Visual Studio CPU Debug and Nsight CUDA Debug

ECE 277

Cheolhong An

Integrated Info VS.

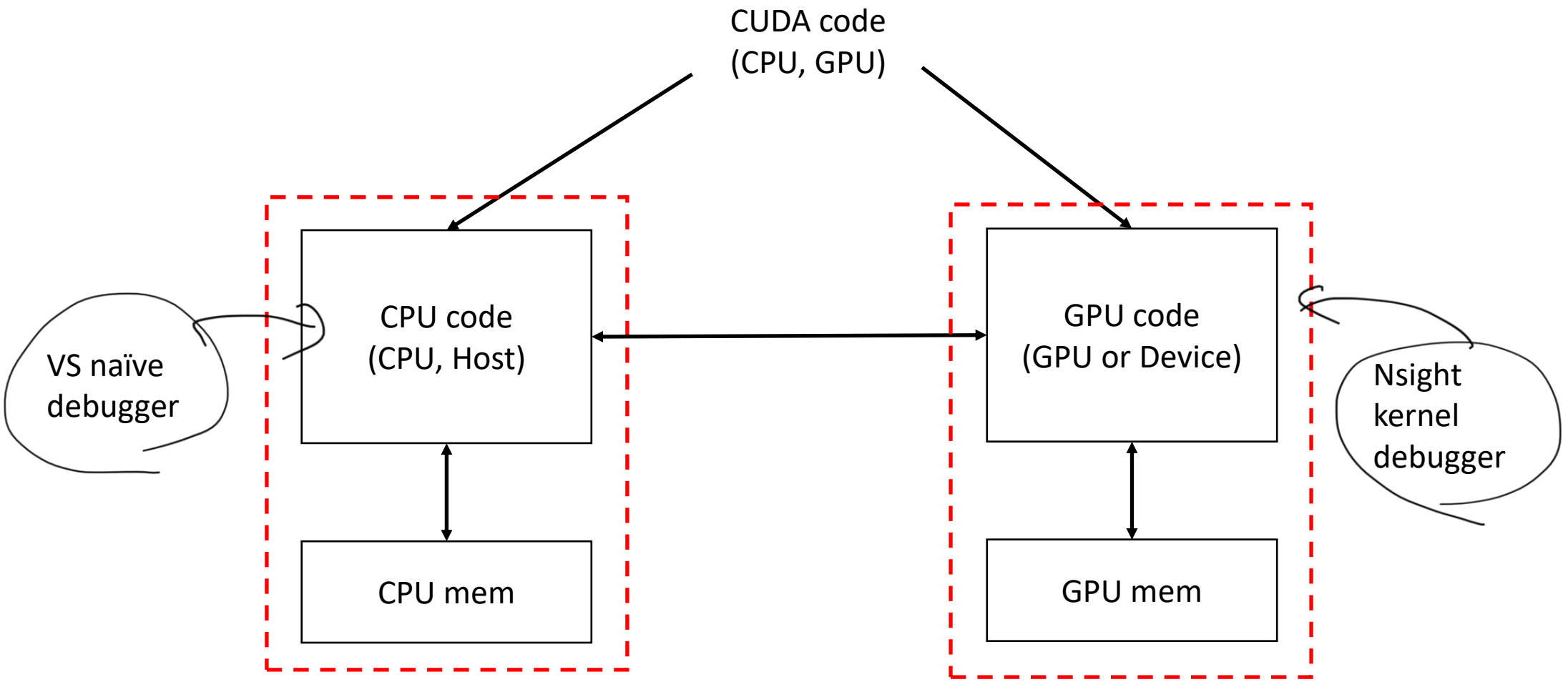
Visual Studio + Nsight VS edition

- CUDA debug (**Nsight VS**) is integrated into Visual Studio for debug and profile
- However, **separate launches are required for CUDA kernel debug (Nsight) and CPU debug (VS naïve)**

⇒ debug seperately.

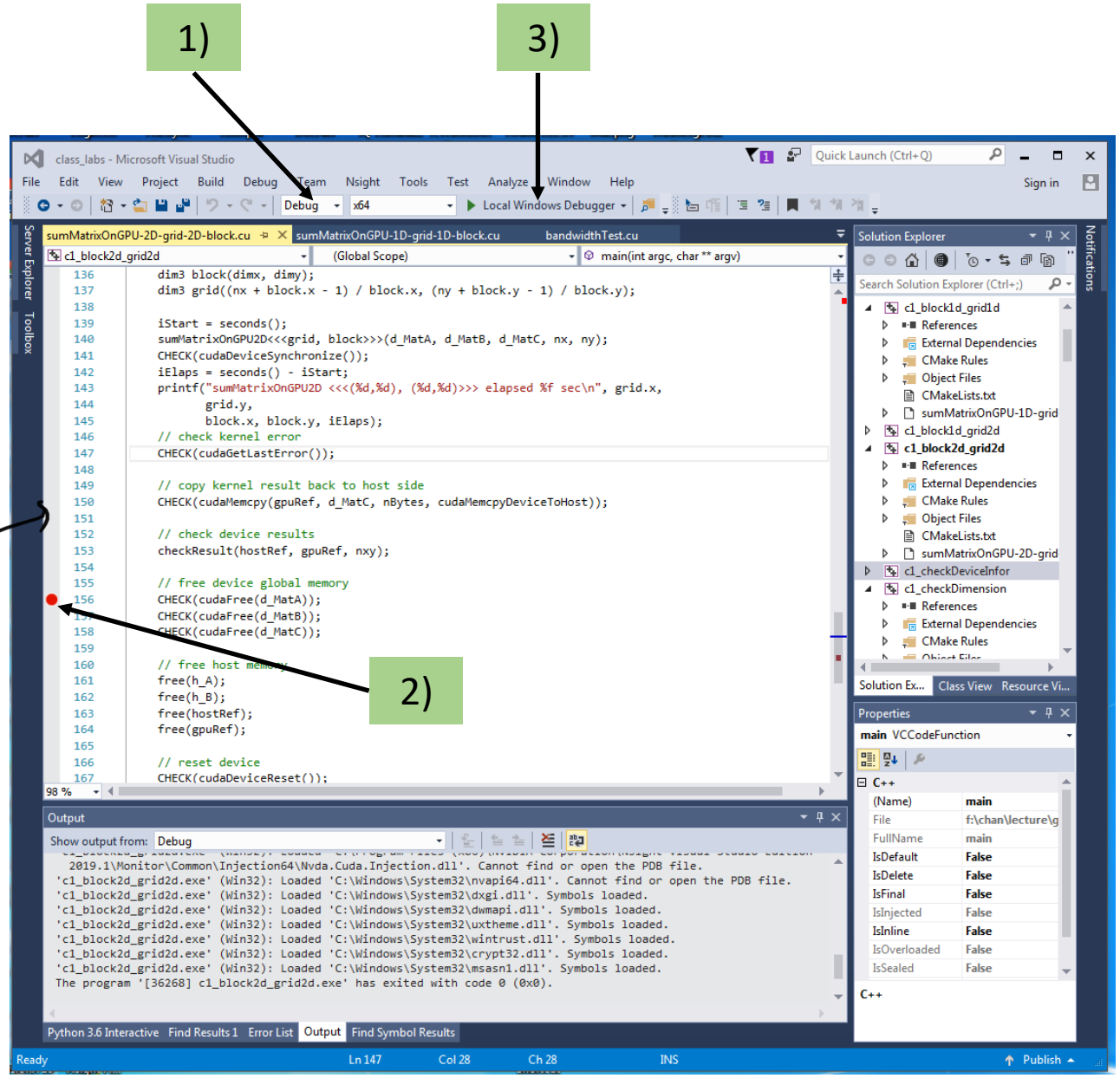
CUDA programming

- support both cpu and gpu (Heterogeneous)



VS Debug: CPU (1/2)

1) Compile a project with “Debug” mode
2) Add a breakpoint
   click the gray area and a red dot appears
3) Run a program



VS Debug: CPU (2/2)

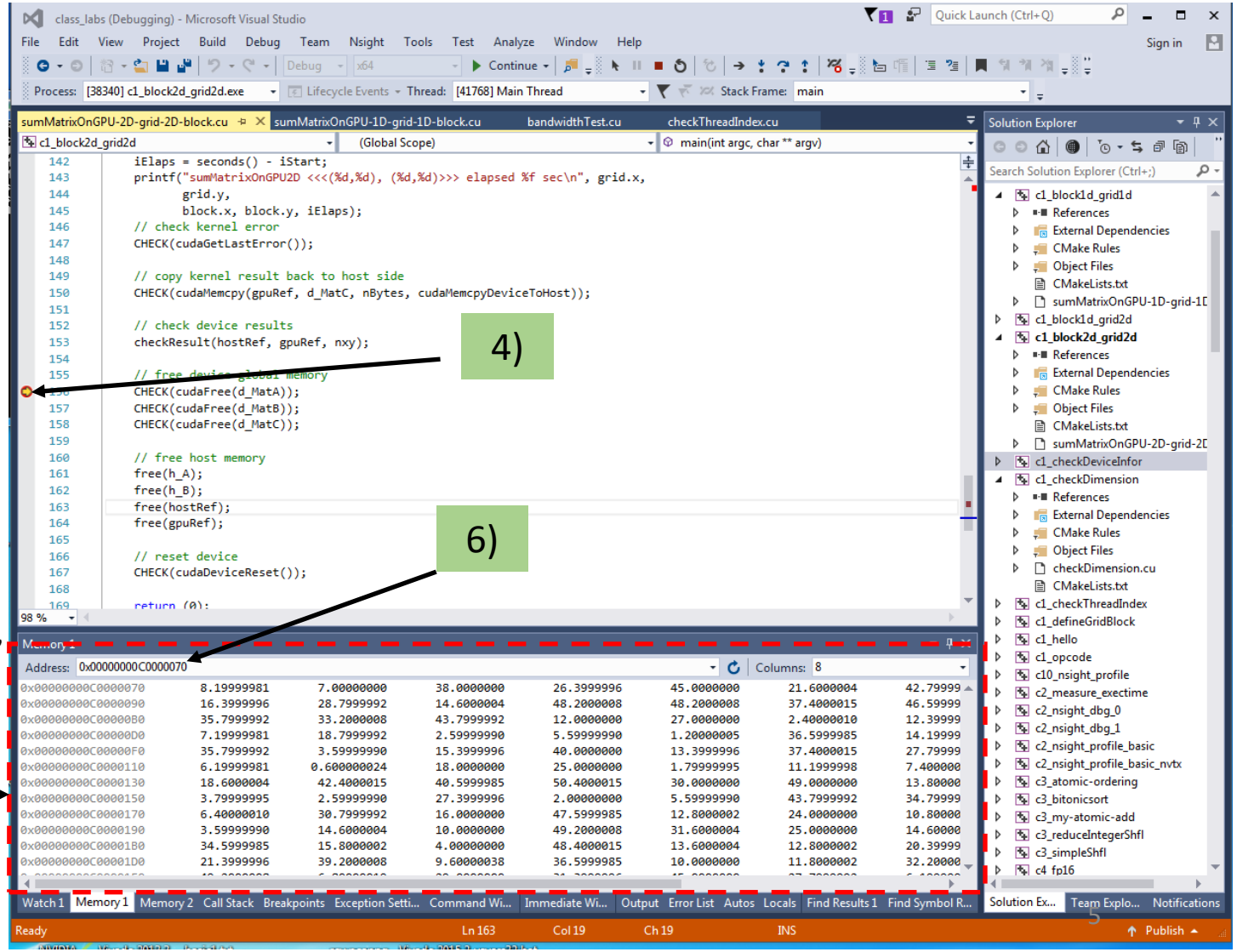
4) When the program stop at the break point, yellow arrow will appear
5) Open “Memory 1” window

   Debug->Windows->Memory->Memory 1
6) View memory contents

   Type a point variable name into the address window

   (copy or drag-drop)

   & 9225 (byte type offset ;)

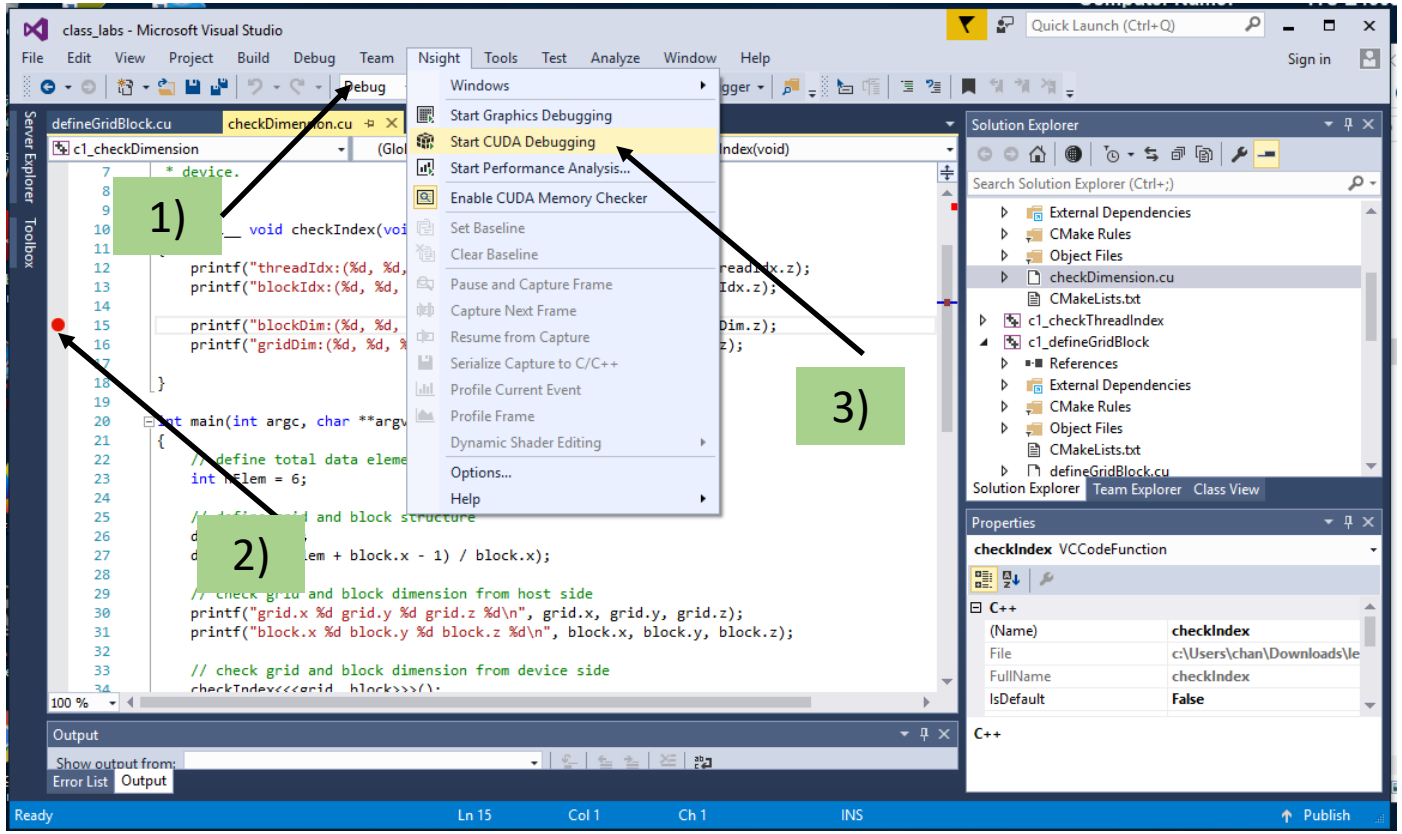


Nsight VS Debug: GPU Kernel (1/3)

- Nsight VS debug is integrated into Visual Studio, but you should invoke “Nsight->Start CUDA Debug”
- CPU and GPU kernel debug cannot run simultaneously
  Only one (CPU debug or GPU kernel debug) at a time

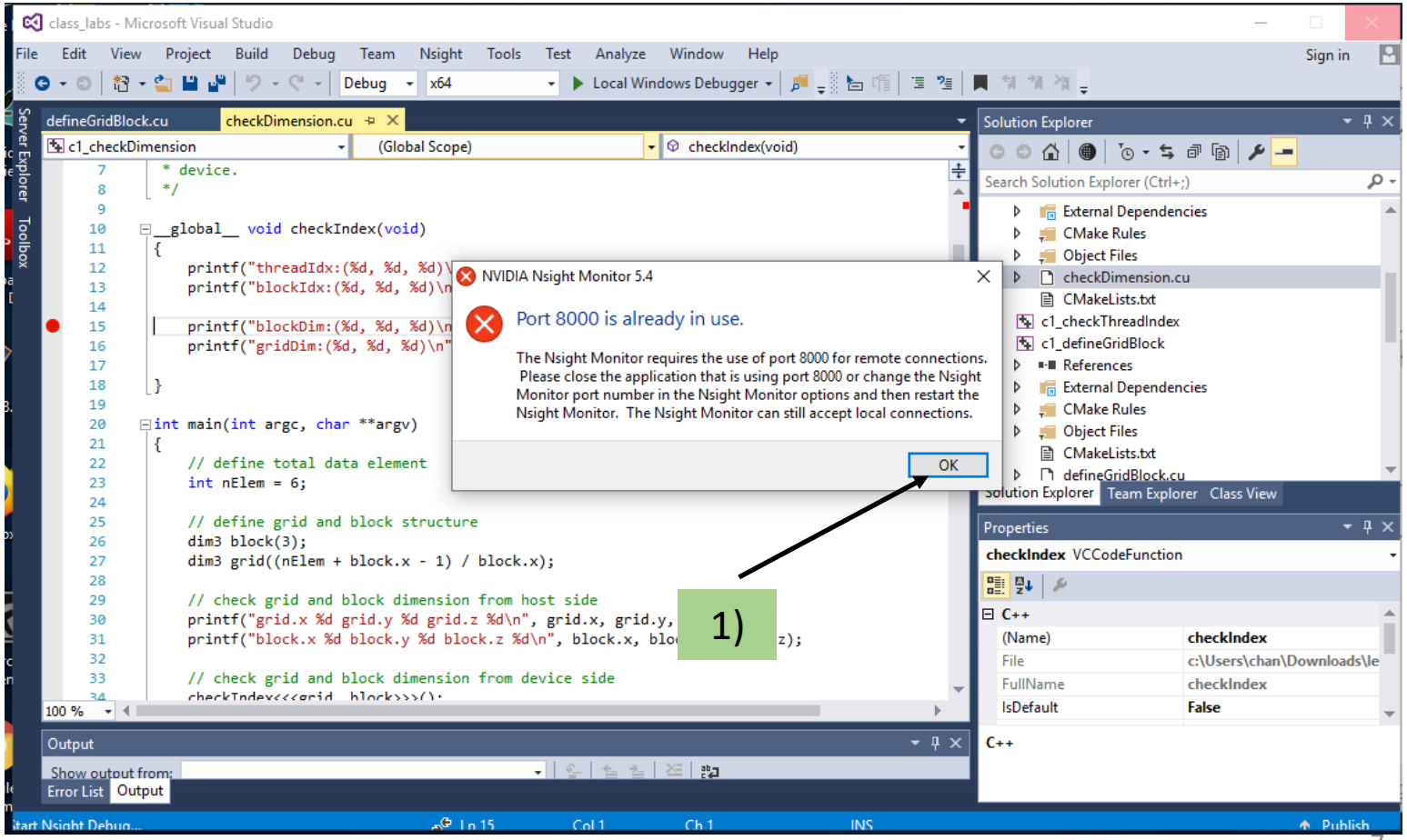
1) Set “Debug” mode
2) Add break points to CUDA kernels
3) “Start CUDA Debugging”

(Legacy) 시험



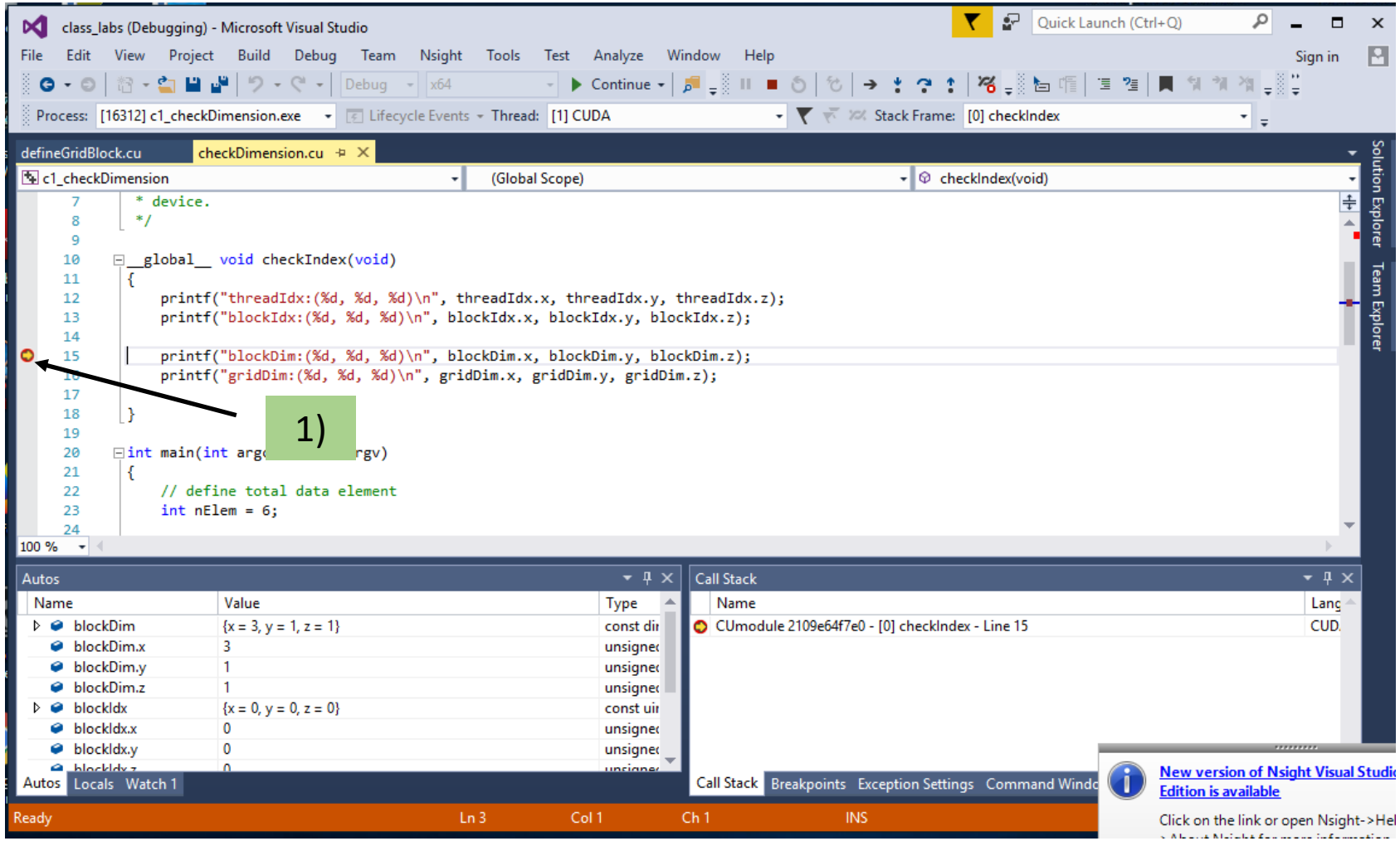
Nsight VS Debug: GPU Kernel (2/3)

1) Just Click “OK” if you see “Port 8000 is already in use”



Nsight VS Debug: GPU Kernel (3/3)

1) Confirm a “yellow arrow” when a kernel program stop at a break point



Watch 1 에서 원하는 variable 입력.

/// not multiple threads debugging ///

⇒ 특정 thread debug ? (conditional breakpoint)

CUDA debugging Tips

1) Include Helper functions
   #include <helper_functions.h>
   #include <helper_cuda.h>
2) Always add checkCudaErrors after calling cuda functions
   checkCudaErrors(cudaMalloc());
3) **Always use a conditional break point for a specific thread** (blockIdx, threadIdx)
   @blockIdx(1,1,0) && @threadIdx(1,0,0)
4) Enable “Cuda-Memcheck” to check Device memory access violations
5) Get error reports (refer to the next slide)

Conditional breakpoints

- @blockIdx(0,1,0) && @threadIdx(0,1,0)

Macro	Expansion	Notes
@threadIdx(x, y, z)	(threadIdx.x == (x) && threadIdx.y == (y) && threadIdx.z == (z))	x, y, and z must be decimal integers.
@threadIdx(#N)	((((threadIdx.z * blockDim.y) + threadIdx.y) * blockDim.x + threadIdx.x) == (N))	N must be a decimal integer.
@blockIdx(x, y, z)	(blockIdx.x == (x) && blockIdx.y == (y) && blockIdx.z == (z))	x, y, and z must be decimal integers.
@blockIdx(#N)	((((blockIdx.z * gridDim.y) + blockIdx.y) * gridDim.x + blockIdx.x) == (N))	N must be a decimal integer.

Reporting Errors

- All CUDA API calls return an error code (**cudaError_t**)
  - Error in the API call itself
    - OR
  - Error in an earlier asynchronous operation (e.g. kernel)

- Get the error code for the last error:

```
cudaError_t cudaGetLastError(void)
```

- Get a string to describe the error:

```
char *cudaGetErrorString(cudaError_t)
printf("%s\n", cudaGetErrorString(cudaGetLastError()));
```

⇒ print 해서 meaningful information 얻을 수 있다.

CUDA Debug memory or registers as CPU Debug

- **View memory**
  - Really useful for CUDA debugging since you can only debug one thread
  - You cannot access the local variables of the other threads
  - However, the Global memory is common to all threads so it is also updated by the other threads
  - You can also check values of the other threads through the Global memory

- View variables
  - **You can only check local variables of a current thread**
- Warp watch

(other thread's variable 은 볼 수 x .