ТИНЬКОФФ

SIMD в .NET

Обработка изображений на интринсиках

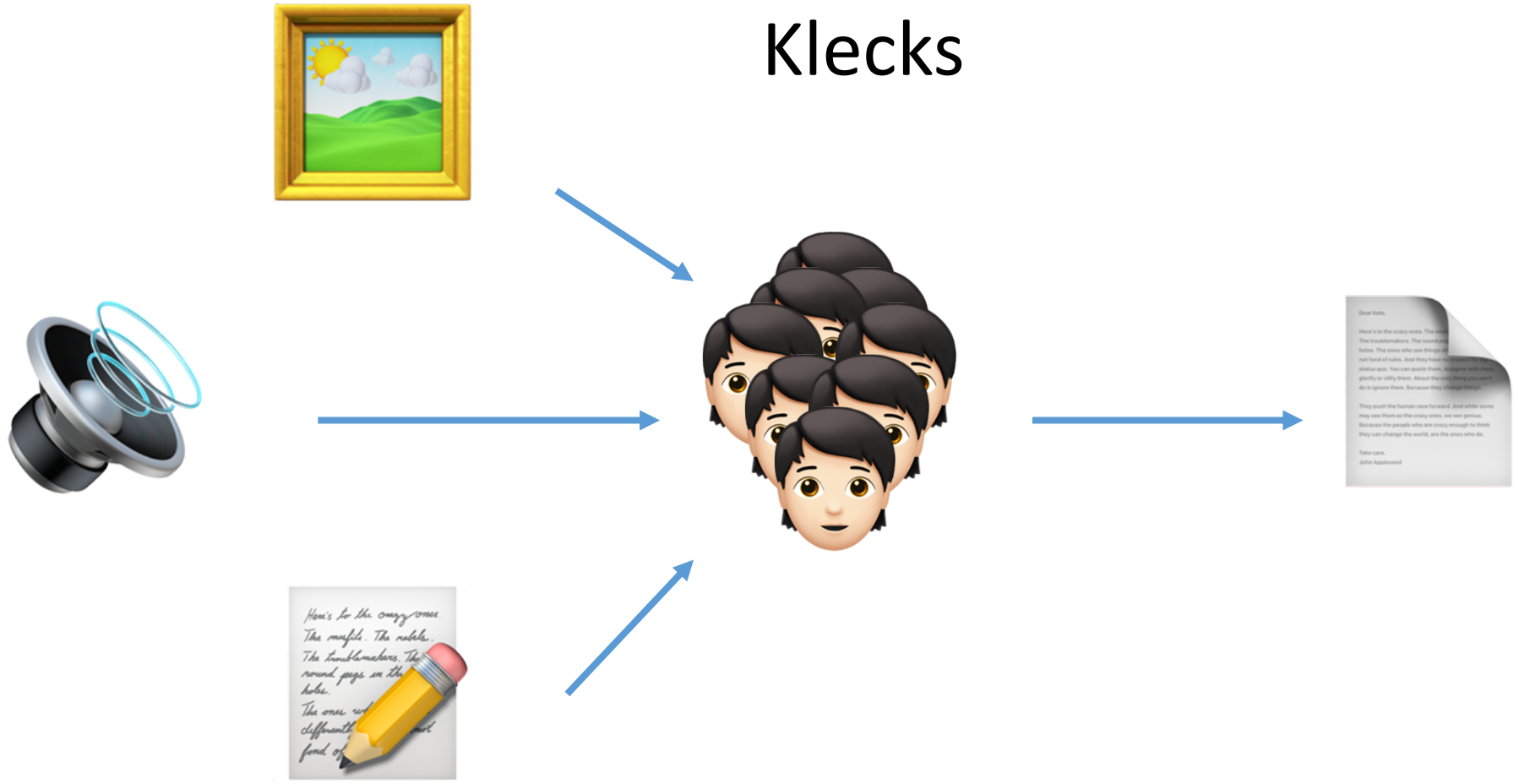
Степан Артемьев

2021

План

- Зачем нам SIMD?
- Что такое SIMD?
- Интринсики в .NET
- Hello World
- Оптимизация алгоритма Erosion
- Оптимизация размытия

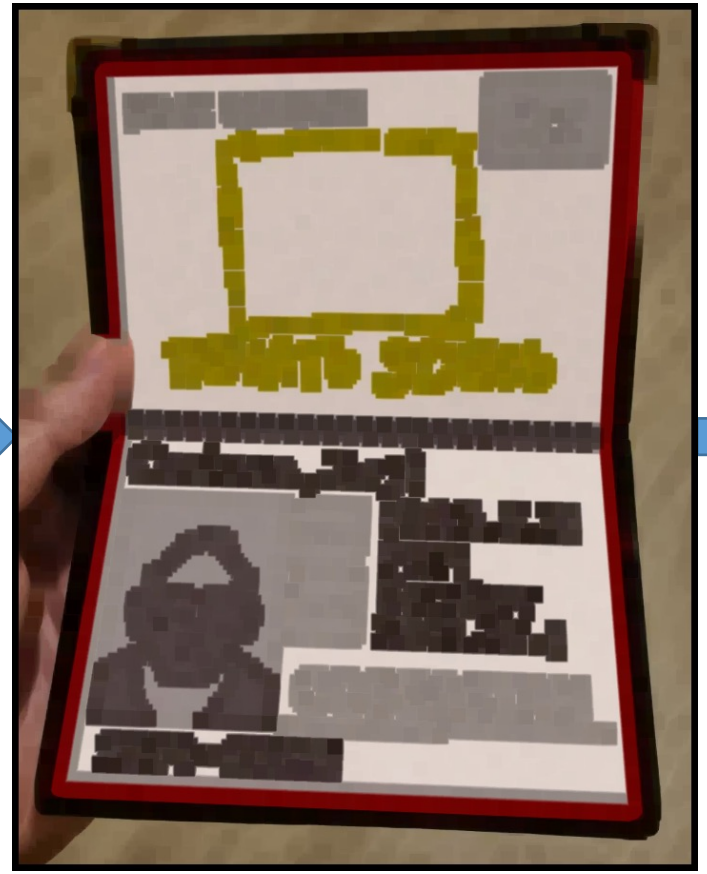
Зачем нам SIMD?



Зачем нам SIMD?

Оригинал

Erosion



Blur

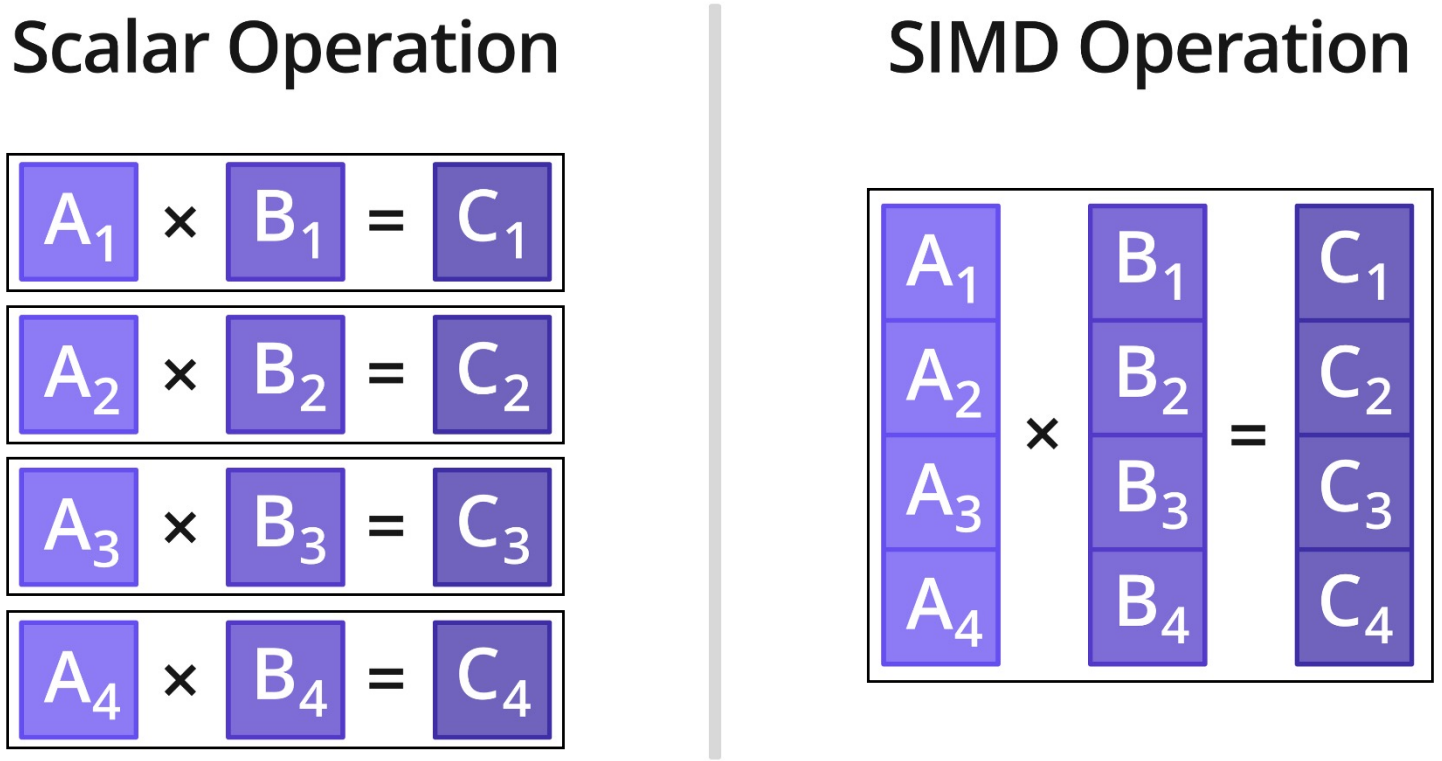
Что было раньше

https://github.com/google/skia

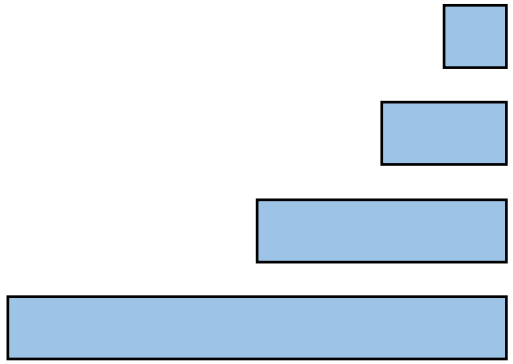
https://github.com/mono/SkiaSharp

Что такое SIMD?

Single Instruction Multiple Data



Скалярные типы в x86-64



8 бит – byte.

16 бит – short (WORD).

32 бита – int, float (DWORD).

64 бита – long, double (QWORD).

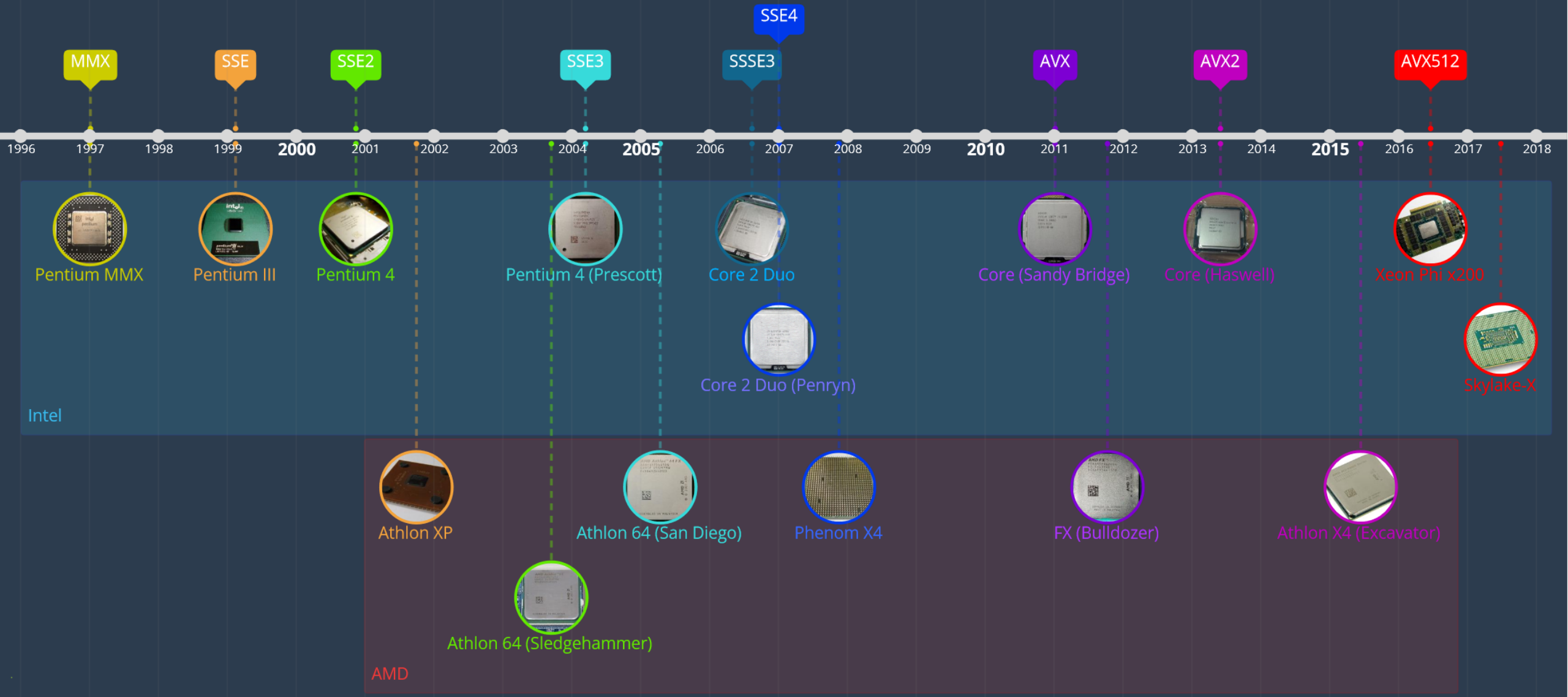
SIMD в x86-64

SSE

128 бит – Vector128, 2x long, 2x double, 4x int, 4x float, 8x short, 16x byte.

AVX

256 бит – Vector256, 4x long, 4x double, 8x int, 8x float, 16x short, 32x byte.


SSE4
MMX
SSE
SSE2
SSE3
SSSE3
AVX
AVX2
AVX512
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
Pentium MMX
Pentium III
Pentium 4
Pentium 4 (Prescott)
Core 2 Duo
Core 2 Duo (Penryn)
Core (Sandy Bridge)
Core (Haswell)
Xeon Phi x200
Skylake-X
Intel
Athlon XP
Athlon 64 (San Diego)
Phenom X4
FX (Bulldozer)
Athlon X4 (Excavator)
Athlon 64 (Sledgehammer)
AMD

Intrinsics / Интринсики

- System.Runtime.Intrinsics
- System.Runtime.Intrinsics.X86
- System.Runtime.Intrinsics.Arm
- System.Numerics

Intrinsics / Интринсики

ASM

```
/// <summary>
/// __m256i _mm256_avg_epu8 (__m256i a, __m256i b)
///   VPAVGB ymm, ymm, ymm/m256
/// </summary>
public static Vector256<byte> Average(Vector256<byte> left, Vector256<byte> right)
```

https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html

```
__m256i _mm256_avg_epu8 (__m256i a, __m256i b)
```

vpavgb

Synopsis

```
__m256i _mm256_avg_epu8 (__m256i a, __m256i b)
#include <immintrin.h>
Instruction: vpavgb ymm, ymm, ymm
CPUID Flags: AVX2
```

Description

Average packed unsigned 8-bit integers in a and b, and store the results in dst.

Operation

```
FOR j := 0 to 31
        i := j*8
        dst[i+7:i] := (a[i+7:i] + b[i+7:i] + 1) >> 1
ENDFOR
dst[MAX:256] := 0
```

Performance

Architecture	Latency	Throughput (CPI)
Icelake	1	0.5
Skylake	1	0.5
Broadwell	1	0.5
Haswell	1	0.5

```
/// <summary>
/// __m128i _mm_max_epu8 (__m128i a,  __m128i b)
///   PMAXUB xmm, xmm/m128
/// </summary>
public static Vector128<byte> Max(Vector128<byte> left, Vector128<byte> right)
/// <summary>
/// __m128i _mm_max_epi16 (__m128i a,  __m128i b)
///   PMAXSW xmm, xmm/m128
/// </summary>
public static Vector128<short> Max(Vector128<short> left, Vector128<short> right)
/// <summary>
/// __m128d _mm_max_pd (__m128d a,  __m128d b)
///   MAXPD xmm, xmm/m128
/// </summary>
public static Vector128<double> Max(Vector128<double> left, Vector128<double> right)
```

Intrinsics / Интринсики

```
public Vector256<byte> Average(Vector256<byte> a, Vector256<byte> b)
{
    return Avx2.Average(a, b);
}
```

```
C.Average(System.Runtime.Intrinsics.Vector256`1<Byte>,
    L0000: vzeroupper
    L0003: vmovupd ymm0, [esp+0x24]
    L0009: vpavgb ymm0, ymm0, [esp+4]
    L000f: vmovupd [edx], ymm0
    L0013: vzeroupper
    L0016: ret 0x40
```

Кроссплатформенность?

Hello, World!

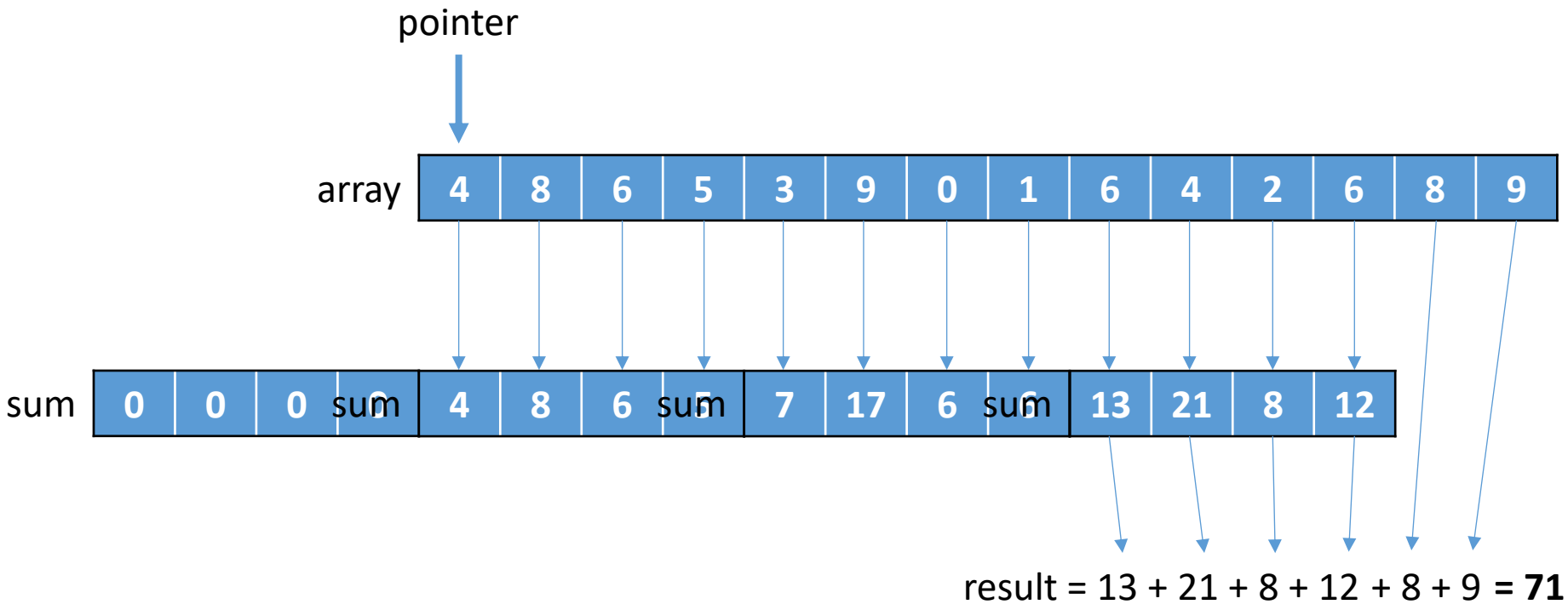
```
public static int SumLinq(int[] array)
{
    return array.Sum();
}

public static int SumForLoop(int[] array)
{
    var sum = 0;
    for (var i = 0; i < array.Length; i++)
        sum += array[i];

    return sum;
}
```

Hello, World!

SSE

128 бит – Vector128, 2x long, 2x double, **4x int**, 4x float, 8x short, 16x byte.

Hello, World!


pointer
array
4 8 6 5 3 9 0 1 6 4 2 6 8 9
sum
0 0 0 sum 4 8 6 sum 7 17 6 sum 13 21 8 12
result = 13 + 21 + 8 + 12 + 8 + 9 = 71

Hello, World!

```
public static unsafe int SumSse2(int[] array)
```

Hello, World!

```
fixed (int* pointer = array)
```

Hello, World!

```
var sum = Vector128<int>.Zero;
var lastIndex = array.Length - array.Length % 4;
```

Hello, World!

```
for (var i = 0; i < lastIndex; i += 4)
{
    sum = Sse2.Add(sum, Sse2.LoadVector128(pointer + i));
}
```

Hello, World!

```
var result =
    sum.GetElement(0) +
    sum.GetElement(1) +
    sum.GetElement(2) +
    sum.GetElement(3);
```

Hello, World!

```
for (var i = lastIndex; i < array.Length; i++)
{
    result += pointer[i];
}
```

```
public static unsafe int SumSse2(int[] array)
{
    fixed (int* pointer = array)
    {
        var sum = Vector128<int>.Zero;
        var lastIndex = array.Length - array.Length % 4;

        for (var i = 0; i < lastIndex; i += 4)
        {
            sum = Sse2.Add(sum, Sse2.LoadVector128(pointer + i));
        }

        var result = sum.GetElement(0) + sum.GetElement(1) + sum.GetElement(2) + sum.GetElement(3);

        for (var i = lastIndex; i < array.Length; i++)
        {
            result += pointer[i];
        }

        return result;
    }
}
```

```
public static unsafe int SumAvx2(int[] array)
{
    fixed (int* pointer = array)
    {
        var sum = Vector256<int>.Zero;
        var lastIndex = array.Length - array.Length % 8;

        for (var i = 0; i < lastIndex; i += 8)
        {
            sum = Avx2.Add(sum, Avx.LoadVector256(pointer + i));
        }

        var result = sum.GetElement(0) + sum.GetElement(1) + sum.GetElement(2) + sum.GetElement(3)
                   + sum.GetElement(4) + sum.GetElement(5) + sum.GetElement(6) + sum.GetElement(7);

        for (var i = lastIndex; i < array.Length; i++)
        {
            result += pointer[i];
        }

        return result;
    }
}
```

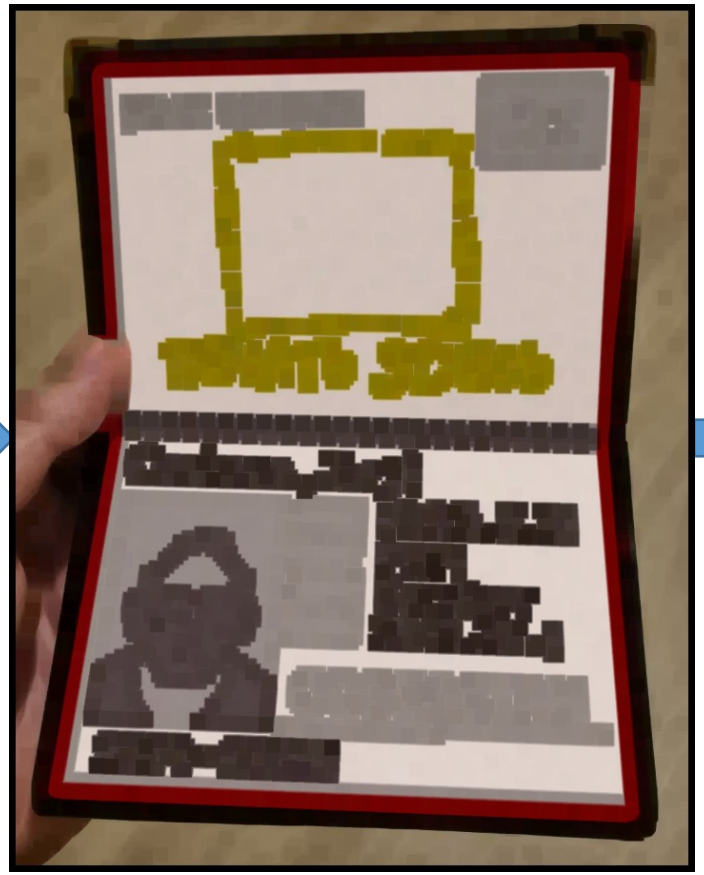
```
public static int Sum(int[] array)
{
    if (Avx2.IsSupported)
    {
        return SumAvx2(array);
    }
    else if (Sse2.IsSupported)
    {
        return SumSse2(array);
    }
    else
    {
        return SumForLoop(array);
    }
}
```

Hello, World!

Method	Mean	Error	StdDev	Ratio	Improve
Sum_Linq	2,905.27 µs	16.569 µs	15.499 µs	1.00	1.00
Sum_ForLoop	216.64 µs	1.528 µs	1.355 µs	0.07	13.41
Sum_Sse2	109.24 µs	0.757 µs	0.708 µs	0.04	26.60
Sum_Avx2	55.83 µs	0.539 µs	0.478 µs	0.02	**52.06**

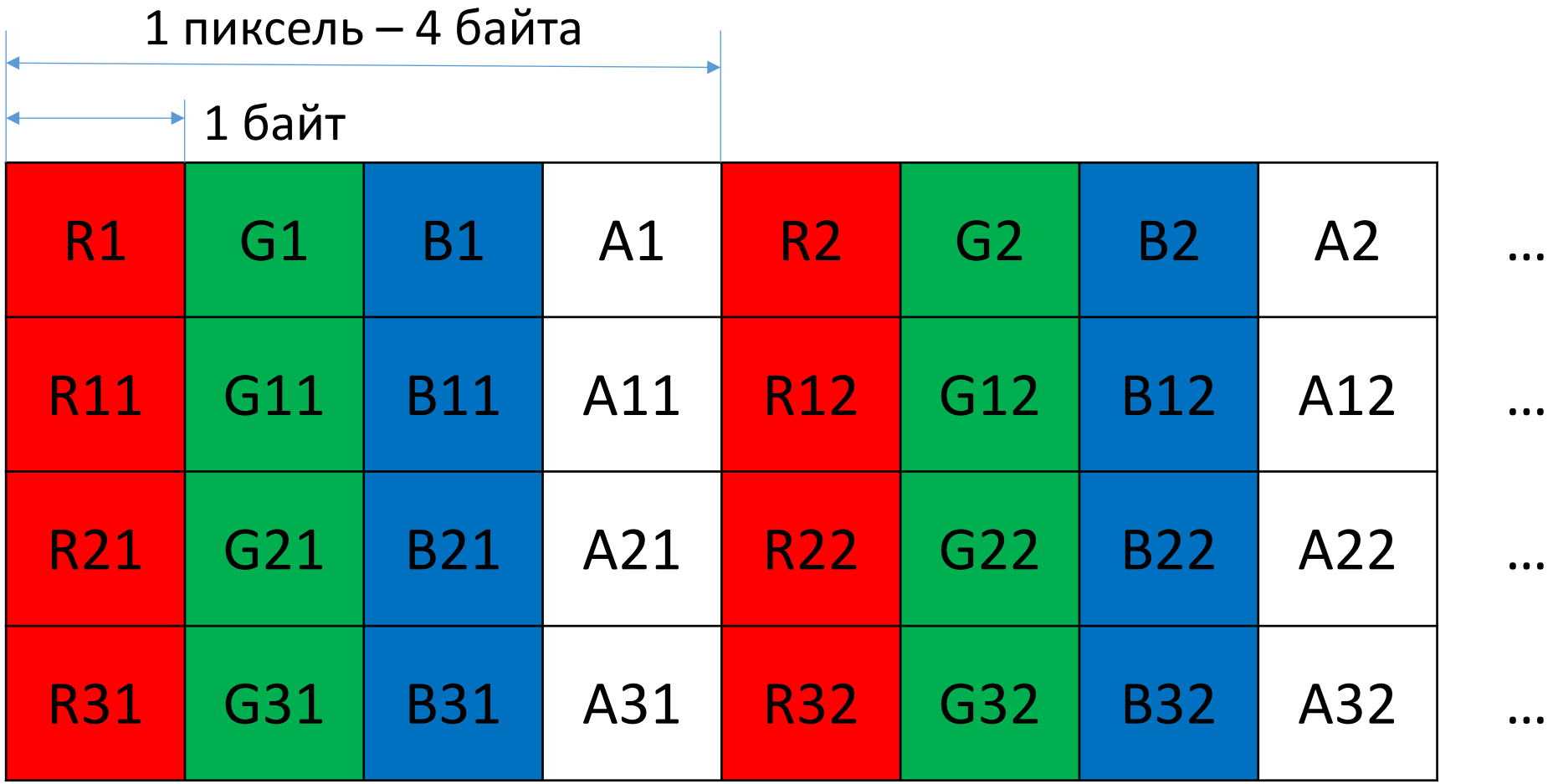
Оригинал

Erosion



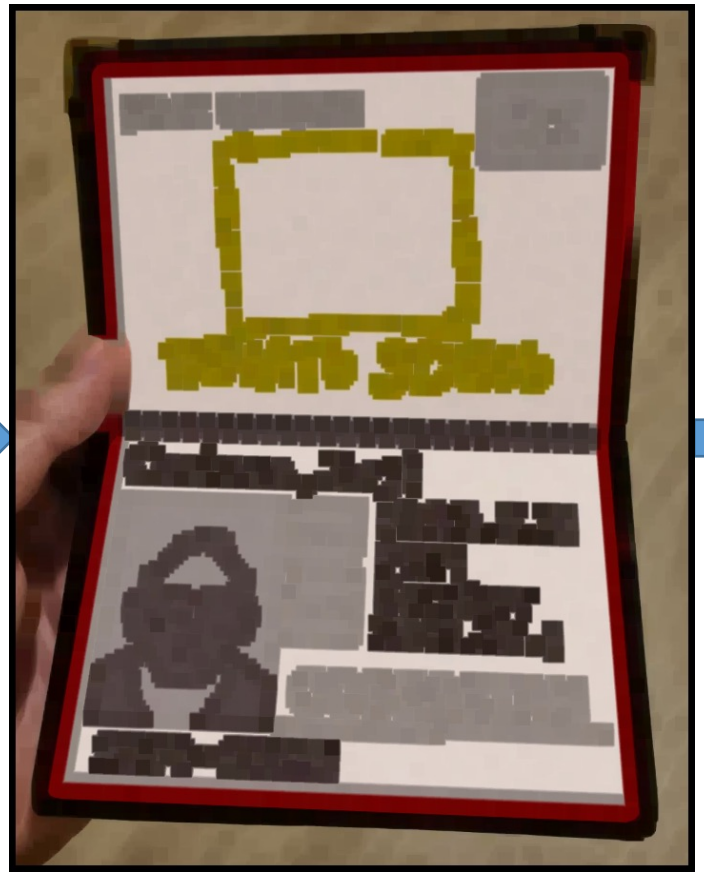
Blur

Структура изображения



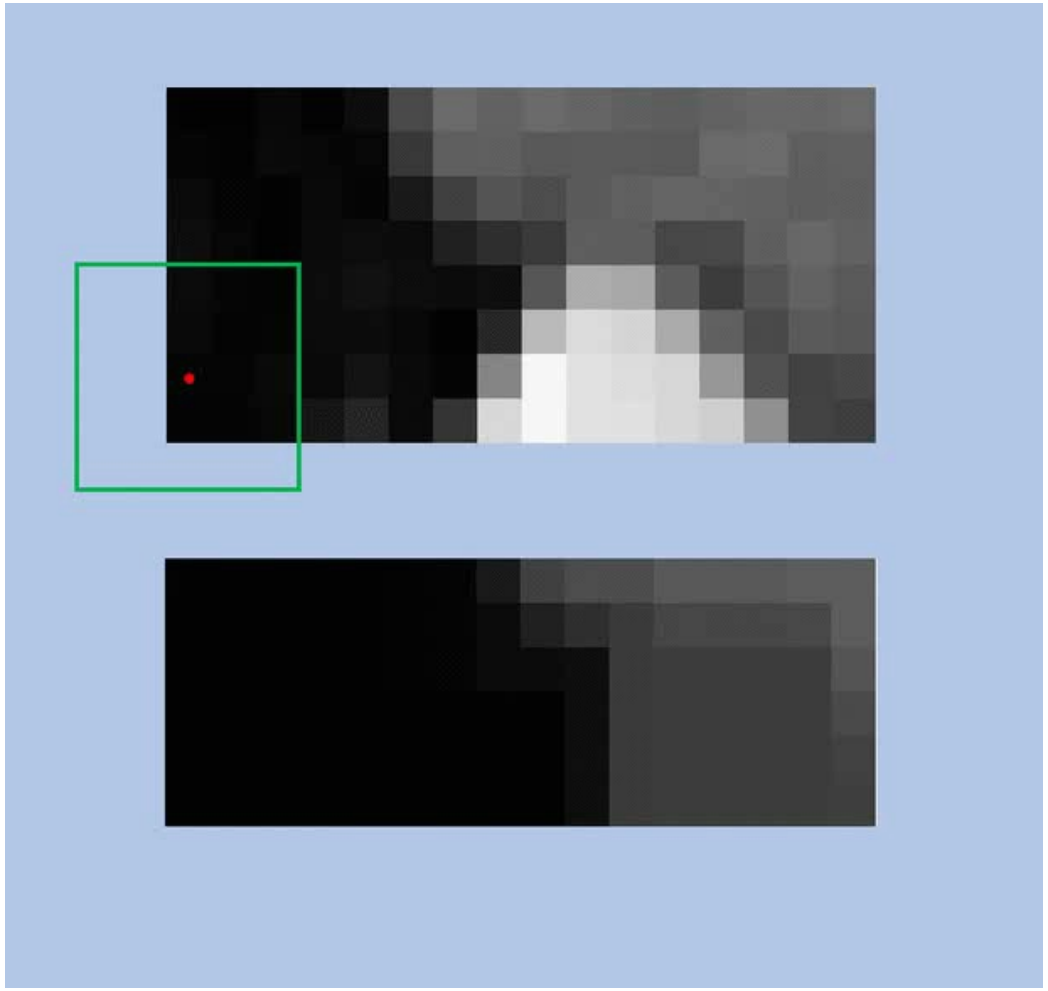
Оригинал

Erosion



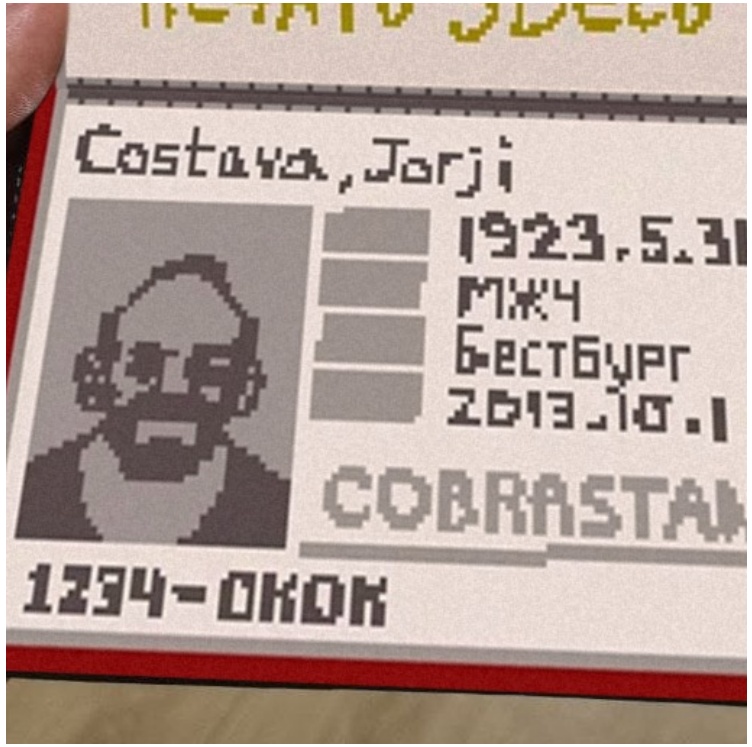
Blur

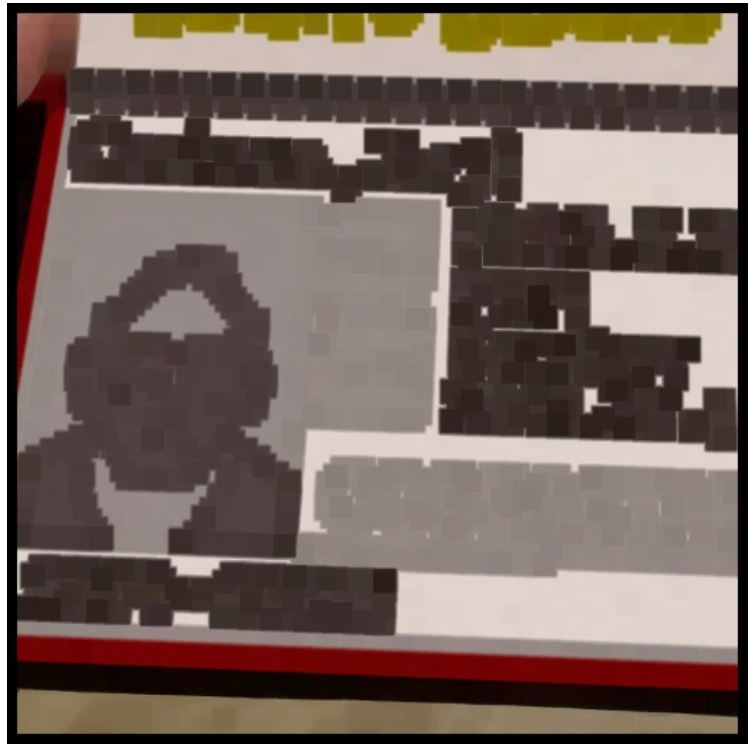
Фильтр Erosion



Фильтр Erosion

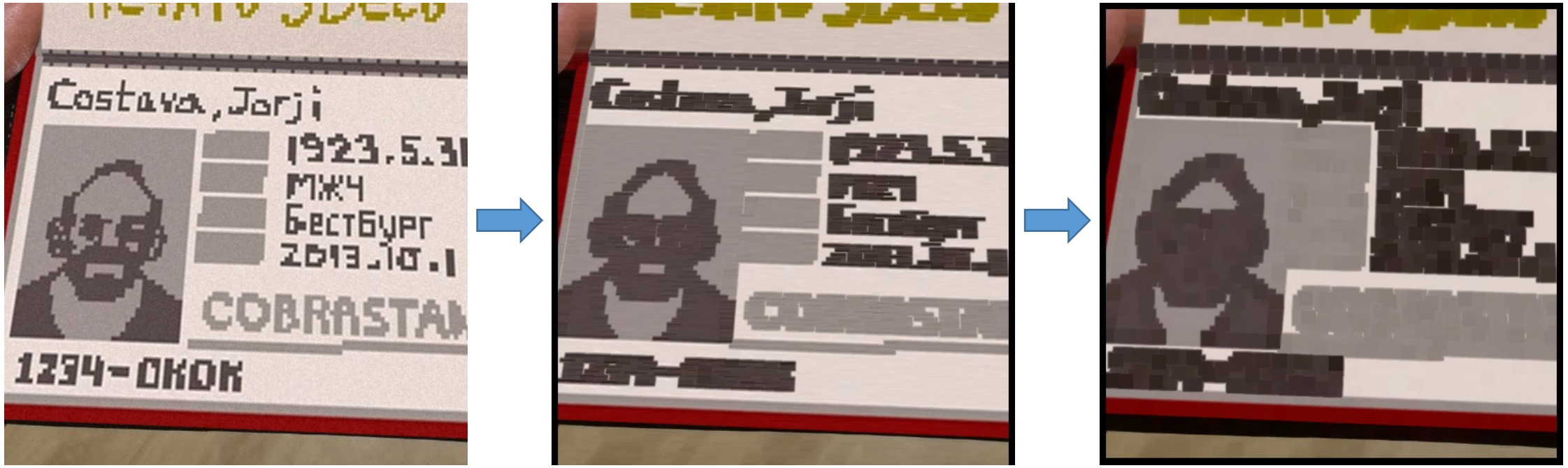
Один проход $O(N \times W^2)$



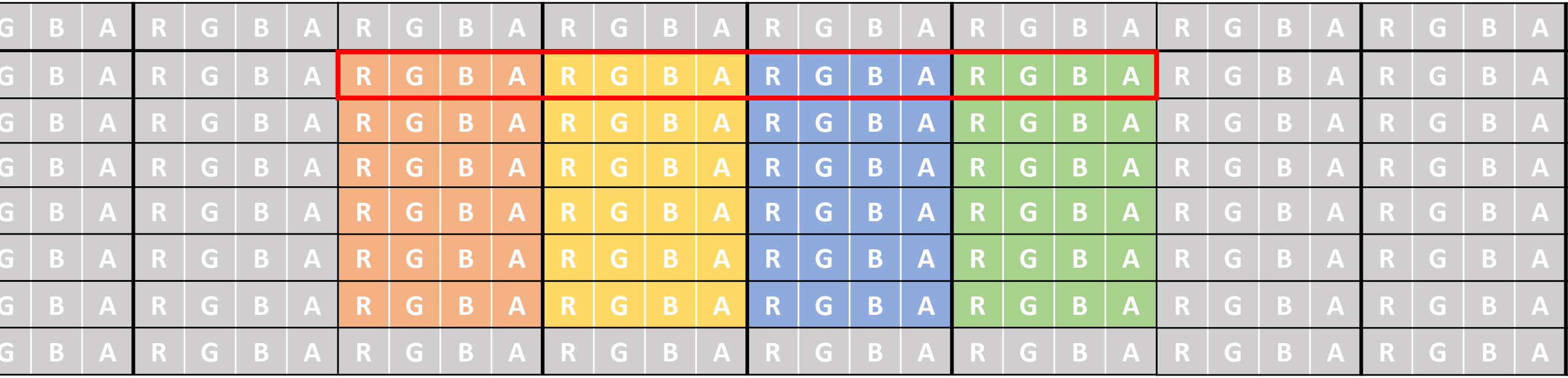


Фильтр Erosion

Два прохода O(N x W)



Вертикальный проход



Вертикальный проход

```
for (var i = radius; i < image.Height - radius - 1; i++)
{
    var middle = (uint*)middleBuffer + (i - radius) * image.Width;
    var output = (uint*)outputBuffer + image.Width * i;
    for (var j = 0; j < image.Width; j += 4)
    {
        ErodeVertical(middle, output, image.Width, windowSize);
        middle += 4;
        output += 4;
    }
}
```

Вертикальный проход

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
ссылка: 1
private static unsafe void ErodeVertical(uint* input, uint* output, int stride, int size)
{
    var min = Sse2.LoadVector128(input).AsByte();
    for (var i = 1; i < size; i++)
    {
        min = Sse2.Min(min, Sse2.LoadVector128(input += stride).AsByte());
    }
    Sse2.Store(output, min.AsUInt32());
}
```

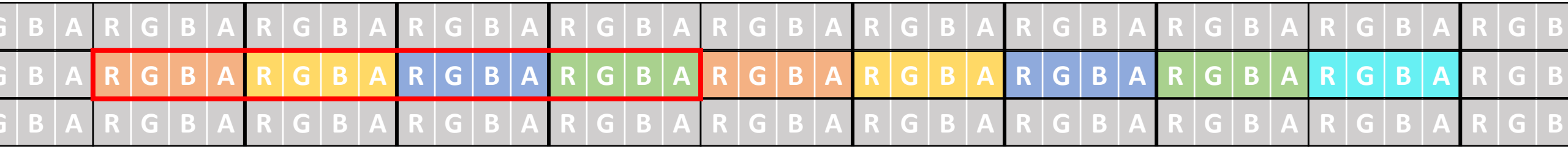
Вертикальный проход

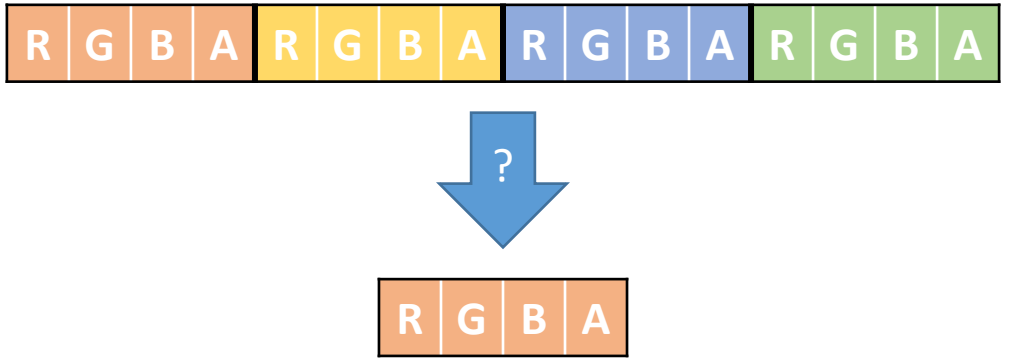
```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
ссылка: 1
private static unsafe void ErodeVertical(uint* input, uint* output, int stride, int size)
{
    var min = Sse2.LoadVector128(input).AsByte();
    for (var i = 1; i < size; i++)
    {
        min = Sse2.Min(min, Sse2.LoadVector128(input += stride).AsByte());
    }
    Sse2.Store(output, min.AsUInt32());
}
```

Вертикальный проход

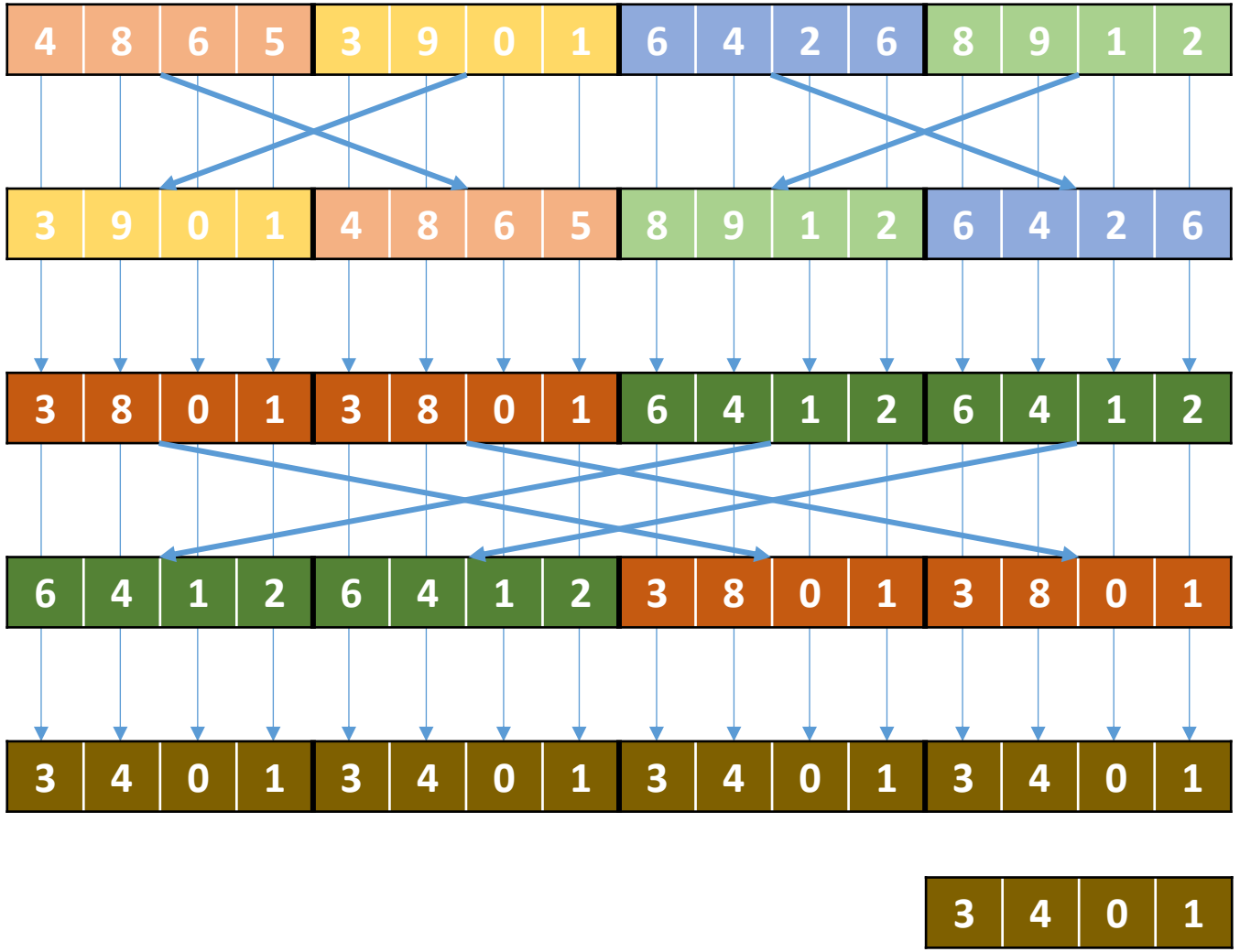
```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
ссылка: 1
private static unsafe void ErodeVertical(uint* input, uint* output, int stride, int size)
{
    var min = Sse2.LoadVector128(input).AsByte();
    for (var i = 1; i < size; i++)
    {
        min = Sse2.Min(min, Sse2.LoadVector128(input += stride).AsByte());
    }
    Sse2.Store(output, min.AsUInt32());
}
```

Горизонтальный проход





Горизонтальный проход



Горизонтальный проход

```
for(var i = 0; i < image.Height; i++)
{
    var input = (uint*)inputBuffer + image.Width * i;
    var middle = (uint*)middleBuffer + image.Width * i + radius;

    for(var j = radius; j < image.Width - radius; j++)
    {
        ErodeHorizontal(input, middle, windowSize);
        input++;
        middle++;
    }
}
```

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
ссылка: 1
private static unsafe void ErodeHorizontal(uint* input, uint* output, int size)
{
    var min = Sse2.LoadVector128(input).AsByte();
    var reminder = size - 4;
    for (var i = 1; i < size / 4; i++)
    {
        input += 4;
        min = Sse2.Min(min, Sse2.LoadVector128(input).AsByte());
        reminder -= 4;
    }
    min = Sse2.Min(min, Sse2.LoadVector128(input + reminder).AsByte());

    min = Sse2.Min(min, Sse2.Shuffle(min.AsInt32(), 0b_10_11_00_01).AsByte());
    min = Sse2.Min(min, Sse2.Shuffle(min.AsInt32(), 0b_01_00_11_10).AsByte());
    output[0] = Sse2.ConvertToUInt32(min.AsUInt32());
}
```

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
ссылка: 1
private static unsafe void ErodeHorizontal(uint* input, uint* output, int size)
{
    var min = Sse2.LoadVector128(input).AsByte();
    var reminder = size - 4;
    for (var i = 1; i < size / 4; i++)
    {
        input += 4;
        min = Sse2.Min(min, Sse2.LoadVector128(input).AsByte());
        reminder -= 4;
    }
    min = Sse2.Min(min, Sse2.LoadVector128(input + reminder).AsByte());

    min = Sse2.Min(min, Sse2.Shuffle(min.AsInt32(), 0b_10_11_00_01).AsByte());
    min = Sse2.Min(min, Sse2.Shuffle(min.AsInt32(), 0b_01_00_11_10).AsByte());
    output[0] = Sse2.ConvertToUInt32(min.AsUInt32());
}
```

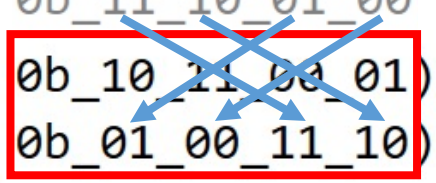
```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
ссылка: 1
private static unsafe void ErodeHorizontal(uint* input, uint* output, int size)
{
    var min = Sse2.LoadVector128(input).AsByte();
    var reminder = size - 4;
    for (var i = 1; i < size / 4; i++)
    {
        input += 4;
        min = Sse2.Min(min, Sse2.LoadVector128(input).AsByte());
        reminder -= 4;
    }
    min = Sse2.Min(min, Sse2.LoadVector128(input + reminder).AsByte());

    min = Sse2.Min(min, Sse2.Shuffle(min.AsInt32(), 0b_10_11_00_01).AsByte());
    min = Sse2.Min(min, Sse2.Shuffle(min.AsInt32(), 0b_01_00_11_10).AsByte());
    output[0] = Sse2.ConvertToUInt32(min.AsUInt32());
}
```

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
ссылка: 1
private static unsafe void ErodeHorizontal(uint* input, uint* output, int size)
{
    var min = Sse2.LoadVector128(input).AsByte();
    var reminder = size - 4;
    for (var i = 1; i < size / 4; i++)
    {
        input += 4;
        min = Sse2.Min(min, Sse2.LoadVector128(input).AsByte());
        reminder -= 4;
    }
    min = Sse2.Min(min, Sse2.LoadVector128(input + reminder).AsByte());

    min = Sse2.Min(min, Sse2.Shuffle(min.AsInt32(), 0b_10_11_00_01).AsByte());
    min = Sse2.Min(min, Sse2.Shuffle(min.AsInt32(), 0b_01_00_11_10).AsByte());
    output[0] = Sse2.ConvertToUInt32(min.AsUInt32());
}
```

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
ссылка: 1
private static unsafe void ErodeHorizontal(uint* input, uint* output, int size)
{
    var min = Sse2.LoadVector128(input).AsByte();
    var reminder = size - 4;
    for (var i = 1; i < size / 4; i++)
    {
        input += 4;
        min = Sse2.Min(min, Sse2.LoadVector128(input).AsByte());
        reminder -= 4;
    }
    min = Sse2.Min(min, Sse2.LoadVector128(input + reminder).AsByte());
                                                  0b_11_10_01_00
    min = Sse2.Min(min, Sse2.Shuffle(min.AsInt32(), 0b_10_11_00_01).AsByte());
    min = Sse2.Min(min, Sse2.Shuffle(min.AsInt32(), 0b_01_00_11_10).AsByte());
    output[0] = Sse2.ConvertToUInt32(min.AsUInt32());
}
```

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
ссылка: 1
private static unsafe void ErodeHorizontal(uint* input, uint* output, int size)
{
    var min = Sse2.LoadVector128(input).AsByte();
    var reminder = size - 4;
    for (var i = 1; i < size / 4; i++)
    {
        input += 4;
        min = Sse2.Min(min, Sse2.LoadVector128(input).AsByte());
        reminder -= 4;
    }
    min = Sse2.Min(min, Sse2.LoadVector128(input + reminder).AsByte());
                                                  0b_11_10_01_00
    min = Sse2.Min(min, Sse2.Shuffle(min.AsInt32(), 0b_10_11_00_01).AsByte());
    min = Sse2.Min(min, Sse2.Shuffle(min.AsInt32(), 0b_01_00_11_10).AsByte());
    output[0] = Sse2.ConvertToUInt32(min.AsUInt32());
}
```

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
ссылка: 1
private static unsafe void ErodeHorizontal(uint* input, uint* output, int size)
{
    var min = Sse2.LoadVector128(input).AsByte();
    var reminder = size - 4;
    for (var i = 1; i < size / 4; i++)
    {
        input += 4;
        min = Sse2.Min(min, Sse2.LoadVector128(input).AsByte());
        reminder -= 4;
    }
    min = Sse2.Min(min, Sse2.LoadVector128(input + reminder).AsByte());

    min = Sse2.Min(min, Sse2.Shuffle(min.AsInt32(), 0b_10_11_00_01).AsByte());
    min = Sse2.Min(min, Sse2.Shuffle(min.AsInt32(), 0b_01_00_11_10).AsByte());
    output[0] = Sse2.ConvertToUInt32(min.AsUInt32());
}
```



Результат

Method	Mean	Error	StdDev	Median	Ratio	Improve
Erode_Skia	40.595 ms	0.9942 ms	0.6576 ms	40.675 ms	1.00	1,00
Erode_Sse	6.900 ms	0.3608 ms	0.1887 ms	6.893 ms	0.17	**5,87**

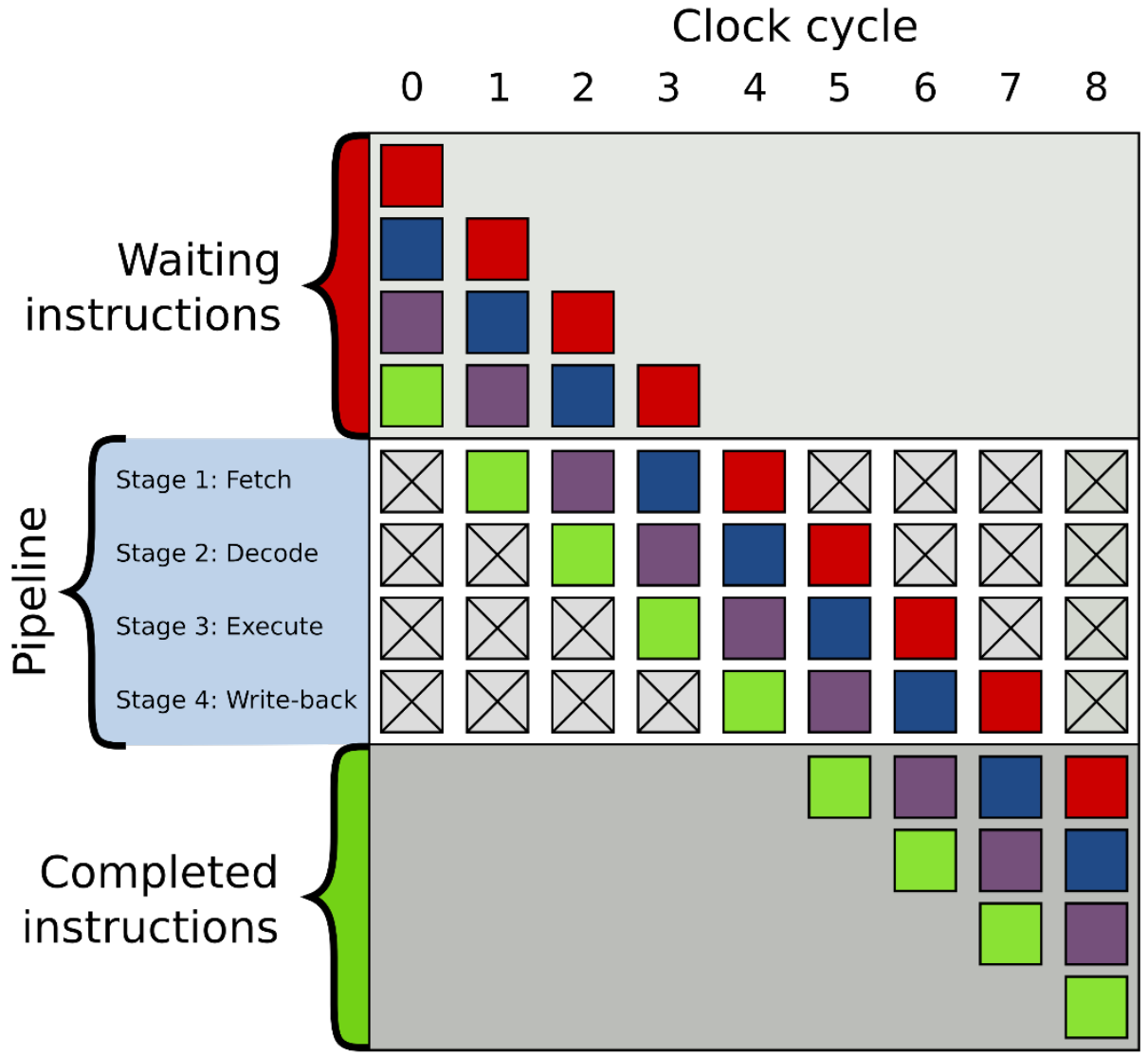
А ещё быстрее?

```
1 for (var i = radius; i < image.Height - radius - 1; i++)
{
    var middle = (uint*)middleBuffer + (i - radius) * image.Width;
    var output = (uint*)outputBuffer + image.Width * i;
  2 for (var j = 0; j < image.Width; j += 4)
    {
        ErodeVertical(middle, output, image.Width, windowSize);
        middle += 4;
        output += 4;
    }
}
```

ссылка: 1

```
private static unsafe void ErodeVertical(uint* input, uint* output, int stride, int size)
{
    var min = Sse2.LoadVector128(input).AsByte();
  3 for (var i = 1; i < size; i++)
    {
        min = Sse2.Min(min, Sse2.LoadVector128(input += stride).AsByte());
    }
    Sse2.Store(output, min.AsUInt32());
}
```

Branch (mis)prediction



```csharp
private static unsafe void ErodeVertical(uint* input, uint* output, int stride)
{
    var min = Sse2.LoadVector128(input).AsByte();
    min = Sse2.Min(min, Sse2.LoadVector128(input += stride).AsByte());
    min = Sse2.Min(min, Sse2.LoadVector128(input += stride).AsByte());
    min = Sse2.Min(min, Sse2.LoadVector128(input += stride).AsByte());
    min = Sse2.Min(min, Sse2.LoadVector128(input += stride).AsByte());
    min = Sse2.Min(min, Sse2.LoadVector128(input += stride).AsByte());
    min = Sse2.Min(min, Sse2.LoadVector128(input += stride).AsByte());
    min = Sse2.Min(min, Sse2.LoadVector128(input += stride).AsByte());
    min = Sse2.Min(min, Sse2.LoadVector128(input += stride).AsByte());
    min = Sse2.Min(min, Sse2.LoadVector128(input += stride).AsByte());
    min = Sse2.Min(min, Sse2.LoadVector128(input += stride).AsByte());
    min = Sse2.Min(min, Sse2.LoadVector128(input += stride).AsByte());
    min = Sse2.Min(min, Sse2.LoadVector128(input += stride).AsByte());
    min = Sse2.Min(min, Sse2.LoadVector128(input += stride).AsByte());
    min = Sse2.Min(min, Sse2.LoadVector128(input += stride).AsByte());
    Sse2.Store(output, min.AsUInt32());
}
```

15

```
private static unsafe void ErodeHorizontal(uint* input, uint* output)
{
    var min = Sse2.LoadVector128(input).AsByte(); // 0-3
    min = Sse2.Min(min, Sse2.LoadVector128(input += 4).AsByte()); // 4-7
    min = Sse2.Min(min, Sse2.LoadVector128(input += 4).AsByte()); // 8-11
    min = Sse2.Min(min, Sse2.LoadVector128(input += 3).AsByte()); // 11-14
    min = Sse2.Min(min, Sse2.Shuffle(min.AsInt32(), 0b_10_11_00_01).AsByte());
    min = Sse2.Min(min, Sse2.Shuffle(min.AsInt32(), 0b_01_00_11_10).AsByte());
    output[0] = Sse2.ConvertToUInt32(min.AsUInt32());
}
```

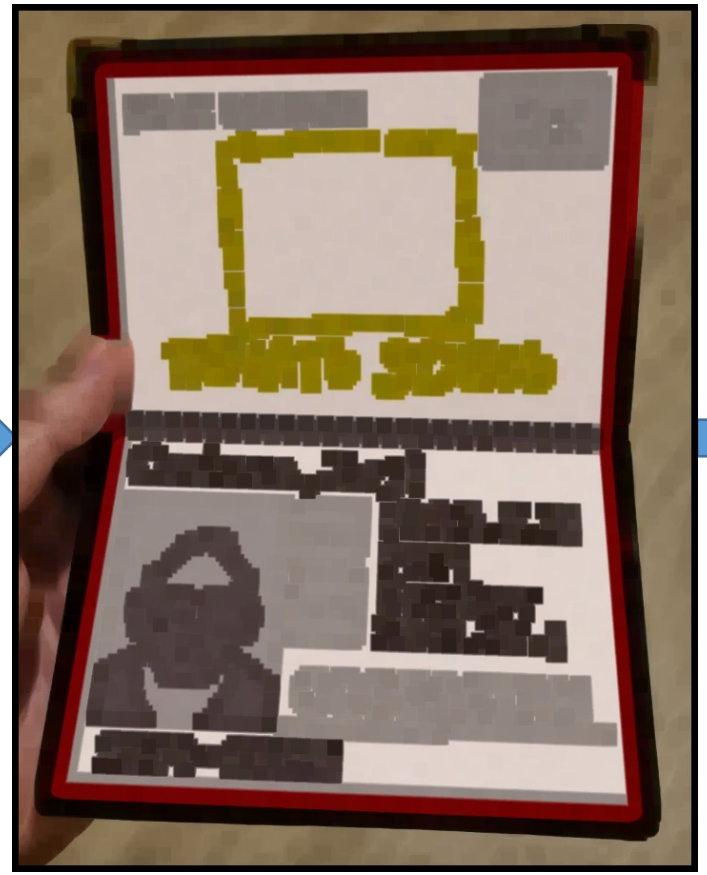
```
private static unsafe void ErodeHorizontal(uint* input, uint* output)
{
    var min = Sse2.LoadVector128(input).AsByte(); // 0-3
    min = Sse2.Min(min, Sse2.LoadVector128(input += 4).AsByte()); // 4-7
    min = Sse2.Min(min, Sse2.LoadVector128(input += 4).AsByte()); // 8-11
    min = Sse2.Min(min, Sse2.LoadVector128(input += 3).AsByte()); // 11-14
    min = Sse2.Min(min, Sse2.Shuffle(min.AsInt32(), 0b_10_11_00_01).AsByte());
    min = Sse2.Min(min, Sse2.Shuffle(min.AsInt32(), 0b_01_00_11_10).AsByte());
    output[0] = Sse2.ConvertToUInt32(min.AsUInt32());
}
```

Результат

Method	Mean	Error	StdDev	Median	Ratio	Improve
Erode_Skia	40.595 ms	0.9942 ms	0.6576 ms	40.675 ms	1.00	1,00
Erode_Sse	6.900 ms	0.3608 ms	0.1887 ms	6.893 ms	0.17	5,87
Erode_Sse15	5.789 ms	0.3542 ms	0.2343 ms	5.766 ms	0.14	**7,03**

Оригинал

Erosion

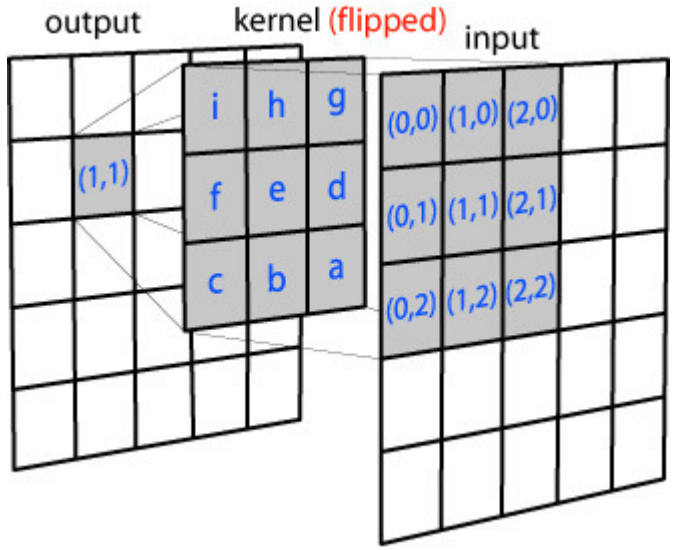


Blur

Фильтр Blur

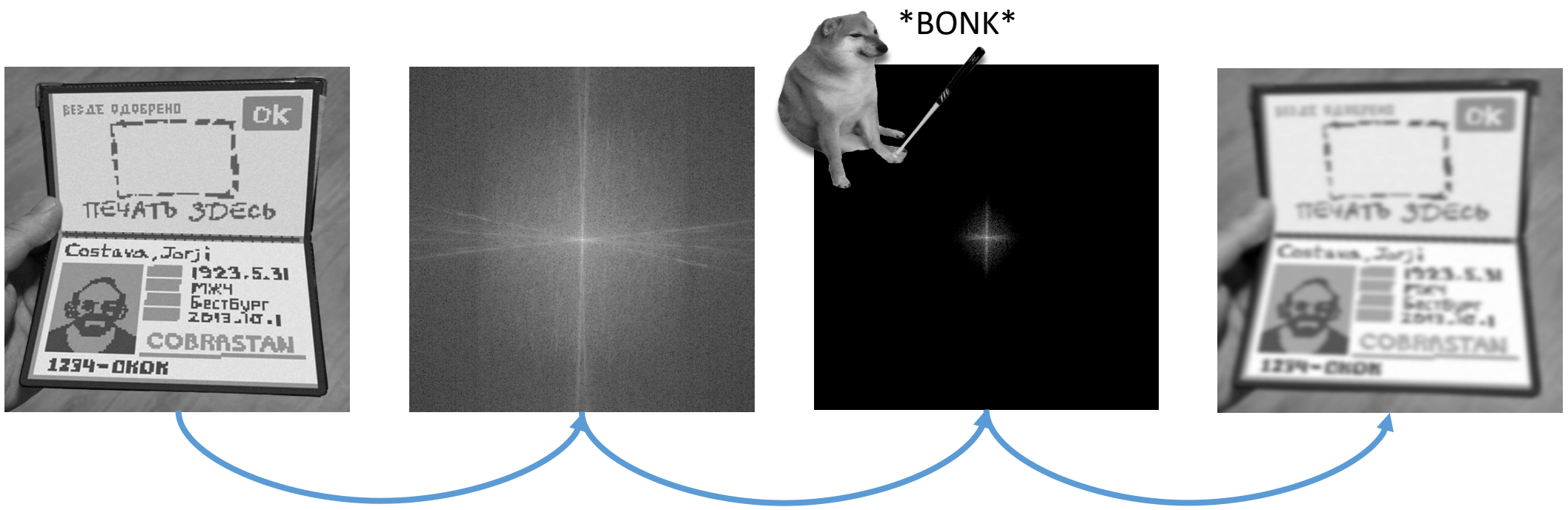
Фильтр Blur

- В наивной имплементации O(N x W^2)
- Может быть разделён на два прохода по O(N x W)



Фильтр Blur

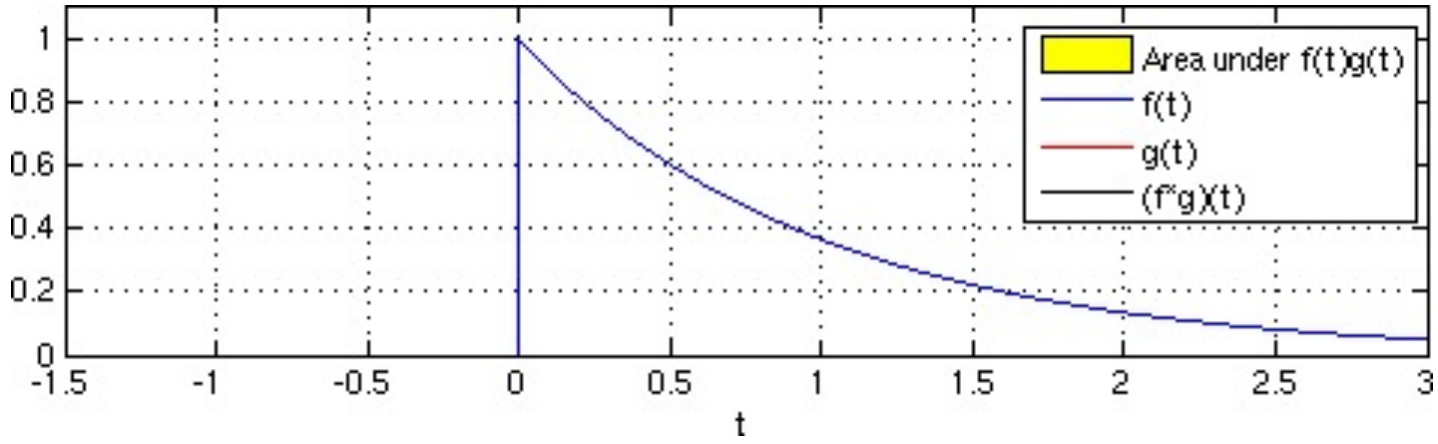
- Свёртка дискретным преобразованием Фурье: O(N log(N))



Фильтр Blur

- Аппроксимация импульсным фильтром: O(N)

$$y[n] = y'[n] + a0 * x[n] + a1 * x[n-1] - b1 * y[n-1] - b2 * y[n-2]$$

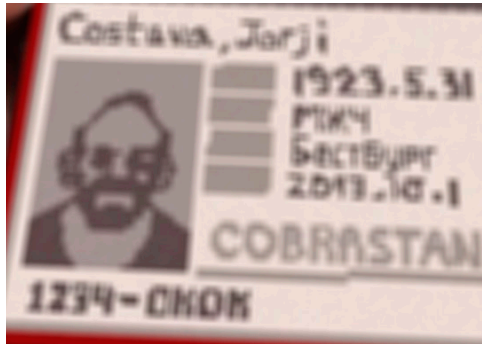


Фильтр Blur

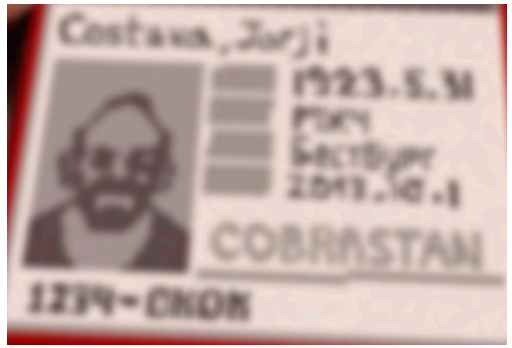
- Аппроксимация через три Box Blur: O(N + W)

Оригинал

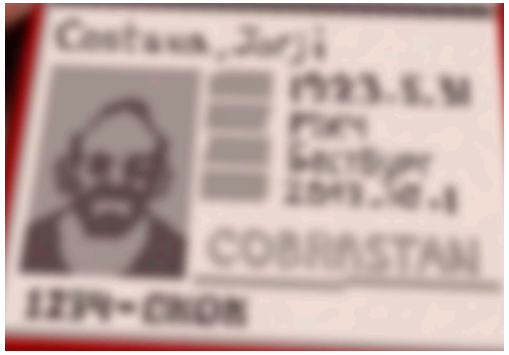
Box blur x1

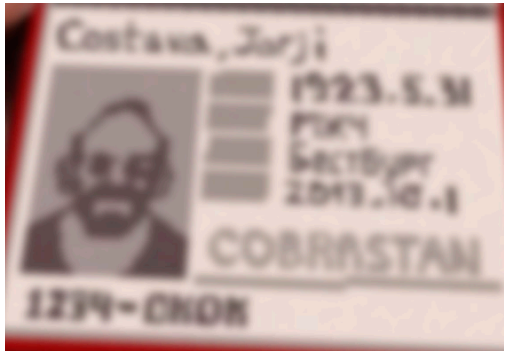


Box blur x2



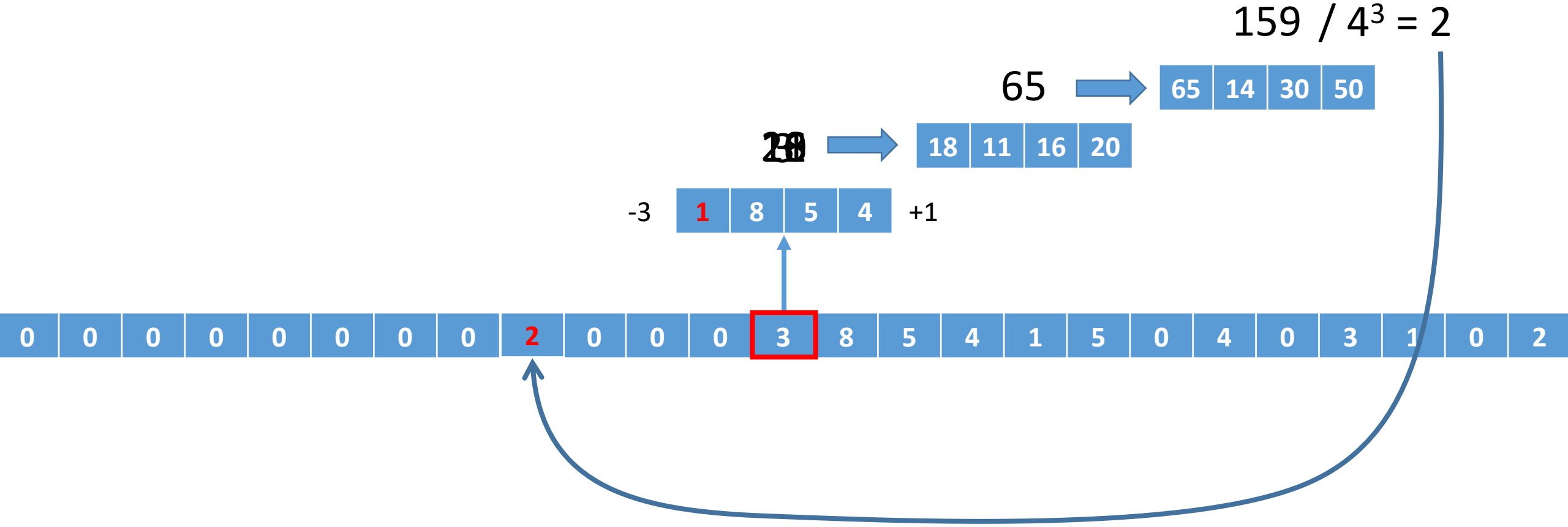
Box blur x3





Gaussian blur

Среднее в кольцевом буфере



Деление через умножение и сдвиг

Пример в десятичном виде

$X / 125 \Leftrightarrow X \cdot 0.008 \Leftrightarrow X \cdot 8 \cdot (0.001)$

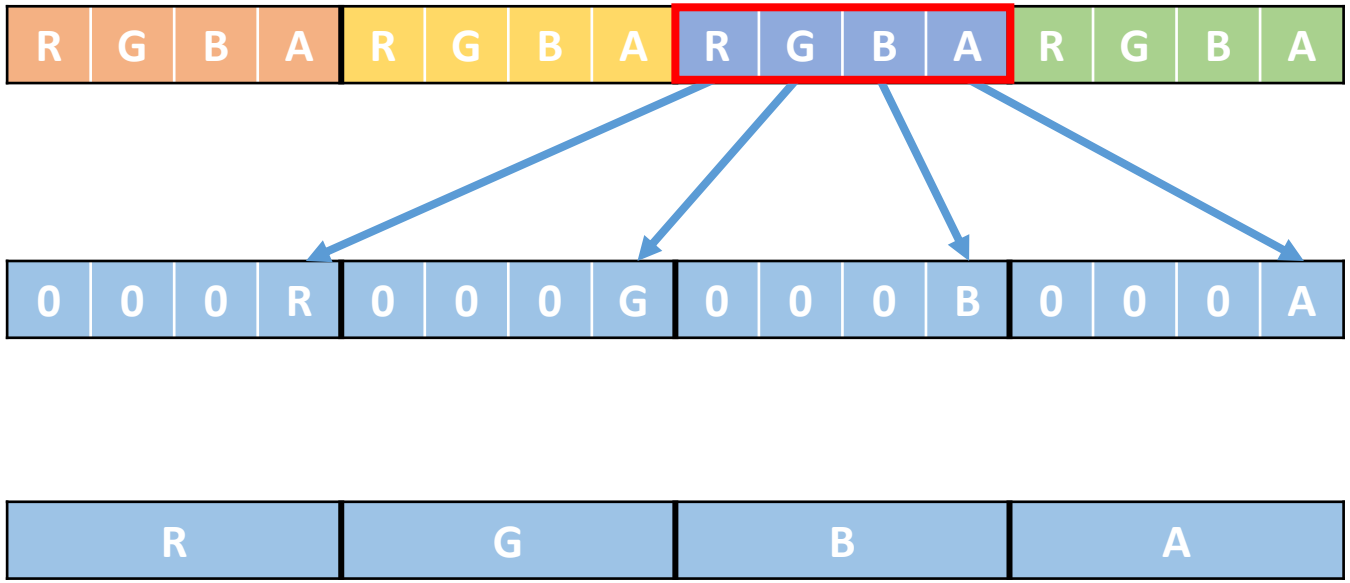
$1672 / 125 \Leftrightarrow 1672 \cdot 0.008 \Leftrightarrow 1672 \cdot 8 \cdot (0.001) \Leftrightarrow 13376$

Формула

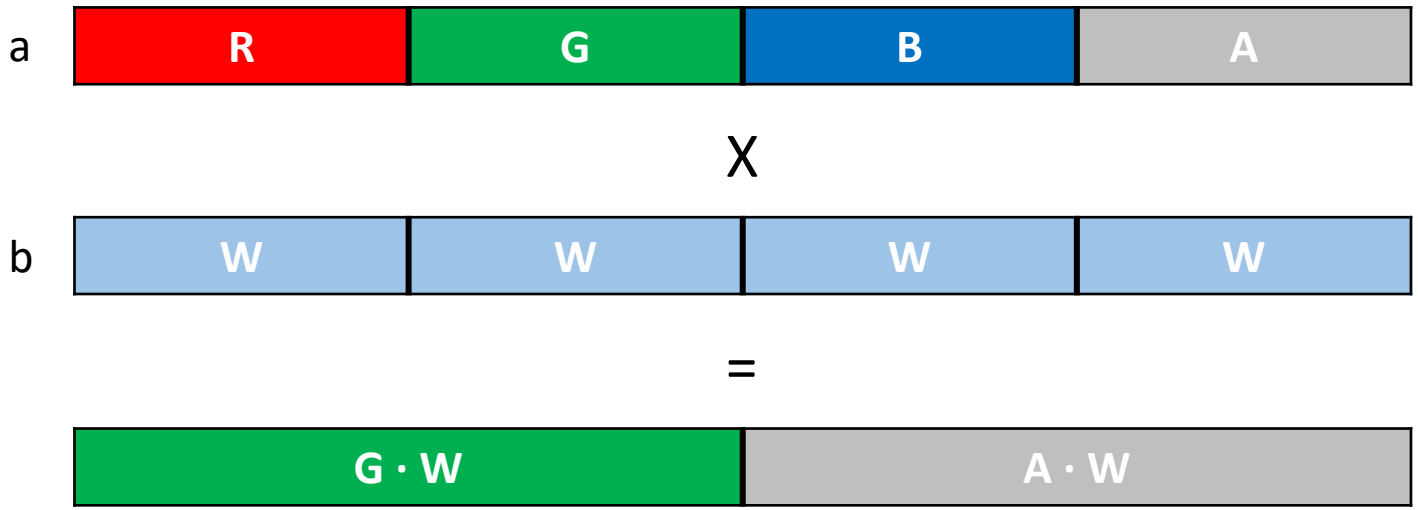
Посчитаем заранее: $W = \text{Round}((1 / M) \cdot (1 << 32))$

Можно умножать: $X / M \Leftrightarrow (X \cdot W) >> 32$

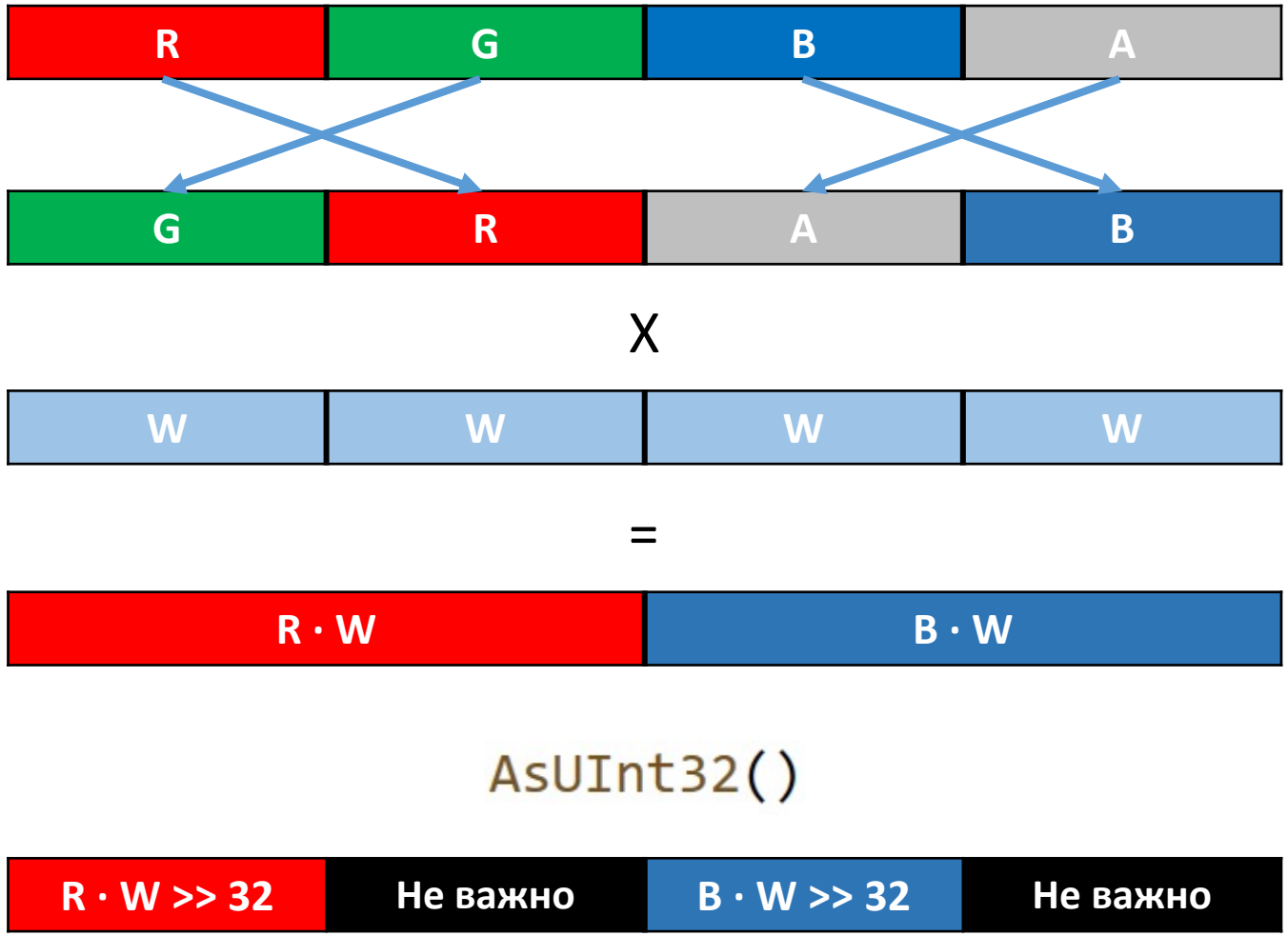
```
Sse41.ConvertToVector128Int32(address).AsUInt32();
```



```
Sse2.Multiply(a, b);
```

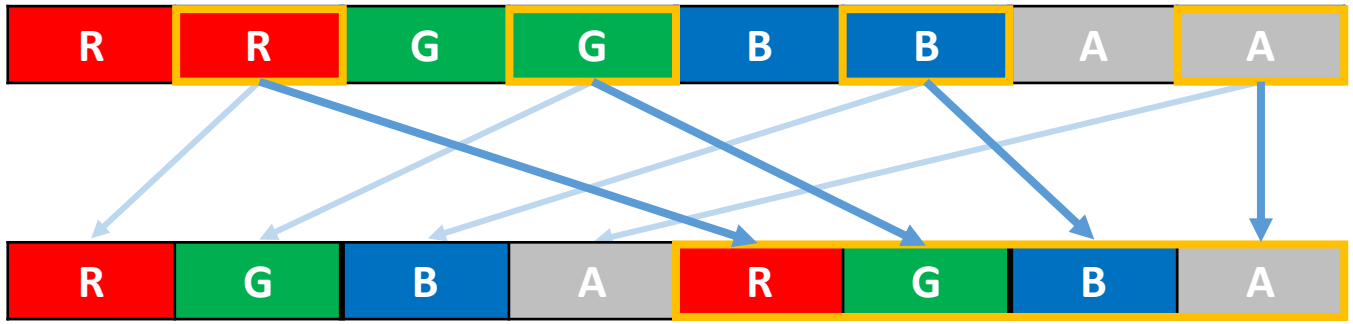



```
Sse2.ShiftRightLogical(vector, 32);
```


R
G
B
A
G
R
A
B
X
W
W
W
W
=
R · W
B · W
AsUInt32()
R · W >> 32
Не важно
B · W >> 32
Не важно


```
Sse41.Blend(vector1, vector2, 0b11_00_11_00);
```


```
Sse41.PackUnsignedSaturate(vector, vector).AsInt16();
```




```
Sse2.PackUnsignedSaturate(vector, vector);
```

Результат

Method	Mean	Error	StdDev	Median	Ratio	RatioSD	Improve
Blur_Skia	18.991 ms	1.5311 ms	0.9111 ms	19.084 ms	1.00	0.00	1,00
Blur_Sse	18.983 ms	1.1705 ms	0.6966 ms	19.286 ms	1.00	0.06	**1,00**

Провал?

Шах и мат, оптимизисты!

```
void twice(int* num) {
    num[0] = num[0] + num[0];
}
```

```
twice(int*): # @twice(int*)
  shl dword ptr [rdi]
  ret
```

Шах и мат, оптимизисты!

```
void twice(int* num) {
    num[0] = num[0] + num[0];
    num[1] = num[1] + num[1];
    num[2] = num[2] + num[2];
    num[3] = num[3] + num[3];
}
```

```
twice(int*): # @twice(int*)
  movdqu xmm0, xmmword ptr [rdi]
  paddd xmm0, xmm0
  movdqu xmmword ptr [rdi], xmm0
  ret
```

Шах и мат, оптимизисты!

```
void twice(int* num) {
    num[0] = num[0] + num[0];
    num[1] = num[1] + num[1];
    num[2] = num[2] + num[2];
    num[3] = num[3] + num[3];
    num[4] = num[4] + num[4];
    num[5] = num[5] + num[5];
    num[6] = num[6] + num[6];
    num[7] = num[7] + num[7];
}
```

```
twice(int*): # @twice(int*)
  vmovdqu ymm0, ymmword ptr [rdi]
  vpaddd ymm0, ymm0, ymm0
  vmovdqu ymmword ptr [rdi], ymm0
  vzeroupper
  ret
```

Переводим на AVX2!

Результат

Method	Mean	Error	StdDev	Median	Ratio	RatioSD	Improve
Blur_Skia	18.991 ms	1.5311 ms	0.9111 ms	19.084 ms	1.00	0.00	1,00
Blur_Sse	18.983 ms	1.1705 ms	0.6966 ms	19.286 ms	1.00	0.06	1,00
Blur_Avx2	9.839 ms	0.6057 ms	0.4006 ms	9.873 ms	0.52	0.02	**1,93**

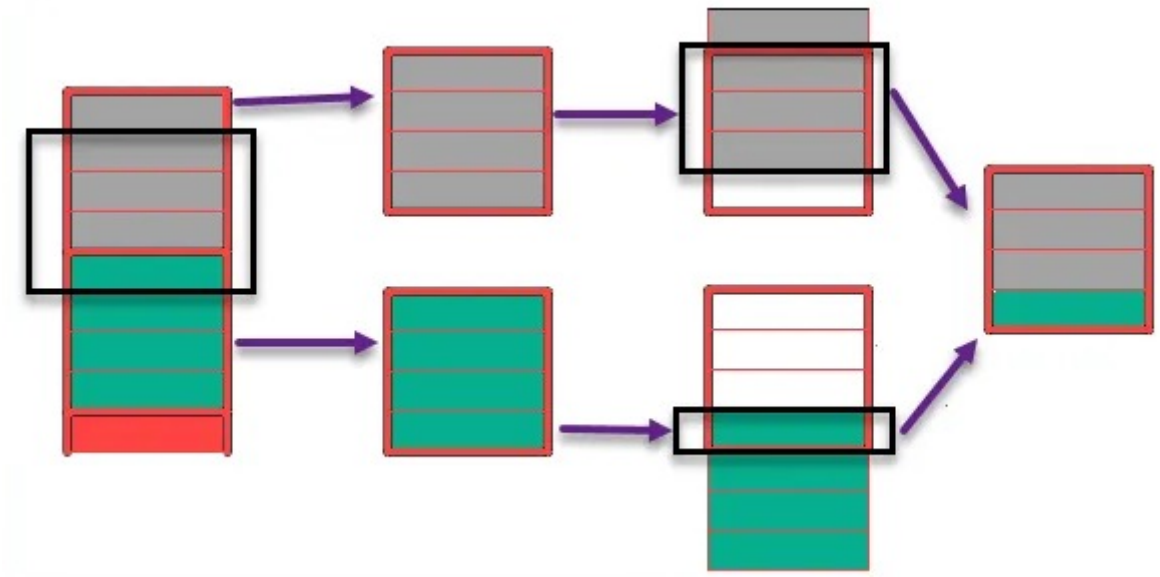
Почти в два раза?

Частотное поведение

Mode	Base	Turbo Frequency/Active Cores													
		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Normal	3,000 MHz	4,500 MHz	4,500 MHz	4,200 MHz	4,200 MHz	4,100 MHz	4,100 MHz	4,100 MHz	4,100 MHz	4,100 MHz	4,100 MHz	4,100 MHz	4,100 MHz	3,900 MHz	3,900 MHz
AVX2		3,900 MHz	3,900 MHz	3,700 MHz	3,700 MHz	3,600 MHz	3,600 MHz	3,600 MHz	3,600 MHz	3,600 MHz	3,600 MHz	3,600 MHz	3,600 MHz	3,400 MHz	3,400 MHz
AVX512		3,700 MHz	3,700 MHz	3,500 MHz	3,500 MHz	3,400 MHz	3,400 MHz	3,400 MHz	3,400 MHz	3,200 MHz	3,200 MHz	3,200 MHz	3,200 MHz	2,900 MHz	2,900 MHz

Что ещё можно оптимизировать?

Выравнивание памяти



```
var buffer = stackalloc Pixel32[bufferSize];
```

```
var buffer = stackalloc Pixel32[bufferSize + 1];
var alignedBuffer = Align(buffer, 32);
```

```
private unsafe T* Align<T>(T* pointer, ulong aligment) where T: unmanaged
{
    if (!IsPowerOfTwo(aligment)) { /* throw */ }

    var mask = aligment - 1;
    return (T*)(((ulong)pointer + mask) & ~mask);
}
```

Пример

Alignment	0010_0000 (32)
Mask	0001_1111 (31)
Pointer	0010_1010 (42)
Pointer + Mask	0100_1001 (73)
~ Mask	1110_0000 (224)
Pointer + Mask & ~ Mask	0100_0000 (64)

Результат

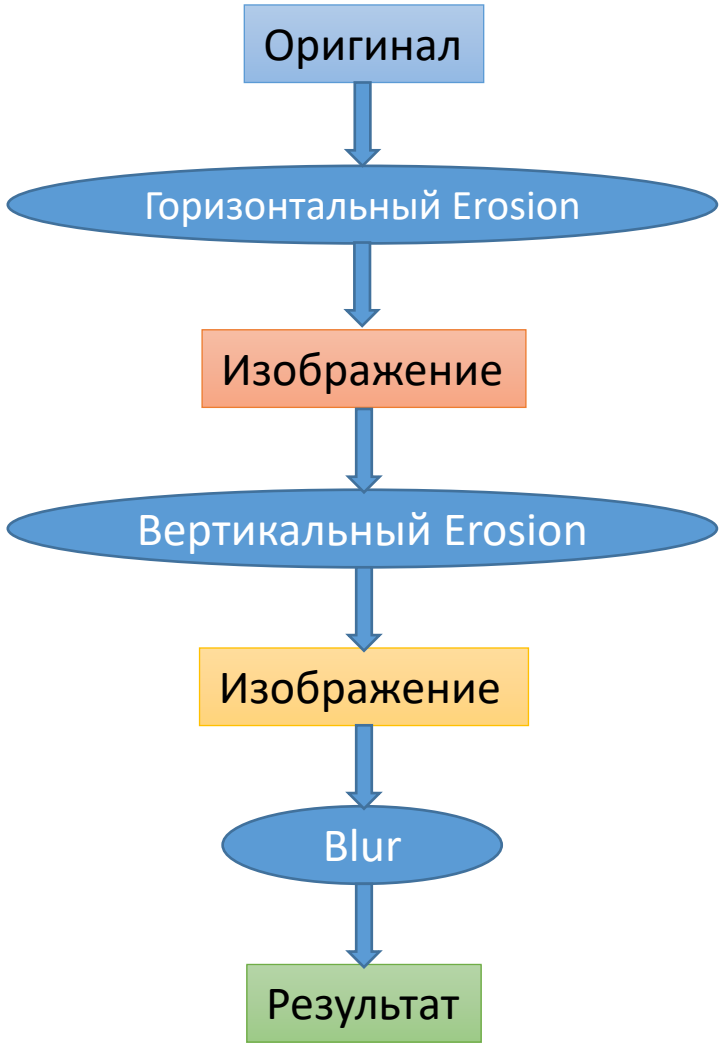
Method	Mean	Error	StdDev	Median	Ratio	RatioSD	Improve
Blur_Skia	18.991 ms	1.5311 ms	0.9111 ms	19.084 ms	1.00	0.00	1,00
Blur_Sse	18.983 ms	1.1705 ms	0.6966 ms	19.286 ms	1.00	0.06	1,00
Blur_Avx2	9.839 ms	0.6057 ms	0.4006 ms	9.873 ms	0.52	0.02	1,93
Blur_Avx2Aligned	9.260 ms	0.3578 ms	0.2367 ms	9.249 ms	0.49	0.03	**2,06**

Erosion + Blur

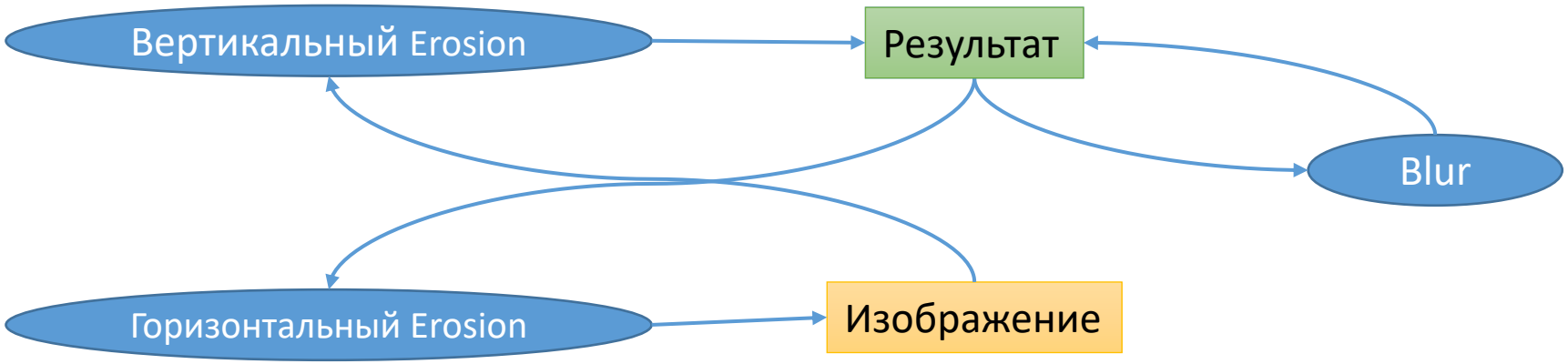
Method	Mean	Error	StdDev	Median	Ratio	Improve
Skia	47.22 ms	1.851 ms	1.224 ms	47.10 ms	1.00	1,00
Intrinsics	15.57 ms	0.609 ms	0.403 ms	15.57 ms	0.33	**3,03**

А можно ещё?

Промежуточная память



Промежуточная память



Финальный результат

Method	Mean	Error	StdDev	Median	Ratio	Improve
Skia	47.22 ms	1.851 ms	1.224 ms	47.10 ms	1.00	1,00
Intrinsics	15.57 ms	0.609 ms	0.403 ms	15.57 ms	0.33	3,03
LessBuffers	13.51 ms	0.329 ms	0.196 ms	13.49 ms	0.29	**3,50**

Итоги

Ссылки

Intel Intrinsic Guide:

https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html

SKIA:

https://github.com/google/skia

https://github.com/mono/SkiaSharp

SharpLab:

https://sharplab.io/

IIR Gaussian Blur Filter:

https://software.intel.com/content/dam/develop/external/us/en/documents/gaussian-filter-181134.pdf

ТИНЬКОФФ

Спасибо за внимание!

Степан Артемьев

2021