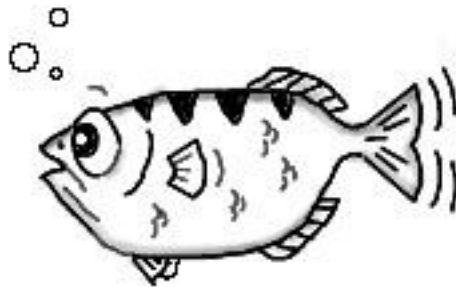


GDB Reverse Debug and Process Record Target



HelloGcc Workshop

October 24, 2009

Hui Zhu

teawater@gmail.com

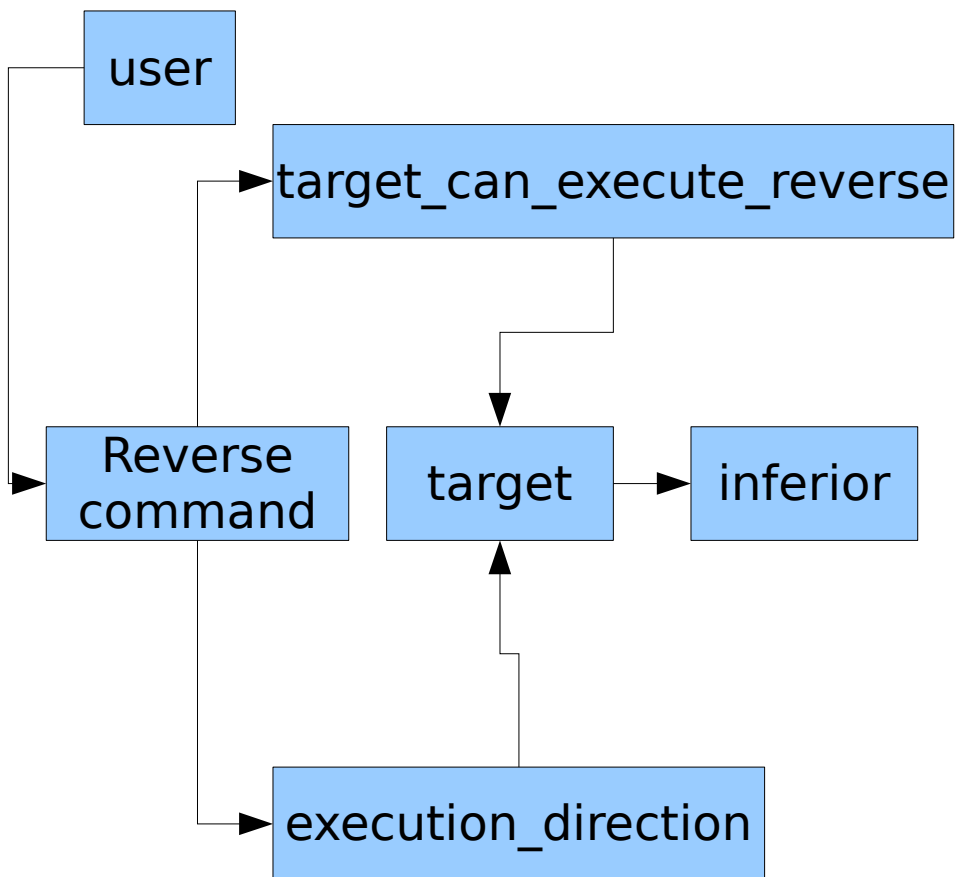
Agenda

- Function show
- Reverse debug introduction
- Process record and replay target introduction
- How to port process record and replay target to an architecture and an OS
- To Do List

Function show

- Base reverse function
reverse-step reverse-next reverse-finish
breakpoint
- Dump function
- <http://sourceware.org/gdb/wiki/ProcessRecord/Tutorial>

Reverse debug introduction



- **reverse.c**

`target_can_execute_reverse`

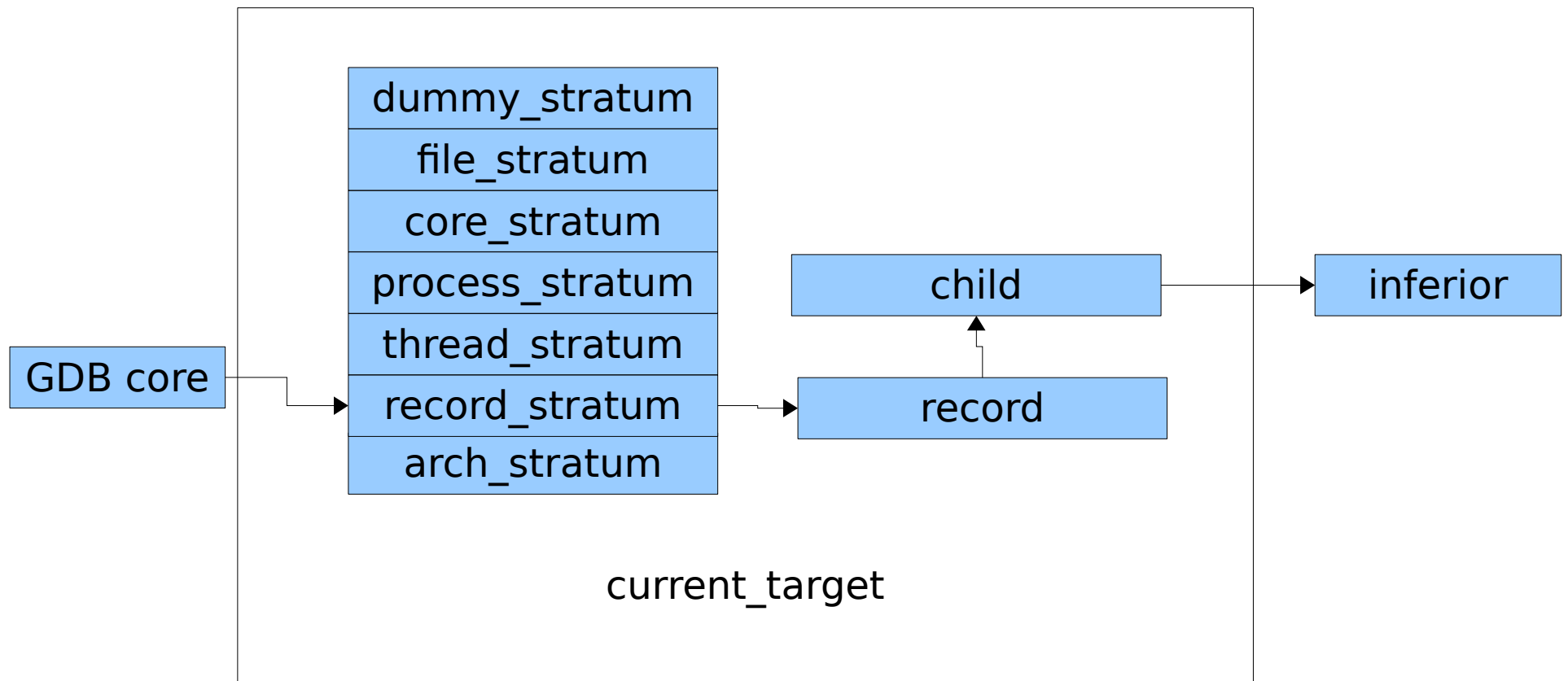
`execution_direction`

- **infrun.c**

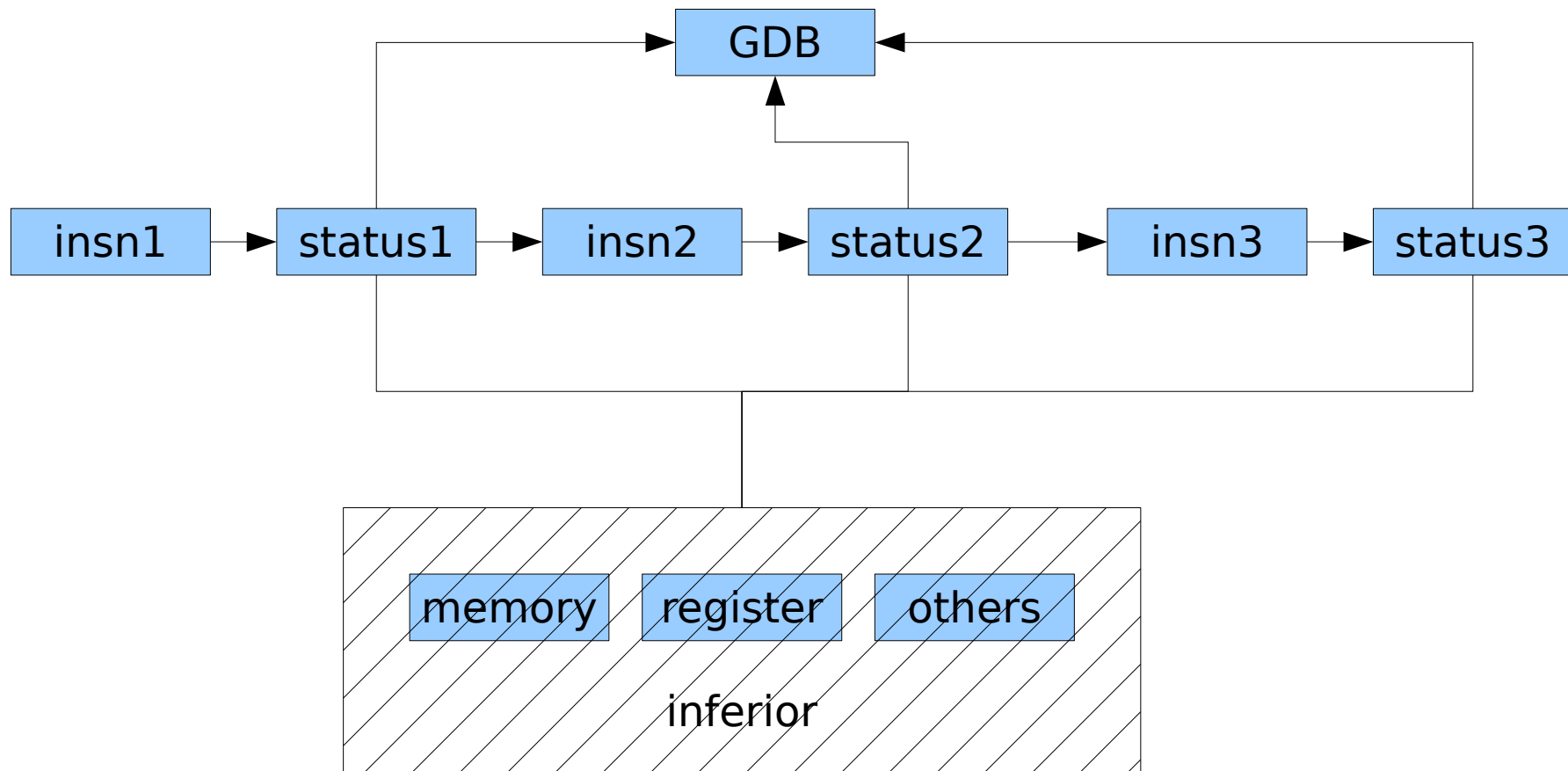
`handle_inferior_event`

```
if (execution_direction == EXEC_REVERSE)
{
.....
}
```

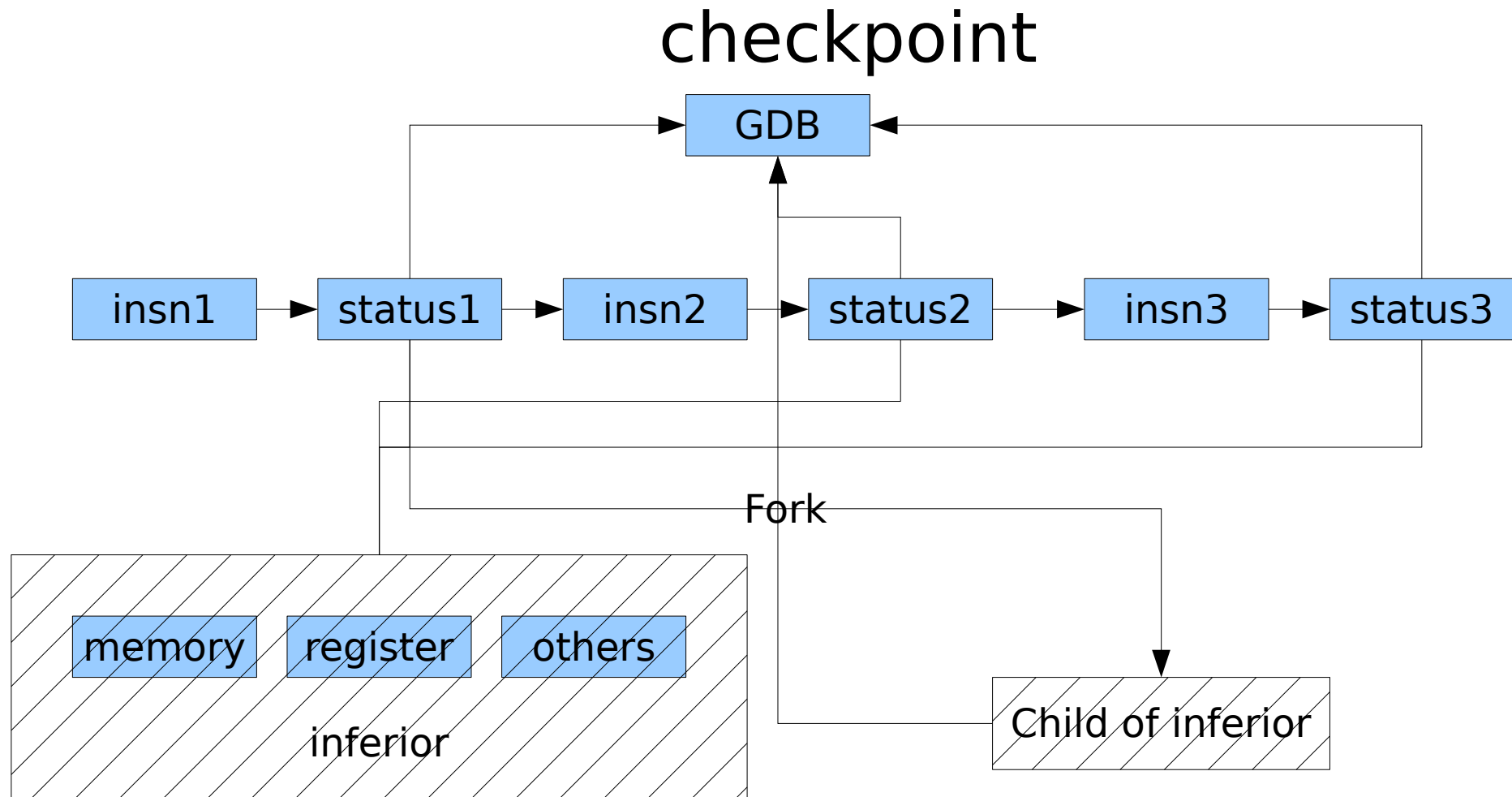
Process record and replay target introduction



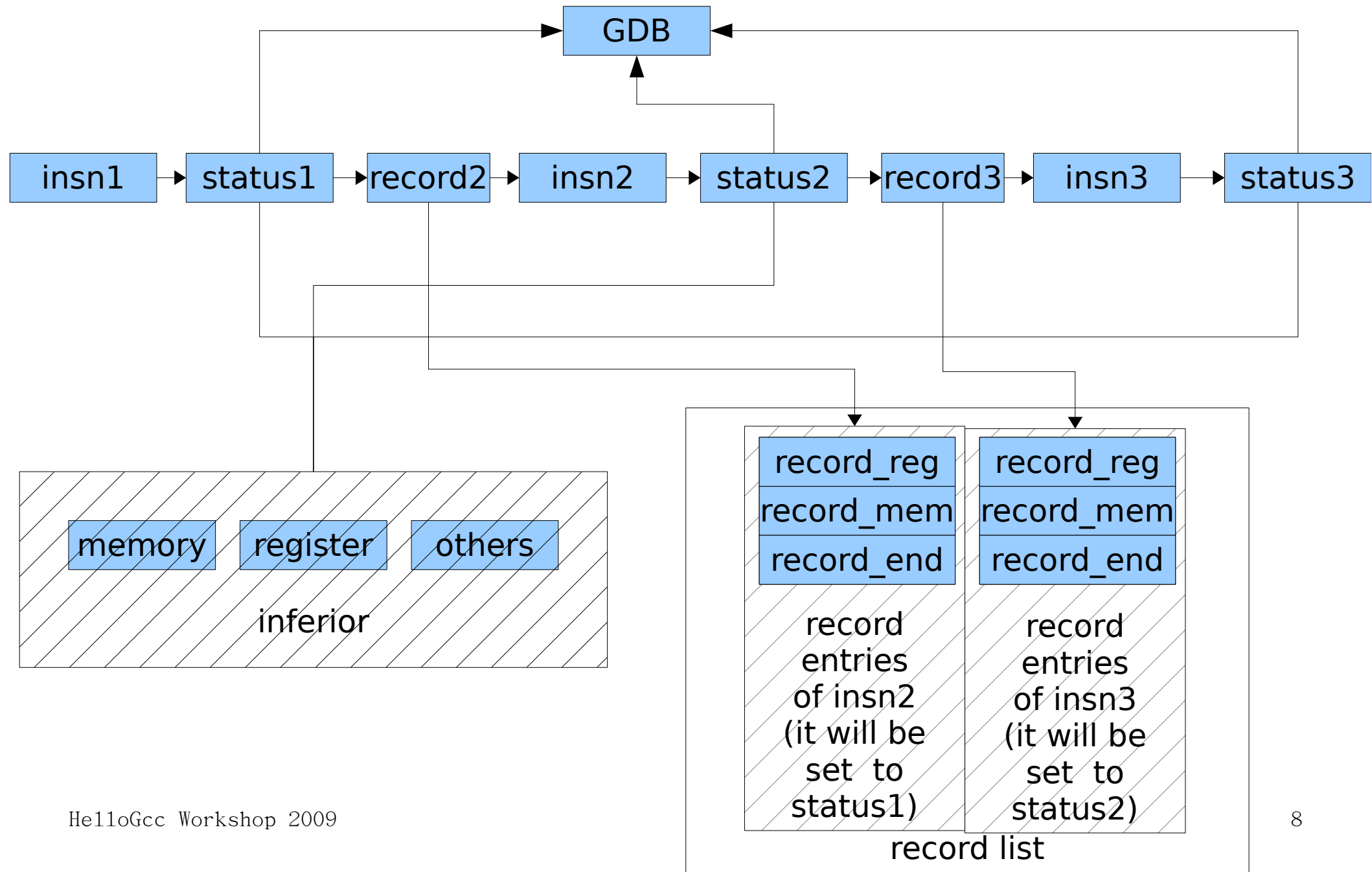
GDB debug a inferior



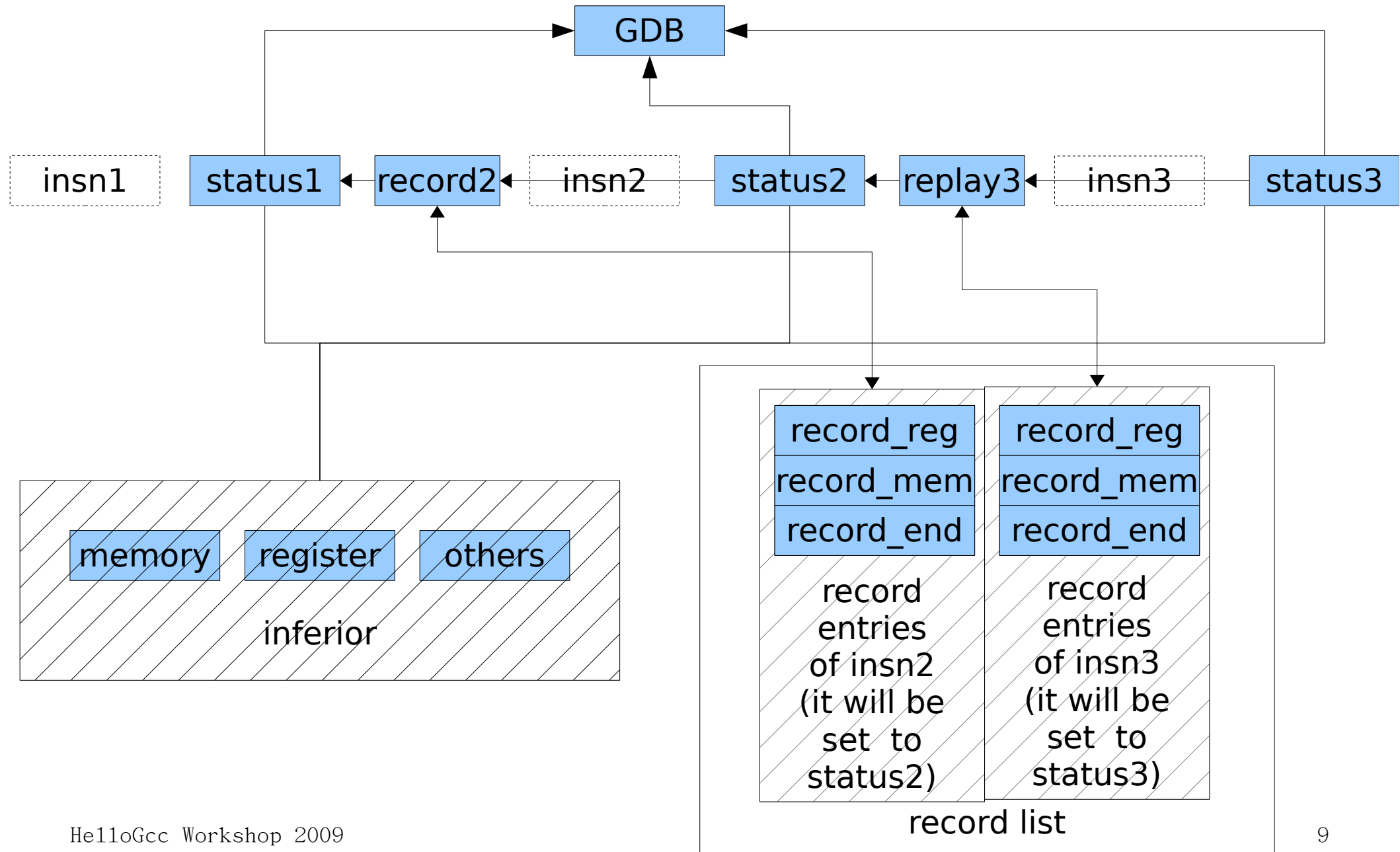
Process record and replay target introduction



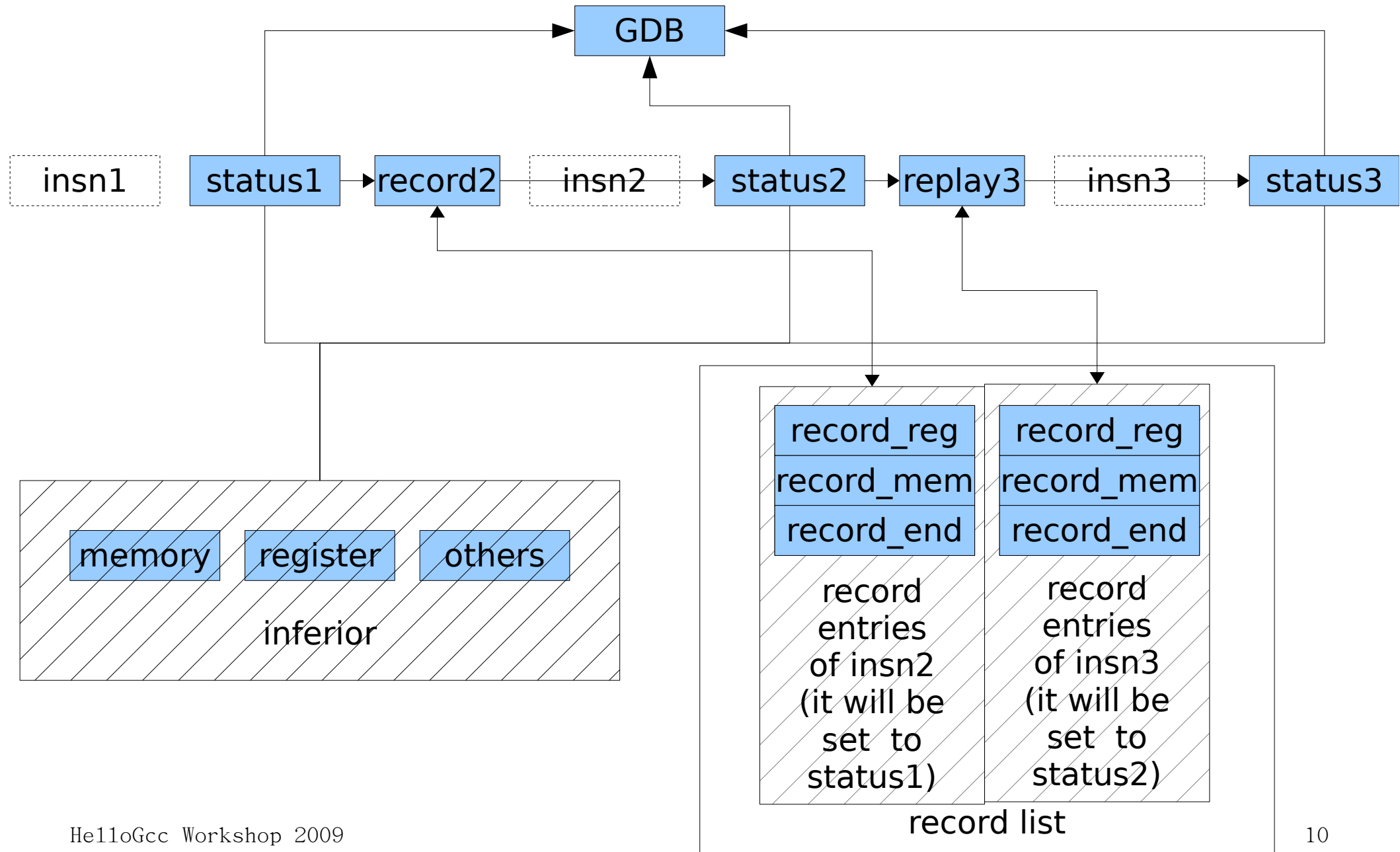
Prec record mode



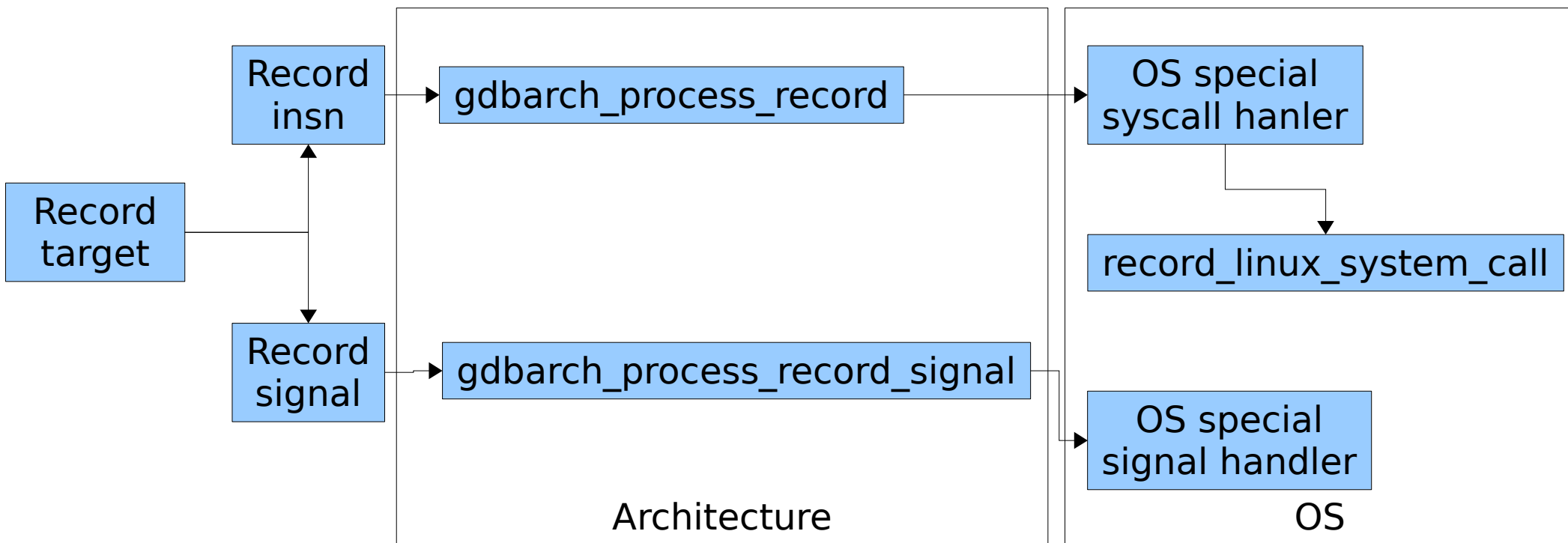
Prec replay reverse mode



Prec replay forward mode



How to port process record and replay target to an architecture and an OS



gdbarch_process_record

```
typedef int (gdbarch_process_record_ftype) (struct gdbarch *gdbarch, struct regcache  
      *regcache, CORE_ADDR addr);
```

Call function `set_gdbarch_process_record` to register the `gdbarch_process_record_ftype` function to `gdbarch`.

Parse the instruction at `ADDR` storing in the record execution log the registers `REGCACHE` and memory ranges that will be affected when the instruction executes, along with their current values.

Return -1 if something goes wrong, 0 otherwise.

Example:

```
i386-linux-tedp.c:set_gdbarch_process_record (gdbarch, i386_process_record);
```

OS special syscall handler

Example:

```
i386-linux-tdep.c:tdep->i386_intx80_record  
= i386_linux_intx80_sysenter_record;
```

record_linux_system_call

Int record_linux_system_call (enum gdb_syscall syscall, struct regcache *regcache, struct linux_record_tdep *tdep)

When the architecture process record get a Linux syscall instruction, it will get a Linux syscall number of this architecture and convert it to the Linux syscall number "num" which is internal to GDB. Most Linux syscalls across architectures in Linux would be similar and mostly differ by sizes of types and structures. This sizes are put to "tdep".

Record the values of the registers and memory that will be changed in current system call.

Return -1 if something wrong.

struct linux_record_tdep

```
struct linux_record_tdep
{
    /* The size of the type that will be used in a system call. */
    ...
    /* The values of the second argument of system call "sys_ioctl". */
    ...
    /* The values of the second argument of system call "sys_fcntl" and "sys_fcntl64". */
    ...
    /* The number of the registers that are used as the arguments of a system call. */
    ...
}
```

gdbarch_process_record_signal

```
typedef int (gdbarch_process_record_signal_ftype) (struct gdbarch *gdbarch, struct  
    regcache *regcache, enum target_signal signal);
```

Call function `set_gdbarch_process_record_signal` to register the `gdbarch_process_record_signal_ftype` function to `gdbarch`.

Save process state after a signal.

Return -1 if something goes wrong, 0 otherwise.

Example:

```
set_gdbarch_process_record_signal (gdbarch, i386_linux_record_signal);
```


OS special signal handler

Example:

i386-linux-

```
tdep.c:set_gdbarch_process_record_signal  
(gdbarch, i386_linux_record_signal);
```

To Do List

- Improve support for intel architectures. There is a patch awaiting approval that will add
- Add support for more processor architectures (mips, arm etc.)
- Add support for more os/abis (currently only linux is supported).
- Improve support for memory free (sys_brk).
- Improve support for multi-thread and multi-process record/replay.
- Improve performance (speed and memory usage).
- Save execution log to a file, and restore it later for replay.
- Add more test cases to the testsuite.
- Improve documentation.
- <http://sourceware.org/gdb/wiki/ReverseDebug>
- <http://sourceware.org/gdb/wiki/ProcessRecord>

Thanks!