P3104

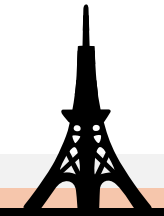
Bit permutations

Tokyo 2024

東京

Author: Jan Schultke
Presenter: Jan Schultke
Audience: SG18
Project: ISO/IEC 14882 Programming Languages — C++, ISO/IEC JTC1/SC22/WG21

Contents



1. Introduction

History

- `<bit>` functions added by **P0553R4:** Bit operations (C++20)
  - Simple utilities (`has_single_bit`)
  - Instruction wrappers (`rotl`, `popcount`, `countl_zero`, ...)
- `<stdbit.h>` functions added by **N3022:** Modern Bit Utilities (C23)

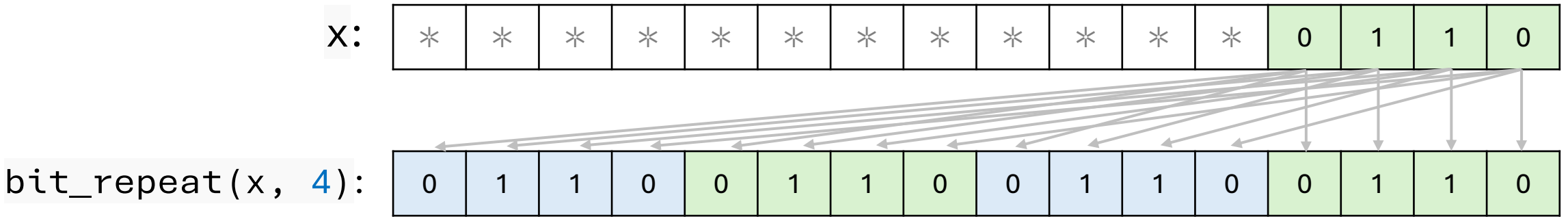
Goals

1. More utilities.
   1. `bit_repeat`, `next_bit_permutation`, `prev_bit_permutation`
2. More instruction wrappers.
   1. `bit_reverse`, `bit_compress`, `bit_expand`

2. Proposal

```
template<unsigned-integral T>
constexpr T bit_repeat(T x, int length);
```

Preconditions: `length` ≥ 0.
Returns: Rightmost length bits in `x`, repeated.
Motivation: Generate recurring bit patterns.
Hardware support: Diverse; depends on `length`.
Example:



2. Proposal

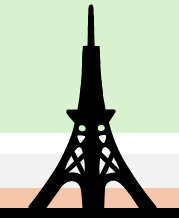
Motivating example for `bit_repeat`

Implementation of `countr_zero(v)` taken from *Bit Twiddling Hacks*:

```
unsigned int v;      // 32-bit word input to count zero bits on right
unsigned int c = 32; // c will be the number of zero bits on the right
v &= -v;
if (v) c--;

if (v & 0x0000FFFF) c -= 16;
if (v & 0x00FF00FF) c -=  8;
if (v & 0x0F0F0F0F) c -=  4;
if (v & 0x33333333) c -=  2;
if (v & 0x55555555) c -=  1;
```

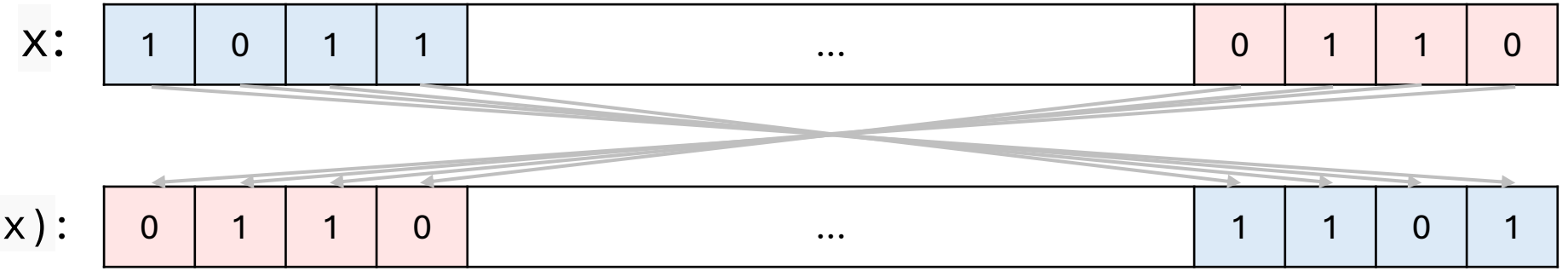
```
for (int i = 16; i != 0; i /= 2) {
    unsigned int mask = bit_repeat((1u << i) - 1, i * 2);
    if (v & mask) c -= i;
}
```



2. Proposal

```
template<unsigned-integral T>
constexpr T bit_reverse(T x) noexcept;
```

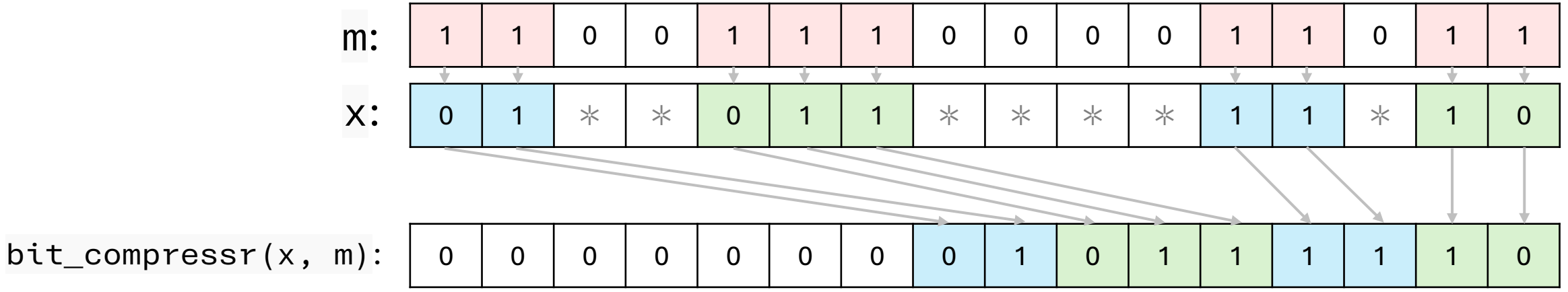
Returns: `x` with the order of bits reversed.
Motivation: PRNGs, FFT, CRC, image processing, ...
Hardware support: `rbit`(ARM) , `bswap`(x86_64), ...
Example:



2. Proposal

```
template<unsigned-integral T>          // analogous for bit_compressl
constexpr T bit_compressr(T x, T m) noexcept;
```

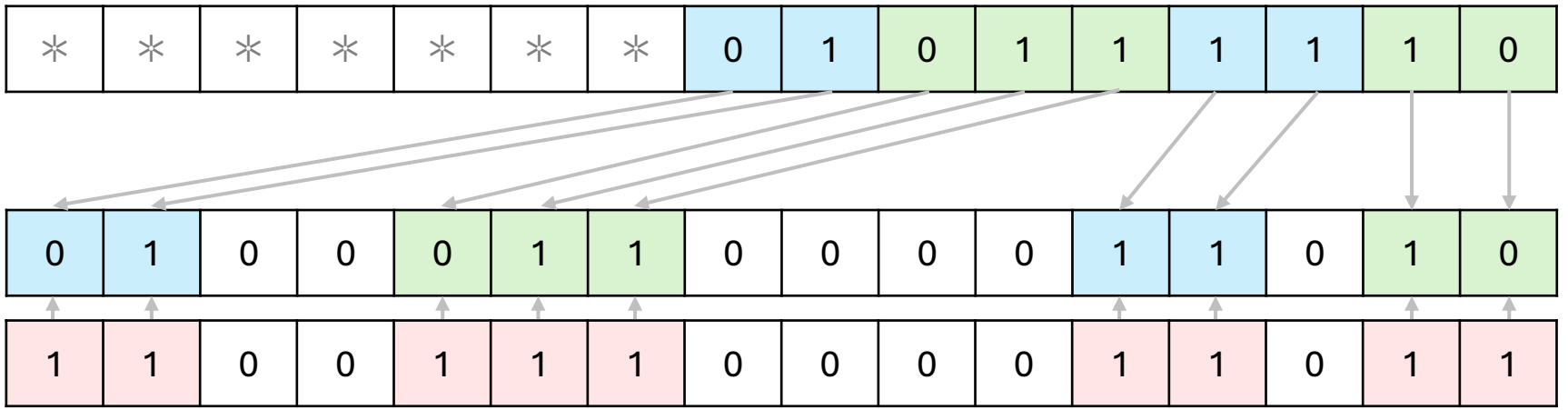
Returns: `x` filtered using “mask“ `m` , tightly packed to the right.
Motivation: Space-filling curves, UTF-8, chess engines, genomics, ...
Hardware support: `bext`(ARM) , `pext`(x86_64).
Example:



2. Proposal

```
template<unsigned-integral T>      // analogous for bit_expandl
constexpr T bit_expandr(T x, T m) noexcept;
```

Returns: `x`'s right bits, unpacked into where "mask" `m` has one-bits.
Motivation: (see `bit_compress`)
Hardware support: `bdep`[(ARM)], `pdep`[(x86_64)].
Example:

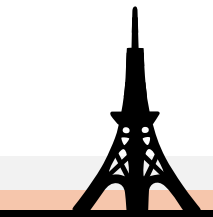


2. Proposal

```
template<unsigned-integral T> // inverse of prev_bit_permutation (kinda)
constexpr T next_bit_permutation(T x) noexcept;
```

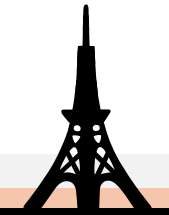
Returns: The lowest integer > `x` with the same amount of one-bits, or zero.
Motivation: Iterating over fixed-length subsets.
Hardware support: `ctz`(ARM), `tzcnt`(x86_64), ... (indirect support).
Examples:

- `next_bit_permutation( 0b111u)` ⟶ `0b1011u // 11`
- `next_bit_permutation(0b1011u)` ⟶ `0b1101u // 13`
- `next_bit_permutation(0b1101u)` ⟶ `0b1110u // 14`
- `next_bit_permutation(0b1110u)` ⟶ `0b10011u // 19`
- ...



4. Implementation experience

- GitHub: `Eisenwave/cxx26-bit-permutations` implements all functions.
  - Hardware support utilization
    - x86_64
    - ARM
  - GCC, clang, MSVC
  - Support for arbitrary N-bit integers (`_BitInt`)



References

Jens Maurer; **P0553R4** Bit operations
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p0553r4.html

Daniil Goncharov; **N3022** Modern Bit Utilities
https://thephd.dev/_vendor/future_cxx/papers/C%20-%20Modern%20Bit%20Utilities.html

Jan Schultke; **P3104** Bit permutations (latest revision)
https://eisenwave.github.io/cpp-proposals/bit-permutations.html

Jan Schultke; C++26 Bit permutations (reference implementation)
https://github.com/Eisenwave/cxx26-bit-permutations

Sean Eron Anderson; Bit Twiddling Hacks
https://graphics.stanford.edu/~seander/bithacks.html