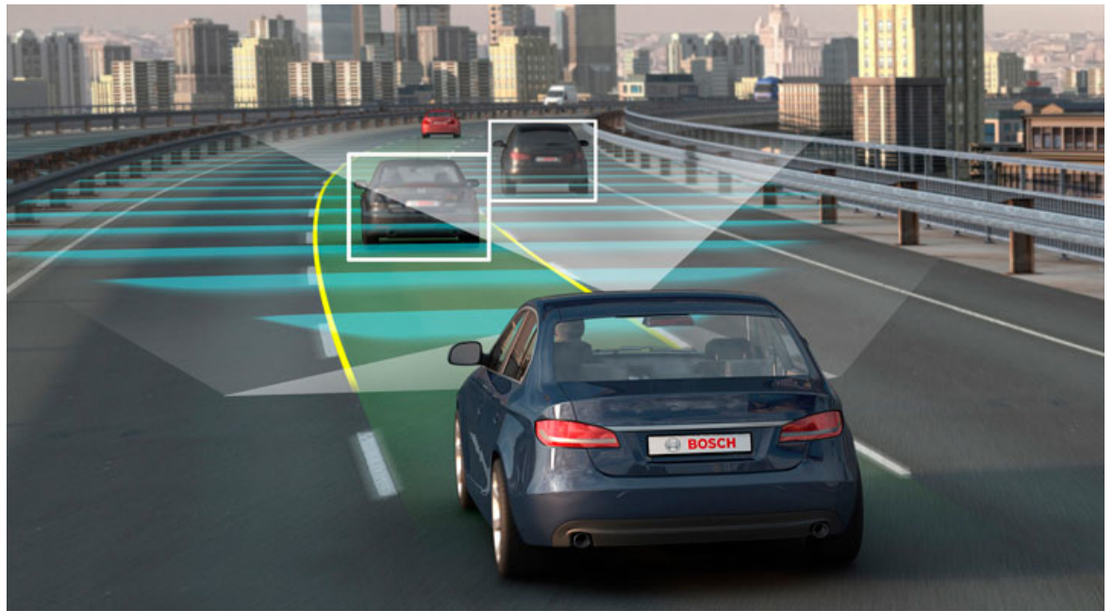
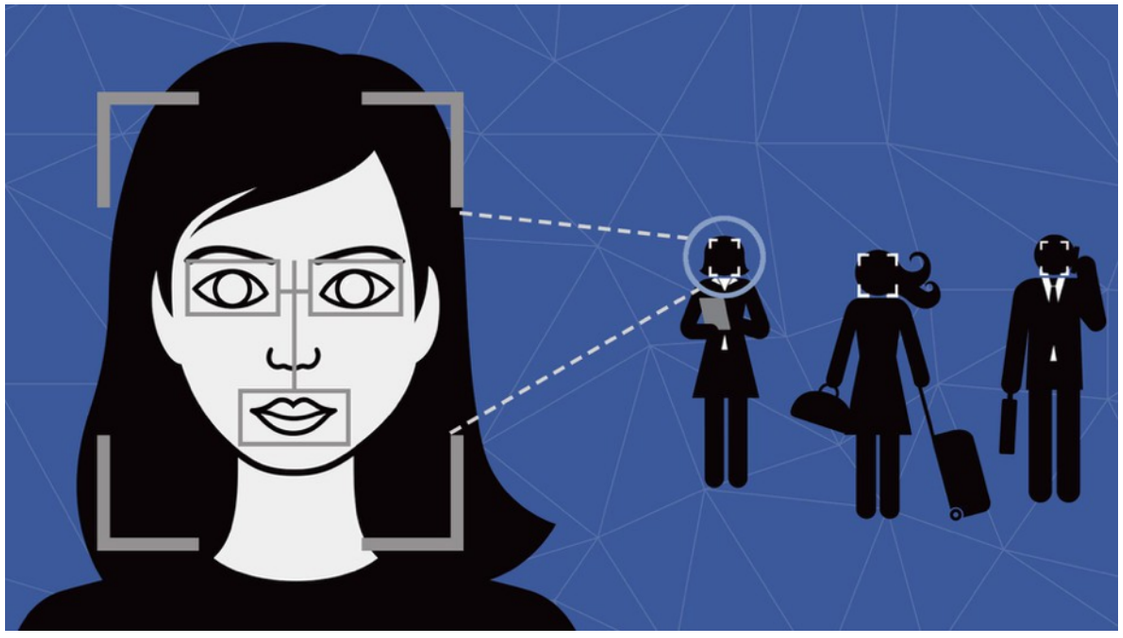
The DNN Dataflow Choice Maybe Overrated
— The Design Space of DNN Accelerators

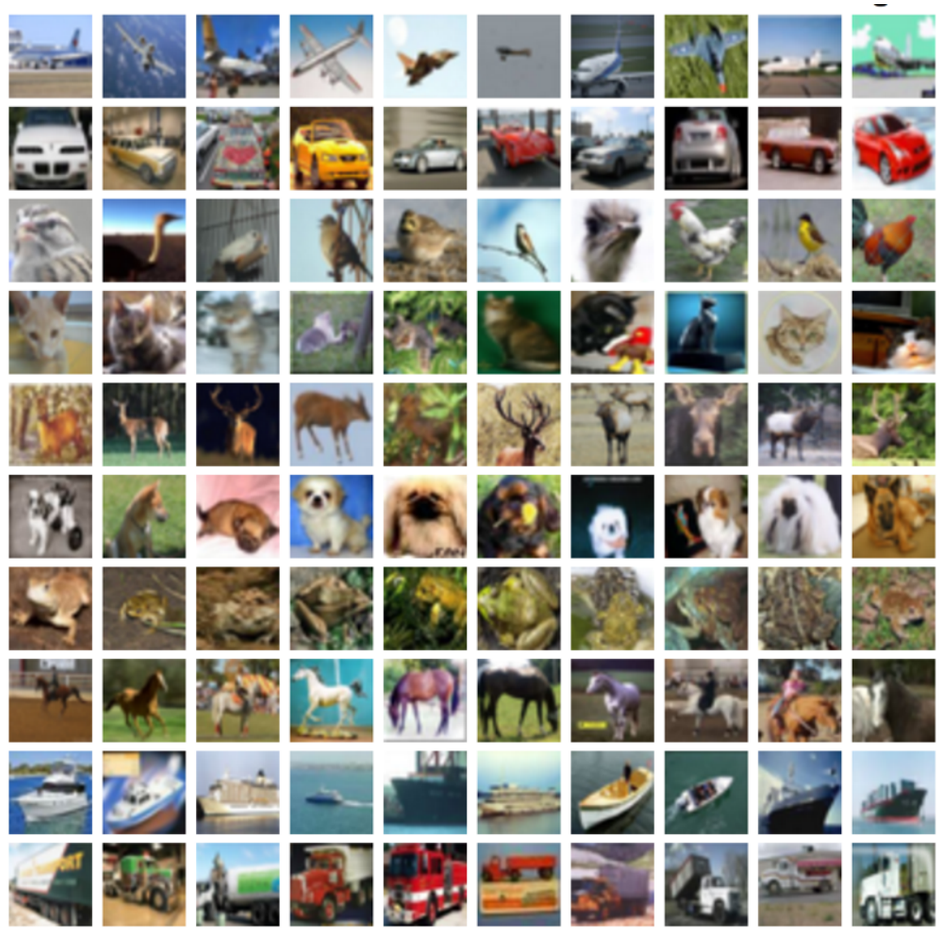
Xuan Yang
Advised by Prof. Mark Horowitz

Deep Neural Networks (DNNs)

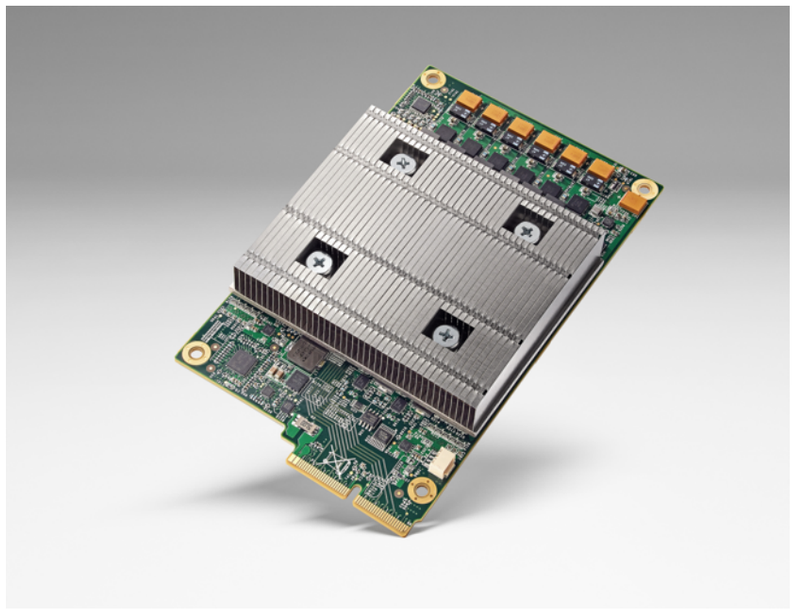




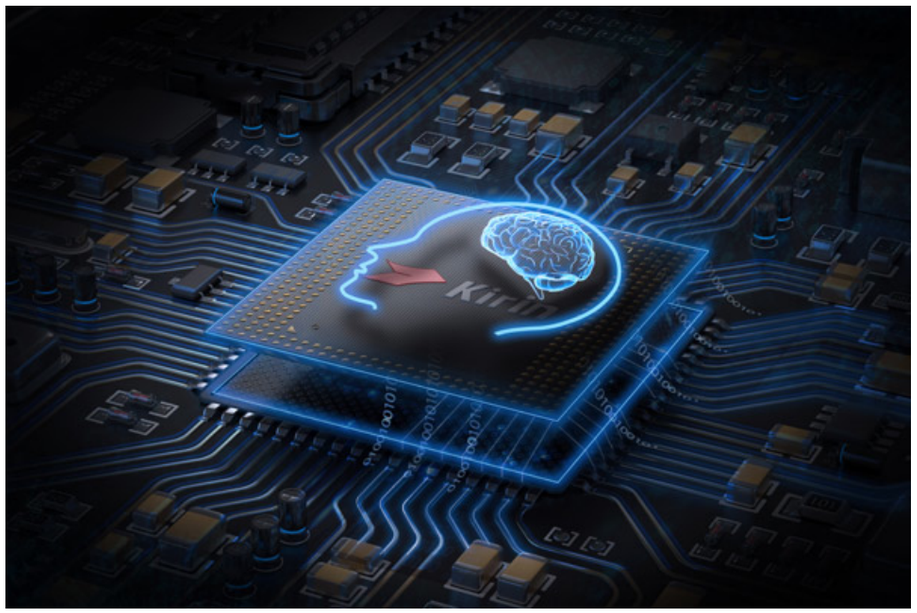
airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck



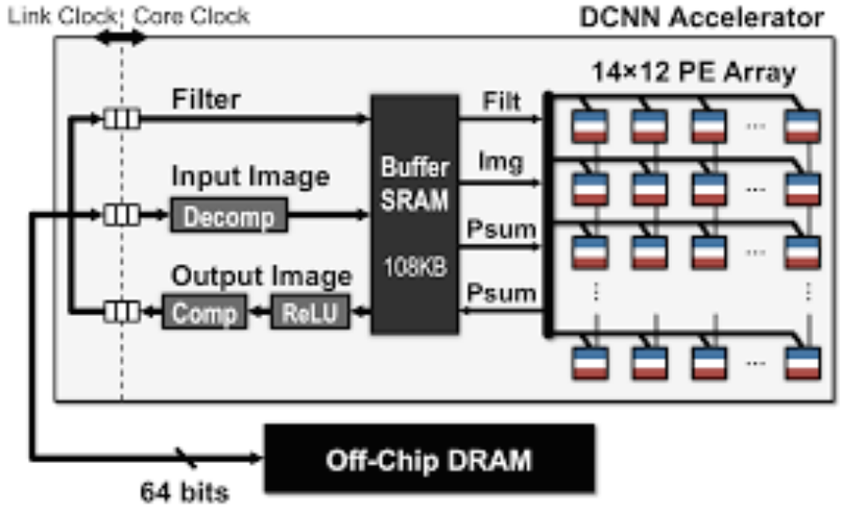
DNN Accelerators



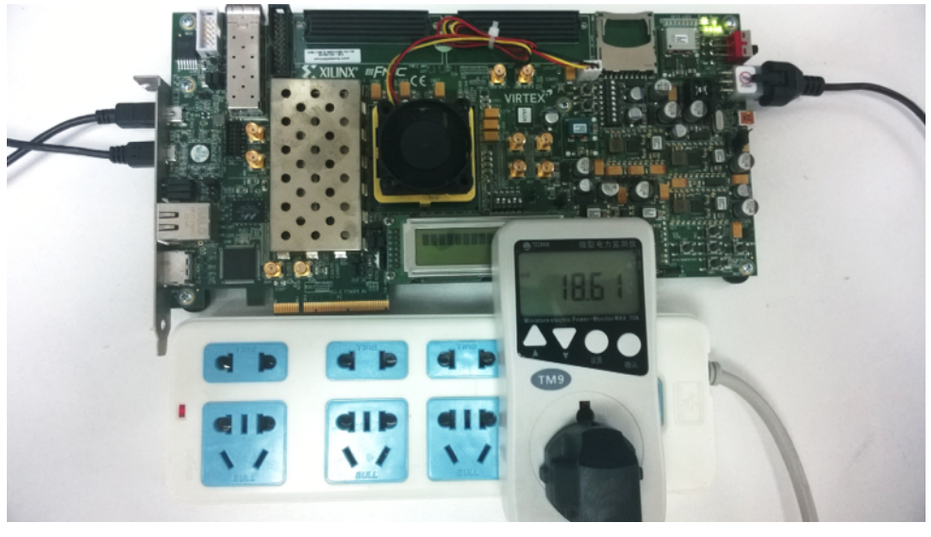
Google TPU



Huawei Kirin NPU



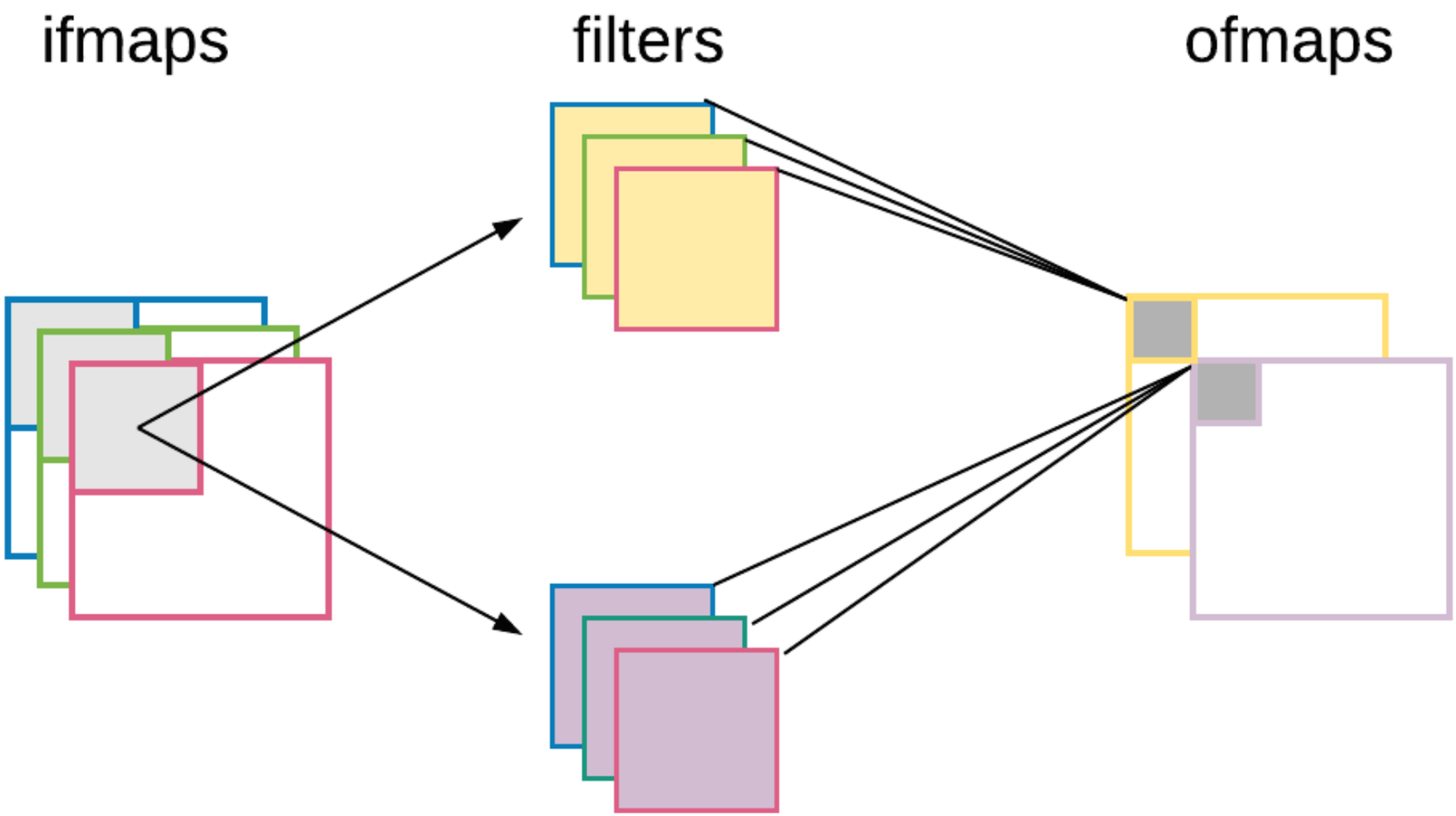
Eyeriss



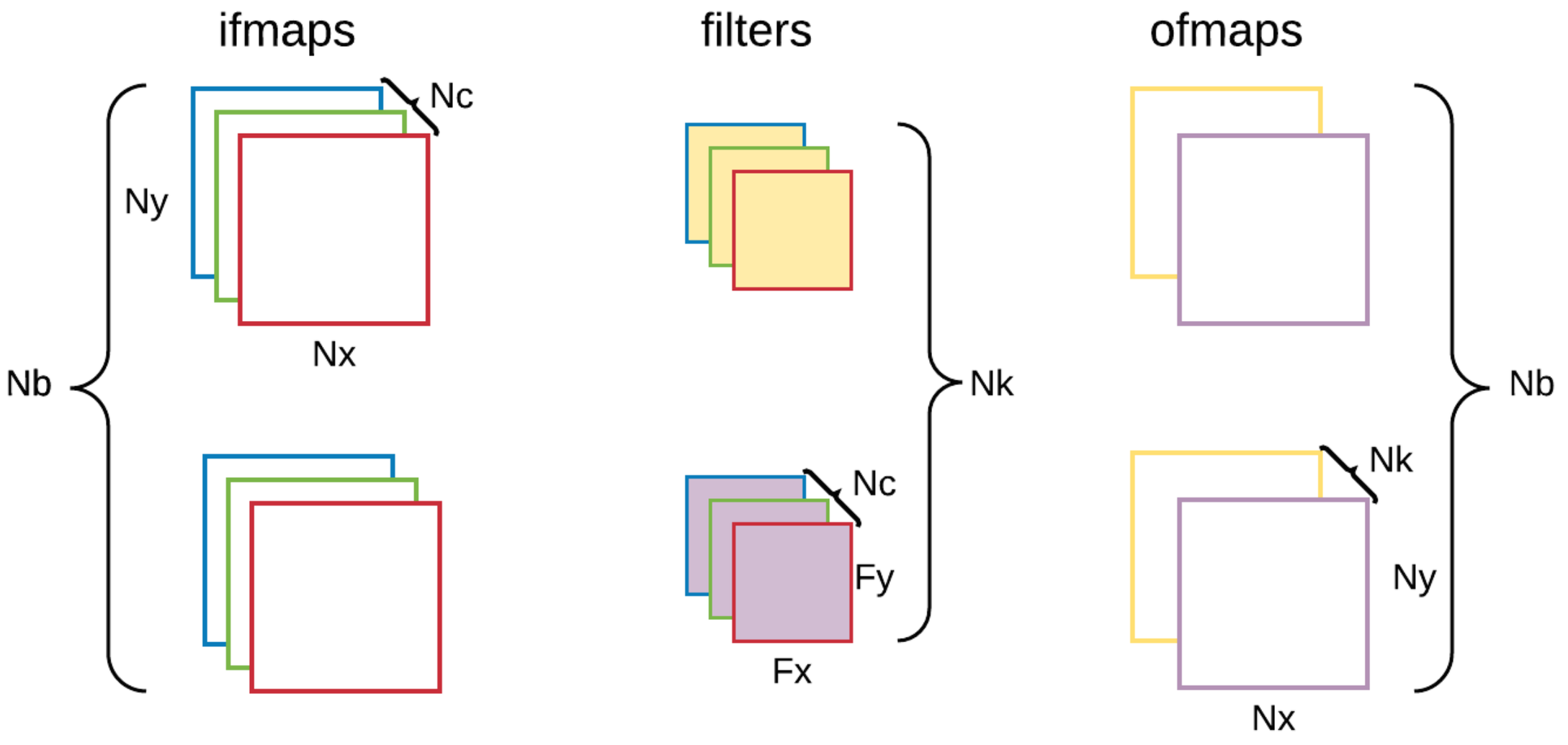
C. Zheng, et al (FPGA)

[1] https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html
[2] https://www.electronicsweekly.com/news/ifa-2017-huawei-reveals-low-power-kirin-970-mobile-ai-chipset-2017-09/
[3] http://eyeriss.mit.edu/

Computation of CNNs

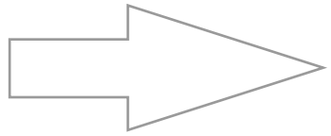


Computation of CNNs with Batching



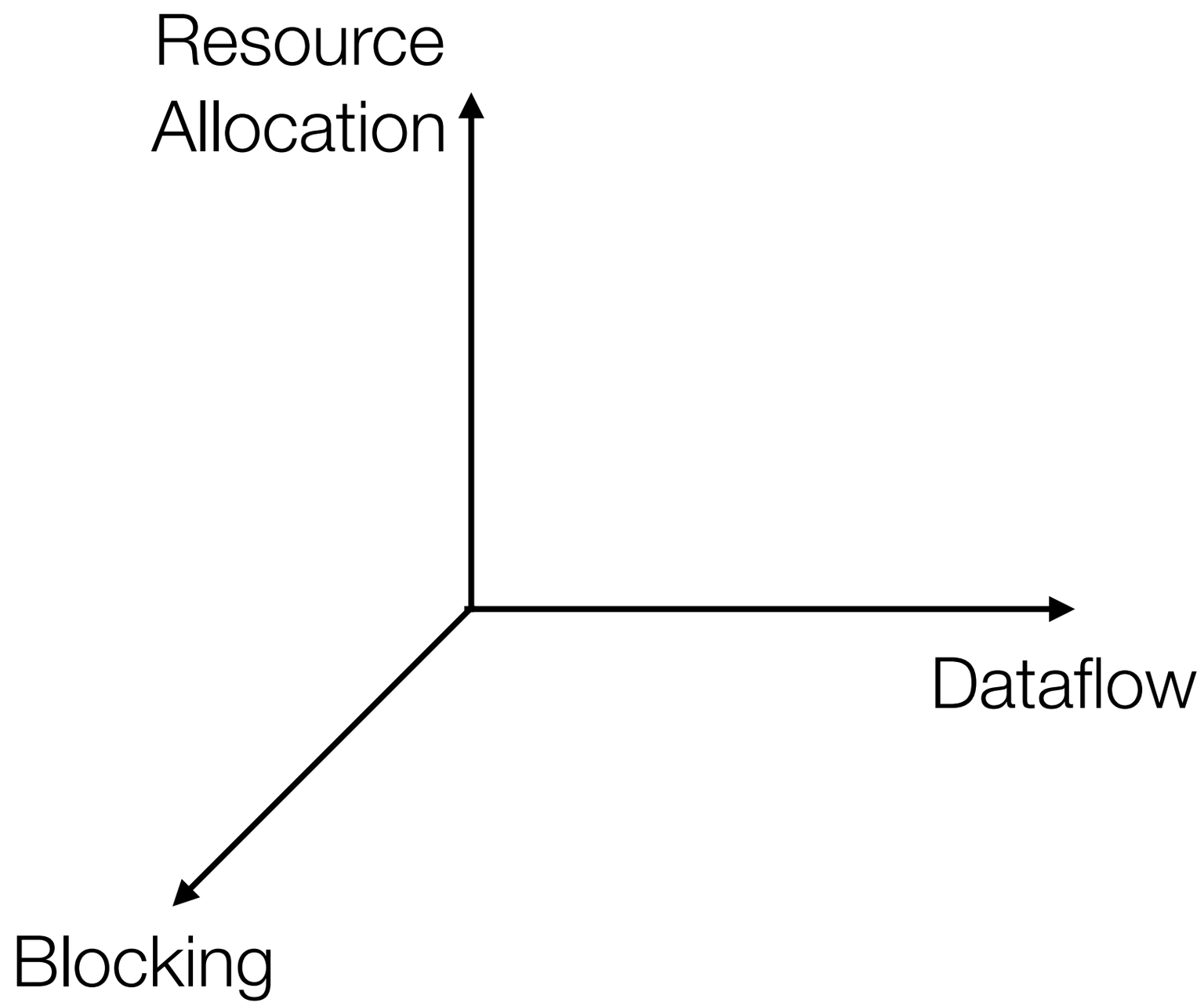
Dimensions of CNNs

```
foreach b in batch Nb
  foreach ifmap c in Nc
    foreach ofmap k in Nk
      foreach y in fmap_height Ny
        foreach x in fmap_width Nx
          foreach fy in filter_height Fy
            foreach fx in filter_width Fx
              out[x][y][k] += in[x+fx][y+fy][c]
                            * weight[fx][fy][c][k];
```

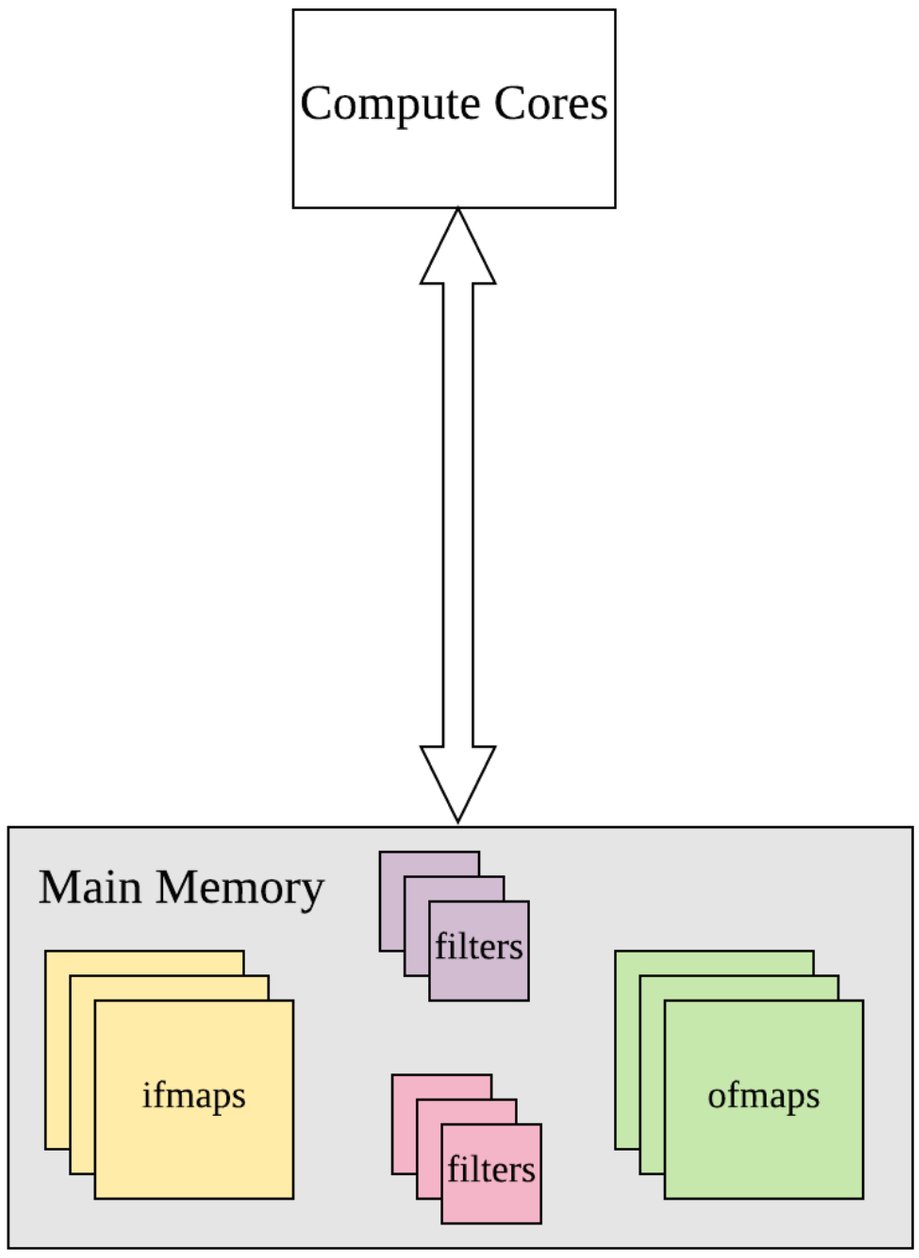


Large Design Space

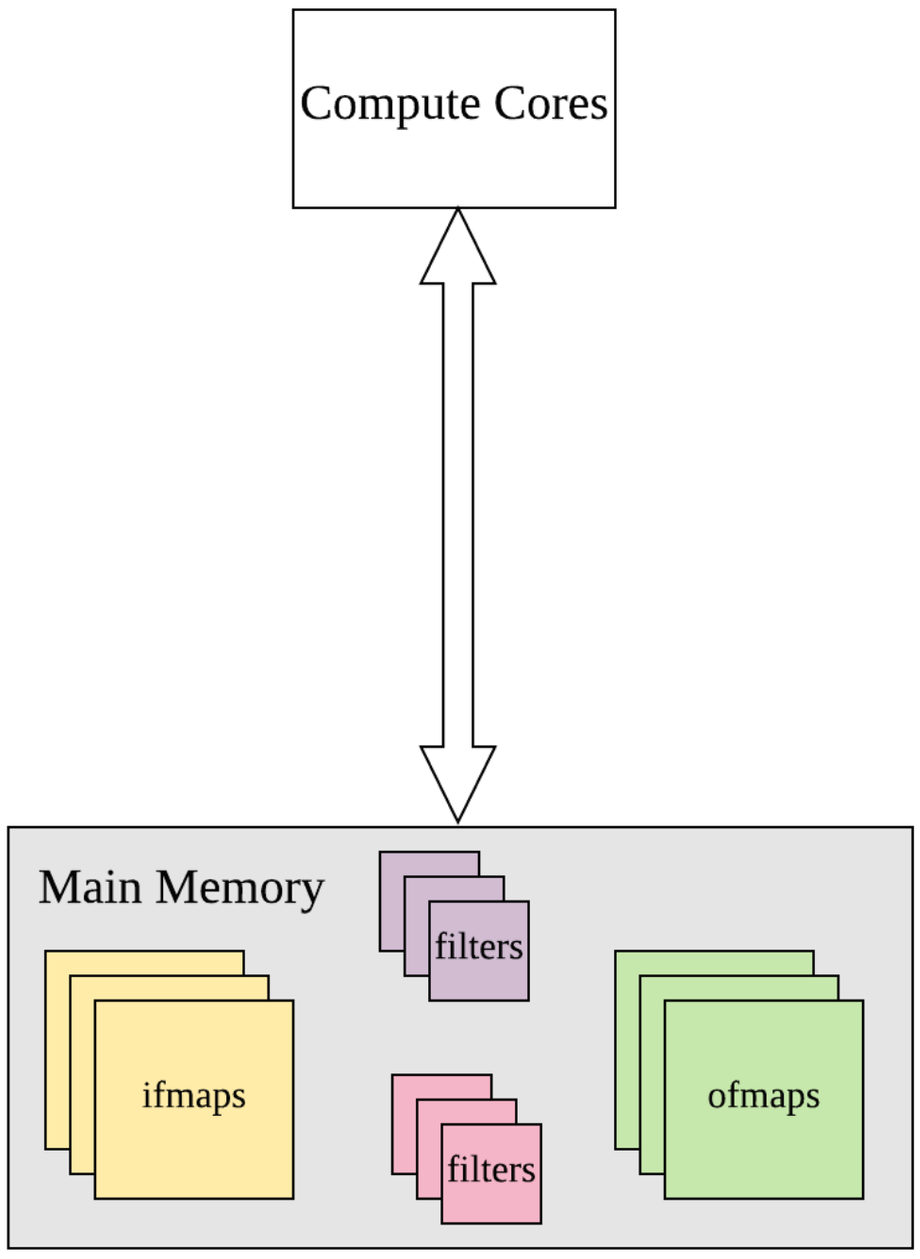
Design Space

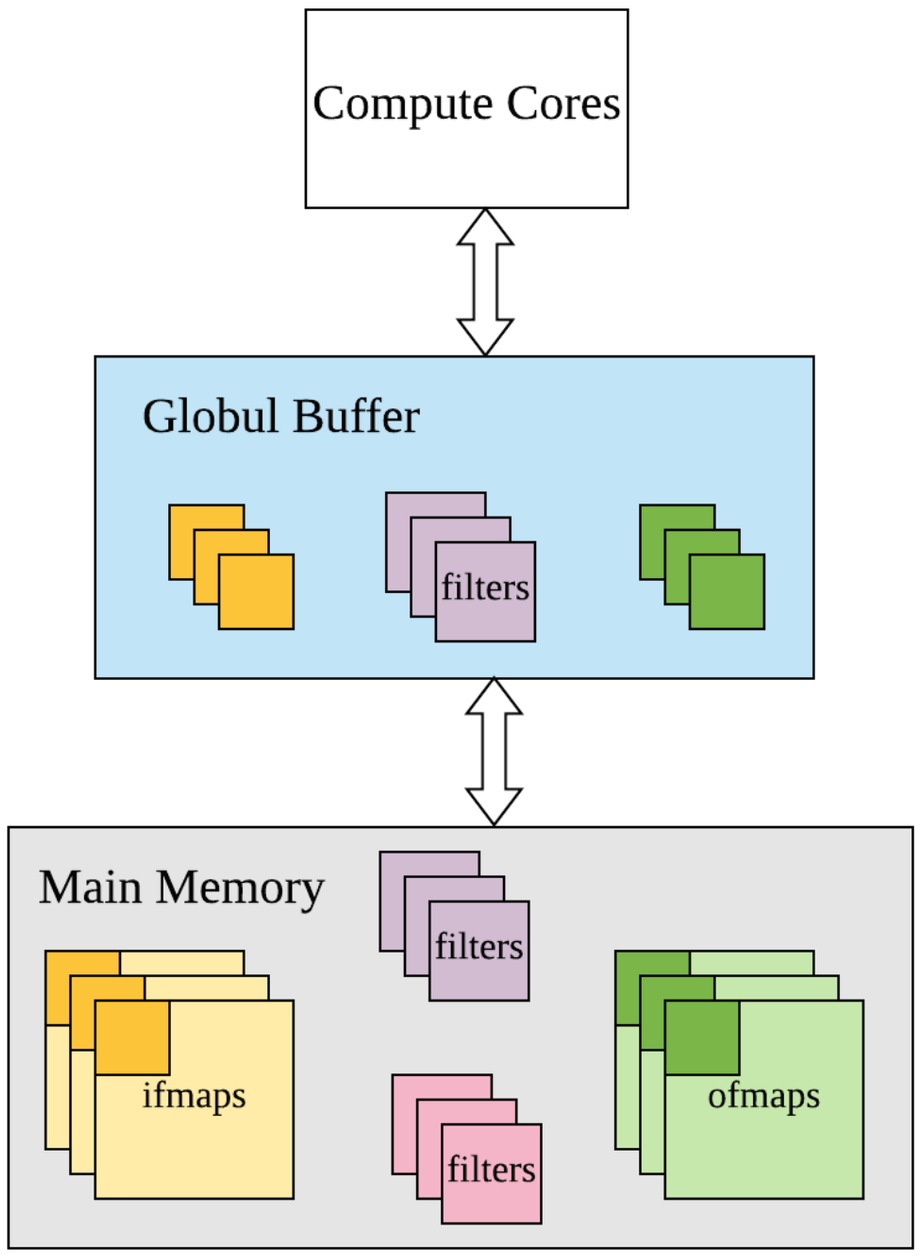


Different Ways of Blocking

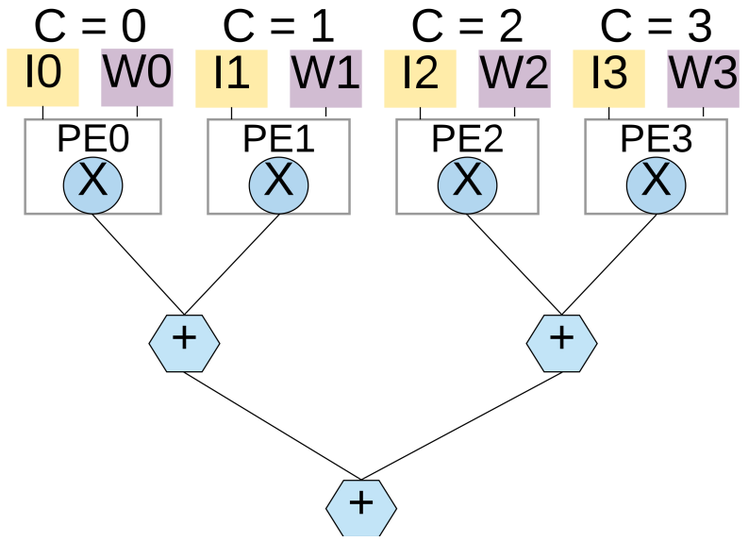


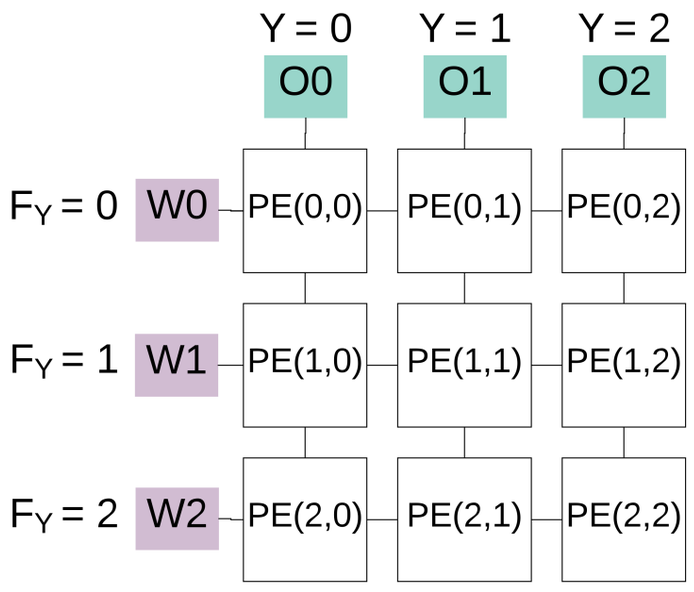
Different Ways of Blocking

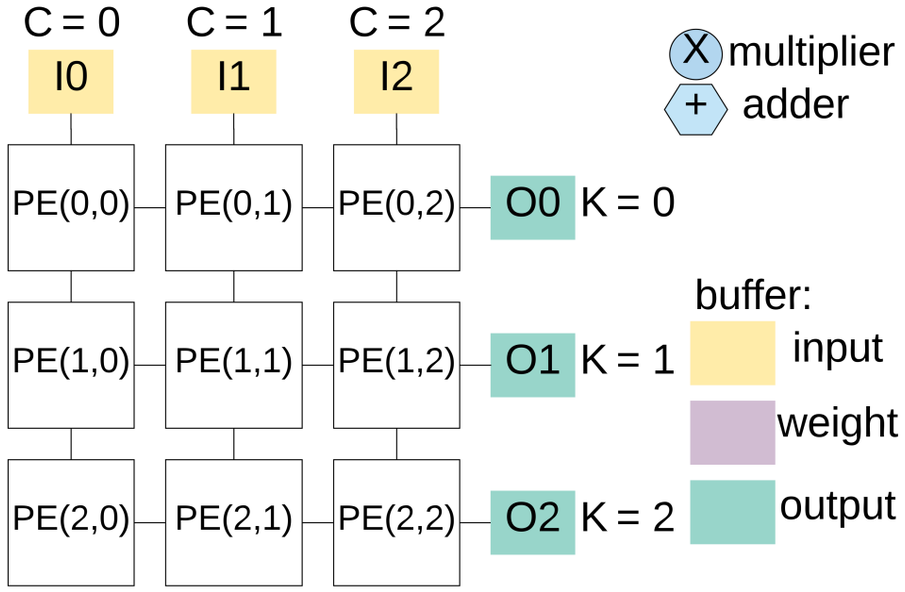




Different Dataflows

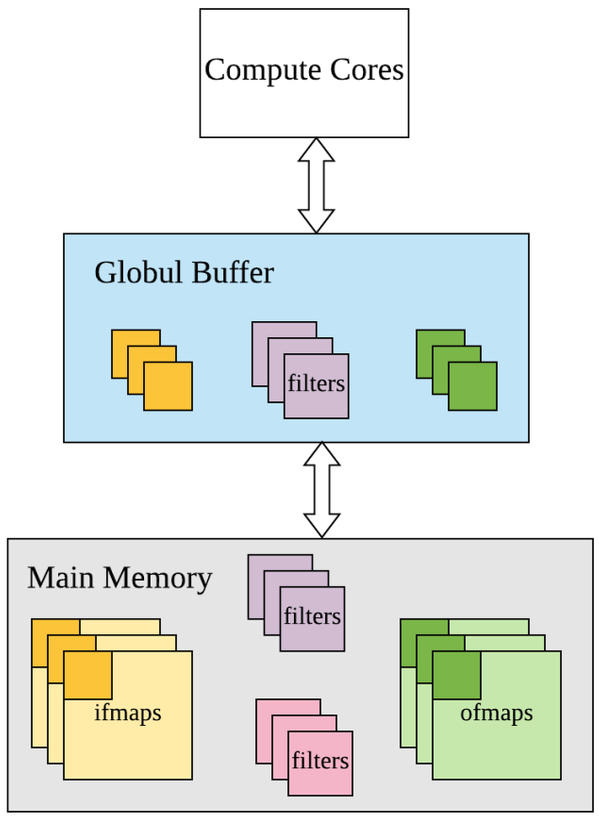




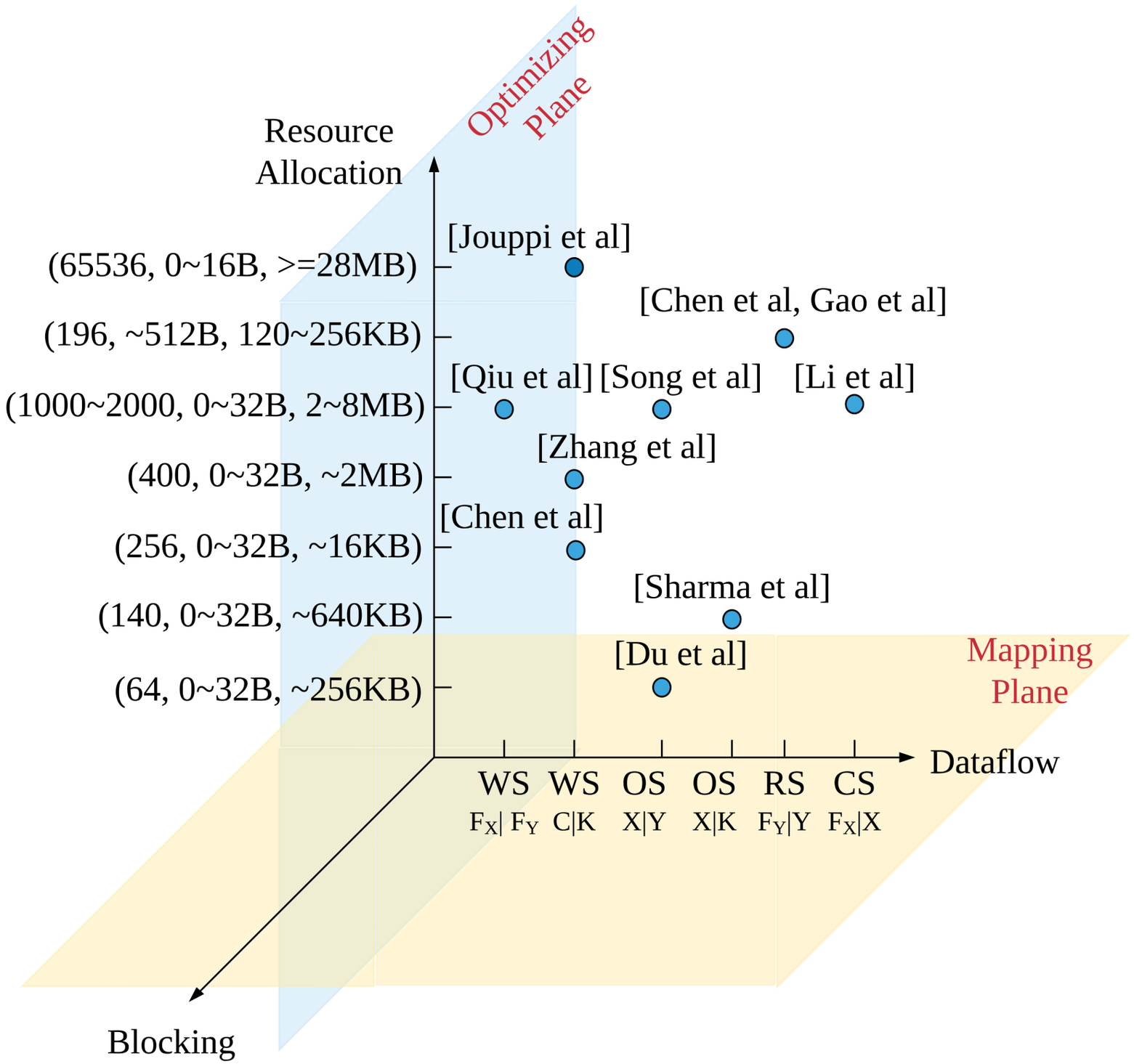


Different Resource Allocations

- How many levels in memory hierarchy?
- What is the right size for each one?
  - Larger size, cache more data, but each access more expensive



Design Space



How to fairly compare various designs?

How to find the most significant factors?

—with different platforms, micro-architectures, schedules.

How to fairly compare various designs?

How to find the most significant factors?

Frame them as different schedules to Halide program.

Why Halide?

- **Decouples algorithms from schedules**
  - Generate different hardware designs without modifying the algorithm code
  - Compact representation of the scheduling space

Why Halide?

- High performance image processing with short code
- Support multiple backends, such as x86, ARM, CUDA …

(a) Clean C++ : 9.94 ms per megapixel

```
void blur(const Image &in, Image &blurred) {
 Image tmp(in.width(), in.height());

 for (int y = 0; y < in.height(); y++)
  for (int x = 0; x < in.width(); x++)
   tmp(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;

 for (int y = 0; y < in.height(); y++)
  for (int x = 0; x < in.width(); x++)
   blurred(x, y) = (tmp(x, y-1) + tmp(x, y) + tmp(x, y+1))/3;
}
```

(b) Fast C++ (for x86) : 0.90 ms per megapixel

```
void fast_blur(const Image &in, Image &blurred) {
 __m128i one_third = _mm_set1_epi16(21846);
 #pragma omp parallel for
 for (int yTile = 0; yTile < in.height(); yTile += 32) {
  __m128i a, b, c, sum, avg;
  __m128i tmp[(256/8)*(32+2)];
  for (int xTile = 0; xTile < in.width(); xTile += 256) {
   __m128i *tmpPtr = tmp;
   for (int y = -1; y < 32+1; y++) {
    const uint16_t *inPtr = &(in(xTile, yTile+y));
    for (int x = 0; x < 256; x += 8) {
     a = _mm_loadu_si128((__m128i*)(inPtr-1));
     b = _mm_loadu_si128((__m128i*)(inPtr+1));
     c = _mm_load_si128((__m128i*)(inPtr));
     sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
     avg = _mm_mulhi_epi16(sum, one_third);
     _mm_store_si128(tmpPtr++, avg);
     inPtr += 8;
   }}
   tmpPtr = tmp;
   for (int y = 0; y < 32; y++) {
    __m128i *outPtr = (__m128i *)(&(blurred(xTile, yTile+y)));
    for (int x = 0; x < 256; x += 8) {
     a = _mm_load_si128(tmpPtr+(2*256)/8);
     b = _mm_load_si128(tmpPtr+256/8);
     c = _mm_load_si128(tmpPtr++);
     sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
     avg = _mm_mulhi_epi16(sum, one_third);
     _mm_store_si128(outPtr++, avg);
}}}}}
```

(c) Halide : 0.90 ms per megapixel

```
Func halide_blur(Func in) {
 Func tmp, blurred;
 Var x, y, xi, yi;

 // The algorithm
 tmp(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;
 blurred(x, y) = (tmp(x, y-1) + tmp(x, y) + tmp(x, y+1))/3;

 // The schedule
 blurred.tile(x, y, xi, yi, 256, 32)
        .vectorize(xi, 8).parallel(y);
 tmp.chunk(x).vectorize(x, 8);

 return blurred;
}
```

Halide Frontend

```
Func conv(Func input, Func weight) {
    Func clamped, output;
    Var x, y, c;
    RDom r(-1, 3, -1, 3, 0, 16);  //3x3 window, input channel Nc is 16
    //padding
    clamped(x, y, c) = BoundaryConditions::constant_exterior(input, 0);
    output(x, y, c) = 0;
    output(x, y, c) += clamped(x+r.x, y+r.y, r.z) * weight(r.x+1, r.y+1, r.z, c);
    return output;
}
```

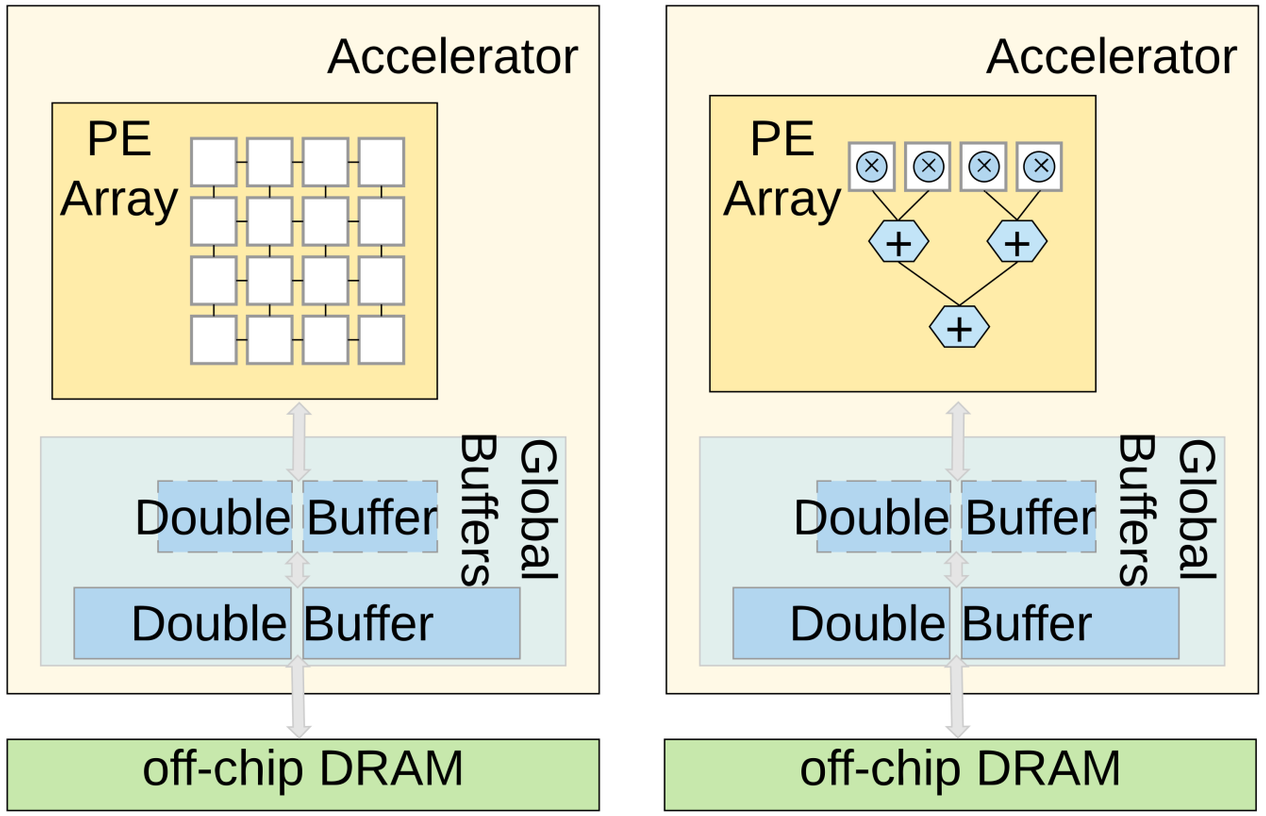
Scheduler Frontend

- Blocking computation: **tile**, **reorder**
- Create another level of memory: **in**
- Compute and store granularity: **compute_at**, **store_at**
- Parallel computation (duplicate units): **unroll**

```
output.tile(x, y, xo, yo, xi, yi, 8, 8).reorder(xi, yi, xo, yo);
clamped.in().compute_at(output, xo);
weight.in().compute_at(output, xo);
output.unroll(xi, 4);
```

Schedules for Micro-architecture

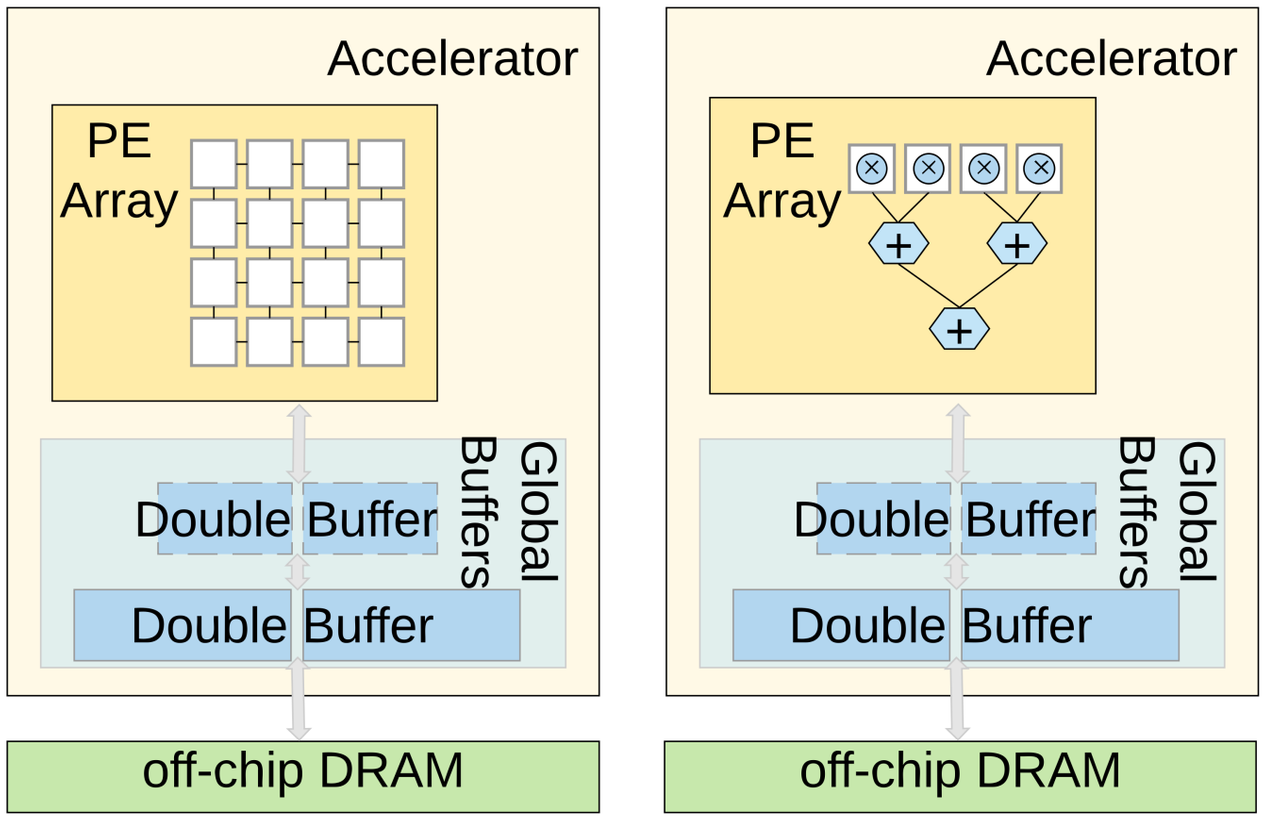
- Memory hierarchy with double buffers: **in**
- Systolic array VS reduction tree: **systolic**



(a) Systolic Array (b) Reduction Tree

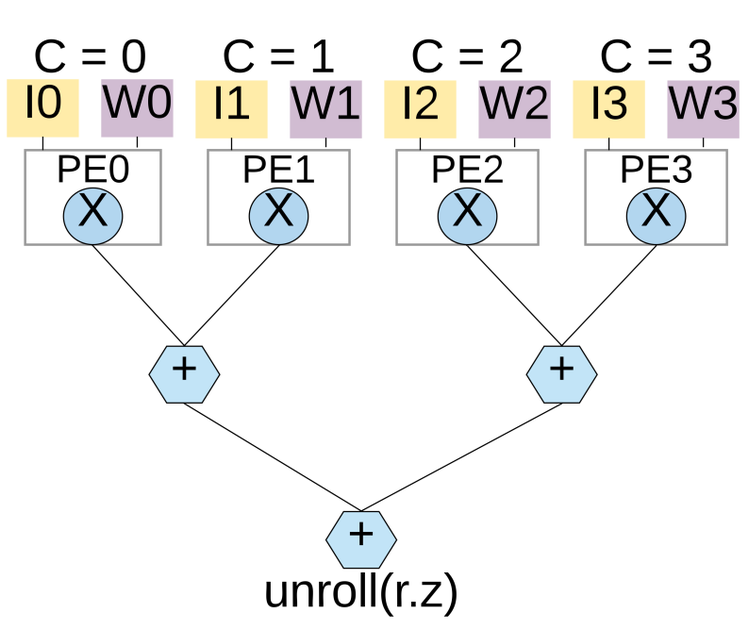
Schedules for Micro-architecture

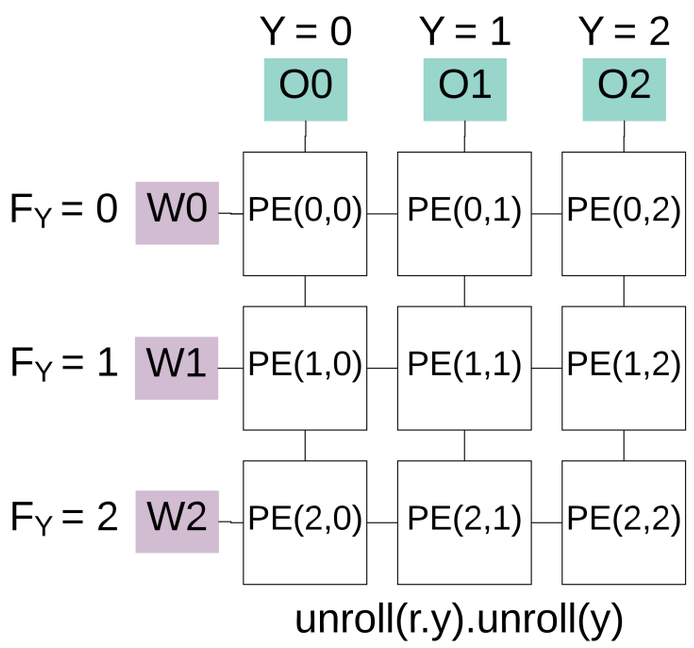
- Memory hierarchy with double buffers: **in**
- Systolic array VS reduction tree: **systolic**
- Parallel computation (duplicate units): **unroll**

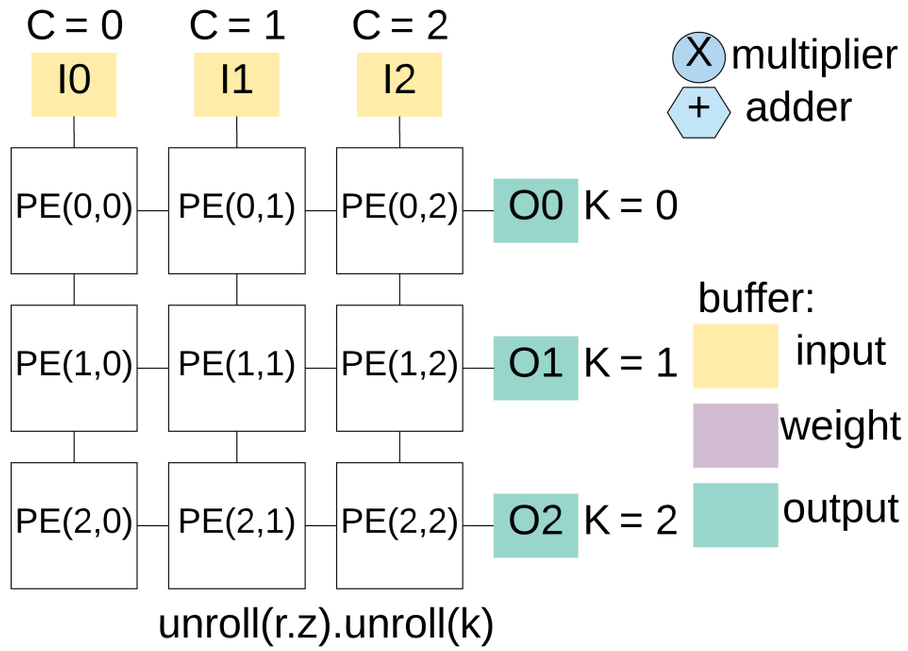


(a) Systolic Array (b) Reduction Tree

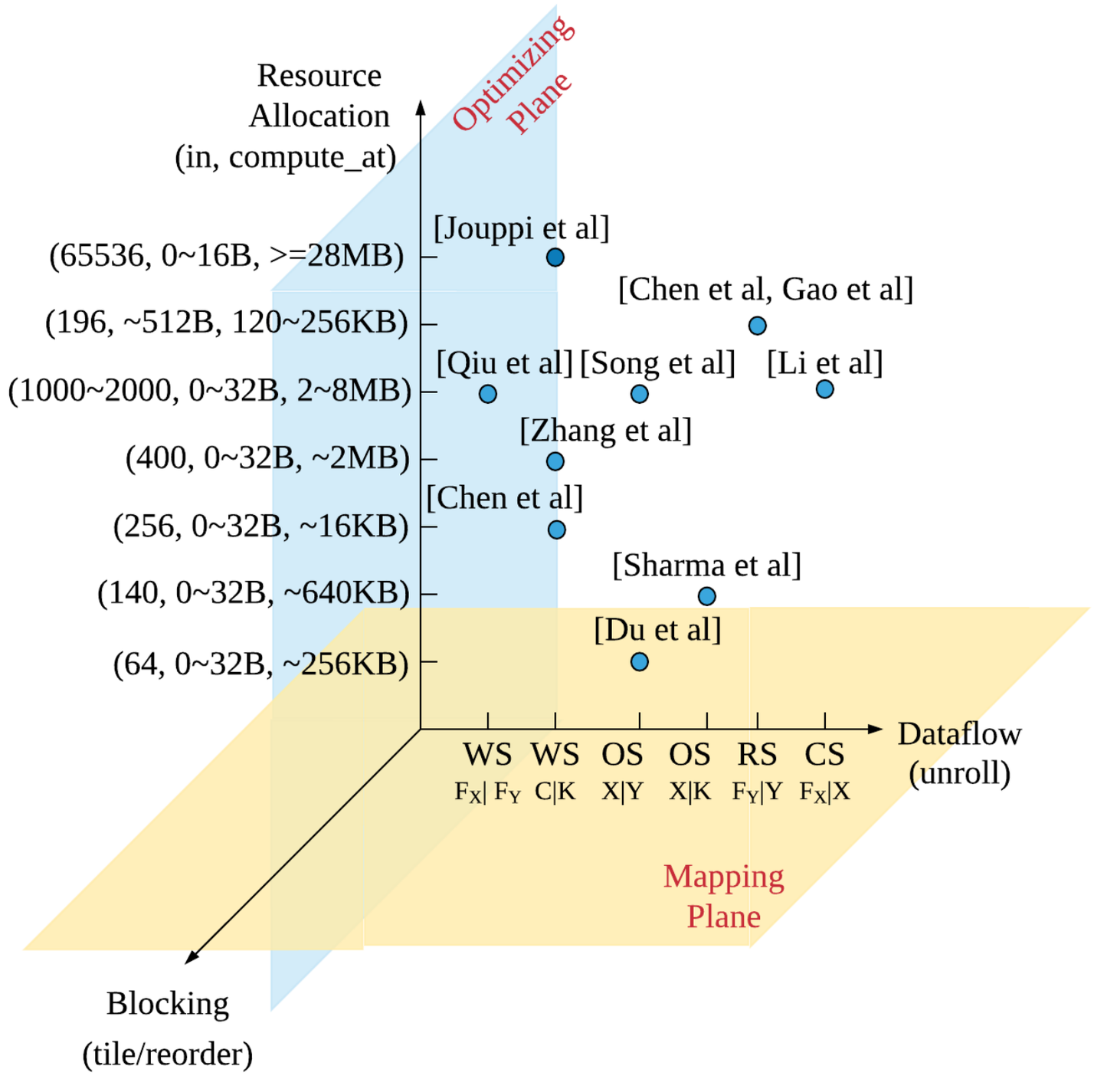
Schedules for Mappings of Previous Accelerators



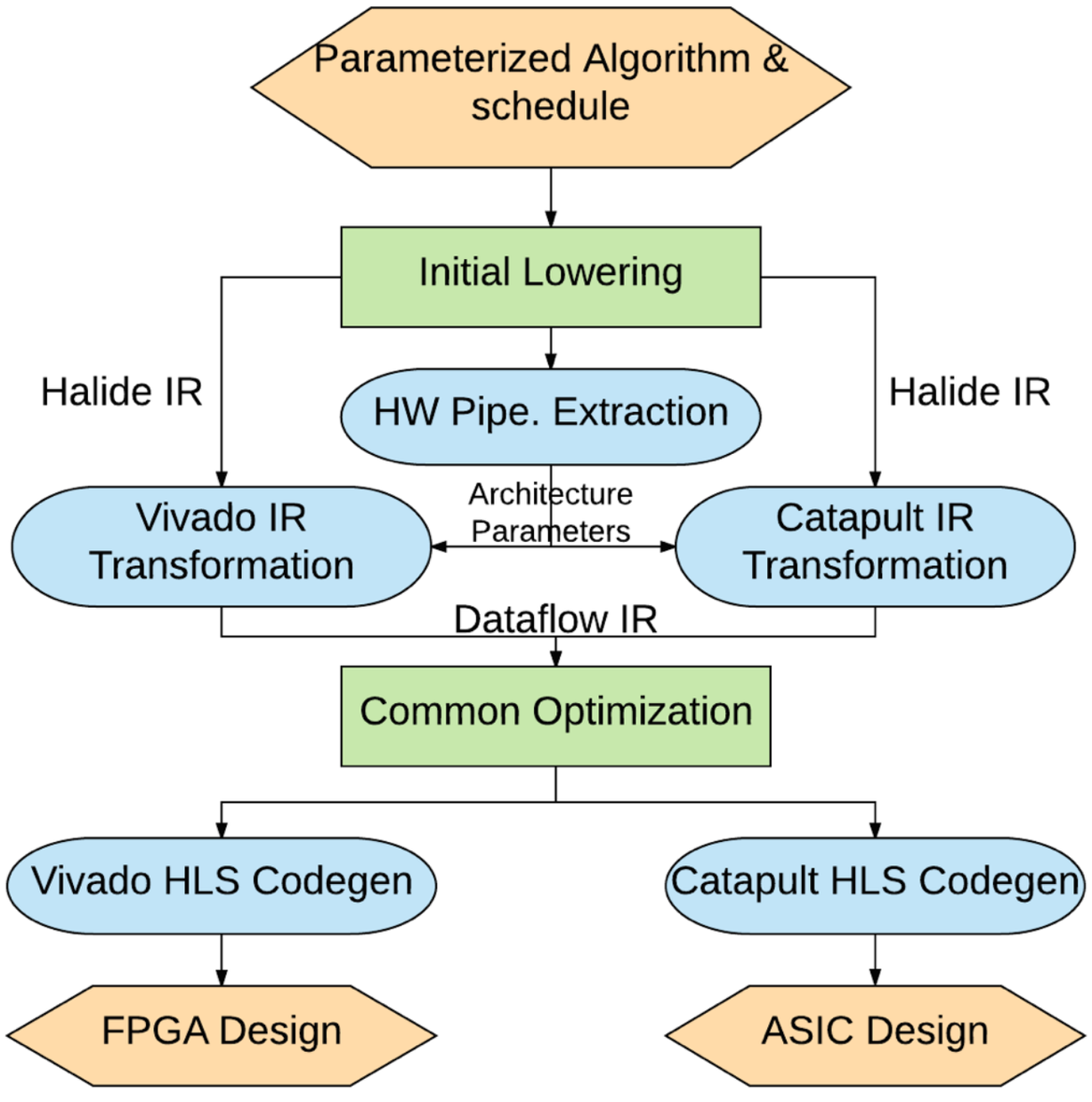




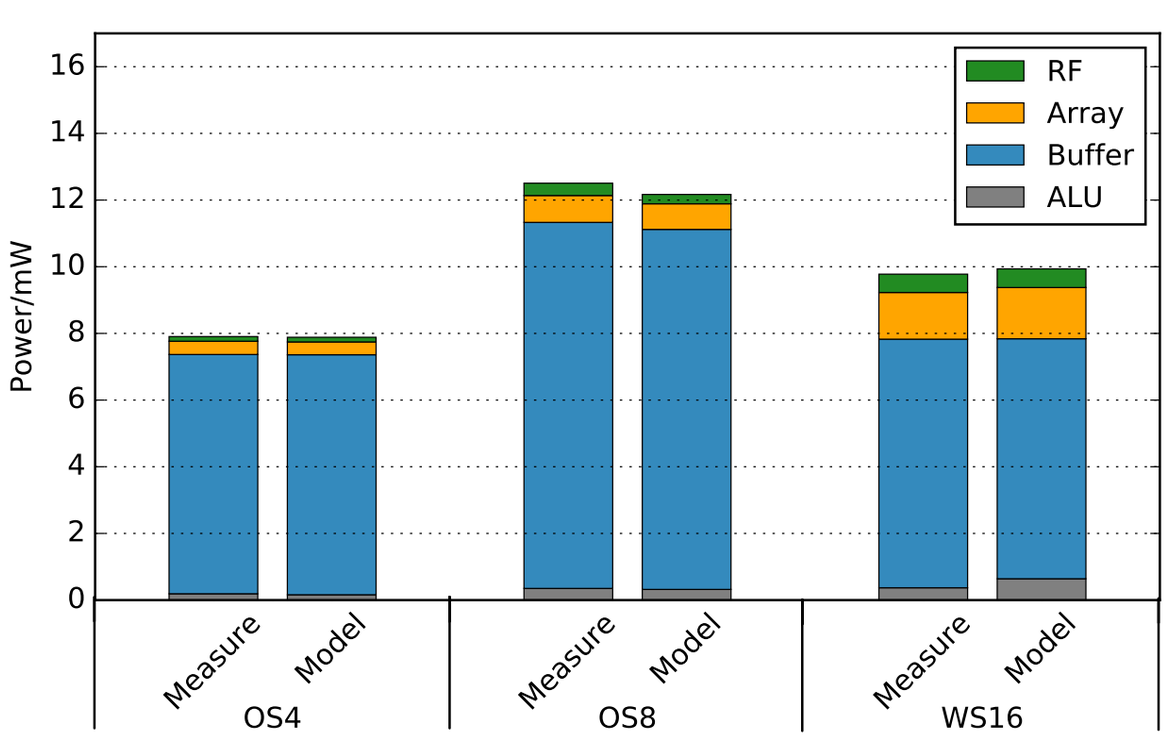
Design Space

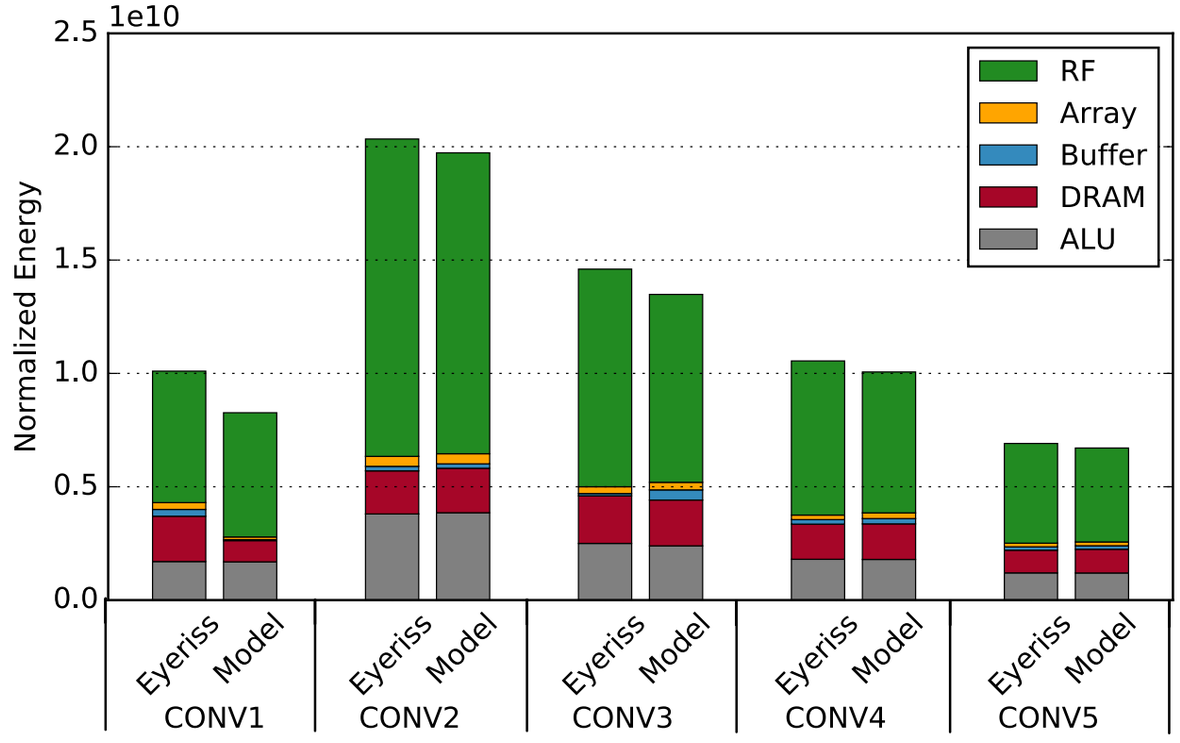


Halide to Hardware Compilation Flow



Model Validation



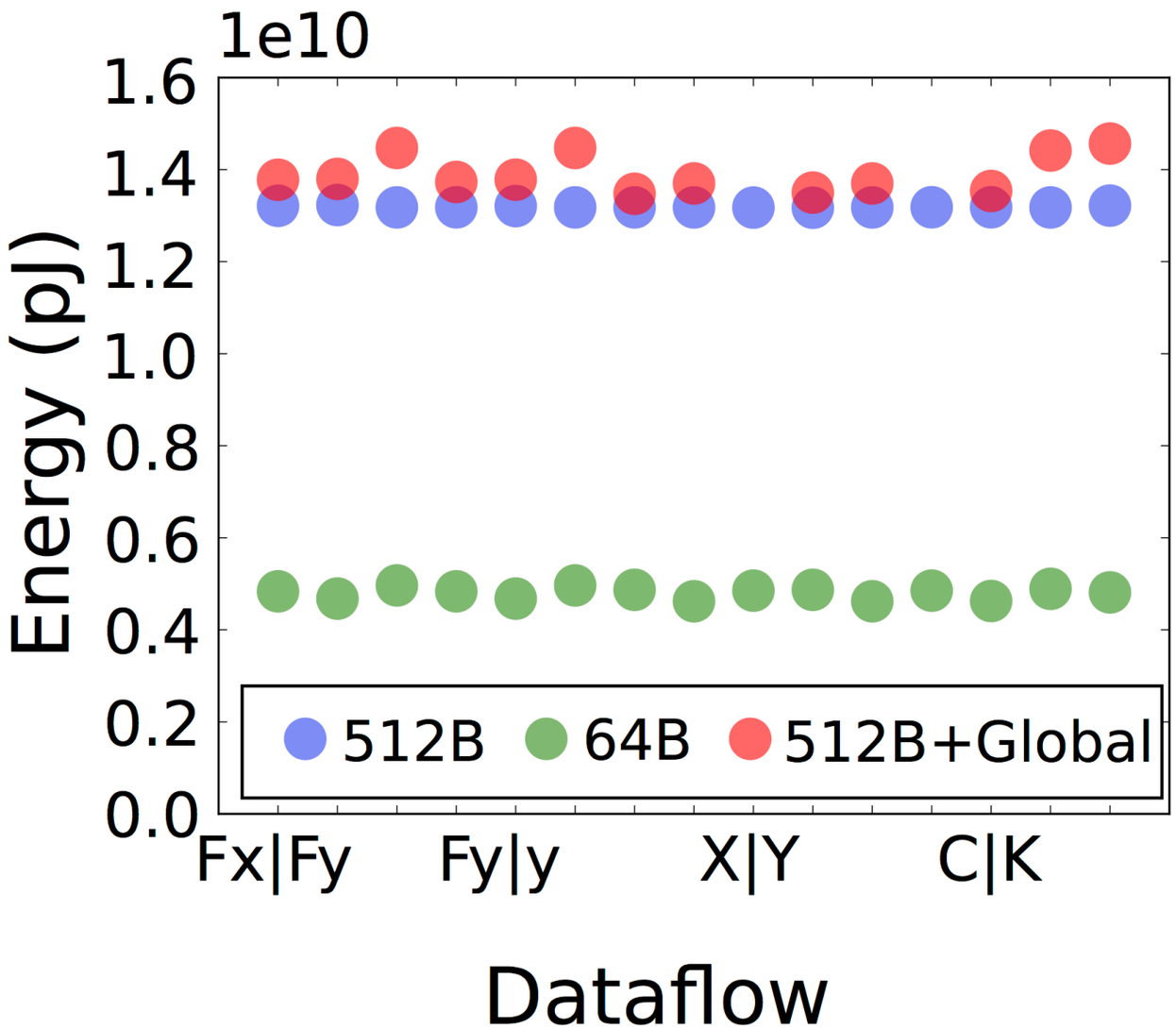


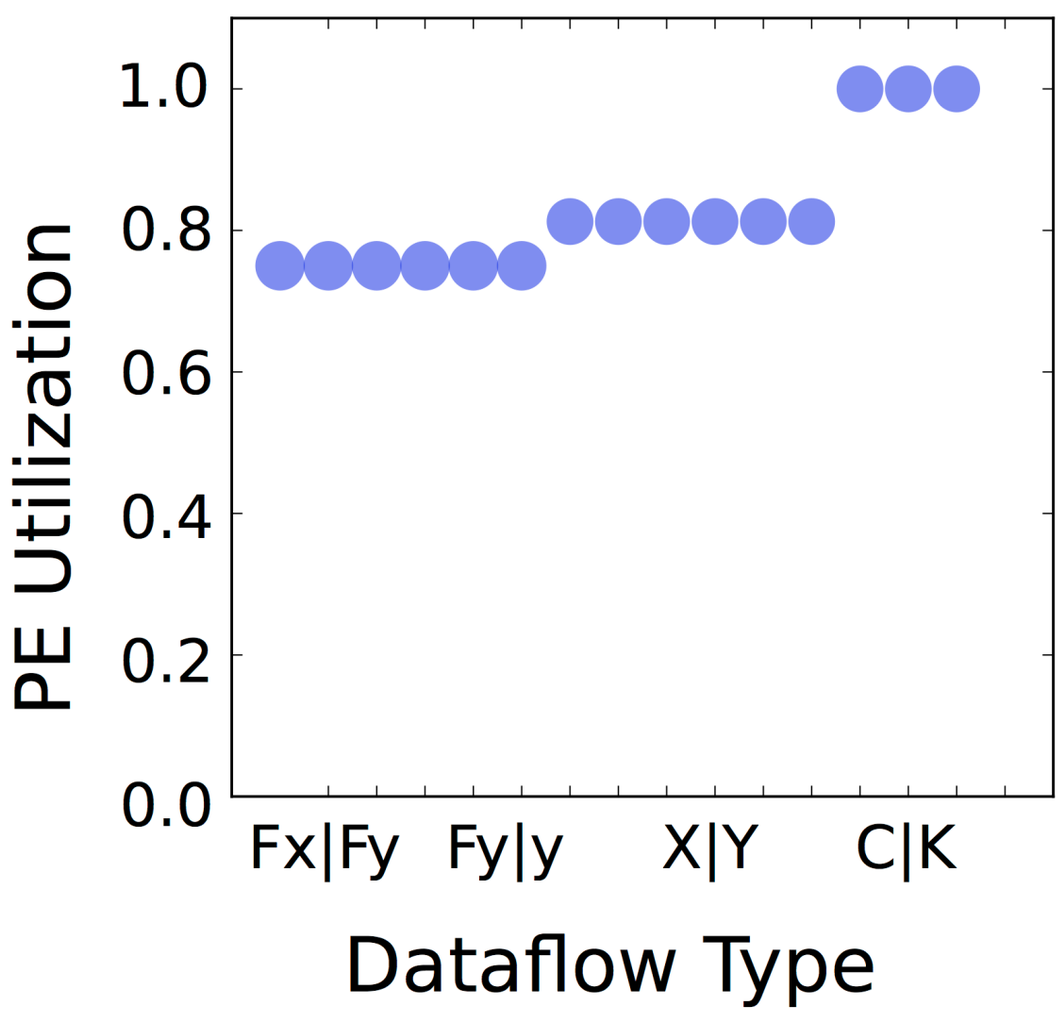
Experiments

- Using AlexNet with batch16
- PE number: 16 x 16
- Precision: 16b
- Global buffer: 128KB
- Register: 512B

Dataflow Design Space

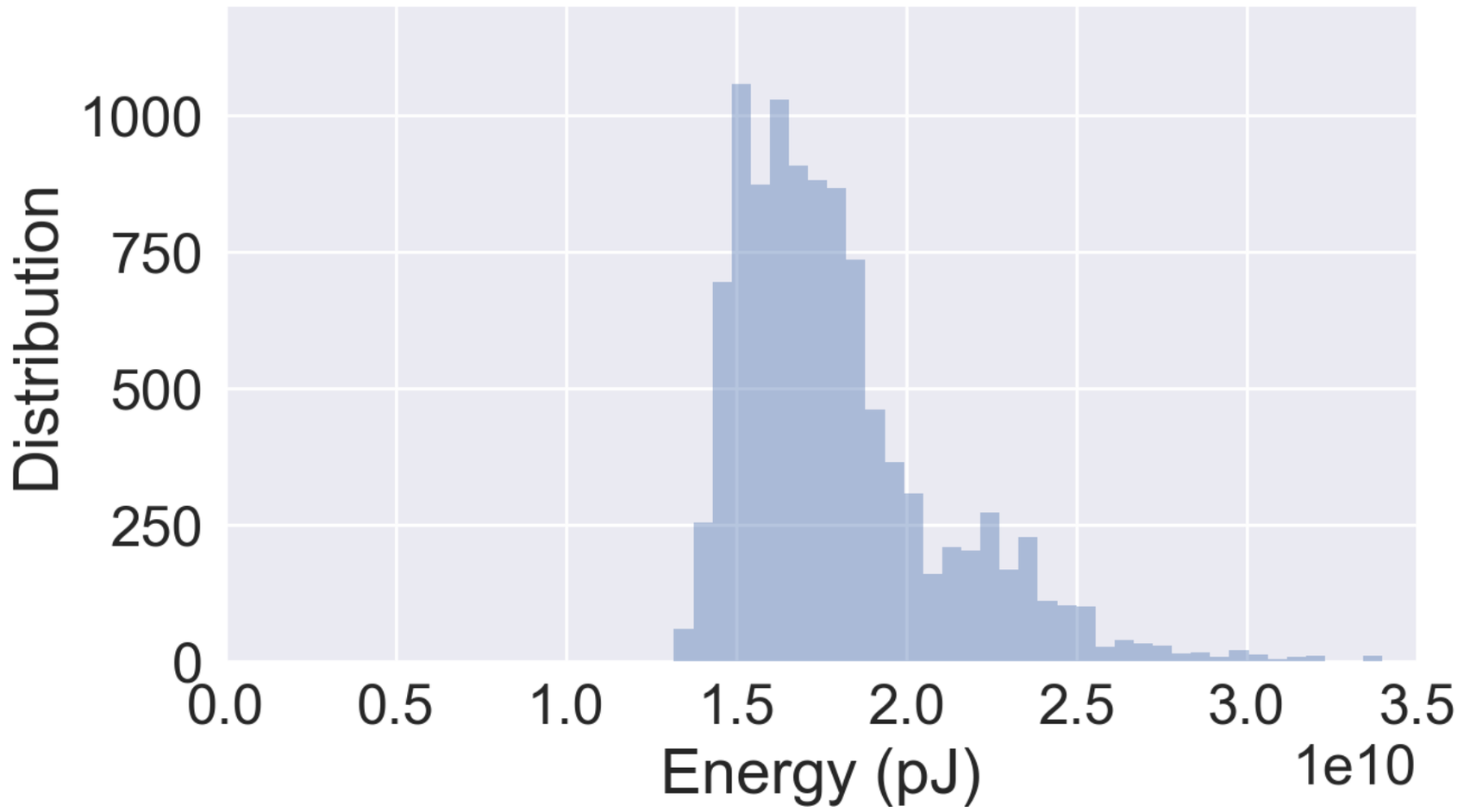
- Various data flows achieve similar energy efficiency
- Throughput difference is less than 25%





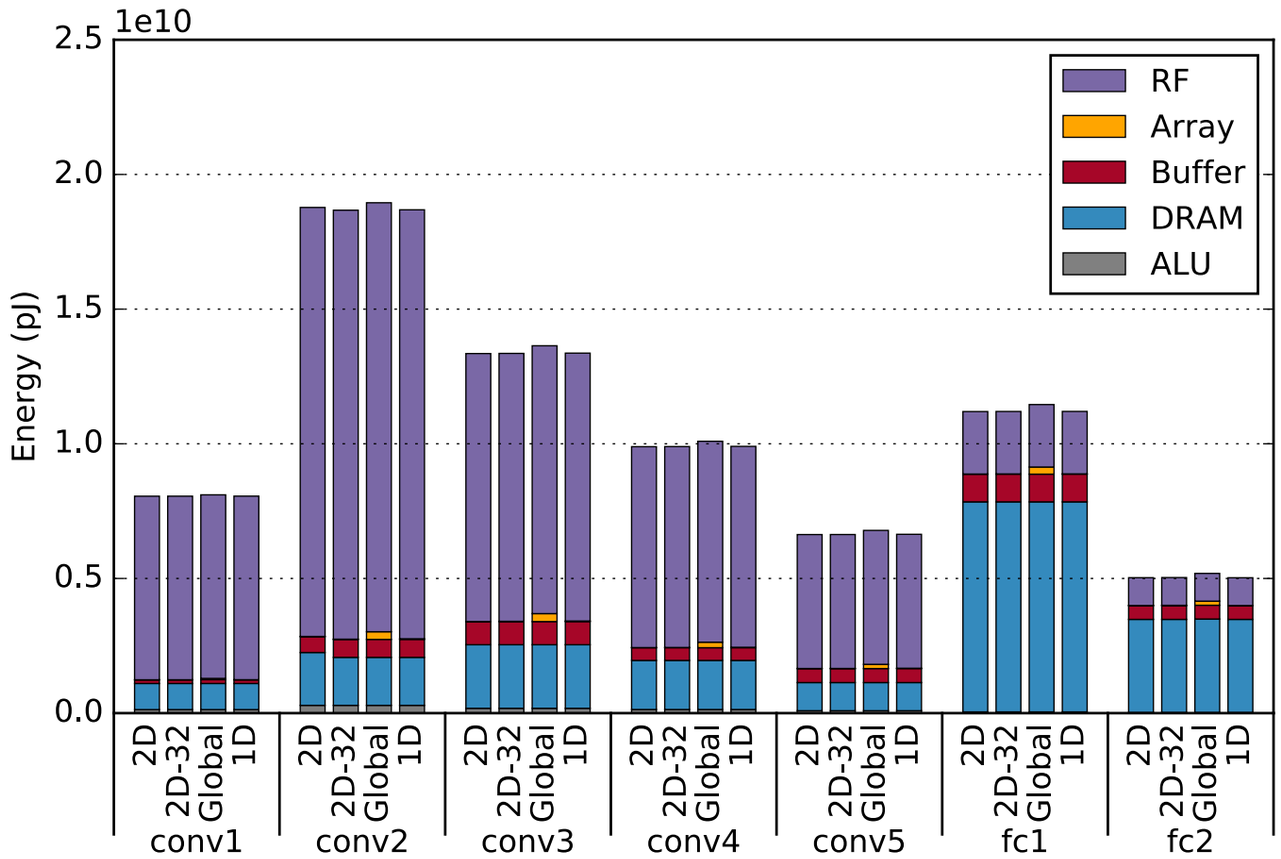
Blocking Design Space

- Blocking Choice has a large impact on energy efficiency

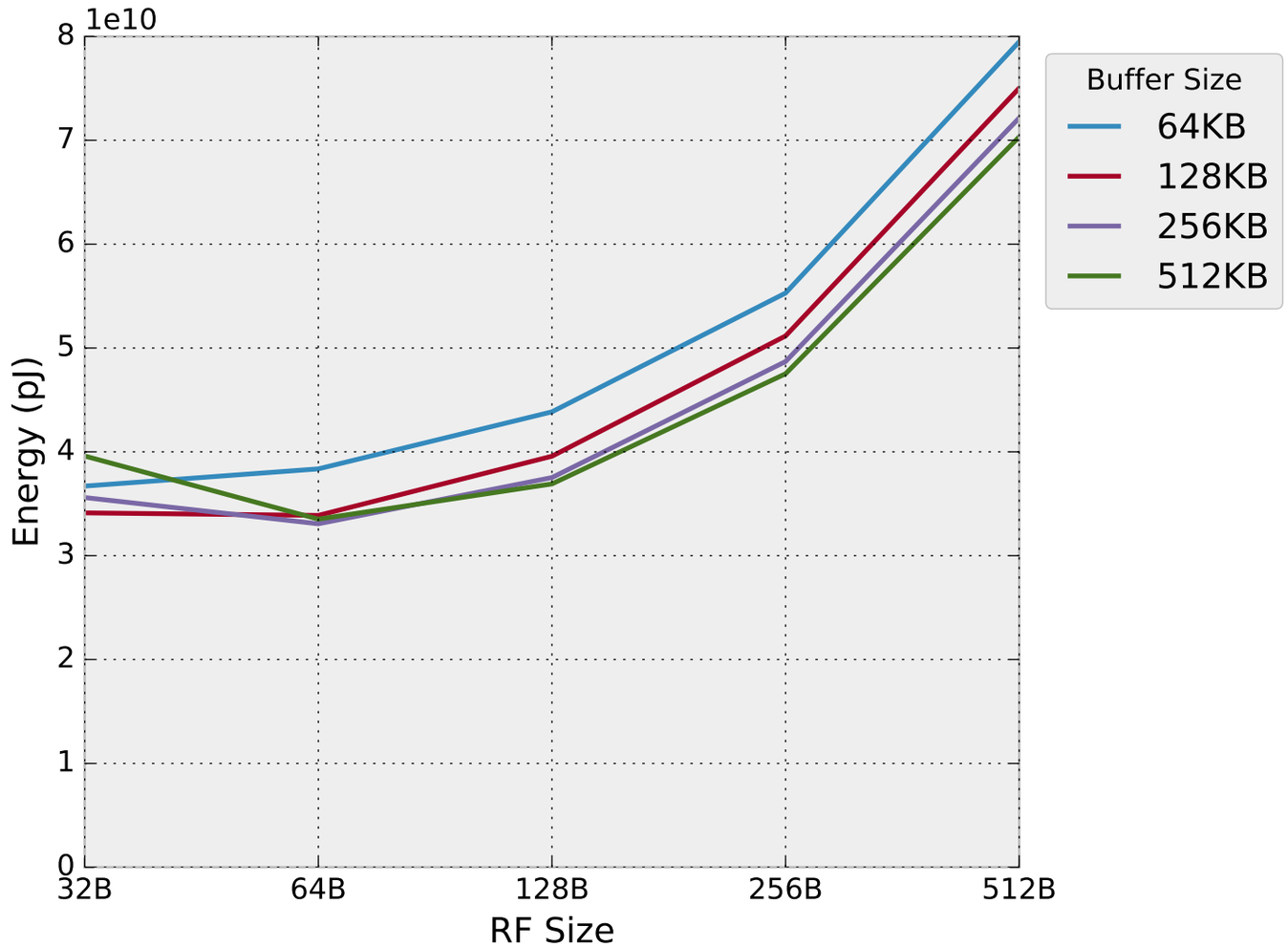


Dataflow Impact

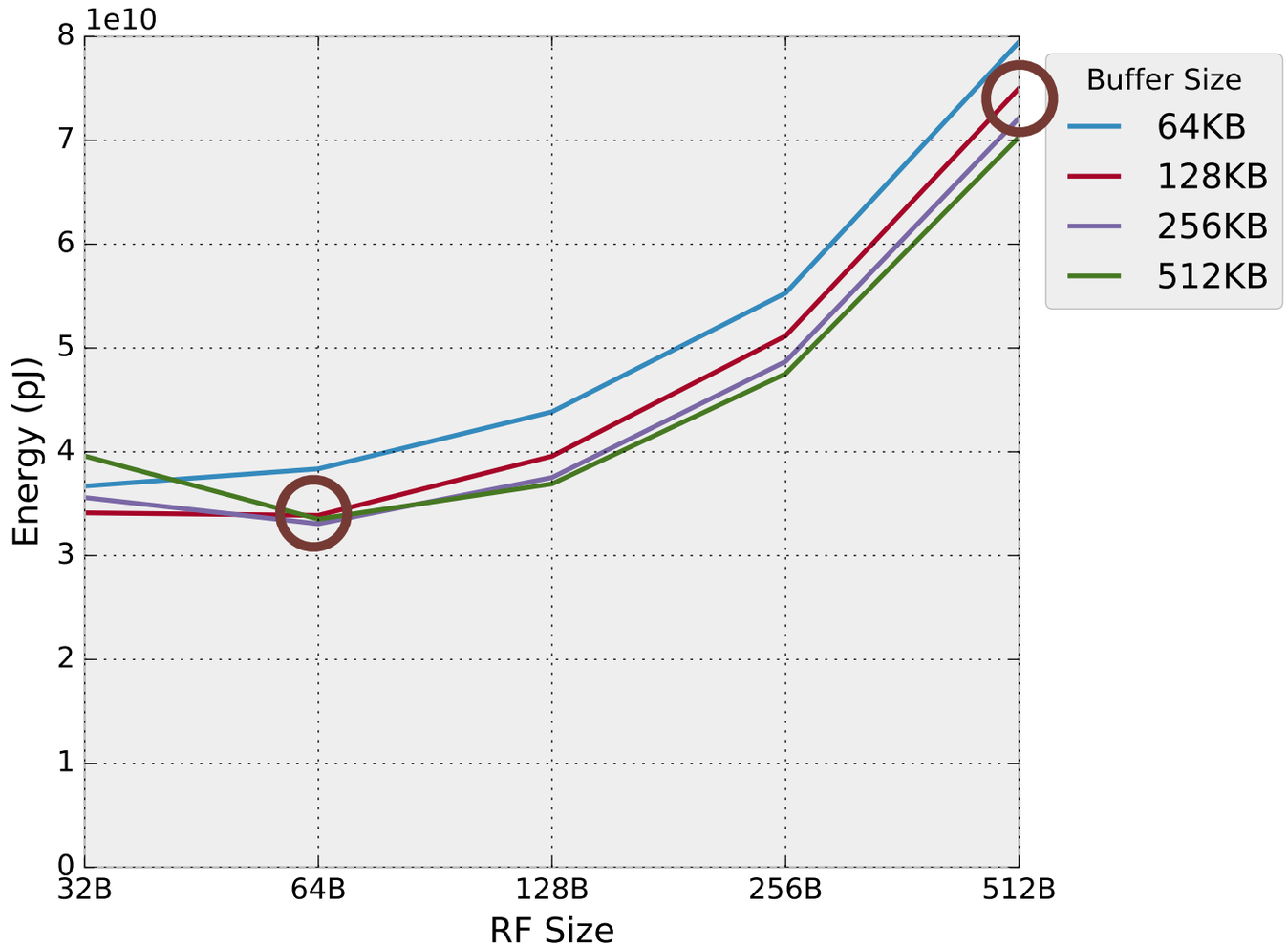
- Various data flows achieve similar energy efficiency
  - Even for different hardware configurations, layers, etc
- Register file energy dominates



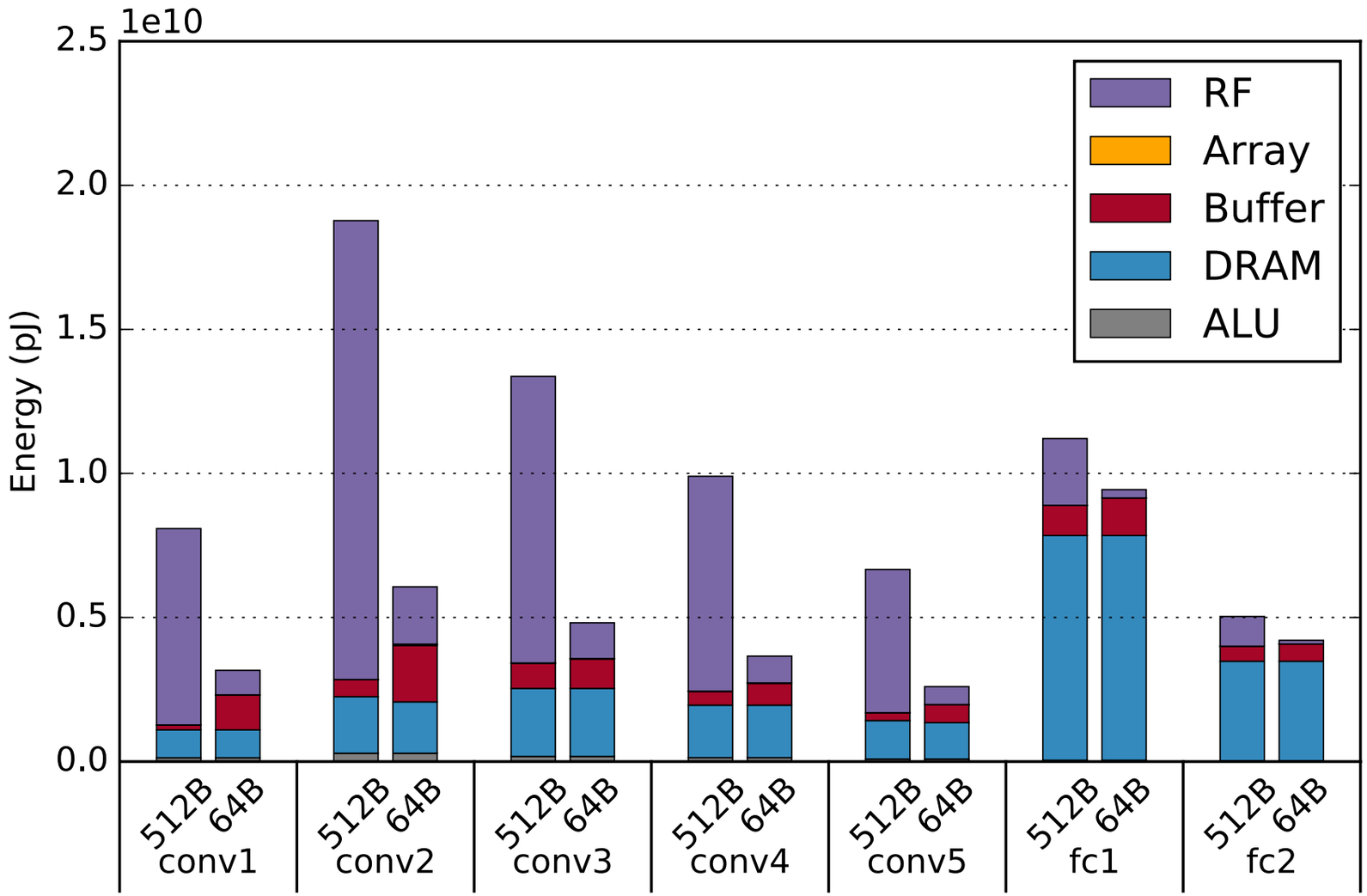
Memory Allocation Impact



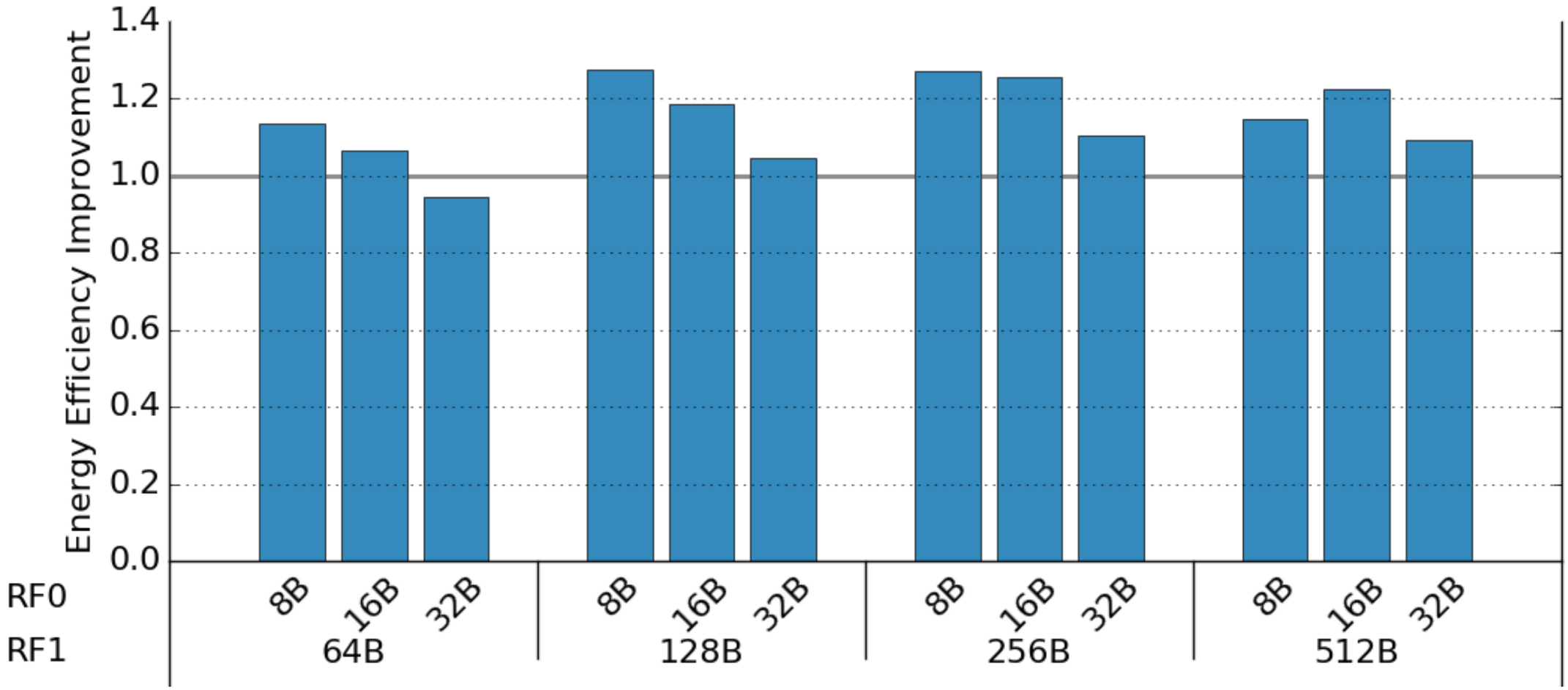
Memory Allocation Impact



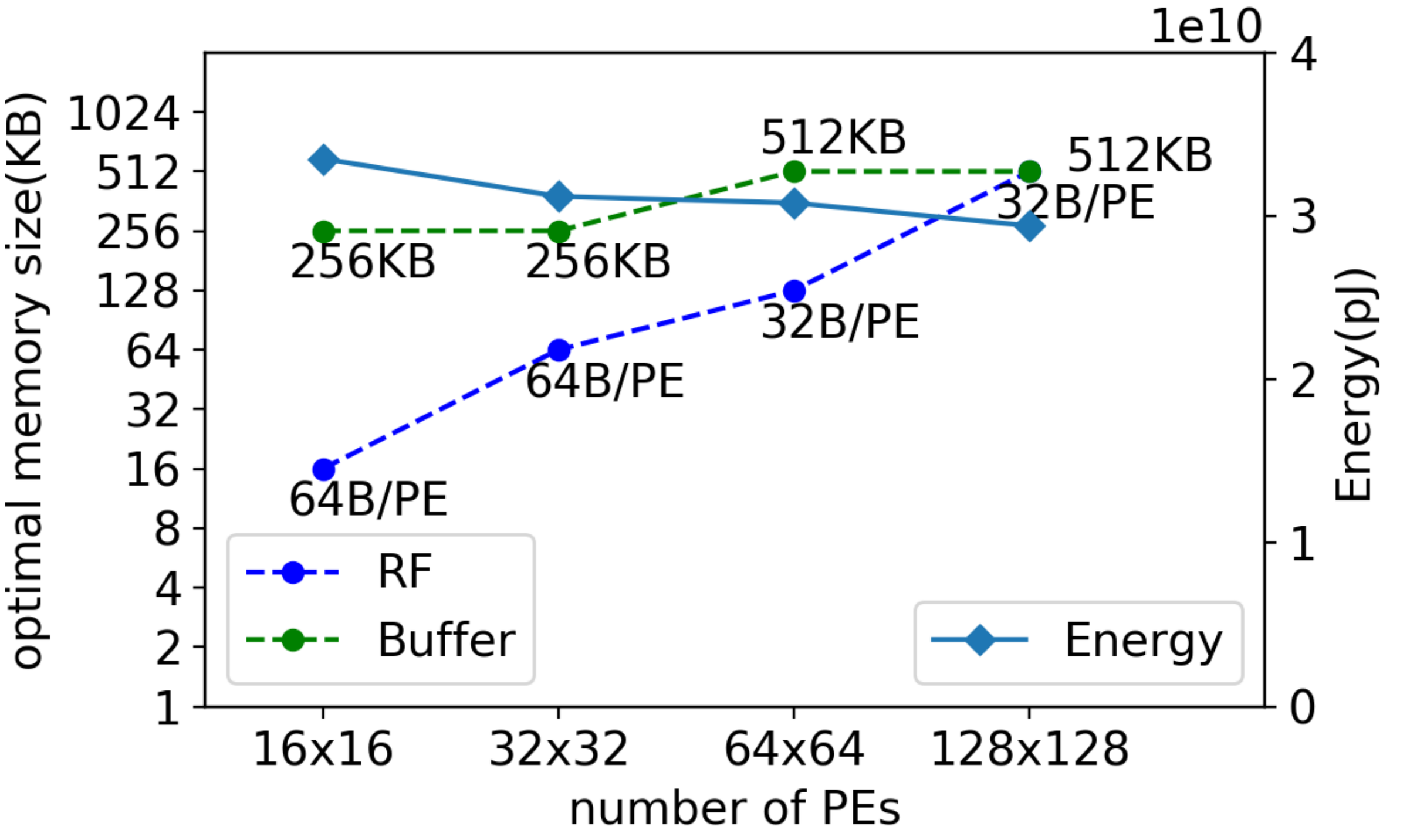
Energy Breakdown Comparison



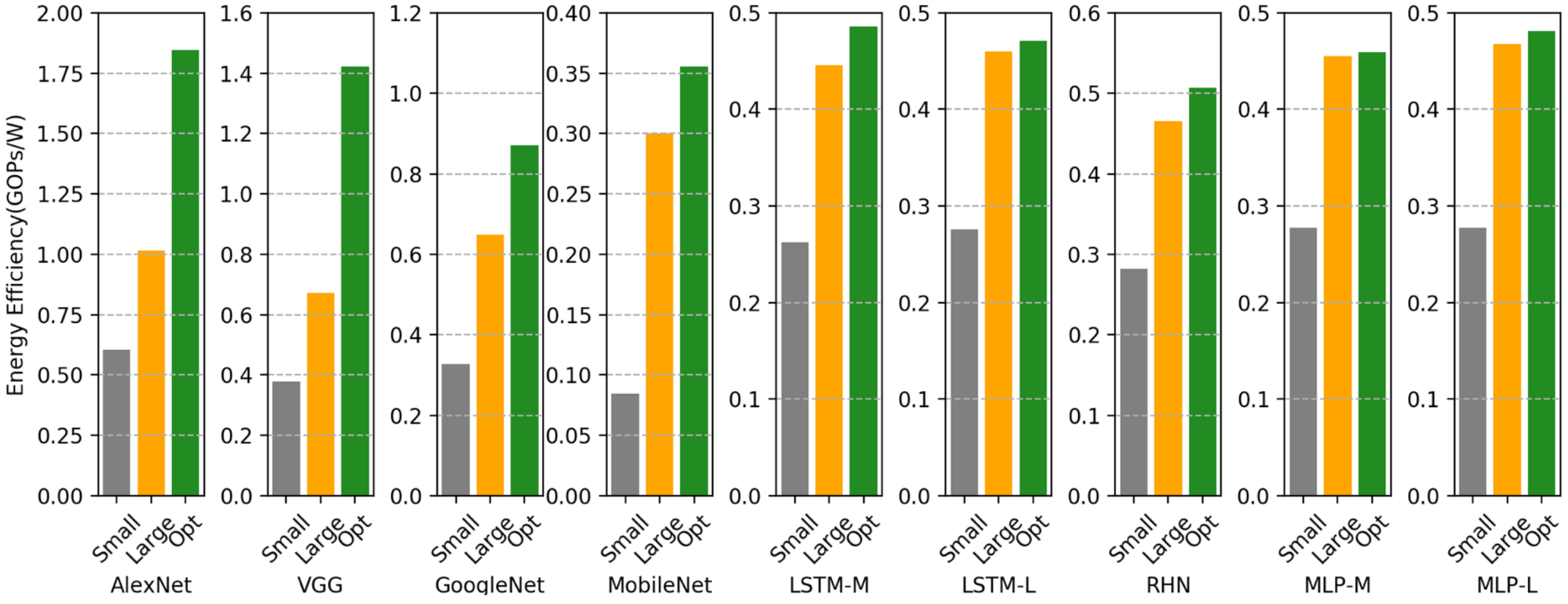
Two-level Register File



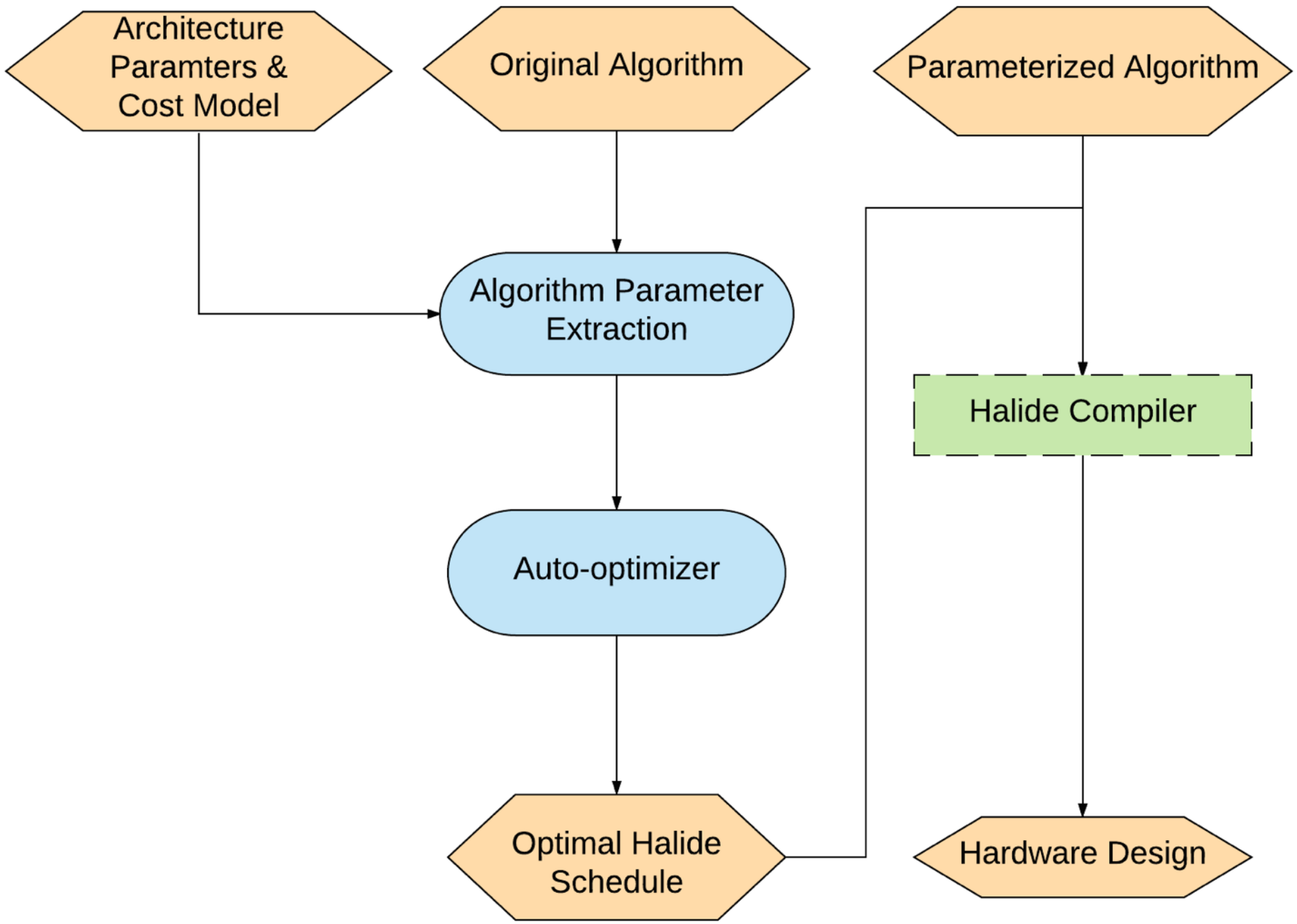
Varying Array Size



Auto-scheduler



Halide to Hardware Auto-scheduler



Thanks!

Q & A