Discussion #8:

Hints for Lab #3 and Mid-Term

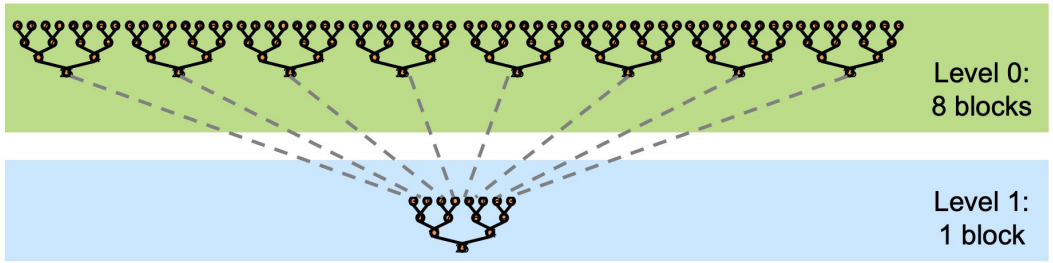
Daniel Tan, Jason Lau, and Jason Cong

Hints for Lab #3

- How to Choose Thread Blocks
- Key Point #1: Coalesced Global Memory Access
- Key Point #2: Avoid Divergence in a Warp
- Key Point #3: Avoid Local Memory Bank Conflicts
- Key Point #4: Loop Unrolling

OpenCL Optimization on GPU: Reduction

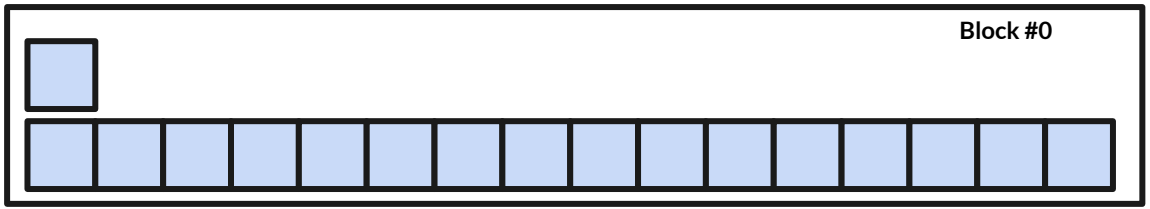
- Some examples borrowed from *Optimizing Parallel Reduction in CUDA, by Mark Harris, NVIDIA Developer Technology*. (original slides here)
  - *"Common and important data parallel primitive"*
  - *"Easy to implement in CUDA"*
  - *"Hard to get it right"*
- *"Tree-based approach used within each thread block"*



Example: Reduction

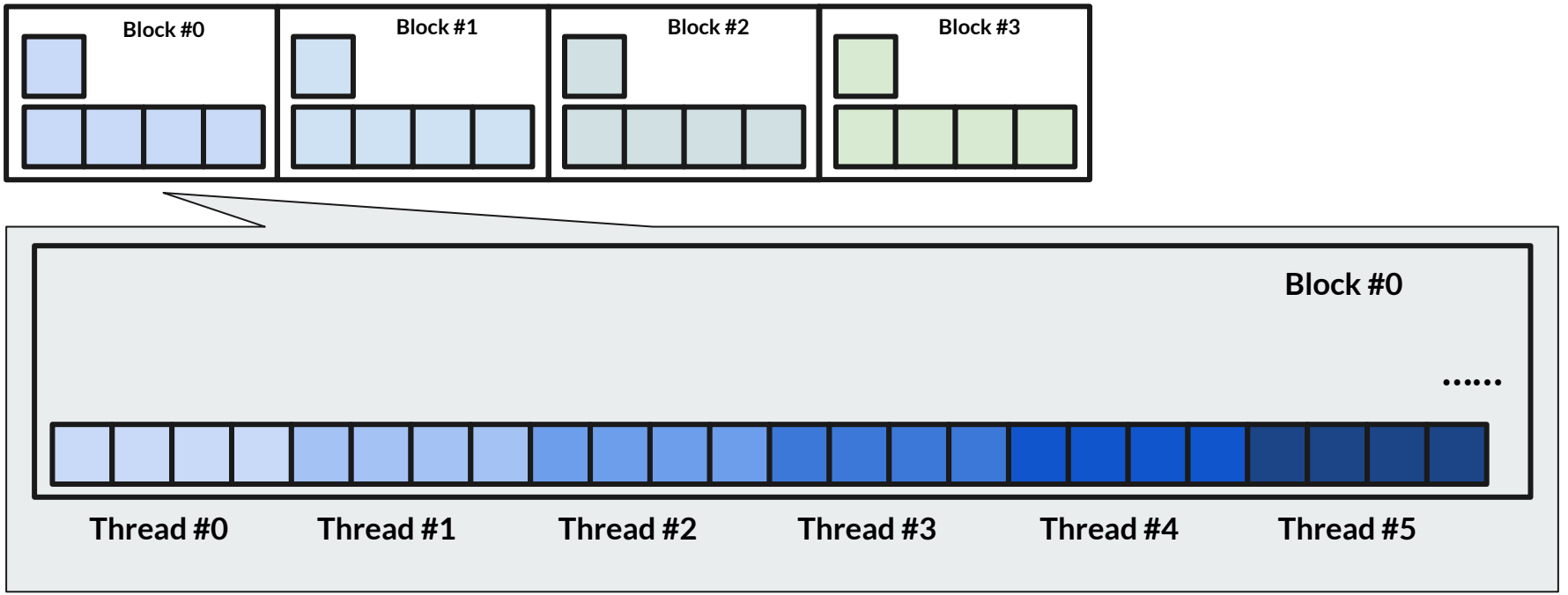
- **Sequential Implementation**

```
__global__ void reduce(int N, float* output, float* input) {
    float sum = 0;
    for (int i = 0; i < N; i++)
      sum += input[i];
    output[0] = sum;
}
```



only for demo
schematic, not tested

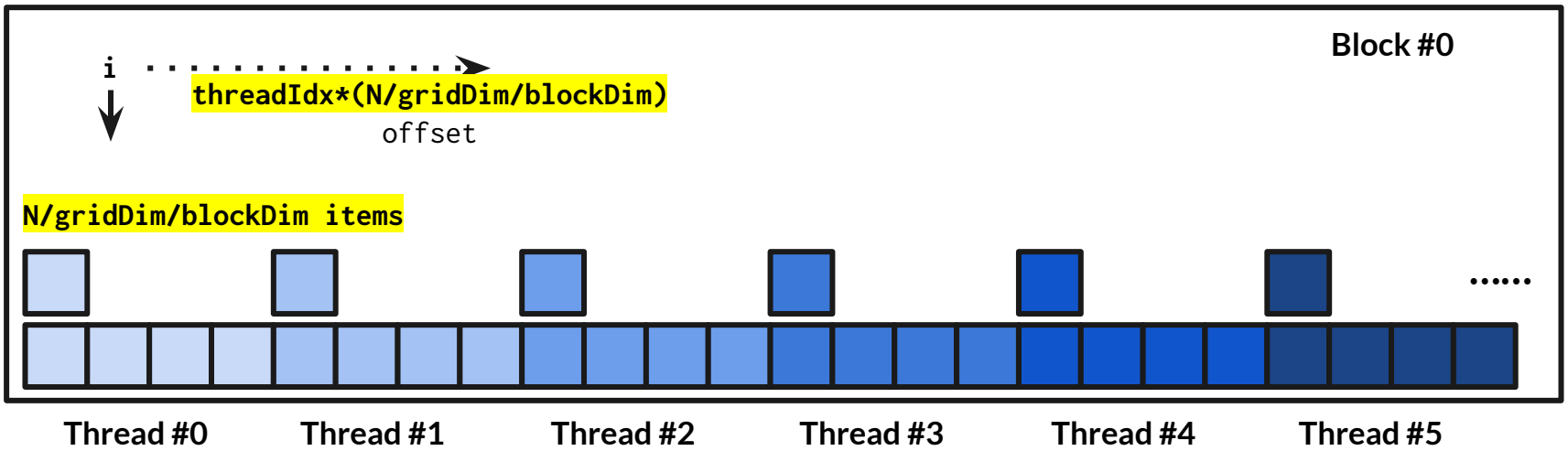
Example: Reduction with threads and blocks



Example: Reduction inside a block

- How to use threads?

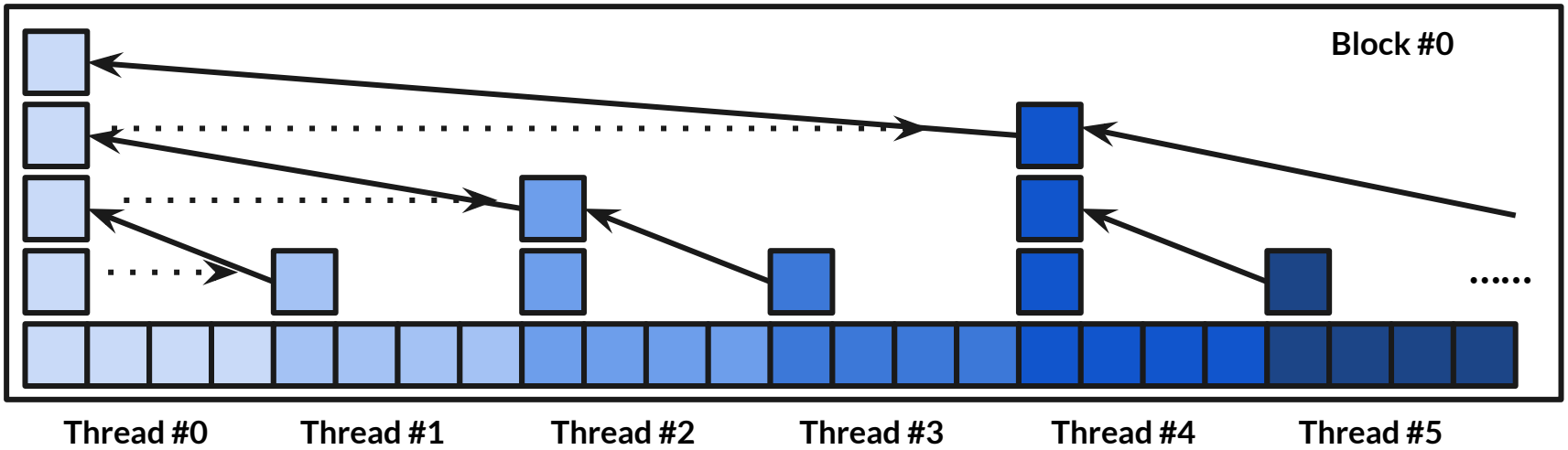
```
for (int i = threadIdx*(N/gridDim/blockDim); i < (1+threadIdx)*(N/gridDim/blockDim); i++)
  sum[threadIdx] += input[blockIdx*(N/gridDim) + i];
```



Example: Reduction inside a block

```
…
__syncthreads();

for (int l = 1; l < blockDim; l *= 2) {
  if (threadIdx % (2*l) == 0) sum[threadIdx] += sum[threadIdx + l];
  __syncthreads(); }
```



CUDA Threads and Blocks

- **CUDA Thread Blocks**
  - **Low overhead in GPU**
  - **As much as you want**
  - **No synchronization in between**
- **CUDA Threads**
  - **Sharing some data**
  - **At least 64 (2 warps)**
  - **Be mindful of limitations**

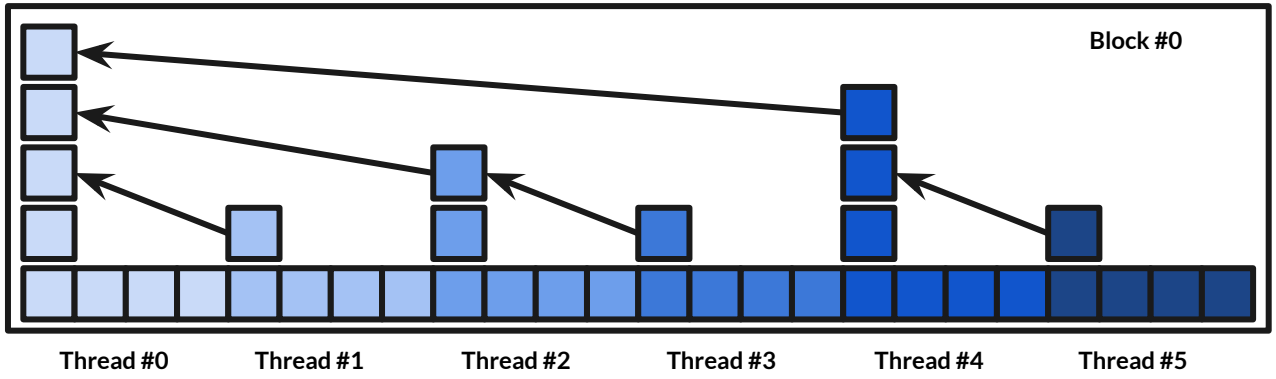
	GPU
Workgroups	**Blocks** Assigned to SM
Useful Number of Blocks	**SM** count **or More**
Threads	**SIMT** Across Warps
Useful Number of Threads inside Each Block	**Multiple of CUDA Cores Per SM**
`__device__`	**Device DRAM**
`__shared__`	**Shared Memory**
Private Variables (Default)	**Registers / Local Mem**

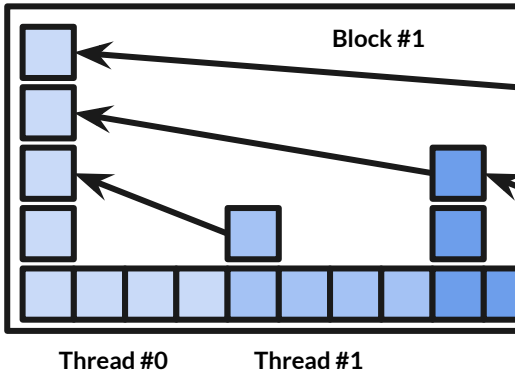
Example: Reduction of all blocks

```
# lab3/params.sh

export GRID='2'
export BLOCK='6'

# 2 Blocks, 6 Threads per Block
```





Blocks and threads for CNN

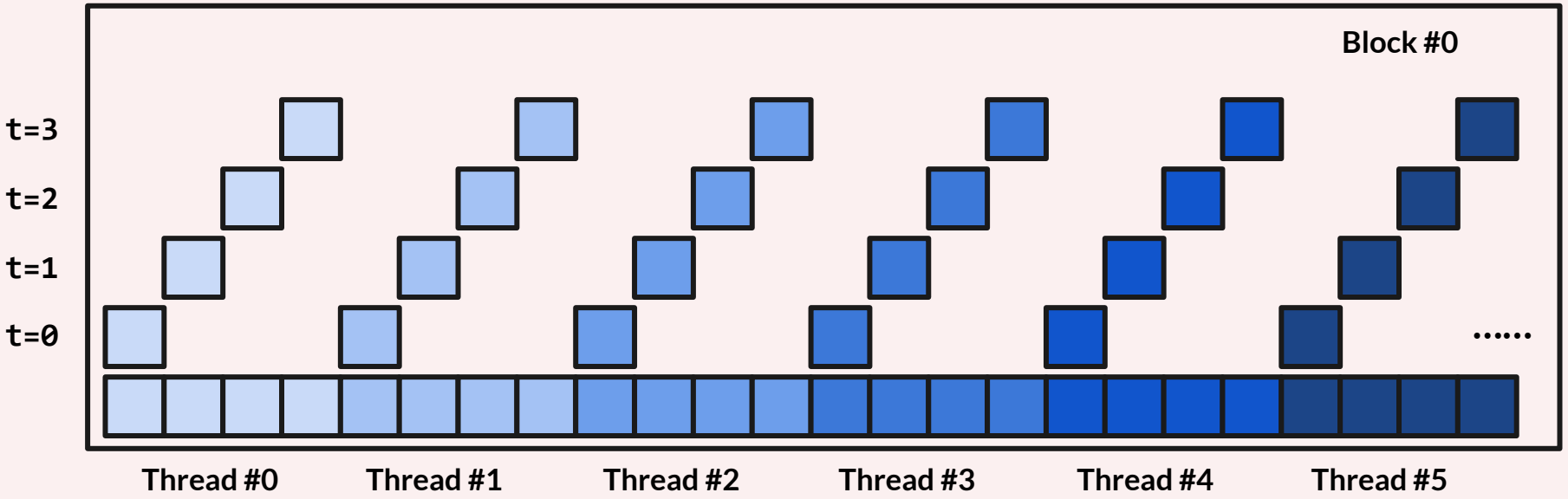
```
// Set bias
for (i, h, w) {{ Ctemp(i, h, w) = bias[i]; }}
// Convolution
for (i, j, h, w, p, q) {{{{{
    Ctemp(i, h, w) += weight[i][j][p][q] * input[j][h + p][w + q];
    }}}}}
// ReLU
for (i, h, w) {{{ C[i][h][w] = max(0.f, C[i][h][w]); }}}
// Max pooling
for (i, h, w) {{{
    output[i][h][w] = max(
        max(C[i][h * 2][w * 2    ], C[i][h * 2 + 1][w * 2    ]),
        max(C[i][h * 2][w * 2 + 1], C[i][h * 2 + 1][w * 2 + 1]));
     }}}
```

Wikipedia:

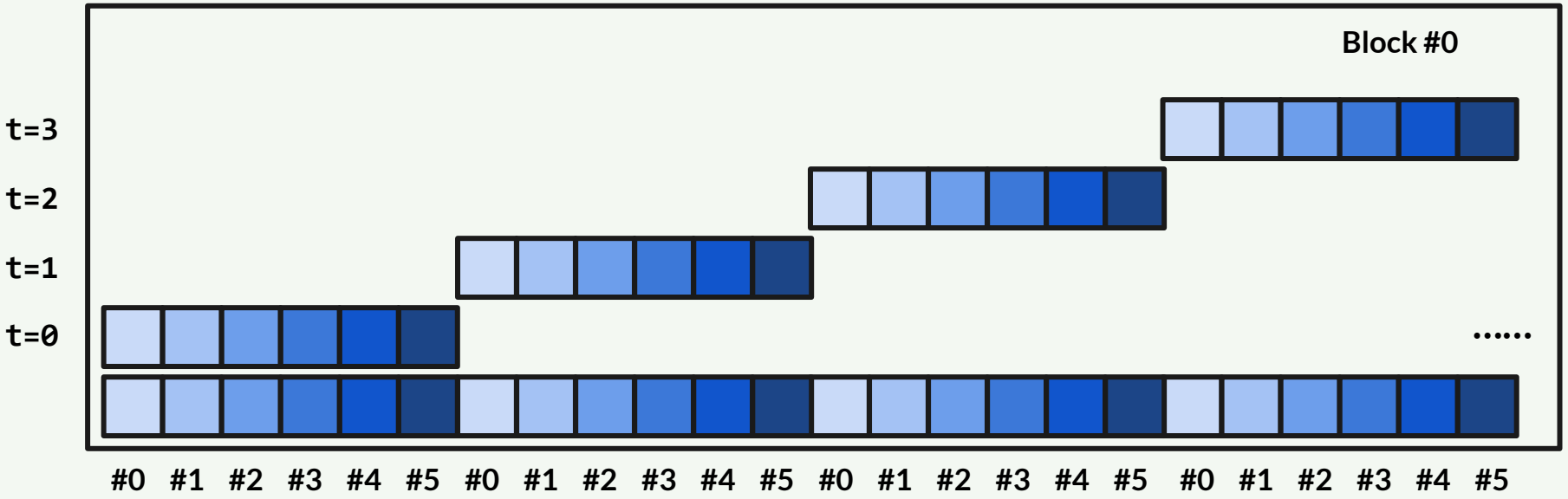
‘ ... The blocks in a grid must be able to be **executed independently,** as communication or cooperation between blocks in a grid is not possible … The maximum **x, y and z** dimensions of a block are **1024, 1024 and 64**, and it should be allocated such that **x × y × z ≤ 1024**, which is the maximum number of threads per block. Blocks can be organized into one- or two-dimensional grids of **up to 65,535 blocks** in each dimension.’

Also, check Programming Guide :: CUDA Toolkit Documentation (nvidia.com) Compute Capability **5.2**

Key Point #1: Poor Global Memory Access



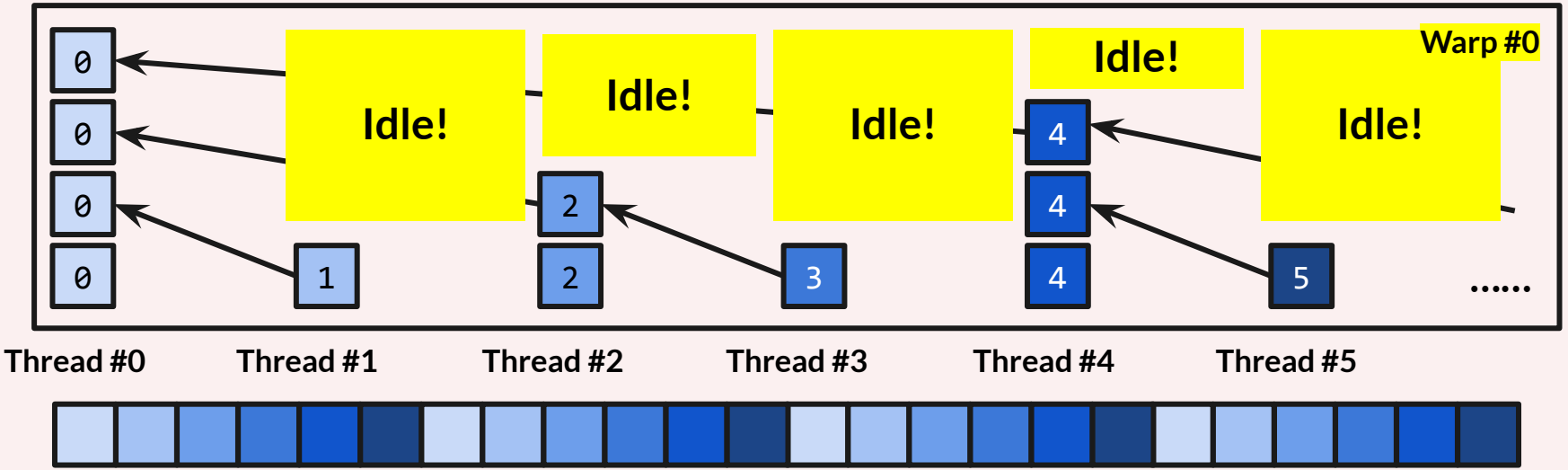
Key Point #1: Good Global Memory Access



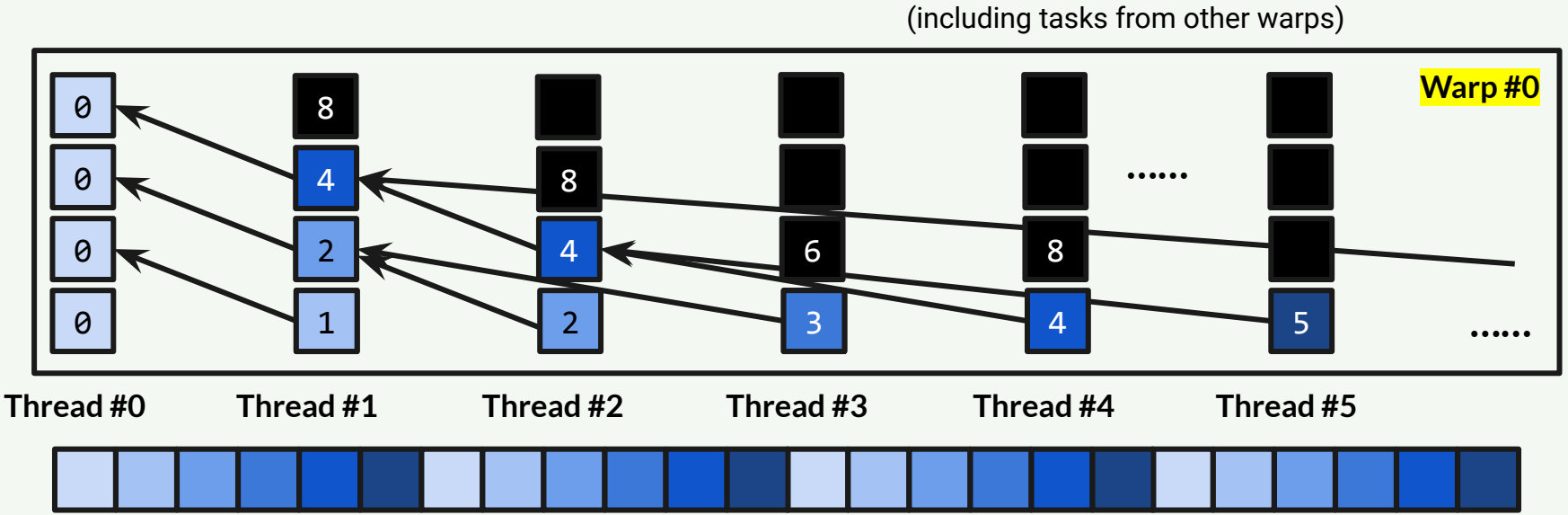
Key Point #2: Divergence of Threads in a Warp

Warp time = Active + Idle (for all warps)

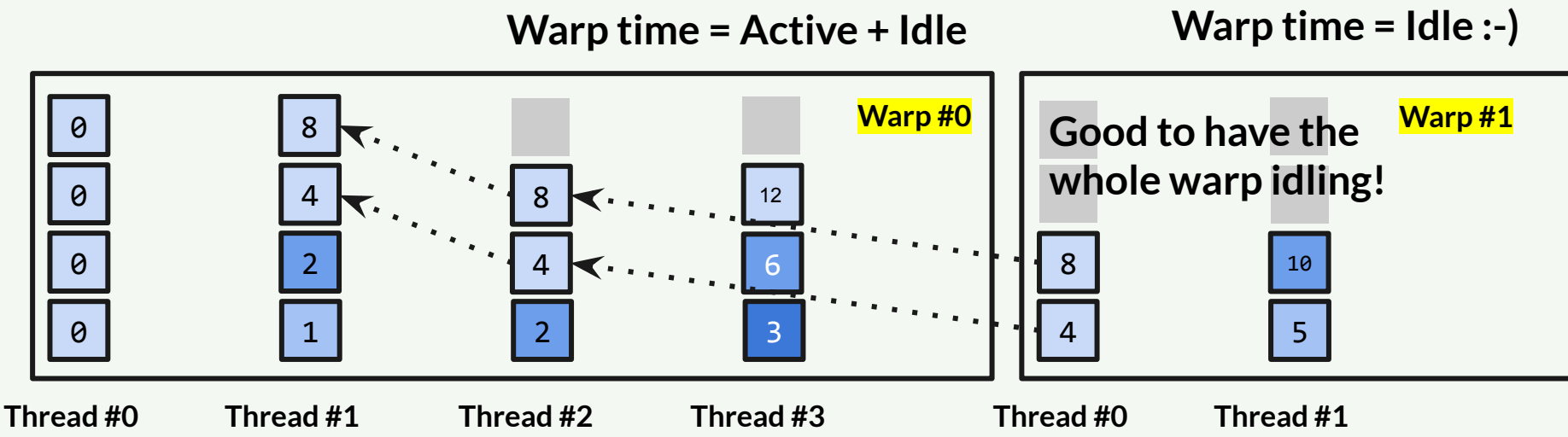
All warps has some thread idling :-(



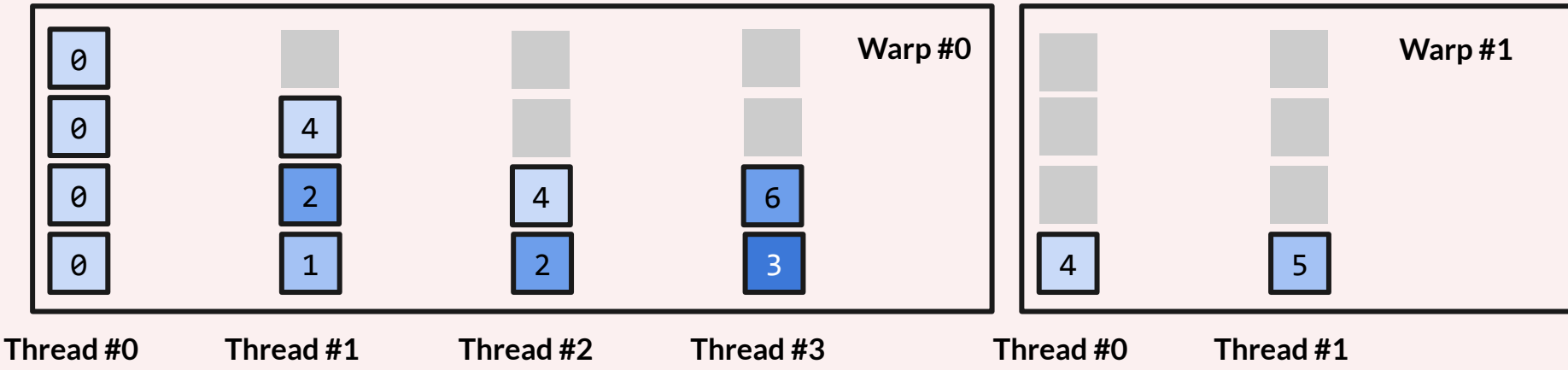
Key Point #2: Less Divergence in a Warp



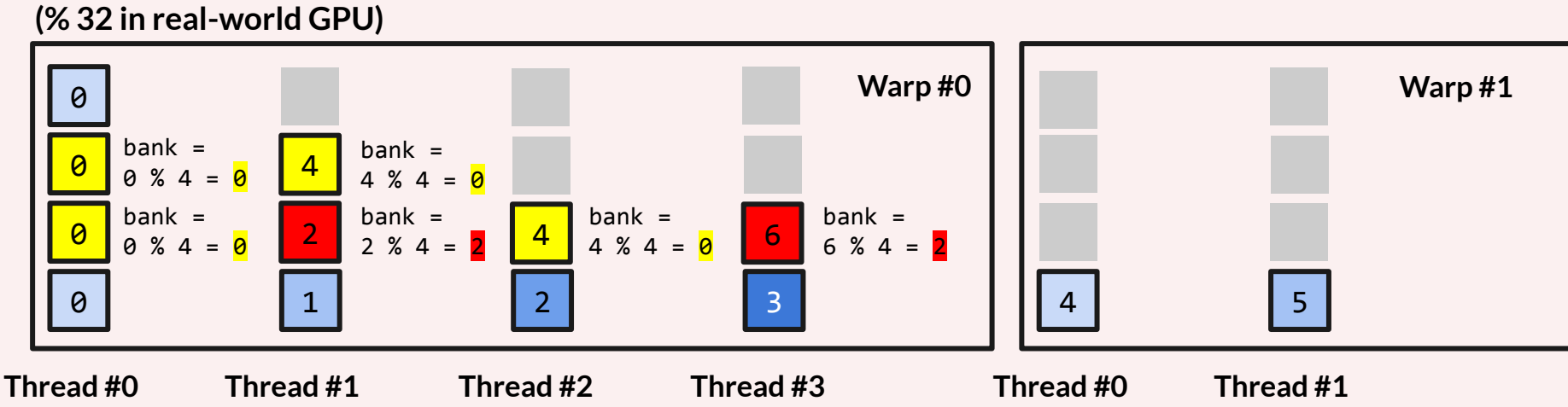
Key Point #2: Less Divergence in a Warp



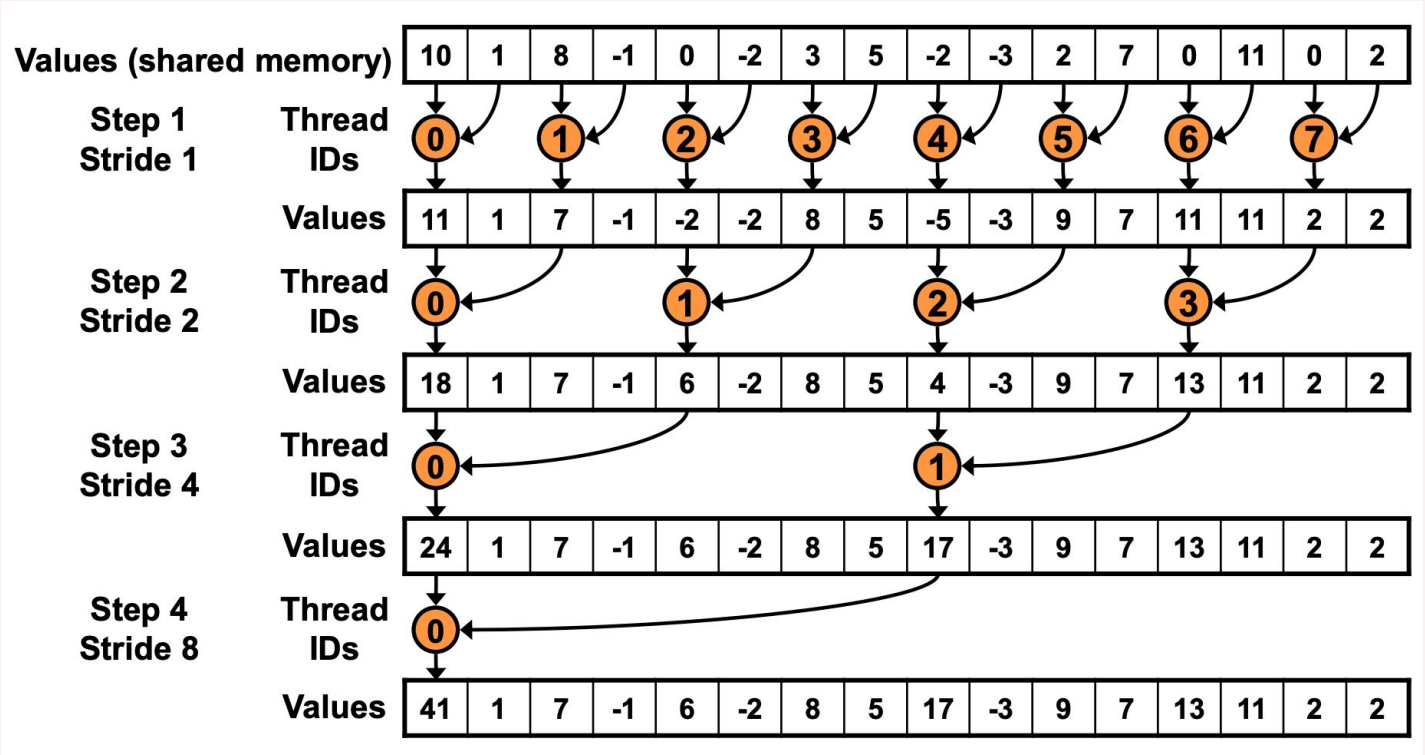
Key Point #3: Local Memory Bank Conflicts



Key Point #3: Local Memory Bank Conflicts

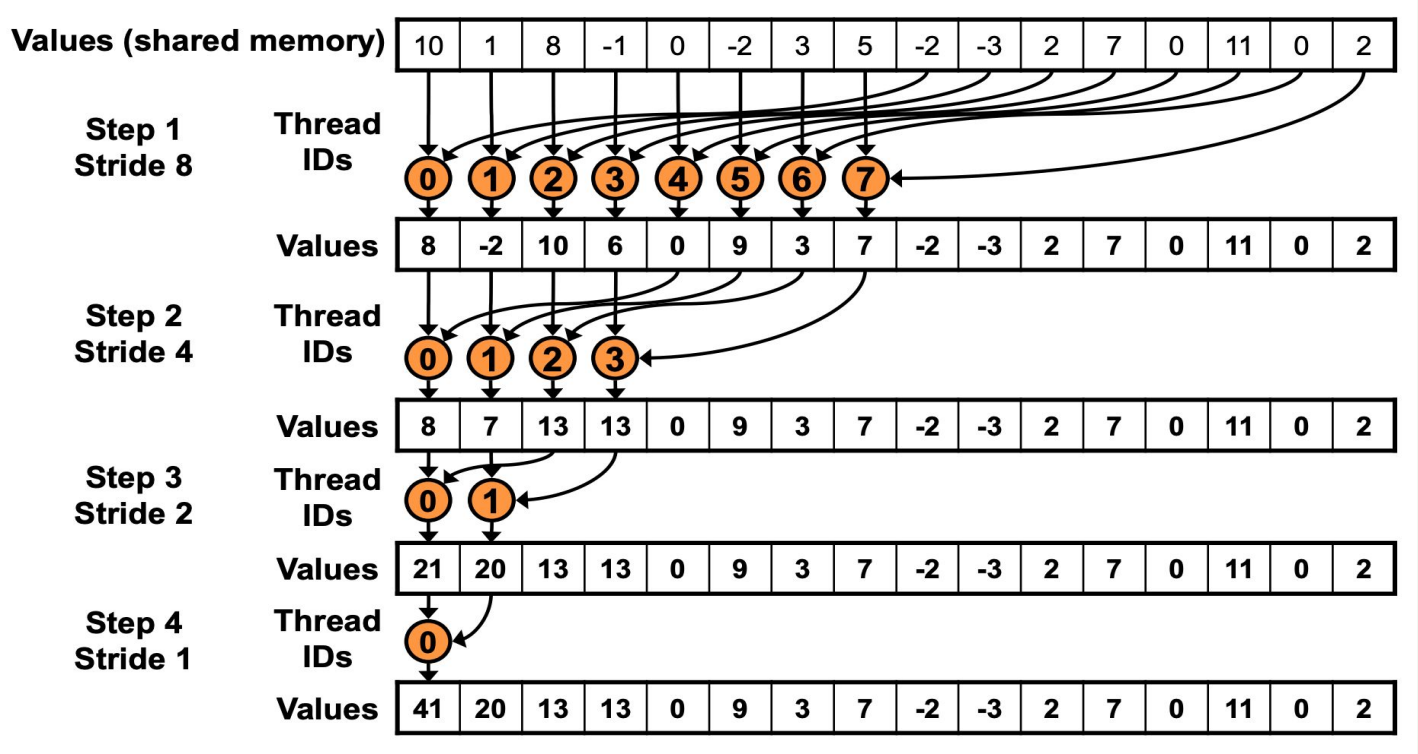


Key Point #3: Local Memory Bank Conflicts



from NVIDIA slides

Key Point #3: Avoid Bank Conflicts



from NVIDIA slides

Key Point #4: Controls are Expensive -> Unroll

```
for (int stride = N/2; stride > 0; l /= 2) {
        if (threadIdx < stride) sum[threadIdx] += sum[threadIdx + stride];
        __syncthreads(); }
```

```
 // ......
      if (...) { sum[threadIdx] += sum[threadIdx + 512]; __syncthreads();}
      if (...) { sum[threadIdx] += sum[threadIdx + 256]; __syncthreads();}
      if (...) { sum[threadIdx] += sum[threadIdx + 128]; __syncthreads();}
      if (...) { sum[threadIdx] += sum[threadIdx + 64 ]; __syncthreads();}
      if (threadIdx < 32) {
        sum[threadIdx] += sum[threadIdx + 32];
        sum[threadIdx] += sum[threadIdx + 16];
        sum[threadIdx] += sum[threadIdx + 8];
        sum[threadIdx] += sum[threadIdx + 4];
        sum[threadIdx] += sum[threadIdx + 2];
        sum[threadIdx] += sum[threadIdx + 1];
      }
```

Q&A