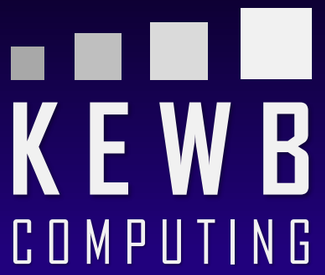
Adventures in SIMD Thinking (Part 2 of 2)

Bob Steagall

CppCon 2020



Agenda

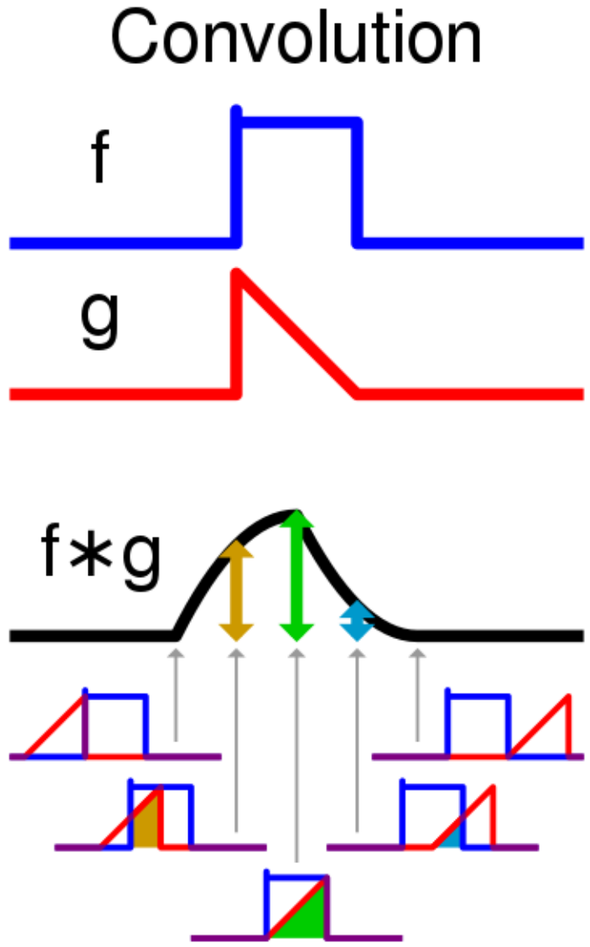
KEWB COMPUTING

- 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"

Small-Kernel Convolution

Convolution

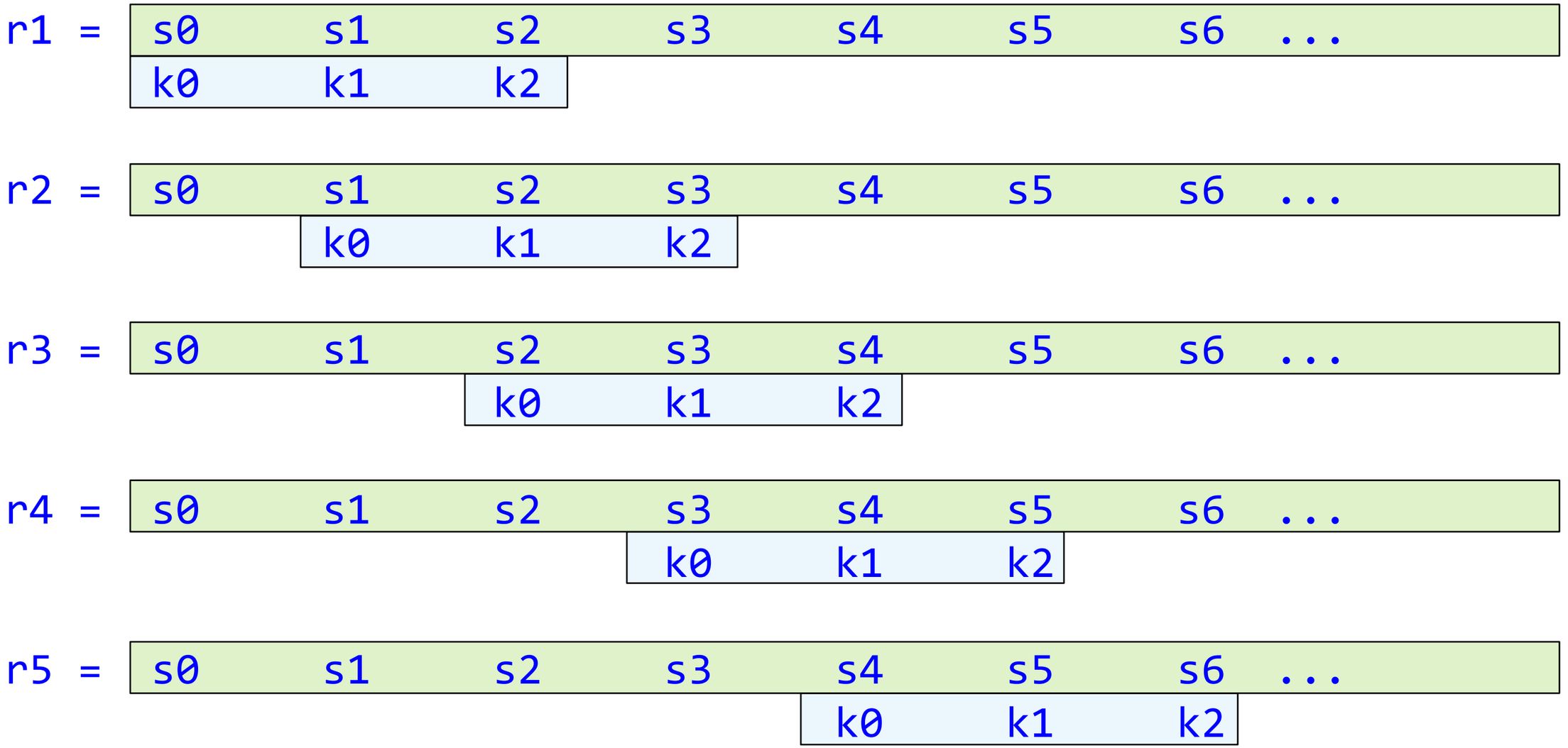
- f is a signal
- g is a kernel
- Output f*g is the convolution
  - Every point of result f*g is f weighted by every point of g
- Useful for smoothing and de-noising



Convolution

```
 S =  s0     s1     s2     s3     s4     s5     s6 ...
 K =  k0     k1     k2

r0 = ...

r1 = s0k0 + s1k1 + s2k2

r2 =        s1k0 + s2k1 + s3k2

r3 =               s2k0 + s3k1 + s4k2

r4 =                      s3k0 + s4k1 + s5k2

r5 =                             s4k0 + s5k1 + s6k2
```

Convolution



Convolution

```
 S =  s0     s1     s2     s3     s4     s5     s6 ...
 K =  k0     k1     k2
```

```
r0 = ...

r1 = s0k0 + s1k1 + s2k2

r2 = s1k0 + s2k1 + s3k2

r3 = s2k0 + s3k1 + s4k2

r4 = s3k0 + s4k1 + s5k2

r5 = s4k0 + s5k1 + s6k2
```

Convolution

```
 S =  s0    s1    s2    s3    s4    s5    s6 ...
 K =  k0    k1    k2
```

```
r0 = ...

r1 = s0k0 + s1k1 + s2k2

r2 = s1k0 + s2k1 + s3k2

r3 = s2k0 + s3k1 + s4k2

r4 = s3k0 + s4k1 + s5k2

r5 = s4k0 + s5k1 + s6k2
```

Convolution

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

    ...
```

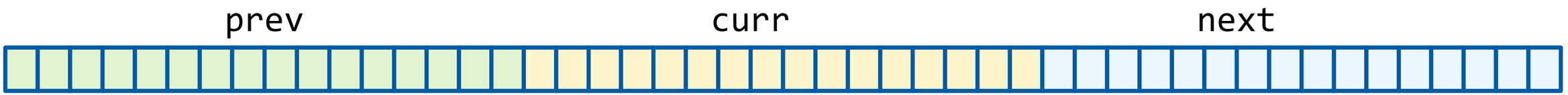
Function Template avx_convolve()

```
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()



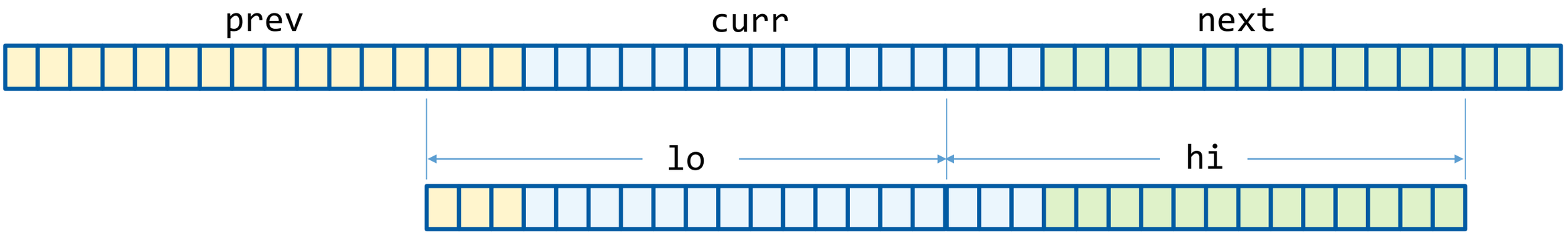
Function Template avx_convolve()

```
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);
    }
}
```

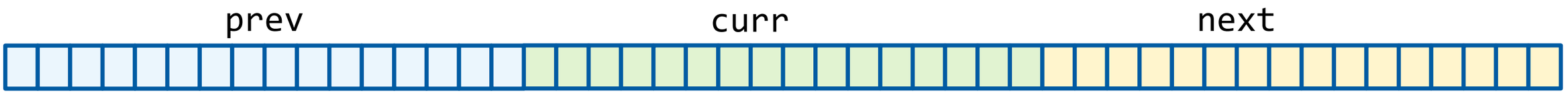
Function Template avx_convolve()



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);
    }
}
```

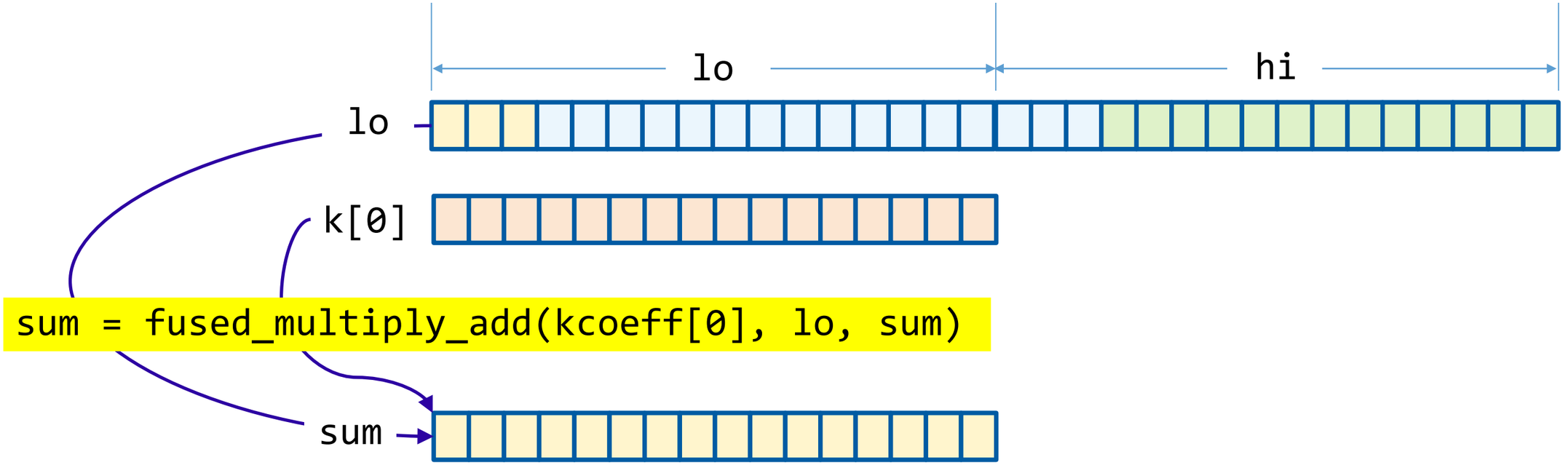
Function Template avx_convolve()



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);
    }
}
```

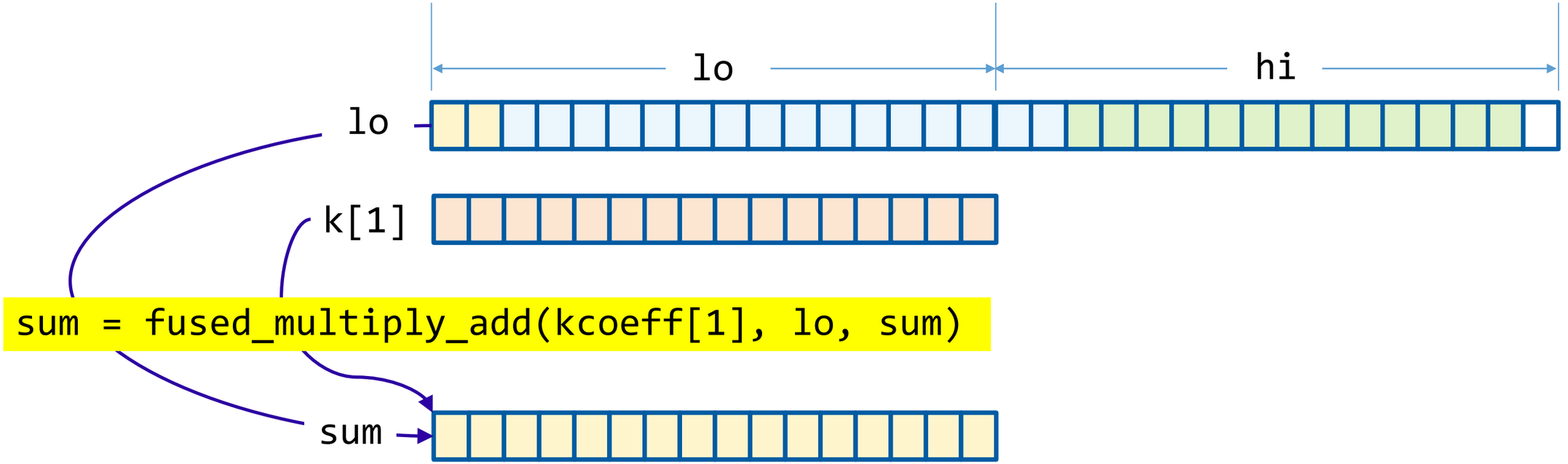
Function Template avx_convolve()



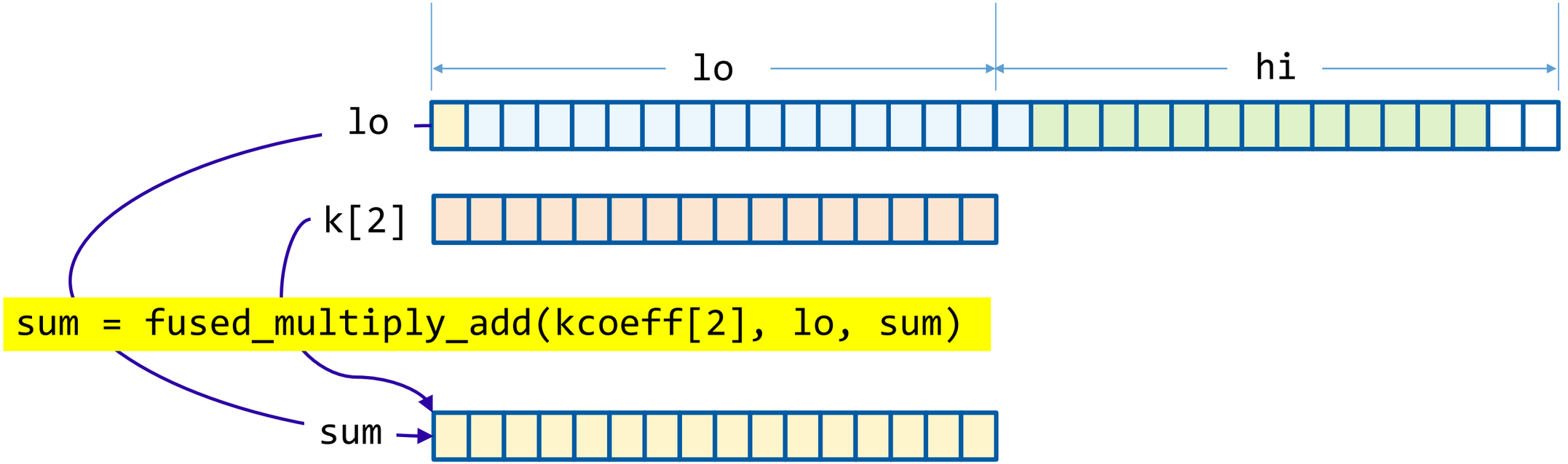
Function Template avx_convolve()

```
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);
    }
}
```

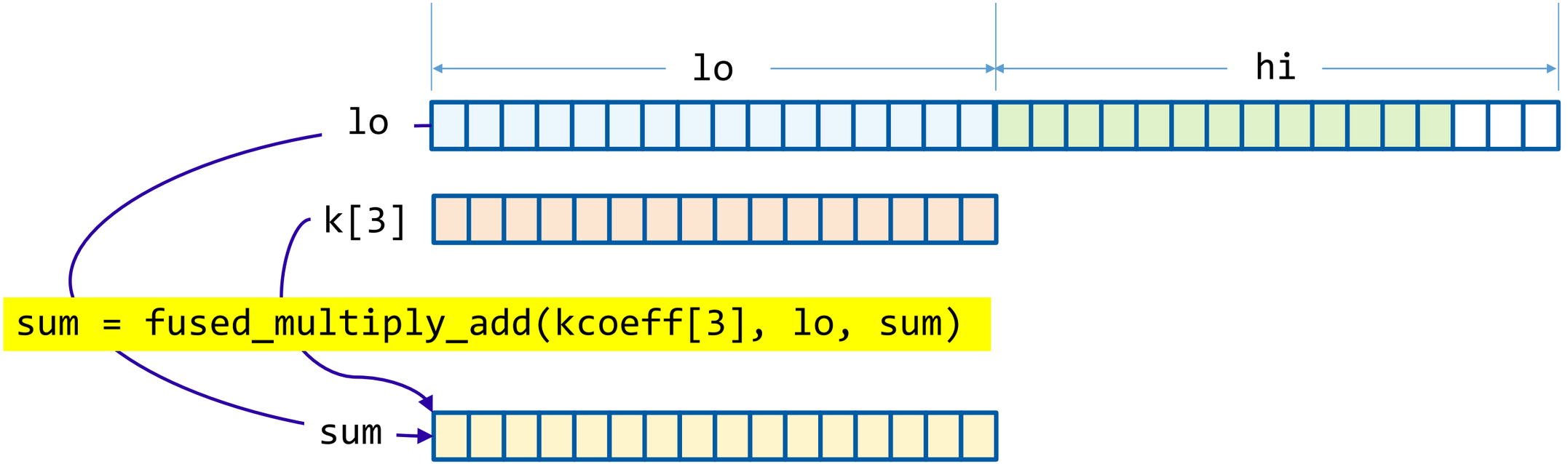
Function Template avx_convolve()



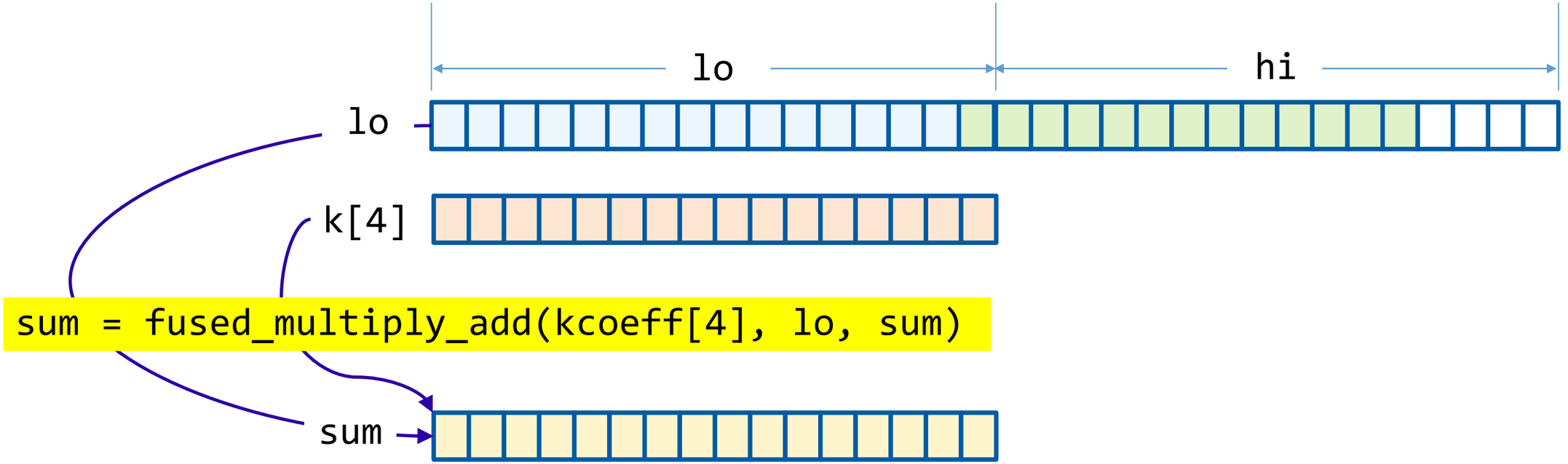
Function Template avx_convolve()



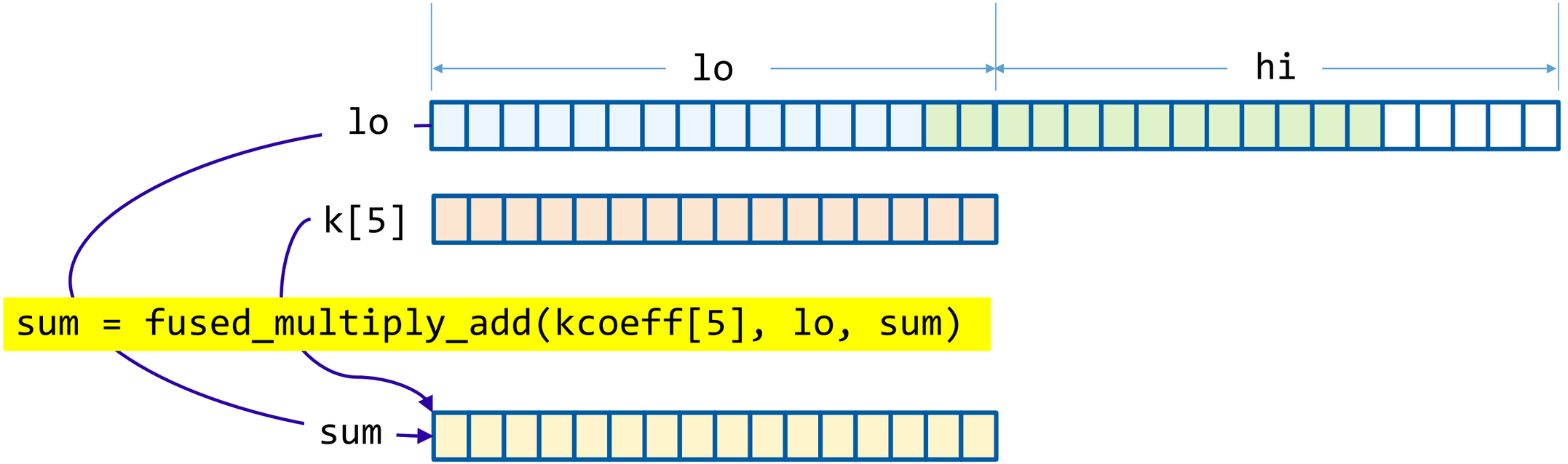
Function Template avx_convolve()



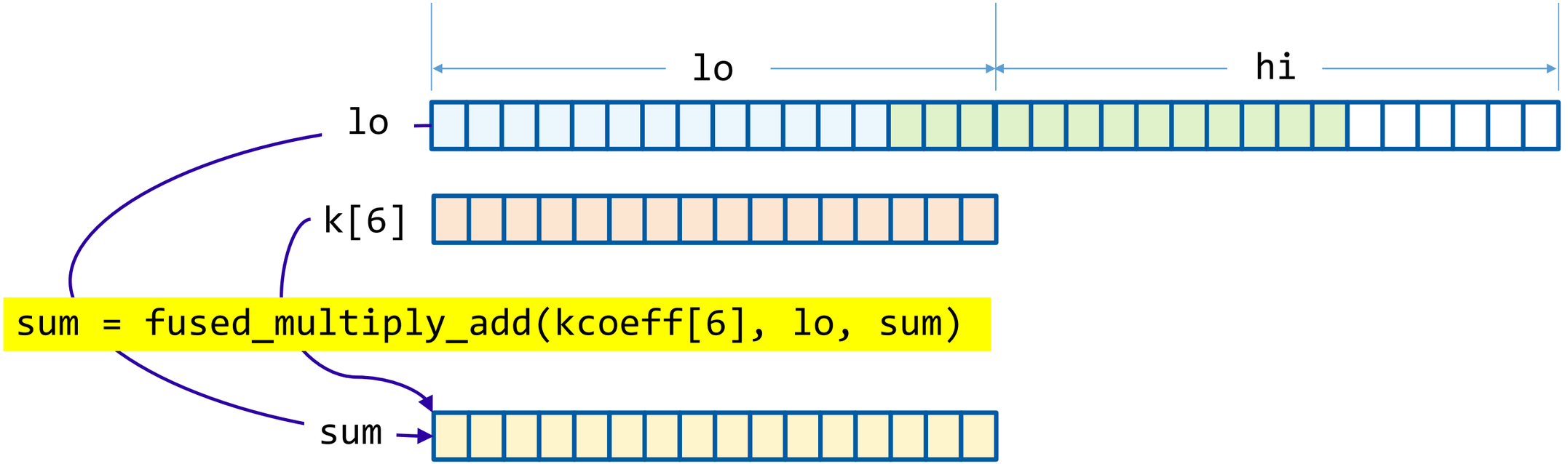
Function Template avx_convolve()



Function Template avx_convolve()



Function Template avx_convolve()



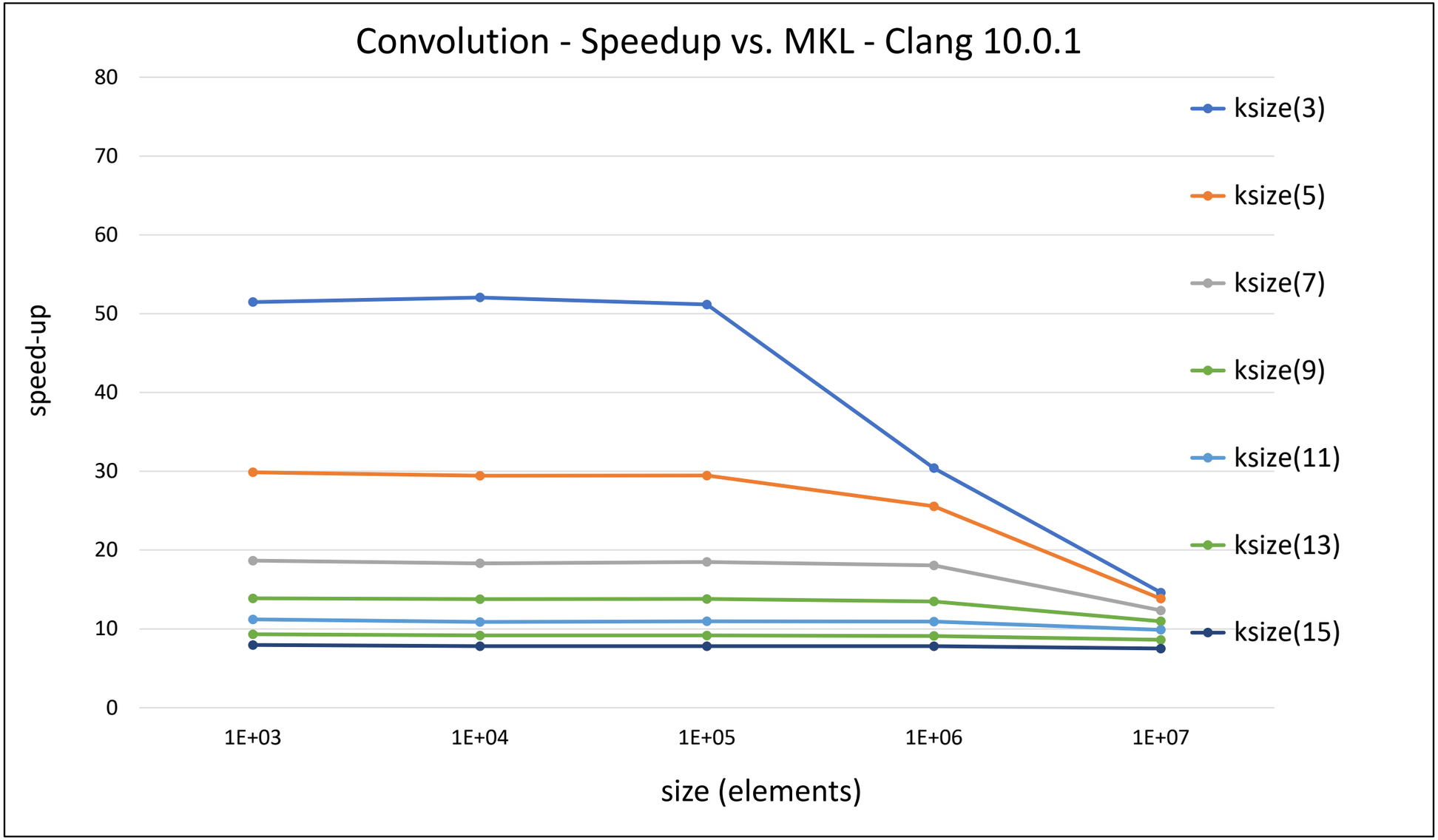
Function Template avx_convolve()

```
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);
    }
}
```

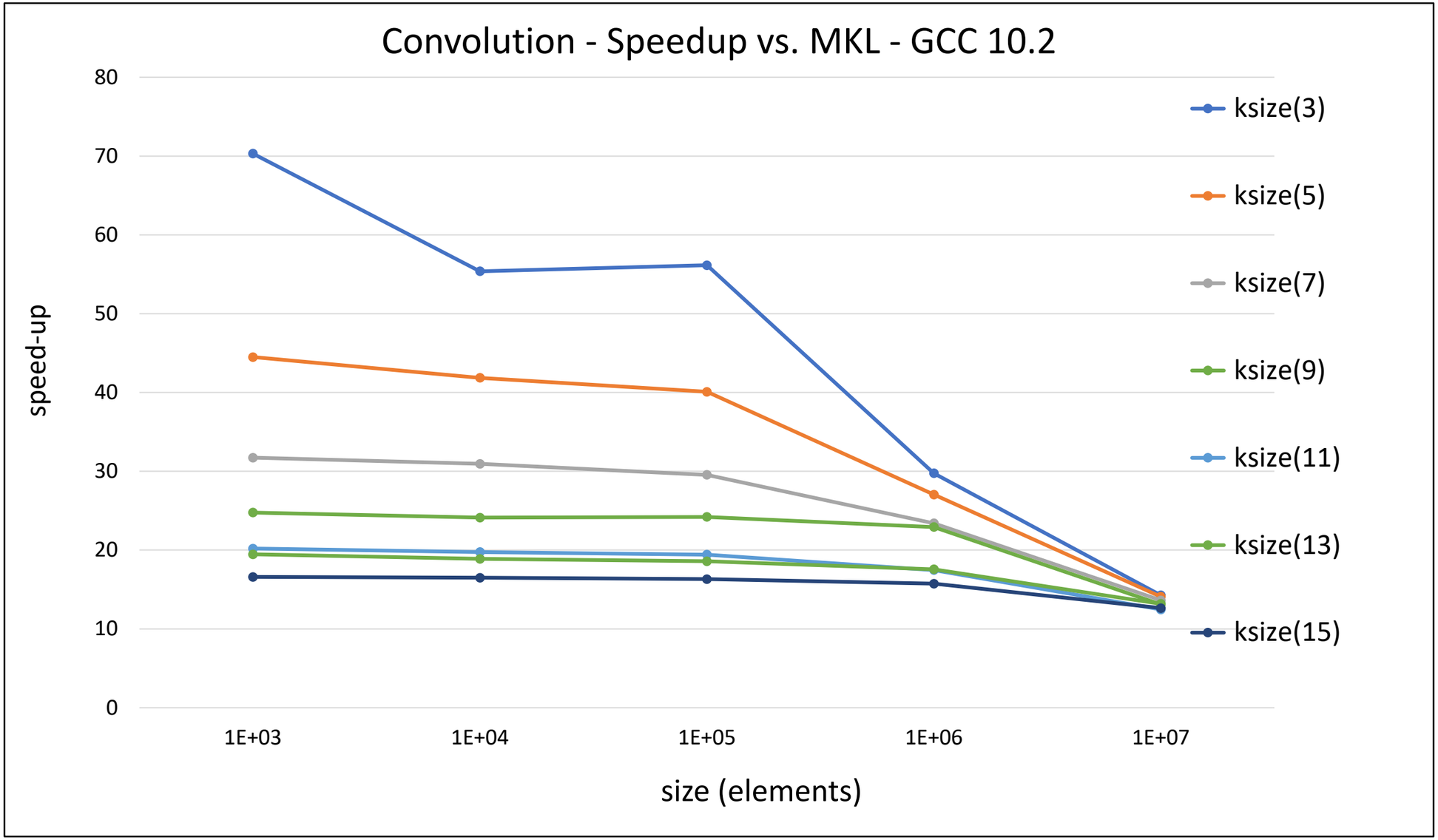
Testing Methodology – Platforms

- 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`
- 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

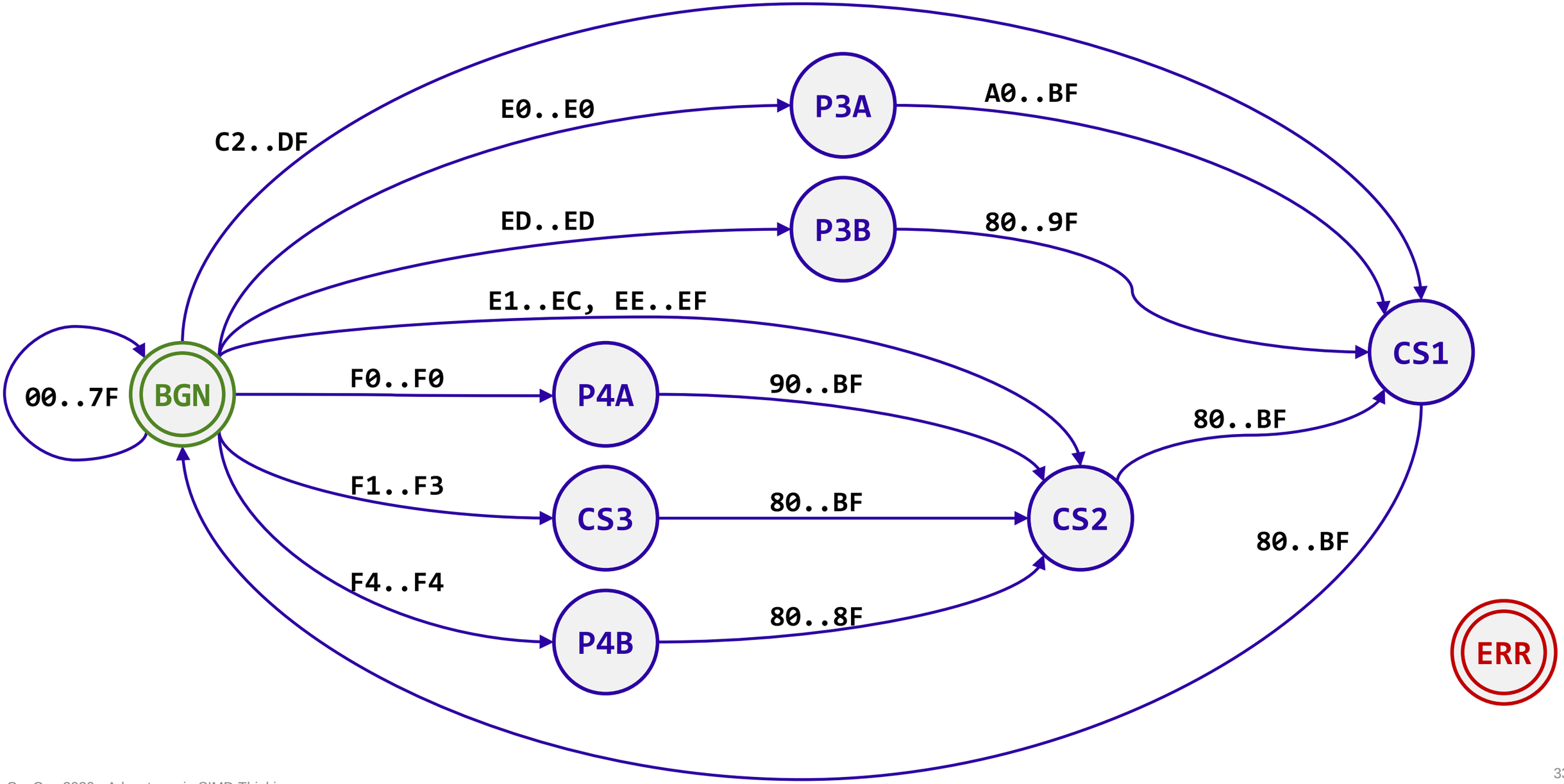


Results – Convolution – GCC 10.2



UTF-8 to UTF-32 Conversion Algorithm

The UTF-8 Decoding DFA



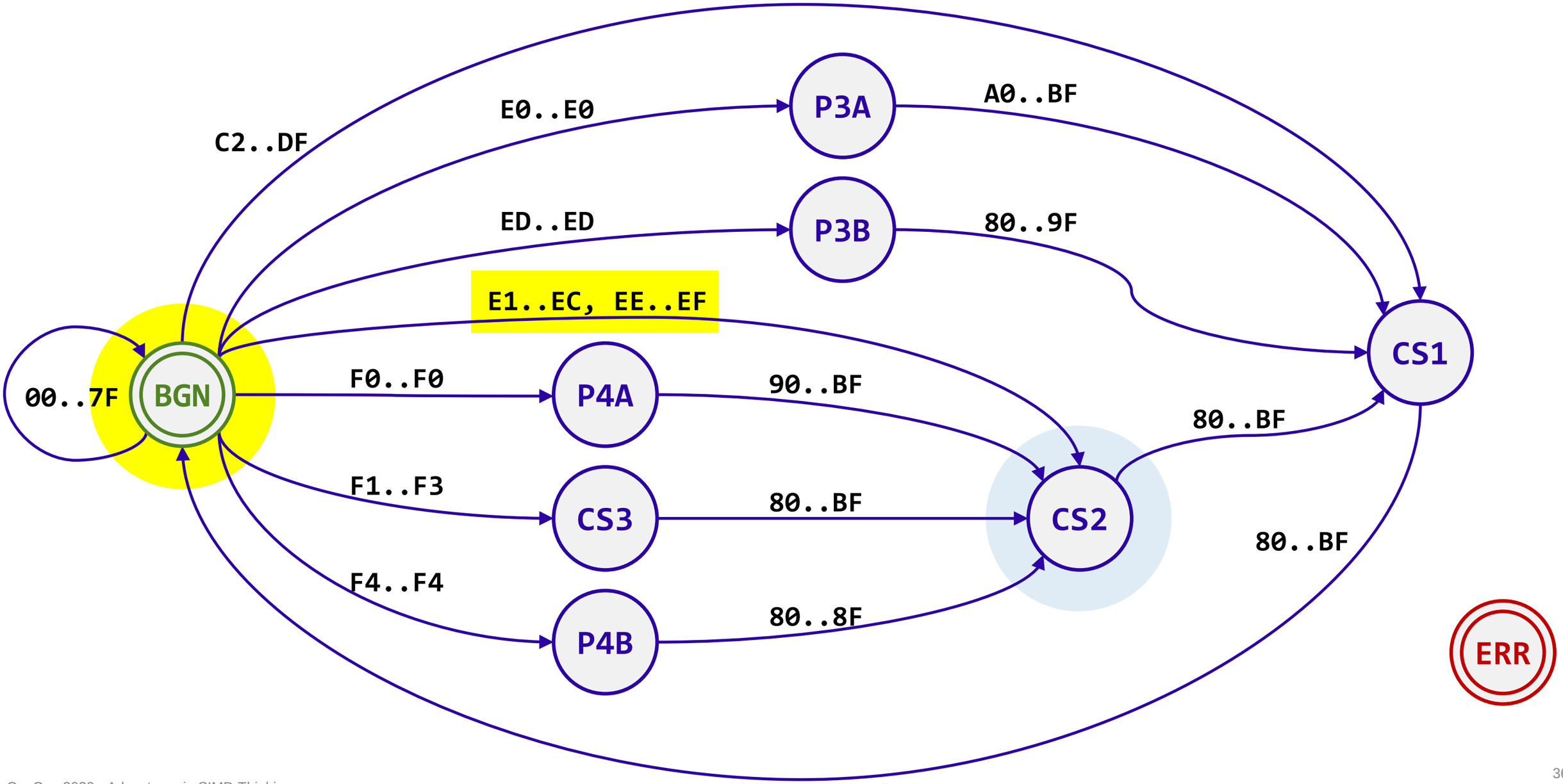
The Basic Conversion Algorithm (UTF-8 to UTF-32)

```
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;
}
```

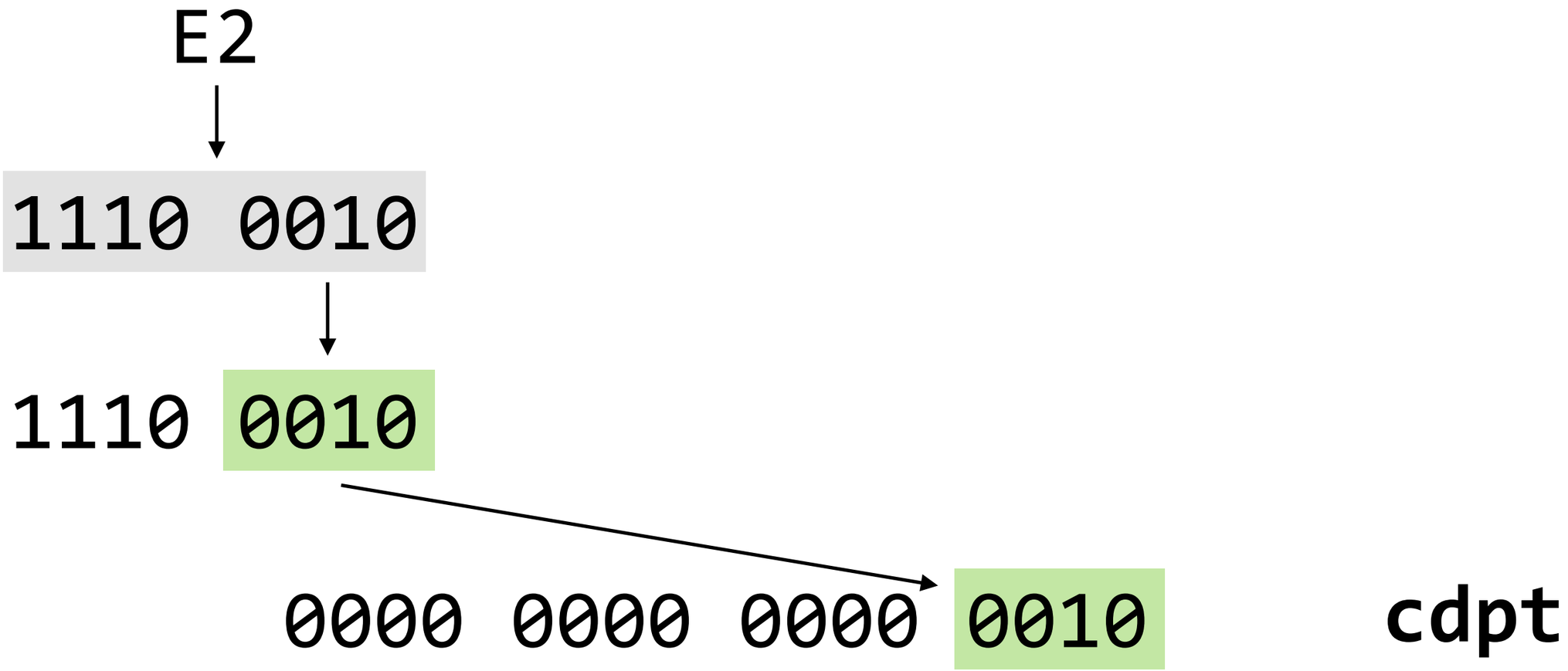
Converting a Single Code Point

```
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;
}
```

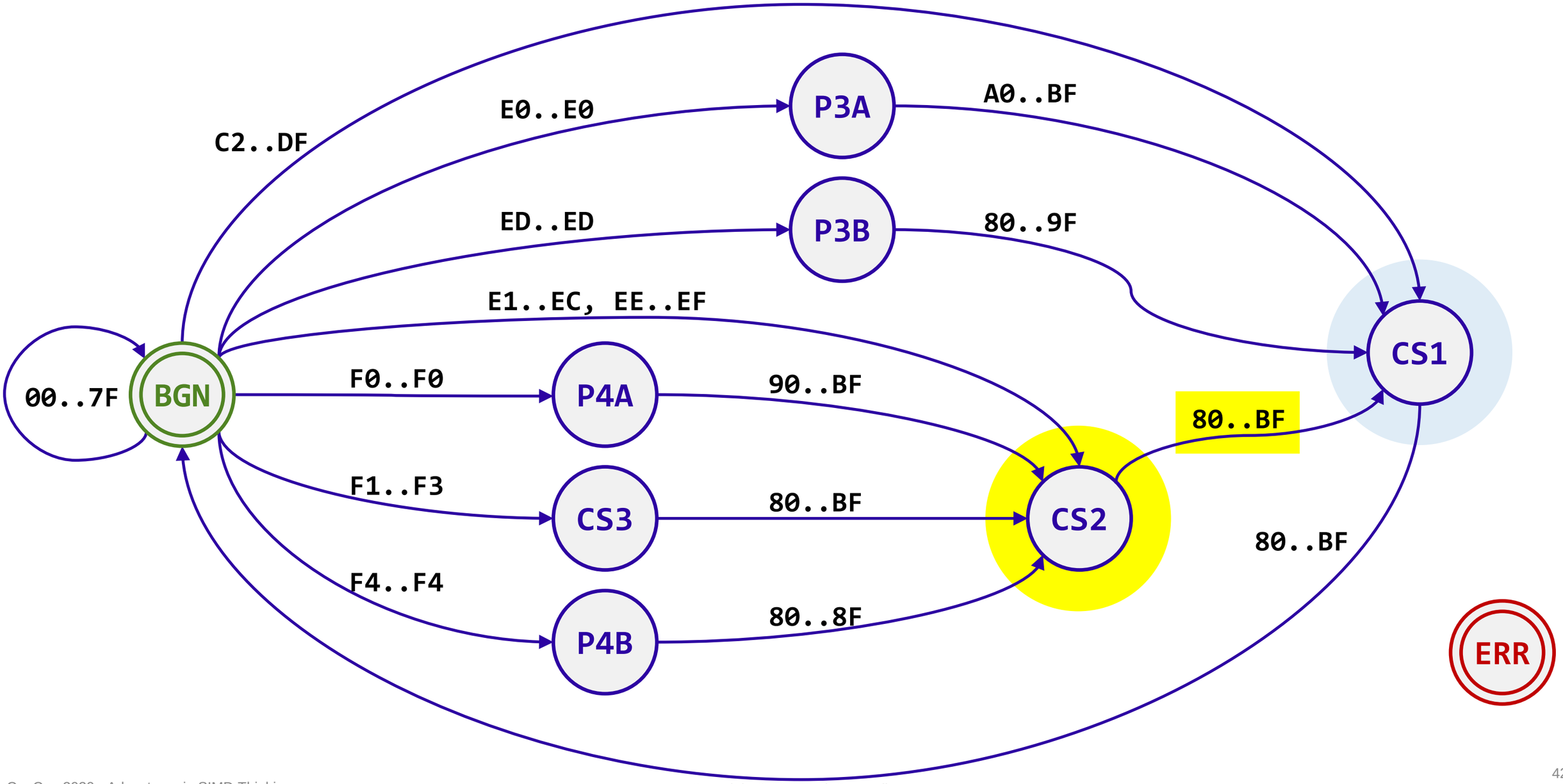
Converting a Single Code Point

```
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;
}
```

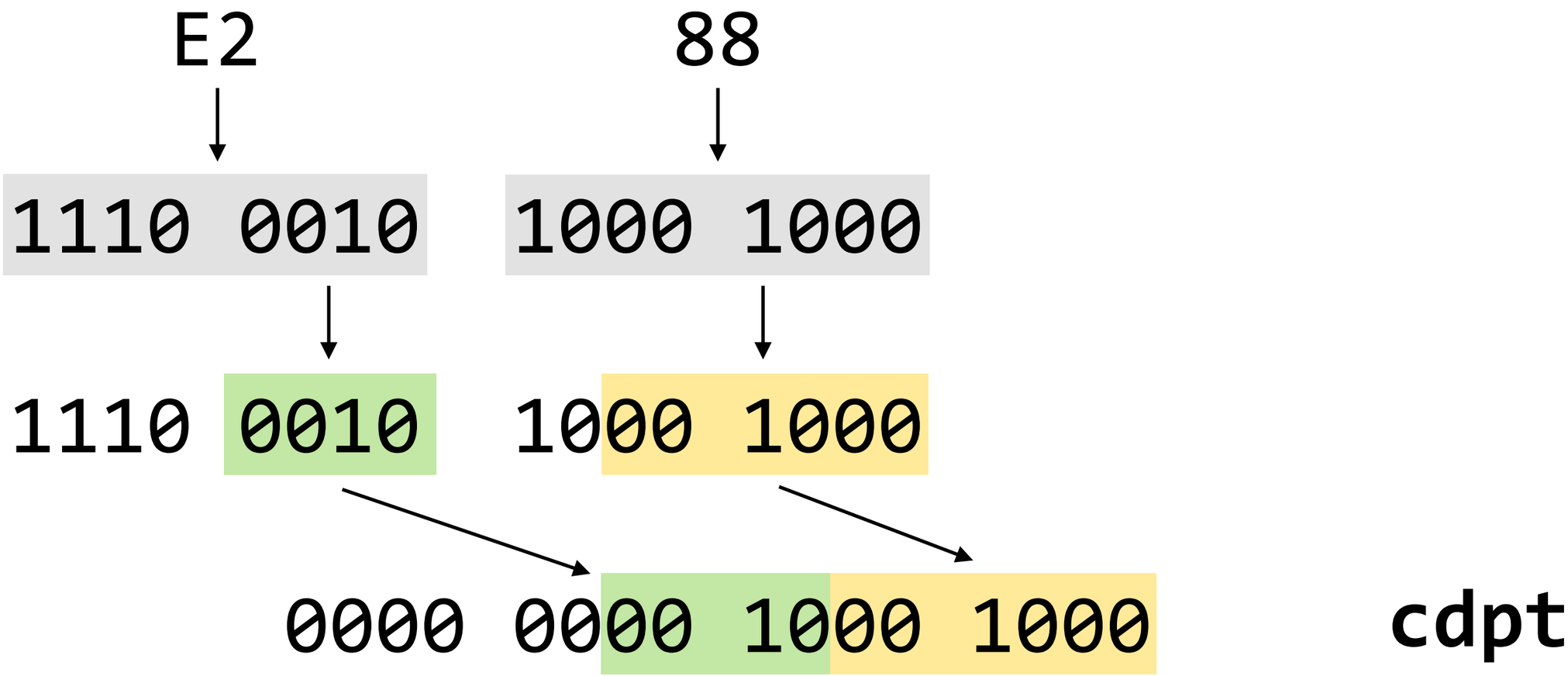
Converting a Single Code Point

```
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;
}
```

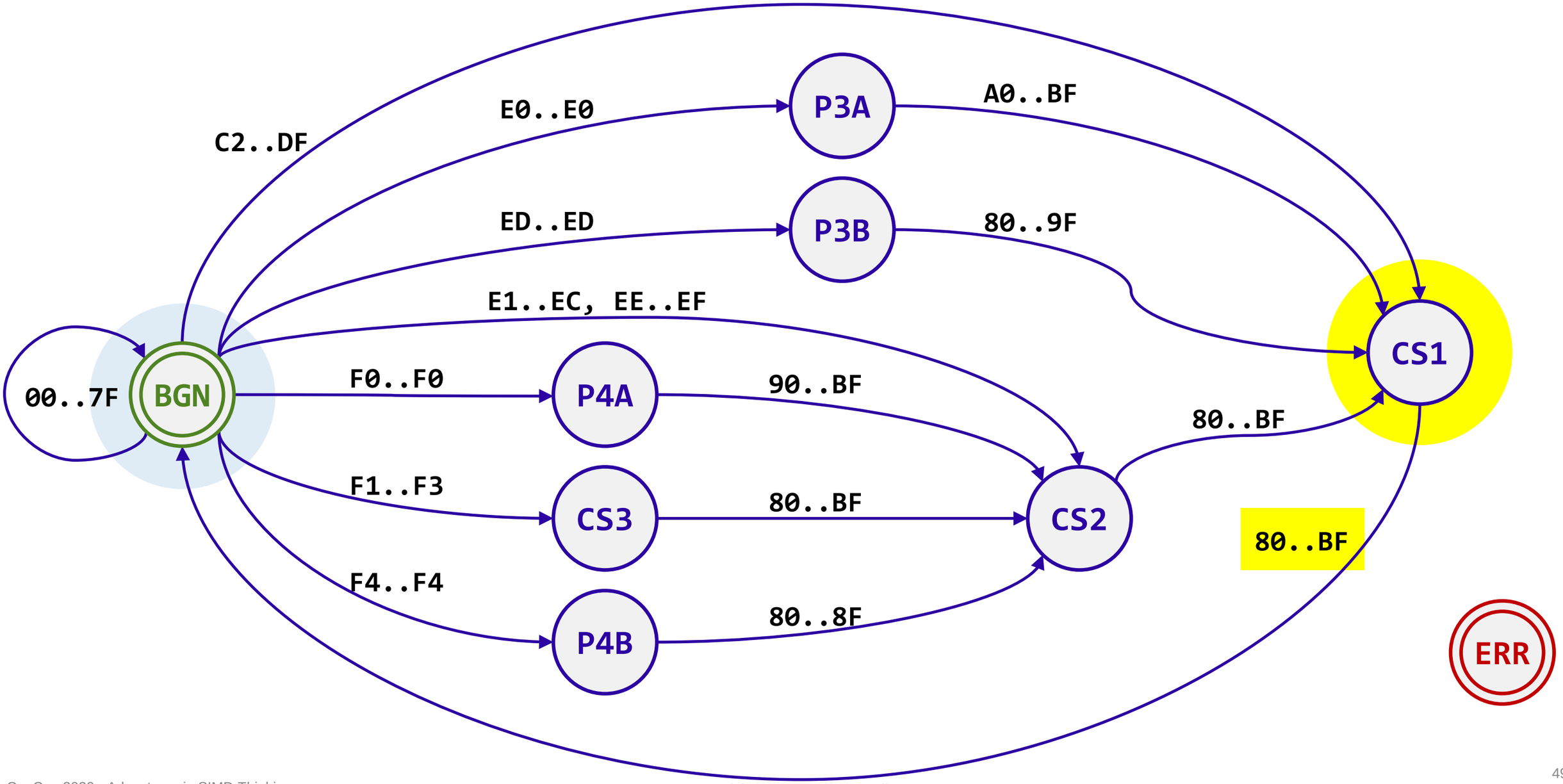
Converting a Single Code Point

```
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 .. }



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;
}
```

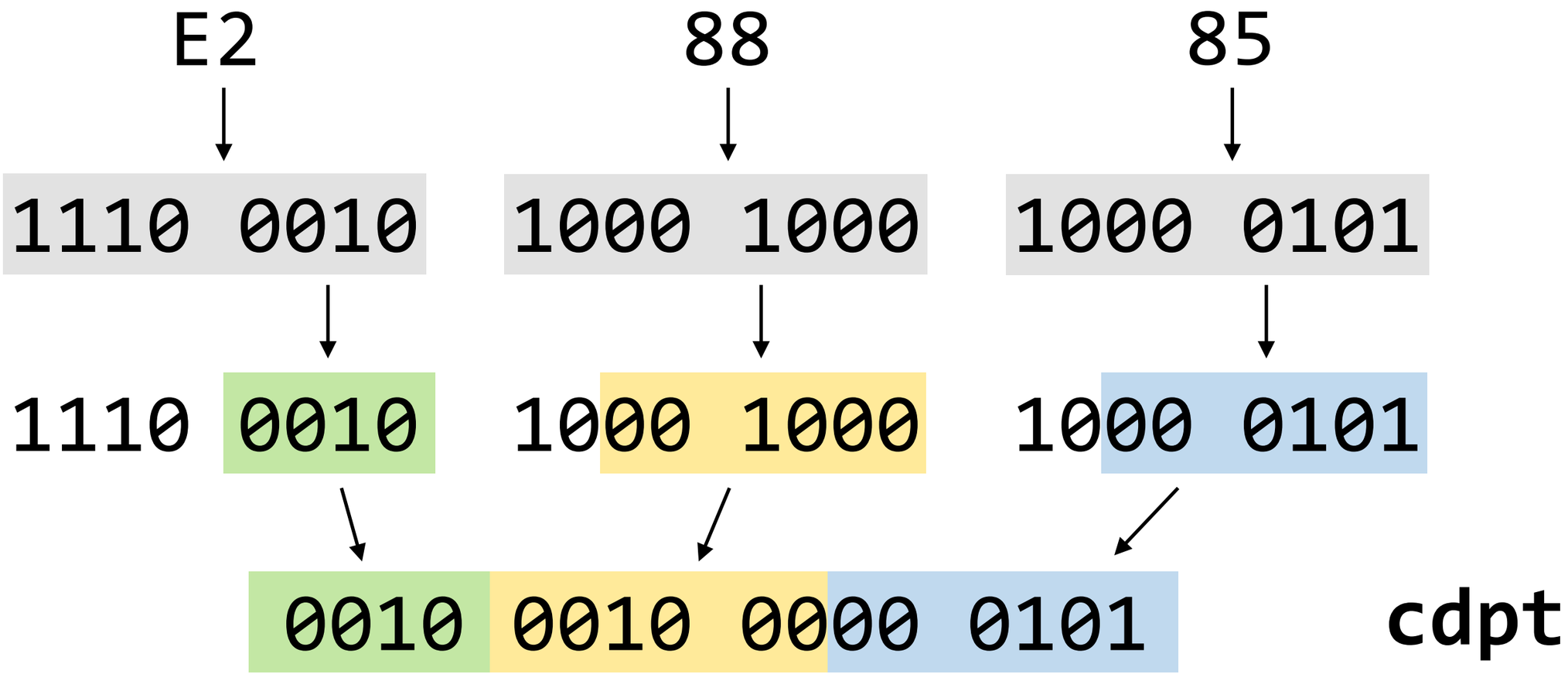
Converting a Single Code Point

```
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;
}
```

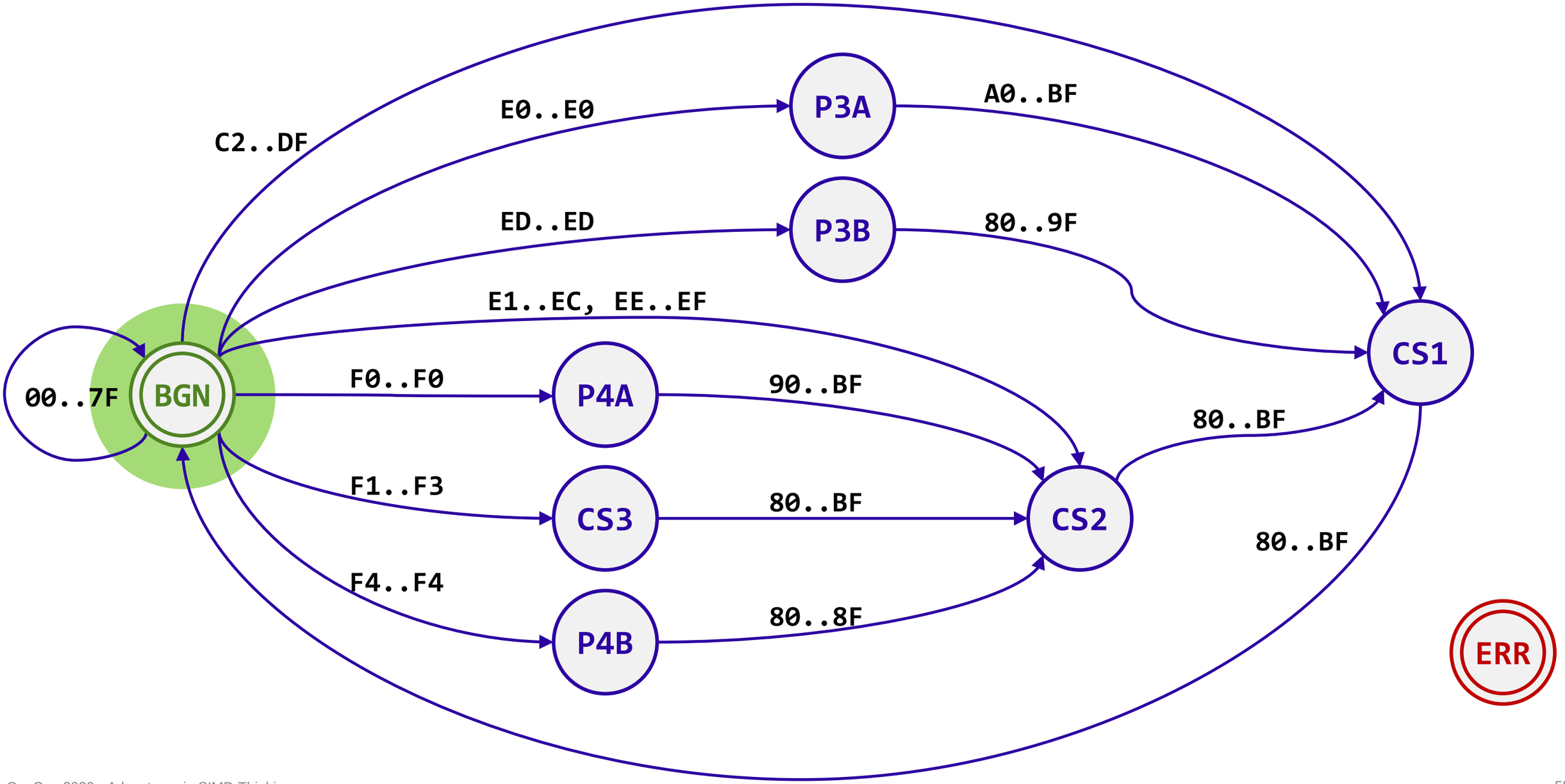
Converting a Single Code Point

```
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 .. }



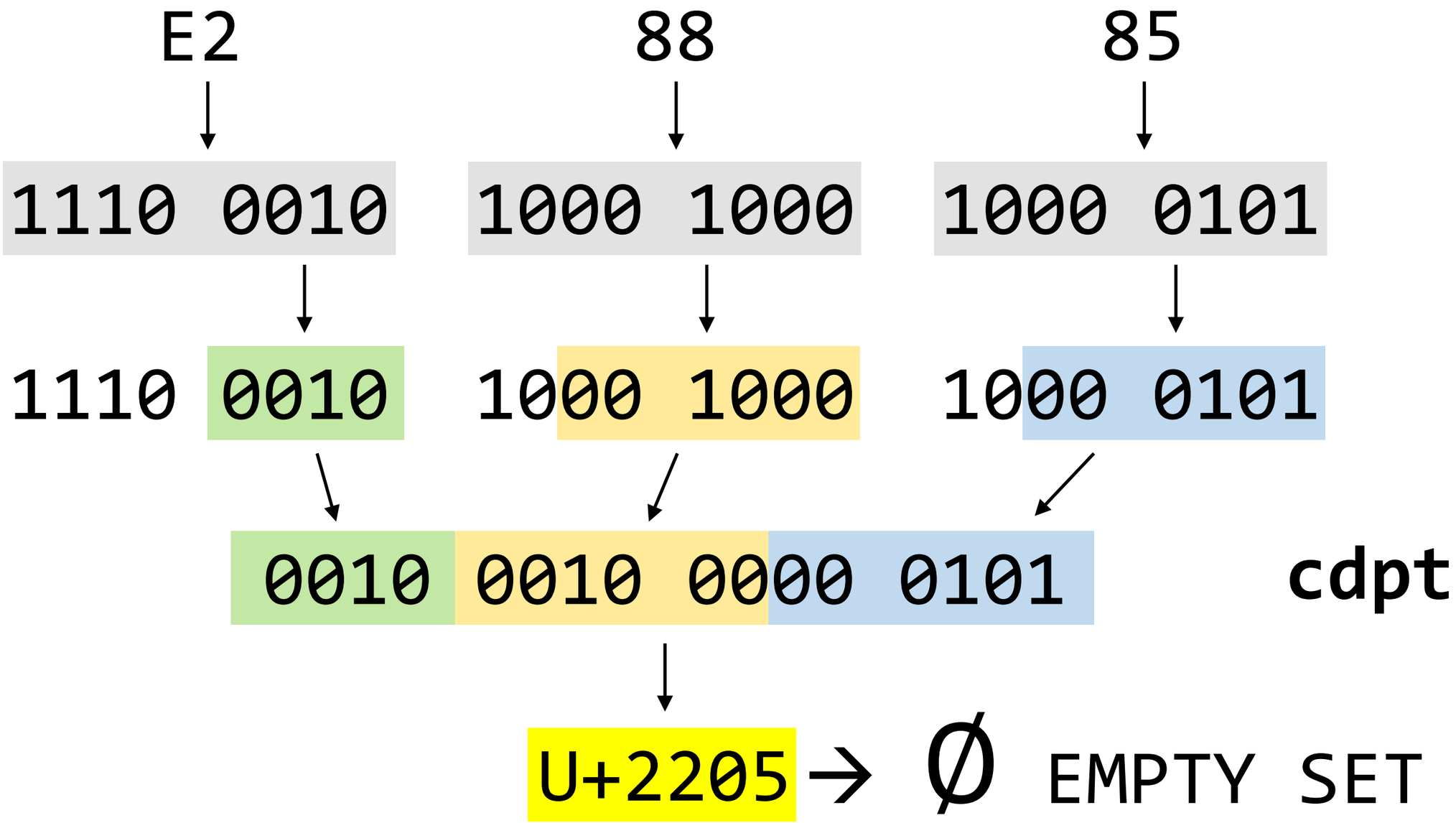
A Decoding Example – { .. E2 88 85 .. }



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 COMPUTING

```
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 COMPUTING

```
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;
            }
        }
    }
    ...
}
```

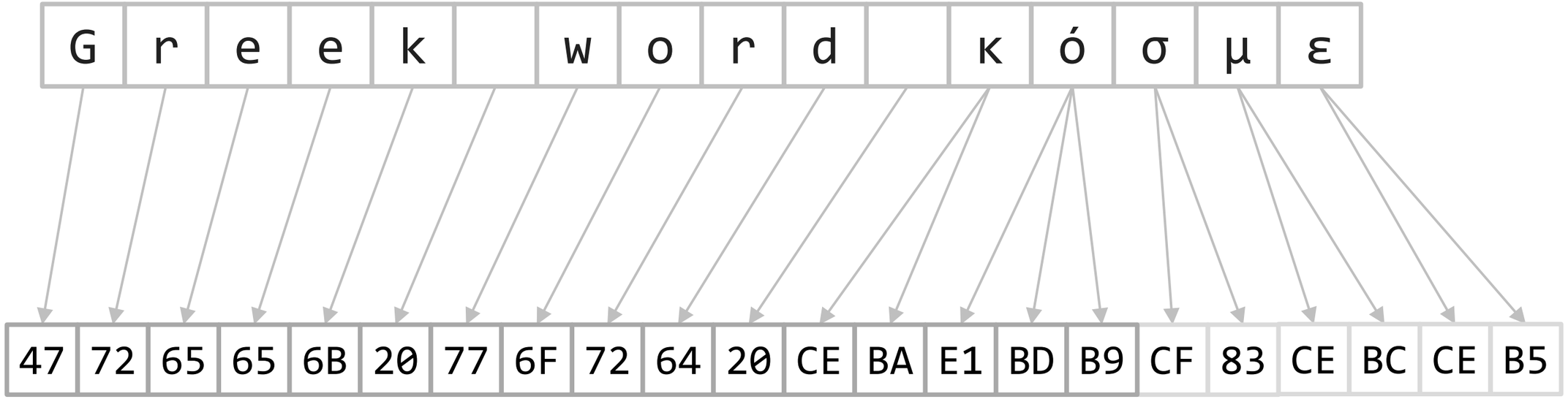
The SSE-Optimized Conversion Algorithm (UTF-8 to UTF-32)

```
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

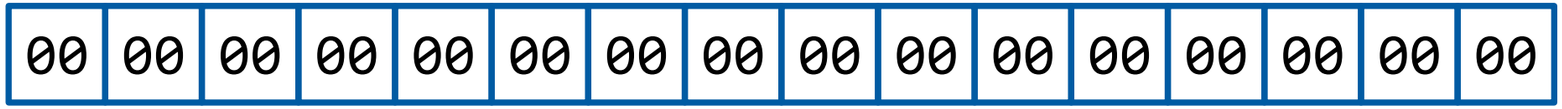
```
KEWB_FORCE_INLINE void
UtfUtils::ConvertAsciiWithSse(char8_t const*& pSrc, char32_t*& pDst) noexcept
{
    __m128i     chunk, half, qrtr, zero;            //- SSE “registers”
    int32_t     mask, incr;                         //- ASCII bit mask and advancement

    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)
```

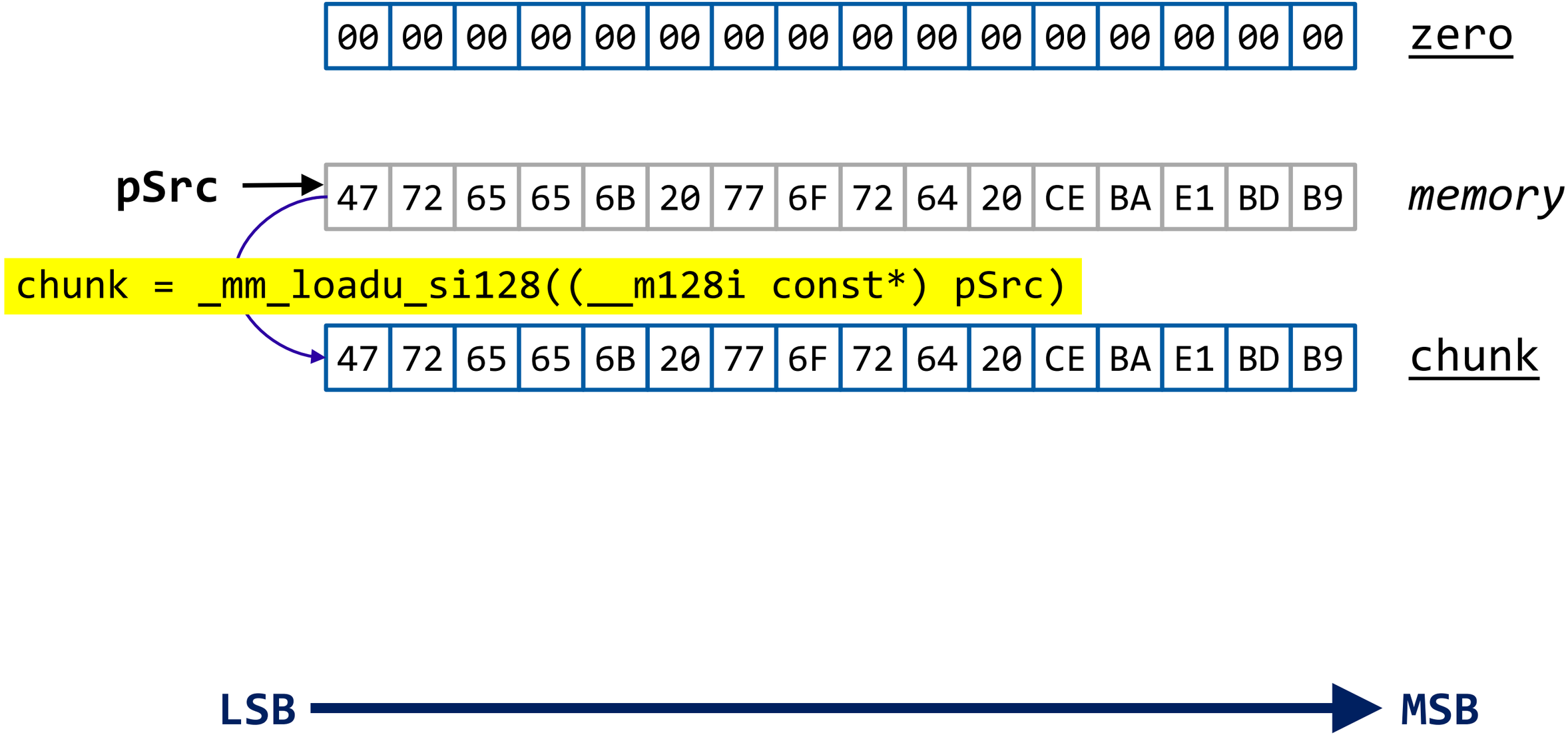


LSB → MSB

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

    ...
}
```

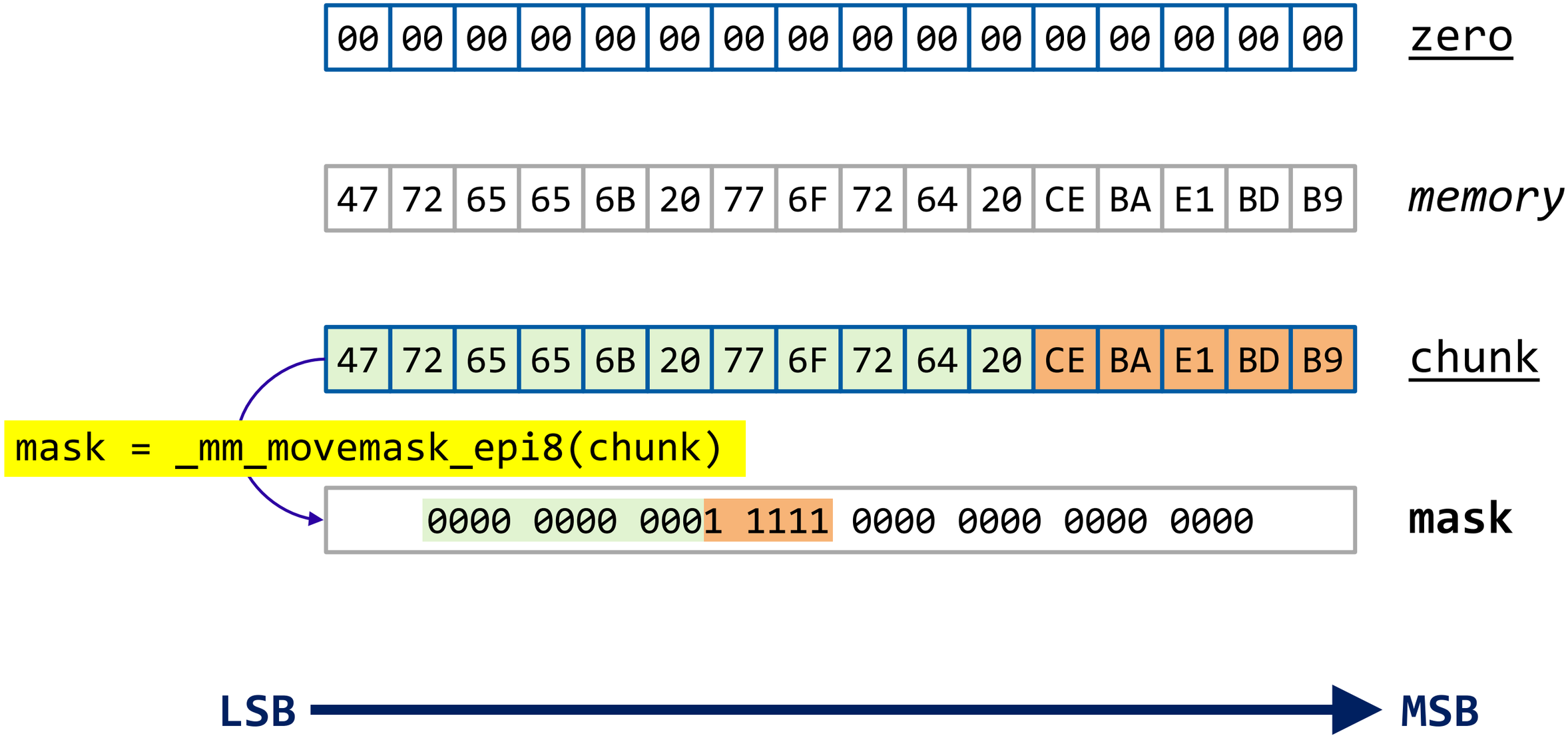
Converting ASCII Character Runs – SSE Example



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

    ...
}
```

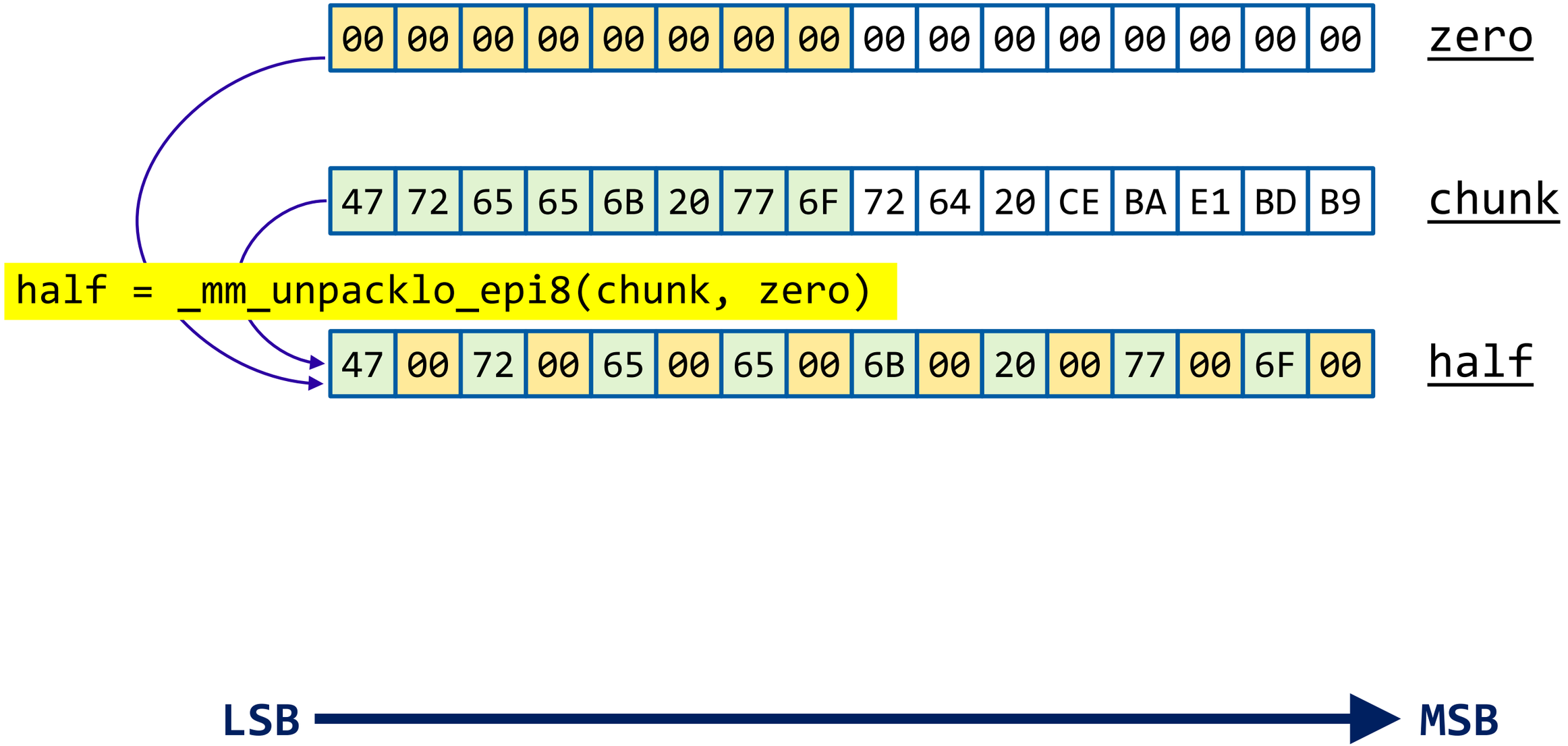
Converting ASCII Character Runs – SSE Example



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

    ...
}
```

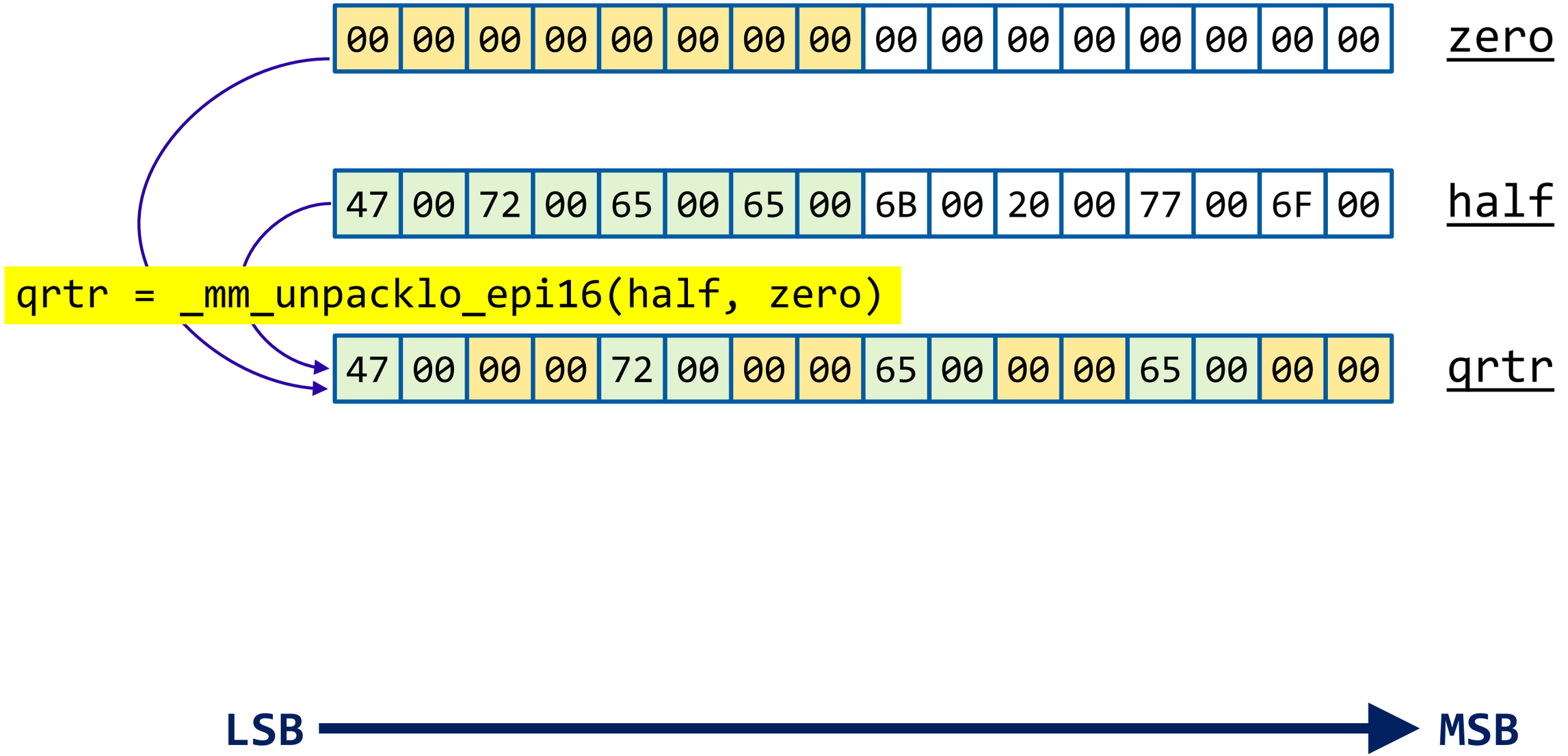
Converting ASCII Character Runs – SSE Example



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

    ...
}
```

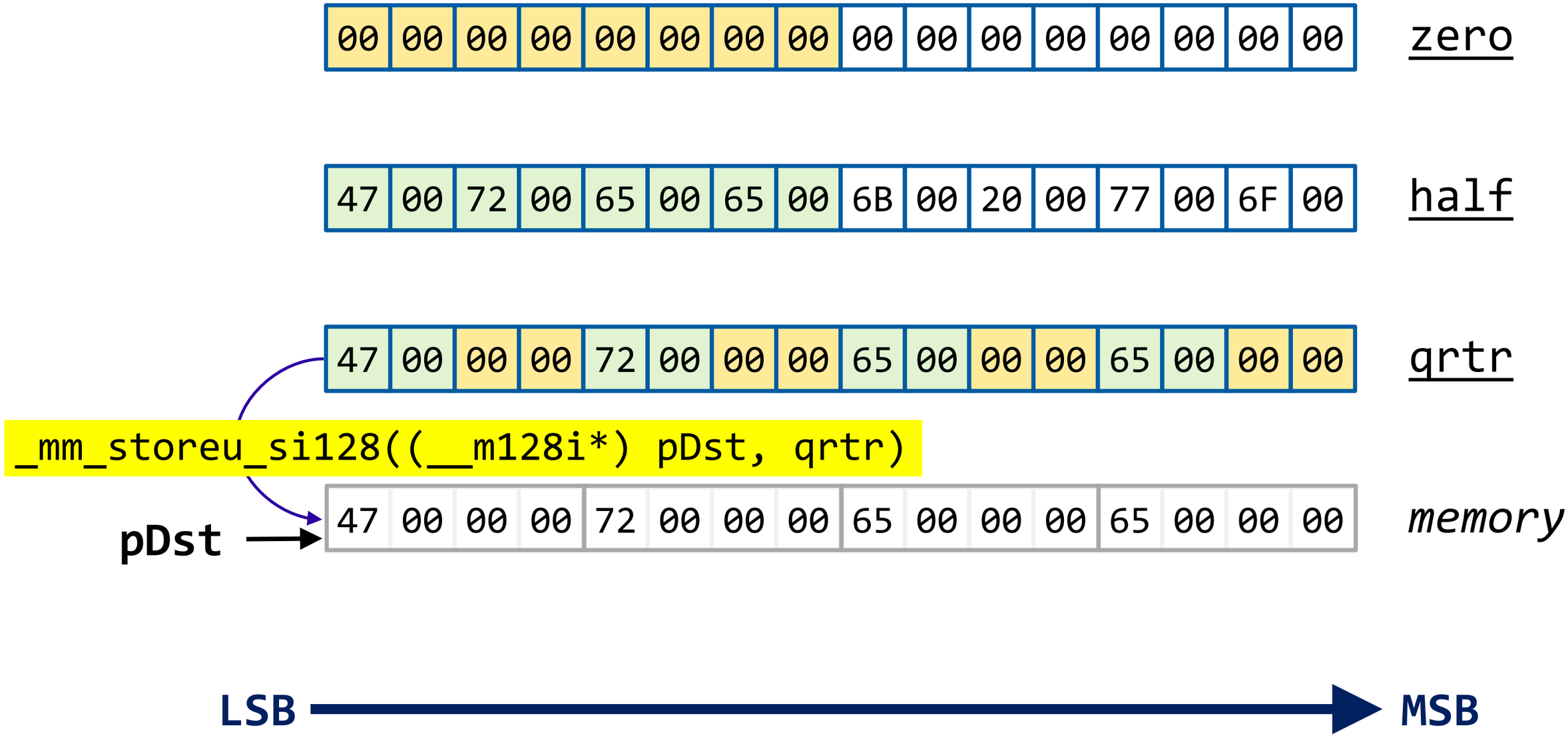
Converting ASCII Character Runs – SSE Example



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

    ...
}
```

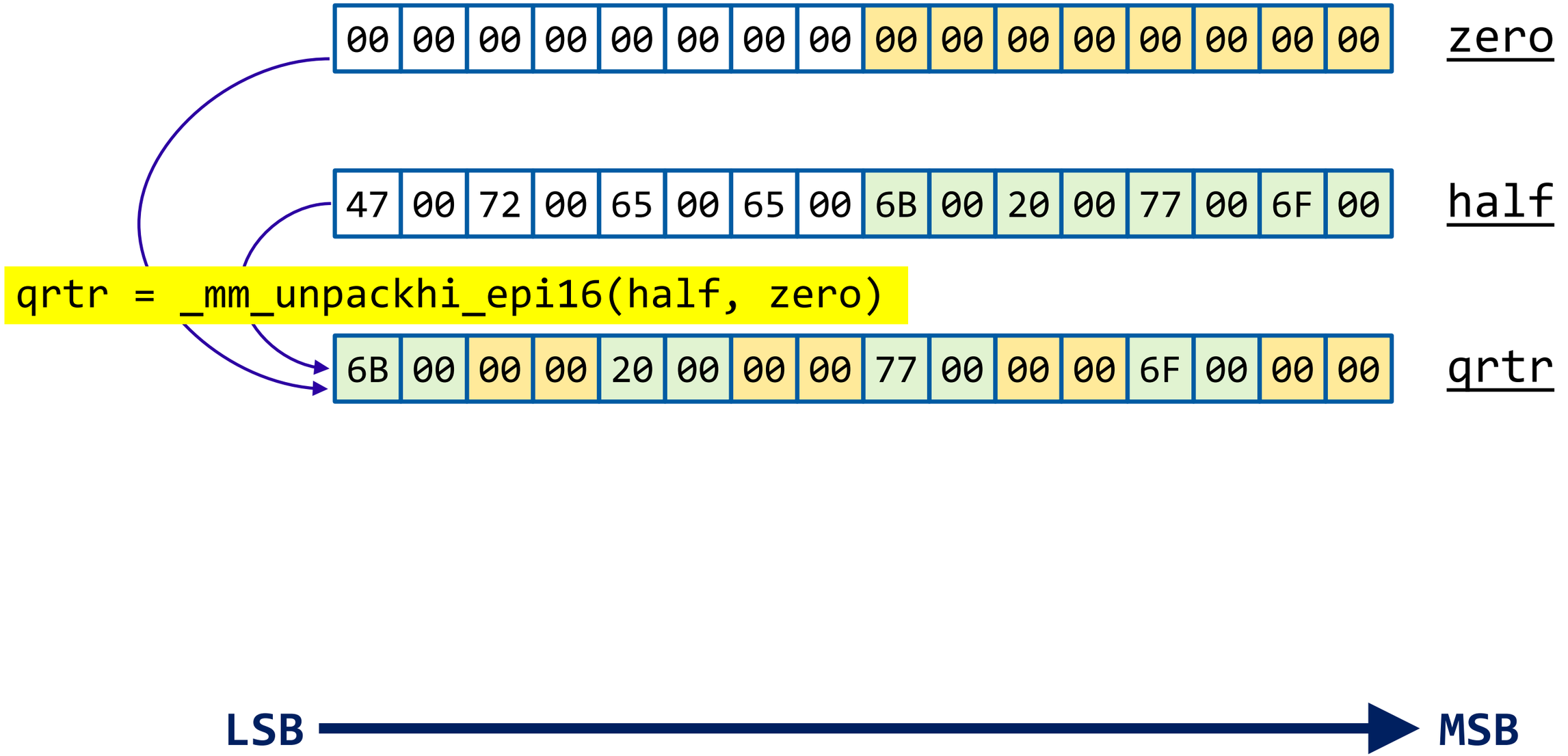
Converting ASCII Character Runs – SSE Example



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

    ...
}
```

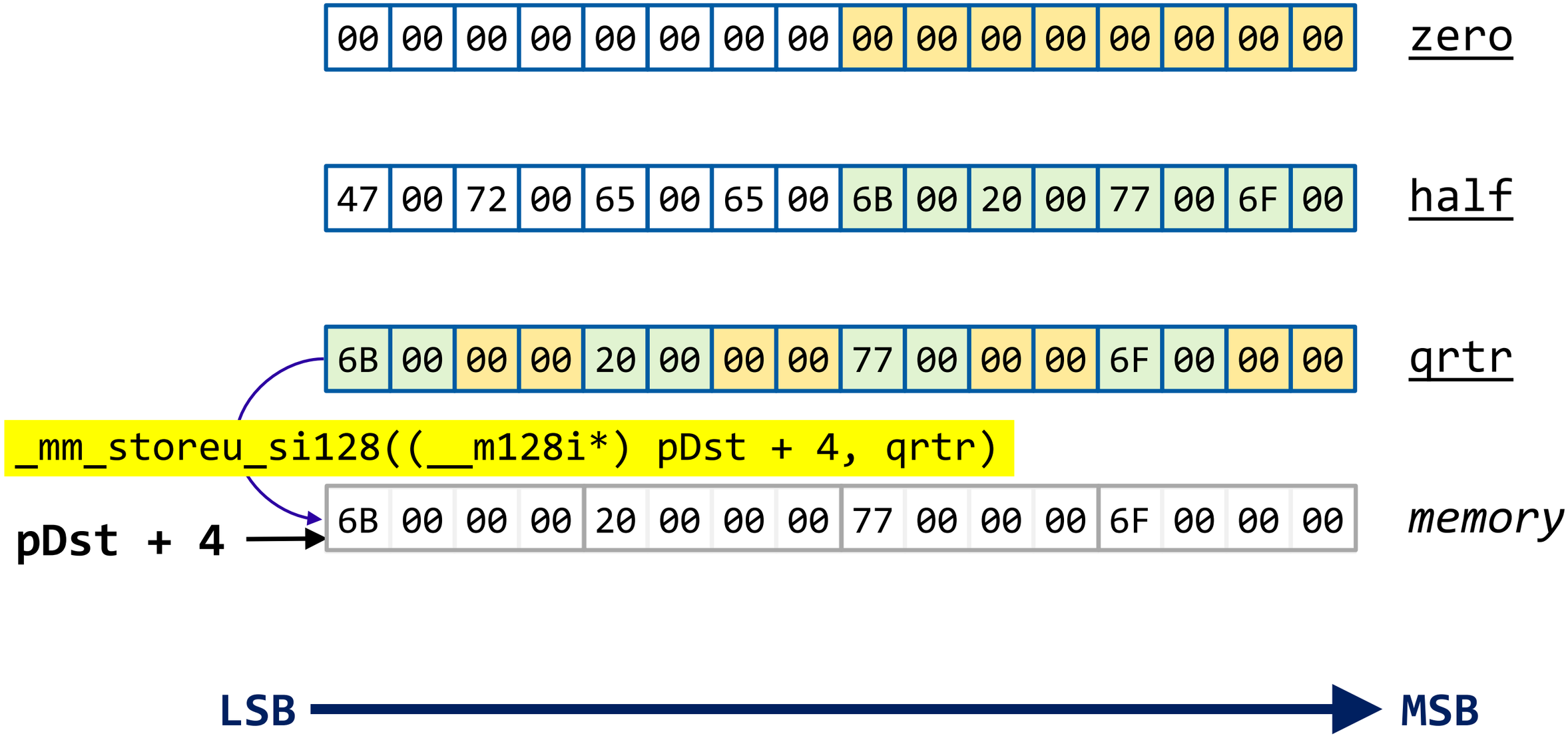
Converting ASCII Character Runs – SSE Example



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

    ...
}
```

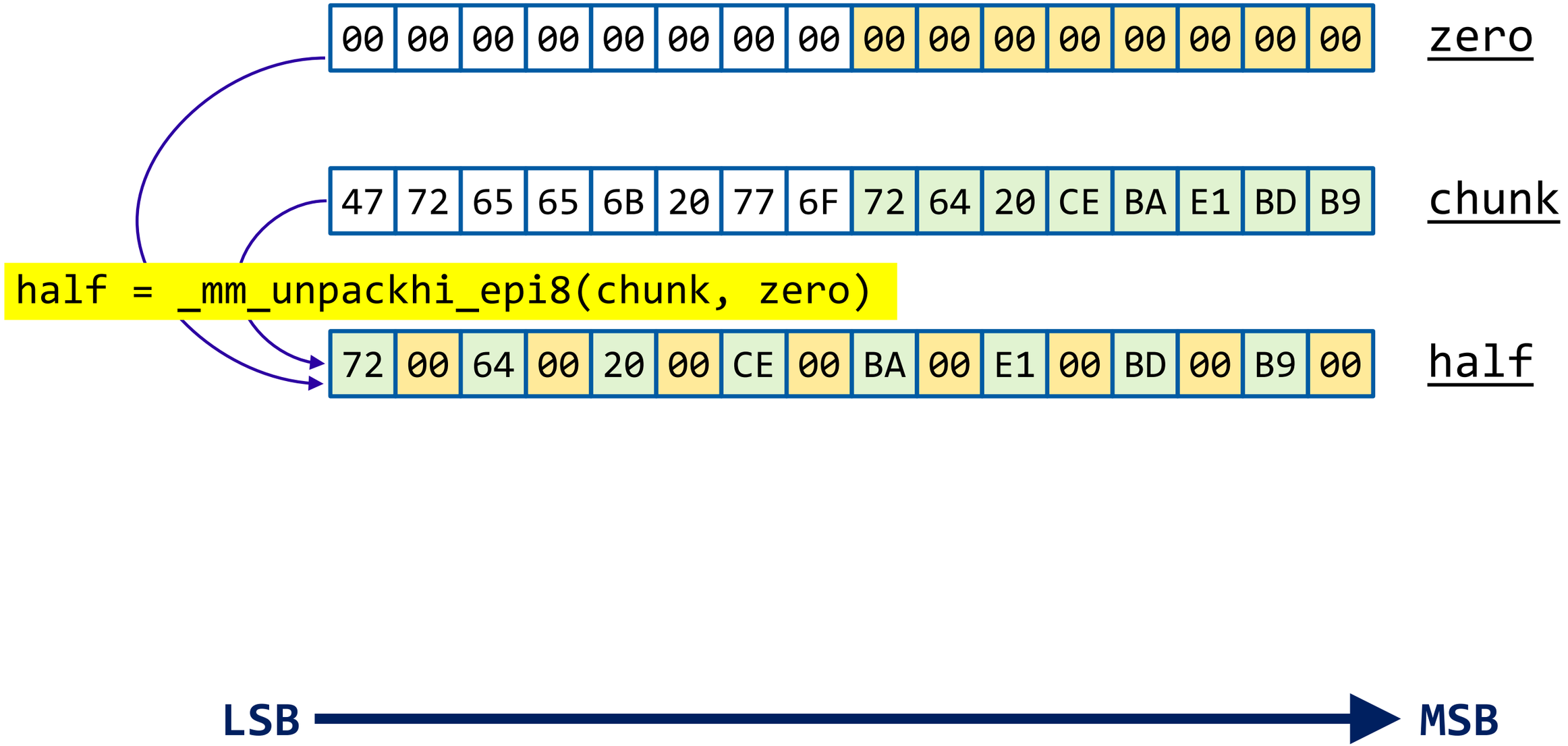
Converting ASCII Character Runs – SSE Example



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

    ...
}
```

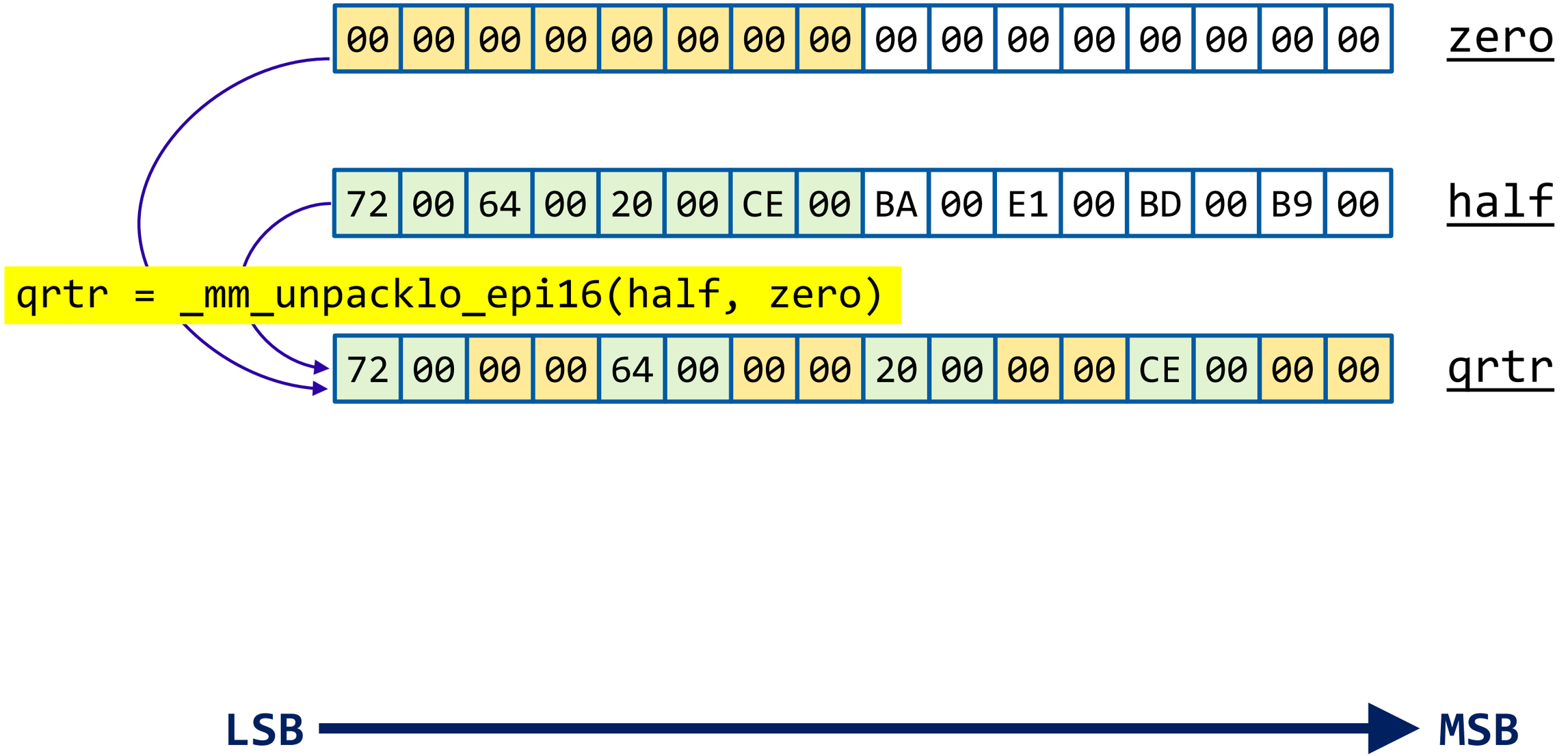
Converting ASCII Character Runs – SSE Example



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

    ...
}
```

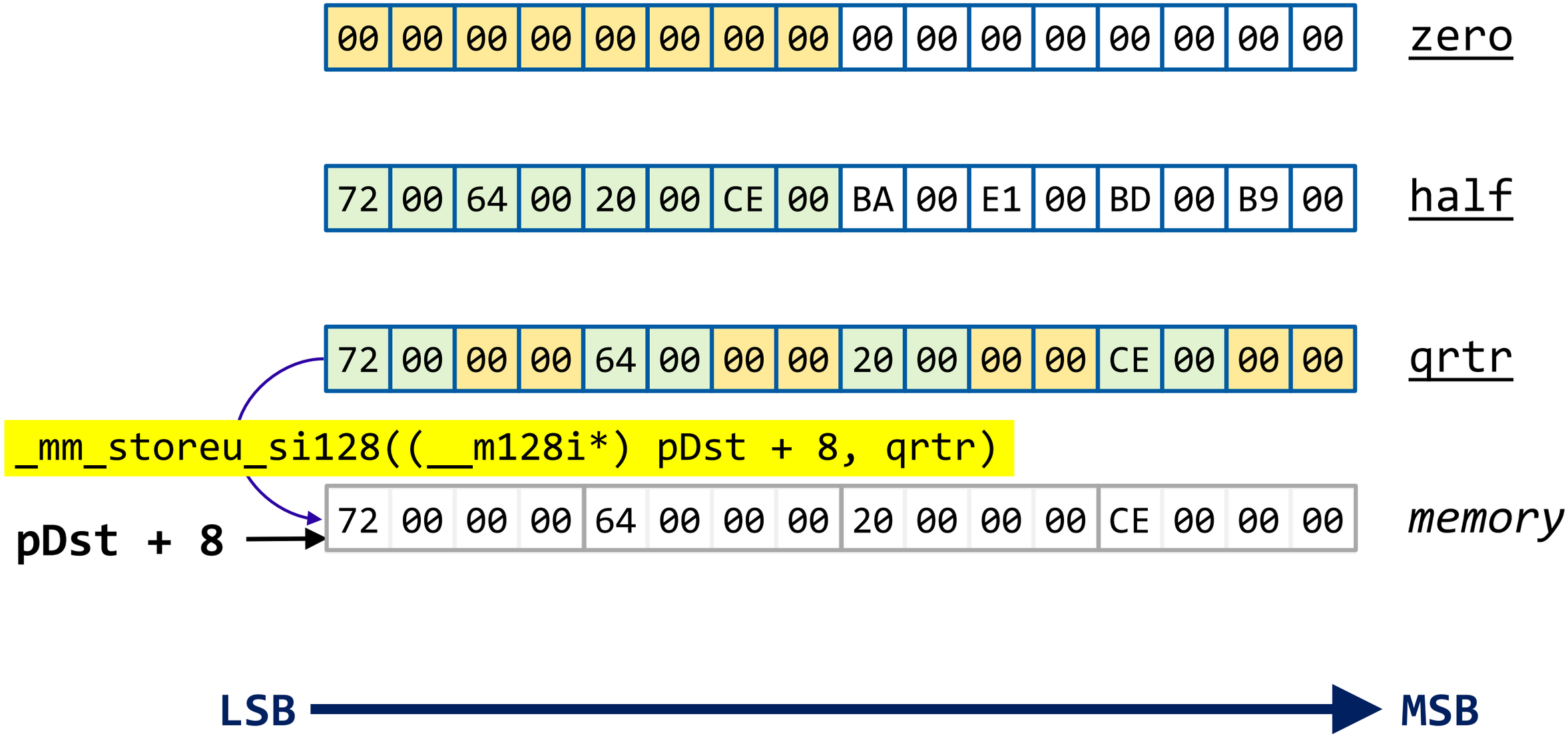
Converting ASCII Character Runs – SSE Example



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

    ...
}
```

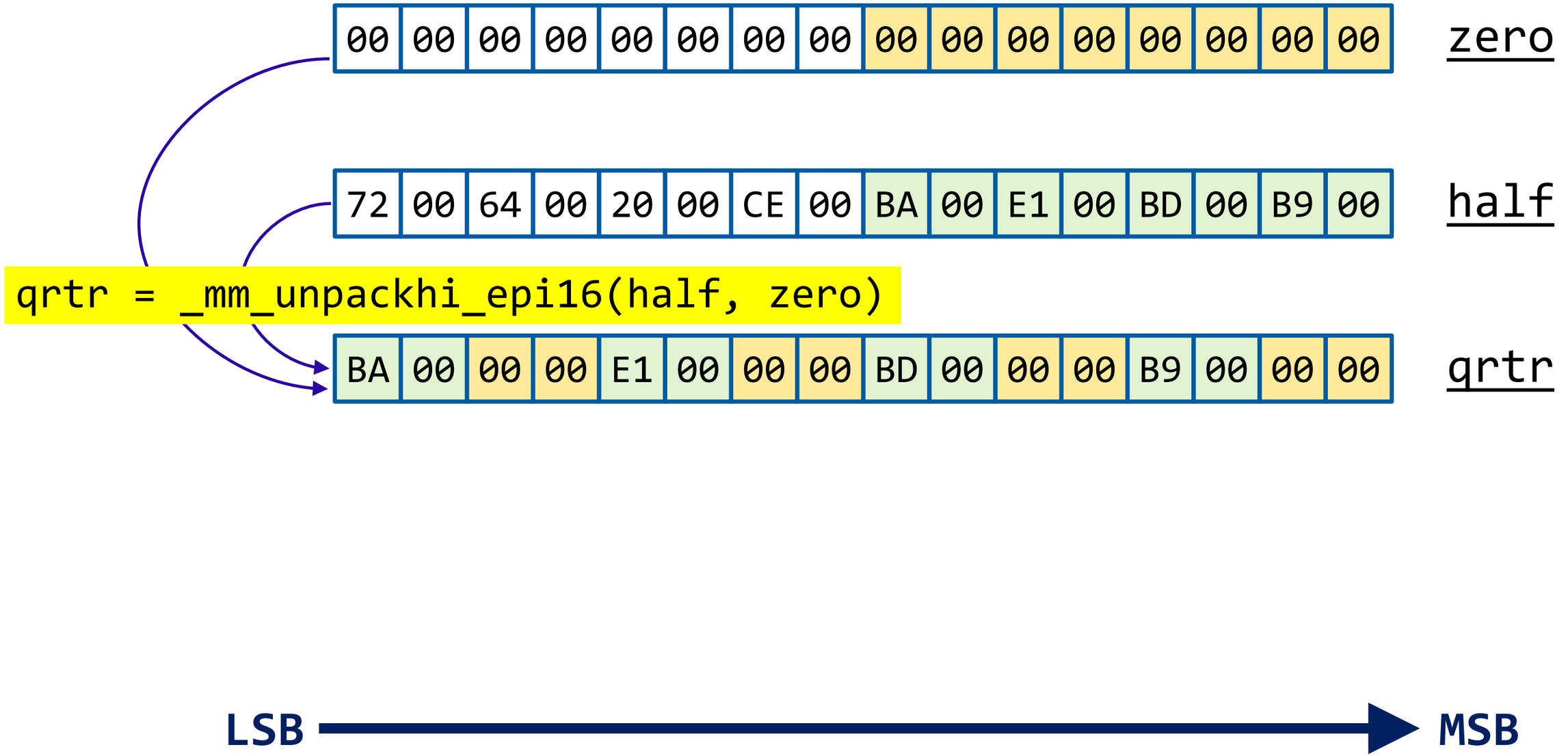
Converting ASCII Character Runs – SSE Example



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

    ...
}
```

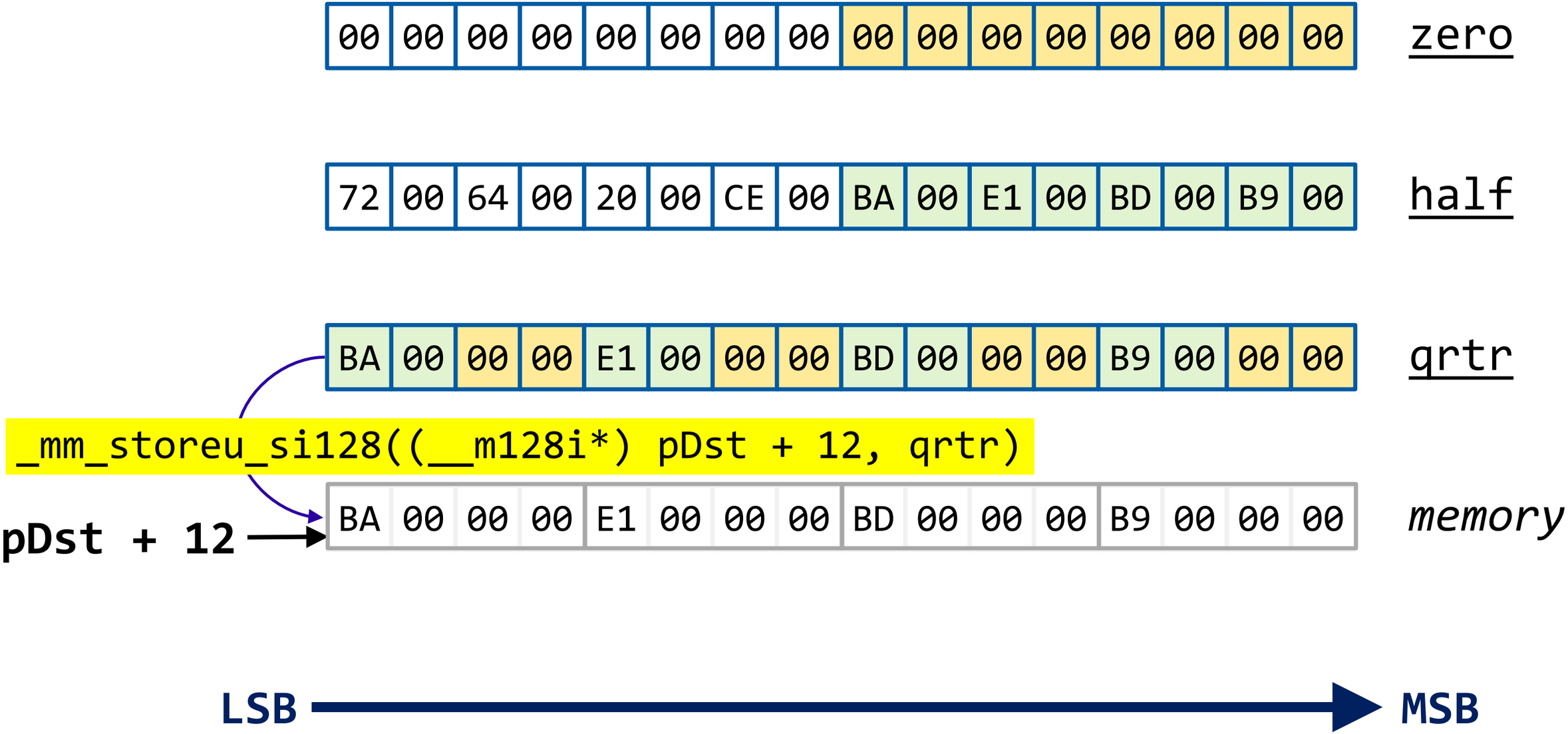
Converting ASCII Character Runs – SSE Example



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



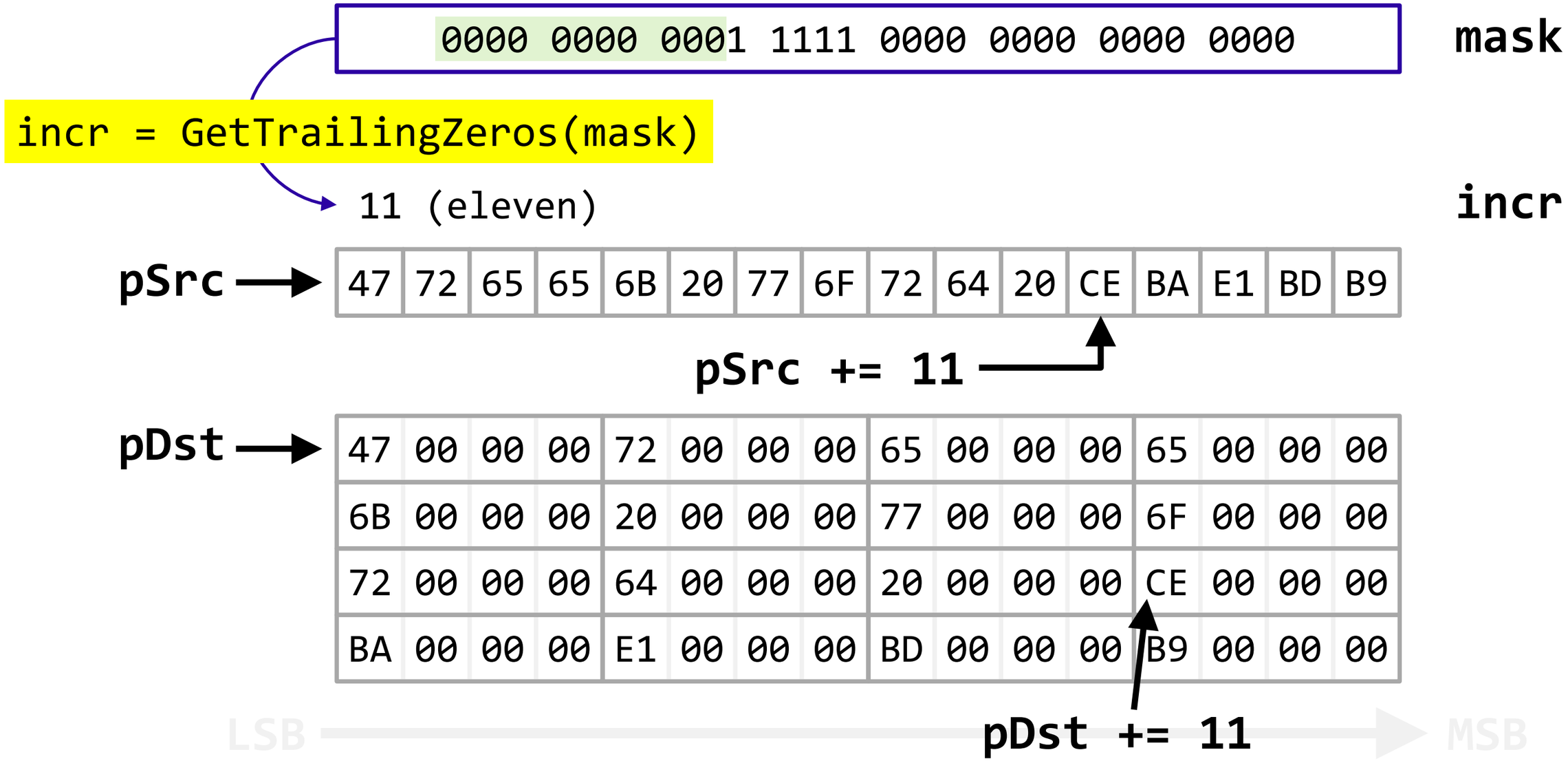
Converting ASCII Character Runs – SSE Code

```
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



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;
            }
        }
    }
    ...
}
```

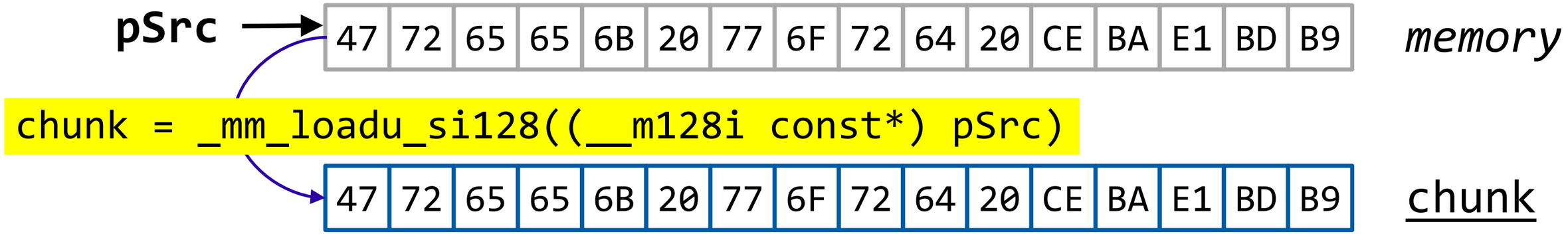
Converting ASCII Character Runs – AVX Example

```
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;                                      //- 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

```
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

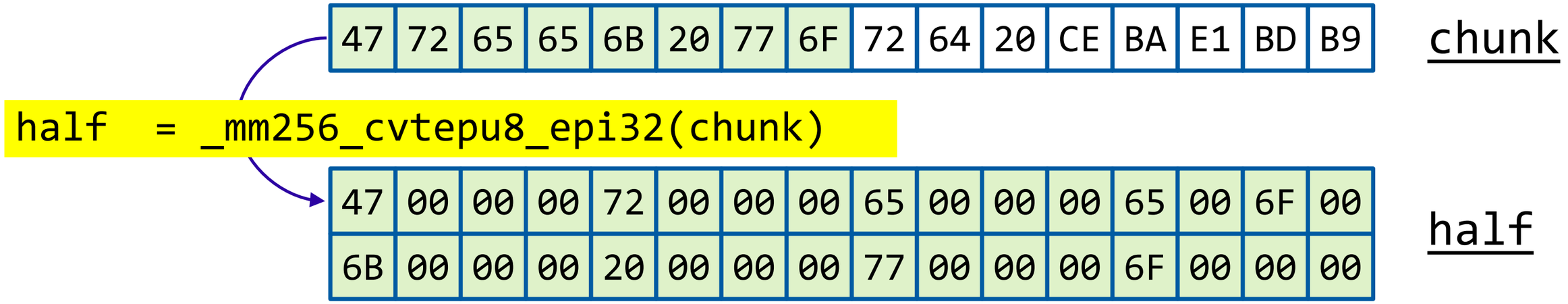


LSB → MSB

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

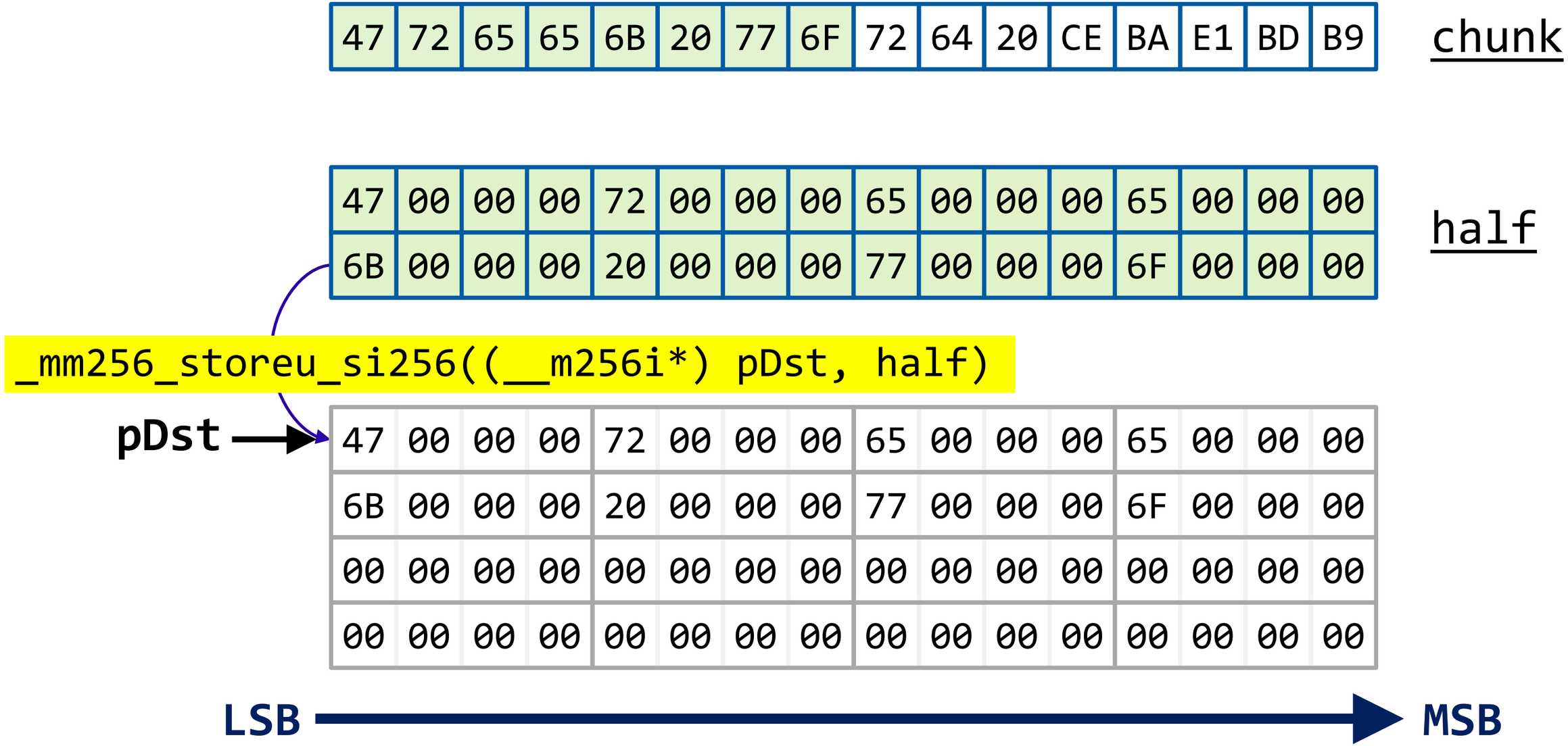


LSB → MSB

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

            ...
}
```

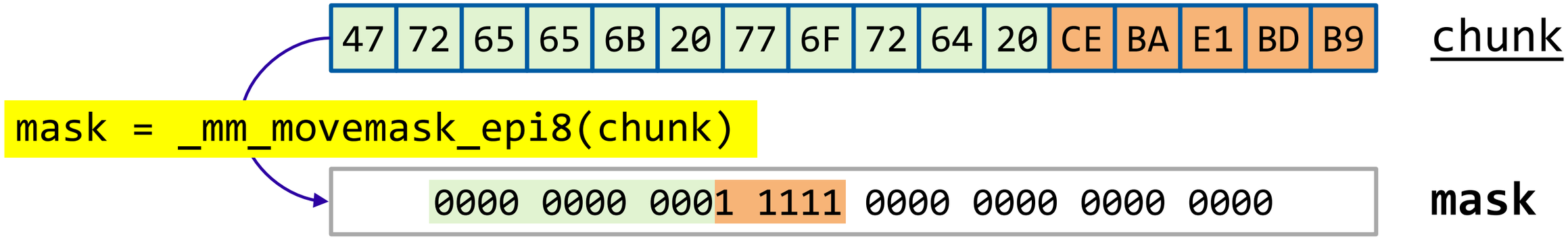
Converting ASCII Character Runs – AVX Example



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

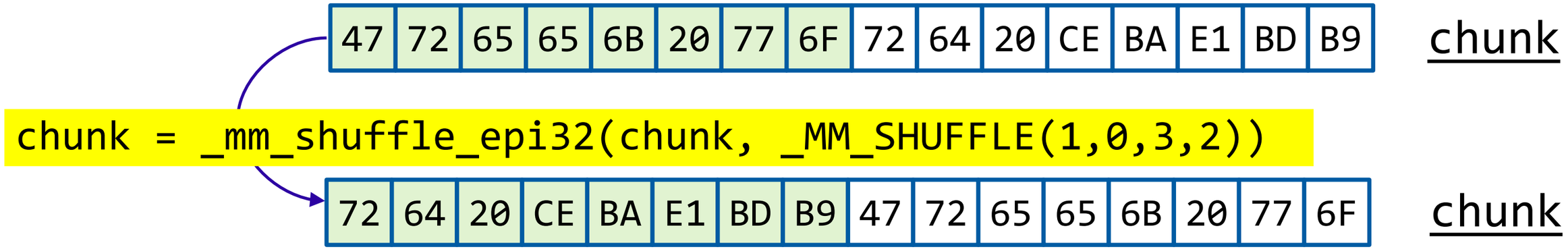


LSB → MSB

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

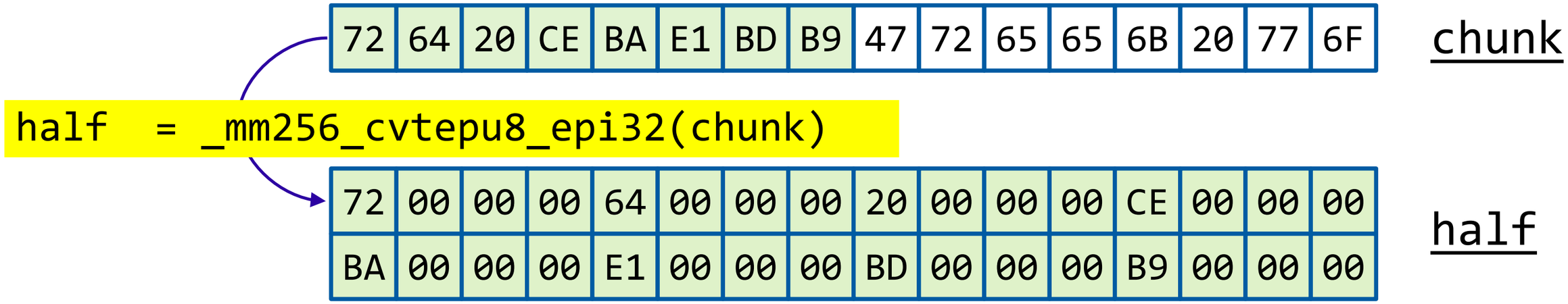


LSB → **MSB**

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

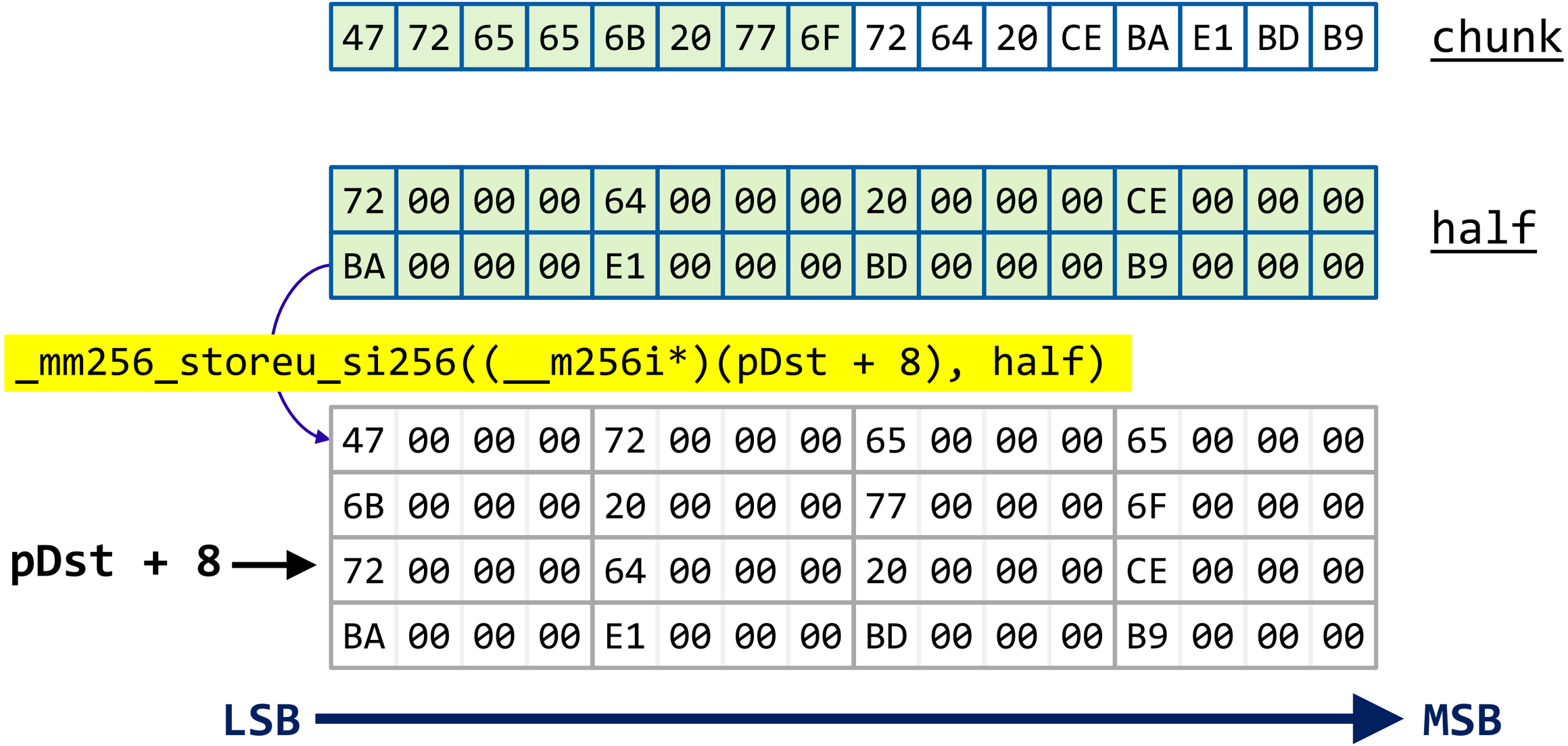


LSB → **MSB**

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

            ...
}
```

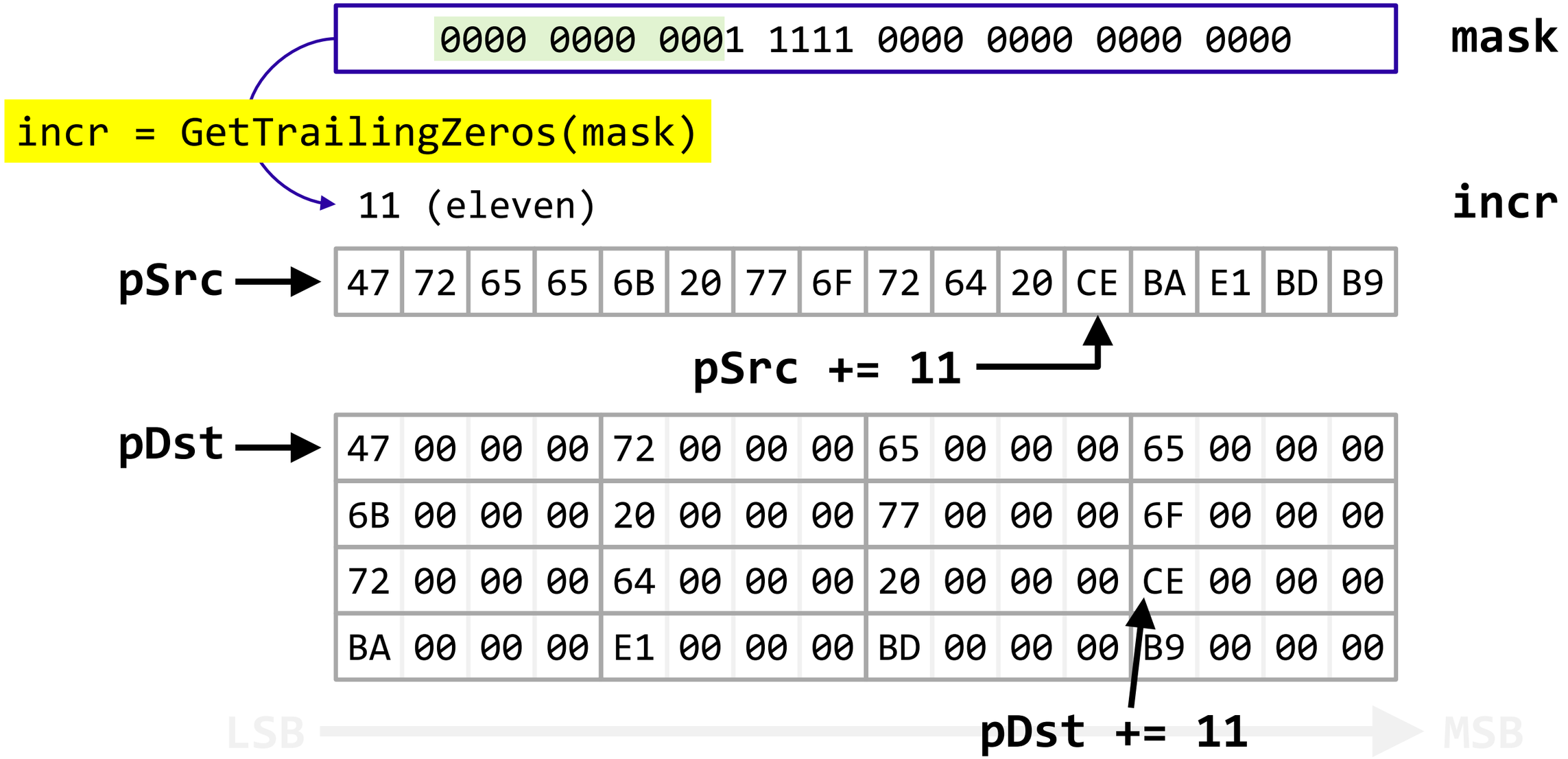
Converting ASCII Character Runs – SSE Example



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



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

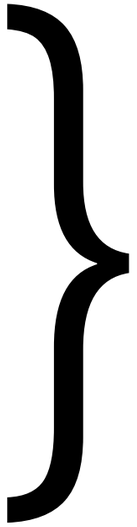
Testing Methodology – Platforms

- 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`

Testing Methodology – Input Data

- Nine input files
  - `english_wiki.txt`
  - `chinese_wiki.txt`
  - `hindi_wiki.txt`
  - `portuguese_wiki.txt`
  - `russian_wiki.txt`
  - `swedish_wiki.txt`



Taken directly from wikipedia.org

  - `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)

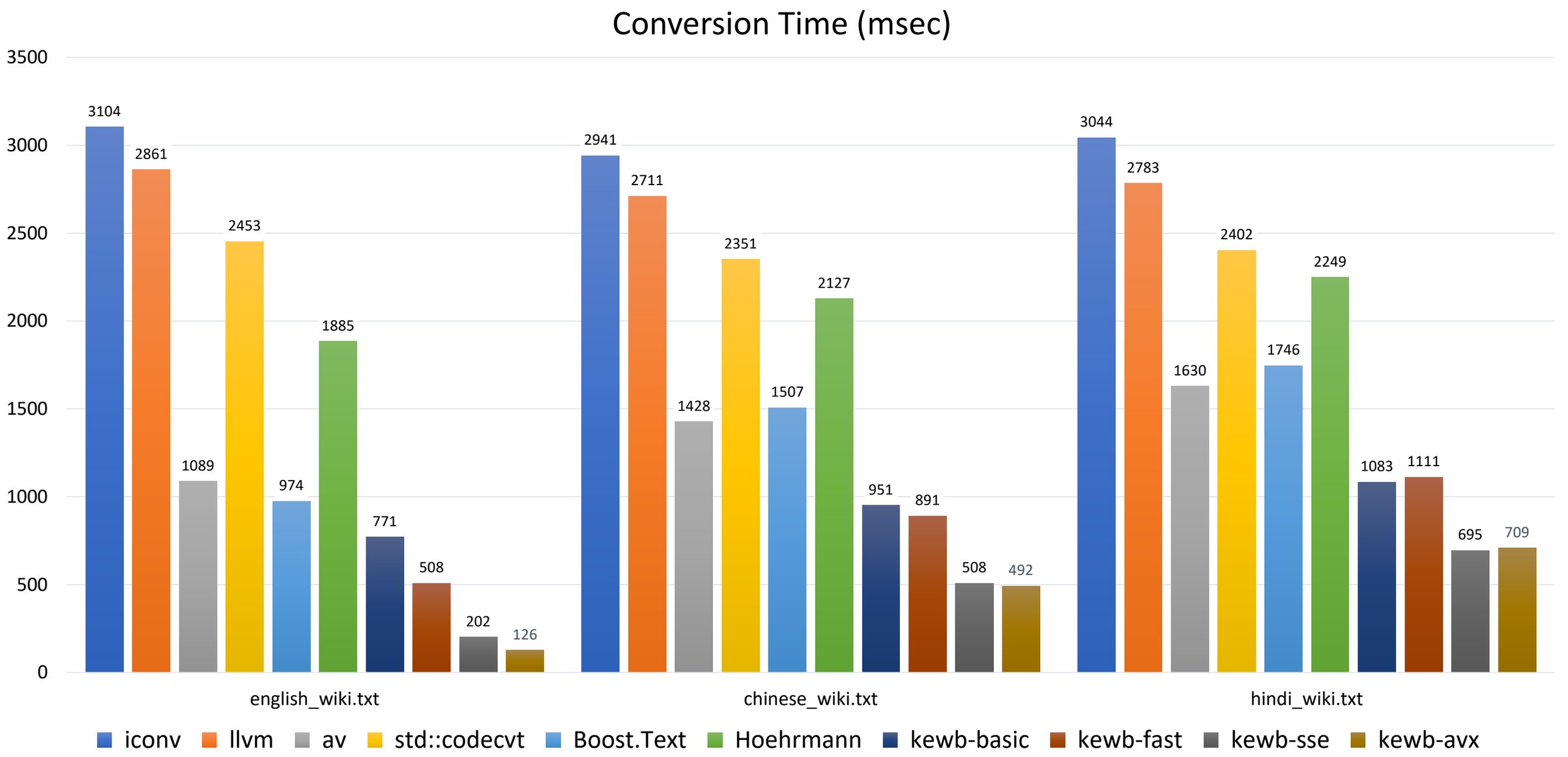
Testing Methodology – Reference Libraries

- 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

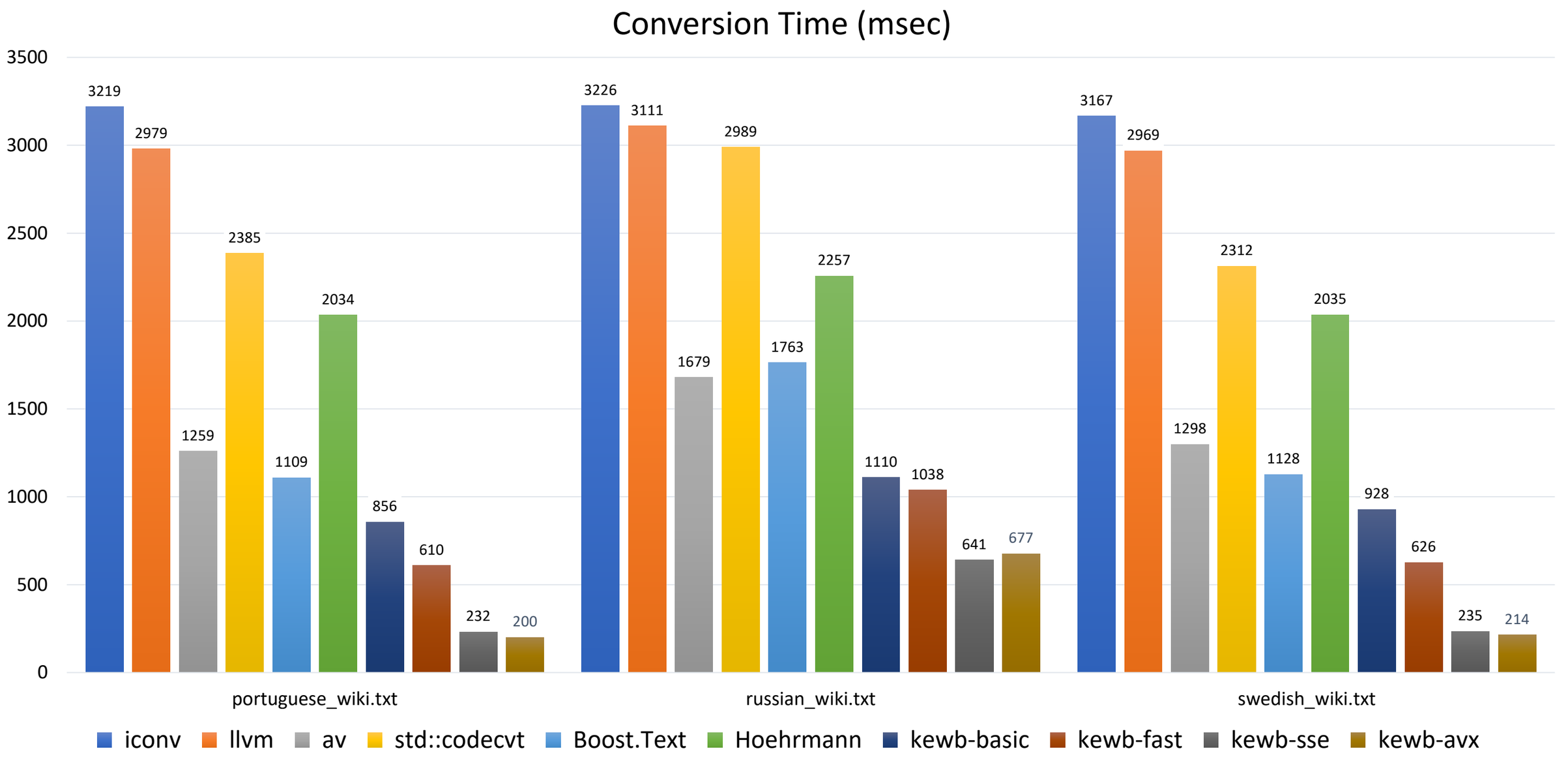
Testing Methodology - Timings

- 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()`

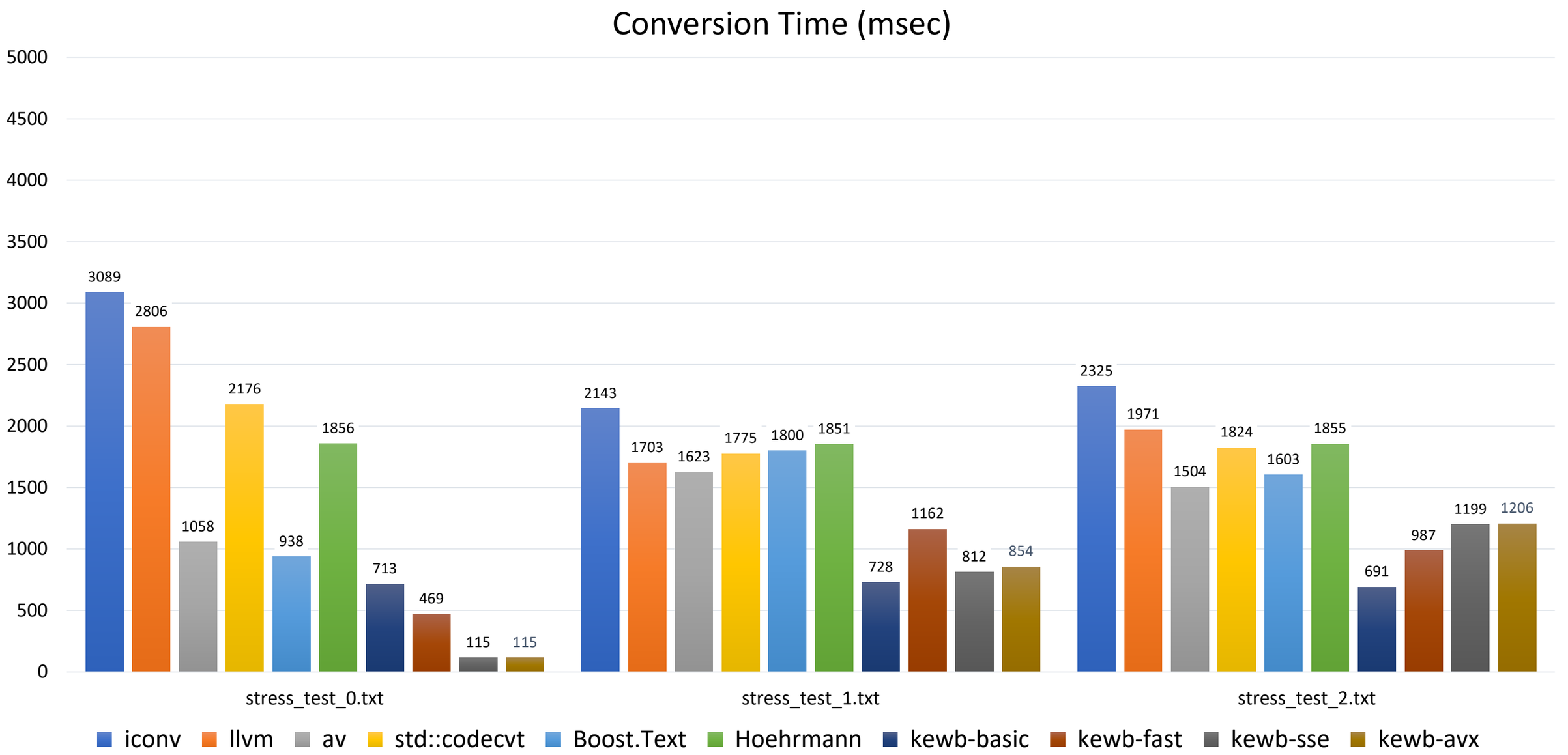
GCC 9.3 – Ubuntu 18.04 – Core i9 – UTF-32



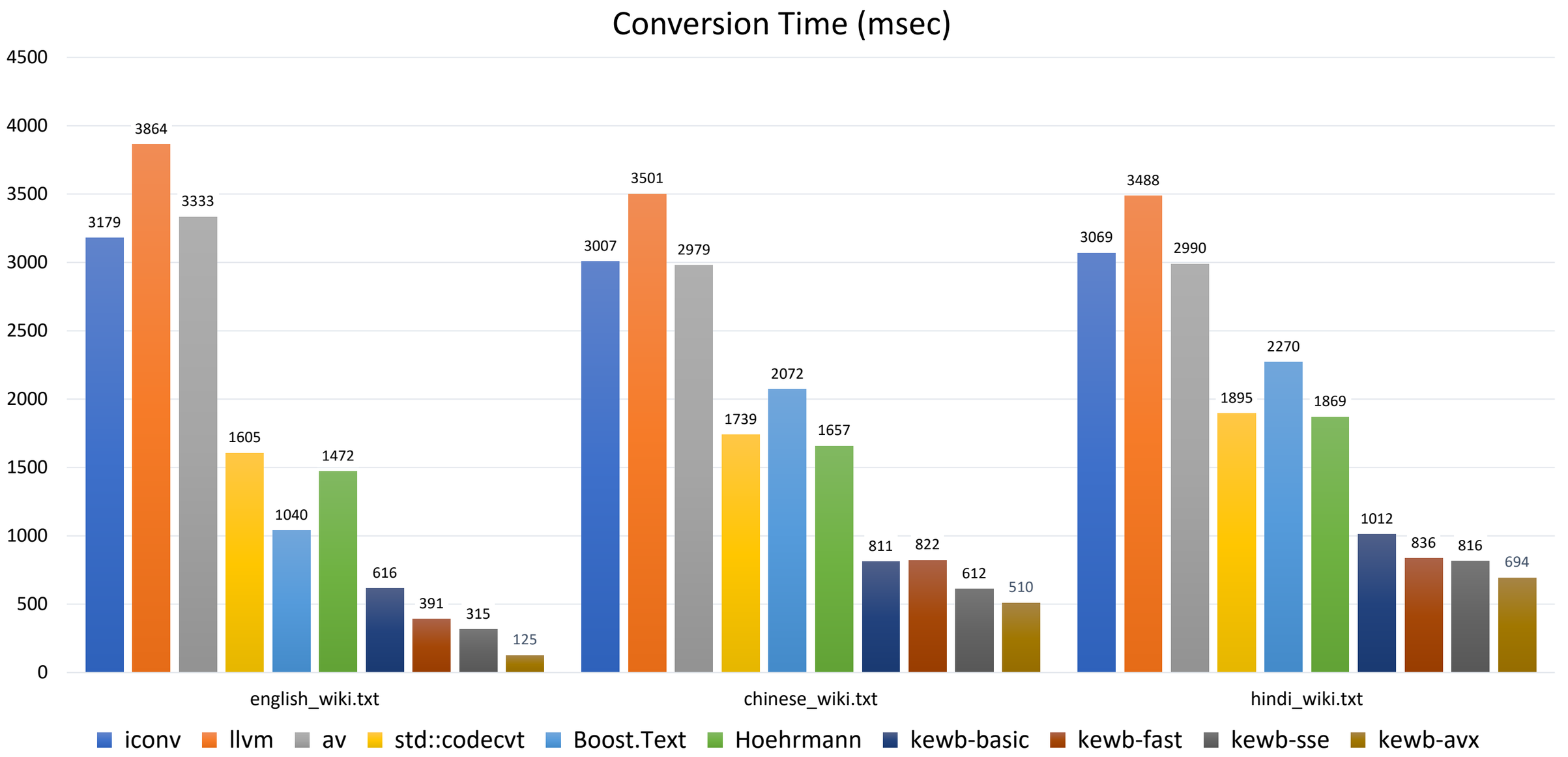
GCC 9.3 – Ubuntu 18.04 – Core i9 – UTF-32



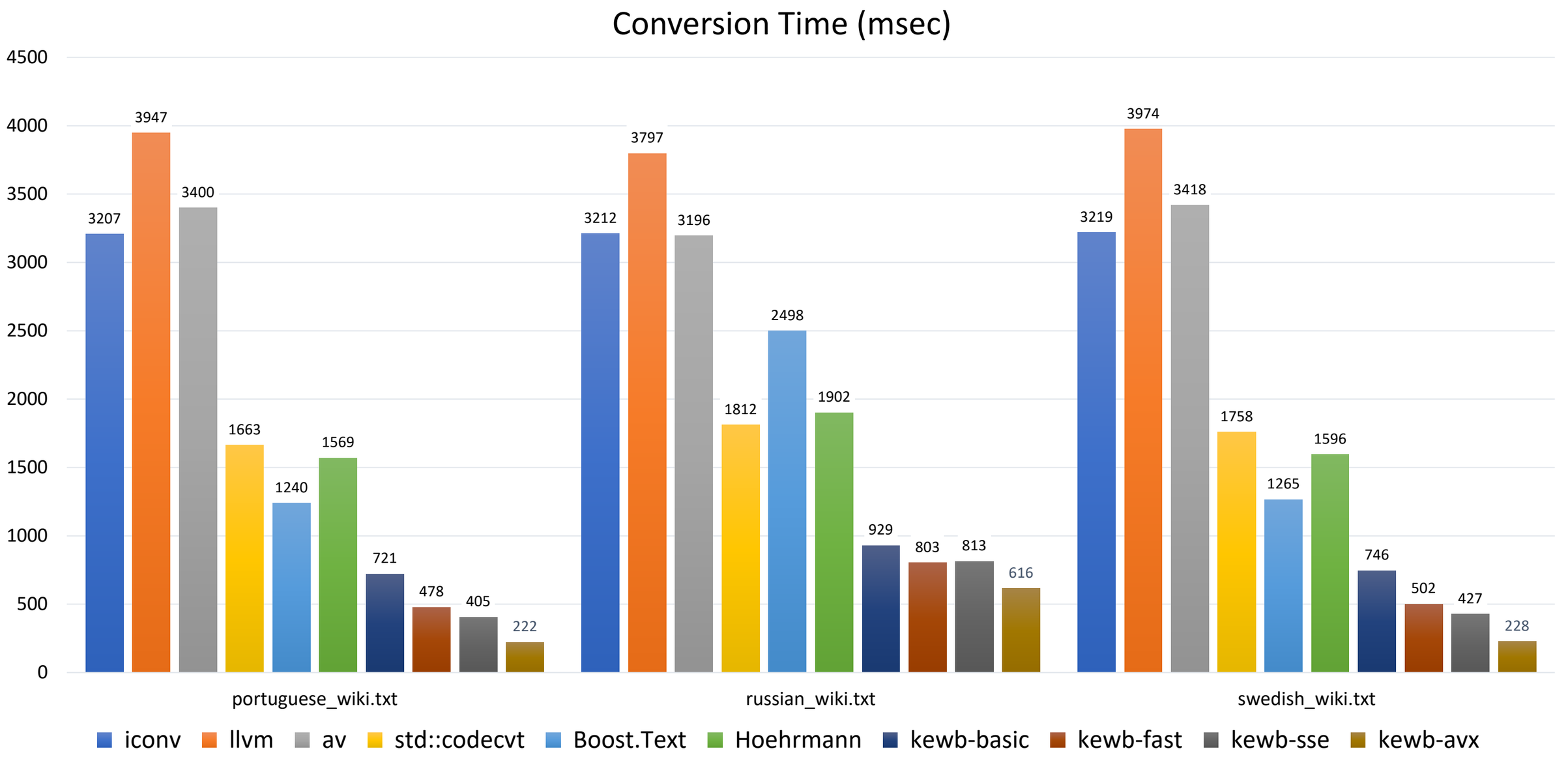
GCC 9.3 – Ubuntu 18.04 – Core i9 – UTF-32



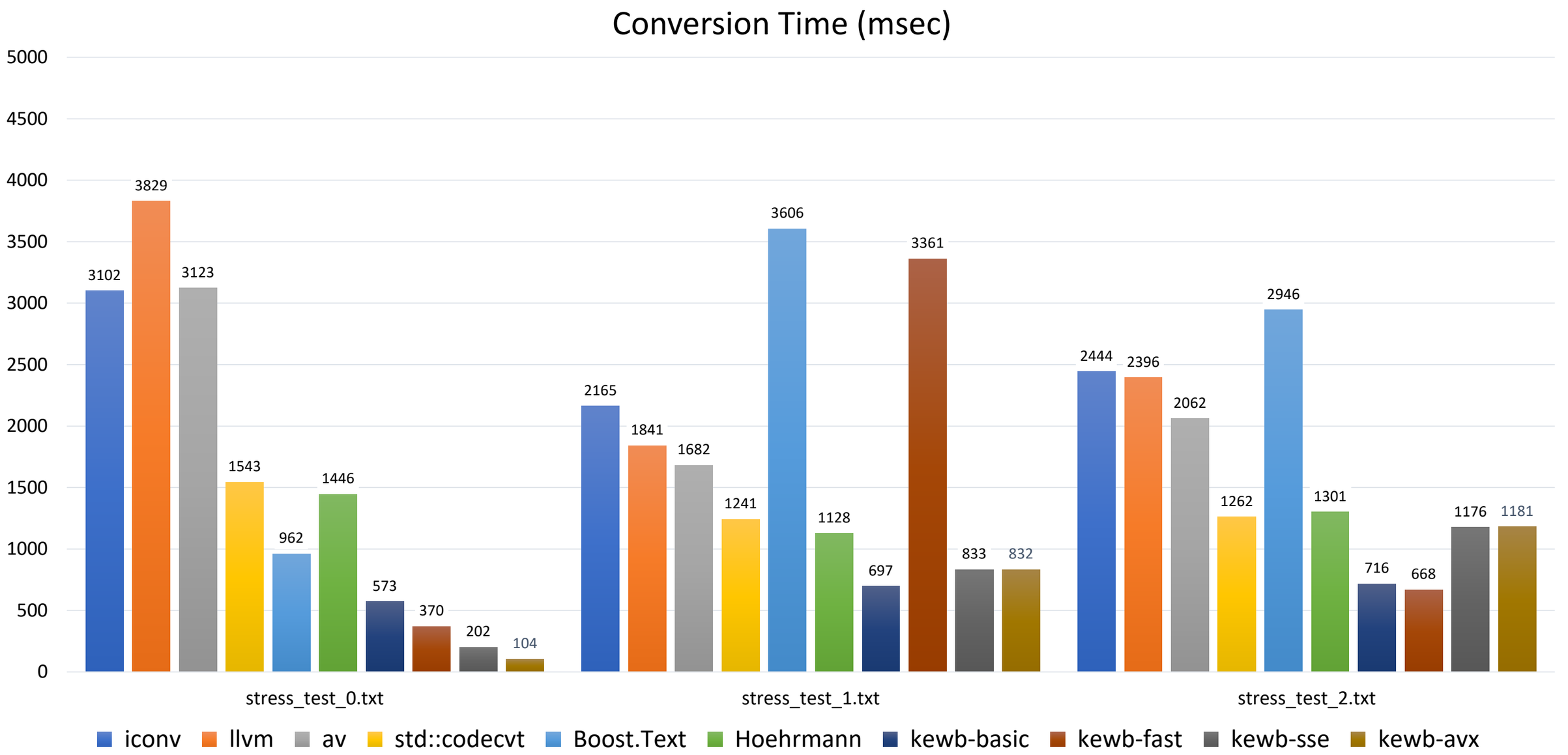
Clang 10.0.1 – Ubuntu 18.04 – Core i9 – UTF-32



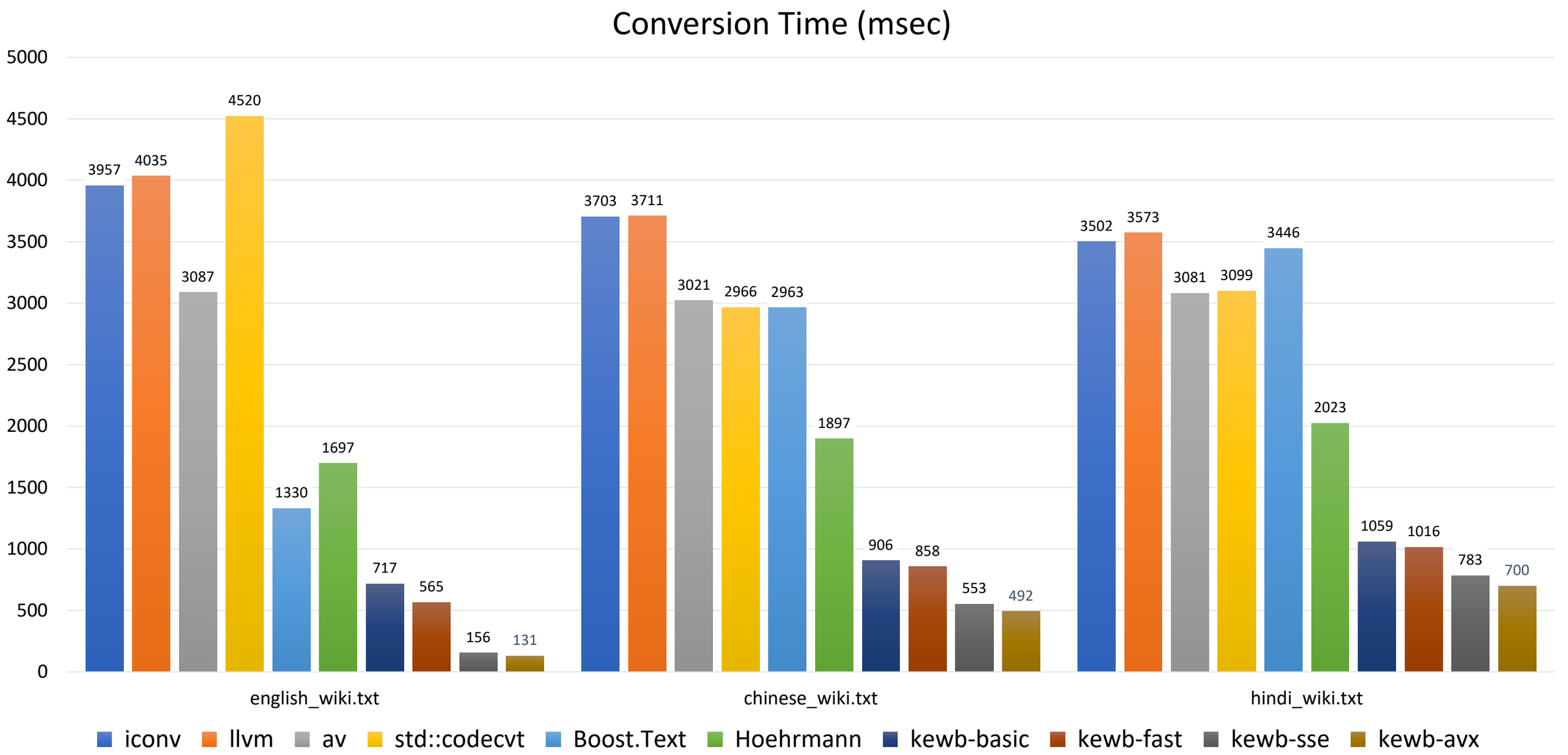
Clang 10.0.1 – Ubuntu 18.04 – Core i9 – UTF-32



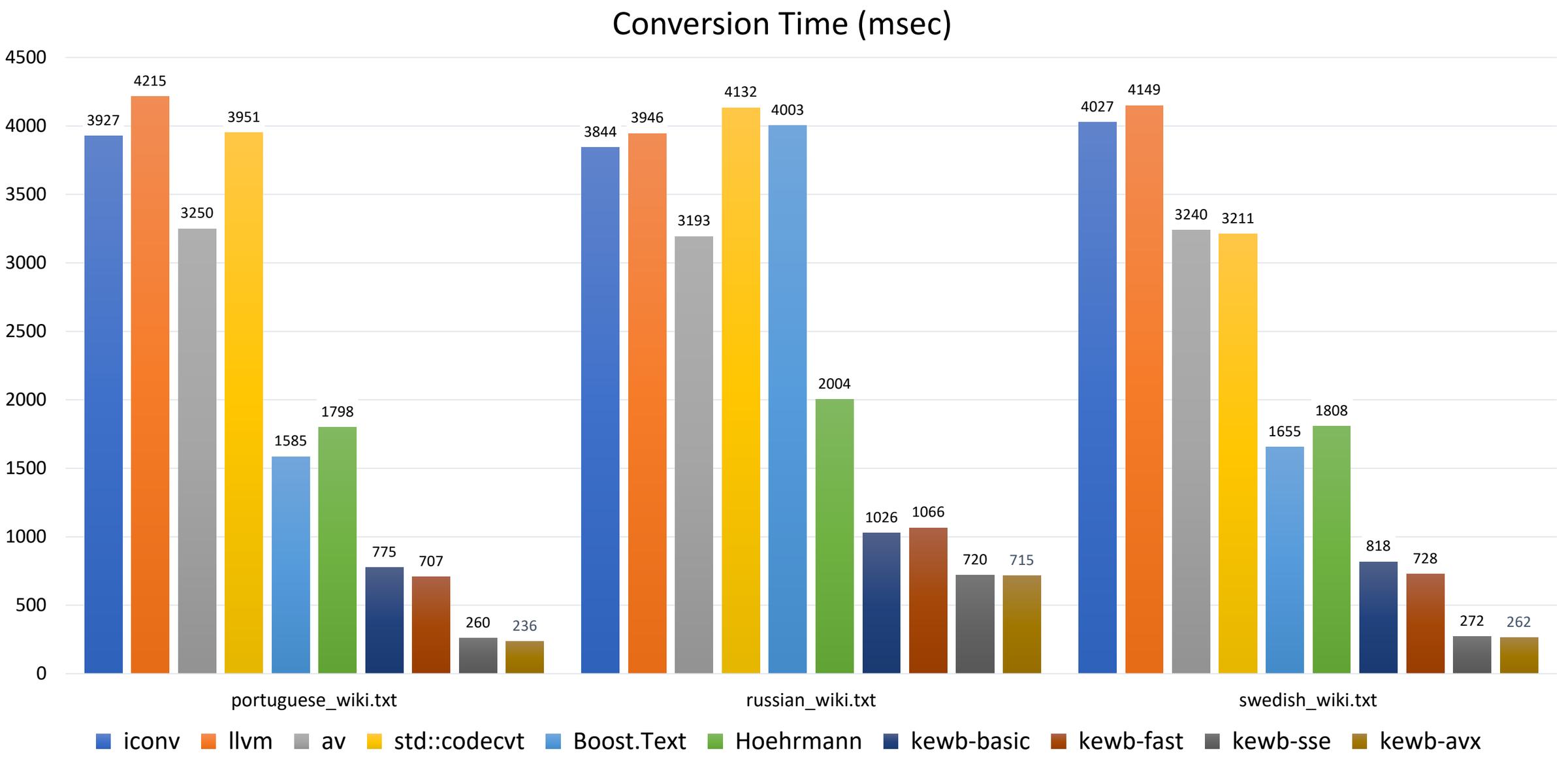
Clang 10.0.1 – Ubuntu 18.04 – Core i9 – UTF-32



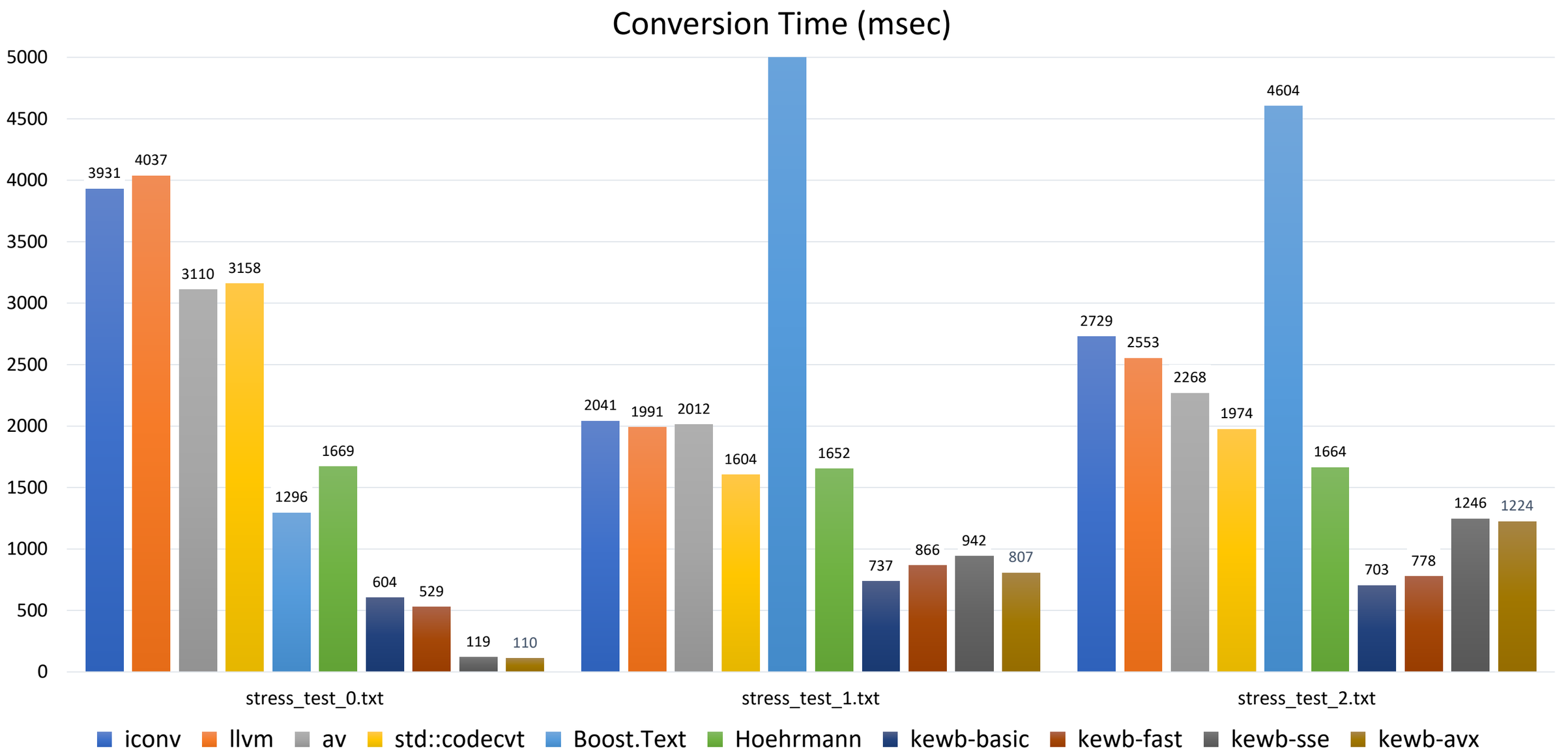
VS2019 Win10 – Core i9 – UTF-32



VS2019 Win10 – Core i9 – UTF-32



VS2019 Win10 – Core i9 – UTF-32



Thank You for Attending!

Talk: `github.com/BobSteagall/CppCon2020`

Blog: `bobsteagall.com`