UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Module 6: Concurrency
Advanced Programming

Prepared by: Ade Azurat

Editor: Teaching Team (Lecturer and Teaching Assistant)

Advanced Programming

Semester Genap 2023/2024

Fasilkom UI

Author: *Ade Azurat*
Any question, feedback and critics can be sent by email to: ade@cs.ui.ac.id

Table of Contents

Learning Objectives

1. Students should know that there is other approach to improve performance other than improving its data structure and algorithms which is by applying concurrency on more machine or (core) processor.
2. Students should understand how performance can be achieved by applying concurrency.
3. Students should understand about several models of concurrency.
4. Students should know how to program concurrency on a modern and suitable language.

References

1. Chapter 4,16,20, *Steve Klabnik and Carol Nichols,* The Rust Programming Language

1. Motivation

Motivation

- Remember this code:

```
private static String generateLargeString(int size) {
    String result = "";
    for (int i = 0; i < size; i++) {
        result += "a";
    }
    return result;
}
```

FACULTY OF COMPUTER SCIENCE

Remember the previous topic. We have learned about profiling, and we managed to identify this function as the longest/slower function. We see the inappropriate use of type String. We have fixed it below. We use StringBuilder.

Motivation

- We manage to improve it performance significantly by refactoring on it's data structure

```
private static String generateLargeString(int size) {
    StringBuilder builder = new StringBuilder(size);
    for (int i = 0; i < size; i++) {
        builder.append("a");
    }
    return builder.toString();
}
```

FACULTY OF COMPUTER SCIENCE

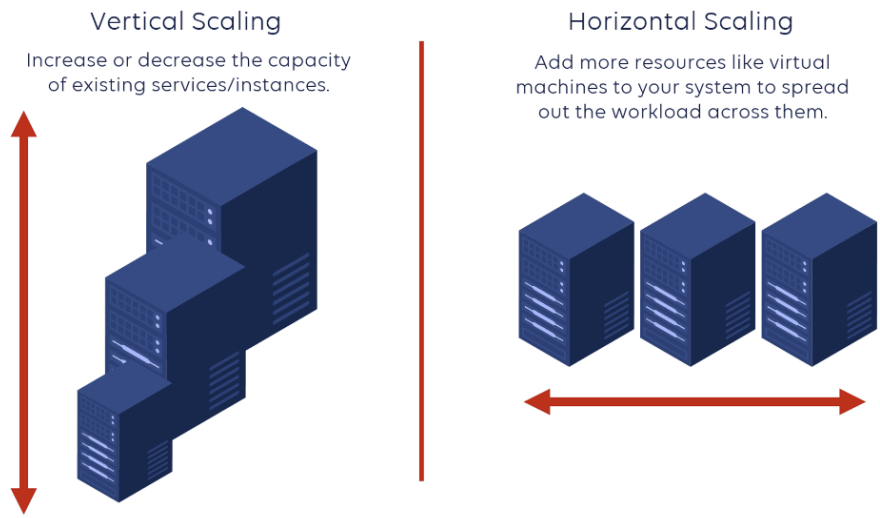
Question?

- Is there any other approach to improve the performance further? Let just say we have chosen the best data structure and algorithm.
- Imagine that you have your application running in production. The users are getting bigger. The transactions are getting more and more. We need higher performance. What can we do?

FACULTY OF COMPUTER SCIENCE

We will learn another approach to improve the performance. The previous approach we'll rely on the algorithm and data structure to improve the performance. The current approach is by make use of the computing power that we have. The computing power can be improved by the following: Vertical Scaling or Horizontal Scaling.

Scalability: Vertical vs Horizontal



FACULTY OF COMPU SCIENC

By Vertical Scaling we use a better machine, replacing the previous one. By Horizontal Scaling, we use more machine. We add the previous machine with more machine. We can do both if necessary. Applying Vertical scaling is lot easier; we just need to allocate a better machine. But applying and take advantage of horizontal scaling is not that easy.

Motivation

- Let's do the vertical scaling first, we buy or rent the best machine.
- We will get a better performance from that machine.
- What if it is not enough? Vertical scalability has its limit.
- Or probably, it is just too expensive.

- Let's do the horizontal scaling, we buy or rent several machine.
- Do we have a better performance?

FACULTY OF COMPUTER SCIENCE

Applying vertical scaling means that our program will run on a better machine. So, it will automatically better performance as well. However, doing vertical scaling has its limit. The limit can be the hardware itself do to heat or production. It can also because of the expensive cost. Combining with horizontal scaling became an alternative. However, we cannot automatically get a better performance. Why? Because we must do something so that the other machine can also serv just as the previously single machine. Otherwise, only one machine will work, and the other machines will just idle. What is the something that we must do?

We must make sure that our program has some parallel aspect. Our program should be able to run in parallel on multiple machines. Regarding the terminology, we will use the term concurrency. Actually, by definition, parallel and concurrency are different. Parallel is defined as the ability to execute independent tasks of a program at the same time on multiple machines. While concurrency is defined as ability to execute several independent tasks of a program. Concurrency does not indicate that it should run at the same time nor in multiple machines. With this point of view, concurrency is a more generic concept. So, we will use the term concurrency which may indicate both concurrency and parallelism. A more specific concept of parallelism will be studied on the Parallel Programming Course

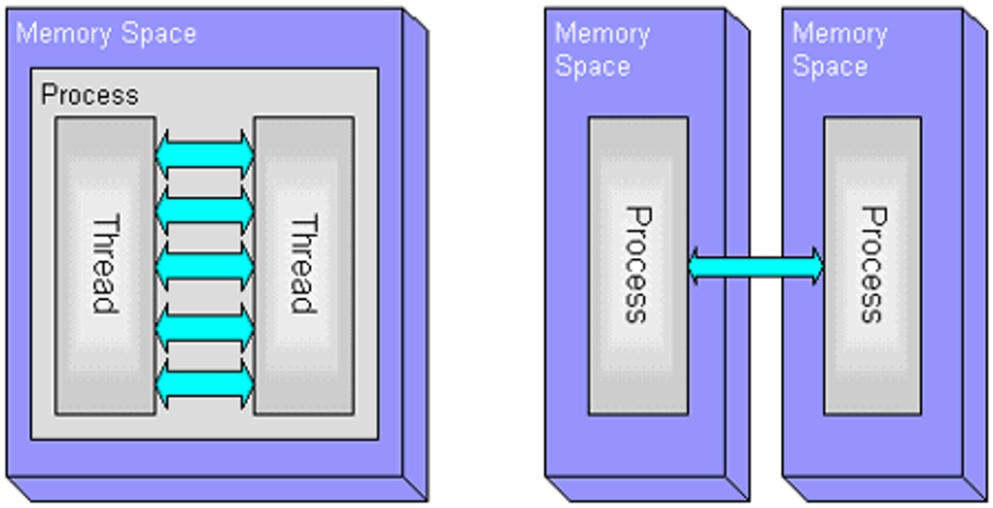
2. Thread and Multi-threading

Before continuing with concurrency, let's review some concepts that has been studied in the Operating System course. Thread and process.

Review Operating System:

- Process
  - A process is an instance of a running program that is *isolated* from other processes on the same machine. In particular, it has its own private section of the machine's memory.
- Thread
  - A thread is a locus of control inside a running program. Think of it as a place in the program that is being run, plus the stack of method calls that led to that place to which it will be necessary to return through.

FACULTY OF COMPUTER SCIENCE



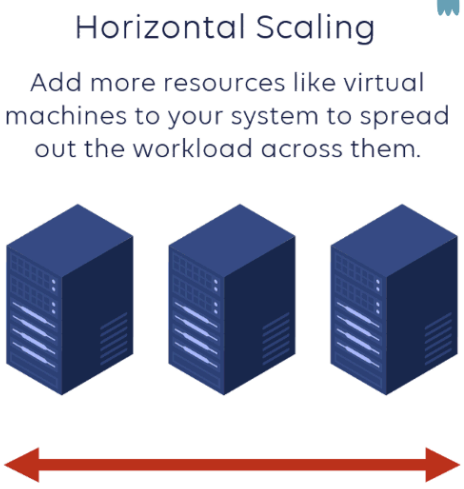
FACULTY OF COMPUTER SCIENCE

From the slides we can see the difference between *process* and *thread*. We can focus on that the thread usually share memory with other threads within one process. While processes usually have their own memory space.

With respect to this characteristic, the approach of concurrency for process and thread are also different.

Horizontal Scaling on process

- We will discuss it later in design pattern and also in software architecture. After mid term.

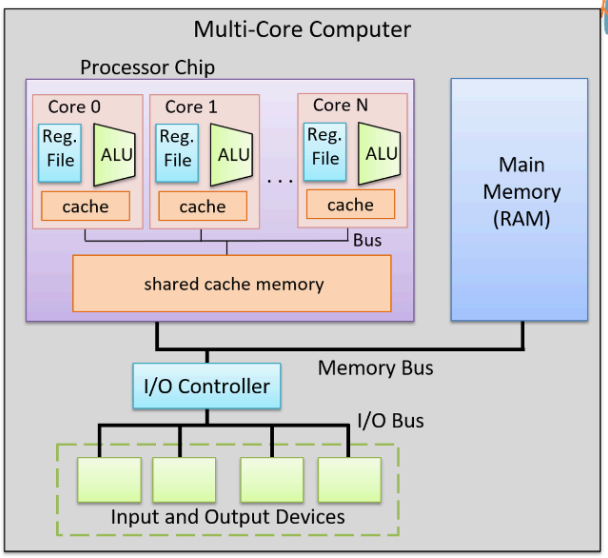


UNIVERSITAS INDONESIA
FACULTY OF COMPUTER SCIENCE

For process we will study how to make use of horizontal scaling. How to make sure that the performance is going to speed up by replicating the process on several machine. This can be done by applying some design pattern in the level programming, and some architecture in the level of design system. We will study about this later.

Horizontal Scaling on thread

- On a computer with multiple core
- We will discuss it this week!



FACULTY OF COMPUTER SCIENCE

This module will give you information about how to do horizontal scaling in the level of threads. The scaling factor usually in the lower-level structure of a machine or to be more precise usually in one machine and make use of several core it has. We will study how to make sure that our program we make use of the available core to speed-up the performance.

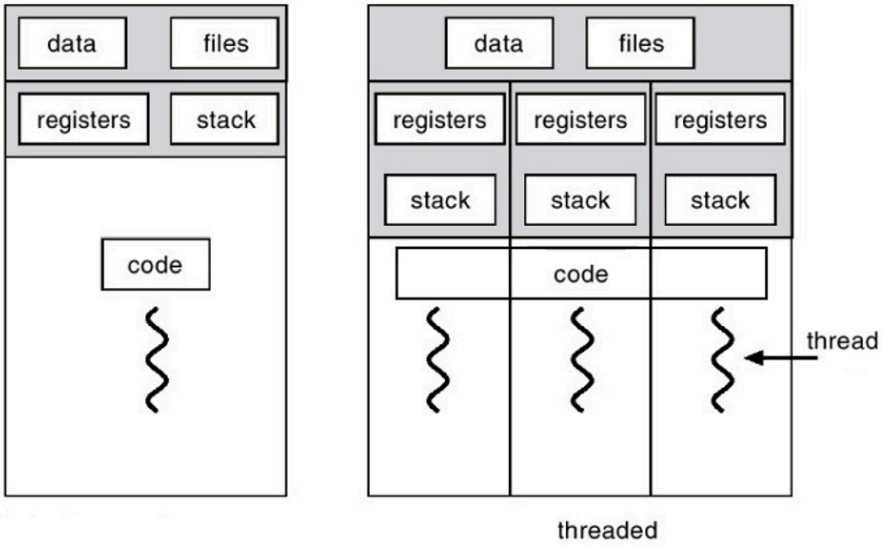
Going back to the previous question

If we run this program on a multi core computer, why it is not faster?

```
private static String generateLargeString(int size) {
    StringBuilder builder = new StringBuilder(size);
    for (int i = 0; i < size; i++) {
        builder.append("a");
    }
    return builder.toString();
}
```

FACULTY OF COMPUTER SCIENCE

The previous code will not be speed up if we run it on multi core computer. Because it will only be run on one single core. There is nothing that can be split-up in the program to be run on other cores.

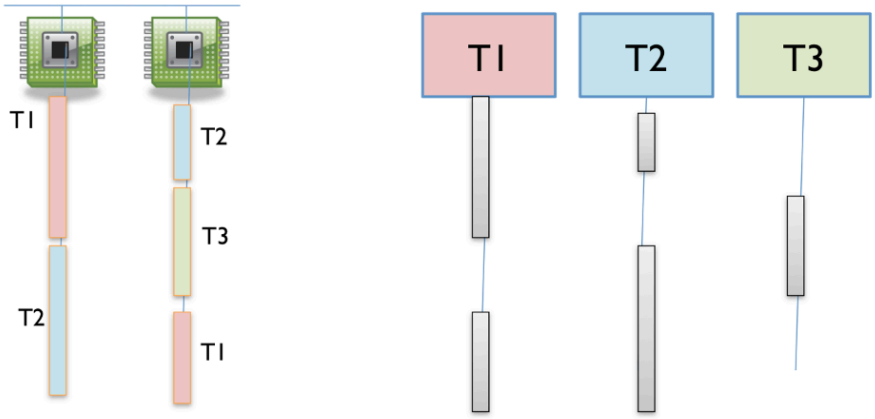


FACULTY OF COMPUTER SCIENCE

To make use of multi-core machine, the code need be split-up into several threads. Where each thread may have their own stack and register, but they share the memory (data and files) with other threads. The slide shows that the multi-threaded example are split-up into 3 threads. Those three threads are run concurrently. If the machine has more than three cores, we can guarantee of speed-up about three times than running it in a single core machine.

Oh, so a multi-threaded program can also be run in a single core machine? Yes, it can. A good programmer should better make use of multi-threaded program so it will make use the available core of the machines. Those multi-threaded programs can run regardless of how many cores a machine has.

How actually a multi-threaded program is run?



Source: https://web.mit.edu/6.005/www/fa14/classes/17-concurrency/#:~:text=Concurrency%20means%20multiple%20computations%20are,cores%20on%20a%20single%20chip)

A multi-threaded program is run in interleaving model. In the left side, we have two cores. We have three threads running. Those three threads will make use the two cores by applying interleaving. It means the scheduling system in the operating system control the amount of time a core can be used to execute a thread, when the time-up, it will switch to another threads. If we have two cores, than on any time, they should be at-least two threads are running simultaneously.

This is how a multi-threaded program is run. When we create a multi-threaded program, we don't need to worry how many cores a machine have. We just need to focus on how many threads we would like to have. More is better, because even if the number of cores is smaller, it will still run and make use of it. But of course, we do not want to create unlimited number of threads. Because it may cause other issues. Later we will see in the case of web server, if we allow to generate unlimited number of threads, we are vulnerable by the DoS (Denial of Service) Attack. The server will run out of memory or port allocation.

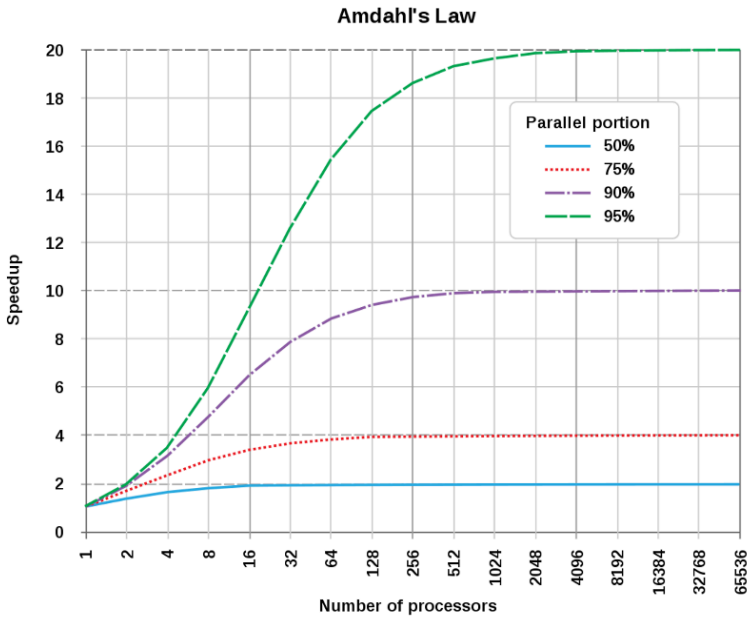
Speedup by multi-threading

We must keep in mind, that creating a multi-threaded program may speed-up the performance if have more cores in our machines. But we must remember, that just adding multiple cores, or multiple machines are not going to useful in improving performance if our program are not multi-threaded (concurrent) program.

There is an Amdahl's Law. It says: "the overall performance improvement gained by optimizing a single part of a system is limited by the fraction of time that the improved part is actually used"

Amdahl's Law

- We need to make a multi-threaded program.
- It is also applied to process.



FACULTY OF COMPUTER SCIENCE

Source: https://en.wikipedia.org/wiki/Amdahl%27s_law

This law is basically on parallelism, but it is also applied on multi-threading on several cores. It says that you can only have maximum speedup up to 2 times, if you only have two threads in your program (two parallel portion) no matter how many machines or cores do you have. It is described by the blue line in the graphic. If you have 95% parallel portion in your program, in other words you have 100/5 threads (or parallel portion), you can have maximum speedup 1/95% = 100/5 times.

Issues on Multi-threading

A multi-threading program is good. It can improve performance on multi-core machines. But it is not an easy task. As we have studied in Operating system, there are several issues in multi-threading programming.

Issues on multi-thread

- Critical section
- Race condition
- Lost update
- Synchronization
- Deadlock

This issues are simplified in Rust!

FACULTY OF COMPUTER SCIENCE

It usually starts by having shared memory. Having several threads in your program, it is possible that those thread may access the same shared memory. This shared memory became a critical section, because it would possible cause lost update when several threads simultaneously try to modify a same variable in the critical section. For examples:

We have one variable initialize with 1 and there are two threads adding the value with 1.

Scenario 1.

Thread 1:
Read var (var is 1)
Execute var = var + 1 (var is 2)

Thread 2:
Read var (var is 2)
Execute var = var + 1 (var is 3)

But in multi-threading program, those programs are run simultaneously. They may interleave each other; such as shown in scenario 2:

Scenario 2:

Thread 1:	Thread 2:
Read var (var is 1)	
	Read var (var is 1)
Execute var = var + 1 (var is 2)	
	Execute var = var + 1 (var is 2)

The result shows the var became 2, it means the update of thread 1 is lost. How can we solve this issue, we can make the critical section become mutual exclusion. It means we only allow one thread to access it at one time. This is some literature is called synchronization. It is a mechanism of locking a variable or a part of shared memory. However, if the variables and the threads are a lot, this can be tricky. Incorrect locking or uncareful ones may cause deadlock. In yield to another process to detect possible deadlock.

Readers are suggested to read again its operating system course about these issues if already forgot.

3. The Rust Programming Language

The Rust programming language is specifically designed to simplify these issues. We will use this language to build multi-threading programs. It is not going to eliminate the issues, but at least it will reduce, and the compiler will help in avoiding dan detecting possible issues. So, the programmer creativity can be focused on the logic of the programs.

This issues are simplified in Rust by explicitly think about:

- Resources
- Ownership
- Lifetime

Before we continue discussing these topics, we would like to raise one common comment that saying Rust is difficult!

Rust is difficult!

- Is it?
- this is hello world program in Rust:

```
fn main() {
    println("Hello World");
}
```

- This is hello world program in Java

```
public class Hello{
   public static void main(String[] argv){
       System.out.println("Hello world");
   }
}
```

FACULTY OF COMPUTER SCIENCE

If we look at the slide. We see two Hello World program. The above one is in Rust, and the second one is in Java. We have studied Java. We found that program is very familiar now. But if we try to remember, how we studied it a couple of semesters ago, we have found there are more unknown or unnecessary concepts are introduced in the beginning. There are keywords of public, class, static, System.out, not to mention the parameter String[] argv which need to be there although we do not need it. So the Java program is actually more complicated in this sense, but yet you are all survived :D

So it is more on whether we already familiar or not. It is the issue of education. The Rust programming language has more features than other programming languages. By teaching you Rust, you can be an expert in Rust, or if you are not going to be a Rust programmer, you will be able to study other language which less feature that Rust. You can even become a better programmer on other language if you already know how the Rust works such as how to secure the memory.

Immutable as the default

Like most Functional programming language, but not common in imperative language, the variable by default is ***immutable***. It will make it quite strange for people who already use common imperative language such as python and java. We also already saw in Java, that by default the type String is immutable, but we can always change it value which may cause to some issues of performance if we are not aware of. But is there a benefit of immutable as the default?

Wait, why a simple case won't work?

- Consider this simple program:

```
fn main() {
    let x = 5;
    println!("The value of x is: {x}");
    x = 6;
    println!("The value of x is: {x}");
}
```

If we compile, it complains!

FACULTY OF COMPUTER SCIENCE

This program seems simple, we would expect the output to be some thing like:

```
The value of x is: 5
The value of x is: 6
```

But this program in Rust is not compiled! It complains about immutable variable be modified!

Wait, why a simple case won't work?

- We should read the compile error messages! It says:

```
error[E0384]: cannot assign twice to immutable variable `x`
 --> src/main.rs:4:5
  |
2 |     let x = 5;
  |         -
  |         |
  |         first assignment to `x`
  |         help: consider making this binding mutable: `mut x`
3 |     println!("The value of x is: {x}");
4 |     x = 6;
  |     ^^^^^ cannot assign twice to immutable variable

For more information about this error, try `rustc --explain E0384`.
```

FACULTY OF COMPUTER SCIENCE

So, by reading the compiler message, we follow it suggestion of adding mut that indicate the variable is mutable (can be changed). We have the program run as we expect.

So, the solution is simple, just add mut

```
fn main() {
    let mut x = 5;
    println!("The value of x is: {x}");
    x = 6;
    println!("The value of x is: {x}");
}
```

```
The value of x is: 5
The value of x is: 6
```

FACULTY OF COMPUTER SCIENCE

But, why bother about (im)mutable?

- Other language such as python and java do not care about mutable and immutable?
- Well, in java type String is immutable, but we can always assign and reassign its values.
- Why should Rust make such different?
  Just to confuse the programmer?

- Well, these issues can be a huge issue when we have concurrency!
- We don't want any process/thread to modify any value, we need to control it!

FACULTY OF COMPUTER SCIENCE

Remember the issues in multi-threading (concurrency). So, Rust is more prepare for concurrency by make immutable as default so it would not be critical sections.

The benefits of the constraints

- The compiler puts some constraints.
- The constraints are given to reduce possibility of errors by the programmer.
- Let's go back to the history of programming.

FACULTY OF COMPUTER SCIENCE

Programming language is our language to communicate to the computer. It is how we instructs the computer to do what we want to achieve. Having a language that can communicate many things is considered to be good at first. But not in programming. We have seen assembly language which considered to be very powerful. We can do basically can create a program that can do almost anything a computer can do. But the assembly language is not going to be used for building web application. Why, is it cannot? It can be. But it lacks the abstraction, and it contains too much detail which is not needed in building web app.

Another issue, the assembly language can do so many things to your register and memory, if we are not careful, we may destroy the memory of our operating system. So Java came with JVM that has its own memory space. Java programmer will never be able to modify the memory used by operating system directly. It is just like limitation for Java programmer, not to be able to modify memory space of the operating system. But this limitation is accepted, but it will secure their program also will make the java programmer focus on its own business.

History also recorded some changes in programming language for the benefit of programmers.

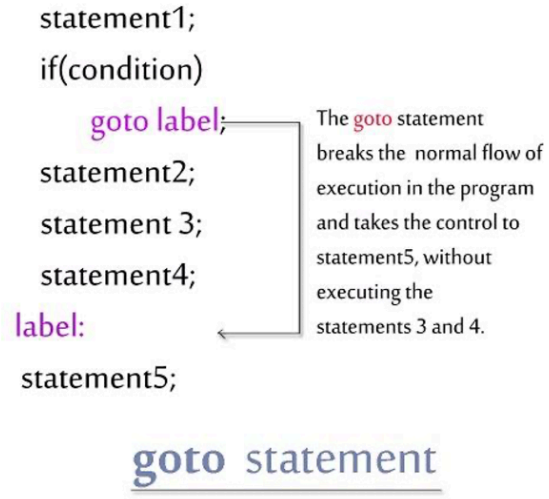
The (was) famous statement: GOTO

- Various languages emerged to foster structured programming, and some existing languages were modified to better support it.
- One of the most notable features of these structured-programming languages was not a feature at all:
  - It was the absence of something that had been around a long time—the GOTO statement.

FACULTY OF COMPUTER SCIENCE

The (was) famous statement: GOTO

- The GOTO statement is used to redirect program execution.
- Instead of carrying out the next statement in sequence,
- the flow of the program is redirected to some other statement,
- the one specified in the GOTO line,
- typically when some condition is met.



FACULTY OF COMPUTER SCIENCE

The goto statement is available in some of the old programming language, such as C, and Basic. You basically can move to any position in your program.

The (was) famous statement: GOTO

- The elimination of the GOTO was based on what programmers had learned from using it—that it made the program very hard to understand.
- Programs with GOTOs were often referred to as ***spaghetti code*** because the sequence of instructions that got executed could be as hard to follow as a single strand in a bowl of spaghetti.

```
template <typename T>
void goto_sort( T array[], size_t n ) {
        size_t i{1}

        first: T current{array[i]};
        size_t j{i};

        second: if ( array[j - 1] <= current ) {
                goto third;
        }

        array[j] = array[j - 1];

        if ( --j ) {
                goto second;
        }

        third: array[j] = current;

        if ( ++i != n ) {
                goto first;
        }
}
```

FACULTY OF COMPUTER SCIENCE

However, it may cause difficulty in reading and analyzing the programing. The above code is actually doing *Insertion Sort* algorithm. But it is not easy to read it, it even more difficult to analyze the complexity of the algorithm. Those goto statement cause the program to move to others line, and it looks like spaghetti. The term ***spaghetti code*** to describe a complicated program may come from this. Nowadays, we the emerging of clean code, it is definitely not accepted to write such complicated code. We should make our program to be easily read by other programmer.

GOTO must go

- The inability of these developers to understand how their code worked, or why it sometimes didn't work, was a complexity problem.
- Software experts of that era believed that those GOTO statements were creating unnecessary complexity and that the GOTO had to, well, go.
- Article: "Letters to the editor: go to statement considered harmful", by ***Edsger W Dijkstra***, Communication of the ACM, vol. 11, No. 3, 1968

Letters to the editor: go to statement considered harmful

Author: Edsger W. Dijkstra Authors Info & Claims

Communications of the ACM, Volume 11, Issue 3 • pp 147–148 • https://doi.org/10.1145/362929.362947

Published: 01 March 1968 Publication History

Check for updates

971 15,807

eReader PDF

Letters to the Editor

Go To Statement Considered Harmful

Key Words and Phrases: go to statement, jump instruction

dynamic progress is only characterized when we also give to which call of the procedure we refer. With the inclusion of procedure

FACULTY OF COMPUTER SCIENCE

In the year 1968, Dijkstra, who is known for his Shortest Path Algorithm on graphs, already complaint about it. He already identified that this goto statement is not good at all and it should be removed from any programming language.

GOTO must go

- Back then, around 70s, this was a radical idea, and many programmers resisted the loss of a statement that they had grown to rely on.
- The debate went on for more than a decade, but in the end, the GOTO went extinct, and no one today would argue for its return.
- That's because its elimination from higher-level programming languages greatly reduced complexity and boosted the reliability of the software being produced.
- It did this by limiting what programmers could do, which ended up making it easier for them to reason about the code they were writing.

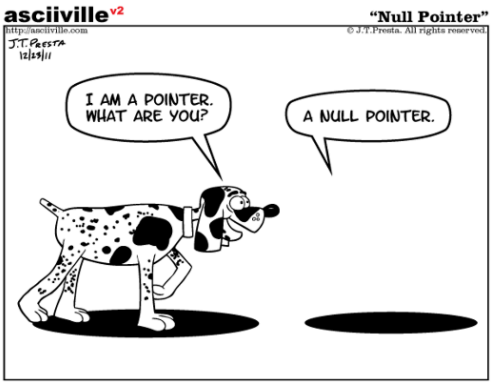
FACULTY OF COMPUTER SCIENCE

Nowadays, we are not seeing any modern language has goto like statement in the language. Finally people realize it although in the some people who already get used to it do not agree.

There is also an issue with null reference, such as we have in Java.

Null reference

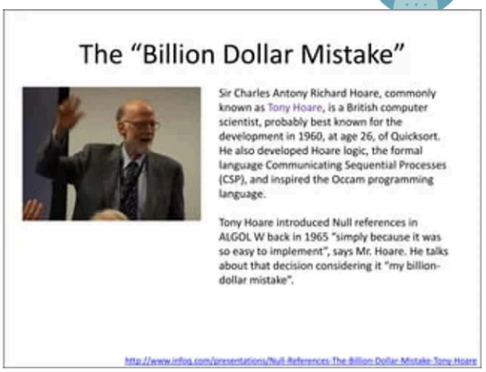
- Another complexity monster lurking in the software quagmire is called a null reference,
- meaning that a reference to a place in memory points to nothing at all.
- If you try to use this reference, an error ensues.
- So programmers have to remember to check whether something is null before trying to read or change what it references.



FACULTY OF COMPUTER SCIENCE

Null reference

- Nearly every popular language today has this flaw. The pioneering computer scientist Tony Hoare introduced null references in the ALGOL language back in 1965, and it was later incorporated into numerous other languages.
- Hoare explained that he did this "simply because it was so easy to implement," but lately he considers it to be a "billion-dollar mistake."
- That's because it has caused countless bugs when a reference that the programmer expects to be valid is really a null reference.



FACULTY OF COMPUTER SCIENCE

Hoare, who developed an algorithm of partion in quick sort, was the one who popularized it and also the one who admitted that is should not be there actually. It was just an easy hacking back then, but it should not be followed by other programming languages. He said it was the "Billion Dollar Mistake" due to so many programming bug of this null reference.

Extra effort from programmer vs Limitation

- Software developers need to be extremely disciplined to avoid such pitfalls, and sometimes they don't take adequate precautions.
- The architects of structured programming knew this to be true for GOTO statements and left developers no escape hatch.
- To guarantee the improvements in clarity that GOTO-free code promised, they knew that they'd have to eliminate it entirely from their structured-programming languages.

FACULTY OF COMPUTER SCIENCE

Extra effort from programmer vs Limitation

- History is proof that ***removing a dangerous feature can greatly improve the quality of code***.
- Today, we have a few of dangerous practices that compromise the robustness and maintainability of software.
- Nearly all modern programming languages have some form of null references, shared global state, and functions with side effects—things that are far worse than the GOTO ever was.

How can those flaws be eliminated?
It turns out that the answer has been around for decades:
purely functional programming languages! ☺

Purely Functional Programming has limit possibility of conducting error. But it has not been popular for business purpose due to its model of computation. The programming world is still dominated by imperative programmer. Functional programming has it own niece. Some applications such as whatsapp, discord, some game or some functionality such as mail spam detection in Facebook can take benefits of functional programming. Interested students may consider to take the Functional Programming as theirs elective course.

Rust Ownership Type

Rust as an imperative language has some language features which are not available is most imperative language. One importance aspect is the ***Ownership Type***.

Rust comes to the rescue!

- Purely functional programming is cool.
- But it lacks pragmatic aspect of today software engineering needs.
- Rust has the pragmatic aspect, but also some constraint and some positive aspect of functional programming as well.
- Such as monad of Haskell can be mimic using trait in Rust due to Rust strong and powerful type system.

FACULTY OF COMPUTER SCIENCE

Rust Ownership Type

- Ownership is Rust's most unique feature and has deep implications for the rest of the language.
- It enables Rust to make memory safety guarantees without needing a garbage collector,
- so it's important to understand how ownership works.

- Ownership is a discipline for ensuring the safety of Rust programs, including concurrency!

FACULTY OF COMPUTER SCIENCE

Safety is the absence of undefined behavior

- Let's modify:

```
fn read(y: bool) {
    if y {
        println!("y is true!");
    }
}

fn main() {
  read(x); // oh no! x isn't defined!
  let x = true;
}
```

FACULTY OF COMPUTER SCIENCE

Safety is the absence of undefined behavior

- When a program like this is executed by an interpreter, then reading ***x*** before it's defined would raise an exception such as Python's NameError or Javascript's ReferenceError.
- But exceptions come at a cost. Each time an interpreted program reads a variable, then the interpreter must check whether that variable is defined.
- Rust is to prevent undefined behavior at *compile-time* instead of *run-time*.

FACULTY OF COMPUTER SCIENCE

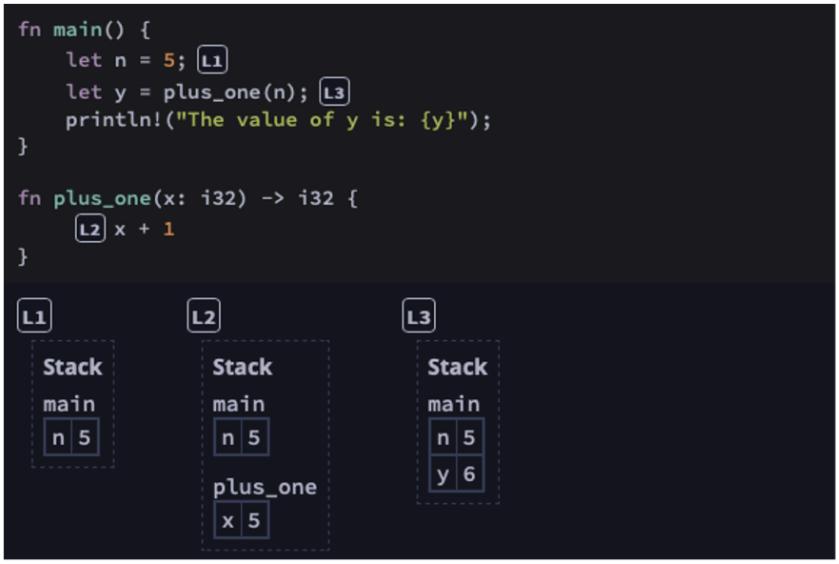
Having a strong type system that support ownership type, allow Rust to be able to check some possible error during compile time. This is definitely a savior for programmer. Checking some possible error during run-time is very difficult and required advanced and experience programmer. It is still needed, but reduce because some can be checked in compile time.

Ownership as a Discipline for Memory Safety

- The Rust Reference maintains a large list of "Behavior considered undefined".
- For now, we will focus on one category: operations on memory.
- Memory is the space where data is stored during the execution of a program. There are many ways to think about memory:
  - If you are unfamiliar with systems programming, you might think of memory at a high level like "memory is the RAM in my computer" or "memory is the thing that runs out if I load too much data".
  - If you are familiar with systems programming, you might think of memory at a low level like "memory is an array of bytes" or "memory is the pointers I get back from malloc".

FACULTY OF COMPUTER SCIENCE

Variables Live in the Stack



FACULTY OF COMPUTER SCIENCE

In Rust, variables live in the stack. When the stack is gone, the variables will automatically also gone.

Boxes Live in the Heap (Construct Box)

We can also create an allocation of memory that stay in the heap memory which is still stay live, even if the method has closed. Here, we use the Box.

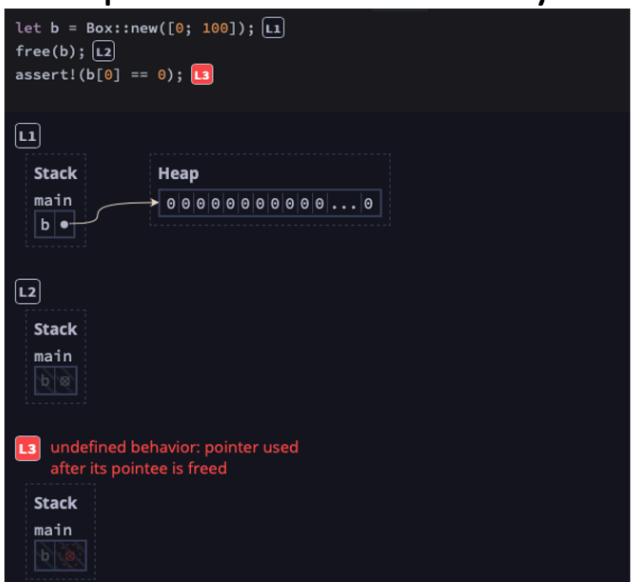
Observe that now, there is only ever a single array at a time. At L1, the value of a is a pointer (represented by dot with an arrow) to the array inside the heap. The statement let b = a copies the pointer from a into b, but the pointed-to data is not copied. Note that a is now grayed out because it has been *moved* — we will see what that means in a moment.

Rust does not permit manual memory management

- Memory management is the process of allocating memory and deallocating memory. In other words, it's the process of finding unused memory and later returning that memory when it is no longer used. Stack frames are automatically managed by Rust. When a function is called, Rust allocates a stack frame for the called function. When the call ends, Rust deallocates the stack frame.
- As we saw previously, heap data is allocated when calling Box::new(..). But when is heap data deallocated? Imagine that Rust had a free() function that frees a heap allocation. Imagine that Rust let a programmer call free whenever they wanted.
- This kind of "manual" memory management easily leads to bugs.

FACULTY OF COMPUTER SCIENCE

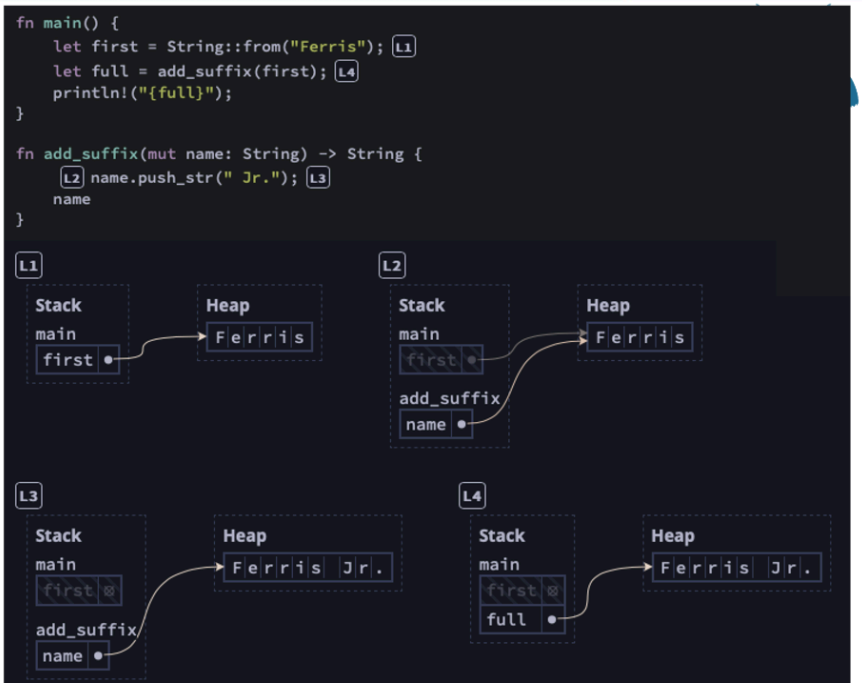
Rust does not permit manual memory management



FACULTY OF COMPUTER SCIENCE

Collections use Boxes

Boxes are used by Rust data structures[1] Vec, String, and HashMap to hold a variable number of elements.



Some other data structure such as Vector, HashMap, and also String are using Box as well. The example show how Box works on String. The data remain in the heap memory allocation. The reference to this data is passing through method call and return value.

Borrowing

Variables cannot be used after being moved

```
fn main() {
    let first = String::from("Ferris");
    let full = add_suffix(first);
    println!("{full}, originally {first}"); L1 // first is now used here
}

fn add_suffix(mut name: String) -> String {
    name.push_str(" Jr.");
    name
}
```

L1 undefined behavior: pointer used after its pointee is freed

The previous program will not be compiled. There is some restriction on using variables.

Variables cannot be used after being moved

```
error[E0382]: borrow of moved value: `first`
 --> test.rs:4:35
  |
2 |     let first = String::from("Ferris");
  |         ----- move occurs because `first` has type `String`, which does not implement the `Copy` trait
3 |     let full = add_suffix(first);
  |                           ----- value moved here
4 |     println!("{full}, originally {first}"); // first is now used here
  |                                   ^^^^^ value borrowed here after move
```

- **Moved heap data principle:**
  if a variable ***x*** moves ownership of heap data to another variable ***y***, then ***x*** cannot be used after the move.

- If you want, you can explicitly clone it. (separate copy)

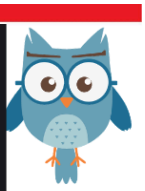
The ownership of variable has to be maintained. The compile message saying that previouly the ownership of first is the main, but then the ownership was moved due to giving it as paramater to the method add_suffix. So, the main does not hold the ownership of first anymore. Therefor it may not read again on line 4. So instead of giving the ownership we introduce the concept of borrowing.

References

`&m1`

meaning

a reference to `m1`



FACULTY OF COMPUTER SCIENCE

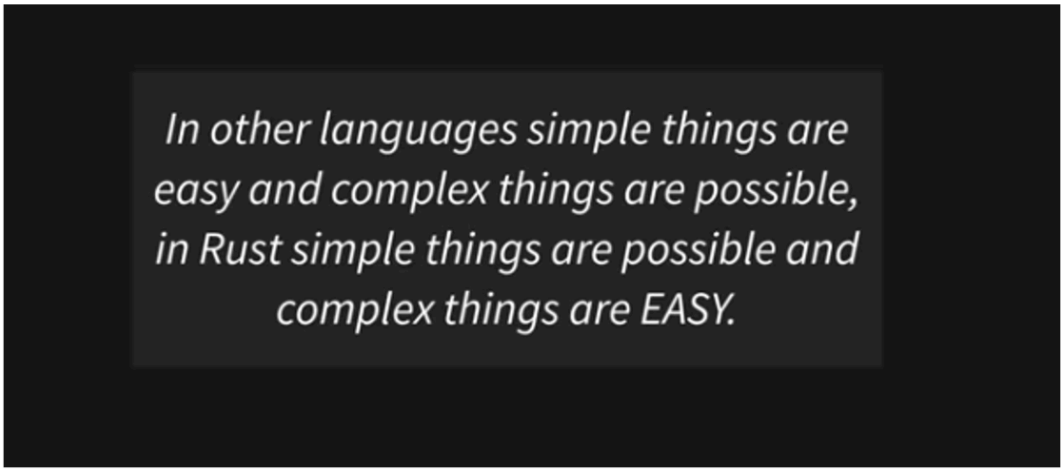
Here, the method greet borrow the variable m1 and m2 by having additional & (ampersand) code in the method parameter. It means the variable m1 and m2 ownership still belong to the main method, but the greet method only borrowing. Therefor after the method greet finish using them, it return it back to the main method, and the main method can access them again.

The symbol & (ampersand) is actually refer to the reference of a variable. It is the same as reference in Java. However, the Rust adding it with the ownership concept.

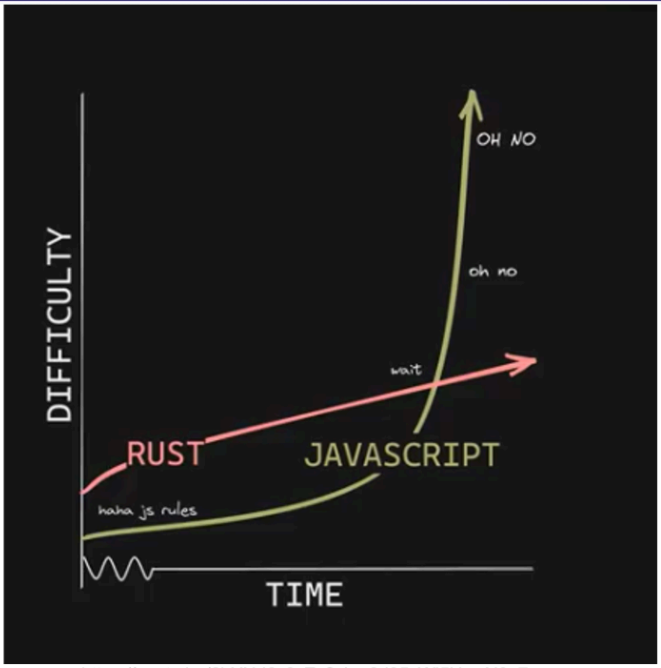
And more….

- Error handling – no exception but Option like Haskell
- Traits – as a replacement for inheritances (re-use)
- Lifetime references – avoiding dangling and garbage collector
- Macro – metaprogramming, code generator

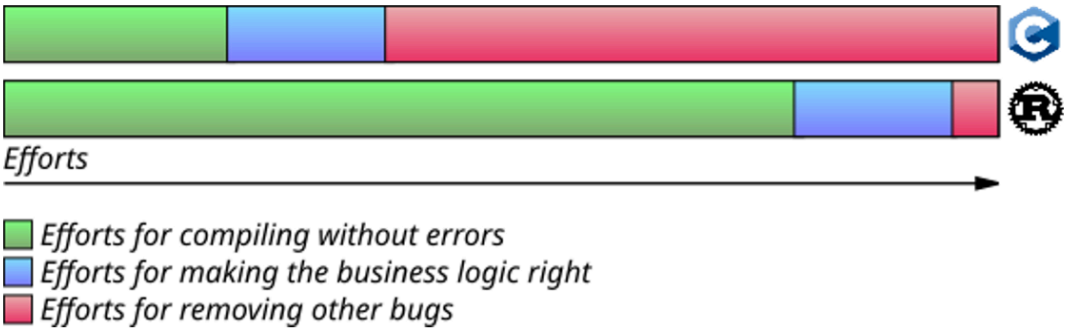
There are more interesting concepts of Rust that are beyond this course. They are quite interesting. Especially because Rust in designed for hard task. Students are suggested to watch some video of experience Rust professional programmer. Such as:
https://youtu.be/2hXNd6x9sZs?si=g8d95vWl5HywN0n7



https://youtu.be/2hXNd6x9sZs?si=g8d95vWl5HywN0n7



https://youtu.be/2hXNd6x9sZs?si=g8d95vWl5HywN0n7



https://www.linkedin.com/pulse/why-rust-easy-tue-henriksen/

Compare to C, the effort for making the business logic right is almost equal, but the effort of to make it compile and handling bugs is significant different. By using Rust, we are less worried about the possible bug or run-time issues. This is definitely a savior for programmer. This is coming from experience professional as can be found in: https://www.linkedin.com/pulse/why-rust-easy-tue-henriksen/

Let's Study Rust

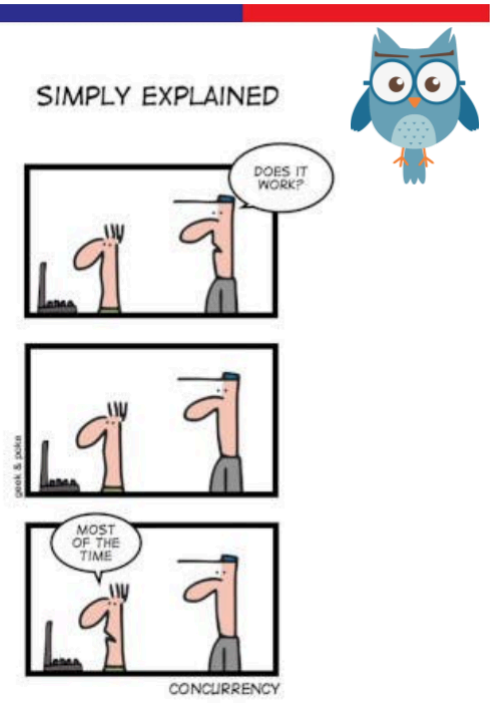
- Try it on: https://play.rust-lang.org
- Online book : https://rust-book.cs.brown.edu/title-page.html
- Install it using rustup : https://rustup.rs
- Build system and more using cargo : https://github.com/rust-lang/cargo
- IDE support using rust-analyzer : https://rust-analyzer.github.io
- Package collection is called crates : https://crates.io
    - grpc using tonic : https://crates.io/crates/tonic
    - Web framework using rocket : https://crates.io/crates/rocket

4. Concurrency

Concurrency is hard!

- Concurrency is difficult!
  - Critical section,
  - Race condition,
  - Lost update,
  - Synchronization (mutex),
  - deadlock detection.
- Concurrency is nondeterministic.
- It is hard to test!
- How to safely and efficiently program concurrency?

FACULTY OF COMPUTER SCIENCE



Concurrency is a next level of programming. The programmer is expected to understand how the machine works. The non-deterministic behavior is mostly due to variation of scheduler and the load that the machine has. It is not completely non-deterministic, but due to huge number of variables it is impossible to guarantee a deterministic behavior. Some experiences programmers may be able to guess the result correctly but it will need a lot of time and deep thinking including understanding of how the system works.

It is also difficulty to rely on unit test and TDD, because the non-deterministic behavior of several threads. Deadlock detection is also not very easy it may be worth to have a course of its own to discuss deadlock and it detection algorithm.

Due to this complexity, it is mostly possible to cause error while doing concurrency which motivated previously to improve its performance. Some programmers choose not to refactor to a multi-threaded program to avoid these issues. There is also a fairy tale of NASA programmer who try to improve the performance of a program that control satellite. It is system level programming, related to multi-threading, and It resulted to the satellite can be control any more due to the error in multi-threading. It is a scary thing.

Fearless Concurrency

- It is major goal in Rust, to be able to safely and efficiently programming concurrency in Rust!
- Ownership and type system -> memory safety, ~~concurrency problem~~
- Many errors are detected during compile time
- ~~Test-Driven Development~~ -> Compiler Driven Development

FACULTY OF COMPUTER SCIENCE

Rust is designed tohelp us in doing concurrency better. We should not be afraid anymore in doing concurrency. Having ownership type system, has lead to a better memory safety and reduce most of common programming bugs. Those errors are already detected during compile time. Since test is hard, it position to guarantee program correct is replace by the compiler. So, the rust community has a slogan of Compiler Driven Development to help the programmer write a trust worthy code.

In general Rust support for concurrency is both useful for high level abstraction and low level abstraction. In Concurreny model there are two model: Message Passing and Shared memory model. Both can be applied in Rust. Usually, message passing is for high level abstraction, and shared memory model for a lower level abstraction. But those method can also be used together in one program.

We will do some programming in Rust, first we do multi-threading, and then apply the shared memory model as the concept has been familiarized in operating system course, and then we will show briefly about message passing. This model is more popular for programmer, but it also to some degree need or may be use shared memory model as well. So students, need to understand those concept well.

Concurrency -> Multi-threading

- A ***process*** can have several ***threads*** running concurrently.
- Creating a new thread with in a program -> `thread::spawn`

```
use std::thread;
use std::time::Duration;

fn main() {
    thread::spawn(|| {
        for i in 1..10 {
            println!("hi number {} from the spawned thread!", i);
            thread::sleep(Duration::from_millis(1));
        }
    });

    for i in 1..5 {
        println!("hi number {} from the main thread!", i);
        thread::sleep(Duration::from_millis(1));
    }
}
```

FACULTY OF COMPUTER SCIENCE

The operator || is related to the closures as the parameter for spawn. Closures is described in Chapter 13 of the Rust book. Closures is related to functional programming. To make the long story short. The operator is used to passed something to the spawned thread. In this example, we don't pass anything. In other cases we can write something like thread:spawn(move || { …} which indicated it will pass the move to the thread. We will discuss about it in a few more slides.

Basically the program we create two threads. The main thread will spawn another thread, lets call it *spawned* thread. This *spawned* thread will run a loop 9 times. It will print a number and sleep for 1 mill second. The sleep is actually just use here to simulate slow response, because if you run in on some very fast computer, the multi-threading may cannot be observed. So, we put additional sleep to slow it down a bit. The main thread after created the child thread, runs a loop for 4 times.

As we understand running it on a computer may have different behavior. Here is an example of result.

Concurrency -> Multi threading

```rust
use std::thread;
use std::time::Duration;

fn main() {
    thread::spawn(|| {
        for i in 1..10 {
            println!("hi number {} from the spawned thread!", i);
            thread::sleep(Duration::from_millis(1));
        }
    });

    for i in 1..5 {
        println!("hi number {} from the main thread!", i);
        thread::sleep(Duration::from_millis(1));
    }
}
```

```
hi number 1 from the main thread!
hi number 1 from the spawned thread!
hi number 2 from the main thread!
hi number 2 from the spawned thread!
hi number 3 from the main thread!
hi number 3 from the spawned thread!
hi number 4 from the main thread!
hi number 4 from the spawned thread!
hi number 5 from the spawned thread!
```

You may try to run in in the book: https://rust-book.cs.brown.edu/ch16-01-threads.html

You may found possible other result of the program.

In this example output, the program stops here. If we look at the code, the spawned thread should print upto 9, but why it already stops on 5 ?

What happen in your tries? Which number are the last to print. Can you see the pattern, or the commonality of several run?

You may see that after the main thread print 4, it will soon finish. If the main thread as finish its job. It will close and close all other thread it has spawned. So the spawned thread since it spawned from the main thread, it will also be close when the main thread finish.

We can force the main thread to wait by adding join().unwrap().

join.unwrap()

```
use std::thread;
use std::time::Duration;

fn main() {
    let handle = thread::spawn(|| {
        for i in 1..10 {
            println!("hi number {} from the spawned thread!", i);
            thread::sleep(Duration::from_millis(1));
        }
    });

    for i in 1..5 {
        println!("hi number {} from the main thread!", i);
        thread::sleep(Duration::from_millis(1));
    }

    handle.join().unwrap();
}
```

FACULTY OF COMPUTER SCIENCE

This join.unwrap() will make sure the main thread to wait for the thread that it handle to finish first before the main thread continue to finish. Good, we have our first multi-threading program. However, this program has no variable yet, let see other example with variable.

Using move *closures* with *threads*

Let's see an example:

```
use std::thread;

fn main() {
    let v = vec![1, 2, 3];

    let handle = thread::spawn(|| {
        println!("Here's a vector: {:?}", v);
    });

    handle.join().unwrap();
}
```

FACULTY OF COMPUTER SCIENCE

Hmm, this program does not compile. Why?

Using move *closures* with *threads*

```
error[E0373]: closure may outlive the current function, but it borrows `v`, which is owned by the current
function
 --> src/main.rs:6:32
  |
6 |     let handle = thread::spawn(|| {
  |                                ^^ may outlive borrowed value `v`
7 |         println!("Here's a vector: {:?}", v);
  |                                           - `v` is borrowed here
  |
note: function requires argument type to outlive `'static`
 --> src/main.rs:6:18
  |
6 |       let handle = thread::spawn(|| {
  |  __________________^
7 | |         println!("Here's a vector: {:?}", v);
8 | |     });
  | |_____^
help: to force the closure to take ownership of `v` (and any other referenced variables), use the `move`
keyword
  |
6 |     let handle = thread::spawn(move || {
  |                                ++++

For more information about this error, try `rustc --explain E0373`.
error: could not compile `threads` due to previous error
```

FACULTY OF COMPUTER SCIENCE

The compiler complaint about the variable v being move? What is the problem with this, what can be wrong? Why the compiler prevents it?

Using move *closures* with *threads*

Why, should the compiler complaints? Let assume the compiler allow it, and we have some modification as below:

```
use std::thread;

fn main() {
    let v = vec![1, 2, 3];

    let handle = thread::spawn(|| {
        println!("Here's a vector: {:?}", v);
    });

    drop(v);   // it may cause the thread to fail!

    handle.join().unwrap();
}
```

FACULTY OF COMPUTER SCIENCE

As a programmer, which later be a professional programmer. You will code in team. You will write a long code together with your fellow programmer. If the compiler not complaint about it,

it is possible that some of your fellow programmer may accidently add the drop(v) statements. This drop(v) statement will remove variable v, while actually variable v is still being used by other threads. This may case unexpected behavior. So, the compiler help us avoiding this issues.

We can also use the move closures to make sure that no drop statement or similar code could be added.

Using move *closures* with *threads*

By adding ***move closures***, we force the *closures* to have *ownership*.

```
use std::thread;

fn main() {
    let v = vec![1, 2, 3];

    let handle = thread::spawn(move || {
        println!("Here's a vector: {:?}", v);
    });

    handle.join().unwrap();
}
```

FACULTY OF COMPUTER SCIENCE

This program will compile, but it will explicitly disallow programmer to add drop statement. So if someone modified the program:

Using move *closures* with *threads*

Since the *ownership* has been ***move*** to the *thread*, the ***main*** cannot **drop(v)**

```
use std::thread;

fn main() {
    let v = vec![1, 2, 3];

    let handle = thread::spawn(move || {
        println!("Here's a vector: {:?}", v);
    });

    drop(v);   // it is not allowed after move!

    handle.join().unwrap();
}
```

FACULTY OF COMPUTER SCIENCE

It will complaint:

Using move *closures* with *threads*

```
$ cargo run
   Compiling threads v0.1.0 (file:///projects/threads)
error[E0382]: use of moved value: `v`
  --> src/main.rs:10:10
   |
4  |     let v = vec![1, 2, 3];
   |         - move occurs because `v` has type `Vec<i32>`, which does not implement the `Copy`
trait
5  |
6  |     let handle = thread::spawn(move || {
   |                                ------- value moved into closure here
7  |         println!("Here's a vector: {:?}", v);
   |                                           - variable moved due to use in closure
...
10 |     drop(v); // oh no!
   |          ^ value used here after move

For more information about this error, try `rustc --explain E0382`.
error: could not compile `threads` due to previous error
```

FACULTY OF COMPUTER SCIENCE

The compiler protects the programmer for accidently make such a mistake. We call it *Compiler Driven Development*. Just let the compiler guide you, it will simplify how you code and reduce your effort to check it on run-time. Just carefully and patiently read the compiler message.

Shared Memory Model

- Remember the issues of multithreading in Operating Systems:
  - Critical section (shared memory)
  - Race condition (several threads try to access them)
  - Lost update (it will happen if we are not careful)
  - Synchronization (***Mutual Exclusion***, an approach to avoid lost update)
  - Deadlock (careless synchronization , may yield to deadlock, need nee to avoid it)

In the operating system course, we have heard about mutex. It is a mechanism to guarantee only mutual exclusion to a shared memory.

The issues on mutex

- Mutex is known to be difficult.
- When using mutex, we have to be careful in these two rules:
  - A thread attempt to acquire lock before using the data
  - The thread should unlock the data after using it.
- In big program, managing these two rules can be very tricky.
- It is possible that a programmer forgot to unlock it. His threads may run properly, but not the other threads.

Example: mutex

```
use std::sync::Mutex;
use std::thread;

fn main() {
    let counter = Mutex::new(0);
    let mut handles = vec![];

    for _ in 0..10 {
        let handle = thread::spawn(move || {
            let mut num = counter.lock().unwrap();

            *num += 1;
        });
        handles.push(handle);
    }

    for handle in handles {
        handle.join().unwrap();
    }

    println!("Result: {}", *counter.lock().unwrap());
}
```

FACULTY OF COMPUTER SCIENCE

This is a standard type of programming with mutex, but it is written in Rust. The program is expected to create 10 threads, each thread add 1 to a critical section of shared memory with. To avoid lost update the counter is consider to be mutex. The main program and then wait until each thread finish then it print the counter. We expect it will print 10. But the Rust compiler complaints!

Example

```
$ cargo run
   Compiling shared-state v0.1.0 (file:///projects/shared-state)
error[E0382]: use of moved value: `counter`
  --> src/main.rs:9:36
   |
5  |     let counter = Mutex::new(0);
   |         ------- move occurs because `counter` has type `Mutex<i32>`, which does not
implement the `Copy` trait
...
9  |         let handle = thread::spawn(move || {
   |                                    ^^^^^^^ value moved into closure here, in previous
iteration of loop
10 |             let mut num = counter.lock().unwrap();
   |                           ------- use occurs due to use in closure

For more information about this error, try `rustc --explain E0382`.
error: could not compile `shared-state` due to previous error
```

FACULTY OF COMPUTER SCIENCE

The compile message says: "value moved into closure here, in previous iteration of the loop". The move, is moving the ownership of variable. The main has move the ownership to one of its spawned thread, so it cannot move it again to another thread. If we really want to do this, we can use Arc<T>. The symbol <T> is similar to generic in Java.

Atomic Reference Counting: Arc <T>

- If we would like go give an ownership to several owner we can use the *Atomic Reference Counting* **Arc <T>**
- Remember Box<T> , it will create a variable of type T in the *heap* memory allocation, not in the *stack*. It is like generic in Java.
- The type Arc<T> will create a variable of type T, but it can share it ownership to several owners in a *thread-safe* manner.
- There is a non thread-safe version called Rc<T>
- Why Rust differentiates it?

FACULTY OF COMPUTER SCIENCE

So this library is allowing as to share the ownership to several owner. The non-thread-safe version is called Rc<T>. The Rust differentiates because Arc<T> has some atomic constraints, which in some cases is slower. But the atomic constraints is significant for multi-threading. The Rc<T> is faster version but not ready for multi-threading.

Example: mutex with Arc<T>

```
use std::sync::{Arc, Mutex};
use std::thread;

fn main() {
    let counter = Arc::new(Mutex::new(0));
    let mut handles = vec![];

    for _ in 0..10 {
        let counter = Arc::clone(&counter);
        let handle = thread::spawn(move || {
            let mut num = counter.lock().unwrap();

            *num += 1;
        });
        handles.push(handle);
    }

    for handle in handles {
        handle.join().unwrap();
    }

    println!("Result: {}", *counter.lock().unwrap());
}
```

Here is a better version.

So, we have an example of shared memory model.

Message Passing

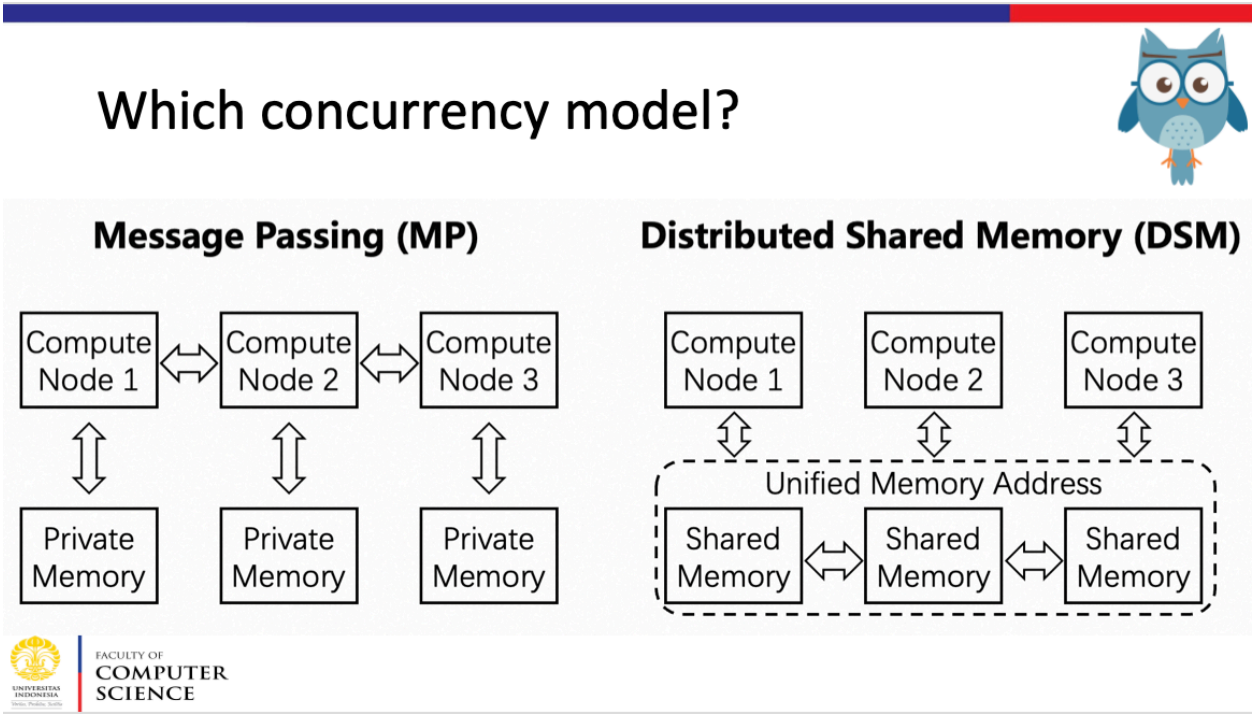


Image Source: https://www.alibabacloud.com/blog/revisiting-distributed-memory-in-the-cxl-era_600809

Other then shared memory model, we have message passing model. This model is more popular for business programmer. It basically not allowing any shared memory, and just let the thread or process communicate through channel. It will remove the issue of lost update, at least beside accessing the channel.

Message Passing in Rust

- Rust's library provides an implementation of ***channel***
- A ***channel*** is a general programming concept by which data is sent from one *thread* to another *thread*
- A ***channel*** has two halves, a *transmitter* and a *receiver*.

```
use std::sync::mpsc;

fn main() {
    let (tx, rx) = mpsc::channel();
}
```

FACULTY OF COMPUTER SCIENCE

The mpsc stands for *Multiple Producer Single Consumer*. So it means many thread can add anything to the channel, but only single thread can read in (or unless it specifically shared by using mutex)

Output:

```
Got: hi
```

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

fn main() {
    let (tx, rx) = mpsc::channel();

    thread::spawn(move || {
        let val = String::from("hi");
        tx.send(val).unwrap();
    });

    let received =
rx.recv().unwrap();
    println!("Got: {}", received);
}
```

This program is compiled. One thread put the string "Hi" in the channel and another thread, read the channel int print it out.

It will not compile.

Why?

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

fn main() {
    let (tx, rx) = mpsc::channel();

    thread::spawn(move || {
        let val = String::from("hi");
        tx.send(val).unwrap();
        println!("val is {}", val);
    });

    let received = rx.recv().unwrap();
    println!("Got: {}", received);
}
```

We make small modification, by adding print statement inside the spawned thread. It did not compile anymore.

Compile messages

```
Compiling message-passing v0.1.0 (file:///projects/message-passing)
error[E0382]: borrow of moved value: `val`
  --> src/main.rs:10:31
   |
8  |         let val = String::from("hi");
   |             --- move occurs because `val` has type `String`, which
does not implement the `Copy` trait
9  |         tx.send(val).unwrap();
   |                 --- value moved here
10 |         println!("val is {}", val);
   |                               ^^^ value borrowed here after move
   |
```


Don’t be afraid of reading compile messages. It is your friends, it will safe you on run-time. It says, it already sent variable val to the channel so the thread do not have ownership any more.

Run it!

```
use std::sync::mpsc;
use std::thread;
use std::time::Duration;

fn main() {
    let (tx, rx) = mpsc::channel();

    thread::spawn(move || {
        let vals = vec![
            String::from("hi"),
            String::from("from"),
            String::from("the"),
            String::from("thread"),
        ];

        for val in vals {
            tx.send(val).unwrap();
            thread::sleep(Duration::from_secs(1));
        }
    });

    for received in rx {
        println!("Got: {}", received);
    }
}
```

If you run, this is one possible output:

```
Got: hi
Got: more
Got: from
Got: messages
Got: for
Got: the
Got: thread
Got: you
```

5. Tutorial WebServer (Student should do it and submit it)

This tutorial is based on Chapter 20 of The Rust Programming Language book. See this URL for more detail: https://rust-book.cs.brown.edu/ch20-00-final-project-a-web-server.html
Students should re-do it, make commit, push to repository, add reflection notes as part of Readme.md in his/her repository, give access to the teaching assistant and submit the repository to scele.

Tutorial: Web Server

- We well review some parts of the tutorial in class.
- We will build our own web server. It is just a simple one. To show how Rust is in action. For a better and production ready web server you may consider Rocket.
- We will do:
  1. Listen for TCP connections on a socket.
  2. Parse a small number of HTTP requests.
  3. Create a proper HTTP response.
  4. Improve the throughput of our server with a thread pool.

Hello!

Hi from Rust

FACULTY OF COMPUTER SCIENCE

In the course of "Pemrograman Berbasis Platform" you already create a web app, using standard web server framework such as Django. It is a complete system, where it consists of the library, template engine, migration, ORM, and the web-server and others. Now, we would like to create a simple web-server program.

First, student can do in their machine:

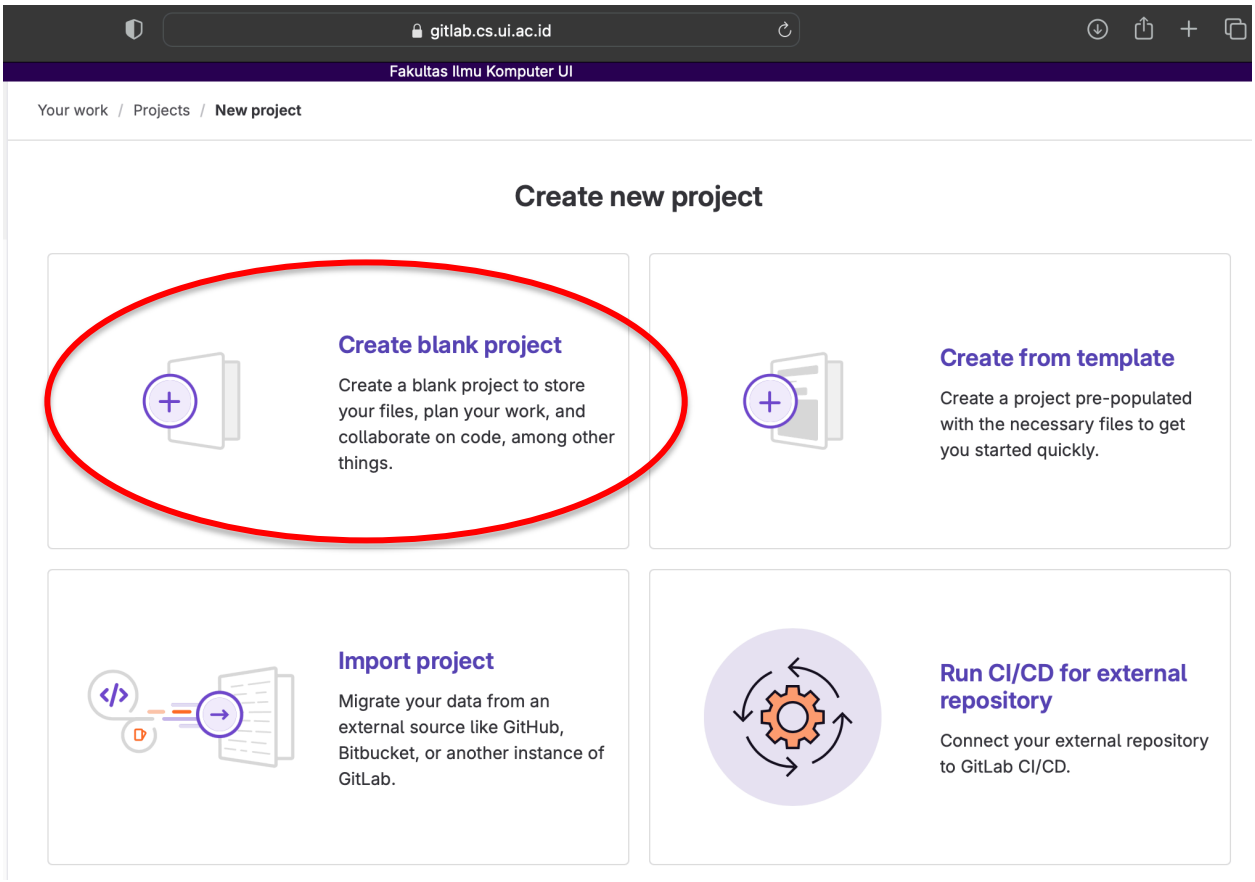
```
$ cargo new hello
     Created binary (application) `hello` project
$ cd hello
```

It will create a directory and it already initiate as git repository. You can try to push to your git repository on the server, but before that, you need to set URL of your repository that should already be initialized previously.

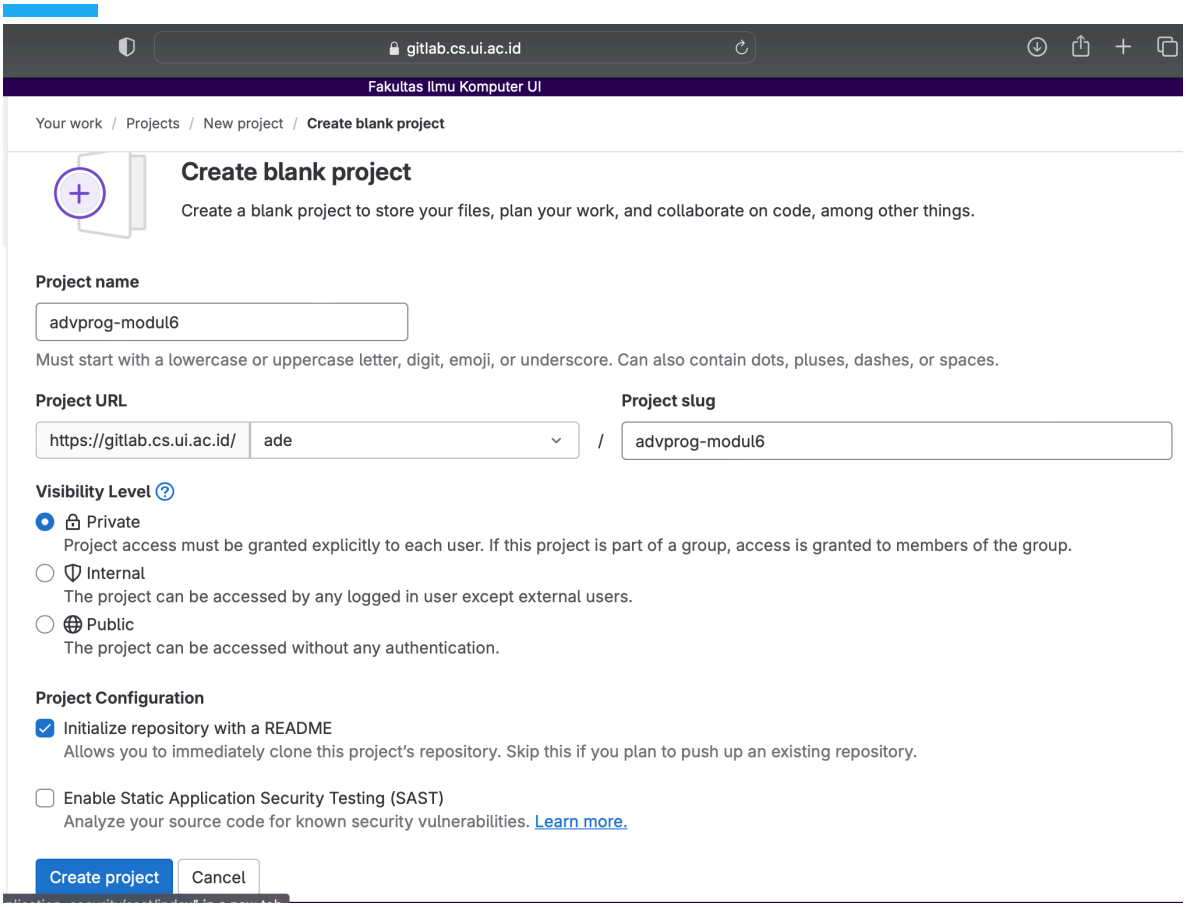
```
λ → git push
fatal: No configured push destination.
Either specify the URL from the command-line or configure a remote
repository using
```

```
    git remote add <name> <url>

and then push using the remote name
    git push <name>
```

This part of information can just be ignored for students who already familiar with git and able to solve the problem in the previous screen. If decided to follow this quick guide I will use gitlab.cs.ui.ac.id it can also be gitlab.com or github.com. For example, previously I already initiate a blank repository in the gitlab.cs.ui.ac.id



Then I write some information about the project, and some default configuration such as initialized with Readme.md



After I get the repository url, such as: https://gitlab.cs.ui.ac.id/ade/advprog-modul6.git

I do this in my machine: (you should use your own repository, and work on your directory)

```
λ → git remote add tutorial6 https://gitlab.cs.ui.ac.id/ade/advprog-
modul6.git

λ → git add .
λ → git commit -m "initialized"
[master (root-commit) 03ebc47] initialized
 3 files changed, 12 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 Cargo.toml
 create mode 100644 src/main.rs
λ → git push tutorial6
Username for 'https://gitlab.cs.ui.ac.id': ade
Password for 'https://ade@gitlab.cs.ui.ac.id':
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 526 bytes | 526.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
To https://gitlab.cs.ui.ac.id/ade/advprog-modul6.git
 * [new branch]      master -> master
```

This is basically can be applied with any git repository, either gitlab.com or github.com, choose your favorite one. Don't forget to give access to your teaching assistant just as the previous tutorial. If you choose to name it as example : tutorial6, make sure every you want to push, you do: `git push tutorial6`.

That is a brief information about git. Let's continue with the tutorial.

Milestone 1: Single threaded web server

Single-Threaded Web Server

- Listening to TCP connection

```
use std::net::TcpListener;

fn main() {
    let listener = TcpListener::bind("127.0.0.1:7878").unwrap();

    for stream in listener.incoming() {
        let stream = stream.unwrap();

        println!("Connection established!");
    }
}
```

FACULTY OF COMPUTER SCIENCE

```
use std::net::TcpListener;

fn main() {
    let listener = TcpListener::bind("127.0.0.1:7878").unwrap();

    for stream in listener.incoming() {
        let stream = stream.unwrap();

        println!("Connection established!");
    }
}
```

Filename: src/main.rs

Here is the code, you can copy paste this code into the mentined files. You can replace the old version of the files. Now you can try run using your ide, or type in your console: `cargo run`

And then form your browser, you try to access the following URL: http://127.0.0.1:7878

You browser would not show anything because we have write much code, but If you have access to your console, we can see something like this:

```
λ ➜ cargo run
   Compiling hello v0.1.0
(/Users/adeazurat/Workspaces/AdvProg/tutorial/hello)
warning: unused variable: `stream`
 --> src/main.rs:7:13
  |
7 |         let stream = stream.unwrap();
  |             ^^^^^^ help: if this is intentional, prefix it with an
underscore: `_stream`
  |
  = note: `#[warn(unused_variables)]` on by default

warning: `hello` (bin "hello") generated 1 warning (run `cargo fix --
bin "hello"` to apply 1 suggestion)
    Finished dev [unoptimized + debuginfo] target(s) in 0.78s
     Running
`/Users/adeazurat/Workspaces/AdvProg/tutorial/hello/target/debug/hello



Connection established!
Connection established!
Connection established!
```

This show that the program already listening to browser request. If you see several messages of "Connection established!" it possibly because your browser attempted to re-try the request when it didn't get anything. It is a normal behavior of a browser.

Let's continue. You may want to stop you server program by typing ctrl-c in the console or if you run an IDE, choose to stop it or kill it. (check you IDE guidance). If you don't do it, you may failed to run it the second time because the port 7878 is still being you by your previous still running program.

We would like to handle the request from the browser, but first we need to know how they communicate. What is a request from the browser?

Try to update your program with this one:

Filename: src/main.rs

```
use std::{
    io::{prelude::*, BufReader},
    net::{TcpListener, TcpStream},
};

fn main() {
    let listener = TcpListener::bind("127.0.0.1:7878").unwrap();
```

```
    for stream in listener.incoming() {
        let stream = stream.unwrap();

        handle_connection(stream);
    }
}

fn handle_connection(mut stream: TcpStream) {
    let buf_reader = BufReader::new(&mut stream);
    let http_request: Vec<_> = buf_reader
        .lines()
        .map(|result| result.unwrap())
        .take_while(|line| !line.is_empty())
        .collect();

    println!("Request: {:#?}", http_request);
}
```

Here we add method name handle_connection. This method should handle how the server will behave when it receives a connection request from a browser.

Run again your program, and then try to access it again from your browser http://127.0.0.1:7878 :

```
λ ➜ cargo run
   Compiling hello v0.1.0
(/Users/adeazurat/Workspaces/AdvProg/tutorial/hello)
    Finished dev [unoptimized + debuginfo] target(s) in 1.14s
     Running
`/Users/adeazurat/Workspaces/AdvProg/tutorial/hello/target/debug/hello`

Request: [
    "GET / HTTP/1.1",
    "Host: 127.0.0.1:7878",
    "Upgrade-Insecure-Requests: 1",
    "Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
    "User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/605.1.15 (KHTML, like Gecko) Version/16.1
Safari/605.1.15",
    "Accept-Language: en-US,en;q=0.9",
    "Accept-Encoding: gzip, deflate",
    "Connection: keep-alive",
]
```

This is an example of my machine. Yours will be different.

You may need to check the Rust documentation to understand what is inside the handle_connection method. Write as reflection notes in the Readme.md. Write it nicely.

(1) [Commit] Add reflection notes in Readme.md, put the title clearly such as Commit 1 Reflection notes. commit your works, put commit message "(1) Handle-connection, check response", and then push it to your repository.

Milestone 2: Returning HTML

We still don't have anything in our browser. We would like to have something. Let's modify the handle_connection method so it can show a simple html page that can be rendered by the browser.

Filename: src/main.rs

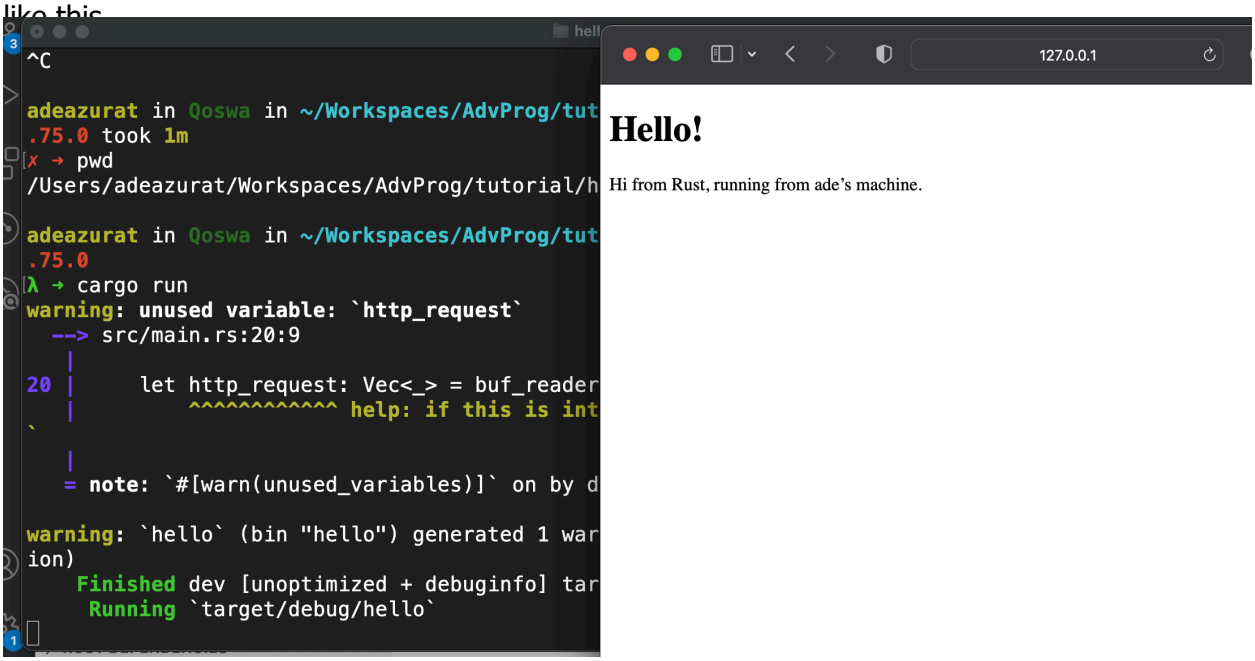
```
use std::{
    fs,
    io::{prelude::*, BufReader},
    net::{TcpListener, TcpStream},
};
// --snip--

fn handle_connection(mut stream: TcpStream) {
    let buf_reader = BufReader::new(&mut stream);
    let http_request: Vec<_> = buf_reader
        .lines()
        .map(|result| result.unwrap())
        .take_while(|line| !line.is_empty())
        .collect();

    let status_line = "HTTP/1.1 200 OK";
    let contents = fs::read_to_string("hello.html").unwrap();
    let length = contents.len();

    let response =
        format!("{status_line}\r\nContent-Length:
{length}\r\n\r\n{contents}");

    stream.write_all(response.as_bytes()).unwrap();
}
```

Filename: hello.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Hello!</title>
  </head>
  <body>
    <h1>Hello!</h1>
    <p>Hi from Rust, running from ade's machine.</p>
  </body>
</html>
```

When you type cargo run, make sure you are in the same directory as the location of the hello.html. You may need to read more regarding some of text that your program should write to the browser such "Content-Length" and others, check the chapter 20 or other resources on that. Write your own reflection of what you have learned about the new code the handle_connection.

When you run it (make sure you already killed the previous program), you may have something like this



But this is from my machine. You may need to edit the hello.html to write your own message. Capture your screen, put it as file such as commit2.pngand put it also the your Readme.md. You can find some guidance to put image into the markdown file of your Readme.md you can do something like this in your readme:

```
![Commit 2 screen capture](/assets/images/commit2.png)
```

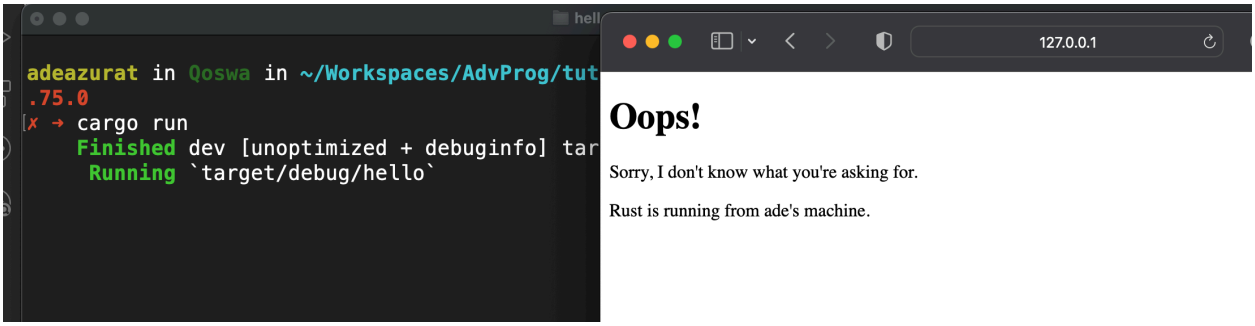
(2) [Commit] Complete your reflection on the new handle_connection in the readme.md, put the title clearly such as Commit 2 Reflection notes. commit with message "(2) Returning HTML", push it to your git repository server.

Milestone 3: Validating request and selectively responding

Now, our server will always return the hello.html no matter what. When you surf internet, sometime the server give respond that a page is not available. We would like to create a simple thing of that part. Just give a correct page and not available page, for the simplicity of this tutorial. Keep in mind, that a production ready web-server has more capability.

Follow the instruction in Chapter 20, part: Validating the Request and Selectively Responding

Now, if you ask for http://127.0.0.1:7878/bad in your browser, it give you something like this.



You better do up to the refactoring one, and you need to explain in your reflection notes, how to split between response and why the refactoring is needed. You also need to capture your own screen shot, with your own message.

(3) [Commit] Add additional reflection notes, put the title clearly such as Commit 3 Reflection notes. Commit your work with message "(3) Validating request and selectively responding". Push your commit.

Milestone 4: Simulation slow response

Our server is running on single thread. This is not good. Let simulate it to see the problem.

Let add this code to your previous program:

```
use std::{
    fs,
    io::{prelude::*, BufReader},
    net::{TcpListener, TcpStream},
    thread,
    time::Duration,
};
// --snip--

fn handle_connection(mut stream: TcpStream) {
    // --snip--

    let (status_line, filename) = match &request_line[..] {
        "GET / HTTP/1.1" => ("HTTP/1.1 200 OK", "hello.html"),
        "GET /sleep HTTP/1.1" => {
            thread::sleep(Duration::from_secs(10));
            ("HTTP/1.1 200 OK", "hello.html")
        }
        _ => ("HTTP/1.1 404 NOT FOUND", "404.html"),
    };

    // --snip--
}
```

Let's open two of browser windows, try 127.0.0.1/sleep in one of them, and try in other windows 127.0.0.1. Pay attention that the browser take some time to load. You can imagine if many users try to access it.

See how it works and try to understand why it works like that.

(4) [Commit] Add additional reflection notes, put the title clearly such as Commit 4 Reflection notes. Commit your work with message "(4) Simulation of slow request. "

Milestone 5: Multithreaded Server

Now we are motivated to improve our server. You can follow detail instruction from the book:

https://rust-book.cs.brown.edu/ch20-02-multithreaded.html

Try to understand how the ThreadPool works.

(5) [Commit] Add additional reflection notes, put the title clearly such as Commit 5 Reflection notes. Commit your work with message "(5) Multithreaded server using Threadpool "

Bonus: Try to create a function build as a replacement to new and compare.

(Bonus) [Commit] Add additional reflection notes, put the title clearly such as Commit Bonus Reflection notes. Commit your work with message "(Bonus) Function improvement"

Grading Scheme Tutorial 6

- Students should follow the instructions in the given module.
- Do the tutorial in the module.
- Make sure to do all the commits as requested.
- Do not squash your commit during the merge process. It may destroy your commits history and the teaching assistants cannot evaluate it.
- Write down your personal reflection notes as markdown in the readme.md on your master branch.
- Push your work to the repository.
- Give access to the teaching assistants.
- Submit the repository in the scele.

Scale

All components will be scored on discrete scale 0,1,2,3,4. Grade 4 is the highest grade, equivalent to A. No Partial score Score 0 is for not submitting.

Components

- 40% - Commits
- 40% - Reflection
- 20% - Correctness
- Bonus 10%:
    - Reflection (5%) and
    - Correctness (5%).

Rubrics Utama - 100 %

	Score 4	Score 3	Score 2	Score 1
Correctness	Correct perfectly as requested.	Something is missing. Or only do the tutorial parts. Exercise is missing.	A lot of requirements are missing, but the program still runs.	Incorrect program
Reflection (for each reflection)	More than 5 sentences. The description is sound.	Less than or equal 5 sentences. The description is sound.	The description is not sound although still related.	The description is not sound. It is not related to the topics.
Commit (evaluated on all commits)	Commit correctly	Commit is not correct.	Commit correctly but late	Incorrect commit