

Defect  
Detect



# Rust

## Memory Thinking

### Version 2

Dmitry Vostokov  
Software Diagnostics Services

# Memory Thinking for Rust

---

Slides with Descriptions and Source Code Illustrations

Second Editon

**Dmitry Vostokov**  
**Software Diagnostics Services**

Memory Thinking for Rust: Slides with Descriptions and Source Code Illustrations, Second Edition

Published by OpenTask, Republic of Ireland

Copyright © 2025 by OpenTask

Copyright © 2025 by Dmitry Vostokov

Copyright © 2025 by Software Diagnostics Services

Copyright © 2025 by Dublin School of Security

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, transmitted in any form or by any means, or used for training artificial intelligence systems without the prior written permission of the publisher.

OpenTask books are available through booksellers and distributors worldwide. For further information or comments, send requests to [press@opentask.com](mailto:press@opentask.com).

Product and company names mentioned in this book may be trademarks of their owners.

A CIP catalog record for this book is available from the British Library.

ISBN-13: 978-1912636488 (Paperback)

Revision 2.00 (April 2025)

## Table of Contents

|                                      |           |
|--------------------------------------|-----------|
| <b>Table of Contents</b>             | <b>3</b>  |
| <b>Preface</b>                       | <b>9</b>  |
| <b>About the Author</b>              | <b>10</b> |
| <b>Introduction</b>                  | <b>11</b> |
| <i>Prerequisites</i>                 | 11        |
| <i>Training Goals</i>                | 12        |
| <i>Warning</i>                       | 12        |
| <i>Training Principles</i>           | 13        |
| <i>Schedule</i>                      | 13        |
| <i>Training Idea</i>                 | 14        |
| <i>General Rust Aspects</i>          | 14        |
| <i>What We Do Not Cover</i>          | 15        |
| <i>Linux Rust Aspects</i>            | 16        |
| <i>Windows Rust Aspects</i>          | 16        |
| <i>Why Rust?</i>                     | 17        |
| <i>My Genealogy of Rust</i>          | 18        |
| <i>Rust Mastery Process</i>          | 19        |
| <i>Thought Process</i>               | 19        |
| <b>Philosophy of Unsafe Pointers</b> | <b>20</b> |
| <i>A General Pointer Concept</i>     | 20        |
| <i>Pointer</i>                       | 21        |

|   |           |
|---|-----------|
| <i>Pointer Dereference</i>                            | 21        |
| <i>One to Many</i>                                    | 22        |
| <i>Many to One</i>                                    | 22        |
| <i>Many to One Dereference</i>                        | 23        |
| <i>Invalid Pointer</i>                                | 23        |
| <i>Invalid Pointer Dereference</i>                    | 24        |
| <i>Wild (Dangling) Pointer</i>                        | 24        |
| <i>Pointer to Pointer</i>                             | 25        |
| <i>Pointer to Pointer Dereference</i>                 | 25        |
| <i>Naming Pointers and Entities</i>                   | 26        |
| <i>Names as Pointer Content</i>                       | 26        |
| <i>Pointers as Entities</i>                           | 27        |
| <b>Unsafe Rust Code Examples</b>                      | <b>28</b> |
| <i>Pointer</i>  | 29        |
| <i>Unsafe Pointer Dereference</i>                     | 30        |
| <i>One to Many</i>                                    | 32        |
| <i>Memory Leak</i>                                    | 34        |
| <i>Many to One</i>                                    | 35        |
| <i>Unsafe Many to One Dereference</i>                 | 37        |
| <i>Invalid Pointer</i>                                | 38        |
| <i>Unsafe Invalid Pointer Dereference (Alignment)</i> | 39        |

|  |           |   |           |
|--|-----------|---|-----------|
| <i>Unsafe Invalid Pointer Dereference (Access Violation)</i> | 40        | <i>Value Lifetime</i>                               | 63        |
| <i>Unsafe Wild (Dangling) Pointer</i>                        | 41        | <i>Owner Lifetime</i>                               | 65        |
| <i>Pointer to Pointer</i>                                    | 42        | <b>Rust Philosophy of Pointers</b>                  | <b>68</b> |
| <i>Unsafe Pointer to Pointer Dereference</i>                 | 43        | <i>Types of Pointers</i>                            | 69        |
| <b>Philosophy of Values</b>                                  | <b>45</b> | <i>Mut Pointers vs. Pointers to Mut</i>             | 69        |
| <i>Values and Owners</i>                                     | 46        | <i>References as Pointer Types</i>                  | 70        |
| <i>Moving Values</i>   | 47        | <i>Mut Refs vs. Refs to Mut</i>                     | 72        |
| <i>Copying Values</i>  | 49        | <i>References as Addresses</i>                      | 73        |
| <i>Dropping Values</i>                                       | 50        | <i>Borrowing References</i>                         | 75        |
| <i>Ownership Tree</i>  | 52        | <i>Reference Lifetime</i>                           | 77        |
| <i>Ownership Tree and Drops</i>                              | 53        | <b>x64 Disassembly Review (WinDbg)</b>              | <b>79</b> |
| <i>Partial Drops</i>   | 54        | <i>x64 CPU Registers</i>                            | 79        |
| <i>Ownership Tree and Moves</i>                              | 56        | <i>Instructions and Registers</i>                   | 80        |
| <i>Multiple Owners (not in Rust)</i>                         | 57        | <i>Memory and Stack Addressing</i>                  | 80        |
| <i>Multiple Owners and Drops</i>                             | 58        | <i>Memory Cell Sizes</i>                            | 81        |
| <i>Owners vs. Pointers</i>                                   | 58        | <i>Memory Load Instructions</i>                     | 81        |
| <b>Rust: A Copernican Revolution</b>                         | <b>59</b> | <i>Memory Store Instructions</i>                    | 82        |
| <i>Values Revolve Around Pointers</i>                        | 59        | <i>Flow Instructions</i>                            | 82        |
| <i>Owners Revolve Around Values</i>                          | 60        | <i>Function Parameters</i>                          | 83        |
| <b>Rust Philosophy of Values</b>                             | <b>61</b> | <i>Struct Function Parameters</i>                   | 83        |
| <i>Restricted Ownership</i>                                  | 62        | <b>x64 Disassembly Review (GDB AT&amp;T Flavor)</b> | <b>84</b> |
|  |           | <i>x64 CPU Registers</i>                            | 84        |

|                                       |           |                                      |            |
|---------------------------------------|-----------|--------------------------------------|------------|
| <i>x64 Instructions and Registers</i> | 85        | <i>Rust Static Memory References</i> | 97         |
| <i>Memory and Stack Addressing</i>    | 85        | <i>Stack Memory</i>                  | 102        |
| <i>x64 Memory Load Instructions</i>   | 86        | <i>Thread Stack Frames</i>           | 102        |
| <i>x64 Memory Store Instructions</i>  | 86        | <i>Local Value Lifecycle</i>         | 103        |
| <i>x64 Flow Instructions</i>          | 87        | <i>Scope</i>                         | 103        |
| <i>x64 Function Parameters</i>        | 87        | <i>Rust Stack Memory Values</i>      | 104        |
| <i>x64 Struct Function Parameters</i> | 88        | <i>Rust Stack Memory References</i>  | 104        |
| <b>ARM64 Disassembly Review</b>       | <b>89</b> | <i>Heap Memory</i>                   | 107        |
| <i>A64 CPU Registers</i>              | 89        | <i>Rust Heap Memory Values</i>       | 108        |
| <i>A64 Instructions and Registers</i> | 90        | <i>Rust Const Values</i>             | 110        |
| <i>Memory and Stack Addressing</i>    | 90        | <i>Useful WinDbg Commands</i>        | 112        |
| <i>A64 Memory Load Instructions</i>   | 91        | <i>Useful GDB Commands</i>           | 112        |
| <i>A64 Memory Store Instructions</i>  | 91        | <b>Memory and Pointers</b>           | <b>113</b> |
| <i>A64 Flow Instructions</i>          | 92        | <i>Mental Exercise</i>               | 114        |
| <i>A64 Function Parameters</i>        | 92        | <i>Debugger Memory Layout</i>        | 114        |
| <i>A64 Struct Function Parameters</i> | 93        | <i>Memory Dereference Layout</i>     | 115        |
| <b>Memory Storage</b>                 | <b>94</b> | <i>Names as Addresses</i>            | 115        |
| <i>Memory Regions</i>                 | 95        | <i>Addresses and Entities</i>        | 116        |
| <i>Dynamic Virtual Memory</i>         | 95        | <i>Addresses and Structures</i>      | 116        |
| <i>Static Memory</i>                  | 96        | <i>Pointers to Structures</i>        | 117        |
| <i>Rust Static Memory Values</i>      | 96        | <i>Arrays</i>                        | 117        |
| <i>Global vs. Local Static</i>        | 97        | <i>Arrays and Pointers to Arrays</i> | 118        |

|  |            |   |            |
|--|------------|---|------------|
| <i>Fat Pointers</i>                    | 118        | <b>Tuples</b>                             | <b>157</b> |
| <i>Array Slices</i>                    | 119        | <b>Structs</b>                            | <b>159</b> |
| <i>String Literals (UTF-8)</i>         | 122        | <i>Tuple-like Structs</i>                 | 160        |
| <i>Byte Strings</i>                    | 124        | <i>Newtypes</i>                           | 163        |
| <i>Vectors</i>                         | 125        | <i>Newtypes (Binary Compatible)</i>       | 165        |
| <i>Vector Slices</i>                   | 126        | <i>Named-field Structs</i>                | 166        |
| <i>Strings</i>                         | 129        | <i>Reference/Pointer to Struct</i>        | 169        |
| <i>String Slices</i>                   | 130        | <i>Ref/Ptr to Struct Dereference</i>      | 170        |
| <i>C-Strings</i>                       | 132        | <i>Dereference with Replacement</i>       | 171        |
| <i>C-String Slices</i>                 | 133        | <i>One Ref/Ptr to Many Structs</i>        | 173        |
| <b>Basic Types</b>                     | <b>135</b> | <i>Memory Leak</i>                        | 173        |
| <i>Bytes, Pointers, and References</i> | 136        | <i>Many Ref/Ptr to One Struct</i>         | 174        |
| <i>u32, Pointers, and References</i>   | 138        | <i>Many to One Dereference</i>            | 174        |
| <i>Little-Endian System</i>            | 140        | <i>Ref/Ptr to Ref/Ptr to Struct</i>       | 175        |
| <i>u64 and Pointers</i>                | 141        | <i>Ref/Ptr to Ref/Ptr Dereference</i>     | 175        |
| <i>Size</i>                            | 143        | <b>Memory and Structs</b>                 | <b>179</b> |
| <i>Alignment</i>                       | 147        | <i>Addresses and Structs</i>              | 180        |
| <b>Entity Conversion</b>               | <b>149</b> | <i>Struct Field Addresses</i>             | 180        |
| <i>Conversion through Pointers</i>     | 150        | <i>Ref/Ptr to Structs</i>                 | 183        |
| <i>Safe Conversion (Explicit Cast)</i> | 151        | <i>Ref/Ptr to Struct and Fields</i>       | 184        |
| <i>Safe Conversion (Coercion)</i>      | 153        | <i>External Struct Alignment</i>          | 187        |
| <i>Forcing</i>                         | 155        | <i>Internal Struct Alignment (WinDbg)</i> | 187        |

|   |            |  |            |
|---|------------|--|------------|
| <i>Internal Struct Alignment (GDB)</i>  | 188        | <i>Boxed Trait Object Layout</i>         | 225        |
| <b>Enums</b>                            | <b>191</b> | <i>Struct Constructors</i>               | 228        |
| <i>Simple Enums</i>                     | 192        | <i>Struct Destructor</i>                 | 230        |
| <i>Enums with Structs</i>               | 195        | <i>Struct Clone</i>                      | 234        |
| <i>Enum Null Pointer Optimization</i>   | 199        | <i>Struct Copy</i>                       | 235        |
| <b>Source Code and Symbols</b>          | <b>201</b> | <i>Parameters by Value</i>               | 237        |
| <i>Conceptual Layer (Modules)</i>       | 202        | <i>Parameters by Ref/Ptr</i>             | 240        |
| <i>Logical Layer (Crates)</i>           | 202        | <i>self</i>                              | 242        |
| <i>Physical Layer (Source Files)</i>    | 203        | <i>Trait Objects as Parameters</i>       | 243        |
| <i>Name Isolation</i>                   | 203        | <i>Struct as Return Value</i>            | 244        |
| <b>Functions</b>                        | <b>206</b> | <b>Closures and Captures</b>             | <b>246</b> |
| <i>Pointers to Functions</i>            | 207        | <i>Closure Struct</i>                    | 247        |
| <i>References to Functions</i>          | 209        | <i>A64 Closure Struct Example</i>        | 248        |
| <i>Function Pointer Types</i>           | 211        | <i>Captures (Borrowing)</i>              | 248        |
| <i>Struct Function Fields</i>           | 212        | <i>Captures (Borrowing) x64 Linux</i>    | 249        |
| <i>Associated Functions</i>             | 213        | <i>Captures (Move)</i>                   | 250        |
| <i>Pointers to Associated Functions</i> | 215        | <i>Captures (Move) x64 Linux</i>         | 251        |
| <i>Type-associated Functions</i>        | 217        | <b>Smart Pointers</b>                    | <b>253</b> |
| <i>Trait Functions</i>                  | 219        | <i>Why Smart Pointers?</i>               | 254        |
| <i>Trait Objects</i>                    | 219        | <i>Types of Smart Pointers</i>           | 254        |
| <i>vtable Memory Layout</i>             | 221        | <i>Interior Immutable Single Owner</i>   | 255        |
| <i>Trait Object Memory Layout</i>       | 221        | <i>Interior Mutable Single Ownership</i> | 257        |



*Shared Ownership* 261

**Pinning** 266

*Use Cases* 267

**Rust Books** 272