

Adventures in SIMD Thinking (Part 1 of 2)

Bob Steagall
CppCon 2020



- Learn a little about Intel's SIMD facilities (disclaimer: I don't work for Intel)
- Create some useful functions in terms of AVX-512 intrinsics
- Try some SIMD-style thinking to tackle some interesting problems
 - Intra-register sorting
 - Fast linear median-of-seven filter
 - Fast small-kernel convolution
 - Faster (?) UTF-8 to UTF-32 conversion (with AVX2)
- No heavy code, but lots of pictures
 - Thinking "vertically"

Getting Started

```
#include <cstdio>
#include <stdint>
#include <type_traits>

#ifdef __OPTIMIZE__
    #include <immintrin.h>
    #define KEWB_FORCE_INLINE    inline __attribute__((__always_inline__))
#else
    #define __OPTIMIZE__
    #include <immintrin.h>
    #undef __OPTIMIZE__
    #define KEWB_FORCE_INLINE    inline
#endif

namespace simd {

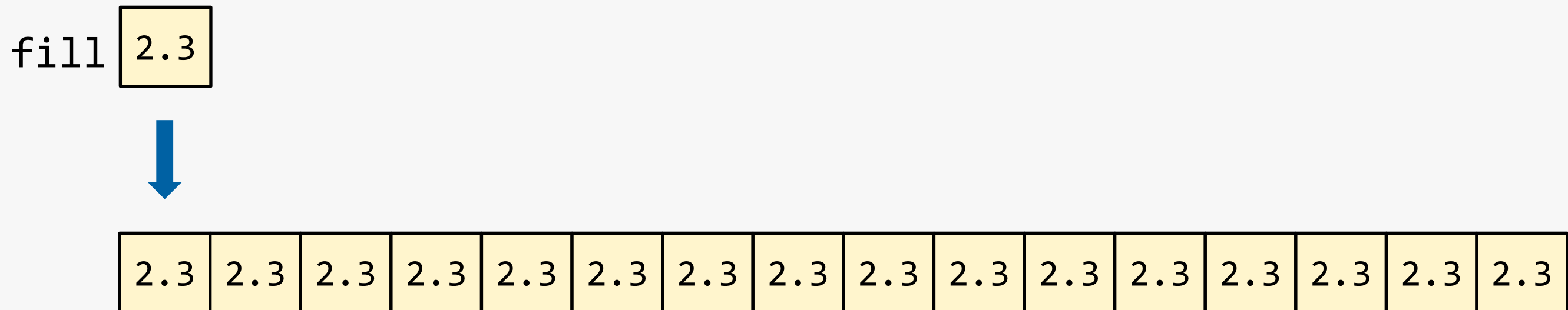
using rf_512    = __m512;
using ri_512    = __m512i;
using msk_512   = uint32_t;

...

}
```

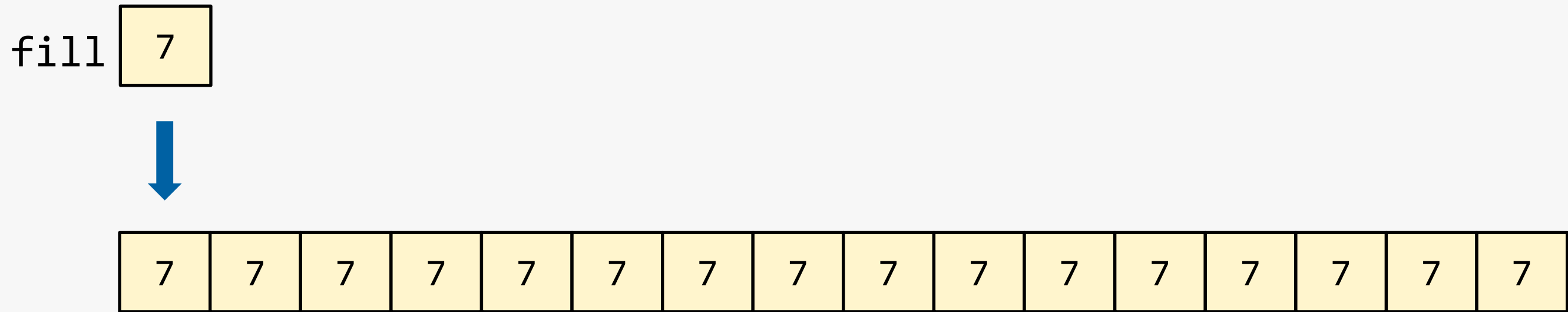
Function load_value()

```
KEWB_FORCE_INLINE rf_512  
load_value(float fill)  
{  
    return _mm512_set1_ps(v);  
}
```



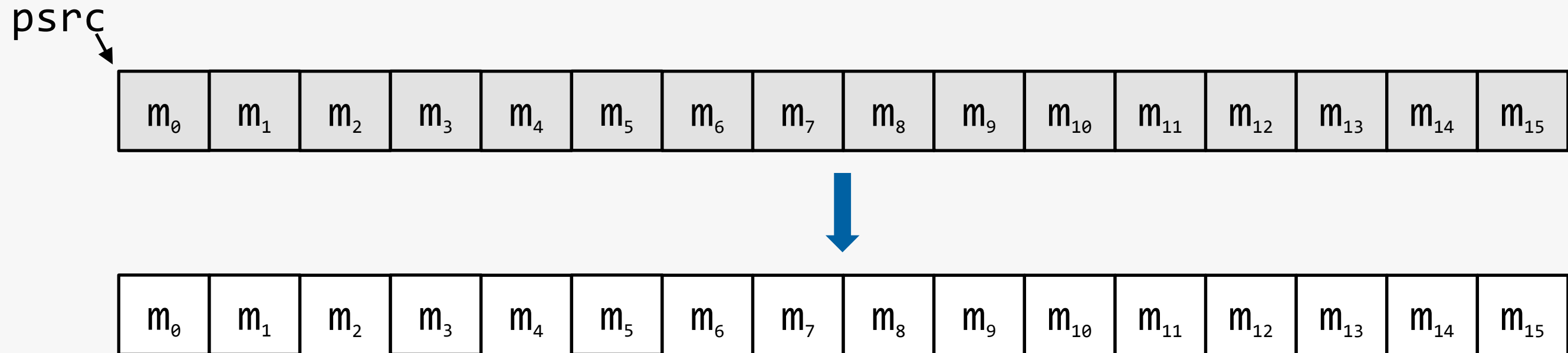
Function load_value()

```
KEWB_FORCE_INLINE ri_512  
load_value(int32_t fill)  
{  
    return _mm512_set1_epi32(i);  
}
```



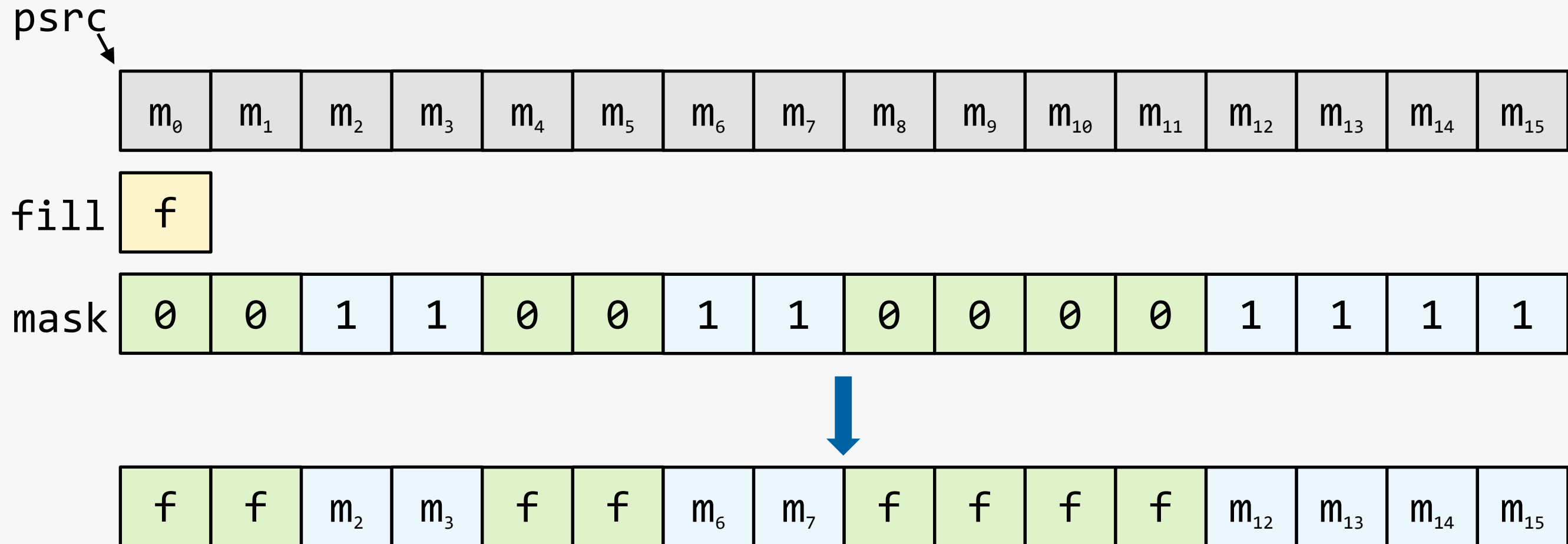
Function load_from()

```
KEWB_FORCE_INLINE rf_512  
load_from(float const* psrc)  
{  
    return _mm512_loadu_ps(psrc);  
}
```



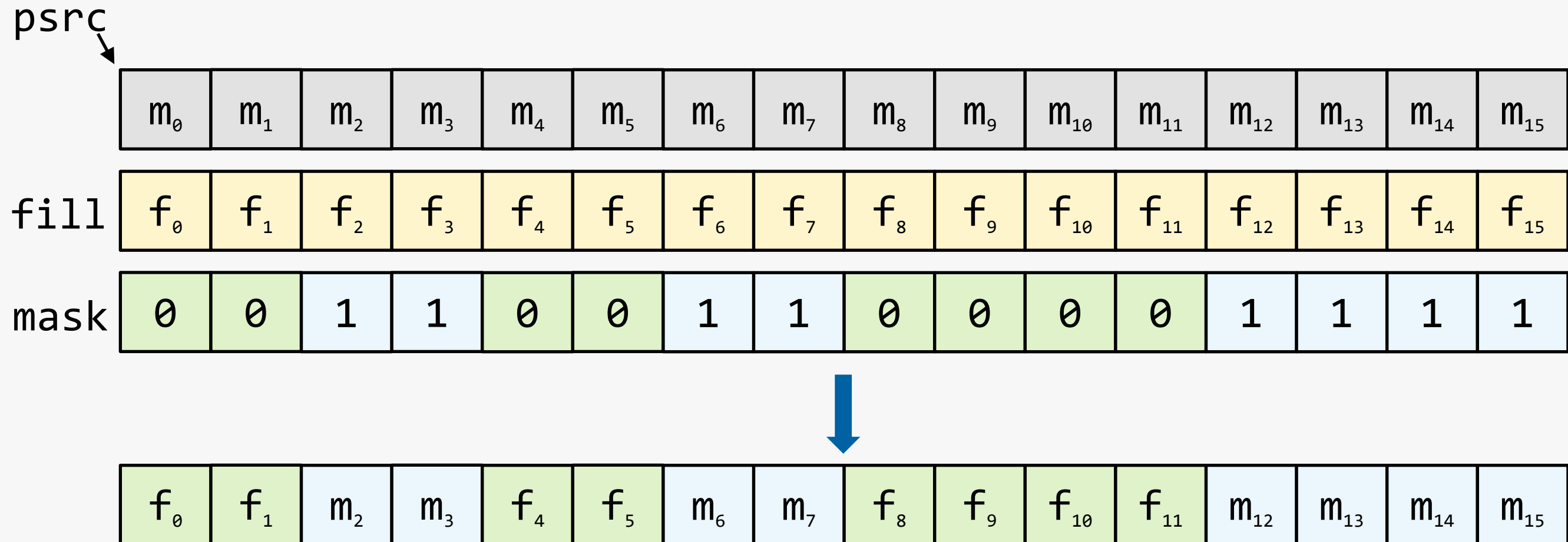
Function masked_load_from()

```
KEWB_FORCE_INLINE rf_512
masked_load_from(float const* psrc, float fill, msk_512 mask)
{
    return _mm512_mask_loadu_ps(_mm512_set1_ps(fill), (__mmask16) mask, psrc);
}
```



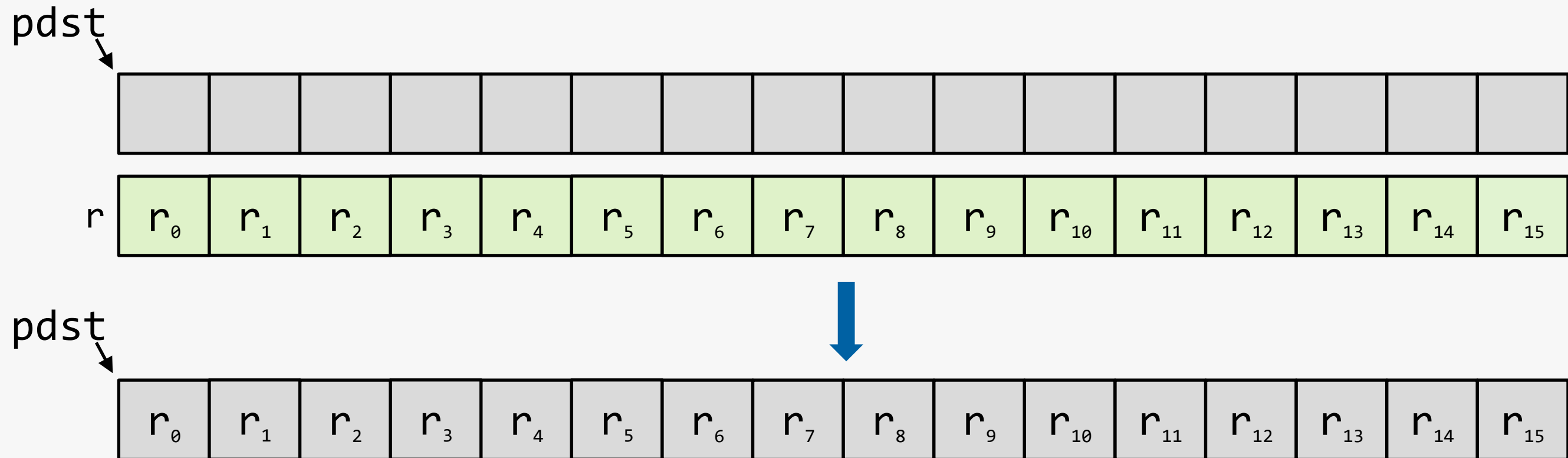
Function masked_load_from()

```
KEWB_FORCE_INLINE rf_512
masked_load_from(float const* psrc, rf_512 fill, msk_512 mask)
{
    return _mm512_mask_loadu_ps(fill, (__mmask16) mask, psrc);
}
```



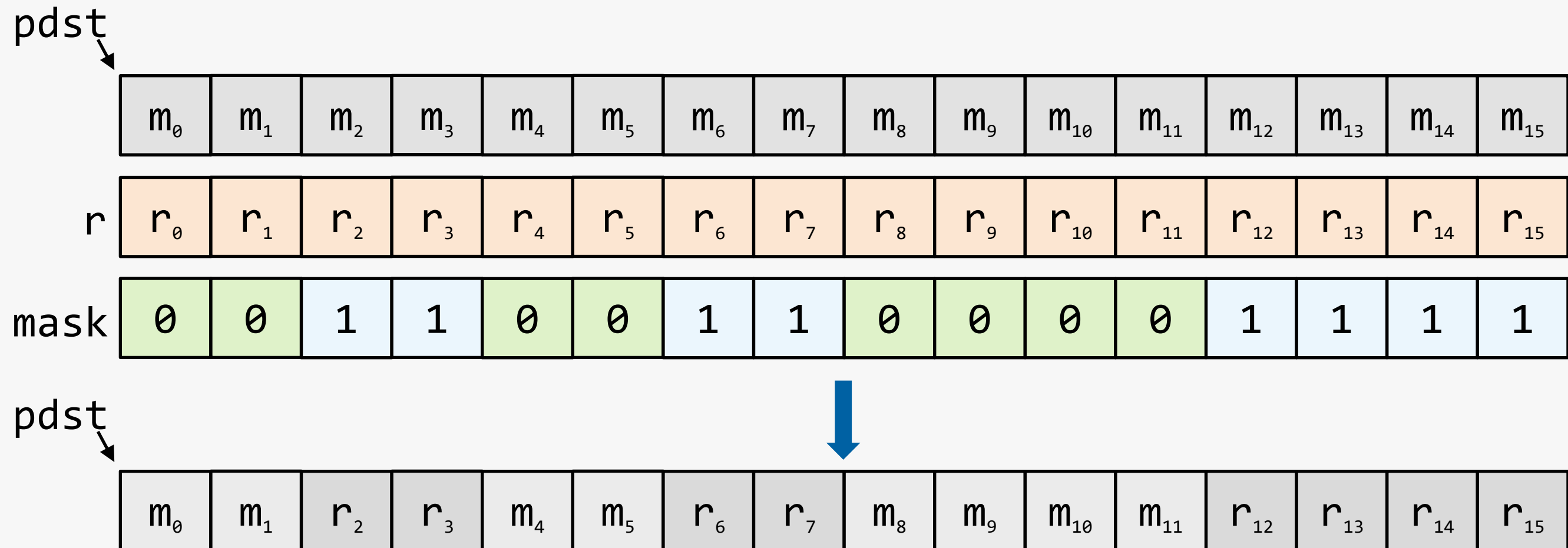
Function store_to()

```
KEWB_FORCE_INLINE void  
store_to(float* pdst, rf_512 r)  
{  
    _mm512_mask_storeu_ps(pdst, (__mmask16) 0xFFFFu, r);  
}
```



Function masked_store_to()

```
KEWB_FORCE_INLINE void  
masked_store_to(float* pdst, rf_512 r, msk_512 mask)  
{  
    _mm512_mask_storeu_ps(pdst, (__mmask16) mask, r);  
}
```



Function make_bit_mask()

```
template<unsigned A=0, unsigned B=0, unsigned C=0, unsigned D=0,
        unsigned E=0, unsigned F=0, unsigned G=0, unsigned H=0,
        unsigned I=0, unsigned J=0, unsigned K=0, unsigned L=0,
        unsigned M=0, unsigned N=0, unsigned O=0, unsigned P=0>
KEWB_FORCE_INLINE constexpr uint32_t
make_bit_mask()
{
    static_assert((A < 2) && (B < 2) && (C < 2) && (D < 2) &&
                  (E < 2) && (F < 2) && (G < 2) && (H < 2) &&
                  (I < 2) && (J < 2) && (K < 2) && (L < 2) &&
                  (M < 2) && (N < 2) && (O < 2) && (P < 2));

    return ((A << 0) | (B << 1) | (C << 2) | (D << 3) |
            (E << 4) | (F << 5) | (G << 6) | (H << 7) |
            (I << 8) | (J << 9) | (K << 10) | (L << 11) |
            (M << 12) | (N << 13) | (O << 14) | (P << 15));
}
```

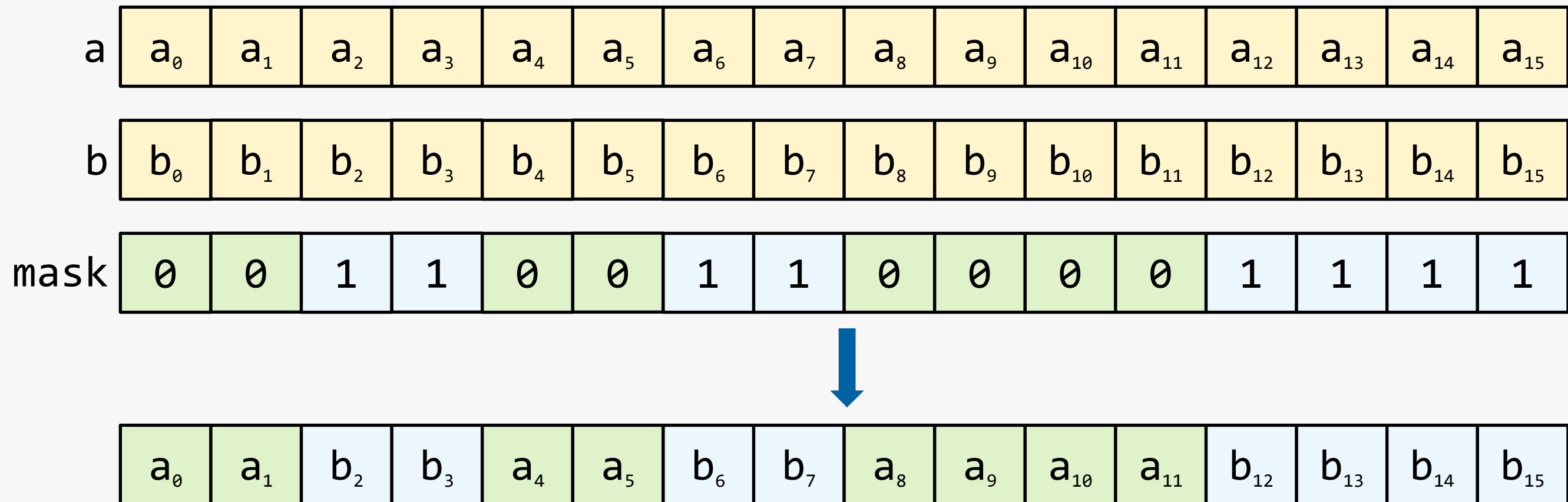


0bPONM' LKJI' HGFE' DCBA

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

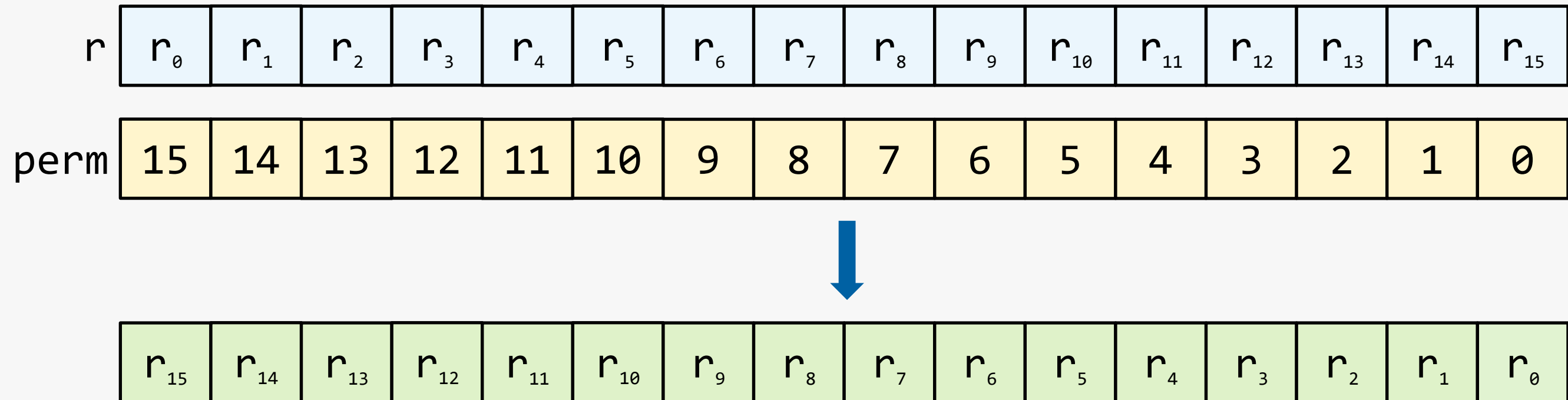
Function blend()

```
KEWB_FORCE_INLINE rf_512  
blend(rf_512 a, rf_512 b, msk_512 mask)  
{  
    return _mm512_mask_blend_ps((__mmask16) mask, a, b);  
}
```



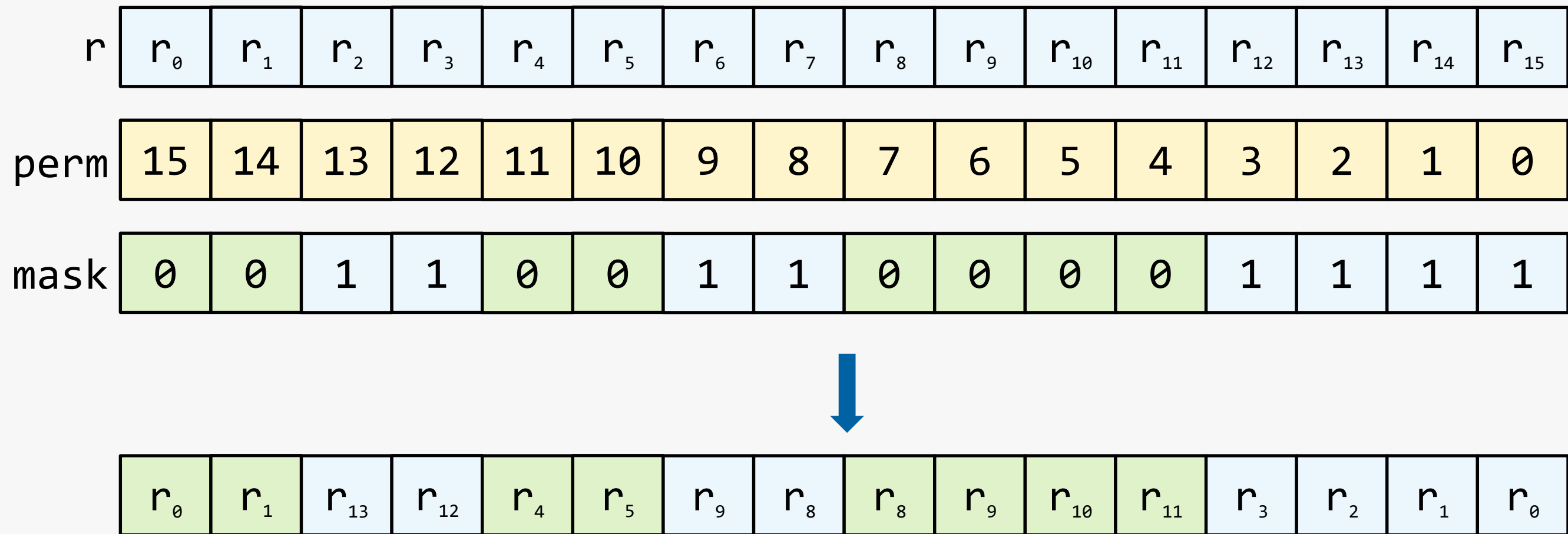
Function permute()

```
KEWB_FORCE_INLINE rf_512  
permute(rf_512 r, ri_512 perm)  
{  
    return _mm512_permutexvar_ps(perm, r);  
}
```



Function masked_permute()

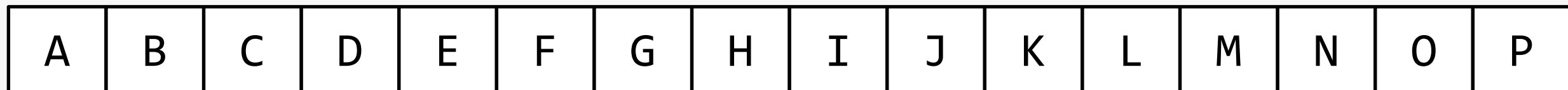
```
KEWB_FORCE_INLINE rf_512
masked_permute(rf_512 a, rf_512 b, ri_512 perm, msk_512 mask)
{
    return _mm512_mask_permutexvar_ps(a, (__mmask16) mask, perm, b);
}
```



Function make_perm_map()

```
template<unsigned A, unsigned B, unsigned C, unsigned D,
        unsigned E, unsigned F, unsigned G, unsigned H,
        unsigned I, unsigned J, unsigned K, unsigned L,
        unsigned M, unsigned N, unsigned O, unsigned P>
KEWB_FORCE_INLINE ri_512
make_perm_map()
{
    static_assert((A < 16) && (B < 16) && (C < 16) && (D < 16) &&
                  (E < 16) && (F < 16) && (G < 16) && (H < 16) &&
                  (I < 16) && (J < 16) && (K < 16) && (L < 16) &&
                  (M < 16) && (N < 16) && (O < 16) && (P < 16));

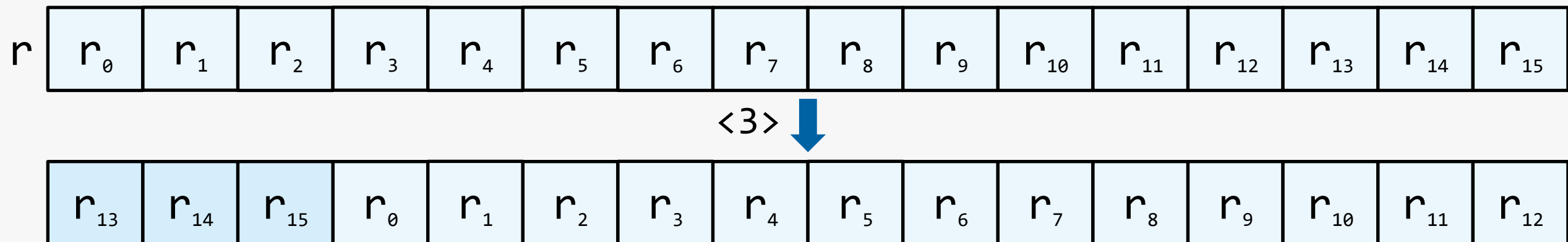
    return _mm512_setr_epi32(A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P);
}
```



Function rotate()

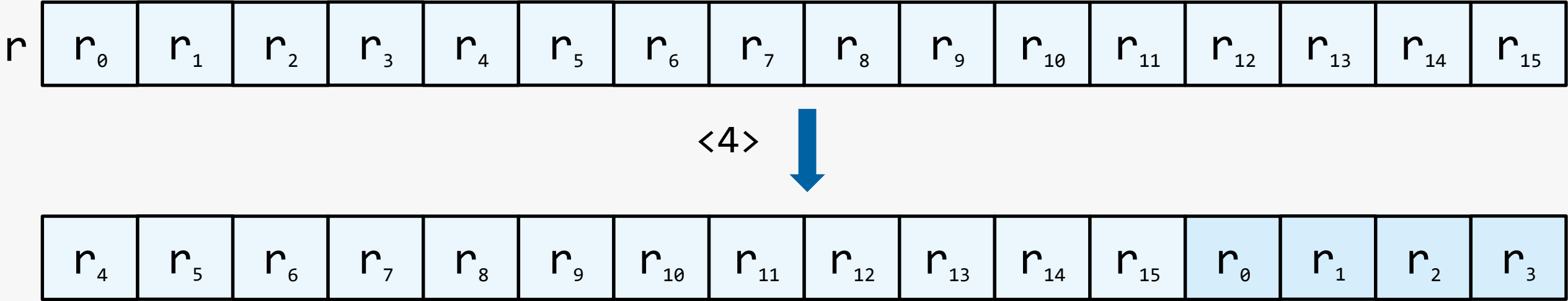
```
template<int R> KEWB_FORCE_INLINE rf_512
rotate(rf_512 r)
{
    if constexpr ((R % 16) == 0)
    {
        return r;
    }
    else
    {
        constexpr int    S = (R > 0) ? (16 - (R % 16)) : -R;
        constexpr int    A = (S + 0) % 16;
        constexpr int    B = (S + 1) % 16;
        ...
        constexpr int    O = (S + 14) % 16;
        constexpr int    P = (S + 15) % 16;

        return _mm512_permutexvar_ps(_mm512_setr_epi32(A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P), r);
    }
}
```



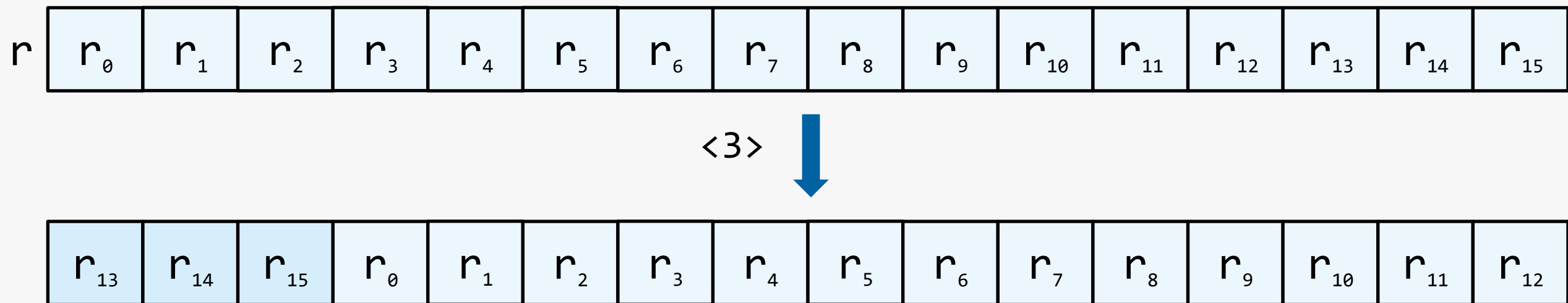
Function rotate_down()

```
template<int R> KEWB_FORCE_INLINE rf_512
rotate_down(rf_512 r)
{
    static_assert(R >= 0);
    return rotate<-R>(r);
}
```



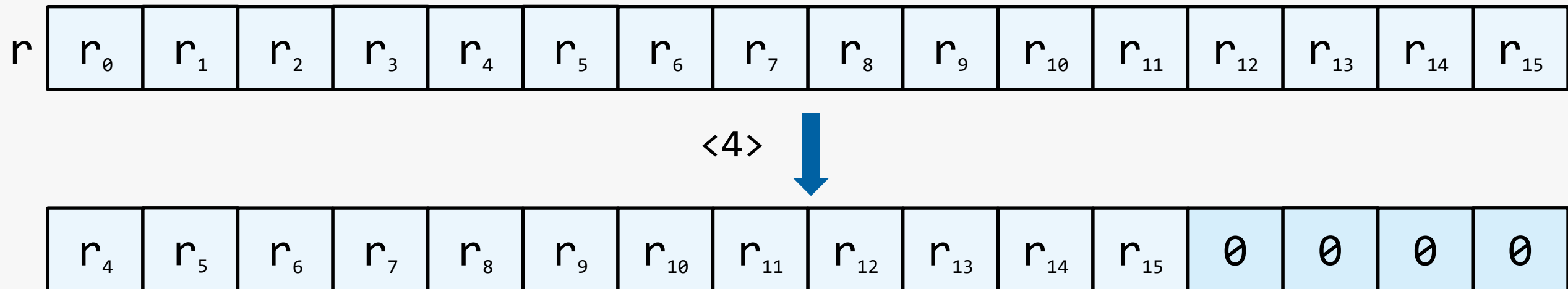
Function rotate_up()

```
template<int R> KEWB_FORCE_INLINE rf_512  
rotate_up(rf_512 r)  
{  
    static_assert(R >= 0);  
    return rotate<R>(r);  
}
```



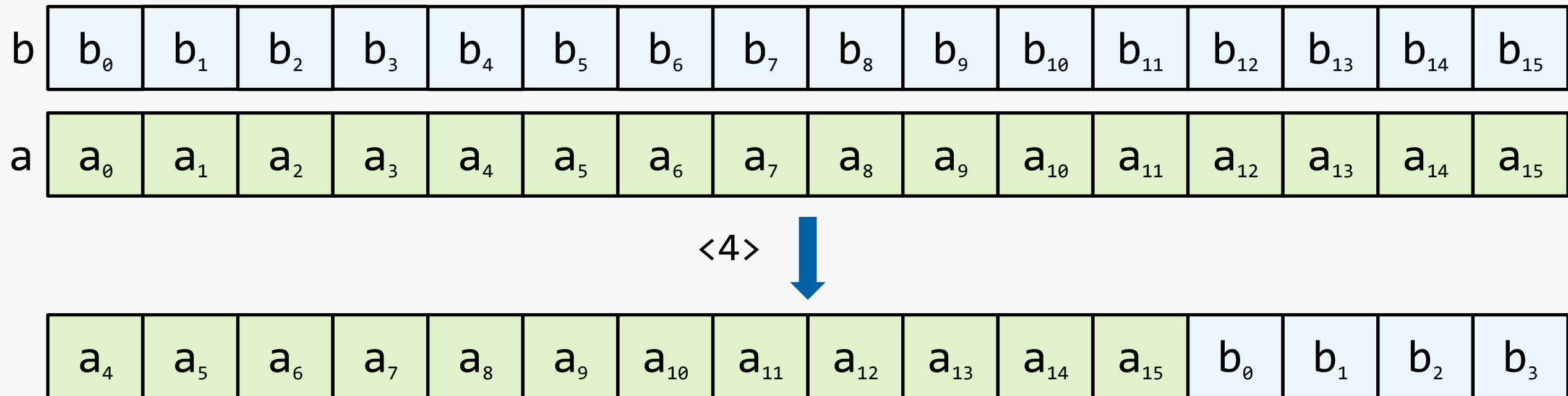
Function shift_down()

```
template<int S> KEWB_FORCE_INLINE rf_512
shift_down(rf_512 r)
{
    static_assert(S >= 0 && S <= 16);
    return blend(rotate_down<S>(r), load_value(0.0f), shift_down_blend_mask<S>());
}
```



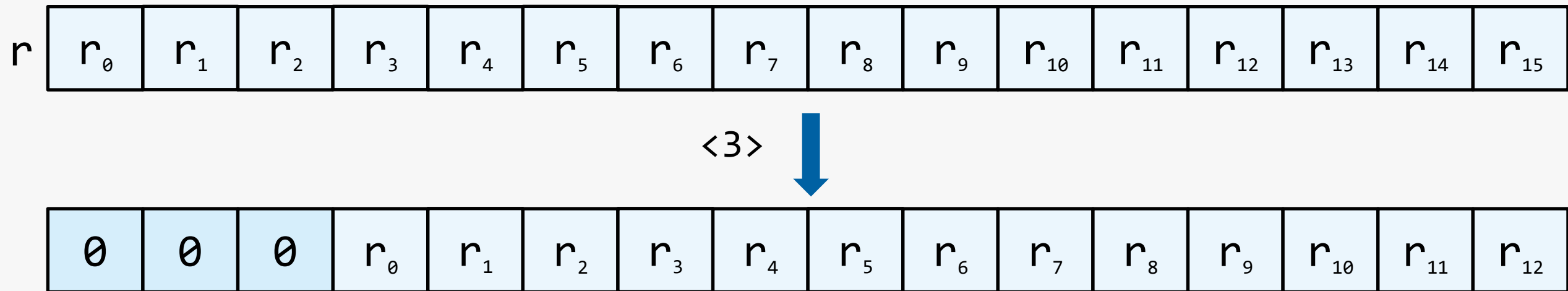
Function shift_down_with_carry()

```
template<int S> KEWB_FORCE_INLINE rf_512
shift_down_with_carry(rf_512 a, rf_512 b)
{
    static_assert(S >= 0 && S <= 16);
    return blend(rotate_down<S>(a), rotate_down<S>(b), shift_down_blend_mask<S>());
}
```



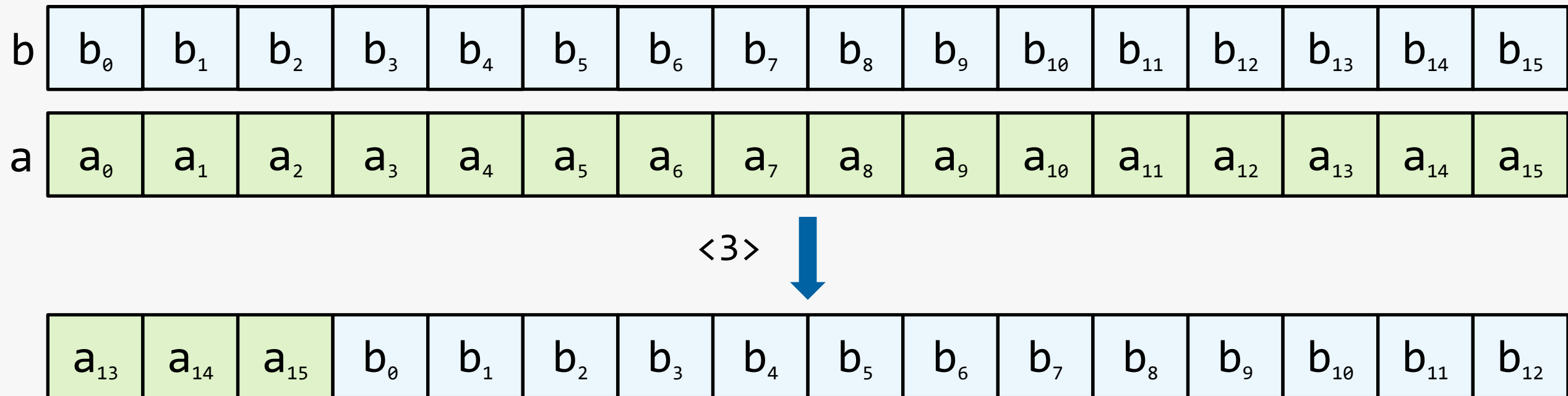
Function shift_up()

```
template<int S> KEWB_FORCE_INLINE rf_512
shift_up(rf_512 r0)
{
    static_assert(S >= 0 && S <= 16);
    return blend(rotate_up<S>(r0), load_value(0.0f), shift_up_blend_mask<S>());
}
```



Function shift_up_with_carry()

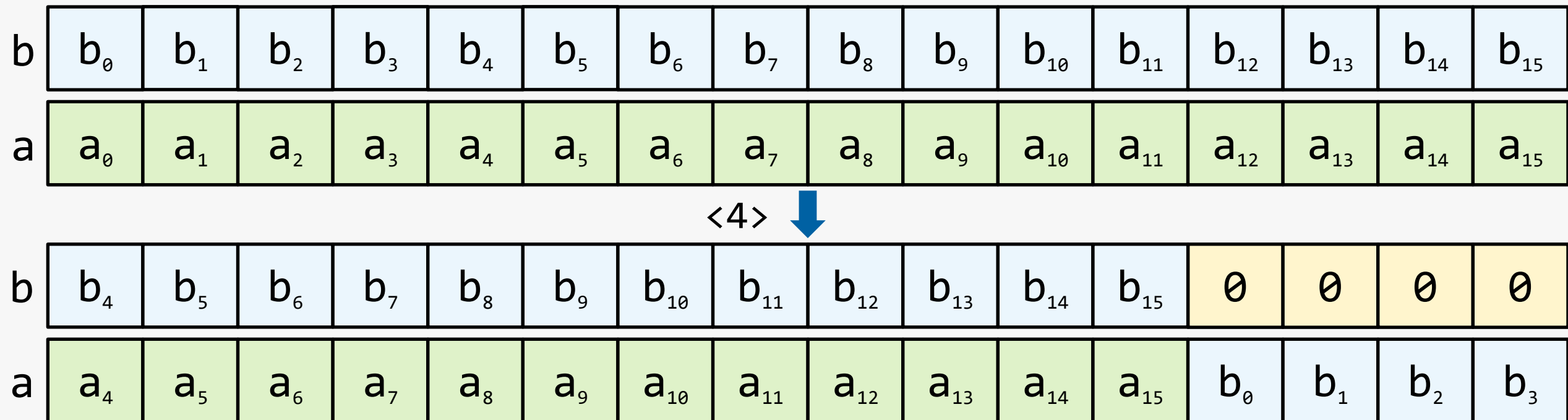
```
template<int S> KEWB_FORCE_INLINE rf_512
shift_up_with_carry(rf_512 lo, rf_512 hi)
{
    static_assert(S >= 0 && S <= 16);
    return blend(rotate_up<S>(lo), rotate_up<S>(hi), shift_up_blend_mask<S>());
}
```



Function in_place_shift_down_with_carry()

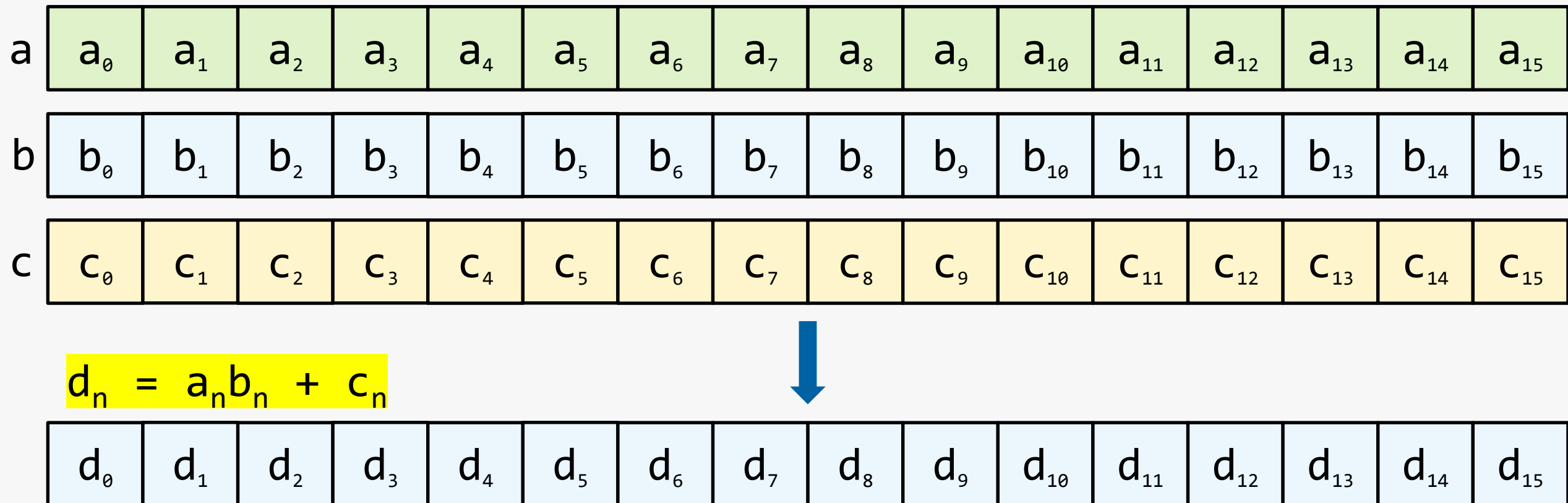
```
template<int S> KEWB_FORCE_INLINE void
in_place_shift_down_with_carry(rf_512& a, rf_512& b)
{
    static_assert(S >= 0 && S <= 16);
    constexpr msk_512 zmask = (0xFFFFu >> (unsigned) S);
    constexpr msk_512 bmask = ~zmask & 0xFFFFu;
    ri_512 perm = make_shift_permutation<S, bmask>();

    a = _mm512_permutex2var_ps(a, perm, hi);
    b = _mm512_maskz_permutex2var_ps((__mmask16) zmask, b, perm, b);
}
```



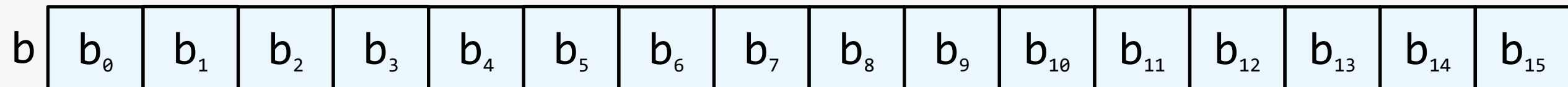
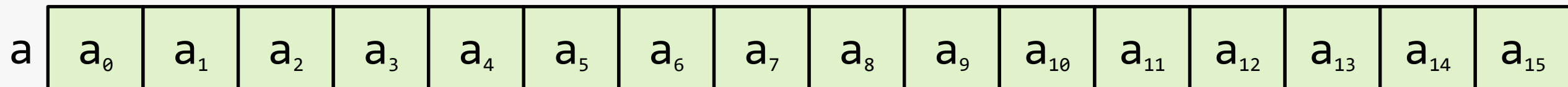
Function fused_multiply_add()

```
KEWB_FORCE_INLINE rf_512
fused_multiply_add(rf_512 a, rf_512 b, rf_512 c)
{
    return _mm512_fmadd_ps(a, b, c);
}
```

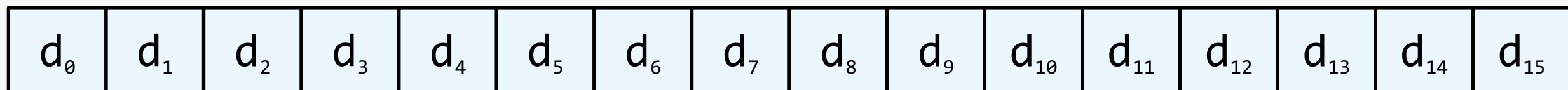


Function minimum()

```
KEWB_FORCE_INLINE rf_512  
minimum(rf_512 a, rf_512 b)  
{  
    return _mm512_min_ps(a, b);  
}
```

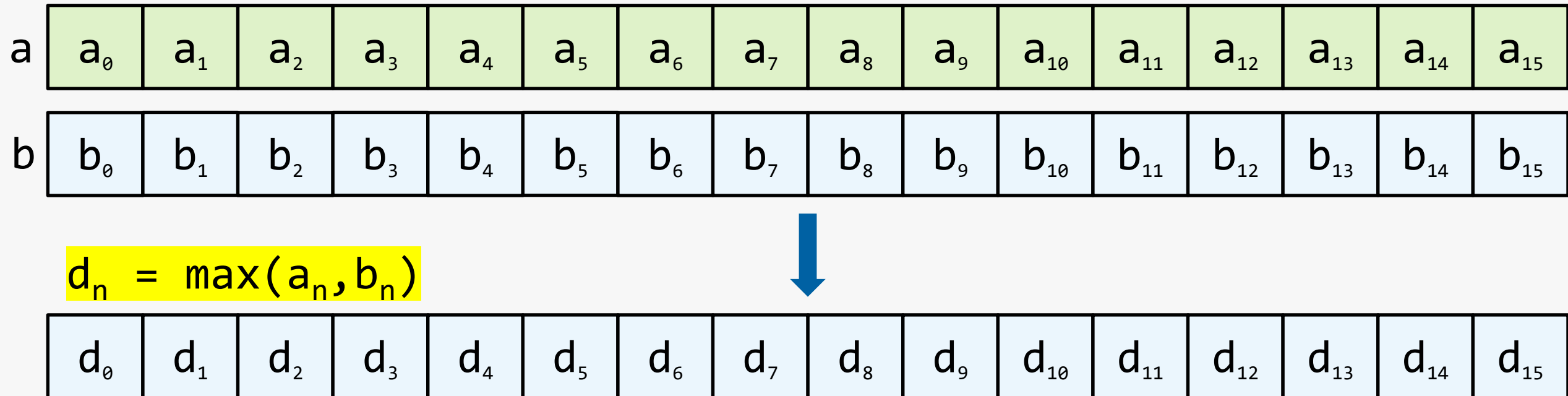


$$d_n = \min(a_n, b_n)$$



Function maximum()

```
KEWB_FORCE_INLINE rf_512  
maximum(rf_512 a, rf_512 a)  
{  
    return _mm512_max_ps(a, b);  
}
```



Function `compare_with_exchange()`

```
KEWB_FORCE_INLINE rf_512
compare_with_exchange(rf_512 vals, ri_512 perm, msk_512 mask)
{
    rf_512  exch = permute(vals, perm);
    rf_512  vmin = minimum(vals, exch);
    rf_512  vmax = maximum(vals, exch);

    return blend(vmin, vmax, mask);
}
```

Function compare_with_exchange()

vals	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
perm	1	0	2	3	4	5	6	7	8	9	10	11	12	13	14	15
mask	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Function compare_with_exchange()

```
KEWB_FORCE_INLINE rf_512
compare_with_exchange(rf_512 vals, ri_512 perm, msk_512 mask)
{
    rf_512  exch = permute(vals, perm);
    rf_512  vmin = minimum(vals, exch);
    rf_512  vmax = maximum(vals, exch);

    return blend(vmin, vmax, mask);
}
```

Function compare_with_exchange()

vals	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
perm	1	0	2	3	4	5	6	7	8	9	10	11	12	13	14	15
mask	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
exch	14	15	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Function compare_with_exchange()

```
KEWB_FORCE_INLINE rf_512
compare_with_exchange(rf_512 vals, ri_512 perm, msk_512 mask)
{
    rf_512  exch = permute(vals, perm);
    rf_512  vmin = minimum(vals, exch);
    rf_512  vmax = maximum(vals, exch);

    return blend(vmin, vmax, mask);
}
```

Function compare_with_exchange()

→	vals	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	perm	1	0	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	mask	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	exch	14	15	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	vmin	14	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Function compare_with_exchange()

```
KEWB_FORCE_INLINE rf_512
compare_with_exchange(rf_512 vals, ri_512 perm, msk_512 mask)
{
    rf_512  exch = permute(vals, perm);
    rf_512  vmin = minimum(vals, exch);
    rf_512  vmax = maximum(vals, exch);

    return blend(vmin, vmax, mask);
}
```

Function compare_with_exchange()

vals	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
perm	1	0	2	3	4	5	6	7	8	9	10	11	12	13	14	15
mask	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
exch	14	15	13	12	11	10	9	8	7	6	5	4	3	2	1	0
vmin	14	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
vmax	15	15	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Function compare_with_exchange()

```
KEWB_FORCE_INLINE rf_512
compare_with_exchange(rf_512 vals, ri_512 perm, msk_512 mask)
{
    rf_512  exch = permute(vals, perm);
    rf_512  vmin = minimum(vals, exch);
    rf_512  vmax = maximum(vals, exch);

    return blend(vmin, vmax, mask);
}
```

Function compare_with_exchange()

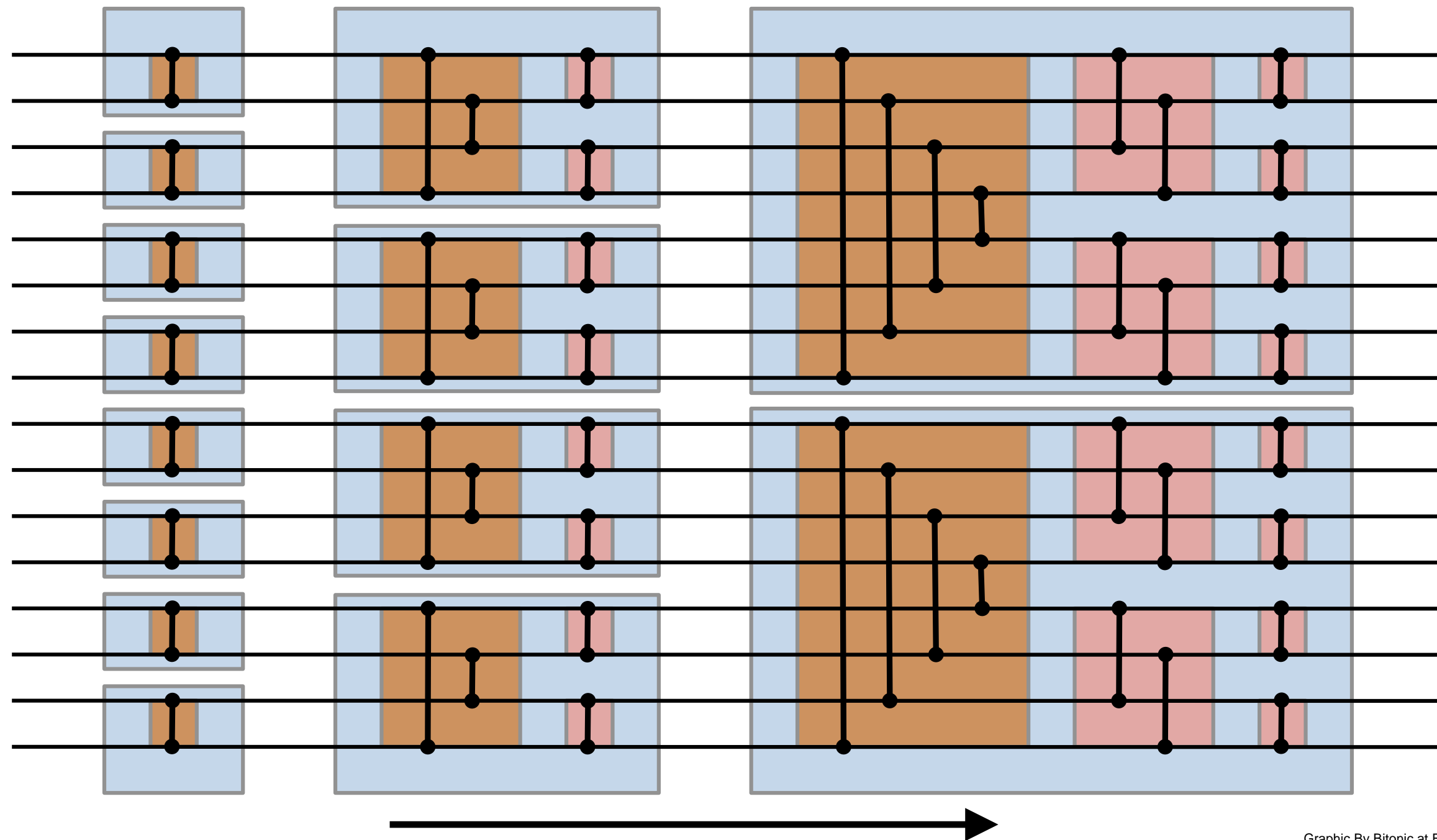
vals	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
perm	1	0	2	3	4	5	6	7	8	9	10	11	12	13	14	15
mask	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
exch	14	15	13	12	11	10	9	8	7	6	5	4	3	2	1	0
vmin	14	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
vmax	15	15	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	14	15	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Function compare_with_exchange()

vals	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
perm	1	0	2	3	4	5	6	7	8	9	10	11	12	13	14	15
mask	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
exch	14	15	13	12	11	10	9	8	7	6	5	4	3	2	1	0
vmin	14	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
vmax	15	15	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	14	15	13	12	11	10	9	8	7	6	5	4	3	2	1	0

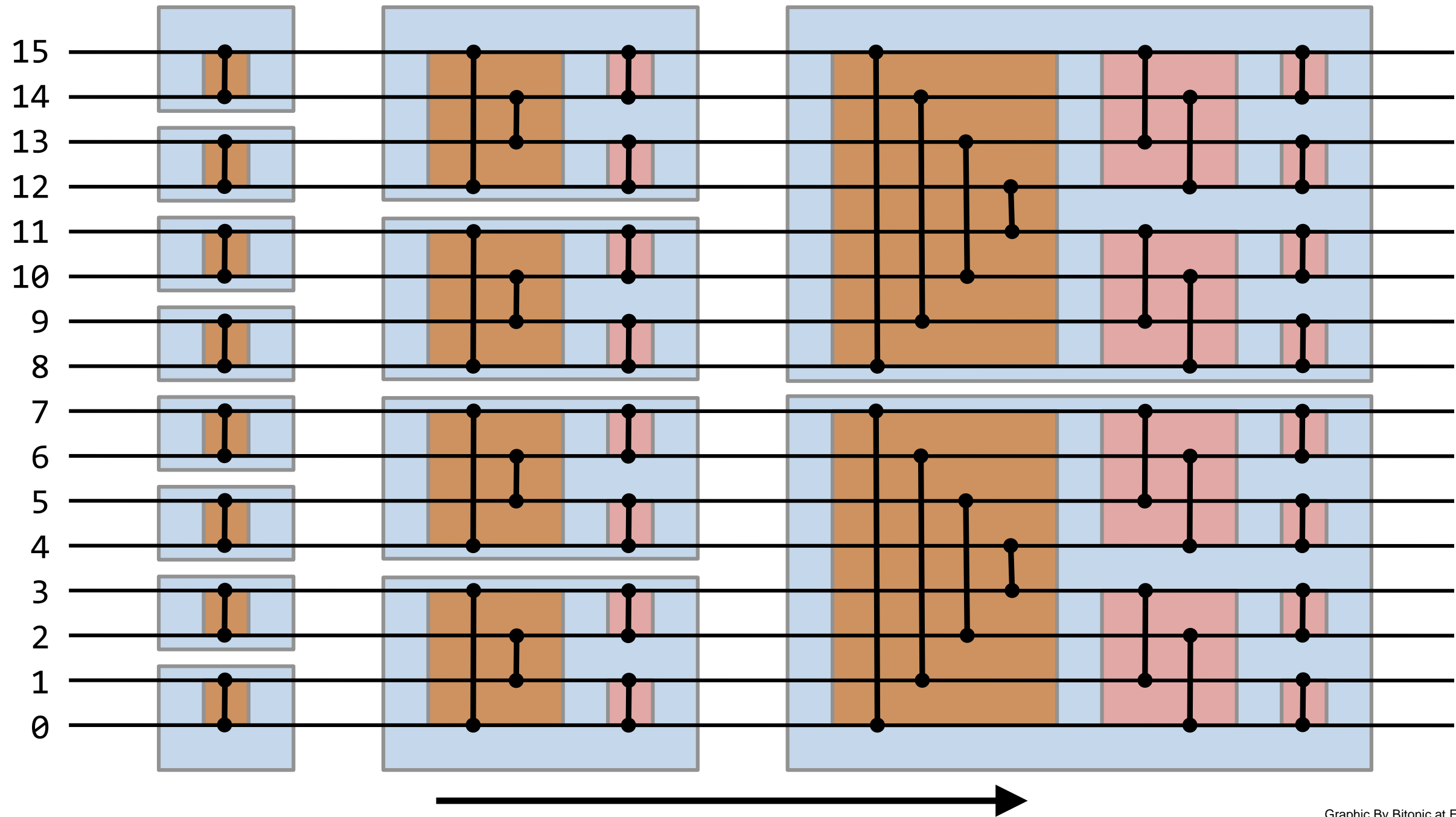
Intra-Register Sorting

Sorting Network for Two Lanes of Eight



Graphic By Bitonic at English Wikipedia - Own work, CC0,
<https://commons.wikimedia.org/w/index.php?curid=21961917>

Sorting Network for Two Lanes of Eight



Graphic By Bitonic at English Wikipedia - Own work, CC0,
<https://commons.wikimedia.org/w/index.php?curid=21961917>

Function sort_two_lanes_of_8()

```
KEWB_FORCE_INLINE rf_512
sort_two_lanes_of_8(rf_512 vals)
{
    //- Precompute the permutations and bitmasks for the 6 stages of this bitonic sorting sequence.
    //
    //
    //
    ri_512 const      perm0 = make_perm_map<1,  0,  3,  2,  5,  4,  7,  6,  9,  8, 11, 10, 13, 12, 15, 14>();
    constexpr msk_512 mask0 = make_bit_mask<0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1>();

    ri_512 const      perm1 = make_perm_map<3,  2,  1,  0,  7,  6,  5,  4, 11, 10,  9,  8, 15, 14, 13, 12>();
    constexpr msk_512 mask1 = make_bit_mask<0,  0,  1,  1,  0,  0,  1,  1,  0,  0,  1,  1,  0,  0,  1,  1>();

    ri_512 const      perm2 = make_perm_map<1,  0,  3,  2,  5,  4,  7,  6,  9,  8, 11, 10, 13, 12, 15, 14>();
    constexpr msk_512 mask2 = make_bit_mask<0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1>();

    ri_512 const      perm3 = make_perm_map<7,  6,  5,  4,  3,  2,  1,  0, 15, 14, 13, 12, 11, 10,  9,  8>();
    constexpr msk_512 mask3 = make_bit_mask<0,  0,  0,  0,  1,  1,  1,  1,  0,  0,  0,  0,  1,  1,  1,  1>();

    ri_512 const      perm4 = make_perm_map<2,  3,  0,  1,  6,  7,  4,  5, 10, 11,  8,  9, 14, 15, 12, 13>();
    constexpr msk_512 mask4 = make_bit_mask<0,  0,  1,  1,  0,  0,  1,  1,  0,  0,  1,  1,  0,  0,  1,  1>();

    ri_512 const      perm5 = make_perm_map<1,  0,  3,  2,  5,  4,  7,  6,  9,  8, 11, 10, 13, 12, 15, 14>();
    constexpr msk_512 mask5 = make_bit_mask<0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1>();

    ...
}
```

Function sort_two_lanes_of_8()

```
KEWB_FORCE_INLINE rf_512
sort_two_lanes_of_8(rf_512 vals)
{
    //- Precompute the permutations and bitmasks for the 6 stages of this bitonic sorting sequence.
    //-
    //-
    //      0  1  2  3  4  5  6  7  0  1  2  3  4  5  6  7
    //      -----
    ri_512 const    perm0 = make_perm_map<1,  0,  3,  2,  5,  4,  7,  6,  9,  8, 11, 10, 13, 12, 15, 14>();
    constexpr msk_512 mask0 = make_bit_mask<0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1>();

    ri_512 const    perm1 = make_perm_map<3,  2,  1,  0,  7,  6,  5,  4, 11, 10,  9,  8, 15, 14, 13, 12>();
    constexpr msk_512 mask1 = make_bit_mask<0,  0,  1,  1,  0,  0,  1,  1,  0,  0,  1,  1,  0,  0,  1,  1>();

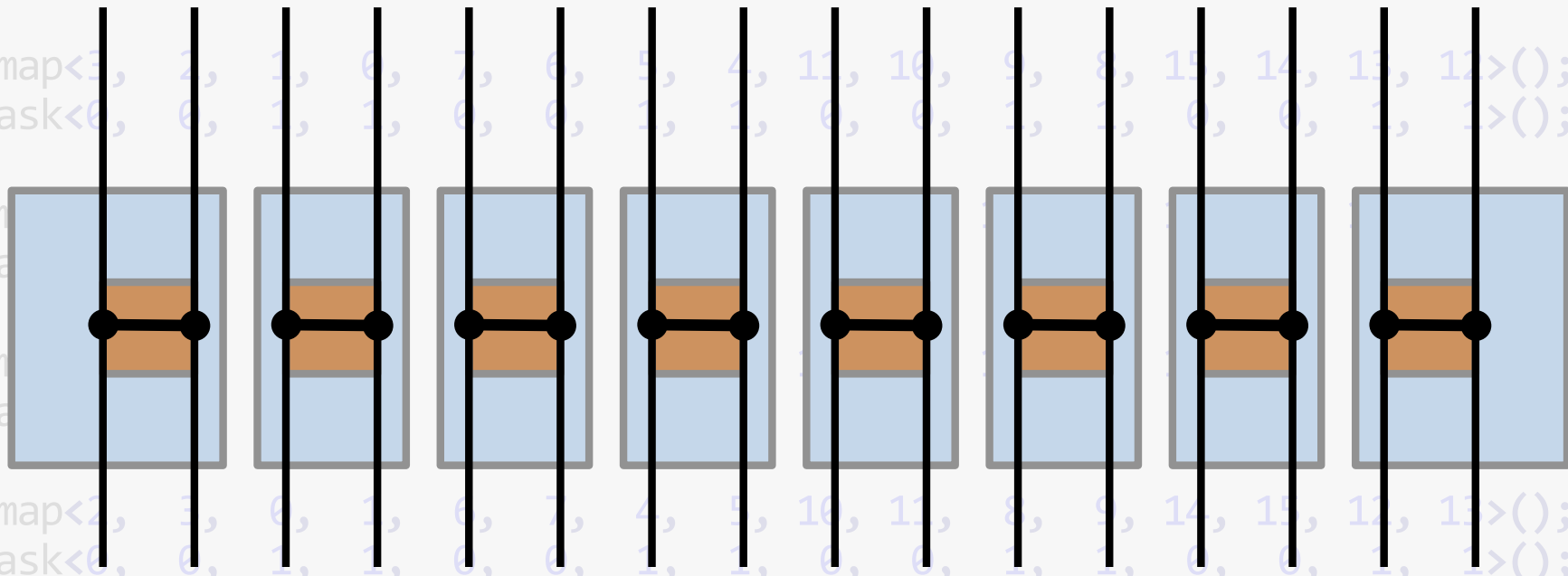
    ri_512 const    perm2 = make_perm_map<1,  0,  3,  2,  5,  4,  7,  6,  9,  8, 11, 10, 13, 12, 15, 14>();
    constexpr msk_512 mask2 = make_bit_mask<0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1>();

    ri_512 const    perm3 = make_perm_map<1,  0,  3,  2,  5,  4,  7,  6,  9,  8, 11, 10, 13, 12, 15, 14>();
    constexpr msk_512 mask3 = make_bit_mask<0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1>();

    ri_512 const    perm4 = make_perm_map<1,  0,  3,  2,  5,  4,  7,  6,  9,  8, 11, 10, 13, 12, 15, 14>();
    constexpr msk_512 mask4 = make_bit_mask<0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1>();

    ri_512 const    perm5 = make_perm_map<1,  0,  3,  2,  5,  4,  7,  6,  9,  8, 11, 10, 13, 12, 15, 14>();
    constexpr msk_512 mask5 = make_bit_mask<0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1>();

    ...
}
```



Function sort_two_lanes_of_8()

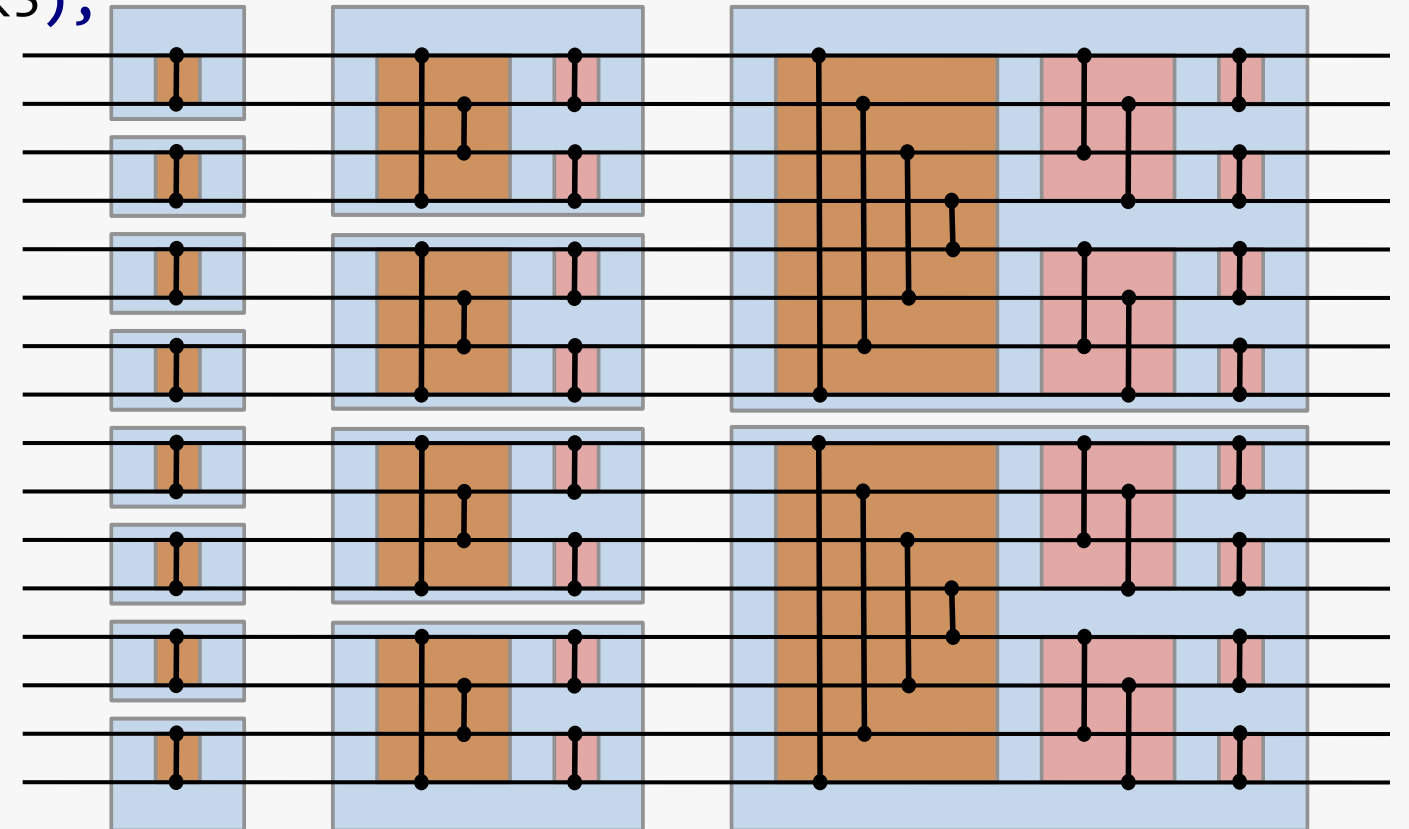
```
KEWB_FORCE_INLINE rf_512  
sort_two_lanes_of_8(rf_512 vals)  
{
```

```
...
```

```
vals = compare_with_exchange(vals, perm0, mask0);  
vals = compare_with_exchange(vals, perm1, mask1);  
vals = compare_with_exchange(vals, perm2, mask2);  
vals = compare_with_exchange(vals, perm3, mask3);  
vals = compare_with_exchange(vals, perm4, mask4);  
vals = compare_with_exchange(vals, perm5, mask5);
```

```
return vals;
```

```
}
```



Function `sort_two_lanes_of_8()`

vals	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
------	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	---

Function sort_two_lanes_of_8()

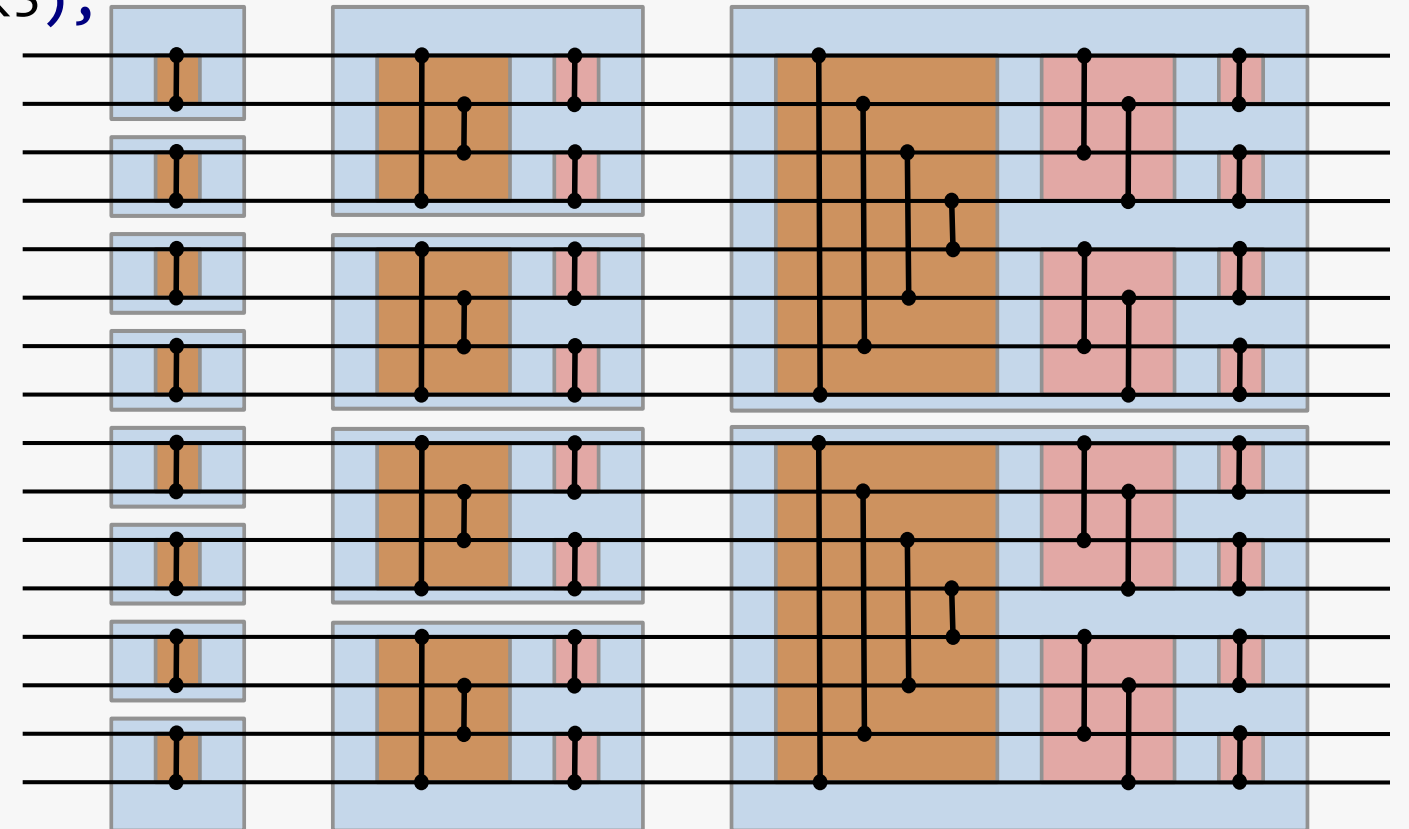
```
KEWB_FORCE_INLINE rf_512  
sort_two_lanes_of_8(rf_512 vals)  
{
```

```
...
```

```
vals = compare_with_exchange(vals, perm0, mask0);  
vals = compare_with_exchange(vals, perm1, mask1);  
vals = compare_with_exchange(vals, perm2, mask2);  
vals = compare_with_exchange(vals, perm3, mask3);  
vals = compare_with_exchange(vals, perm4, mask4);  
vals = compare_with_exchange(vals, perm5, mask5);
```

```
return vals;
```

```
}
```



Function sort_two_lanes_of_8()

vals	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
vals ₀	15	16	13	14	11	12	9	10	7	8	5	6	3	4	1	2

Function sort_two_lanes_of_8()

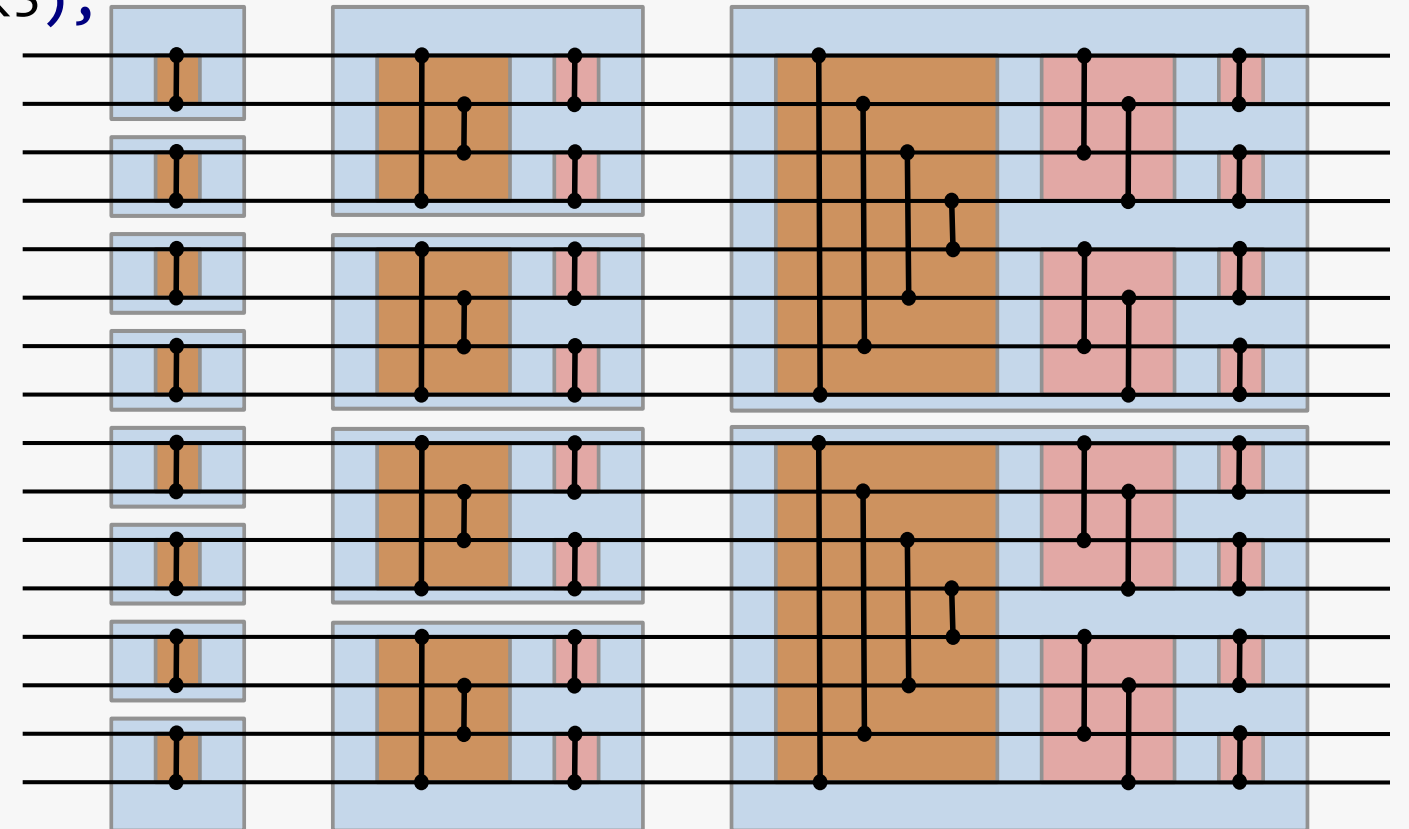
```
KEWB_FORCE_INLINE rf_512  
sort_two_lanes_of_8(rf_512 vals)  
{
```

```
...
```

```
vals = compare_with_exchange(vals, perm0, mask0);  
vals = compare_with_exchange(vals, perm1, mask1);  
vals = compare_with_exchange(vals, perm2, mask2);  
vals = compare_with_exchange(vals, perm3, mask3);  
vals = compare_with_exchange(vals, perm4, mask4);  
vals = compare_with_exchange(vals, perm5, mask5);
```

```
return vals;
```

```
}
```



Function sort_two_lanes_of_8()

vals	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
vals ₀	15	16	13	14	11	12	9	10	7	8	5	6	3	4	1	2
vals ₁	14	13	16	15	10	9	12	11	6	5	8	7	2	1	4	3

Function sort_two_lanes_of_8()

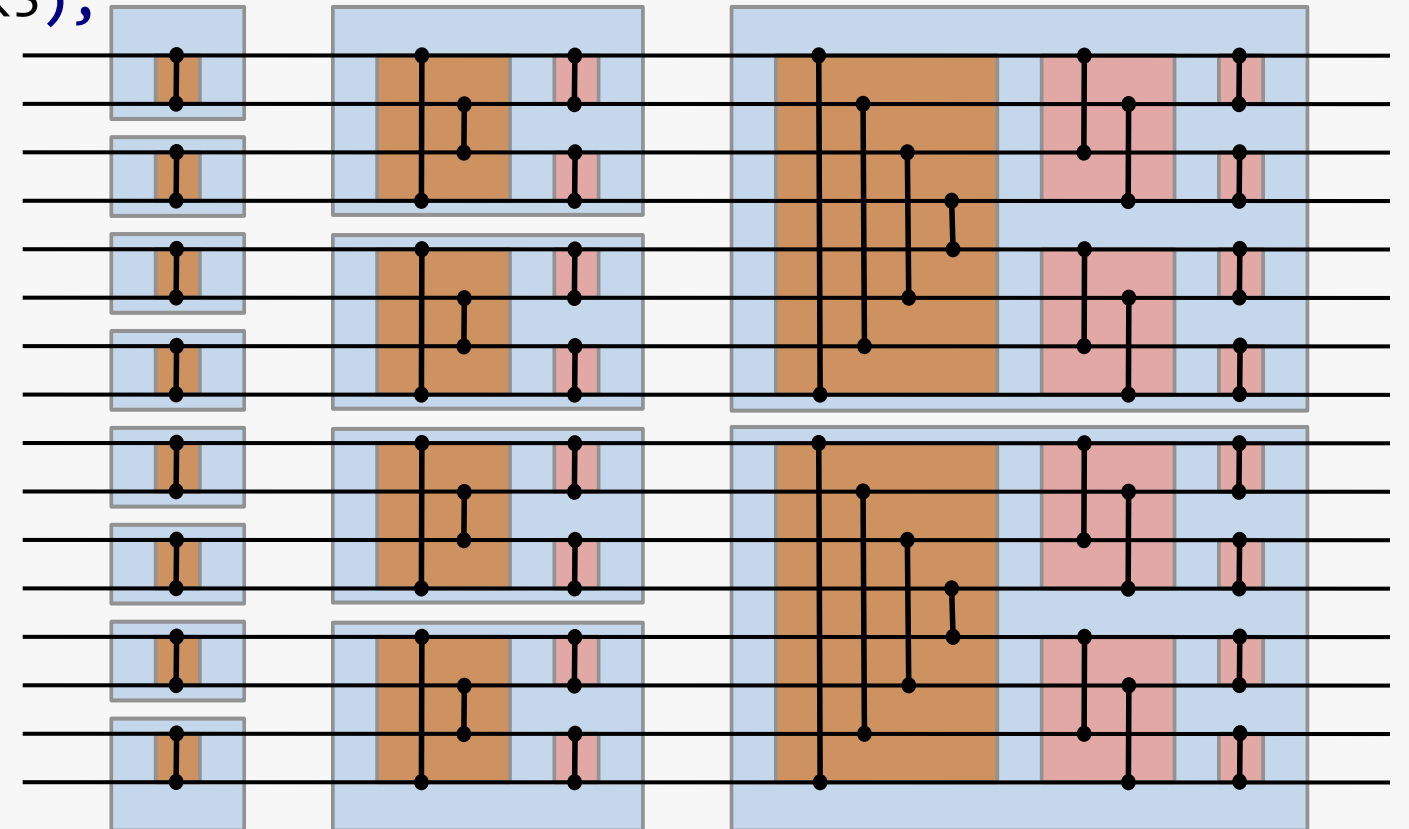
```
KEWB_FORCE_INLINE rf_512  
sort_two_lanes_of_8(rf_512 vals)  
{
```

```
...
```

```
vals = compare_with_exchange(vals, perm0, mask0);  
vals = compare_with_exchange(vals, perm1, mask1);  
vals = compare_with_exchange(vals, perm2, mask2);  
vals = compare_with_exchange(vals, perm3, mask3);  
vals = compare_with_exchange(vals, perm4, mask4);  
vals = compare_with_exchange(vals, perm5, mask5);
```

```
return vals;
```

```
}
```



Function `sort_two_lanes_of_8()`

<code>vals</code>	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
<code>vals₀</code>	15	16	13	14	11	12	9	10	7	8	5	6	3	4	1	2
<code>vals₁</code>	14	13	16	15	10	9	12	11	6	5	8	7	2	1	4	3
<code>vals₂</code>	13	14	15	16	9	10	11	12	5	6	7	8	1	2	3	4

Function sort_two_lanes_of_8()

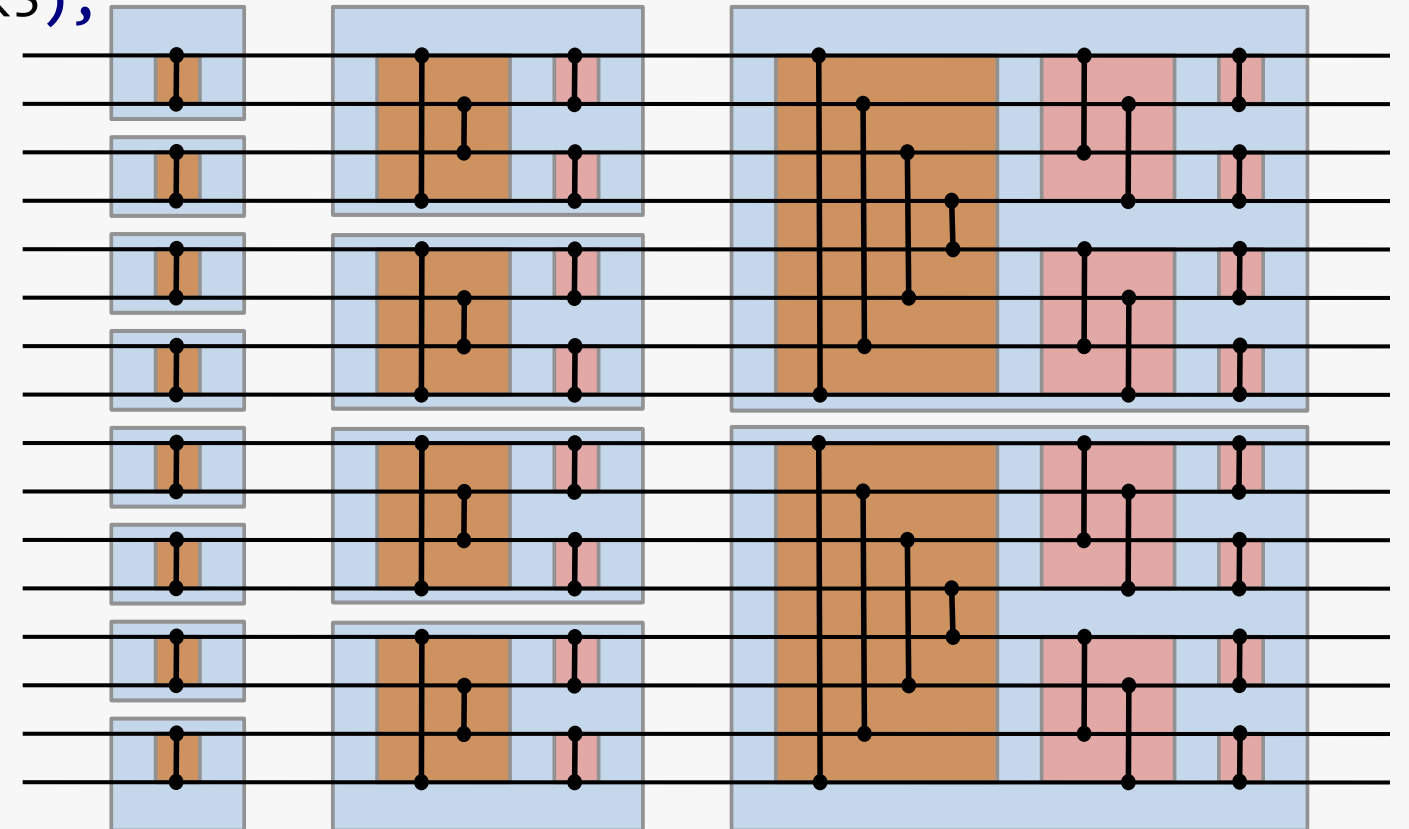
```
KEWB_FORCE_INLINE rf_512  
sort_two_lanes_of_8(rf_512 vals)  
{
```

```
...
```

```
vals = compare_with_exchange(vals, perm0, mask0);  
vals = compare_with_exchange(vals, perm1, mask1);  
vals = compare_with_exchange(vals, perm2, mask2);  
vals = compare_with_exchange(vals, perm3, mask3);  
vals = compare_with_exchange(vals, perm4, mask4);  
vals = compare_with_exchange(vals, perm5, mask5);
```

```
return vals;
```

```
}
```



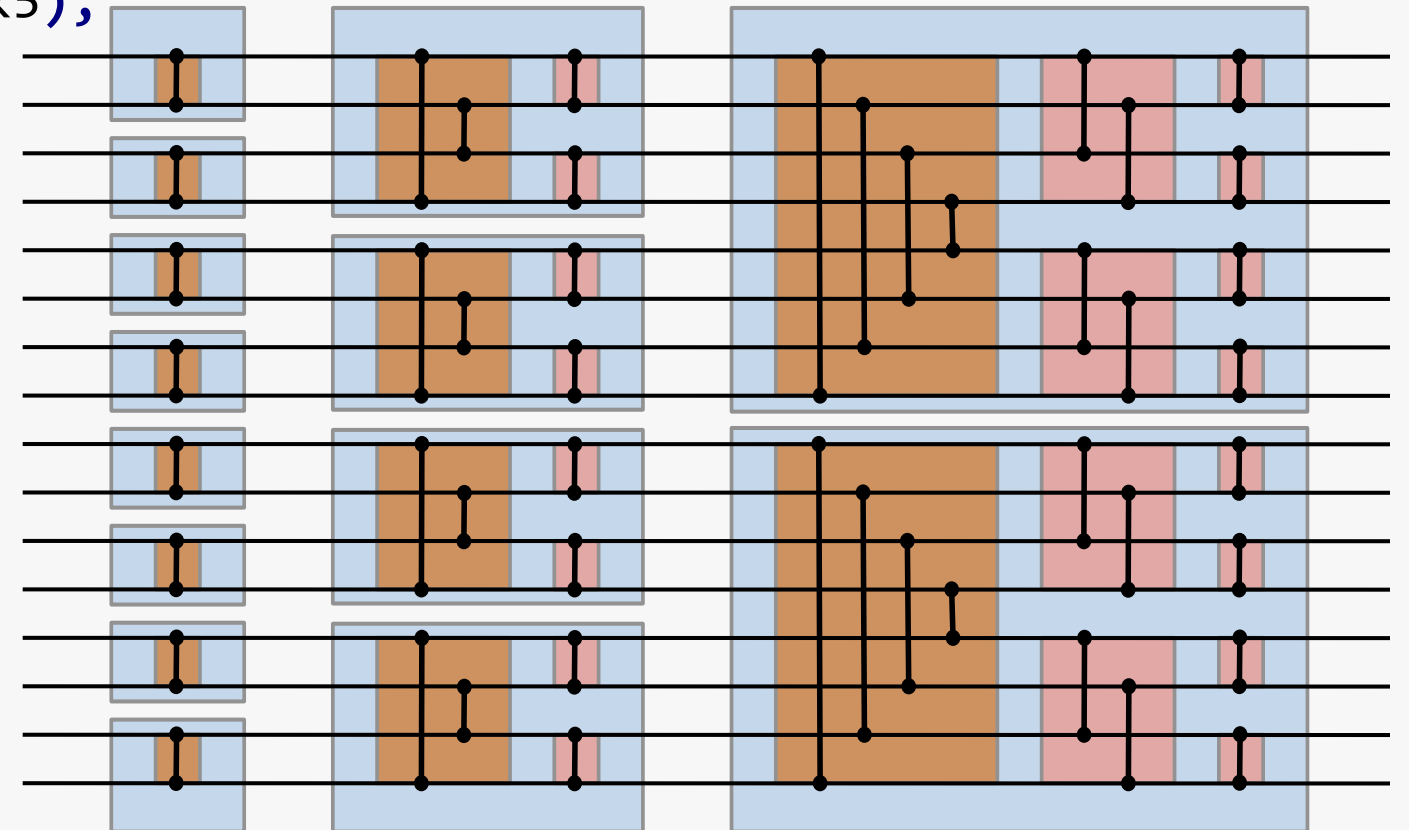
Function sort_two_lanes_of_8()

vals	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
vals ₀	15	16	13	14	11	12	9	10	7	8	5	6	3	4	1	2
vals ₁	14	13	16	15	10	9	12	11	6	5	8	7	2	1	4	3
vals ₂	13	14	15	16	9	10	11	12	5	6	7	8	1	2	3	4
vals ₃	12	11	10	9	16	15	14	13	4	3	2	1	8	7	6	5

Function sort_two_lanes_of_8()

```
KEWB_FORCE_INLINE rf_512
sort_two_lanes_of_8(rf_512 vals)
{
    ...
    vals = compare_with_exchange(vals, perm0, mask0);
    vals = compare_with_exchange(vals, perm1, mask1);
    vals = compare_with_exchange(vals, perm2, mask2);
    vals = compare_with_exchange(vals, perm3, mask3);
    vals = compare_with_exchange(vals, perm4, mask4);
    vals = compare_with_exchange(vals, perm5, mask5);

    return vals;
}
```



Function sort_two_lanes_of_8()

vals	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
vals ₀	15	16	13	14	11	12	9	10	7	8	5	6	3	4	1	2
vals ₁	14	13	16	15	10	9	12	11	6	5	8	7	2	1	4	3
vals ₂	13	14	15	16	9	10	11	12	5	6	7	8	1	2	3	4
vals ₃	12	11	10	9	16	15	14	13	4	3	2	1	8	7	6	5
vals ₄	10	9	12	11	14	13	16	15	2	1	4	3	6	5	8	7

Function sort_two_lanes_of_8()

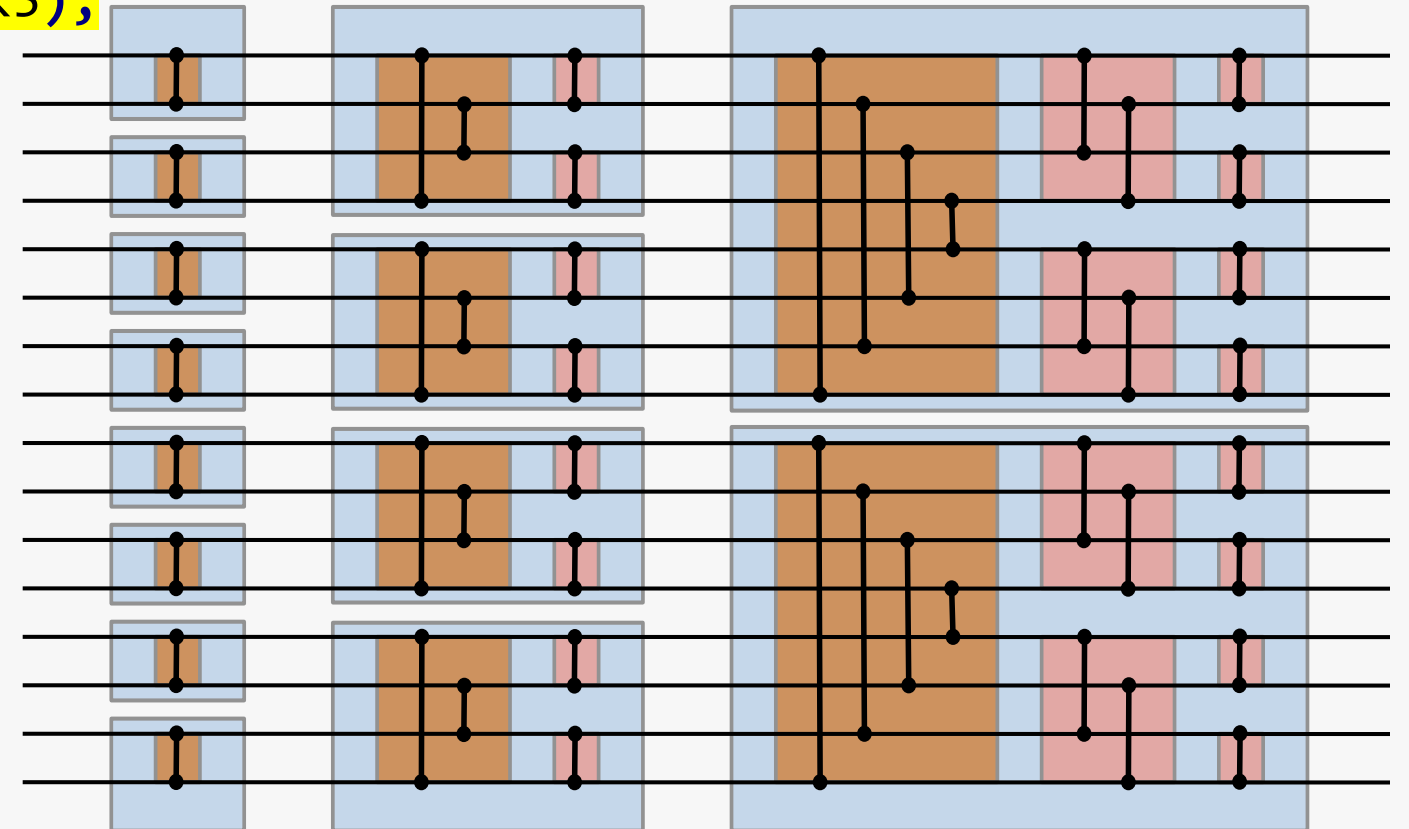
```
KEWB_FORCE_INLINE rf_512  
sort_two_lanes_of_8(rf_512 vals)  
{
```

```
...
```

```
vals = compare_with_exchange(vals, perm0, mask0);  
vals = compare_with_exchange(vals, perm1, mask1);  
vals = compare_with_exchange(vals, perm2, mask2);  
vals = compare_with_exchange(vals, perm3, mask3);  
vals = compare_with_exchange(vals, perm4, mask4);  
vals = compare_with_exchange(vals, perm5, mask5);
```

```
return vals;
```

```
}
```

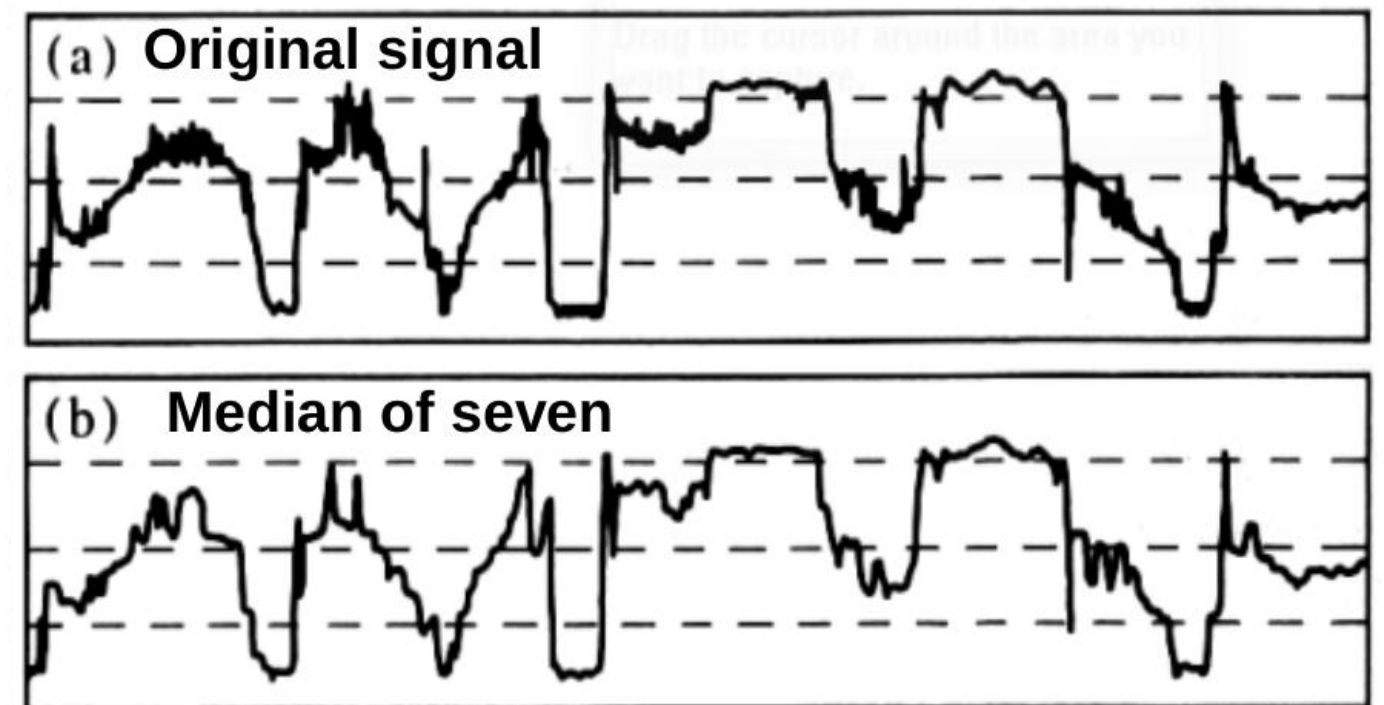
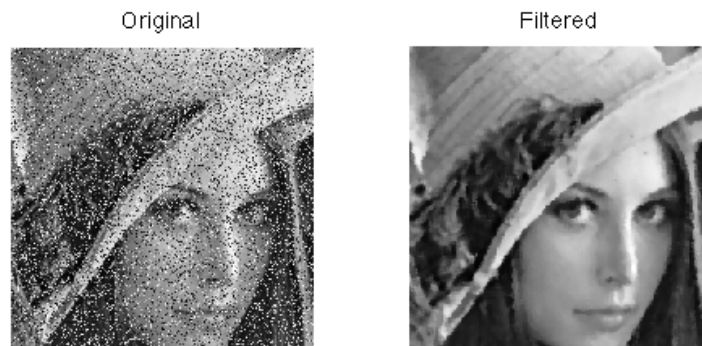


Function sort_two_lanes_of_8()

vals	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
vals ₀	15	16	13	14	11	12	9	10	7	8	5	6	3	4	1	2
vals ₁	14	13	16	15	10	9	12	11	6	5	8	7	2	1	4	3
vals ₂	13	14	15	16	9	10	11	12	5	6	7	8	1	2	3	4
vals ₃	12	11	10	9	16	15	14	13	4	3	2	1	8	7	6	5
vals ₄	10	9	12	11	14	13	16	15	2	1	4	3	6	5	8	7
vals ₅	9	10	11	12	13	14	15	16	1	2	3	4	5	6	7	8

Fast Linear Median Filter

- If we can sort two lanes of eight, why not two lanes of seven?
- With a fast sort, we could implement a fast median-of-seven linear filter
 - **median**: a value separating the higher half from the lower half of a data sample
 - If I have an array **a** of seven integers, and they are sorted, the median is **a[3]**
- Median filters are good at
 - Preserving edge features in a signal
 - Eliminating outliers without blur
 - Preserving large discontinuities
 - De-noising



Function avx_median_of_7()

```
void
avx_median_of_7(float* pdst, float const* psrc, size_t const buf_len)
{
    rf_512 prev;    //- Bottom of the input data window
    rf_512 curr;    //- Middle of the input data window
    rf_512 next;    //- Top of the input data window
    rf_512 lo;      //- Primary work register
    rf_512 hi;      //- Upper work data register; feeds values into the top of 'lo'
    rf_512 data;    //- Holds output prior to store operation
    rf_512 work;    //- Accumulator

    rf_512 const first = load_value(psrc[0]);

    //- This permutation specifies how to load the two lanes of 7.
    //
    ri_512 const load_perm = make_perm_map<0,1,2,3,4,5,6,7,1,2,3,4,5,6,7,8>();

    //- This permutation specifies which elements to save.
    //
    ri_512 const save_perm = make_perm_map<3,11,3,11,3,11,3,11,3,11,3,11,3,11,3,11>();

    ...
}
```

Function avx_median_of_7()

```
void
avx_median_of_7(float* pdst, float const* psrc, size_t const buf_len)
{
    rf_512 prev;    //- Bottom of the input data window
    rf_512 curr;    //- Middle of the input data window
    rf_512 next;    //- Top of the input data window
    rf_512 lo;      //- Primary work register
    rf_512 hi;      //- Upper work data register; feeds values into the top of 'lo'
    rf_512 data;    //- Holds output prior to store operation
    rf_512 work;    //- Accumulator

    rf_512 const first = load_value(psrc[0]);

    //- This permutation specifies how to load the two lanes of 7.
    //
    ri_512 const load_perm = make_perm_map<0,1,2,3,4,5,6,7,1,2,3,4,5,6,7,8>();

    //- This permutation specifies which elements to save.
    //
    ri_512 const save_perm = make_perm_map<3,11,3,11,3,11,3,11,3,11,3,11,3,11,3,11>();

    ...
}
```

Function avx_median_of_7()

```
void
avx_median_of_7(float* pdst, float const* psrc, size_t const buf_len)
{
    rf_512 prev;    //- Bottom of the input data window
    rf_512 curr;    //- Middle of the input data window
    rf_512 next;    //- Top of the input data window
    rf_512 lo;      //- Primary work register
    rf_512 hi;      //- Upper work data register; feeds values into the top of 'lo'
    rf_512 data;    //- Holds output prior to store operation
    rf_512 work;    //- Accumulator

    rf_512 const first = load_value(psrc[0]);

    //- This permutation specifies how to load the two lanes of 7.
    //
    ri_512 const load_perm = make_perm_map<0,1,2,3,4,5,6,7,1,2,3,4,5,6,7,8>();

    //- This permutation specifies which elements to save.
    //
    ri_512 const save_perm = make_perm_map<3,11,3,11,3,11,3,11,3,11,3,11,3,11,3,11>();

    ...
}
```

Function avx_median_of_7()

```
void
avx_median_of_7(float* pdst, float const* psrc, size_t const buf_len)
{
    rf_512 prev;    //- Bottom of the input data window
    rf_512 curr;    //- Middle of the input data window
    rf_512 next;    //- Top of the input data window
    rf_512 lo;      //- Primary work register
    rf_512 hi;      //- Upper work data register; feeds values into the top of 'lo'
    rf_512 data;    //- Holds output prior to store operation
    rf_512 work;    //- Accumulator

    rf_512 const first = load_value(psrc[0]);

    //- This permutation specifies how to load the two lanes of 7.
    //
    ri_512 const load_perm = make_perm_map<0,1,2,3,4,5,6,7,1,2,3,4,5,6,7,8>();

    //- This permutation specifies which elements to save.
    //
    ri_512 const save_perm = make_perm_map<3,11,3,11,3,11,3,11,3,11,3,11,3,11,3,11>();

    ...
}
```

Function avx_median_of_7()

```
void
avx_median_of_7(float* pdst, float const* psrc, size_t const buf_len)
{
    ...

    //- This permutation specifies how to load the two lanes of 7.
    //
    ri_512 const load_perm = make_perm_map<0,1,2,3,4,5,6,7,1,2,3,4,5,6,7,8>();

    //- This permutation specifies which elements to save.
    //
    ri_512 const save_perm = make_perm_map<3,11,3,11,3,11,3,11,3,11,3,11,3,11,3,11>();

    //- This is a bitmask pattern for picking out adjacent elements.
    //
    constexpr msk_512 save = make_bit_mask<1,1>();

    //- This array of bitmasks specifies which pair of elements to blend into the result.
    //
    constexpr msk_512 save_mask[8] = {save << 0, save << 2, save << 4, save << 6,
                                       save << 8, save << 10, save << 12, save << 14};

    ...
}
```

Function avx_median_of_7()

```
void
avx_median_of_7(float* pdst, float const* psrc, size_t const buf_len)
{
    ...

    //- This permutation specifies how to load the two lanes of 7.
    //
    ri_512 const load_perm = make_perm_map<0,1,2,3,4,5,6,7,1,2,3,4,5,6,7,8>();

    //- This permutation specifies which elements to save.
    //
    ri_512 const save_perm = make_perm_map<3,11,3,11,3,11,3,11,3,11,3,11,3,11,3,11>();

    //- This is a bitmask pattern for picking out adjacent elements.
    //
    constexpr msk_512 save = make_bit_mask<1,1>();

    //- This array of bitmasks specifies which pair of elements to blend into the result.
    //
    constexpr msk_512 save_mask[8] = {save << 0, save << 2, save << 4, save << 6,
                                     save << 8, save << 10, save << 12, save << 14};

    ...
}
```


Function avx_median_of_7()

```
void
avx_median_of_7(float* pdst, float const* psrc, size_t const buf_len)
{
    ...

    //- Preload the initial input data window
    //
    size_t  read  = 0;
    size_t  wrote = 0;

    curr  = first;
    next  = load_from(psrc);
    read += 16;

    ...
}
```

Function avx_median_of_7()



Function avx_median_of_7()

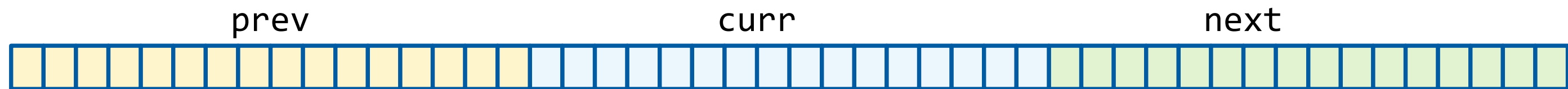
```
void
avx_median_of_7(float* pdst, float const* psrc, size_t const buf_len)
{
    ...
    while (used < (buf_len + 16))
    {
        prev = curr;
        curr = next;
        next = load_from(psrc + read);
        read += 16;

        lo = shift_up_with_carry<3>(prev, curr); // - Init the work data registers to the
        hi = shift_up_with_carry<3>(curr, next); // correct offset in the input data window

        for (int i = 0; i < 8; ++i) // - Perform two sorts of 7 at a time, in lanes of 8
        {
            work = permute(lo, load_perm);
            work = sort_two_lanes_of_7(work);
            data = masked_permute(data, work, save_perm, save_mask[i]);
            in_place_shift_down_with_carry<2>(lo, hi);
        }

        store_to(pdst + wrote, data);
        wrote += 16;
    }
}
```

Function avx_median_of_7()



Function avx_median_of_7()

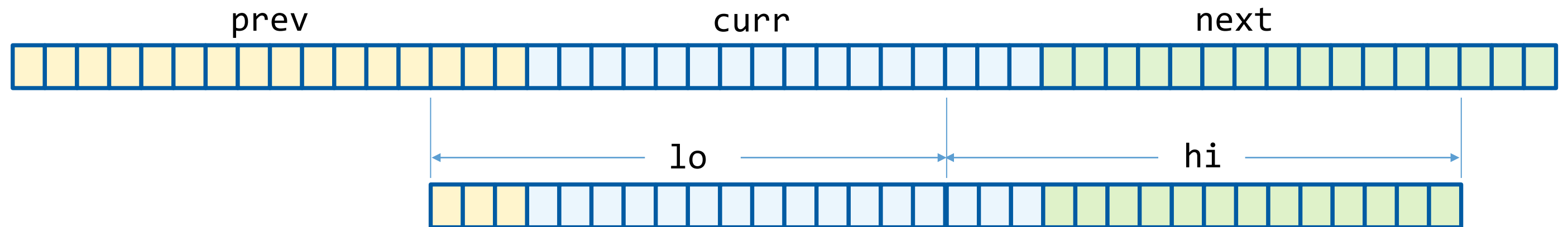
```
void
avx_median_of_7(float* pdst, float const* psrc, size_t const buf_len)
{
    ...
    while (used < (buf_len + 16))
    {
        prev = curr;
        curr = next;
        next = load_from(psrc + read);
        read += 16;

        lo = shift_up_with_carry<3>(prev, curr); //- Init the work data registers to the
        hi = shift_up_with_carry<3>(curr, next); // correct offset in the input data window

        for (int i = 0; i < 8; ++i)                //- Perform two sorts of 7 at a time, in lanes of 8
        {
            work = permute(lo, load_perm);
            work = sort_two_lanes_of_7(work);
            data = masked_permute(data, work, save_perm, save_mask[i]);
            in_place_shift_down_with_carry<2>(lo, hi);
        }

        store_to(pdst + wrote, data);
        wrote += 16;
    }
}
```

Function avx_median_of_7()



Function avx_median_of_7()

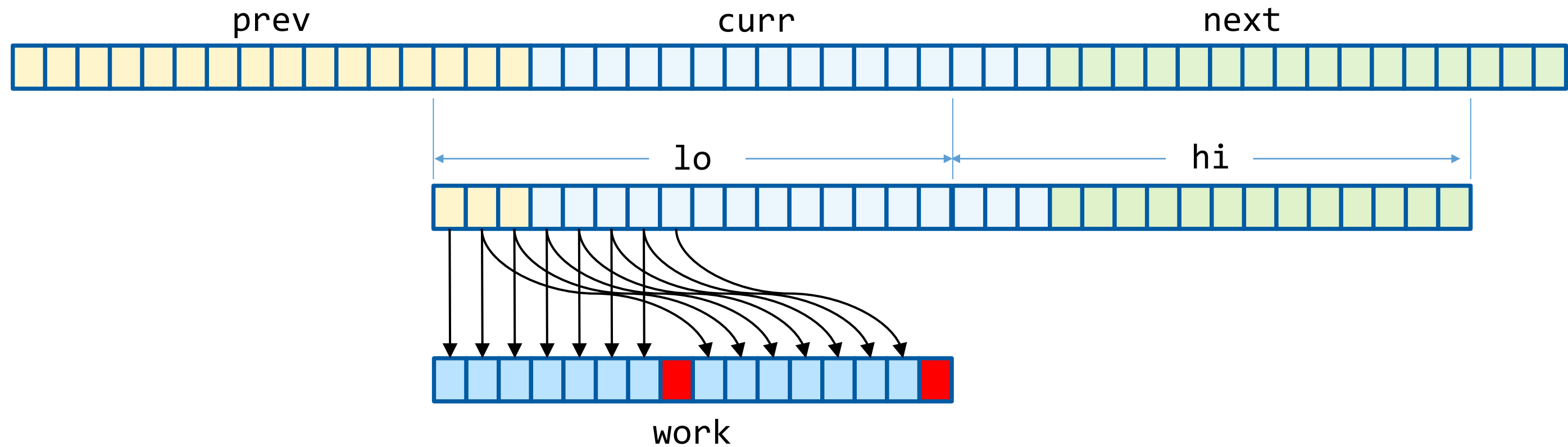
```
void
avx_median_of_7(float* pdst, float const* psrc, size_t const buf_len)
{
    ...
    while (used < (buf_len + 16))
    {
        prev = curr;
        curr = next;
        next = load_from(psrc + read);
        read += 16;

        lo = shift_up_with_carry<3>(prev, curr); // - Init the work data registers to the
        hi = shift_up_with_carry<3>(curr, next); // correct offset in the input data window

        for (int i = 0; i < 8; ++i) // - Perform two sorts of 7 at a time, in lanes of 8
        {
            work = permute(lo, load_perm);
            work = sort_two_lanes_of_7(work);
            data = masked_permute(data, work, save_perm, save_mask[i]);
            in_place_shift_down_with_carry<2>(lo, hi);
        }

        store_to(pdst + wrote, data);
        wrote += 16;
    }
}
```

Function avx_median_of_7()



Function avx_median_of_7()

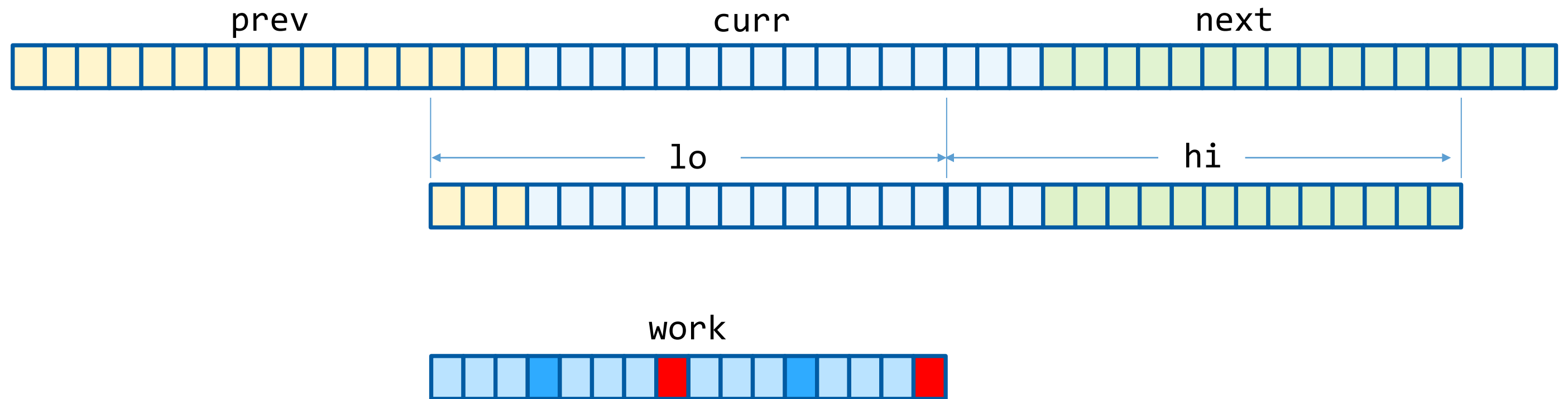
```
void
avx_median_of_7(float* pdst, float const* psrc, size_t const buf_len)
{
    ...
    while (used < (buf_len + 16))
    {
        prev = curr;
        curr = next;
        next = load_from(psrc + read);
        read += 16;

        lo = shift_up_with_carry<3>(prev, curr); // - Init the work data registers to the
        hi = shift_up_with_carry<3>(curr, next); // correct offset in the input data window

        for (int i = 0; i < 8; ++i) // - Perform two sorts of 7 at a time, in lanes of 8
        {
            work = permute(lo, load_perm);
            work = sort_two_lanes_of_7(work);
            data = masked_permute(data, work, save_perm, save_mask[i]);
            in_place_shift_down_with_carry<2>(lo, hi);
        }

        store_to(pdst + wrote, data);
        wrote += 16;
    }
}
```

Function avx_median_of_7()



Function avx_median_of_7()

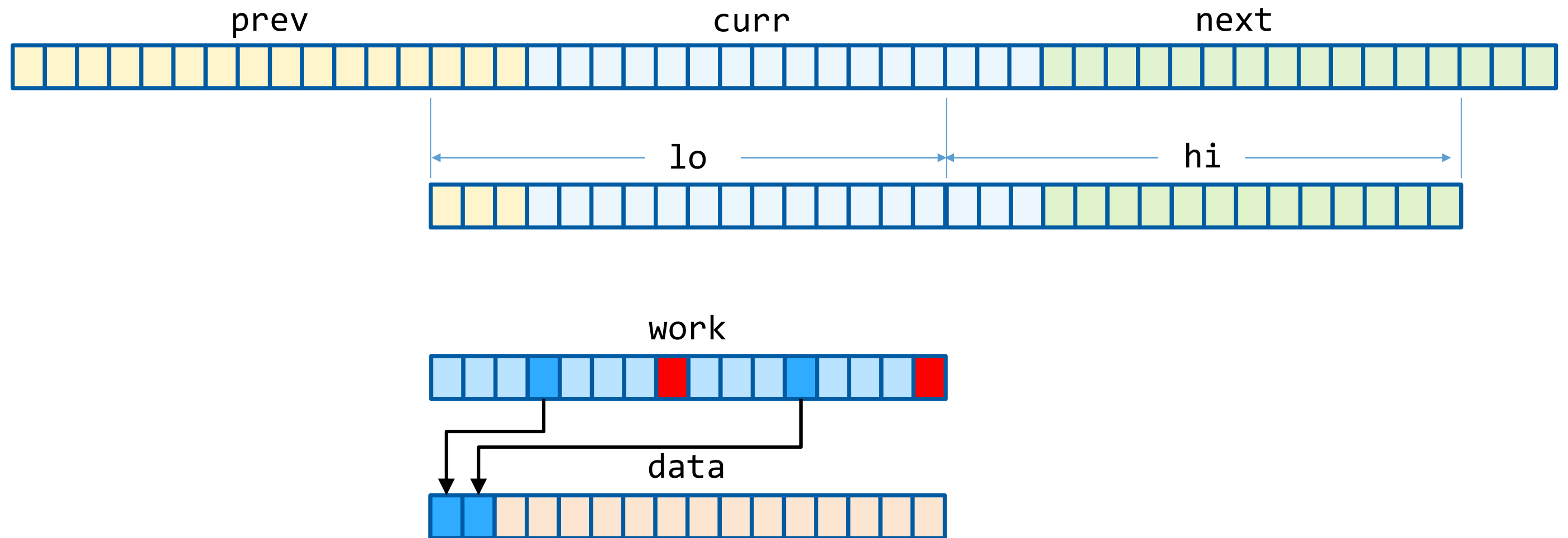
```
void
avx_median_of_7(float* pdst, float const* psrc, size_t const buf_len)
{
    ...
    while (used < (buf_len + 16))
    {
        prev = curr;
        curr = next;
        next = load_from(psrc + read);
        read += 16;

        lo = shift_up_with_carry<3>(prev, curr); // - Init the work data registers to the
        hi = shift_up_with_carry<3>(curr, next); // correct offset in the input data window

        for (int i = 0; i < 8; ++i) // - Perform two sorts of 7 at a time, in lanes of 8
        {
            work = permute(lo, load_perm);
            work = sort_two_lanes_of_7(work);
            data = masked_permute(data, work, save_perm, save_mask[i]);
            in_place_shift_down_with_carry<2>(lo, hi);
        }

        store_to(pdst + wrote, data);
        wrote += 16;
    }
}
```

Function avx_median_of_7()



Function avx_median_of_7()

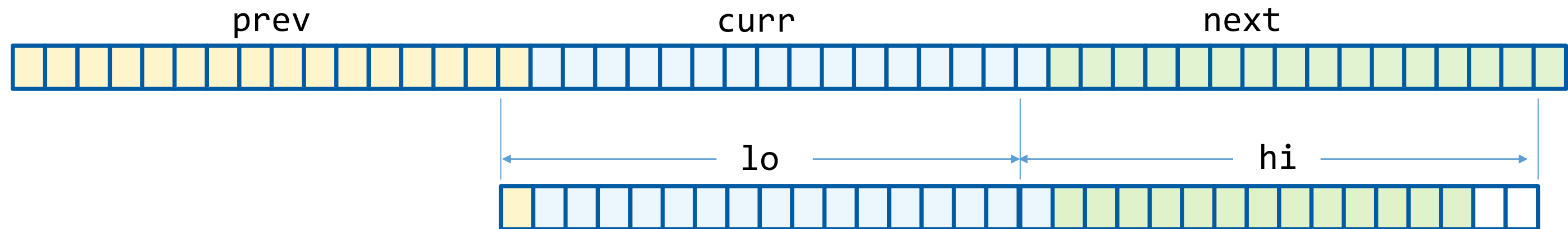
```
void
avx_median_of_7(float* pdst, float const* psrc, size_t const buf_len)
{
    ...
    while (used < (buf_len + 16))
    {
        prev = curr;
        curr = next;
        next = load_from(psrc + read);
        read += 16;

        lo = shift_up_with_carry<3>(prev, curr); // - Init the work data registers to the
        hi = shift_up_with_carry<3>(curr, next); // correct offset in the input data window

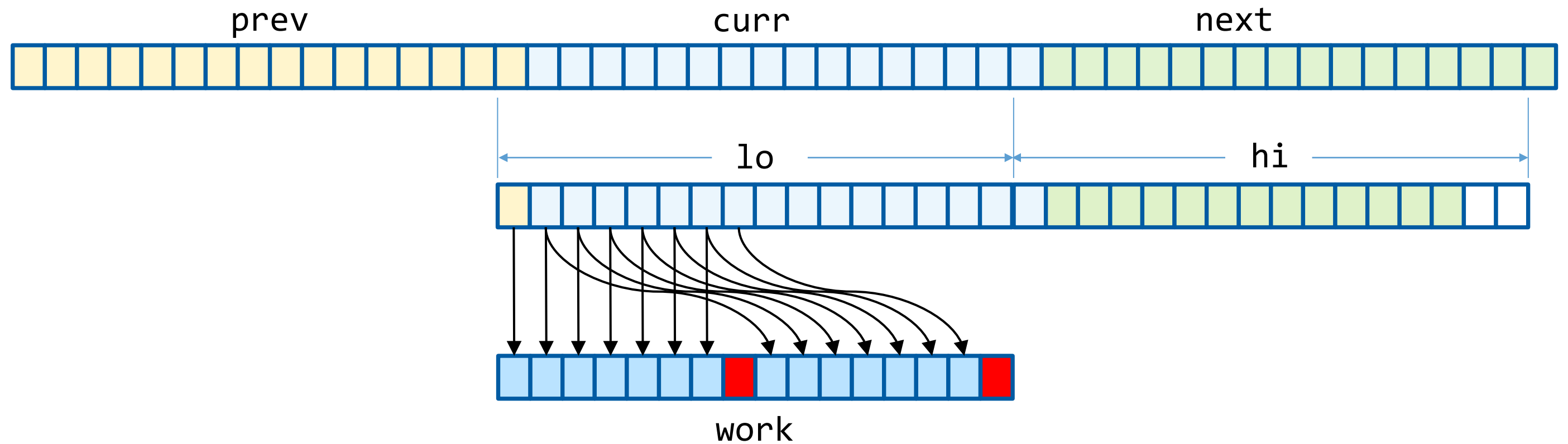
        for (int i = 0; i < 8; ++i) // - Perform two sorts of 7 at a time, in lanes of 8
        {
            work = permute(lo, load_perm);
            work = sort_two_lanes_of_7(work);
            data = masked_permute(data, work, save_perm, save_mask[i]);
            in_place_shift_down_with_carry<2>(lo, hi);
        }

        store_to(pdst + wrote, data);
        wrote += 16;
    }
}
```

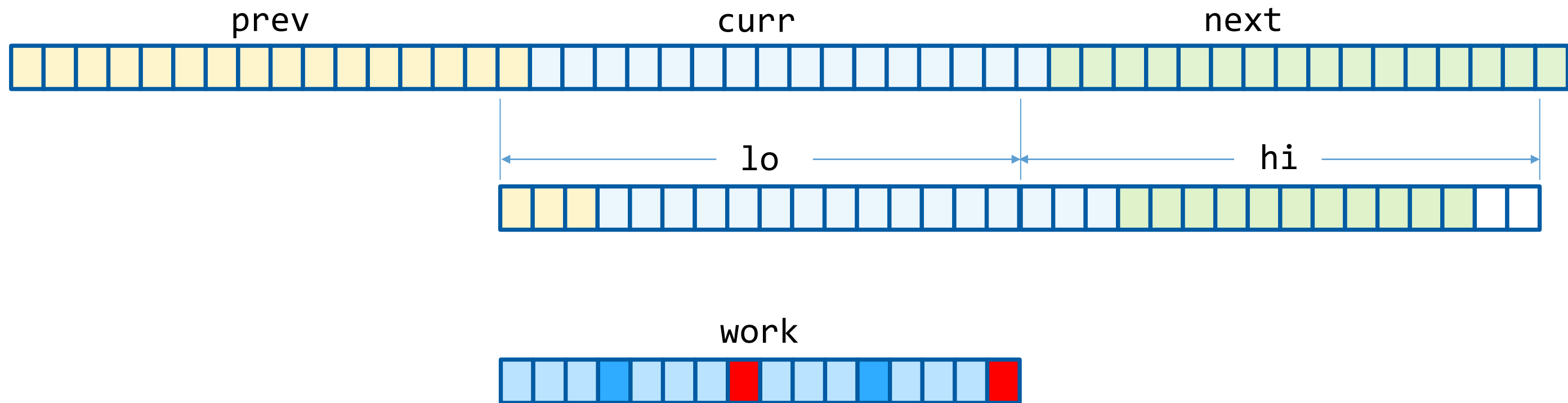
Function avx_median_of_7()



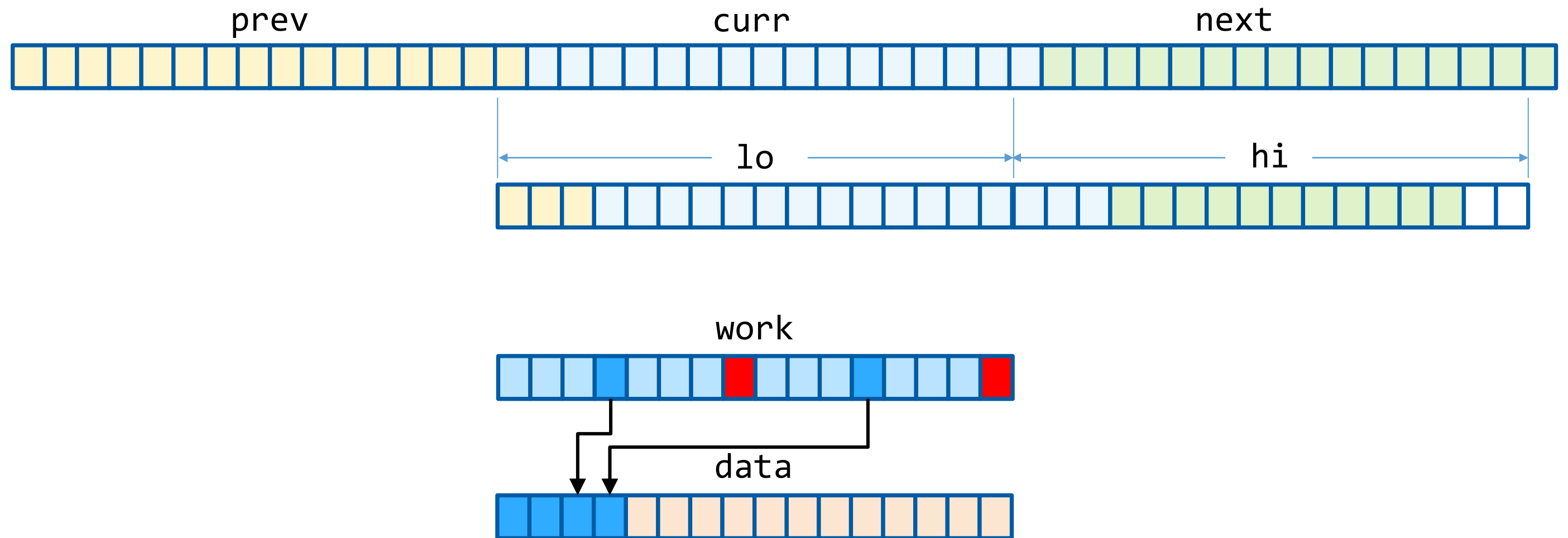
Function avx_median_of_7()



Function avx_median_of_7()



Function avx_median_of_7()



Function avx_median_of_7()

```
void
avx_median_of_7(float* pdst, float const* psrc, size_t const buf_len)
{
    ...
    while (used < (buf_len + 16))
    {
        prev = curr;
        curr = next;
        next = load_from(psrc + read);
        read += 16;

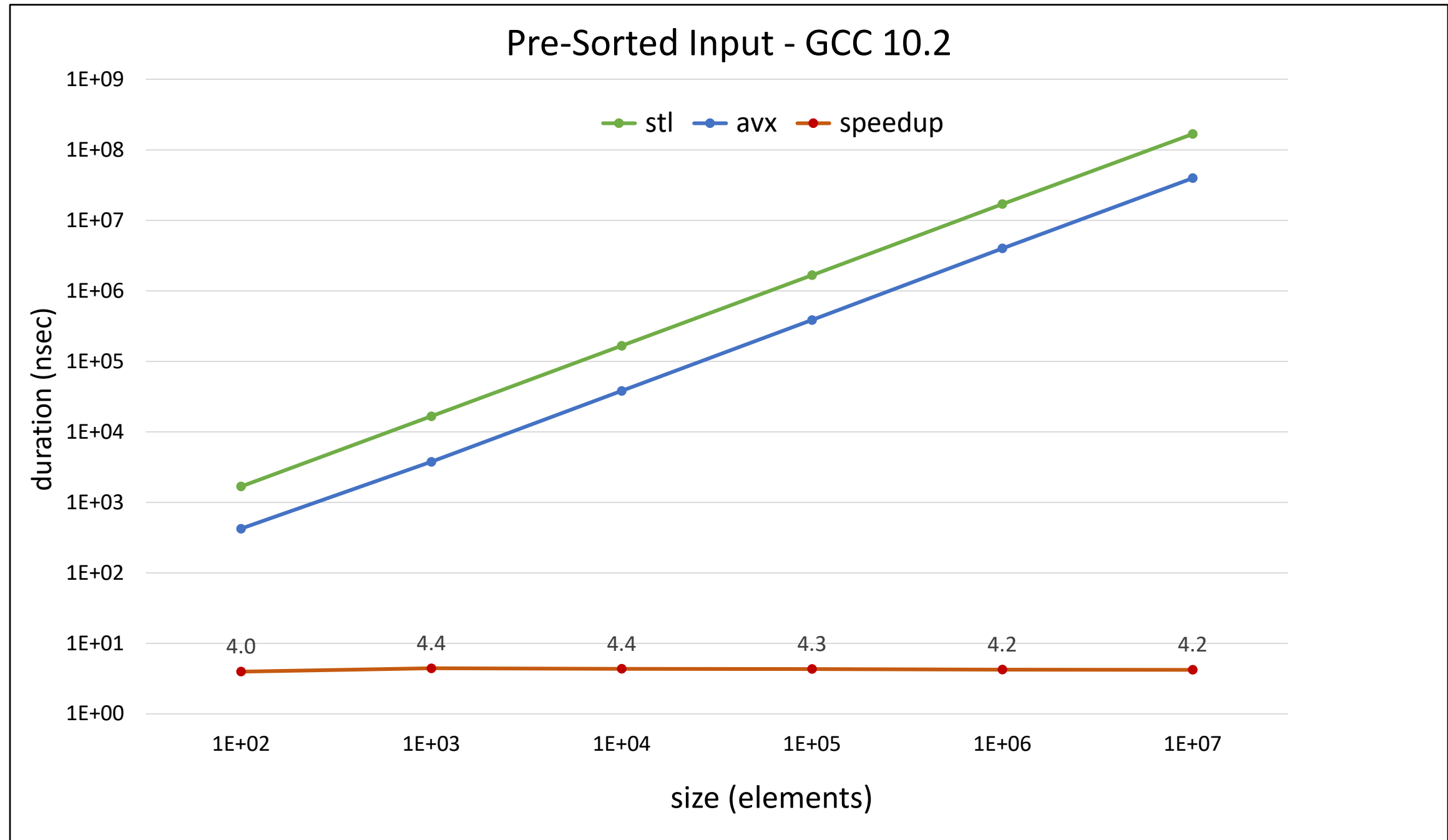
        lo = shift_up_with_carry<3>(prev, curr); // - Init the work data registers to the
        hi = shift_up_with_carry<3>(curr, next); // correct offset in the input data window

        for (int i = 0; i < 8; ++i) // - Perform two sorts of 7 at a time, in lanes of 8
        {
            work = permute(lo, load_perm);
            work = sort_two_lanes_of_7(work);
            data = masked_permute(data, work, save_perm, save_mask[i]);
            in_place_shift_down_with_carry<2>(lo, hi);
        }

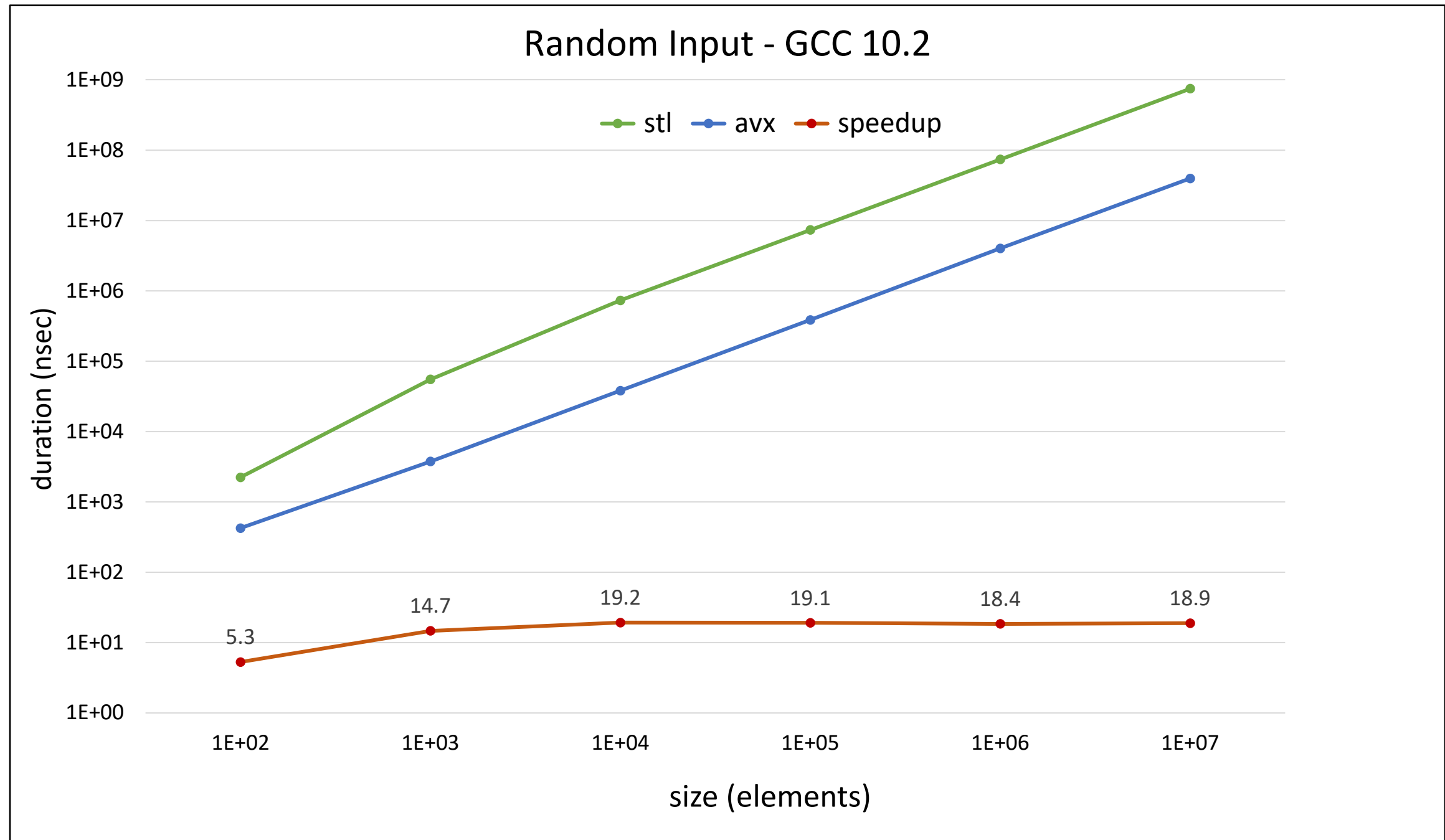
        store_to(pdst + wrote, data);
        wrote += 16;
    }
}
```

- Ubuntu 18.04 on Cascade Lake
 - **GCC 10.2**, all code compiled with `-O3 -mavx512 -march=skylake`
 - **Clang 10.0.1**, all code compiled with `-O3 -mavx512 -march=skylake`
- Element counts of 1E02 through 1E07 (by 10s)
 - Pre-sorted integers increasing monotonically
 - Randomly generated signed integers [-50, +50]
 - Always generated from the same seed
- Collect timings for each combination using two approaches
 - STL-based, performing `std::sort()` for every sequence of seven elements
 - Small-Kernel AVX

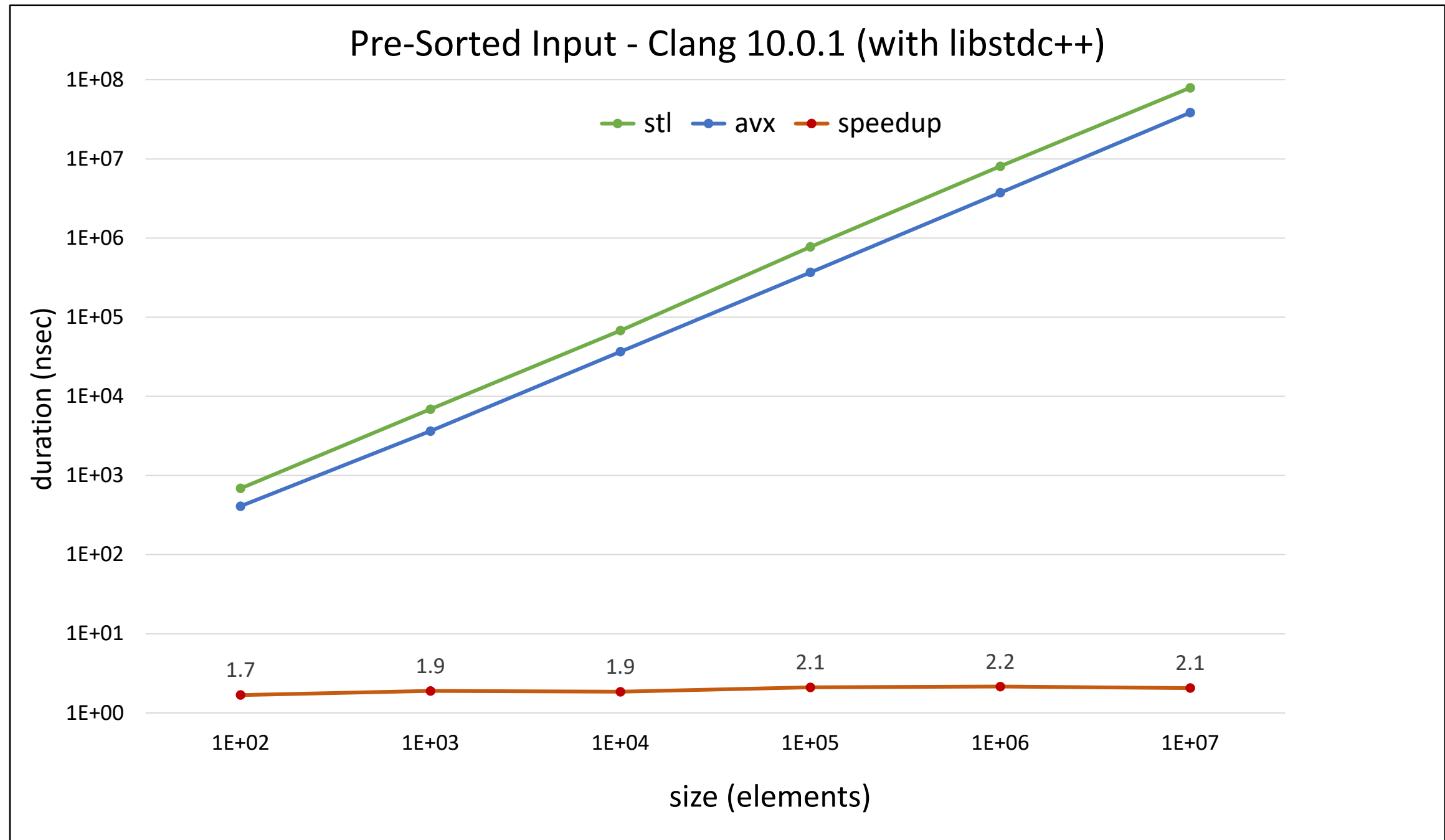
Results – Median-of-Seven – Sorted Input – GCC



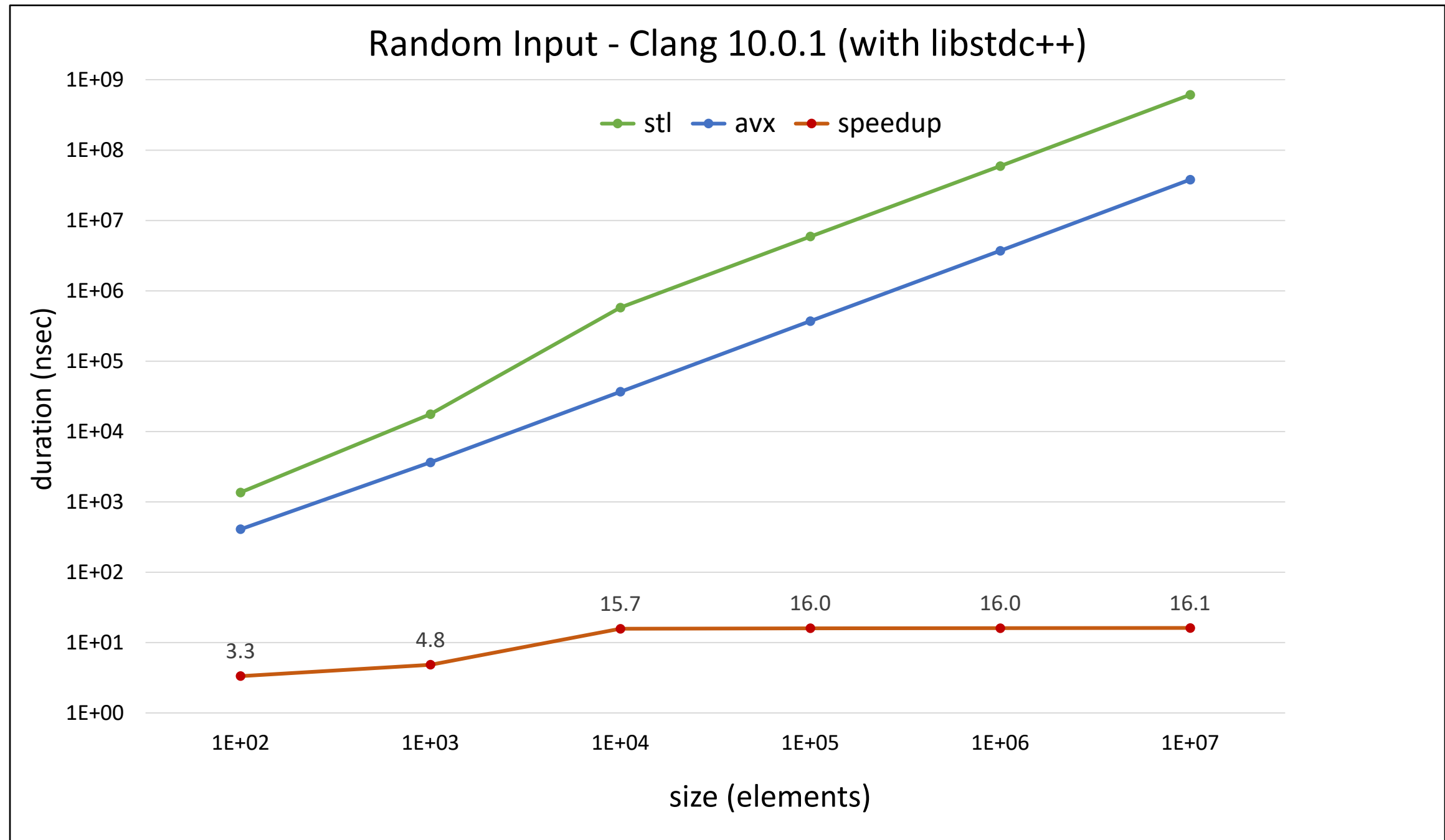
Results – Median-of-Seven – Random Input – GCC



Results – Median-of-Seven – Sorted Input – Clang



Results – Median-of-Seven – Random Input – Clang



Thank You for Attending!

Talk: github.com/BobSteagall/CppCon2020

Blog: bobsteagall.com

Adventures in SIMD Thinking (Part 2 of 2)

Bob Steagall
CppCon 2020

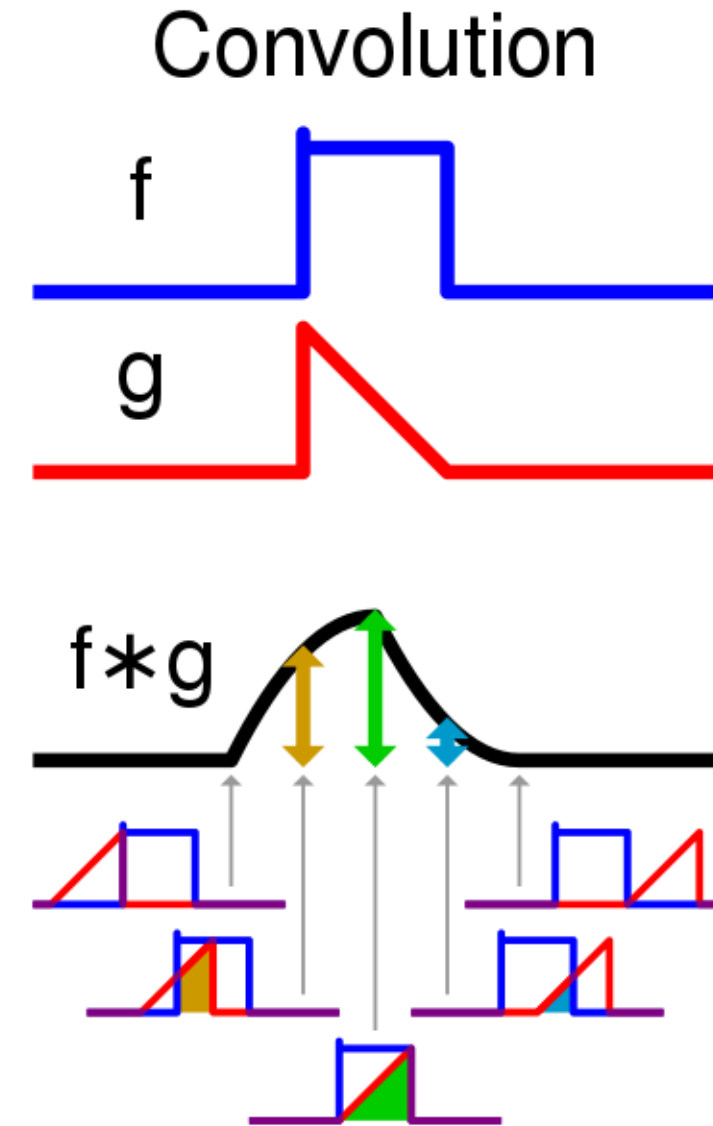


- Learn a little about Intel's SIMD facilities (disclaimer: I don't work for Intel)
- Create some useful functions in terms of AVX-512 intrinsics
- Try some SIMD-style thinking to tackle some interesting problems
 - Intra-register sorting
 - Fast linear median-of-seven filter
 - Fast small-kernel convolution
 - Faster (?) UTF-8 to UTF-32 conversion (with AVX2)
- No heavy code, but lots of pictures
 - Thinking "vertically"

Small-Kernel Convolution

Convolution

- f is a signal
- g is a kernel
- Output $f * g$ is the convolution
 - Every point of result $f * g$ is f weighted by every point of g
- Useful for smoothing and de-noising



Convolution

$$\begin{array}{rcccccccc} S = & s_0 & s_1 & s_2 & s_3 & s_4 & s_5 & s_6 \dots \\ K = & k_0 & k_1 & k_2 & & & & \end{array}$$

$$r_0 = \dots$$

$$r_1 = s_0k_0 + s_1k_1 + s_2k_2$$

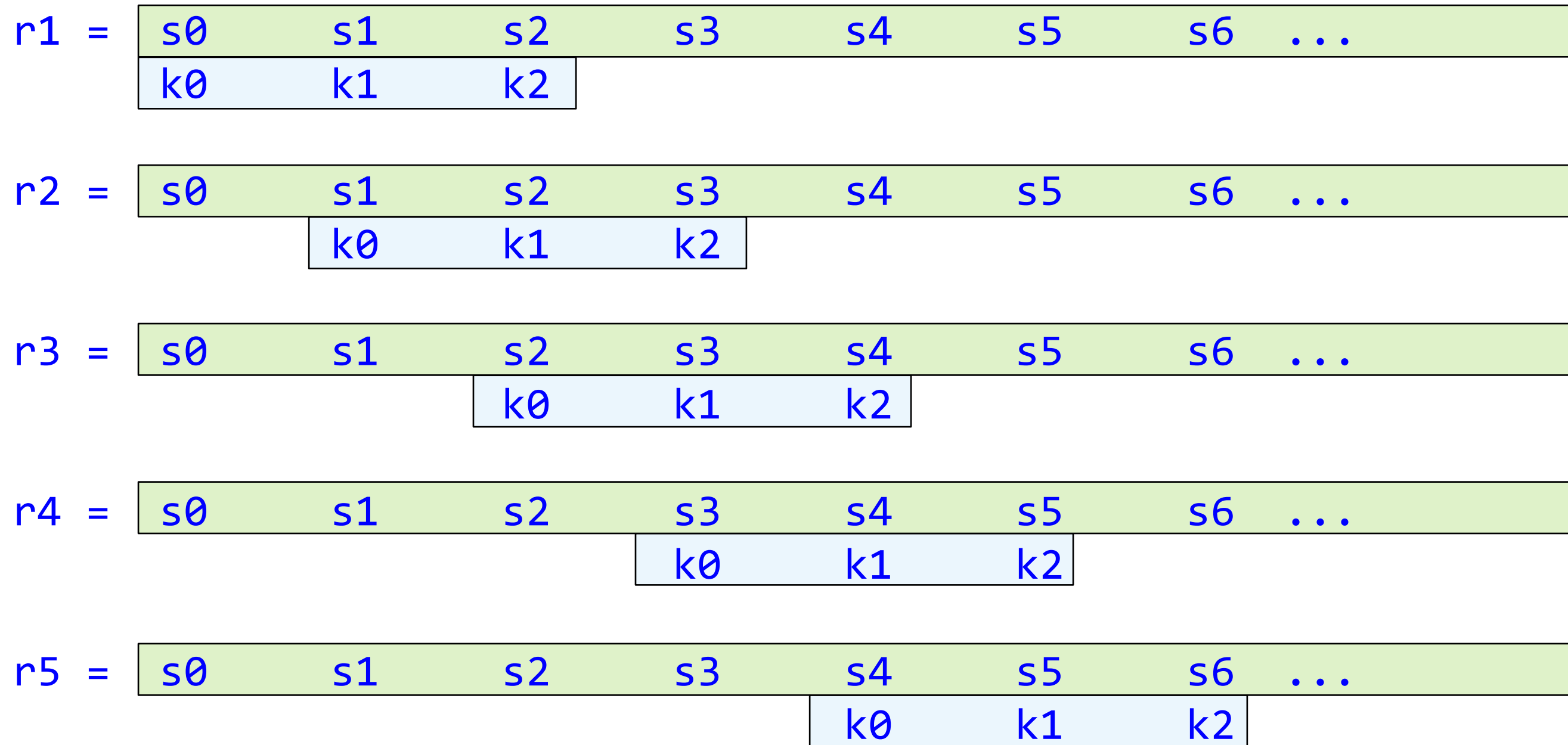
$$r_2 = \quad s_1k_0 + s_2k_1 + s_3k_2$$

$$r_3 = \quad \quad s_2k_0 + s_3k_1 + s_4k_2$$

$$r_4 = \quad \quad \quad s_3k_0 + s_4k_1 + s_5k_2$$

$$r_5 = \quad \quad \quad \quad s_4k_0 + s_5k_1 + s_6k_2$$

Convolution



Convolution

$S = s_0 \quad s_1 \quad s_2 \quad s_3 \quad s_4 \quad s_5 \quad s_6 \dots$
 $K = k_0 \quad k_1 \quad k_2$

$r_0 = \dots$

$r_1 = s_0k_0 + s_1k_1 + s_2k_2$

$r_2 = s_1k_0 + s_2k_1 + s_3k_2$

$r_3 = s_2k_0 + s_3k_1 + s_4k_2$

$r_4 = s_3k_0 + s_4k_1 + s_5k_2$

$r_5 = s_4k_0 + s_5k_1 + s_6k_2$

Convolution

$$\begin{array}{rcllclclcl} S & = & s_0 & & s_1 & & s_2 & & s_3 & & s_4 & & s_5 & & s_6 & \dots \\ K & = & k_0 & & k_1 & & k_2 & & & & & & & & & \end{array}$$

$$r_0 = \dots$$

$$\begin{array}{l} r_1 = s_0 k_0 + s_1 k_1 + s_2 k_2 \\ r_2 = s_1 k_0 + s_2 k_1 + s_3 k_2 \\ r_3 = s_2 k_0 + s_3 k_1 + s_4 k_2 \\ r_4 = s_3 k_0 + s_4 k_1 + s_5 k_2 \\ r_5 = s_4 k_0 + s_5 k_1 + s_6 k_2 \end{array}$$

s0	s1	s2	s3	s4	s5	s6	s7
k0	k0	k0	k0	k0	k0	k0	k0

s1	s2	s3	s4	s5	s6	s7	s8
k1	k1	k1	k1	k1	k1	k1	k1

s2	s3	s4	s5	s6	s7	s8	s9
k2	k2	k2	k2	k2	k2	k2	k2

r1	r2	r3	r4	r5	r6	r7	r8
----	----	----	----	----	----	----	----

Function Template avx_convolve()

```
template<int KernelSize, int KernelCenter> void
avx_convolve(float* pdst, float const* pkrnl, float const* psrc, size_t len)
{
    //- The convolution kernel must have non-negative size and fit with a single register.
    //
    static_assert(KernelSize > 1 && KernelSize <= 16);

    //- The index of the kernel center must be valid.
    //
    static_assert(KernelCenter >= 0 && KernelCenter < KernelSize);

    //- Convolution flips the kernel, so the kernel center must be adjusted.
    //
    constexpr int WindowCenter = KernelSize - KernelCenter - 1;

    rf_512 prev;    //- Bottom of the input data window
    rf_512 curr;    //- Middle of the input data windows
    rf_512 next;    //- Top of the input data window
    rf_512 lo;      //- Primary work data register, used to multiply kernel coefficients
    rf_512 hi;      //- Upper work data register, supplies values to the top of 'lo'
    rf_512 sum;     //- Accumulated value

    ...
}
```

Function Template avx_convolve()

```
template<int KernelSize, int KernelCenter> void
avx_convolve(float* pdst, float const* pkrnl, float const* psrc, size_t len)
{
    ...
    rf_512 kcoeff[KernelSize];    //- Coefficients of the convolution kernel

    //- Broadcast each kernel coefficient into its own register, to be used later in the FMA call.
    //
    for (int i = 0, j = KernelSize - 1; i < KernelSize; ++i, --j)
    {
        kcoeff[i] = load_value(pkrnl[j]);
    }

    //- Preload the initial input data window; note the zeroes in the register representing data
    // preceding the input array.
    //
    prev = load_value(0.0f);
    curr = load_from(psrc);
    next = load_from(psrc + 16);

    ...
}
```

Function Template avx_convolve()

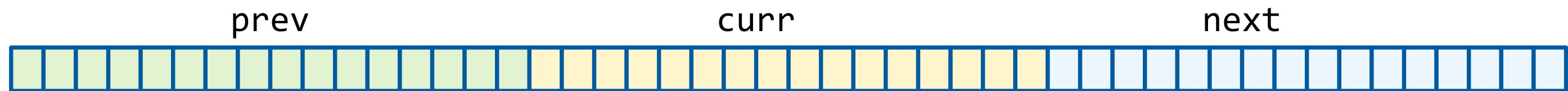
```
template<int KernelSize, int KernelCenter> void
avx_convolve(float* pdst, float const* pkrnl, float const* psrc, size_t len)
{
    ...
    rf_512 kcoeff[KernelSize];    //- Coefficients of the convolution kernel

    //- Broadcast each kernel coefficient into its own register, to be used later in the FMA call.
    //
    for (int i = 0, j = KernelSize - 1; i < KernelSize; ++i, --j)
    {
        kcoeff[i] = load_value(pkrnl[j]);
    }

    //- Preload the initial input data window; note the zeroes in the register representing data
    // preceding the input array.
    //
    prev = load_value(0.0f);
    curr = load_from(psrc);
    next = load_from(psrc + 16);

    ...
}
```

Function Template `avx_convolve()`



Function Template avx_convolve()

```
template<int KernelSize, int KernelCenter> void
avx_convolve(float* pdst, float const* pkrnl, float const* psrc, size_t len)
{
    ...
    for (auto pEnd = psrc + len - 16; psrc < pEnd; psrc += 16, pdst += 16)
    {
        sum = load_value(0.0f); // - Init the accumulator

        lo = shift_up_with_carry<WindowCenter>(prev, curr); // - Init the work data registers
        hi = shift_up_with_carry<WindowCenter>(curr, next);

        prev = curr; // - Slide the input data window up
        curr = next; // by a register's work of values
        next = load_from(psrc + 32);

        for (int k = 0; k < KernelSize; ++k)
        {
            sum = fused_multiply_add(kcoeff[k], lo, sum); // - Update the accumulator
            in_place_shift_down_with_carry<1>(lo, hi);
        }
        store_to(pdst, sum);
    }
}
```

Function Template avx_convolve()

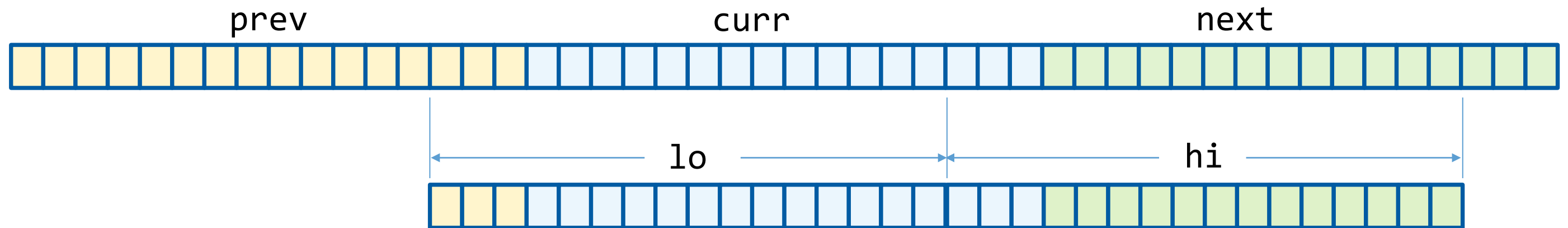
```
template<int KernelSize, int KernelCenter> void
avx_convolve(float* pdst, float const* pkrnl, float const* psrc, size_t len)
{
    ...
    for (auto pEnd = psrc + len - 16; psrc < pEnd; psrc += 16, pdst += 16)
    {
        sum = load_value(0.0f); // - Init the accumulator

        lo = shift_up_with_carry<WindowCenter>(prev, curr); // - Init the work data registers
        hi = shift_up_with_carry<WindowCenter>(curr, next);

        prev = curr; // - Slide the input data window up
        curr = next; // by a register's work of values
        next = load_from(psrc + 32);

        for (int k = 0; k < KernelSize; ++k)
        {
            sum = fused_multiply_add(kcoeff[k], lo, sum); // - Update the accumulator
            in_place_shift_down_with_carry<1>(lo, hi); // - Slide the input data down by 1
        }
        store_to(pdst, sum);
    }
}
```

Function Template `avx_convolve()`



Function Template avx_convolve()

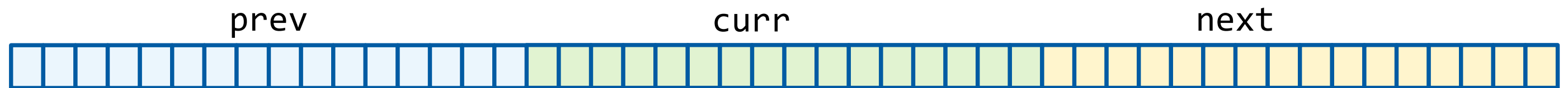
```
template<int KernelSize, int KernelCenter> void
avx_convolve(float* pdst, float const* pkrnl, float const* psrc, size_t len)
{
    ...
    for (auto pEnd = psrc + len - 16; psrc < pEnd; psrc += 16, pdst += 16)
    {
        sum = load_value(0.0f); // - Init the accumulator

        lo = shift_up_with_carry<WindowCenter>(prev, curr); // - Init the work data registers
        hi = shift_up_with_carry<WindowCenter>(curr, next);

        prev = curr; // - Slide the input data window up
        curr = next; // by a register's work of values
        next = load_from(psrc + 32);

        for (int k = 0; k < KernelSize; ++k)
        {
            sum = fused_multiply_add(kcoeff[k], lo, sum); // - Update the accumulator
            in_place_shift_down_with_carry<1>(lo, hi);
        }
        store_to(pdst, sum);
    }
}
```

Function Template `avx_convolve()`



Function Template avx_convolve()

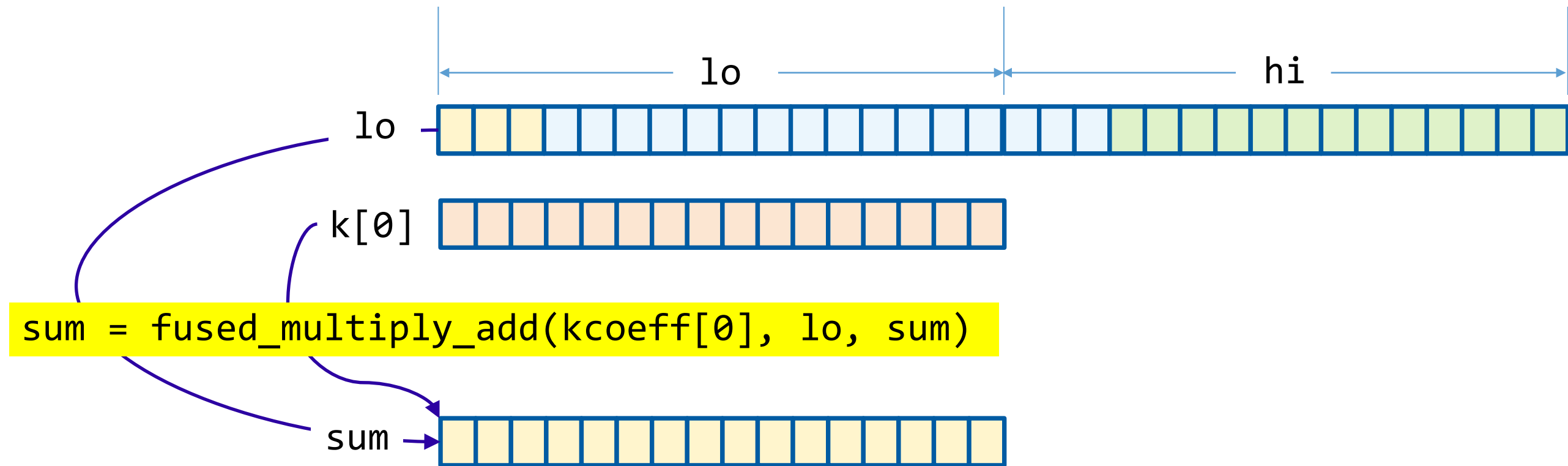
```
template<int KernelSize, int KernelCenter> void
avx_convolve(float* pdst, float const* pkrnl, float const* psrc, size_t len)
{
    ...
    for (auto pEnd = psrc + len - 16; psrc < pEnd; psrc += 16, pdst += 16)
    {
        sum = load_value(0.0f); // - Init the accumulator

        lo = shift_up_with_carry<WindowCenter>(prev, curr); // - Init the work data registers
        hi = shift_up_with_carry<WindowCenter>(curr, next);

        prev = curr; // - Slide the input data window up
        curr = next; // by a register's work of values
        next = load_from(psrc + 32);

        for (int k = 0; k < KernelSize; ++k)
        {
            sum = fused_multiply_add(kcoeff[k], lo, sum); // - Update the accumulator
            in_place_shift_down_with_carry<1>(lo, hi);
        }
        store_to(pdst, sum);
    }
}
```

Function Template avx_convolve()



Function Template avx_convolve()

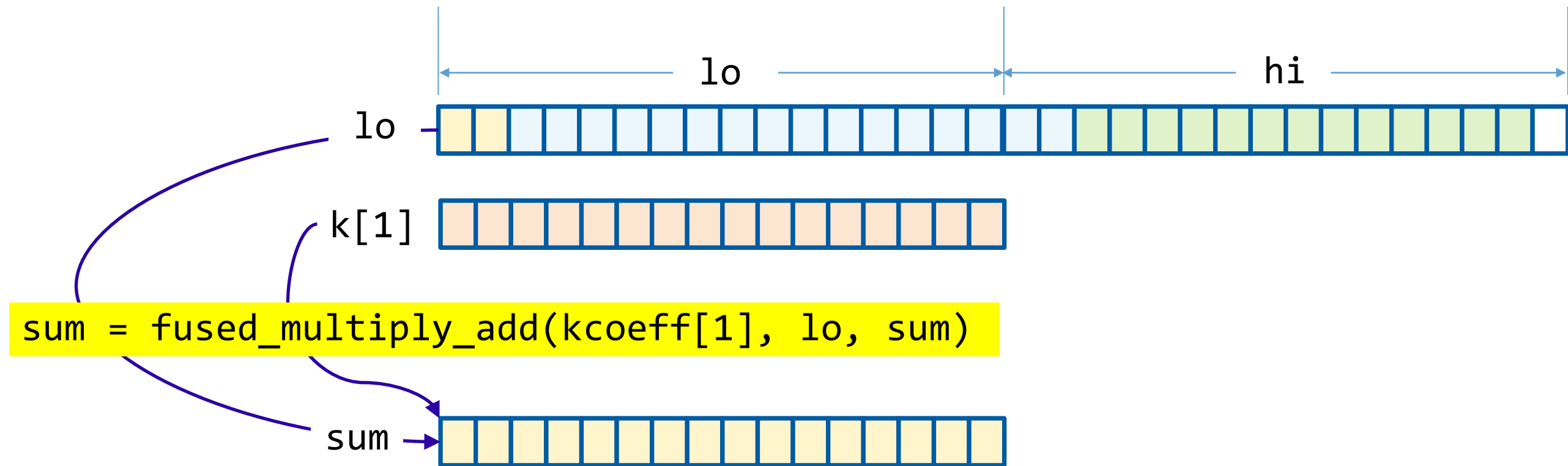
```
template<int KernelSize, int KernelCenter> void
avx_convolve(float* pdst, float const* pkrnl, float const* psrc, size_t len)
{
    ...
    for (auto pEnd = psrc + len - 16; psrc < pEnd; psrc += 16, pdst += 16)
    {
        sum = load_value(0.0f); // - Init the accumulator

        lo = shift_up_with_carry<WindowCenter>(prev, curr); // - Init the work data registers
        hi = shift_up_with_carry<WindowCenter>(curr, next);

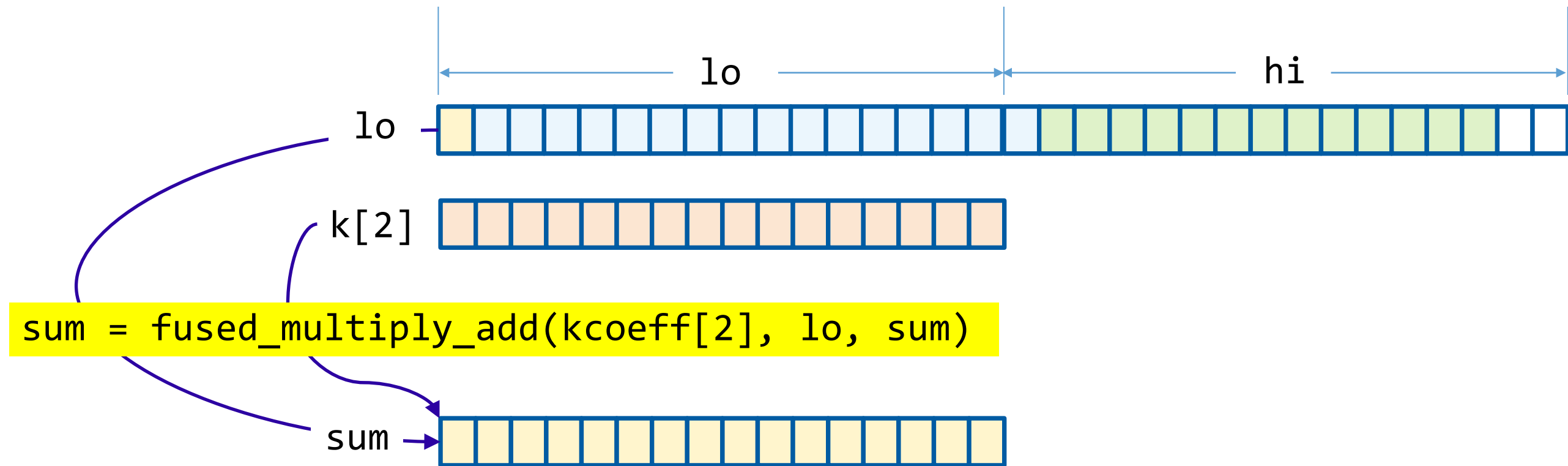
        prev = curr; // - Slide the input data window up
        curr = next; // by a register's work of values
        next = load_from(psrc + 32);

        for (int k = 0; k < KernelSize; ++k)
        {
            sum = fused_multiply_add(kcoeff[k], lo, sum); // - Update the accumulator
            in_place_shift_down_with_carry<1>(lo, hi);
        }
        store_to(pdst, sum);
    }
}
```

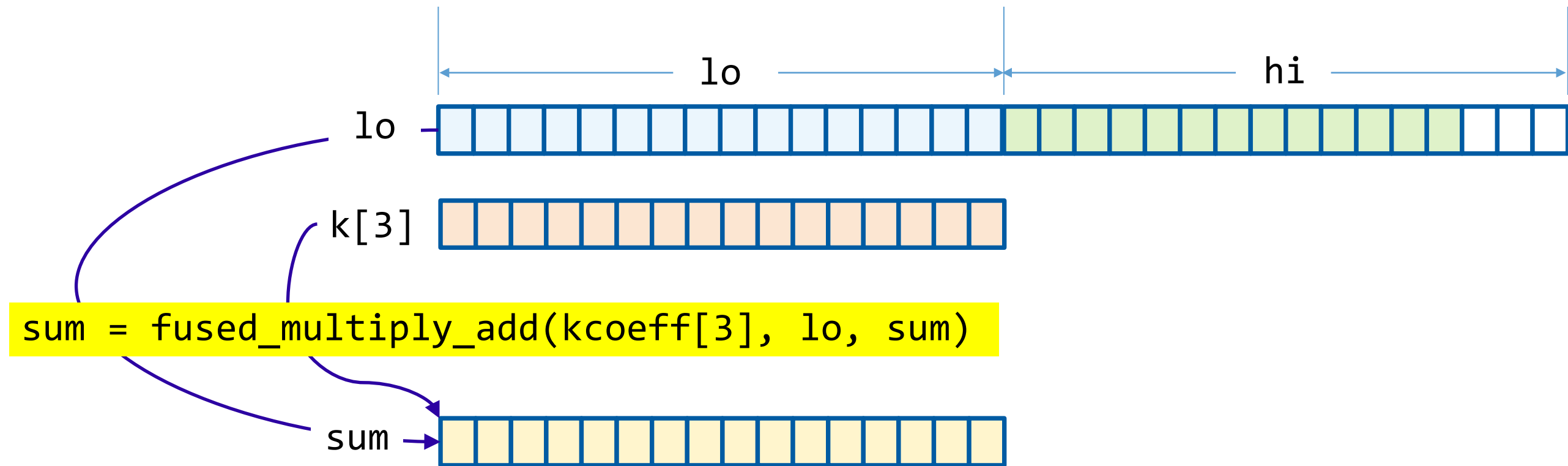
Function Template avx_convolve()



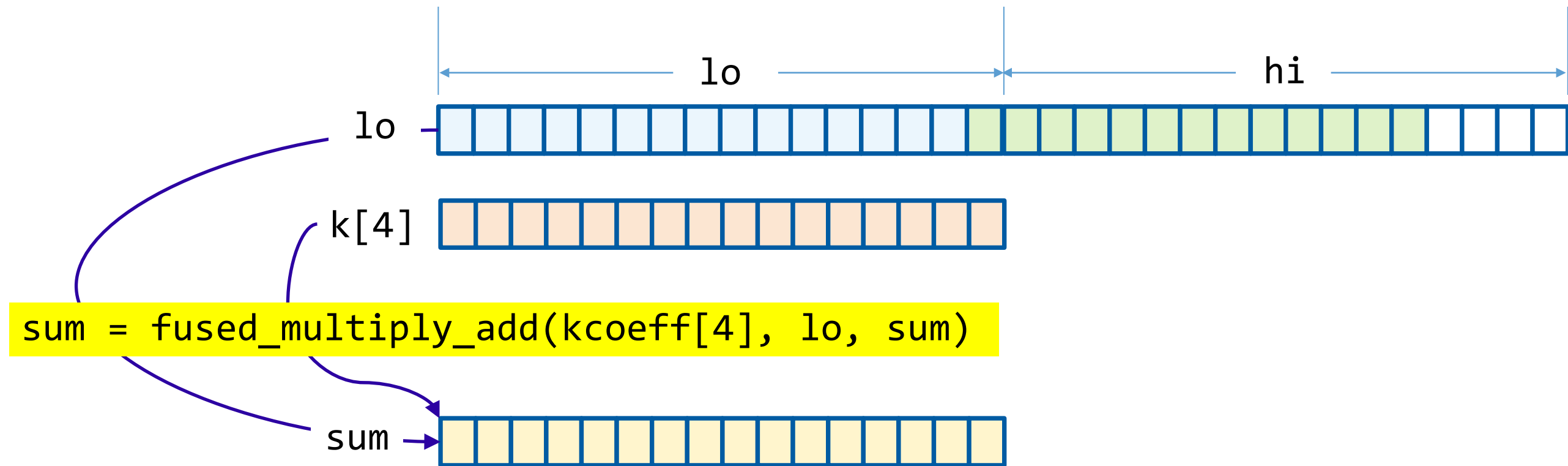
Function Template avx_convolve()



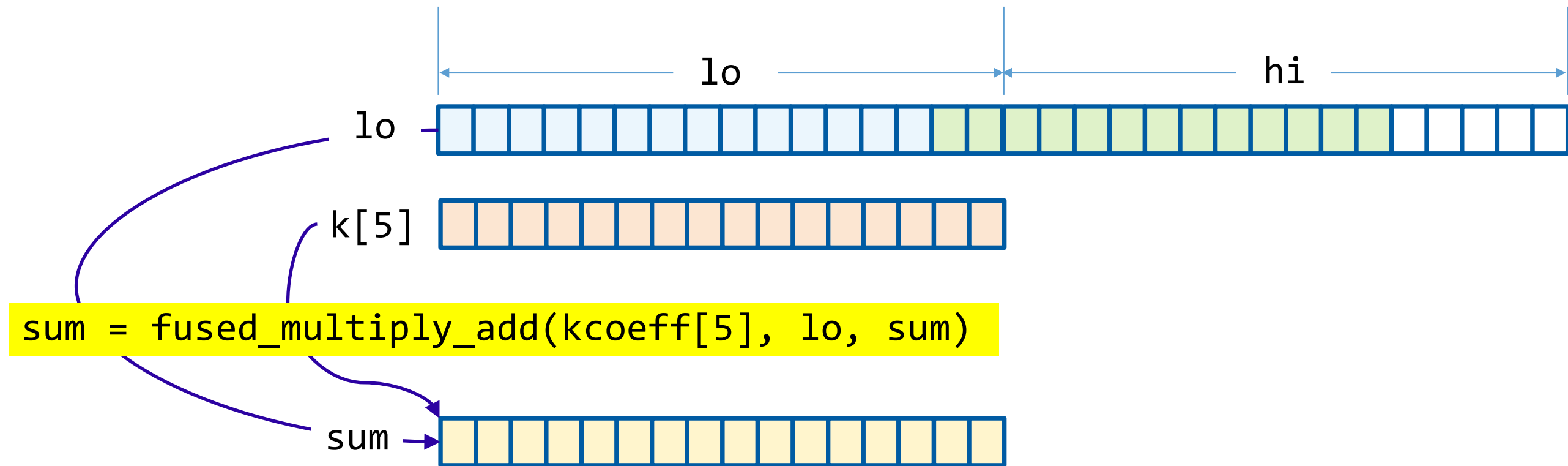
Function Template avx_convolve()



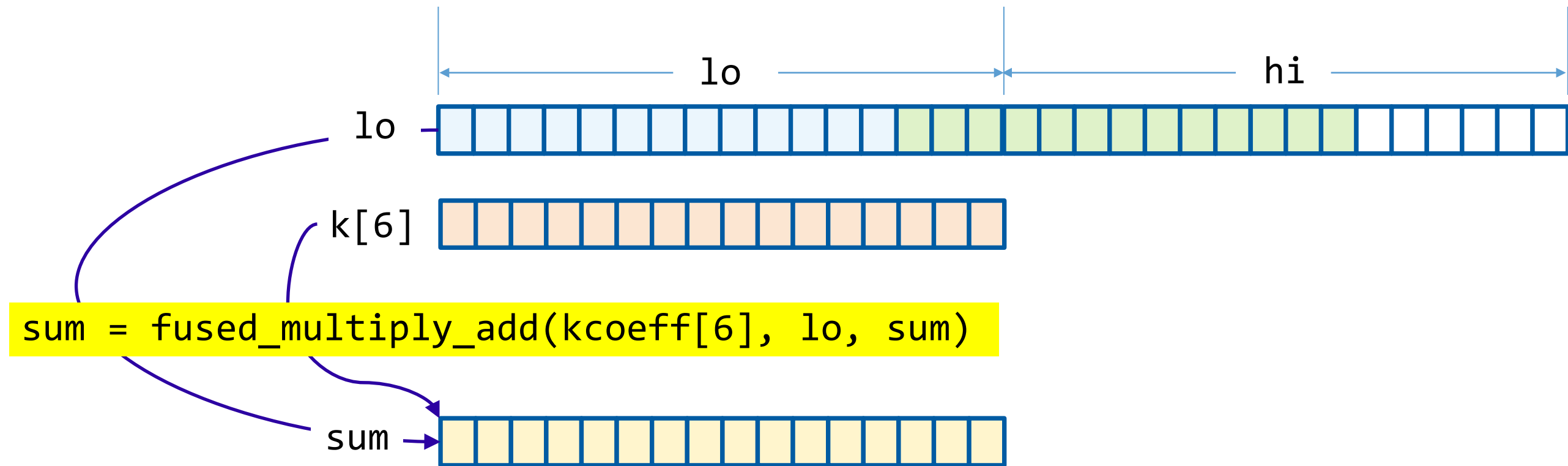
Function Template avx_convolve()



Function Template avx_convolve()



Function Template avx_convolve()



Function Template avx_convolve()

```
template<int KernelSize, int KernelCenter> void
avx_convolve(float* pdst, float const* pkrnl, float const* psrc, size_t len)
{
    ...
    for (auto pEnd = psrc + len - 16; psrc < pEnd; psrc += 16, pdst += 16)
    {
        sum = load_value(0.0f); // - Init the accumulator

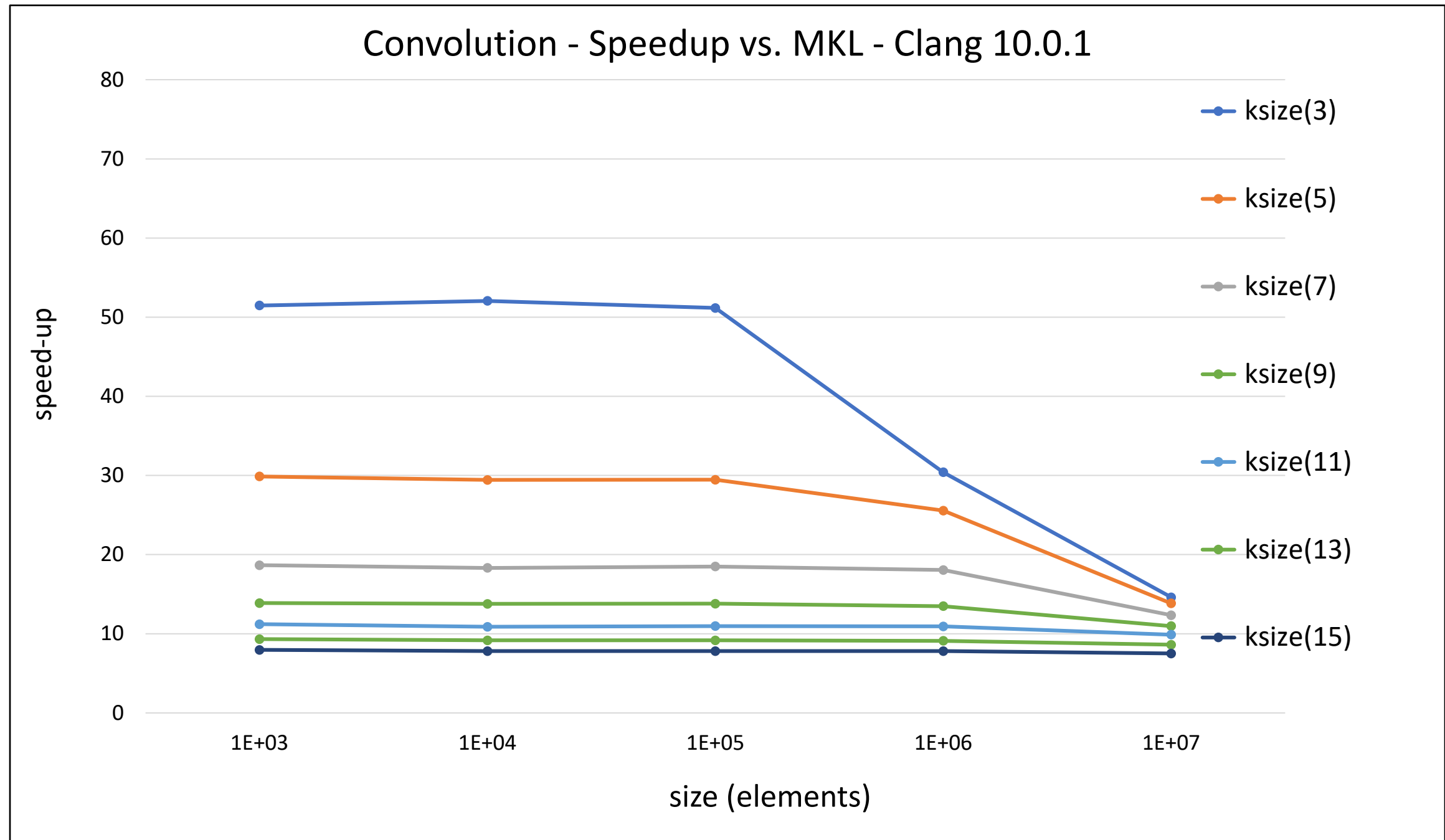
        lo = shift_up_with_carry<WindowCenter>(prev, curr); // - Init the work data registers
        hi = shift_up_with_carry<WindowCenter>(curr, next);

        prev = curr; // - Slide the input data window up
        curr = next; // by a register's work of values
        next = load_from(psrc + 32);

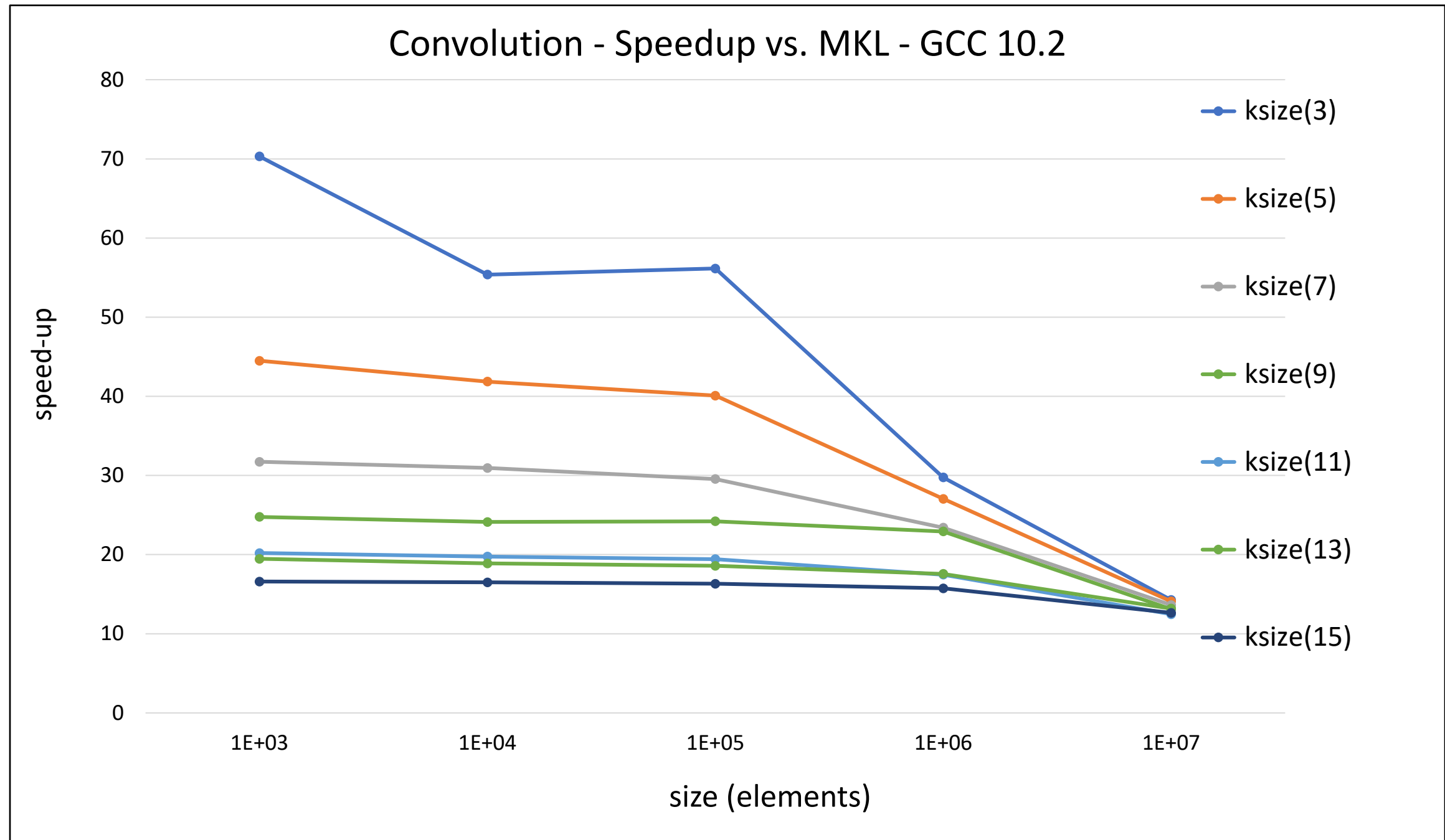
        for (int k = 0; k < KernelSize; ++k)
        {
            sum = fused_multiply_add(kcoeff[k], lo, sum); // - Update the accumulator
            in_place_shift_down_with_carry<1>(lo, hi);
        }
        store_to(pdst, sum);
    }
}
```

- Ubuntu 18.04 on Cascade Lake
 - **GCC 10.2**, all code compiled with `-O3 -mavx512 -march=skylake`
 - **Clang 10.0.1**, all code compiled with `-O3 -mavx512 -march=skylake`
- Intel MKL 2020.1.217
- Odd-number kernel sizes 3 – 15
- Element counts of 1E03 through 1E07 (by 10s)
- Collect timings for each combination using MKL and Small-Kernel AVX

Results – Convolution – Clang 10.0.1

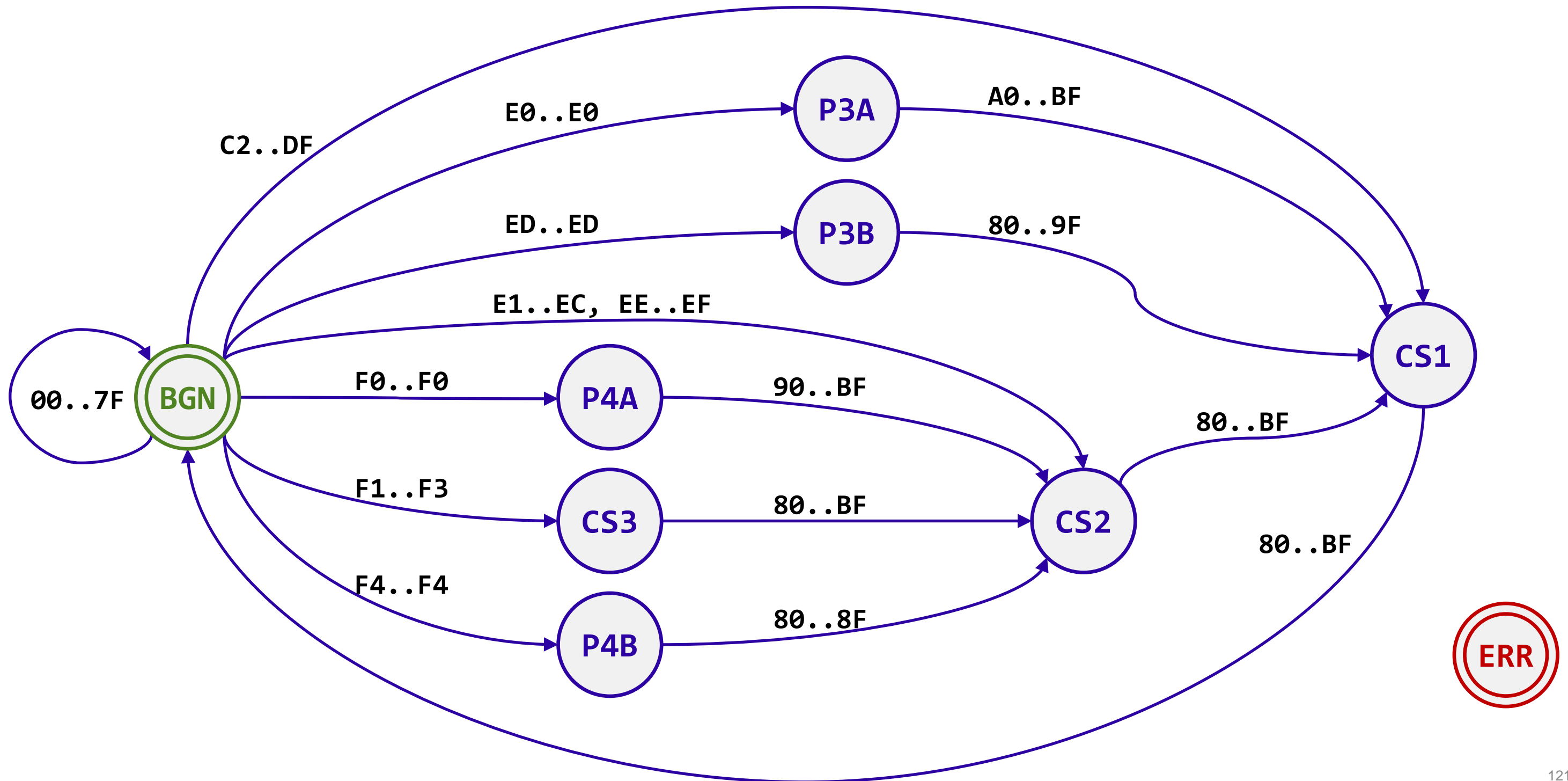


Results – Convolution – GCC 10.2



UTF-8 to UTF-32 Conversion Algorithm

The UTF-8 Decoding DFA



The Basic Conversion Algorithm (UTF-8 to UTF-32)

```
KEWB_ALIGN_FN std::ptrdiff_t
UtfUtils::BasicConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept
{
    char32_t*    pDstOrig = pDst;
    char32_t     cdpt;

    while (pSrc < pSrcEnd)
    {
        if (Advance(pSrc, pSrcEnd, cdpt) != ERR)
        {
            *pDst++ = cdpt;
        }
        else
        {
            return -1;
        }
    }
    return pDst - pDstOrig;
}
```

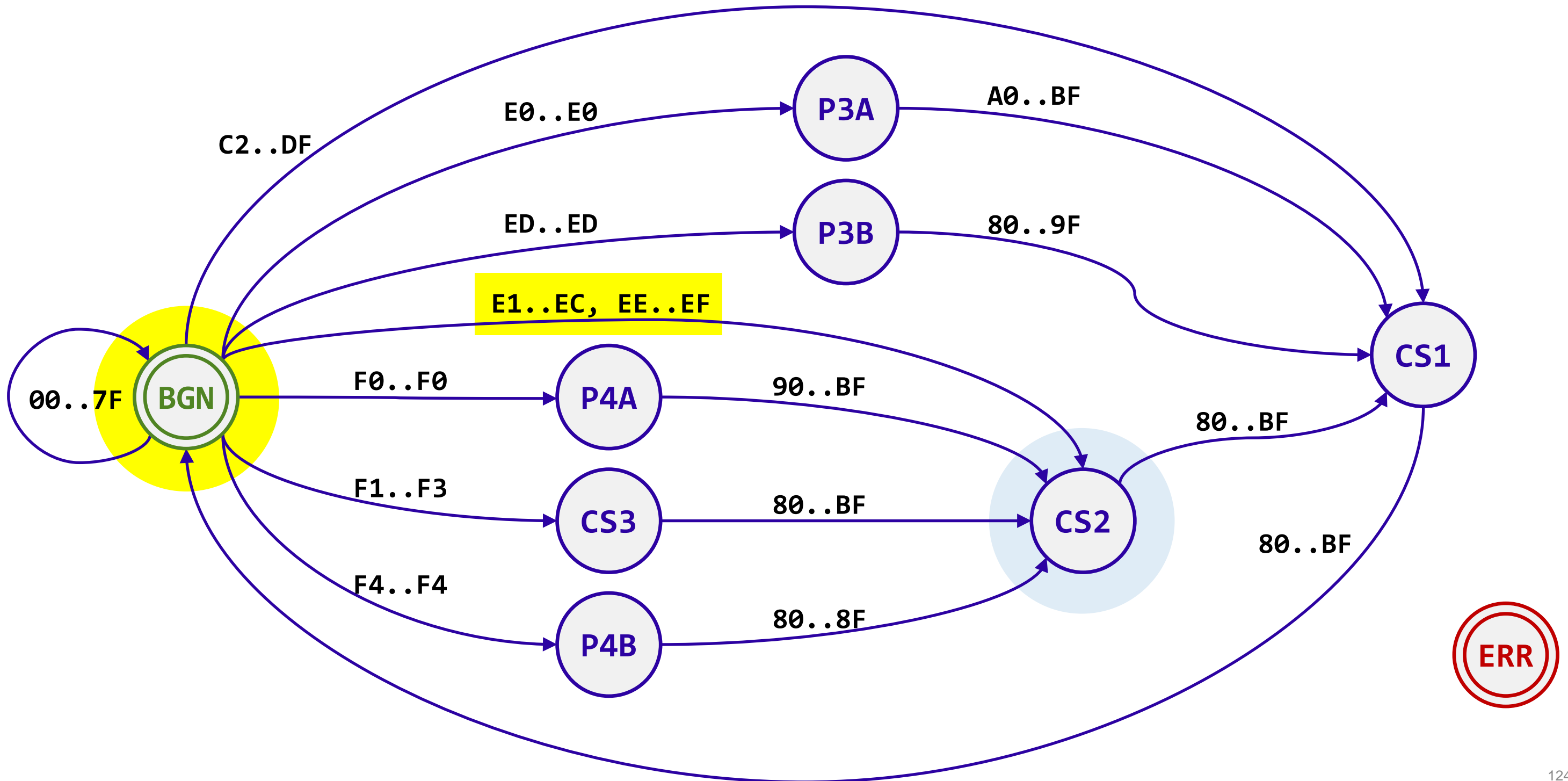
Converting a Single Code Point - Overview

```
KEWB_FORCE_INLINE int32_t
UtfUtils::Advance(char8_t const*& pSrc, char8_t const* pSrcEnd, char32_t& cdpt) noexcept
{
    FirstUnitInfo    info;    //- The descriptor for the first code unit
    char32_t          unit;    //- The current UTF-8 code unit
    int32_t           type;    //- The code unit's character class
    int32_t           curr;    //- The current DFA state

    info = smTables.maFirstUnitTable[*pSrc++];    //- Look up the first descriptor
    cdpt = info.mFirstOctet;    //- Get the initial code point value
    curr = info.mNextState;    //- Advance to the next state

    while (curr > ERR)    //- Loop over subsequent units
    {
        if (pSrc < pSrcEnd)
        {
            unit = *pSrc++;    //- Cache the current code unit
            cdpt = (cdpt << 6) | (unit & 0x3F);    //- Adjust code point with continuation bits
            type = smTables.maOctetCategory[unit];    //- Look up the code unit's character class
            curr = smTables.maTransitions[curr + type];    //- Advance to the next state
        }
        else
        {
            return ERR;
        }
    }
    return curr;
}
```

A Decoding Example – { .. **E2** 88 85 .. }



The Basic Conversion Algorithm (UTF-8 to UTF-32)

```
KEWB_ALIGN_FN std::ptrdiff_t
UtfUtils::BasicConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept
{
    char32_t*    pDstOrig = pDst;
    char32_t     cdpt;

    while (pSrc < pSrcEnd)
    {
        if (Advance(pSrc, pSrcEnd, cdpt) != ERR)
        {
            *pDst++ = cdpt;
        }
        else
        {
            return -1;
        }
    }
    return pDst - pDstOrig;
}
```

Converting a Single Code Point

```
KEWB_FORCE_INLINE int32_t
UtfUtils::Advance(char8_t const*& pSrc, char8_t const* pSrcEnd, char32_t& cdpt) noexcept
{
    FirstUnitInfo    info;    //- The descriptor for the first code unit
    char32_t          unit;    //- The current UTF-8 code unit
    int32_t           type;    //- The code unit's character class
    int32_t           curr;    //- The current DFA state

    info = smTables.maFirstUnitTable[*pSrc++];    //- Look up the first code unit descriptor
    cdpt = info.mFirstOctet;                      //- Get the initial code point value
    curr = info.mNextState;                       //- Advance to the next state

    while (curr > ERR)                            //- Loop over subsequent code units
    {
        ...
    }
    return curr;
}
```

Converting a Single Code Point

```
KEWB_FORCE_INLINE int32_t
UtfUtils::Advance(char8_t const*& pSrc, char8_t const* pSrcEnd, char32_t& cdpt) noexcept
{
    FirstUnitInfo    info;    //- The descriptor for the first code unit
    char32_t          unit;    //- The current UTF-8 code unit
    int32_t           type;    //- The code unit's character class
    int32_t           curr;    //- The current DFA state

    info = smTables.maFirstUnitTable[*pSrc++];    //- Look up the first code unit descriptor
    cdpt = info.mFirstOctet;                    //- Get the initial code point value
    curr = info.mNextState;                      //- Advance to the next state

    while (curr > ERR)                            //- Loop over subsequent code units
    {
        ...
    }
    return curr;
}
```

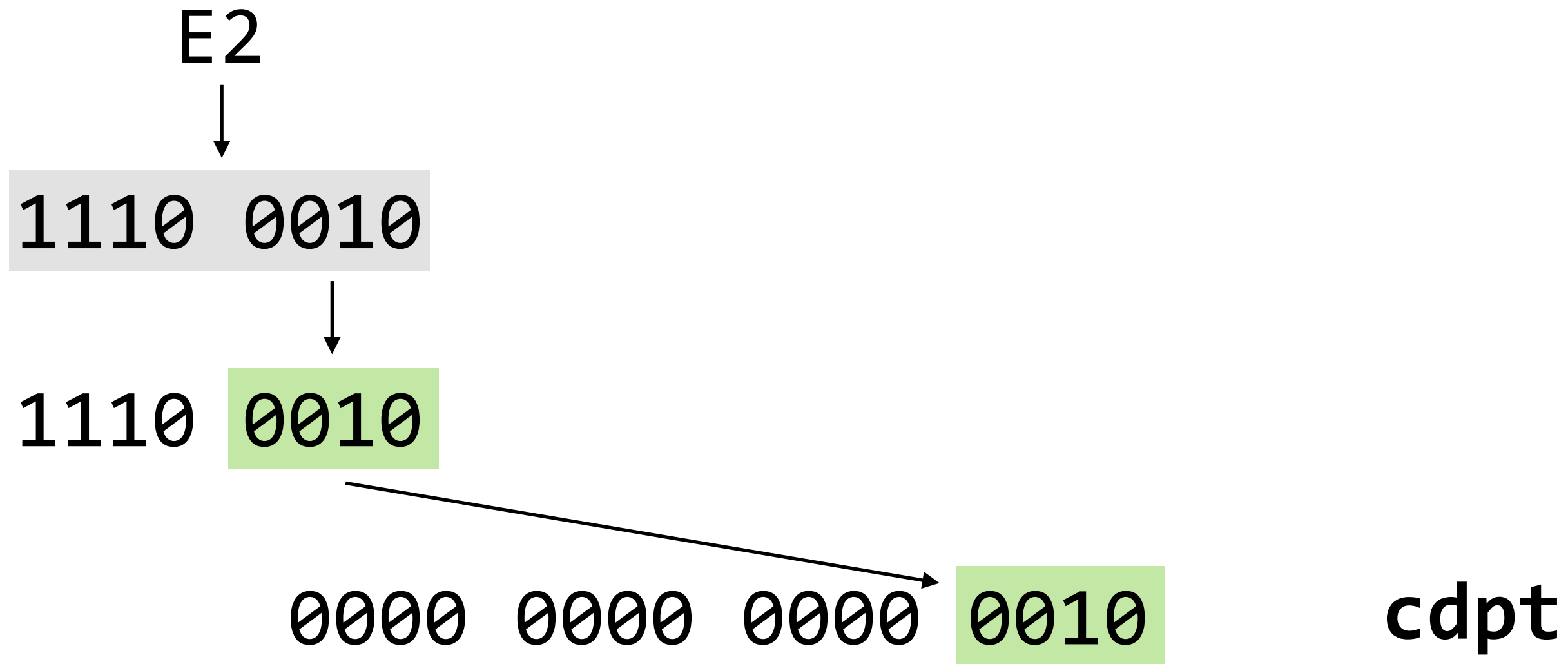
Converting a Single Code Point

```
KEWB_FORCE_INLINE int32_t
UtfUtils::Advance(char8_t const*& pSrc, char8_t const* pSrcEnd, char32_t& cdpt) noexcept
{
    FirstUnitInfo    info;    //- The descriptor for the first code unit
    char32_t          unit;    //- The current UTF-8 code unit
    int32_t           type;    //- The code unit's character class
    int32_t           curr;    //- The current DFA state

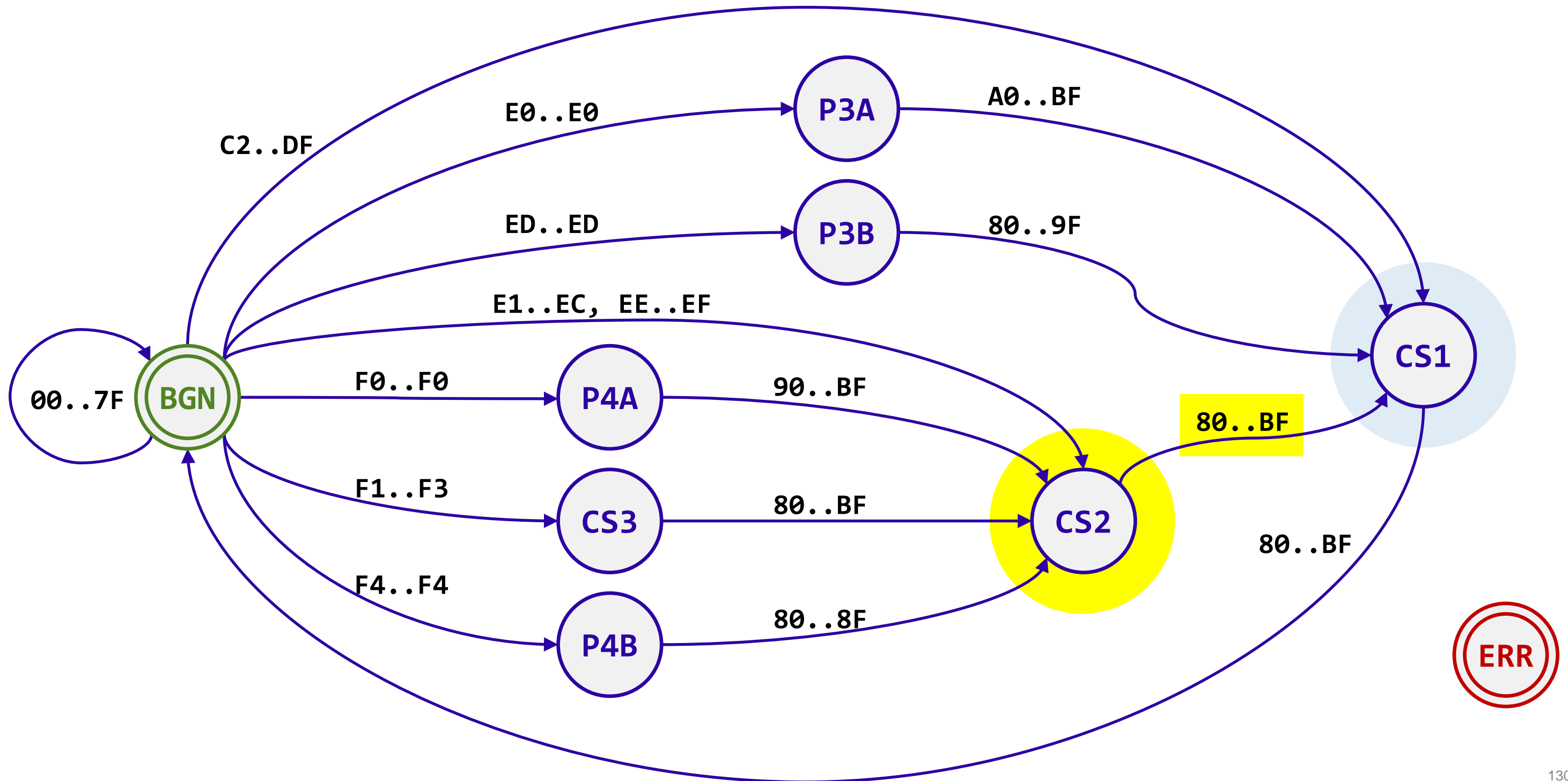
    info = smTables.maFirstUnitTable[*pSrc++];    //- Look up the first code unit descriptor
    cdpt = info.mFirstOctet;                      //- Get the initial code point value
    curr = info.mNextState;                      //- Advance to the next state

    while (curr > ERR)                            //- Loop over subsequent code units
    {
        ...
    }
    return curr;
}
```


A Decoding Example – { .. E2 88 85 .. }



A Decoding Example – { .. E2 88 85 .. }



Converting a Single Code Point

```
KEWB_FORCE_INLINE int32_t
UtfUtils::Advance(char8_t const*& pSrc, char8_t const* pSrcEnd, char32_t& cdpt) noexcept
{
    ...

    while (curr > ERR)                                //- Loop over subsequent code units
    {
        if (pSrc < pSrcEnd)
        {
            unit = *pSrc++;                            //- Cache the current code unit
            cdpt = (cdpt << 6) | (unit & 0x3F);         //- Adjust code point with new bits
            type = smTables.maOctetCategory[unit];      //- Look up the code unit's char class
            curr = smTables.maTransitions[curr + type]; //- Advance to the next state
        }
        else
        {
            return ERR;
        }
    }
    return curr;
}
```

Converting a Single Code Point

```
KEWB_FORCE_INLINE int32_t
UtfUtils::Advance(char8_t const*& pSrc, char8_t const* pSrcEnd, char32_t& cdpt) noexcept
{
    ...

    while (curr > ERR)                                     //- Loop over subsequent code units
    {
        if (pSrc < pSrcEnd)
        {
            unit = *pSrc++;                                //- Cache the current code unit
            cdpt = (cdpt << 6) | (unit & 0x3F);             //- Adjust code point with new bits
            type = smTables.maOctetCategory[unit];          //- Look up the code unit's char class
            curr = smTables.maTransitions[curr + type];     //- Advance to the next state
        }
        else
        {
            return ERR;
        }
    }
    return curr;
}
```

Converting a Single Code Point

```
KEWB_FORCE_INLINE int32_t
UtfUtils::Advance(char8_t const*& pSrc, char8_t const* pSrcEnd, char32_t& cdpt) noexcept
{
    ...

    while (curr > ERR)                                     //- Loop over subsequent code units
    {
        if (pSrc < pSrcEnd)
        {
            unit = *pSrc++;                                //- Cache the current code unit
            cdpt = (cdpt << 6) | (unit & 0x3F);             //- Adjust code point with new bits
            type = smTables.maOctetCategory[unit];          //- Look up the code unit's char class
            curr = smTables.maTransitions[curr + type];     //- Advance to the next state
        }
        else
        {
            return ERR;
        }
    }
    return curr;
}
```

Converting a Single Code Point

```
KEWB_FORCE_INLINE int32_t
UtfUtils::Advance(char8_t const*& pSrc, char8_t const* pSrcEnd, char32_t& cdpt) noexcept
{
    ...

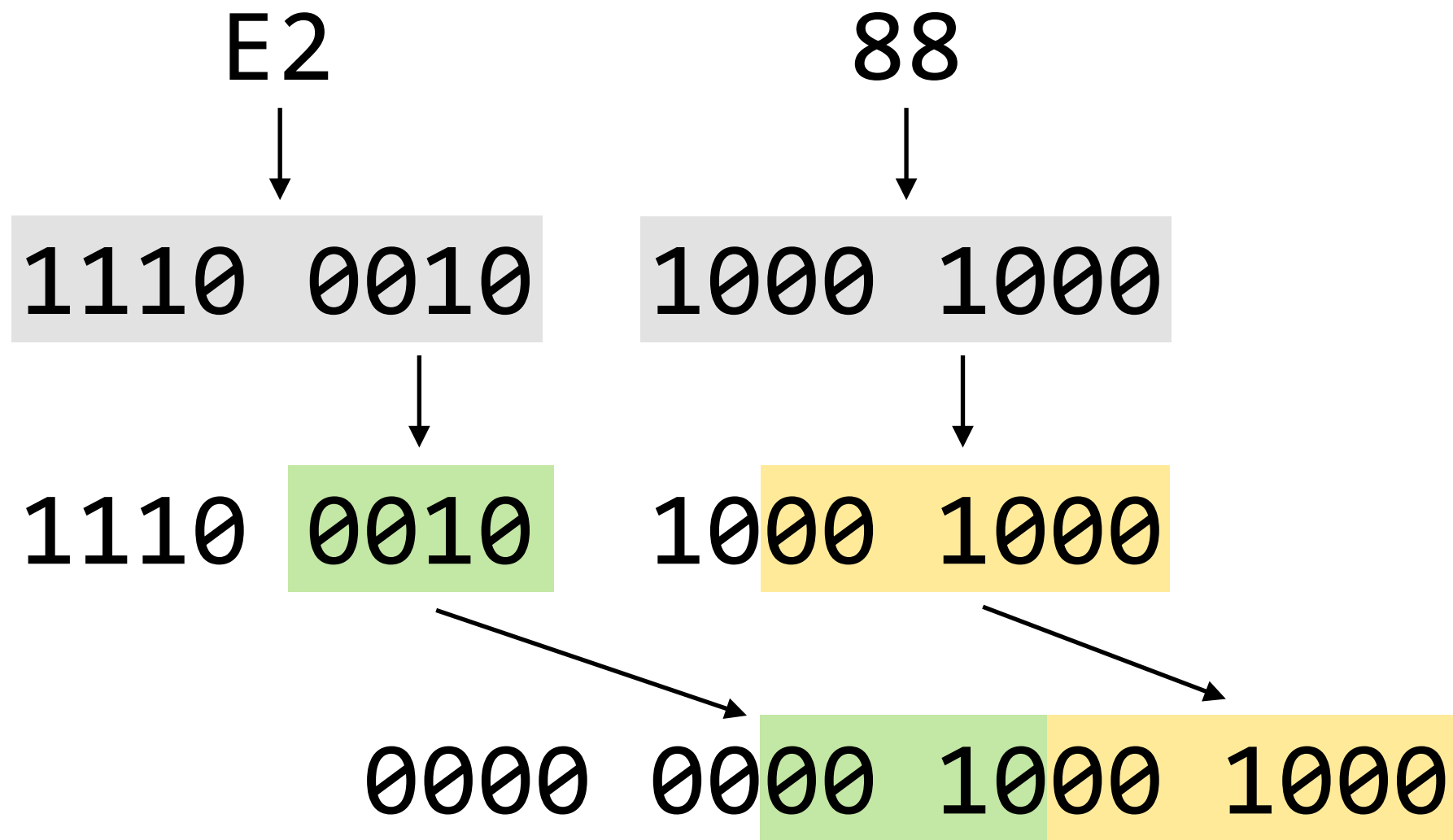
    while (curr > ERR)                                     //- Loop over subsequent code units
    {
        if (pSrc < pSrcEnd)
        {
            unit = *pSrc++;                                //- Cache the current code unit
            cdpt = (cdpt << 6) | (unit & 0x3F);            //- Adjust code point with new bits
            type = smTables.maOctetCategory[unit];         //- Look up the code unit's char class
            curr = smTables.maTransitions[curr + type];    //- Advance to the next state
        }
        else
        {
            return ERR;
        }
    }
    return curr;
}
```

Converting a Single Code Point

```
KEWB_FORCE_INLINE int32_t
UtfUtils::Advance(char8_t const*& pSrc, char8_t const* pSrcEnd, char32_t& cdpt) noexcept
{
    ...

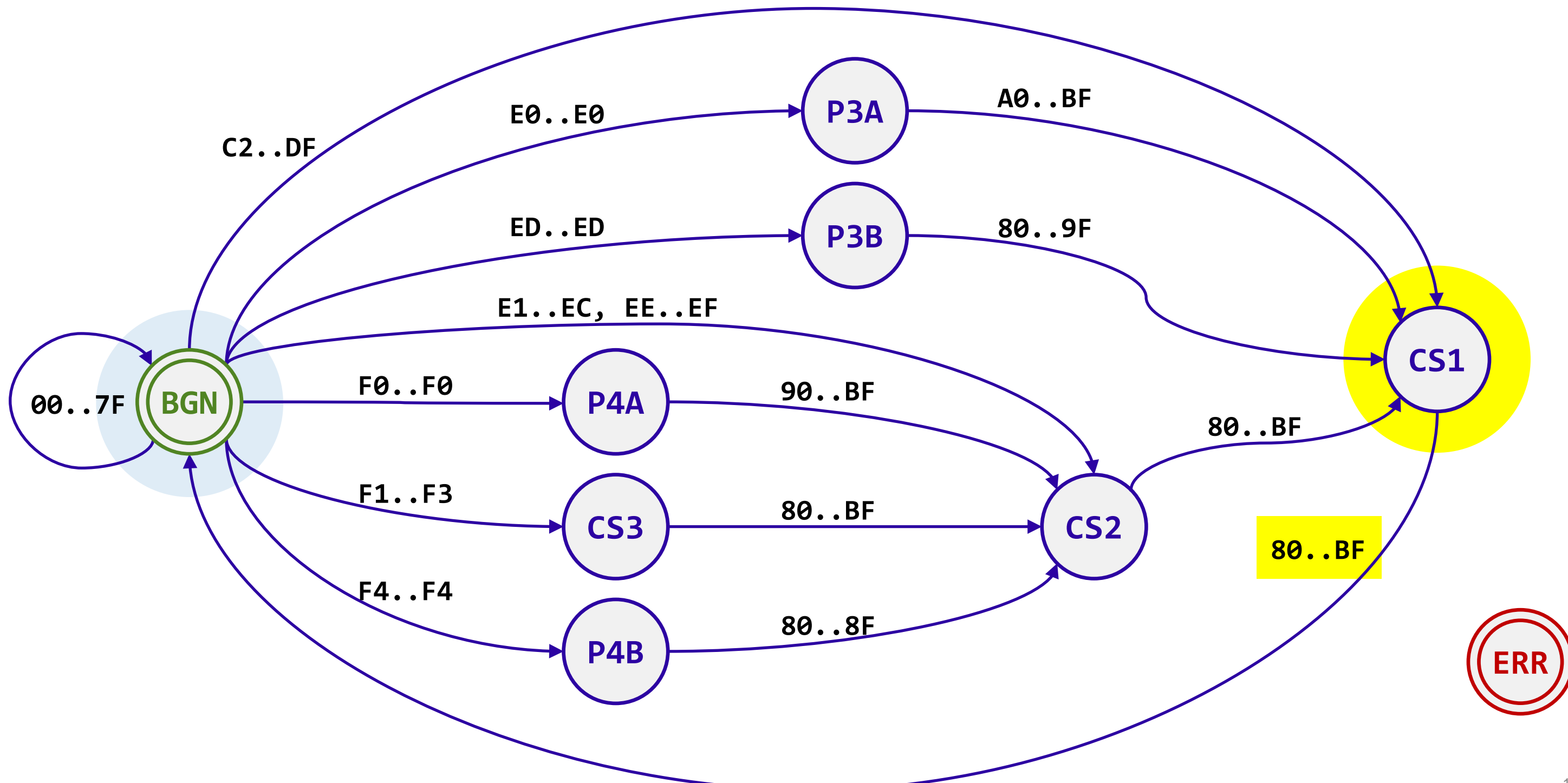
    while (curr > ERR)                                     //- Loop over subsequent code units
    {
        if (pSrc < pSrcEnd)
        {
            unit = *pSrc++;                                //- Cache the current code unit
            cdpt = (cdpt << 6) | (unit & 0x3F);             //- Adjust code point with new bits
            type = smTables.maOctetCategory[unit];          //- Look up the code unit's char class
            curr = smTables.maTransitions[curr + type];     //- Advance to the next state
        }
        else
        {
            return ERR;
        }
    }
    return curr;
}
```

A Decoding Example – { .. E2 88 85 .. }



cdpt

A Decoding Example – { .. E2 88 85 .. }



Converting a Single Code Point

```
KEWB_FORCE_INLINE int32_t
UtfUtils::Advance(char8_t const*& pSrc, char8_t const* pSrcEnd, char32_t& cdpt) noexcept
{
    ...

    while (curr > ERR)                                     //- Loop over subsequent code units
    {
        if (pSrc < pSrcEnd)
        {
            unit = *pSrc++;                                //- Cache the current code unit
            cdpt = (cdpt << 6) | (unit & 0x3F);             //- Adjust code point with new bits
            type = smTables.maOctetCategory[unit];          //- Look up the code unit's char class
            curr = smTables.maTransitions[curr + type];     //- Advance to the next state
        }
        else
        {
            return ERR;
        }
    }
    return curr;
}
```

Converting a Single Code Point

```
KEWB_FORCE_INLINE int32_t
UtfUtils::Advance(char8_t const*& pSrc, char8_t const* pSrcEnd, char32_t& cdpt) noexcept
{
    ...

    while (curr > ERR)                                     //- Loop over subsequent code units
    {
        if (pSrc < pSrcEnd)
        {
            unit = *pSrc++;                                //- Cache the current code unit
            cdpt = (cdpt << 6) | (unit & 0x3F);             //- Adjust code point with new bits
            type = smTables.maOctetCategory[unit];          //- Look up the code unit's char class
            curr = smTables.maTransitions[curr + type];     //- Advance to the next state
        }
        else
        {
            return ERR;
        }
    }
    return curr;
}
```

Converting a Single Code Point

```
KEWB_FORCE_INLINE int32_t
UtfUtils::Advance(char8_t const*& pSrc, char8_t const* pSrcEnd, char32_t& cdpt) noexcept
{
    ...

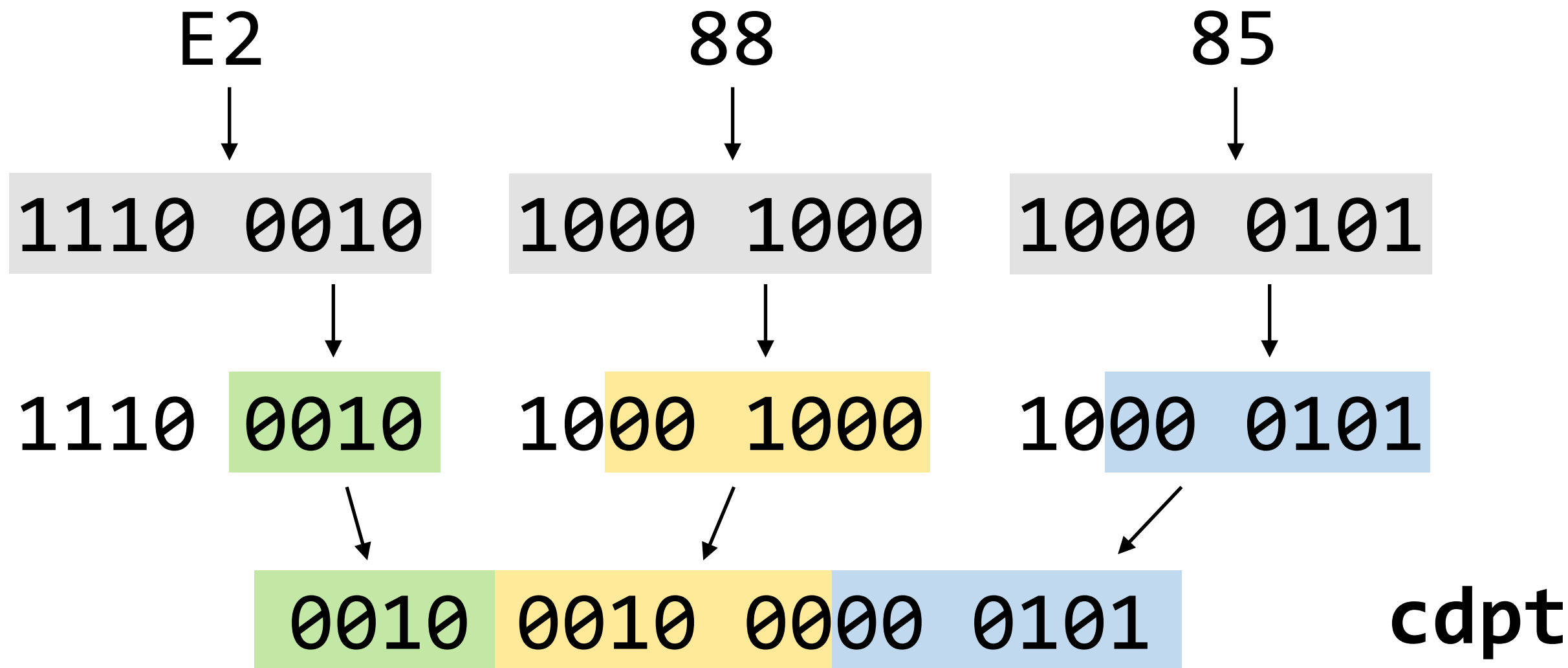
    while (curr > ERR)                                     //- Loop over subsequent code units
    {
        if (pSrc < pSrcEnd)
        {
            unit = *pSrc++;                                //- Cache the current code unit
            cdpt = (cdpt << 6) | (unit & 0x3F);             //- Adjust code point with new bits
            type = smTables.maOctetCategory[unit];          //- Look up the code unit's char class
            curr = smTables.maTransitions[curr + type];     //- Advance to the next state
        }
        else
        {
            return ERR;
        }
    }
    return curr;
}
```

Converting a Single Code Point

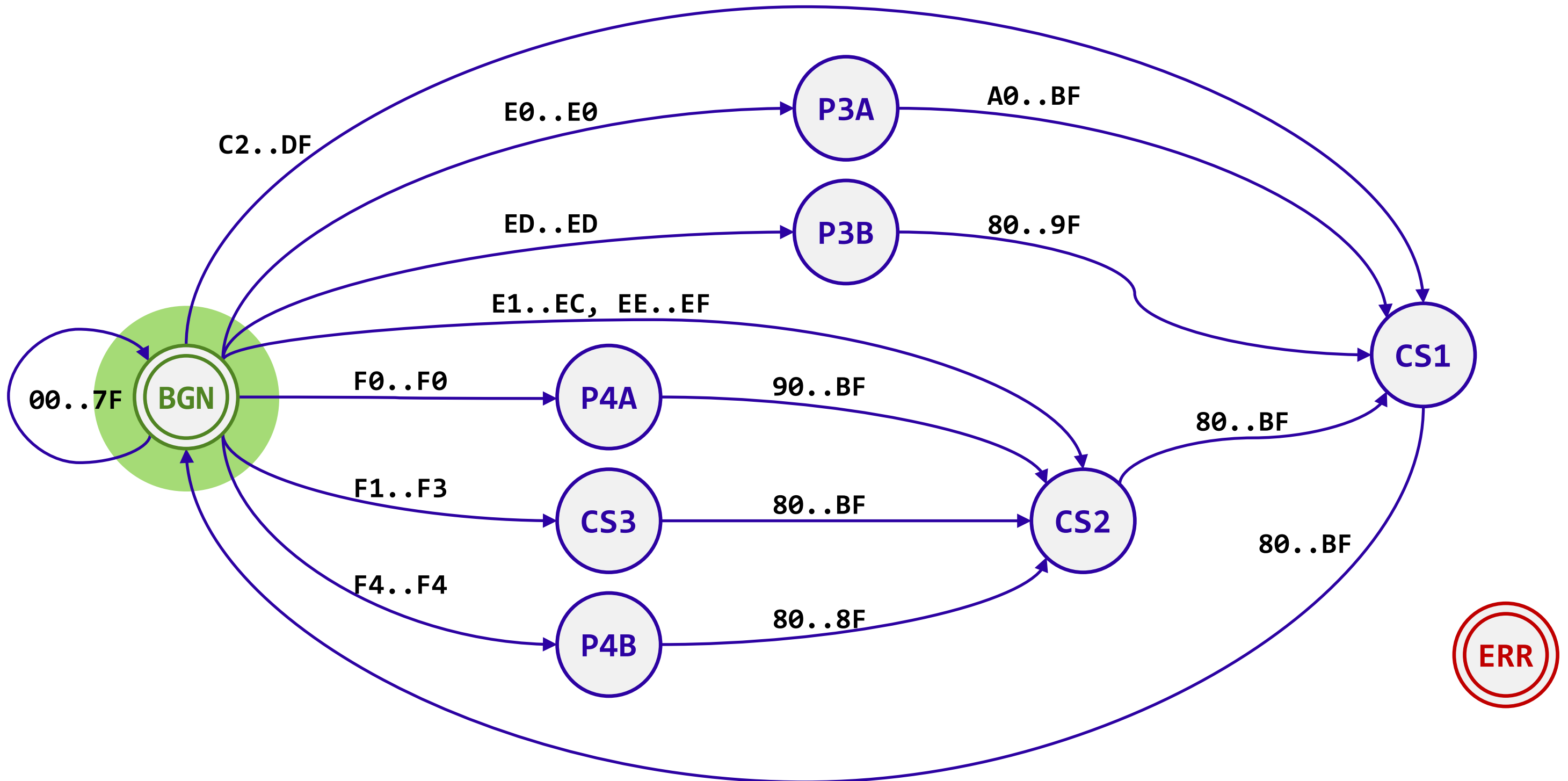
```
KEWB_FORCE_INLINE int32_t
UtfUtils::Advance(char8_t const*& pSrc, char8_t const* pSrcEnd, char32_t& cdpt) noexcept
{
    ...

    while (curr > ERR)                                     //- Loop over subsequent code units
    {
        if (pSrc < pSrcEnd)
        {
            unit = *pSrc++;                                //- Cache the current code unit
            cdpt = (cdpt << 6) | (unit & 0x3F);             //- Adjust code point with new bits
            type = smTables.maOctetCategory[unit];          //- Look up the code unit's char class
            curr = smTables.maTransitions[curr + type];     //- Advance to the next state
        }
        else
        {
            return ERR;
        }
    }
    return curr;
}
```

A Decoding Example – { .. E2 88 85 .. }



A Decoding Example – { .. E2 88 85 .. }

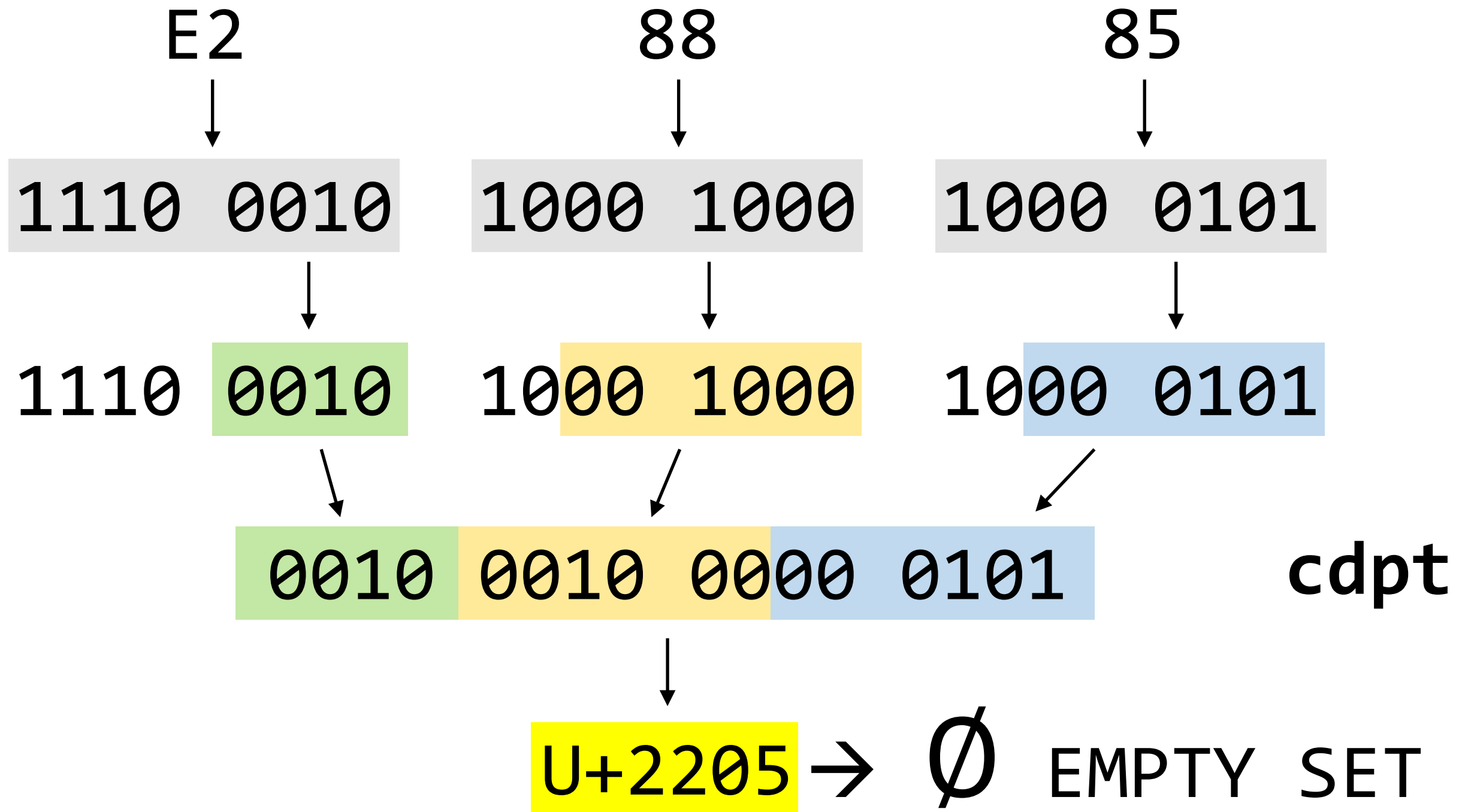


Converting a Single Code Point

```
KEWB_FORCE_INLINE int32_t
UtfUtils::Advance(char8_t const*& pSrc, char8_t const* pSrcEnd, char32_t& cdpt) noexcept
{
    ...

    while (curr > ERR)                                     //- Loop over subsequent code units
    {
        if (pSrc < pSrcEnd)
        {
            unit = *pSrc++;                                //- Cache the current code unit
            cdpt = (cdpt << 6) | (unit & 0x3F);             //- Adjust code point with new bits
            type = smTables.maOctetCategory[unit];          //- Look up the code unit's char class
            curr = smTables.maTransitions[curr + type];     //- Advance to the next state
        }
        else
        {
            return ERR;
        }
    }
    return curr;
}
```


A Decoding Example – { .. E2 88 85 .. }



The Basic Conversion Algorithm (UTF-8 to UTF-32)

```
KEWB_ALIGN_FN std::ptrdiff_t
UtfUtils::BasicConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept
{
    char32_t*    pDstOrig = pDst;
    char32_t      cdpt;

    while (pSrc < pSrcEnd)
    {
        if (Advance(pSrc, pSrcEnd, cdpt) != ERR)
        {
            *pDst++ = cdpt;
        }
        else
        {
            return -1;
        }
    }
    return pDst - pDstOrig;
}
```

Optimizing for ASCII

The Basic Conversion Algorithm (UTF-8 to UTF-32)

```
KEWB_ALIGN_FN std::ptrdiff_t
UtfUtils::BasicConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept
{
    char32_t*    pDstOrig = pDst;
    char32_t     cdpt;

    while (pSrc < pSrcEnd)
    {
        if (Advance(pSrc, pSrcEnd, cdpt) != ERR)
        {
            *pDst++ = cdpt;
        }
        else
        {
            return -1;
        }
    }
    return pDst - pDstOrig;
}
```

Converting a Single Code Point

```
KEWB_FORCE_INLINE int32_t
UtfUtils::Advance(char8_t const*& pSrc, char8_t const* pSrcEnd, char32_t& cdpt) noexcept
{
    FirstUnitInfo    info;    //- The descriptor for the first code unit
    char32_t          unit;    //- The current UTF-8 code unit
    int32_t           type;    //- The code unit's character class
    int32_t           curr;    //- The current DFA state

    info = smTables.maFirstUnitTable[*pSrc++];    //- Look up the first code unit descriptor
    cdpt = info.mFirstOctet;                      //- Get the initial code point value
    curr = info.mNextState;                       //- Get the second state

    while (curr > ERR)                            //- Loop over subsequent code units
    {
        ...
    }
    return curr;
}
```

The ASCII-Optimized Conversion Algorithm (UTF-8 to UTF-32)

```
KEWB_ALIGN_FN std::ptrdiff_t
UtfUtils::FastConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept
{
    char32_t*    pDstOrig = pDst;
    char32_t     cdpt;

    while (pSrc < pSrcEnd)
    {
        if (*pSrc < 0x80)
        {
            *pDst++ = *pSrc++;
        }
        else
        {
            if (Advance(pSrc, pSrcEnd, cdpt) != ERR)
            {
                *pDst++ = cdpt;
            }
            else
            {
                return -1;
            }
        }
    }
    return pDst - pDstOrig;
}
```

Optimizing for ASCII with SSE

The ASCII-Optimized Conversion Algorithm (UTF-8 to UTF-32)

```
KEWB_ALIGN_FN std::ptrdiff_t
UtfUtils::FastConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept
{
    char32_t*    pDstOrig = pDst;
    char32_t     cdpt;

    while (pSrc < pSrcEnd)
    {
        if (*pSrc < 0x80)
        {
            *pDst++ = *pSrc++;
        }
        else
        {
            if (Advance(pSrc, pSrcEnd, cdpt) != ERR)
            {
                *pDst++ = cdpt;
            }
            else
            {
                return -1;
            }
        }
    }
    return pDst - pDstOrig;
}
```


The SSE-Optimized Conversion Algorithm (UTF-8 to UTF-32)

```
KEWB_ALIGN_FN std::ptrdiff_t
UtfUtils::SseConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept
{
    char32_t*    pDstOrig = pDst;
    char32_t     cdpt;

    while (pSrc < (pSrcEnd - sizeof(__m128i)))
    {
        if (*pSrc < 0x80)
        {
            ConvertAsciiWithSse(pSrc, pDst);
        }
        else
        {
            if (Advance(pSrc, pSrcEnd, cdpt) != ERR)
            {
                *pDst++ = cdpt;
            }
            else
            {
                return -1;
            }
        }
    }
    ...
}
```

The SSE-Optimized Conversion Algorithm (UTF-8 to UTF-32)

```
KEWB_ALIGN_FN std::ptrdiff_t  
UtfUtils::SseConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept  
{
```

```
...
```

```
while (pSrc < pSrcEnd)  
{  
    if (*pSrc < 0x80)  
    {  
        *pDst++ = *pSrc++;  
    }  
    else  
    {  
        if (Advance(pSrc, pSrcEnd, cdpt) != ERR)  
        {  
            *pDst++ = cdpt;  
        }  
        else  
        {  
            return -1;  
        }  
    }  
}  
return pDst - pDstOrig;  
}
```

Converting ASCII Character Runs - Overview

```
KEWB_FORCE_INLINE void
UtfUtils::ConvertAsciiWithSse(char8_t const*& pSrc, char32_t*& pDst) noexcept
{
    __m128i    chunk, half, qrtr, zero;           //- SSE "registers"
    int32_t    mask, incr;                        //- ASCII bit mask and advancement

    zero = _mm_set1_epi8(0);                      //- Zero out the interleave register
    chunk = _mm_loadu_si128((__m128i const*) pSrc); //- Load a register with 8-bit values
    mask = _mm_movemask_epi8(chunk);               //- Find the octets with high bit set

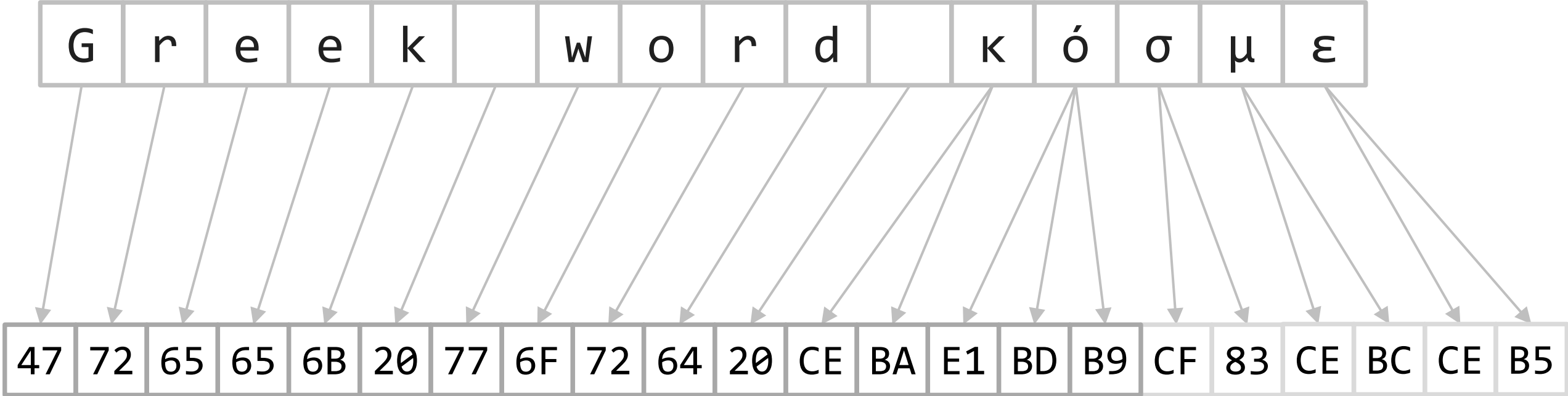
    half = _mm_unpacklo_epi8(chunk, zero);         //- Unpack bytes 0-7 into 16-bit words
    qrtr = _mm_unpacklo_epi16(half, zero);          //- Unpack words 0-3 into 32-bit dwords
    _mm_storeu_si128((__m128i*) pDst, qrtr);        //- Write to memory
    qrtr = _mm_unpackhi_epi16(half, zero);          //- Unpack words 4-7 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 4), qrtr);  //- Write to memory

    half = _mm_unpackhi_epi8(chunk, zero);         //- Unpack bytes 8-15 into 16-bit words
    qrtr = _mm_unpacklo_epi16(half, zero);          //- Unpack words 8-11 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 8), qrtr);  //- Write to memory
    qrtr = _mm_unpackhi_epi16(half, zero);          //- Unpack words 12-15 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 12), qrtr); //- Write to memory

    //- If no bits were set in the mask, then all 16 code units were ASCII.
    //-
    if (mask == 0)
    {
        pSrc += 16;
        pDst += 16;
    }

    //- Otherwise, the number of trailing (low-order) zero bits in the mask is
    //- the number of ASCII code units starting from the lowest byte address.
    else
    {
        incr = GetTrailingZeros(mask);
        pSrc += incr;
        pDst += incr;
    }
}
```

Converting ASCII Character Runs – SSE Example



LSB  **MSB**

Converting ASCII Character Runs

```
KEWB_FORCE_INLINE void
UtfUtils::ConvertAsciiWithSse(char8_t const*& pSrc, char32_t*& pDst) noexcept
{
    __m128i    chunk, half, qrtr, zero;           //- SSE "registers"
    int32_t    mask, incr;                        //- ASCII bit mask and advancement

    zero  = _mm_set1_epi8(0);                     //- Zero out the interleave register
    chunk = _mm_loadu_si128((__m128i const*) pSrc); //- Load a register with 8-bit values
    mask  = _mm_movemask_epi8(chunk);              //- Find octets with high bit set

    half = _mm_unpacklo_epi8(chunk, zero);         //- Unpack bytes 0-7 into 16-bit words
    qrtr  = _mm_unpacklo_epi16(half, zero);         //- Unpack words 0-3 into 32-bit dwords
    _mm_storeu_si128((__m128i*) pDst, qrtr);        //- Write to memory
    qrtr  = _mm_unpackhi_epi16(half, zero);         //- Unpack words 4-7 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 4), qrtr); //- Write to memory

    ...
}
```

Converting ASCII Character Runs

```
KEWB_FORCE_INLINE void
UtfUtils::ConvertAsciiWithSse(char8_t const*& pSrc, char32_t*& pDst) noexcept
{
    __m128i    chunk, half, qrtr, zero;                //- SSE "registers"
    int32_t    mask, incr;                             //- ASCII bit mask and advancement

    zero  = _mm_set1_epi8(0);                          //- Zero out the interleave register
    chunk = _mm_loadu_si128((__m128i const*) pSrc);      //- Load a register with 8-bit values
    mask  = _mm_movemask_epi8(chunk);                  //- Find octets with high bit set

    half = _mm_unpacklo_epi8(chunk, zero);              //- Unpack bytes 0-7 into 16-bit words
    qrtr  = _mm_unpacklo_epi16(half, zero);              //- Unpack words 0-3 into 32-bit dwords
    _mm_storeu_si128((__m128i*) pDst, qrtr);            //- Write to memory
    qrtr  = _mm_unpackhi_epi16(half, zero);              //- Unpack words 4-7 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 4), qrtr);      //- Write to memory

    ...
}
```

Converting ASCII Character Runs

```
KEWB_FORCE_INLINE void
UtfUtils::ConvertAsciiWithSse(char8_t const*& pSrc, char32_t*& pDst) noexcept
{
    __m128i    chunk, half, qrtr, zero;           //- SSE “registers”
    int32_t    mask, incr;                        //- ASCII bit mask and advancement

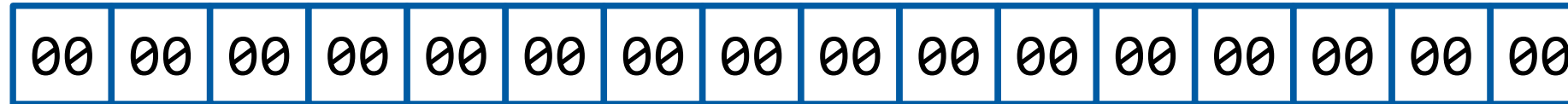
    zero = _mm_set1_epi8(0);                      //- Zero out the interleave register
    chunk = _mm_loadu_si128((__m128i const*) pSrc); //- Load a register with 8-bit values
    mask = _mm_movemask_epi8(chunk);               //- Find octets with high bit set

    half = _mm_unpacklo_epi8(chunk, zero);         //- Unpack bytes 0-7 into 16-bit words
    qrtr = _mm_unpacklo_epi16(half, zero);          //- Unpack words 0-3 into 32-bit dwords
    _mm_storeu_si128((__m128i*) pDst, qrtr);        //- Write to memory
    qrtr = _mm_unpackhi_epi16(half, zero);          //- Unpack words 4-7 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 4), qrtr); //- Write to memory

    ...
}
```

Converting ASCII Character Runs – SSE Example

```
zero = _mm_set1_epi8(0)
```



zero

LSB



MSB

Converting ASCII Character Runs

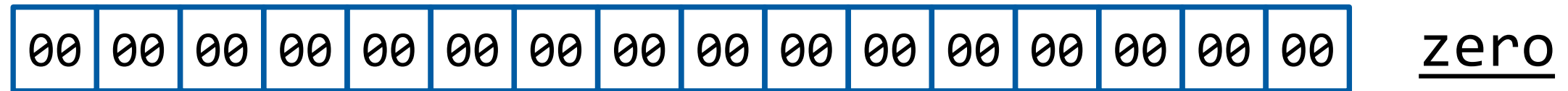
```
KEWB_FORCE_INLINE void
UtfUtils::ConvertAsciiWithSse(char8_t const*& pSrc, char32_t*& pDst) noexcept
{
    __m128i    chunk, half, qrtr, zero;           //- SSE "registers"
    int32_t    mask, incr;                        //- ASCII bit mask and advancement

    zero  = _mm_set1_epi8(0);                    //- Zero out the interleave register
    chunk = _mm_loadu_si128((__m128i const*) pSrc); //- Load a register with 8-bit values
    mask  = _mm_movemask_epi8(chunk);             //- Find octets with high bit set

    half = _mm_unpacklo_epi8(chunk, zero);        //- Unpack bytes 0-7 into 16-bit words
    qrtr  = _mm_unpacklo_epi16(half, zero);        //- Unpack words 0-3 into 32-bit dwords
    _mm_storeu_si128((__m128i*) pDst, qrtr);      //- Write to memory
    qrtr  = _mm_unpackhi_epi16(half, zero);        //- Unpack words 4-7 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 4), qrtr); //- Write to memory

    ...
}
```

Converting ASCII Character Runs – SSE Example



```
chunk = _mm_loadu_si128((__m128i const*) pSrc)
```



LSB



MSB

Converting ASCII Character Runs

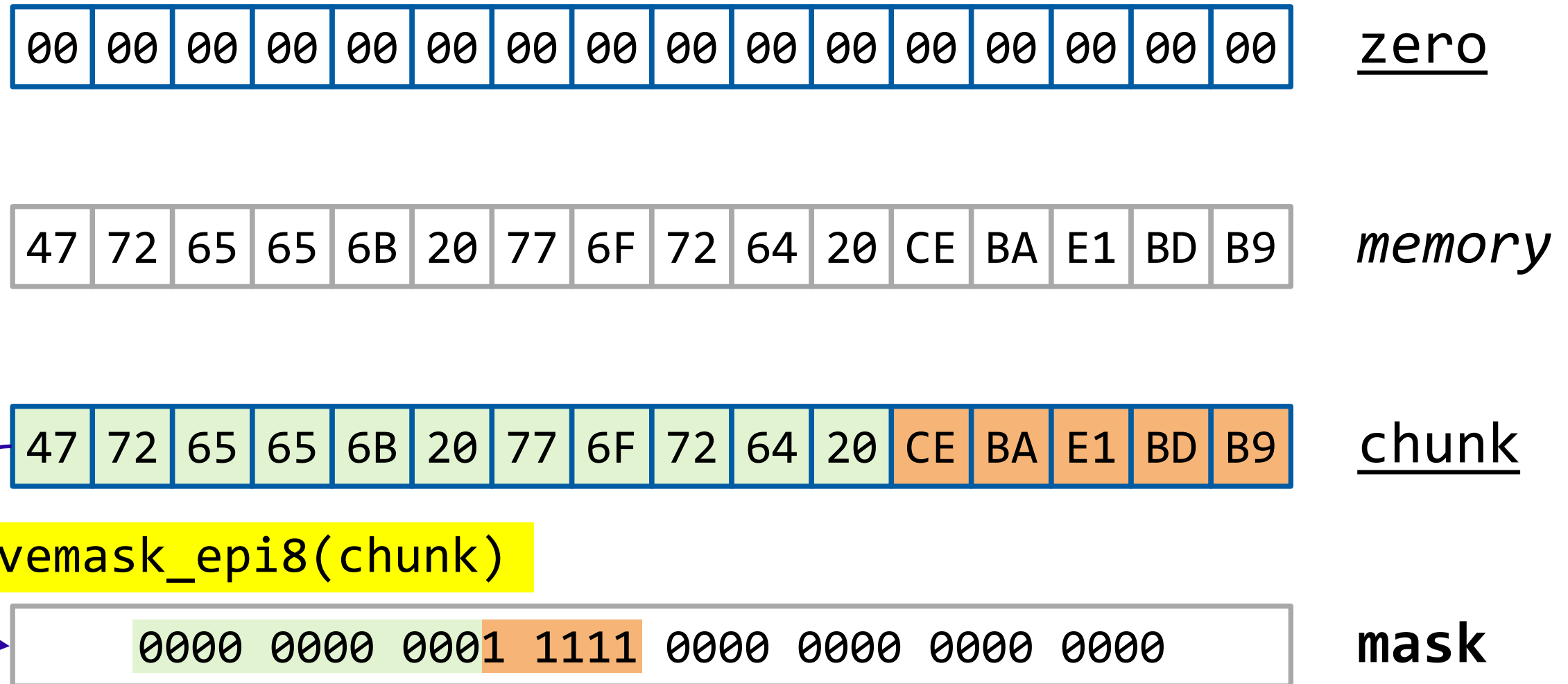
```
KEWB_FORCE_INLINE void
UtfUtils::ConvertAsciiWithSse(char8_t const*& pSrc, char32_t*& pDst) noexcept
{
    __m128i    chunk, half, qrtr, zero;           //- SSE "registers"
    int32_t    mask, incr;                        //- ASCII bit mask and advancement

    zero  = _mm_set1_epi8(0);                     //- Zero out the interleave register
    chunk = _mm_loadu_si128((__m128i const*) pSrc); //- Load a register with 8-bit values
    mask  = _mm_movemask_epi8(chunk);              //- Find octets with high bit set

    half = _mm_unpacklo_epi8(chunk, zero);         //- Unpack bytes 0-7 into 16-bit words
    qrtr  = _mm_unpacklo_epi16(half, zero);         //- Unpack words 0-3 into 32-bit dwords
    _mm_storeu_si128((__m128i*) pDst, qrtr);        //- Write to memory
    qrtr  = _mm_unpackhi_epi16(half, zero);         //- Unpack words 4-7 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 4), qrtr); //- Write to memory

    ...
}
```

Converting ASCII Character Runs – SSE Example



LSB



MSB

Converting ASCII Character Runs

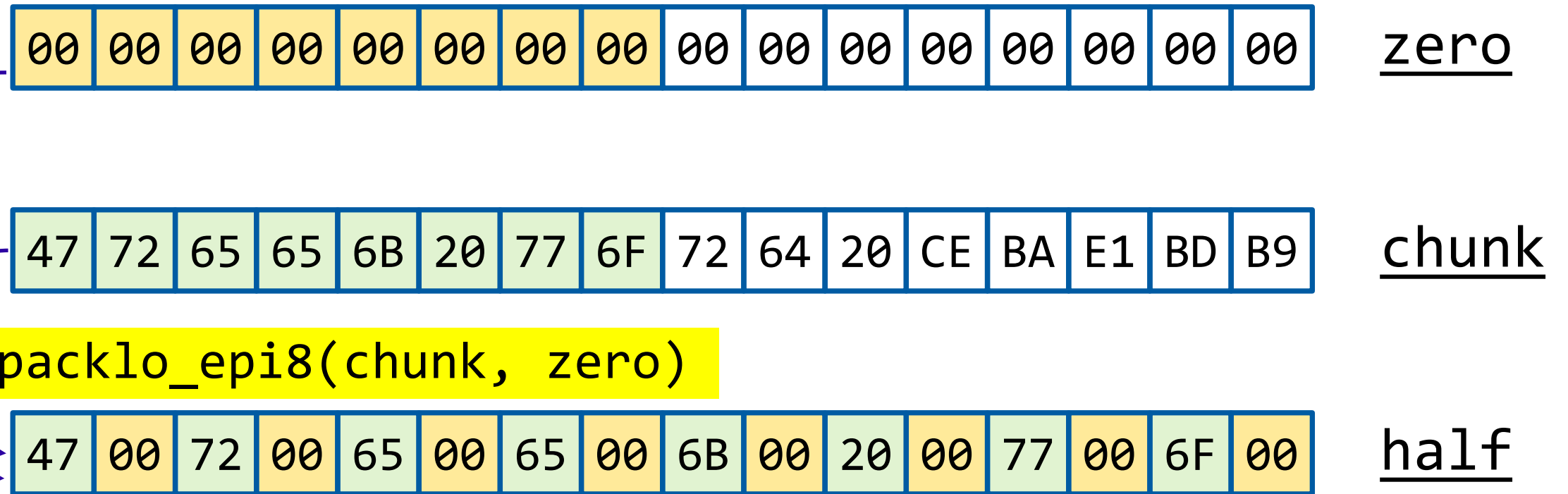
```
KEWB_FORCE_INLINE void
UtfUtils::ConvertAsciiWithSse(char8_t const*& pSrc, char32_t*& pDst) noexcept
{
    __m128i    chunk, half, qrtr, zero;           //- SSE “registers”
    int32_t    mask, incr;                        //- ASCII bit mask and advancement

    zero  = _mm_set1_epi8(0);                     //- Zero out the interleave register
    chunk = _mm_loadu_si128((__m128i const*) pSrc); //- Load a register with 8-bit values
    mask  = _mm_movemask_epi8(chunk);              //- Find octets with high bit set

    half = _mm_unpacklo_epi8(chunk, zero);         //- Unpack bytes 0-7 into 16-bit words
    qrtr  = _mm_unpacklo_epi16(half, zero);         //- Unpack words 0-3 into 32-bit dwords
    _mm_storeu_si128((__m128i*) pDst, qrtr);        //- Write to memory
    qrtr  = _mm_unpackhi_epi16(half, zero);         //- Unpack words 4-7 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 4), qrtr);  //- Write to memory

    ...
}
```

Converting ASCII Character Runs – SSE Example



LSB



MSB

Converting ASCII Character Runs

```
KEWB_FORCE_INLINE void
UtfUtils::ConvertAsciiWithSse(char8_t const*& pSrc, char32_t*& pDst) noexcept
{
    __m128i    chunk, half, qrtr, zero;           //- SSE "registers"
    int32_t    mask, incr;                        //- ASCII bit mask and advancement

    zero  = _mm_set1_epi8(0);                     //- Zero out the interleave register
    chunk = _mm_loadu_si128((__m128i const*) pSrc); //- Load a register with 8-bit values
    mask  = _mm_movemask_epi8(chunk);              //- Find octets with high bit set

    half = _mm_unpacklo_epi8(chunk, zero);         //- Unpack bytes 0-7 into 16-bit words
    qrtr  = _mm_unpacklo_epi16(half, zero);         //- Unpack words 0-3 into 32-bit dwords
    _mm_storeu_si128((__m128i*) pDst, qrtr);        //- Write to memory
    qrtr = _mm_unpackhi_epi16(half, zero);          //- Unpack words 4-7 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 4), qrtr);  //- Write to memory

    ...
}
```

Converting ASCII Character Runs – SSE Example

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

zero

47	00	72	00	65	00	65	00	6B	00	20	00	77	00	6F	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

half

```
qrtr = _mm_unpacklo_epi16(half, zero)
```

47	00	00	00	72	00	00	00	65	00	00	00	65	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

qrtr

LSB



MSB

Converting ASCII Character Runs

```
KEWB_FORCE_INLINE void
UtfUtils::ConvertAsciiWithSse(char8_t const*& pSrc, char32_t*& pDst) noexcept
{
    __m128i    chunk, half, qrtr, zero;           //- SSE "registers"
    int32_t    mask, incr;                        //- ASCII bit mask and advancement

    zero  = _mm_set1_epi8(0);                     //- Zero out the interleave register
    chunk = _mm_loadu_si128((__m128i const*) pSrc); //- Load a register with 8-bit values
    mask  = _mm_movemask_epi8(chunk);              //- Find octets with high bit set

    half = _mm_unpacklo_epi8(chunk, zero);         //- Unpack bytes 0-7 into 16-bit words
    qrtr  = _mm_unpacklo_epi16(half, zero);         //- Unpack words 0-3 into 32-bit dwords
    _mm_storeu_si128((__m128i*) pDst, qrtr);        //- Write to memory
    qrtr  = _mm_unpackhi_epi16(half, zero);         //- Unpack words 4-7 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 4), qrtr); //- Write to memory

    ...
}
```

Converting ASCII Character Runs – SSE Example

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

zero

47	00	72	00	65	00	65	00	6B	00	20	00	77	00	6F	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

half

47	00	00	00	72	00	00	00	65	00	00	00	65	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

qrtr

`_mm_storeu_si128((__m128i*) pDst, qrtr)`

pDst →

47	00	00	00	72	00	00	00	65	00	00	00	65	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

memory

LSB



MSB

Converting ASCII Character Runs

```
KEWB_FORCE_INLINE void
UtfUtils::ConvertAsciiWithSse(char8_t const*& pSrc, char32_t*& pDst) noexcept
{
    __m128i    chunk, half, qrtr, zero;           //- SSE "registers"
    int32_t    mask, incr;                        //- ASCII bit mask and advancement

    zero  = _mm_set1_epi8(0);                     //- Zero out the interleave register
    chunk = _mm_loadu_si128((__m128i const*) pSrc); //- Load a register with 8-bit values
    mask  = _mm_movemask_epi8(chunk);              //- Find octets with high bit set

    half = _mm_unpacklo_epi8(chunk, zero);         //- Unpack bytes 0-7 into 16-bit words
    qrtr  = _mm_unpacklo_epi16(half, zero);         //- Unpack words 0-3 into 32-bit dwords
    _mm_storeu_si128((__m128i*) pDst, qrtr);        //- Write to memory
    qrtr = _mm_unpackhi_epi16(half, zero);         //- Unpack words 4-7 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 4), qrtr); //- Write to memory

    ...
}
```

Converting ASCII Character Runs – SSE Example

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

zero

47	00	72	00	65	00	65	00	6B	00	20	00	77	00	6F	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

half

```
qrtr = _mm_unpackhi_epi16(half, zero)
```

6B	00	00	00	20	00	00	00	77	00	00	00	6F	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

qrtr

LSB



MSB

Converting ASCII Character Runs

```
KEWB_FORCE_INLINE void
UtfUtils::ConvertAsciiWithSse(char8_t const*& pSrc, char32_t*& pDst) noexcept
{
    __m128i    chunk, half, qrtr, zero;           //- SSE "registers"
    int32_t    mask, incr;                        //- ASCII bit mask and advancement

    zero  = _mm_set1_epi8(0);                     //- Zero out the interleave register
    chunk = _mm_loadu_si128((__m128i const*) pSrc); //- Load a register with 8-bit values
    mask  = _mm_movemask_epi8(chunk);              //- Find octets with high bit set

    half = _mm_unpacklo_epi8(chunk, zero);         //- Unpack bytes 0-7 into 16-bit words
    qrtr  = _mm_unpacklo_epi16(half, zero);         //- Unpack words 0-3 into 32-bit dwords
    _mm_storeu_si128((__m128i*) pDst, qrtr);        //- Write to memory
    qrtr  = _mm_unpackhi_epi16(half, zero);         //- Unpack words 4-7 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 4), qrtr); //- Write to memory

    ...
}
```

Converting ASCII Character Runs – SSE Example

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

zero

47	00	72	00	65	00	65	00	6B	00	20	00	77	00	6F	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

half

6B	00	00	00	20	00	00	00	77	00	00	00	6F	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

qrtr

`_mm_storeu_si128((__m128i*) pDst + 4, qrtr)`

pDst + 4 →

6B	00	00	00	20	00	00	00	77	00	00	00	6F	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

memory

LSB



MSB

Converting ASCII Character Runs

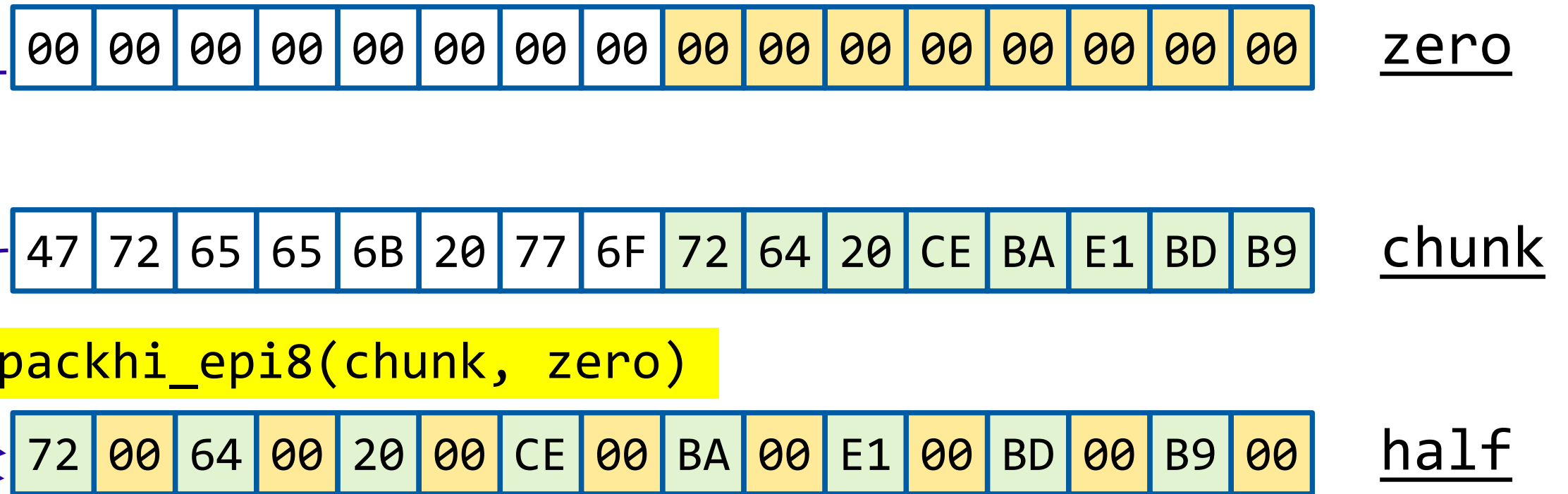
```
KEWB_FORCE_INLINE void
UtfUtils::ConvertAsciiWithSse(char8_t const*& pSrc, char32_t*& pDst) noexcept
{
    ...

    half = _mm_unpacklo_epi8(chunk, zero);           //- Unpack bytes 0-7 into 16-bit words
    qrtr = _mm_unpacklo_epi16(half, zero);           //- Unpack words 0-3 into 32-bit dwords
    _mm_storeu_si128((__m128i*) pDst, qrtr);         //- Write to memory
    qrtr = _mm_unpackhi_epi16(half, zero);           //- Unpack words 4-7 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 4), qrtr);  //- Write to memory

    half = _mm_unpackhi_epi8(chunk, zero);           //- Unpack bytes 8-15 into 16-bit words
    qrtr = _mm_unpacklo_epi16(half, zero);           //- Unpack words 8-11 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 8), qrtr);  //- Write to memory
    qrtr = _mm_unpackhi_epi16(half, zero);           //- Unpack words 12-15 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 12), qrtr); //- Write to memory

    ...
}
```

Converting ASCII Character Runs – SSE Example



LSB



MSB

Converting ASCII Character Runs

```
KEWB_FORCE_INLINE void
UtfUtils::ConvertAsciiWithSse(char8_t const*& pSrc, char32_t*& pDst) noexcept
{
    ...

    half = _mm_unpacklo_epi8(chunk, zero);           //- Unpack bytes 0-7 into 16-bit words
    qrtr = _mm_unpacklo_epi16(half, zero);           //- Unpack words 0-3 into 32-bit dwords
    _mm_storeu_si128((__m128i*) pDst, qrtr);         //- Write to memory
    qrtr = _mm_unpackhi_epi16(half, zero);           //- Unpack words 4-7 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 4), qrtr);   //- Write to memory

    half = _mm_unpackhi_epi8(chunk, zero);           //- Unpack bytes 8-15 into 16-bit words
    qrtr = _mm_unpacklo_epi16(half, zero);           //- Unpack words 8-11 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 8), qrtr);   //- Write to memory
    qrtr = _mm_unpackhi_epi16(half, zero);           //- Unpack words 12-15 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 12), qrtr);  //- Write to memory

    ...
}
```

Converting ASCII Character Runs – SSE Example

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

zero

72	00	64	00	20	00	CE	00	BA	00	E1	00	BD	00	B9	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

half

```
qrtr = _mm_unpacklo_epi16(half, zero)
```

72	00	00	00	64	00	00	00	20	00	00	00	CE	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

qrtr

LSB



MSB

Converting ASCII Character Runs

```
KEWB_FORCE_INLINE void
UtfUtils::ConvertAsciiWithSse(char8_t const*& pSrc, char32_t*& pDst) noexcept
{
    ...

    half = _mm_unpacklo_epi8(chunk, zero);           //- Unpack bytes 0-7 into 16-bit words
    qrtr = _mm_unpacklo_epi16(half, zero);           //- Unpack words 0-3 into 32-bit dwords
    _mm_storeu_si128((__m128i*) pDst, qrtr);         //- Write to memory
    qrtr = _mm_unpackhi_epi16(half, zero);           //- Unpack words 4-7 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 4), qrtr);   //- Write to memory

    half = _mm_unpackhi_epi8(chunk, zero);           //- Unpack bytes 8-15 into 16-bit words
    qrtr = _mm_unpacklo_epi16(half, zero);           //- Unpack words 8-11 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 8), qrtr);   //- Write to memory
    qrtr = _mm_unpackhi_epi16(half, zero);           //- Unpack words 12-15 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 12), qrtr);  //- Write to memory

    ...
}
```

Converting ASCII Character Runs – SSE Example

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

zero

72	00	64	00	20	00	CE	00	BA	00	E1	00	BD	00	B9	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

half

72	00	00	00	64	00	00	00	20	00	00	00	CE	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

qrtr

```
_mm_storeu_si128((__m128i*) pDst + 8, qrtr)
```

pDst + 8 →

72	00	00	00	64	00	00	00	20	00	00	00	CE	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

memory

LSB

MSB

Converting ASCII Character Runs

```
KEWB_FORCE_INLINE void
UtfUtils::ConvertAsciiWithSse(char8_t const*& pSrc, char32_t*& pDst) noexcept
{
    ...

    half = _mm_unpacklo_epi8(chunk, zero);           //- Unpack bytes 0-7 into 16-bit words
    qrtr = _mm_unpacklo_epi16(half, zero);           //- Unpack words 0-3 into 32-bit dwords
    _mm_storeu_si128((__m128i*) pDst, qrtr);         //- Write to memory
    qrtr = _mm_unpackhi_epi16(half, zero);           //- Unpack words 4-7 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 4), qrtr);   //- Write to memory

    half = _mm_unpackhi_epi8(chunk, zero);           //- Unpack bytes 8-15 into 16-bit words
    qrtr = _mm_unpacklo_epi16(half, zero);           //- Unpack words 8-11 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 8), qrtr);   //- Write to memory
    qrtr = _mm_unpackhi_epi16(half, zero);           //- Unpack words 12-15 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 12), qrtr);  //- Write to memory

    ...
}
```

Converting ASCII Character Runs – SSE Example

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

zero

72	00	64	00	20	00	CE	00	BA	00	E1	00	BD	00	B9	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

half

```
qrtr = _mm_unpackhi_epi16(half, zero)
```

BA	00	00	00	E1	00	00	00	BD	00	00	00	B9	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

qrtr

LSB



MSB

Converting ASCII Character Runs

```
KEWB_FORCE_INLINE void
UtfUtils::ConvertAsciiWithSse(char8_t const*& pSrc, char32_t*& pDst) noexcept
{
    ...

    half = _mm_unpacklo_epi8(chunk, zero);           //- Unpack bytes 0-7 into 16-bit words
    qrtr = _mm_unpacklo_epi16(half, zero);           //- Unpack words 0-3 into 32-bit dwords
    _mm_storeu_si128((__m128i*) pDst, qrtr);          //- Write to memory
    qrtr = _mm_unpackhi_epi16(half, zero);           //- Unpack words 4-7 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 4), qrtr);    //- Write to memory

    half = _mm_unpackhi_epi8(chunk, zero);           //- Unpack bytes 8-15 into 16-bit words
    qrtr = _mm_unpacklo_epi16(half, zero);           //- Unpack words 8-11 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 8), qrtr);    //- Write to memory
    qrtr = _mm_unpackhi_epi16(half, zero);           //- Unpack words 12-15 into 32-bit dwords
    _mm_storeu_si128((__m128i*) (pDst + 12), qrtr);    //- Write to memory

    ...
}
```

Converting ASCII Character Runs – SSE Example

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

zero

72	00	64	00	20	00	CE	00	BA	00	E1	00	BD	00	B9	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

half

BA	00	00	00	E1	00	00	00	BD	00	00	00	B9	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

qrtr

`_mm_storeu_si128((__m128i*) pDst + 12, qrtr)`

pDst + 12 →

BA	00	00	00	E1	00	00	00	BD	00	00	00	B9	00	00	00
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

memory

LSB → MSB

Converting ASCII Character Runs – SSE Code

```
KEWB_FORCE_INLINE void
UtfUtils::ConvertAsciiWithSse(char8_t const*& pSrc, char32_t*& pDst) noexcept
{
    ...
    //- If no bits were set in the mask, then all 16 code units were ASCII.
    //
    if (mask == 0)
    {
        pSrc += 16;
        pDst += 16;
    }

    //- Otherwise, the number of trailing (low-order) zero bits in the mask is
    // the number of ASCII code units.
    //
    else
    {
        incr = GetTrailingZeros(mask);
        pSrc += incr;
        pDst += incr;
    }
}
```

Finding the Trailing Zero-Bit Count

```
#if defined KEWB_PLATFORM_LINUX  && (defined KEWB_COMPILER_CLANG  ||  defined KEWB_COMPILER_GCC)
```

```
    KEWB_FORCE_INLINE int32_t  
    UtfUtils::GetTrailingZeros(int32_t x) noexcept  
    {  
        return  __builtin_ctz((unsigned int) x);  
    }
```

```
#elif defined KEWB_PLATFORM_WINDOWS  &&  defined KEWB_COMPILER_MSVC
```

```
    KEWB_FORCE_INLINE int32_t  
    UtfUtils::GetTrailingZeros(int32_t x) noexcept  
    {  
        unsigned long    indx;  
        _BitScanForward(&indx, (unsigned long) x);  
        return (int32_t) indx;  
    }
```

```
#endif
```

Converting ASCII Character Runs – SSE Example

0000 0000 0001 1111 0000 0000 0000 0000

mask

`incr = GetTrailingZeros(mask)`

11 (eleven)

incr

pSrc →

47	72	65	65	6B	20	77	6F	72	64	20	CE	BA	E1	BD	B9
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

pSrc += 11 →

pDst →

47	00	00	00	72	00	00	00	65	00	00	00	65	00	00	00
6B	00	00	00	20	00	00	00	77	00	00	00	6F	00	00	00
72	00	00	00	64	00	00	00	20	00	00	00	CE	00	00	00
BA	00	00	00	E1	00	00	00	BD	00	00	00	B9	00	00	00

pDst += 11 →

LSB

MSB

The SSE-Optimized Conversion Algorithm (UTF-8 to UTF-32)

```
KEWB_ALIGN_FN std::ptrdiff_t
UtfUtils::SseConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept
{
    char32_t*    pDstOrig = pDst;
    char32_t     cdpt;

    while (pSrc < (pSrcEnd - sizeof(__m128i)))
    {
        if (*pSrc < 0x80)
        {
            ConvertAsciiWithSse(pSrc, pDst);
        }
        else
        {
            if (Advance(pSrc, pSrcEnd, cdpt) != ERR)
            {
                *pDst++ = cdpt;
            }
            else
            {
                return -1;
            }
        }
    }
    ...
}
```

Optimizing for ASCII with AVX

The AVX-Optimized Conversion Algorithm (UTF-8 to UTF-32)

```
KEWB_ALIGN_FN std::ptrdiff_t
UtfUtils::AvxConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept
{
    char32_t*    pDstOrig = pDst;
    char32_t     cdpt;

    while (pSrc < (pSrcEnd - sizeof(__m128i)))
    {
        if (*pSrc < 0x80)
        {
            ConvertAsciiWithAvx(pSrc, pDst);
        }
        else
        {
            if (Advance(pSrc, pSrcEnd, cdpt) != ERR)
            {
                *pDst++ = cdpt;
            }
            else
            {
                return -1;
            }
        }
    }
    ...
}
```

Converting ASCII Character Runs – AVX Example

```
KEWB_ALIGN_FN std::ptrdiff_t
```

```
UtfUtils::AvxConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept  
{
```

```
    ...
```

```
    if (*pSrc < 0x80)
```

```
    {
```

```
        int32_t    mask, incr;
```

```
        __m128i    chunk;
```

```
        __m256i    half;
```

```
        //- SSE register
```

```
        //- AVX register
```

```
        chunk = _mm_loadu_si128((__m128i const*) pSrc);
```

```
        //- Load a register with bytes
```

```
        half = _mm256_cvtepu8_epi32(chunk);
```

```
        //- Zero-extend lower half
```

```
        _mm256_storeu_si256((__m256i*) pDst, half);
```

```
        //- Store to memory
```

```
        mask = _mm_movemask_epi8(chunk);
```

```
        //- Find bytes w/ high bit set
```

```
        chunk = _mm_shuffle_epi32(chunk, _MM_SHUFFLE(1,0,3,2));
```

```
        //- Swap upper and lower
```

```
        half = _mm256_cvtepu8_epi32(chunk);
```

```
        //- Zero-extend lower half
```

```
        _mm256_storeu_si256((__m256i*) (pDst + 8), half);
```

```
        //- Store to memory
```

```
    ...
```

```
}
```

Converting ASCII Character Runs – AVX Example

```
KEWB_ALIGN_FN std::ptrdiff_t
UtfUtils::AvxConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept
{
    ...

    if (*pSrc < 0x80)
    {
        int32_t      mask, incr;
        __m128i      chunk;                                     //- SSE register
        __m256i      half;                                     //- AVX register

        chunk = _mm_loadu_si128((__m128i const*) pSrc);         //- Load a register with bytes
        half = _mm256_cvtepu8_epi32(chunk);                     //- Zero-extend lower half
        _mm256_storeu_si256((__m256i*) pDst, half);             //- Store to memory

        mask = _mm_movemask_epi8(chunk);                        //- Find bytes w/ high bit set
        chunk = _mm_shuffle_epi32(chunk, _MM_SHUFFLE(1,0,3,2)); //- Swap upper and lower
        half = _mm256_cvtepu8_epi32(chunk);                     //- Zero-extend lower half
        _mm256_storeu_si256((__m256i*) (pDst + 8), half);      //- Store to memory

        ...
    }
}
```


Converting ASCII Character Runs – AVX Example

pSrc →

47	72	65	65	6B	20	77	6F	72	64	20	CE	BA	E1	BD	B9
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

memory

```
chunk = _mm_loadu_si128((__m128i const*) pSrc)
```

47	72	65	65	6B	20	77	6F	72	64	20	CE	BA	E1	BD	B9
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

chunk

LSB



MSB

The AVX-Optimized Conversion Algorithm (UTF-8 to UTF-32)

```
KEWB_ALIGN_FN std::ptrdiff_t
UtfUtils::AvxConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept
{
    ...

    if (*pSrc < 0x80)
    {
        int32_t      mask, incr;
        __m128i      chunk;                                     //- SSE register
        __m256i      half;                                     //- AVX register

        chunk = _mm_loadu_si128((__m128i const*) pSrc);         //- Load a register with bytes
        half = _mm256_cvtepu8_epi32(chunk);                     //- Zero-extend lower half
        _mm256_storeu_si256((__m256i*) pDst, half);             //- Store to memory

        mask = _mm_movemask_epi8(chunk);                        //- Find bytes w/ high bit set
        chunk = _mm_shuffle_epi32(chunk, _MM_SHUFFLE(1,0,3,2)); //- Swap upper and lower
        half = _mm256_cvtepu8_epi32(chunk);                     //- Zero-extend lower half
        _mm256_storeu_si256((__m256i*) (pDst + 8), half);      //- Store to memory

        ...
    }
}
```

Converting ASCII Character Runs – AVX Example

47	72	65	65	6B	20	77	6F	72	64	20	CE	BA	E1	BD	B9
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

chunk

half = `_mm256_cvtepu8_epi32(chunk)`

47	00	00	00	72	00	00	00	65	00	00	00	65	00	6F	00
6B	00	00	00	20	00	00	00	77	00	00	00	6F	00	00	00

half

LSB



MSB

Converting ASCII Character Runs – AVX Example

```
KEWB_ALIGN_FN std::ptrdiff_t
UtfUtils::AvxConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept
{
    ...

    if (*pSrc < 0x80)
    {
        int32_t      mask, incr;
        __m128i      chunk;                                     //- SSE register
        __m256i      half;                                     //- AVX register

        chunk = _mm_loadu_si128((__m128i const*) pSrc);         //- Load a register with bytes
        half  = _mm256_cvtepu8_epi32(chunk);                   //- Zero-extend lower half
        _mm256_storeu_si256((__m256i*) pDst, half);            //- Store to memory

        mask  = _mm_movemask_epi8(chunk);                      //- Find bytes w/ high bit set
        chunk = _mm_shuffle_epi32(chunk, _MM_SHUFFLE(1,0,3,2)); //- Swap upper and lower
        half  = _mm256_cvtepu8_epi32(chunk);                   //- Zero-extend lower half
        _mm256_storeu_si256((__m256i*) (pDst + 8), half);      //- Store to memory

        ...
    }
}
```

Converting ASCII Character Runs – AVX Example

47	72	65	65	6B	20	77	6F	72	64	20	CE	BA	E1	BD	B9
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

chunk

47	00	00	00	72	00	00	00	65	00	00	00	65	00	00	00
6B	00	00	00	20	00	00	00	77	00	00	00	6F	00	00	00

half

```
_mm256_storeu_si256((__m256i*) pDst, half)
```

pDst →

47	00	00	00	72	00	00	00	65	00	00	00	65	00	00	00
6B	00	00	00	20	00	00	00	77	00	00	00	6F	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00



Converting ASCII Character Runs – AVX Example

```
KEWB_ALIGN_FN std::ptrdiff_t
UtfUtils::AvxConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept
{
    ...

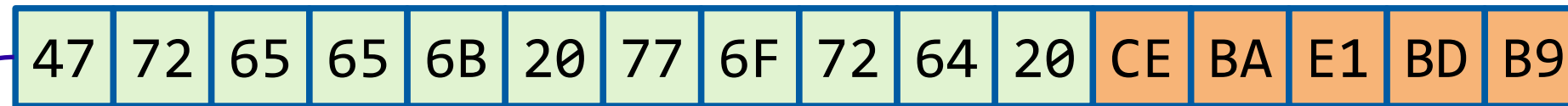
    if (*pSrc < 0x80)
    {
        int32_t      mask, incr;
        __m128i      chunk;                                     //- SSE register
        __m256i      half;                                     //- AVX register

        chunk = _mm_loadu_si128((__m128i const*) pSrc);         //- Load a register with bytes
        half  = _mm256_cvtepu8_epi32(chunk);                   //- Zero-extend lower half
        _mm256_storeu_si256((__m256i*) pDst, half);            //- Store to memory

        mask  = _mm_movemask_epi8(chunk);                      //- Find bytes w/ high bit set
        chunk = _mm_shuffle_epi32(chunk, _MM_SHUFFLE(1,0,3,2)); //- Swap upper and lower
        half  = _mm256_cvtepu8_epi32(chunk);                   //- Zero-extend lower half
        _mm256_storeu_si256((__m256i*) (pDst + 8), half);      //- Store to memory

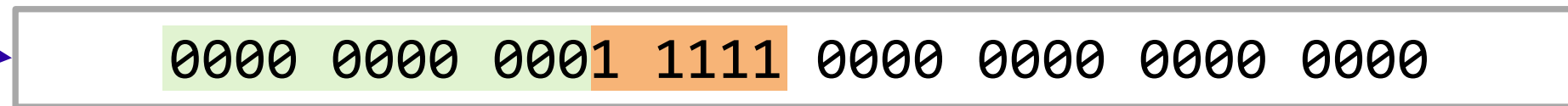
        ...
    }
}
```

Converting ASCII Character Runs – AVX Example



chunk

```
mask = _mm_movemask_epi8(chunk)
```



mask

LSB



MSB

Converting ASCII Character Runs – AVX Example

```
KEWB_ALIGN_FN std::ptrdiff_t
UtfUtils::AvxConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept
{
    ...

    if (*pSrc < 0x80)
    {
        int32_t      mask, incr;
        __m128i      chunk;                                     //- SSE register
        __m256i      half;                                     //- AVX register

        chunk = _mm_loadu_si128((__m128i const*) pSrc);         //- Load a register with bytes
        half  = _mm256_cvtepu8_epi32(chunk);                   //- Zero-extend lower half
        _mm256_storeu_si256((__m256i*) pDst, half);            //- Store to memory

        mask  = _mm_movemask_epi8(chunk);                      //- Find bytes w/ high bit set
        chunk = _mm_shuffle_epi32(chunk, _MM_SHUFFLE(1,0,3,2)); //- Swap upper and lower
        half  = _mm256_cvtepu8_epi32(chunk);                   //- Zero-extend lower half
        _mm256_storeu_si256((__m256i*) (pDst + 8), half);      //- Store to memory

        ...
    }
}
```


Converting ASCII Character Runs – AVX Example

47	72	65	65	6B	20	77	6F	72	64	20	CE	BA	E1	BD	B9
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

chunk

```
chunk = _mm_shuffle_epi32(chunk, _MM_SHUFFLE(1,0,3,2))
```

72	64	20	CE	BA	E1	BD	B9	47	72	65	65	6B	20	77	6F
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

chunk

LSB



MSB

The AVX-Optimized Conversion Algorithm (UTF-8 to UTF-32)

```
KEWB_ALIGN_FN std::ptrdiff_t
UtfUtils::AvxConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept
{
    ...

    if (*pSrc < 0x80)
    {
        int32_t      mask, incr;
        __m128i      chunk;                                     //- SSE register
        __m256i      half;                                     //- AVX register

        chunk = _mm_loadu_si128((__m128i const*) pSrc);         //- Load a register with bytes
        half = _mm256_cvtepu8_epi32(chunk);                    //- Zero-extend lower half
        _mm256_storeu_si256((__m256i*) pDst, half);            //- Store to memory

        mask = _mm_movemask_epi8(chunk);                       //- Find bytes w/ high bit set
        chunk = _mm_shuffle_epi32(chunk, _MM_SHUFFLE(1,0,3,2)); //- Swap upper and lower
        half = _mm256_cvtepu8_epi32(chunk);                    //- Zero-extend lower half
        _mm256_storeu_si256((__m256i*) (pDst + 8), half);      //- Store to memory

        ...
    }
}
```

Converting ASCII Character Runs – AVX Example

72	64	20	CE	BA	E1	BD	B9	47	72	65	65	6B	20	77	6F
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

chunk

```
half = _mm256_cvtepu8_epi32(chunk)
```

72	00	00	00	64	00	00	00	20	00	00	00	CE	00	00	00
BA	00	00	00	E1	00	00	00	BD	00	00	00	B9	00	00	00

half

LSB



MSB

Converting ASCII Character Runs – AVX Example

```
KEWB_ALIGN_FN std::ptrdiff_t
UtfUtils::AvxConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept
{
    ...

    if (*pSrc < 0x80)
    {
        int32_t      mask, incr;
        __m128i      chunk;                                     //- SSE register
        __m256i      half;                                     //- AVX register

        chunk = _mm_loadu_si128((__m128i const*) pSrc);         //- Load a register with bytes
        half = _mm256_cvtepu8_epi32(chunk);                    //- Zero-extend lower half
        _mm256_storeu_si256((__m256i*) pDst, half);            //- Store to memory

        mask = _mm_movemask_epi8(chunk);                       //- Find bytes w/ high bit set
        chunk = _mm_shuffle_epi32(chunk, _MM_SHUFFLE(1,0,3,2)); //- Swap upper and lower
        half = _mm256_cvtepu8_epi32(chunk);                    //- Zero-extend lower half
        _mm256_storeu_si256((__m256i*) (pDst + 8), half);      //- Store to memory

        ...
    }
}
```

Converting ASCII Character Runs – SSE Example

47	72	65	65	6B	20	77	6F	72	64	20	CE	BA	E1	BD	B9
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

chunk

72	00	00	00	64	00	00	00	20	00	00	00	CE	00	00	00
BA	00	00	00	E1	00	00	00	BD	00	00	00	B9	00	00	00

half

```
_mm256_storeu_si256((__m256i*)(pDst + 8), half)
```

pDst + 8 →

47	00	00	00	72	00	00	00	65	00	00	00	65	00	00	00
6B	00	00	00	20	00	00	00	77	00	00	00	6F	00	00	00
72	00	00	00	64	00	00	00	20	00	00	00	CE	00	00	00
BA	00	00	00	E1	00	00	00	BD	00	00	00	B9	00	00	00



Converting ASCII Character Runs – AVX Example

```
KEWB_ALIGN_FN std::ptrdiff_t
UtfUtils::AvxConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept
{
    ...

    if (*pSrc < 0x80)
    {
        ...

        if (mask == 0)
        {
            pSrc += 16;
            pDst += 16;
        }
        else
        {
            incr = GetTrailingZeros(mask);
            pSrc += incr;
            pDst += incr;
        }
    }
    ...
}
```

Converting ASCII Character Runs – AVX Example

0000 0000 0001 1111 0000 0000 0000 0000

mask

`incr = GetTrailingZeros(mask)`

11 (eleven)

incr

pSrc →

47	72	65	65	6B	20	77	6F	72	64	20	CE	BA	E1	BD	B9
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

pSrc += 11 →

pDst →

47	00	00	00	72	00	00	00	65	00	00	00	65	00	00	00
6B	00	00	00	20	00	00	00	77	00	00	00	6F	00	00	00
72	00	00	00	64	00	00	00	20	00	00	00	CE	00	00	00
BA	00	00	00	E1	00	00	00	BD	00	00	00	B9	00	00	00

pDst += 11 →

LSB

MSB

The AVX-Optimized Conversion Algorithm (UTF-8 to UTF-32)

```
KEWB_ALIGN_FN std::ptrdiff_t
UtfUtils::AvxConvert(char8_t const* pSrc, char8_t const* pSrcEnd, char32_t* pDst) noexcept
{
    char32_t*    pDstOrig = pDst;
    char32_t     cdpt;

    while (pSrc < (pSrcEnd - sizeof(__m128i)))
    {
        if (*pSrc < 0x80)
        {
            ConvertAsciiWithAvx(pSrc, pDst);
        }
        else
        {
            if (Advance(pSrc, pSrcEnd, cdpt) != ERR)
            {
                *pDst++ = cdpt;
            }
            else
            {
                return -1;
            }
        }
    }
    ...
}
```


Testing and Benchmarks

- Ubuntu 18.04 Core i9
 - **GCC 9.3**, all code compiled with `-O3 -mavx2 -march=skylake`
 - **Clang 10.0.1**, all code compiled with `-O3 -mavx2 -march=skylake`
- Windows 10 Core i9
 - **Visual Studio 16.7.4**, all code compiled with `/O2 /Ob2 /Oi /Ot`

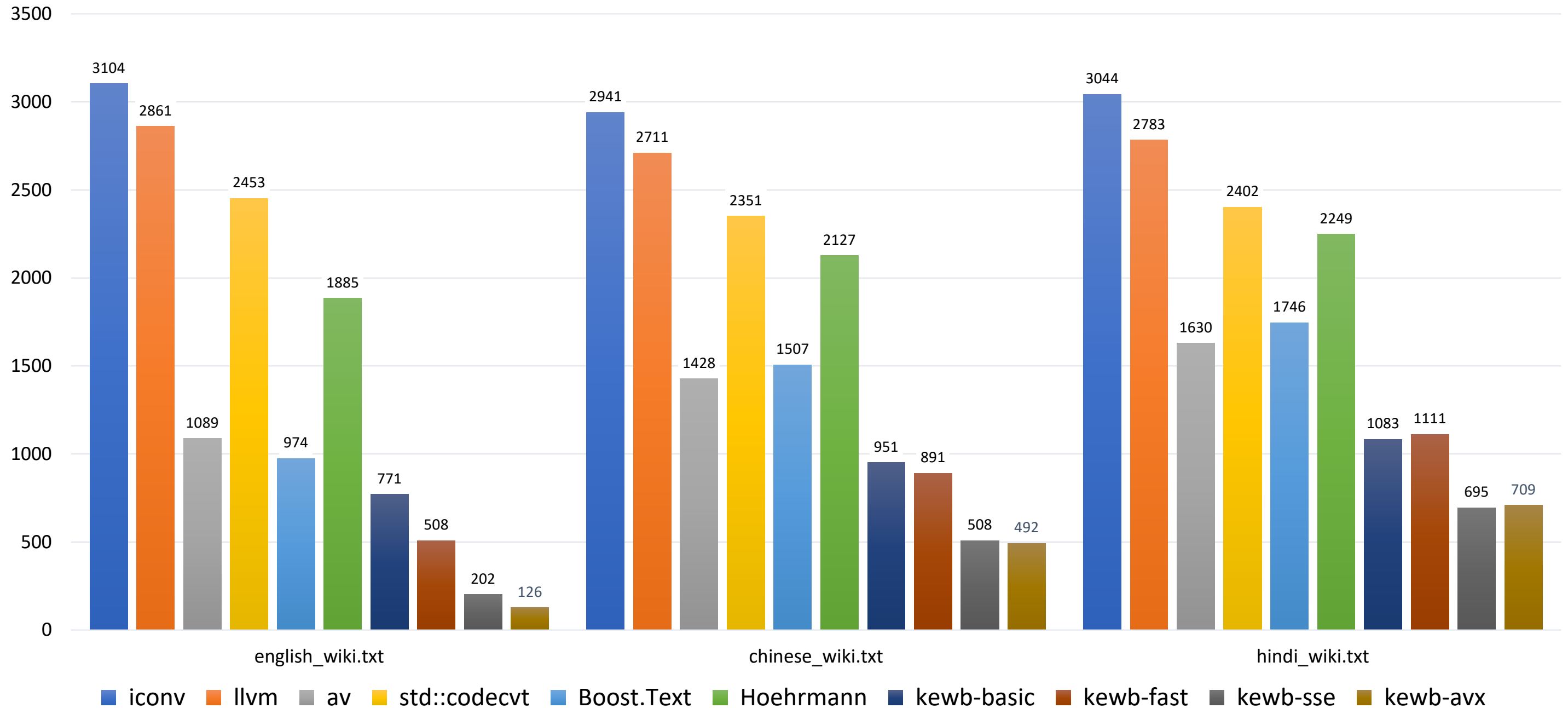
- Nine input files
 - english_wiki.txt
 - chinese_wiki.txt
 - hindi_wiki.txt
 - portuguese_wiki.txt
 - russian_wiki.txt
 - swedish_wiki.txt
 - stress_test_0.txt – 100K ASCII code points (100K code units)
 - stress_test_1.txt – 100K Chinese code points (300K code units)
 - stress_test_2.txt – 50K Chinese code points interleaved with 50K ASCII code points (200K code units)
- Taken directly from wikipedia.org

Testing Methodology – Reference Libraries

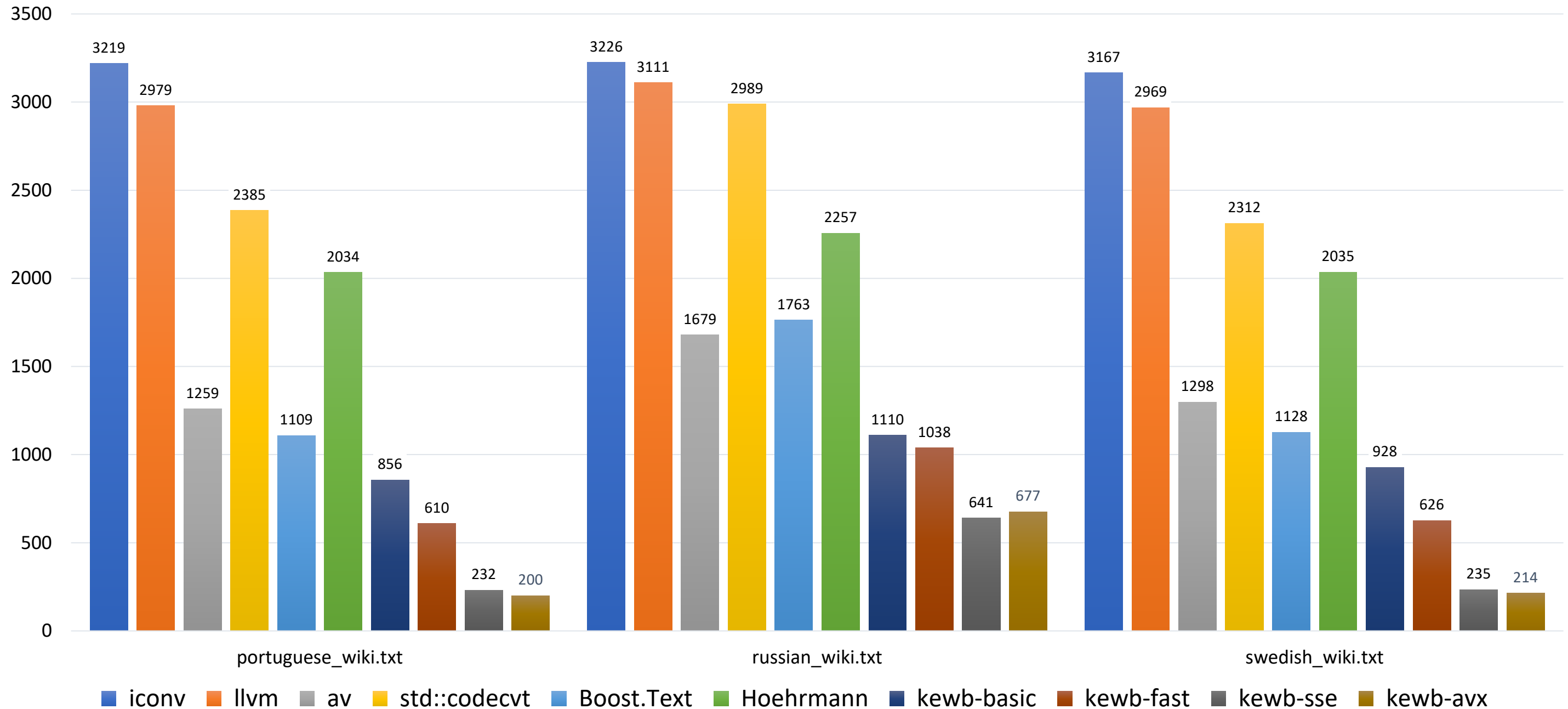
- `iconv` – GNU libiconv, used here as the “gold standard”
- `LLVM` – UTF conversion functions from the LLVM distribution
- `AV` – UTF-8 to UTF-32 conversion by Alexey Vatchenko
- `std::codecvt` – Standard library’s UTF conversion
- `Boost.Text` – Iterator-based interface to UTF conversion by Zach Laine
- `BH` – Alternative DFA-based conversion by Bjoern Hoehrmann

- Timings for each file were obtained by:
 1. Reading the input file
 2. Creating an oversized output buffer
 3. Starting the timer
 4. Entering the timing loop
 5. Performing conversion of the input buffer multiple times
 - The number of repetitions was such that 1GB of input text was processed
 6. Exiting the timing loop
 7. Stopping the timer
 8. Collecting and collating results
- To pass, a library's result had to agree with `iconv()`

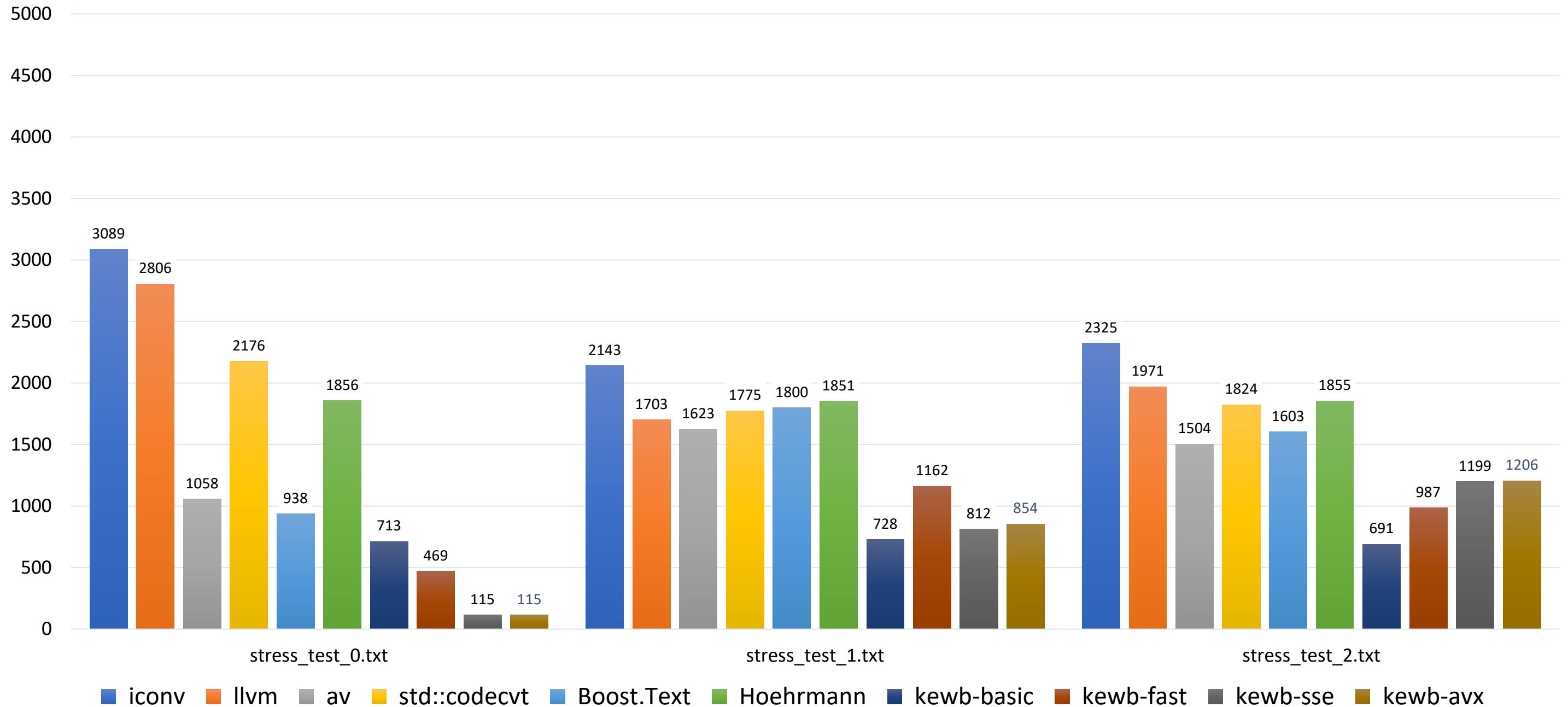
Conversion Time (msec)



Conversion Time (msec)

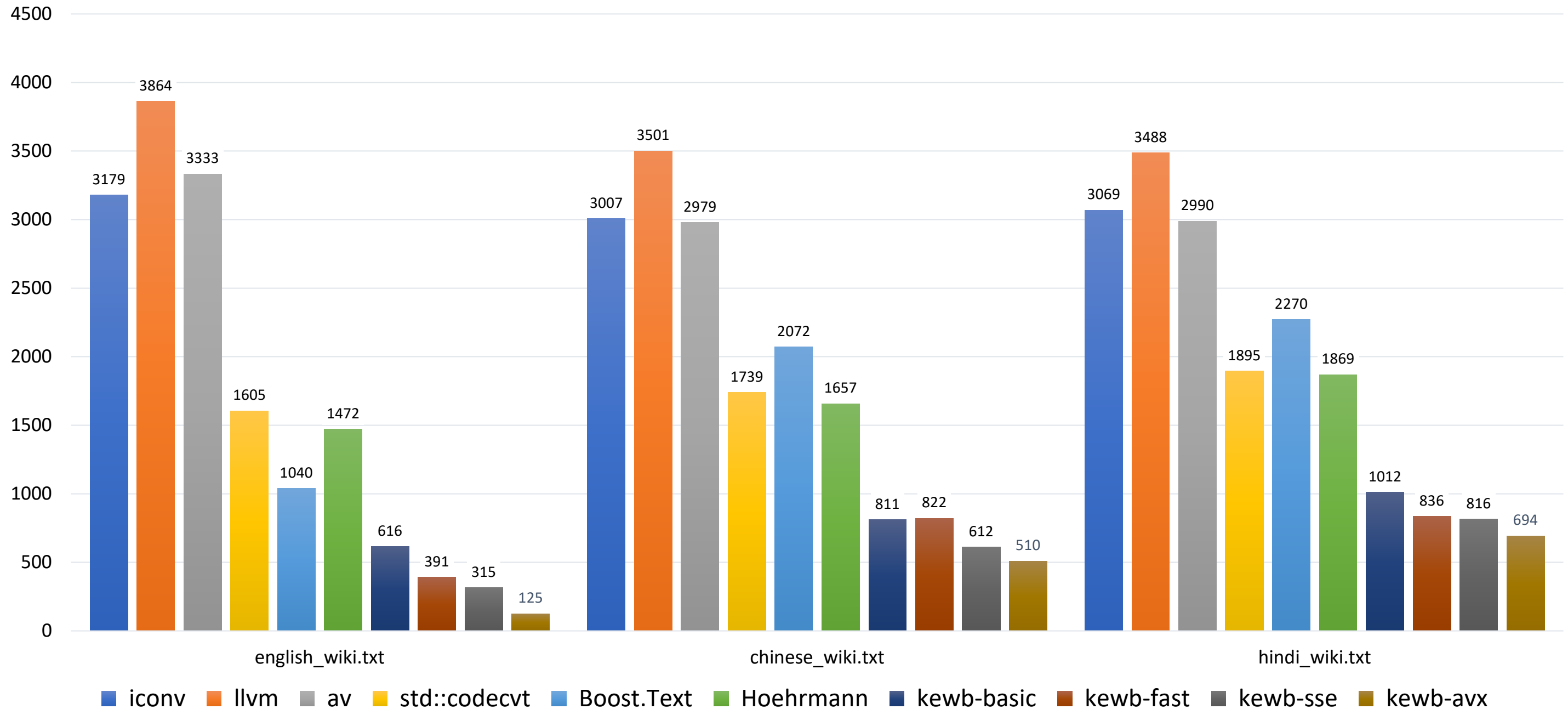


Conversion Time (msec)



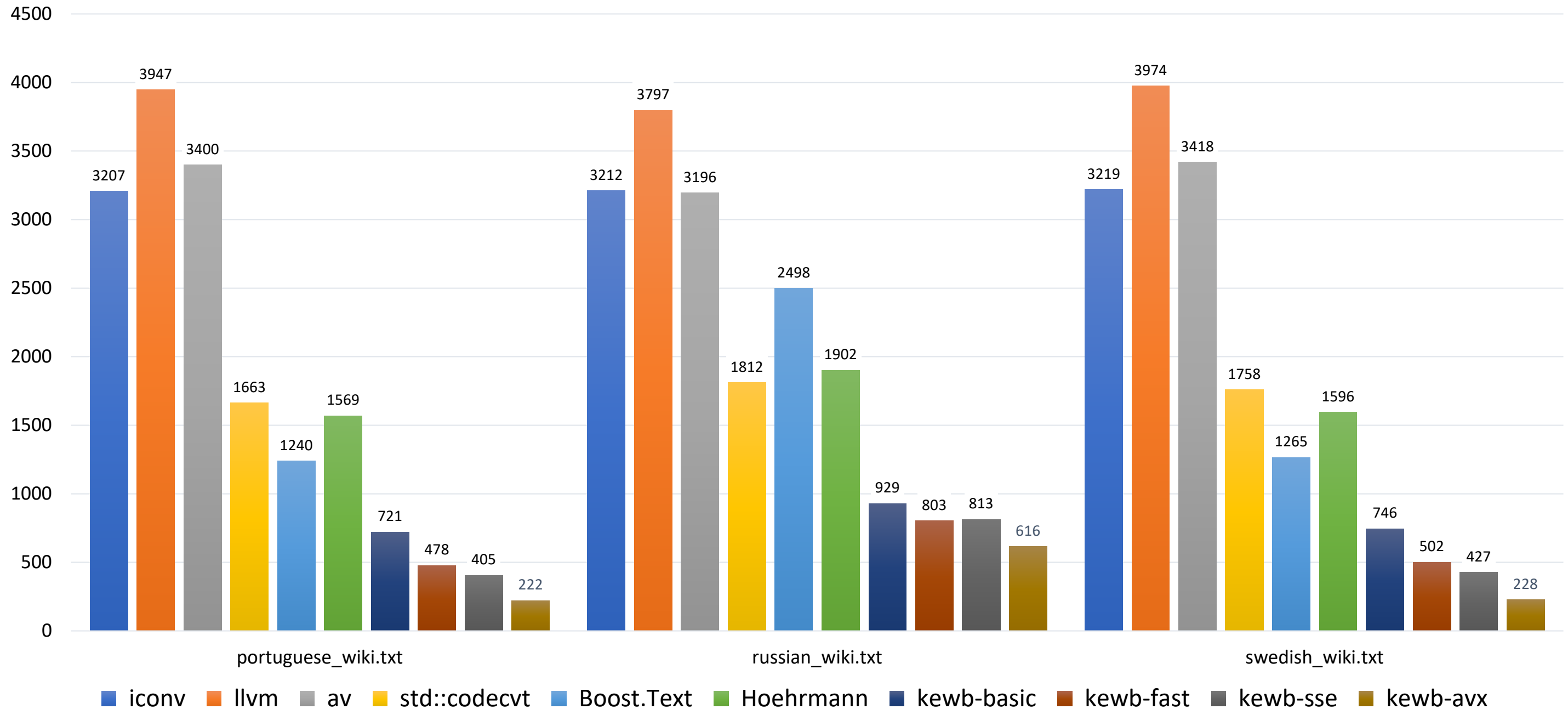
Clang 10.0.1 – Ubuntu 18.04 – Core i9 – UTF-32

Conversion Time (msec)



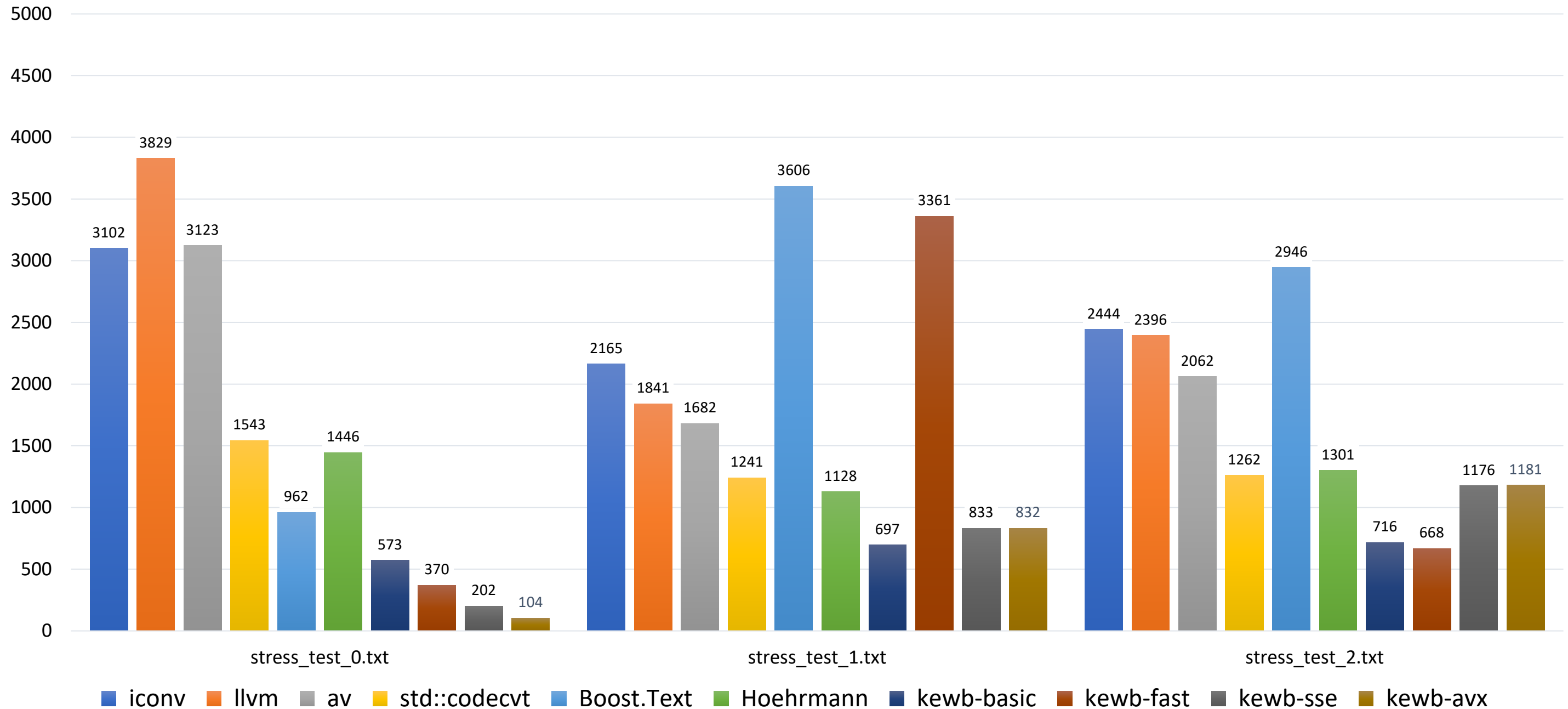
Clang 10.0.1 – Ubuntu 18.04 – Core i9 – UTF-32

Conversion Time (msec)

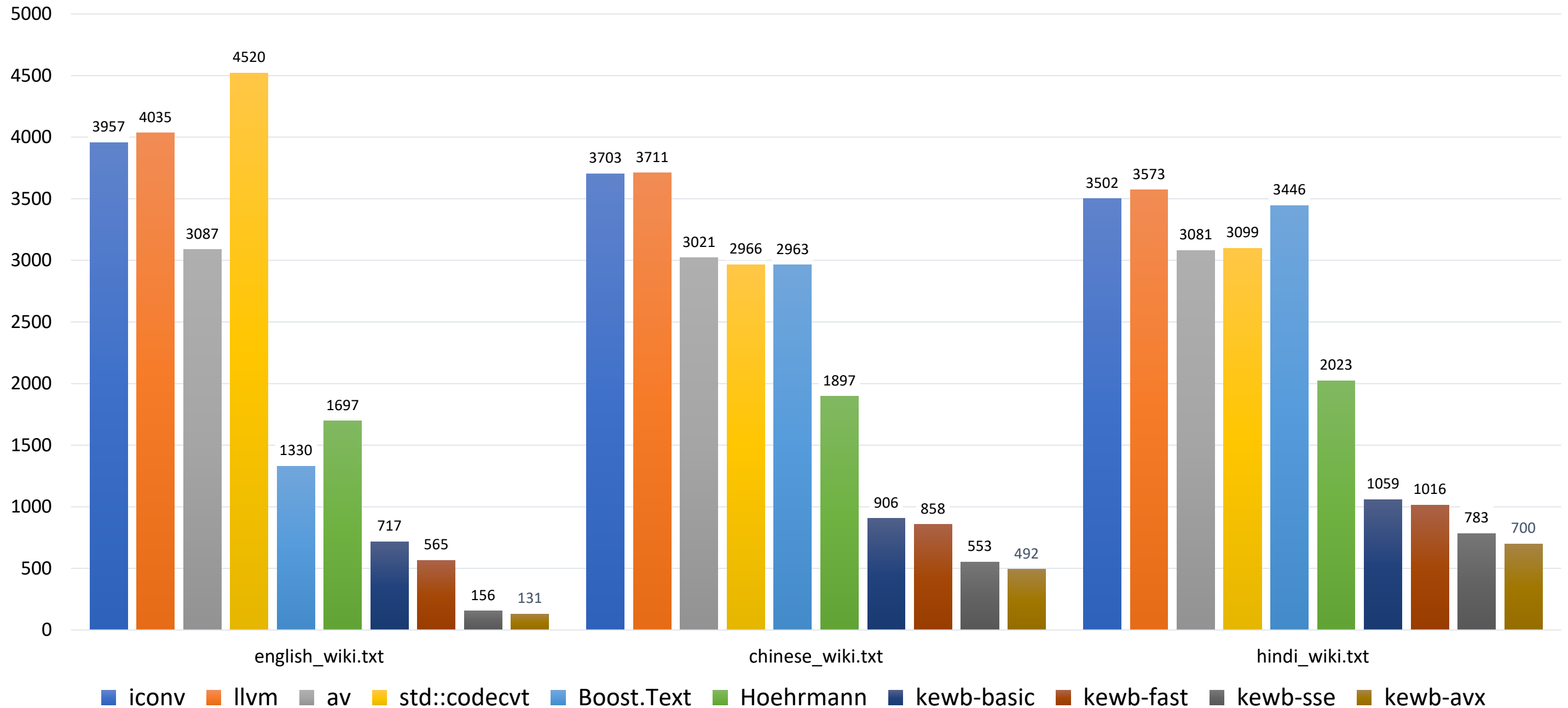


Clang 10.0.1 – Ubuntu 18.04 – Core i9 – UTF-32

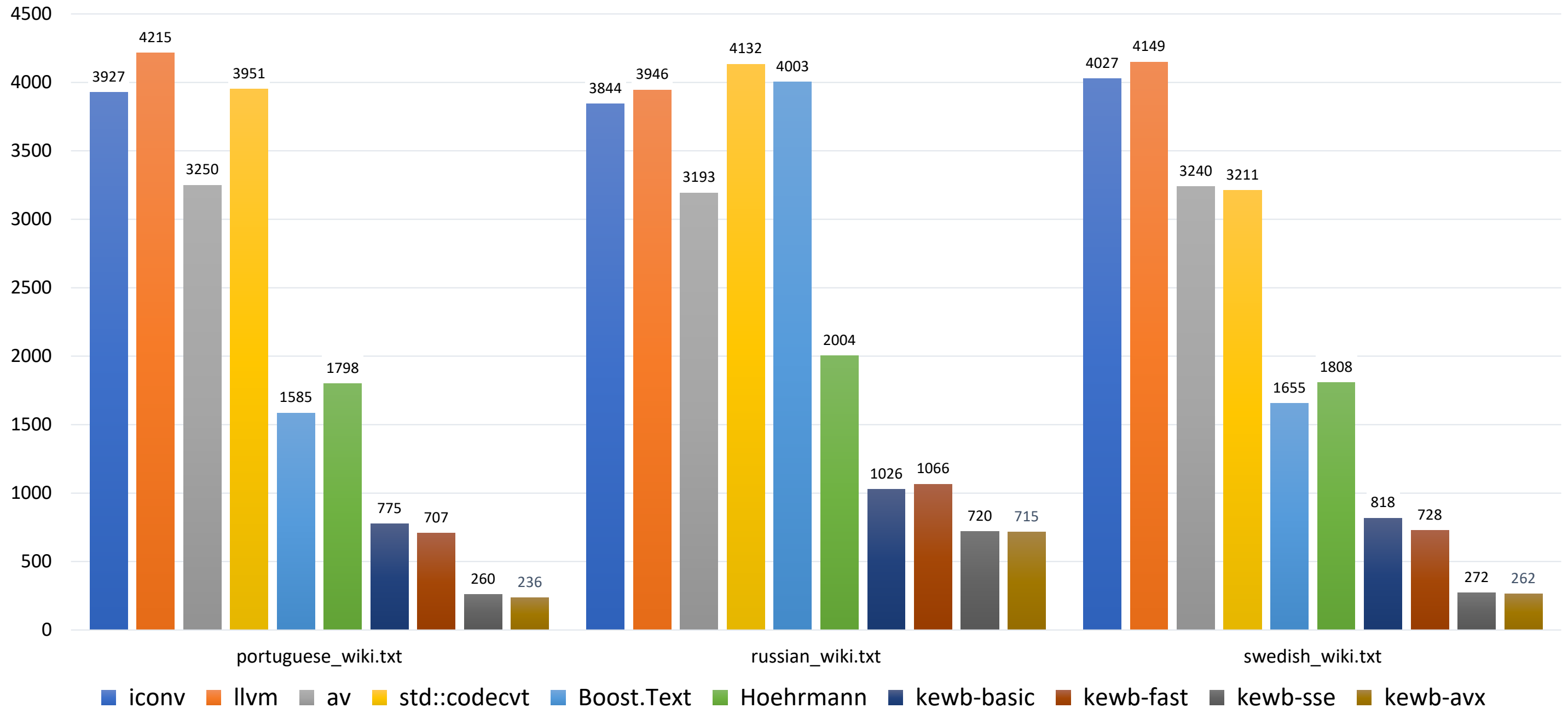
Conversion Time (msec)



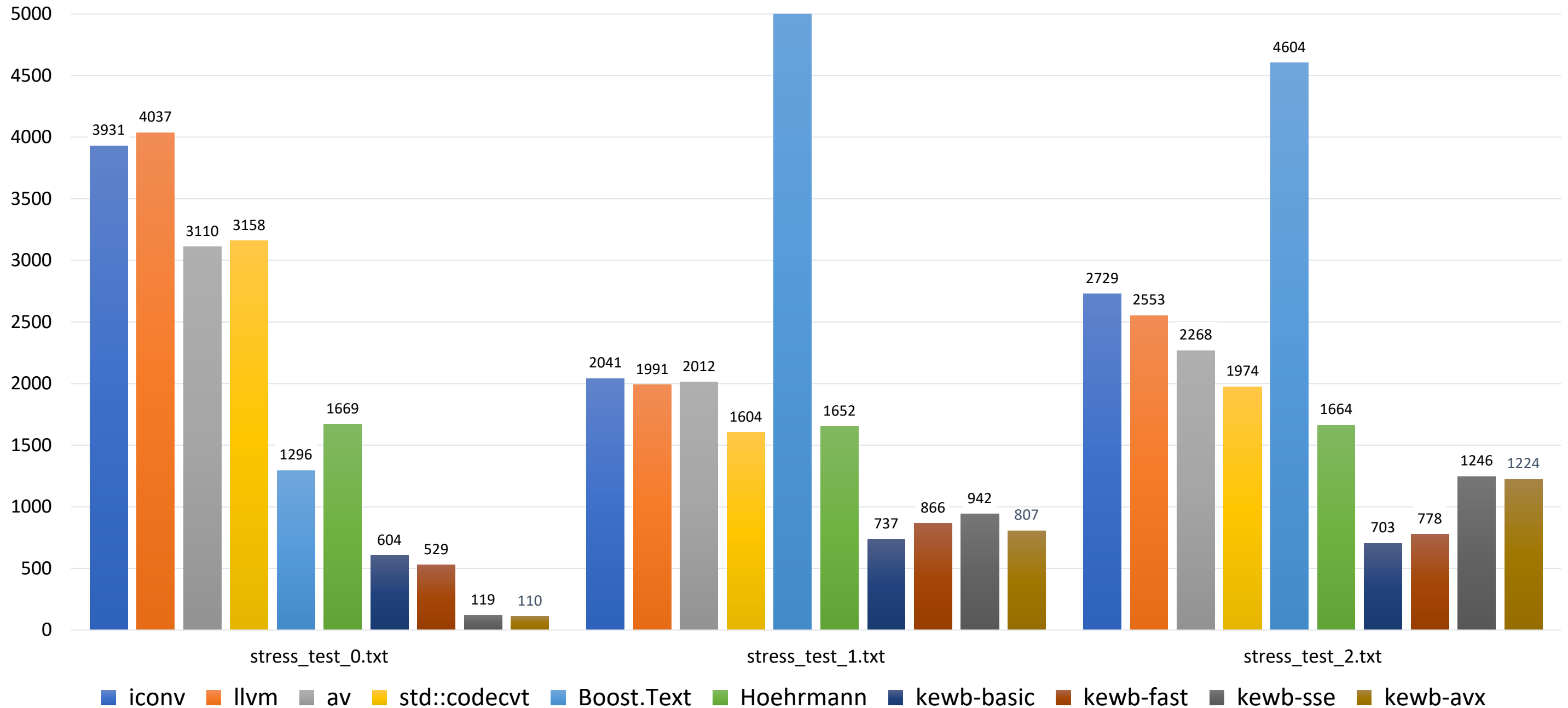
Conversion Time (msec)



Conversion Time (msec)



Conversion Time (msec)



Thank You for Attending!

Talk: github.com/BobSteagall/CppCon2020

Blog: bobsteagall.com