



Windows Memory Dump Analysis

Accelerated

Version 7

Part 2: Kernel and Complete Spaces

Dmitry Vostokov
Software Diagnostics Services

New

- ⦿ Overview of relevant Windows internals

After learning memory dump analysis!

- ⦿ Memory dump collection methods and patterns

After learning memory dump analysis and reviewing relevant Windows internals!

- ⦿ Additional memory analysis patterns

- ⦿ Windows ARM64

Prerequisites

WinDbg Commands

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

Basic Windows troubleshooting

* Part 1: Process User Space

Training Goals

- Part 1A: Review fundamentals
- Part 1B: Review x64 disassembly (optional)
- Part 1C: Review ARM64 disassembly (optional)
- Part 1D: Learn how to analyze process dumps
- Part 1E: Review relevant Windows internals
- Part 1F: Learn how to collect process dumps
- Part 2A: Review fundamentals
- Part 2B: Review x64 disassembly (optional)
- **Part 2C: Review ARM64 disassembly (optional)**
- Part 2D: Learn how to analyze kernel dumps
- Part 2E: Learn how to analyze complete (physical memory) dumps
- **Part 2F: Review relevant Windows internals**
- **Part 2G: Learn how to collect kernel and complete dumps**
- Part 2H: Learn how to analyze minidumps

Training Principles

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

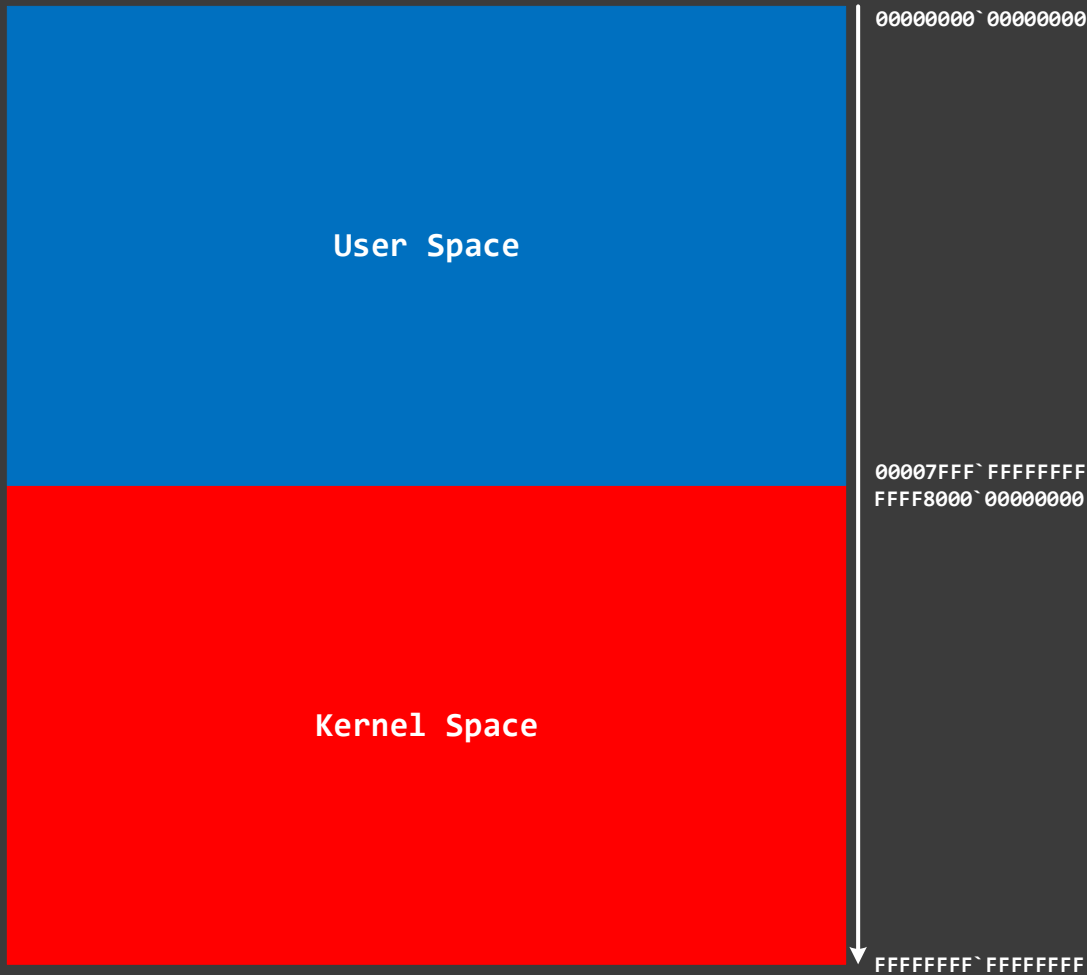
Coverage (Part 2)

- ⦿ Windows 10 and 11
- ⦿ Both x64* and x86 code, WOW64, ARM64
- ⦿ x64 and ARM64 disassembly review
- ⦿ Kernel and complete (physical) memory dumps; minidumps
- ⦿ Blue screens (BSOD), hangs, memory and handle leaks, CPU spikes

* Most of the exercises are focused on x64 code. For their x86 equivalents from older Windows versions, please refer to the previous fourth edition of this course.

Part 2A: Fundamentals

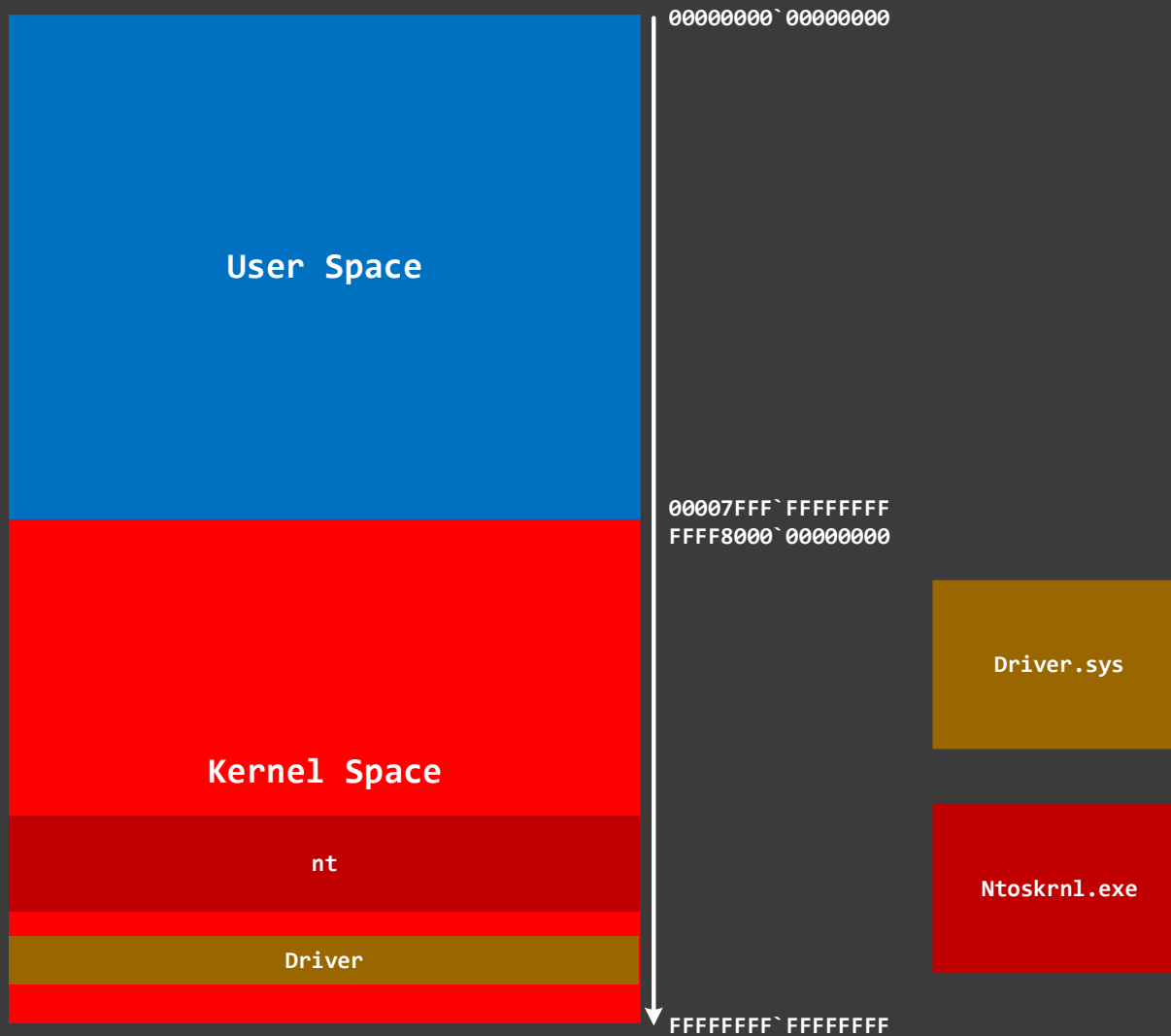
Process Space (x64, ARM64)



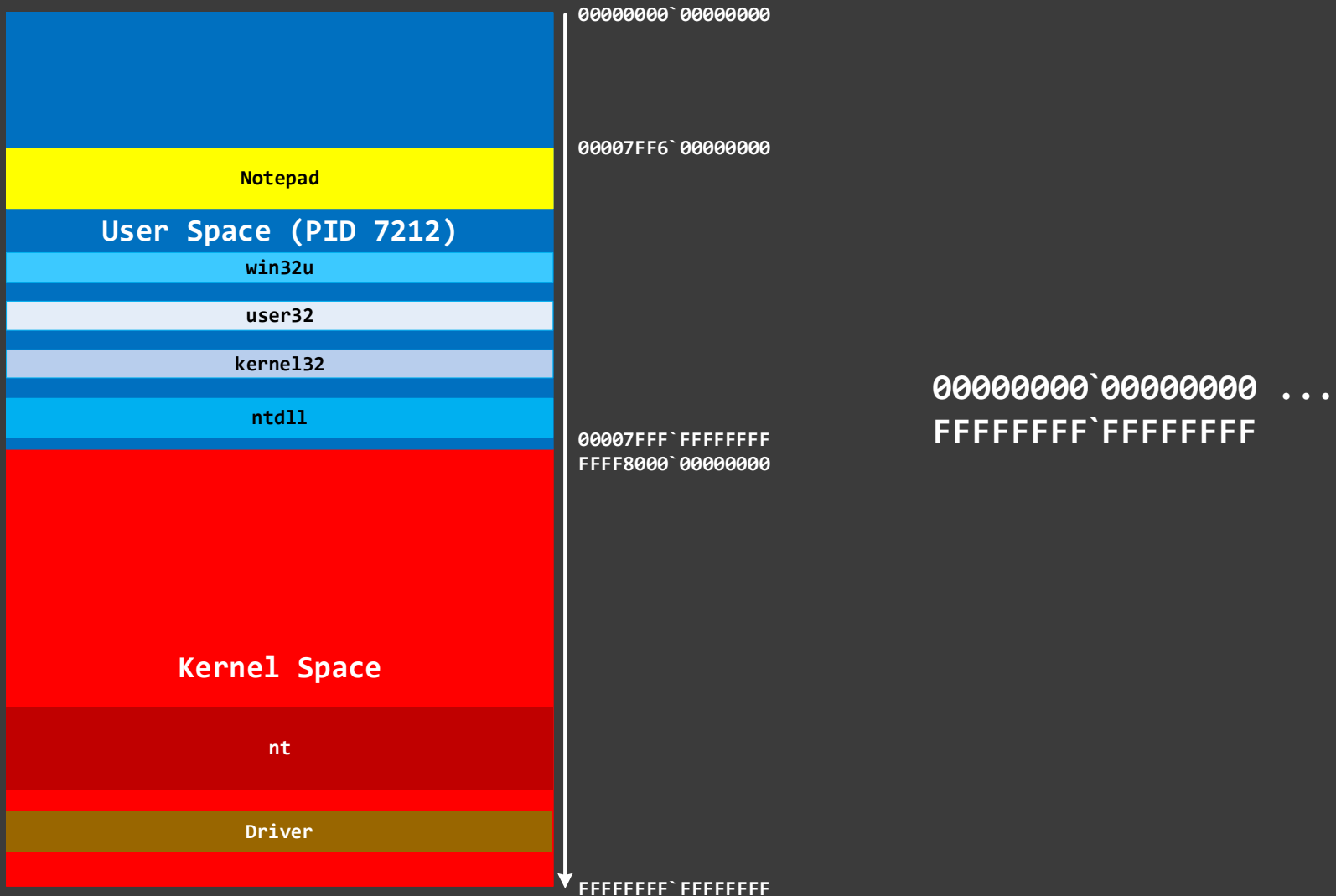
Application/Process/Module



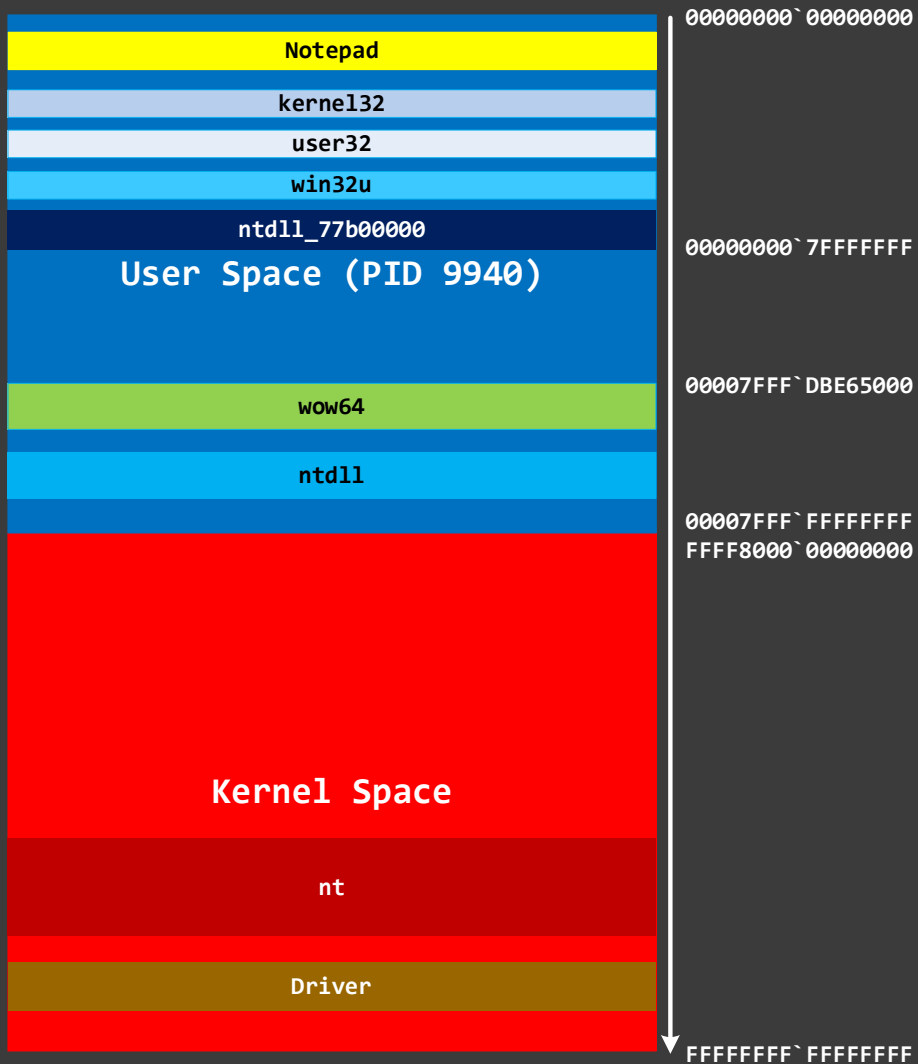
OS Kernel/Driver/Module



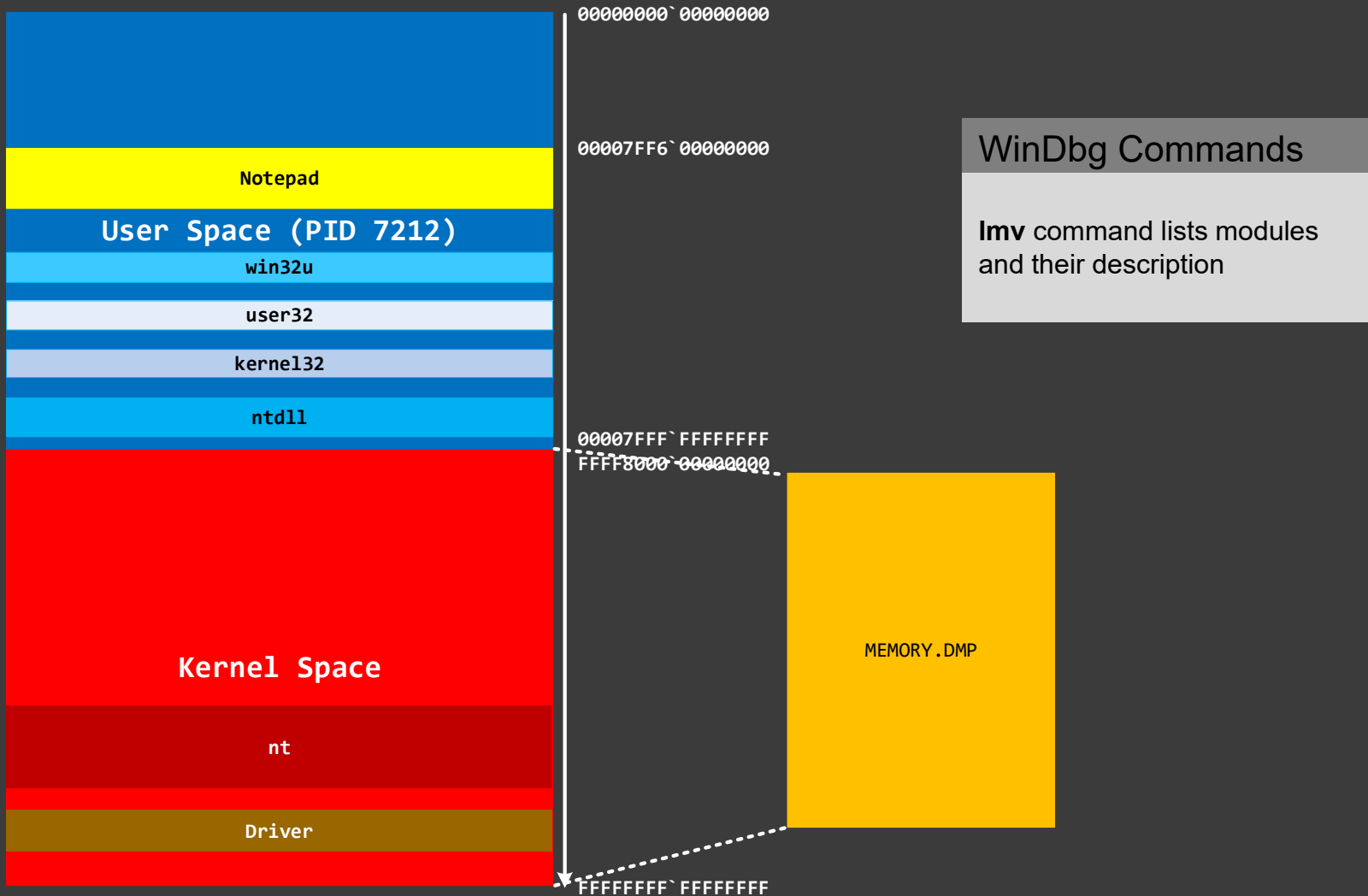
Process Virtual Space (x64, A64)



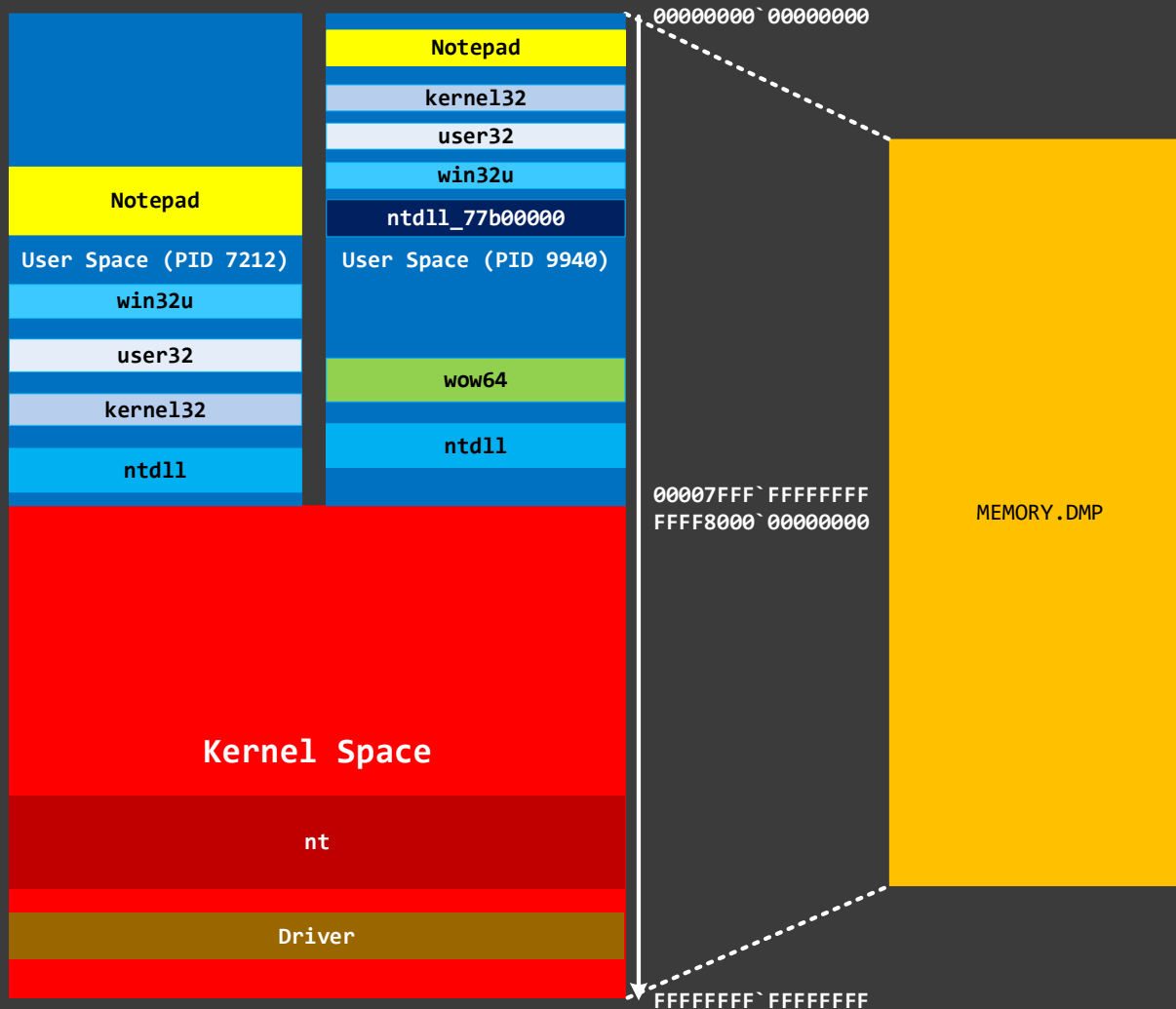
Process Virtual Space (WOW64)



Kernel Memory Dump (x64, A64)



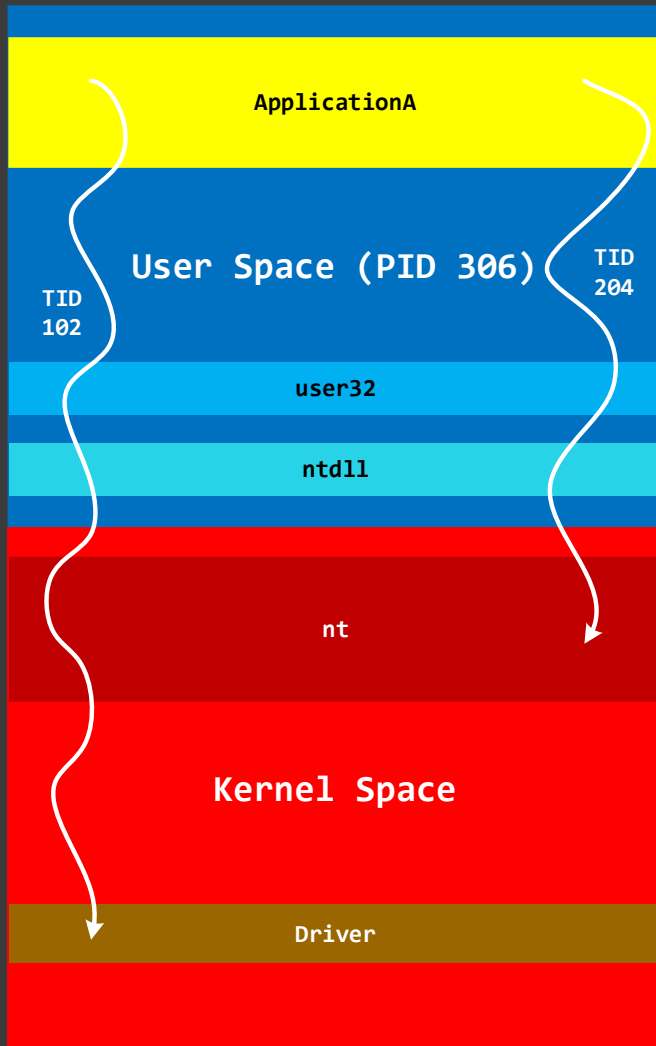
Complete Memory Dump



WinDbg Commands

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

Process Threads



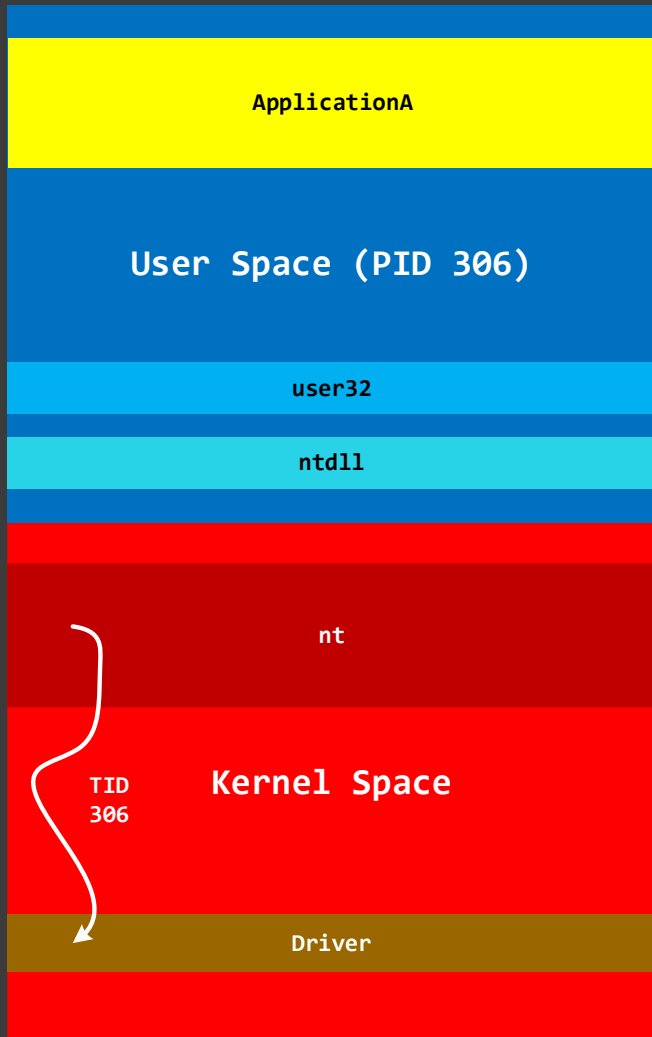
WinDbg Commands

Kernel/Complete dumps:

`~<n>s` switches between processors

`.thread` switches between threads

System Threads



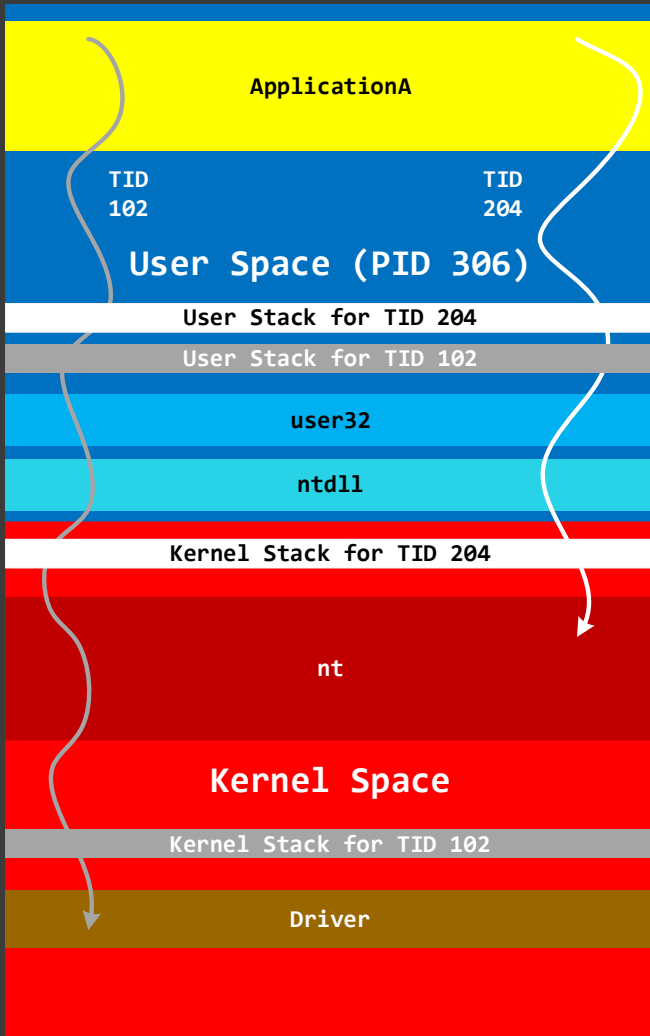
WinDbg Commands

Kernel/Complete dumps:

`~<n>s` switches between processors

`.thread` switches between threads

Thread Stack Raw Data



WinDbg Commands

Kernel dumps:

!thread

Complete dumps:

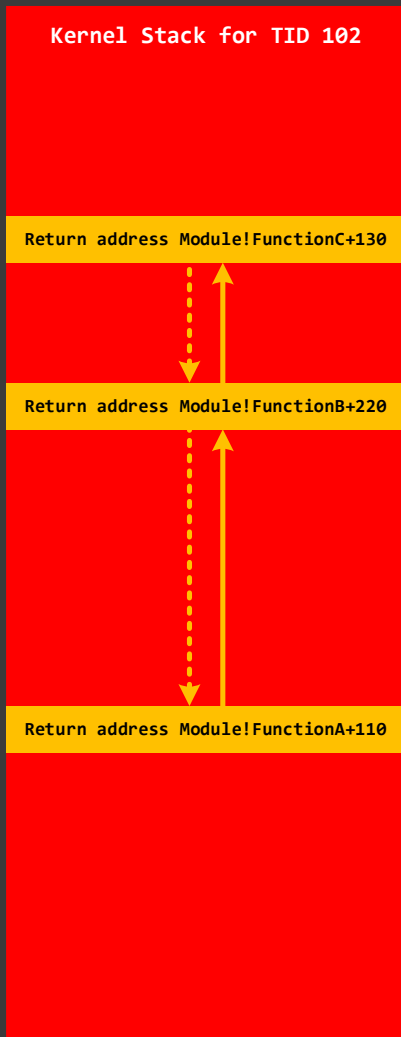
!teb for user space

!thread for kernel space

Data:

dc / dps / dpp / dpa / dpu

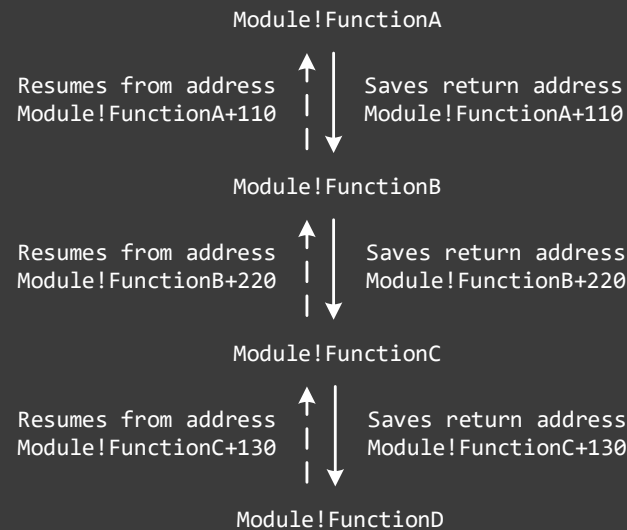
Thread Stack Trace



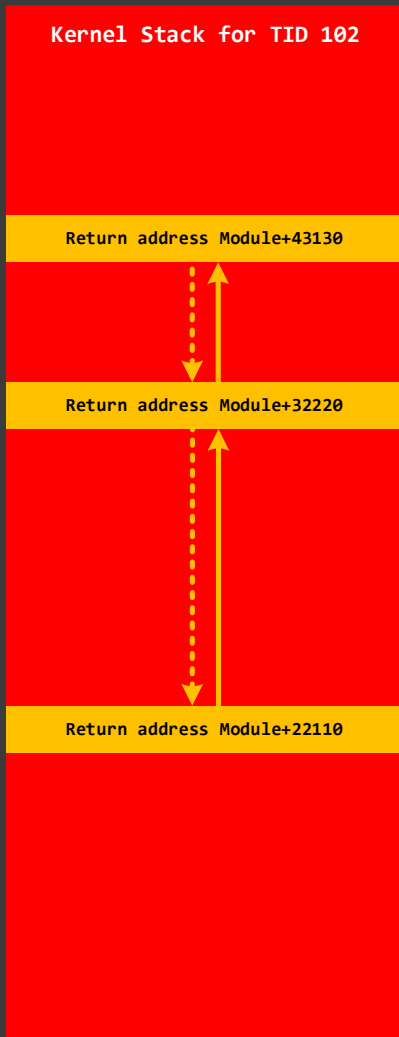
```
FunctionA()  
{  
  ...  
  FunctionB();  
  ...  
}  
FunctionB()  
{  
  ...  
  FunctionC();  
  ...  
}  
FunctionC()  
{  
  ...  
  FunctionD();  
  ...  
}
```

WinDbg Commands

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



Thread Stack Trace (no PDB)

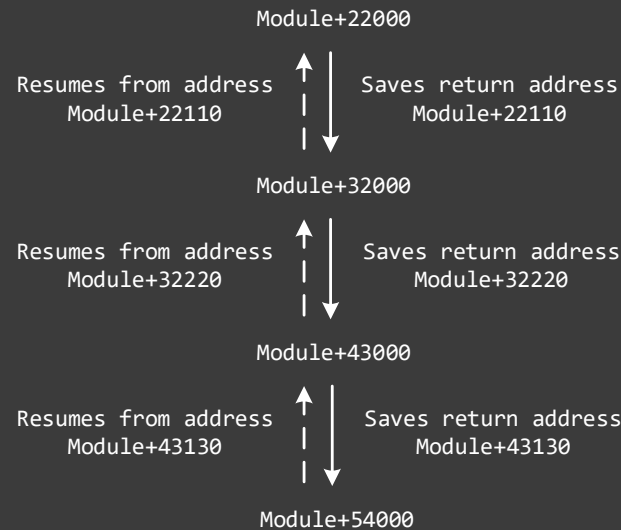


```
FunctionA()
{
  ...
  FunctionB();
  ...
}
FunctionB()
{
  ...
  FunctionC();
  ...
}
FunctionC()
{
  ...
  FunctionD();
  ...
}
```

Symbol file Module.pdb

FunctionA	22000	-	23000
FunctionB	32000	-	33000
FunctionC	43000	-	44000
FunctionD	54000	-	55000

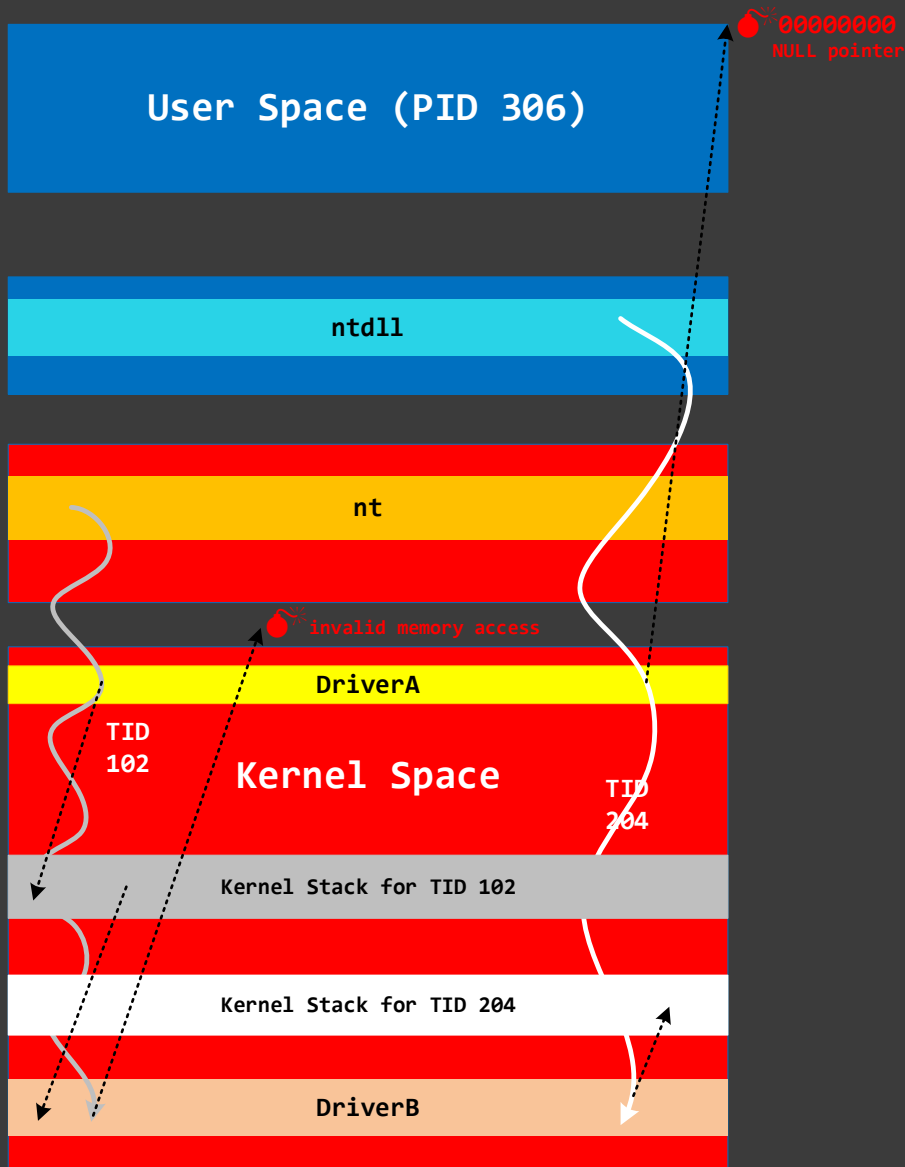
No symbols for Module



WinDbg Commands

```
0: kd> k
Module+0
Module+43130
Module+32220
Module+22110
```

Exceptions (Access Violation)



WinDbg Commands

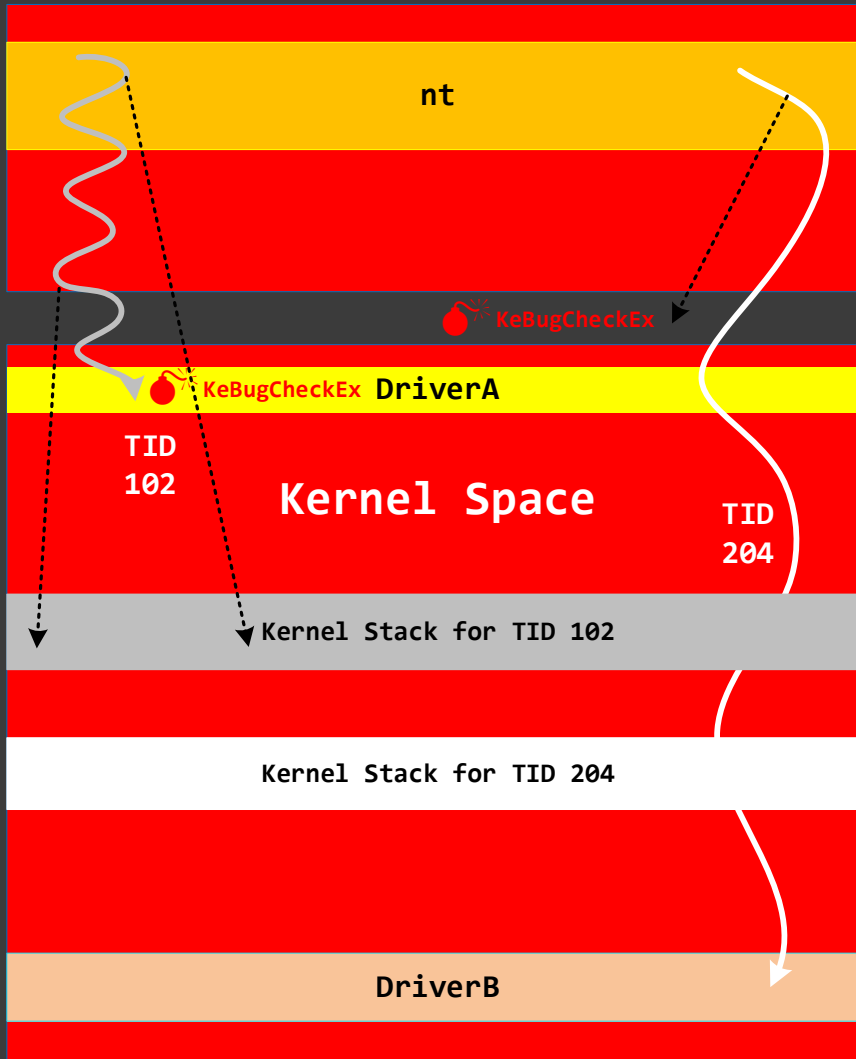
`address=????????`

Set exception context:
`.cxr`

Set trap context:
`.trap`

Check address:
`!pte`

Bugchecks (Runtime)



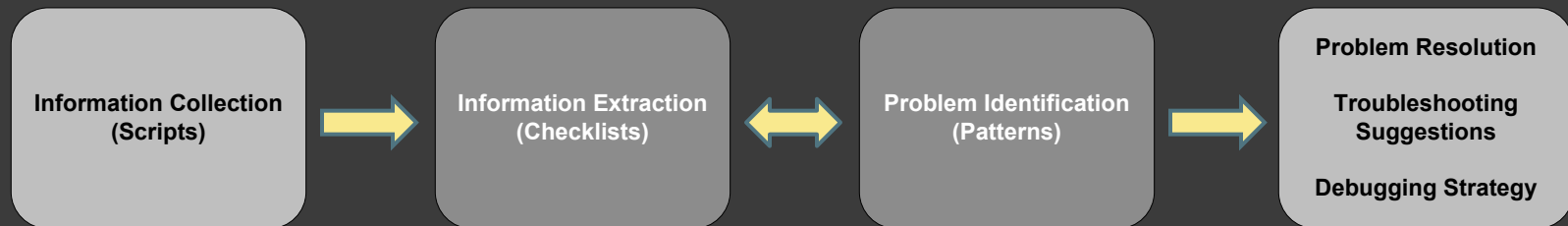
Pattern-Oriented Diagnostic Analysis

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

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

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, macOS, Linux, ...



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

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

Part 2B: 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**

⦿ Next instruction: **RIP**

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

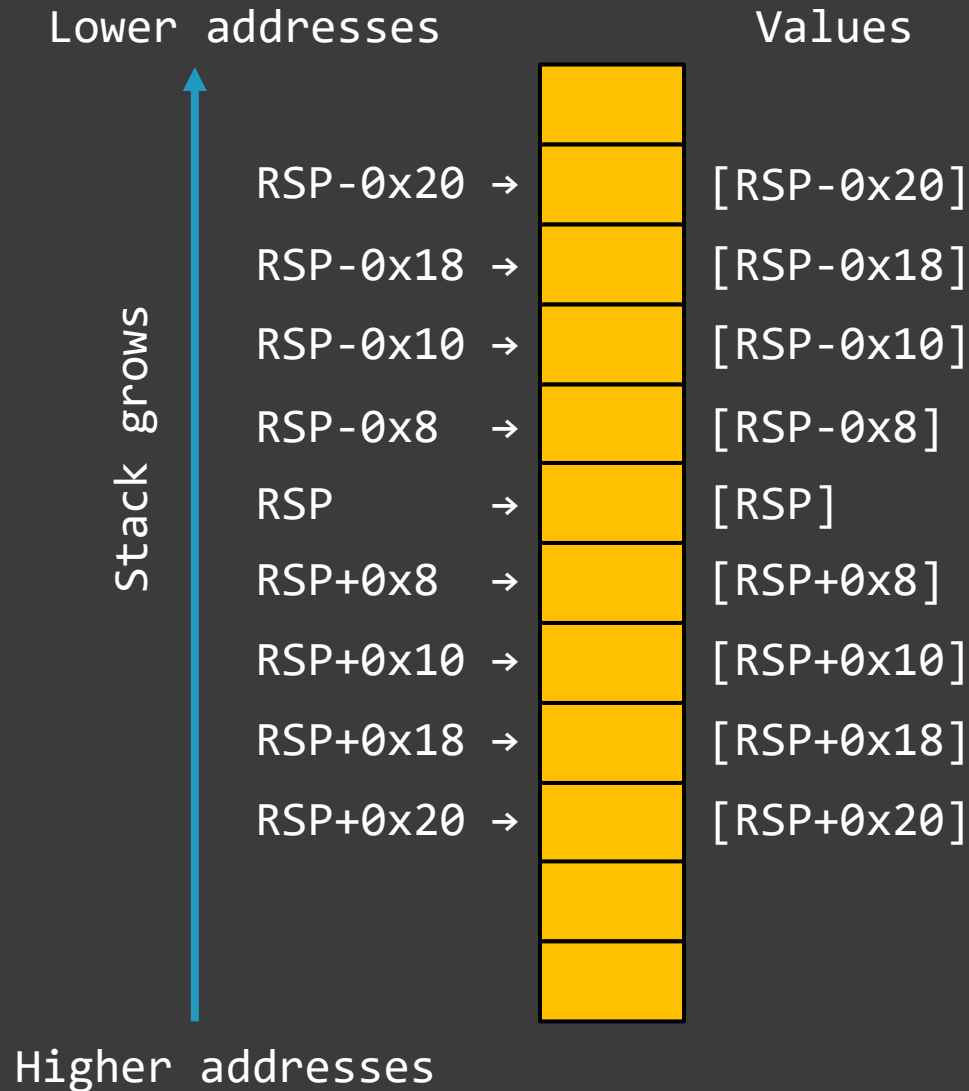
x64 Instructions and Registers

- ◎ Opcode DST, SRC

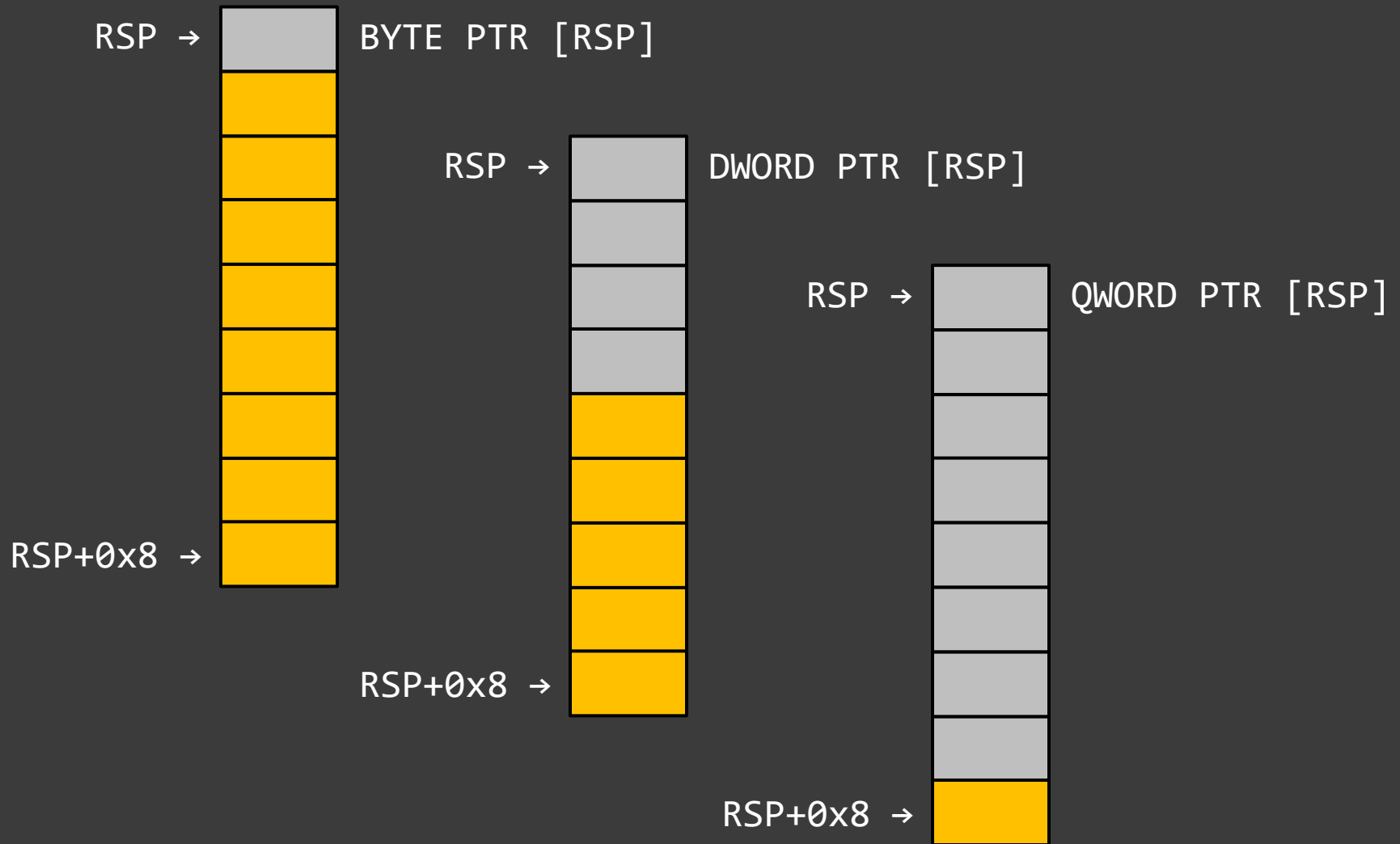
- ◎ Examples:

```
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
```

x64 Stack Addressing



x64 Memory Cell Sizes



x64 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
```

x64 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
```

x64 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)
```

x64 Windows API Parameters

- ⦿ x86: Right to left **PUSH**

Args to Child are 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
...
```

- ⦿ Return value: **EAX/RAX**

Part 2C: ARM64 Disassembly

A64 CPU Registers

- ◎ **X0 – X28**, **W0 – W28**
- ◎ **XIP0 (X16)**, **XIP1 (X17)**
- ◎ Stack: **SP**, **FP (X29)**
- ◎ Next instruction: **PC**
- ◎ Link register: **LR (X30)**
- ◎ Zero register: **XZR**, **WZR**

X 64-bit

W 32-bit

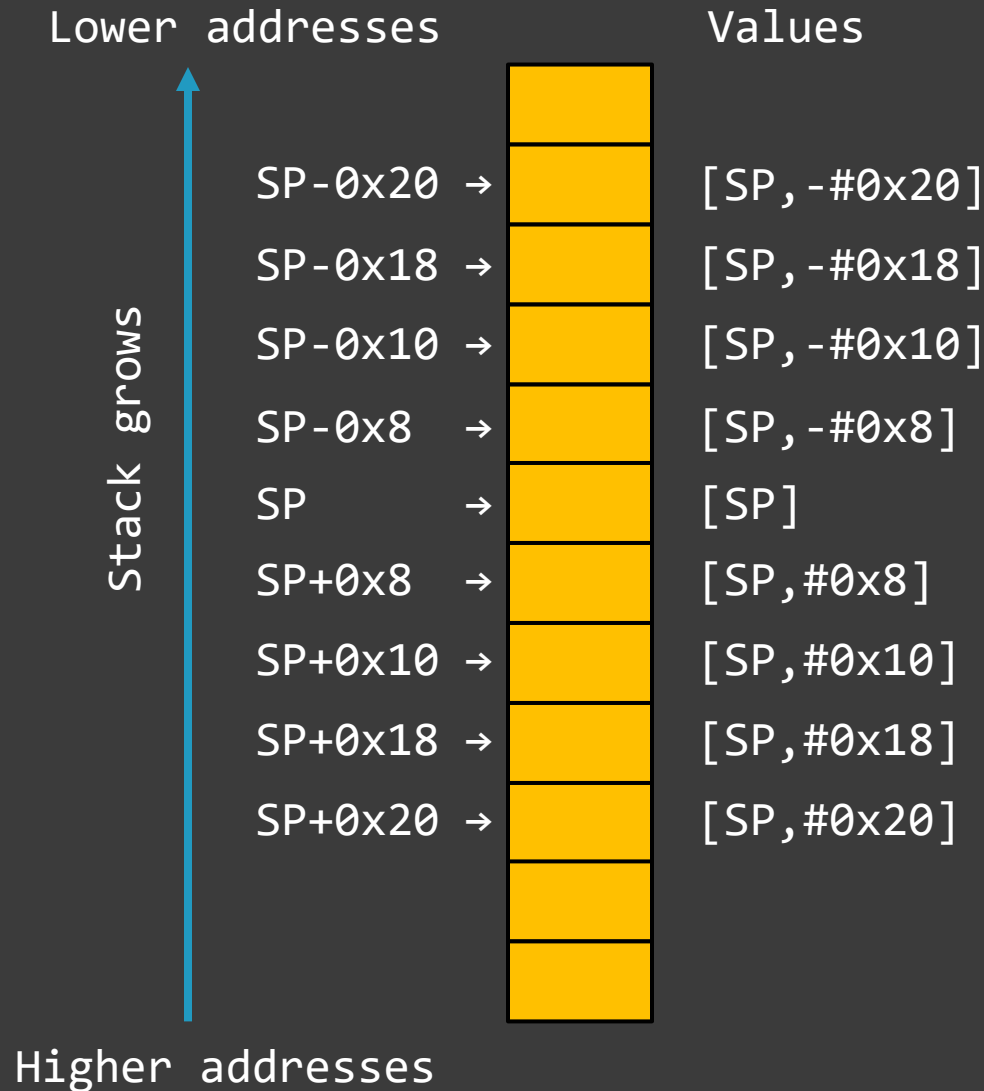
A64 Instructions and Registers

- ◉ Opcode DST, SRC, SRC₂

- ◉ Examples:

```
mov    x0, #16           // X0 ← 16 (0x10)
mov    fp, sp           // FP ← SP
add    x1, x2, #16      // X1 ← X2+16 (0x10)
mul    x1, x2, x3       // X1 ← X2*X3
blr    x8               // X8 already contains
                        // the address of func (&func)
                        // LR ← PC+4; PC ← &func
sub    sp, sp, #48      // SP ← SP-48 (-0x30)
                        // make room for local variables
```

A64 Stack Addressing



A64 Memory Load Instructions

- ◉ Opcode `DST, DST2, [SRC, Offset]`
- ◉ Opcode `DST, DST2, [SRC], Offset // Postincrement`
- ◉ Examples:

```
ldr    x0, [sp]           // X0 ← value at address SP+0
ldr    w0, [sp, #-8]      // W0 ← value at address SP-8
ldp    fp, lr, [sp, #32]  // FP ← value at address SP+32 (0x20)
                          // LR ← value at address SP+40 (0x28)
ldp    fp, lr, [sp], #16  // FP ← value at address SP+0
                          // LR ← value at address SP+8
                          // SP ← SP+16 (0x10)
```

A64 Memory Store Instructions

- ◎ **Opcode** SRC, SRC₂, [DST, Offset]
- ◎ **Opcode** SRC, SRC₂, [DST, Offset]! // Preincrement
- ◎ Examples:

```
str    x0, [sp, #16]           // X0 → value at address SP+16 (0x10)
str    w0, [sp, #-8]           // W0 → value at address SP-8
stp    fp, lr, [sp, #32]       // FP → value at address SP+32 (0x20)
                                           // LR → value at address SP+40 (0x28)
stp    fp, lr, [sp, #-16]!     // SP ← SP-16 (-0x10)
                                           // FP → set value at address SP
                                           // LR → set value at address SP+8
```

A64 Flow Instructions

- ◉ Opcode DST, SRC

- ◉ Examples:

```
adrp x0, 0x420000 // X0 ← 0x420000
```

```
b 00007ff7`5340ee00 // PC ← 00007ff7`5340ee00  
// (goto 00007ff7`5340ee00)
```

```
br xip0 // PC ← the value of XIP0
```

```
00007ff7`53410640: // PC == 00007ff7`53410640
```

```
b1 00007ff7`5340d518 // LR ← PC+4 (00007ff7`53410648)  
// PC ← 00007ff7`5340d518  
// (goto 00007ff7`5340d518)
```

A64 Windows API Parameters

- Left to right via $X0 - X7$, $[SP]$, $[SP+8]$, $[SP+16]$, ...

Args to Child are **not** parameters

WinDbg Commands

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

- Return value: $X0$

Parts 2D–2E: Practice Exercises

Links

- Memory Dumps:

Included in Exercise 0

- Exercise Transcripts:

Included in the book

Exercise 0

- ⦿ **Goal:** Install WinDbg and check that symbols are set up correctly
- ⦿ **Patterns:** Stack Trace; Incorrect Stack Trace
- ⦿ [\AWMDA-Dumps\Exercise-0-Download-Setup-WinDbg.pdf](#)

Kernel Memory Dumps

Exercises K1 – K8

Types of WinDbg Commands

- ⦿ Debugger commands: data from a memory dump (k)
- ⦿ Metacommands: .-commands, debugger control
- ⦿ Extension commands: !-commands from additional loaded modules (!extension_name.command)

WinDbg Commands

`.chain` shows loaded debugger extensions

`.hh` launches the help window

`!extension_name.help` lists extension DLL commands (if available)

Exercise K1

- ◎ **Goal:** Learn how to get various information related to hardware, system, sessions, processes, threads, and modules
- ◎ **Patterns:** Exception Module; NULL Pointer (Data); False Effective Address; Invalid Pointer (General); Virtualized System (WOW64); Stack Trace Collection (Unmanaged Space); Unloaded Module; Not My Version (Software); Not My Version (Hardware); Fault Context
- ◎ [\AWMDA-Dumps\Exercise-K1-Analysis-normal-kernel-dump-x64.pdf](#)

Exercise K2

- ◎ **Goal:** Learn how to check and compare kernel pool usage
- ◎ **Patterns:** Manual Dump (Kernel); Shared Thread; Insufficient Memory (Kernel Pool)
- ◎ [\AWMDA-Dumps\Exercise-K2-Analysis-kernel-dump-leak-x64.pdf](#)

Exercise K3

- ◎ **Goal:** Learn how to recognize pool corruption and check pool data
- ◎ **Patterns:** Dynamic Memory Corruption (Kernel Pool); Regular Data; Execution Residue (Unmanaged Space, Kernel); Exception Stack Trace; Origin Module
- ◎ [\AWMDA-Dumps\Exercise-K3-Analysis-kernel-dump-pool-corruption-x64.pdf](#)

Exercise K4

- ◎ **Goal:** Learn how to check memory access violations, hooked or invalid code, and kernel raw stack
- ◎ **Patterns:** Invalid Pointer (General); Hooked Functions (Kernel Space); Execution Residue (Unmanaged Space, Kernel); Coincidental Symbolic Information; Past Stack Trace; Rough Stack Trace (Unmanaged Space); Effect Component
- ◎ [\AWMDA-Dumps\Exercise-K4-Analysis-kernel-dump-code-corruption-x64.pdf](#)

Exercise K5

- ◎ **Goal:** Learn how to check I/O requests
- ◎ **Patterns:** Blocking File; One-Thread Process
- ◎ [\AWMDA-Dumps\Exercise-K5-Analysis-kernel-dump-hang-io-x64.pdf](#)

Exercise K6

- ◎ **Goal:** Learn how to recognize stack overflow and find its start
- ◎ **Patterns:** Stack Overflow (Kernel Mode); Execution Residue (Unmanaged Space, Kernel)
- ◎ [\AWMDA-Dumps\Exercise-K6-Analysis-kernel-dump-stack-overflow-x64.pdf](#)

Exercise K7

- ◎ **Goal:** Learn how to recognize stack overwrite and reconstruct the stack trace
- ◎ **Patterns:** Truncated Stack Trace; NULL Pointer (Data); Execution Residue (Unmanaged Space, Kernel); Local Buffer Overflow (Kernel Space); Glued Stack Trace
- ◎ [\AWMDA-Dumps\Exercise-K7-Analysis-kernel-dump-stack-overwrite-x64.pdf](#)

Exercise K8

- ◎ **Goal:** Learn how to recognize input threads in kernel space
- ◎ **Patterns:** Dual Stack Trace; Input Thread
- ◎ [\AWMDA-Dumps\Exercise-K8-Analysis-kernel-dump-blocked-service-x64.pdf](#)

BSOD Analysis Pattern Strategy

- [Stack Trace](#)
- [Active Thread](#)
- [Rough Stack Trace](#)
- [Past Stack Trace](#)
- [Execution Residue](#)
- [Historical Information](#)
- [Unloaded Module](#)
- [Black Box](#)
- [Fault Context](#)
- [Multiple Exceptions](#)
- [Hidden Exception](#)
- [Exception Module](#)
- [Wild Pointer](#)
- [DPC Stack Collection](#)
- [Interrupt Stack Collection](#)
- [Dump Context](#)

Pattern Links

[Manual Dump \(Kernel\)](#)

[Virtualized System \(WOW64\)](#)

[Insufficient Memory \(Kernel Pool\)](#)

[Hooked Functions \(Kernel Space\)](#)

[Blocking File](#)

[Past Stack Trace](#)

[Effect Component](#)

[One-Thread Process](#)

[Local Buffer Overflow \(Kernel Space\)](#)

[NULL Pointer \(Code\)](#)

[Dual Stack Trace](#)

[**Input Thread**](#)

[Stack Overflow \(Kernel Mode\)](#)

[Not My Version \(Hardware\)](#)

[Fault Context](#)

[Origin Module](#)

[Invalid Pointer \(General\)](#)

[Stack Trace Collection \(Unmanaged Space\)](#)

[NULL Pointer \(Data\)](#)

[Coincidental Symbolic Information](#)

[Regular Data](#)

[Rough Stack Trace \(Unmanaged Space\)](#)

[False Effective Address](#)

[Shared Thread](#)

[Truncated Stack Trace](#)

[Unloaded Module](#)

[Dynamic Memory Corruption \(Kernel Pool\)](#)

[Execution Residue \(Unmanaged Space, Kernel\)](#)

[Not My Version \(Software\)](#)

[Exception Module](#)

[Exception Stack Trace](#)

[Glued Stack Trace](#)

Additional Pattern Links

[ERESOURCE patterns and case studies](#)

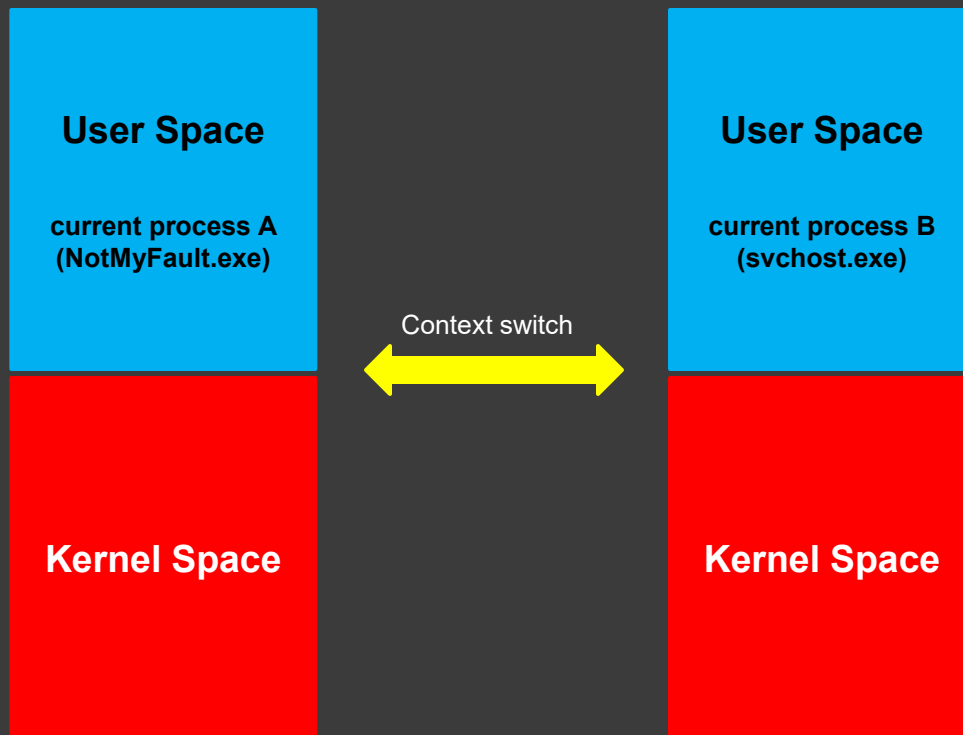
Wait Chain (Executive Resources) pattern is reprinted in this course from Memory Dump Analysis Anthology, Revised Edition, Volume 2, pages 147 – 150

Complete Memory Dumps

Exercises C1 – C6

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
.....
```

Most Frequent Commands

- ◉ **!process 0 3f**

Lists all processes and threads

- ◉ **!process <address> 3f**

Lists the specific process and its threads

- ◉ **!thread <address> 3f**

Lists the specific thread

- ◉ **.thread /r /p <address>**

Sets the current thread and its process

Exercise C1

- ◎ **Goal:** Learn how to get various information related to processes, threads, and modules
- ◎ **Patterns:** Stack Trace Collection (Unmanaged Space); Incorrect Stack Trace
- ◎ [\AWMDA-Dumps\Exercise-C1-Analysis-normal-complete-dump-x64.pdf](#)

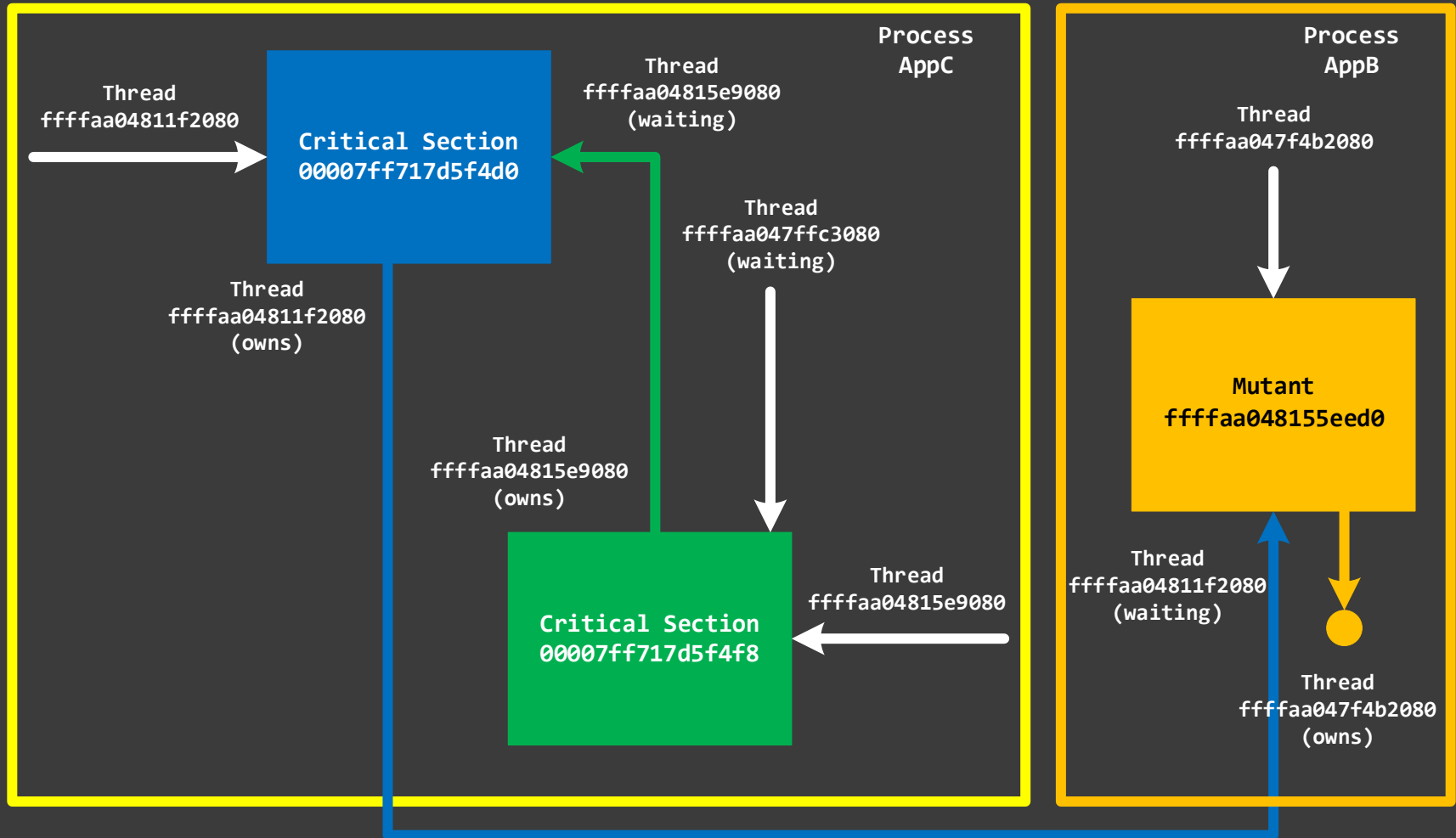
Exercise C2

- ◎ **Goal:** Learn how to recognize various abnormal software behavior patterns
- ◎ **Patterns:** Special Process; Insufficient Memory (Handle Leak); Spiking Thread; Wait Chain (Thread Objects); Dialog Box; Suspended Thread; Wait Chain (Process Objects); Exception Stack Trace; NULL Pointer (Data)
- ◎ [\AWMDA-Dumps\Exercise-C2-Analysis-problem-complete-dump-x64.pdf](#)

Exercise C3

- ◎ **Goal:** Learn how to recognize various abnormal software behavior patterns
- ◎ **Patterns:** Stack Trace Collection (Unmanaged Space); Message Box; Wait Chain (Critical Sections); Wait Chain (Mutex Objects)
- ◎ [\AWMDA-Dumps\Exercise-C3-Analysis-problem-complete-dump-x64.pdf](#)

Wait Chain



Exercise C4

- ◎ **Goal:** Learn how to recognize various abnormal software behavior patterns in x64 memory dumps
- ◎ **Patterns:** Virtualized Process (WOW64); Message Box; Wait Chain (ALPC); Frozen Process
- ◎ [\AWMDA-Dumps\Exercise-C4-Analysis-problem-complete-dump-x64.pdf](#)

Exercise C5

- ◎ **Goal:** Learn how to recognize input threads in kernel space
- ◎ **Patterns:** Input Thread; Message Box
- ◎ [\AWMDA-Dumps\Exercise-C5-Analysis-complete-dump-blocked-service-x64.pdf](#)

Exercise C6

- ◎ **Goal:** Learn how to generate and analyze ARM64 complete memory dumps
- ◎ **Patterns:** Manual Dump (Kernel); Interrupt Stack; Virtualized Process (ARM64EC); ISA-Specific Code; Encoded Pointer
- ◎ [\AWMDA-Dumps\Exercise-C6-Generation-analysis-complete-dump-ARM64.pdf](#)

Pattern Links

[Special Process](#)

[Spiking Thread](#)

[Message Box](#)

[Exception Stack Trace](#)

[Virtualized Process \(WOW64\)](#)

[Incorrect Stack Trace](#)

[Dialog Box](#)

[Frozen Process](#)

[Virtualized Process \(ARM64EC\)](#)

[ISA-Specific Code](#)

[Interrupt Stack](#)

[Insufficient Memory \(Handle Leak\)](#)

[Stack Trace Collection \(Unmanaged Space\)](#)

[Wait Chain \(Critical Sections\)](#)

[Wait Chain \(Thread Objects\)](#)

[Wait Chain \(LPC/ALPC\)](#)

[Wait Chain \(Process Objects\)](#)

[Suspended Thread](#)

[**Input Thread**](#)

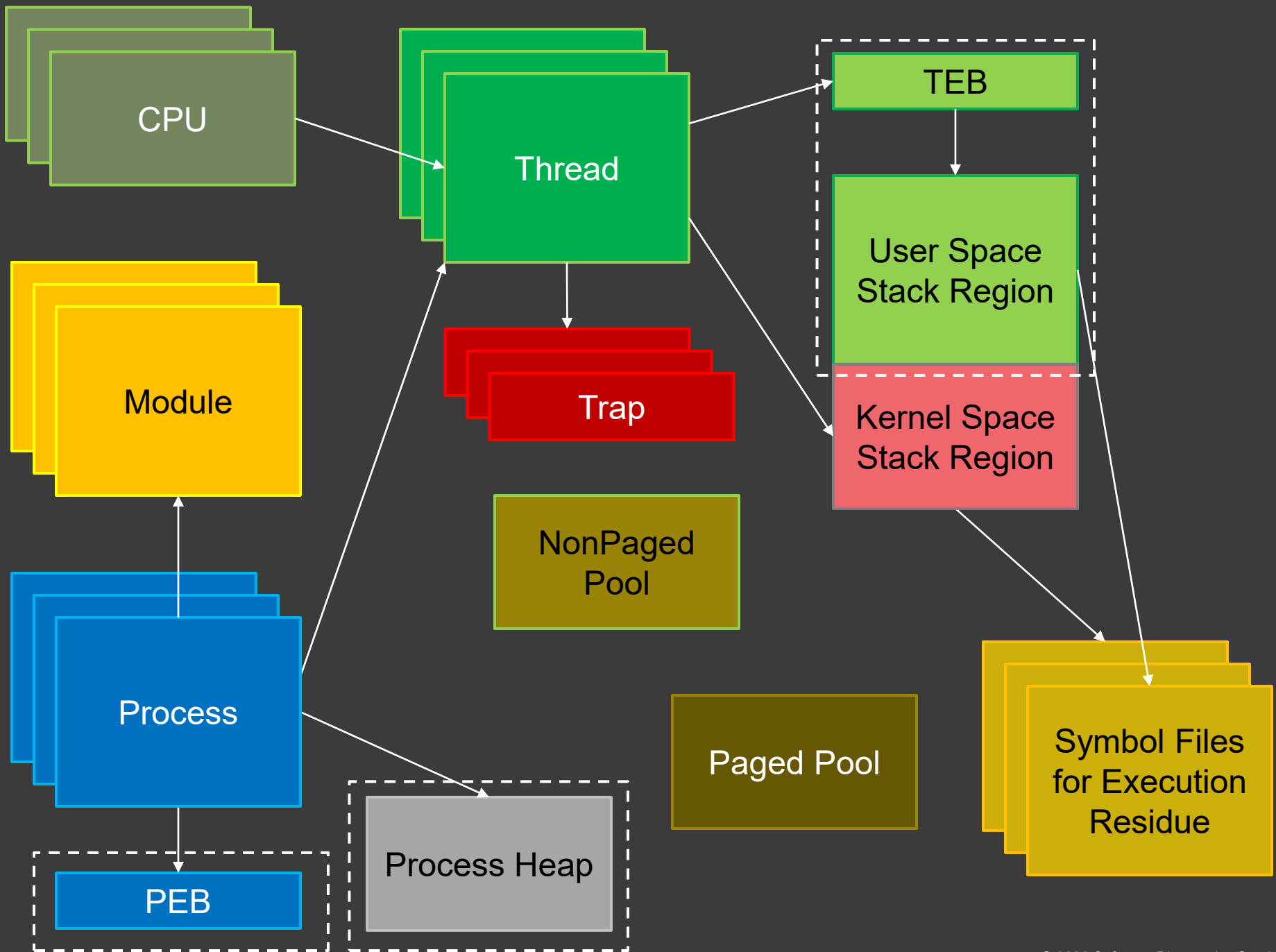
[Encoded Pointer](#)

[Manual Dump \(Kernel\)](#)

Common Mistakes

- ⦿ Not switching to the appropriate context
- ⦿ Not looking at full stack traces
- ⦿ Not looking at all stack traces
- ⦿ Not using checklists
- ⦿ Not looking past the first found evidence
- ⦿ Not listing both x86 and x64 stack traces

Part 2F: Windows Internals



Part 2G: Collection Methods

Kernel/Complete Dump Setup

- Control Panel

View advanced system settings \ Start-up and Recovery

Disable automatic deletion

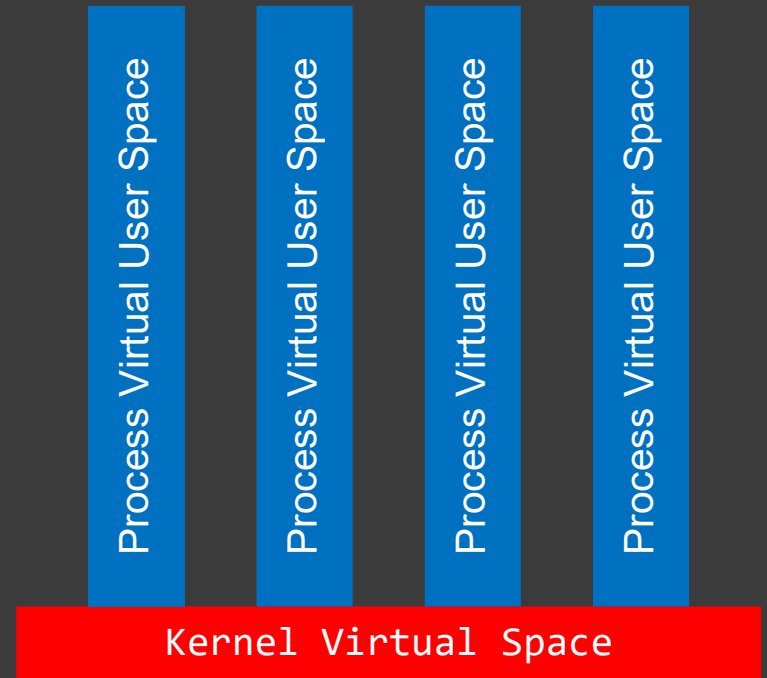
- Page file size (and free disk space) should be greater than the amount of physical memory by 260 MB
- [How to determine the appropriate page file size for 64-bit versions of Windows](#)
- [Configure memory dump files for Server Core installation](#)
- [Active memory dump](#)

Complete Dump Generation

- ◎ Generate a kernel or complete crash dump
- ◎ Forcing a system crash from the keyboard
- ◎ Tools: NotMyFault
- ◎ VMware memory snapshot + vmss2core

Fiber Bundle Memory Dump

- ⦿ When a complete memory dump is not an option due to its size and other concerns
- ⦿ Save full process memory dumps of interest and then trigger a kernel memory dump



Kernel Minidumps

Part 2H: Reading

Memory Dump Analysis Anthology, Revised Edition, Volume 1
pages 43 – 67

Reprinted in this course

Based on: [Kernel Minidump Analysis](#)

Pattern Classification

[Space/Mode](#)

[Hookware](#)

[DLL Link Patterns](#)

[Contention Patterns](#)

[Stack Trace Patterns](#)

[Exception Patterns](#)

[Module Patterns](#)

[Thread Patterns](#)

[Dynamic Memory Corruption Patterns](#)

[.NET / CLR / Managed Space Patterns](#)

[Falsity and Coincidence Patterns](#)

[Hidden Artifact Patterns](#)

[Frame Patterns](#)

[Region Patterns](#)

[Memory dump type](#)

[Wait Chain Patterns](#)

[Insufficient Memory Patterns](#)

[Stack Overflow Patterns](#)

[Symbol Patterns](#)

[Meta-Memory Dump Patterns](#)

[Optimization Patterns](#)

[Process Patterns](#)

[Deadlock and Livelock Patterns](#)

[Executive Resource Patterns](#)

[RPC, LPC and ALPC Patterns](#)

[Pointer Patterns](#)

[CPU Consumption Patterns](#)

[Collection Patterns](#)

Pattern Case Studies

More than 70 multiple pattern case studies:

<https://www.dumpanalysis.org/blog/index.php/pattern-cooperation/>

Pattern Interaction chapters in
Memory Dump Analysis Anthology

Additional Resources

- WinDbg Help / WinDbg.org (quick links)
- DumpAnalysis.org / SoftwareDiagnostics.Institute / PatternDiagnostics.com
- Debugging.TV / YouTube.com/DebuggingTV / YouTube.com/PatternDiagnostics
- Windows Internals, 6th ed. (Chapter 14. Crash Dump Analysis), 7th ed.
- Windows Kernel Programming, 2nd ed.
- Advanced Windows Debugging
- Inside Windows Debugging
- [Principles of Memory Dump Analysis](#)
- [Fundamentals of Physical Memory Analysis: Anniversary Edition](#)
- [Encyclopedia of Crash Dump Analysis Patterns, 3rd edition](#)
- [Memory Dump Analysis Anthology \(Diagnomicon\)](#)



Further Training Courses

- [Accelerated Windows Memory Dump Analysis, 7th Edition, Part 1](#)
- [Practical Foundations of Windows Debugging, Disassembling, Reversing, 2nd Edition \(x86 and x64\)](#)
- [Practical Foundations of Windows Debugging, Disassembling, Reversing, 3rd Edition \(x64 only\)](#)
- [Practical Foundations of ARM64 Windows Debugging, Disassembling, Reversing](#)
- [Advanced Windows Memory Dump Analysis with Data Structures, 5th Edition](#)
- [Accelerated .NET Memory Dump Analysis, 7th Edition](#)
- [Accelerated Windows Malware Analysis with Memory Dumps, 3rd Edition](#)
- [Accelerated Disassembly, Reconstruction and Reversing, 3rd Edition](#)
- [Accelerated Windows Debugging⁴, 4th Edition](#)
- [Extended Windows Memory Dump Analysis, 2nd Edition](#)
- [Accelerated Windows API for Software Diagnostics, 2nd Edition](#)
- [Accelerated Rust Windows Memory Dump Analysis](#)

Q&A

Please send your feedback using the contact form on PatternDiagnostics.com

Thank you for attendance!