CS516: Parallelization of Programs

GPU Memories

Vishwesh Jatala

Assistant Professor

Department of CSE

Indian Institute of Technology Bhilai

vishwesh@iitbhilai.ac.in

2023-24 W

References

- Miscellaneous resources from internet
- CS6023 GPU Programming
  - https://www.cse.iitm.ac.in/~rupesh/teaching/gpu/jan20/

Recap

- GPUs and CUDA Programming
  - Introduction
  - Thread Organization
  - Instruction Execution

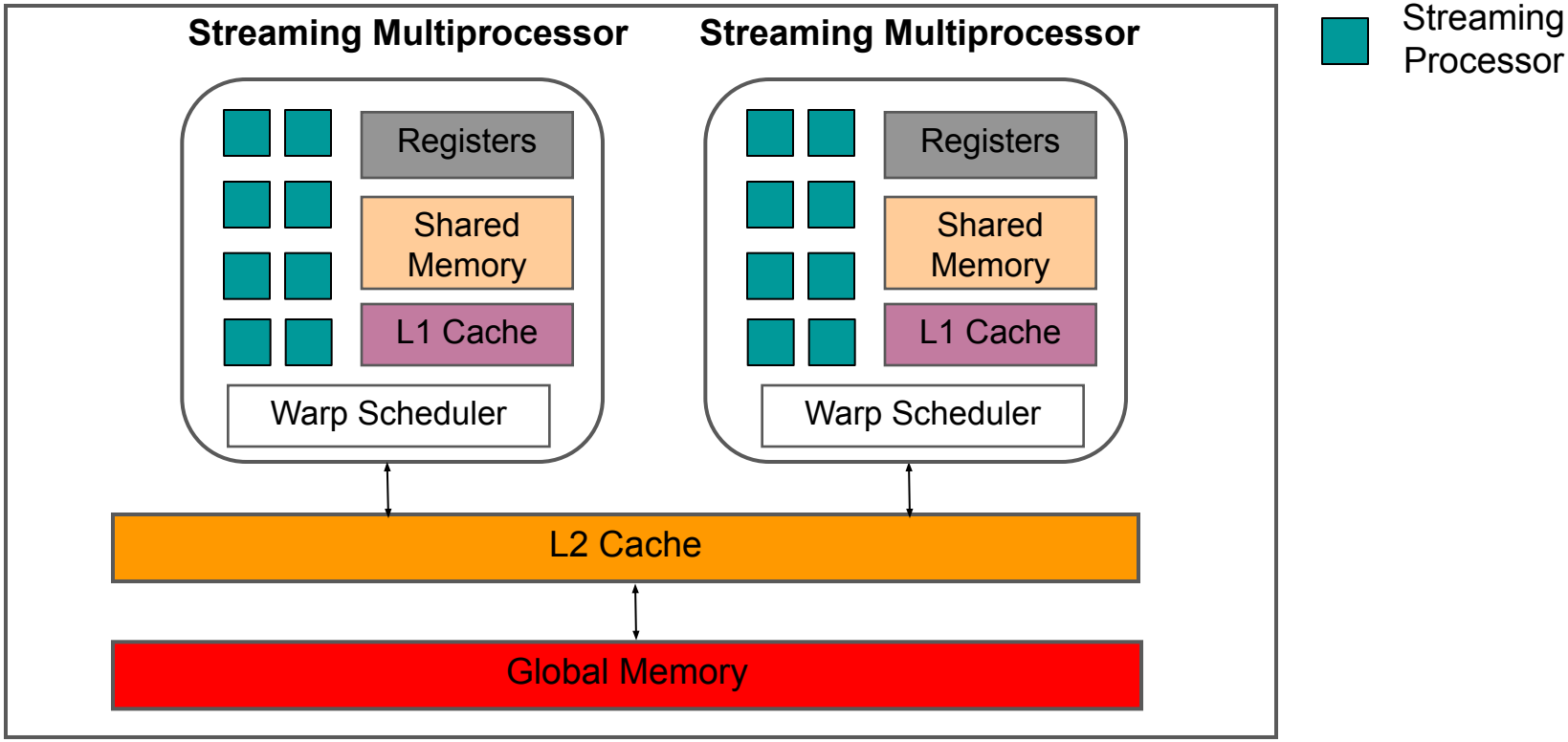
Outline

- GPUs and CUDA Programming
  - Overview of GPU memories
  - Shared Memory
  - How to program with shared memory
  - Example using shared memory

Outline

- GPUs and CUDA Programming
  - Overview of GPU memories
  - Shared Memory
  - How to program with shared memory
  - Example using shared memory

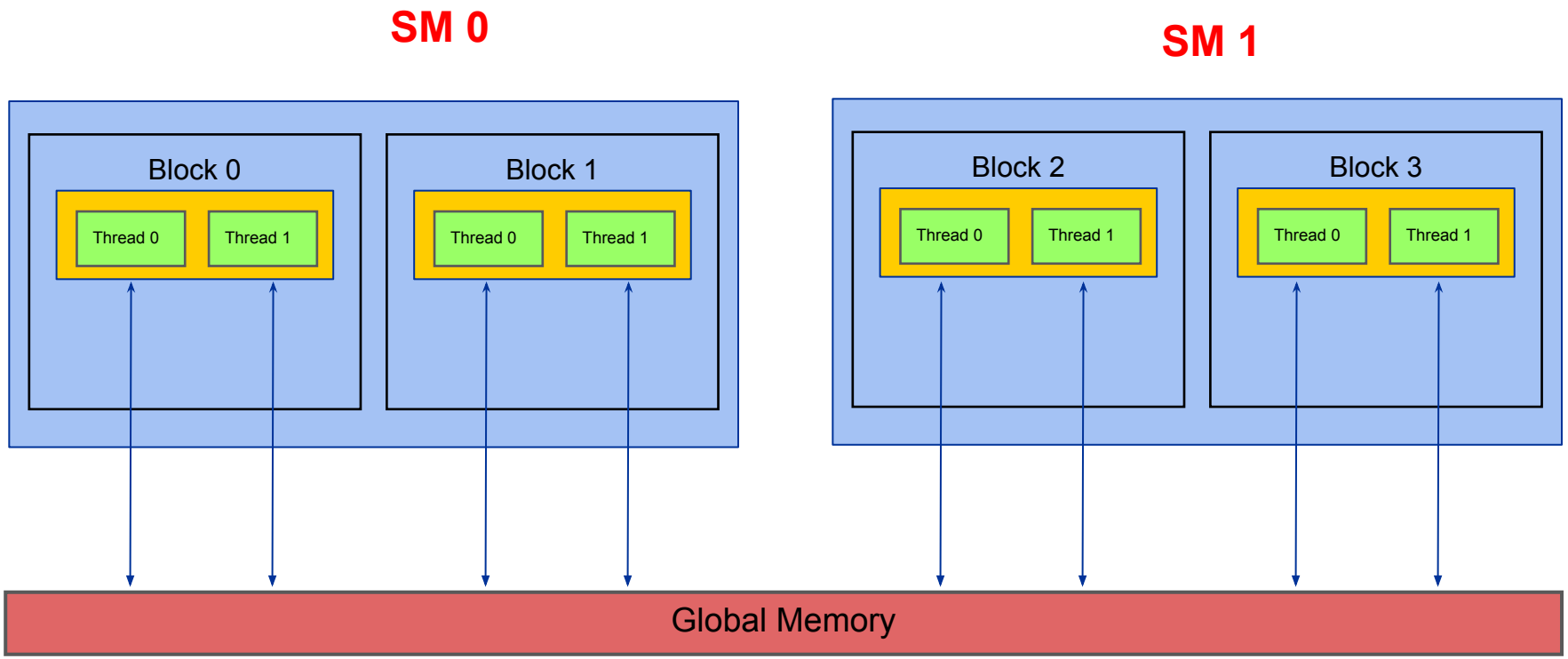
GPU Architecture



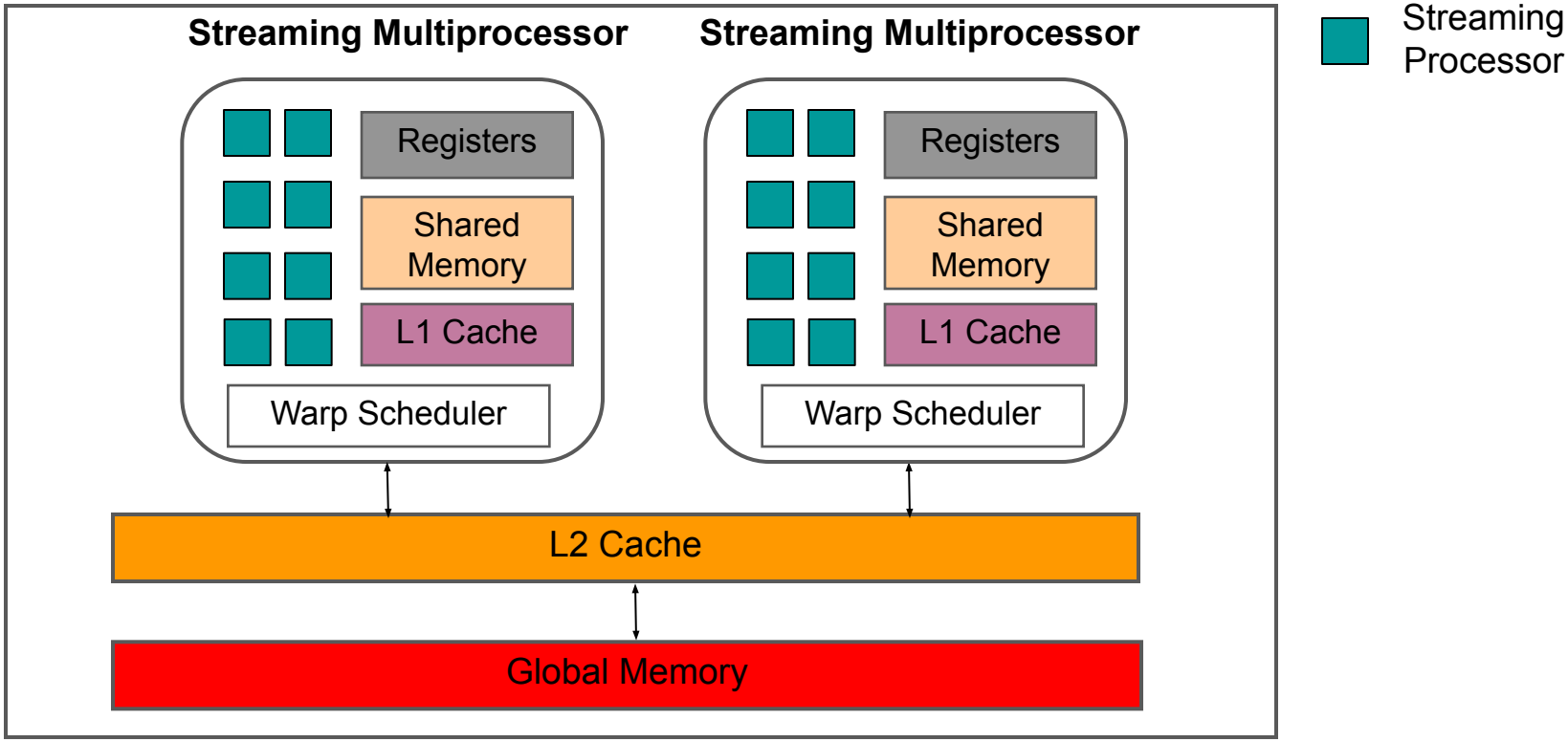
GPU Memories: Global Memory

- Global Memory
  - Main means of communicating data between host and device (cudaMemCpy)
  - Contents visible to all GPU threads
  - Long latency access (400-800 cycles)
  - Throughput ~200 GBPS
  - A100 is available in 40GB and 80GB

GPU Memories: Global Memory



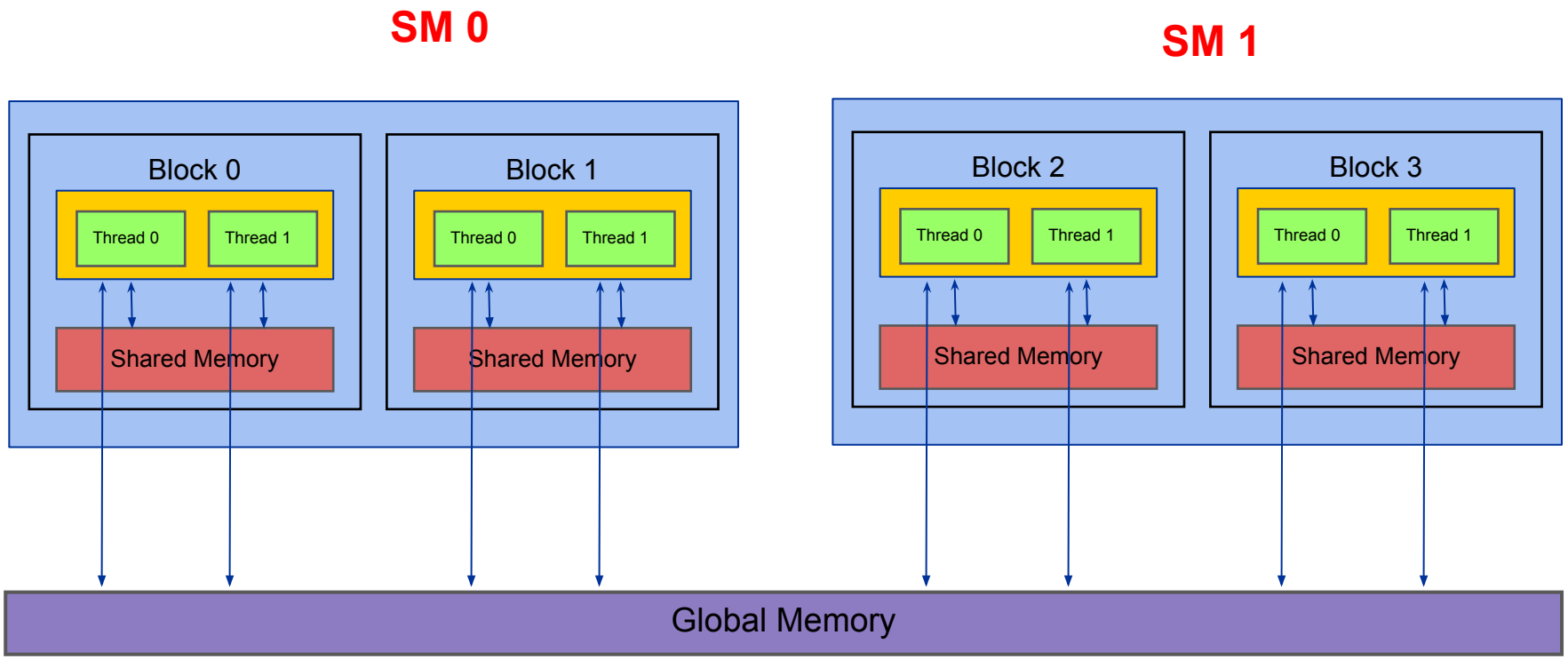
GPU Architecture



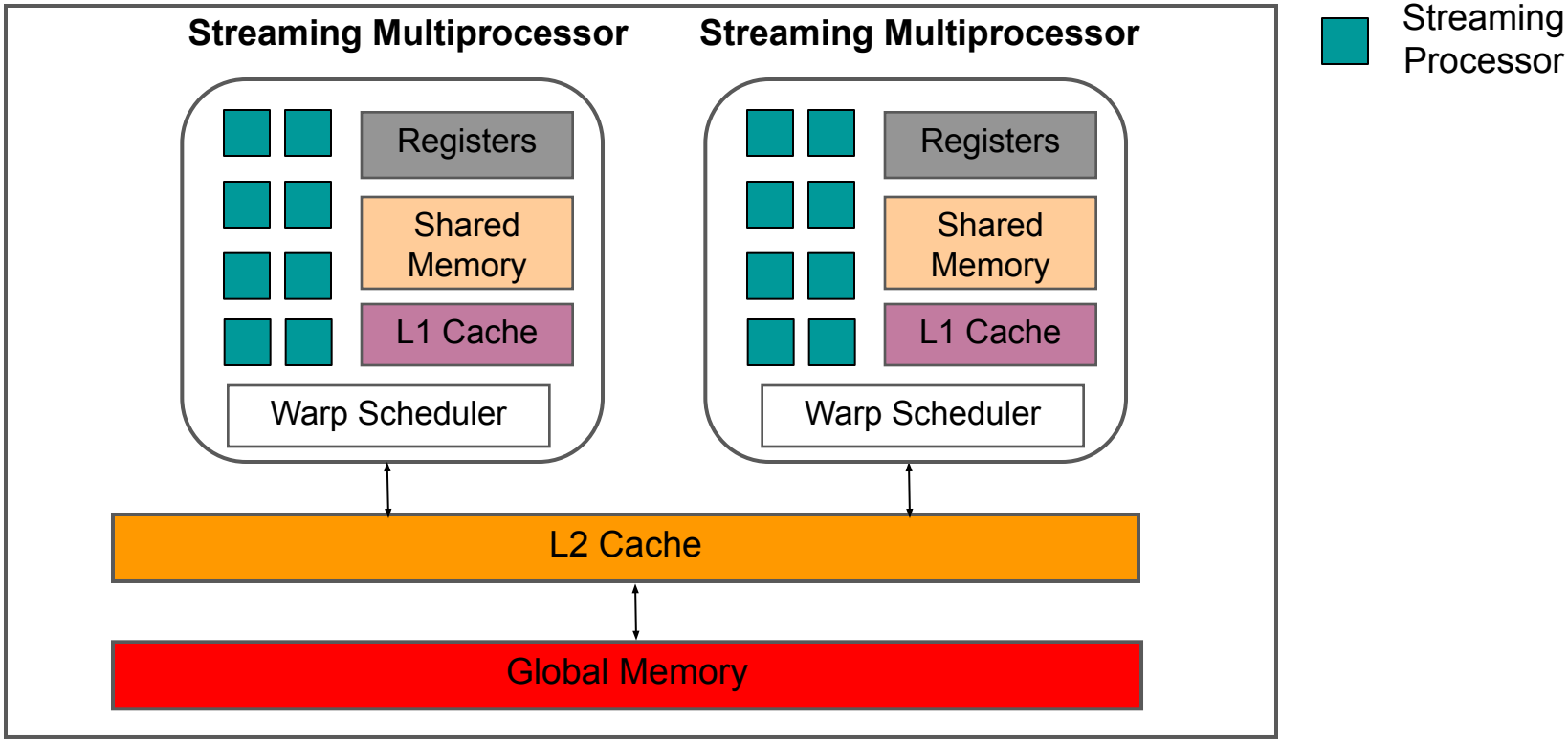
GPU Memories: Shared Memory

- Shared Memory
  - Fast: Low latency (20-30 cycles)
  - Limited: 64 KB per SM (depends on GPU)
  - High bandwidth (~1 TBPS)
  - Accessible by only the threads within thread block

GPU Memories: Shared Memory



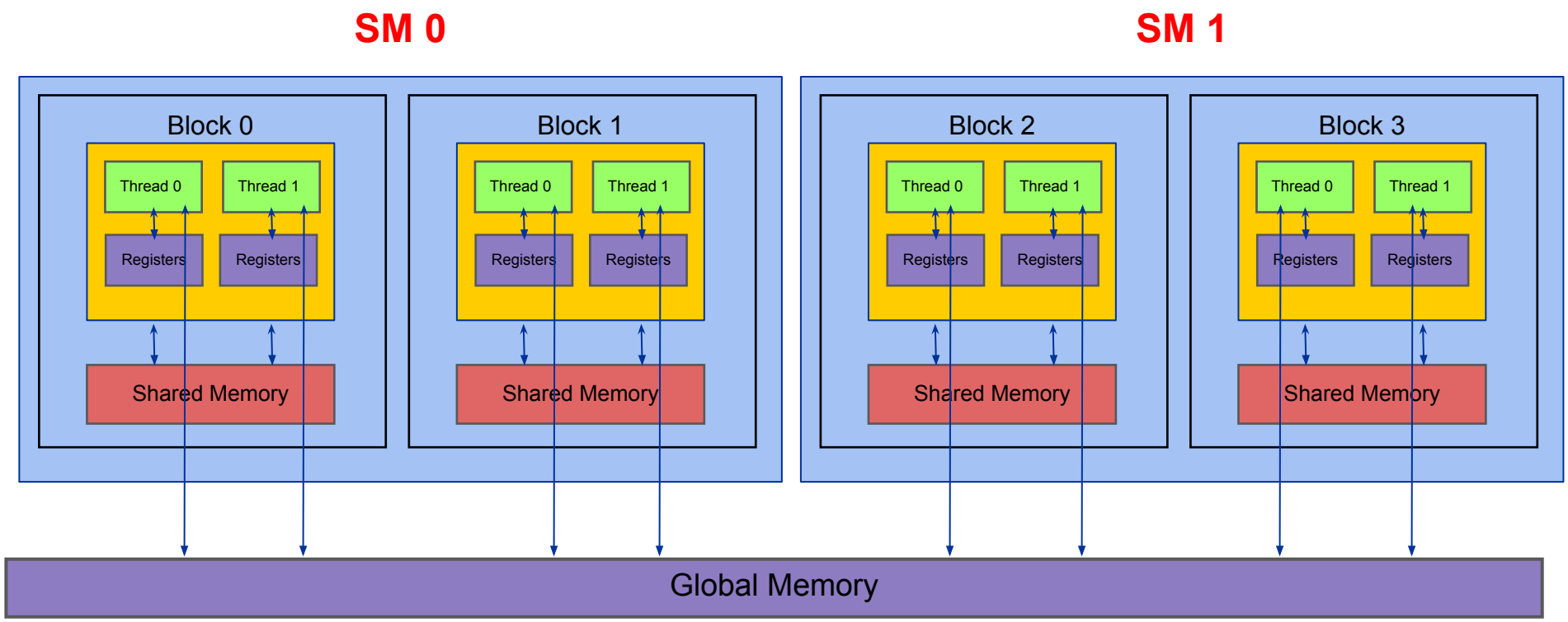
GPU Architecture



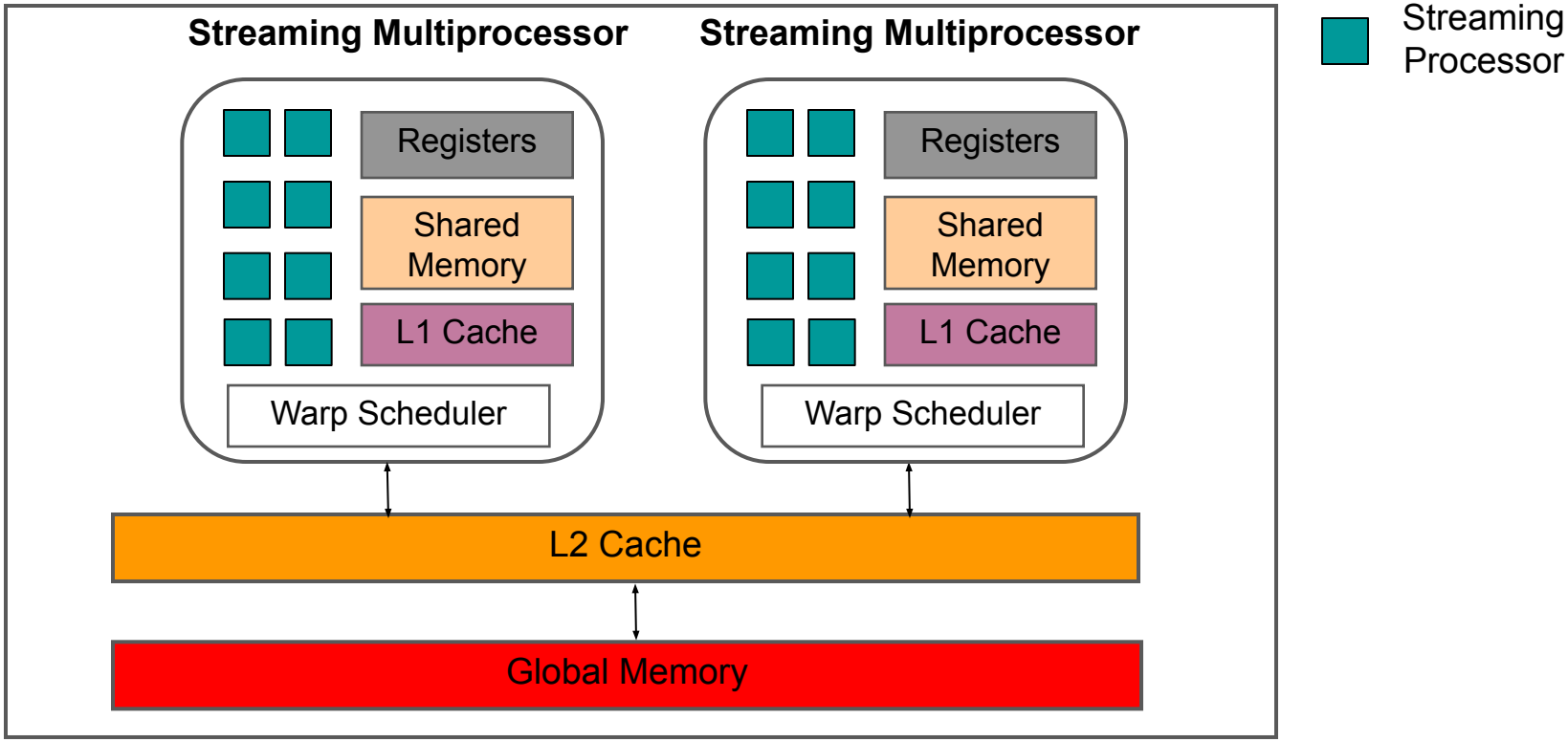
GPU Memories: Registers

- Registers
  - 32 K in number per SM (depends on GPU)
  - ~Max. 32 registers per thread (Max 1024 threads per TB)
  - Very high bandwidth (~8 TBPS)

GPU Memories: Registers



GPU Architecture



GPU Memories: Caches

- L1 cache
  - Fast storage
  - Limited in size
  - Unlike shared memory, it is system managed
  - Private to each SM
  - Configurable
    - 16 KB L1 + 48 KB Shared Memory Vice versa
- L2 cache
  - Shared among SMs
  - Latency higher than L1 cache
  - Around 768 KB (depends on SM)

Outline

- GPUs and CUDA Programming
  - Overview of GPU memories
  - Shared Memory
  - How to program with shared memory
  - Example using shared memory

Shared Memory

- Programmable L1 cache / Scratchpad memory
- Accessible only in a thread block
- Useful for repeated small data or coordination

```
__shared__  float a[N];
__shared__  unsigned s;


a[id] = id;
if (id == 0) s = 1;
```

Shared Memory: Example 1

```
#include <stdio.h>
#include <cuda.h>
#define BLOCKSIZE 1024
__global__  void dkernel() {
        __shared__  unsigned s;
        if (threadIdx.x == 0) s = 0;
        if (threadIdx.x == 50) s += 1;
        if (threadIdx.x == 100) s += 2;
        if (threadIdx.x == 0)
             printf("s=%d\n", s);
}
int main() {
    dkernel<<<1, BLOCKSIZE>>>();
    cudaDeviceSynchronize();
}
```

What is the output?

Shared Memory: Example 2

```
#include <stdio.h>

#include <cuda.h>

#define BLOCKSIZE 1024

__global__  void dkernel() {

        __shared__  unsigned s;

        if (threadIdx.x == 0) s = 0;
                __syncthreads(); // barrier across threads in a block

        if (threadIdx.x == 50) s += 1;

                __syncthreads();

        if (threadIdx.x == 100) s += 2;

                __syncthreads();

        if (threadIdx.x == 0)

                printf("s=%d\n", s);

}

int main() {

    dkernel<<<1, BLOCKSIZE>>>();

    cudaDeviceSynchronize();
}
```

s=3

Shared Memory: Example 3

```
#include <stdio.h>

#include <cuda.h>

#define BLOCKSIZE 1024

__global__  void dkernel() {

        __shared__  unsigned s;

        if (threadIdx.x == 0) s = 0;
                __syncthreads(); // barrier across threads in a block

        if (threadIdx.x == 50) s += 1;

                __syncthreads();

        if (threadIdx.x == 100) s += 2;

                __syncthreads();

        if (threadIdx.x == 0)

                printf("s=%d\n", s);

}

int main() {

    dkernel<<<2, BLOCKSIZE>>>();

    cudaDeviceSynchronize();
}
```

```
s=3
s=3
```

Shared Memory: Example 4

```
#include <stdio.h>

#include <cuda.h>

#define BLOCKSIZE 1024

__global__  void dkernel() {

        __shared__  unsigned s;

        if (threadIdx.x == 0) s = 0;
                __syncthreads(); // barrier across threads in a block

        if (threadIdx.x == 50) s += 1;

                __syncthreads();

        if (threadIdx.x == 100) s += 2;

                __syncthreads();

        if (threadIdx.x == 0)

                printf("s=%d\n", s);

}
int i;
for (i = 0; i < 10; ++i) {
        dkernel<<<2, BLOCKSIZE>>>();
        cudaDeviceSynchronize();
}
```

```
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
```

Shared Memory: Example 5

```
#include <stdio.h>
#include <cuda.h>
__global__  void dkernel() {
        __shared__  unsigned s;
        s = 5;
        if (threadIdx.x == 0) s = blockIdx.x;
        if (threadIdx.x == 1)
           printf("s=%d\n", s);
}
int main() {
    dkernel<<<2, 2>>>();
    cudaDeviceSynchronize();
}
```

What are the possible outputs?

Shared Memory: Example 5

```
#include <stdio.h>
#include <cuda.h>
__global__  void dkernel() {
        __shared__  unsigned s;
        s = 5;
        if (threadIdx.x == 0) s = blockIdx.x;
        if (threadIdx.x == 1)
            printf("s=%d\n", s);
}
int main() {
    dkernel<<<2, 2>>>();
    cudaDeviceSynchronize();
}
```

What are the possible outputs?

- s=0 s=1
- s=0 s=5
- s=5 s=1
- s=5 s=5

Shared Memory: Example 6

```
#include <stdio.h>
#include <cuda.h>
__global__  void dkernel() {
          unsigned s;
         s = 5;
         if (threadIdx.x == 0) s = blockIdx.x;
         if (threadIdx.x == 1)
             printf("s=%d\n", s);
}
int main() {
     dkernel<<<2, 2>>>();
     cudaDeviceSynchronize();
}
```

What are the possible outputs?

s=5

Exercise

- Consider a GPU:
  - Shared memory size 48KB per SM
  - No of. SMs = 1

```
__shared__  float a[512];

__shared__  unsigned s;


a[id] = id;
if (id == 0) s = 1;
```

- What is the shared memory usage per thread block? (float 4 bytes; unsigned 1 byte)
- How many max thread blocks can reside at a time in SM?

Thank you!