Windows Memory Dump Analysis

Accelerated

Version 6

Part 2: Kernel and Complete Spaces

Dmitry Vostokov
Software Diagnostics Services

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
- Part 1C: Learn how to analyze process dumps
- Part 2A: Review fundamentals
- Part 2B: Review x64 disassembly
- Part 2C: Learn how to analyze kernel dumps
- Part 2D: Learn how to analyze complete (physical memory) dumps
- Part 2E: Learn how to analyze minidumps

Training Principles

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

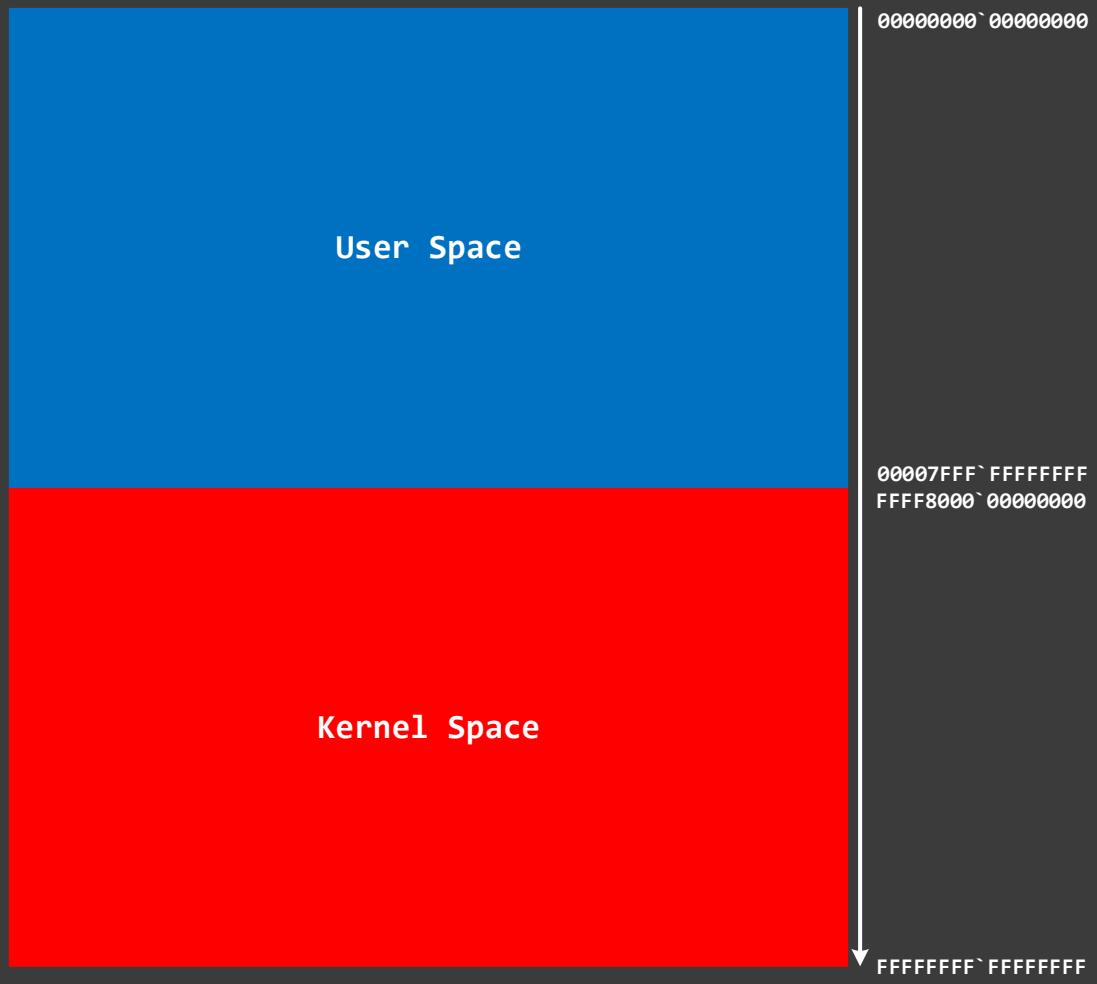
Coverage (Part 2)

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

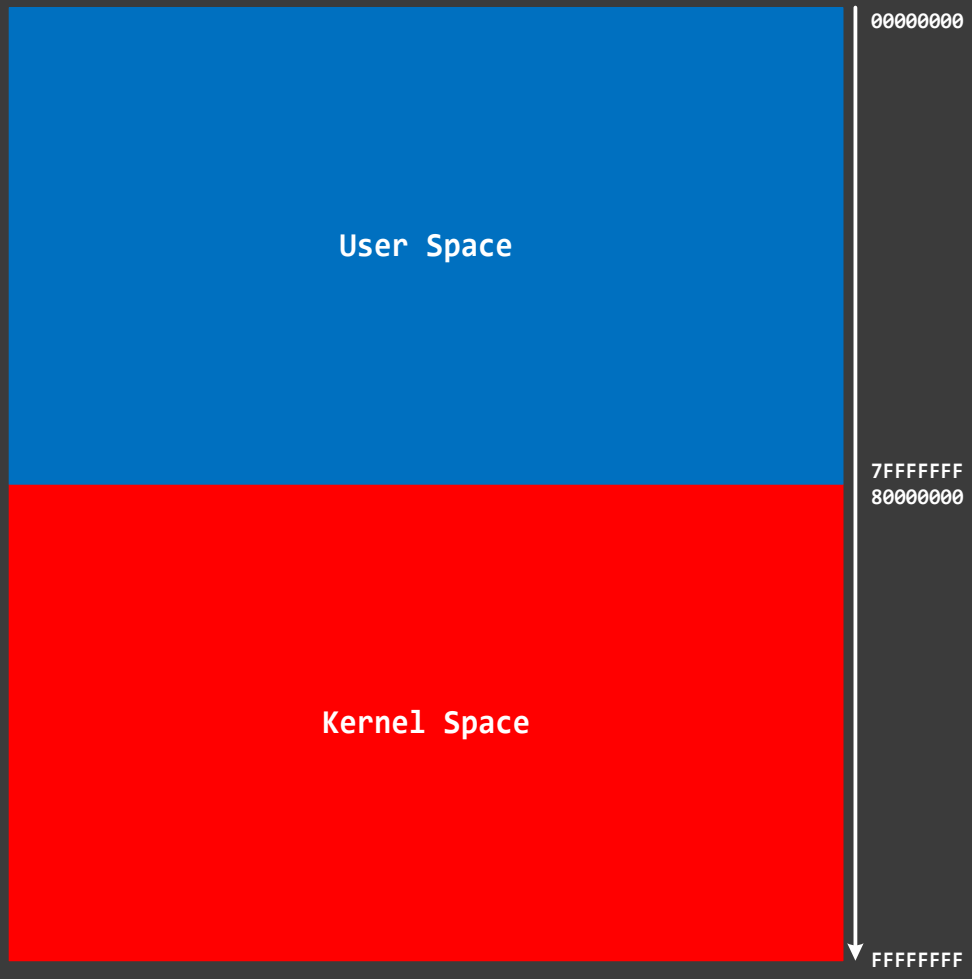
* Most of the exercises are focused on x64 code. For their x86 equivalents from older Windows versions, please refer to the previous fourth edition of this course.

Part 2A: Fundamentals

Process Space (x64)

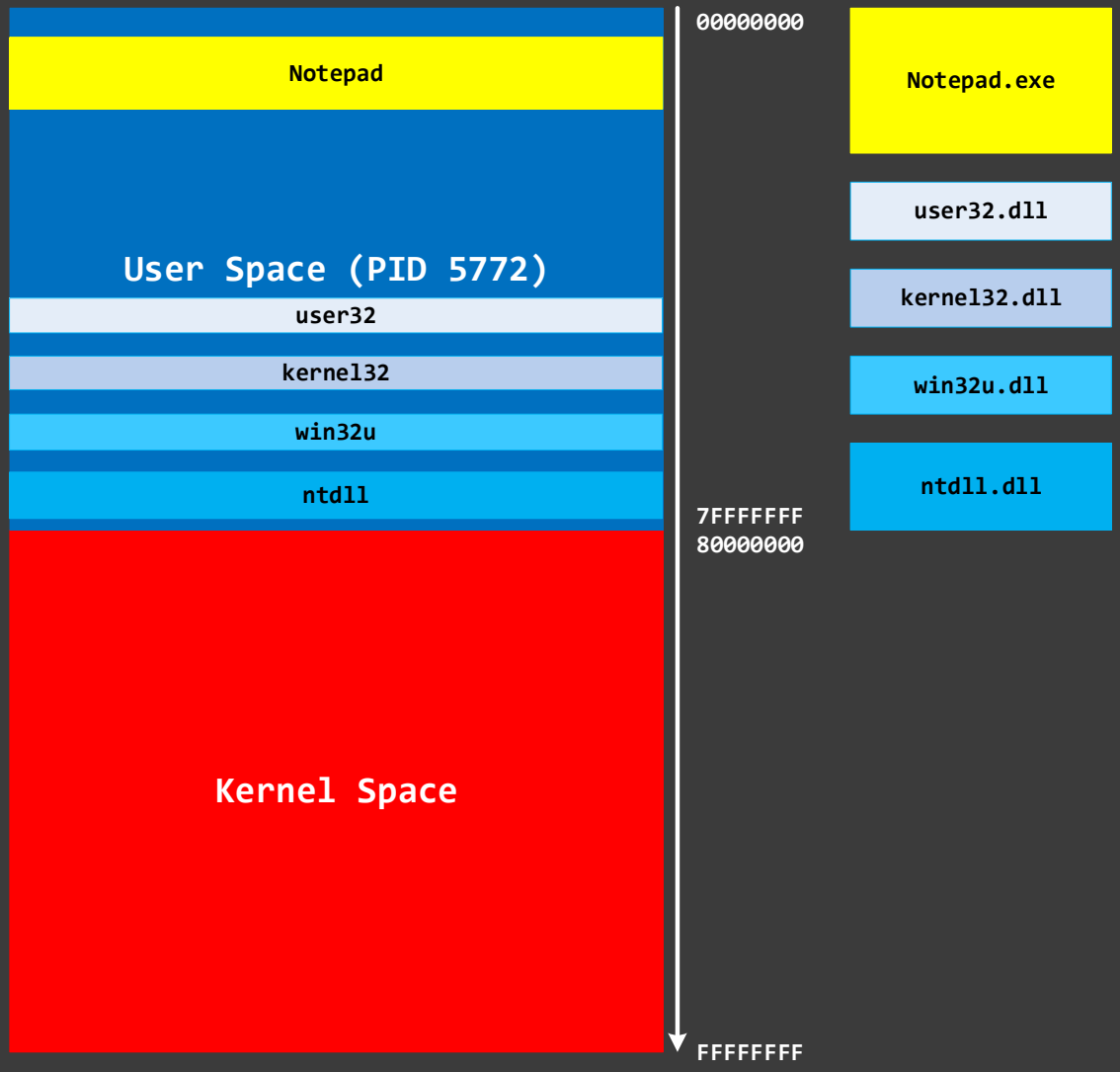


Process Space (x86)

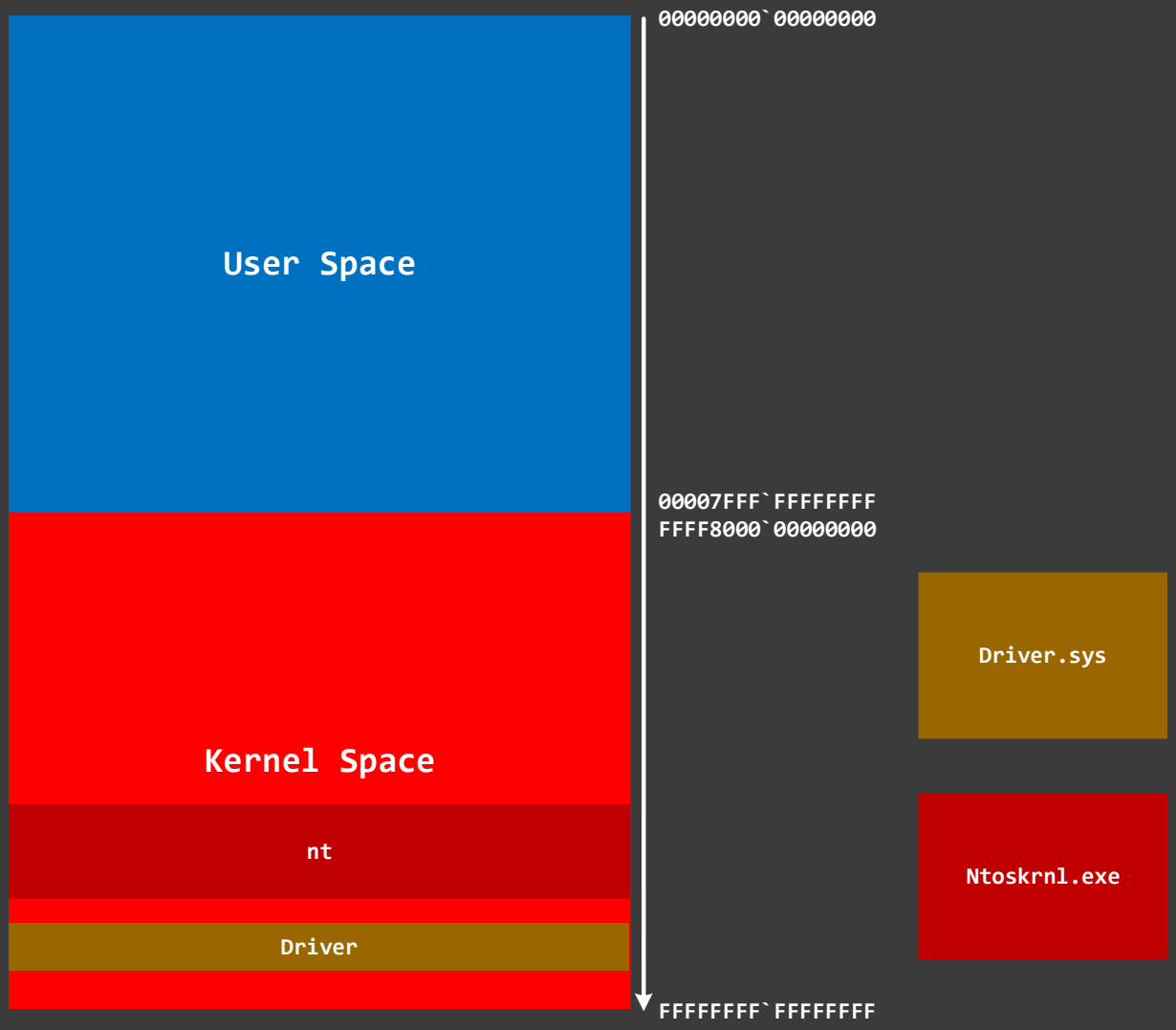


Application/Process/Module (x64)

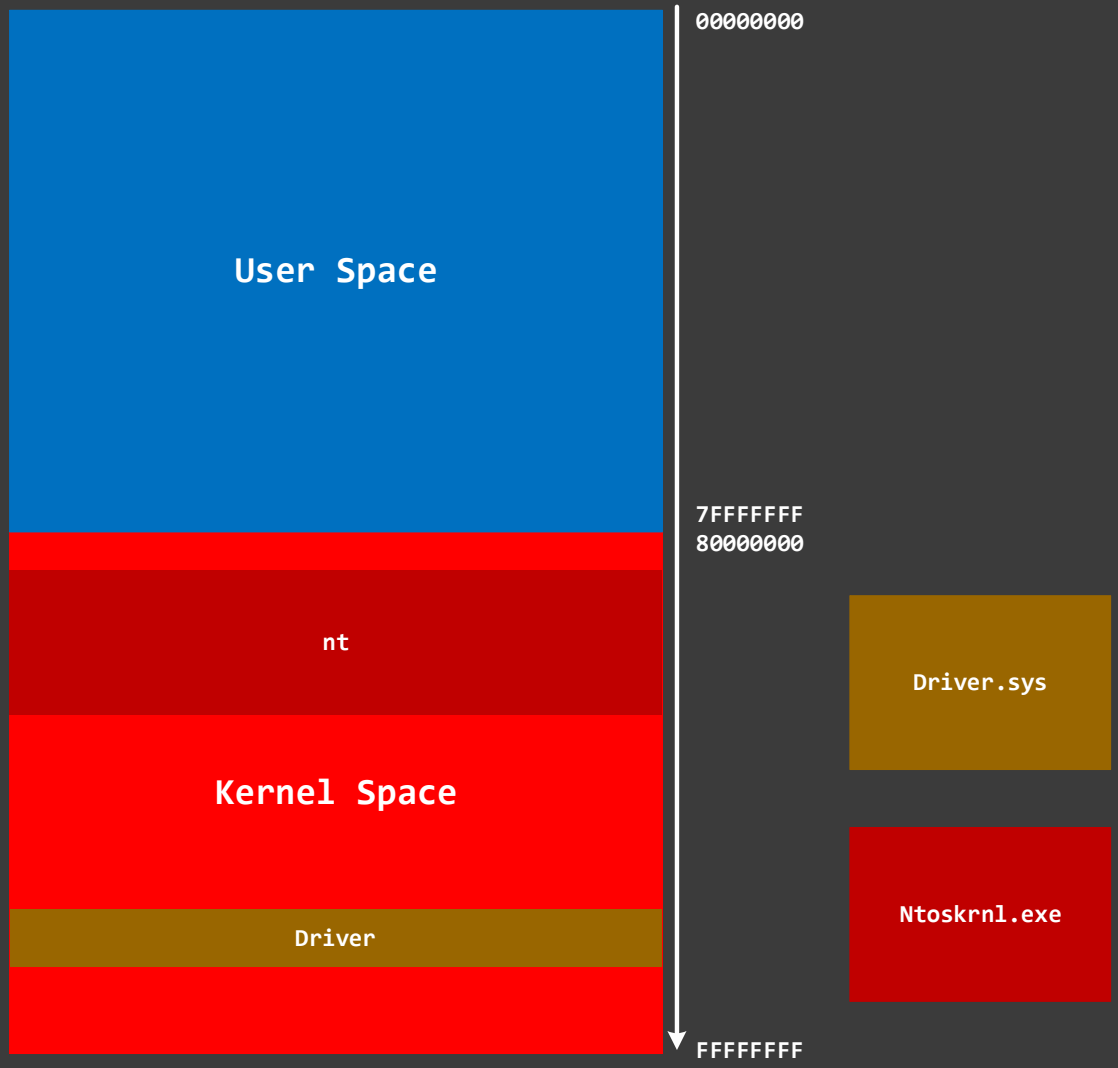
Application/Process/Module (x86)



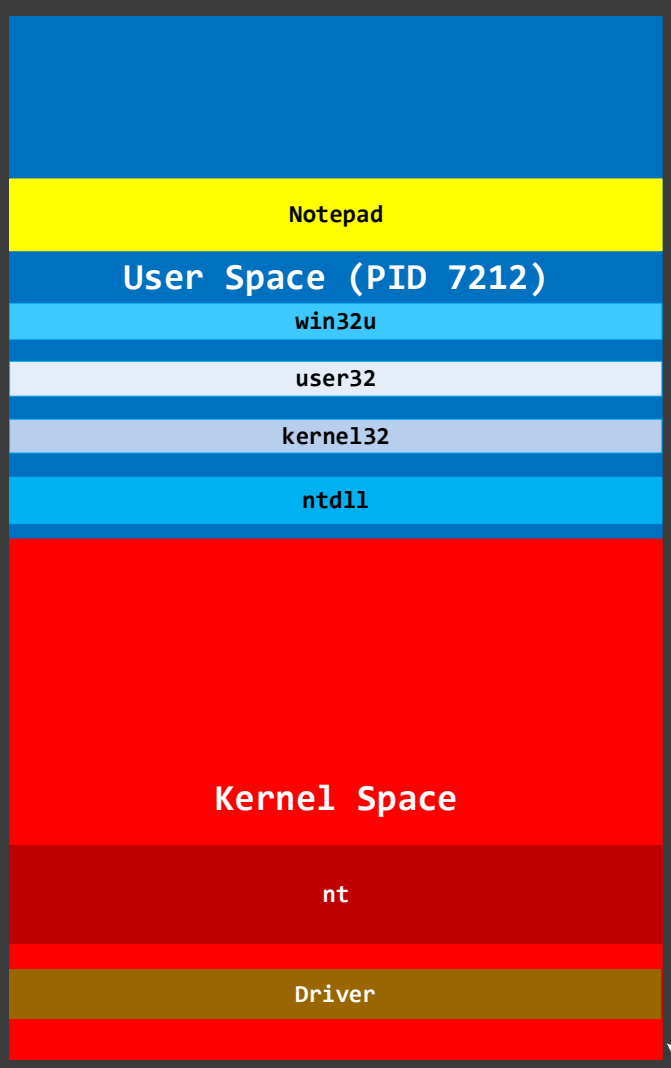
OS Kernel/Driver/Module (x64)



OS Kernel/Driver/Module (x86)



Process Virtual Space (x64)



00000000`00000000

00007FF6`00000000

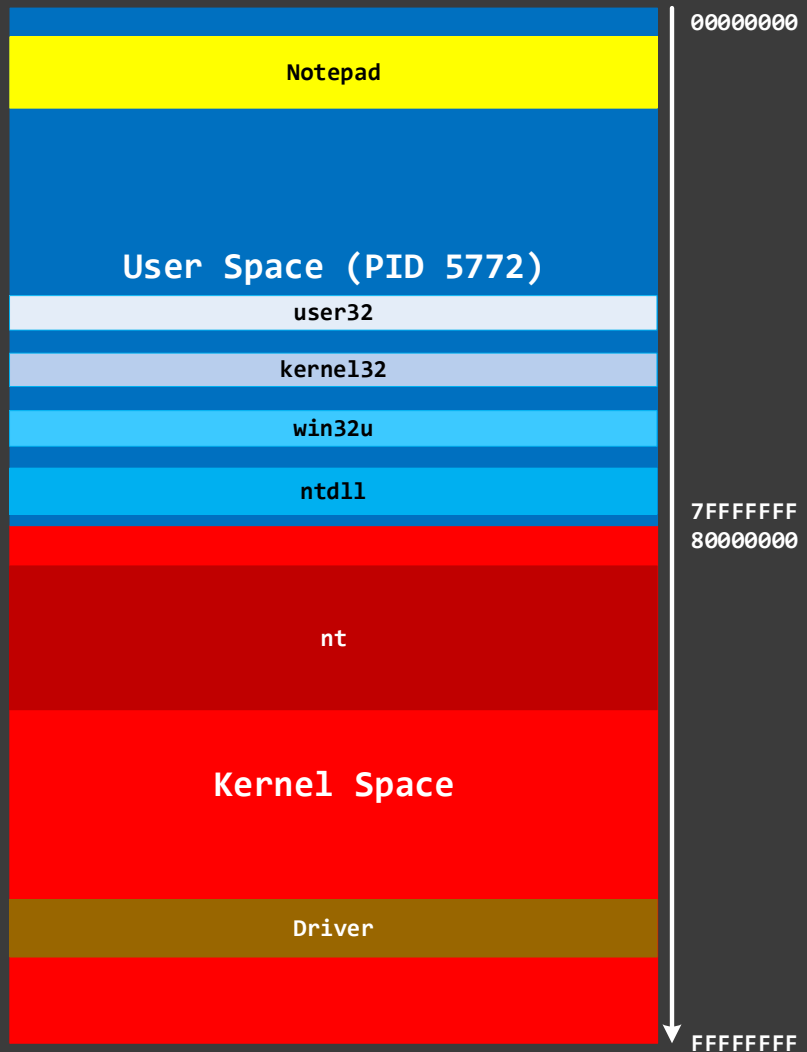
00007FFF`FFFFFFFF

FFFF8000`00000000

FFFFFFFF`FFFFFFFF

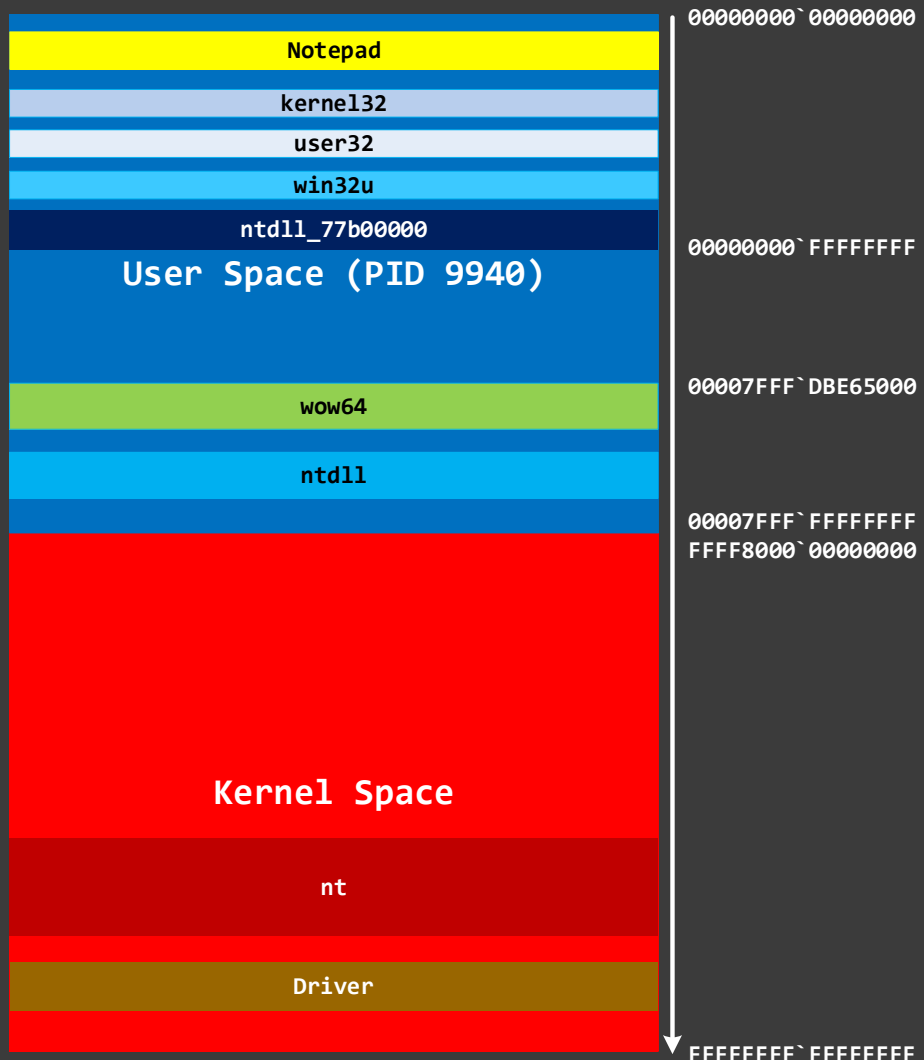
00000000`00000000 ...
FFFFFFFF`FFFFFFFF

Process Virtual Space (x86)

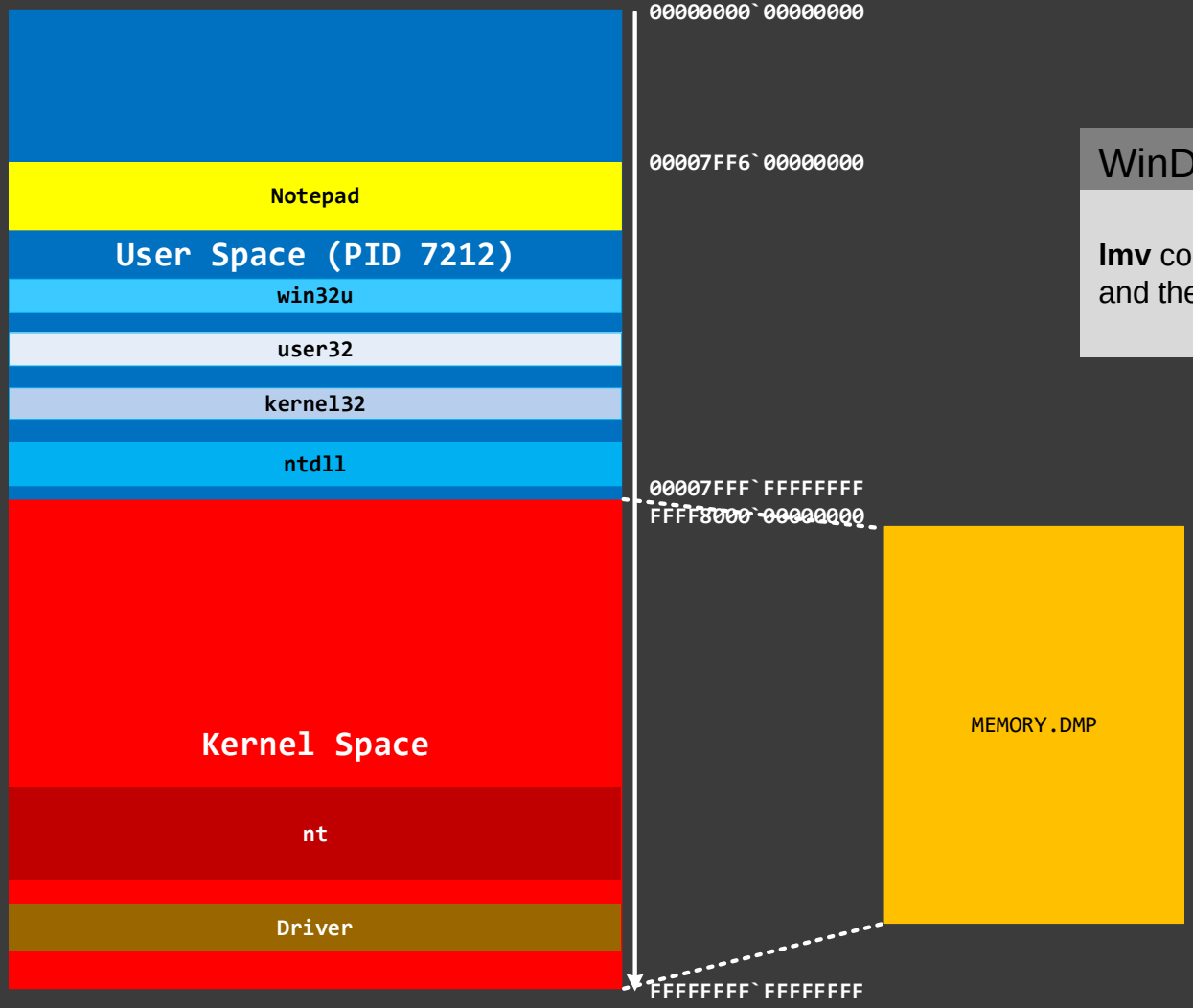


00000000 ... FFFFFFFF

Process Virtual Space (WOW64)



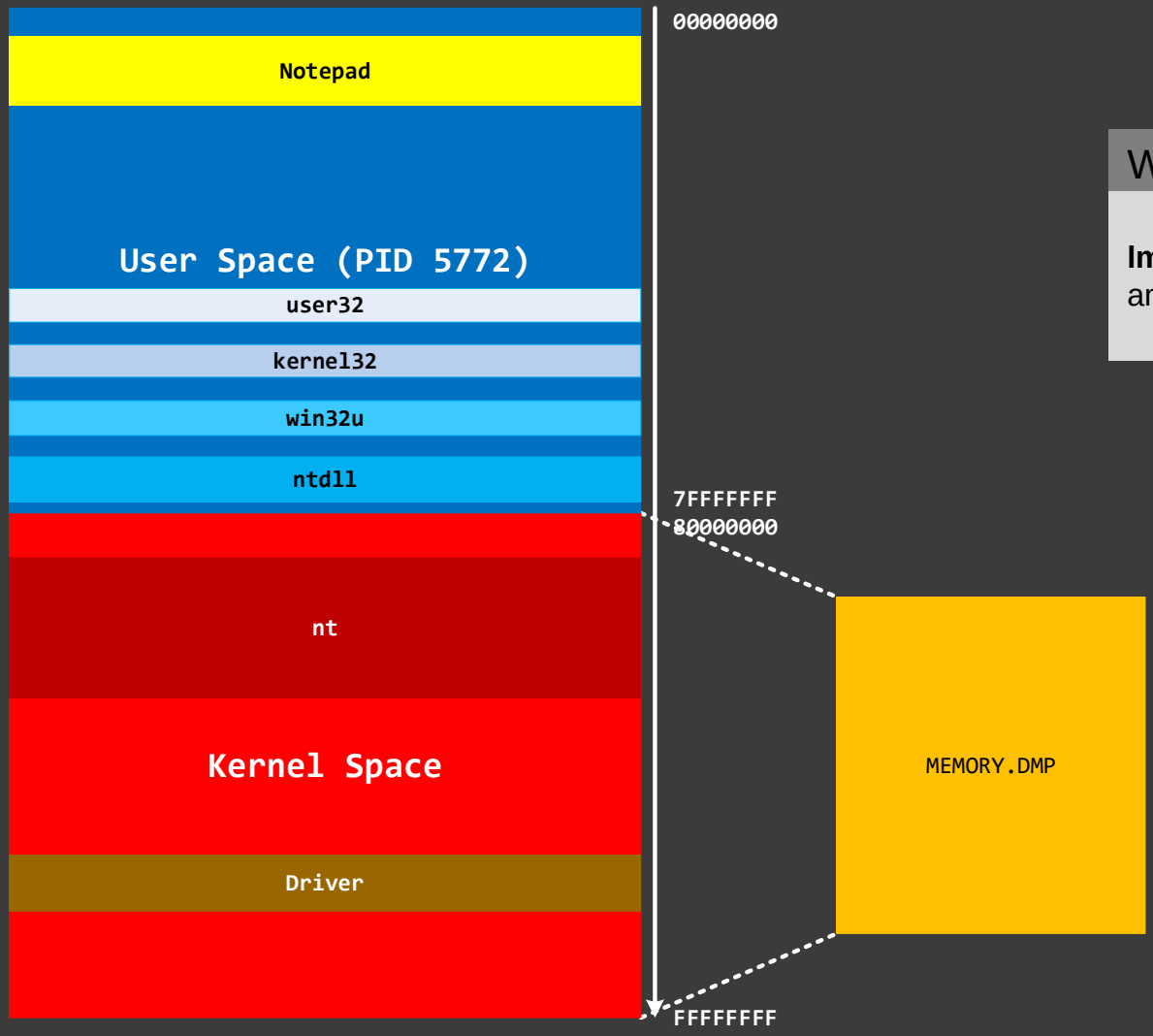
Kernel Memory Dump (x64)



WinDbg Commands

lmv command lists modules and their description

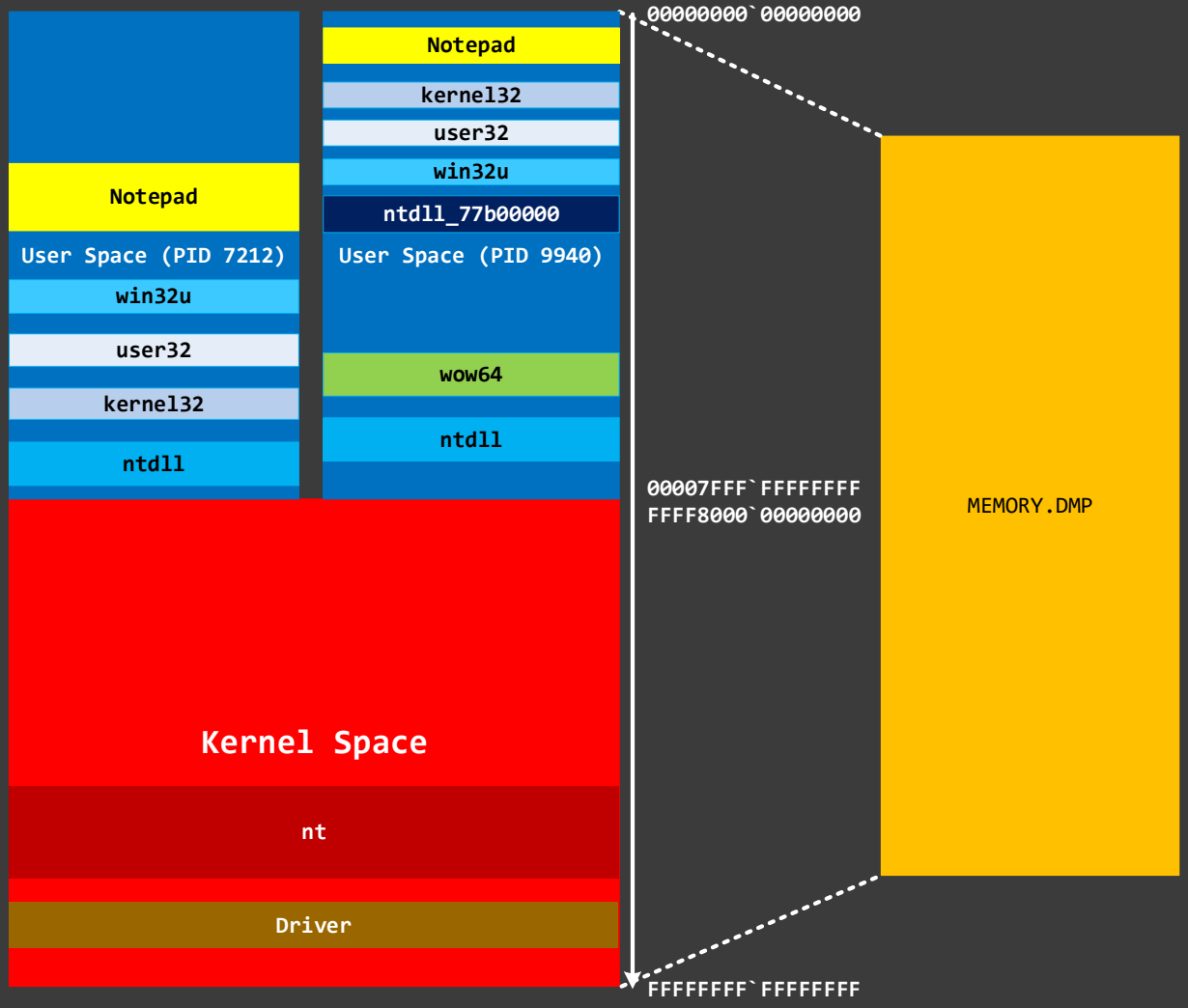
Kernel Memory Dump (x86)



WinDbg Commands

lmv command lists modules and their description

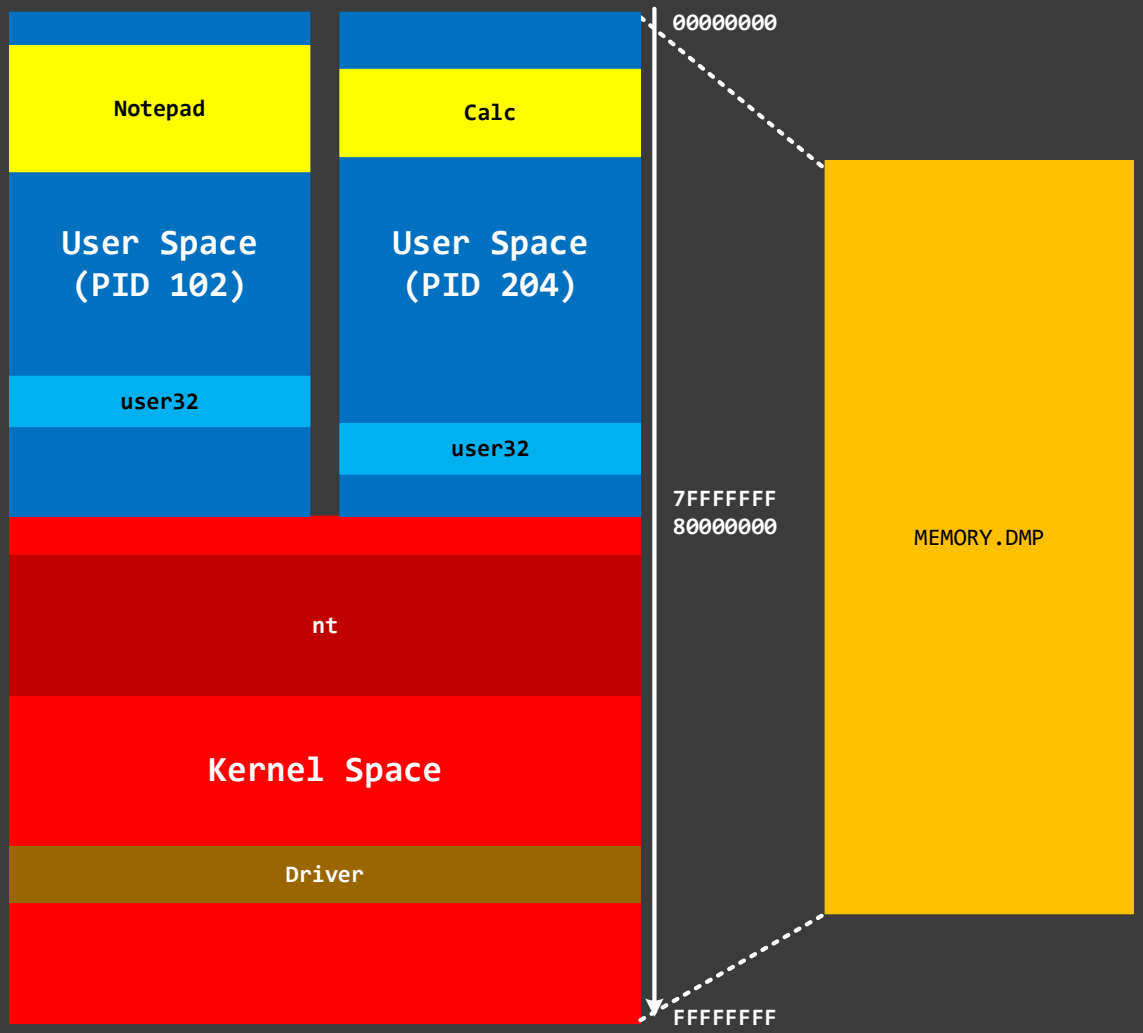
Complete Memory Dump (x64)



WinDbg Commands

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

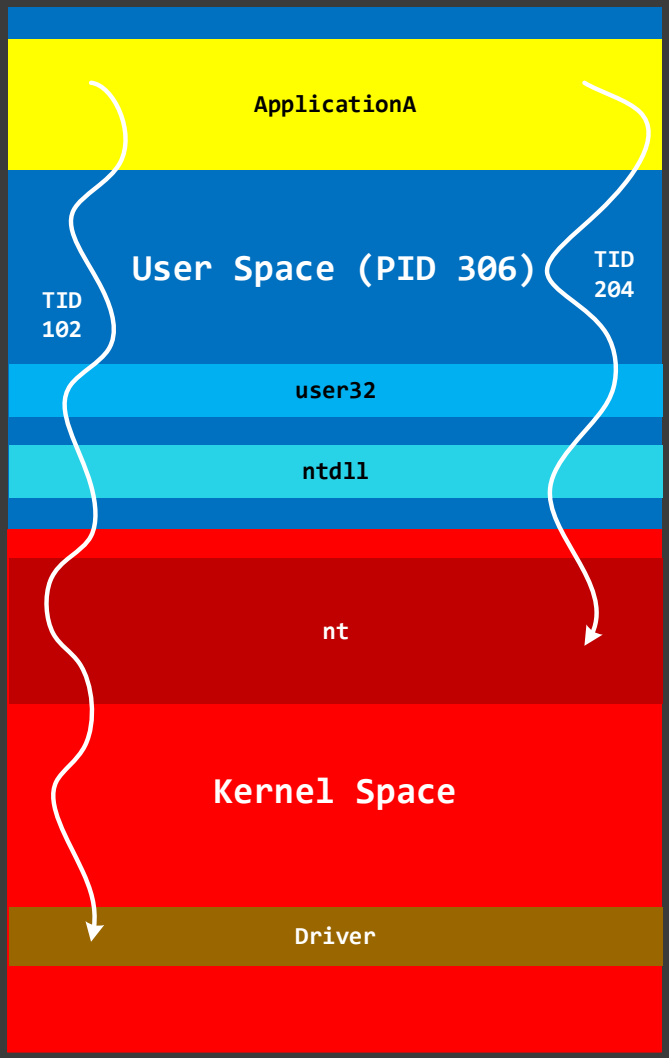
Complete Memory Dump (x86)



WinDbg Commands

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

Process Threads



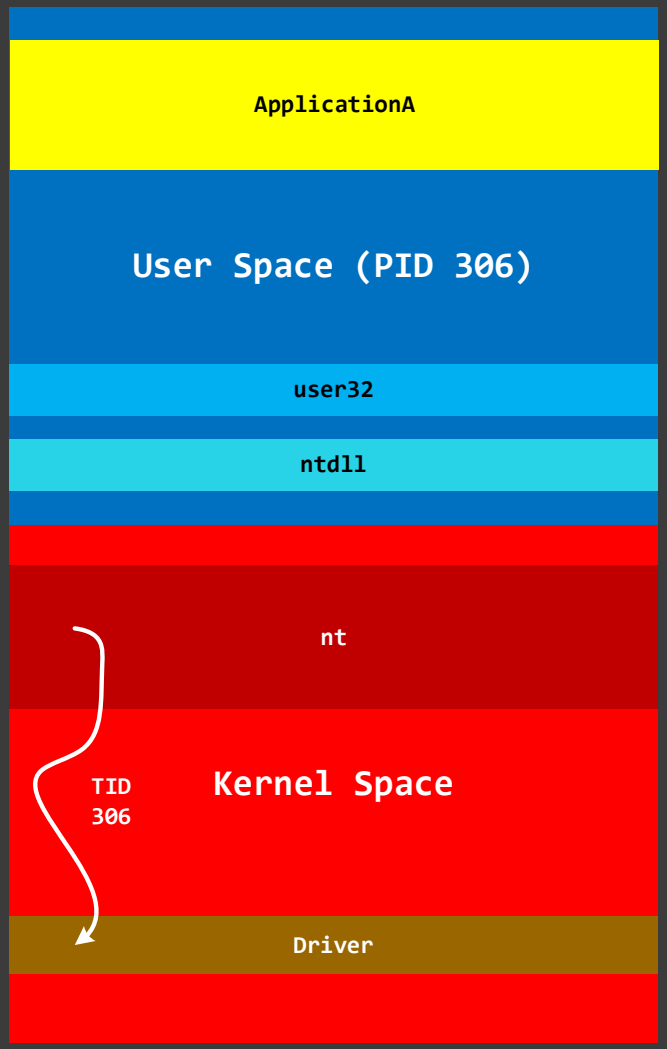
WinDbg Commands

Kernel/Complete dumps:

~<n>s switches between processors

.thread switches between threads

System Threads



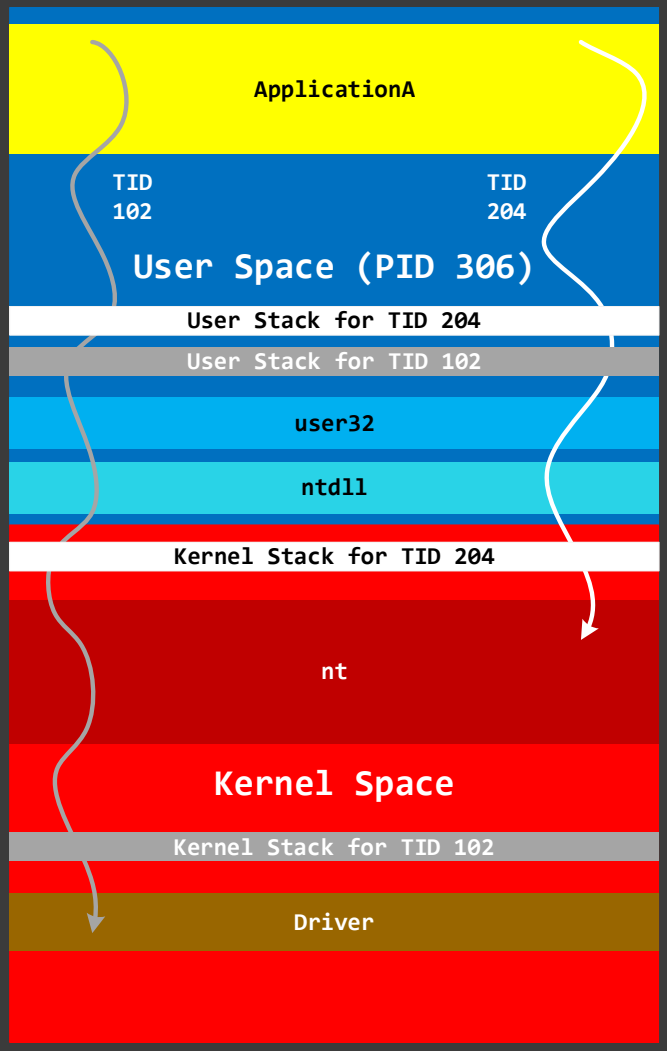
WinDbg Commands

Kernel/Complete dumps:

~<n>s switches between processors

.thread switches between threads

Thread Stack Raw Data



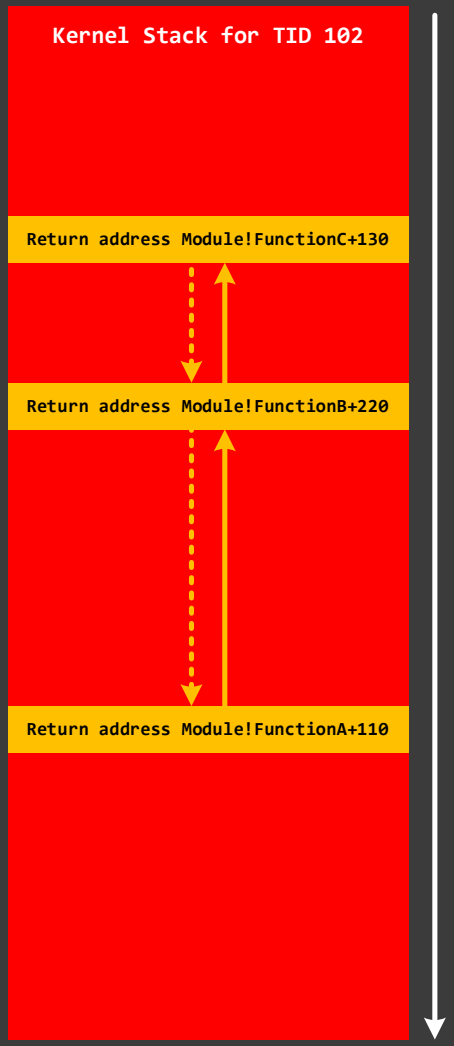
WinDbg Commands

Kernel dumps:
!thread

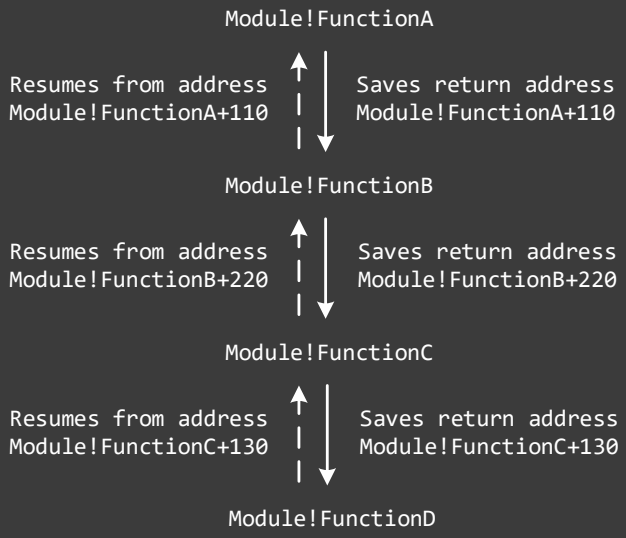
Complete dumps:
!teb for user space
!thread for kernel space

Data:
dc / **dps** / **dpp** / **dpa** / **dpu**

Thread Stack Trace



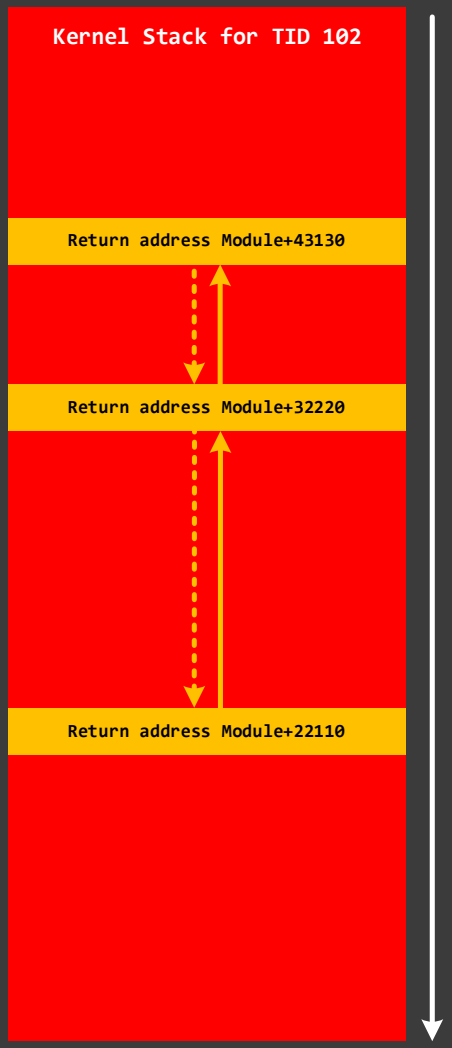

```
FunctionA()
{
  ...
  FunctionB();
  ...
}

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)

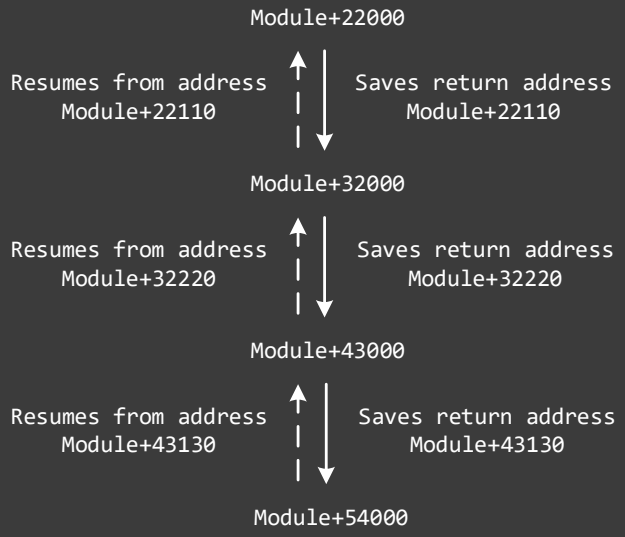



```
FunctionA()
{
  ...
  FunctionB();
  ...
}

FunctionB()
{
  ...
  FunctionC();
  ...
}

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

Symbol file Module.pdb

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

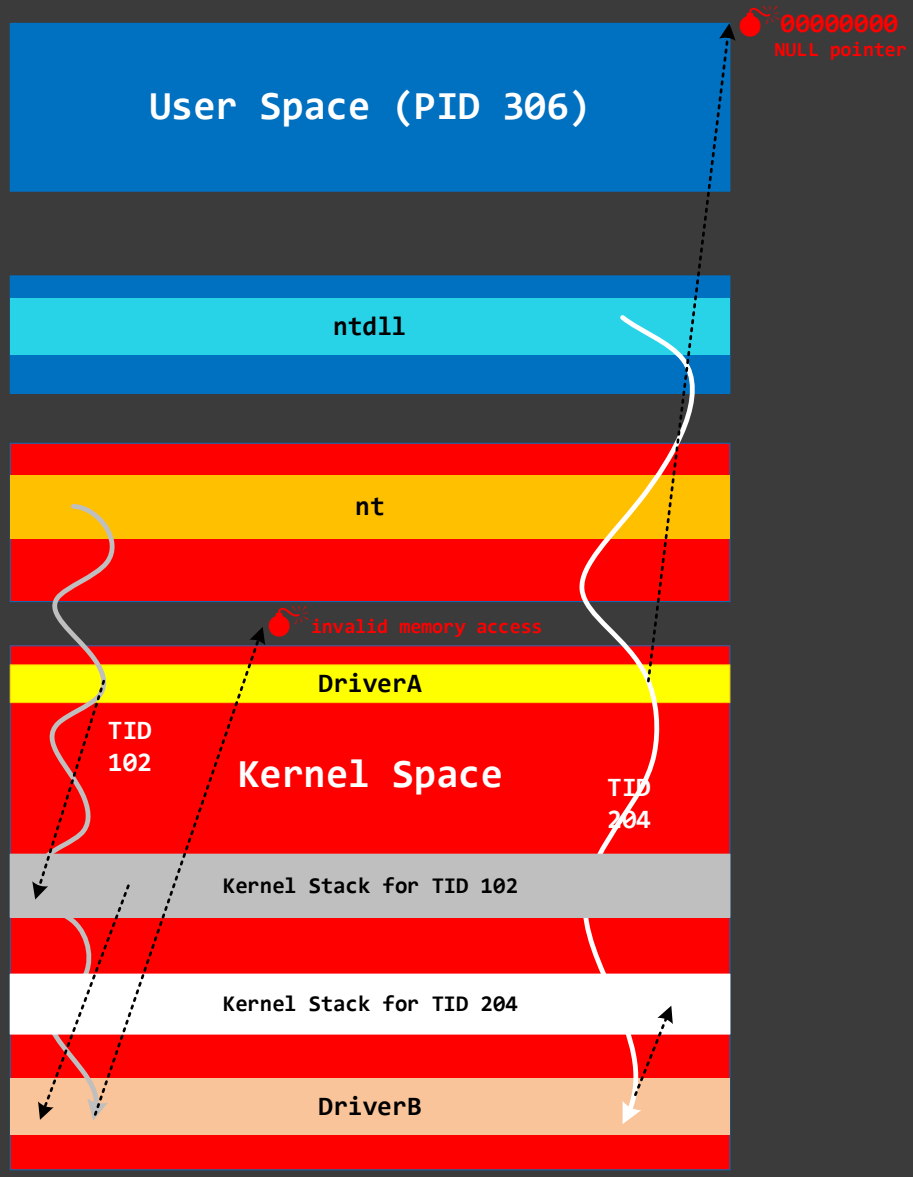
No symbols for Module



WinDbg Commands

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

Exceptions (Access Violation)



WinDbg Commands

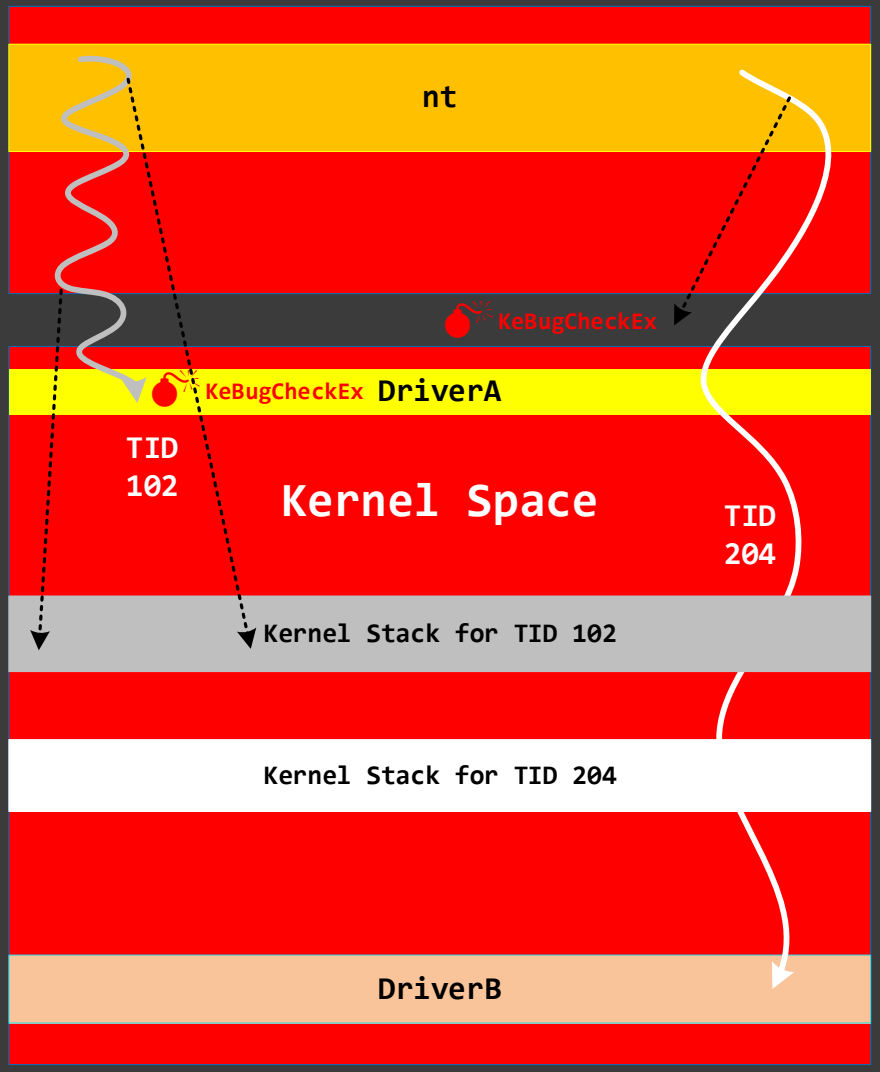
`address=????????`

Set exception context:
.cxr

Set trap context:
.trap

Check address:
!pte

Bugchecks (Runtime)



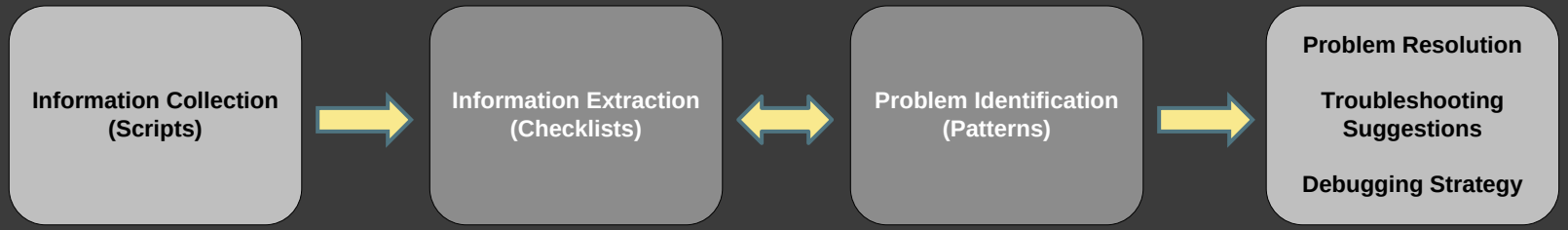
Pattern-Oriented Diagnostic Analysis

Diagnostic 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/

Part 2B: x64 Disassembly

x64 CPU Registers

- **RAX** ⊃ **EAX** ⊃ **AX** ⊇ {**AH**, **AL**}

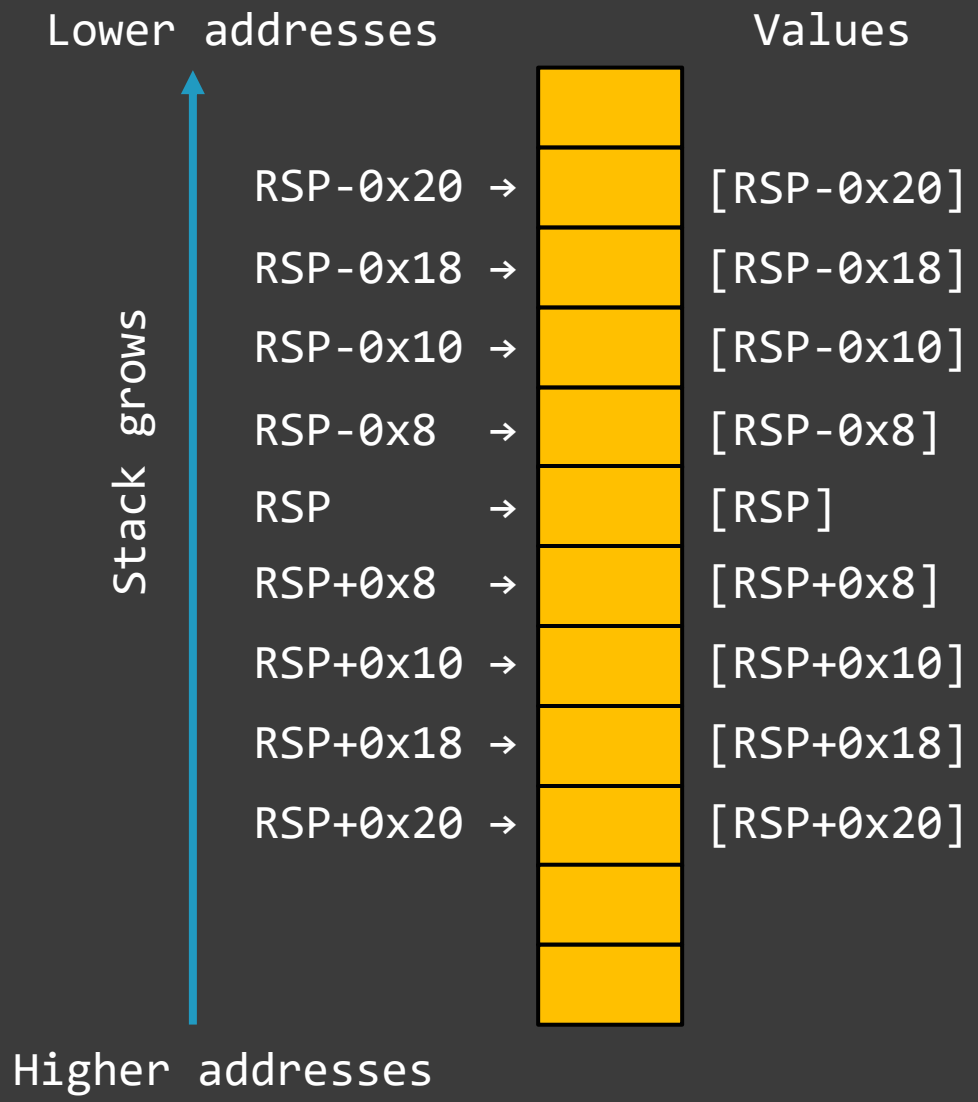
- ALU: **RAX**, **RDX**
- Counter: **RCX**
- Memory copy: **RSI** (src), **RDI** (dst)
- Stack: **RSP**
- Next instruction: **RIP**
- New: **R8** – **R15**, **Rx**(**D**|**W**|**B**)

Instructions and Registers

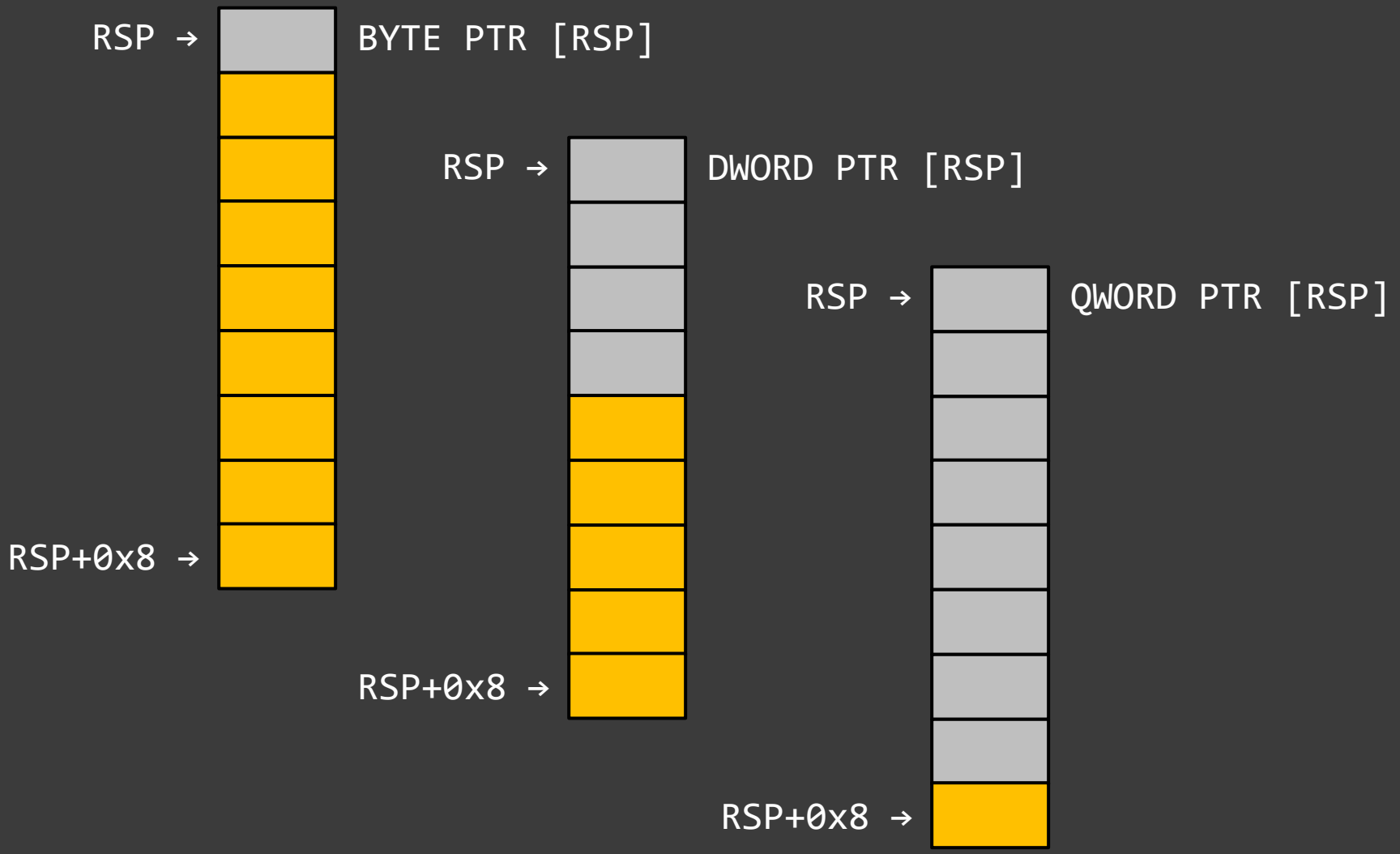
- `Opcode DST, SRC`
- Examples:

```
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; &func → RIP
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)
```

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
…
```

Parts 2C–2D: Practice Exercises

Links

- Memory Dumps:

Included in Exercise 0

- Exercise Transcripts:

Included in the book

Exercise 0

- **Goal:** Install WinDbg or Debugging Tools for Windows, or pull Docker image, 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

Exercise K1

- **Goal:** Learn how to get various information related to hardware, system, sessions, processes, threads, and modules
- **Patterns:** NULL Pointer (Data); False Effective Address; Invalid Pointer (General); Virtualized System (WOW64); Stack Trace Collection (Unmanaged Space); Unloaded Module
- \AWMDA-Dumps\Exercise-K1-Analysis-normal-kernel-dump-64.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-64.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)
- \AWMDA-Dumps\Exercise-K3-Analysis-kernel-dump-pool-corruption-64.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-64.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-64.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-64.pdf

Exercise K7

- **Goal:** Learn how to recognize stack overwrite and reconstruct stack trace
- **Patterns:** Truncated Stack Trace; NULL Pointer (Data); Execution Residue (Unmanaged Space, Kernel); Local Buffer Overflow (Kernel Space)
- \AWMDA-Dumps\Exercise-K7-Analysis-kernel-dump-stack-overwrite-64.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-64.pdf

BSOD Analysis Pattern Strategy

- Stack Trace
- Active Thread
- Rough Stack Trace
- Execution Residue
- Historical Information
- Unloaded Module
- Black Box
- Fault Context
- Multiple Exceptions
- Hidden Exception
- Exception Module
- Wild Pointer

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)
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)

Additional Pattern Links

ERESOURCE patterns and case studies

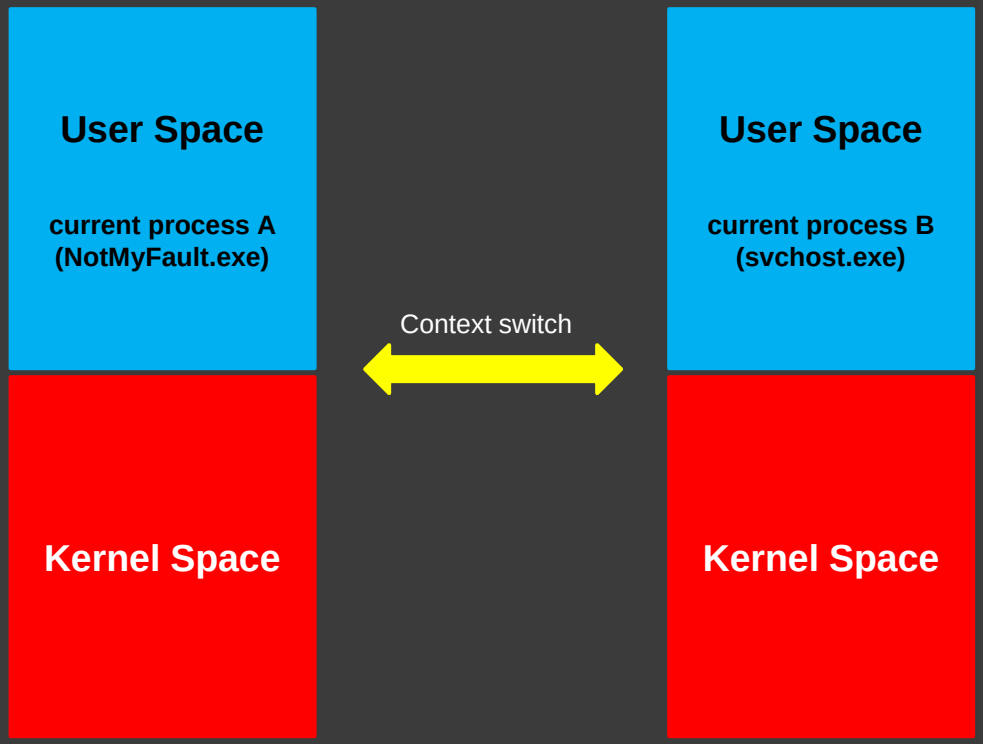
Wait Chain (Executive Resources) pattern is reprinted in this course from Memory Dump Analysis Anthology, Revised Edition, Volume 2, pages 147 – 150

Complete Memory Dumps

Exercises C1 – C5

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
...............................
```

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-64.pdf

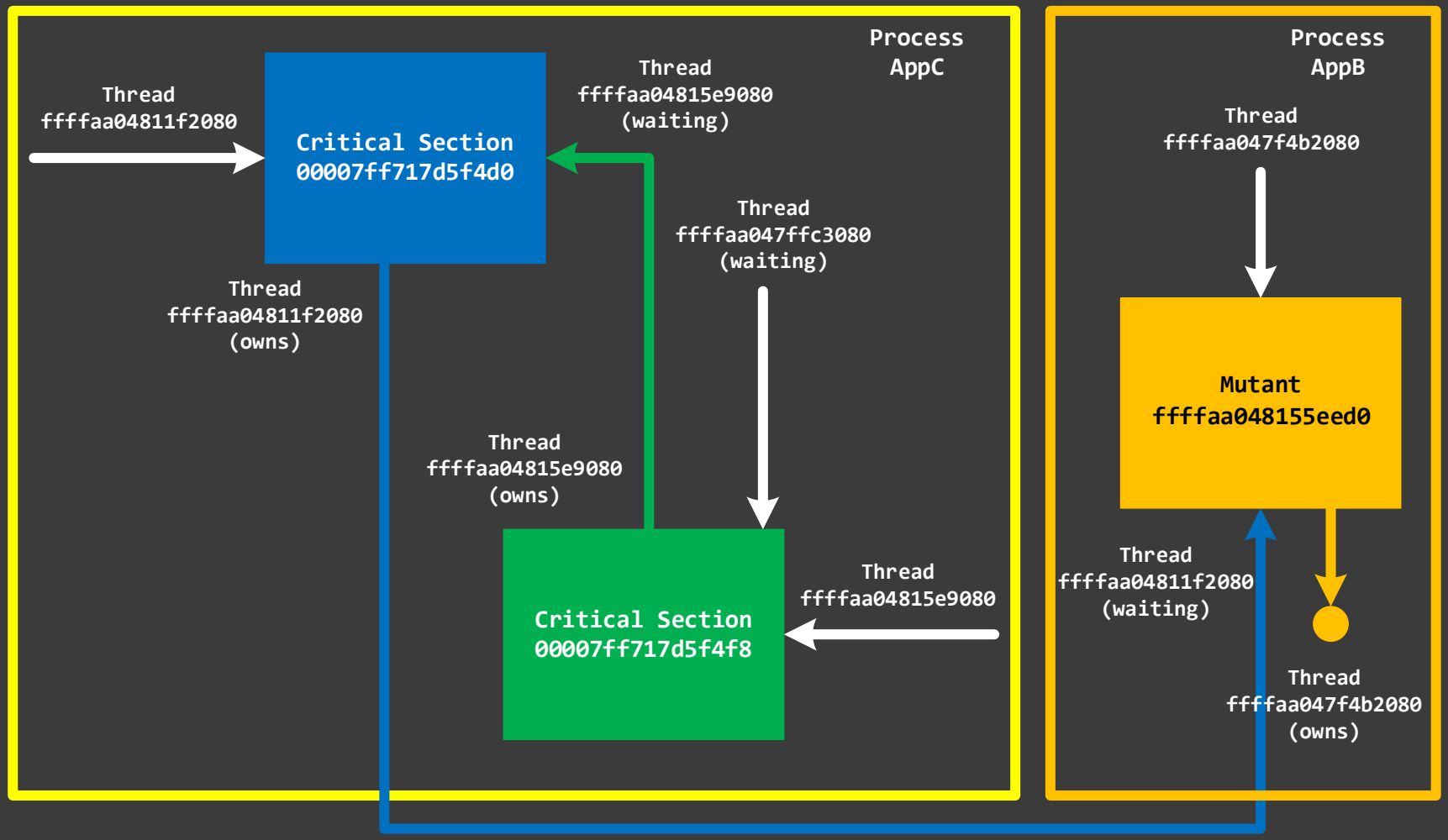
Exercise C2

- **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
- \AWMDA-Dumps\Exercise-C2-Analysis-problem-complete-dump-64.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-64.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-64.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-64.pdf

Pattern Links

Special Process
Spiking Thread
Message Box
Exception Stack Trace
Virtualized Process (WOW64)
Incorrect Stack Trace
Dialog Box
Frozen Process
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

Kernel Minidumps

Part 2E: Reading

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

Reprinted in this course

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

Pattern Classification

Space/Mode
Hooksware
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

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

Pattern Case Studies

More than 70 multiple pattern case studies:

http://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, 6th Edition, Part 1
- Practical Foundations of Windows Debugging, Disassembling, Reversing, 2nd Edition
- Advanced Windows Memory Dump Analysis with Data Structures, 4th Edition, Revised
- Accelerated .NET Core Memory Dump Analysis, Revised Edition
- Accelerated Windows Malware Analysis with Memory Dumps, 3rd Edition
- Accelerated Disassembly, Reconstruction and Reversing, 2nd Revised Edition
- Accelerated Windows Debugging4, 3rd Edition
- Extended Windows Memory Dump Analysis
- Accelerated Windows API for Software Diagnostics

Q&A

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

Thank you for attendance!