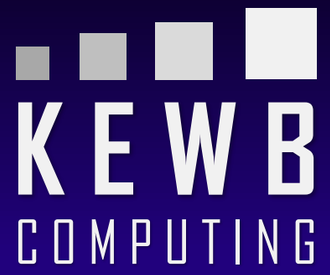
Adventures in SIMD Thinking (Part 1 of 2)

Bob Steagall

CppCon 2020



Agenda

KEWB COMPUTING

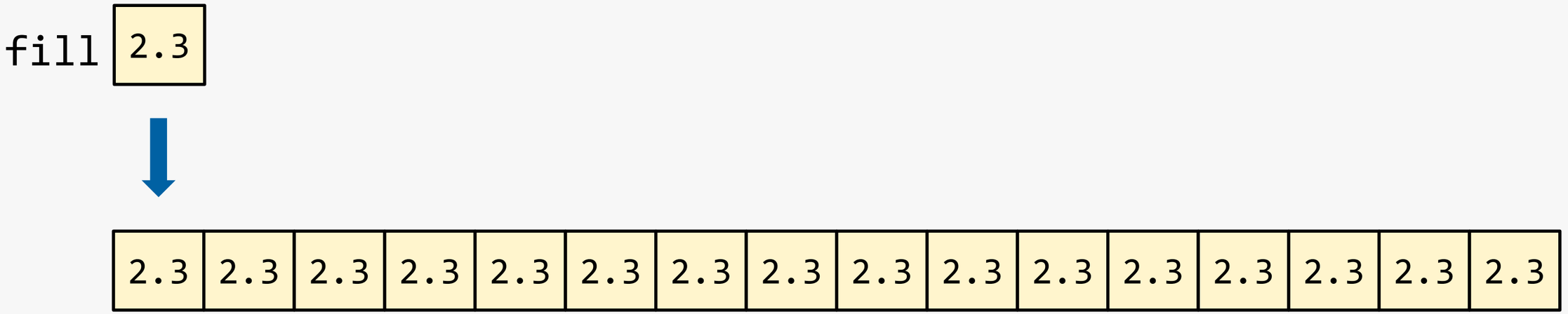
- Learn a little about Intel's SIMD facilities (disclaimer: I don't work for Intel)
- Create some useful functions in terms of AVX-512 intrinsics
- Try some SIMD-style thinking to tackle some interesting problems
  - Intra-register sorting
  - Fast linear median-of-seven filter
  - Fast small-kernel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

Getting Started

```
#include <cstdio>
#include <cstdint>
#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;

...

}
```

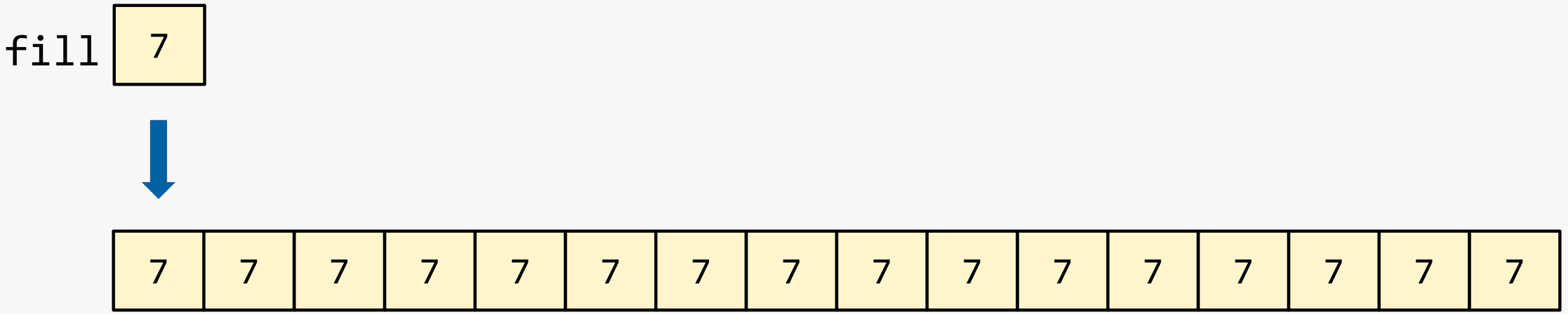
Function load_value()

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



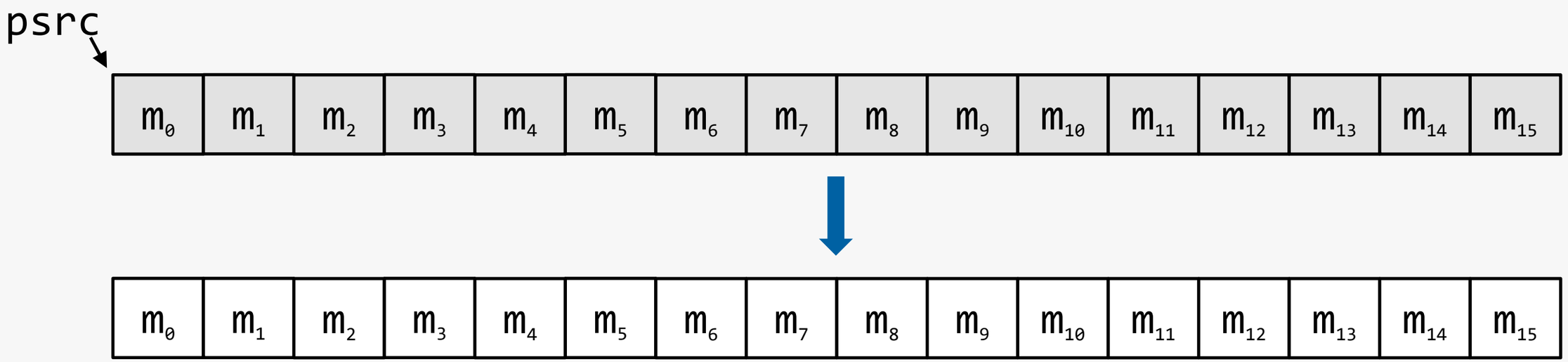
Function load_value()

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



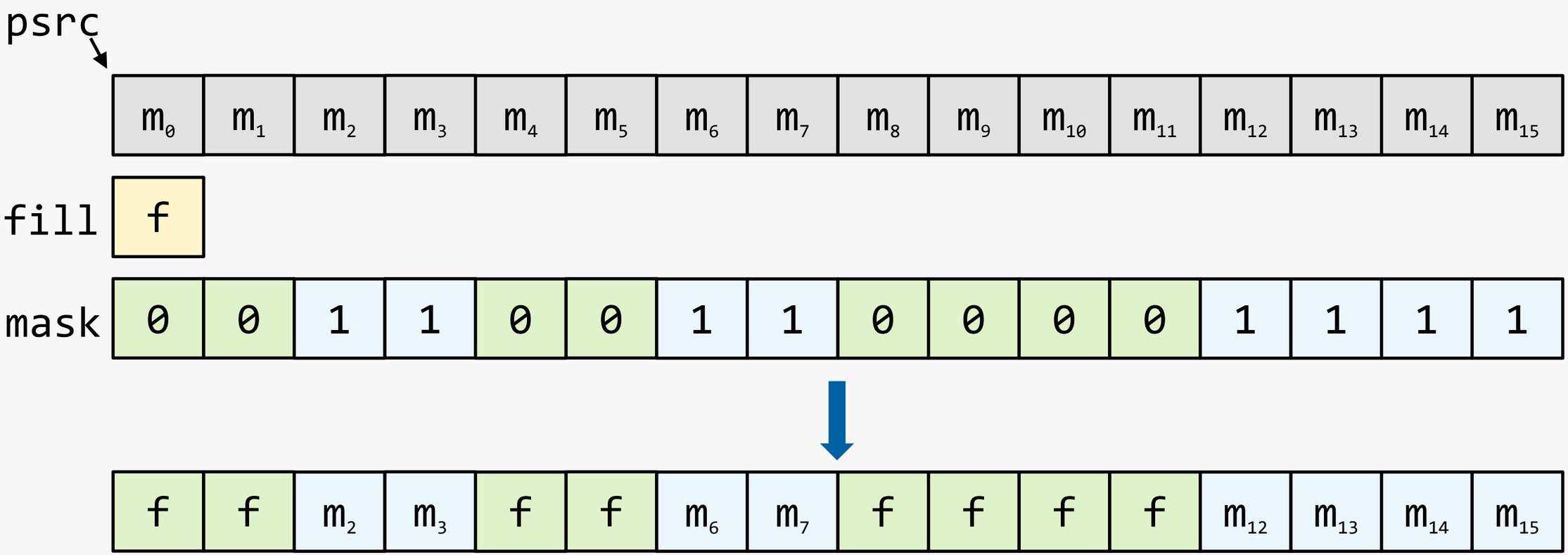
Function load_from()

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



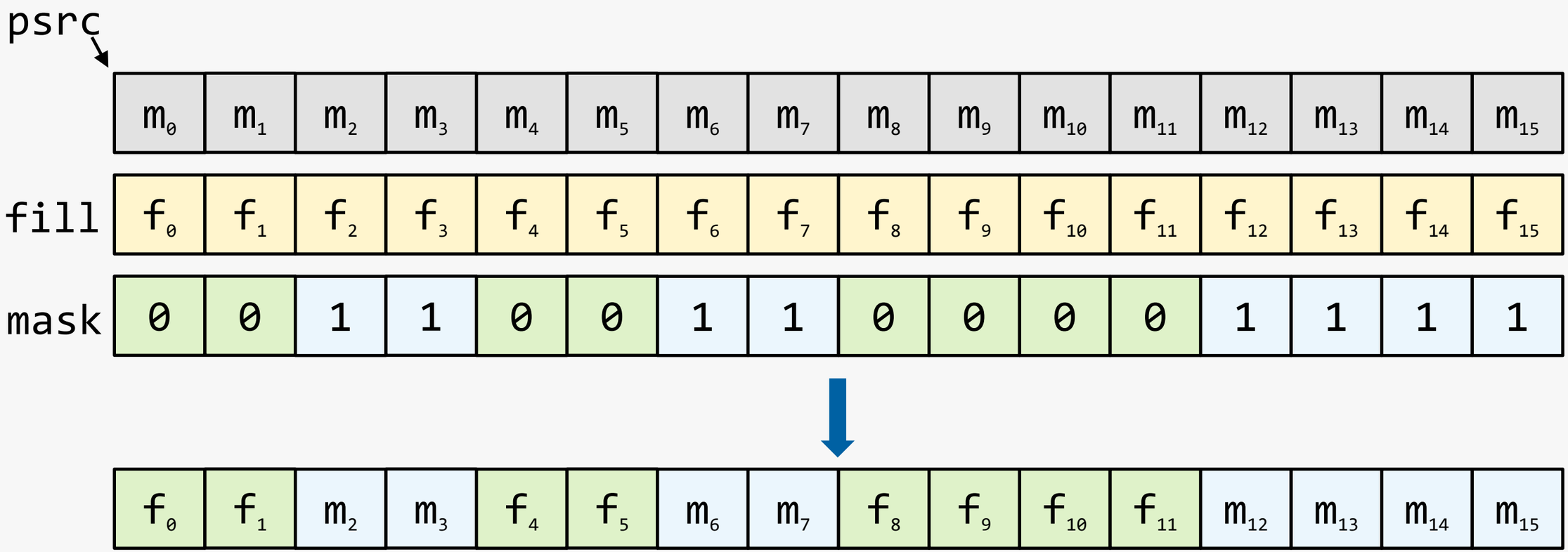
Function masked_load_from()

```
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);
}
```



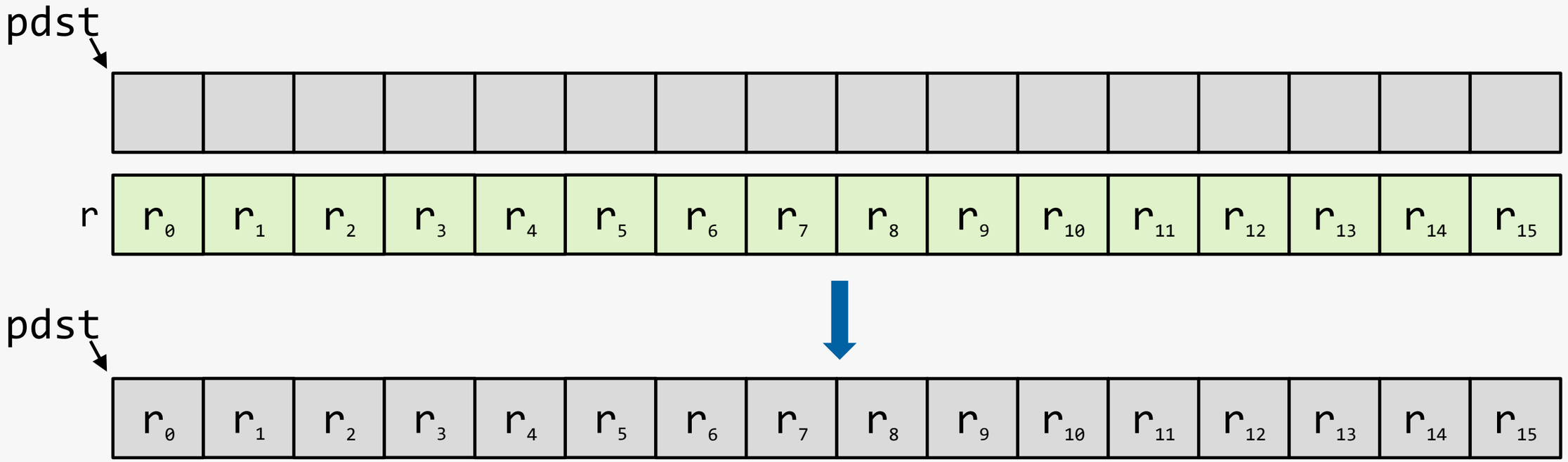
Function masked_load_from()

```
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);
}
```



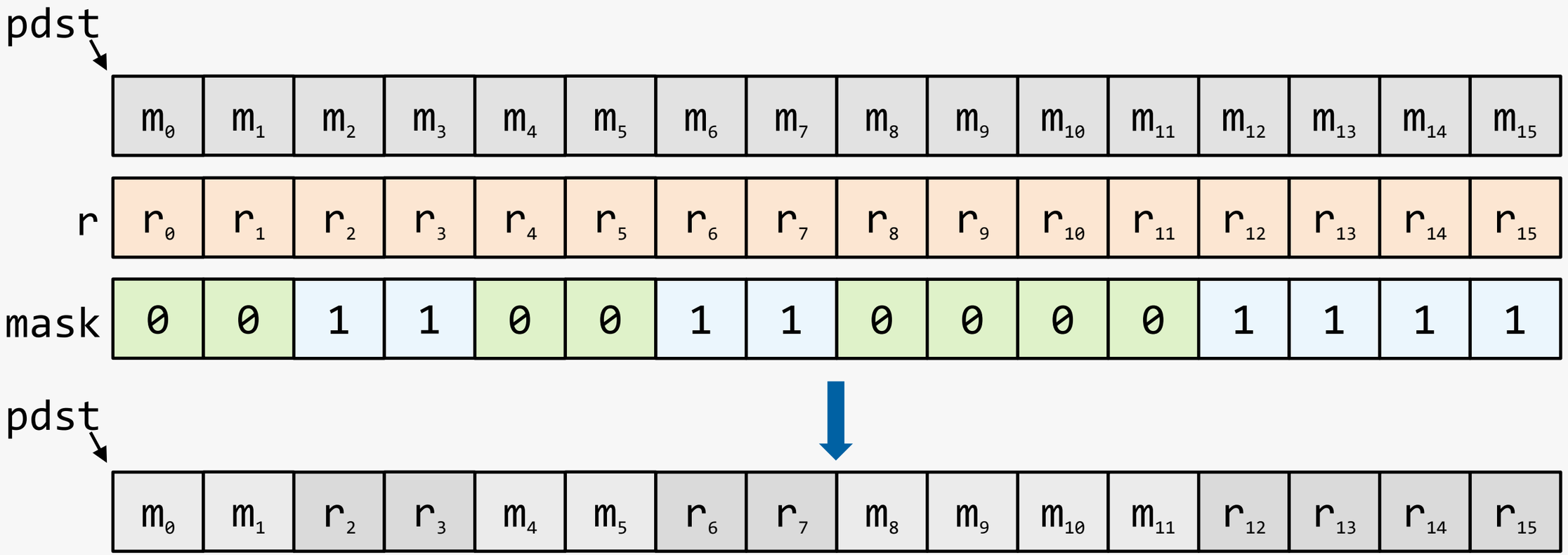
Function store_to()

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



Function masked_store_to()

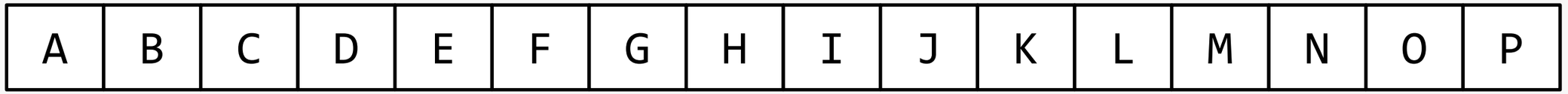
```
KEWB_FORCE_INLINE void
masked_store_to(float* pdst, rf_512 r, msk_512 mask)
{
    _mm512_mask_storeu_ps(pdst, (__mmask16) mask, r);
}
```



Function make_bit_mask()

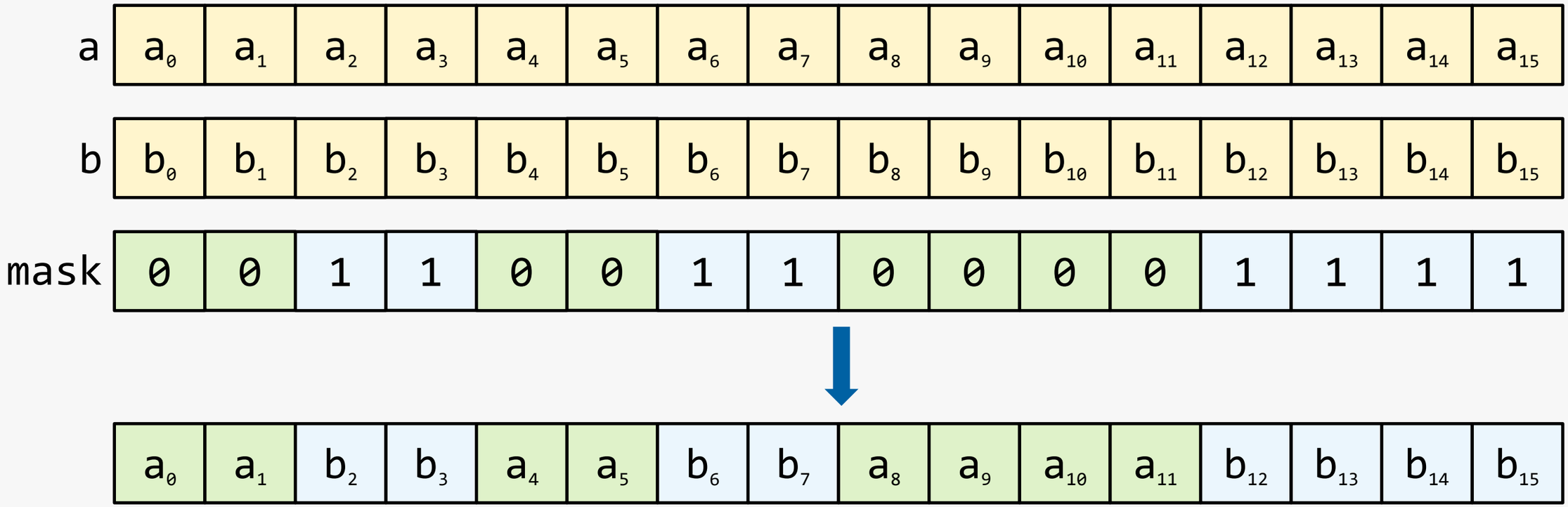
```
template<unsigned A=0, unsigned B=0, unsigned C=0, unsigned D=0,
         unsigned E=0, unsigned F=0, unsigned G=0, unsigned H=0,
         unsigned I=0, unsigned J=0, unsigned K=0, unsigned L=0,
         unsigned M=0, unsigned N=0, unsigned O=0, unsigned P=0>
KEWB_FORCE_INLINE constexpr uint32_t
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



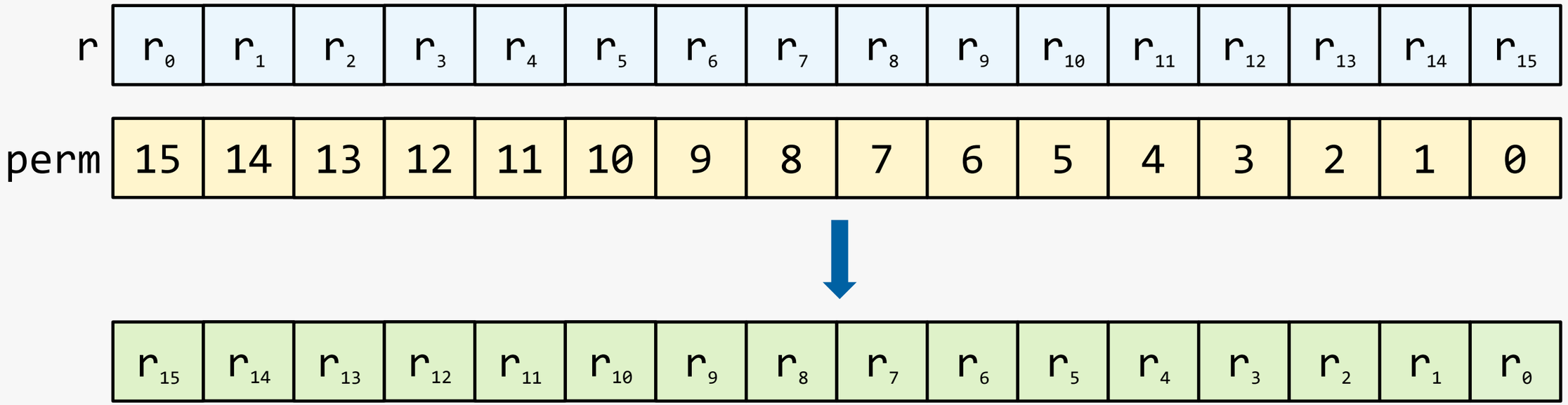
Function blend()

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



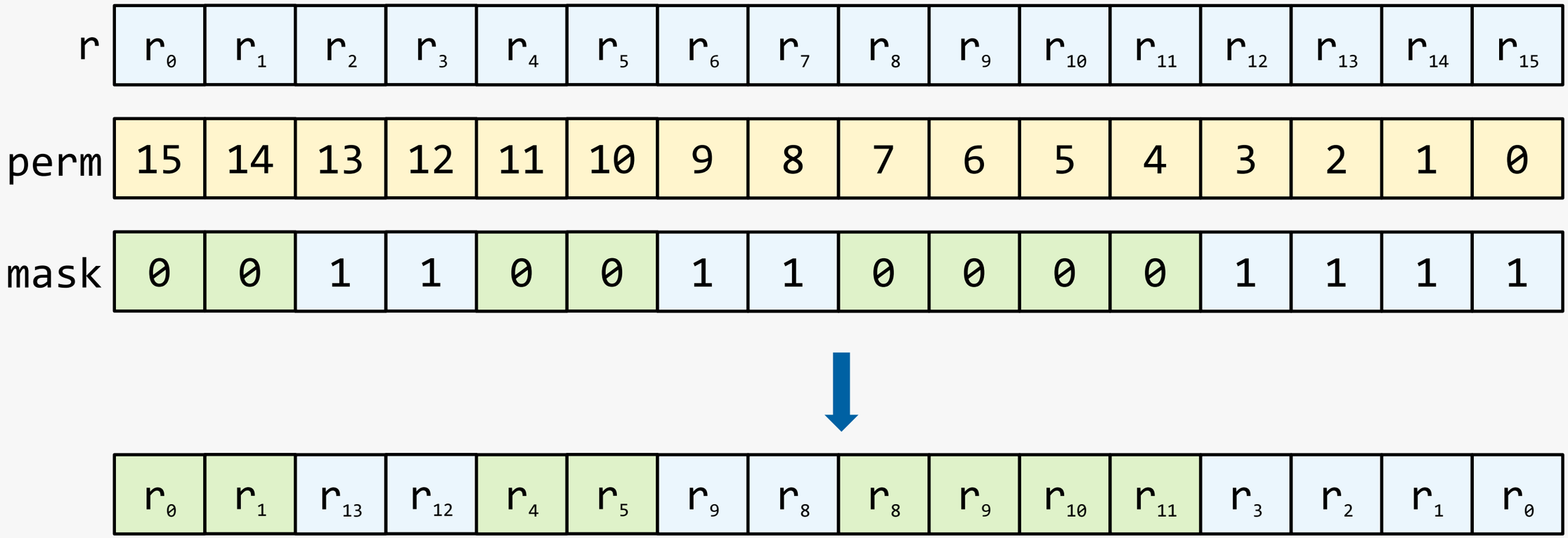
Function permute()

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



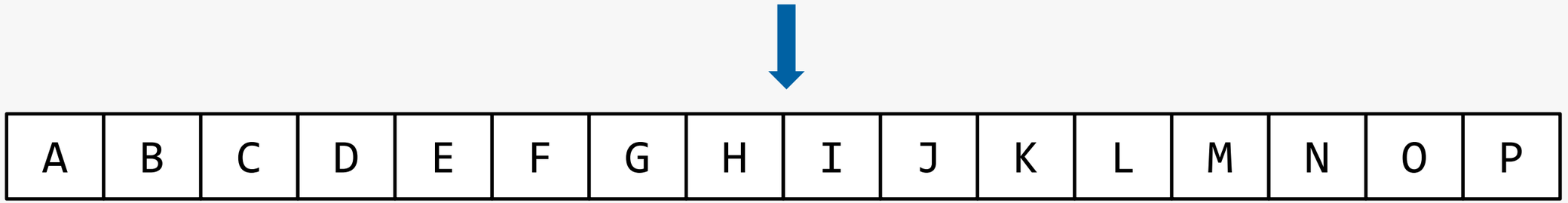
Function masked_permute()

```
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);
}
```



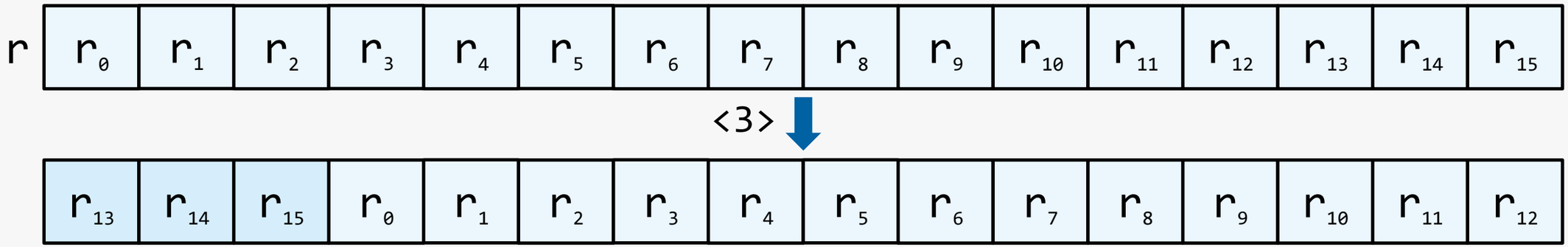
Function make_perm_map()

```
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);
}
```



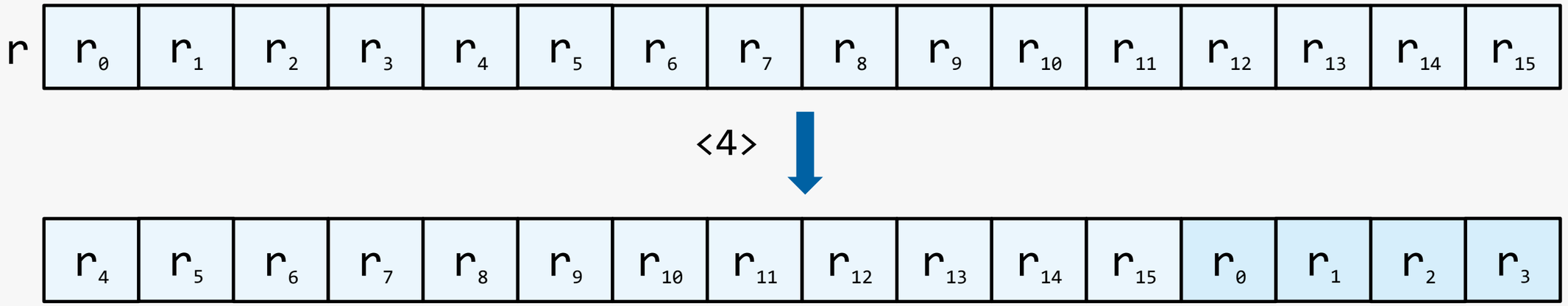
Function rotate()

```
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);
    }
}
```



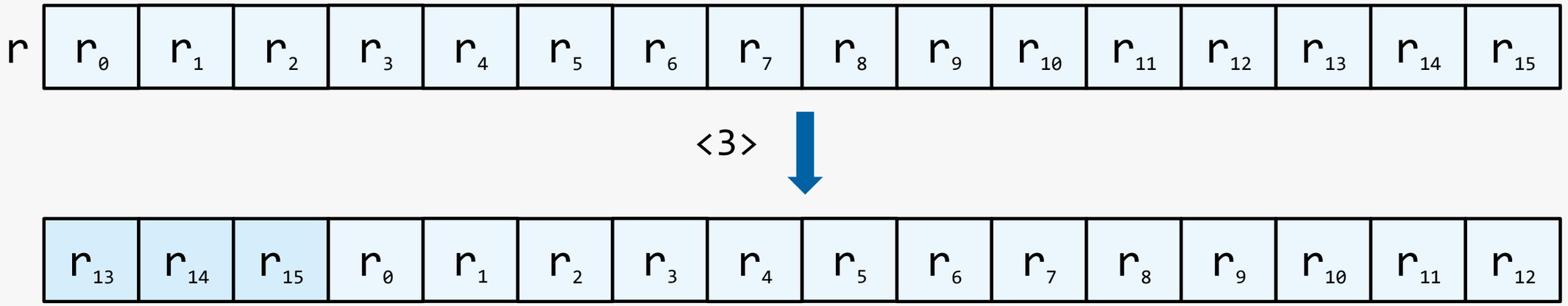
Function rotate_down()

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



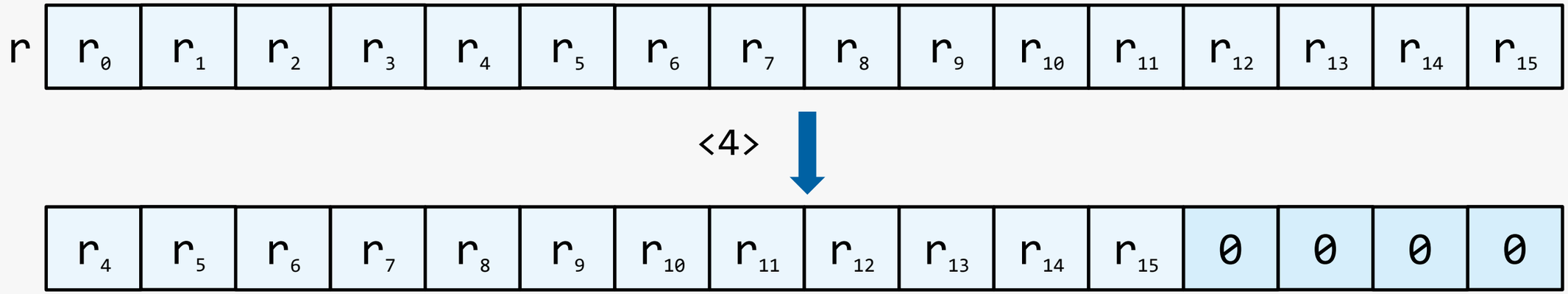
Function rotate_up()

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



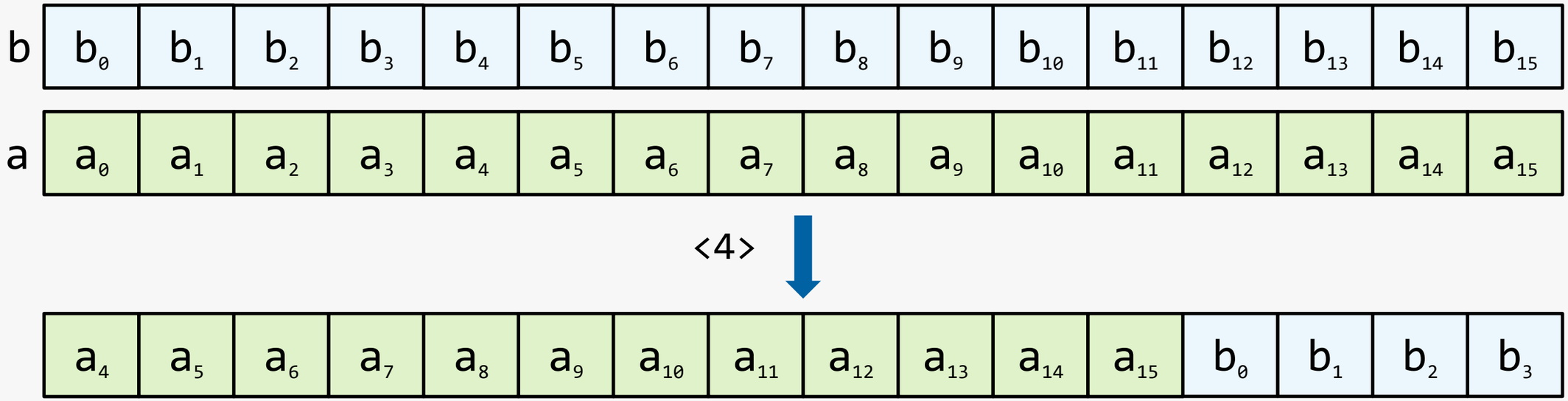
Function shift_down()

```
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>());
}
```



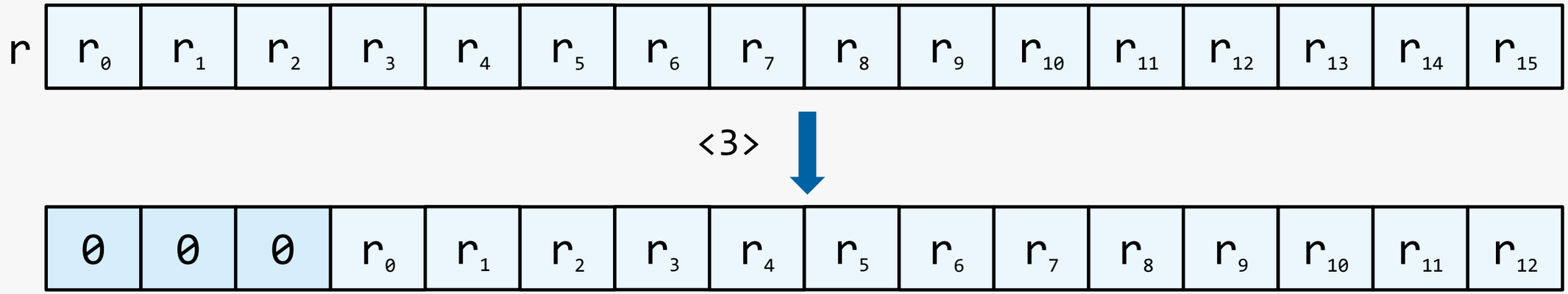
Function shift_down_with_carry()

```
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>());
}
```



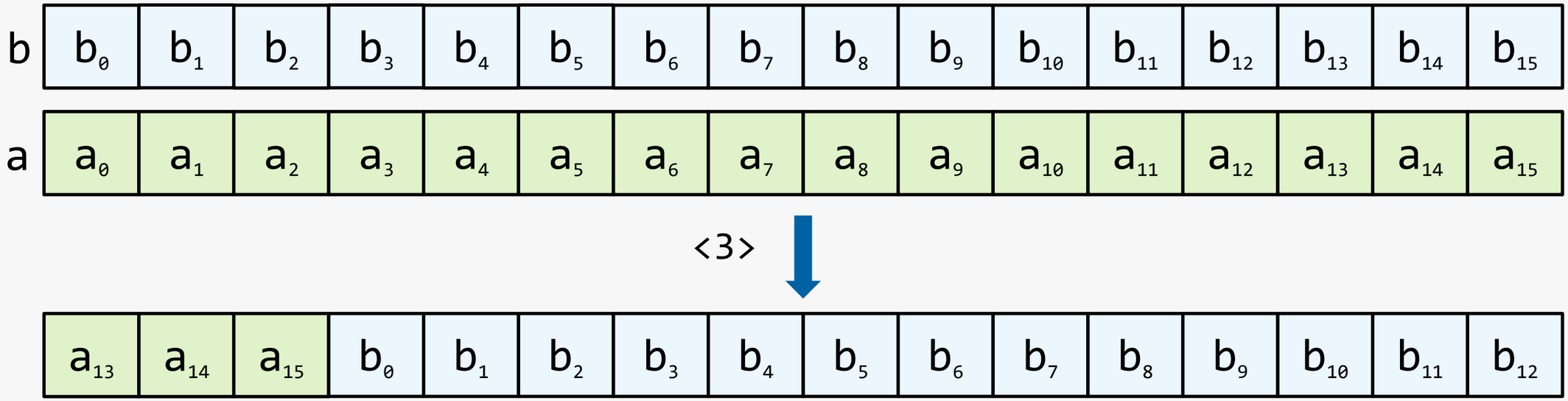
Function shift_up()

```
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>());
}
```



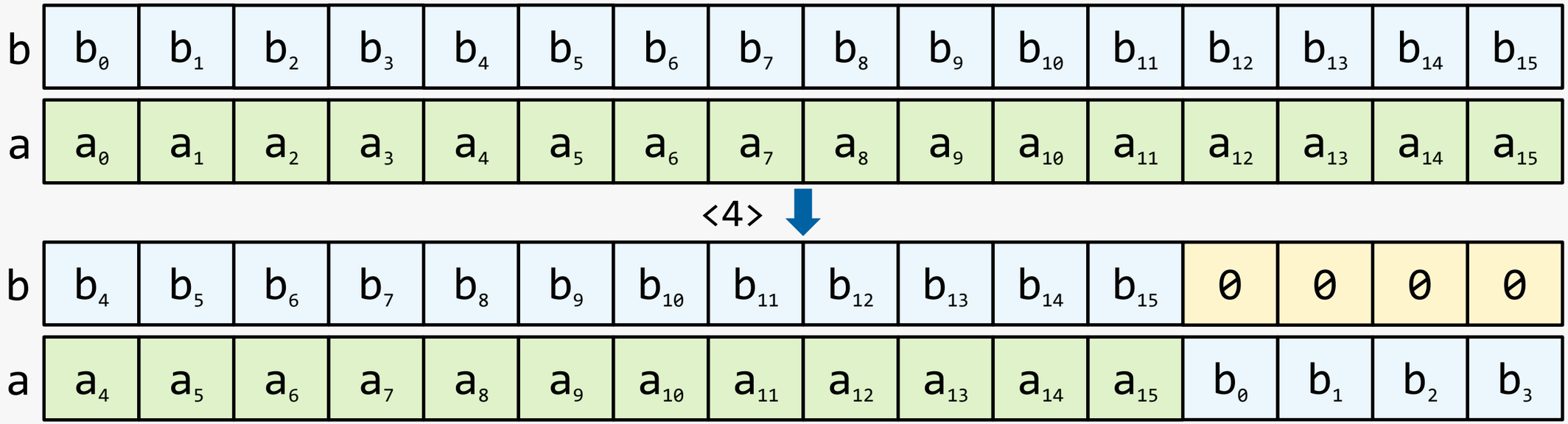
Function shift_up_with_carry()

```
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>());
}
```



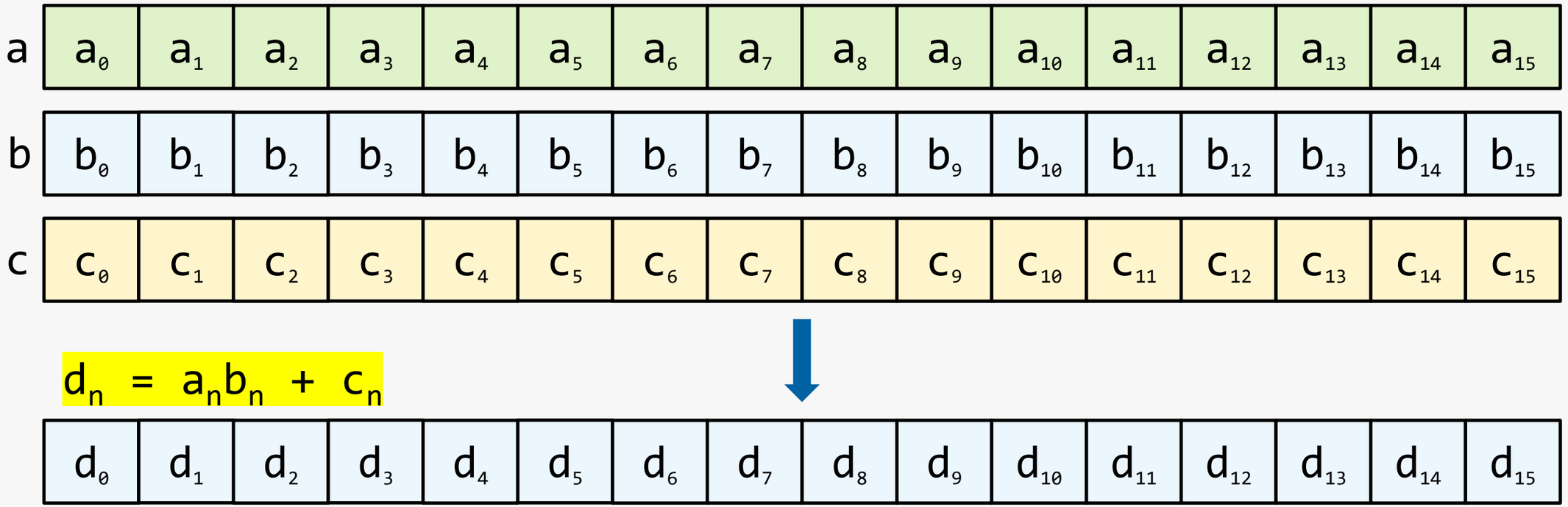
Function in_place_shift_down_with_carry()

```
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);
}
```



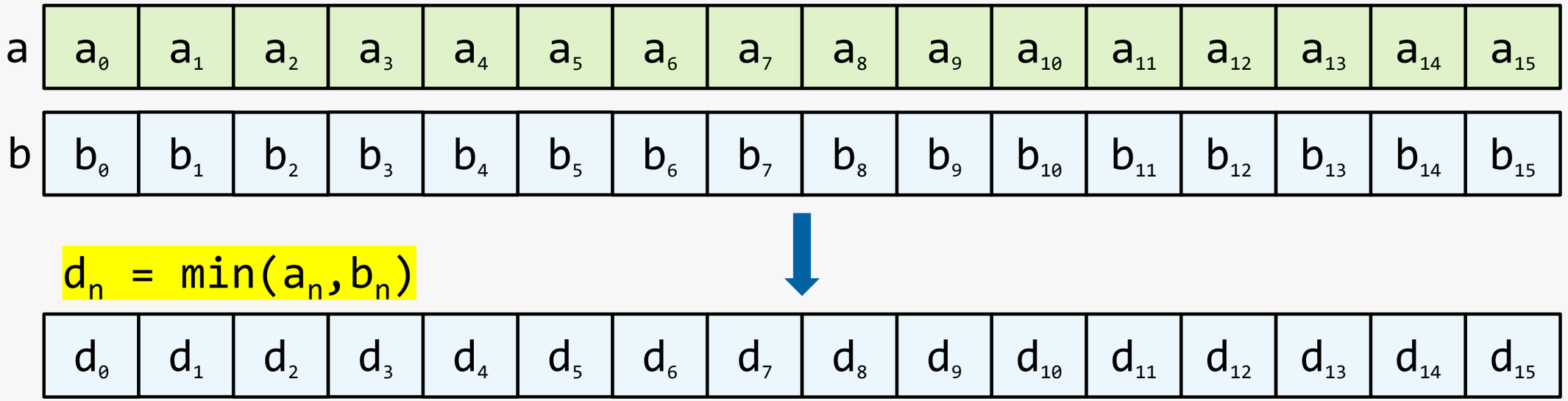
Function fused_multiply_add()

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



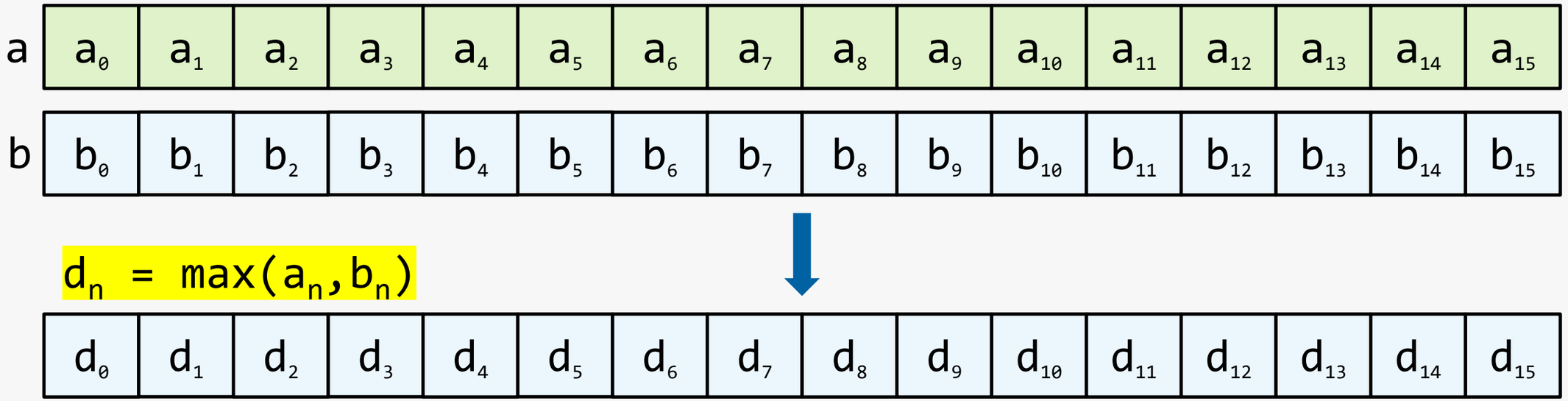
Function minimum()

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



Function maximum()

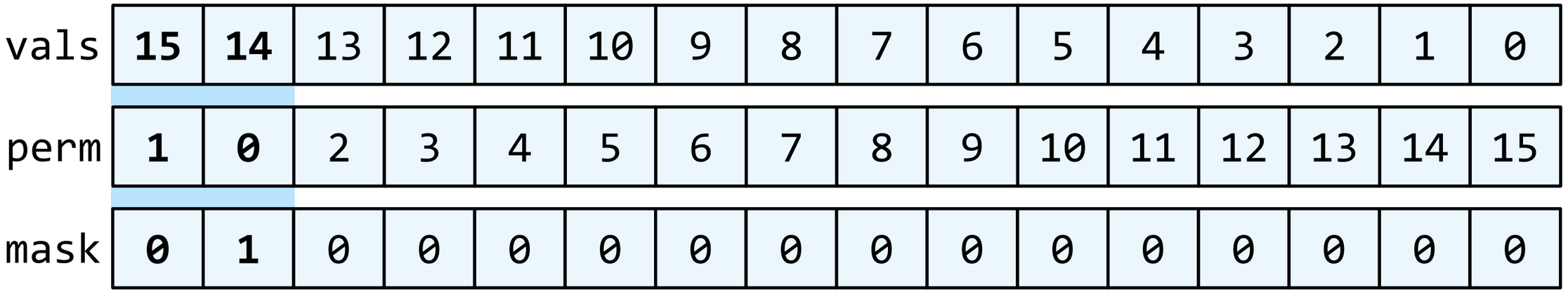
```
KEWB_FORCE_INLINE rf_512
maximum(rf_512 a, rf_512 a)
{
    return _mm512_max_ps(a, b);
}
```



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()

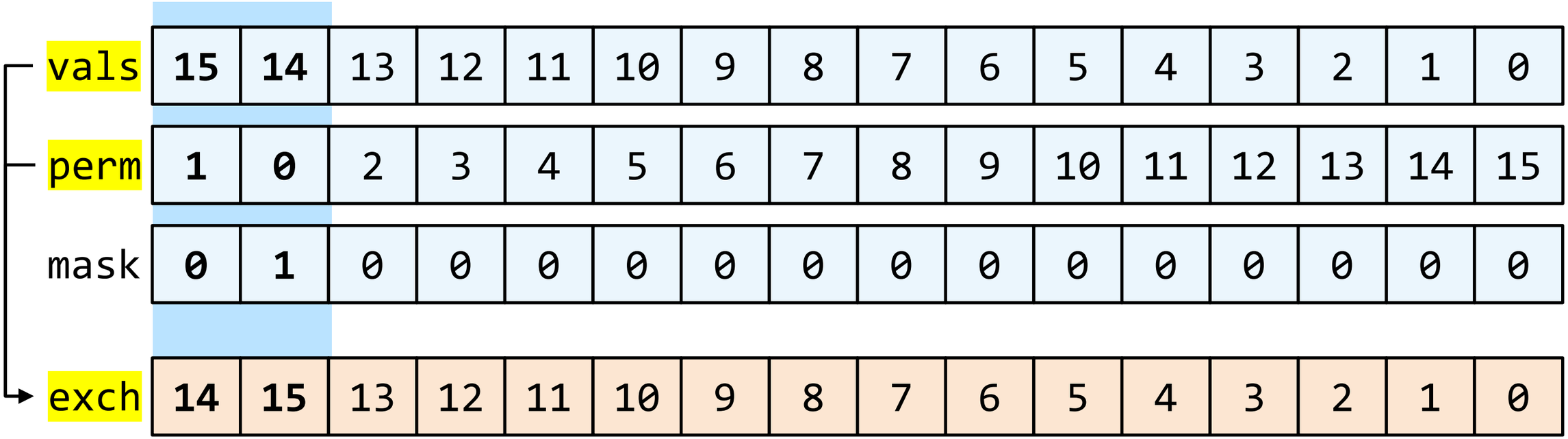


vals	**15**	**14**	13	12	11	10	9	8	7	6	5	4	3	2	1	0
perm	**1**	**0**	2	3	4	5	6	7	8	9	10	11	12	13	14	15
mask	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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);
}
```

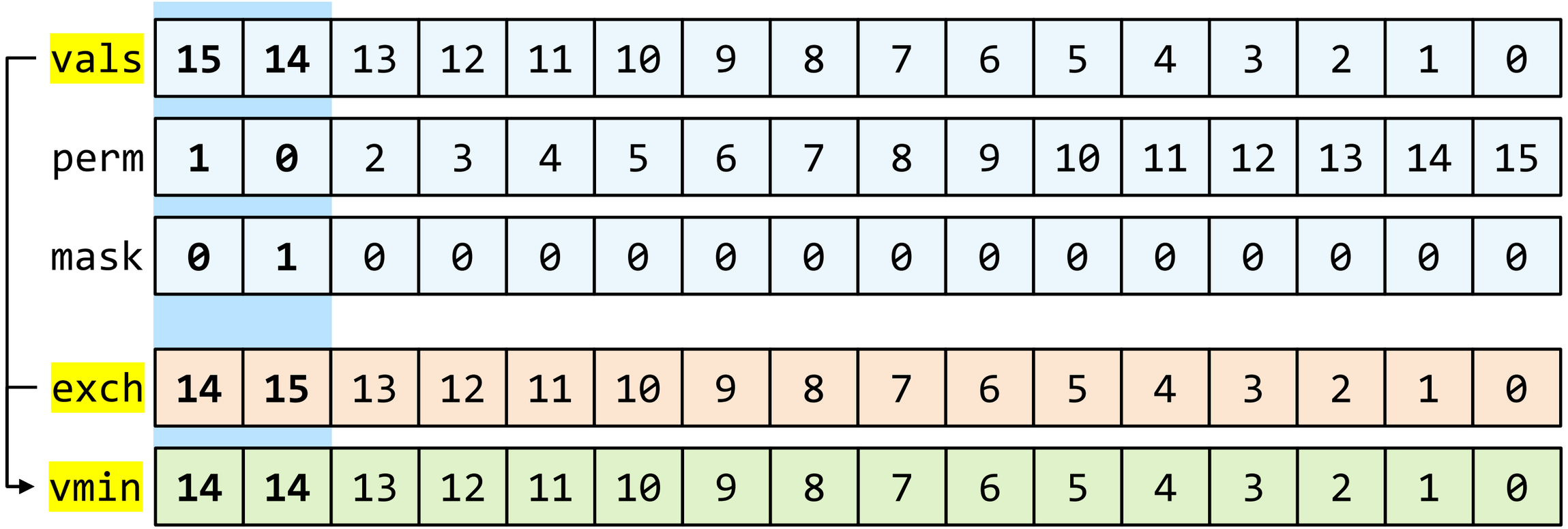
Function compare_with_exchange()



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);
}
```

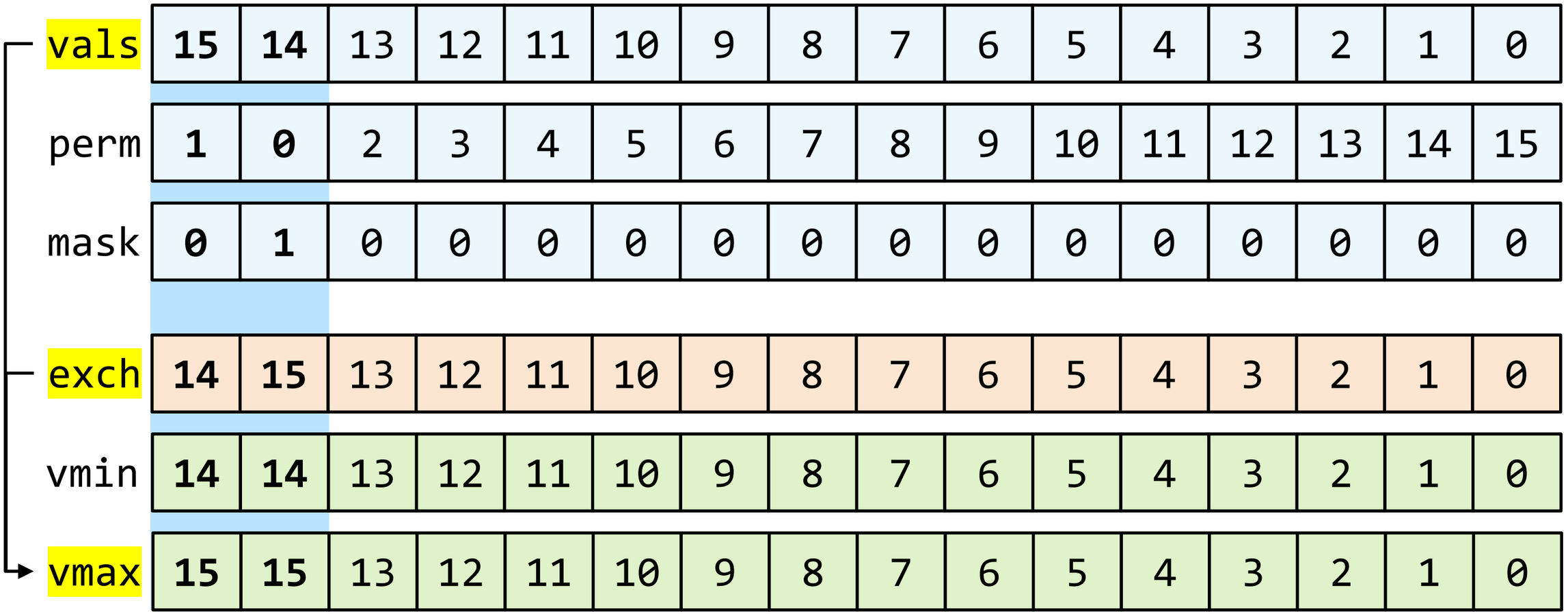
Function compare_with_exchange()



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);
}
```

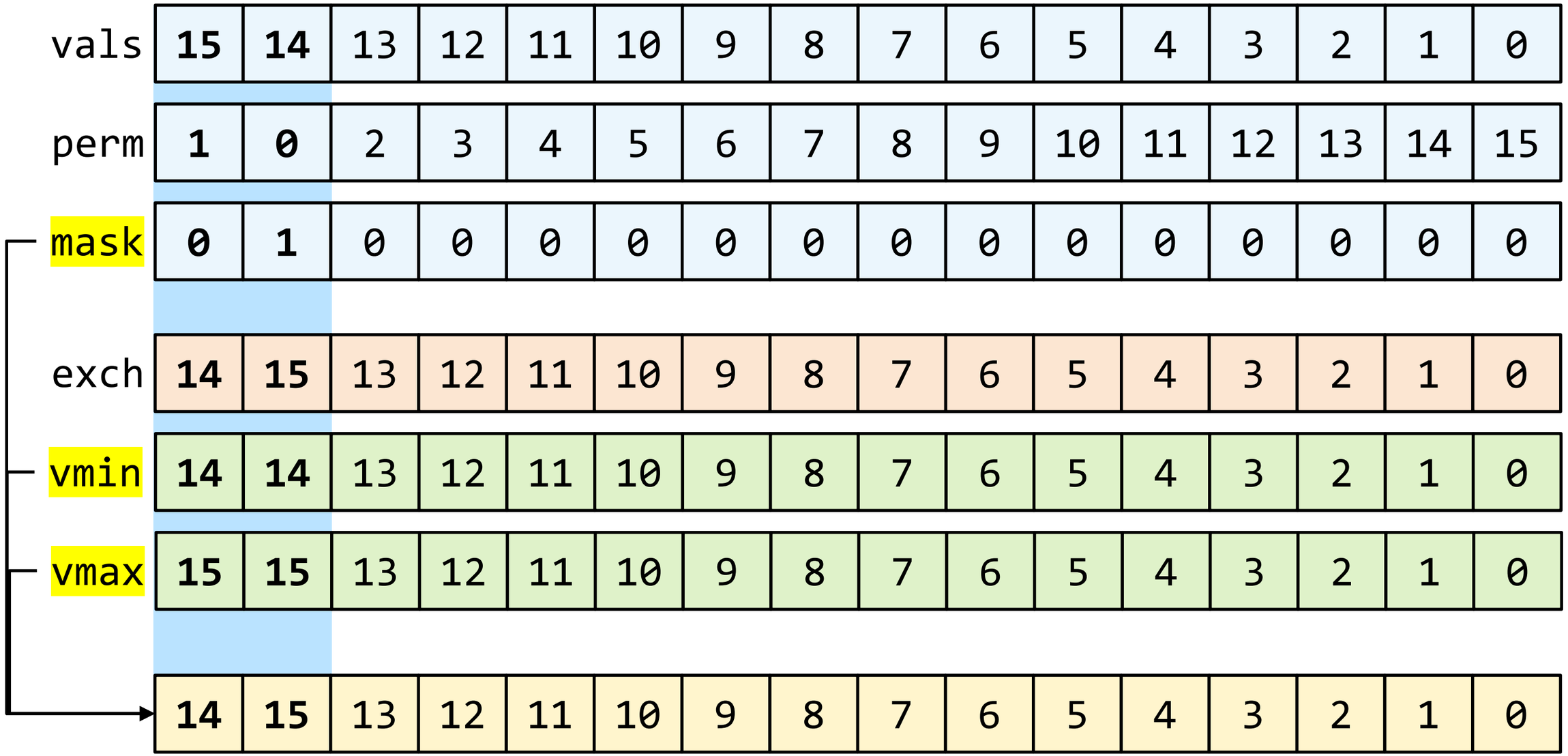
Function compare_with_exchange()



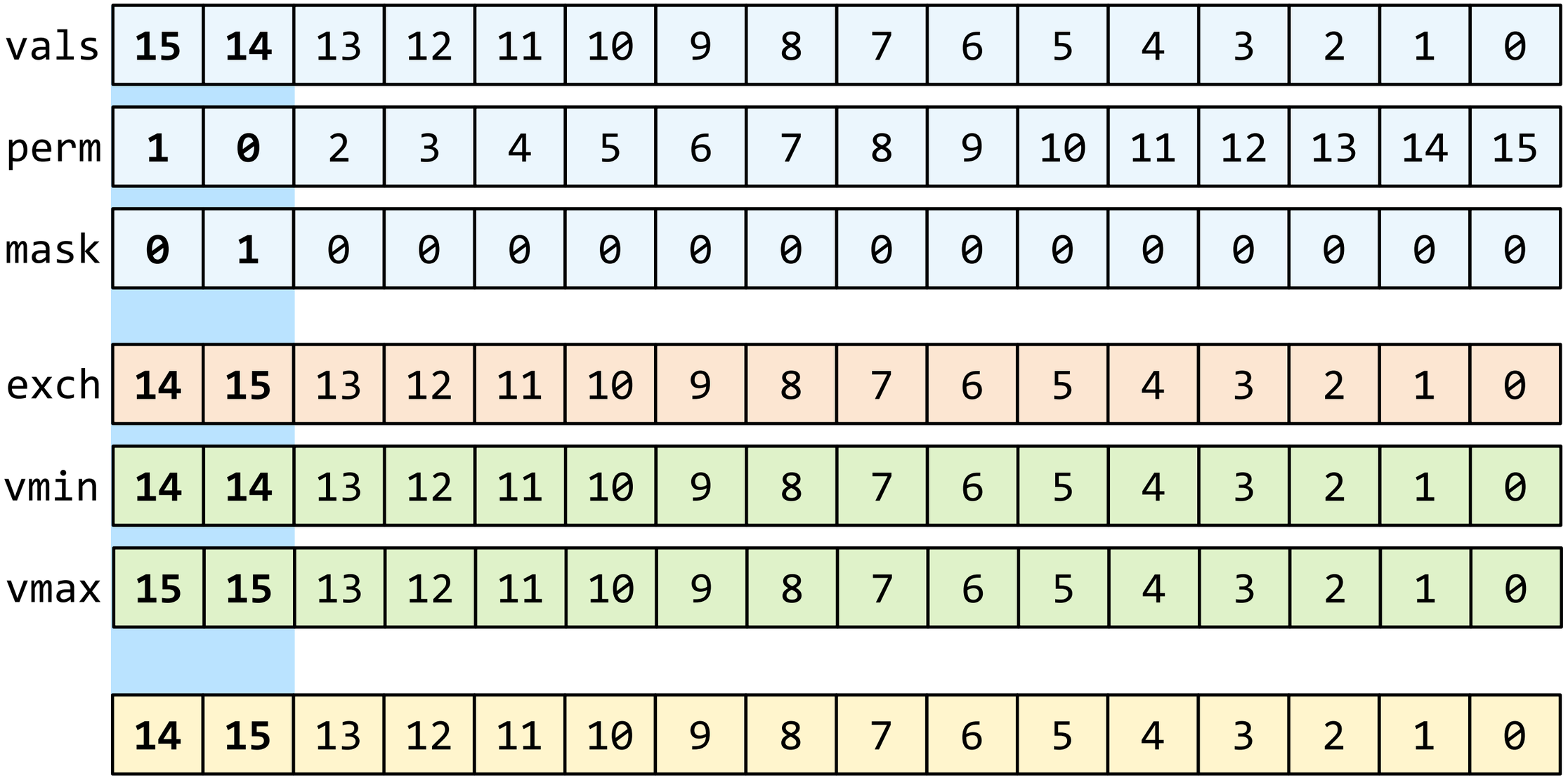
Function compare_with_exchange()

```
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()

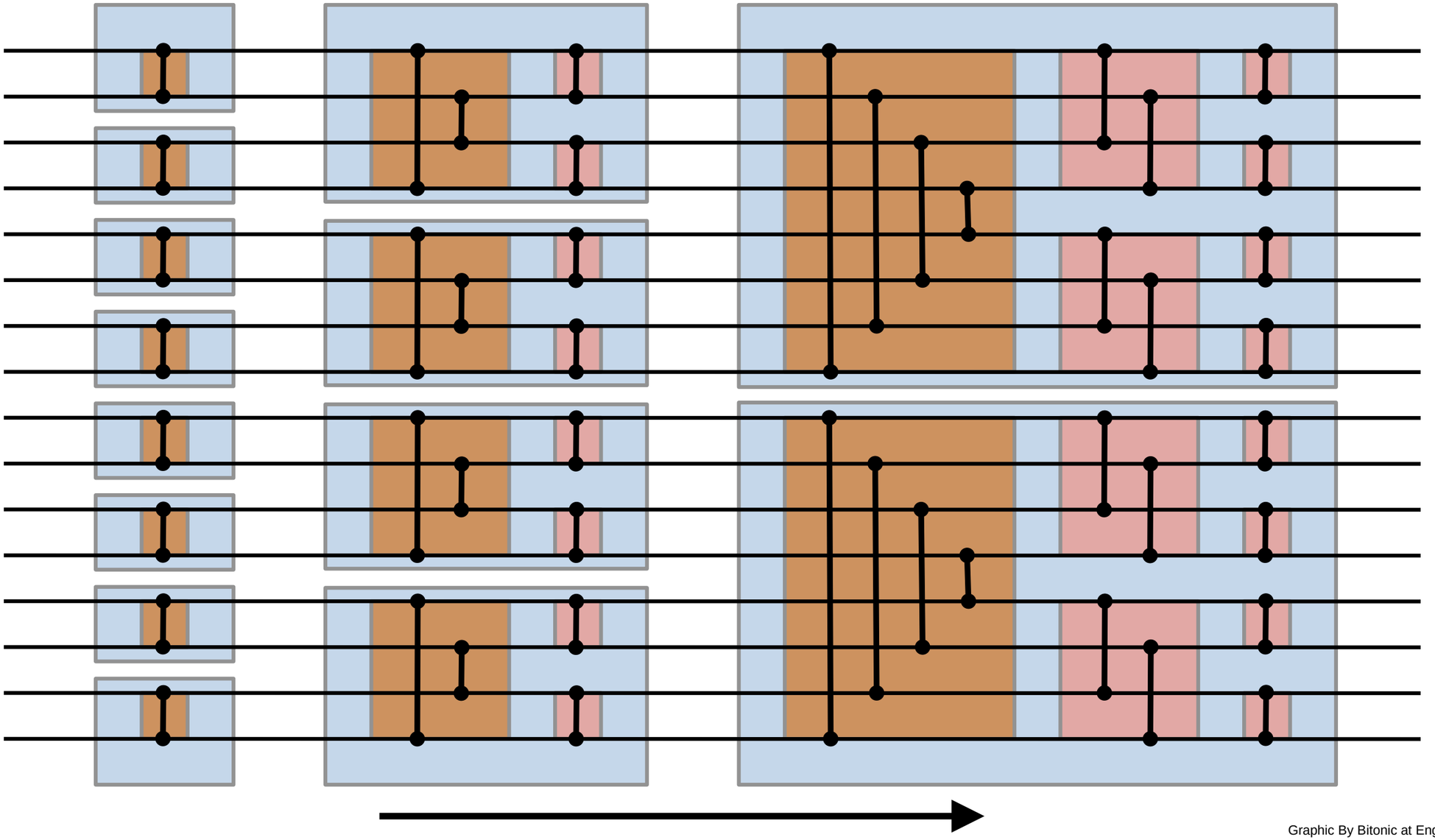


Function compare_with_exchange()



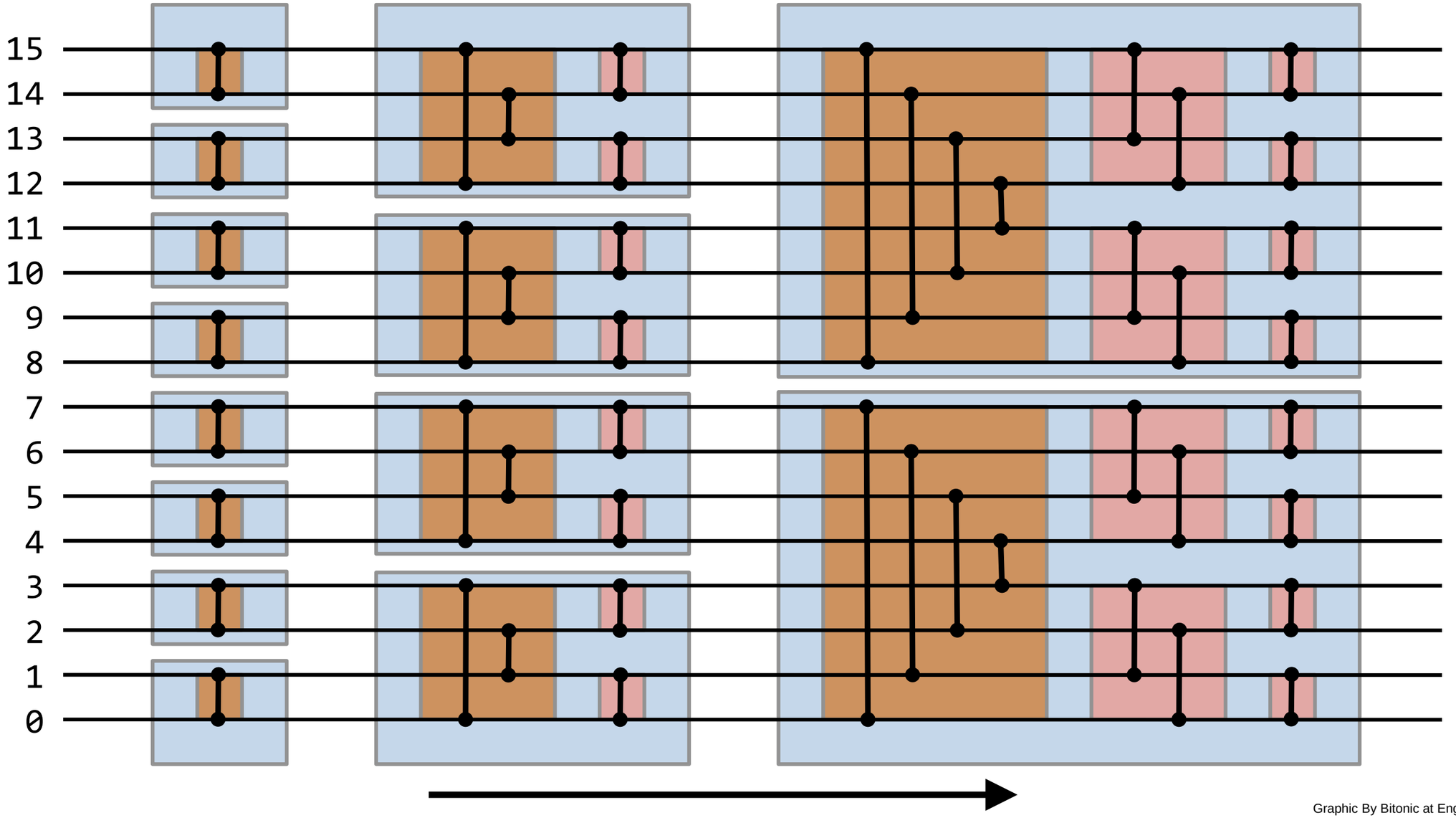
Intra-Register Sorting

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



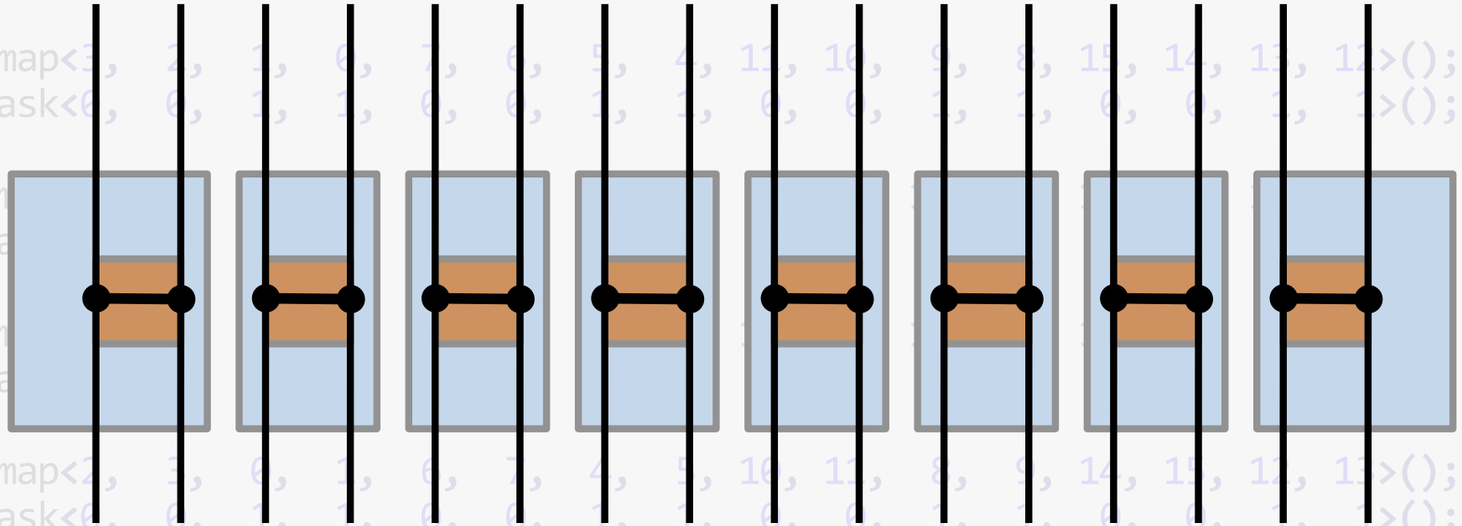
Graphic By Bitonic at English Wikipedia - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=21961917

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.
    //                                      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<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>();
    ...
}
```

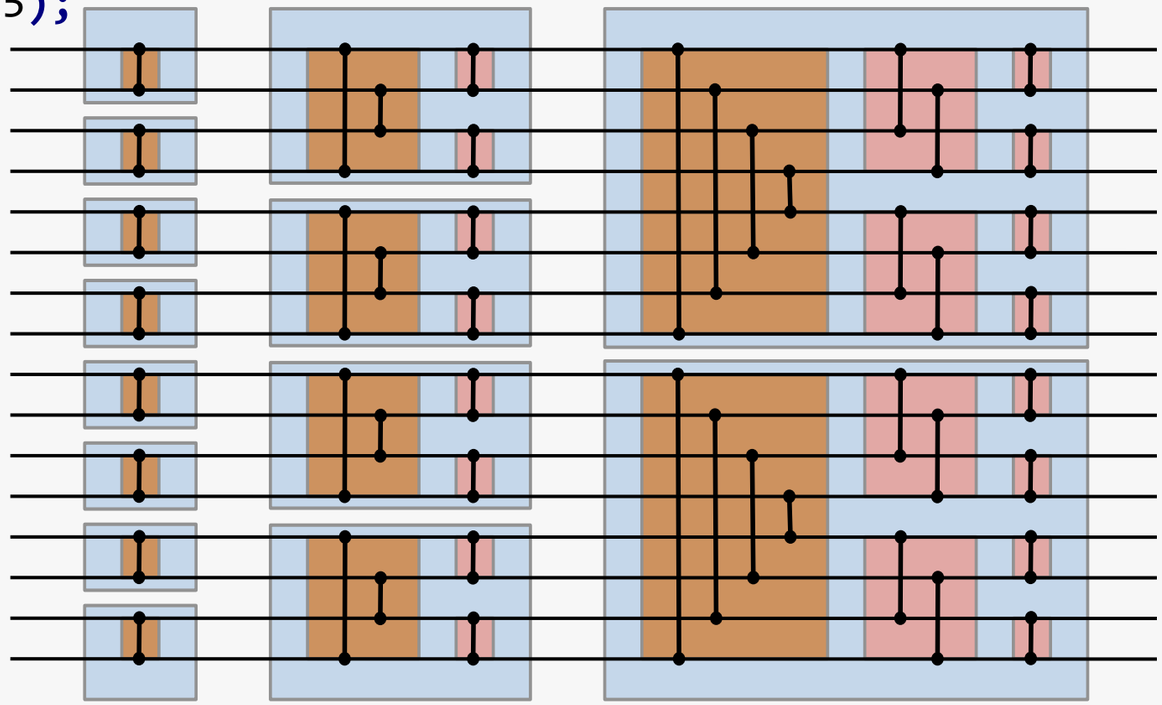
Function sort_two_lanes_of_8()

```
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_m
    constexpr msk_512 mask2 = make_bit_ma

    ri_512 const      perm3 = make_perm_m
    constexpr msk_512 mask3 = make_bit_ma

    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>();
    ...
}
```

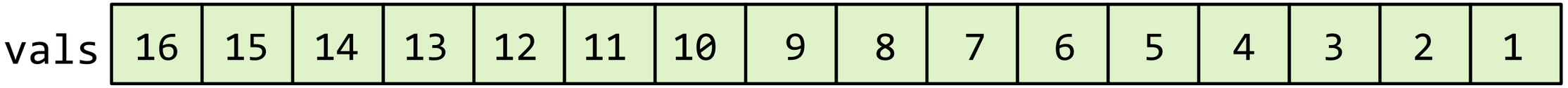


Function sort_two_lanes_of_8()

```
KEWB_FORCE_INLINE rf_512
sort_two_lanes_of_8(rf_512 vals)
{
    ...
    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;
}
```

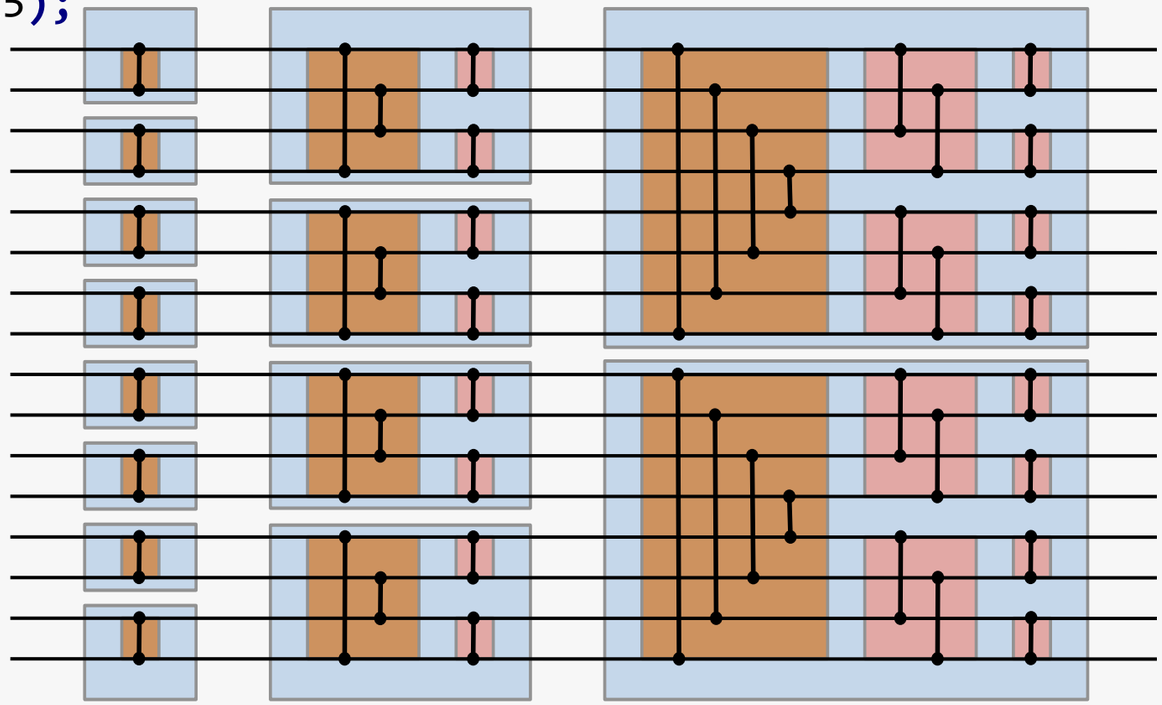


Function sort_two_lanes_of_8()

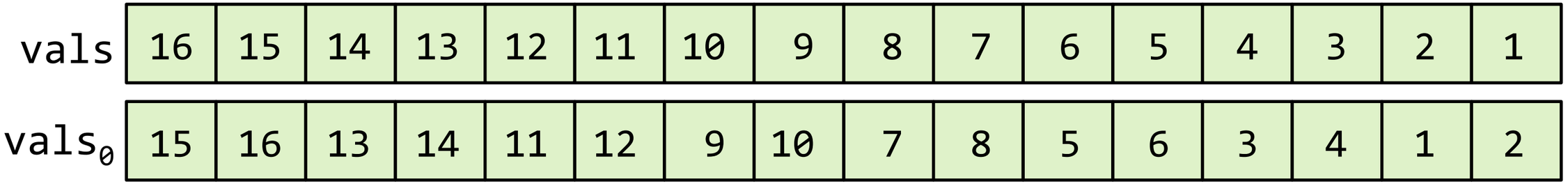


Function sort_two_lanes_of_8()

```
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;
}
```

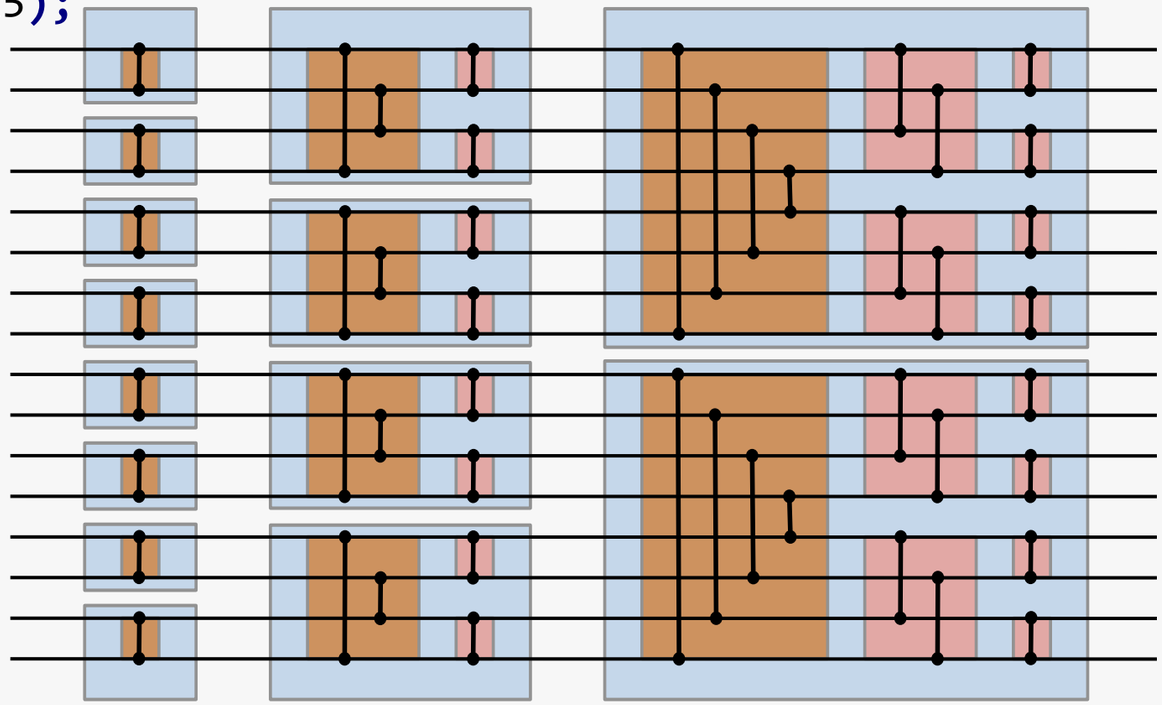


Function sort_two_lanes_of_8()

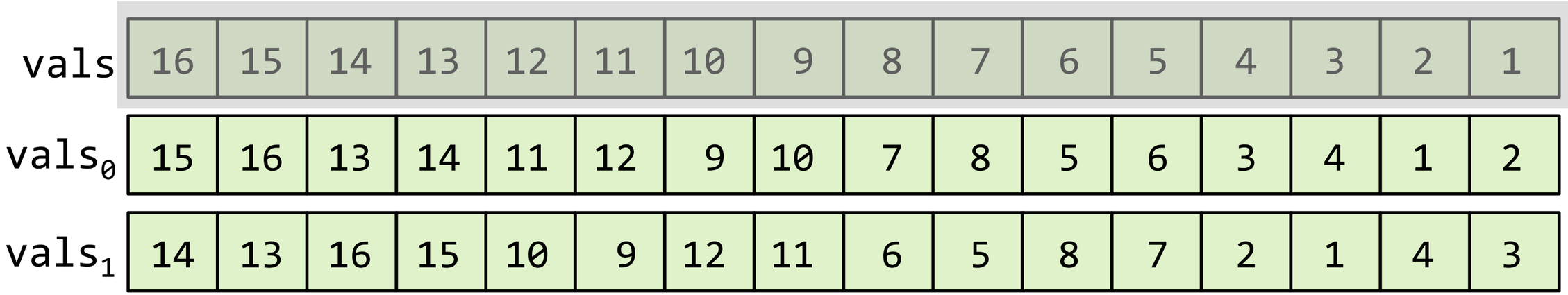


Function sort_two_lanes_of_8()

```
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;
}
```

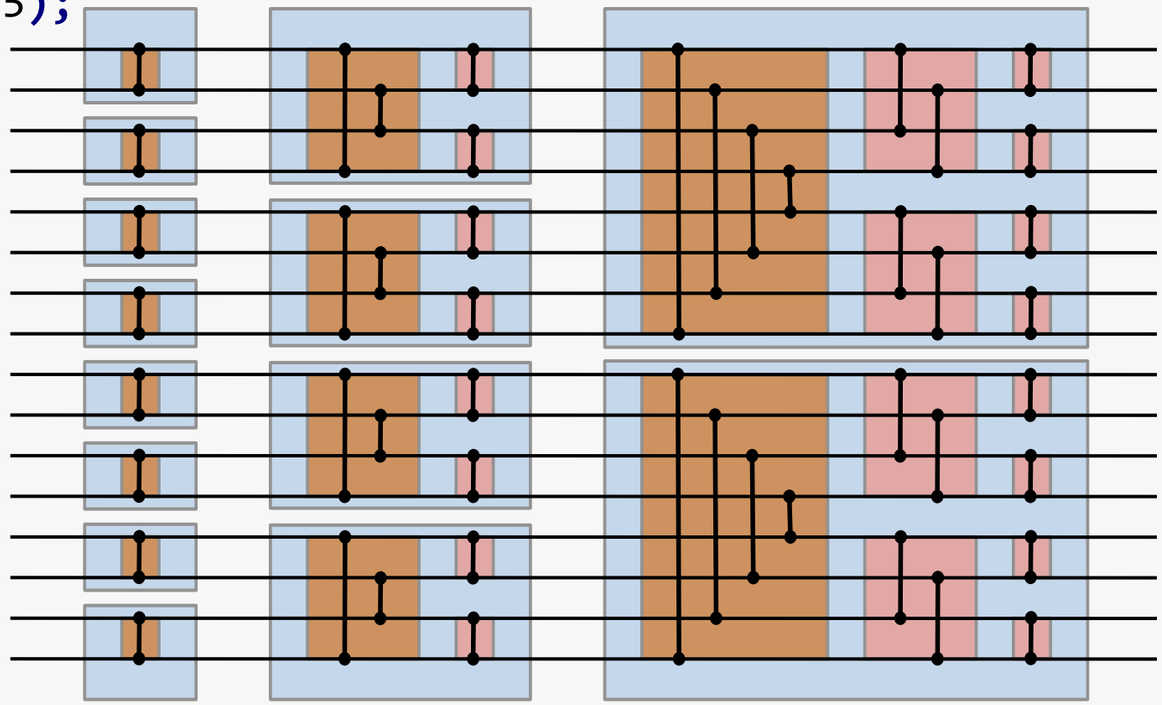


Function sort_two_lanes_of_8()

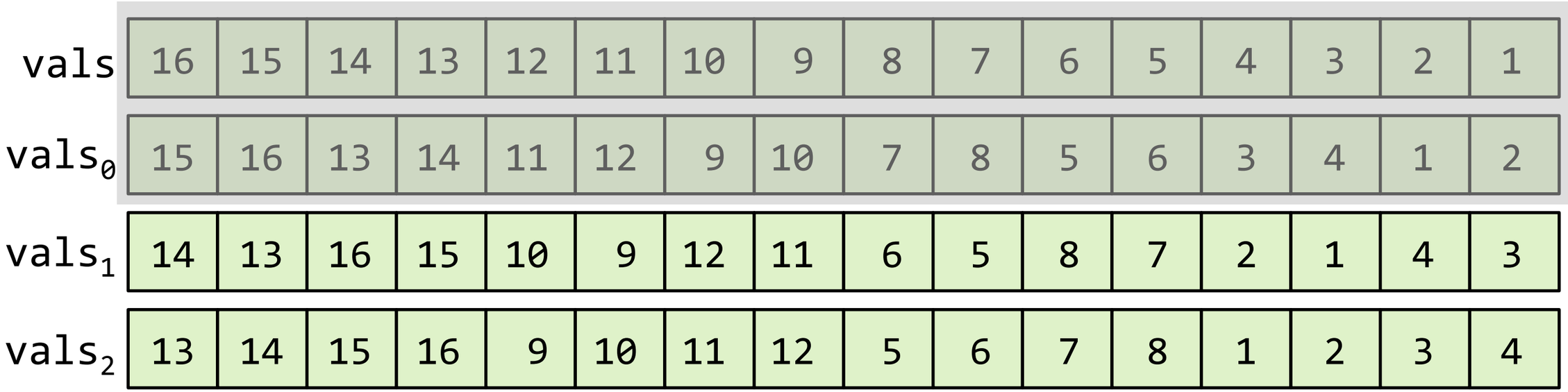


Function sort_two_lanes_of_8()

```
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;
}
```

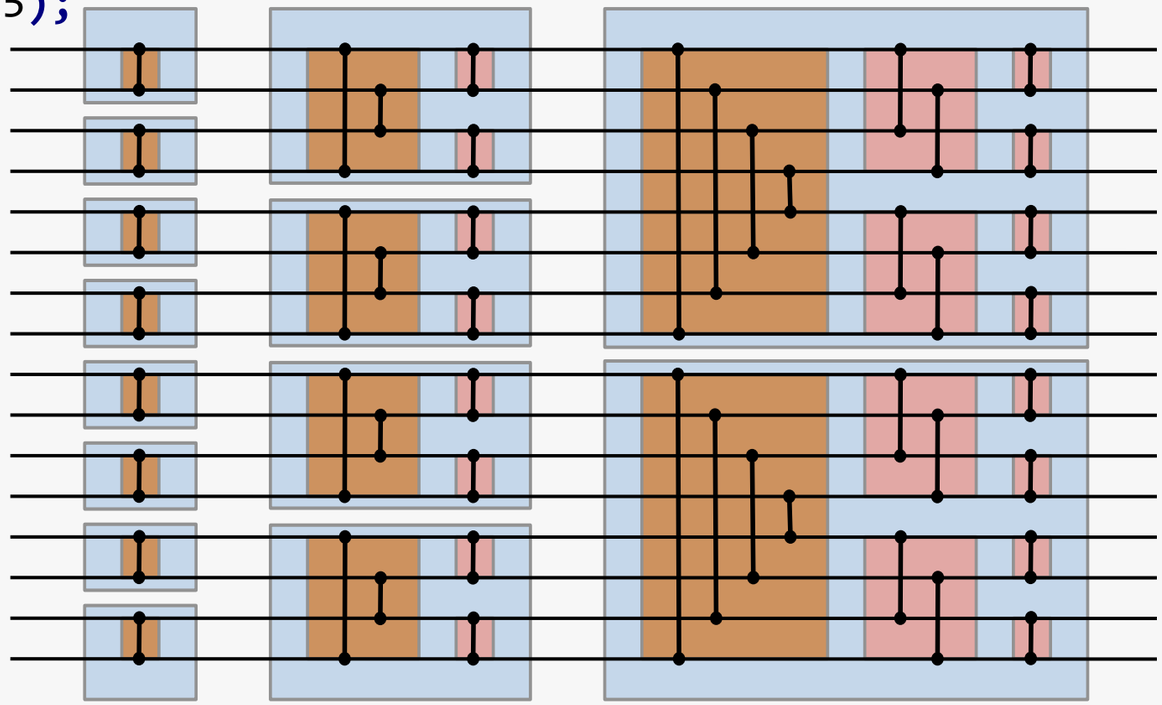


Function sort_two_lanes_of_8()

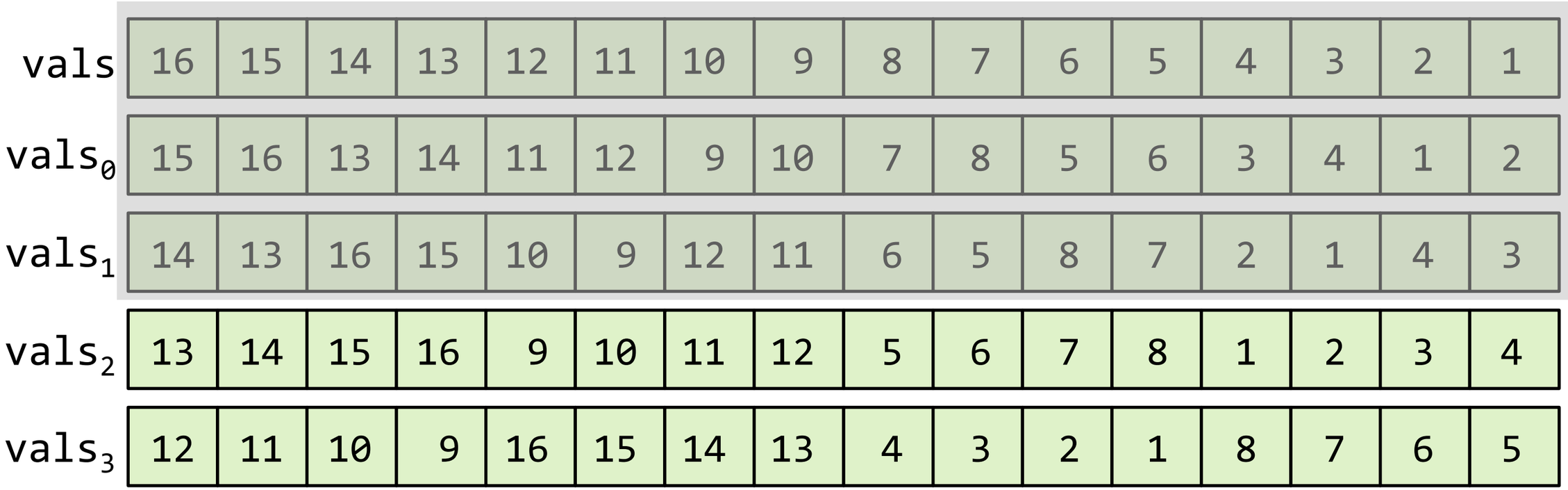


Function sort_two_lanes_of_8()

```
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;
}
```

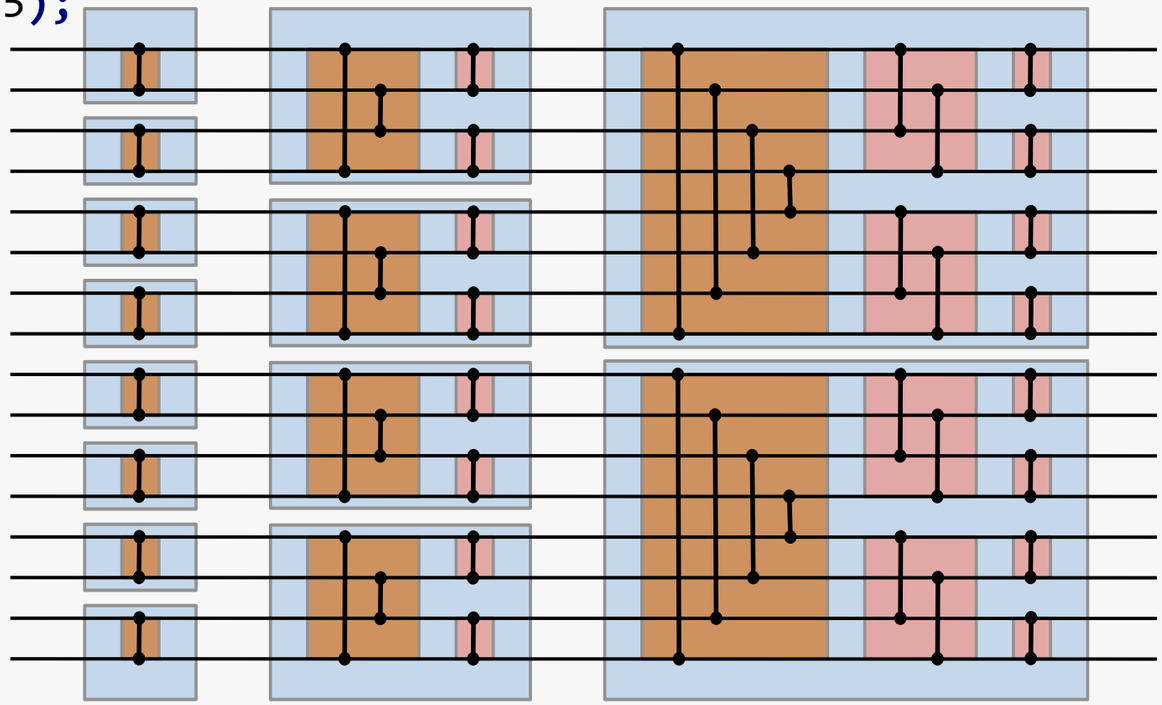


Function sort_two_lanes_of_8()

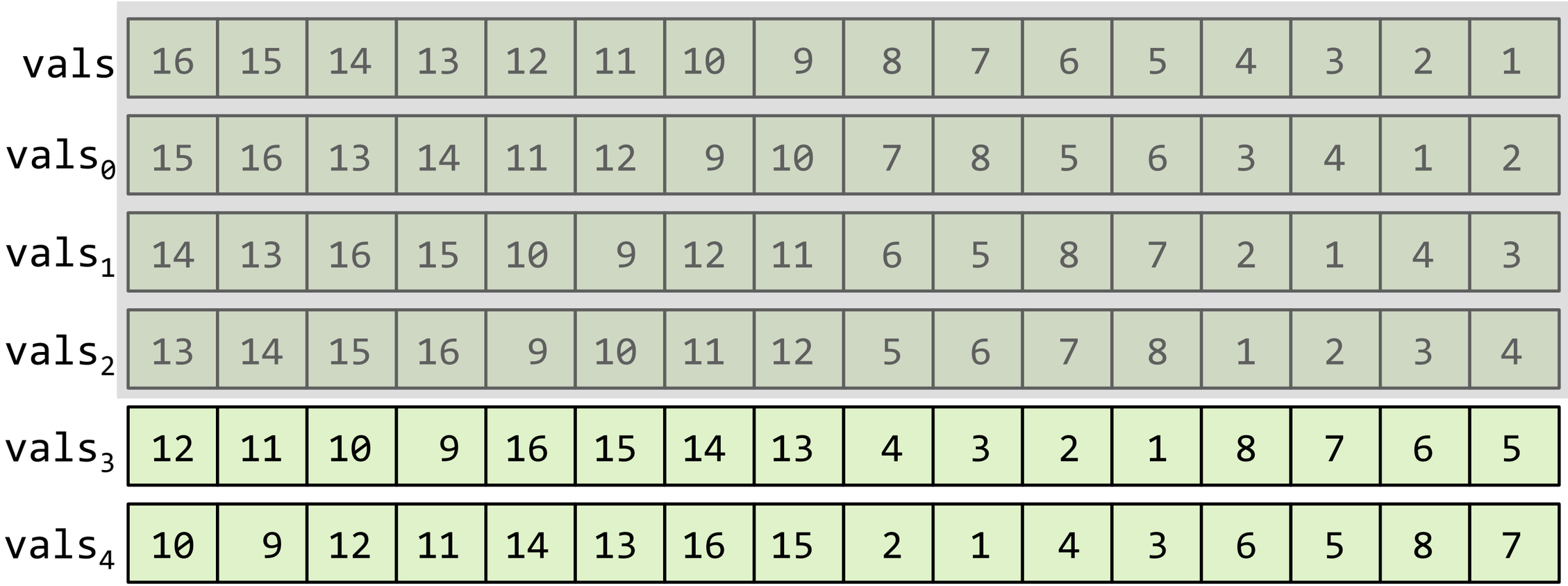


Function sort_two_lanes_of_8()

```
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;
}
```

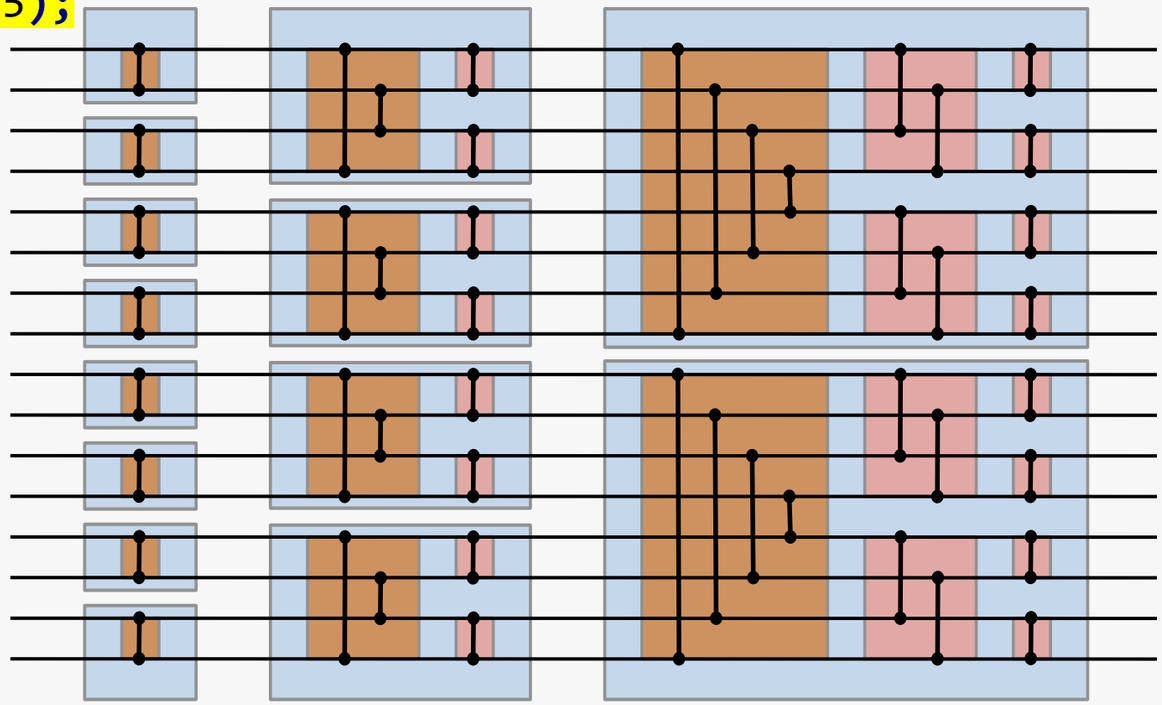


Function sort_two_lanes_of_8()

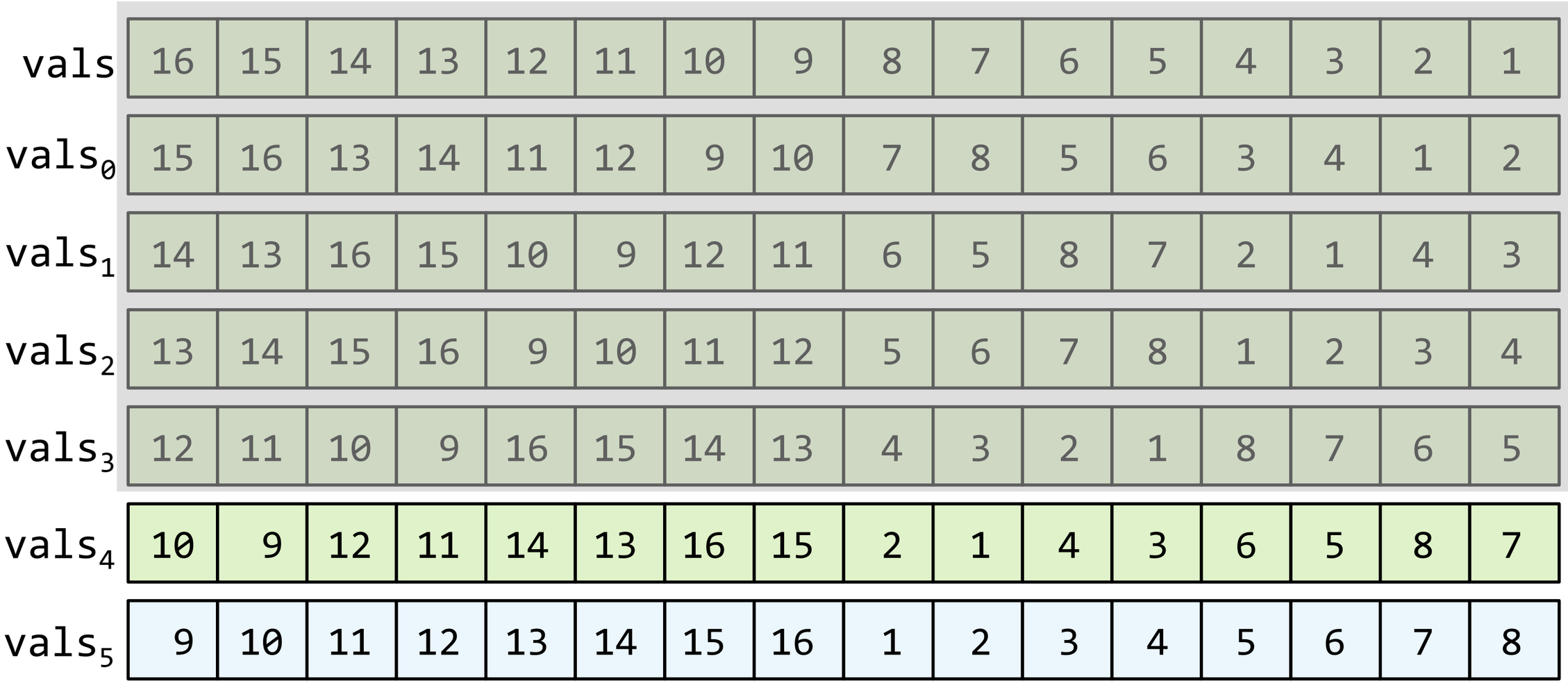


Function sort_two_lanes_of_8()

```
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()



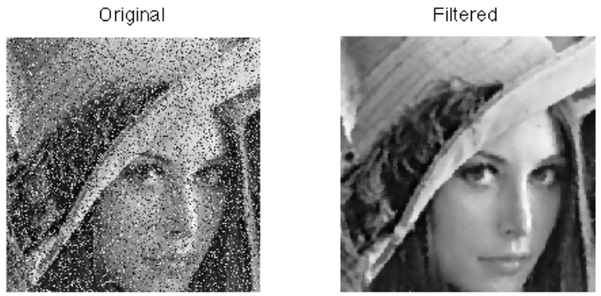
Fast Linear Median Filter

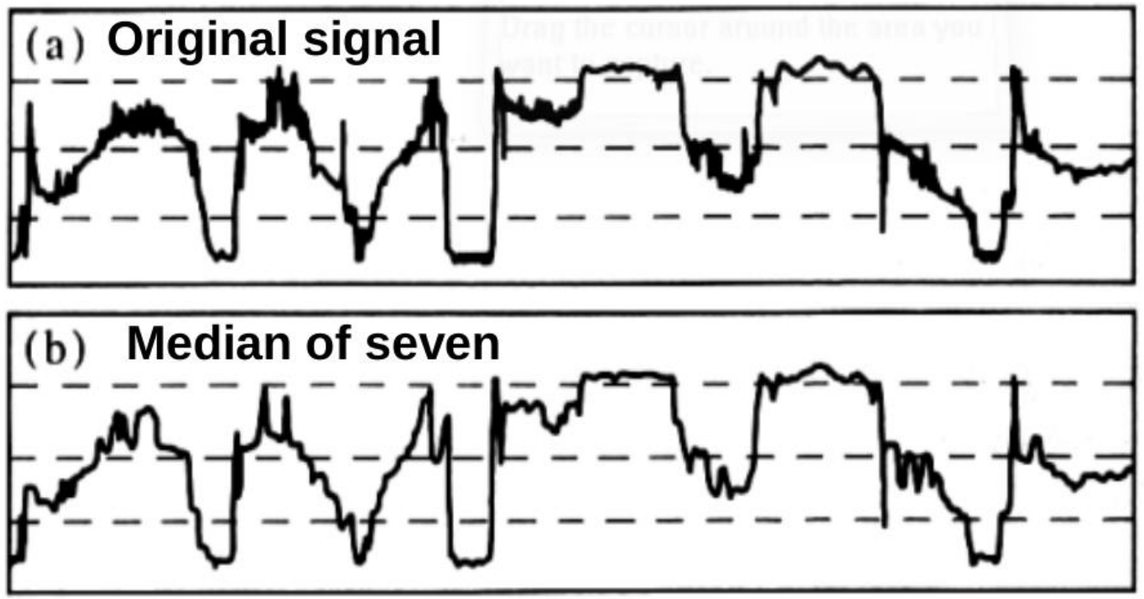
Fast Median Filter?

- 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};
    ...
```

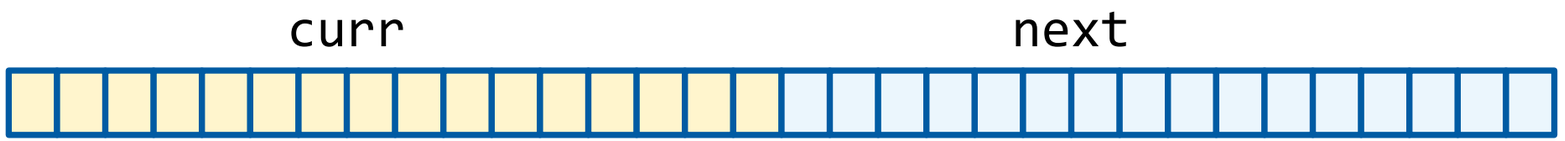
Function avx_median_of_7()

```
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;

    ...
}
```

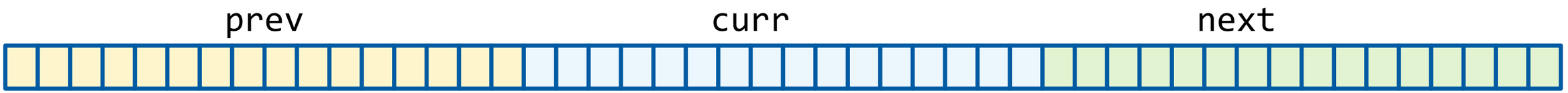
Function avx_median_of_7()



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;
    }
}
```

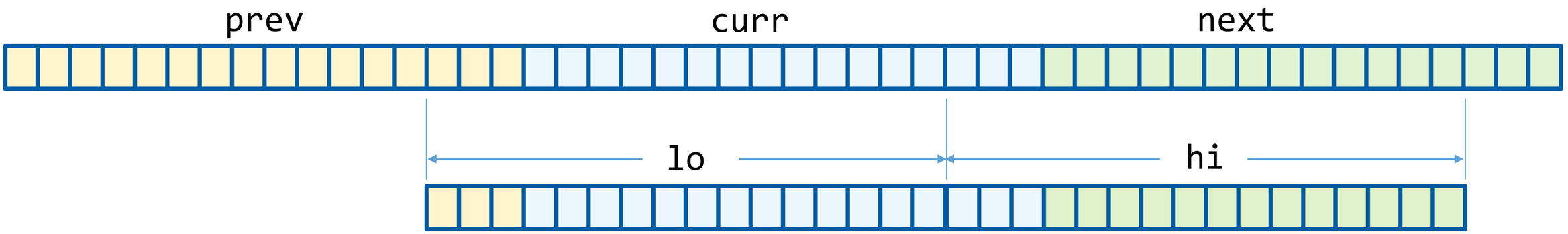
Function avx_median_of_7()



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;
    }
}
```

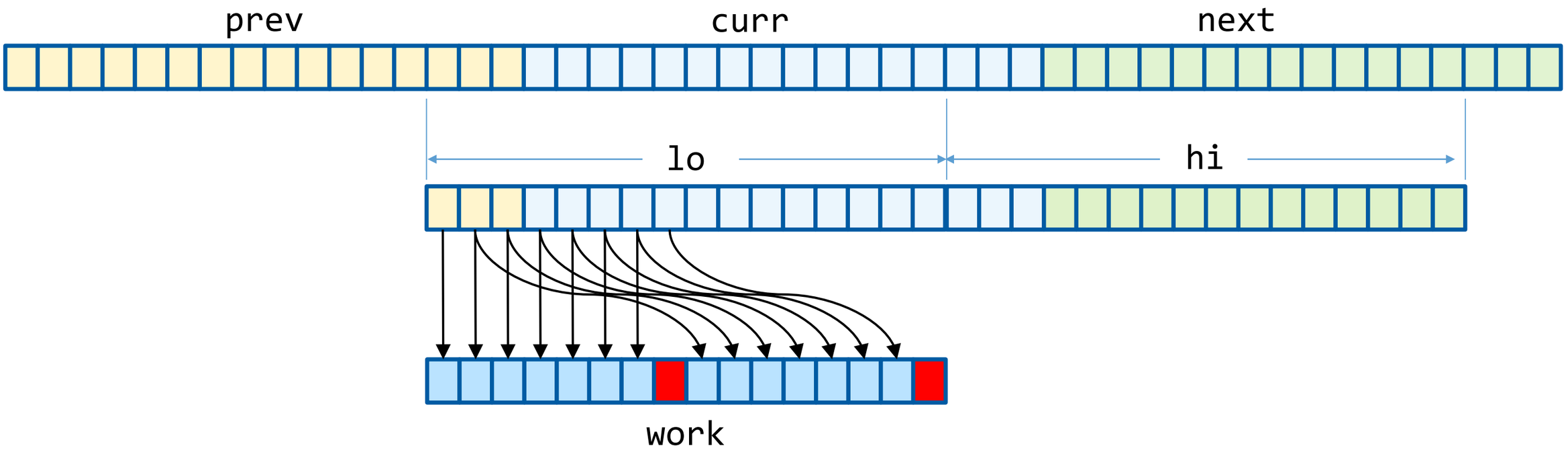
Function avx_median_of_7()



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;
    }
}
```

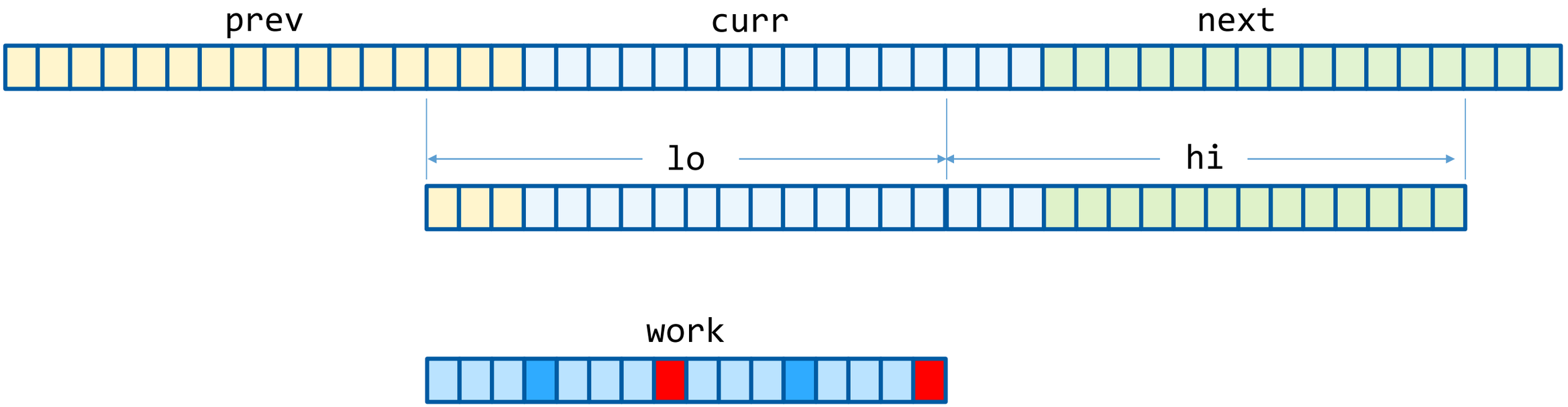
Function avx_median_of_7()



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;
    }
}
```

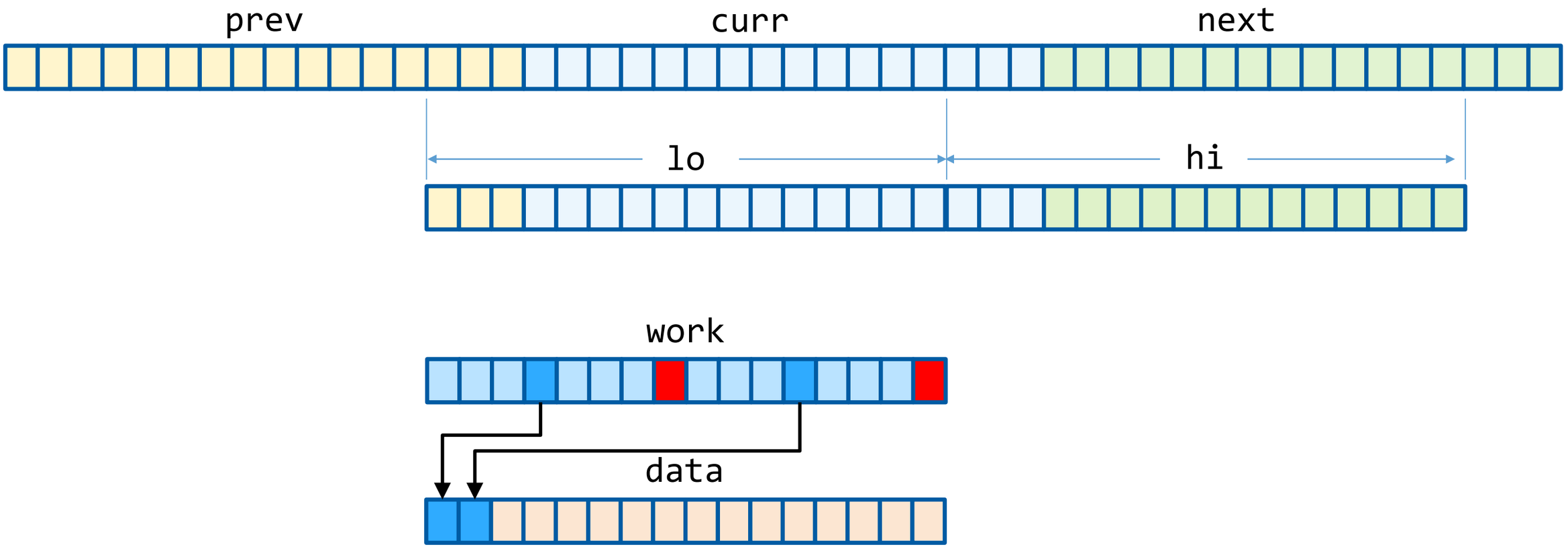
Function avx_median_of_7()



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;
    }
}
```

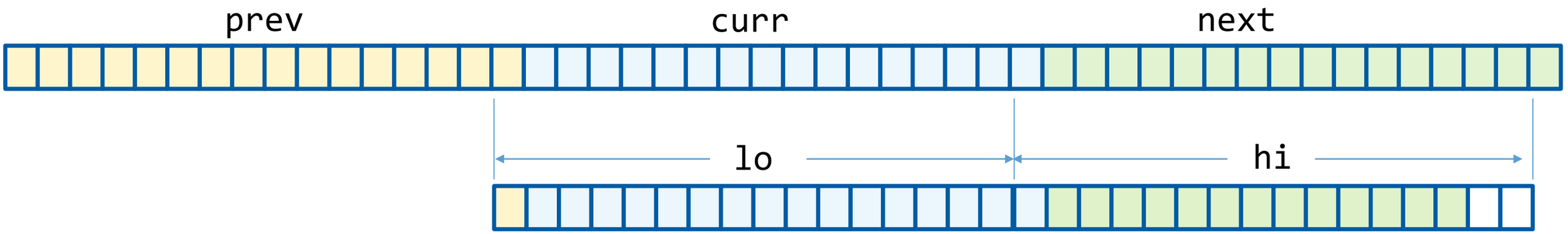
Function avx_median_of_7()



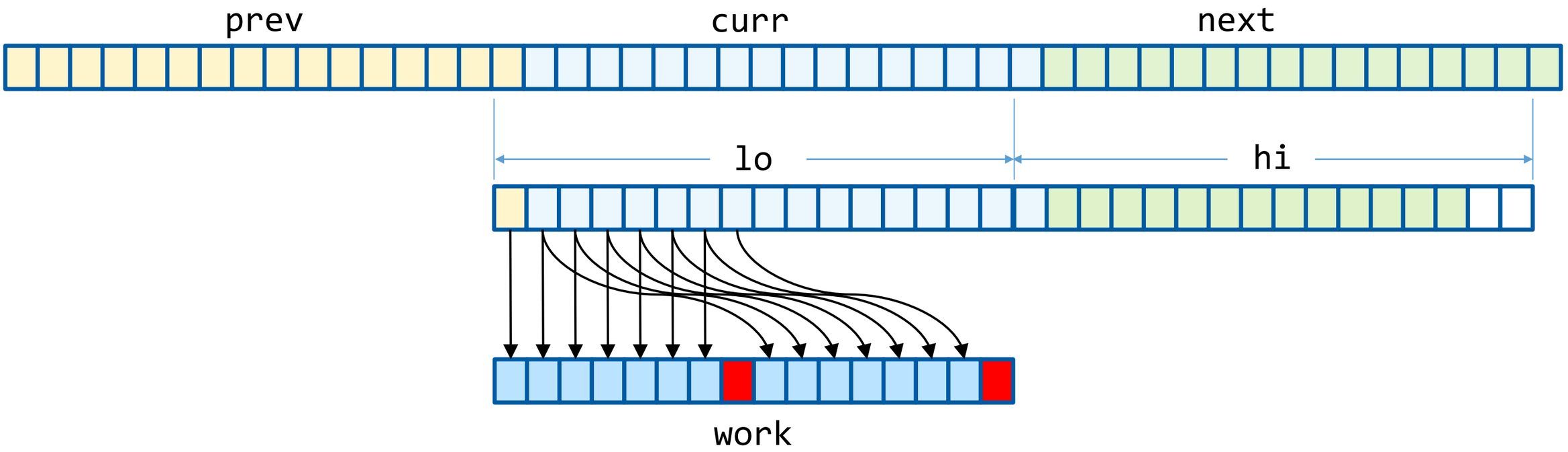
Function avx_median_of_7()

```
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;
    }
}
```

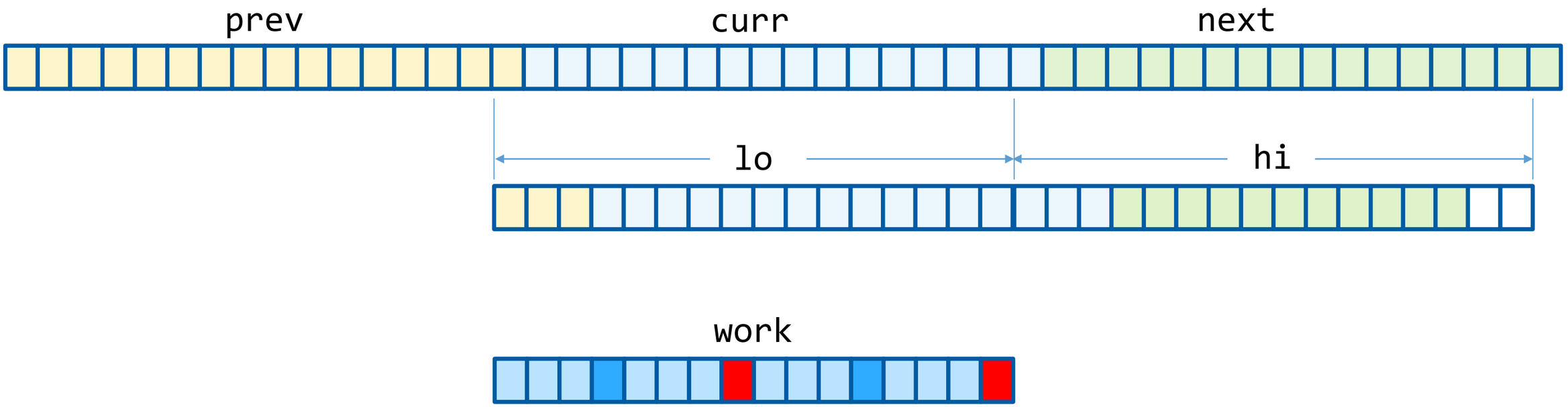
Function avx_median_of_7()



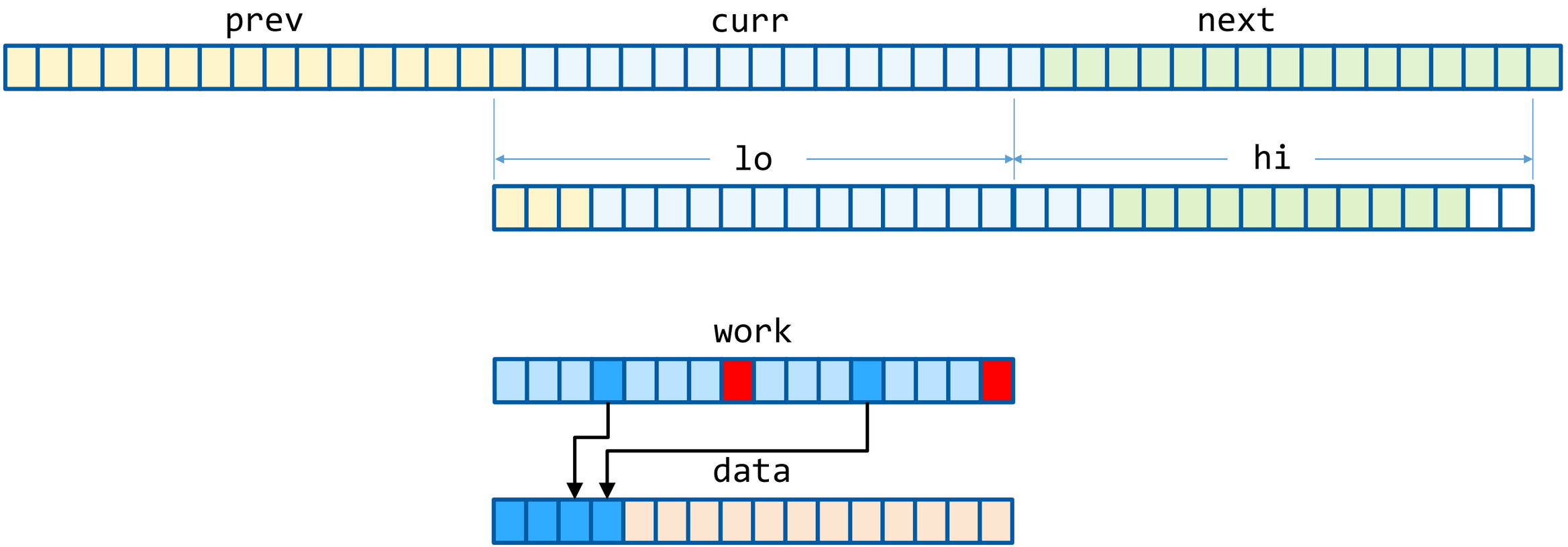
Function avx_median_of_7()



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;
    }
}
```

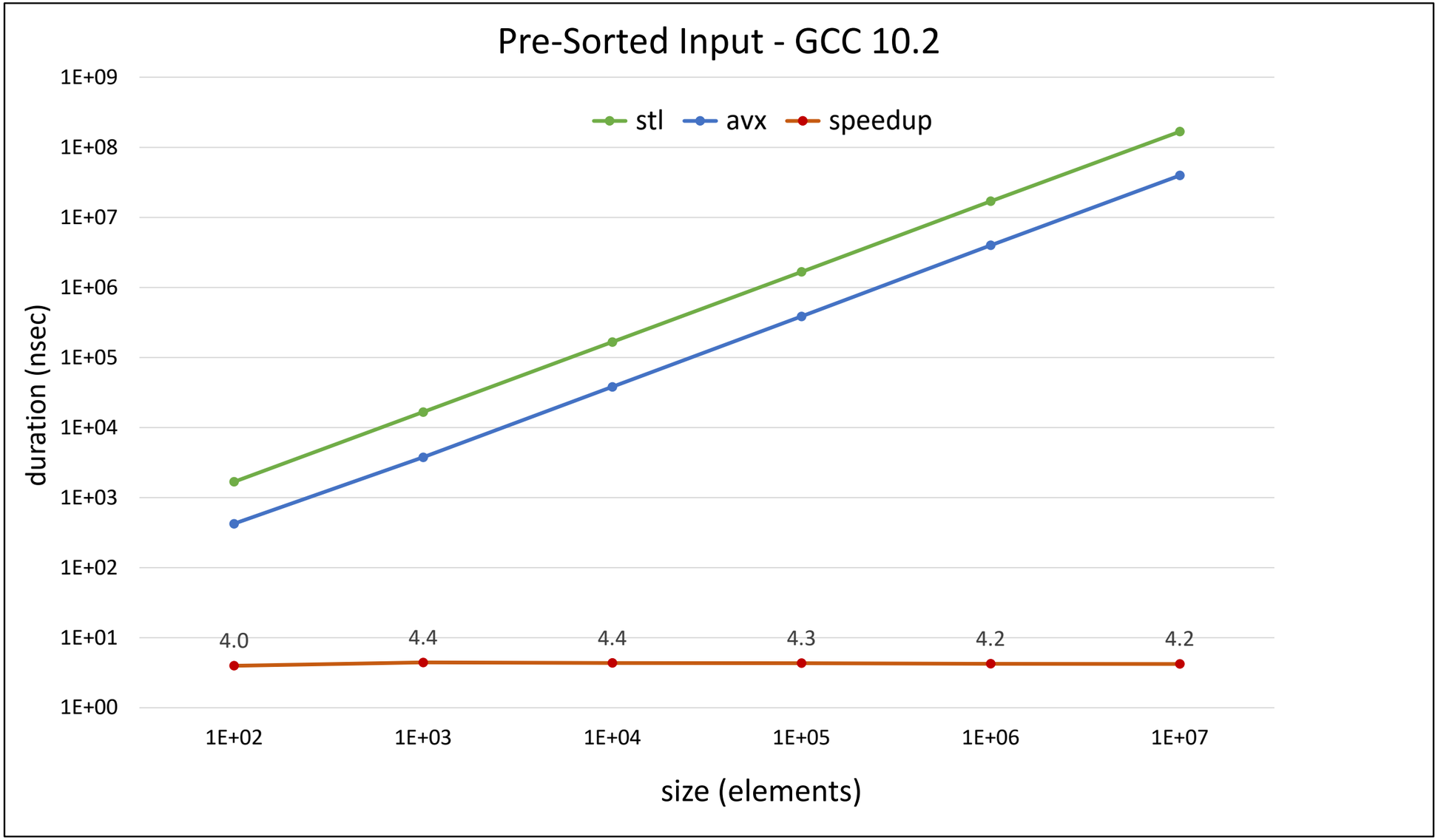
Testing Methodology – Platforms

- 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`

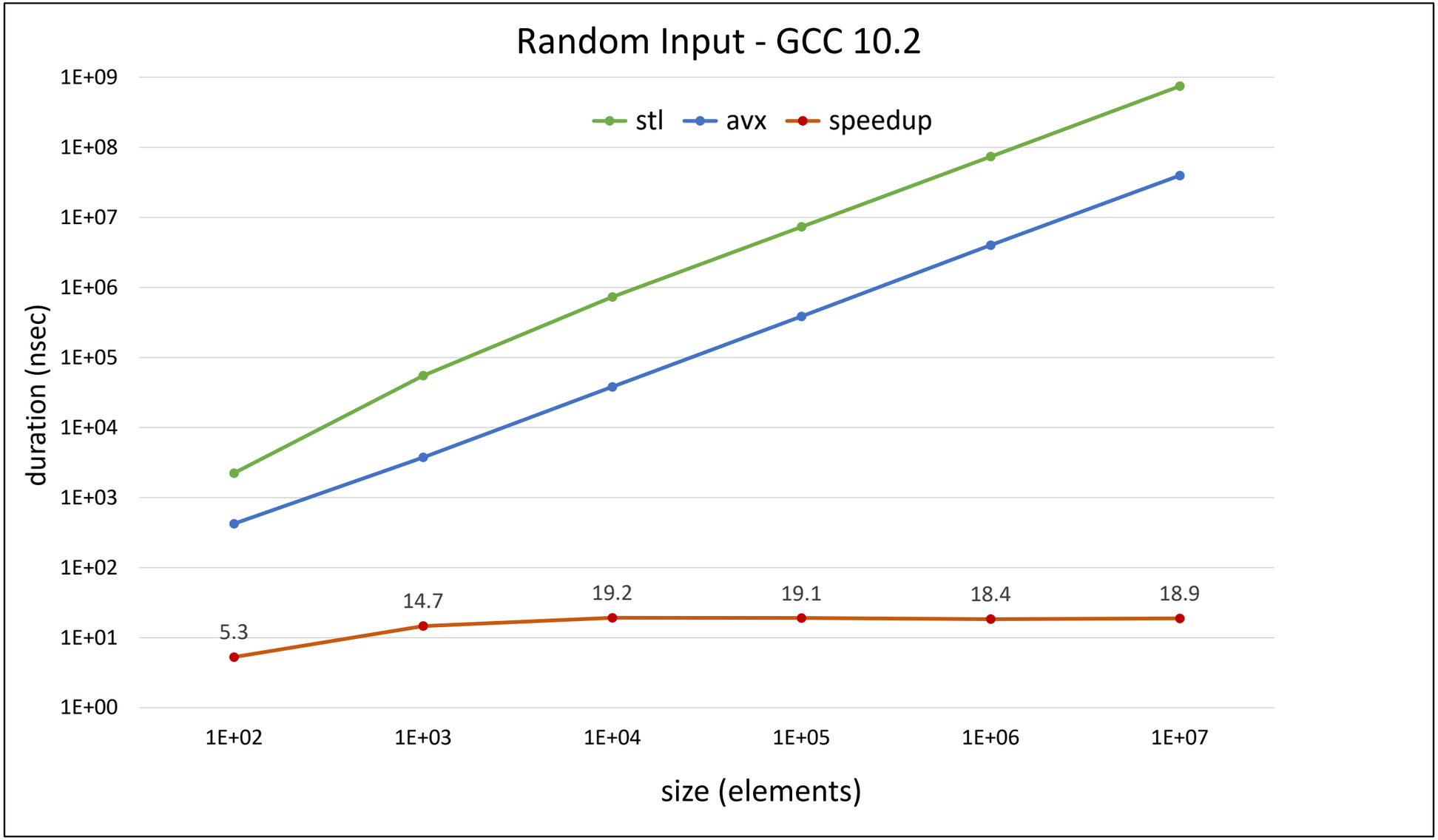
- Element counts of 1E02 through 1E07 (by 10s)
  - Pre-sorted integers increasing monotonically
  - Randomly generated signed integers [-50, +50]
    - Always generated from the same seed

- Collect timings for each combination using two approaches
  - STL-based, performing `std::sort()` for every sequence of seven elements
  - Small-Kernel AVX

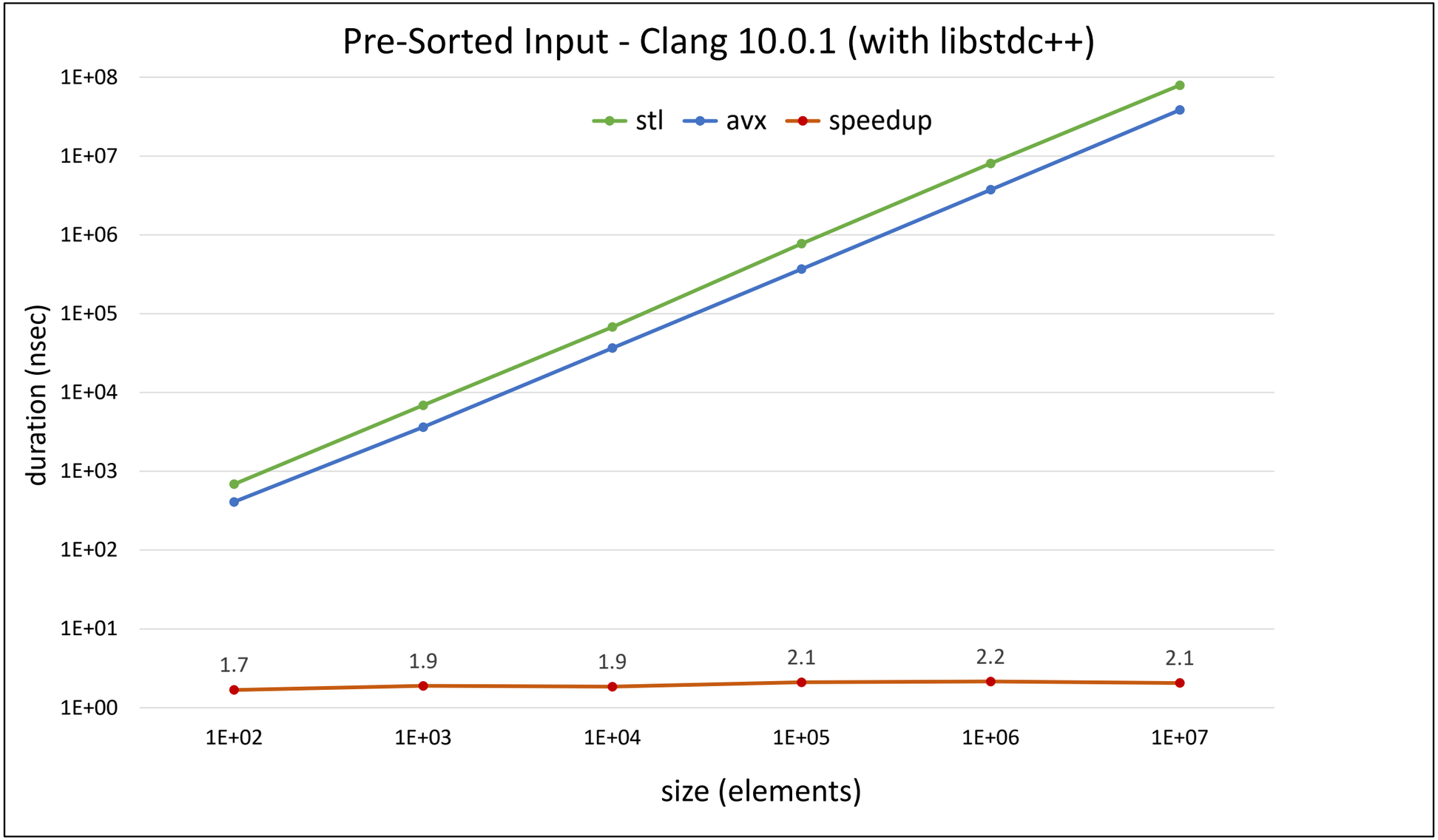
Results – Median-of-Seven – Sorted Input – GCC



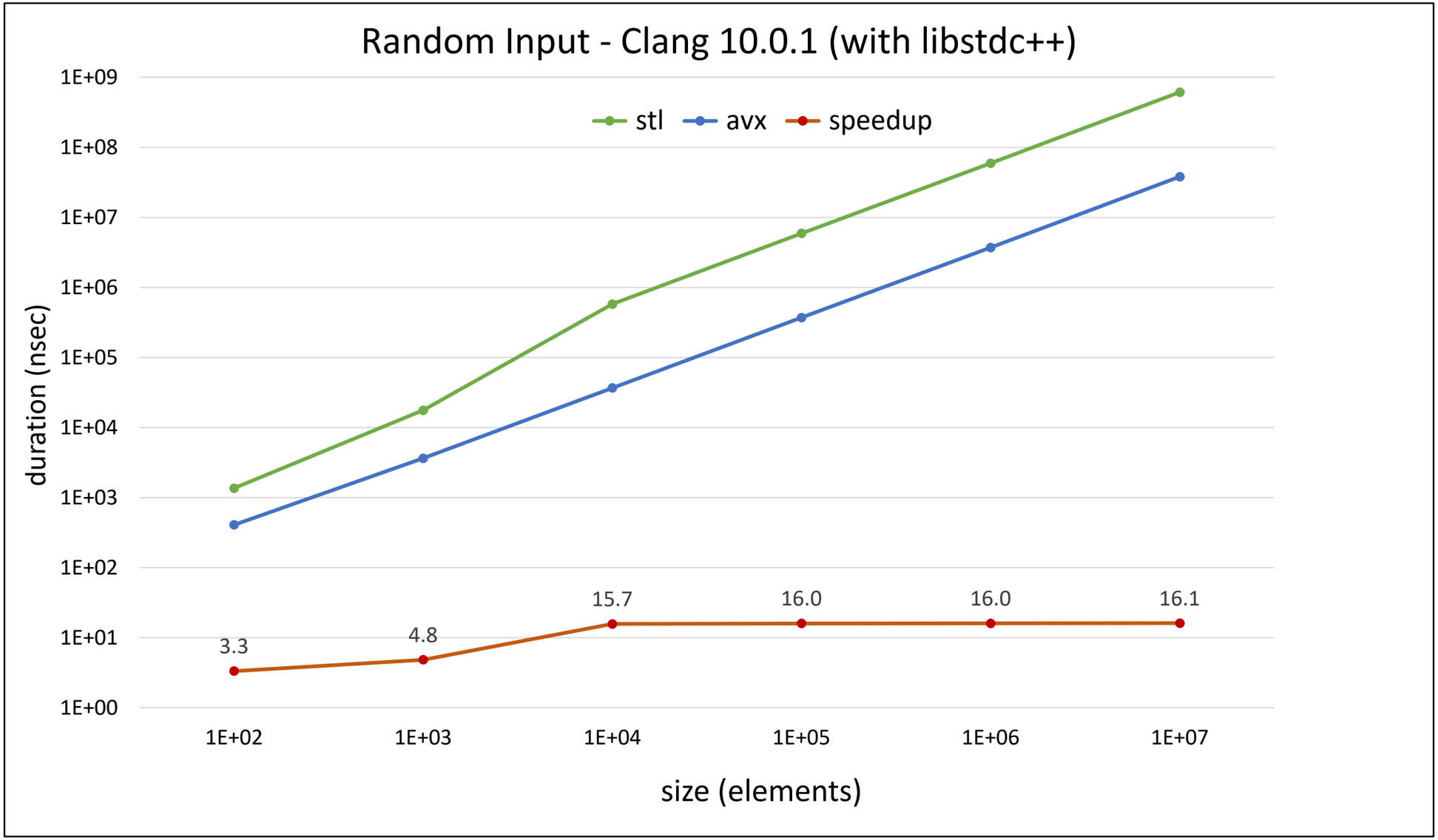
Results – Median-of-Seven – Random Input – GCC



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`