

# Overview of GPU Architecture and CUDA Programming Models



**Georgia  
Tech**

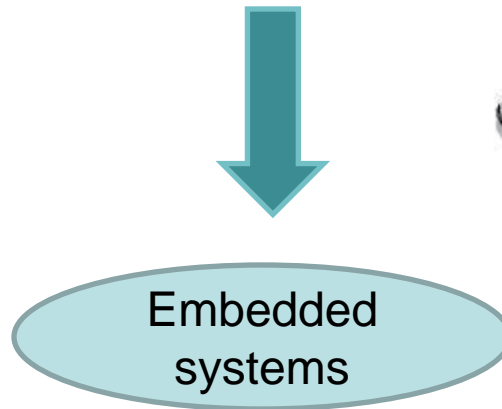
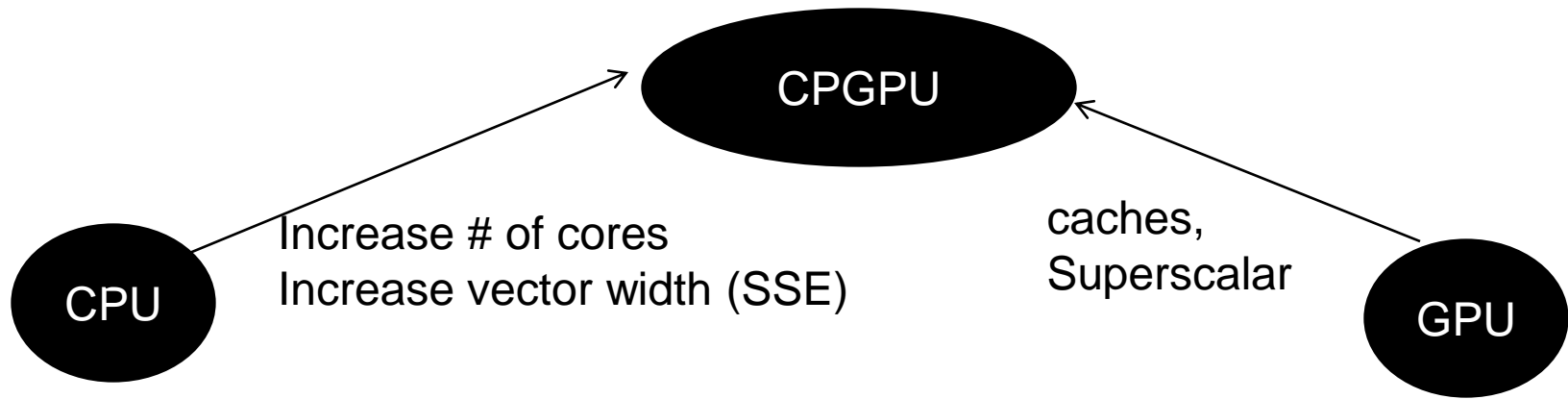


Hyesoon Kim

**comparch**



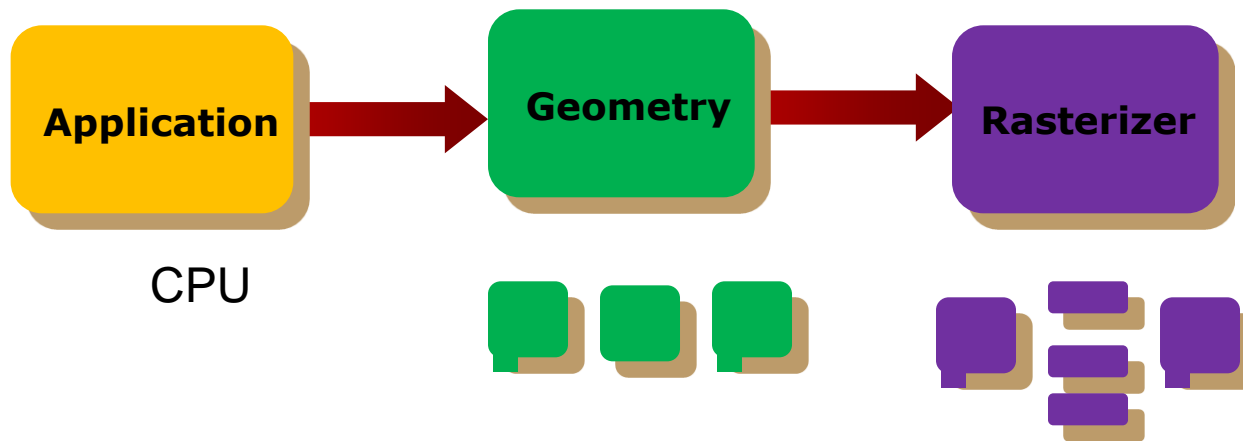
# Processor's Trend





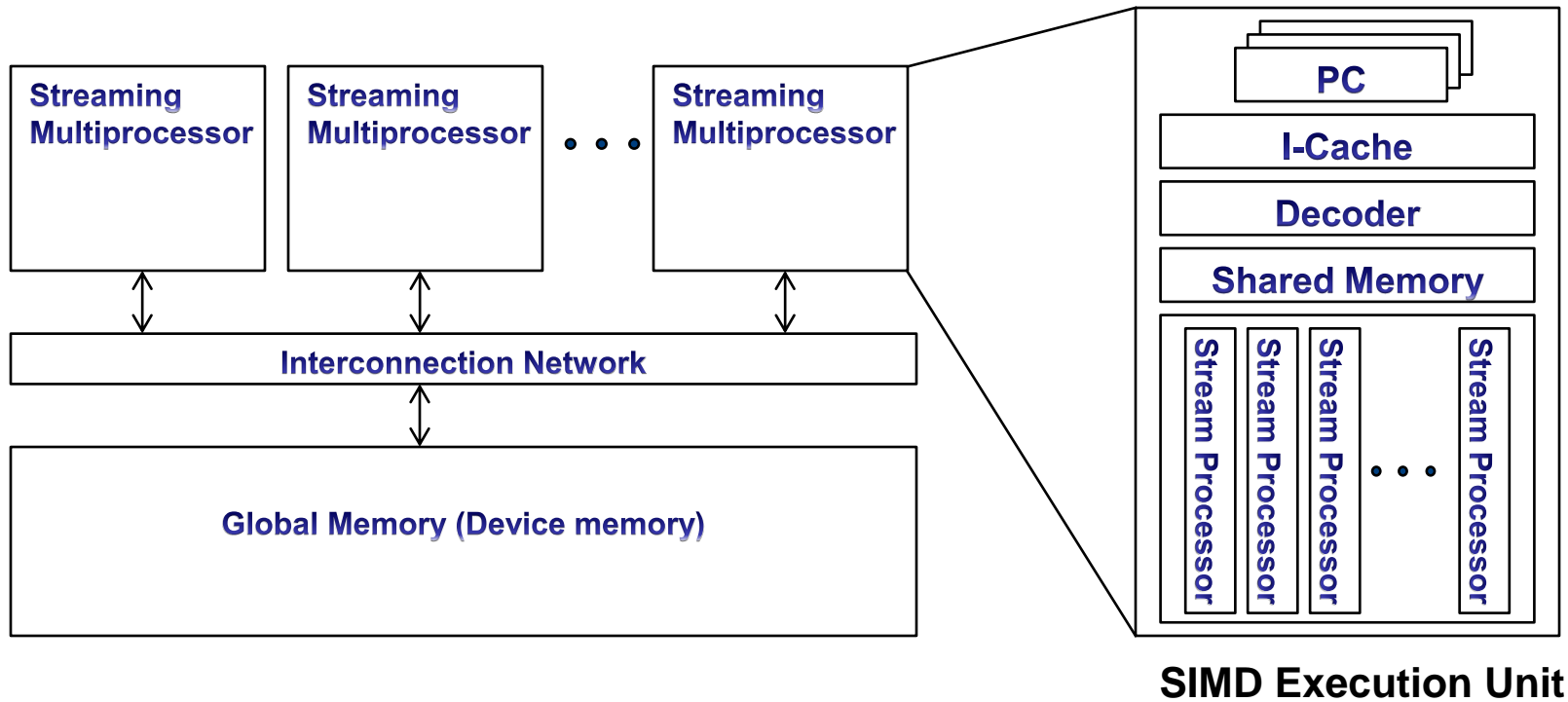
# GPU Architecture Trend

- Fixed pipelines → programmable cores → unified programmable cores → more cores → GPGPU support





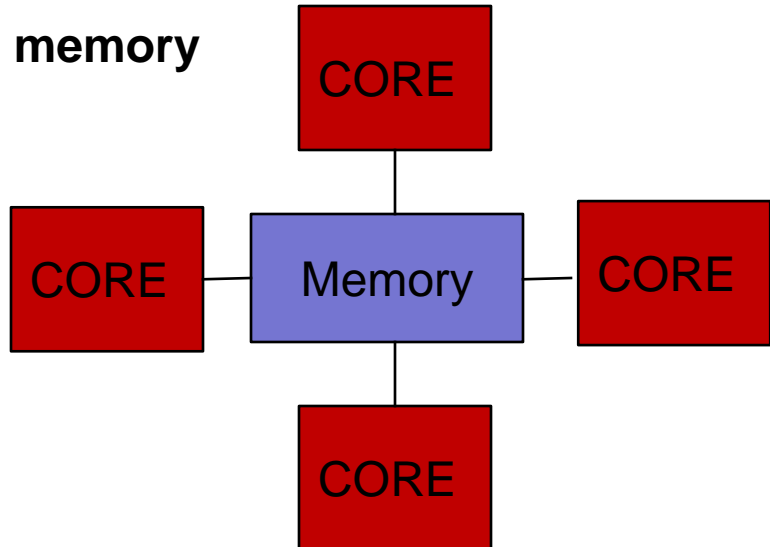
# Overview of GPU (Tesla) Architecture



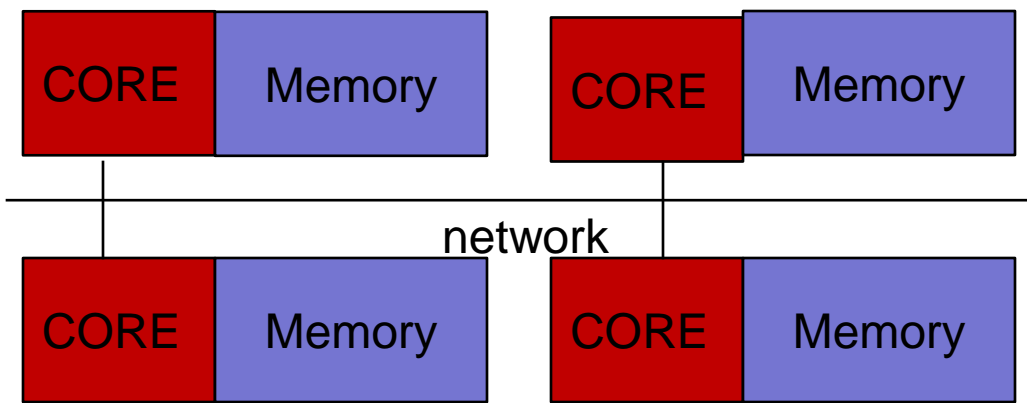
# Parallel Programming Models Based on Memory Spaces



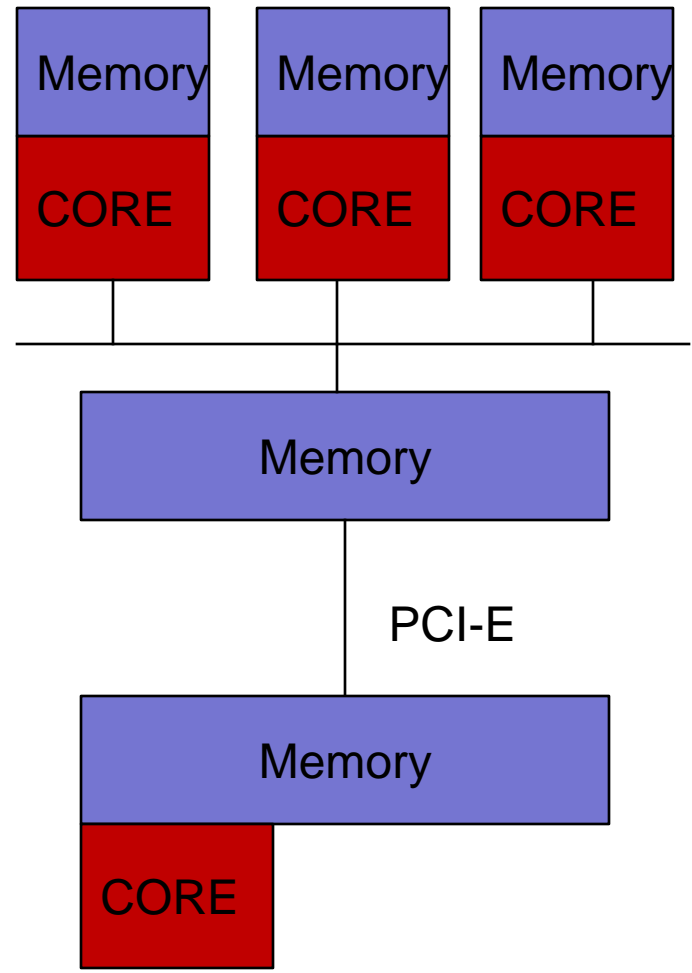
Shared memory



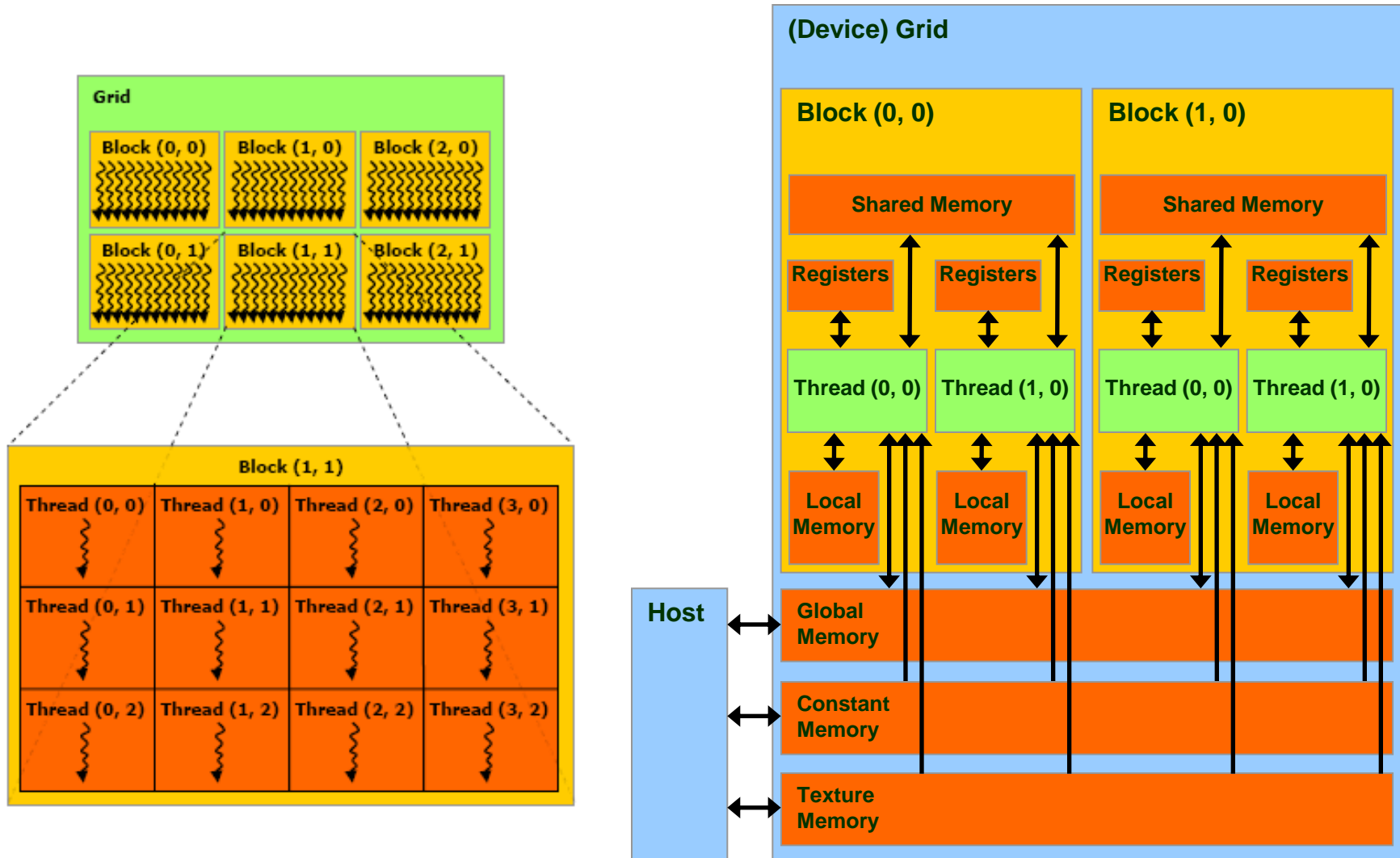
Distributed memory



CUDA



# CUDA Programming Model





# Kernel Creation

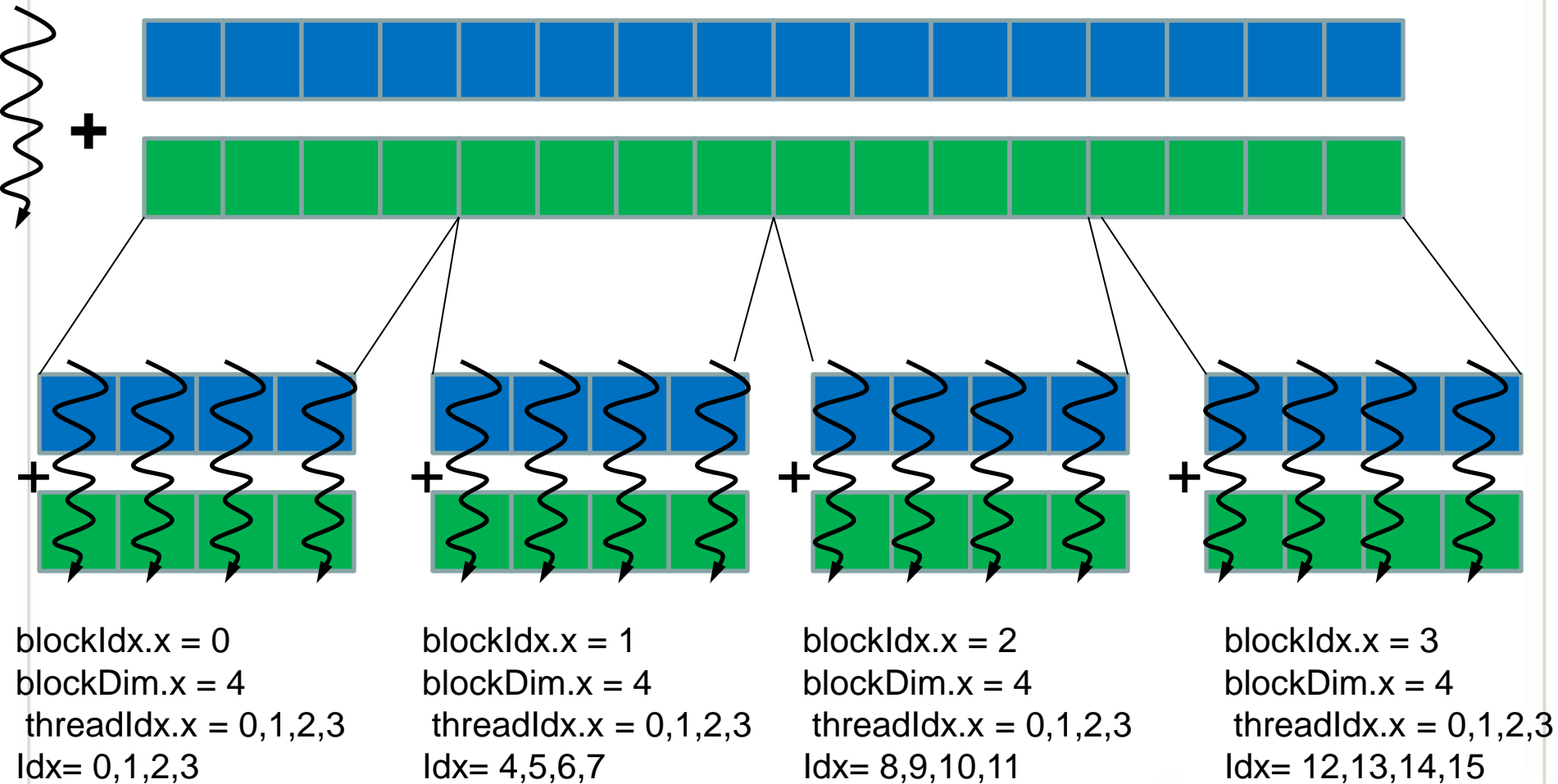
- A kernel function must be called with an **execution configuration**:

```
__global__ void KernelFunc(...);  
dim3    DimGrid(100, 50);    // 5000 thread blocks  
dim3    DimBlock(4, 8, 8);  // 256 threads per  
    block  
size_t  SharedMemBytes = 64; // 64 bytes of shared  
    memory  
KernelFunc<<< DimGrid, DimBlock, SharedMemBytes  
    >>> (...);
```



# Elementwise Matrix Addition

- Let's assume  $N=16$ ,  $\text{blockDim}=4 \rightarrow 4$  blocks



# Elementwise Matrix Addition



## CPU Program

```
void add_matrix
( float *a, float* b, float *c, int N) {
  int index;
  for (int i = 0; i < N; ++i)
    for (int j = 0; j < N; ++j) {
      index = i + j*N;
      c[index] = a[index] + b[index];
    }
}

int main () {
  add_matrix (a, b, c, N);
}
```

## GPU Program

```
__global__ add_matrix
( float *a, float *b, float *c, int N) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  int j = blockIdx.y * blockDim.y + threadIdx.y;
  int index = i + j*N;
  if (i < N && j < N)
    c[index] = a[index]+b[index];
}

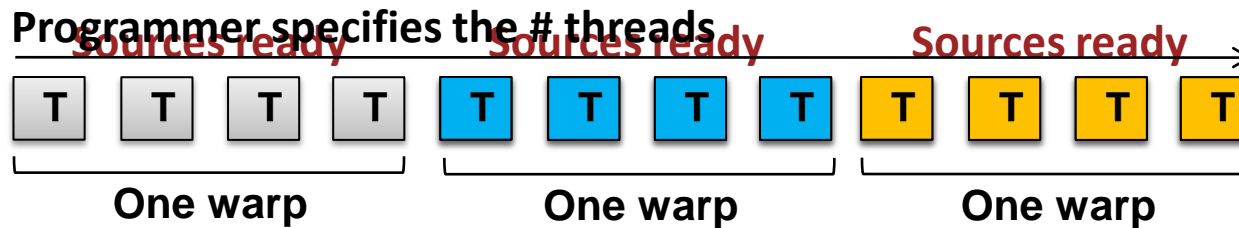
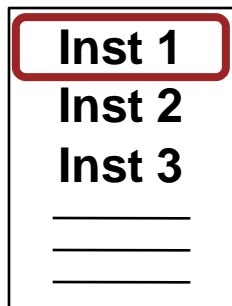
int main() {
  dim3 dimBlock( blocksize, blocksize) ;
  dim3 dimGrid (N/dimBlock.x, N/dimBlock.y);
  add_matrix<<<dimGrid, dimBlock>>>( a, b, c, N);
}
```



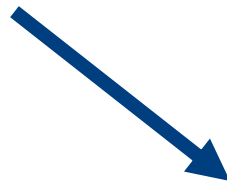
# Warp (CUDA's Term)

- **Warp is the basic unit of execution**
  - A group of threads (e.g. 32 threads for the Tesla GPU architecture)

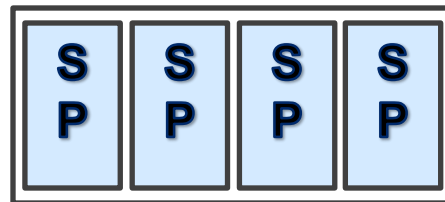
## Warp Execution



**Finite number of streaming processors**



**SIMD Execution Unit**





# OpenCL vs. CUDA

	<b>OpenCL</b>	<b>CUDA</b>
Execution Model	Work-groups/work-items	Block/Thread
Memory model	Global/constant/local/private	Global/constant/shared/local + Texture
Memory consistency	Weak consistency	Weak consistency
Synchronization	Synchronization using a work-group barrier (between work-items)	Using <code>synch_threads</code> Between threads



# GPGPU Friendly Features in Fermi

- Improve double precision
- ECC support
- Cache hierarchy
- More shared memory
- Fast atomic operations
- Dual warp scheduler (superscalar)
- Concurrent kernel execution
- Fully pipelined integer unit (fast INT operations)

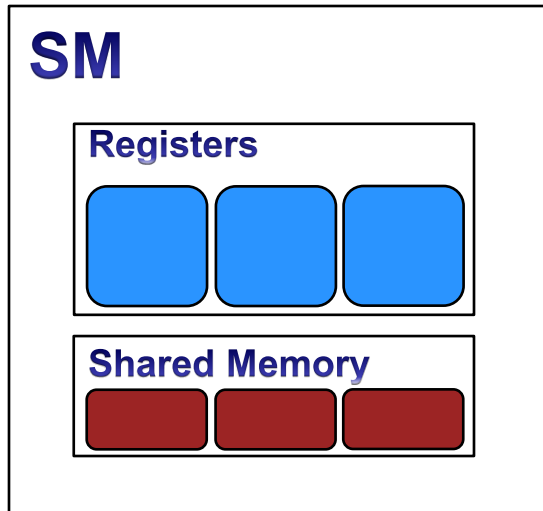


# OCCUPANCY

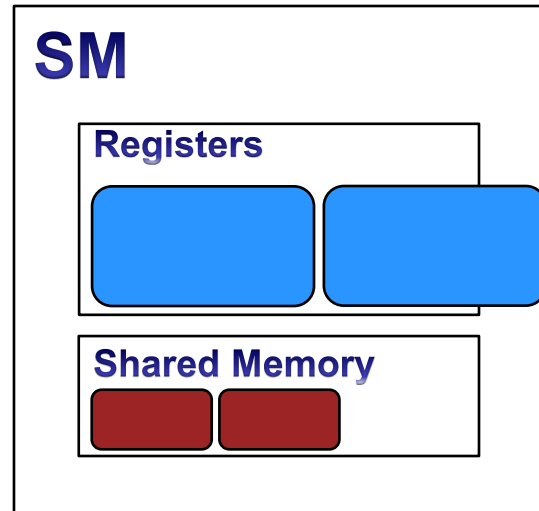


# Occupancy



- Shows how many blocks are assigned to the SM
- Programmer specifies the number of threads per block



**100% Occupancy**



**Only one block is allocated**

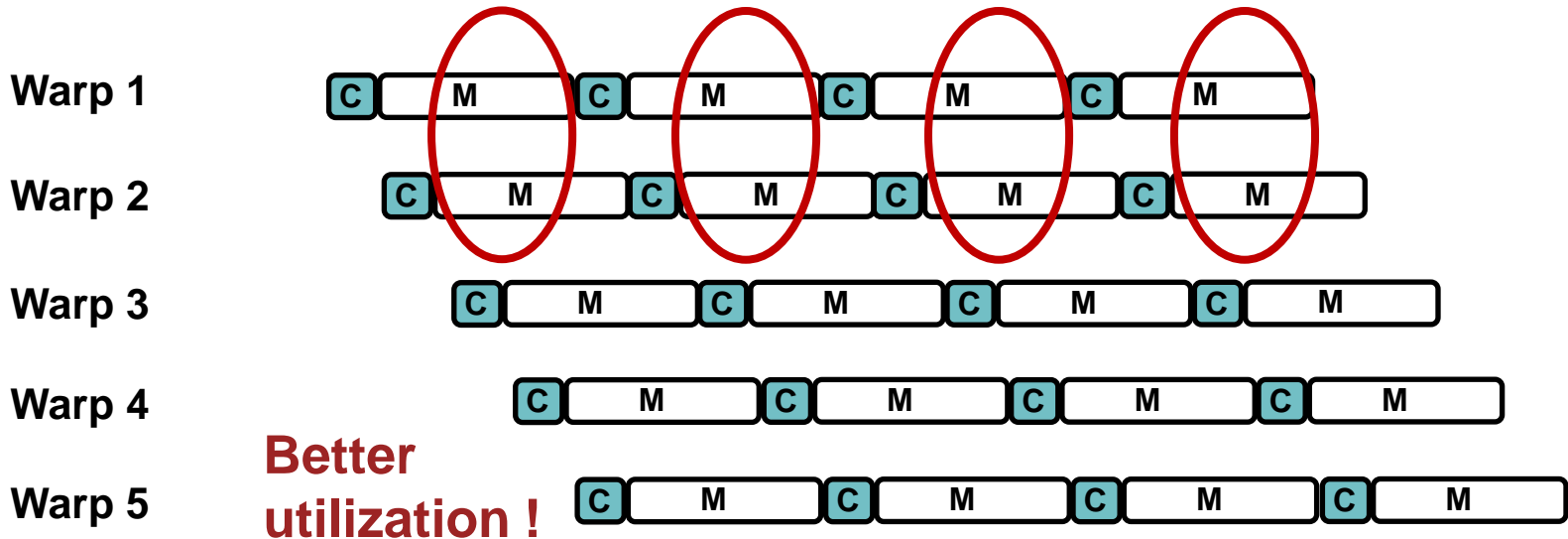
-  Register requirements per *block*
-  Shared memory requirements per *block*



# Higher Occupancy

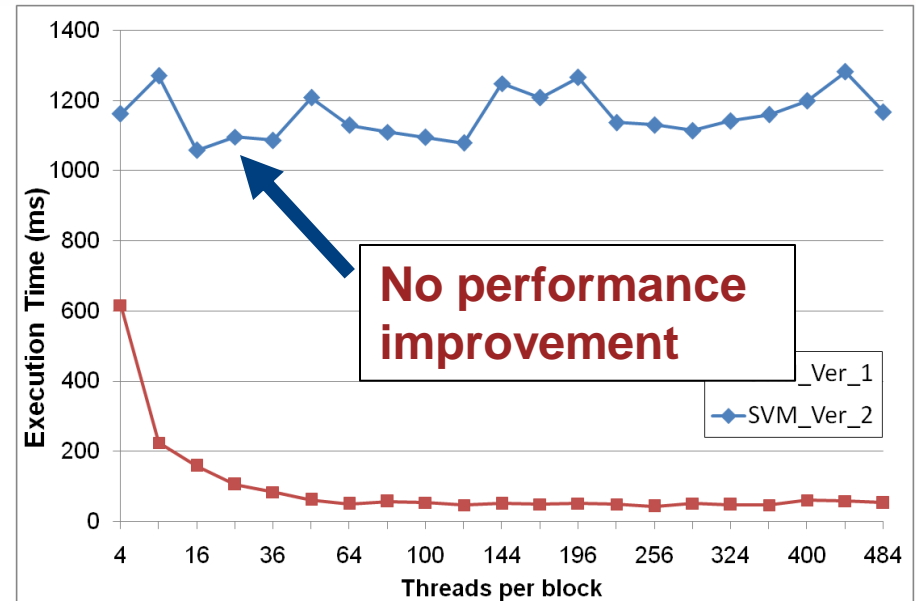
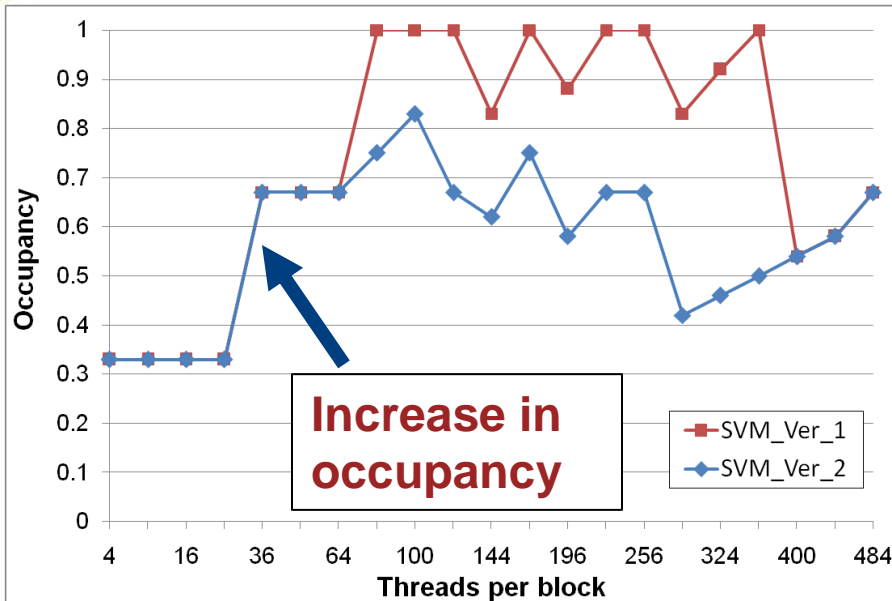
- Better processor utilization
- Hide the memory latency

Processor is not utilized





# High Occupancy $\neq$ High Performance



- Programmers try to optimize programs for occupancy
- No performance improvement from increased occupancy