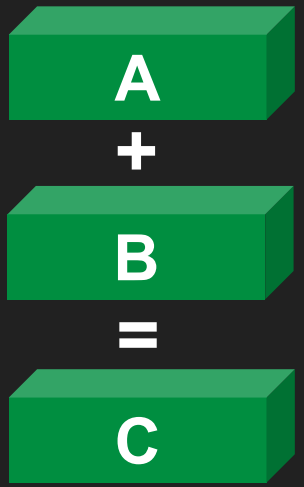
Instruções SIMD para comparação

André Tupinambá

Processamento SIMD

Instruções do processador sobre múltiplos dados

Escalar (SISD)
- 1 instrução
- 1 operação



Vetorial (SIMD)
- 1 instrução
- *n* operações

SIMD além dos cálculos

Inúmeros exemplos e bibliotecas usam SIMD para cálculos

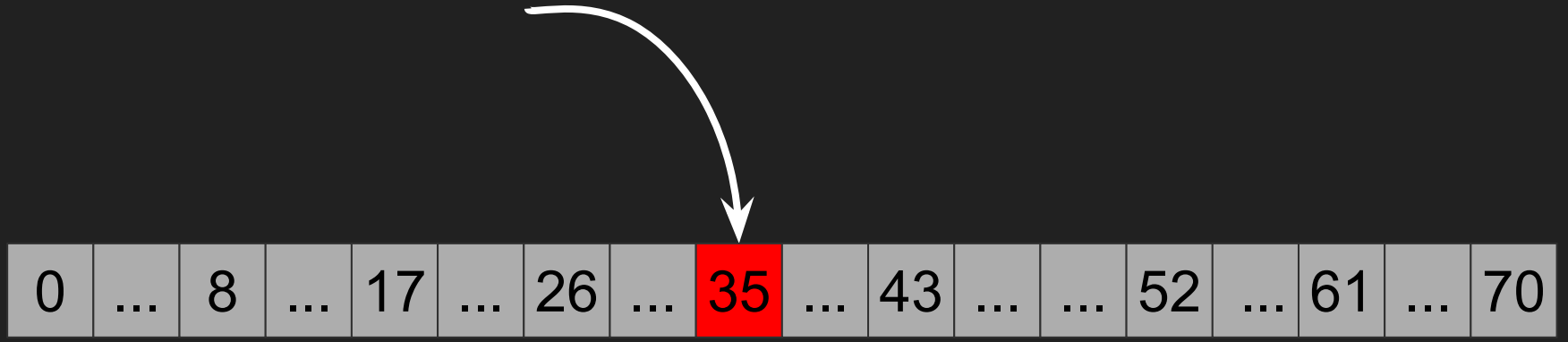
- Álgebra linear, gráficos, som, criptografia, ...

Porém o x86 possui várias outros tipos de instruções SIMD

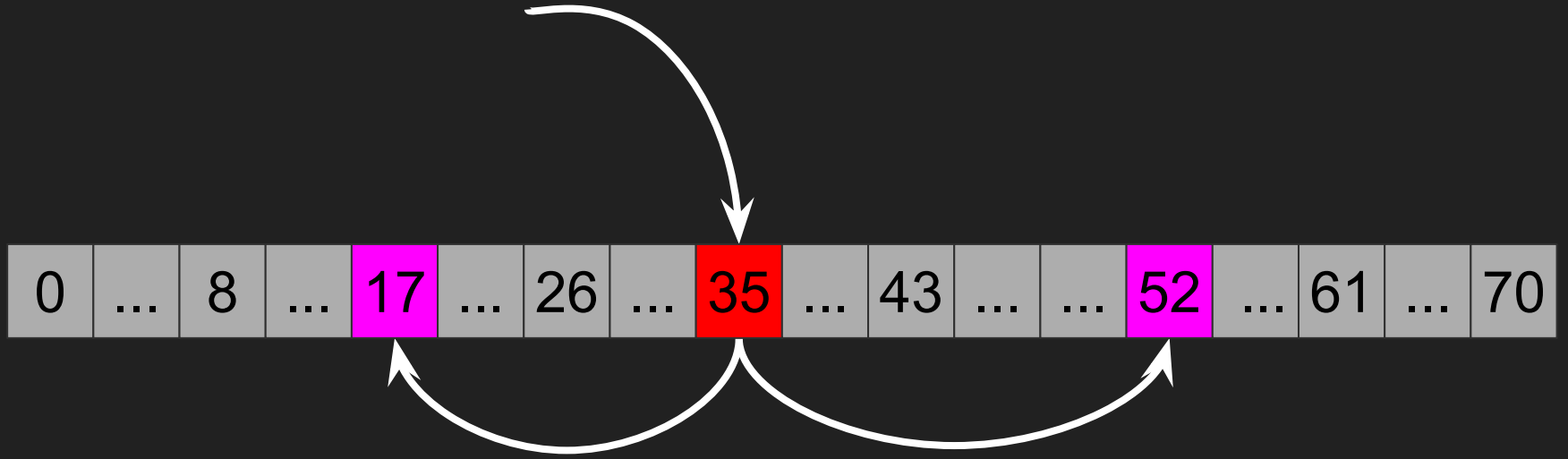
- Como utilizá-las?
- Algumas ideias com as instruções de comparação

Busca binária

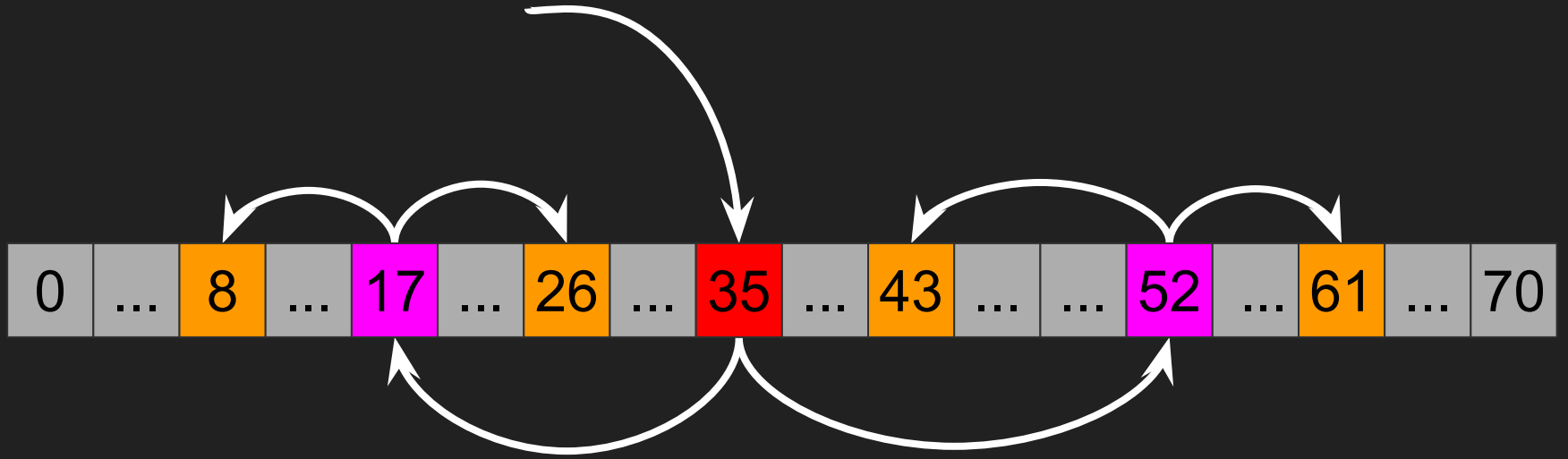
Busca binária



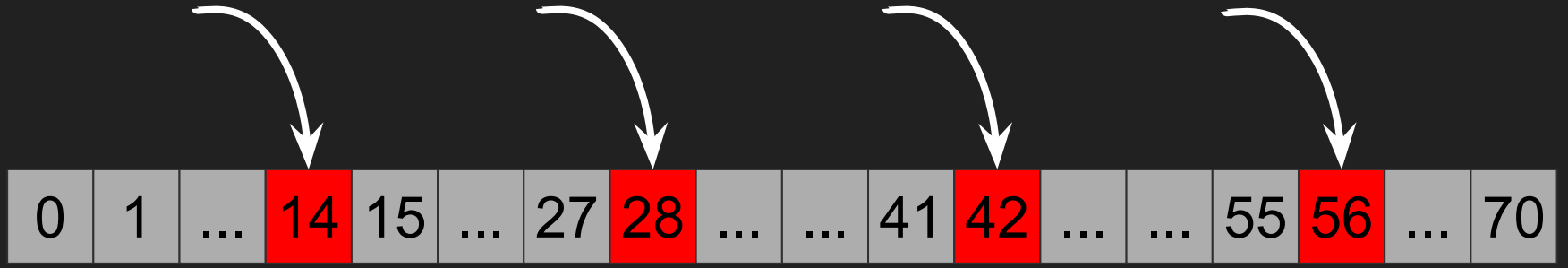
Busca binária



Busca binária

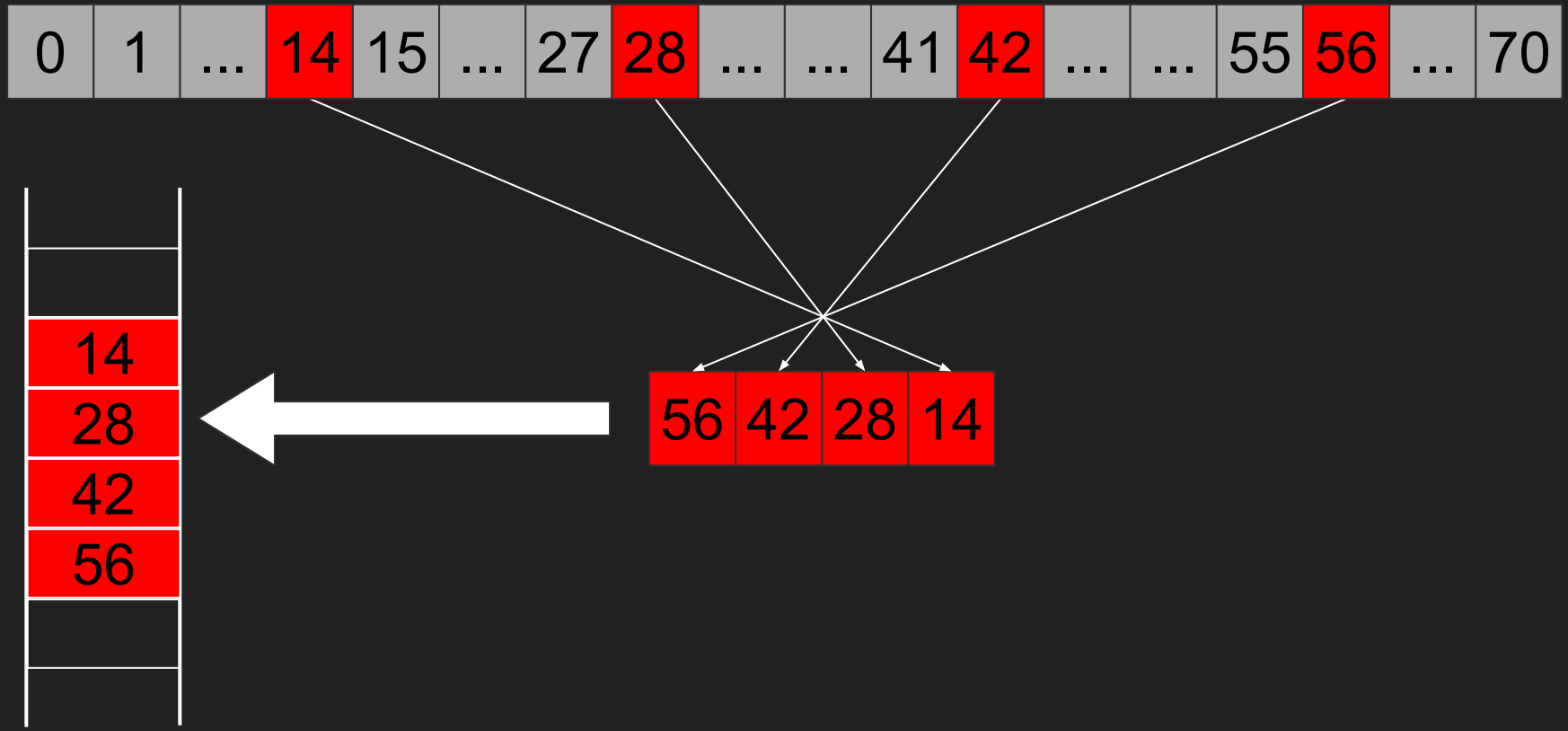


E se?

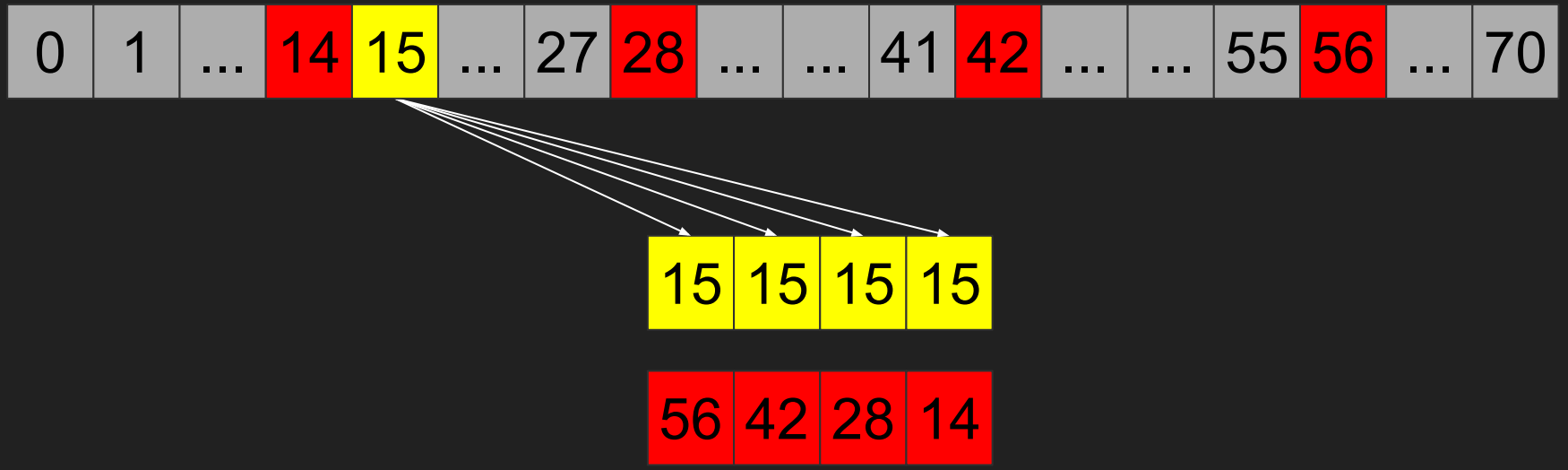


Procura n-Way, estilo SIMD

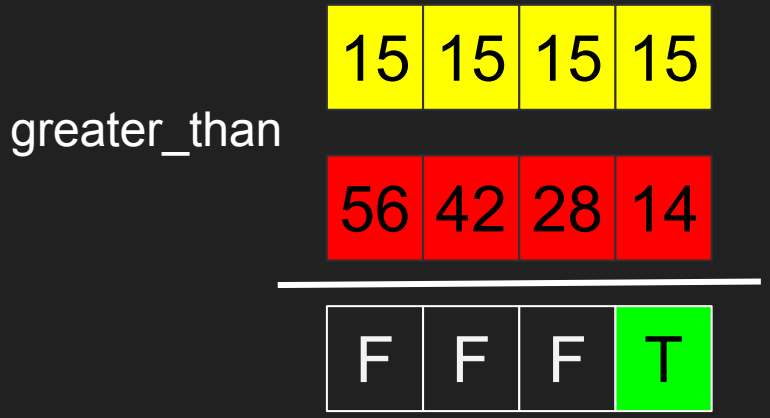
Procura n-Way, estilo SIMD



Procura n-Way, estilo SIMD

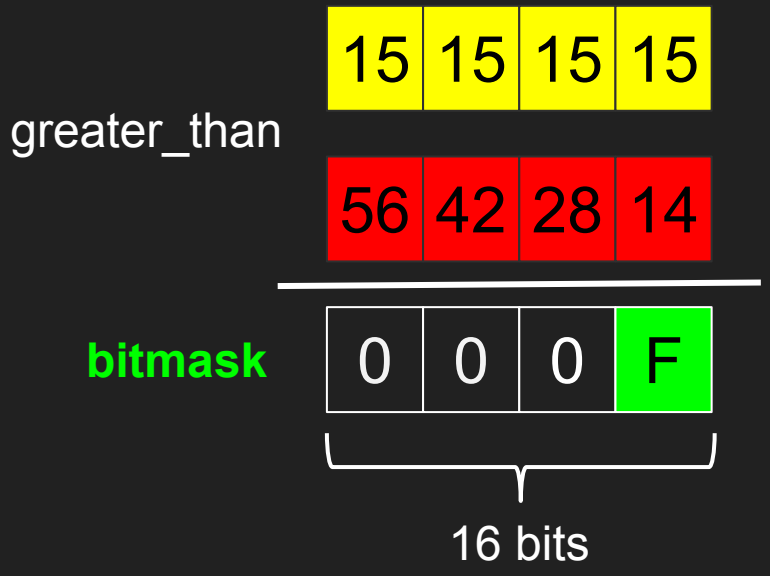


Procura n-Way, estilo SIMD



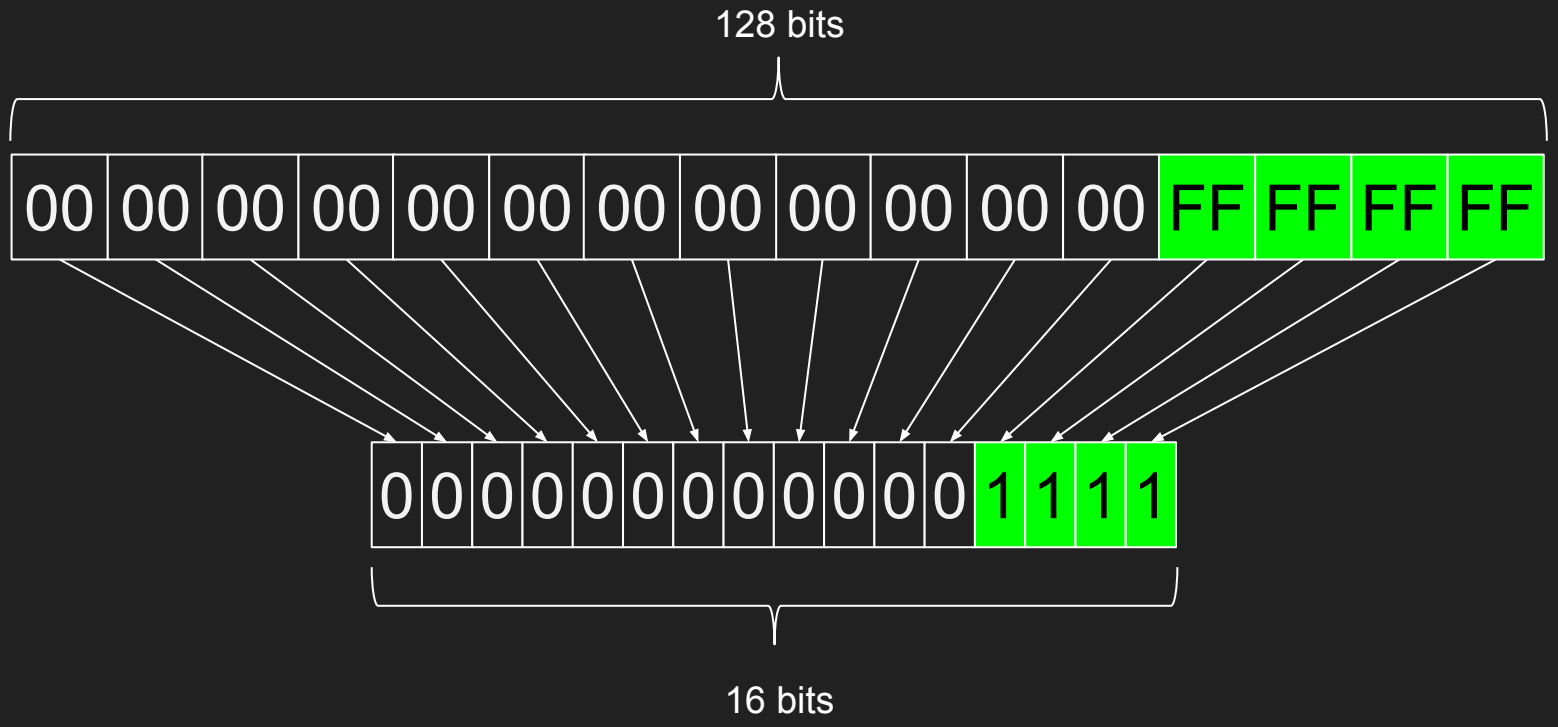
Procura n-Way, estilo SIMD

0	1	...	14	15	...	27	28	...	...	41	42	...	...	55	56	...	70



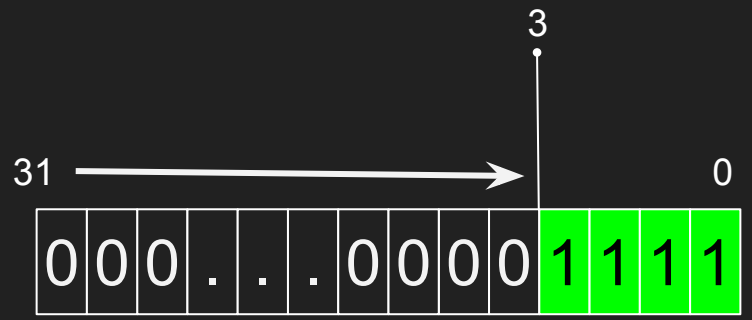
Criando o bitmask

Copia o bit mais alto de cada byte

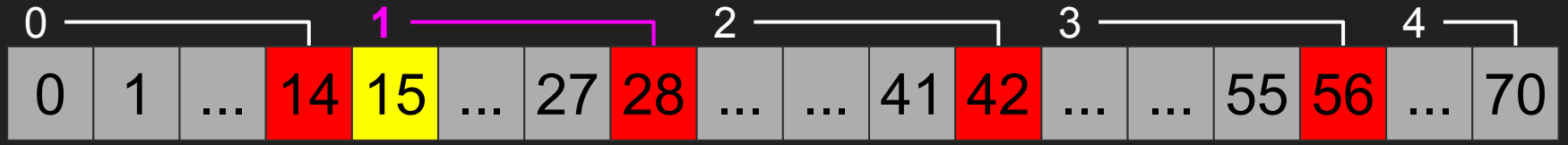


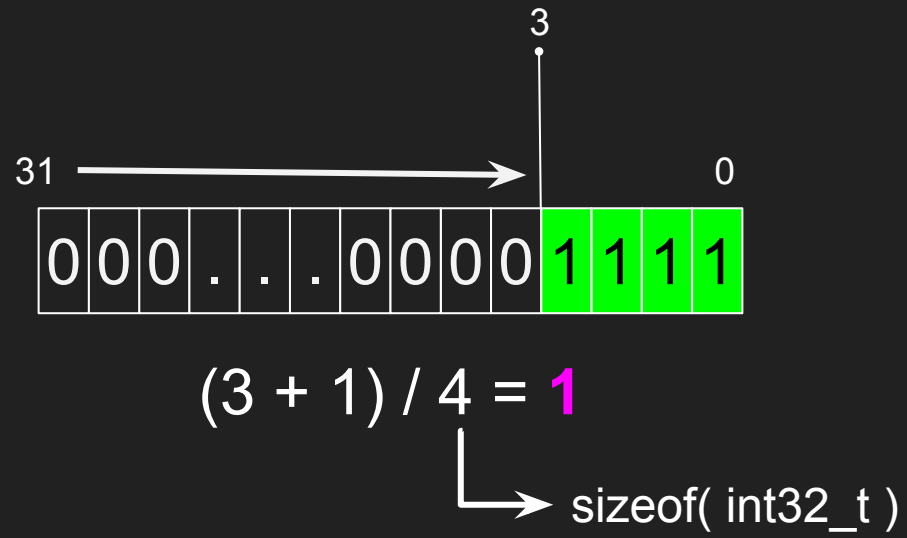
Bit scan reverse

Procura o primeiro bit 1, do mais alto para o mais baixo



Bit scan reverse





SIMD greater than

```
uint32_t greater_than_index( int32_t key,
                             __m128i cmp ) {
   __m128i simdKey = _mm_set1_epi32( key );
   __m128i gtMask = _mm_cmpgt_epi32( simdKey, cmp );
   uint32_t mask = _mm_movemask_epi8( gtMask );
   return ( mask == 0 ) ? 0
          : (_bit_scan_reverse( mask ) + 1) / 4;
}
```

N-Way class

```
class nway {
public:
   using const_iterator = std::vector< int32_t >::const_iterator;
   nway( const std::vector< int32_t >& v ) : vec_(v) {}
   void build_index();
   const_iterator find( int32_t key );

private:
    __m128i cmp_;
    const std::vector< int32_t >& vec_;
    std::array< const_iterator, 4 + 2 > ranges_;
};
```

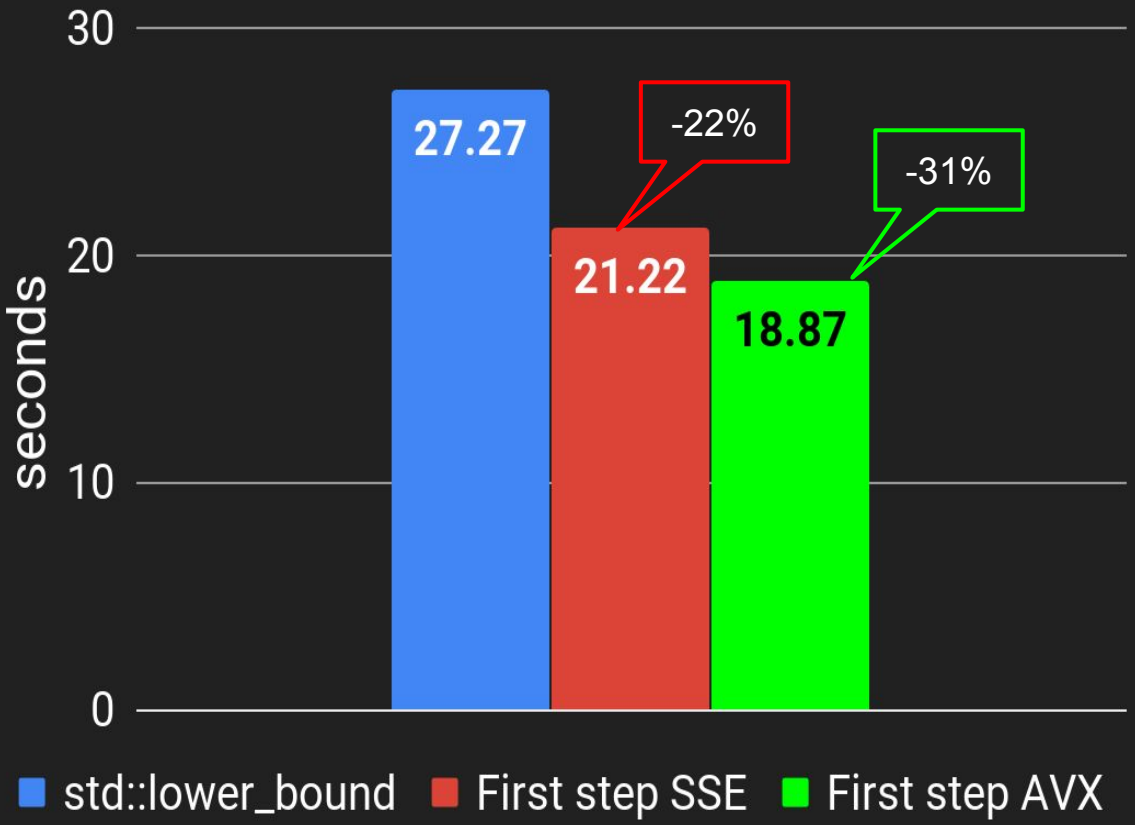
N-Way index

```
void nway::build_index() {
  size_t step = vec_.size() / 5;    // 5 = 1 + SIMD size
  int32_t* pCmp = (int32_t*) &cmp_;
  auto it = vec_.begin();
  ranges_[ 0 ] = it;
  for( size_t i = 0; i < 4; ++i ) { // 4 = SIMD size
    std::advance( it, step );
    ranges_[ i + 1 ] = it;
    pCmp[ i ] = *it;
  }
  ranges_[ 5 ] = std::prev( vec_.end() );
}
```

N-Way find

```
const_iterator nway::find( int32_t key ) {
  uint32_t idx = greater_than_index( key, cmp_ );
  auto end = std::next( ranges_[ idx + 1 ] );
  auto it = std::lower_bound( ranges_[ idx ],
                              end, key );
  return (it!=end && !(key<*it))
              ? it
              : vec_.end();
}
```

Resultados N-Way search



Core i7-4870HQ
MacOSX El Captain 10.11.6
Clang 6.0.1 (homebrew)
-std=c++14 -mavx2 -O3
-fno-strict-aliasing -Wall

std::vector< int32_t >
Size = 0x00400000
Busca todos os dados em ordem randômica por 10 vezes
Tempo médio após 100 execuções

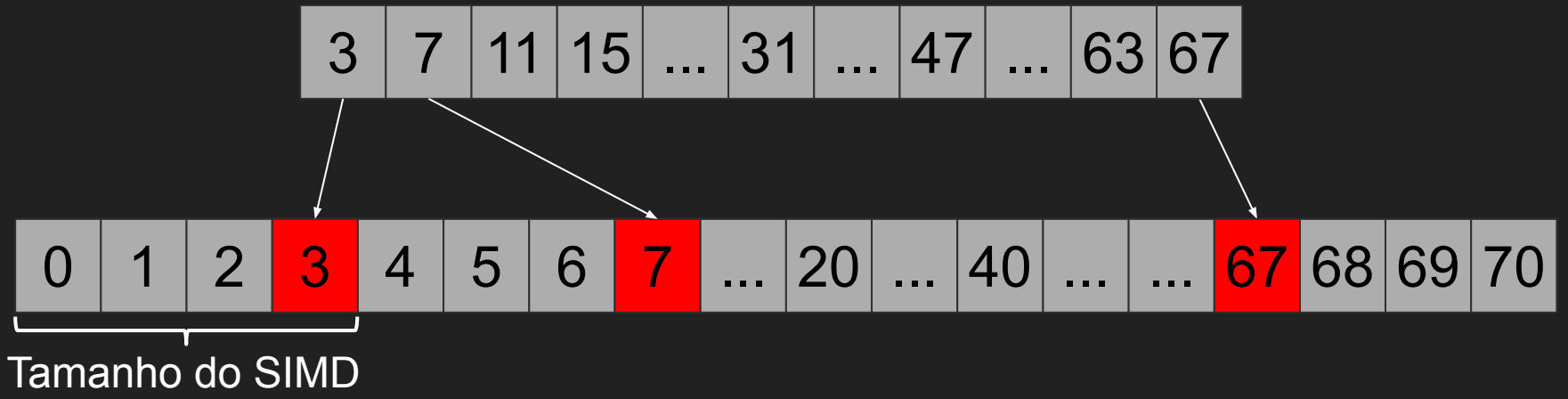
Limitações do SSE/AVX com inteiros

- Somente 2 operações de comparação
  - Maior que
  - Igual

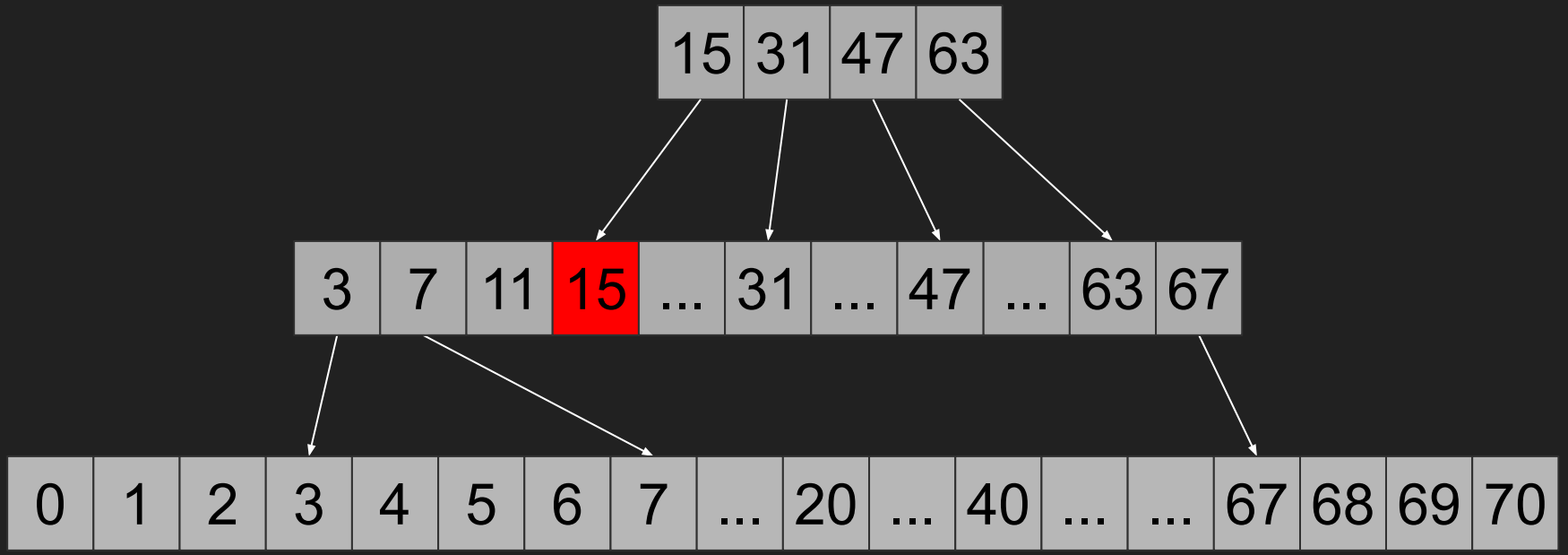
- Somente inteiros com sinal
  - int8_t, int16_t, int32_t, int64_t

Árvore n-Way

Árvore n-Way



Árvore n-Way



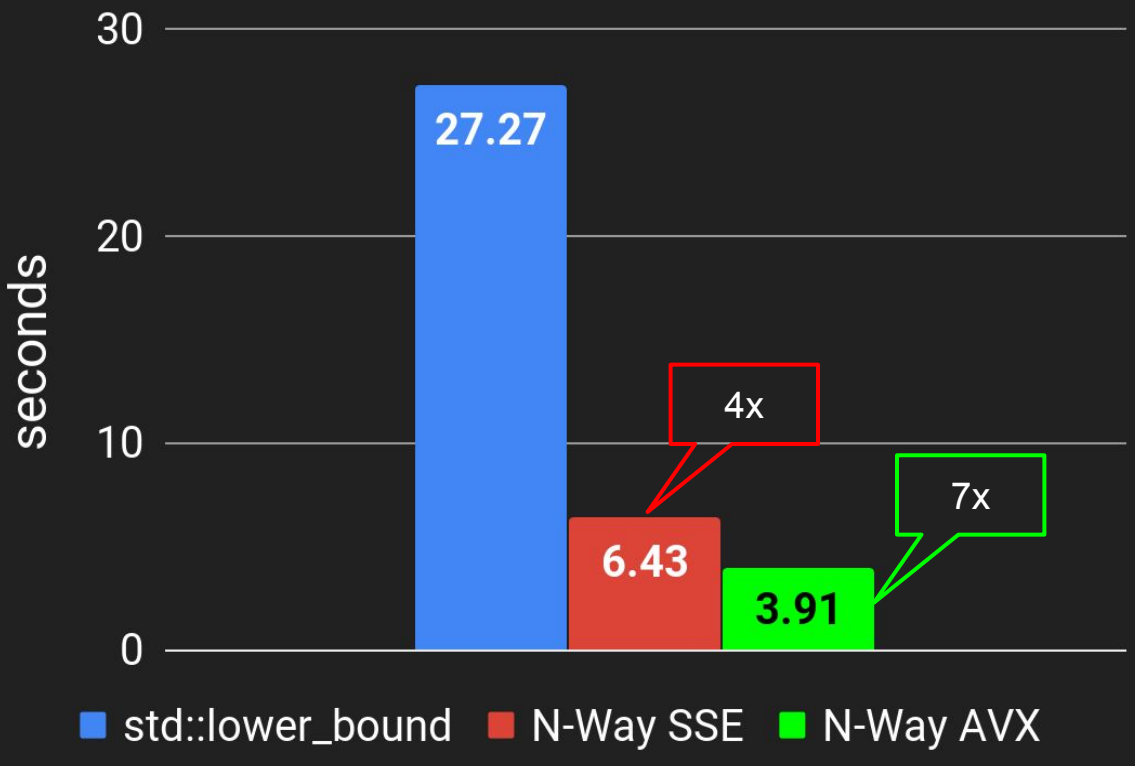
Árvore n-Way

```
const_iterator find( int32_t key ) {
  size_t idx = 0;
  for( std::vector< int32_t >& level : tree_ ) {
    uint32_t li = greater_than_index( key,
                  *(__m128i*) &level[ idx * 4 ] );
    idx = idx * 4 + li; // < 4 = SIMD size ^
  }
```

Árvore n-Way (cont.)

```
  __m128i cmp =
    *(__m128i*) &vec_[ idx * 4 ]; // 4 = SIMD size
  uint32_t off = equal_index( key, cmp );
  if( off == 0 )    // 0 => key not found in cmp
    return vec_.end();
  auto it = vec_.begin();
  std::advance( it, idx * 4 + off - 1 );
  return it;                // ^ 4 = SIMD size
}
```

Resultados árvore n-Way



Core i7-4870HQ
MacOSX El Captain 10.11.6
Clang 6.0.1 (homebrew)
-std=c++14 -mavx2 -O3
-fno-strict-aliasing -Wall

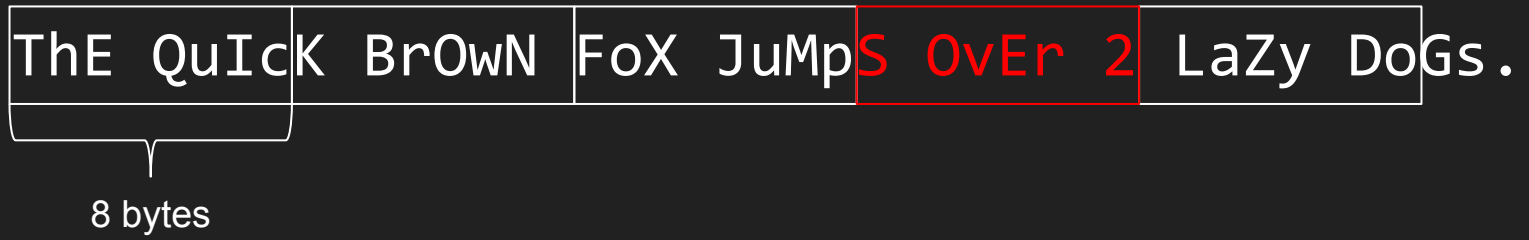
std::vector< int32_t >
Size = 0x00400000
Busca todos os dados em ordem randômica por 10 vezes
Tempo médio após 100 execuções

to_lower(std::string& str)

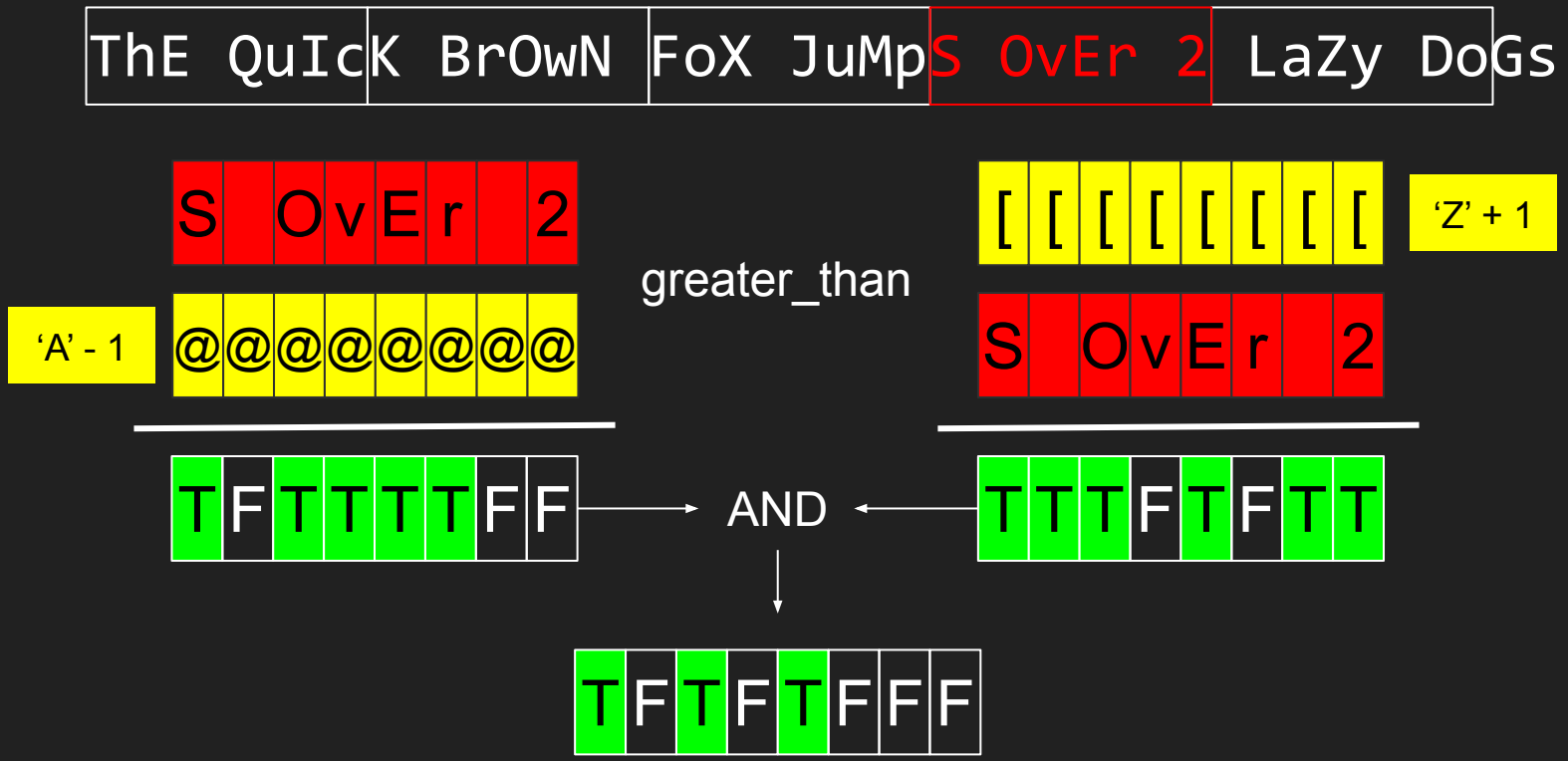
“ThE QuIcK BrOwN FoX JuMpS OvEr 2 LaZy DoGs.”

“the quick brown fox jumps over 2 lazy dogs.”

to_lower(std::string& str)

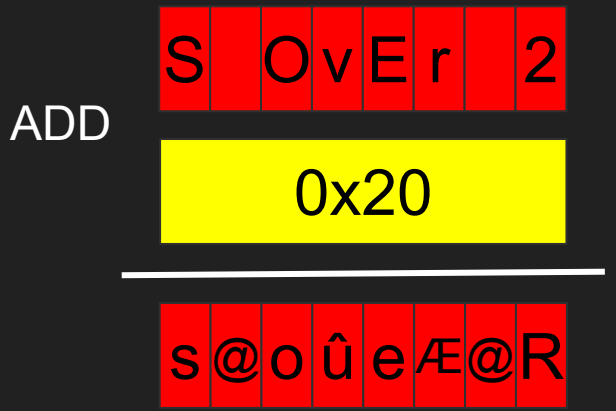


to_lower(std::string& str)



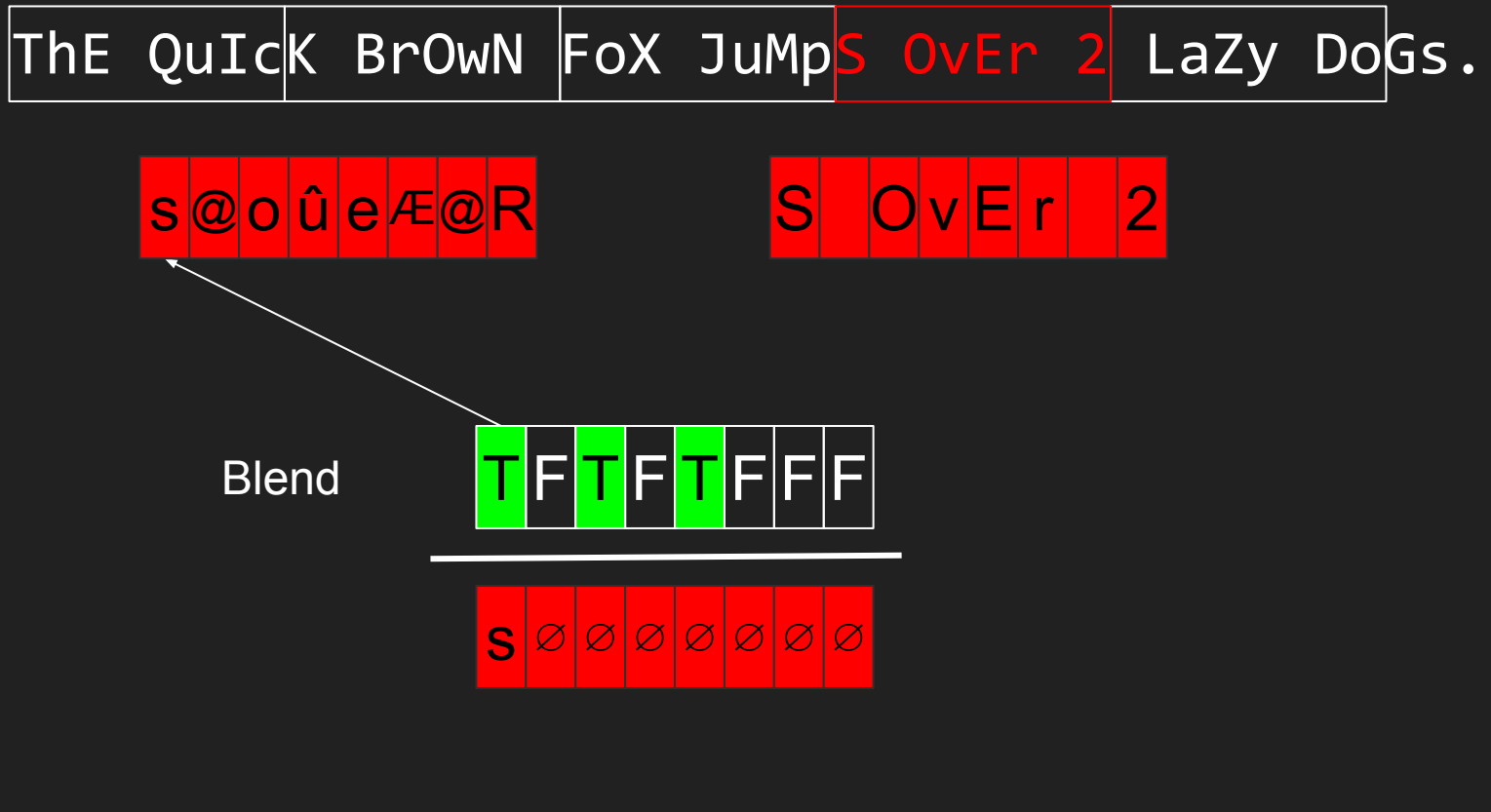
to_lower(std::string& str)

ThE QuIcK BrOwN	FoX JuMp	S OvEr 2	LaZy DoGs.

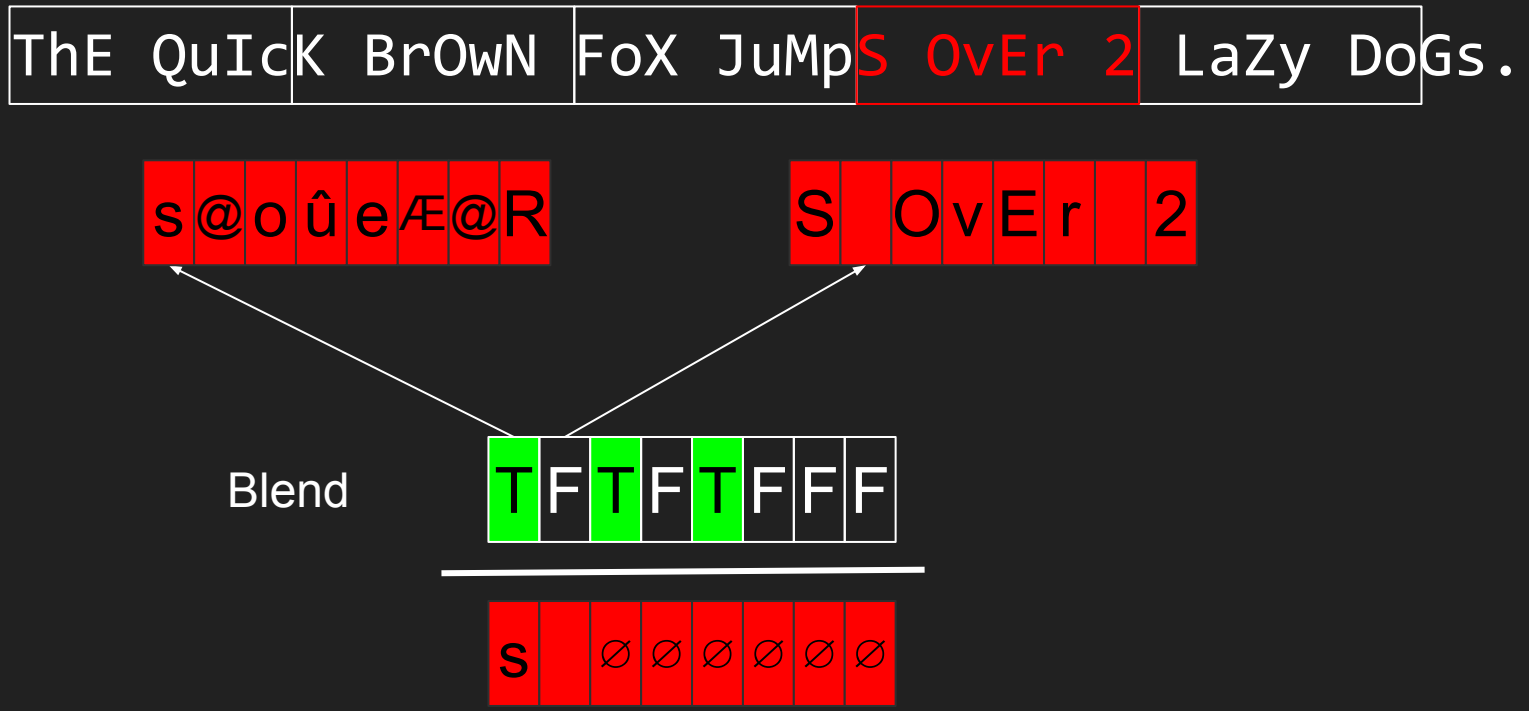


to_lower(std::string& str)

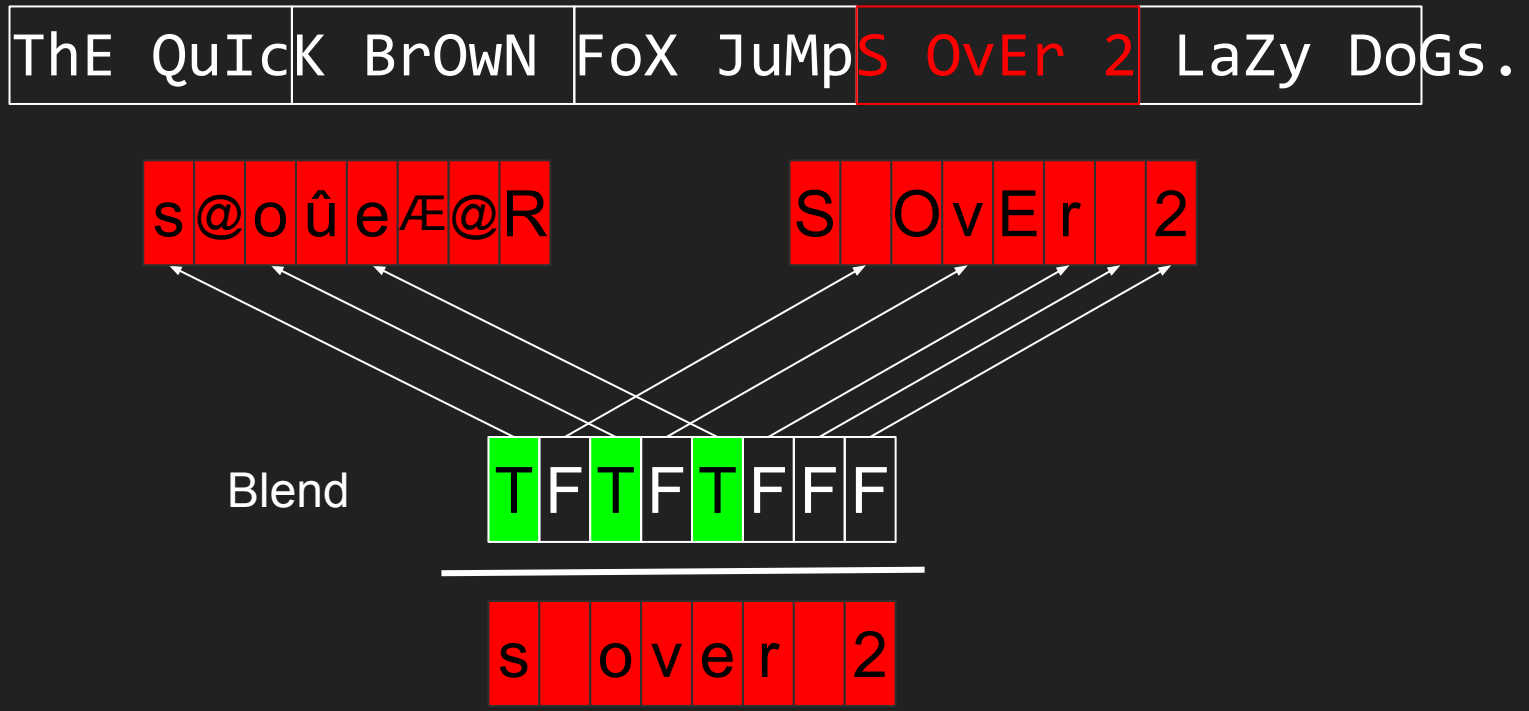
to_lower(std::string& str)



to_lower(std::string& str)



to_lower(std::string& str)



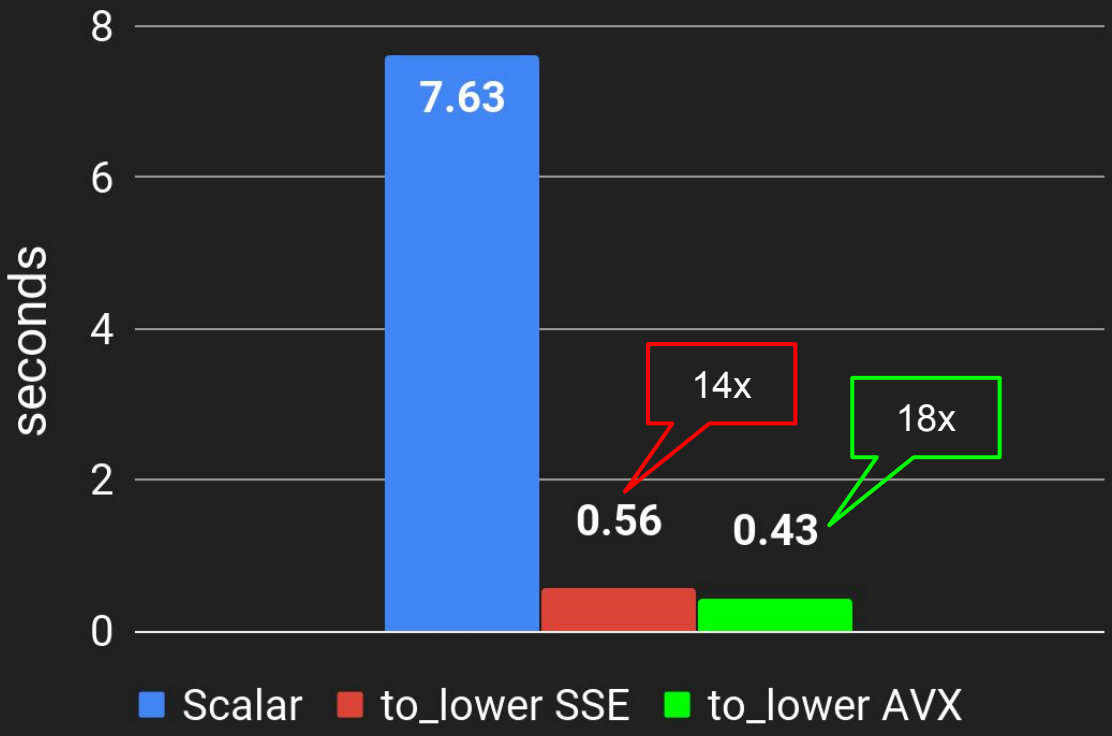
to_lower(std::string& str)

```
__m128i* data = (__m128i*) str.data();
size_t sz = str.size() & ~(16-1); // 16=SIMD size
for( size_t i = 0; i < sz; i += 16 ) {
  __m128i low = _mm_cmpgt_epi8( *data,
                          _mm_set1_epi8( 'A'-1 ) );
  __m128i high = _mm_cmpgt_epi8(
                   _mm_set1_epi8( 'Z'+1 ), *data );
  __m128i mask = _mm_and_si128( low, high );
```

to_lower(std::string& str) // cont...

```
  __m128i lower = _mm_add_epi8( *data,
                                _mm_set1_epi8( 0x20 ) );
  *data = _mm_blendv_epi8( *data, lower, mask );
  ++data;
}
for( size_t i = sz; i < str.size(); ++i ) {
  if( 'A' <= str[i] && str[i] <= 'Z' )
    str[i] += 0x20;
}
```

Resultados do to_lower SIMD



Core i7-4870HQ
MacOSX El Captain 10.11.6
Clang 6.0.1 (homebrew)
-std=c++14 -mavx2 -O3
-fno-strict-aliasing -Wall

std::string
Size = 0x00100001
Chamando to_lower 10000 vezes em cada execução
Tempo médio após 100 execuções

Baixe o código!

- Código e apresentação estão no github
  - Também com bubble sort

https://github.com/andrelrt/simd_algorithms

- Qualquer dúvida
  - andrelrt@gmail.com

Instruções SIMD para comparação

André Tupinambá