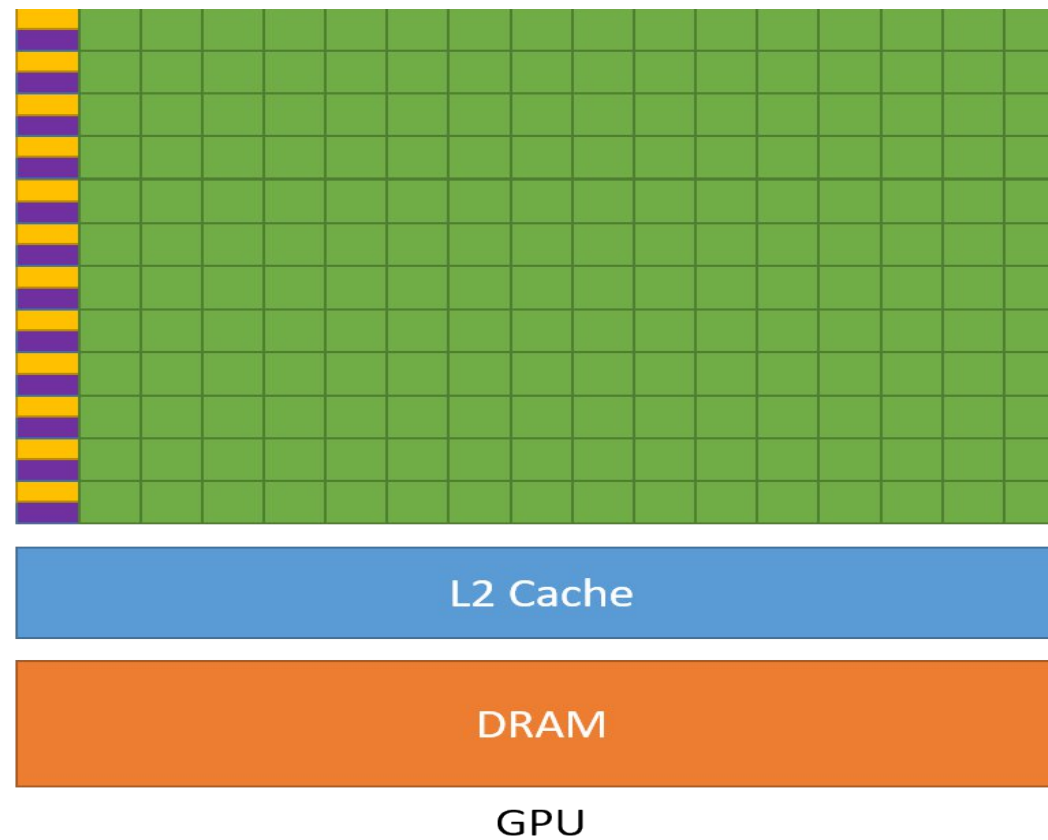
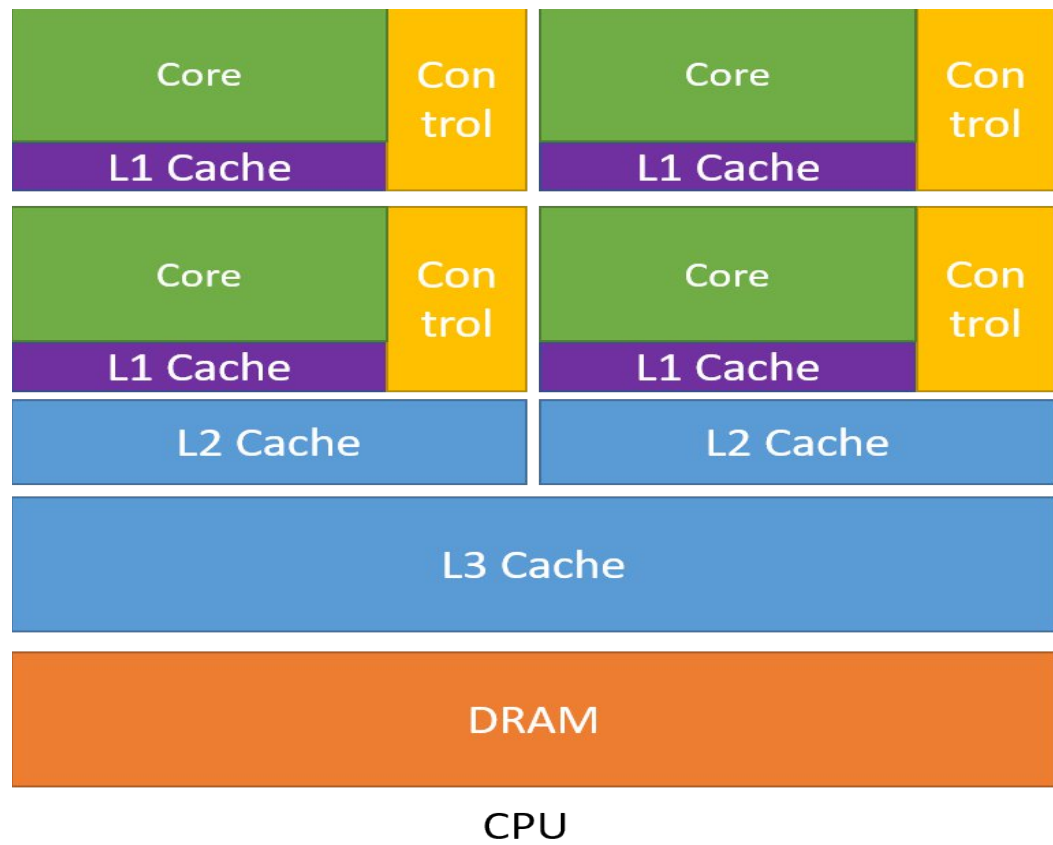


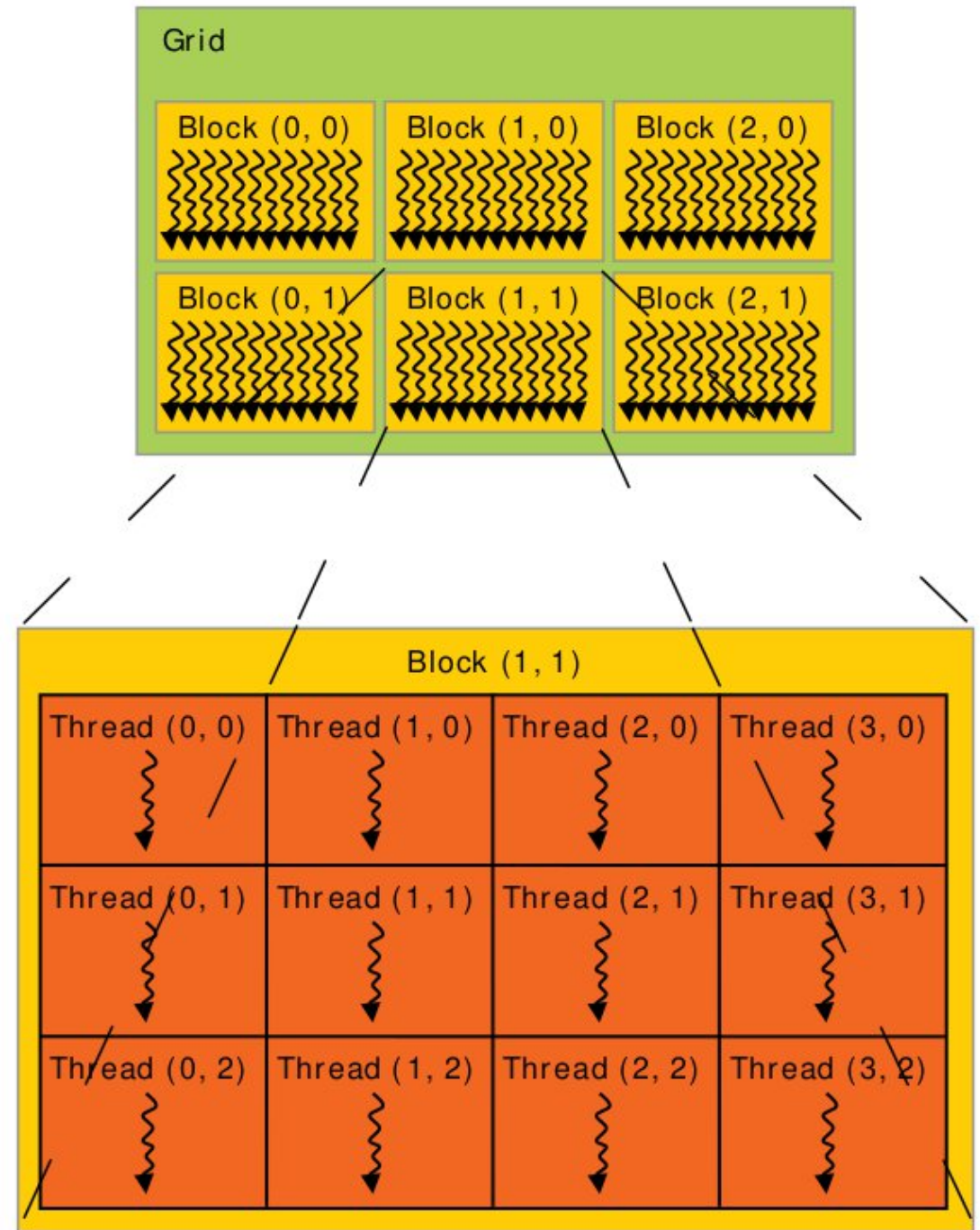
CUDA

# Архитектура CUDA



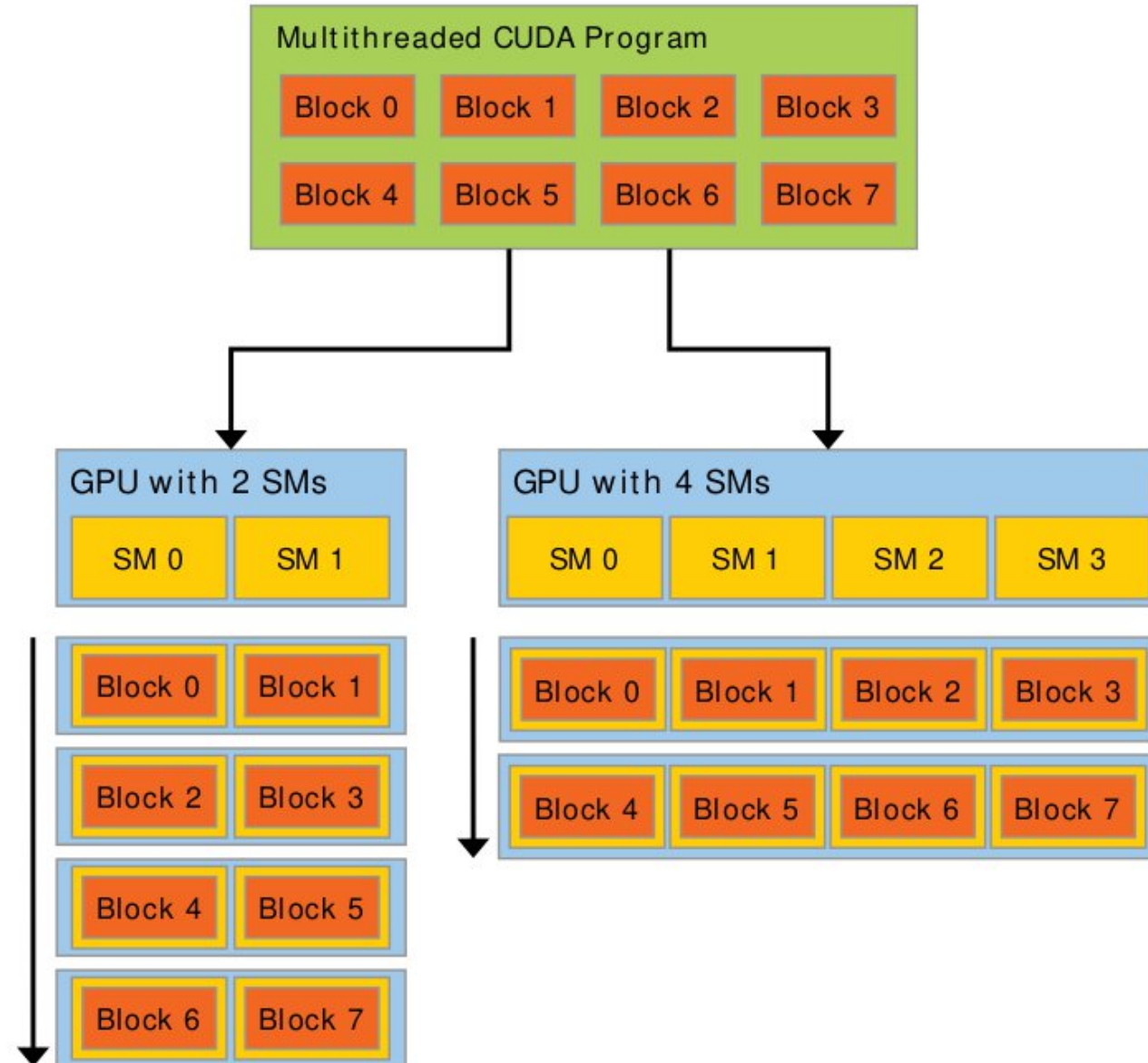
# Архитектура CUDA

- Потоки объединены в блоки.
- Каждый блок контролируется SM (Streaming Multiprocessor)
- Блоки объединены в общий Grid



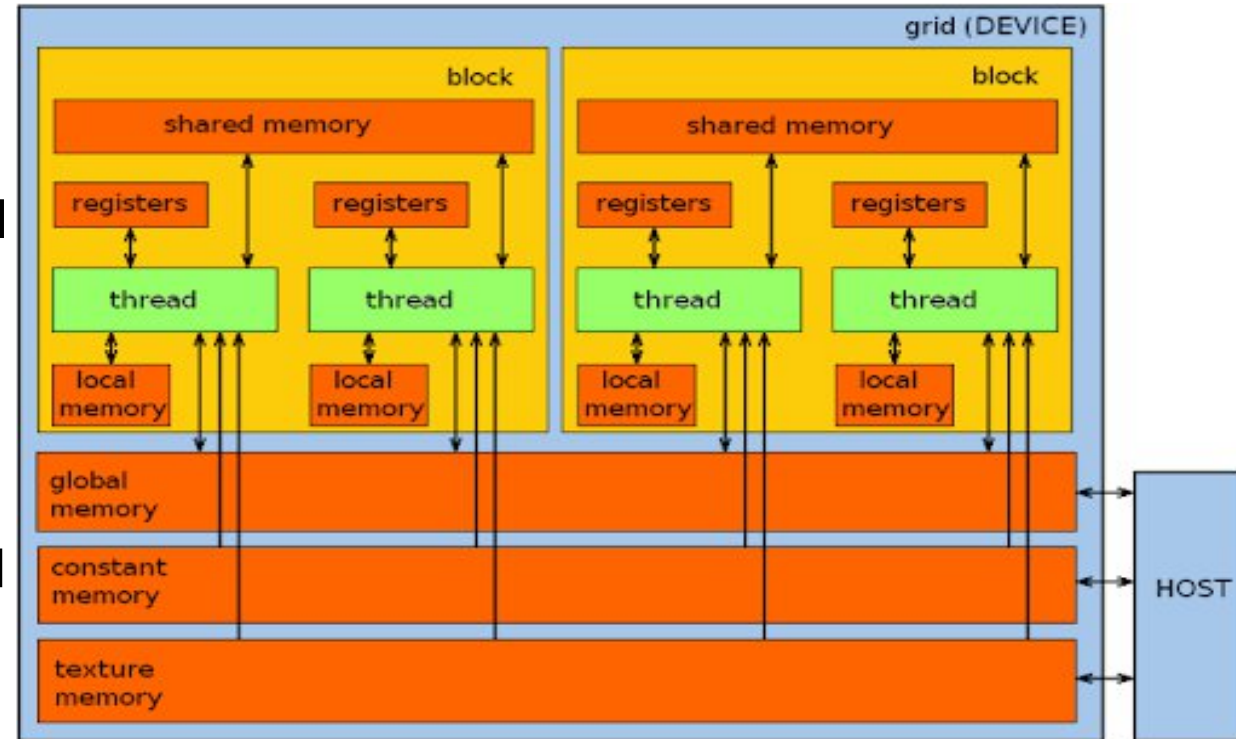
# Архитектура CUDA

- SM управляет warp'ами по 32 потока в каждом
- Так как flow потоков в warp контролирует один SM, необходимо, что бы потоки выполняли одинаковые действия. Иначе, те потоки, чьи действия отличаются, будут приостановлены.



# Память в CUDA

- Каждый поток обладает локальной памятью.
- Каждый блок обладает общей памятью, доступ к которой имеют все потоки в блоке.
- Так же существует общая память GPU, доступ к которой имеет HOST.



# Kernel

- Kernel - функция для всех потоков.
- Объявляется с помощью `__global__`.
- Вызов функции должен сопровождаться execution configuration `<<<GridSize,BlockSize>>>`
- Параметры для конфигурации имеют тип `dim3`.

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

# Execution configuration

- Блоки и Grid могут быть одномерными, двухмерными и трехмерными.
- **threadIdx** - dim3 индекс потока в блоке
- **blockIdx** - dim3 индекс блока
- **blockDim** - dim3 размер блока.

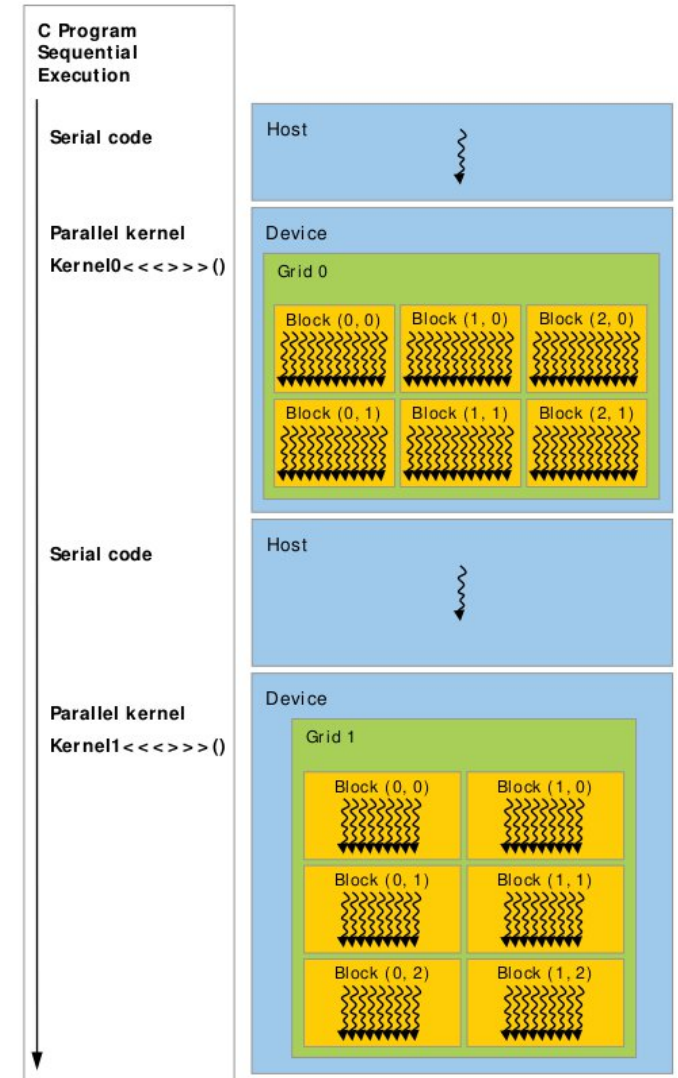
```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N],
                      float C[N][N])
{
    int i = threadIdx.x;
    int j = threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    ...
    // Kernel invocation with one block of N * N * 1 threads
    int numBlocks = 1;
    dim3 threadsPerBlock(N, N);
    MatAdd<<<numBlocks, threadsPerBlock>>>>(A, B, C);
    ...
}
```



# Пример работы программы

Kernel можно вызывать несколько раз в программе.





# Работа с памятью

- `cudaMalloc` ( `void** devPtr, size_t size` )
- `cudaMemcpy` ( `void* dst, const void* src, size_t count, cudaMemcpyKind kind` )
- `cudaMemcpyKind`:
  - `cudaMemcpyHostToDevice`
  - `cudaMemcpyDeviceToHost`
- `cudaFree` ( `void* devPtr` )

# Shared memory

- `__shared__` - модификатор, который необходимо указать при объявлении массива в функциях с модификатором `__global__` и `__device__`. В таком случае массив будет объявлен в общей памяти блока.