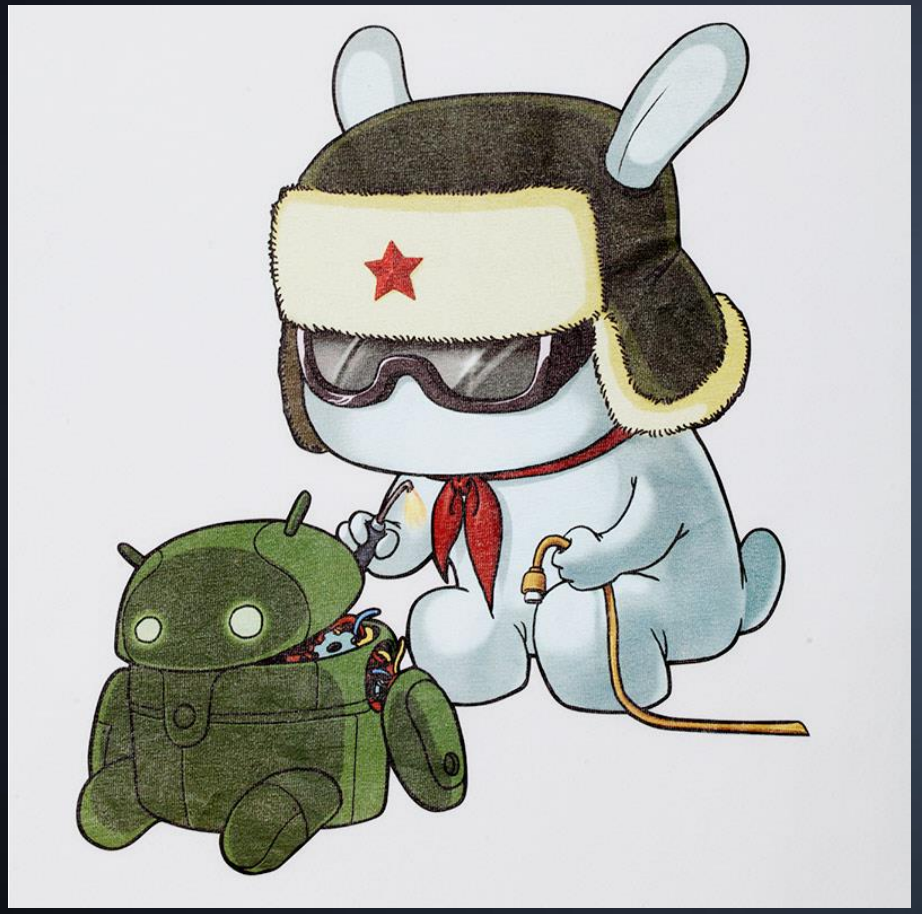
支持任意数量watchpoint的建议



朱辉
zhuhui@xiaomi.com
teawater.github.io

现在Linux下watchpoint功能（见下图）

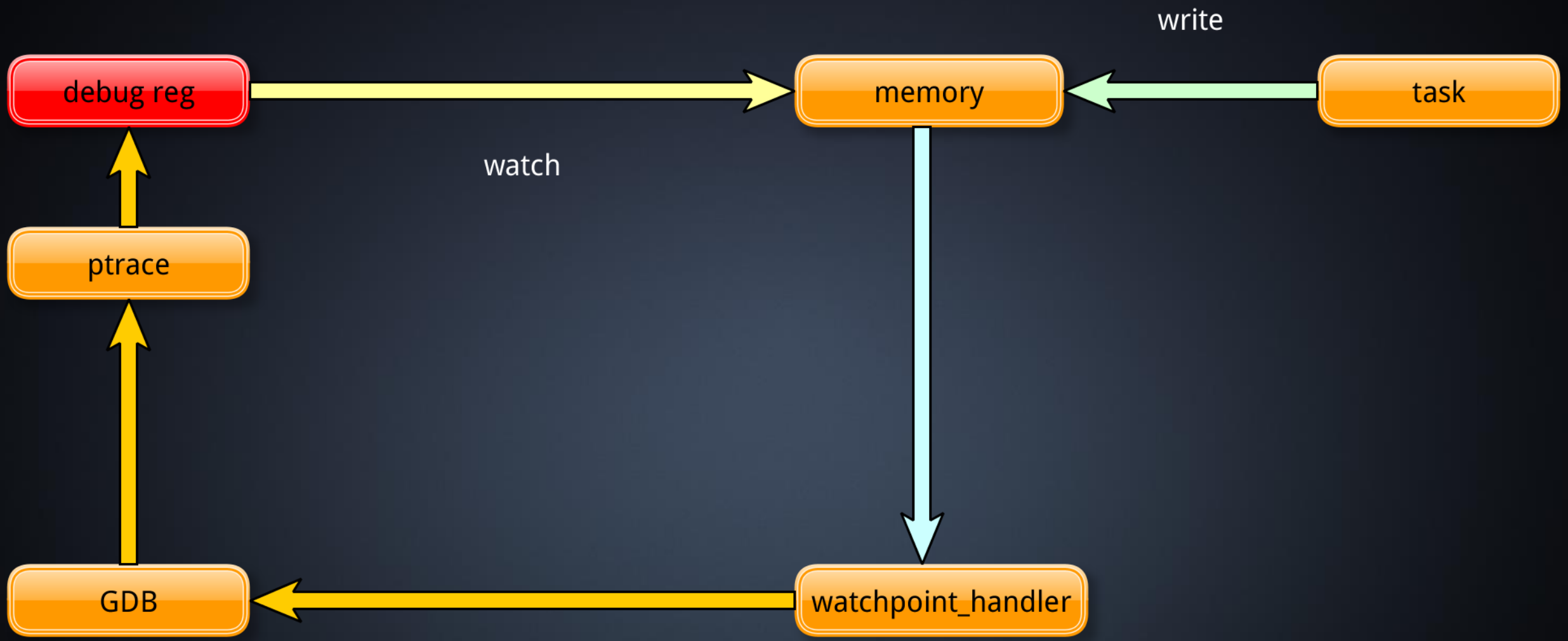
- GDB通过ptrace写地址和相关信息到调试寄存器控制硬件watchpoint。

  程序运行，写watch的地址，CPU抛出异常。

  内核处理异常。

  返回GDB，GDB再做处理，最后显示watchpoint。

- 缺点：CPU的硬件watchpoint只有几个，有数量限制。


write
debug reg
memory
task
watch
ptrace
GDB
watchpoint_handler

顺便讲个ARM八卦

- 这一段是ARM取watchpoint寄存器数量的代码。
- ARM调试结构7.1以前因为没有直接接口判断哪个watchpoint被触发，所以只支持了一个watchpoint。
- 可以通过反汇编分析代码，不过我估计这工作太辛苦没人肯做。
- 另外疑似DFAR标记出了哪个被触发，但是ARMARM说是UNKNOWN。

```
/* Determine number of usable WRPs available. */
static int get_num_wrps(void)
{
        /*
         * On debug architectures prior to 7.1, when a watchpoint fires, the
         * only way to work out which watchpoint it was is by disassembling
         * the faulting instruction and working out the address of the memory
         * access.
         *
         * Furthermore, we can only do this if the watchpoint was precise
         * since imprecise watchpoints prevent us from calculating register
         * based addresses.
         *
         * Providing we have more than 1 breakpoint register, we only report
         * a single watchpoint register for the time being. This way, we always
         * know which watchpoint fired. In the future we can either add a
         * disassembler and address generation emulator, or we can insert a
         * check to see if the DFAR is set on watchpoint exception entry
         * [the ARM ARM states that the DFAR is UNKNOWN, but experience shows
         * that it is set on some implementations].
         */
        if (get_debug_arch() < ARM_DEBUG_ARCH_V7_1)
          return 1;

        return get_num_wrp_resources();
}
```

解决这个问题的方案来源（见下图）

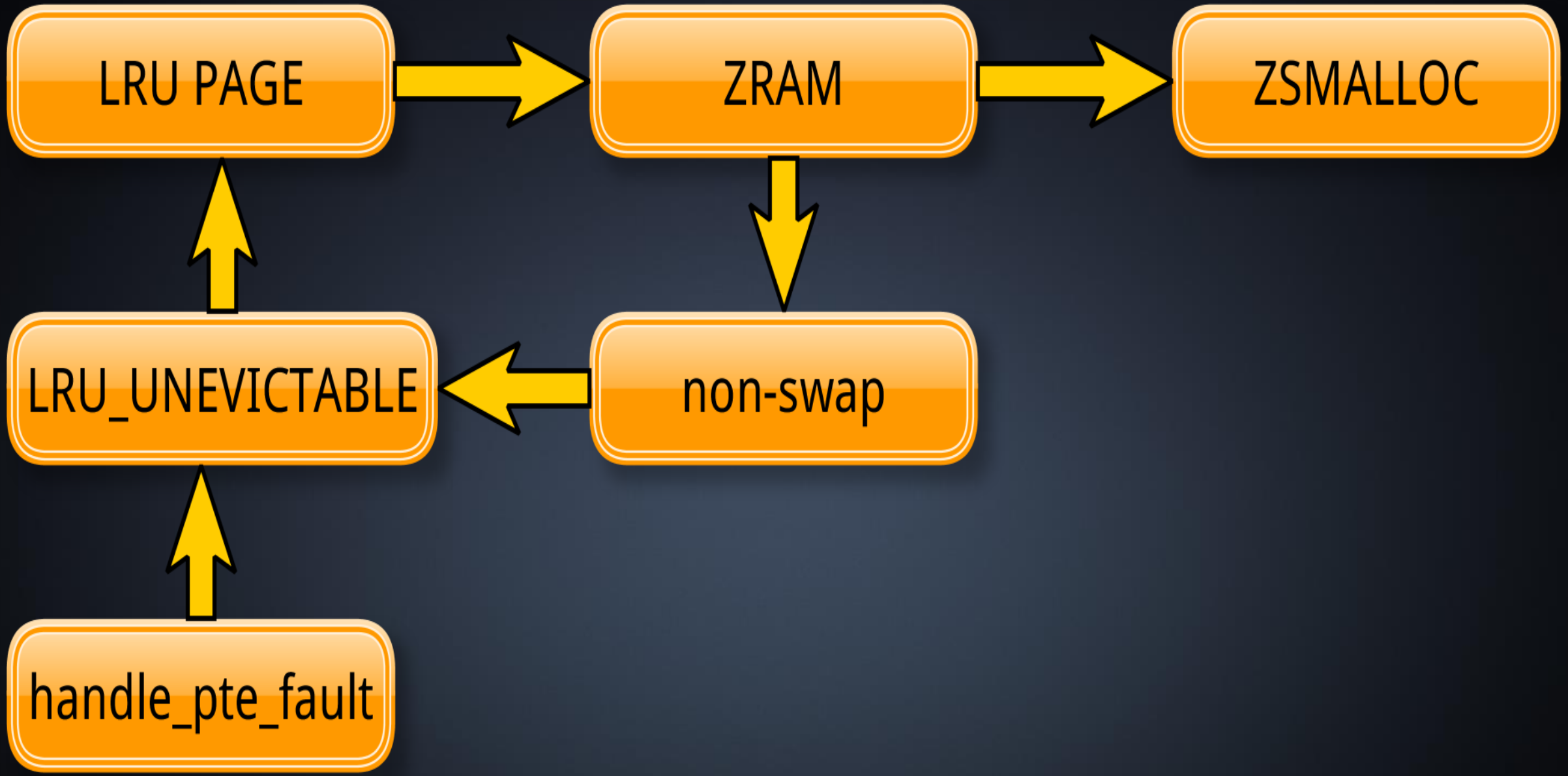
- 我为了提高ZRAM压缩率而做的ZRAM Non Swap功能。
- 其思路是把压缩率不高的页面不写入ZRAM，同时将其从LRU列表中抽出去放到单独的页列表保证其不会再次被写入ZRAM。

  当这些页面再次被写时，表明其有可能在压缩率上出现变化，将其再次放回LRU列表。

- PATCH： https://lkml.org/lkml/2016/8/22/151

监视页面被写的方法（见下图）

- 为了监视NonSwap页面是否被写。
- 让内核中将页面pte标志设置为只读。
- 当写的时候就会触发异常，内核就可以对这次写作出处理。


LRU PAGE
ZRAM
ZSMALLOC
LRU_UNEVICTABLE
non-swap
handle_pte_fault

新的watchpoint方法（见下图）

- 既然可以监视页面被写，也可以用来实现watchpoint，只需要将调试寄存器换成pte。
- 整个流程变为：

  GDB通过接口将要监视的地址等信息传给内核。

  内核存储这些信息，并对页面进行处理记录，将页面PTE标记为只读。

  程序运行，写watch的地址，CPU抛出异常。

  系统对这个写异常进行处理，确定是否写的是watch地址，如果是返回GDB。

  GDB再做处理，最后显示watchpoint。
- 优点：

  应用程序每个页面都有PTE，理论上可以设置无限数量的watchpoint。

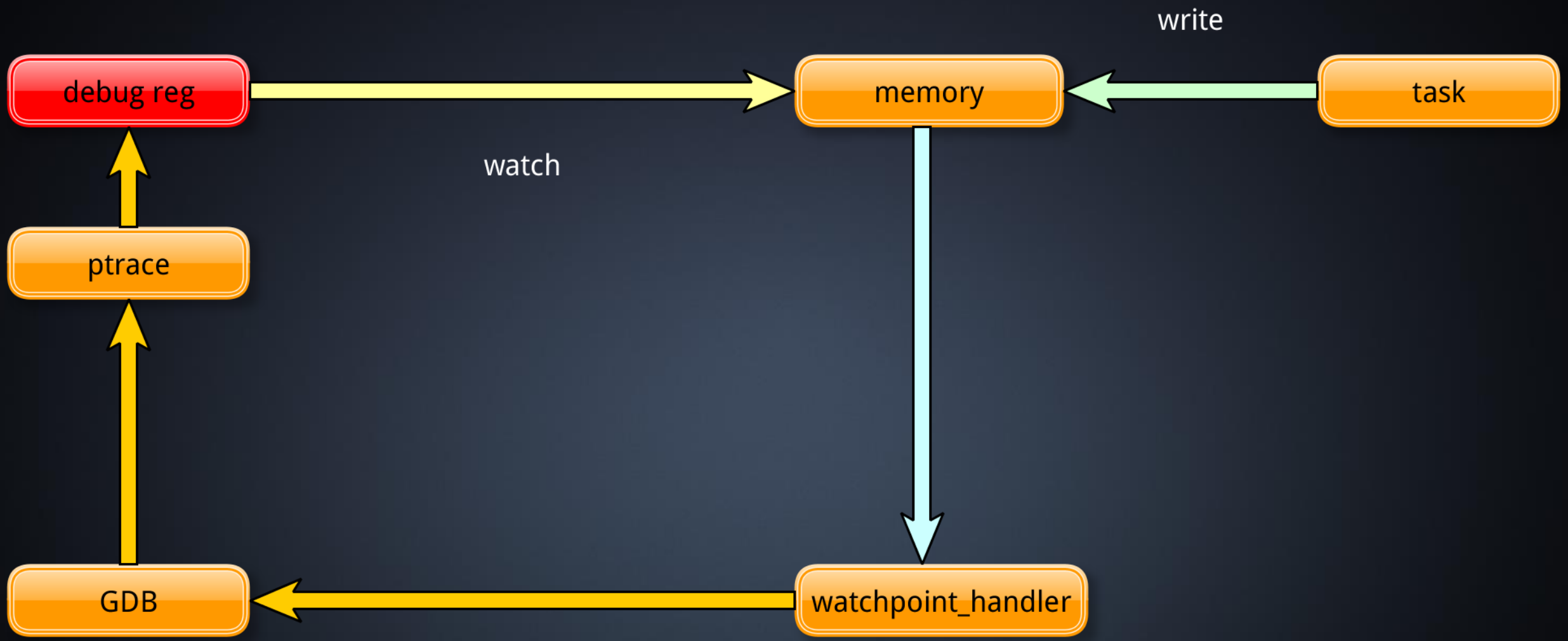
  不依赖于特定体系结构，基本上支持MMU的CPU都可以使用。

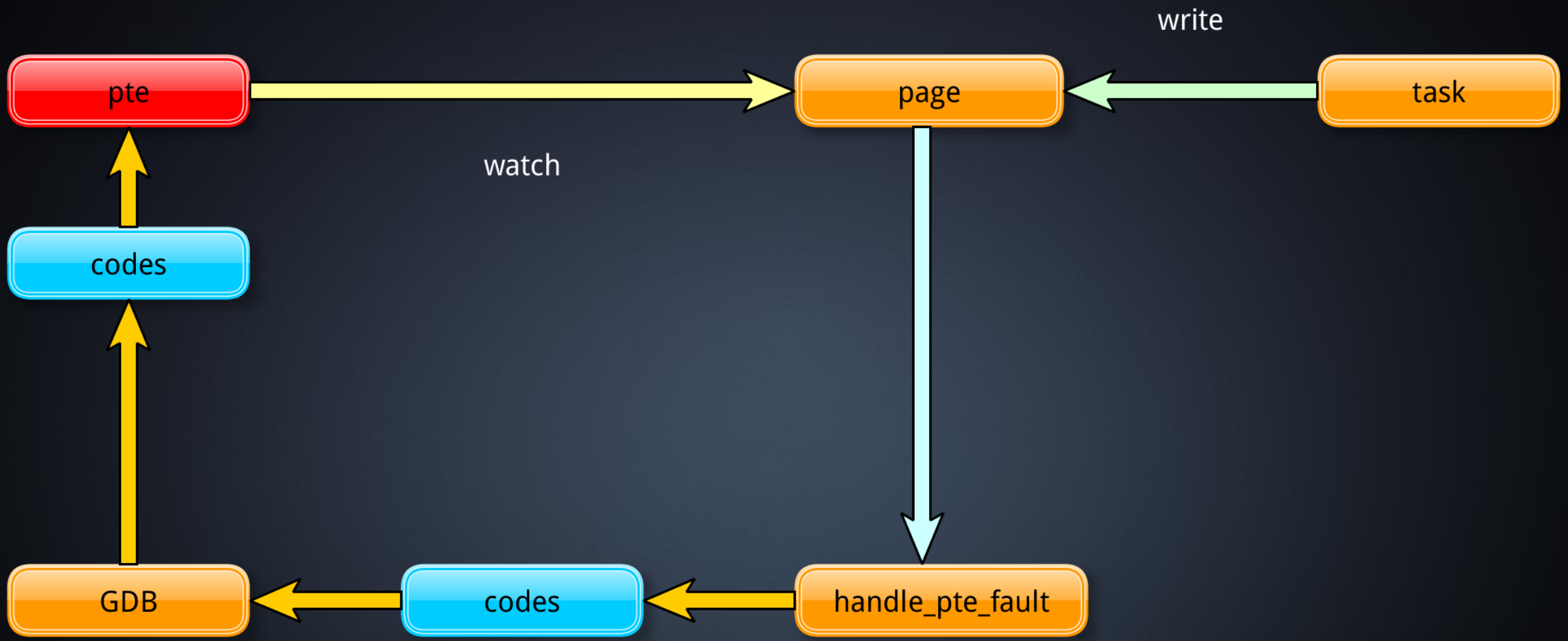
  PTE异常有地址，可以比较方便判断哪个watchpoint被触发。
- 缺点：

  PTE是针对一个页面触发，页面任何地址被写都会触发异常，再做判断，速度会比调试寄存器慢。

  因为内核大部分内存都是直接访问（vmalloc内存除外），所以不能监控内核中的内存读写。

  就差一个程序员啦。:P


write
debug reg
memory
task
watch
ptrace
GDB
watchpoint_handler


write
pte
page
task
watch
codes
GDB
codes
handle_pte_fault

谢谢！问题？

weibo: @teawater_z 欢迎在线吐槽

小米电视招聘内核优化工程师，欢迎发简历到 zhuhui@xiaomi.com。