20 years of Linux Virtual Memory

Red Hat, Inc.

Andrea Arcangeli <aarcange at redhat.com>

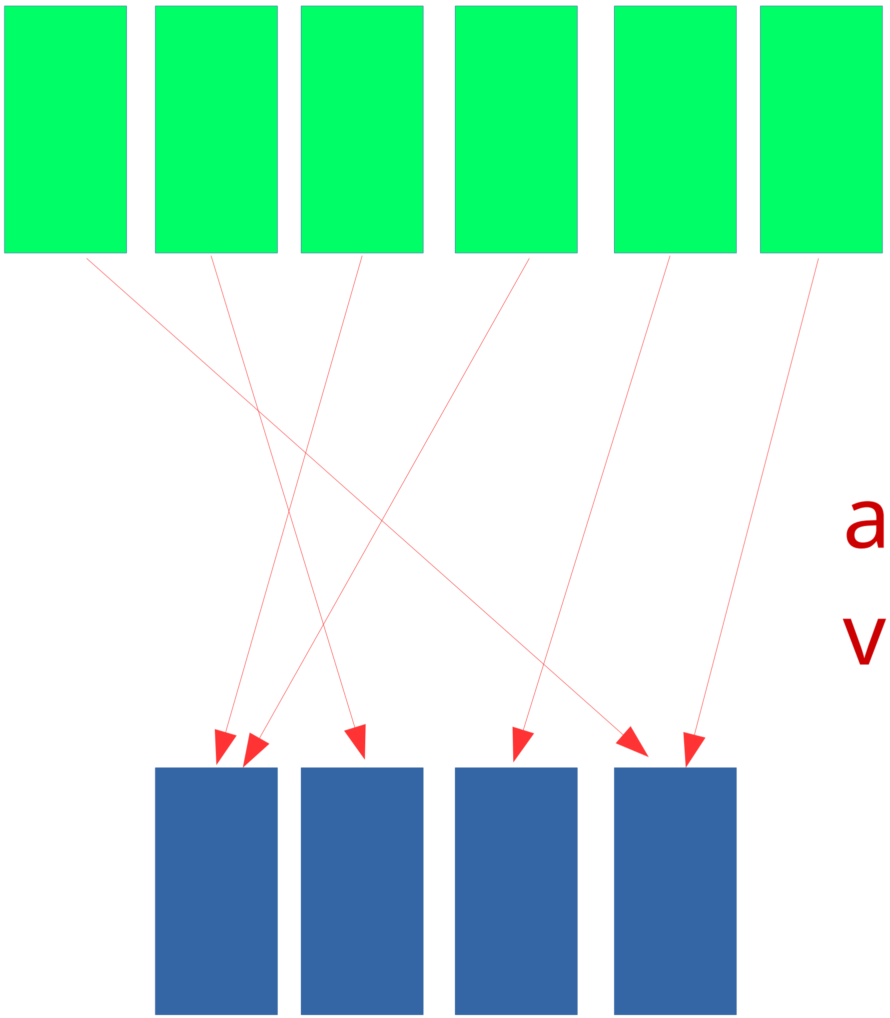
Kernel Recipes, Paris

29 Sep 2017

Topics

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

Virtual Memory (simplified)



Virtual pages

They cost “nothing”

Practically unlimited

on 64bit archs

arrows = pagetables

virtual to physical mapping

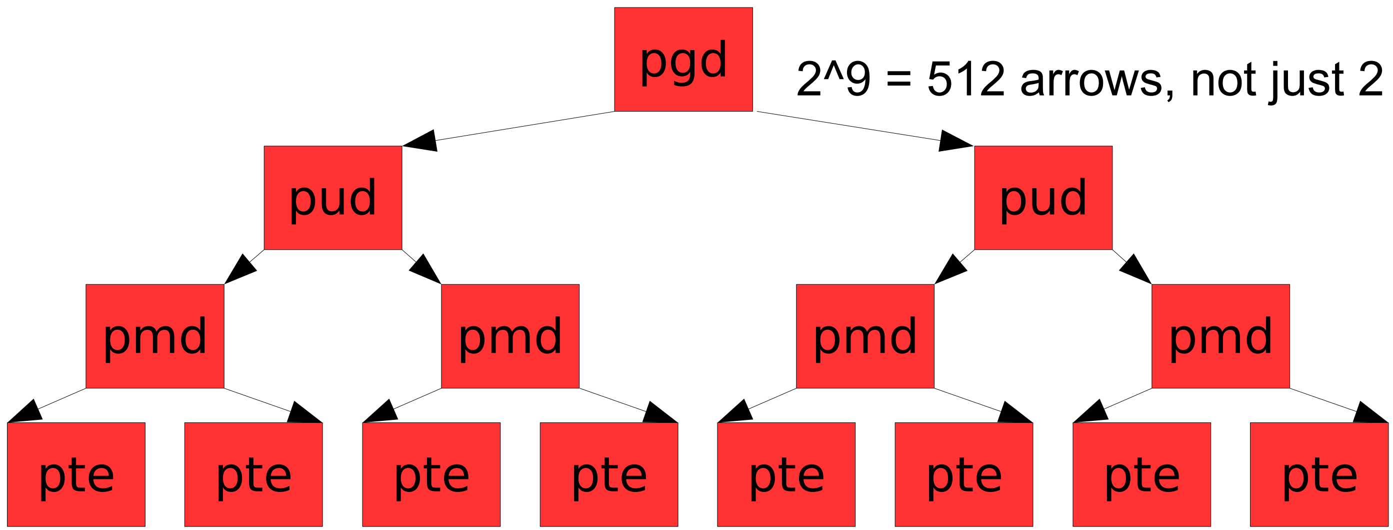
Physical pages

They cost money!

This is the RAM

PageTables

- Common code and x86 pagetable format is a tree

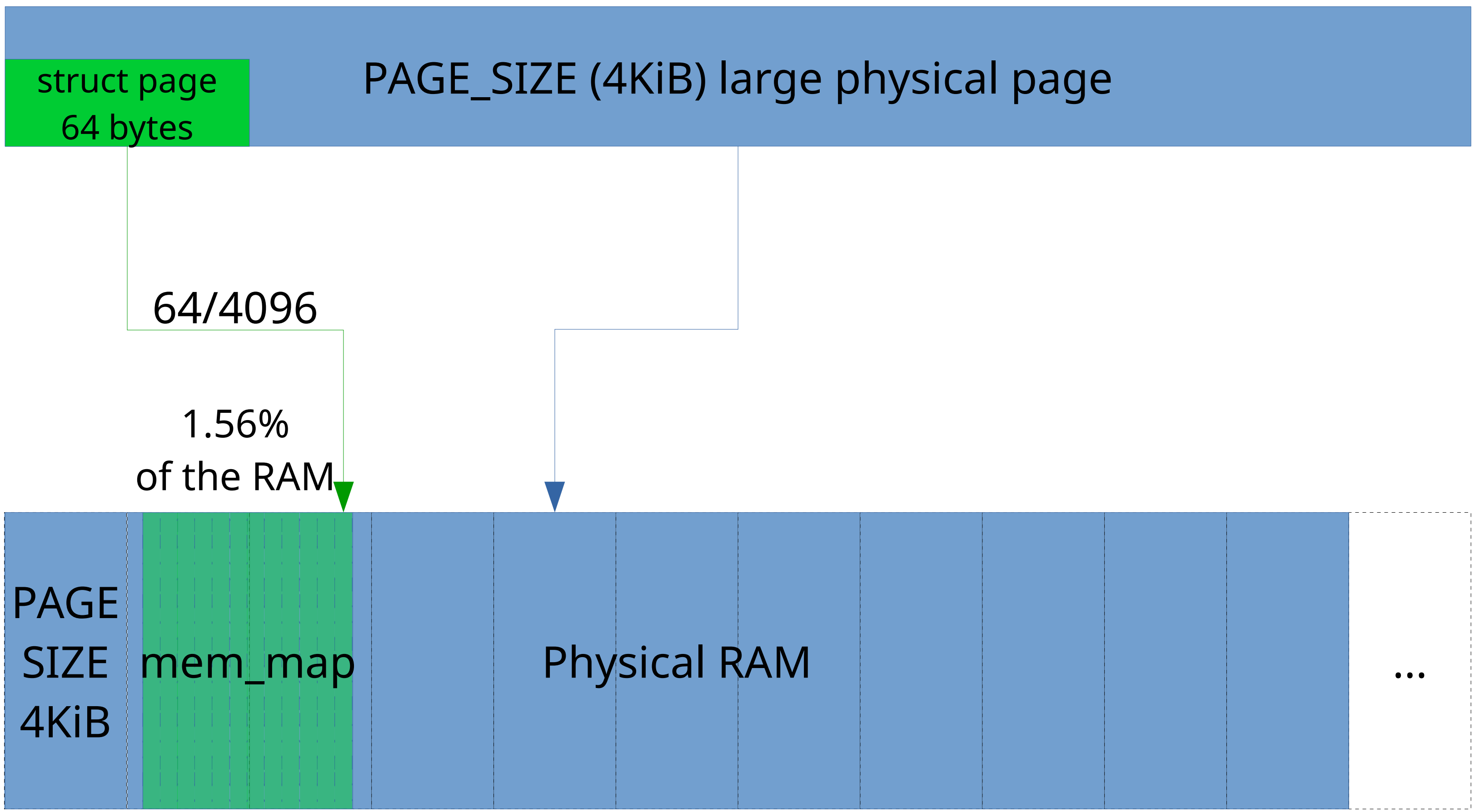


- All pagetables are 4KB in size
- Total: grep PageTables /proc/meminfo
- (((2**9)**4)*4096)>>48 = 1 → 48bits → 256 TiB
- **5 levels in v4.13** → (48+9) bits → 57bits → 128 PiB
  - Build time to avoid slowdown

The Fabric of the Virtual Memory

- The fabric are all those data structures that connects to the hardware constrained 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

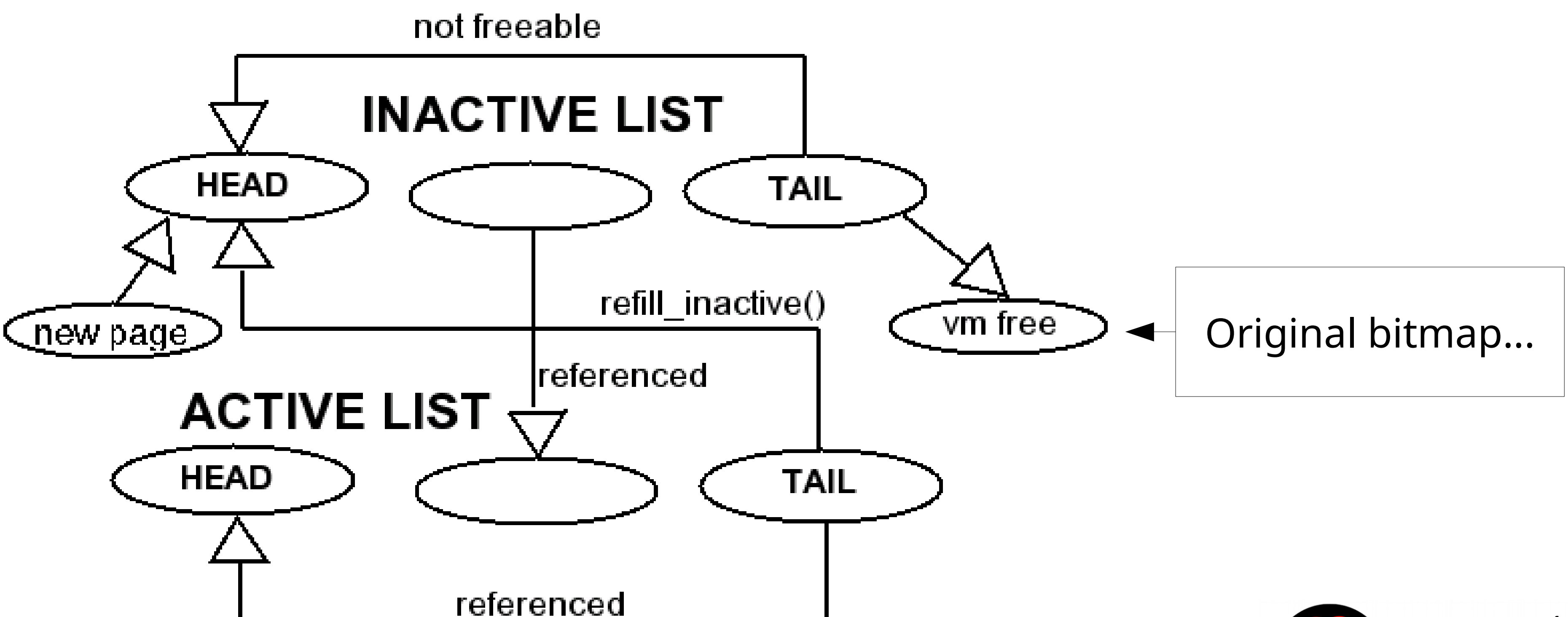


MM & VMA

- 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”

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 re-faults



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
```

Active LRU working-set detection

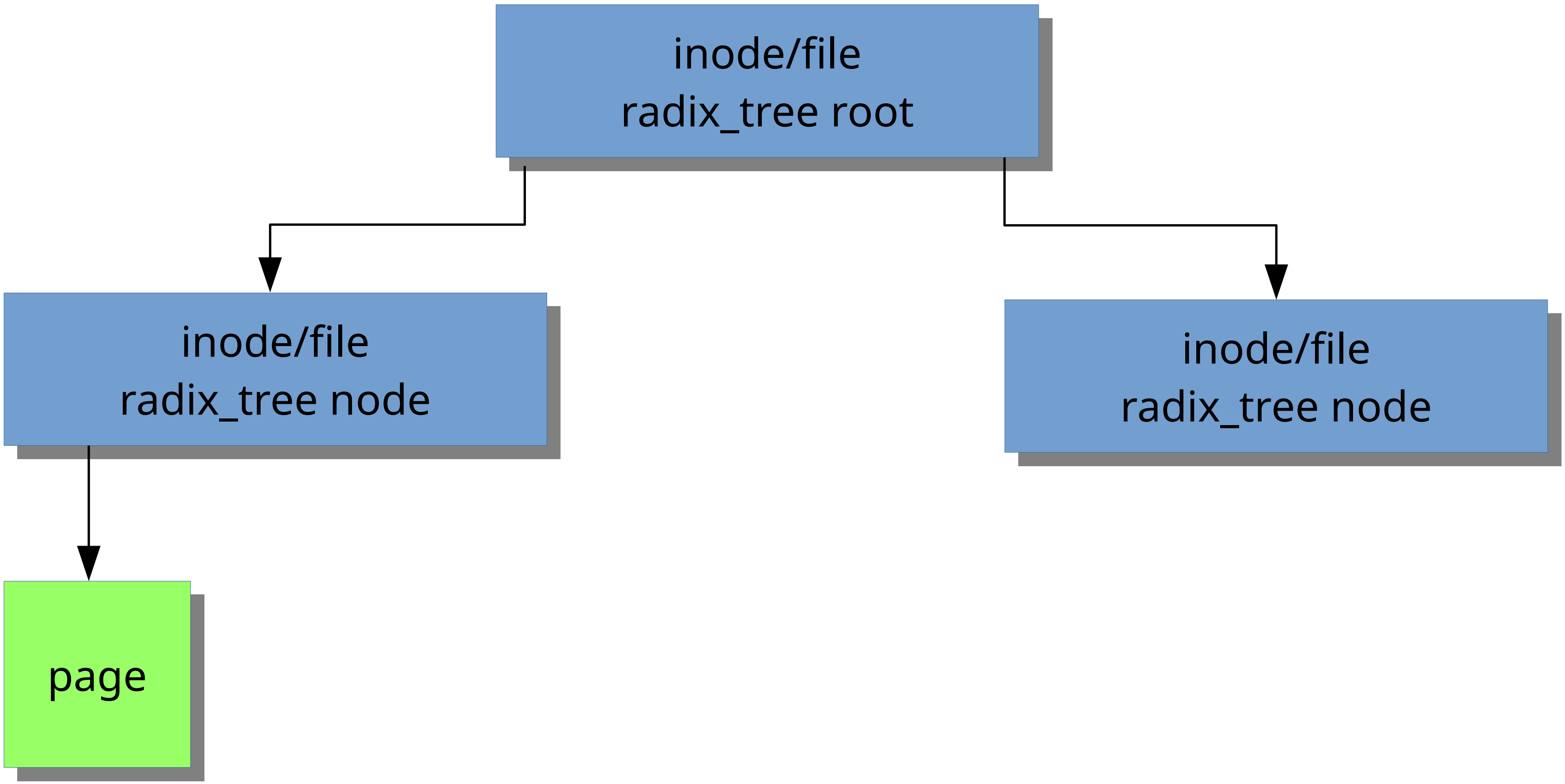
- From *Johannes Weiner*

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

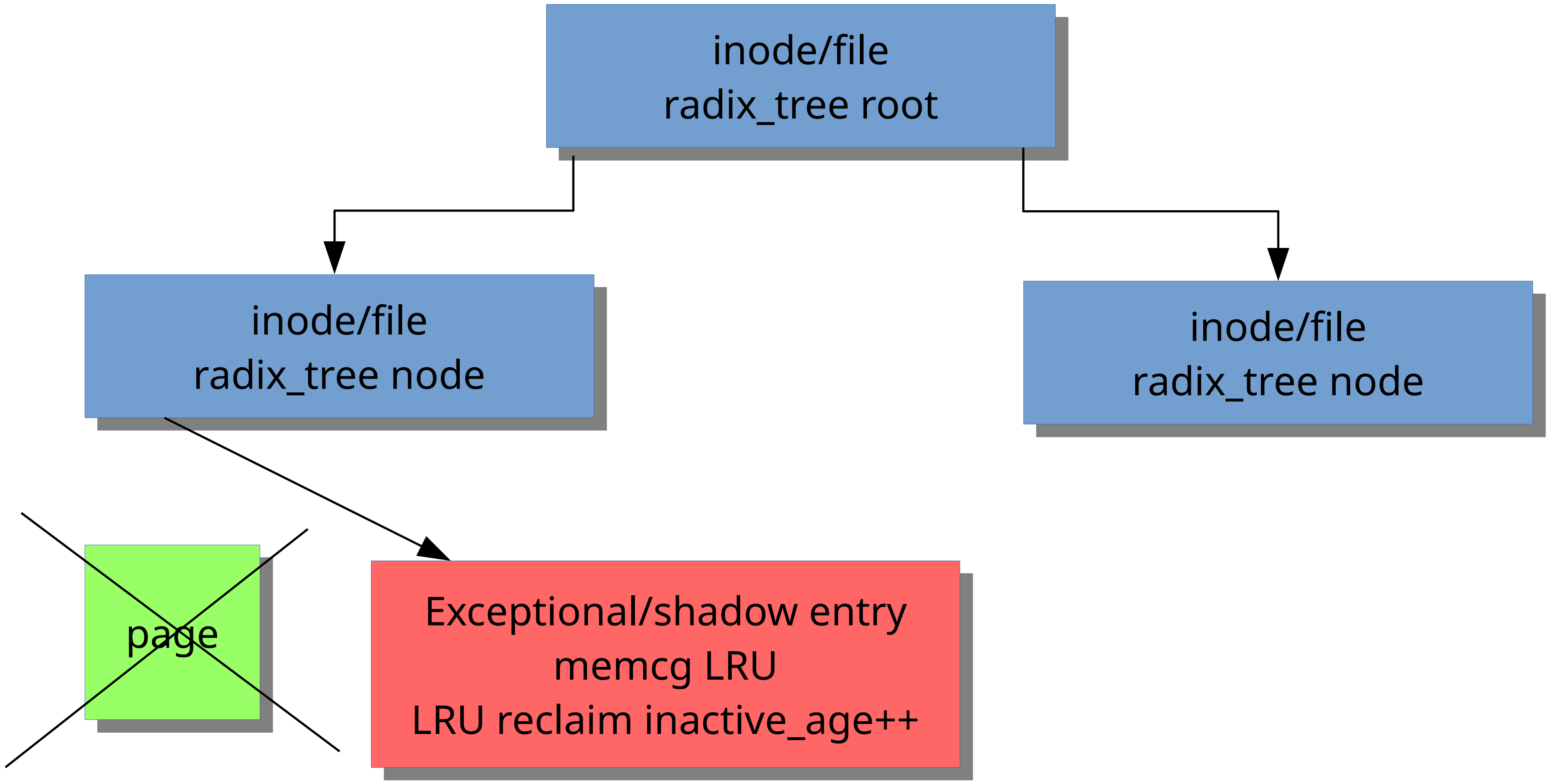
```
H = Head, T = Tail
```

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

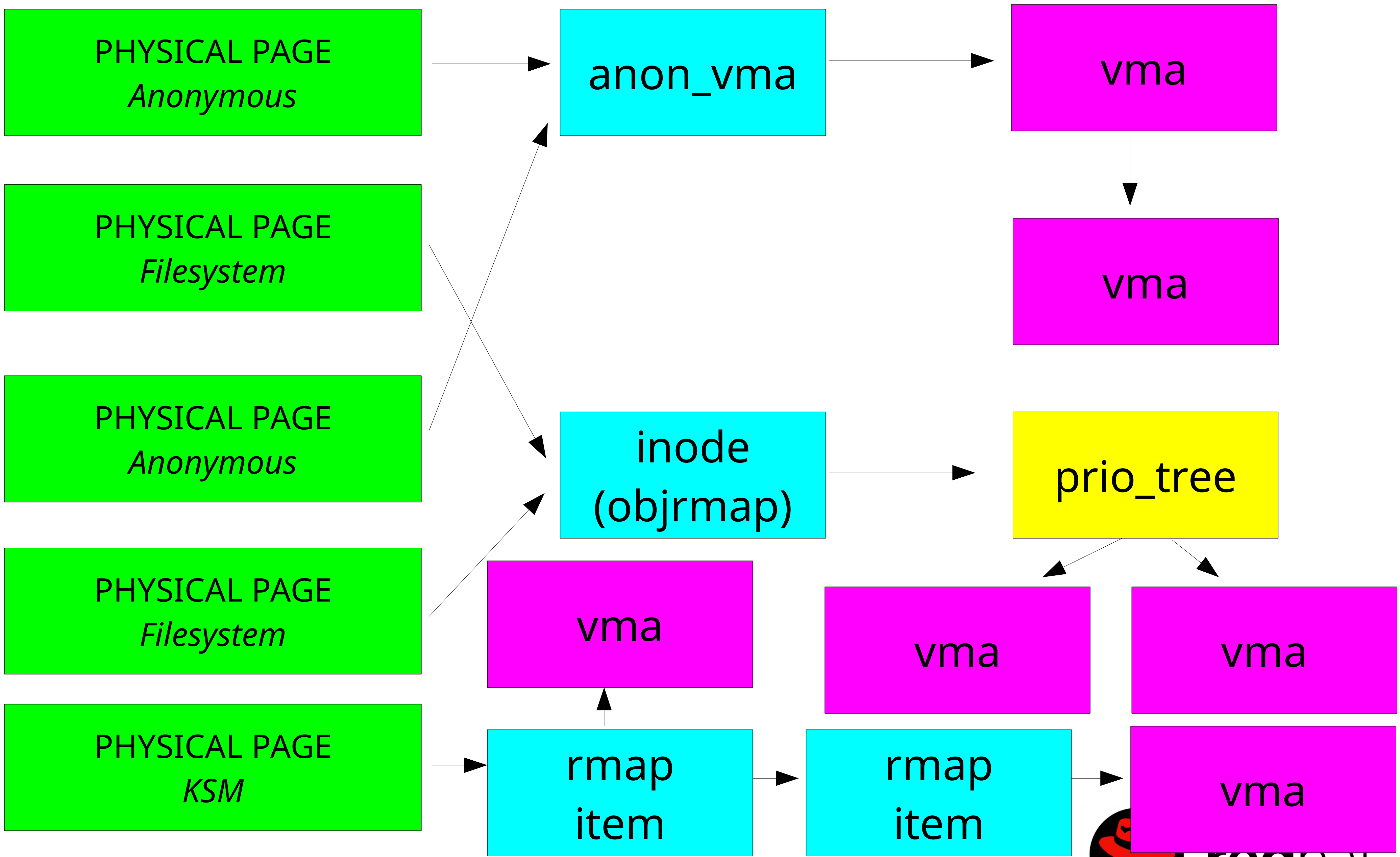
lru→inactive_age and radix tree shadow entries



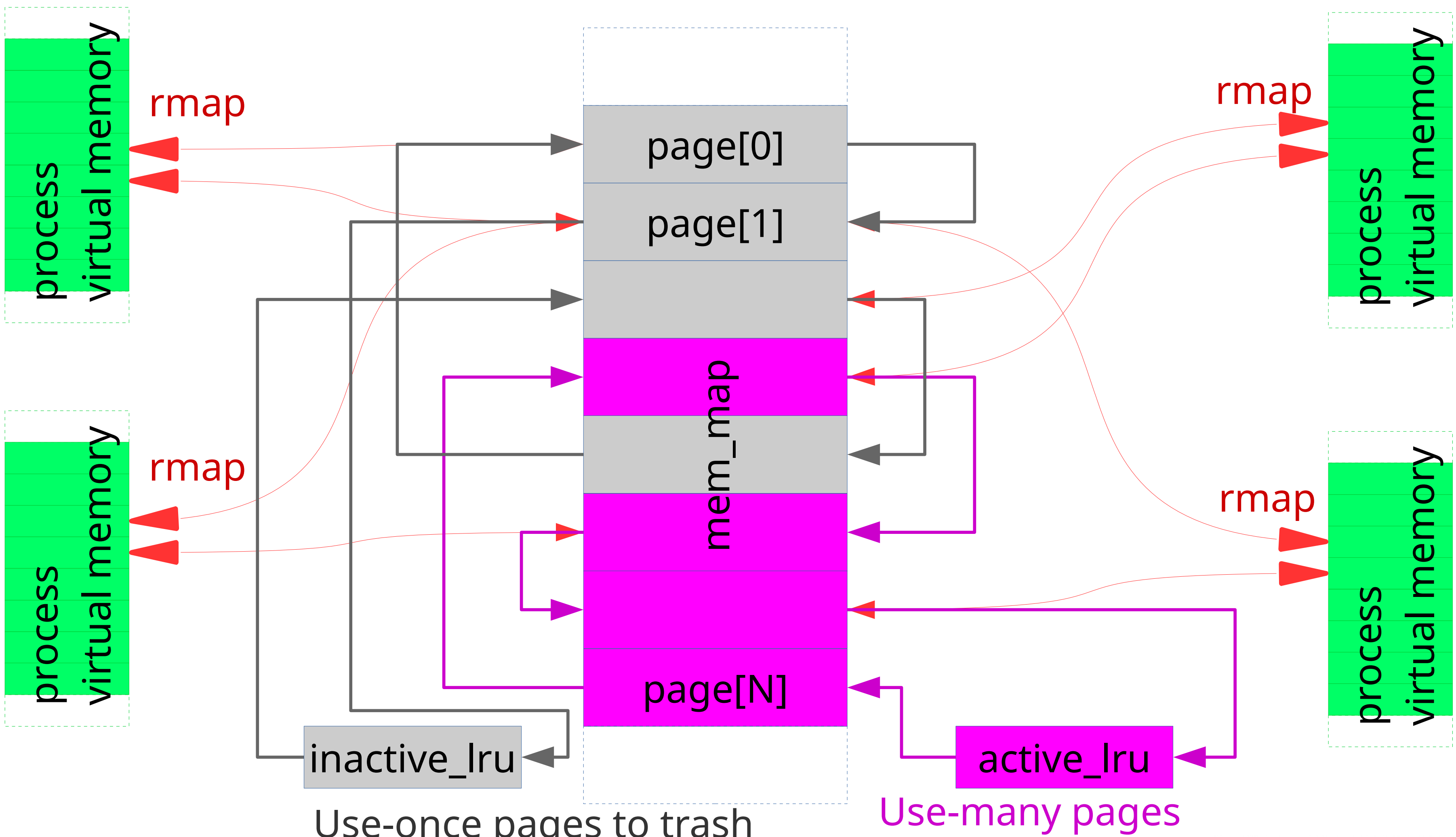
Reclaim saving inactive_age



Object-based reverse mapping



Active & inactive + rmap



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

- Optimizing the workloads for you, without manual tuning
  - NUMA hard bindings (numactl) → Automatic NUMA Balancing
  - Hugetlbfs → Transparent Hugepage
  - 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***

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)

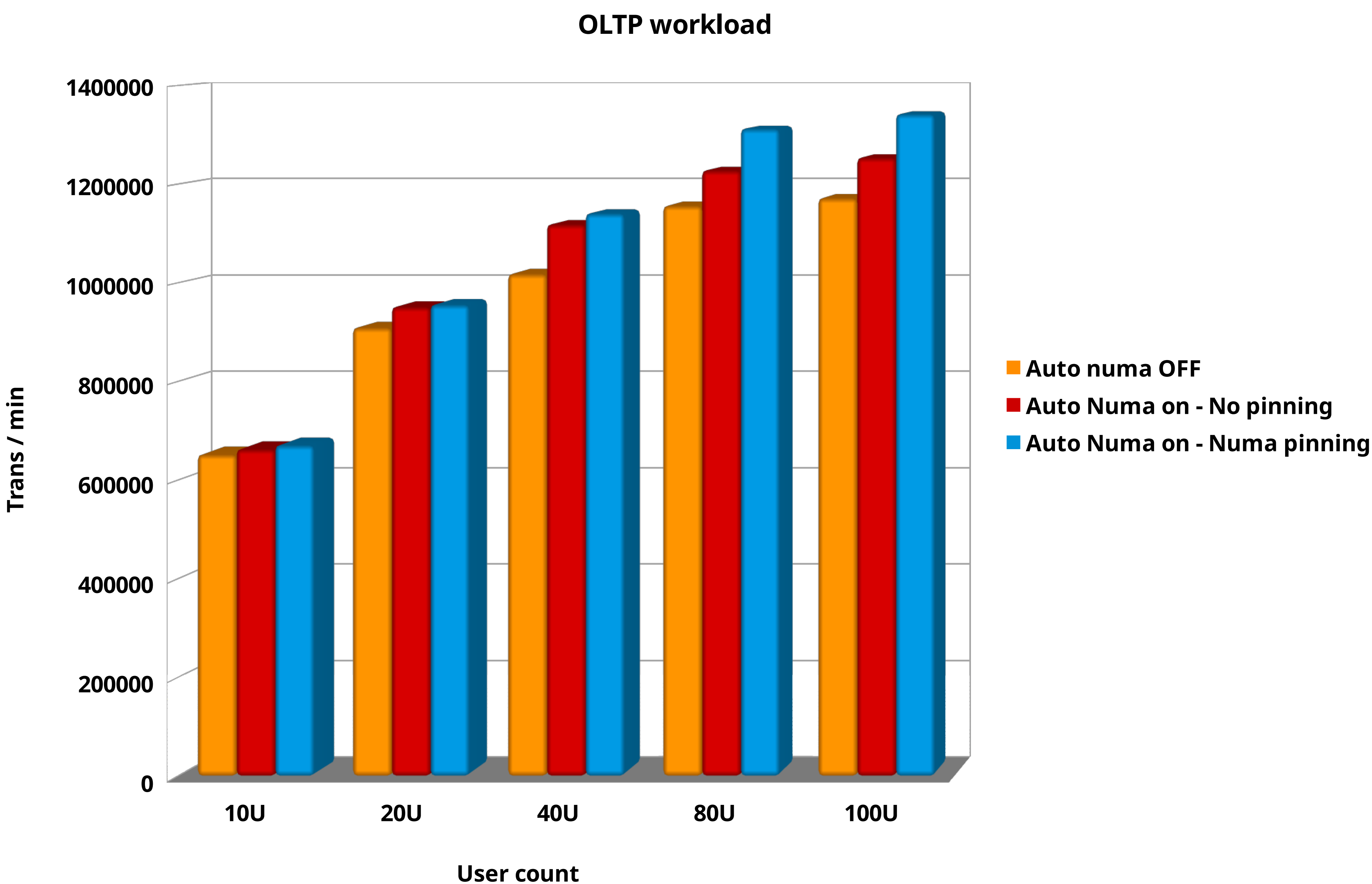
VM – 32P , 160G (Optimized for Server)

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



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
  ```

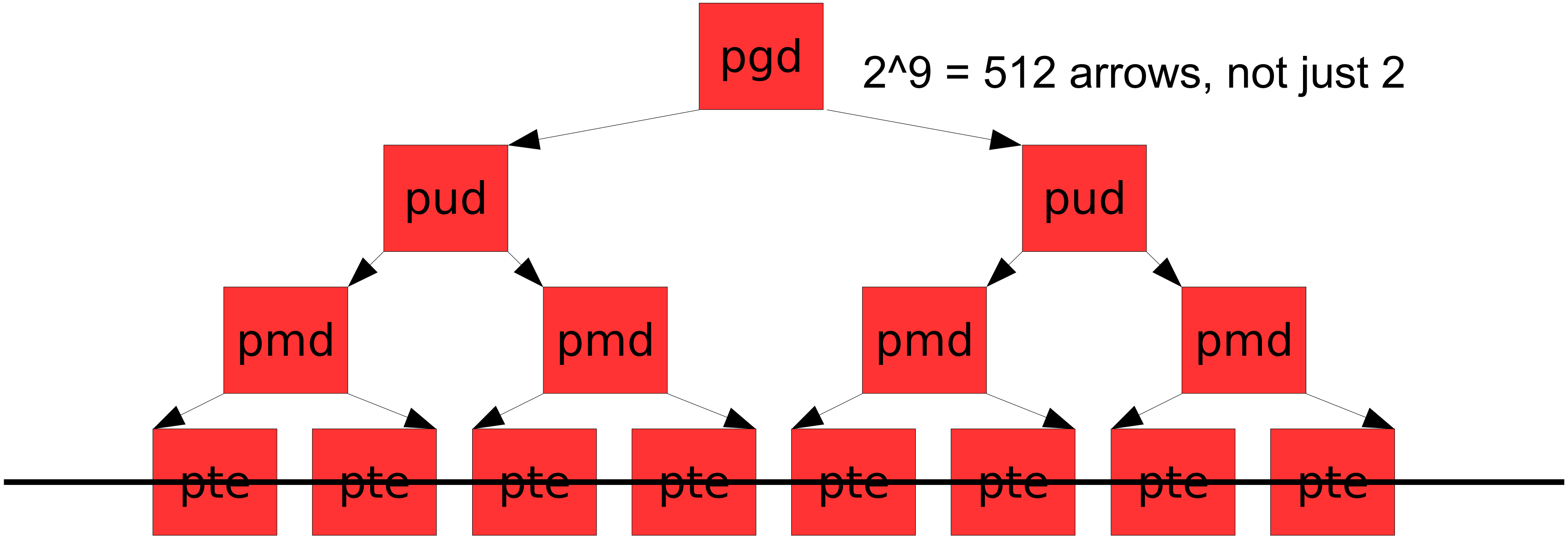
- At boot:

  ```
  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

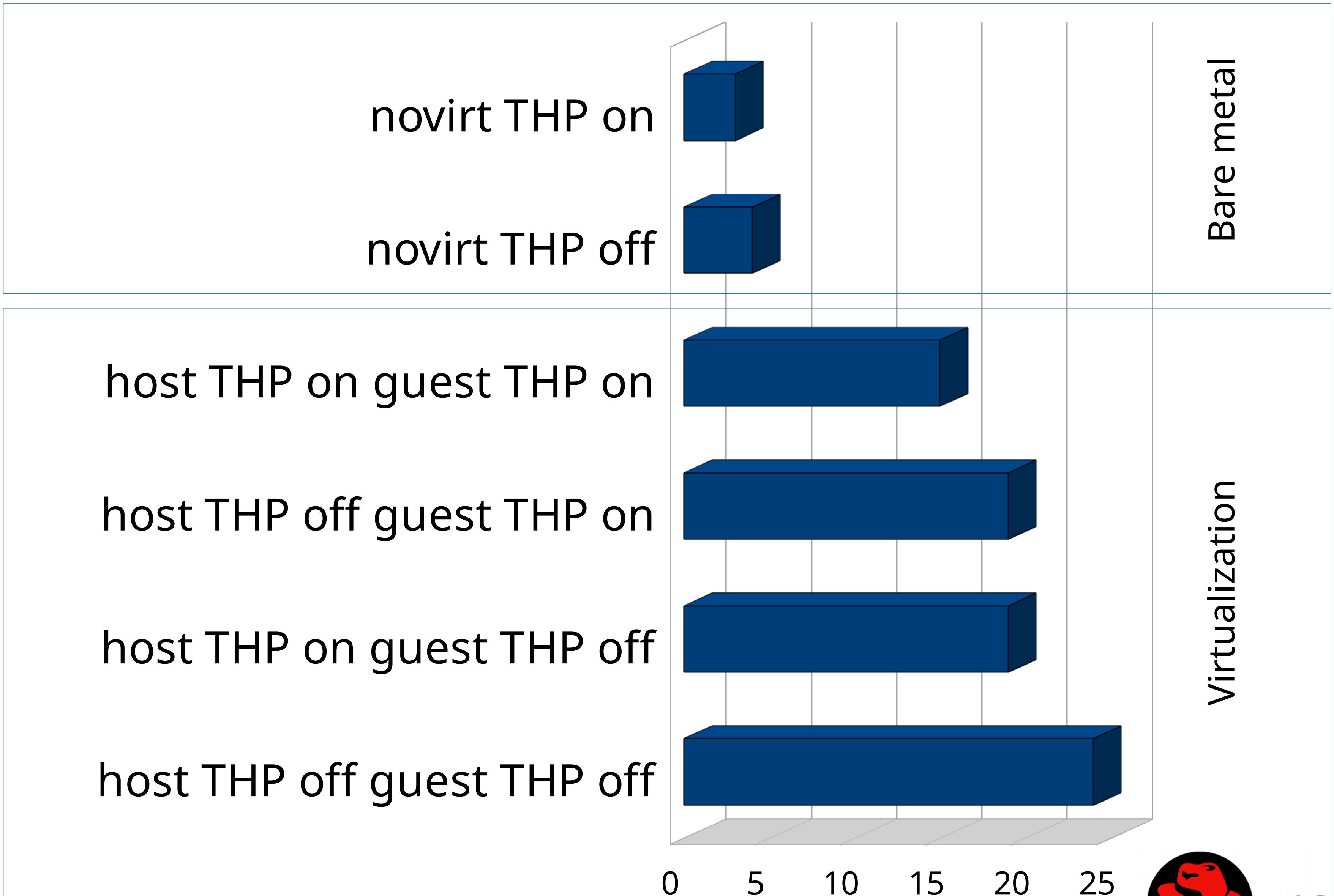

pgd
2^9 = 512 arrows, not just 2
pud
pud
pmd
pmd
pmd
pmd
pte
pte
pte
pte
pte
pte
pte
pte

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
    - **Clear faulting subpage last included in v4.13 from Andi Kleen and Ying Hhuang**
      - **28.3% increase in vm-scalability anon-w-seq**
  - higher memory footprint sometime
  - Direct compaction takes time

redhat

TLB miss cost: number of accesses to memory



Transparent Hugepage design

- How do we get the benefits of hugetlbfs without having to configure anything?
  - Any Linux process will receive 2M pages
    - if the mmap region is 2M naturally aligned
    - If compaction succeeds 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 MADV_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 defer+madvise madvise never`
    - Always use direct compaction (ideal for long lived allocations)
  - `always [defer] defer+madvise madvise never`
    - Defer compaction asynchronously (kswapd/kcompactd)
  - `always defer [defer+madvise] madvise never`
    - `Direct compaction in MADV_HUGEPAGE, async otherwise`
  - **`always defer defer+madvise [madvise] never`**
    - **Use direct compaction only inside MADV_HUGEPAGE**
    - **khugepaged enabled in the background**
  - `always defer defer+madvise madvise [never]`
    - Never use compaction, quit khugepaged too
- Disabling THP is excessive if only direct compaction is too expensive
- KVM uses MADV_HUGEPAGE
  - ***MADV_HUGEPAGE will still use direct compaction***

Priming the VM

- To achieve maximum THP utilization you can run the two commands below

```
# cat /proc/buddyinfo
Node 0, zone      DMA      0      0      1      1      1      1      1      1      0      1      3
Node 0, zone    DMA32   8357   4904   2670    472    126     82      8      1      0      3      5
Node 0, zone   Normal   5364  38097    608     63     11      1      1     55      0      0      0
# echo 3 >/proc/sys/vm/drop_caches
# cat /proc/buddyinfo
Node 0, zone      DMA      0      0      1      1      1      1      1      1      0      1      3
Node 0, zone    DMA32   5989   5445   5235   5022   4420   3583   2424   1331    471    119     25
Node 0, zone   Normal  71579  58338  36733  19523   6431   1572    559    282    115     29      2
# echo >/proc/sys/vm/compact_memory
# cat /proc/buddyinfo
Node 0, zone      DMA      0      0      1      1      1      1      1      1      0      1      3
Node 0, zone    DMA32   1218   1161   1227   1240   1135    925    688    502    342    305    369
Node 0, zone   Normal   7479   5147   5124   4240   2768   1959   1391    814    389    270    149
                          4k     8k    16k    32k    64k                                2M     4M
```

THP on tmpfs

- From Kirill A. Shutemov and Hugh Dickins
  - Merged in the **v4.8** kernel
- `# mount -o remount,huge=always none /dev/shm`
  - Including small files (i.e. < 2MB in size)
    - Very high memory usage on small files
- `# mount -o remount,huge=never none /dev/shm`
  - **Current default**
- `# mount -o remount,huge=within_size none /dev/shm`
  - Allocate THP if inode i_size can contain hugepages, or if there’s a madvise/fadvise hint
    - **Ideal for apps benefiting from THP on tmpfs!**
- `# mount -o remount,huge=advise none /dev/shm`
  - Only if requested through madvise/fadvise

THP on shmem

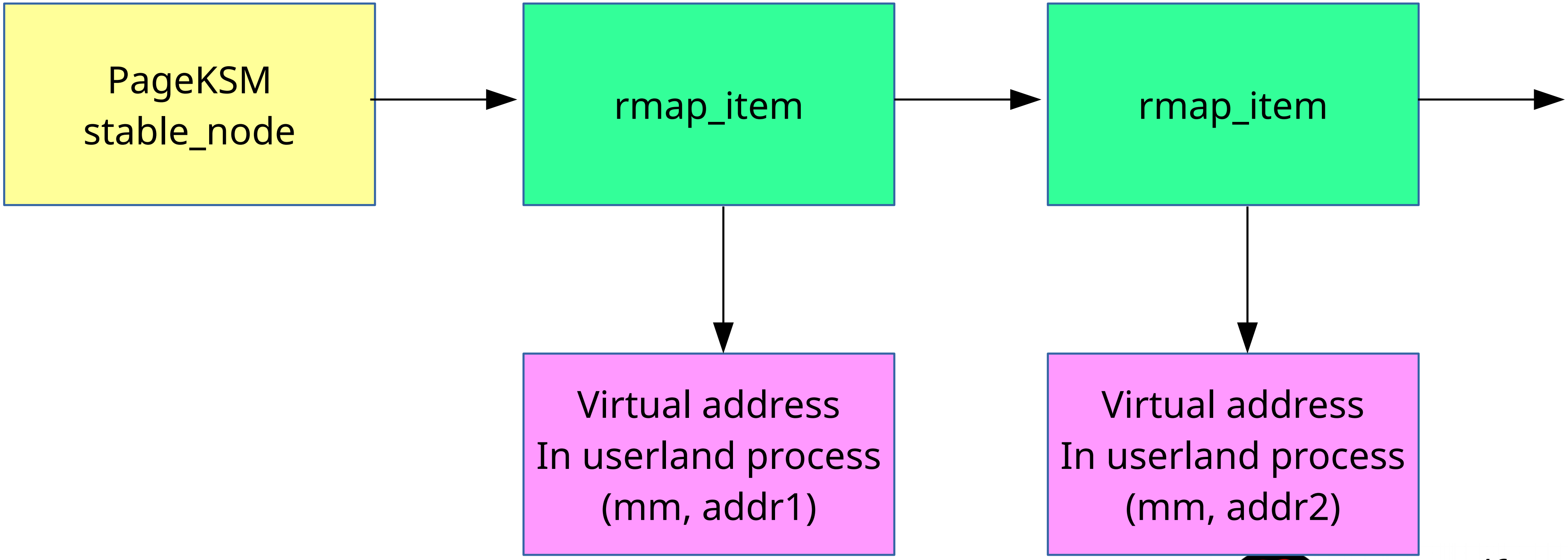
- shmem uses an internal *tmpfs* mount for: SysV SHM, memfd, MAP_SHARED of /dev/zero or MAP_ANONYMOUS, DRM objects, ashmem
- Internal mount is controlled by:

```
$ cat /sys/kernel/mm/transparent_hugepage/shmem_enabled
always within_size advise [never] deny force
```

- *`deny`*
  - Disable THP for all tmpfs mounts
    - For debugging
- *`force`*
  - Always use THP for all tmpfs mounts
    - For testing

KSMscale

- *Virtual memory* to dedup is practically **unlimited** (even on 32bit if you deduplicate across multiple processes)
- The same page content can be deduplicated an **unlimited** number of times



KSMscale

```
$ cat /sys/kernel/mm/ksm/max_page_sharing
256
```

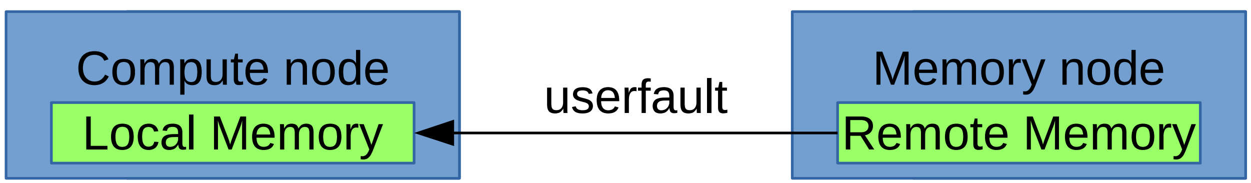
- KSMscale limits the maximum deduplication for each physical PageKSM allocated
  - Limiting “compression” to x256 is enough
  - Higher maximum deduplication generates diminishing returns
- To alter the max page sharing:

```
$ echo 2 > /sys/kernel/mm/ksm/run
$ echo 512 > /sys/kernel/mm/ksm/max_page_sharing
```

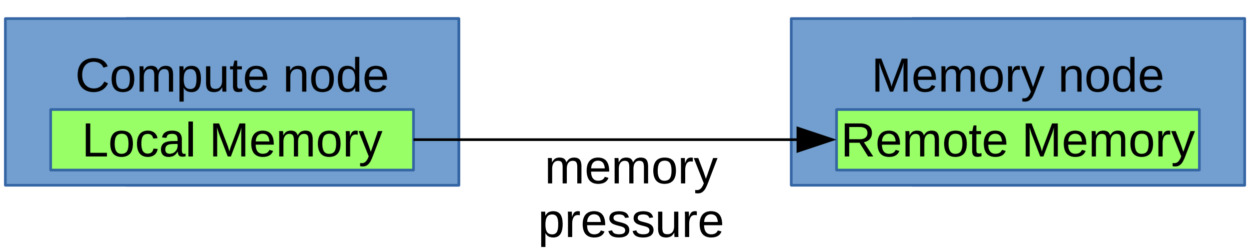
- **Included in v4.13**

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

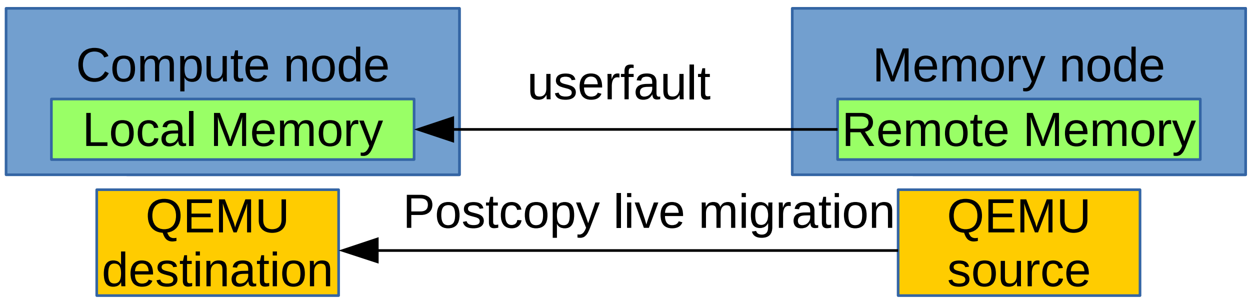


- Memory can be transferred from the compute node to the memory node if it's not frequently used during memory pressure



Postcopy live migration

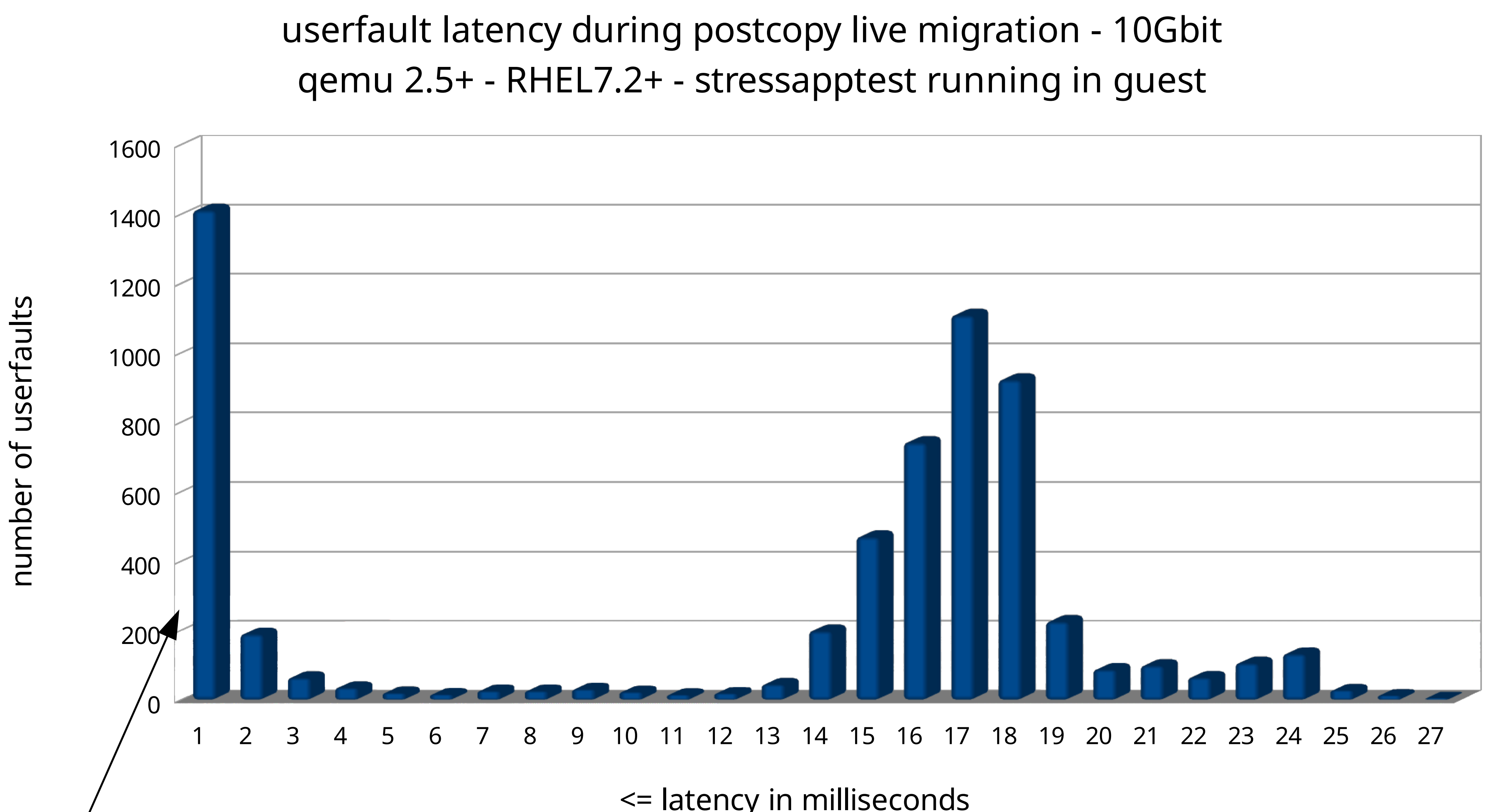
- **Postcopy live migration** is a form of memory externalization



- 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

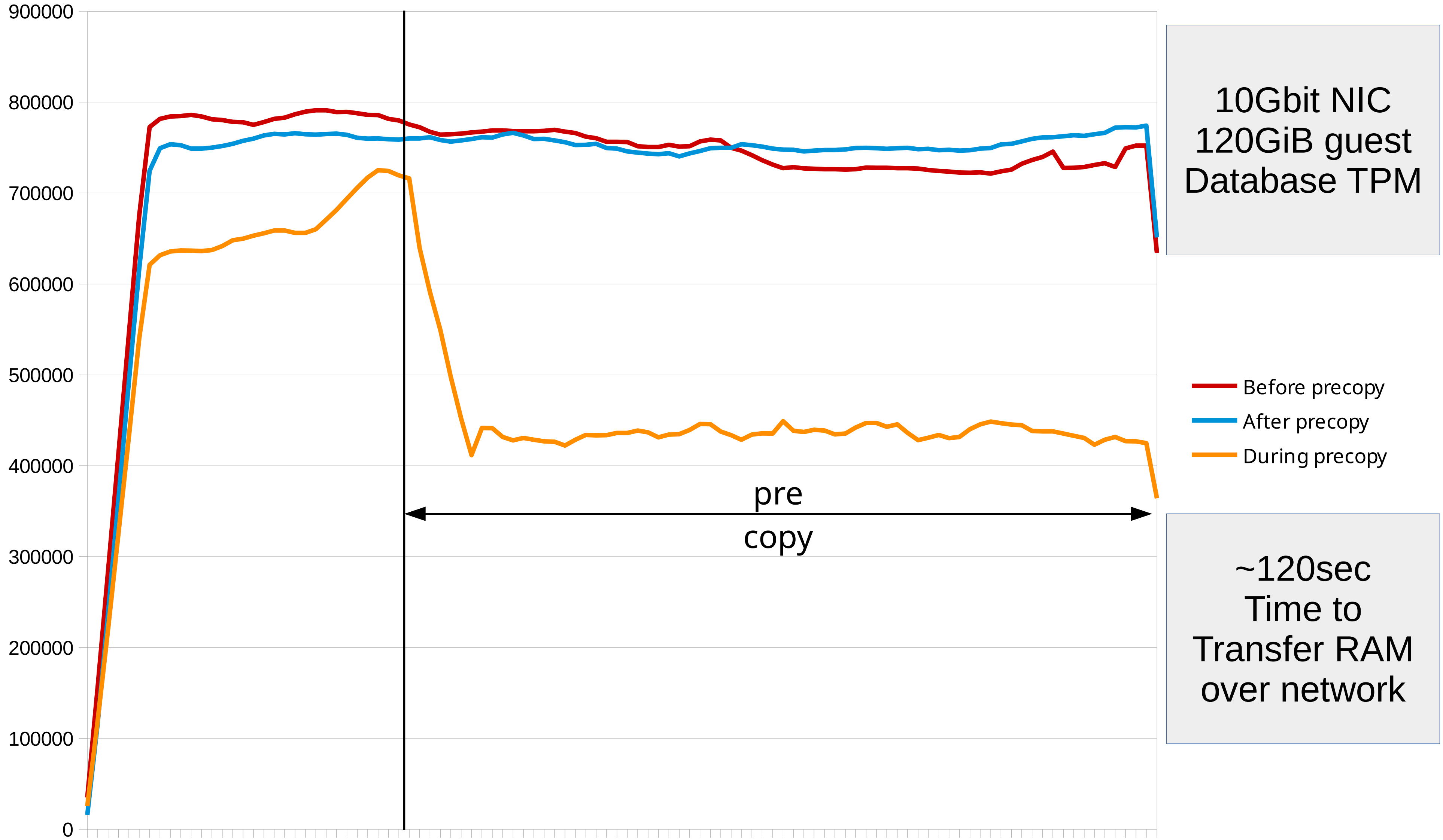
Partially funded by the Orbit *European Union* project

userfaultfd latency



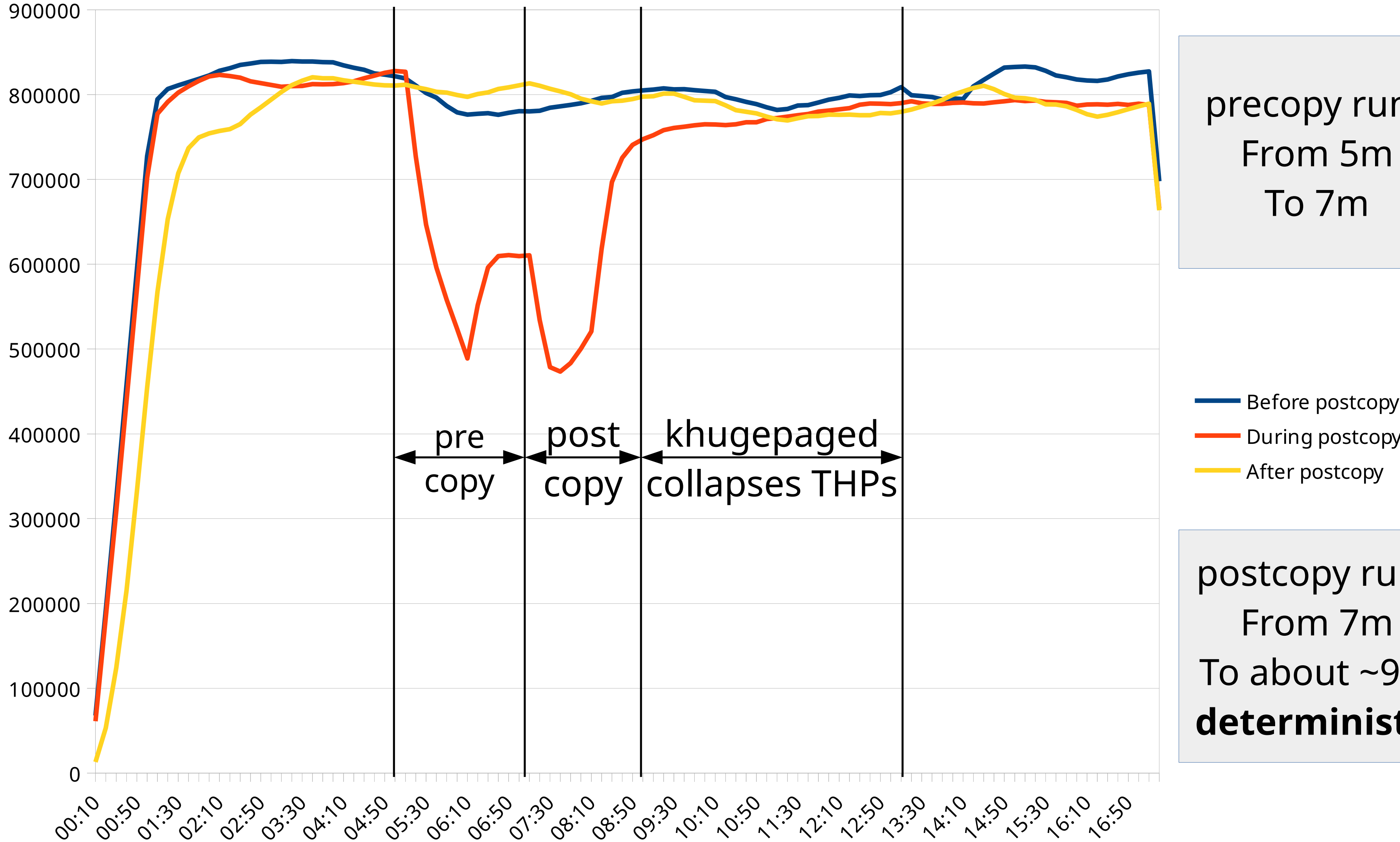
Userfaults triggered on pages that were already in network-flight are instantaneous. Background transfer seeks at the last userfault address.

KVM precopy live migration



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

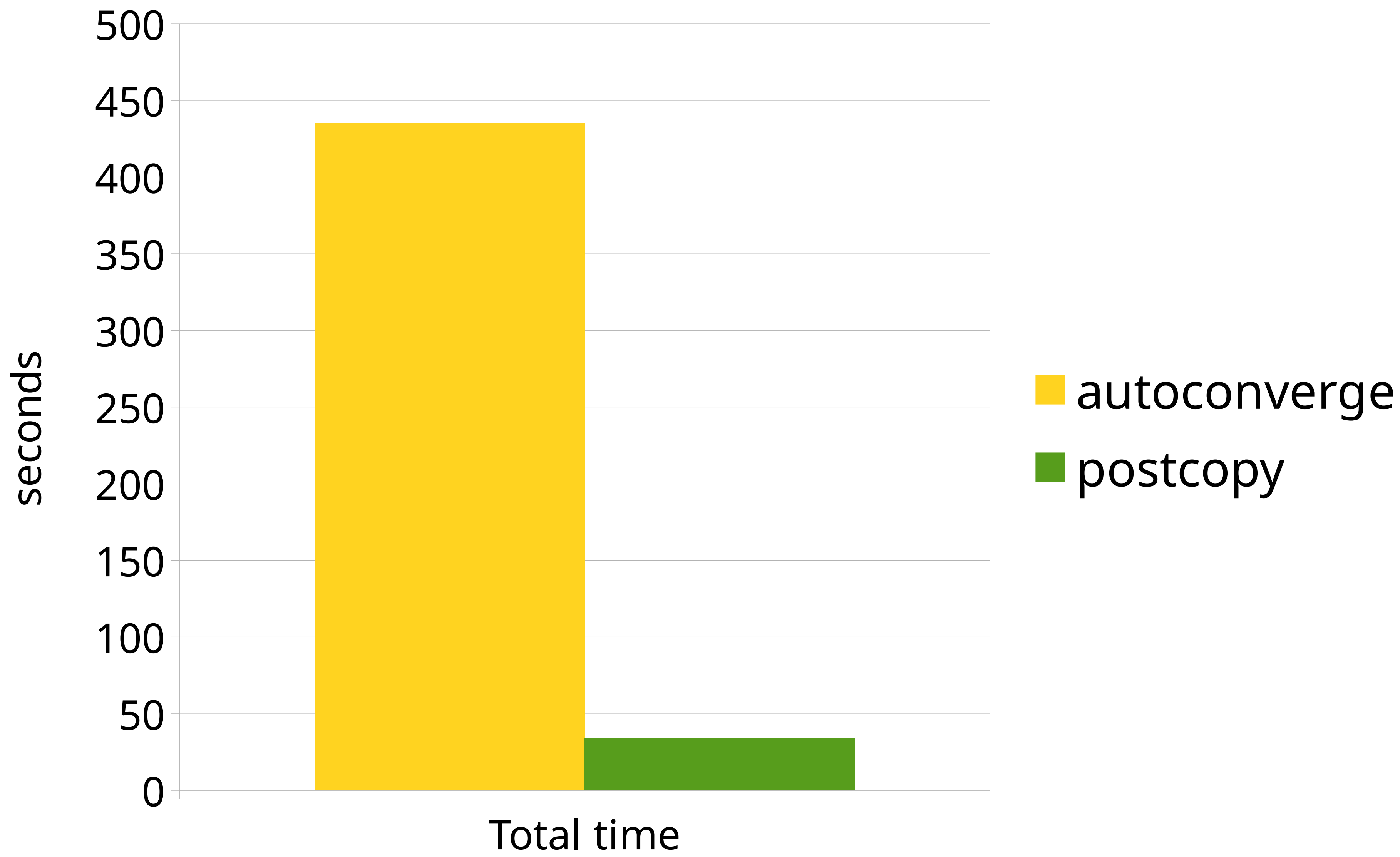
virsh migrate .. --postcopy --timeout <sec> --timeout-postcopy

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

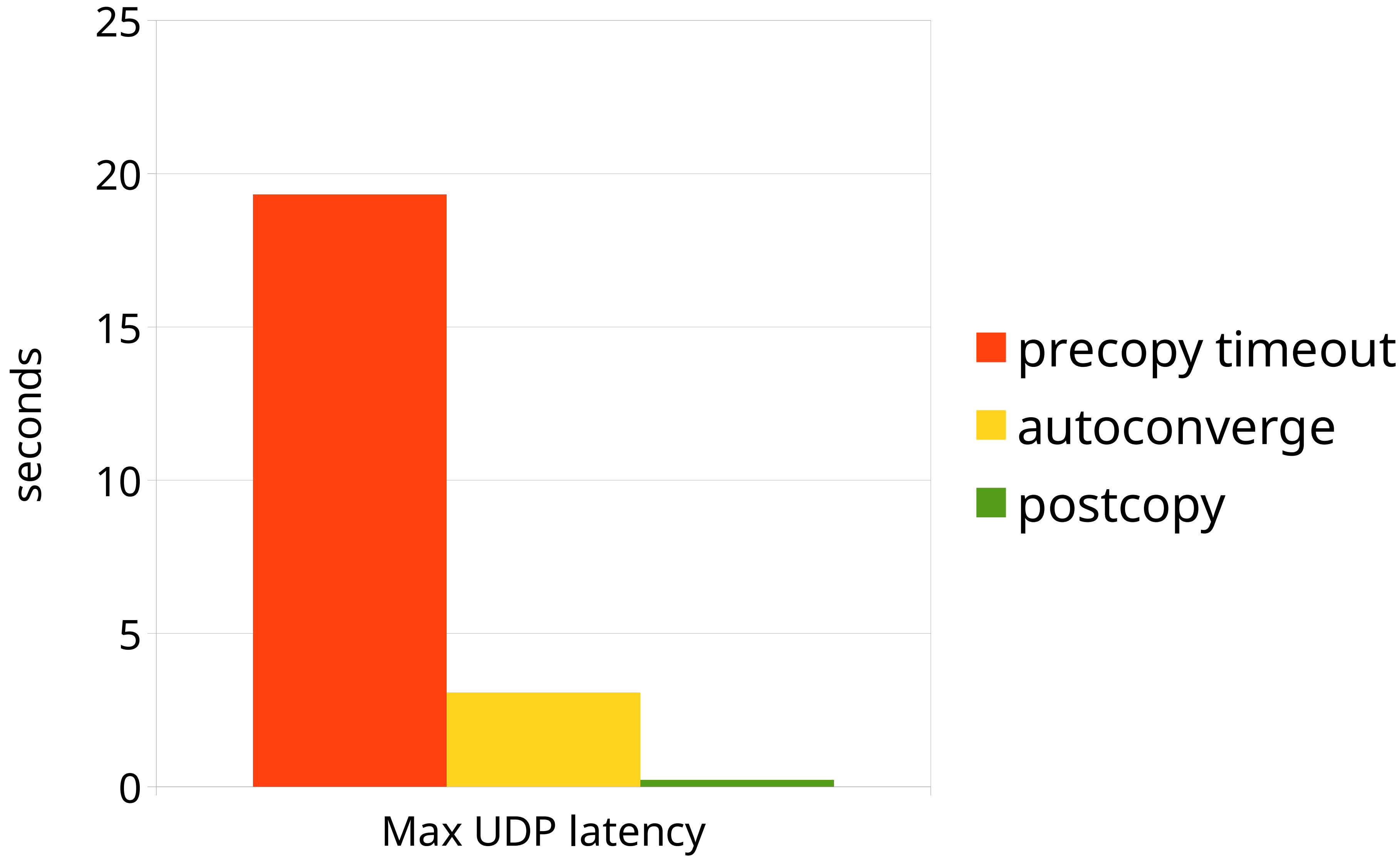
All available upstream

- 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
- In production since **RHEL 7.3**

Live migration total time



Live migration max perceived downtime latency



redhat

userfaultfd v4.13 features

- The current v4.13 upstream kernel supports:
  - **Missing** faults (i.e. missing pages **EVENT_PAGEFAULT**)
    - **Anonymous** (UFFDIO_COPY, UFFDIO_ZEROPAGE, UFFDIO_WAKE)
    - **Hugetlbfs** (UFFDIO_COPY, UFFDIO_WAKE)
    - **Shmem** (UFFDIO_COPY, UFFDIO_ZEROPAGE, UFFDIO_WAKE)
  - **UFFD_FEATURE_THREAD_ID** (to collect vcpu statistics)
  - **UFFD_FEATURE_SIGBUS** (raises SIGBUS instead of userfault)
  - Non cooperative **Events**
    - **EVENT_REMOVE** (MADV_REMOVE/DONTNEED/FREE)
    - **EVENT_UNMAP** (munmap notification to stop background transfer)
    - **EVENT_REMAP** (non-cooperative process called mremap)
    - **EVENT_FORK** (create new uffd over fork())
    - **UFFDIO_COPY** returns **-ESRCH** after exit()

userfaultfd in-progress features

- v4.13 rebased aa.git for anonymous memory supports:
  - **UFFDIO_WRITEPROTECT**
  - **UFFDIO_COPY_MODE_WP**
    - Same as UFFDIO_COPY but wrprotected
  - **UFFDIO_REMAP**
    - monitor can remove memory atomically
- Floating patchset for synchronous EVENT_REMOVE to allow multithreaded non cooperative manager

struct uffd_msg

```
/* read() structure */
struct uffd_msg {
	__u8	event;

	__u8	reserved1;
	__u16	reserved2;
	__u32	reserved3;

	union {
		struct {
			__u64	flags;
			__u64	address;
			union {
				__u32 ptid;
			} feat;
		} pagefault;

		struct {
			__u32	ufd;
		} fork;

		struct {
			__u64	from;
			__u64	to;
			__u64	len;
		} remap;

		struct {
			__u64	start;
			__u64	end;
		} remove;

		struct {
			/* unused reserved fields */
			__u64	reserved1;
			__u64	reserved2;
			__u64	reserved3;
		} reserved;
	} arg;
} __packed;
```

UFFD_EVENT_* tells which part of the union is valid

sizeof(struct uffd_msg) 32bit/64bit ABI enforcement
Zeros here, can extend with UFFD_FEATURE flags

Default cooperative support tracking pagefaults
UFFD_EVENT_PAGEFAULT

Non cooperative support tracking MM syscalls
UFFD_EVENT_FORK

UFFD_EVENT_REMAP

UFFD_EVENT_REMOVE|UNMAP

userfaultfd potential use cases

- Efficient snapshotting (drop fork())
- JIT/to-native compilers (drop write bits)
- Add robustness to hugetlbfs/tmpfs
- Host enforcement for virtualization memory ballooning
- Distributed shared memory projects: at Berkeley and University of Colorado
- Tracking of anon pages written by other threads: at llnl.gov
- Obsoletes soft-dirty
- Obsoletes volatile pages SIGBUS

Git userfaultfd branch

- https://git.kernel.org/cgit/linux/kernel/git/andrea/aa.git/log/?h=userfault

Virtual Memory evolution

- 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 **<u>entirety</u>**