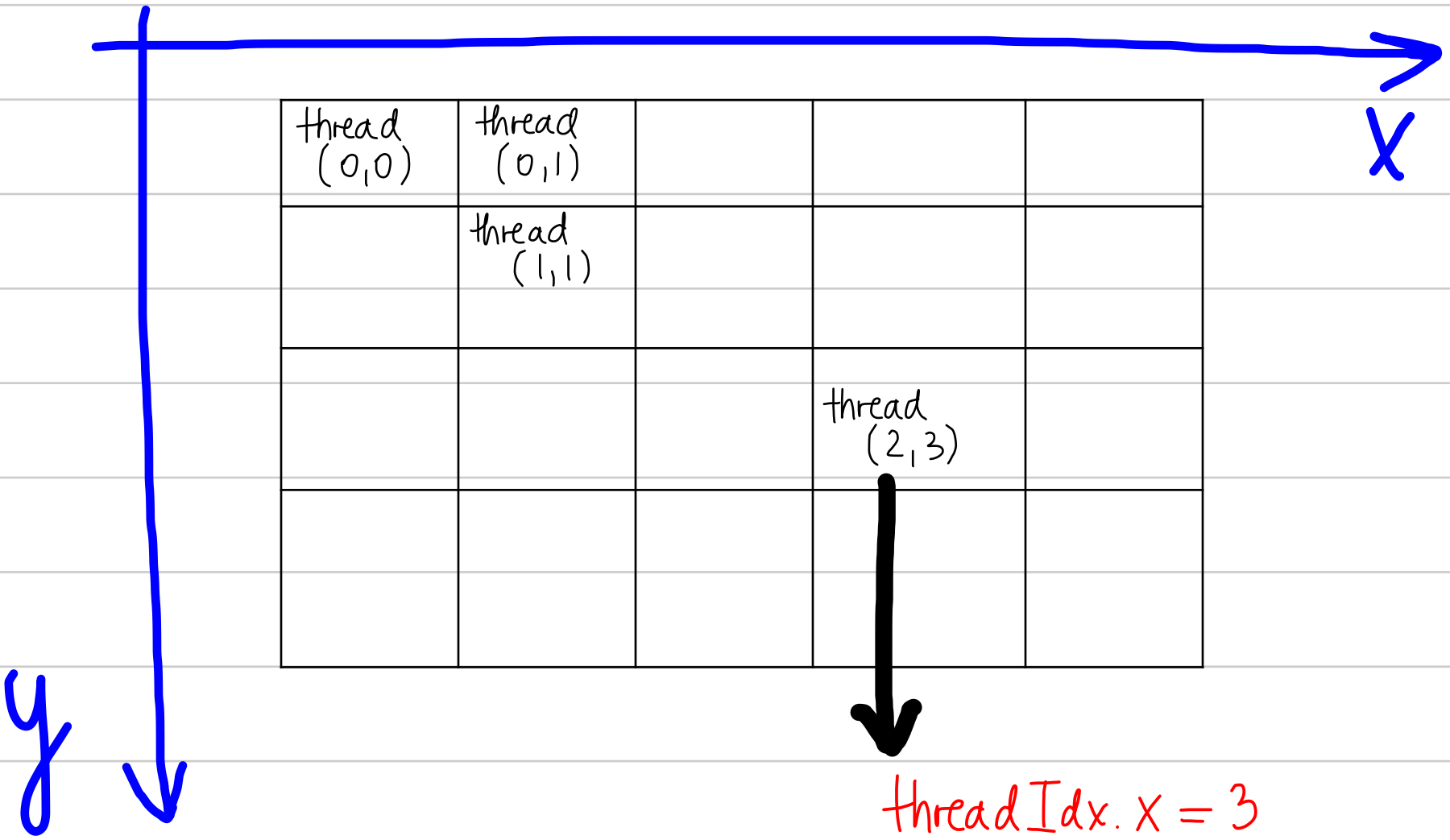
1/13/12

- Kernel = function which is executed in parallel threads.
- the index of a thread can be accessed via threadIdx variable

threadIdx = (threadIdx.x, threadIdx.y, threadIdx.z)

Example: 2D array of threads:



threadIdx.x = 3

threadIdx.y = 2

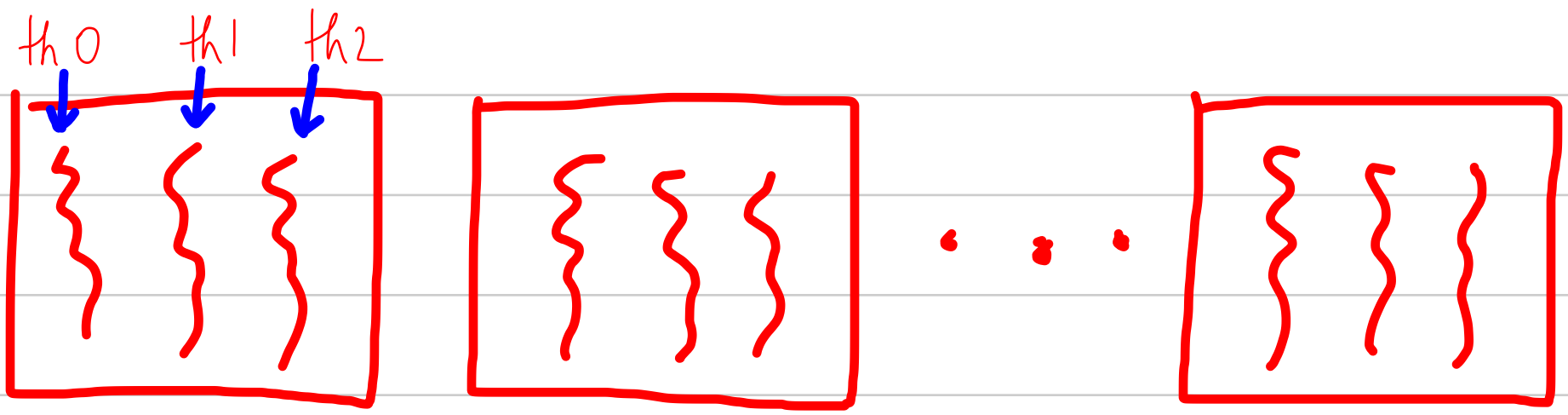
- threads are organized in blocks

- Recall the kernel call : kernel<<<1, N>>>(...)

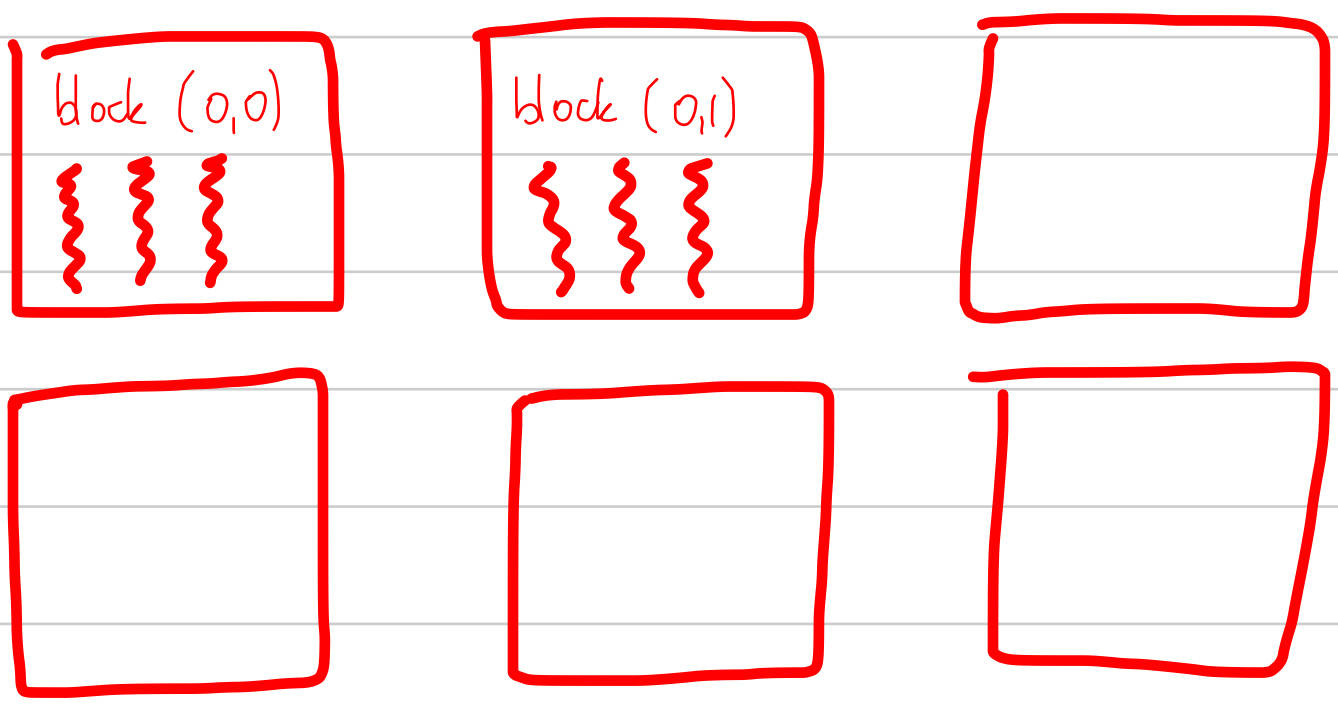
1 = # of blocks

N = # of threads per block

kernel<<<5, 3>>> : = total of 15 parallel threads



- block index can be accessed via blockIdx variable



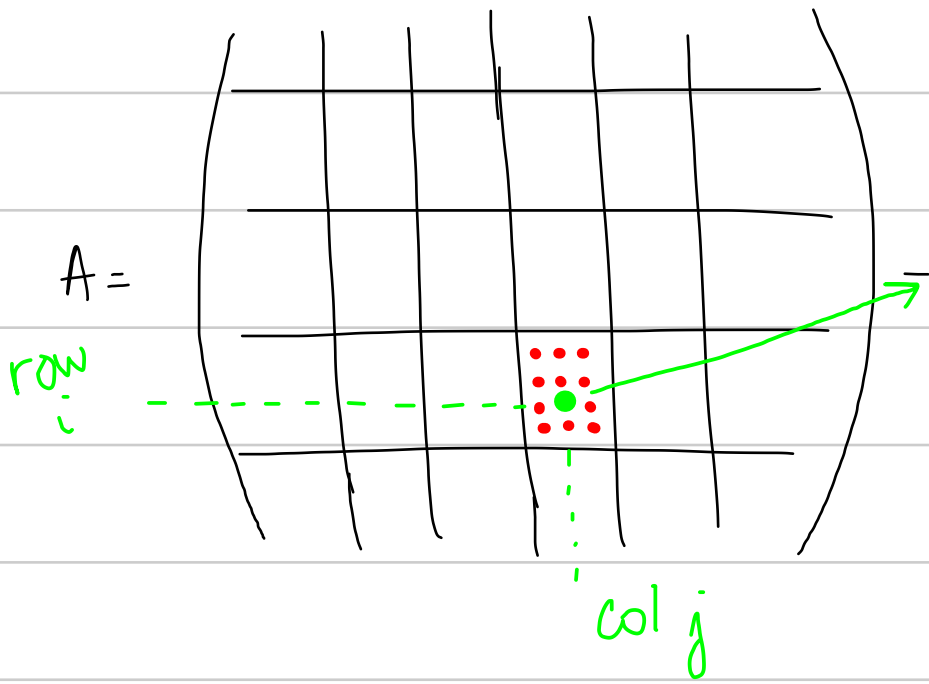
2D array of blocks

Matrix multiplication

$$C_{ij} = \sum_k a_{ik} \cdot b_{kj}$$

Given that A and B are in the GPU memory, each C_{ij} can be computed in a different thread.

- max 1024 threads in a block
- need to organize threads in blocks
- imagine A, B, C being organized in tiles



- each tile will correspond to 1 block of threads
- each thread will compute one C_{ij}

```
__global__ void matrixMul(Matrix A, Matrix B, Matrix C){

  int tidx=threadIdx.x;
  int tidy=threadIdx.y;

  int bidx=blockIdx.x;
  int bidy=blockIdx.y;

  int row = bidy*TILE+tidy;
  int col = bidx*TILE+tidx;


  int i;
  double S=0.0;
  for(i=0;i<MS;i++){
    S += A.elements[IDX2(row,i)]*B.elements[IDX2(i,col)];
  }
  C.elements[IDX2(row,col)] = S;
}
```

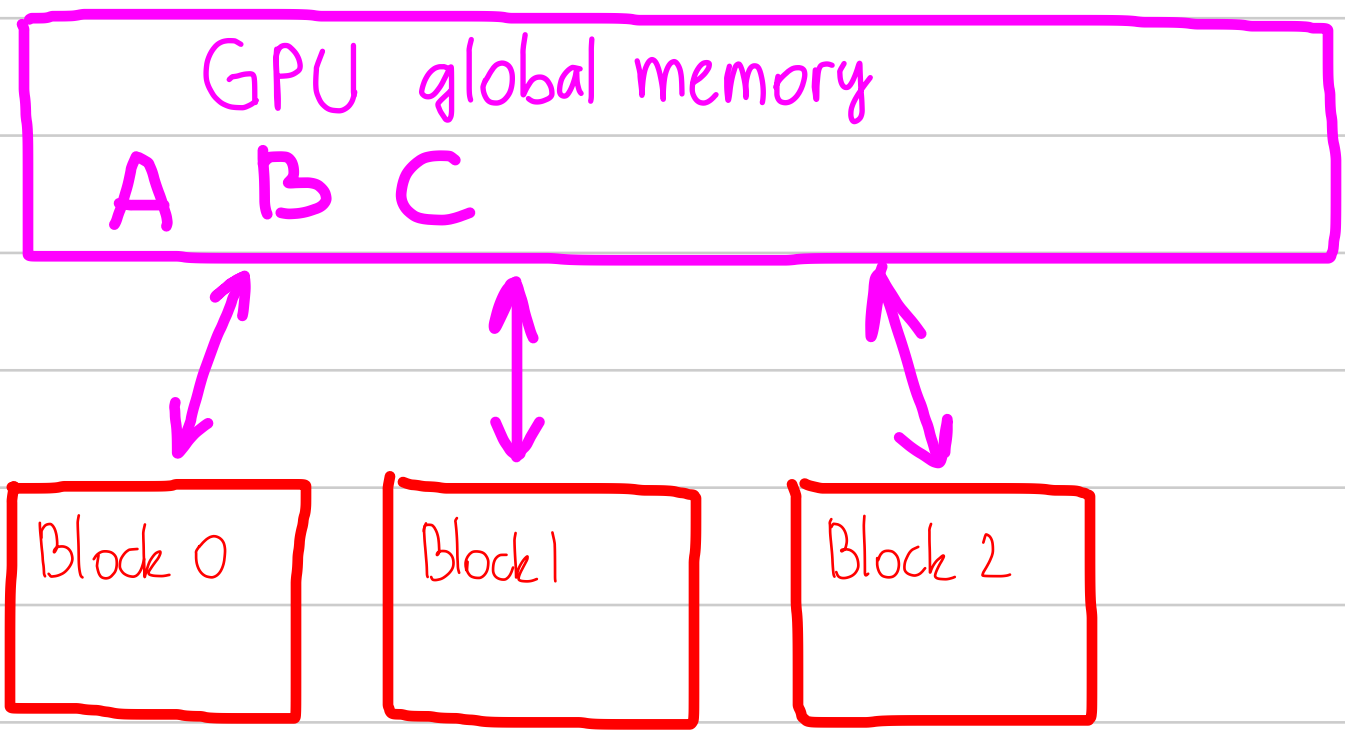
thread coordinates

block coordinates

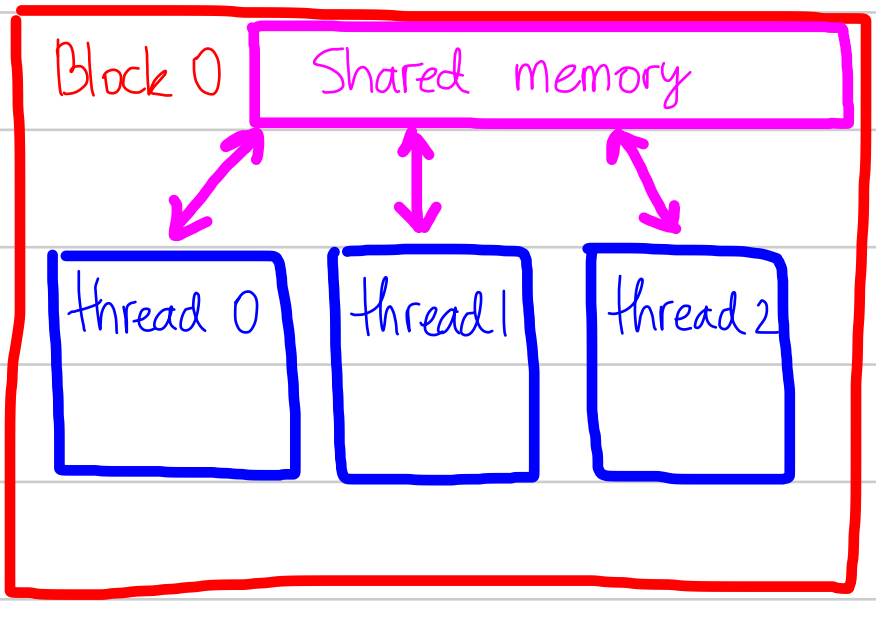
corresp row and col in A and B

I am a speed junkie!

- improvement over CPU code is obvious.
- GPU version does not beat MATLAB.
- unless ....
- We optimize computation by using shared memory



Zoom in:



- Shared memory has very fast access
- can only be accessed by the threads in one specific block

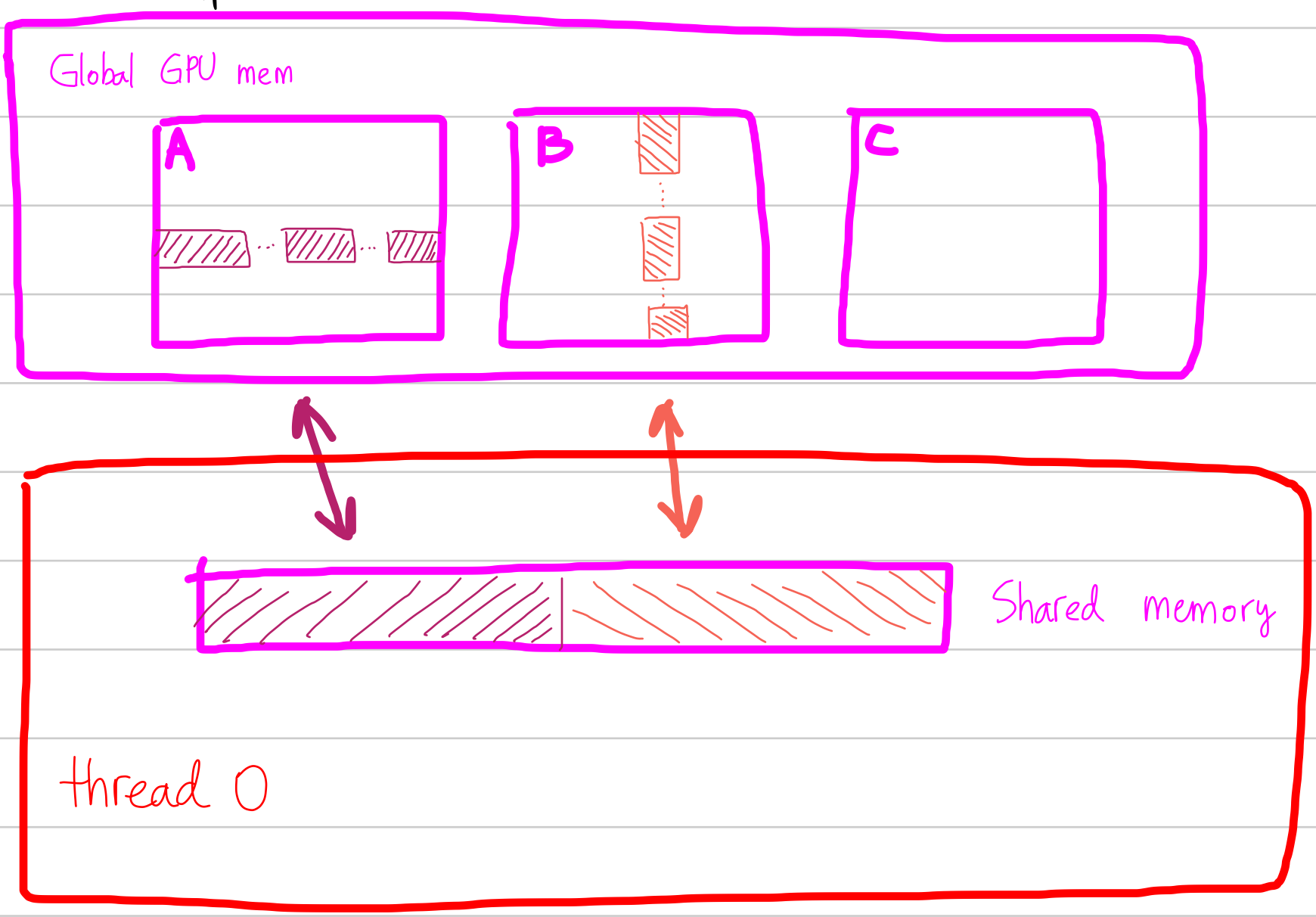
Matrix multiplication ver 2.0

- each thread computes one element of C = AXB

- rather than accessing A and B from global memory,

- do the following:

  - within each block, populate shared memory with required info
  - perform calculations
  - repeat



```
__global__ void matrixMul(Matrix A, Matrix B, Matrix C){

  int tidx=threadIdx.x;
  int tidy=threadIdx.y;

  int bidx=blockIdx.x;
  int bidy=blockIdx.y;

  int row = bidy*TILE+tidy;
  int col = bidx*TILE+tidx;


  double S=0.0;
  int i;
  for(i=0;i<NT;i++){
     __shared__ double shared_A1[TILE][TILE];
     __shared__ double shared_B1[TILE][TILE];

     shared_A1[tidy][tidx]=A.elements[IDX2(row,i*TILE+tidx)];
     shared_B1[tidy][tidx]=B.elements[IDX2(i*TILE+tidy,col)];

     __syncthreads();

     for (int k = 0; k < TILE; k++)
            S += shared_A1[tidy][k] * shared_B1[k][tidx];

     __syncthreads();

  }


  C.elements[IDX2(row,col)] = S;
}
```