When do you need to notify inside page table lock ?

When clearing a pte/pmd we are given a choice to notify the event through (notify version of *_clear_flush call mmu_notifier_invalidate_range) under the page table lock. But that notification is not necessary in all cases.

For secondary TLB (non CPU TLB) like IOMMU TLB or device TLB (when device use thing like ATS/PASID to get the IOMMU to walk the CPU page table to access a process virtual address space). There is only 2 cases when you need to notify those secondary TLB while holding page table lock when clearing a pte/pmd:

A. page backing address is free before mmu_notifier_invalidate_range_end()
B. a page table entry is updated to point to a new page (COW, write fault on zero page, __replace_page(), ...)

Case A is obvious you do not want to take the risk for the device to write to a page that might now be used by some completely different task.

Case B is more subtle. For correctness it requires the following sequence to happen:

- take page table lock
- clear page table entry and notify ([pmd/pte]p_huge_clear_flush_notify())
- set page table entry to point to new page

If clearing the page table entry is not followed by a notify before setting the new pte/pmd value then you can break memory model like C11 or C++11 for the device.

Consider the following scenario (device use a feature similar to ATS/PASID):

Two address addrA and addrB such that |addrA - addrB| >= PAGE_SIZE we assume they are write protected for COW (other case of B apply too).

```
[Time N] --------------------------------------------------------------------
CPU-thread-0  {try to write to addrA}
CPU-thread-1  {try to write to addrB}
CPU-thread-2  {}
CPU-thread-3  {}
DEV-thread-0  {read addrA and populate device TLB}
DEV-thread-2  {read addrB and populate device TLB}
[Time N+1] ------------------------------------------------------------------
CPU-thread-0  {COW_step0: {mmu_notifier_invalidate_range_start(addrA)}}
CPU-thread-1  {COW_step0: {mmu_notifier_invalidate_range_start(addrB)}}
CPU-thread-2  {}
CPU-thread-3  {}
DEV-thread-0  {}
DEV-thread-2  {}
[Time N+2] ------------------------------------------------------------------
CPU-thread-0  {COW_step1: {update page table to point to new page for addrA}}
CPU-thread-1  {COW_step1: {update page table to point to new page for addrB}}
CPU-thread-2  {}
CPU-thread-3  {}
DEV-thread-0  {}
DEV-thread-2  {}
[Time N+3] ------------------------------------------------------------------
CPU-thread-0  {preempted}
CPU-thread-1  {preempted}
CPU-thread-2  {write to addrA which is a write to new page}
CPU-thread-3  {}
DEV-thread-0  {}
DEV-thread-2  {}
[Time N+3] ------------------------------------------------------------------
CPU-thread-0  {preempted}
CPU-thread-1  {preempted}
CPU-thread-2  {}
CPU-thread-3  {write to addrB which is a write to new page}
DEV-thread-0  {}
DEV-thread-2  {}
[Time N+4] ------------------------------------------------------------------
CPU-thread-0  {preempted}
CPU-thread-1  {COW_step3: {mmu_notifier_invalidate_range_end(addrB)}}
CPU-thread-2  {}
CPU-thread-3  {}
DEV-thread-0  {}
DEV-thread-2  {}
[Time N+5] ------------------------------------------------------------------
CPU-thread-0  {preempted}
CPU-thread-1  {}
CPU-thread-2  {}
CPU-thread-3  {}
```

```
   DEV-thread-0  {read addrA from old page}
   DEV-thread-2  {read addrB from new page}
```

So here because at time N+2 the clear page table entry was not pair with a notification to invalidate the secondary TLB, the device see the new value for addrB before seeing the new value for addrA. This break total memory ordering for the device.

When changing a pte to write protect or to point to a new write protected page with same content (KSM) it is fine to delay the mmu_notifier_invalidate_range call to mmu_notifier_invalidate_range_end() outside the page table lock. This is true even if the thread doing the page table update is preempted right after releasing page table lock but before call mmu_notifier_invalidate_range_end().