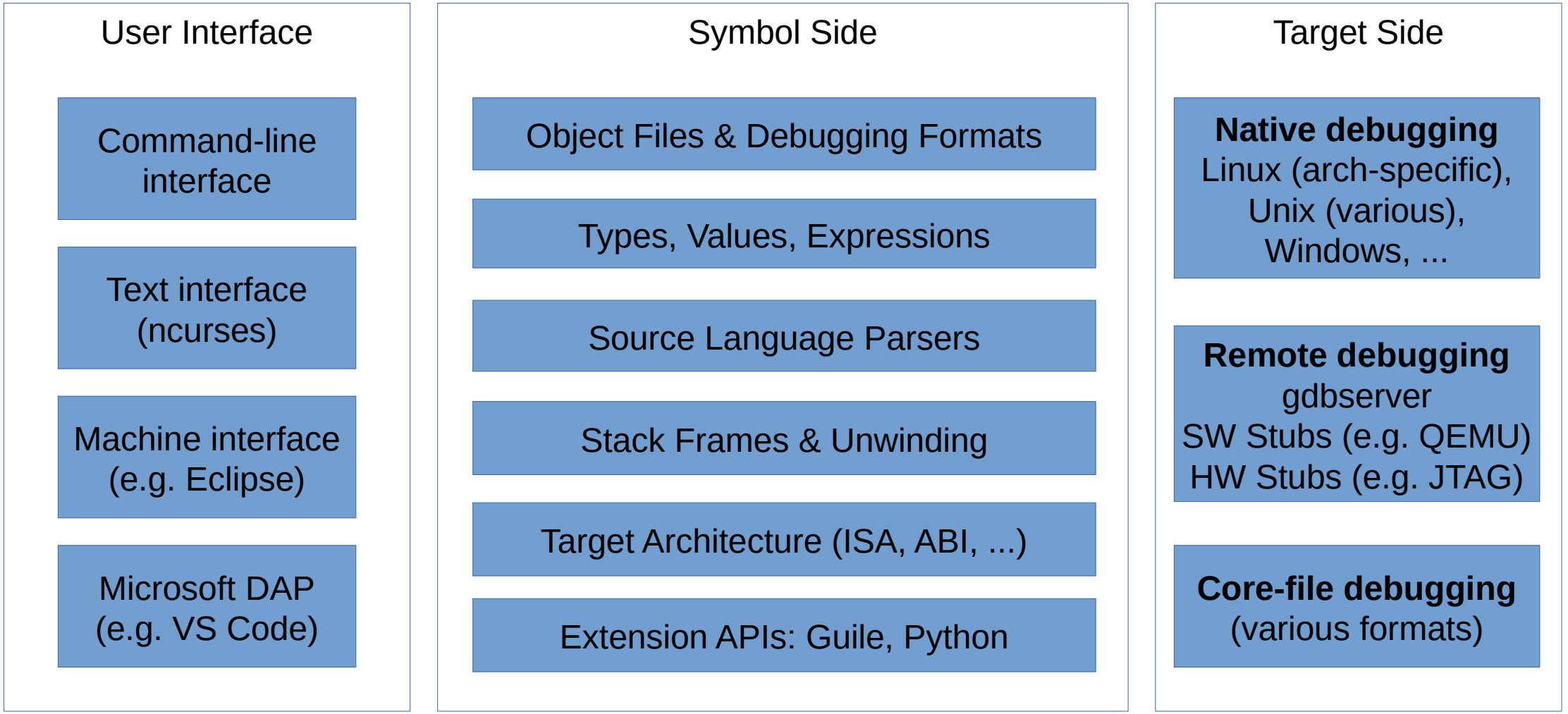
GDB Overview

GDB Overall Structure
GDB Target Interface & Remote Protocol
GDB User-Space Library Interfaces

GDB Overall Structure



GDB Target Interface

- Manage “inferiors” (processes or other entities being debugged)
  - Start up new inferior
  - Attach to or detach from existing inferior
  - Detect fork & exec events
  - Query (kernel-level) threads and detect thread startup/exit events
- Manage execution state
  - Stop / resume / wait on inferior (per thread)
  - Handle (hardware/kernel-level) single-stepping
  - Handle (hardware/kernel-level) breakpoints & watchpoints
- Raw “bits & bytes” access
  - Read/write register contents (per thread)
  - Read/write memory contents (per process)
  - Misc. target/OS features like address spaces, memory maps, memory tags, thread-local storage, signals, AUXV data, dynamic linker data, …
- Miscellaneous special features
  - Remote file system access primitives
  - Tracepoints and agent expressions
  - Reverse debugging / record & replay

Note: GDB remote protocol is basically a text-based serialization of the target interface via a serial or network connection.

GDB User-Space Library Interfaces

- Some targets use a user-space dynamic loader to handle shared libraries (e.g. ld.so)
  - GDB needs to query data managed by the loader (e.g. list of currently loaded libraries)
  - GDB needs to be notified whenever this data changes (e.g. due to dlopen/dlclose)
  - No kernel involvement – therefore GDB does not use target interface
    - Instead, GDB observes the operation of the user-space loader via debugger means
  - Dynamic loader uses a well-defined ABI to interact with the debugger
    - Linked list of shared libraries using well-defined data type
    - Head of the linked list pointed to by well-known symbol
    - Loader calls well-known function whenever list changes – GDB sets breakpoint
- Some targets use a user-space library to handle threads (e.g. libpthread)
  - GDB uses target interface to handle **kernel** thread, user-space ABI for **user** threads
  - User-space ABI similar to dynamic loader case
    - However, details abstracted into a helper library (libthread_db)
    - Helper provided by libpthread package, used by GDB
    - GDB callbacks to provide register/memory access primitives
- JIT code generators may use similar user-space ABI to allow GDB to debug JITed code
  - JIT generates in-memory object file(s) (e.g. ELF/DWARF) describing JITed code
  - JIT manages linked list pointing to those objects using well-defined data types
  - JIT calls well-known function (GDB breakpoint) whenever that list changes

?