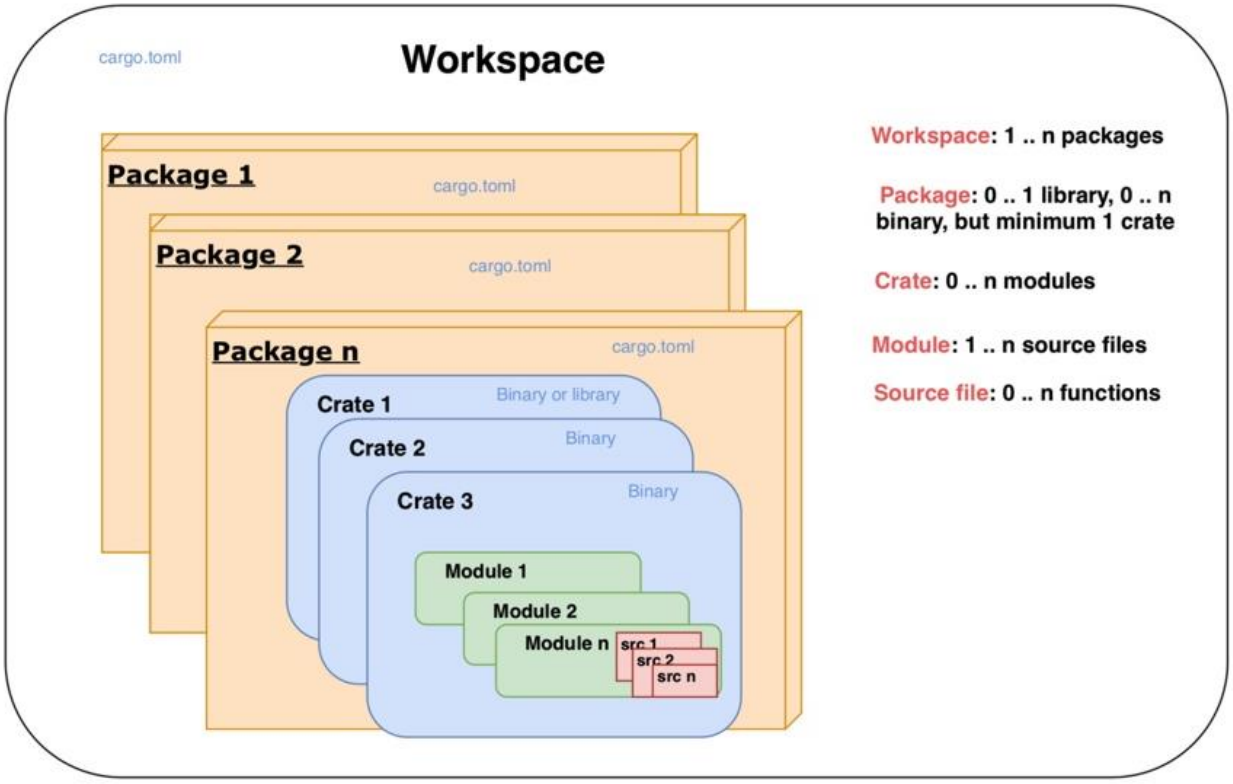
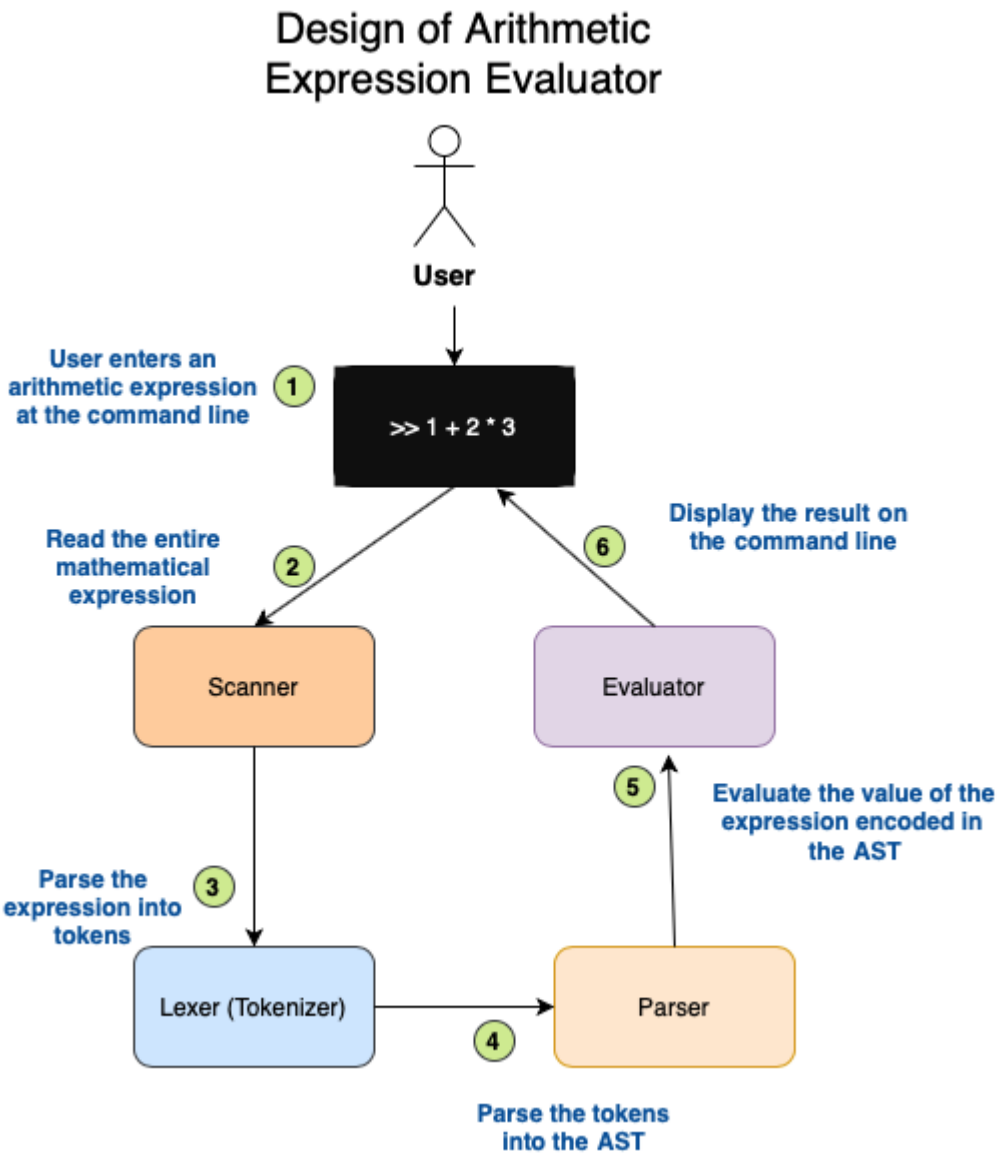
Chapter 1: Tools of the Trade – Rust Toolchains and Project Structures




```
├── Cargo.toml
├── src
│   ├── main.rs
```

Chapter 2: A Tour of the Rust Programming Language



Programming Language

Token enum design

enum Token

```
pub enum Token {
Add
Subtract
Multiply
Divide
Caret
LeftParen
RightParen
Num(f64)
EOF
}
```

Tokenizer data design

struct Tokenizer
Fields: **expr:** Peekable<Chars<'a>>
methods: TBD

Tokenizer with methods

new() : Creates a new instance of Tokenizer struct and stores the arithmetic expression in expression field

struct Tokenizer
Fields: **expr:** Peekable<Chars<'a>>
Public Methods: **new**() -> Tokenizer **next**() -> Option<Token>

next() : Converts arithmetic expression to Tokens

Design of Tokenizer module

Has methods to convert arithmetic expression to Tokens

enum Token

```
pub enum Token {
Add
Subtract
Multiply
Divide
Caret
LeftParen
RightParen
Num(f64)
EOF
}
```

struct Tokenizer

Fields:

expr: Peekable<Chars<'a>>

Public Methods:

new() -> Tokenizer
next() -> Option<Token>

User input:

5 + 2.2 * (3 - 4)

Tokenizer output:

Num(5.0), Add, Num(2.2), Mul, LeftParen, Num(3.0), Subtract, Num(4.0), RightParen, EOF

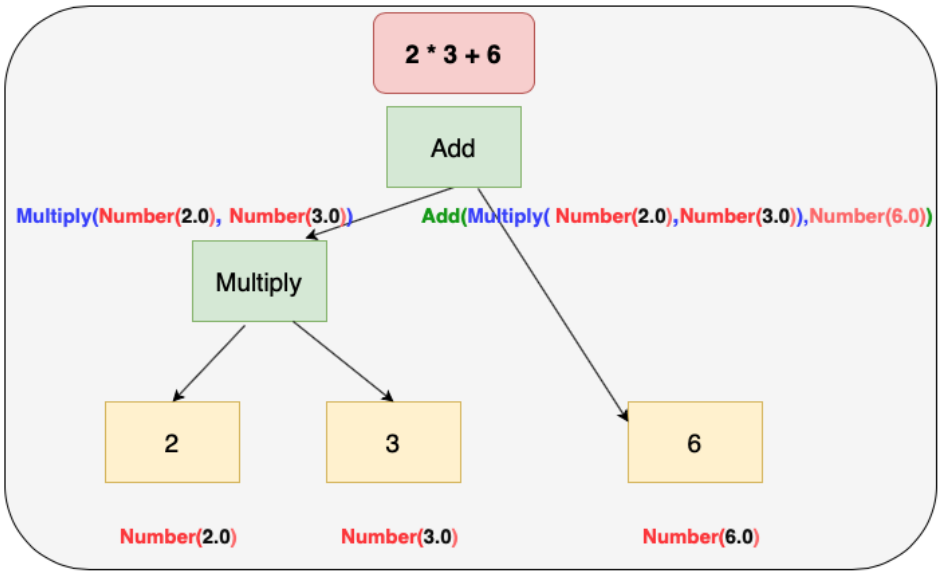
User

Example of Abstract Syntax Tree for expression
2 * 3 + 6

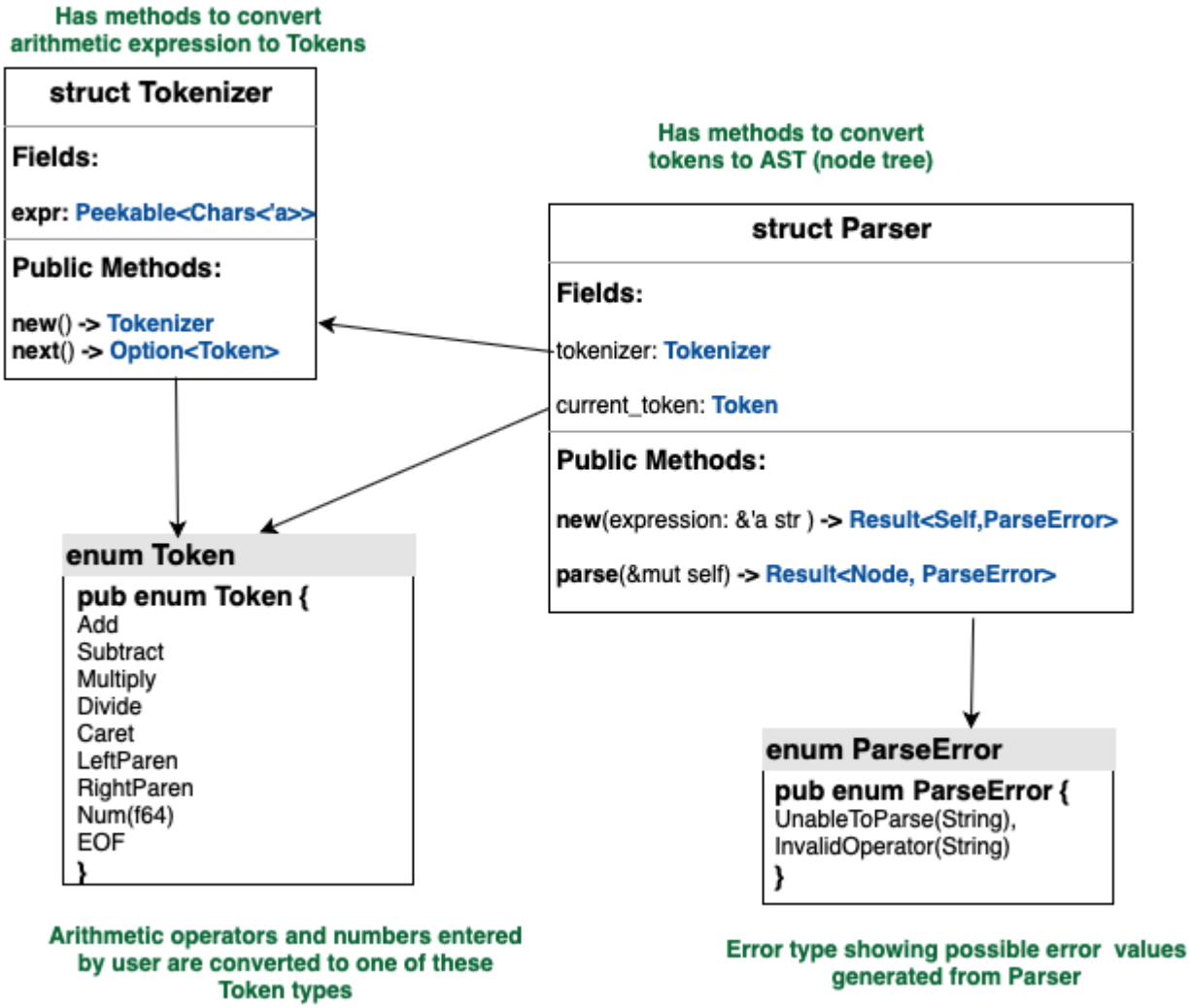
Types of AST nodes defined in enum

enum Node

```
pub enum Node {
  Add(Box<Node>, Box<Node>),
  Subtract(Box<Node>, Box<Node>)
  Multiply(Box<Node>, Box<Node>)
  Divide(Box<Node>, Box<Node>)
  Caret(Box<Node>, Box<Node>)
  Negative(Box<Node>)
  Number(f64)
}
```



Parser design



Parser with public and private methods

struct Parser

Fields:

tokenizer: **Tokenizer**

current_token: **Token**

Public Methods:

```
// Creates a new instance of Parser
new(expression: &str ) -> Result<Self,ParseError>

// Parses the tokens returned by Tokenizer, and computes AST
parse(&mut self) -> Result<Node, ParseError>
```

Private Methods:

```
// Main method that is called recursively
generate_ast(&mut self, oper_prec: OperPrec) -> Result<Node, ParseError>

// Retrieves number tokens
parse_number(&mut self) -> Result<Node, ParseError>

// Parses operators and converts to AST
convert_token_to_node(&mut self, left_expr: Node) -> Result<Node, ParseError>

// Checks for matching parenthesis in expression
check_paren(&mut self, expected: Token) -> Result<(), ParseError>

// Retrieves next Token from tokenizer and sets current_token field
get_next_token(&mut self) -> Result<(), ParseError>
```

Operator Precedence Enum

enum OperPrec

```
pub enum OperPrec {
  DefaultZero,
  AddSub,
  MulDiv,
  Power,
  Negative,
}
```

Error handling approach

Tokenizer

Option<Token>

Parser

Result<f64, ParseError>

ParseError

custom error type

AST

Result<64, Box<dyn error::Error>>

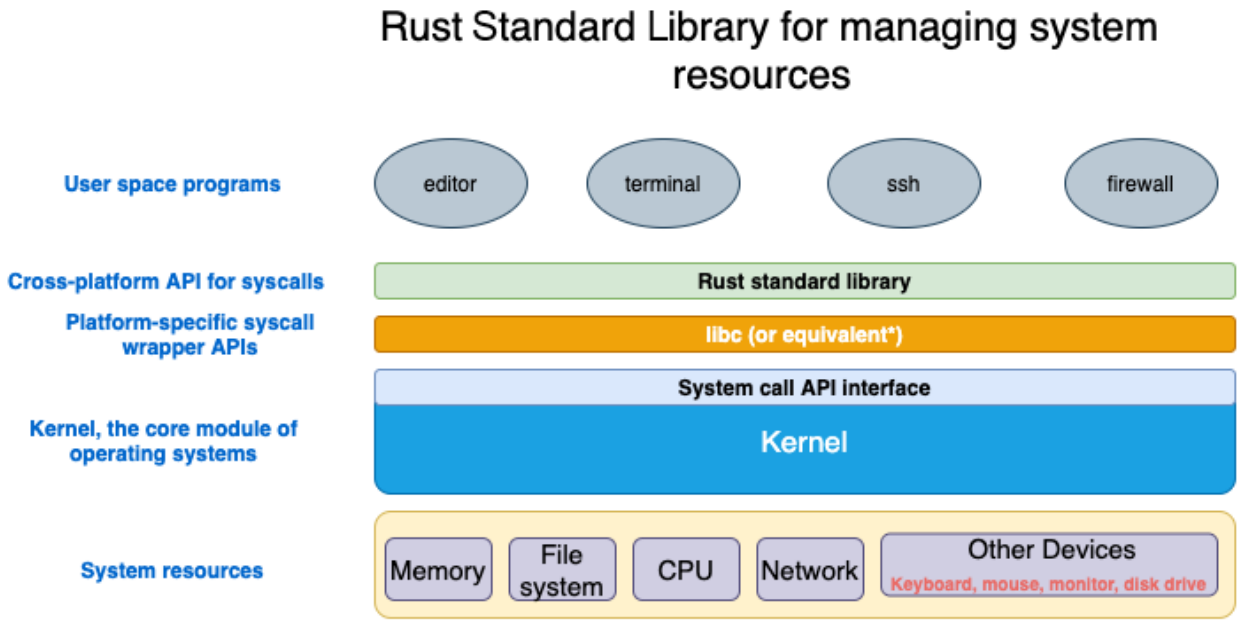
Token

No error returned

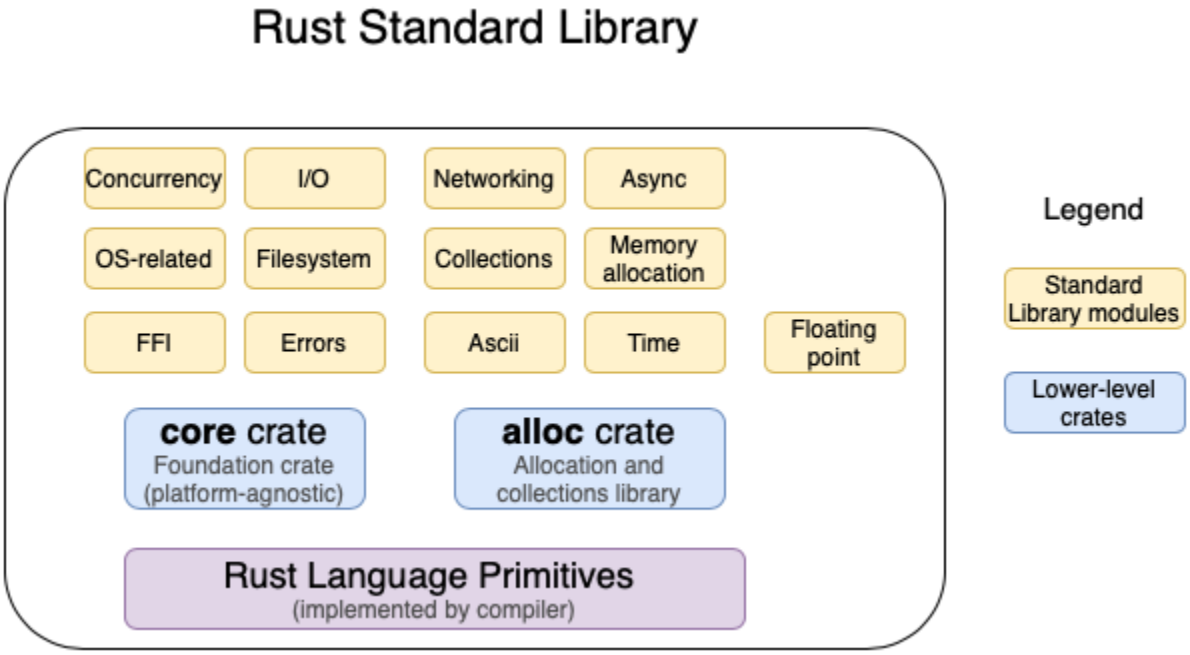
ParseError

```
pub enum ParseError {
  UnableToParse(String),
  InvalidOperator(String),
}
```

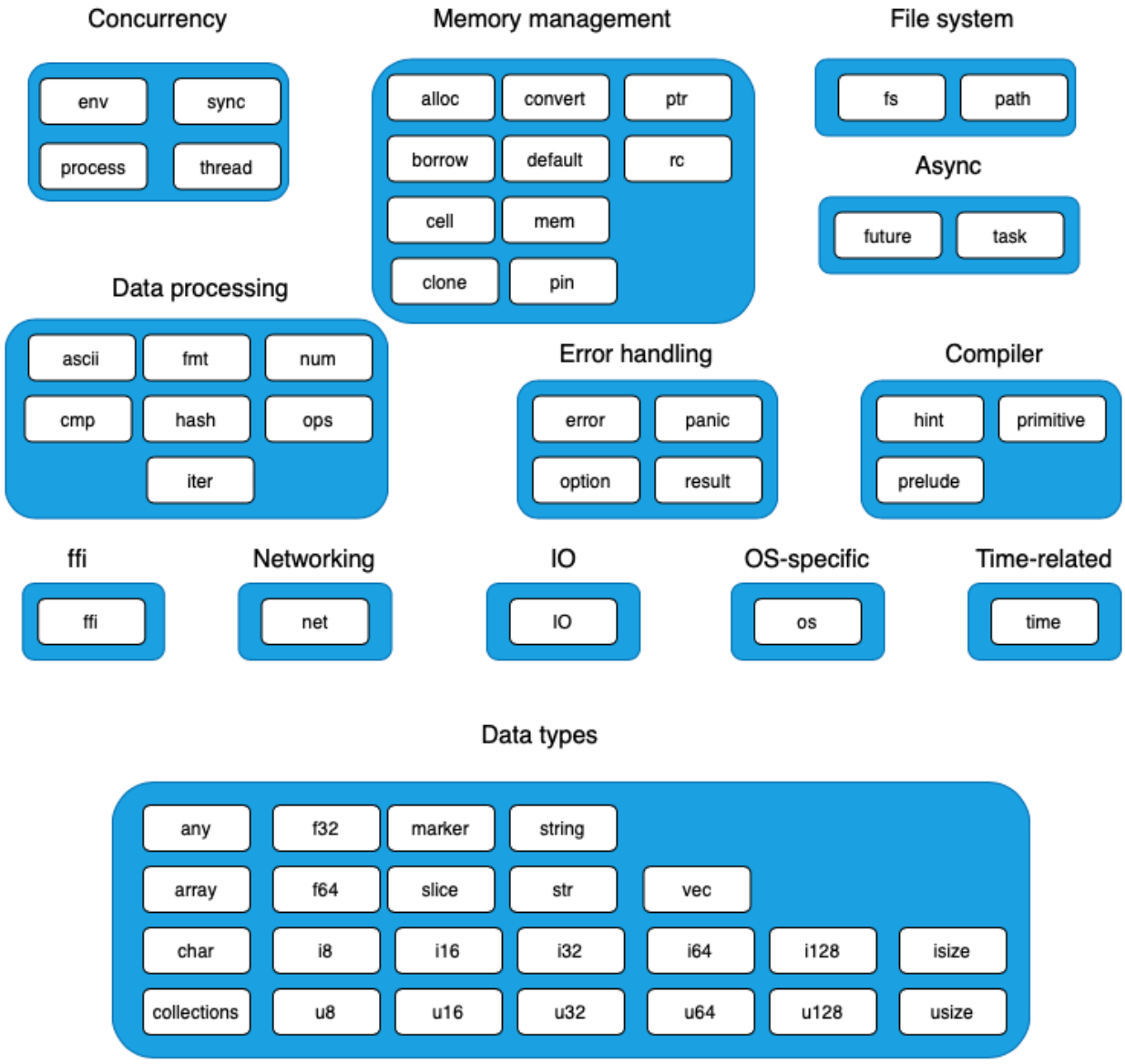
Chapter 3: Introduction to the Rust Standard Library



Note: **libc** (or one of its variants) provides a wrapper for system calls on Unix-like operating systems. For Windows, there are equivalent APIs for syscalls

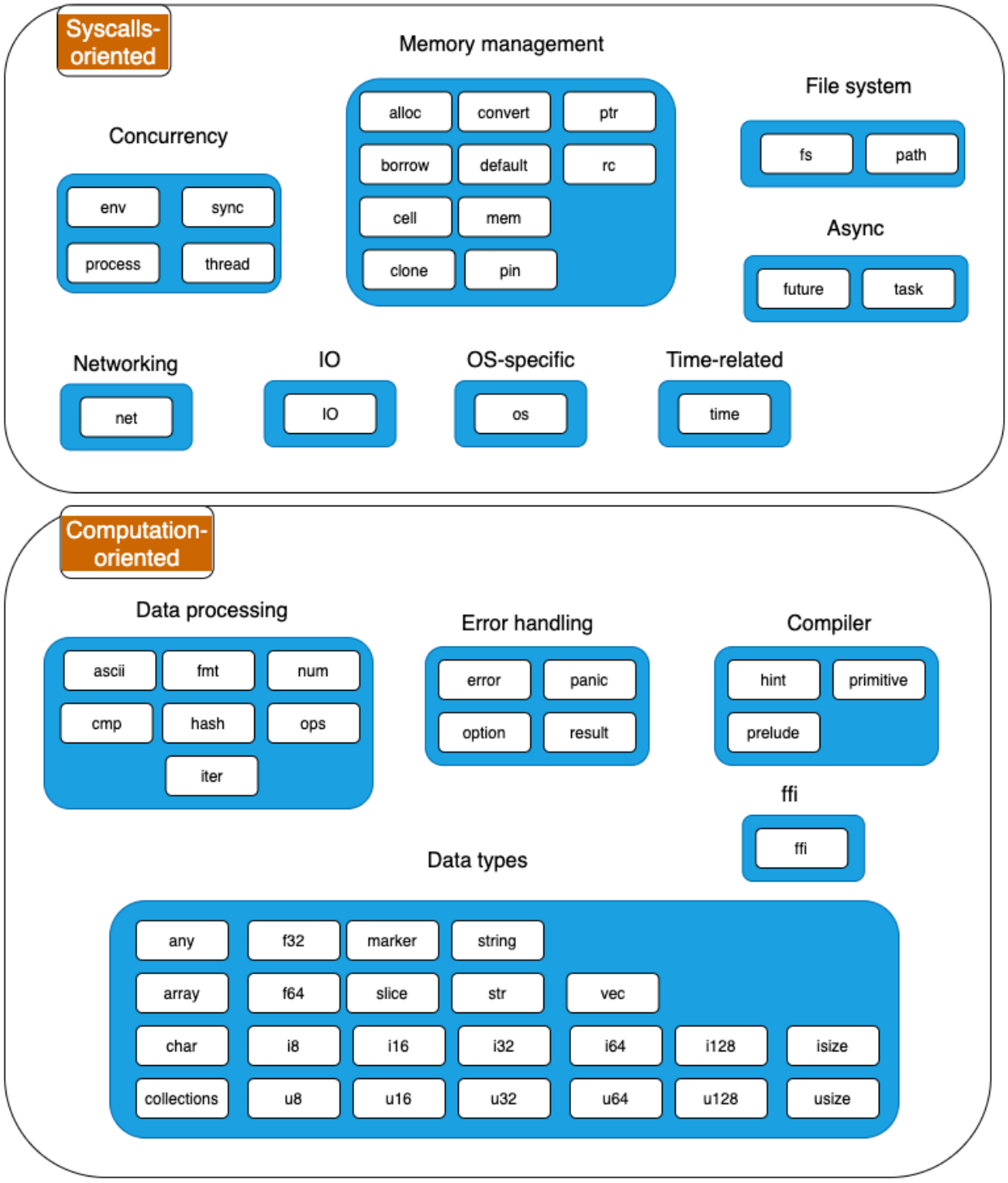


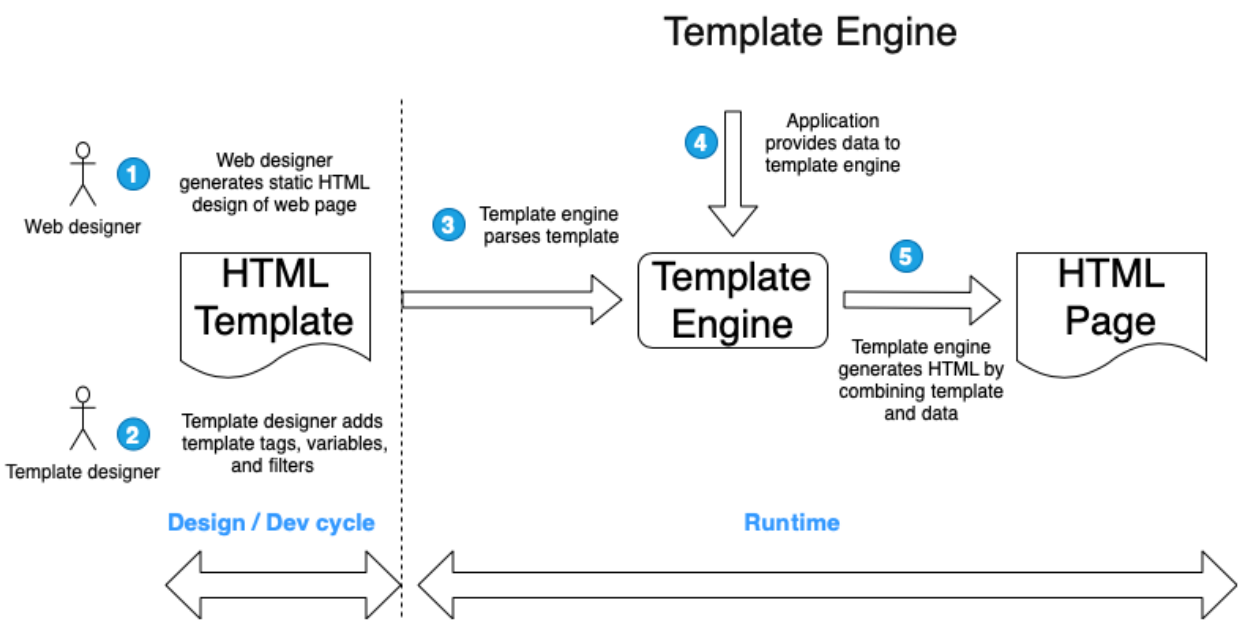
Rust Standard Library modules



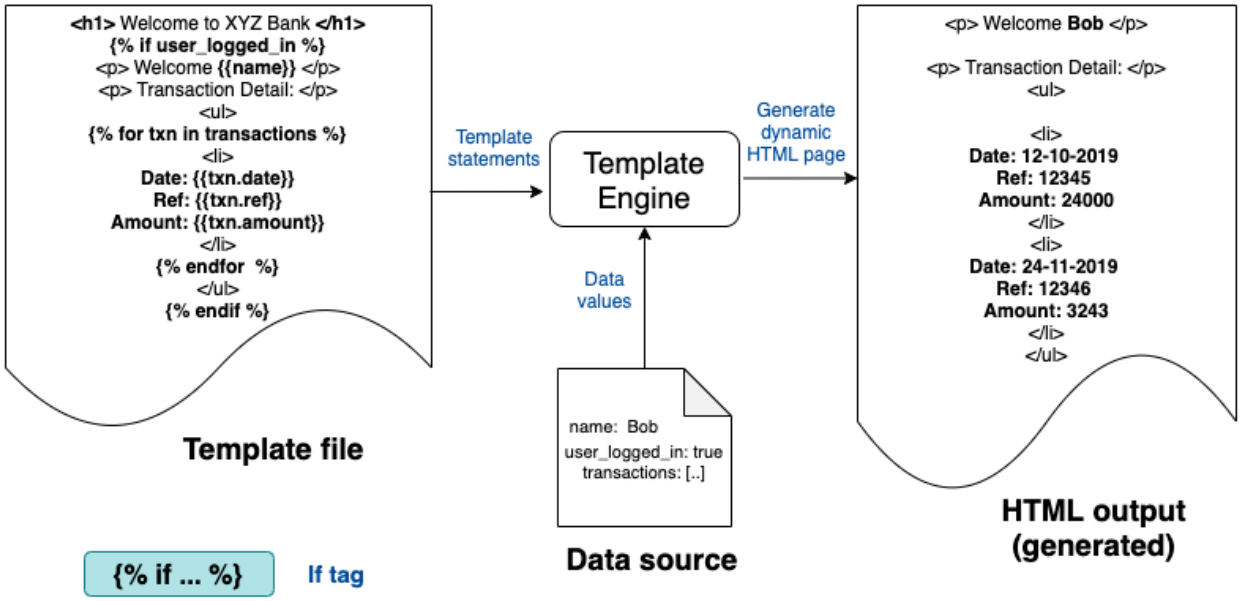
Data types

Rust standard library classification





HTML Generation from template

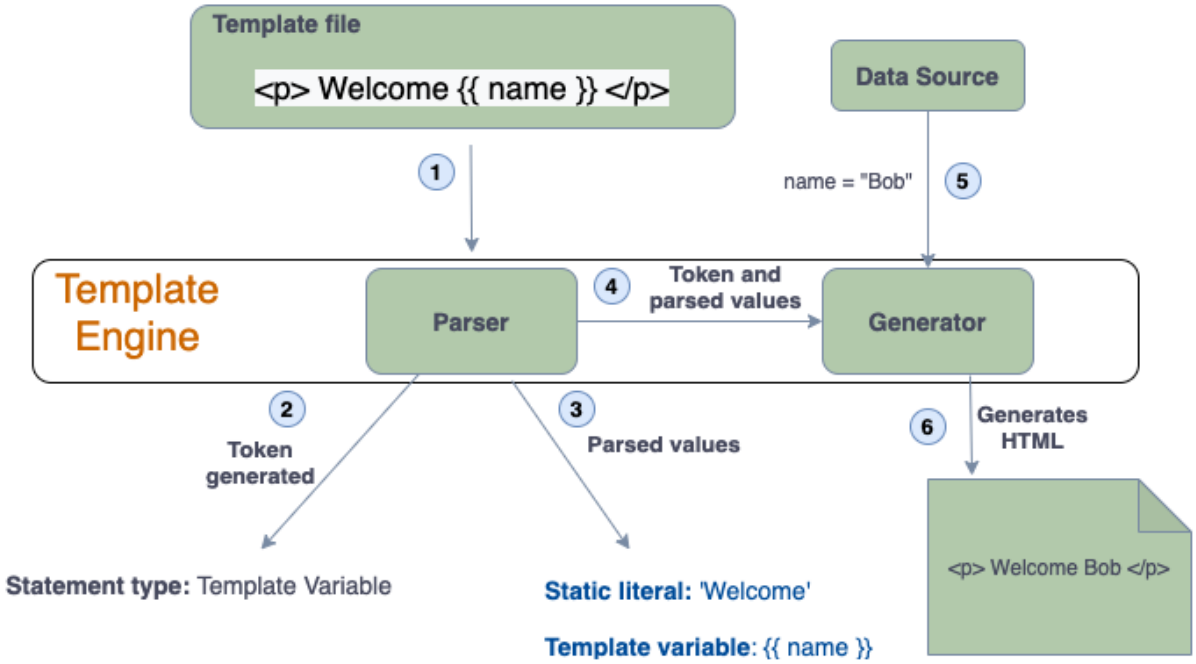


{% if ... %} If tag

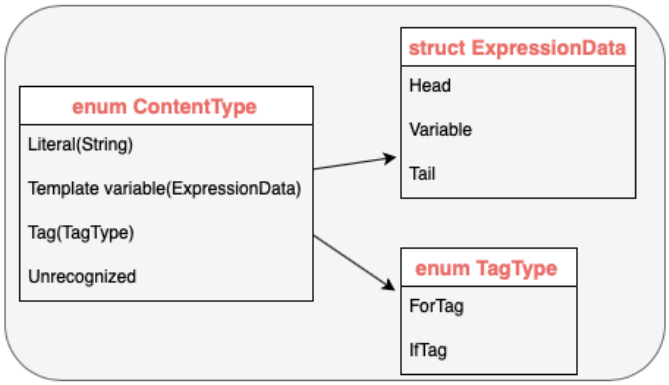
{% for ... %} for tag

{{name}} Template variable

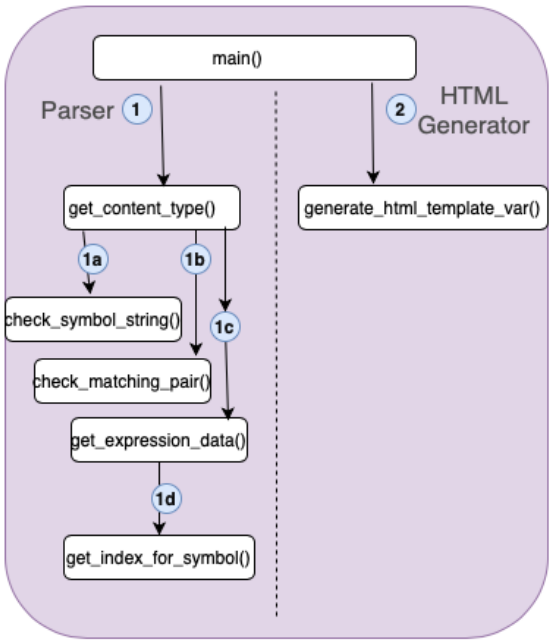
Illustrated working of template engine



Design of Template Engine

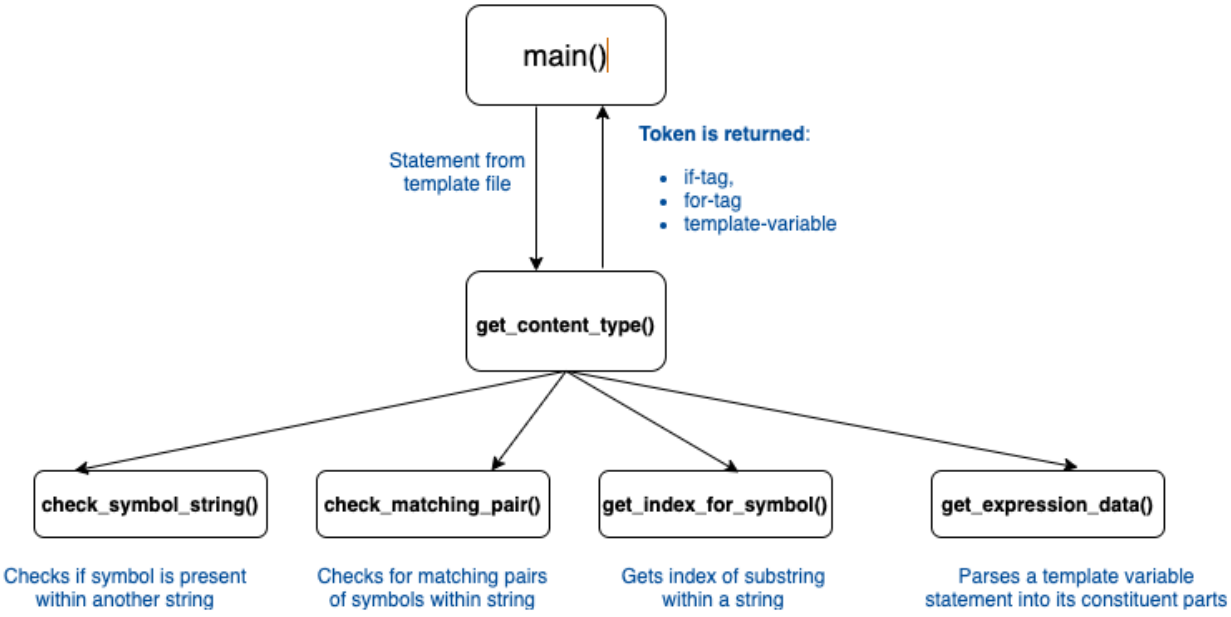


Data Structures

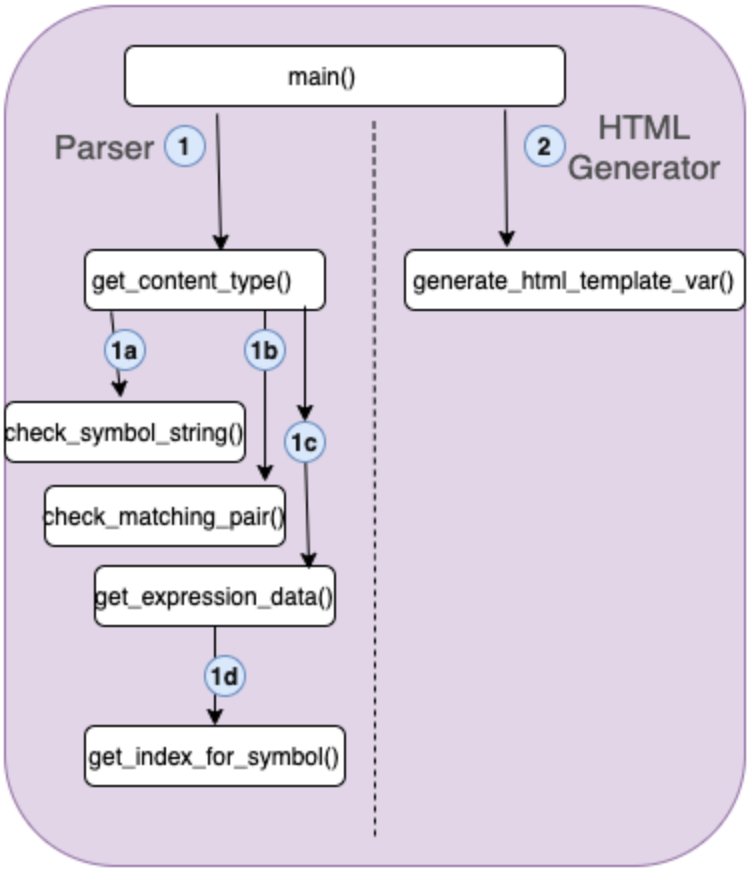


Program structure

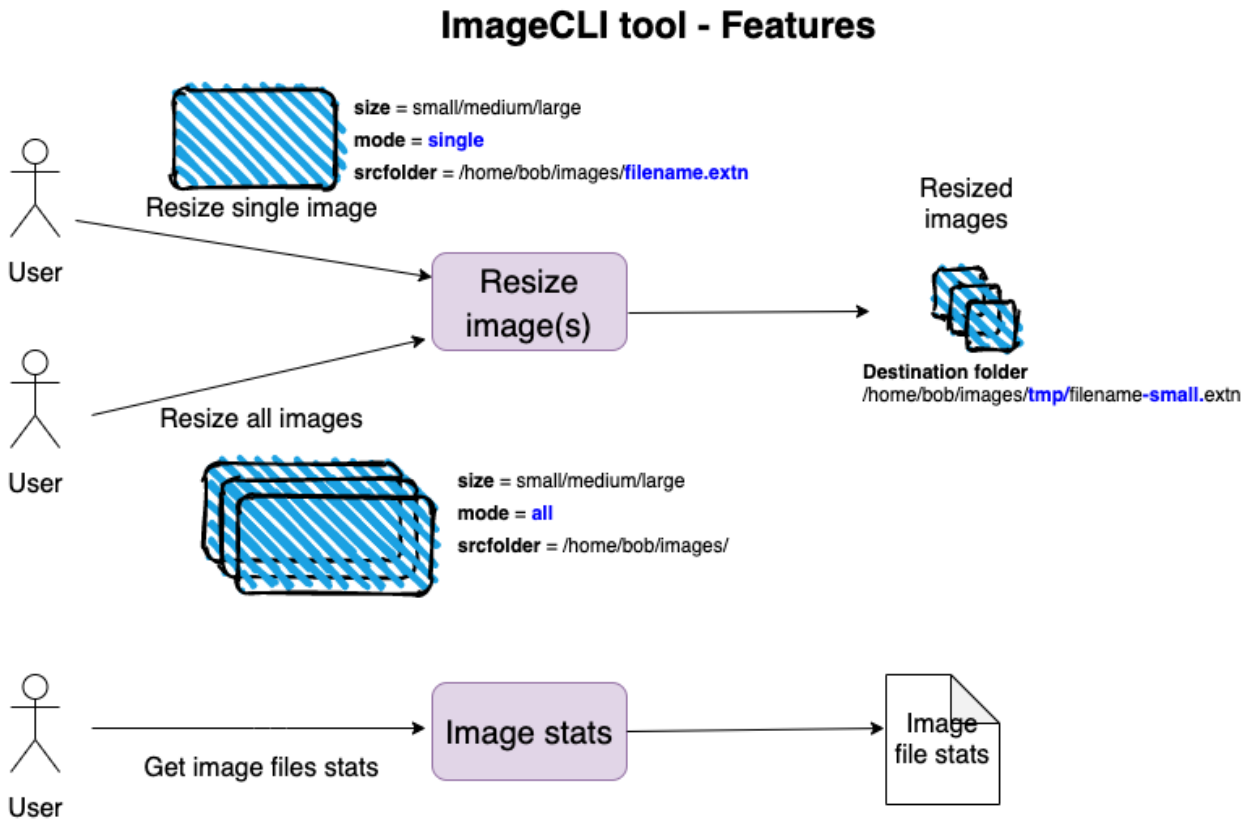
Design of Parser

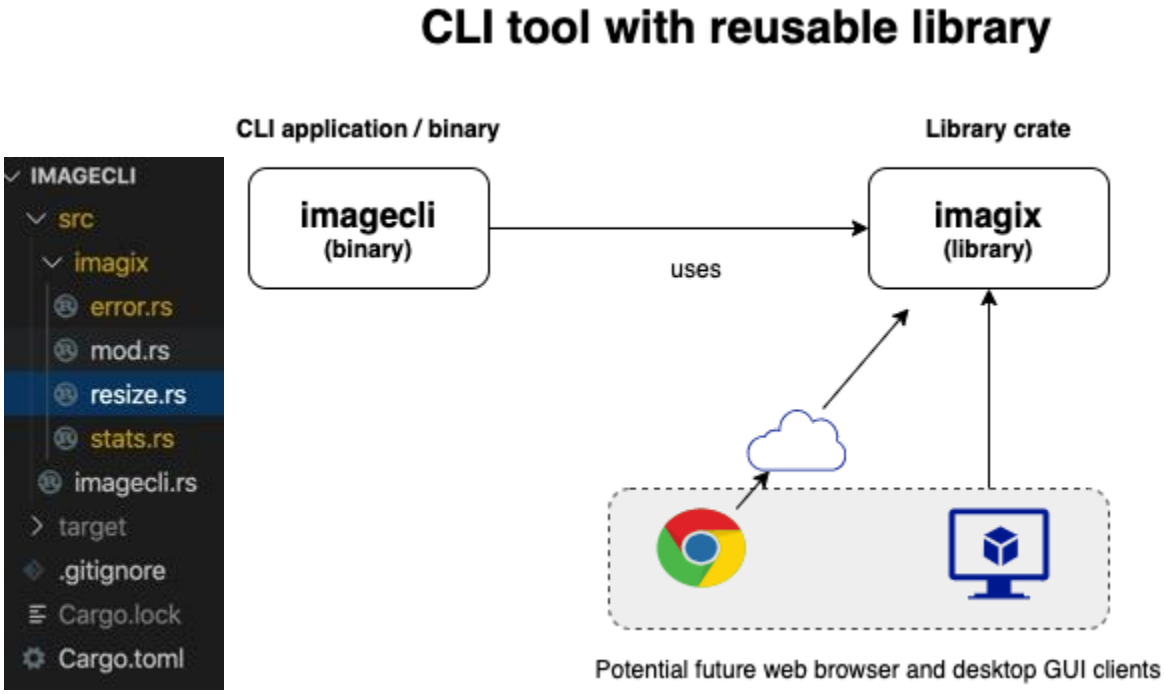


main() program structure

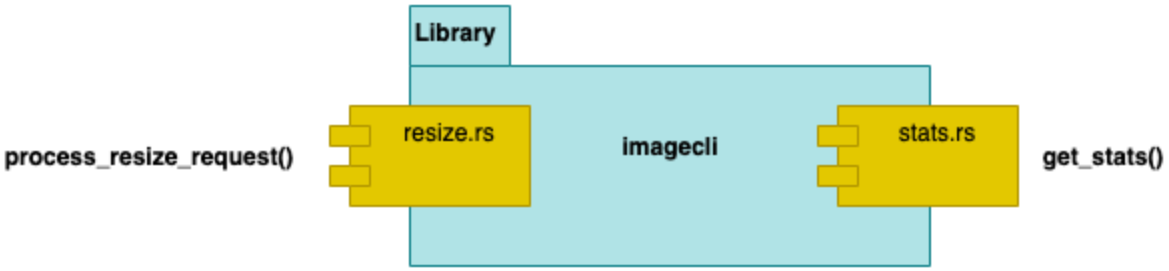


Chapter 4: Managing Environment, Command Line, and Time





Design of imagix library

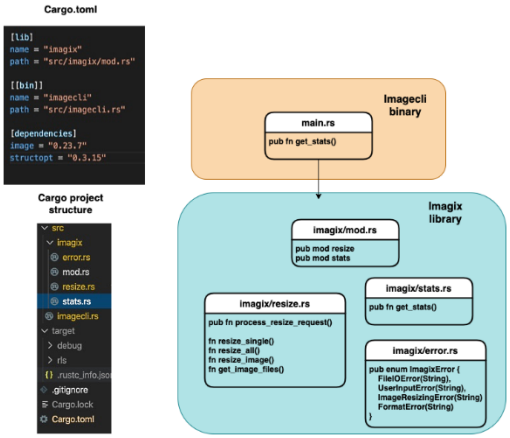


enum SizeOption
Small
Medium
Large

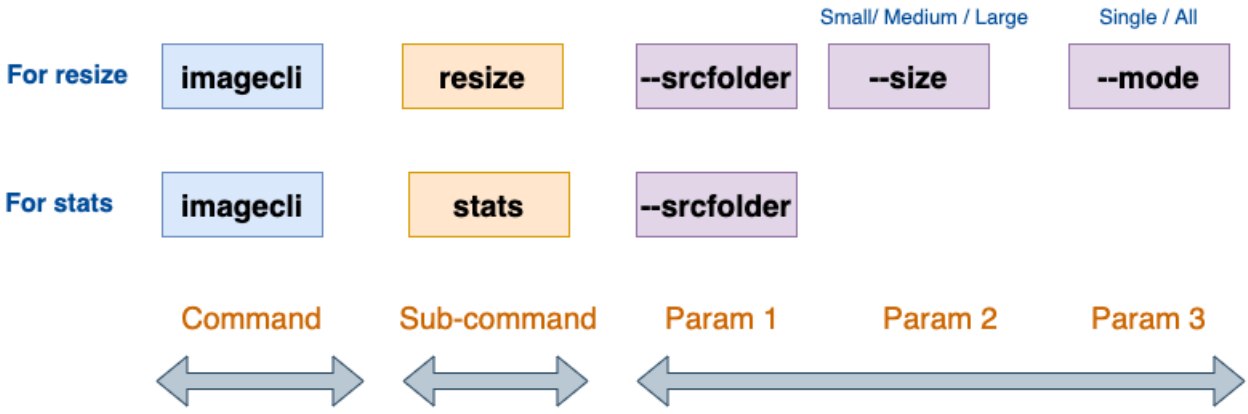
enum Mode
Single
All

struct Elapsed
Duration

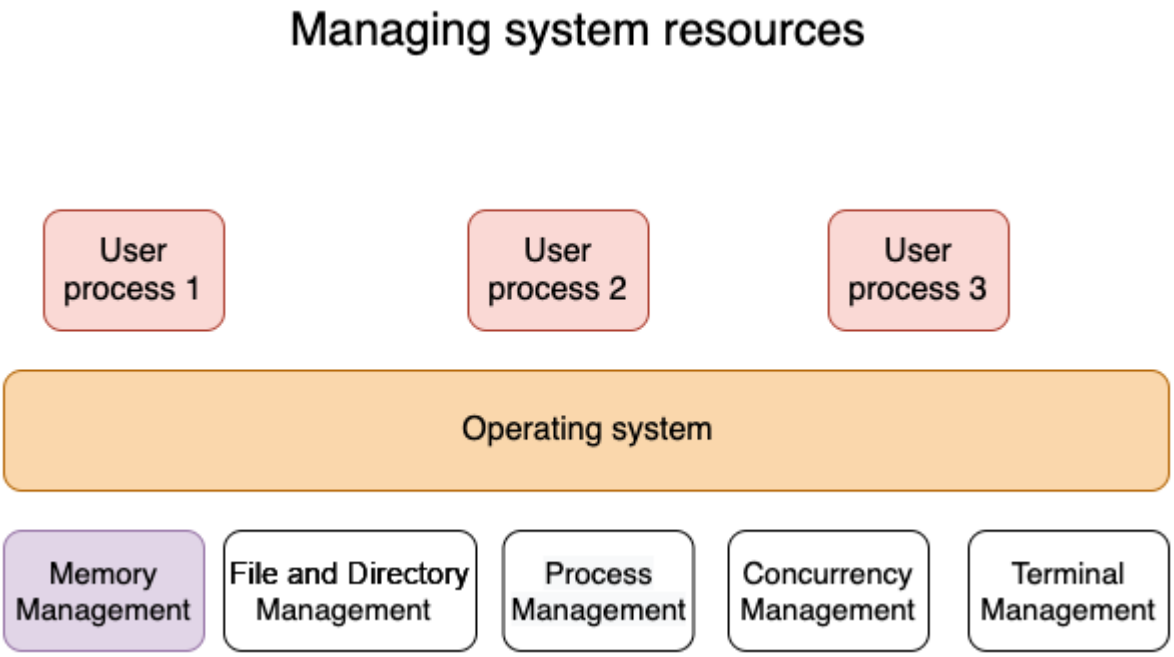
Code organisation for Imagecli project



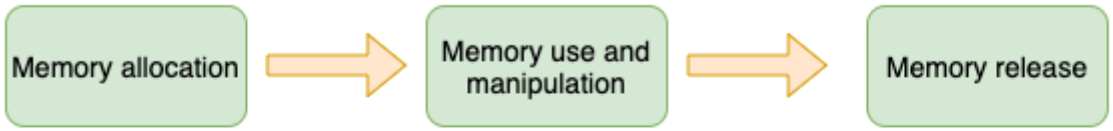
Design of CLI commands



Chapter 5: Memory Management in Rust



Memory management lifecycle



- Static memory
- Dynamic memory

- Initialize
- Assign
- Modify
- Copy
- Move
- Create a reference
- Pass a reference

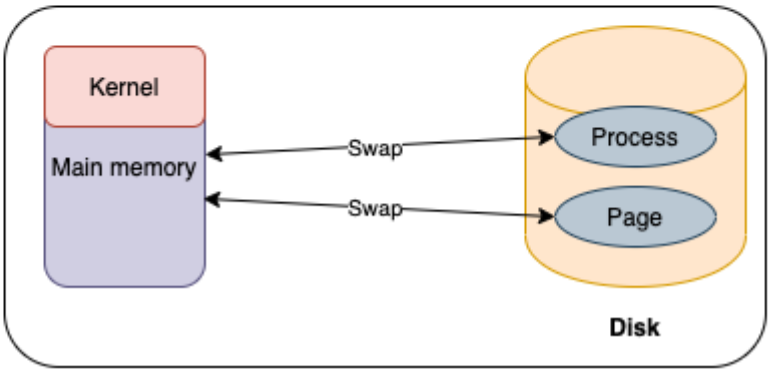
- Programmer-controlled
- Garbage-collected
- Compiler-generated

By Kernel

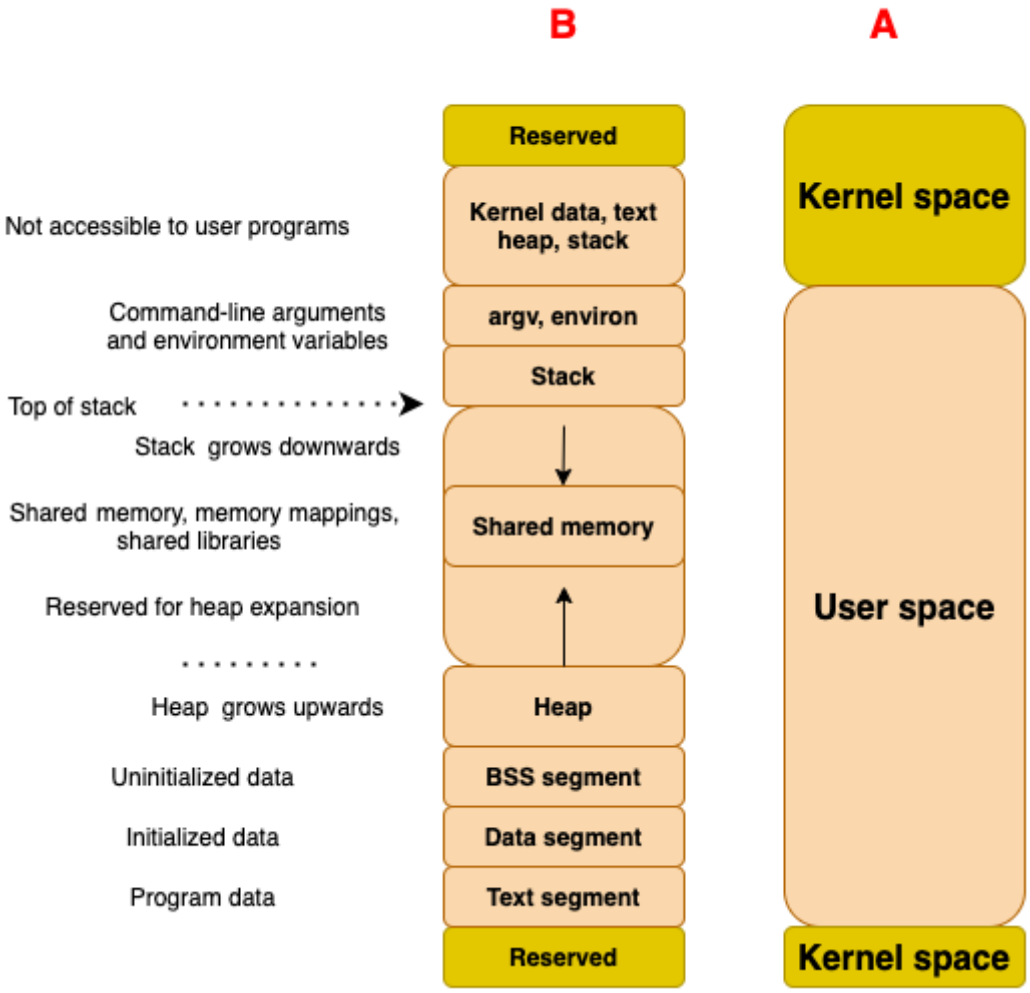
Track memory usage

- Is a memory location allocated to a process or free?
- How much memory is allocated to which process?
- How much memory is released by each process?
- When should memory be allocated to a process?

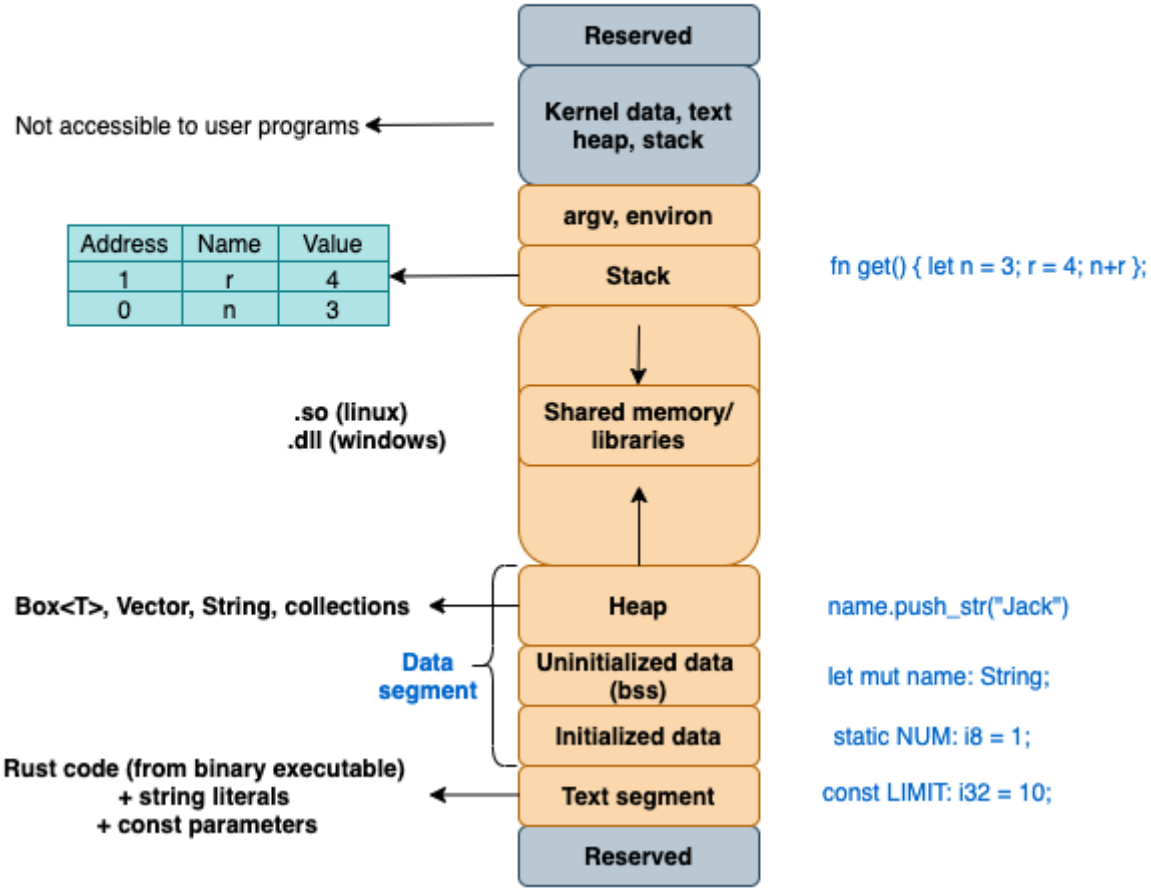
Swapping / Paging



Process memory layout



Rust program memory layout

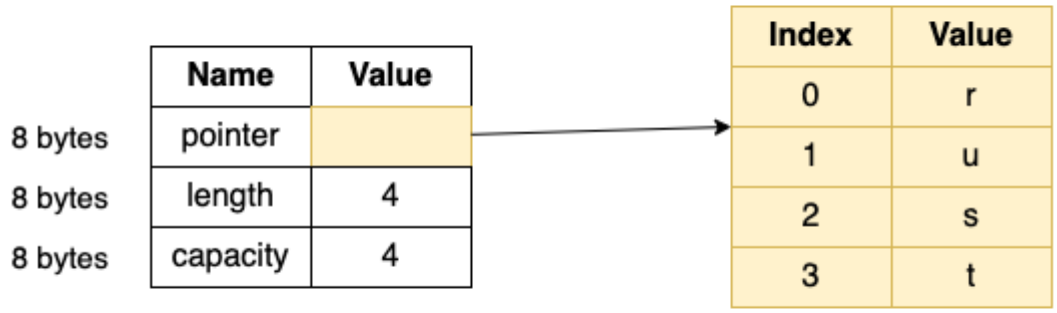


Memory management in various programming languages

	Memory allocation	Memory use and manipulation	Memory release
High-level languages (Java, Python, JavaScript Ruby)	Programmer only initiates, but language determines memory layout	Programmer has some control but no access to pointers	**Automated** using garbage-collector
Low-level (system) languages (C / C++)	Programmer determines memory layout	Programmer has **unrestricted** control	Programmer-controlled **(Manual)** Smart-pointers (C++)
Rust	Programmer determines memory layout	Programmer has **restricted** control and needs to adhere to Rust ownership & lifetime model. For **unrestricted** access, unsafe Rust can be used	**Automated** using Rust **Drop** trait. Governed by Rust ownership and lifetime model and scoping rules

String smart pointer in Rust

(example for 64-bit arch)

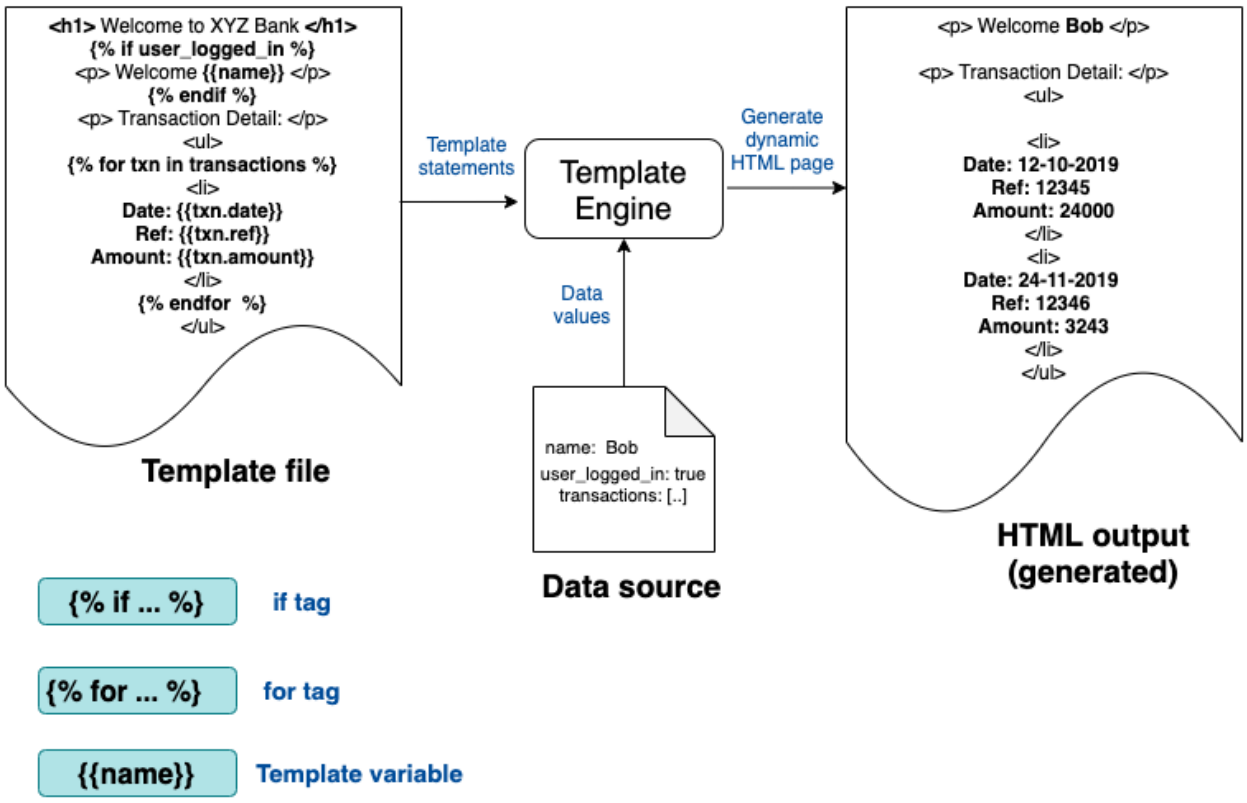


String smart pointer stored on **stack** memory

Actual value stored on the **heap** memory

In 64-bit computer architectures,
machine word size = 64 bits (8 bytes)

HTML Generation from Template



{% if ... %} if tag

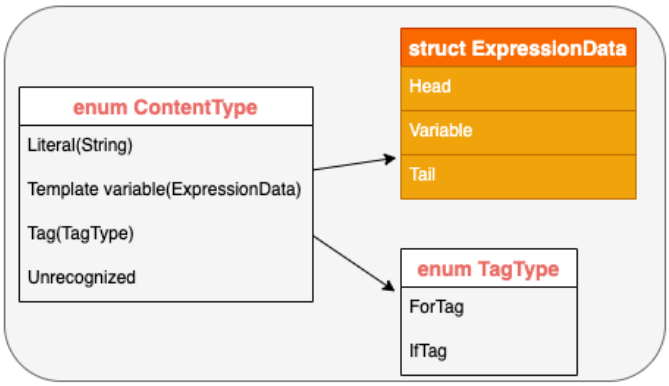
{% for ... %} for tag

{{name}} Template variable

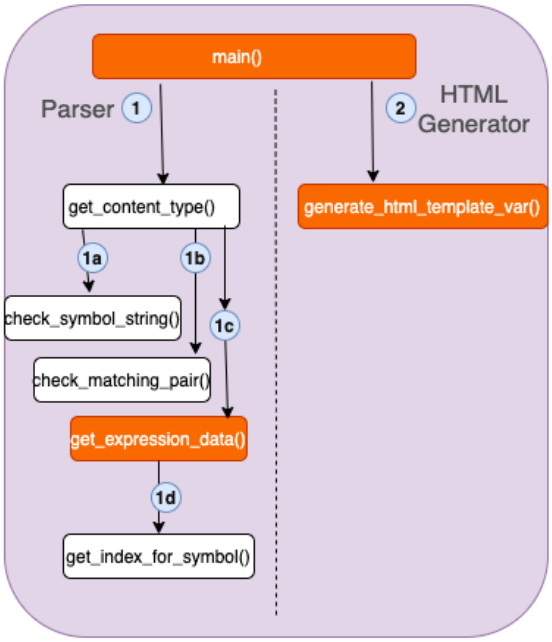
Changes to design of template engine

Legend

Components to be modified

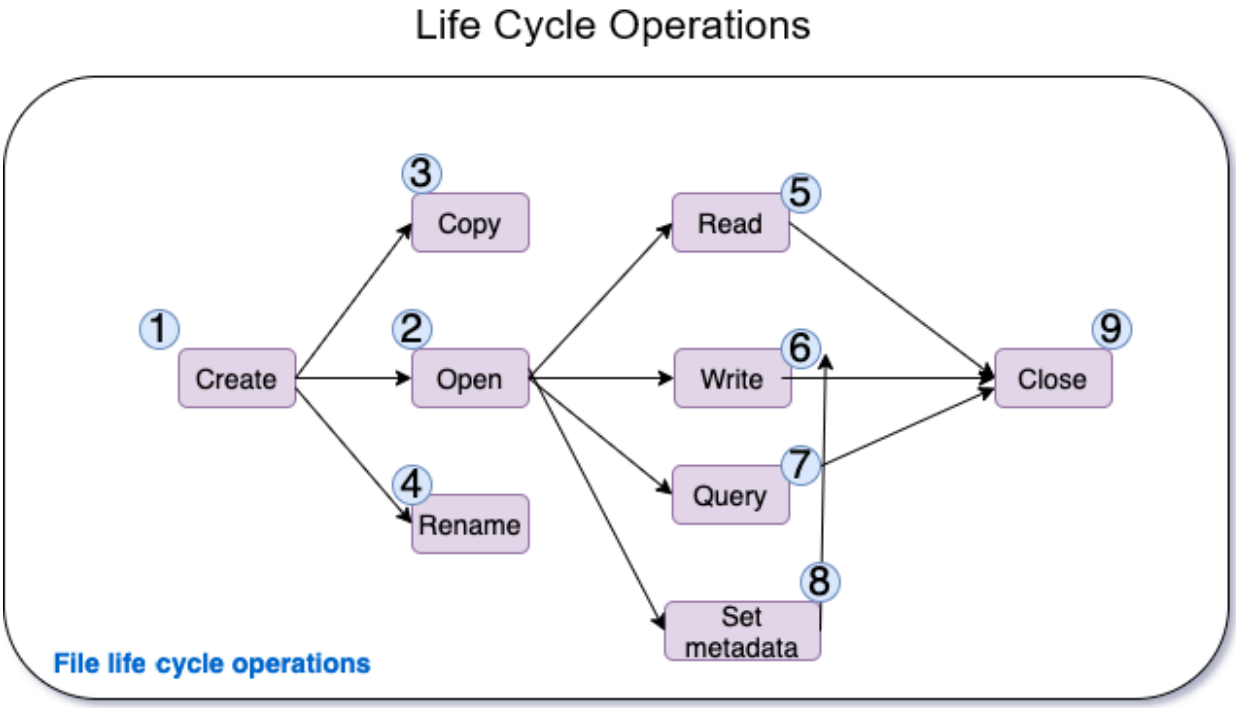


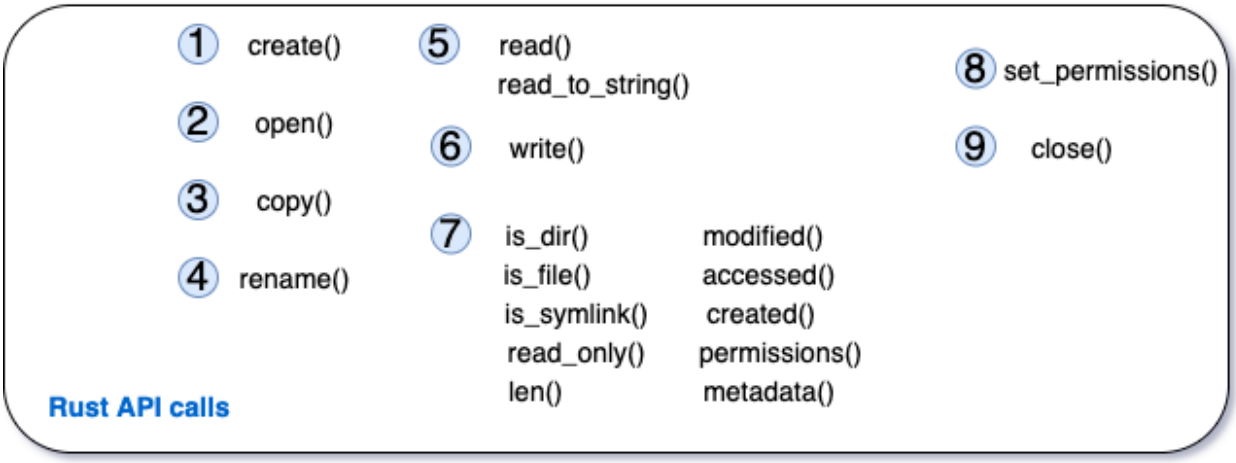
Data Structures



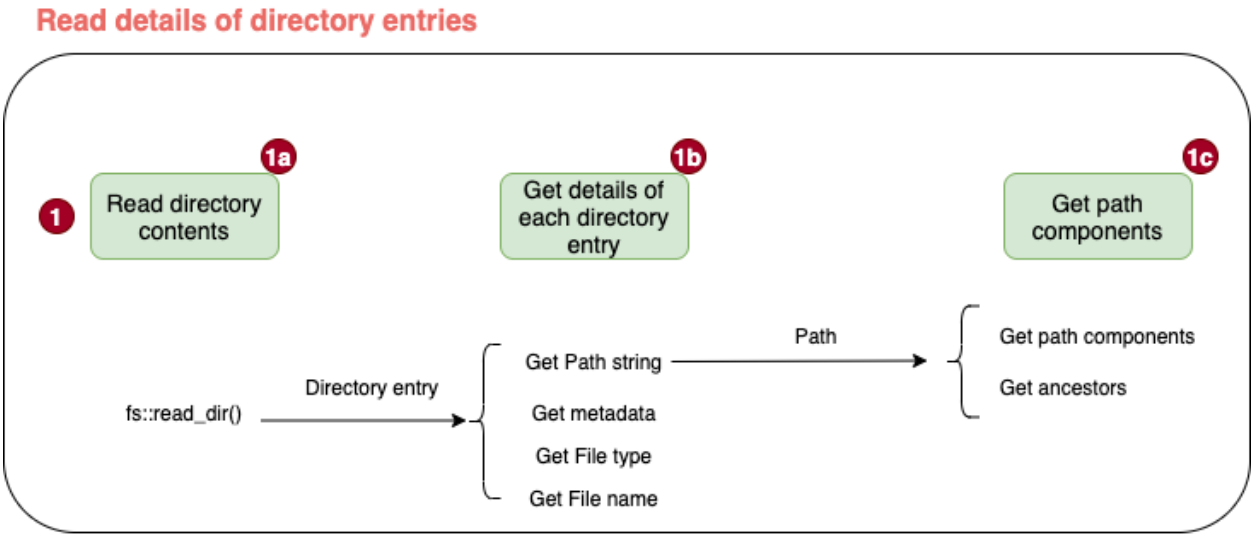
Program Structure

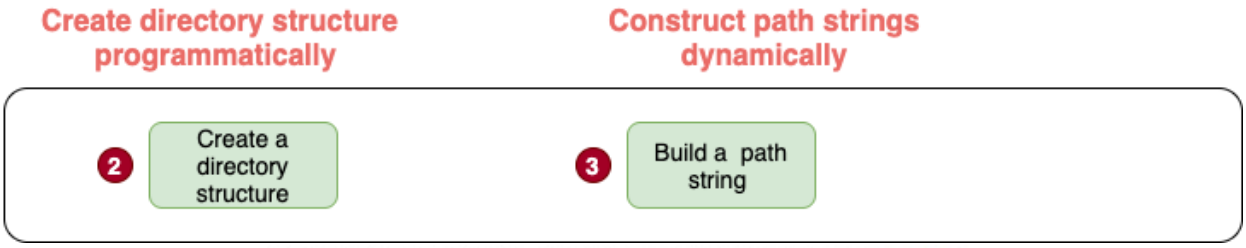
Chapter 6: Working with Files and Directories in Rust





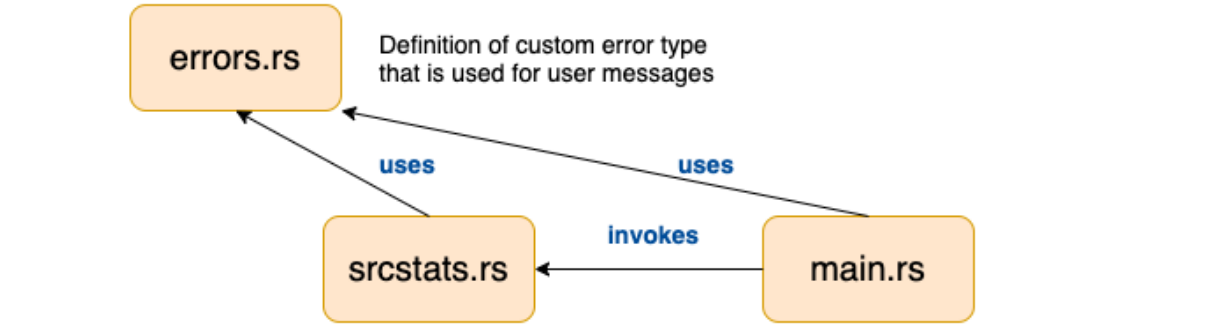
Common directory and path operations






```
256029 drwxrwxr-x. 3 ap ap 4096 Aug 20 12:53 rust
```

Shell command (rstat) design



struct SrcStats

Code metrics definition

Command-line parameter definition

struct Opt

function get_src_stats_for_file()

Calculate source metrics for single file

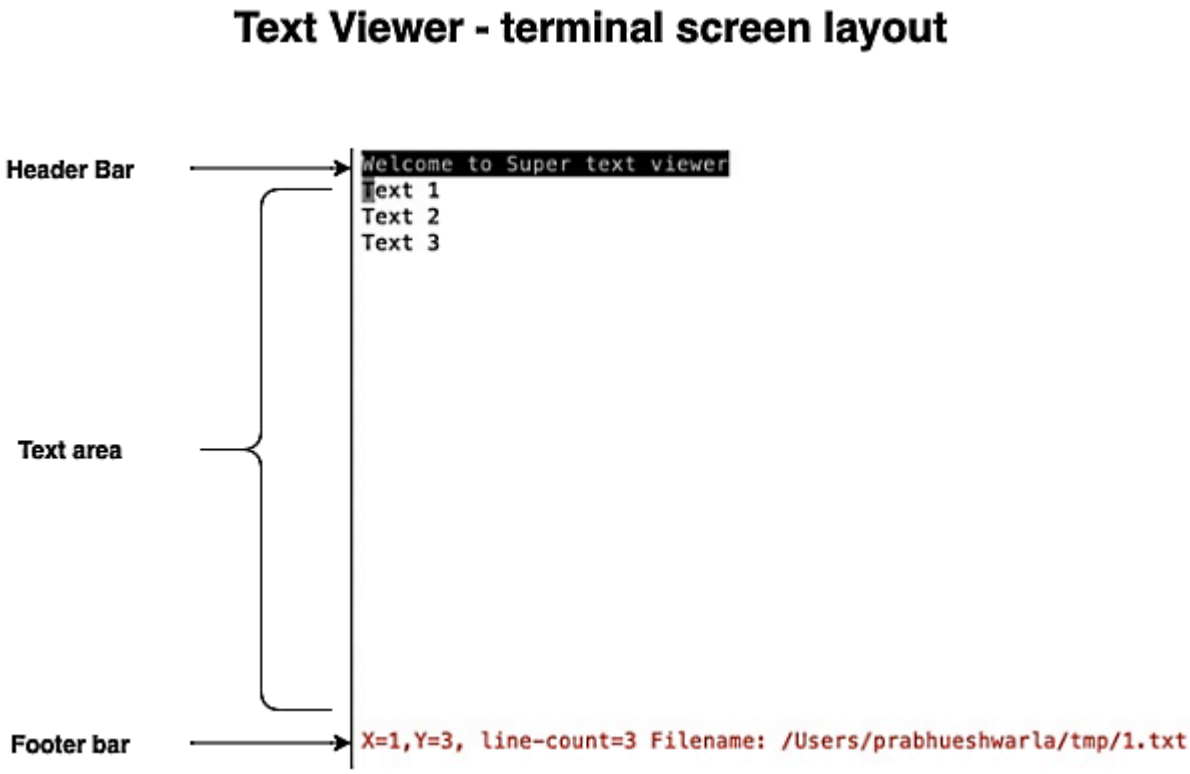
function get_summary_src_stats()

Calculate source metrics for all files in a directory root

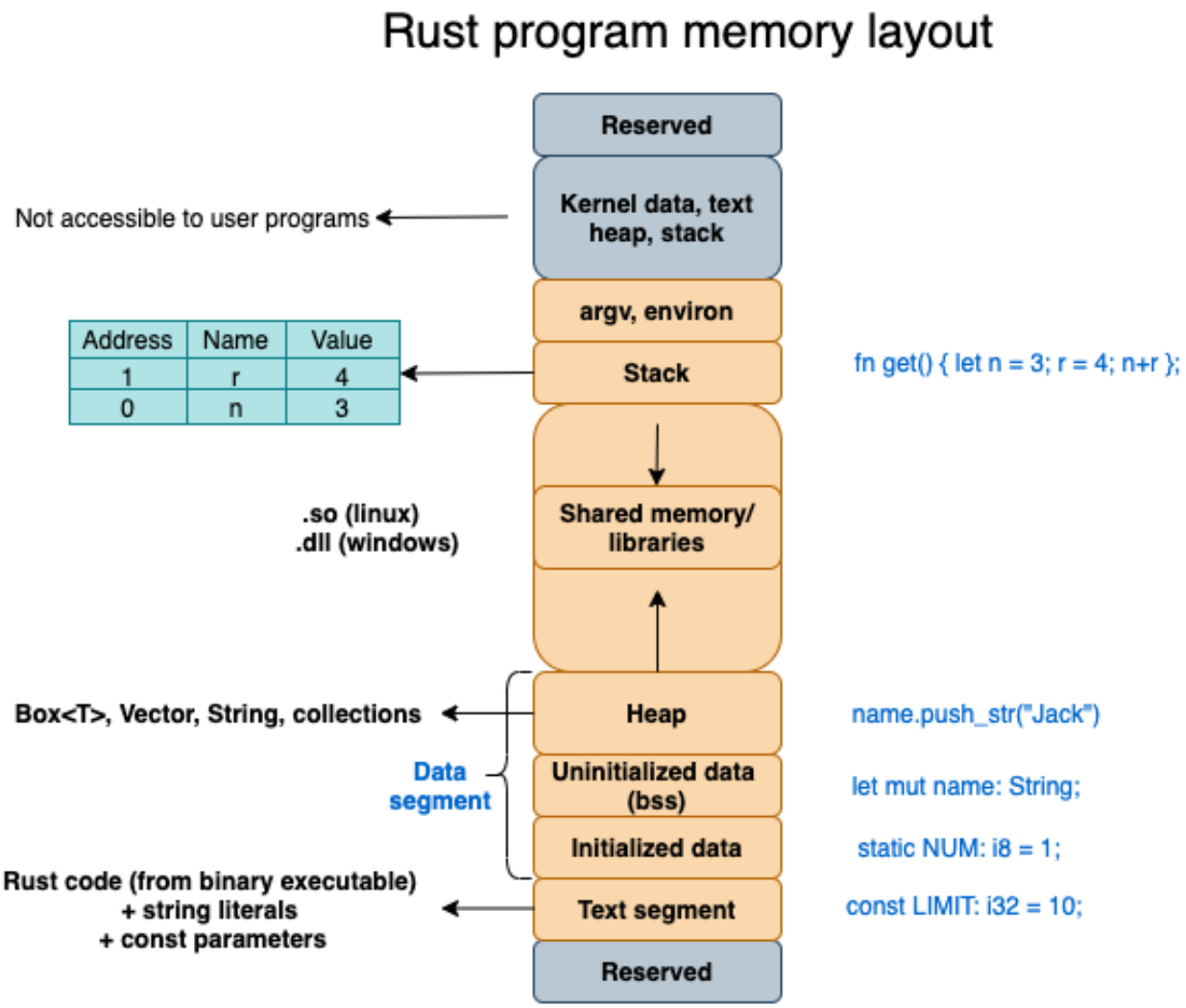
Program entry point. Accepts command-line inputs and invokes functions in the **srcstats** module

function main()

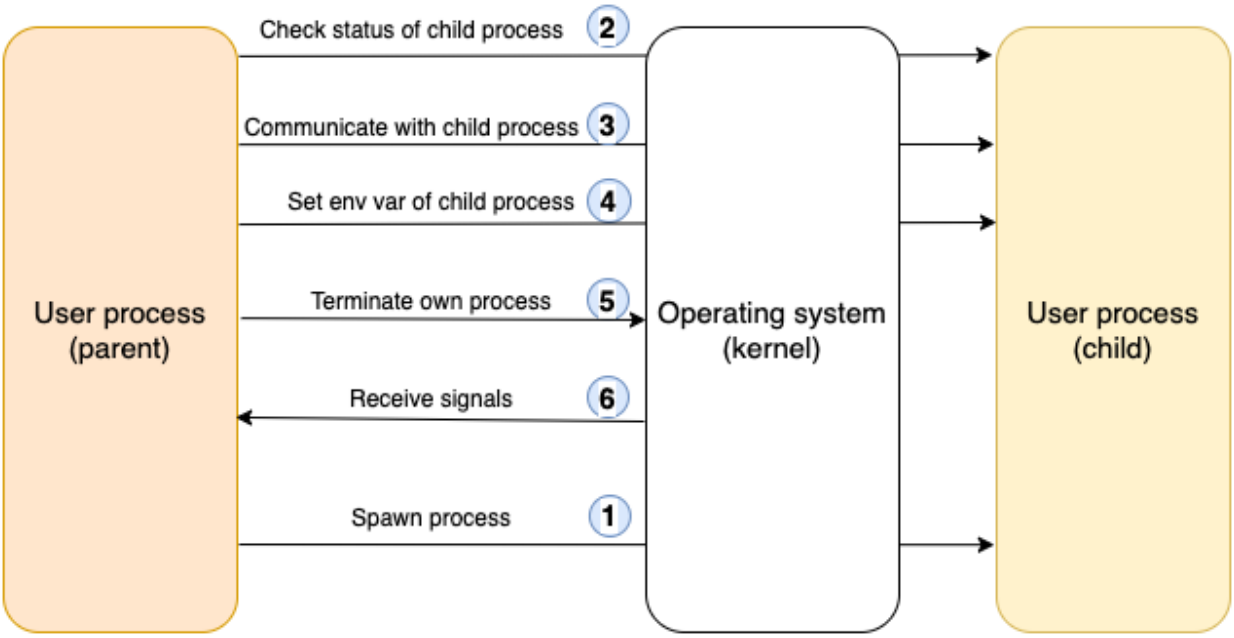
Chapter 7: Implementing Terminal I/O in Rust



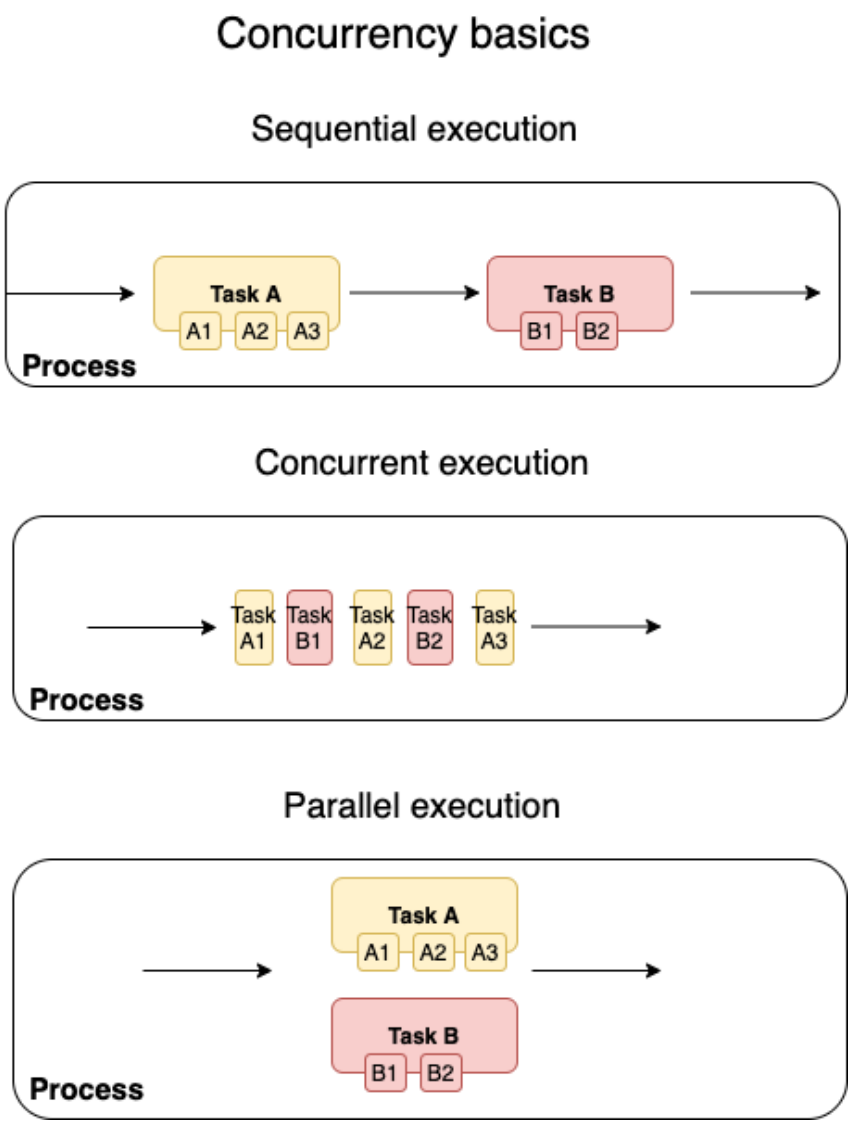
Chapter 8: Working with Processes and Signals



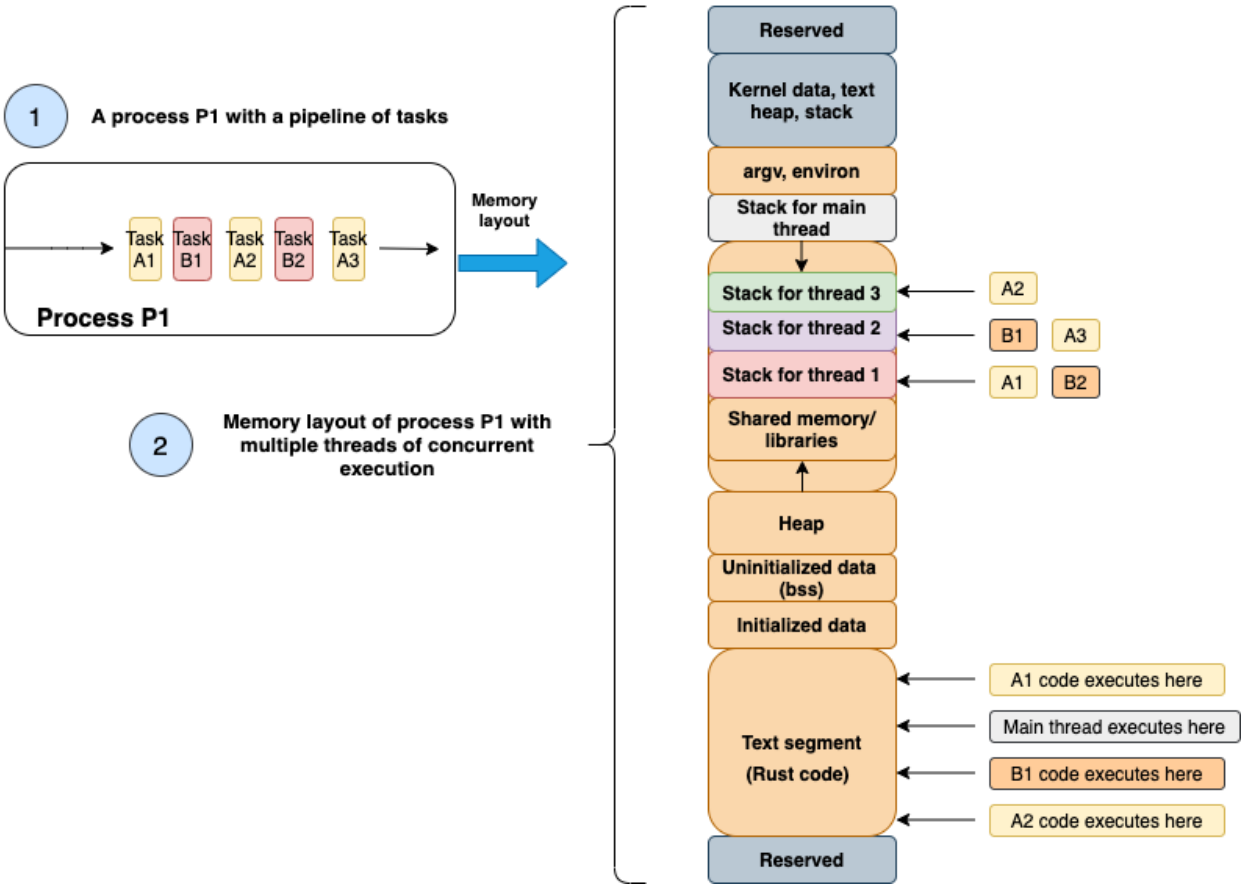
Working with processes in Linux



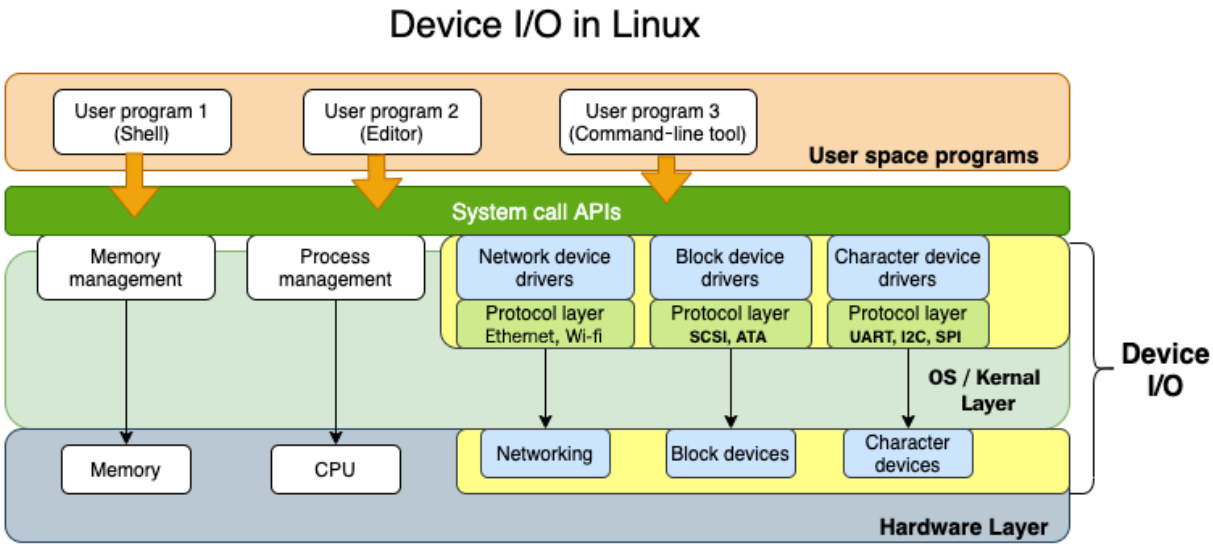
Chapter 9: Managing Concurrency



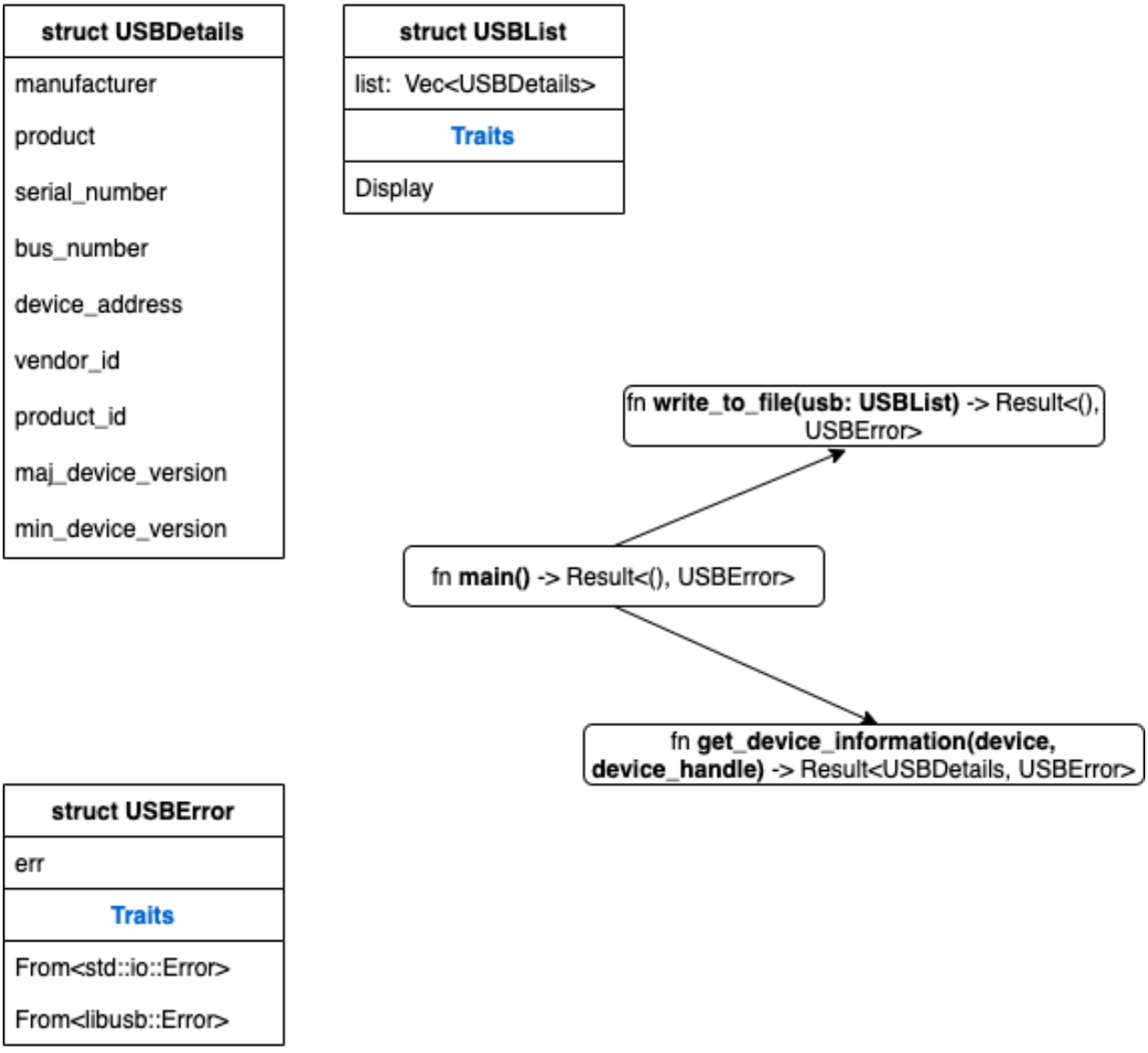
Process memory layout with threads



Chapter 10: Working with Device I/O

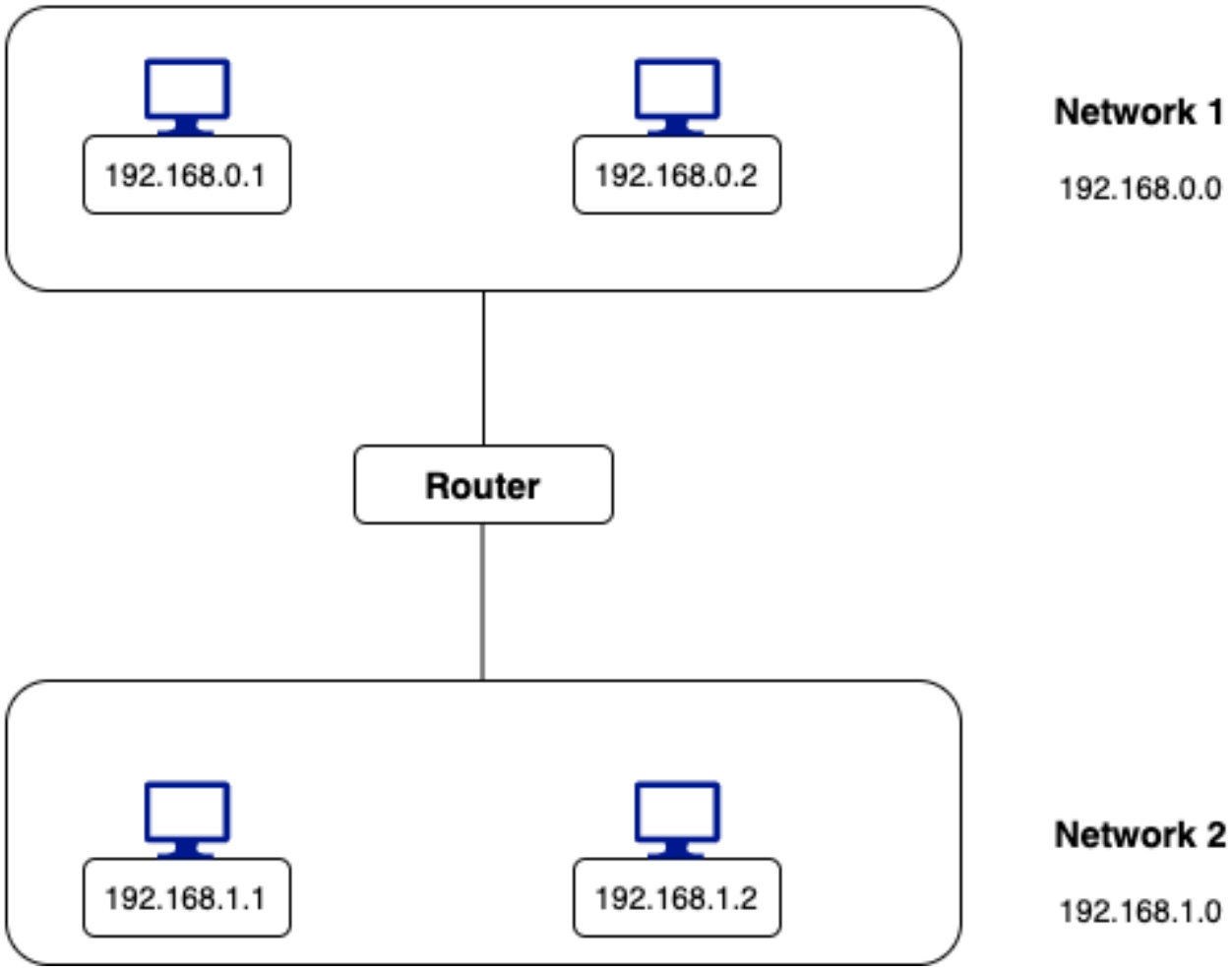


Design of USB detector project

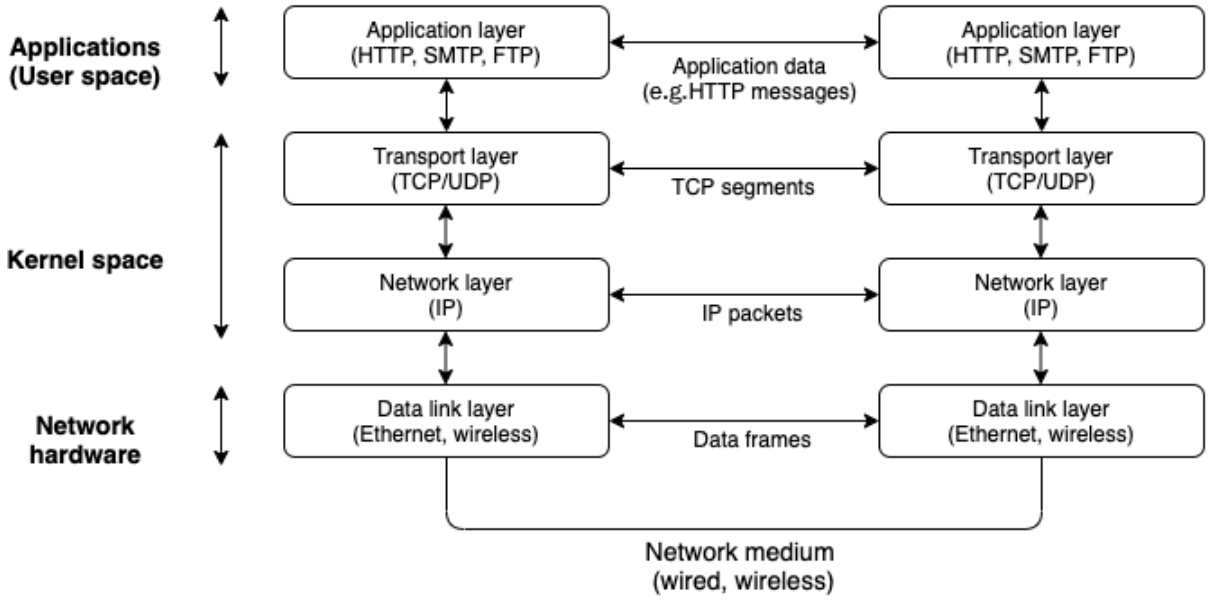


Chapter 11: Learning Network Programming

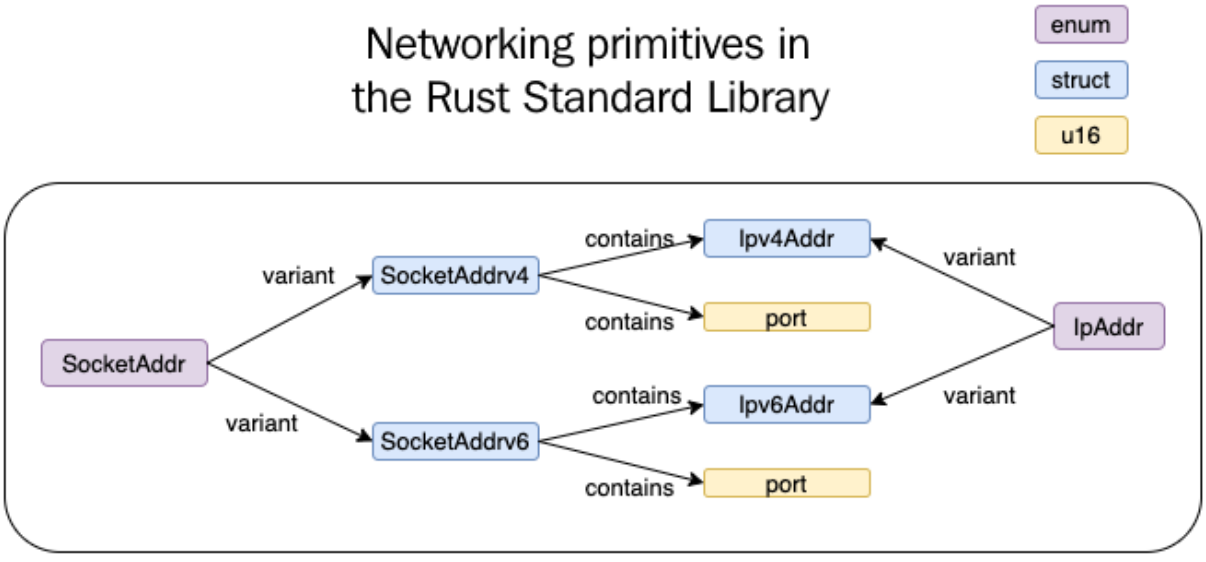
Internet router connecting two networks



Network communications with TCP/IP stack

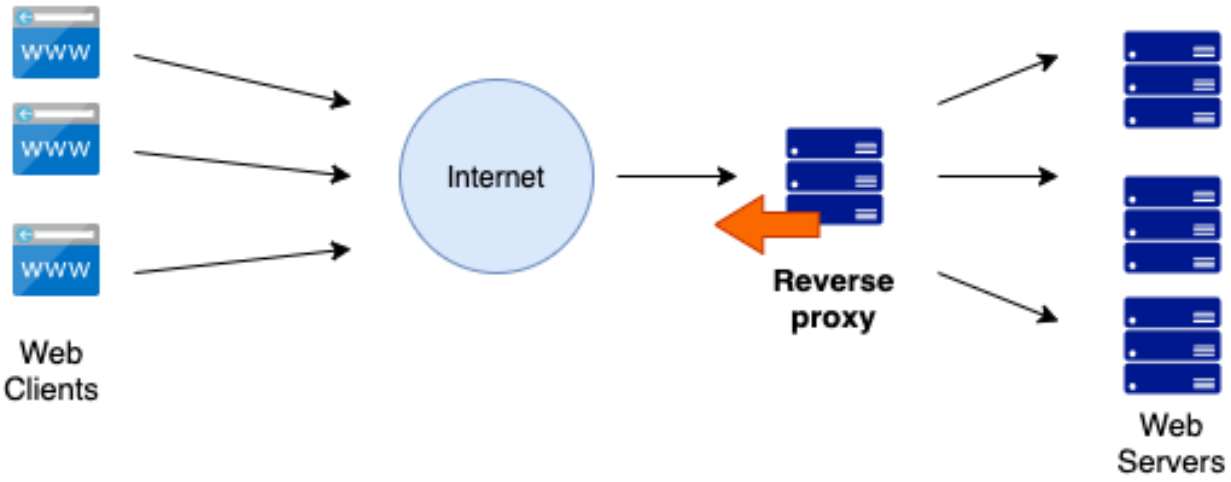


Networking primitives in the Rust Standard Library

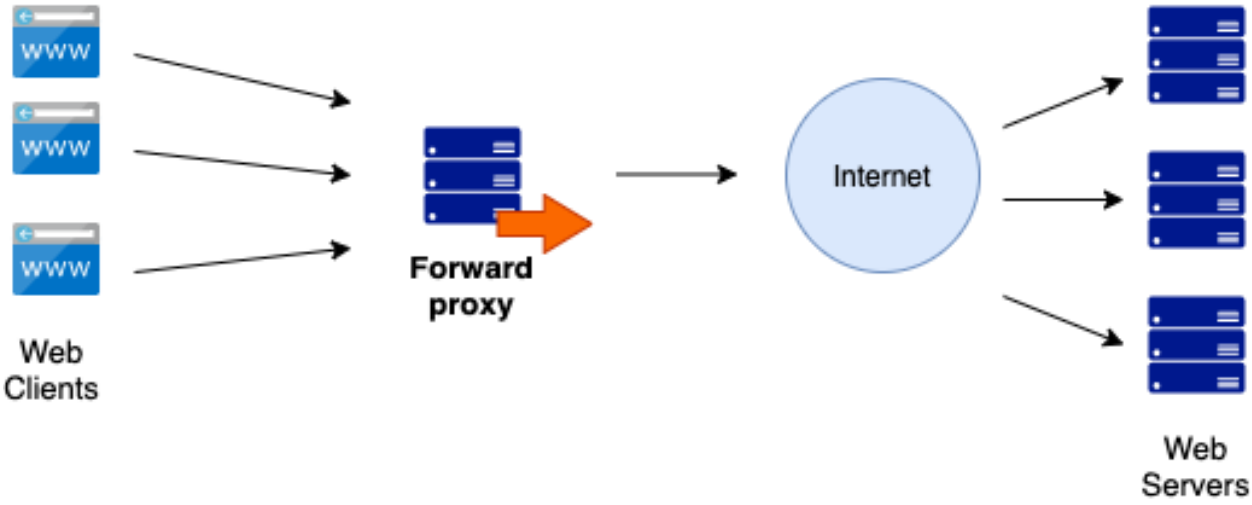


Types of proxy servers

Reverse proxy setup



Forward proxy setup



Chapter 12: Writing Unsafe Rust and FFI

No images