Userland Page Faults and Beyond
Why How and What's Next

Red Hat, Inc.

Andrea Arcangeli <aarcange at redhat.com>

LinuxCon North America, Toronto

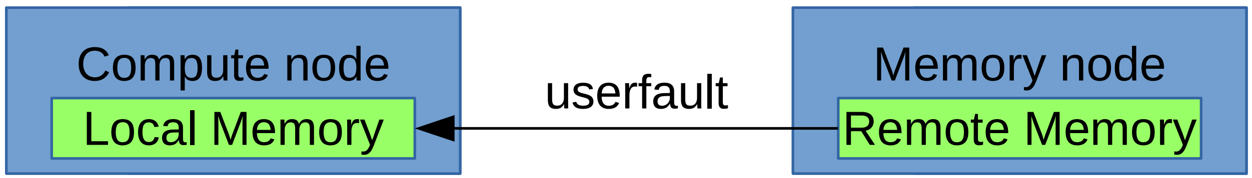
24 Aug 2016

Topics

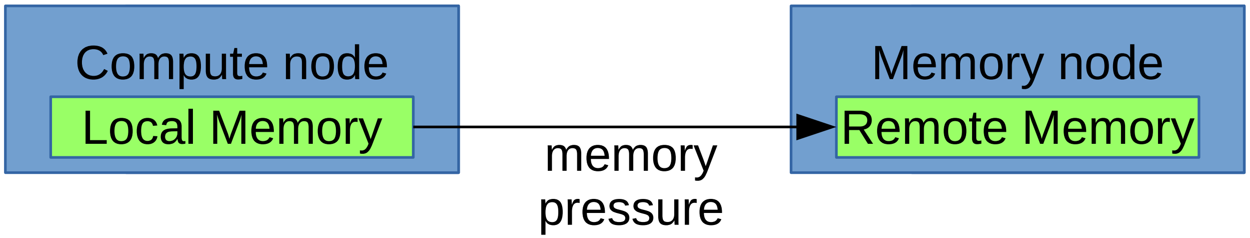
- Normally page faults are a kernel internal thing..
  - Why offload page faults to userland?
    - Initial use case that required it
- Upstream/production status
- How the userfaultfd API works
- Other use cases
- Development status
- Demo

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

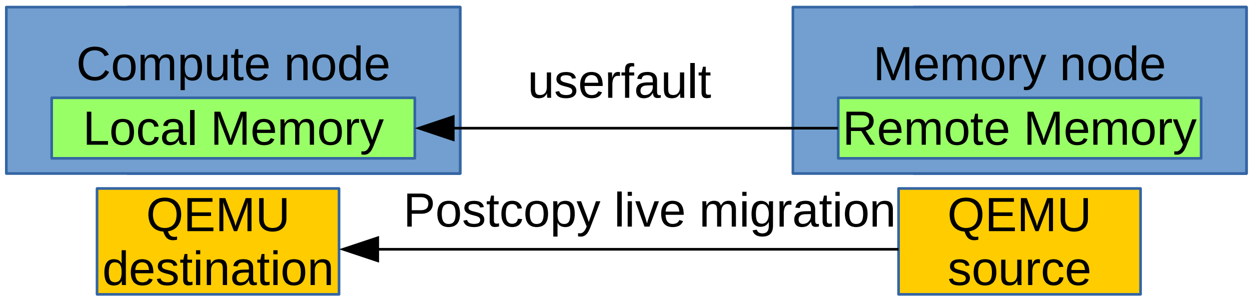


- Memory can be transferred from the compute node to the memory node if it's not frequently used during memory pressure



Postcopy live migration

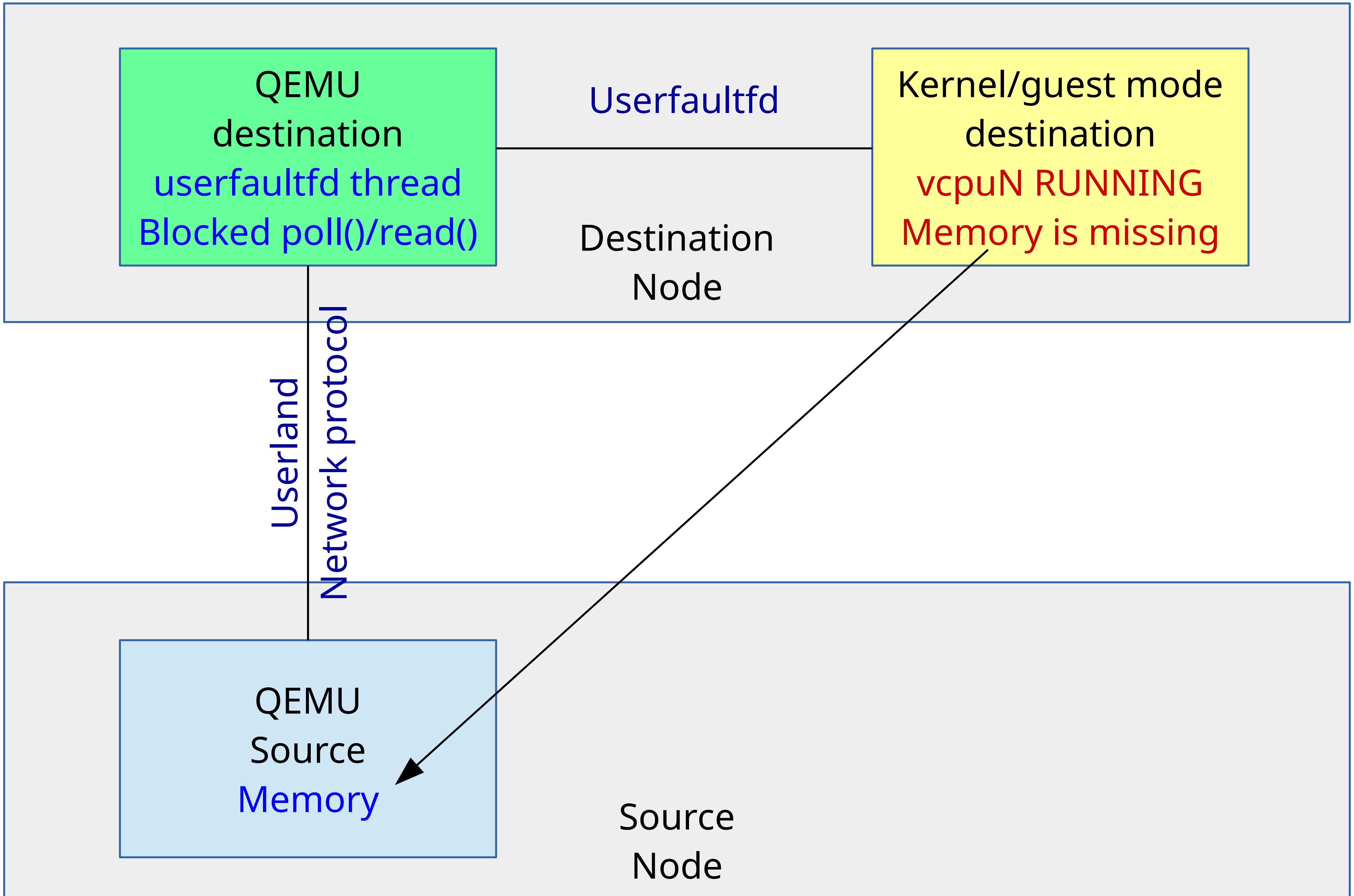
- **Postcopy live migration** is a form of memory externalization



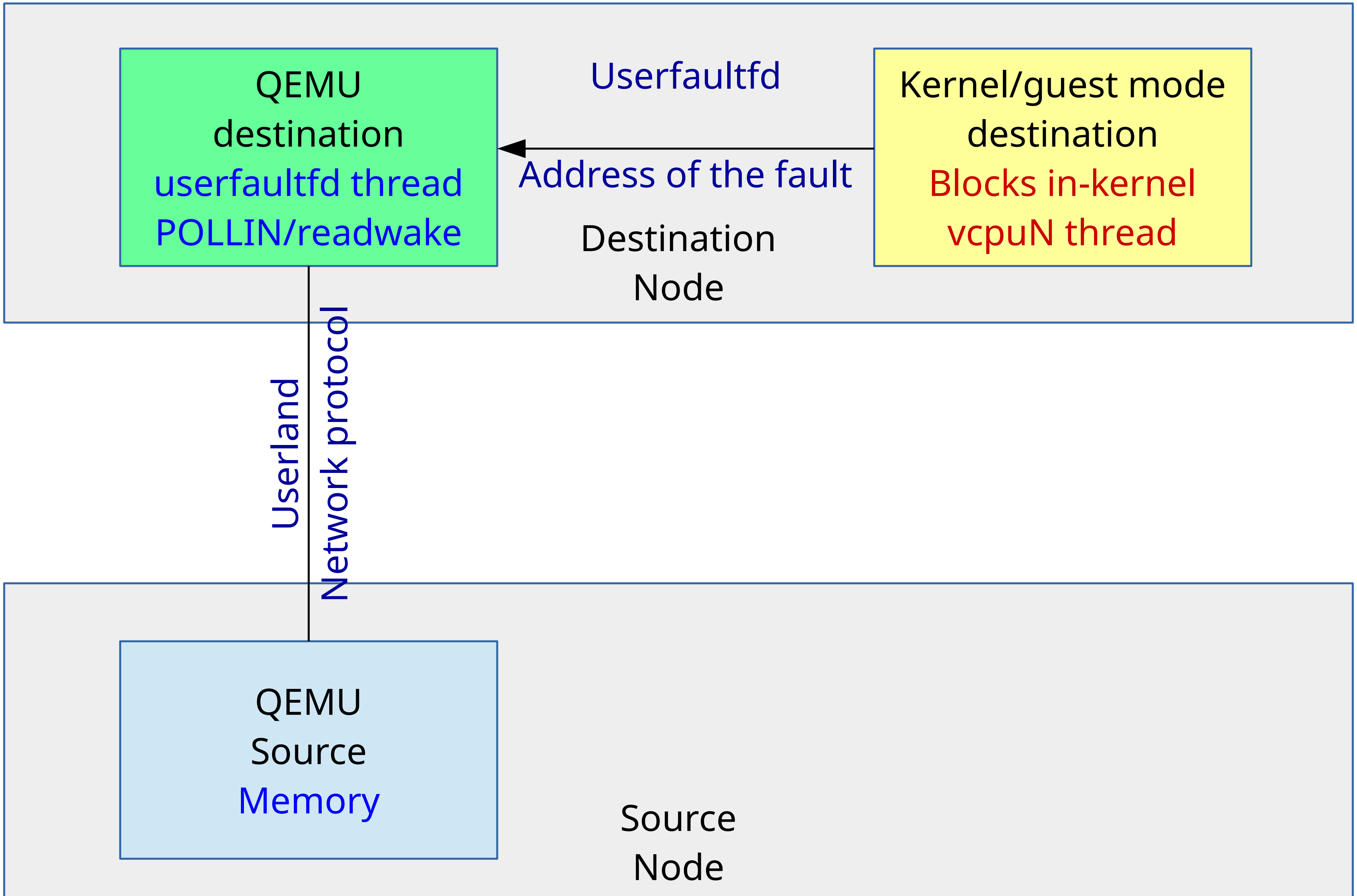
- When the QEMU compute node (destination) faults on a missing page that resides in the memory node (source) the kernel has no way to fetch the page
  - Solution: let QEMU in userland handle the pagefault

Partially funded by the Orbit *European Union* project

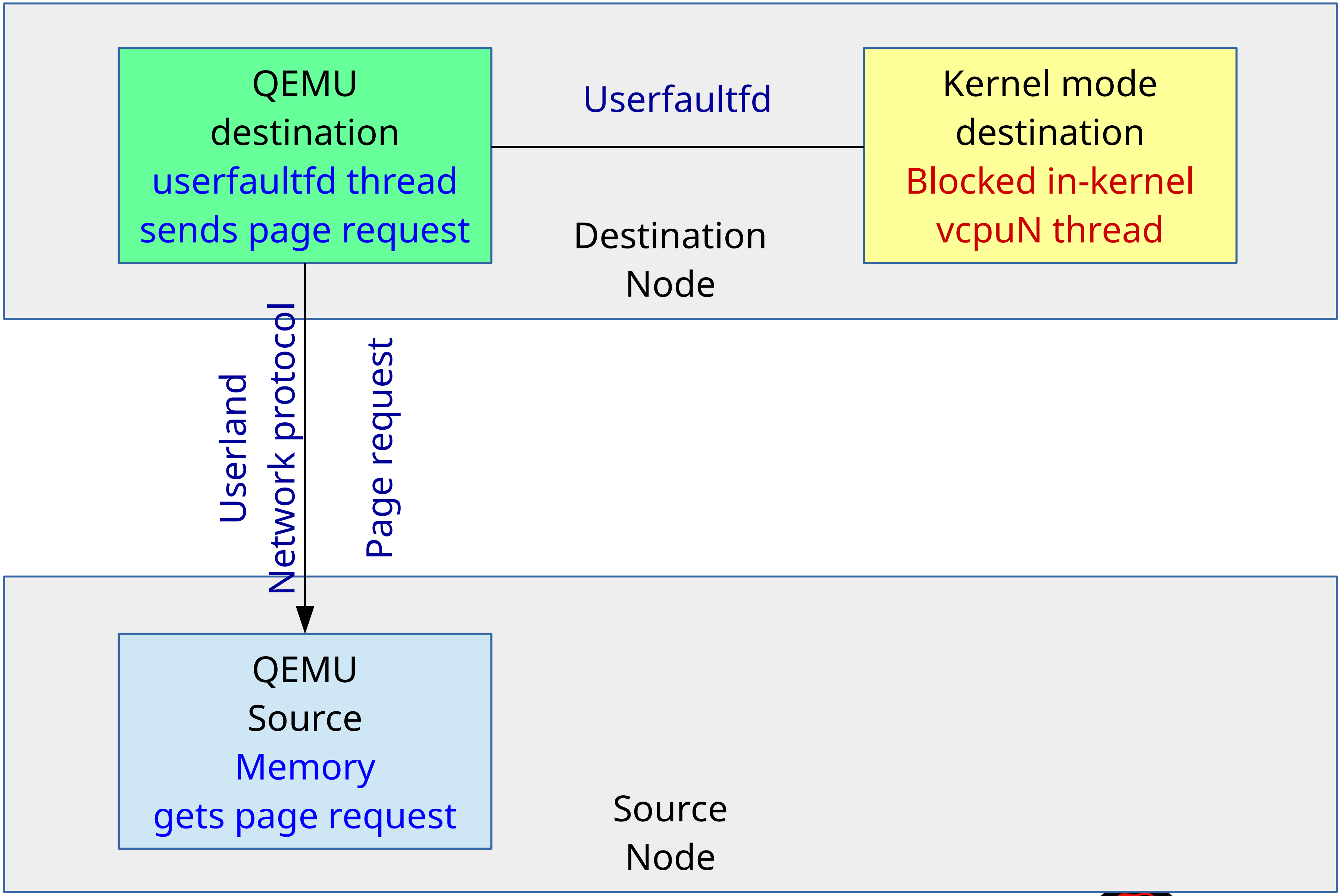
uffd postcopy live migration



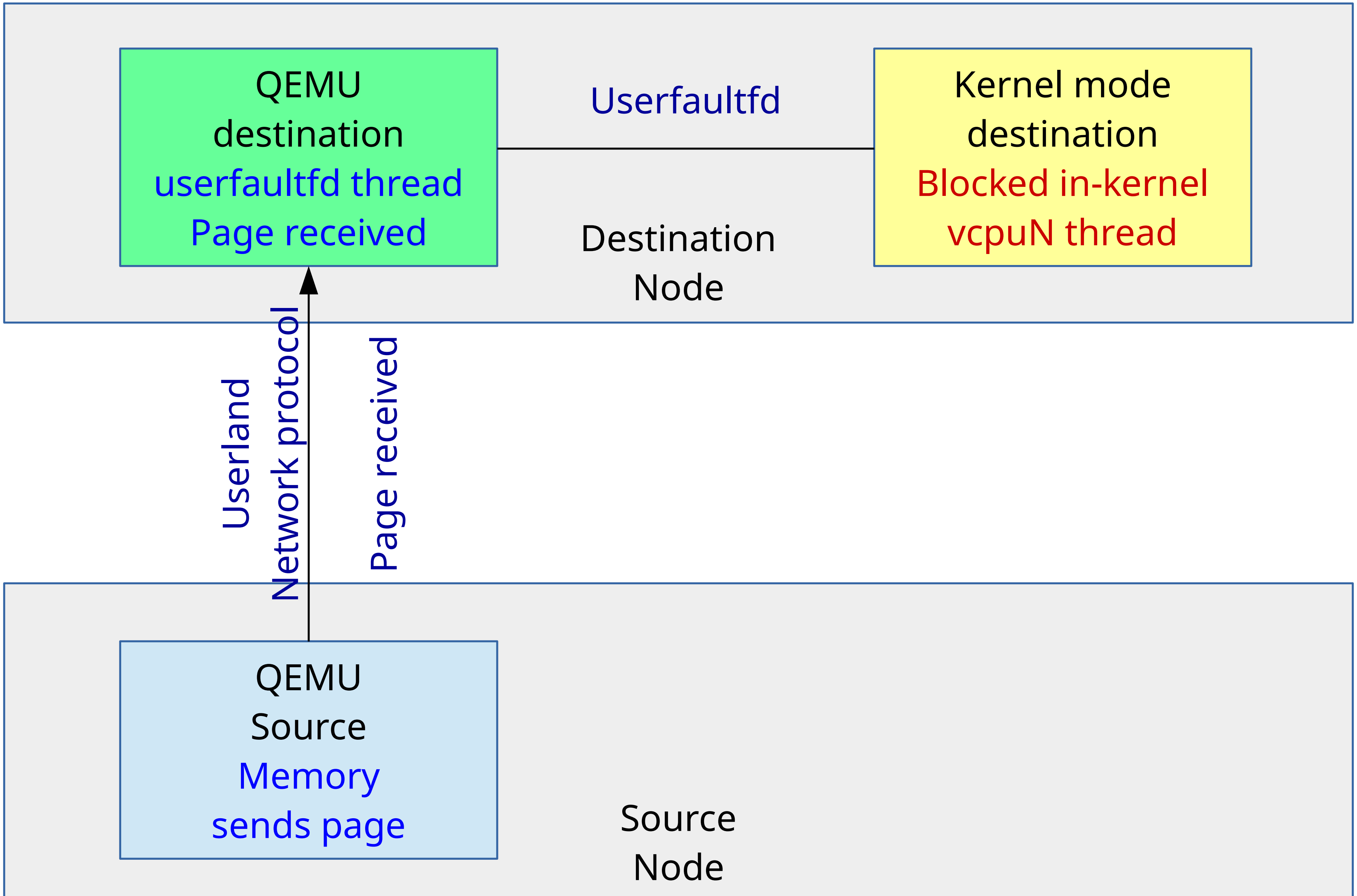
userfaultfd event notification



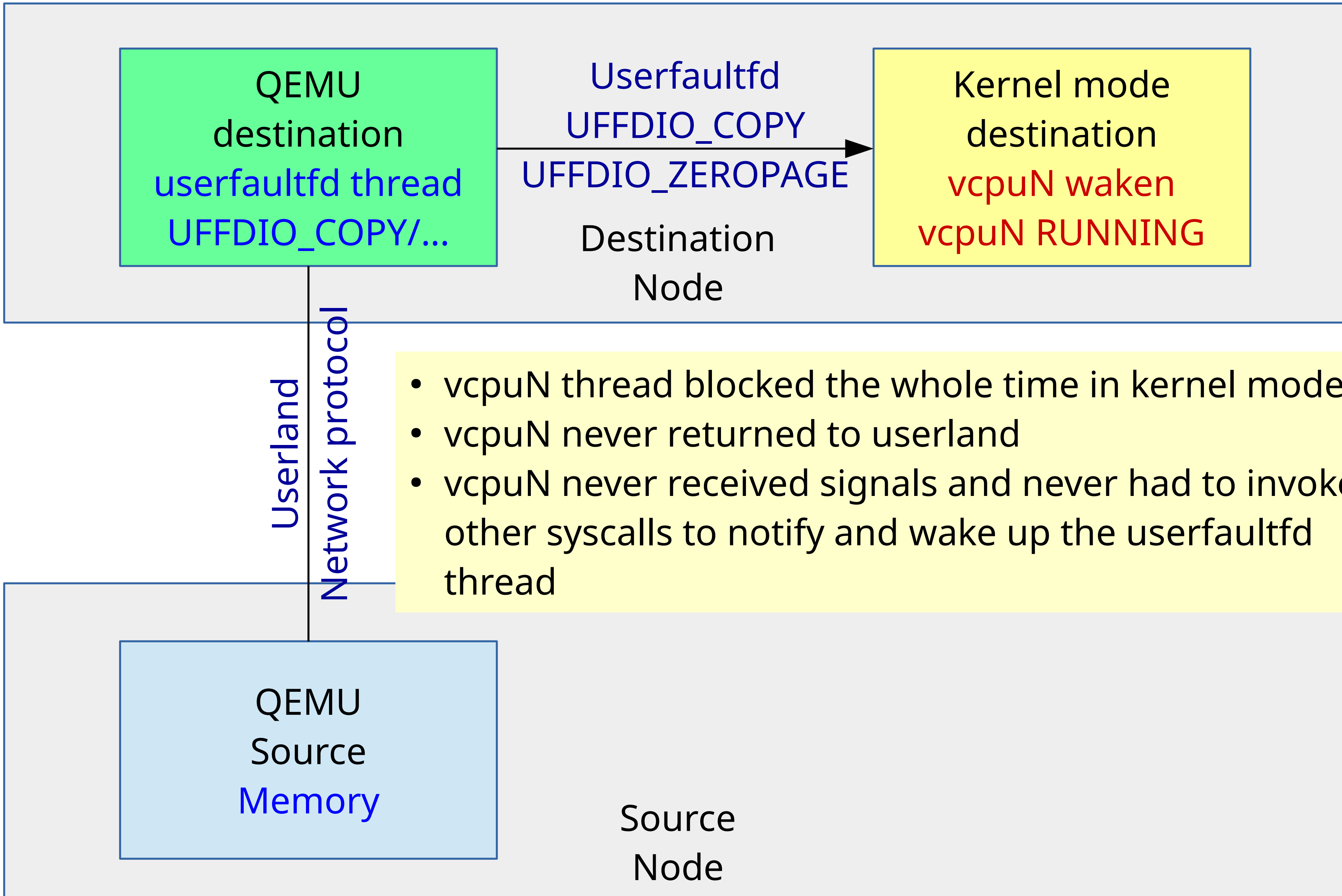
QEMU network page request



QEMU receives page



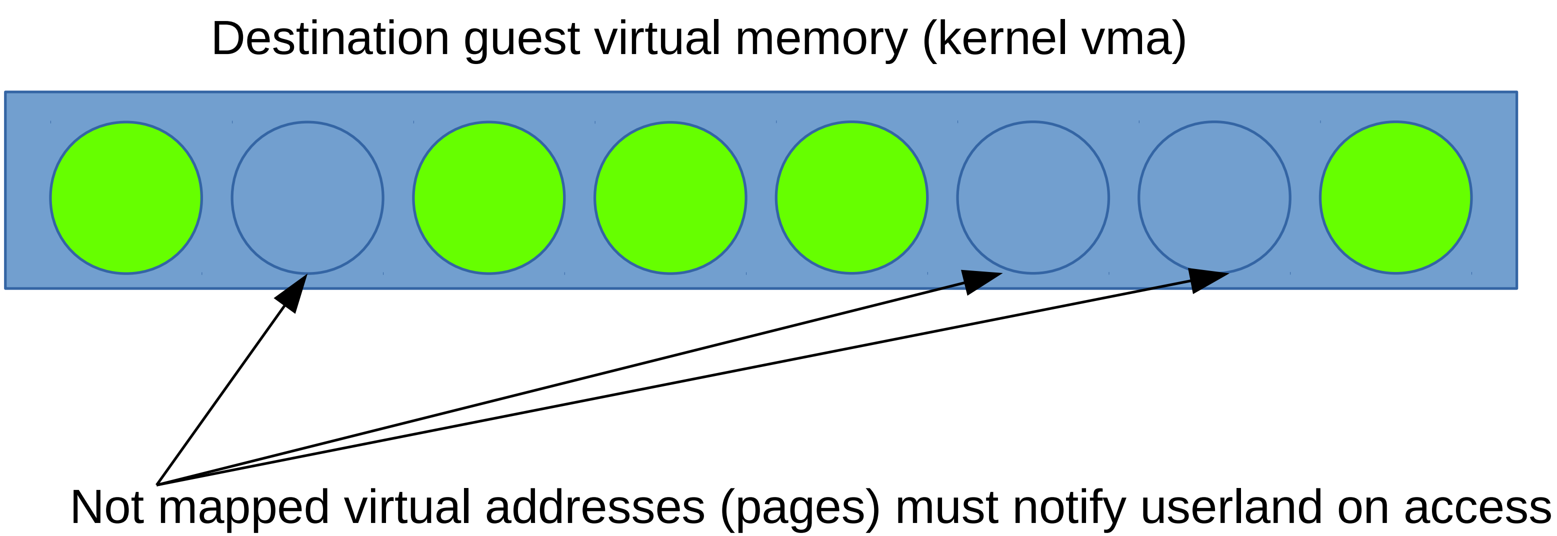
UFFDIO_ZEROPAGE/COPY



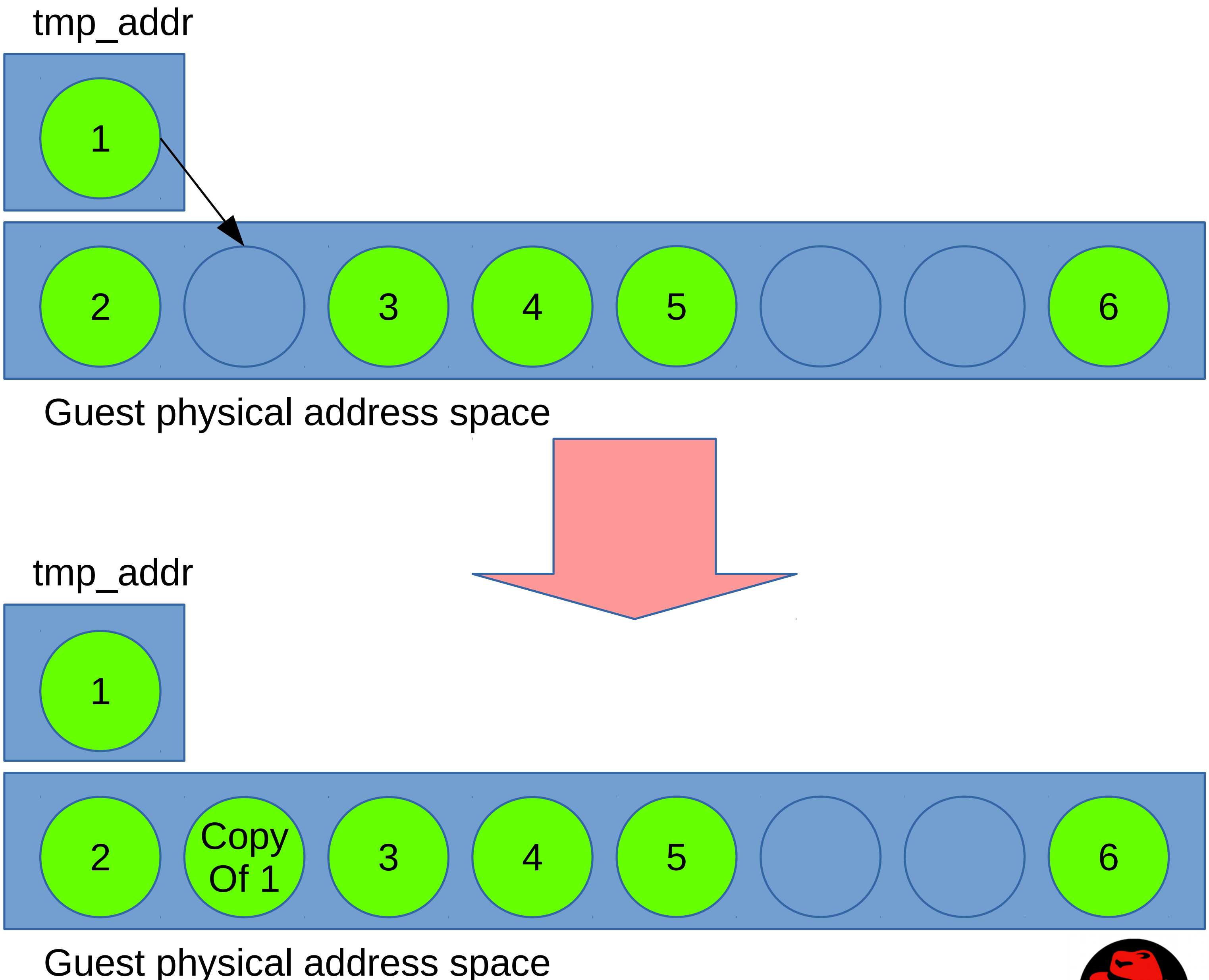
- vcpuN thread blocked the whole time in kernel mode
- vcpuN never returned to userland
- vcpuN never received signals and never had to invoke other syscalls to notify and wake up the userfaultfd thread

Missing pages notification

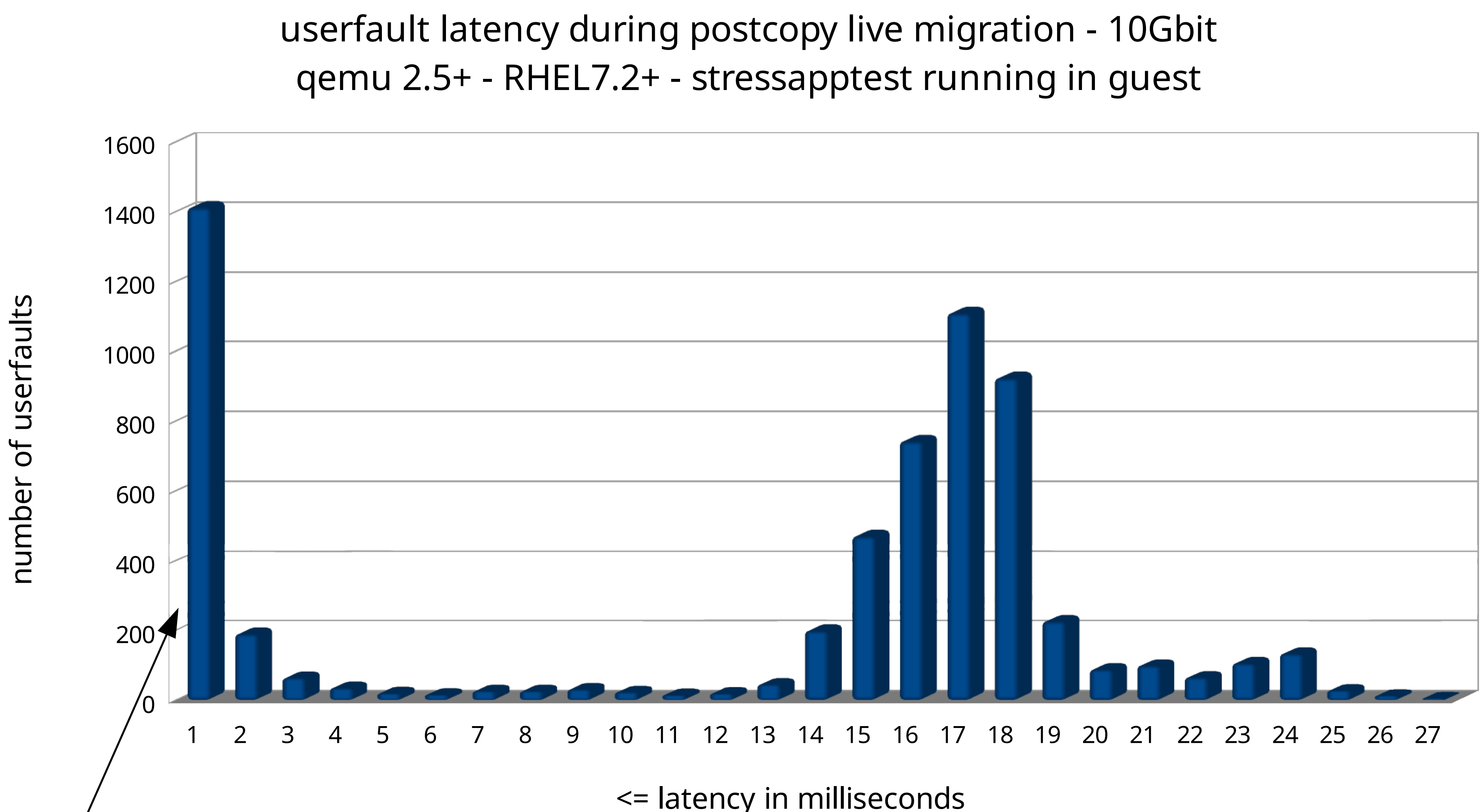
- QEMU destination running in the compute node must be notified the first time a page fault happens if a page is still missing



UFFDIO_COPY - *atomic* memcpy()

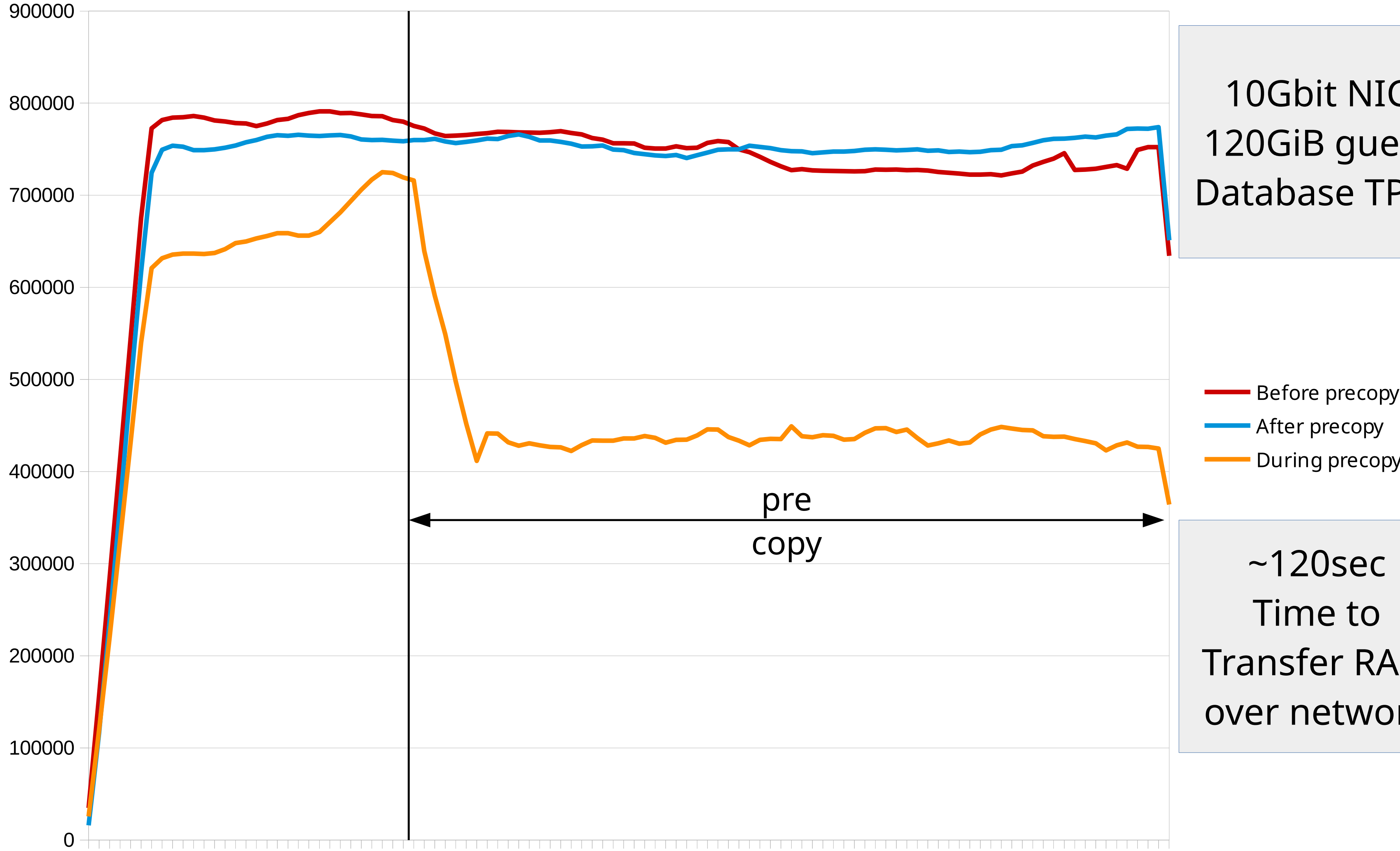


userfaultfd latency



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

KVM precopy live migration

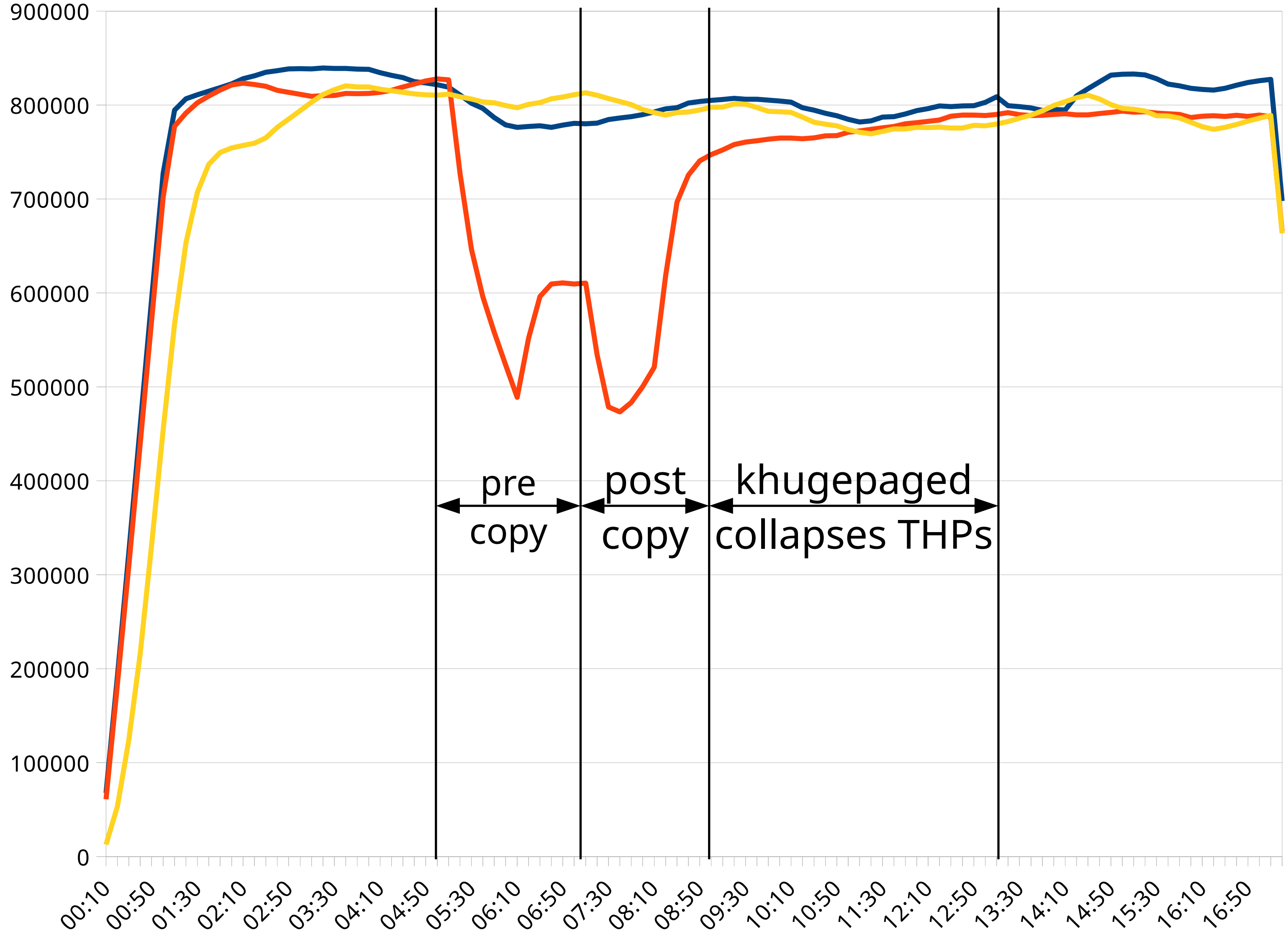


10Gbit NIC
120GiB guest
Database TPM

~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**

What's Next?

- The current upstream kernel support is limited to:
  - **Missing** faults (i.e. missing pages)
  - **Anonymous** memory (i.e. malloc)
- What about:
  - other types of memory
    - **tmpfs**
    - **hugetlbfs**
    - perhaps real **filesystem pagecache** too
  - **Write Protect faults**
  - **Removing the memory** atomically... after adding it with UFFDIO_COPY
  - sending the opened userfaultfd to a different "manager" process
    - so that it can manage the memory behinds its back in a **non-cooperative** way

userfaultfd API

- The ioctl(uffd,...) API is versioned
- Can be extended in a backwards compatible way
- The current version of the API can already provide all features mentioned in the previous slide
  - Thanks to the Linux Kernel community feedback
- Not specific to postcopy:
  - new way to manage the memory to do things that weren't possible before

UFFDIO_API

```
ufd = syscall(__NR_userfaultfd, O_CLOEXEC)

struct uffdio_api api_struct;
api_struct.api = UFFD_API;
/* = 0 → userland asks for default pagefault support
   WP support if kernel returns UFFD_FEATURE_PAGEFAULT_FLAG_WP */
api_struct.features = 0;

/* non cooperative mode */
/* api_struct.features = UFFD_FEATURE_EVENT_FORK | UFFD_FEATURE_EVENT_REMAP |
                         UFFD_FEATURE_EVENT_MADVDONTNEED; */

if (ioctl(ufd, UFFDIO_API, &api_struct)) {
   /* err */
}
ioctl_mask = (__u64)1 << _UFFDIO_REGISTER |
             (__u64)1 << _UFFDIO_UNREGISTER;
if ((api_struct.ioctls & ioctl_mask) != ioctl_mask) {
   /* err */
}
```

UFFDIO_REGISTER

```
struct uffdio_register reg_struct;

reg_struct.range.start = (uintptr_t)host_addr;
reg_struct.range.len = length;
reg_struct.mode = UFFDIO_REGISTER_MODE_MISSING;

if (ioctl(mis->userfault_fd, UFFDIO_REGISTER, &reg_struct)) {
   /* err */
}
```

Wait for event

- poll() / epoll() (/ select)

```
struct pollfd pollfd[1];
pollfd[0].fd = uffd;
pollfd[0].events = POLLIN;
ret = poll(pollfd, 1, -1);
```

- Read()

```
struct uffd_msg msg;
ret = read(uffd, &msg, sizeof(msg));
```

- Check uffd_msg event

```
if (msg.event != UFFD_EVENT_PAGEFAULT) { /* err */ }
offset = (char *)(unsigned long)msg.arg.pagefault.address -
area_dst;
```

struct uffd_msg

```
struct uffd_msg {
    __u8   event;
    __u8   reserved1;
    __u16  reserved2;
    __u32  reserved3;
    union {
        struct {
            __u64  flags;
            __u64  address;
        } pagefault;
        struct {
            __u32  ufd;
        } fork;
        struct {
            __u64  from;
            __u64  to;
            __u64  len;
        } remap;
        struct {
            __u64  start;
            __u64  end;
        } madv_dn;
        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_MADVDONTNEED

Handle fault

- UFFDIO_COPY

```
struct uffdio_copy uffdio_copy;
uffdio_copy.dst = (unsigned long) area_dst + offset;
uffdio_copy.src = (unsigned long) area_src + offset;
uffdio_copy.len = page_size;
uffdio_copy.mode = 0;
uffdio_copy.copy = 0;
if (ioctl(uffd, UFFDIO_COPY, &uffdio_copy)) {
   /* err */
}
```

- UFFDIO_ZEROPAGE

```
struct uffdio_zeropage zero_struct;
zero_struct.range.start = (uint64_t)(uintptr_t)host;
zero_struct.range.len = getpagesize();
zero_struct.mode = 0;
if (ioctl(mis->userfault_fd, UFFDIO_ZEROPAGE, &zero_struct)) {
   /* err */
}
```

Possible use cases

- Replace mprotect/mremap+SIGSEGV
- Efficient snapshotting
- JIT/to-native compilers removal of write bits
- Non cooperative usage
- CRIU lazy restore from disk
- Add robustness to shared memory
- Add host enforcement to QEMU balloon driver backend
- Distributed shared memory
- Obsoletes soft_dirty
- Obsoletes “volatile pages” SIGBUS notification

userfaultfd vs SIGSEGV

- Note: the translation is not strictly 1:1 and many more combinations are possible...

UFFDIO_REGISTER_MODE_MISSING UFFDIO_COPY UFFDIO_ZEROPAGE poll() / epoll() / blocking read()	mprotect(PROT_NONE) mprotect(PROT_READ|PROT_WRITE) SIGSEGV signal
UFFDIO_REGISTER_MODE_WP UFFDIO_WRITEPROTECT UFFDIO_WRITEPROTECT_MODE_WP poll() / epoll() / blocking read()	mprotect(PROT_READ) mprotect(PROT_READ|PROT_WRITE) SIGSEGV signal
UFFDIO_REMAP UFFDIO_* poll() / epoll() / blocking read()	mremap() SIGSEGV signal

userfaultfd vs SIGSEGV

- The poll() / epoll() / read() notification is much faster than a SIGSEGV signal
  - No signal masking complexities and inefficiencies
  - The blocked task blocks in-kernel and never returns to userland
- UFFDIO_ ioctls are faster and more SMP scalable than their syscall equivalents like mprotect/mremap
  - All locks are lightweight
  - *“vmas”* are **never** modified in fast paths
  - No risk of running out “vmas”

https://lab.nexedi.cn/kirr/wendelin.core/blob/master/bigfile/virtmem.c

Efficient Snapshotting

- Huawei working on QEMU/KVM postcopy live snapshotting
- Redis can use it too
- No need of fork()
  - Use threads instead
- COW faults can be throttled
  - COW faults after fork() cannot be throttled
- THP always enabled will run optimally
  - It is userland that decides the granularity of the fault so if it wants to COW at 4KiB granularity it can
  - THP becomes more “transparent” than it already was
- Requires WP support

https://lists.nongnu.org/archive/html/qemu-devel/2016-01/msg00664.html

http://redis.io/topics/latency

Optimize away JIT/to-native compilers write bits

- JIT/to-native compilers always set a bit to track which “Cards”/pages were modified
- WP support can allow the garbage collector to track which pages were modified to avoid collecting write bits
- Similar to dirty logging mode in QEMU/KVM
- IIRC this was one the targeted optimizations of the OSv unikernel
  - userfaultfd ***might*** achieve the equivalent result but without linking the JVM run time in the kernel
- Requires WP support

https://medium.com/@MartinCracauer/generational-garbage-collection-write-barriers-write-protection-and-userfaultfd-2-8b0e796b8f7f

http://jcdav.is/2015/11/09/More-JVM-Signal-Tricks/

Non cooperative usage

- Virtuozzo and IBM working on it
- QEMU is aware of the uffd, the guest apps are not
- The app sends a “uffd” to a “manager” and then it keeps running unaware the “manager” is managing the memory behinds its back
- Container postcopy live migration
- fork()/mremap() and other VM syscalls must send userfaultfd events
  - Not only page faults will send events to userland
- Nesting of uffds is going to be *“interesting”*

http://www.slideshare.net/kerneltlv/userfaultfd-and-postcopy-migration

CRIU lazy restore from disk

- Start the restored task immediately
  - Despite the load of its memory from disk isn't complete
- Conceptually similar to postcopy live migration
  - The program memory is read from disk instead of being transferred through the network

https://criu.org/Userfaultfd

Add robustness to shmem

- Oracle contributed the UFFDIO_MISSING support to hugetlbfs
- IBM contributed the UFFDIO_MISSING support to tmpfs
- If the database hits a bug and accidentally writes a byte in a shmem file hole, a new page is ***silently*** allocated and the corruption goes unnoticed
- An accidental read of zeros from a shmem file hole would also go unnoticed
- userfaultfd **at zero cost** notifies the database that something unexpectedly tried to read or write to a hugetlbfs or tmpfs file hole
  - “no vma split” and UFFDIO_COPY are the main feature here
  - For this use case if an userfaultfd event ever happens, it is sign a bug is causing memory corruption

Host enforcement for memory ballooning

- The QEMU/KVM memory balloon driver is not host enforced
- After the host backend driver calls MADV_DONTNEED, the guest can still deflate the memory balloon
- userfaultfd UFFDIO_MISSING support, at **zero cost**, can allow QEMU/KVM to notice the guest is malicious and terminate it

Distributed shared memory

- Distributed Shared Memory project at Berkeley
- Other research interest
- Needs **UFFDIO_REMAP** to extract memory
  - mremap() equivalent
- Needs WP support to allow read-shared, write-exclusive model
- Can interact with HMM (Heterogeneous Memory Management) and NVIDIA's unified memory
  - GPU userfaults?

redhat

Obsoletes soft_dirty

- Soft dirty tells userland, which pages were modified during a certain runtime
- It has to scan all pagetables of all memory that could possibly have been modified to find out
  - O(N) complexity where N is the number of pages
  - It still requires an initial wrprotect fault
- WP support is likely to simply obsolete soft_dirty because it will still scale well with an unlimited amount of memory
  - Similar to Intel VMM PML hw feature (Page Modification Logging)
  - Main drawback is that userfaultfd blocks the WP fault
  - It should be possible to add an event async queue model to turn on and off at run time with an UFFDIO_ASYNC ioctl, to prevent the wrprotect fault to block

Obsoletes volatile pages SIGBUS

- *“Volatile pages”* may be reclaimed and freed by the Linux Virtual Memory at any time
  - They could contain uncompressed graphic bitmaps that can be re-created at low cost
    - Worth to keep in memory only as long as there is no Virtual Memory pressure
  - A prominent *“volatile pages”* use case is Android
- userfaultfd can tell userland if the volatile page is missing
- No need of their own missing *“volatile page”* SIGBUS framework to notify userland

Development status

- **tmpfs, hugetlbfs support works** (selftests provided for both)
- **Write Protect support works**
  - Needs accuracy with ***swapping***, blocker but in progress..
- WP|MISSING support works but it may need some extension
  - UFFDIO_COPY will always add the page read-write
  - Zeropages created by UFFDIO_ZEROCOPY remains readonly forever
- **non cooperative support works**
  - Minor issues:
    - Cannot recreate shared anon pages
    - Cannot handle uffd nesting
- **UFFDIO_REMAP is implemented** but not in aa.git and shall be revisited
  - Only distributed shared memory needs it

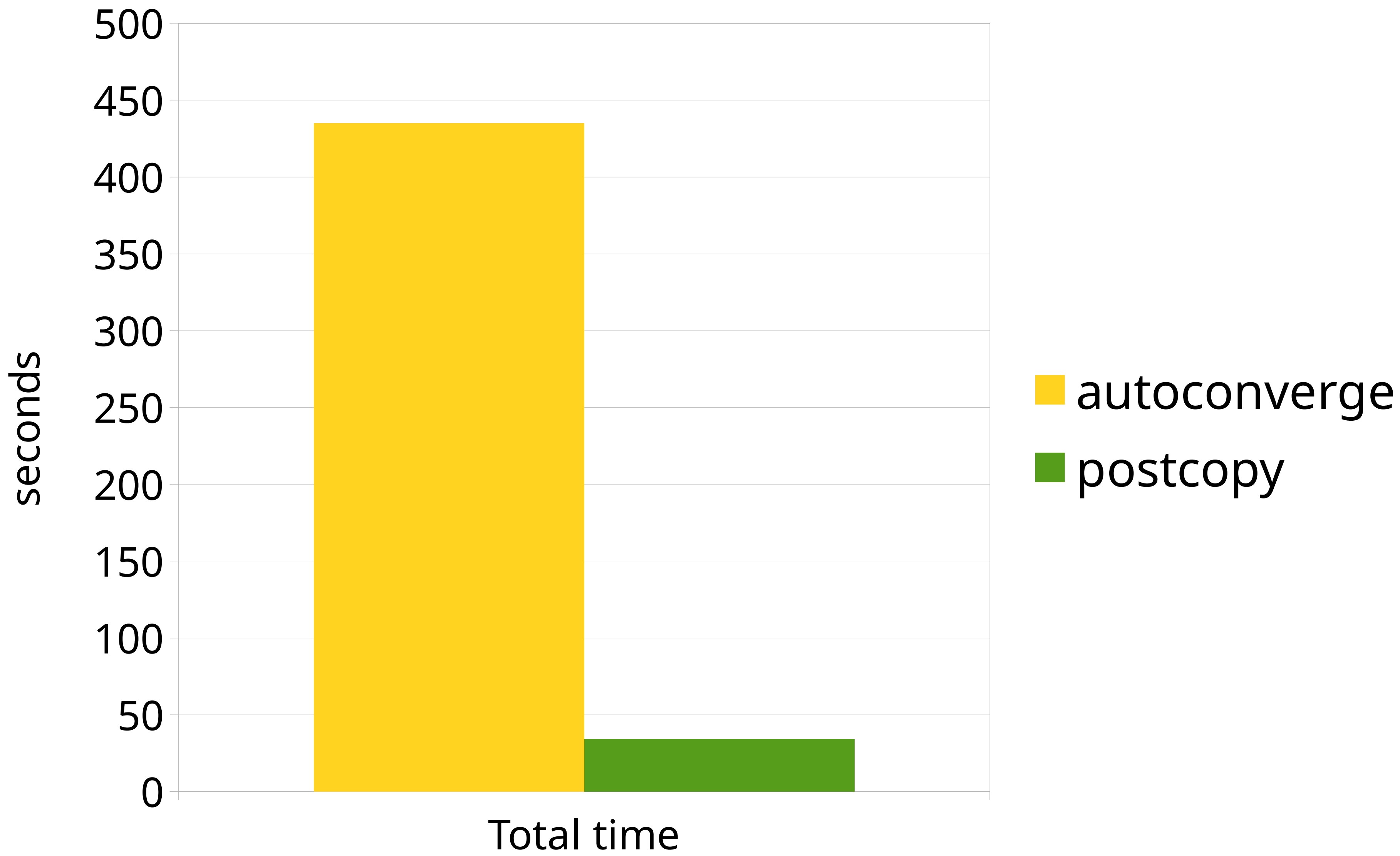
Git userfault branch

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

userfaultfd in action

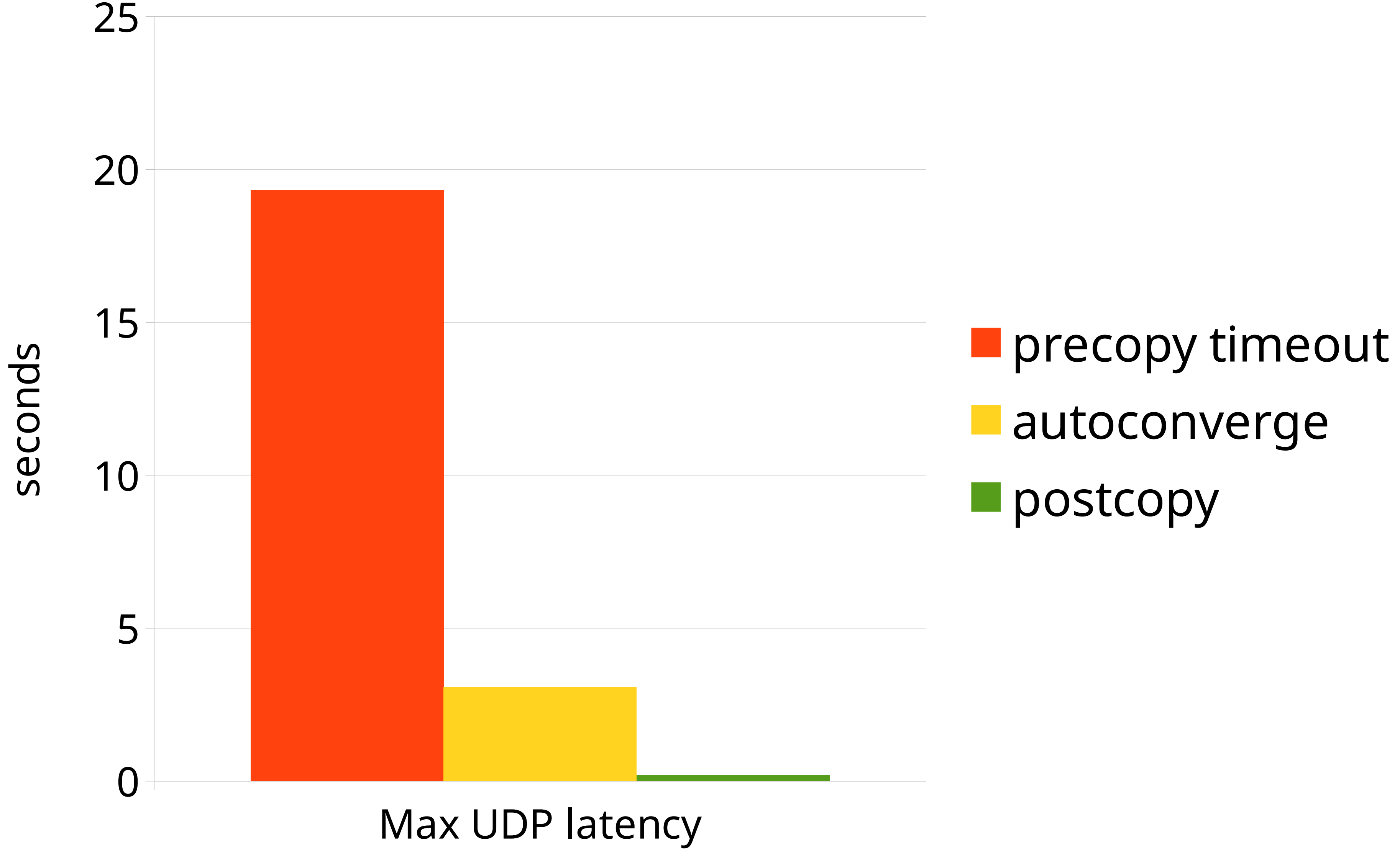
- Time for a live migration demo!

Live migration total time



redhat.

Live migration max perceived downtime latency



Q/A