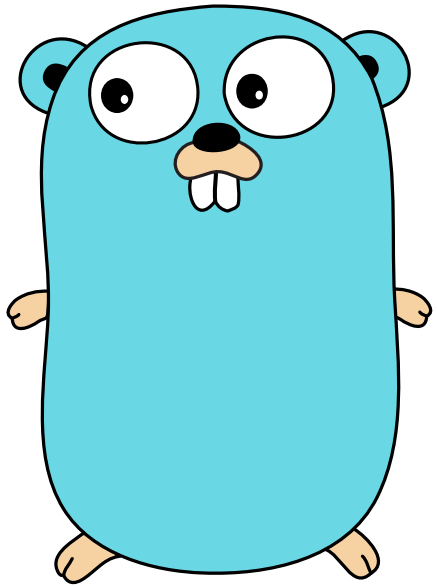
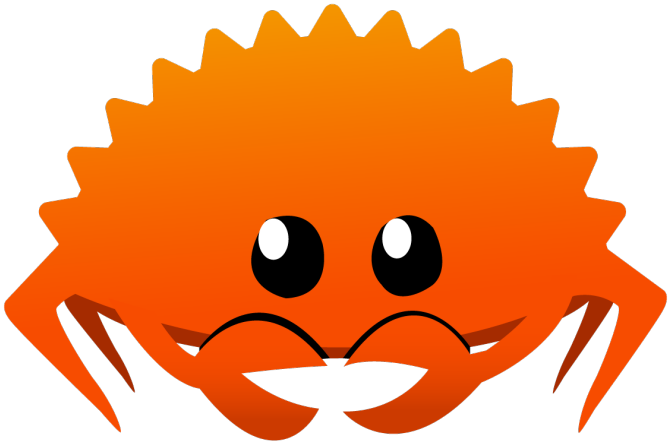
Parallel Programming – Go vs Rust



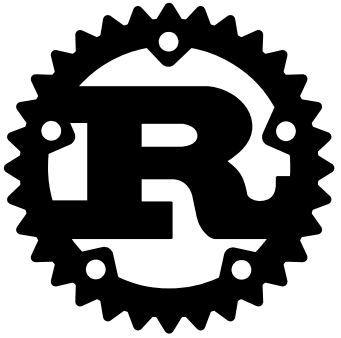


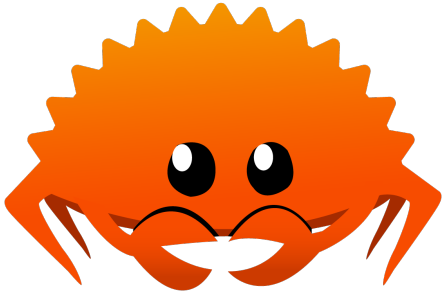
Overview of Rust

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Since 2015
- Compiled
- Static typed
- High performance

- Design goals:
  - Safety
  - Control over memory
  - Highly concurrent





Concepts in common

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Channels
- Mutexes
- Read write locks
- Atomic types
- Condition variables
- Once

Things **not** in common

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Go

- WaitGroups
- select
- Pools of temporary objects
- Thread-safe map

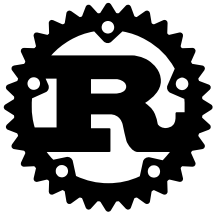
Rust

- Join
- Barrier
- Arc

  atomically reference counting pointers

Comparison of syntax

Technische Hochschule Rosenheim
Technical University of Applied Sciences



```go
package main

import "fmt"

func sum(numbers []int, channel chan int) {
    sum := 0
    for _, value := range numbers {
        sum += value
    }
    channel <- sum
}

func main() {
    numbers := []int{7, 2, 8, -9, 4, 0}
    channel := make(chan int)

    go sum(numbers[:len(numbers)/2], channel)
    go sum(numbers[len(numbers)/2:], channel)

    x := <-channel
    y := <-channel

    fmt.Println(x, y, x+y)
}

```

```rust
use std::sync::mpsc;
use std::thread;

fn sum(numbers: &[i32], tx: &mpsc::Sender<i32>) {
    let mut sum = 0;
    for value in numbers {
        sum += value;
    }
    tx.send(sum).unwrap();
}

fn main() {
    let numbers = [7, 2, 8, -9, 4, 0];
    let (tx, rx): (mpsc::Sender<i32>, mpsc::Receiver<i32>) = mpsc::channel();
    let tx1 = mpsc::Sender::clone(&tx);

    thread::spawn(move || sum(&numbers[..numbers.len() / 2], &tx1));
    thread::spawn(move || sum(&numbers[numbers.len() / 2..], &tx));

    let x = rx.recv().unwrap();
    let y = rx.recv().unwrap();

    println!("{} {} {}", x, y, x + y);
}

```

Comp

chnische
chschule
senheim
University of Applied Sciences

```go
package main

import "fmt"

func sum(numbers []int, cha
    sum := 0
    for _, value := range n
        sum += value
    }
    channel <- sum
}

func main() {
    numbers := []int{7, 2, 
    channel := make(chan in

    go sum(numbers[:len(numb
    go sum(numbers[len(numbe

    x := <-channel
    y := <-channel

    fmt.Println(x, y, x+y)
}
```

```rust
{

iver<i32>) = mpsc::channel();

.len() / 2], &tx1));
en() / 2..], &tx));
```

Threading model

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Goroutines

Several program threads
get multiplexed
on several system threads

by the goruntime

Rust Threads

One program thread
is
one system thread

Conclusion

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Most of the concepts are realised in both languages
- Biggest difference is the threading model
- Go is more easy to read and write
- Rust enforces safety

- M
  la
- Bi
- G
- R


NOT SURE IF I SHOULD GO WITH SIMPLE CODE
OR EXTRA SECURITY
memegenerator.net

Sources

- https://www.rust-lang.org/logos/rust-logo-blk.svg
- http://rustacean.net/assets/rustacean-orig-noshadow.svg
- https://golang.org/s/logos
- https://github.com/golang-samples/gopher-vector/raw/master/gopher.svg
- https://ih0.redbubble.net/image.15463304.6901/sticker,375x360.u4.png
- https://memegenerator.net/img/instances/82573730/not-sure-if-i-should-go-with-simple-code-or-extra-security.jpg
- https://memegenerator.net/img/instances/82573761/one-does-not-simply-ignore-the-ownership-system.jpg