



Windows Memory Dump Analysis

Accelerated

Version 7

Part 1: Process User Space

Dmitry Vostokov
Software Diagnostics Services

New

- ◉ Additional review of fundamentals
- ◉ 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!

- ◉ Defect mechanism patterns
- ◉ Additional memory analysis patterns
- ◉ Windows ARM64
- ◉ Troubleshooting symbol problems

Prerequisites

WinDbg Commands

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

Basic Windows troubleshooting

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

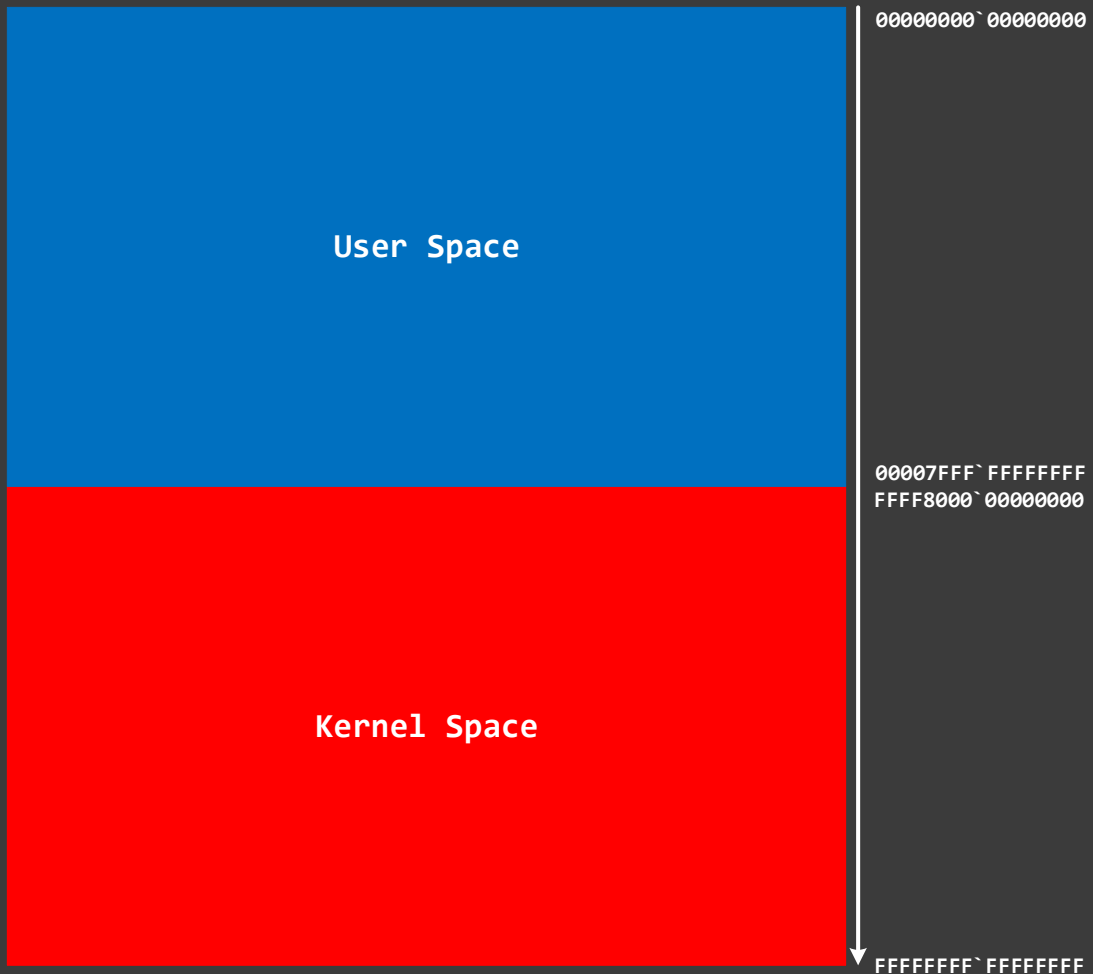
Coverage (Part 1)

- ⦿ Windows 10 and 11
- ⦿ Both x64* and x86 code, WOW64
- ⦿ ARM64, ARM64EC, and CHPE code
- ⦿ x64 and ARM64 disassembly review
- ⦿ Preliminary .NET analysis
- ⦿ Process memory dumps
- ⦿ Crashes, hangs, memory and handle leaks, CPU spikes

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

Part 1A: Fundamentals

Process Space (x64, ARM64)



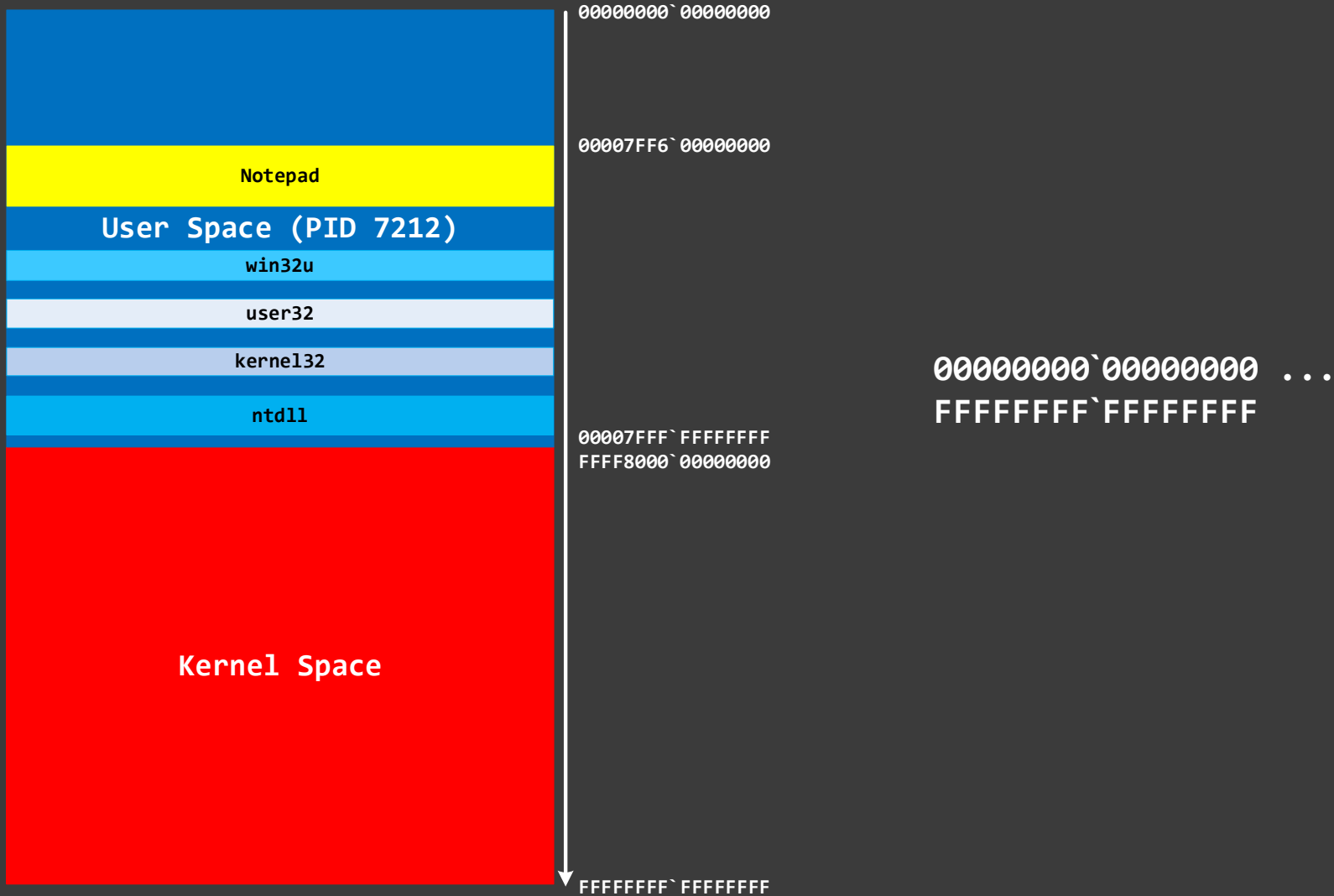
WinDbg Commands

!address provides address range description

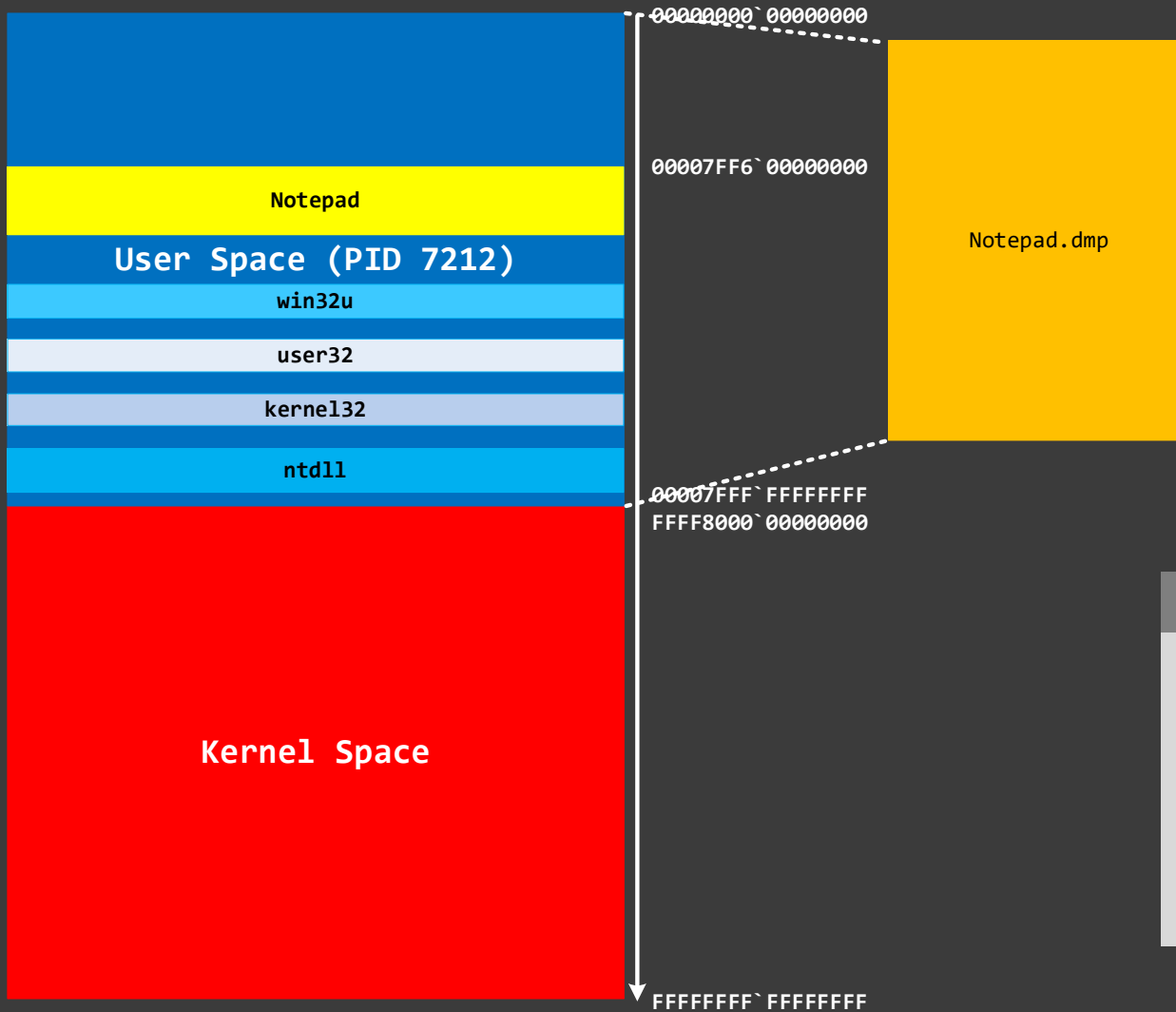
Application/Process/Module



Process Virtual Space (x64, A64)



Process Memory Dump (x64, A64)



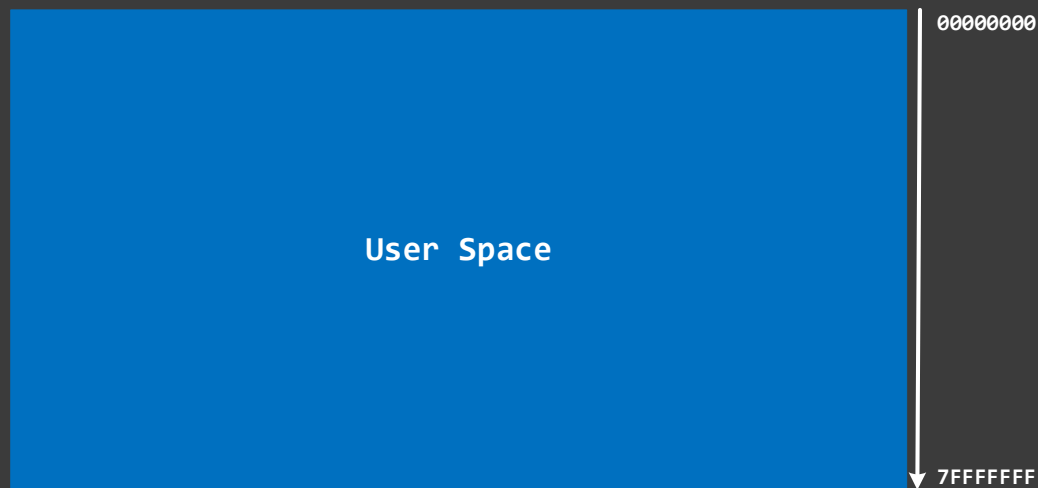
WinDbg Commands

!mv command lists modules and their description

.effmach switches between architectures (AMD64 and ARM64EC)

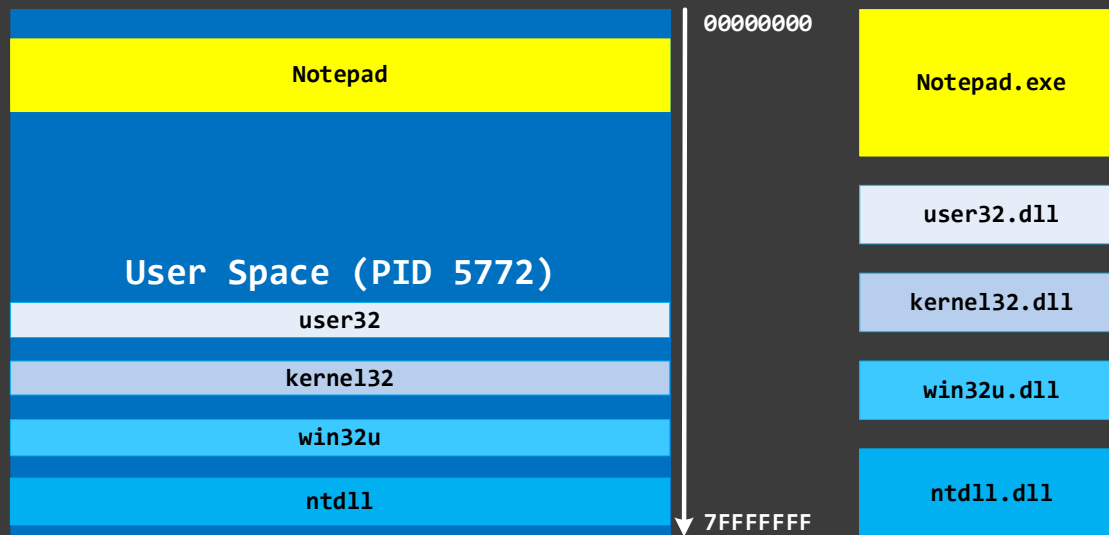
Process Space (x86)

View from a 32-bit memory dump



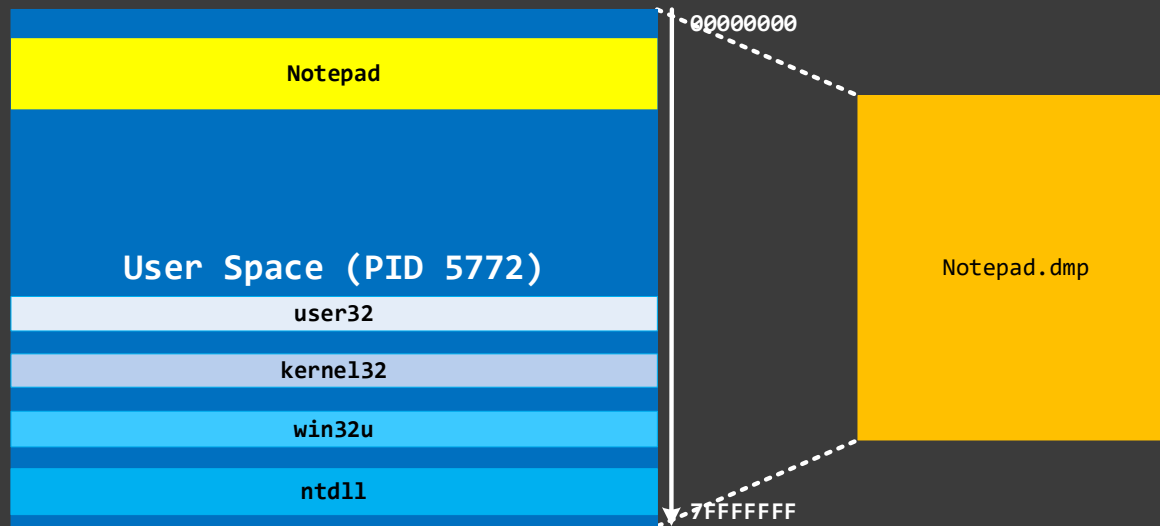
Application/Process/Module (x86)

View from a 32-bit memory dump



Process Memory Dump (x86)

32-bit memory dump of a 32-bit process



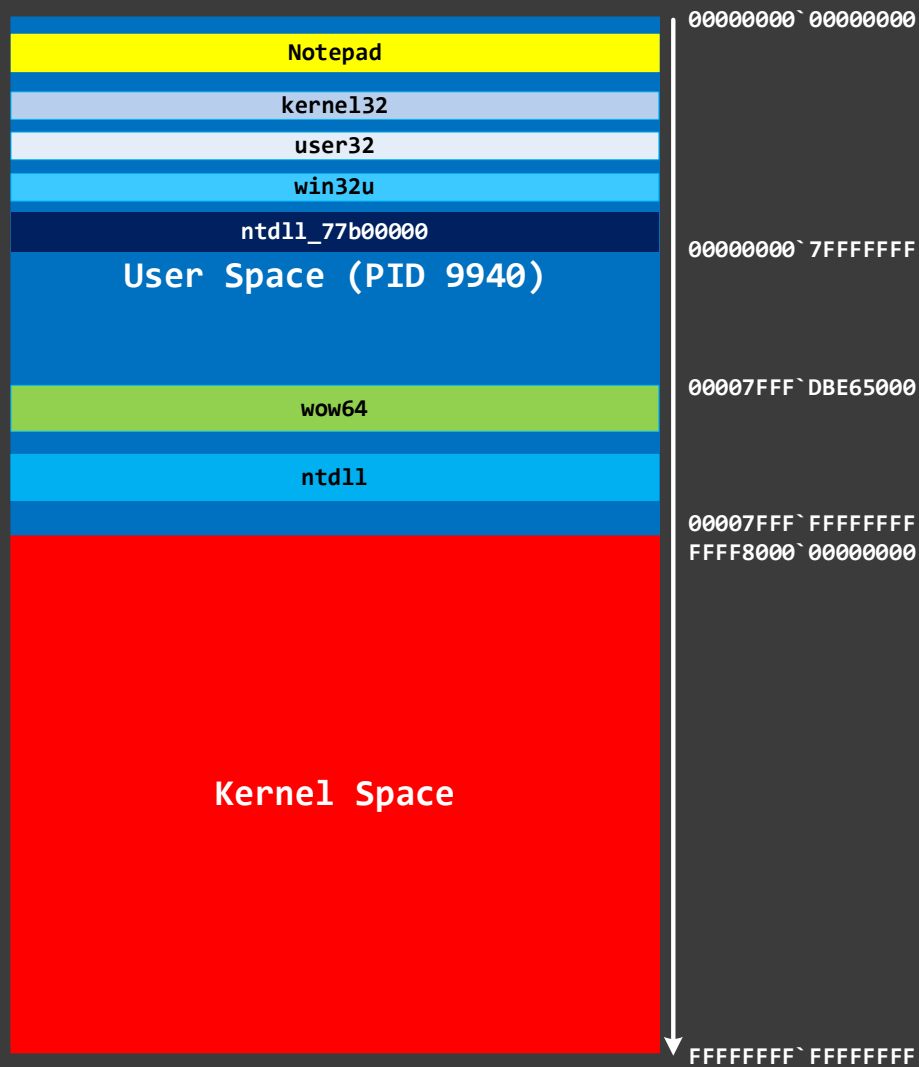
WinDbg Commands

!mv command lists modules and their description

.effmach switches between architectures (x86 and CHPE)

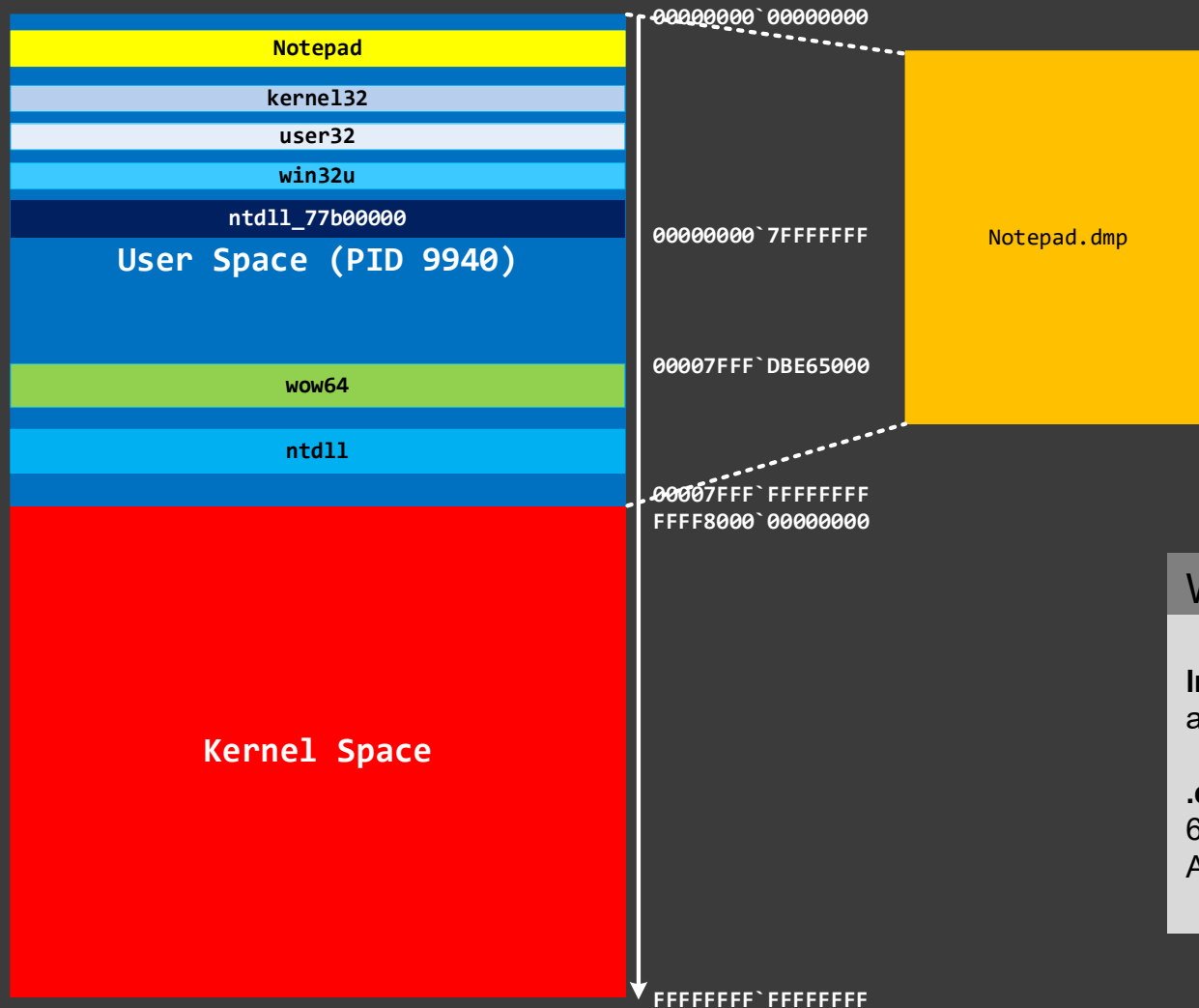
Process Virtual Space (WOW64)

Emulation of a 32-bit process



Process Memory Dump (WOW64)

64-bit memory dump of a 32-bit process

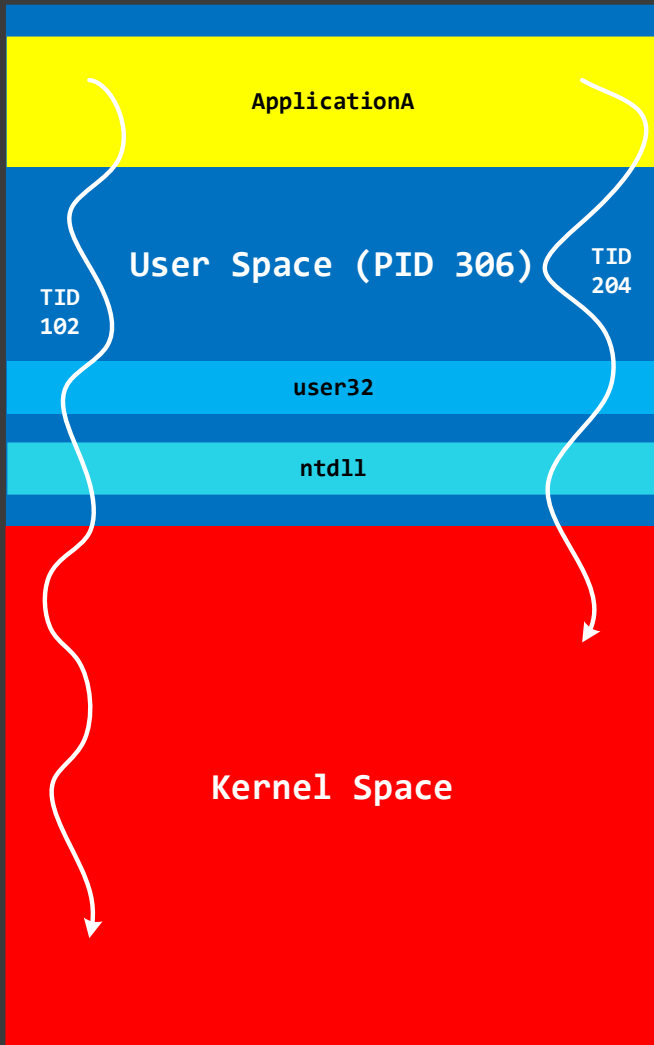


WinDbg Commands

!mv command lists modules and their description

.effmach switches between 64-bit architectures (AMD64 or ARM64) and x86

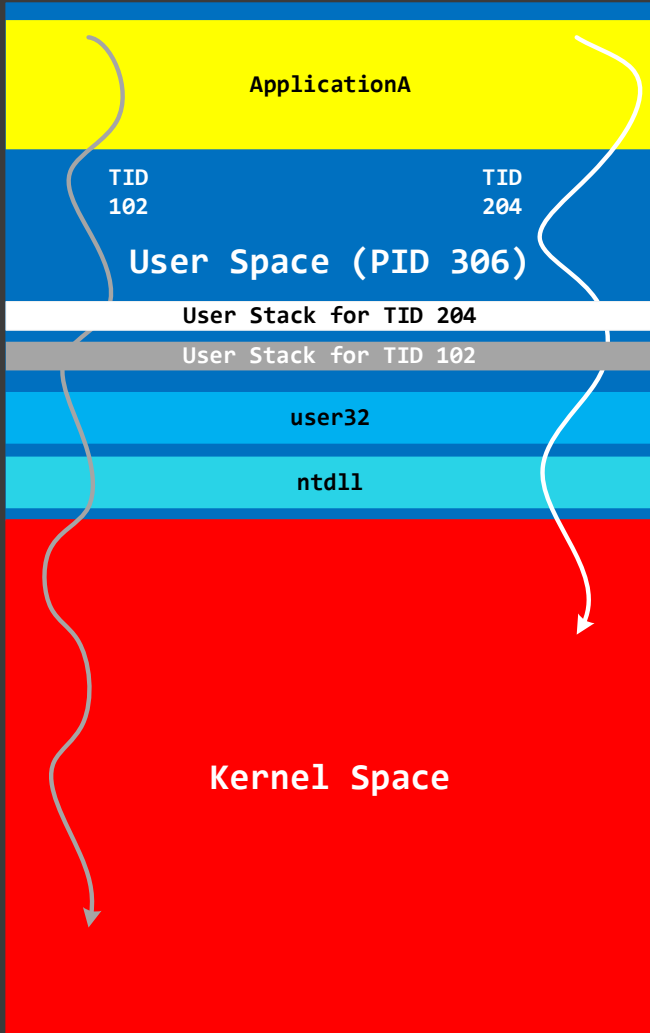
Process Threads



WinDbg Commands

Process dumps:
~<n>s switches between threads

Thread Stack Raw Data



WinDbg Commands

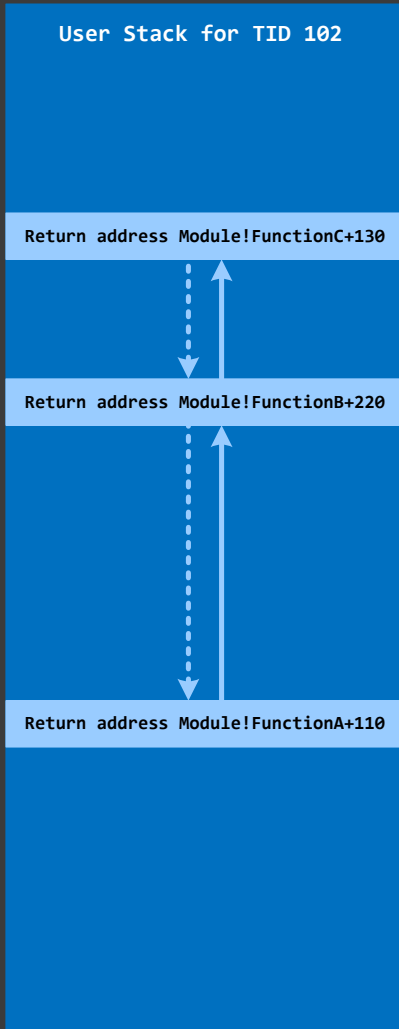
Process dumps:

!teb

Data:

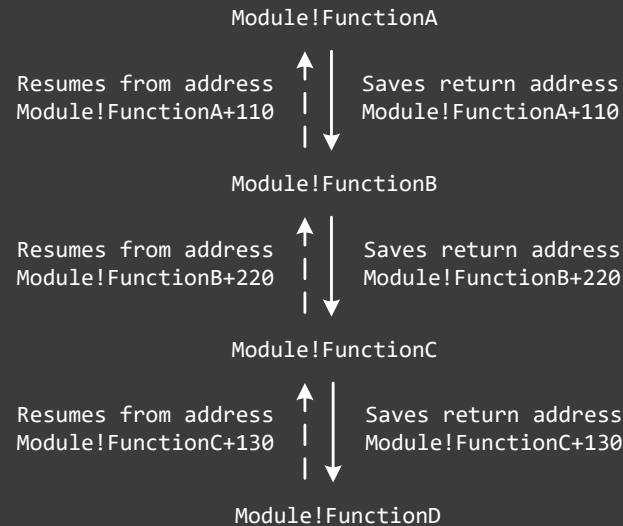
dc / dps / dpp / dpa / dpu

Thread Stack Trace

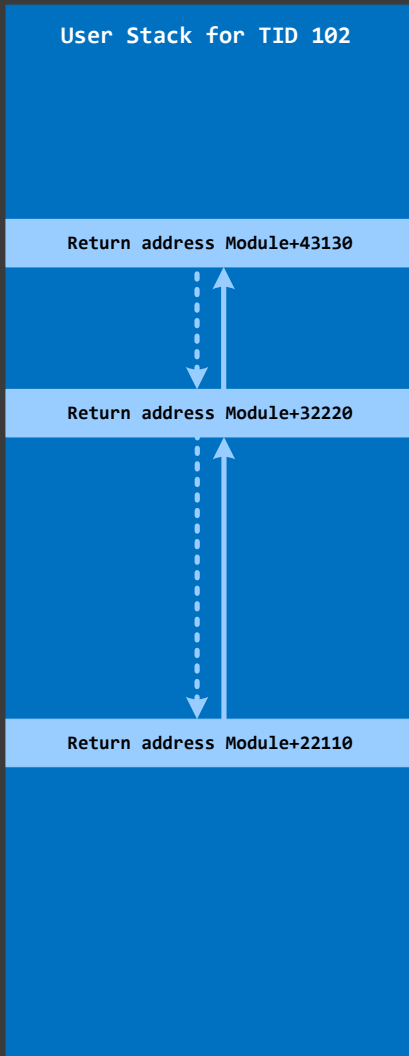


```
FunctionA()  
{  
  ...  
  FunctionB();  
  ...  
}  
FunctionB()  
{  
  ...  
  FunctionC();  
  ...  
}  
FunctionC()  
{  
  ...  
  FunctionD();  
  ...  
}
```

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



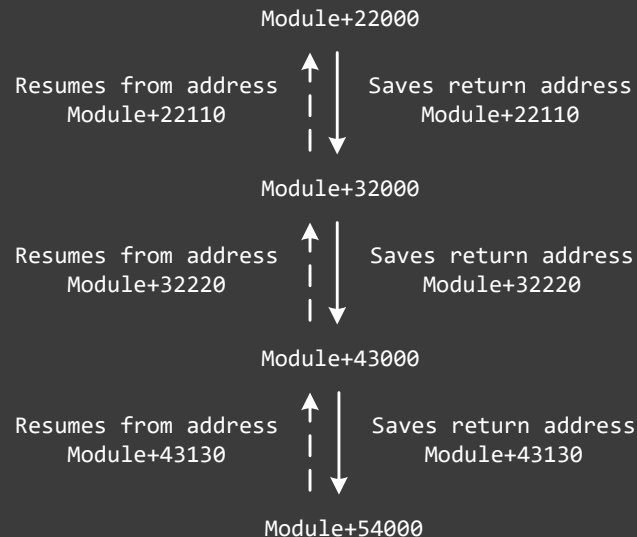
Thread Stack Trace (no PDB)



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

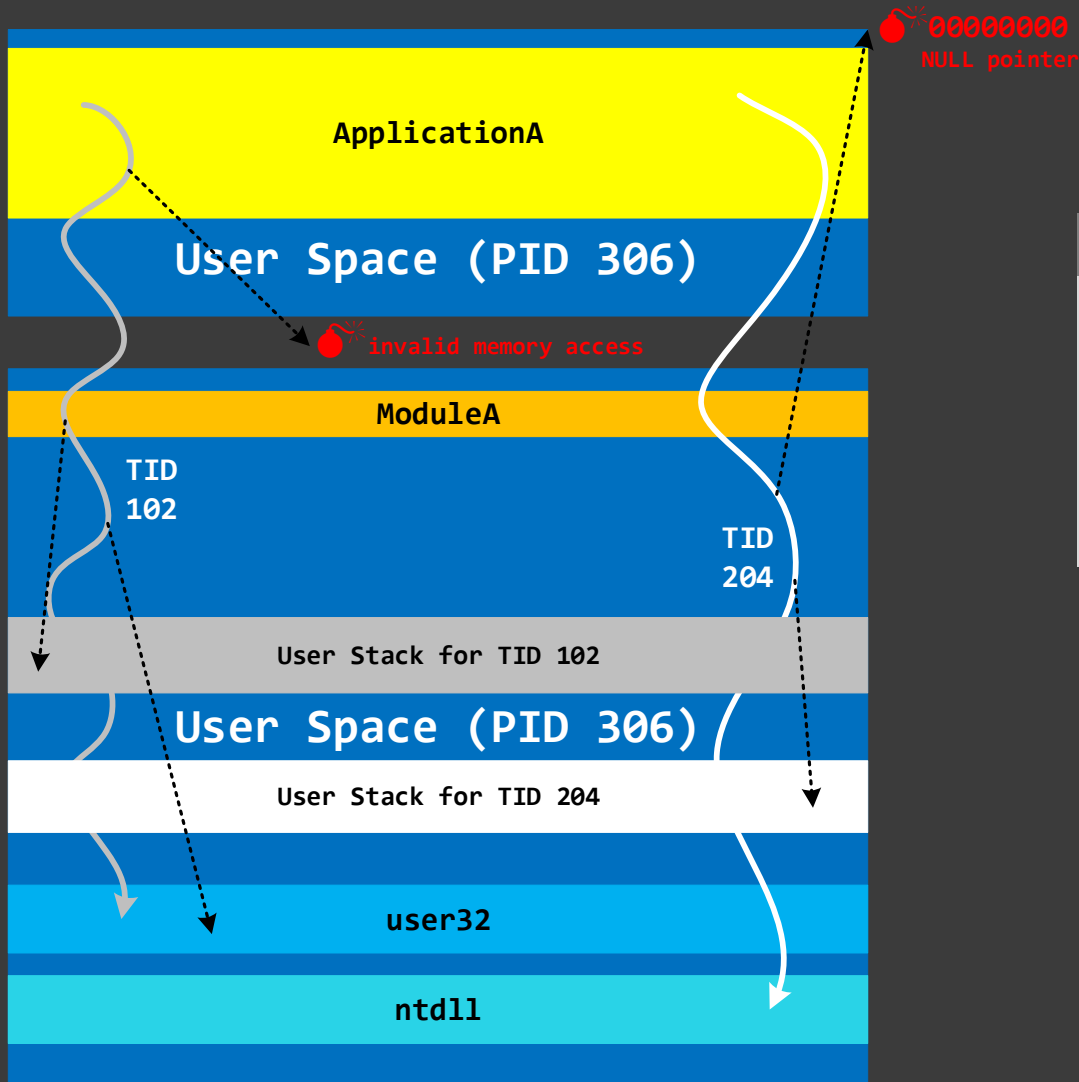
```
Symbol file Module.pdb  
  
FunctionA 22000 - 23000  
FunctionB 32000 - 33000  
FunctionC 43000 - 44000  
FunctionD 54000 - 55000
```

No symbols for Module



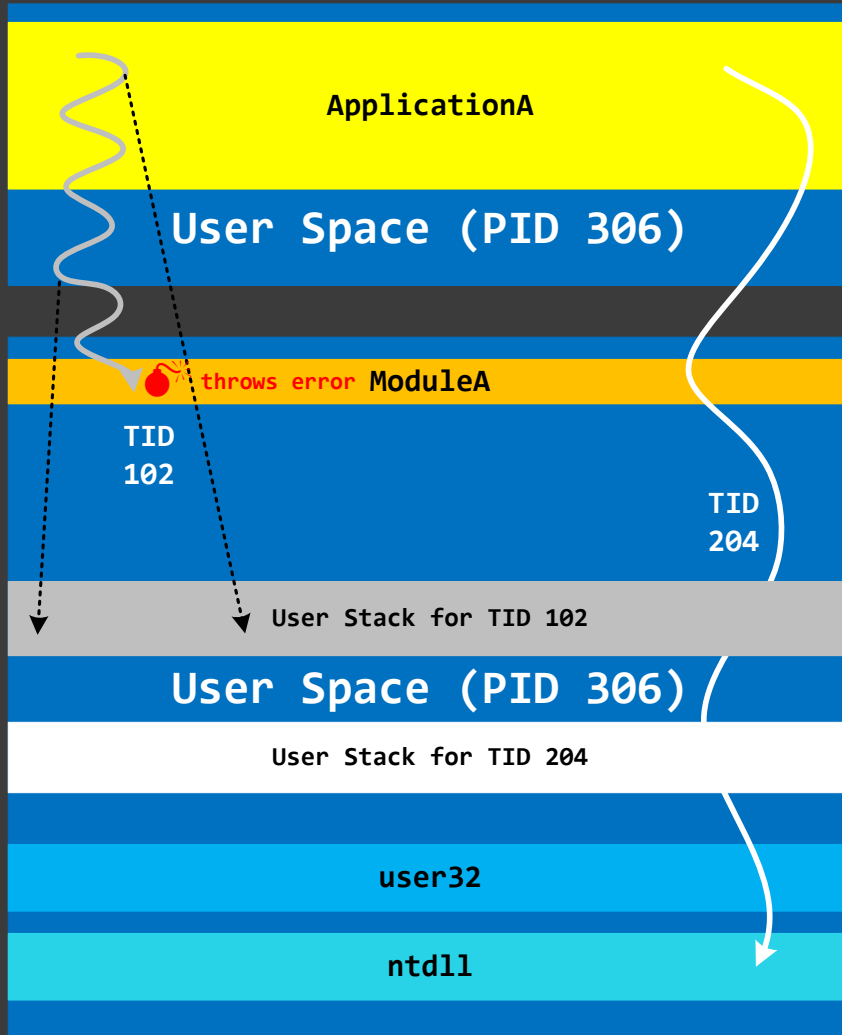
```
WinDbg Commands  
  
0:000> k  
Module+0  
Module+43130  
Module+32220  
Module+22110
```

Exceptions (Access Violation)



```
WinDbg Commands  
  
address=?????????  
  
Set exception context  
(process dump):  
.cxr
```

Exceptions (Runtime)



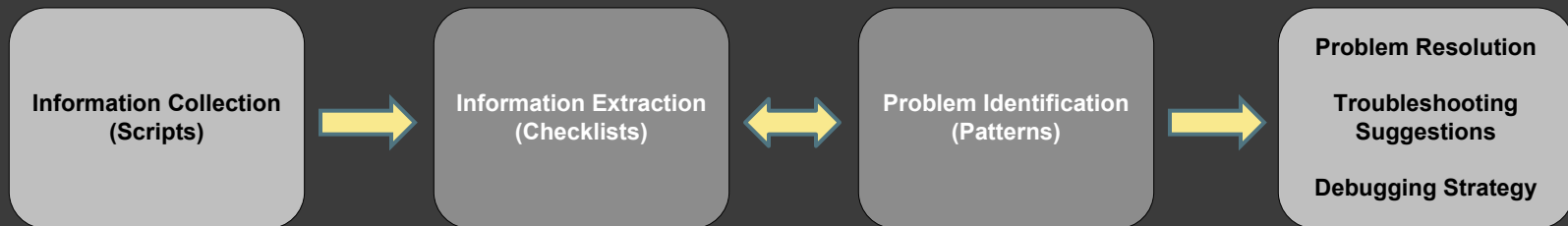
Pattern-Oriented Diagnostic Analysis

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/crash-dump-analysis-patterns/>

Part 1B: 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)**

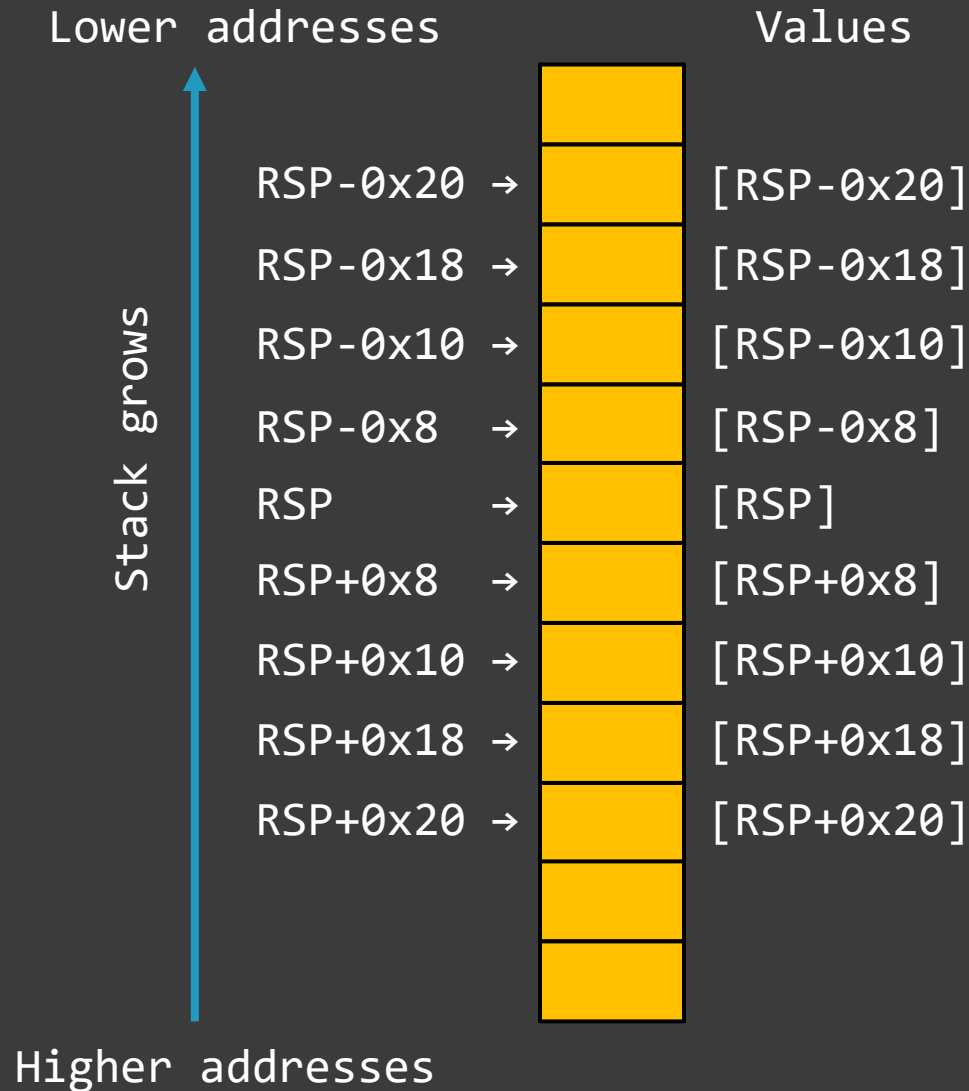
x64 Instructions and Registers

- ◎ Opcode DST, SRC

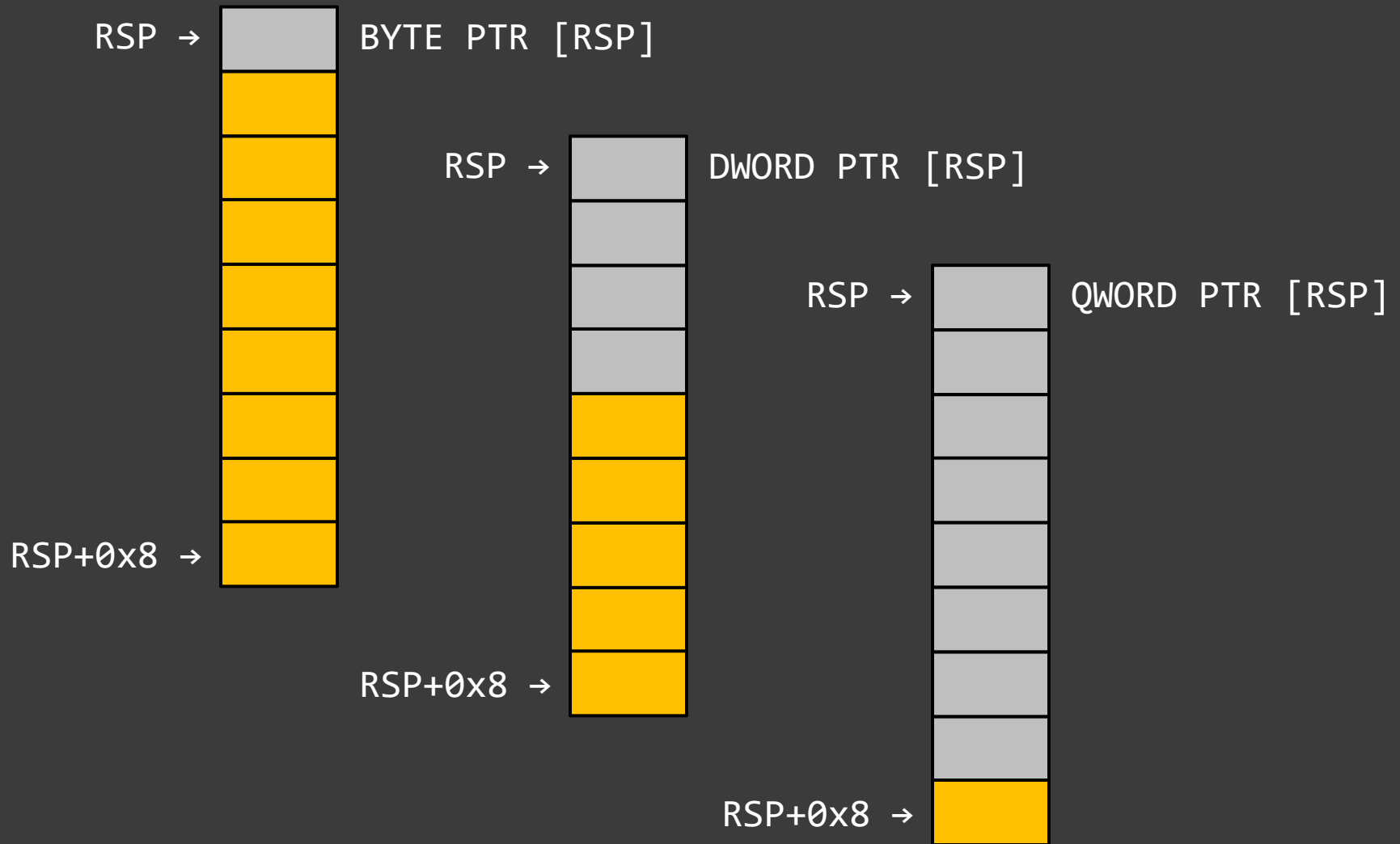
- ◎ Examples:

```
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 1C: 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

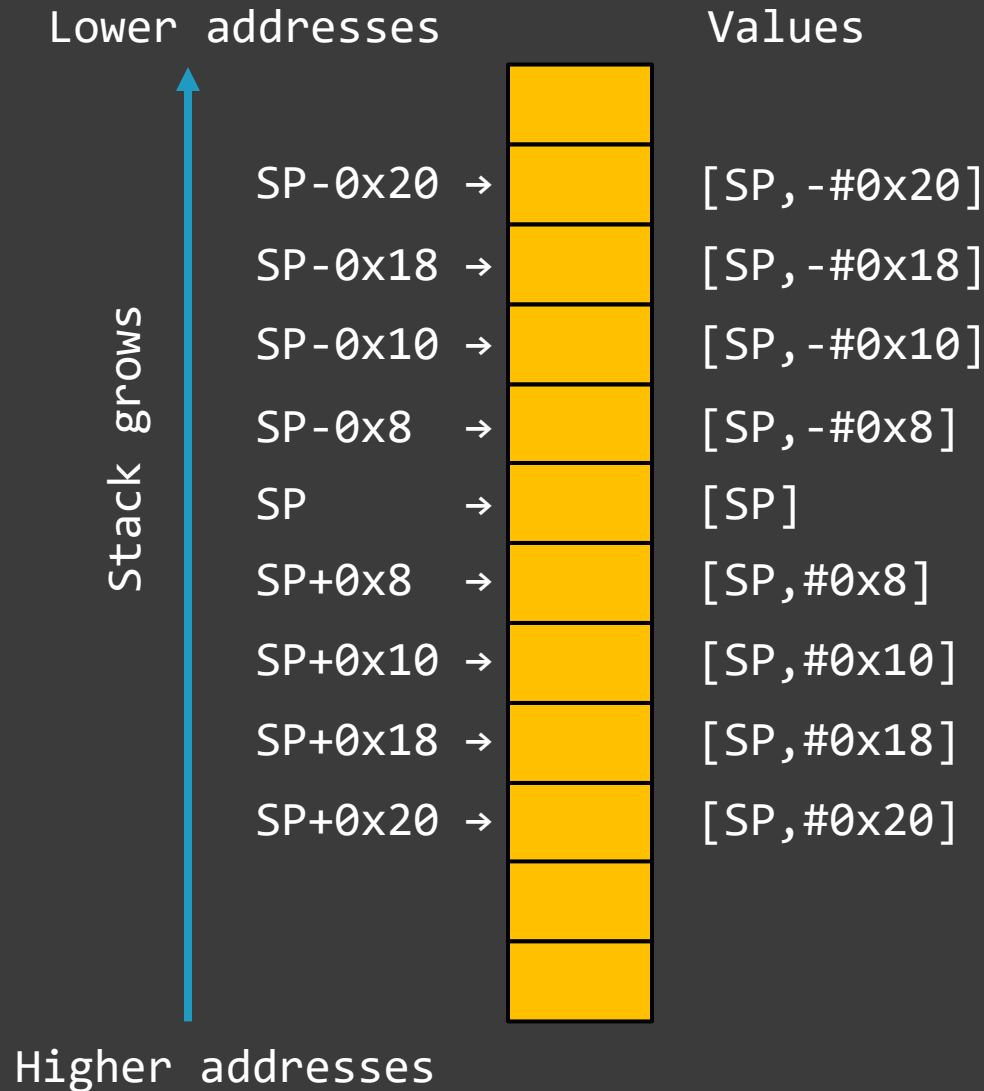
A64 Instructions and Registers

- ◉ 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: $X0$

Part 1D: Practice Exercises

Links

- Memory Dumps:

Included in Exercise 0

- Exercise Transcripts:

Included in this 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](#)

Process Memory Dumps

Exercises P1 – P25

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 P1

- ◎ **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); Stack Trace; Not My Version (Software); Environment Hint; Unknown Component; Historical Information
- ◎ [\AWMDA-Dumps\Exercise-P1-Analysis-normal-process-dump-wordpad-x64.pdf](#)

Exercise P2

- ◎ **Goal:** Repeat exercise P1 using the 32-bit memory dump of an x86 WordPad process running on x64 Windows
- ◎ [\AWMDA-Dumps\Exercise-P2-Analysis-normal-process-dump-wordpad-32-x64.pdf](#)

Exercise P3

- ◎ **Goal:** Learn how to list stack traces, check their correctness, perform default analysis, list modules, check their version information, and check thread age and CPU consumption
- ◎ **Patterns:** Stack Trace Collection (Unmanaged Space); Stack Trace Motif
- ◎ [\AWMDA-Dumps\Exercise-P3-Analysis-normal-process-dump-msedge-x64.pdf](#)

Exercise P4

- ◎ **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)
- ◎ [\AWMDA-Dumps\Exercise-P4-Analysis-process-dump-AppK-x64-no-symbols.pdf](#)

Exercise P5

- ◎ **Goal:** Learn how to load application symbols
- ◎ [\AWMDA-Dumps\Exercise-P5-Analysis-process-dump-AppK-x64-with-symbols.pdf](#)

Exercise P6

- ◎ **Goal:** Learn how to recognize heap corruption, dump memory contents, follow critical section wait chains, and check error and status codes
- ◎ **Patterns:** Software Exception; Dynamic Memory Corruption (Process Heap); Wait Chain (Critical Sections); Execution Residue (Unmanaged Space, User); Hidden Parameter; Last Error Collection
- ◎ [\AWMDA-Dumps\Exercise-P6-Analysis-process-dump-AppL-x64.pdf](#)

Exercise P7

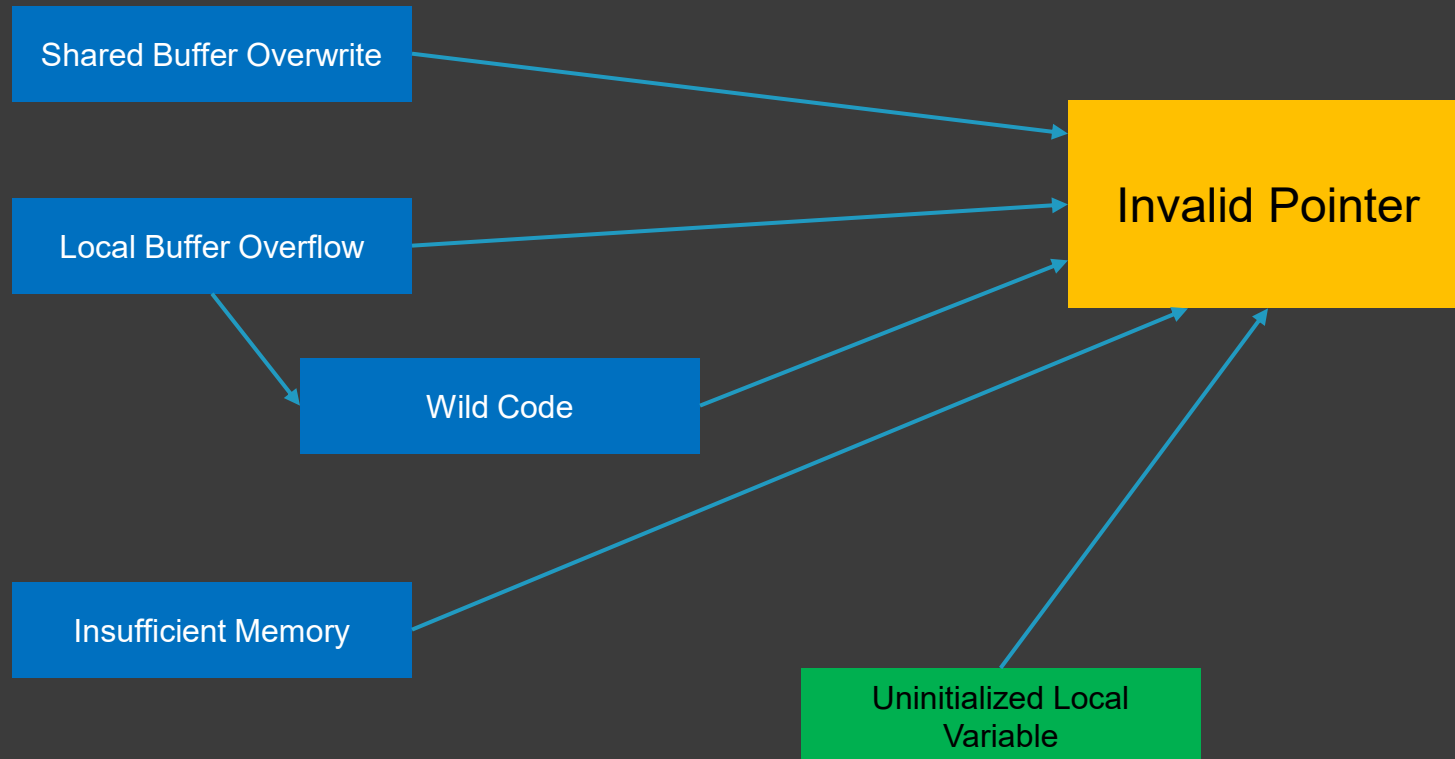
- ◎ **Goal:** Learn how to debug heap corruption using page heap
- ◎ **Patterns:** Invalid Pointer (General); Instrumentation Information
- ◎ [\AWMDA-Dumps\Exercise-P7-Analysis-process-dump-AppL2-x64.pdf](#)

Exercise P8

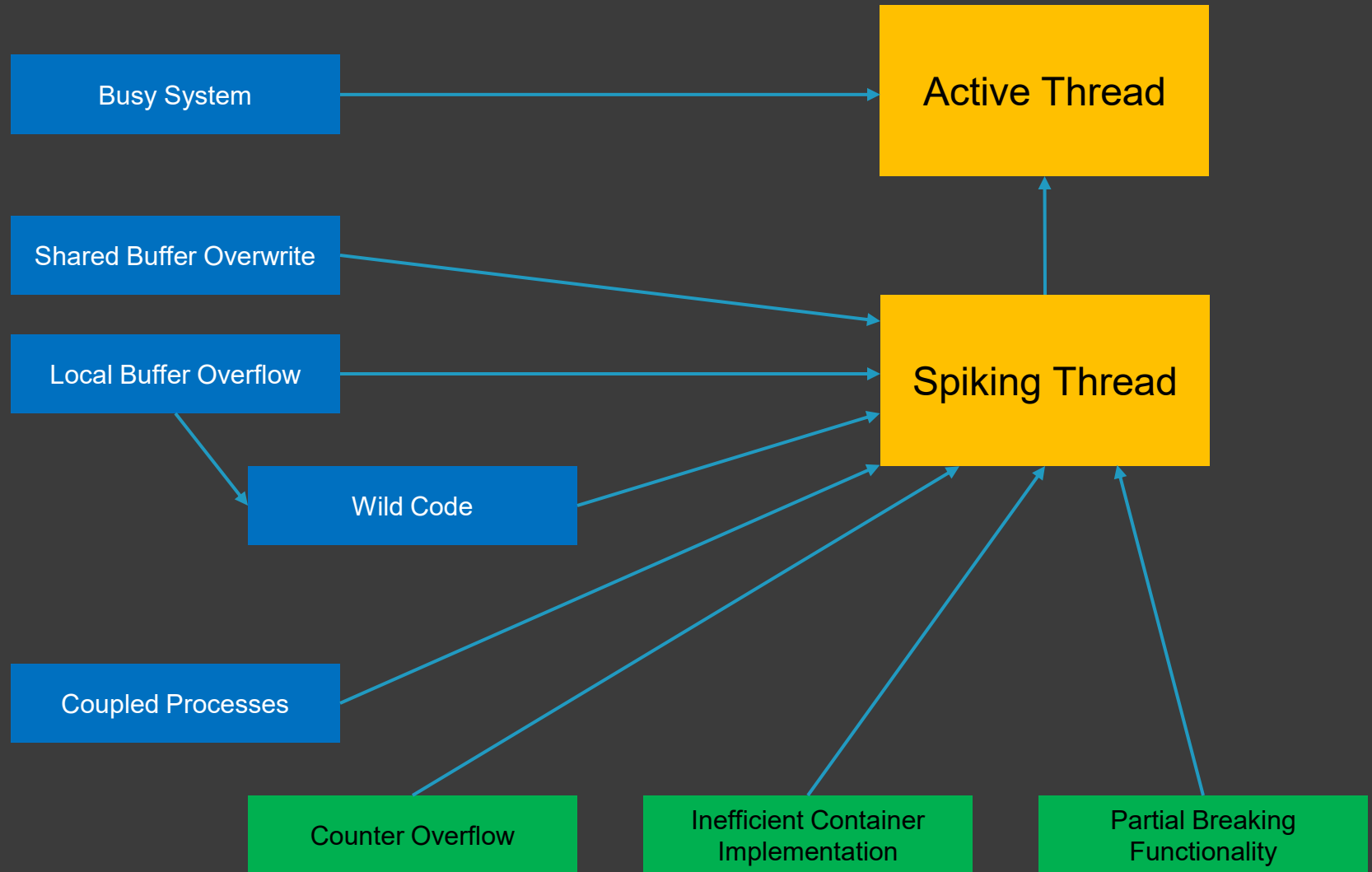
- ◎ **Goal:** Learn how to recognize CPU spikes, invalid pointers, disassemble code, and reconstruct stack traces
- ◎ **Patterns:** Wild Code; Active Thread; Spiking Thread; NULL Pointer (Code); Truncated Stack Trace; Stored Exception; Latent Structure
- ◎ [\AWMDA-Dumps\Exercise-P8-Analysis-process-dump-AppM-x64.pdf](#)

Mechanisms (Invalid Pointer)

Patterns-Based Root Cause Analysis Methodology Defect Mechanism Patterns (DMP)



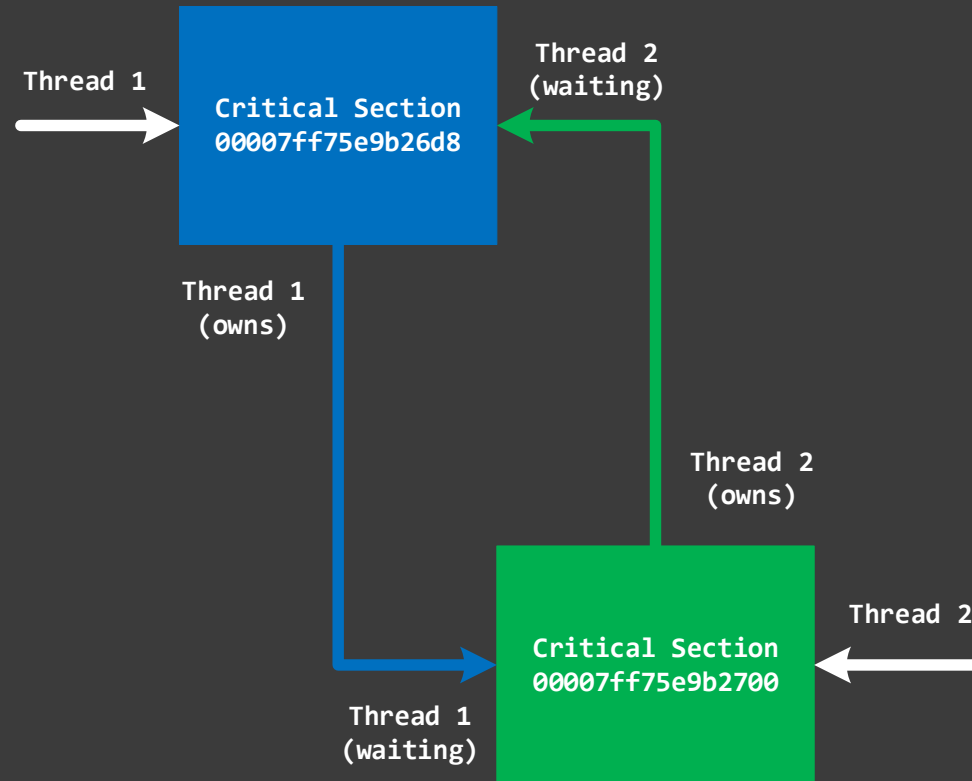
Mechanisms (Active Thread)



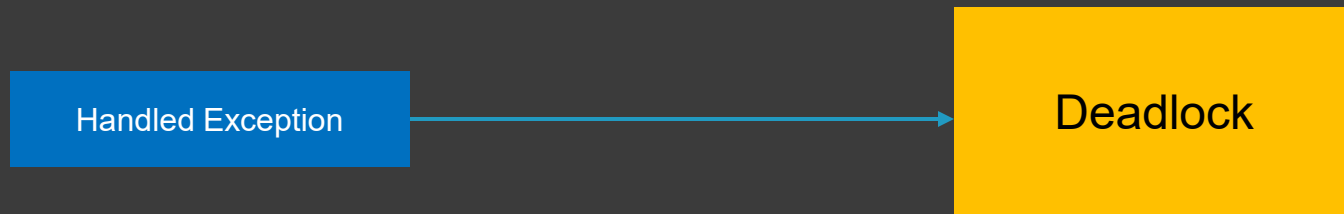
Exercise P9

- ◎ **Goal:** Learn how to recognize critical section waits and deadlocks, dump raw stack data, and see hidden exceptions
- ◎ **Patterns:** Deadlock (Critical Sections); Hidden Exception (User Space); Constant Subtrace
- ◎ [\AWMDA-Dumps\Exercise-P9-Analysis-process-dump-AppN-x64.pdf](#)

Deadlock



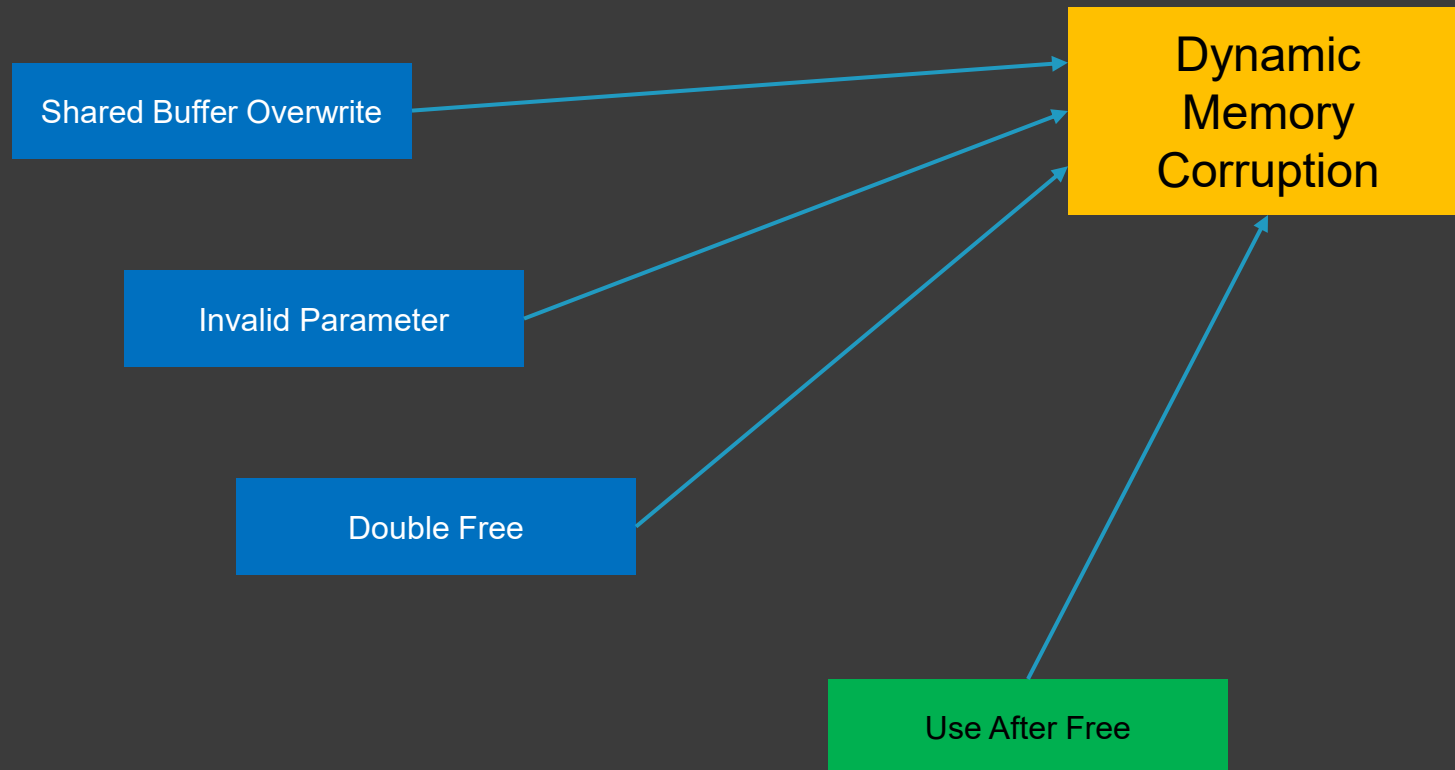
Mechanisms (Deadlock)



Exercise P10

- ◎ **Goal:** Learn how to recognize application heap problems, buffer and stack overflow patterns, and analyze raw stack data
- ◎ **Patterns:** Double Free (Process Heap); Local Buffer Overflow (User Space); Stack Overflow (User Mode)
- ◎ [\AWMDA-Dumps\Exercise-P10-Analysis-process-dump-AppO-x64.pdf](#)

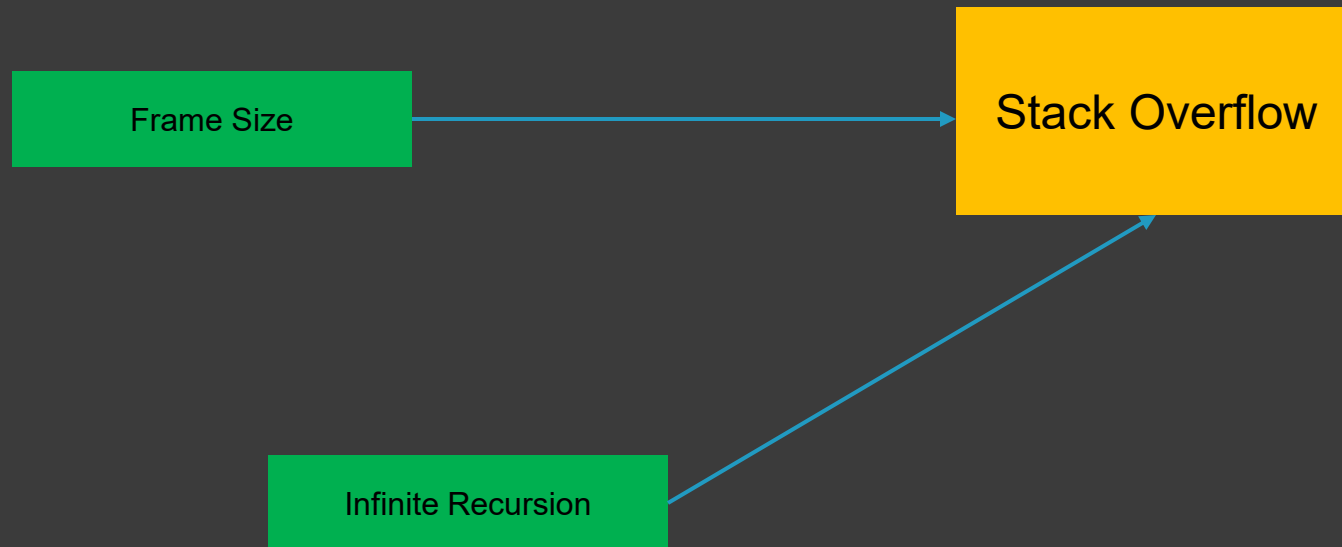
Mechanisms (Heap Corruption)



Mechanisms (Stack Corruption)



Mechanisms (Stack Overflow)



Exercise P11

- ◎ **Goal:** Learn how to analyze exception patterns, raw stacks, and execution residue
- ◎ **Patterns:** Divide by Zero (User Mode); C++ Exception; Execution Residue (Unmanaged Space, User)
- ◎ [\AWMDA-Dumps\Exercise-P11-Analysis-process-dump-AppP-x64.pdf](#)

Exercise P12

- ◎ **Goal:** Learn how to analyze managed space
- ◎ **Patterns:** Platform-Specific Debugger; CLR Thread; JIT Code (.NET); Managed Code Exception; Managed Stack Trace
- ◎ [\AWMDA-Dumps\Exercise-P12-Analysis-process-dump-AppR2-x64.pdf](#)

Exercise P13

- ◎ **Goal:** Learn how to analyze the x86 process saved as an x64 process memory dump
- ◎ **Patterns:** Virtualized Process (WOW64); Message Box; Debugger Bug; Rough Stack Trace (Unmanaged Space)
- ◎ [\AWMDA-Dumps\Exercise-P13-Analysis-process-dump-AppA-WOW64-x64.pdf](#)

Exercise P14

- ◎ **Goal:** Learn how to analyze process memory leaks
- ◎ **Patterns:** Thread Age; Memory Leak (Process Heap)
- ◎ [\AWMDA-Dumps\Exercise-P14-Analysis-process-dump-AppS-x64.pdf](#)

Mechanisms (Memory Leak)

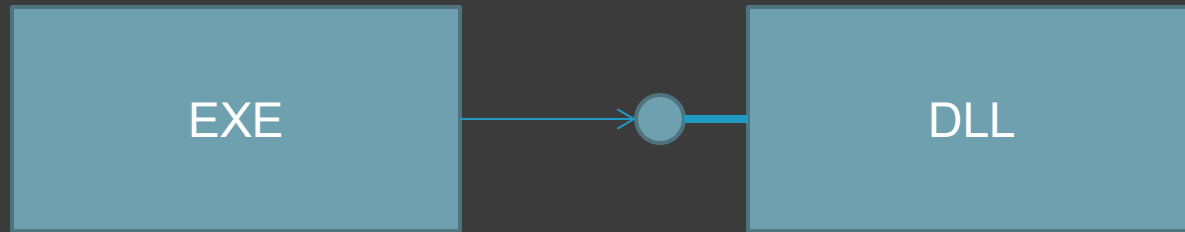


Parameters and Locals

[Debugging TV Frames episode 0x18](#)

Symbol Types

- Exported and imported names



- Function and variable names
- Data types

Exercise P15

- ◎ **Goal:** Learn how to navigate function parameters in cases of reduced symbolic information in 32-bit process memory dumps from x64 Windows
- ◎ **Patterns:** Reduced Symbolic Information
- ◎ [\AWMDA-Dumps\Exercise-P15-Analysis-process-dump-notepad-32-x64.pdf](#)

Exercise P16

- ◎ **Goal:** Learn how to navigate function parameters in x64 process memory dumps
- ◎ **Patterns:** False Function Parameters; Injected Symbols
- ◎ [\AWMDA-Dumps\Exercise-P16-Analysis-process-dump-notepad-x64.pdf](#)

Exercise P17

- ◎ **Goal:** Learn how to navigate object wait chains in 32-bit memory dumps saved with ProcDump on x64 Windows
- ◎ **Patterns:** Embedded Comments; Wait Chain (General); No Data Types; Deadlock (Mixed Objects, User Space); Small Value
- ◎ [\AWMDA-Dumps\Exercise-P17-Analysis-process-dump-AppQ-32-x64.pdf](#)

Exercise P18

- ◎ **Goal:** Learn how to navigate object wait chains in x64 memory dumps saved with ProcDump on x64 Windows
- ◎ **Patterns:** Not My Thread; Blocked Thread (Software); Main Thread; Passive Thread (User Space); Coincidental Symbolic Information
- ◎ [\AWMDA-Dumps\Exercise-P18-Analysis-process-dump-AppQ-x64.pdf](#)

Exercise P19

- ◎ **Goal:** Learn how to analyze process handle leaks
- ◎ **Patterns:** Active Space; Handle Leak
- ◎ [\AWMDA-Dumps\Exercise-P19-Analysis-process-dump-AppT-x64.pdf](#)

Exercise P20

- ◎ **Goal:** Learn how to analyze service memory dumps
- ◎ **Patterns:** Input Thread; Blocking Module
- ◎ [\AWMDA-Dumps\Exercise-P20-Analysis-process-dump-ServiceA-x64.pdf](#)

Exercise P21

- ◎ **Goal:** Learn how to analyze memory dumps from Rust processes
- ◎ **Patterns:** Language-Specific Subtrace (Rust)
- ◎ [\AWMDA-Dumps\Exercise-P21-Analysis-process-dump-rusty-x64.pdf](#)

Exercise P22

- ◎ **Goal:** Learn how to analyze native ARM64 memory dumps
- ◎ **Patterns:** Encoded Pointer
- ◎ [\AWMDA-Dumps\Exercise-P22-Analysis-process-dump-AppA-ARM64.pdf](#)

Exercise P23

- ◎ **Goal:** Learn how to analyze ARM64 memory dumps from x64 processes
- ◎ **Patterns:** Virtualized Process (ARM64EC); ISA-Specific Code
- ◎ [\AWMDA-Dumps\Exercise-P23-Analysis-process-dump-AppA-ARM64EC.pdf](#)

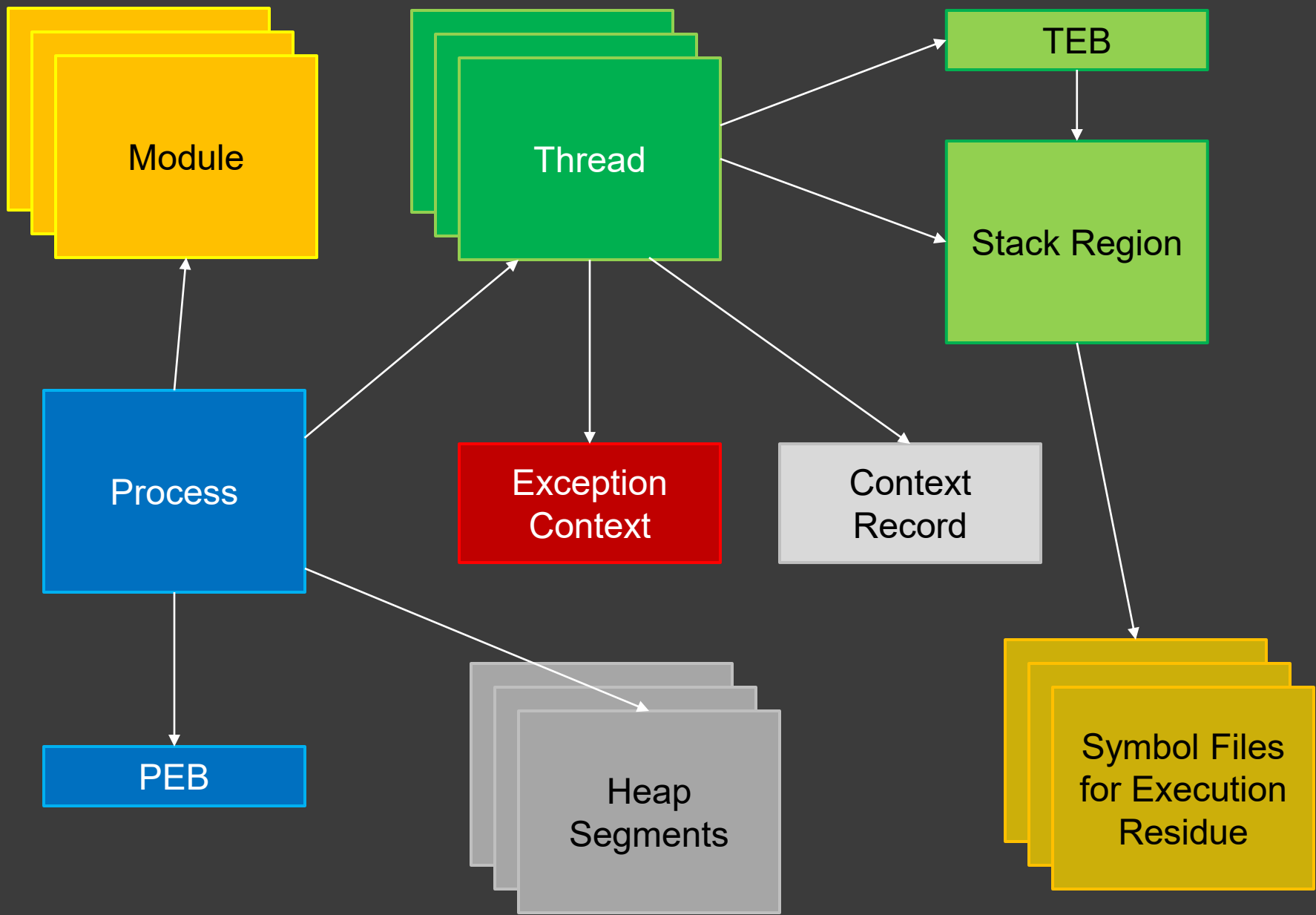
Exercise P24

- ◎ **Goal:** Learn how to analyze 32-bit memory dumps from x86 processes on ARM64 machines
- ◎ **Patterns:** Virtualized Process (CHPE)
- ◎ [\AWMDA-Dumps\Exercise-P24-Analysis-process-dump-AppA-x86-CHPE.pdf](#)

Exercise P25

- ◎ **Goal:** Learn how to analyze 64-bit memory dumps from x86 processes on ARM64 machines
- ◎ **Patterns:** Virtualized Process (WOW64, CHPE)
- ◎ [\AWMDA-Dumps\Exercise-P25-Analysis-process-dump-AppA-WOW64-ARM64.pdf](#)

Part 1E: Windows Internals



Part 1F: Collection Methods

Process Dump Generation

- ◎ Crash or Hang, ... ?

 - PID in Task Manager

- ◎ 32-bit vs. 64-bit?

 - 32-bit Task Manager (from \Windows\SysWOW64) vs. 64-bit Task Manager

- ◎ Windows XP / W2K3

 - Crash: set default debugger or Userdump monitoring
 - Hang / Leak / Spike: userdump.exe

- ◎ Windows Vista / W2K8 / W7 / W8 / W10 / W11

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

- ◎ [Window2Dump](#)

- ◎ Attached WinDbg and [.dump](#) / [.dumpcab](#)

Pattern Links

[Spiking Thread](#)

[C++ Exception](#)

[Divide by Zero \(User Mode\)](#)

[Dynamic Memory Corruption \(Process Heap\)](#)

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

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

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

[Managed Stack Trace](#)

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

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

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

[Environment Hint](#)

[Unknown Component](#)

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

[False Function Parameters](#)

[Reduced Symbolic Information](#)

[Stored Exception](#)

[Instrumentation Information](#)

[JIT Code \(.NET\)](#)

[Embedded Comment](#)

[Deadlock \(Mixed Object, User Space\)](#)

[Blocked Thread \(Software\)](#)

[Passive Thread \(User Space\)](#)

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

[Memory Leak \(Process Heap\)](#)

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

[CLR Thread](#)

[Deadlock \(Critical Sections\)](#)

[Double Free \(Process Heap\)](#)

[Exception Stack Trace](#)

[Hidden Exception \(User Space\)](#)

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

[Managed Code Exception](#)

[Multiple Exceptions \(User Mode\)](#)

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

[Stack Trace](#)

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

[Wild Code](#)

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

[Message Box](#)

[Injected Symbols](#)

[Truncated Stack Trace](#)

[Incorrect Stack Trace](#)

[Active Thread](#)

[Thread Age](#)

[Wait Chain \(General\)](#)

[Not My Thread](#)

[Main Thread](#)

[Coincidental Symbolic Information](#)

[Platform-Specific Debugger](#)

[Language-Specific Subtrace \(Rust\)](#)

[Active Space](#)

[Debugger Bug](#)

[Exception Module](#)

[Blocking Module](#)

[Last Error Collection](#)

[Handle Leak](#)

[Input Thread](#)

[Historical Information](#)

[Software Exception](#)

[Hidden Parameter](#)

[Stack Trace Motif](#)

[Encoded Pointer](#)

[ISA-Specific Code](#)

[Latent Structure](#)

[Constant Subtrace](#)

[Small Value](#)

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

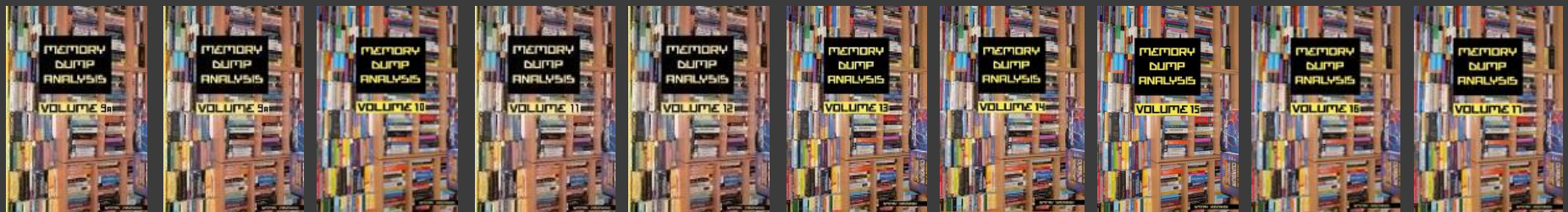
More than 70 multiple pattern case studies:

<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.
- Advanced Windows Debugging
- Inside Windows Debugging
- [Principles of Memory Dump Analysis](#)
- [Windows Debugging Notebook: Essential User Space WinDbg Commands](#)
- [Encyclopedia of Crash Dump Analysis Patterns, 3rd edition](#)
- [Memory Dump Analysis Anthology \(Diagnomicon\)](#)



Further Training Courses

- [Accelerated Windows Memory Dump Analysis, 7th Edition, Part 2](#)
- [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!