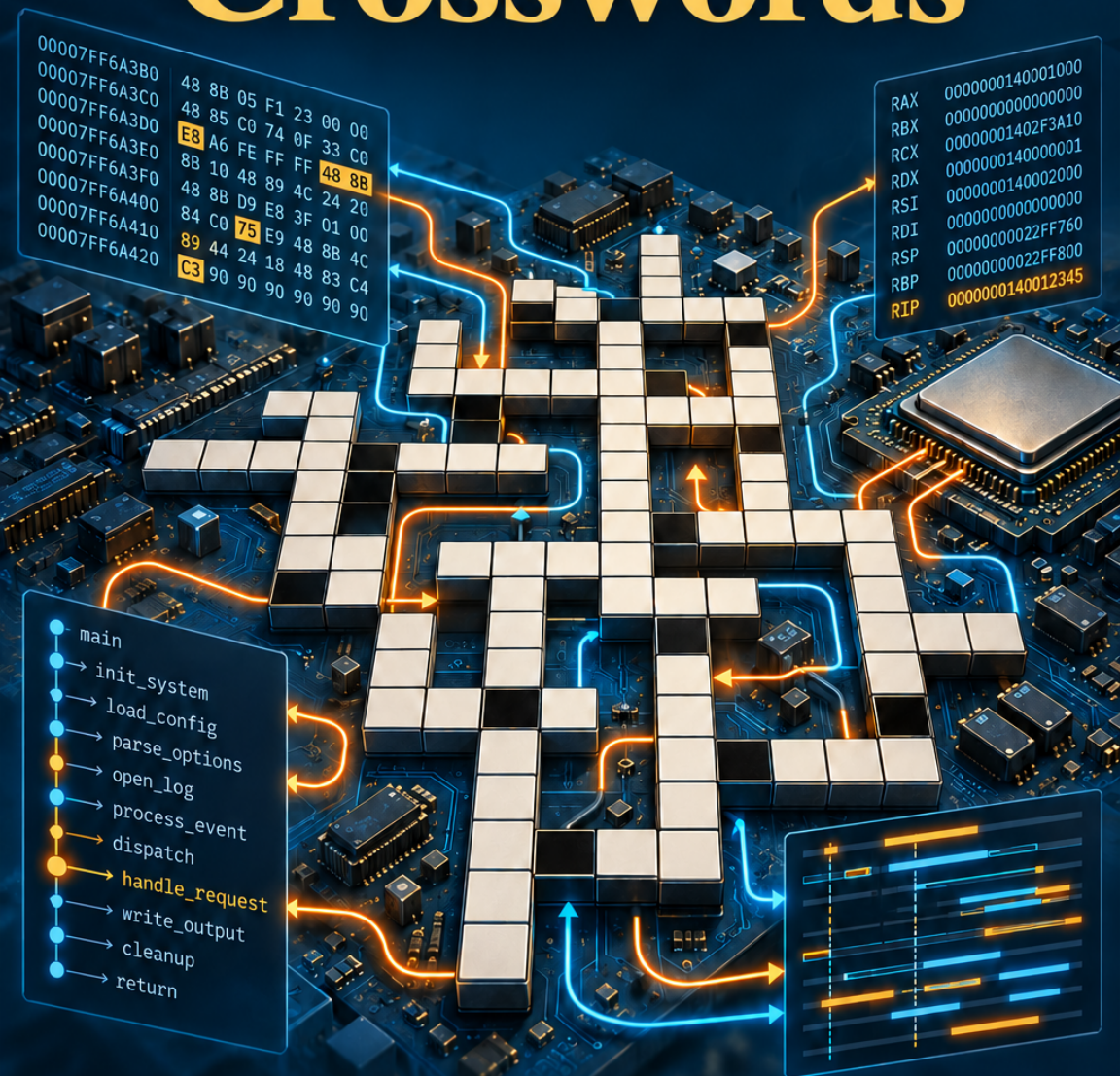


Software Diagnostics, Observability, and Debugging Crosswords



Concept by Dmitry Vostokov
DumpAnalysis.org + TraceAnalysis.org

Software Diagnostics, Observability, and Debugging Crosswords

Version 9

Developed with AI assistance (GPT-6 Astra Max and Fable 5.1 Extra)

Twenty-four crosswords cover Windows, Linux, and macOS memory dump vocabulary, analysis patterns, trace and log analysis, WinDbg, GDB and LLDB commands, live debugging, x86, x64, and ARM64 instructions and registers, and observability.

Each puzzle is followed by its clues. Reference notes accompany each part. All answer keys are collected at the end of the book.

Concept by Dmitry Vostokov

DumpAnalysis.org + TraceAnalysis.org

Published by OpenTask, Republic of Ireland

Copyright © 2026 by Dmitry Vostokov

Copyright © 2026 by OpenTask

ISBN-13: 978-1-919135-04-5

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without the publisher's prior written permission.

Product and company names mentioned in this book may be trademarks of their owners.

OpenTask books are available through booksellers and distributors worldwide. For further information or comments, send requests to press@opentask.com.

Contents

1 Memory Dump Analysis Crosswords	5
1 Windows memory dump vocabulary	6
2 Linux memory dump vocabulary	8
3 macOS memory dump vocabulary	10
4 Memory dump analysis patterns	12
2 Trace and Log Analysis Crosswords	16
5 Trace and log fundamentals	17
6 Distributed tracing	19
7 Trace analysis patterns	21
3 WinDbg Command Crossword	24
8 WinDbg commands	25
4 GDB Command Crossword	28
9 GDB core analysis commands	29
5 LLDB Memory Dump Analysis Crossword	32
10 LLDB memory dump analysis commands	33
6 GDB Live Debugging Crossword	36
11 GDB live debugging commands	37
7 WinDbg Live Debugging Crossword	40
12 WinDbg live debugging commands	41

Contents continued

8 LLDB Live Debugging Crossword	44
13 LLDB live debugging commands	45
9 Live Debugging Concepts Crosswords	48
14 Established terminology.....	49
15 Unified debugging patterns.....	51
10 x86 Crash Dump Crosswords	55
16 x86 instructions.....	56
17 x86 registers	58
11 x64 Crash Dump Crosswords	61
18 x64 instructions.....	62
19 x64 registers	64
12 ARM64 Crash Dump Crosswords	67
20 ARM64 instructions	68
21 ARM64 registers and roles	70
13 Observability Crosswords	73
22 Observability foundations	74
23 Metrics and reliability	76
24 Tracing and telemetry pipelines.....	78
Answer keys	82

The answer keys follow the same puzzle order.

Memory Dump Analysis Crosswords

Four practice puzzles with separate answer keys

Three vocabulary crosswords cover Windows memory dumps, Linux core files and macOS core files. A fourth crossword covers memory dump analysis patterns whose names are independent of the operating system and debugger. Each puzzle has 14 clues.

Puzzle 1 Windows memory dump vocabulary

Windows memory structures, execution state and dump evidence.

Puzzle 2 Linux memory dump vocabulary

ELF core files, signals, memory mappings and saved process state.

Puzzle 3 macOS memory dump vocabulary

Mach-O core files, Mach resources, program images and debug symbols.

Puzzle 4 Memory dump analysis patterns

Diagnostic scenarios identified by published pattern names.

How to solve

1. Write one capital letter in each white square. Shaded squares are unused.
2. Answers run left to right or top to bottom. Each answer crosses another answer.
3. Omit spaces and hyphens. Word-part lengths appear in parentheses; (4,5) uses nine squares.
4. Enter vocabulary or pattern names. WinDbg, GDB and LLDB command practice has separate parts of this collection.

Answer keys: 83–86. Reference notes begin on page 14.

Windows and Linux reference notes

The platform labels identify the vocabulary context. Shared terms such as thread, stack and register can occur in more than one puzzle; they are not exclusive to one operating system.

Here, minidump means the Windows user-mode dump format. Capture flags determine its contents; full process memory can be included.

The Linux puzzle concerns user-space ELF core files. Core-dumping signals do not guarantee a saved file: resource limits, permissions and collection settings can prevent it. Capture filters can omit memory mappings. A core file is not a running process.

Primary references

New vocabulary answers and all pattern answers in the digital keys link to primary references.

Microsoft Learn. About Processes and Threads.

<https://learn.microsoft.com/en-us/windows/win32/procthread/about-processes-and-threads>

Microsoft Learn. MINIDUMP_TYPE enumeration.

https://learn.microsoft.com/en-us/windows/win32/api/minidumpapiset/ne-minidumpapiset-minidump_type

Linux man-pages via Debian. core 5.

<https://manpages.debian.org/trixie/manpages/core.5.en.html>

Linux man-pages via Debian. signal 7.

<https://manpages.debian.org/trixie/manpages/signal.7.en.html>

Linux man-pages via Debian. elf 5.

<https://manpages.debian.org/trixie/manpages/elf.5.en.html>

DWARF Committee. Debugging Information Format.

<https://dwarfstd.org/>

GNU GDB manual. Commands to Specify Files.

<https://sourceware.org/gdb/download/onlinedocs/gdb.html/Files.html>

After solving

Choose one shared term and one platform-specific term. Explain what each would let you interpret in a captured dump.

References checked 10 October 2026. Clues and grid arrangements are original.

macOS and pattern reference notes

The macOS puzzle concerns native Mach-O core files and the images and symbols needed to interpret them. A text or JSON crash report summarizes selected evidence; it is not a binary core file with captured memory. A dSYM supplies debug information, not the process memory image.

Match each image with its build UUID and corresponding dSYM. Interpret addresses using the recorded load locations. The clues distinguish Mach resource ownership in a task from execution in its threads.

Pattern names follow Dmitry Vostokov's published catalogue and are kept separate from operating-system vocabulary and debugger commands. Their manifestations and available evidence depend on the target and capture. Some clues describe one common manifestation of a broader pattern.

A waiting thread needs interpretation in its application context; a single busy frame does not establish excessive CPU consumption.

Primary references

The pattern answer key links each name to its individual source page. Platform references support the vocabulary, not a platform-specific naming scheme for patterns.

Apple. Mach Overview.

<https://developer.apple.com/library/archive/documentation/Darwin/Conceptual/KernelProgramming/Mach/Mach.html>

Apple open source. Mach-O loader definitions.

https://github.com/apple-oss-distributions/xnu/blob/main/EXTERNAL_HEADERS/mach-o/loader.h

Apple WWDC21. Symbolication Beyond the Basics.

<https://developer.apple.com/videos/play/wwdc2021/10211/>

Apple. Overview of Dynamic Libraries.

<https://developer.apple.com/library/archive/documentation/DeveloperTools/Conceptual/DynamicLibraries/100-Articles/OverviewOfDynamicLibraries.html>

Dmitry Vostokov. Crash Dump Analysis Patterns.

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

After solving

Choose three patterns. For each, name a supporting observation, an alternative explanation and an additional dump check. Consider how you would obtain equivalent evidence on another operating system.

References checked 10 October 2026. Clues and grid arrangements are original.

Trace and Log Analysis

Crosswords

Three practice puzzles with separate answer keys

Three crosswords for practicing the vocabulary of logs, distributed tracing and pattern-oriented software diagnostics. Each puzzle has 14 clues. The set progresses from log fundamentals to OpenTelemetry concepts and diagnostic patterns.

Puzzle 5 Trace and log fundamentals

Log fields, text formats and record processing.

Puzzle 6 Distributed tracing

Fourteen concepts used to follow work across service boundaries.

Puzzle 7 Trace analysis patterns

Short diagnostic scenarios drawn from the published pattern catalogue.

How to solve

1. Write one capital letter in each white square. Shaded squares are unused.
2. Answers run left to right or top to bottom. Each answer crosses at least one other answer.
3. Numbers in parentheses show word lengths. For example, (4,5) means nine squares with no space.
4. Omit spaces and hyphens. Enter abbreviations one letter per square. The answer key restores readable spelling.

Answer keys: 87–89. Reference notes begin on page 23.

Reference notes

Here, trace covers both diagnostic message sequences and span-based distributed traces. Puzzle 6 follows OpenTelemetry terminology; Puzzle 7 uses Dmitry Vostokov's published pattern names.

In Puzzle 5, Message is the general term for a log's human-readable text; OpenTelemetry stores such text in Body, which can also hold structured data.

An event timestamp can differ from the time a collector observes it. Buffering and clock differences can affect apparent order; timestamps across sources alone do not establish causality.

Sparse Trace concerns insufficient tracing statements. Silent Messages describes possible entries in empty time slots. Missing Message concerns a particular expected entry that is absent.

An error also seen in healthy runs is not automatically harmless. Compare its context and recurrence before deciding whether it explains the incident.

Primary references

Answers in the digital answer keys link to their individual source pages. The starting points below also work for readers using a printed copy.

OpenTelemetry. Logs Data Model.

<https://opentelemetry.io/docs/specs/otel/logs/data-model/>

OpenTelemetry. Traces.

<https://opentelemetry.io/docs/concepts/signals/traces/>

OpenTelemetry. Context propagation.

<https://opentelemetry.io/docs/concepts/context-propagation/>

Dmitry Vostokov. Trace Analysis Patterns.

<https://www.dumpanalysis.org/blog/index.php/trace-analysis-patterns/>

After solving

Choose three diagnostic patterns. For each, name one observation that would support it, one alternative explanation, and one additional check you would make in the trace or log.

References checked 10 October 2026. Clues and grid arrangements are original.

WinDbg Command Crossword

Fourteen clues with a separate answer key

Fourteen commands and extensions for inspecting Windows memory dumps with WinDbg. The puzzle covers exception context, symbols, process structures and captured diagnostic data.

Memory dump inspection

Recognize the command or extension from the evidence it displays or the analysis context it selects.

How to solve

1. Write one capital letter in each white square. Shaded squares are unused.
2. Answers run left to right or top to bottom. Each answer crosses at least one other answer.
3. Use full command spellings. Do not use abbreviations or aliases.
4. Omit leading dots, exclamation marks, options and arguments. Numbers in parentheses show the letter count. The answer key restores the command spelling.

Answer keys: 90. Reference notes begin on page 27.

Reference notes

Command output depends on the target, symbols and captured data. For user-mode dumps, `.dump /mt` includes thread times for `!runaway`; `/mh` includes handle data for `!handle`; `/ma` includes both.

The command `.excr` is a documented synonym for `.ecx`. Crossing letters in this puzzle select ECXR.

WinDbg has several extension commands named `!locks`. This puzzle uses the user-mode `Ntsdexts.dll` version, which displays critical sections.

The crossword omits command punctuation and arguments. The answer key restores the leading dot or exclamation mark; use the linked documentation for the full syntax.

Primary references

The digital answer key links each command to its primary documentation. The references below also provide starting points for readers using a printed copy.

Microsoft Learn. Debugger Commands.

<https://learn.microsoft.com/en-us/windows-hardware/drivers/debuggercmds/debugger-commands>

Microsoft Learn. `.dump` Create Dump File.

<https://learn.microsoft.com/en-us/windows-hardware/drivers/debuggercmds/-dump--create-dump-file->

Microsoft Learn. `.ecx` Display Exception Context Record.

<https://learn.microsoft.com/en-us/windows-hardware/drivers/debuggercmds/-ecx--display-exception-context-record->

Microsoft Learn. `!locks` `Ntsdexts.locks`.

<https://learn.microsoft.com/en-us/windows-hardware/drivers/debuggercmds/-locks---ntsdexts-locks->

After solving

Choose three commands. For each, identify the evidence you expect, the symbols or captured data it needs, and what a missing or empty result would mean.

References checked 10 October 2026. Clues and grid arrangements are original.

GDB Command Crossword

Fourteen clues with a separate answer key

Fourteen GDB commands for inspecting core files and saved program state. The puzzle covers stack frames, threads, registers, values, types, shared libraries and machine code.

Core file inspection

Recognize the command from the state it inspects. The type-inspection clues use C and C++ terminology.

How to solve

1. Write one capital letter in each white square. Shaded squares are unused.
2. Answers run left to right or top to bottom. Each answer crosses at least one other answer.
3. Use full command spellings. Do not use abbreviations or aliases.
4. For multiword commands, include the subcommand but omit spaces. Omit options and arguments. For example, (4,5) means nine squares with no space. The answer key restores readable spelling.

Answer keys: 91. Reference notes begin on page 31.

Reference notes

Commands inspect the selected target, thread and frame. Postmortem analysis needs a matching executable and core file; a core file does not provide a running process.

Results depend on captured memory and registers, debug information and matching shared libraries. Optimized code can make some variables unavailable.

Register values are relative to the selected frame. Select frame 0 to inspect the stopped thread's hardware register contents where they are available. Type inspection uses debug information; it does not reconstruct missing type definitions.

Primary references

The digital answer key links each command to its primary documentation. The references below also provide starting points for readers using a printed copy.

GNU GDB manual. Backtraces.

<https://sourceware.org/gdb/download/onlinedocs/gdb.html/Backtrace.html>

GNU GDB manual. Selecting a Frame.

<https://sourceware.org/gdb/download/onlinedocs/gdb.html/Selection.html>

GNU GDB manual. Information About a Frame.

<https://sourceware.org/gdb/download/onlinedocs/gdb.html/Frame-Info.html>

GNU GDB manual. Registers.

<https://sourceware.org/gdb/download/onlinedocs/gdb.html/Registers.html>

GNU GDB manual. Debugging Programs with Multiple Threads.

<https://sourceware.org/gdb/download/onlinedocs/gdb.html/Threads.html>

GNU GDB manual. Examining Data.

<https://sourceware.org/gdb/download/onlinedocs/gdb.html/Data.html>

GNU GDB manual. Examining the Symbol Table.

<https://sourceware.org/gdb/download/onlinedocs/gdb.html/Symbols.html>

GNU GDB manual. Machine Code.

<https://sourceware.org/gdb/download/onlinedocs/gdb.html/Machine-Code.html>

GNU GDB manual. Commands to Specify Files.

<https://sourceware.org/gdb/download/onlinedocs/gdb.html/Files.html>

After solving

Choose one thread command, one frame command and one data command. Explain how the selected thread and frame affect the evidence each command can display.

References checked 11 October 2026; GDB release manual: 18.1. Clues and grid arrangements are original.

LLDB Memory Dump Analysis

Crossword

Fourteen clues with a separate answer key

Fourteen LLDB commands for memory dump analysis using supported core files. The puzzle covers target setup, loaded images, threads, frames, saved registers, memory and debug information.

Core file inspection

Recognize the command from its evidence or inspection context. The clues use complete command paths. LLDB defines image as an alias of target modules.

How to solve

1. Write one capital letter in each white square. Shaded squares are unused.
2. Answers run left to right or top to bottom. Each answer crosses another answer.
3. Use full command and subcommand names. Omit spaces, hyphens, options and arguments; do not use aliases.
4. Word-part lengths appear in parentheses. For example, (6,4,2) uses twelve squares. The key restores spaces, hyphens and lowercase spelling.

Answer keys: 92. Reference notes begin on page 35.

Reference notes

A supported core file must be loaded for this puzzle. Support depends on LLDB's target and core-file plugins. The snapshot cannot resume execution or call functions. Use the live debugging crossword for execution-control commands.

With a matching executable available, target create --core <core-file> <executable> creates the postmortem target. Inspect the recorded state with matching program images and debug information; on macOS this can include matching dSYM bundles.

Results depend on captured memory, register records and debug information. Frame 0 exposes the selected thread's saved hardware state where available; older frames use recovered register values. Optimized or uncaptured variables can be unavailable.

The variable clues concern ordinary data inspection. Expressions or object descriptions that require target execution cannot be evaluated as if a core file were a live process. A missing value is not evidence that it was zero.

Primary references

The digital answer key links each command to its official documentation or implementation help.

LLVM LLDB. GDB to LLDB Command Map.

<https://lldb.llvm.org/use/map.html>

LLVM LLDB. Tutorial.

<https://lldb.llvm.org/use/tutorial.html>

LLVM LLDB. Target command definitions.

https://lldb.llvm.org/cpp_reference/CommandObjectTarget_8cpp_source.html

LLVM LLDB. Frame command definitions.

https://lldb.llvm.org/cpp_reference/CommandObjectFrame_8cpp_source.html

LLVM LLDB. Type command definitions.

https://lldb.llvm.org/cpp_reference/CommandObjectType_8cpp_source.html

LLVM LLDB. Symbolication.

<https://lldb.llvm.org/use/symbolication.html>

After solving

Choose a thread, a frame and a data-inspection command. Explain what each selects or reveals, and which missing capture or symbol information could limit the result.

References checked 11 October 2026. Clues and grid arrangements are original.

GDB Live Debugging Crossword

Fourteen clues with a separate answer key

Fourteen GDB commands for live debugging of a local process. The crossword covers starting and attaching, continuing, source and instruction stepping, breakpoints, watchpoints and detaching.

Live debugging practice

Recognize the command from its effect on execution. The clues assume normal forward execution, default repeat counts and usable source line information where needed.

How to solve

1. Write one capital letter in each white square. Shaded squares are unused.
2. Answers run left to right or top to bottom. Each answer crosses at least one other answer.
3. Enter the command itself. Numbers in parentheses show its letter count.
4. Use the full spellings shown in the manual, without aliases or abbreviations. Omit options and arguments. The answer key restores lowercase command spelling.

Answer keys: 93. Reference notes begin on page 39.

Reference notes

The launch clues assume native local process debugging with an executable selected. Some remote targets do not support run. The start command aims for the main procedure; a breakpoint or startup event can stop execution earlier.

Source stepping depends on line information. Under its default setting, step can pass over functions without it. Machine-instruction stepping does not require source lines. Breakpoints and signals can interrupt execution before a stepping command reaches its intended endpoint.

The finish command executes the function until it returns. It differs from return, which forces a return without executing the remaining function body.

The watch command detects a changed expression value. A write that leaves the value unchanged does not satisfy that condition. GDB can use hardware or software for watch, but rwatch requires hardware support. Watchpoint resources and supported sizes depend on the target.

Primary references

Each answer in the digital key links to its command documentation. The references below provide starting points for readers using a printed copy.

GNU GDB manual. Starting your Program.

<https://sourceware.org/gdb/download/onlinedocs/gdb.html/Starting.html>

GNU GDB manual. Continuing and Stepping.

<https://sourceware.org/gdb/download/onlinedocs/gdb.html/Continuing-and-Stepping.html>

GNU GDB manual. Setting Breakpoints.

<https://sourceware.org/gdb/download/onlinedocs/gdb.html/Set-Breaks.html>

GNU GDB manual. Setting Watchpoints.

<https://sourceware.org/gdb/download/onlinedocs/gdb.html/Set-Watchpoints.html>

GNU GDB manual. Debugging an Already-running Process.

<https://sourceware.org/gdb/download/onlinedocs/gdb.html/Attach.html>

After solving

At a function call, predict where source stepping into and over the call will stop, then compare the two instruction-stepping commands. Explain which watchpoint would help investigate an unexpected value change and which would help investigate a read.

References checked 11 October 2026; GDB release manual: 18.1. Clues and grid arrangements are original.

WinDbg Live Debugging Crossword

Fourteen clues with a separate answer key

Fourteen commands for live user-mode debugging with WinDbg. The crossword covers process control, software and processor breakpoints, execution to calls and returns, and exception break settings.

Live debugging practice

Recognize the command from the change it makes to the debugging session. The stepping clues assume assembly mode and the default count of one.

How to solve

1. Write one capital letter in each white square. Shaded squares are unused.
2. Answers run left to right or top to bottom. Each answer crosses at least one other answer.
3. Enter the command itself. Numbers in parentheses show its letter count.
4. Omit leading dots, options and arguments. Two-letter and three-letter command names are complete answers. The answer key restores leading dots and lowercase spelling.

Answer keys: 94. Reference notes begin on page 43.

Reference notes

These clues concern invasive live user-mode debugging. In an active WinDbg session, .attach and .create take effect when an execution command is issued. They cannot be issued while WinDbg is dormant. The .restart clue applies to a target created by the debugger.

The bp clue assumes an immediately resolved address. An unresolved bp location becomes a bu breakpoint. By default, bm creates symbolic bu breakpoints; its /d option creates address-based bp breakpoints. Processor breakpoints have hardware limits on their number, size and alignment.

The stepping clues use assembly mode. If already at a call, pc steps over it before looking for the next call. If already at a return, pt executes it before looking for the next return. In source mode, instructions on the current source line can be skipped. Breakpoints can stop execution earlier.

The exception clues specify av without -h, so they change break status. With -h, the same commands change handling status. Disabling the first-chance break does not remove the second-chance break.

Primary references

Each answer in the digital key links to its command documentation. The references below provide starting points for readers using a printed copy.

Microsoft Learn. Debugger Commands.

<https://learn.microsoft.com/en-us/windows-hardware/drivers/debuggercmds/debugger-commands>

Microsoft Learn. bp bu bm Set Breakpoint.

<https://learn.microsoft.com/en-us/windows-hardware/drivers/debuggercmds/bp--bu--bm--set-breakpoint->

Microsoft Learn. ba Break on Access.

<https://learn.microsoft.com/en-us/windows-hardware/drivers/debuggercmds/ba--break-on-access->

Microsoft Learn. Set Exceptions.

<https://learn.microsoft.com/en-us/windows-hardware/drivers/debuggercmds/sx--sxd--sxe--sxi--sxn--sxr--sx---set-exceptions->

Microsoft Learn. Controlling the Target Using WinDbg Classic.

<https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/controlling-the-target>

After solving

Choose a command that stops before a return instruction and one that waits for the current function to finish. Explain what evidence each stopping point exposes. Then compare an address breakpoint with a symbolic breakpoint when a module is reloaded.

References checked 10 October 2026. Clues and grid arrangements are original.

LLDB Live Debugging Crossword

Fourteen clues with a separate answer key

Fourteen LLDB commands for live debugging. The puzzle covers launching and attaching, continuing and interrupting, source and instruction stepping, breakpoints, watchpoints and detaching.

Live debugging practice

Recognize the command from its effect on a live target. The stepping clues assume forward execution, default counts and usable source information where needed.

How to solve

1. Write one capital letter in each white square. Shaded squares are unused.
2. Answers run left to right or top to bottom. Each answer crosses another answer.
3. Use full command and subcommand names. Omit spaces, hyphens, options and arguments; do not use aliases.
4. Word-part lengths appear in parentheses. For example, (6,4,2) uses twelve squares. The key restores spaces, hyphens and lowercase spelling.

Answer keys: 95. Reference notes begin on page 47.

Reference notes

These commands require a live target with a supporting process plugin. Launching and attaching depend on platform permissions and the selected target. They do not turn a loaded core file into a running process.

Source stepping uses line information and stepping settings, including rules that avoid entering some functions. Instruction stepping operates on machine instructions. Breakpoints, watchpoints and other stop events can interrupt a step before its intended endpoint.

Stepping out runs the selected frame until it returns; it does not force an immediate return. Detaching ends debugger control. Whether the process remains stopped depends on detach options and the process plugin.

A variable watchpoint uses the variable's memory location. Hardware support, available watchpoint resources and address size or alignment can limit it. The default watch type is modify.

Answers are the complete command paths. For thread step-inst-over, enter `THREADSTEPINSTOVER` and use (6,4,4,4); restore the spaces and hyphens before using the command.

Primary references

The digital answer key links each command to its official documentation or implementation help.

LLVM LLDB. GDB to LLDB Command Map.

<https://lldb.llvm.org/use/map.html>

LLVM LLDB. Tutorial.

<https://lldb.llvm.org/use/tutorial.html>

LLVM LLDB. Process command definitions.

https://lldb.llvm.org/cpp_reference/CommandObjectProcess_8cpp_source.html

LLVM LLDB. Thread command definitions.

https://lldb.llvm.org/cpp_reference/CommandObjectThread_8cpp_source.html

LLVM LLDB. Watchpoint command definitions.

https://lldb.llvm.org/cpp_reference/CommandObjectWatchpoint_8cpp_source.html

After solving

Compare source-level and instruction-level stepping at a call. Then explain the different reasons to interrupt, detach, set a breakpoint or watch a variable.

References checked 10 October 2026. Clues and grid arrangements are original.

Live Debugging Concepts

Crosswords

Two puzzles with separate answer keys

Two crosswords for live debugging practice. The first covers established terminology; the second focuses on the named Debugging Implementation Patterns used in Dmitry Vostokov's Debugging 4D courses.

Puzzle 14 Established terminology

Twelve clues on execution control, program state, concurrency and replay. Use the standard term described by each clue.

Puzzle 15 Unified debugging patterns

Eight clues on selected implementation patterns from the Windows and Linux Debugging 4D courses. Use the published pattern name described by each scenario.

How to solve

1. Write one capital letter in each white square. Shaded squares are unused.
2. Answers run left to right or top to bottom. Each answer crosses at least one other answer.
3. Omit spaces and hyphens. Numbers in parentheses give word-part lengths: (5,2) uses seven squares.
4. Each crossword has its own numbering. The answer keys restore spaces, hyphens and readable spelling.

Answer keys: 96–97. Reference notes begin on page 53.

Course and reference notes

Puzzle 14 practices established debugging terminology. Puzzle 15 uses the names of eight Debugging Implementation Patterns from the Windows and Linux Debugging 4D courses. These are selected implementation patterns within the unified approach, which connects diagnostic analysis with debugging architecture, design, implementation, and usage.

Unified Debugging Patterns introductory article.

<https://www.dumpanalysis.org/blog/index.php/2010/12/03/analysis-architectural-design-implementation-and-usage-debugging-patterns-part-0/>

Course map for Puzzle 15

Page numbers count from the first page of each public slide PDF.

Pattern	Windows 4D	Linux 4D
Break-in	UD1 / p. 45	UD1 / p. 63
Scope; Variable Value; Type Structure	UD3 / p. 47	UD3 / p. 67
Code Breakpoint	UD1 / p. 45	Code Breakpoints / p. 64
Data Breakpoint	UD2 / p. 46	UD2 / p. 65
Breakpoint Action	UD4 / p. 48	UD4 / p. 68
Frozen Thread	UD7 / p. 53	UD7 / p. 73

Execution and observation

A data trigger may detect a value change, read or write; its precise behavior depends on the debugger and mechanism. Freezing or stepping threads changes scheduling and can alter the failure being investigated. Symbols, optimization and the selected frame affect what state can be recovered. Replay uses recorded history; it is distinct from controlling current live execution.

Primary references

The digital answer keys link each answer to course slides or primary technical documentation. The course slides establish the pattern names. The clues are original teaching prompts.

Windows Debugging 4D slides 2024.

<https://www.patterndiagnostics.com/Training/Accelerated-Windows-Debugging-4D-Slides.pdf>

Linux Debugging 4D slides 2024.

<https://www.patterndiagnostics.com/Training/Accelerated-Linux-Debugging-4D-Slides.pdf>

GNU GDB manual.

<https://sourceware.org/gdb/download/onlinedocs/gdb.html/>

Microsoft Learn on race conditions and deadlocks.

<https://learn.microsoft.com/en-us/troubleshoot/developer/visualstudio/visual-basic/language-compilers/race-conditions-deadlocks>

After solving

Choose a corruption, hang or timing failure. Select two patterns from Puzzle 15 and explain what evidence each would expose. Use terms from Puzzle 14 to describe the state you would inspect and how controlling execution might change the observed behavior.

References checked 11 October 2026; GDB release manual: 18.1. Clues and grid arrangements are original.

x86 Crash Dump Crosswords

Two practice puzzles with separate answer keys

Two crosswords practice 32-bit IA-32 instructions and register names useful when reading crash-dump disassembly and saved processor state. The first puzzle has 20 instruction clues; the second has 16 register clues. They cover memory access, calls and returns, condition tests and register roles.

Puzzle 16 CPU instructions

Recognize the operation described by each clue.

Puzzle 17 CPU registers

Recognize register names and roles from widths, aliases or stated conventions.

How to solve

1. Write one capital letter or digit in each white square. Shaded squares are unused.
2. Answers run left to right or top to bottom. Each answer crosses another answer.
3. Enter the instruction mnemonic, register name or requested register role. Omit operands, spaces and punctuation; keep digits.
4. Numbers in parentheses count characters, including digits. Use zero, not the letter O, in register numbers.

Answer keys: 98–99. Reference notes begin on page 60.

Reference notes

This booklet uses x86 to mean 32-bit IA-32. Clues assume 32-bit address and operand sizes where stated, Intel mnemonics and ordinary near calls and returns. JNE and JNZ name the same condition; the clue requests JNE.

A frame pointer is a convention, not a guarantee that EBP forms a valid frame chain. Segment registers hold selectors in protected mode. The segment clues describe architectural roles, even when the operating system uses a flat memory model.

Write ST0 for the x87 name ST(0), omitting its parentheses. Retain the zero in ST0 and XMM0. Available x87 and SSE state depends on the processor, operating system and captured context. Read a 32-bit process in its 32-bit context even on a 64-bit host.

Primary references

The digital answer keys link to supporting primary references. Use the instruction and register sections for fuller semantics and exceptions.

Intel. Software Developer Manuals volumes 1 and 2.

<https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html>

Microsoft Learn. x86 Architecture.

<https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/x86-architecture>

Microsoft Learn. x86 Instructions.

<https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/x86-instructions>

After solving

Explain how a traditional EBP-based frame is dismantled and how the return address is then consumed. Distinguish an instruction address, an effective data address and the current stack pointer in a 32-bit crash context.

References checked 10 October 2026. Clues and grid arrangements are original.

x64 Crash Dump Crosswords

Two practice puzzles with separate answer keys

Two crosswords practice x64 instructions and registers used in crash-dump disassembly and saved processor state. The first has 20 instruction clues covering sign extension, atomic-update primitives, string operations, SIMD and system calls. The second has 16 register clues.

Puzzle 18 CPU instructions

Recognize the operation described by each clue.

Puzzle 19 CPU registers

Recognize register names and roles from widths, aliases or stated conventions.

How to solve

1. Write one capital letter or digit in each white square. Shaded squares are unused.
2. Answers run left to right or top to bottom. Each answer crosses another answer.
3. Enter the instruction mnemonic, register name or requested register role. Omit operands, spaces and punctuation; keep digits.
4. Numbers in parentheses count characters, including digits. Use zero, not the letter O, in register numbers.

Answer keys: 100–101. Reference notes begin on page 66.

Reference notes

Puzzle 18 complements the x86 instruction puzzle with sign extension, system calls, atomic-update primitives, string operations and SIMD moves. Several instructions are shared with 32-bit mode. String clues assume 64-bit addresses; SIMD clues use legacy 128-bit forms.

Only the two argument-register clues assume the standard Windows x64 calling convention. Linux and macOS use different conventions. RBP need not hold a frame pointer, and register values at a crash need not still contain entry arguments.

Use the exception context and surrounding code. CMPXCHG16B needs processor support; ENDBR64 relates to CET indirect-branch tracking. Atomic memory updates need a LOCK prefix where specified. Check SIMD alignment and captured state. UD2 may be an intentional trap.

Primary references

Instruction answers link to the Intel SDM, revision 093, with page-specific references. Register answers use the manuals and ABI documentation below.

Intel. Software Developer Manuals volumes 1 and 2.

<https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html>

Microsoft Learn. x64 Architecture and Registers.

<https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/x64-architecture>

Microsoft Learn. x64 Calling Convention.

<https://learn.microsoft.com/en-us/cpp/build/x64-calling-convention?view=msvc-170>

After solving

Compare aligned and unaligned SIMD moves. Explain a sign-extension instruction and an atomic-update primitive, then identify the register values needed to interpret a string operation.

References checked 11 October 2026. Clues and grid arrangements are original.

ARM64 Crash Dump Crosswords

Two practice puzzles with separate answer keys

Two crosswords practice ARM64 (AArch64) instructions and register names useful when reading crash-dump disassembly and saved processor state. The first puzzle has 20 instruction clues; the second has 16 register clues. They cover memory access, calls and returns, condition tests and register roles.

Puzzle 20 CPU instructions

Recognize the operation described by each clue.

Puzzle 21 CPU registers

Recognize register names and roles from widths, aliases or stated conventions.

How to solve

1. Write one capital letter or digit in each white square. Shaded squares are unused.
2. Answers run left to right or top to bottom. Each answer crosses another answer.
3. Enter the instruction mnemonic, register name or requested register role. Omit operands, spaces and punctuation; keep digits.
4. Numbers in parentheses count characters, including digits. Use zero, not the letter O, in register numbers.

Answer keys: 102–103. Reference notes begin on page 72.

Reference notes

ARM64 here means AArch64 and its A64 instruction set. Clues use ordinary scalar forms and baseline AAPCS64 register roles. Platform ABIs can add rules; values at an arbitrary crash location need not still be function arguments.

MOV and CMP are assembly aliases. A64 RET normally branches through X30; it does not pop a return address. LDP and STP access pairs and are not inherently stack operations. ADRP uses a 4 KiB address boundary regardless of the operating system page size.

Use the captured exception context and surrounding instructions. PC is separate from the X registers. A register field encoded as 31 can denote SP or a zero register, depending on the instruction. Available floating-point and SIMD state depends on the capture.

Primary references

The digital answer keys link to supporting primary references. Use the instruction and register sections for fuller semantics and exceptions.

Arm. Armv8 A Instruction Set Architecture.

<https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/Learn%20the%20Architecture/Armv8-A%20Instruction%20Set%20Architecture.pdf>

Arm. A64 Instruction Set Overview.

https://developer.arm.com/-/media/Files/pdf/graphics-and-multimedia/ARMv8_InstructionSetOverview.pdf

Arm. Procedure Call Standard for the Arm 64 bit Architecture.

<https://github.com/ARM-software/abi-aa/blob/main/aapcs64/aapcs64.rst>

Arm. Armv8 A Self hosted Debug.

<https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/Learn%20the%20Architecture/Arm%20V8A%20self-hosted%20debug.pdf>

After solving

Explain what a nearby load, paired stack access and branch imply about a crash. Identify the registers needed to calculate a memory address, and separate captured values from values reconstructed for an older frame.

References checked 10 October 2026. Clues and grid arrangements are original.

Observability Crosswords

Three practice puzzles with separate answer keys

Three crosswords explore the evidence emitted by running systems, the measurements used to assess service reliability, and the tracing and pipeline mechanisms that connect telemetry. Each puzzle has 14 clues.

Puzzle 22 Observability foundations

Signals, instrumentation, context, metadata and profiling.

Puzzle 23 Metrics and reliability

Prometheus metric types, service objectives and error budgets.

Puzzle 24 Tracing and telemetry pipelines

OpenTelemetry span relationships, sampling, collection and pipeline processing.

How to solve

1. Write one capital letter in each white square. Shaded squares are unused.
2. Answers run left to right or top to bottom. Each answer crosses another answer.
3. Omit spaces, hyphens and slashes. Word-part lengths appear in parentheses; (4,4) uses eight squares.
4. Use established terminology. Where a clue requests an abbreviation, enter the abbreviation.

Answer keys: 104–106. Reference notes begin on page 80.

Foundations and metrics reference notes

These puzzles use software observability terminology. Emitting logs, metrics and traces helps investigation, but their presence alone does not guarantee enough evidence to explain a failure. Profiling can relate resource consumption to code.

Metric types use Prometheus terminology. Counters can reset. Histogram quantiles are calculated from bucket data; summaries can estimate quantiles in the application. Averaging precomputed quantiles from several instances does not produce a valid combined quantile.

Cardinality concerns distinct series, not the number of samples in one series. An SLI measures service behavior; an SLO sets a target. For a success-ratio objective, the allowed unsuccessful fraction is one minus the target.

Primary references

Each answer in the digital keys links to its supporting documentation.

OpenTelemetry. Observability primer and related concepts.

<https://opentelemetry.io/docs/concepts/observability-primer/>

OpenTelemetry. Profiles.

<https://opentelemetry.io/docs/concepts/signals/profiles/>

Prometheus. Metric types.

https://prometheus.io/docs/concepts/metric_types/

Prometheus. Data model.

https://prometheus.io/docs/concepts/data_model/

Prometheus. Histograms and summaries.

<https://prometheus.io/docs/practices/histograms/>

Google SRE. Service Level Objectives.

<https://sre.google/sre-book/service-level-objectives/>

After solving

For a slow request, choose evidence from two signals. Then define a useful SLI and explain why a counter and a gauge answer different questions.

References checked 10 October 2026. Clues and grid arrangements are original.

Tracing and pipeline reference notes

Shared context makes signals easier to correlate. A parent-child relation and a span link express different relationships; links can connect spans from different traces. An exemplar can connect a metric observation to trace context.

Instrumentation scope identifies the software emitting telemetry. Resource detection discovers host or environment attributes. Sampling can omit evidence; this implies that a tail sampler cannot recover spans already discarded by a head sampler.

Receivers obtain data, processors transform it, and exporters deliver it. Connectors join pipelines; the batch processor groups records for efficient export. OTLP/HTTP uses HTTP POST. A sustained burn rate of one spends the error budget over the SLO window.

Primary references

Answer keys also link to batch processing, resource detection, instrumentation scope and exemplar definitions.

OpenTelemetry. Traces.

<https://opentelemetry.io/docs/concepts/signals/traces/>

OpenTelemetry. Instrumentation scope.

<https://opentelemetry.io/docs/concepts/instrumentation-scope/>

OpenTelemetry. Sampling.

<https://opentelemetry.io/docs/concepts/sampling/>

OpenTelemetry. Collector configuration.

<https://opentelemetry.io/docs/collector/configuration/>

OpenTelemetry. OTLP specification.

<https://opentelemetry.io/docs/specs/otlp/>

Google SRE. Alerting on SLOs.

<https://sre.google/workbook/alerting-on-slos/>

After solving

Follow a request through a receiver, processor and connector. Explain how resource detection, batching and sampling change the evidence available for diagnosis.

References checked 11 October 2026. Clues and grid arrangements are original.

Answer keys

Completed grids and answer lists for all 24 crosswords appear in puzzle order. The numbers below refer to answer-key pages.

Memory Dump Analysis Crosswords	83–86
Trace and Log Analysis Crosswords.....	87–89
WinDbg Command Crossword.....	90
GDB Command Crossword.....	91
LLDB Memory Dump Analysis Crossword	92
GDB Live Debugging Crossword	93
WinDbg Live Debugging Crossword	94
LLDB Live Debugging Crossword	95
Live Debugging Concepts Crosswords	96–97
x86 Crash Dump Crosswords	98–99
x64 Crash Dump Crosswords	100–101
ARM64 Crash Dump Crosswords	102–103
Observability Crosswords	104–106

Windows memory dump vocabulary

Puzzle on page 6. Answers link to primary references where available.



Across

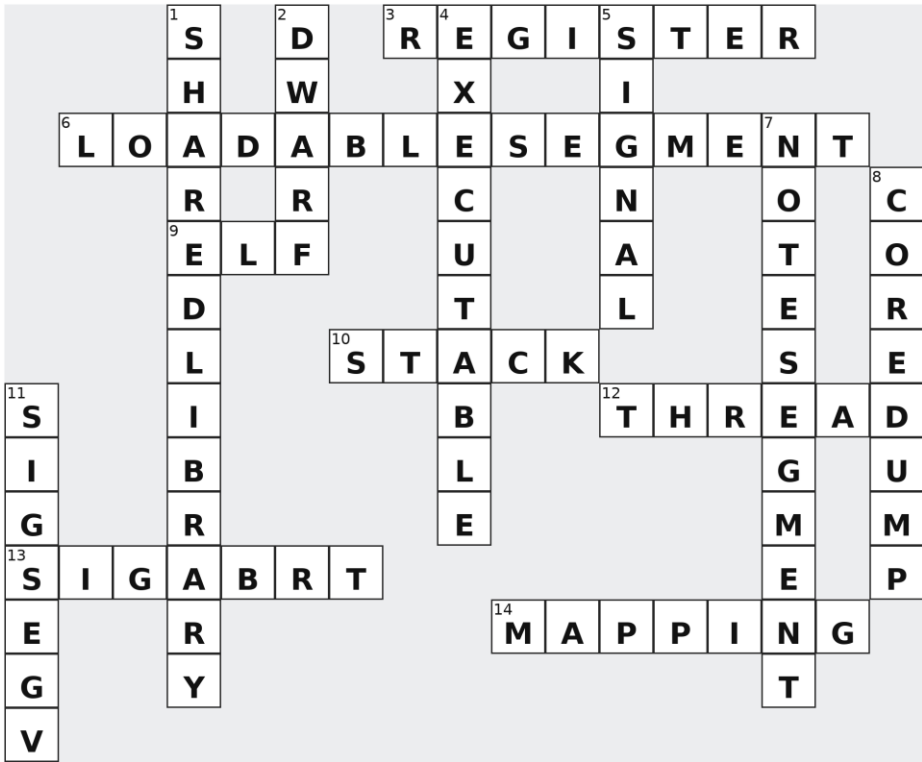
- 3. THREAD
- 5. KERNEL
- 6. REGISTER
- 8. HANDLE
- 11. PROCESS
- 13. MINIDUMP
- 14. POINTER

Down

- 1. SYMBOLS
- 2. ADDRESS
- 4. CONTEXT
- 5. KERNEL
- 7. EXCEPTION
- 9. MODULE
- 10. HEAP
- 12. STACK

Linux memory dump vocabulary

Puzzle on page 8. Answers link to primary references where available.



Across

3. [Register](#)
6. [Loadable segment](#)
9. [ELF](#)
10. [Stack](#)
12. [Thread](#)
13. [SIGABRT](#)
14. [Mapping](#)

Down

1. [Shared library](#)
2. [DWARF](#)
4. [Executable](#)
5. [Signal](#)
7. [Note segment](#)
8. [Core dump](#)
11. [SIGSEGV](#)

[illegible]

2. [Mach-O](#)
4. [Segment](#)
5. [dylib](#)
8. [Register](#)
10. [Load co](#)
13. [Thread](#)

1. [Port](#)
3. [UUID](#)
4. [Slide](#)
6. [dSYM](#)
7. [ASLR](#)
9. [Task](#)
11. [Mach](#)
12. [dyld](#)

Memory dump analysis patterns

Puzzle on page 12. Answers link to primary references where available.

								¹ M									
				² B		³ D	E	A	D	L	O	C	K				
				L			M									⁴ M	
				O			O				⁵ N		⁶ S			I	
				C			R		⁷ S		U		P			S	
				K			Y		T		L		I			S	
			⁸ W	E			L		A		L		K			I	
		⁹ H	I	D	D	E	N	E	X	C	E	P	T	I	O	N	
¹⁰ S		L		T			A		K		O		N			G	
T		D		H			K		T		I		G			C	
A		P		R						R		N		T		O	
C		O		E						A		T		H		M	
K		I		A						C		E		R		P	
O		N		¹¹ D	O	U	B	L	E	F	R	E				O	
V		T												A		N	
¹² E	X	E	C	U	T	I	O	N	R	E	S	I	D	U	E		
R		R														N	
F																	T
¹³ L	A	T	E	R	A	L	D	A	M	A	G	E					
O																	
¹⁴ W	A	I	T	C	H	A	I	N									

Across

- 3. [Deadlock](#)
- 9. [Hidden Exception](#)
- 11. [Double Free](#)
- 12. [Execution Residue](#)
- 13. [Lateral Damage](#)
- 14. [Wait Chain](#)

Down

- 1. [Memory Leak](#)
- 2. [Blocked Thread](#)
- 4. [Missing Component](#)
- 5. [NULL Pointer](#)
- 6. [Spiking Thread](#)
- 7. [Stack Trace](#)
- 8. [Wild Pointer](#)
- 10. [Stack Overflow](#)

Trace and log fundamentals

Puzzle on page 17. Answers link to primary references where available.



Across

2. [Buffer](#)
5. [Delimiter](#)
7. [Severity](#)
9. [Rotation](#)
10. [Schema](#)
11. [Record](#)
14. [Message](#)

Down

1. [Regex](#)
3. [Filter](#)
4. [Timestamp](#)
6. [Retention](#)
8. [Parser](#)
12. [Event](#)
13. [JSON](#)

Distributed tracing

Puzzle on page 19. Answers link to primary references where available.



Across

- 4. [Propagation](#)
- 6. [Context](#)
- 8. [Sampling](#)
- 10. [Parent](#)
- 11. [Trace ID](#)
- 12. [Resource](#)
- 14. [Collector](#)

Down

- 1. [Link](#)
- 2. [Span](#)
- 3. [Baggage](#)
- 5. [Attributes](#)
- 7. [Exporter](#)
- 9. [Span ID](#)
- 13. [Root](#)

Trace analysis patterns

Puzzle on page 21. Answers link to primary references where available.

D	I	S	C	O	N	T	I	N	U	I	T	Y				F
										N						A
		4	C							T	5	T	6	M		L
7	S	I				8	M			E	H		E			S
9	P	E	R	I	O	D	I	C	E	R	R	O	R		S	E
	A	C				S				C	E		S			P
	R	U				S				O	A		A			O
	S	L				I				R	D		G			S
	E	A		10	A	N				R	O		E			I
	T	R		C	G					E	F		I			T
	R	T		11	T	I	M	E	D	E	L	T	A		N	I
	A	R		I		E				A	C		V			V
	C	A		V		S		12	M	E	T	A	T	R	A	C
	E	C		I		S				I	I		R			E
		E		T		A				O	V		I			R
				Y		G				N		I		A		R
				R		E						T		N		O
				E								Y		T		R
				G												
			13	S	I	L	E	N	T	M	E	S	S	A	G	E
				O												
			14	A	N	C	H	O	R	M	E	S	S	A	G	E

Across

- 1. [Discontinuity](#)
- 9. [Periodic Error](#)
- 11. [Time Delta](#)
- 12. [Meta Trace](#)
- 13. [Silent Messages](#)
- 14. [Anchor Messages](#)

Down

- 2. [Inter-Correlation](#)
- 3. [False Positive Error](#)
- 4. [Circular Trace](#)
- 5. [Thread of Activity](#)
- 6. [Message Invariant](#)
- 7. [Sparse Trace](#)
- 8. [Missing Message](#)
- 10. [Activity Region](#)

WinDbg commands

Puzzle on page 25. Answers link to primary references where available.



Across

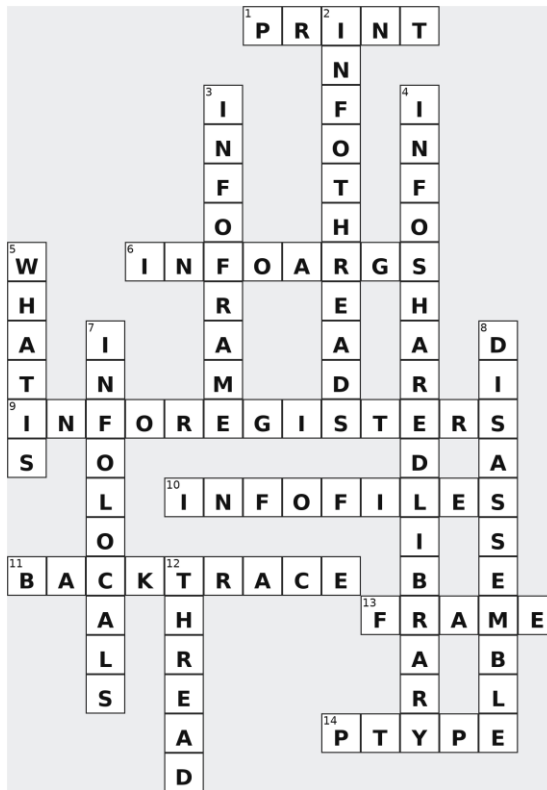
- 2. [.reload](#)
- 5. [.cxr](#)
- 8. [!address](#)
- 10. [!heap](#)
- 12. [.exr](#)
- 13. [!handle](#)
- 14. [!teb](#)

Down

- 1. [.ecxr](#)
- 3. [!locks](#)
- 4. [!analyze](#)
- 6. [!runaway](#)
- 7. [!vertarget](#)
- 9. [.sympath](#)
- 11. [!peb](#)

GDB core analysis commands

Puzzle on page 29. Answers link to primary references where available.



Across

1. [print](#)
6. [info args](#)
9. [info registers](#)
10. [info files](#)
11. [backtrace](#)
13. [frame](#)
14. [pty](#)

Down

2. [info threads](#)
3. [info frame](#)
4. [info sharedlibrary](#)
5. [whatis](#)
7. [info locals](#)
8. [disassemble](#)
12. [thread](#)

LLDB memory dump analysis commands

Puzzle on page 33. Answers link to primary references where available.

			¹ F	R	A	M	E	S	E	L	E	C	² T			
			R										A			
³ T		A		⁴ R	E	G	I	S	⁵ T	E	R	R	E	A	D	
A		M							A			G				
R		E			⁶ M	E	M	O	R	Y	R	E	A	D		
G		V							G			T			⁷ T	
E		A							E			M			A	
⁸ T	H	R	E	A	D	L	I	S	T			O			R	
V		I							M			D			G	
A		A							O			U			E	
R		B		⁹ T	H	R	E	A	D	S	E	L	E	C	T	
I		L							U			E			C	
A		E							L			S			R	
B			¹⁰ D	I	S	A	S	S	E	M	B	L	E		E	
L									S			I			A	
E									L			S			T	
	¹¹ F	R	A	M	E	I	N	F	O			T			E	
									O							
¹² T	H	R	E	A	D	B	A	C	K	T	R	A	C	E		
									U							
¹³ T	Y	P	E	L	O	O	K	U	P							

Across

- [1. frame select](#)
- [4. register read](#)
- [6. memory read](#)
- [8. thread list](#)
- [9. thread select](#)
- [10. disassemble](#)
- [11. frame info](#)
- [12. thread backtrace](#)
- [13. type lookup](#)

Down

- [1. frame variable](#)
- [2. target modules list](#)
- [3. target variable](#)
- [5. target modules lookup](#)
- [7. target create](#)

[illegible]

3. step
4. attach
8. watch
9. run
11. break
12. next
13. stepi

1. finish
2. detach
3. start
5. tbreak
6. continue
7. rwatch
10. nexti

[illegible]

1. sxd
2. bp
5. .create
9. .bpcmds
10. gu
11. ba
12. pt

1. sxe
3. pc
4. .detach
6. .restart
7. .attach
8. bm
9. bu

LLDB live debugging commands

Puzzle on page 45. Answers link to primary references where available.

				1	T	H	R	E	A	D	S	T	E	P	O	U	T
					H												
2	W			3	P	R	O	C	E	S	S	L	A	U	N	C	H
	A		4	B	E												
5	T	H	R	E	A	D	S	T	E	6	P	I	N	S	T		
	C		E		D					R					7	T	
	H		A		S					O						H	
	P		K		T		8	P		C						R	
	O		P		E		R			E		9	W			E	
	I		O		10	P	R	O	C	E	S	S	A	T	T	A	C
	N		I		I		C			S		T			D		
	T		N		N		E			D		C			S	11	B
	S		T		S		S			E		H			T		R
	E		D		T		S			T		P			E		E
	T		E		O		C			A		O			P		A
	V		L		V		O			C		I			O		K
	A		E		E		N			H		N			V		P
	R		T		R		T					T			E		O
	I		E				I					D			R		I
	A						N					E					N
	B						U					L					T
	L				12	T	H	R	E	A	D	S	T	E	P	I	N
	E													T			E
	13	P	R	O	C	E	S	S	I	N	T	E	R	R	U	P	T

Across

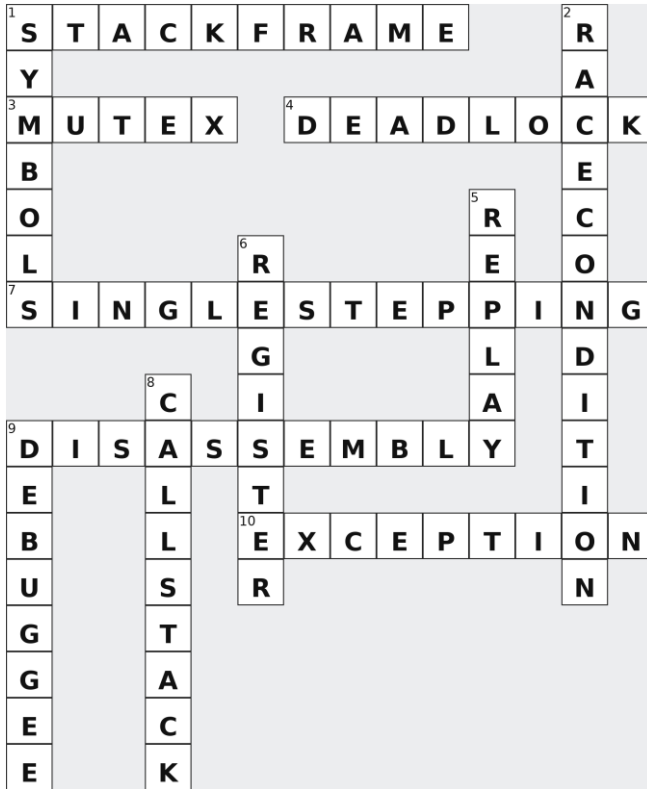
- 1. [thread step-out](#)
- 3. [process launch](#)
- 5. [thread step-inst](#)
- 10. [process attach](#)
- 12. [thread step-in](#)
- 13. [process interrupt](#)

Down

- 1. [thread step-inst-over](#)
- 2. [watchpoint set variable](#)
- 4. [breakpoint delete](#)
- 6. [process detach](#)
- 7. [thread step-over](#)
- 8. [process continue](#)
- 9. [watchpoint delete](#)
- 11. [breakpoint set](#)

Established terminology

Puzzle on page 49. Answers link to primary references where available.



Across

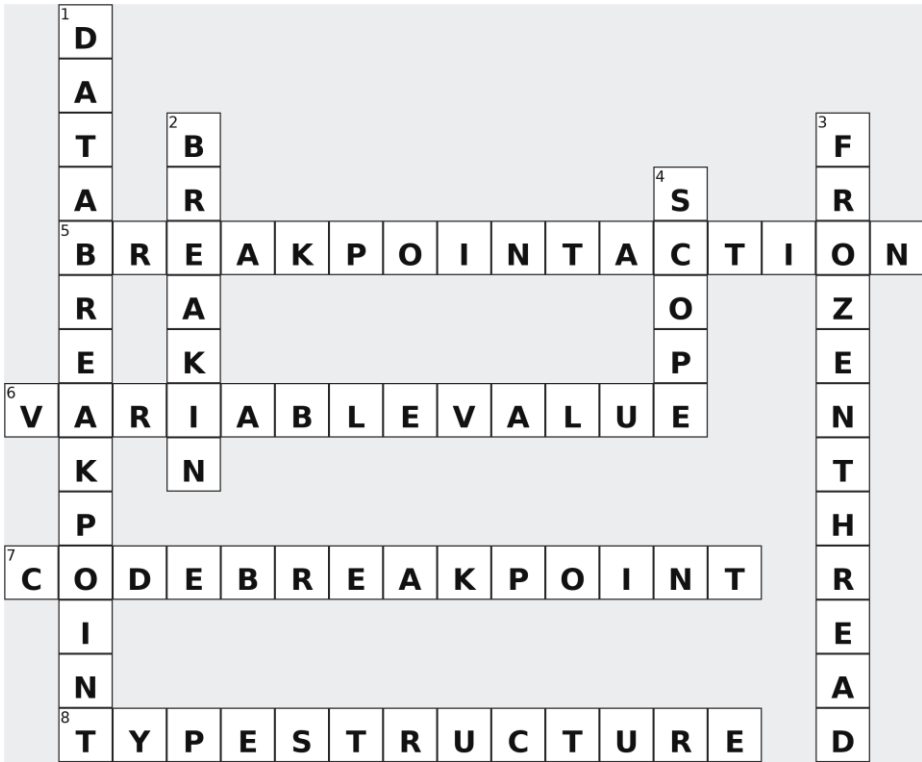
1. [Stack frame](#)
3. [Mutex](#)
4. [Deadlock](#)
7. [Single stepping](#)
9. [Disassembly](#)
10. [Exception](#)

Down

1. [Symbols](#)
2. [Race condition](#)
5. [Replay](#)
6. [Register](#)
8. [Call stack](#)
9. [Debugger](#)

Unified debugging patterns

Puzzle on page 51. Answers link to primary references where available.



Across

- 5. [Breakpoint Action](#)
- 6. [Variable Value](#)
- 7. [Code Breakpoint](#)
- 8. [Type Structure](#)

Down

- 1. [Data Breakpoint](#)
- 2. [Break-in](#)
- 3. [Frozen Thread](#)
- 4. [Scope](#)

[illegible]

4. CMOVNE
5. ADD
6. MOVZX
9. MOV
10. RET
12. JMP
13. SUB
14. LEA
15. INT3
17. PUSH

1. POP
2. LEAVE
3. CALL
4. CDQ
6. MOVSB
7. XOR
8. CMP
11. TEST
12. JNE
16. NOP

[illegible]

2. ESI
3. EBP
4. EDI
5. ECX
6. EAX
8. ES
9. SS
10. XMM0

1. CS
2. ESP
3. EIP
4. EBX
5. EFLAGS
7. DS
8. EDX
9. ST0

x64 instructions

Puzzle on page 62. Answers link to primary references where available.

						1	C									
					2	M		3	M	O	V	D	Q	4	U	
						O			P						D	
						V		5	X	A	D	D			2	
						S			C							
6	C	7	M	P	X	C	H	G	1	6	8	B				
			O		D		G					9	S	A	R	
			V						10		R		W			
11	C	D	Q		12	E			13	M	O	V	A	P	14	S
			Q			N					L		P			Y
			A			D										S
						B						15	S			C
16	P	17	X	O	R								T			A
			C			6				18	C	Q	O			L
			H			4							19	S	H	L
			G					20	M	O	V	S	Q			

Across

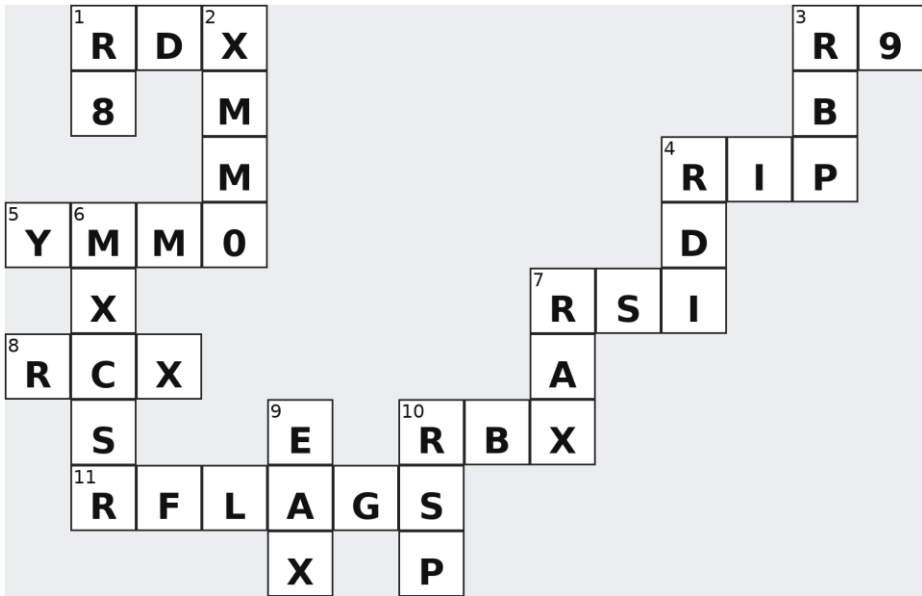
- 3. [MOVDQU](#)
- 5. [XADD](#)
- 6. [CMPXCHG16B](#)
- 9. [SAR](#)
- 11. [CDQE](#)
- 13. [MOVAPS](#)
- 16. [PXOR](#)
- 18. [CQO](#)
- 19. [SHL](#)
- 20. [MOVSQ](#)

Down

- 1. [CMPXCHG](#)
- 2. [MOVSD](#)
- 4. [UD2](#)
- 7. [MOVDQA](#)
- 8. [BSWAP](#)
- 10. [ROL](#)
- 12. [ENDBR64](#)
- 14. [SYSCALL](#)
- 15. [STOSQ](#)
- 17. [XCHG](#)

x64 registers

Puzzle on page 64. Answers link to primary references where available.



Across

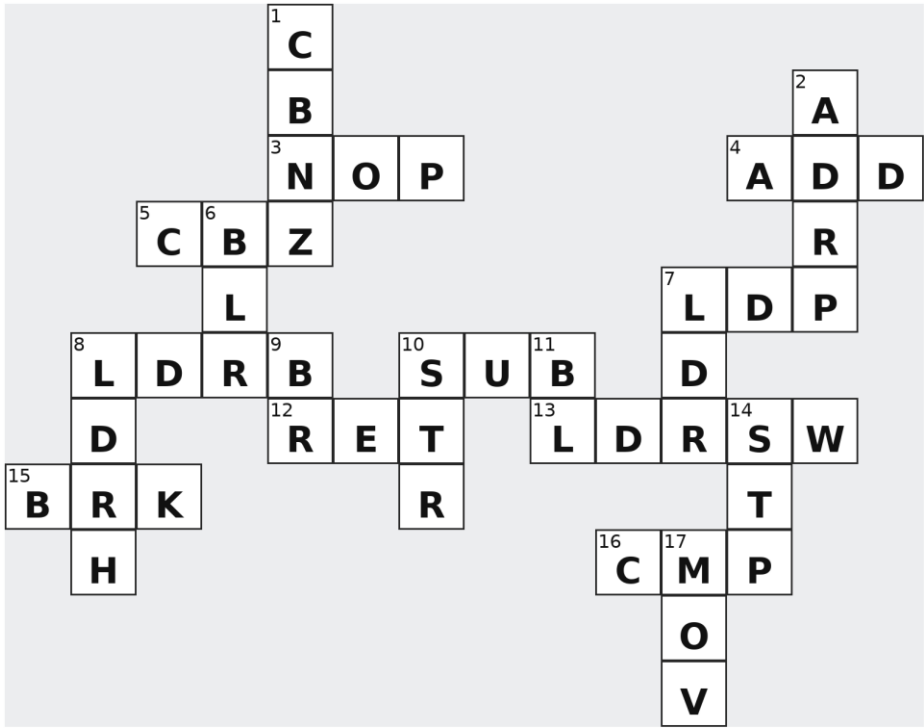
1. [RDX](#)
3. [R9](#)
4. [RIP](#)
5. [YMMO](#)
7. [RSI](#)
8. [RCX](#)
10. [RBX](#)
11. [RFLAGS](#)

Down

1. [R8](#)
2. [XMM0](#)
3. [RBP](#)
4. [RDI](#)
6. [MXCSR](#)
7. [RAX](#)
9. [EAX](#)
10. [RSP](#)

ARM64 instructions

Puzzle on page 68. Answers link to primary references where available.



Across

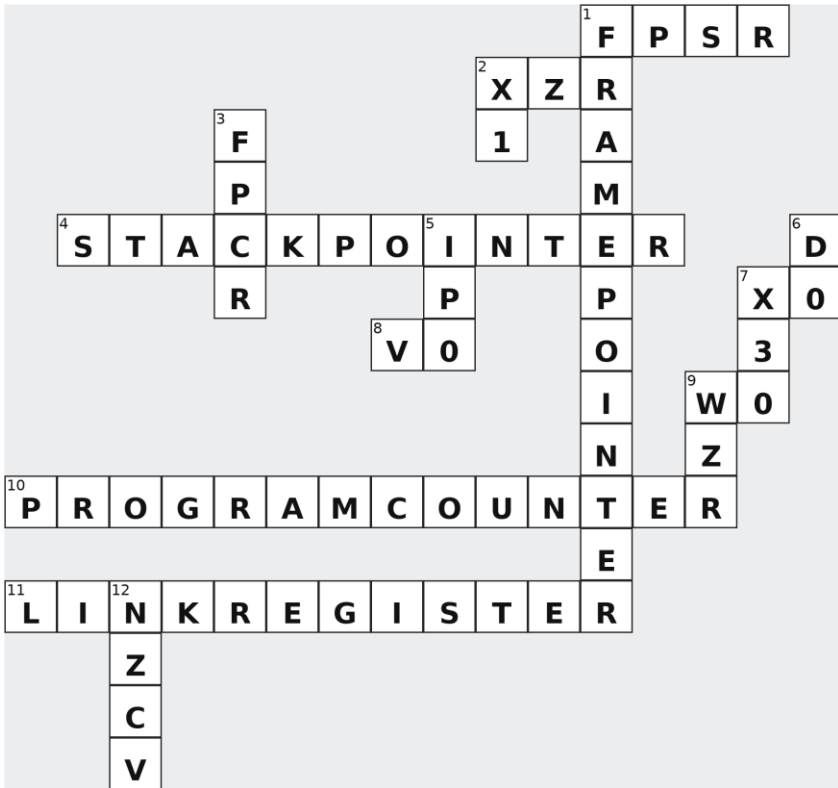
- 3. [NOP](#)
- 4. [ADD](#)
- 5. [CBZ](#)
- 7. [LDP](#)
- 8. [LDRB](#)
- 10. [SUB](#)
- 12. [RET](#)
- 13. [LDRSW](#)
- 15. [BRK](#)
- 16. [CMP](#)

Down

- 1. [CBNZ](#)
- 2. [ADRP](#)
- 6. [BLR](#)
- 7. [LDR](#)
- 8. [LDRH](#)
- 9. [BR](#)
- 10. [STR](#)
- 11. [BL](#)
- 14. [STP](#)
- 17. [MOV](#)

ARM64 registers and roles

Puzzle on page 70. Answers link to primary references where available.



Across

1. [FPSR](#)
2. [XZR](#)
4. [Stack pointer](#)
7. [X0](#)
8. [V0](#)
9. [W0](#)
10. [Program counter](#)
11. [Link register](#)

Down

1. [Frame pointer](#)
2. [X1](#)
3. [FPCR](#)
5. [IPO](#)
6. [D0](#)
7. [X30](#)
9. [WZR](#)
12. [NZCV](#)

Observability foundations

Puzzle on page 74. Answers link to primary references where available.



Across

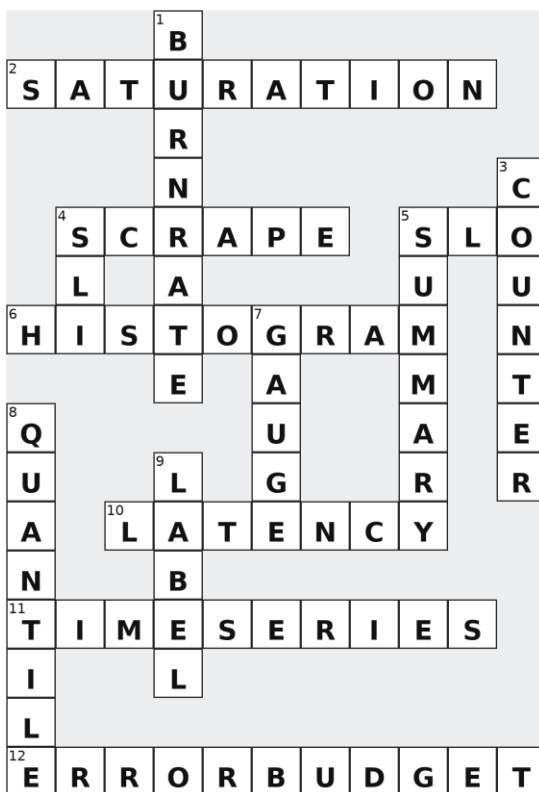
2. [Metric](#)
4. [Profiling](#)
7. [Span](#)
8. [Trace](#)
10. [Resource](#)
11. [Correlation](#)
12. [Attribute](#)
13. [Telemetry](#)
14. [Log](#)

Down

1. [Semantic conventions](#)
3. [Context propagation](#)
5. [Instrumentation](#)
6. [Cardinality](#)
9. [Observability](#)

Metrics and reliability

Puzzle on page 76. Answers link to primary references where available.



Across

2. [Saturation](#)
4. [Scrape](#)
5. [SLO](#)
6. [Histogram](#)
10. [Latency](#)
11. [Time series](#)
12. [Error budget](#)

Down

1. [Burn rate](#)
3. [Counter](#)
4. [SLI](#)
5. [Summary](#)
7. [Gauge](#)
8. [Quantile](#)
9. [Label](#)

Tracing and telemetry pipelines

Puzzle on page 78. Answers link to primary references where available.

	¹ T	A	² I	L	S	A	M	P	L	I	N	G	
			N										
³ R			S				⁴ B		⁵ H				⁶ O
E			T		⁷ S	P	A	N	E	V	E	N	T
S			R				T		A				L
O			U				C		D				P
U			M				H		S				H
R			⁸ E	X	E	M	P	L	A	R			T
C			N				R		M	⁹ P			T
E			T				O		P	A			P
D			A				C		L		R		
E		¹⁰ O	T	L	¹¹ P		E		I		E		
T			I		R		S		N		N		
E			O		O		S		G		T		
¹² C	O	N	N	E	C	T	O	R			S		
T			S		E		R				P		
I			C		S						A		
O			O		¹³ S	P	A	N	L	I	N	K	
N			P		O								
			E		¹⁴ R	E	C	E	I	V	E	R	

Across

1. [Tail sampling](#)
7. [Span event](#)
8. [Exemplar](#)
10. [OTLP](#)
12. [Connector](#)
13. [Span link](#)
14. [Receiver](#)

Down

2. [Instrumentation scope](#)
3. [Resource detection](#)
4. [Batch processor](#)
5. [Head sampling](#)
6. [OTLP/HTTP](#)
9. [Parent span](#)
11. [Processor](#)