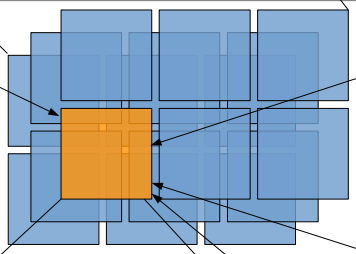
CUDA GPU logical execution model.

Version 1.0

<<< **Kernel** (execution on single GPU, **common code** for all blocks) >>>

Kernel divided into 1-D, 2-D or 3-D **grid** of **blocks**. Identical code, so only variability comes from thread and block indices.



Block executes on **single** SM, divided deterministically into warps of many (now 32) threads. Warps may execute in any order.

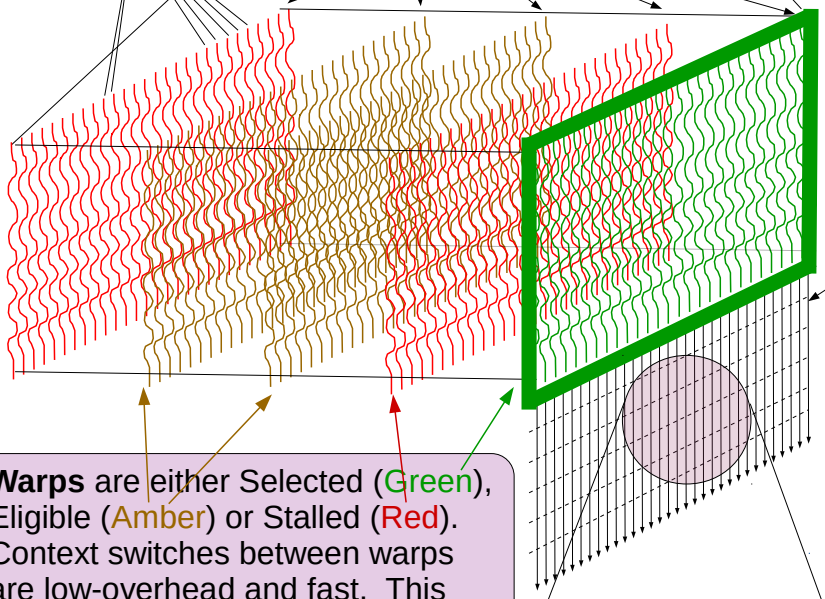
Blocks are assumed to be **independent** to enable parallelism. If required, they must enforce inter-block synchronisation themselves (e.g. via global memory).

Block requirements are known at compile time. Shares resources of SM. Shared memory available **within** block, since it is on single SM. However SM resources are committed for block lifetime.

Block can synchronise itself via __syncthreads command.

Threads

Warps

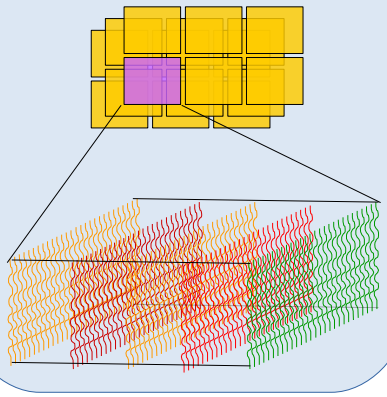


Warps are inherently synchronised. Selected warp operates in SIMT lock-step until it stalls.

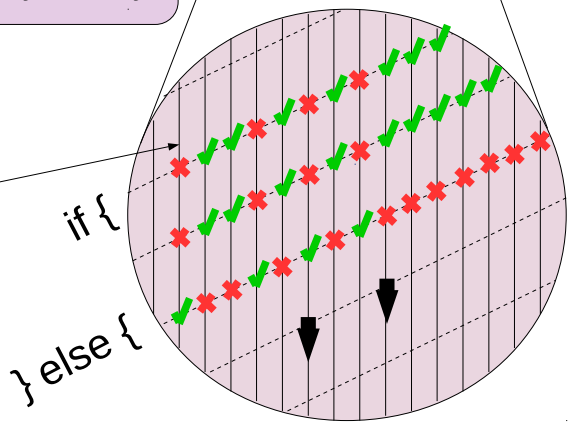
Clock

Warps are either Selected (Green), Eligible (Amber) or Stalled (Red). Context switches between warps are low-overhead and fast. This switching can hide memory latency.

Another kernel can execute on same GPU and blocks can use same SMs (if resources are available) or execute concurrently on different SM.



Warp parallelism is reduced if not all threads can follow same path. Unused threads are de-selected for that instruction.



Ideally all threads in warp execute same instruction on different data (Single Instruction Multiple Data, SIMD) but machine is actually **SIMT** (Single Instruction Multiple Thread).

Key execution unit is **Streaming Multiprocessor (SM)**, containing Streaming Processors (SP), schedulers, arithmetic units etc. In this *logical* model this is only assumed to isolate blocks. In a physical model, the resource constraints of the SM will dictate suitable optimisations.