



# Rust

Windows Memory Dump Analysis

Accelerated

Dmitry Vostokov  
Software Diagnostics Services

# Prerequisites

- ⦿ Basic Windows troubleshooting
- ⦿ Basic **Rust** knowledge

## WinDbg Commands

We use these boxes to introduce WinDbg commands used in practice exercises

# Training Goals

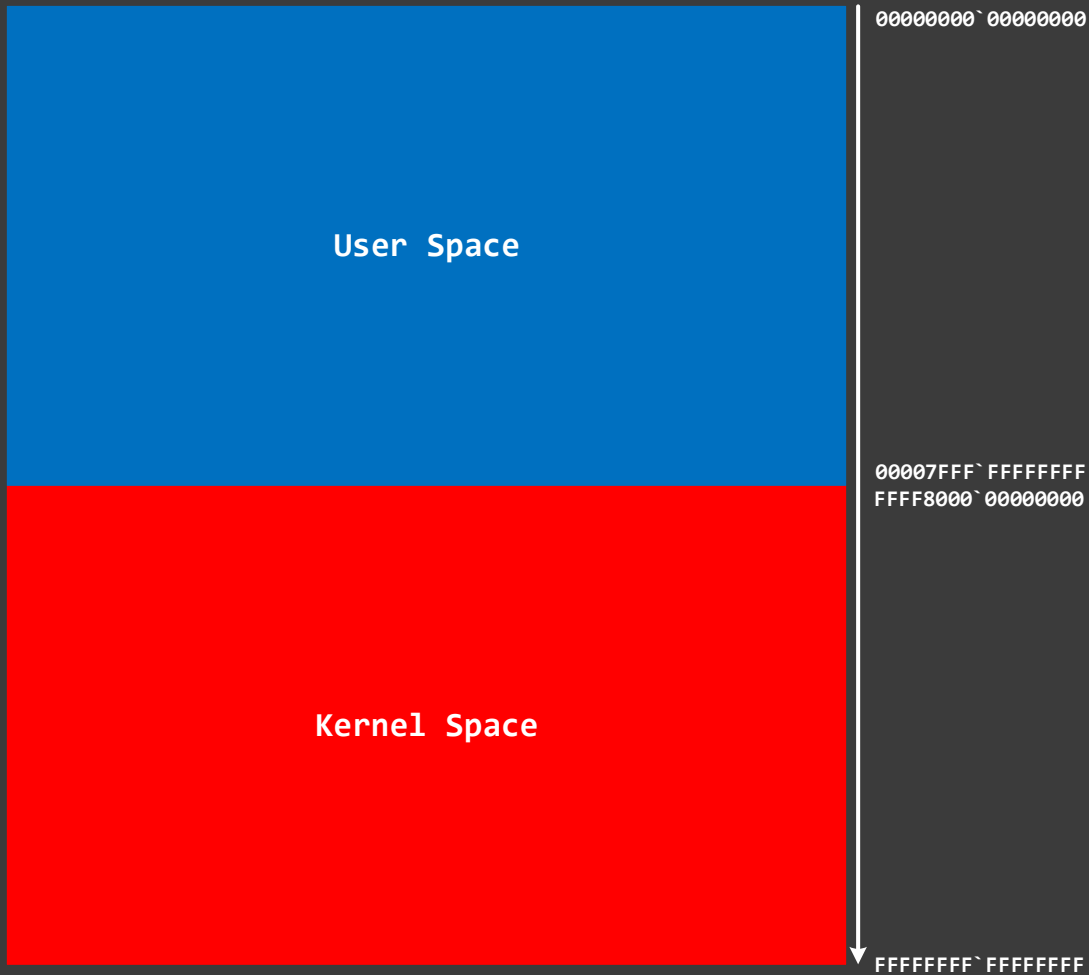
- Review fundamentals
- Review x64 disassembly
- Learn how to analyze process dumps
- Learn how to analyze complete (physical memory) dumps

# Training Principles

- Talk only about what I can show
- Lots of pictures
- Lots of examples
- Original content and examples

# Fundamentals

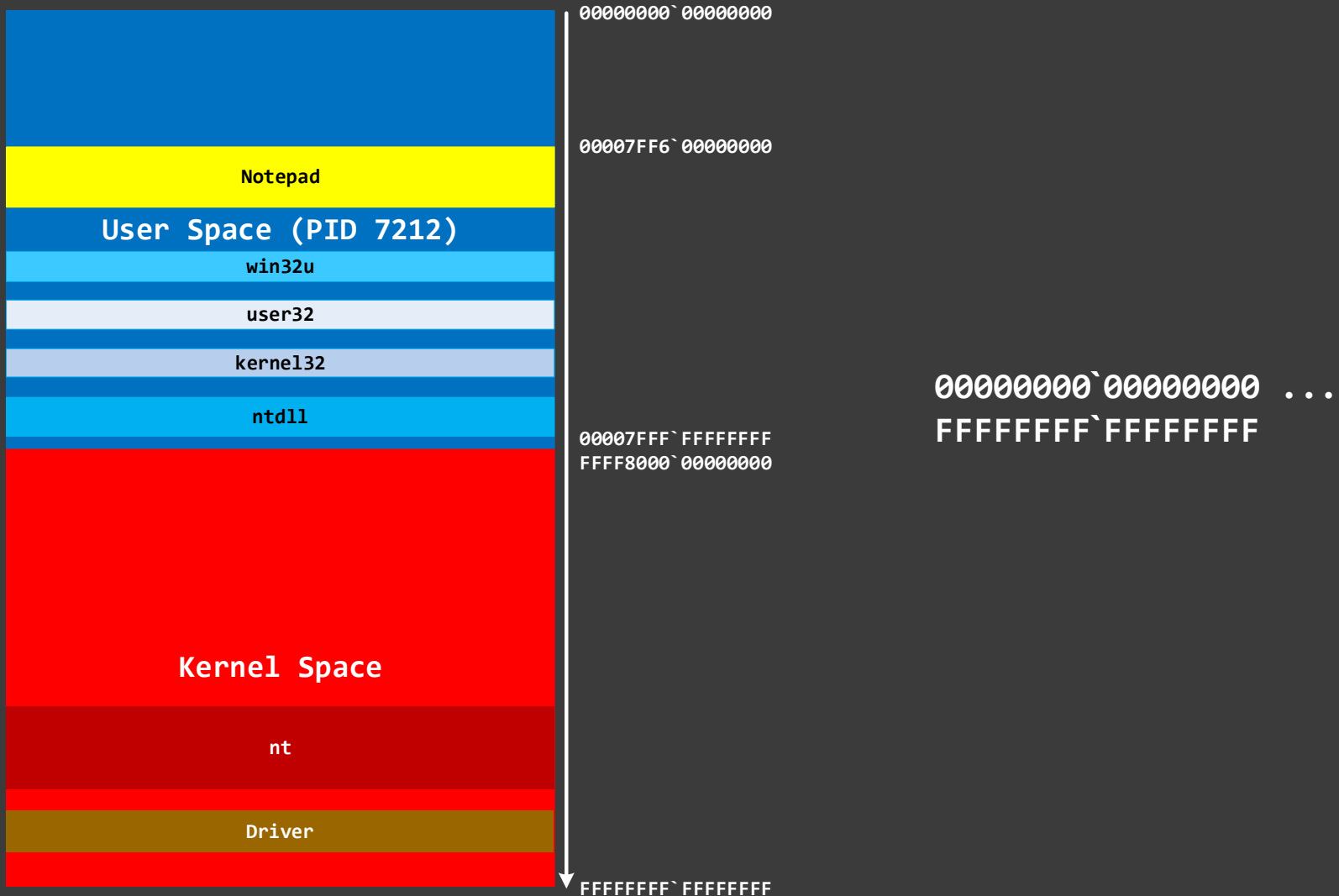
# Process Space (x64)



# Application/Process/Module (x64)

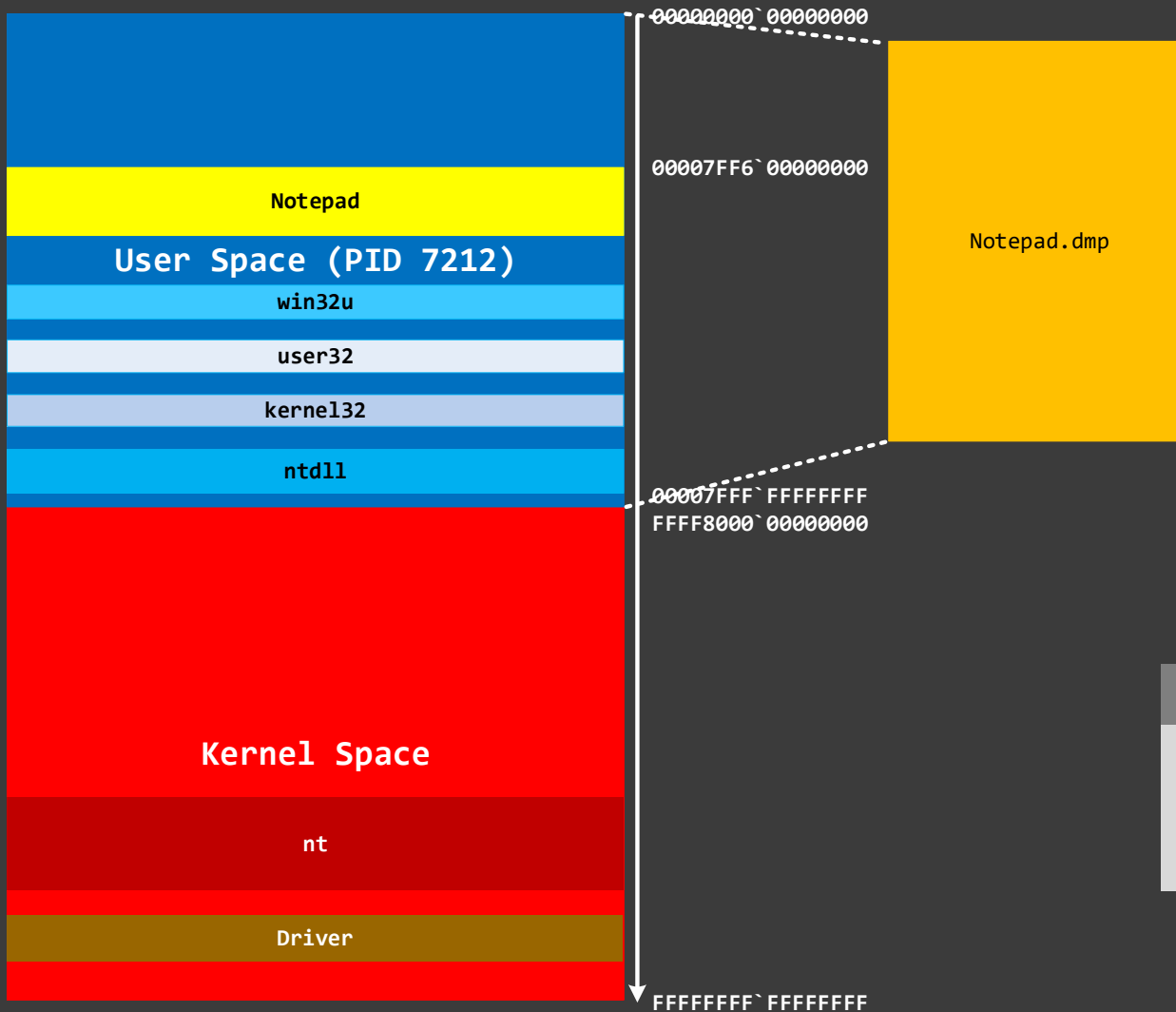


# Process Virtual Space (x64)





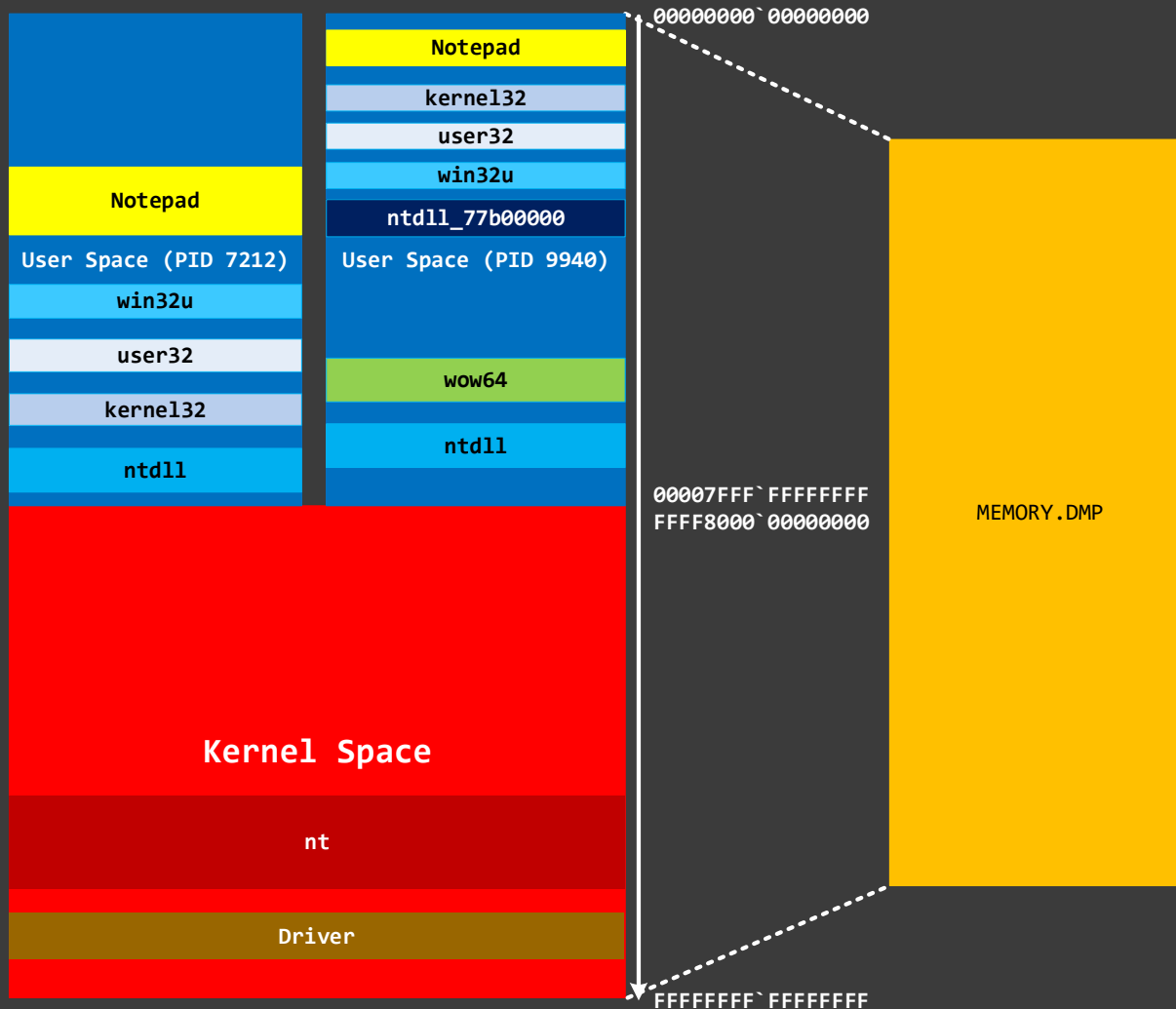
# Process Memory Dump (x64)



## WinDbg Commands

**!mv** command lists modules and their description

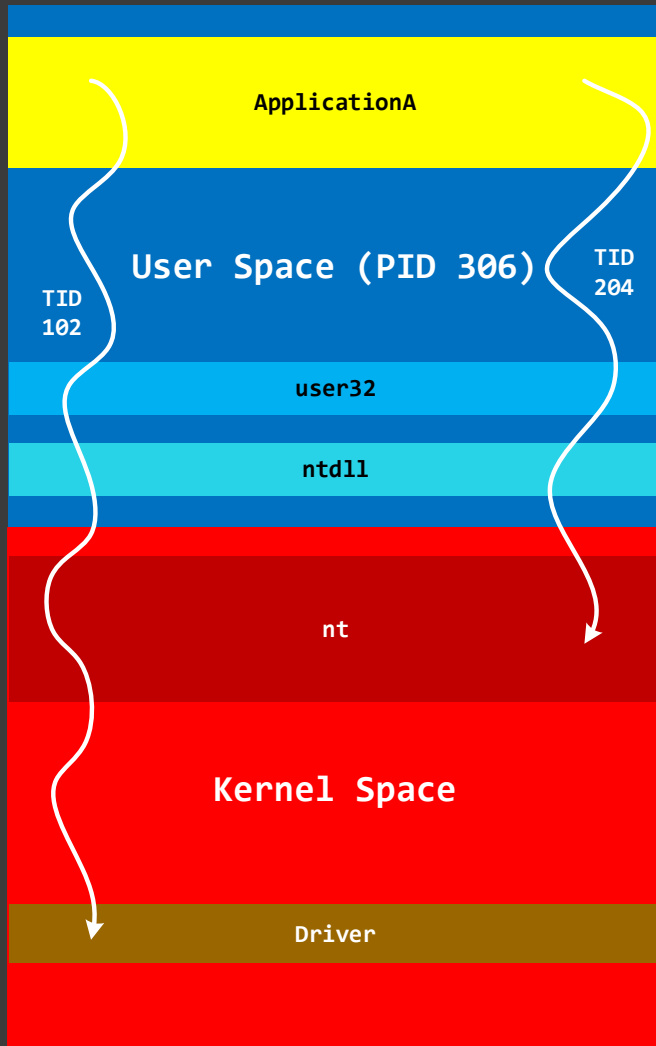
# Complete Memory Dump (x64)



**WinDbg Commands**

**.process** switches between process virtual spaces (kernel space part remains the same)

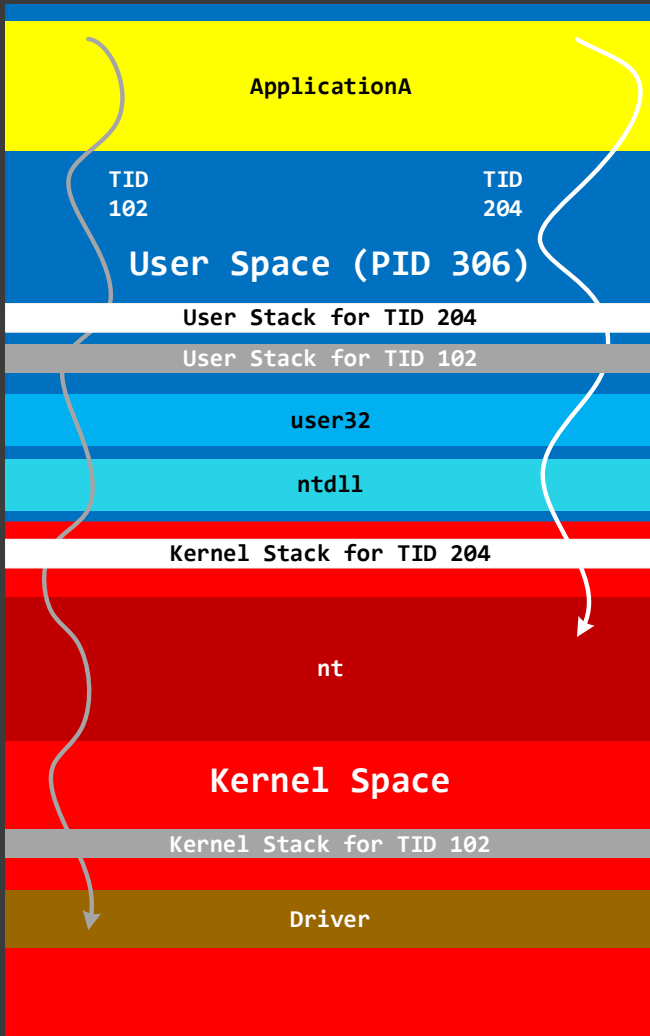
# Process Threads



## WinDbg Commands

Process dumps:  
~<n>s switches between threads

# Thread Stack Raw Data



## WinDbg Commands

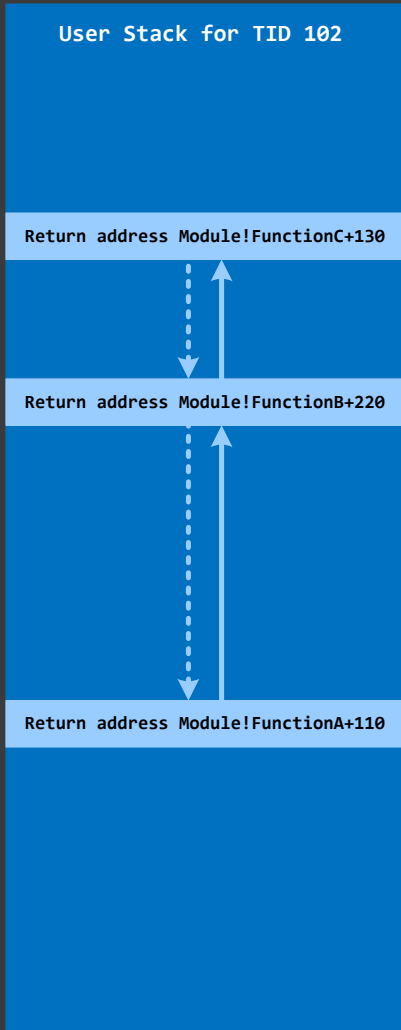
Process dumps:

**!teb**

Data:

**dc / dps / dpp / dpa / dpu**

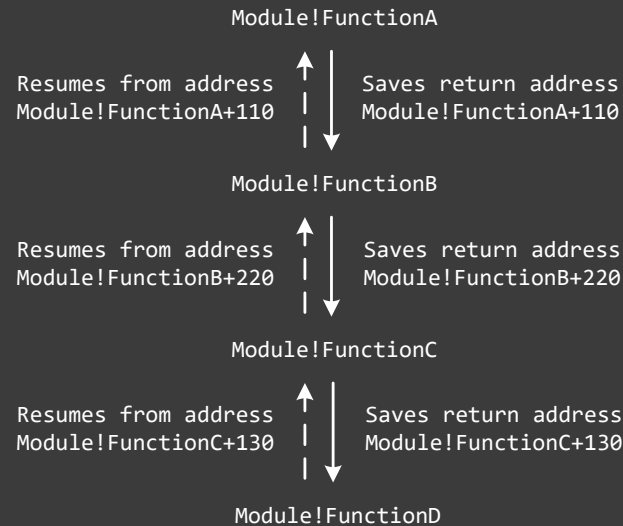
# Thread Stack Trace



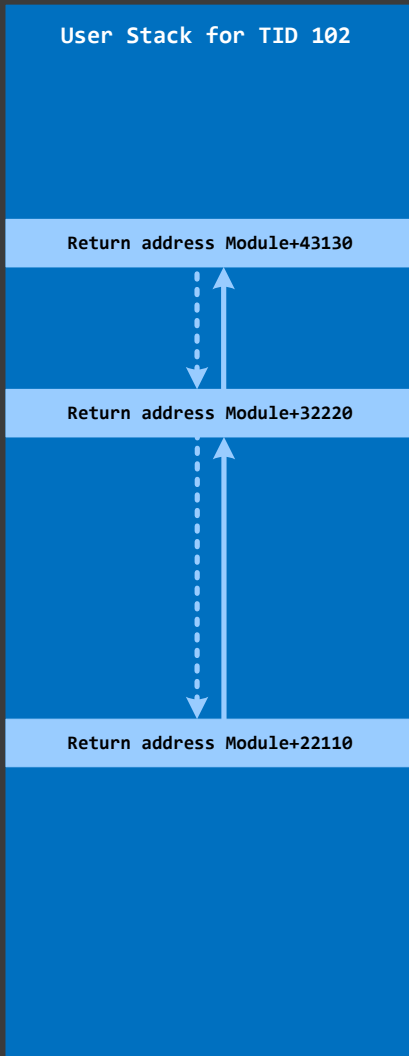
```
FunctionA()  
{  
  ...  
  FunctionB();  
  ...  
}  
FunctionB()  
{  
  ...  
  FunctionC();  
  ...  
}  
FunctionC()  
{  
  ...  
  FunctionD();  
  ...  
}
```

## WinDbg Commands

```
0:000> k  
Module!FunctionD  
Module!FunctionC+130  
Module!FunctionB+220  
Module!FunctionA+110
```



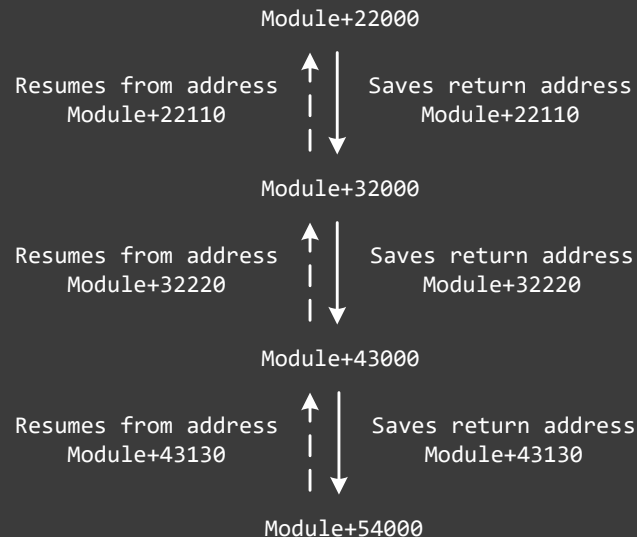
# Thread Stack Trace (no PDB)



```
FunctionA()  
{  
  ...  
  FunctionB();  
  ...  
}  
FunctionB()  
{  
  ...  
  FunctionC();  
  ...  
}  
FunctionC()  
{  
  ...  
  FunctionD();  
  ...  
}
```

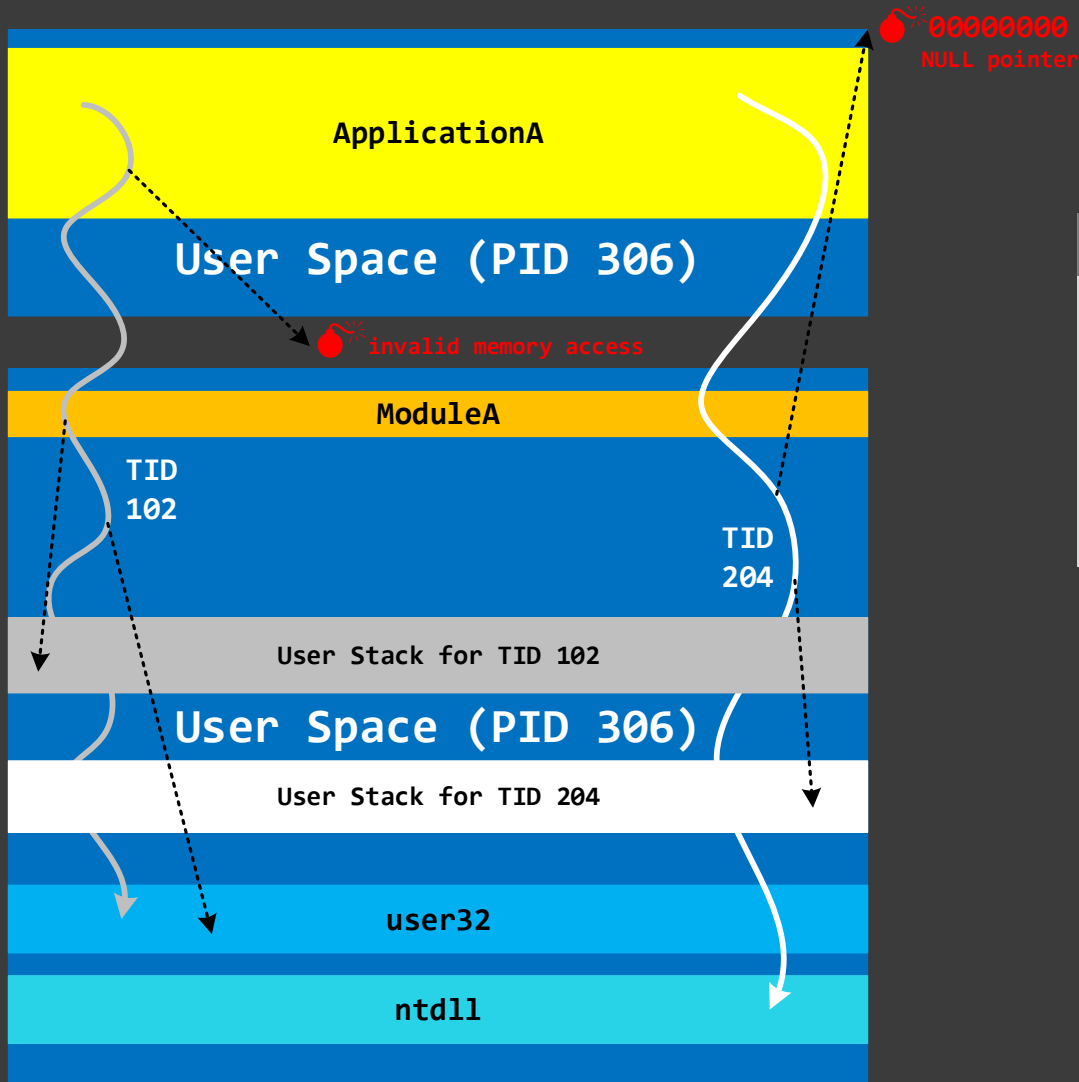
```
Symbol file Module.pdb  
  
FunctionA 22000 - 23000  
FunctionB 32000 - 33000  
FunctionC 43000 - 44000  
FunctionD 54000 - 55000
```

No symbols for Module



```
WinDbg Commands  
  
0:000> k  
Module+0  
Module+43130  
Module+32220  
Module+22110
```

# Exceptions (Access Violation)



```
WinDbg Commands

address=????????

Set exception context
(process dump):
.cxr
```

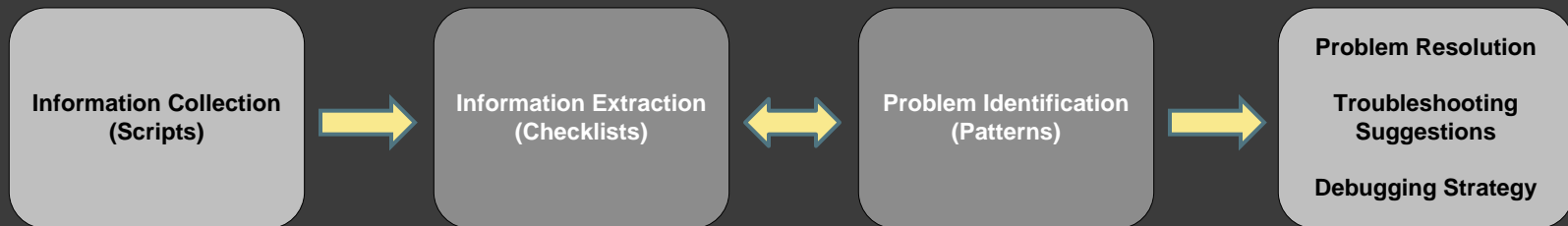
# Pattern-Oriented Diagnostic Analysis

**Diagnostic Pattern:** a common recurrent identifiable problem together with a set of recommendations and possible solutions to apply in a specific context.

**Diagnostic Problem:** a set of indicators (symptoms, signs) describing a problem.

**Diagnostic Analysis Pattern:** a common recurrent analysis technique and method of diagnostic pattern identification in a specific context.

**Diagnostics Pattern Language:** common names of diagnostic and diagnostic analysis patterns. The same language for any operating system: Windows, Mac OS X, Linux, ...



**Checklist:** <http://www.dumpanalysis.org/windows-memory-analysis-checklist>

**Patterns:** <http://www.dumpanalysis.org/blog/index.php/crash-dump-analysis-patterns/>



# Memory Dump Collection

# Process Dump Generation

- ◎ Crash or Hang, ... ?

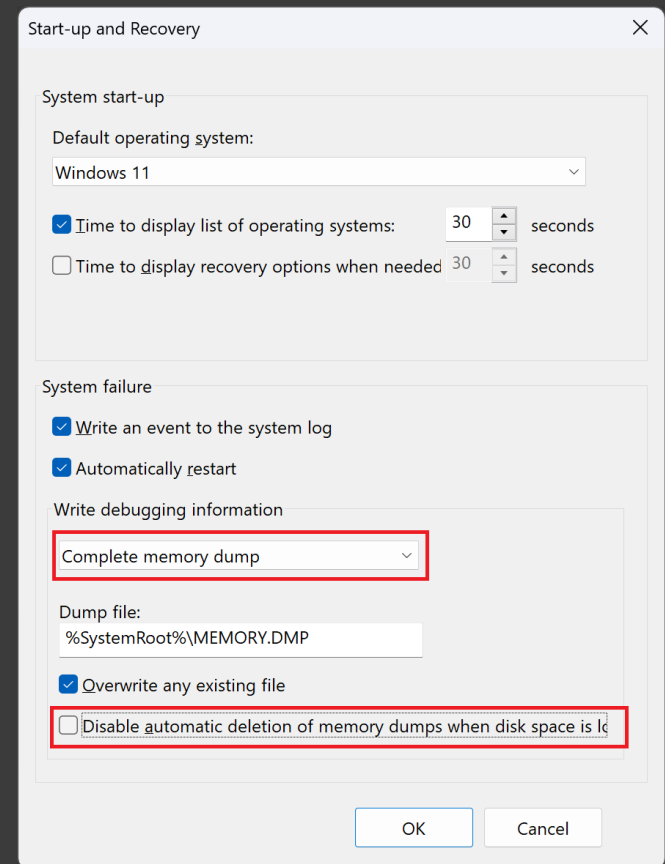
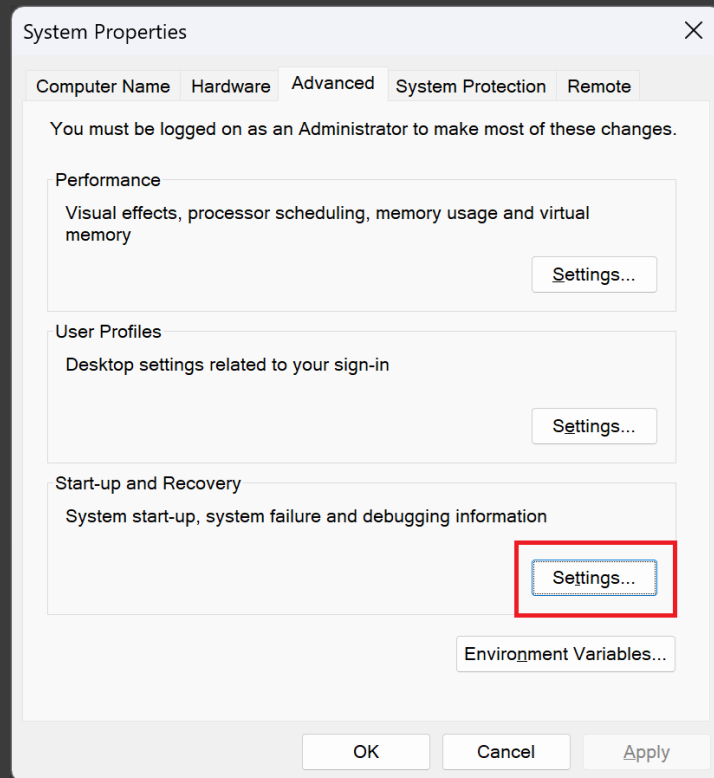
PID in Task Manager

- ◎ Windows W10, W11

- Crash: [LocalDumps](#)
- Hang / Leak / Spike: Task Manager, [procdump -ma](#)

# Complete Dump Setup

- View Advanced System Settings (Control Panel)
- Page file size > physical memory + 100 MB



# Complete Dump Generation

- ◎ [Keyboard](#), [NMI button](#)
- ◎ Tools: [NotMyFault](#)
- ◎ VMware memory snapshot + [vmss2core](#)

# Common Issues

- ⦿ Truncated complete dumps
- ⦿ No “Complete memory dump” option

HKLM \ SYSTEM \ CurrentControlSet \ Control \ CrashControl  
CrashDumpEnabled = 1 (DWORD)

# x64 Disassembly

# x64 CPU Registers

⦿ **RAX**  $\supset$  **EAX**  $\supset$  **AX**  $\supseteq$  {**AH**, **AL**}

**RAX 64-bit**

**EAX 32-bit**

⦿ ALU: **RAX**, **RDX**

⦿ Counter: **RCX**

⦿ Memory copy: **RSI** (src), **RDI** (dst)

⦿ Stack: **RSP**

⦿ Frame Pointer: **RSP**, **RBP**

⦿ Next instruction: **RIP**

⦿ New: **R8** – **R15**, **Rx(D|W|B)**

# Instructions and Registers

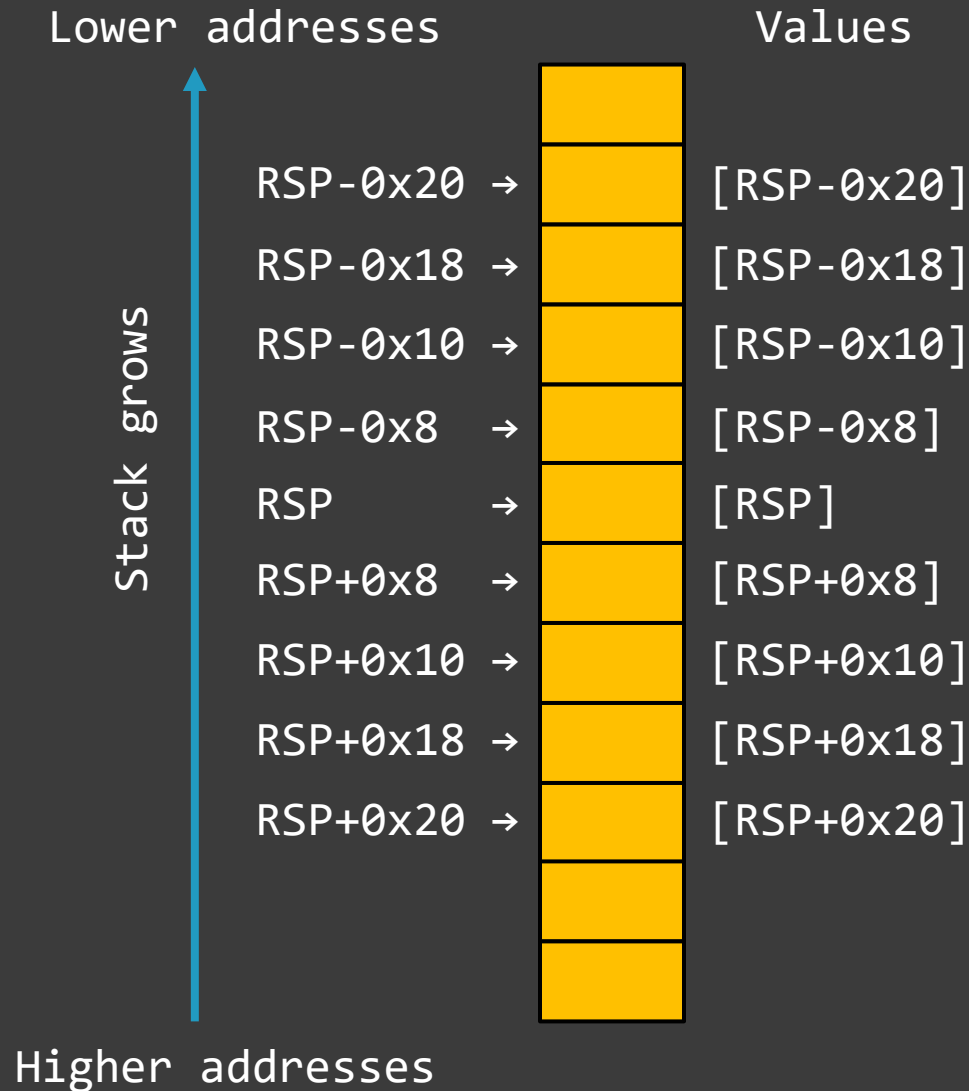
◎ Opcode DST, SRC

◎ Examples:

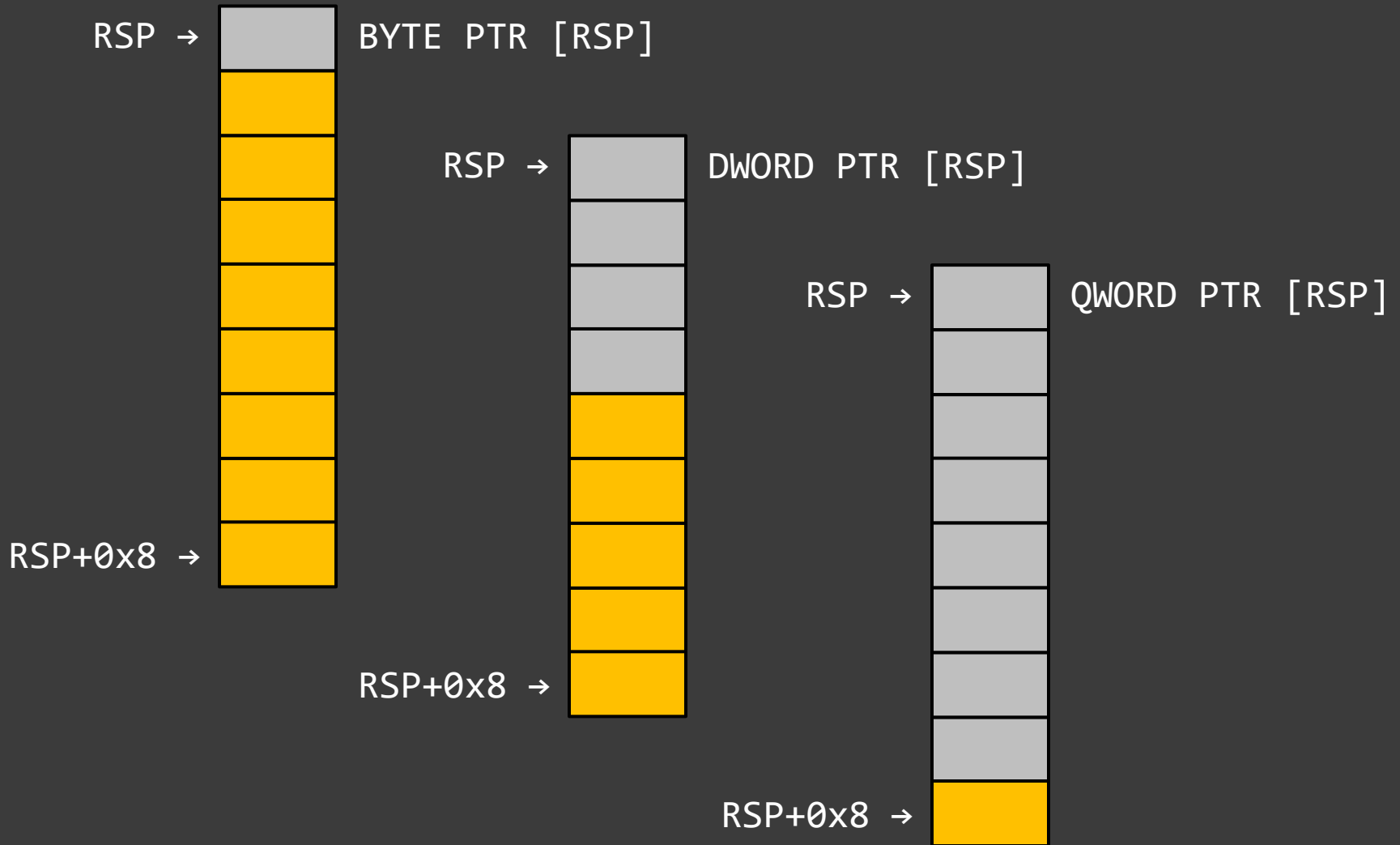
```
mov    rax, 10h           ; RAX ← 0x10
mov    r13, rdx           ; R13 ← RDX
add    r10, 10h           ; R10 ← R10 + 0x10
imul   edx, ecx           ; EDX ← EDX * ECX
call   rdx                ; RDX already contains
                          ; the address of func (&func)
                          ; PUSH RIP; RIP ← &func
sub    rsp, 30h           ; RSP ← RSP-0x30
                          ; make room for local variables
```



# Memory and Stack Addressing



# Memory Cell Sizes



# Memory Load Instructions

- ◉ Opcode **DST**, PTR [SRC+Offset]

- ◉ Opcode **DST**

- ◉ Examples:

```
mov    rax, qword ptr [rsp+10h] ; RAX ←  
                                           ; 64-bit value at address RSP+0x10  
mov    ecx, dword ptr [20]      ; ECX ←  
                                           ; 32-bit value at address 0x20  
pop    rdi                      ; RDI ← value at address RSP  
                                           ; RSP ← RSP + 8  
lea    r8, [rsp+20h]           ; R8 ← address RSP+0x20
```

# Memory Store Instructions

- ◉ Opcode PTR [DST+Offset], SRC

- ◉ Opcode DST|SRC

- ◉ Examples:

```
mov    qword ptr [rbp-20h], rcx ; 64-bit value at address RBP-0x20  
; ← RCX
```

```
mov    byte ptr [0], 1 ; 8-bit value at address 0 ← 1
```

```
push  rsi ; RSP ← RSP - 8  
; value at address RSP ← RSI
```

```
inc    dword ptr [rcx] ; 32-bit value at address RCX ←  
; 1 + 32-bit value at address RCX
```

# Flow Instructions

- ◉ Opcode DST

- ◉ Opcode PTR [DST]

- ◉ Examples:

```
jmp    00007ff6`9ef2f008    ; RIP ← 0x7ff69ef2f008  
                                ; (“goto” 0x7ff69ef2f008)
```

```
jmp    qword ptr [rax+10h] ; RIP ← value at address RAX+0x10
```

```
call   00007ff6`9ef21400    ; RSP ← RSP - 8
```

```
00007ff6`9ef21057:        ; value at address RSP ← 0x7ff69ef21057  
                                ; RIP ← 0x7ff69ef21400  
                                ; (“goto” 0x7ff69ef21400)
```

# Function Parameters

- ⦿ `fn func(...);`
- ⦿ Left to right via `RCX`, `RDX`, `R8`, `R9`, `stack`
- ⦿ `stack`: `[RSP+20]`, `[RSP+28]`, `[RSP+30]`, ...

# Struct Function Parameters

- ◎ **RCX**

Implicit struct object memory address (**&myStruct**)

```
let myStruct: Struct = ...;  
myStruct.func(...);
```

- ◎ **RDX, R8, R9, stack**

The rest of the struct function parameters

```
impl Struct {  
    fn func(&self, ...);  
}
```

# Windows API Parameters

x64: Left to right **RCX**, **RDX**, **R8**, **R9**, stack

Args to Child are **not** parameters

## WinDbg Commands

```
0:000> kv
# Child-SP  RetAddr      : Args to Child    : Call Site
...
```



# Practice Exercises

# Links

- Memory Dumps:

Included in Exercise RW1

- Exercise Transcripts:

Included in this book

# Process Memory Dumps

Exercises RW1 – RW8

# Exercise RW1

- ◎ **Goal:** Learn how to see dump file type and version, get a stack trace, check its correctness, perform default analysis, list threads and modules, check module version information, dump module data, and check the process environment
- ◎ **Patterns:** Manual Dump (Process); Incorrect Stack Trace; Stack Trace; Stack Trace Collection (Unmanaged Space); Main Thread; System Call; Not My Version (Software); Environment Hint; Unknown Component
- ◎ [\ARWMDA-Dumps\Exercise-RW1.pdf](#)

# Supportability Best Practice

Keep PDB files from each release

# Exercise RW2

- ◎ **Goal:** Learn how to analyze stack traces from debug versions
- ◎ **Patterns:** Technology-Specific Subtrace (Rust)
- ◎ [\ARWMDA-Dumps\Exercise-RW2.pdf](#)

# Exercise RW3

- ◎ **Goal:** Learn how to analyze stack traces from release versions
- ◎ **Patterns:** Adjoint Stack Trace; Hidden Frame; Inline Function Optimization (Unmanaged Code)
- ◎ [\ARWMDA-Dumps\Exercise-RW3.pdf](#)

# Exercise RW4

- ◎ **Goal:** Learn to recognize exceptions in process memory dumps and get their context
- ◎ **Patterns:** Exception Stack Trace; Exception Module; Multiple Exceptions (User Mode); NULL Pointer (Data); Invalid Pointer (General); NULL Pointer (Code)
- ◎ [\ARWMDA-Dumps\Exercise-RW4.pdf](#)



# Exercise RW5

- ◎ **Goal:** Learn how to recognize heap corruption, dump memory contents, follow critical section wait chains, and check error and status codes
- ◎ **Patterns:** Exception Thread; Dynamic Memory Corruption (Process Heap); Wait Chain (Critical Sections); Execution Residue (Unmanaged Space, User); Last Error Collection
- ◎ [\ARWMDA-Dumps\Exercise-RW5.pdf](#)

# Exercise RW6

- ◎ **Goal:** Learn how to debug heap corruption using page heap
- ◎ **Patterns:** Instrumentation Information
- ◎ [\ARWMDA-Dumps\Exercise-RW6.pdf](#)

# Exercise RW7

- ◎ **Goal:** Learn how to recognize CPU spikes, memory leaks, stack overflow, and handle leaks
- ◎ **Patterns:** Active Thread; Spiking Thread; Thread Age; Stack Overflow (User Mode); Near Exception; Memory Leak (Process Heap); Active Space; Handle Leak; Small Value; Punctuated Memory Leak
- ◎ [\ARWMDA-Dumps\Exercise-RW7.pdf](#)

# Exercise RW8

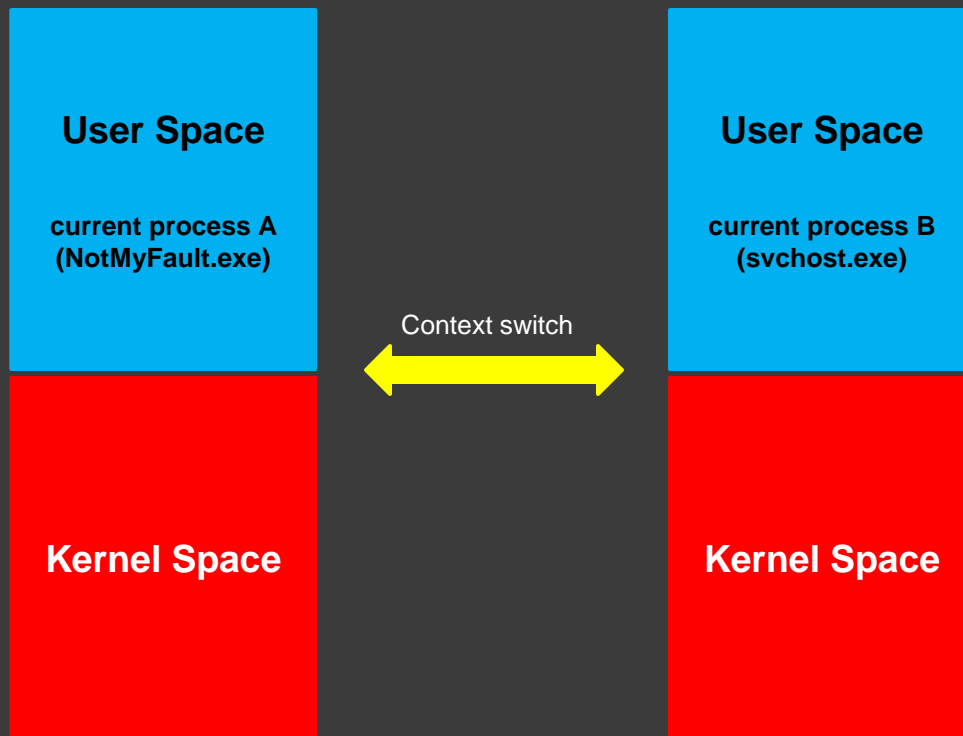
- ◎ **Goal:** Learn how to recognize waits and deadlocks, dump and analyze raw stack data
- ◎ **Patterns:** Deadlock (Futex)
- ◎ [\ARWMDA-Dumps\Exercise-RW8.pdf](#)

# Complete Memory Dumps

Exercises RC1 – RC2

# Memory Spaces

- Complete memory == Physical memory
- We always see the current process space
- Kernel space is the same for any process



## WinDbg Commands

switching to a different process context:

```
.process /r /p
```

# Major Challenges

- ◉ Multiple processes (user spaces) to examine
- ◉ User space view needs to be correct when we examine another thread



## WinDbg Commands

dump all stack traces:

```
!process 0 3f
```

# Common Commands

- ◉ **.logopen <file>**  
Opens a log file to save all subsequent output
- ◉ **View commands**  
Dump everything or selected processes and threads (context changes automatically)
- ◉ **Switch commands**  
Switch to a specific process or thread for a fine-grain analysis



# View Commands

- ◉ **!process 0 3f**  
Lists all processes (including times, environment, modules) and their thread stack traces
- ◉ **!process 0 1f**  
The same as the previous command but without PEB information (more secure)
- ◉ **!process <address> 3f or !process <address> 1f**  
The same as the previous commands but only for an individual process
- ◉ **!thread <address> 3f**  
Shows thread information and stack trace
- ◉ **!thread <address> 36**  
The same as the previous command but shows the first 4 parameters for every function

# Switch Commands

- **.process /r /p <address>**

Switches to a specified process. Its context becomes current. Reloads symbol files for user space. Now we can use commands like !cs

```
0: kd> .process /r /p fffffa80044d8b30
Implicit process is now fffffa80`044d8b30
Loading User Symbols
.....
```

- **.thread <address>**

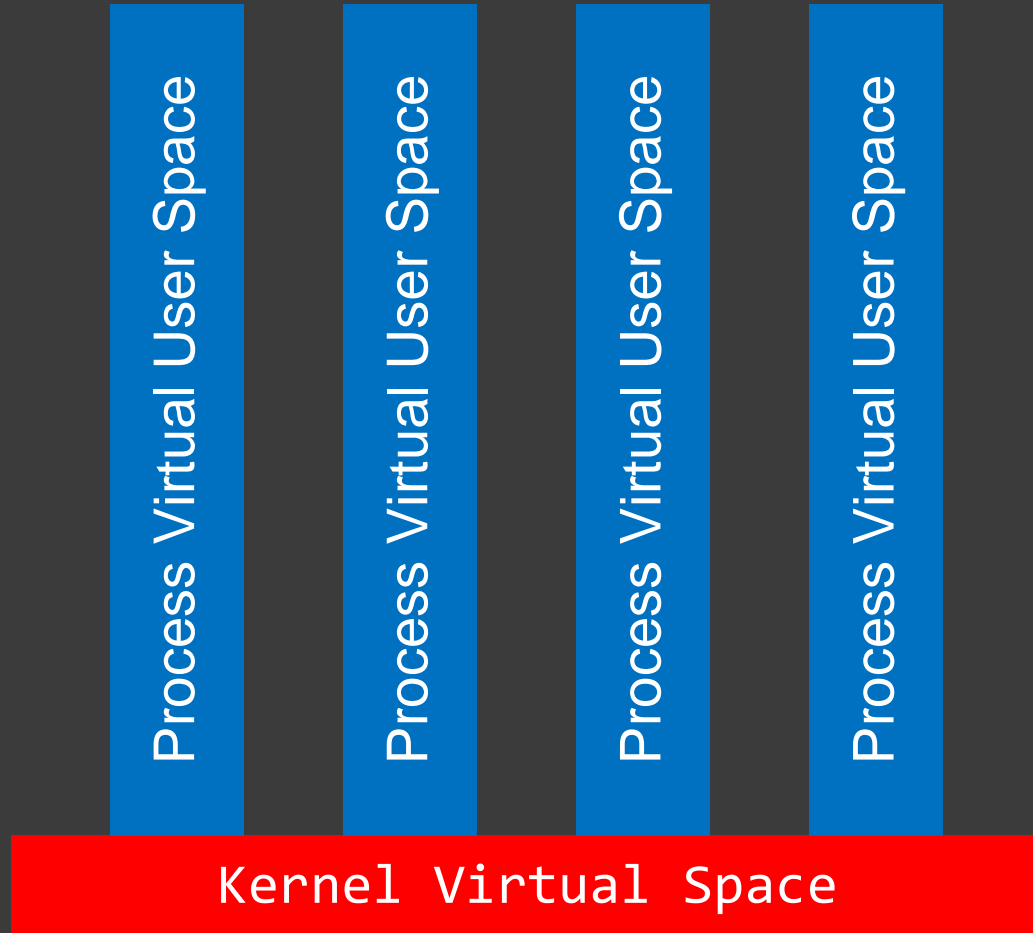
Switches to a specified thread. Assumes the current process context. Now we can use commands like k\*

- **.thread /r /p <address>**

The same as the previous command but makes the thread process context current and reloads symbol files for user space:

```
0: kd> .thread /r /p fffffa80051b7060
Implicit thread is now fffffa80`051b7060
Implicit process is now fffffa80`044d8b30
Loading User Symbols
.....
```

# Fiber Bindle Memory Dump



# Exercise RC1

- ◎ **Goal:** Learn how to get various information related to processes, threads, and modules
- ◎ **Patterns:** Stack Trace Collection (Unmanaged Space); Incorrect Stack Trace; Truncated Stack Trace; Dual Stack Trace; Raw Stack Trace; Coincidental Symbolic Information; Source Stack Trace
- ◎ [\ARWMDA-Dumps\Exercise-RC1.pdf](#)

# Exercise RC2

- ◎ **Goal:** Learn how to recognize various abnormal software behavior patterns in a physical memory context
- ◎ **Patterns:** Handle Leak; Stack Trace Collection (CPUs); Active Thread; Spiking Thread
- ◎ [\ARWMDA-Dumps\Exercise-RC2.pdf](#)

# Additional Resources

- WinDbg Help / [WinDbg.org](http://WinDbg.org) (quick links)
- [DumpAnalysis.org](http://DumpAnalysis.org) / [SoftwareDiagnostics.Institute](http://SoftwareDiagnostics.Institute) / [PatternDiagnostics.com](http://PatternDiagnostics.com)
- [Debugging.TV](http://Debugging.TV) / [YouTube.com/DebuggingTV](http://YouTube.com/DebuggingTV) / [YouTube.com/PatternDiagnostics](http://YouTube.com/PatternDiagnostics)
- [Principles of Memory Dump Analysis](#)
- [Fundamentals of Physical Memory Analysis: Anniversary Edition](#)
- [Encyclopedia of Crash Dump Analysis Patterns, 3<sup>rd</sup> edition](#)
- [Memory Thinking for Rust](#)
- [Accelerated Windows API for Software Diagnostics](#)
- [Memory Dump Analysis Anthology \(Diagnomicon\)](#)



# Q&A

Please send your feedback using the contact form on [PatternDiagnostics.com](https://PatternDiagnostics.com)

Thank you for attendance!