# Killing the mmap_sem's contention

# VMA Locking

Laurent DUFOUR  - IBM

Jérôme Glisse – Red Hat

# Why?

- Large number of CPUs

- mmap_sem
- LRU lock

- Massively threaded applications

- Too much usage of the mmap_sem

- Bottlenecks

- Another big kernel lock

# Process’s Virtual Memory

- Per process MM descriptor (mm_struct)
- Most of the fields of the mm_struct are protected using the mm.mmap_sem
- VMA defines a memory VMM memory area
  - Ordered double linked list (mm.mmap)
  - Augmented RB tree (mm.mm_rb)
    - Allows quick find of a gap (based on size and start node)
- Page table entries (pgd/pud/pmd/pte)
  - Protected by mmap_sem (root level) and split pmd locks mechanism.

# VMA’s access

- All manipulations are protected through the mmap_sem
- A writer prevents readers
- A reader prevents writers
- Special case, VMA’s growing (stack) is done under the protection of the page_table_lock and the mmap_sem in read mode.
  - commit 4128997b5f0e ("mm: protect against concurrent vma expansion")
- Sometimes, release the mmap_sem, do stuff, take the mmap_sem back and revalidate the VMA (like in `collapse_huge_page()`)
- Sometimes, downgrade the mmap_sem to read mode to relax the contention

# VMA range Locking

- Needs to be done based on the VMA’s boundaries
  - Merging of neighbors VMA
  - Splitting of a VMA
  - VMA’s growing up or down
- Put the VMA’s range lock within the VMA’s data

# VMA’s locking rules

- To prevent dead lock, area must be locked from the lowest to highest (by convention)
- If 2 areas must be locked, the lowest must be locked first, the highest may have to be unlocked for this
  - Drawback : need to revalidate the highest VMA
  - Only mremap() is concerned

# VMA’s locking rules cont.

- Locking must be done at VMA's boundaries because locking a part of a VMA doesn't prevent that VMA to be split or merged.
  - the VMA may hold its own lock.
- The locked area may covers multiple VMAs
  - the lock must be attached to the VMA
- The locked area may cover part between 2 VMAs
  - the lock may cover space between 2 VMAs
- The locked area may be before or after an existing VMA. We must prevent that VMA to grow over our locked area.
  - the lock area may cover a VMA and an area before and or after a VMA.
- The locked area may not cover an existing VMA
  - a dummy VMA needs to be inserted to hold the lock.

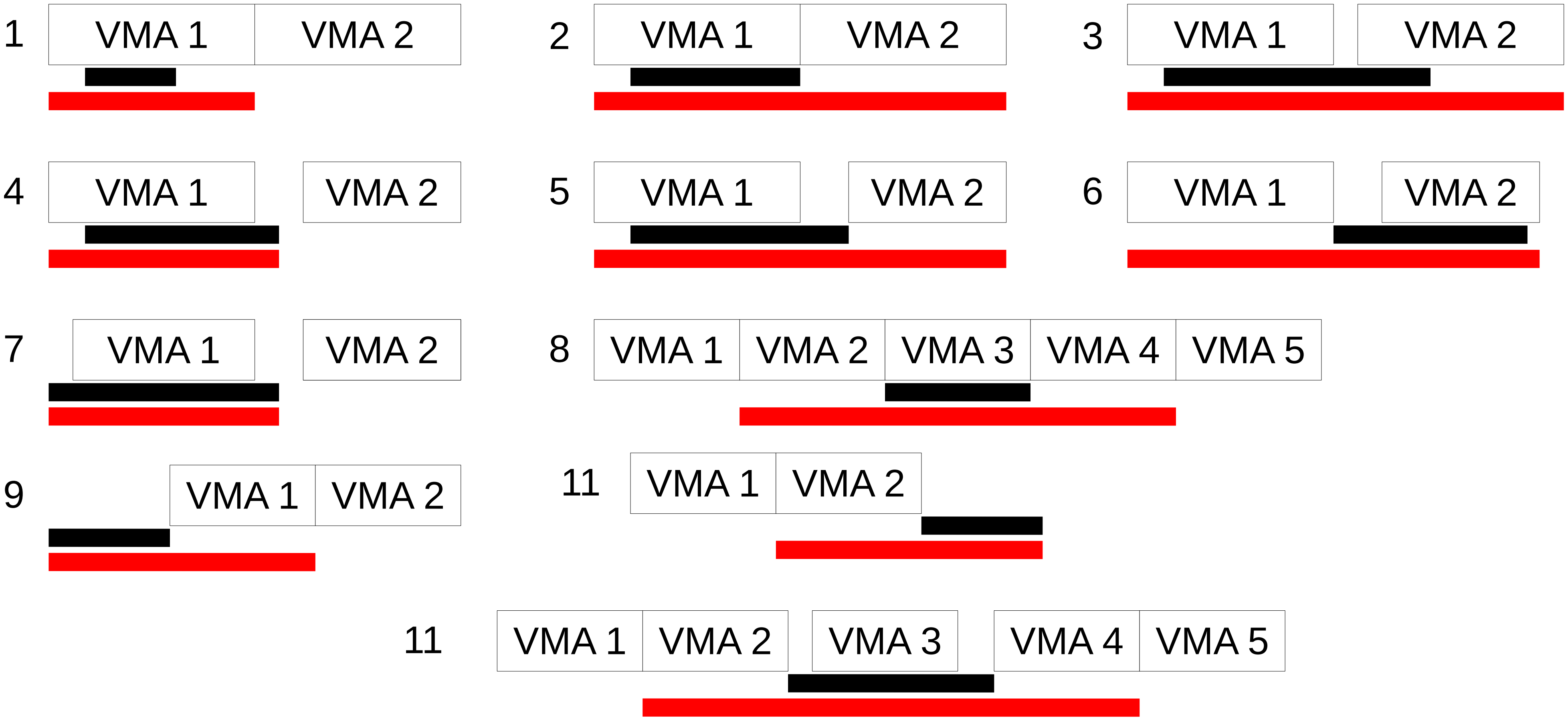
# VMA Lock’s contagion

- Merging a VMA with an adjacent one is a common operation
- When locking an area, the VMAs adjacent to that area must be locked too
- There is no need to extend to the VMAs next or prior to the adjacent one

# The unmap case

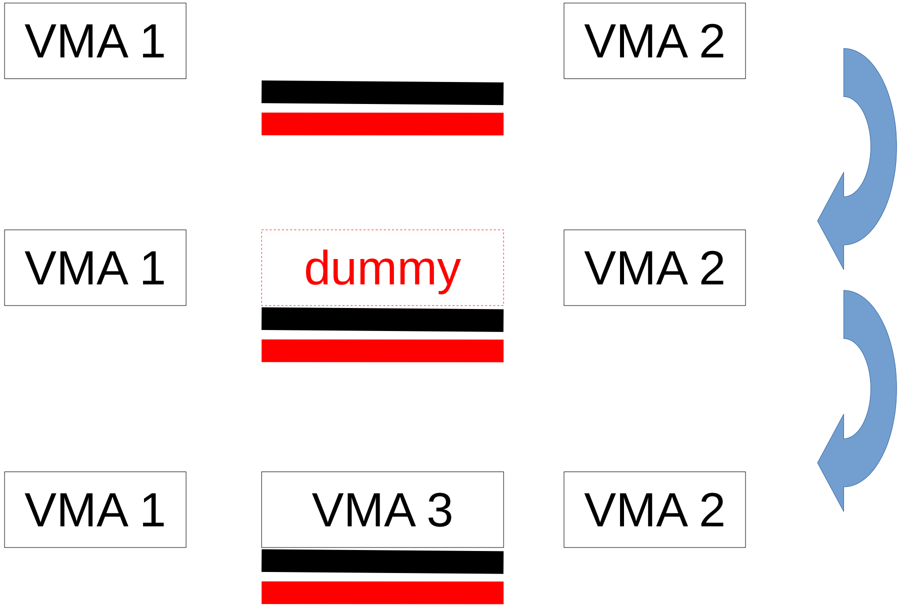
- The area is locked then the VMAs are detached and the cleanup is done.
- While the cleanup is in progress the area need to remain locked to prevent other threads to map again in this area.
- Need to insert a *dummy* VMA to hold the lock while the operation is in progress.

# VMA Locking cases



# Without an existing VMA

Mapping case

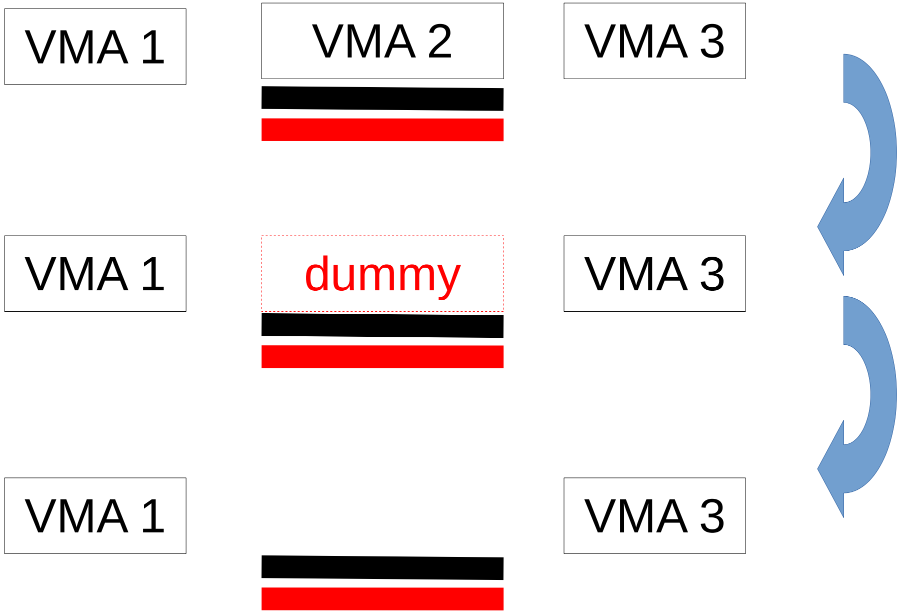


Create a *dummy* VMA to hold the lock.

The *dummy* VMA is converted in a regular VMA, or removed if the operation is aborted.

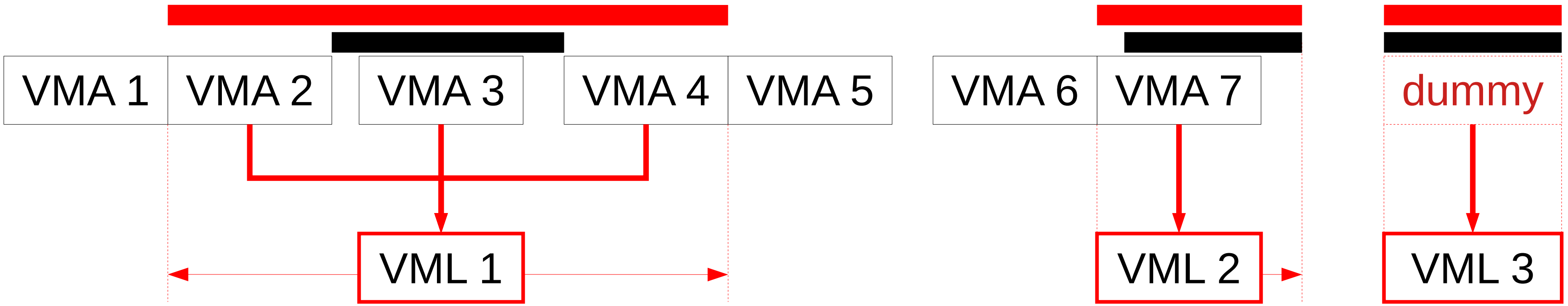
# Without an existing VMA

Unmapping case



Insert a *dummy* VMA to hold the lock.

# VMA locking structure



# Merging and splitting VMAs

- Merge should only happen on locked VMAs using the same vm_area_lock structure.
  - Just need to remove the link in the removed VMA and update the lock’s reference count
- When splitting VMAs, the new VMA is inheriting the lock (reference count ++)

# The get_unmapped_area’s case 1/2

- Take care of the unmapped locked areas
- The *dummy* VMAs are helping here, no need for an additional processing
- Areas before and after adjacent VMAs are easy to access through the lock structure attached to the VMA
  - Similar to the VMA’s gap
- No changes needed to the existing augmented RB tree’s data structure

# The get_unmapped_area’s case 2/2

- get_unmapped_area() should not fail if there is enough locked unmapped area
- Record the best unmapped but locked area if none is unlocked and wait for it to be released
  - While waiting for this area to be released, the area may have been mapped by the thread owning the lock.
    - A retry is needed in that case
    - Not an usual case, meaning concurrent thread’s access to the same area
- Returned area is locked

# Hazards without the mmap_sem

- Device driver or filesystem relying implicitly on mmap_sem for internal protection
- Buggy userspace program that works out of pure luck thanks to the mmap_sem
- Kernel core (arch code, huge pages, ...)

# Updating VMA locking part 1

- Keep the mmap_sem as is
- Introduce the new locking mechanism
  - Core mm
  - No concurrency because of the mmap_sem
- Tests are done by deactivating the mmap_sem for specific processes to avoid impacts of device drivers, file system, arch code, huges pages...

# Updating VMA locking part 2

- Convert
  - Arch code
  - File system
  - Device Drivers
  - Huge Pages support
- Then remove the mmap_sem

# Questions?