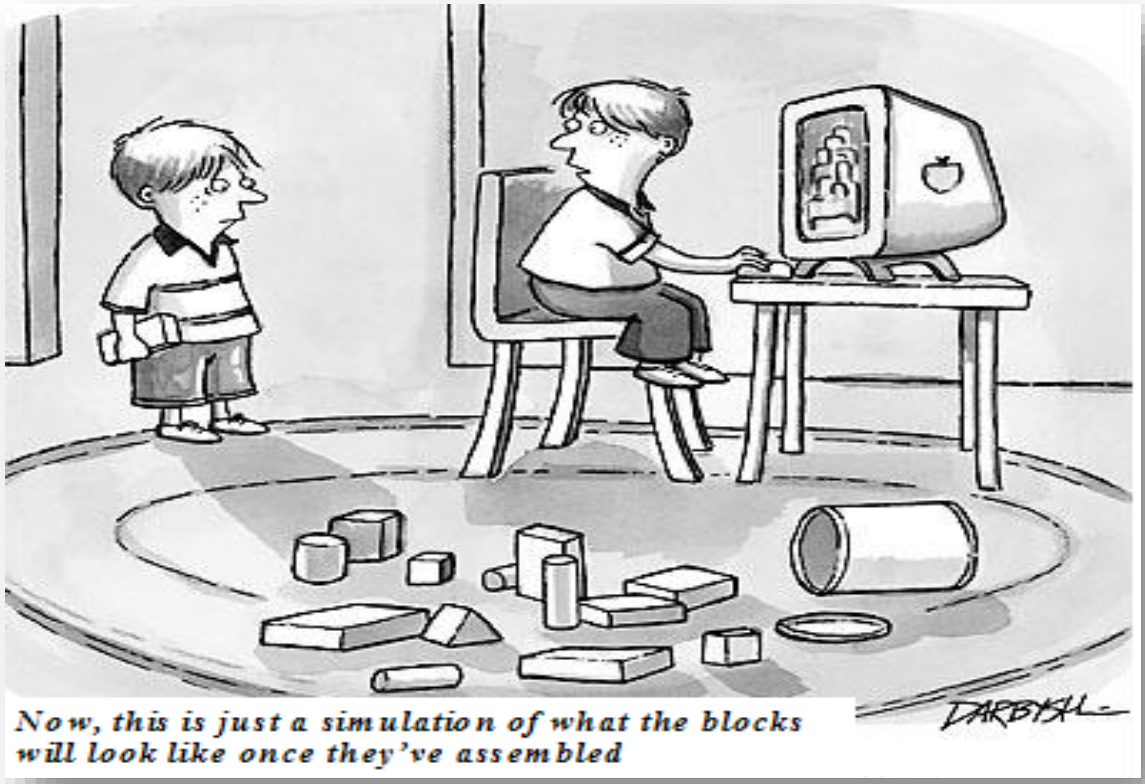
ECE569
Module 25



- Matrix Multiplication – Boundary Conditions

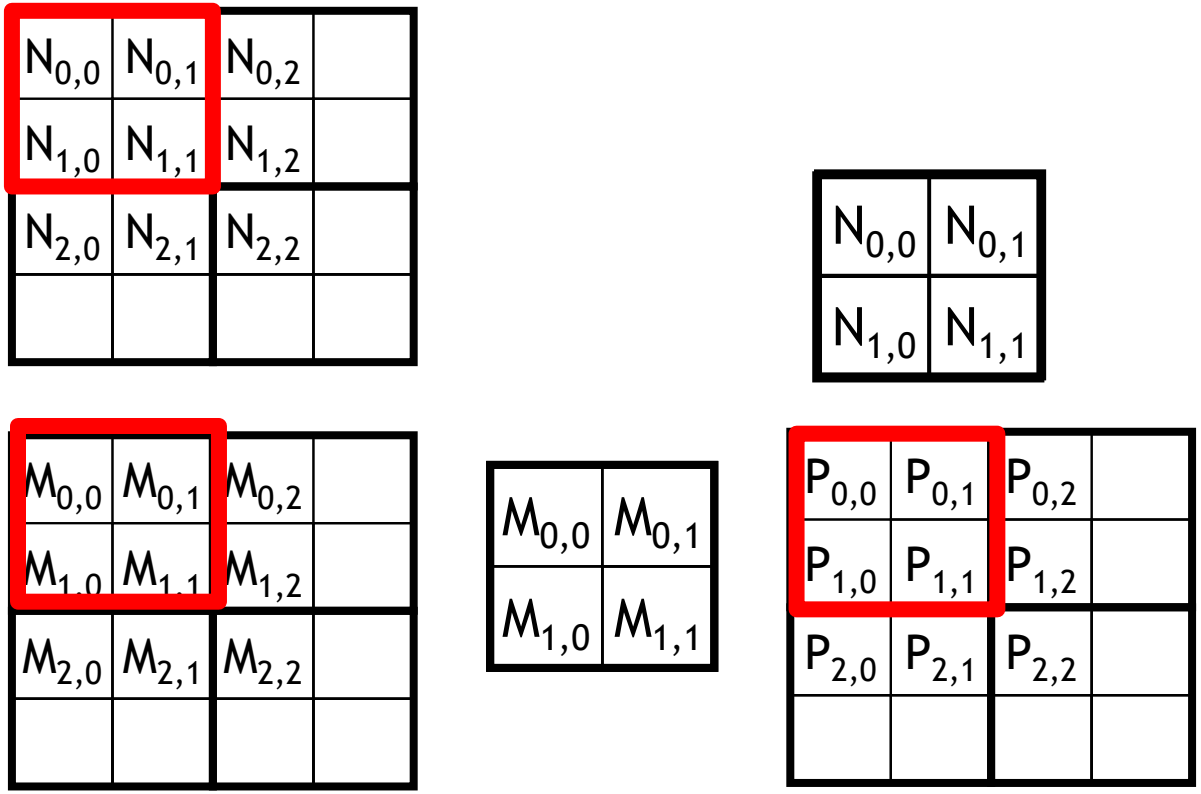
Tiled Matrix Multiplication Kernel

```
__global__ void MatrixMulKernel(float* M, float* N, float* P, int Width)
{
  __shared__ float ds_M[TILE_WIDTH][TILE_WIDTH];
  __shared__ float ds_N[TILE_WIDTH][TILE_WIDTH];
  int bx = blockIdx.x;  int by = blockIdx.y;
  int tx = threadIdx.x; int ty = threadIdx.y;

  int Row = by * TILE_WIDTH + ty;
  int Col = bx * TILE_WIDTH + tx;
  float Pvalue = 0;

 // Loop over the M and N tiles required to compute the P element
 for (int p = 0; p < Width/TILE_WIDTH; ++p) {
    // Collaborative loading of M and N tiles into shared memory
    ds_M[ty][tx] = M[Row*Width + p*TILE_WIDTH+tx];
    ds_N[ty][tx] = N[(p*TILE_WIDTH+ty)*Width + Col];
    __syncthreads();

    for (int i = 0; i < TILE_WIDTH; ++i)
      Pvalue += ds_M[ty][i] * ds_N[i][tx];
    __synchthreads();
  }
  P[Row*Width+Col] = Pvalue;
}
```

Handling Matrix of Arbitrary Size

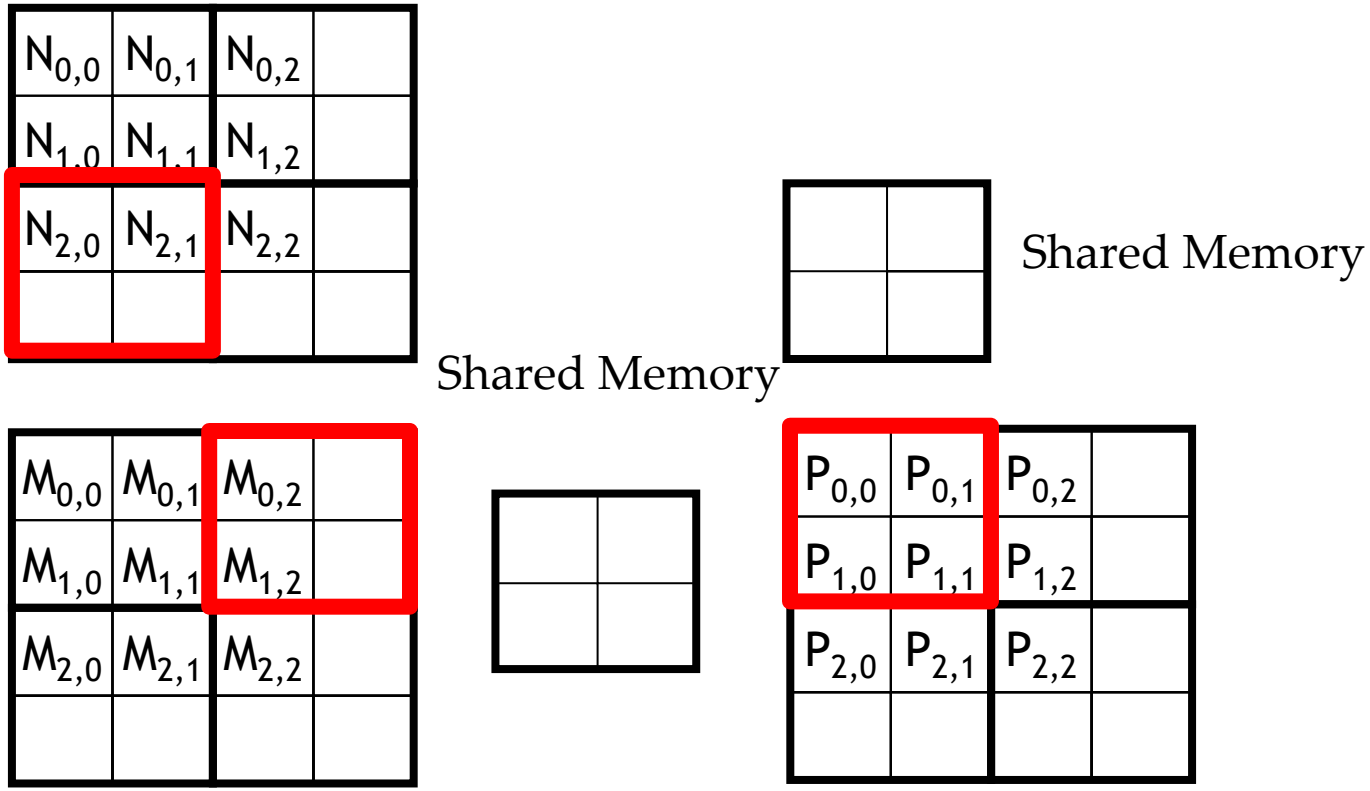
- **Real applications need to handle arbitrary sized matrices.**
  - *One could pad (add elements to) the rows and columns into multiples of the tile size, but would have significant space and data transfer time overhead.*
- **We will take a different approach.**
  - Boundary condition checking
  - Regularizing tile contents

Phase 0 Loads for Block (0,0) for a 3x3 Example

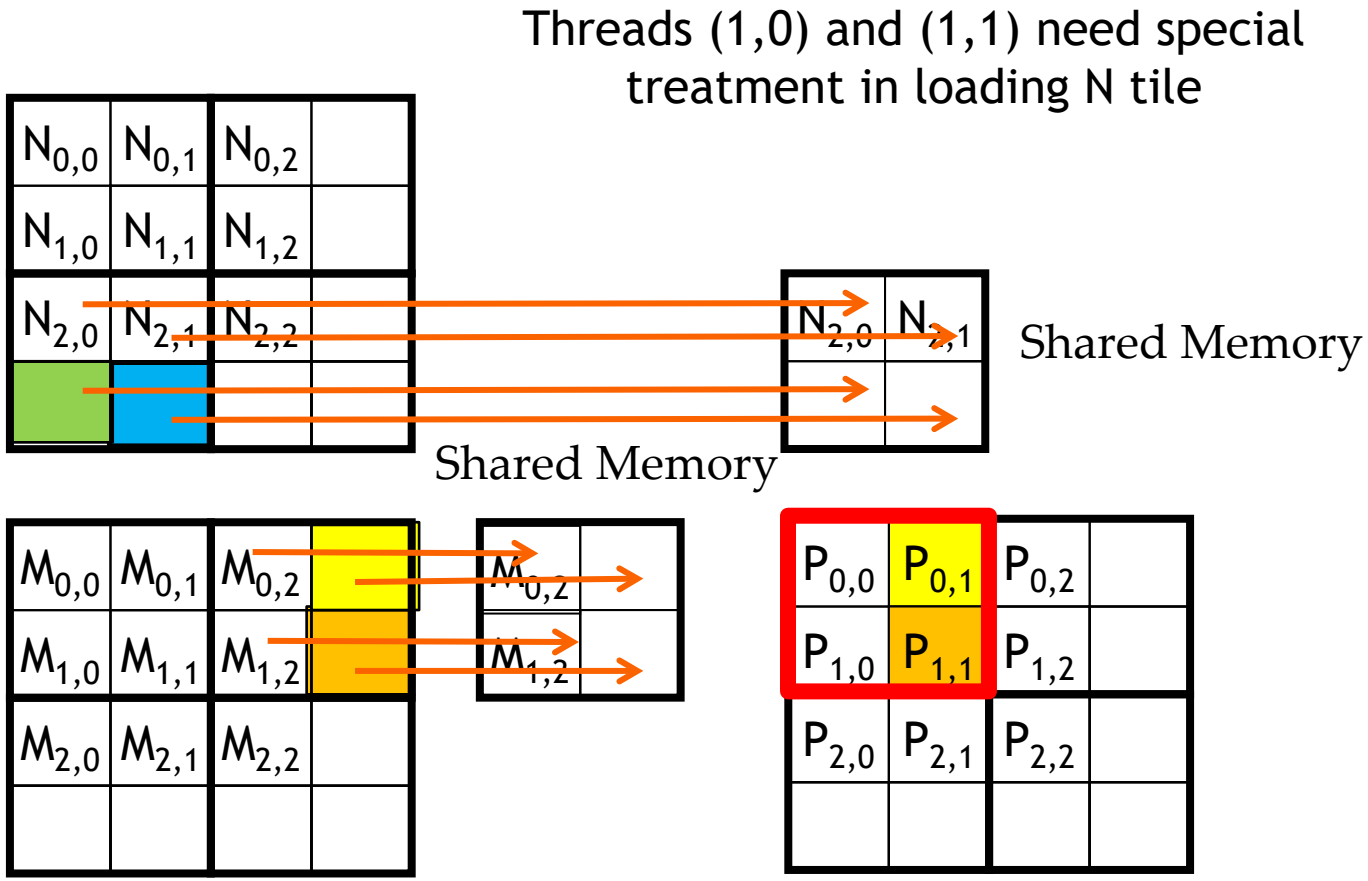


Phase 0 is ok: all threads access a legitimate address

Phase 1 Loads for Block (0,0) for a 3x3 Example



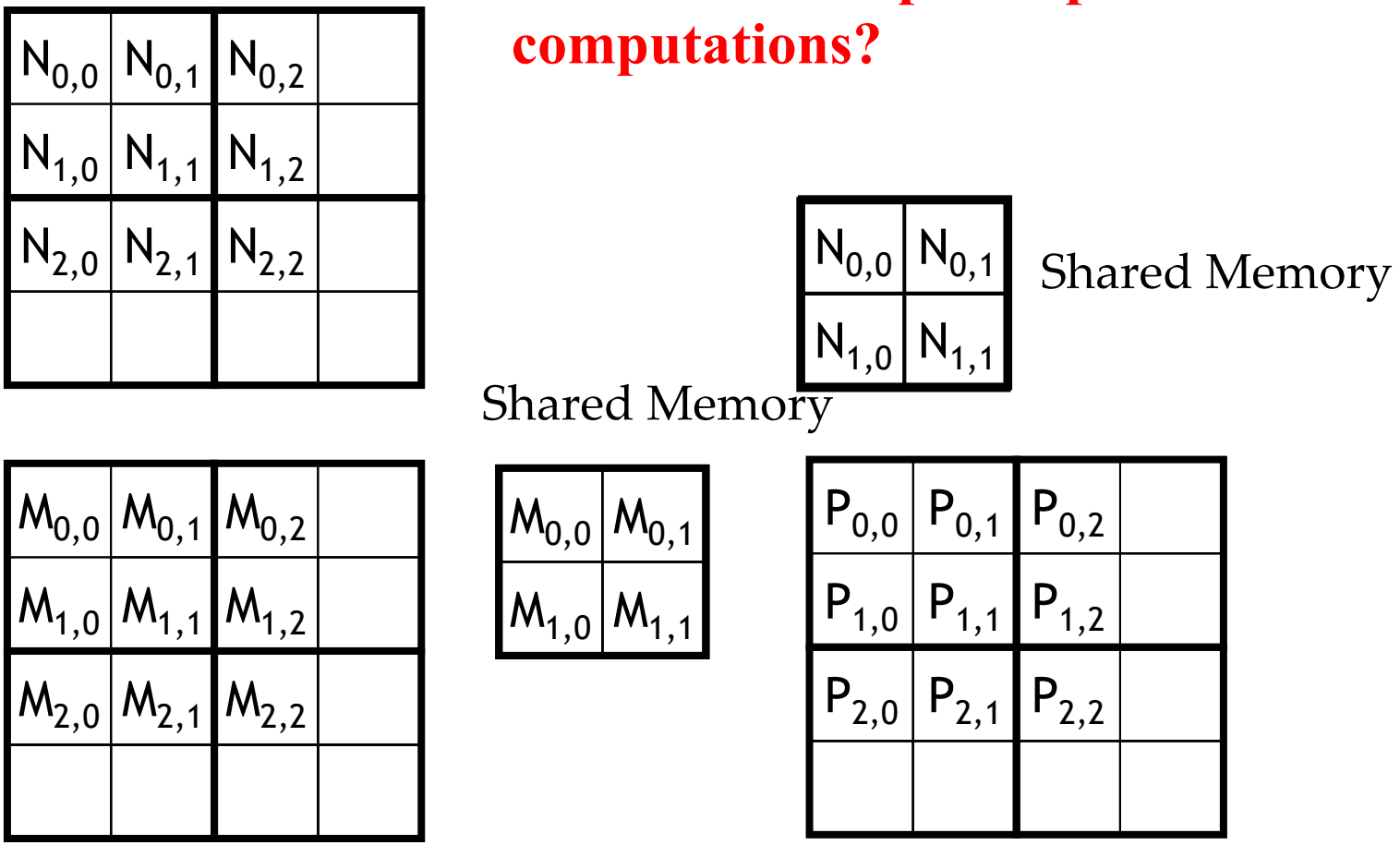
Phase 1 Loads for Block (0,0) for a 3x3 Example



If we let thread(0,1) read from M(0,3), what will actually be read into SM(0,1)?

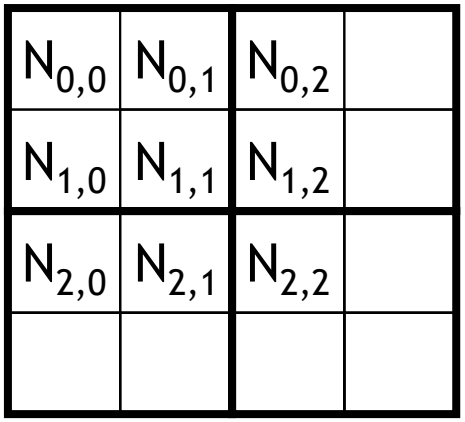
Phase 0 Use for Block (0,0) (iterations 0 and 1)

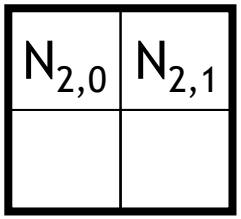
Which threads participate in computations?



Phase 1 Use for Block (0,0) (iteration 0)

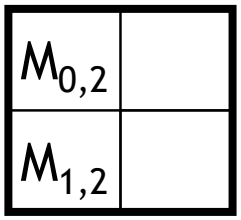
Which threads participate in computations?

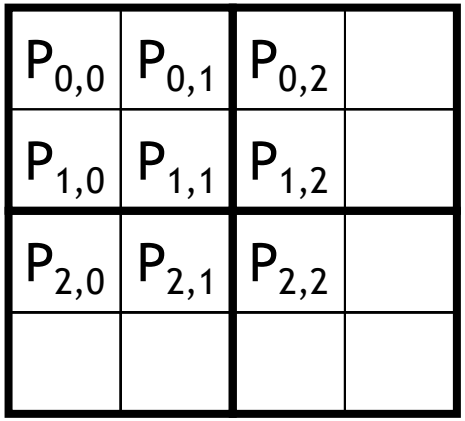




Shared Memory

Shared Memory

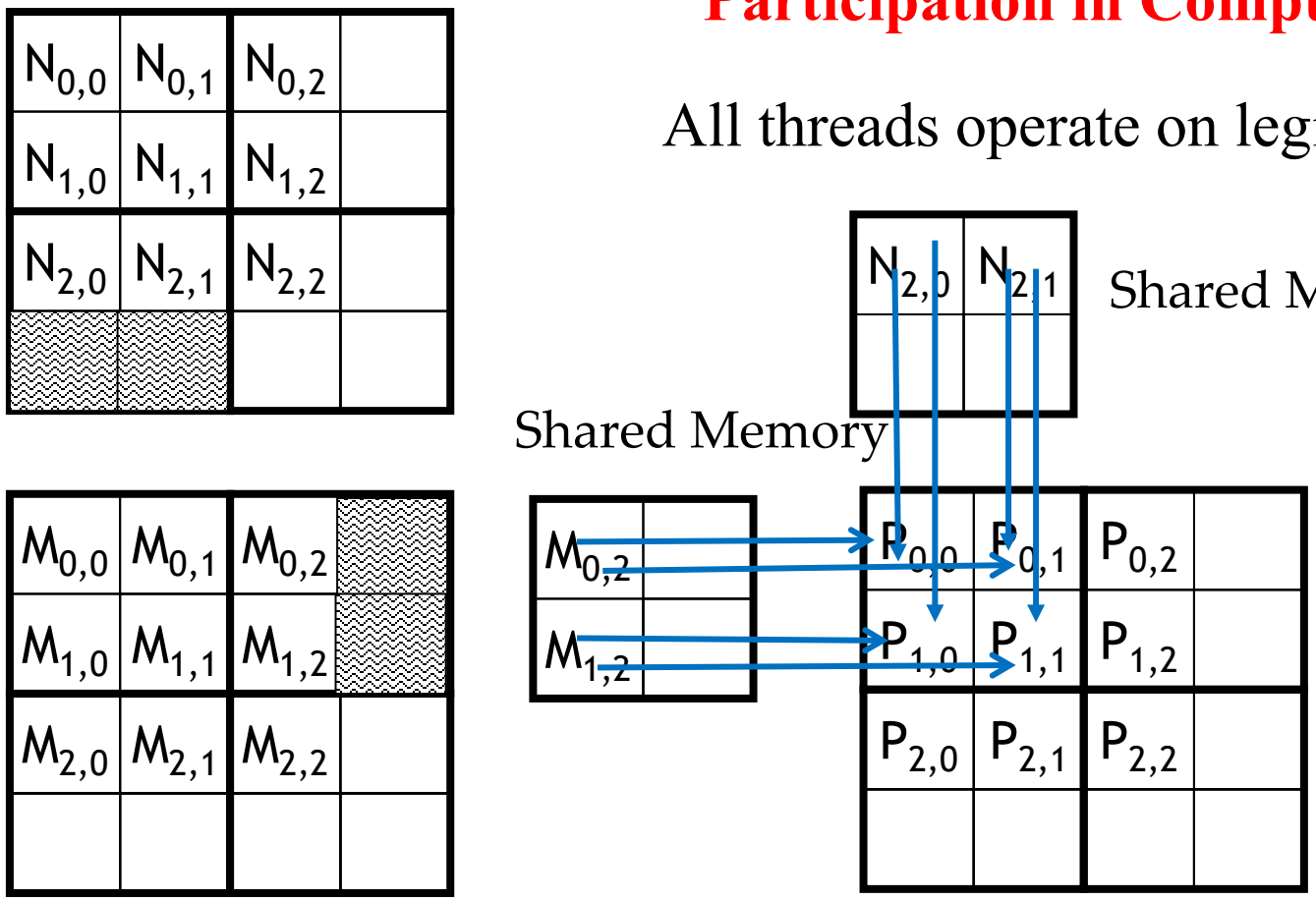




Phase 1 Use for Block (0,0) (iteration 0)

Participation in Computations!

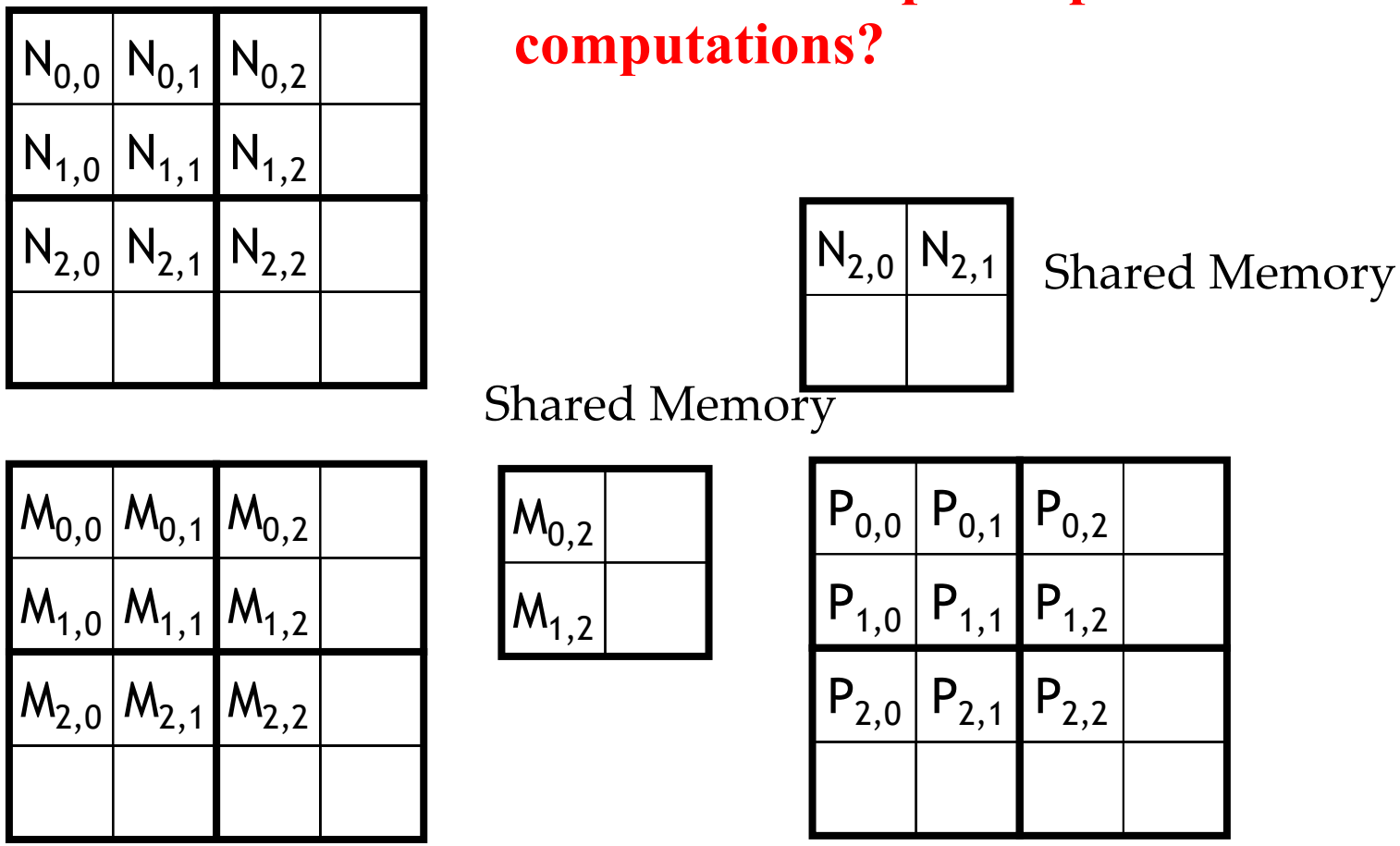
All threads operate on legitimate data



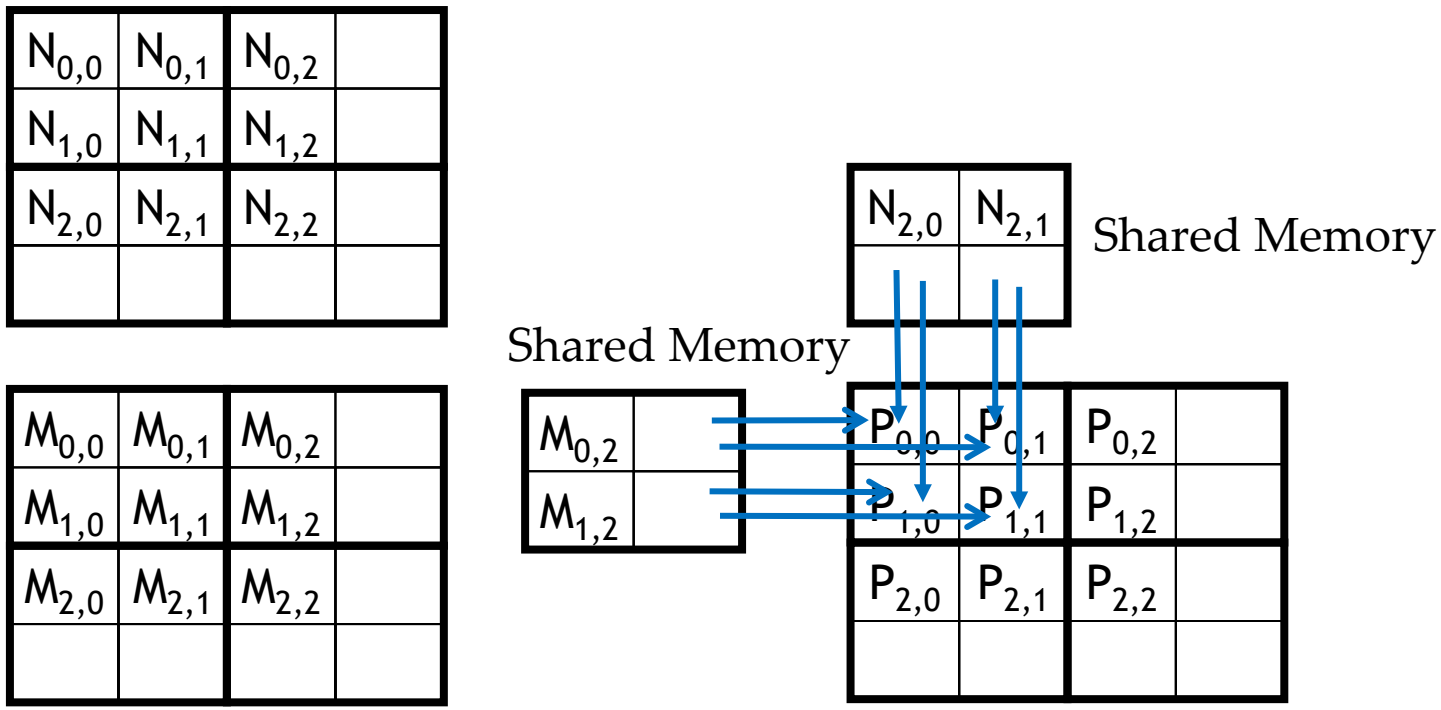
Thread(0,1) participates in [M(0,2)*N(2,1)] but should not read M(0,3)!
Thread(1,1) participates in (M(1,2)*N(2,1)] but should not read M(1,3)!
Similar for Thread(1,0) and Thread(1,1) on N(3,0) ad N(3,1)

Phase 1 Use for Block (0,0) (iteration 1)

Which threads participate in computations?

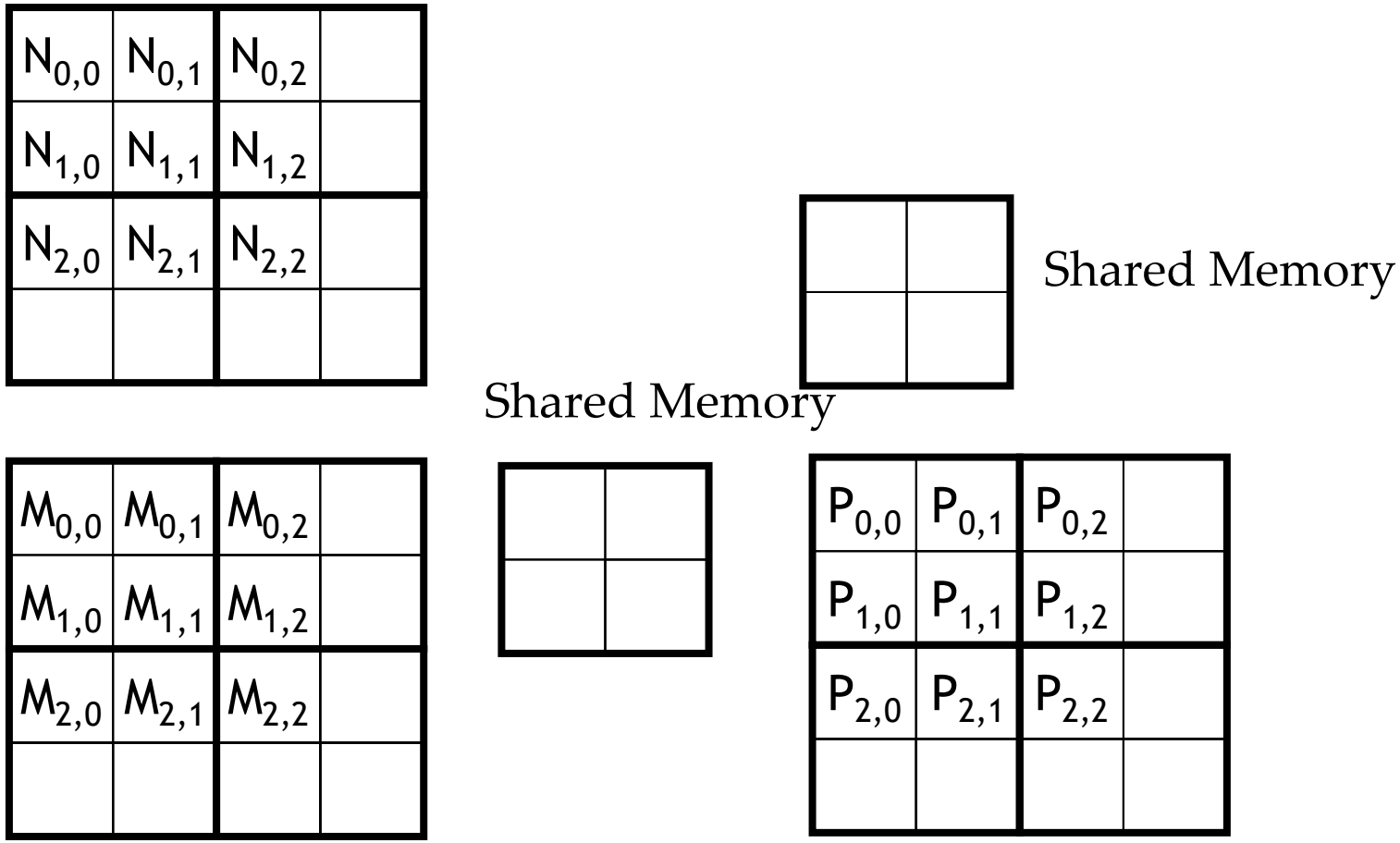


Phase 1 Use for Block (0,0) (iteration 1)



All Threads need special treatment. None of them should introduce invalidate contributions to their P elements.

Load problems can occur even during phase 0!

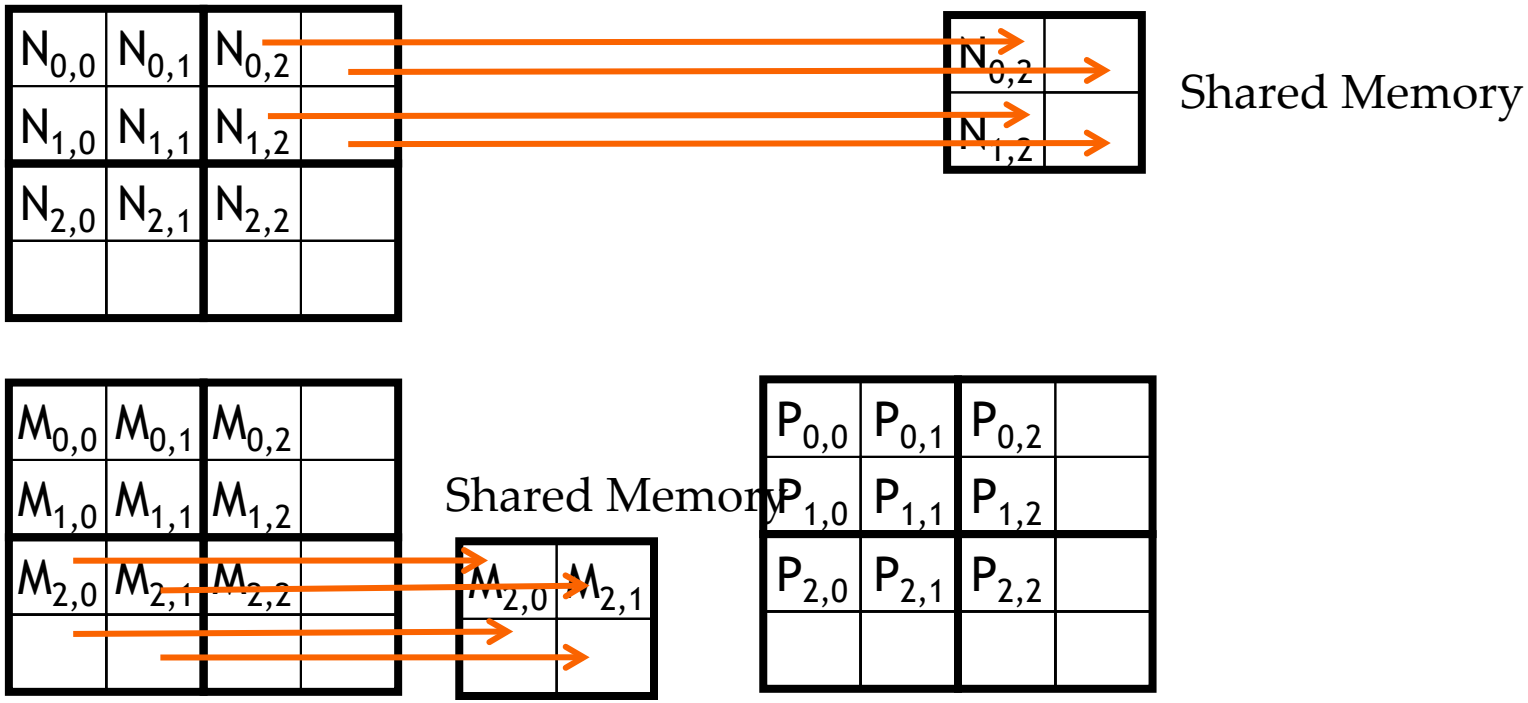


Which thread block will have threads not participating in loads during Phase 0?

Phase 0 Loads for Block (1,1) for a 3x3 Example

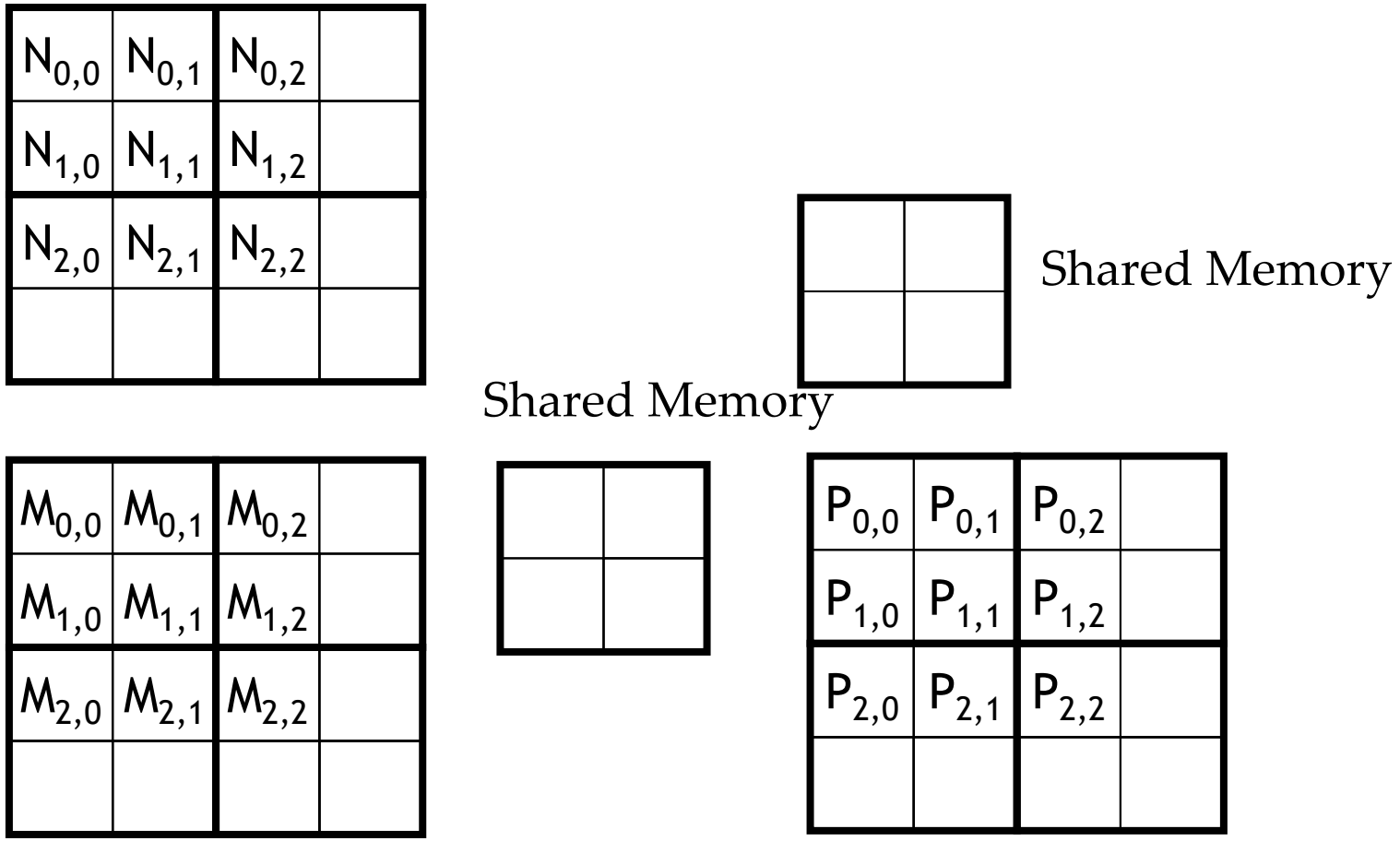
Problems can occur even during phase 0!

Threads (0,1) and (1,1) need special treatment in loading N tile



Threads (1,0) and (1,1) need special treatment in loading M tile

Threads not participating in computations in phase 0 may participate in loads. Really???

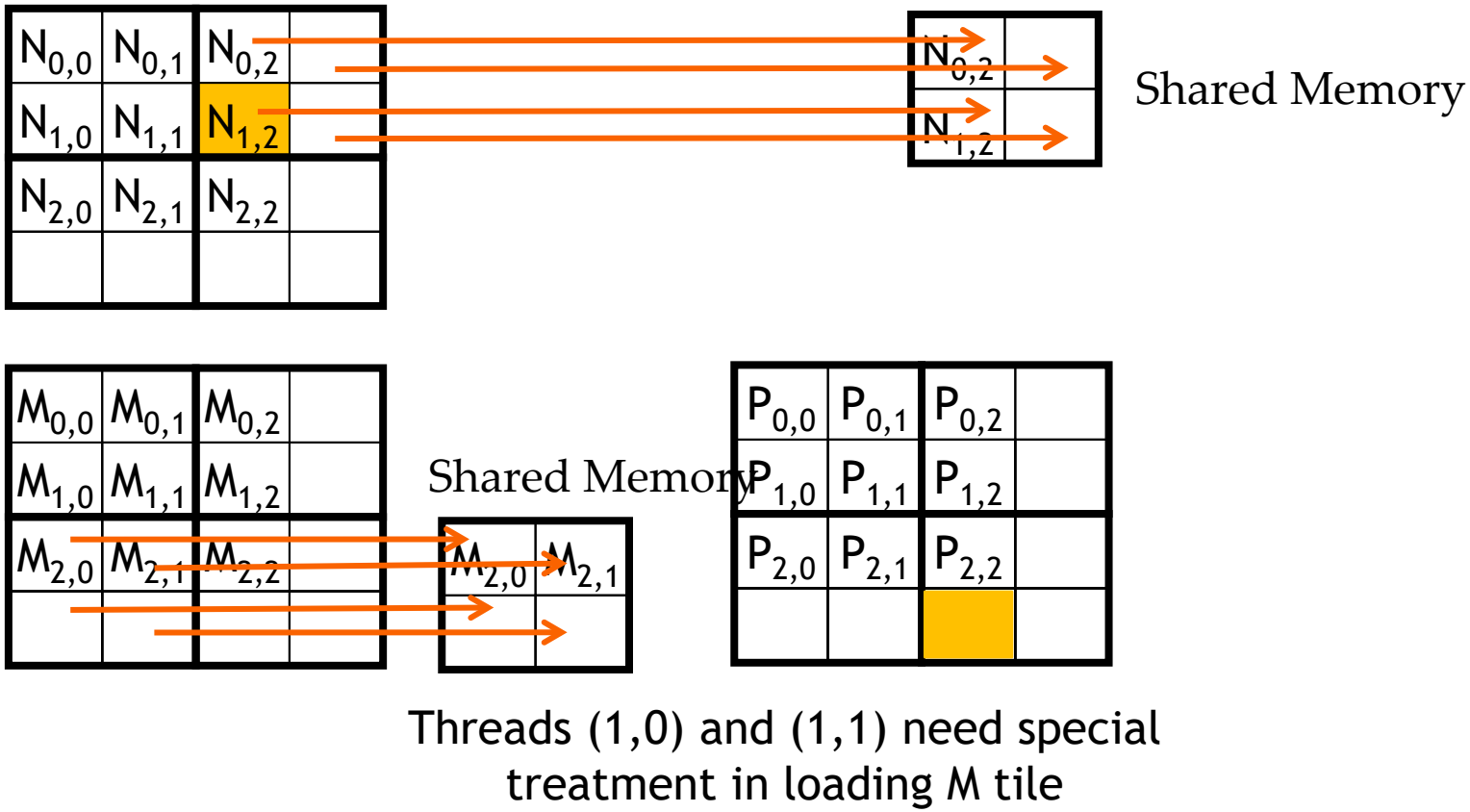


Which thread block?

Phase 0 Loads for Block (1,1) for a 3x3 Example

Problems can occur even during phase 0!

Threads (0,1) and (1,1) need special treatment in loading N tile



Threads (1,0) and (1,1) need special treatment in loading M tile

Phase 0 of Block(1,1), Thread(1,0), assigned to calculate non-existent P[3,2] but need to participate in loading tile element N[1,2]

Major Cases in Toy Example

- **We can not just simply exclude the threads that do not calculate valid P elements**
  - they still need to participate in loading the input tiles
  - Phase 0 of Block(1,1), Thread(1,0), assigned to calculate non-existent P[3,2] but need to participate in loading tile element N[1,2]
- **Threads that calculate valid P elements may attempt to load non-existing input elements when loading input tiles**
  - Phase 0 of Block(0,0), Thread(1,0), assigned to calculate valid P[1,0] but attempts to load non-existing N[3,0]

Next

- **Writing boundary conditions for matrix multiplication**