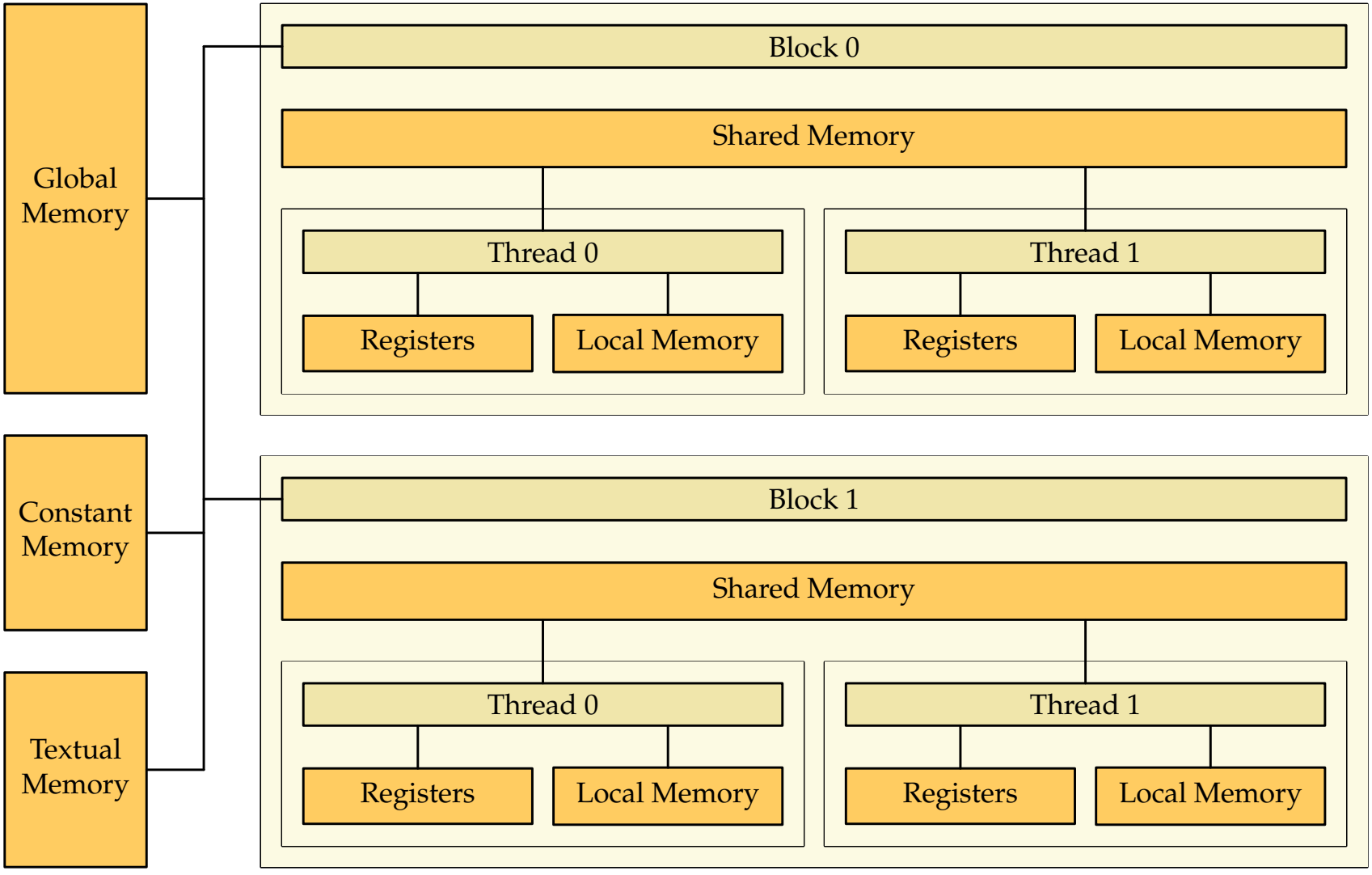
High Performance Computing I
WS 2019/2020
Session #29
Virtual vs. physical GPU architecture

Andreas F. Borchert and Michael C. Lehn
Ulm University

February 7, 2020

Storage areas of a GPU

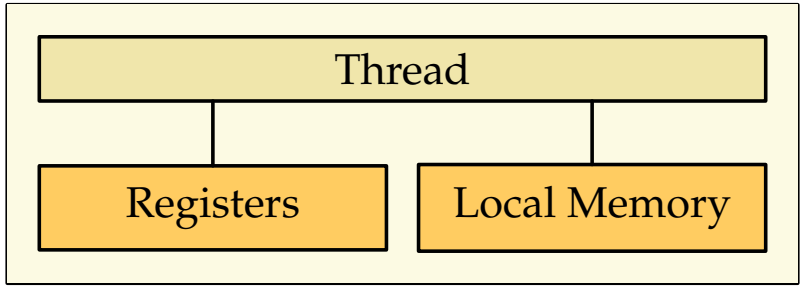


Summary of storage usage

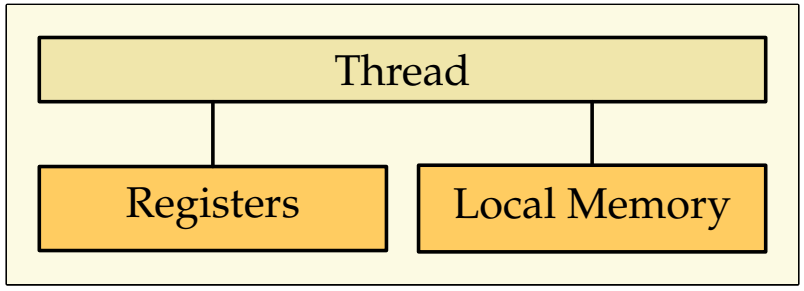

```
livingstone$ make
nvcc -o simpson -I/home/numerik/pub/hpc/ws19/session29 --ptxas-
options=-v simpson.cu
ptxas info    : 0 bytes gmem
ptxas info    : Compiling entry function '_Z7simpsonddPd' for 'sm_30'
ptxas info    : Function properties for _Z7simpsonddPd
    0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
ptxas info    : Used 26 registers, 344 bytes cmem[0], 40 bytes cmem[2]
livingstone$
```

- The option "-v" asks *ptxas* to summarize the storage use of the individual storage areas for all kernel functions.
- *gmem* represents global memory, *cmem* constant memory which is possibly split into different segments in dependence of the GPU architecture.
- Local memory is required for the stack frame and for saving registers (*spill stores*). But *ptxas* is just able to count spill stores and spill loads in dependence of the static code.

Registers

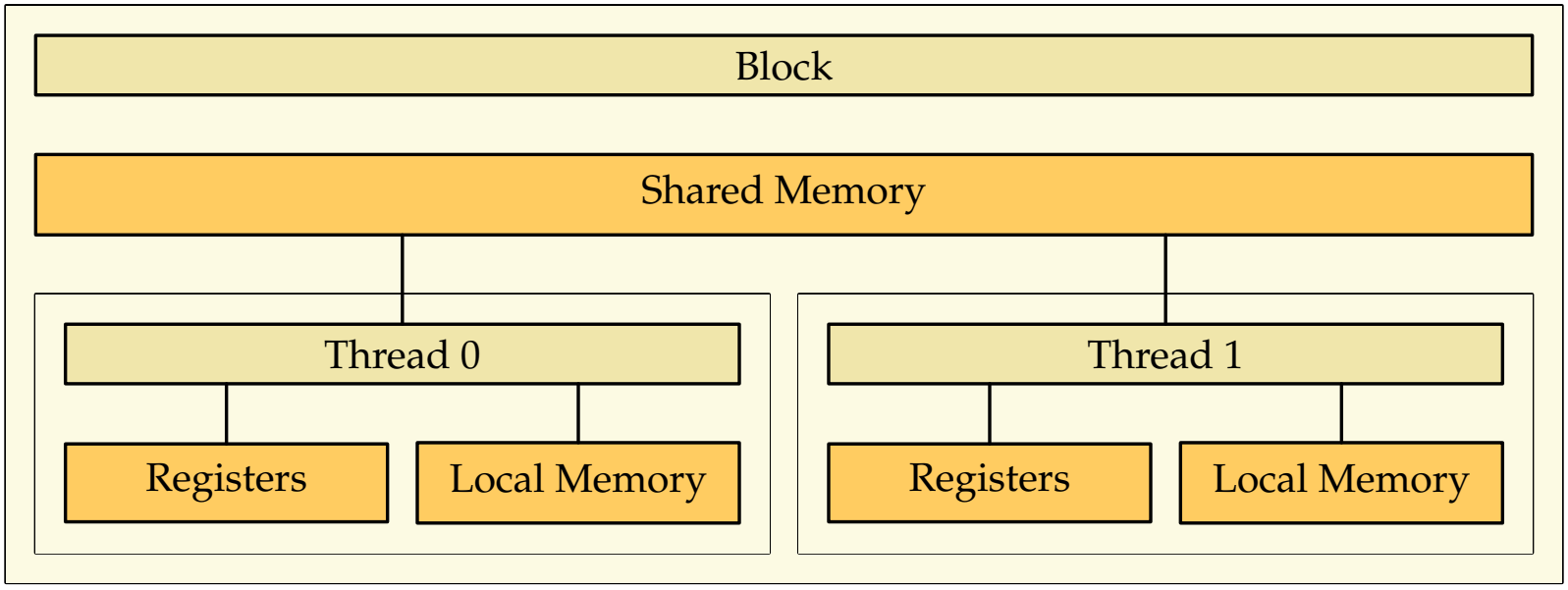


- 32 bit registers can be used for whole or floating point numbers.
- Local variables are stored as far as possible in registers.
- A kernel with a very high demand of registers has possibly a more limited number of threads per block.
- The GPU on Livingstone offers 65536 registers per block. The limit of 1024 threads per block allows 64 registers per thread. If more registers are required, less threads are available per block.



- Local memory is stored in global GPU memory.
- However, caching is optimized for local memory as no cache coherence has to be established with concurrent threads.
- Global memory is significantly slower than access of registers or shared variables.
- Local memory is used for the stack if a kernel function is running out of registers or for local arrays where indices are computed at runtime.

Shared memory



- Shared memory provides a speed similar to the L1 cache.
- Just registers are faster than shared memory.
- This is the fastest option for arrays where indices are computed at runtime.
- The capacity of shared memory is very limited. Livingstone offers 48 KiB per block and 96 KiB per multiprocessor (which is possibly executing multiple blocks in parallel).

- Shared memory is cyclically partitioned into banks.
- The first word (32 bits) in shared memory belongs to the first bank, the second word to the second bank etc. Livingstone has 32 banks. Hence the 33rd bank belongs to the first bank etc.
- A memory access by a warp where each thread accesses a different bank can be executed concurrently. Multiple accesses of the same bank need to be serialized.
- Broadcasts and multicasts are possible in dependence of the GPU architecture.

- Global memory is shared among all threads on the device and also by the host (using *cudaMemcpy*).
- Paging is not supported, i.e. we do not have virtually more memory than physical memory.
- On Livingstone we have 1,96 GiB.
- Memory accesses go through L1 and L2 where (on our GPU) cache coherence is established on L2. This has the consequence that write operators must immediately be propagated to L2.
- On Livingstone, L2 provides 512 KiB.
- Global variables can be declared using the storage class specifier **__global__** or by asking for dynamic memory.

Performance of global memory access is best if following requirements are met:

- Access should be word-wise (32 bit) or multiples of words (like **double**).
- The memory locations by a warp shall be consecutively ordered.
- The first word (accessed by the first thread) shall be properly aligned:

access size	alignment
32 Bit	64 Byte
64 Bit	128 Byte
128 Bit	256 Byte

In case of the Pascal microarchitecture (our GPU on Livingstone) all memory accesses are done through L2. Special compilation flags are required for accesses through L1.

- Cache lines of L1 and L2 have 128 bytes with an alignment of 128 bytes.
- A cache line consists of four segments of 32 bytes each.
- In case of a miss, a segment is filled but not necessarily the whole cache line.
- If the required data are already in L1, a cache line can be transfered within one transaction.
- If the required data are in L2, one segment (i.e. 32 bytes) can be transfered within each transaction.
- Consecutive memory locations in the order of threads of a warp are no longer required. However, to achieve best performance, all memory locations of a warp should fit into a cache line.

- Constant memory is a segment of global GPU memory with an optimized cache access as cache coherence is not required.
- (Updates of constant memory are theoretically possible but their effect is undefined.)
- Compilers use constant memory for the parameters of kernel functions. It is also possible to use the storage class specifier **__constant__**.
- Livingstone provides 64 KiB constant memory.

tracer.cu

```
__constant__ char sphere_storage[sizeof(Sphere)*SPHERES];
```

- **__constant__** can be used as storage class specifier for variables in constant memory.
- Data types with non-trivial constructors or destructors are not supported in constant memory.
- *cudaMemcpyToSymbol* allows the host to initialize such a variable.

tracer.cu

```
Sphere host_spheres[SPHERES];
// fill host_spheres...
// copy spheres to constant memory
CHECK_CUDA(cudaMemcpyToSymbol, sphere_storage,
   host_spheres, sizeof(host_spheres));
```