Lecture 7: Optimizing CUDA Programs Part 2: Convolution & MatMul

Programming Massively Parallel Multiprocessors and Heterogeneous Systems (Understanding and programming the devices powering AI)

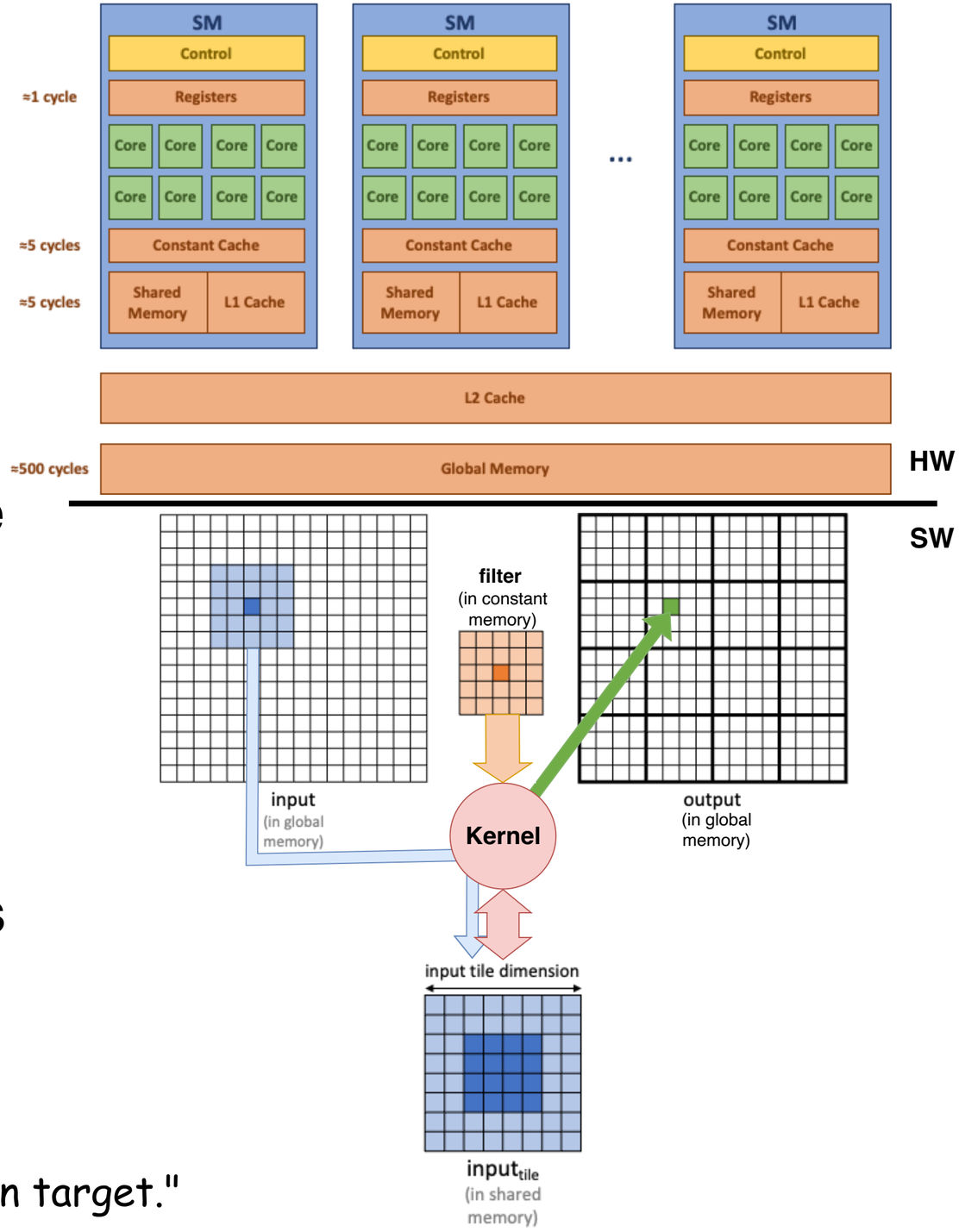
Jonathan Appavoo

Convolution & CUDA

on-SM memory

"Use the ~~force~~ Luke"

- Exploit the on-SM memories to magnify arithmetic intensity (mitigate GMEM Latency)
  - Constant Memory can make a big difference
  - What about the L1 and L2
    - truth is, you have to be smart
  - There are plenty of other approaches (eg, register the kernel, embed in instructions, exploit L1, ...
    - not clear who knows (or can know) what is "optimal"
- Until you measure, nothing is real!



"... just like Beggar's Canyon back home ... stay on target, stay on target."

Matrix Multiplication

WARNING: "Any sufficiently advanced technology is indistinguishable from magic.", Arthur C Clarke, 1969

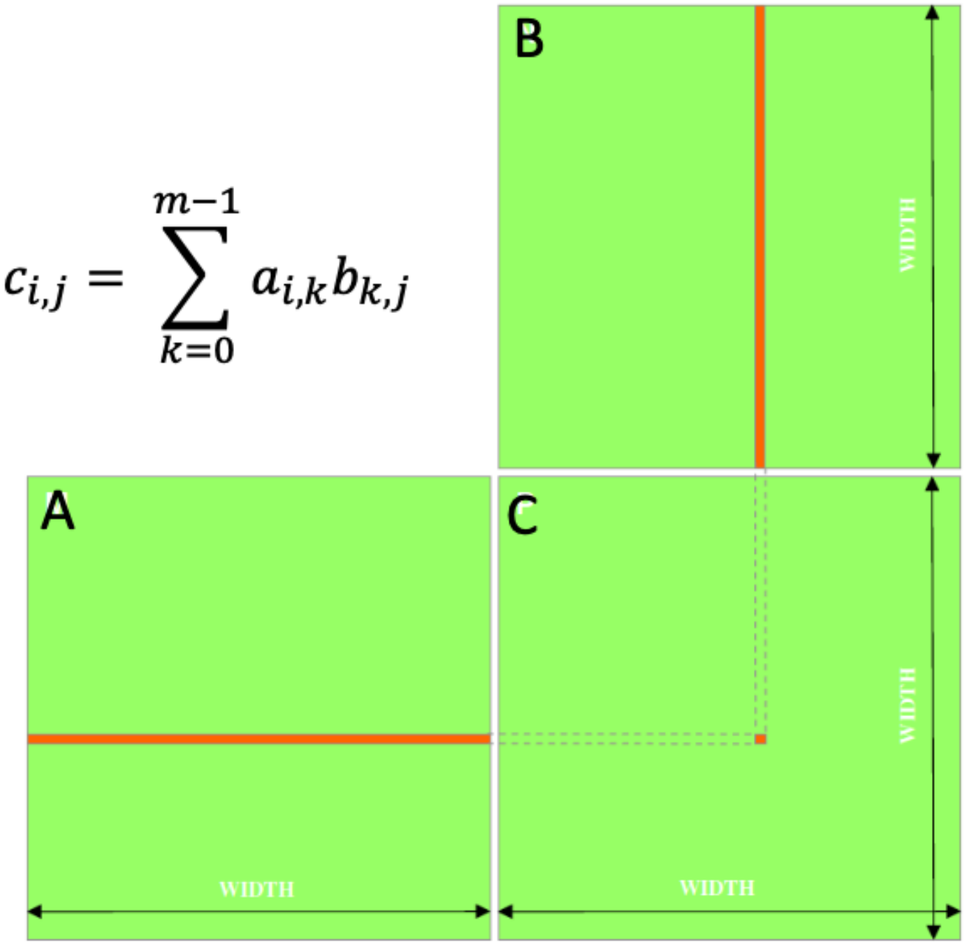
https://en.wikipedia.org/wiki/Clarke%27s_three_laws

Matrix Multiplication: highly parallelizable!

- C= A x B : key computation within many scientific applications particularly those in deep learning
- Compute: $O(N^3)$ -- Memory $O(N^2)$
  - high compute intensity

- each element C dot product of a row of A and a col of B

$$c_{i,j} = \sum_{k=0}^{m-1} a_{i,k} b_{k,j}$$



GPU challenge: minimize GMEM accesses

How many GMEM accesses?

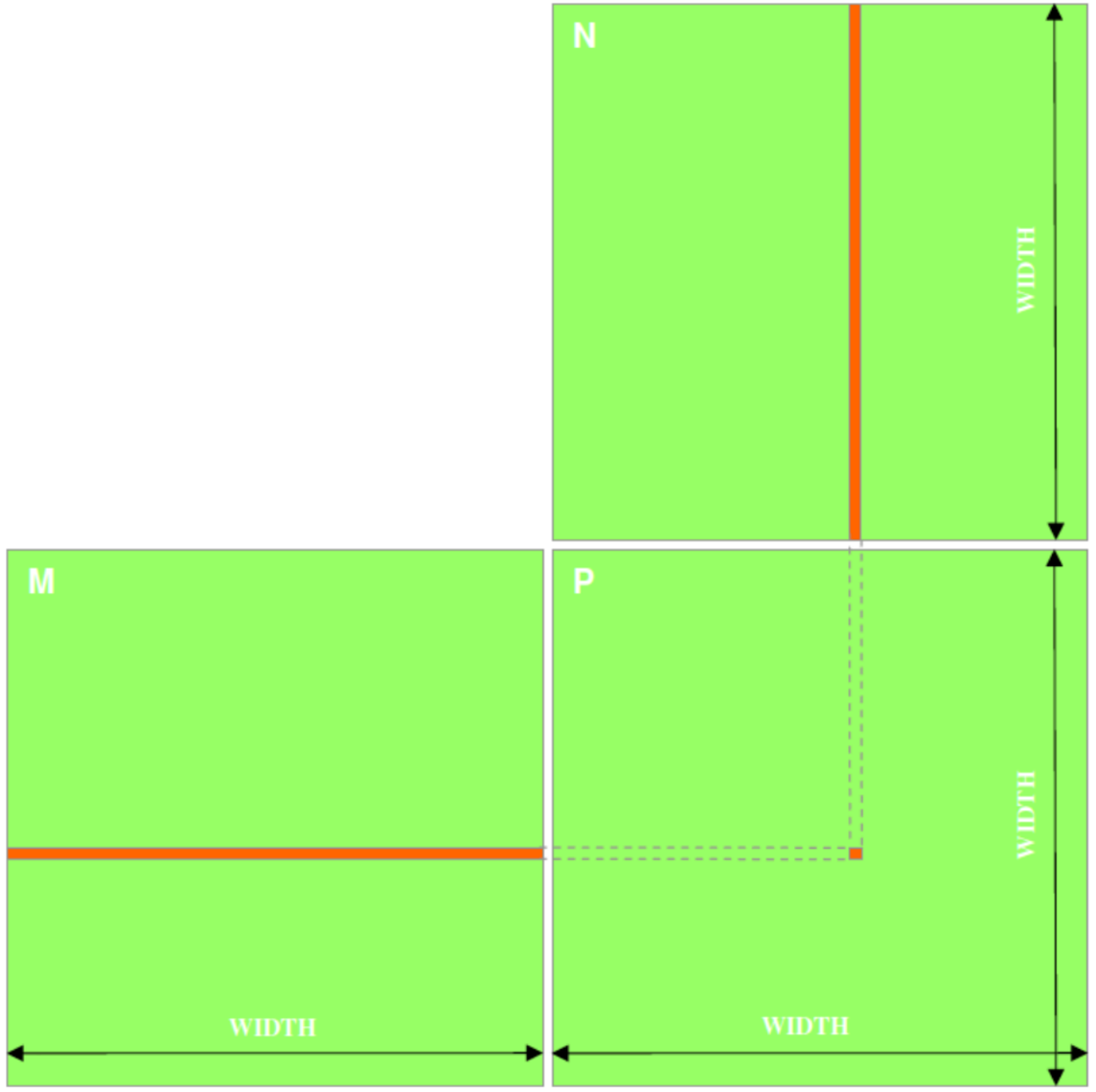
```
for (int i = 0; i < M; ++i)
    for (int j = 0; j < N; ++j)
        for (int k = 0; k < K; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

For square matrices:

$3 * N^3$

Idea: use SMEM

?

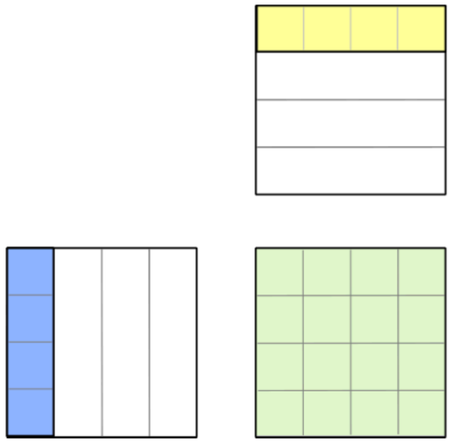


Idea: use SMEM

- But: A & B & C will not fit in SMEM ☹
- Idea: rearrange loops

```
for (int i = 0; i < M; ++i)
   for (int j = 0; j < N; ++j)
      for (int k = 0; k < K; ++k)
         C[i][j] += A[i][k] * B[k][j];
```

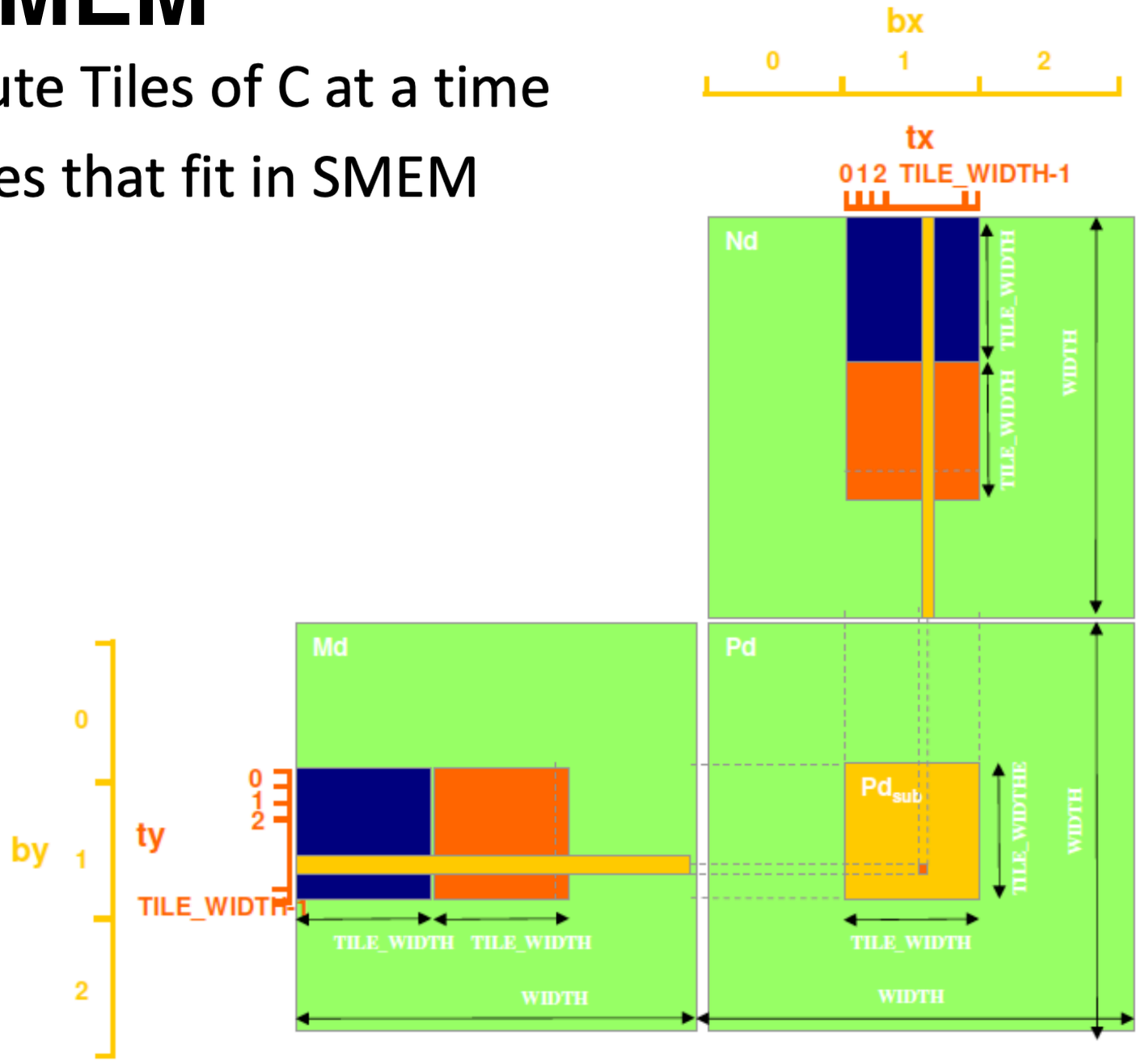
```
for (int k = 0; k < K; ++k)
   for (int i = 0; i < M; ++i)
      for (int j = 0; j < N; ++j)
         C[i][j] += A[i][k] * B[k][j];
```



- copy col of A & row of B into SMEM
- compute its outer product and accumulate in C
- col of A & row of B never used again
- But all of C accessed many time ➔ doesn't fit in SMEM

Idea: use SMEM

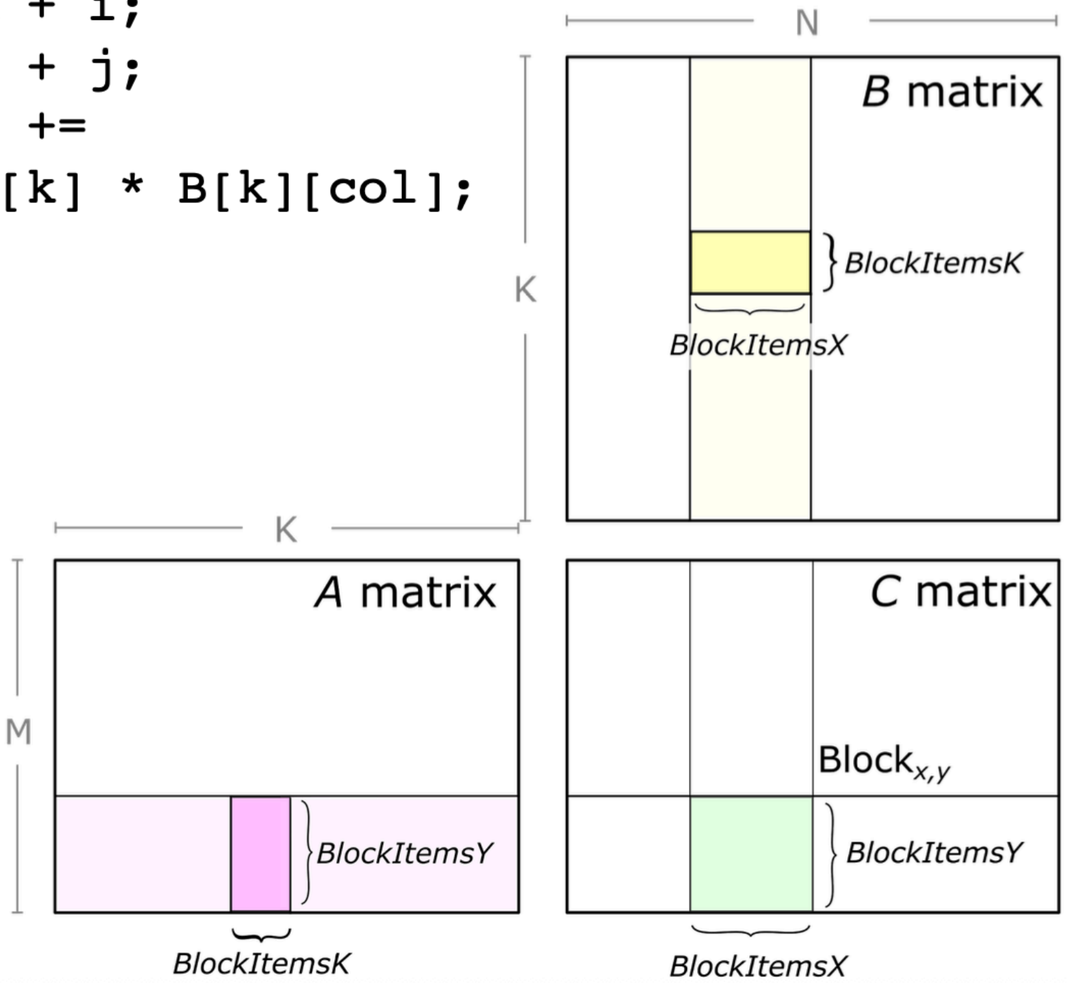
- Compute Tiles of C at a time
- Use tiles that fit in SMEM
- . . .



Tiled Computation

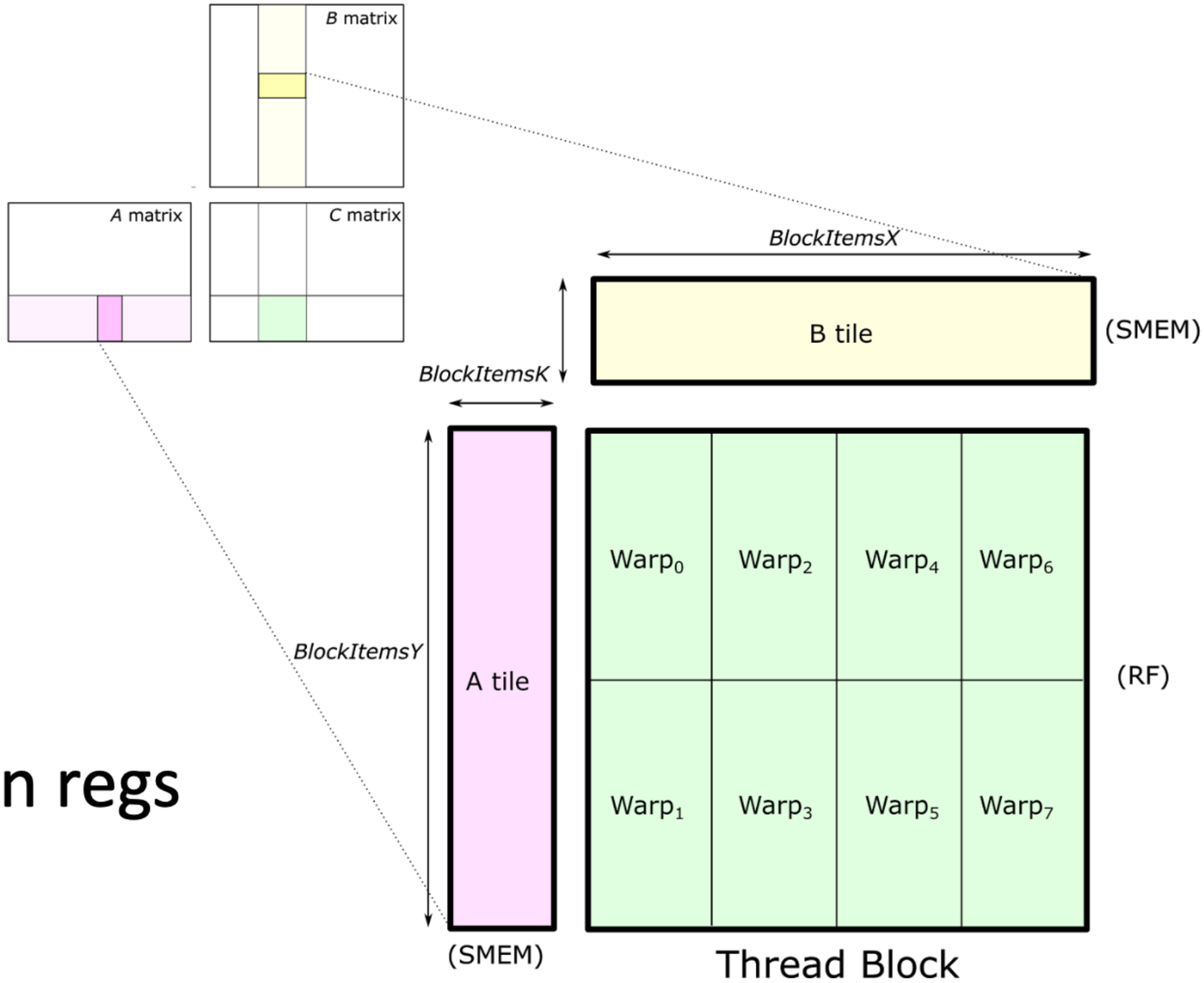
```
for (int m = 0; m < M; m += Mtile) // iterate over M dim
   for (int n = 0; n < N; n += Ntile) // iterate over N dim
      for (int k = 0; k < K; ++k)
         for (int i = 0; i < Mtile; ++i) // compute one tile
            for (int j = 0; j < Ntile; ++j) {
               int row = m + i;
               int col = n + j;
               C[row][col] +=
                    A[row][k] * B[k][col];
             }
```

- For each C-tile:
  tiles of A & B fetched
  exactly once
- Ensure the 3 tiles
  fit in SMEM



Further considerations

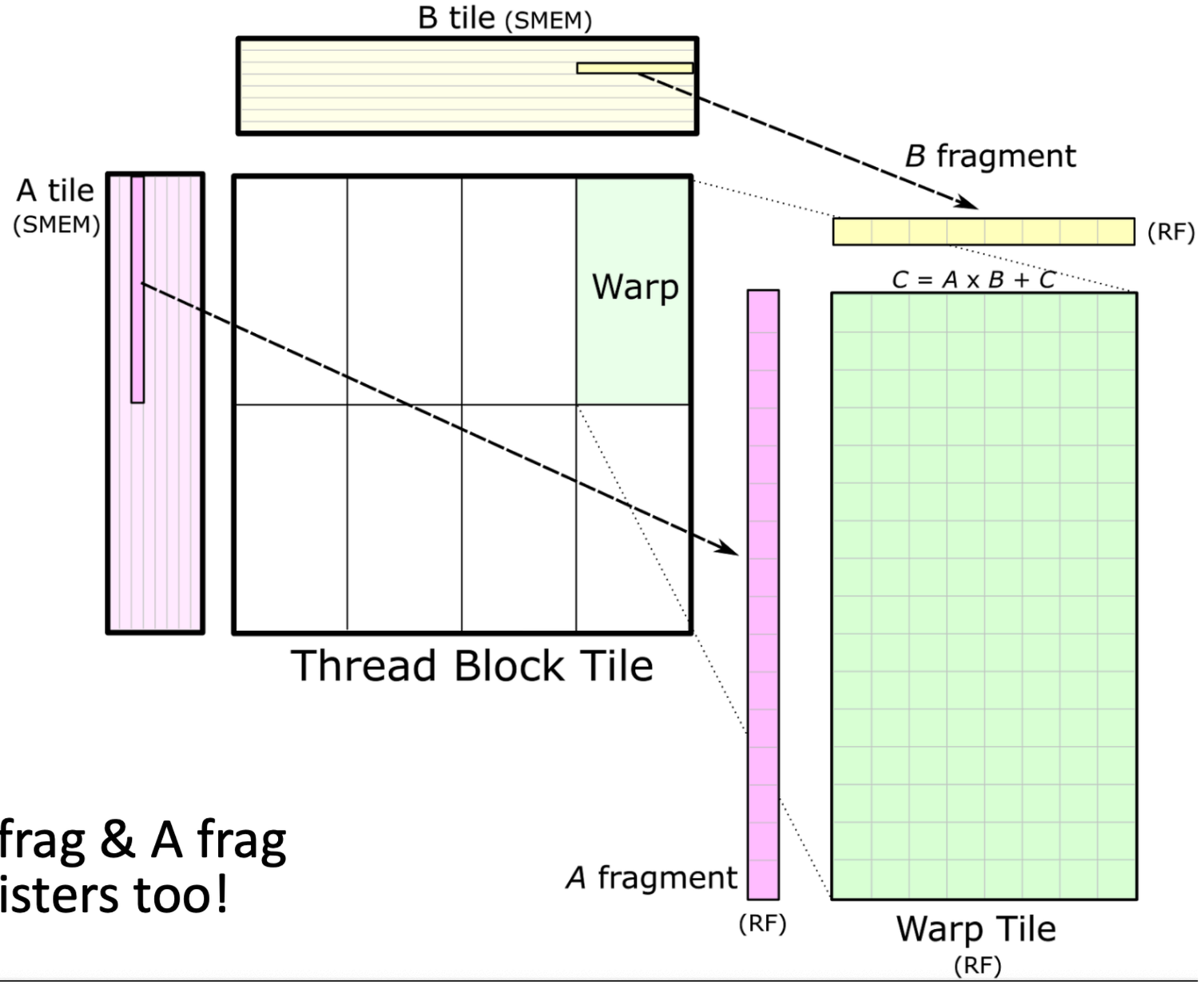
- Decide what each warp does:



- keep c-tile in regs

Further considerations II

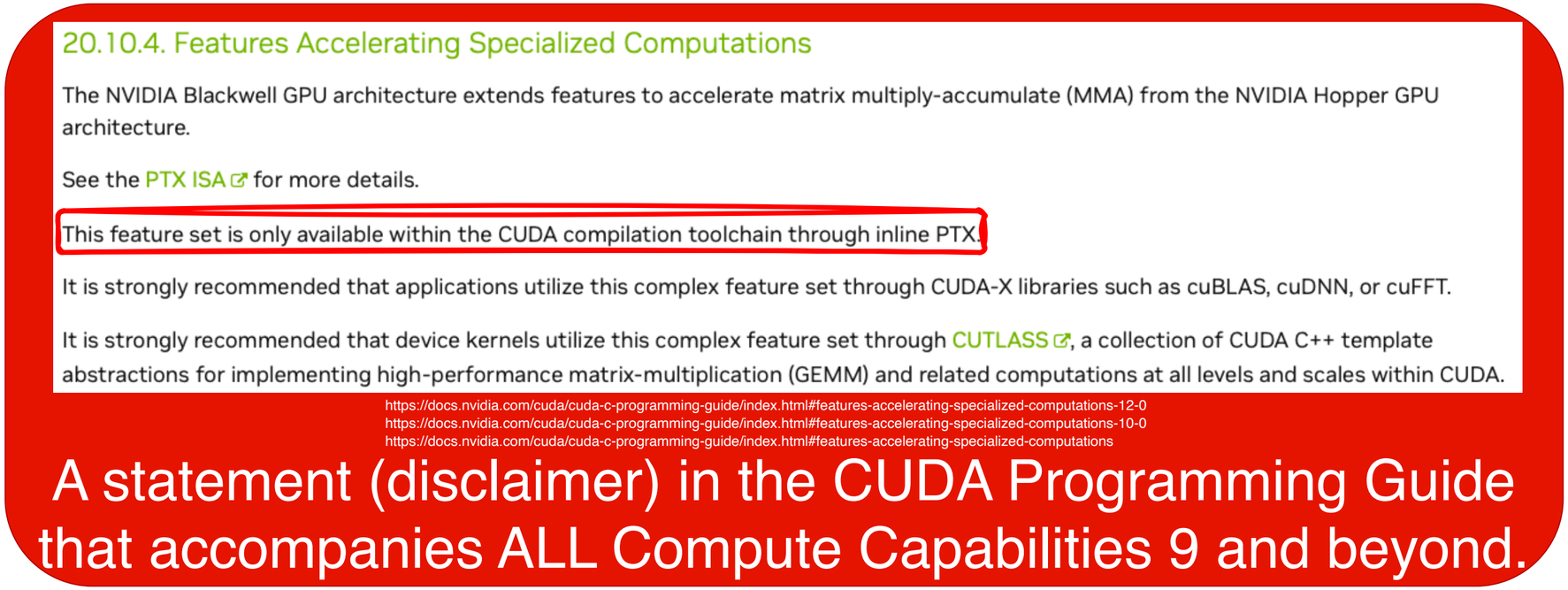
- Optimize warps:



keep B frag & A frag
in registers too!

Bottom line for Matrix Multiplication: Abracadabra!

(and maybe any core routines)



20.10.4. Features Accelerating Specialized Computations

The NVIDIA Blackwell GPU architecture extends features to accelerate matrix multiply-accumulate (MMA) from the NVIDIA Hopper GPU architecture.

See the PTX ISA for more details.

This feature set is only available within the CUDA compilation toolchain through inline PTX.

It is strongly recommended that applications utilize this complex feature set through CUDA-X libraries such as cuBLAS, cuDNN, or cuFFT.

It is strongly recommended that device kernels utilize this complex feature set through CUTLASS, a collection of CUDA C++ template abstractions for implementing high-performance matrix-multiplication (GEMM) and related computations at all levels and scales within CUDA.

https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#features-accelerating-specialized-computations-12-0
https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#features-accelerating-specialized-computations-10-0
https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#features-accelerating-specialized-computations

A statement (disclaimer) in the CUDA Programming Guide that accompanies ALL Compute Capabilities 9 and beyond.

- Surprisingly non-trivial
- Hard to get right from a performance PoV
- Nvidia CUBLAS cannot be beat!
- CUTLESS library is closest to CUBLAS
  - See https://github.com/NVIDIA/cutlass