Public Preview Version

# Linux

## Core Dump Analysis

## Accelerated

Dmitry Vostokov
Software Diagnostics Services

# Prerequisites

Basic Linux troubleshooting

**GDB Commands**

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

# Training Goals

- Review fundamentals
- Learn how to collect core dumps
- Learn how to analyze core dumps

# Training Principles

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

# Schedule Summary

## Day 1

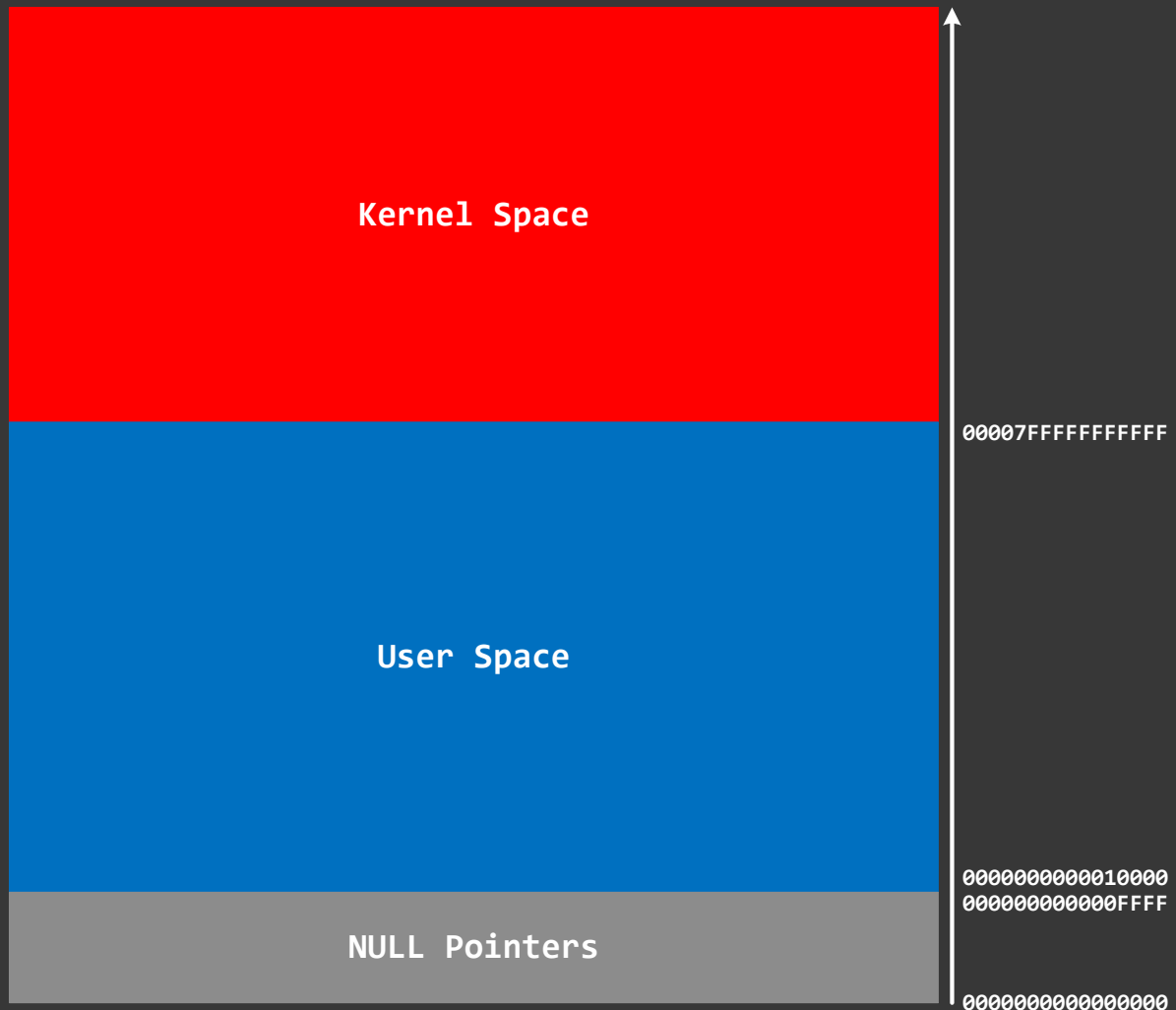
- Analysis Fundamentals (30 minutes)
- Core dump collection methods (10 minutes)
- Basic Core Memory Dumps (1 hour 20 minutes)

## Day 2

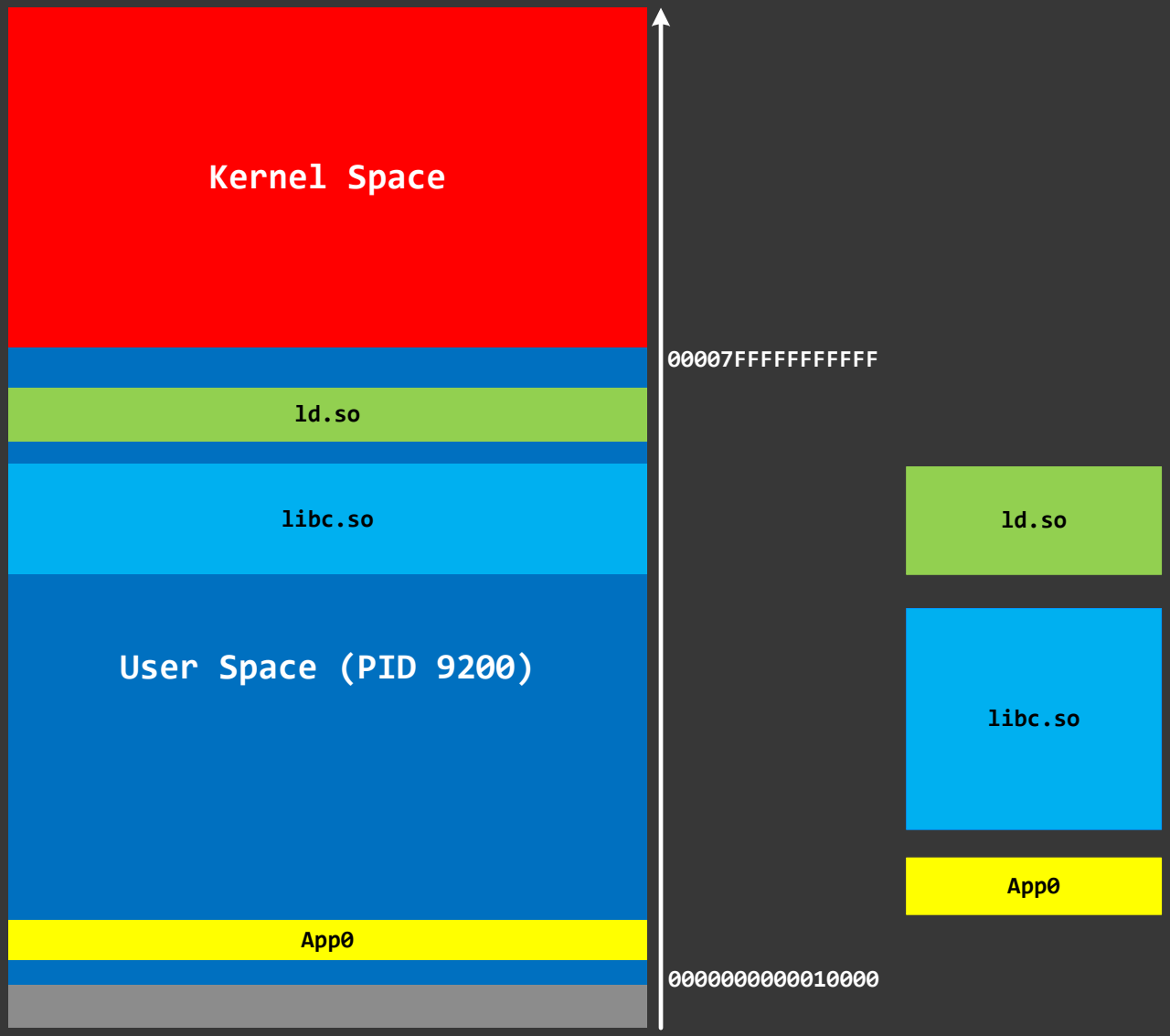
- Core Memory Dumps (2 hours)

# Part 1: Fundamentals

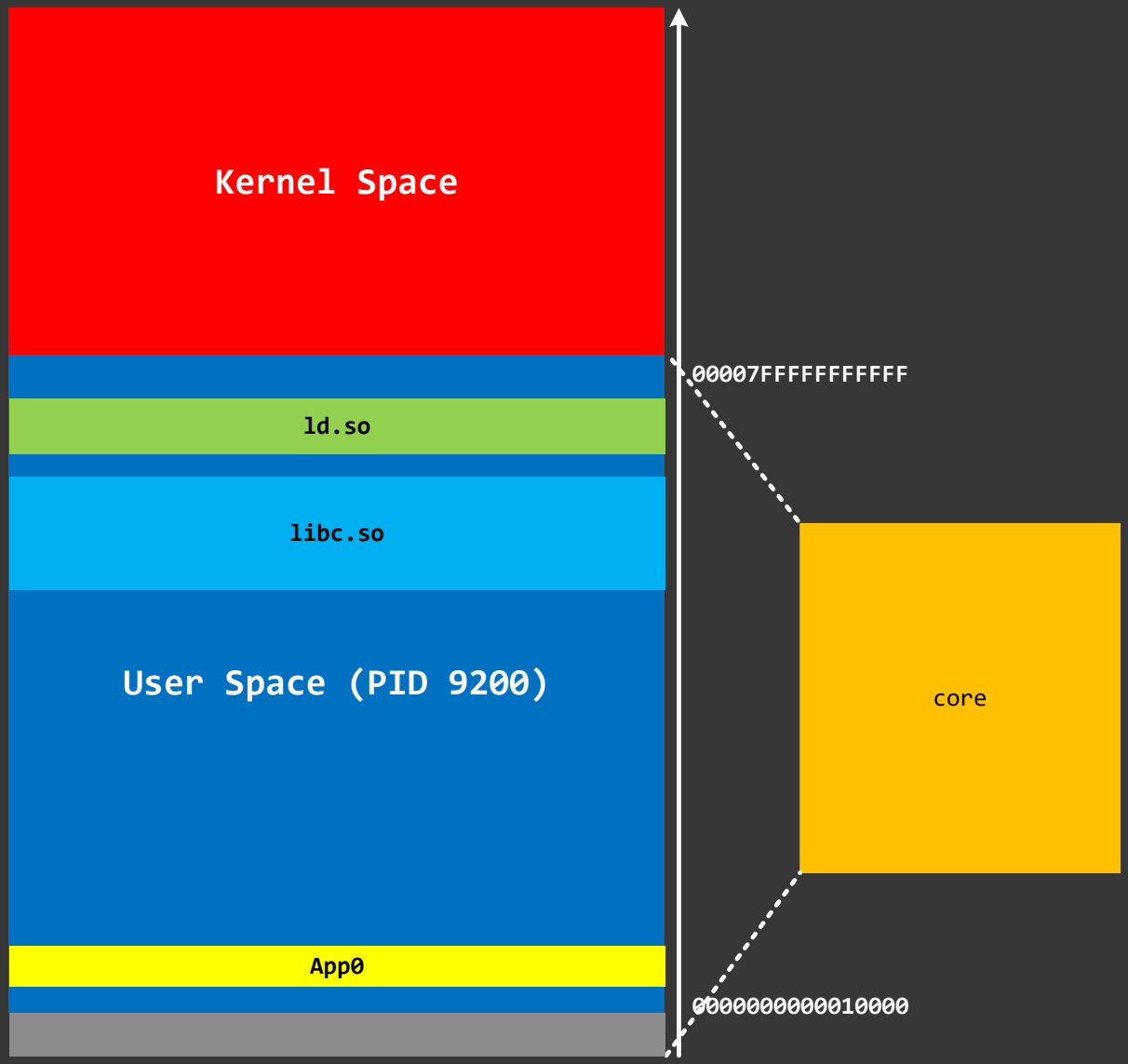
# Memory/Kernel/User Space



# App/Process/Library



# Process Memory Dump

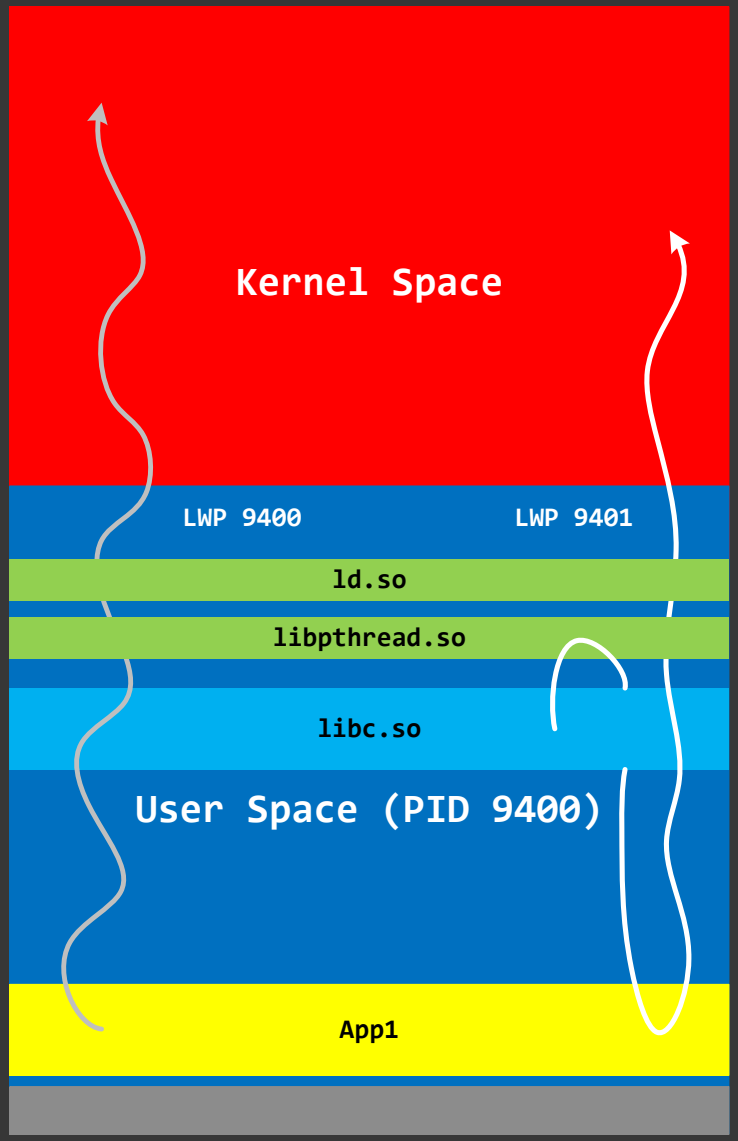



## GDB Commands

**info sharedlibrary**
Lists dynamic libraries

**maintenance info sections**
Lists memory regions

# Lightweight Processes (Threads)



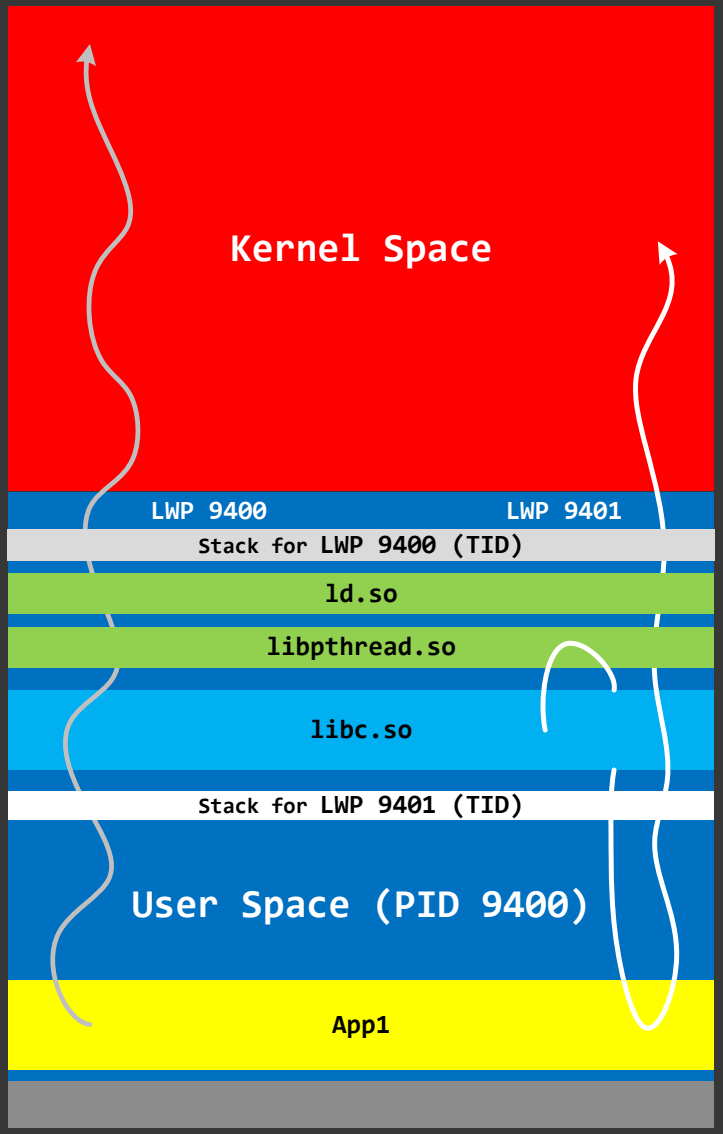

## GDB Commands

**info threads**
Lists threads

**thread <n>**
Switches between threads

**thread apply all bt**
Lists stack traces from all threads

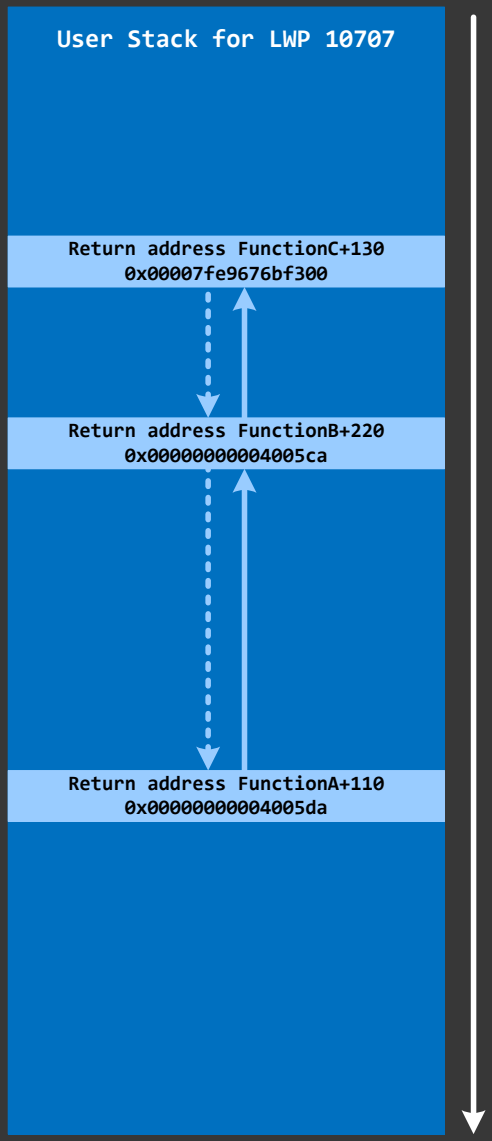
# Thread Stack Raw Data




## GDB Commands

**x/<n>a <address>**
Prints n addresses with corresponding symbol mappings if any

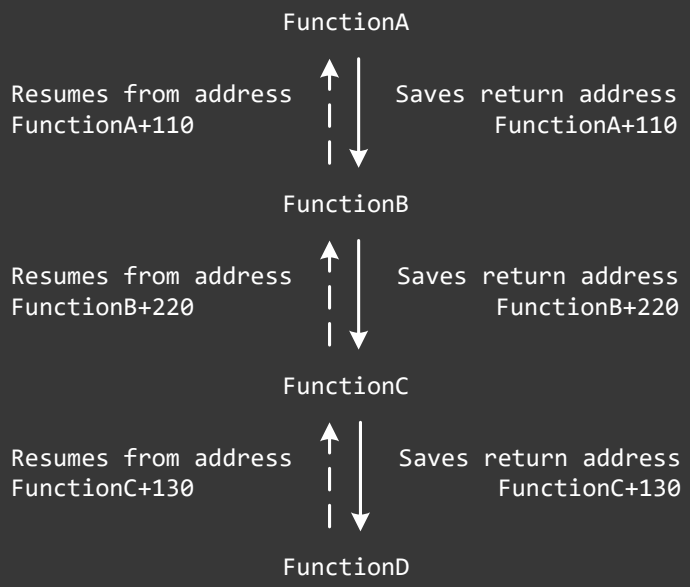
# Thread Stack Trace




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

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

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




## GDB Commands

```
(gdb) bt
#0 0x00007fe9676bf48d in FunctionD ()
#1 0x00007fe9676bf300 in FunctionC ()
#2 0x00000000004005ca in FunctionB ()
#3 0x00000000004005da in FunctionA ()
```

# GDB vs. WinDbg

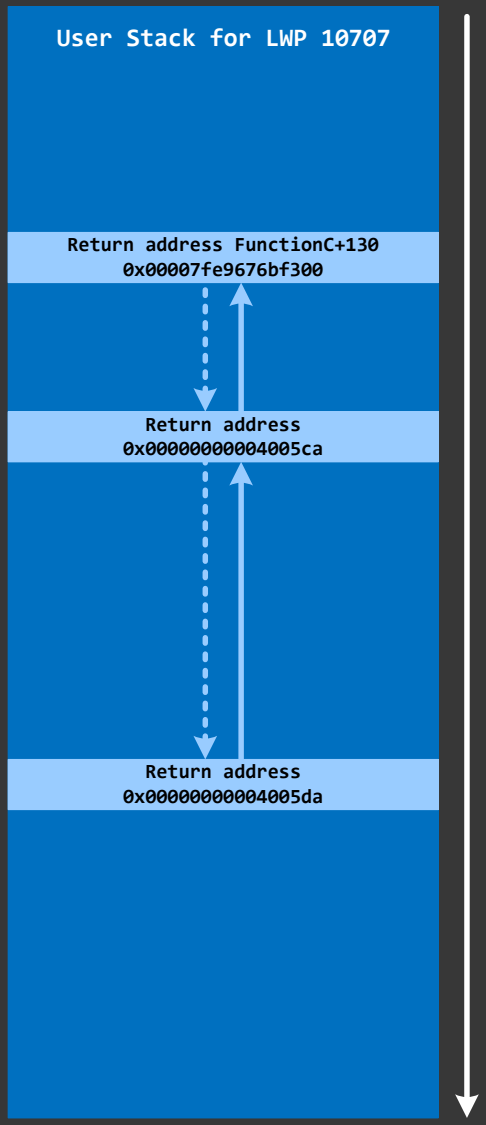
## GDB Commands

```
(gdb) bt
#0 0x00007fe9676bf48d in FunctionD ()
#1 0x00007fe9676bf300 in FunctionC ()
#2 0x00000000004005ca in FunctionB ()
#3 0x00000000004005da in FunctionA ()
```

## WinDbg Commands

```
0:000> kn
00 00007fe9676bf300 Module!FunctionD+offset
01 00000000004005ca Module!FunctionC+130
02 00000000004005da AppA!FunctionB+220
03 0000000000000000 AppA!FunctionA+110
```

# Thread Stack Trace (no symbols)



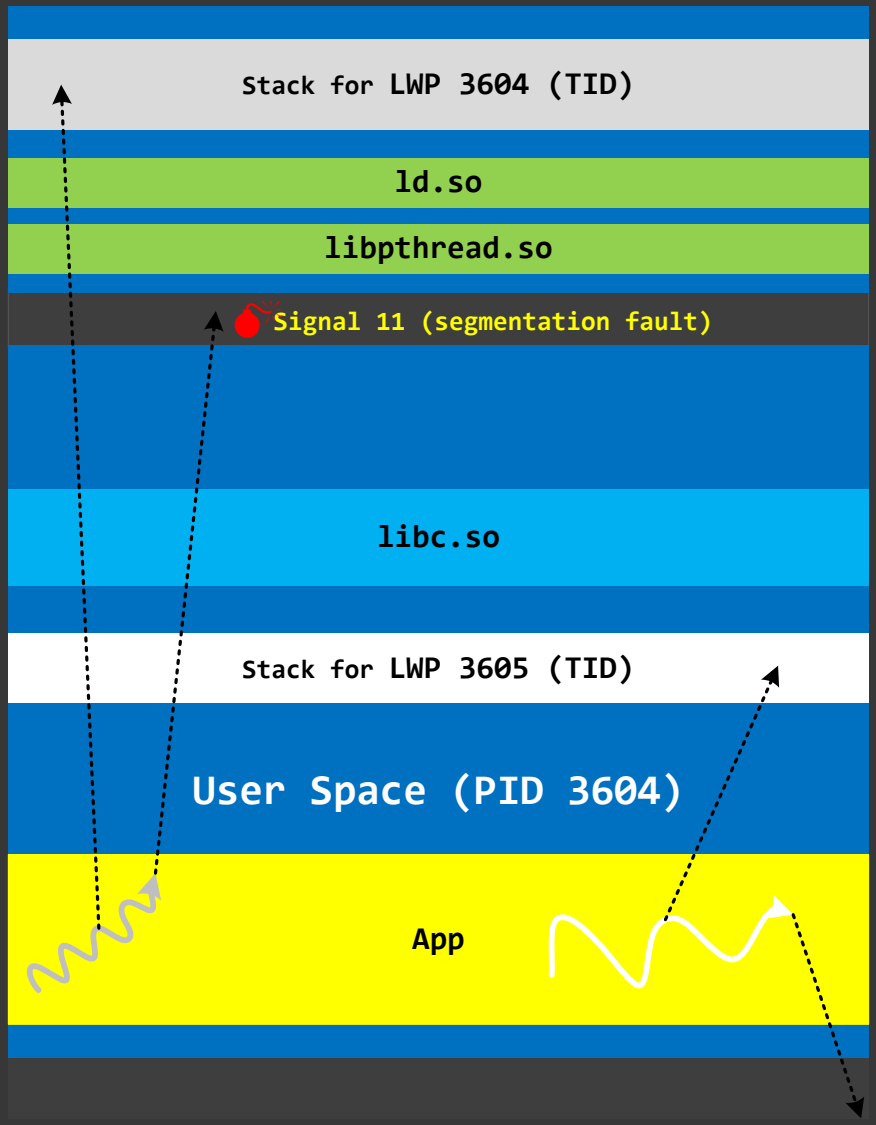

**Symbol file App.sym**

**FunctionA 22000 - 23000**
**FunctionB 32000 - 33000**

## GDB Commands

```
(gdb) bt
#0 0x00007fe9676bf48d in FunctionD ()
#1 0x00007fe9676bf300 in FunctionC ()
#2 0x00000000004005ca in ?? ()
#3 0x00000000004005da in ?? ()
```

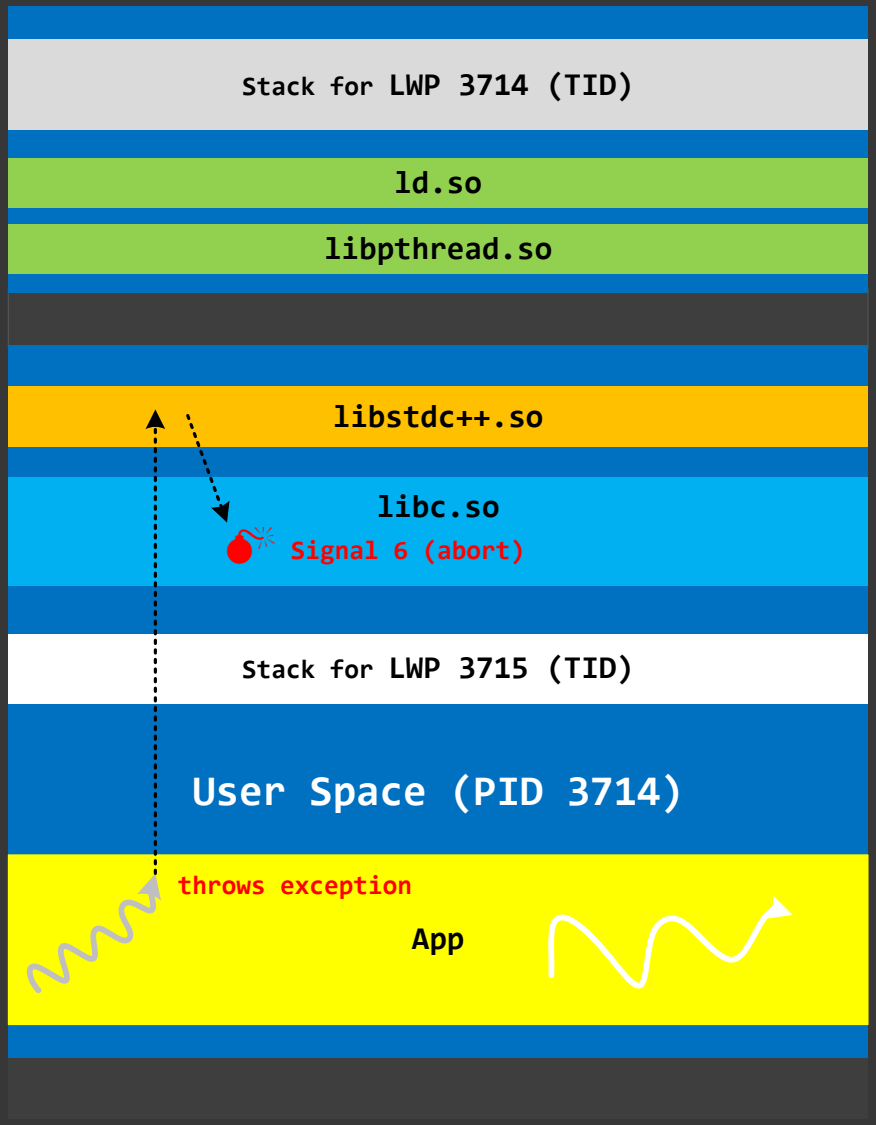
# Exceptions (Access Violation)




## GDB Commands

```
(gdb) x <address>
0x<address>: Cannot access
memory at address 0x<address>
```

# Exceptions (Runtime)



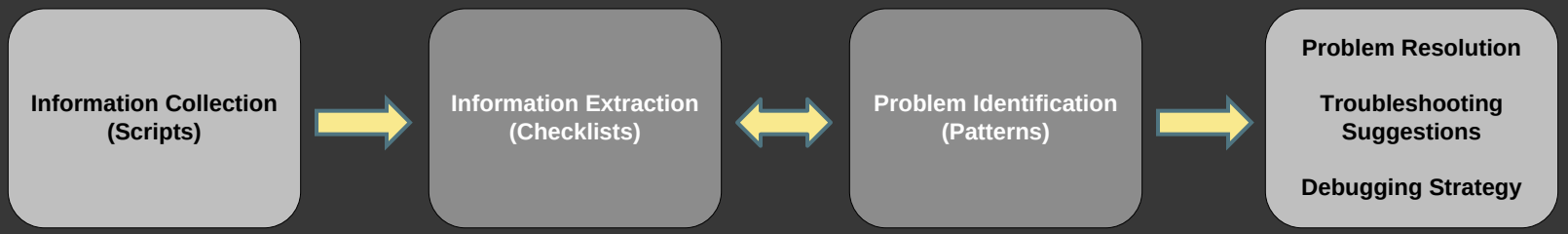
# Pattern-Oriented Diagnostic Analysis

**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, ...



# Part 2: Core Dump Collection

# Enabling Collection

- Temporary for the current user:

```
$ ulimit -c unlimited
```

- Permanent for every user except root:

Edit the file: `/etc/security/limits.conf`

Add or uncomment the line:

```
*      soft   core   unlimited
```

To limit root to 1GB add or uncomment this line:

```
root   hard   core   1000000
```

# Generation Methods

- kill (requires ulimit):

```
$ kill –s SIGQUIT PID
$ kill –s SIGABRT PID
```

- gcore:

```
$ gcore PID
```

# Part 3: Practice Exercises

# Links

- Memory Dumps:

NOT IN THE PUBLIC PREVIEW VERSION

- Exercise Transcripts:

NOT IN THE PUBLIC PREVIEW VERSION

# Exercise 0

- **Goal:** Install GDB and check if GDB loads a core dump correctly
- **Patterns:** Incorrect Stack Trace
- \ALCDA-Dumps\Exercise-A0.pdf

# Process Core Dumps

Exercises A1-A12

# Exercise A1

- **Goal:** Learn how to list stack traces, disassemble functions, check their correctness, dump data, get environment
- **Patterns:** Manual Dump, Stack Trace, Stack Trace Collection, Annotated Disassembly, Paratext, Not My Version, Environment Hint
- \ALCDA-Dumps\Exercise-A1.pdf

# Exercise A2D

- **Goal:** Learn how to identify exceptions, find problem threads and CPU instructions
- **Patterns:** NULL Pointer (data), Active Thread
- \ALCDA-Dumps\Exercise-A2D.pdf

# Exercise A2C

- **Goal:** Learn how to identify exceptions, find problem threads and CPU instructions
- **Patterns:** NULL Pointer (code), Active Thread
- \ALCDA-Dumps\Exercise-A2C.pdf

# Exercise A3

- **Goal:** Learn how to identify spiking threads
- **Patterns:** Spiking Thread
- \ALCDA-Dumps\Exercise-A3.pdf

# Exercise A4

- **Goal:** Learn how to identify heap regions and heap corruption
- **Patterns:** Heap Corruption
- \ALCDA-Dumps\Exercise-A4.pdf

# Exercise A5

- **Goal:** Learn how to identify stack corruption
- **Patterns:** Local Buffer Overflow, Execution Residue
- \ALCDA-Dumps\Exercise-A5.pdf

# Exercise A6

- **Goal:** Learn how to identify stack overflow, stack boundaries, reconstruct stack trace
- **Patterns:** Stack Overflow, Execution Residue
- \ALCDA-Dumps\Exercise-A6.pdf

# Exercise A7

- **Goal:** Learn how to identify active threads
- **Patterns:** Divide by Zero, Active Thread
- \ALCDA-Dumps\Exercise-A7.pdf

# Exercise A8

- **Goal:** Learn how to identify runtime exceptions, past execution residue and stack traces, identify handled exceptions
- **Patterns:** C++ Exception, Execution Residue, Coincidental Symbolic Information, Handled Exception
- \ALCDA-Dumps\Exercise-A8.pdf

# Exercise A9

- **Goal:** Learn how to identify heap leaks
- **Patterns:** Heap Leak, Execution Residue, Module Hint
- \ALCDA-Dumps\Exercise-A9.pdf

# Exercise A10

- **Goal:** Learn how to identify heap contention wait chains, synchronization issues, advanced disassembly, dump arrays
- **Patterns:** Double Free, Heap Contention, Wait Chain, Critical Region, Self-Diagnosis
- \ALCDA-Dumps\Exercise-A10.pdf

# Exercise A11

- **Goal:** Learn how to identify synchronization wait chains, deadlocks, hidden and handled exceptions
- **Patterns:** Wait Chains, Deadlock, Execution Residue, Handled Exception
- \ALCDA-Dumps\Exercise-A11.pdf

# Exercise A12

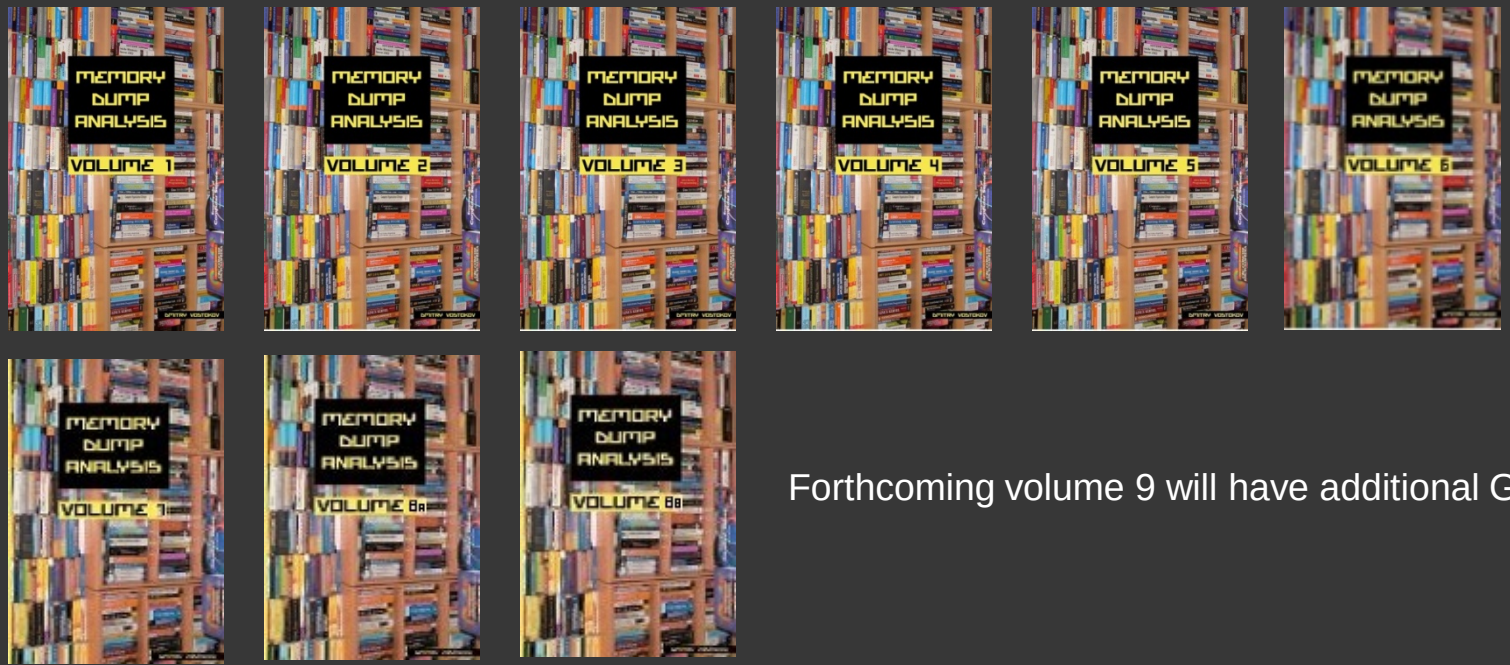
- **Goal:** Learn how to dump memory for post-processing, get the list of functions and module variables, load symbols, inspect arguments and local variables
- **Patterns:** Module Variable
- \ALCDA-Dumps\Exercise-A12.pdf

# Pattern Links (Linux and GDB)

Active Thread
C++ Exception
Critical Region
Divide by Zero
Execution Residue
Heap Contention
Heap Leak
Local Buffer Overflow
Module Hint
Not My Version
NULL Pointer (data)
Self-Diagnosis
Stack Overflow
Stack Trace Collection
Annotated Disassembly
Coincidental Symbolic Information
Deadlock
Environment Hint
Handled Exception
Heap Corruption
Lateral Damage
Manual Dump
Module Variable
NULL Pointer (code)
Paratext
Spiking Thread
Stack Trace
Wait Chain

# Resources

- Software Diagnostics Institute
- Pattern-Driven Software Diagnostics
- Pattern-Based Software Diagnostics
- Debugging TV
- Rosetta Stone for Debuggers
- Accelerated Mac OS X Core Dump Analysis
- GDB Pocket Reference
- Memory Dump Analysis Anthology (some articles in volumes 1 and 7 cover GDB)




Forthcoming volume 9 will have additional GDB articles

# Q&A

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

Thank you for attendance!