

Crate | rayon

■ **rayon**

- ◆ *rayon* provides parallel iteration
- ◆ No boilerplate needed to enable parallel processing of data
- ◆ Automatically uses all processing cores available on the machine

■ Prelude

```
use rayon::prelude::*;
```

Example - Sequential

```
let ids = vec![
    " 1234", "5678 ", " 1155 ", "555.5",
    "-9999", "5", "twelve", "1001", "9999"
];
let ids = ids
    .iter()
    .map(|id| id.trim())
    .filter_map(|id| id.parse().ok())
    .filter(|num| num >= &1000)
    .collect::
```

Example - Parallel

```
let ids = vec![  
    " 1234", "5678 ", " 1155 ", "555.5",  
    "-9999", "5", "twelve", "1001", "9999"  
];  
let ids = ids  
    .par_iter()  
    .map(|id| id.trim())  
    .filter_map(|id| id.parse().ok())  
    .filter(|num| num >= &1000)  
    .collect::
```

■ Example - Sorting

```
let mut ids: Vec<usize> = ids;  
ids.par_sort();  
for id in ids {  
    println!("{}", id);  
}
```

1001

1155

1234

5678

9999

■ Example - *for..in* (Error)

```
let ids = vec![
    " 1234", "5678 ", " 1155 ", "555.5",
    "-9999", "5", "twelve", "1001", "9999"
];

for id in ids.par_iter() {
    println!("{}", id);
}
```


Example - *for_each* Correct

```
let ids = vec![
    " 1234", "5678 ", " 1155 ", "555.5",
    "-9999", "5", "twelve", "1001", "9999"
];

ids.par_iter()
    .for_each(|id| println!("{}", id));
```

```
1234
-9999
5
1155
1001
twelve
555.5
9999
5678
```

Recap

- ◆ *rayon* provides simple to use parallel execution
- ◆ Parallel iterators have much of the same functionality as standard library sequential iterators
- ◆ Use *par_iter()* to create a parallel iterator
- ◆ Cannot use *for .. in* with parallel iterators
 - Use *.for_each()* if you want to execute code on each item