Integrating KVM with the Linux Memory Management

Qumranet Inc.

Andrea Arcangeli
andrea@qumranet.com
KVM Forum 2008
Napa Valley, California
12 June 2008

KVM Forum
Napa 2008

KVM VM integration

- KVM as a kernel module is almost *self contained*
- KVM is generally *not intrusive* to the Linux VM and most of other Linux kernel code
- As we add more advanced features (and as we optimize for performance) we need a closer integration with the Linux Kernel (whenever possible we try in a generic way not tied to KVM)
- We'll cover in this presentation how we intend to provide the needed integration with the Linux VM using a functionality we called "***MMU Notifier***"
- Over time this also was sometime called "mm notifier", "EMM", and other aliases, but the objective of all those patches was the same

Guest Physical Memory

- All the virtualized guest physical ram (what guest thinks it is ram, and by far the largest amount of KVM RSS) is allocated with a single memalign() glibc call (memalign() is the same as malloc() but it provides page alignment)
- Allocating guest physical ram with a malloc equivalent and finding the physical pages to map into sptes with get_user_pages() allows KVM memory to be ***almost*** transparent to the Linux VM
- Linux immediately tries to swap the malloced memory representing the guest physical memory like if it was any other malloc done by any other task in the system, often it succeeds

MMU Notifier objective

- The objective of the 'MMU Notifier' is to make the KVM malloced memory (representing the guest physical ram) ***entirely*** transparent to the Linux VM
- The Linux VM will then work the same regardless if it's paging KVM guest mapped memory or any other regular Linux task
- The 'MMU Notifier' functionality can be also used by other subsystems like GRU and XPMEM to export the user virtual address space of computational tasks to other nodes
- This will also allow KVM guest physical ram itself to be exported to other nodes through GRU and XPMEM or any other RDMA engine

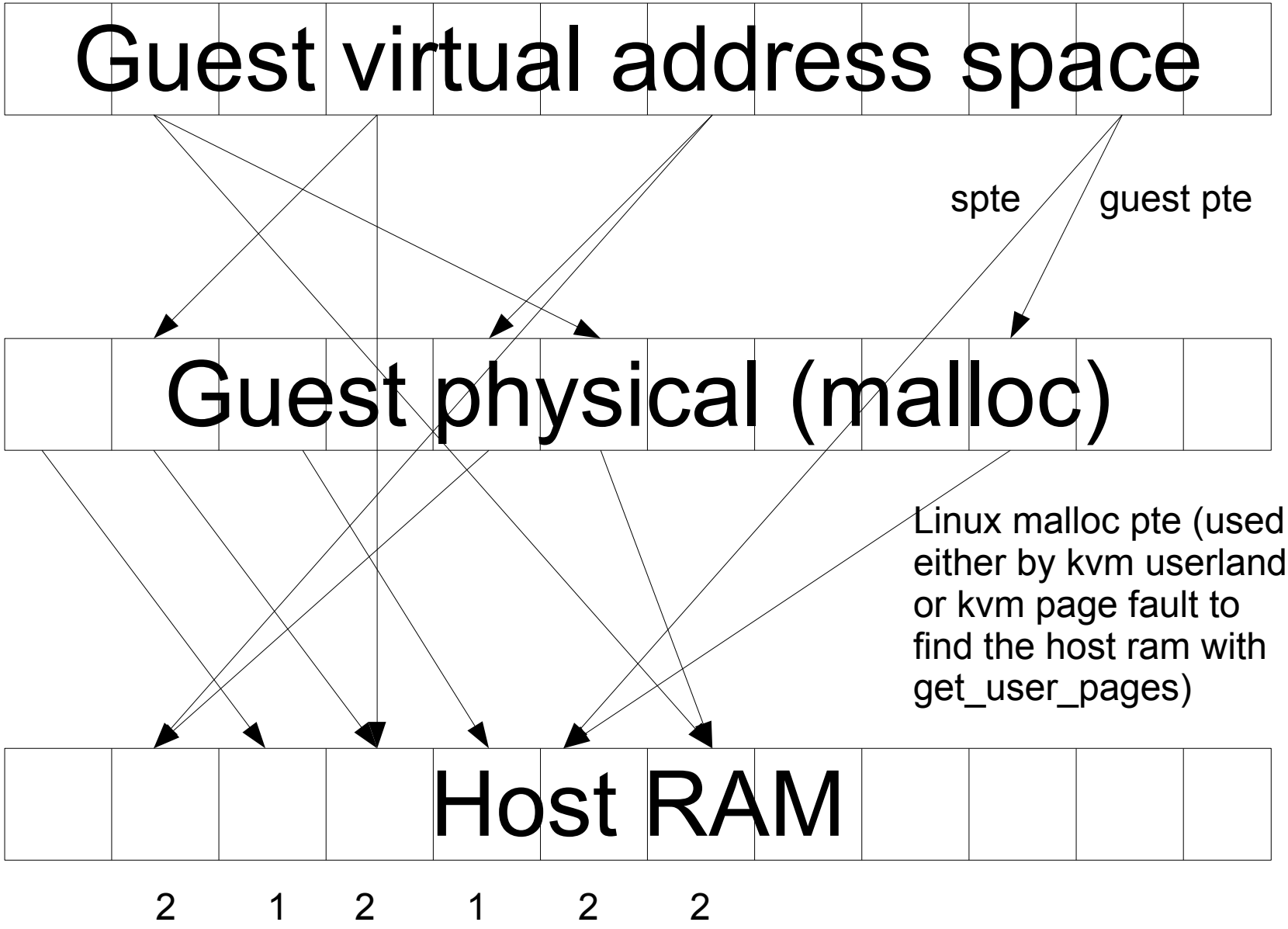
Avoidance of memory pinning

- It's not like RDMA or KVM can't work without MMU Notifiers, but whenever the memory is mapped by any secondary MMU (either through secondary sptes or secondary tlbs, or both like in KVM case), the drivers have to pin the pages to be safe
- Pinning the pages means all pages mapped in the secondary MMU cannot be swapped or paged out
- Without 'MMU Notifier' functionality in the host kernel, the KVM task is not really entirely pageable, and in fact it's possible for guests to make the 'malloced' region almost completely unswappable by touching all guest physical pages in a loop

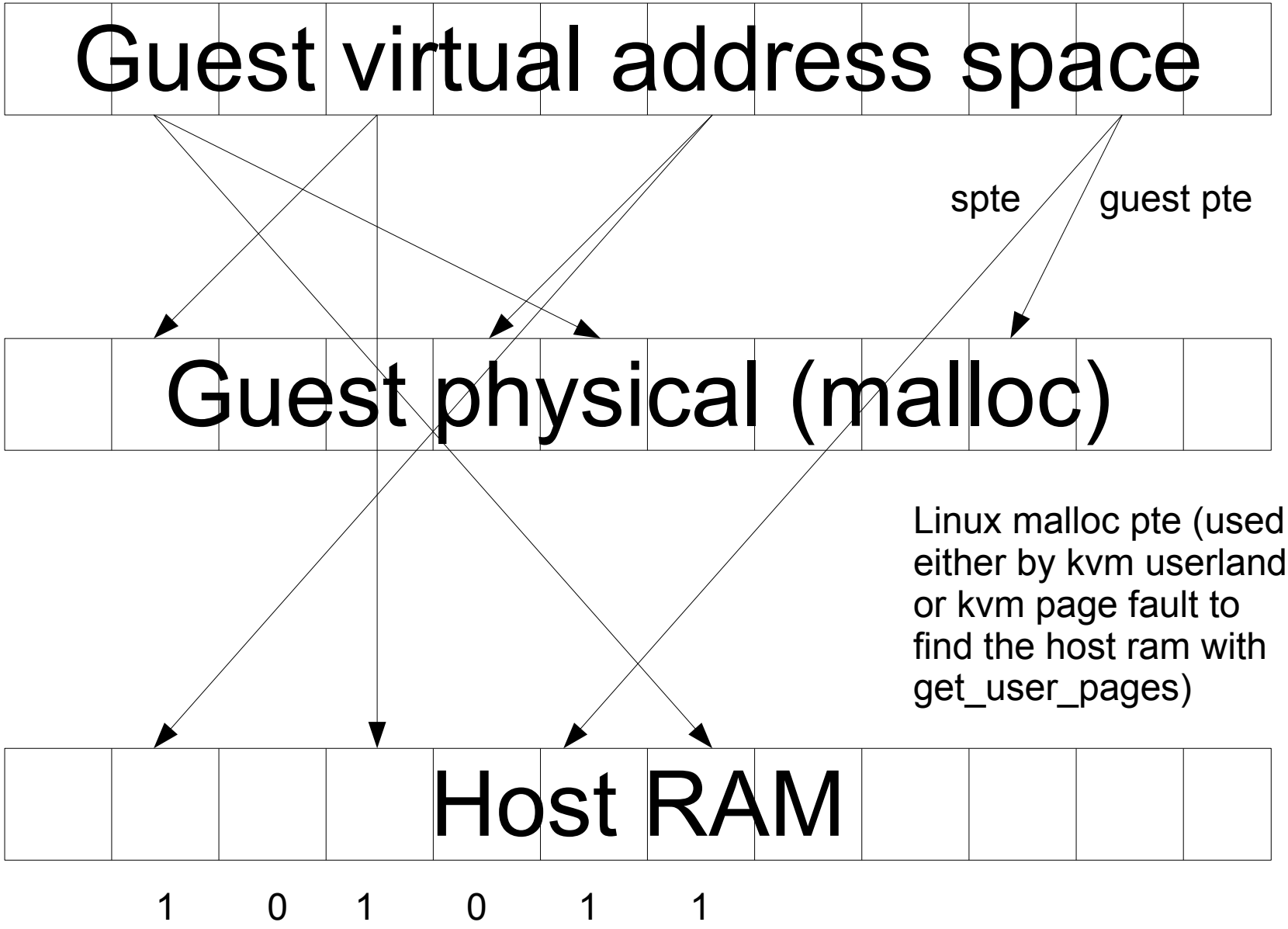
KVM page fault and sptes

- The KVM page fault is the one that instantiates the shadow pagetables
- Shadow pagetables works similarly to a TLB
- They translate a virtual (or physical with EPT/NPT) guest address to a physical host address
- They can be discarded at any time and they will be recreated later as new KVM page fault triggers, just like the primary CPU TLB can be flushed at any time and the CPU will refill it from the ptes
- The sptes are recreated by the KVM page fault by calling get_user_pages (i.e. looking at the Linux ptes) to translate a guest physical address (the malloced region) to a host physical address

VM layout with sptes



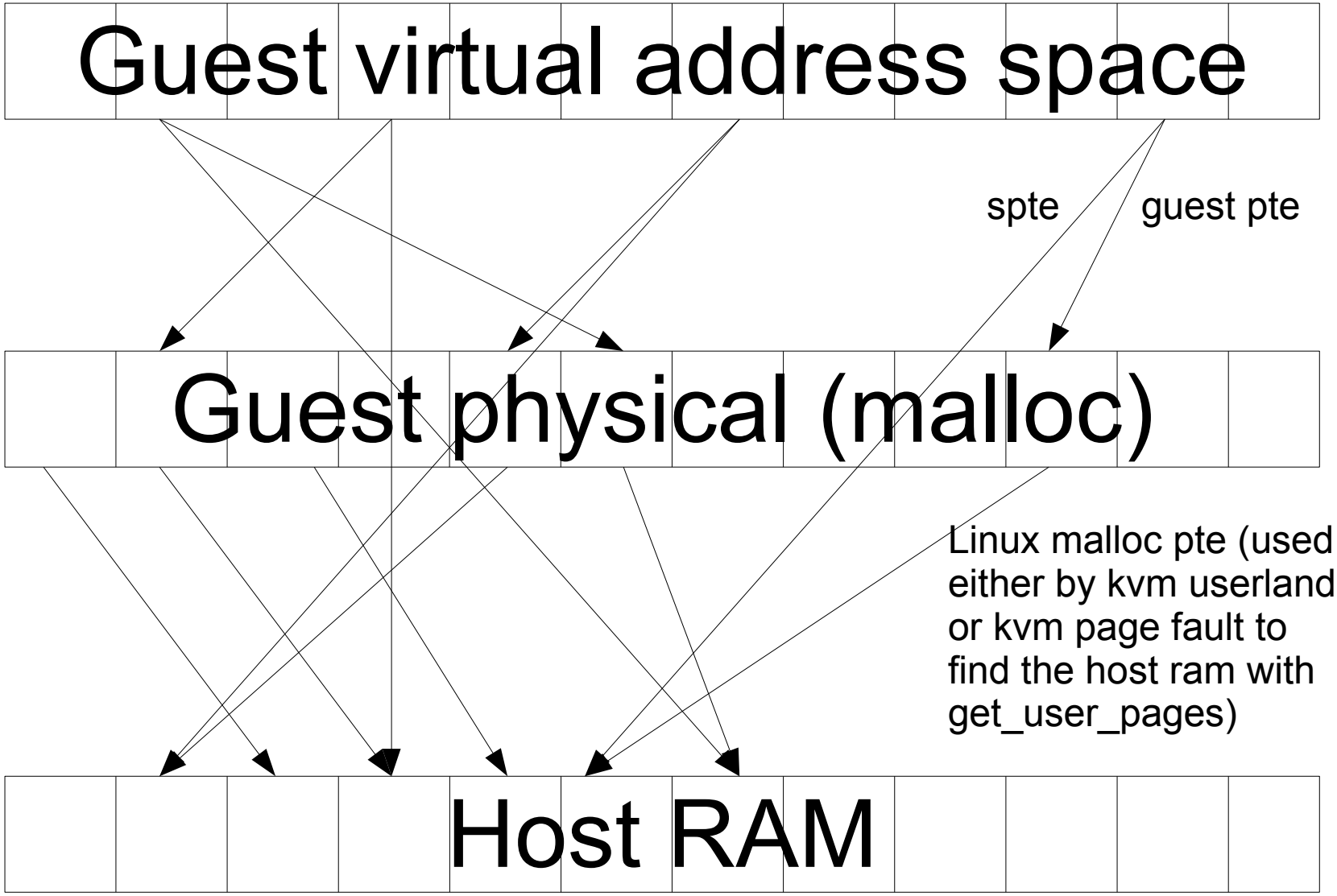
VM swap attempt with spte pins



sptes must be unmapped too

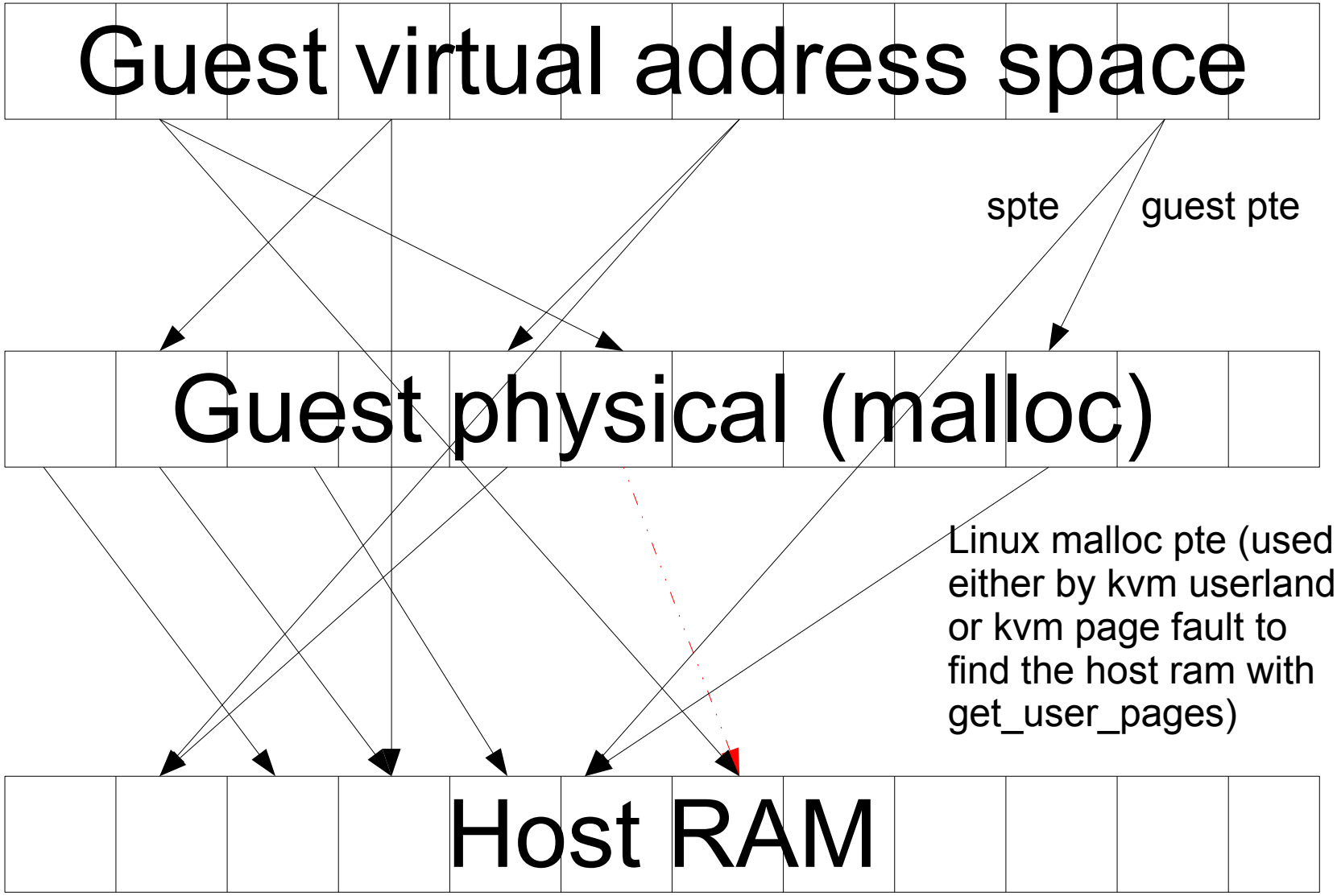
- The Linux VM is unaware of Linux sptes
- The VM is successful at unmapping the Linux ptes (the one used by the CPU when KVM userland accesses the guest physical memory)
- But the sptes must be zapped too if we want to swap all the guest pages
- sptes are like a secondary MMU, that creates a separate address space built in function of the primary MMU ptes
- so for Linux to unmap sptes, Linux must become capable of invalidating all secondary MMUs mappings
- That's what MMU Notifiers are all about

VM layout with mmu notifiers without spte page pinning



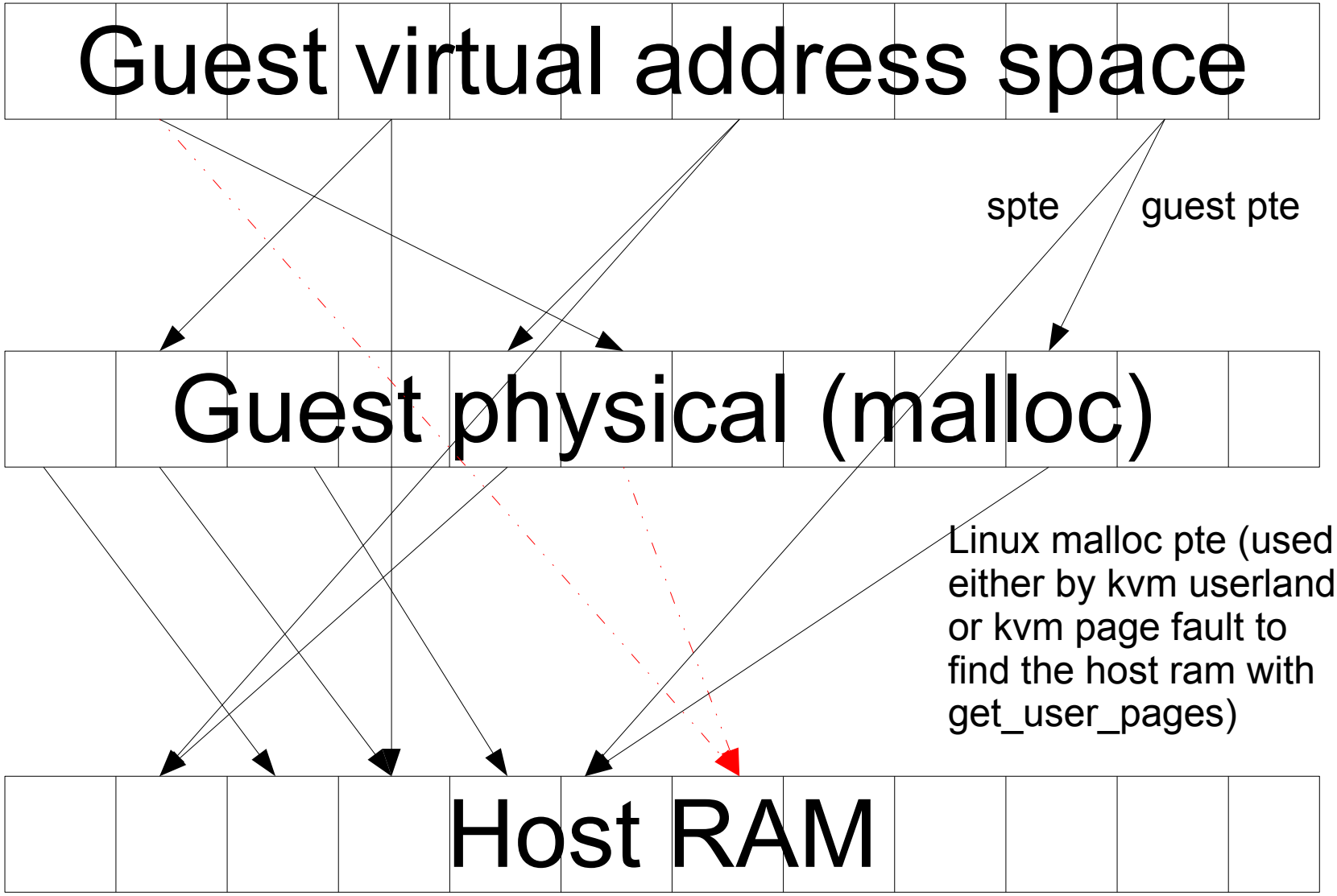
For simplicity this doesn't include the temporary VM and swapcache references

try_to_unmap_one() calls ptep_clear_flush_notify() ptezap



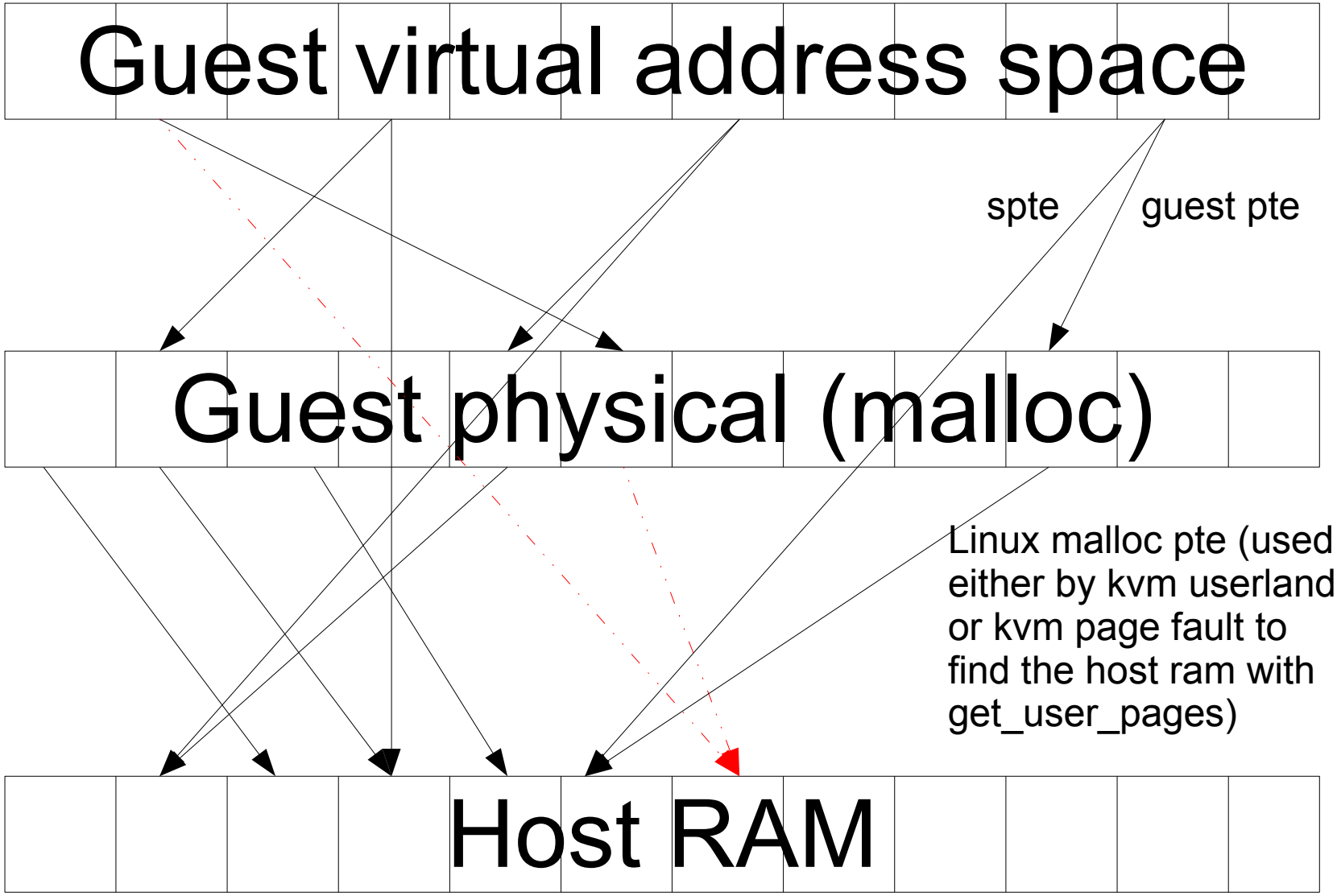
For simplicity this doesn't include the temporary VM and swapcache references

ptep_clear_flush_notify() calls ->invalidate_page() sptezap



For simplicity this doesn't include the temporary VM and swapcache references

try_to_unmap_one() unmap the page and the VM swap it out



For simplicity this doesn't include the temporary VM and swapcache references

MMU Notifier v18 API

```
void (*release)(struct mmu_notifier *mn,
                struct mm_struct *mm);
int (*clear_flush_young)(struct mmu_notifier *mn,
                         struct mm_struct *mm,
                         unsigned long address);
void (*invalidate_page)(struct mmu_notifier *mn,
                        struct mm_struct *mm,
                        unsigned long address);
void (*invalidate_range_start)(struct mmu_notifier *mn,
                               struct mm_struct *mm,
                               unsigned long start,
                               unsigned long end);
void (*invalidate_range_end)(struct mmu_notifier *mn,
                             struct mm_struct *mm,
                             unsigned long start,
                             unsigned long end);
```

Secondary MMU invalidate_page

1) pte_clear (primary MMU)
2) tlb_flush (primary MMU)
2) mmu_notifier_invalidate_page (secondary MMU)
   2.1) invalidate sptes
   2.2) invalidate secondary TLB
   2.3) restart any in progress secondary page fault
3) put_page()

Secondary MMU invalidate_range

1) mmu_notifier_range_start()
   1.1) block secondary MMU page fault
   1.2) invalidate secondary MMU

2) tlb_gather (primary MMU)

3) mmu_notifier_range_end()
   3.1) restart any in progress kvm page fault
   3.2) unblock any in progress kvm page fault

MMU Notifier and scheduling

- XPMEM wants to schedule inside the methods
- By solving the race without depending on PT lock later it will be possible to schedule inside the MMU Notifier methods by:
  - changing the anon_vma->lock and i_mmap_lock from spinlock to rwsem
  - replacing rcu to srcu
- invalidate_range_start/end by blocking the secondary MMU page fault inside the _start/_end critical section, can be called outside the tlb_gather kind of loops
  - _range_start before taking the PT lock
  - _range_end after releasing the PT lock

Lock constraints

- The major race condition to take care of is the invalidate of the sptes against the secondary mmu page fault
- ->invalidate_page is called after clearing the linux pte and before freeing the page, it has to teardown the sptes internally, and any parallel kvm page fault is not allowed to establish sptes if invalidate_page has run after get_user_pages
- ->invalidate_range_start must teardown the sptes in the host virtual range and prevent any secondary MMU page fault to establish sptes until invalidate_range_end is called, and get_user_pages must be repeated then

Host kernel patch example

```
+       mmu_notifier_invalidate_range_start(mm, start, start + size);
        err = populate_range(mm, vma, start, size, pgoff);
+       mmu_notifier_invalidate_range_end(mm, start, start + size);
```

- The invalidate_range_start call prevents any further spte establishment and at the same time invalidates all sptes before populate_range goes ahead to rewrite the linux ptes and free the pages pointed by the old pte values (so the freed pages are guaranteed to have no sptes mapping them)
- After all page freeing and pte mangling is complete, invalidate_range_end unblocks the kvm page fault and ensures to reply all in-flight get_user_pages, so sptes are guaranteed to see the new linux pte values written by populate_range

KVM MMU Notifier invalidates

- The invalidate operation done by the KVM methods implementing the MMU Notifier invalidate_page and invalidate_range_start ops will:
  - Find the gfn to invalidated by seraching memslots where the host virtual address or address range fits
  - Find the rmap structure that maps a certain gfn to all sptes possibly mapping it
  - Mark all sptes invalid and remove them from the rmap structure

invalidate vs page fault locking

- This is achieved ***readonly*** for page fault fast path:
- An atomic counter is increased in invalidate_range_start and is decreased in invalidate_range_end
- A sequence number is incremented before returning from invalidate_page, and before decreasing the atomic counter in invalidate_range_end
- After get_user_pages returns, the kvm page fault takes the kvm->mmu_lock (that serializes all spte and spte-rmap operations) and proceeds to establish the spte only if the sequence number is the same as before calling get_user_pages and if the counter is zero

MMU Notifier not just for swap

- The MMU Notifier is needed for more than secure and reliable swapping:
  - Ballooning needs madvise to unmap sptes too in a way that ensures coherency between userland and guest address space
  - KSM must invalidate sptes when do_wp_page fires on a PageKSM shared page (otherwise the wrprotected sptes would still point to the oldpage)
    - This is lucky and can be simulated with kprobes
  - Removing the need of the page pinning allows the refcounting to be the same as the one of the sptes mapping mmio regions with pci-passthrough

MMU Notifier not just for swap

- The MMU Notifier is needed for more than secure and reliable swapping:
  - Last but not the least it avoids to replicate the smp-host safe tlb_gather logic inside KVM by guaranteeing that the VM holds reference on the pages mapped by the sptes at all times
  - Thanks to the page pin dependency removal, KVM in turn will never put guest-mapped pages in the freelist itself, and it can flush the tlb once (just before releasing the mmu_lock that is taken by the kvm lowlevel mmu notifier methods too) no matter how many sptes it zapped

Q/A

- You're very welcome!