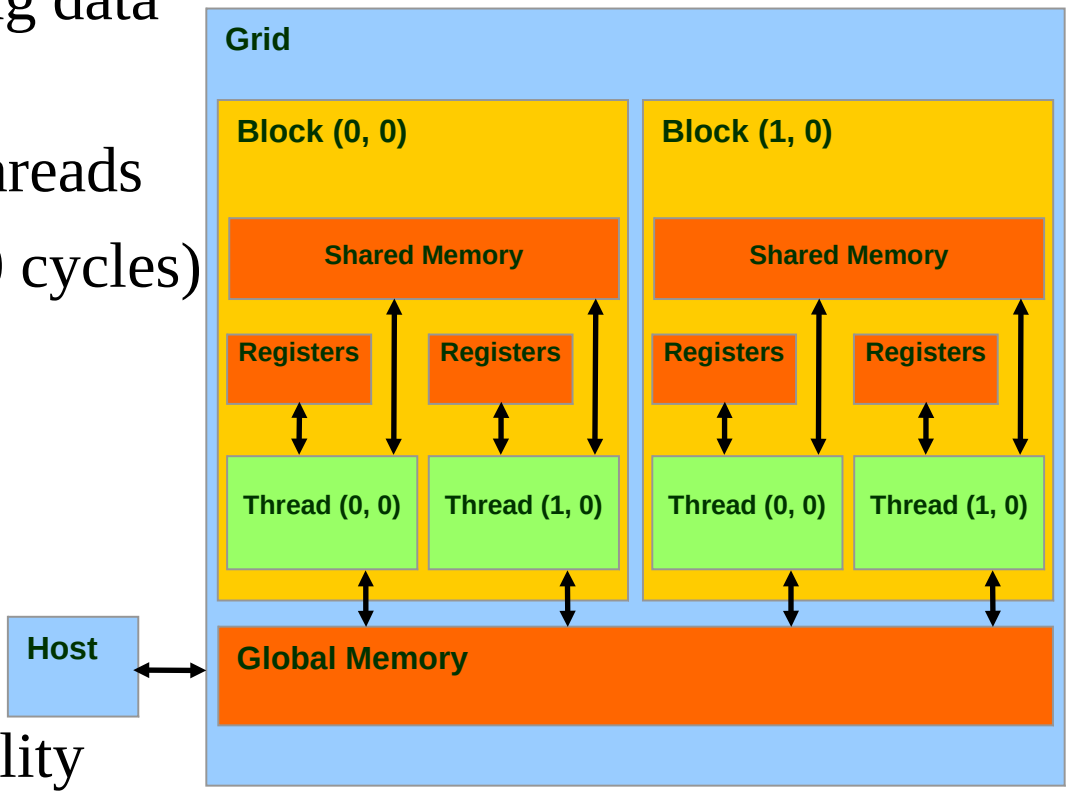
CUDA Programming

Recap - CUDA Memory Model Overview

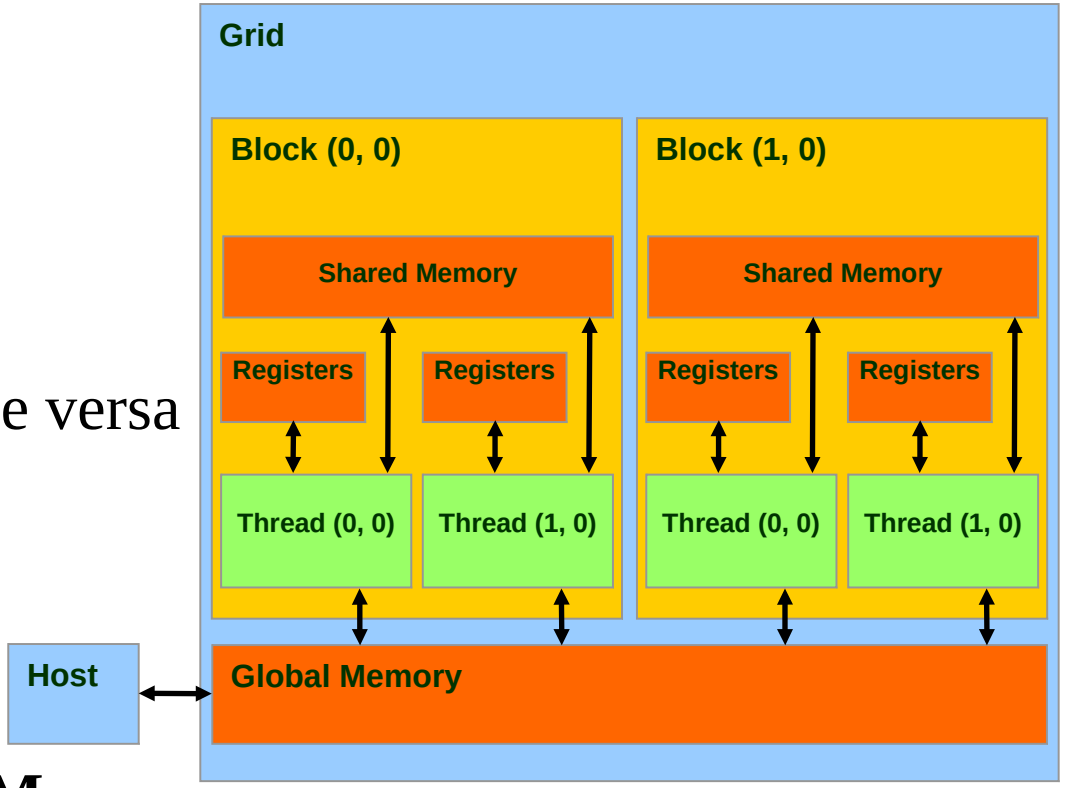
- **Global / Video memory**
  - Main means of communicating data between host and device
  - Contents visible to all GPU threads
  - Long latency access (400-800 cycles)
  - Throughput ~200 GBPS
- **Texture Memory**
  - Read-only (12 KB)
  - ~800 GBPS
  - Optimized for 2D spatial locality
- **Constant Memory**
  - Read-only (64 KB)



The numbers are typical values.

CUDA Memory Model Overview

- **L2 Cache**
  - 768 KB
  - Shared among SMs
  - Fast atomics
- **L1 / Shared Memory**
  - Configurable 64 KB per SM
  - 16 KB shared+48 KB L1 or vice versa
  - Low latency (20-30 cycles)
  - High bandwidth (~1 TBPS)
- **Registers**
  - 32 K in number, unified, **per SM**
  - ~Max. 21 registers per thread
  - Very high bandwidth (~8 TBPS)



The numbers are typical values.

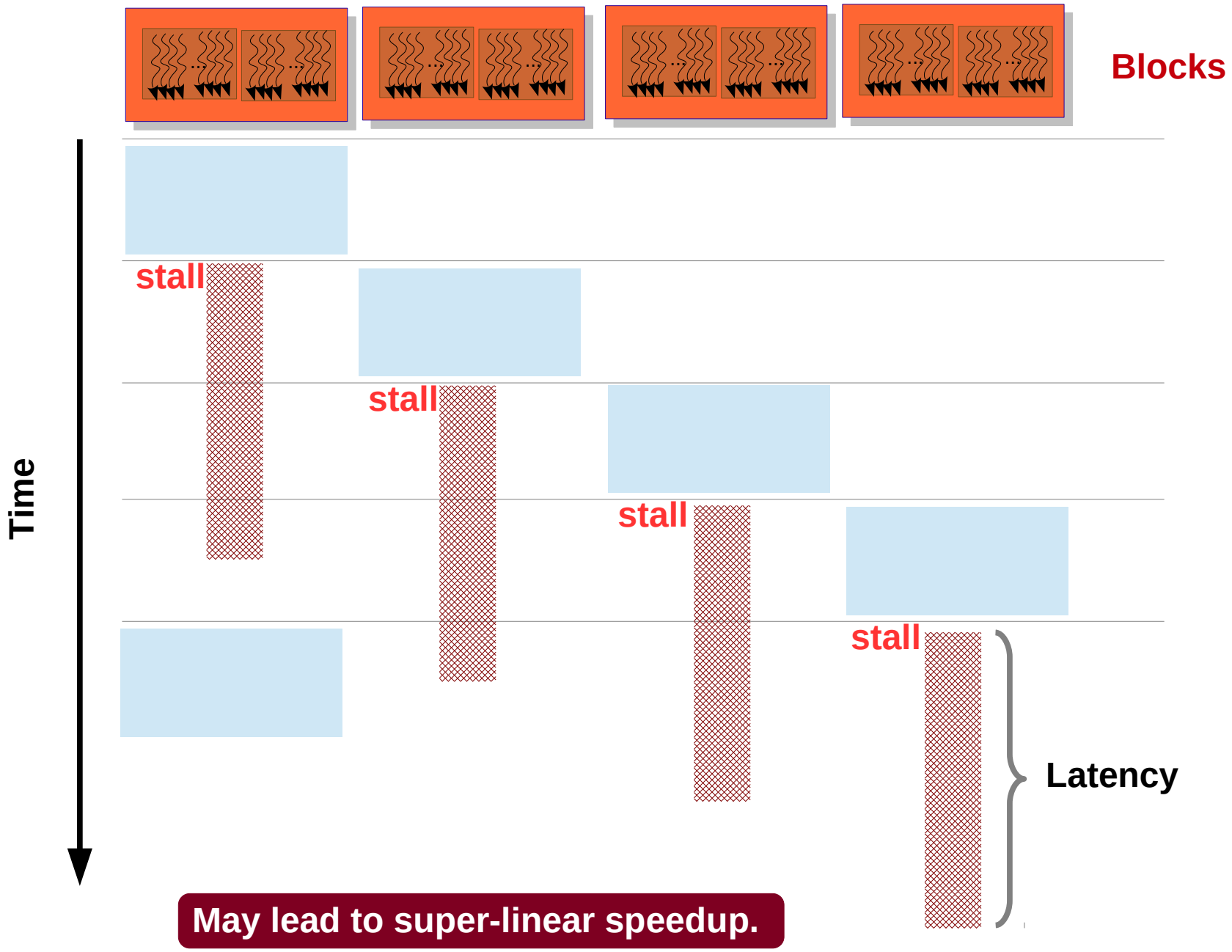
Bandwidth

- Big (wide) data bus rather than fast data bus
- Parallel data transfer
- Techniques to improve bandwidth:
  - Share / reuse data
  - Data compression
  - Recompute than store + fetch

Latency

- Latency is time required for I/O.
- Latency should be minimized; ideally zero.
  - Processor should have data available in no time.
  - In practice, memory I/O becomes the bottleneck.
- Latency can be reduced using caches.
  - CPU: Registers, L1, L2, L3, L4, RAM
  - GPUs have small L1 and L2, and many threads.
- Latency hiding on GPUs is done by exploiting massive multi-threading.

Latency Hiding



May lead to super-linear speedup.

Locality

- Locality is important on GPUs also.
- All threads in a thread-block access their L1 cache.
  - This cache on Pascal GPU is 64 KB.
  - It can be configured as 48 KB L1 + 16 KB scratchpad (or 16 KB L1 + 48 KB scratchpad or 32 KB + 32 KB).
- Programmer can help exploit locality.
- In the GPU setting, another form of spacial locality is critical.

Locality

Spatial

If **a[i]** is accessed, **a[i+k]** would also be accessed.

```
for (i = 0; i < N; ++i)
    a[i] = 0;
```

Temporal

If a[i] is accessed **now**, it would be accessed soon **again**.

```
for (i = 0; i < N; ++i) {
    a[i]   = i;
    a[i] += N;
    b[i]   = a[i] * a[i];
}
```

The localities are applicable on both CPU as well as GPU. But more applicable on CPUs.

Classwork

- Check how the localities are in the following matrix multiplication programs (on CPU).

```
for (i = 0; i < M; ++i)
        for (j = 0; j < N; ++j)
                for (k = 0; k < P; ++k)
                        C[i][j] += A[i][k] * B[k][j];
```

```
for (i = 0; i < M; ++i)
        for (k = 0; k < P; ++k)
                for (j = 0; j < N; ++j)
                        C[i][j] += A[i][k] * B[k][j];
```

Times taken for (M, N, P) = (1024, 1024, 1024) are 9.5 seconds and 4.7 seconds.

What happens on a GPU?