GEMM Opt. and Conv.

王振华，April 2019

Author

王振华 Zhenhua WANG （aka. 黎明灰烬 Jackwish）

System design and performance optimization wrt. machine learning system, virtual machine and computer architecture.

Mail: i@jackwish.net

Blog: https://jackwish.net

GEMM Optimization

QNNPACK Implementation

Pointwise Convolution

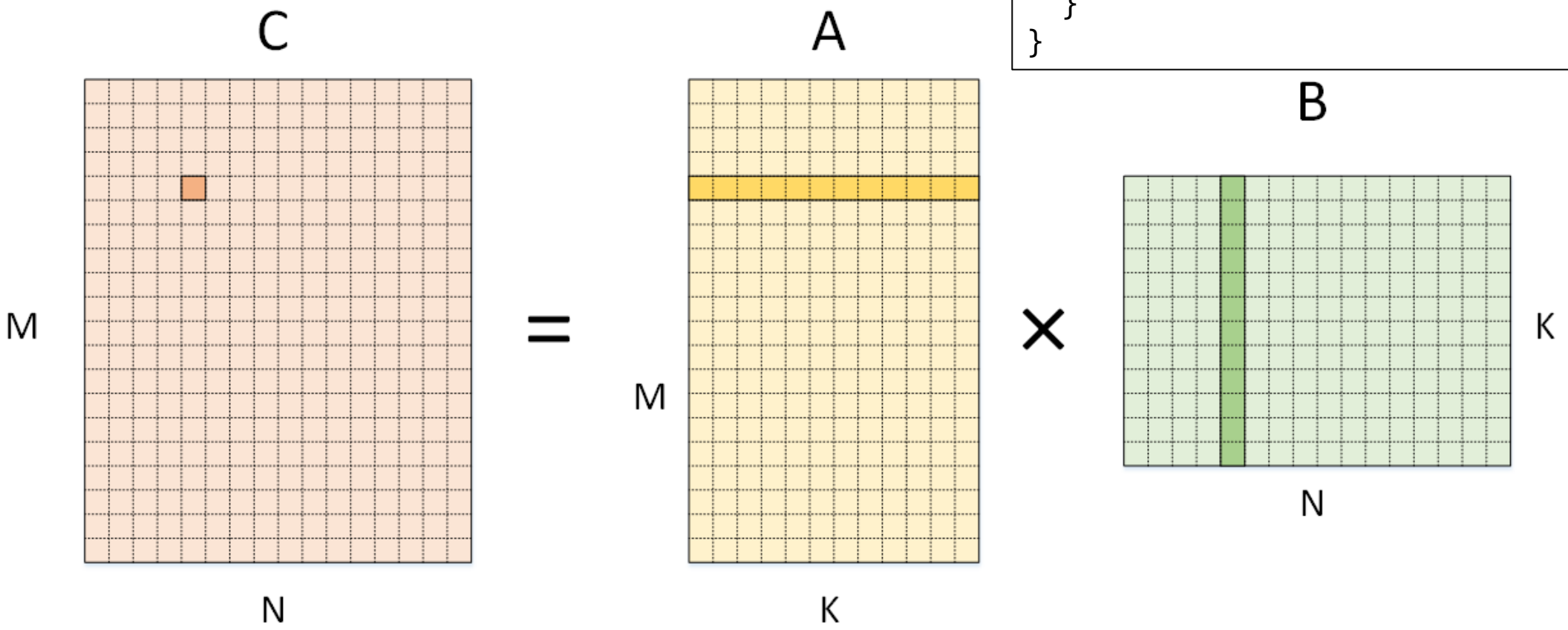
GEMM Optimization

QNNPACK Implementation

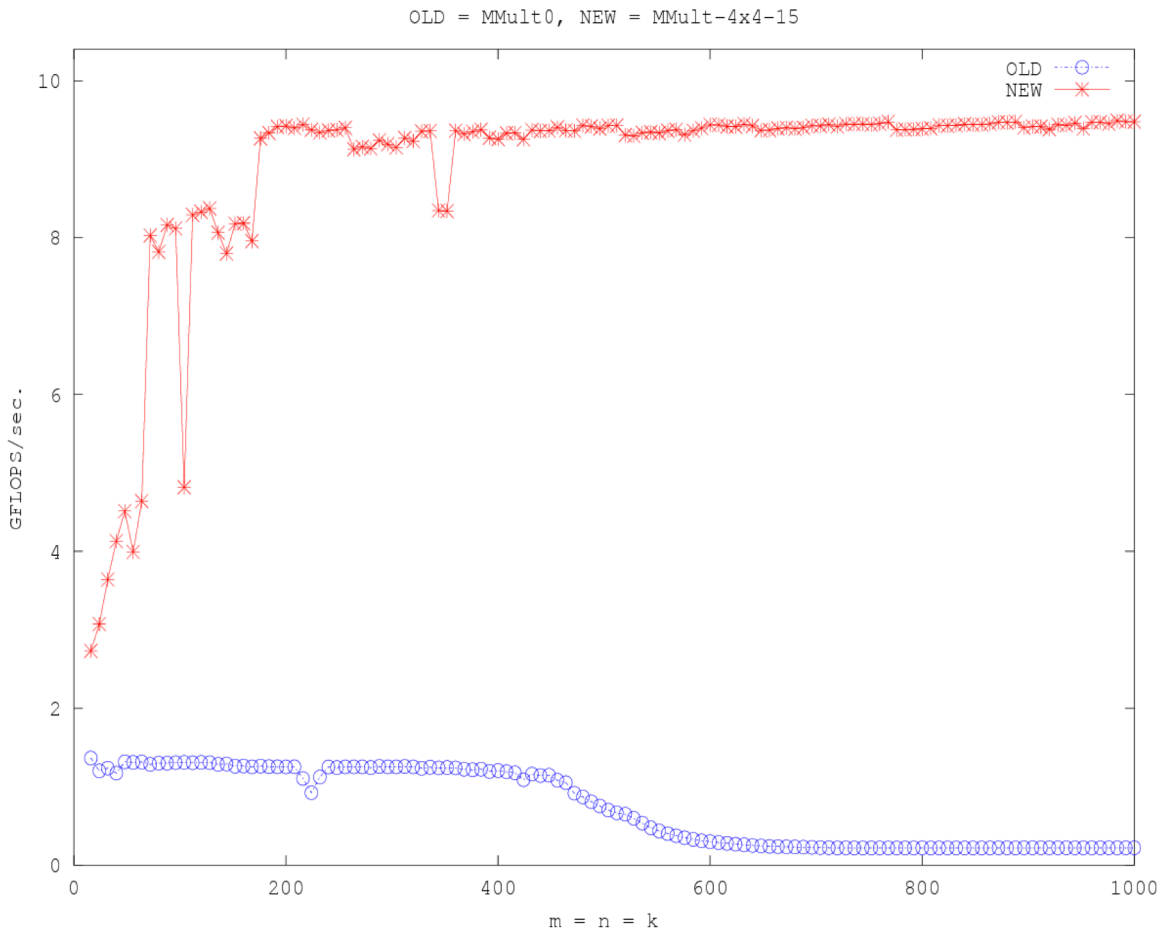
Pointwise Convolution

Matrix-matrix Multiplication

```
for (m in [0, M), step 1) {
  for (n in [0, N), step 1) {
    C[m][n] = 0;
    for (k in [0, K), step 1) {
      C[m][n] += A[m][k] * B[k][n];
    }
  }
}
```



How To Optimize Gemm

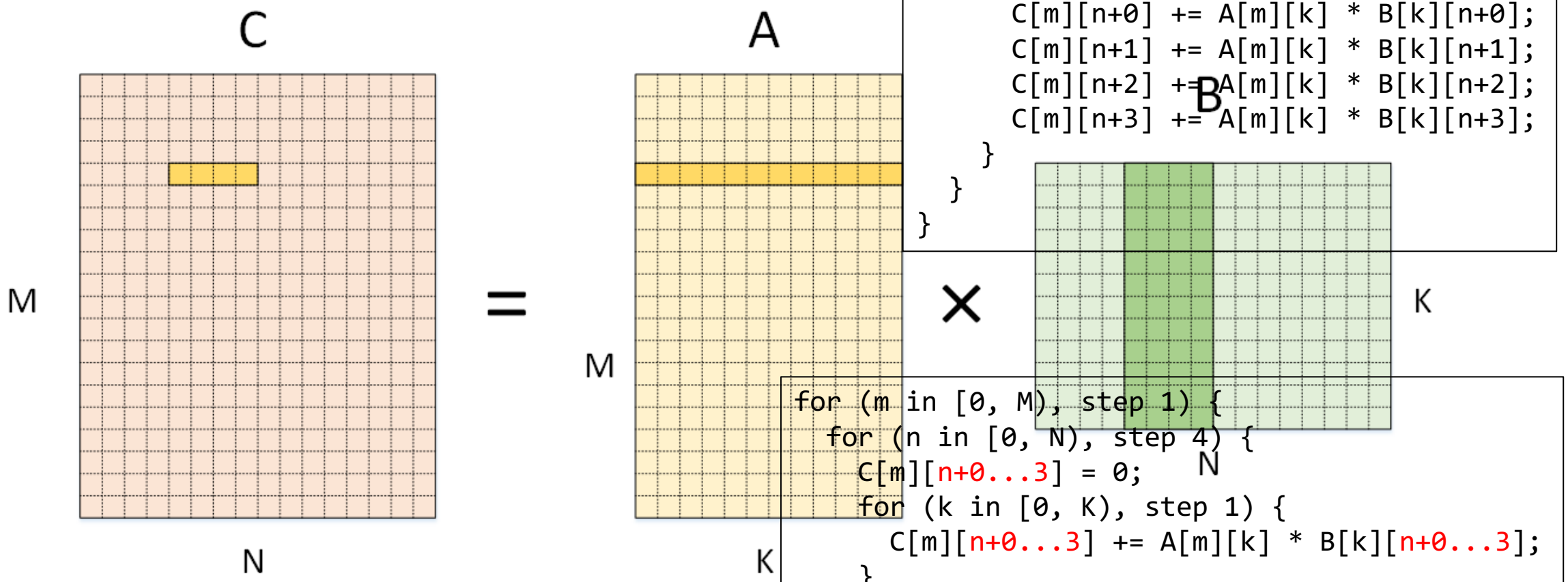


- Compute 1x4/4x4 – reuse data
  - Use register rather than cache
  - Unroll (with a factor)
  - Vector load/store/arithmetic
  - Use pointer to address matrix*
  - Indirect addressing*
- Block matrix (large)
- Pack into contiguous memory

cite: https://github.com/flame/how-to-optimize-gemm/wiki

* Address later in QNNPACK conv

Compute 1x4 Result (split, unroll)

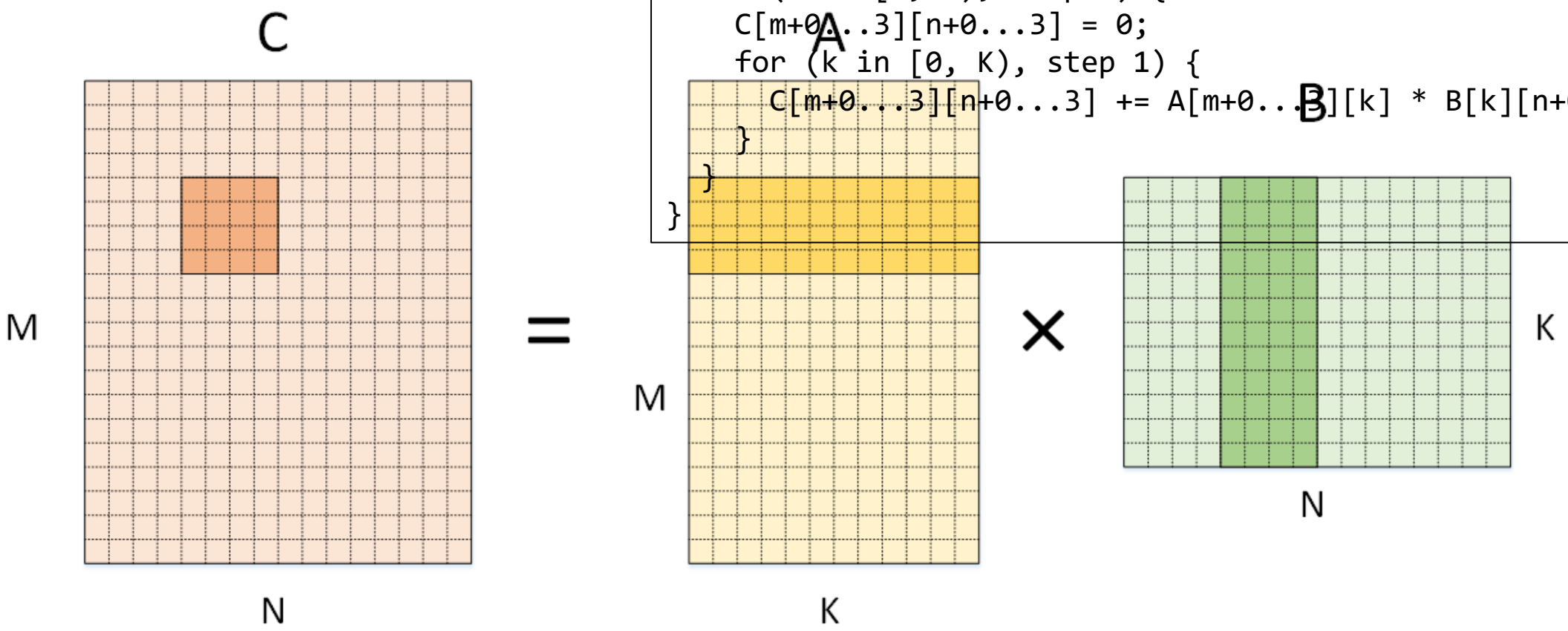



```
for (m in [0, M), step 1) {
  for (n in [0, N), step 4) {
    C[m][n+0] = 0;
    C[m][n+1] = 0;
    C[m][n+2] = 0;
    C[m][n+3] = 0;
    for (k in [0, K), step 1) {
      C[m][n+0] += A[m][k] * B[k][n+0];
      C[m][n+1] += A[m][k] * B[k][n+1];
      C[m][n+2] += A[m][k] * B[k][n+2];
      C[m][n+3] += A[m][k] * B[k][n+3];
    }
  }
}
```

```
for (m in [0, M), step 1) {
  for (n in [0, N), step 4) {
    C[m][n+0...3] = 0;
    for (k in [0, K), step 1) {
      C[m][n+0...3] += A[m][k] * B[k][n+0...3];
    }
  }
}
```

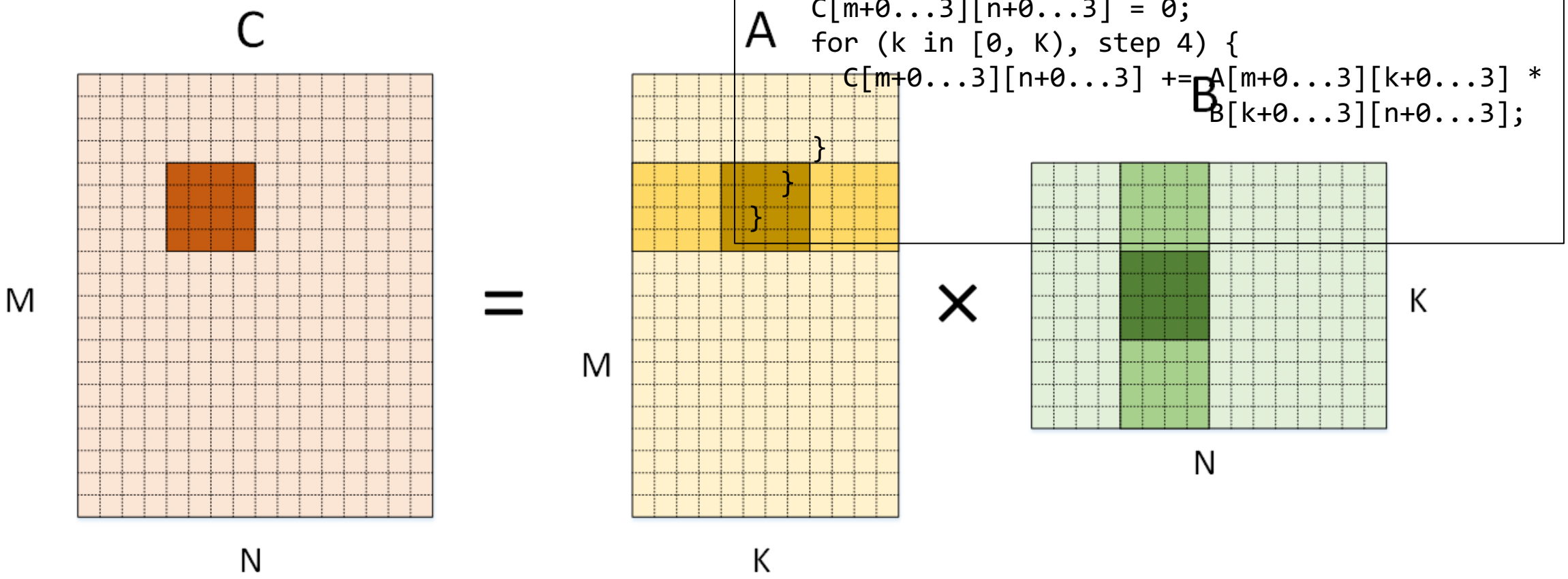
Compute 4x4 Result (tile, unroll)

```
for (m in [0, M), step 4) {
  for (n in [0, N), step 4) {
    C[m+0...3][n+0...3] = 0;
    for (k in [0, K), step 1) {
      C[m+0...3][n+0...3] += A[m+0...3][k] * B[k][n+0...3];
    }
  }
}
```

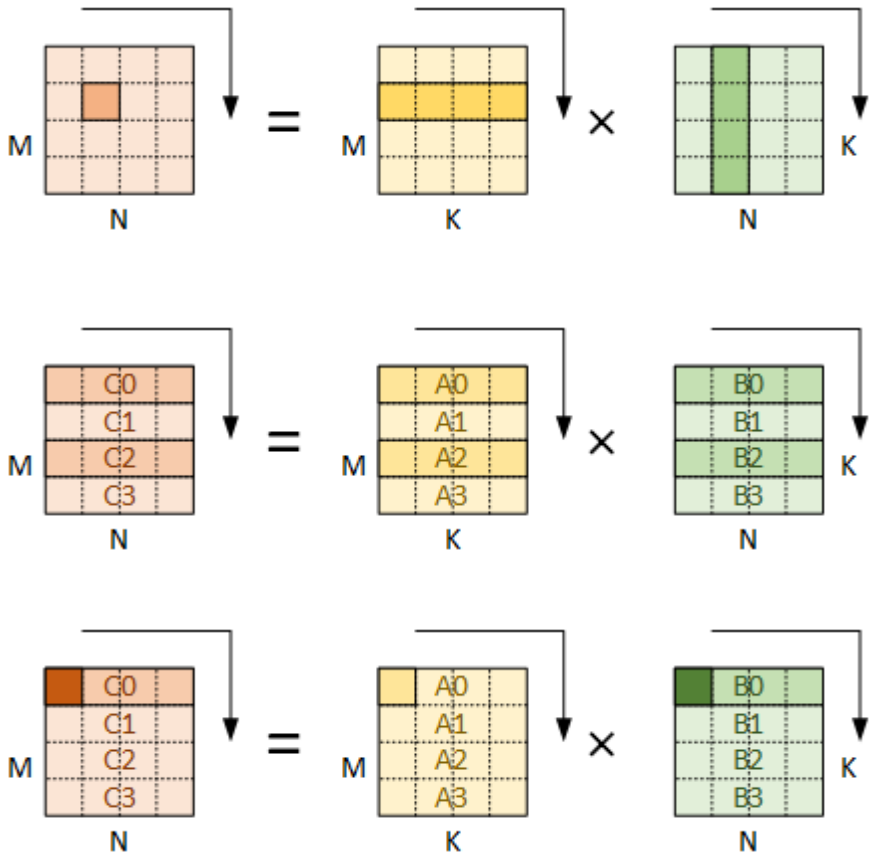


(4x4)x(4x4) reduction of 4x4 output

```
for (m in [0, M), step 4) {
  for (n in [0, N), step 4) {
    C[m+0...3][n+0...3] = 0;
    for (k in [0, K), step 4) {
      C[m+0...3][n+0...3] += A[m+0...3][k+0...3] *
                             B[k+0...3][n+0...3];
    }
  }
}
```



(4x4)x(4x4) Vector load/store/arithmetic

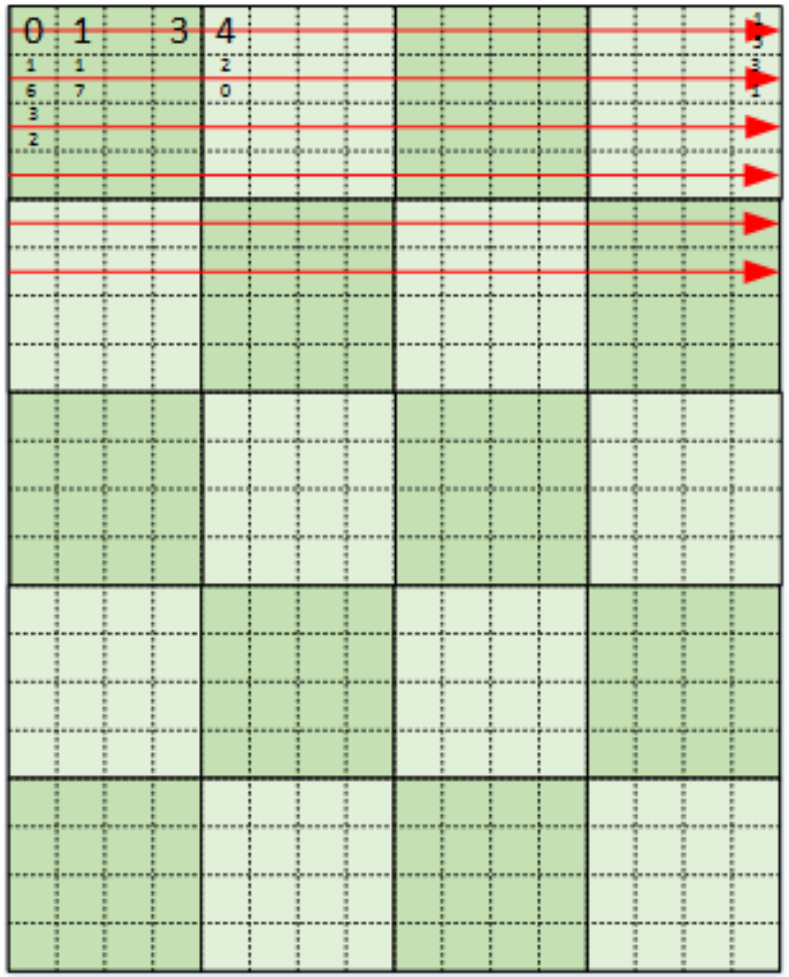


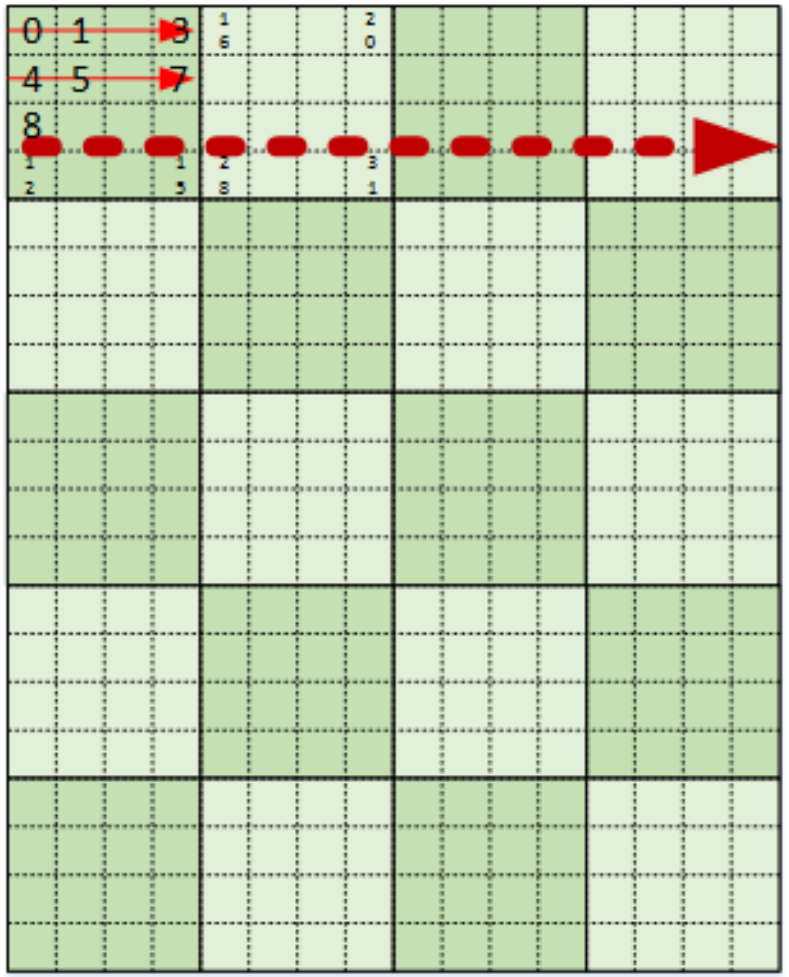

```
// C0 in detail

C0[0] += A0[0] * B0[0]
C0[0] += A0[1] * B1[0]
C0[0] += A0[2] * B2[0]
C0[0] += A0[3] * B3[0]

C0[1] += A0[0] * B0[1]
C0[1] += A0[1] * B1[1]
C0[1] += A0[2] * B2[1]
C0[1] += A0[3] * B3[1]

C0[2] += A0[0] * B0[2]
C0[2] += A0[1] * B1[2]
C0[2] += A0[2] * B2[2]
C0[2] += A0[3] * B3[2]

C0[3] += A0[0] * B0[3]
C0[3] += A0[1] * B1[3]
C0[3] += A0[2] * B2[3]
C0[3] += A0[3] * B3[3]
```

```
// C0 scheduled

C0[0] += A0[0] * B0[0]
C0[1] += A0[0] * B0[1]
C0[2] += A0[0] * B0[2]
C0[3] += A0[0] * B0[3]

C0[0] += A0[1] * B1[0]
C0[1] += A0[1] * B1[1]
C0[2] += A0[1] * B1[2]
C0[3] += A0[1] * B1[3]

C0[0] += A0[2] * B2[0]
C0[1] += A0[2] * B2[1]
C0[2] += A0[2] * B2[2]
C0[3] += A0[2] * B2[3]

C0[0] += A0[3] * B3[0]
C0[1] += A0[3] * B3[1]
C0[2] += A0[3] * B3[2]
C0[3] += A0[3] * B3[3]
```

```
// (4x4)*(4x4)

Load C0-C3
Load A0-A3
Load B0
C0 += A0[0] * B0
C1 += A1[0] * B0
C2 += A2[0] * B0
C3 += A3[0] * B0
Load B1
C0 += A0[1] * B1
C1 += A1[1] * B1
C2 += A2[1] * B1
C3 += A3[1] * B1
...
Load B3
C0 += A0[3] * B3
C1 += A1[3] * B3
C2 += A2[3] * B3
C3 += A3[3] * B3
Store C0-C3
```

Pack into Contiguous Memory



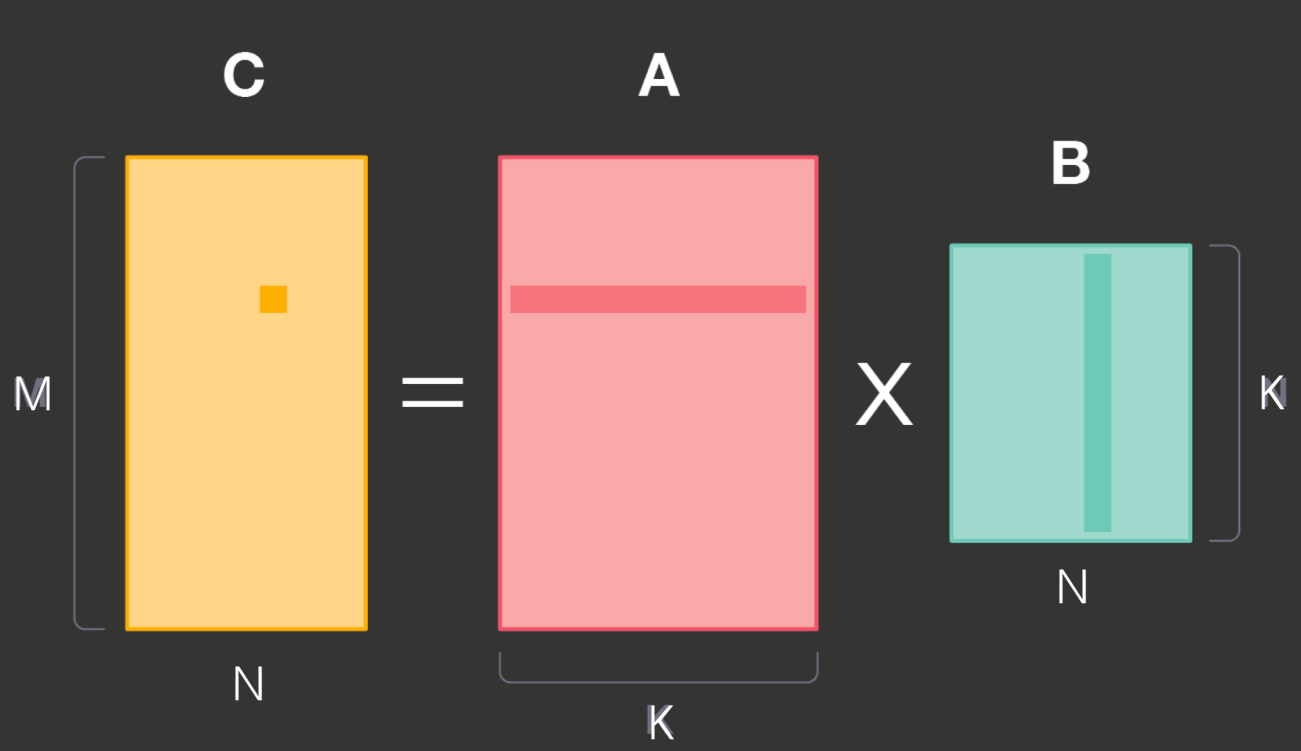


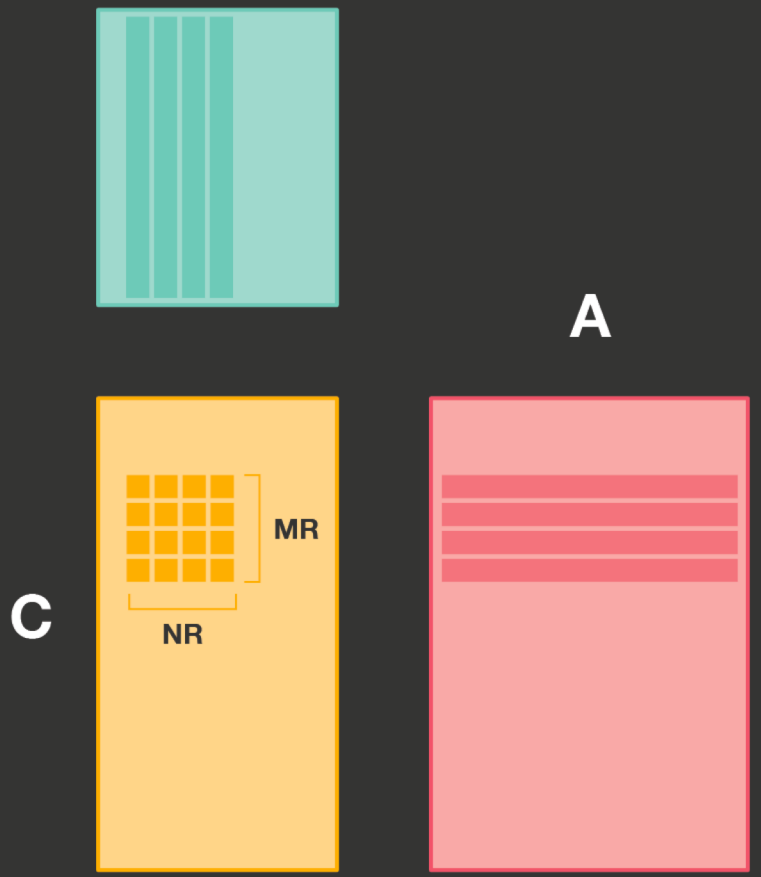
GEMM Optimization

QNNPACK Implementation

Pointwise Convolution

QNNPACK: Matrix-matrix Multiplication



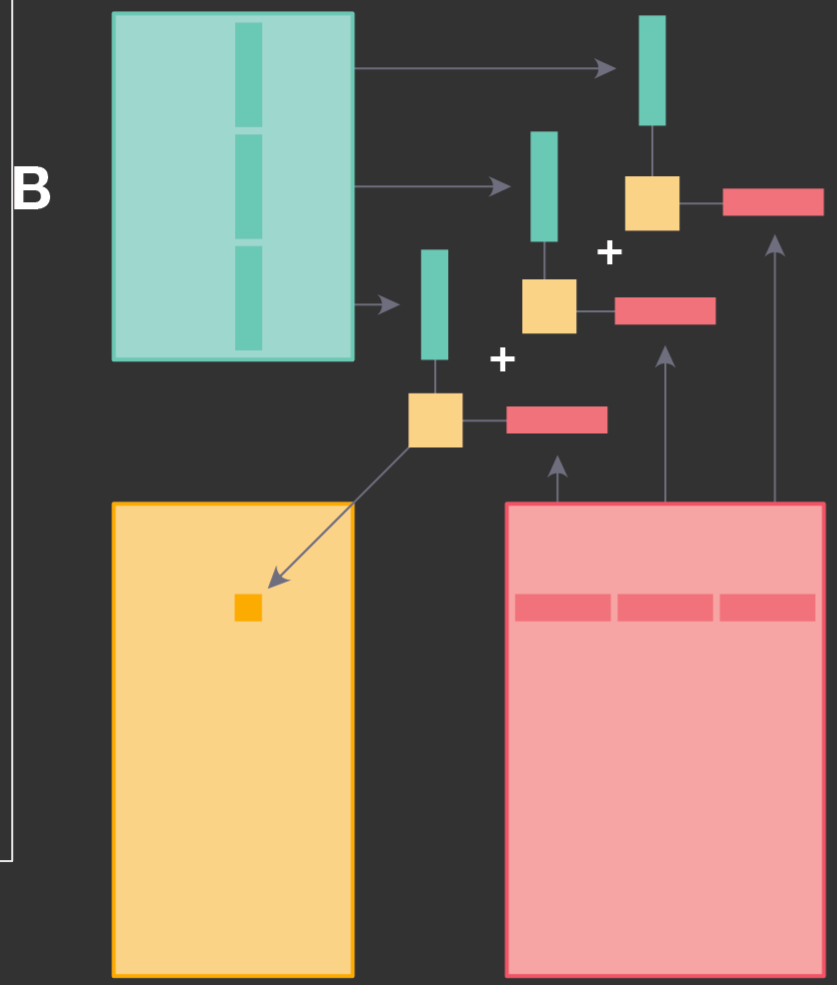


cite: https://code.fb.com/ml-applications/qnnpack/

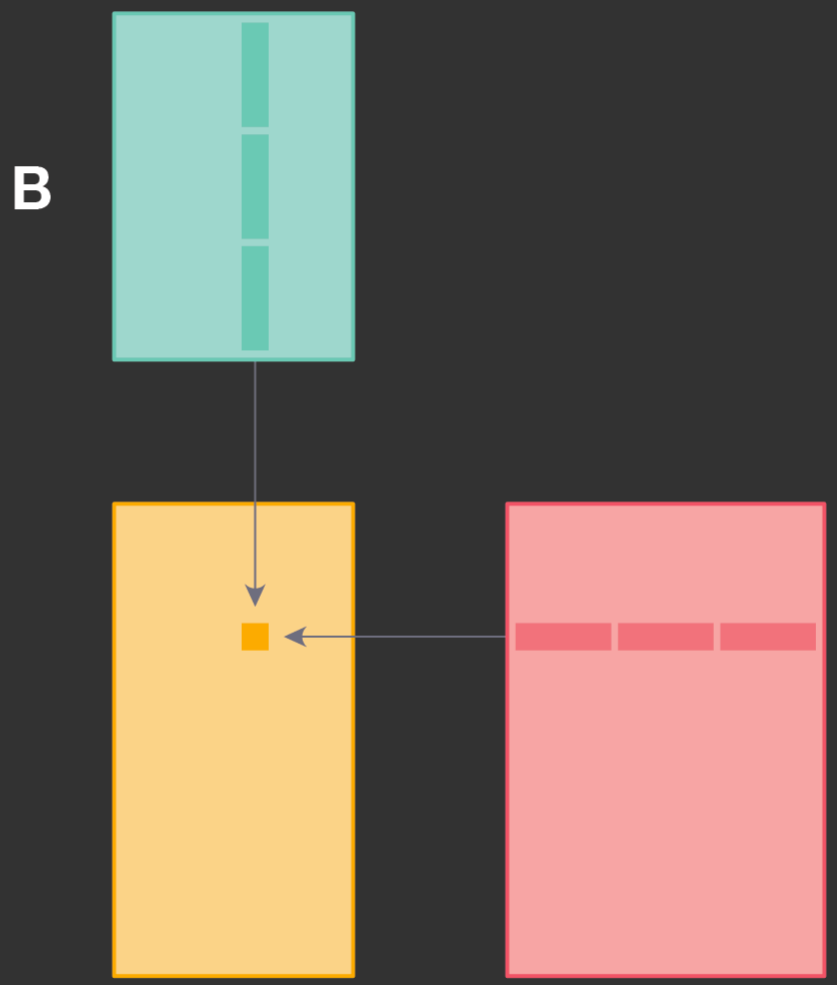
QNNPACK: Compute Along Reduction

Traditional implementation

```
Load C
Load A, B
C += A *B
Store C

Load C
Load A, B
C += A *B
Store C

...

Load C
Load A, B
C += A *B
Store C
```



QNNPACK implementation



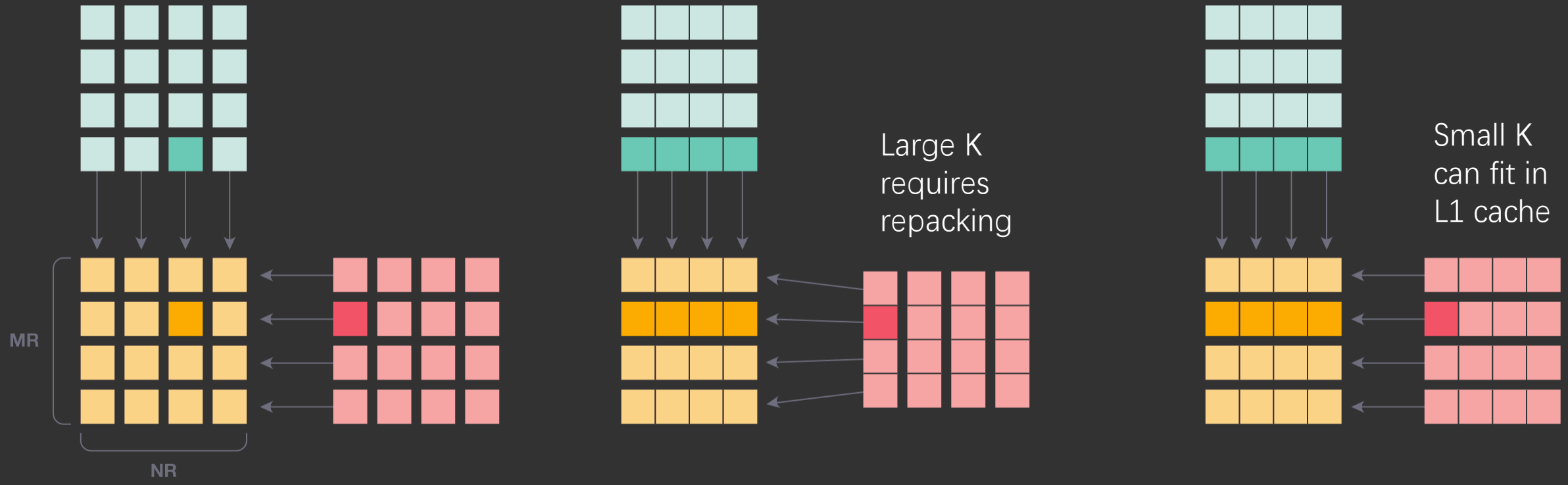

```
Load C

Load A, B
C += A *B

Load A, B
C += A *B

...

Load A, B
C += A *B

Store C
```

QNNPACK: No Input Repacking

PDOT microkernel

Traditional implementation

QNNPACK implementation

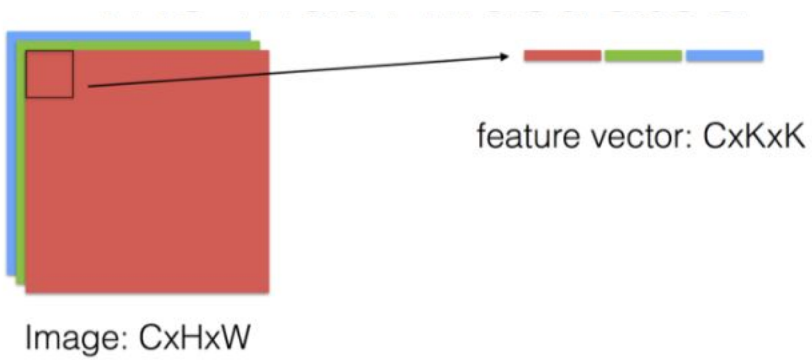


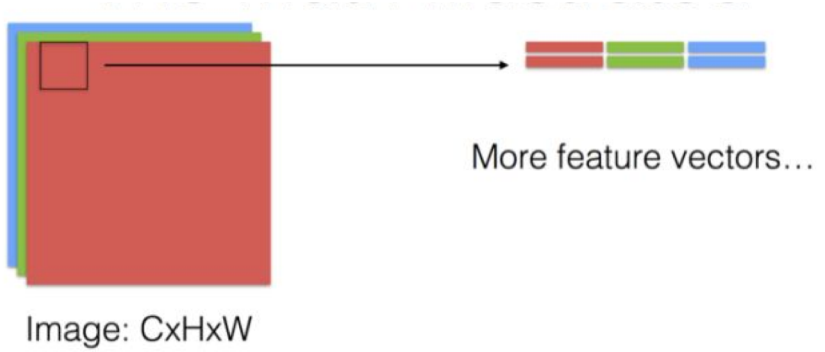
GEMM Optimization

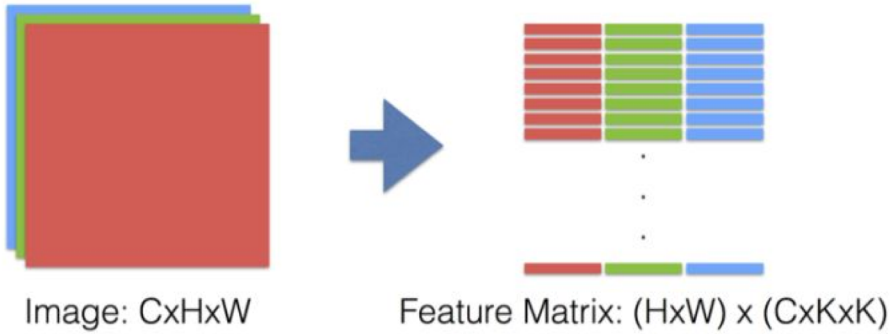
QNNPACK Implementation

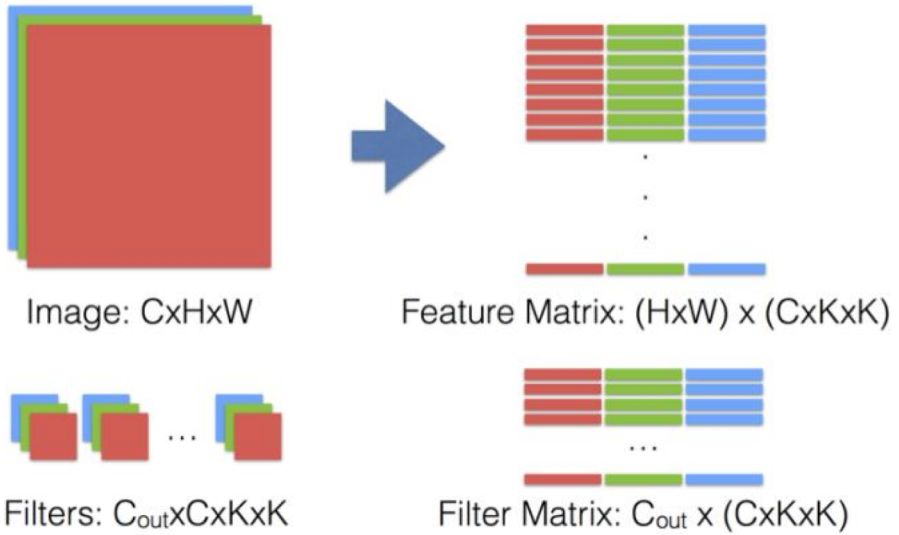
Pointwise Convolution

Convolution as Matrix Multiplication (im2col)



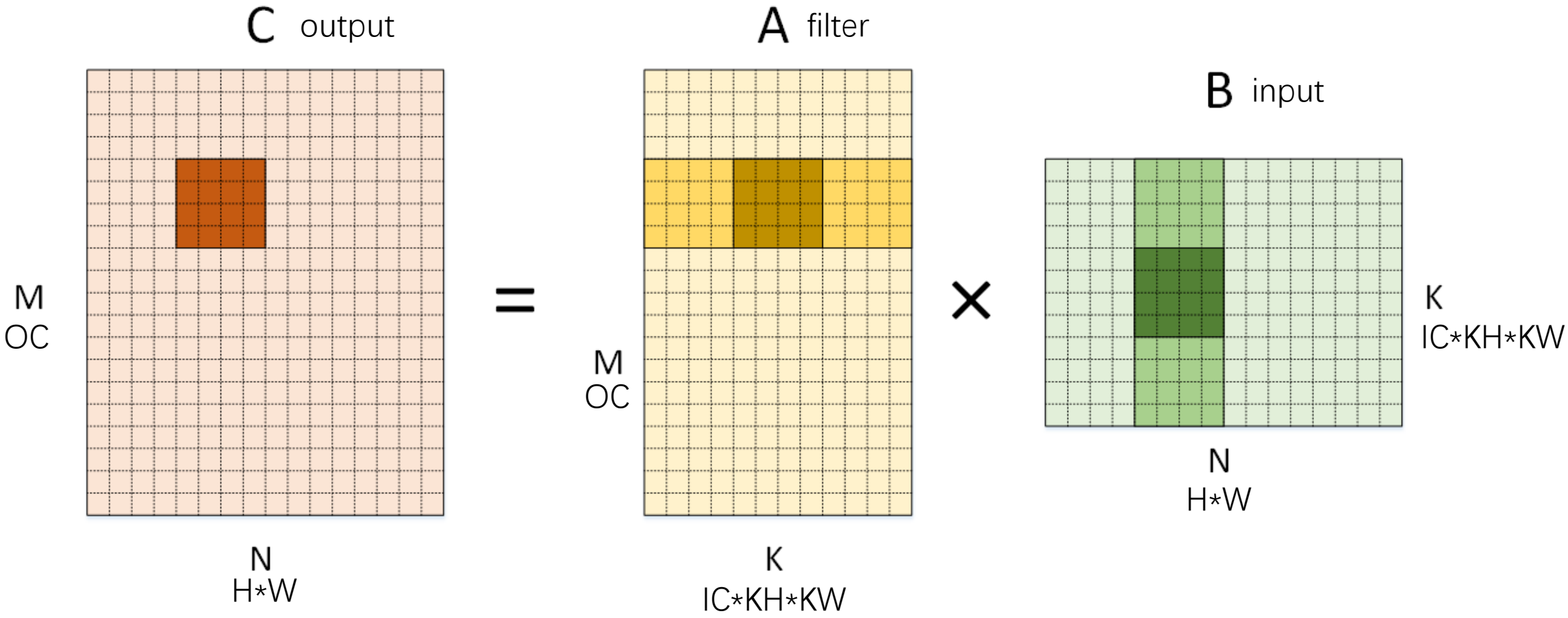




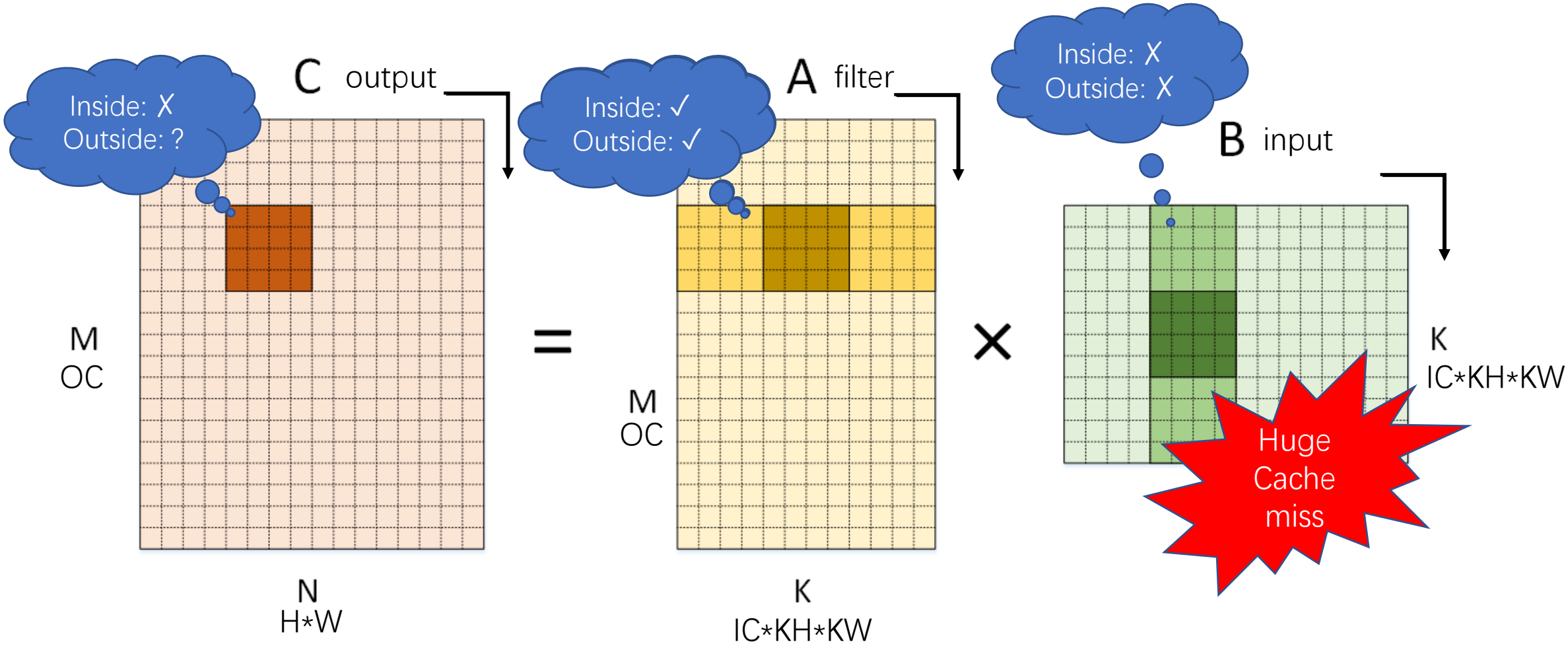


Cite: https://www.zhihu.com/question/28385679/answer/44297845

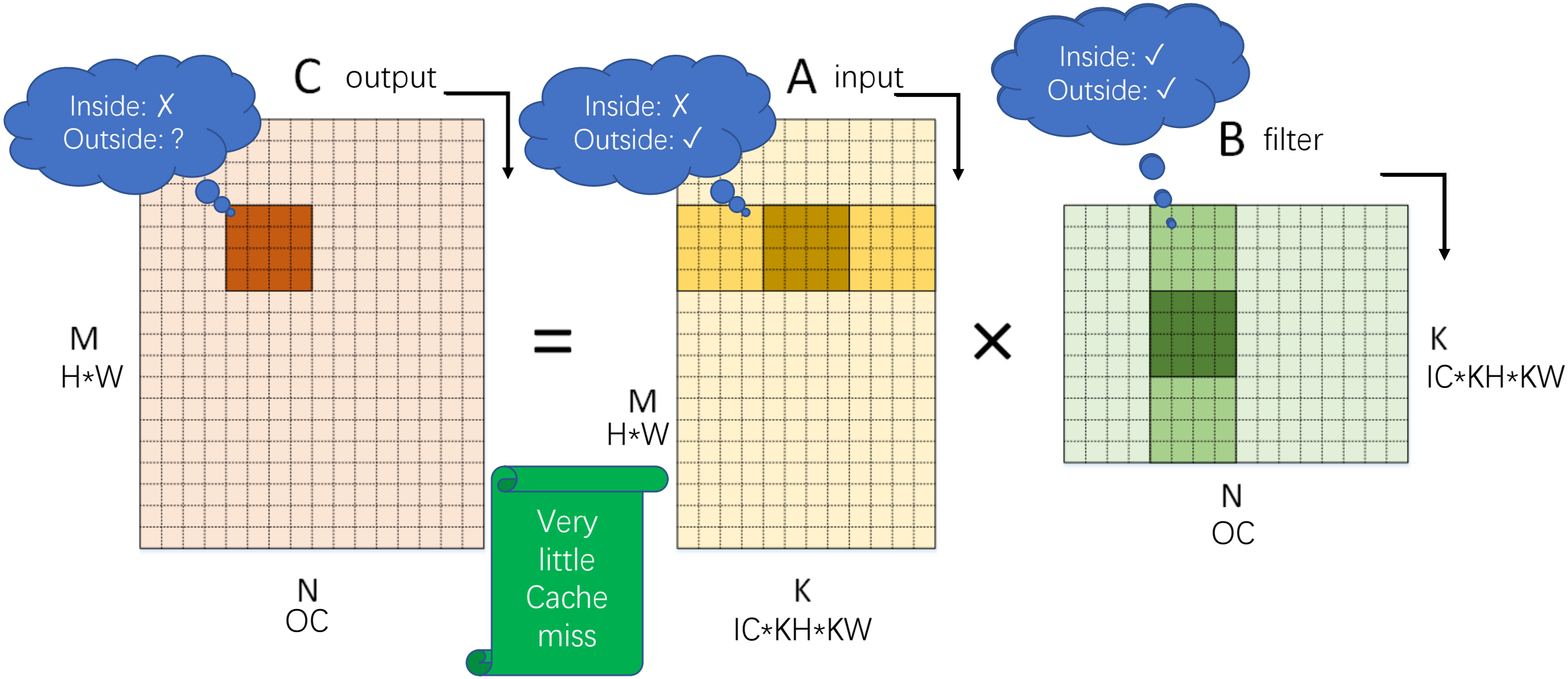
1x1 Conv *equals* Matrix Multiplication (NCHW)



Memory Access Pattern: NCHW



Memory Access Pattern: NHWC



Thanks