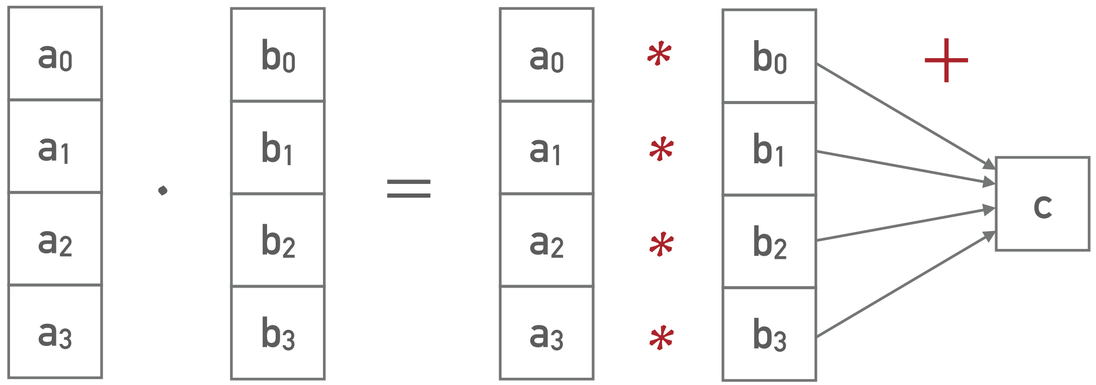
ME 471/571

Week 13 - Shared memory and thread synchronization

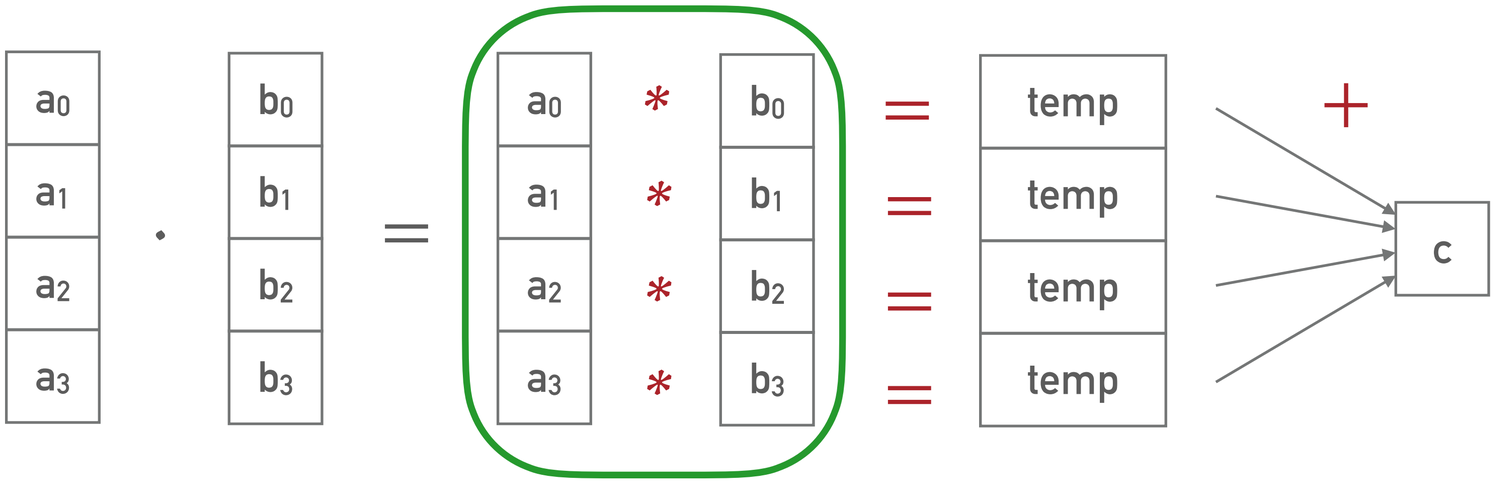
THREADS SYNCHRONIZATION

Consider a computation of dot product:



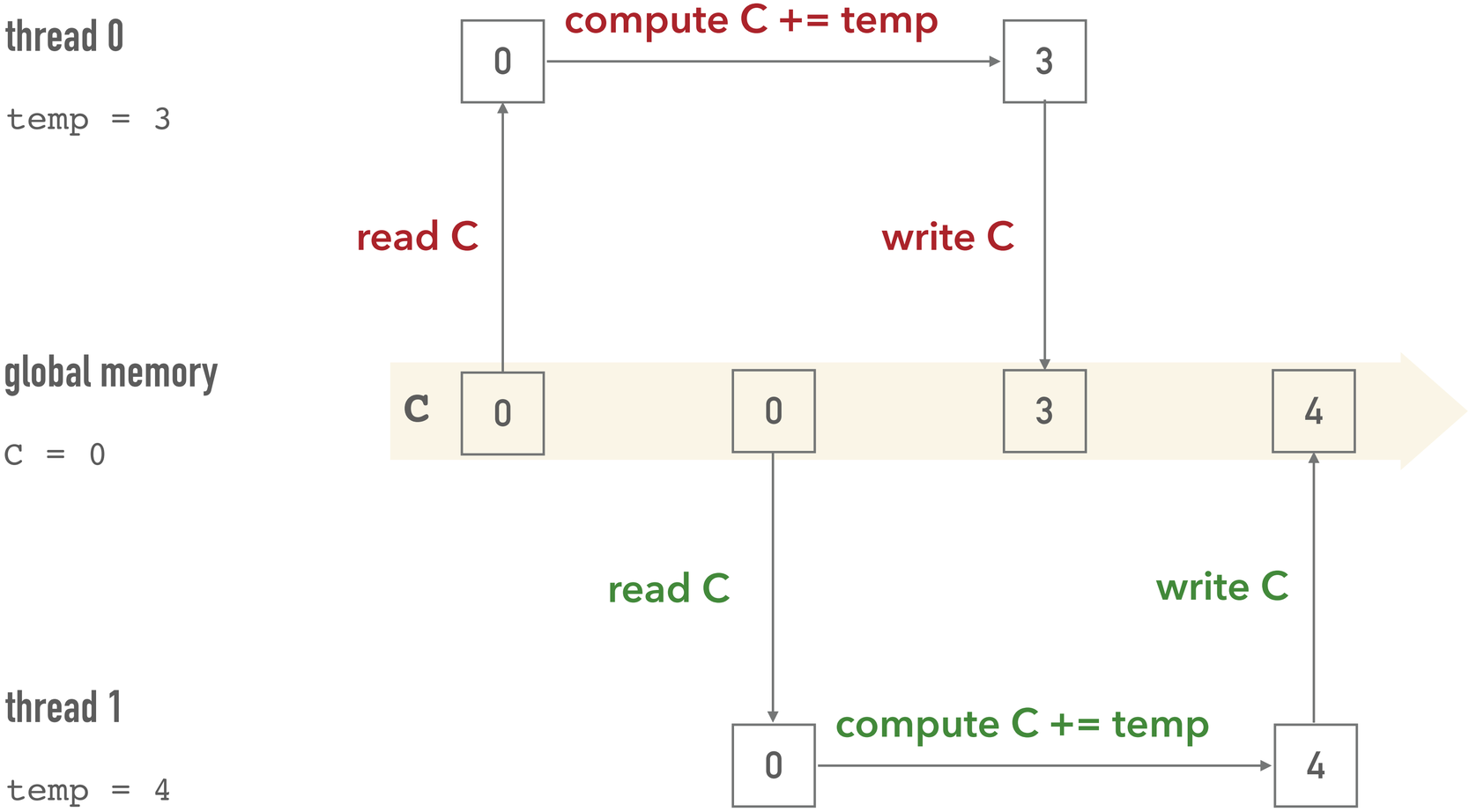
THREADS SYNCHRONIZATION

Parallel threads have no problem with the multiplication part:



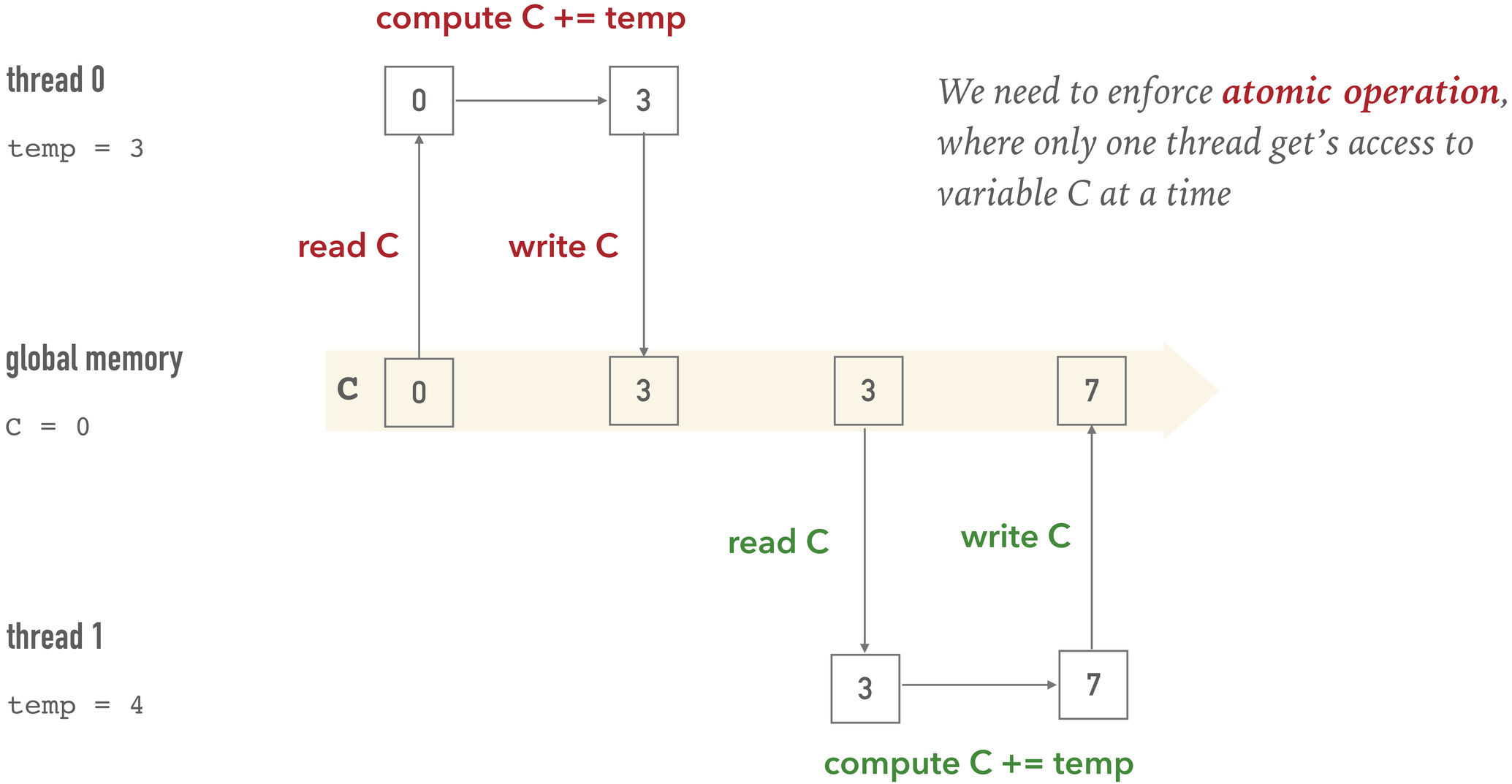

```
__global__ void dot( float *a, float *b, float *c, int N ) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    // Each thread computes a pairwise product
    float temp += a[idx] * b[idx];
    c += temp;
}
```

DATA RACE CONDITION



*We cannot guarantee when each thread will read the memory, which leads to the **data race condition**.*

DATA RACE CONDITION



*We need to enforce **atomic operation**, where only one thread get's access to variable C at a time*

ATOMIC OPERATIONS

Atomic operations ensure that only one thread can access and modify a memory location. Other threads need to wait until atomic operation is completed.

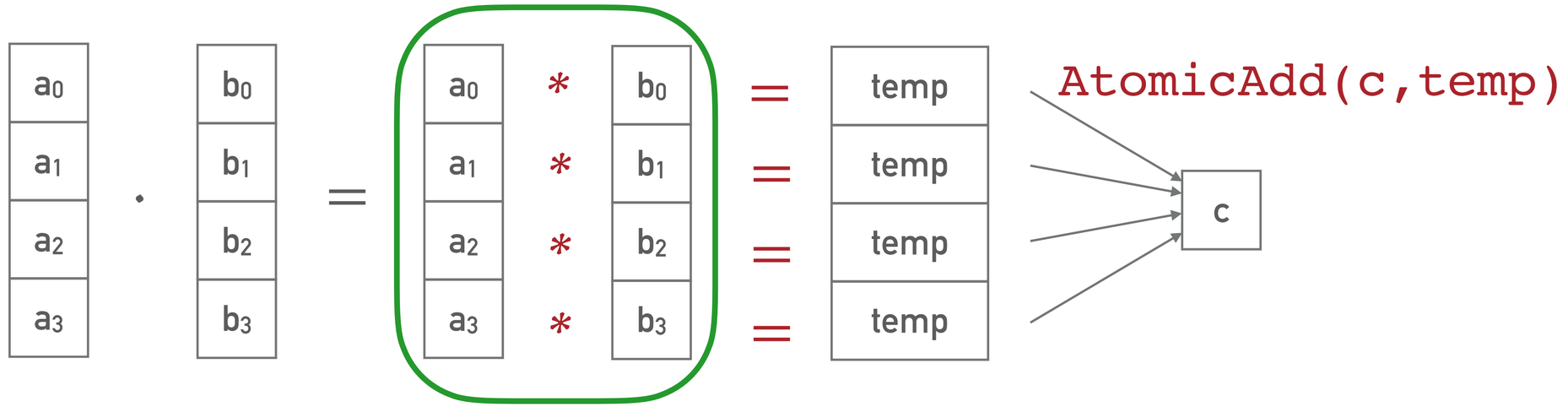
- atomicAdd *addition*
- atomicSub *subtraction*
- atomicMin *minimum*
- atomicMax *maximum*
- atomicInc *increment*
- atomicDec *decrement*
- atomicExch *exchange*
- atomicCAS *compare and swap*

atomicAdd_block *atomic with respect to block only*

atomicAdd_system *atomic with respect to GPU and CPU (for unified memory)*

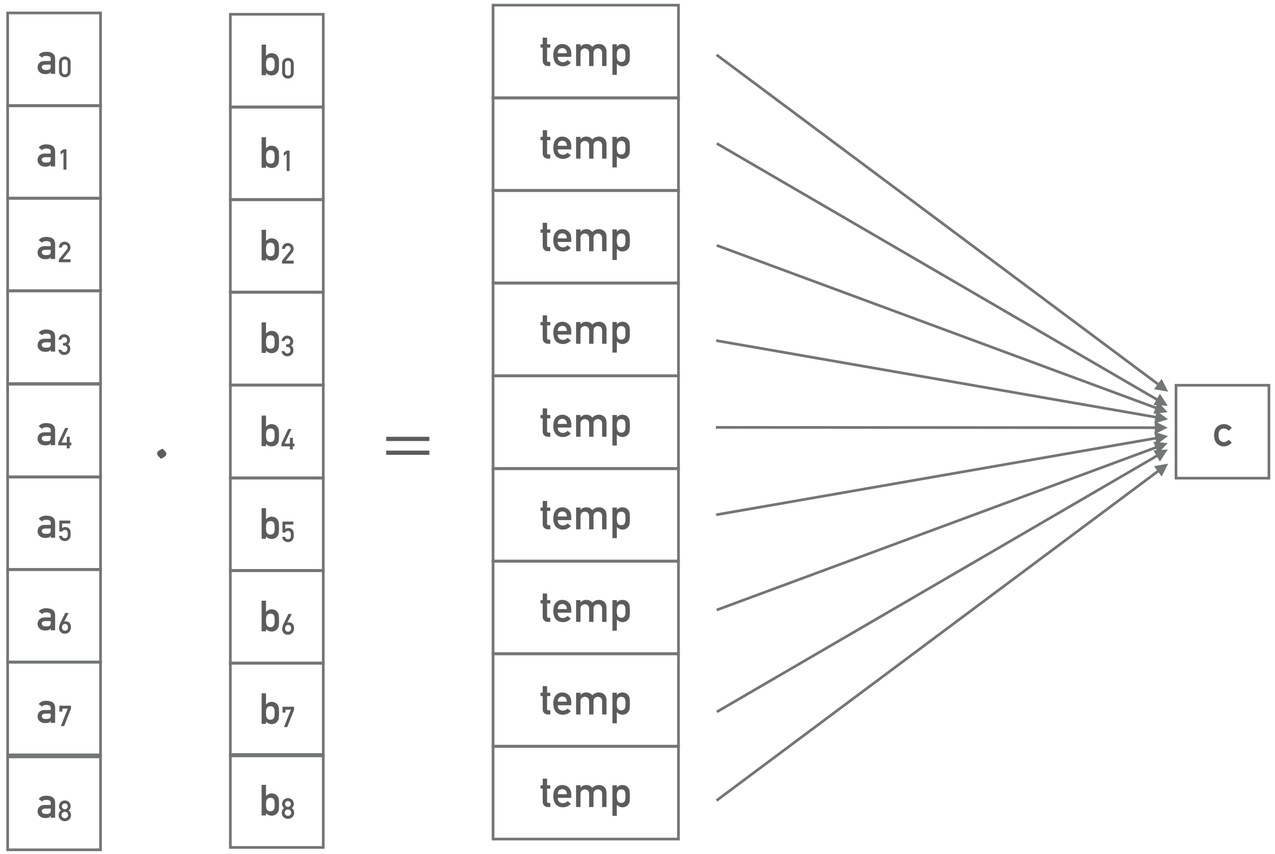
THREADS SYNCHRONIZATION

Parallel threads have no problem with the multiplication part:

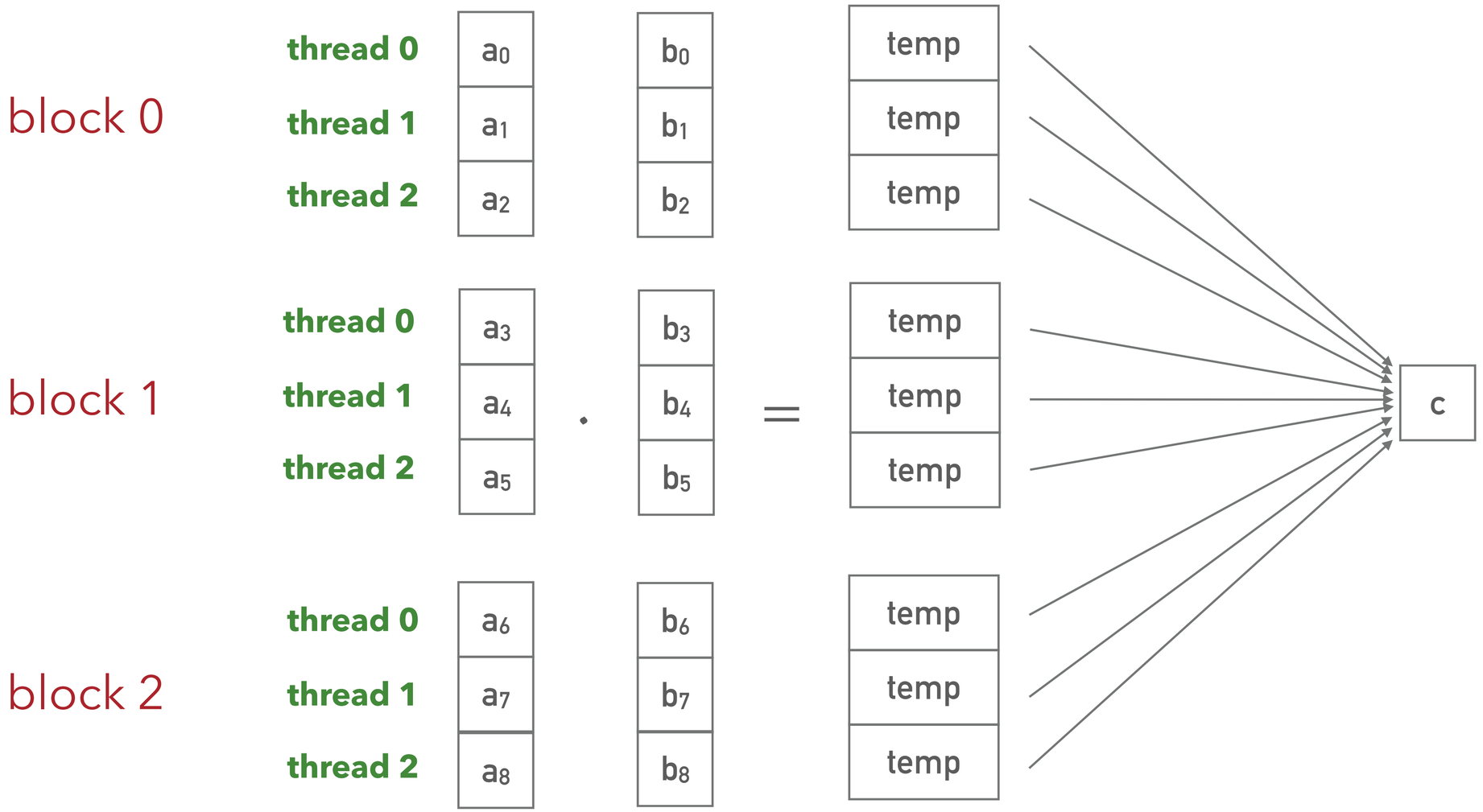



```
__global__ void dot( float *a, float *b, float *c, int N ) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    // Each thread computes a pairwise product
    float temp += a[idx] * b[idx];
    AtomicAdd(c,temp);
}
```

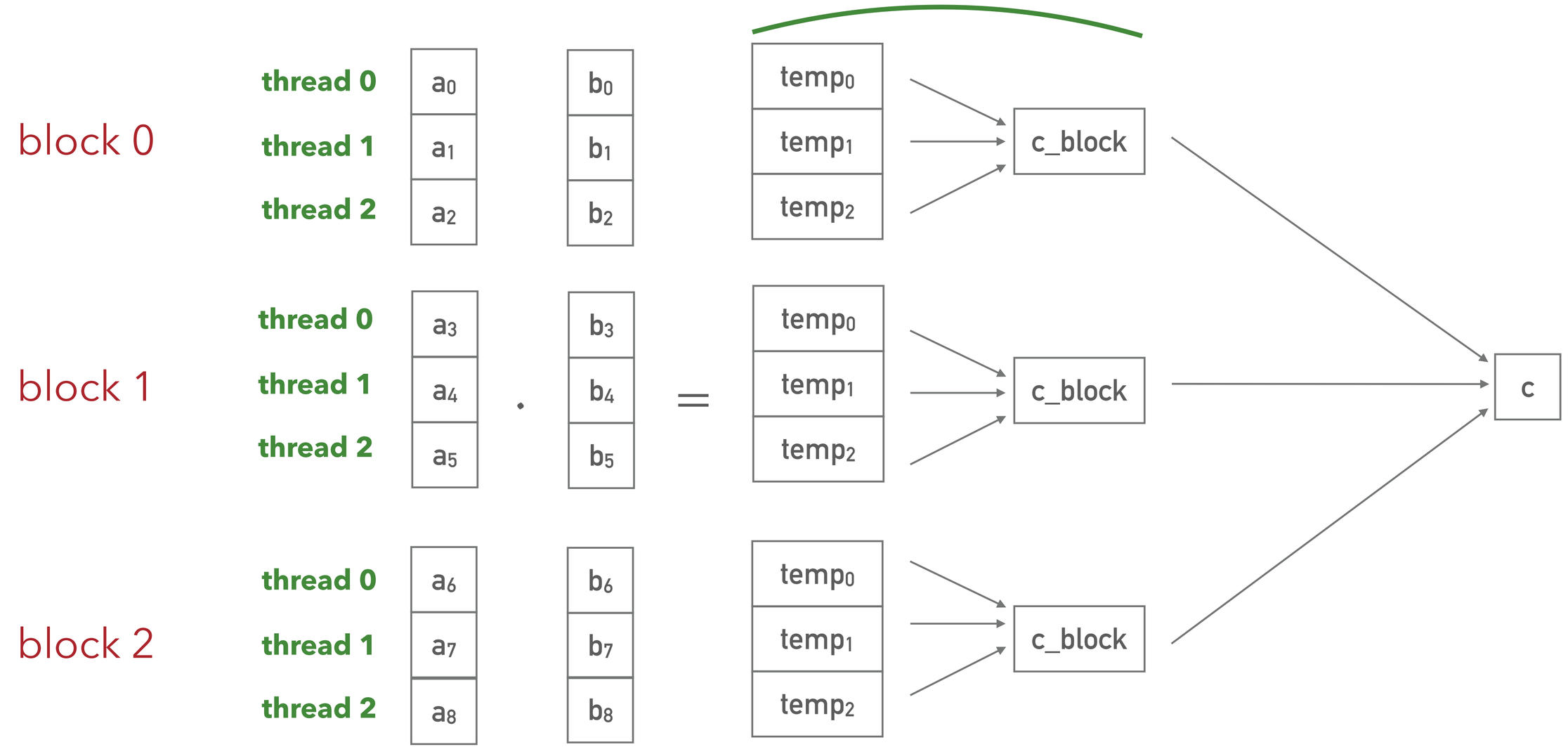
BLOCKS AND THREADS



BLOCKS AND THREADS



BLOCKS AND THREADS



THREADS SYNCHRONIZATION

Shared memory:

- is shared among the threads, but private to each block
- is extremely fast (think cache, but shared among threads)

```
__global__ void dot( int *a, int *b, int *c, int N  ) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
// Each thread computes a pairwise product
    __shared__ int temp[THREADS_PER_BLOCK];
    temp[threadIdx.x] = a[idx] * b[idx];
}
```

THREADS SYNCHRONIZATION

```
__global__ void dot( int *a, int *b, int *c, int N  ) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;

// Each thread computes a pairwise product
    __shared__ int temp[TREADS_PER_BLOCK];
    temp[threadIdx.x] = a[idx] * b[idx];

    __syncthreads();

    // Thread 0 sums up the pairwise products
    if(threadIdx.x == 0) {
        float block_c = 0;
        for (int i = 0; i<N; i++)
            block_c += temp[i];
        atomicAdd(c,block_c);
    }
```