Kernel calls

main: add <<< N_1, N_2 >>> (...)

N_1: gridDim.x
N_2: blockDim.x

kernel: gridDim.x blockDim.x

general form of kernel call:

Kernel <<< X_1, X_2, [n, s] >>> (...)

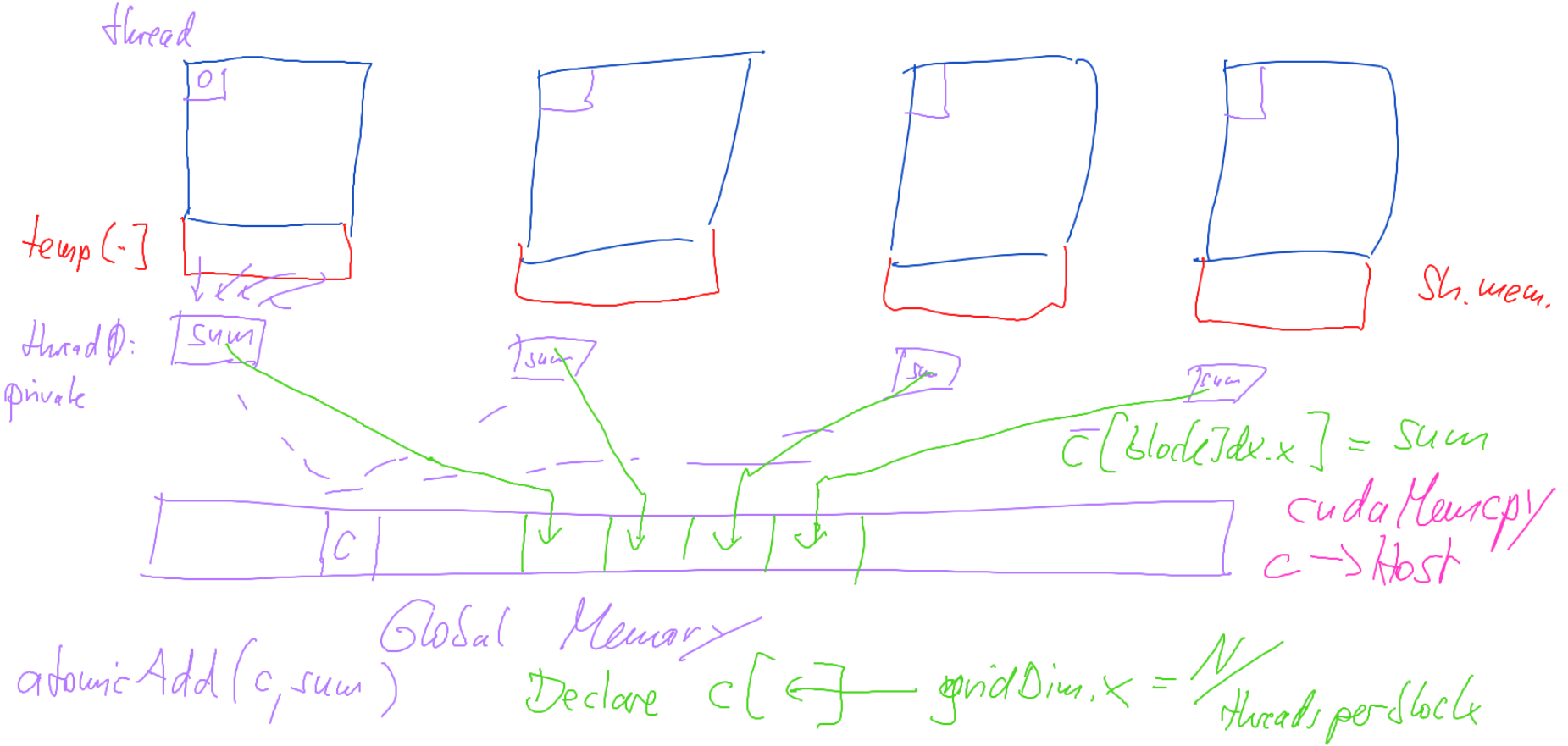
X_1, X_2: dim3 objects
$X_1, X_2 = (n_1, n_2, n_3) = (m_1, m_2, m_3)$

just integers N_1, N_2: $X_1 = (N_1, 0, 0)$
$X_2 = (N_2, 0, 0)$

$n_{1..3}$ = gridDim.x, y, z
$m_{1..3}$ = blockDim.x, y, z

64000 > n: number of bytes to be allocated in shared memory for each block! (dynamic)

s: stream id ⟷ grid identification


thread
0
temp[-]
Sh. mem.
thread 0:
sum
private
sum
sum
sum
c[blockIdx.x] = sum
cudaMemcpy
c -> Host
c
Global Memory
atomicAdd(c, sum)
Declare c[ ] — gridDim.x = N / threadsperblock

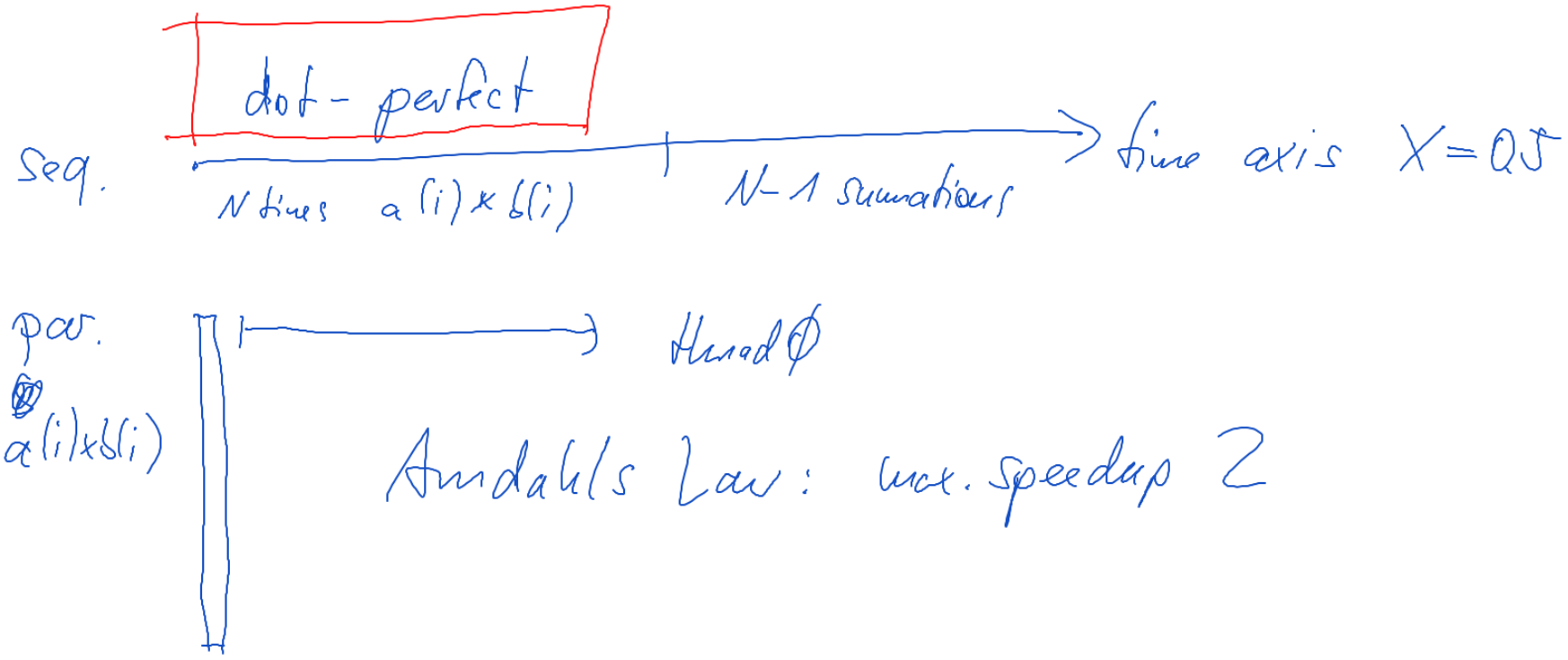
Version 1: AtomicAdd(c, sum)

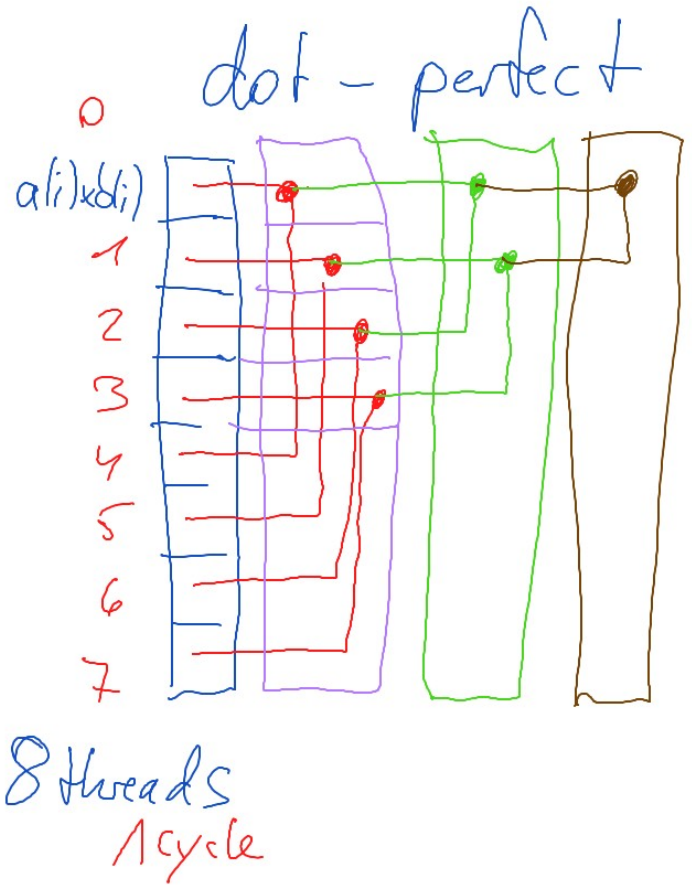
Sum Result of every block into one scalar c (global mem.)

Version 2: c[blockIdx.x] = sum

Store result of every block in a separate vector element of c, sum on the host

Avoids race condition!


dot - perfect
seq.
N times a(i) * b(i)
N-1 summations
time axis X = 05
par.
a(i)xb(i)
Hund ∅
Amdahls Law: max. speedup 2



4 cycles to get all
a(i) x b(i) into sum in thread 0

(previous one: 7 cycles needed!)