Parallelism:

→ Instruction-level

→ Thread-level (separate registers, state)

→ Data-level.

(SIMD)

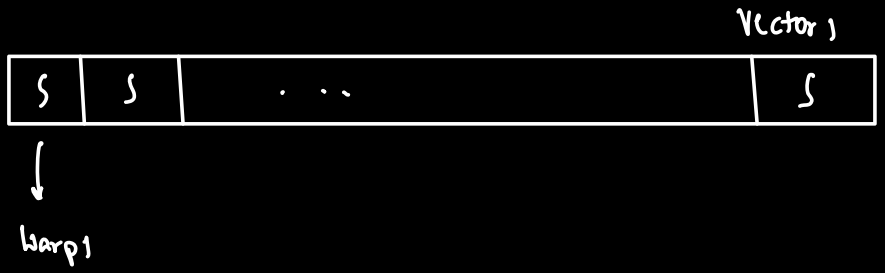
We can have vector registers and vector operations (like vecmul).

→ Process one vector at a time.

SIMT-GPU Core:

Assume vector is a collection of threads.

The threads that together form a vector is WARPs.

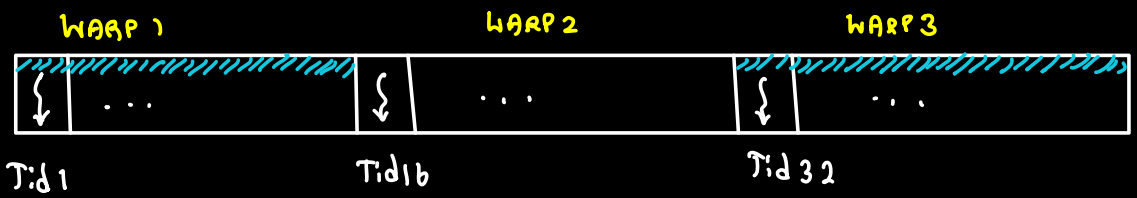


SIMT:

* Merge vector and thread parallelism.

→ Take massive array of threads

→ Group threads to execute together as a vector (WARP)



So SIMT steps will be

Warp Select → Warp Fetch → Decode → Execute → Complete.

H/W :	So, unit of execution is a 'warp'. Warp has registers, context [Not threads]
S/W :	Unit is a thread.

When to use SIMT:

→ Extreme parallelism

→ Negligible locality. So each thread can be seen independent → don't share much data with others.

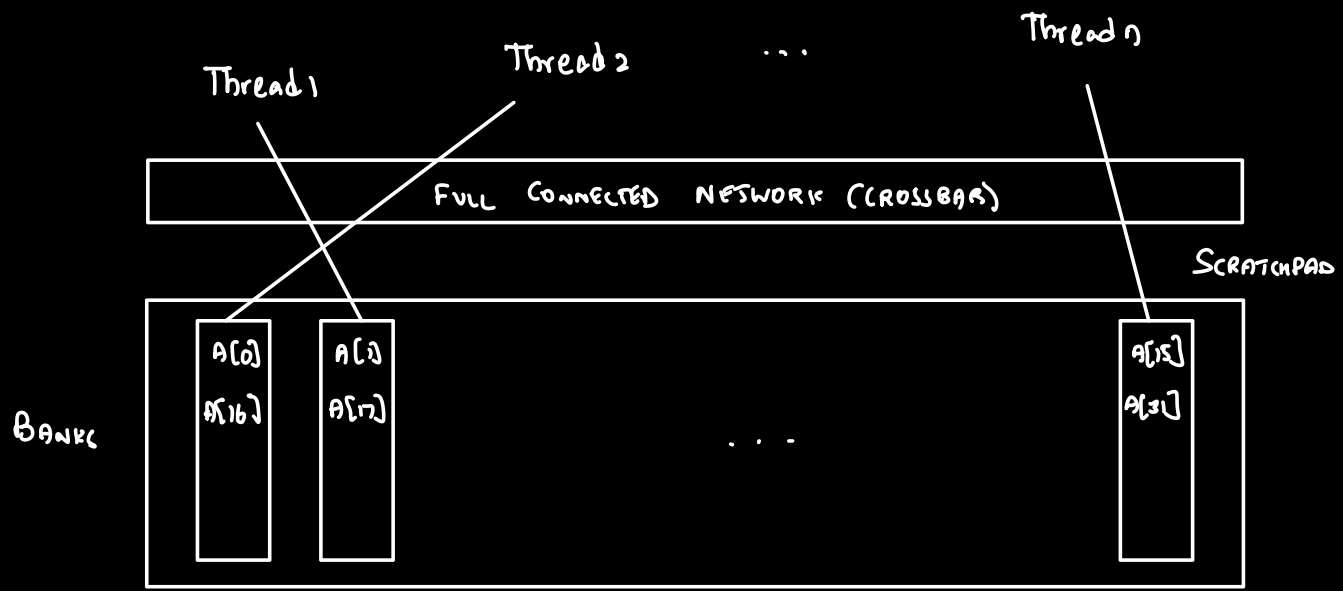
So, threads do NOT have Program Counter (PC) or context. Warp has. So all the threads in a warp has to be in the same program state.

So if the threads have if-else, both if and else are run for all threads, just that the threads will be active during only one of those.

[Divergence stack]

SCRATCHPAD:

Software Cache



→ Load input into scratchpad. Programmer does it.

→ Stripe the data across the banks.

→ As long as 2 threads do not use same banks, completely parallel.

ADVANTAGES:

* Parallel
* No unwanted cache miss / tag check etc.
* Load data according to program requirements. (Code need not have any locality).

What if 2 threads need same bank access?

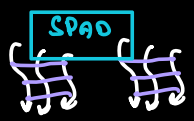
Divide the threads in a warp into groups such that within a grap, no same bank access. eg: 32 threads in a warp.

2 groups 16 each. 2 cycles needed.

Who shares SPAD?

Block: Group of threads that can access the same shared SPAD. [Group of warps]

Grid (kernel) → Block → Warp → Thread



SYNCHRONIZATION:

-- Syncthreads()

Synchronize within a block!

1 Block has to be within 1 SM.

1 SM → 1 SPAD