Helium

Lifting High-Performance Stencil Kernels from Stripped x86 Binaries to Halide DSL Code

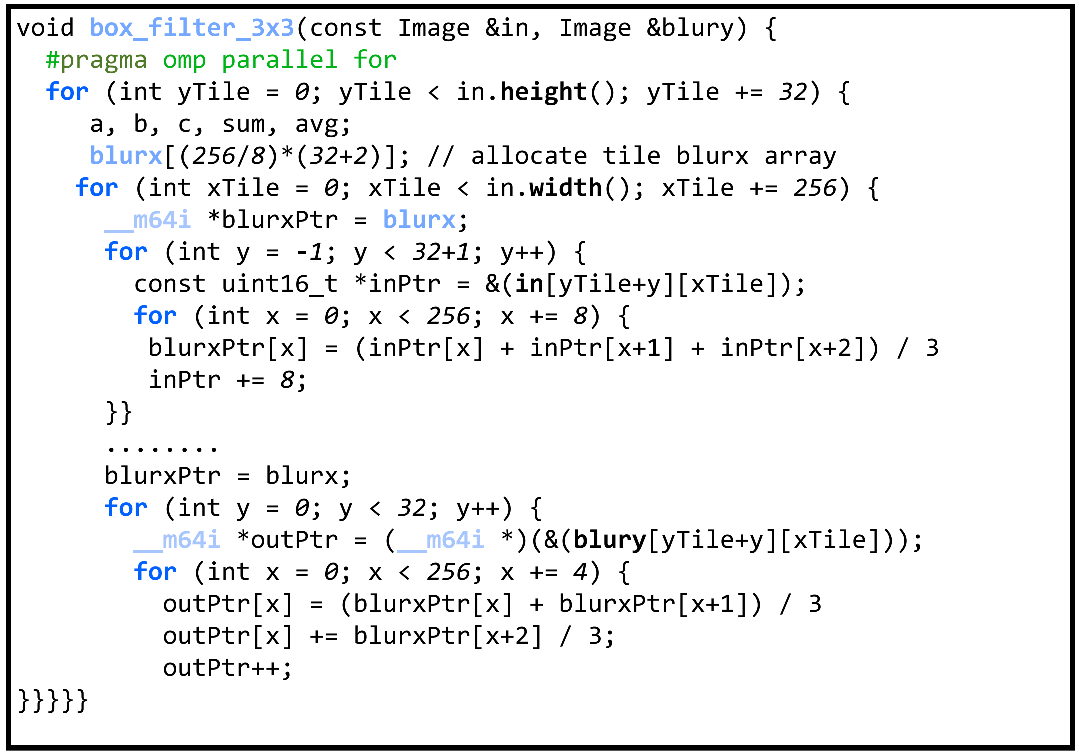
Charith Mendis[1]
Jeffrey Bosboom[1]
Kevin Wu[1]
Shoaib Kamil[1]
Jonathan Ragan-Kelley[2]
Sylvain Paris[4]
Qin Zhao[3]
Saman Amarasinghe[1]

[1] CSAIL

[4] Adobe

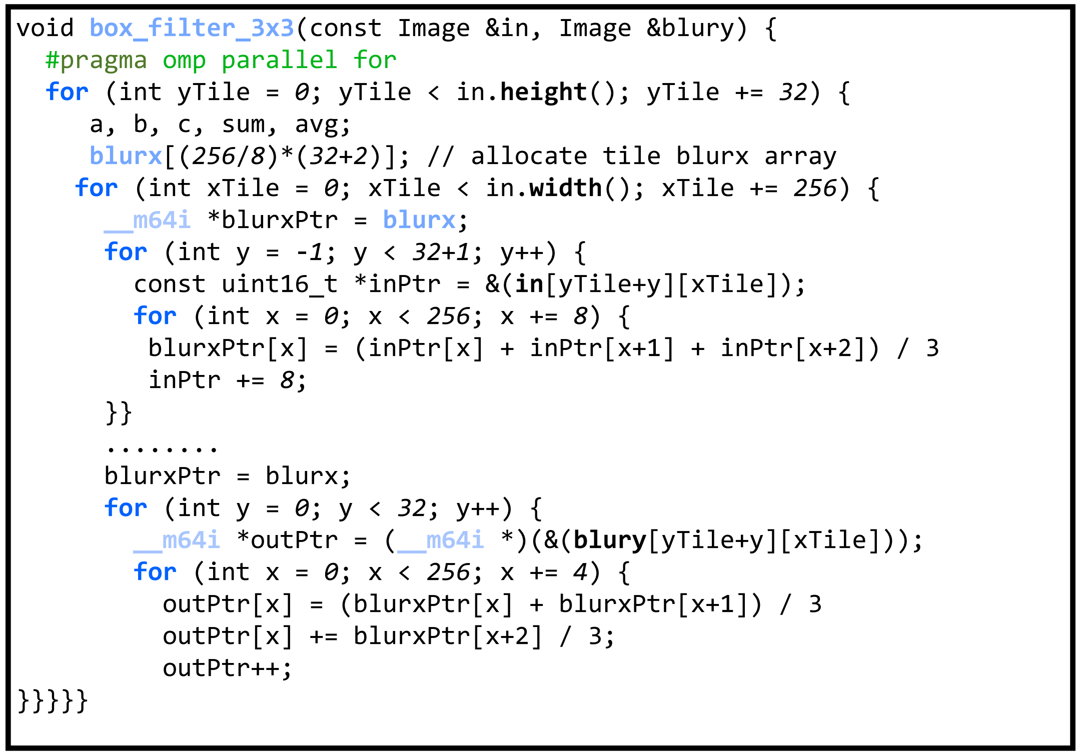
[2]

[3] Google

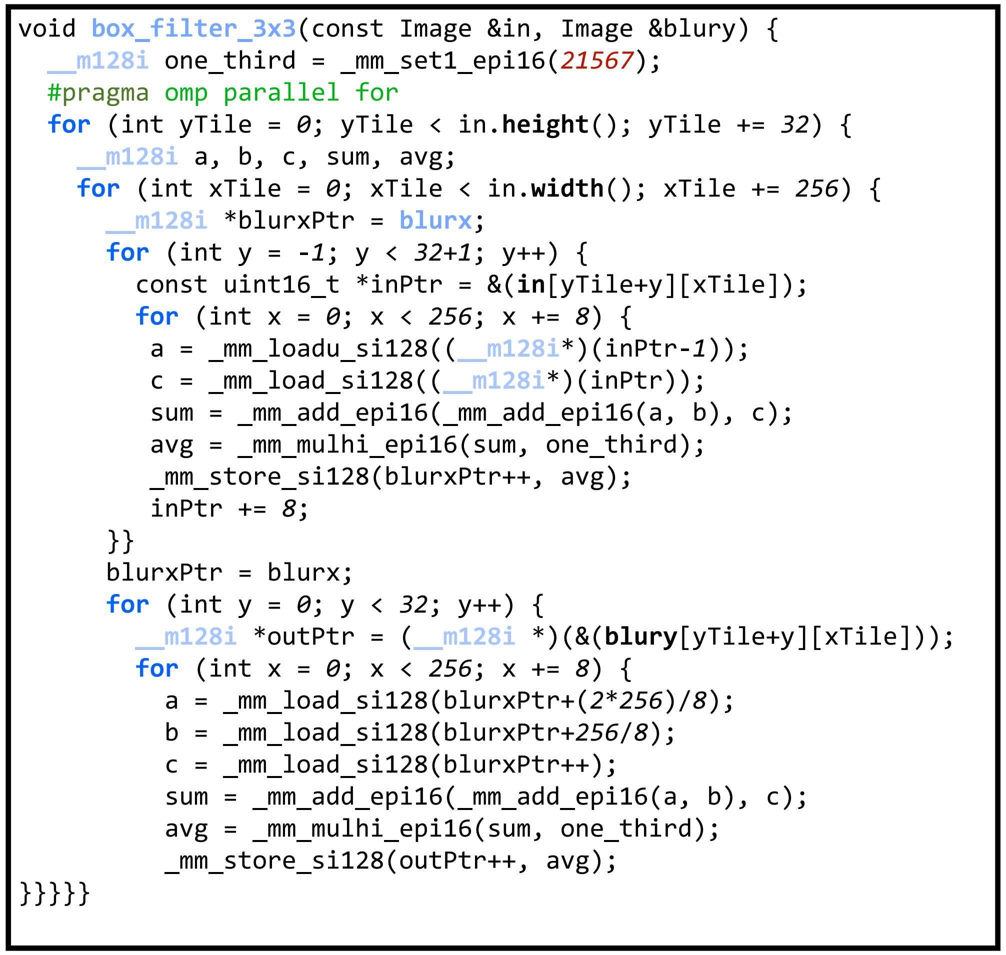


```
void box_filter_3x3(const Image &in, Image &blury) {
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
     a, b, c, sum, avg;
     blurx[(256/8)*(32+2)]; // allocate tile blurx array
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m64i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         blurxPtr[x] = (inPtr[x] + inPtr[x+1] + inPtr[x+2]) / 3
         inPtr += 8;
      }}
      ........
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m64i *outPtr = (__m64i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 4) {
          outPtr[x] = (blurxPtr[x] + blurxPtr[x+1]) / 3
          outPtr[x] += blurxPtr[x+2] / 3;
          outPtr++;
}}}}}
```

Older Architecture



```
void box_filter_3x3(const Image &in, Image &blury) {
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
     a, b, c, sum, avg;
     blurx[(256/8)*(32+2)]; // allocate tile blurx array
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m64i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         blurxPtr[x] = (inPtr[x] + inPtr[x+1] + inPtr[x+2]) / 3
         inPtr += 8;
      }}
      ........
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m64i *outPtr = (__m64i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 4) {
          outPtr[x] = (blurxPtr[x] + blurxPtr[x+1]) / 3
          outPtr[x] += blurxPtr[x+2] / 3;
          outPtr++;
}}}}}
```



```
void box_filter_3x3(const Image &in, Image &blury) {
  __m128i one_third = _mm_set1_epi16(21567);
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
    __m128i a, b, c, sum, avg;
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m128i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         a = _mm_loadu_si128((__m128i*)(inPtr-1));
         c = _mm_load_si128((__m128i*)(inPtr));
         sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
         avg = _mm_mulhi_epi16(sum, one_third);
         _mm_store_si128(blurxPtr++, avg);
         inPtr += 8;
      }}
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m128i *outPtr = (__m128i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 8) {
          a = _mm_load_si128(blurxPtr+(2*256)/8);
          b = _mm_load_si128(blurxPtr+256/8);
          c = _mm_load_si128(blurxPtr++);
          sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
          avg = _mm_mulhi_epi16(sum, one_third);
          _mm_store_si128(outPtr++, avg);
}}}}}
```

Older Architecture

Newer Architecture

```
void box_filter_3x3(const Image &in, Image &blury) {
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
     a, b, c, sum, avg;
     blurx[(256/8)*(32+2)]; // allocate tile blurx array
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m64i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         blurxPtr[x] = (inPtr[x] + inPtr[x+1] + inPtr[x+2]) / 3
         inPtr += 8;
      }}
      ........
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m64i *outPtr = (__m64i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 4) {
          outPtr[x] = (blurxPtr[x] + blurxPtr[x+1]) / 3
          outPtr[x] += blurxPtr[x+2] / 3;
          outPtr++;
}}}}}
```

Rewrite??

```
void box_filter_3x3(const Image &in, Image &blury) {
  __m128i one_third = _mm_set1_epi16(21567);
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
    __m128i a, b, c, sum, avg;
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m128i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         a = _mm_loadu_si128((__m128i*)(inPtr-1));
         c = _mm_load_si128((__m128i*)(inPtr));
         sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
         avg = _mm_mulhi_epi16(sum, one_third);
         _mm_store_si128(blurxPtr++, avg);
         inPtr += 8;
      }}
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m128i *outPtr = (__m128i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 8) {
          a = _mm_load_si128(blurxPtr+(2*256)/8);
          b = _mm_load_si128(blurxPtr+256/8);
          c = _mm_load_si128(blurxPtr++);
          sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
          avg = _mm_mulhi_epi16(sum, one_third);
          _mm_store_si128(outPtr++, avg);
}}}}}
```

Older Architecture

Newer Architecture

```
void blur(const Image &in, Image &blurred){

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

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

Simple Algorithm

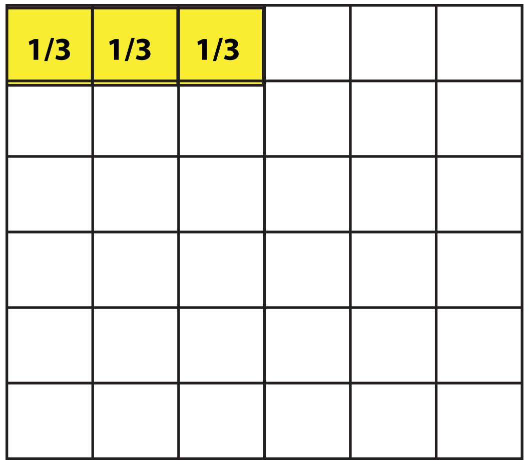
```
void box_filter_3x3(const Image &in, Image &blury) {
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
     a, b, c, sum, avg;
     blurx[(256/8)*(32+2)]; // allocate tile blurx array
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m64i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         blurxPtr[x] = (inPtr[x] + inPtr[x+1] + inPtr[x+2]) / 3
         inPtr += 8;
      }}
      ........
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m64i *outPtr = (__m64i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 4) {
          outPtr[x] = (blurxPtr[x] + blurxPtr[x+1]) / 3
          outPtr[x] += blurxPtr[x+2] / 3;
          outPtr++;
}}}}}
```

Rewrite??

```
void box_filter_3x3(const Image &in, Image &blury) {
  __m128i one_third = _mm_set1_epi16(21567);
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
    __m128i a, b, c, sum, avg;
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m128i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         a = _mm_loadu_si128((__m128i*)(inPtr-1));
         c = _mm_load_si128((__m128i*)(inPtr));
         sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
         avg = _mm_mulhi_epi16(sum, one_third);
         _mm_store_si128(blurxPtr++, avg);
         inPtr += 8;
      }}
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m128i *outPtr = (__m128i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 8) {
          a = _mm_load_si128(blurxPtr+(2*256)/8);
          b = _mm_load_si128(blurxPtr+256/8);
          c = _mm_load_si128(blurxPtr++);
          sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
          avg = _mm_mulhi_epi16(sum, one_third);
          _mm_store_si128(outPtr++, avg);
}}}}}
```

Older Architecture

Newer Architecture



```
void blur(const Image &in, Image &blurred){

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

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

Simple Algorithm

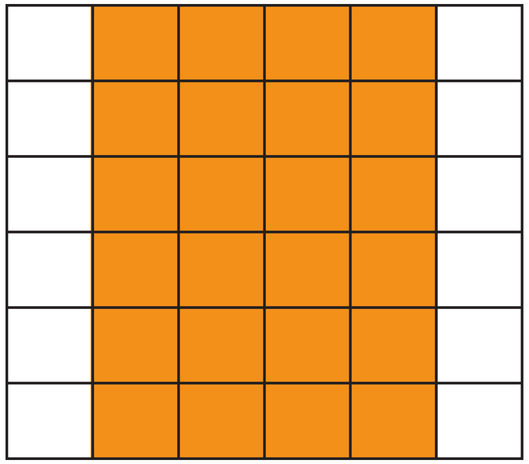
```
void box_filter_3x3(const Image &in, Image &blury) {
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
     a, b, c, sum, avg;
     blurx[(256/8)*(32+2)]; // allocate tile blurx array
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m64i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         blurxPtr[x] = (inPtr[x] + inPtr[x+1] + inPtr[x+2]) / 3
         inPtr += 8;
      }}
      ........
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m64i *outPtr = (__m64i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 4) {
          outPtr[x] = (blurxPtr[x] + blurxPtr[x+1]) / 3
          outPtr[x] += blurxPtr[x+2] / 3;
          outPtr++;
}}}}}
```

Rewrite??

```
void box_filter_3x3(const Image &in, Image &blury) {
  __m128i one_third = _mm_set1_epi16(21567);
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
    __m128i a, b, c, sum, avg;
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m128i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         a = _mm_loadu_si128((__m128i*)(inPtr-1));
         c = _mm_load_si128((__m128i*)(inPtr));
         sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
         avg = _mm_mulhi_epi16(sum, one_third);
         _mm_store_si128(blurxPtr++, avg);
         inPtr += 8;
      }}
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m128i *outPtr = (__m128i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 8) {
          a = _mm_load_si128(blurxPtr+(2*256)/8);
          b = _mm_load_si128(blurxPtr+256/8);
          c = _mm_load_si128(blurxPtr++);
          sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
          avg = _mm_mulhi_epi16(sum, one_third);
          _mm_store_si128(outPtr++, avg);
}}}}}
```

Older Architecture

Newer Architecture



```
void blur(const Image &in, Image &blurred){

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

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

Simple Algorithm

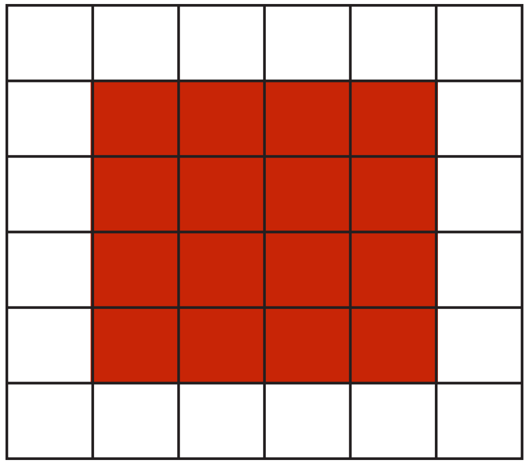
```
void box_filter_3x3(const Image &in, Image &blury) {
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
     a, b, c, sum, avg;
     blurx[(256/8)*(32+2)]; // allocate tile blurx array
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m64i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         blurxPtr[x] = (inPtr[x] + inPtr[x+1] + inPtr[x+2]) / 3
         inPtr += 8;
      }}
      ........
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m64i *outPtr = (__m64i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 4) {
          outPtr[x] = (blurxPtr[x] + blurxPtr[x+1]) / 3
          outPtr[x] += blurxPtr[x+2] / 3;
          outPtr++;
}}}}}
```

Rewrite??

```
void box_filter_3x3(const Image &in, Image &blury) {
  __m128i one_third = _mm_set1_epi16(21567);
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
    __m128i a, b, c, sum, avg;
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m128i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         a = _mm_loadu_si128((__m128i*)(inPtr-1));
         c = _mm_load_si128((__m128i*)(inPtr));
         sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
         avg = _mm_mulhi_epi16(sum, one_third);
         _mm_store_si128(blurxPtr++, avg);
         inPtr += 8;
      }}
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m128i *outPtr = (__m128i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 8) {
          a = _mm_load_si128(blurxPtr+(2*256)/8);
          b = _mm_load_si128(blurxPtr+256/8);
          c = _mm_load_si128(blurxPtr++);
          sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
          avg = _mm_mulhi_epi16(sum, one_third);
          _mm_store_si128(outPtr++, avg);
}}}}}
```

Older Architecture

Newer Architecture



```
void blur(const Image &in, Image &blurred){

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

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

Simple Algorithm

Rewrite??

```
void box_filter_3x3(const Image &in, Image &blury) {
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
     a, b, c, sum, avg;
     blurx[(256/8)*(32+2)]; // allocate tile blurx array
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m64i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         blurxPtr[x] = (inPtr[x] + inPtr[x+1] + inPtr[x+2]) / 3
         inPtr += 8;
      }}
      ........
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m64i *outPtr = (__m64i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 4) {
          outPtr[x] = (blurxPtr[x] + blurxPtr[x+1]) / 3
          outPtr[x] += blurxPtr[x+2] / 3;
          outPtr++;
}}}}}
```

```
void box_filter_3x3(const Image &in, Image &blury) {
  __m128i one_third = _mm_set1_epi16(21567);
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
    __m128i a, b, c, sum, avg;
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m128i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         a = _mm_loadu_si128((__m128i*)(inPtr-1));
         c = _mm_load_si128((__m128i*)(inPtr));
         sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
         avg = _mm_mulhi_epi16(sum, one_third);
         _mm_store_si128(blurxPtr++, avg);
         inPtr += 8;
      }}
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m128i *outPtr = (__m128i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 8) {
          a = _mm_load_si128(blurxPtr+(2*256)/8);
          b = _mm_load_si128(blurxPtr+256/8);
          c = _mm_load_si128(blurxPtr++);
          sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
          avg = _mm_mulhi_epi16(sum, one_third);
          _mm_store_si128(outPtr++, avg);
}}}}}
```

Older Architecture

Newer Architecture

Simple Algorithm

```
void blur(const Image &in, Image &blurred){

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

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

Halide

```
ImageParam input(UInt(8), 2);

Func blur_x("blur_x"), blur_y("blur_y");
Var x("x"), y("y"), xi("xi"), yi("yi"), xo("xo"), yo("yo");
```

```
// The algorithm
blur_x(x, y) = (input(x, y) + input(x + 1, y) + input(x + 2, y)) / 3;
blur_y(x, y) = (blur_x(x, y) + blur_x(x, y + 1) + blur_x(x, y + 2)) / 3;
```

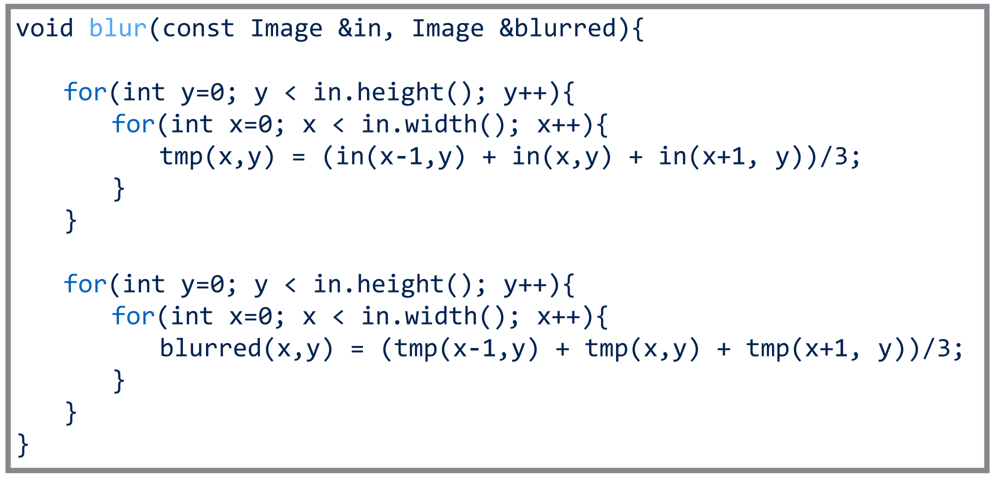
Algorithm

```
// How to schedule it
blur_y.split(y, y, yi, 8).parallel(y).vectorize(x, 8);
blur_x.store_at(blur_y, y).compute_at(blur_y, yi).vectorize(x, 8);
```

Schedule

[Ragan-Kelley et al PLDI'13]

Simple Algorithm

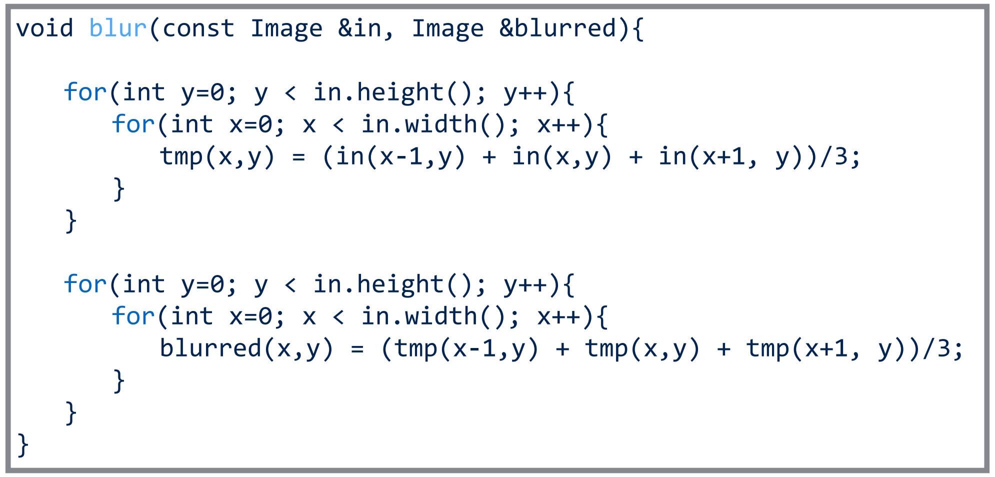


```
void blur(const Image &in, Image &blurred){

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

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

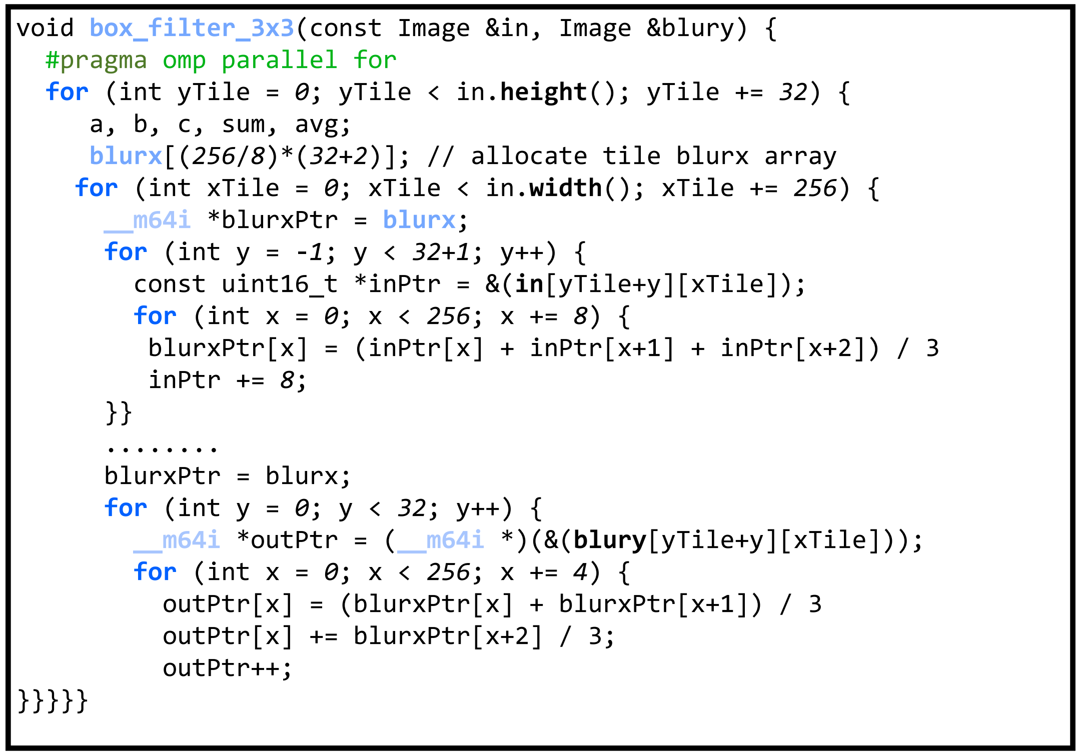
Halide

```
ImageParam input(UInt(8), 2);

Func blur_x("blur_x"), blur_y("blur_y");
Var x("x"), y("y"), xi("xi"), yi("yi"), xo("xo"), yo("yo");


// The algorithm
blur_x(x, y) = (input(x, y) + input(x + 1, y) + input(x + 2, y)) / 3;
blur_y(x, y) = (blur_x(x, y) + blur_x(x, y + 1) + blur_x(x, y + 2)) / 3;


// How to schedule it
blur_y.split(y, y, yi, 8).parallel(y).vectorize(x, 8);
blur_x.store_at(blur_y, y).compute_at(blur_y, yi).vectorize(x, 8);
```

Simple Algorithm

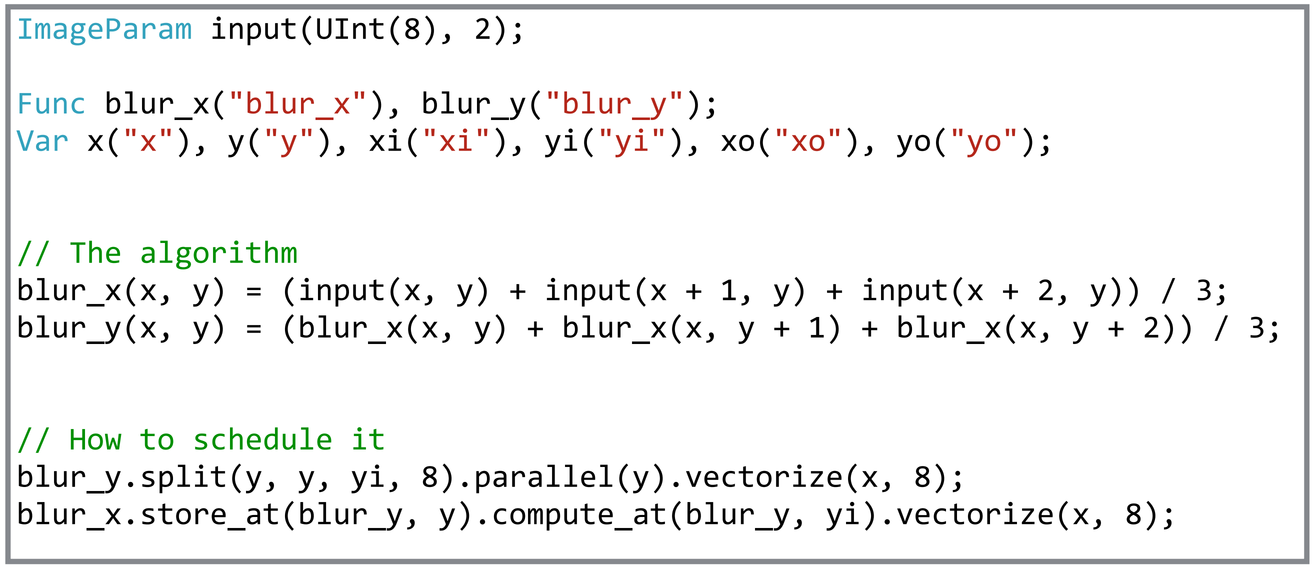


```
void blur(const Image &in, Image &blurred){

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

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

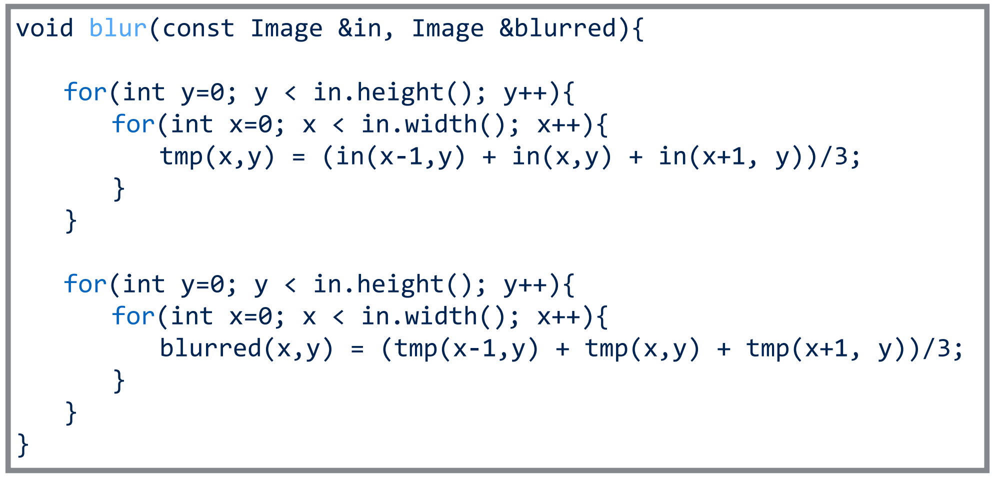


```
void box_filter_3x3(const Image &in, Image &blury) {
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
     a, b, c, sum, avg;
     blurx[(256/8)*(32+2)]; // allocate tile blurx array
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m64i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         blurxPtr[x] = (inPtr[x] + inPtr[x+1] + inPtr[x+2]) / 3
         inPtr += 8;
      }}
      ........
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m64i *outPtr = (__m64i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 4) {
          outPtr[x] = (blurxPtr[x] + blurxPtr[x+1]) / 3
          outPtr[x] += blurxPtr[x+2] / 3;
          outPtr++;
}}}}}
```

Halide

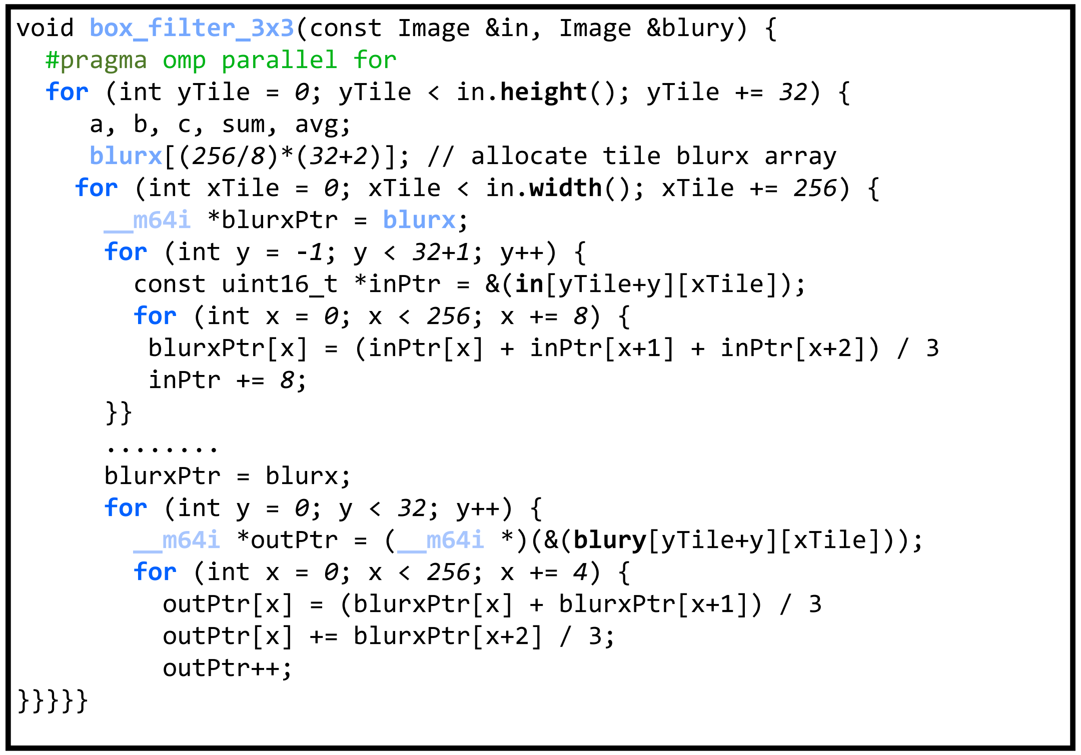


```
ImageParam input(UInt(8), 2);

Func blur_x("blur_x"), blur_y("blur_y");
Var x("x"), y("y"), xi("xi"), yi("yi"), xo("xo"), yo("yo");


// The algorithm
blur_x(x, y) = (input(x, y) + input(x + 1, y) + input(x + 2, y)) / 3;
blur_y(x, y) = (blur_x(x, y) + blur_x(x, y + 1) + blur_x(x, y + 2)) / 3;


// How to schedule it
blur_y.split(y, y, yi, 8).parallel(y).vectorize(x, 8);
blur_x.store_at(blur_y, y).compute_at(blur_y, yi).vectorize(x, 8);
```



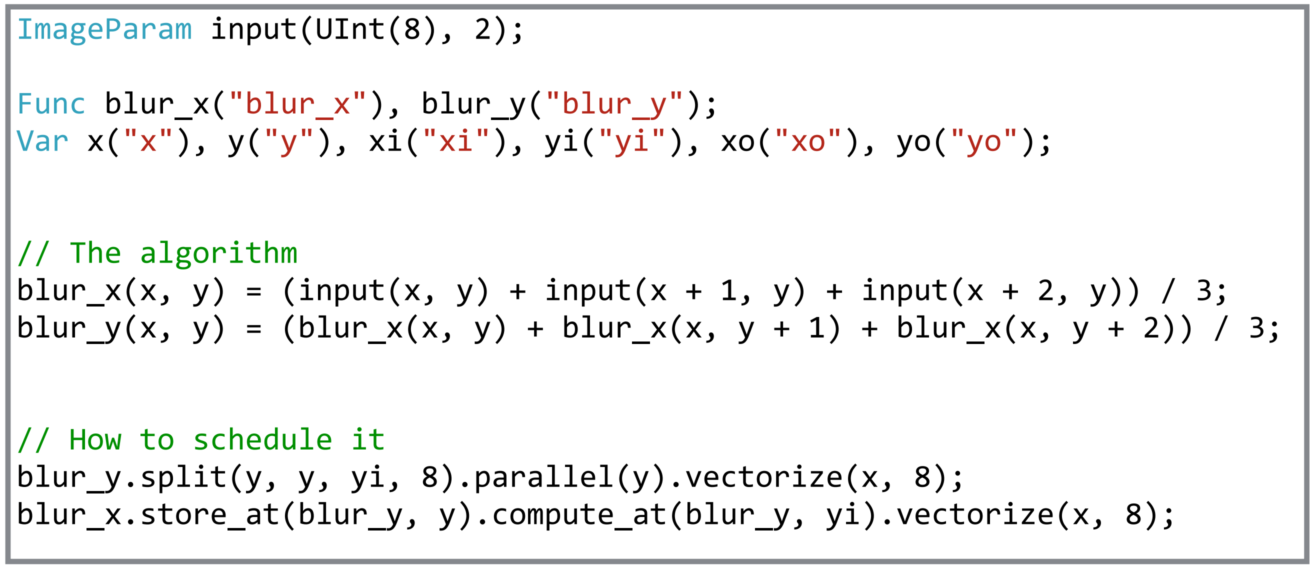
```
void blur(const Image &in, Image &blurred){

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

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

Simple Algorithm



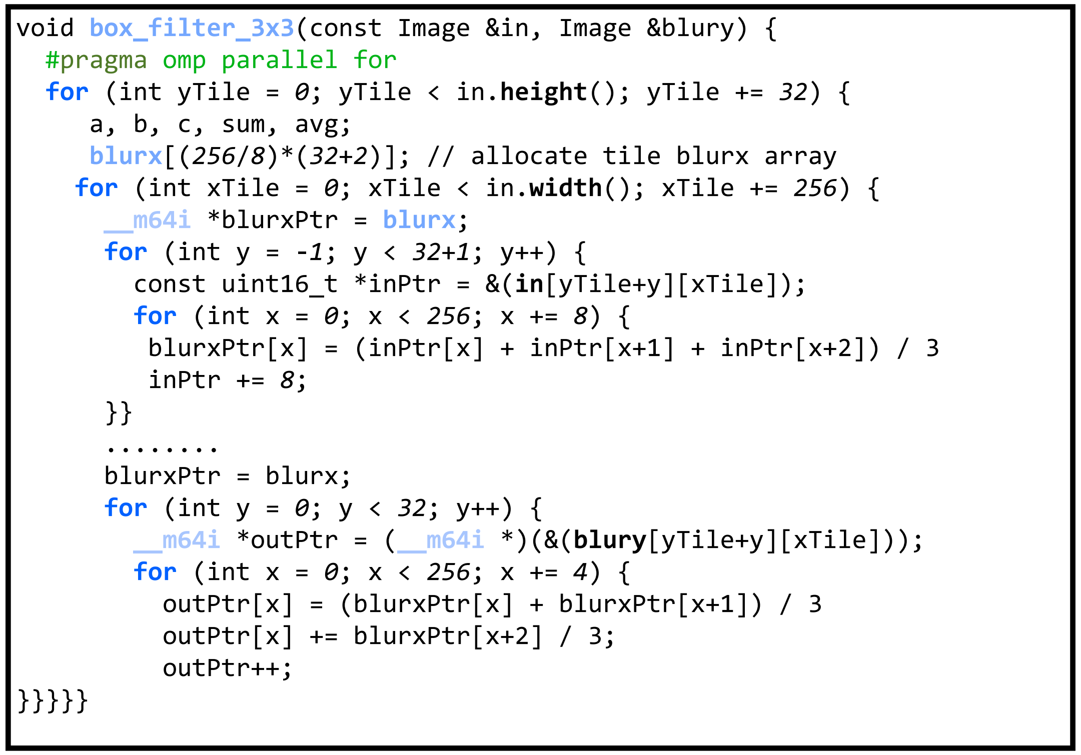
```
void box_filter_3x3(const Image &in, Image &blury) {
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
     a, b, c, sum, avg;
     blurx[(256/8)*(32+2)]; // allocate tile blurx array
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m64i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         blurxPtr[x] = (inPtr[x] + inPtr[x+1] + inPtr[x+2]) / 3
         inPtr += 8;
      }}
      ........
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m64i *outPtr = (__m64i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 4) {
          outPtr[x] = (blurxPtr[x] + blurxPtr[x+1]) / 3
          outPtr[x] += blurxPtr[x+2] / 3;
          outPtr++;
}}}}}
```

Halide

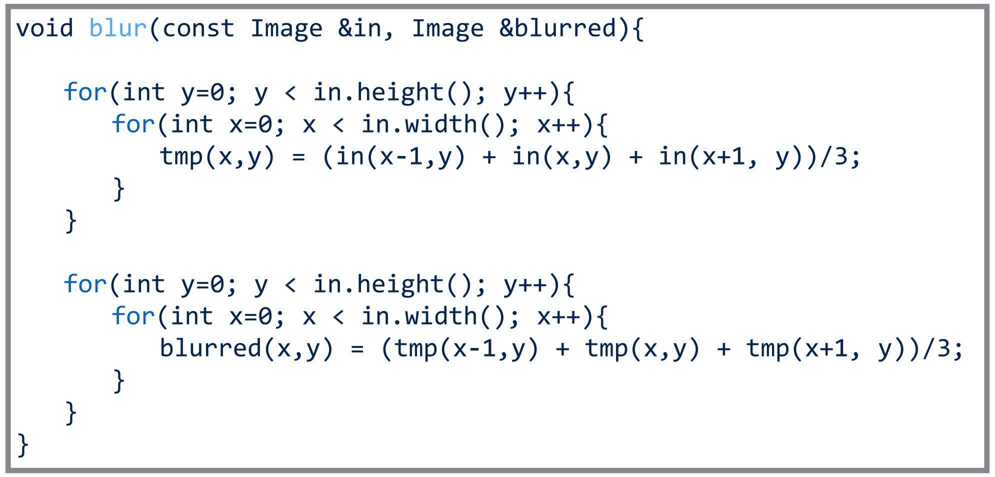


```
ImageParam input(UInt(8), 2);

Func blur_x("blur_x"), blur_y("blur_y");
Var x("x"), y("y"), xi("xi"), yi("yi"), xo("xo"), yo("yo");


// The algorithm
blur_x(x, y) = (input(x, y) + input(x + 1, y) + input(x + 2, y)) / 3;
blur_y(x, y) = (blur_x(x, y) + blur_x(x, y + 1) + blur_x(x, y + 2)) / 3;


// How to schedule it
blur_y.split(y, y, yi, 8).parallel(y).vectorize(x, 8);
blur_x.store_at(blur_y, y).compute_at(blur_y, yi).vectorize(x, 8);
```

Helium



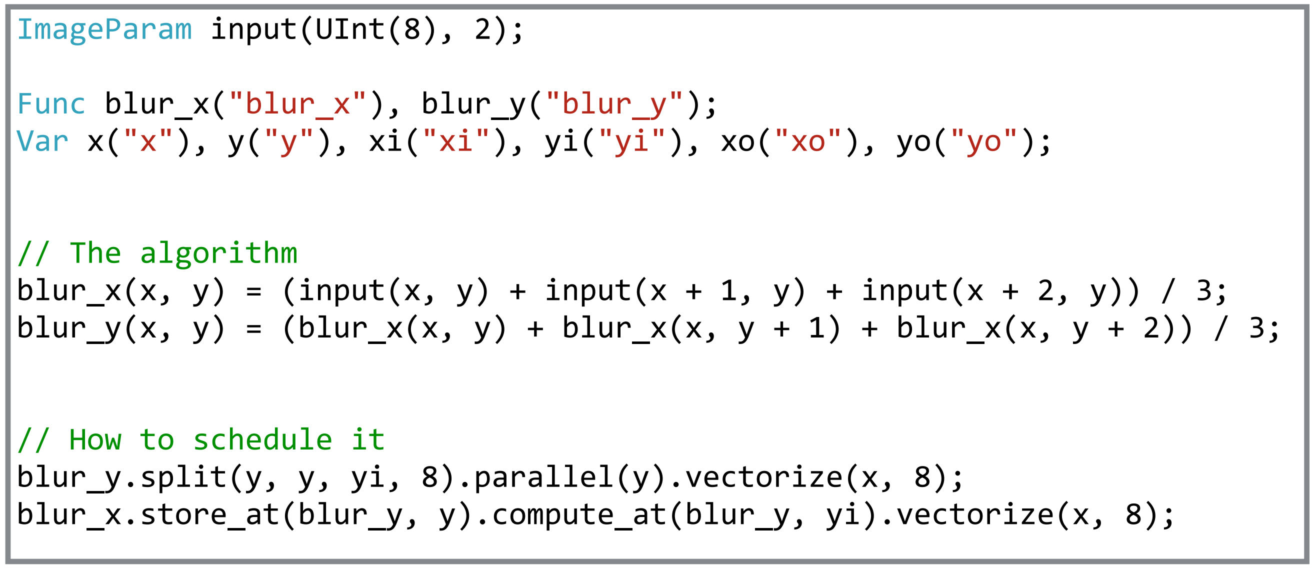
```
void box_filter_3x3(const Image &in, Image &blury) {
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
     a, b, c, sum, avg;
     blurx[(256/8)*(32+2)]; // allocate tile blurx array
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m64i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         blurxPtr[x] = (inPtr[x] + inPtr[x+1] + inPtr[x+2]) / 3
         inPtr += 8;
      }}
      ........
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m64i *outPtr = (__m64i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 4) {
          outPtr[x] = (blurxPtr[x] + blurxPtr[x+1]) / 3
          outPtr[x] += blurxPtr[x+2] / 3;
          outPtr++;
}}}}}
```



```
void blur(const Image &in, Image &blurred){

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

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

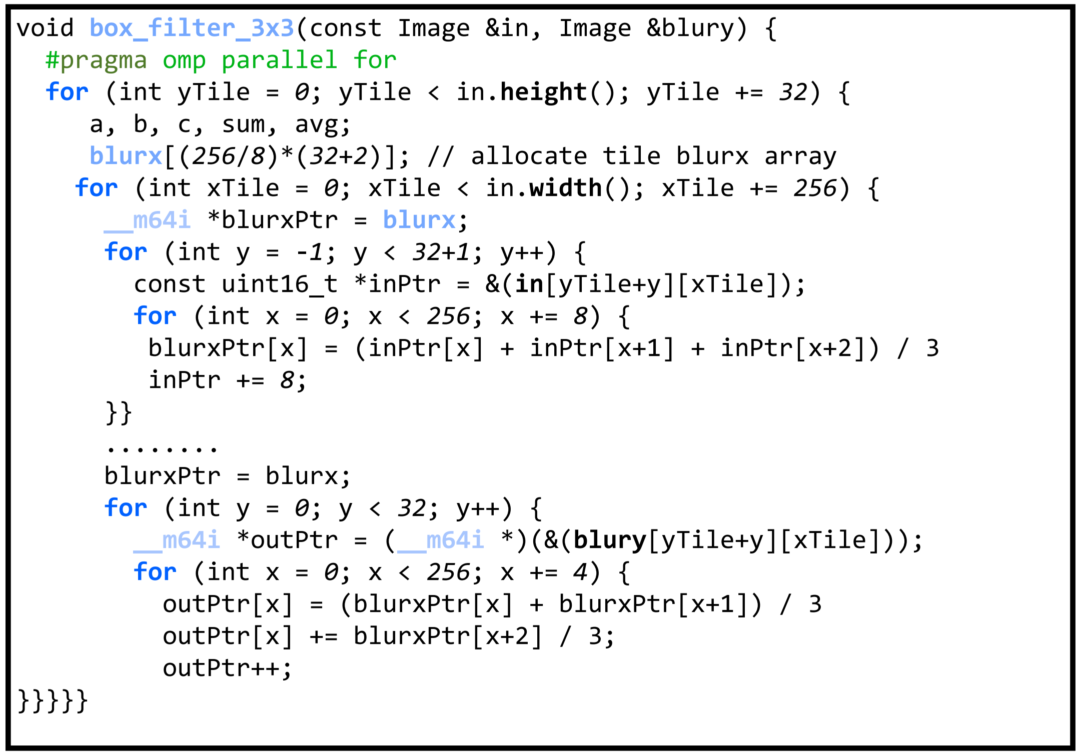
Simple Algorithm

Halide



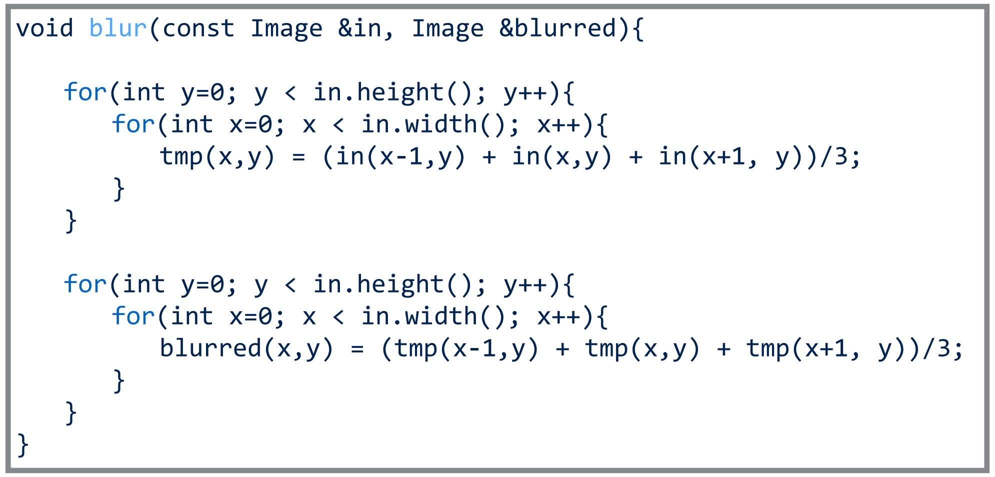
```
ImageParam input(UInt(8), 2);

Func blur_x("blur_x"), blur_y("blur_y");
Var x("x"), y("y"), xi("xi"), yi("yi"), xo("xo"), yo("yo");


// The algorithm
blur_x(x, y) = (input(x, y) + input(x + 1, y) + input(x + 2, y)) / 3;
blur_y(x, y) = (blur_x(x, y) + blur_x(x, y + 1) + blur_x(x, y + 2)) / 3;


// How to schedule it
blur_y.split(y, y, yi, 8).parallel(y).vectorize(x, 8);
blur_x.store_at(blur_y, y).compute_at(blur_y, yi).vectorize(x, 8);
```

Helium



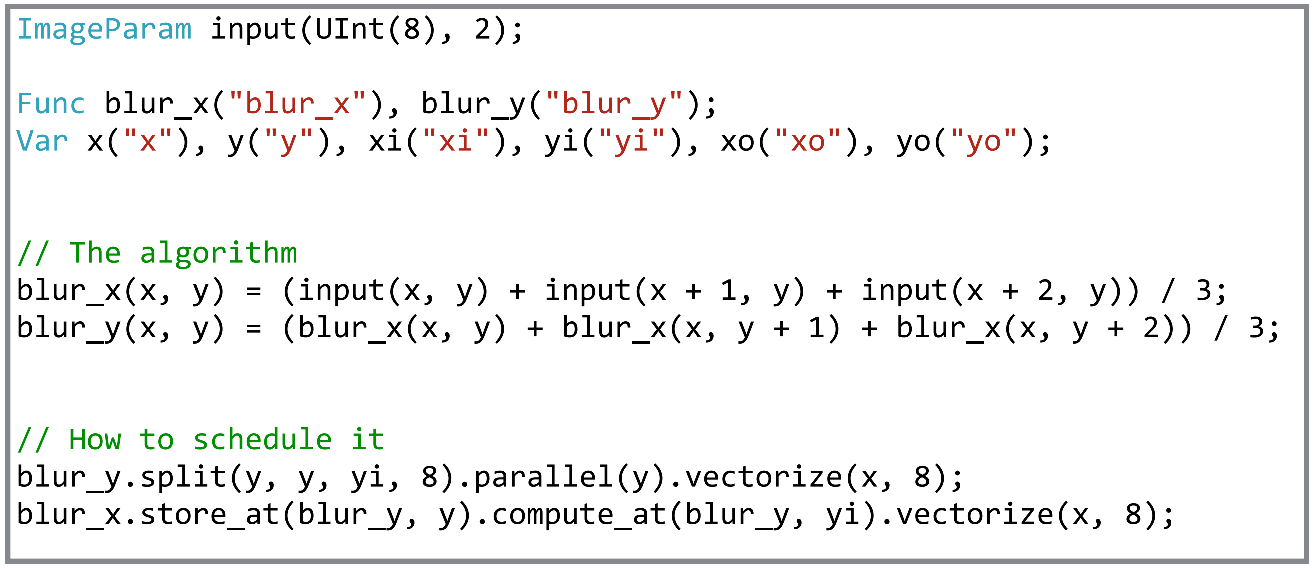
```
void box_filter_3x3(const Image &in, Image &blury) {
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
     a, b, c, sum, avg;
     blurx[(256/8)*(32+2)]; // allocate tile blurx array
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m64i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         blurxPtr[x] = (inPtr[x] + inPtr[x+1] + inPtr[x+2]) / 3
         inPtr += 8;
      }}
      ........
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m64i *outPtr = (__m64i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 4) {
          outPtr[x] = (blurxPtr[x] + blurxPtr[x+1]) / 3
          outPtr[x] += blurxPtr[x+2] / 3;
          outPtr++;
}}}}}
```

Simple Algorithm



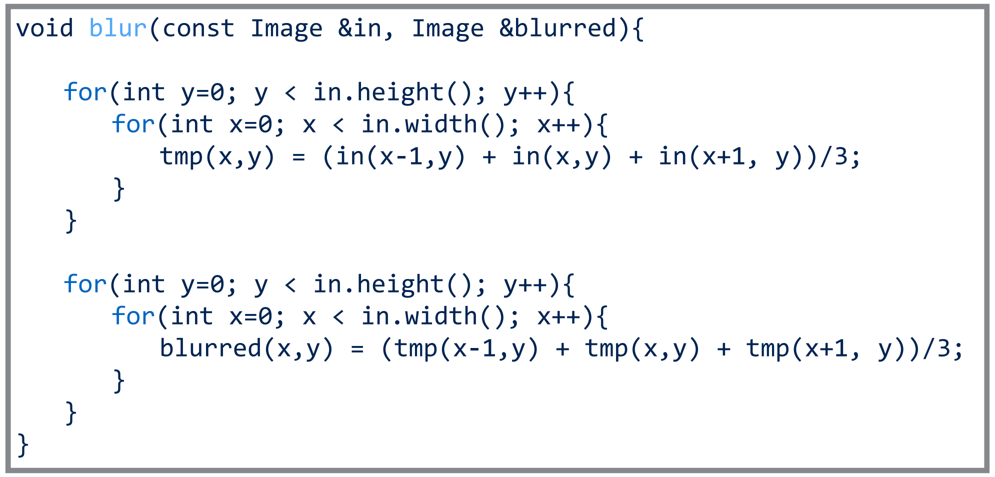
```
void blur(const Image &in, Image &blurred){

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

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

Halide



```
ImageParam input(UInt(8), 2);

Func blur_x("blur_x"), blur_y("blur_y");
Var x("x"), y("y"), xi("xi"), yi("yi"), xo("xo"), yo("yo");


// The algorithm
blur_x(x, y) = (input(x, y) + input(x + 1, y) + input(x + 2, y)) / 3;
blur_y(x, y) = (blur_x(x, y) + blur_x(x, y + 1) + blur_x(x, y + 2)) / 3;


// How to schedule it
blur_y.split(y, y, yi, 8).parallel(y).vectorize(x, 8);
blur_x.store_at(blur_y, y).compute_at(blur_y, yi).vectorize(x, 8);
```

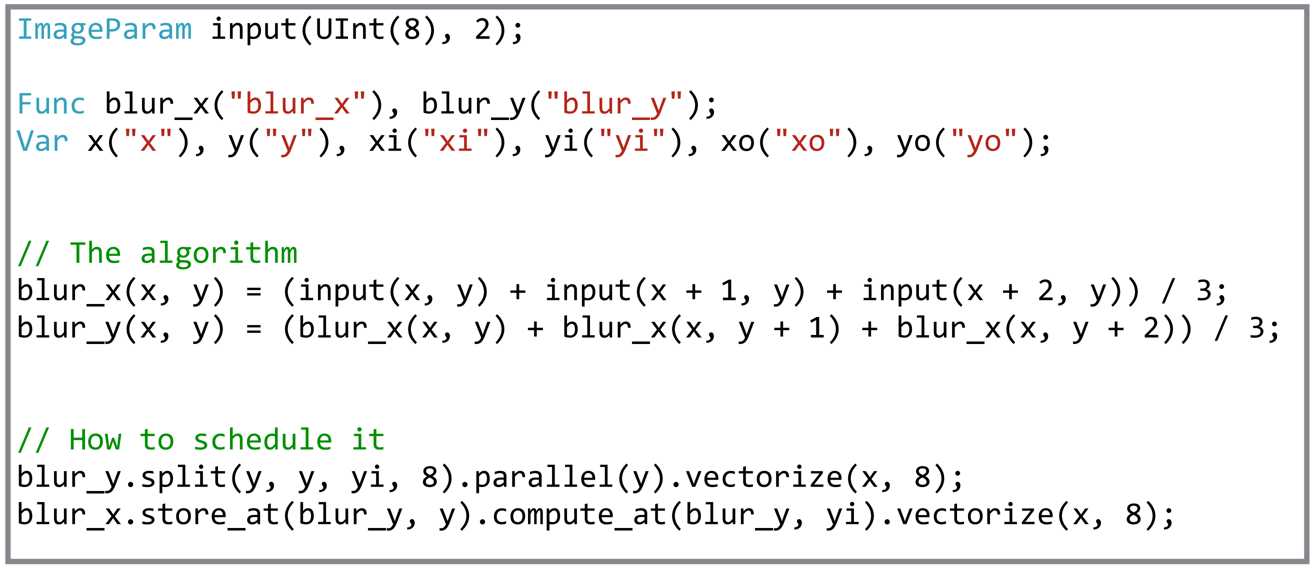
Helium



```
void blur(const Image &in, Image &blurred){

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

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

Simple Algorithm

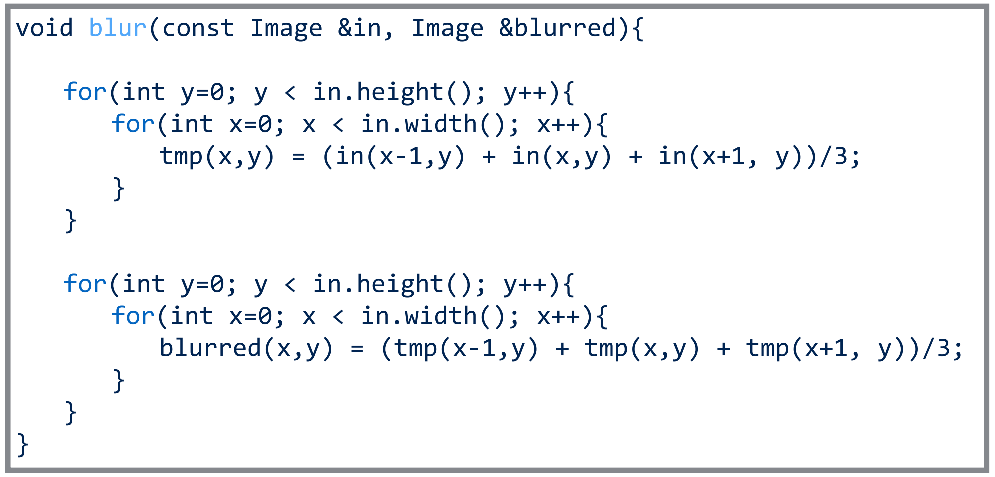
Halide



```
ImageParam input(UInt(8), 2);

Func blur_x("blur_x"), blur_y("blur_y");
Var x("x"), y("y"), xi("xi"), yi("yi"), xo("xo"), yo("yo");


// The algorithm
blur_x(x, y) = (input(x, y) + input(x + 1, y) + input(x + 2, y)) / 3;
blur_y(x, y) = (blur_x(x, y) + blur_x(x, y + 1) + blur_x(x, y + 2)) / 3;


// How to schedule it
blur_y.split(y, y, yi, 8).parallel(y).vectorize(x, 8);
blur_x.store_at(blur_y, y).compute_at(blur_y, yi).vectorize(x, 8);
```

Executable

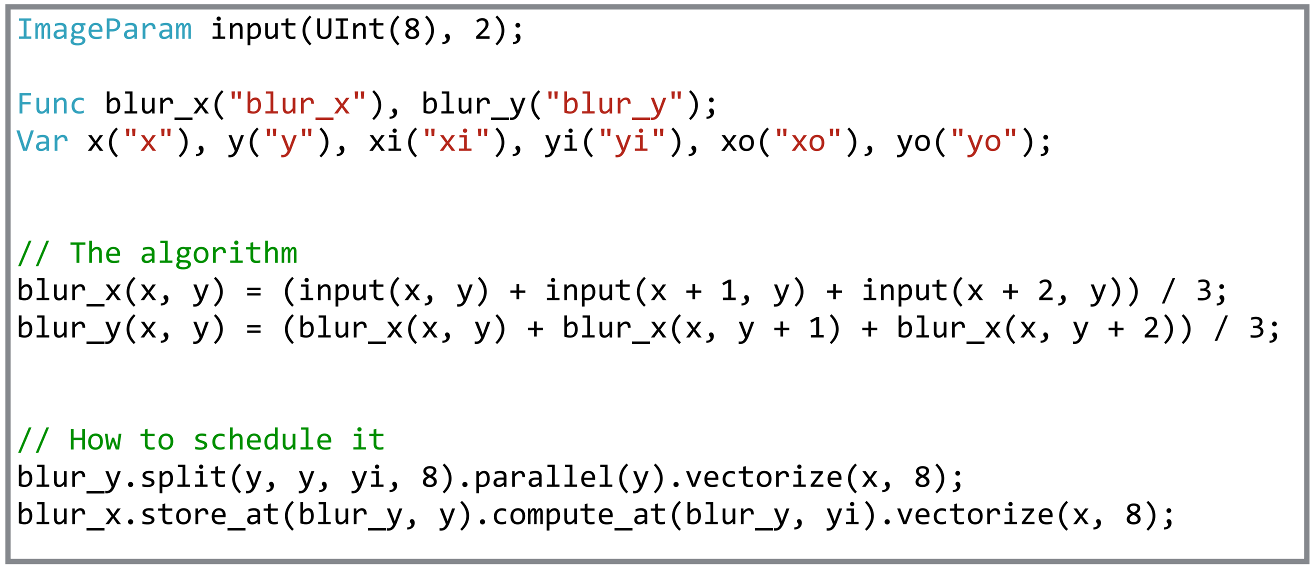
Helium



```
void blur(const Image &in, Image &blurred){

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

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

Simple Algorithm

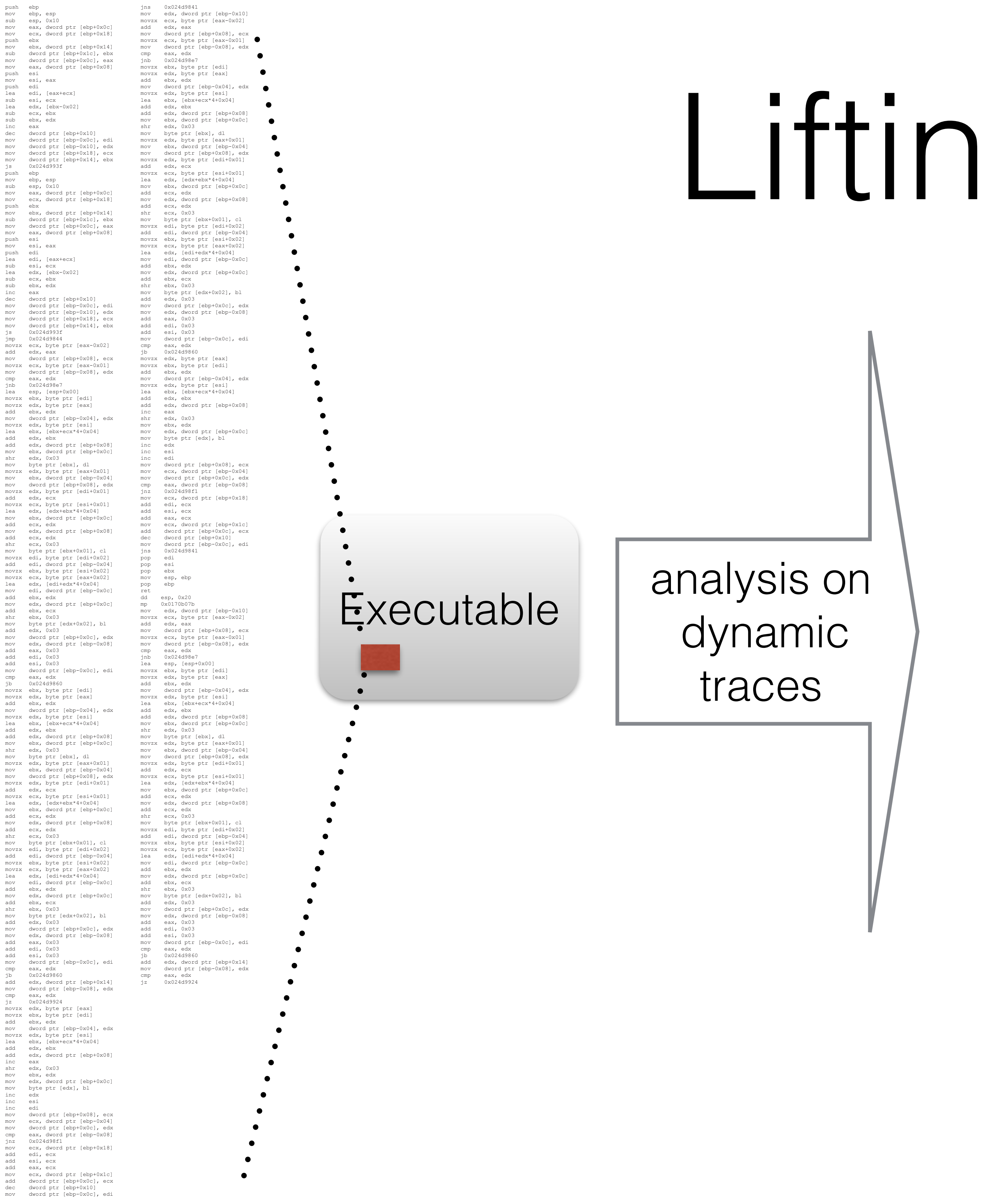
Halide



```
ImageParam input(UInt(8), 2);

Func blur_x("blur_x"), blur_y("blur_y");
Var x("x"), y("y"), xi("xi"), yi("yi"), xo("xo"), yo("yo");


// The algorithm
blur_x(x, y) = (input(x, y) + input(x + 1, y) + input(x + 2, y)) / 3;
blur_y(x, y) = (blur_x(x, y) + blur_x(x, y + 1) + blur_x(x, y + 2)) / 3;


// How to schedule it
blur_y.split(y, y, yi, 8).parallel(y).vectorize(x, 8);
blur_x.store_at(blur_y, y).compute_at(blur_y, yi).vectorize(x, 8);
```

Executable

Lifting



Assembly instructions

```
#include <Halide.h>
#include <vector>
using namespace std;
using namespace Halide;

int main(){
  Var x_0;
  Var x_1;
  ImageParam input_1(UInt(8),2);
  Func output_1;
  output_1(x_0,x_1) =
    cast<uint8_t>(((((2+
      (2*cast<uint32_t>(input_1(x_0+1,x_1+1))) +
         cast<uint32_t>(input_1(x_0,  x_1+1)) +
         cast<uint32_t>(input_1(x_0+2,x_1+1)))
      >> cast<uint32_t>(2))) & 255));
  vector<Argument> args;
  args.push_back(input_1);
  output_1.compile_to_file("halide_out_0",args);
  return 0;
}
```

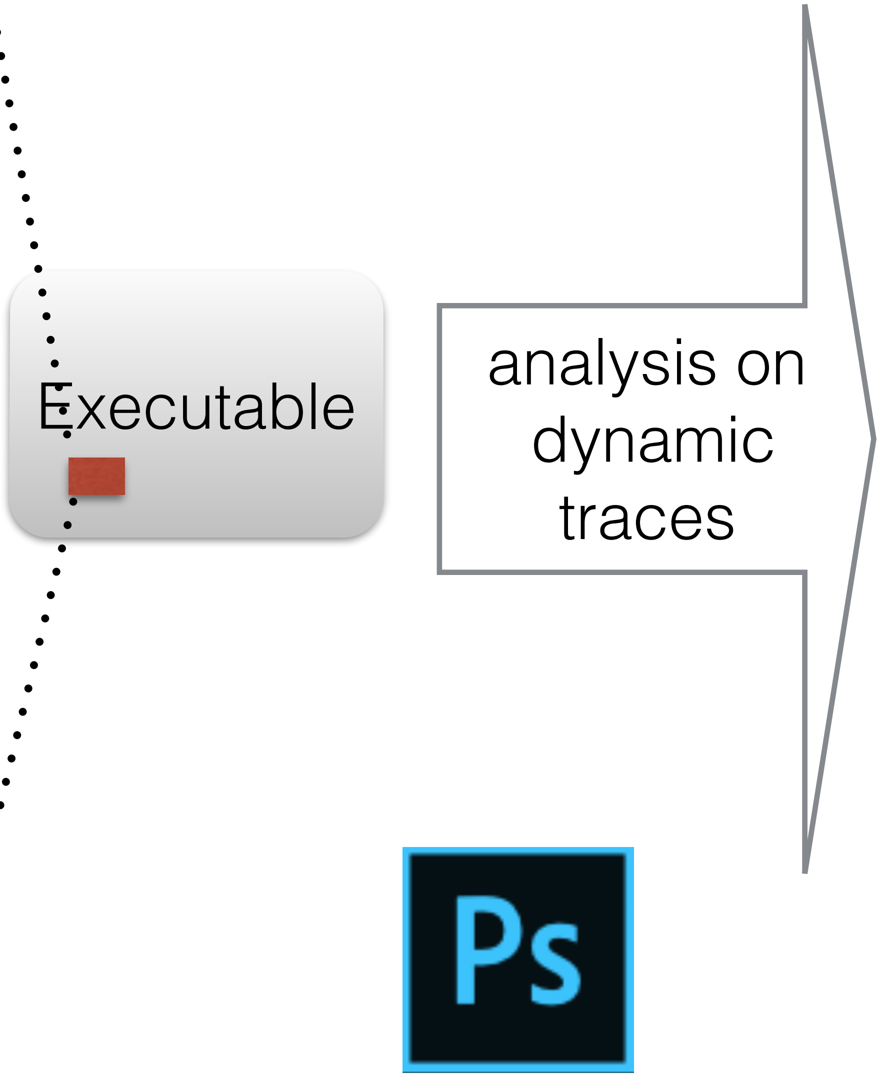
Generated Halide DSL code

Lifting

```
push   ebp
mov    ebp, esp
sub    esp, 0x10
mov    eax, dword ptr [ebp+0x0c]
mov    ecx, dword ptr [ebp+0x18]
push   ebx
mov    ebx, dword ptr [ebp+0x14]
sub    dword ptr [ebp+0x1c], ebx
mov    dword ptr [ebp+0x0c], eax
mov    eax, dword ptr [ebp+0x08]
push   esi
mov    esi, eax
push   edi
lea    edi, [eax+ecx]
sub    esi, ecx
lea    edx, [ebx-0x02]
sub    ecx, ebx
sub    ebx, edx
inc    eax
dec    dword ptr [ebp+0x10]
mov    dword ptr [ebp-0x0c], edi
mov    dword ptr [ebp-0x10], edx
mov    dword ptr [ebp+0x18], ecx
mov    dword ptr [ebp+0x14], ebx
js     0x024d993f
push   ebp
mov    ebp, esp
sub    esp, 0x10
mov    eax, dword ptr [ebp+0x0c]
mov    ecx, dword ptr [ebp+0x18]
push   ebx
mov    ebx, dword ptr [ebp+0x14]
sub    dword ptr [ebp+0x1c], ebx
mov    dword ptr [ebp+0x0c], eax
mov    eax, dword ptr [ebp+0x08]
push   esi
mov    esi, eax
push   edi
lea    edi, [eax+ecx]
sub    esi, ecx
lea    edx, [ebx-0x02]
sub    ecx, ebx
sub    ebx, edx
inc    eax
dec    dword ptr [ebp+0x10]
mov    dword ptr [ebp-0x0c], edi
mov    dword ptr [ebp-0x10], edx
mov    dword ptr [ebp+0x18], ecx
mov    dword ptr [ebp+0x14], ebx
js     0x024d993f
jmp    0x024d9844
movzx  ecx, byte ptr [eax-0x02]
add    edx, eax
mov    dword ptr [ebp+0x08], ecx
movzx  ecx, byte ptr [eax-0x01]
mov    dword ptr [ebp-0x08], edx
cmp    eax, edx
jnb    0x024d98e7
lea    esp, [esp+0x00]
movzx  ebx, byte ptr [edi]
movzx  edx, byte ptr [eax]
add    ebx, edx
mov    dword ptr [ebp-0x04], edx
movzx  edx, byte ptr [esi]
lea    ebx, [ebx+ecx*4+0x04]
add    edx, ebx
add    edx, dword ptr [ebp+0x08]
mov    ebx, dword ptr [ebp+0x0c]
shr    edx, 0x03
mov    byte ptr [ebx], dl
movzx  edx, byte ptr [eax+0x01]
mov    ebx, dword ptr [ebp-0x04]
mov    dword ptr [ebp+0x08], edx
movzx  edx, byte ptr [edi+0x01]
add    edx, ecx
movzx  ecx, byte ptr [esi+0x01]
lea    edx, [edx+ebx*4+0x04]
mov    ebx, dword ptr [ebp+0x0c]
add    ecx, edx
mov    edx, dword ptr [ebp+0x08]
add    ecx, edx
shr    ecx, 0x03
mov    byte ptr [ebx+0x01], cl
movzx  edi, byte ptr [edi+0x02]
add    edi, dword ptr [ebp-0x04]
movzx  ebx, byte ptr [esi+0x02]
movzx  ecx, byte ptr [eax+0x02]
lea    edx, [edi+edx*4+0x04]
mov    edi, dword ptr [ebp-0x0c]
add    ebx, edx
mov    edx, dword ptr [ebp+0x0c]
add    ebx, ecx
shr    ebx, 0x03
mov    byte ptr [edx+0x02], bl
add    edx, 0x03
mov    dword ptr [ebp+0x0c], edx
mov    edx, dword ptr [ebp-0x08]
add    eax, 0x03
add    edi, 0x03
add    esi, 0x03
mov    dword ptr [ebp-0x0c], edi
cmp    eax, edx
jb     0x024d9860
movzx  ebx, byte ptr [edi]
movzx  edx, byte ptr [eax]
add    ebx, edx
mov    dword ptr [ebp-0x04], edx
movzx  edx, byte ptr [esi]
lea    ebx, [ebx+ecx*4+0x04]
add    edx, ebx
add    edx, dword ptr [ebp+0x08]
mov    ebx, dword ptr [ebp+0x0c]
shr    edx, 0x03
mov    byte ptr [ebx], dl
movzx  edx, byte ptr [eax+0x01]
mov    ebx, dword ptr [ebp-0x04]
mov    dword ptr [ebp+0x08], edx
movzx  edx, byte ptr [edi+0x01]
add    edx, ecx
movzx  ecx, byte ptr [esi+0x01]
lea    edx, [edx+ebx*4+0x04]
mov    ebx, dword ptr [ebp+0x0c]
add    ecx, edx
mov    edx, dword ptr [ebp+0x08]
add    ecx, edx
shr    ecx, 0x03
mov    byte ptr [ebx+0x01], cl
movzx  edi, byte ptr [edi+0x02]
add    edi, dword ptr [ebp-0x04]
movzx  ebx, byte ptr [esi+0x02]
movzx  ecx, byte ptr [eax+0x02]
lea    edx, [edi+edx*4+0x04]
mov    edi, dword ptr [ebp-0x0c]
add    ebx, edx
mov    edx, dword ptr [ebp+0x0c]
add    ebx, ecx
shr    ebx, 0x03
mov    byte ptr [edx+0x02], bl
add    edx, 0x03
mov    dword ptr [ebp+0x0c], edx
mov    edx, dword ptr [ebp-0x08]
add    eax, 0x03
add    edi, 0x03
add    esi, 0x03
mov    dword ptr [ebp-0x0c], edi
cmp    eax, edx
jb     0x024d9860
add    edx, dword ptr [ebp+0x14]
mov    dword ptr [ebp-0x08], edx
cmp    eax, edx
jz     0x024d9924
movzx  edx, byte ptr [eax]
movzx  ebx, byte ptr [edi]
add    ebx, edx
mov    dword ptr [ebp-0x04], edx
movzx  edx, byte ptr [esi]
lea    ebx, [ebx+ecx*4+0x04]
add    edx, ebx
add    edx, dword ptr [ebp+0x08]
inc    eax
shr    edx, 0x03
mov    ebx, edx
mov    edx, dword ptr [ebp+0x0c]
mov    byte ptr [edx], bl
inc    edx
inc    esi
inc    edi
mov    dword ptr [ebp+0x08], ecx
mov    ecx, dword ptr [ebp-0x04]
mov    dword ptr [ebp+0x0c], edx
cmp    eax, dword ptr [ebp-0x08]
jnz    0x024d98f1
mov    ecx, dword ptr [ebp+0x18]
add    edi, ecx
add    esi, ecx
add    eax, ecx
mov    ecx, dword ptr [ebp+0x1c]
add    dword ptr [ebp+0x0c], ecx
dec    dword ptr [ebp+0x10]
mov    dword ptr [ebp-0x0c], edi
```

```
jns    0x024d9841
mov    edx, dword ptr [ebp-0x10]
movzx  ecx, byte ptr [eax-0x02]
add    edx, eax
mov    dword ptr [ebp+0x08], ecx
movzx  ecx, byte ptr [eax-0x01]
mov    dword ptr [ebp-0x08], edx
cmp    eax, edx
jnb    0x024d98e7
movzx  ebx, byte ptr [edi]
movzx  edx, byte ptr [eax]
add    ebx, edx
mov    dword ptr [ebp-0x04], edx
movzx  edx, byte ptr [esi]
lea    ebx, [ebx+ecx*4+0x04]
add    edx, ebx
add    edx, dword ptr [ebp+0x08]
mov    ebx, dword ptr [ebp+0x0c]
shr    edx, 0x03
mov    byte ptr [ebx], dl
movzx  edx, byte ptr [eax+0x01]
mov    ebx, dword ptr [ebp-0x04]
mov    dword ptr [ebp+0x08], edx
movzx  edx, byte ptr [edi+0x01]
add    edx, ecx
movzx  ecx, byte ptr [esi+0x01]
lea    edx, [edx+ebx*4+0x04]
mov    ebx, dword ptr [ebp+0x0c]
add    ecx, edx
mov    edx, dword ptr [ebp+0x08]
add    ecx, edx
shr    ecx, 0x03
mov    byte ptr [ebx+0x01], cl
movzx  edi, byte ptr [edi+0x02]
add    edi, dword ptr [ebp-0x04]
movzx  ebx, byte ptr [esi+0x02]
movzx  ecx, byte ptr [eax+0x02]
lea    edx, [edi+edx*4+0x04]
mov    edi, dword ptr [ebp-0x0c]
add    ebx, edx
mov    edx, dword ptr [ebp+0x0c]
add    ebx, ecx
shr    ebx, 0x03
mov    byte ptr [edx+0x02], bl
add    edx, 0x03
mov    dword ptr [ebp+0x0c], edx
mov    edx, dword ptr [ebp-0x08]
add    eax, 0x03
add    edi, 0x03
add    esi, 0x03
mov    dword ptr [ebp-0x0c], edi
cmp    eax, edx
jb     0x024d9860
movzx  edx, byte ptr [eax]
movzx  ebx, byte ptr [edi]
add    ebx, edx
mov    dword ptr [ebp-0x04], edx
movzx  edx, byte ptr [esi]
lea    ebx, [ebx+ecx*4+0x04]
add    edx, ebx
add    edx, dword ptr [ebp+0x08]
inc    eax
shr    edx, 0x03
mov    ebx, edx
mov    edx, dword ptr [ebp+0x0c]
mov    byte ptr [edx], bl
inc    edx
inc    esi
inc    edi
mov    dword ptr [ebp+0x08], ecx
mov    ecx, dword ptr [ebp-0x04]
mov    dword ptr [ebp+0x0c], edx
cmp    eax, dword ptr [ebp-0x08]
jnz    0x024d98f1
mov    ecx, dword ptr [ebp+0x18]
add    edi, ecx
add    esi, ecx
add    eax, ecx
mov    ecx, dword ptr [ebp+0x1c]
add    dword ptr [ebp+0x0c], ecx
dec    dword ptr [ebp+0x10]
mov    dword ptr [ebp-0x0c], edi
jns    0x024d9841
pop    edi
pop    esi
pop    ebx
mov    esp, ebp
pop    ebp
ret
dd   esp, 0x20
mp   0x0170b07b
mov    edx, dword ptr [ebp-0x10]
movzx  ecx, byte ptr [eax-0x02]
add    edx, eax
mov    dword ptr [ebp+0x08], ecx
movzx  ecx, byte ptr [eax-0x01]
mov    dword ptr [ebp-0x08], edx
cmp    eax, edx
jnb    0x024d98e7
lea    esp, [esp+0x00]
movzx  ebx, byte ptr [edi]
movzx  edx, byte ptr [eax]
add    ebx, edx
mov    dword ptr [ebp-0x04], edx
movzx  edx, byte ptr [esi]
lea    ebx, [ebx+ecx*4+0x04]
add    edx, ebx
add    edx, dword ptr [ebp+0x08]
mov    ebx, dword ptr [ebp+0x0c]
shr    edx, 0x03
mov    byte ptr [ebx], dl
movzx  edx, byte ptr [eax+0x01]
mov    ebx, dword ptr [ebp-0x04]
mov    dword ptr [ebp+0x08], edx
movzx  edx, byte ptr [edi+0x01]
add    edx, ecx
movzx  ecx, byte ptr [esi+0x01]
lea    edx, [edx+ebx*4+0x04]
mov    ebx, dword ptr [ebp+0x0c]
add    ecx, edx
mov    edx, dword ptr [ebp+0x08]
add    ecx, edx
shr    ecx, 0x03
mov    byte ptr [ebx+0x01], cl
movzx  edi, byte ptr [edi+0x02]
add    edi, dword ptr [ebp-0x04]
movzx  ebx, byte ptr [esi+0x02]
movzx  ecx, byte ptr [eax+0x02]
lea    edx, [edi+edx*4+0x04]
mov    edi, dword ptr [ebp-0x0c]
add    ebx, edx
mov    edx, dword ptr [ebp+0x0c]
add    ebx, ecx
shr    ebx, 0x03
mov    byte ptr [edx+0x02], bl
add    edx, 0x03
mov    dword ptr [ebp+0x0c], edx
mov    edx, dword ptr [ebp-0x08]
add    eax, 0x03
add    edi, 0x03
add    esi, 0x03
mov    dword ptr [ebp-0x0c], edi
cmp    eax, edx
jb     0x024d9860
add    edx, dword ptr [ebp+0x14]
mov    dword ptr [ebp-0x08], edx
cmp    eax, edx
jz     0x024d9924
```

Assembly instructions



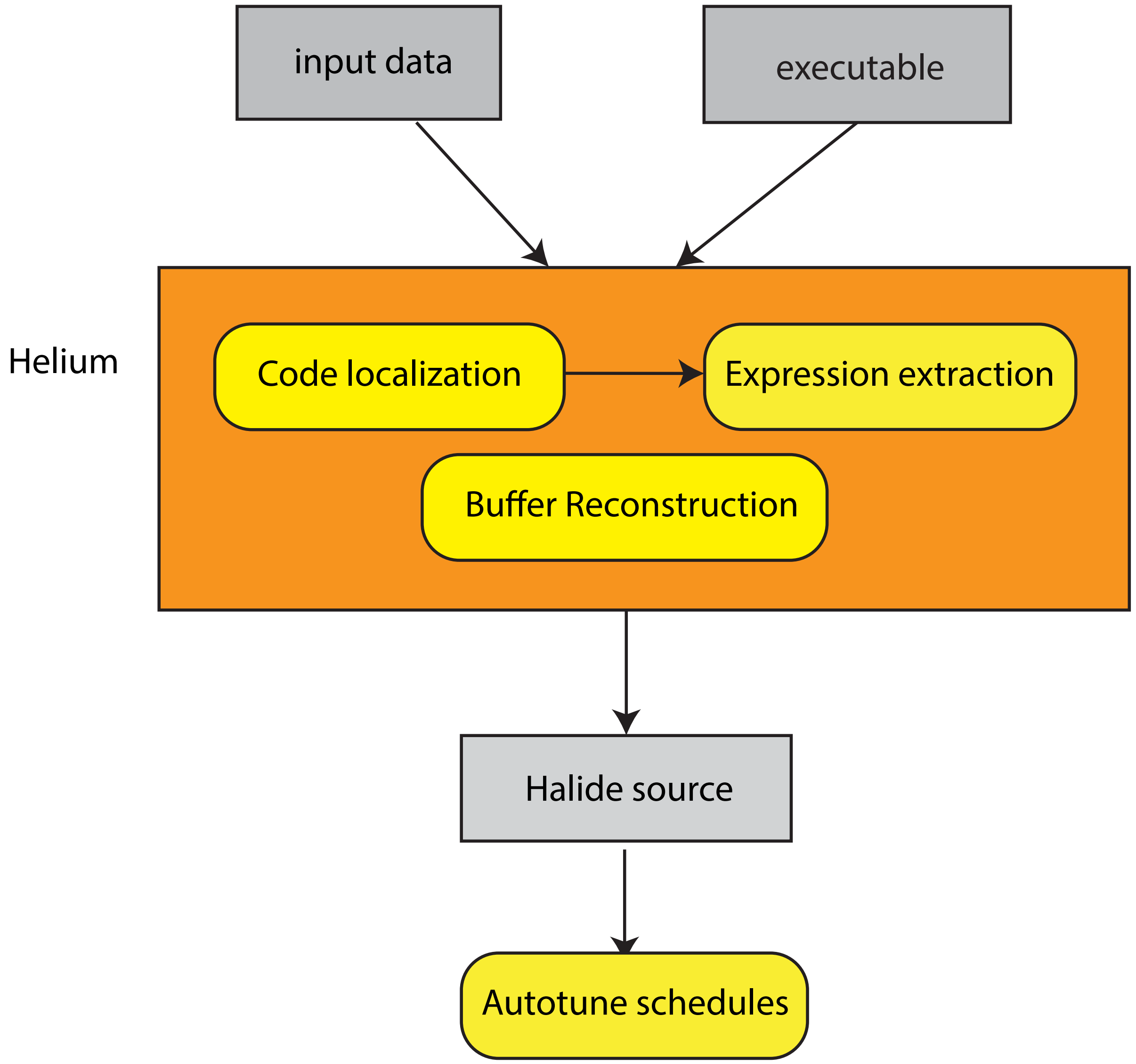
analysis on dynamic traces

75% faster

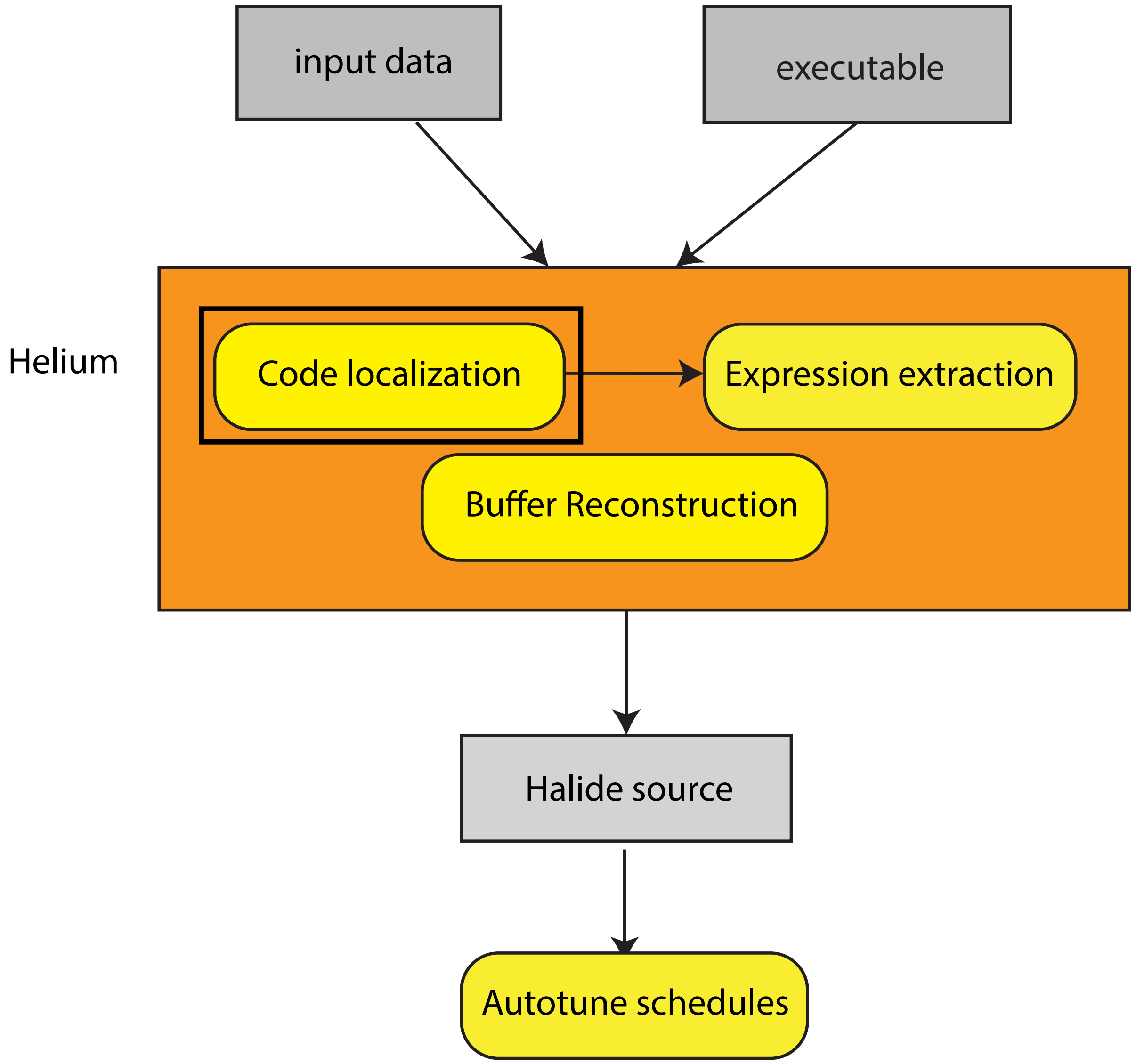
```
#include <Halide.h>
#include <vector>
using namespace std;
using namespace Halide;

int main(){
  Var x_0;
  Var x_1;
  ImageParam input_1(UInt(8),2);
  Func output_1;
  output_1(x_0,x_1) =
    cast<uint8_t>(((((2+
      (2*cast<uint32_t>(input_1(x_0+1,x_1+1))) +
         cast<uint32_t>(input_1(x_0,  x_1+1)) +
         cast<uint32_t>(input_1(x_0+2,x_1+1)))
      >> cast<uint32_t>(2))) & 255));
  vector<Argument> args;
  args.push_back(input_1);
  output_1.compile_to_file("halide_out_0",args);
  return 0;
}
```

Generated Halide DSL code

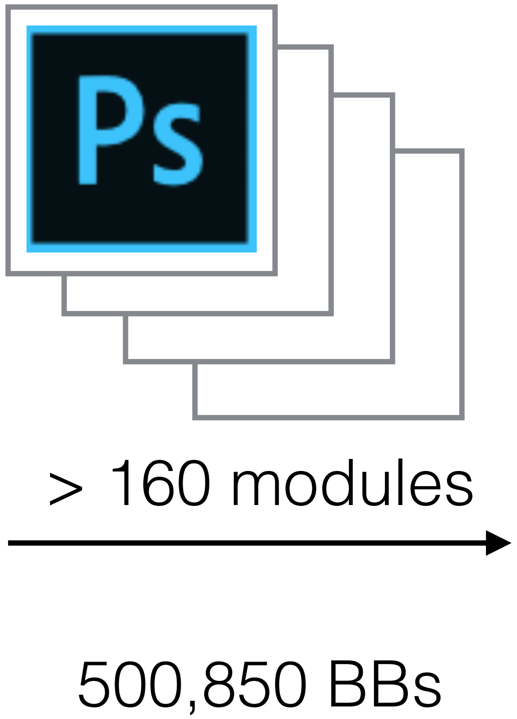
Workflow


input data
executable
Helium
Code localization
Expression extraction
Buffer Reconstruction
Halide source
Autotune schedules

Workflow


input data
executable
Helium
Code localization
Expression extraction
Buffer Reconstruction
Halide source
Autotune schedules

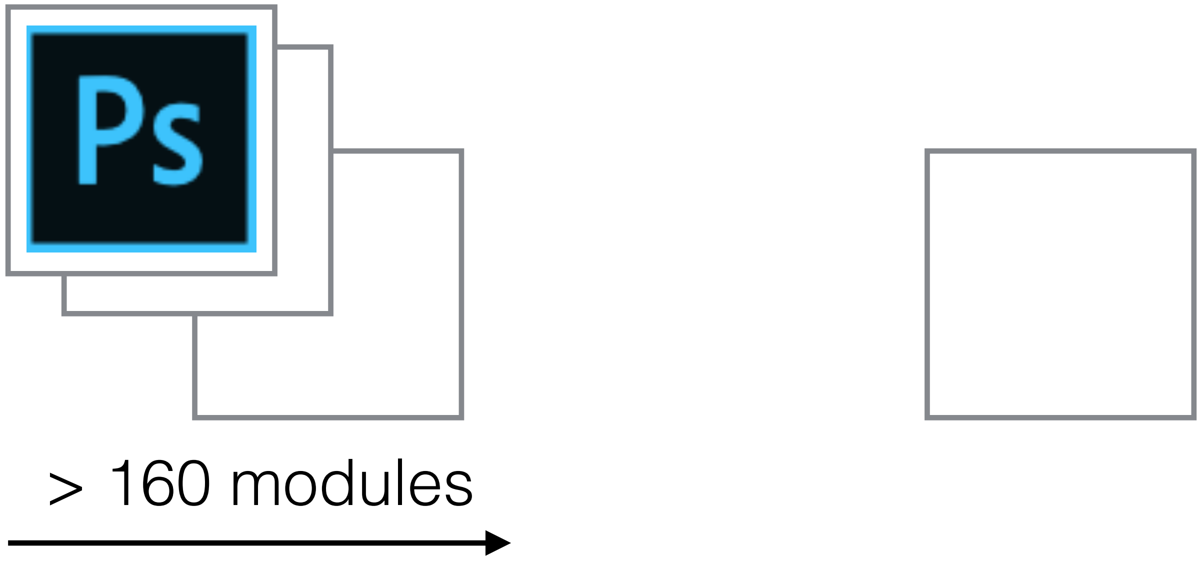
Code Localization



> 160 modules

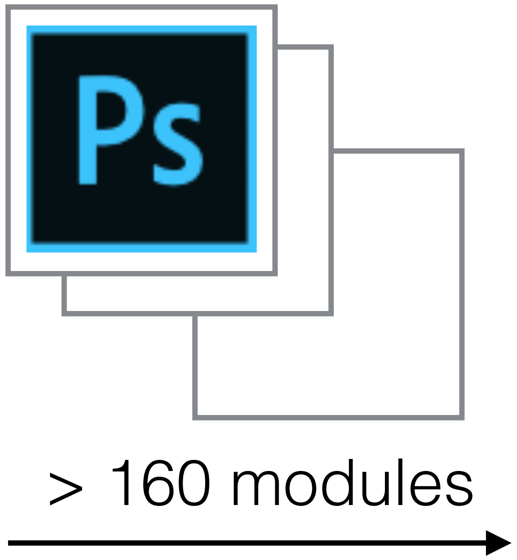
500,850 BBs

Code Localization



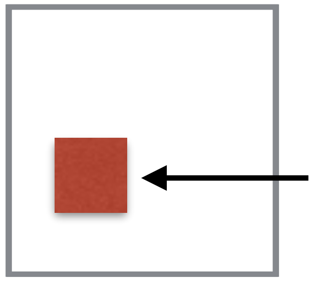
500,850 BBs

Code Localization



> 160 modules

500,850 BBs

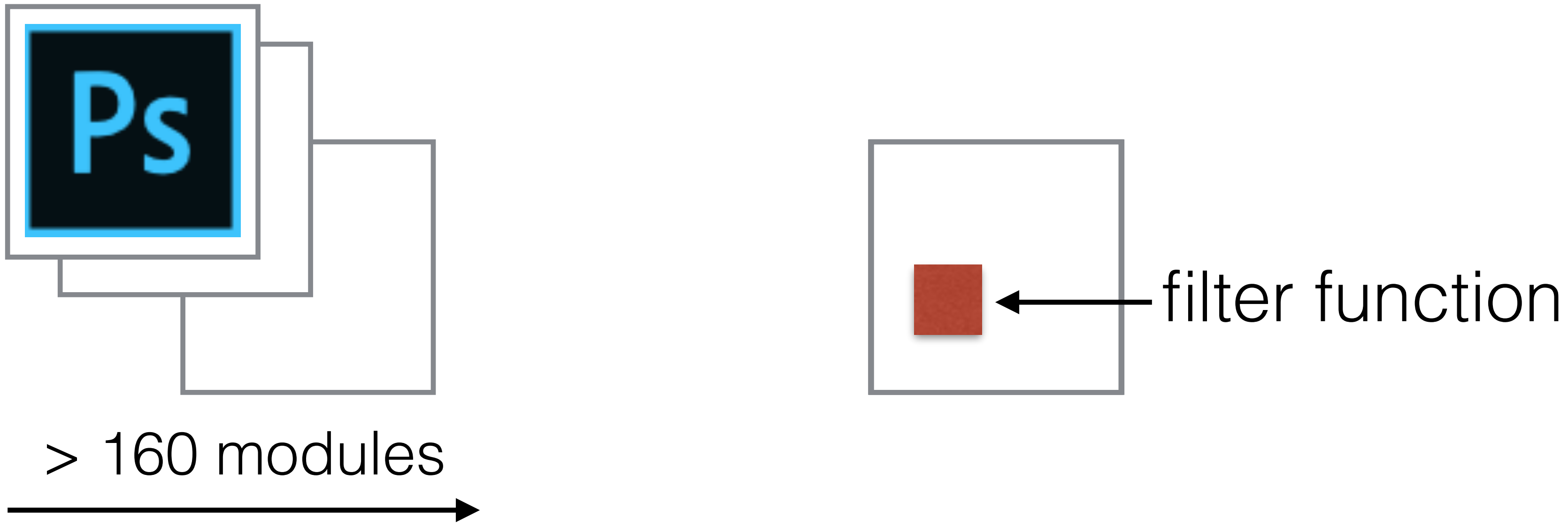


filter function

14 BBs

blur filter

Code Localization



> 160 modules

filter function

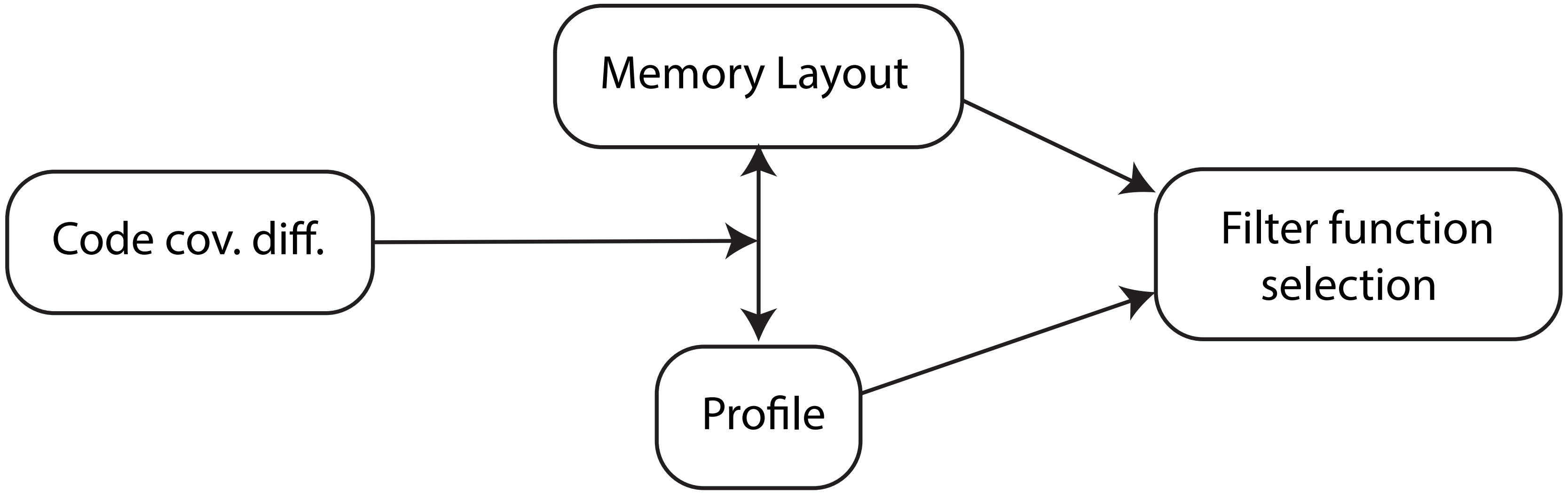
500,850 BBs

14 BBs

blur filter

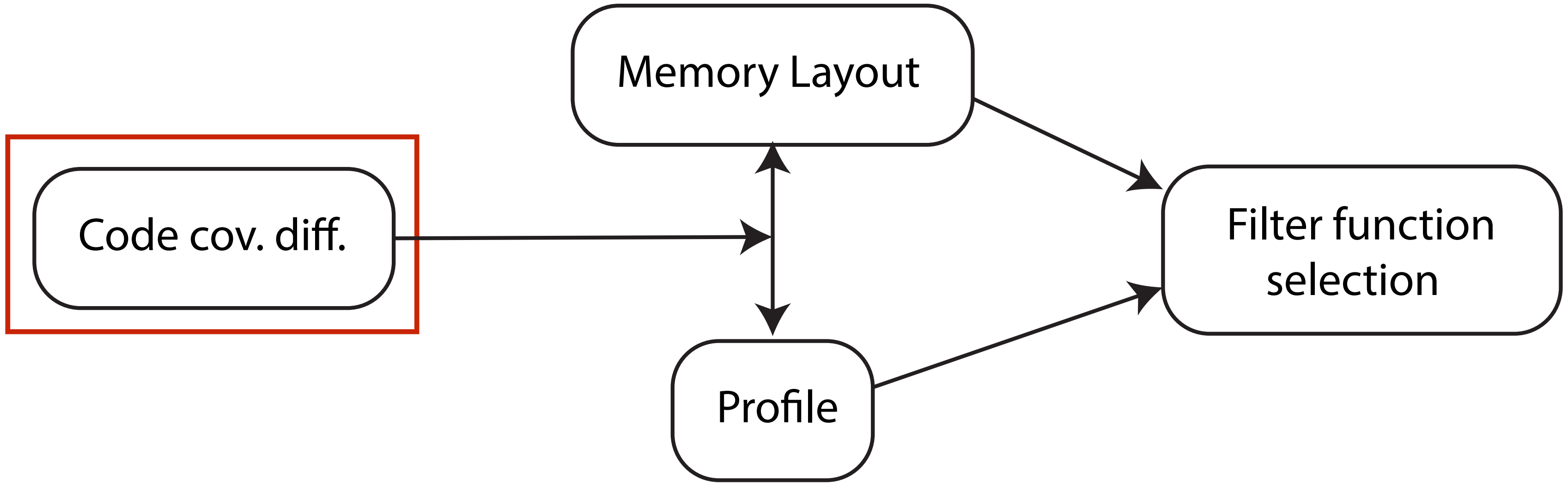
- Analyzing the entire application at runtime is not scalable
- Strategy - Progressively localize

Code Localization



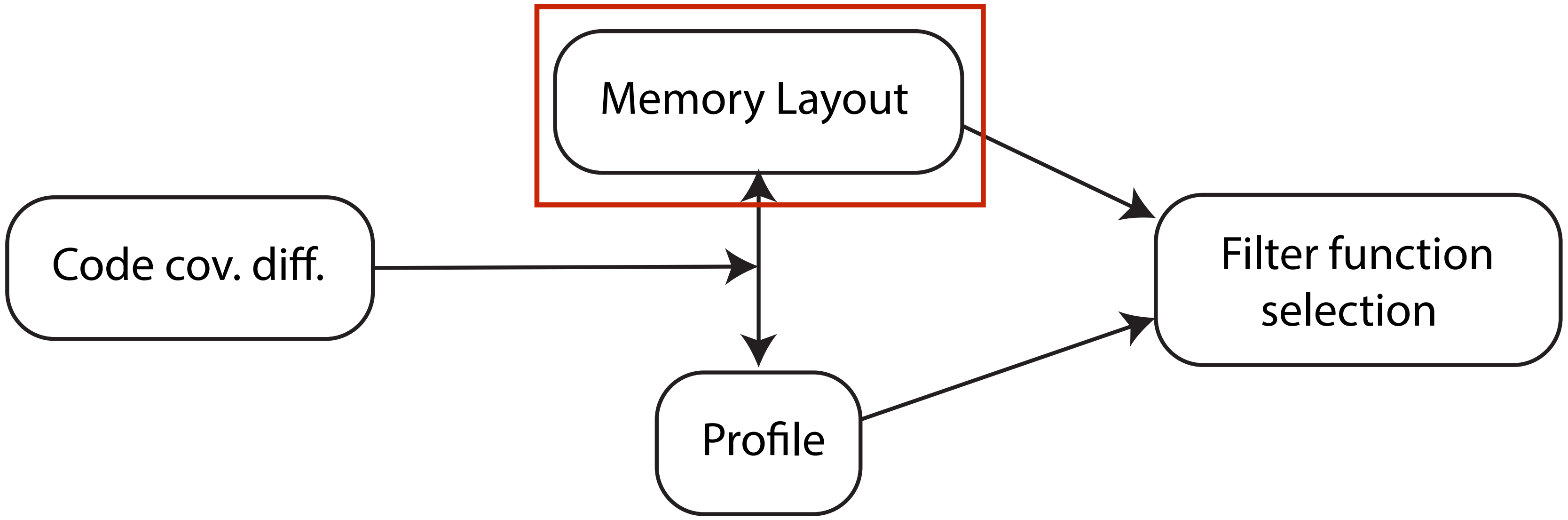
Scope(BBs)	500,850	3,850	14	⬇
Detail(kB / BB)	0.04	3.87	271.42	⬆

Code Localization



Scope(BBs)	500,850	3,850	14	↓
Detail(kB / BB)	0.04	3.87	271.42	↑

Code Localization



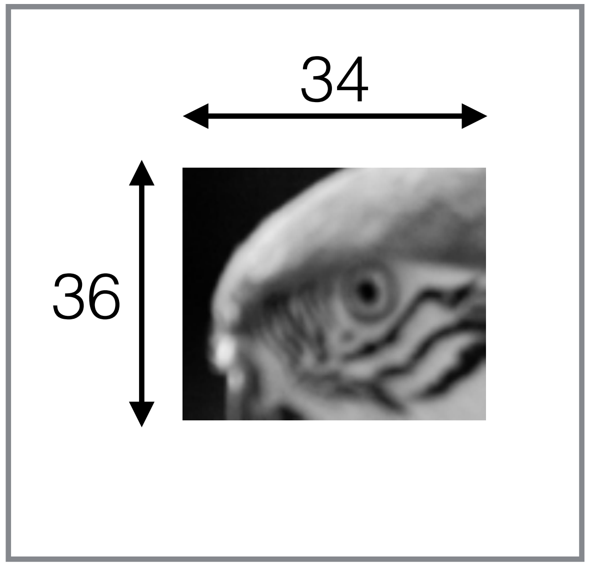
Scope(BBs)	500,850	3,850	14	↓
Detail(kB / BB)	0.04	3.87	271.42	↑

Buffer Reconstruction

```
//code for computing PADDING - 16 byte aligned

uint8_t input_b(WIDTH + 2 + PADDING, HEIGHT);
uint8_t output_b(WIDTH + PADDING, HEIGHT);

//code for loading image with padding

for(int y = 0; y < HEIGHT; y++){
  for(int x = 0; x < WIDTH; x++){
    output_b(x, y) = (input_b(x, y)
                          + input_b(x+1, y) + input_b(x+2, y)) / 3;
  }
}
```

Buffer Reconstruction

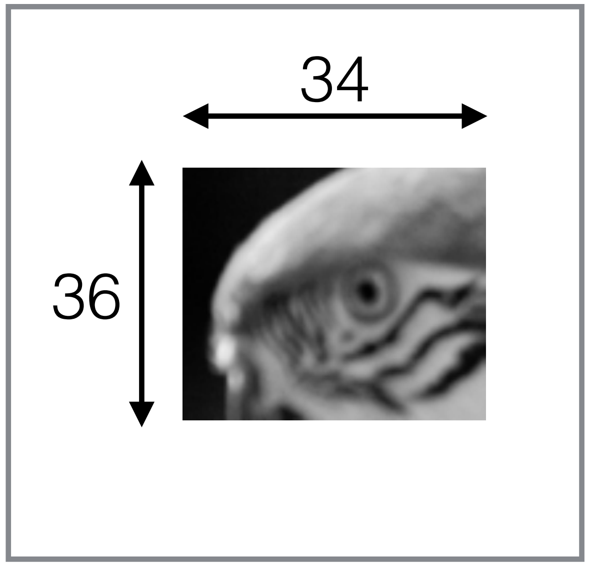
```
//code for computing PADDING - 16 byte aligned

uint8_t input_b(WIDTH + 2 + PADDING, HEIGHT);
uint8_t output_b(WIDTH + PADDING, HEIGHT);

//code for loading image with padding

for(int y = 0; y < HEIGHT; y++){
  for(int x = 0; x < WIDTH; x++){
    output_b(x, y) = (input_b(x, y)
                          + input_b(x+1, y) + input_b(x+2, y)) / 3;
  }
}
```



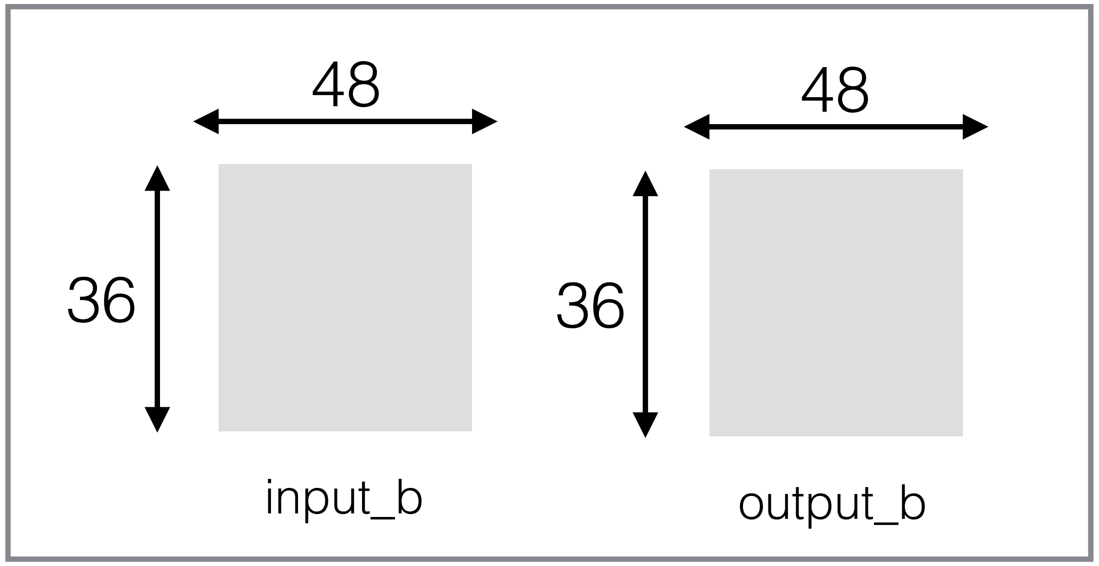
Input

Buffer Reconstruction

```
//code for computing PADDING - 16 byte aligned

uint8_t input_b(WIDTH + 2 + PADDING, HEIGHT);
uint8_t output_b(WIDTH + PADDING, HEIGHT);

//code for loading image with padding

for(int y = 0; y < HEIGHT; y++){
  for(int x = 0; x < WIDTH; x++){
    output_b(x, y) = (input_b(x, y)
                          + input_b(x+1, y) + input_b(x+2, y)) / 3;
  }
}
```

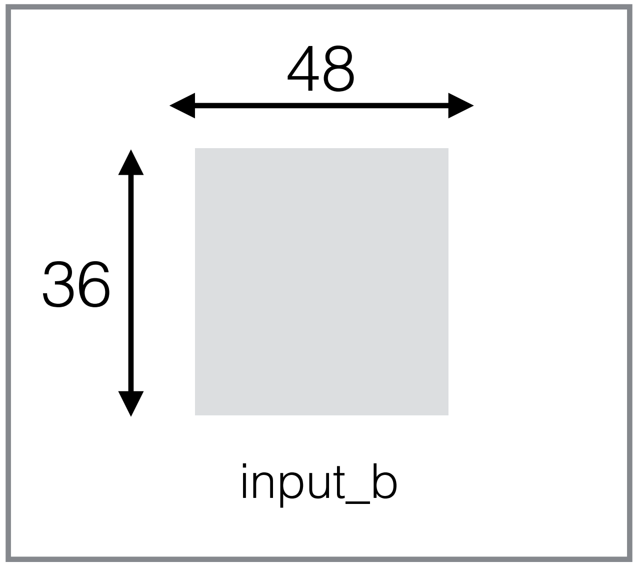


Input

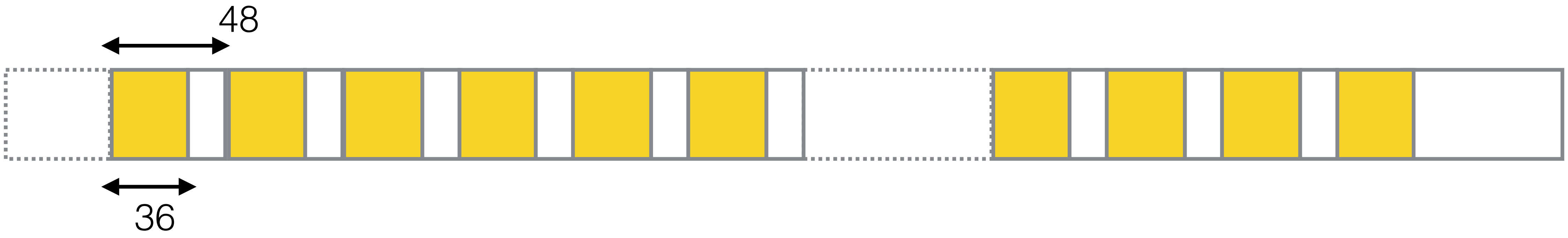


Logical buffers

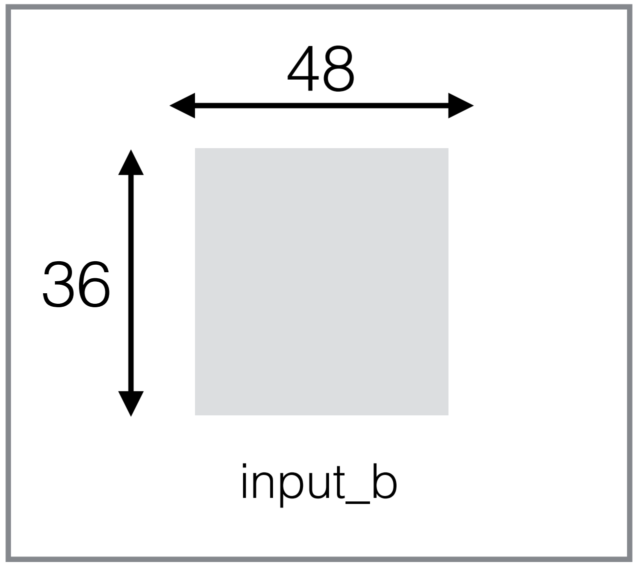
Buffer Reconstruction



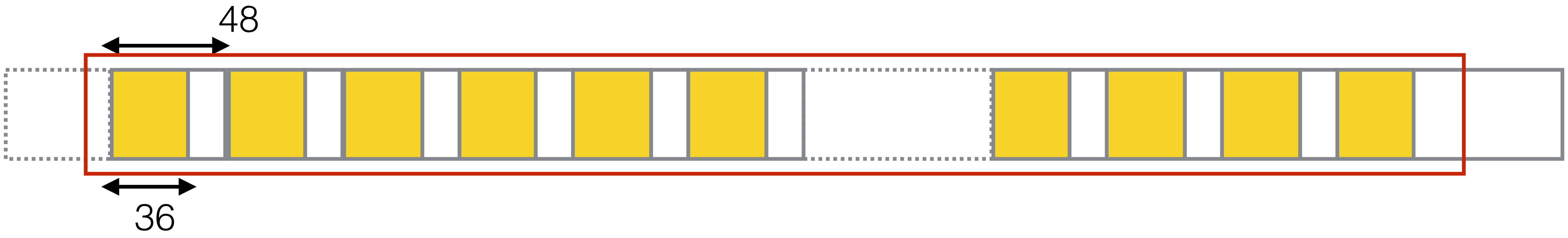
Linearized Memory



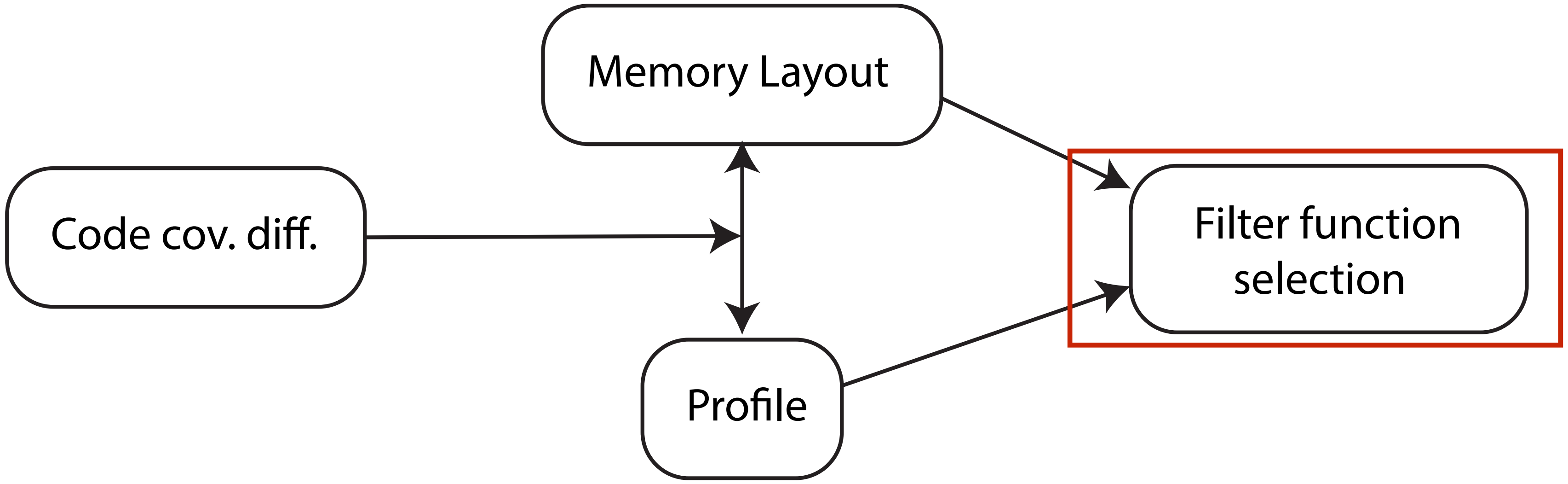
Buffer Reconstruction



Linearized Memory

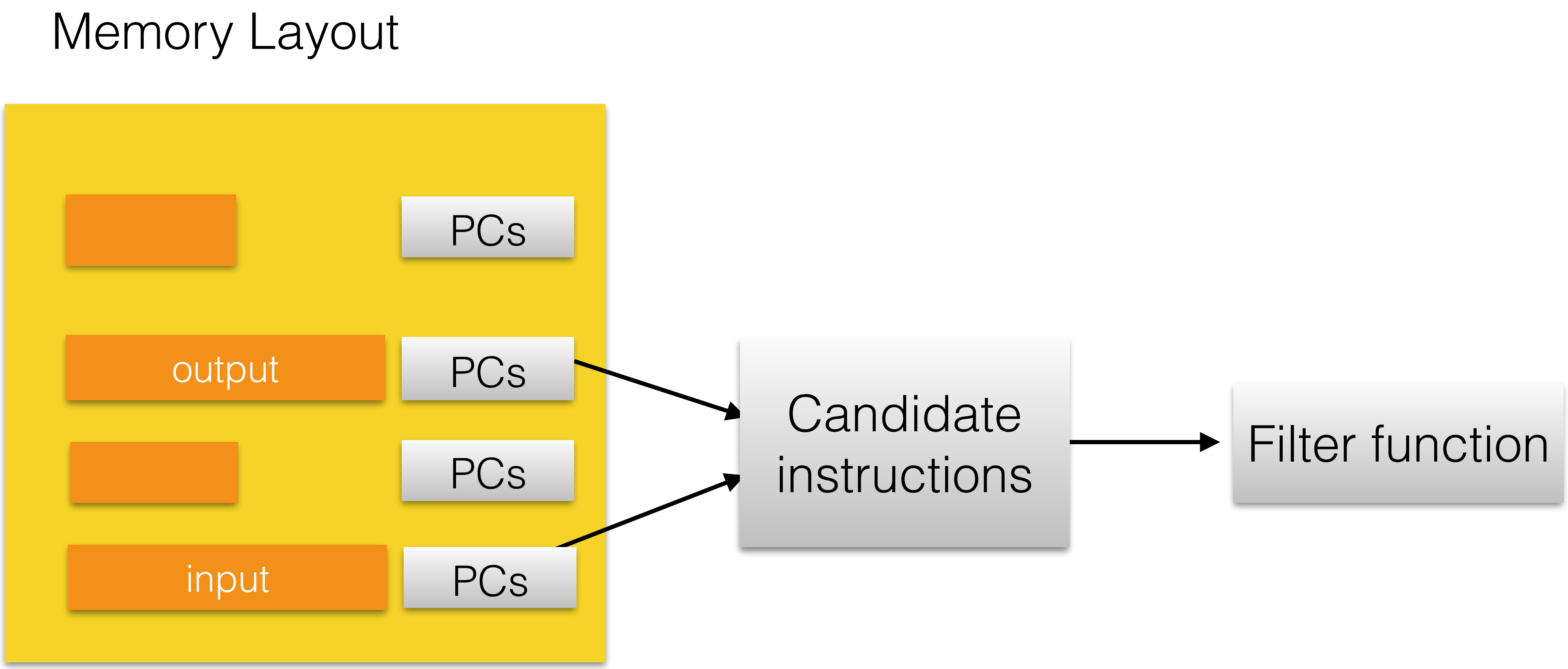


Code Localization



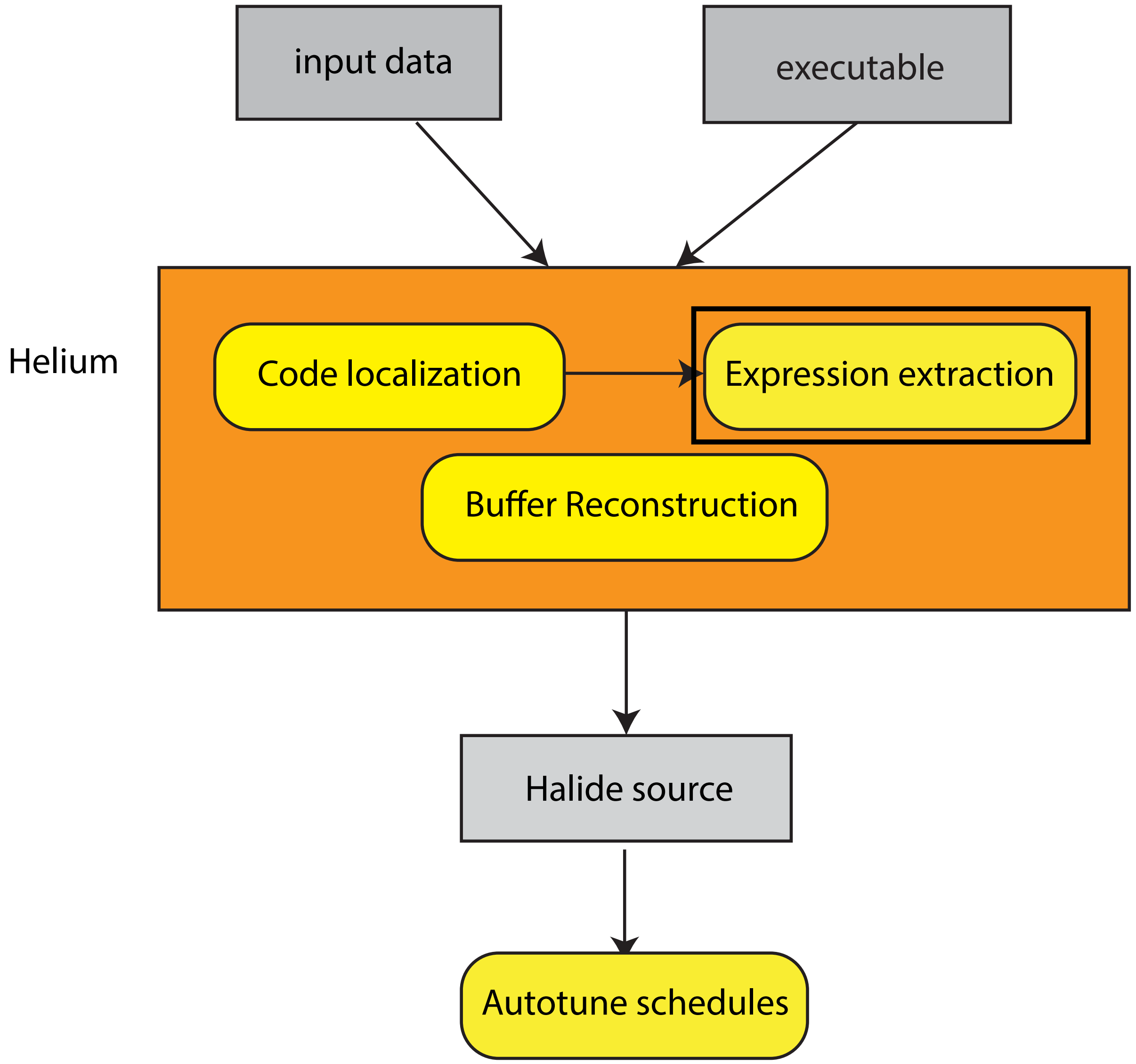
Scope(BBs)	500,850	3,850	14	⬇
Detail(kB / BB)	0.04	3.87	271.42	⬆

Filter Function Selection



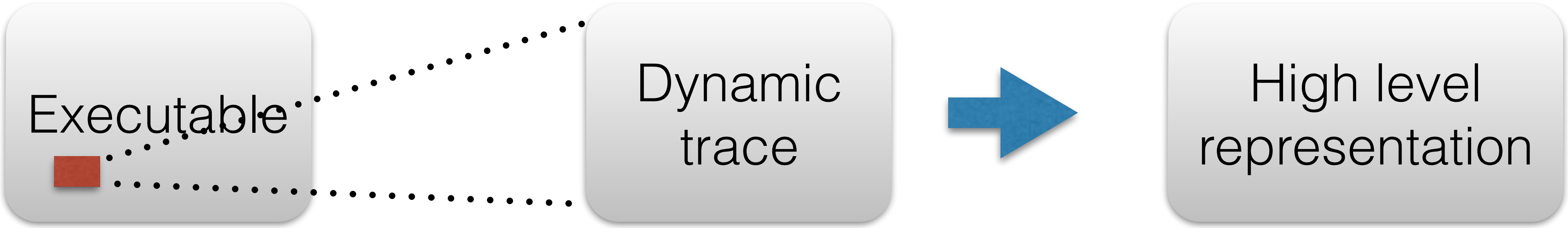
- Filter function is chosen as the function with most candidate instructions

Workflow


input data
executable
Helium
Code localization
Expression extraction
Buffer Reconstruction
Halide source
Autotune schedules

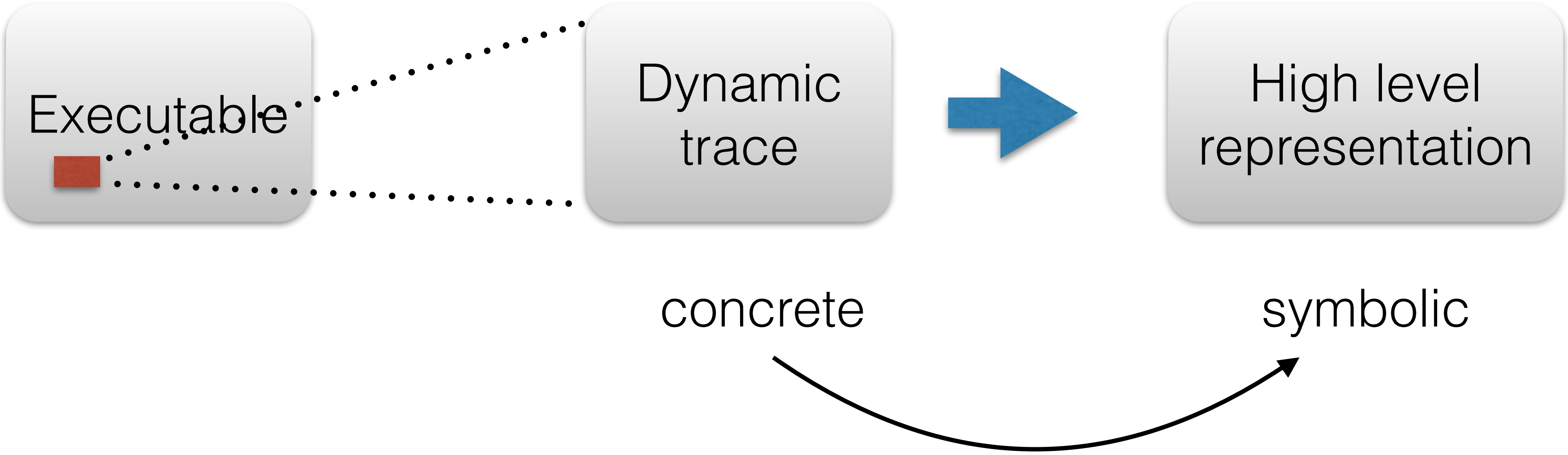
Expression Extraction

- Recover stencil computation without scheduling information

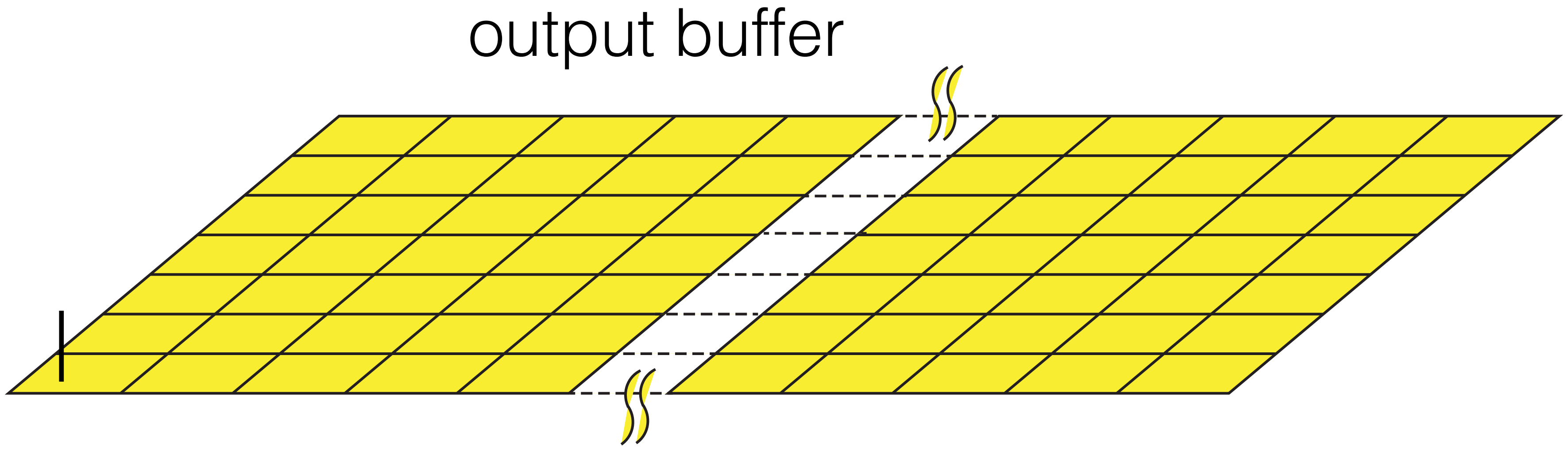


Expression Extraction

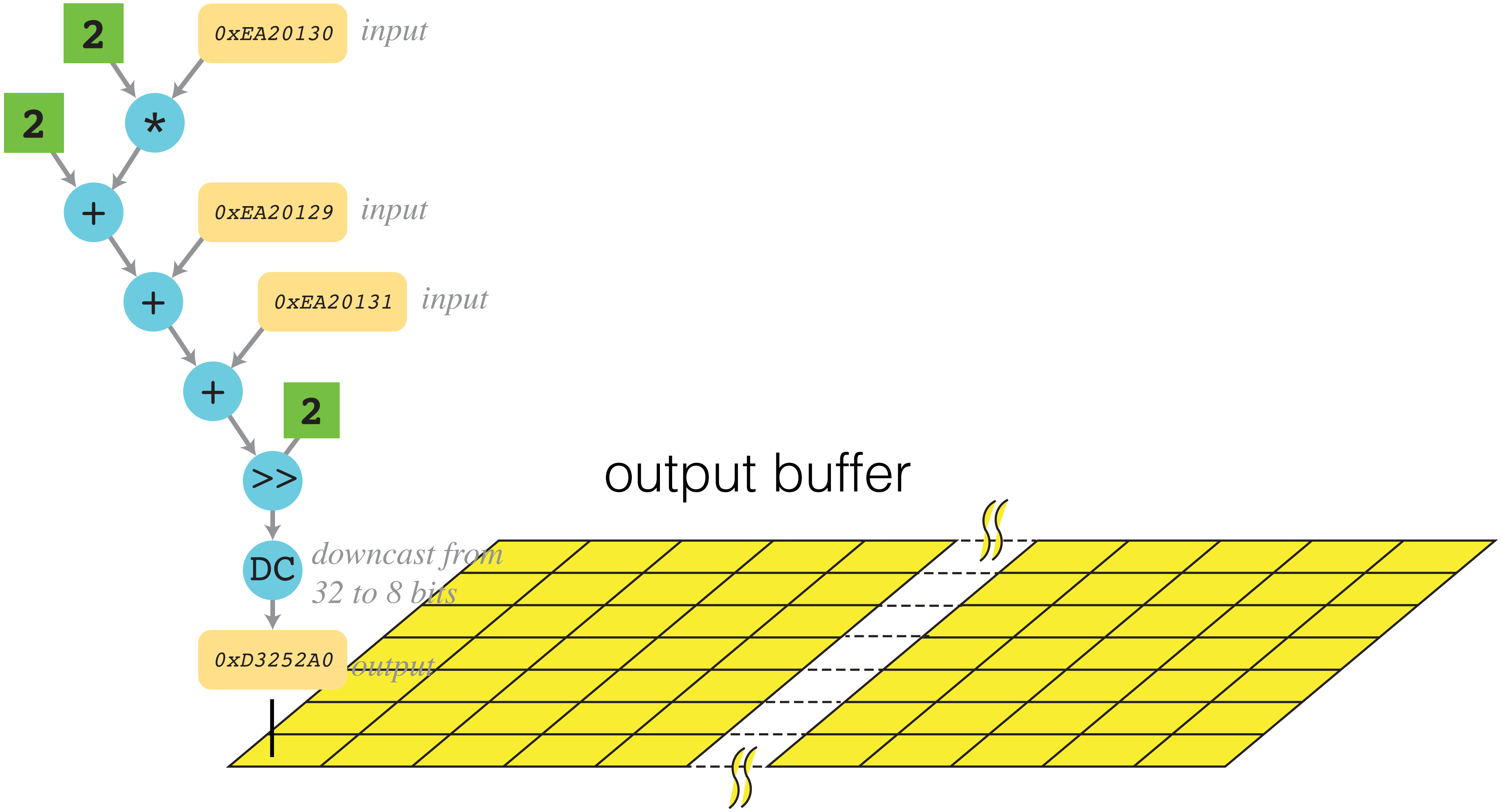
- Recover stencil computation without scheduling information



Concrete Data Dependency Trees

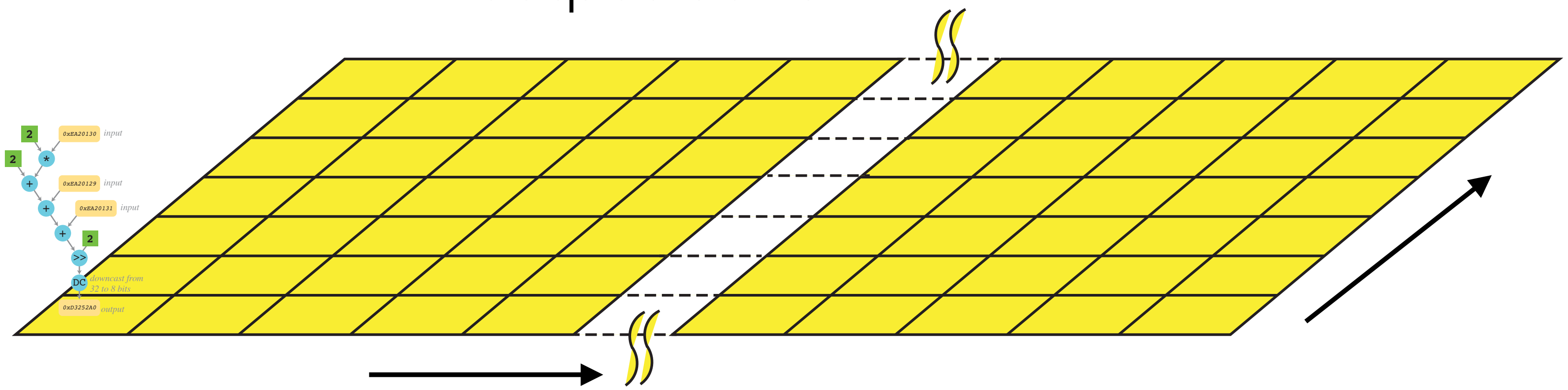


Concrete Data Dependency Trees



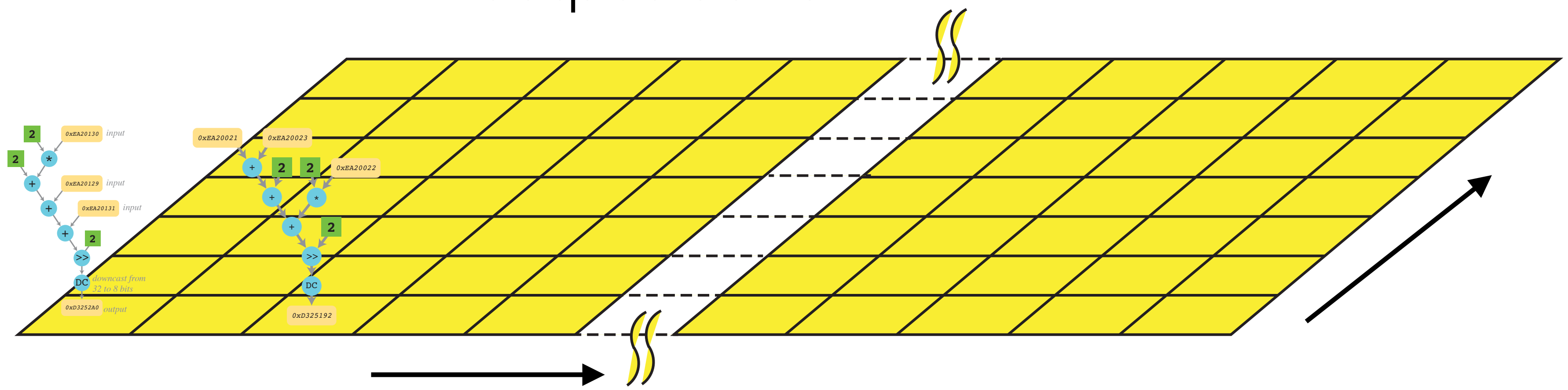
Concrete Data Dependency Trees

output buffer



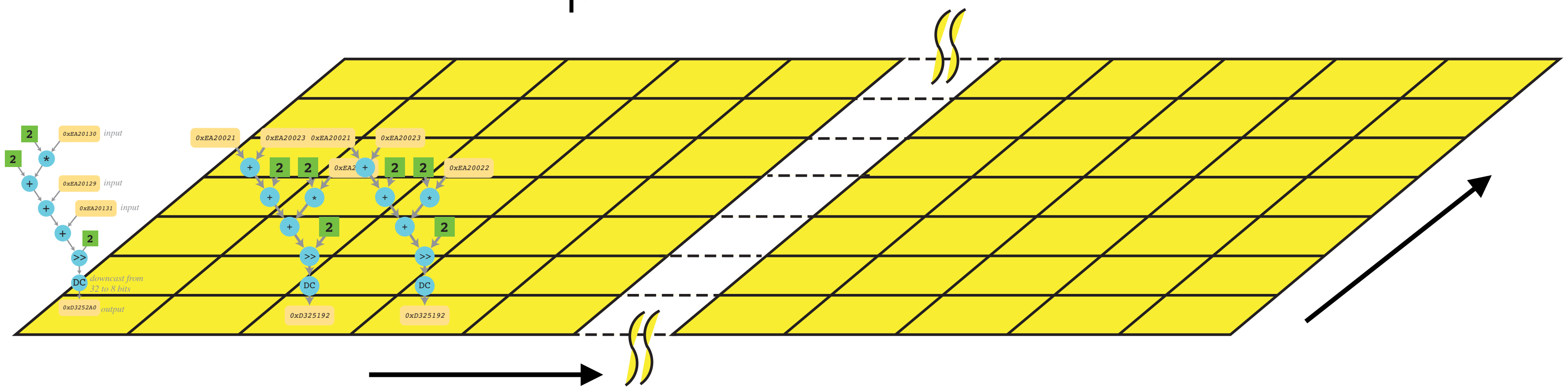
Concrete Data Dependency Trees

output buffer



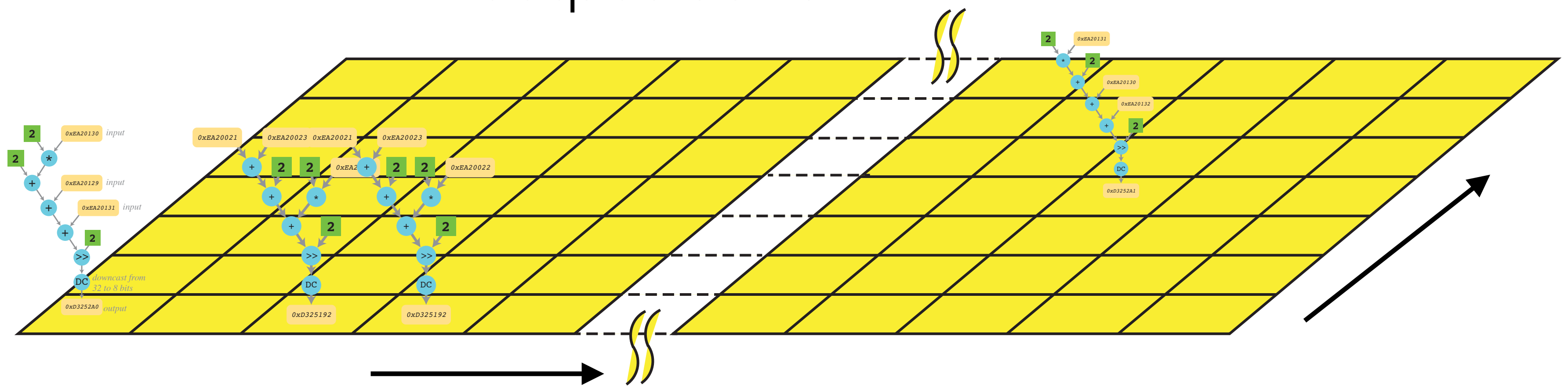
Concrete Data Dependency Trees

output buffer

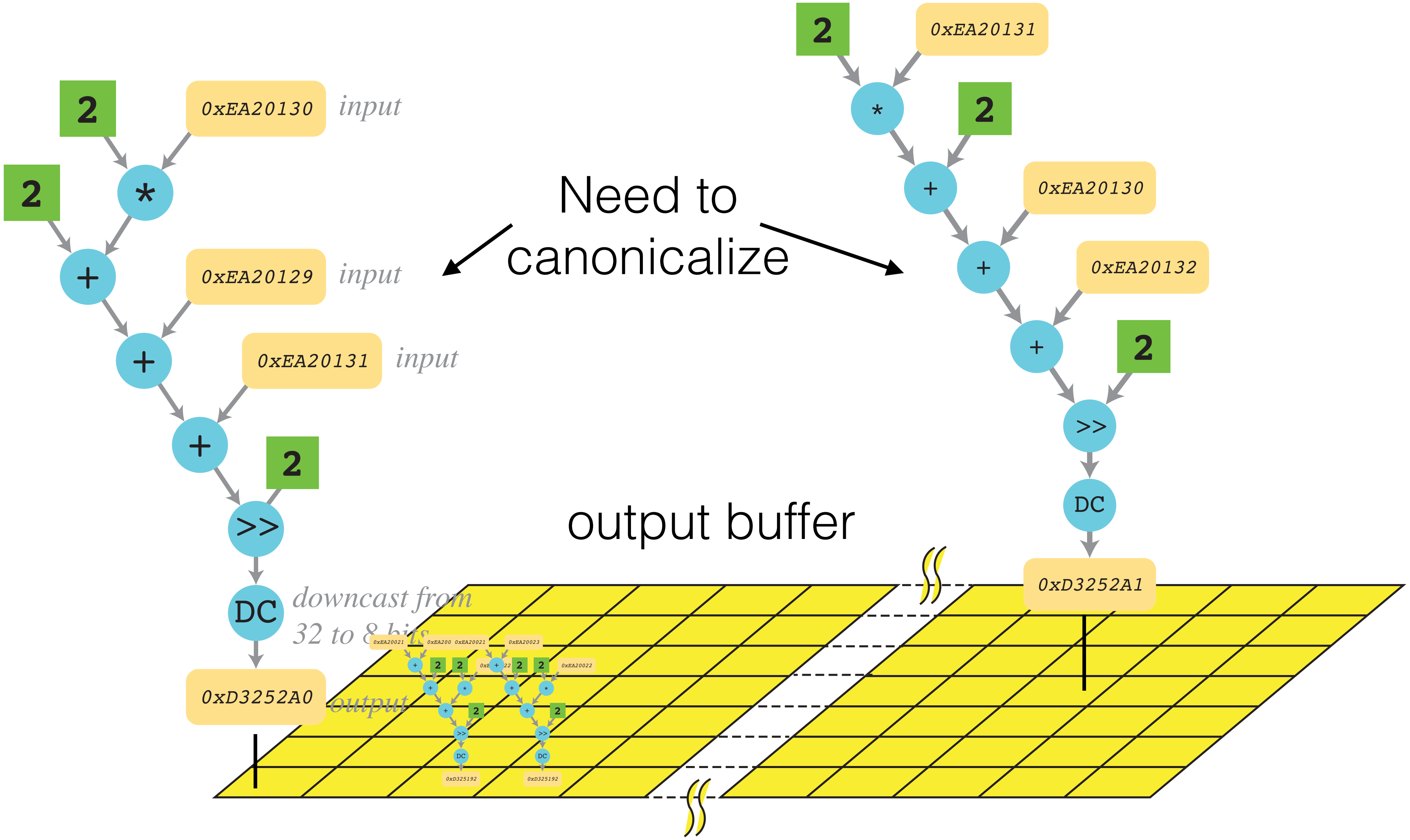


Concrete Data Dependency Trees

output buffer



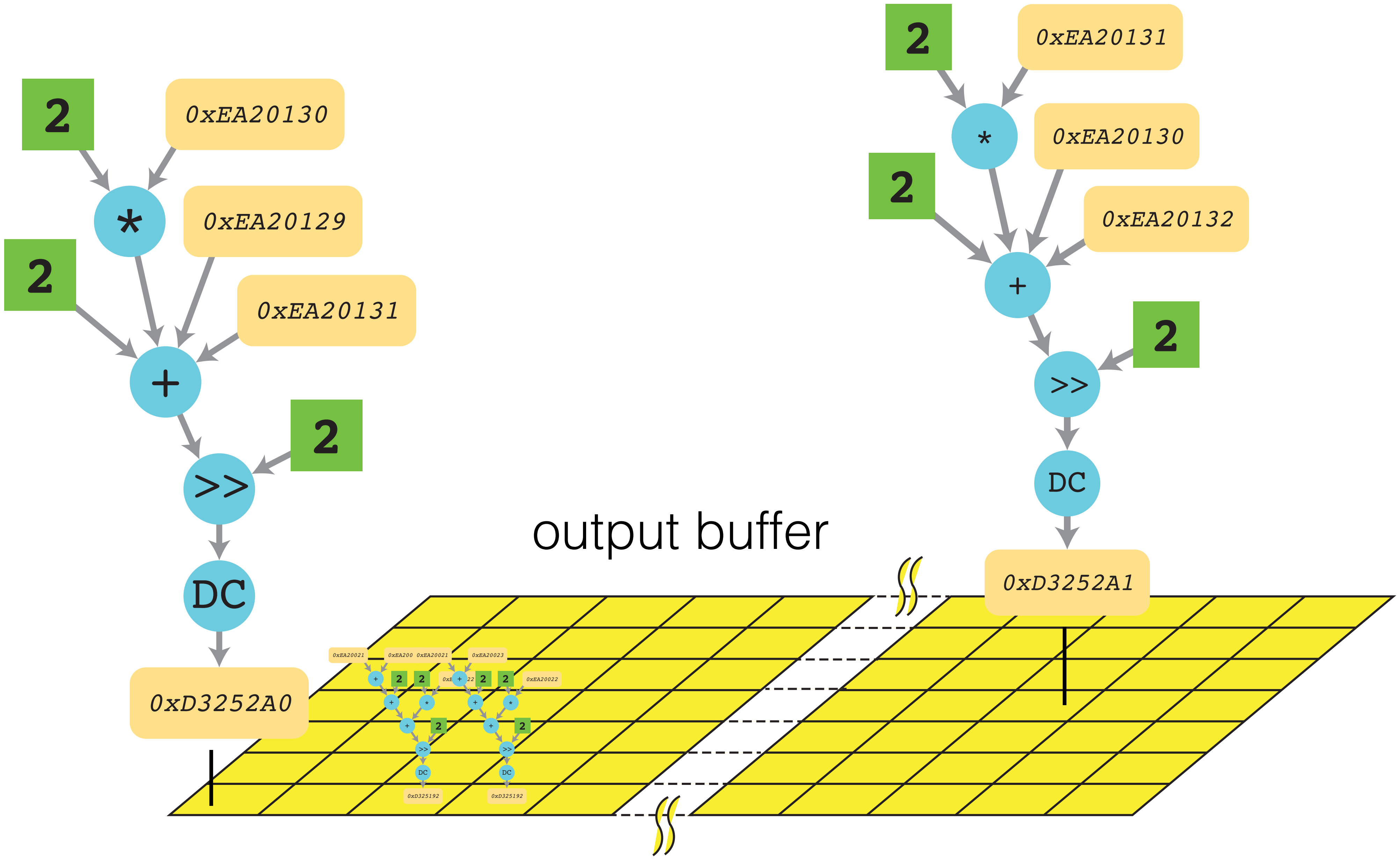
Concrete Data Dependency Trees



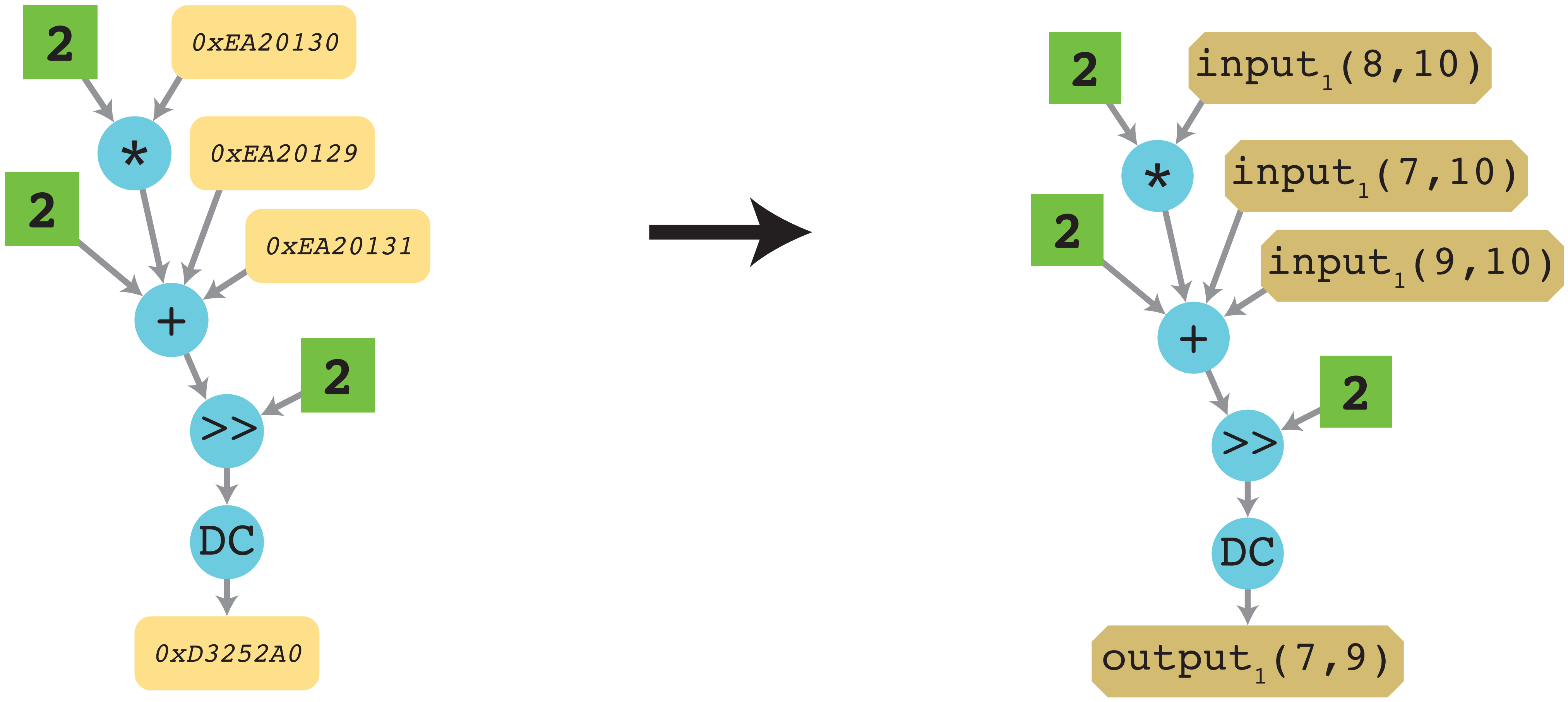
Need to canonicalize

output buffer

Canonicalization



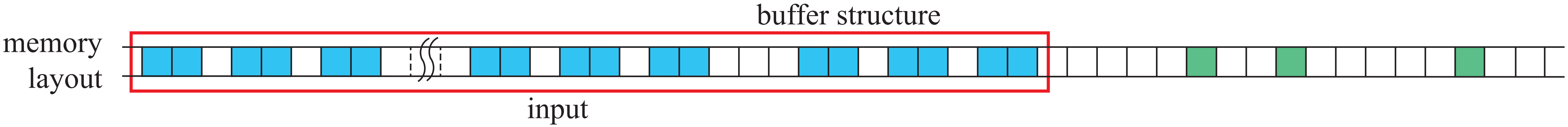
Abstract Trees

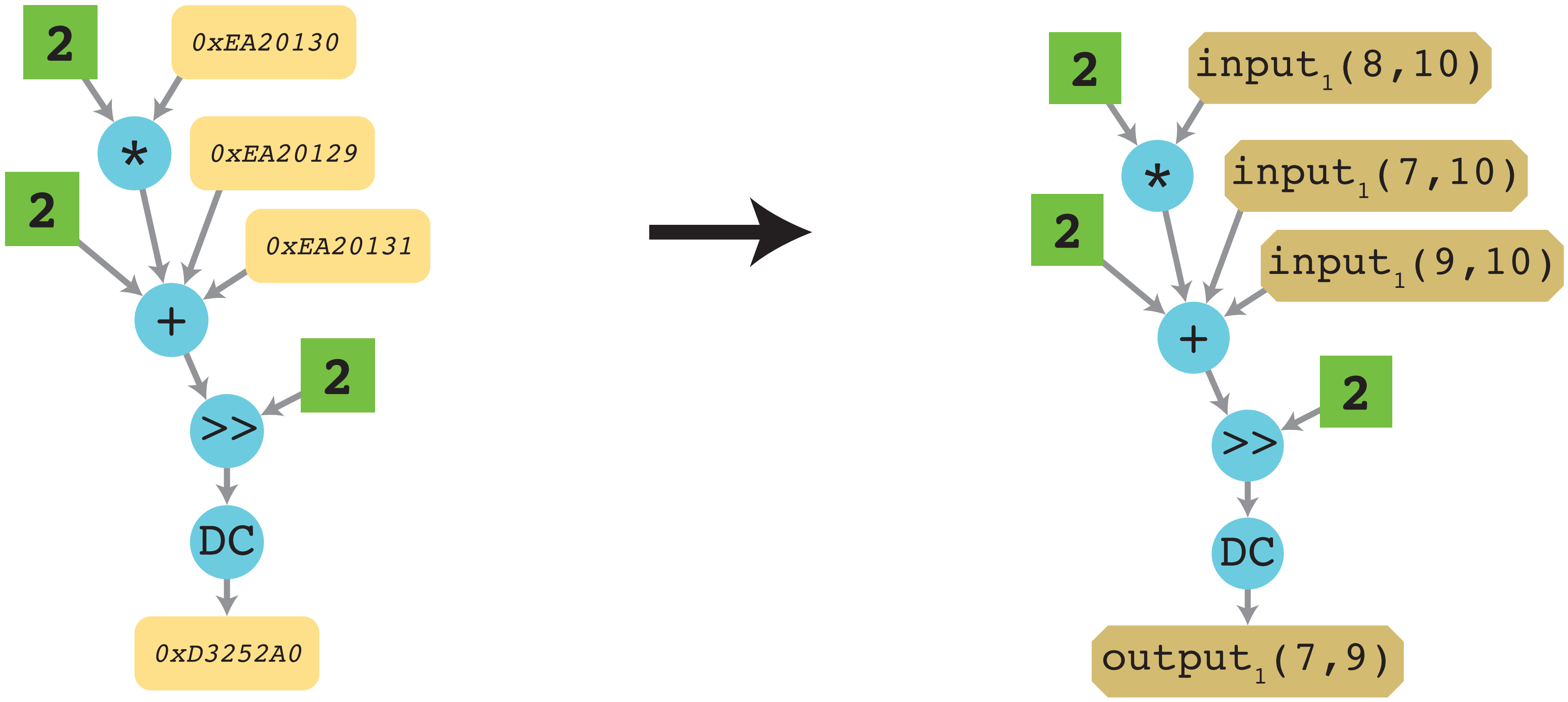

2
0xEA20130
*
0xEA20129
2
0xEA20131
+
2
>>
DC
0xD3252A0
2
$\texttt{input}_1\texttt{(8,10)}$
*
$\texttt{input}_1\texttt{(7,10)}$
2
$\texttt{input}_1\texttt{(9,10)}$
+
2
>>
DC
$\texttt{output}_1\texttt{(7,9)}$

concrete tree

abstract tree

Abstract Trees

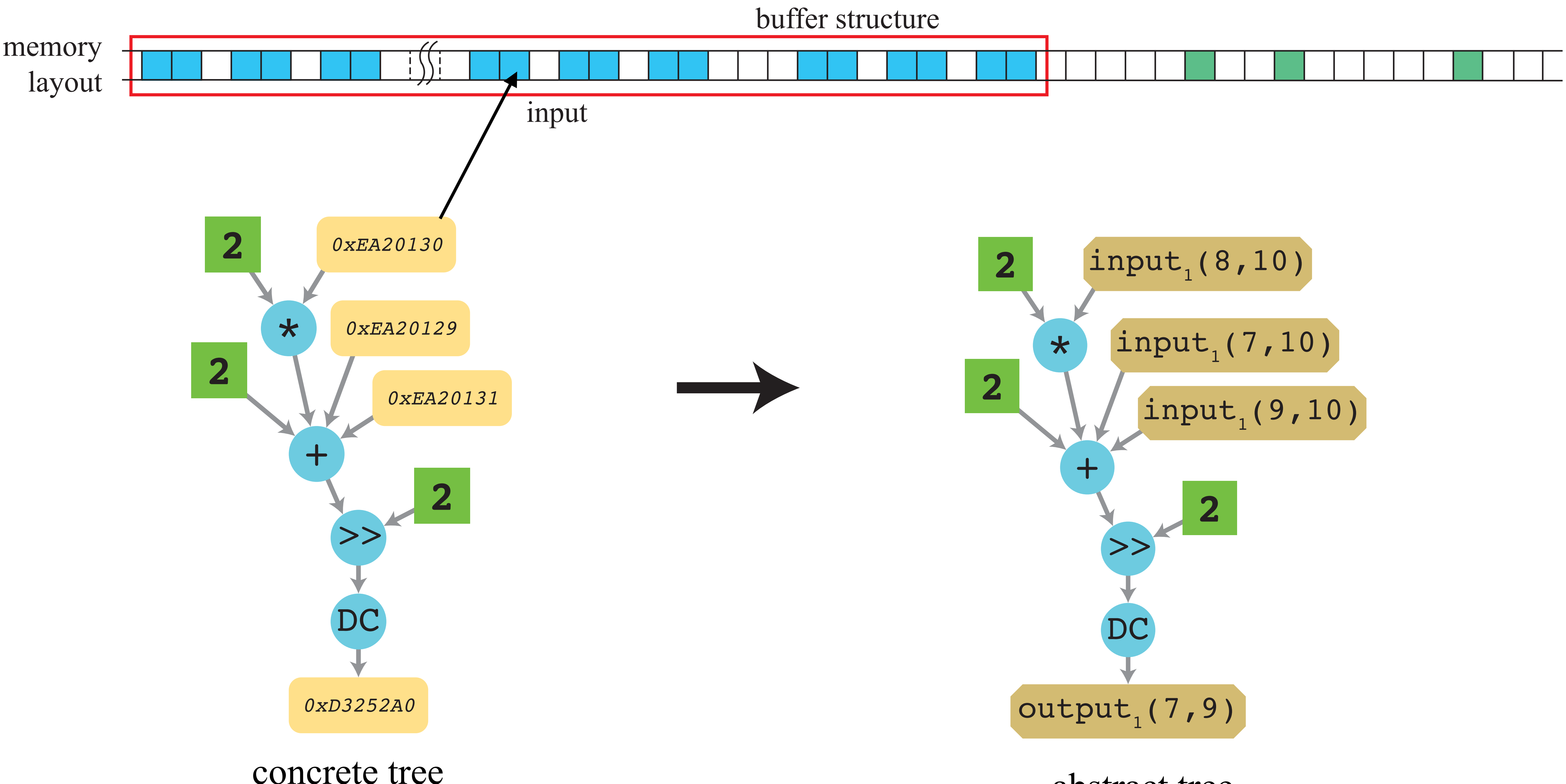




concrete tree

abstract tree

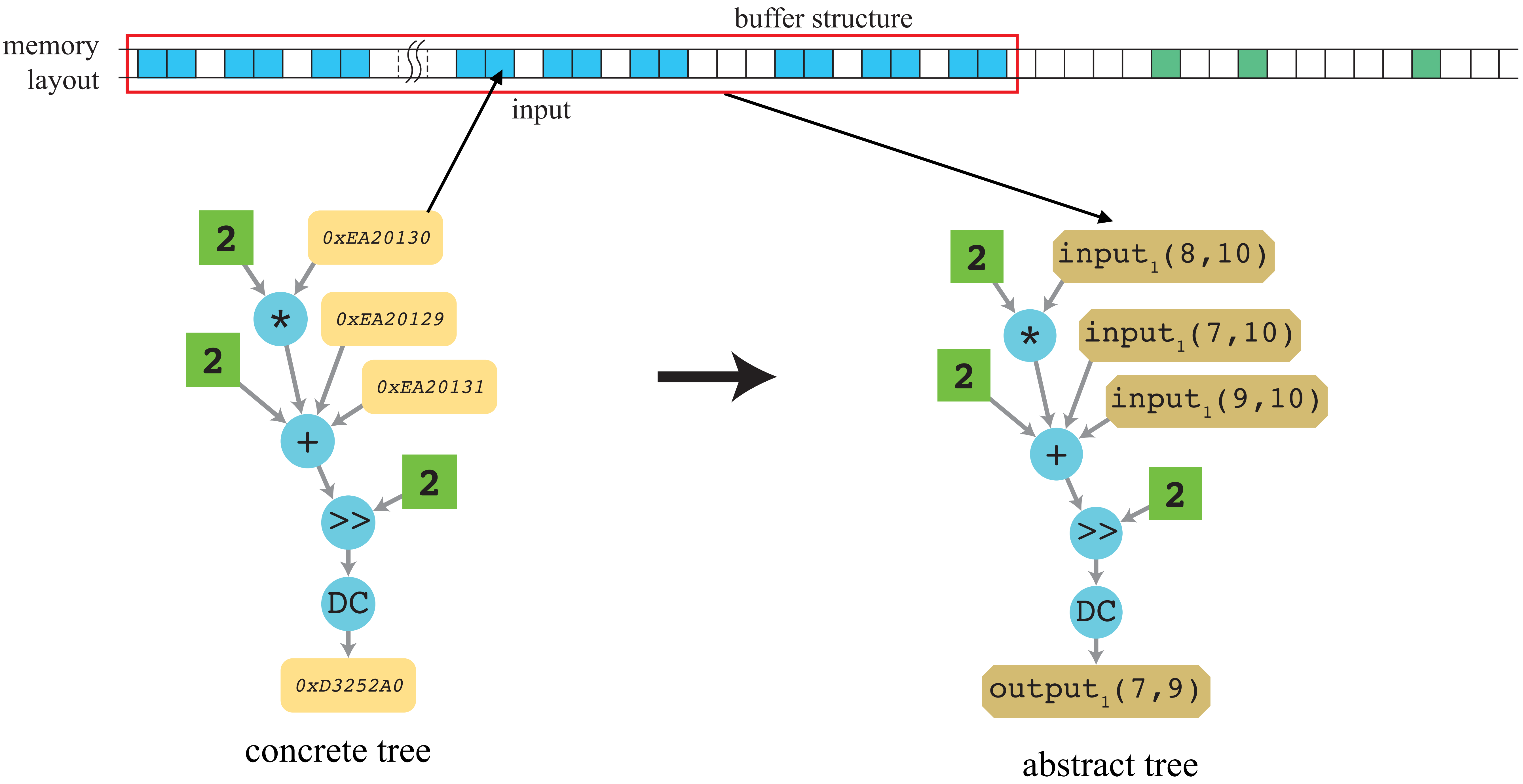
Abstract Trees



concrete tree

abstract tree

Abstract Trees



Symbolic Expressions

Stencil

Symbolic Expressions

Stencil

$$\mathrm{output}[i_1][i_2]....[i_D]$$

Symbolic Expressions

Stencil

$$\text{output}[i_1][i_2]....[i_D] \quad = \quad \begin{array}{c} \text{buffer}_1[f_{1,1}(i_1,...,i_D)]....[f_{1,D}(i_1,..,i_D)] \\ \bigoplus.....\bigoplus \\ \text{buffer}_n[f_{n,1}(i_1,...,i_D)]....[f_{n,D}(i_1,..,i_D)] \end{array}$$

Symbolic Expressions

Stencil

$$\text{output}[i_1][i_2]....[i_D] = \begin{array}{c} \text{buffer}_1[f_{1,1}(i_1, ..., i_D)]....[f_{1,D}(i_1, .., i_D)] \\ \bigoplus.....\bigoplus \\ \text{buffer}_n[\boxed{f_{n,1}(i_1, ..., i_D)}]....[f_{n,D}(i_1, .., i_D)] \end{array}$$

Symbolic Expressions

Stencil

$$\text{output}[i_1][i_2]....[i_D] = \begin{array}{c} \text{buffer}_1[f_{1,1}(i_1, ..., i_D)]....[f_{1,D}(i_1, .., i_D)] \\ \bigoplus .....\bigoplus \\ \text{buffer}_n[\boxed{f_{n,1}(i_1, ..., i_D)}]....[f_{n,D}(i_1, .., i_D)] \end{array}$$

Index Function

Symbolic Expressions

Stencil

$$\text{output}[i_1][i_2]....[i_D] \quad = \quad \begin{array}{c} \text{buffer}_1[f_{1,1}(i_1, ..., i_D)]....[f_{1,D}(i_1, .., i_D)] \\ \bigoplus .....\bigoplus \\ \text{buffer}_n[\boxed{f_{n,1}(i_1, ..., i_D)}]....[f_{n,D}(i_1, .., i_D)] \end{array}$$

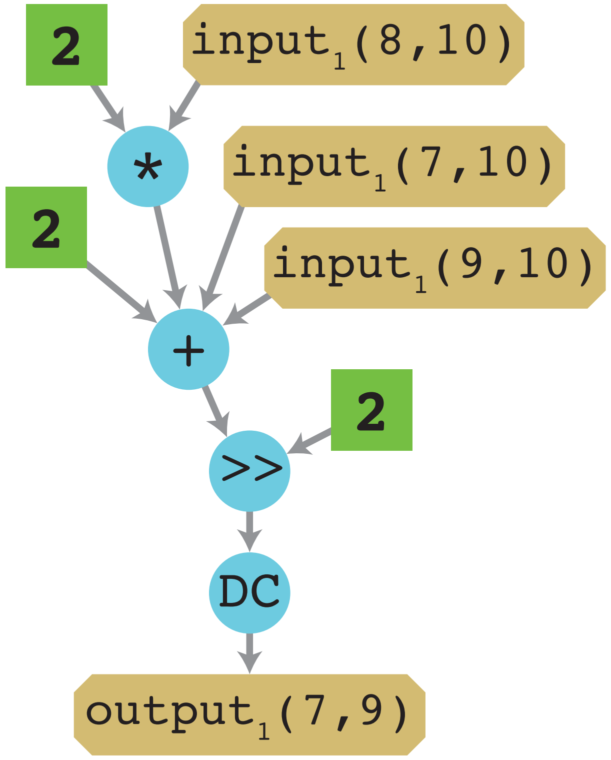
Index Function

$$f_{l,d}(i_1, ..., i_D) = a_1 \cdot i_1 + a_2 \cdot i_2 + ... + a_D \cdot i_D + a_{D+1}$$

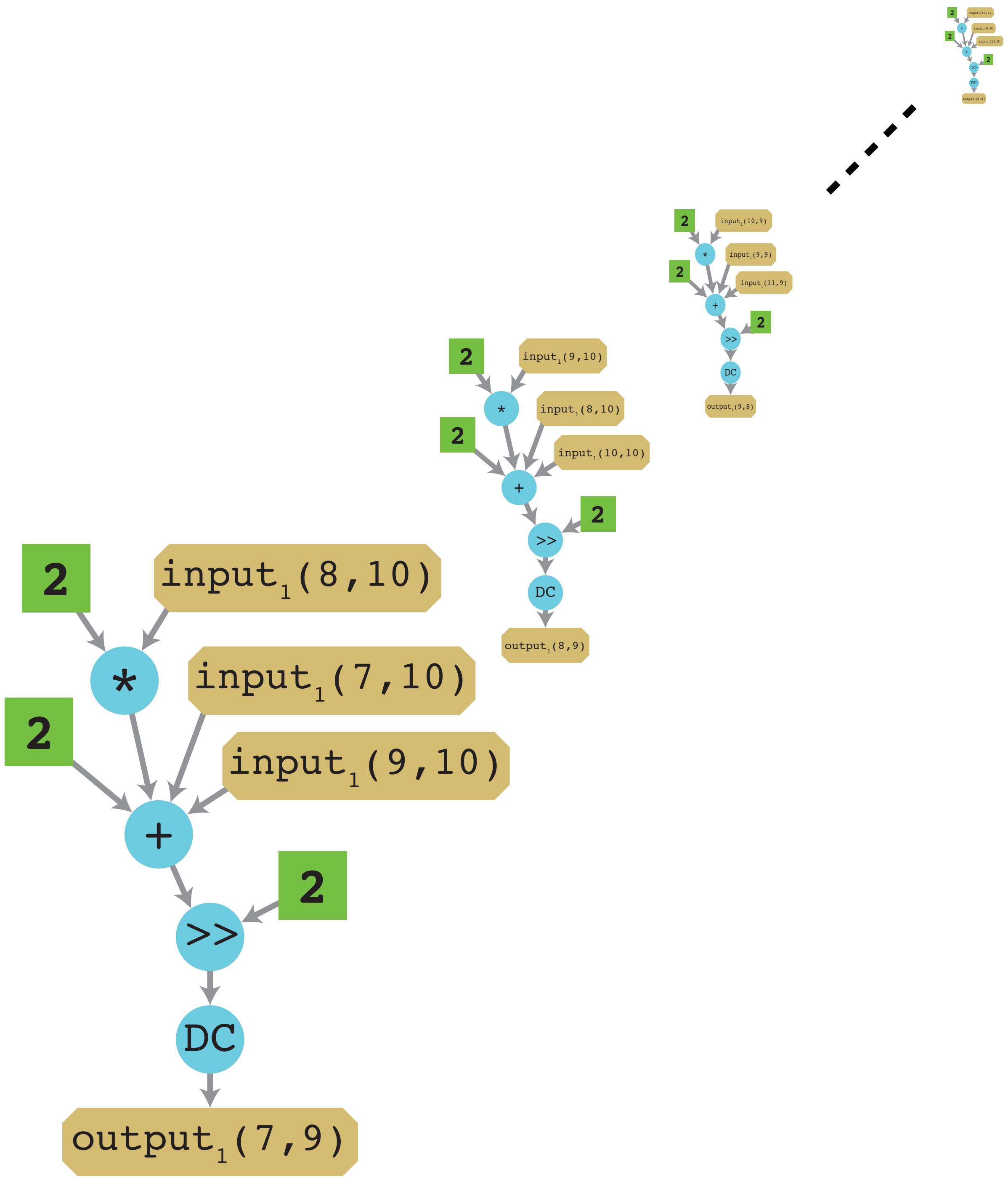
$$f_{l,d}(\vec{i}) = [\vec{i}; 1] \cdot \vec{a}$$

Affine function

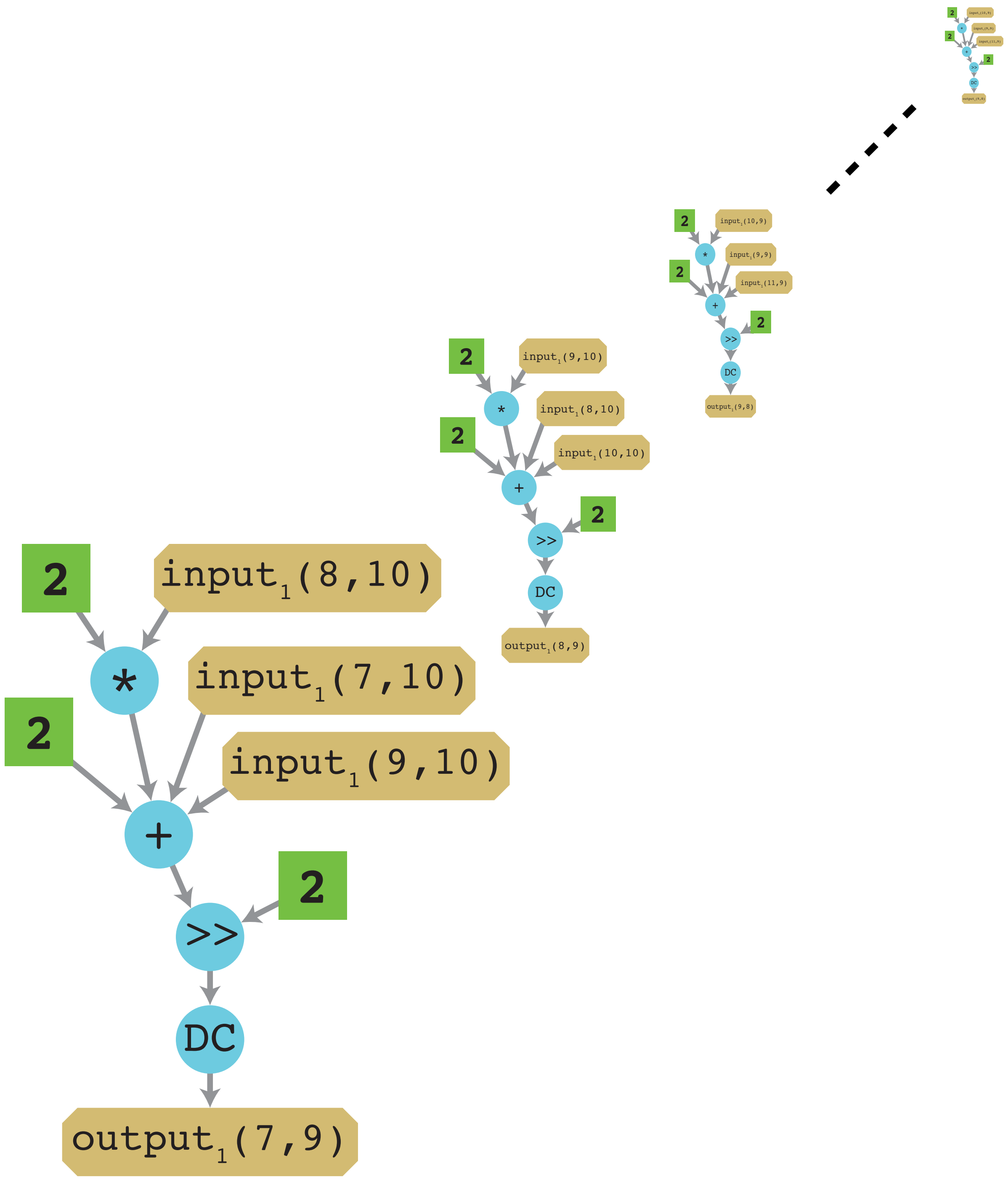
Linear Systems



Linear Systems



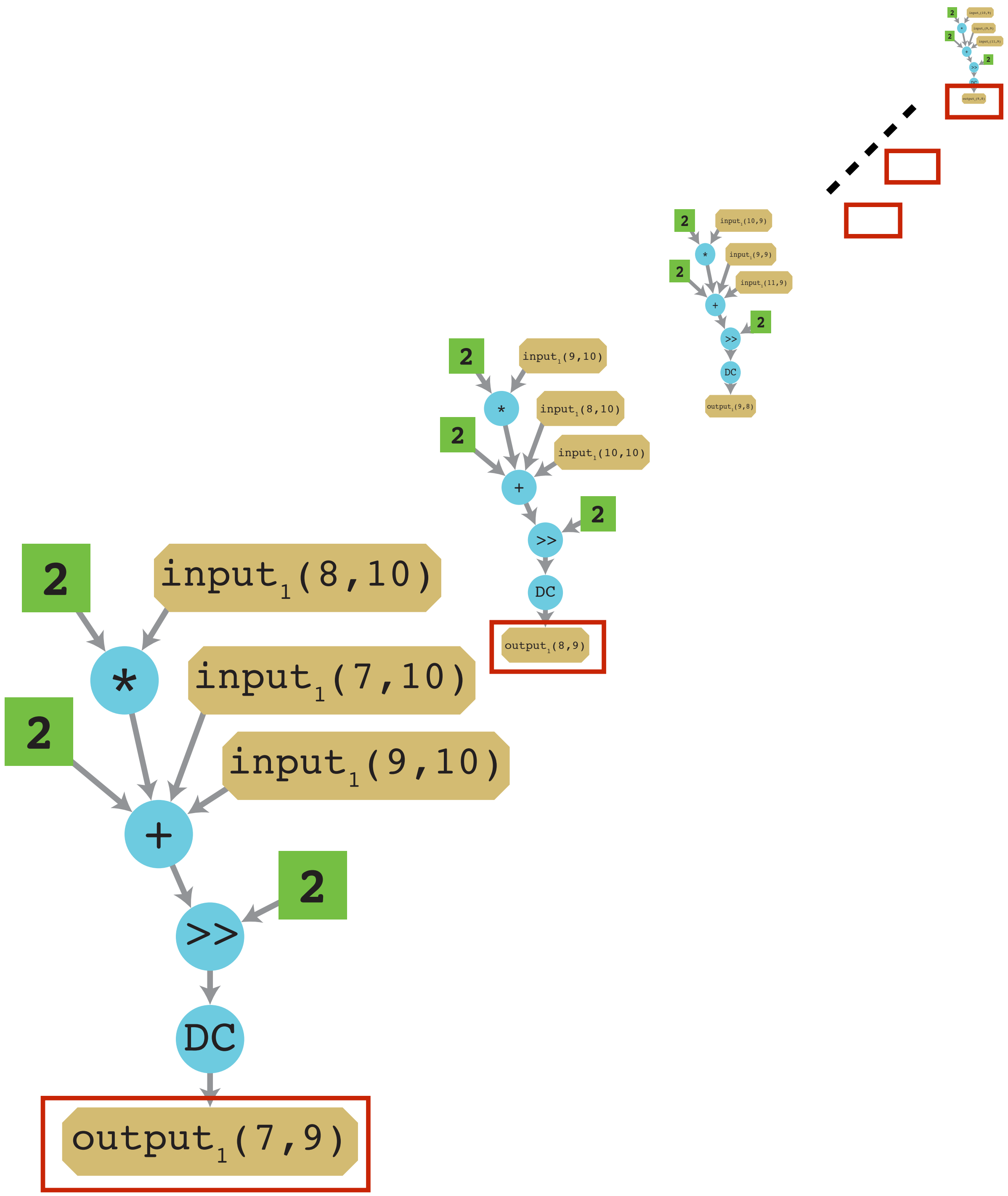
Linear Systems



$$\begin{pmatrix} 8 \\ 9 \\ 10 \\ 11 \\ 10 \end{pmatrix} = \begin{pmatrix} 7 & 9 & 1 \\ 8 & 9 & 1 \\ 9 & 8 & 1 \\ 10 & 9 & 1 \\ 9 & 11 & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

System of linear equations
for the 1st dimension
of the left-most leaf node

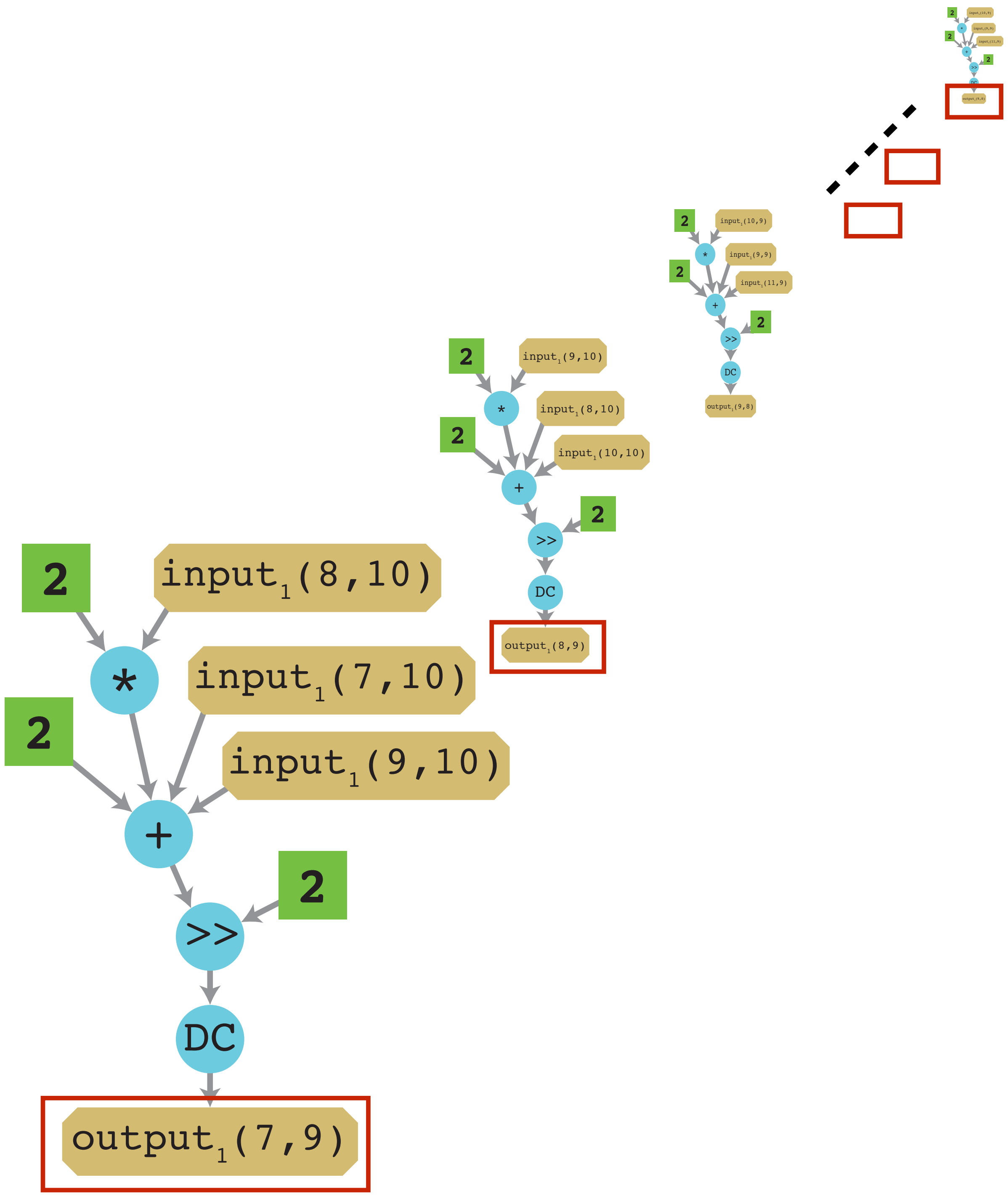
Linear Systems



$$\begin{pmatrix} 8 \\ 9 \\ 10 \\ 11 \\ 10 \end{pmatrix} = \begin{pmatrix} 7 & 9 & 1 \\ 8 & 9 & 1 \\ 9 & 8 & 1 \\ 10 & 9 & 1 \\ 9 & 11 & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

System of linear equations
for the 1st dimension
of the left-most leaf node

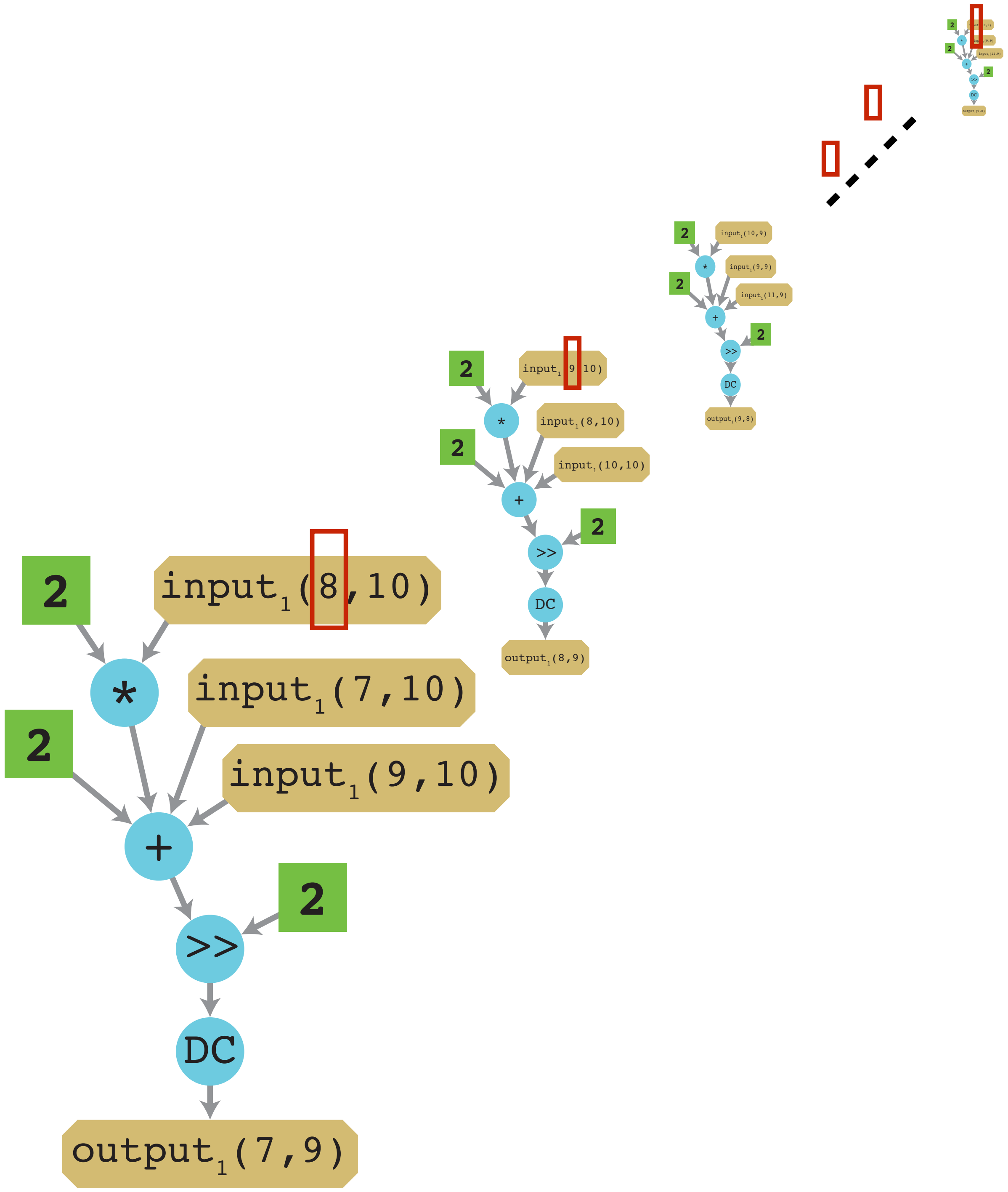
Linear Systems



$$\begin{pmatrix} 8 \\ 9 \\ 10 \\ 11 \\ 10 \end{pmatrix} = \begin{pmatrix} \boxed{\begin{matrix} 7 & 9 & 1 \\ 8 & 9 & 1 \\ 9 & 8 & 1 \\ 10 & 9 & 1 \\ 9 & 11 & 1 \end{matrix}} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

System of linear equations
for the 1st dimension
of the left-most leaf node

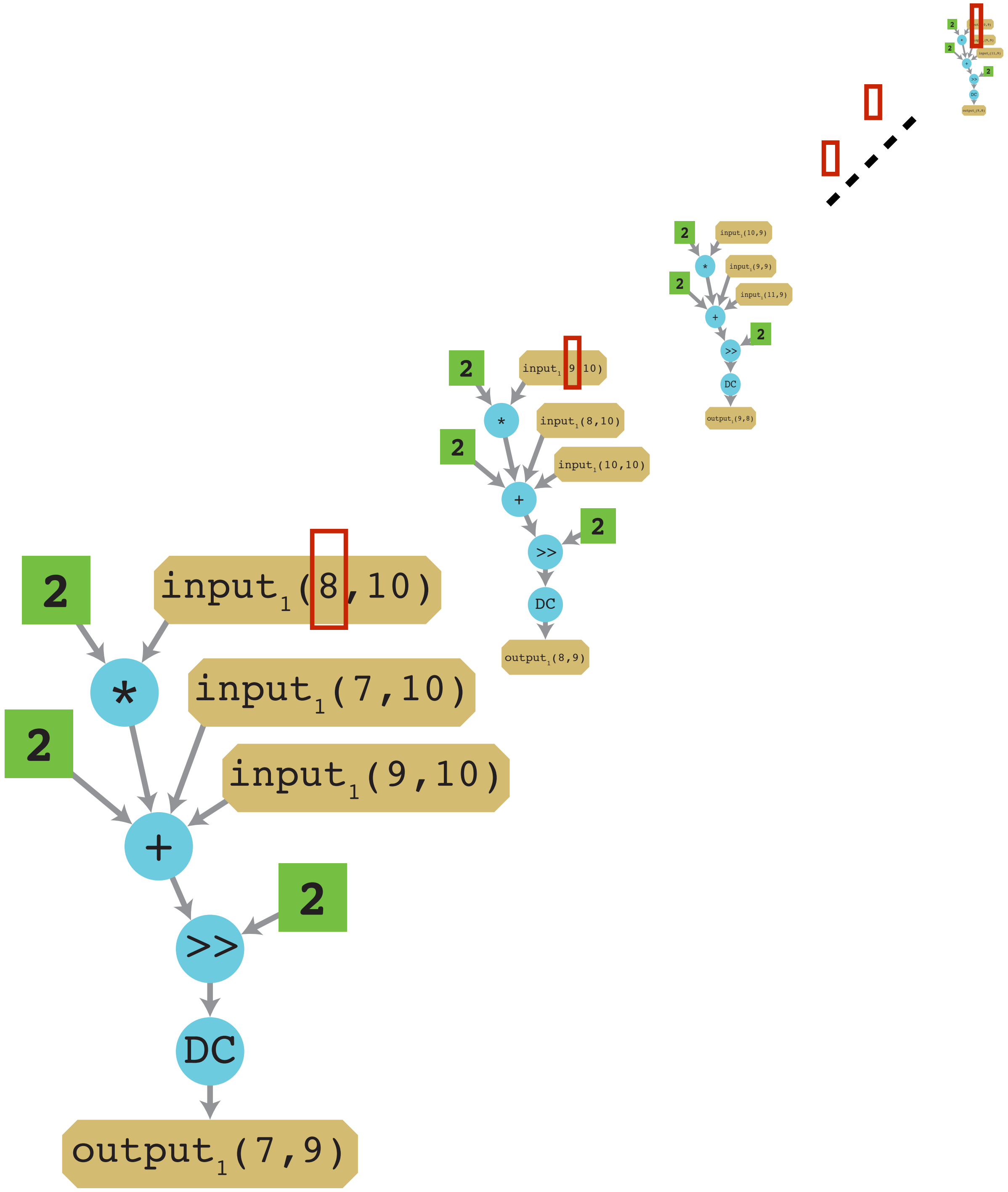
Linear Systems



$$\begin{pmatrix}8\\9\\10\\11\\10\end{pmatrix} = \begin{pmatrix}7&9&1\\8&9&1\\9&8&1\\10&9&1\\9&11&1\end{pmatrix}\begin{pmatrix}a_1\\a_2\\a_3\end{pmatrix}$$

System of linear equations
for the 1st dimension
of the left-most leaf node

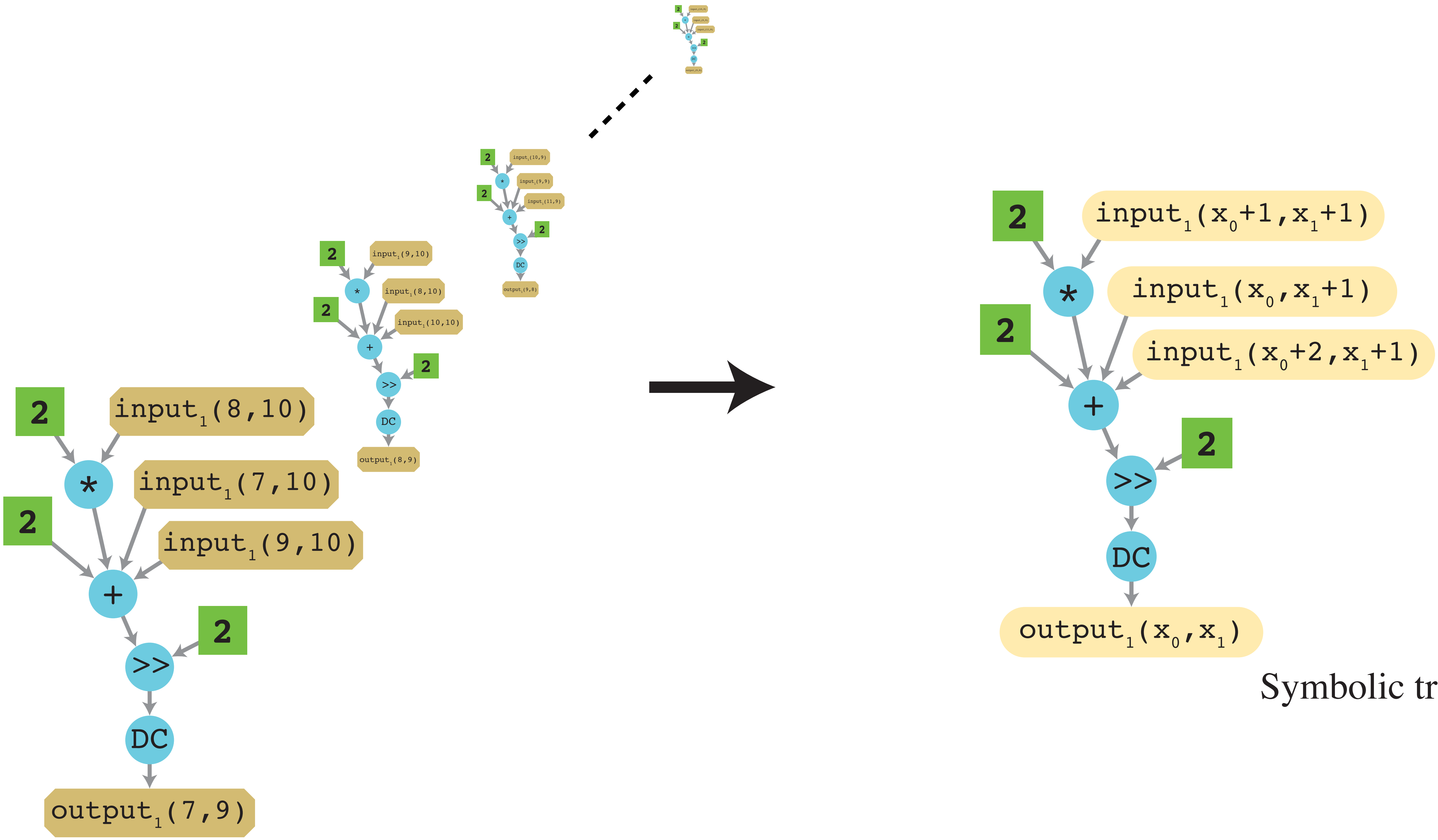
Linear Systems



$$\boxed{\begin{pmatrix} 8 \\ 9 \\ 10 \\ 11 \\ 10 \end{pmatrix}} = \begin{pmatrix} 7 & 9 & 1 \\ 8 & 9 & 1 \\ 9 & 8 & 1 \\ 10 & 9 & 1 \\ 9 & 11 & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

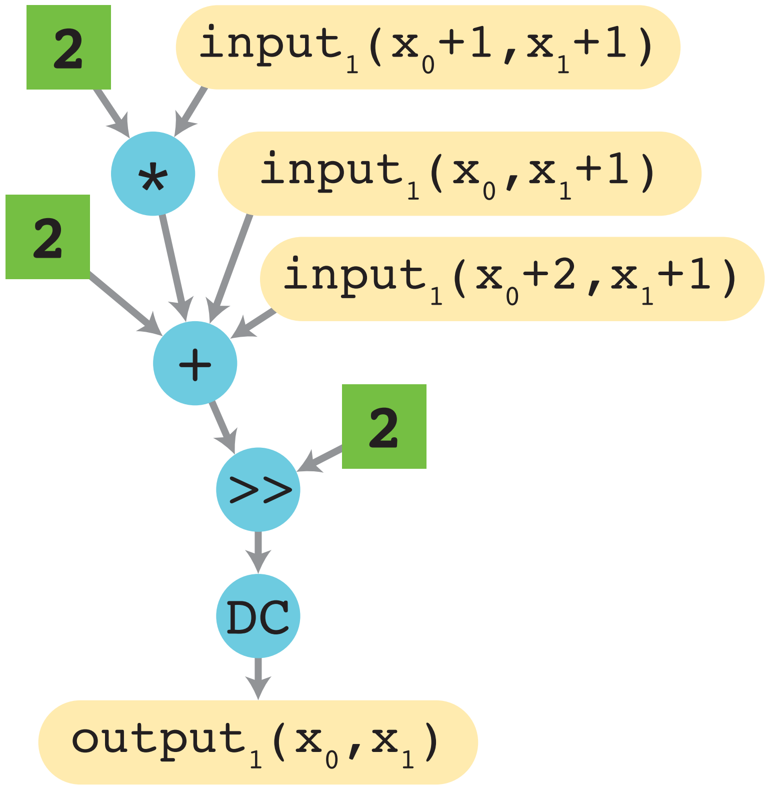
System of linear equations
for the 1st dimension
of the left-most leaf node

Linear Systems



Symbolic tree

Symbolic Tree



Symbolic tree

```
#include <Halide.h>
#include <vector>
using namespace std;
using namespace Halide;

int main(){
  Var x_0;
  Var x_1;
  ImageParam input_1(UInt(8),2);
  Func output_1;
  output_1(x_0,x_1) =
    cast<uint8_t>(((((2+
      (2*cast<uint32_t>(input_1(x_0+1,x_1+1))) +
         cast<uint32_t>(input_1(x_0,   x_1+1)) +
         cast<uint32_t>(input_1(x_0+2,x_1+1)))
      >> cast<uint32_t>(2))) & 255));
  vector<Argument> args;
  args.push_back(input_1);
  output_1.compile_to_file("halide_out_0",args);
  return 0;
}
```

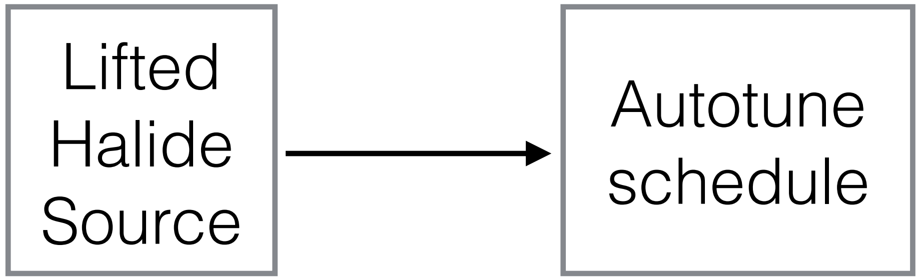
Generated Halide DSL code

Evaluation

- Successfully lifted
  - 11 Photoshop filters - 7 fully, 4 partially
  - 4 IrfanView filters
  - Smooth kernel from miniGMG

- Bit identical results on a suite of inputs

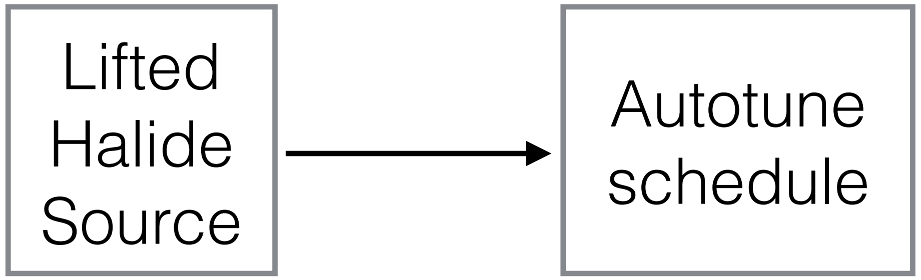
Performance Results



Lifted Halide Source

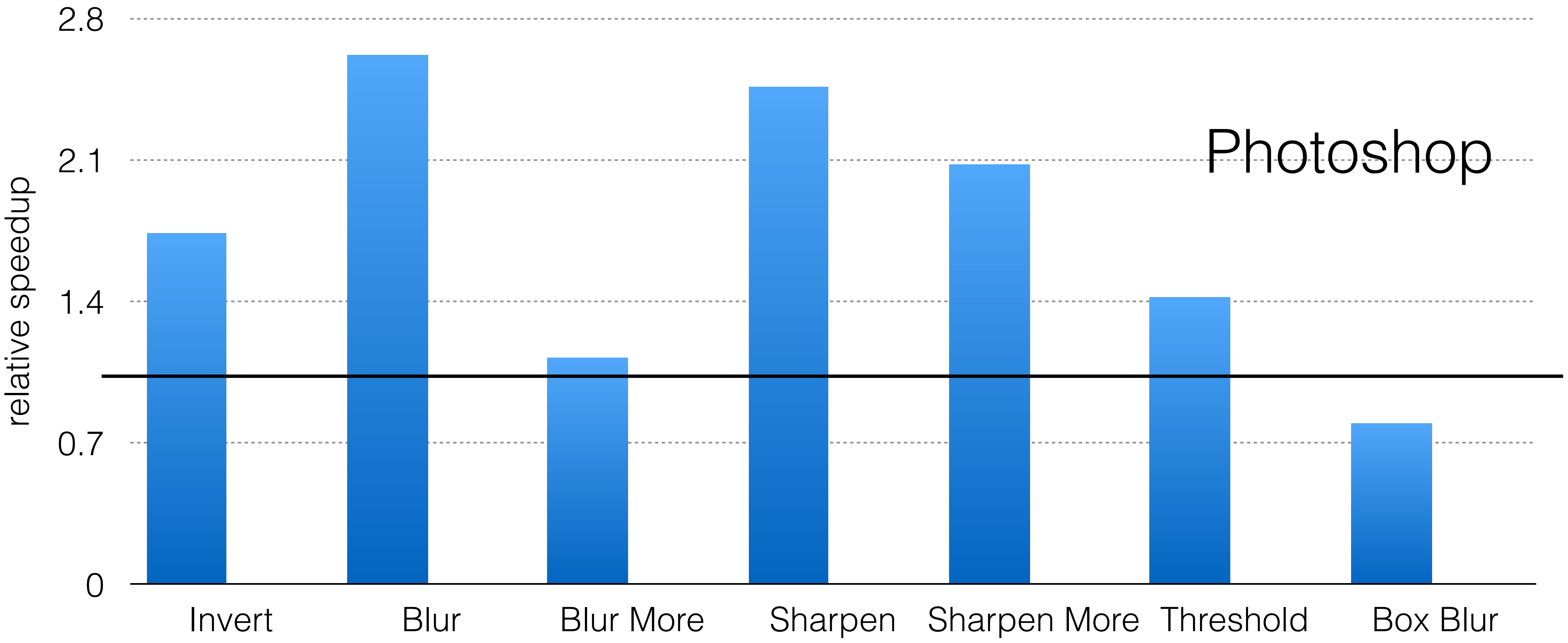
Autotune schedule

Performance Results

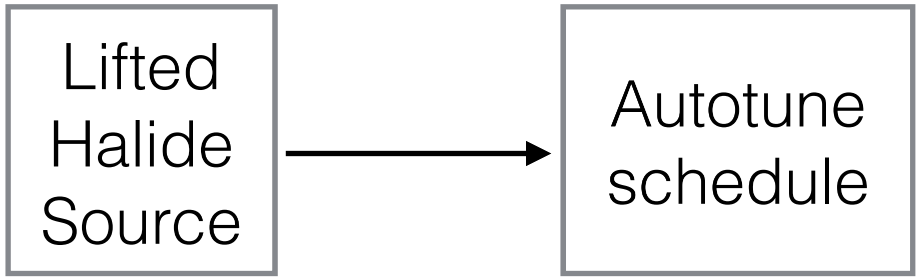


Lifted Halide Source

Autotune schedule



Performance Results



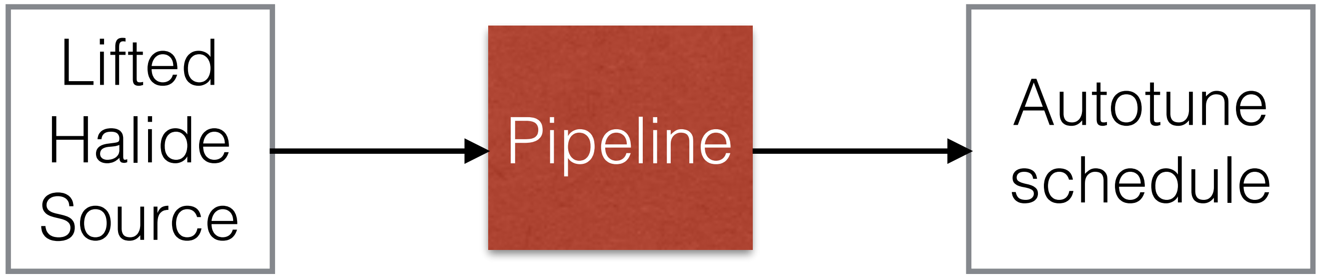
Lifted Halide Source

Autotune schedule

Avg. speed ups

- Photoshop -1.75x
- IrfanView - 4.97x
- miniGMG - 4.25x

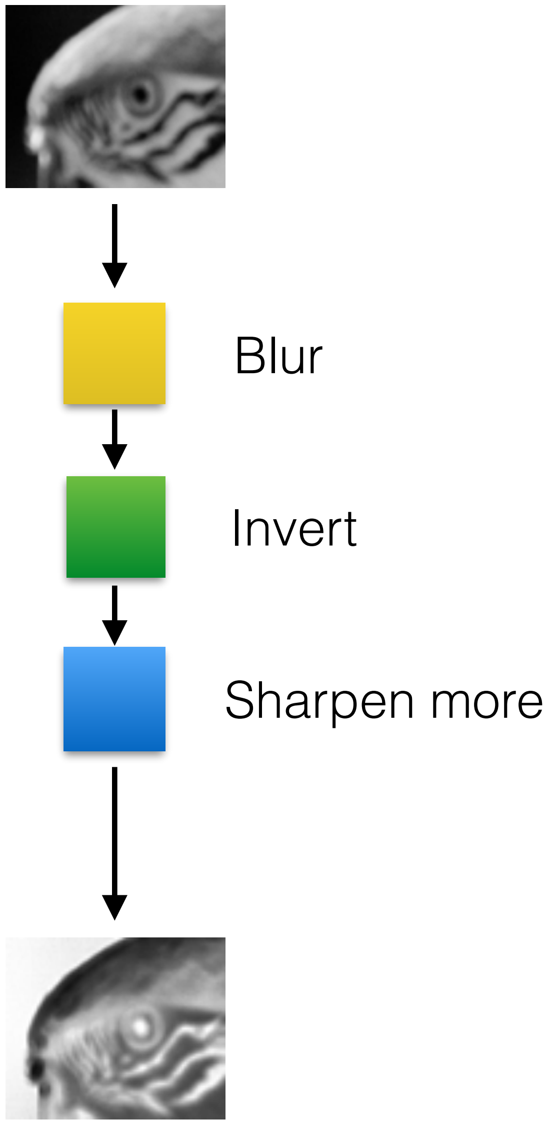
Pipeline



Lifted Halide Source

Pipeline

Autotune schedule

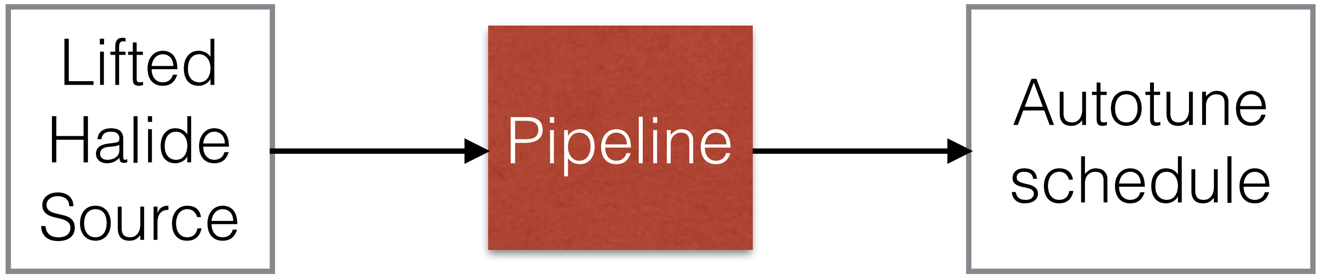


Blur

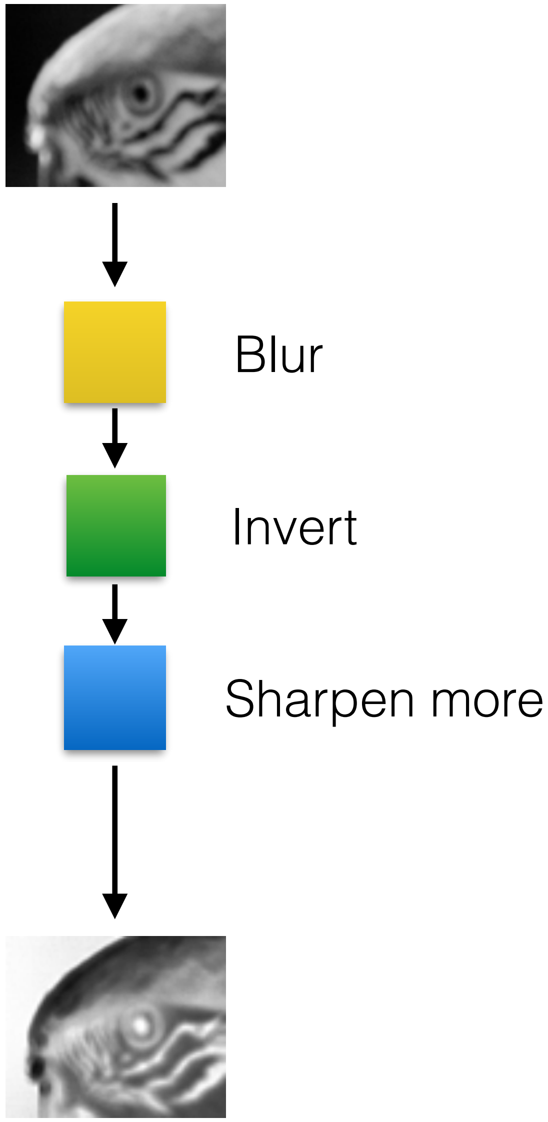
Invert

Sharpen more

Pipeline



Lifted Halide Source → Pipeline → Autotune schedule

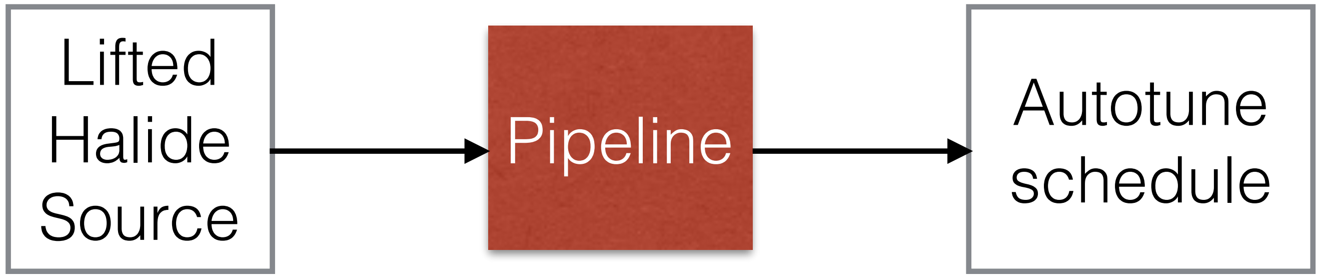


Blur

Invert

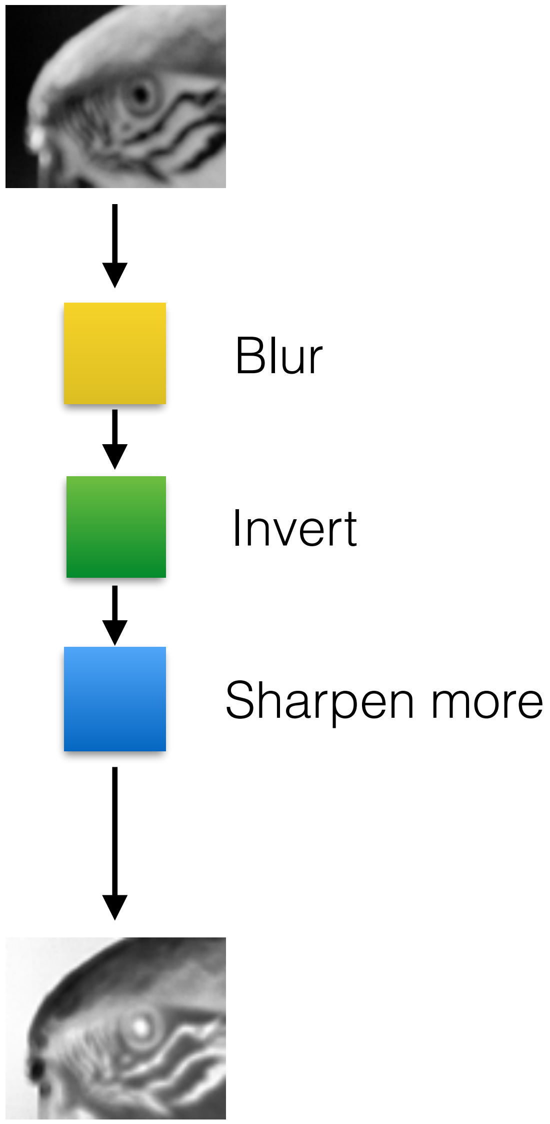
Sharpen more

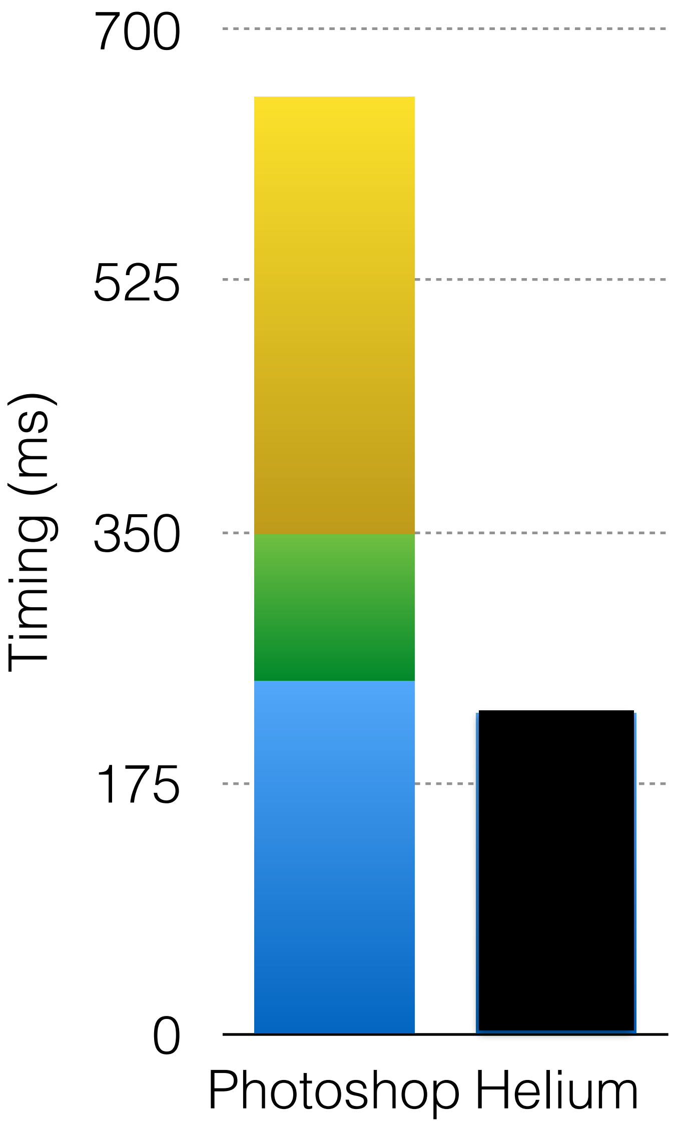
Pipeline


Lifted Halide Source

Pipeline

Autotune schedule





Avg. speed up

- Photoshop - 2.91x
- IrfanView - 5.17x

Conclusion

- Analysis on dynamic traces is able to lift stencils from **stripped binaries** to a **high level DSL**

- Lifted stencil kernels of legacy applications can be re-optimized to achieve **program rejuvenation**

- Explore Helium - http://projects.csail.mit.edu/helium