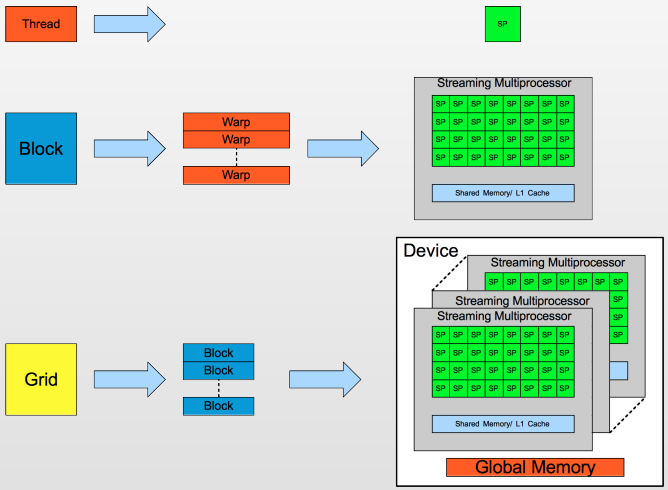
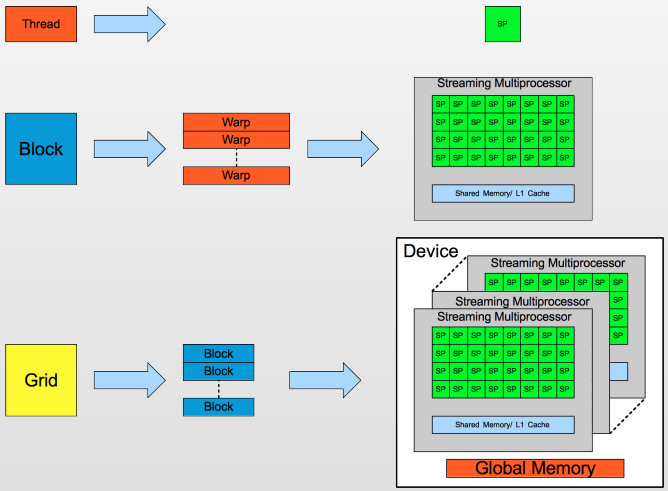
Outline

CUDA Execution Model

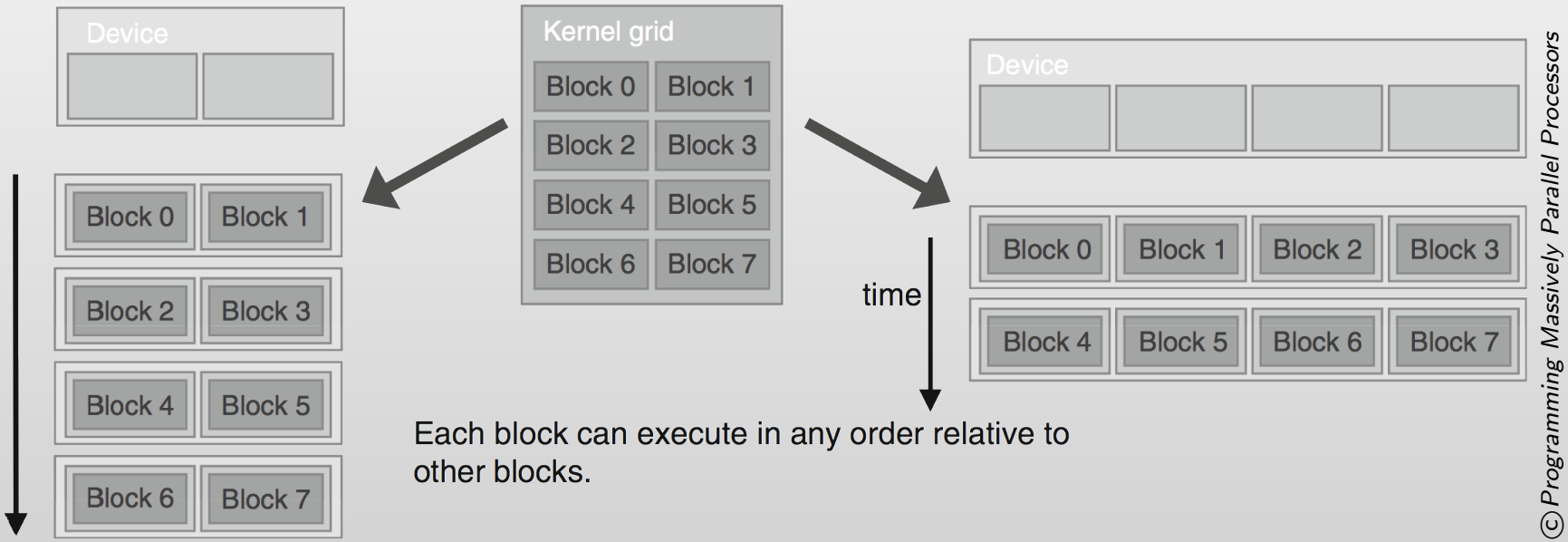


- 32 Threads = 1 *warp*
  - Smallest execution unit
  - Execute in lock-step
- Single Instruction Multiple Thread (SIMT)
  - Similar to SIMD
- Threads within the same threadblock can synchronize

CUDA Execution Model

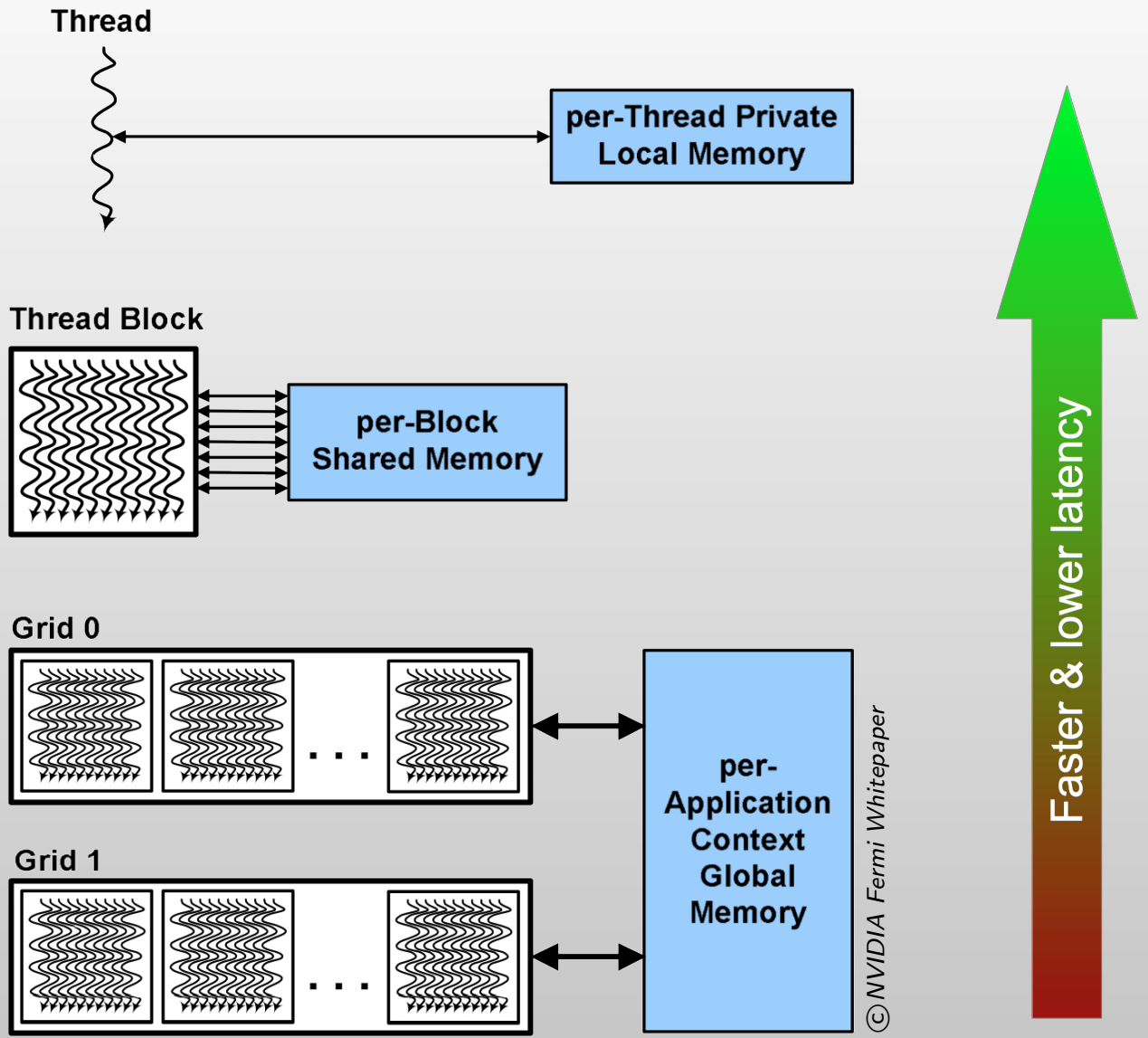


- Threadblocks can not synchronize!
- Multiple threadblocks run concurrently on the same SM
  - High thread-level parallelism
  - Hide memory latencies
- Multiple grids can be executed simultaneously for CC $\geq$ 2.0



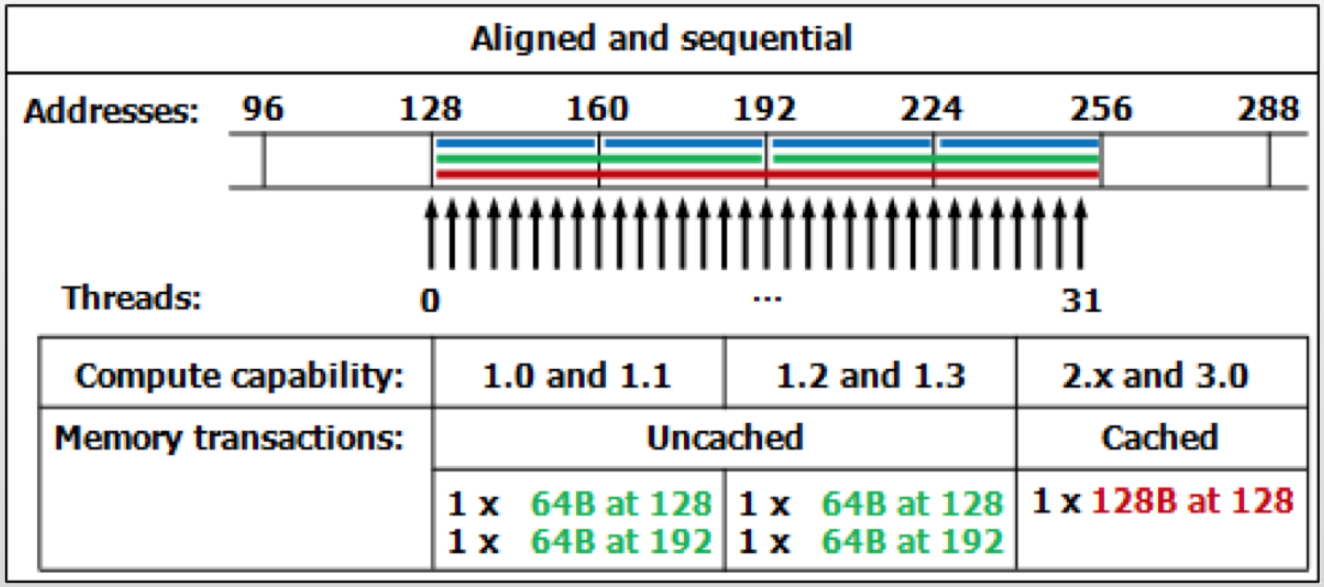
SAXPY example

CUDA Memory Model



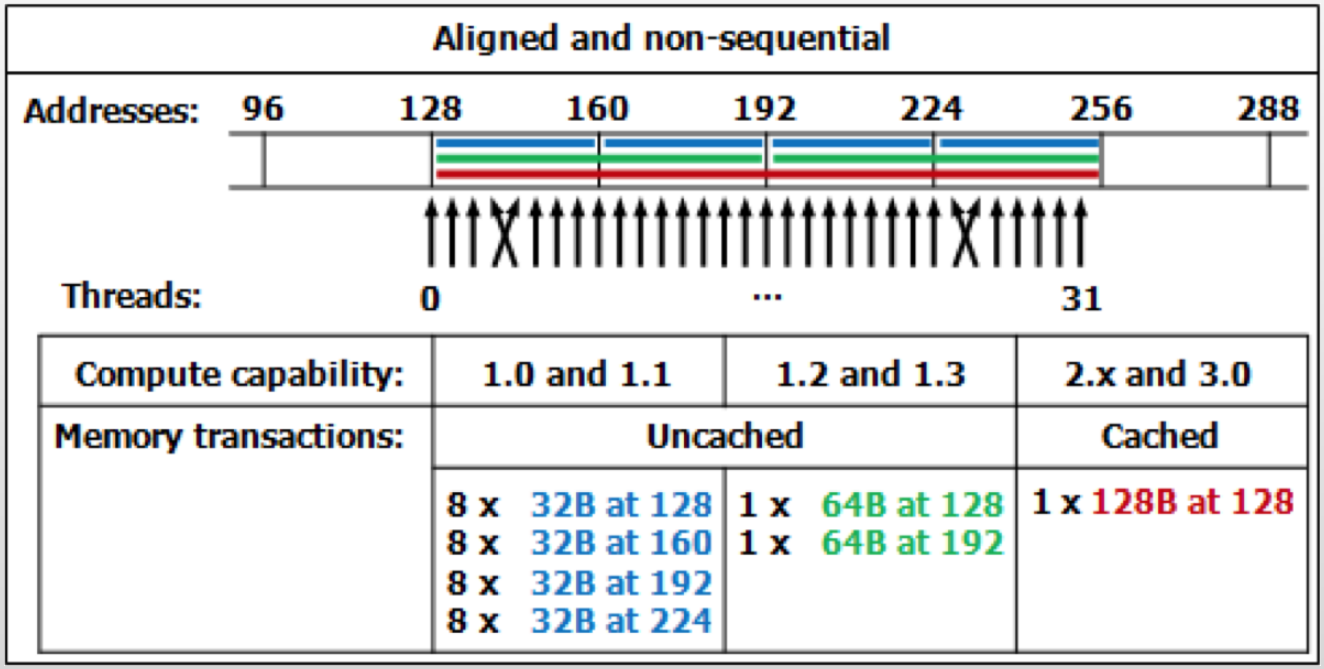
Global Memory

- Cached in L1 and L2 by default
  - Cache line is 128 bytes
- Cache only in L2 by compiling with *-Xptxas -dlcm=cg*
  - Cache line is 32 bytes
  - Reduce over-fetch for scattered memory accesses (e.g. SPMV)
- Memory accessed by . . .
  - . . . a half-warp for CC $1.x$
  - . . . a warp for CC $\geq 2.x$



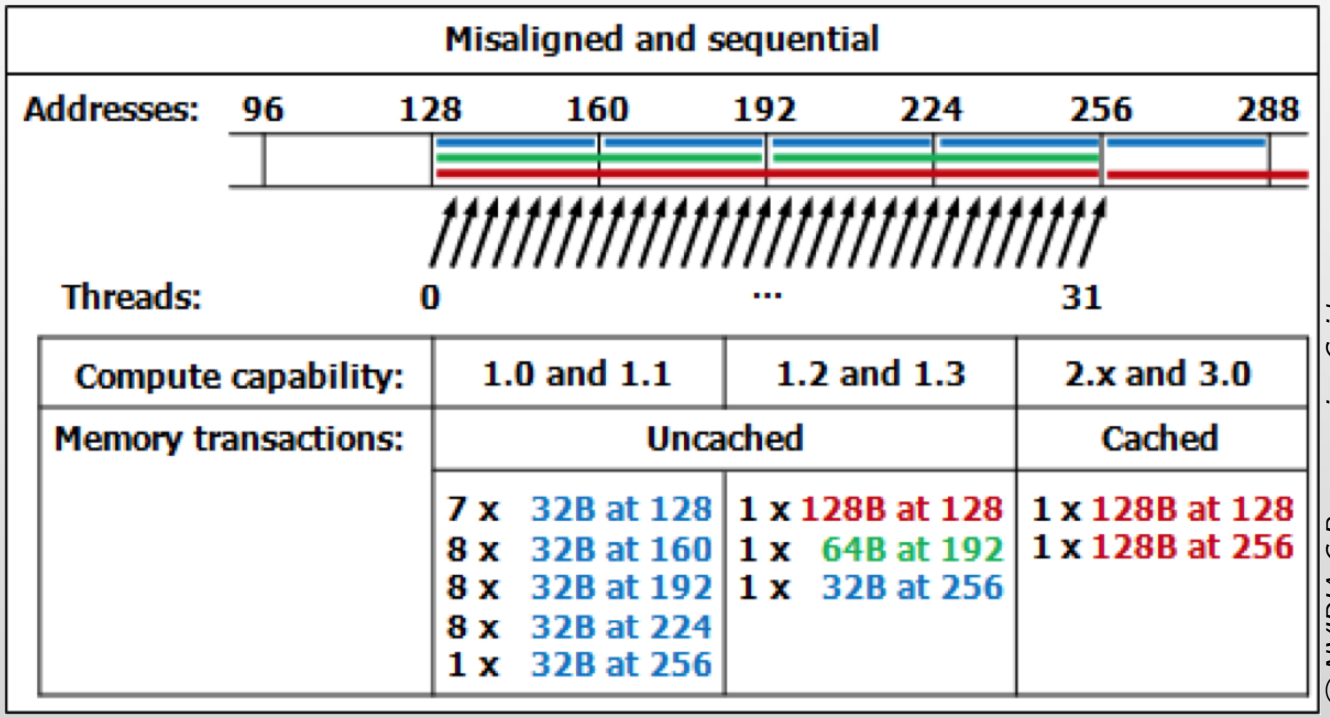
©NVIDIA C Programming Guide

- Optimization: Use coalesced memory accesses
- Threads may access the same word without issuing another memory transaction (CC $\geq$ 2.0)



© NVIDIA C Programming Guide

- Optimization: Use coalesced memory accesses
- Threads may access the same word without issuing another memory transaction (CC $\geq$ 2.0)



Compute capability:	1.0 and 1.1	1.2 and 1.3	2.x and 3.0
Memory transactions:	Uncached		Cached
	7 x 32B at 128 8 x 32B at 160 8 x 32B at 192 8 x 32B at 224 1 x 32B at 256	1 x 128B at 128 1 x 64B at 192 1 x 32B at 256	1 x 128B at 128 1 x 128B at 256

©NVIDIA C Programming Guide

- Optimization: Use coalesced memory accesses
- Threads may access the same word without issuing another memory transaction (CC $\geq$ 2.0)

Coalescing example

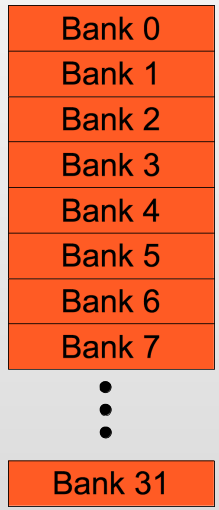
Matrix transpose example

SGEMM example

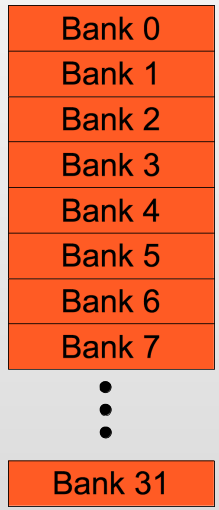
Shared Memory

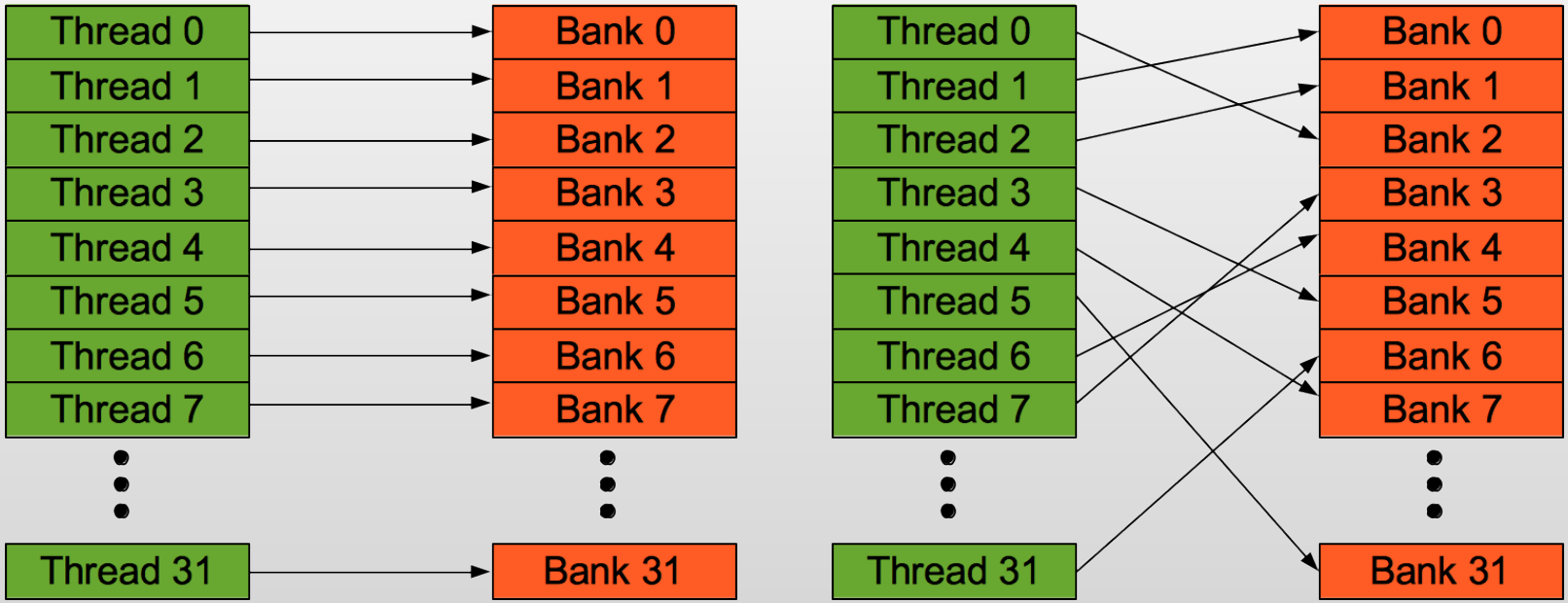
- Each SM has its own shared memory
- Higher throughput than global memory
- Lower latency than global memory
- Software-managed cache of size 16kB, 32kB or 48kB
- Threads can cooperate via shared memory
  - E.g. via *__syncthreads()*

- Divided into 32 banks
  - Successive 4bytes map to successive banks
  - Each bank can service 4bytes per cycle
- Access to the same bank *may* result in serialization
  - So called *bank conflicts*
- Use *__shared__* keyword to declare a variable as shared
- Memory accessed by . . .
  - . . . a half-warp for CC 1.x
  - . . . a warp for CC $\geq$ 2.x



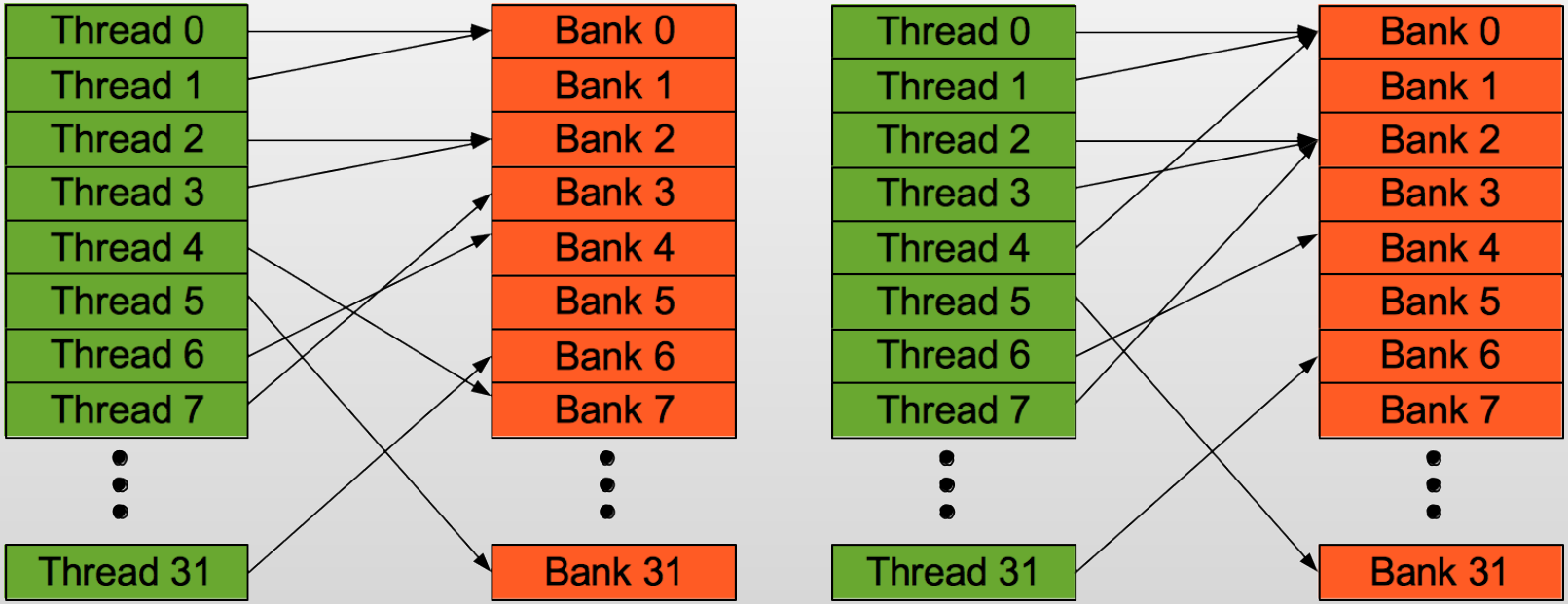
- Divided into 32 banks
  - Successive 4bytes map to successive banks
  - Each bank can service 4bytes per cycle
- Access to the same bank *may* result in serialization
  - So called *bank conflicts*
- Use *__shared__* keyword to declare a variable as shared
- Memory accessed by . . .
  - . . . a half-warp for CC $1.x$
  - . . . a warp for CC $\geq 2.x$
    - **No bank conflicts between different warps**





(a) Linear addressing. (b) Random permutation.

Figure: No bank conflicts.



(a) Two-way bank conflict. (b) Three-way bank conflict.

Figure: Bank conflicts.

Shared memory matrix transpose example

Homework - SGEMM

- Modify the SGEMM example of the lecture such that it utilizes the shared memory of the SMs
- Objective:
  - The results must be correct
  - The shared-memory version should be faster than the original version
- **Deadline**: Monday, June 3rd at 2pm.

Thread divergence

```
if( isOdd( idx ) )
    data[idx] = sin( data[idx] );
else
    data[idx] = 1.0 / data[idx];
```

- Threads within the same warp can follow different execution paths
- Why is it important?
  - Performance penalty of up to 32x
- How can we avoid this?
  - Keep divergence to threads belonging to different warps

SDOT example

Pinned Memory

- *cudaMallocHost* allocates pinned memory
- Pinned memory is required for
  - Asynchronous memory transfers
  - Overlapping computation with communication
- Higher bandwidth than non-pinned memory
- Warning: Too much pinned memory can decrease performance!

Pinned Memory example

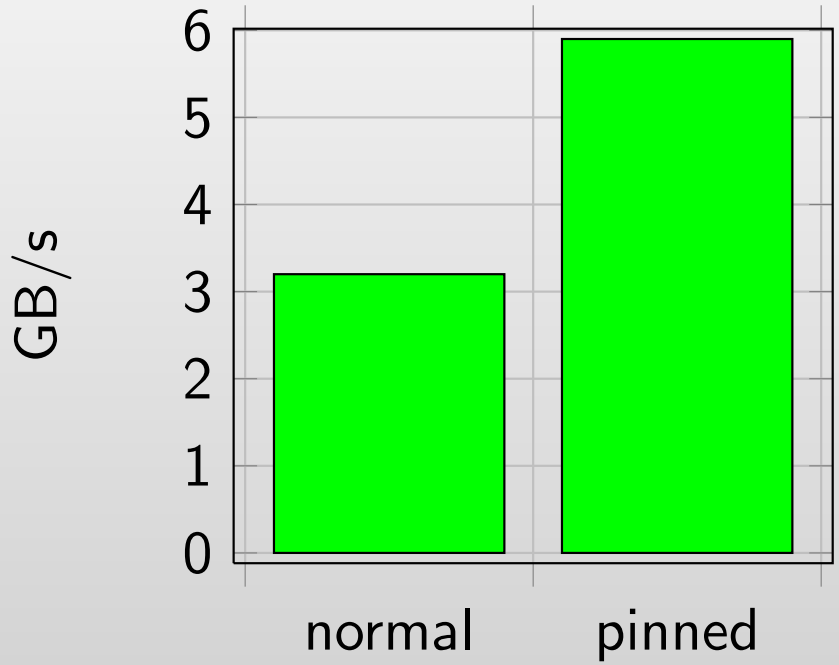


Figure: HtD data transfer for NVIDA Quadro 6000.