CUDA 프로그래밍

CUDA Programming

biztripcru@gmail.com

Memory Copy

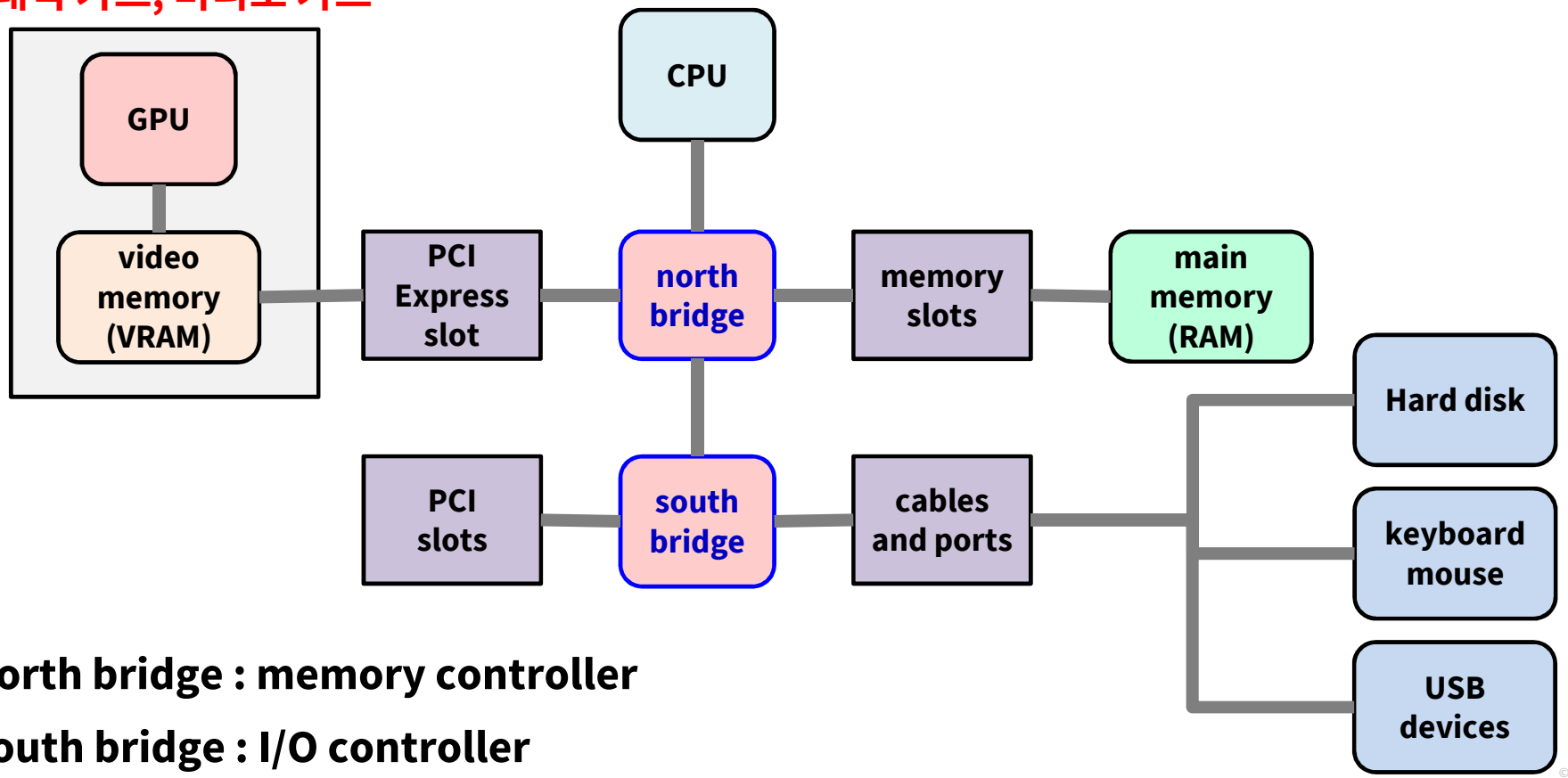
메모리 복사

내용 contents

- **CUDA 프로그래밍 모델**
  - PC 구조
  - CUDA 컴파일러 구조
- **CUDA 프로그램 시나리오**
  - host (CPU) → device (CUDA, GPU)
  - device 처리
  - device → host
- **실습: memcpy.cu**

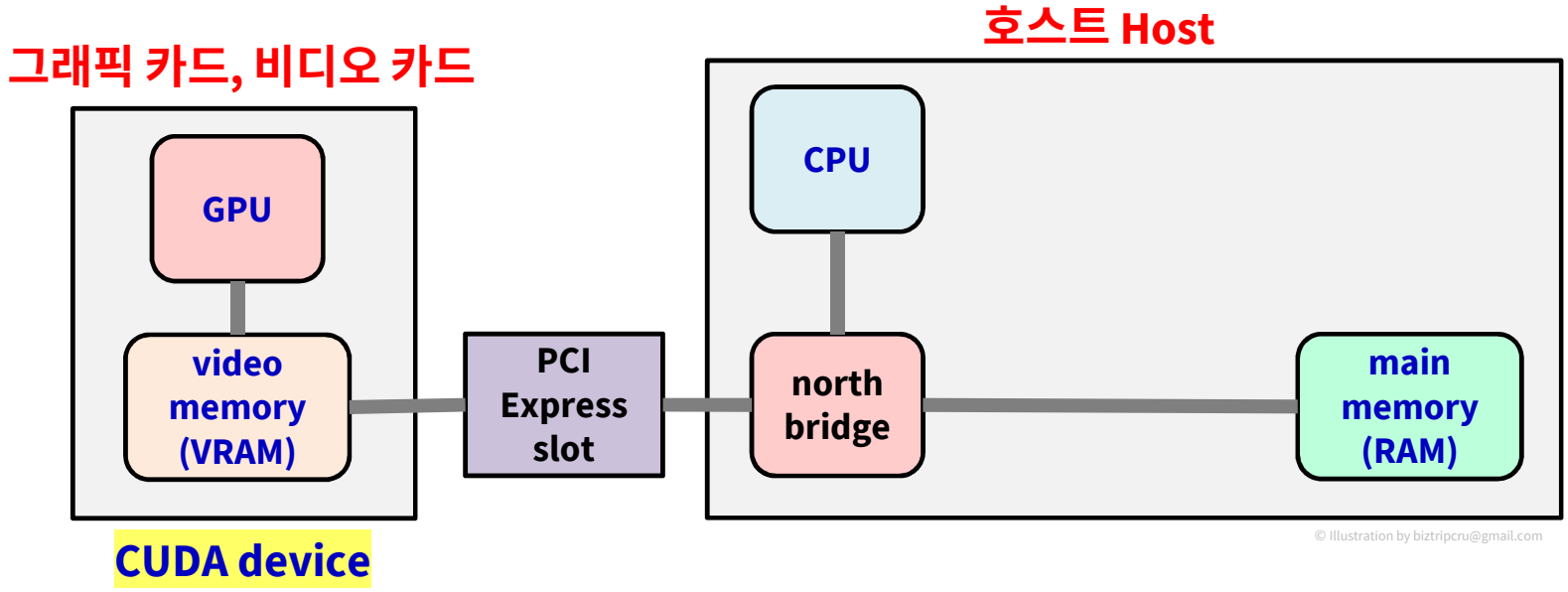
PC 구조

그래픽 카드, 비디오 카드



- north bridge : memory controller
- south bridge : I/O controller

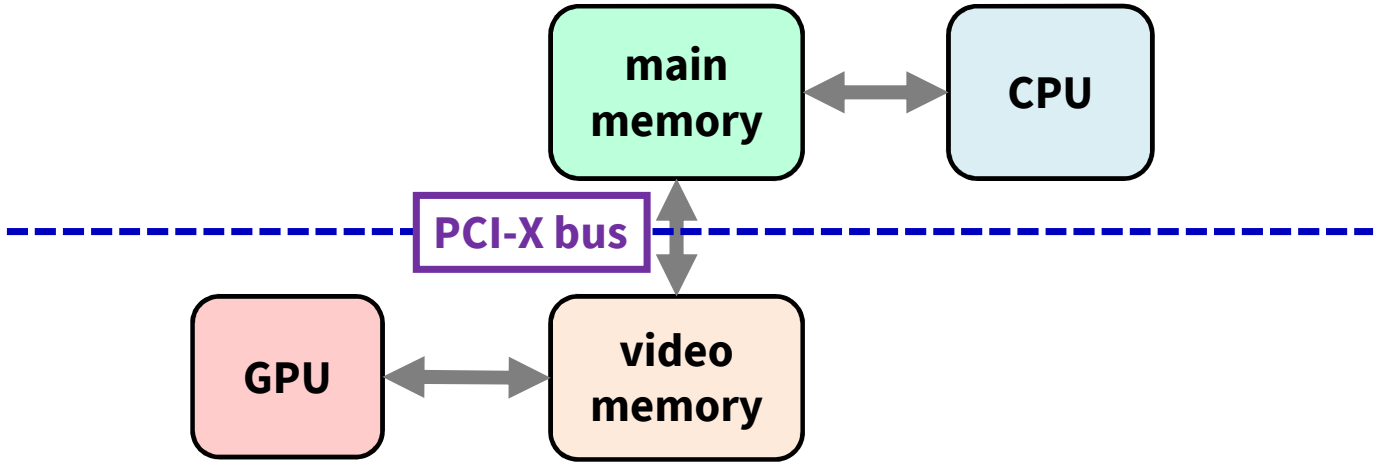
PC 구조 간략화



- CPU 쪽: main memory, RAM, CPU memory, host memory
- GPU 쪽: video memory, graphics memory, VRAM (Video RAM), GPU memory, CUDA memory, device memory

CUDA 프로그래밍 모델

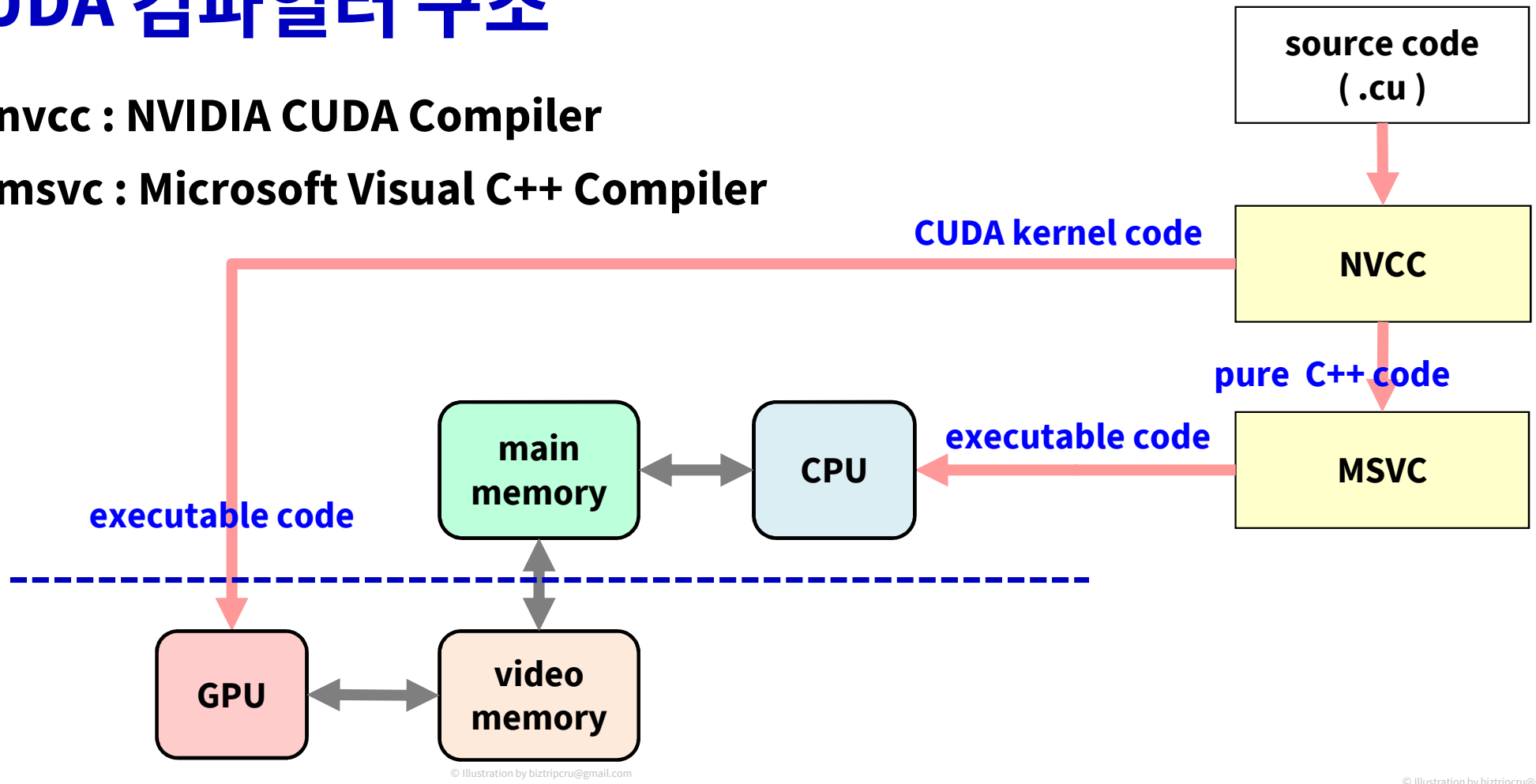
- 호스트 host : CPU + main memory (메인 메모리)
- 디바이스 device : GPU + video memory (비디오 메모리)
  - CPU ↔ video memory 직접 연결 불가
  - GPU ↔ main memory 직접 연결 불가



© Illustration by biztripcru@gmail.com

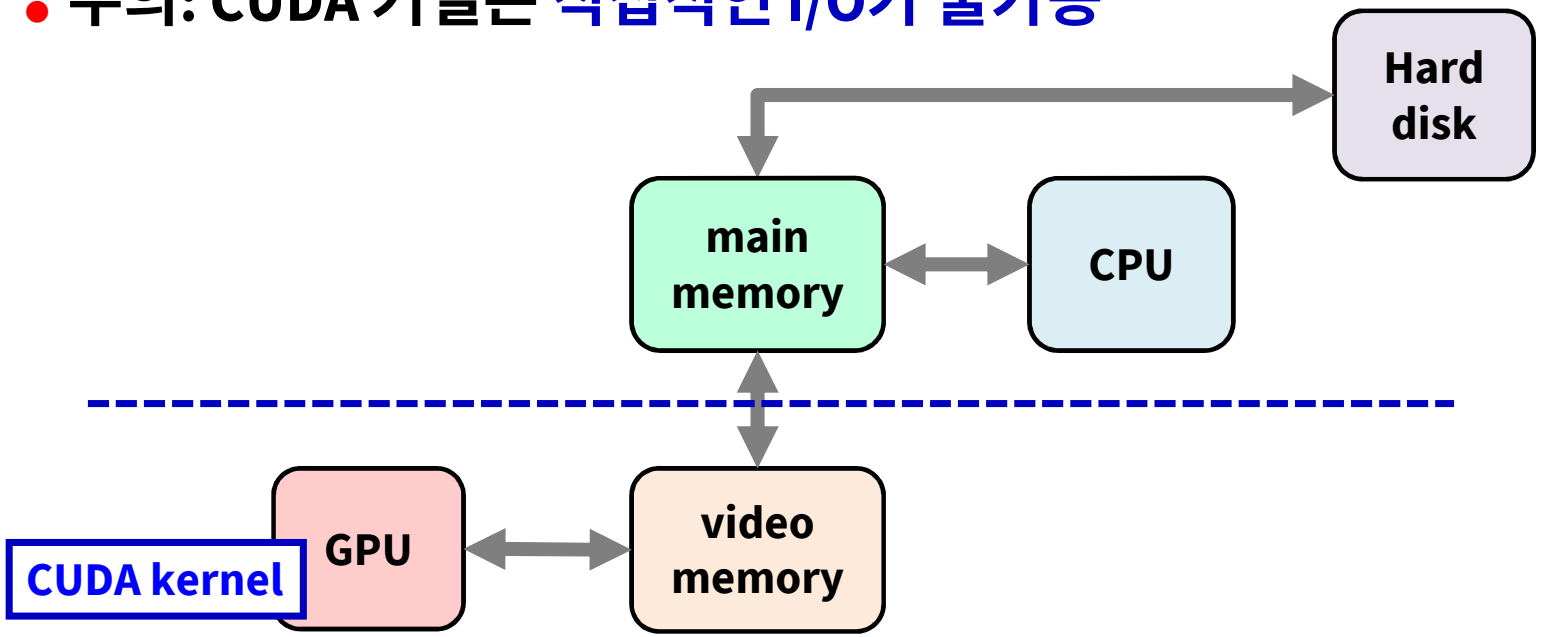
CUDA 컴파일러 구조

- nvcc : NVIDIA CUDA Compiler
- msvc : Microsoft Visual C++ Compiler



CUDA 컴파일러 구조 계속

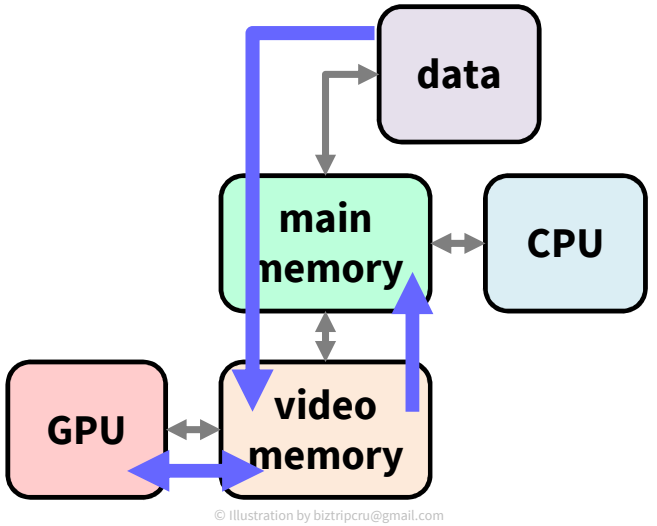
- CUDA 커널 kernel : GPU가 실행하는 작은 (병렬) 프로그램
- CUDA 가 사용하는 메모리 : video memory
- 주의: CUDA 커널은 직접적인 I/O가 불가능



© Illustration by biztripcru@gmail.com

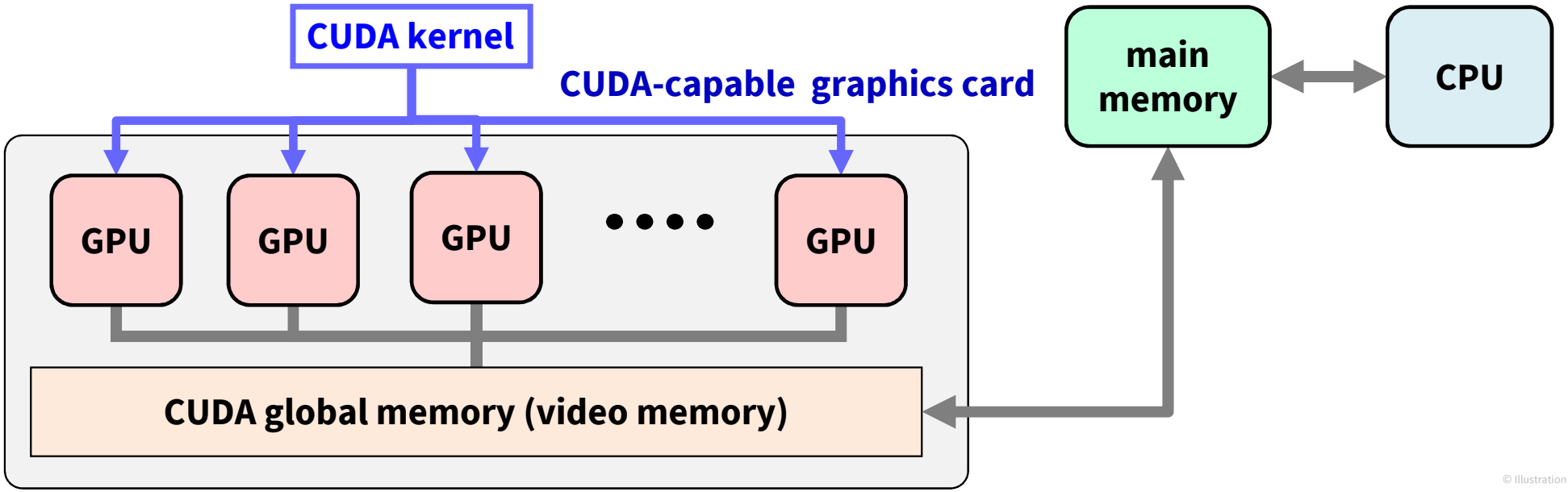
CUDA 프로그램 시나리오

- **step 1. 호스트host CPU**
  - 외부 데이터 → 메인 메모리
  - 메인 메모리 → 비디오 메모리
- **step 2. 커널 프로그램 (디바이스 device)**
  - CUDA 커널kernel 실행
  - 비디오 메모리의 데이터를 사용
  - 비디오 메모리 ↔ GPU로 병렬 처리
  - 처리 결과는 비디오 메모리에
- **step 3. 호스트host CPU**
  - 비디오 메모리 → 메인 메모리
  - 외부로 보내거나, I/O 출력



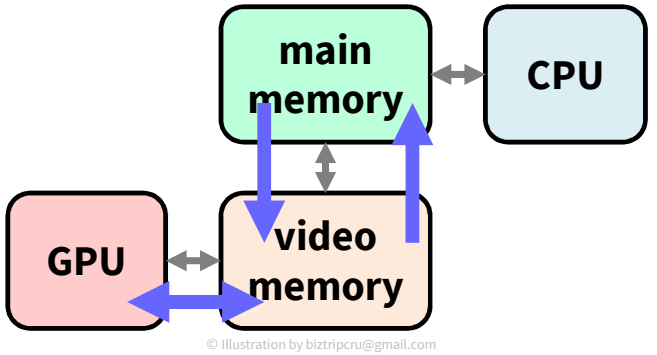
CUDA 그래픽 카드 내부 모델

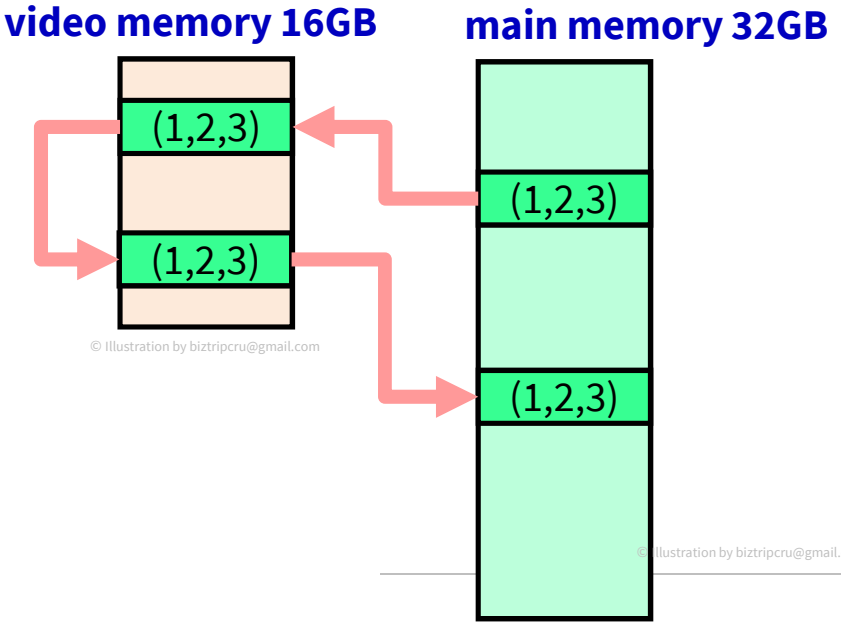
- **CUDA 그래픽 카드 = 여러 개의 GPU 프로세서 + 글로벌 메모리 (video memory)**
  - 자세한 구조는 나중에 설명
  - GPU 프로세서들이 글로벌 메모리global memory를 공유



Example: Host-Device Memory copy

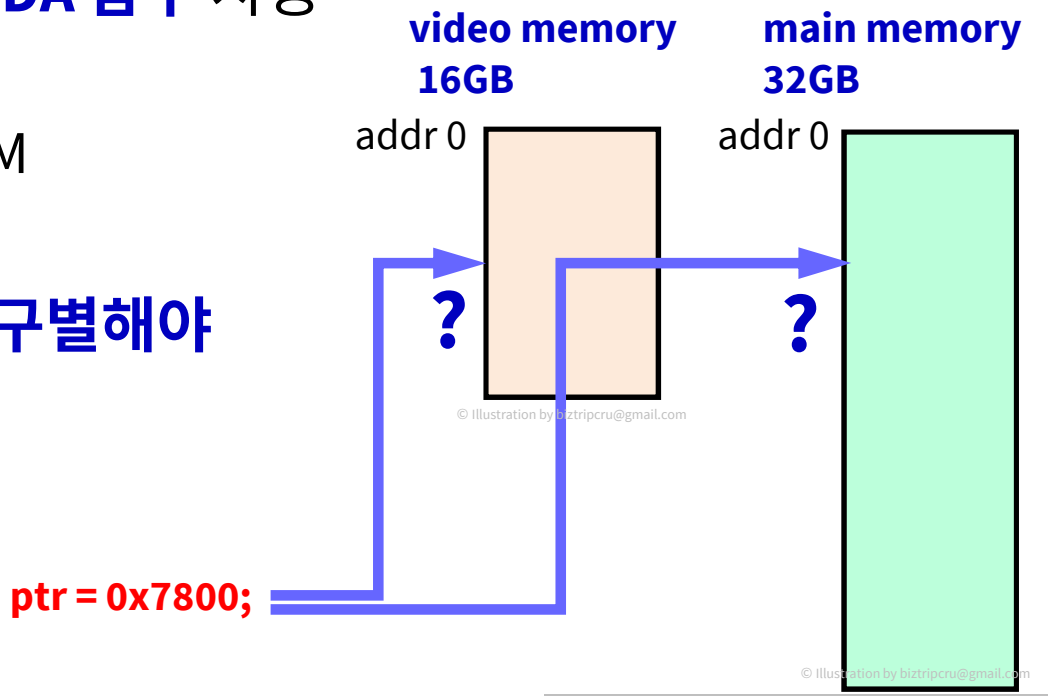
- **step 1. 호스트host CPU**
  - make the source data
  - print out the source data
  - main memory → video memory
- **step 2. 커널 프로그램 (디바이스)**
  - video memory → video memory
- **step 3. 호스트host CPU**
  - video memory → main memory
  - print out the result





분리된 메모리 공간 memory space

- **CPU, GPU 메모리 공간은 서로 분리됨**
  - **메인 메모리** 할당/복사 : **C++ 함수** 사용
  - **비디오 메모리** 할당/복사 : **별도의 CUDA 함수** 사용
  - host = CPU, main memory, RAM
  - device = CUDA, video memory, VRAM
- **메모리 어드레스 memory address 문제**
  - **어느 쪽 어드레스인지 프로그래머가 구별해야**
  - 반대쪽 어드레스를 넣으면
    시스템 크래쉬crash 발생 가능
  - 해결책: device 에는 "**dev_**" 사용

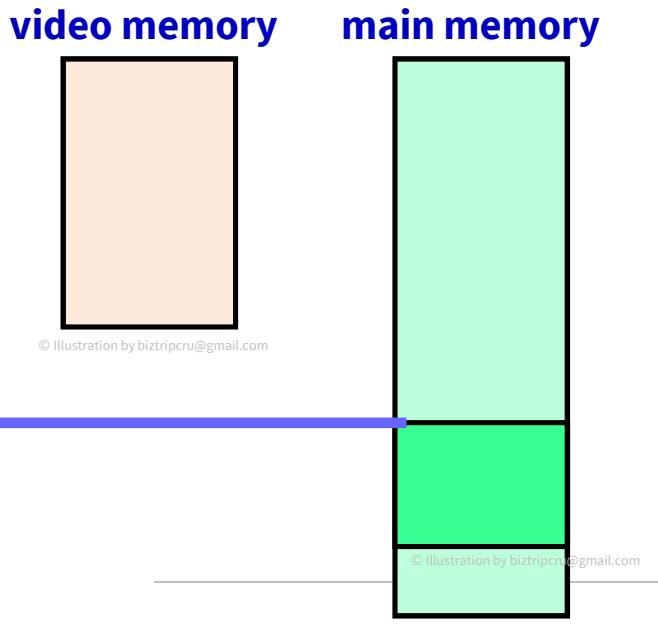


C/C++ 메모리 관련 함수

- **void* malloc(size_t *nbytes*);**
- **void free(void* *ptr*);**
- **void* memset(void* *ptr*, int *value*, size_t *count*);**
- **void* memcpy(void* *dst*, const void* *src*, size_t *num*);**

nbyte ← number of bytes
ptr ← pointer 포인터
dst ← destination 목적지
src ← source 원천
num ← number (of …)

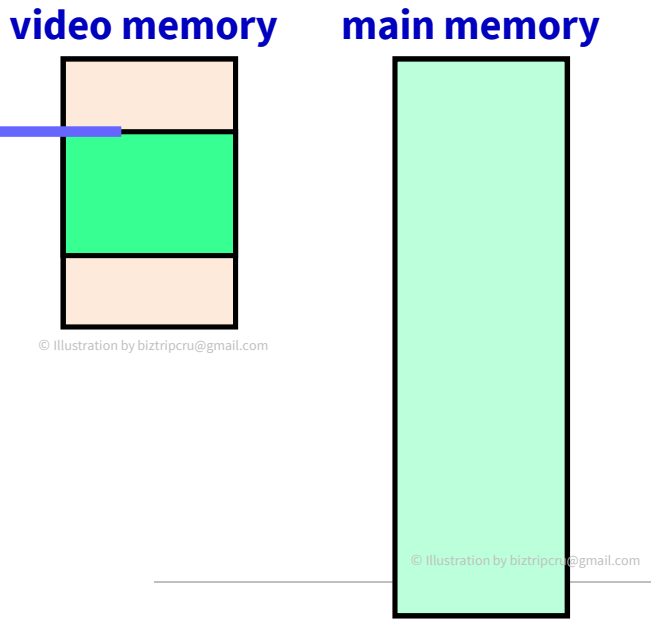
```
int  nbytes = 1024 * sizeof(int);
int* ptr =  nullptr;
ptr = malloc( nbytes );
memset( ptr, 0, nbytes );
free( ptr );
```



CUDA 메모리 관련 함수

- **cudaError_t cudaMalloc(void** *dev_ptr*, size_t *nbytes*);**
- **cudaError_t cudaMemset(void* *dev_ptr*, int *value*, size_t *count*);**
- **cudaError_t cudaFree(void* *dev_ptr*);**

```
int  nbytes = 1024 * sizeof(int);
int* dev_ptr =  nullptr;
cudaMalloc( (void**)&dev_ptr,  nbytes );
cudaMemset( dev_ptr, 0, nbytes );
cudaFree( dev_ptr );
```



CUDA function rules

- 모든 CUDA 함수는 “cuda”로 시작
- 대부분은 에러 코드error code를 리턴 (성공시는 cudaSuccess).

- **Example:**

```
if (cudaMalloc( &devPtr, SIZE ) != cudaSuccess) {
   exit(1);
}
```

CUDA malloc

- **cudaError_t cudaMalloc(void** *devPtr*, size_t *nbytes*);**
  - CUDA 메모리 (video memory) 영역에 *nbyte*를 할당
  - 시작 주소는 *devPtr*에 저장
  - **메모리는 clear 하지 않음**
  - 리턴 값: cudaSuccess, cudaErrorMemoryAllocation

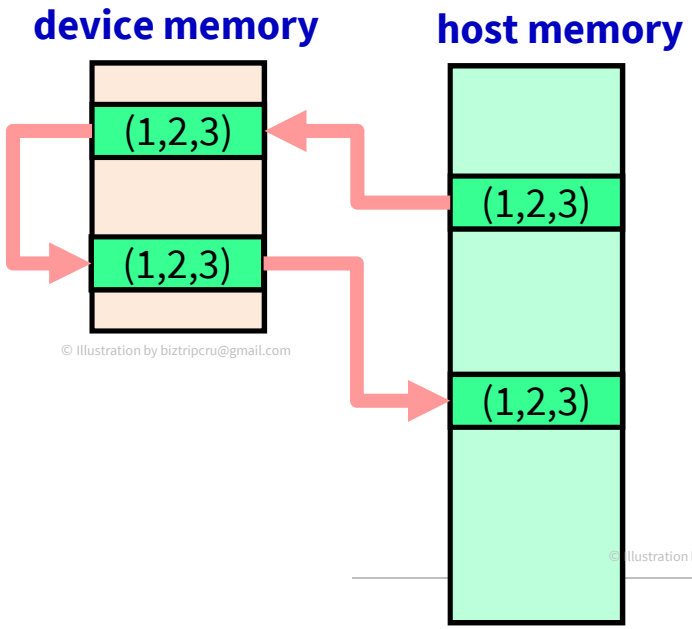
- **cudaError_t cudaFree(void* *devPtr*);**
  - *devPtr*이 가리키는 메모리를 반환
  - **devPtr == nullptr 또는 devPtr == 0 인 경우는 무시**
  - 리턴 값: cudaSuccess, cudaErrorInvalidDevicePointer

CUDA memset

- **cudaError_t cudaMemset(void* *devPtr*, int *value*, size_t *nbytes*);**
  - *devPtr*이 가르키는 영역의 최초 *nbytes*를 *value*로 설정
    - byte 단위로 설정됨
    - *value* = 0 → 모든 byte 가 0 으로 (clear)
    - *value* = 0x77 → 모든 byte 가 0x77, 4byte는 0x77777777 로 설정
  - 리턴 값: cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidDevicePointer

CUDA memcpy

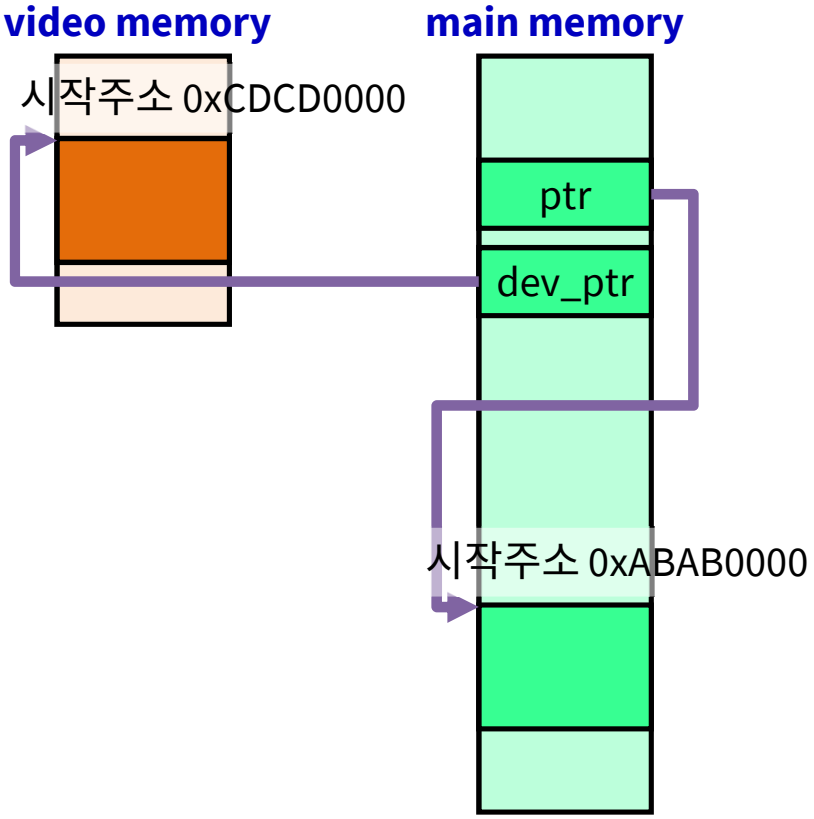
- **cudaError_t cudaMemcpy(void* *dst*, void* *src*, size_t *nbytes*, enum cudaMemcpyKind *direction*);**
  - 이전 CUDA 함수들이 모두 종료되어야 복사가 시작됨
  - copy 중에는 CPU 쓰레드도 정지, 작업이 완료되어야 리턴
  - host = CPU, main memory, RAM
  - device = CUDA, video memory, VRAM
  - enum cudaMemcpyKind
    - cudaMemcpyHostToDevice
    - cudaMemcpyDeviceToHost
    - cudaMemcpyDeviceToDevice
    - cudaMemcpyHostToHost



메모리 공간

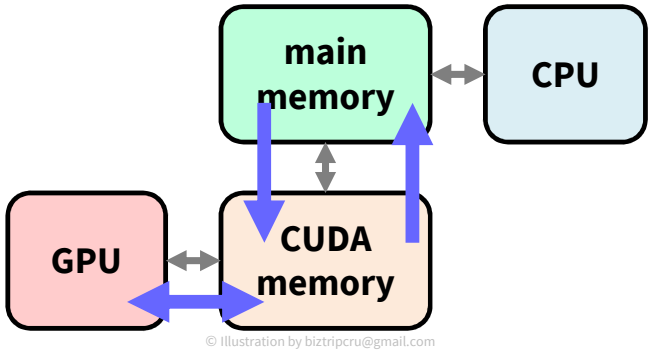
- **포인터 변수 : main memory에 위치**
  - int* ptr = nullptr;
  - int* dev_ptr = nullptr;

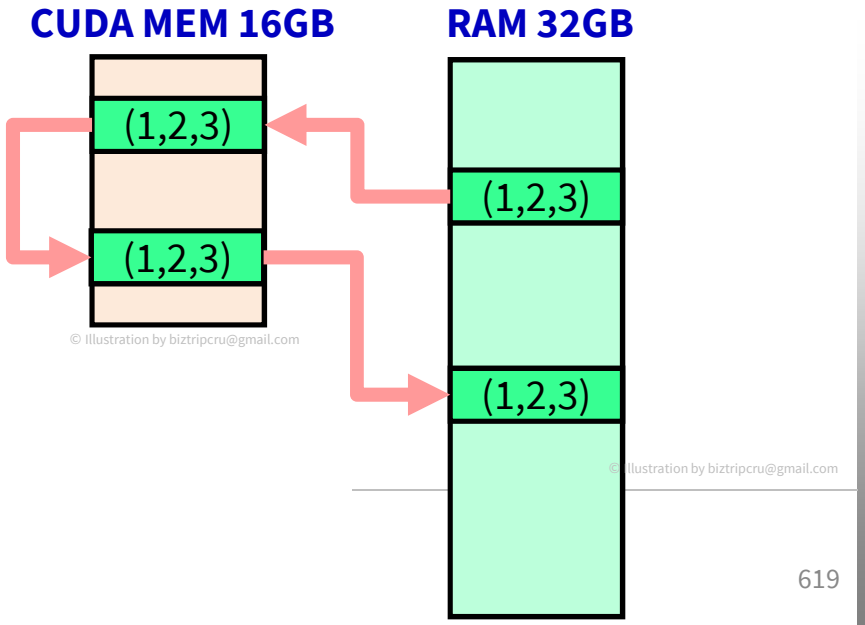
- **할당 받는 공간은 함수 별로 다름**
  - ptr = malloc(nbytes);
  - cudaMalloc((void**)&dev_ptr, nbytes);



Example: Host-Device Memory copy

- **step 1. 호스트host CPU**
  - make the source data
  - print out the source data
  - main memory → CUDA memory
- **step 2. 커널 프로그램 (디바이스)**
  - CUDA memory → CUDA memory
- **step 3. 호스트host CPU**
  - CUDA memory → main memory
  - print out the result

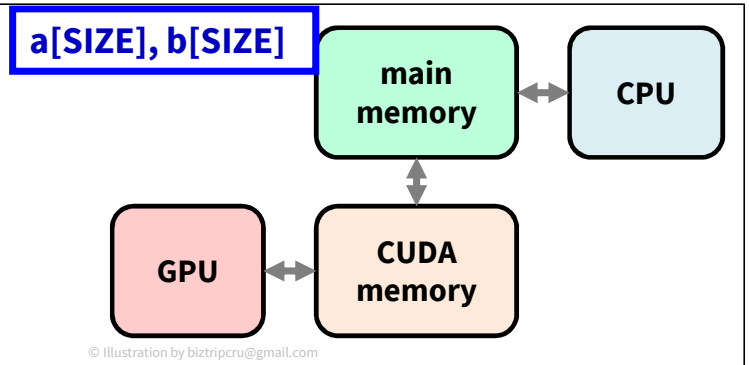




Program: memcpy.cu

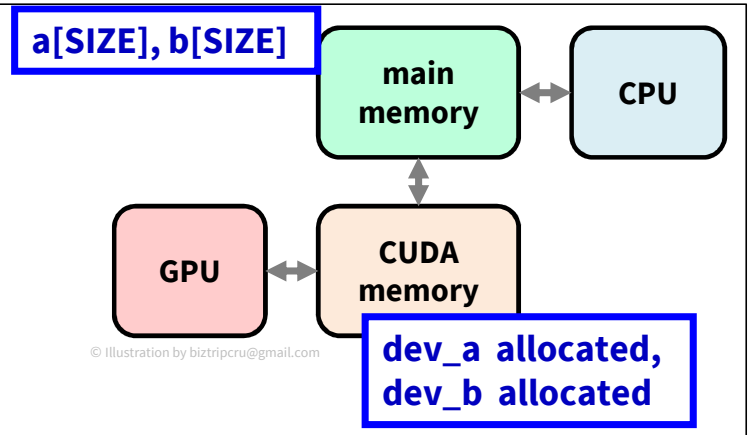
```
#include <stdio.h>

int main(void) {
  // host-side data
  const int SIZE = 8;
  const float a[SIZE] = { 1., 2., 3., 4., 5., 6., 7., 8. };
  float b[SIZE] = { 0., 0., 0., 0., 0., 0., 0., 0. };
  // print source
  printf("a = {%f,%f,%f,%f,%f,%f,%f,%f}\n", a[0], a[1], a[2], a[3], a[4], a[5], a[6], a[7]);
  fflush( stdout );
```



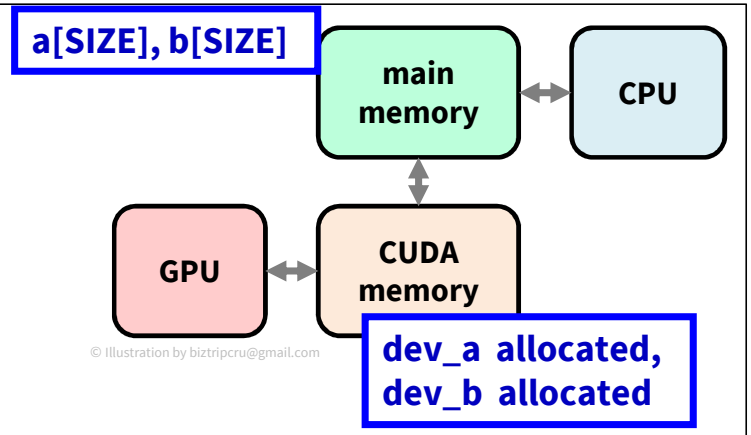
Program: memcpy.cu

```
// device-side data
float* dev_a = nullptr;
float* dev_b = nullptr;
// allocate device memory
cudaMalloc( (void**)&dev_a, SIZE * sizeof(float) );
cudaMalloc( (void**)&dev_b, SIZE * sizeof(float) );
// 3 copies
cudaMemcpy( dev_a, a, SIZE * sizeof(float), cudaMemcpyHostToDevice); // dev_a = a;
cudaMemcpy( dev_b, dev_a, SIZE * sizeof(float), cudaMemcpyDeviceToDevice); // dev_b = dev_a;
cudaMemcpy( b, dev_b, SIZE * sizeof(float), cudaMemcpyDeviceToHost); // b = dev_b;
```



Program: memcpy.cu 계속

```
    // free device memory
    cudaFree( dev_a );
    cudaFree( dev_b );
    // print the result
    printf("b = {%f,%f,%f,%f,%f,%f,%f,%f}\n", b[0], b[1], b[2], b[3], b[4], b[5], b[6], b[7]);
    fflush( stdout );
    // done
    return 0;
}
```



Program: memcpy.cu 계속

- 실행 결과

```
const int SIZE = 8;
const float a[SIZE] = { 1., 2., 3., 4., 5., 6., 7., 8. };
float b[SIZE] = { 0., 0., 0., 0., 0., 0., 0., 0. };
```

~/secure/

```
linux/cuda-work > ./06a-memcpy.exe
a = {1.000000,2.000000,3.000000,4.000000,5.000000,6.000000,7.000000,8.000000}
b = {1.000000,2.000000,3.000000,4.000000,5.000000,6.000000,7.000000,8.000000}
linux/cuda-work >
```

내용 contents

- **CUDA 프로그래밍 모델**
  - PC 구조
  - CUDA 컴파일러 구조
- **CUDA 프로그램 시나리오**
  - host (CPU) → device (CUDA, GPU)
  - device 처리
  - device → host
- **실습: memcpy.cu**

Memory Copy

폰트 끝단 일치 → 큰 교자 타고 혼례 치른 날
정참판 양반댁 규수 큰 교자 타고 혼례 치른 날
정참판 양반댁 규수 큰 교자 타고 혼례 치른 날
본고딕 Noto Sans KR

The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog
Source Sans Pro

Mathematical Notations $O(n \log n)$
Source Serif Pro