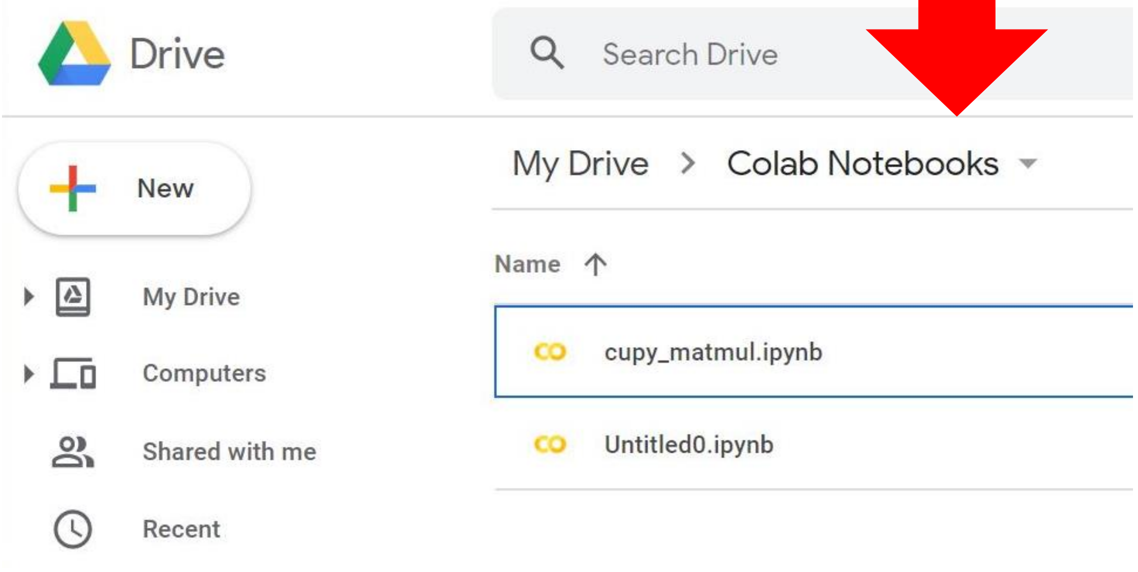
- Python interactive web interface for Google GPU cloud



Google Colab

- All Google Colab files is saved in google drive

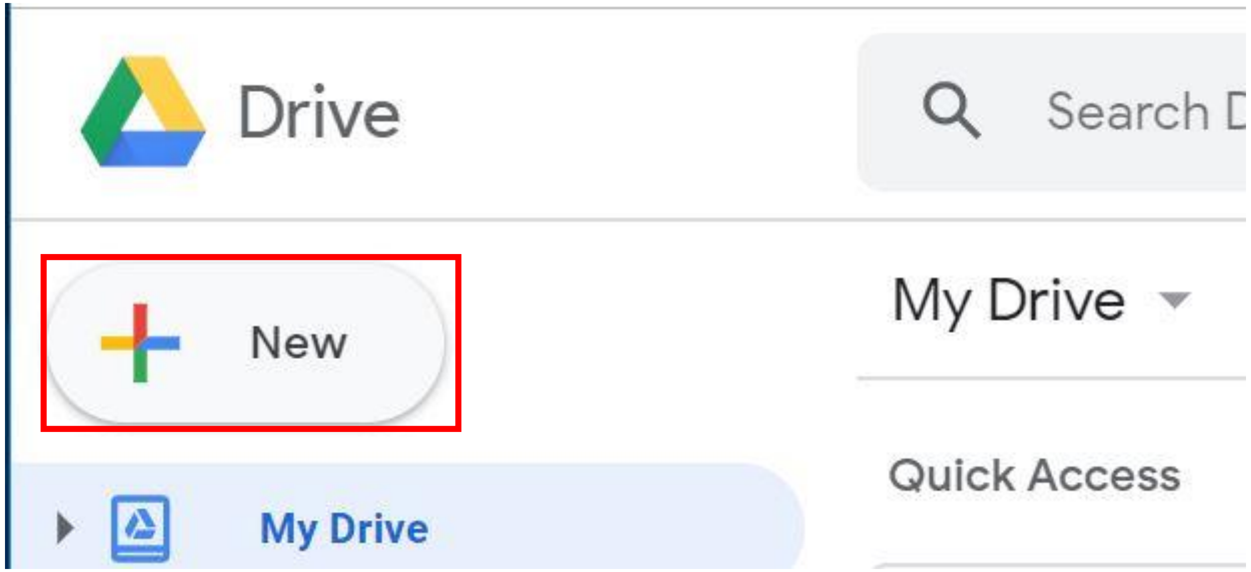


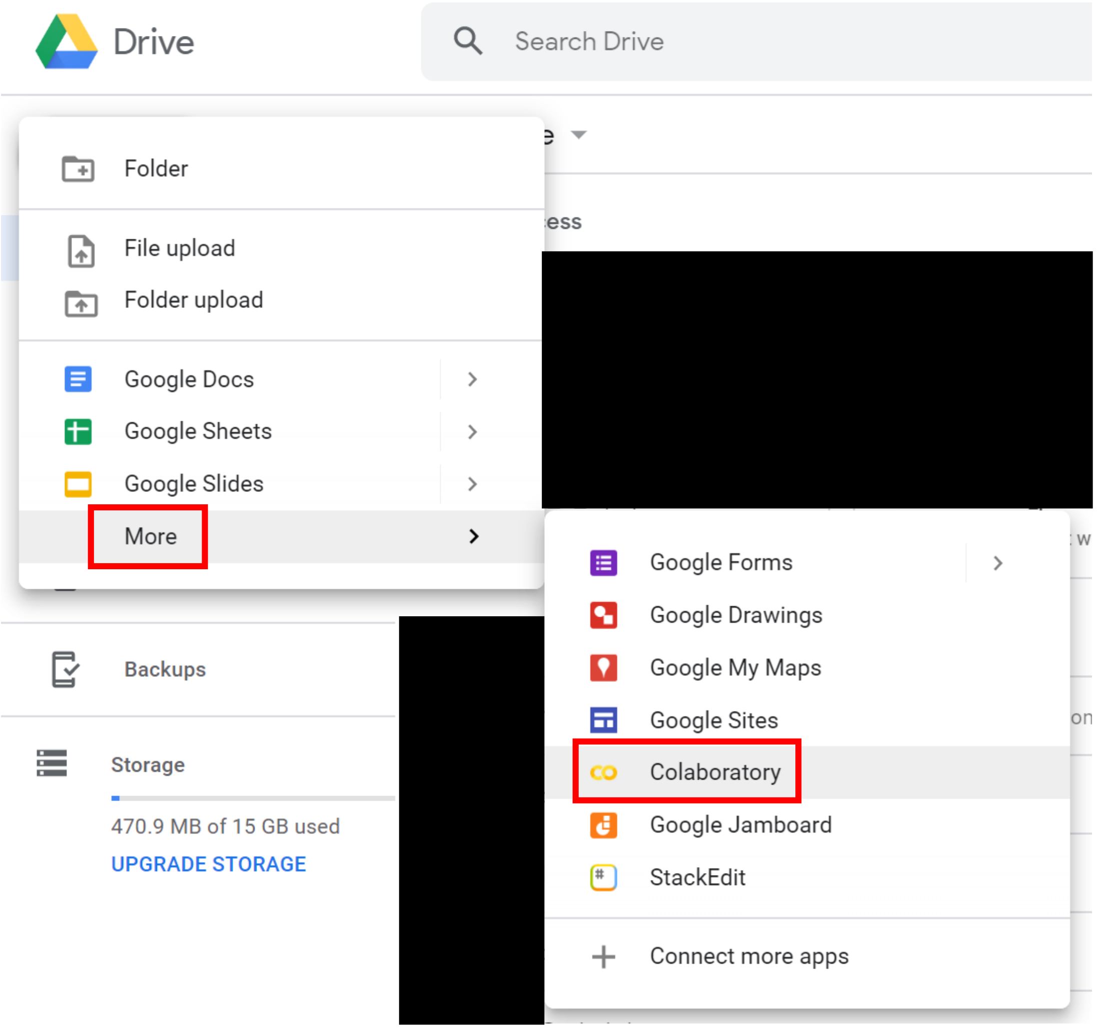


Google Colab – How to Start

1. Go to https://drive.google.com and login

2. ‘New’ -> ‘More’ -> ‘Colaboratory’

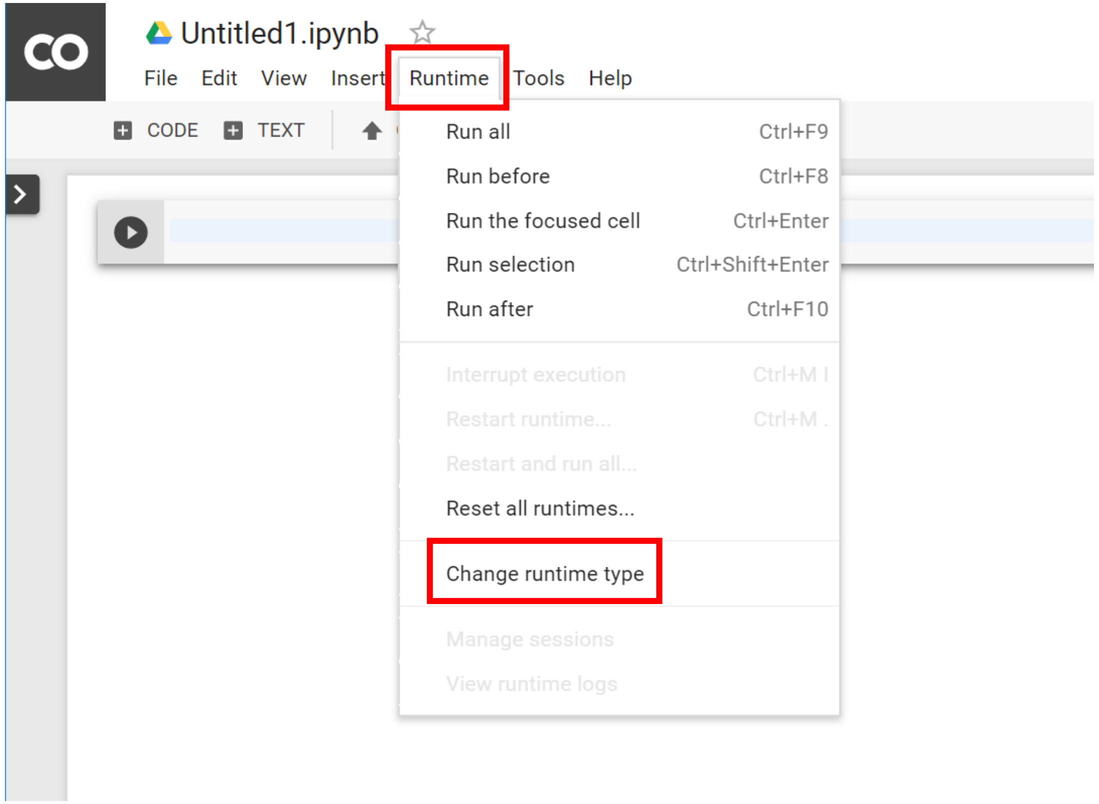


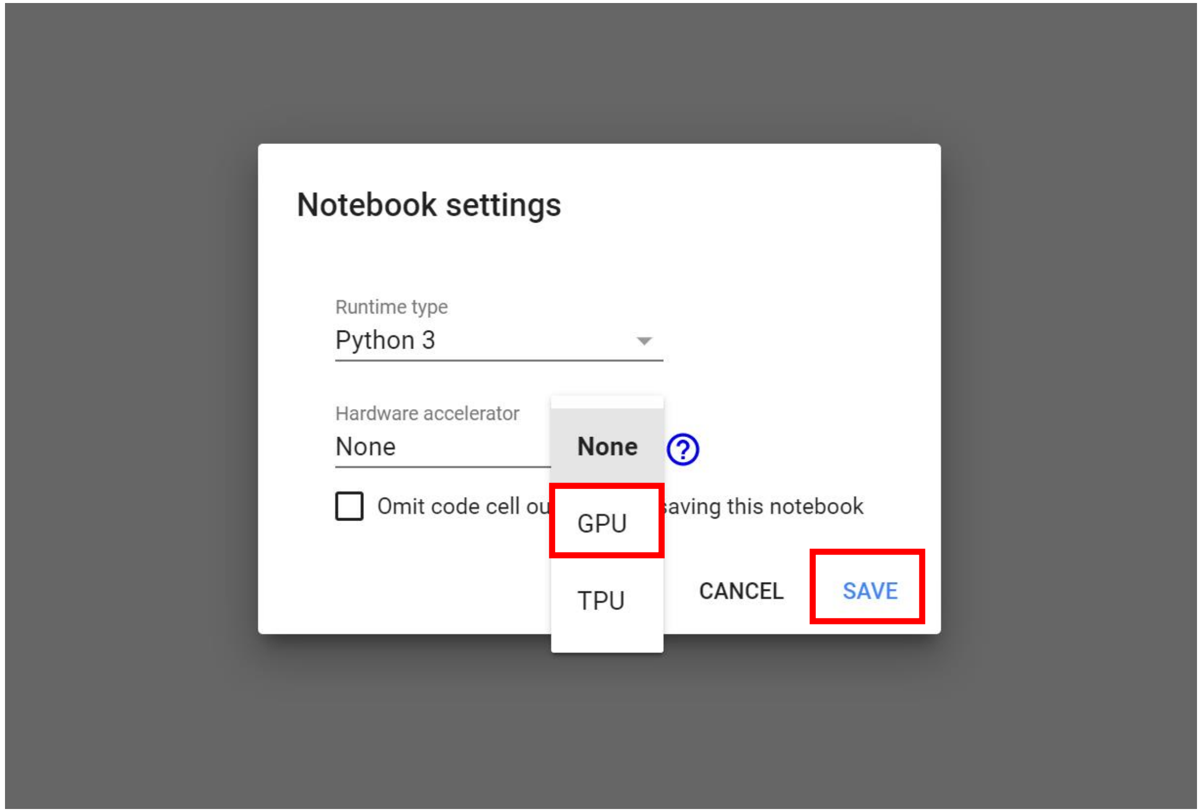


Google Colab – GPU Setting

3. In Google Colab, 'Runtime' -> 'Change runtime type'

4. Change 'Hardware accelerator' from 'None' to 'GPU' and save



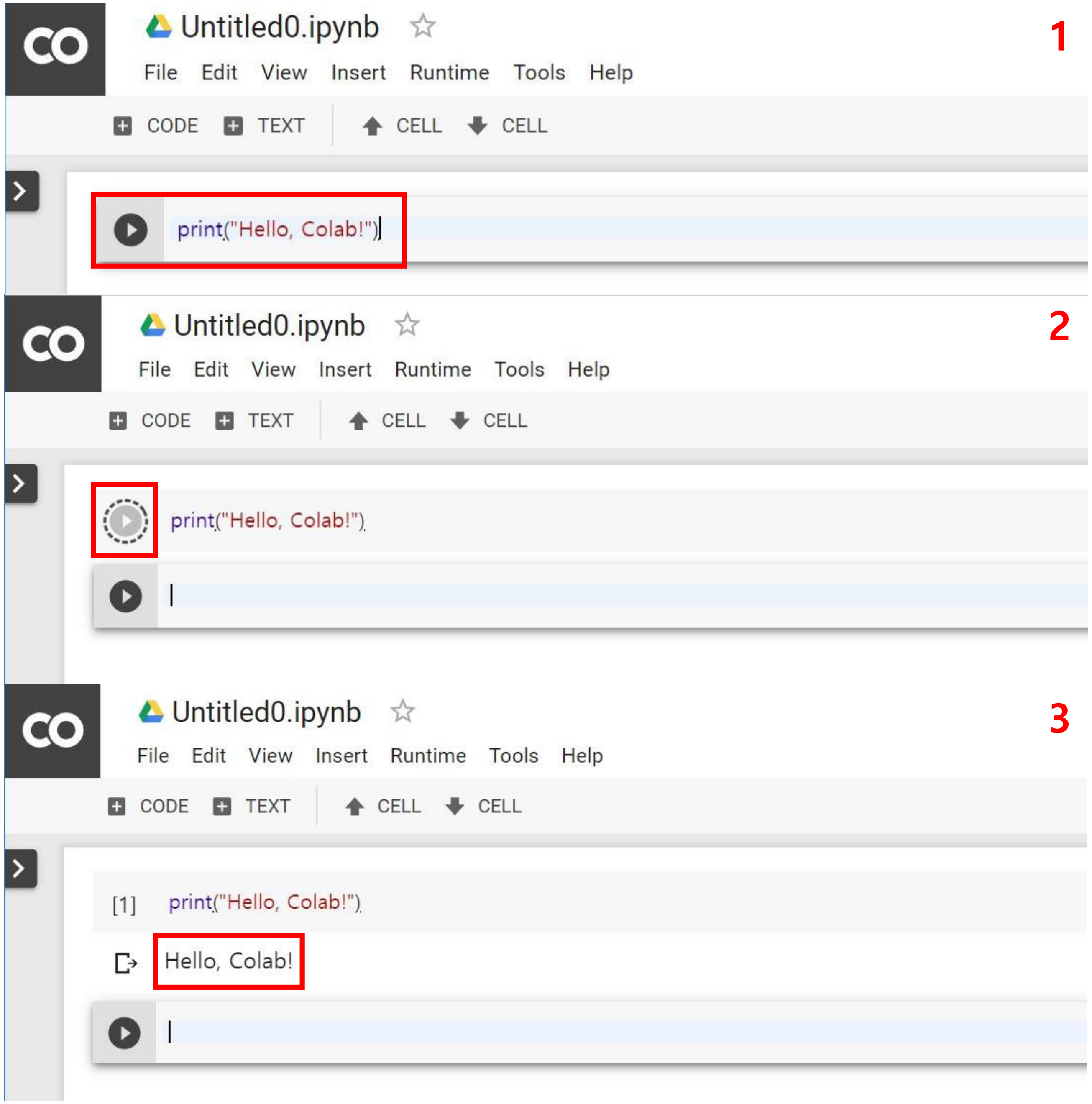


Google Colab - Usage

1. Write a python code in the cell

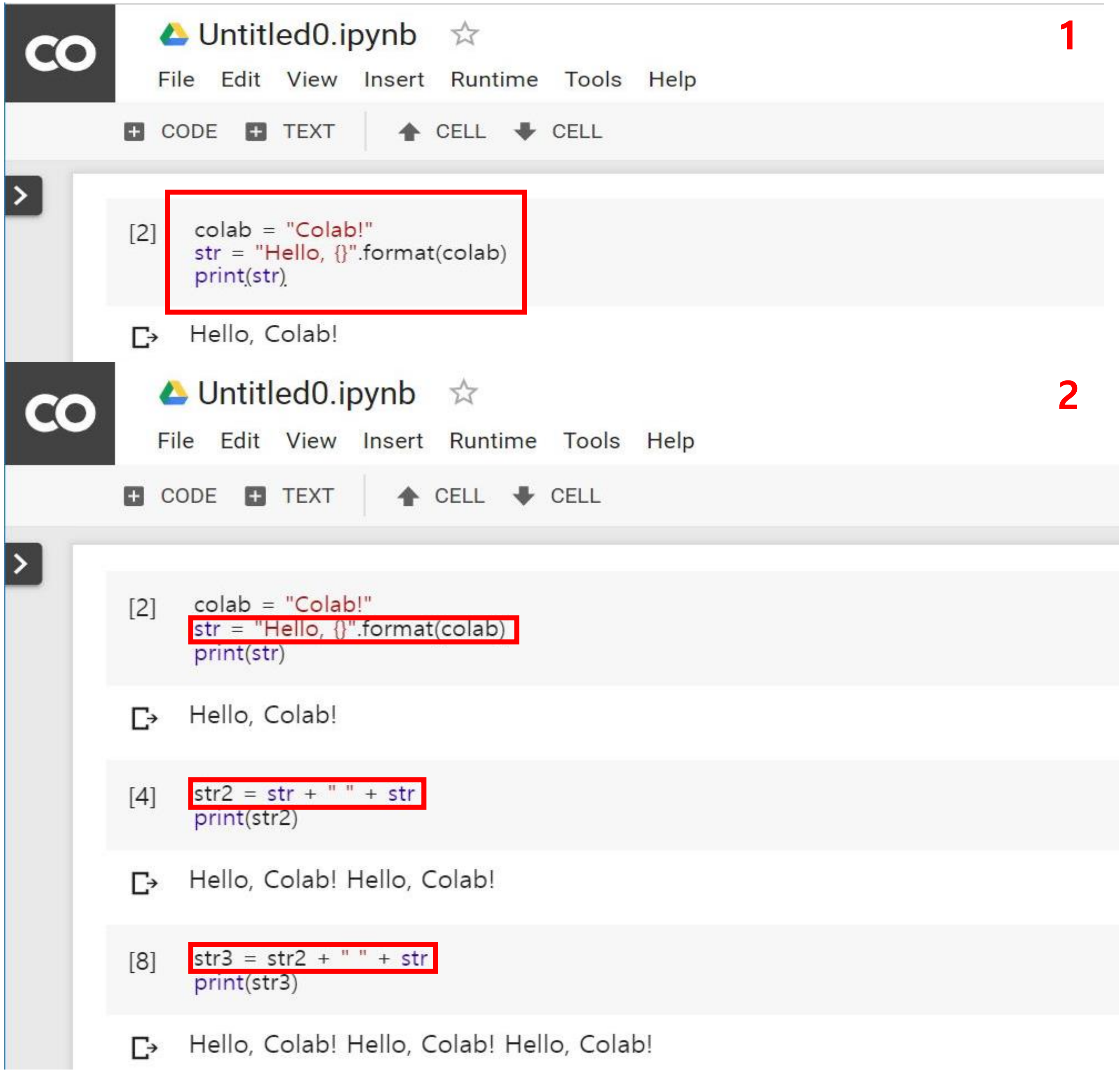
2. To execute the cell, press ‘**shift + enter**’

3. Check the result



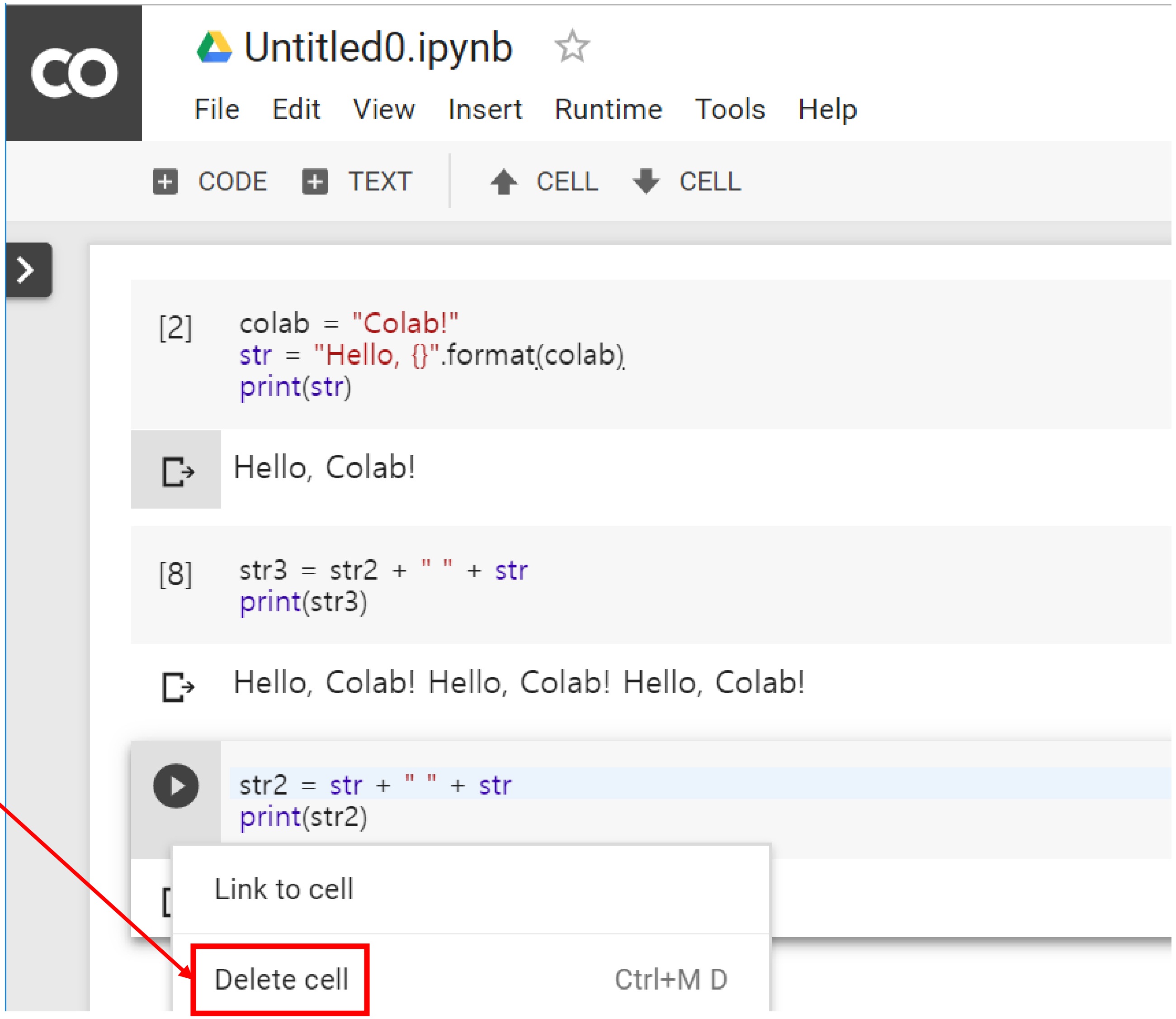
Google Colab - Usage

- You can write multiple lines of code in an one cell
- A code in one cell can use variables in upper cells.



Google Colab - Usage

- You can delete a cell, Just right click on the cell you want to delete



Google Colab – Running terminal command

- Terminal command can be used in a cell with a '!' symbol before the command.(e.g. !ls, !pwd, …)
  - The example picture below is a result of 'nvidia-smi' command.

```
[9]  !nvidia-smi

Sun May 26 07:14:57 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.67       Driver Version: 410.79       CUDA Version: 10.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   48C    P8    16W /  70W |      0MiB / 15079MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Google Colab – DEMO

NumPy Short Intro

- NumPy is python package for scientific computation
- NumPy supports an efficient ‘ndarray’ that is similar to C/C++ array

NumPy Short Intro

- ndarray operations

1. **array(python_list)** makes a ndarray using python list
   e.g. np.array([1, 2, 3]) -> [1 2 3]

2. **arange(int)** makes a ndarray [1, … , int]
   e.g. np.arange(5) -> [1 2 3 4 5]

3. **zeros(size_tuple)** makes a ndarray full of zeros
   e.g. np.zeros((2, 3)) -> [[0. 0. 0.]
   [0. 0. 0.]]

4. **ones(size_tuple)** makes a ndarray full of ones
   e.g. np.ones((2, 2)) -> [[1. 1.]
   [1. 1.]]

NumPy Short Intro

```
[1]  import numpy as np
```

```
[2]  array = [[1, 2, 3, 4, 5], [5, 6, 7, 8, 9], [1, 3, 5, 7, 9]]
     np_array = np.array(array)
     print(type(np_array))
     print(np_array)
```

```
<class 'numpy.ndarray'>
[[1 2 3 4 5]
 [5 6 7 8 9]
 [1 3 5 7 9]]
```

```
[16] np_arange_array = np.arange(7)
     print(type(np_arange_array), np_arange_array)
```

```
<class 'numpy.ndarray'> [0 1 2 3 4 5 6]
```

```
[17] np_zeros_array = np.zeros((3, 4))
     print(np_zeros_array)
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

```
[18] np_ones_array = np.ones((4, 6))
     print(np_ones_array)
```

```
[[1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]]
```

NumPy Short Intro

- Type conversion method

1. **ndarray.astype(dtype)** converts the data type of the ndarray into 'dtype'

2. You can set the data type of a ndarray at initialization

```
[2]  np_array = np.array([[1, 2], [3, 4]])
     print(np_array, np_array.dtype)
```

```
[[1 2]
 [3 4]] int64
```

```
[3]  np_array = np_array.astype(np.float32)
     print(np_array, np_array.dtype)
```

```
[[1. 2.]
 [3. 4.]] float32
```

```
[4]  np_array_float32 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]], dtype=np.float32)
     print(np_array_float32, np_array_float32.dtype)
```

```
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]] float32
```

NumPy Short Intro

- ndarray Indexing

1. Same as C/C++ array indexing

2. Other indexing method

```
[2] array = [[1, 2, 3, 4, 5], [5, 6, 7, 8, 9], [1, 3, 5, 7, 9]]
    np_array = np.array(array)
    print(type(np_array))
    print(np_array)
```

```
<class 'numpy.ndarray'>
[[1 2 3 4 5]
 [5 6 7 8 9]
 [1 3 5 7 9]]
```

```
[3] np_array[1][2]
```

```
7
```

Same

```
[4] np_array[1, 2]
```

```
7
```

NumPy Short Intro

- NumPy array has some properties about its shape and size

1. **ndarray.ndim** is the number of dimensions of the ndarray
2. **ndarray.shape** is the dimension of the ndarray
3. **ndarray.size** is the size of the ndarray
4. **ndarray.dtype** is the data type of the ndarray

```
[17] np_zeros_array = np.zeros((3, 4))
     print(np_zeros_array)
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

```
[19] print(np_zeros_array.ndim)
```

```
2
```

```
[20] print(np_zeros_array.shape)
```

```
(3, 4)
```

```
[21] print(np_zeros_array.size)
```

```
12
```

```
[22] print(np_zeros_array.dtype)
```

```
float64
```

NumPy – DEMO

NumPy Short Intro

- Quick Start Guide - https://docs.scipy.org/doc/numpy/user/quickstart.html
- API - https://docs.scipy.org/doc/numpy/reference/

CuPy

- CuPy allows compiling & running CUDA kernel in python

```
>>> import numpy as np
>>> import cupy as cp
>>> add_kernel = cp.RawKernel(r'''
... extern "C" __global__
... void my_add(const float* x1, const float* x2, float* y) {
...     int tid = blockDim.x * blockIdx.x + threadIdx.x;
...     y[tid] = x1[tid] + x2[tid];
... }
... ''', 'my_add')
>>> x1 = cupy.arange(25, dtype=cupy.float32).reshape(5, 5)
>>> x2 = cupy.arange(25, dtype=cupy.float32).reshape(5, 5)
>>> y = cupy.zeros((5, 5), dtype=cupy.float32)
>>> add_kernel((5,), (5,), (x1, x2, y))  # grid, block and arguments
>>> y
array([[ 0.,  2.,  4.,  6.,  8.],
       [10., 12., 14., 16., 18.],
       [20., 22., 24., 26., 28.],
       [30., 32., 34., 36., 38.],
       [40., 42., 44., 46., 48.]], dtype=float32)
```

Support CUDA C/C++ kernel code

CuPy – Usage

1. After the GPU setting of Google Colab, Type ‘import cupy’

2. You can change NumPy ndarray into CuPy array using **asarray(ndarray)**. The ndarray from main memory is moved to GPU memory

3. You can also change CuPy array into NumPy ndarray using **asnumpy(cupy_array)** The ndarray from GPU memory is move to main memory

```
[24] import cupy as cp
     import numpy as np

[25] np_array = np.arange(10)
     print(type(np_array), np_array)

<class 'numpy.ndarray'> [0 1 2 3 4 5 6 7 8 9]

[27] cp_array = cp.asarray(np_array)
     print(type(cp_array), cp_array)

<class 'cupy.core.core.ndarray'> [0 1 2 3 4 5 6 7 8 9]

[28] np_array_again = cp.asnumpy(cp_array)
     print(type(np_array_again), np_array_again)

<class 'numpy.ndarray'> [0 1 2 3 4 5 6 7 8 9]
```

CuPy - Use with CUDA C/C++

1. Using **RawKernel** , CUDA C/C++ kernel code can be ran on Python

2. ‘extern “C”’ must be needed before the kernel code.
   - Because the kernel code is compiled and the compiled binary is used in python.

3. The second parameter of ‘RawKernel’ must be same as the kernel name.

```
import cupy as cp
import numpy as np

def get_matmul():
    matmul = cp.RawKernel("""
    extern "C" __global__
    void matmul(float *dest, float *a, float *b, int n, int k, int m)
    {
      const int col = blockDim.x * blockIdx.x + threadIdx.x;
      const int row = blockDim.y * blockIdx.y + threadIdx.y;

      // dimension of dest is n * m
      if (row < n && col < m) {
        float tmp = 0.0f;
        for (int i=0; i<k; i++) {
          tmp += a[row*k+i]*b[m*i+col];
        }
        dest[row*m+col] = tmp;
      }
    }
    """, 'matmul')
    return matmul
```

CuPy - Use with CUDA C/C++

1. Before using the python function from CUDA C/C++ kernel, Make two NumPy ndarray

2. Move two NumPy ndarray to CuPy ndarray (the ndarray is moved to GPU memory)

3. Make variable that save the result and move it to GPU memory

```
def run_matmul():
    matmul = get_matmul()
    N = 1024
    K = 512
    M = 2048
    BLOCK = (32, 32, 1)
    GRID = (128, 128)

    m_a = np.random.randn(N*K).astype(np.float32)
    m_b = np.random.randn(K*M).astype(np.float32)

    a_gpu = cp.asarray(m_a)
    b_gpu = cp.asarray(m_b)

    dest = np.zeros(N*M).astype(np.float32)
    d_gpu = cp.asarray(dest)
    print("launching matmul kernel")

    matmul(GRID, BLOCK, (d_gpu, a_gpu, b_gpu, np.int32(N), np.int32(K), np.int32(M)))

    print("dest-m_a*m_b:",
          cp.asnumpy(d_gpu).reshape(N,M) - np.matmul(m_a.reshape(N,K), m_b.reshape

if __name__=='__main__':
    run_matmul()
```

CuPy - Use with CUDA C/C++

1. Call python function from CUDA C/C++
   1. First parameter(In picture, ‘GRID’) is tuple describe grid size
   2. Second parameter(In picture, ‘BLOCK’) is tuple describe block size
   3. Third parameter is tuple that means inputs of CUDA kernel
2. Move the result of CUDA kernel to main memory

```python
def run_matmul():
    matmul = get_matmul()
    N = 1024
    K = 512
    M = 2048
    BLOCK = (32, 32, 1)
    GRID = (128, 128)

    m_a = np.random.randn(N*K).astype(np.float32)
    m_b = np.random.randn(K*M).astype(np.float32)

    a_gpu = cp.asarray(m_a)
    b_gpu = cp.asarray(m_b)

    dest = np.zeros(N*M).astype(np.float32)
    d_gpu = cp.asarray(dest)
    print("launching matmul kernel")

    matmul(GRID, BLOCK, (d_gpu, a_gpu, b_gpu, np.int32(N), np.int32(K), np.int32(M)))

    print("dest-m_a*m_b:",
          cp.asnumpy(d_gpu).reshape(N,M) - np.matmul(m_a.reshape(N,K), m_b.reshape

if __name__=='__main__':
    run_matmul()
```

CuPy – DEMO

CuPy

- Official Tutorial : https://docs-cupy.chainer.org/en/stable/tutorial/basic.html
- CuPy API : https://docs-cupy.chainer.org/en/stable/reference/index.html

Support CUDA C/C++ kernel code

More Information of CUDA

1. CUDA Tutorial : https://www.nvidia.com/docs/IO/116711/sc11-cuda-c-basics.pdf

2. CUDA API : https://docs.nvidia.com/cuda/cuda-runtime-api/index.html

Appendix I. Installing CUDA toolkit

- If you have a NVIDIA GPU laptop or desktop, you can use CUDA C/C++ after installing CUDA toolkit.

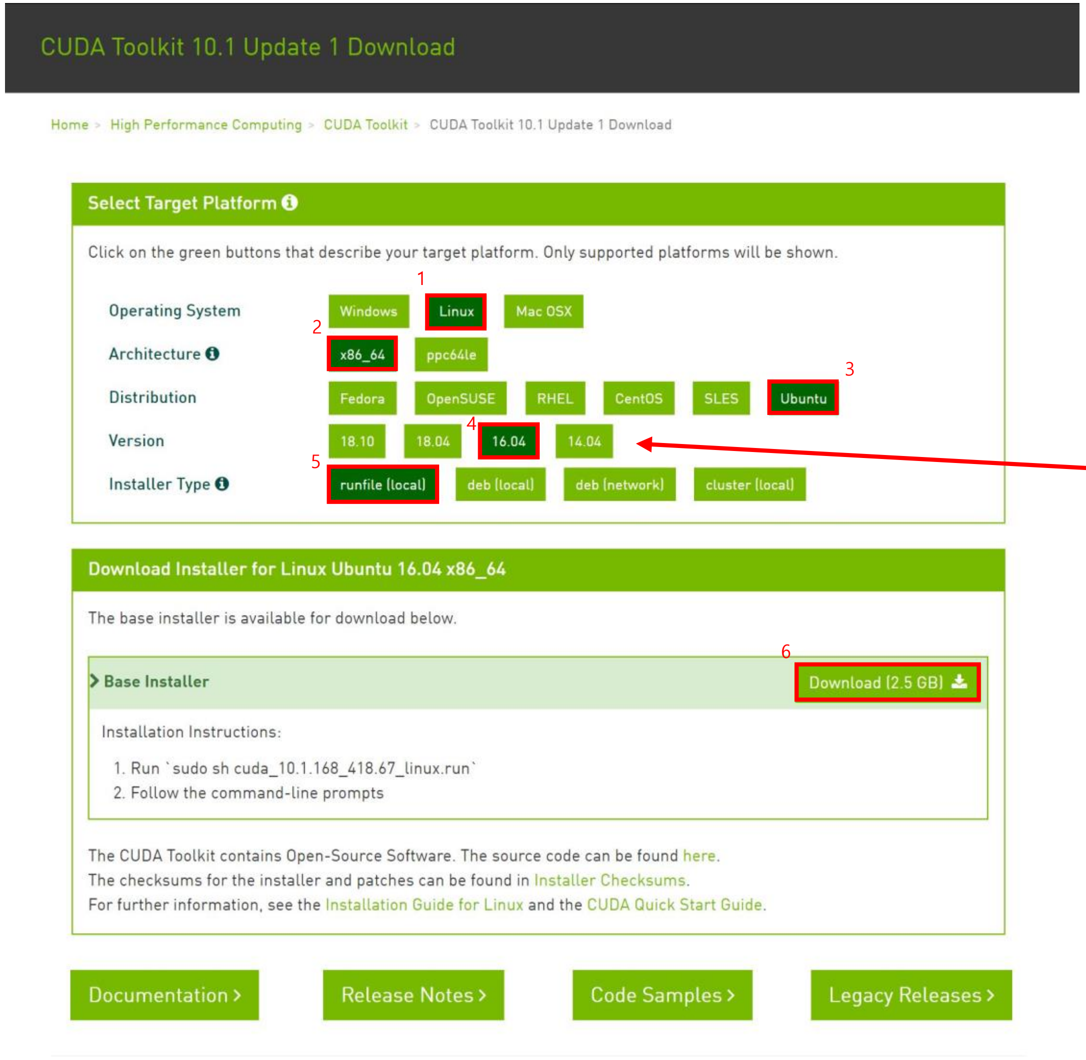
- In Windows, just download .exe installer file and install
  - Go to https://developer.nvidia.com/cuda-downloads and get the installer file

Appendix Ⅰ. Installing CUDA toolkit

- However, In Ubuntu, Installing is difficult.
  - Because CUDA graphic driver can crash on Ubuntu
  - Be careful : Ubuntu must not be on a virtual machine(Vmware, virtualbox, …)

1. First, download CUDA installer(.run file)
   - Go to https://developer.nvidia.com/cuda-downloads and follow the pictures at the next slide

Appendix Ⅰ. Installing CUDA toolkit



Choose the version of your Ubuntu

Appendix I. Installing CUDA toolkit

2. Delete Nouveau driver
    1. Type 'sudo vim /etc/modprobe.d/blacklist.conf'
    2. Add

       ```
       blacklist nouveau
       blacklist lbm-nouveau
       options nouveau modeset=0
       alias nouveau off
       alias lbm-nouveau off
       ```

       to bottom of the 'blacklist.conf' file
    3. Type 'echo options nouveau modeset=0 | sudo tee -a /etc/modprobe.d/nouveau-kms.conf'
    4. Type 'sudo update-initramfs -u'
    5. Reboot

Appendix Ⅰ. Installing CUDA toolkit

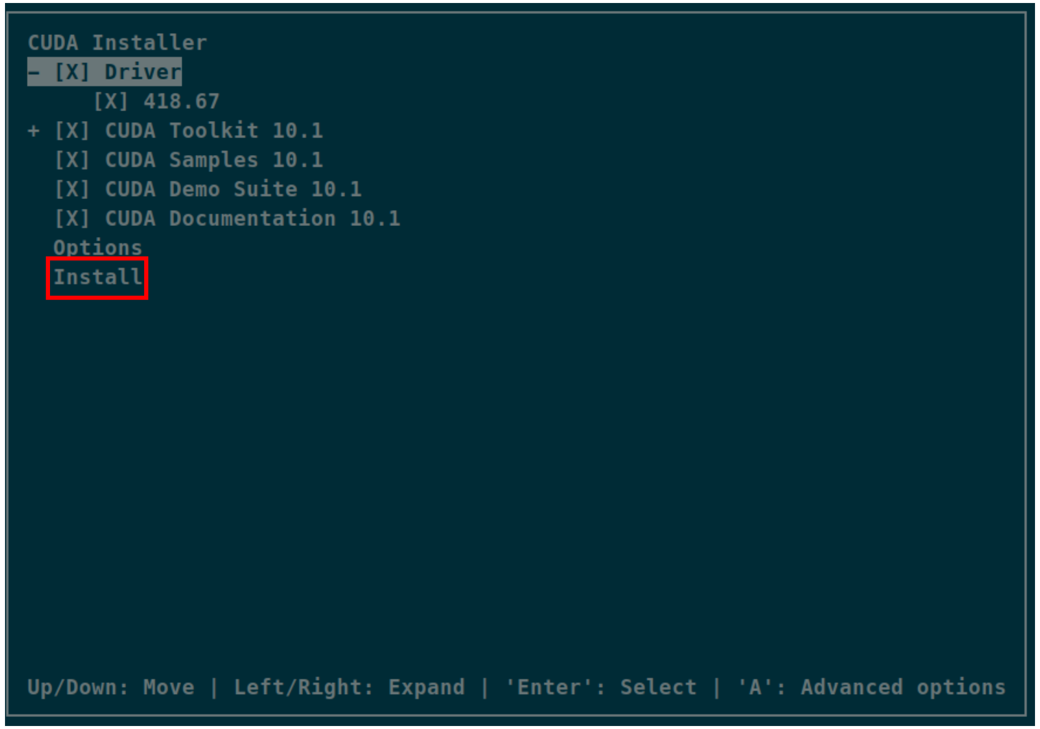
2. After rebooting, press ‘ctrl + alt + F1’ and login

3. Type ‘sudo service lightdm stop’

Appendix I. Installing CUDA toolkit

4. Find CUDA installer(.run file) and execute it with ‘sudo’ command
    1. You must install graphic driver and CUDA toolkit. Please read the installation instruction carefully and install both.

1. Type ‘accept’



2. Move the cursor down to ‘Install’ and press ‘Enter’

Appendix I. Installing CUDA toolkit

5. After the Installing, add

```
export PATH=/usr/local/cuda/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
```

   to bottom of ~/.bashrc file and type ‘source ~/.bashrc’ on the terminal

Appendix I. Installing CUDA toolkit

6. Check the installation using 'nvidia-smi' or 'nvcc' command

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 378.13                 Driver Version: 378.13                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 1080    Off  | 0000:01:00.0      On |                  N/A |
| 34%   52C    P2    39W / 180W |   2553MiB /  8112MiB |     14%      Default |
+-------------------------------+----------------------+----------------------+
|   1  GeForce GTX 1080    Off  | 0000:02:00.0     Off |                  N/A |
| 27%   36C    P8     7W / 180W |      1MiB /  8114MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|    0      1292    G   /usr/lib/xorg/Xorg                            1719MiB |
|    0      1760    G   compiz                                         386MiB |
|    0      2431    G   ...nabled/SubresourceFilter/EnabledForPhishi   446MiB |
+-----------------------------------------------------------------------------+
```

Right result of 'nvidia-smi' command
(The version can be different)

```
pradeep@workhorse: ~/Downloads/CUDA
pradeep@workhorse:~/Downloads/CUDA$
pradeep@workhorse:~/Downloads/CUDA$
pradeep@workhorse:~/Downloads/CUDA$ nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2016 NVIDIA Corporation
Built on Tue_Jan_10_13:22:03_CST_2017
Cuda compilation tools, release 8.0, V8.0.61
pradeep@workhorse:~/Downloads/CUDA$
```

Right result of 'nvcc --version' command
(The version can be different)

- Reference : http://www.kwangsiklee.com/2017/07/%EC%9A%B0%EB%B6%84%ED%88%AC-16-04%EC%97%90%EC%84%9C-cuda-%EC%84%B1%EA%B3%B5%EC%A0%81%EC%9C%BC%EB%A1%9C-%EC%84%A4%EC%B9%98%ED%95%98%EA%B8%B0/ (Written in Korean)

End

Q & A : pl.hanyang@gmail.com