Data Is All You Need for Fusion

MANYA BANSAL

High Performance code is about **Hardware**

```
#include <cblas.h>
#include <vector>


                Matrix Multiply
int main(..){
    std::vector<float> a;
    a.reserve(100);
    // Initialize other data
    // data-structures.
    cblas_sgemm(a.data()...);
    cblas_sgemm(....);

 return 0;
}
```

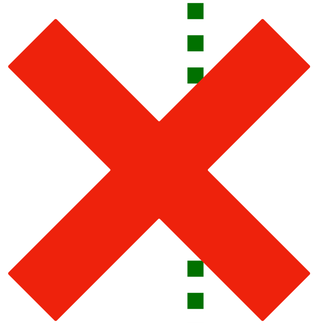
High Performance code is about **Hardware**

```
#include <cblas.h>
#include <vector>
```

Matrix Multiply

```
int main(..){
    std::vector<float> a;
    a.reserve(100);
    // Initialize other data
    // data-structures.
    cblas_sgemm(a.data()...);
    cblas_sgemm(....);

 return 0;
}
```

```
for (int i = 0; i < R1; i++) {
    for (int j = 0; j < C2; j++) {
        result[i][j] = 0;
        for (int k = 0; k < R2; k++) {
            result[i][j] += A[i][k] * B[k][j];
        }
    }
}
```



High Performance code is about **Hardware**

```cpp
#include <cblas.h>
#include <vector>

int main(..){
    std::vector<float> a;
    a.reserve(100);
    // Initialize other data
    // data-structures.
    cblas_sgemm(a.data()...);
    cblas_sgemm(....);

 return 0;
}
```

Matrix Multiply

```
    "3243431:\n\t"\
    "cmpq %8,%%r15; jb 3243433f;"\
    "3243432:\n\t"\
    COMPUTE_m8n4 "subq %8,%%r15; cmpq %8,%%r15; jnb 3243432b;"\
    "3243433:\n\t"\
    "cmpq $2,%%r15; jb 3243435f;"\
    "3243434:\n\t"\
    COMPUTE_m2n4 "subq $2,%%r15; cmpq $2,%%r15; jnb 3243434b;"\
    "3243435:\n\t"\
    "movq %%r14,%1;"\

  ↪  :"+r"(a_ptr),"+r"(b_ptr),"+r"(c_ptr),"+r"(c_tmp),"+r"(ldc_in_bytes),"+r"(b_pref):"m"(K),"m"(M):"r10","r11","r12","r13","r14","r15","cc","memory",\
    "zmm0","zmm1","zmm2","zmm3","zmm4","zmm5","zmm6","zmm7","zmm8","zmm9","zmm10","zmm11","zmm12","zmm13","zmm14","zmm15",\

    ↪  "zmm16","zmm17","zmm18","zmm19","zmm20","zmm21","zmm22","zmm23","zmm24","zmm25","zmm26","zmm27","zmm28","zmm29","zmm30","zmm31");\
  a_ptr -= M * K; b_ptr += 4 * K; c_ptr += 4 * ldc - M;\
}

#define macro_n2 {\
  b_pref = b_ptr + 2 * K;\
  __asm__ __volatile__(\
    "movq %7,%%r15; movq %1,%%r14; movq %6,%%r11; salq $4,%%r11;"\
    "cmpq $24,%%r15; jb 3243231f;"\
    "3243230:\n\t"\
    COMPUTE_m24n2 "subq $24,%%r15; cmpq $24,%%r15; jnb 3243230b;"\
    "3243231:\n\t"\
    "cmpq $8,%%r15; jb 3243233f;"\
    "3243232:\n\t"\
    COMPUTE_m8n2 "subq $8,%%r15; cmpq $8,%%r15; jnb 3243232b;"\
    "3243233:\n\t"\
    "cmpq $2,%%r15; jb 3243235f;"\
    "3243234:\n\t"\
    COMPUTE_m2n2 "subq $2,%%r15; cmpq $2,%%r15; jnb 3243234b;"\
    "3243235:\n\t"\
    "movq %%r14,%1;"\

  ↪  :"+r"(a_ptr),"+r"(b_ptr),"+r"(c_ptr),"+r"(c_tmp),"+r"(ldc_in_bytes),"+r"(b_pref):"m"(K),"m"(M):"r10","r11","r12","r13","r14","r15","cc","memory",\
    "zmm0","zmm1","zmm2","zmm3","zmm4","zmm5","zmm6","zmm7","zmm8","zmm9","zmm10","zmm11","zmm12","zmm13","zmm14","zmm15",\

    ↪  "zmm16","zmm17","zmm18","zmm19","zmm20","zmm21","zmm22","zmm23","zmm24","zmm25","zmm26","zmm27","zmm28","zmm29","zmm30","zmm31");\
  a_ptr -= M * K; b_ptr += 2 * K; c_ptr += 2 * ldc - M;\
}


void mydgemm_cpu_v18(\
    int M, \
    int N, \
    int K, \
    double alpha, \
    double *A, \
    int LDA, \
    double *B, \
    int LDB, \
    double beta, \
    double *C, \
    int LDC)\
{
    int i,j,k;

    if (beta != 1.0) scale_c_k18(C,M,N,LDC,beta);
    if (alpha == 0.||K==0) return;
    int M4,N8=N&-8,K4;
    double *a_buffer = (double *)aligned_alloc(4096,K_BLOCKING*M_BLOCKING*sizeof(double));
    double *b_buffer = (double *)aligned_alloc(4096,K_BLOCKING*N_BLOCKING*sizeof(double));
    int second_m_count,second_n_count,second_m_inc,second_n_inc;
    int m_count,n_count,k_count;
    int m_inc,n_inc,k_inc;
    for (n_count=0;n_count<N;n_count+=n_inc){
      n_inc=(N-n_count>N_BLOCKING)?N_BLOCKING:N-n_count;
      for (k_count=0;k_count<K;k_count+=k_inc){
          k_inc=(K-k_count>K_BLOCKING)?K_BLOCKING:K-k_count;
          m_inc=M>M_BLOCKING?M_BLOCKING:M;
          packing_a_k18(alpha,A+k_count*LDA,a_buffer,LDA,m_inc,k_inc);
          for (second_n_count=n_count;second_n_count<n_count+n_inc;second_n_count+=second_n_inc){
            second_n_inc=n_count+n_inc-second_n_count>16?16:n_count+n_inc-second_n_count;
            packing_b_k18(B+k_count+second_n_count*LDB,b_buffer+(second_n_count-n_count)*k_inc,LDB,k_inc,second_n_inc);

            ↪  macro_kernel_k18(a_buffer,b_buffer+(second_n_count-n_count)*k_inc,m_inc,second_n_inc,k_inc,&C(0,second_n_count),LDC);
          }
          for (m_count=m_inc;m_count<M;m_count+=m_inc){
              m_inc=(M-m_count>M_BLOCKING)?M_BLOCKING:M-m_count;
              packing_a_k18(alpha,A+m_count+k_count*LDA,a_buffer,LDA,m_inc,k_inc);
              macro_kernel_k18(a_buffer,b_buffer,m_inc,n_inc,k_inc,&C(m_count,n_count),LDC);
          }
```

High Performance code is about **Hardware**

```cpp
#include <cblas.h>
#include <vector>

int main(..){
    std::vector<float> a;
    a.reserve(100);
    // Initialize other data
    // data-structures.
    cblas_sgemm(a.data()...);
    cblas_sgemm(....);

 return 0;
}
```

Matrix Multiply

Tiled

```
    "3243431:\n\t"\
    "cmpq %8,%%r15; jb 3243433f;"\
    "3243432:\n\t"\
    COMPUTE_m8n4 "subq %8,%%r15; cmpq %8,%%r15; jnb 3243432b;"\
    "3243433:\n\t"\
    "cmpq $2,%%r15; jb 3243435f;"\
    "3243434:\n\t"\
    COMPUTE_m2n4 "subq $2,%%r15; cmpq $2,%%r15; jnb 3243434b;"\
    "3243435:\n\t"\
    "movq %%r14,%1;"\

  ↪  :"+r"(a_ptr),"+r"(b_ptr),"+r"(c_ptr),"+r"(c_tmp),"+r"(ldc_in_bytes),"+r"(b_pref):"m"(K),"m"(M):"r10","r11
    "zmm0","zmm1","zmm2","zmm3","zmm4","zmm5","zmm6","zmm7","zmm8","zmm9","zmm10","zmm11","zmm12","zmm13","zmm1

    ↪  "zmm16","zmm17","zmm18","zmm19","zmm20","zmm21","zmm22","zmm23","zmm24","zmm25","zmm26","zmm27","zmm28"
  a_ptr -= M * K; b_ptr += 4 * K; c_ptr += 4 * ldc - M;\
}

#define macro_n2 {\
  b_pref = b_ptr + 2 * K;\
  __asm__ __volatile__(\
    "movq %7,%%r15; movq %1,%%r14; movq %6,%%r11; salq $4,%%r11;"\
    "cmpq $24,%%r15; jb 3243231f;"\
    "3243230:\n\t"\
    COMPUTE_m24n2 "subq $24,%%r15; cmpq $24,%%r15; jnb 3243230b;"\
    "3243231:\n\t"\
    "cmpq $8,%%r15; jb 3243233f;"\
    "3243232:\n\t"\
    COMPUTE_m8n2 "subq $8,%%r15; cmpq $8,%%r15; jnb 3243232b;"\
    "3243233:\n\t"\
    "cmpq $2,%%r15; jb 3243235f;"\
    "3243234:\n\t"\
    COMPUTE_m2n2 "subq $2,%%r15; cmpq $2,%%r15; jnb 3243234b;"\
    "3243235:\n\t"\
    "movq %%r14,%1;"\

  ↪  :"+r"(a_ptr),"+r"(b_ptr),"+r"(c_ptr),"+r"(c_tmp),"+r"(ldc_in_bytes),"+r"(b_pref):"m"(K),"m"(M):"r10","r11","r12","r13","r14","r15","cc","memory",\
    "zmm0","zmm1","zmm2","zmm3","zmm4","zmm5","zmm6","zmm7","zmm8","zmm9","zmm10","zmm11","zmm12","zmm13","zmm14","zmm15",\

    ↪  "zmm16","zmm17","zmm18","zmm19","zmm20","zmm21","zmm22","zmm23","zmm24","zmm25","zmm26","zmm27","zmm28","zmm29","zmm30","zmm31");\
  a_ptr -= M * K; b_ptr += 2 * K; c_ptr += 2 * ldc - M;\
}


void mydgemm_cpu_v18(\
    int M, \
    int N, \
    int K, \
    double alpha, \
    double *A, \
    int LDA, \
    double *B, \
    int LDB, \
    double beta, \
    double *C, \
    int LDC)\
{
    int i,j,k;

    if (beta != 1.0) scale_c_k18(C,M,N,LDC,beta);
    if (alpha == 0.||K==0) return;
    int M4,N8=N&-8,K4;
    double *a_buffer = (double *)aligned_alloc(4096,K_BLOCKING*M_BLOCKING*sizeof(double));
    double *b_buffer = (double *)aligned_alloc(4096,K_BLOCKING*N_BLOCKING*sizeof(double));
    int second_m_count,second_n_count,second_m_inc,second_n_inc;
    int m_count,n_count,k_count;
    int m_inc,n_inc,k_inc;
    for (n_count=0;n_count<N;n_count+=n_inc){
      n_inc=(N-n_count>N_BLOCKING)?N_BLOCKING:N-n_count;
      for (k_count=0;k_count<K;k_count+=k_inc){
          k_inc=(K-k_count>K_BLOCKING)?K_BLOCKING:K-k_count;
          m_inc=M>M_BLOCKING?M_BLOCKING:M;
          packing_a_k18(alpha,A+k_count*LDA,a_buffer,LDA,m_inc,k_inc);
          for (second_n_count=n_count;second_n_count<n_count+n_inc;second_n_count+=second_n_inc){
            second_n_inc=n_count+n_inc-second_n_count>16?16:n_count+n_inc-second_n_count;
            packing_b_k18(B+k_count+second_n_count*LDB,b_buffer+(second_n_count-n_count)*k_inc,LDB,k_inc,second_n_inc);

            ↪  macro_kernel_k18(a_buffer,b_buffer+(second_n_count-n_count)*k_inc,m_inc,second_n_inc,k_inc,&C(0,second_n_count),LDC);
          }
          for (m_count=m_inc;m_count<M;m_count+=m_inc){
              m_inc=(M-m_count>M_BLOCKING)?M_BLOCKING:M-m_count;
              packing_a_k18(alpha,A+m_count+k_count*LDA,a_buffer,LDA,m_inc,k_inc);
              macro_kernel_k18(a_buffer,b_buffer,m_inc,n_inc,k_inc,&C(m_count,n_count),LDC);
          }
```

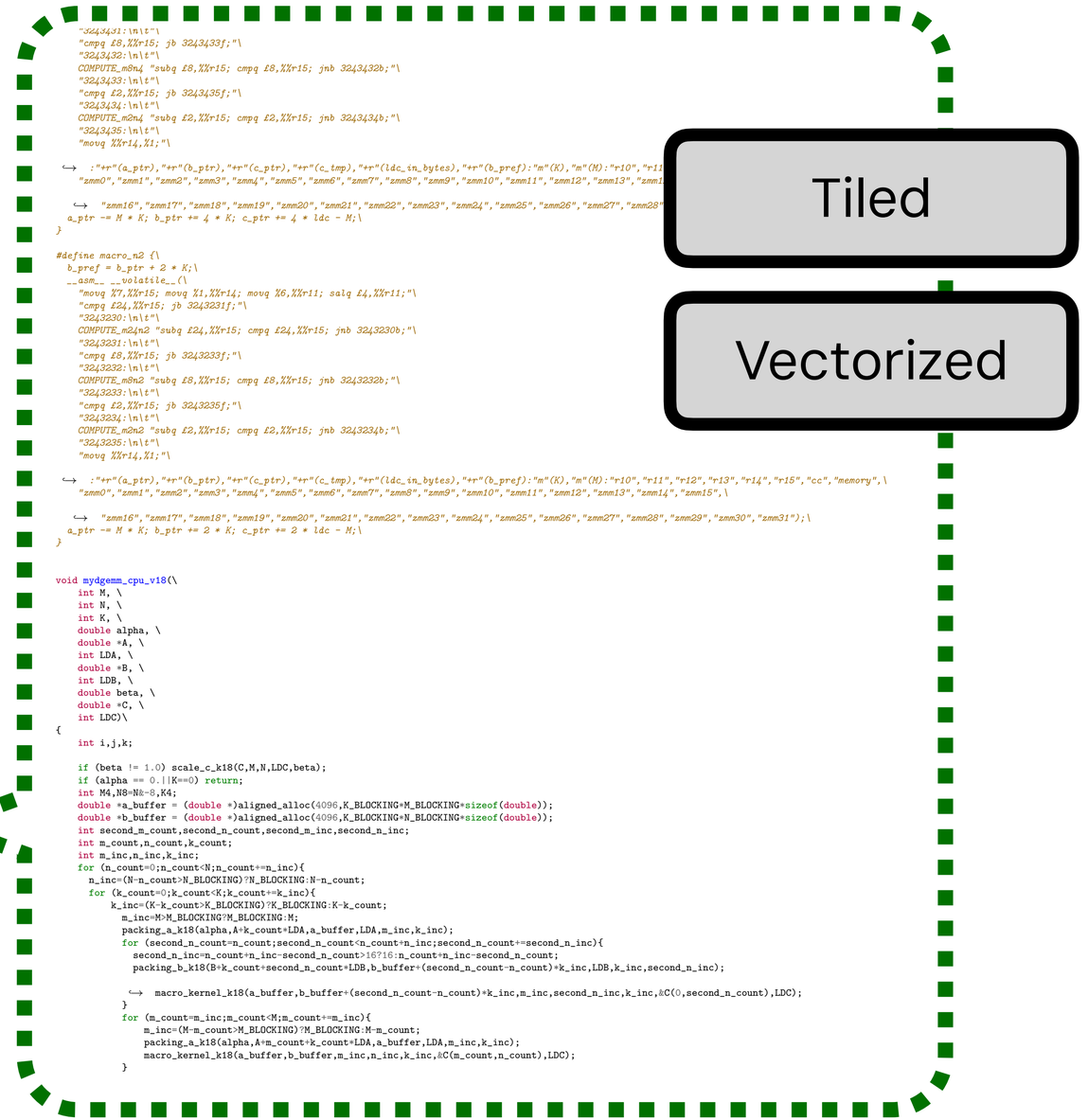
High Performance code is about **Hardware**

```cpp
#include <cblas.h>
#include <vector>

int main(..){
    std::vector<float> a;
    a.reserve(100);
    // Initialize other data
    // data-structures.
    cblas_sgemm(a.data()...);
    cblas_sgemm(....);

 return 0;
}
```

Matrix Multiply

Tiled

Vectorized



```
    "3243431:\n\t"\
    "cmpq %8,%%r15; jb 3243433f;"\
    "3243432:\n\t"\
    COMPUTE_m8n4 "subq $8,%%r15; cmpq $8,%%r15; jnb 3243432b;"\
    "3243433:\n\t"\
    "cmpq $2,%%r15; jb 3243435f;"\
    "3243434:\n\t"\
    COMPUTE_m2n4 "subq $2,%%r15; cmpq $2,%%r15; jnb 3243434b;"\
    "3243435:\n\t"\
    "movq %%r14,%1;"\

  ↪  :"+r"(a_ptr),"+r"(b_ptr),"+r"(c_ptr),"+r"(c_tmp),"+r"(ldc_in_bytes),"+r"(b_pref):"m"(K),"m"(M):"r10","r11
    "zmm0","zmm1","zmm2","zmm3","zmm4","zmm5","zmm6","zmm7","zmm8","zmm9","zmm10","zmm11","zmm12","zmm13","zmm1

    ↪  "zmm16","zmm17","zmm18","zmm19","zmm20","zmm21","zmm22","zmm23","zmm24","zmm25","zmm26","zmm27","zmm28"
  a_ptr -= M * K; b_ptr += 4 * K; c_ptr += 4 * ldc - M;\
}

#define macro_n2 {\
  b_pref = b_ptr + 2 * K;\
  __asm__ __volatile__(\
    "movq %7,%%r15; movq %1,%%r14; movq %6,%%r11; salq $4,%%r11;"\
    "cmpq $24,%%r15; jb 3243231f;"\
    "3243230:\n\t"\
    COMPUTE_m24n2 "subq $24,%%r15; cmpq $24,%%r15; jnb 3243230b;"\
    "3243231:\n\t"\
    "cmpq $8,%%r15; jb 3243233f;"\
    "3243232:\n\t"\
    COMPUTE_m8n2 "subq $8,%%r15; cmpq $8,%%r15; jnb 3243232b;"\
    "3243233:\n\t"\
    "cmpq $2,%%r15; jb 3243235f;"\
    "3243234:\n\t"\
    COMPUTE_m2n2 "subq $2,%%r15; cmpq $2,%%r15; jnb 3243234b;"\
    "3243235:\n\t"\
    "movq %%r14,%1;"\

  ↪  :"+r"(a_ptr),"+r"(b_ptr),"+r"(c_ptr),"+r"(c_tmp),"+r"(ldc_in_bytes),"+r"(b_pref):"m"(K),"m"(M):"r10","r11","r12","r13","r14","r15","cc","memory",\
    "zmm0","zmm1","zmm2","zmm3","zmm4","zmm5","zmm6","zmm7","zmm8","zmm9","zmm10","zmm11","zmm12","zmm13","zmm14","zmm15",\

    ↪  "zmm16","zmm17","zmm18","zmm19","zmm20","zmm21","zmm22","zmm23","zmm24","zmm25","zmm26","zmm27","zmm28","zmm29","zmm30","zmm31");\
  a_ptr -= M * K; b_ptr += 2 * K; c_ptr += 2 * ldc - M;\
}


void mydgemm_cpu_v18(\
    int M, \
    int N, \
    int K, \
    double alpha, \
    double *A, \
    int LDA, \
    double *B, \
    int LDB, \
    double beta, \
    double *C, \
    int LDC)\
{
    int i,j,k;

    if (beta != 1.0) scale_c_k18(C,M,N,LDC,beta);
    if (alpha == 0.||K==0) return;
    int M4,N8=N&-8,K4;
    double *a_buffer = (double *)aligned_alloc(4096,K_BLOCKING*M_BLOCKING*sizeof(double));
    double *b_buffer = (double *)aligned_alloc(4096,K_BLOCKING*N_BLOCKING*sizeof(double));
    int second_m_count,second_n_count,second_m_inc,second_n_inc;
    int m_count,n_count,k_count;
    int m_inc,n_inc,k_inc;
    for (n_count=0;n_count<N;n_count+=n_inc){
      n_inc=(N-n_count>N_BLOCKING)?N_BLOCKING:N-n_count;
      for (k_count=0;k_count<K;k_count+=k_inc){
          k_inc=(K-k_count>K_BLOCKING)?K_BLOCKING:K-k_count;
          m_inc=M>M_BLOCKING?M_BLOCKING:M;
          packing_a_k18(alpha,A+k_count*LDA,a_buffer,LDA,m_inc,k_inc);
          for (second_n_count=n_count;second_n_count<n_count+n_inc;second_n_count+=second_n_inc){
            second_n_inc=n_count+n_inc-second_n_count>16?16:n_count+n_inc-second_n_count;
            packing_b_k18(B+k_count+second_n_count*LDB,b_buffer+(second_n_count-n_count)*k_inc,LDB,k_inc,second_n_inc);

            ↪  macro_kernel_k18(a_buffer,b_buffer+(second_n_count-n_count)*k_inc,m_inc,second_n_inc,k_inc,&C(0,second_n_count),LDC);
          }
          for (m_count=m_inc;m_count<M;m_count+=m_inc){
              m_inc=(M-m_count>M_BLOCKING)?M_BLOCKING:M-m_count;
              packing_a_k18(alpha,A+m_count+k_count*LDA,a_buffer,LDA,m_inc,k_inc);
              macro_kernel_k18(a_buffer,b_buffer,m_inc,n_inc,k_inc,&C(m_count,n_count),LDC);
          }
```

High Performance code is about **Hardware**

```
#include <cblas.h>
#include <vector>

Matrix Multiply

int main(..){
    std::vector<float> a;
    a.reserve(100);
    // Initialize other data
    // data-structures.
    cblas_sgemm(a.data()...);
    cblas_sgemm(....);

 return 0;
}
```

```
"3243431:\n\t"\
"cmpq £8,%%r15; jb 3243433f;"\
"3243432:\n\t"\
COMPUTE_m8n4 "subq £8,%%r15; cmpq £8,%%r15; jnb 3243432b;"\
"3243433:\n\t"\
"cmpq £2,%%r15; jb 3243435f;"\
"3243434:\n\t"\
COMPUTE_m2n4 "subq £2,%%r15; cmpq £2,%%r15; jnb 3243434b;"\
"3243435:\n\t"\
"movq %%r14,%1;"\

 ↪  :"+r"(a_ptr),"+r"(b_ptr),"+r"(c_ptr),"+r"(c_tmp),"+r"(ldc_in_bytes),"+r"(b_pref):"m"(K),"m"(M):"r10","r11
   "zmm0","zmm1","zmm2","zmm3","zmm4","zmm5","zmm6","zmm7","zmm8","zmm9","zmm10","zmm11","zmm12","zmm13","zmm1

   ↪  "zmm16","zmm17","zmm18","zmm19","zmm20","zmm21","zmm22","zmm23","zmm24","zmm25","zmm26","zmm27","zmm28"
a_ptr -= M * K; b_ptr += 4 * K; c_ptr += 4 * ldc - M;\
}

#define macro_n2 {\
  b_pref = b_ptr + 2 * K;\
  __asm__ __volatile__(\
    "movq %7,%%r15; movq %1,%%r14; movq %6,%%r11; salq £4,%%r11;"\
    "cmpq £24,%%r15; jb 3243231f;"\
    "3243230:\n\t"\
    COMPUTE_m24n2 "subq £24,%%r15; cmpq £24,%%r15; jnb 3243230b;"\
    "3243231:\n\t"\
    "cmpq £8,%%r15; jb 3243233f;"\
    "3243232:\n\t"\
    COMPUTE_m8n2 "subq £8,%%r15; cmpq £8,%%r15; jnb 3243232b;"\
    "3243233:\n\t"\
    "cmpq £2,%%r15; jb 3243235f;"\
    "3243234:\n\t"\
    COMPUTE_m2n2 "subq £2,%%r15; cmpq £2,%%r15; jnb 3243234b;"\
    "3243235:\n\t"\
    "movq %%r14,%1;"\

 ↪  :"+r"(a_ptr),"+r"(b_ptr),"+r"(c_ptr),"+r"(c_tmp),"+r"(ldc_in_bytes),"+r"(b_pref):"m"(K),"m"(M):"r10","r11
   "zmm0","zmm1","zmm2","zmm3","zmm4","zmm5","zmm6","zmm7","zmm8","zmm9","zmm10","zmm11","zmm12","zmm13","zmm1

   ↪  "zmm16","zmm17","zmm18","zmm19","zmm20","zmm21","zmm22","zmm23","zmm24","zmm25","zmm26","zmm27","zmm28"
  a_ptr -= M * K; b_ptr += 2 * K; c_ptr += 2 * ldc - M;\
}


void mydgemm_cpu_v18(\
    int M, \
    int N, \
    int K, \
    double alpha, \
    double *A, \
    int LDA, \
    double *B, \
    int LDB, \
    double beta, \
    double *C, \
    int LDC)\
{
    int i,j,k;

    if (beta != 1.0) scale_c_k18(C,M,N,LDC,beta);
    if (alpha == 0.||K==0) return;
    int M4,N8=N&~8,K4;
    double *a_buffer = (double *)aligned_alloc(4096,K_BLOCKING*M_BLOCKING*sizeof(double));
    double *b_buffer = (double *)aligned_alloc(4096,K_BLOCKING*N_BLOCKING*sizeof(double));
    int second_m_count,second_n_count,second_m_inc,second_n_inc;
    int m_count,n_count,k_count;
    int m_inc,n_inc,k_inc;
    for (n_count=0;n_count<N;n_count+=n_inc){
      n_inc=(N-n_count>N_BLOCKING)?N_BLOCKING:N-n_count;
      for (k_count=0;k_count<K;k_count+=k_inc){
          k_inc=(K-k_count>K_BLOCKING)?K_BLOCKING:K-k_count;
            m_inc=M>M_BLOCKING?M_BLOCKING:M;
            packing_a_k18(alpha,A+k_count*LDA,a_buffer,LDA,m_inc,k_inc);
            for (second_n_count=n_count;second_n_count<n_count+n_inc;second_n_count+=second_n_inc){
              second_n_inc=n_count+n_inc-second_n_count>16?16:n_count+n_inc-second_n_count;
              packing_b_k18(B+k_count+second_n_count*LDB,b_buffer+(second_n_count-n_count)*k_inc,LDB,k_inc,second_n_inc);

               ↪  macro_kernel_k18(a_buffer,b_buffer+(second_n_count-n_count)*k_inc,m_inc,second_n_inc,k_inc,&C(0,second_n_count),LDC);
            }
            for (m_count=m_inc;m_count<M;m_count+=m_inc){
                m_inc=(M-m_count>M_BLOCKING)?M_BLOCKING:M-m_count;
                packing_a_k18(alpha,A+m_count+k_count*LDA,a_buffer,LDA,m_inc,k_inc);
                macro_kernel_k18(a_buffer,b_buffer,m_inc,n_inc,k_inc,&C(m_count,n_count),LDC);
            }
```

Tiled

Vectorized

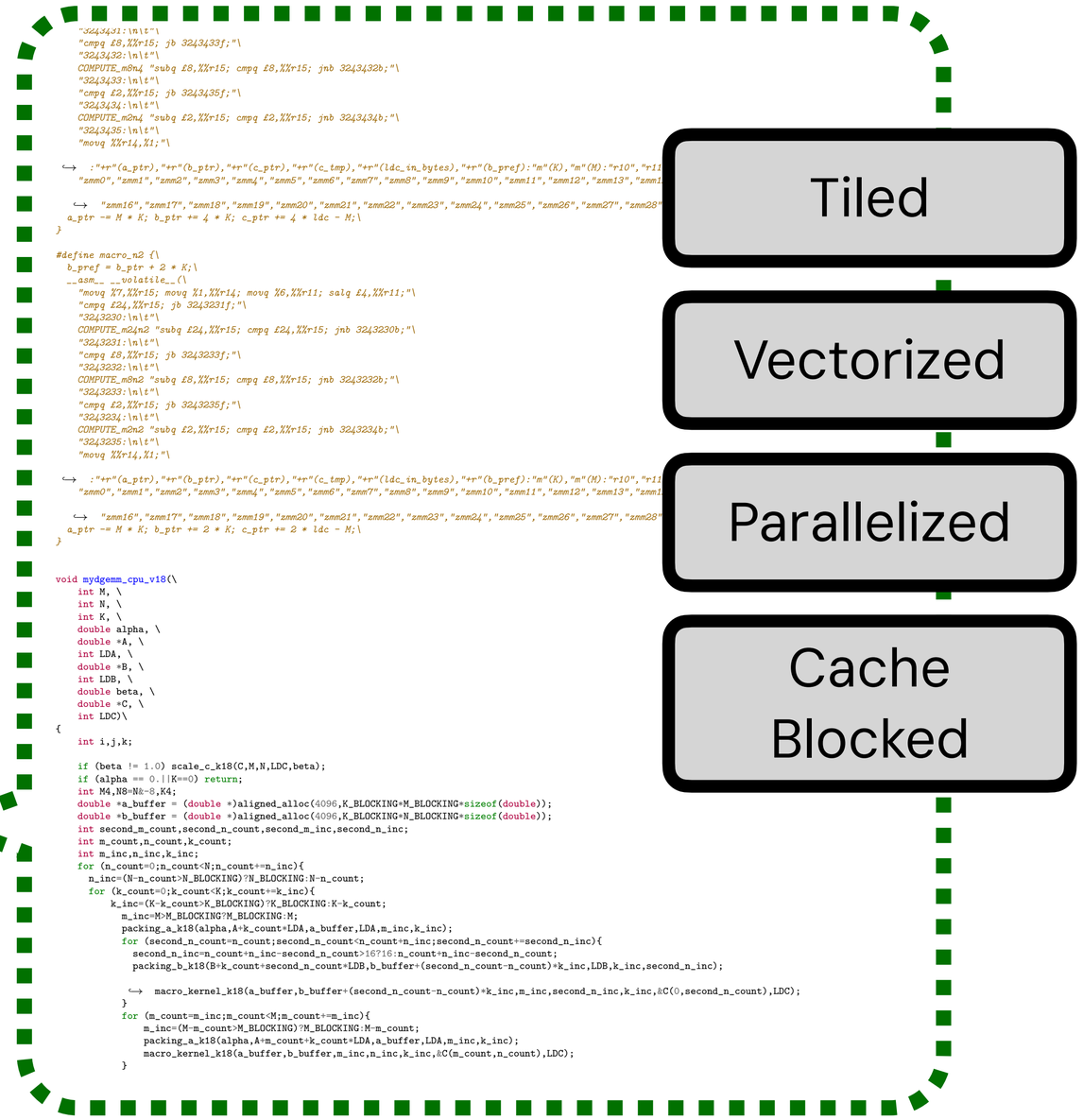
Parallelized

High Performance code is about **Hardware**

```
#include <cblas.h>
#include <vector>
```

Matrix Multiply

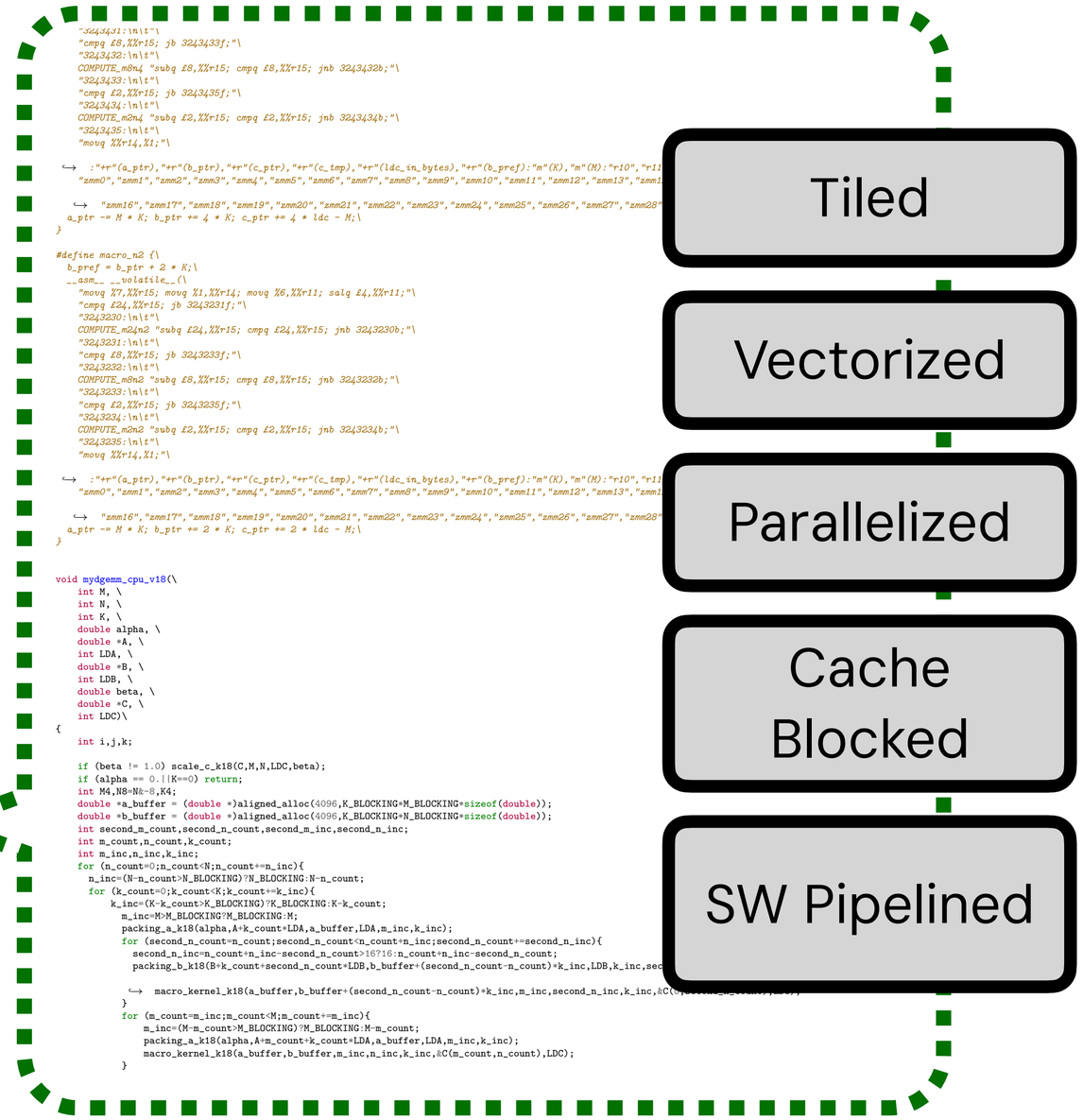
```
int main(..){
    std::vector<float> a;
    a.reserve(100);
    // Initialize other data
    // data-structures.
    cblas_sgemm(a.data()...);
    cblas_sgemm(....);

 return 0;
}
```



High Performance code is about **Hardware**

```
#include <cblas.h>
#include <vector>

int main(..){
    std::vector<float> a;
    a.reserve(100);
    // Initialize other data
    // data-structures.
    cblas_sgemm(a.data()...);
    cblas_sgemm(....);

 return 0;
}
```

Matrix Multiply

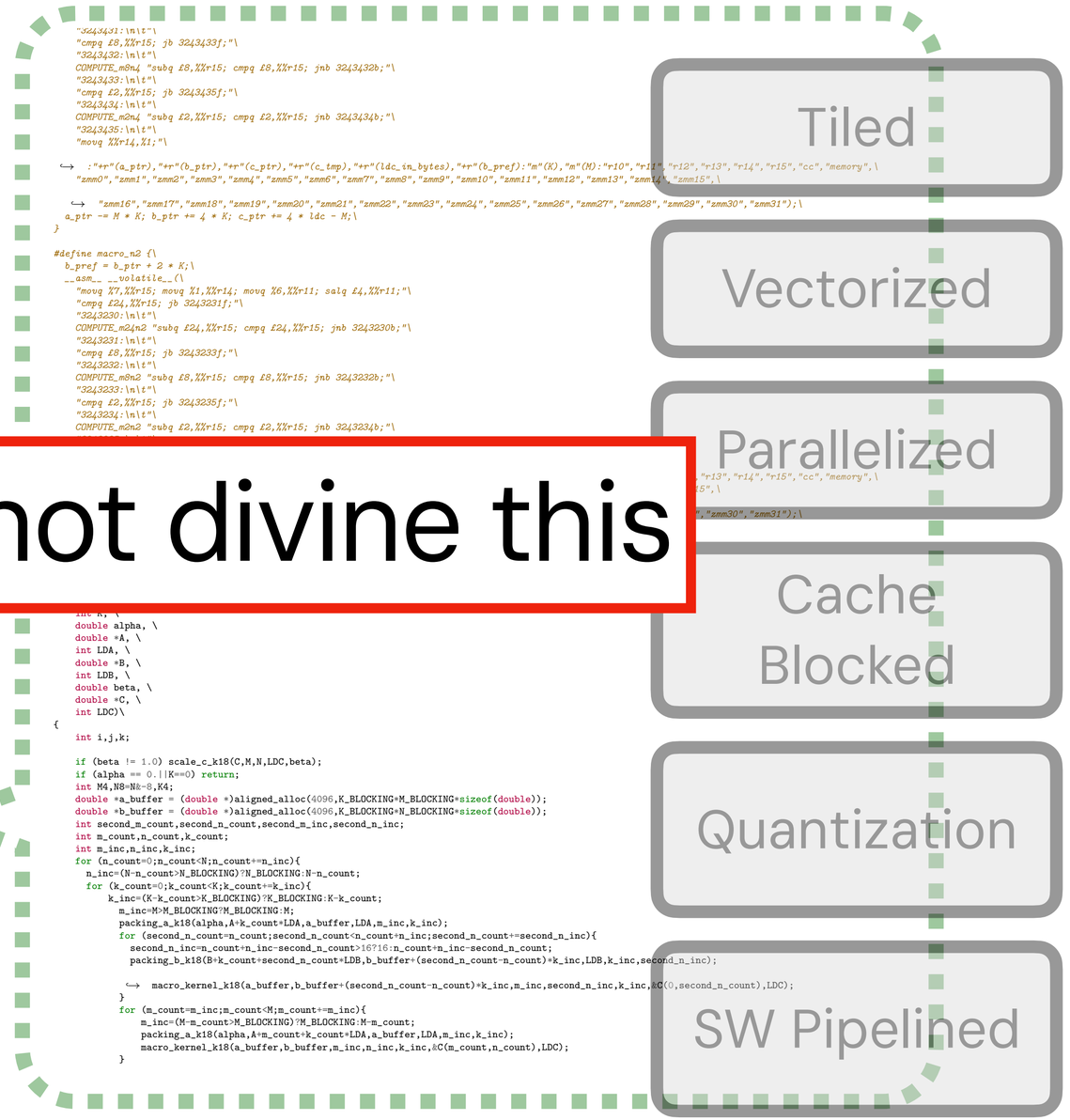


High Performance code is about **Hardware**

```
#include <cblas.h>
#include <vector>


#include "benchmark.h"
#include "matrix_lib.h"

int mai
    sto
    a.reserve(100);
    // Initialize other data
    // data-structures.
    cblas_sgemm(a.data()...);
    cblas_sgemm(....);


 return 0;
}
```

Matrix Multiply

Compilers cannot divine this



High Performance code is about **Hardware**

```
#include <cblas.h>
#include <vector>

#include "benchmark.h"
#include "matrix_lib.h"

int ma
    std::vector<float> a;
    a.reserve(100);
    // Initialize othe
    // data-structures
    cblas_sgemm(a.data()...);
    cblas_sgemm(....);

 return 0;
}
```

Compilers cannot divine this

…and hardware is **evolving**

```
#include <cblas.h>
#include <vector>
```

Matrix Multiply

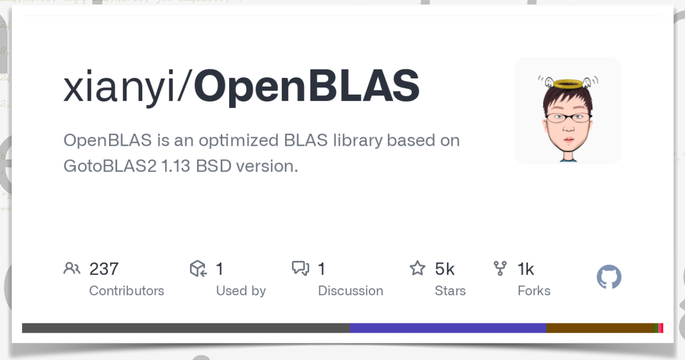
```
int main(..){
    std::vector<float> a;
    a.reserve(100);
    // Initialize other data
    // data-structures.
    cblas_sgemm(a.data()...);
    cblas_sgemm(....);

 return 0;
}
```

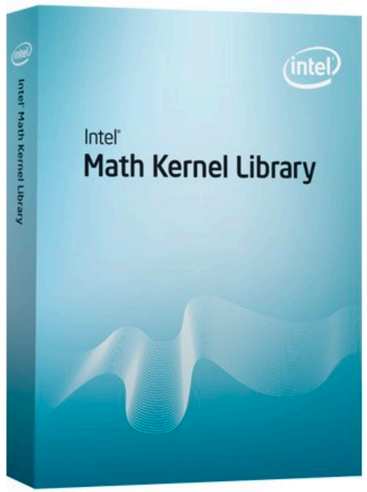
High performance implementations are often provided through **function interfaces.**

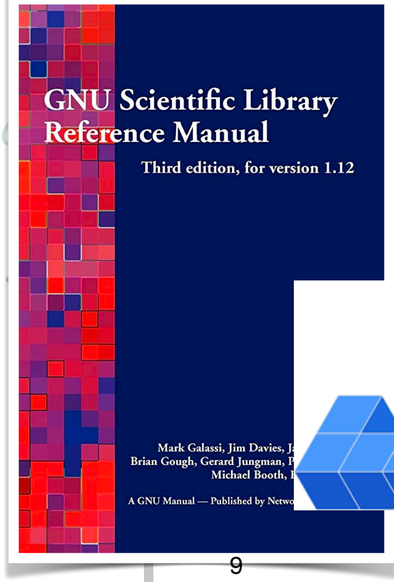
Key Observation #1: **Black boxing** high performance implementations through function interfaces is a powerful tool.

Intel® oneAPI Deep Neural Network Library

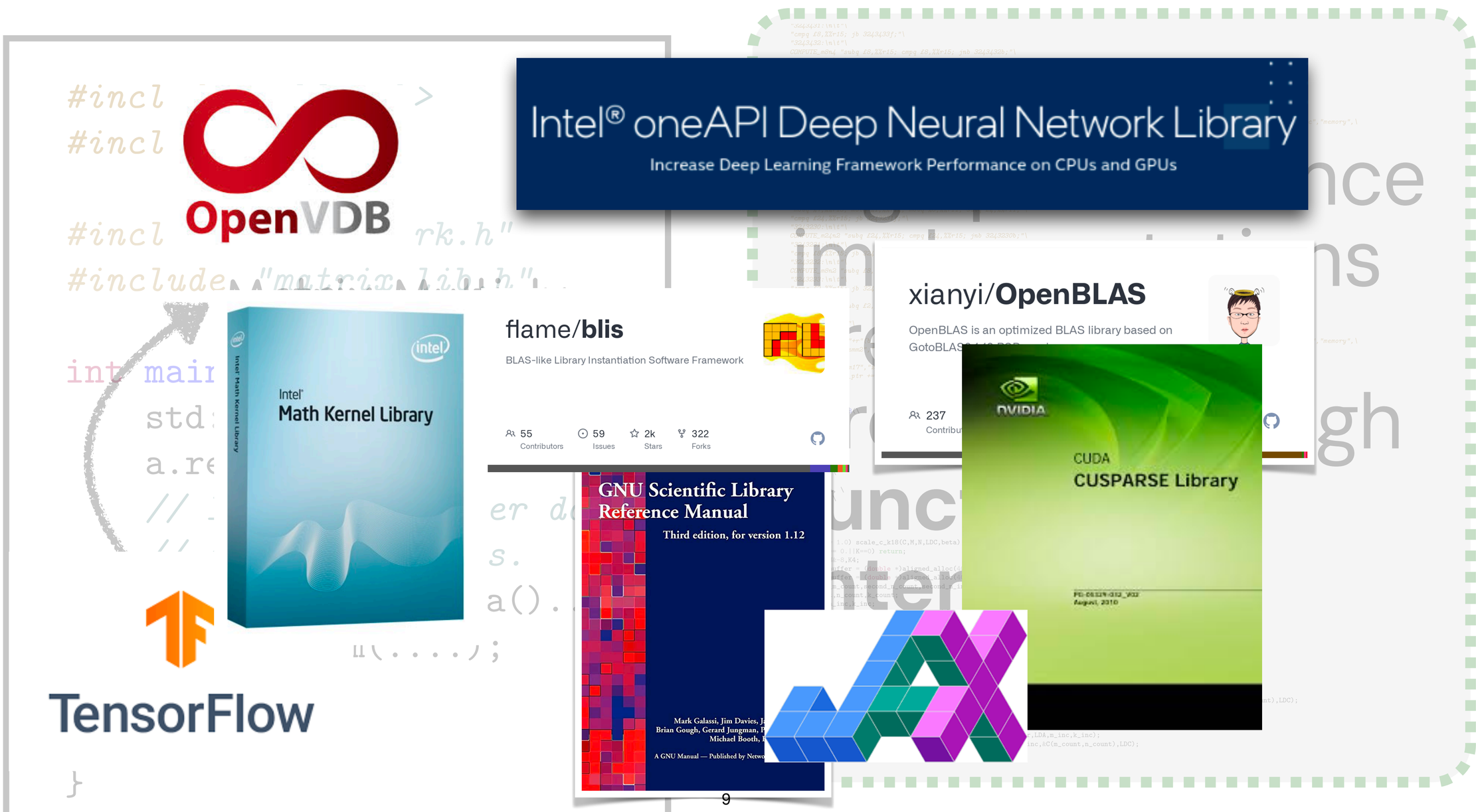
Increase Deep Learning Framework Performance on CPUs and GPUs







TensorFlow


OpenVDB
Intel® oneAPI Deep Neural Network Library
Increase Deep Learning Framework Performance on CPUs and GPUs
xianyi/OpenBLAS
OpenBLAS is an optimized BLAS library based on
237
Contribu
flame/blis
BLAS-like Library Instantiation Software Framework
55
Contributors
59
Issues
2k
Stars
322
Forks
Intel
Math Kernel Library
GNU Scientific Library
Reference Manual
Third edition, for version 1.12
Mark Galassi, Jim Davies, J
Brian Gough, Gerard Jungman, P
Michael Booth, M
A GNU Manual — Published by Netwo
NVIDIA
CUDA
CUSPARSE Library
August, 2010
TensorFlow

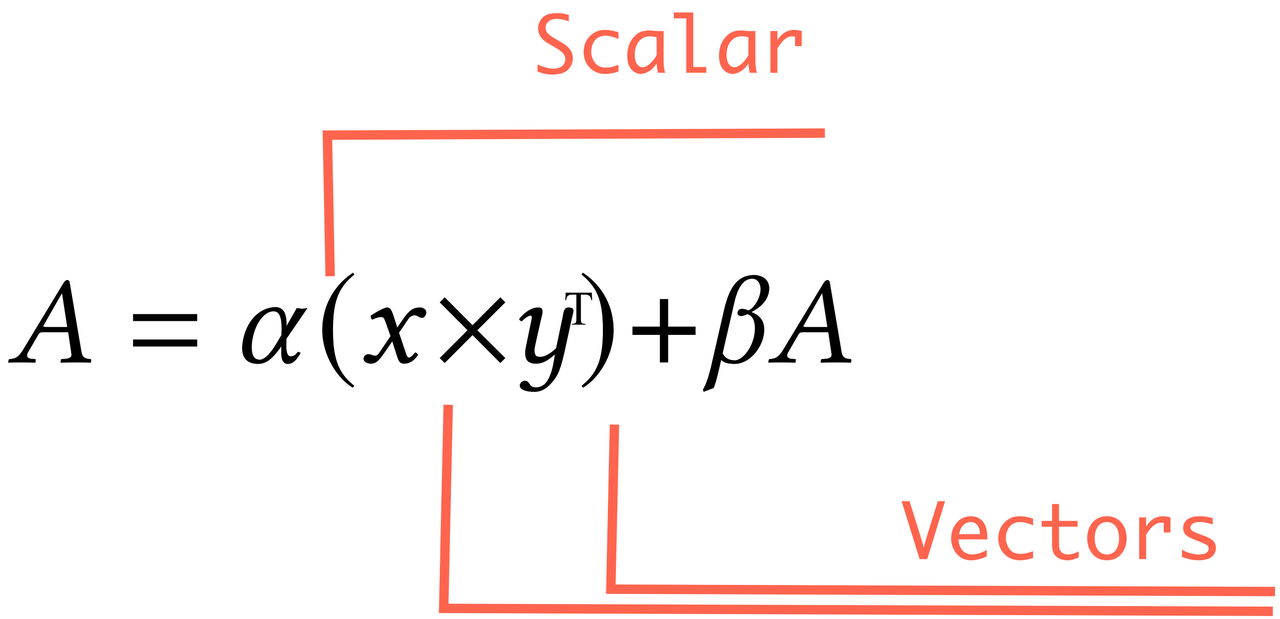
A **Simple** Computation

$$A = \alpha(x \times y^T) + \beta A$$

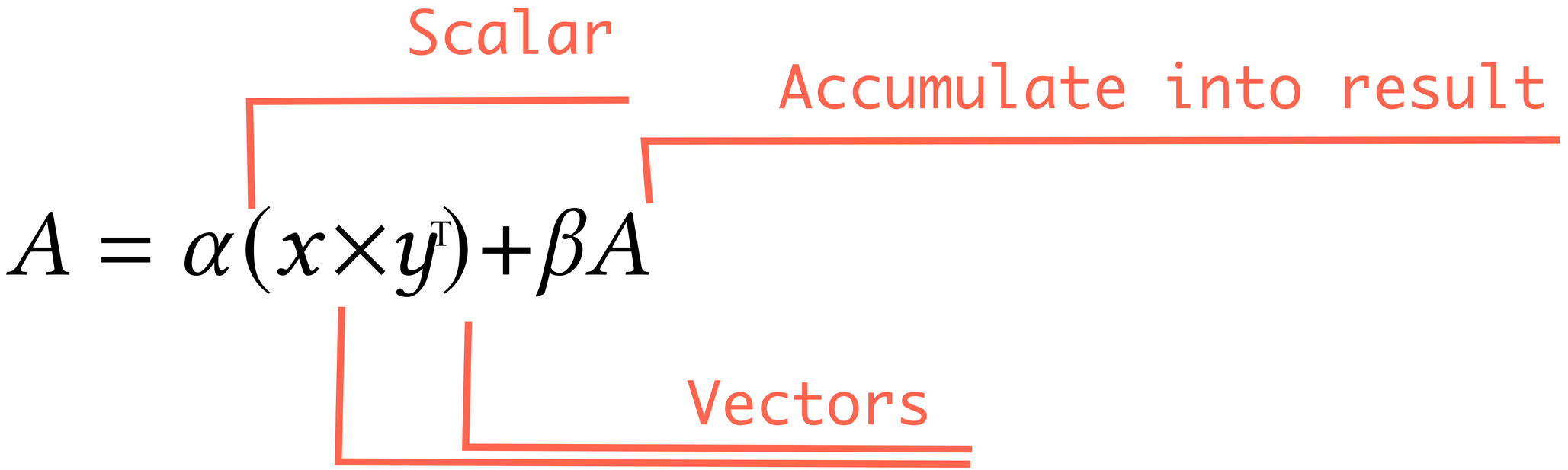
A **Simple** Computation

$$A = \alpha(x \times y^T) + \beta A$$

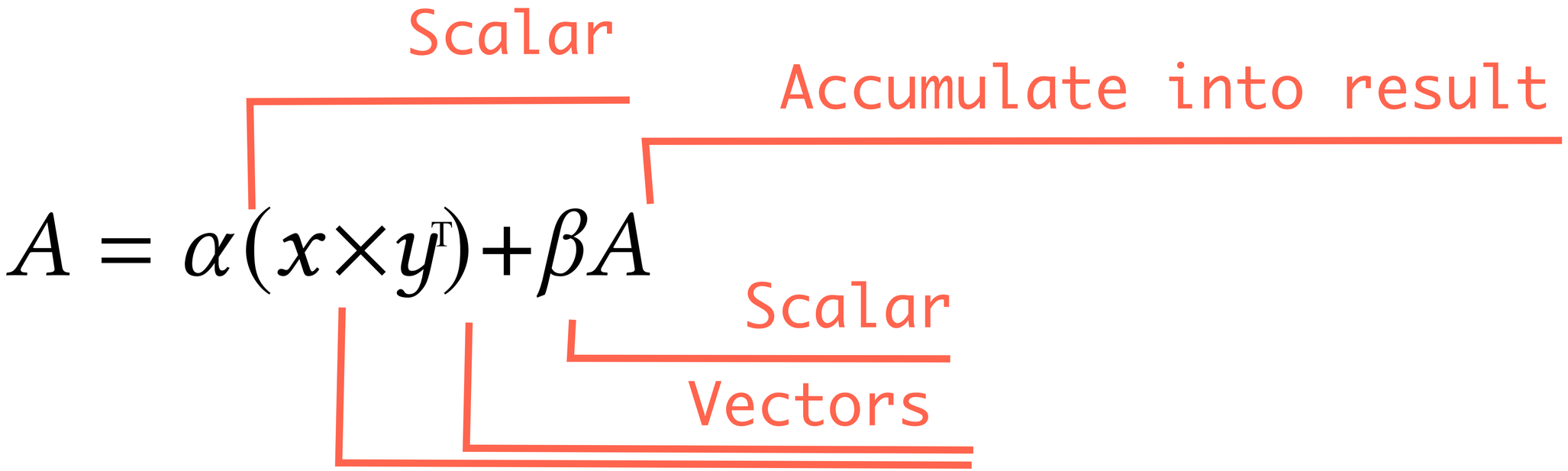
A **Simple** Computation



A **Simple** Computation



A **Simple** Computation



A **Simple** Computation

$$A = \alpha(x \times y^T) + \beta A$$

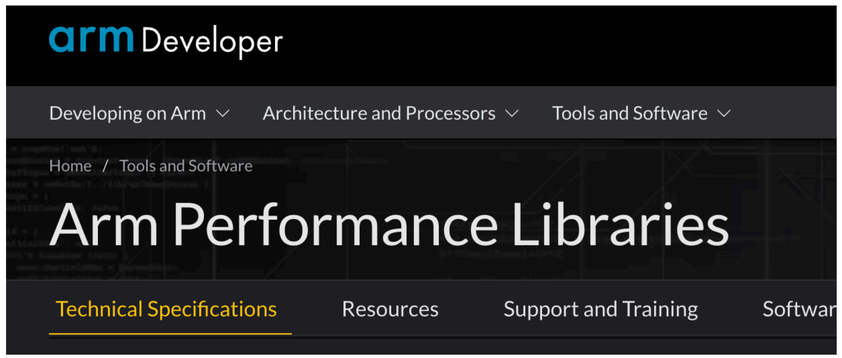
AArch64

A **Simple** Computation

$$A = \alpha(x \times y^{T}) + \beta A$$

```
#include <armpl.h>
```

AArch64



A **Simple** Computation

$$A = \alpha(x \times y^{T}) + \beta A$$

```
#include <Accelerate.h>
#include <armpl.h>
```

AArch64

Technology

Accelerate

Make large-scale mathemat
performance and low energ

Accelerate | simd | iOS 4.0+ |
watchOS 2.0+

arm Developer

Developing on Arm | Architecture and Processors | Tools and Software

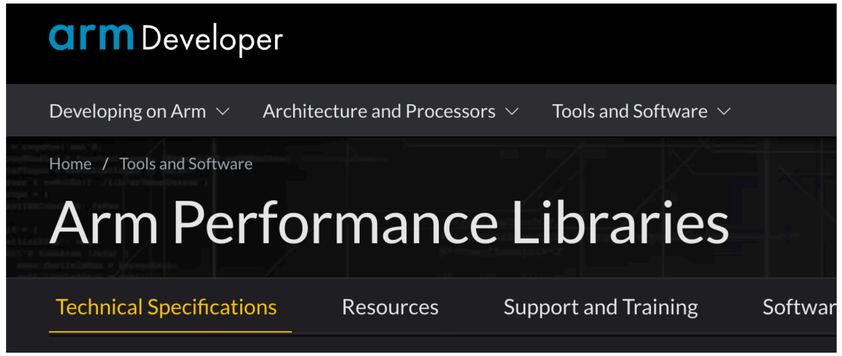
Home / Tools and Software

Arm Performance Libraries

Technical Specifications | Resources | Support and Training | Softwar

A **Simple** Computation

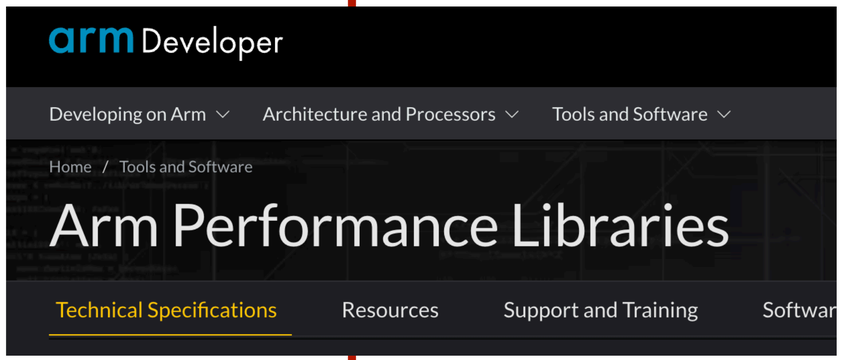
$$A = \alpha(x \times y^{T}) + \beta A$$



A **Simple** Computation

$$A = \alpha(x \times y^{T}) + \beta A$$

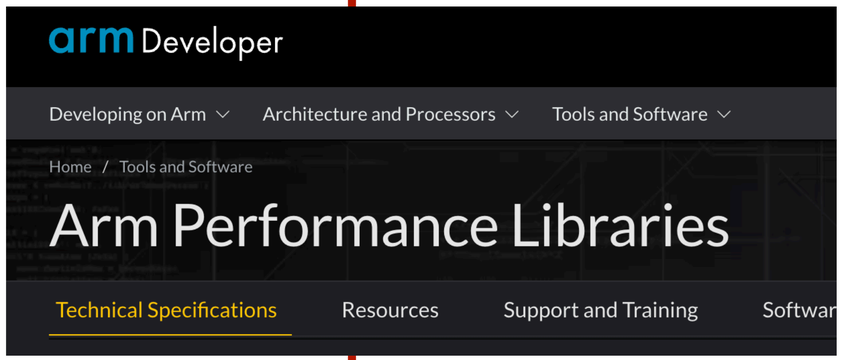
```
#include <armpl.h>

void armpl_sgerb(....){



}
```



A **Simple** Computation

$$A = \alpha(x \times y^T) + \beta A$$

```
#include <armpl.h>

void armpl_sgerb(....){
        sgerb_(&m, &n, &alpha,
                x.get_data(), &inc_x_y,
                y.get_data(), &inc_x_y, &beta,
                A.get_data(), &m);
}
```

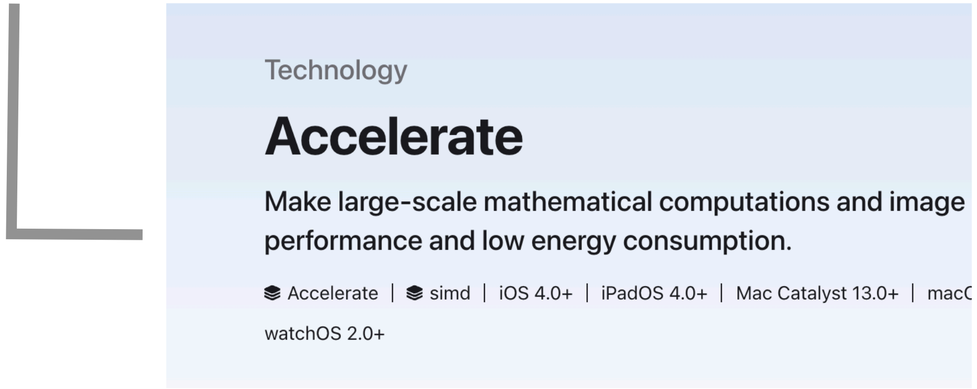


A **Simple** Computation

$$A = \alpha(x \times y^T) + \beta A$$

A **Simple** Computation

$$A = \alpha(x \times y^{T}) + \beta A$$



Technology

Accelerate

Make large-scale mathematical computations and image performance and low energy consumption.

Accelerate | simd | iOS 4.0+ | iPadOS 4.0+ | Mac Catalyst 13.0+ | macO

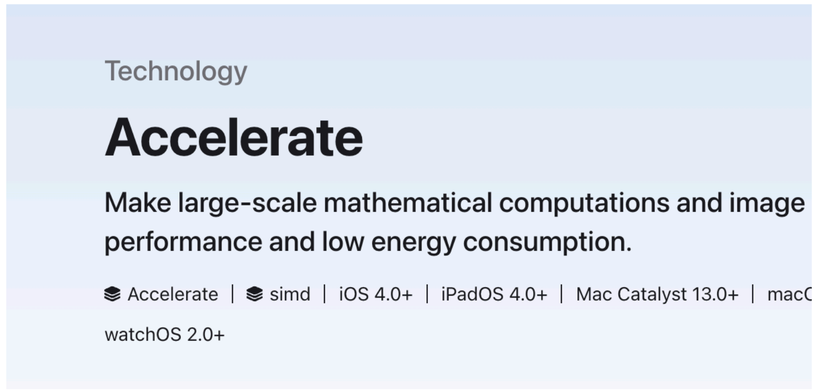
watchOS 2.0+

```c
#include <Accelerate.h>

void accelerate_sgerb(....){




}
```

A **Simple** Computation

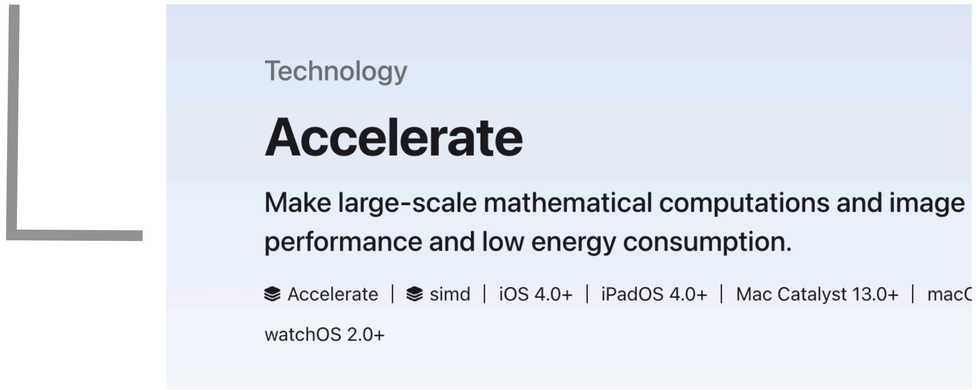
$$A = \alpha(x \times y^{T}) + \beta A$$

```
#include <Accelerate.h>

void accelerate_sgerb(....){

    cblas_sger(CblasColMajor, x.m, y.n, alpha,
               x.get_data(), 1,
               y.get_data(), 1,
               A.get_data(), R.m);

}
```



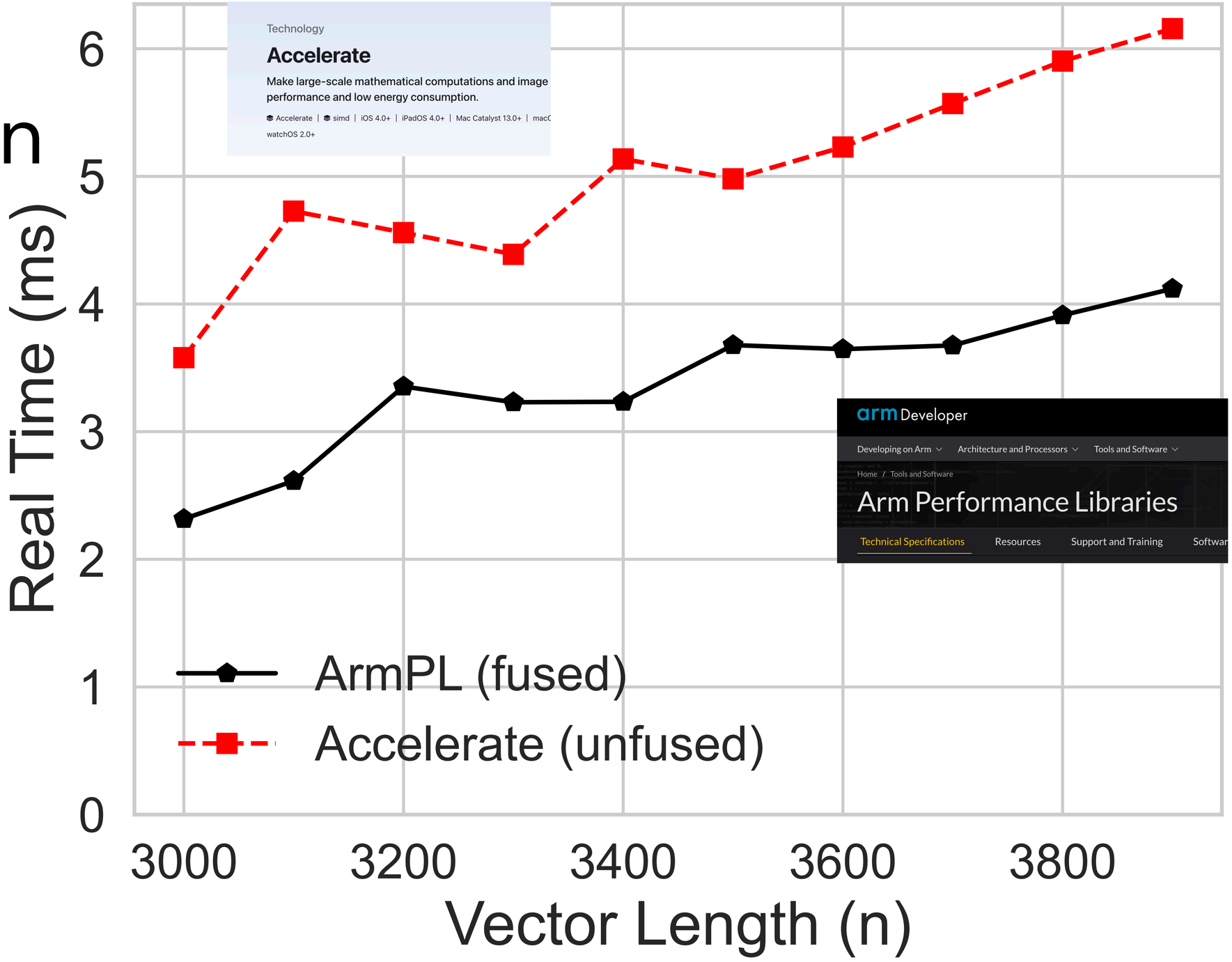
A **Simple** Computation

$$A = \alpha(x \times y^{T}) + \beta A$$

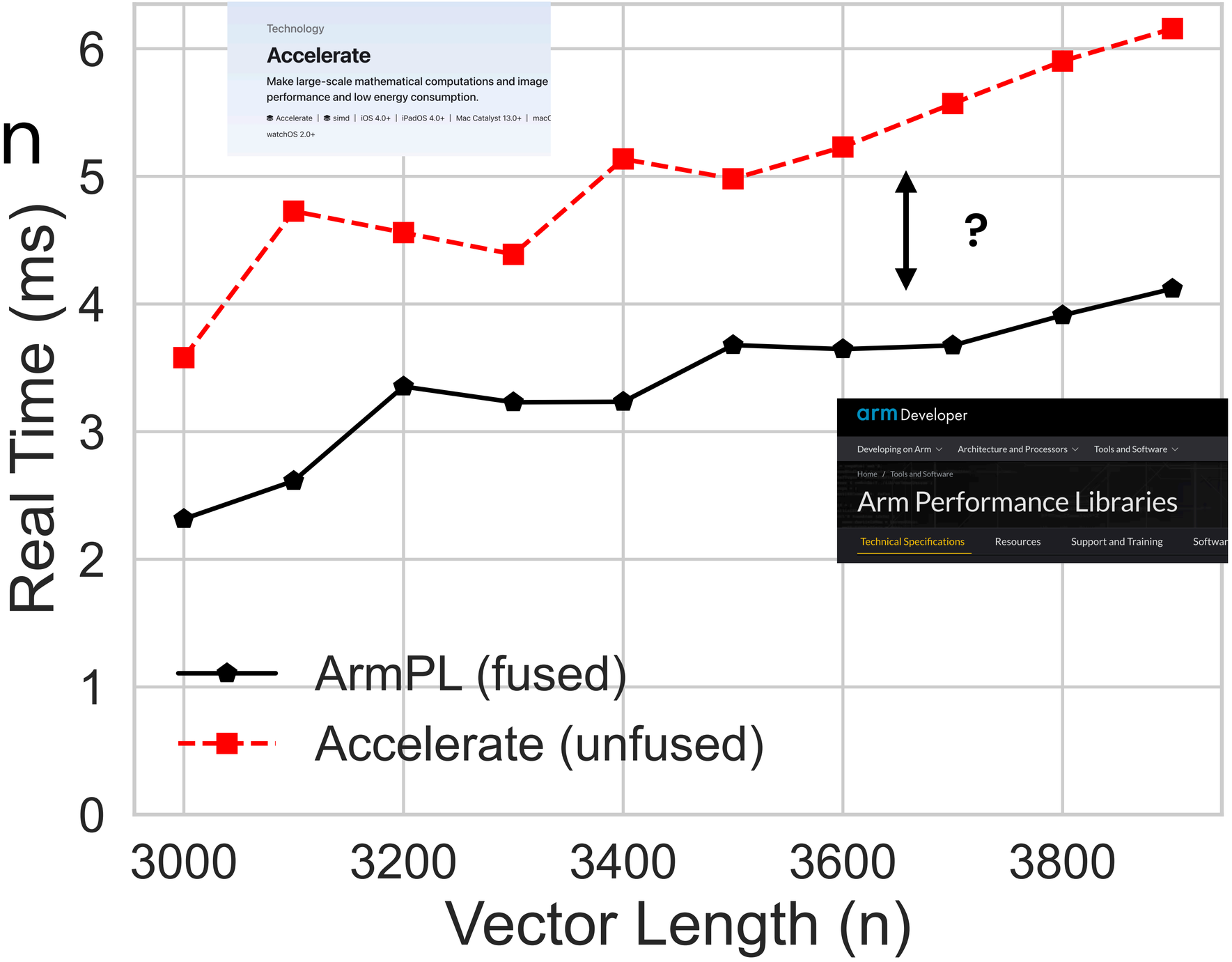



```c
#include <Accelerate.h>

void accelerate_sgerb(....){

    cblas_sger(CblasColMajor, x.m, y.n, alpha,
               x.get_data(), 1,
               y.get_data(), 1,
               A.get_data(), R.m);
    cblas_saxpy(m * n, beta,
               A.get_data(), inc_x_y,
               A.get_data(), inc_x_y);
}
```

A **Simple** Computation



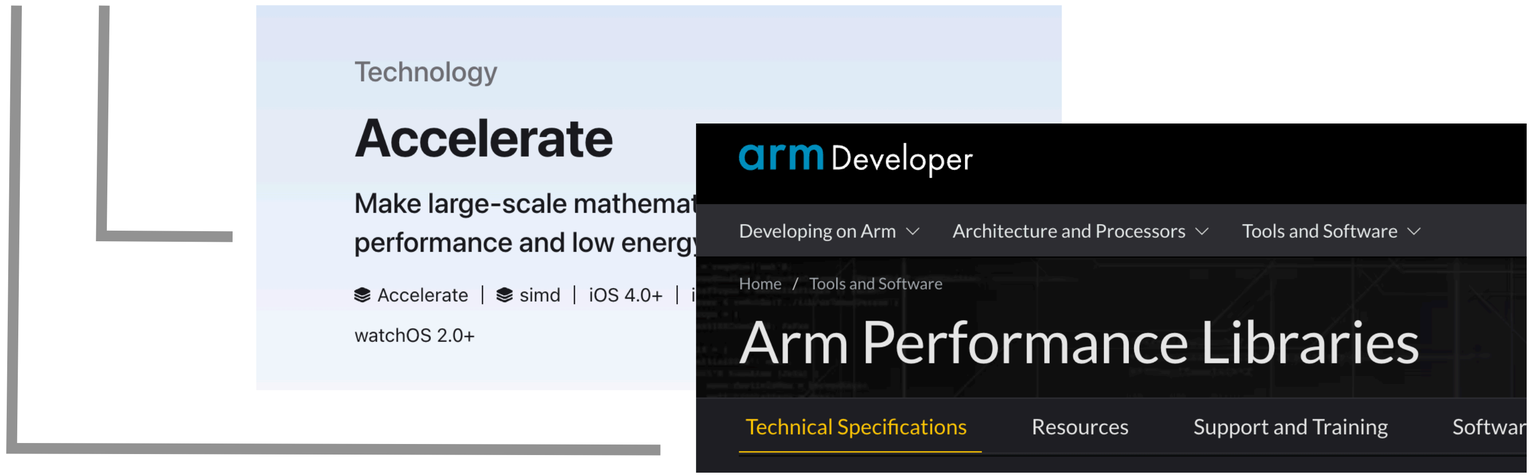
A **Simple** Computation



A **Simpler** Computation

$$A = \alpha(x \times y^{T})$$

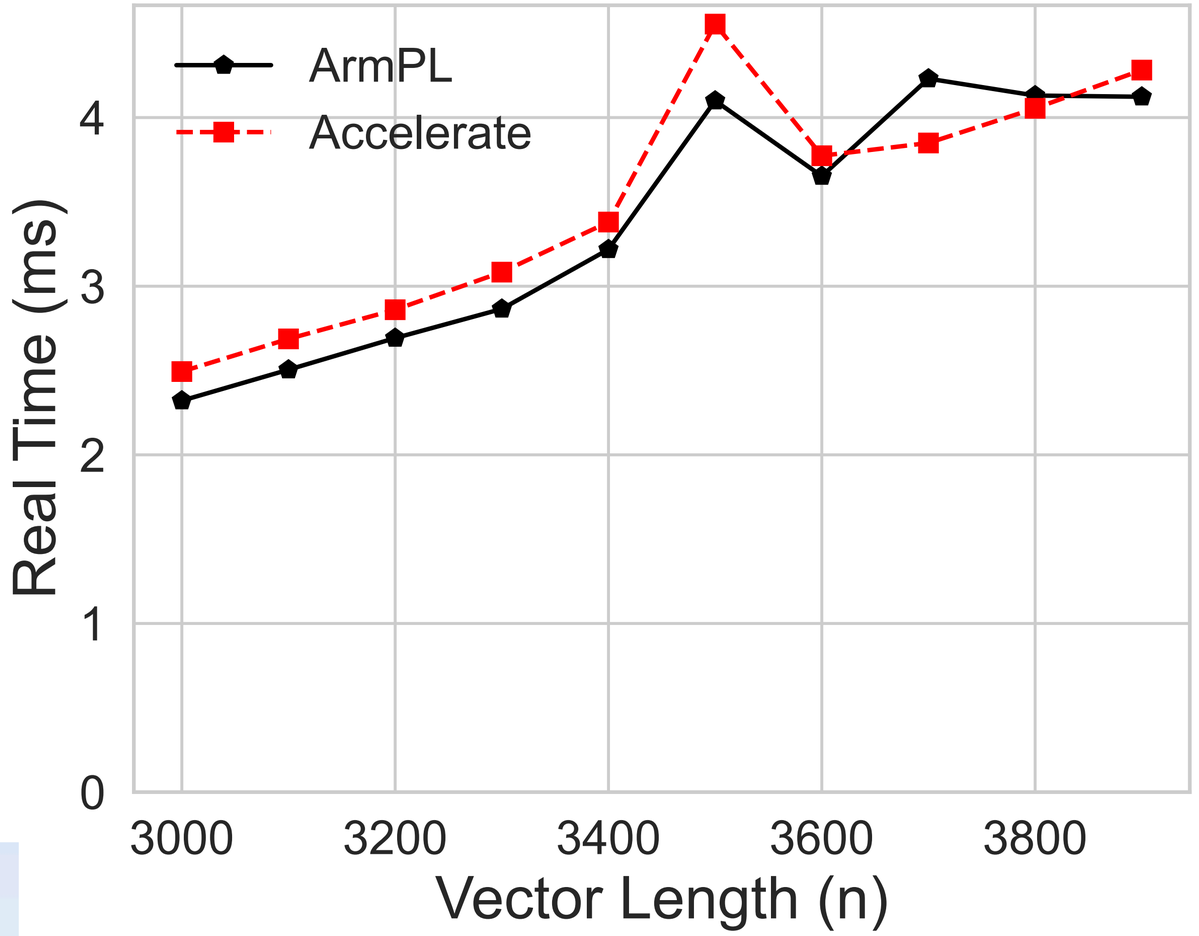
```
#include <Accelerate.h>
#include <armpl.h>
```



A **Simpler** Computation

$$A = \alpha(x \times y^{T})$$

```
#include <Accelerate.h>
#include <armpl.h>
```



Technology

Accelerate

Make large-scale mathemat
performance and low energ

Accelerate | simd | iOS 4.0+ |
watchOS 2.0+

arm Developer

Developing on Arm | Architecture and Processors | Tools and Software

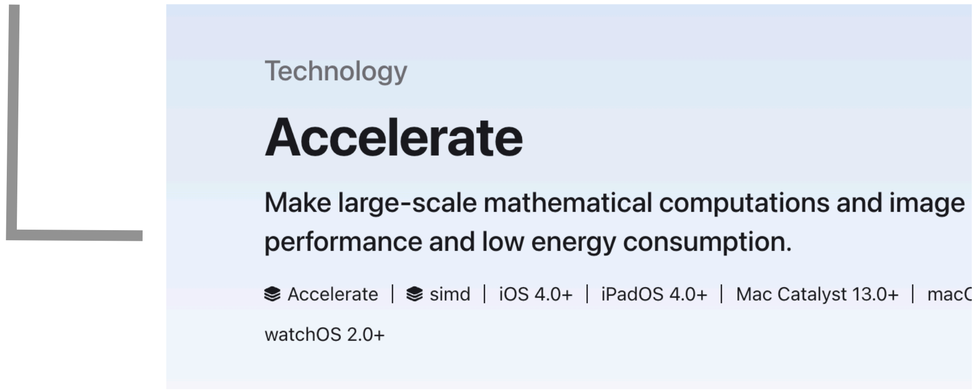
Home / Tools and Software

Arm Performance Libraries

Technical Specifications | Resources | Support and Training | Softwar

A **Simple** Computation

$$A = \alpha(x \times y^{T}) + \beta A$$



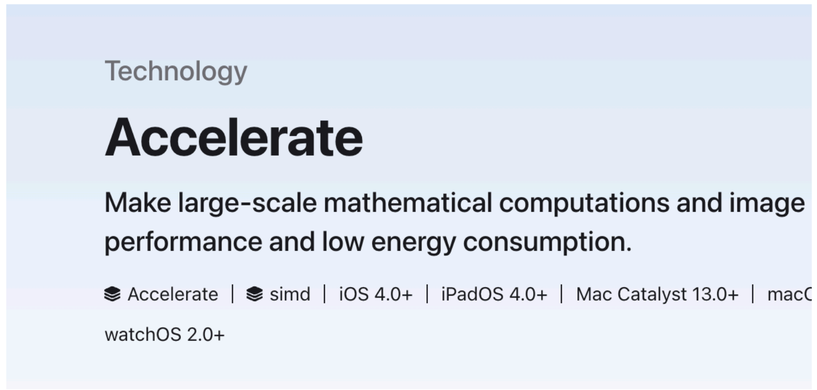

```c
#include <Accelerate.h>

void accelerate_sgerb(....){

    cblas_sger(CblasColMajor, x.m, y.n, alpha,
               x.get_data(), 1,
               y.get_data(), 1,
               A.get_data(), R.m);
    cblas_saxpy(m * n, beta,
               A.get_data(), inc_x_y,
               A.get_data(), inc_x_y);
}
```

A **Simple** Computation

First element of intermediate is **out of cache**

$$A = \alpha(x \times y^{T}) + \beta A$$

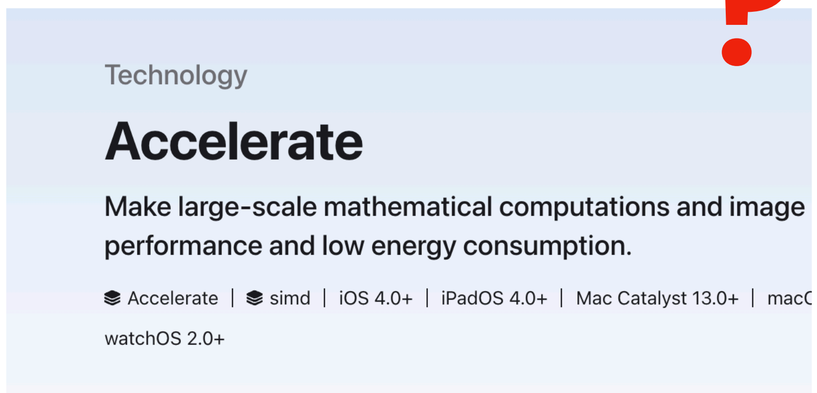
```
#include <Accelerate.h>

void accelerate_sgerb(....){

    cblas_sger(CblasColMajor, x.m, y.n, alpha,
               x.get_data(), 1,
               y.get_data(), 1,
               A.get_data(), R.m);
    cblas_saxpy(m * n, beta,
               A.get_data(), inc_x_y,
               A.get_data(), inc_x_y);
}
```



A **Simple** Computation

$$A = \alpha(x \times y) + \beta A$$

?



A **Simple** Computation

$$A = \alpha(x \times y) + \beta A$$

?

Technology

Accelerate

Make large-scale mathematical computations and image performance and low energy consumption.

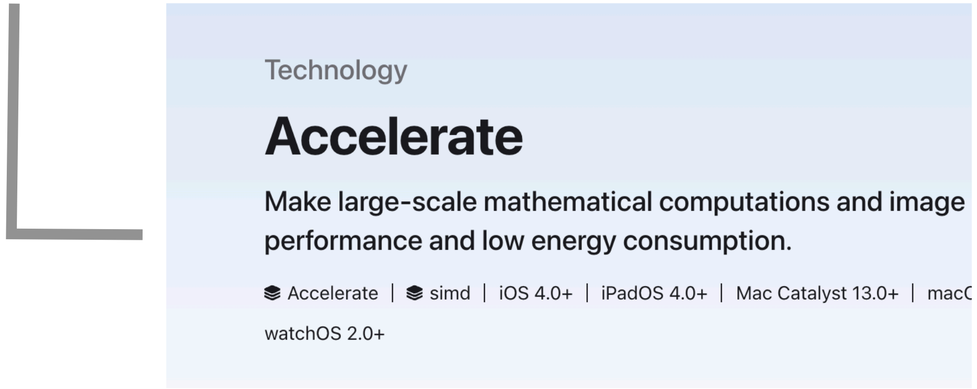
Accelerate | simd | iOS 4.0+ | iPadOS 4.0+ | Mac Catalyst 13.0+ | macO watchOS 2.0+

Proprietary....

Super Secret Super Fast Accelerator ™

A **Simple** Computation

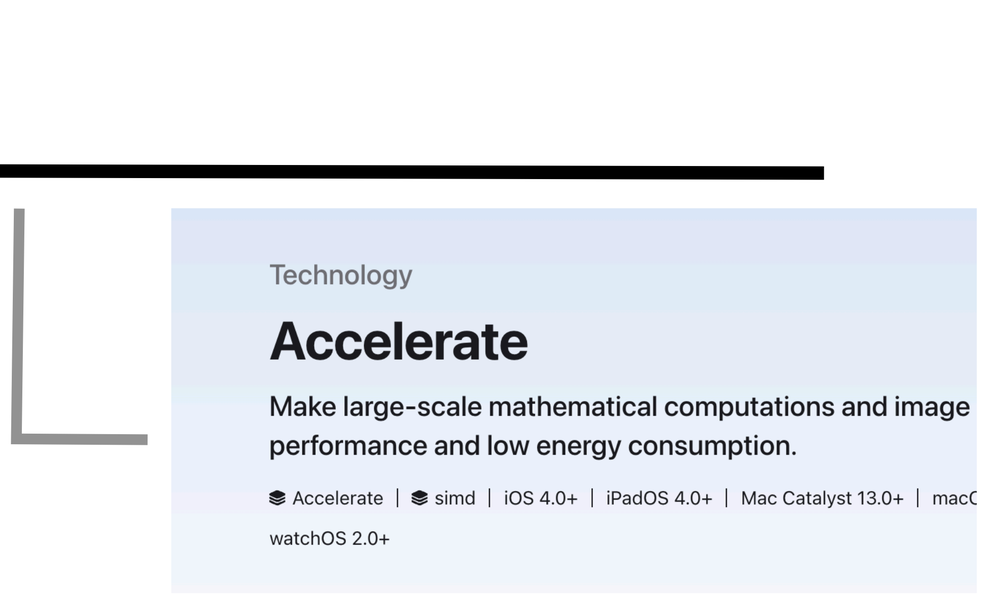
$$A = \alpha(x \times y) + \beta A$$




```
#include <Accelerate.h>

void accelerate_sgerb(....){



    cblas_sger(CblasColMajor, x.m, y.n, alpha,
                x.get_data(), 1,
                y.get_data(), 1,
                A.get_data(), R.m);
    cblas_saxpy(m * n, beta,
                A.get_data(), inc_x_y,
                A.get_data(), inc_x_y);
}
```

A **Simple** Computation

$$A = \alpha(x \times y) + \beta A$$

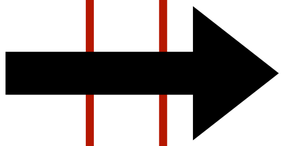



```
#include <Accelerate.h>

void accelerate_sgerb(....){



    cblas_sger(CblasColMajor, x.m, y.n, alpha,
               x.get_data(), 1,
               y.get_data(), 1,
               A.get_data(), R.m);
    cblas_saxpy(m * n, beta,
               A.get_data(), inc_x_y,
               A.get_data(), inc_x_y);
}
```

Process my data in chunks...?

A **Simple** Computation

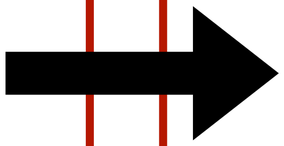
```
#include <Accelerate.h>

void accelerate_sgerb(....){



    cblas_sger(CblasColMajor, x.m, y.n, alpha,
                x.get_data(), 1,
                y.get_data(), 1,
                A.get_data(), R.m);
    cblas_saxpy(m * n, beta,
                A.get_data(), inc_x_y,
                A.get_data(), inc_x_y);
}
```



```
#include <Accelerate.h>

void accelerate_sgerb(....){

  for (int j = 0; j < R.n; j += ColTile) {
    cblas_sger(CblasColMajor, m, 1, alpha,
                x.get_data(), 1, &y(0, j),
                1, &R(0, j), R.m);
    cblas_saxpy(m * ColTile, 1.0f,
                &A(0, j), inc_x_y,
                &R(0, j), inc_x_y);
  }

}
```

A **Simple** Computation

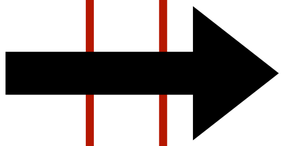
```
#include <Accelerate.h>

void accelerate_sgerb(....){



    cblas_sger(CblasColMajor, x.m, y.n, alpha,
                x.get_data(), 1,
                y.get_data(), 1,
                A.get_data(), R.m);
    cblas_saxpy(m * n, beta,
                A.get_data(), inc_x_y,
                A.get_data(), inc_x_y);
}
```



```
#include <Accelerate.h>

void accelerate_sgerb(....){

  for (int j = 0; j < R.n; j += ColTile) {
    cblas_sger(CblasColMajor, m, 1, alpha,
                x.get_data(), 1, &y(0, j),
                1, &R(0, j), R.m);
    cblas_saxpy(m * ColTile, 1.0f,
                &A(0, j), inc_x_y,
                &R(0, j), inc_x_y);
  }

}
```

A **Simple** Computation

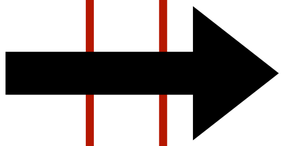
```
#include <Accelerate.h>

void accelerate_sgerb(....){



    cblas_sger(CblasColMajor, x.m, y.n, alpha,
                x.get_data(), 1,
                y.get_data(), 1,
                A.get_data(), R.m);
    cblas_saxpy(m * n, beta,
                A.get_data(), inc_x_y,
                A.get_data(), inc_x_y);
}
```



```
#include <Accelerate.h>

void accelerate_sgerb(....){

  for (int j = 0; j < R.n; j += ColTile) {
    cblas_sger(CblasColMajor, m, 1, alpha,
                x.get_data(), 1, &y(0, j),
                1, &R(0, j), R.m);
    cblas_saxpy(m * ColTile, 1.0f,
                &A(0, j), inc_x_y,
                &R(0, j), inc_x_y);
  }

}
```

A **Simple** Computation

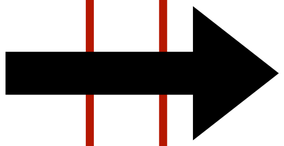
```
#include <Accelerate.h>

void accelerate_sgerb(....){



    cblas_sger(CblasColMajor, x.m, y.n, alpha,
                x.get_data(), 1,
                y.get_data(), 1,
                A.get_data(), R.m);
    cblas_saxpy(m * n, beta,
                A.get_data(), inc_x_y,
                A.get_data(), inc_x_y);
}
```



```
#include <Accelerate.h>

void accelerate_sgerb(....){

  for (int j = 0; j < R.n; j += ColTile) {
    cblas_sger(CblasColMajor, m, 1, alpha,
                x.get_data(), 1, &y(0, j),
                1, &R(0, j), R.m);
    cblas_saxpy(m * ColTile, 1.0f,
                &A(0, j), inc_x_y,
                &R(0, j), inc_x_y);
  }

}
```

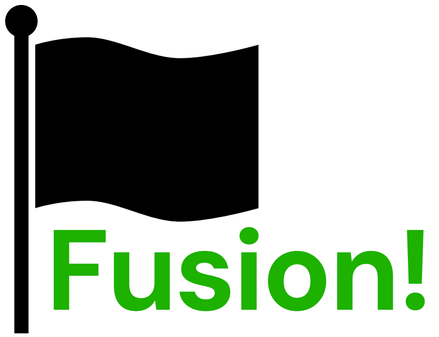
A **Simple** Computation

```
#include <Accelerate.h>

void accelerate_sgerb(....){



    cblas_sger(CblasColMajor, x.m, y.n, alpha,
                x.get_data(), 1,
                y.get_data(), 1,
                A.get_data(), R.m);
    cblas_saxpy(m * n, beta,
                A.get_data(), inc_x_y,
                A.get_data(), inc_x_y);
}
```

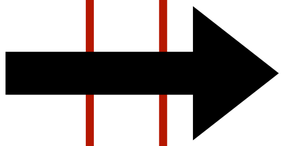


```
#include <Accelerate.h>

void accelerate_sgerb(....){

  for (int j = 0; j < R.n; j += ColTile) {
    cblas_sger(CblasColMajor, m, 1, alpha,
                x.get_data(), 1, &y(0, j),
                1, &R(0, j), R.m);
    cblas_saxpy(m * ColTile, 1.0f,
                &A(0, j), inc_x_y,
                &R(0, j), inc_x_y);
  }

}
```

A **Simple** Computation

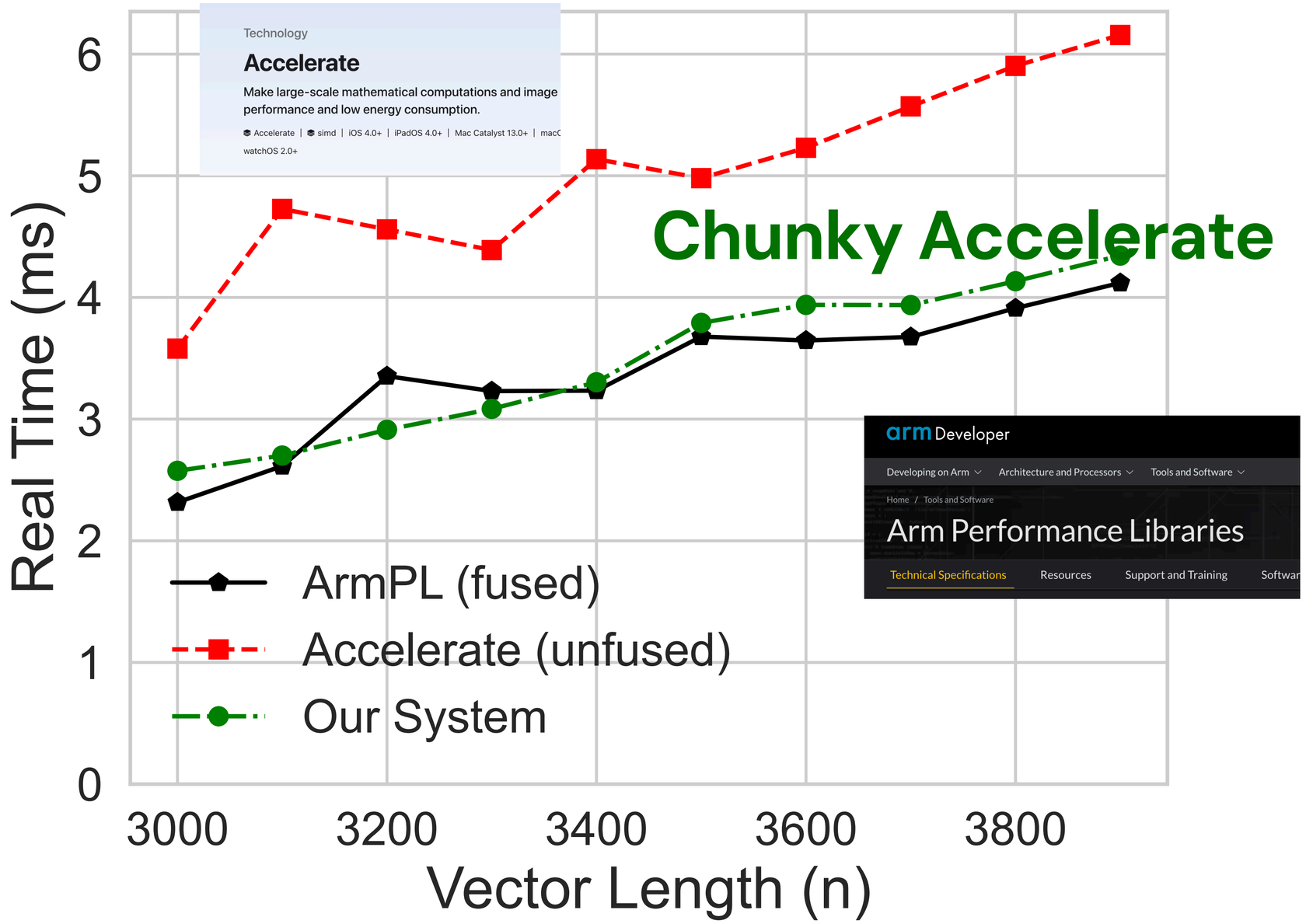



```
#include <Accelerate.h>

void accelerate_sgerb(....){



    cblas_sger(CblasColMajor, x.m, y.n, alpha,
                x.get_data(), 1,
                y.get_data(), 1,
                A.get_data(), R.m);
    cblas_saxpy(m * n, beta,
                A.get_data(), inc_x_y,
                A.get_data(), inc_x_y);
}
```



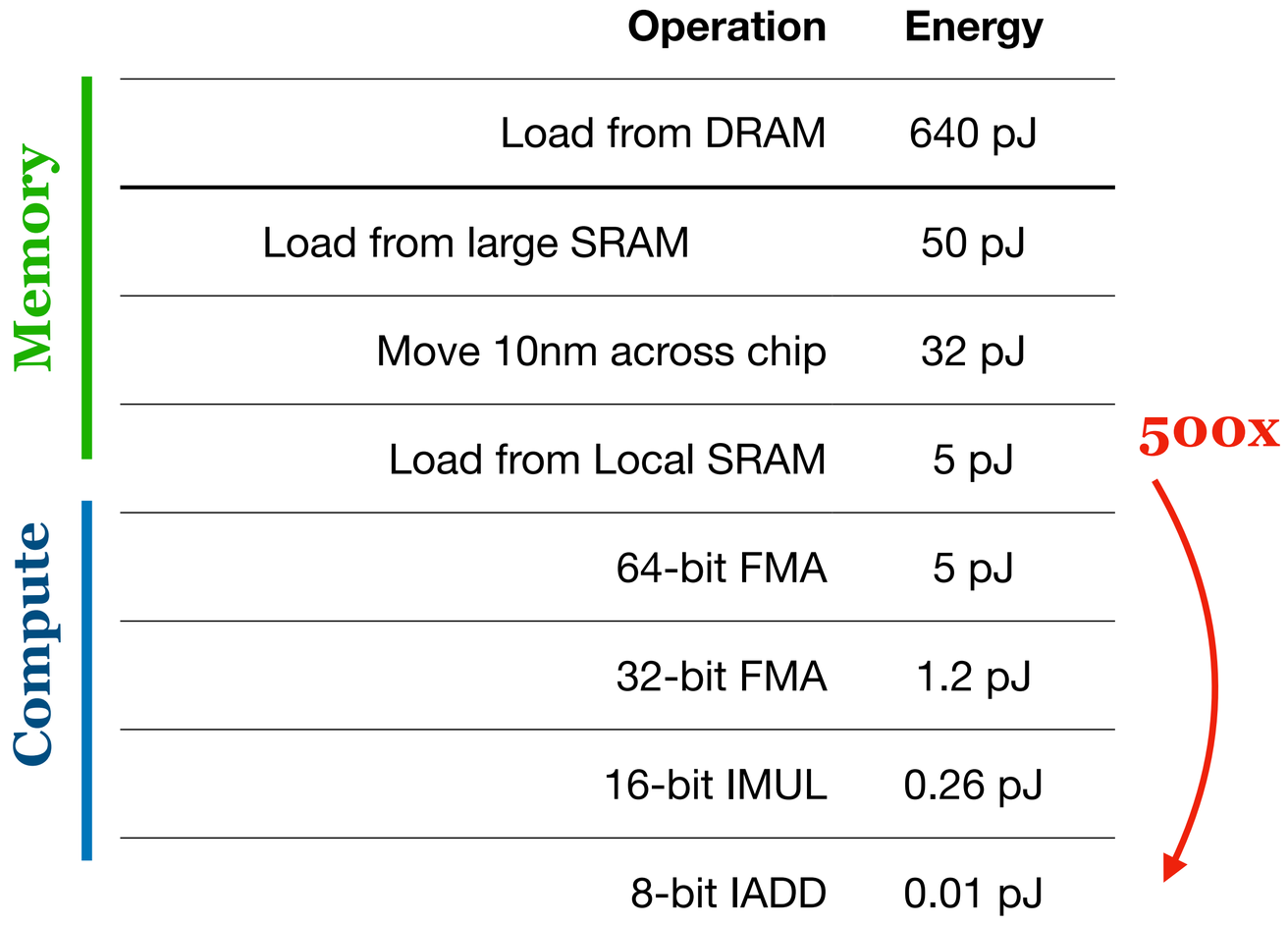
```
#include <Accelerate.h>

void accelerate_sgerb(....){

  for (int j = 0; j < R.n; j += ColTile) {
    cblas_sger(CblasColMajor, m, 1, alpha,
                x.get_data(), 1, &y(0, j),
                1, &R(0, j), R.m);
    cblas_saxpy(m * ColTile, 1.0f,
                &A(0, j), inc_x_y,
                &R(0, j), inc_x_y);
  }

}
```

A **Simple** Computation



Key Observation #2: Naive function composition results in **locality** losses.

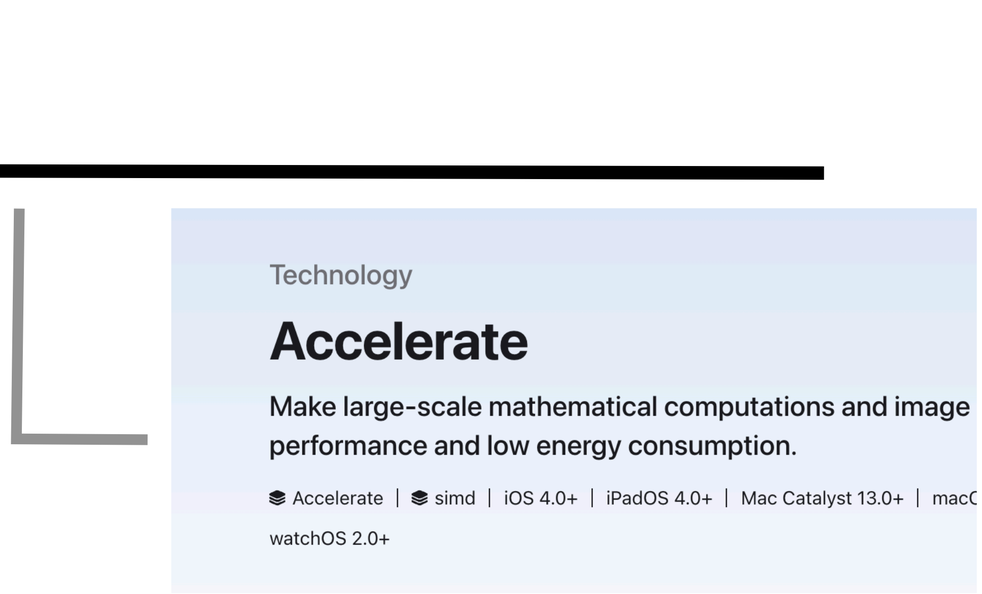
Data movement dominates **energy**



(Bill Dally 14nm foundry process)

A **Simple** Computation

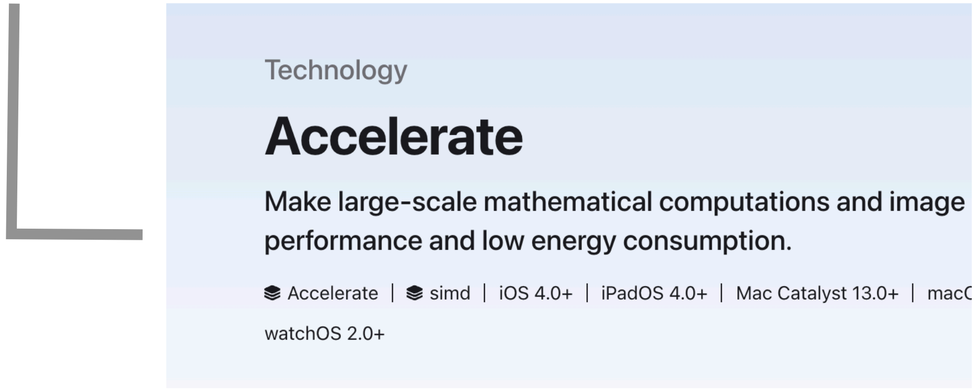
$$A = \alpha(x \times y) + \beta A$$




```
#include <Accelerate.h>

void accelerate_sgerb(....){

  for (int j = 0; j < R.n; j += ColTile) {
    cblas_sger(CblasColMajor, m, 1, alpha,
                x.get_data(), 1, &y(0, j),
                1, &R(0, j), R.m);
    cblas_saxpy(m * ColTile, 1.0f,
                &A(0, j), inc_x_y,
                &R(0, j), inc_x_y);
  }

}
```

A **Simple** Computation...for a **compiler**?

$$A = \alpha(x \times y) + \beta A$$



Technology

Accelerate

Make large-scale mathematical computations and image performance and low energy consumption.

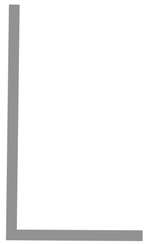
Accelerate | simd | iOS 4.0+ | iPadOS 4.0+ | Mac Catalyst 13.0+ | macC

watchOS 2.0+

```
#include <Accelerate.h>

void accelerate_sgerb(....){

  for (int j = 0; j < R.n; j += ColTile) {
    cblas_sger(CblasColMajor, m, 1, alpha,
                x.get_data(), 1, &y(0, j),
                1, &R(0, j), R.m);
    cblas_saxpy(m * ColTile, 1.0f,
                &A(0, j), inc_x_y,
                &R(0, j), inc_x_y);
  }

}
```

A **Simple** Computation…for a **compiler**?

$$A = \alpha(x \times y) + \beta A$$



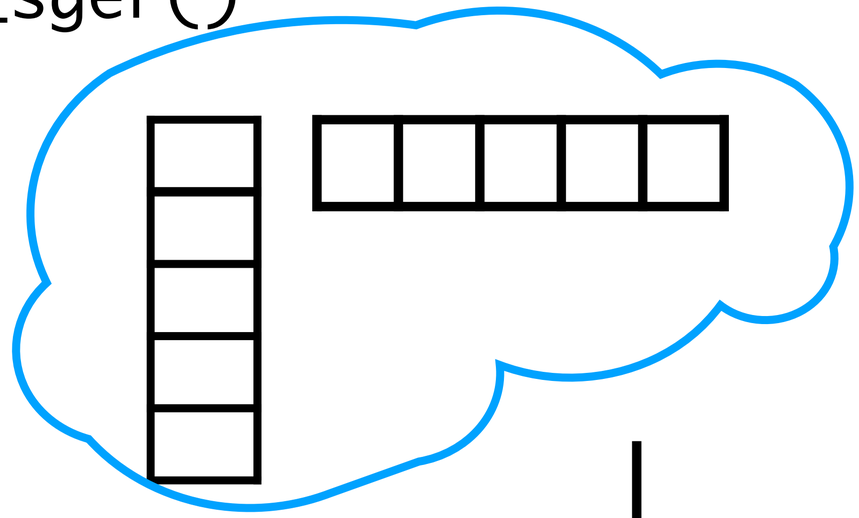
Technology

Accelerate

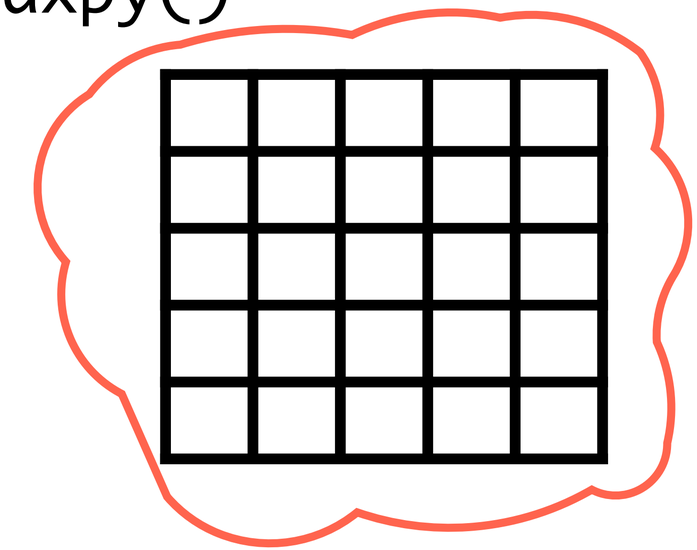
Make large-scale mathematical computations and image
performance and low energy consumption.

Accelerate | simd | iOS 4.0+ | iPadOS 4.0+ | Mac Catalyst 13.0+ | macO
watchOS 2.0+

`cblas_sger()`

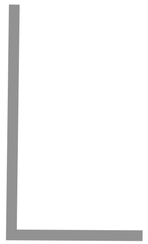


`cblas_saxpy()`



A **Simple** Computation…for a **compiler**?

$$A = \alpha(x \times y) + \beta A$$



Technology

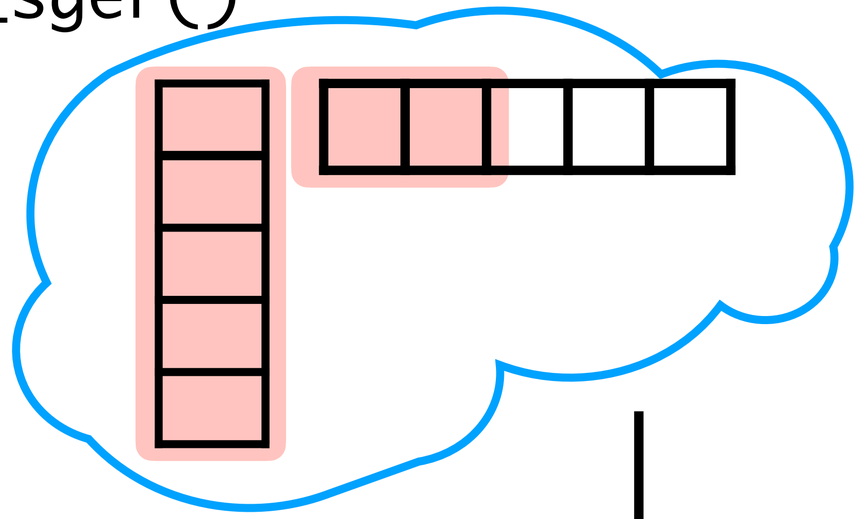
Accelerate

Make large-scale mathematical computations and image
performance and low energy consumption.

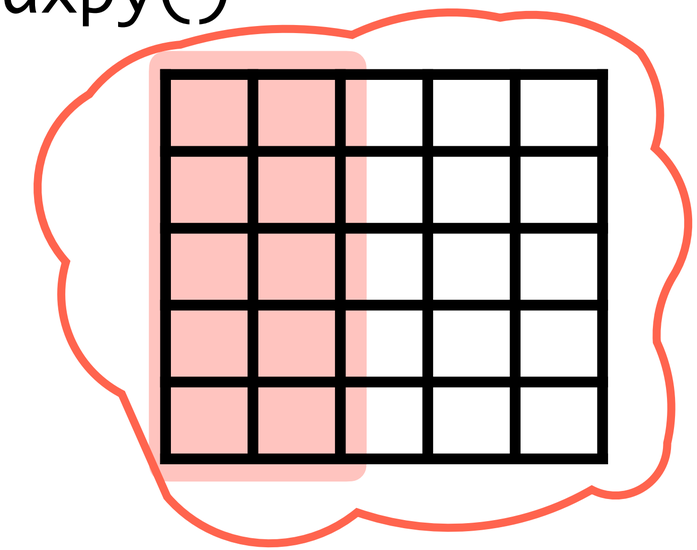
Accelerate | simd | iOS 4.0+ | iPadOS 4.0+ | Mac Catalyst 13.0+ | macO

watchOS 2.0+

`cblas_sger()`

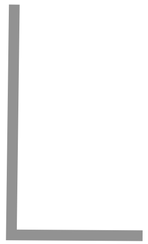


`cblas_saxpy()`



A **Simple** Computation…for a **compiler**?

$$A = \alpha(x \times y) + \beta A$$



Technology

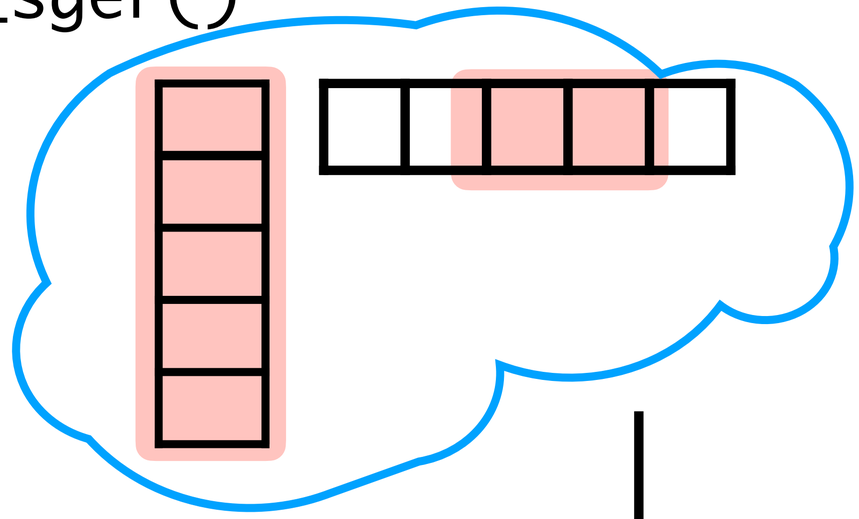
Accelerate

Make large-scale mathematical computations and image
performance and low energy consumption.

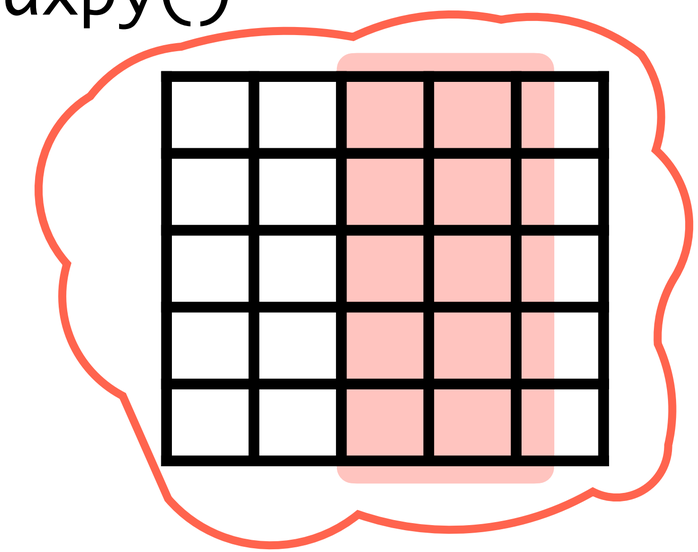
Accelerate | simd | iOS 4.0+ | iPadOS 4.0+ | Mac Catalyst 13.0+ | macO

watchOS 2.0+

`cblas_sger()`

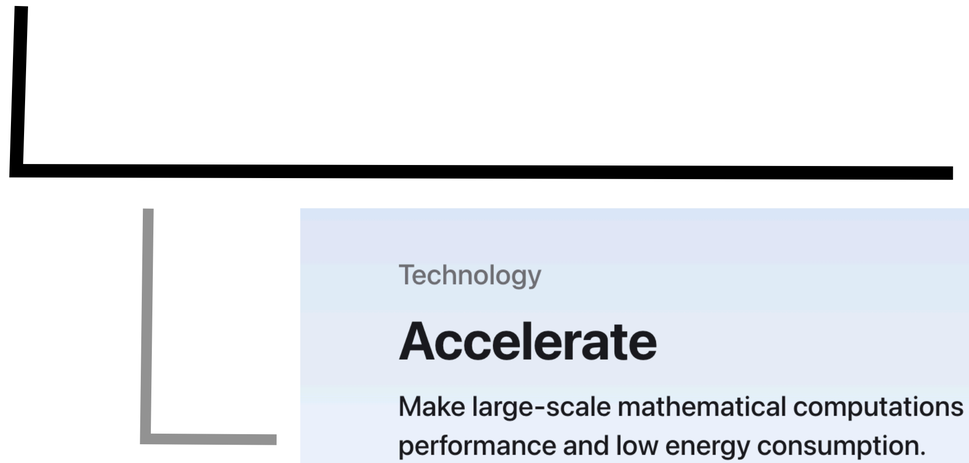


`cblas_saxpy()`



A **Simple** Computation…for a **compiler**?

$$A = \alpha(x \times y) + \beta A$$



Technology

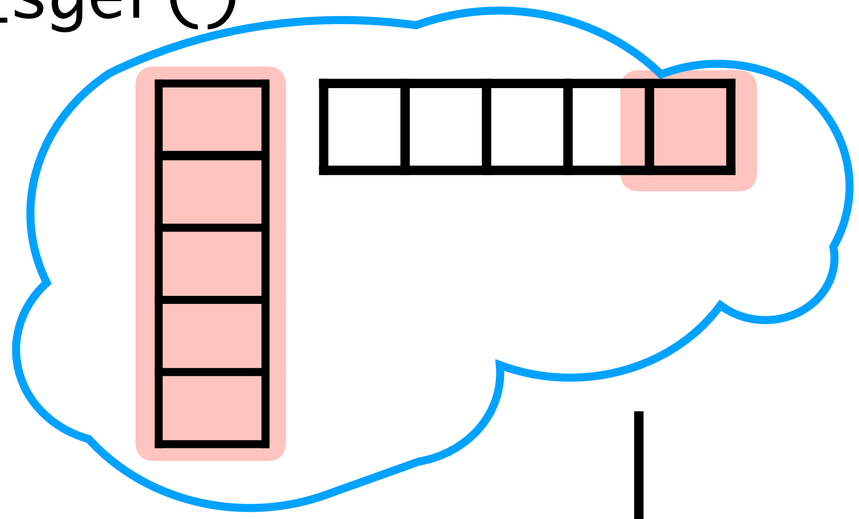
Accelerate

Make large-scale mathematical computations and image performance and low energy consumption.

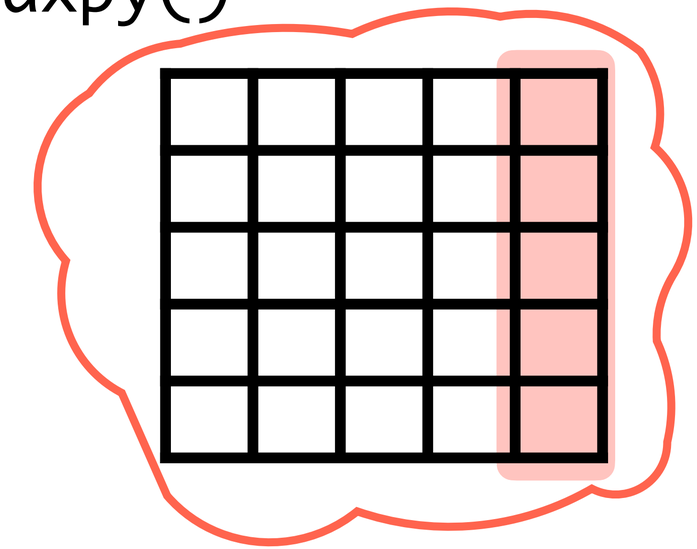
Accelerate | simd | iOS 4.0+ | iPadOS 4.0+ | Mac Catalyst 13.0+ | macO

watchOS 2.0+

`cblas_sger()`

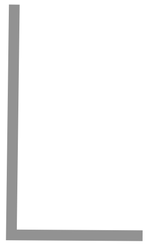


`cblas_saxpy()`



A **Simple** Computation…for a **compiler**?

$$A = \alpha(x \times y) + \beta A$$



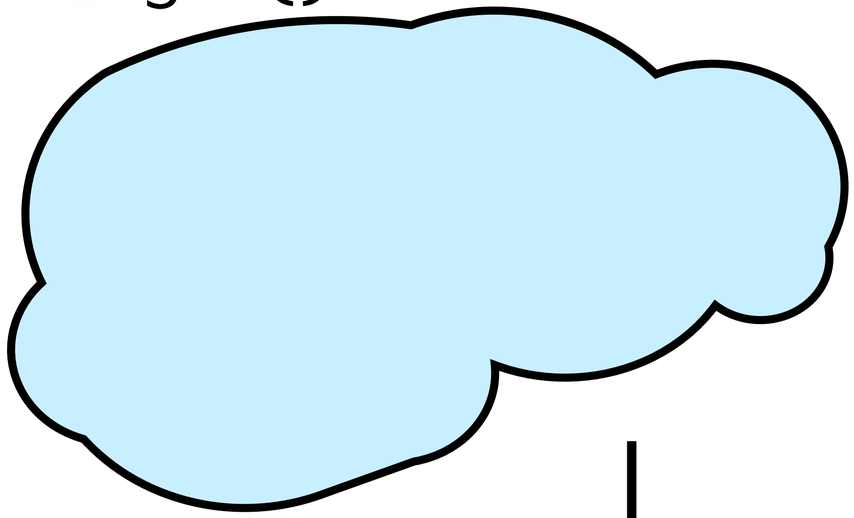
Technology

Accelerate

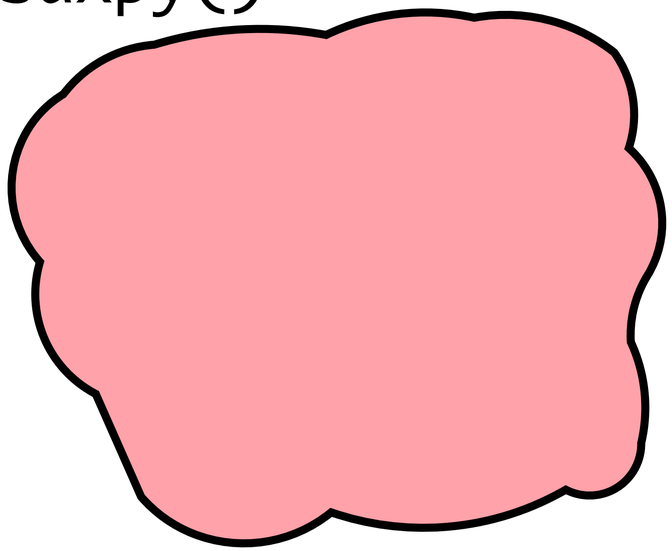
Make large-scale mathematical computations and image
performance and low energy consumption.

Accelerate | simd | iOS 4.0+ | iPadOS 4.0+ | Mac Catalyst 13.0+ | macO
watchOS 2.0+

`cblas_sger()`



`cblas_saxpy()`



A **Simple** Computation…for a **compiler**?

$$A = \alpha(x \times y) + \beta A$$

cblas_sger()

cblas_saxpy()

Technology

Accelerate

Make large-scale mathematical computations and image performance and low energy consumption.

Accelerate | simd | iOS 4.0+ | iPadOS 4.0+ | Mac Catalyst 13.0+ | macO

watchOS 2.0+

A **Simple** Computation…for a **compiler**?

$$A = \alpha(x \times y) + \beta A$$

Technology

Accelerate

Make large-scale mathematical computations and image performance and low energy consumption.

Accelerate | simd | iOS 4.0+ | iPadOS 4.0+ | Mac Catalyst 13.0+ | macO

watchOS 2.0+

cblas_sger()

Key Observation #1: **Black boxing** high performance implementations through function interfaces is a powerful tool.

Key Observation #2: Naive function composition results in **locality** losses.

Question: What information does a compiler need to generate code that exploits locality?

```
#include <Accelerate.h>

void accelerate_sgerb(....){

  for (int j = 0; j < R.n; j += ColTile) {
    cblas_sger(CblasColMajor, m, 1, alpha,
                x.get_data(), 1, &y(0, j),
                1, &R(0, j), R.m);
    cblas_saxpy(m * ColTile, 1.0f,
                &A(0, j), inc_x_y,
                &R(0, j), inc_x_y);
  }

}
```

Question: What information does a compiler need to keep the **control and opacity of function interfaces** while still making their **composition performant**?

```
#include <Accelerate.h>

void accelerate_sgerb(....){

  for (int j = 0; j < R.n; j += ColTile) {
    cblas_sger(CblasColMajor, m, 1, alpha,
                  x.get_data(), 1, &y(0, j),
                  1, &R(0, j), R.m);
    cblas_saxpy(m * ColTile, 1.0f,
                  &A(0, j), inc_x_y,
                  &R(0, j), inc_x_y);
  }

}
```

Question: What information does a compiler need to preserve the **control and opacity of function interfaces** while still making their **composition performant**?

Perf. Eng. can still write whatever they'd like

```
#include <Accelerate.h>

void accelerate_sgerb(....){

  for (int j = 0; j < R.n; j += ColTile) {
    cblas_sger(CblasColMajor, m, 1, alpha,
                  x.get_data(), 1, &y(0, j),
                  1, &R(0, j), R.m);
    cblas_saxpy(m * ColTile, 1.0f,
                  &A(0, j), inc_x_y,
                  &R(0, j), inc_x_y);
  }

}
```

Question: What information does a comp the **control and opacity of function interfaces** while **composition performant**?

Perf. Eng. can still write whatever they'd like

Locality!

```
#include <Accelerate.h>

void accelerate_sgerb(....){

  for (int j = 0; j < R.n; j += ColTile) {
    cblas_sger(CblasColMajor, m, 1, alpha,
                  x.get_data(), 1, &y(0, j),
                  1, &R(0, j), R.m);
    cblas_saxpy(m * ColTile, 1.0f,
                  &A(0, j), inc_x_y,
                  &R(0, j), inc_x_y);
  }

}
```

Question: What information does a compiler need to keep the **control and opacity of function interfaces** while still making their **composition performant**?

```
#include <Accelerate.h>

void accelerate_sgerb(....){

  for (int j = 0; j < R.n; j += ColTile) {
    cblas_sger(CblasColMajor, m, 1, alpha,
                  x.get_data(), 1, &y(0, j),
                  1, &R(0, j), R.m);
    cblas_saxpy(m * ColTile, 1.0f,
                  &A(0, j), inc_x_y,
                  &R(0, j), inc_x_y);
  }

}
```

The Big Idea: **Enrich function interfaces** with **data production and consumption pattern** to automatically **fuse** computation.

Only function signature

The Big Idea: **Enrich function interfaces** with **data production and consumption pattern** to automatically **fuse** computation.

The Big Idea: **Enrich function interfaces** with **da** **nd consumption pattern** to automatically **fuse** computation.

Compute independent subsets

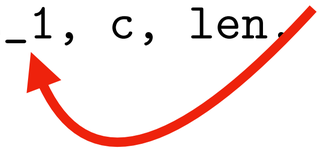
Fern!

Fern!

```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```

Fern!

```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```



Fern!

```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```

```
void my_fused_impl(const Array<float> a,
                   const Array<float> b,
                   const Array<float> c
                   Array<float> out_2,
                   int64_t len){
 int64_t x2 = 0;
 int64_t x0 = x2;
 int64_t len1 = len3;
 Array<float> out_1_q = array_alloc<float>(x0, len1);

 for(int64_t x2 = out_2.idx; x2 < out_2.idx + out_2.size;
 ↪  x2+=len){
    int64_t x0 = x2;
    int64_t len1 = len;
    Array<float> a_q = a.query(x0, len1);
    Array<float> b_q = b.query(x0, len1);
    Array<float> c_q = c.query(x0, len1);
    Array<float> out_2_q = out_2.query(x0, len1);

    vadd(a_q, b_q, len, out_1_q);
    vadd(out_1_q, c_q, len, out_2_q);
}

  out_1_q.destroy();
}
```

Fern!

```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```

```
void my_fused_impl(const Array<float> a,
                   const Array<float> b,
                   const Array<float> c
                   Array<float> out_2,
                   int64_t len){
 int64_t x2 = 0;
 int64_t x0 = x2;
 int64_t len1 = len3;
 Array<float> out_1_q = array_alloc<float>(x0, len1);

 for(int64_t x2 = out_2.idx; x2 < out_2.idx + out_2.size;
 ↪  x2+=len){
    int64_t x0 = x2;
    int64_t len1 = len;
    Array<float> a_q = a.query(x0, len1);
    Array<float> b_q = b.query(x0, len1);
    Array<float> c_q = c.query(x0, len1);
    Array<float> out_2_q = out_2.query(x0, len1);

    vadd(a_q, b_q, len, out_1_q);
    vadd(out_1_q, c_q, len, out_2_q);
}

  out_1_q.destroy();
}
```

Dependency

Fern!

```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```

```
void my_fused_impl(const Array<float> a,
                   const Array<float> b,
                   const Array<float> c
                   Array<float> out_2,
                   int64_t len){
 int64_t x2 = 0;
 int64_t x0 = x2;
 int64_t len1 = len3;
 Array<float> out_1_q = array_alloc<float>(x0, len1);

 for(int64_t x2 = out_2.idx; x2 < out_2.idx + out_2.size;
 ↪  x2+=len){
    int64_t x0 = x2;
    int64_t len1 = len;
    Array<float> a_q = a.query(x0, len1);
    Array<float> b_q = b.query(x0, len1);
    Array<float> c_q = c.query(x0, len1);
    Array<float> out_2_q = out_2.query(x0, len1);

    vadd(a_q, b_q, len, out_1_q);
    vadd(out_1_q, c_q, len, out_2_q);
}

  out_1_q.destroy();
}
```

Query

Fern!

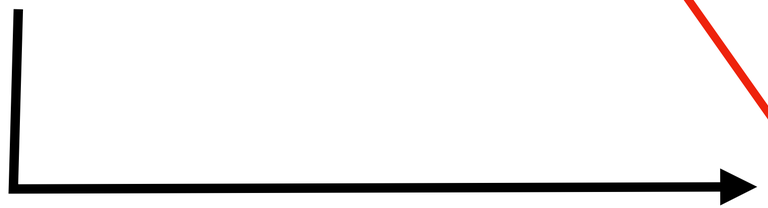
```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```

```
void my_fused_impl(const Array<float> a,
                   const Array<float> b,
                   const Array<float> c
                   Array<float> out_2,
                   int64_t len){
 int64_t x2 = 0;
 int64_t x0 = x2;
 int64_t len1 = len3;
 Array<float> out_1_q = array_alloc<float>(x0, len1);

 for(int64_t x2 = out_2.idx; x2 < out_2.idx + out_2.size;
 ↪  x2+=len){
    int64_t x0 = x2;
    int64_t len1 = len;
    Array<float> a_q = a.query(x0, len1);
    Array<float> b_q = b.query(x0, len1);
    Array<float> c_q = c.query(x0, len1);
    Array<float> out_2_q = out_2.query(x0, len1);

    vadd(a_q, b_q, len, out_1_q);
    vadd(out_1_q, c_q, len, out_2_q);
}

  out_1_q.destroy();
}
```

Compute

Fern!

```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```

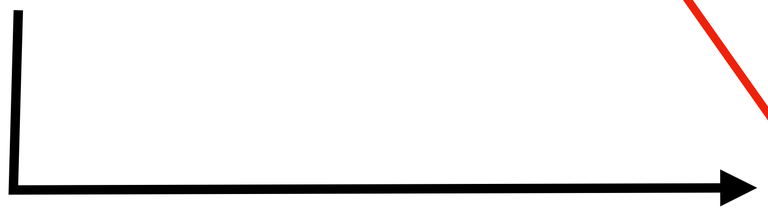


```
void my_fused_impl(const Array<float> a,
                   const Array<float> b,
                   const Array<float> c
                   Array<float> out_2,
                   int64_t len){
 int64_t x2 = 0;
 int64_t x0 = x2;
 int64_t len1 = len3;
 Array<float> out_1_q = array_alloc<float>(x0, len1);

 for(int64_t x2 = out_2.idx; x2 < out_2.idx + out_2.size;
 ↪  x2+=len){
    int64_t x0 = x2;
    int64_t len1 = len;
    Array<float> a_q = a.query(x0, len1);
    Array<float> b_q = b.query(x0, len1);
    Array<float> c_q = c.query(x0, len1);
    Array<float> out_2_q = out_2.query(x0, len1);

    vadd(a_q, b_q, len, out_1_q);
    vadd(out_1_q, c_q, len, out_2_q);
}

  out_1_q.destroy();
}
```

Compute

Fern!

```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```



```
void my_fused_impl(const Array<float> a,
                   const Array<float> b,
                   const Array<float> c
                   Array<float> out_2,
                   int64_t len){
 int64_t x2 = 0;
 int64_t x0 = x2;
 int64_t len1 = len3;
 Array<float> out_1_q = array_alloc<float>(x0, len1);

 for(int64_t x2 = out_2.idx; x2 < out_2.idx + out_2.size;
 ↪  x2+=len){
    int64_t x0 = x2;
    int64_t len1 = len;
    Array<float> a_q = a.query(x0, len1);
    Array<float> b_q = b.query(x0, len1);
    Array<float> c_q = c.query(x0, len1);
    Array<float> out_2_q = out_2.query(x0, len1);

    vadd(a_q, b_q, len, out_1_q);
    vadd(out_1_q, c_q, len, out_2_q);
}

  out_1_q.destroy();
}
```

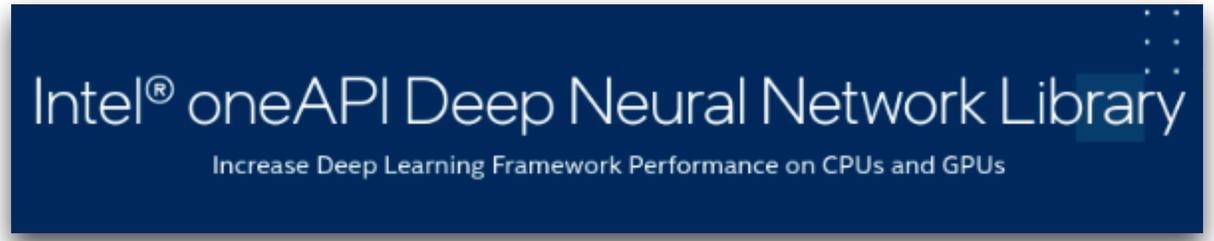
Complete

Fern!

```
for (int i = 0…..){

    // Calculate Dependency

    func1

    func2

    func3

}
```

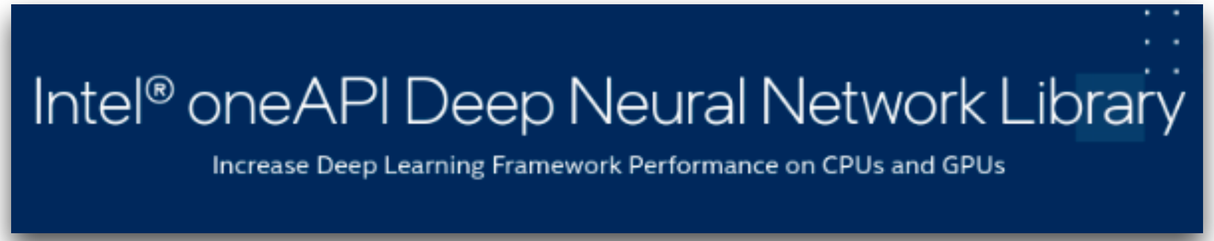
Fern!

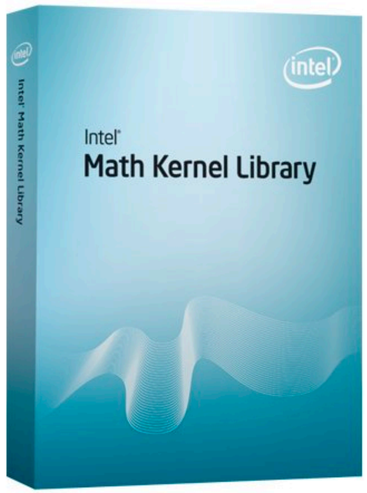
```
for (int i = 0…..){

    // Calculate Dependency

    func1

    func2

    func3

}
```



Fern!

```
for (int i = 0…..){

    // Calculate Dependency

        func1

        func2

        func3

}
```



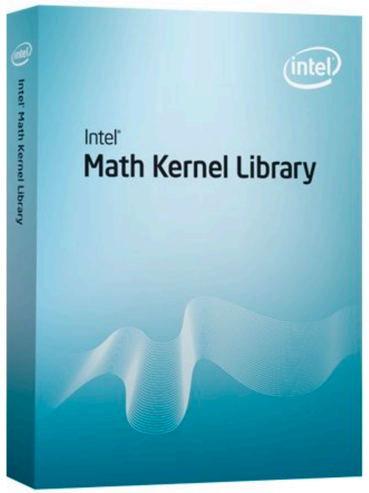


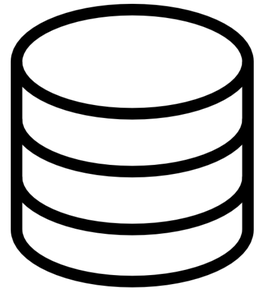
Fern!

```
for (int i = 0…..){

    // Calculate Dependency

    func1

    func2

    func3

}
```

Intel® oneAPI Deep Neural Network Library

Increase Deep Learning Framework Performance on CPUs and GPUs

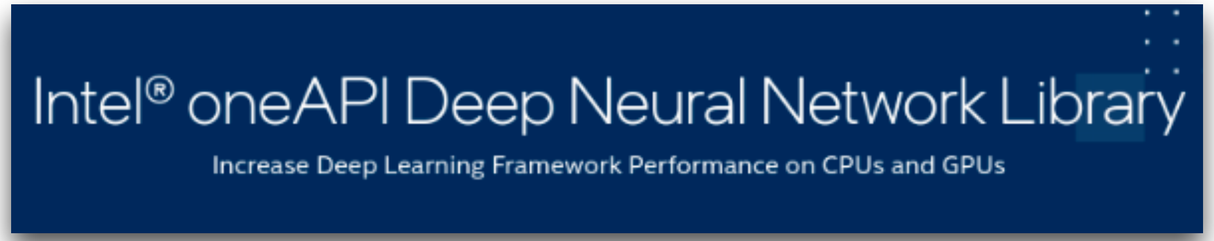


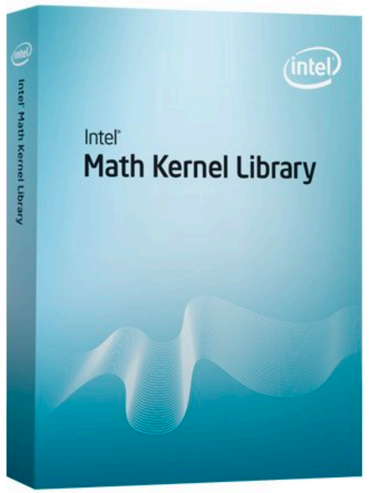


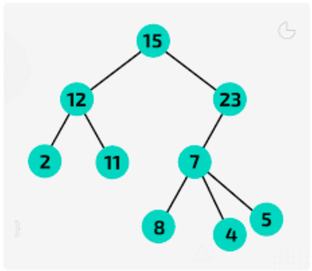
Database

Fern!

```
for (int i = 0…..){

    // Calculate Dependency

    func1

    func2

    func3

}
```

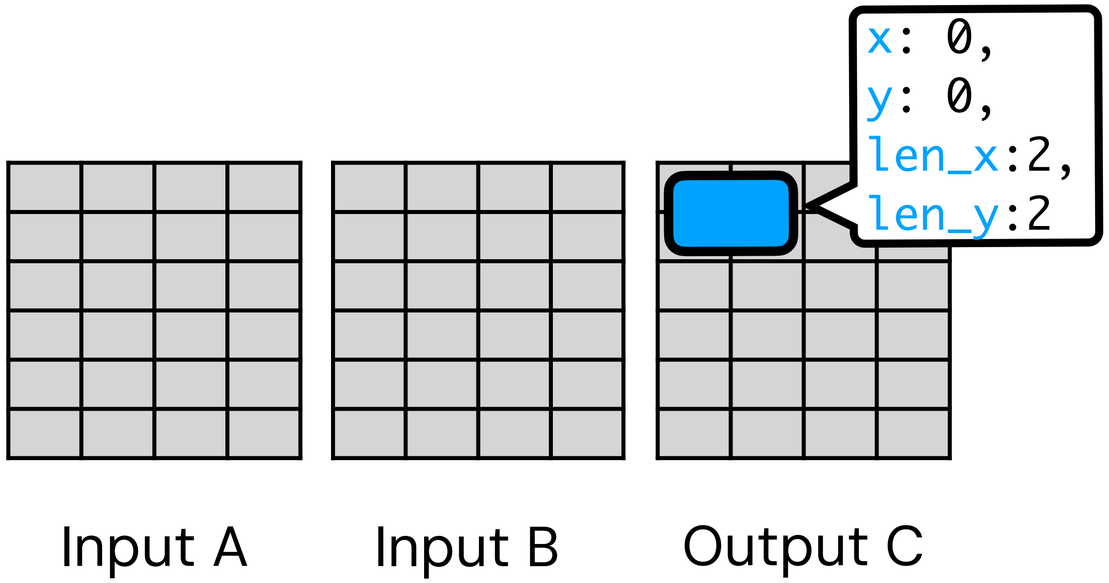




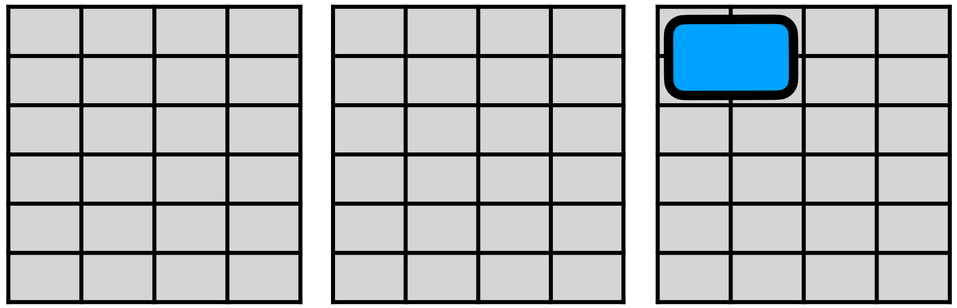


Matrix Multiplication

Matrix Multiplication



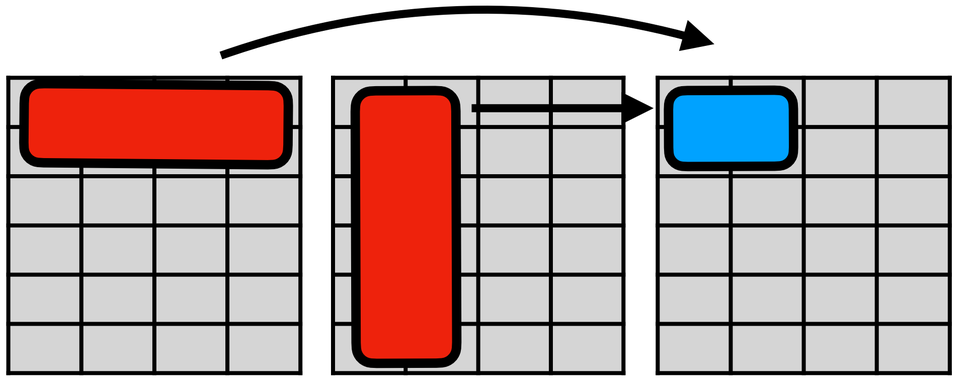
Matrix Multiplication



Input A Input B Output C

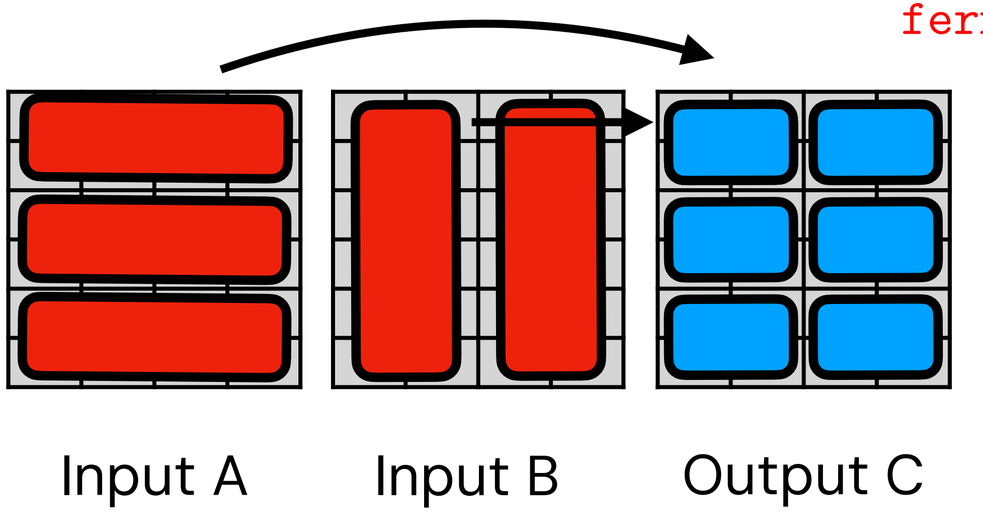
```
fern::Compute(
    fern::Producer(
        C (x, y, len_x, len_y)
    ),
```

Matrix Multiplication



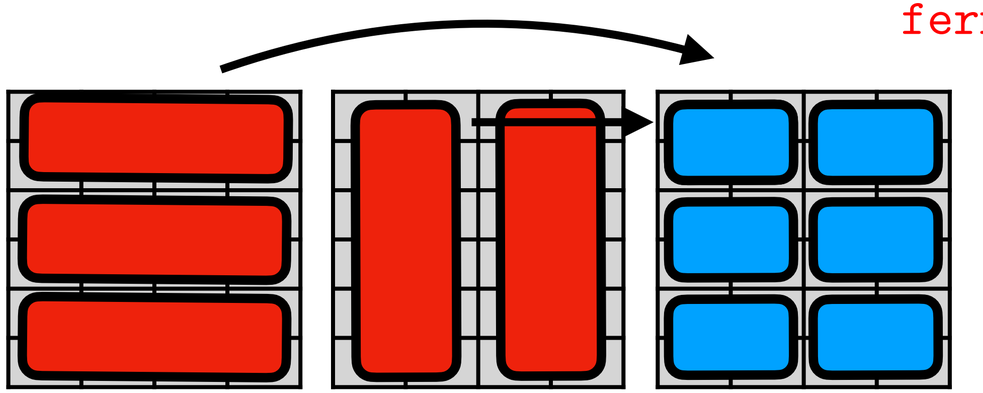

```
fern::Compute(
    fern::Producer(
        C (x, y, len_x, len_y)
    ),
    fern::Consumer({
        A ( x, 0, A.R_len, len_y),
    fern::Consumer(
        B ( 0, y, len_x,  A.C_len)
    })
```

Matrix Multiplication



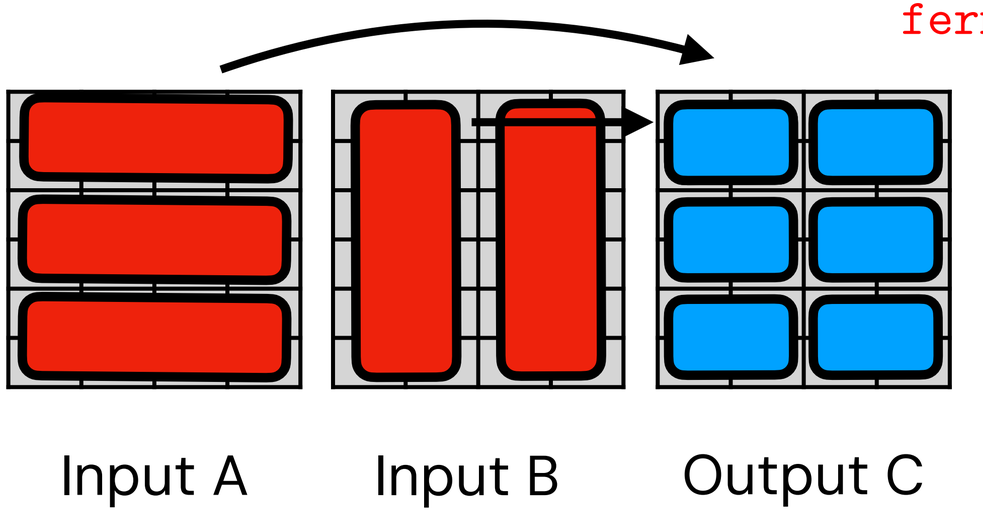

```
fern::Interval (x, C.R_start, C.R_start + C.R_len, len_x)(
fern::Interval (y, C.C_start, C.C_start + C.C_len, len_y)(
    fern::Compute(
        fern::Producer(
            C (x, y, len_x, len_y)
        ),
        fern::Consumer({
            A ( x, 0, A.R_len, len_y),
        fern::Consumer(
            B ( 0, y, len_x,  A.C_len)
        })

))
```

Matrix Multiplication



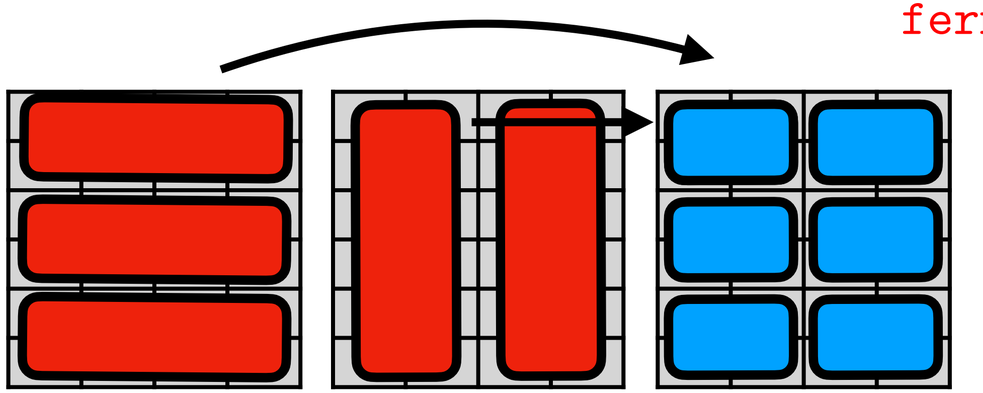

```
fern::Interval (x, C.R_start, C.R_start + C.R_len, len_x)(
fern::Interval (y, C.C_start, C.C_start + C.C_len, len_y)(
    fern::Compute(
        fern::Producer(
            C (x, y, len_x, len_y)
        ),
        fern::Consumer({
            A ( x, 0, A.R_len, len_y),
        fern::Consumer(
            B ( 0, y, len_x,  A.C_len)
        })

))
```

Matrix Multiplication



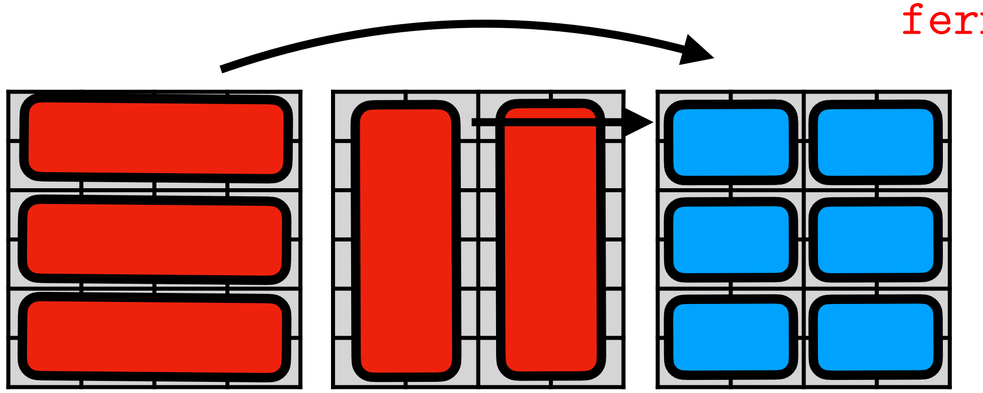

```
fern::Interval (x, C.R_start, C.R_start + C.R_len, len_x)(
fern::Interval (y, C.C_start, C.C_start + C.C_len, len_y)(
    fern::Compute(
        fern::Producer(
            C (x, y, len_x, len_y)
        ),
        fern::Consumer({
            A ( x, 0, A.R_len, len_y),
        fern::Consumer(
            B ( 0, y, len_x,  A.C_len)
        })

))
```

Matrix Multiplication



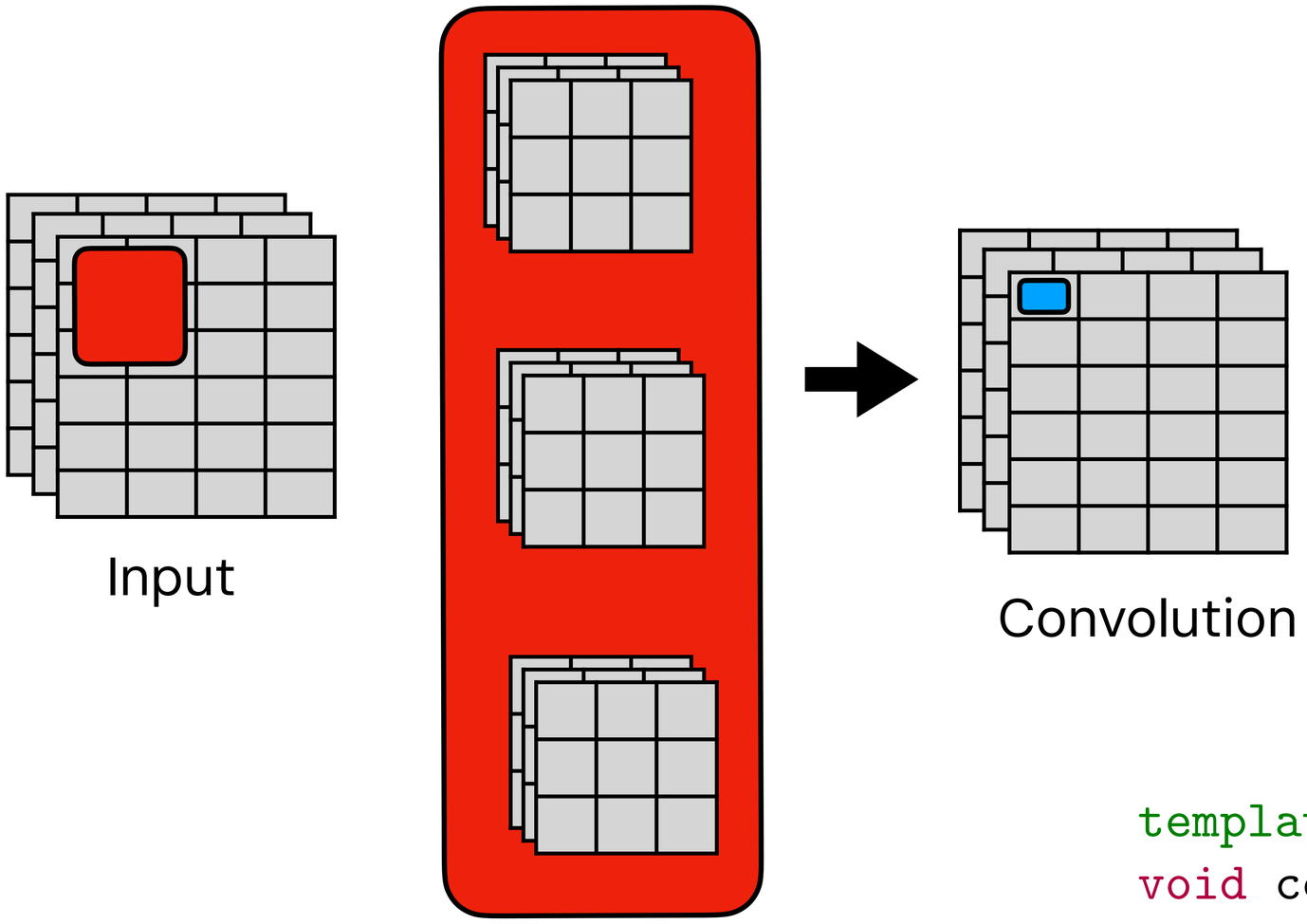

```
fern::Interval (x, C.R_start, C.R_start + C.R_len, len_x)(
fern::Interval (y, C.C_start, C.C_start + C.C_len, len_y)(
    fern::Compute(
        fern::Producer(
            C (x, y, len_x, len_y)
        ),
        fern::Consumer({
            A ( x, 0, A.R_len, len_y),
        fern::Consumer(
            B ( 0, y, len_x,  A.C_len)
        })

))
```

Matrix Multiplication



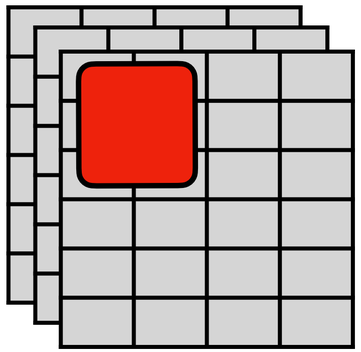

```
fern::Interval (x, C.R_start, C.R_start + C.R_len, len_x)(
fern::Interval (y, C.C_start, C.C_start + C.C_len, len_y)(
    fern::Compute(
        fern::Producer(
            C (x, y, len_x, len_y)
        ),
        fern::Consumer({
            A ( x, 0, A.R_len, len_y),
        fern::Consumer(
            B ( 0, y, len_x,  A.C_len)
        })
))
template <typename T>
void gemm(Matrix<T> A,Matrix<T> B,Matrix<T> C);
```

Convolution

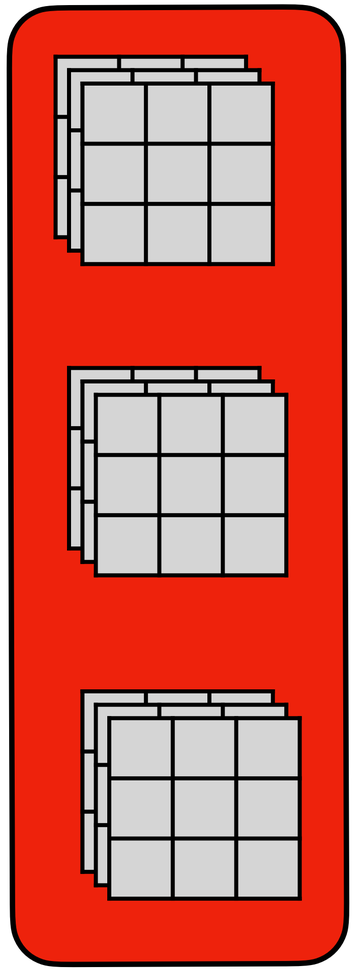



```
template <typename T>
void conv(image<T> input, image<T> filter,
          int StrideArg, image<T> out);
```

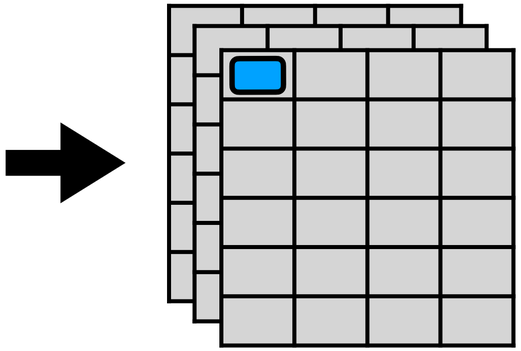
Convolution



Input



Filters

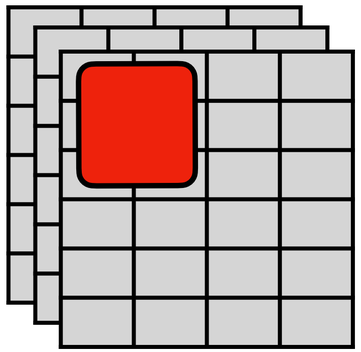


Convolution

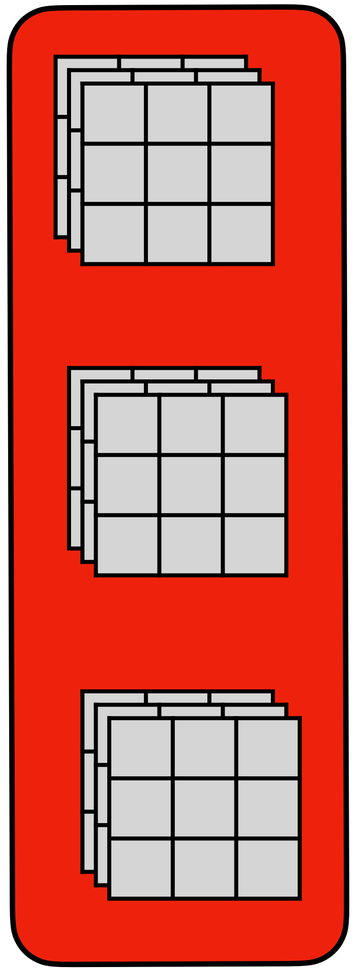
```
fern::Compute(
    fern::Producer(
        out (x, y, len_x, len_y)
    ),
```

```
template <typename T>
void conv(image<T> input, image<T> filter,
          int StrideArg, image<T> out);
```

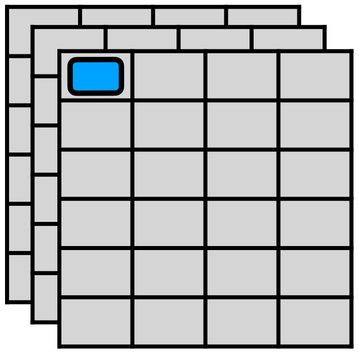
Convolution



Input



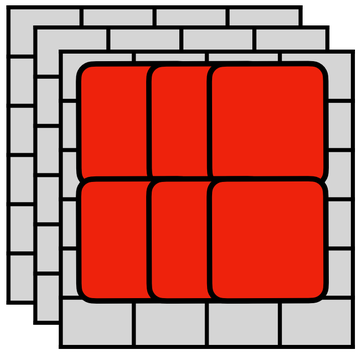
Filters



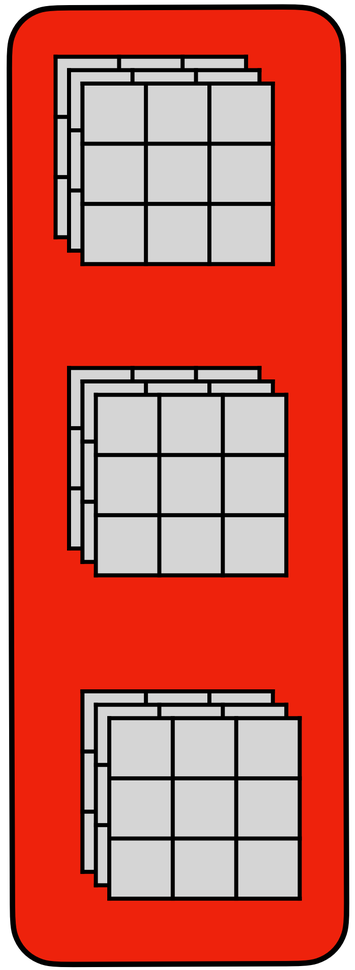
Convolution

```
fern::Compute(
    fern::Producer(
        out (x, y, len_x, len_y)
    ),
    fern::Consumer({
        A ( x * StrideArg,
        y * StrideArg,
        len_x + filter.W - 1,
        len_y + filter.H - 1)
    })
))
template <typename T>
void conv(image<T> input, image<T> filter,
          int StrideArg, image<T> out);
```

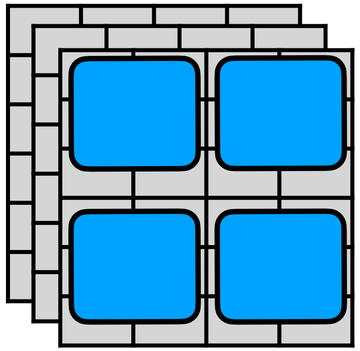
Convolution



Input



Filters



Convolution

```
fern::Interval (x, out.x_start, out.x_start
fern::Interval (y, out.y_start, out.y_start
    fern::Compute(
        fern::Producer(
            out (x, y, len_x, len_y)
        ),
        fern::Consumer({
            A ( x * StrideArg,
            y * StrideArg,
            len_x + filter.W - 1,
            len_y + filter.H - 1)
        })
))
template <typename T>
void conv(image<T> input, image<T> filter,
          int StrideArg, image<T> out);
```

Convolution — In Actual Code

Convolution — In Actual Code

```
class Convolution_Mkl : public fern::AbstractFunctionCall {
public:
  Convolution_Mkl()
      : input(Weights("input")), filter(Weights("filter")),
        bias(FloatPtr("bias")),
        ↪  stride_arg(fern::Variable("stride_arg", true)),
        output(Weights("output")) {}

  std::string getName() const override { return
  ↪  "convolution_mkl"; }

    std::vector<fern::Argument> getArguments() override {
    return {&input,
            &filter,
            &bias,
            &stride_arg,
            &output};
  }
```

Convolution — In Actual Code

```
class Convolution_Mkl : public fern::AbstractFunctionCall {
public:
  Convolution_Mkl()
      : input(Weights("input")), filter(Weights("filter")),
        bias(FloatPtr("bias")),
        ↪  stride_arg(fern::Variable("stride_arg", true)),
        output(Weights("output")) {}

    std::string getName() const override { return
    ↪  "convolution_mkl"; }

    std::vector<fern::Argument> getArguments() override {
    return {&input,
            &filter,
            &bias,
            &stride_arg,
            &output};
  }
```

Convolution — In Actual Code

```
class Convolution_Mkl : public fern::AbstractFunctionCall {
public:
  Convolution_Mkl()
      : input(Weights("input")), filter(Weights("filter")),
        bias(FloatPtr("bias")),
        ↪  stride_arg(fern::Variable("stride_arg", true)),
        output(Weights("output")) {}

  std::string getName() const override { return
  ↪  "convolution_mkl"; }

    std::vector<fern::Argument> getArguments() override {
    return {&input,
            &filter,
            &bias,
            &stride_arg,
            &output};
  }
```

```
                &stride_arg,
                &output};
}
```

Convolution — In Actual Code

```
fern::DependencySubset getDataRelationship() const override {
  fern::Variable x("x"), y("y"), x_tile("x_tile"),
  ↪  y_tile("y_tile");

  fern::DataStructure block_out("out", &output);
  fern::DataStructure block_input("in", &input);

  return
  fern::Interval x_loop(x, output["W_start"], ...)(
      fern::Interval y_loop(y, output["H_start"], ...)(
          fern::ComputationAnnotation(
              fern::Producer(block_out(x, y, x_tile, y_tile)),
              fern::Consumer(
              {block_input(x * stride_arg,
              y * stride_arg,
              x_tile + filter["H"] - 1,
              y_tile + filter["W"] - 1)}))));
}
```

```
                &stride_arg,
                &output};
}
```

Convolution — In Actual Code

```
fern::DependencySubset getDataRelationship() const override {
  fern::Variable x("x"), y("y"), x_tile("x_tile"),
  ↪  y_tile("y_tile");

  fern::DataStructure block_out("out", &output);
  fern::DataStructure block_input("in", &input);

  return
  fern::Interval x_loop(x, output["W_start"], ...)(
      fern::Interval y_loop(y, output["H_start"], ...)(
          fern::ComputationAnnotation(
              fern::Producer(block_out(x, y, x_tile, y_tile)),
              fern::Consumer(
              {block_input(x * stride_arg,
              y * stride_arg,
              x_tile + filter["H"] - 1,
              y_tile + filter["W"] - 1)}))));
}
```

```
                &stride_arg,
                &output};
}
```

Convolution — In Actual Code

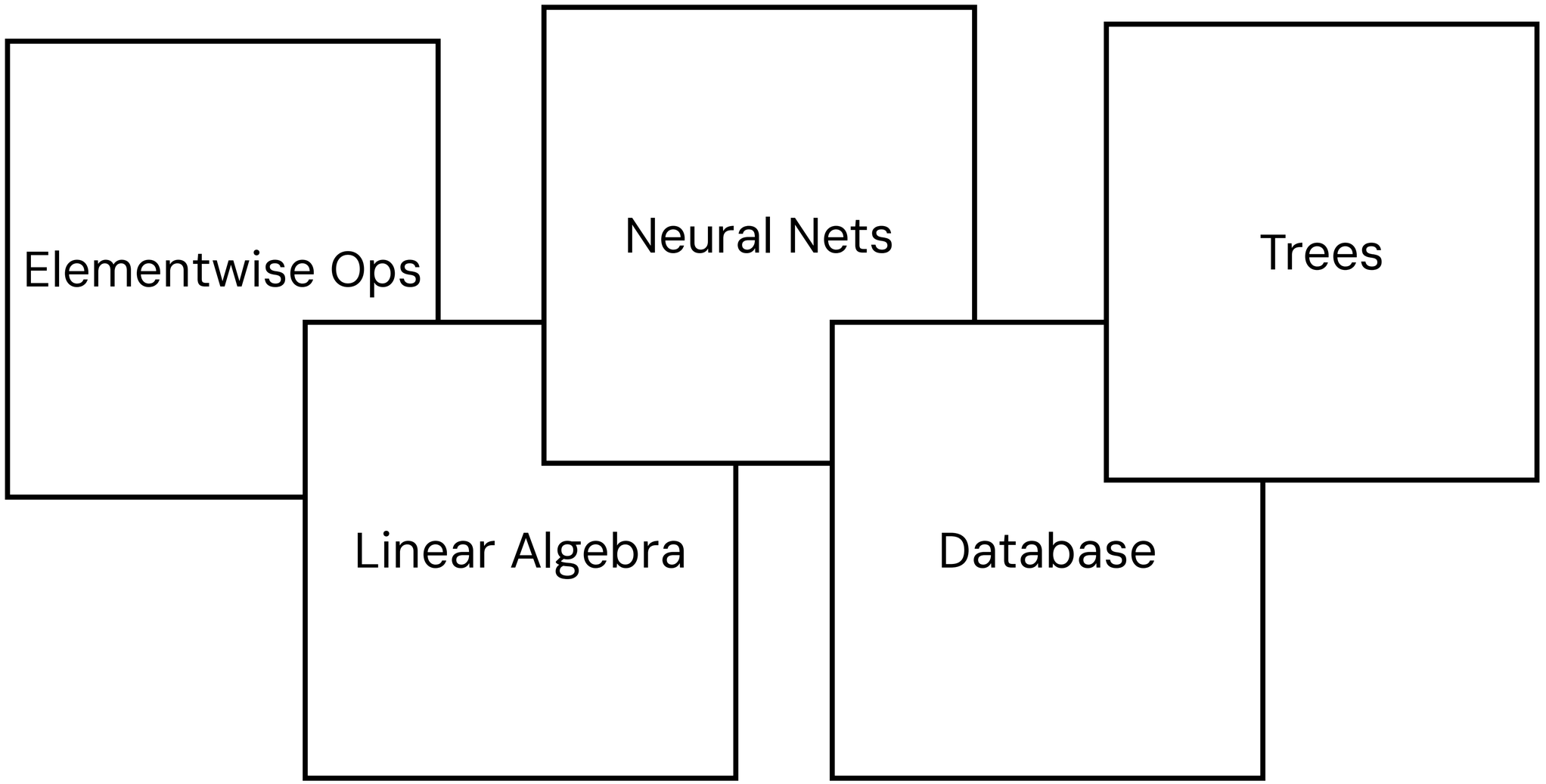
```
fern::DependencySubset getDataRelationship() const override {
  fern::Variable x("x"), y("y"), x_tile("x_tile"),
  ↪  y_tile("y_tile");

  fern::DataStructure block_out("out", &output);
  fern::DataStructure block_input("in", &input);

  return
  fern::Interval x_loop(x, output["W_start"], ...)(
      fern::Interval y_loop(y, output["H_start"], ...)(
          fern::ComputationAnnotation(
              fern::Producer(block_out(x, y, x_tile, y_tile)),
              fern::Consumer(
              {block_input(x * stride_arg,
              y * stride_arg,
              x_tile + filter["H"] - 1,
              y_tile + filter["W"] - 1)}))));
}
```

Convolution — In Actual Code

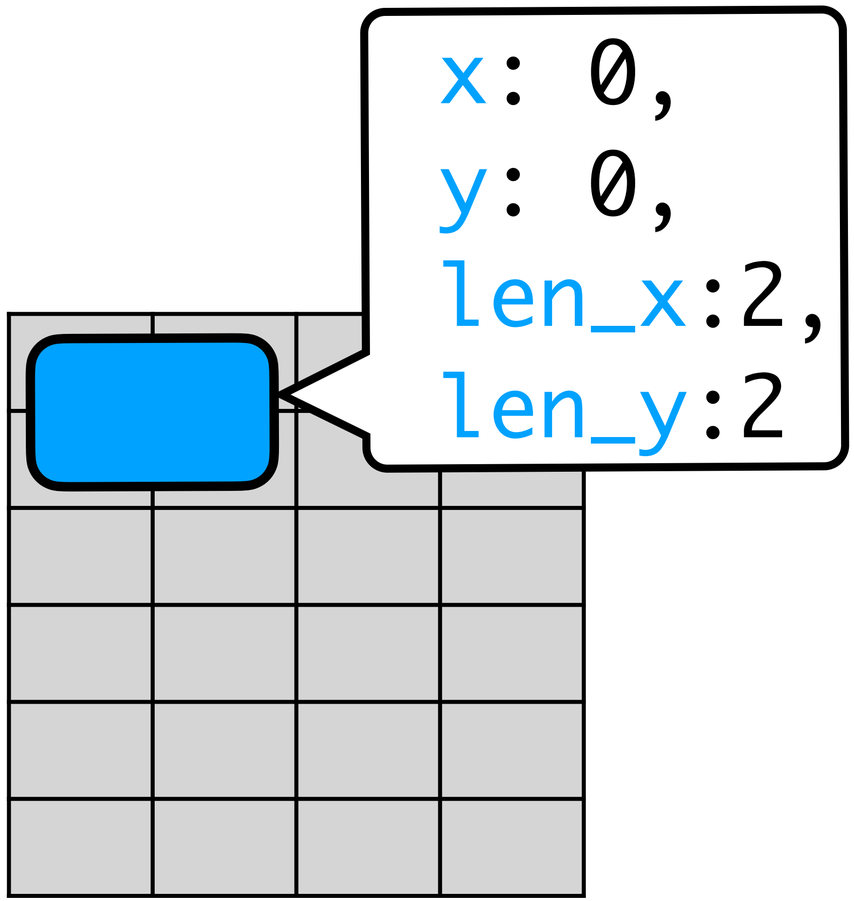
- Pre-processor
- Automatically generated

```
class Convolution_Mkl : public fern::AbstractFunctionCall {
public:
  Convolution_Mkl()
      : input(Weights("input")), filter(Weights("filter")),
        bias(FloatPtr("bias")),
        ↪  stride_arg(fern::Variable("stride_arg", true)),
        output(Weights("output")) {}

  std::string getName() const override { return
  ↪  "convolution_mkl"; }

    std::vector<fern::Argument> getArguments() override {
    return {&input,
            &filter,
            &bias,
            &stride_arg,
            &output};
  }
```

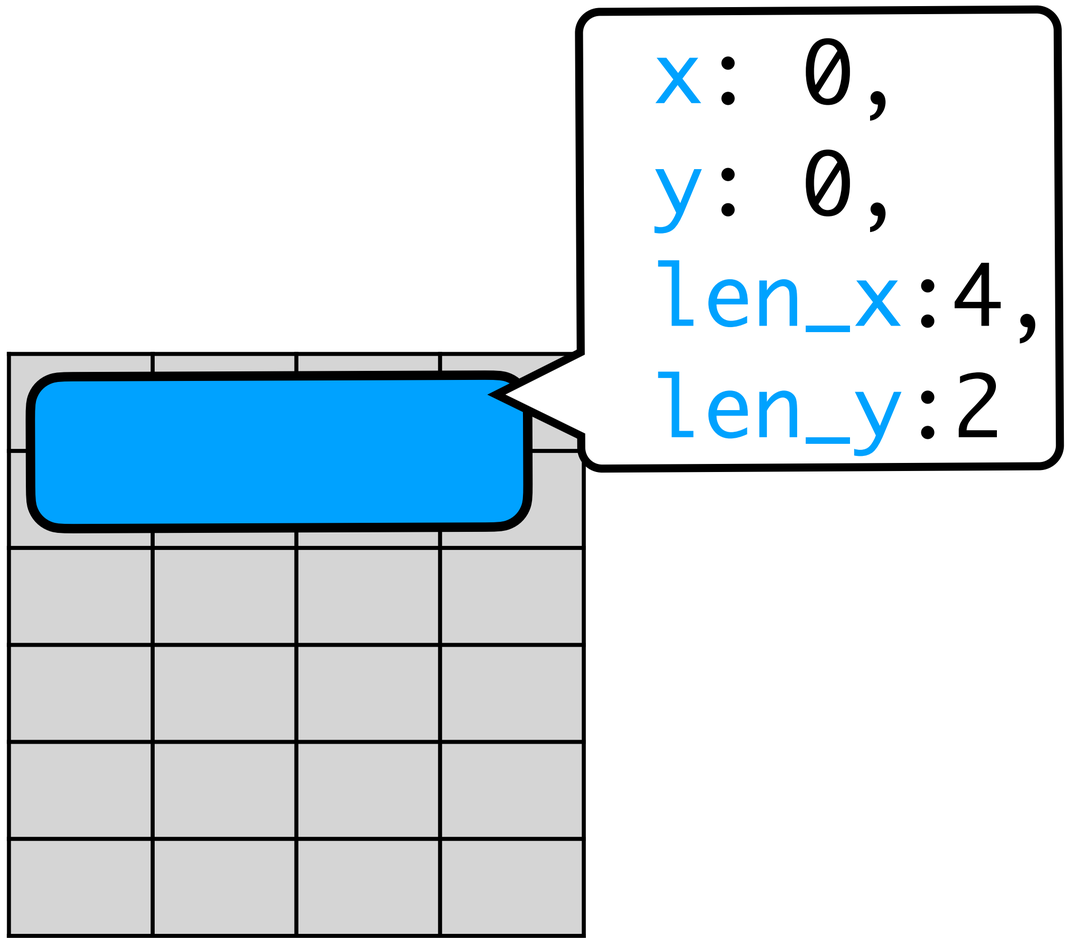
A Library of Data Dependencies



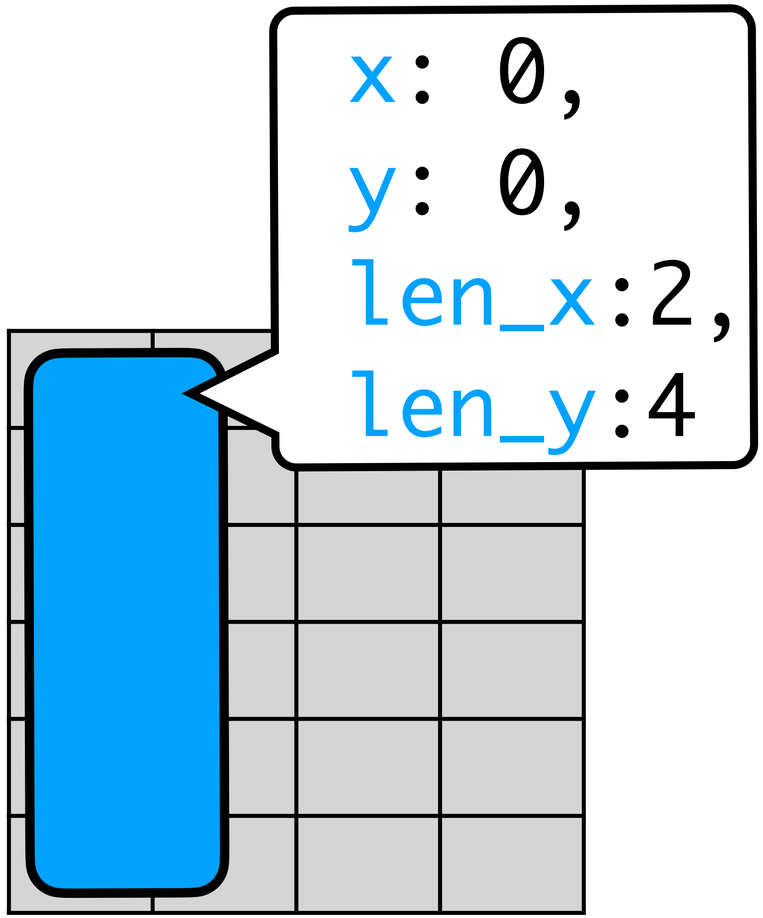
Subset Decomposition



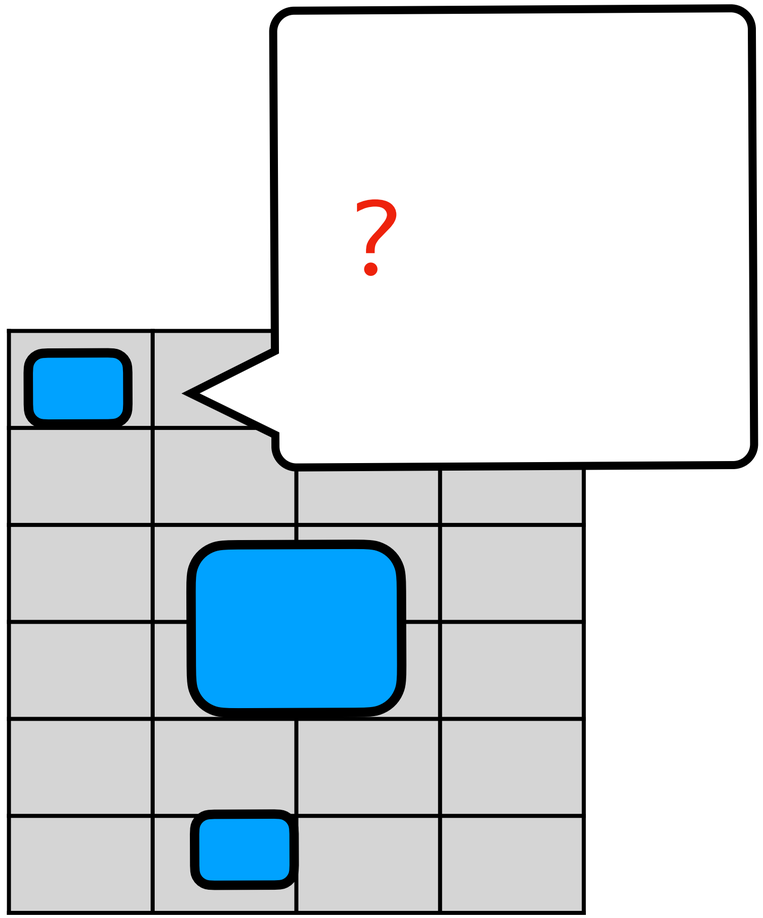
Subset Decomposition



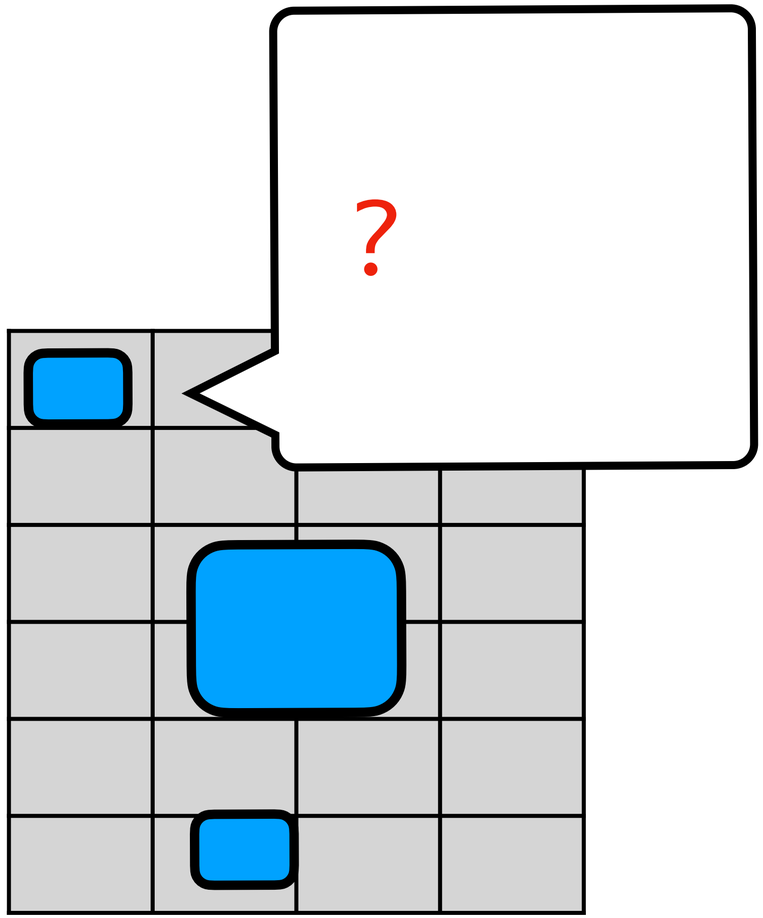
Subset Decomposition



Subset Decomposition

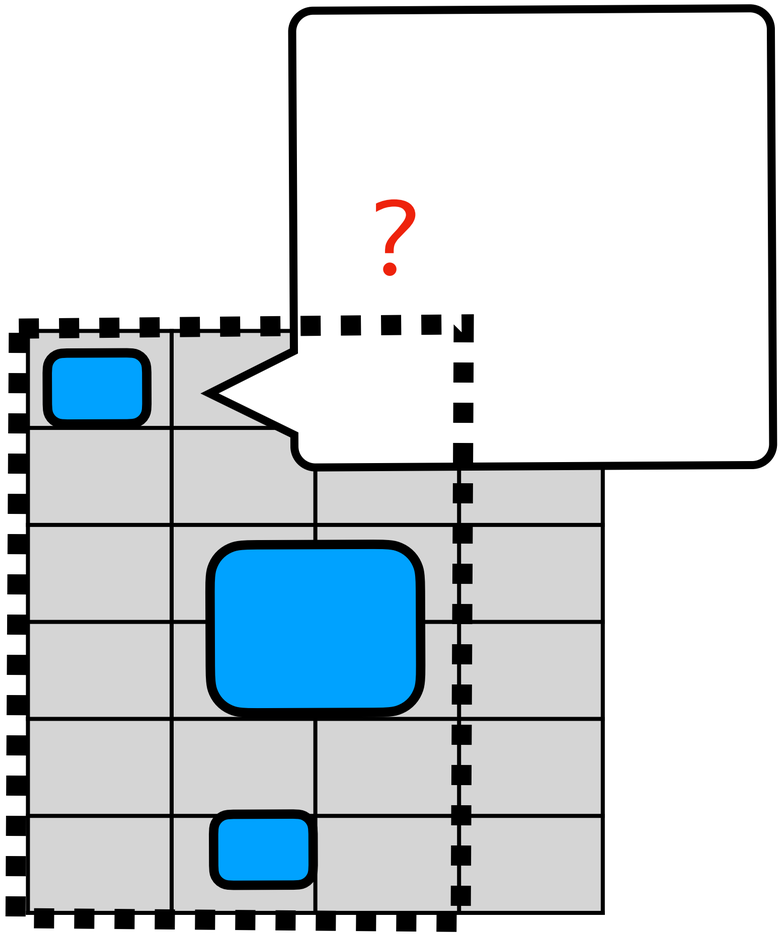


Subset Decomposition



Functions act on **self-similar** data-structures. Fern can **hierarchal decompose** the data-structures.

Subset Decomposition



Functions act on **self-similar** data-structures. Fern can **hierarchal decompose** the data-structures.

Adding Data-Structures to Fern

```
class Weights : public fern::AbstractDataStructure {
public:
  Weights(const std::string &name) : name(name) {}

  Weights() : Weights(fern::util::uniqueName("weight")) {}

  std::string getTypeName() const override { return
  ↪  "Weights<float>"; }

  std::string getDataQueryInterface() const override {
    return {"query_materialize"};
  }

  std::vector<std::string> getMetaData() const override {
    return {"x", "y", "len_x", "lex_y"};
```

Adding Data-Structures to Fern

```
class Weights : public fern::AbstractDataStructure {
public:
  Weights(const std::string &name) : name(name) {}

  Weights() : Weights(fern::util::uniqueName("weight")) {}

  std::string getTypeName() const override { return
  ↪  "Weights<float>"; }

  std::string getDataQueryInterface() const override {
    return {"query_materialize"};
  }

  std::vector<std::string> getMetaData() const override {
    return {"x", "y", "len_x", "lex_y"};
```

Adding Data-Structures to Fern

```
Weights() : Weights(fern::util::uniqueName("weight")) {}

std::string getTypeName() const override { return
↪  "Weights<float>"; }

std::string getDataQueryInterface() const override {
  return {"query_materialize"};
}

std::vector<std::string> getMetaData() const override {
  return {"x", "y", "len_x", "lex_y"};
}

std::string getDataInsertInterface() const override {
  return {"insert_materialize"};
}
```

Adding Data-Structures to Fern

```
class Weights : public fern::AbstractDataStructure {
public:
  Weights(const std::string &name) : name(name) {}

  Weights() : Weights(fern::util::uniqueName("weight")) {}

  std::string getTypeName() const override { return
  ↪  "Weights<float>"; }

  std::string getDataQueryInterface() const override {
    return {"query_materialize"};
  }

  std::vector<std::string> getMetaData() const override {
    return {"x", "y", "len_x", "lex_y"};
  }
```

Adding Data-Structures to Fern

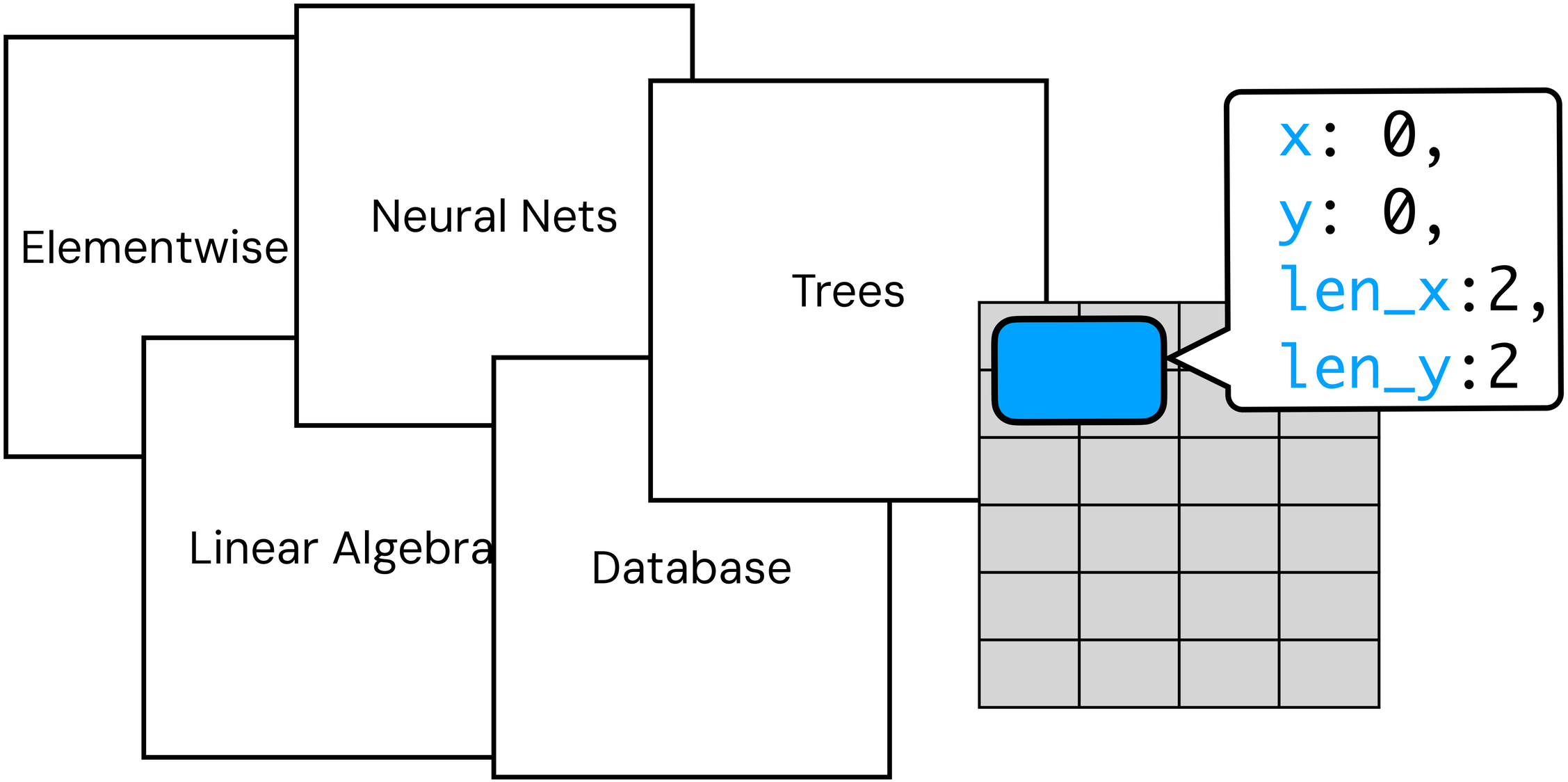
```
    ↪  "Weights<float>"; }

    std::string getDataQueryInterface() const override {
      return {"query_materialize"};
    }

    std::vector<std::string> getMetaData() const override {
      return {"x", "y", "len_x", "lex_y"};
    }

    std::string getDataInsertInterface() const override {
      return {"insert_materialize"};
    }

    std::string getAllocData() const override { return
    ↪  {"weight_alloc"}; }
```

Adding Data-Structures to Fern

```
    return {"query_materialize"};
}

std::vector<std::string> getMetaData() const override {
  return {"x", "y", "len_x", "lex_y"};
}

std::string getDataInsertInterface() const override {
  return {"insert_materialize"};
}
```

```
std::string getAllocData() const override { return
↪  {"weight_alloc"}; }
```

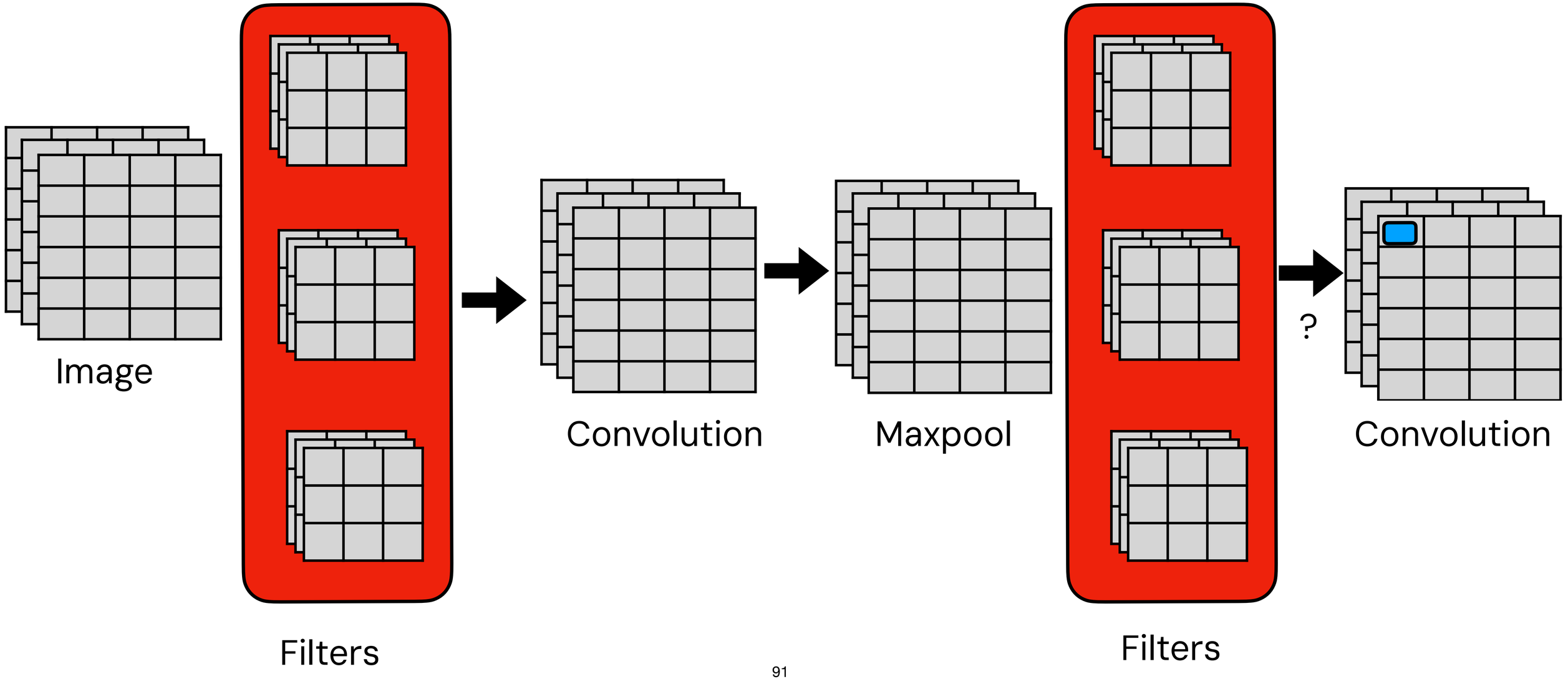
A Library of Data Dependencies & Data Structures



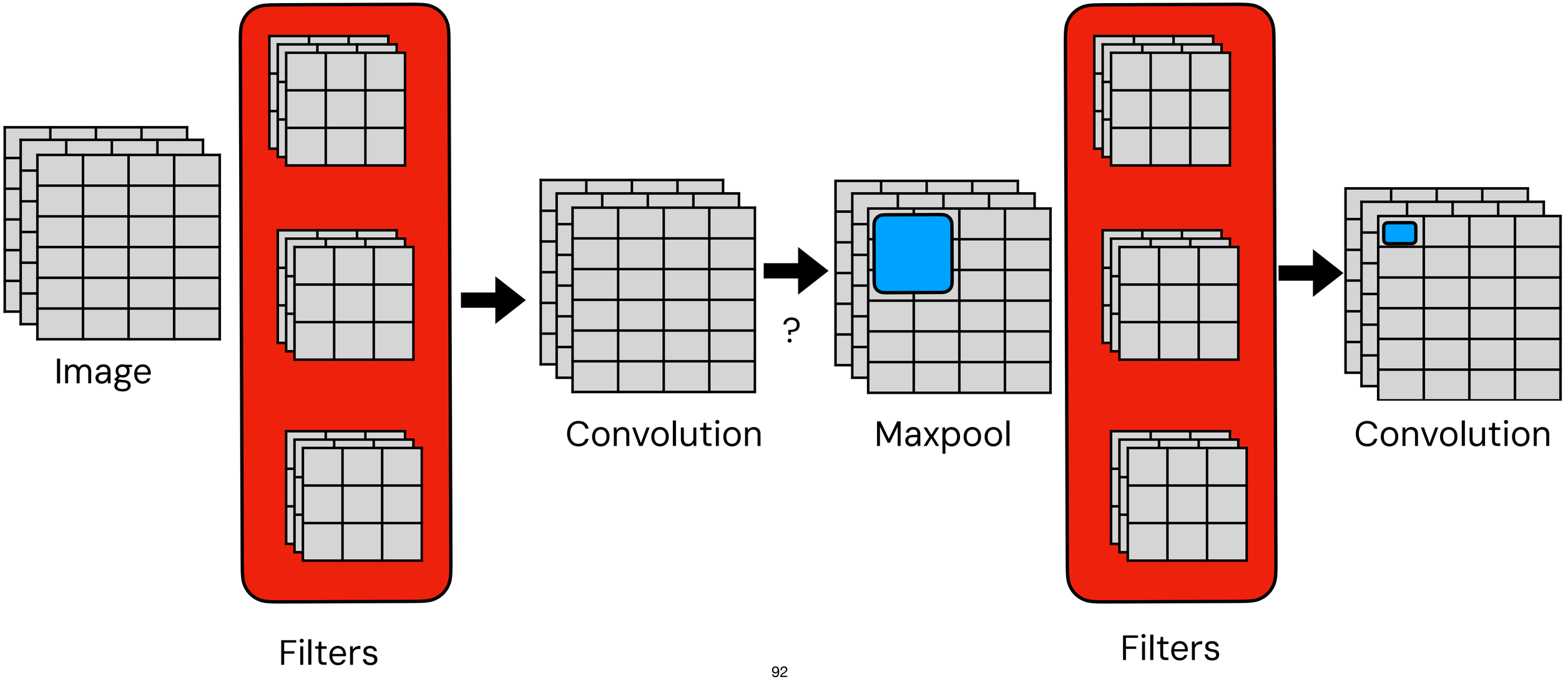
Write a Pipeline

```
/* Make a pipeline of functions using the classes inhertied from fern::AbstractFunctionCall
The () operator is overloaded to mmimc a normal function call. */
Pipeline pipeline({
      convolution_mkl(&input, &filter, &bias, getNode(stride_arg), &output_conv1),
      maxpool(&output_conv1, getNode(maxpool_dim), &output_max1),
      convolution_mkl(&output_max1, &filter, &bias, getNode(stride_arg), &output_conv2),

});
pipeline.constructPipeline().finalize();
```

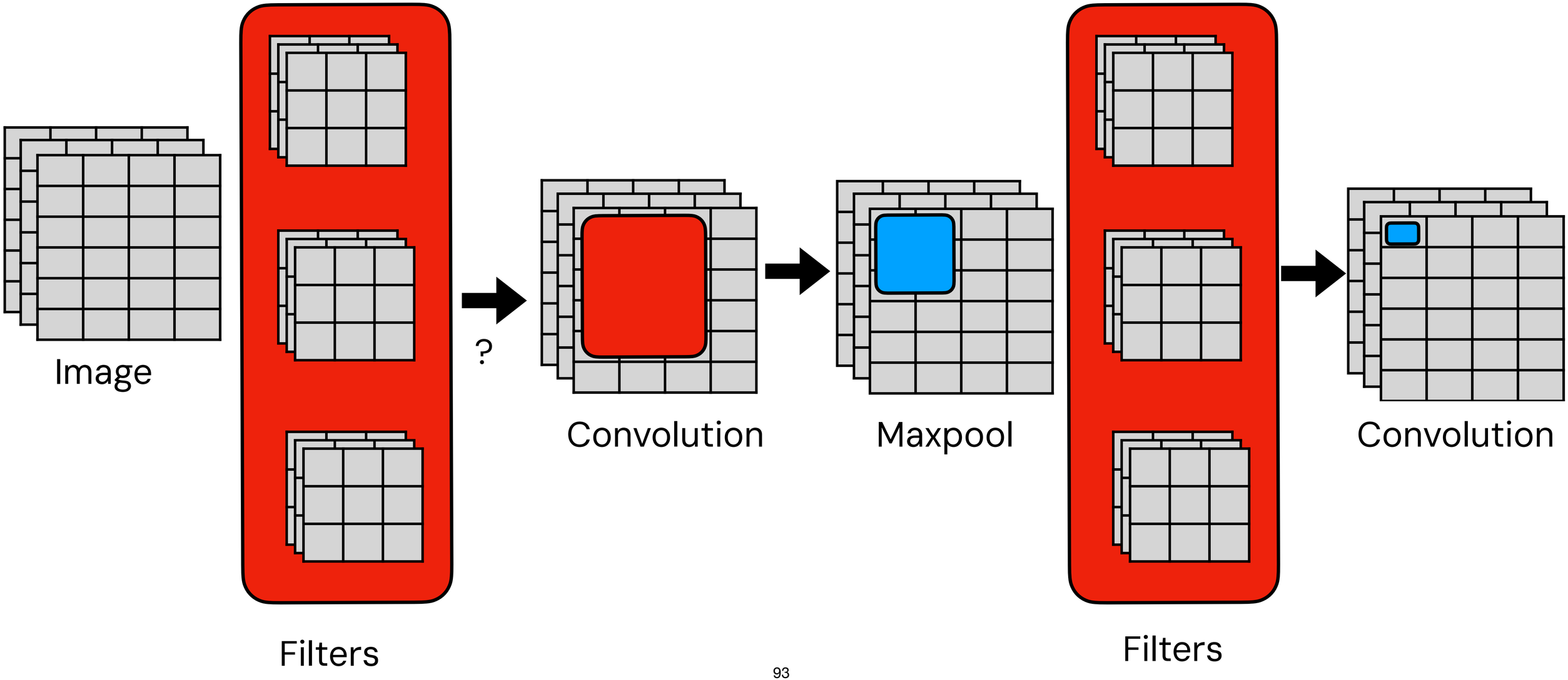
Convolution — Maxpool — Convolution



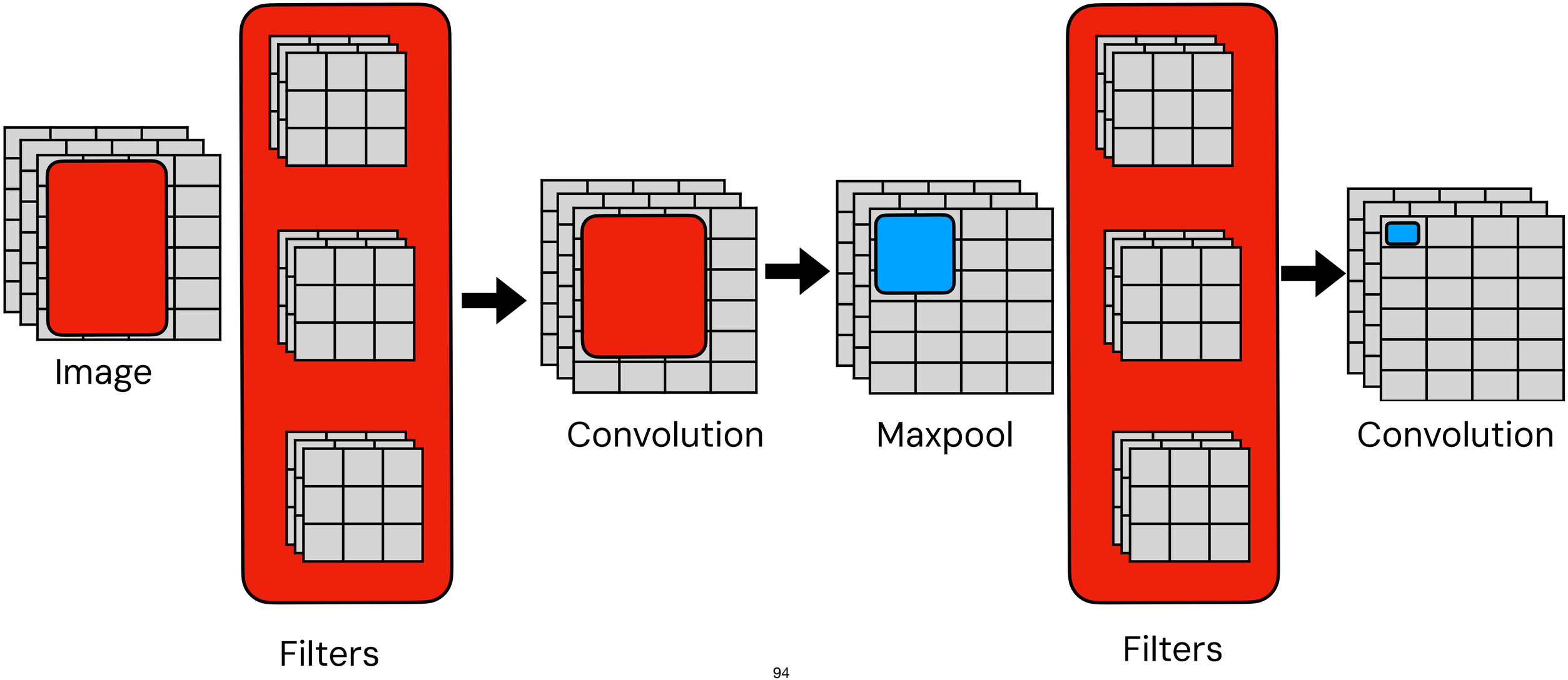
Convolution — Maxpool — Convolution



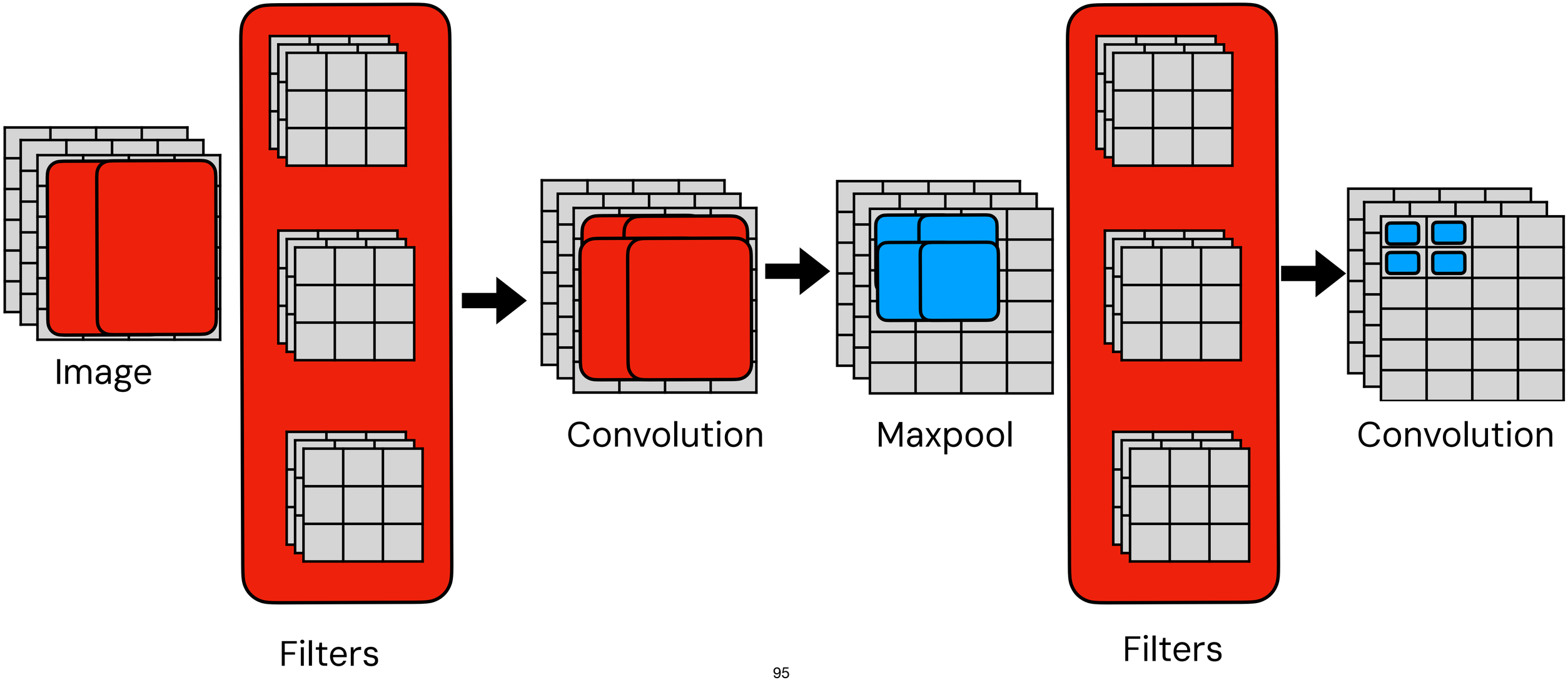
Convolution — Maxpool — Convolution



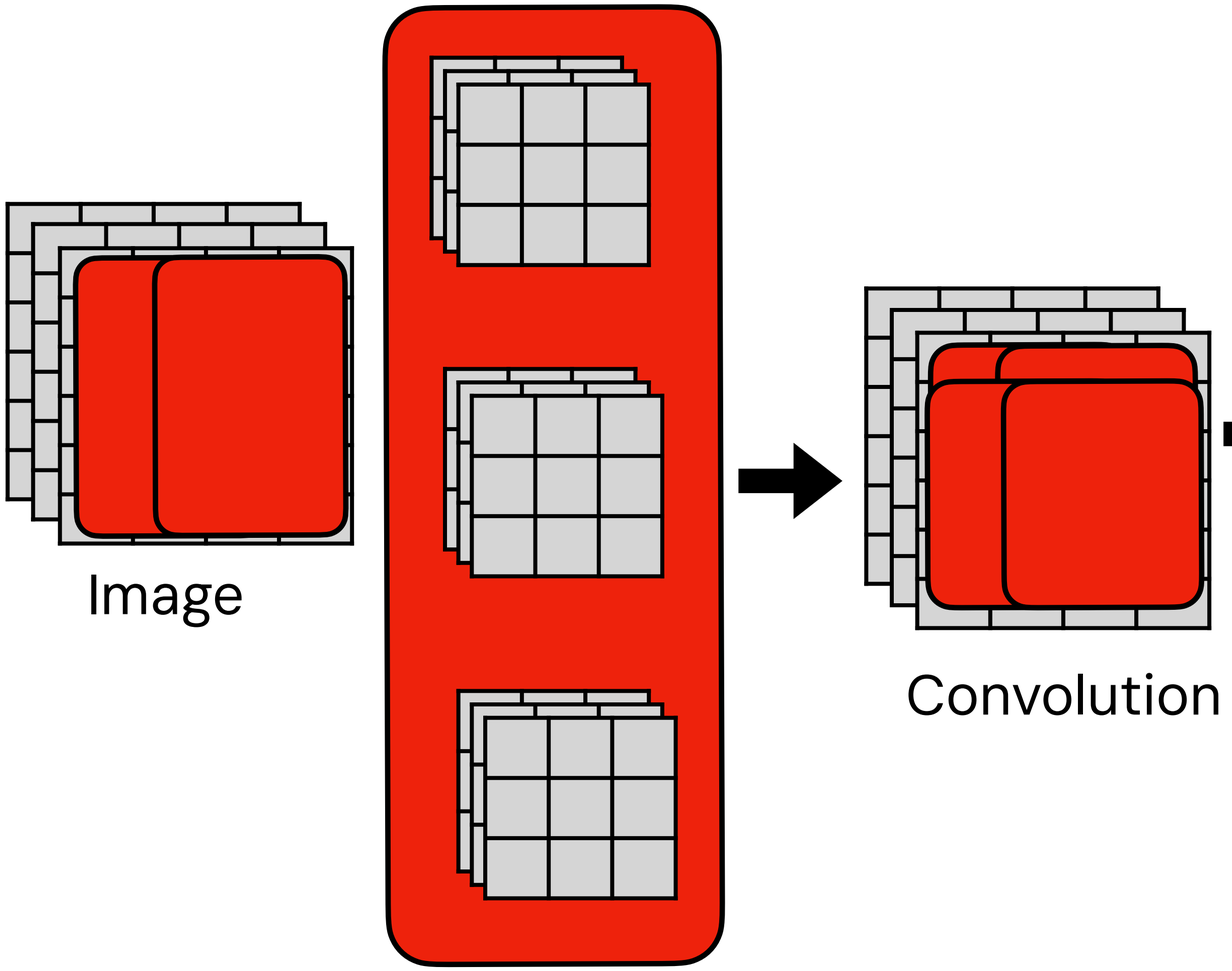
Convolution — Maxpool — Convolution



Convolution — Maxpool — Convolution

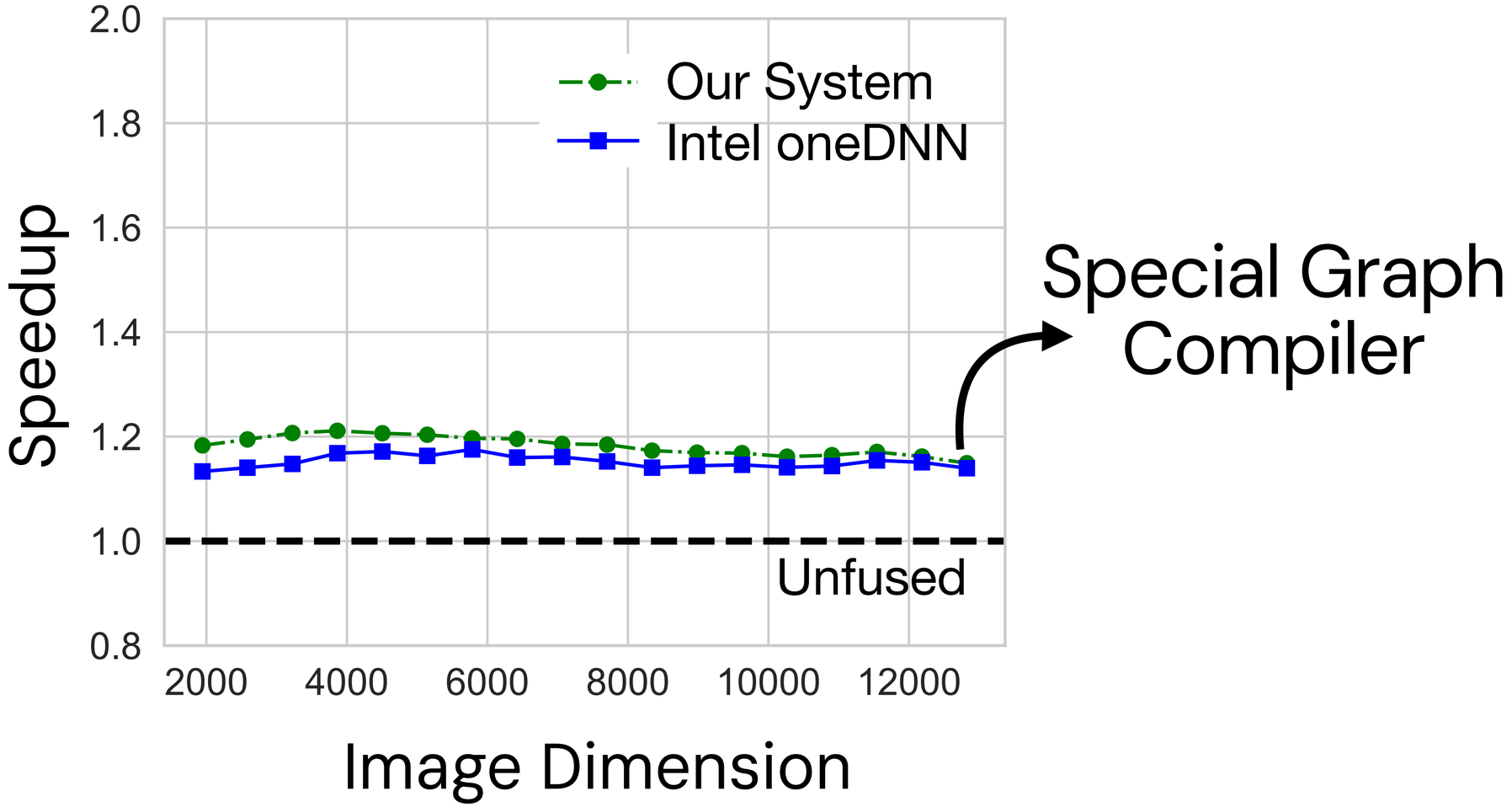


Convolution — Maxp




```
for(int64_t x85 = output_t.W_start;x85 < output_t.W_start +
↪  output_t.LW; x85+=x_tile84){
 for(int64_t y83 = output_t.H_start;y83 < output_t.H_start +
 ↪  output_t.LH; y83+=y_tile86){
     int64_t x78 = maxpool_dim87*x85;
      int64_t y79 = y83*maxpool_dim87;
      int64_t x_tile80 = maxpool_dim87*x_tile84;
      int64_t y_tile81 = maxpool_dim87*y_tile86;
      int64_t x73 = x78*stride_arg82;
      int64_t y75 = y79*stride_arg82;
      int64_t x_tile74 = -1+x_tile80+filter0.H;
      int64_t y_tile76 = -1+y_tile81+filter0.W;
      int64_t x70 = maxpool_dim77*x73;
      int64_t y71 = maxpool_dim77*y75;
      int64_t x_tile68 = maxpool_dim77*x_tile74;
      int64_t y_tile72 = maxpool_dim77*y_tile76;
      Weights<float> input_q18 = input.query_materialize(x70 *
      ↪  stride_arg69, y71 * stride_arg69, x_tile68 + filter0.H
      ↪  - 1, y_tile72 + filter0.W - 1);
      Weights<float> output_t_q =
      ↪  output_t.query_materialize(x85, y83, x_tile84,
      ↪  y_tile86);
         conv_new_mkl(input_q18, filter0, bias, stride_arg69,
         ↪  output_conv1_q);
         maxpool(output_conv1_q, maxpool_dim77, output_max1_q);
         conv_new_mkl(output_max1_q, filter0, bias,
         ↪  stride_arg82, output_conv2_q);
         maxpool(output_conv2_q, maxpool_dim87, output_t_q);
         output_t.insert_materialize(x85, y83, x_tile84,
         ↪  y_tile86, output_t_q);
         input_q18.free_weight();
   }
}
```

Performance



Controlling Properties of the Fused Code

Controlling Properties of the Fused Code

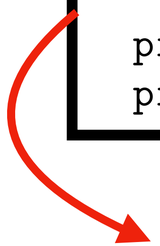
```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```

Controlling Properties of the Fused Code

```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```

Controlling Properties of the Fused Code

```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```



```
pipeline.split(1, outer, inner, outer_step, 4);
```

Controlling Properties of the Fused Code

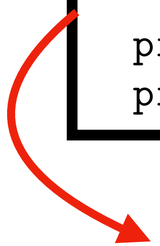
```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```

```
pipeline.split(1, outer, inner, outer_step, 4);
```

```
for (int x=start,x<end; x++){

  …

  }
```

Controlling Properties of the Fused Code

```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```



```
pipeline.split(1, outer, inner, outer_step, 4);
```

```
for (int o=start,i<o;o+=4){
  for (int i=0, i<4; i++){
  …
  …
    }
  }
```

Controlling Properties of the Fused Code

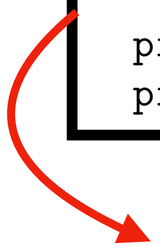
```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```

```
pipeline.split(1, outer, inner, outer_step, 4);
pipeline.parrallelize(1);
```

```
for (int o=start,i<o;o+=4){
  for (int i=0, i<4; i++){
    …
    …
  }
}
```

Controlling Properties of the Fused Code

```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```



```
pipeline.split(1, outer, inner, outer_step, 4);
pipeline.parrallelize(1);
pipeline.bind(len_var, 4);
```

Controlling Properties of the Fused Code

```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```

```
pipeline.split(1, outer, inner, outer_step, 4);
pipeline.parrallelize(1);
pipeline.bind(len_var, 4);
pipeline.breakPipeline(0);
```

```
for x {
  f1();
  f2();
  }
```

```
for x {
  f1();
  }
  - - - Break
for y{
  f2();
  }
```

Controlling Properties of the Fused Code

```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```

```
pipeline.split(1, outer, inner, outer_step, 4);
pipeline.parrallelize(1);
pipeline.bind(len_var, 4);
pipeline.breakPipeline(0);
```

```
for x {
  f1();
  f2();
}
```

```
for x {
  f1();
  }
  - - - Break
for y{
  f2();
  }
```

Controlling Properties of the Fused Code

```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```

```
pipeline.split(1, outer, inner, outer_step, 4);
pipeline.parrallelize(1);
pipeline.bind(len_var, 4);
pipeline.breakPipeline(0);
subpipe = pipeline.subpipeline(0, 1);
```

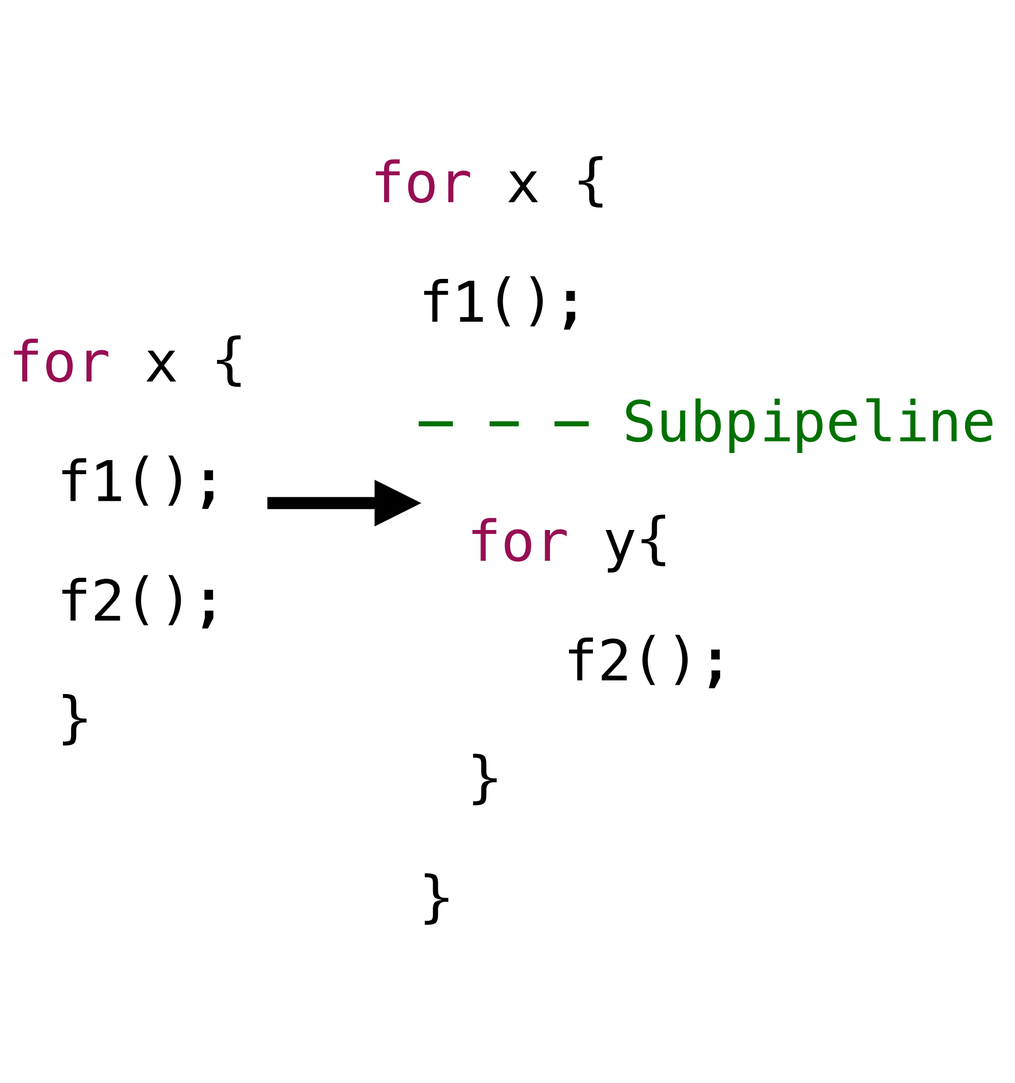
```
for x {
  f1();
  f2();
}
```

```
for x {
  f1();
  - - - Subpipeline
    for y{
      f2();
    }
}
```

Controlling Properties of the Fused Code

```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```

```
pipeline.split(1, outer, inner, outer_step, 4);
pipeline.parrallelize(1);
pipeline.bind(len_var, 4);
pipeline.breakPipeline(0);
subpipe = pipeline.subpipeline(0, 1);
```



Controlling Properties of the Fused Code

```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```

```
pipeline.split(1, outer, inner, outer_step, 4);
pipeline.parrallelize(1);
pipeline.bind(len_var, 4);
pipeline.breakPipeline(0);
subpipe = pipeline.subpipeline(0, 1);
```

```
for x {
  f1();
  f2();
}
```

```
for x {
  f1();
  - - - Subpipeline
    for y{
      f2();
    }
}
```

Controlling Properties of the Fused Code

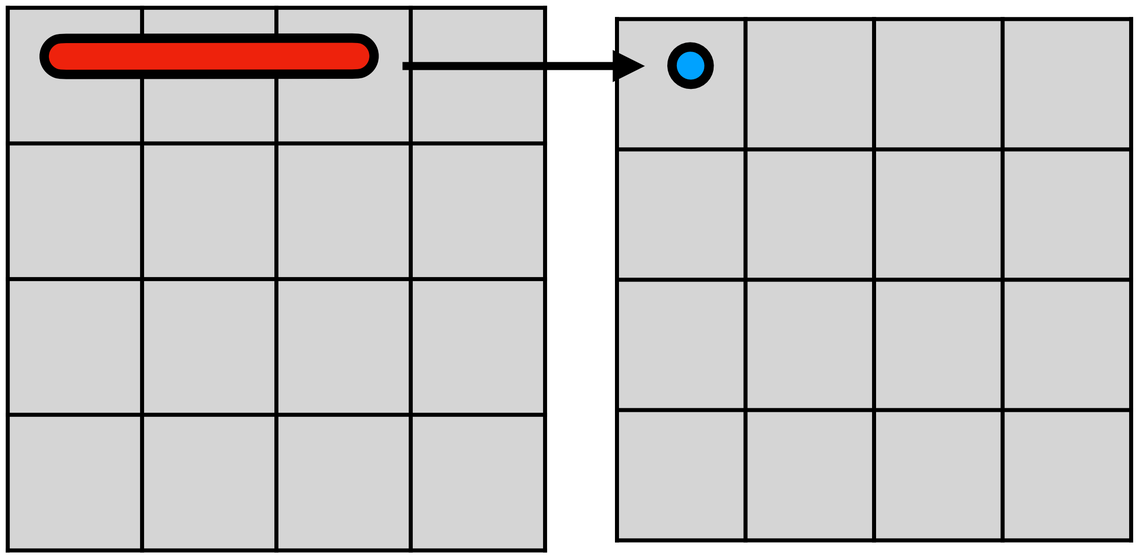
```
Pipeline pipeline({
      vadd(a, b, len, out_1),
      vadd(out_1, c, len, out_2),
  });

pipeline.constructPipeline();
pipeline = pipeline.finalize();
```

```
pipeline.split(1, outer, inner, outer_step, 4);
pipeline.parrallelize(1);
pipeline.bind(len_var, 4);
pipeline.breakPipeline(0);
subpipe = pipeline.subpipeline(0, 1);
```

```
for x {
  f1();
  f2();
}
```

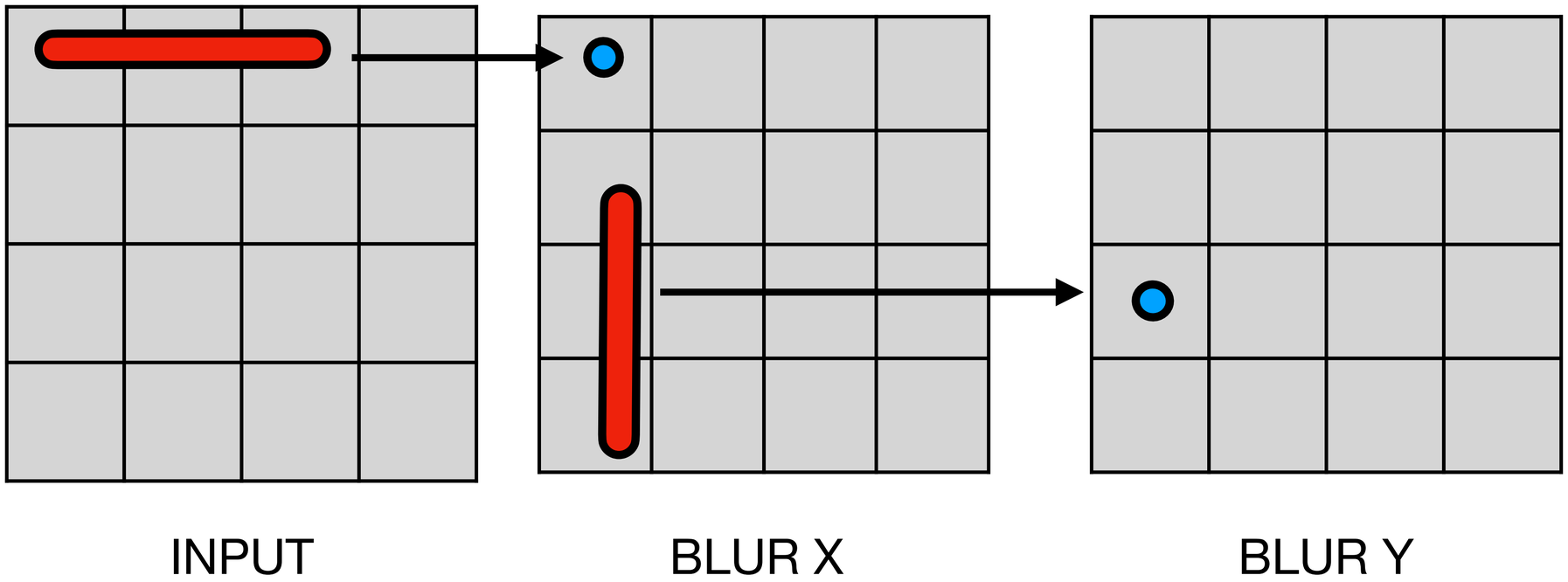
```
for x {
  f1();
  - - - Subpipeline
    for y{
      f2();
    }
}
```

Enough to model complicated dependencies

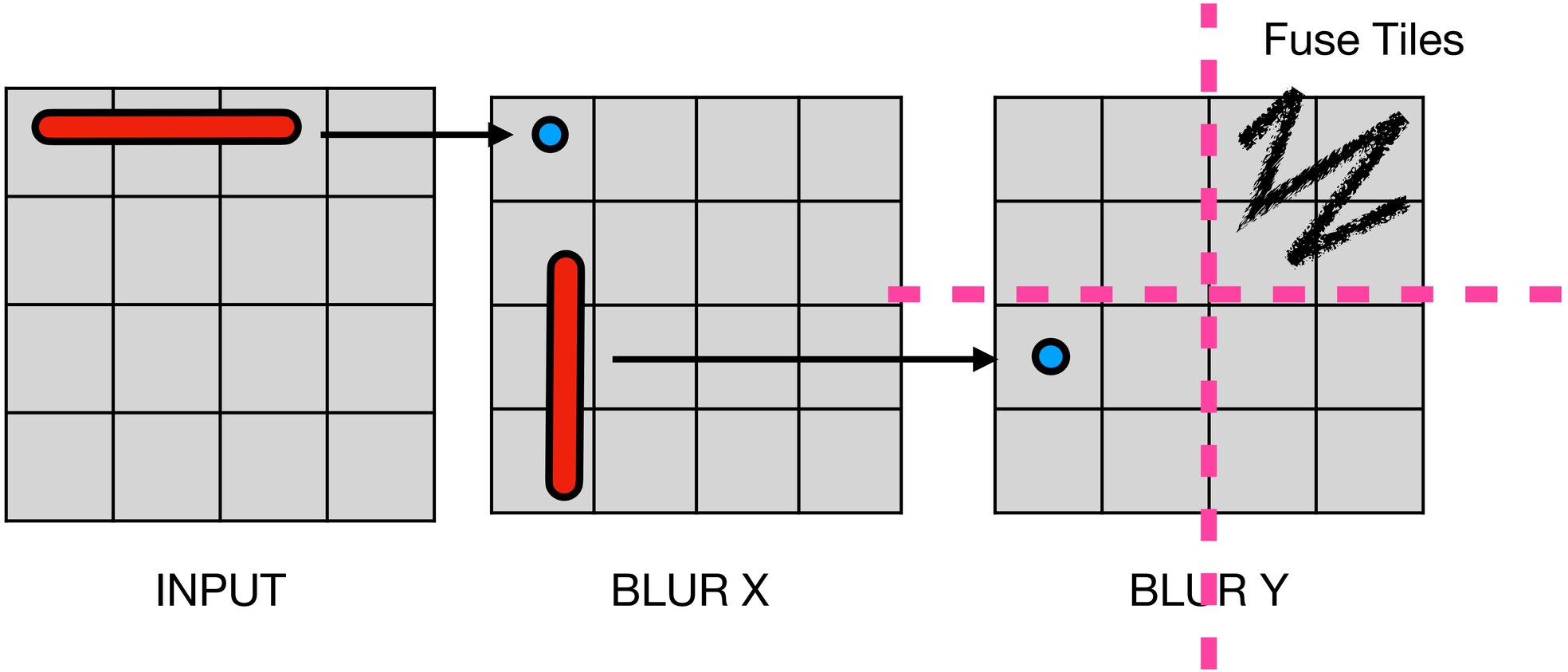


INPUT BLUR X

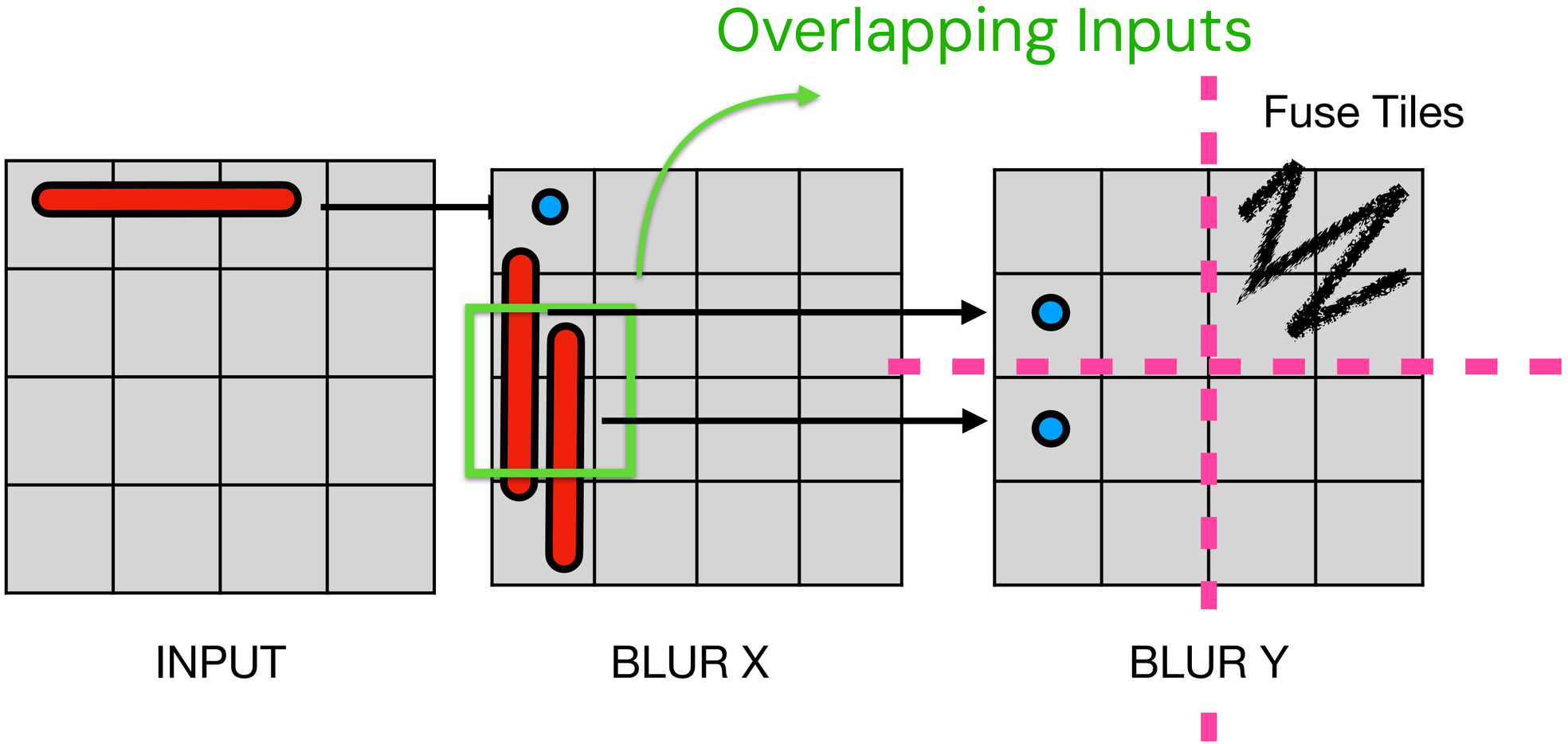
Enough to model complicated dependencies



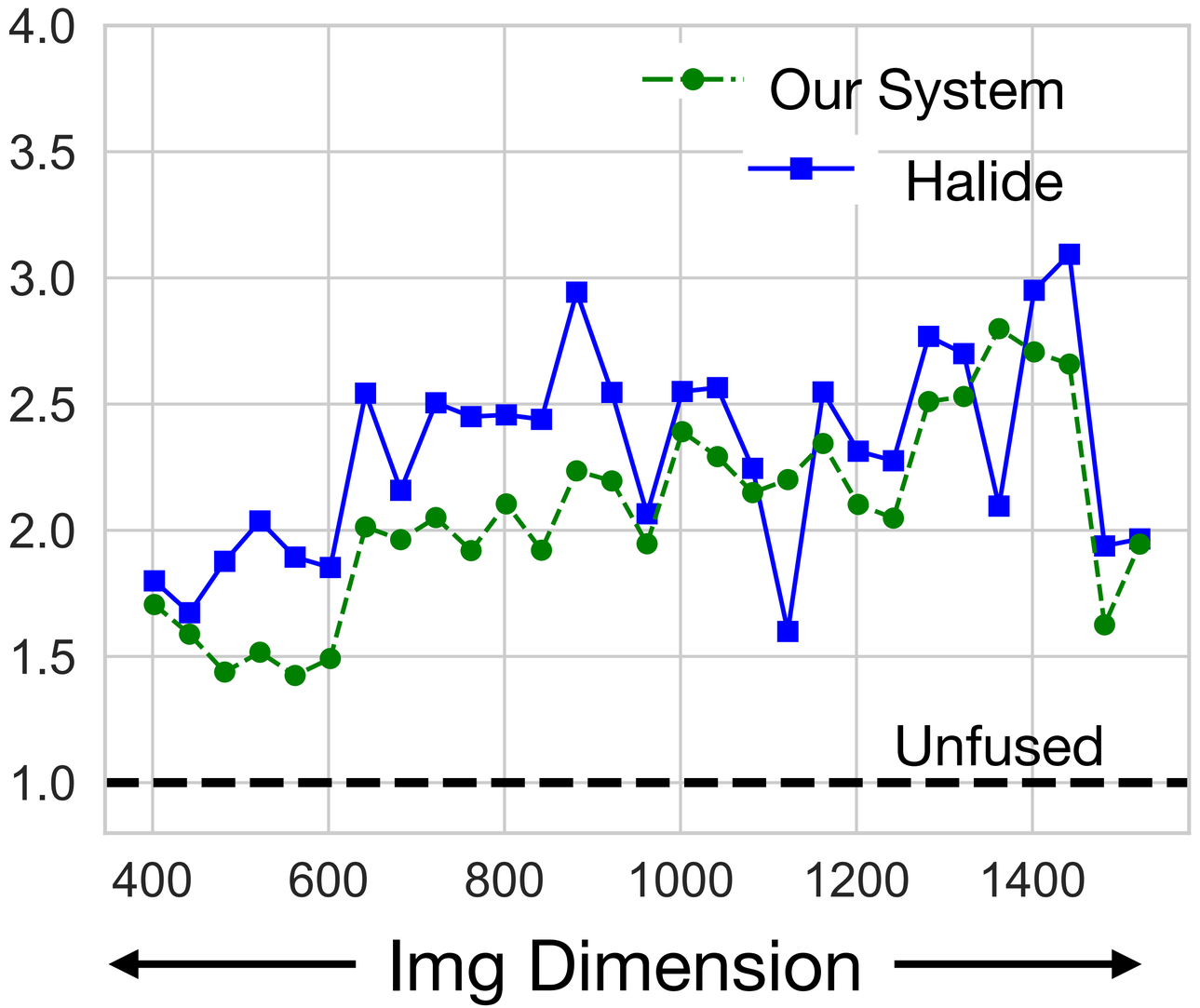
Enough to model complicated dependencies



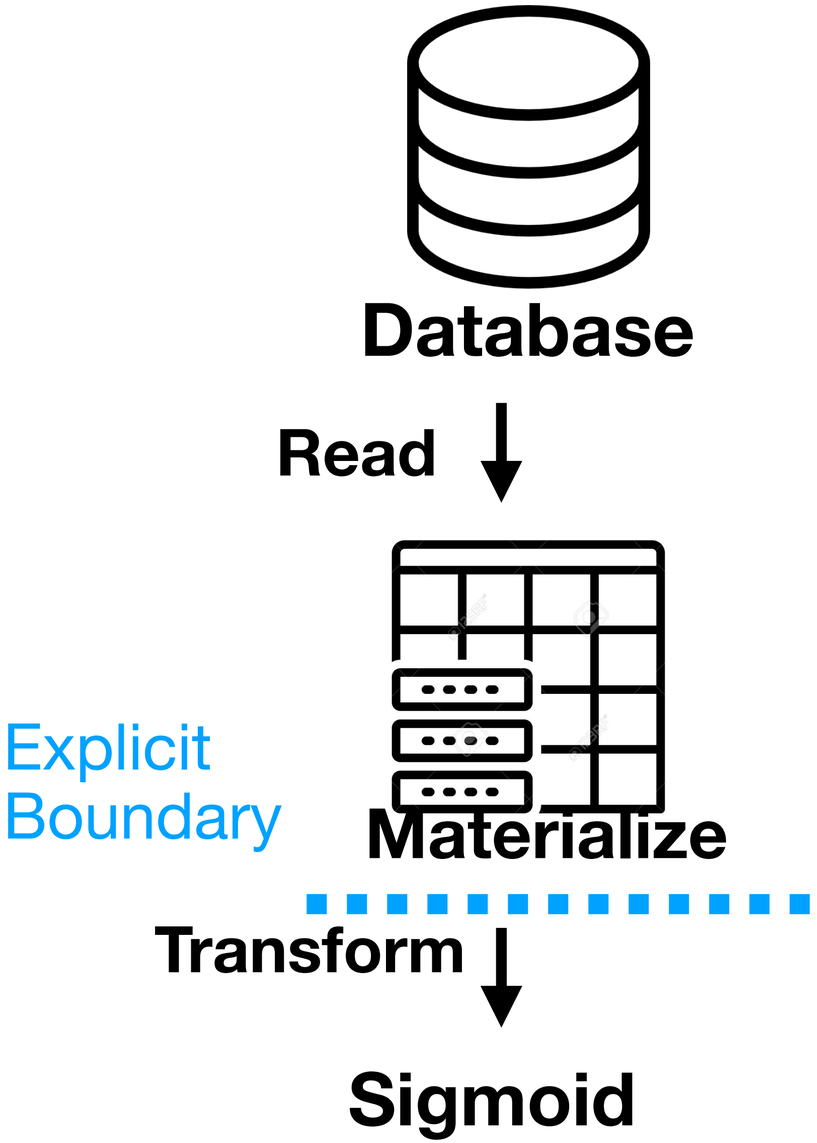
Enough to model complicated dependencies

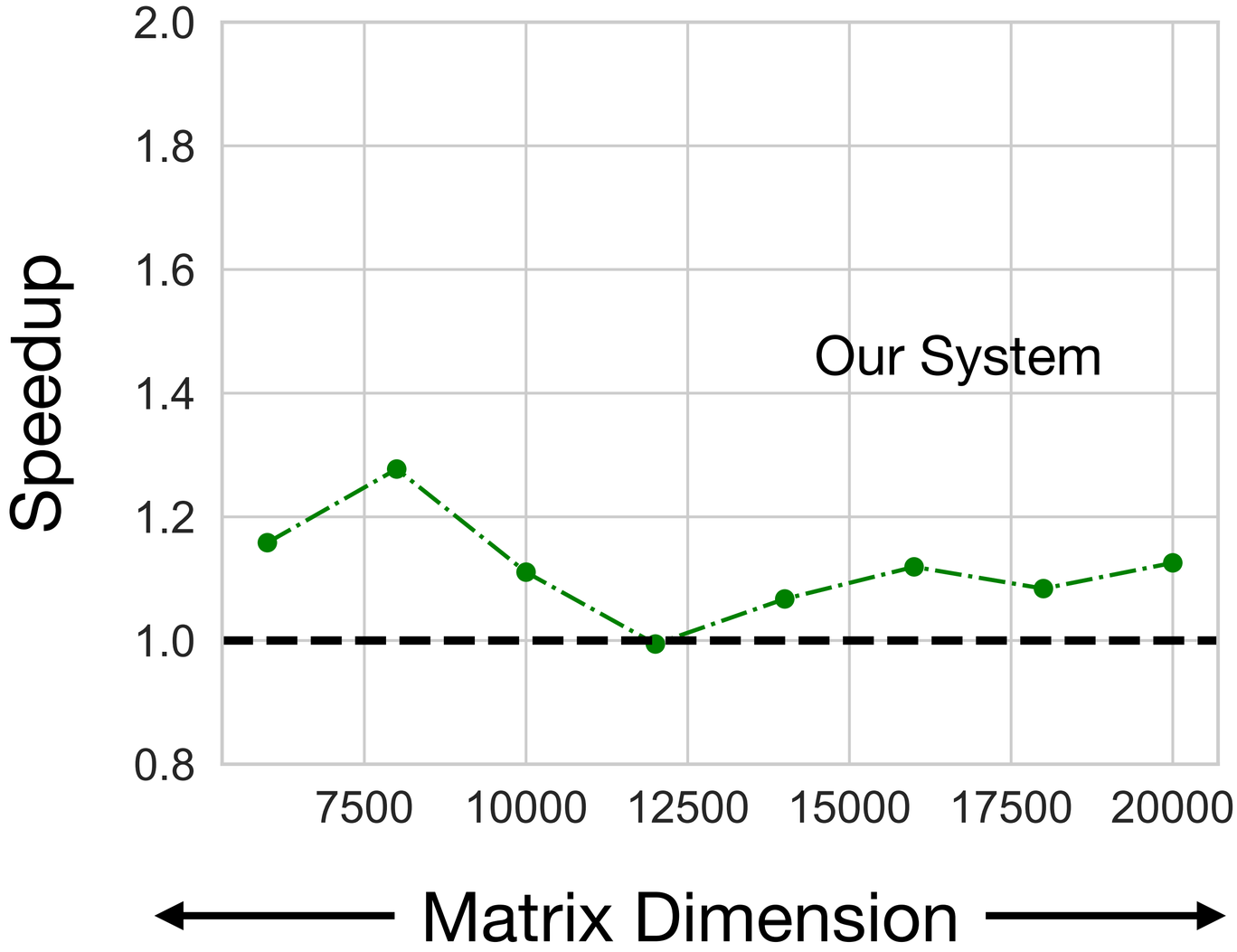


Enough to model complicated dependencies

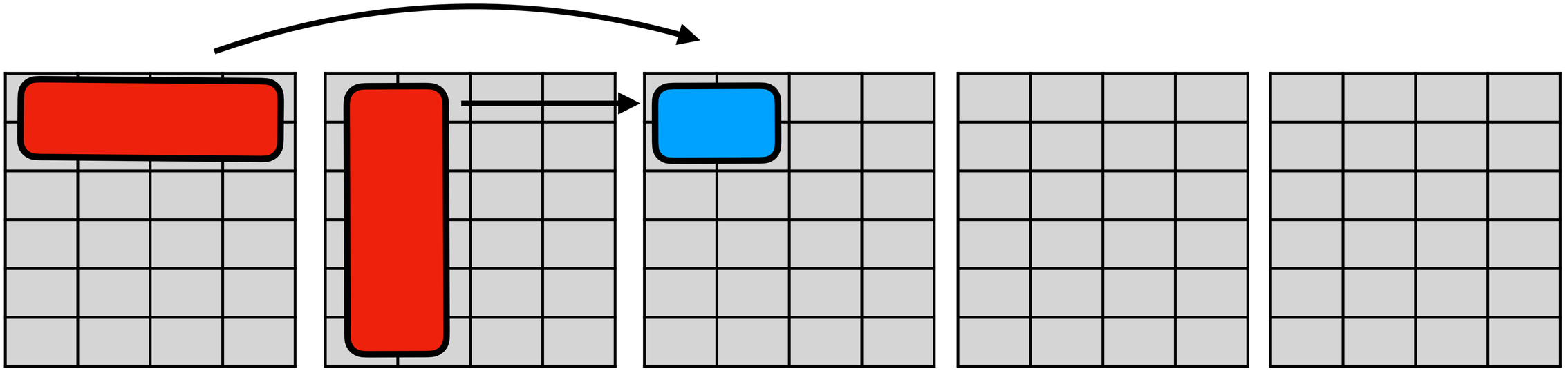


Fusion Across Domains

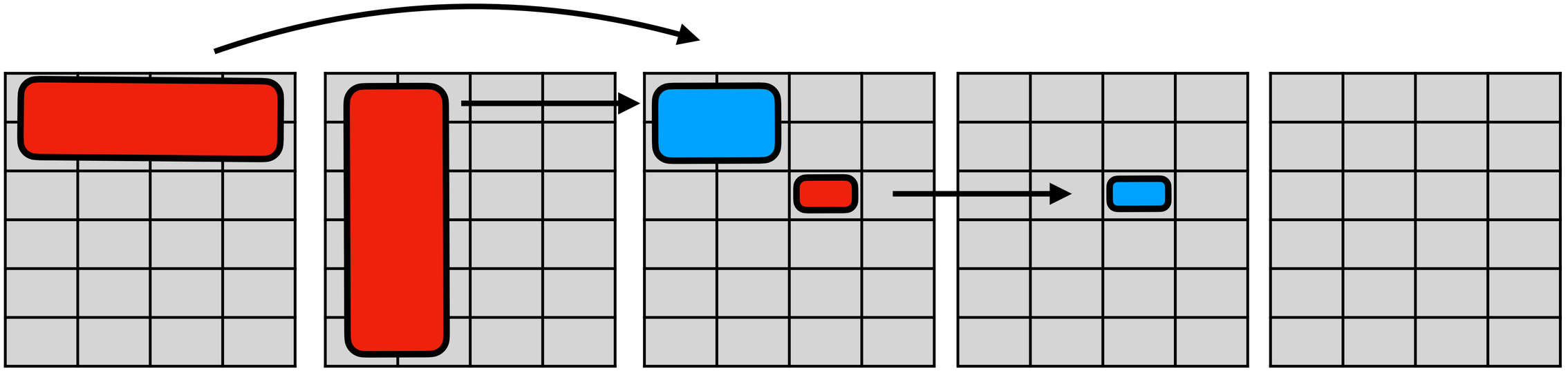




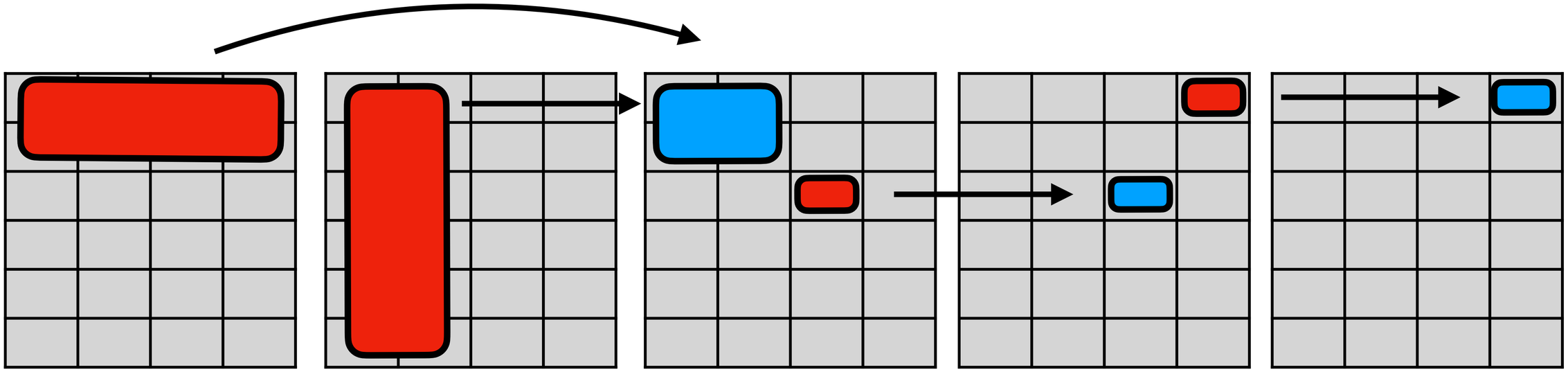
Data is all you need for fusion



Data is all you need for fusion



Data is all you need for fusion



Collaborators

Jonathan
Ragan-Kelley

Saman
Amarasinghe

Thank You!

Data is all you need for fusion