

OPENCL

Episode 4 - Memory Layout and Access

David W. Gohara, Ph.D.

Center for Computational Biology

Washington University School of Medicine, St. Louis

email: sdg0919@gmail.com twitter: iGotchi

THANK YOU



Q&A

- Function calls from within kernels
- Use of `clFinish()`

Q&A

- Function calls from within kernels
 - Only functions that the OpenCL spec supports may be called from kernels
 - Functions like `rand()`, `printf()` are NOT guaranteed to work
 - May work on CPU if implementation allows it

Q&A

- Use of clFinish()
 - clFinish is a brute force method that causes the CPU to block
 - Use it to force the command queue to perform all operations before proceeding
 - Can be used to force synchronization
 - On blocking operations and in-order execution it's not necessary if the implementation is compliant with the spec (sections 5.1 and 5.8, OpenCL reference)

GPU ARCHITECTURE

- Will be talking about NVIDIA hardware specifically
- Will be using NVIDIA terminology
- Remember:
 - Thread == Work-item
 - Thread Block == Work-group

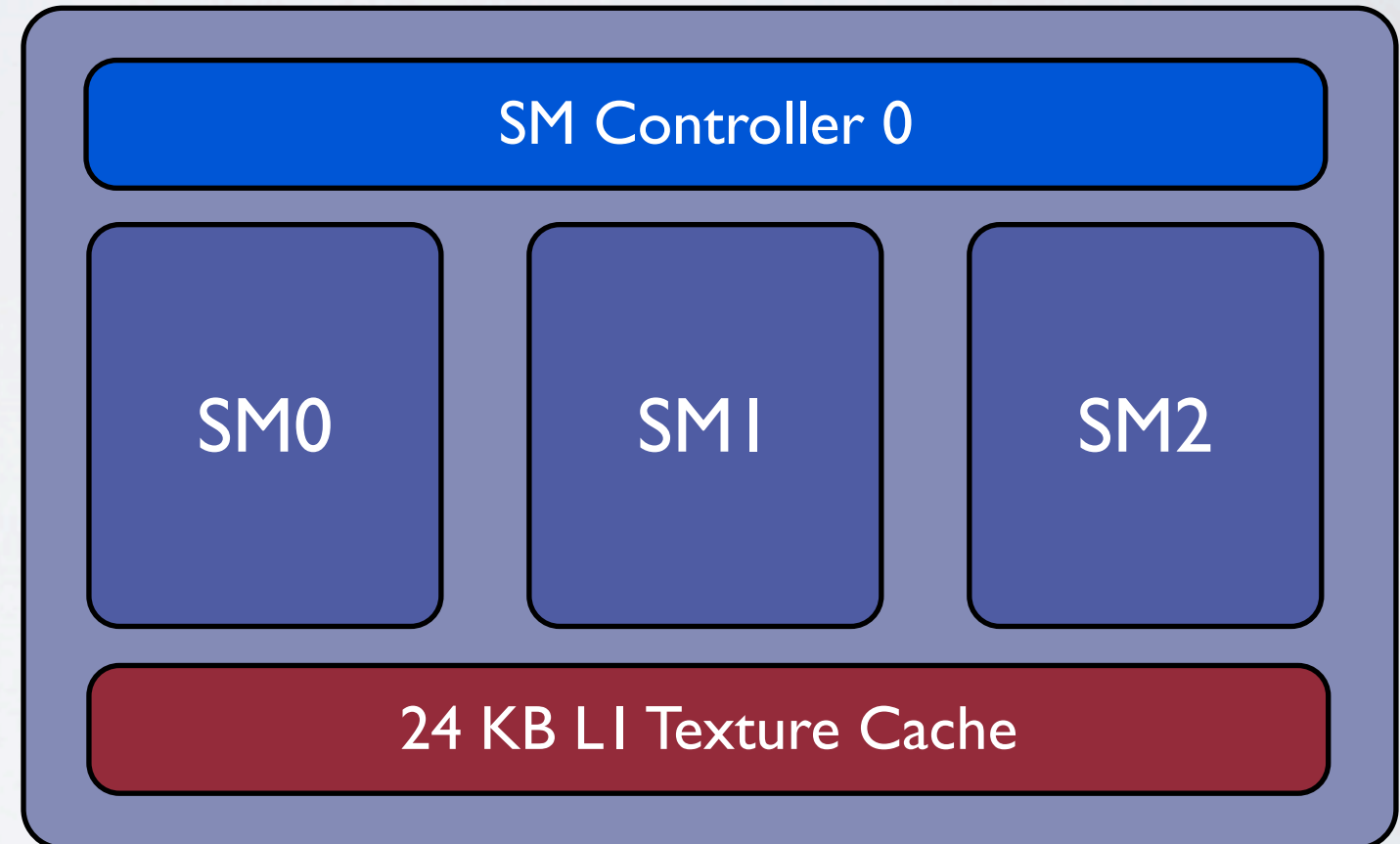
GPU ARCHITECTURE - GTX 285

- 10 Thread Processing Clusters (TPC)
- 30 Streaming Multiprocessors (SM)
 - Contains 8 Streaming Processors (SP)
 - 2 Special Function Units (SFU)
 - 1 double precision unit
 - Local memory (shared memory) for the 8 streaming processors

THREAD PROCESSING CLUSTER

- One SM controller
- Three SM's per TPC
- 24 KB LI Texture Cache

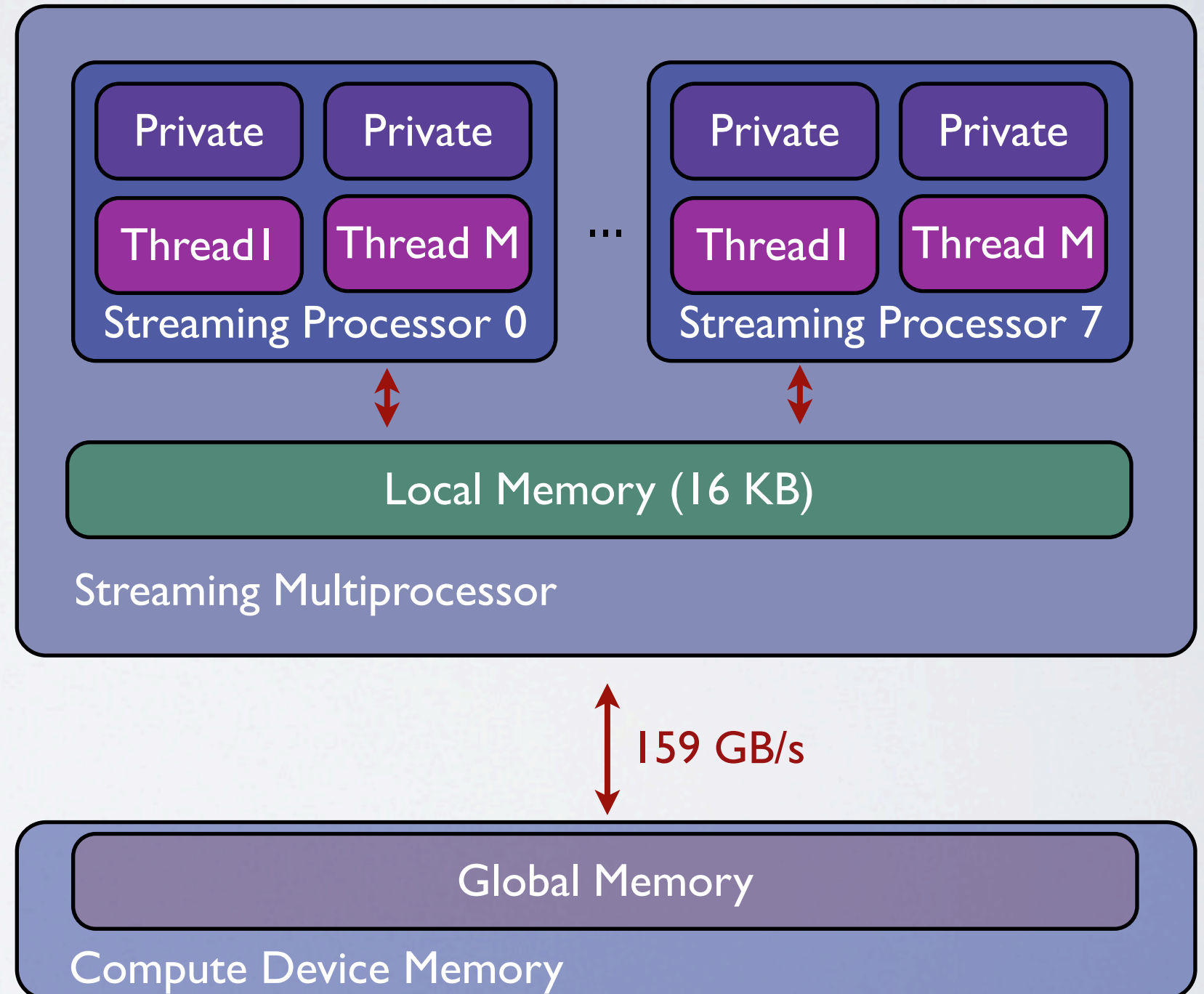
TPC 0



Compute Device

STREAMING MULTIPROCESSOR

- Each SM can execute 8 thread blocks concurrently
- Each SM thread scheduler groups threads within thread blocks into warps
- Warps are defined as consisting of 32 threads



WARPS

Half-warp

Half-warp

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

- Thread blocks are partitioned into warps - each warp is 32 threads
- Each thread will invoke an instance of your kernel
- Each thread moves in lock step with the other
- Only threads in the same thread block can share data between each other

INSTRUCTION SCHEDULING

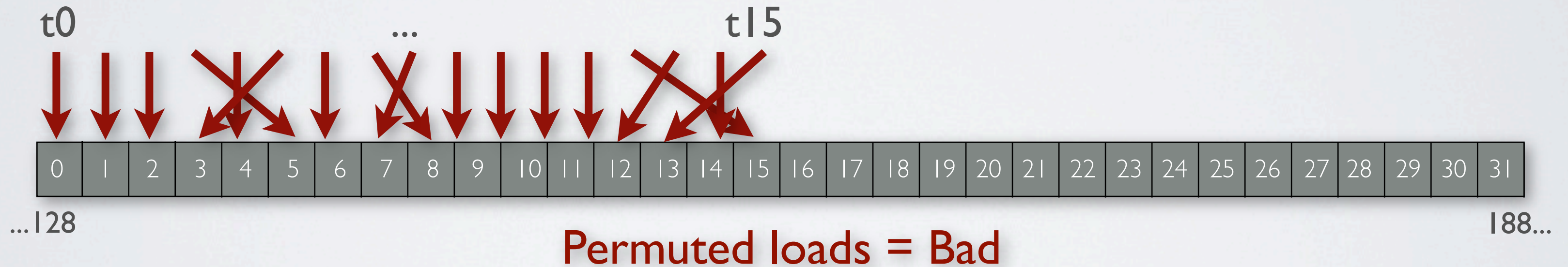
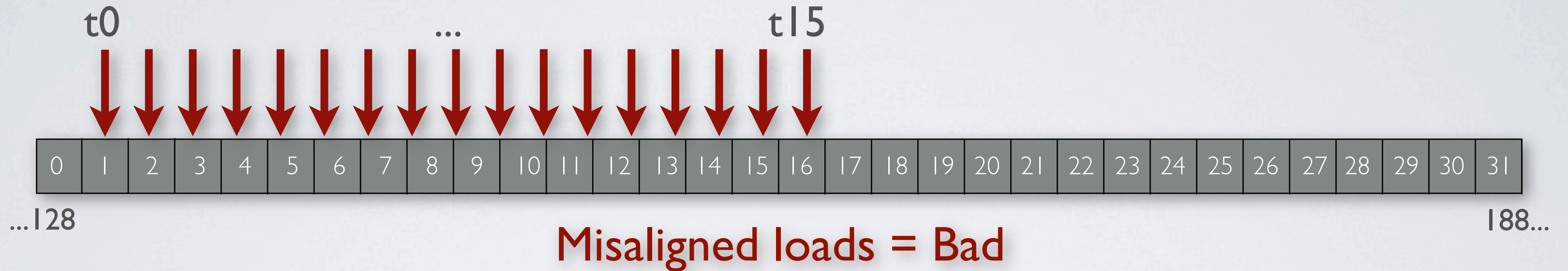
- Each instruction is executed at the same time by each thread in a warp
- Context switching between warps is fast and is used to hide memory latency
- Barriers are used to force synchronization points between threads in a thread block (work group)

```
if((iatom + lid) < natoms){
    shared[lid]          = ax[iatom + lid];
    shared[lid + lsize]  = ay[iatom + lid];
    shared[lid + 2*lsize] = az[iatom + lid];
    shared[lid + 3*lsize] = charge[iatom + lid];
    shared[lid + 4*lsize] = size[iatom + lid];
}else{
    shared[lid]          = ax[iatom + gid];
    shared[lid + lsize]  = ay[iatom + gid];
    shared[lid + 2*lsize] = az[iatom + gid];
    shared[lid + 3*lsize] = charge[iatom + gid];
    shared[lid + 4*lsize] = size[iatom + gid];
}
```

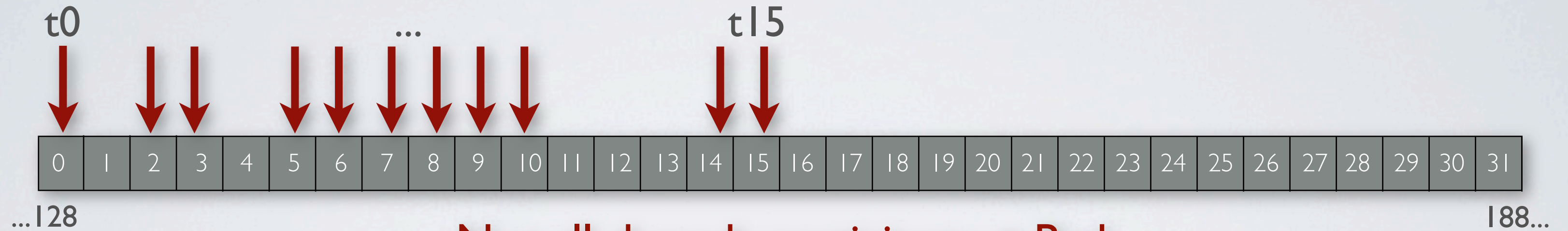

LOCAL MEMORY

- 16 KB per SM
- 4096 entries organized into 16 banks
- Assuming no bank conflicts shared memory is as fast as the register file
- Used to coordinate data sharing between threads within a work group
- Threads can cooperatively load data into shared memory (coalescing)
- A thread in a work group can load data on behalf of another thread
- Once loaded into shared memory data can be accessed element wise with no performance penalty

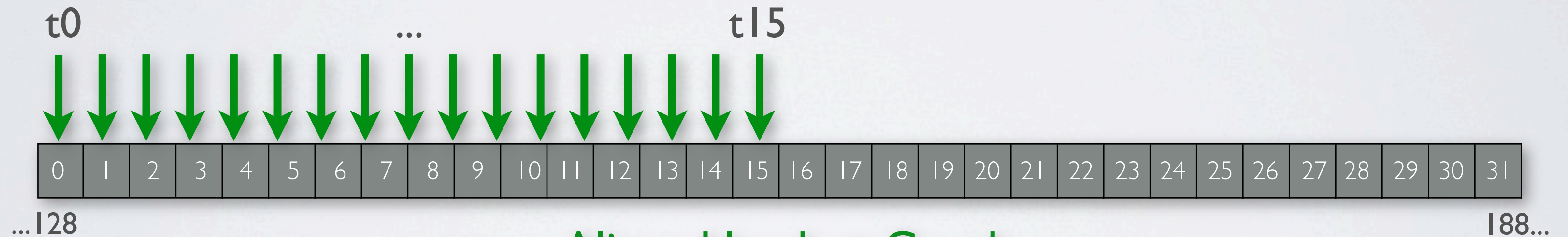
MEMORY ALIGNMENT



MEMORY ALIGNMENT



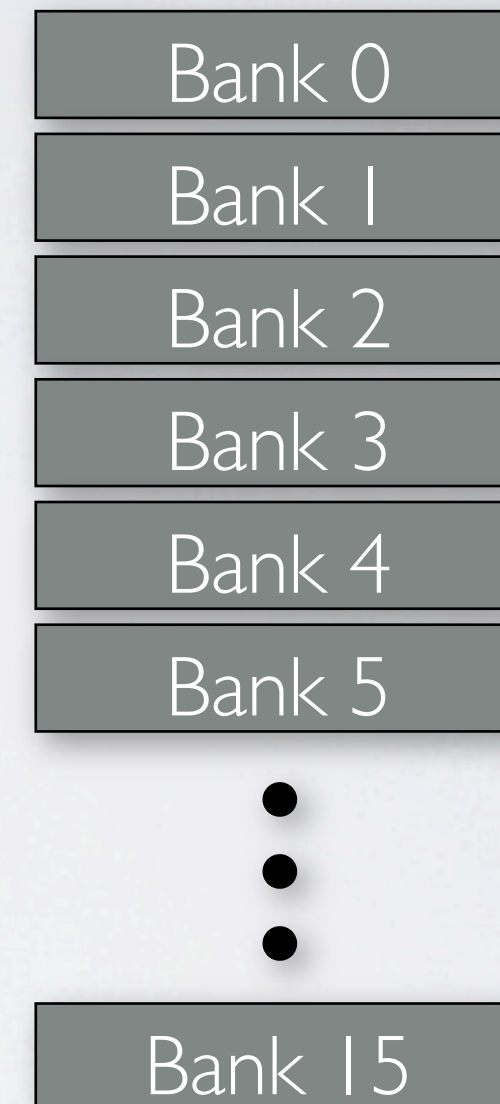
Not all threads participate = Bad



Aligned loads = Good

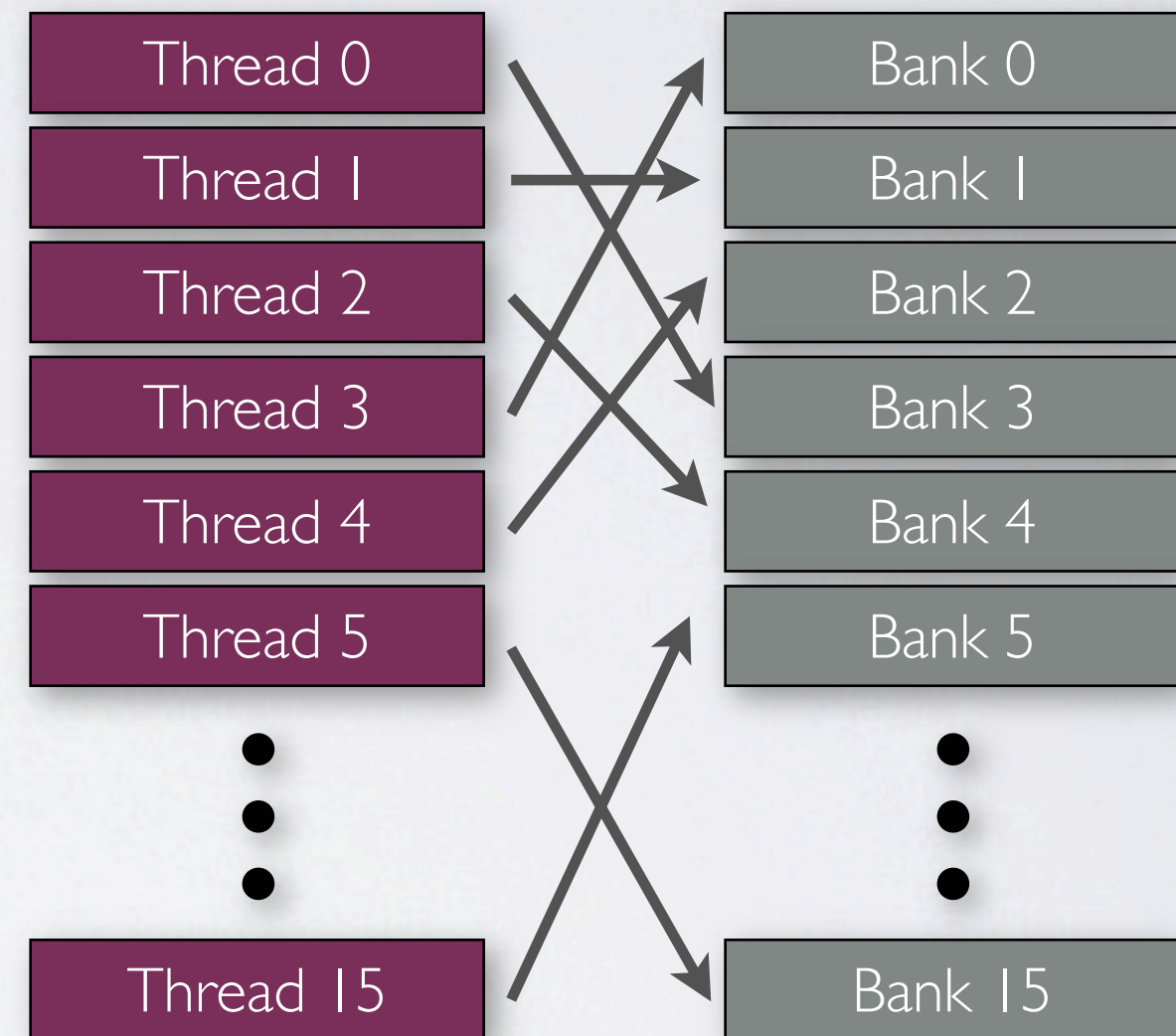
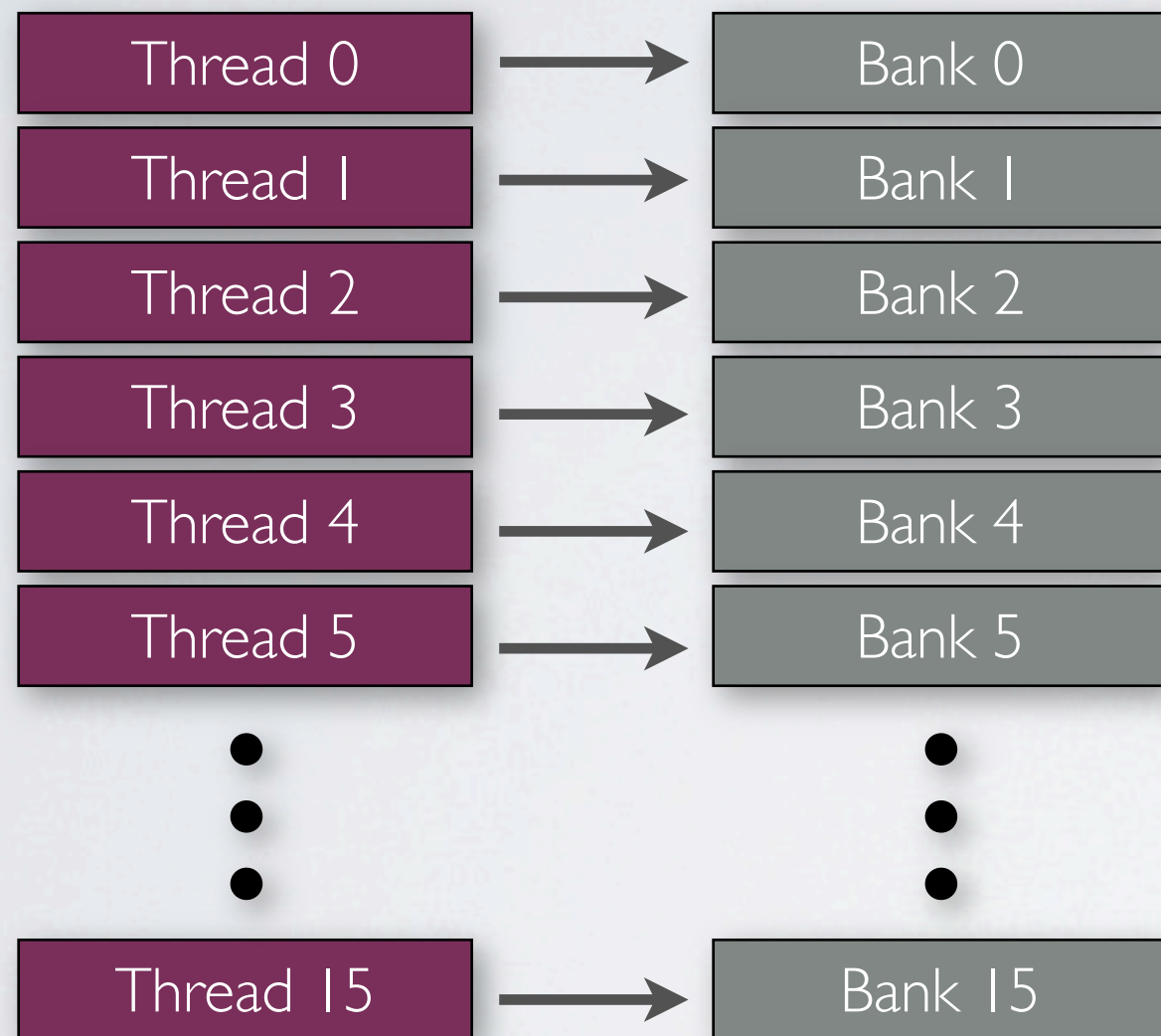
DATA LOADS

- Many threads access memory simultaneously so memory is divided into banks
- 16 banks on current hardware
- Multiple simultaneous accesses to the same bank **may** result in a bank conflict



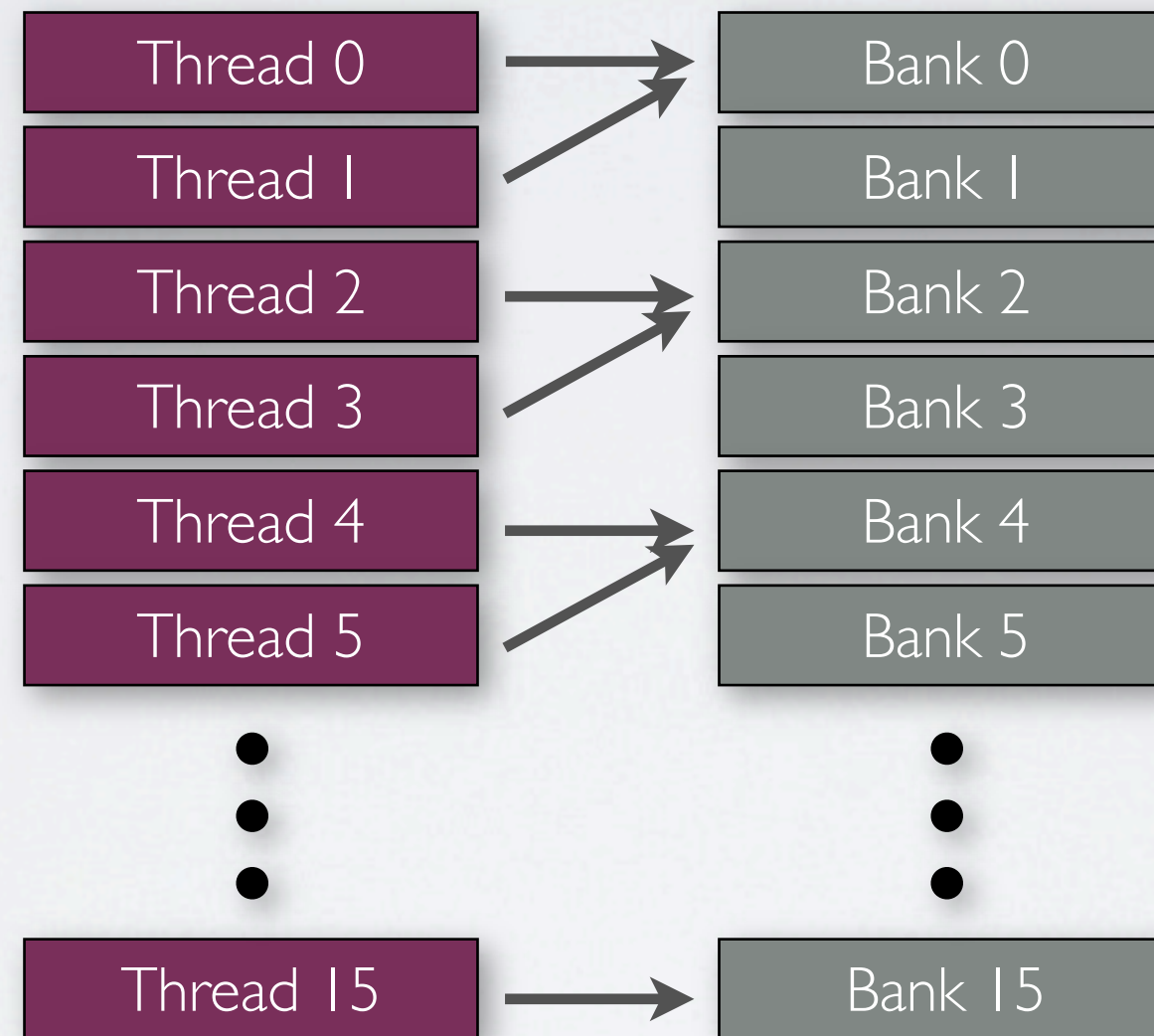
BANK ADDRESSING

No Conflicts



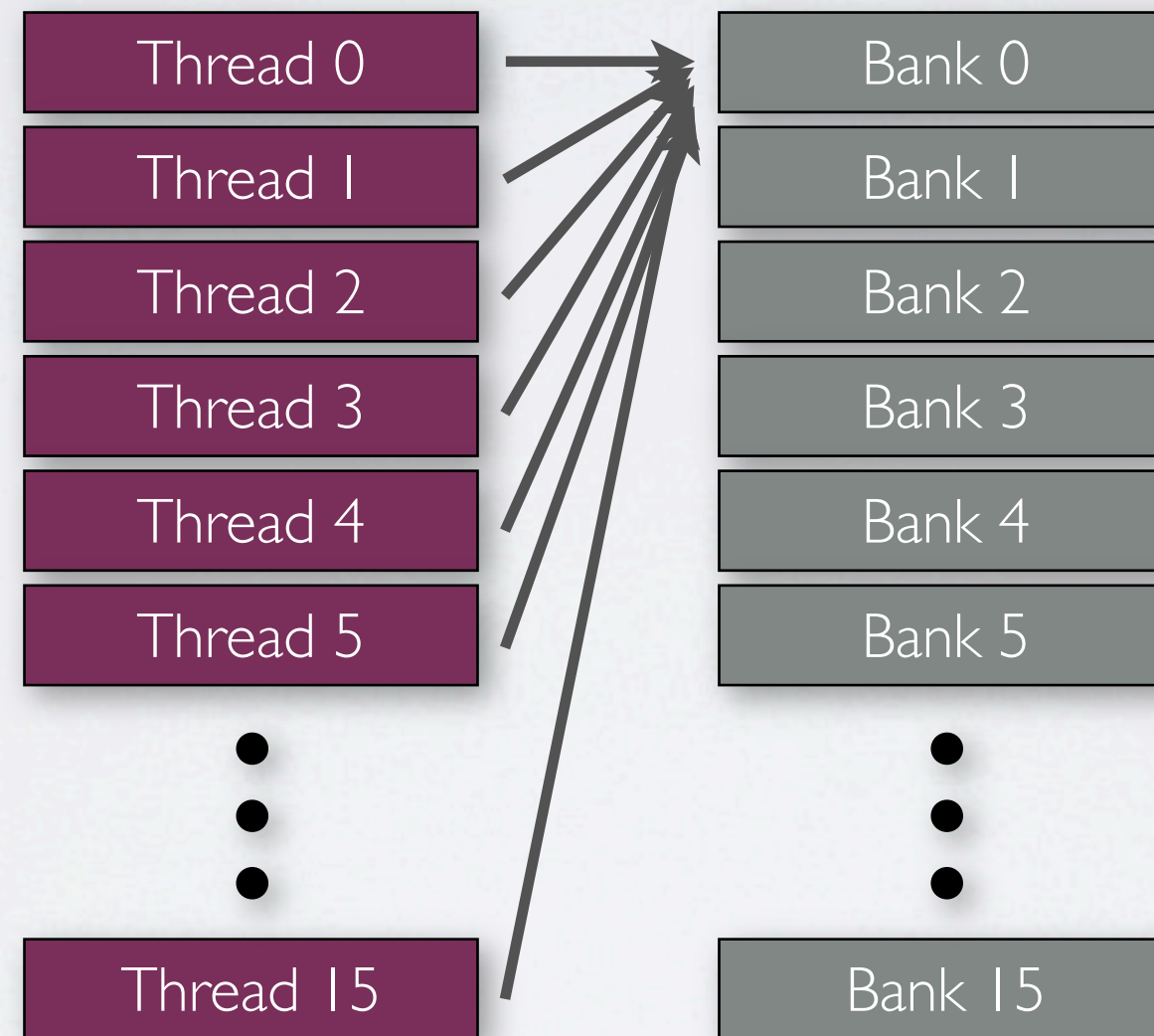
BANK ADDRESSING

Conflicts



BANK ADDRESSING

No Conflicts - Exception

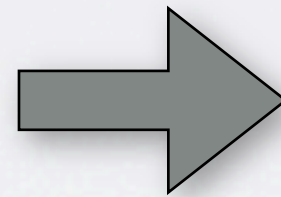


Broadcast

MATRIX TRANSPOSE

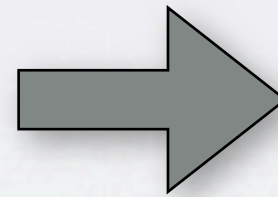
- Effectively swap elements along each axis (x,y) to (y,x) in a multidimensional array
- Excellent example of the benefits of using shared memory
- Excellent example for illustrating coalesced loads and stores
- **Assume for this example a half warp consists of 4 threads (it doesn't in real life!)**

MATRIX TRANSPOSE



MATRIX TRANSPOSE

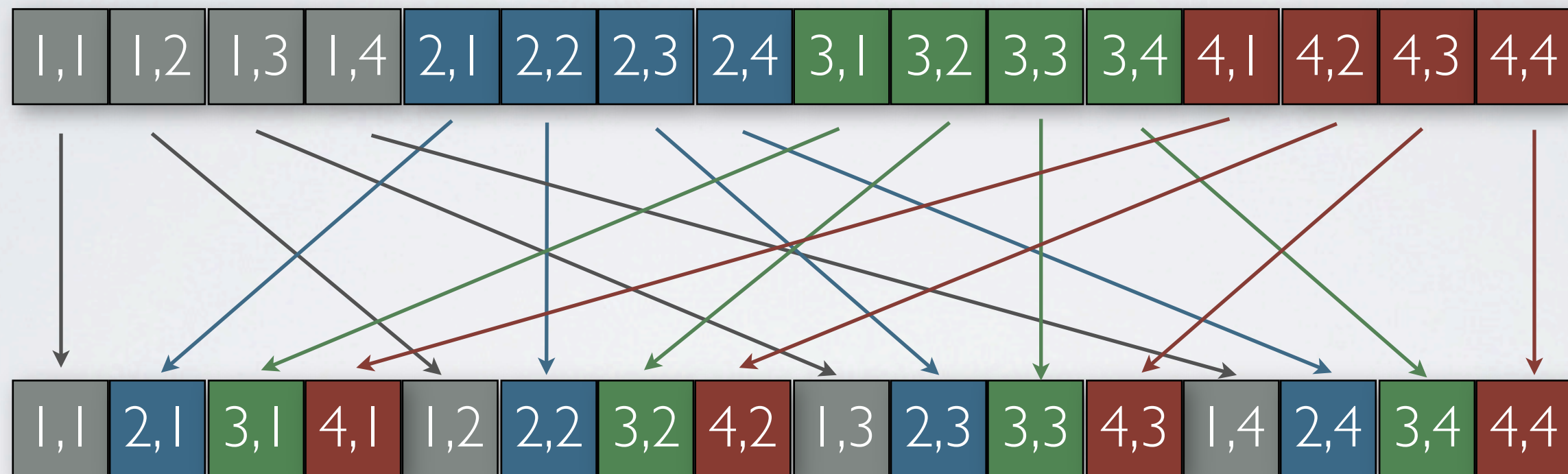
1,1	1,2	1,3	1,4
2,1	2,2	2,3	2,4
3,1	3,2	3,3	3,4
4,1	4,2	4,3	4,4



1,1	2,1	3,1	4,1
1,2	2,2	3,2	4,2
1,3	2,3	3,3	4,3
1,4	2,4	3,4	4,4

MATRIX TRANSPOSE

Reading from memory (coalesced), Stride = 1



Writing to memory (uncoalesced), Stride = 4

MATRIX TRANSPOSE

Read from global mem

1,1	1,2	1,3	1,4	2,1	2,2	2,3	2,4	3,1	3,2	3,3	3,4	4,1	4,2	4,3	4,4
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Write to shared mem

1,1	1,2	1,3	1,4	2,1	2,2	2,3	2,4	3,1	3,2	3,3	3,4	4,1	4,2	4,3	4,4
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Once in shared memory any thread (work item) in a thread block
(work group) can read the data fast

Read data transposed

1,1	2,1	3,1	4,1	1,2	2,2	3,2	4,2	1,3	2,3	3,3	4,3	1,4	2,4	3,4	4,4
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Write to global mem

1,1	2,1	3,1	4,1	1,2	2,2	3,2	4,2	1,3	2,3	3,3	4,3	1,4	2,4	3,4	4,4
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

MATRIX TRANSPOSE

- Caveat: In a real half warp (16 element coalesced load in to shared memory) reading the memory will result in bank conflicts (stride of 16)!
- High performance matrix transpose code will pad the local memory by 1 element to resolve this.

MORE INFORMATION

- MacResearch.org
 - OpenCL - <http://www.macresearch.org/opengl>
 - Amazon Store - <http://astore.amazon.com/macresearch-20>
- Khronos OpenCL - <http://www.khronos.org/opengl>