CUDA 프로그래밍

CUDA Programming

biztripcru@gmail.com

CUDA Kernel Launch

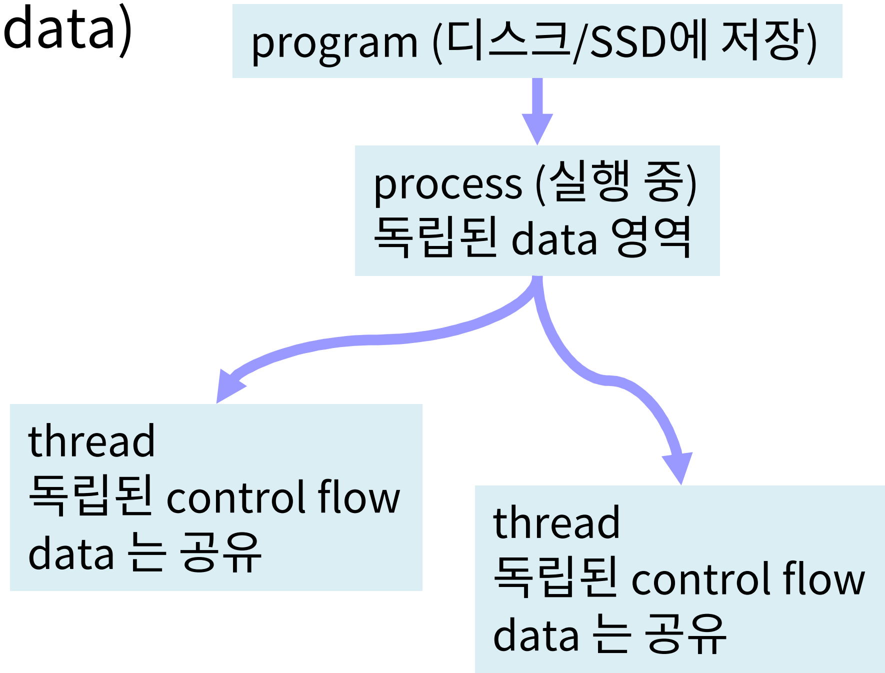
CUDA 커널 실행

내용 contents

- process and thread
- CUDA programming model
- kernel launch
- pre-defined variables
- CUDA architecture

Process and Thread

- **프로세스 computer process**
  - **an instance of a computer program** that is being executed
  - program code + current activity (or status data)
  - **독립적인 데이터 공간 확보**
- **쓰레드 thread**
  - **a control flow in a computer process**
  - the smallest sequence of instructions, that can be managed independently by a (operating system) scheduler
  - 쓰레드 끼리 **데이터 공유 data share 가능**



Thread

- **thread in real world**
  - thread 쓰레드 : 실

weft thread : warp threads 사이를 관통해서 엮이는 실 thread 1줄

pixabay license
https://pixabay.com/ko/photos/%EC%A7%81%EB%AC%BC-%ED%95%B8%EB%93%9C-%EC%B9%B4%ED%8E%AB-%EB%B2%A0%ED%8B%80-4079366/

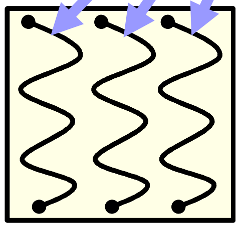
pixabay license
https://pixabay.com/photos/yarn-ball-knitting-wool-thread-731515/

warp threads : 평행하게 함께 움직이는 여러 개의 실 thread

- **thread in a computer**
  - 독립적 실행의 단위

쓰레드 thread

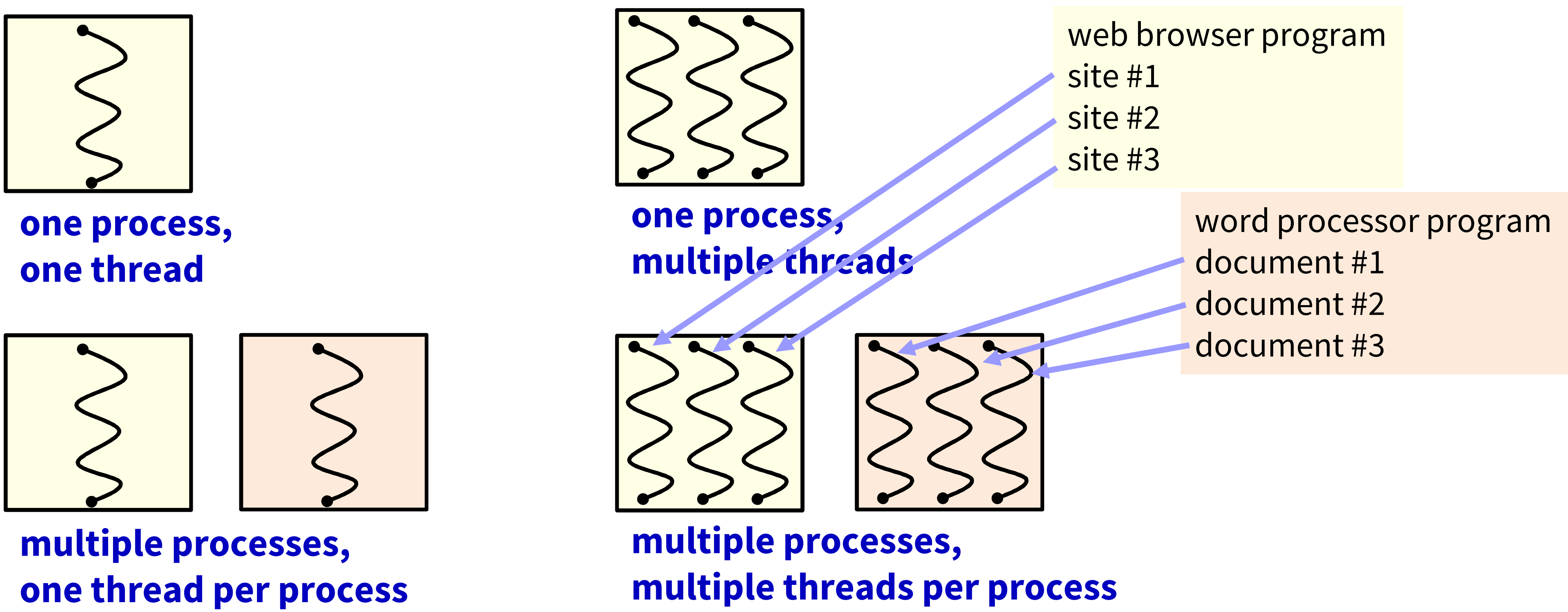
© Illustration by biztripcru@gmail.com



one process, multiple threads

O/S Supports

- process and threads are supported by **operating systems** 운영 체제



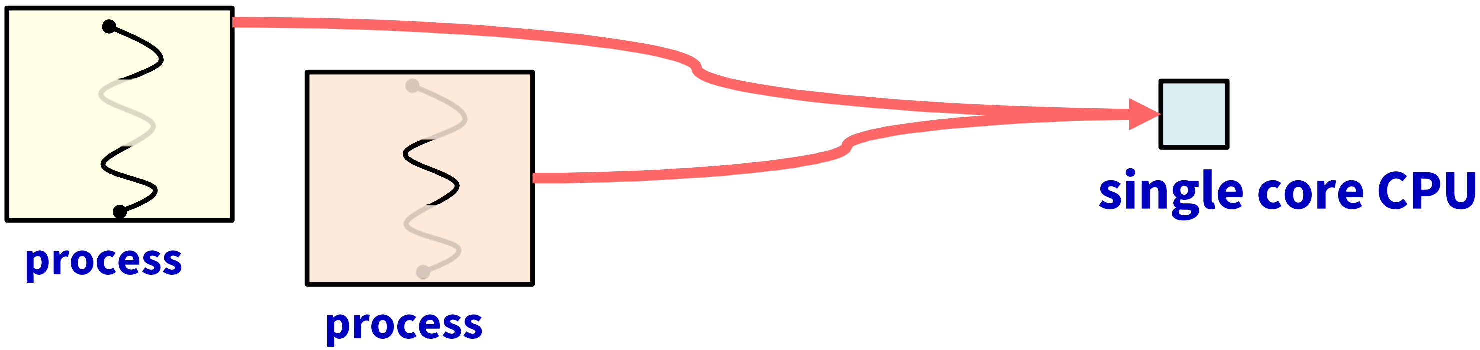
Single Core Processors

- **single thread**

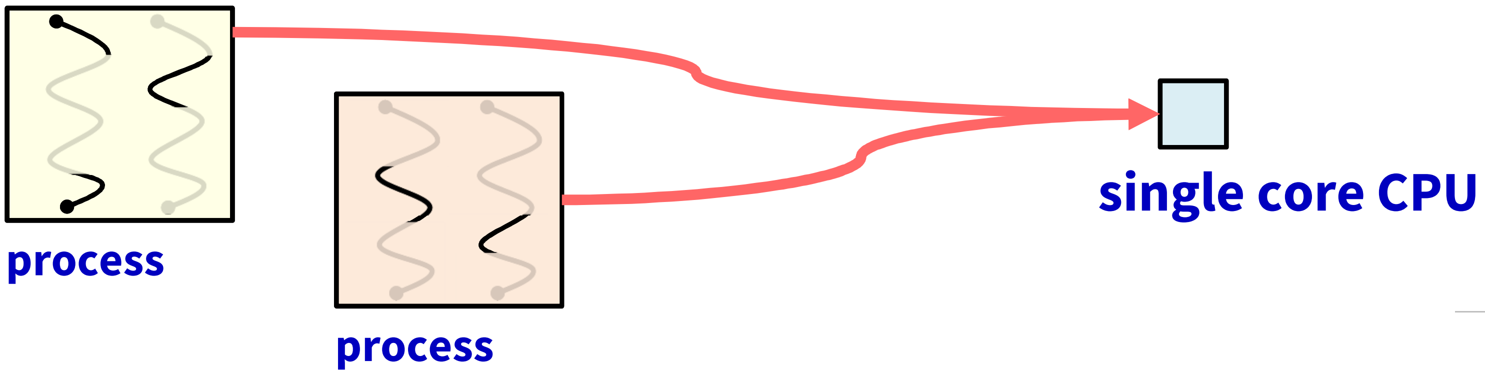
- **multiple thread → time sharing**

Time Sharing

- multiple process → time sharing

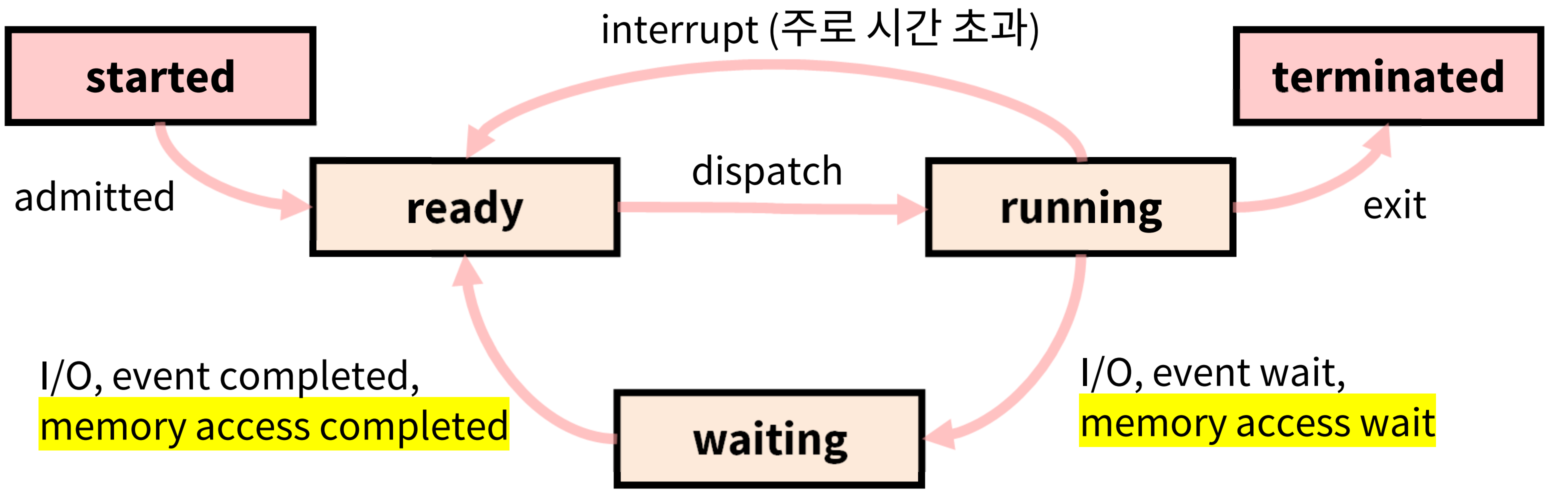


- multiple process, multiple thread → time sharing



Time Sharing

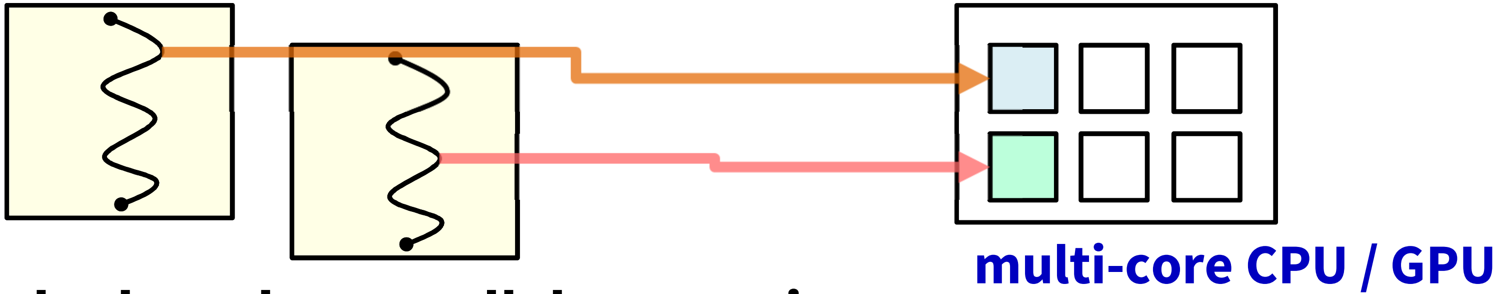
- **time sharing 시의 process / thread 실행**
- **목표: (가장 비싼 자원인) processor 가 쉬지 않도록 한다 → 최고 효율 달성**
  - 해야 할 일이 있는데, CPU가 쉬고 있으면 안된다!



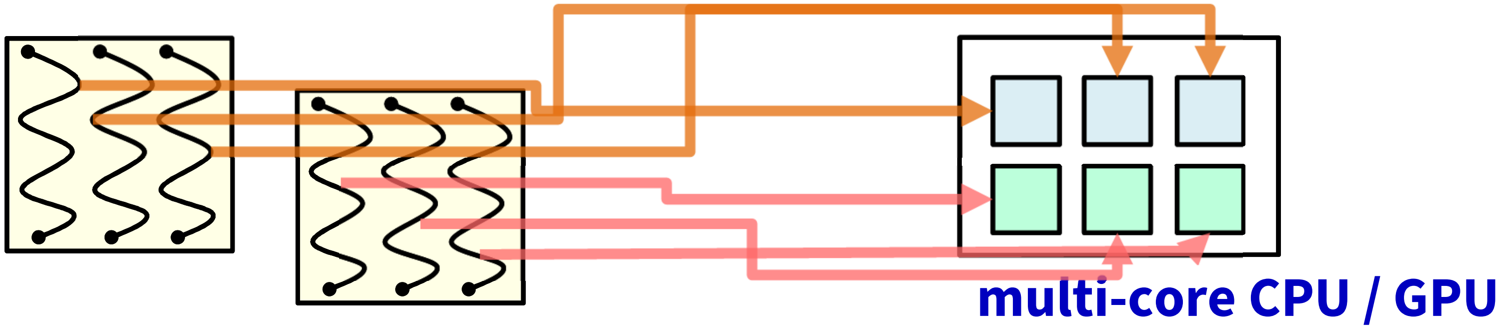
© Illustration by biztripcru@gmail.com

Multiple Core Processors

- single thread, multiple process → parallel processing



- multiple thread → parallel processing

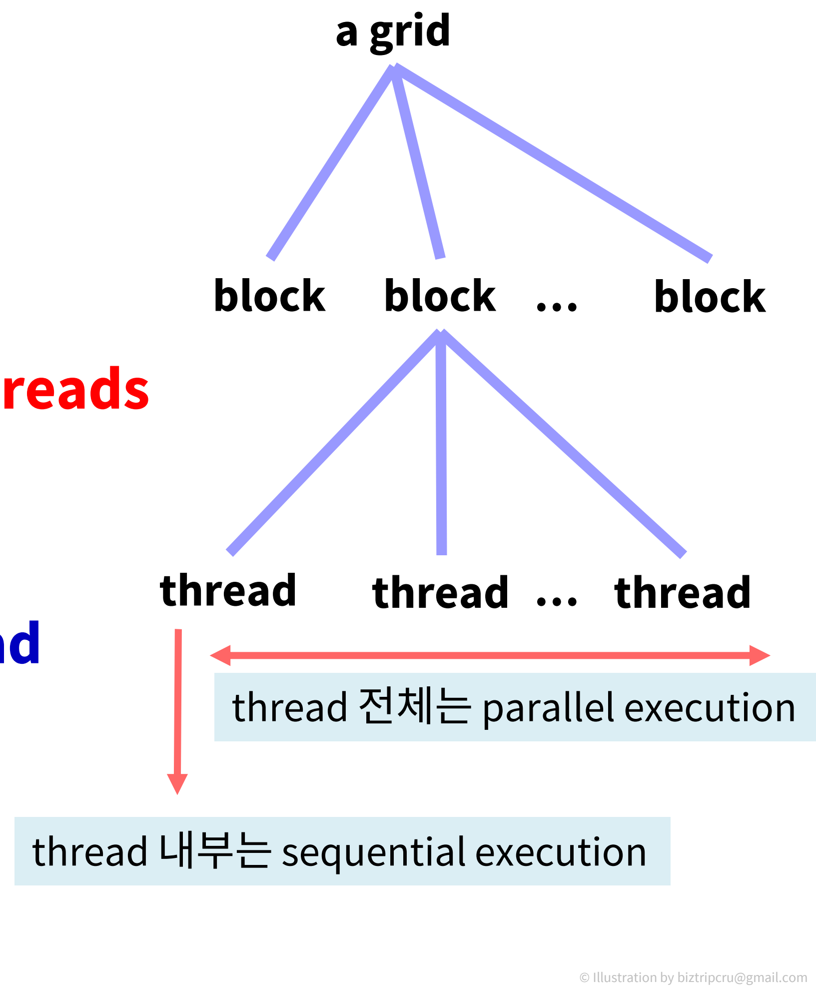


CUDA Programming Model

- **parallel code (kernel) is launched and executed on a device by many threads**
  - multiple threads → ~10 threads
  - many threads → 1,000+ threads → 실제로는 1M+ threads
- **on the many-core GPUs**
  - multi-core CPU → 10– cores
  - many-core GPU → 1,000+ cores → 실제로는 10K+ cores
- **보통, (thread 개수 ≫ core 개수)**
  - big-size data ! → 실제로는 1G ~ 1T items

CUDA Programming Model

- **many threads on many-core**
  - for example,
    - 1,000,000 threads on 1,000 cores
- **launches are hierarchical: grids – blocks – threads**
  - Threads are grouped into blocks
  - Blocks are grouped into grids
- **familiar sequential code is written for a thread**
  - 각 thread 내부는 사실상 sequential code
  - Built-in thread and block ID variables



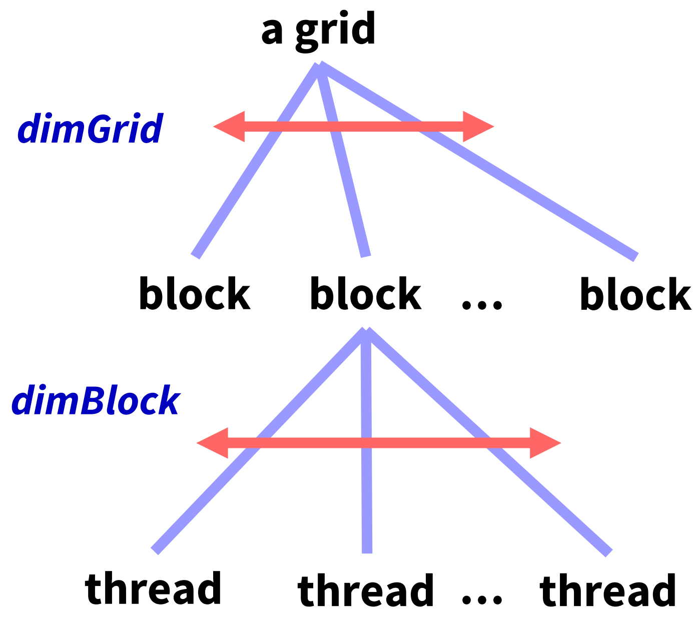
Calling a Kernel Function

- **kernel function 은 정해진 규약 대로 선언되고, 불러야 함**
- **kernel function 선언**
  - 독립적인 함수 형태로,

```
__global__  void  kernel_func( ... ) {
    ...
}
```

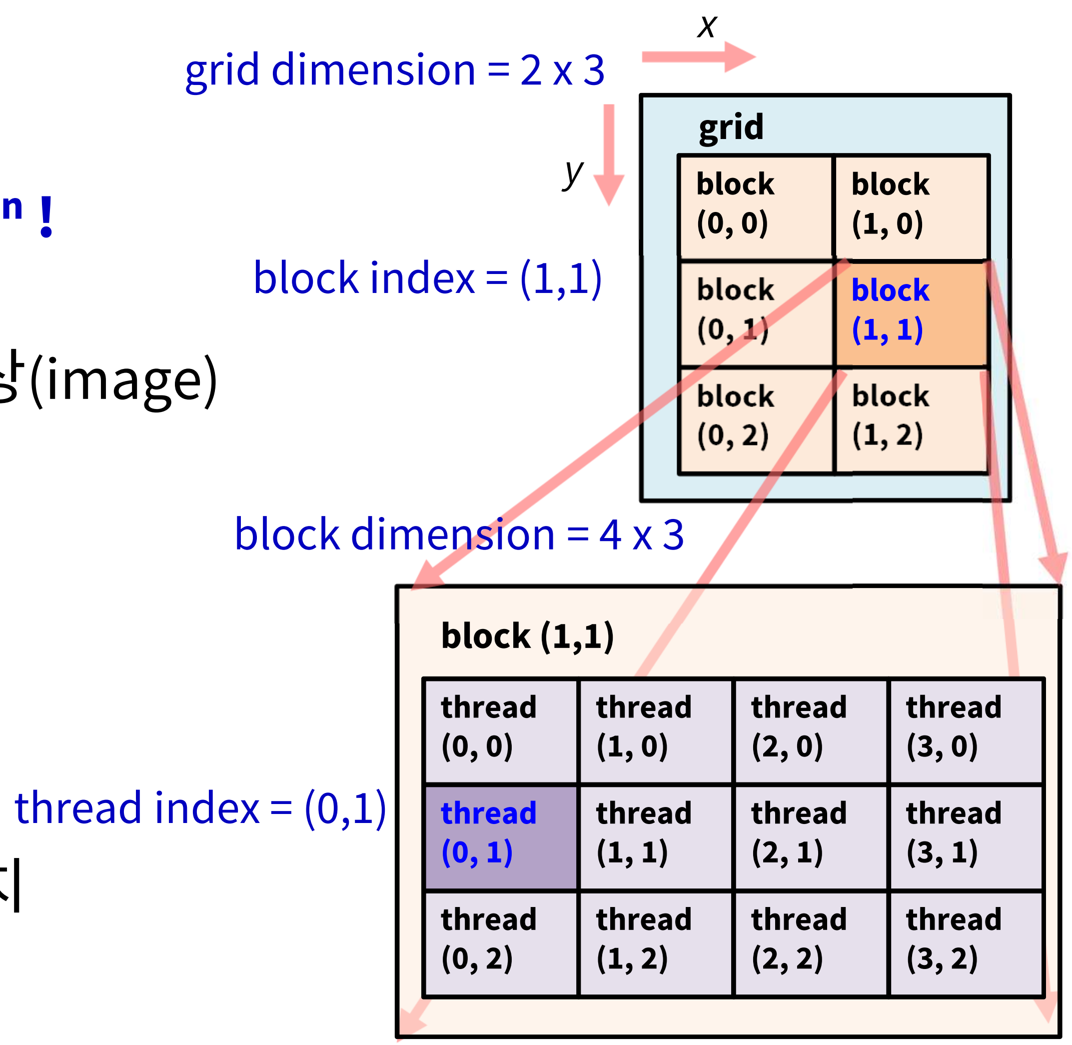
- **kernel function 의 호출**
  - __host__ function 에서,

```
kernel_func<<< dimGrid, dimBlock >>>( ... );
kernel_func<<< 8, 16 >>>( ... );
```



IDs and Dimensions

- **grid, block 구조는 최대 3차원 dimension !**
  - 1D : 1차원 배열
  - 2D : 2차원 배열, 행렬 (matrix), 영상(image)
  - 3D : 3차원 그래픽 자료
  - ID = identification number 식별 번호
- **grid : kernel 마다 1개**
  - grid dimension : 내부 block 배치
- **block: (x,y,z)** block index (ID)
  - block dimension: 내부 thread 배치
- **thread: (x,y,z)** thread index (ID)



CUDA pre-defined data types

- **Vector types**
  - char1, uchar1, short1, ushort1, int1, uint1, long1, ulong1, float1
  - char2, uchar2, short2, ushort2, int2, uint2, long2, ulong2, float2
  - char3, uchar3, short3, ushort3, int3, uint3, long3, ulong3, float3
  - char4, uchar4, short4, ushort4, int4, uint4, long4, ulong4, float4
  - longlong1/2/3/4, ulonglong1/2/3/4, double1/2/3/4
  - dim3
- **Components are accessible as variable.x, variable.y, variable.z, variable.w.**
  - we can consider it as a coordinate value: (x, y, z) or (x, y, z, w)
  - 생성자는 __host__ __device__ make_float4(x, y, z, w);

Predefined Variables

- **uint3**

```
class uint3 {
public:
  unsigned int x;
  unsigned int y;
  unsigned int z;
public:
  … (operations)
};
```

- **dim3**

```
class dim3 {
public:
  unsigned int x;
  unsigned int y;
  unsigned int z;
public:
  … (operations)
};
```

- **char1**

```
typedef char char1;
```

C++ class designs

- **default arguments** 디폴트 매개변수
  - constructor: dim3(unsigned x = 1, unsigned y = 1, unsigned z = 1);
  - dim3 can take 1, 2, or 3 arguments:
    - dim3 dimBlock1D(5); → (5, 1, 1) 을 의미
    - dim3 dimBlock2D(5, 6); → (5, 6, 1) 을 의미
    - dim3 dimBlock3D(5, 6, 7);
- **implicit type conversion** 암시적 형 변환
  - int 1개 → dim3 로 자동 변환 가능
    - kernelFunc <<< 3, 4 >>>(. . .);
    - kernelFunc <<< dim3(3), dim3(4) >>>(. . .);
    - kernelFunc <<< dim3(3,1,1), dim3(4,1,1) >>>(. . .);

Kernel Launch Syntax

- **kernel function 의 호출**
  - __host__ function 에서,

```
dim3  dimGrid ( 100, 50, 1 ); // 100 * 50 * 1 = 5000 thread blocks
dim3  dimBlock ( 4, 8, 8 );  // 4 * 8 * 8 = 256 threads per block
kernel_func<<< dimGrid, dimBlock >>>( ... );
```

totally, 5000 * 256 threads !

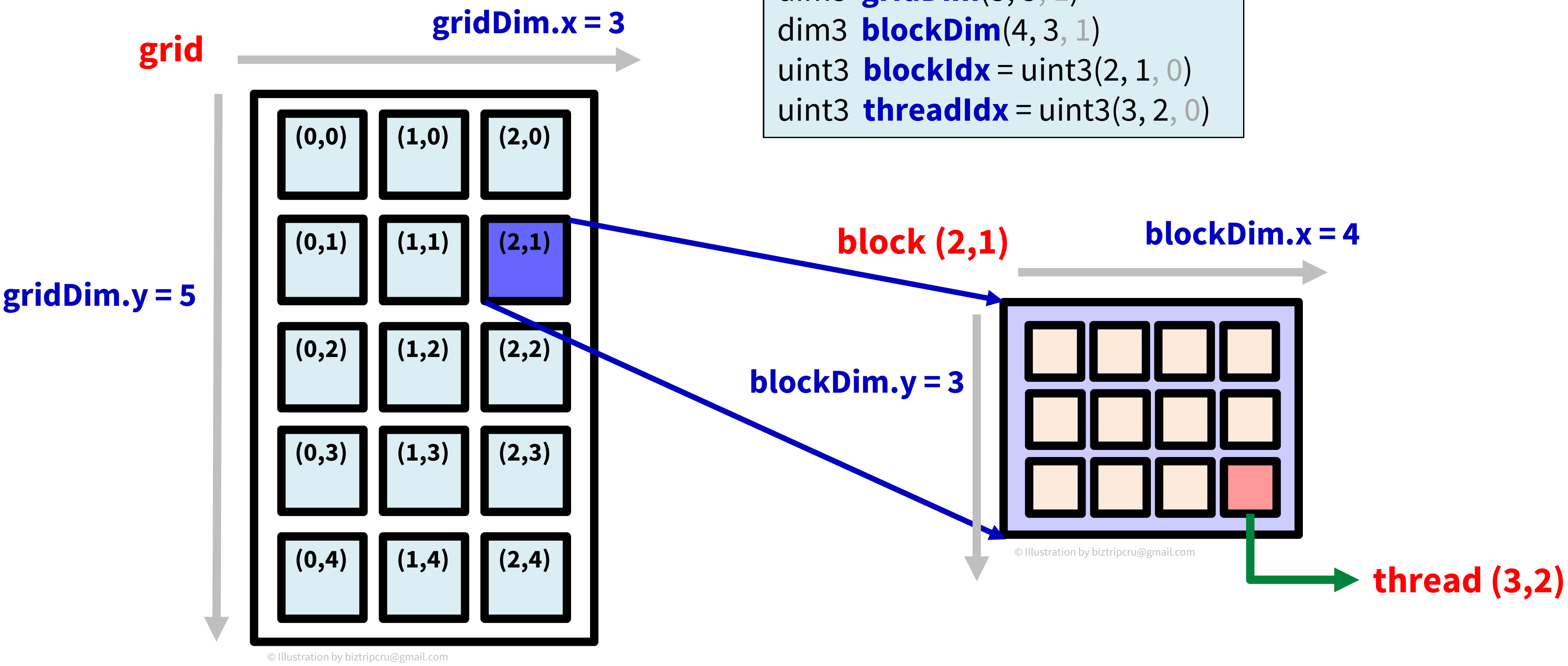
CUDA pre-defined variables

- **pre-defined variables**
  - dim3 **gridDim** dimensions of grid → gridDim.x
  - dim3 **blockDim** dimensions of block → blockDim.x
  - uint3 **blockIdx** block index within grid → blockIdx.x
  - uint3 **threadIdx** thread index within block → threadIdx.x
  - int **warpSize** number of threads in warp

- **모든 thread 에서 사용 가능**
  - gridDim.x, gridDim.y, gridDim.z, blockDim.x, blockDim.y, blockDim.z,
  - blockIdx.x, blockIdx.y, blockIdx.z, threadIdx.x, threadIdx.y, thrreadIdx.z,

Thread Layout

```
dim3  gridDim(3, 5, 1)
dim3  blockDim(4, 3, 1)
uint3  blockIdx = uint3(2, 1, 0)
uint3  threadIdx = uint3(3, 2, 0)
```

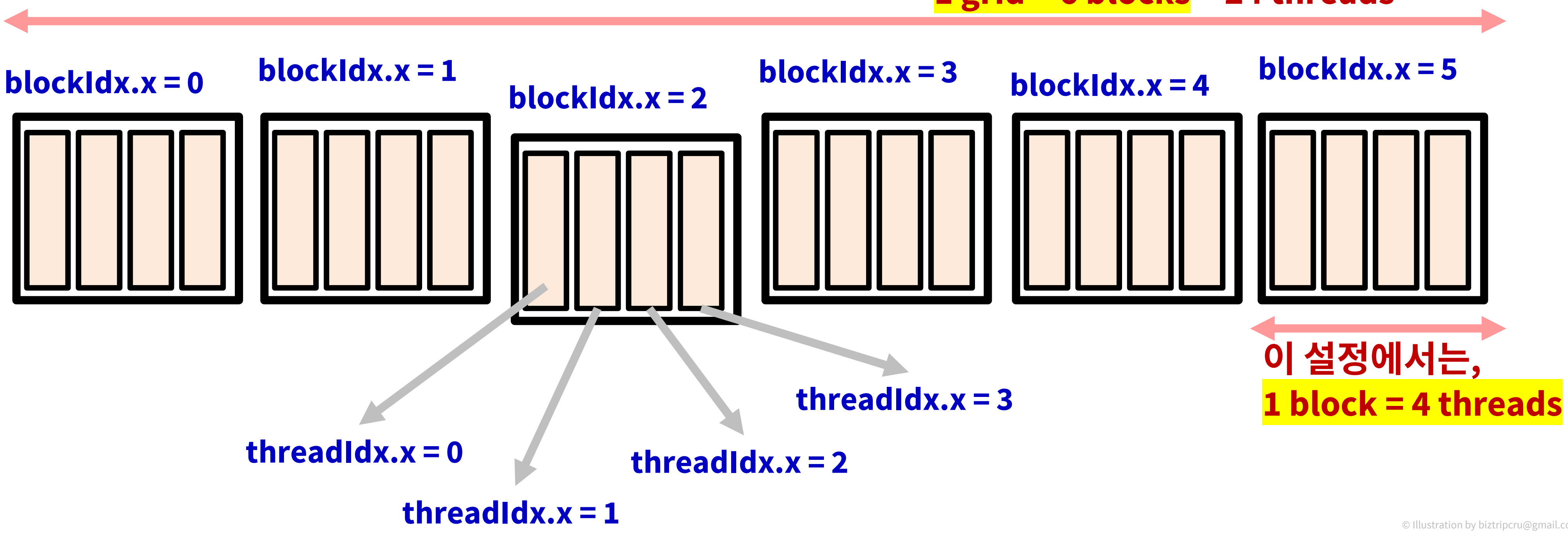


Example: 1D Layout

- dim3 gridDim(6);
- dim3 blockDim(4);

```
dim3  gridDim
dim3  blockDim
uint3 blockIdx
uint3 threadIdx
```

이 설정에서는,
1 grid = 6 blocks = 24 threads

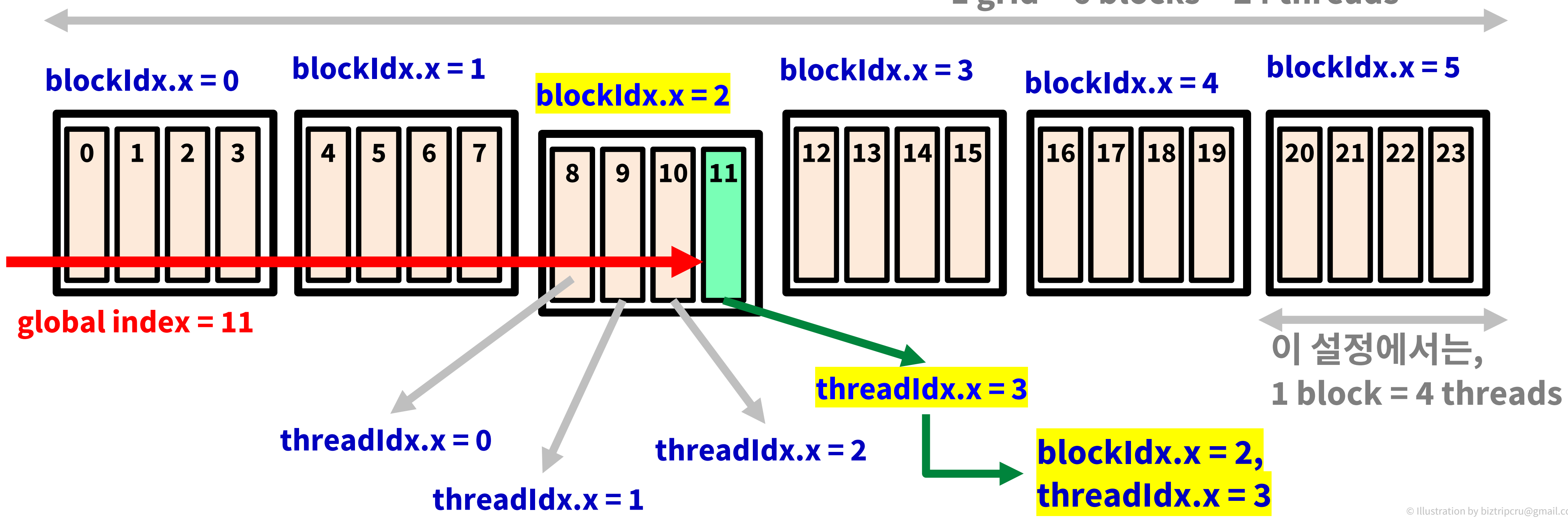


Example: 1D Layout

- dim3 gridDim(6);
- dim3 blockDim(4);

```
dim3  gridDim
dim3  blockDim
uint3  blockIdx
uint3  threadIdx
```

이 설정에서는,
1 grid = 6 blocks = 24 threads



Kernel with 1D Indexing

- **dim3 gridDim(6);** → blockIdx = 0 ~ 5
- **dim3 blockDim(4);** → threadIdx = 0 ~ 3
- **blockIdx, threadIdx : unique for each thread**

dim3 **gridDim**
dim3 **blockDim**
uint3 **blockIdx**
uint3 **threadIdx**

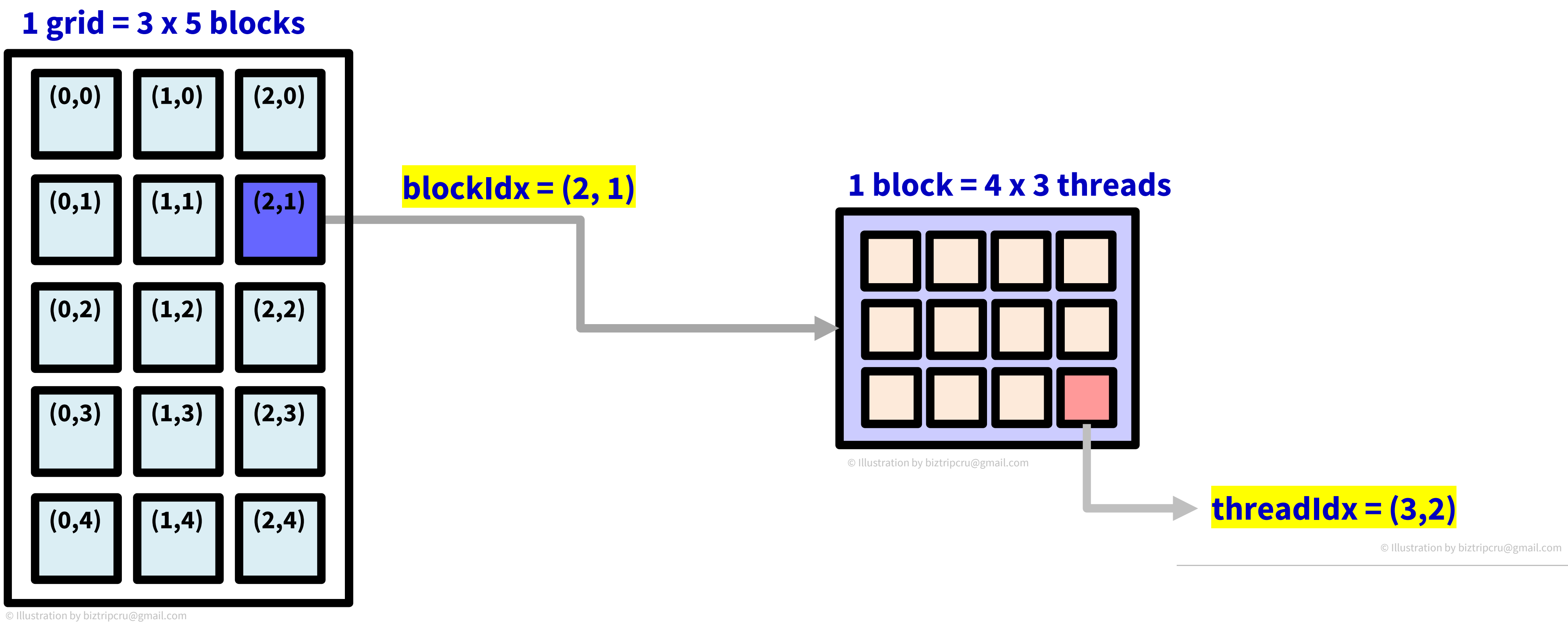
```
__global__ void kernel( int* a, int dimx ) {
   int idx = blockIdx.x * blockDim.x + threadIdx.x;
   if (idx < dimx) {
      a[idx] = a[idx] + 1;
   }
}
```

2 * 4 + 3 = 11 I am the thread #11 !

blockIdx.x = 2,
threadIdx.x = 3

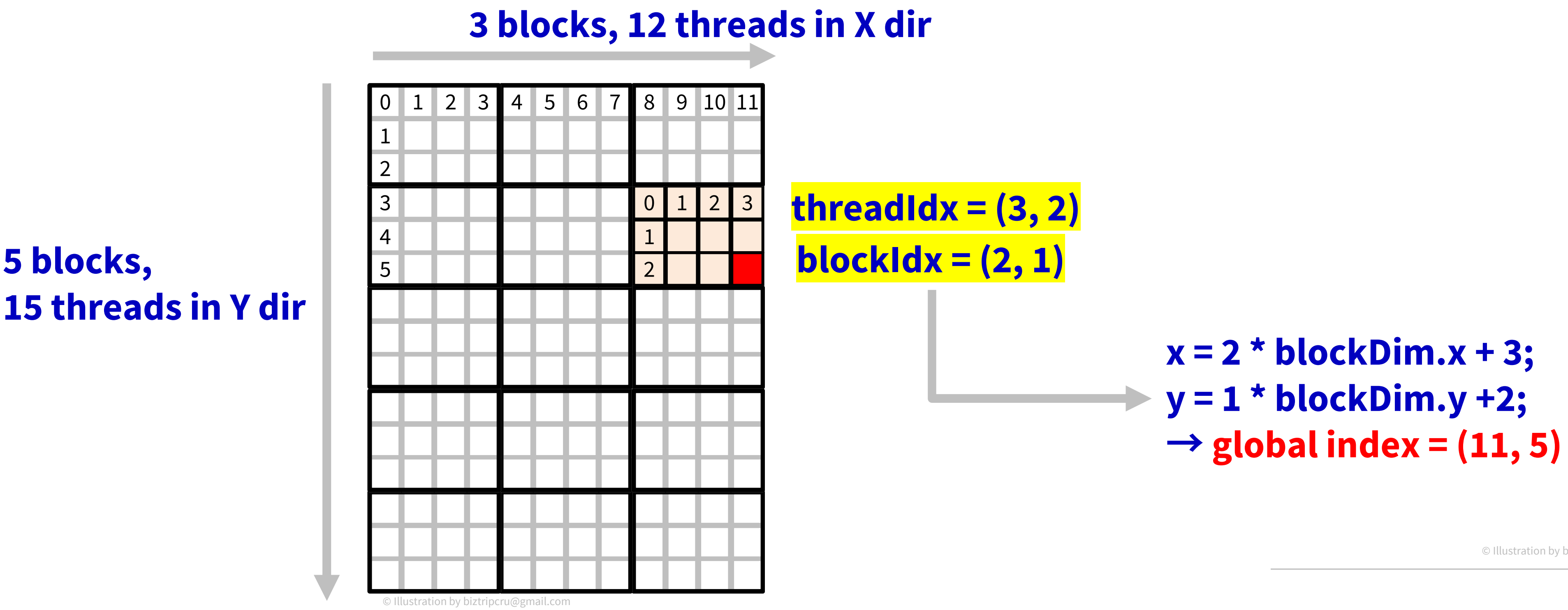
Example: 2D Layout

- dim3 gridDim(3,5); → 2D block index (x, y) in the grid
- dim3 blockDim(4,3); → 2D thread index (x, y) in a block



Example: 2D Layout

- dim3 gridDim(3,5); → 2D block index (x, y) in the grid
- dim3 blockDim(4,3); → 2D thread index (x, y) in a block



Kernel with 2D Indexing

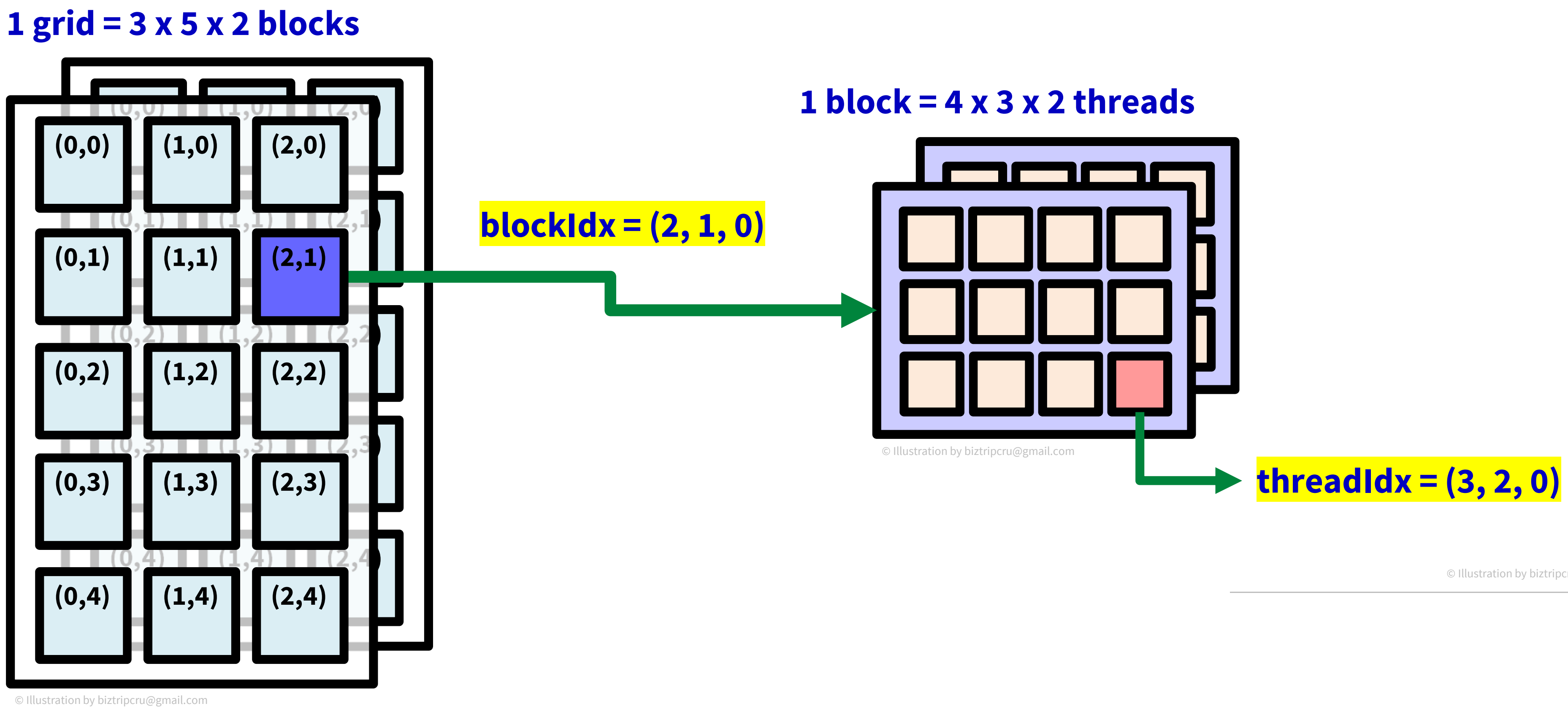
- **dim3 gridDim(3,5);**
- **dim3 blockDim(4,3);**
  - blockIdx, threadIdx : unique for each thread
- **(gx, gy) : global index**

```
__global__ void kernel( int* a, int dimx, int dimy ) {
   int gx = blockIdx.x * blockDim.x + threadIdx.x;
   int gy = blockIdx.y * blockDim.y + threadIdx.y;
   int idx = gy * dimx + gx;
   a[idx] = a[idx] + 1;
}
```

threadIdx = (3, 2)
blockIdx = (2, 1)

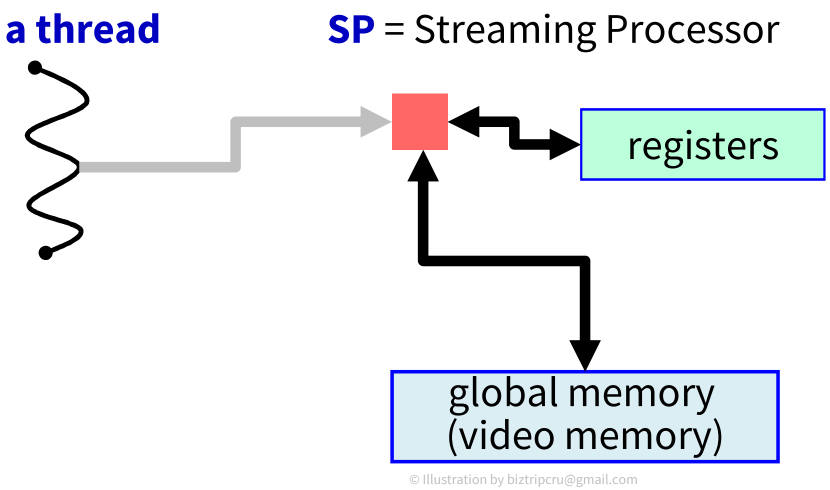
Example: 3D Layout

- **dim3 gridDim(3,5,2);** → **3D block index (x, y, z) in the grid**
- **dim3 blockDim(4,3,2);** → **3D thread index (x, y, z) in a block**



CUDA Architecture for threads

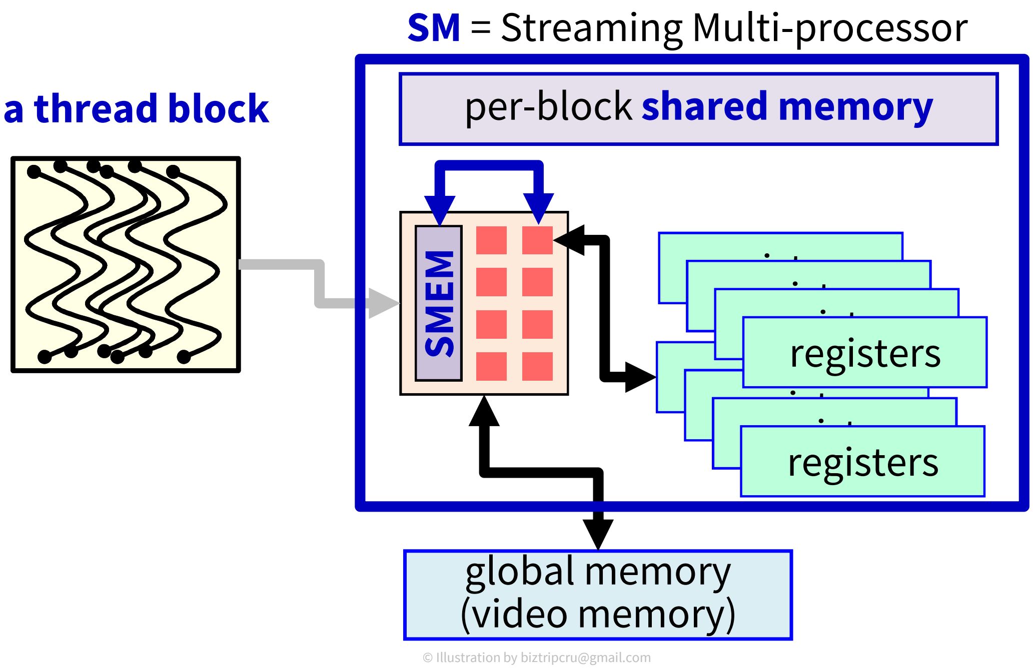
- **SP (streaming processor)**
  - for a single thread
  - ALU 정도의 낮은 성능
    - also known as "**core**"



- **주의: time sharing !**
  - 최대 효율이 나오게 해야
- **해결책: zero context switching**
  - thread 전환에 비용이 거의 없음
  - 매우 많은 register 를 할당
  - 예를 들어, 64K registers per SM

CUDA Architecture for threads 계속

- **SM (streaming multi-processor)**
  - for a thread block
  - ALU's + CU (control unit)

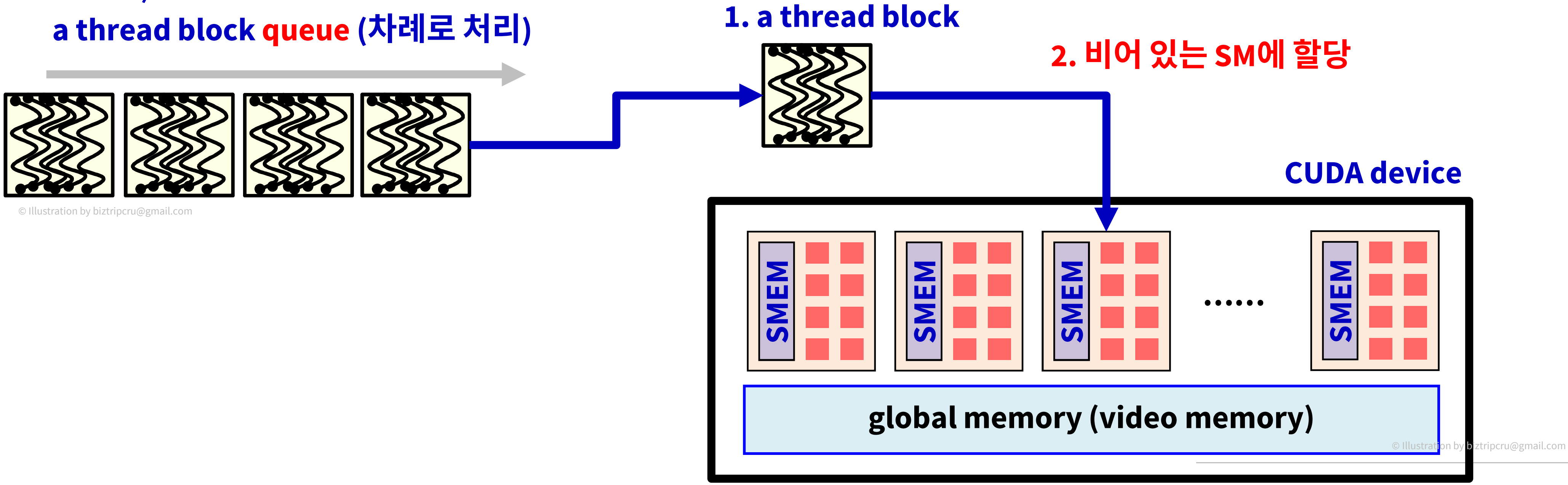


- **a thread block = many threads**
  - 예를 들어, 1024 threads
- **SM = a set of SP**
  - 예를 들어, 32 개의 SP
- **SM의 물리적 한계**
  = thread block의 최대 크기
- **time sharing again !**
  - SM 은 동시에 32개의 thread 실행
  - 1024 개의 thread 중 대부분은 대기

CUDA Architecture for threads 계속

- **CUDA device = many SM's**
  - for many thread-blocks
  - 보통, (thread block 개수 ≫ SM 개수)

- **5,000 thread blocks**
- **1,024 threads in a block**
- **16 SM's**



thread block queue 의 구현

- **정확히는 queue가 아니라, queue-like 구조**
  - (엄밀한 의미의) 큐 queue : FIFO (first-in, first-out)
  - (엄밀한 의미의) 우선순위 큐 priority queue : 정확한 우선순위 부여 필요
- **thread block queue의 요구 사항**
  - thread block 들을 저장
  - **하나씩 가져가서, 실행하고, 제거**
  - 정확한 우선순위가 필요한가? 또는 정확한 우선순위를 계산 가능한가?
  - 느슨한 queue-like 자료구조로 관리해도 충분

내용 contents

- process and thread
- CUDA programming model
- kernel launch
- pre-defined variables
- CUDA architecture

CUDA Kernel Launch

CUDA 커널 실행

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