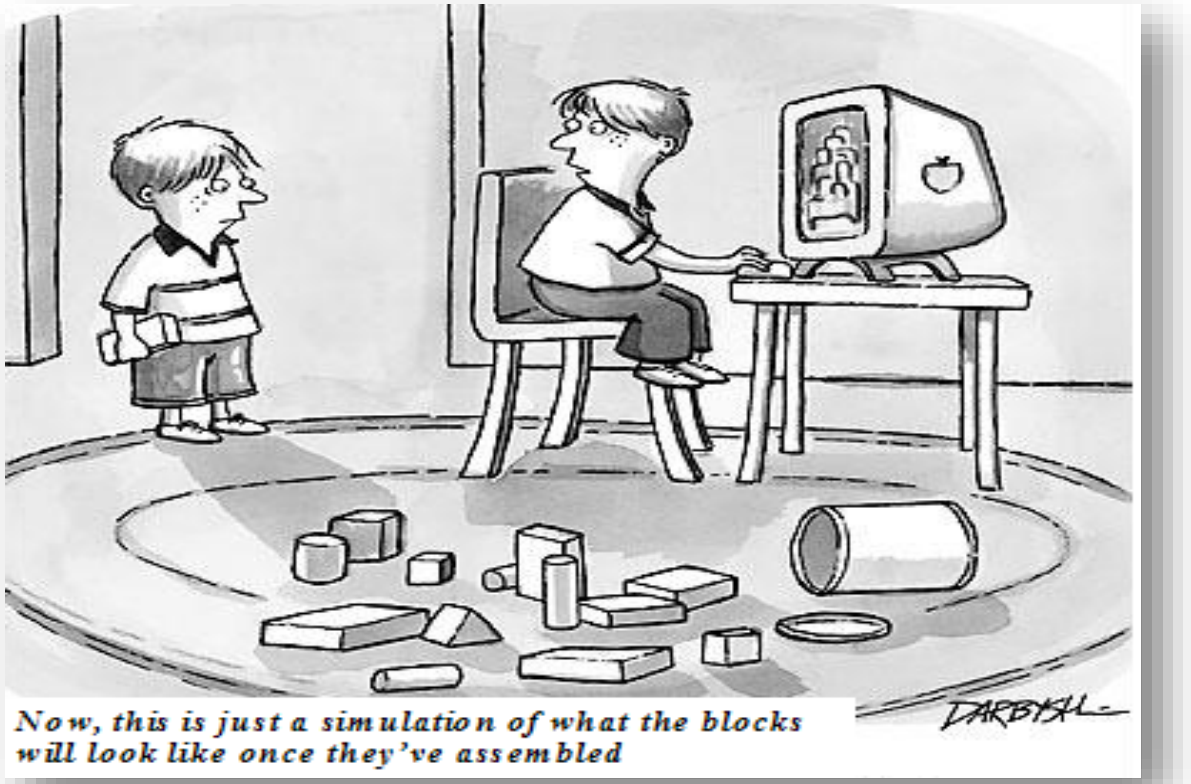
ECE569
Module 45



- Convolution – 1D

Convolution

- Widely used in audio, image and video processing
- In high-performance computing, convolution is often referred to as stencil computation
  - stencil computation used in many science and engineering applications
  - appears widely in numerical methods for solving differential equations.
  - basis of many force calculation algorithms

Convolution

- Involves significant number of arithmetic operations on each data element.
  - Each output data element calculated independently

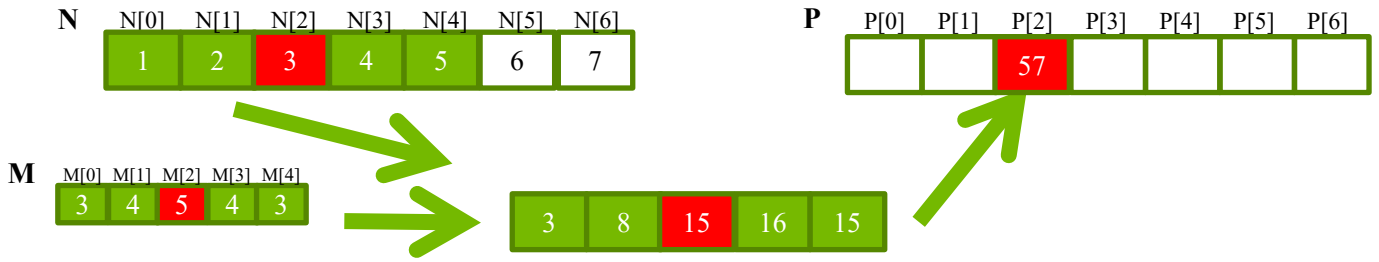
- Challenges
  - input data sharing
  - boundary conditions.
    - use case of sophisticated tiling methods and input data staging

Convolution as a filter

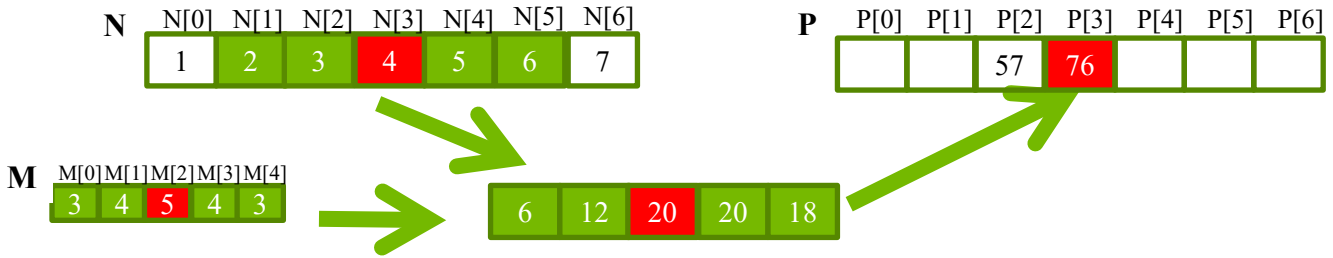
- **Each output data element is a weighted sum of a collection of neighboring input elements**
  - The weights defined by an input mask array
    - *Image blur is a special case with all weights being the same*
- **Transforms inputs**
  - smooth signal values so that one can see the big-picture trend
  - sharpen boundaries and edges of objects in images

1D Convolution

Mask size of 5:

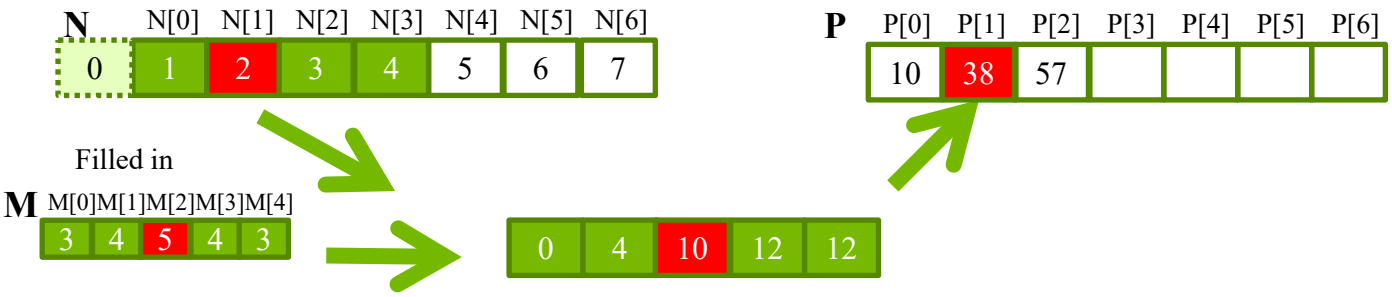


$$P[2] = N[0]*M[0] + N[1]*M[1] + N[2]*M[2] + N[3]*M[3] + N[4]*M[4]$$



1D Convolution – Boundary Condition

- **Calculation of output elements near the boundaries (beginning and end) of the array need to deal with "ghost" elements**
  - Different policies (0, replicates of boundary values, etc.)



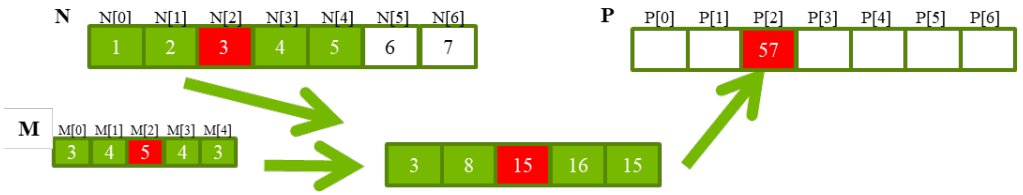
P[1] = 0 * M[0] + N[0]*M[1] + N[1]*M[2] + N[2]*M[3] + N[3]*M[4]

= 0 * 3 + 1*4 + 2*5 + 3*4 + 4*3

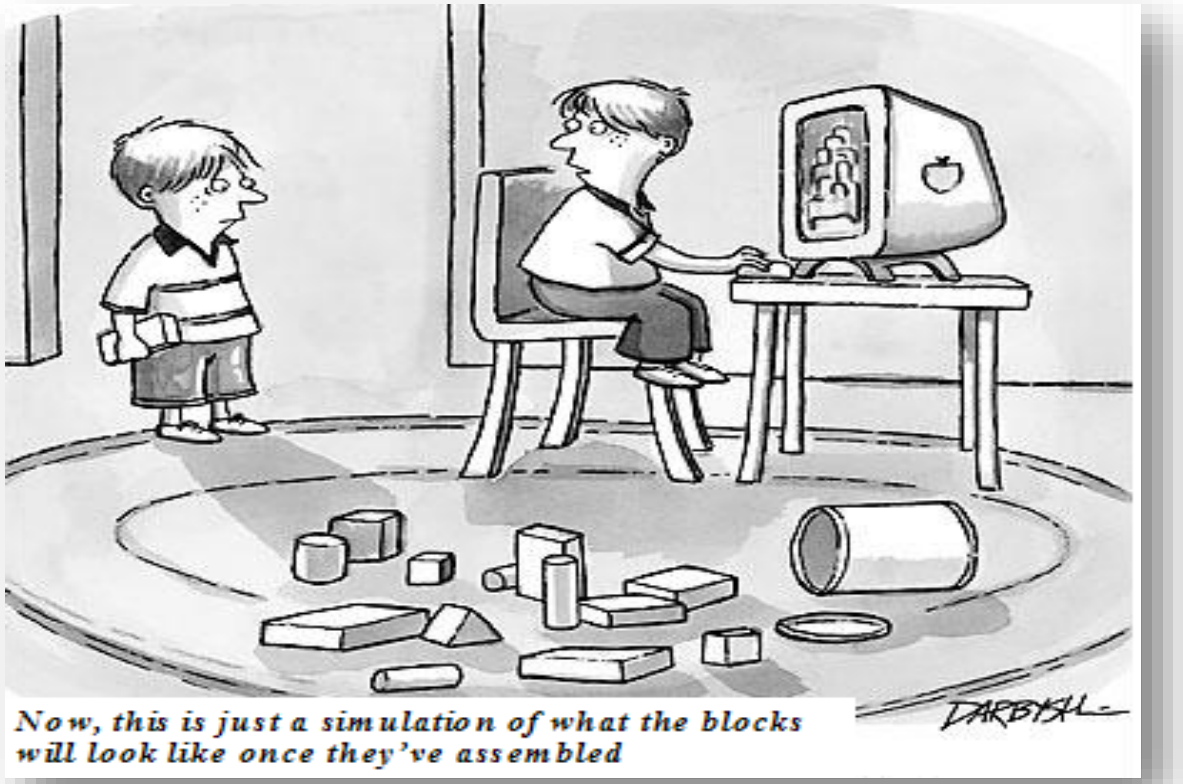
= 38

1D Convolution with Boundary Condition: Mask width is typically an odd number

```
__global__ void convolution_1D_basic_kernel(float *N,
float *M, float *P, int Mask_Width, int Width) {

  int i = ____________________________
  float Pvalue = 0;
  int start_index =_____________________________


   for (_____________________________________________) {
   //exclude zero cells to participate in multiplication

   if (_____________________________________________) {

        Pvalue += N[_____________________________]*M[_____];
   }
  }
  P[__________] = Pvalue;

}
```



ECE569
Module 46



- Convolution – 1D, Constant Memory, 2D

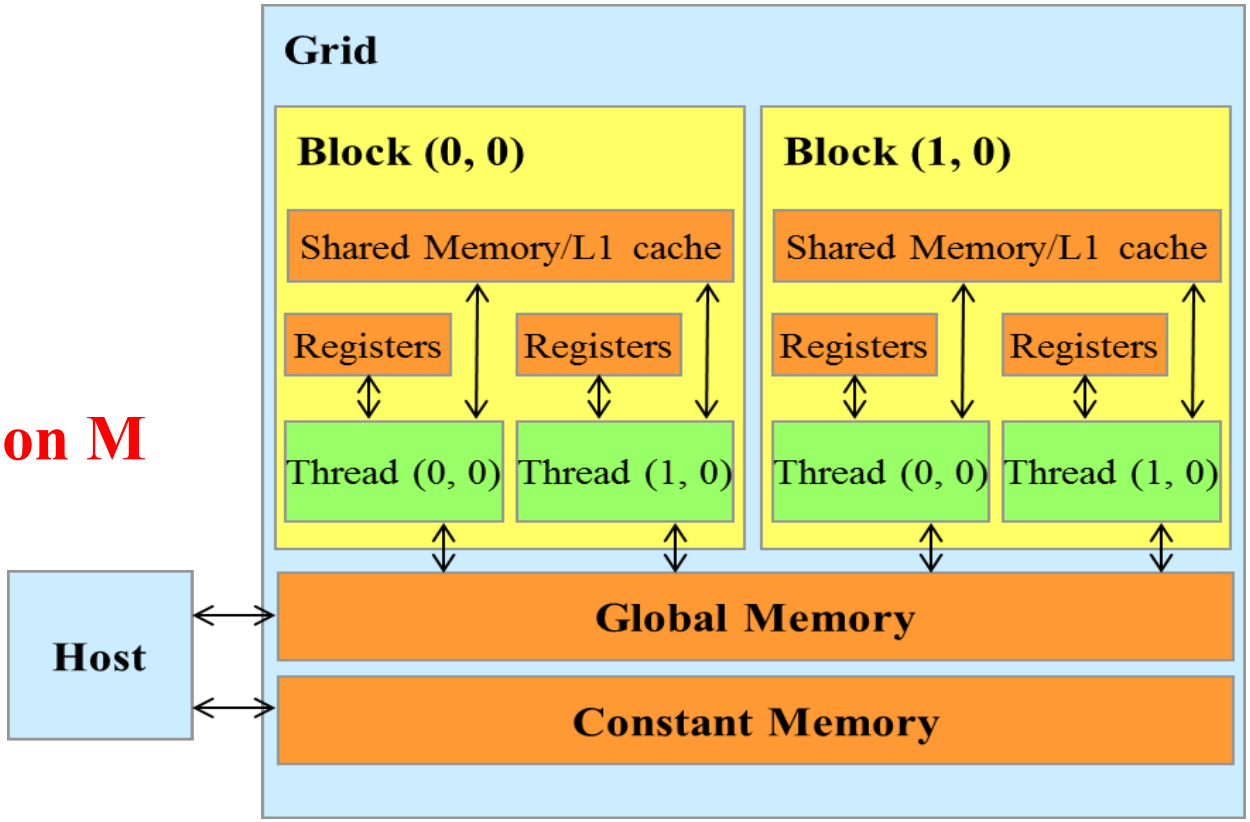
1D Convolution with Boundary Condition: Mask width is typically an odd number

```
__global__ void convolution_1D_basic_kernel(float *N,
float *M, float *P, int Mask_Width, int Width) {

  int i = ____________________________
  float Pvalue = 0;
  int start_index =_____________________________


   for (_____________________________________________) {
   //exclude zero cells to participate in multiplication

   if (_____________________________________________) {

        Pvalue += N[_____________________________]*M[_____];
   }
  }
  P[__________] = Pvalue;

}
```

Mask Array Accesses and Access Patterns

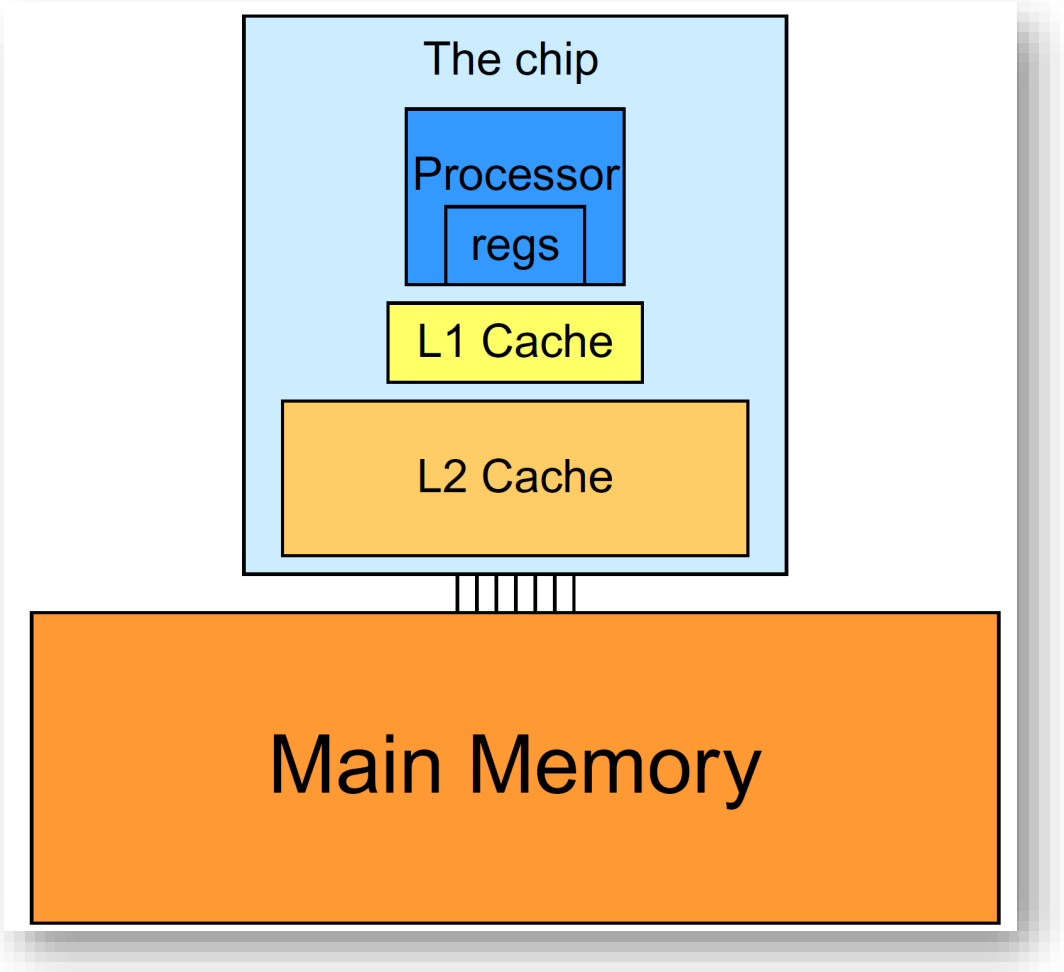
- Three critical properties in convolution
  - Small size M array
  - Contents of M don't change
  - All threads access the mask elements.

Temporal Locality on M



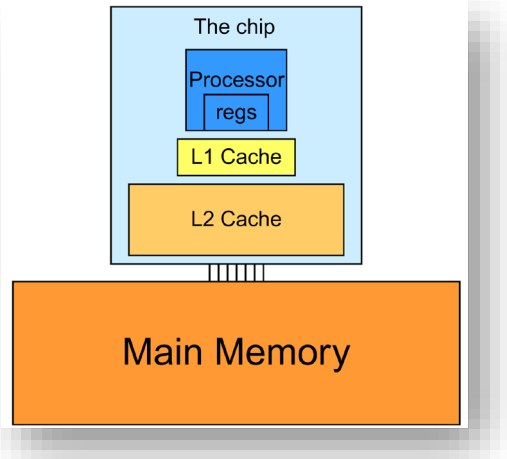
Caching in Modern Processors

- Multi level caches
- Cache coherence



Caching

- **Mask values fixed**
  - no cache coherence protocol needed I
    - hardware caches constants in L1.
- **Caches in GPUs optimized to broadcast**
  - all threads in a warp access the same constant memory variable,
  - all M elements are effectively always in the cache.
    - no DRAM bandwidth spent on M accesses.



Constant Memory

- constant memory variables are visible to all thread blocks.
  - The main difference is that a constant memory variable cannot be changed by threads during kernel execution.
- the host code needs to allocate and copy constant memory variables in a different way than global memory variables.

Constant Memory – Option 1

  - a constant memory variable cannot be changed by threads during kernel execution.
- Use of **const __restrict__** qualifiers for the mask parameter informs the compiler that it is eligible for constant caching

```
__global__ void convolution_1D_basic_kernel(float *N,
const float __restrict__ *M, float *P, int Mask_Width, int
Width) {
```

Declare host and device pointers and transfer as usual

Constant Memory – Declaration and Data Transfer Option 2:

- To declare an M array in constant memory, the host code declares it as a global variable as follows:

```
#define MAX_MASK_WIDTH 10
__constant__ float M[MAX_MASK_WIDTH];
```

- declaration should be outside any function

Constant Memory – Option 2

- Assume host code allocated and initialized mask "M_h" array in the host memory with Mask_Width elements.
- Contents of M_h can be transferred to M in the device constant memory using:
  - **cudaMemcpyToSymbol(dest, src, size)**
    - dest: pointer to the destination location in the constant memory,
    - src: pointer to the source data in the host memory,
    - size: number of bytes to be copied.

```
cudaMemcpyToSymbol(M, M_h,
                   Mask_Width*sizeof(float));
```

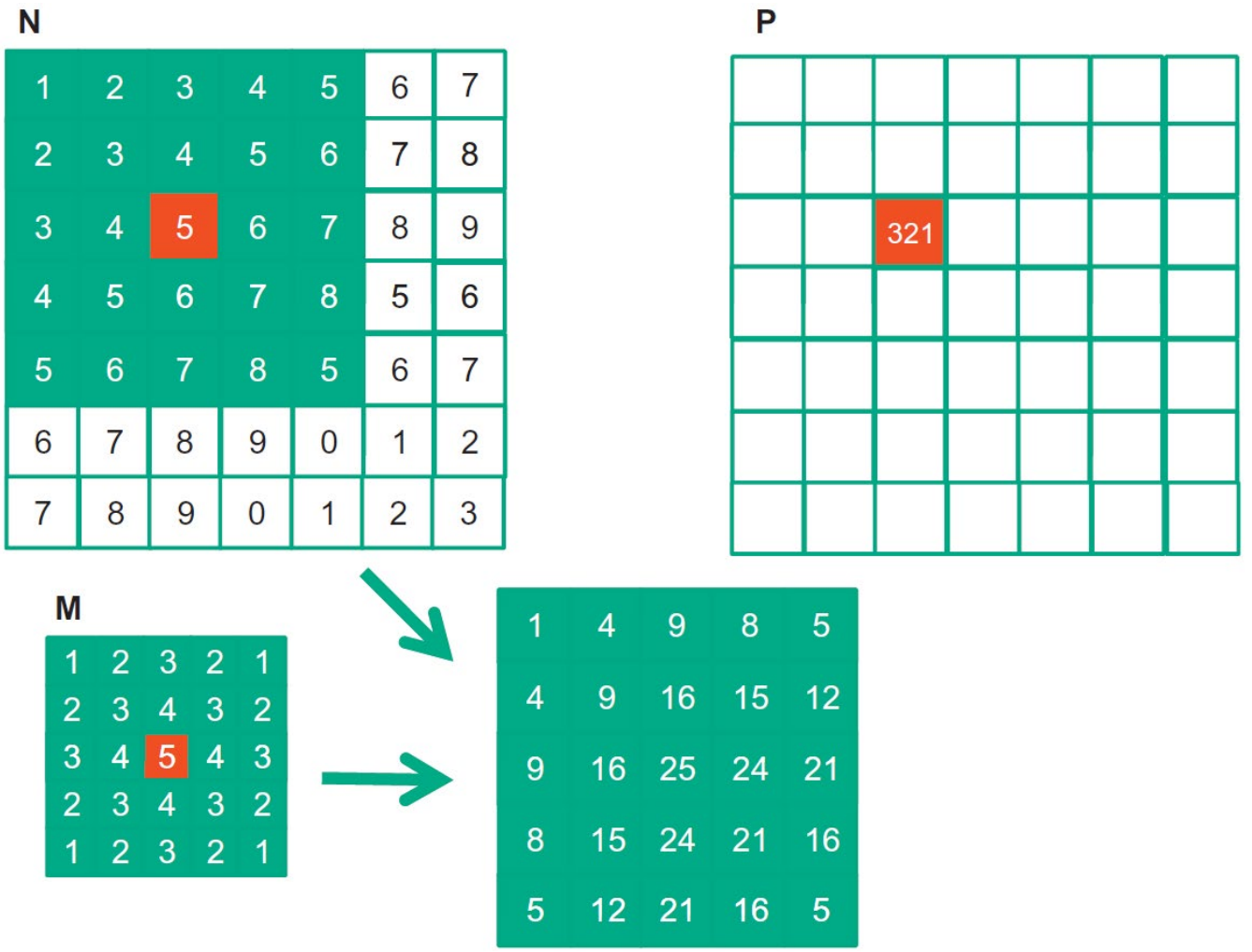
Constant Memory

- **Kernel functions access constant memory variables as global variables.**
  - their pointers do not need to be passed to the kernel as parameters.

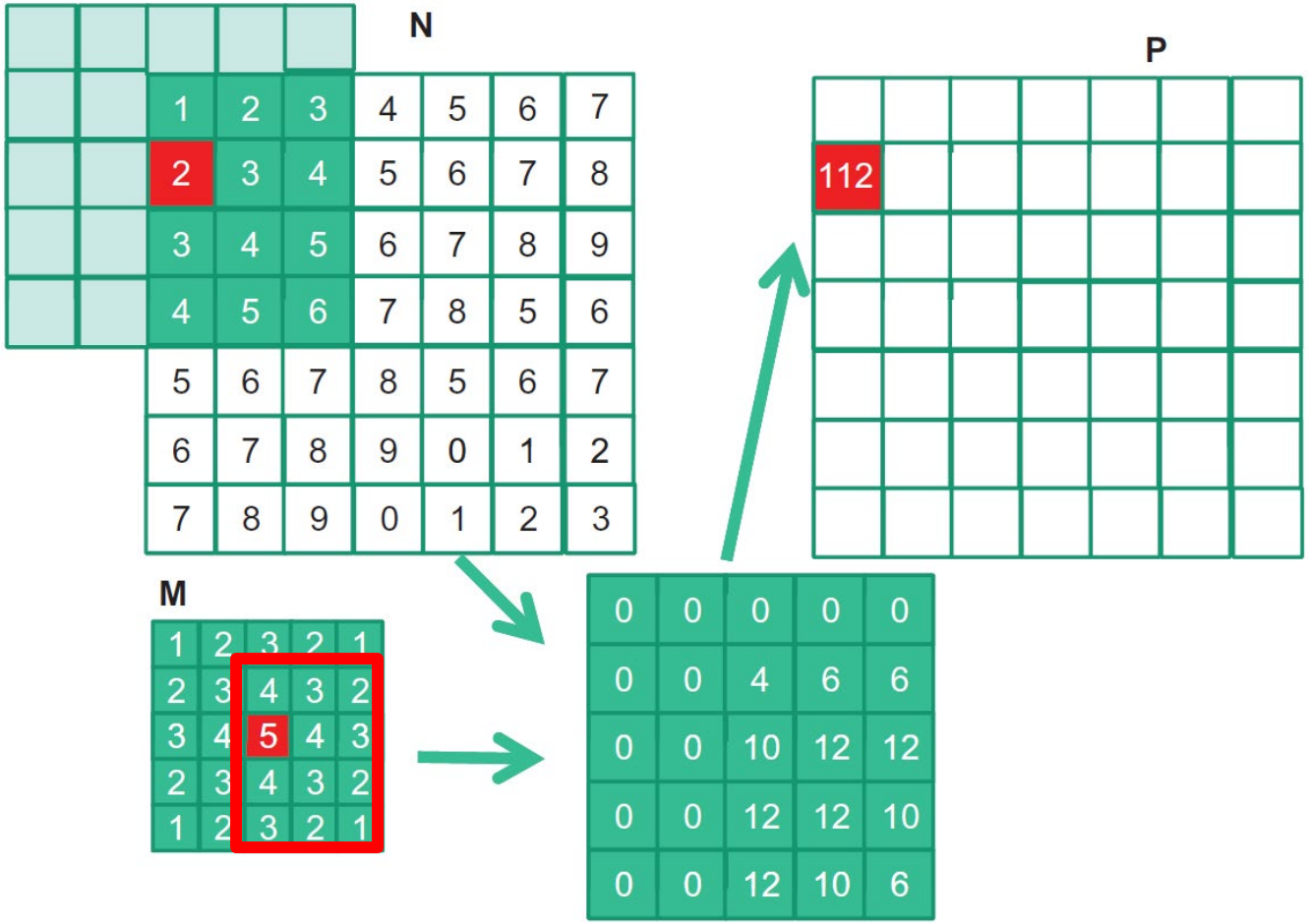
```
__global__ void
convolution_1D_basic_kernel(float *N, float
*P, int Mask_Width, int Width) {
```

- **C language scoping rules apply**
  - kernel code must include external declaration information to ensure that M is visible to the kernel.

2D Convolution



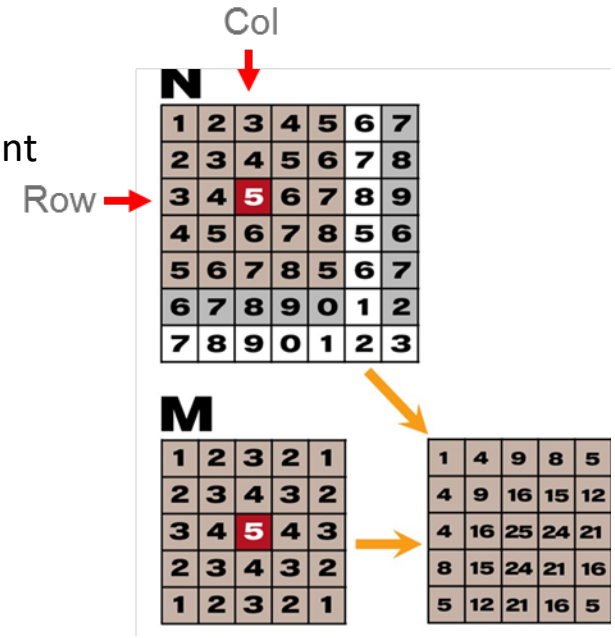
2d Convolution – Boundary Condition



```
__global__  void convolution_2D_basic(unsigned char * in, unsigned char * mask,
                                      unsigned char * out, int maskwidth, int w, int h) {
```

//Mask is square, odd sized

```
  int Col  =
  int Row  =
  if (Col < w && Row < h) {  // boundary checking for each output element
     int pixVal = 0;  // initialize local product value to 0.
     start_col  =

     start_row =

     // Get the of the surrounding box
     for(int j = 0; j < maskwidth; ++j) {  // row
        for(int k = 0; k < maskwidth; ++k) { //column

           int curRow =

           int curCol =
           // Verify we have a valid image pixel
           if(                                                                    ) {

              pixVal += in[                              ] * mask[                          ];
   } } }
    // Write our new pixel value out

    out[                              ] = (unsigned char)(pixVal);
  }
}
```



Next

- Tiling in 1D Convolution