CS/EE 217

GPU Architecture and Parallel Programming

Lecture 6: DRAM Bandwidth

Objective

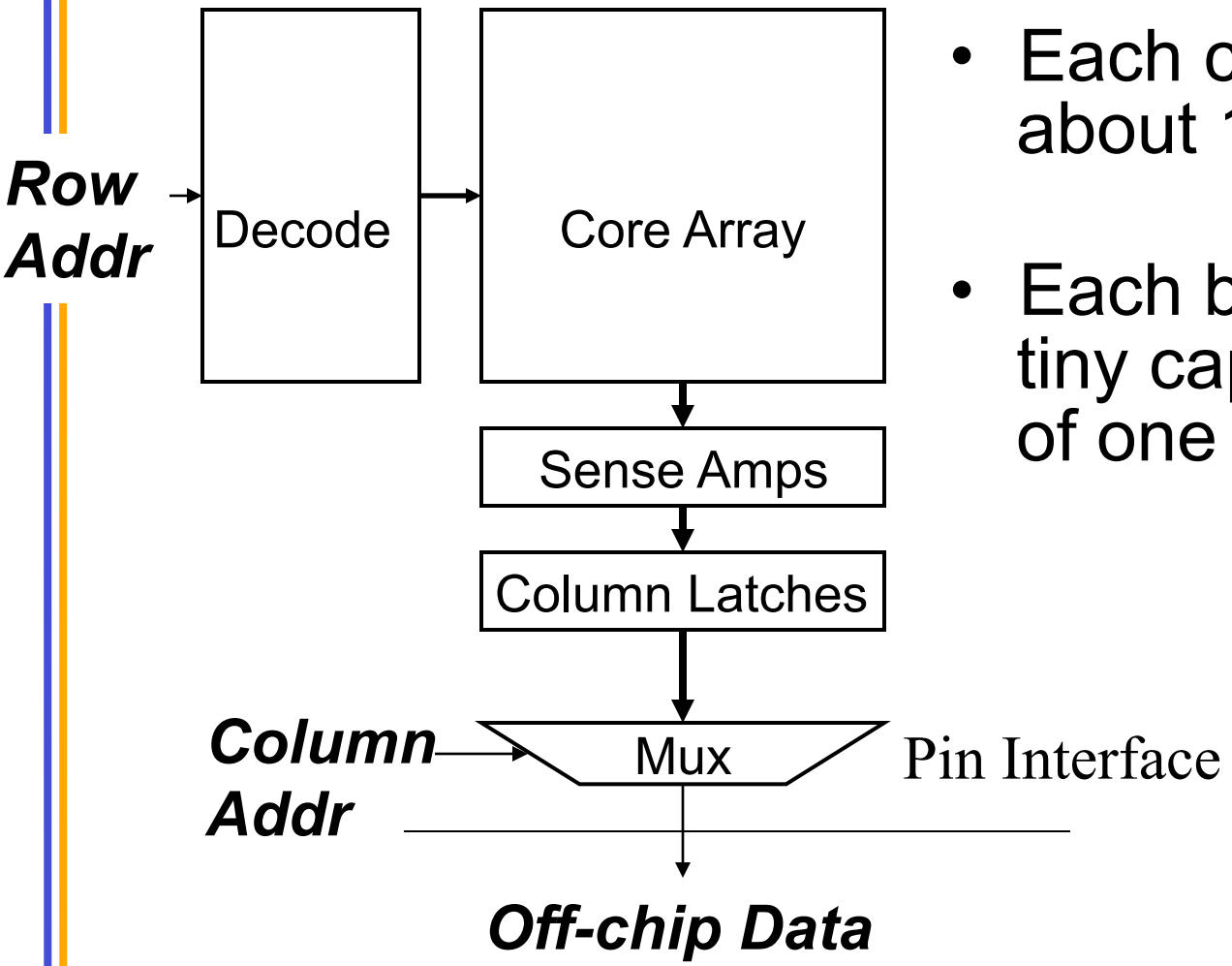
- To understand DRAM bandwidth
  - Cause of the DRAM bandwidth problem
  - Programming techniques that address the problem: memory coalescing, corner turning,

Global Memory (DRAM) Bandwidth

Ideal

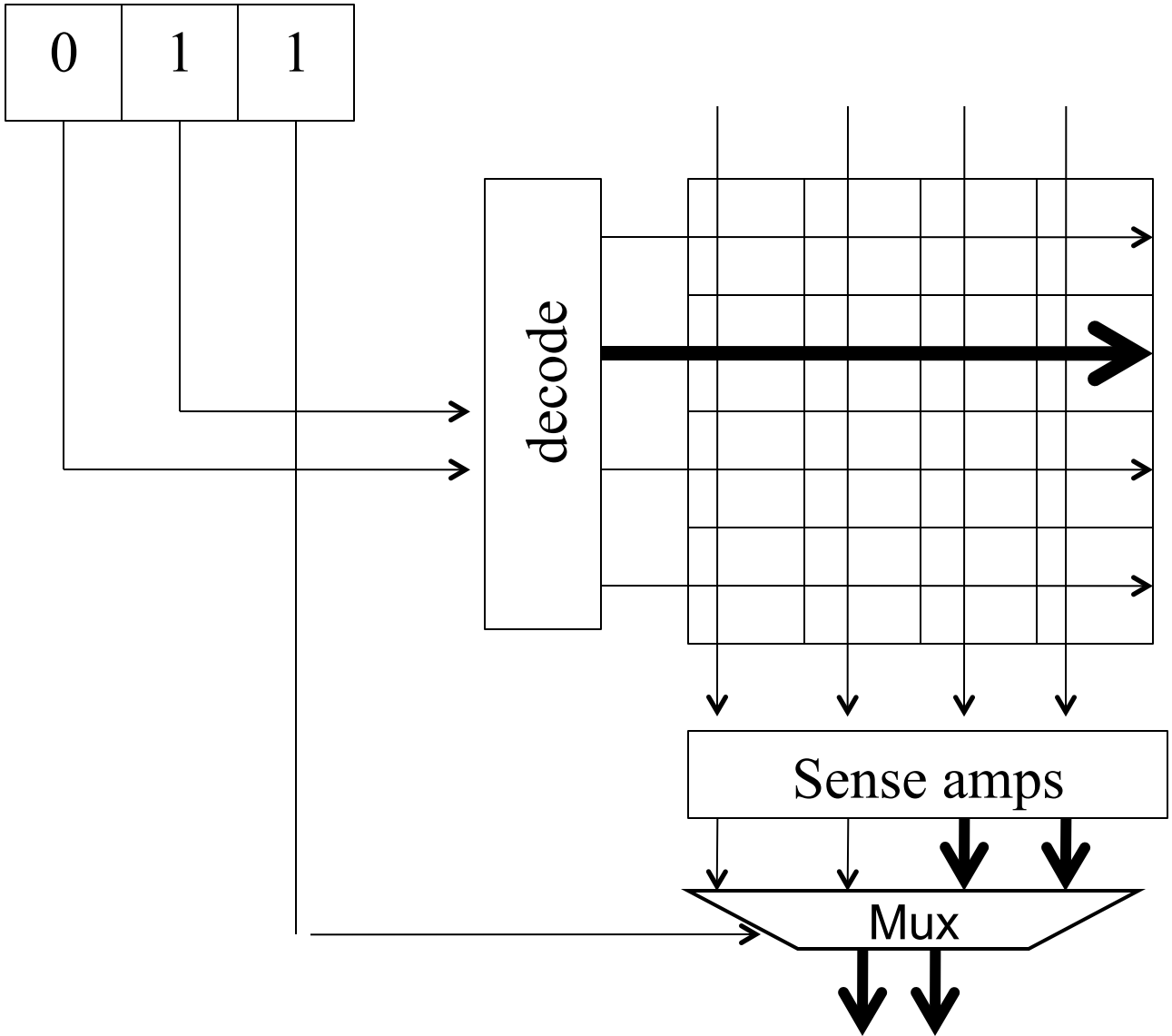
Reality

DRAM Bank Organization



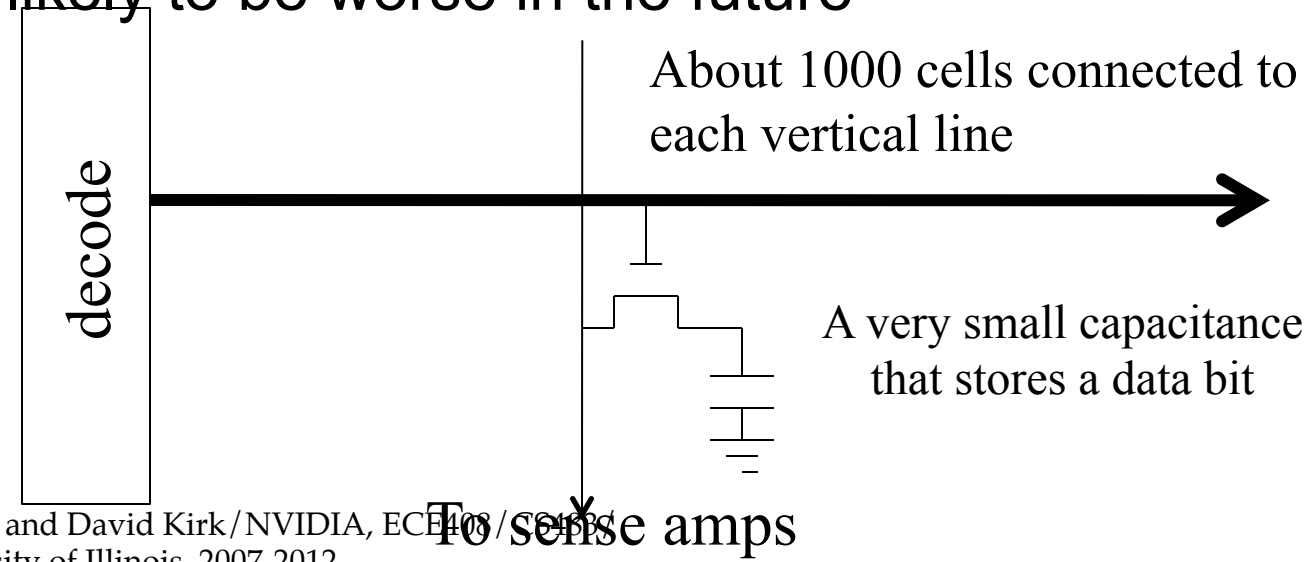
- Each core array has about 1M bits
- Each bit is stored in a tiny capacitor, made of one transistor

A very small (8x2 bit) DRAM Bank



DRAM core arrays are slow.

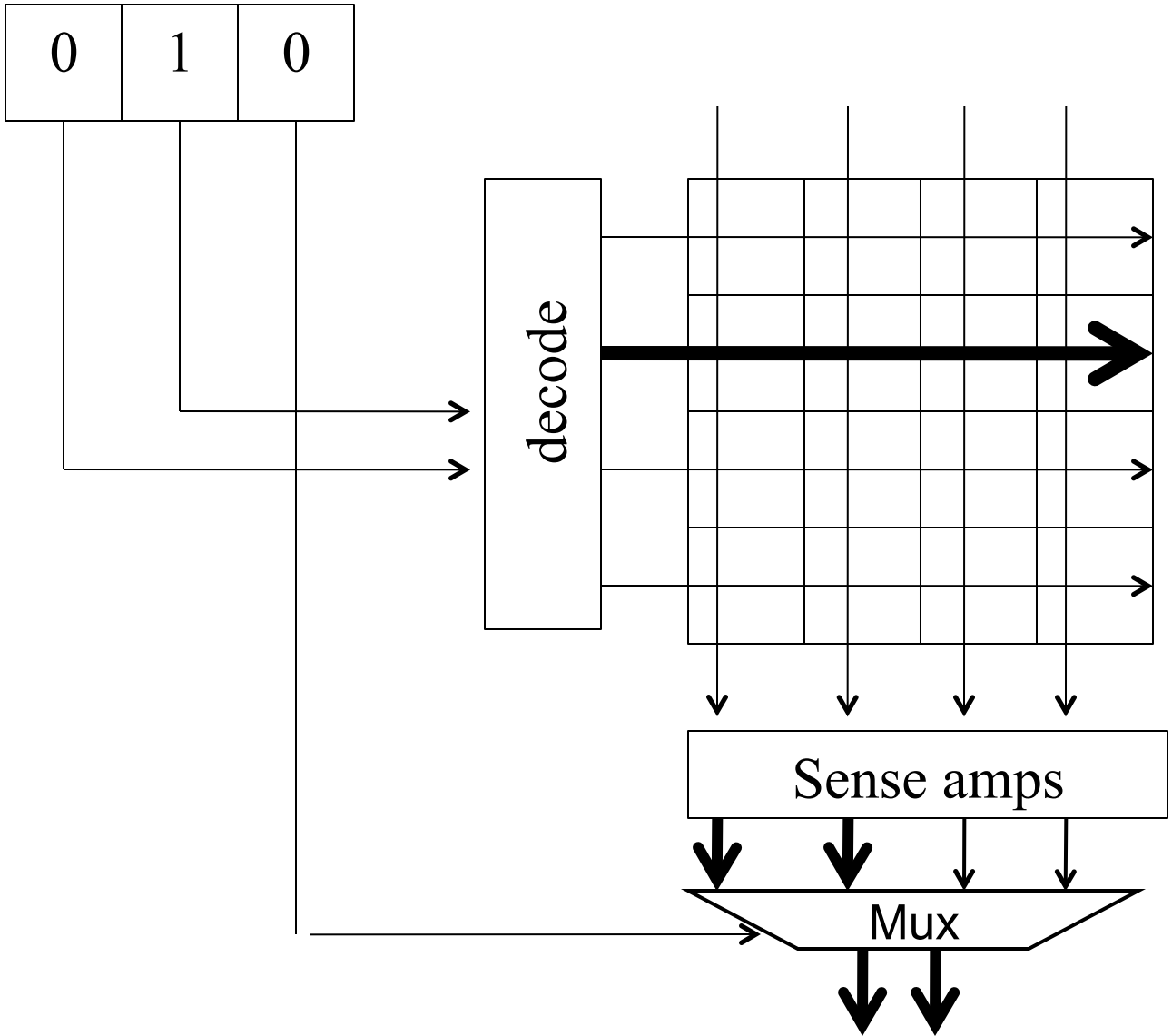
- Reading from a cell in the core array is a very slow process
  - DDR: Core speed = ½ interface speed
  - DDR2/GDDR3: Core speed = ¼ interface speed
  - DDR3/GDDR4: Core speed = ⅛ interface speed
  - … likely to be worse in the future



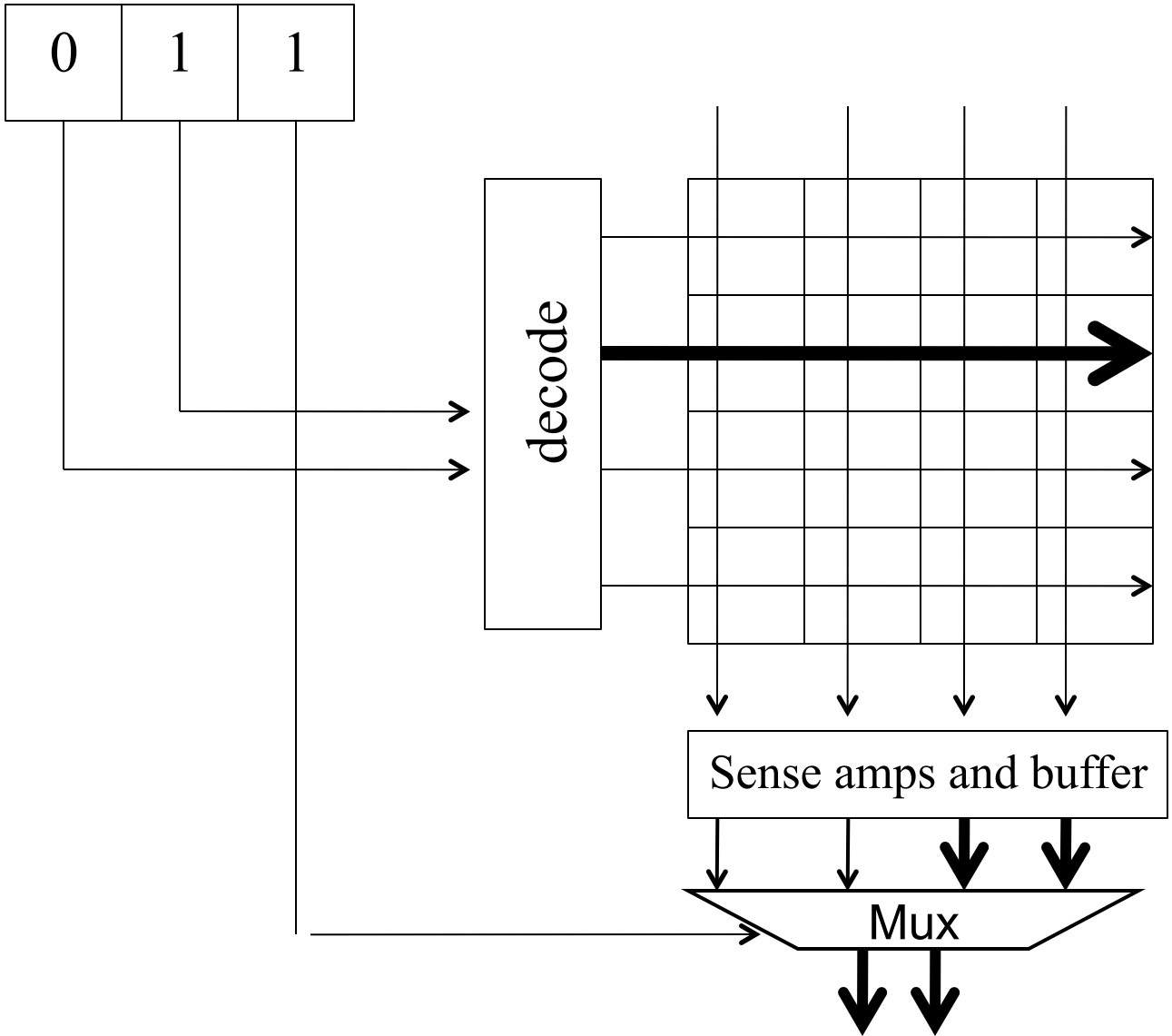
DRAM Bursting.

- For DDR{2,3} SDRAM cores clocked at 1/N speed of the interface:
  - Load (N × interface width) of DRAM bits from the same row at once to an internal buffer, then transfer in N steps at interface speed
  - DDR2/GDDR3: buffer width = 4X interface width

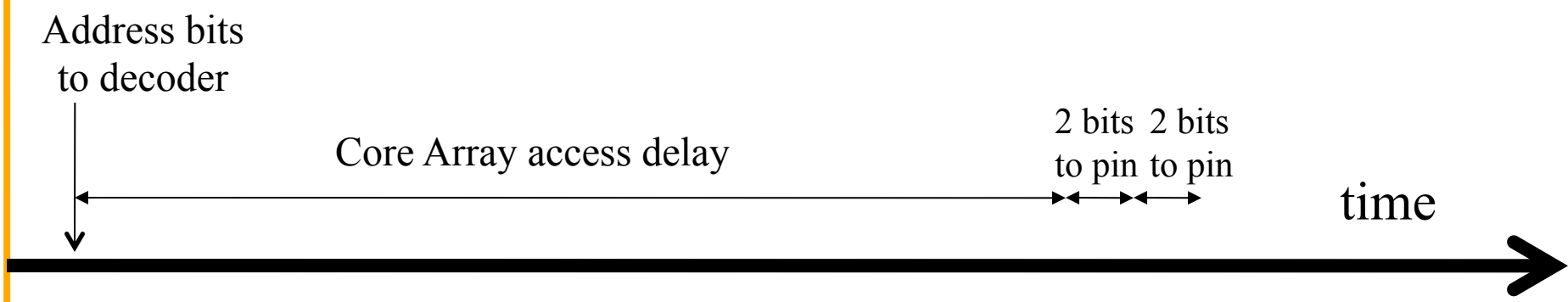
DRAM Bursting



DRAM Bursting



DRAM Bursting for the 8x2 Bank

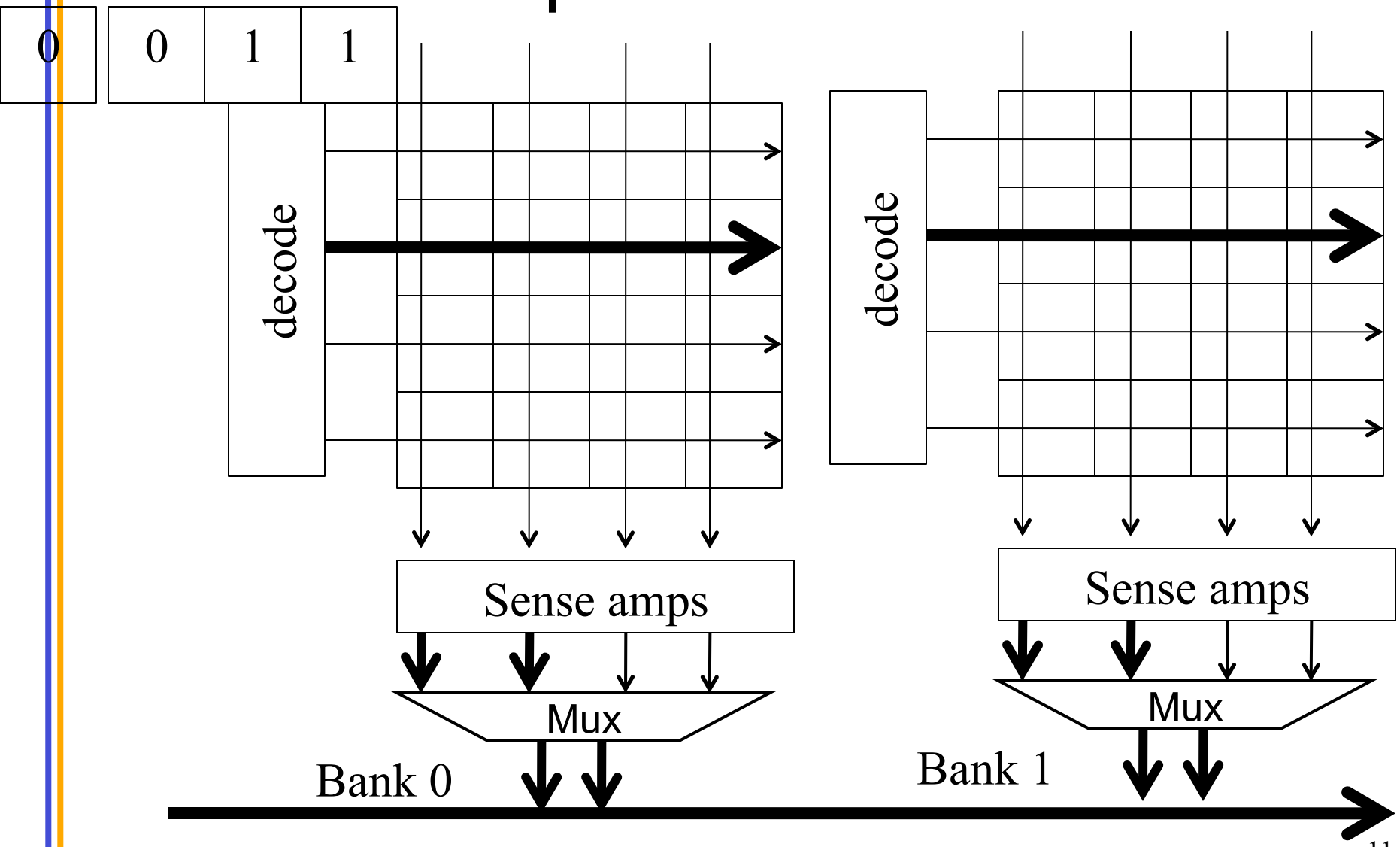


Non-burst timing

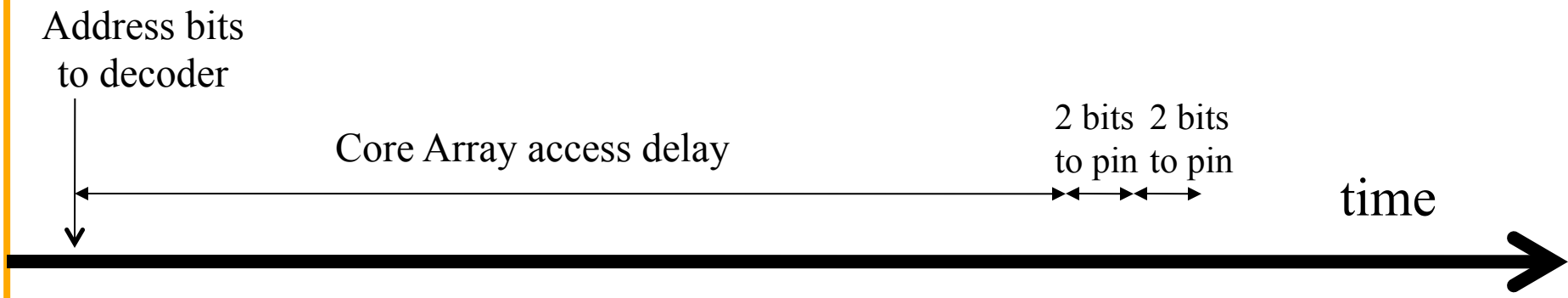
Burst timing

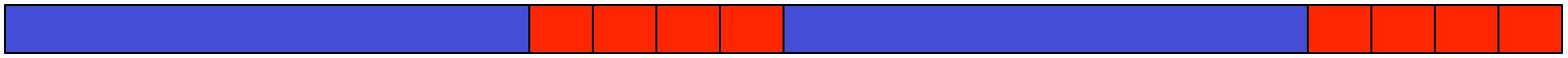
Modern DRAM systems are designed to be always accessed in burst mode. Burst bytes are transferred but discarded when accesses are not to sequential locations.

Multiple DRAM Banks

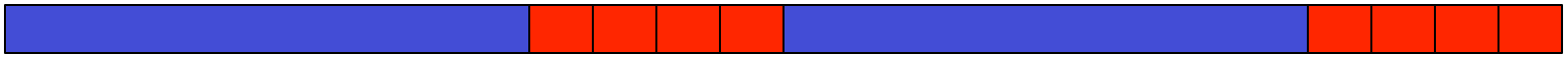


DRAM Bursting for the 8x2 Bank





Single-Bank burst timing, dead time on interface



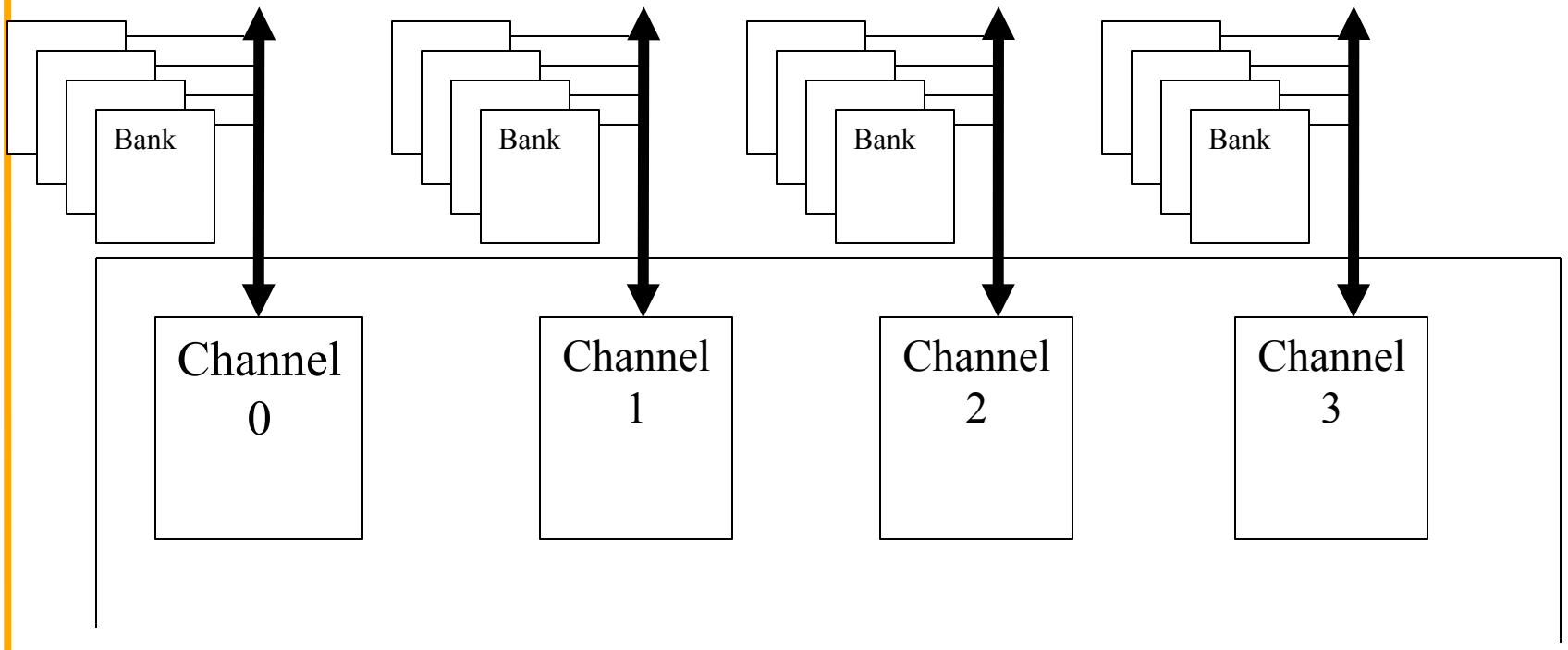
Multi-Bank burst timing, reduced dead time

First-order Look at the GPU off-chip memory subsystem

- nVidia GTX280 GPU:
  - Peak global memory bandwidth = 141.7GB/s
- Global memory (GDDR3) interface @ 1.1GHz
  - (Core speed @ 276Mhz)
  - For a typical 64-bit interface, we can sustain only about 17.6 GB/s (Recall DDR - 2 transfers per clock)
  - We need a lot more bandwith (141.7 GB/s) – thus 8 memory channels

Multiple Memory Channels

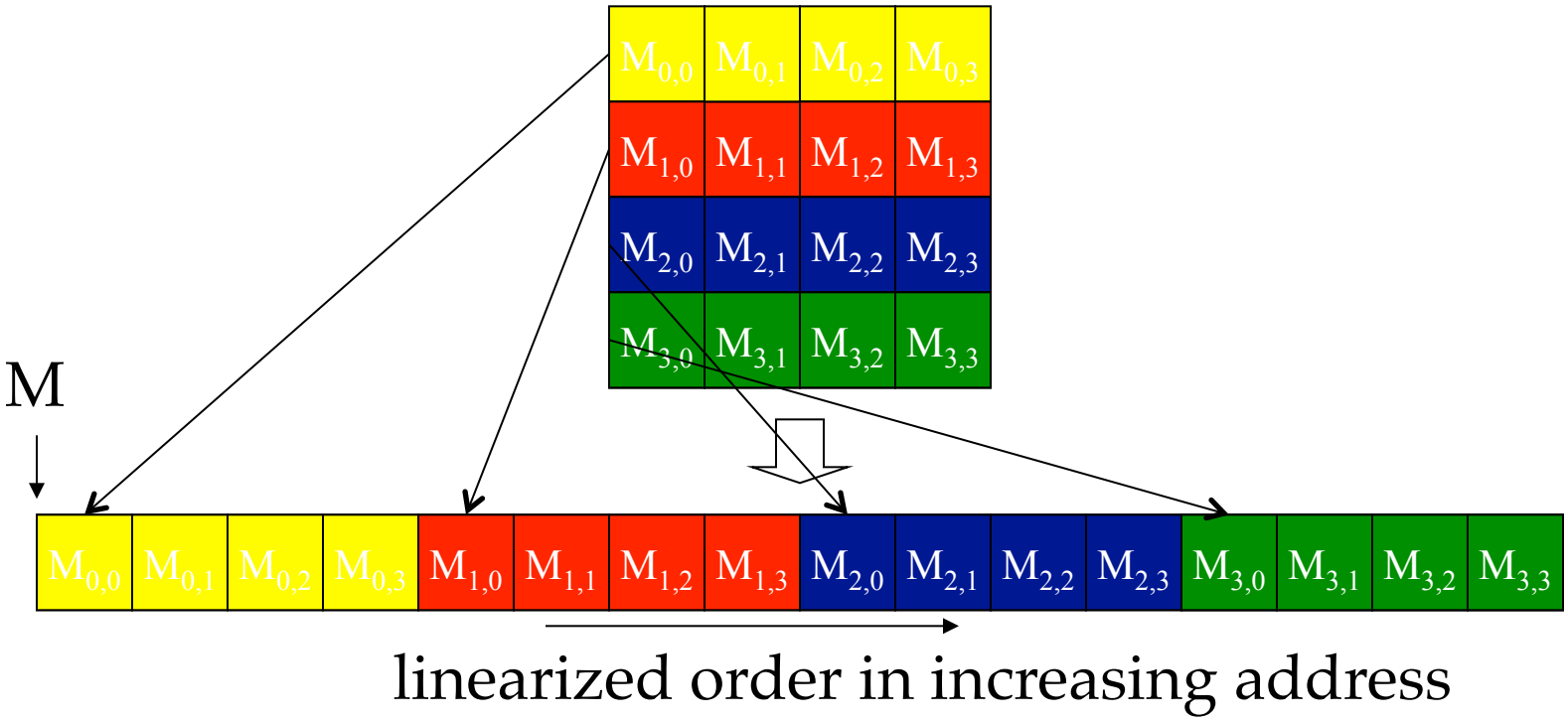
- Divide the memory address space into N parts
  - N is number of memory channels
  - Assign each portion to a channel



Memory Controller Organization of a Many-Core Processor

- GTX280: 30 Stream Multiprocessors (SM) connected to 8-channel DRAM controllers through interconnect
  - DRAM controllers are interleaved
  - Within DRAM controllers (channels), DRAM banks are interleaved for incoming memory requests

Placing a 2D C array into linear memory space



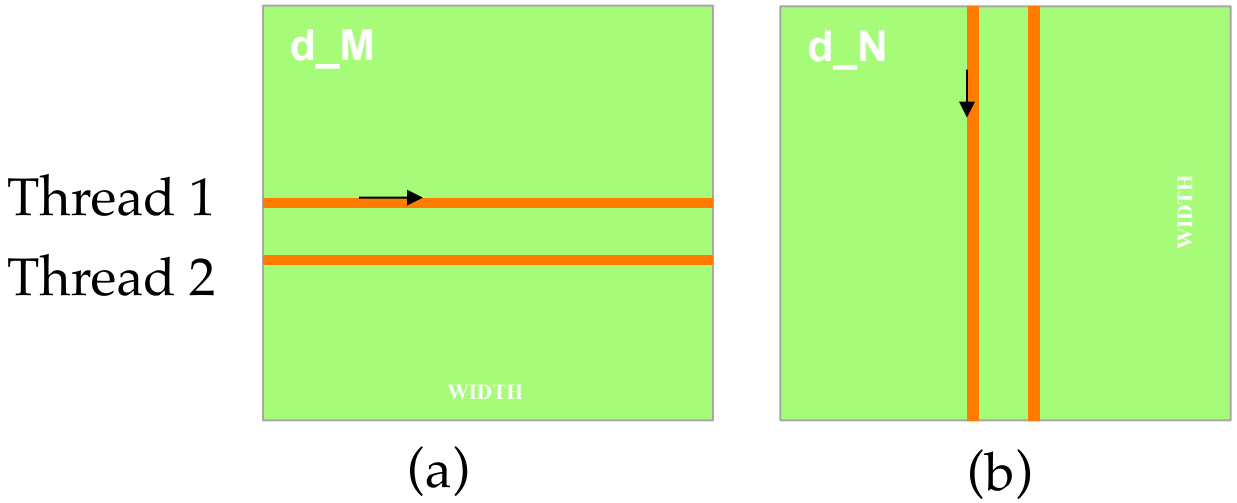
Base Matrix Multiplication Kernel

```
__global__ void MatrixMulKernel(float* d_M, float* d_N, float* d_P, int Width)
{
// Calculate the row index of the Pd element and M
int Row = blockIdx.y*TILE_WIDTH + threadIdx.y;
// Calculate the column idenx of Pd and N
int Col = blockIdx.x*TILE_WIDTH + threadIdx.x;

float Pvalue = 0;
// each thread computes one element of the block sub-matrix
for (int k = 0; k < Width; ++k)
  Pvalue += d_M[Row*Width+k]* d_N[k*Width+Col];

d_P[Row*Width+Col] = Pvalue;
}
```

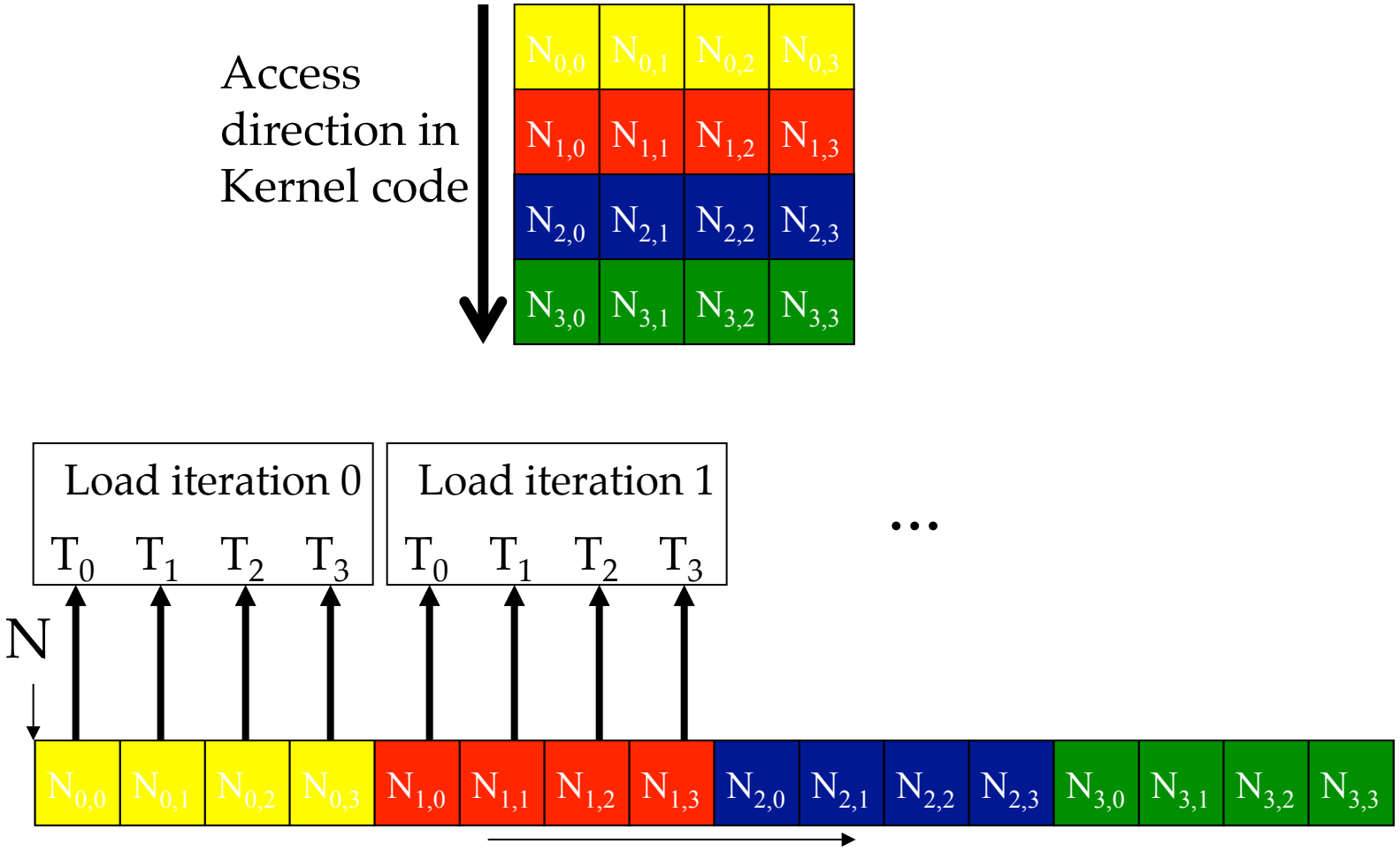
Two Access Patterns



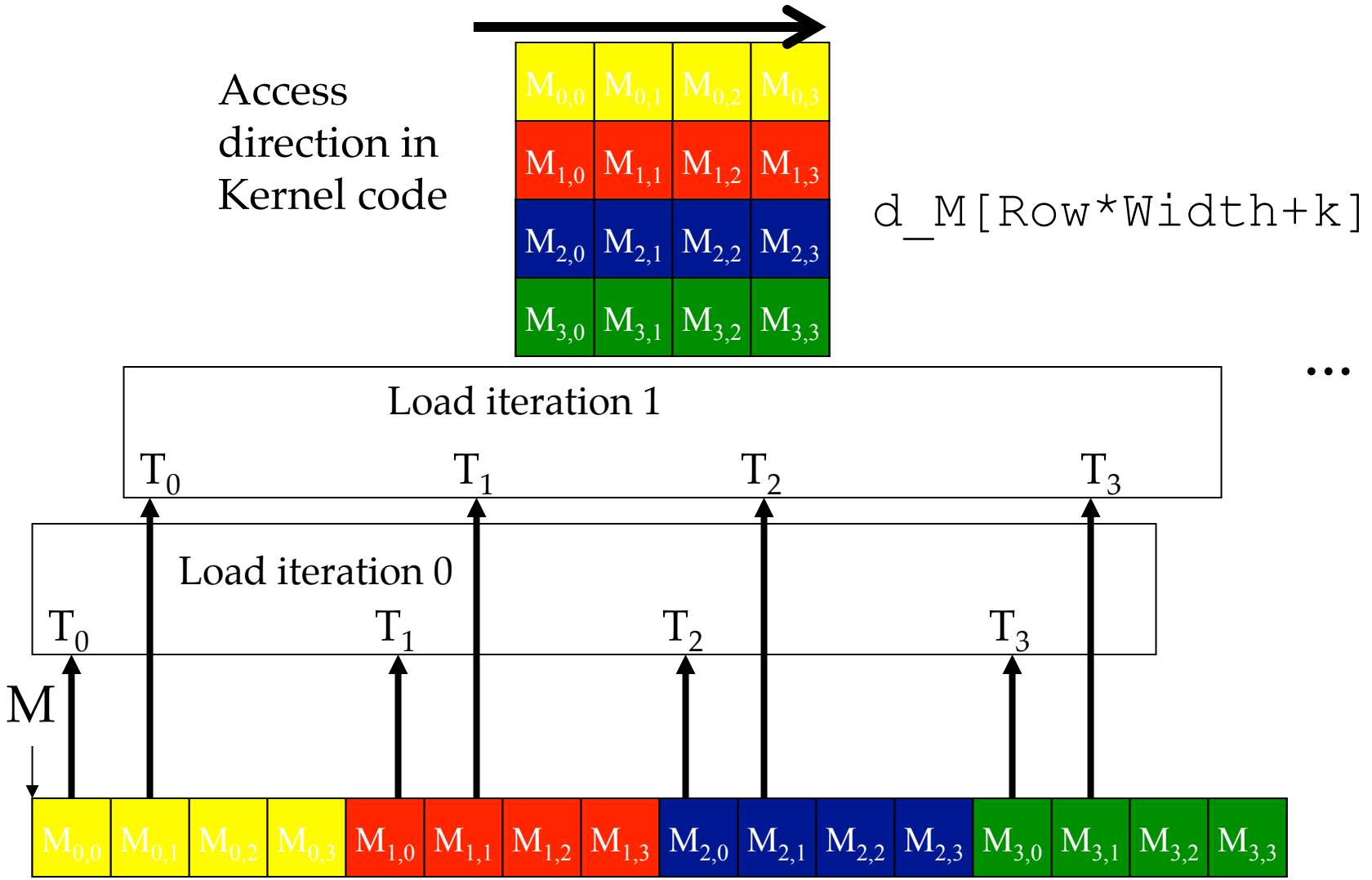
`d_M[Row*Width+k]` `d_N[k*Width+Col]`

k is loop counter in the inner product loop of the kernel code

N accesses are coalesced.



M accesses are not coalesced.



```
__global__ void MatrixMulKernel(float* d_M, float* d_N, float* d_P, int Width)
{
1.   __shared__float Mds[TILE_WIDTH][TILE_WIDTH];
2.   __shared__float Nds[TILE_WIDTH][TILE_WIDTH];
3.   int bx = blockIdx.x;  int by = blockIdx.y;
4.   int tx = threadIdx.x; int ty = threadIdx.y;
// Identify the row and column of the d_P element to work on
5.   int Row = by * TILE_WIDTH + ty;
6.   int Col = bx * TILE_WIDTH + tx;

7.    float Pvalue = 0;
// Loop over the d_M and d_N tiles required to compute the d_P element
8.    for (int m = 0; m < Width/TILE_WIDTH; ++m) {

// Coolaborative loading of d_M and d_N tiles into shared memory
9.     Mds[tx][ty] = d_M[Row*Width + m*TILE_WIDTH+tx];
10.    Nds[tx][ty] = d_N[(m*TILE_WIDTH+ty)*Width + Col];
11.    __syncthreads();
12.   for (int k = 0; k < TILE_WIDTH; ++k)
13.      Pvalue += Mds[tx][k] * Nds[k][ty];
14.    __synchthreads();
   }
15.   d_P[Row*Width+Col] = Pvalue;
}
```

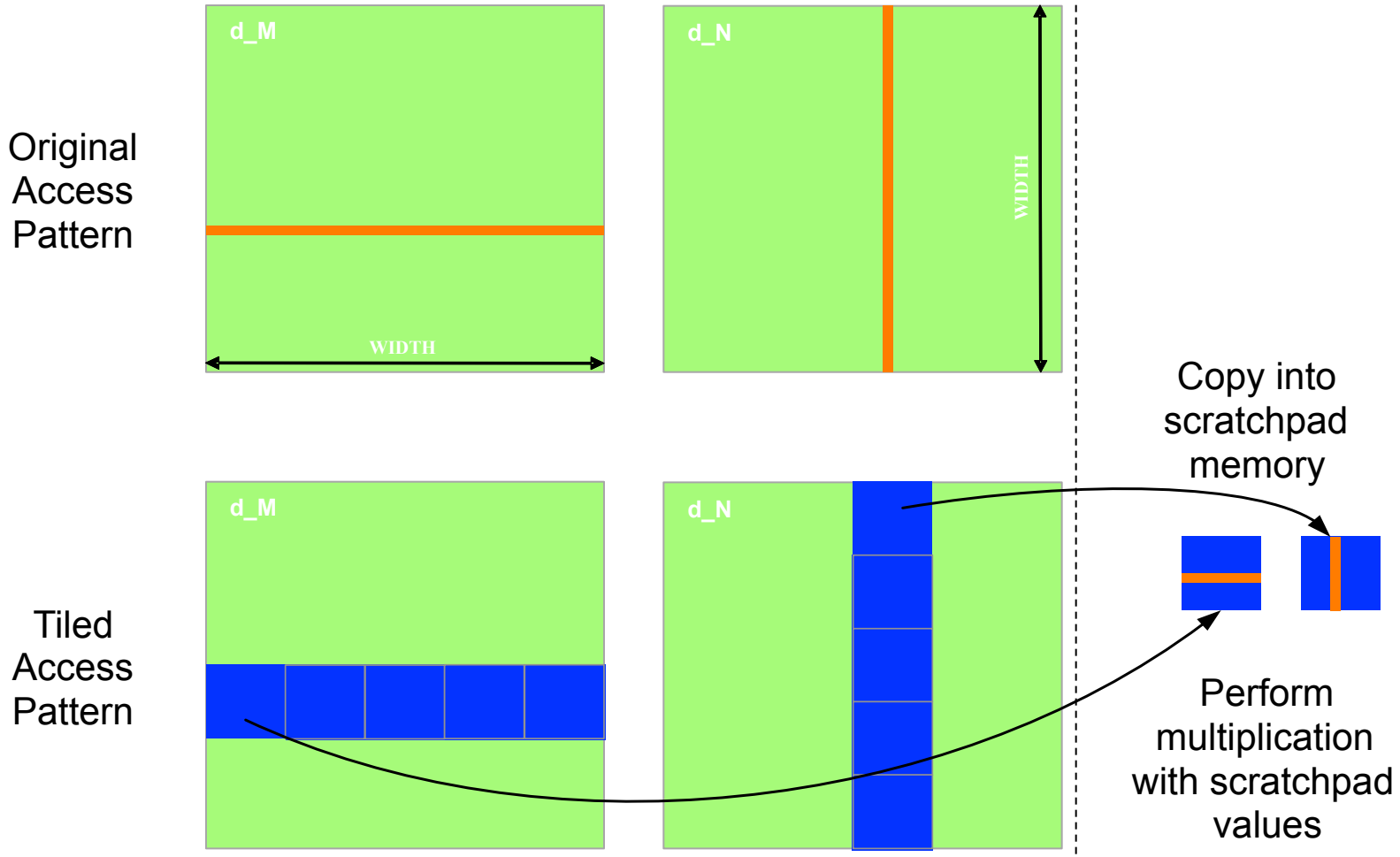


Figure 6.10: Using shared memory to enable coalescing

ANY MORE QUESTIONS?
READ CHAPTER 6