Document Number:	P3922R1
Date:	2025-11-07
Reply-to:	Matthias Kretz <m.kretz@gsi.de>
Audience:	LEWG, LWG
Target:	C++26

Missing deduction guide from `simd::mask` to `simd::vec`

ABSTRACT

After a fix to the return type of the unary operators of `basic_mask`, the natural corresponding integral `basic_vec` cannot be determined from the specification anymore. The implementation has some freedom to choose an integer type and an ABI tag, that the user can only determine via `decltype` on the result of a unary operator. To match the intended analogous usage in scalar code, it would be helpful if CTAD from `basic_mask` to `basic_vec` just works.

CONTENTS

1 CHANGELOG

(placeholder)

2 STRAW POLLS

(placeholder)

3 Motivation

In scalar code we sometimes convert booleans into integers (typically to implement branch-less algorithms). E.g.

```
template <typename T>
void f(T x, T y) {
  int b = x < y;
  // ...
}
```

The equivalent is possible with std::simd:

```
template <simd_vec_type V>
void f(V x, V y) {
  vec<int> b = x < y;
  // ...
}
```

Except that the type of `b` is probably wrong and so the code is ill-formed. But what is the correct type?

```
template <simd_vec_type V>
void f(V x, V y) {
  auto b = x < y;
  // ...
}
```

This doesn't do the conversion to integral `basic_vec`, but rather keeps `b` of type `basic_mask`. What we can do is:

```
template <simd_vec_type V>
void f(V x, V y) {
  auto b = +(x < y);
  // ...
}
```

The unary plus operator will convert the mask to an integral `basic_vec`, analogous to unary plus on `bool`. But it would be less cryptic if the user could write instead:

```
template <simd_vec_type V>
void f(V x, V y) {
  basic_vec b = x < y;
  // ...
}
```

However, this CTAD requires a deduction guide, which has long been part of my implementation but somehow didn't make it into the wording.

4 Proposed Poll

Poll: Add the deduction guide from `basic_mask` to `basic_vec` from P3922R1 to C++26, resolving DE-287

SF	F	N	A	SA

5 WORDING

5.1 FEATURE TEST MACRO

In [version.syn] bump the `__cpp_lib_simd` version.

5.2 MODIFY [SIMD.SYN] AND [SIMD.EXPOS]

In [simd.overview], insert:

[simd.syn]

```
template<class R, class... Ts>
  basic_vec(R&& r, Ts...) -> see below;

template<size_t Bytes, class Abi>
  basic_vec(basic_mask<Bytes, Abi>) -> see below;
```

In [simd.ctor], insert:

[simd.ctor]

```
template<class R, class... Ts>
  basic_vec(R&& r, Ts...) -> see below;
```

1 *Constraints*:

- `R` models `ranges::contiguous_range` and `ranges::sized_range`, and
- `ranges::size(r)` is a constant expression.

2 *Remarks:* The deduced type is equivalent to `vec<ranges::range_value_t<R>, ranges::size(r)>`.

```
template<size_t Bytes, class Abi>
  basic_vec(basic_mask<Bytes, Abi> k) -> see below;
```

3 *Constraints*: `basic_mask<Bytes, Abi>` is an enabled specialization of `basic_mask` and `decltype(+k)` is a valid type.

4 *Remarks:* The deduced type is equivalent to `decltype(+k)`.
