Convolution & Optimization

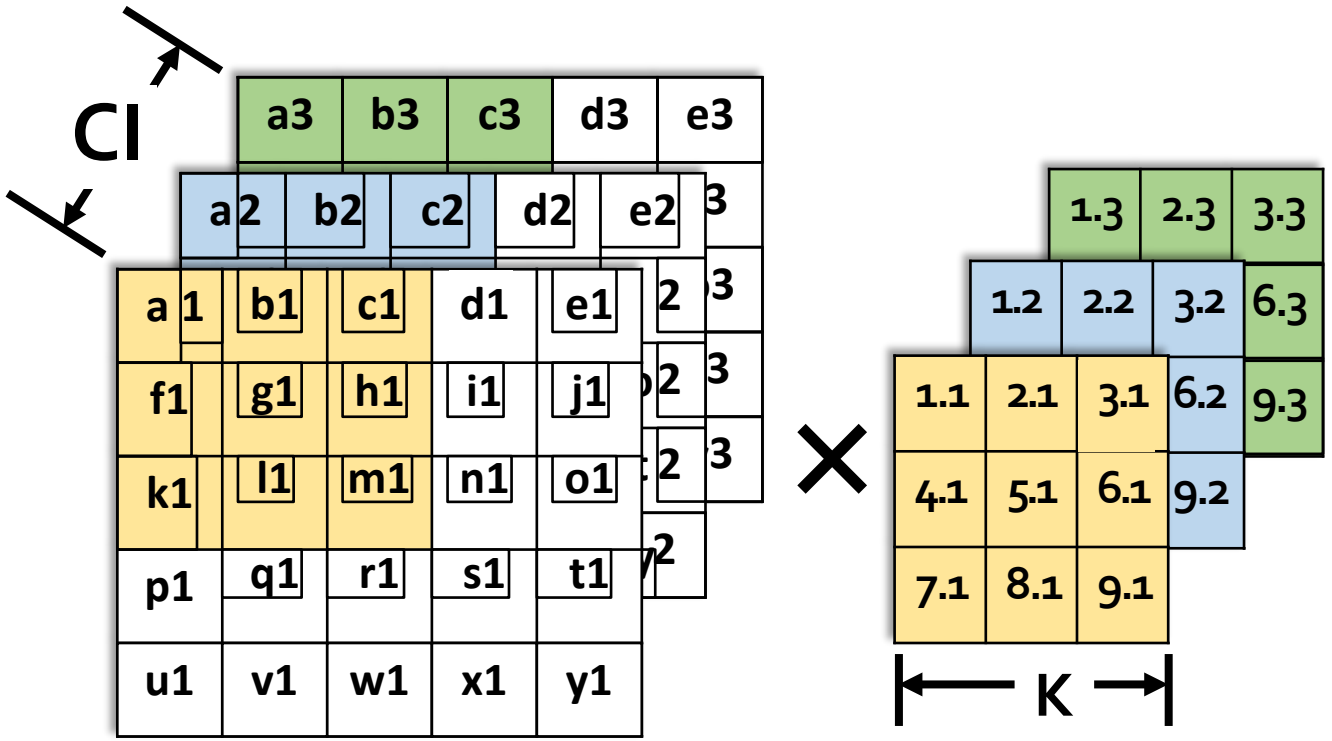
Kang Zhao

zhaokang@bupt.edu.cn

Outline

- **Basic Convolution Computation**
  - **Useful resource:** https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks
- **Data Reuse: Input/Output/Weight stationary**
- **Img2Col and advanced Img2Col**
- **Sparse Matrix Multiplication**

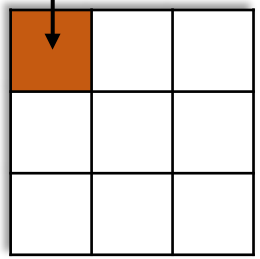
CNN Computation with Multiple Channels



Input

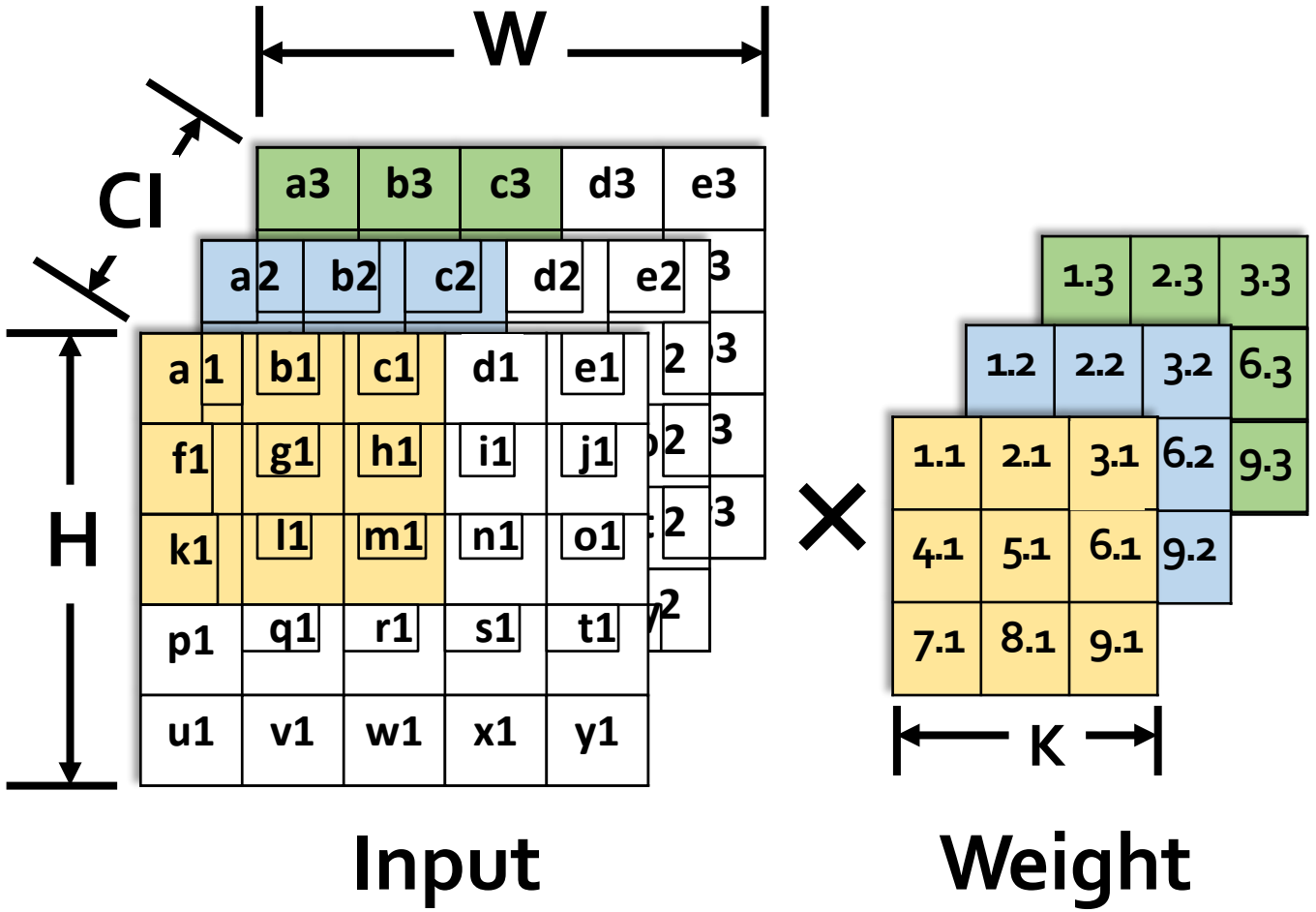
Weight

```
for(int h = 0; h < H-2; h++) {
  for(int w = 0; w < W-2; w++) {
    float sum = 0;
    for(int ci = 0; ci < CI; ci++) {
      for(int m = 0; m < K; m++) {
        for(int n = 0; n < K; n++) {
          sum += A[ci][h+m][w+n] * W[ci][m][n];
        }
      }
    }
    B[h][w] = sum;
  }
}
```



Output

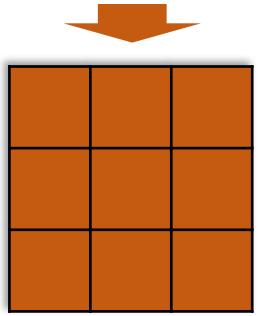
CNN Computation with Multiple Channels



Input

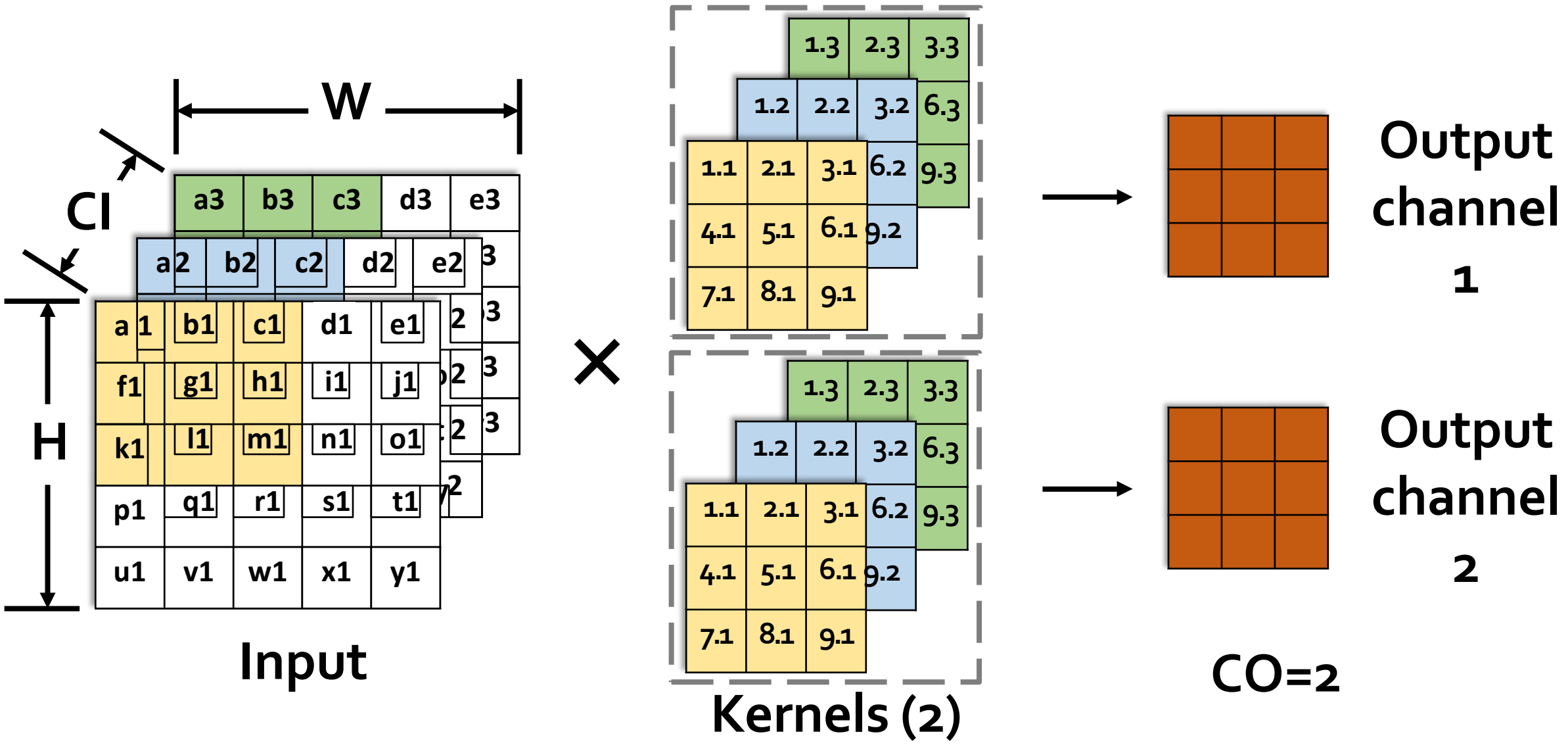
Weight

```
for(int h = 0; h < H-2; h++) {
  for(int w = 0; w < W-2; w++) {
    float sum = 0;
    for(int ci = 0; ci < CI; ci++) {
      for(int m = 0; m < K; m++) {
        for(int n = 0; n < K; n++) {
          sum += A[ci][h+m][w+n] * W[ci][m][n];
        }
      }
    }
    B[h][w] = sum;
  }
}
```

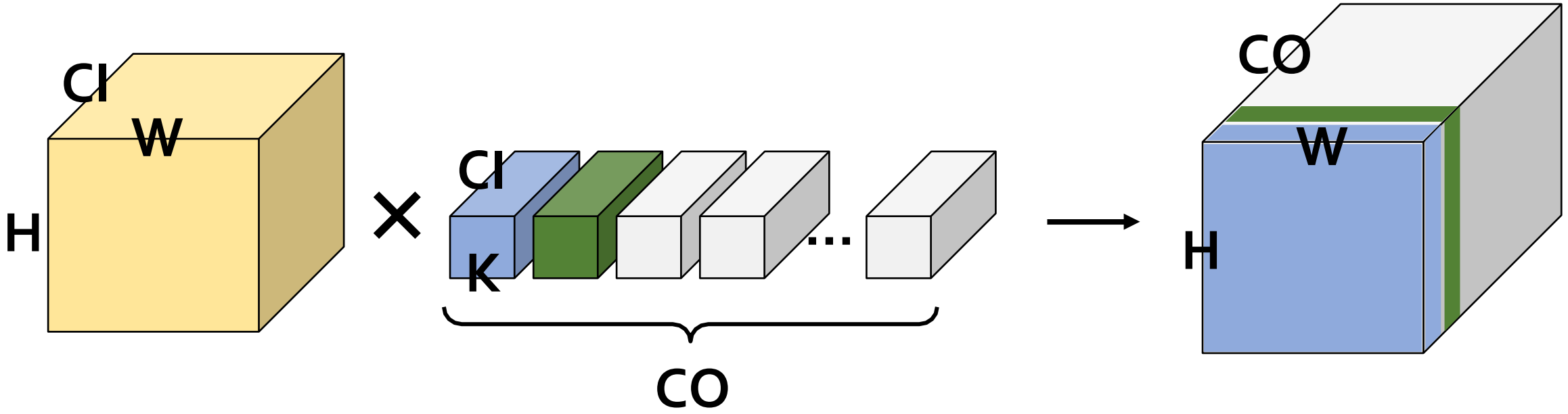


Output

CNN Computation with Multiple Kernels



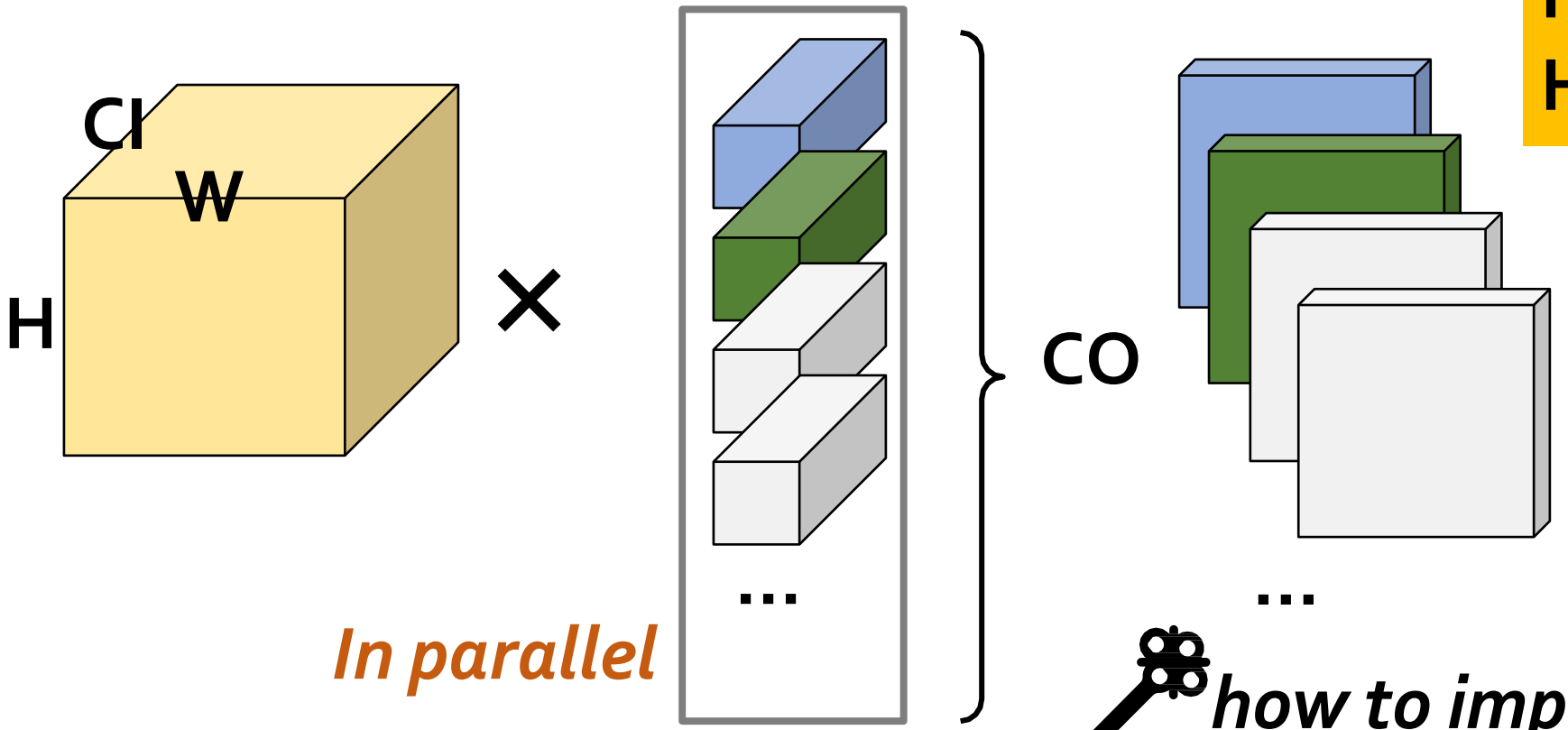
CNN Computation with Multiple Kernels



How do we explore parallelism?

Convolution Acceleration

- **Parallelism 1: across output channel (kernel)**



Naive:
H*W*CI*K*K*CO

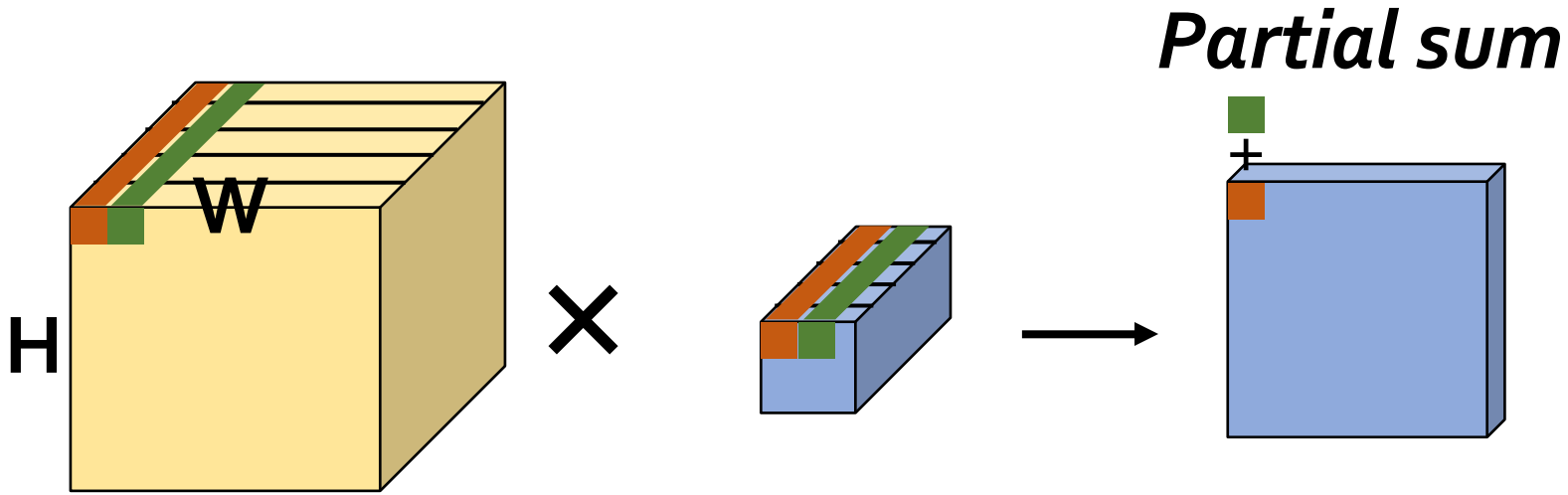
Speedup:
CO

In parallel

how to implement it in HLS?

Convolution Acceleration

- **Parallelism 2: across input channel**

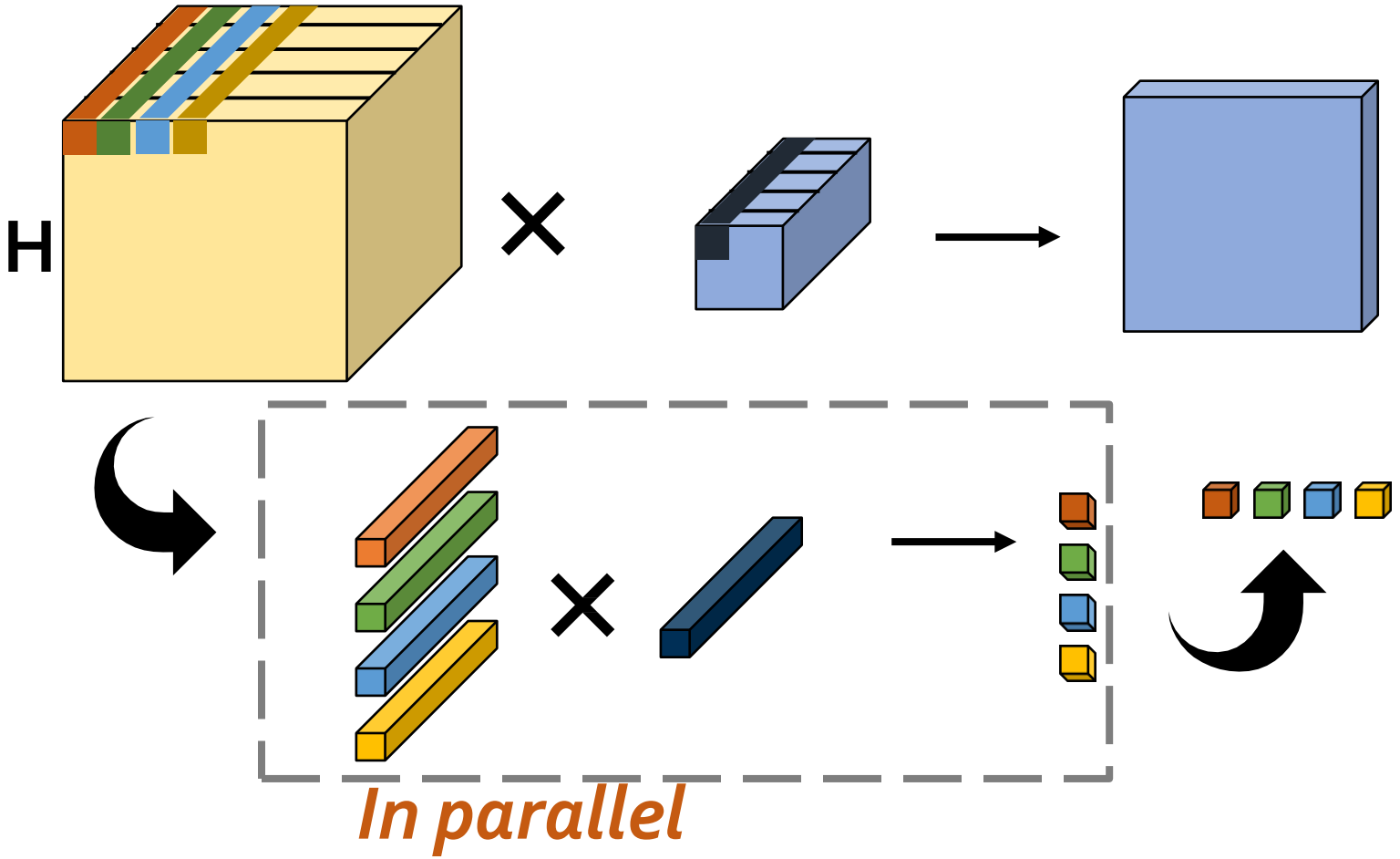


Naive:
H*W*CI*K*K*CO

Speedup:
CO*CI

Convolution Acceleration

- **Parallelism 3: across width dimension**



Naive:
H*W*CI*K*K*CO

Speedup:
CO*CI*W

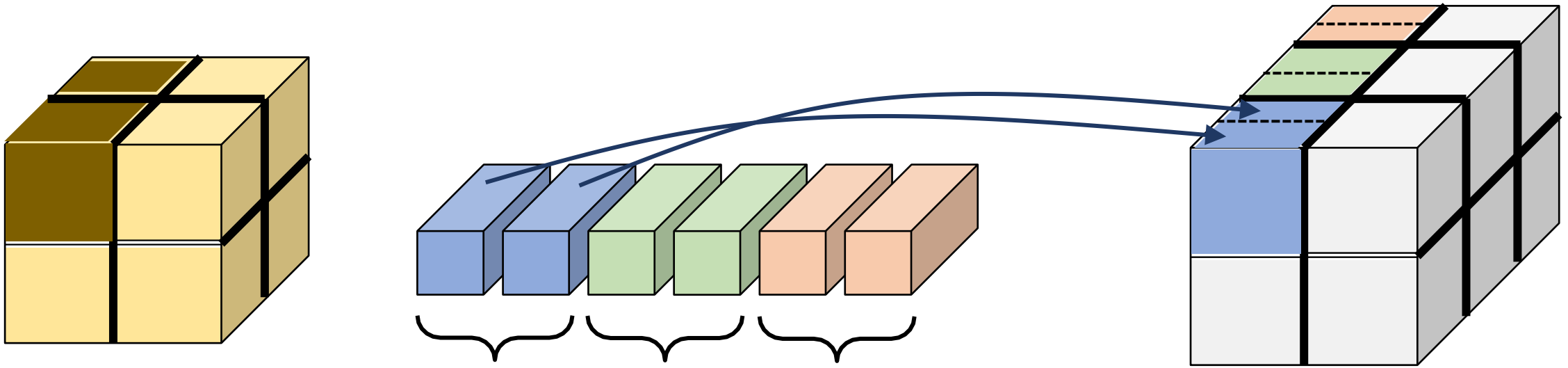
But… pay attention to scalability

Address the Scalability

- **Both memory and computation can explode under aggressive parallelism**
- **Approaches**
  - Try not partition more than 2 dimensions
  - Consider tiling: cut the input image and intermediate feature maps into small cubes
    - Need to **consider tile boundary** – significantly affect the accuracy
  - Parallelize at PE-granularity
- **Read more:**
  - Parallelize at PE level:
    - https://sharc-knowledgebase.netlify.app/articles/pynq/u50/multiple_kernel_execution_demo/
  - Convolution tiling:
    - https://sharc-knowledgebase.netlify.app/articles/cnn/tiling-based_convolution_for_hls/
  - Thanks to Akshay Kamath @ Sharc

Cutting One Layer into Tiles

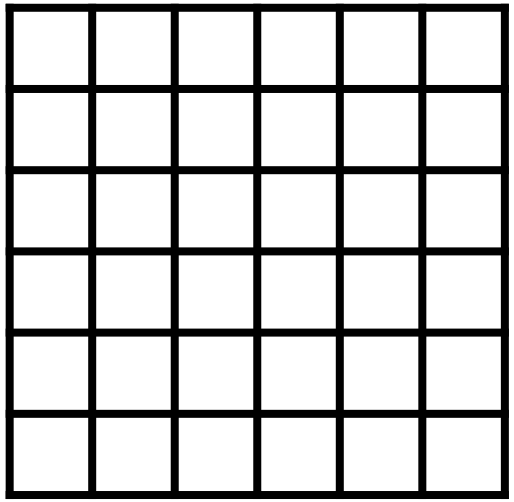
- 8 Input Tiles, 12 Output Tiles



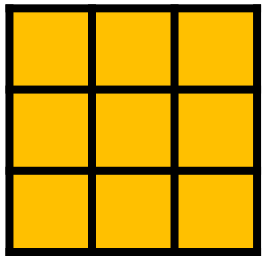
Output Tile 1 Tile 2 Tile 3
(2 channels)

Layer Padding and Tile Padding

- **Layer Padding: zero padding**



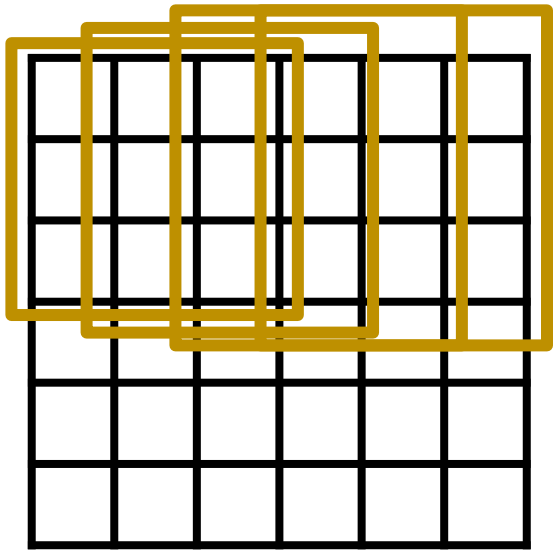
6x6 input



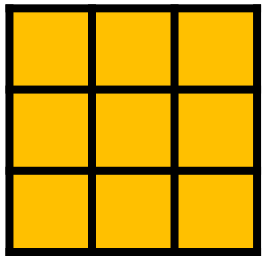
3x3 kernel

Layer Padding and Tile Padding

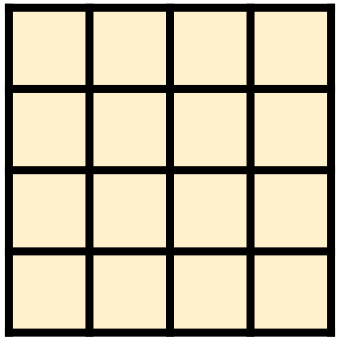
- **Layer Padding: zero padding**



6x6 input



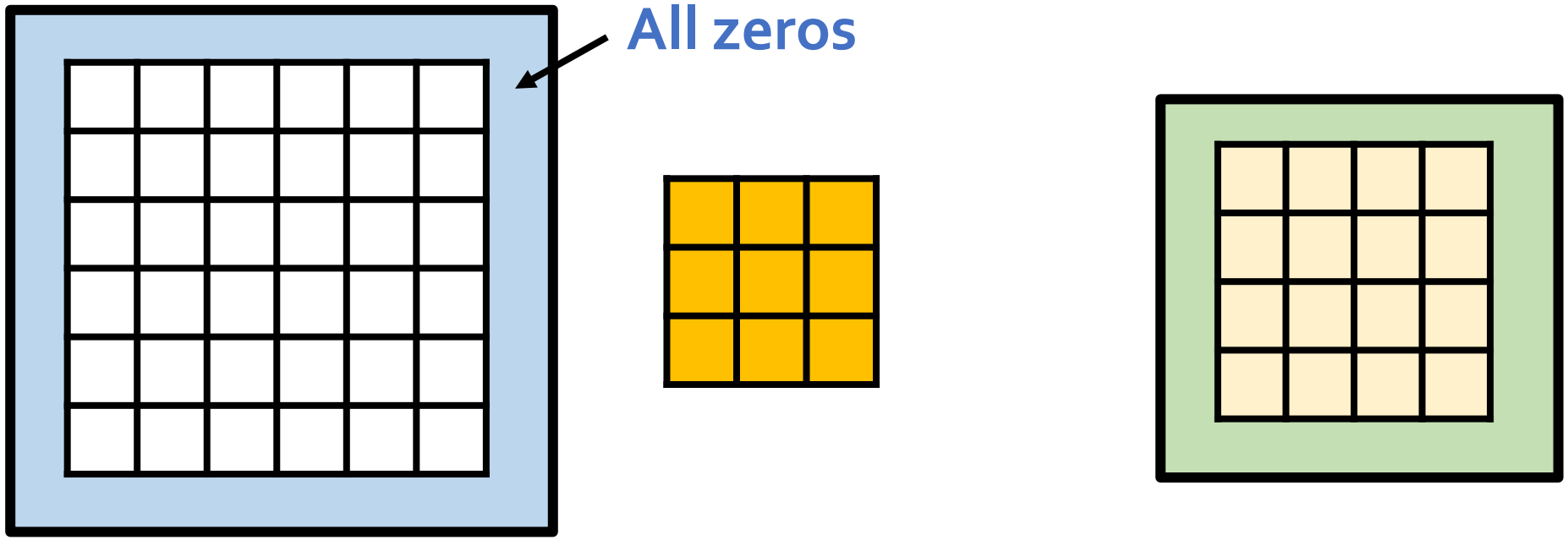
3x3 kernel



4x4 output

Layer Padding and Tile Padding

- **Layer Padding: zero padding**



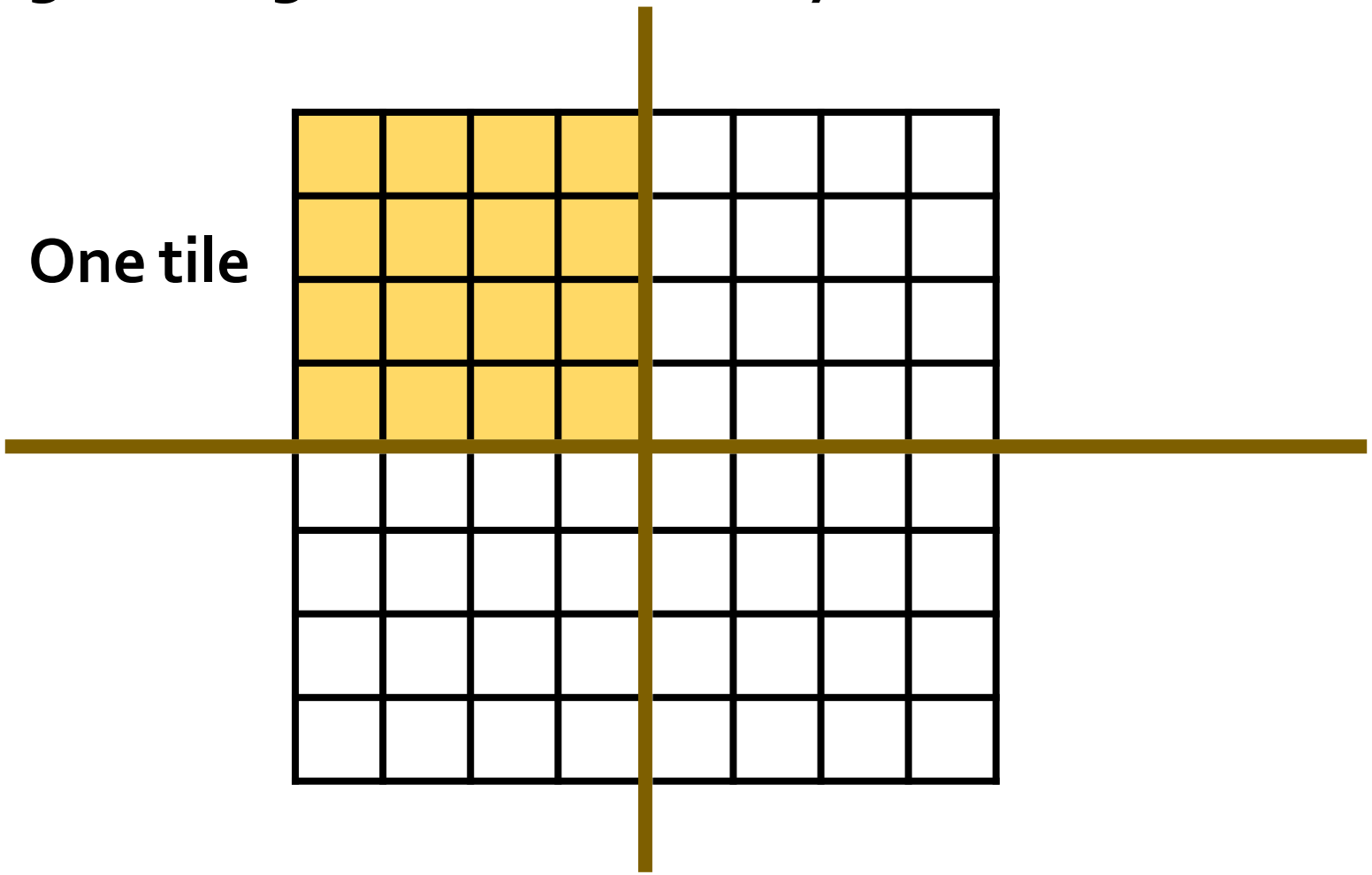
6x6 input
→ 8x8

3x3 kernel

4x4 output
→ 6x6

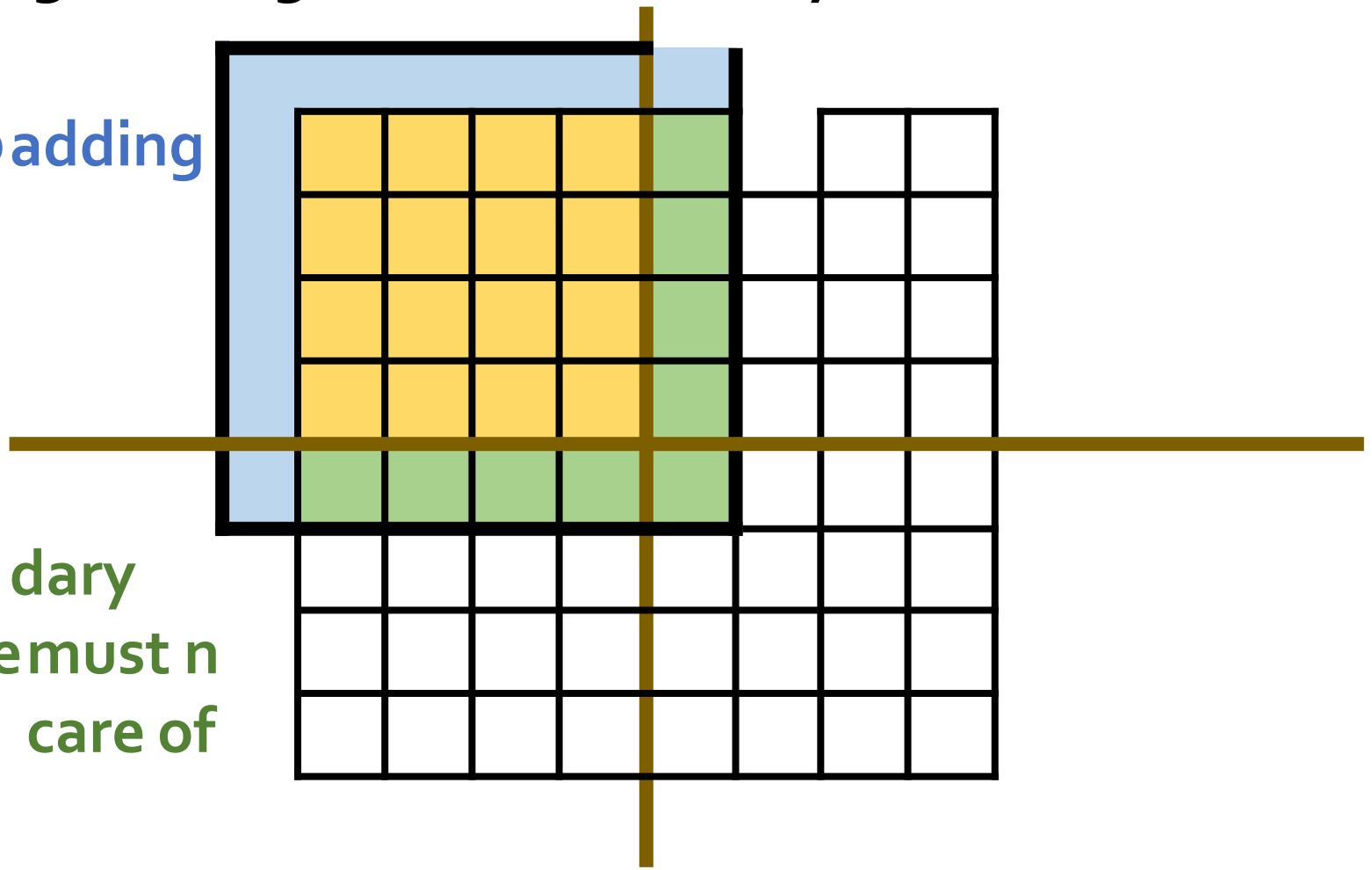
Layer Padding and Tile Padding

- **Tile Padding: dealing with tile boundary**



Layer Padding and Tile Padding

- **Tile Padding: dealing with tile boundary**



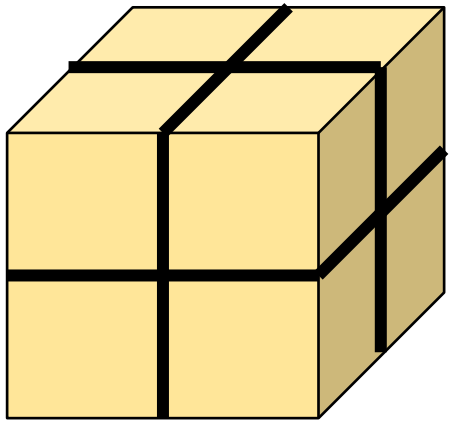
Data Reuse: Optimize Latency & Energy

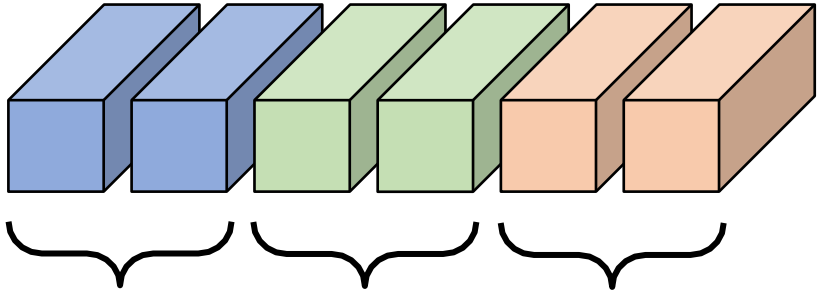
- **Data movement is slow and expensive (energy)**
  - Once we load the data from DRAM to BRAM, let's try to use them as much as possible

- **Ways of reusing data in BRAM:**
  - Input stationary
  - Weight stationary
  - Output stationary
  - Note: the "stationary" definition here is a little bit different as in systolic array

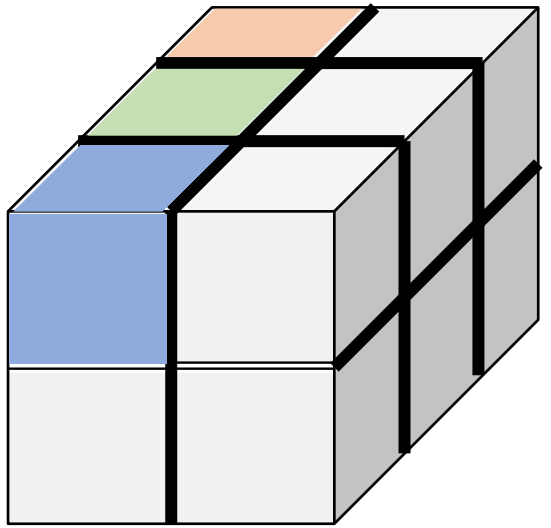
Input Stationary

- Assume only 2 tiles (1-in, 1-out) and a few kernels can fit into BRAM



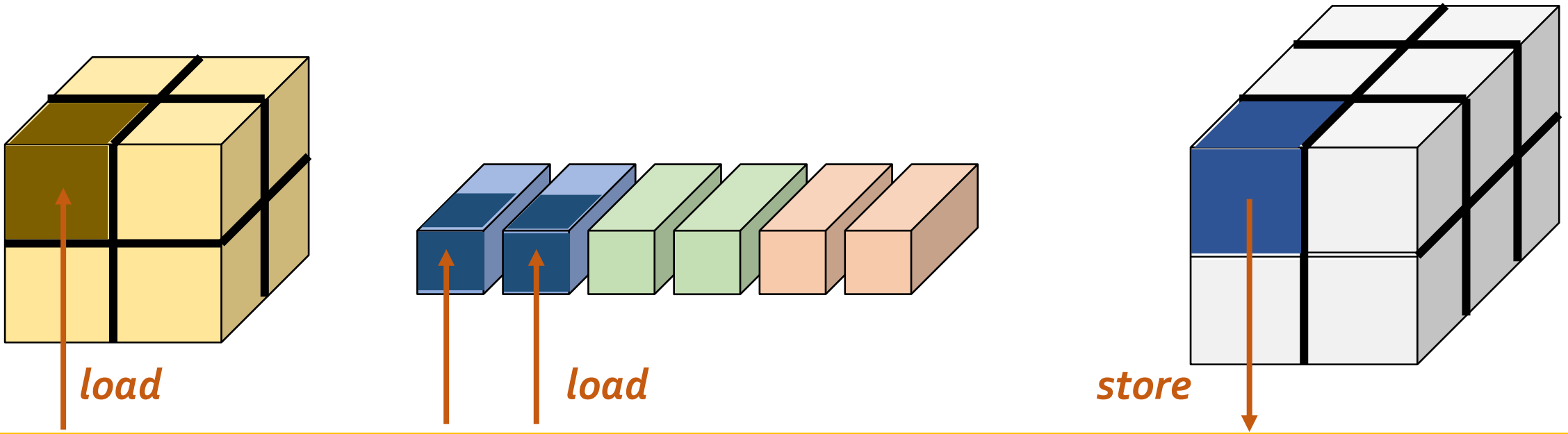


Tile 1 Tile 2 Tile 3



Input Stationary

- Assume only 2 tiles (1-in, 1-out) and a few kernels can fit into BRAM

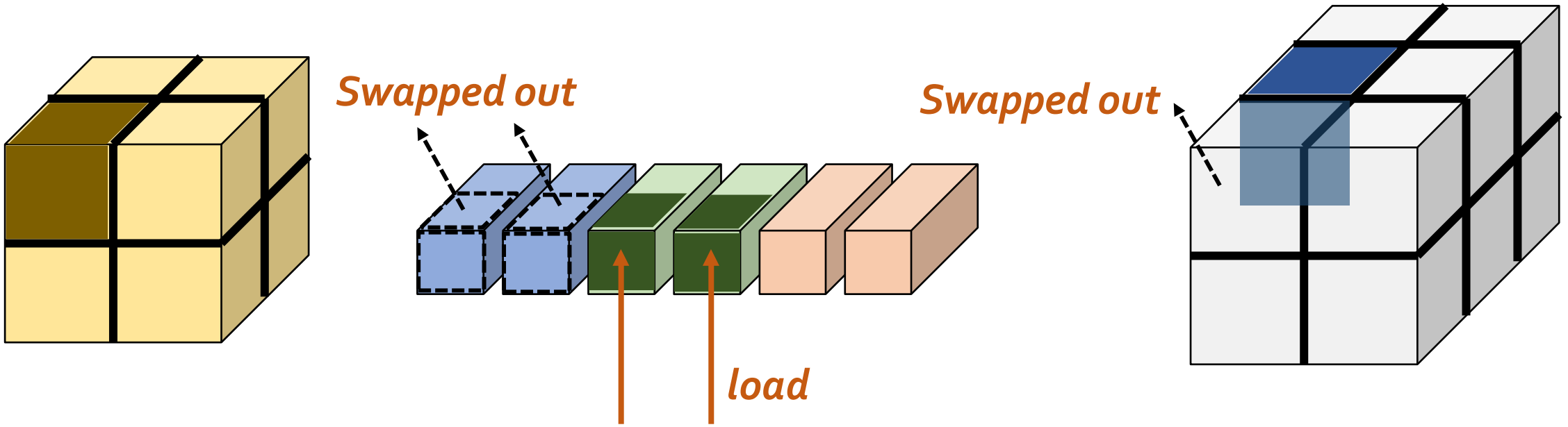


Load/store into/from BRAM and used for computation

Next: which tile(s) to swap out?

Input Stationary

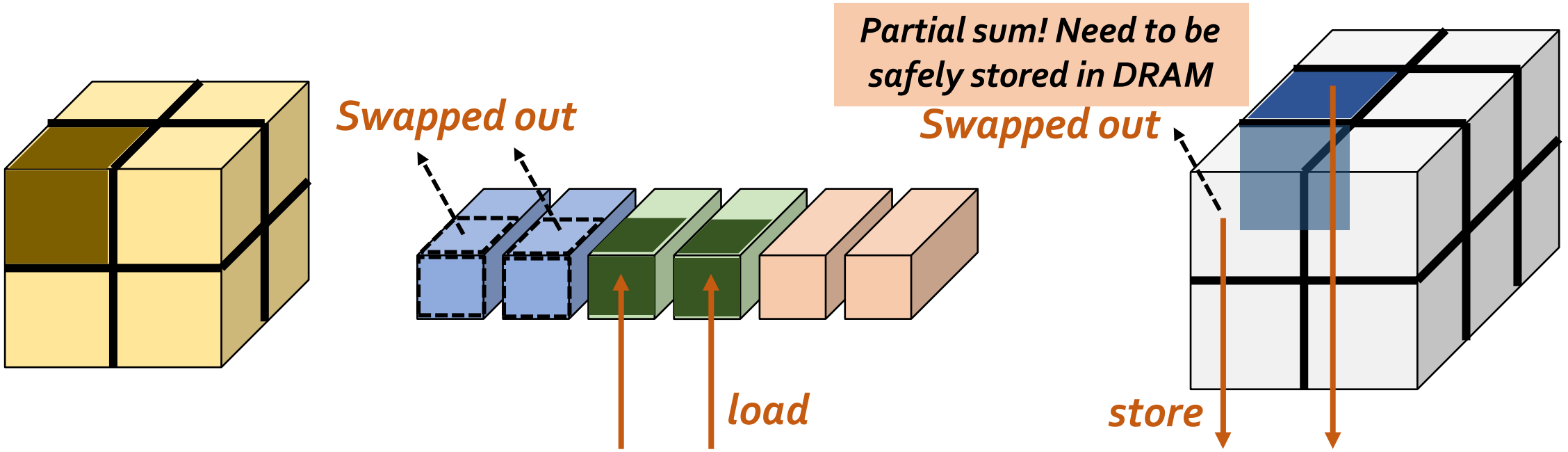
- Try to keep the input tile(s) in BRAM as long as possible



Load/store into/from BRAM and used for computation

Input Stationary

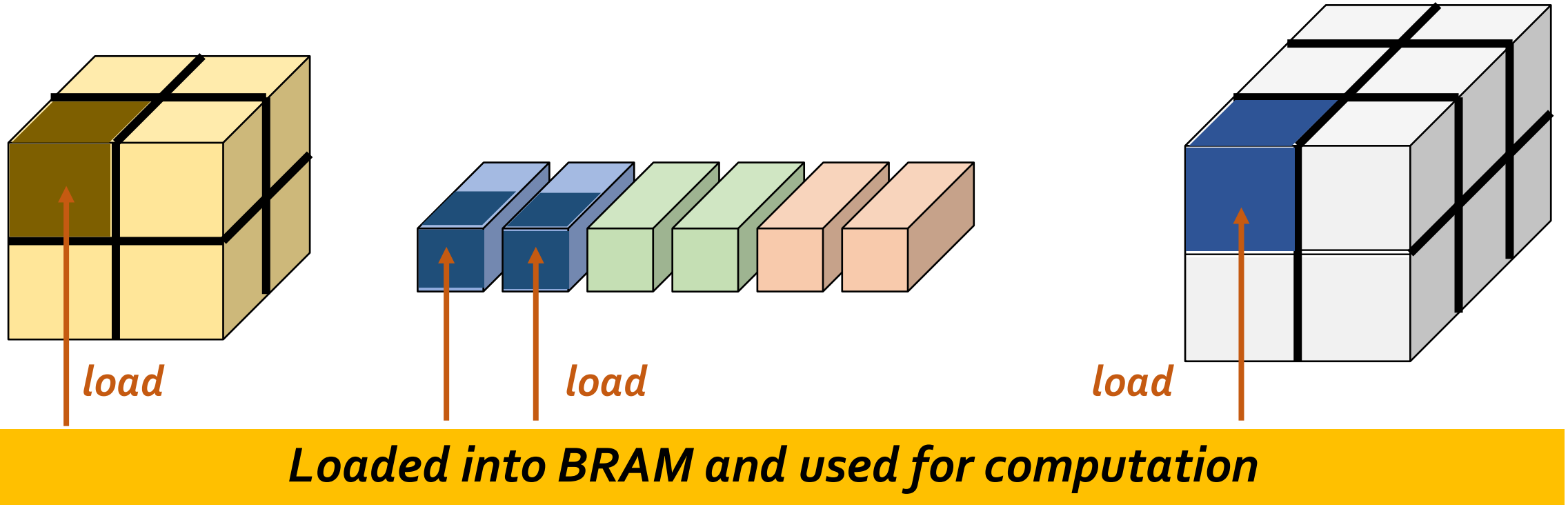
- Try to keep the input tile(s) in BRAM as long as possible



Load/store into/from BRAM and used for computation

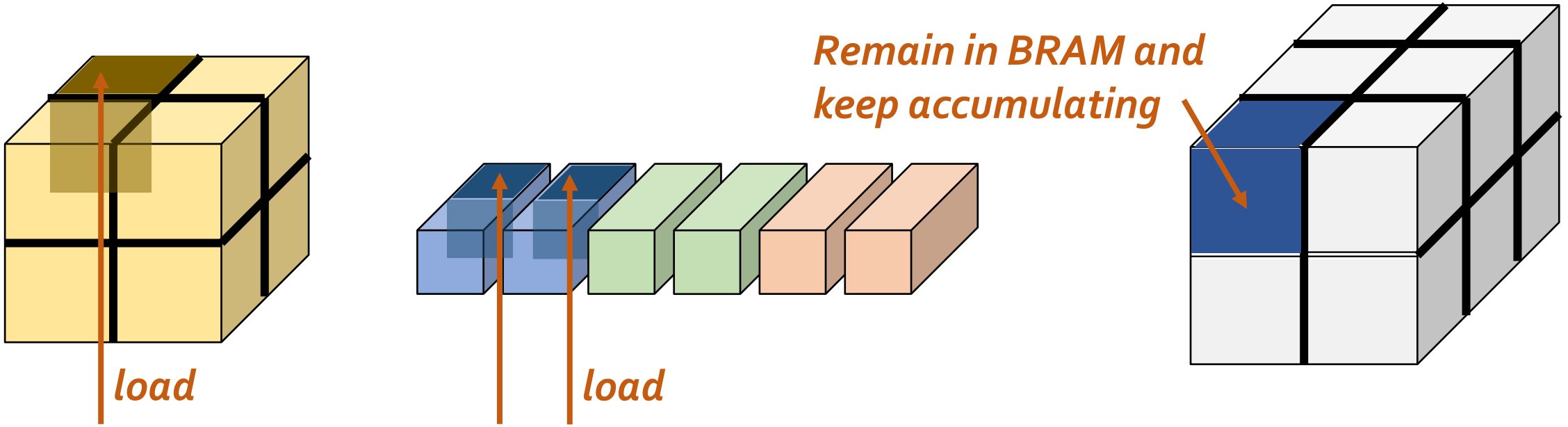
Output Stationary

- Try to keep the output tile(s) in BRAM as long as possible



Output Stationary

- Try to keep the output tile(s) in BRAM as long as possible



Loaded into BRAM and used for computation

Which is Better?

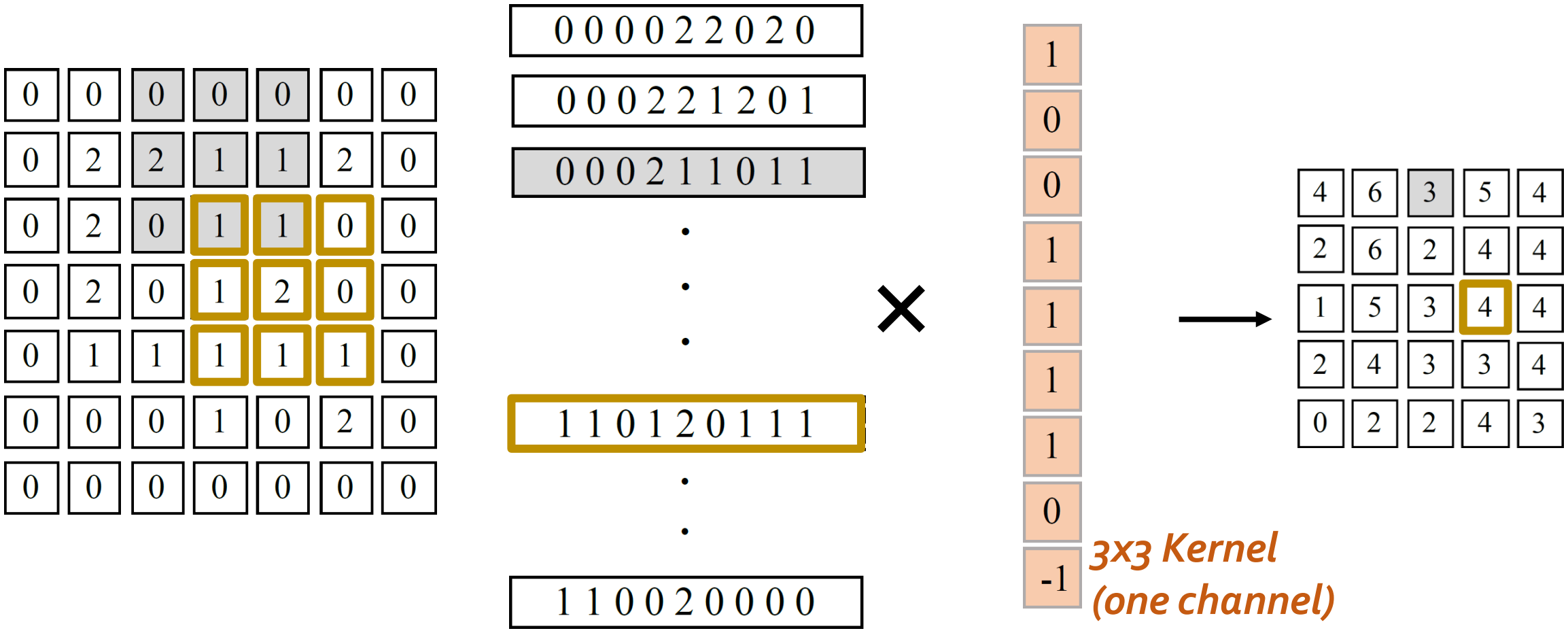
- **Good question… I personally like output stationary**
- **Design space exploration (DSE)**
  - Depending on your tile size, partition scheme, etc.
  - Please try to calculate if you're interested
- **Read more:**
  - Zhang, Chen, et al., "Optimizing fpga-based accelerator design for deep convolutional neural networks." FPGA 2015

Outline

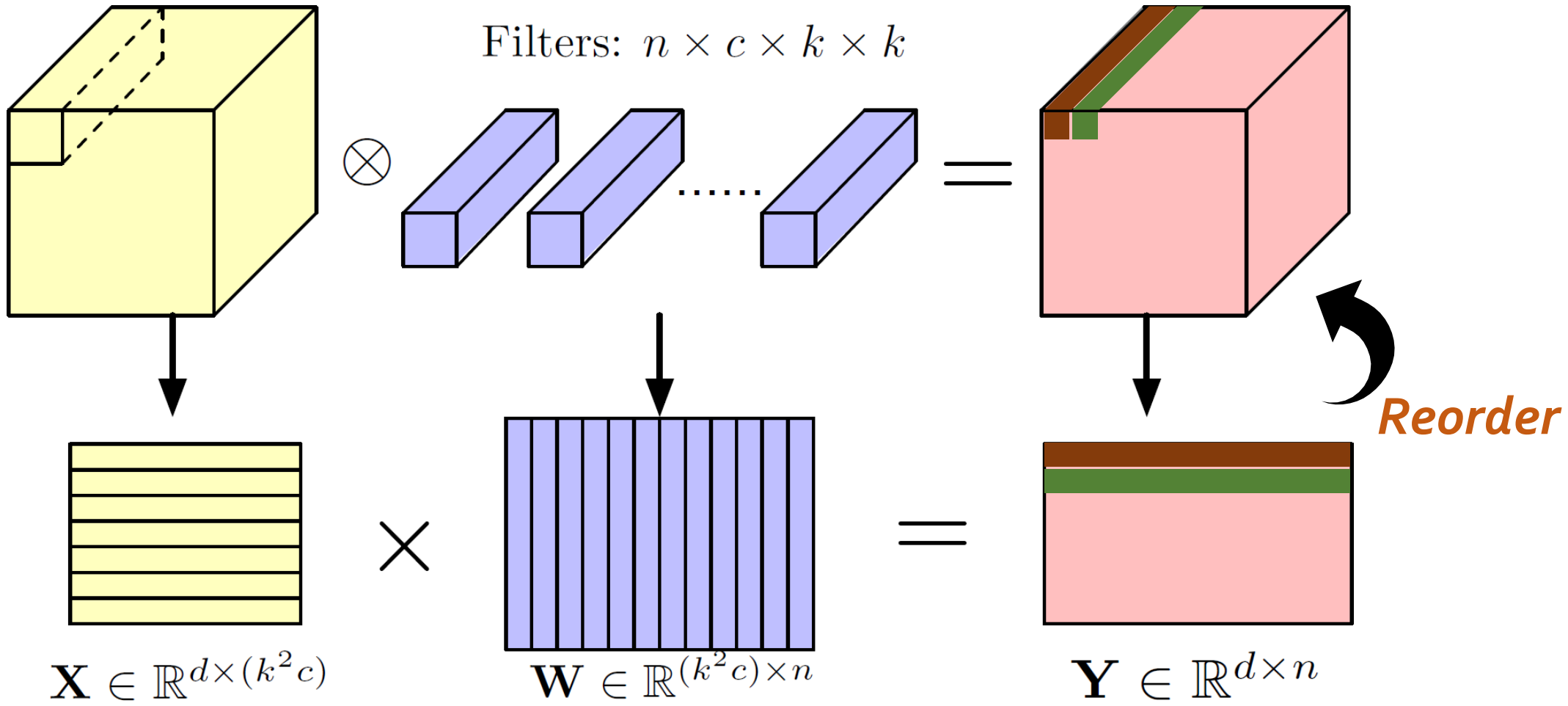
- Basic Convolution Computation
- Data Reuse: Input/Output/Weight stationary
- Img2Col and advanced Img2Col
- Sparse Matrix Multiplication

From Convolution to Matrix Multiplication

- **Img2Col (image-to-column) convolution – GEMM** (General Matrix Multiply)



From Convolution to Matrix Multiplication



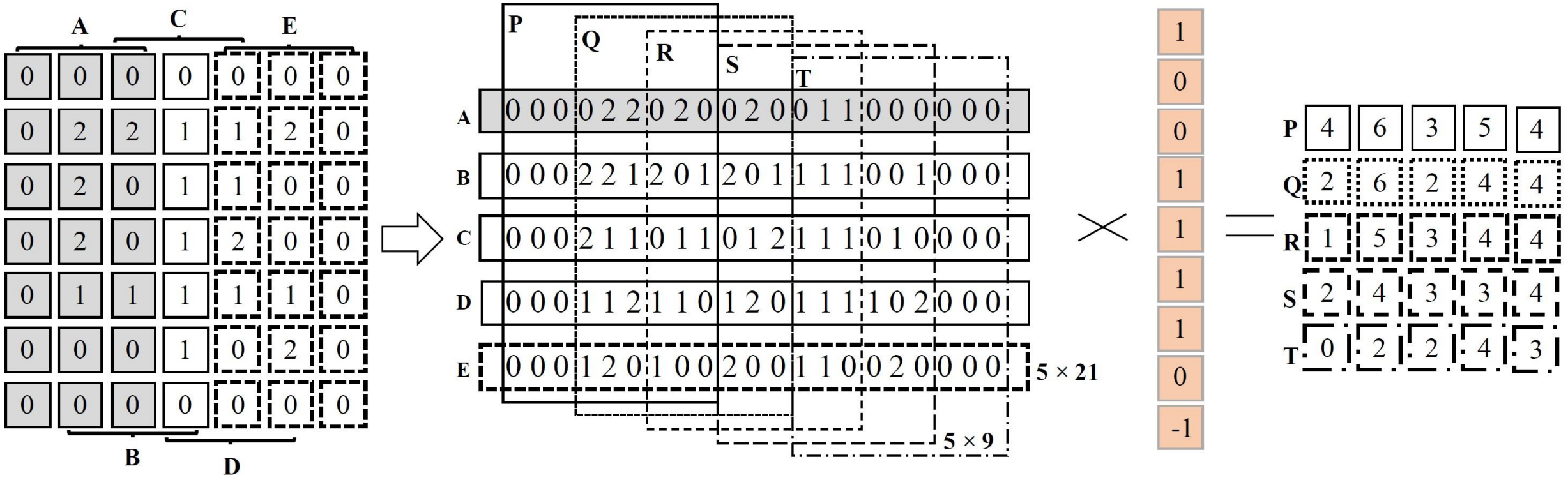
Img2Col Pros and Cons

- **Pros**
  - **Good performance** and easy to implement (especially on GPUs)
  - Applicable for any convolution configuration on any platform
  - BLAS-friendly memory layout to enjoy SIMD/locality/parallelism

- **Cons**
  - **Large** extra **memory** overhead

Memory-efficient Convolution

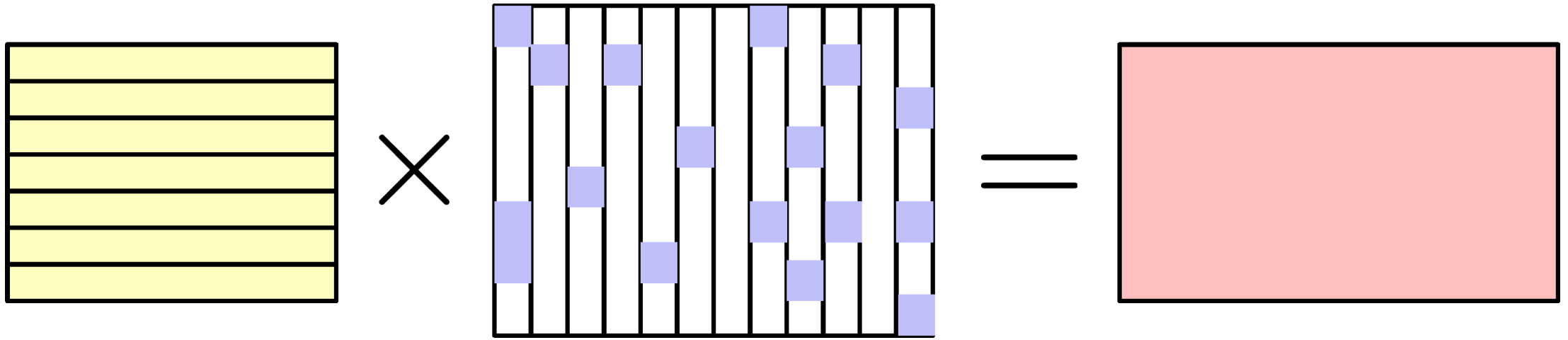
- **An improvement: remove "spatial" redundancy**
  - Smaller memory footprint, cache locality, and explicit parallelism



Cho, Minsik, and Daniel Brand. "MEC: memory-efficient convolution for deep neural network." In *International Conference on Machine Learning*. PMLR, 2017.

Sparse Matrix Multiplication (SpMM)

- DNNs may be redundant (over-parameterized), and filters may be sparse

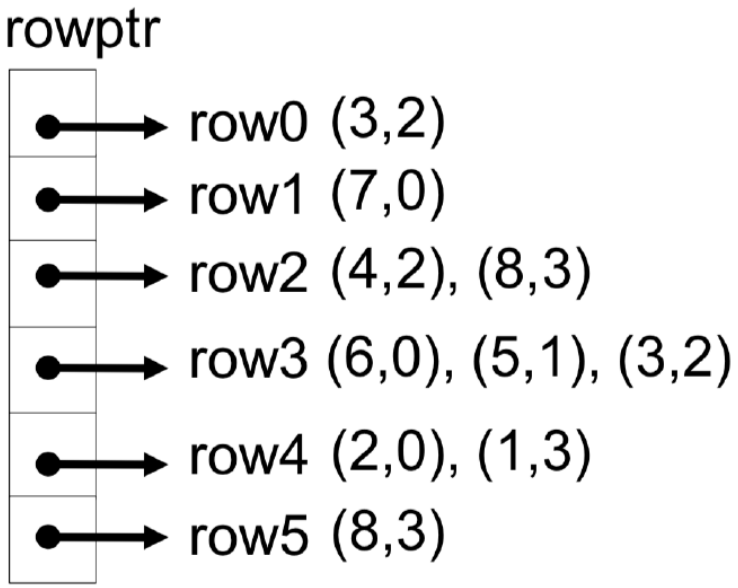


Sparse Matrix Representation

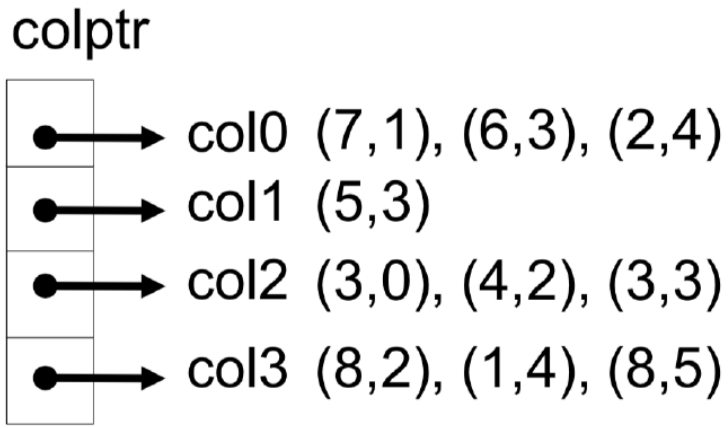
A

0	0	3	0
7	0	0	0
0	0	4	8
6	5	3	0
2	0	0	1
0	0	0	8

A matrix example



Compressed Sparse Row (CSR)



Compressed Sparse Column (CSC)

Related Papers

- Lin, Colin Yu, Ngai Wong, and Hayden Kwok-Hay So. "**Design space exploration for sparse matrix-matrix multiplication on FPGAs.**" *International Journal of Circuit Theory and Applications* 41, no. 2 (2013): 205-219.
- Zhuo, Ling, and Viktor K. Prasanna. "**Sparse matrix-vector multiplication on FPGAs**." In *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pp. 63-74. 2005.
- Fowers, Jeremy, Kalin Ovtcharov, Karin Strauss, Eric S. Chung, and Greg Stitt. "**A high memory bandwidth fpga accelerator for sparse matrix-vector multiplication**." In *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 36-43. IEEE, 2014.
- Zhang, Yan, Yasser H. Shalabi, Rishabh Jain, Krishna K. Nagar, and Jason D. Bakos. "**FPGA vs. GPU for sparse matrix vector multiply**." In *2009 International Conference on Field-Programmable Technology*, pp. 255-262. IEEE, 2009.
- Jain, Abhishek Kumar, Hossein Omidian, Henri Fraisse, Mansimran Benipal, Lisa Liu, and Dinesh Gaitonde. "**A domain-specific architecture for accelerating sparse matrix vector multiplication on fpgas**." In *2020 30th International conference on field-programmable logic and applications (FPL)*, pp. 127-132. IEEE, 2020.
- ...

Summary

- **Basic Convolution Computation**
  - Parallelism across Height, Width, Channel, Kernel, …
  - Scalability is an issue, and tiling is almost unavoidable
- **Data Reuse: Input/Output/Weight stationary**
  - Depends on how you partition your tiles
- **Img2Col and advanced Img2Col**
  - Good for regular memory and GPU matrix multiplication but bad for memory
- **Sparse Matrix Multiplication**
- **Converting Convolution to Matrix Multiplication can explore sparsity but has to pay extra cost**