AI
on Nvidia Jetson Edge System

Jeong-Gun Lee

AI Accelerator Computing (AIAC) Lab

Division of Software, College of Info. Science, Hallym University

Jeonggun.lee@gmail.com

차례

- Nvidia Jetson System 소개
- Jetson Nano System 설정
- Jetson Nano에서 즐기는 Parallel Computing (CUDA)
- Jetson Nano에서 즐기는 Deep Learning

Jetson Nano에서 즐기는 Parallel Computing (CUDA)

Parallel Programming on a Manycore

We have one code part …

```
for(i=0; i < 100; i++)
        C[i] = A[i] + B[i];
```

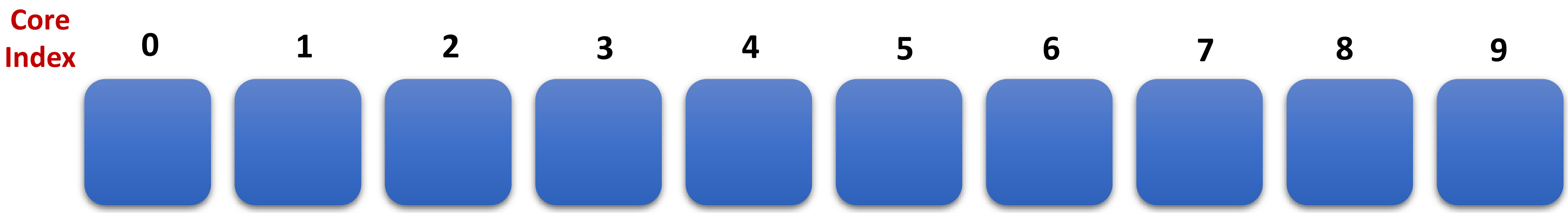
Yes! It is simple !

But what about on ***10 cores*** on-chip processor ?

Parallel Programming on a Manycore

But what about on 10 cores on-chip processor ?

```
for(i=0; i < 100; i++)
   C[i] = A[i] + B[i];
```

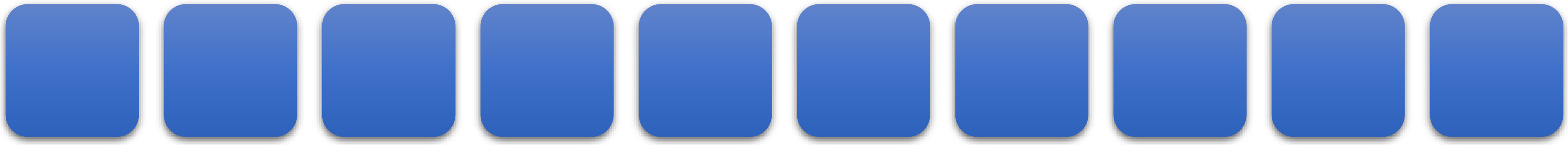


Parallel Programming on a Manycore

But what about on 10 cores on-chip processor ?

```
for(i=0; i < 100; i++)
   C[i] = A[i] + B[i];
```

Core Index 0 1 2 3 4 5 6 7 8 9



For each core

```
for(i=0; i < 10; i++)
   C[CoreIndex*10+i] = A[CoreIndex*10+i] + B[CoreIndex*10+i];
```

Parallel Programming on a Manycore

What about Threading ?

```
for(i=0; i < 100; i++)
   C[i] = A[i] + B[i];
```

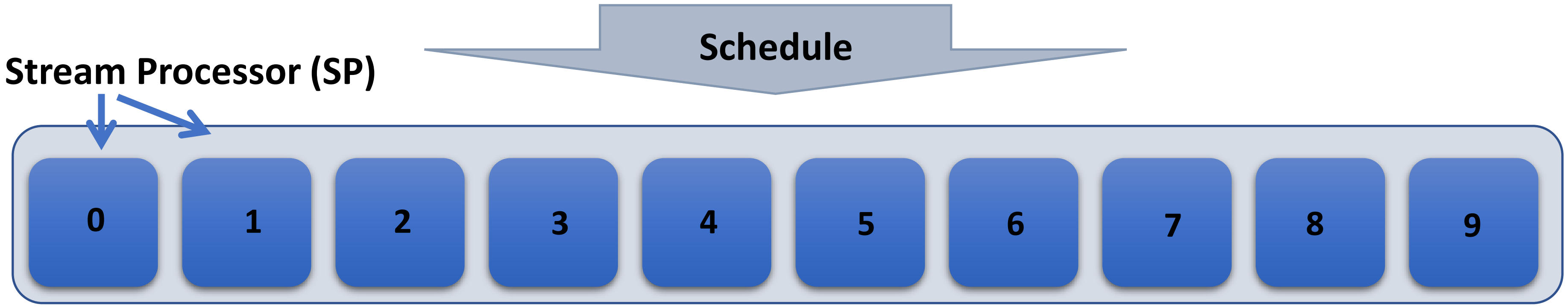
Thread 0:	Thread 1:	…	Thread 98:	Thread 99:
C[0] = A[0] + B[0];	C[1] = A[1] + B[1];		C[98] = A[98] + B[98];	C[99] = A[99] + B[99];

Parallel Programming on a Manycore

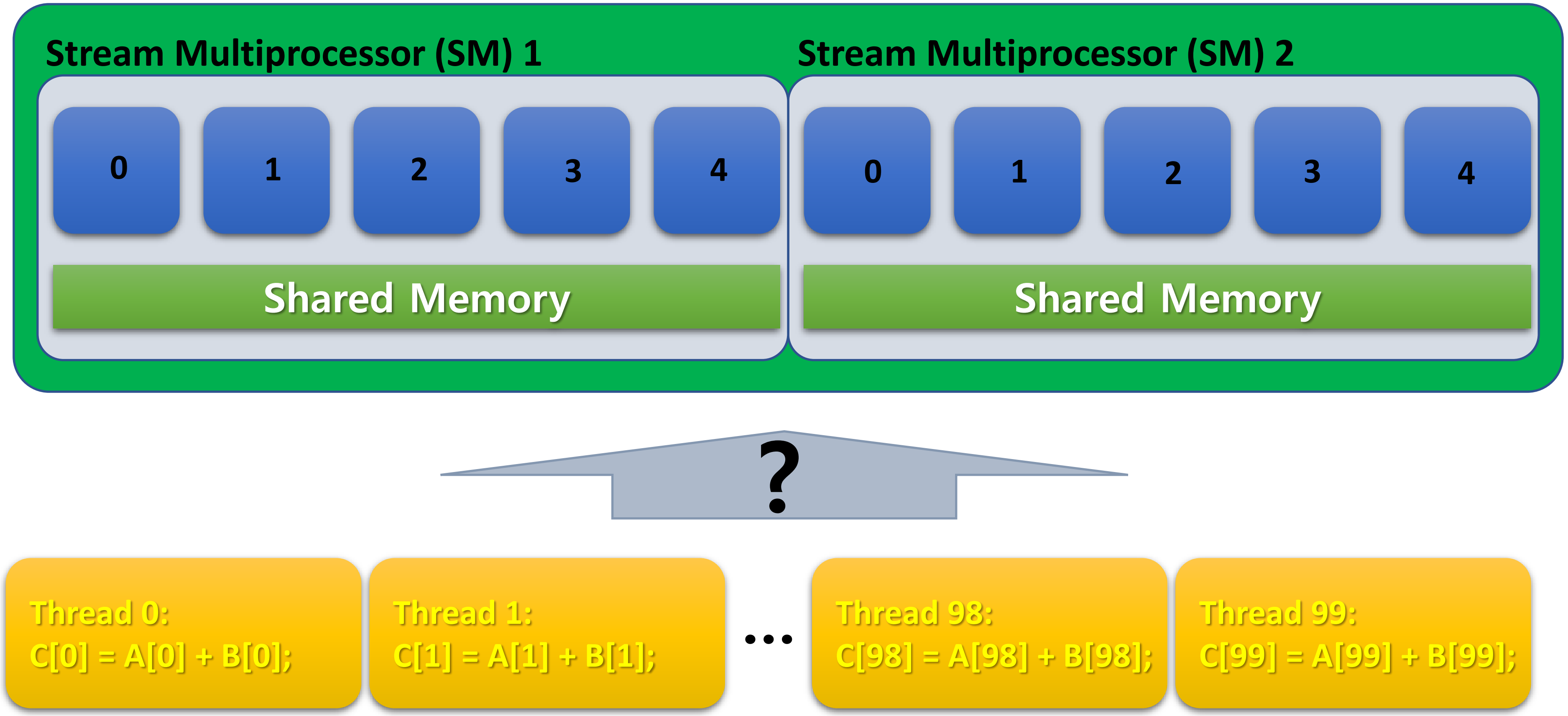
What about Threading ?

```
for(i=0; i < 100; i++)
   C[i] = A[i] + B[i];
```

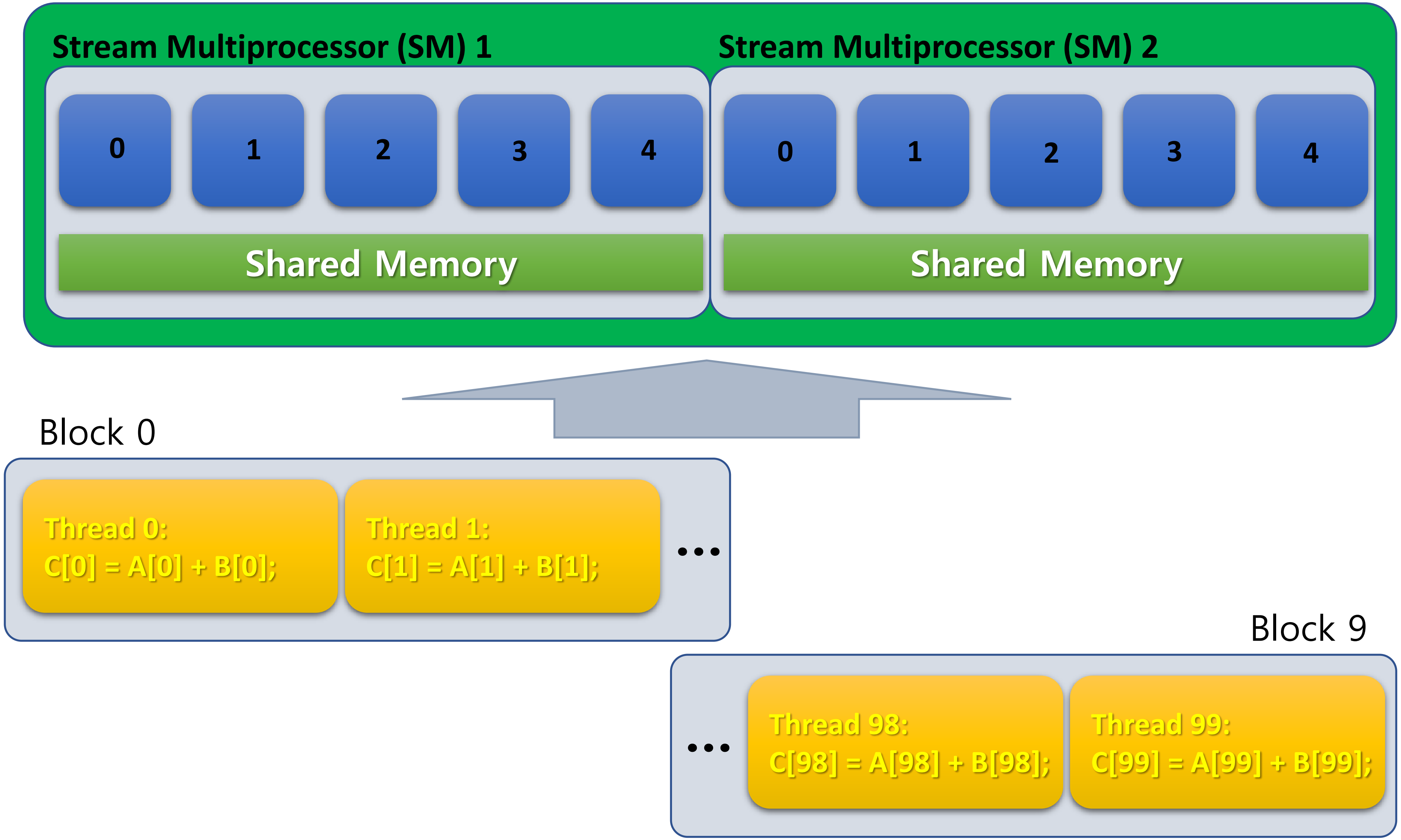
Thread 0:	Thread 1:	…	Thread 98:	Thread 99:
C[0] = A[0] + B[0];	C[1] = A[1] + B[1];		C[98] = A[98] + B[98];	C[99] = A[99] + B[99];



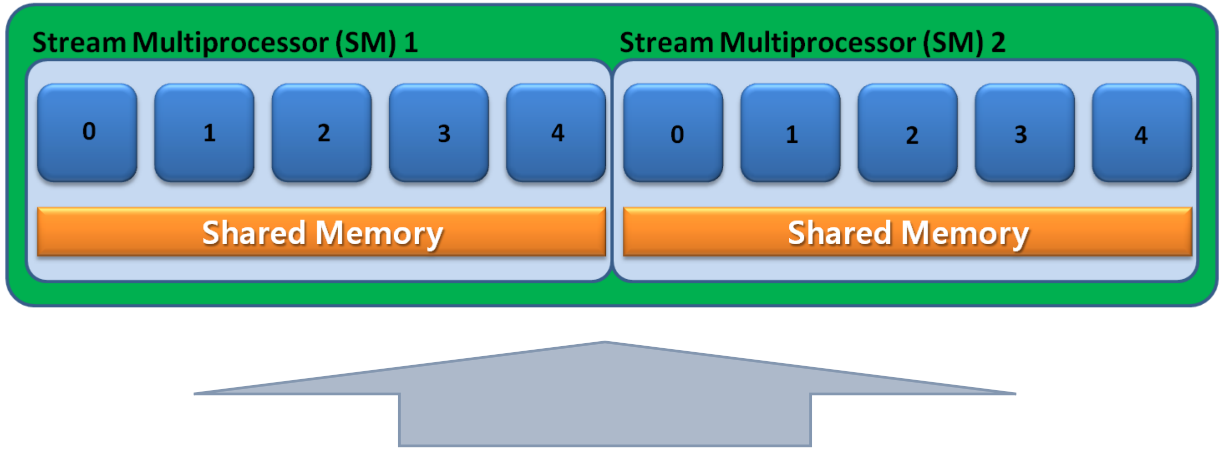
Parallel Programming on a Manycore

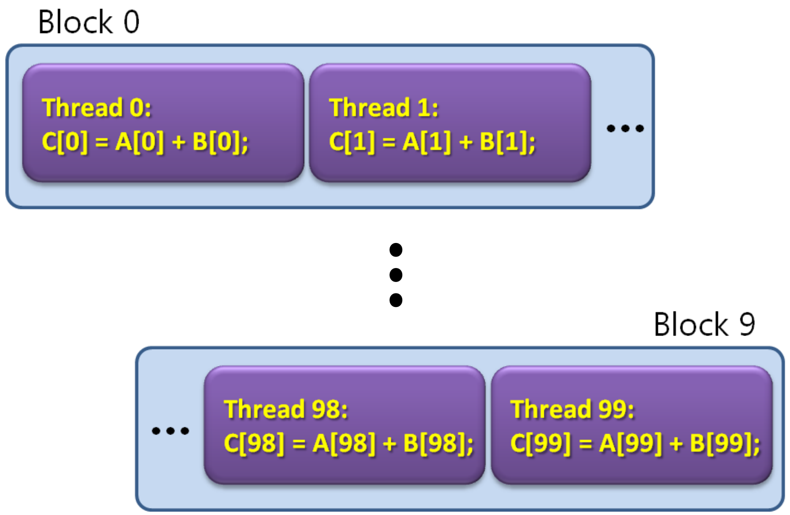


Parallel Programming on a Manycore



Parallel Programming on a Manycore





1 Thread per block & 100 Blocks

Or

5 Thread per block & 20 Blocks

Or

➡ 10 Thread per block & 10 Blocks

Or

20 Thread per block & 5 Blocks

…

Parallel Programming on a Manycore

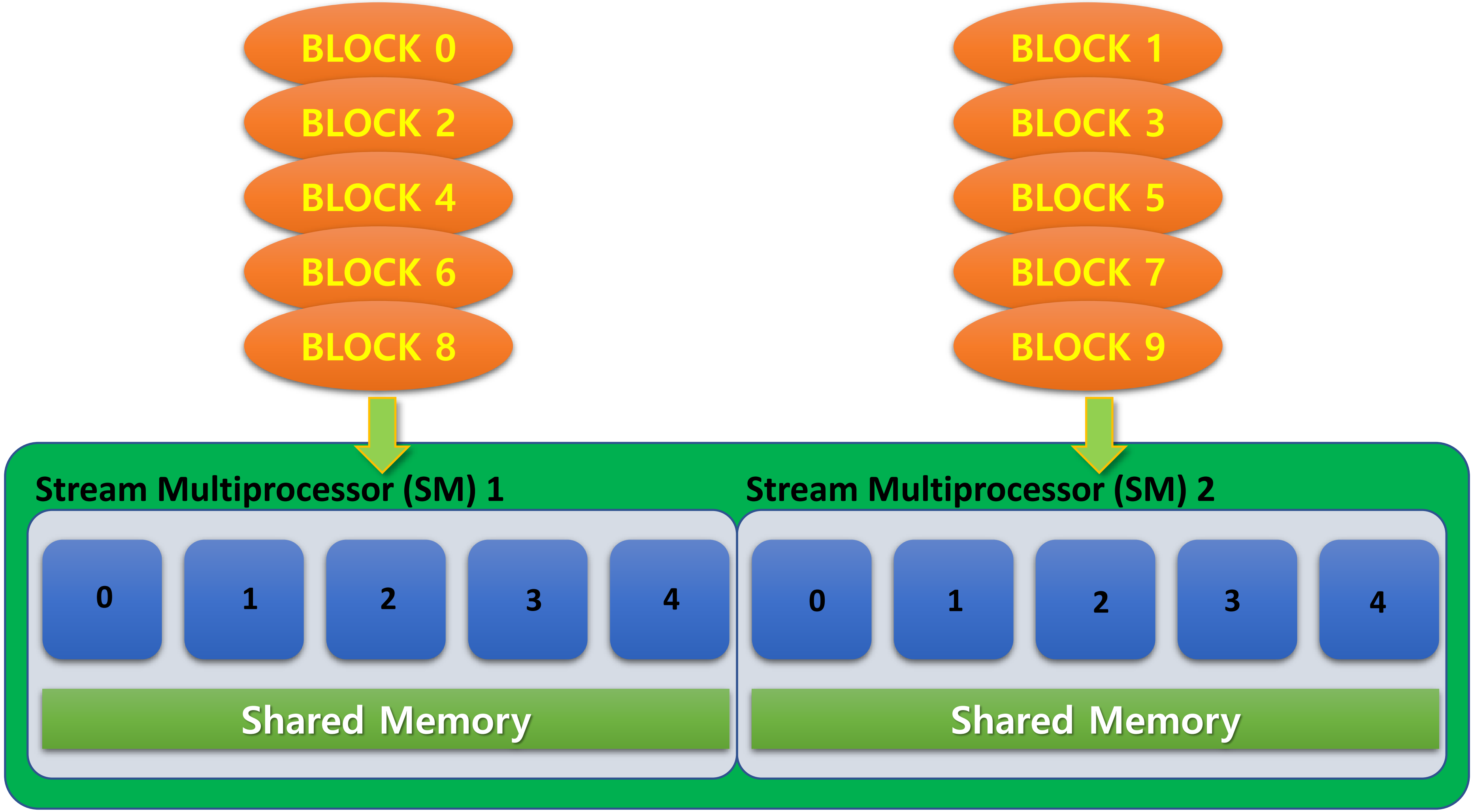
We have one code part …

```
for(i=0; i < 100; i++)
        C[i] = A[i] + B[i];
```

Now, **10 blocks** and **10 threads** per a block !

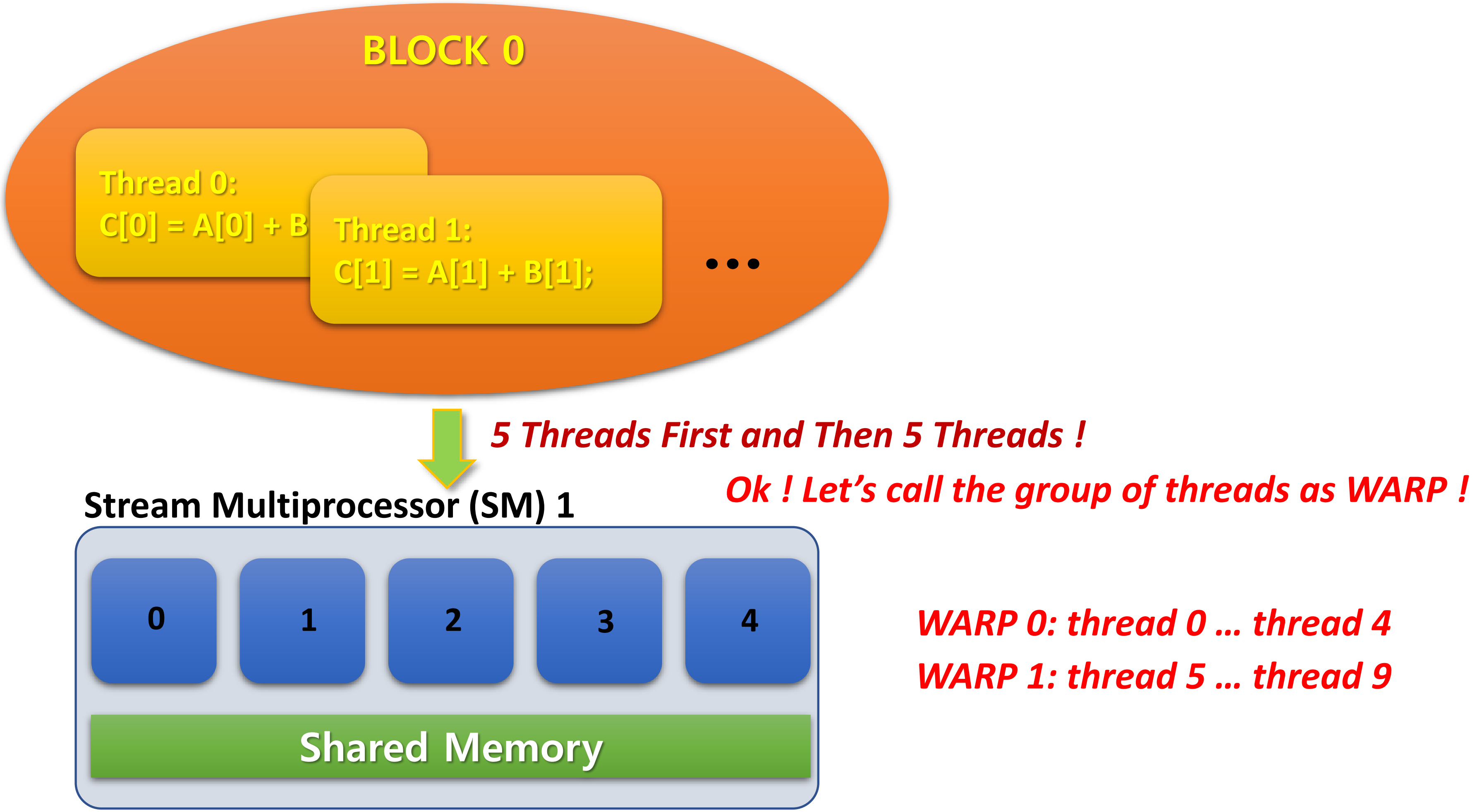
Parallel Programming on a Manycore

Now, **10 blocks** and **10 threads** per a block !



Parallel Programming on a Manycore

Now, **10 blocks** and **10 threads** per a block !



WARP 0: thread 0 … thread 4

WARP 1: thread 5 … thread 9

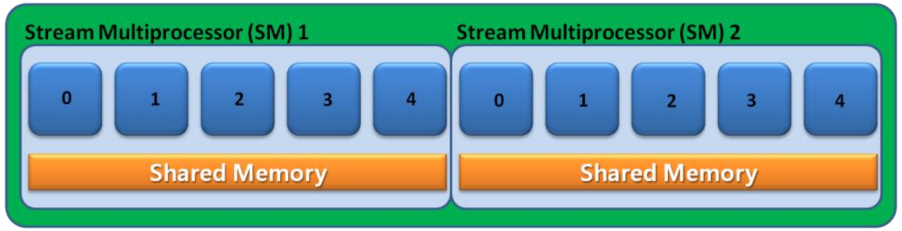
Parallel Programming on a Manycore

Hm… Where are A, B, C arrays in a system ?

Parallel Programming on a Manycore

Hm… Where are A, B, C arrays in a system ?

A system ?

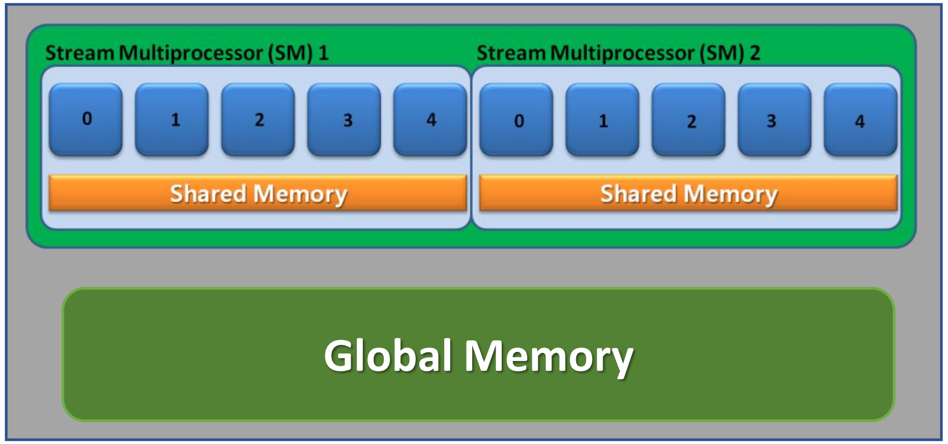


Parallel Programming on a Manycore

Hm… Where are A, B, C arrays in a system ?

A system ?

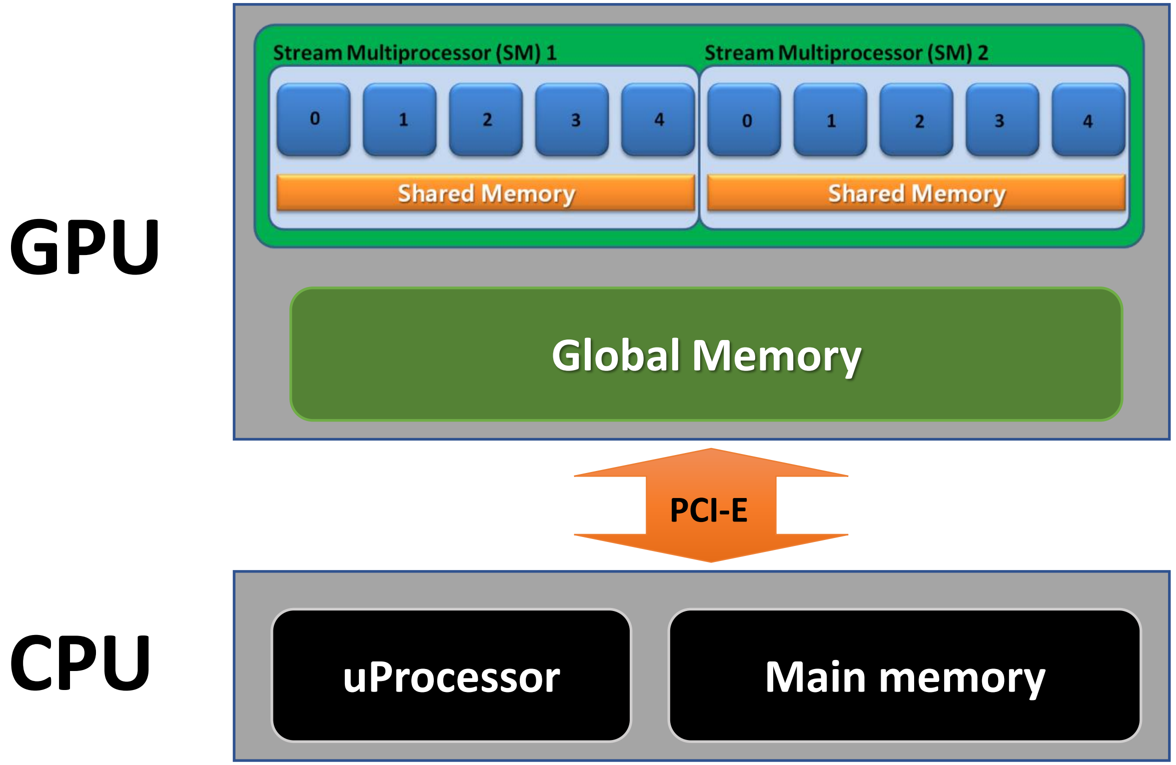
GPU



Parallel Programming on a Manycore

Hm… Where are A, B, C arrays in a system ?

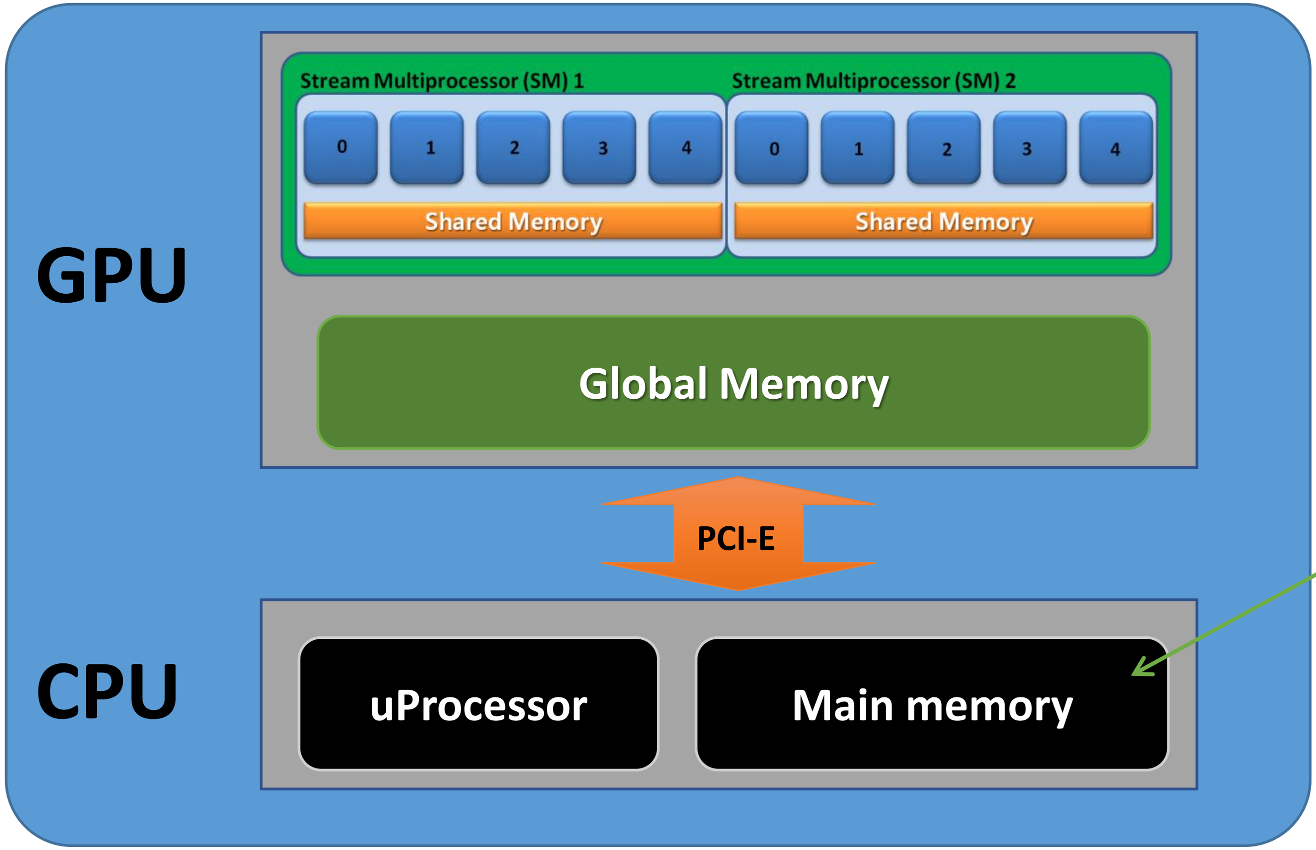
A system ?



Parallel Programming on a Manycore

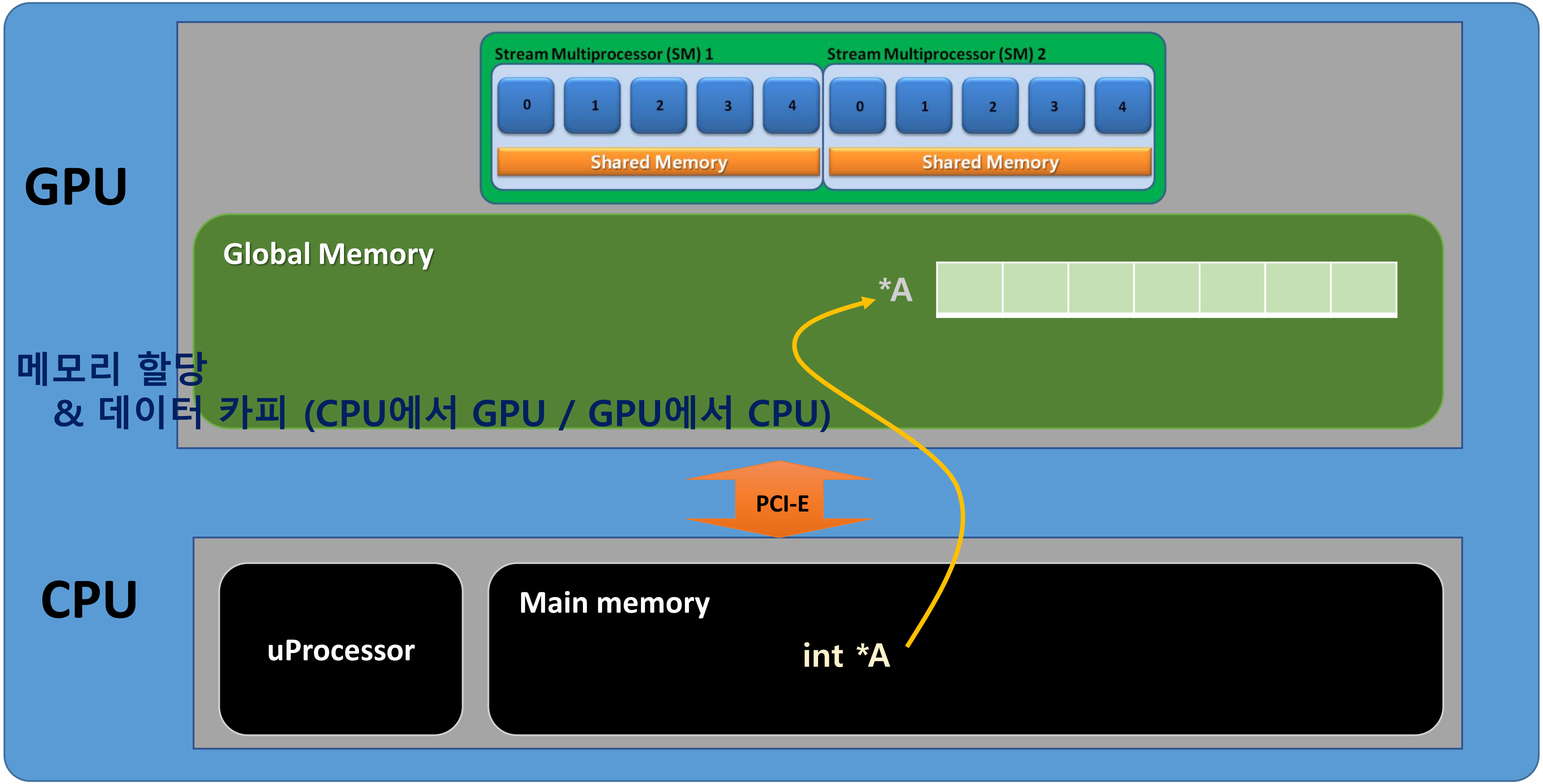
Hm… Where are A, B, C arrays in a system ?

A system !

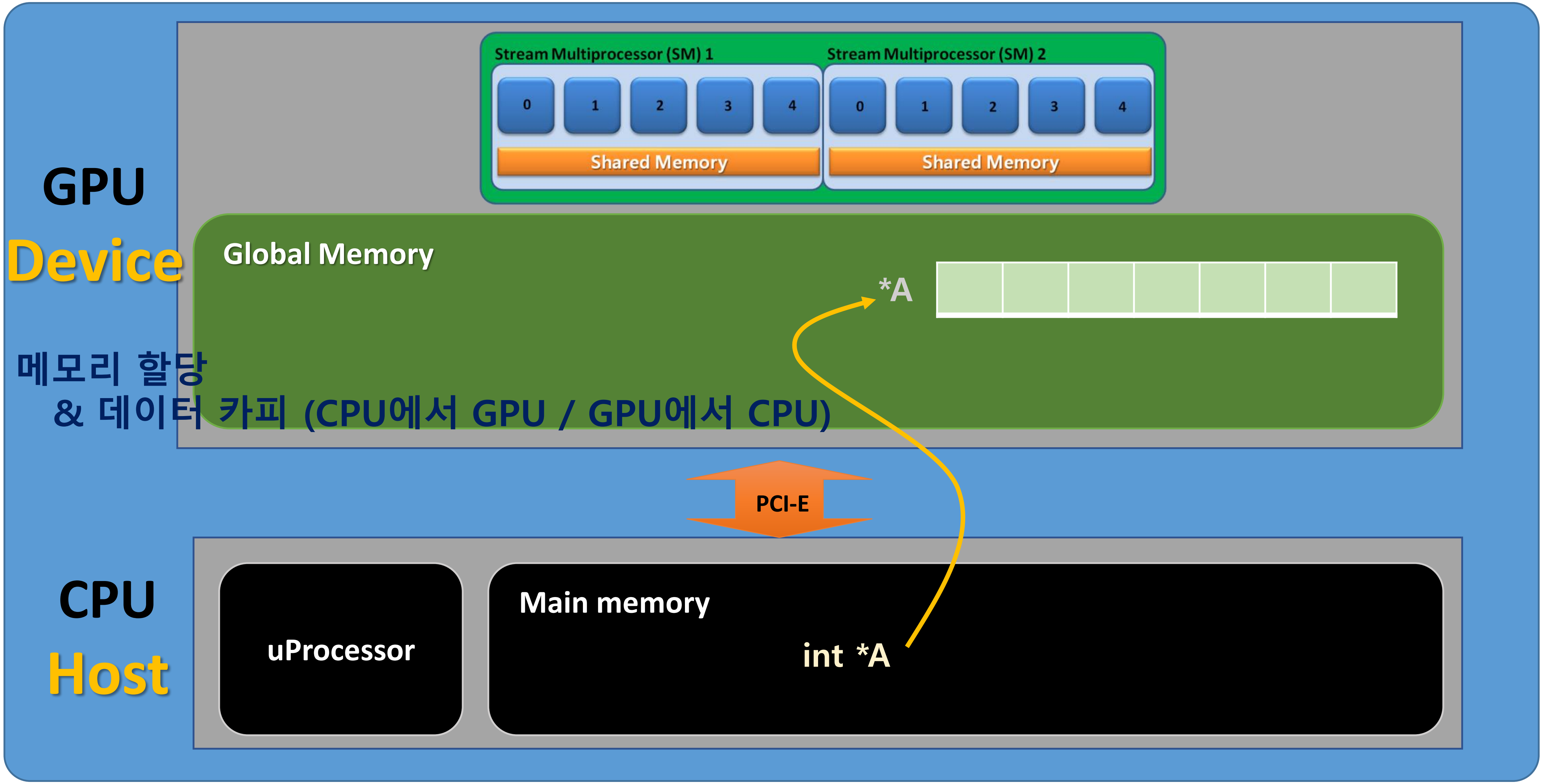


A, B are Here !

Parallel Programming on a Manycore

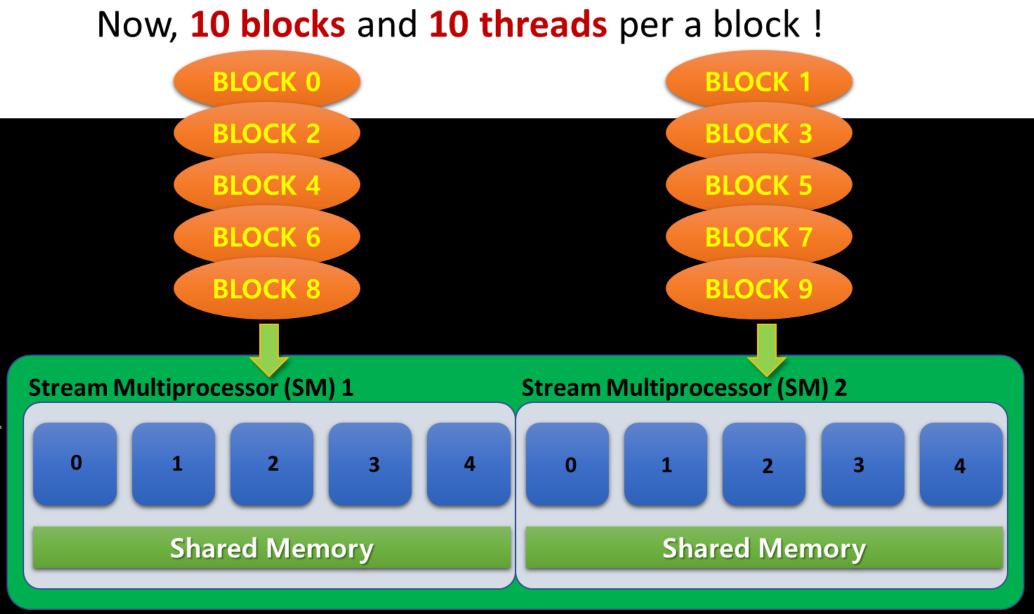


Parallel Programming on a Manycore



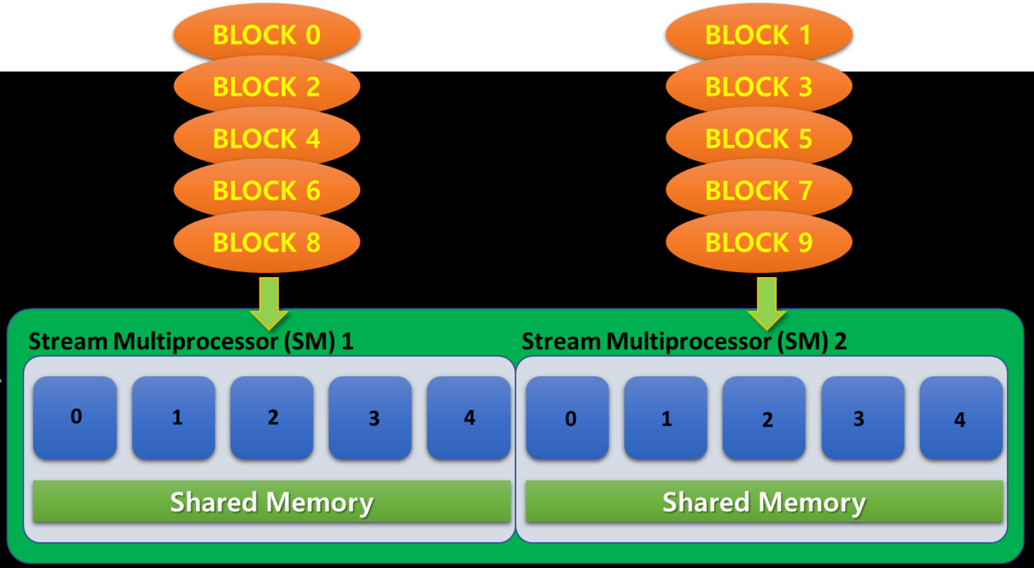
CUDA 실습하자!

```
#include <stdio.h>

__device__ void hiDeviceFunction(void)
{ //printf("Hello! This is in hiDeviceFunction. \n");}

__global__ void helloCUDA(void)
{
  printf("Hello thread %d\n", threadIdx.x);
  hiDeviceFunction();
}

int main()
{
  helloCUDA<<<1, 1>>>();
  return 0;
}
```



CUDA 실습하자!

Now, **10 blocks** and **10 threads** per a block !



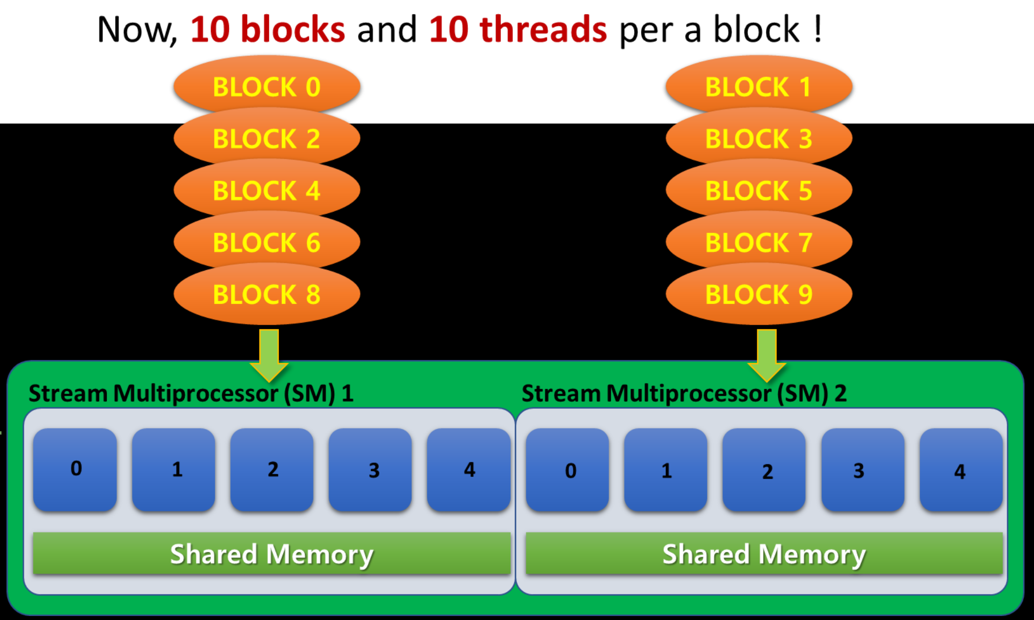

```
#include <stdio.h>

__device__ void hiDeviceFunction(void)
{ //printf("Hello! This is in hiDeviceFunction. \n");}

__global__ void helloCUDA(void)
{
  printf("Hello thread %d\n", threadIdx.x);
  hiDeviceFunction();
}

int main()
{
  helloCUDA<<<1, 1>>>();
  return 0;
}
```

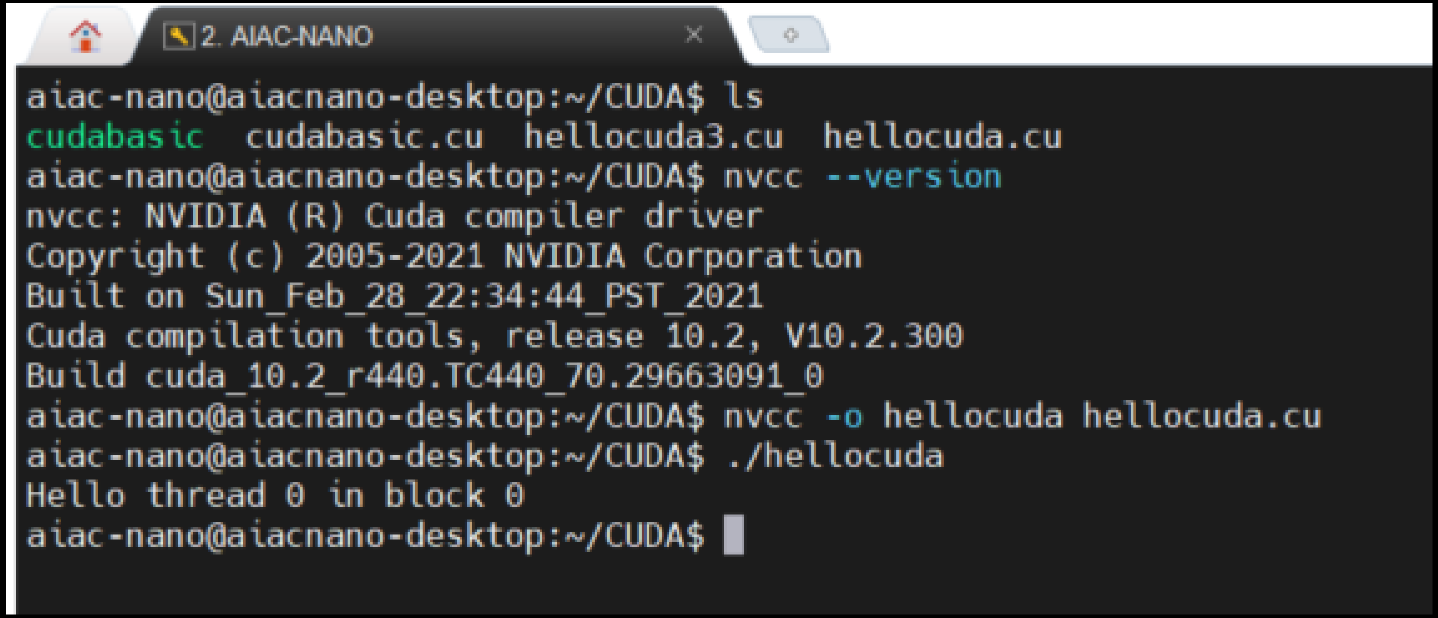
2. AIAC-NANO

```
aiac-nano@aiacnano-desktop:~/CUDA$ ls
cudabasic  cudabasic.cu  hellocuda3.cu  hellocuda.cu
aiac-nano@aiacnano-desktop:~/CUDA$ nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2021 NVIDIA Corporation
Built on Sun_Feb_28_22:34:44_PST_2021
Cuda compilation tools, release 10.2, V10.2.300
Build cuda_10.2_r440.TC440_70.29663091_0
aiac-nano@aiacnano-desktop:~/CUDA$ nvcc -o hellocuda hellocuda.cu
aiac-nano@aiacnano-desktop:~/CUDA$ ./hellocuda
Hello thread 0 in block 0
aiac-nano@aiacnano-desktop:~/CUDA$
```

CUDA 실습하자!

```
#include <stdio.h>

__device__ void hiDeviceFunction(void)
{ //printf("Hello! This is in hiDeviceFunction. \n");}

__global__ void helloCUDA(void)
{
  printf("Hello thread %d\n", threadIdx.x);
  hiDeviceFunction();
}

int main()
{
  helloCUDA<<<1, 1>>>();
  // printf 함수가 완료될 때 까지 대기
  cudaDeviceSynchronize();
  return 0;
}
```

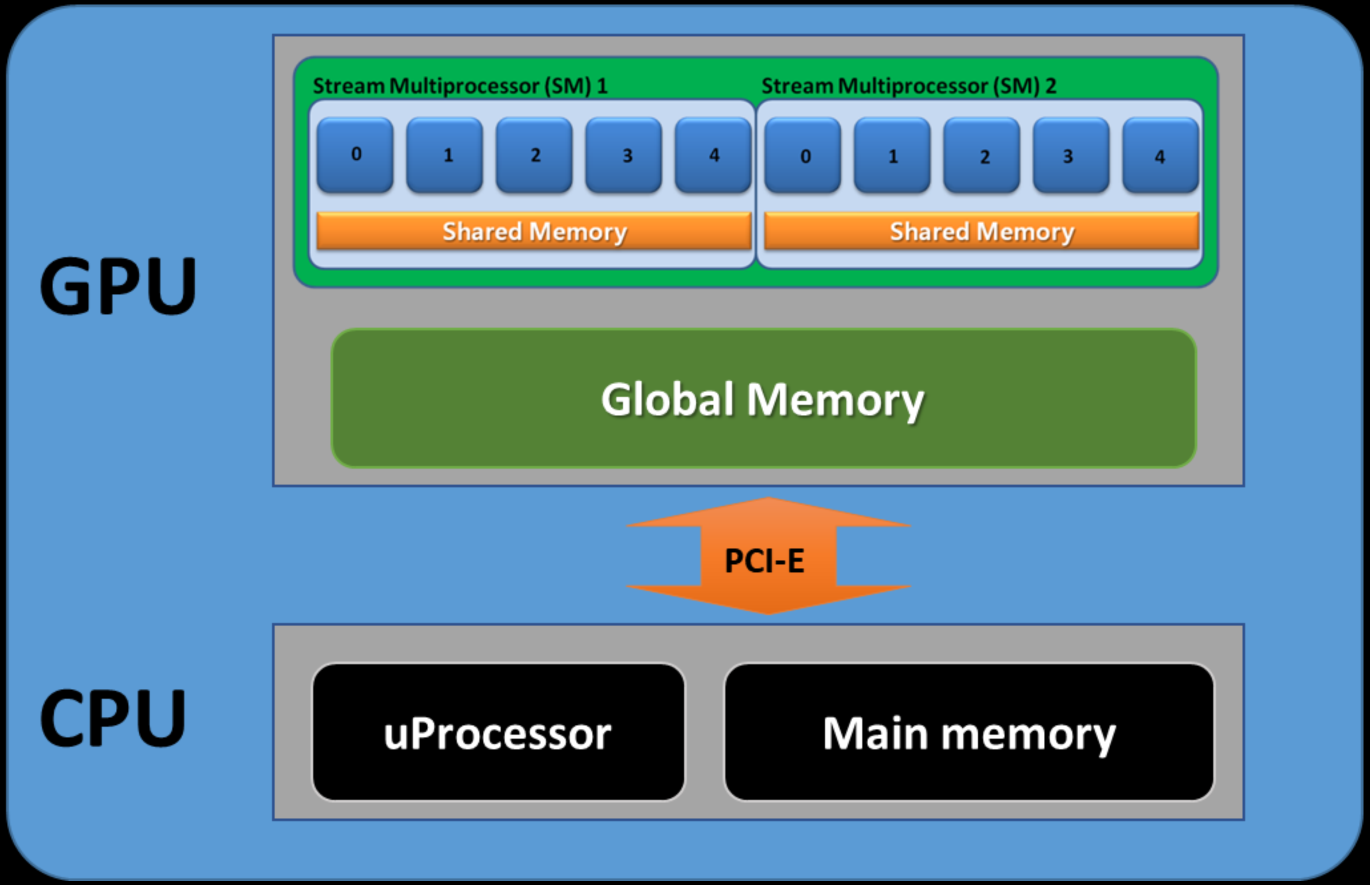




```
2. AIAC-NANO
aiac-nano@aiacnano-desktop:~/CUDA$ ls
cudabasic  cudabasic.cu  hellocuda3.cu  hellocuda.cu
aiac-nano@aiacnano-desktop:~/CUDA$ nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2021 NVIDIA Corporation
Built on Sun_Feb_28_22:34:44_PST_2021
Cuda compilation tools, release 10.2, V10.2.300
Build cuda_10.2_r440.TC440_70.29663091_0
aiac-nano@aiacnano-desktop:~/CUDA$ nvcc -o hellocuda hellocuda.cu
aiac-nano@aiacnano-desktop:~/CUDA$ ./hellocuda
Hello thread 0 in block 0
aiac-nano@aiacnano-desktop:~/CUDA$
```

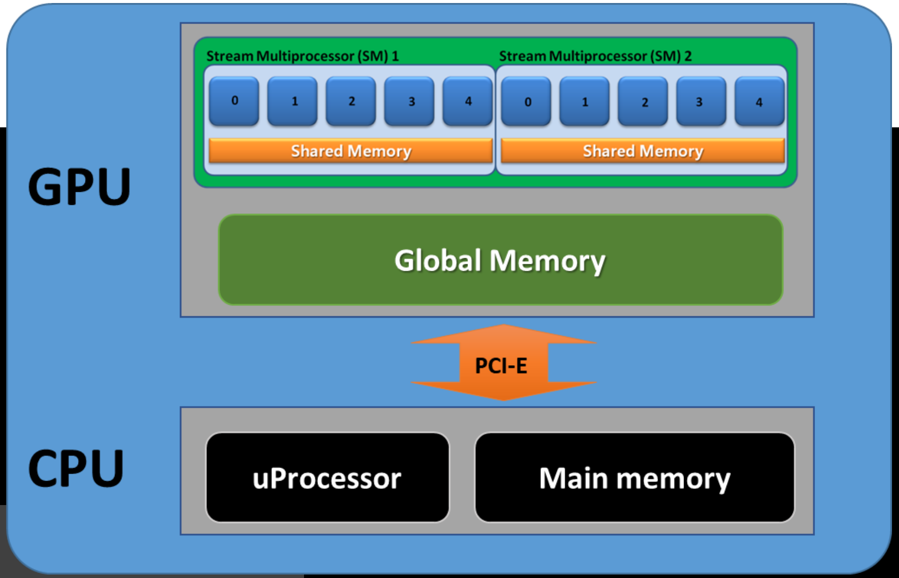
CUDA 실습하자!

```
cudaMalloc((void **)&device, n*sizeof(int));
```




```
cudaMemcpy(device, host, n*sizeof(int),cudaMemcpyHostToDevice);
cudaMemcpy(host, device, n*sizeof(int),cudaMemcpyDeviceToHost);
```

CUDA 실습하자!

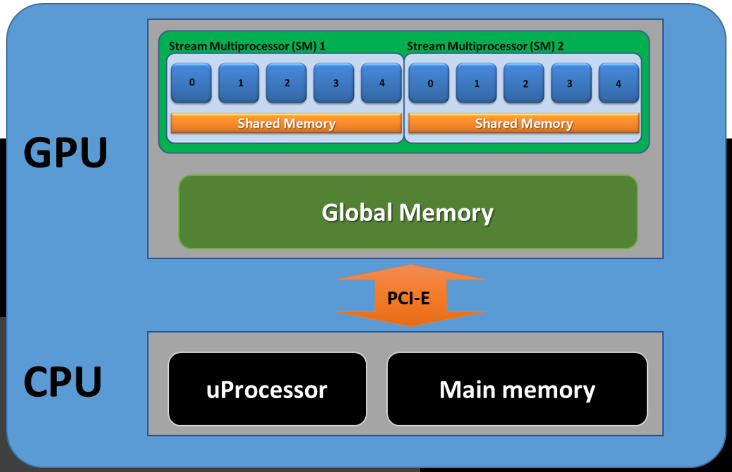



```
cudaMalloc((void **)&device, n*sizeof(int));
```

```
int *host_array;
int *dev_array;

host_array = (int *) malloc(sizeof(int)*16);
cudaMalloc(&dev_array, sizeof(int)*16);
cudaMemset(dev_array, 0, 16);
```

```
cudaMemcpy(device, host, n*sizeof(int),cudaMemcpyHostToDevice);
cudaMemcpy(host, device, n*sizeof(int),cudaMemcpyDeviceToHost);
```

CUDA 실습하자!



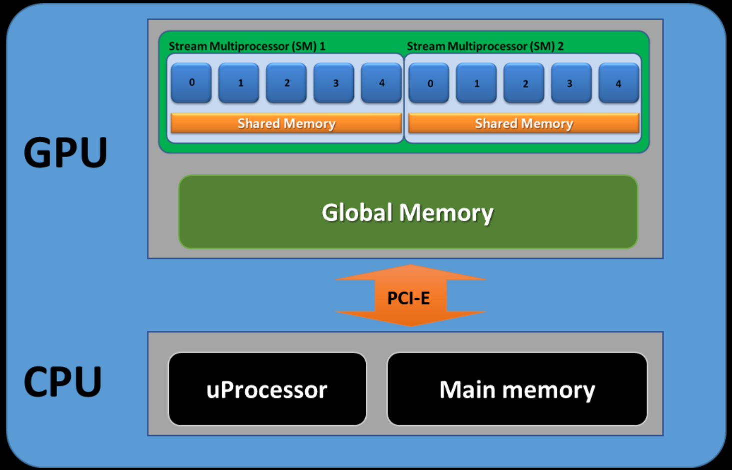

```
cudaMalloc((void **)&device, n*sizeof(int));
```

```
__global__ void kernel1( int *a )
{
   int idx = blockIdx.x*blockDim.x + threadIdx.x;
   a[idx] = 7;          // output: 7 7 7 7   7 7 7 7   7 7 7 7   7 7 7 7
}

__global__ void kernel2( int *a )
{
 int idx = blockIdx.x*blockDim.x + threadIdx.x;
   a[idx] = blockIdx.x; // output: 0 0 0 0   1 1 1 1   2 2 2 2   3 3 3 3
}

__global__ void kernel3( int *a )
{
 int idx = blockIdx.x*blockDim.x + threadIdx.x;
   a[idx] = threadIdx.x;        // output: 0 1 2 3   1 2 3 4   0 1 2 3   0 1 2 3
}
```

```
cudaMemcpy(device, host, n*sizeof(int),cudaMemcpyHostToDevice);
cudaMemcpy(host, device, n*sizeof(int),cudaMemcpyDeviceToHost);
```

CUDA 실습하자!

```c
#include <stdio.h>
#include <cuda.h>

int *host_A, *host_C1, *host_C2;        // host data
int *device_A, *device_C;    // results

__global__ void vecAddOne(int *A, int *C, int N)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;

   if( i < N )
      C[i] = A[i] + 1;
}

void vecAddOne_h(int *A1, int *C1, int N)
{
   for(int i=0;i<N;i++)
      C1[i] = A1[i] + 1;
}

int main(int argc,char **argv)
{
   int n=1024*1024;
   int nBytes = n*sizeof(int);
   int block_size = 32, block_no = n / block_size;

   // ================================================================
   // CPU 메모리 설정
   //
   host_A = (int *)malloc(nBytes);
   host_C1 = (int *)malloc(nBytes);
   host_C2 = (int *)malloc(nBytes);

   // ================================================================
   printf("Allocating device memory on host..\n");
   cudaMalloc((void **)&device_A, n*sizeof(int));
   cudaMalloc((void **)&device_C, n*sizeof(int));

   // ================================================================
   printf("Copying to device..\n");
   cudaMemcpy(device_A, host_A, n*sizeof(int),cudaMemcpyHostToDevice);
   // ================================================================

   printf("Doing GPU Vector + 1 \n");
   vecAddOne<<<block_no,block_size>>>(device_A, device_C, n);
   cudaDeviceSynchronize();
   // ================================================================

   printf("Doing a CPU Vector add\n");
   vecAddOne_h(host_A, host_C1, n);

   cudaMemcpy(host_C2, device_C, n*sizeof(int), cudaMemcpyDeviceToHost);

   // 결과 비교
   printf("결과 비교\n");
   for(int i=0; i<n;i++)
   {
       if(host_C1[i] != host_C2[i])
       {
           printf("Something Wrong ! \n");
           break;
       }
   }
   cudaFree(device_A);
   cudaFree(device_C);
   free(host_A);
   free(host_C1);
   free(host_C2);
   return 0;
}
```



CUDA 실

```
#include <stdio.h>
#include <cuda.h>

int *host_A, *host_C1, *host_(
int *device_A, *device_C;   //

__global__ void vecAddOne(int
{
   int i = blockIdx.x * blockI

   if( i < N )
      C[i] = A[i] + 1;
}

void vecAddOne_h(int *A1, int
{
   for(int i=0;i<N;i++)
      C1[i] = A1[i] + 1;
}

int main(int argc,char **argv)
{
   int n=1024*1024;
   int nBytes = n*sizeof(int).
   int block_size = 32, block_

   // ========================
   // CPU 메모리 설정
   //
   host_A = (int *)malloc(nByt
   host_C1 = (int *)malloc(nBy
   host_C2 = (int *)malloc(nBy

   // ========================
   printf("Allocating device r
   cudaMalloc((void **)&device
   cudaMalloc((void **)&device
```

```
================

ncpyHostToDevice);
================

:e_C, n);

================

MemcpyDeviceToHost);
```

CUDA 실습하자!

https://github.com/inducer/pycuda

PyCUDA

In a .bashrc file

```
export PATH="/usr/local/cuda-10.2/bin:$PATH"
export LD_LIBRARY_PATH="/usr/local/cuda-10.2/lib64:$LD_LIBRARY_PATH"

export CPATH=$CPATH:/usr/local/cuda-10.0/targets/aarch64-linux/include
export LIBRARY_PATH=$LIBRARY_PATH:/usr/local/cuda-10.0/targets/aarch64-linux/lib
```

```
pip3 install pycuda --user
```

CUDA 실습하자!


```
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule
import numpy

mod = SourceModule("""
  #include <stdio.h>

  __global__ void hellocuda()
  {
    printf("I am tx:%d.ty:%d bx:%d.by:%d\\n",
                                      threadIdx.x, threadIdx.y, blockIdx.x, blockIdx.y);
  }
  """)

func = mod.get_function("hellocuda")
func(block=(1,1,1), grid=(1,1))

# Flush context printf buffer
cuda.Context.synchronize()
```

CUDA 실습하자!


```python
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule
import numpy

mod = SourceModule("""
  __global__ void doublify(float *a)
  {
    int idx = threadIdx.x + threadIdx.y*4;
    a[idx] *= 2;
  }
  """)
```

```python
a = numpy.random.randn(4,4)
a = a.astype(numpy.float32)
a_gpu = cuda.mem_alloc(a.nbytes)
cuda.memcpy_htod(a_gpu, a)

func = mod.get_function("doublify")
func(a_gpu, block=(4,4,1))

a_doubled = numpy.empty_like(a)
cuda.memcpy_dtoh(a_doubled, a_gpu)
print(a_doubled)
print(a)
```

CUDA 실습하자!


```python
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule
import numpy

mod = SourceModule("""
  __global__ void doublify(float *a)
  {
    int idx = threadIdx.x + blockIdx.x*blockDim.x;
    int idy = threadIdx.y + blockIdx.y*blockDim.y;
    int id = idx + idy*4;
    a[id] *= 2;
  }
  """)
# create two timers so we can speed-test
start = cuda.Event()
end = cuda.Event()

a = numpy.random.randn(4,4)
a = a.astype(numpy.float32)
a_doubled = numpy.empty_like(a)
```

```python
start.record() # start timing
a_gpu = cuda.mem_alloc(a.nbytes)
cuda.memcpy_htod(a_gpu, a)

func = mod.get_function("doublify")
func(a_gpu, block=(2,2,1), grid=(2,2))

cuda.memcpy_dtoh(a_doubled, a_gpu)
end.record() # end timing
# calculate the run length
end.synchronize()
secs = start.time_till(end)*1e-3
print("SourceModule time and first
three results:")
print("%fs" % (secs))
print(a_doubled)
print(a)
```

딥러닝 안에서의 CUDA ?

```
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule
import numpy

mod

  {

  }
  """)
# create two timers so we can speed-test
start = cuda.Event()
end = cuda.Event()

a = numpy.random.randn(4,4)
a = a.astype(numpy.float32)
a_doubled = numpy.empty_like(a)
```

```
start.record() # start timing

# calculate the run length
end.synchronize()
secs = start.time_till(end)*1e-3
print("SourceModule time and first
three results:")
print("%fs" % (secs))
print(a_doubled)
print(a)
```

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

딥러닝 안에서의 CUDA ?

```
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule
import numpy

mod = SourceModule("""
  __global__ void doublify(float *a)
  {
    int idx = threadIdx.x + blockIdx.x*blockDim.x;
    int idy = threadIdx.y + blockIdx.y*blockDim.y;
    int id = idx + idy*4;
    a[id] *= 2;
  }
  """)
# create two timers so we can speed-test
start = cuda.Event()
end = cuda.Event()

a = numpy.random.randn(4,4)
a = a.astype(numpy.float32)
a_doubled = numpy.empty_like(a)
```

```
start.record() # start timing
a_gpu = cuda.mem_alloc(a.nbytes)
cuda.memcpy_htod(a_gpu, a)

func = mod.get_function("doublify")
func(a_gpu, block=(2,2,1), grid=(2,2))

cuda.memcpy_dtoh(a_doubled, a_gpu)
end.record() # end timing
# calculate the run length
end.synchronize()
secs = start.time_till(end)*1e-3
print("SourceModule time and first
three results:")
print("%fs" % (secs))
print(a_doubled)
print(a)
```

딥러닝 안에서의 CUDA ?

```
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule
import numpy
```

Model

```
# create two timers so we can speed-test
start = cuda.Event()
end = cuda.Event()

a = numpy.random.randn(4,4)
a = a.astype(numpy.float32)
a_doubled = numpy.empty_like(a)
```

```
start.record() # start timing
a_gpu = cuda.mem_alloc(a.nbytes)
cuda.memcpy_htod(a_gpu, a)

func = mod.get_function("doublify")
func(a_gpu, block=(2,2,1), grid=(2,2))

cuda.memcpy_dtoh(a_doubled, a_gpu)
end.record() # end timing
# calculate the run length
end.synchronize()
secs = start.time_till(end)*1e-3
print("SourceModule time and first
three results:")
print("%fs" % (secs))
print(a_doubled)
print(a)
```

딥러닝 안에서의 CUDA ?

```
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule
import numpy
```

Model.to(device)

```
# create two timers so we can speed-test
start = cuda.Event()
end = cuda.Event()

a = numpy.random.randn(4,4)
a = a.astype(numpy.float32)
a_doubled = numpy.empty_like(a)
```

```
start.record() # start timing
```

X_train = X_train.to(device)

```
func = mod.get_function("doublify")
func(a_gpu, block=(2,2,1), grid=(2,2))

cuda.memcpy_dtoh(a_doubled, a_gpu)
end.record() # end timing
# calculate the run length
end.synchronize()
secs = start.time_till(end)*1e-3
print("SourceModule time and first
three results:")
print("%fs" % (secs))
print(a_doubled)
print(a)
```

딥러닝 안에서의 CUDA ?

```python
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule
import numpy
```

Model.to(device)

```python
# create two timers so we can speed-test
start = cuda.Event()
end = cuda.Event()

a = numpy.random.randn(4,4)
a = a.astype(numpy.float32)
a_doubled = numpy.empty_like(a)
```

```python
start.record() # start timing
```

X_train = X_train.to(device)

```python
func = mod.get_function("doublify")
func(a_gpu, block=(2,2,1), grid=(2,2))
```

Y = X.detach().cpu().numpy()

```python
secs = start.time_till(end)*1e-3
print("SourceModule time and first
three results:")
print("%fs" % (secs))
print(a_doubled)
print(a)
```

Jetson Nano에서 즐기는 Deep Learning

돌려보자 Deep Learning!

- 다양한 딥러닝 모델을 Jetson에서!

https://github.com/dusty-nv/jetson-inference

Deploying Deep Learning

Welcome to our instructional guide for inference and realtime vision DNN library for NVIDIA Jetson devices. This project uses TensorRT to run optimized networks on GPUs from C++ or Python, and PyTorch for training models.

Supported DNN vision primitives include `imageNet` for image classification, `detectNet` for object detection, `segNet` for semantic segmentation, `poseNet` for pose estimation, and `actionNet` for action recognition. Examples are provided for streaming from live camera feeds, making webapps with WebRTC, and support for ROS/ROS2.

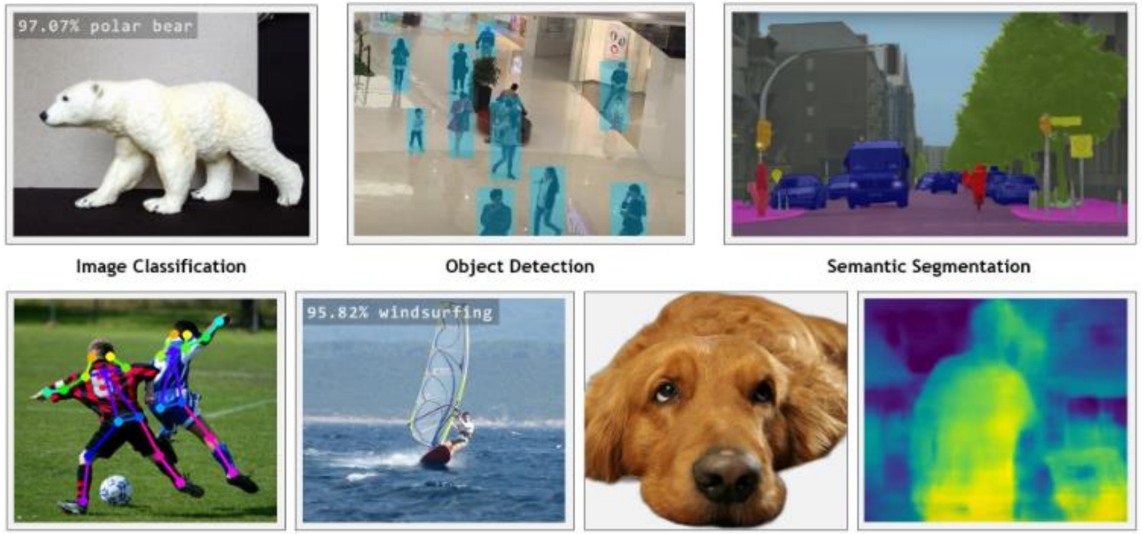


Image Classification | Object Detection | Semantic Segmentation

Pose Estimation | Action Recognition | Background Removal | Mono Depth

Follow the Hello AI World tutorial for running inference and transfer learning onboard your Jetson, including collecting your own datasets, training your own models with PyTorch, and deploying them with TensorRT.

돌려보자 Deep Learning!

- 다양한 딥러닝 모델을 Jetson에서!

https://github.com/dusty-nv/jetson-inference

Running the Docker Container

Pre-built Docker container images for this project are hosted on DockerHub. Alternatively, you can Bu

Below are the currently available container tags:

Container Tag	L4T version	JetPack version
`dustynv/jetson-inference:r35.3.1`	L4T R35.3.1	JetPack 5.1.1
`dustynv/jetson-inference:r35.2.1`	L4T R35.2.1	JetPack 5.1
`dustynv/jetson-inference:r35.1.0`	L4T R35.1.0	JetPack 5.0.2
`dustynv/jetson-inference:r34.1.1`	L4T R34.1.1	JetPack 5.0.1
`dustynv/jetson-inference:r32.7.1`	L4T R32.7.1	JetPack 4.6.1

Deploying Deep Learning

Welcome to our instructional guide for inference and realtime vision DNN library for NVIDIA Jetson devices. This project uses TensorRT to run optimized networks on GPUs from C++ or Python, and PyTorch for training models.

Supported DNN vision primitives include `imageNet` for image classification, `detectNet` for object detection, `segNet` for semantic segmentation, `poseNet` for pose estimation, and `actionNet` for action recognition. Examples are provided for streaming from live camera feeds, making webapps with WebRTC, and support for ROS/ROS2.

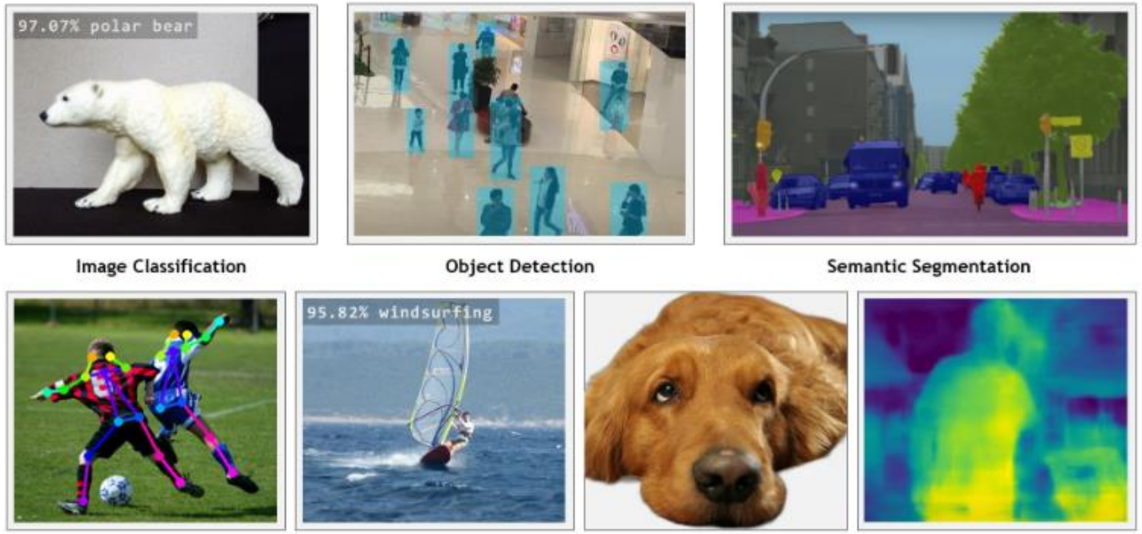


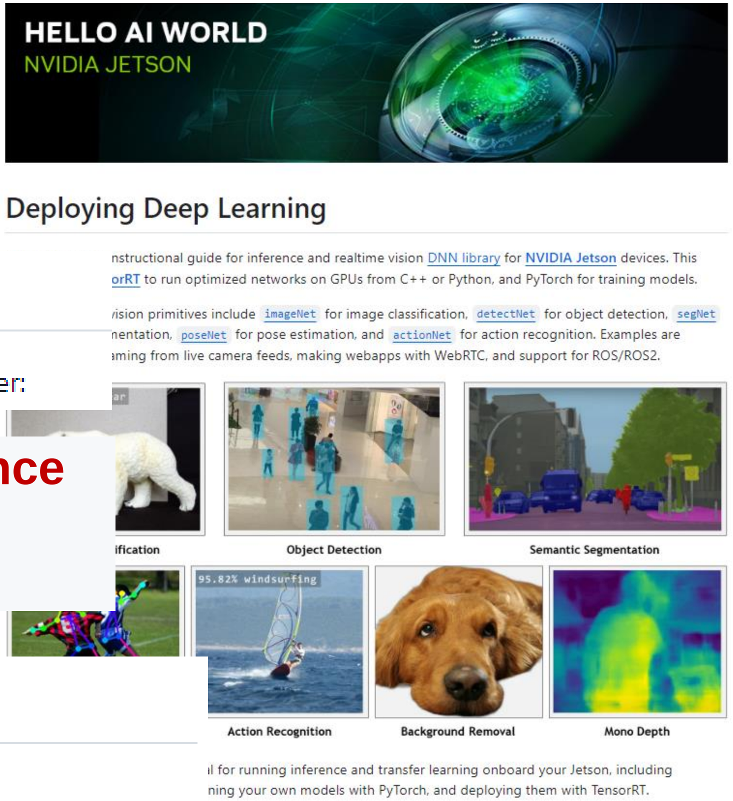
Image Classification · Object Detection · Semantic Segmentation

Pose Estimation · Action Recognition · Background Removal · Mono Depth

Follow the Hello AI World tutorial for running inference and transfer learning onboard your Jetson, including collecting your own datasets, training your own models with PyTorch, and deploying them with TensorRT.

돌려보자 Deep Learning!

- 다양한 딥러닝 모델을 Jetson에서!



Launching the Container

Due to various mounts and devices needed to run the container, it's recommended to use the `docker/run.sh` script to run the container:

```
$ git clone --recursive --depth=1 https://github.com/dusty-nv/jetson-inference
$ cd jetson-inference
$ docker/run.sh
```

Running Applications

Once the container is up and running, you can then run example programs from the tutorial like normal inside the container:

```
$ cd build/aarch64/bin
$ ./video-viewer /dev/video0
$ ./imagenet images/jellyfish.jpg images/test/jellyfish.jpg
$ ./detectnet images/peds_0.jpg images/test/peds_0.jpg
# (press Ctrl+D to exit the container)
```

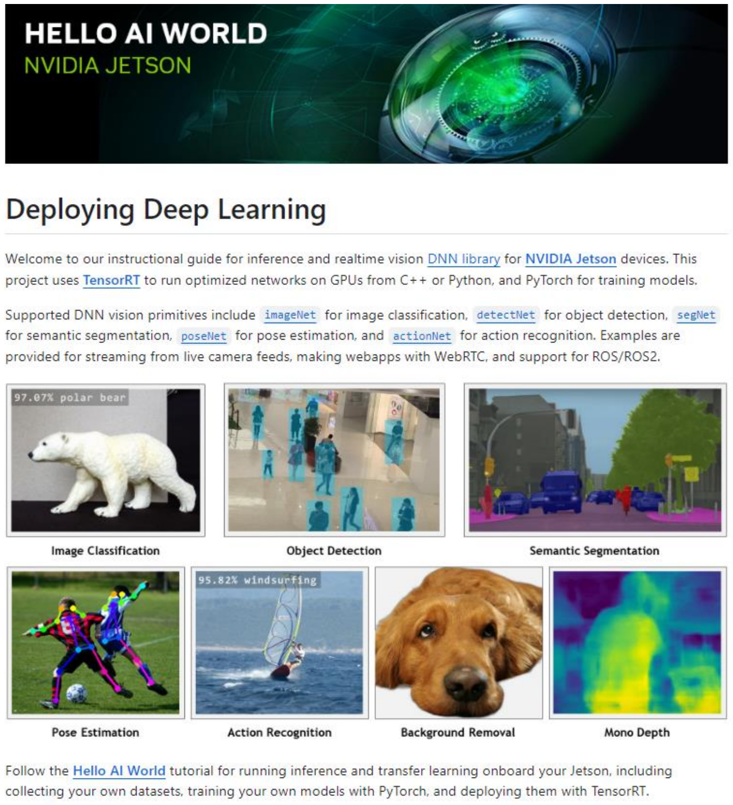
> **note:** when you are saving images from one of the sample programs (like imagenet or detectnet), it's recommended to save them to `images/test`. These images will then be easily viewable from your host device in the `jetson-inference/data/images/test` directory.

https://github.com/dusty-nv/jetson-inference/blob/master/docs/aux-docker.md

돌려보자 Deep Learning!

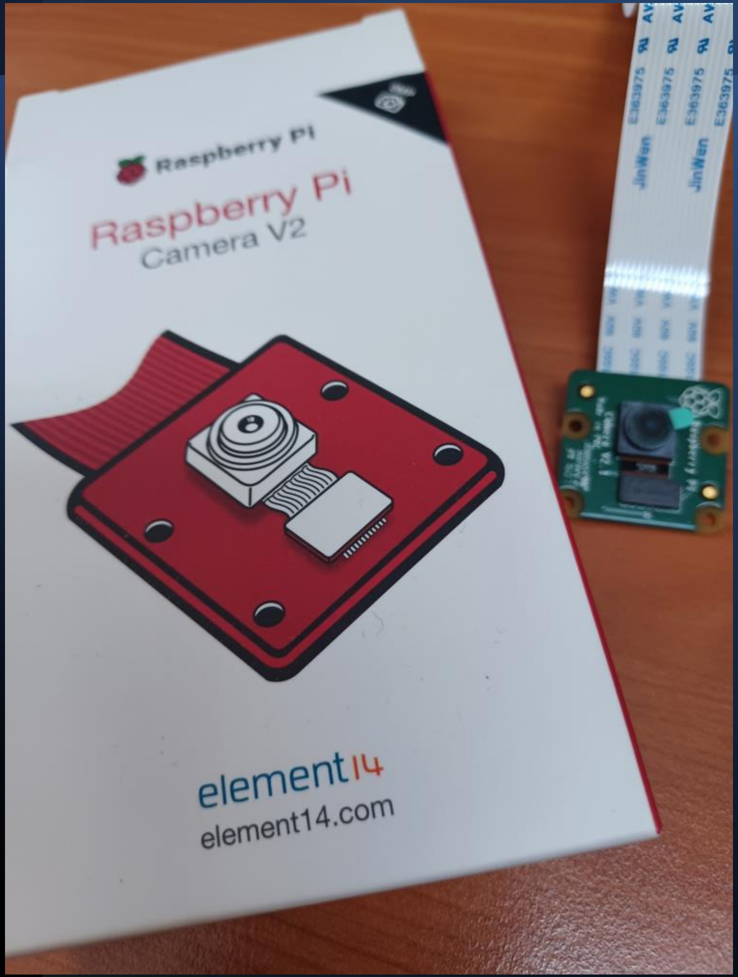
- **Building the Project from Source**

```
sudo apt-get update
sudo apt-get install git cmake libpython3-dev python3-numpy
git clone --recursive --depth=1 https://github.com/dusty-nv/jetson-inference
cd jetson-inference
mkdir build
cd build
cmake ../
make -j$(nproc)
sudo make install
sudo ldconfig
```



https://github.com/dusty-nv/jetson-inference/blob/master/docs/building-repo-2.md

돌려보자 Deep Learning!




```
$ ./detectnet.py csi://0 # MIPI CSI camera
$ ./detectnet.py /dev/video0 # V4L2 camera
$ ./detectnet.py /dev/video0 output.mp4 # save to video file
```