CS128 Honors

Types

CS196-128 Rust 101

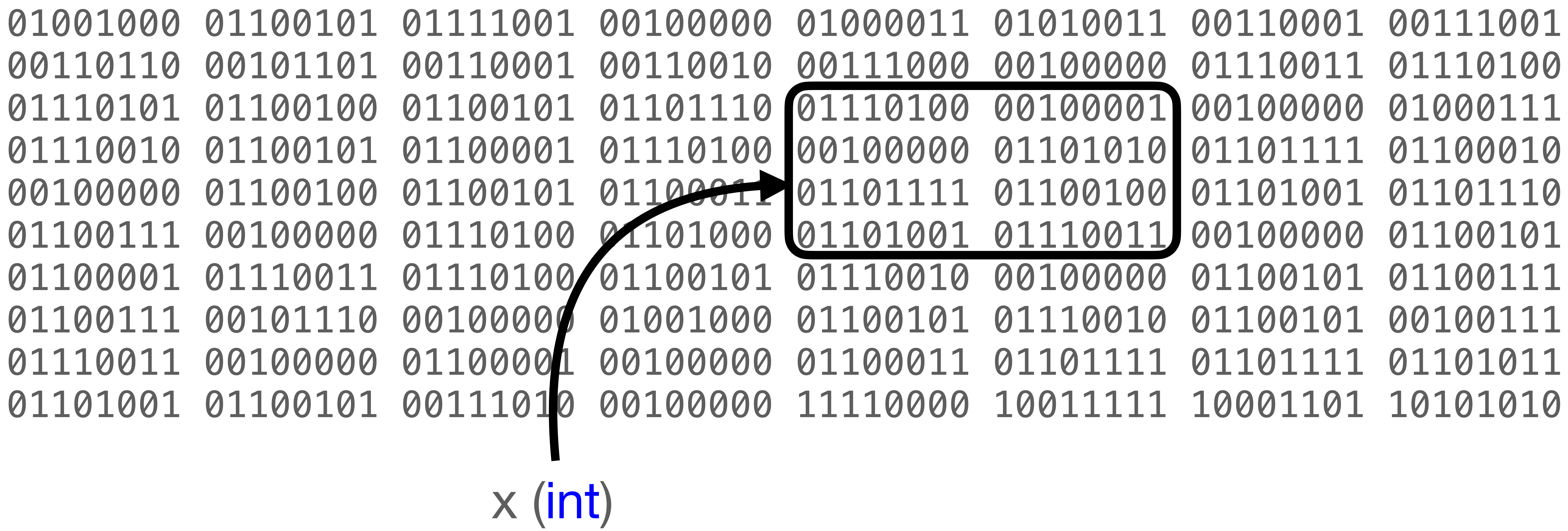
Slides by Matt Geimer (FA21)
Presented 9/1/2021

Types
Variables

- In CS 124 or AP Computer Science, you're told about types
- Examples include:
  - `int`
  - `String`
  - `char`
  - `double`

Types

Why are we talking about types again?



```
01001000 01100101 01111001 00100000 01000011 01010011 00110001 00111001
00110110 00101101 00110001 00110010 00111000 00100000 01110011 01110100
01110101 01100100 01100101 01101110 01110100 00100001 00100000 01000111
01110010 01100101 01100001 01110100 00100000 01101010 01101111 01100010
00100000 01100100 01100101 01100011 01101111 01100100 01101001 01101110
01100111 00100000 01110100 01101000 01101001 01110011 00100000 01100101
01100001 01110011 01110100 01100101 01110010 00100000 01100101 01100111
01100111 00101110 00100000 01001000 01100101 01110010 01100101 00100111
01110011 00100000 01100001 00100000 01100011 01101111 01101111 01101011
01101001 01100101 00111010 00100000 11110000 10011111 10001101 10101010
```

Types

Variables

- In CS 124 or AP Computer Science, you're told about types
- Examples include:
  - `int`
  - `String`
  - `char`
  - `double`
- What's nice about Rust is that the compiler can infer what type a variable is

Scalar Types

Types in Rust

- **Scalar Types** represent a singular value
- Rust has 4 primary scalar types:
  - Integers
  - Floating-point numbers (Doubles or Floats)
  - Booleans
  - Characters

Integers

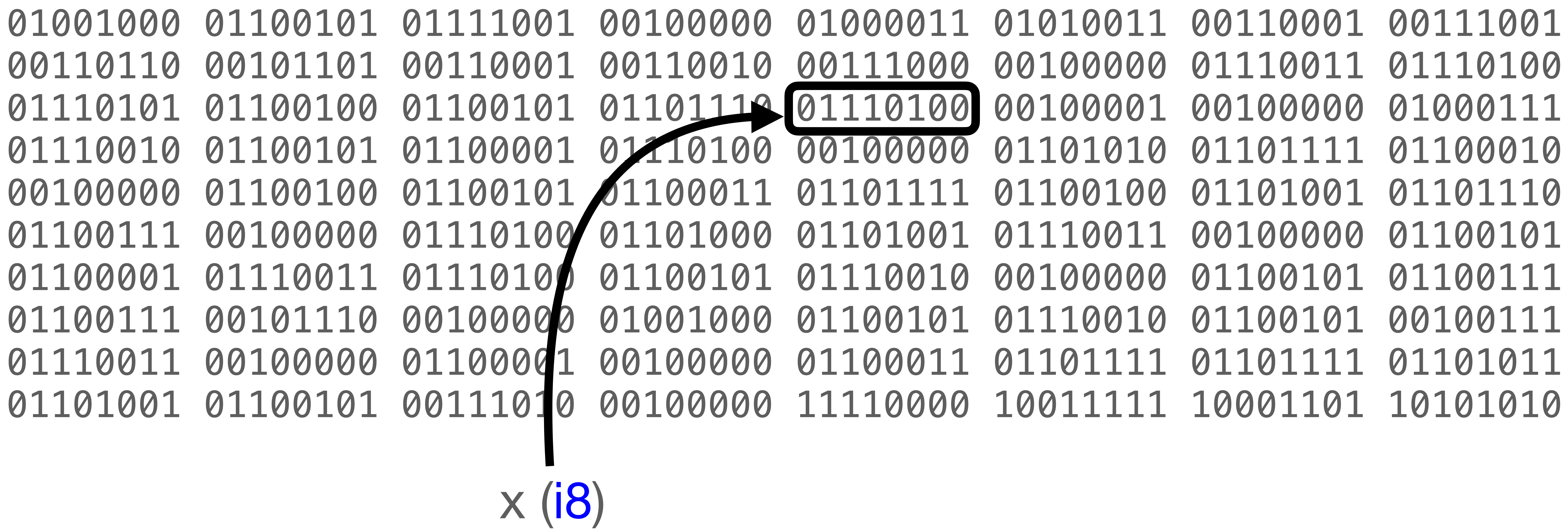
Scalar Types in Rust

- Refresher: Integers are numbers that have no decimal places
- There are many types of integers in Rust
  - They can be different sizes
  - They can be either signed or unsigned

Length	Signed	Unsigned
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`

Integers

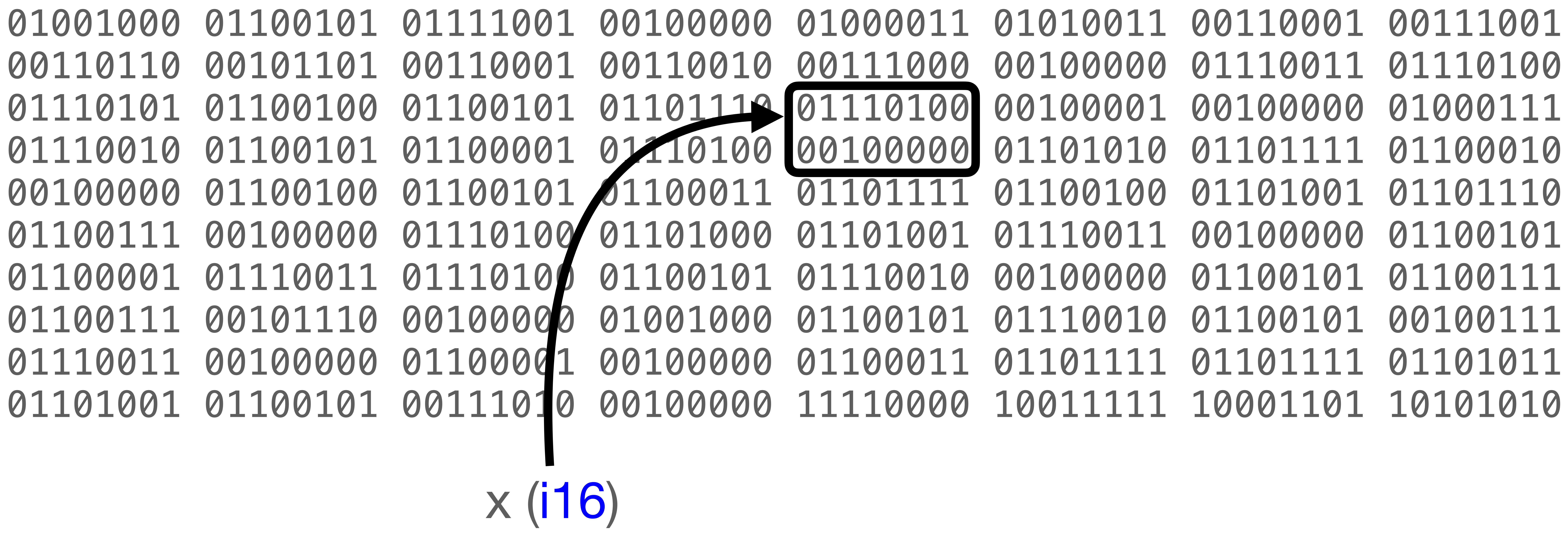
What does length mean



```
01001000 01100101 01111001 00100000 01000011 01010011 00110001 00111001
00110110 00101101 00110001 00110010 00111000 00100000 01110011 01110100
01110101 01100100 01100101 01101110 01110100 00100001 00100000 01000111
01110010 01100101 01100001 01110100 00100000 01101010 01101111 01100010
00100000 01100100 01100101 01100011 01101111 01100100 01101001 01101110
01100111 00100000 01110100 01101000 01101001 01110011 00100000 01100101
01100001 01110011 01110100 01100101 01110010 00100000 01100101 01100111
01100111 00101110 00100000 01001000 01100101 01110010 01100101 00100111
01110011 00100000 01100001 00100000 01100011 01101111 01101111 01101011
01101001 01100101 00111010 00100000 11110000 10011111 10001101 10101010
```

Integers

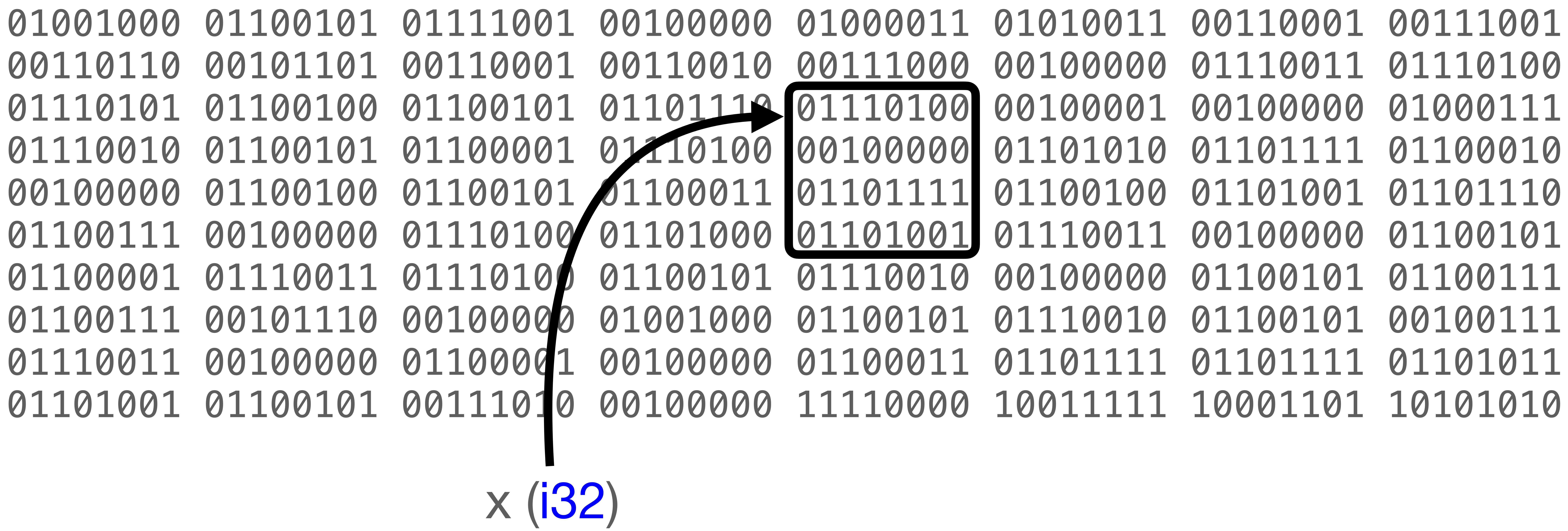
What does length mean



```
01001000 01100101 01111001 00100000 01000011 01010011 00110001 00111001
00110110 00101101 00110001 00110010 00111000 00100000 01110011 01110100
01110101 01100100 01100101 01101110 01110100 00100001 00100000 01000111
01110010 01100101 01100001 01110100 00100000 01101010 01101111 01100010
00100000 01100100 01100101 01100011 01101111 01100100 01101001 01101110
01100111 00100000 01110100 01101000 01101001 01110011 00100000 01100101
01100001 01110011 01110100 01100101 01110010 00100000 01100101 01100111
01100111 00101110 00100000 01001000 01100101 01110010 01100101 00100111
01110011 00100000 01100001 00100000 01100011 01101111 01101111 01101011
01101001 01100101 00111010 00100000 11110000 10011111 10001101 10101010
```

Integers

What does length mean



```
01001000 01100101 01111001 00100000 01000011 01010011 00110001 00111001
00110110 00101101 00110001 00110010 00111000 00100000 01110011 01110100
01110101 01100100 01100101 01101110 01110100 00100001 00100000 01000111
01110010 01100101 01100001 01110100 00100000 01101010 01101111 01100010
00100000 01100100 01100101 01100011 01101111 01100100 01101001 01101110
01100111 00100000 01110100 01101000 01101001 01110011 00100000 01100101
01100001 01110011 01110100 01100101 01110010 00100000 01100101 01100111
01100111 00101110 00100000 01001000 01100101 01110010 01100101 00100111
01110011 00100000 01100001 00100000 01100011 01101111 01101111 01101011
01101001 01100101 00111010 00100000 11110000 10011111 10001101 10101010
```

Integers

Scalar Types in Rust

- Refresher: Integers are numbers that have no decimal places
- There are many types of integers in Rust
  - They can be different sizes
  - They can be either signed or unsigned

Length	Signed	Unsigned
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`

Integers

Signed versus Unsigned

`i8`

Positive/Negative Bit

Magnitude

$\boxed{1}\boxed{0010100}$

-108

`u8`

Magnitude

$\boxed{10010100}$

148

Integers

Scalar Types in Rust

- Refresher: Integers are numbers that have no decimal places
- There are many types of integers in Rust
  - They can be different sizes
  - They can be either signed or unsigned

Length	Signed	Unsigned
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`

Integers

A note on arch

- arch types depend on the architecture type of your machine
- arch types:
  - isize
  - usize
- Size for machine type:
  - 32-bit architecture → 32 bits
  - 64-bit architecture → 64 bits

Integers

Examples

```
let eight_bit_int: u8 = 128;                  // An unsigned 8-bit integer
let signed_sixteen_bits: i16 = 196;         // A signed 16-bit integer
let architecture_size: isize = 42178094271; // A signed 64-bit integer
```

Floats

Scalar Types in Rust

- There are only two types of floats in rust
    - f32
    - f64
- The default is f64 since on most CPUs it's the same speed but more precise
- Similarly to integers, the number is based on the number of bits the variable stores

Booleans

Scalar Types in Rust

- Just like in other languages, booleans are used to store values that can either be true or false
- Booleans require 1 byte in size (despite only needing one bit)

```rust
let implicitly_typed = true;
let explicitly_typed: bool = false;
```

Characters

Scalar Types in Rust

- Characters are used to store letters and are the underlying components of strings
- Characters in Rust are 4 bytes large (only 1 byte in C++)
- Characters are defined using single quotes (double quotes are strings)
- Rust uses Unicode Scalar Values meaning...
  - Rust ❤️ emojis!

```
let rustacean_emoji = '🦀';
```

Strings

Not a Scalar Type

- Strings are used to represent groups of characters
- Strings are defined using double quotes (single quotes are characters)
- Strings are **not** a scalar type, but since you'll need them, here's an example of how to declare one:

```
let my_string = "Hello, World!";
```

Summary

Basic Types in Rust

- Scalar types:
  - Integers
  - Floats
  - Booleans
  - Characters
- Non-scalar type:
  - Strings

CS128 Honors

Types

CS196-128 Rust 101

Slides by Matt Geimer (FA21)
Presented 9/1/2021