20 years of Linux Virtual Memory: from simple server workloads to cloud virtualization

Red Hat, Inc.

Andrea Arcangeli <aarcange at redhat.com>

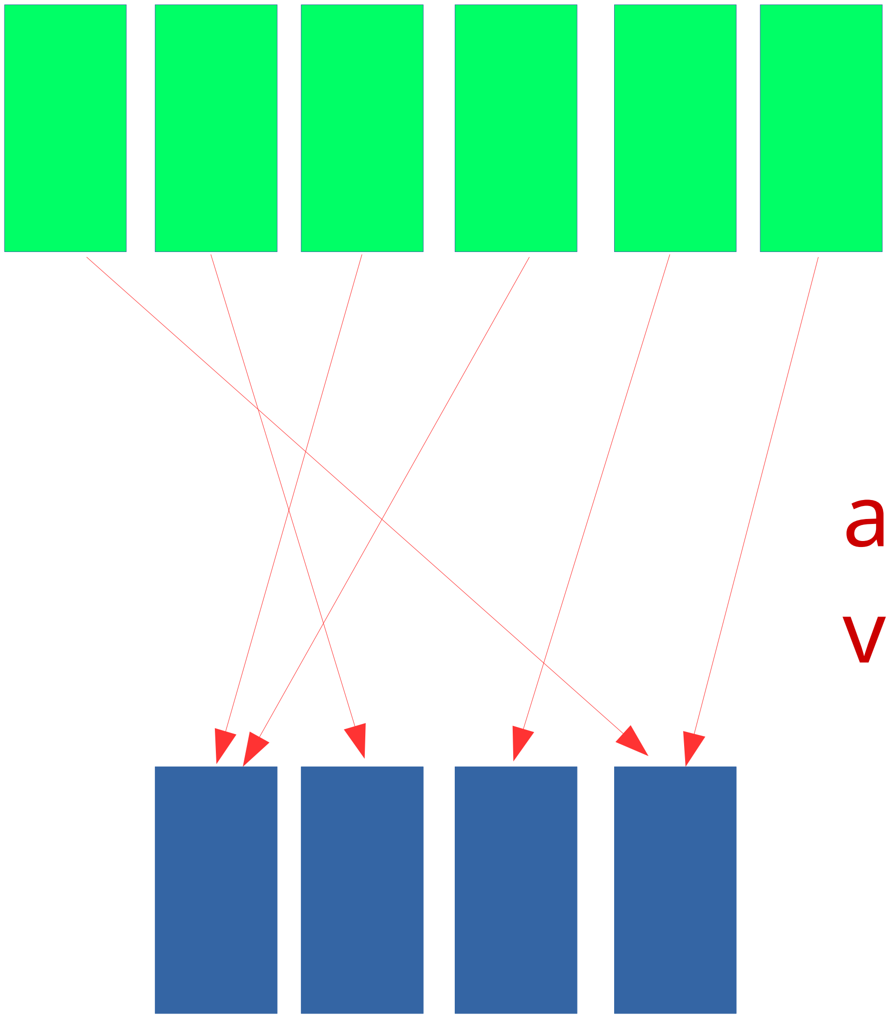
FOSDEM, Brussels

4 Feb 2017

Topics

- Milestones in the evolution of the Virtual Memory subsystem
- Kernel Virtual Machine design
- Virtual Memory latest innovations
  - Automatic NUMA balancing
  - THP developments
  - userfaultfd
    - Postcopy live Migration, etc..

Virtual Memory (simplified)



Virtual pages

They cost “nothing”

Practically unlimited on 64bit archs

arrows = pagetables

virtual to physical mapping

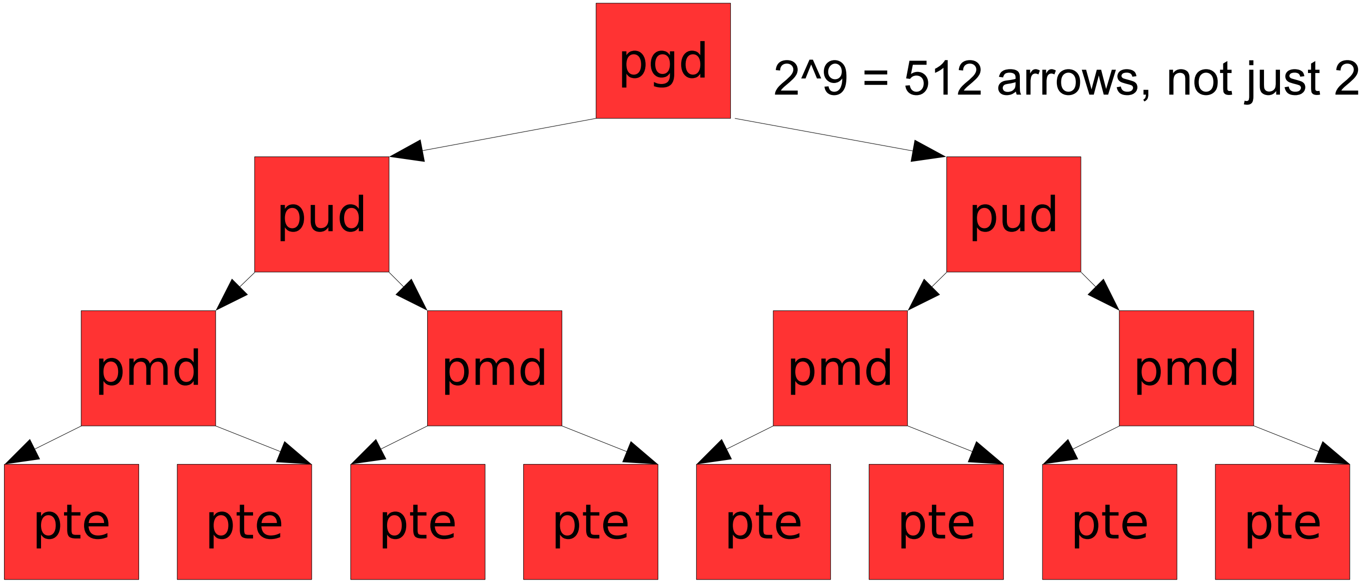
Physical pages

They cost money!

This is the RAM

PageTables

- Common code and x86 pagetable format is a tree

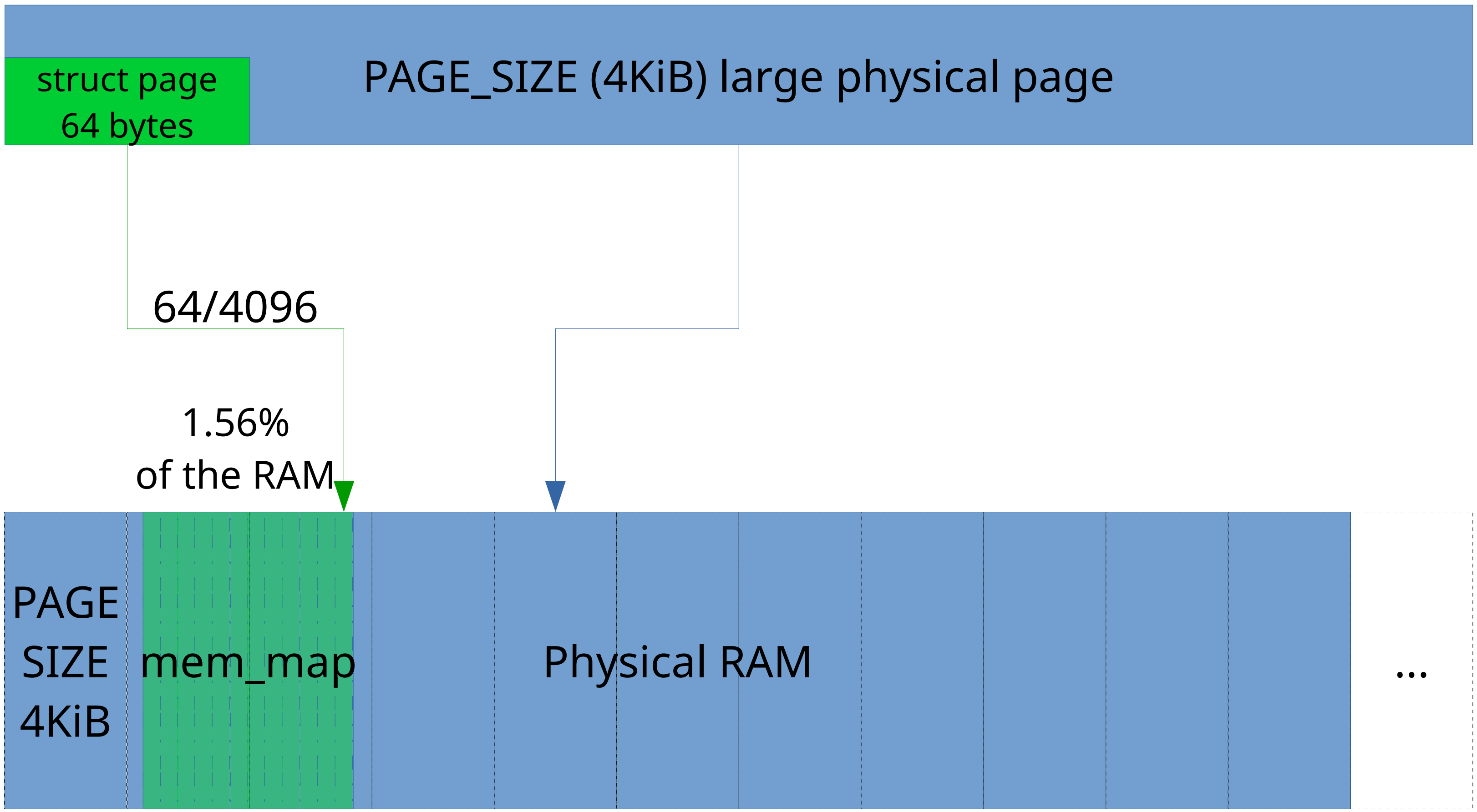


- All pagetables are 4KB in size
- Total: grep PageTables /proc/meminfo
- (((2**9)**4)*4096)>>48 = 1 → 48bits

The Fabric of the Virtual Memory

- The fabric are all those data structures that connects to the hardware constrainted structures like pagetables and that collectively create all the software abstractions we're accustomed to
  - tasks, processes, virtual memory areas, mmap (glibc malloc) ...
- The fabric is the most black and white part of the Virtual Memory
- The algorithms doing the computations on those data structures are the Virtual Memory heuristics
  - They need to solve hard problems with no guaranteed perfect solution
  - i.e. when it's the right time to start to unmap pages (swappiness)
    - Some of the design didn't change: we still measure how hard it is to free memory while we're trying to free it
- All free memory is used as cache and we overcommit by default (not excessively by default)
  - Android uses: `echo 1 >/proc/sys/vm/overcommit_memory`

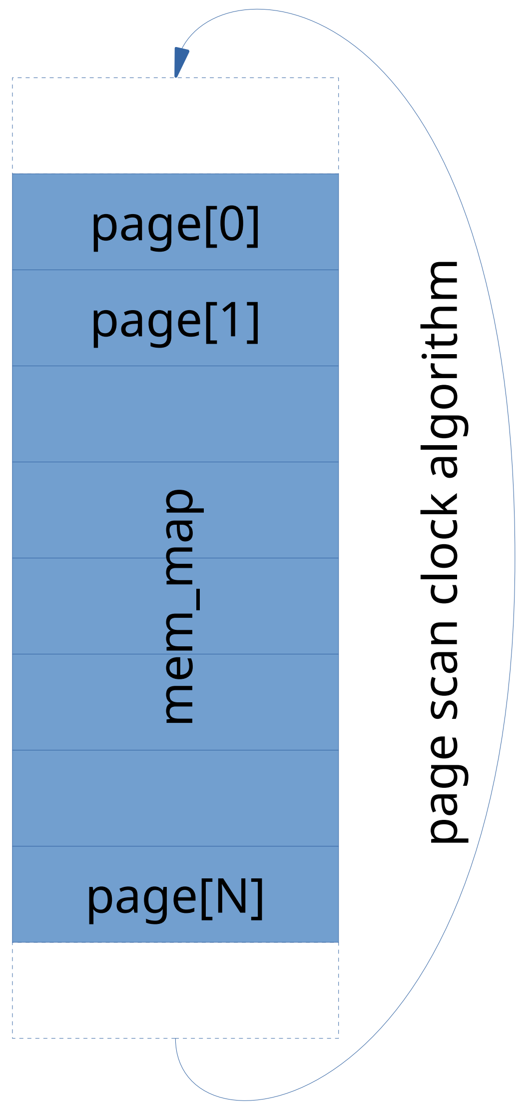
Physical page and struct page



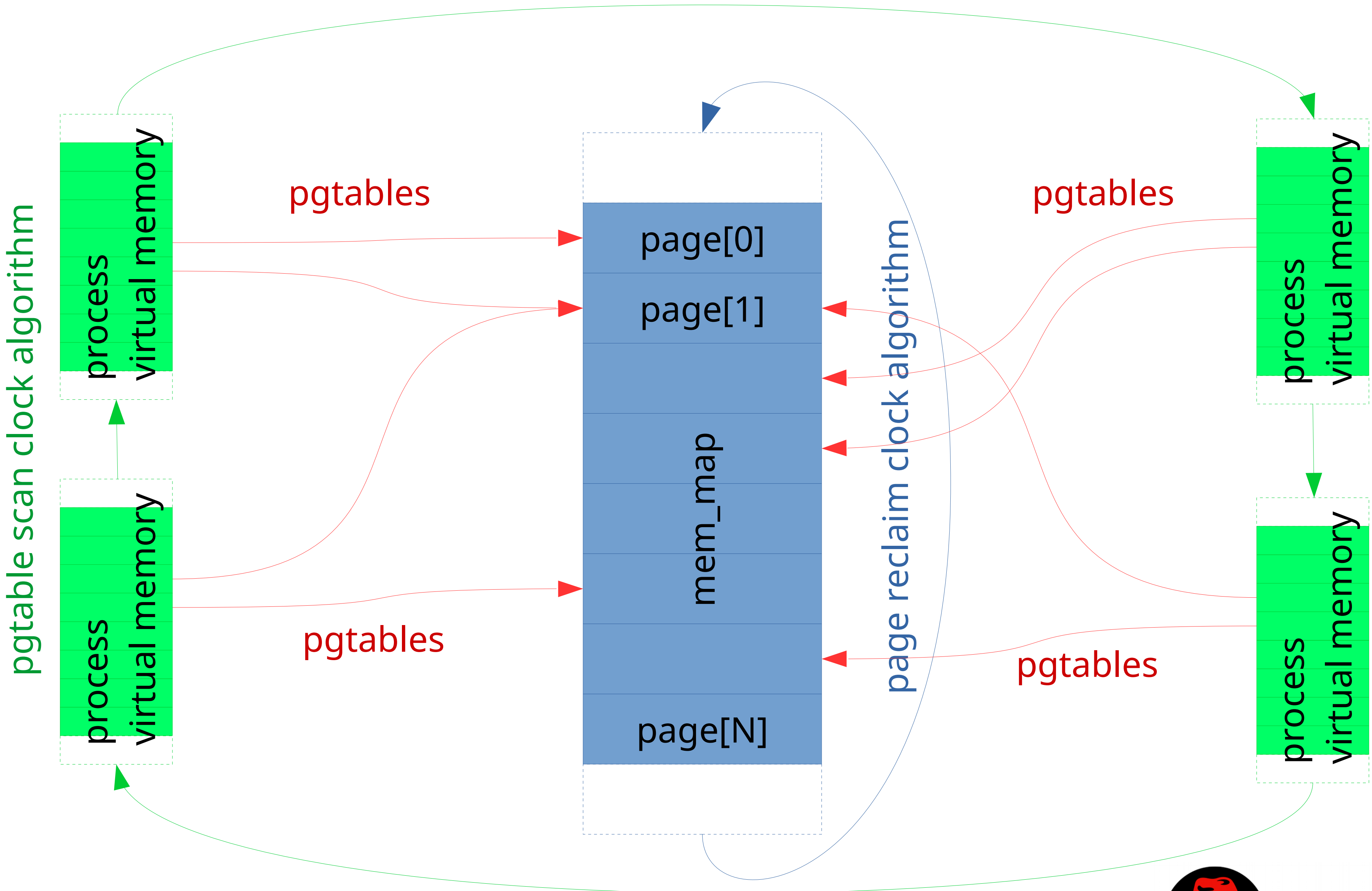
MM & VMA

- mm_struct aka MM
  - Memory of the process
    - Shared by threads
- vm_area_struct aka VMA
  - Virtual Memory Area
    - Created and teardown by mmap and munmap
    - Defines the virtual address space of an "MM"

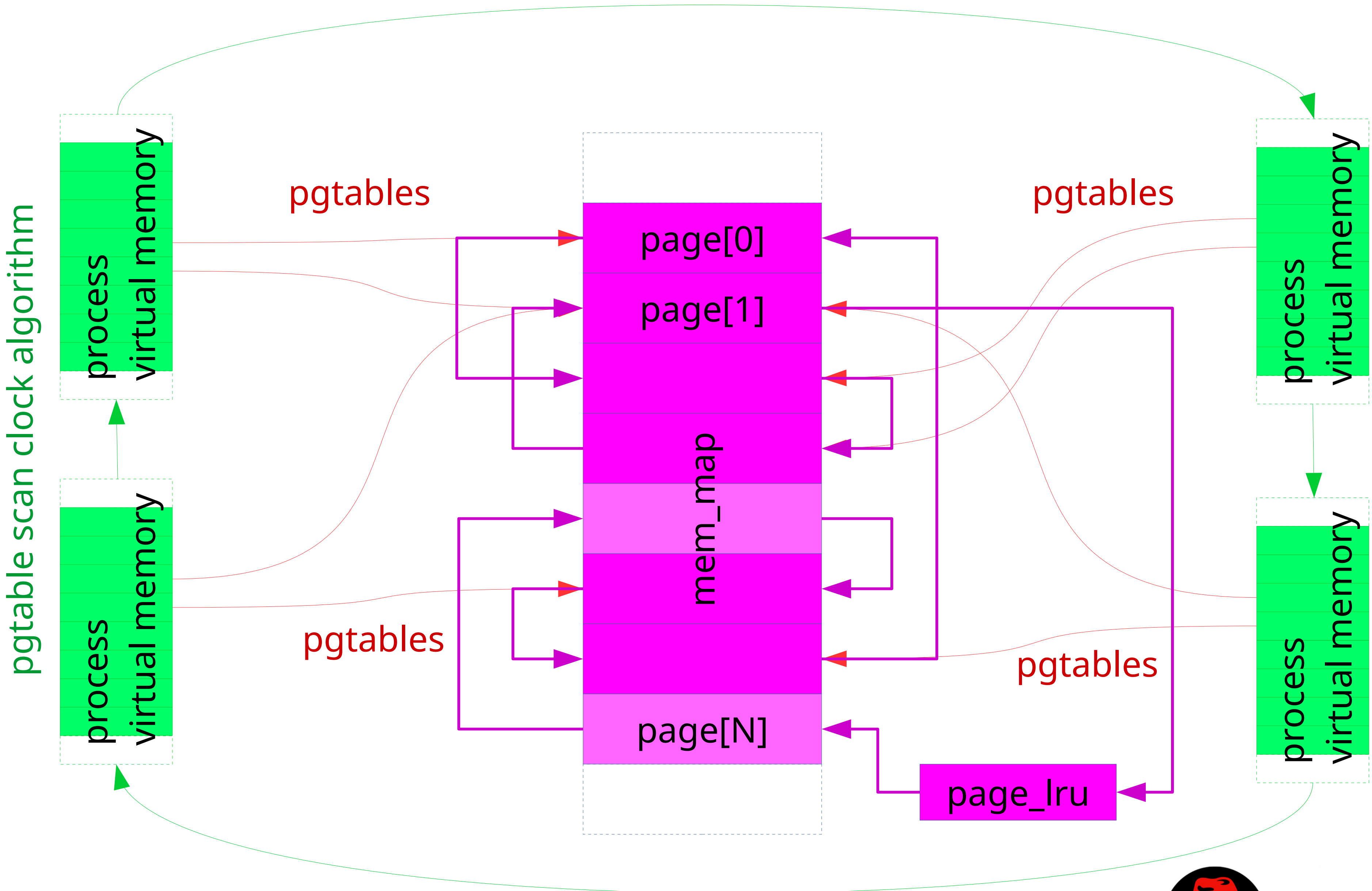
Page reclaim clock algorithm



Pgtable scan clock algorithm

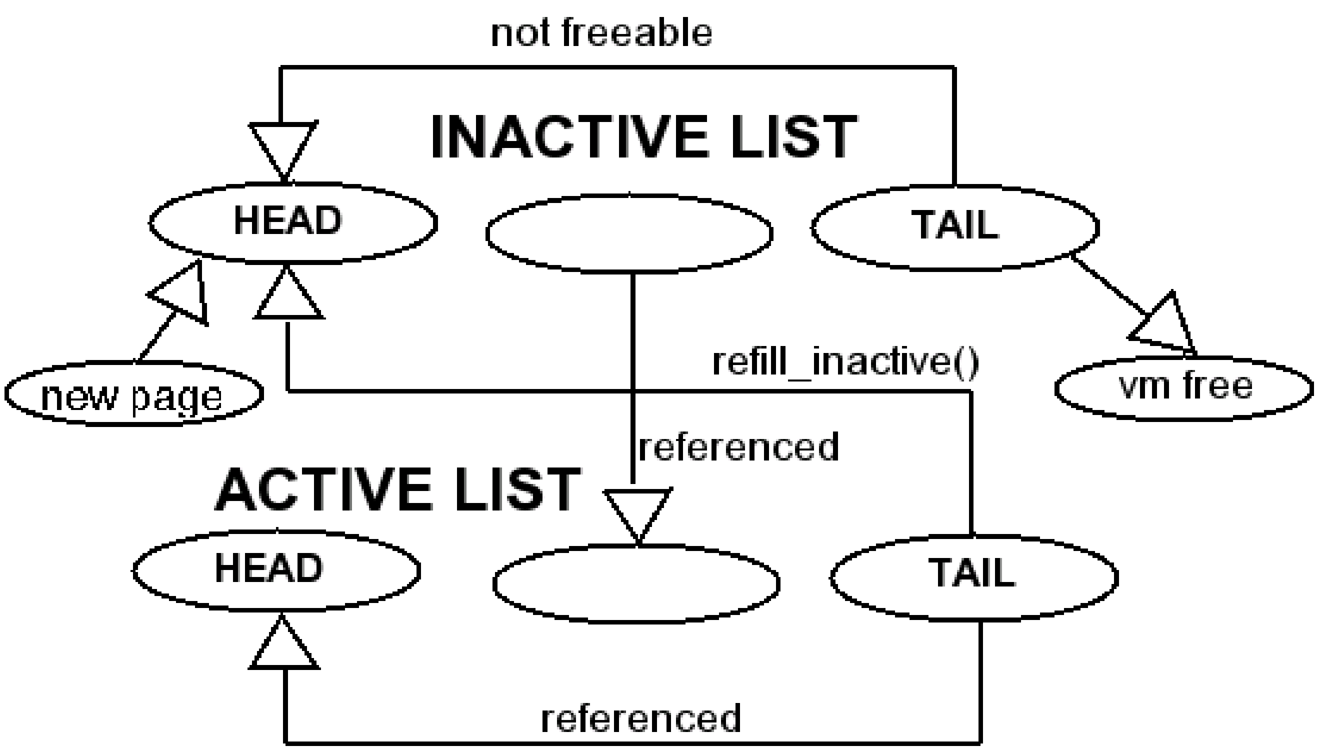


Last Recently Used list



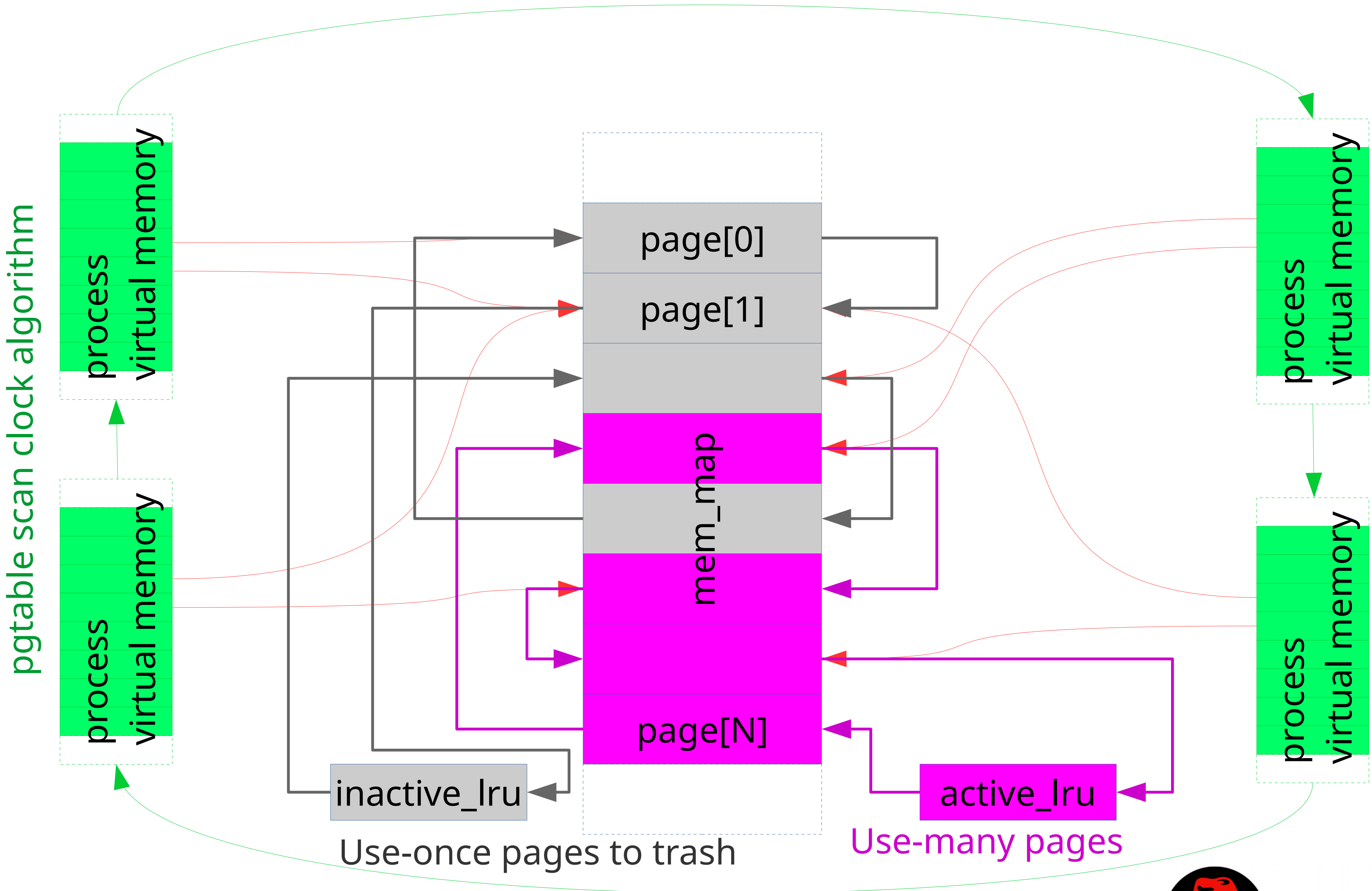
Active and Inactive list LRU

- The active page LRU preserves the the active memory working set
  - only the inactive LRU loses information as fast as use-once I/O goes
  - Introduced in 2001, it works good enough also with an arbitrary balance
  - Active/inactive list optimum balancing algorithm was solved in 2012-2014
    - shadow radix tree nodes that detect refaults (more patches last month)



Original bitmap...

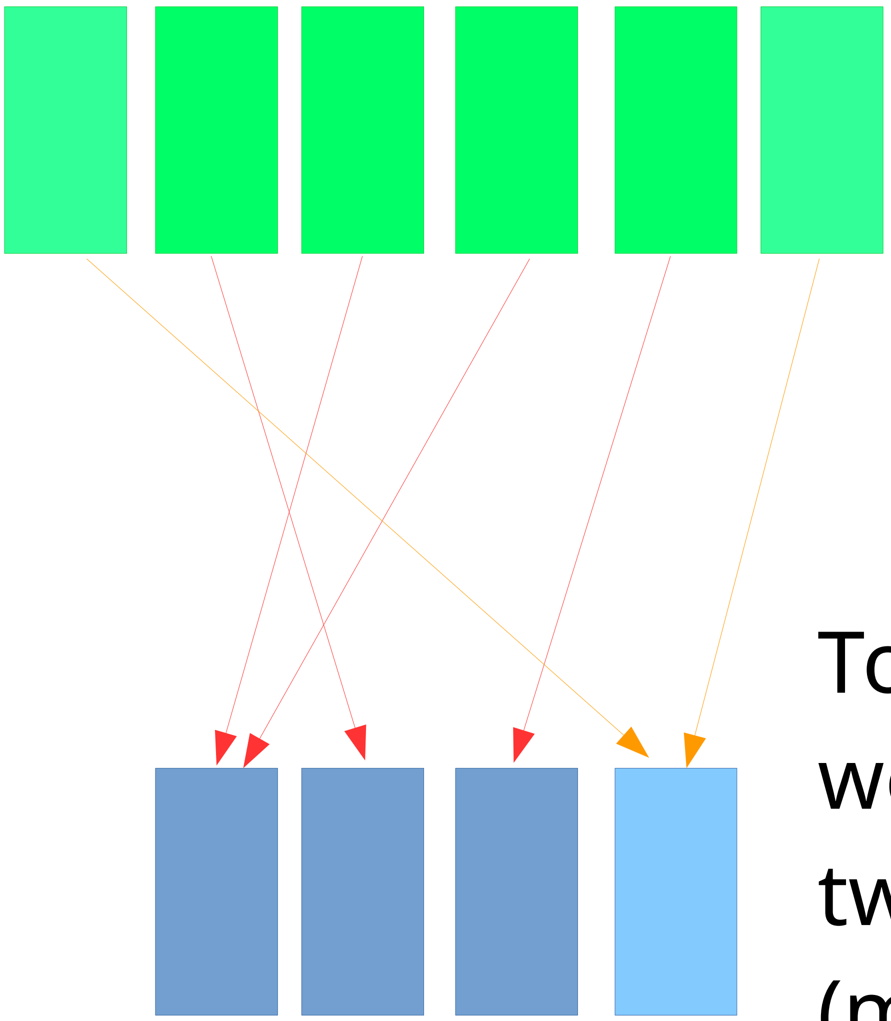
Active & inactive LRU lists



Active and Inactive list LRU

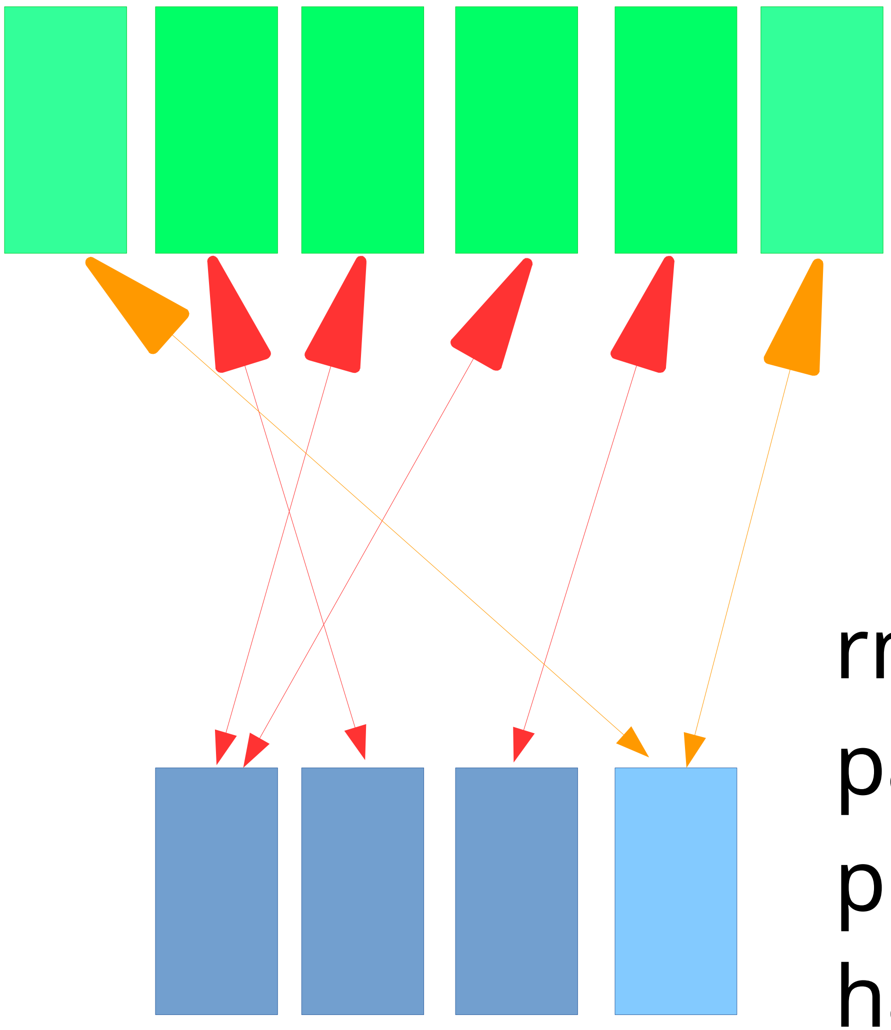
```
$ grep -i active /proc/meminfo
Active:            3555744 kB
Inactive:          2511156 kB
Active(anon):      2286400 kB
Inactive(anon):    1472540 kB
Active(file):      1269344 kB
Inactive(file):    1038616 kB
```

rmap obsoleted the pgtable scan clock algorithm



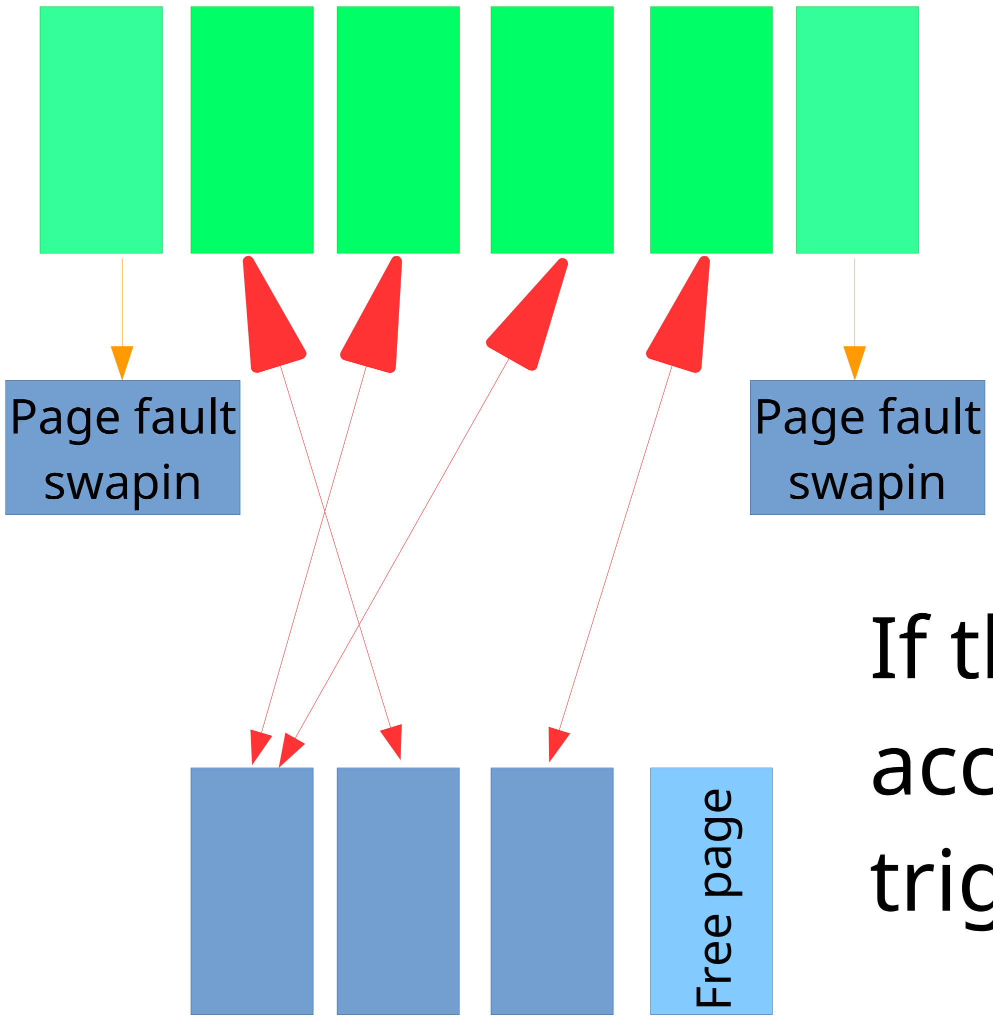
To free the candidate page we must first first drop the two arrows (mark the pte non-present)

rmap as in reverse mapping



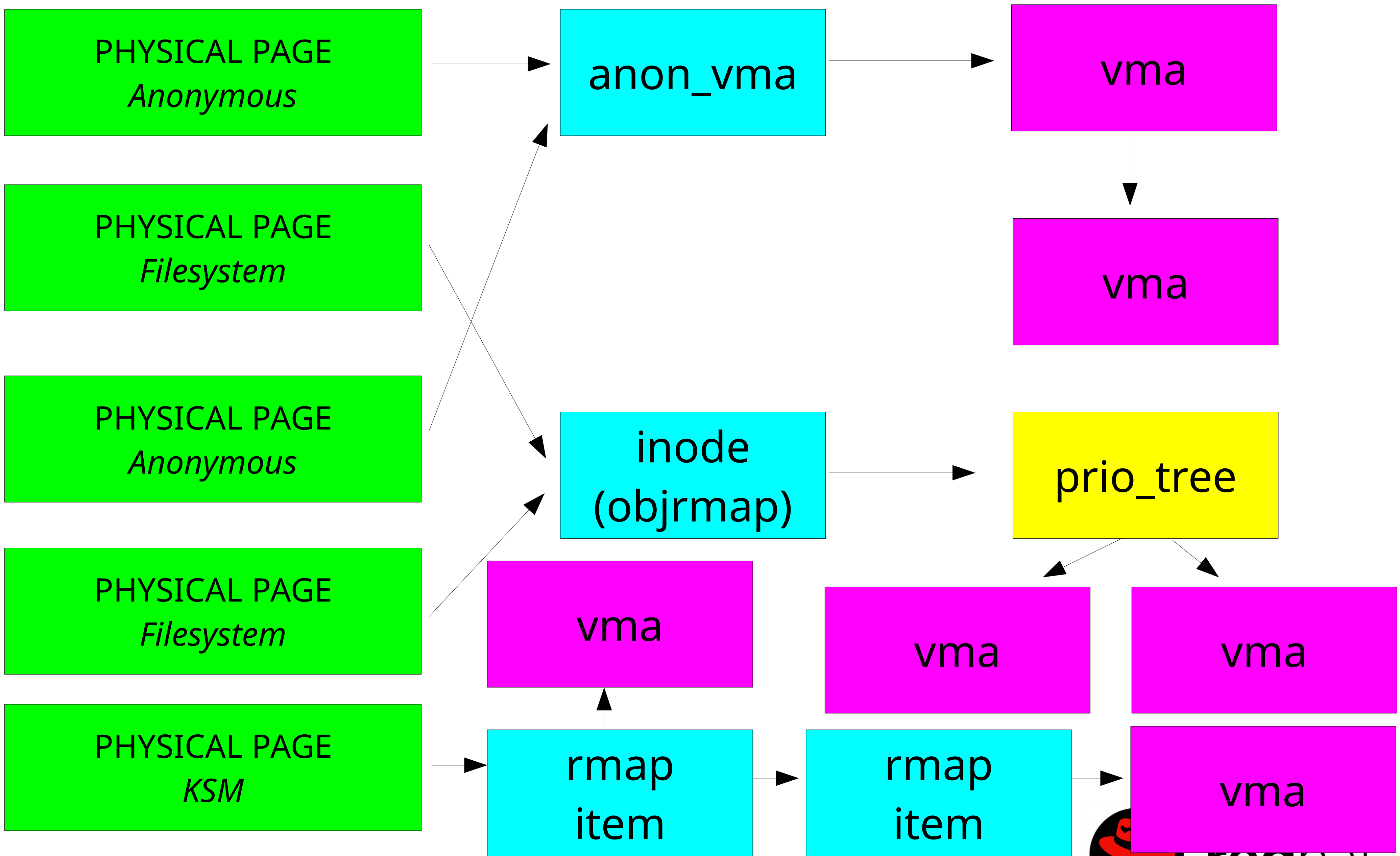
rmap allows to reach the pagetables of any given physical page without having to scan them all

rmap unmap event

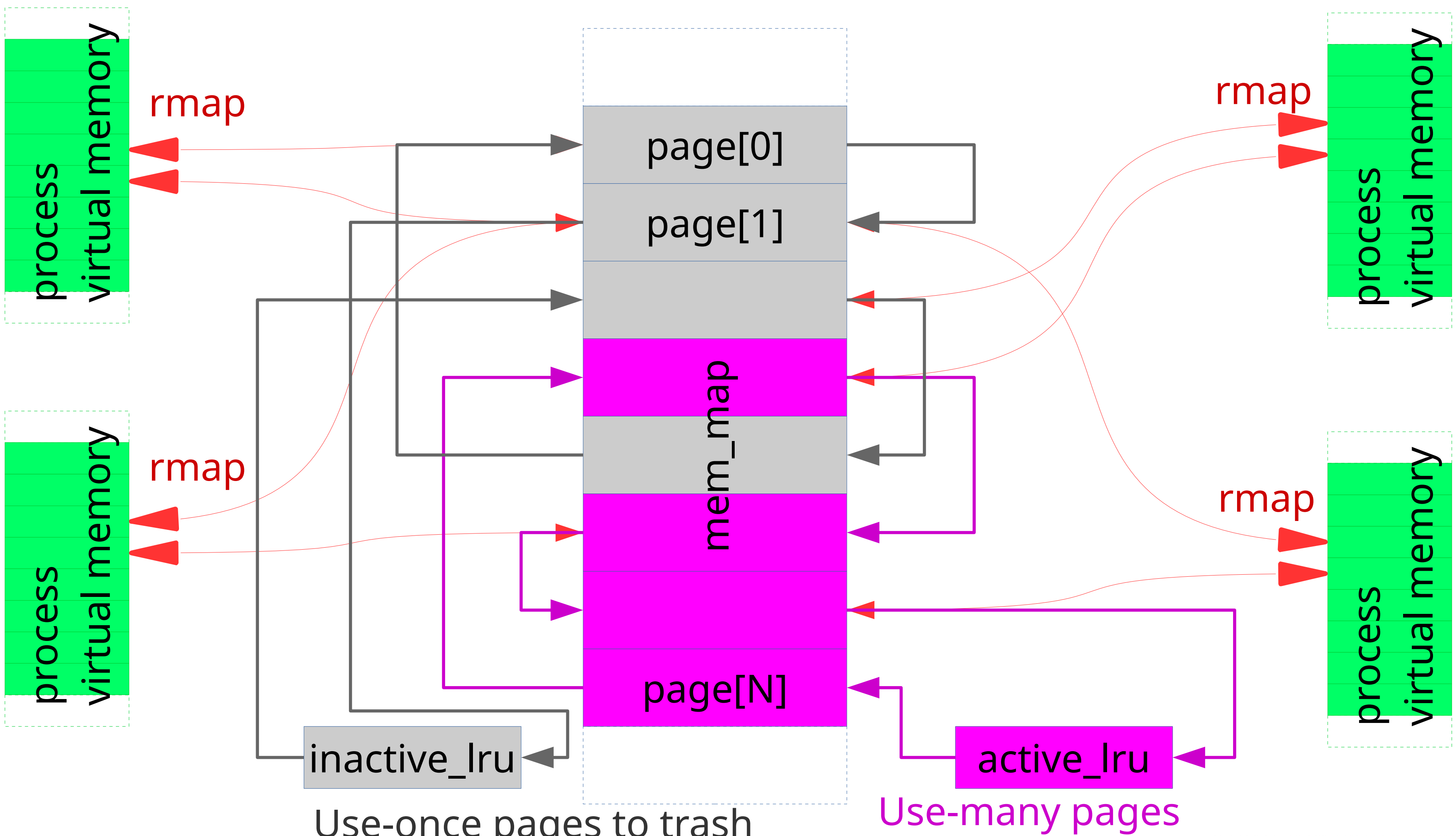


If the userland program accesses the page it will trigger a pagein/swapin

objrmap/anon-vma



Active & inactive + rmap

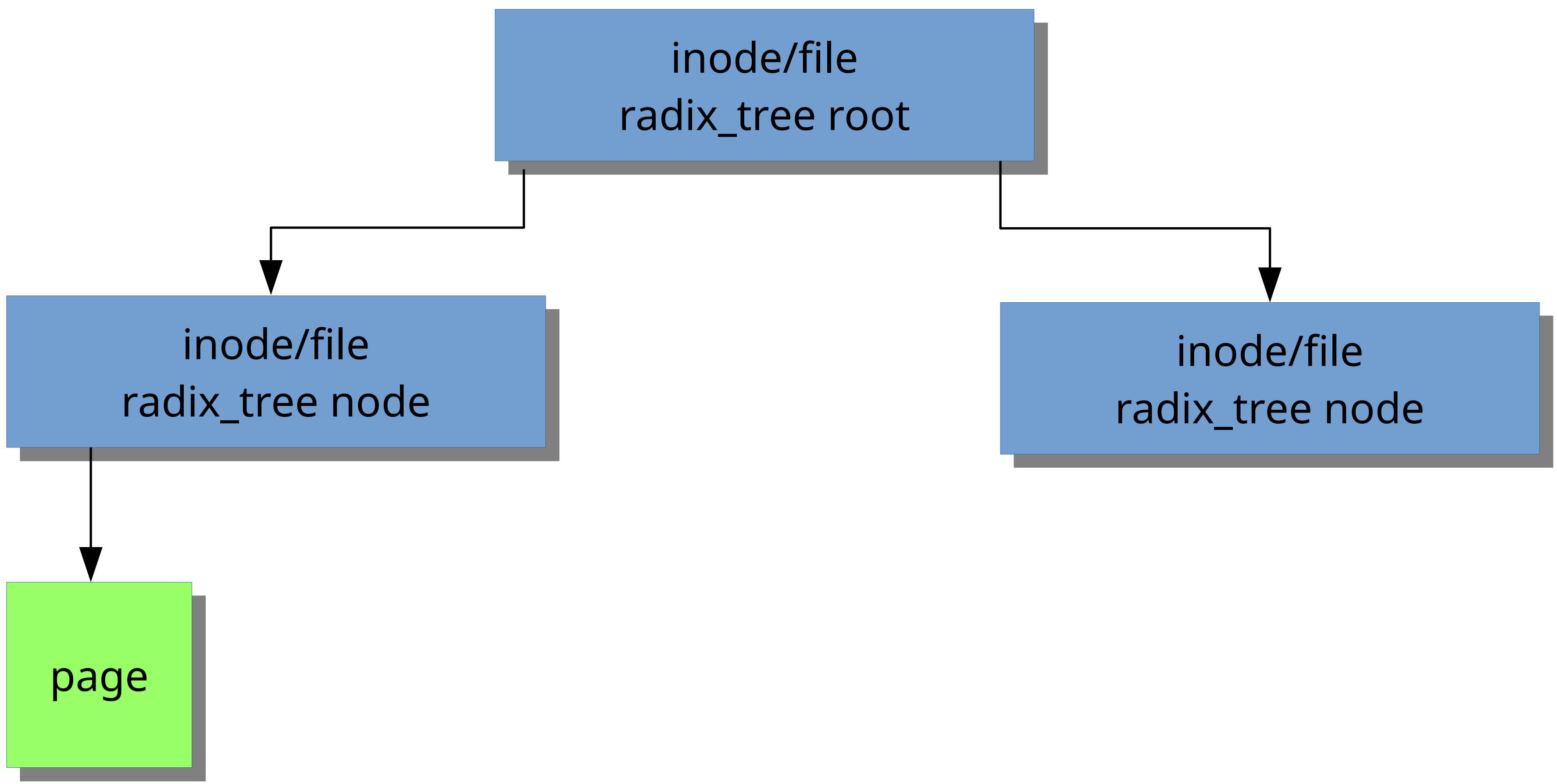


Active LRU workingset detection

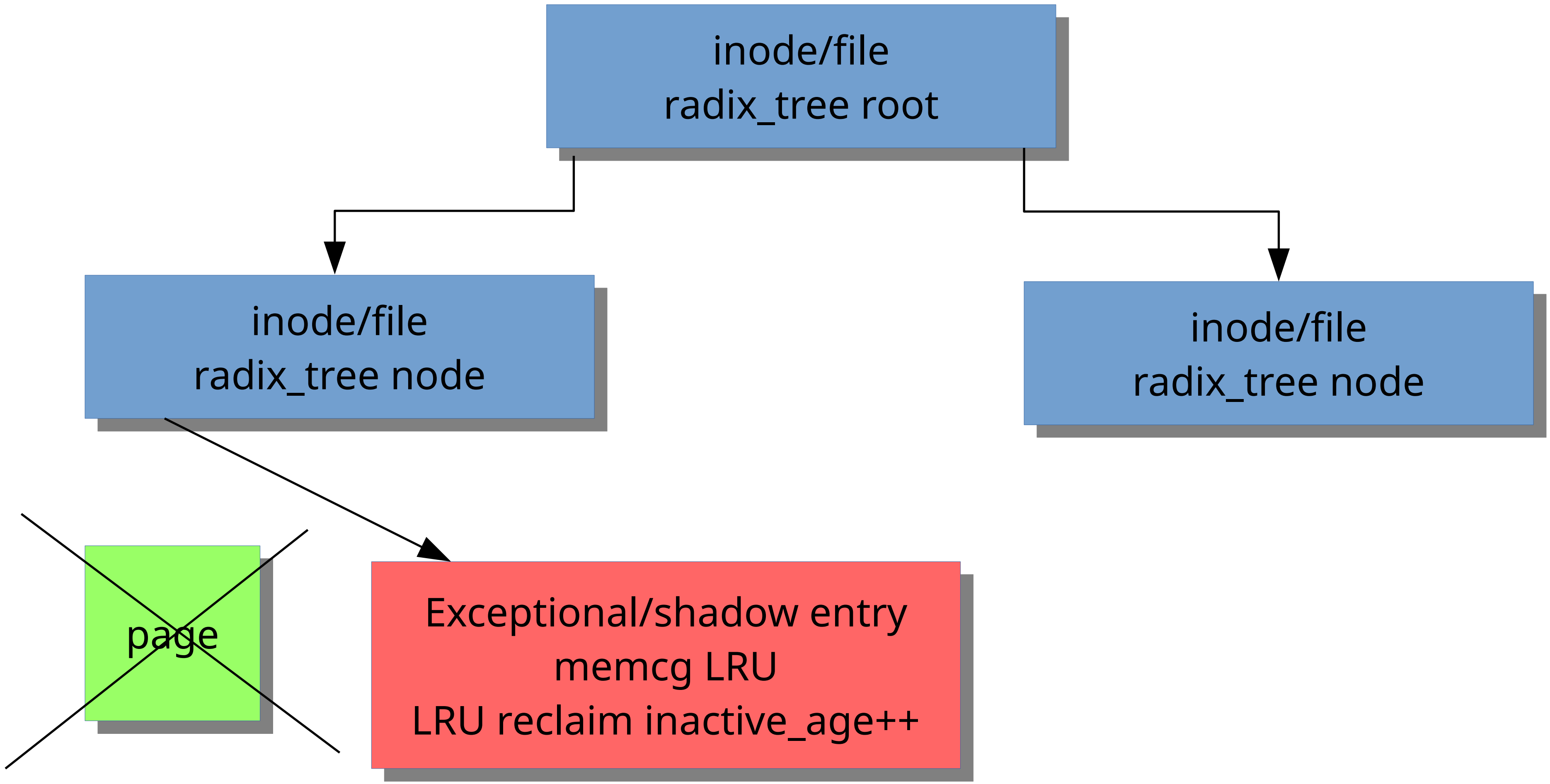
```
fault ------------------------+
                              |
           +--------------+   |             +-------------+
reclaim <- |   inactive   | <-+-- demotion |    active   | <--+
           +--------------+                +-------------+    |
                  |                                           |
                  +-------------- promotion ------------------+
```

```
    +-memory available to cache-+
    |                           |
    +-inactive------+-active----+
a b | c d e f g h i | J K L M N |
    +---------------+-----------+
```

lru→inactive_age and radix tree shadow entries



Reclaim saving inactive_age



Many more LRUs

- Separated LRU for anon and file backed mappings
- Memcg (memory cgroups) introduced per-memcg LRUs
- Removal of unfreeable pages from LRUs
  - anonymous memory with no swap
  - mlocked memory
- Transparent Hugepages in the LRU increase scalability further (lru size decreased 512 times)

Recent Virtual Memory trends

- Optimzing the workloads for you, without manual tuning
  - NUMA hard bindings (numactl) → Automatic NUMA Balancing
  - Hugetlbfs → Transparent Hugepage
    - THP in tmpfs was merged in Kernel v4.8
  - Programs or Virtual Machines duplicating memory → KSM
  - Page pinning (RDMA/KVM shadow MMU) -> MMU notifier
  - Private device memory managed by hand and pinned → HMM/UVM (unified virtual memory) for GPU seamlessly computing in GPU memory
- ***The optimizations can be optionally disabled***

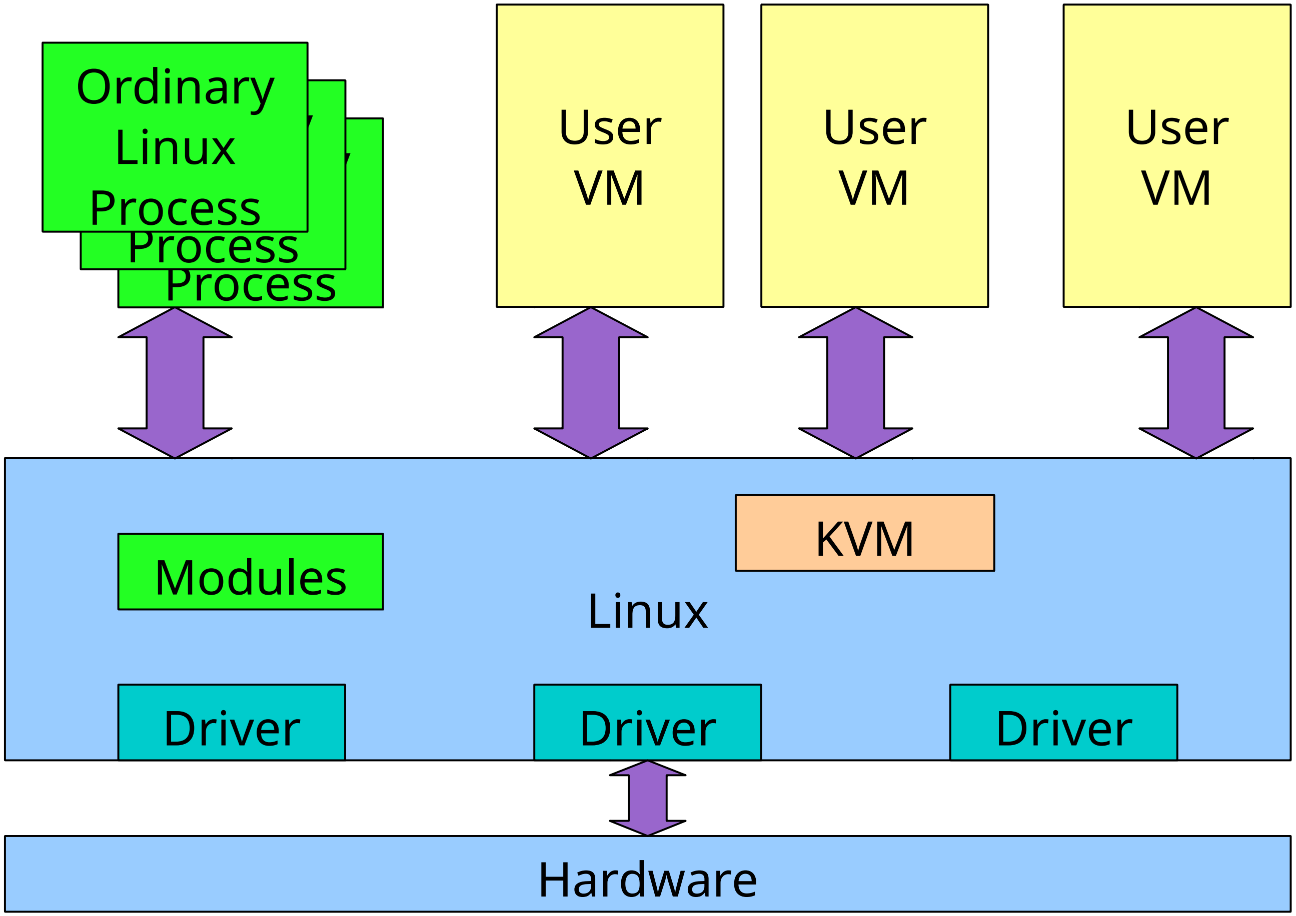
Virtual Memory in hypervisors

- Can we use all these nice features to manage the memory of Virtual Machines?
  - i.e. in hypervisors?
- Why should we reinvent anything?
  - We don't... with KVM

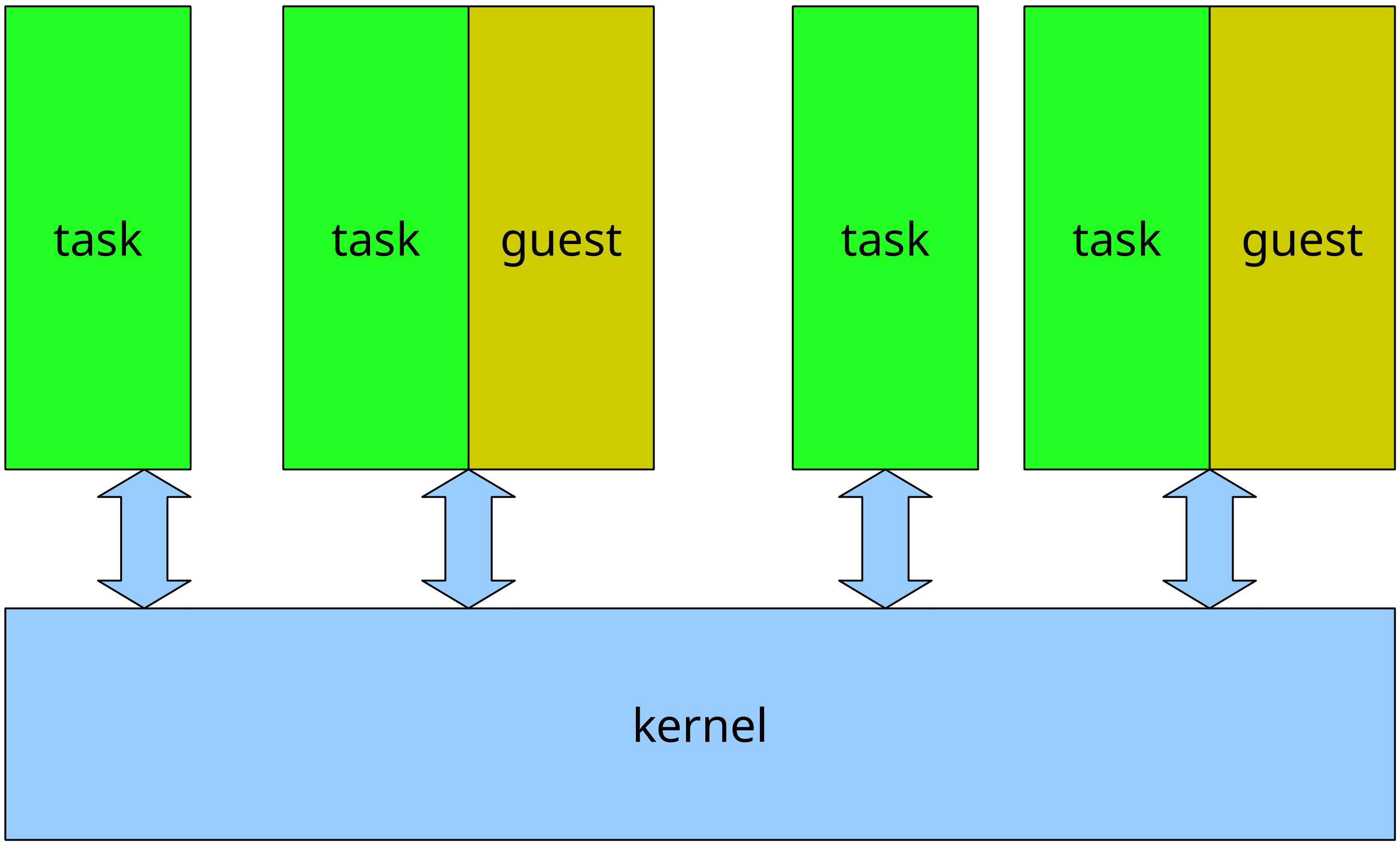
KVM philosophy

- Reuse Linux code as much as possible
- Focus on virtualization only, leave other things to respective developers
  - VM
  - cpu scheduler
  - Drivers
  - Numa
  - Powermanagement
- Integrate well into existing infrastructure
  - just a kernel module + mmu/sched notifier

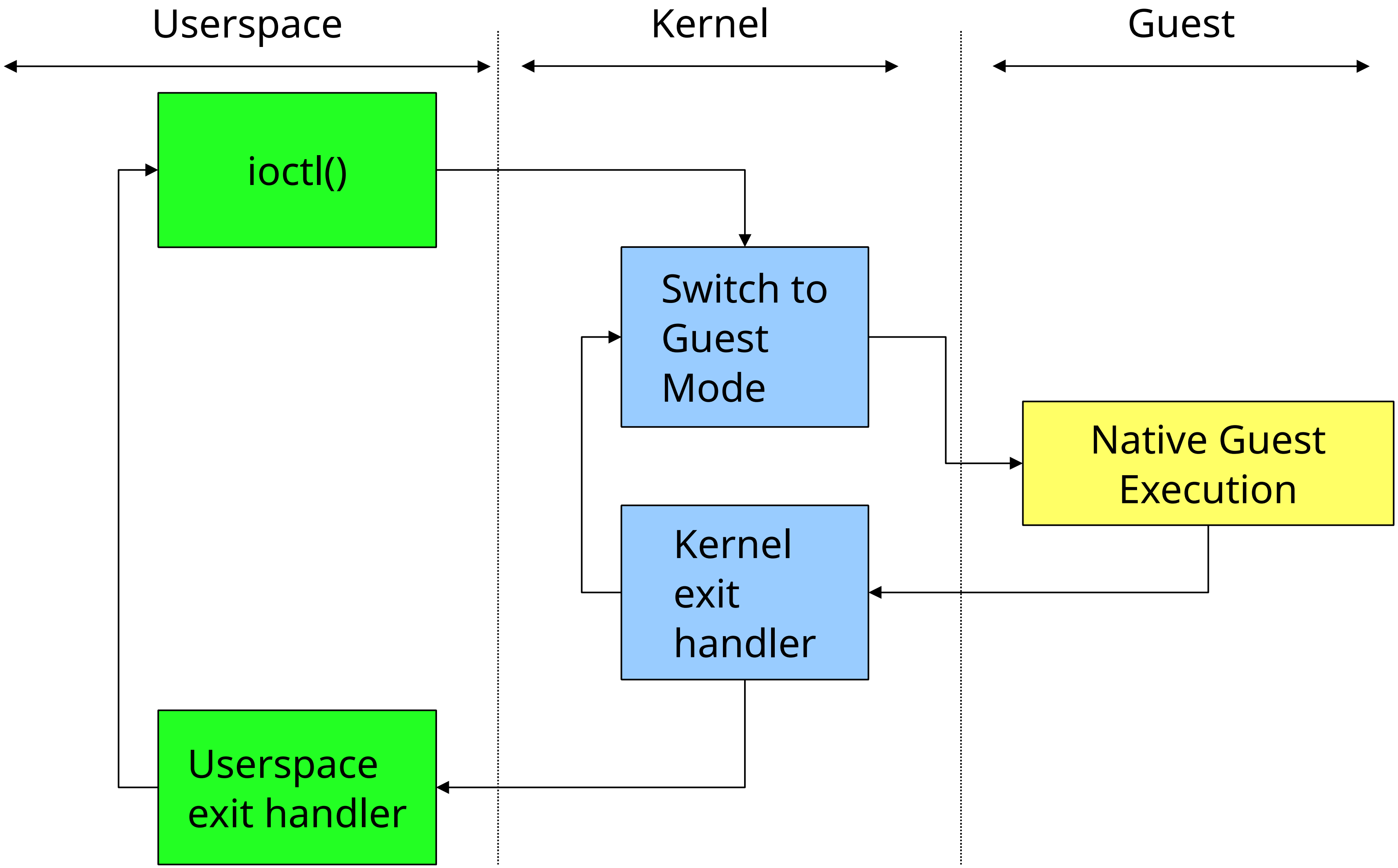
KVM design... way to go!!



KVM task model



KVM userland <-> KVM kernel



Automatic NUMA Balancing benchmark

Intel SandyBridge (Intel(R) Xeon(R) CPU E5-2690 0 @ 2.90GHz)

2 Sockets – 32 Cores with Hyperthreads

256G Memory

RHEV 3.6

Host bare metal – 3.10.0-327.el7 (RHEL7.2)

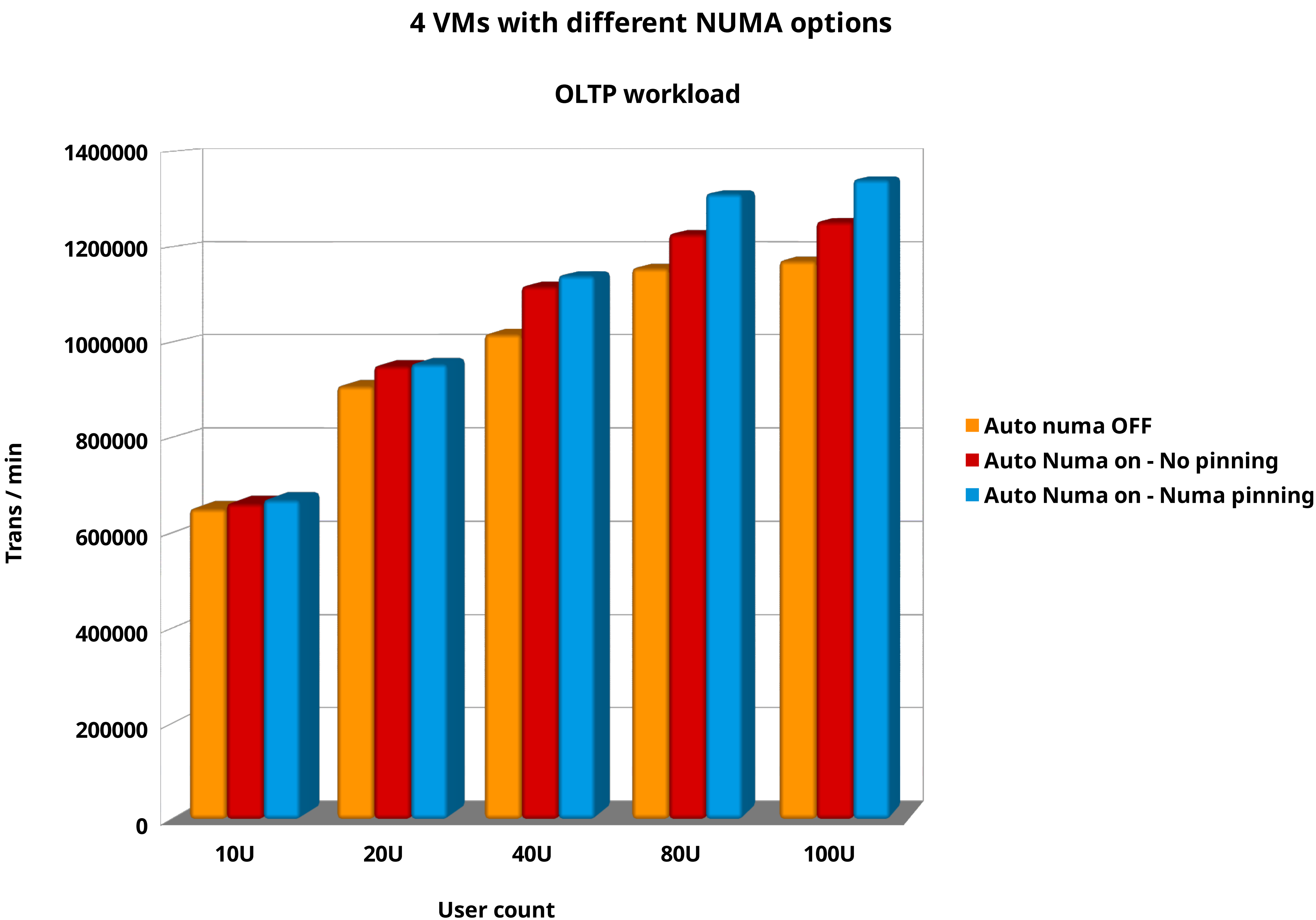
VM guest – 3.10.0-324.el7 (RHEL7.2)

VM – 32P , 160G (Optimized for Server)

Storage – Violin 6616 – 16G Fibre Channel

Oracle – 12C , 128G SGA

Test – Running Oracle OLTP workload with increasing user count and measuring Trans / min for each run as a metric for comparison


4 VMs with different NUMA options
OLTP workload
1400000
1200000
1000000
800000
600000
400000
200000
0
Trans / min
10U
20U
40U
80U
100U
User count
Auto numa OFF
Auto Numa on - No pinning
Auto Numa on - Numa pinning

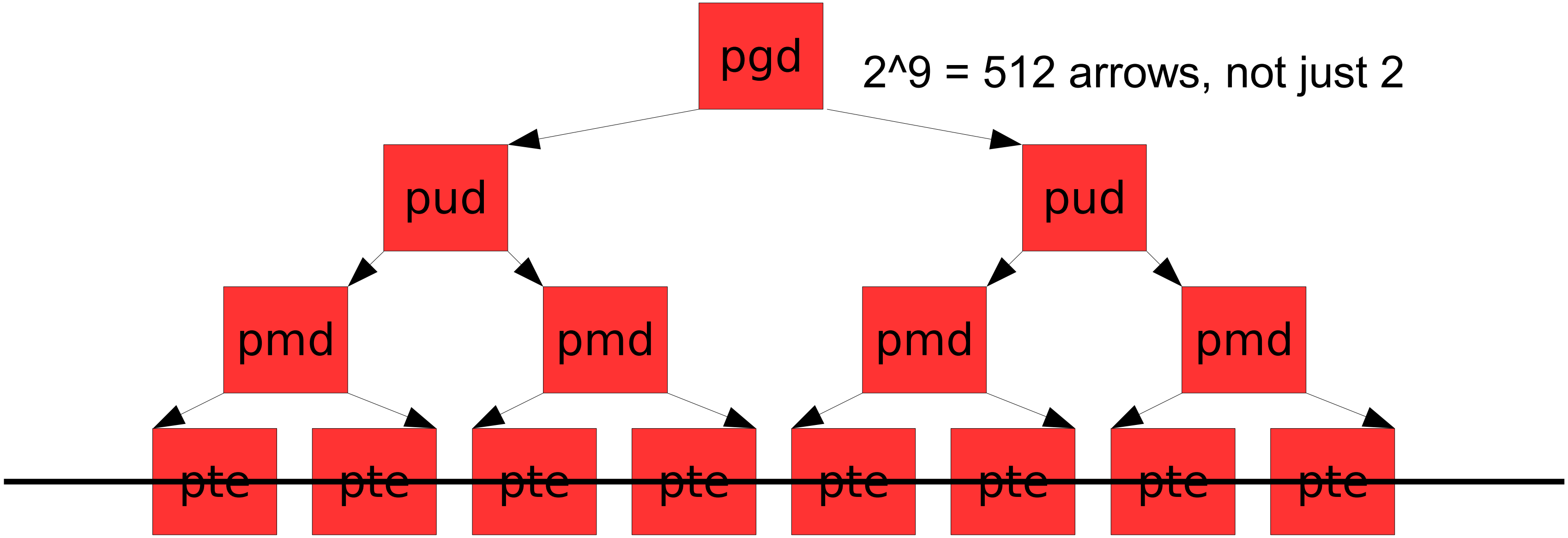
Automatic NUMA balancing configuration

- https://tinyurl.com/zupp9v3
  https://access.redhat.com/
- In RHEL7 Automatic NUMA balancing is enabled when:
  - `# numactl --hardware` *shows multiple nodes*
- To disable automatic NUMA balancing:
  - `# echo 0 > /proc/sys/kernel/numa_balancing`
- To enable automatic NUMA balancing:
  - `# echo 1 > /proc/sys/kernel/numa_balancing`
- At boot:
  - `numa_balancing=enable|disable`

Hugepages

- Traditionally x86 hardware gave us 4KiB pages
- The more memory the bigger the overhead in managing 4KiB pages
- What if you had bigger pages?
  - 512 times bigger → 2MiB

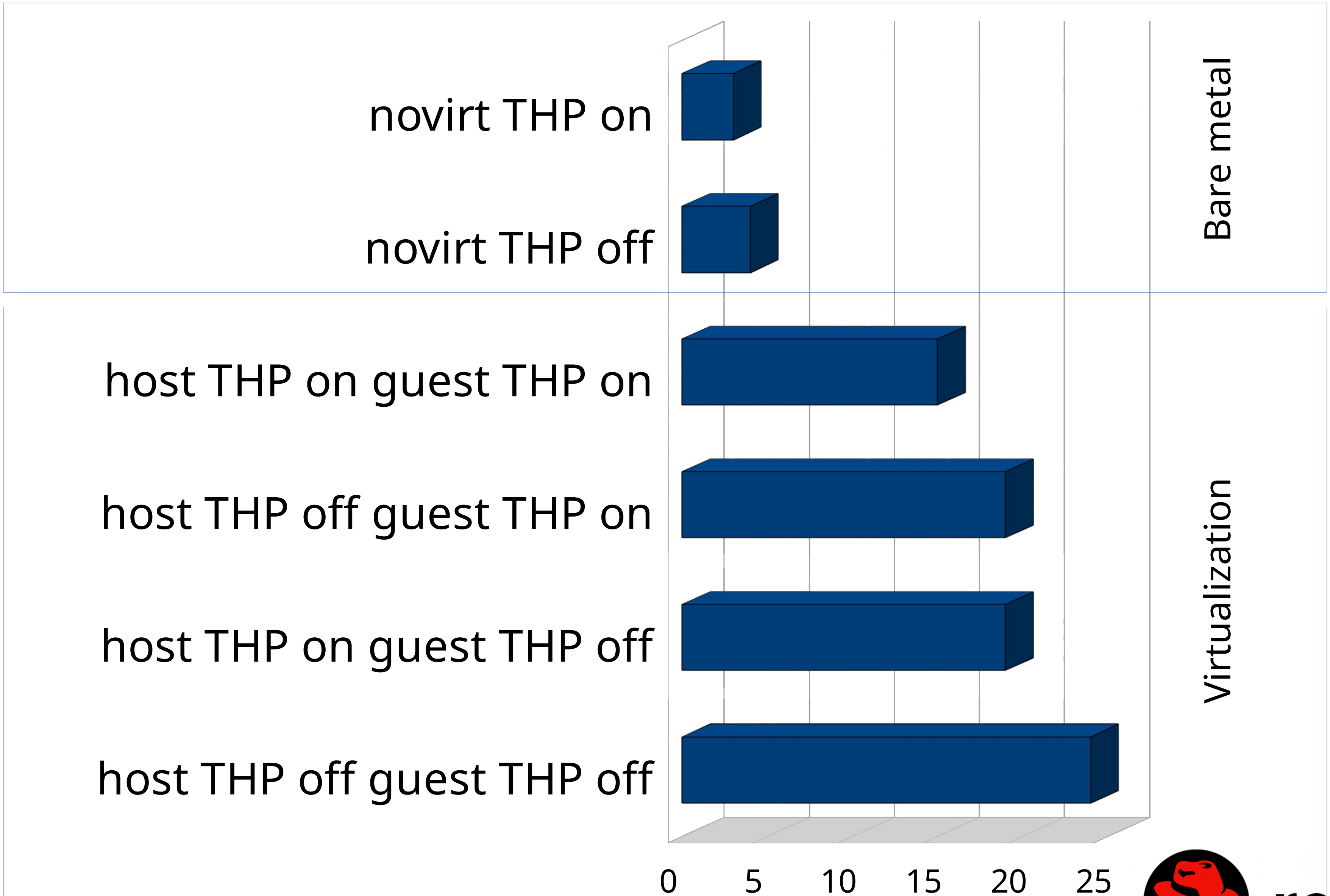
PageTables


pgd
2^9 = 512 arrows, not just 2
pud
pud
pmd
pmd
pmd
pmd
pte
pte
pte
pte
pte
pte
pte
pte

Benefit of hugepages

- Improve CPU performance
  - Enlarge TLB size (essential for KVM)
  - Speed up TLB miss (essential for KVM)
    - Need 3 accesses to memory instead of 4 to refill the TLB
  - Faster to allocate memory initially (minor)
  - Page colouring inside the hugepage (minor)
  - Higher scalability of the page LRUs
- Cons
  - clear_page/copy_page less cache friendly
  - higher memory footprint sometime
  - Direct compaction takes time

TLB miss cost: number of accesses to memory



redhat

Transparent Hugepage design

- How do we get the benefits of hugetlbfs without having to configure anything?
  - Transparent Hugepage
    - Any Linux process will receive 2M pages
      - if the mmap region is 2M naturally aligned
      - If compaction succeeeds in producing hugepages
    - Entirely transparent to userland

THP sysfs enabled

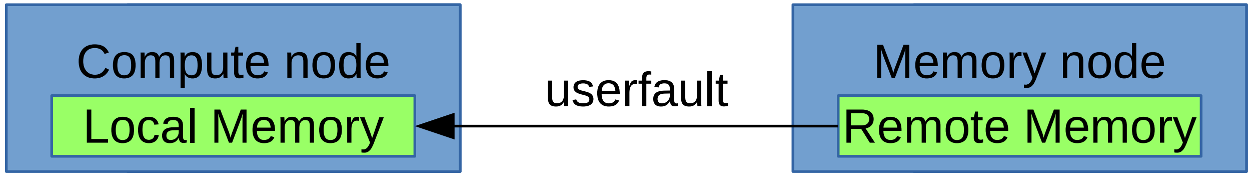
- `/sys/kernel/mm/transparent_hugepage/enabled`
  - `[always] madvise never`
    - Always use THP if vma start/end permits
  - `always [madvise] never`
    - Use THP only inside MAD_HUGEPAGE
      - Applies to khugepaged too
  - `always madvise [never]`
    - Never use THP
      - khugepaged quits
- Default selected at build time

THP defrag - compaction control

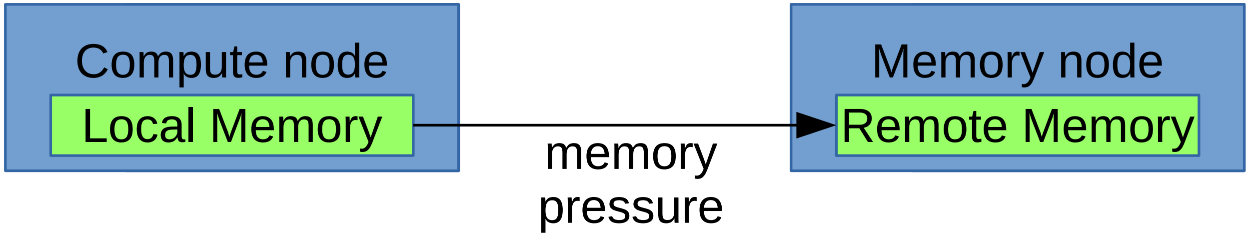
- `/sys/kernel/mm/transparent_hugepage/defrag`
  - `[always] defer madvise never`
    - Use direct compaction (ideal for long lived allocations)
  - `always [defer] madvise never`
    - Defer compaction asynchronously (kswapd/kcompact)
  - `always defer [madvise] never`
    - **Use direct compaction only inside MAD_HUGEPAGE**
  - `always defer madvise [never]`
    - Never use compaction
- Disabling THP is excessive if direct compaction is too expensive
- Default will change to defer to reduce allocation latency
- KVM uses MADV_HUGEPAGE
  - MADV_HUGEPAGE will still use direct compaction

Why: Memory Externalization

- Memory externalization is about running a program with part (or all) of its memory residing on a remote node
- Memory is transferred from the memory node to the compute node on access

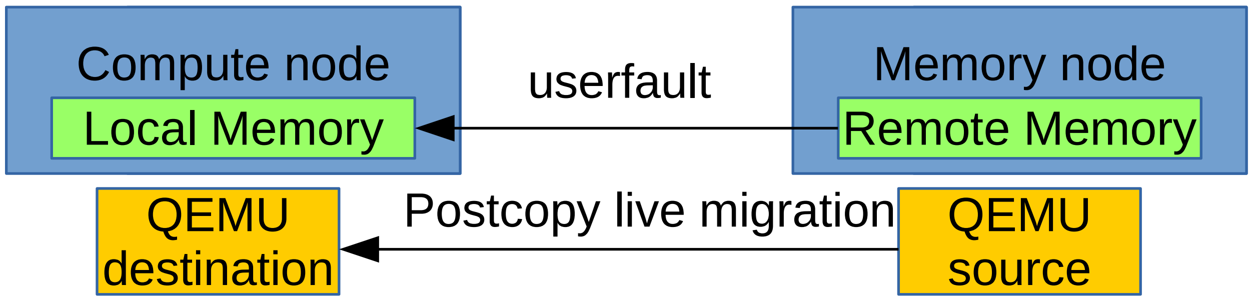


- Memory can be transferred from the compute node to the memory node if it's not frequently used during memory pressure



Postcopy live migration

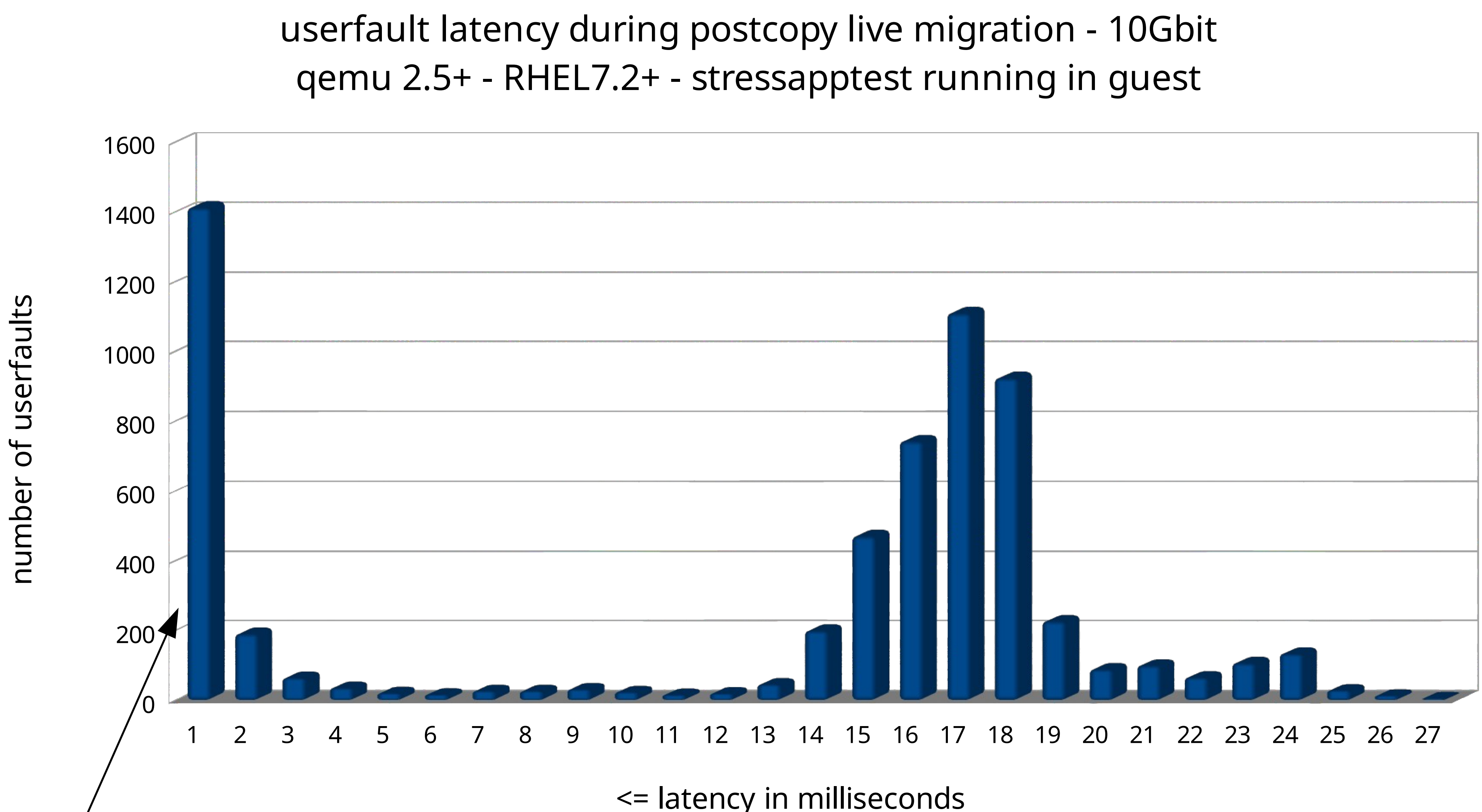
- **Postcopy live migration** is a form of memory externalization



- When the QEMU compute node (destination) faults on a missing page that resides in the memory node (source) the kernel has no way to fetch the page
  - Solution: let QEMU in userland handle the pagefault

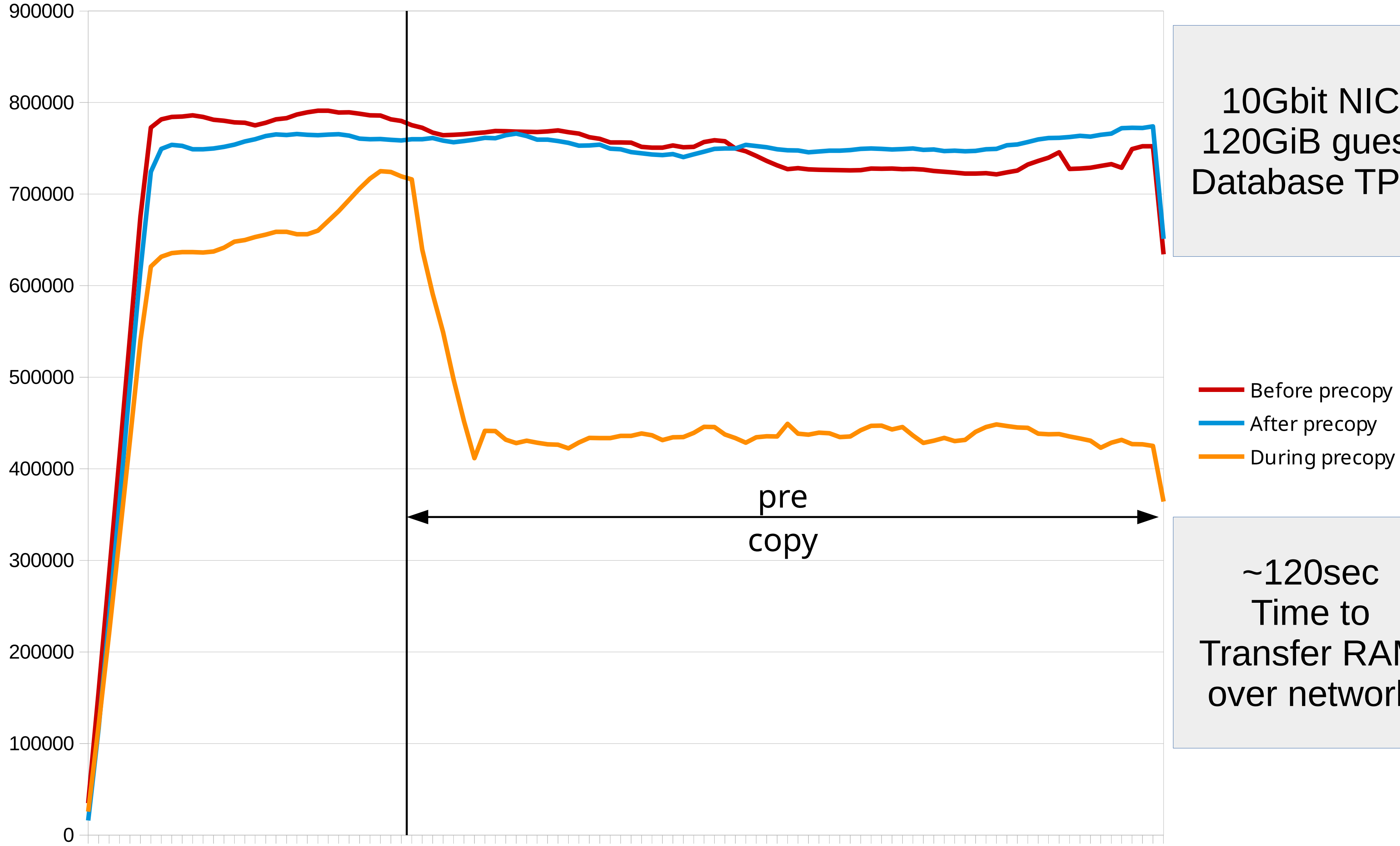
Partially funded by the Orbit *European Union* project

userfaultfd latency



Userfaults triggered on pages that were already in network-flight are instantaneous. Background transfer seeks at the last userfault address.

KVM precopy live migration

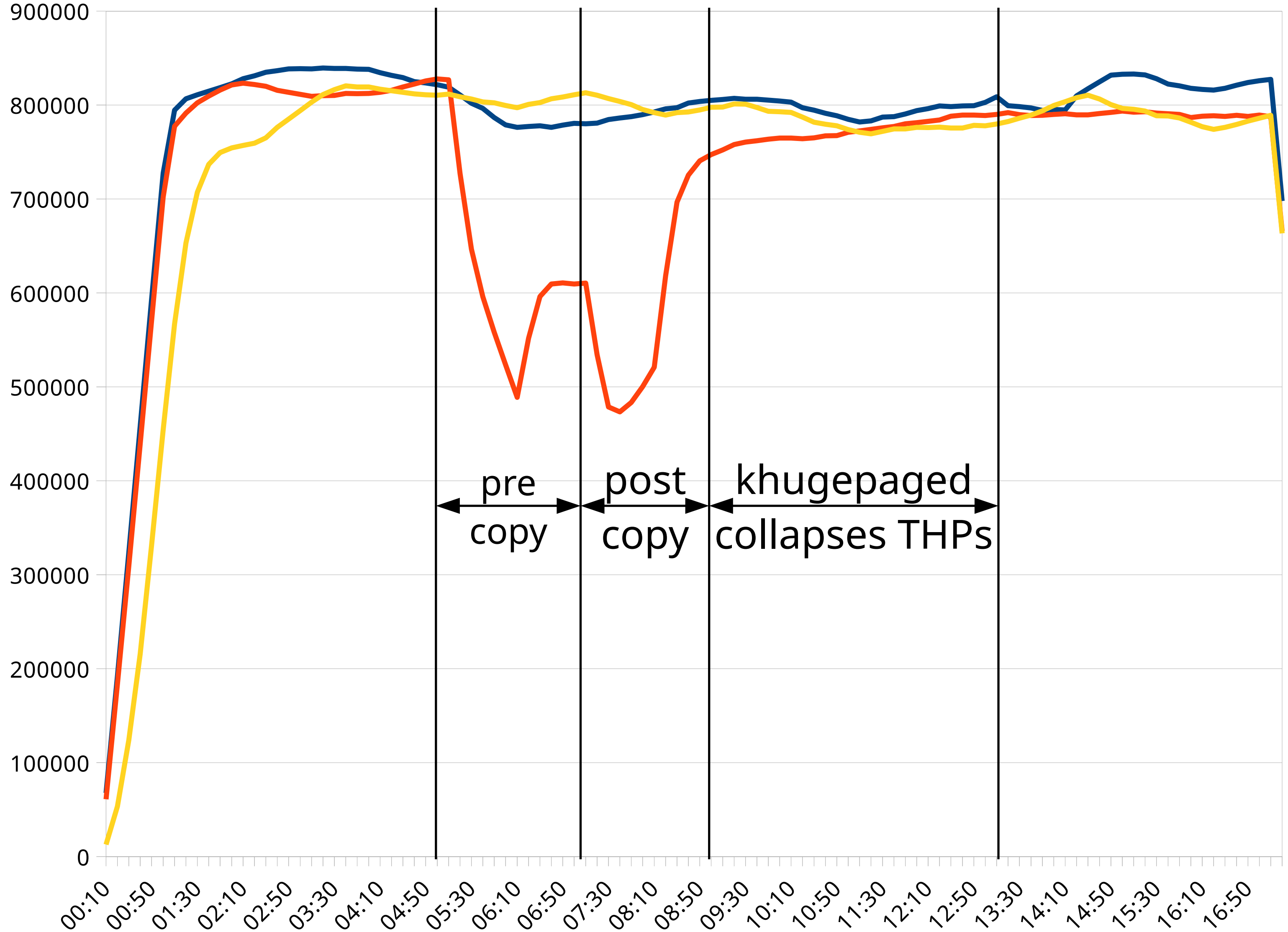


10Gbit NIC
120GiB guest
Database TPM

~120sec
Time to
Transfer RAM
over network

Precopy never completes until the database benchmark completes

KVM postcopy live migration



precopy runs
From 5m
To 7m

postcopy runs
From 7m
To about ~9m
deterministic

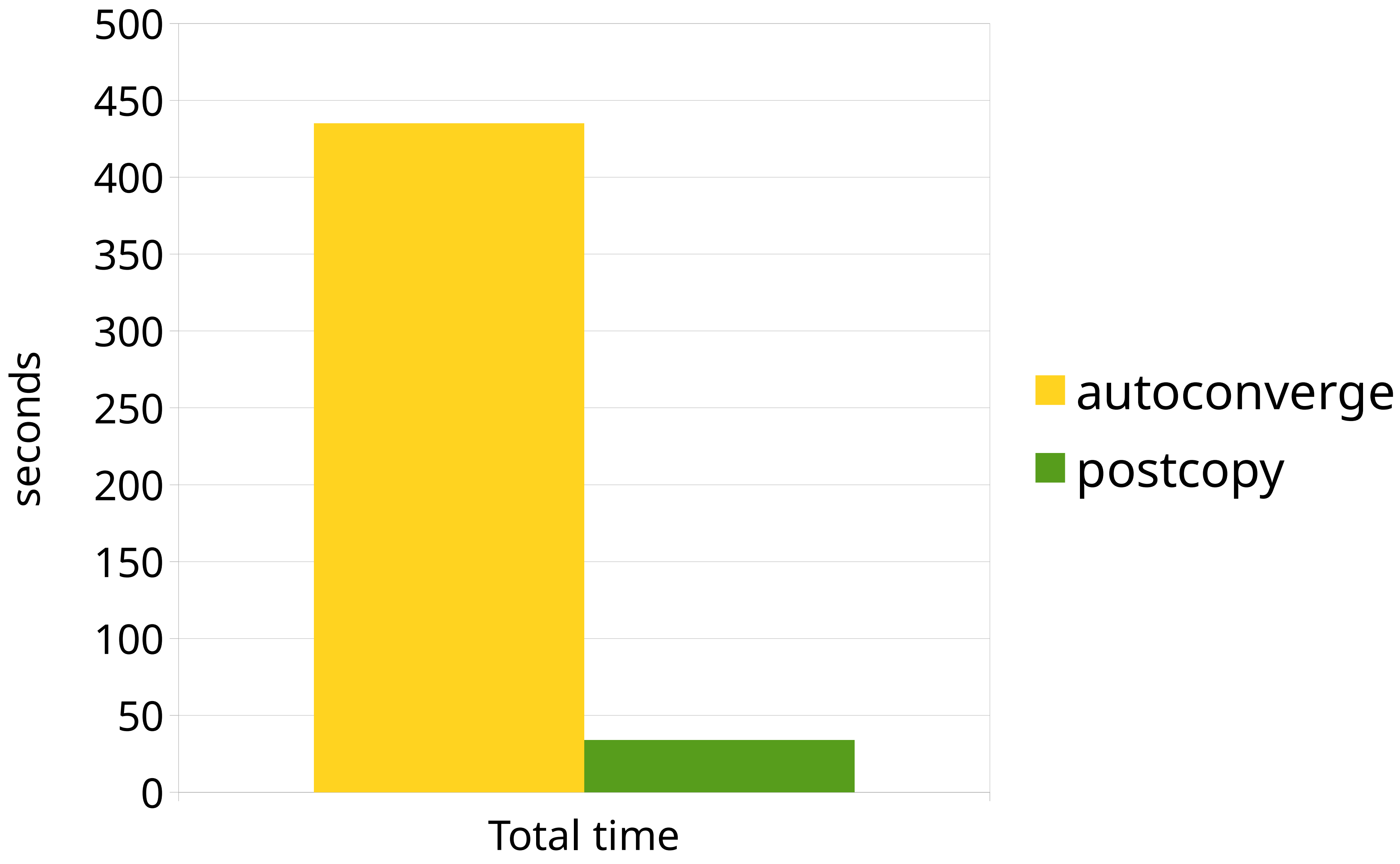
virsh migrate .. --postcopy --timeout <sec> --timeout-postcopy

 # virsh migrate .. --postcopy --postcopy-after-precopy

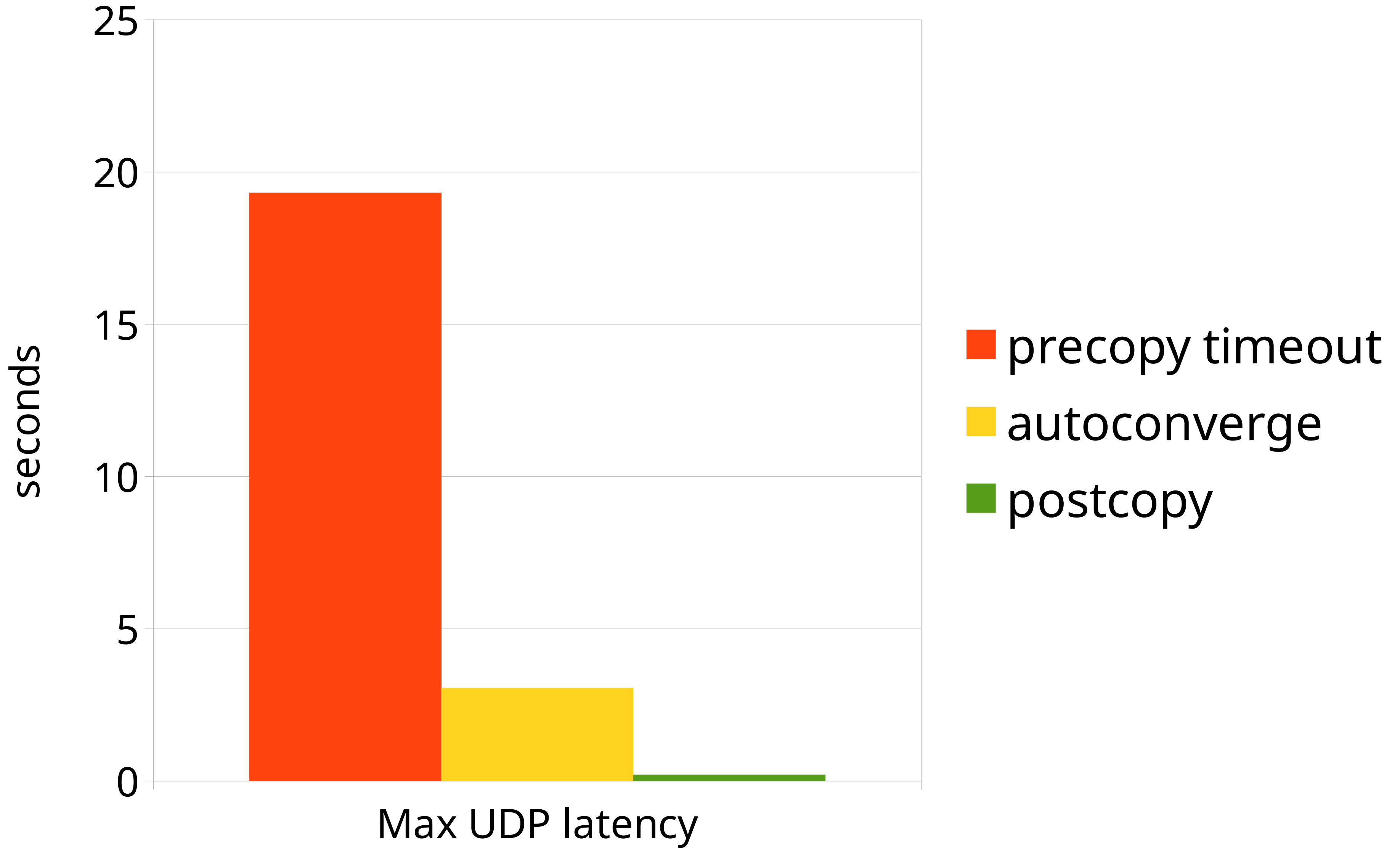
All available upstream

- Userfaultfd() syscall in Linux Kernel >= v4.3
- Postcopy live migration in:
  - QEMU >= v2.5.0
    - Author: *David Gilbert @ Red Hat Inc.*
  - Postcopy in Libvirt >= 1.3.4
  - OpenStack Nova >= Newton
- ... and coming soon in production starting with:
  - **RHEL 7.3**

Live migration total time



Live migration max perceived downtime latency



Virtual Memory evolution since '99

- Amazing to see the room for further innovation there was back then
  - Things constantly looks pretty mature
    - They may actually have been considering my hardware back then was much less powerful and not more complex than my cellphone
    - Unthinkable to maintain the current level of mission critical complexity by reinventing the wheel in a not Open Source way
      - Can perhaps be still done in a limited set of laptops and cellphones models, but for how long?
- Innovation in the Virtual Memory space is probably one among the plenty of factors that contributed to Linux success and the KVM lead in OpenStack user base too
  - KVM (unlike the preceding Hypervisor designs) leverages the power of the Linux Virtual Memory in its **entirety**