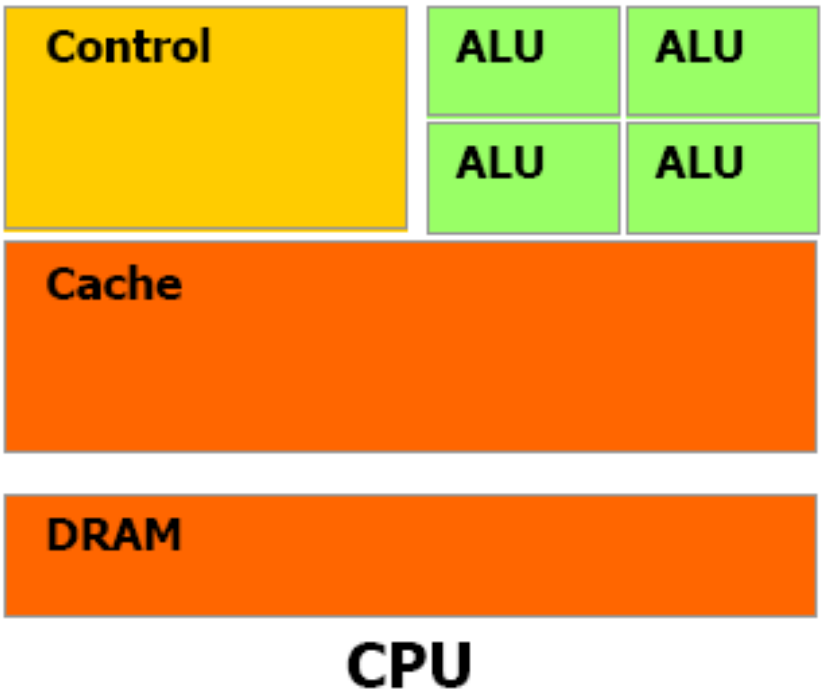
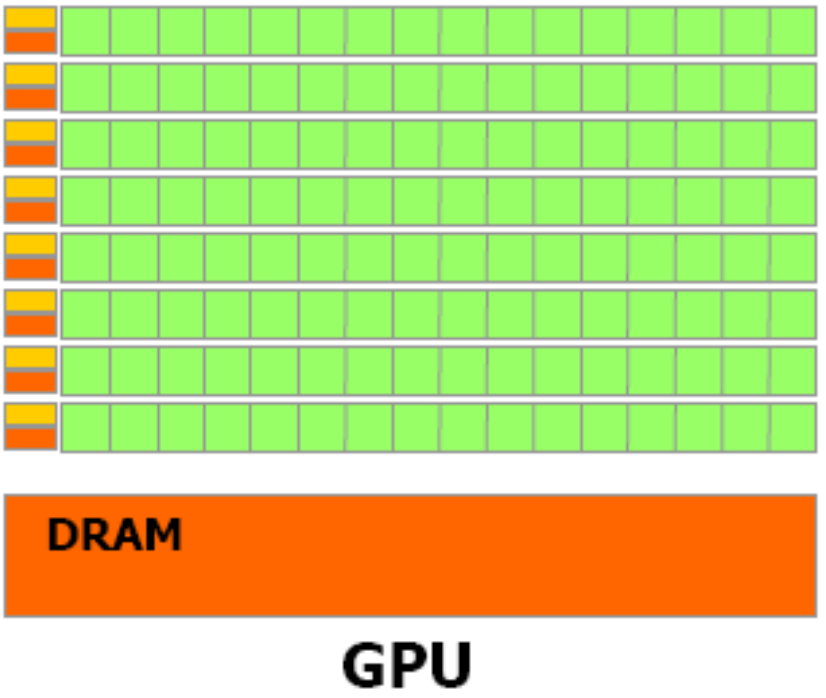
Parallel Computing with CUDA

Emil Dotchevski
emil@revergestudios.com

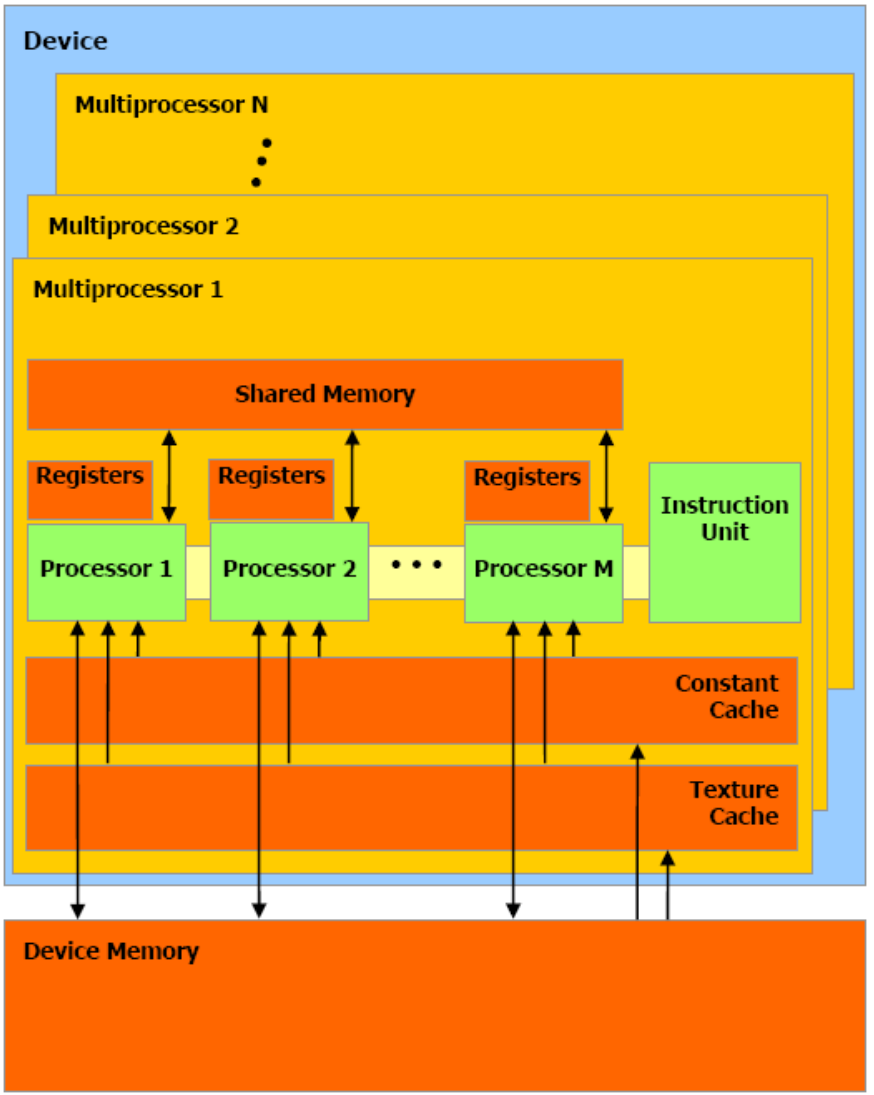
CPU and GPU architecture





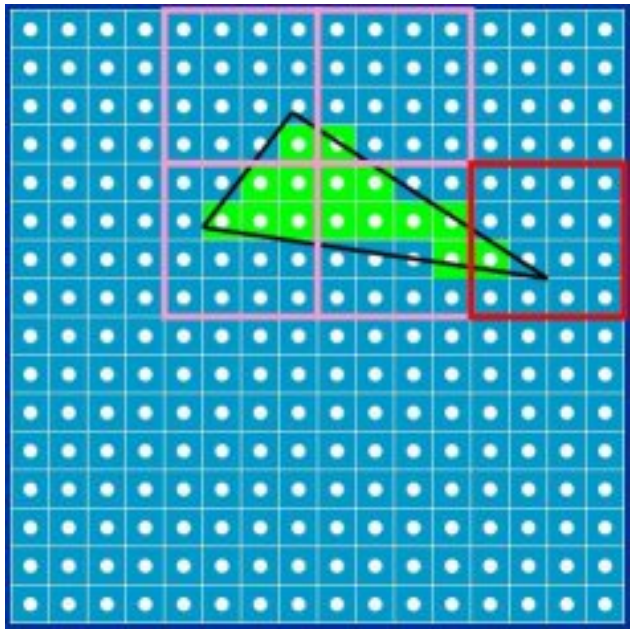
NVIDIA device hardware model

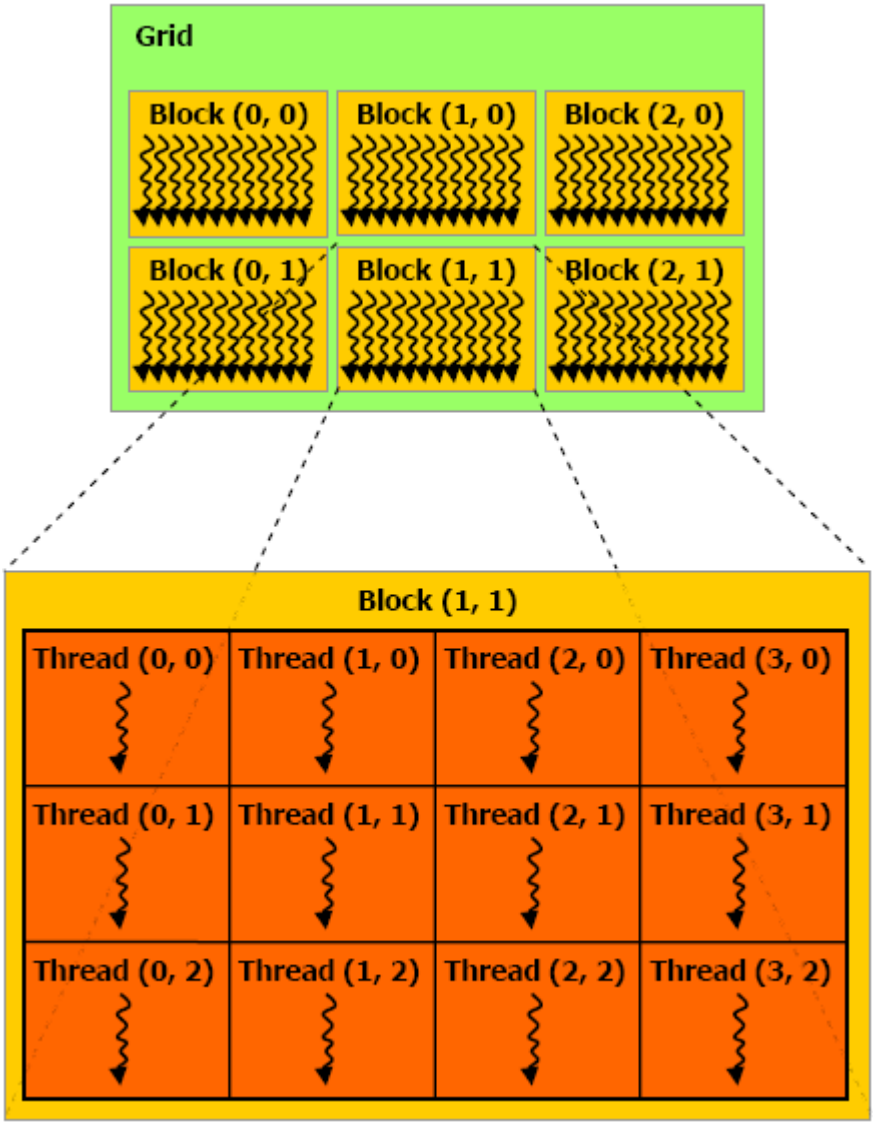
- Several multiprocessors:
  - GeForce 9800: 16
  - Tesla C1060: 4x30
- Each multiprocessor:
  - is a SIMD unit (lockstep)
  - has its own texture cache and local shared memory
  - shares global memory access with all other multiprocessors



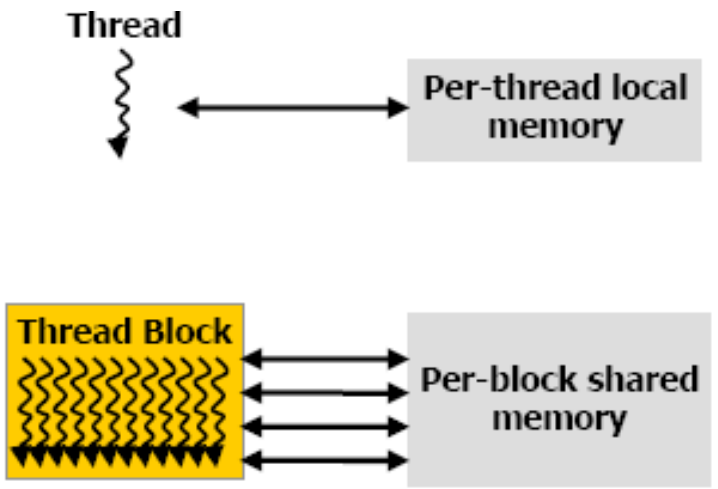
Thread hierarchy

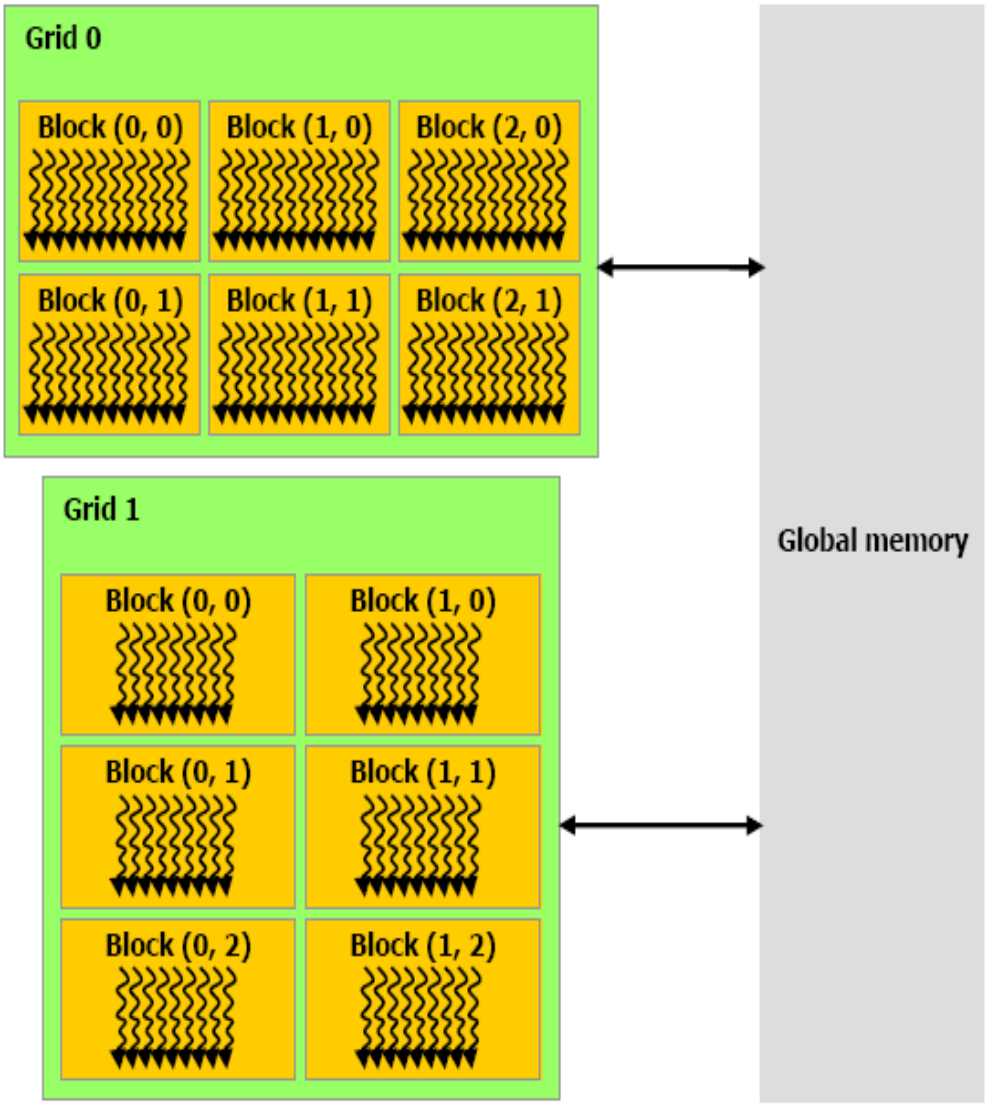
- You launch a 2D thread grid to do work
- The grid is made up of 2D thread blocks
- A 2D thread block is made up of individual threads that execute in lockstep.





Memory architecture





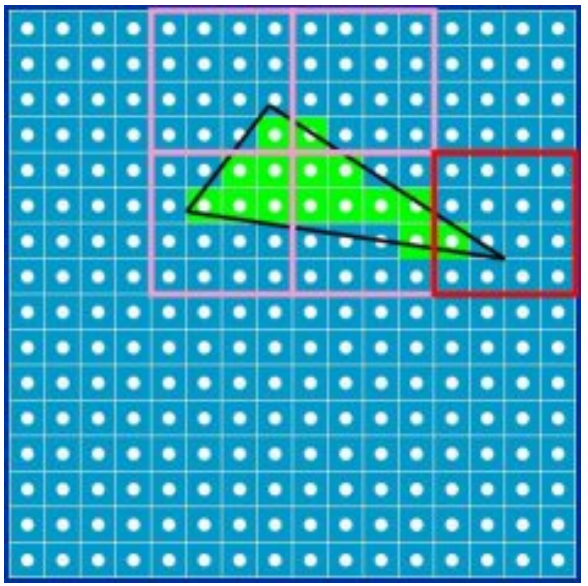
Memory	Where	Cache	R/W	Scope
Register	on-chip	N/A	R/W	thread
Local	off-chip	No	R/W	thread
Shared	on-chip	N/A	R/W	block
Global	off-chip	No	R/W	app
Constant	off-chip	Yes	R	app
Texture	off-chip	Yes	R	app

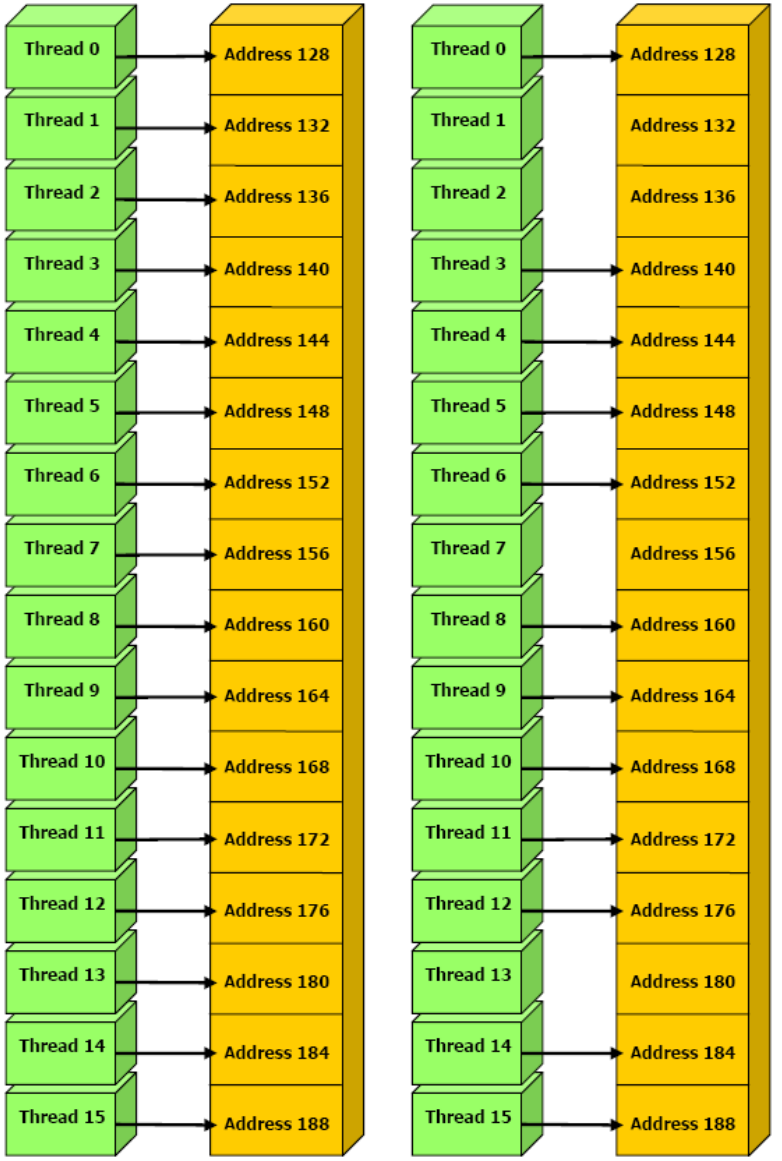
Global memory access

- Not cached
- All threads within a thread block should synchronize before each access
- This ensures that (subject to additional requirements) multiple threads share the same memory transaction
- If not, the penalty is huge

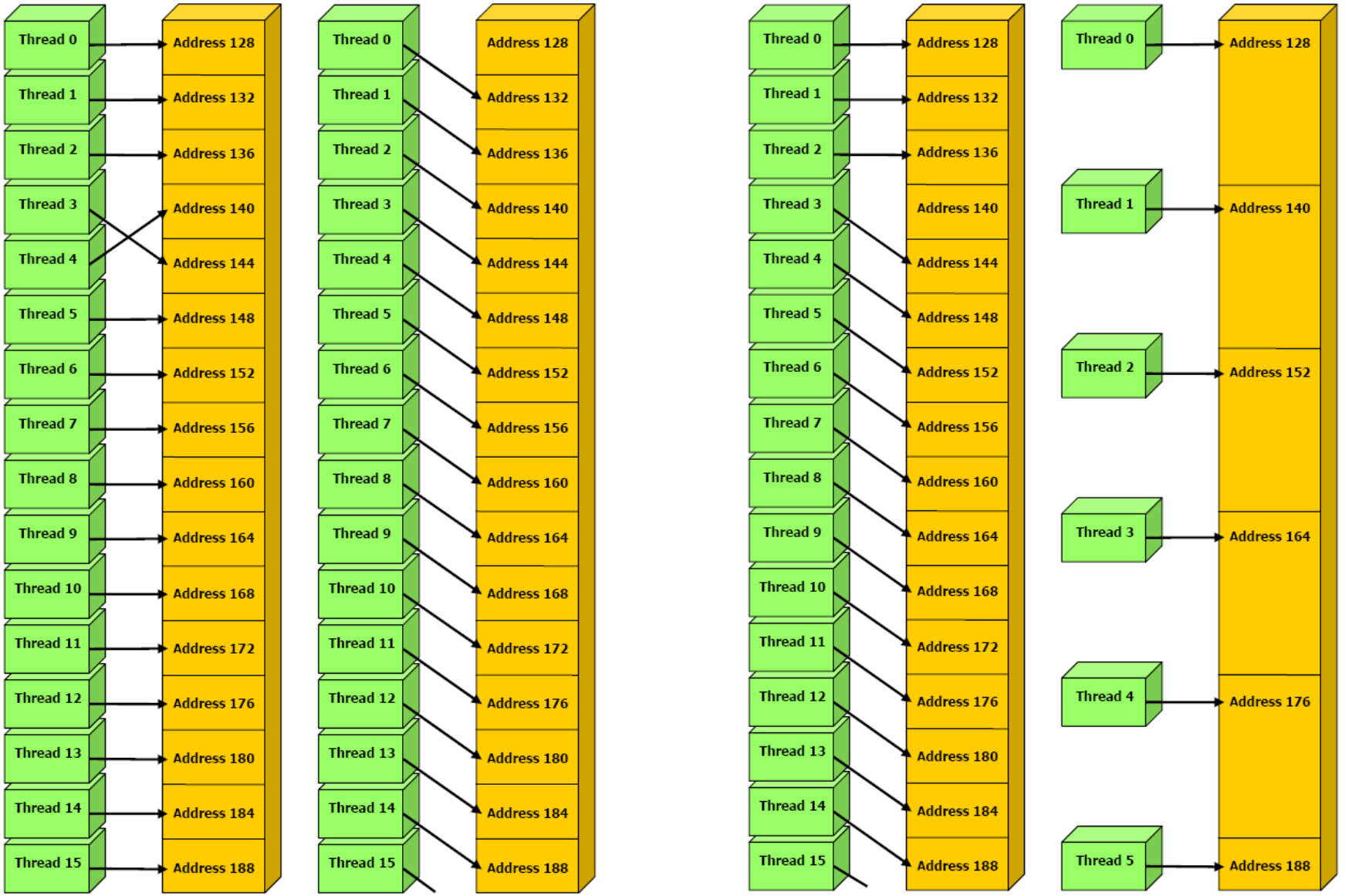
Coalesced global memory access

- Left: threads read/write their own “pixel”
- Right: same thing, except some threads choose not to read or write



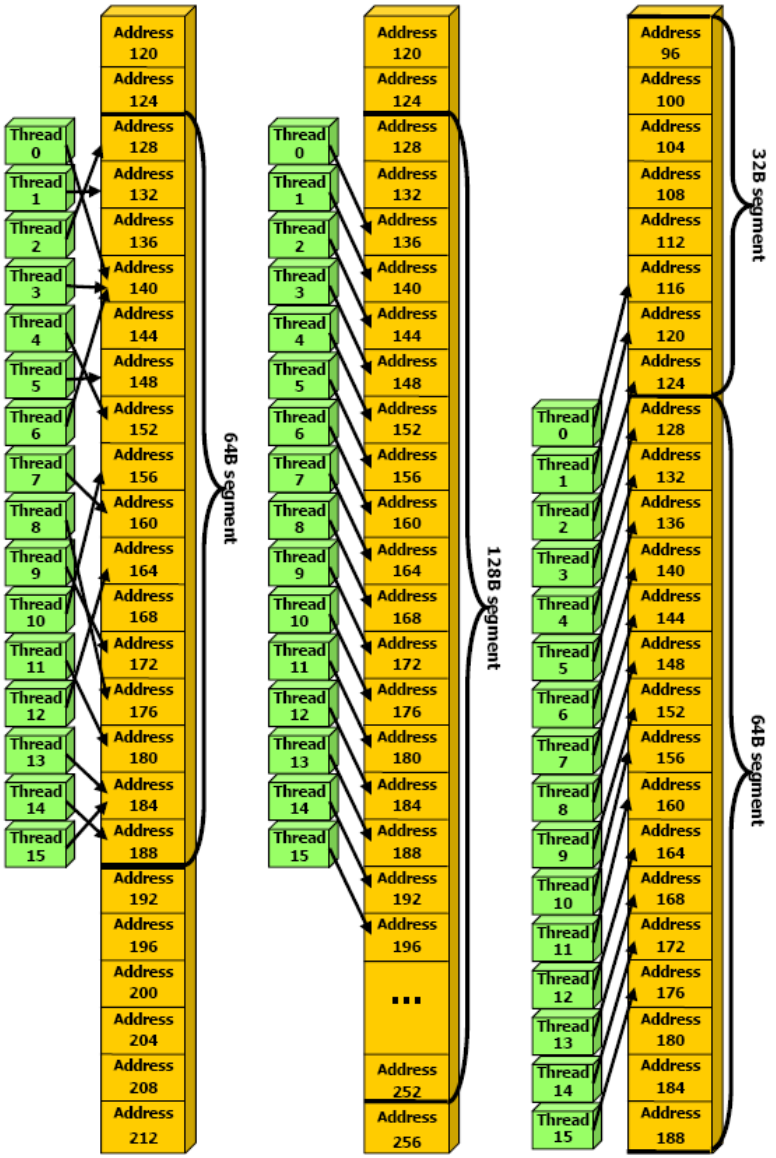


Non-coalesced memory access



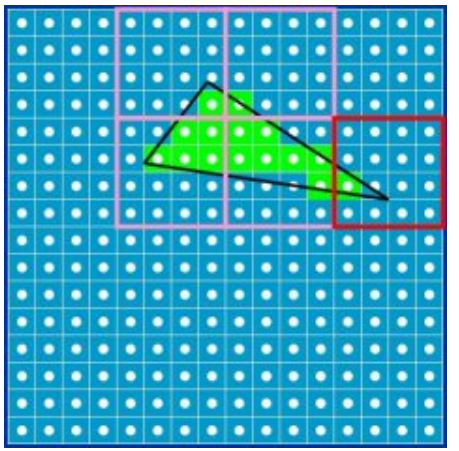
Compute capability 1.2 and up

- Left: random access within a 64-byte segment, one transaction
- Center: misaligned access, one transaction
- Right: misaligned access, two transactions
- Synchronization is still required



Global memory reads using the texture cache

- Textures are 1D, 2D or 3D data arrays
- Texture cache is optimized for spacial locality
- Not subject to access pattern limitations, much better performance even on cache misses
- 8-bit and 16-bit integer data can be optionally converted to 32-bit float in the range [0,1] or [-1,1] for free
- Bilinear filtering and several address modes available for free

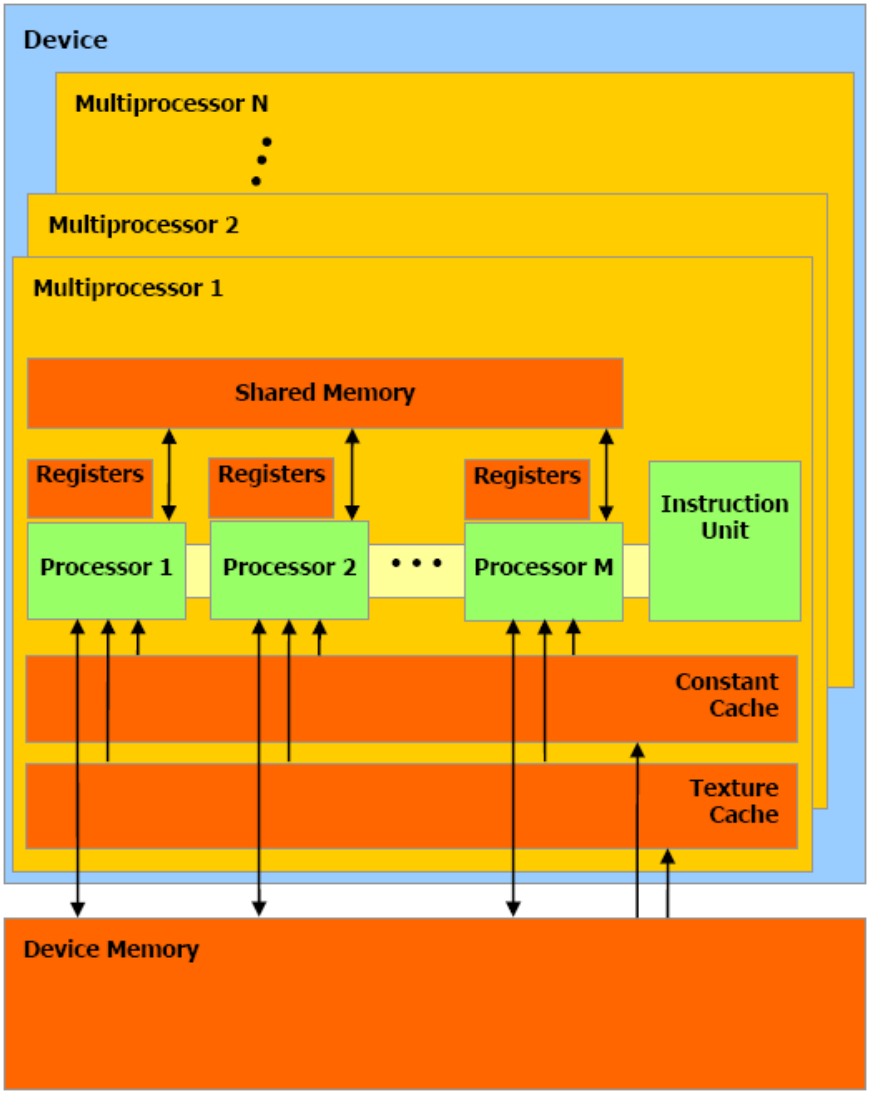


Local memory access

- Private memory for each thread
- The same latency as global memory
- Always coalesced
- No synchronization necessary
- Up to the compiler to place objects in local memory vs. registers

Register/shared memory

- Both use the same memory pool local to each multiprocessor
- Amount of memory used per thread affects the maximum number of blocks that can be active on a multiprocessor
- Not a lot of memory (16K or so)
- Programmer can tinker with max registers used

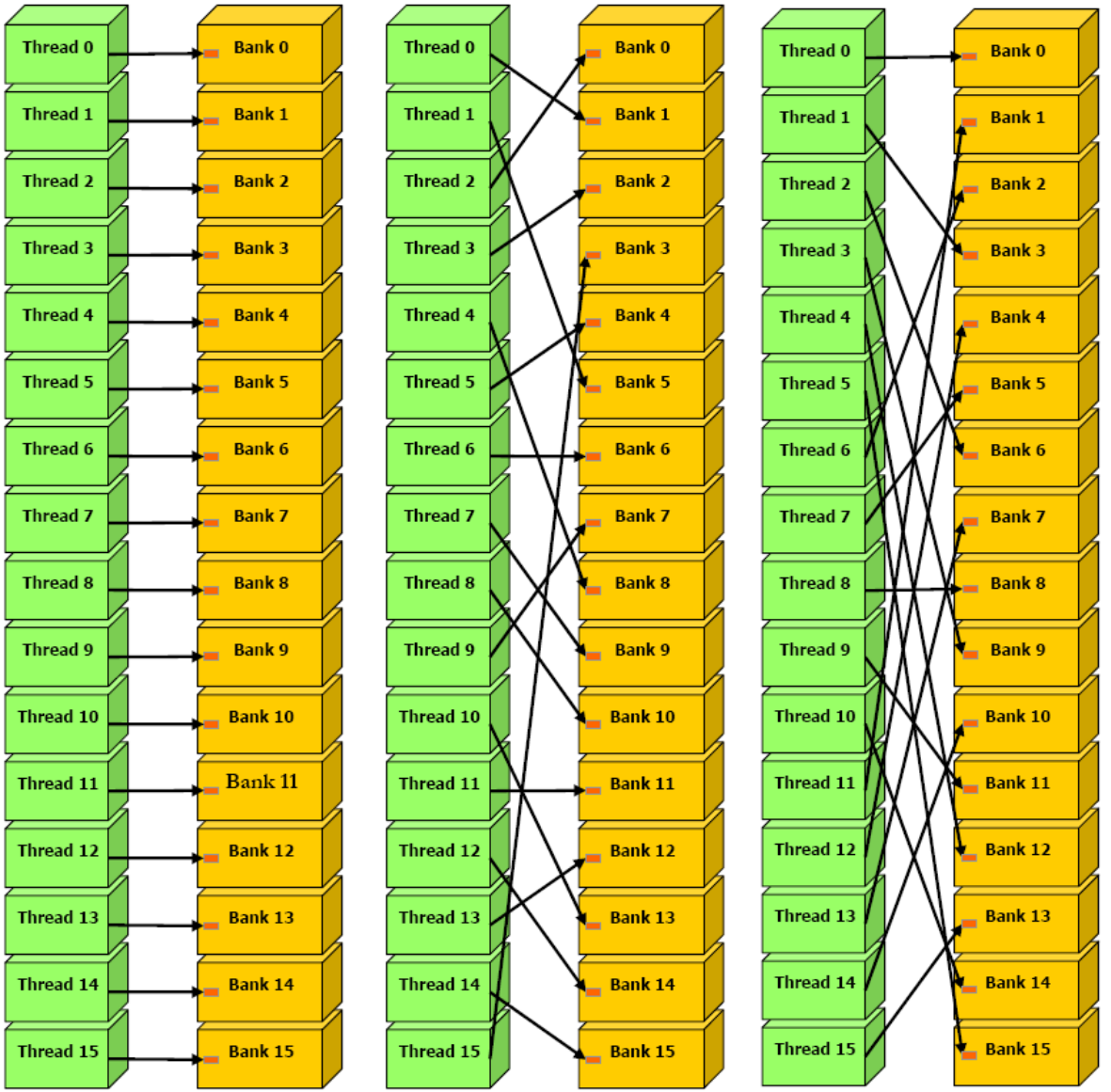


Shared memory access

- Shared memory is actually “manual” cache
- As fast as registers, subject to avoidance of memory bank conflicts
- Can be used by threads within the same thread block to collaborate on a common task
- Eats up register space

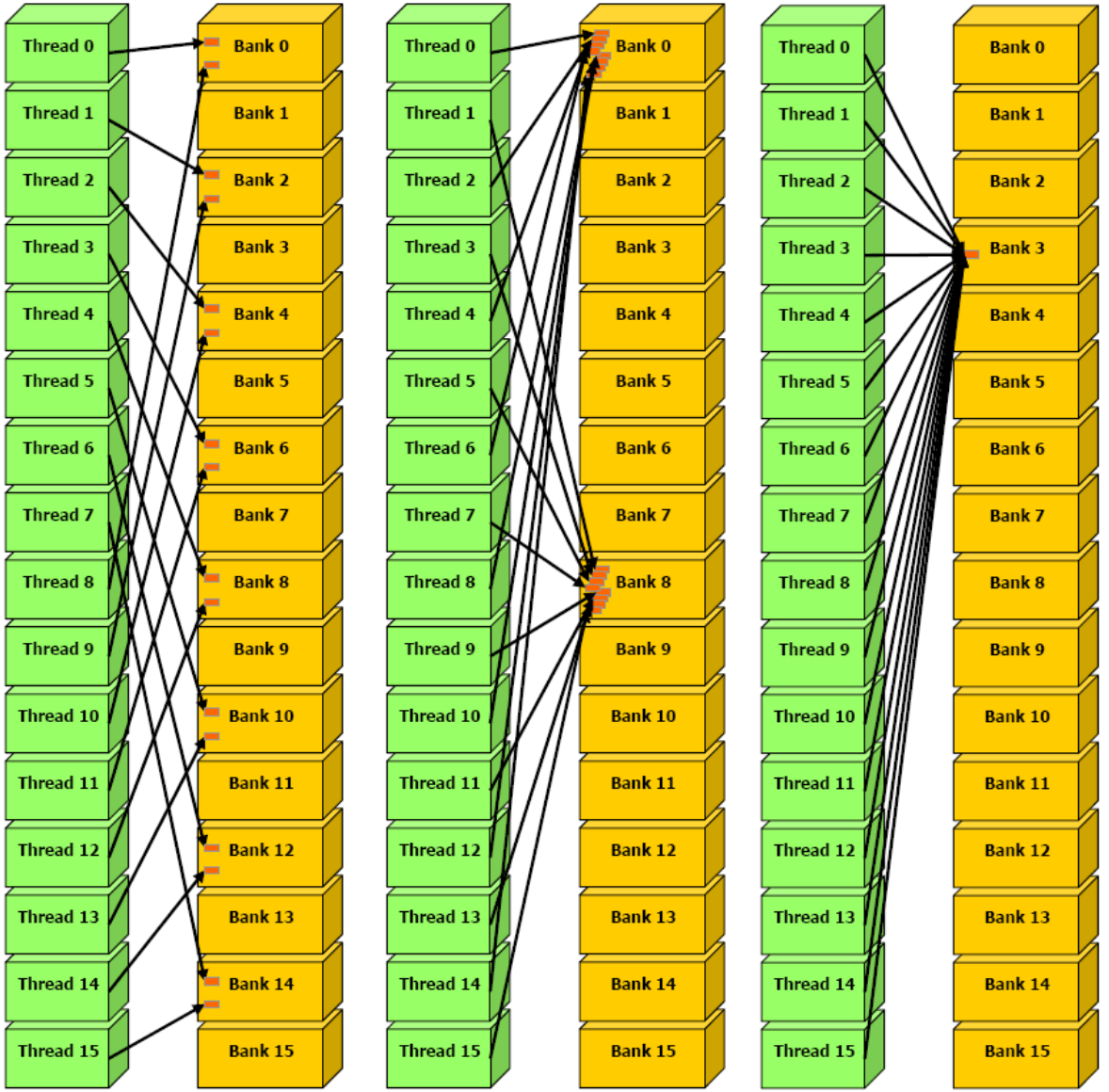
Optimal shared memory access

- Left: linear with stride one 32-bit word
- Center: random permutation
- Right: linear with stride three 32-bit words



Shared memory access

- Left: linear with stride two 32-bit words, 2-way bank conflicts
- Center: linear with stride eight 32-bit words, 8-way conflicts
- Right: conflict-free (broadcast)



Latency/occupancy

- Global memory access has great latency (400-600 cycles)
- However, when a thread hits a memory read or write instruction, the scheduler is able to run arithmetic instructions from other threads
- Therefore, if you have enough stuff to compute the memory latency can be ignored (almost)

Interfacing

- You can copy data from host to device memory. Use to feed data to CUDA kernels.
- You can copy data from device to host memory. Use to get results from CUDA kernels.
- You can map device memory to the host address space. Consumes physical OS memory.
- You can map OpenGL or Direct3D objects for access by CUDA kernels