6.593[01]
Hardware Architectures for Deep Learning

Lab 2 Mapping Code Tutorial

February 13, 2026

Michael Gilbert

Massachusetts Institute of Technology
Electrical Engineering & Computer Science

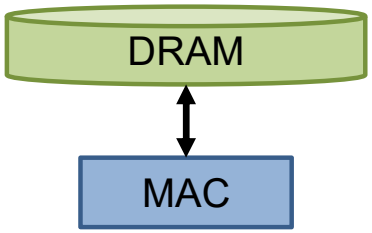
Goals of Today's Recitation

- Brief overview of mapping
- A tutorial on how to read and write a mapping in Lab 2.

Mapping: Scheduling Operations and Data Movement

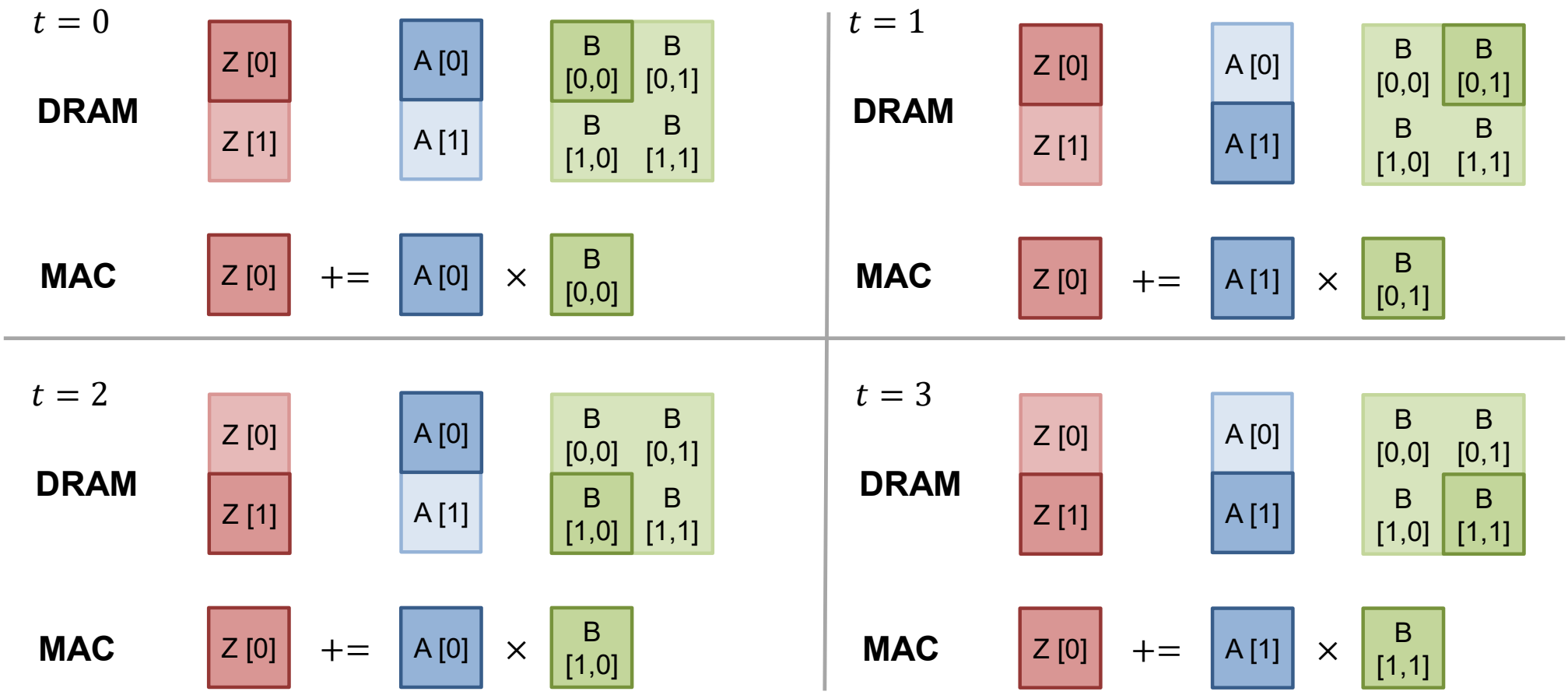
Our workload: $Z_n = A_k B_{k,n}$

Our architecture:

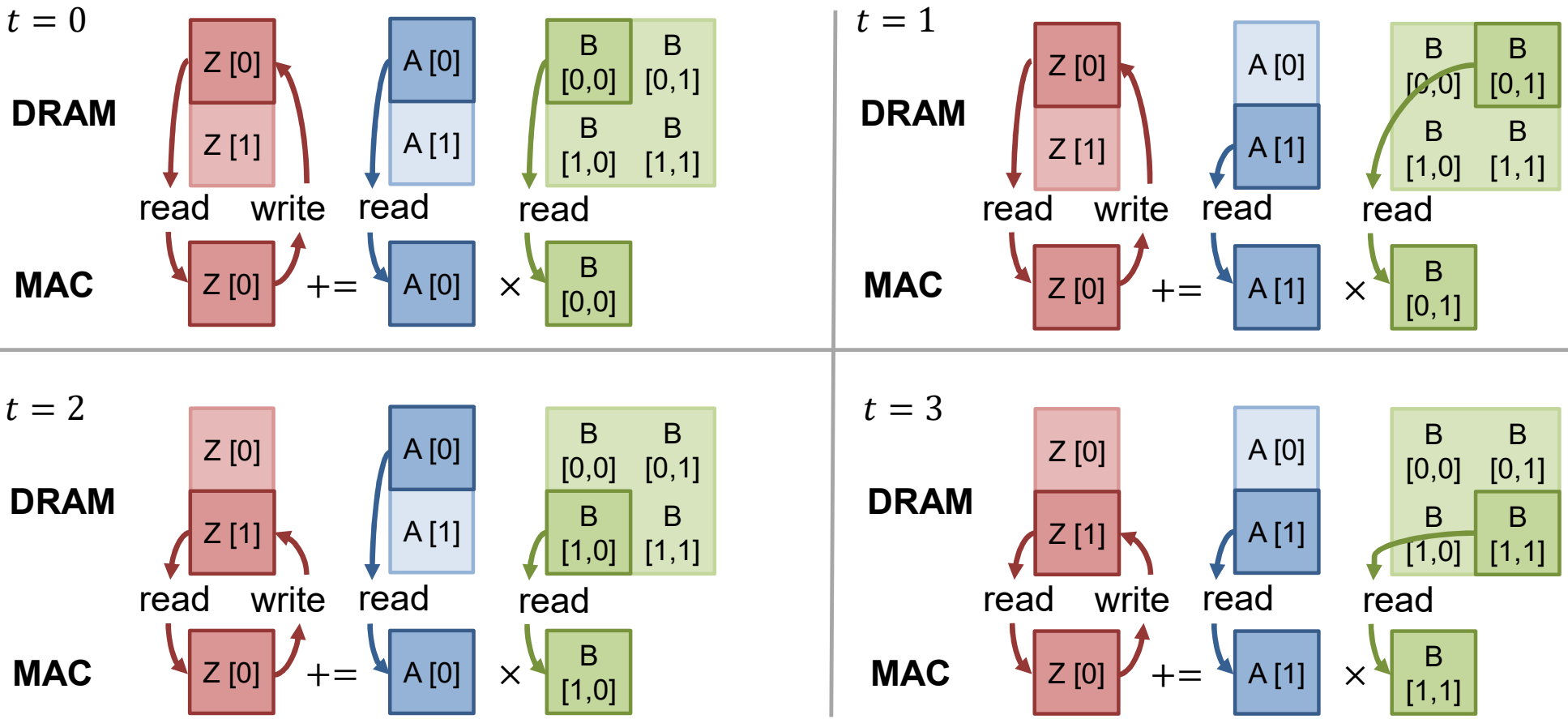


Mapping (of workload to architecture): the schedule of operations and data movement

Mapping: Scheduling Operations and Data Movement



Mapping: Scheduling Operations and Data Movement



Writing a Mapping in Python in Lab 2

```
3   K = 2
4   N = 2
5
6   # Starting tensors
7   A = Tensor("A", shape=(K))
8   B = Tensor("B", shape=(K, N))
9   Z = Tensor("Z", shape=(N), is_output=True)
10
11  # Memory levels
12  DRAM = MemoryLevel("DRAM")
13
14  # A loop nest that performs Z = A x B
15  with DRAM.allocate(A) as A_in_DRAM:
16      with DRAM.allocate(B) as B_in_DRAM:
17          with DRAM.allocate(Z) as Z_in_DRAM:
18              for n in range(N):
19                  for k in range(K):
20                      Z_in_DRAM[n] += A_in_DRAM[k] * B_in_DRAM[k,n]
```

Create tensors

Create memory levels

Mapping

Writing a Mapping in Python in Lab 2

```
3   K = 2
4   N = 2
5
6   # Starting tensors
7   A = Tensor("A", shape=(K))
8   B = Tensor("B", shape=(K, N))
9   Z = Tensor("Z", shape=(N), is_output=True)
10
11  # Memory levels
12  DRAM = MemoryLevel("DRAM")
13
14  # A loop nest that performs Z = A x B
15  with DRAM.allocate(A) as A_in_DRAM:
16      with DRAM.allocate(B) as B_in_DRAM:
17          with DRAM.allocate(Z) as Z_in_DRAM:
18              for n in range(N):
19                  for k in range(K):
20                      Z_in_DRAM[n] += A_in_DRAM[k] * B_in_DRAM[k,n]
```

Allocate the entirety of tensor A in DRAM

Reference to tensor A in DRAM

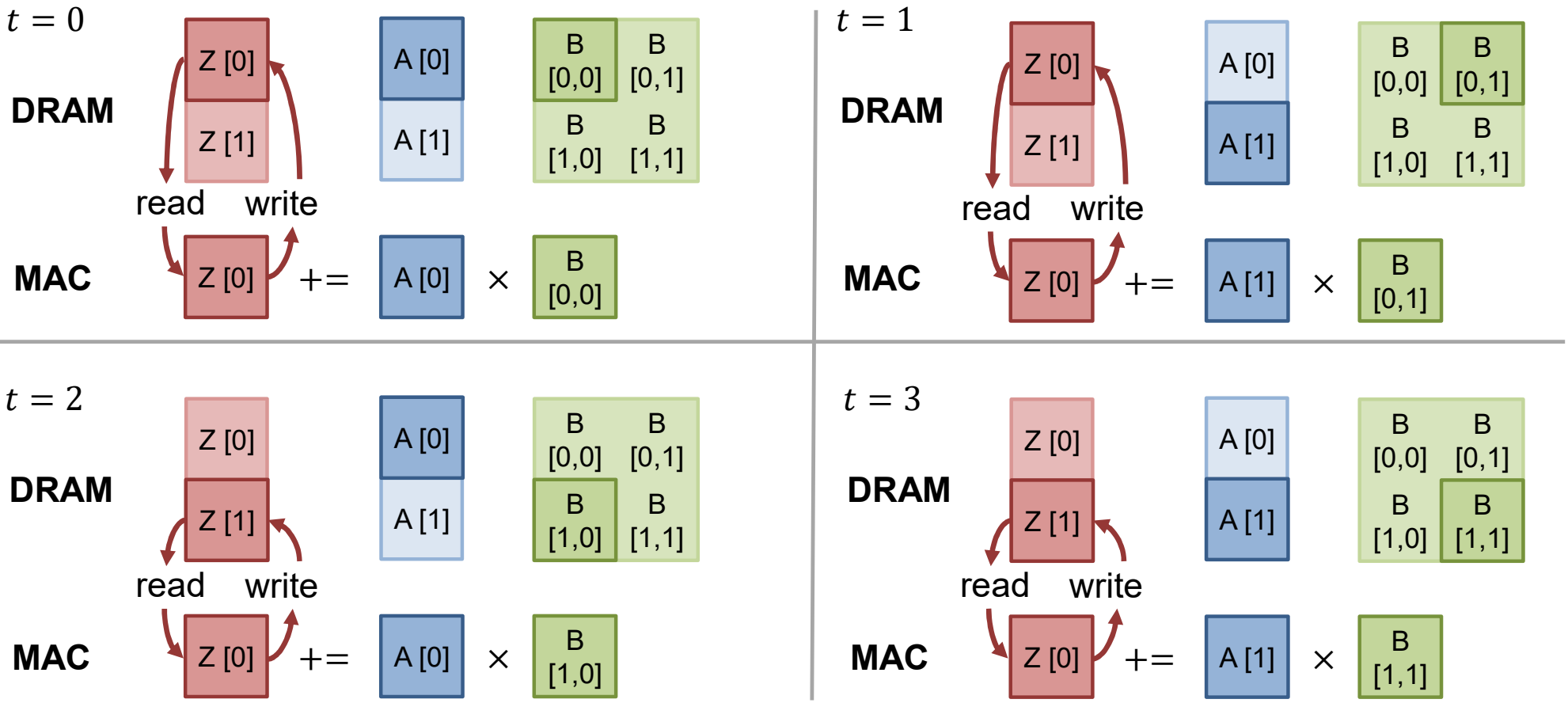
Accesses to A_in_DRAM are recorded

Writing a Mapping in Python in Lab 2

```
3   K = 2
4   N = 2
5
6   # Starting tensors
7   A = Tensor("A", shape=(K))
8   B = Tensor("B", shape=(K, N))
9   Z = Tensor("Z", shape=(N), is_output=True)
10
11  # Memory levels
12  DRAM = MemoryLevel("DRAM")
13
14  # A loop nest that performs Z = A x B
15  with DRAM.allocate(A) as A_in_DRAM:
16      with DRAM.allocate(B) as B_in_DRAM:
17          with DRAM.allocate(Z) as Z_in_DRAM:
18              for n in range(N):
19                  for k in range(K):
20                      Z_in_DRAM[n] += A_in_DRAM[k] * B_in_DRAM[k,n]
```

The order of loops determines the schedule of operations

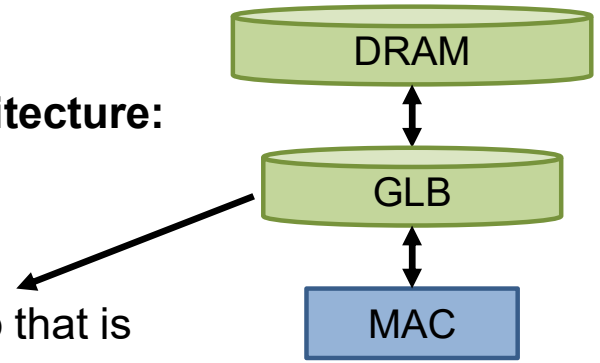
Inefficient Movement: Same Z Moved Repeatedly



Adding a Memory Level for Reuse

Our workload: $Z_n = A_k B_{k,n}$

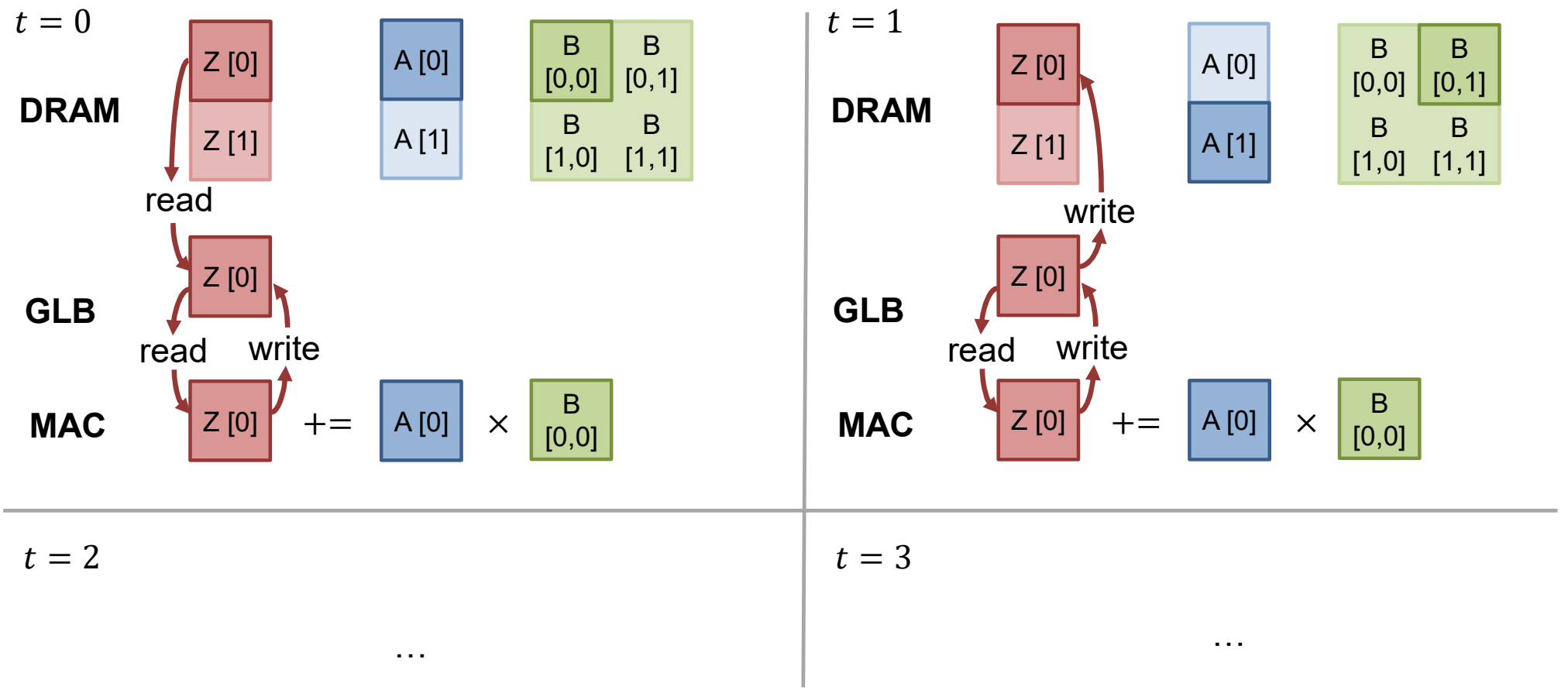
Our architecture:



Create a buffer on-chip that is cheaper and faster to access

[Source: Stanford cs231n]

Inefficient Movement: Same Z Moved Repeatedly



Adding a New Memory Level “GLB”

```
11  # Memory levels
12  DRAM = MemoryLevel("DRAM")
13  GLB = MemoryLevel("GLB")
14
15  # A loop nest that performs Z = A x B
16  with DRAM.allocate(A) as A_in_DRAM:
17      with DRAM.allocate(B) as B_in_DRAM:
18          with DRAM.allocate(Z) as Z_in_DRAM:
19              for n in range(N):
20                  with GLB.allocate(Z_in_DRAM[n]) as Z_in_GLB:
21                      for k in range(K):
22                          Z_in_GLB[0] += A_in_DRAM[k] * B_in_DRAM[k,n]
```

New memory level

Adding a New Memory Level “GLB”

```
11  # Memory levels
12  DRAM = MemoryLevel("DRAM")
13  GLB = MemoryLevel("GLB")
14
15  # A loop nest that performs Z = A x B
16  with DRAM.allocate(A) as A_in_DRAM:
17      with DRAM.allocate(B) as B_in_DRAM:
18          with DRAM.allocate(Z) as Z_in_DRAM:
19              for n in range(N):
20                  with GLB.allocate(Z_in_DRAM[n]) as Z_in_GLB:
21                      for k in range(K):
22                          Z_in_GLB[0] += A_in_DRAM[k] * B_in_DRAM[k,n]
```

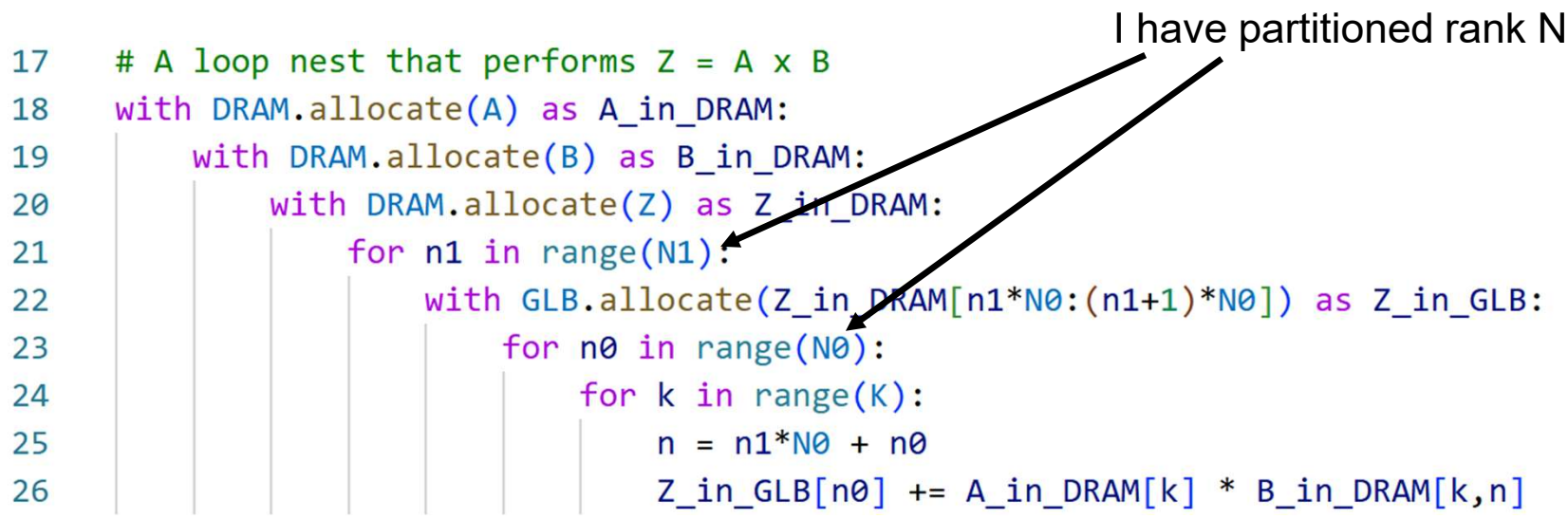
Space is allocated in the GLB

Reference to Z in GLB

The address of accesses is *local*

The GLB keeps only one element of Z, which is Z[n]

Allocating Tiles



Allocating Tiles

```
17  # A loop nest that performs Z = A x B
18  with DRAM.allocate(A) as A_in_DRAM:
19      with DRAM.allocate(B) as B_in_DRAM:
20          with DRAM.allocate(Z) as Z_in_DRAM:
21              for n1 in range(N1):
22                  with GLB.allocate(Z_in_DRAM[n1*N0:(n1+1)*N0]) as Z_in_GLB:
23                      for n0 in range(N0):
24                          for k in range(K):
25                              n = n1*N0 + n0
26                              Z_in_GLB[n0] += A_in_DRAM[k] * B_in_DRAM[k,n]
```

The slice notation is used to get a *tile* of Z from DRAM

The address of accesses is *local*

The GLB keeps a tile of N0 elements