SIMD - ЧТО ЭТО ТАКОЕ?

И ЗАЧЕМ МОЖЕТ ПРИГОДИТЬСЯ?

SIMD - ЧТО ЭТО ТАКОЕ?

И ЗАЧЕМ МОЖЕТ ПРИГОДИТЬСЯ?

Single Instruction Multiple Data

- SIMD - это расширение процессора
- SIMD позволяет выполнить одну операцию над набором данных одновременно

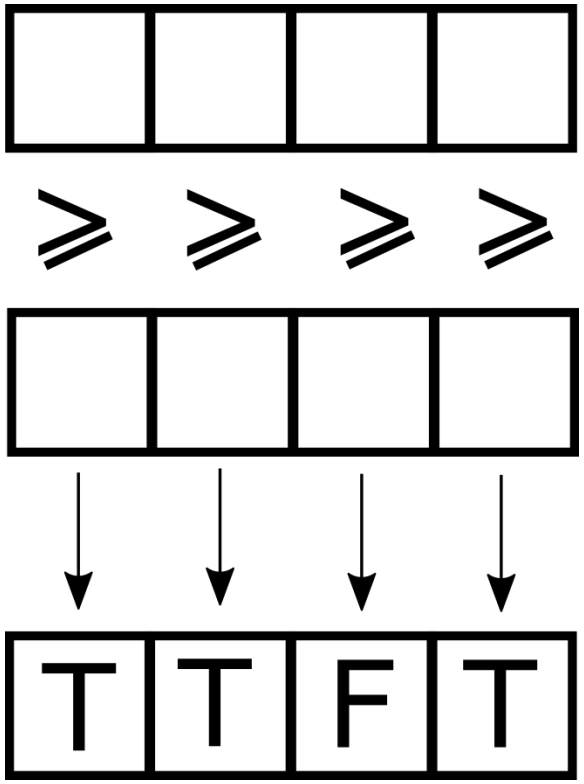
Что за данные?

- Непрерывный отрезок массива
- Геометрические примитивы – точка, вектор, матрица поворота и т.п.

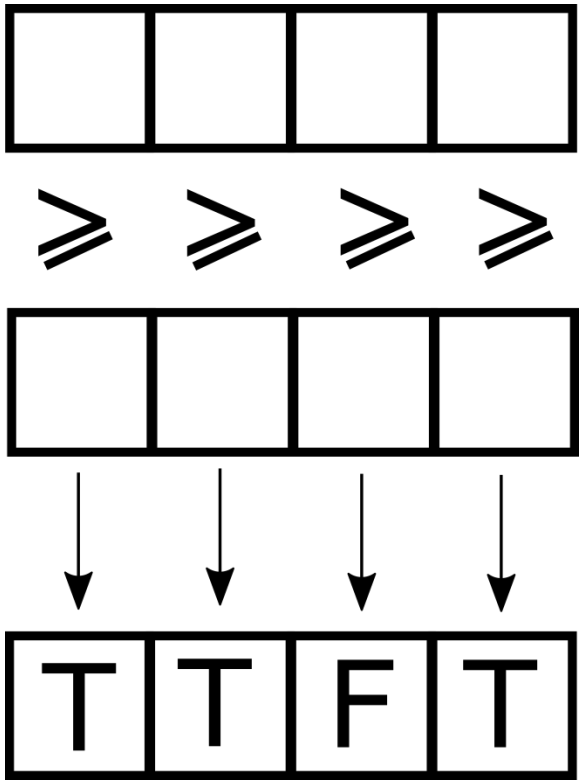
Что за операции?

- Простые арифметические операции
- Различные виды условий
- Еще куча всего

Пример SIMD условия

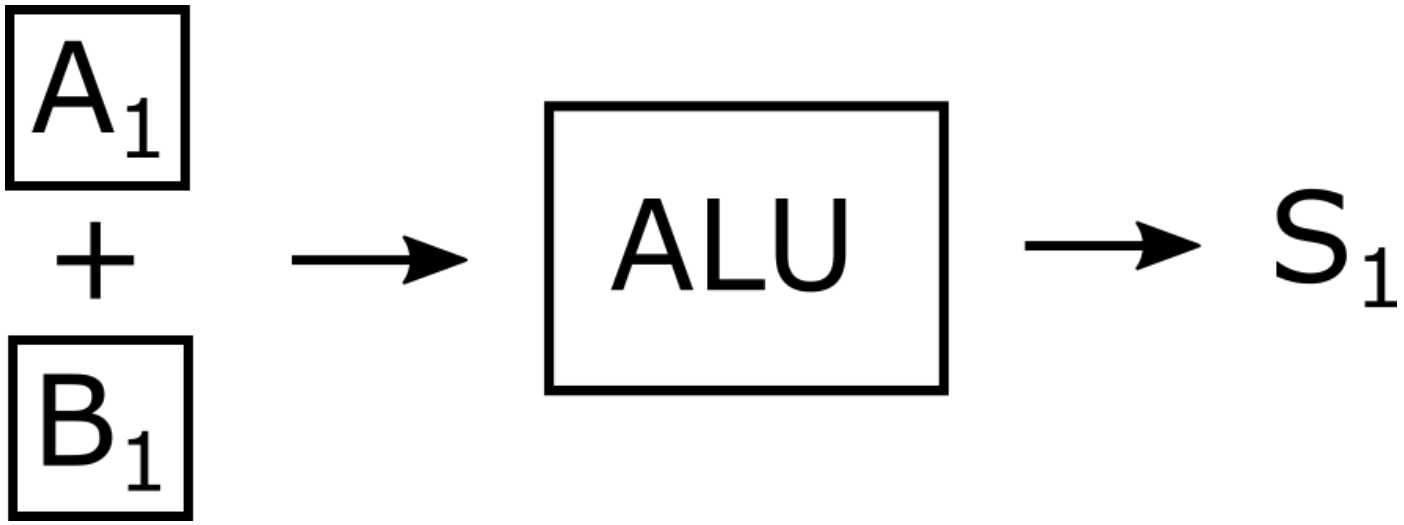


Пример SIMD условия

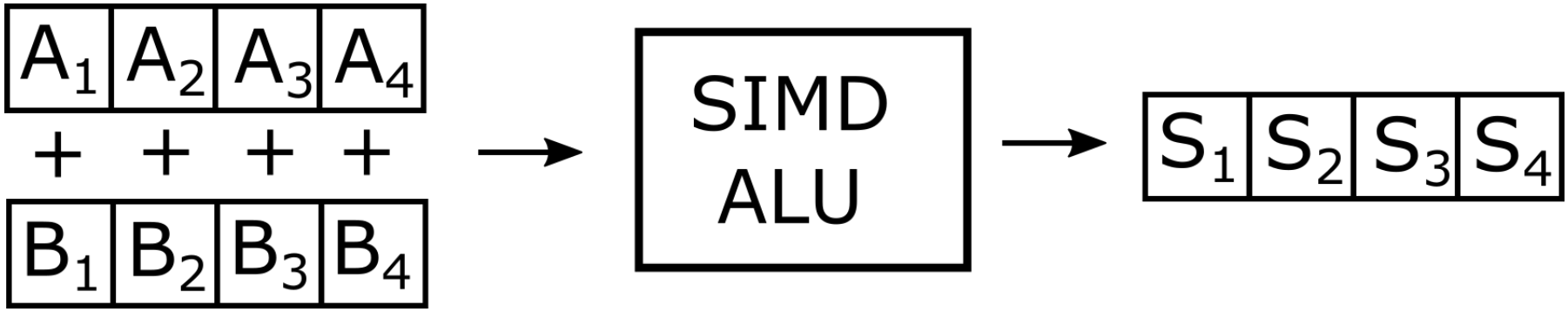


Как SIMD это делает?

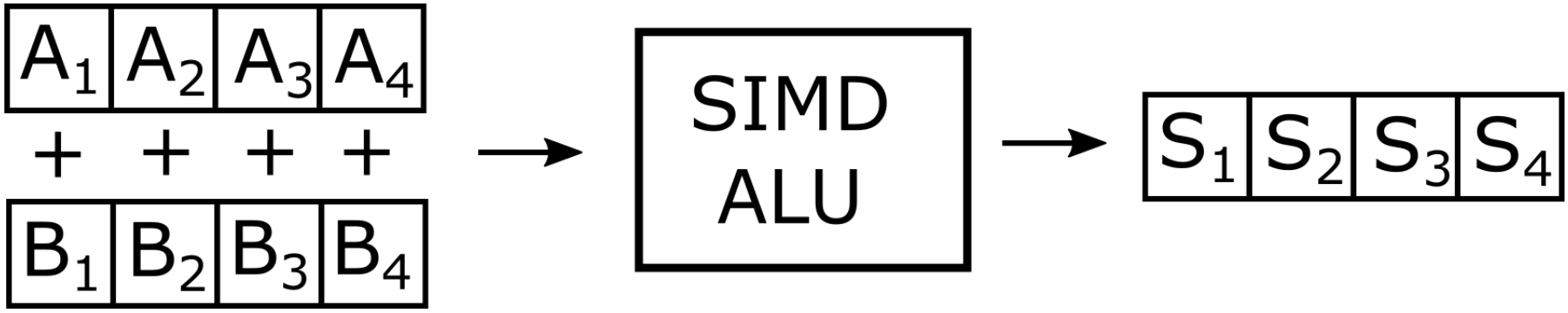
Обычное ALU



SIMD ALU

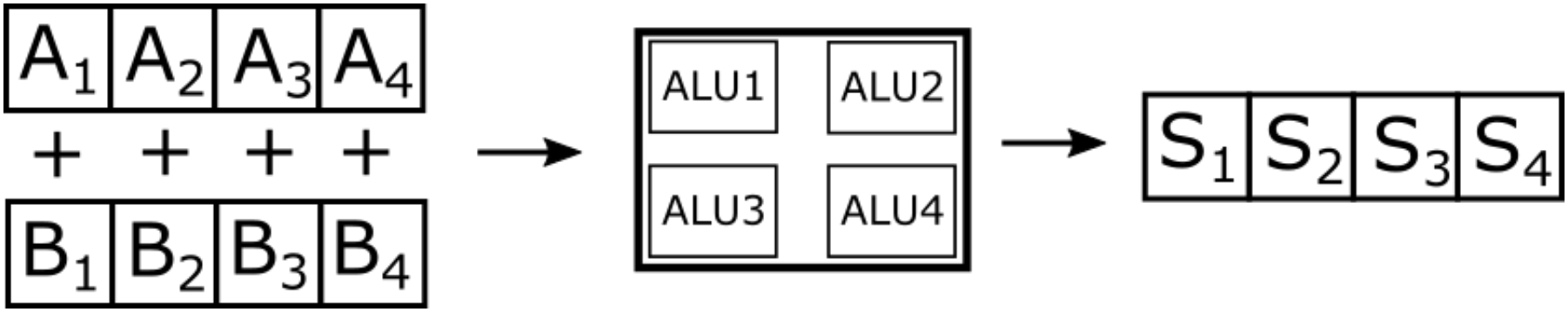


SIMD ALU



В чем секрет?

SIMD ALU



SIMD в процессоре

- SIMD в процессорах Intel
  - Операции над 128-битными векторами (SSE2, SSSE3, SSE4.x)
  - Операции над 256-битными векторами (AVX, AVX 2.0)
  - Современные процессоры обычно поддерживают все перечисленные расширения

Что это значит?

SIMD в процессоре

- SIMD в процессорах Intel
  - Операции над 128-битными векторами (SSE2, SSSE3, SSE4.x)
  - Операции над 256-битными векторами (AVX, AVX 2.0)
  - Современные процессоры обычно поддерживают все перечисленные расширения

Какого ускорения в теории можно добиться?

```
int[] data = ...;
int sum = 0;
for (int i = 0; i < 128; i++)
        sum += data[i];
```

SSE	AVX
?	?

SIMD в процессоре

- SIMD в процессорах Intel
  - Операции над 128-битными векторами (SSE2, SSSE3, SSE4.x)
  - Операции над 256-битными векторами (AVX, AVX 2.0)
  - Современные процессоры обычно поддерживают все перечисленные расширения

Какого ускорения **в теории** можно добиться?

```
int[] data = ...;
int sum = 0;
for (int i = 0; i < 128; i++)
        sum += data[i];
```

SSE	AVX
В **4 раза** быстрее	В **8 раз** быстрее

ТЕХНИЧЕСКИЕ АСПЕКТЫ

SIMD регистры

Выравнивание памяти

Нет времени объяснять, пошли смотреть примеры

SIMD регистры

- Во все SIMD вычисления вовлечены специальные регистры
- 16 штук 128-битных регистров (XMM0-XMM15)
- 16 штук 256-битных регистров (YMM0-YMM15)

- Важно понимать, что их **конечное** число
  - Если в процессе вычислений получается много временных переменных – может произойти замедление программы

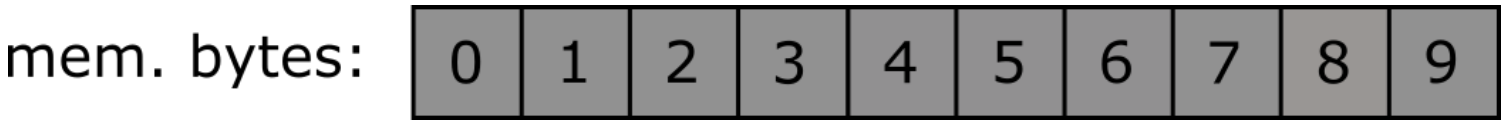
Выравнивание памяти

- Контроллер памяти – чтение/запись данных в памяти.
  - Основная единица работы с памятью – машинное слово
  - Оперирует данными, адрес которых **кратен** размеру машинного слова.

Выравнивание памяти

- Контроллер памяти – чтение/запись данных в памяти.
  - Основная единица работы с памятью – машинное слово
  - Оперирует данными, адрес которых **кратен** размеру машинного слова.

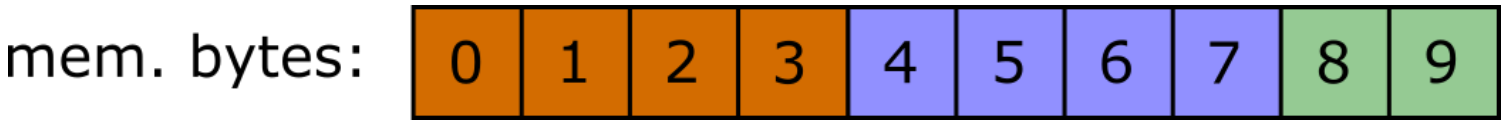
Так мы можем представить себе ячейки памяти



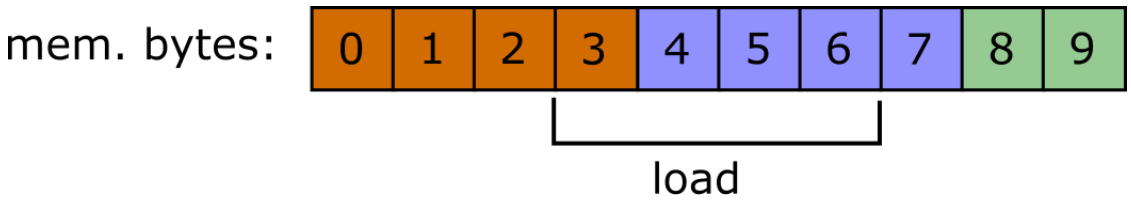
Выравнивание памяти

- Контроллер памяти – чтение/запись данных в памяти.
  - Основная единица работы с памятью – машинное слово
  - Оперирует данными, адрес которых **кратен** размеру машинного слова.

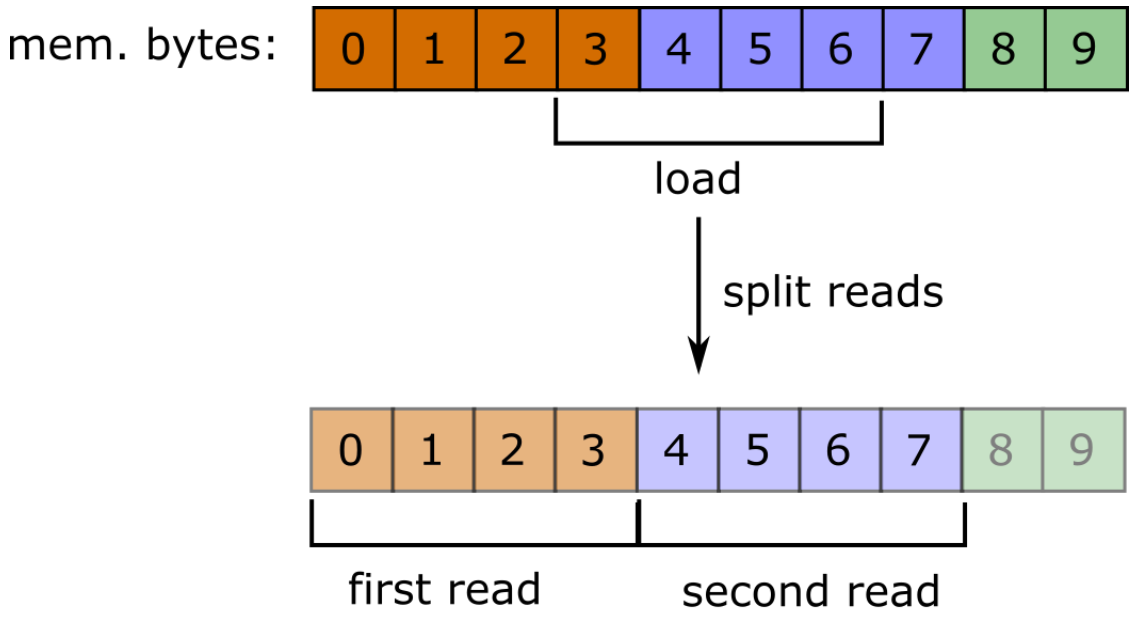
Но контроллер видит их по-другому



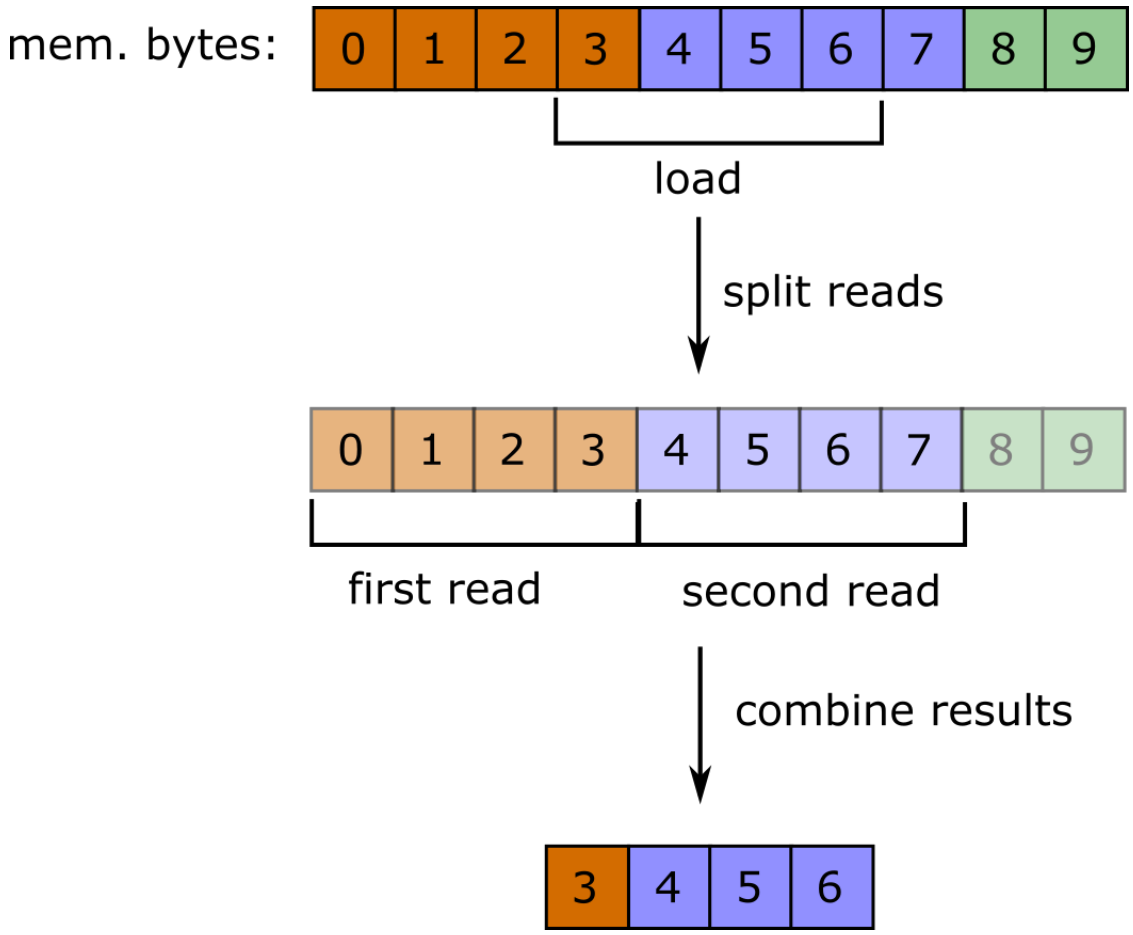
Чтение по невыровненному адресу



Чтение по невыровненному адресу



Чтение по невыровненному адресу



Выравнивание и SIMD

- Старые SIMD расширения часто требовали выровненности обрабатываемых данных
- Новые допускают работу с невыровненными данными (AVX например)
- Однако это может повлечь снижение производительности
  - Из-за специфики работы процессора, контроллера памяти и т.п.

И что делать?

Выравнивание и SIMD

- Старые SIMD расширения часто требовали выровненности обрабатываемых данных
- Новые допускают работу с невыровненными данными (AVX например)
- Однако это может повлечь снижение производительности
  - Из-за специфики работы процессора, контроллера памяти и т.п.

Если вы пишете на языке, имеющем инструменты ручного выравнивания – не забывайте про такую особенность SIMD расширений.

SIMD И C#

Немного истории

Требования

Примеры

SIMD в C#

- Поддержка SIMD в C# появилась относительно недавно – в 2014 году
- Такая задержка связана особенностями CLR
- Если нужна полная мощь и эффективность от SIMD расширений – стоит посмотреть в сторону других языков

Требования к C#

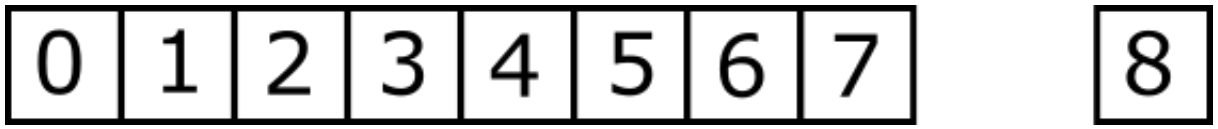
- **RyuJIT** компилятор
- .NET версии $\geq$ **4.6**
- 64-битное приложение
- Необходимые для работы инструменты находятся в **System.Numerics.Vectors**

Простой подход к векторизации кода

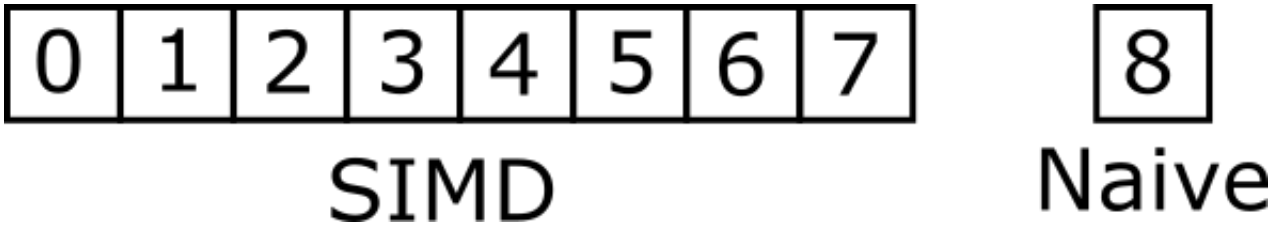
1. Отбросить немного данных
   - Чтобы остаток имел размер кратный размеру SIMD регистра
2. Обработать основной массив данных с помощью SIMD расширений
3. Наивным образом обработать откинутую часть данных
4. Скомбинировать результаты, если нужно

Простой подход к векторизации кода

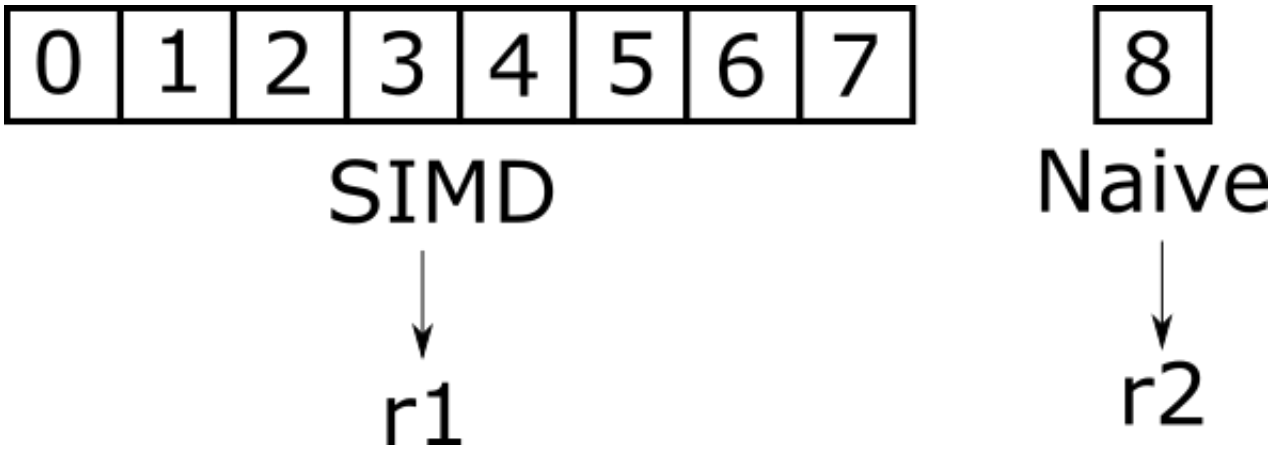
Простой подход к векторизации кода



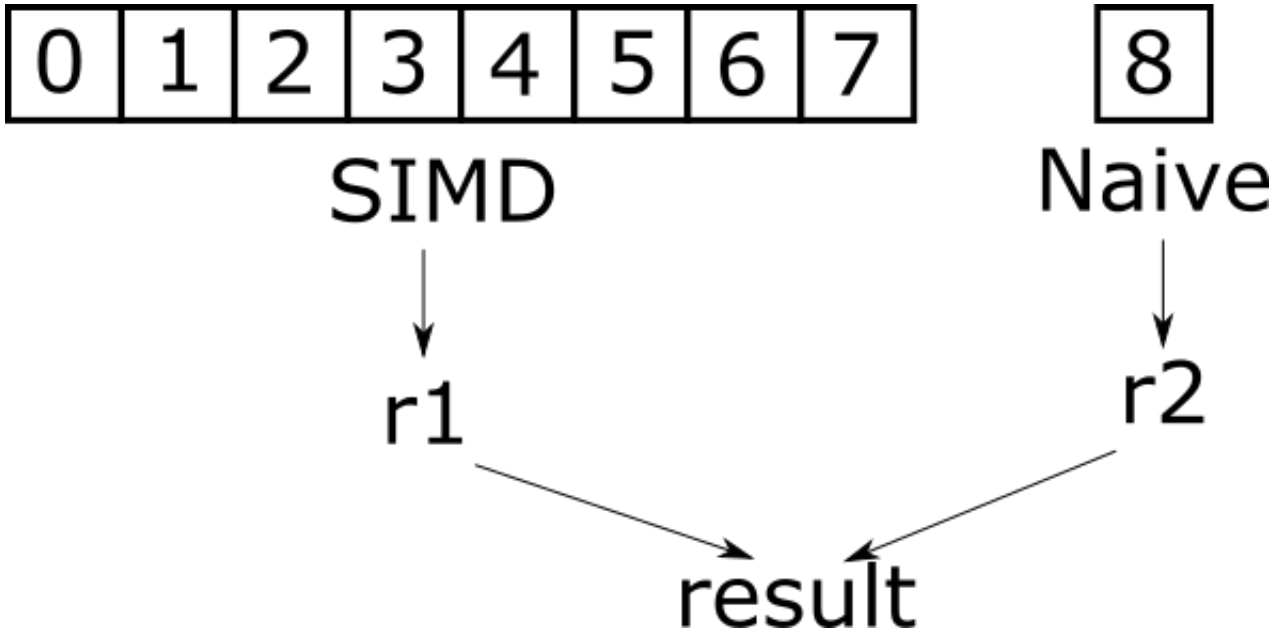
Простой подход к векторизации кода



Простой подход к векторизации кода



Простой подход к векторизации кода



Перед тем как векторизовать код

- Хорошо покройте его тестами. Тесты должны включать в себя:
  - Тесты на **маленьком** объеме данных (массив длины 2)
  - Тесты на **большом** объеме данных (массив длины 100)
  - Тесты на **большом** объеме данных **нечетного размера** (массив длины 55)
- Напишите нормальный **benchmark** для будущих реализаций
  - Не нужно мерять время исполнения “на глаз” или “на stopwatch”

ПРИМЕР 1: A + B

ПРИМЕР 1: A + B + C + …

Задача

- Дан массив из 100 миллионов чисел типа `long`
- Посчитать их сумму

- Данные хранятся в некотором массиве `long[] data`

Подготавливаемся

- Написали хорошие тесты
- Подготовили инфраструктуру для бенчмарка
- Будем сравнивать векторизованную версию с эталонной не векторизованной:

```
public long Sum()
{
    return data.Sum();
}
```

Пока что все просто. Да?

Подготавливаемся

- Написали хорошие тесты
- Подготовили инфраструктуру для бенчмарка
- Будем сравнивать векторизованную версию с эталонной не векторизованной:

```
public long Sum()
{
    return data.Sum();
}
```

Эта реализация далека от оптимальной!

Не совсем.

Подготавливаемся

- Написали хорошие тесты
- Подготовили инфраструктуру для бенчмарка
- Будем сравнивать векторизованную версию с эталонной не векторизованной:

```
public long SimpleForSum()
{
    long sum = 0;
    for (int i = 0; i < data.Length; i++)
        sum += data[i];
    return sum;
}
```

Так лучше?

Foreach vs For-loop vs LINQ

```
public long LinqSum()
{
    return data.Sum();
}
```

715.3 ms

```
public long SimpleForSum()
{
    long sum = 0;
    for (int i = 0; i < data.Length; i++)
        sum += data[i];
    return sum;
}
```

91.0 ms

```
public long ForeachSum()
{
    long sum = 0;
    foreach (var element in data)
        sum += element;
    return sum;
}
```

85.1 ms

Эталонные решения для сравнения

```
public long SimpleForSum()
{
    long sum = 0;
    for (int i = 0; i < data.Length; i++)
        sum += data[i];
    return sum;
}
```

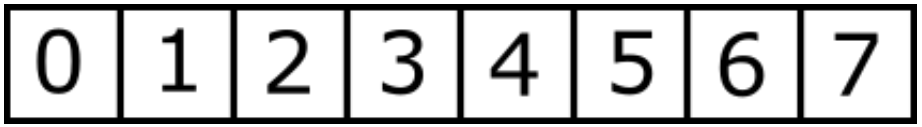
91.0 ms

```
public long ForeachSum()
{
    long sum = 0;
    foreach (var element in data)
        sum += element;
    return sum;
}
```

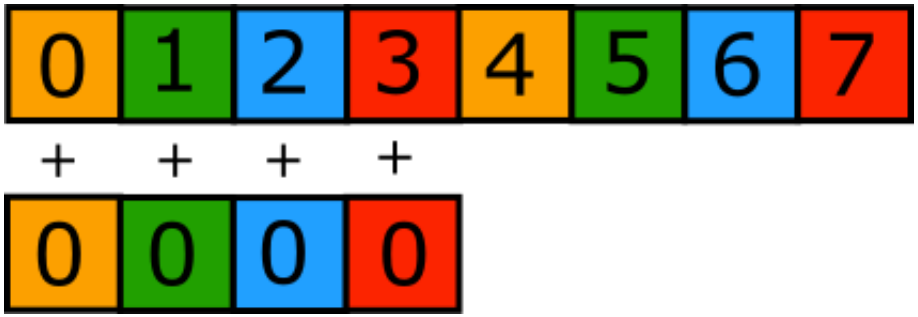
85.1 ms

Как решать с помощью SIMD?

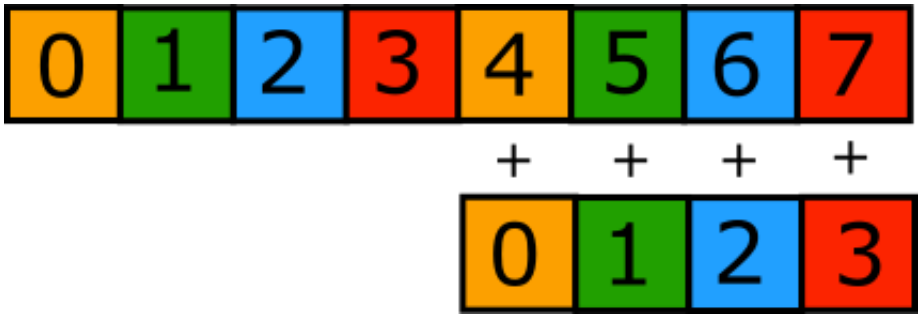
Как решать с помощью SIMD?



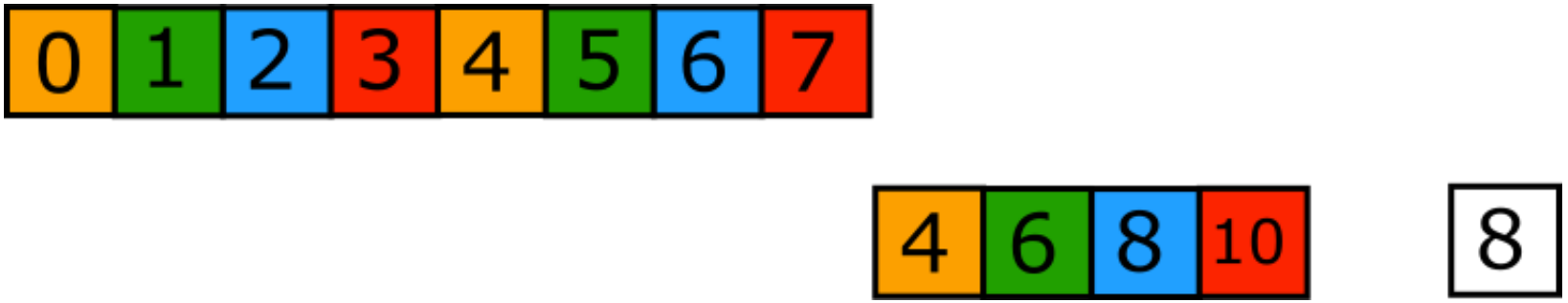
Как решать с помощью SIMD?



Как решать с помощью SIMD?



Как решать с помощью SIMD?



Пишем векторизованную версию

Не пугайтесь

Пишем векторизованную версию

```
var sums = Vector<long>.Zero;
var vectorSize = Vector<long>.Count;
for (int i = 0; i < data.Length; i += vectorSize)
    sums += new Vector<long>(data, i);
long sum = 0;
for (int i = 0; i < vectorSize; i++)
    sum += sums[i];
return sum;
```

Пишем векторизованную версию

```
var sums = Vector<long>.Zero;
var vectorSize = Vector<long>.Count;
for (int i = 0; i < data.Length; i += vectorSize)
    sums += new Vector<long>(data, i);
long sum = 0;
for (int i = 0; i < vectorSize; i++)
    sum += sums[i];
return sum;
```

`Vector<T>` a:

- Хранит несколько экземпляров структуры `T`
- “Автоматически определяет” наличие SIMD расширений создает вектор из нулей типа long
- `Vector<long>.Zero` создает вектор из нескольких нулей типа `long`
  - Количество зависит от доступных расширений. У меня 8 нулей (AVX 2.0)

Пишем векторизованную версию

```
var sums = Vector<long>.Zero;
var vectorSize = Vector<long>.Count;
for (int i = 0; i < data.Length; i += vectorSize)
    sums += new Vector<long>(data, i);
long sum = 0;
for (int i = 0; i < vectorSize; i++)
    sum += sums[i];
return sum;
```

`Vector<T>.Count`

- Позволяет узнать количество экземпляров структуры `T`, которое будет хранится в векторе
- Даже если компилятор не может использовать SIMD расширения, то хранится будет скорее всего более одного элемента

Пишем векторизованную версию

```
var sums = Vector<long>.Zero;
var vectorSize = Vector<long>.Count;
for (int i = 0; i < data.Length; i += vectorSize)
    sums += new Vector<long>(data, i);
long sum = 0;
for (int i = 0; i < vectorSize; i++)
    sum += sums[i];
return sum;
```

```
new Vector<long>(array, offset)
```

- Создает вектор из данных массива `array` начиная с позиции `offset`
- Если элементов меньше, то будет создан вектор меньшего размера

Пишем векторизованную версию

```
var sums = Vector<long>.Zero;
var vectorSize = Vector<long>.Count;
for (int i = 0; i < data.Length; i += vectorSize)
    sums += new Vector<long>(data, i);
long sum = 0;
for (int i = 0; i < vectorSize; i++)
    sum += sums[i];
return sum;
```

Не так сложно, как могло показаться

Запускаем benchmark

for loop	foreach	simd for
91.0 ms	85.1 ms	?

Ожидание: ускорение **в 4 раза**

Запускаем benchmark

for loop	foreach	simd for
91.0 ms	85.1 ms	79.9 ms

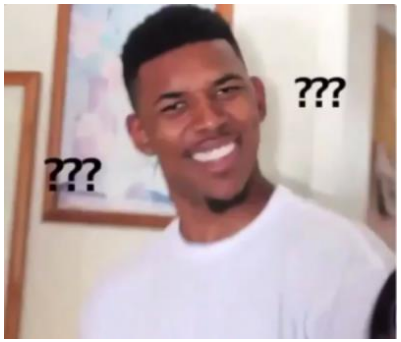
Ожидание: ускорение **в 4 раза**

Запускаем benchmark

for loop	foreach	simd for
91.0 ms	85.1 ms	79.9 ms

Ожидание: ускорение **в 4 раза**

Реальность: ускорение **на 6%**



Мое объяснение ситуации

- Код достаточно простой и хорошо оптимизируется сам по себе компилятором
- Слаженная работа конвейера, предсказателя ветвлений, кэша инструкций и других элементов процессора делают этот код еще быстрее
- Невыровненность данных сильно сказывается на производительности

Извлекаем уроки

- SIMD далеко не всегда ускоряет код во столько раз, во сколько вы ожидаете
- Важно сделать всевозможно оптимизации и максимально ускорить код, который есть у вас сейчас
  - Возможно после этого вам даже не понадобится SIMD

ПРИМЕР 2: КОЛИЧЕСТВО ЧИСЕЛ ИЗ ОТРЕЗКА

Задача

- Дан массив из 100 миллионов чисел типа `int`
- Посчитать количество чисел лежащих в отрезке `l..r`

- Данные хранятся в некотором массиве `int[] data`
- `l, r` заданы

Эталонные решения

```
public int SimpleForCountInRange()
{
    var count = 0;
    for (int i = 0; i < data.Length; i++)
        if (L <= data[i] && data[i] <= R)
            count++;
    return count;
}
```

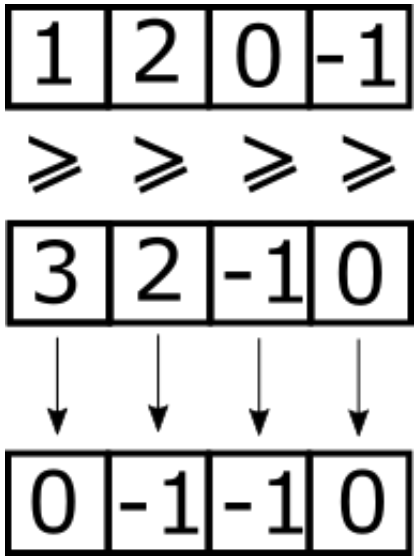
496.7 ms

```
public int ForeachCountInRange()
{
    var count = 0;
    foreach (var element in data)
        if (L <= element && element <= R)
            count++;
    return count;
}
```

479.7 ms

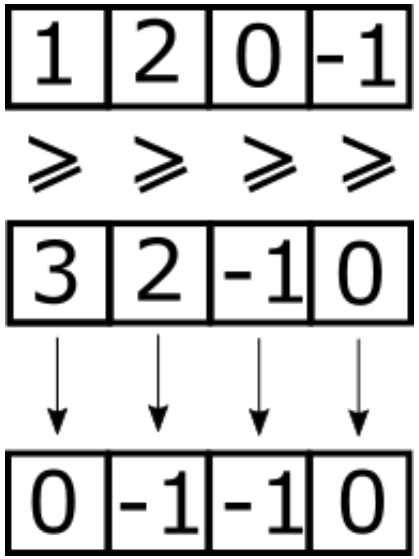
Как решать с помощью SIMD?

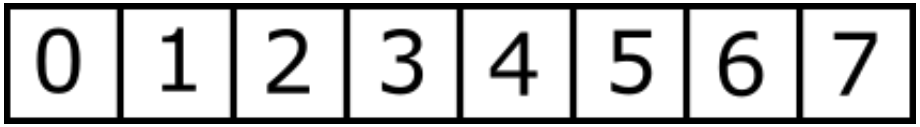
- В SSE/AVX есть операция, которая покомпонентно сравнивает два вектора



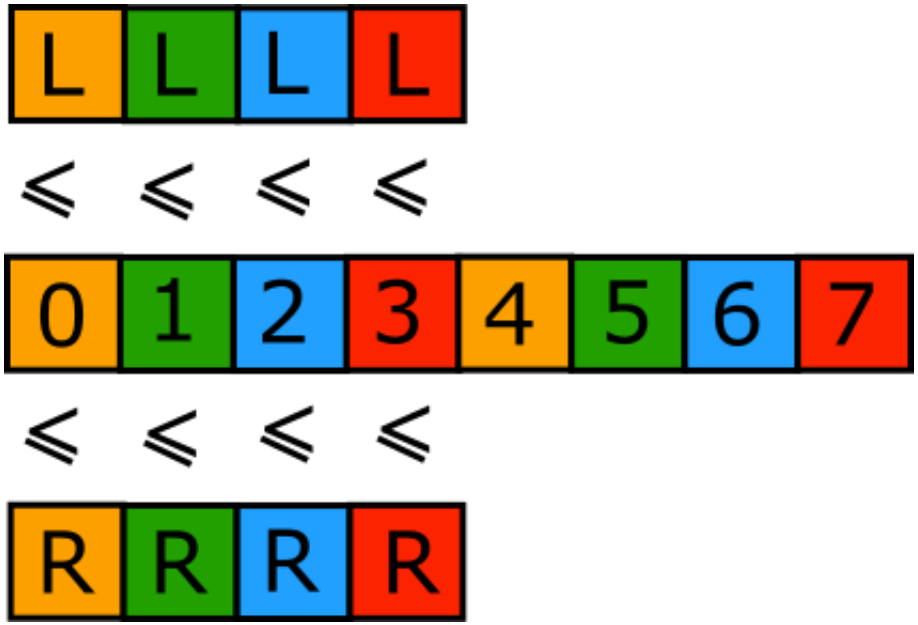
Как решать с помощью SIMD?

- В SSE/AVX есть операция, которая покомпонентно сравнивает два вектора

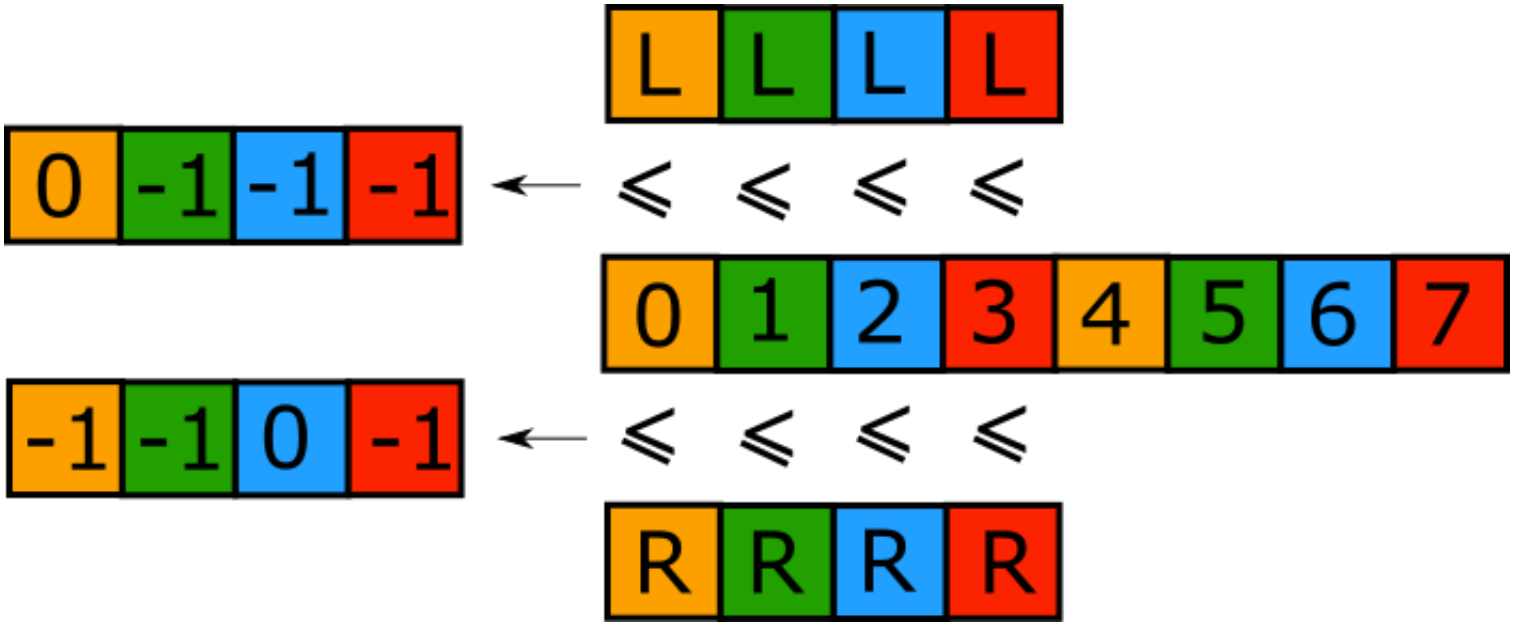




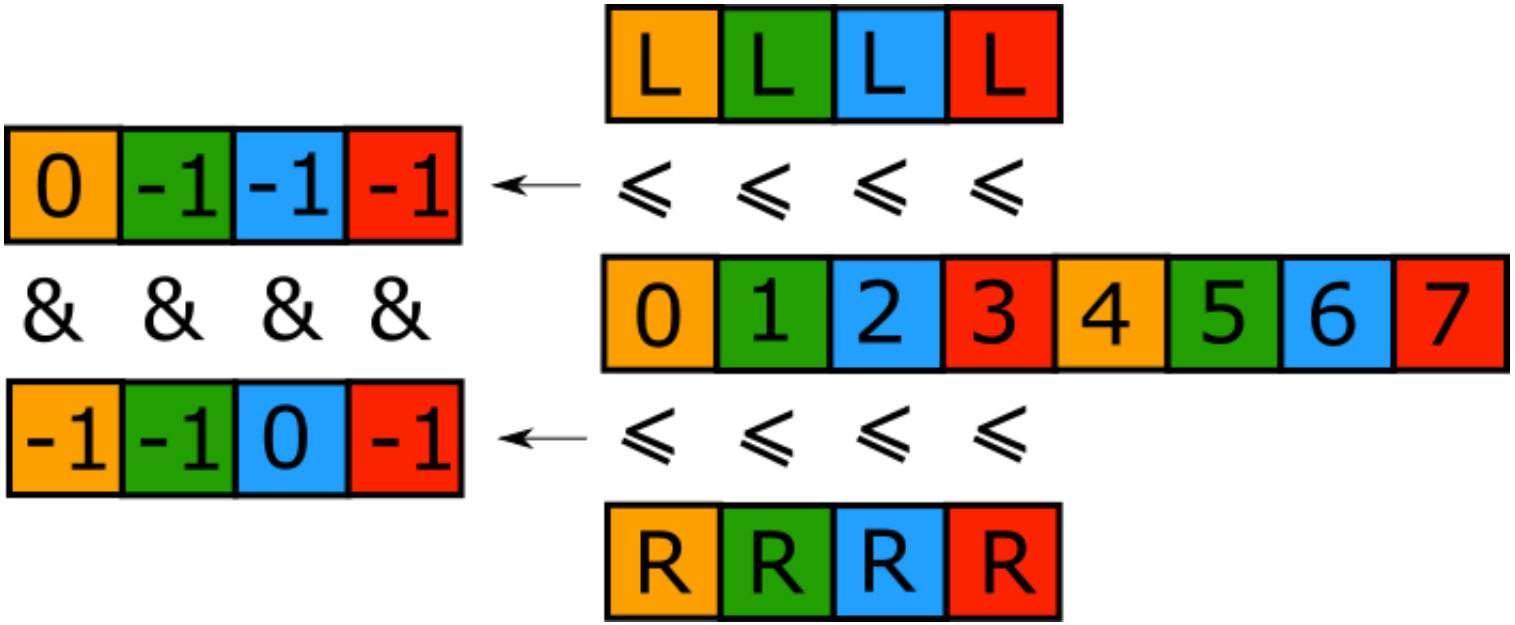
Как решать с помощью SIMD?



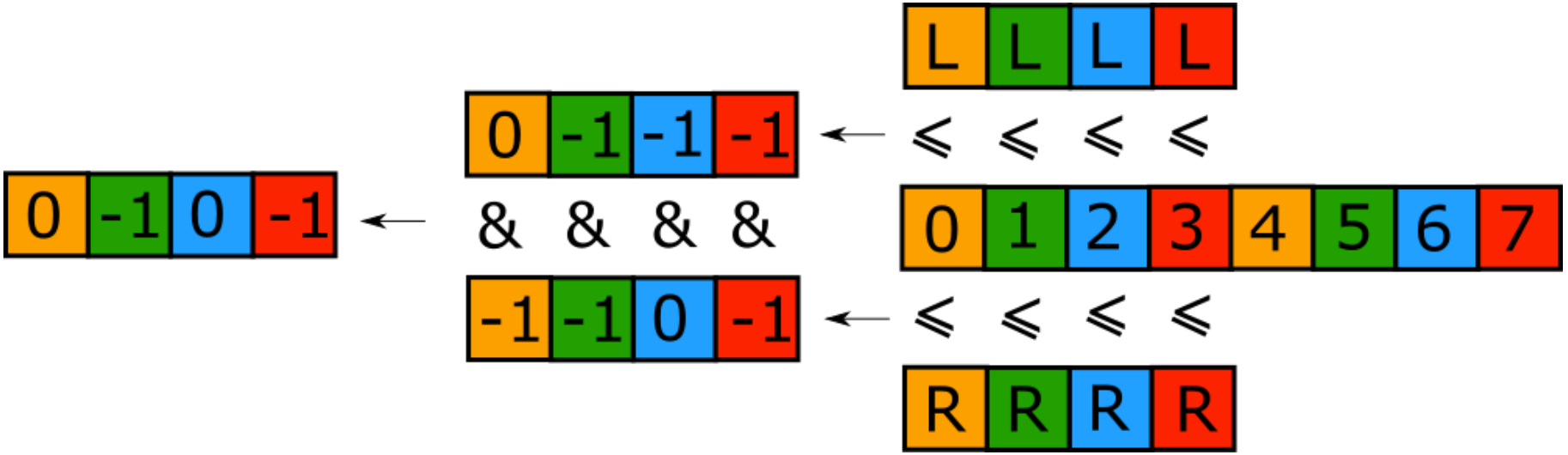
Как решать с помощью SIMD?



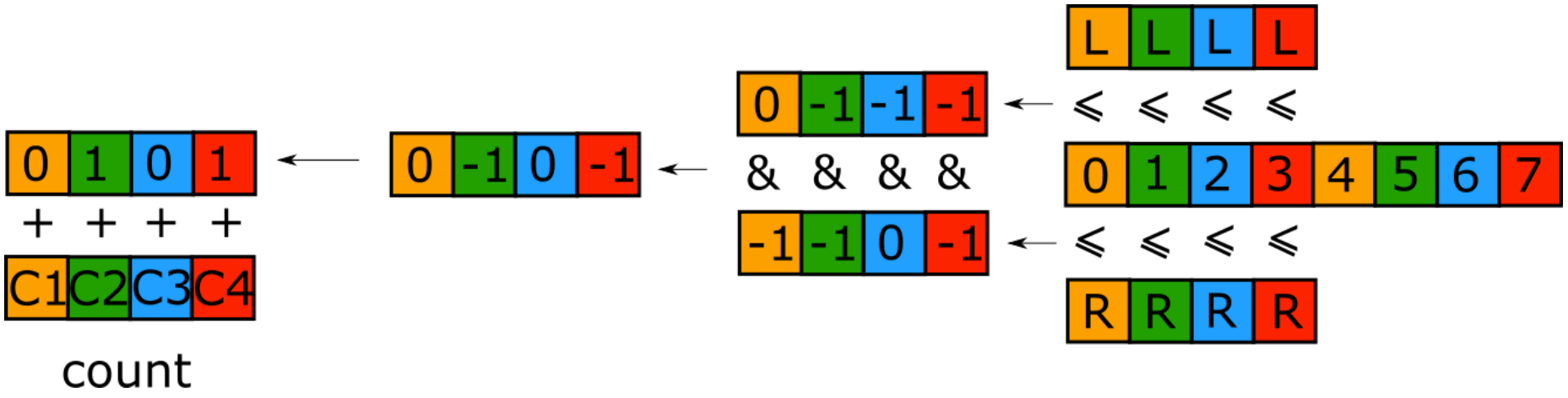
Как решать с помощью SIMD?



Как решать с помощью SIMD?



Как решать с помощью SIMD?



Реализация

```
var left = new Vector<int>(L);
var right = new Vector<int>(R);
var count = Vector<int>.Zero;
for (int i = 0; i < data.Length; i += vectorSize)
{
    var slice = new Vector<int>(data, i);
    var result =
        Vector.LessThanOrEqual(left, slice) &
        Vector.LessThanOrEqual(slice, right);
    result = Vector.Negate(result);
    count = count + result;
}
// return (сумма элементов вектора count);
```

Реализация

```
var left = new Vector<int>(L);
var right = new Vector<int>(R);
var count = Vector<int>.Zero;
for (int i = 0; i < data.Length; i += vectorSize)
{
    var slice = new Vector<int>(data, i);
    var result =
        Vector.LessThanOrEqual(left, slice) &
        Vector.LessThanOrEqual(slice, right);
    result = Vector.Negate(result);
    count = count + result;
}
// return (сумма элементов вектора count);
```

Реализация

```
var left = new Vector<int>(L);
var right = new Vector<int>(R);
var count = Vector<int>.Zero;
for (int i = 0; i < data.Length; i += vectorSize)
{
    var slice = new Vector<int>(data, i);
    var result =
        Vector.LessThanOrEqual(left, slice) &
        Vector.LessThanOrEqual(slice, right);
    result = Vector.Negate(result);
    count = count + result;
}
// return (сумма элементов вектора count);
```

Реализация

```
var left = new Vector<int>(L);
var right = new Vector<int>(R);
var count = Vector<int>.Zero;
for (int i = 0; i < data.Length; i += vectorSize)
{
    var slice = new Vector<int>(data, i);
    var result =
        Vector.LessThanOrEqual(left, slice) &
        Vector.LessThanOrEqual(slice, right);
    result = Vector.Negate(result);
    count = count + result;
}
// return (сумма элементов вектора count);
```

Реализация

```
var left = new Vector<int>(L);
var right = new Vector<int>(R);
var count = Vector<int>.Zero;
for (int i = 0; i < data.Length; i += vectorSize)
{
    var slice = new Vector<int>(data, i);
    var result =
        Vector.LessThanOrEqual(left, slice) &
        Vector.LessThanOrEqual(slice, right);
    result = Vector.Negate(result);
    count = count + result;
}
// return (сумма элементов вектора count);
```

Реализация

```
var left = new Vector<int>(L);
var right = new Vector<int>(R);
var count = Vector<int>.Zero;
for (int i = 0; i < data.Length; i += vectorSize)
{
    var slice = new Vector<int>(data, i);
    var result =
        Vector.LessThanOrEqual(left, slice) &
        Vector.LessThanOrEqual(slice, right);
    result = Vector.Negate(result);
    count = count + result;
}
// return (сумма элементов вектора count);
```

Реализация

```
var left = new Vector<int>(L);
var right = new Vector<int>(R);
var count = Vector<int>.Zero;
for (int i = 0; i < data.Length; i += vectorSize)
{
    var slice = new Vector<int>(data, i);
    var result =
        Vector.LessThanOrEqual(left, slice) &
        Vector.LessThanOrEqual(slice, right);
    result = Vector.Negate(result);
    count = count + result;
}
// return (сумма элементов вектора count);
```

Тестируем производительность

for loop	foreach	simd for
496.7 ms	479.7 ms	?

Ожидание: ускорение **в 8 раза**

Тестируем производительность

for loop	foreach	simd for
496.7 ms	479.7 ms	43.7 ms

Ожидание: ускорение **в 8 раза**

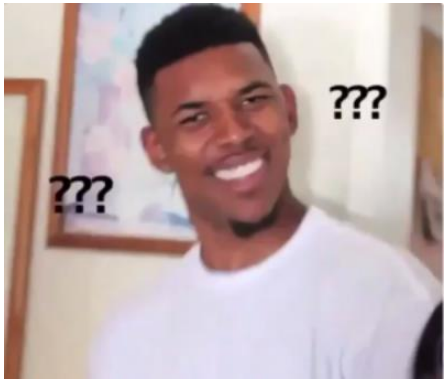
Реальность: ускорение **в 11 раз**

Тестируем производительность

for loop	foreach	simd for
496.7 ms	479.7 ms	43.7 ms

Ожидание: ускорение **в 8 раза**

Реальность: ускорение **в 11 раз**



Объяснение ситуации

- Тело цикла в эталонных решениях содержит условные переходы
- В SIMD коде jump не используется

Найдите неоптимальное место

```
var left = new Vector<int>(L);
var right = new Vector<int>(R);
var count = Vector<int>.Zero;
for (int i = 0; i < data.Length; i += vectorSize)
{
    var slice = new Vector<int>(data, i);
    var result =
        Vector.LessThanOrEqual(left, slice) &
        Vector.LessThanOrEqual(slice, right);
    result = Vector.Negate(result);
    count = count + result;
}
// return (сумма элементов вектора count);
```

Найдите неоптимальное место

```
var left = new Vector<int>(L);
var right = new Vector<int>(R);
var count = Vector<int>.Zero;
for (int i = 0; i < data.Length; i += vectorSize)
{
    var slice = new Vector<int>(data, i);
    var result =
        Vector.LessThanOrEqual(left, slice) &
        Vector.LessThanOrEqual(slice, right);
//  result = Vector.Negate(result);
    count = count + result;
}
count = Vector.Negate(count);
// return (сумма элементов вектора count);
```

Извлекаем уроки

- SIMD способен реально ускорить код
- SIMD алгоритм нужно писать внимательно и аккуратно, чтобы он получился максимально эффективным
  - Например, не нужно делать лишних действий в цикле. Лучше их вынести за его пределы

ПОДВОДИМ ИТОГИ

Когда не нужно использовать SIMD?

Когда нужно?

Когда не нужно использовать SIMD?

- Когда вы чувствуете, что не до конца хорошо владеете технологией
  - Конечно, когда-то нужно начать. Но начинайте с простых примеров
- Когда существуют места, где можно применить другие оптимизации
- Когда вы собираетесь оптимизировать не самое узкое место
- Не стоит заменять обычный параллелизм SIMD расширениями

Когда можно использовать SIMD?

- Когда ни один из предыдущих пунктов не выполнен
- Когда у вас есть алгоритм, который хорошо формулируется в терминах операций над векторами

- Не забываем, что у нас уже должна быть пачка тестов и бенчмарков

Если вы решились векторизовать код

- Будьте готовы к тому, что SIMD расширения не оправдают ваших ожиданий
- Будьте готовы работать с низкоуровневым C#
- Будьте готовы к тому, что в C# SIMD-расширения недостаточно быстрые. В C++ быстрее
- Будьте готовы к тому, что в итоге оптимизации не принесут существенного выигрыша
- Будьте готовы пожертвовать незначительными оптимизациями в пользу читаемости кода

Если вы решились векторизовать код

- Будьте готовы к тому, что SIMD расширения не оправдают ваших ожиданий
- Будьте готовы работать с низкоуровневым C#
- Будьте готовы к тому, что в C# SIMD-расширения недостаточно быстрые. В C++ быстрее
- Будьте готовы к тому, что в итоге оптимизации не принесут существенного выигрыша
- Будьте готовы пожертвовать незначительными оптимизациями в пользу читаемости кода

Если вы решились векторизовать код

- Будьте готовы к тому, что SIMD расширения не оправдают ваших ожиданий
- Будьте готовы работать с низкоуровневым C#
- Будьте готовы к тому, что в C# SIMD-расширения недостаточно быстрые. В C++ быстрее
- Будьте готовы к тому, что в итоге оптимизации не принесут существенного выигрыша
- Будьте готовы пожертвовать незначительными оптимизациями в пользу читаемости кода

Если вы решились векторизовать код

- Будьте готовы к тому, что SIMD расширения не оправдают ваших ожиданий
- Будьте готовы работать с низкоуровневым C#
- Будьте готовы к тому, что в C# SIMD-расширения недостаточно быстрые. В C++ быстрее
- Будьте готовы к тому, что в итоге оптимизации не принесут существенного выигрыша
- Будьте готовы пожертвовать незначительными оптимизациями в пользу читаемости кода

Если вы решились векторизовать код

- Будьте готовы к тому, что SIMD расширения не оправдают ваших ожиданий
- Будьте готовы работать с низкоуровневым C#
- Будьте готовы к тому, что в C# SIMD-расширения недостаточно быстрые. В C++ быстрее
- Будьте готовы к тому, что в итоге оптимизации не принесут существенного выигрыша
- Будьте готовы пожертвовать незначительными оптимизациями в пользу читаемости кода

Если вы решились векторизовать код

- Будьте готовы к тому, что SIMD расширения не оправдают ваших ожиданий
- Будьте готовы работать с низкоуровневым C#
- Будьте готовы к тому, что в C# SIMD-расширения недостаточно быстрые. В C++ быстрее
- Будьте готовы к тому, что в итоге оптимизации не принесут существенного выигрыша
- Будьте готовы пожертвовать незначительными оптимизациями в пользу читаемости кода

Реализация #1

```
int memcmp(const char *src_1, const char* src_2, int len)
{
        for (int i = 0; i < len; i++){
                if (src_1[i] != src_2[i])
                        return src_1[i] < src_2[i] ? -1 : 1;
        return 0;
}
```

Реализация #2

```
ATTR int memcmp_unaligned_unrolled(const char* src_1, const char*
src_2, int len)
{
        const Vec* ptr1 = (const Vec*)src_1;
        const Vec* ptr2 = (const Vec*)src_2;
        for (int i = 0; i < len; i += 32, ptr1 += 2, ptr2 += 2){
        Vec M11 = _mm_loadu_si128(ptr1);
                Vec M12 = _mm_loadu_si128(ptr2);
                Vec M21 = _mm_loadu_si128(ptr1 + 1);
                Vec M22 = _mm_loadu_si128(ptr2 + 1);
                Vec MX1 = _mm_xor_si128(M11, M12);
                Vec MX2 = _mm_xor_si128(M21, M22);
                MX1 = _mm_or_si128(MX1, MX2);
                if (!_mm_testz_si128(MX1, MX1)){
                        for (int a = 0; a < 32; a++)
                                if (src_1[i + a] != src_2[i + a])
                                        return src_1[i + a] < src_2[i +
a] ? -1 : 1;
                }
        }
        return 0;
}
```

Стоит задуматься

Или порефакторить реализацию #2

Реабилитация SIMD

- Может ускорить ваш код в **десятки** раз
- Нестандартные и интересные задачи
  - Часто векторизация является не такой тривиальной задачей
- В процессе работы начнете понимать недры C# и архитектуры ЭВМ

ВОПРОСЫ?

СПАСИБО ЗА ВНИМАНИЕ