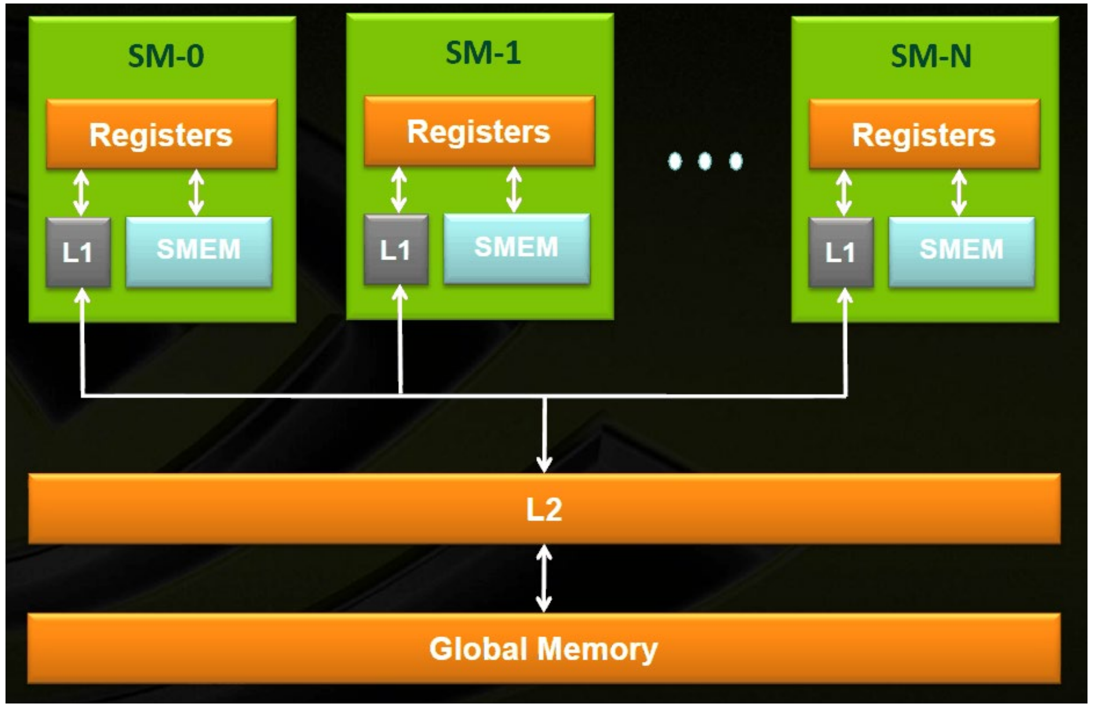
GPU memory hierarchy

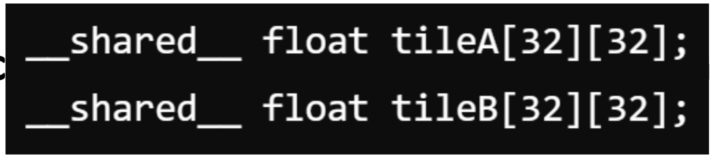
- Global memory
  - Accessible by all threads and the CPU –
- L2 cache
  - Accessible by all threads
- L1 cache
  - Accessible by all threads within an SM.
  - (Hardware Cache)
- Shared memory
  - Accessible by all threads inside the block
  - Programmable (Software cache)
  - Allows for threads to communicate and share data between one another
- Registers
  - Per thread (lifetime is the thread lifetime)



Src:http://xiongz-hft.blogspot.com/2011/11/nvidia-gpu-fermi-memory-hierarchy.html

Why shared memory:

- Shared memory is a **small, fast memory** located on-chip (inside the GPU).
- Lower latency (25 cycles vs 33 to 300 cycles).
- **Data reuse:** If multiple threads need the same data, loading it once into shared memory minimizes **global memory traffic**.
- It is shared among all threads within the same **same block**, allowing them to **exc** slow global memory.



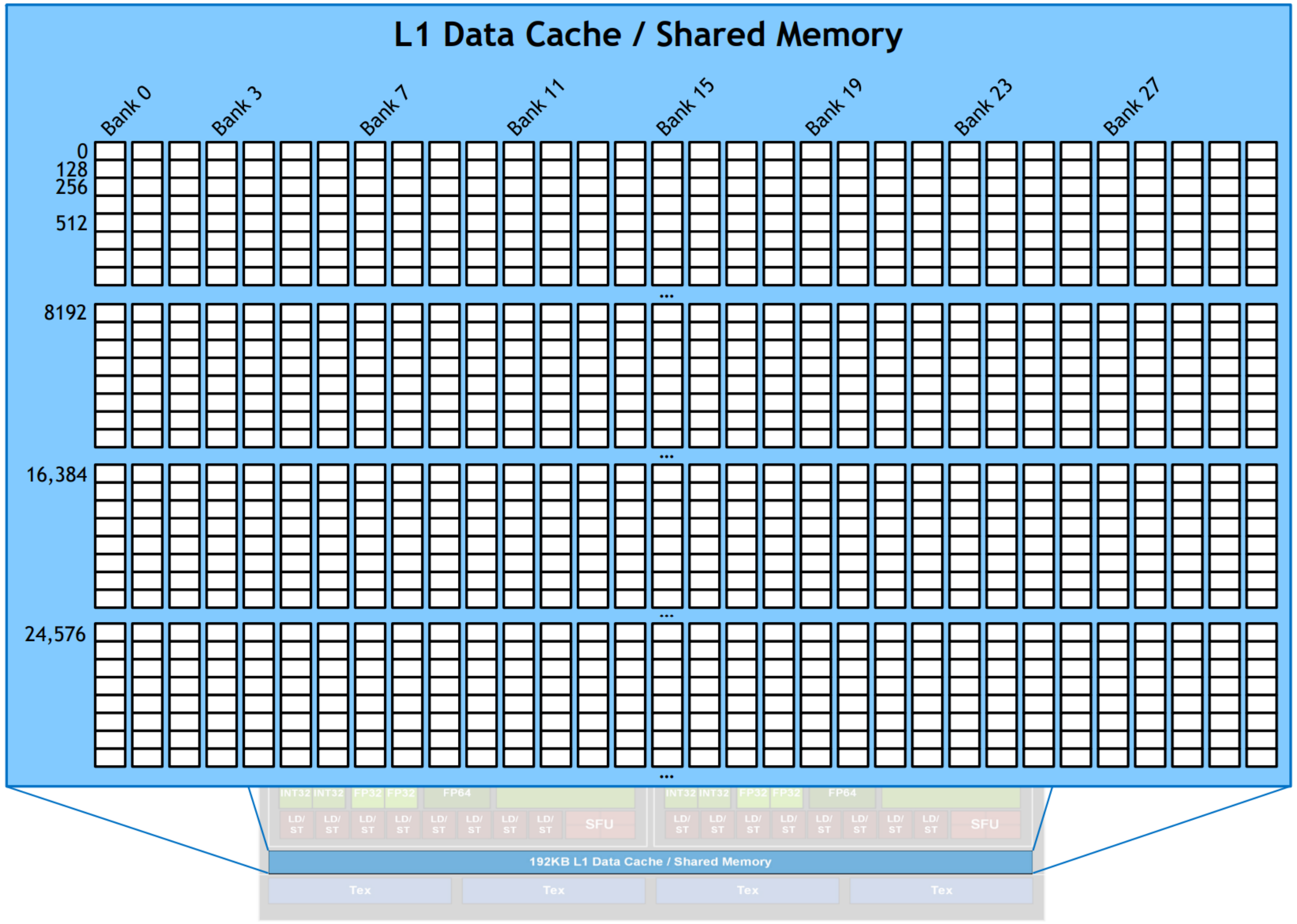
```
__shared__ float tileA[32][32];
__shared__ float tileB[32][32];
```

- Managed by software code.
- Thus, reduce the memory bottlenecks.

The shared memory size in the ampere architecture:

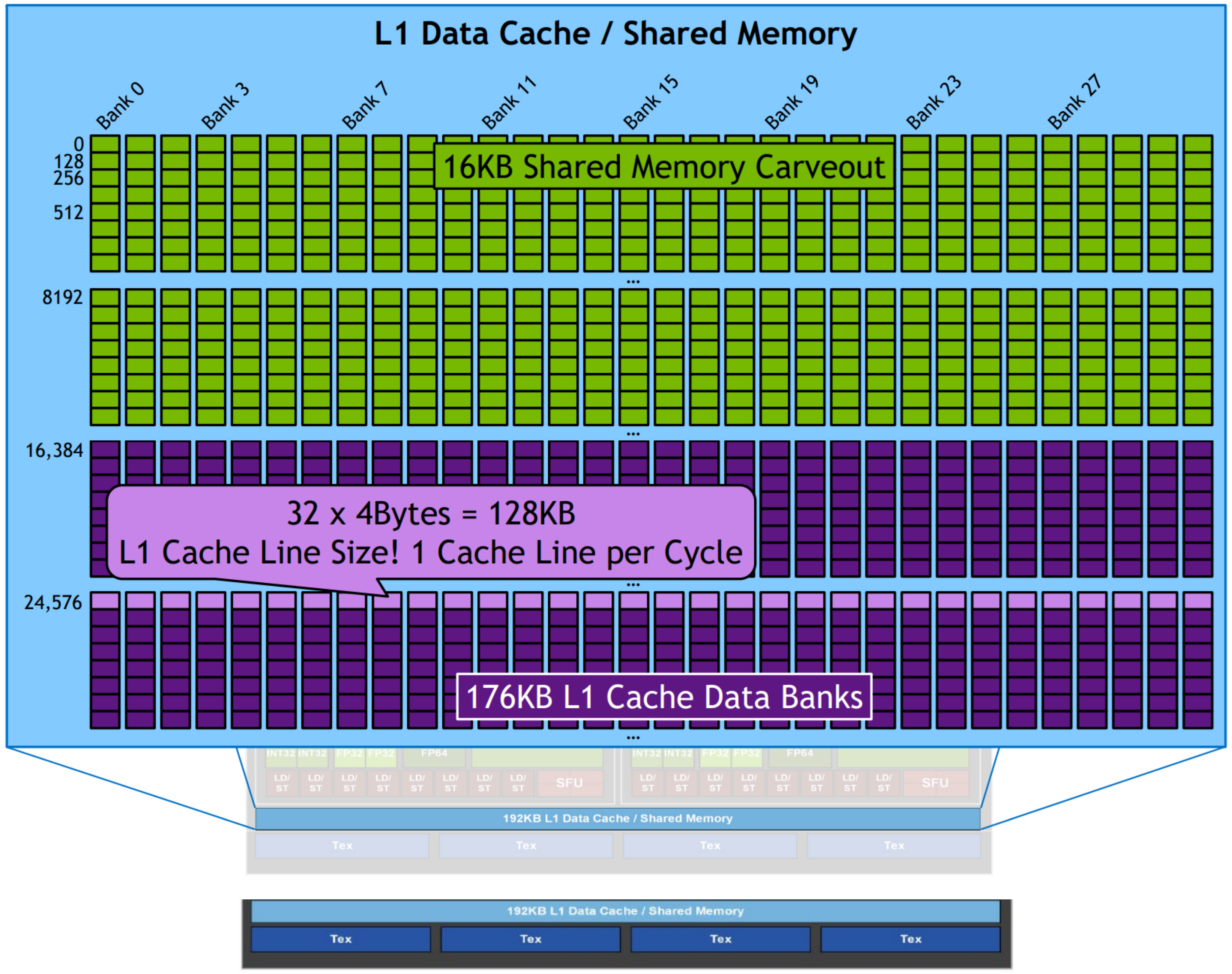
- Shared memory and L1 cache are built with 1 physical unit with a fixed total size.
- This means their sizes are configurable.
- Example, in A100 GPU their physical unit size is 192KB.
- Shared memory can be configured with specific sizes (0,8,16,32,64,100,132,164) KB.
- For example, if we choose 100 KB for the shared memory, then the l1 size will be 92 KB.
- The compiler manipulates this according to the application usage of shared memory.

The shared memory architecture:



- Shared memory is divided into cache lines with a size of 128 Byte.
- Each cache line consists of 32 banks.
- The bank size is 4 bytes.
- 4Byte data access per bank per cycle.
- This means, once cache line can be read per cycle.

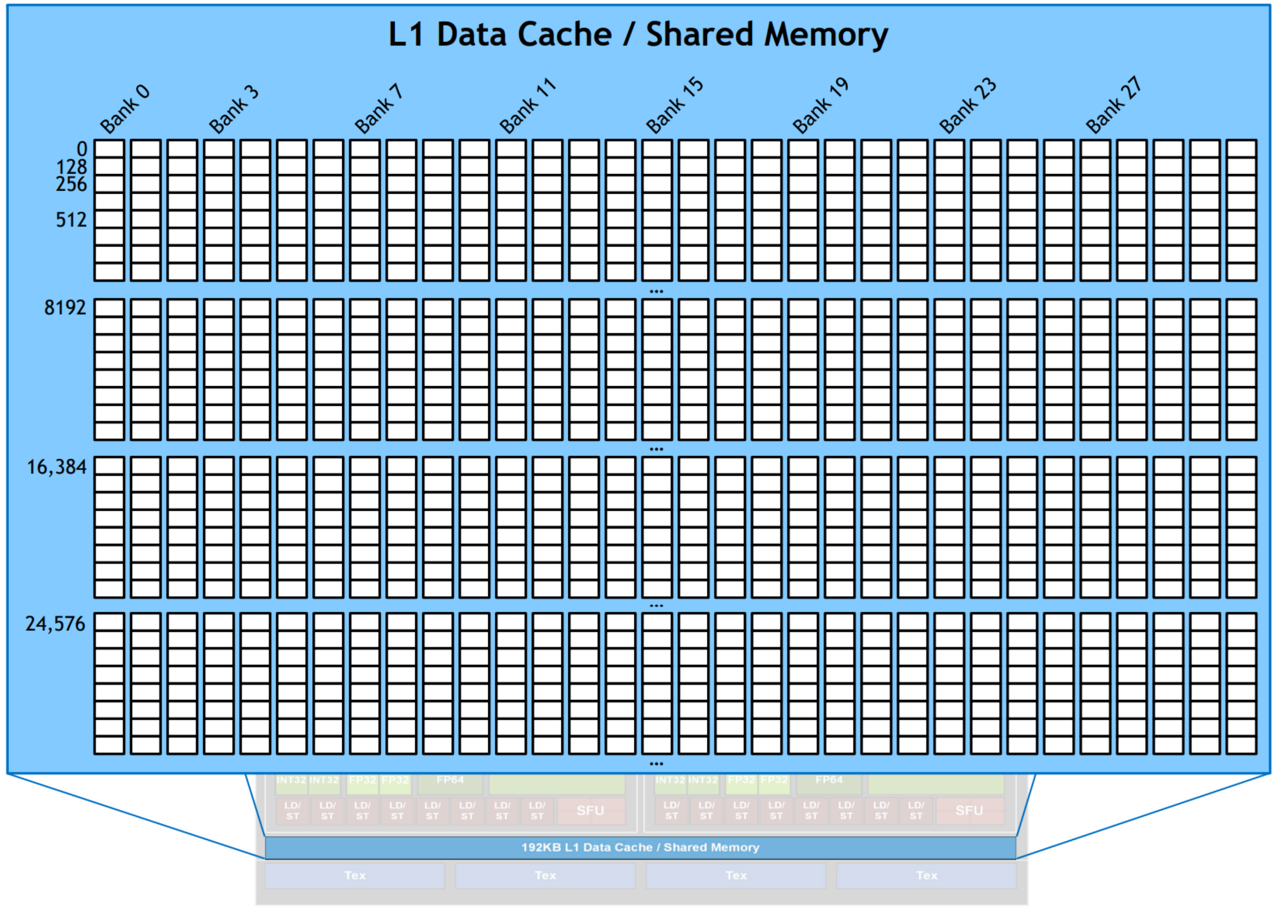
The shared memory architecture:


L1 Data Cache / Shared Memory
Bank 0
Bank 3
Bank 7
Bank 11
Bank 15
Bank 19
Bank 23
Bank 27
0
128
256
512
8192
16,384
24,576
16KB Shared Memory Carveout
32 x 4Bytes = 128KB
L1 Cache Line Size! 1 Cache Line per Cycle
176KB L1 Cache Data Banks
192KB L1 Data Cache / Shared Memory
Tex
SFU

- The figure shows physical unit itself.

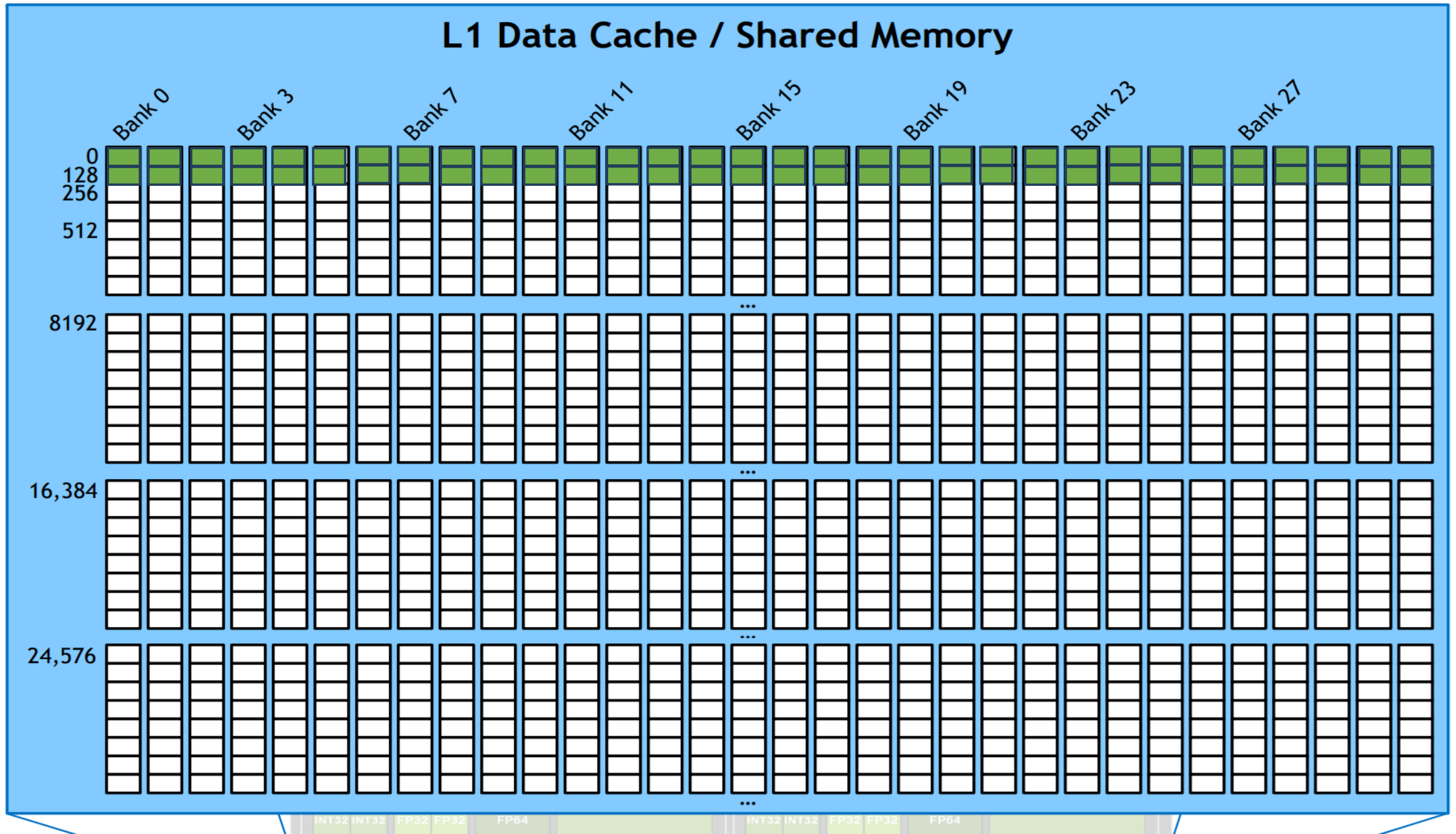
The shared memory architecture:

- Try to minimize the number of conflicts as much as you can.
- Use the shared memory for re-using data as much as you can.
- We will apply the shared memory optimizations on the vector reduction application.

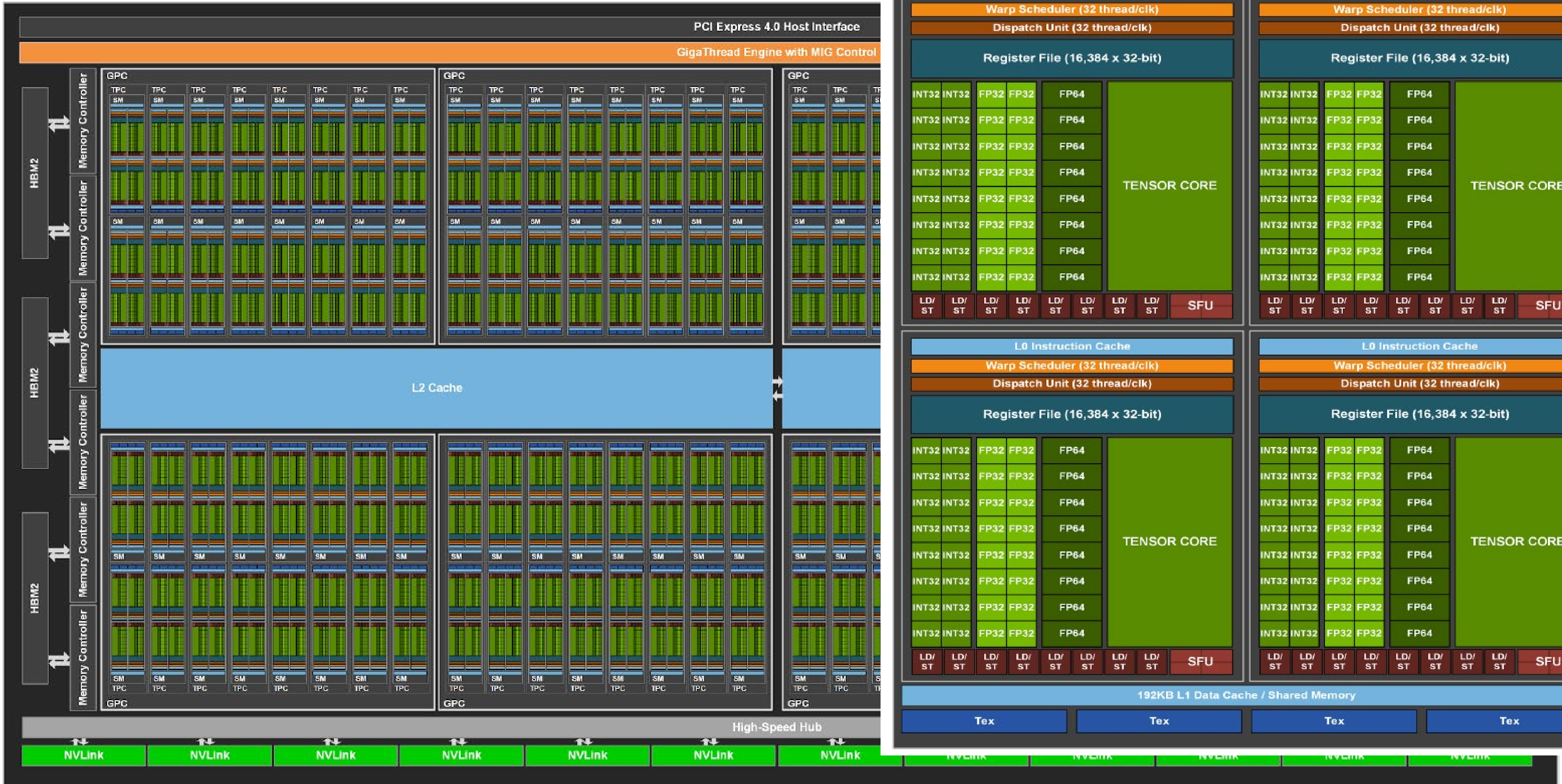


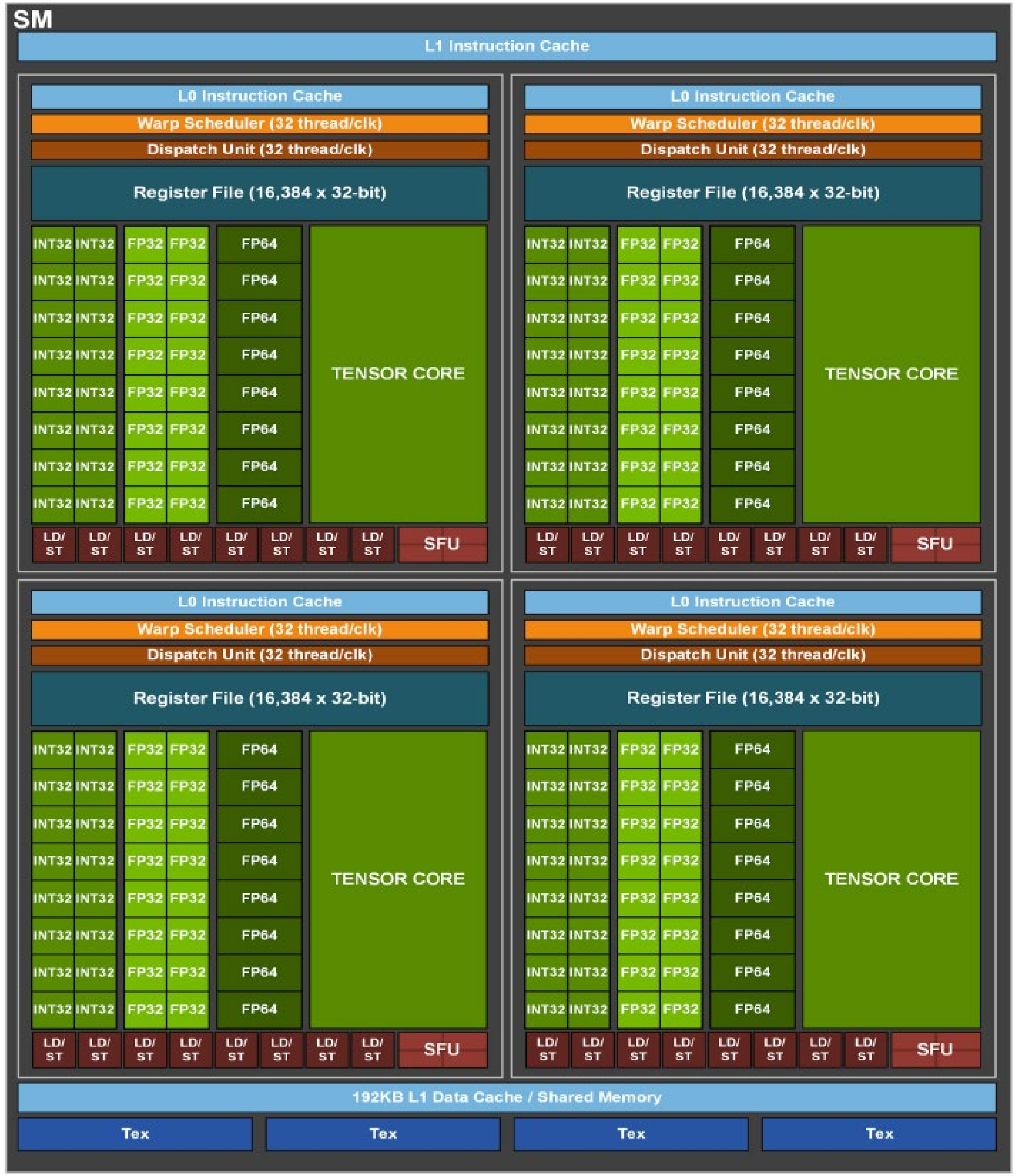
The shared memory architecture:

- Ex: how many conflicts if a warp is going to read a 32 double precision elements ?!
  - Double precision element size is 8 bytes.
  - Total size required is 8*32 threads = 256 Bytes.
  - We must have at least one conflict (because we can’t read more than 128 Bytes per cycle).
  - If the elements are stored contiguously, we will have two accesses per column.
  - This provides at least 1 conflict.



GPU hardware in general (


PCI Express 4.0 Host Interface
GigaThread Engine with MIG Control
GPC
TPC
SM
Memory Controller
HBM2
L2 Cache
High-Speed Hub
NVLink


SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
INT32
FP32
FP64
TENSOR CORE
LD/ ST
SFU
192KB L1 Data Cache / Shared Memory
Tex

Important example

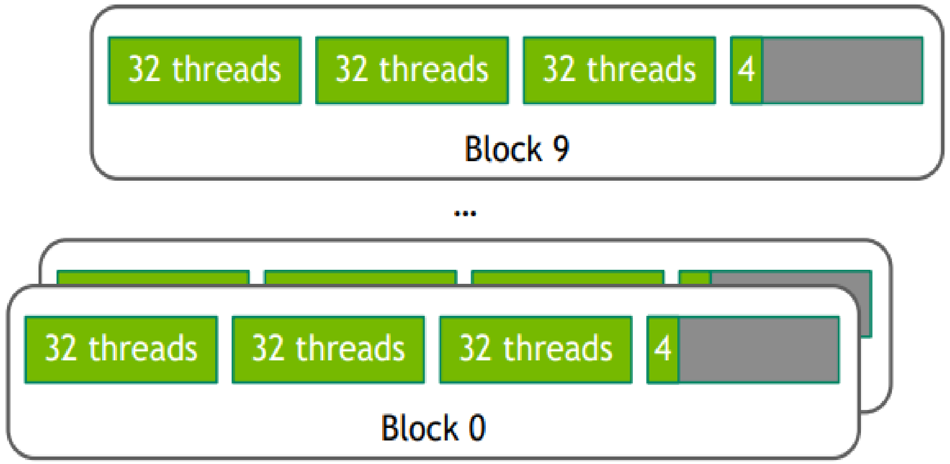
- In previous apps:
  - The number of elements = a Multiple of 32 value .
  - For example, 1024 elements per vector. (or 2048)
  - Minimum size of any cuda application is 1 warp (32 threads).
  - We can use 32 threads but assign an operation to only one.
  - But we can’t use less than 32.
  - So, we have to use 32 or multiple of 32.
  - That make it easy to determine the number of warps (block size).

Important example

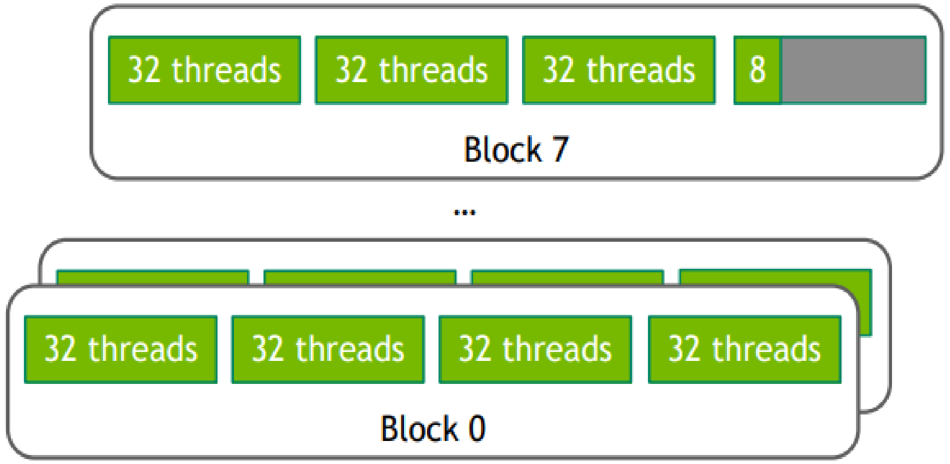
Question:

- What if the total elements are not equal to a multiple of 2 value ?

- Let's say we want to add two vectors of size N = 1000.
  - **Scenario #1:** 1-D grid of 10 1-D blocks of size 100.
  - **Scenario #2:** 1-D grid of 8 1-D blocks of size 128.
- Which option is better in terms of thread resource utilization?



Scenario #1:
3 full warps and 1 warp with 4 active threads per block
Average thread utilization = 78.125%



Scenario #2:
4 full warps per block, except last block
Average thread utilization = 97.656%