

**Slices** | Slice Patterns

# ■ Use Case

- ◆ Read the first few bytes to determine header information
  - Take different actions based on the data using *match*
- ◆ Get the first or last elements of a slice
- ◆ No need for bounds checking on slices
  - Compiler ensures access are always within bounds

# Example

```
let chars = vec!['A', 'B', 'C', 'D'];
match chars.as_slice() {
    [first, .., last] => (),
    [single] => (),
    [] => (),
}
```

```
let chars = vec!['A', 'B', 'C', 'D'];
match chars.as_slice() {
    [one, two, ..] => (),
    [.., last] => (),
    [] => (),
}
```

# Overlapping Patterns

- ◆ Patterns easily overlap.
- ◆ Minimize number of match arms to avoid bugs

```
match slice {  
  [first, ..] => (),  
  [.., last] => (),  
  [] => (),  
}
```

**Second arm always ignored**

# Prevent Overlapping Patterns

- Match the largest patterns first, followed by smaller patterns

```
match slice {  
  [] => (),  
  [a, ..] => (),  
  [a, b, ..] => (),  
  [a, b, c, ..] => (),  
  [a, b, c, d, ..] => (),  
}
```

First two arms cover all cases,  
remaining will be ignored

```
match slice {  
  [a, b, c, d, ..] => (),  
  [a, b, c, ..] => (),  
  [a, b, ..] => (),  
  [a, ..] => (),  
  [] => (),  
}
```

All arms can be matched

# Guards

```
let nums = vec![7, 8, 9];
match nums.as_slice() {
    [first @ 1..=3, rest @ ..] => {
        // 'first' is always 1, 2 or 3
        // 'rest' is the remaining slice
    },
    [single] if single == &5 || single == &6 => (),
    [a, b] => (),
    [..] => (),
    [] => (),
}
```

# Recap

- ◆ Slices can be matched on specific patterns
  - These patterns can include *match guards*
- ◆ Match on largest patterns first, followed by smaller patterns
  - Smaller patterns tend to be *greedy*
- ◆ Minimize the number of match arms to avoid bugs