trace-cmd virt-server

Tracing your virtual machines

Steven Rostedt
rostedt@goodmis.org
srostedt@redhat.com

ftrace - review

- The official tracer of the Linux kernel
- Located in the tracefs/debugfs directory
  - /sys/kernel/debug/tracing
  - /sys/kernel/tracing
- Access and controlled via normal shell commands
  - cat, echo

ftrace - review

- Traces events
  - schedule context changes
  - interrupts
  - IO
  - etc
- Traces functions
- “plugin” tracers for special tracing
  - interrupt / preemption latency
  - schedule wake up latency
  - etc

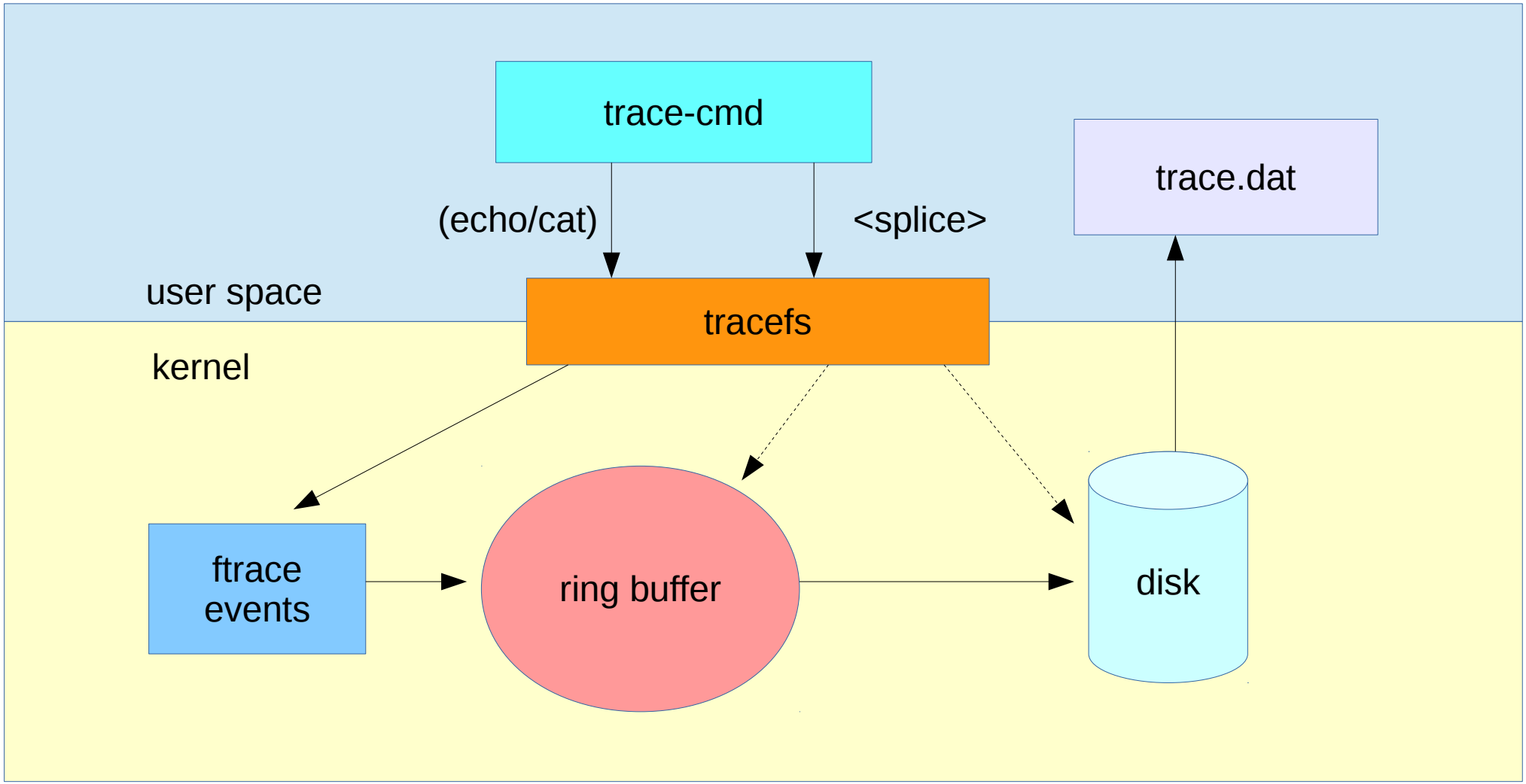
ftrace - review

```
# ls /sys/kernel/tracing
available_events             max_graph_depth      stack_trace
available_filter_functions   options              stack_trace_filter
available_tracers            per_cpu              trace
buffer_size_kb               printk_formats       trace_clock
buffer_total_size_kb         README               trace_marker
current_tracer               saved_cmdlines       trace_options
dyn_ftrace_total_info        saved_cmdlines_size  trace_pipe
enabled_functions            set_event            trace_stat
enum_map                     set_ftrace_filter    tracing_cpumask
events                       set_ftrace_notrace   tracing_max_latency
free_buffer                  set_ftrace_pid       tracing_on
function_profile_enabled     set_graph_function   tracing_thresh
instances                    set_graph_notrace    uprobe_events
kprobe_events                snapshot             uprobe_profile
kprobe_profile               stack_max_size
```

trace-cmd - review

- **Command Line Interface for ftrace**
- **Automatically mounts tracefs file system**
- **Can read the binary data directly**
  - **Uses splice to directly write from kernel buffer to file**
- **You don't need to worry about the tracefs files!**

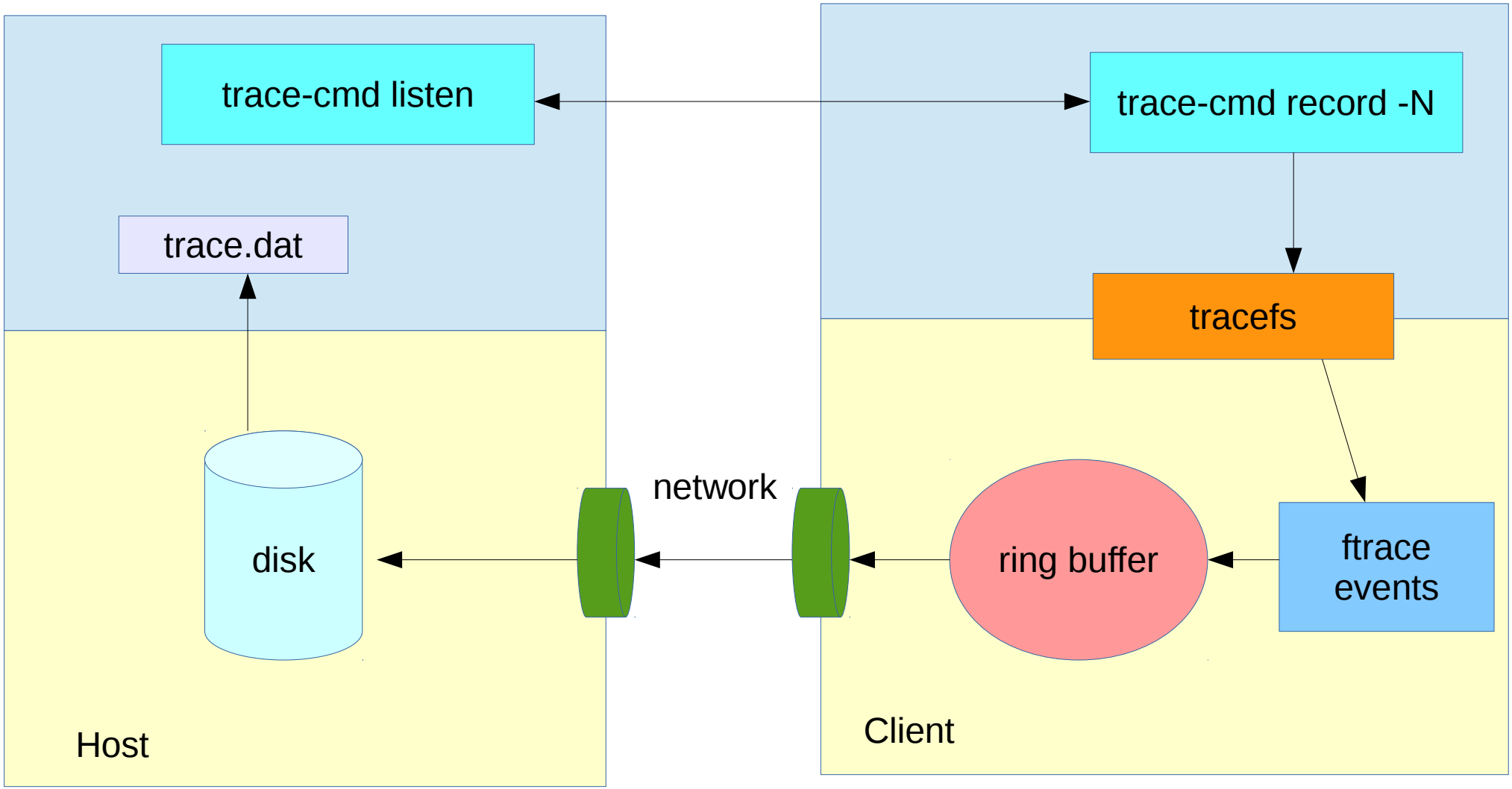
trace-cmd / ftrace flow



trace-cmd listen

- A daemon to read traces from other machines
- Can use either TCP or UDP
  - UDP is faster, but may lose events without notice
  - TCP is the same as file system, but guaranteed
- Can read from multiple computers
- trace-cmd record -N <host>:<port>
  - --date will map a “time of day” timestamp
    - used with multiple machines to sync events

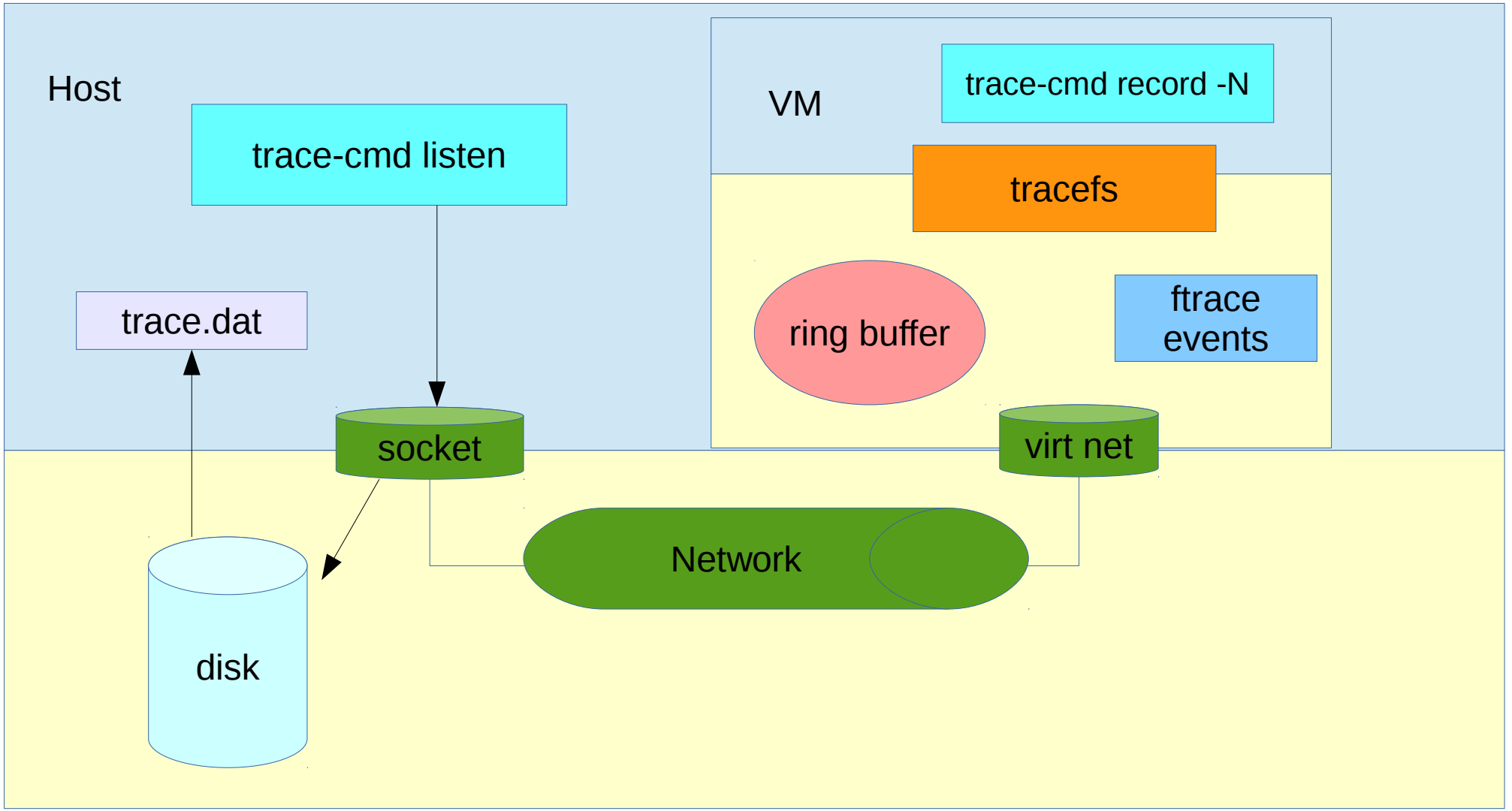
trace-cmd listen / ftrace flow



trace-cmd listen

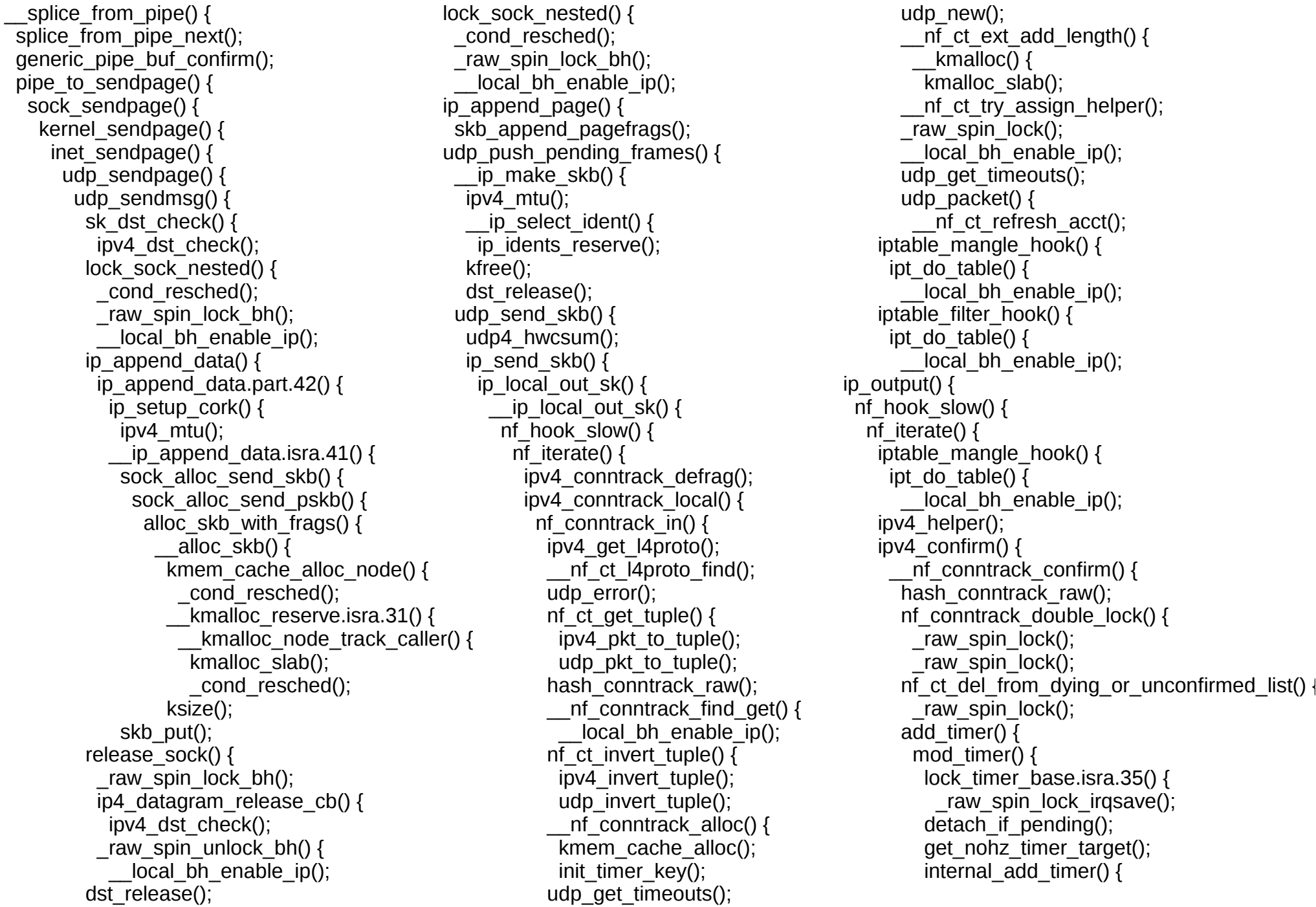
- Used to send trace data to a server
- Uses the network
- Can also work for virtual machines

trace-cmd listen / ftrace flow



trace-cmd listen

- Goes via the network
- Added networking headers
  - UDP
  - IP
- Should be a better way


__splice_from_pipe() {
splice_from_pipe_next();
generic_pipe_buf_confirm();
pipe_to_sendpage() {
sock_sendpage() {
kernel_sendpage() {
inet_sendpage() {
udp_sendpage() {
udp_sendmsg() {
sk_dst_check() {
ipv4_dst_check();
lock_sock_nested() {
_cond_resched();
_raw_spin_lock_bh();
__local_bh_enable_ip();
ip_append_data() {
ip_append_data.part.42() {
ip_setup_cork() {
ipv4_mtu();
__ip_append_data.isra.41() {
sock_alloc_send_skb() {
sock_alloc_send_pskb() {
alloc_skb_with_frags() {
__alloc_skb() {
kmem_cache_alloc_node() {
_cond_resched();
__kmalloc_reserve.isra.31() {
__kmalloc_node_track_caller() {
kmalloc_slab();
_cond_resched();
ksize();
skb_put();
release_sock() {
_raw_spin_lock_bh();
ip4_datagram_release_cb() {
ipv4_dst_check();
_raw_spin_unlock_bh() {
__local_bh_enable_ip();
dst_release();
lock_sock_nested() {
_cond_resched();
_raw_spin_lock_bh();
__local_bh_enable_ip();
ip_append_page() {
skb_append_pagefrags();
udp_push_pending_frames() {
__ip_make_skb() {
ipv4_mtu();
__ip_select_ident() {
ip_idents_reserve();
kfree();
dst_release();
udp_send_skb() {
udp4_hwcsum();
ip_send_skb() {
ip_local_out_sk() {
__ip_local_out_sk() {
nf_hook_slow() {
nf_iterate() {
ipv4_conntrack_defrag();
ipv4_conntrack_local() {
nf_conntrack_in() {
ipv4_get_l4proto();
__nf_ct_l4proto_find();
udp_error();
nf_ct_get_tuple() {
ipv4_pkt_to_tuple();
udp_pkt_to_tuple();
hash_conntrack_raw();
__nf_conntrack_find_get() {
__local_bh_enable_ip();
nf_ct_invert_tuple() {
ipv4_invert_tuple();
udp_invert_tuple();
__nf_conntrack_alloc() {
kmem_cache_alloc();
init_timer_key();
udp_get_timeouts();
udp_new();
__nf_ct_ext_add_length() {
__kmalloc() {
kmalloc_slab();
__nf_ct_try_assign_helper();
_raw_spin_lock();
__local_bh_enable_ip();
udp_get_timeouts();
udp_packet() {
__nf_ct_refresh_acct();
iptable_mangle_hook() {
ipt_do_table() {
__local_bh_enable_ip();
iptable_filter_hook() {
ipt_do_table() {
__local_bh_enable_ip();
ip_output() {
nf_hook_slow() {
nf_iterate() {
iptable_mangle_hook() {
ipt_do_table() {
__local_bh_enable_ip();
ipv4_helper();
ipv4_confirm() {
__nf_conntrack_confirm() {
hash_conntrack_raw();
nf_conntrack_double_lock() {
_raw_spin_lock();
_raw_spin_lock();
nf_ct_del_from_dying_or_unconfirmed_list() {
_raw_spin_lock();
add_timer() {
mod_timer() {
lock_timer_base.isra.35() {
_raw_spin_lock_irqsave();
detach_if_pending();
get_nohz_timer_target();
internal_add_timer() {

```
            __internal_add_timer();
            wake_up_nohz_cpu();
          _raw_spin_unlock_irqrestore();
        __nf_conntrack_hash_insert();
        __local_bh_enable_ip();
      nf_ct_deliver_cached_events();
ip_finish_output() {
  ipv4_mtu();
  ip_finish_output2() {
    neigh_resolve_output() {
      eth_header() {
        skb_push();
      dev_queue_xmit_sk() {
        __dev_queue_xmit() {
          dst_release();
          netdev_pick_tx();
          _raw_spin_lock();
          sch_direct_xmit() {
            validate_xmit_skb_list() {
              validate_xmit_skb.isra.103.part.104() {
                netif_skb_features() {
                  skb_network_protocol();
            _raw_spin_lock();
            dev_hard_start_xmit() {
              skb_clone() {
                kmem_cache_alloc();
                __skb_clone() {
                  __copy_skb_header();
              ktime_get_with_offset();
              packet_rcv() {
                skb_push();
                __bpf_prog_run();
                consume_skb() {
                  skb_release_all() {
                    skb_release_head_state();
                    skb_release_data();
                  kfree_skbmem() {
                    kmem_cache_free();
```

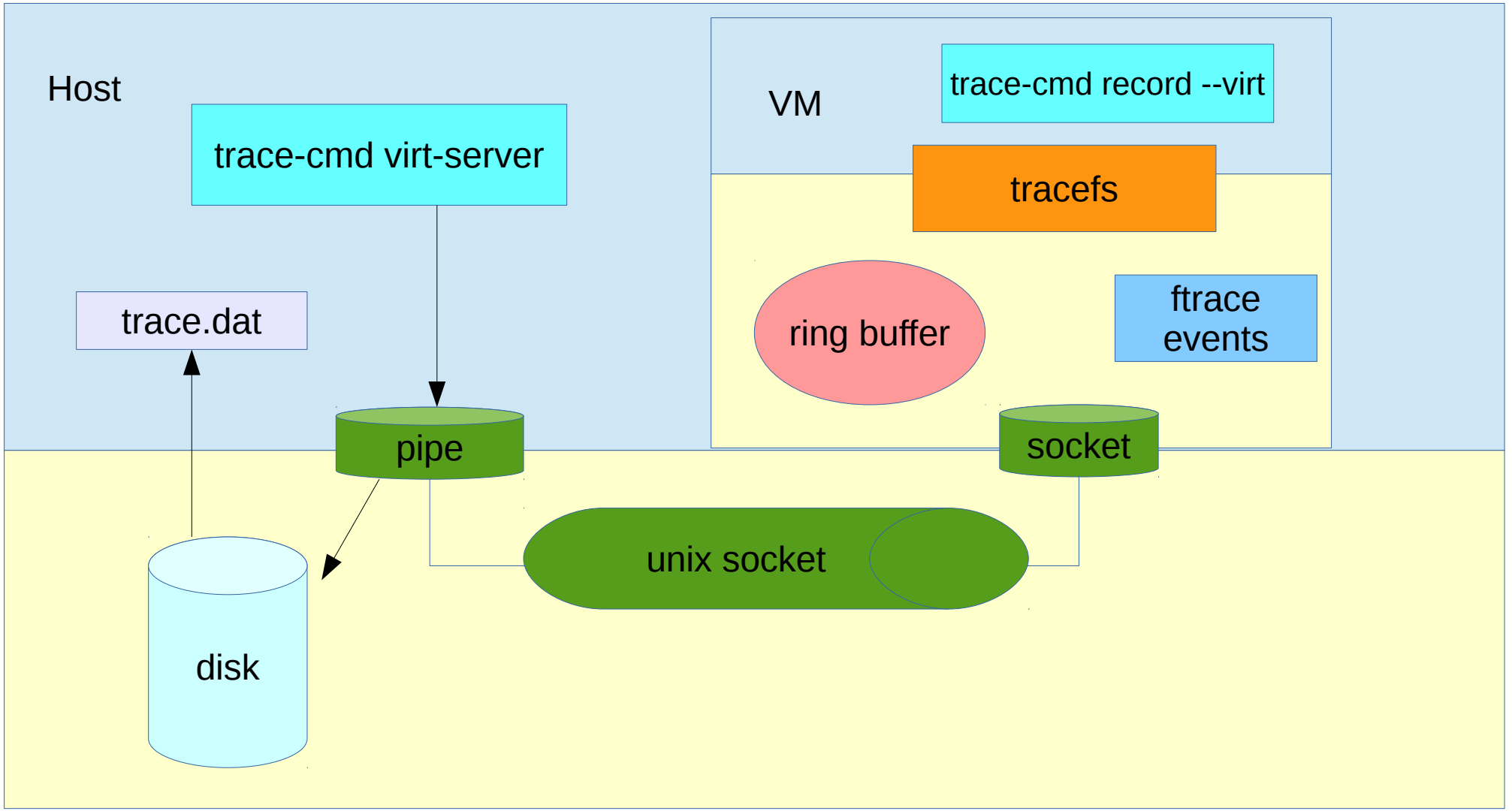
```
start_xmit() {
  free_old_xmit_skbs.isra.39() {
    virtqueue_get_buf() {
      detach_buf();
    __dev_kfree_skb_any() {
      consume_skb() {
        skb_release_all() {
          skb_release_head_state();
          skb_release_data() {
            kfree() {
              __slab_free();
        kfree_skbmem() {
          kmem_cache_free() {
            __slab_free();
    virtqueue_get_buf();
  skb_clone_tx_timestamp() {
    classify();
  skb_to_sgvec() {
    __skb_to_sgvec();
  virtqueue_add_outbuf() {
    alloc_indirect.isra.4() {
      __kmalloc() {
        kmalloc_slab();
  sock_wfree() {
    sock_def_write_space();
  virtqueue_kick() {
    virtqueue_kick_prepare();
    vp_notify();
_raw_spin_lock();
__local_bh_enable_ip();
__local_bh_enable_ip();
```

trace-cmd listen --virt

- **Introducing the “virt” option!**
  - **Created by Yoshihiro Yunomai (originally called “virt-server”)**
  - **Continued by Masami Hiromatsu**
  - **Continued by myself**
- **Host interacts directly with virtual machines**
- **No going through the network layer**
- **Fast transactions of events**
- **Consistent time stamps**
- **Interleaving of events between host and guest.**

```
__splice_from_pipe() {
  splice_from_pipe_next();
  generic_pipe_buf_confirm();
  pipe_to_sg() {
    generic_pipe_buf_steal() {
      _cond_resched();
    }
    unlock_page() {
      __wake_up_bit();
    }
  }
```

trace-cmd listen --virt / ftrace flow



trace-cmd listen --virt [setup]

- **trace-cmd listen --virt --dom <guest> -c <cpus>**
  - **Creates (if does not exist)**
    - **/var/lib/trace-cmd/virt/agent-ctl-path**
      - **socket**
    - **/var/lib/trace-cmd/virt/<guest>/trace-path-cpu0.in**
    - **/var/lib/trace-cmd/virt/<guest>/trace-path-cpu0.out**
      - **FIFO**
  - **Needs to be done only once (FIFO not deleted)**
  - **agent-ctr-path only exists when running**
- **trace-cmd listen --virt**
  - **Creates agent-ctr-path while running**

trace-cmd listen --virt [setup]

- **qemu**

```
-device virtio-serial-pci,id=virtio-serial0,bus=pci.0

-chardev socket,id=charchannel0,path=/var/lib/trace-cmd/virt/agent-ctl-path

-device virtserialport,bus=virtio-serial0.0,nr=1,chardev=charchannel0,id=channel0,
    name=agent-ctl-path

-chardev pipe,id=charchannel1,path=/var/lib/trace-cmd/virt/merry/trace-path-cpu0

-device virtserialport,bus=virtio-serial0.0,nr=2,chardev=charchannel1,id=channel1,
    name=trace-path-cpu0
```

trace-cmd listen --virt [setup]

- **libvirt**
  - **/etc/libvirt/qemu/<guest>.xml**

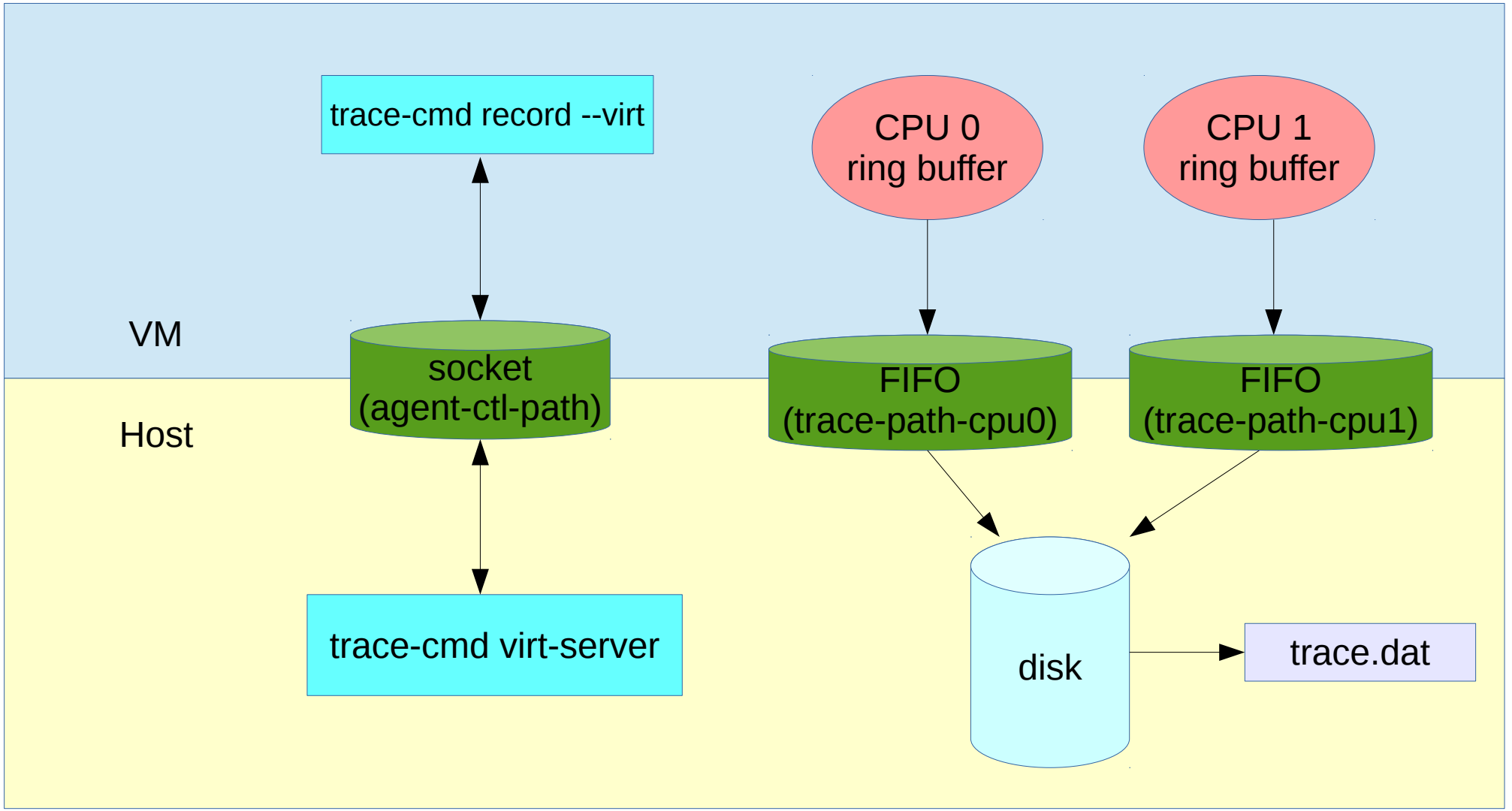
```
<channel type='unix'>
  <source mode='connect' path='/var/lib/trace-cmd/virt/agent-ctl-path'/>
  <target type='virtio' name='agent-ctl-path'/>
  <address type='virtio-serial' controller='0' bus='0' port='3'/>
</channel>

<channel type='pipe'>
  <source path='/var/lib/trace-cmd/virt/merry/trace-path-cpu0'/>
  <target type='virtio' name='trace-path-cpu0'/>
  <address type='virtio-serial' controller='0' bus='0' port='4'/>
</channel>
```

trace-cmd listen --virt [setup]

- **libvirt**
  - **Make sure to include all CPUs**

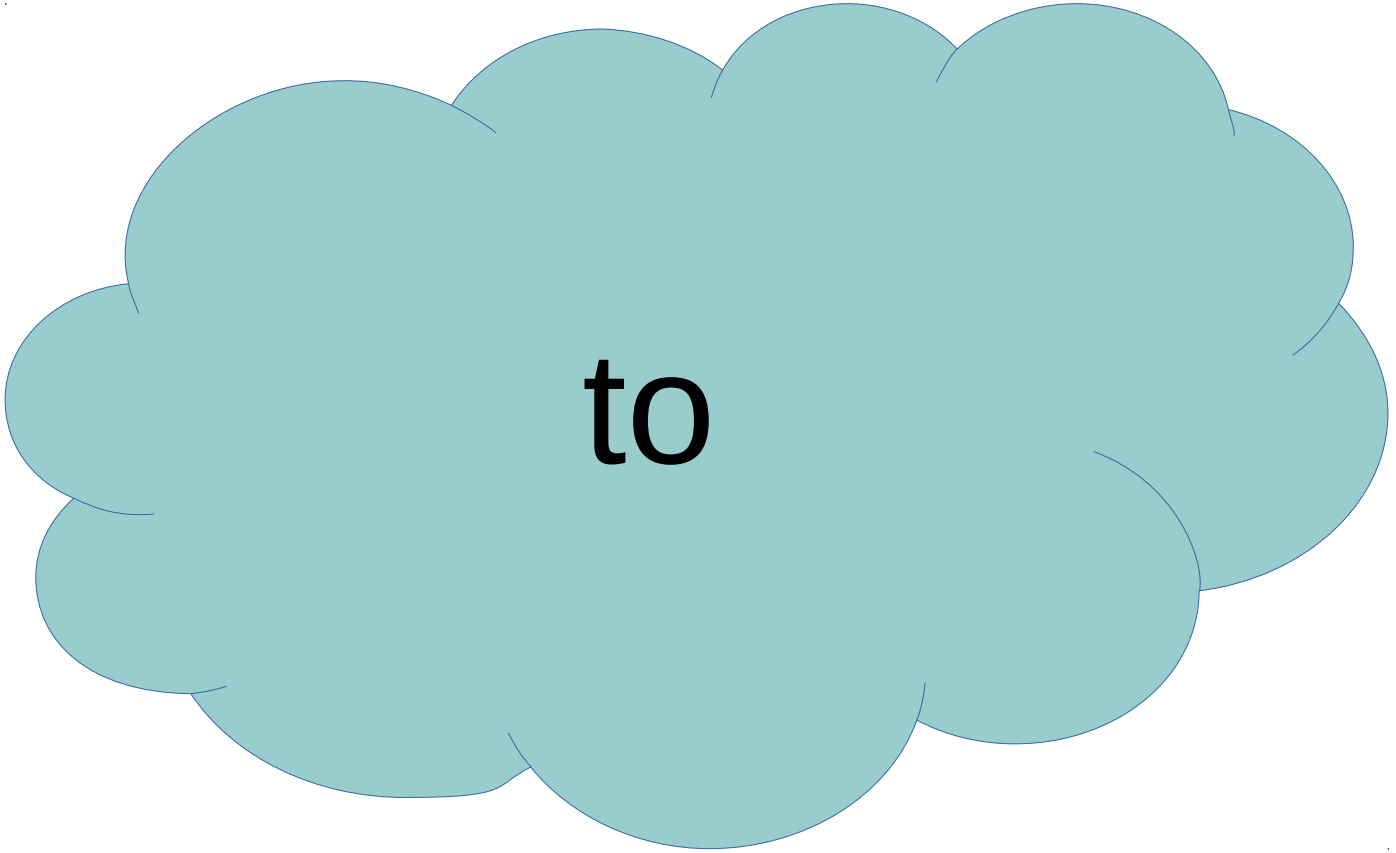
```
<channel type='pipe'>
  <source path='/var/lib/trace-cmd/virt/merry/trace-path-cpu1'/>
  <target type='virtio' name='trace-path-cpu1'/>
  <address type='virtio-serial' controller='0' bus='0' port='5'/>
</channel>
<channel type='pipe'>
  <source path='/var/lib/trace-cmd/virt/merry/trace-path-cpu2'/>
  <target type='virtio' name='trace-path-cpu2'/>
  <address type='virtio-serial' controller='0' bus='0' port='6'/>
</channel>
<channel type='pipe'>
  <source path='/var/lib/trace-cmd/virt/merry/trace-path-cpu3'/>
  <target type='virtio' name='trace-path-cpu3'/>
  <address type='virtio-serial' controller='0' bus='0' port='7'/>
</channel>
```

trace-cmd virt-server / ftrace flow

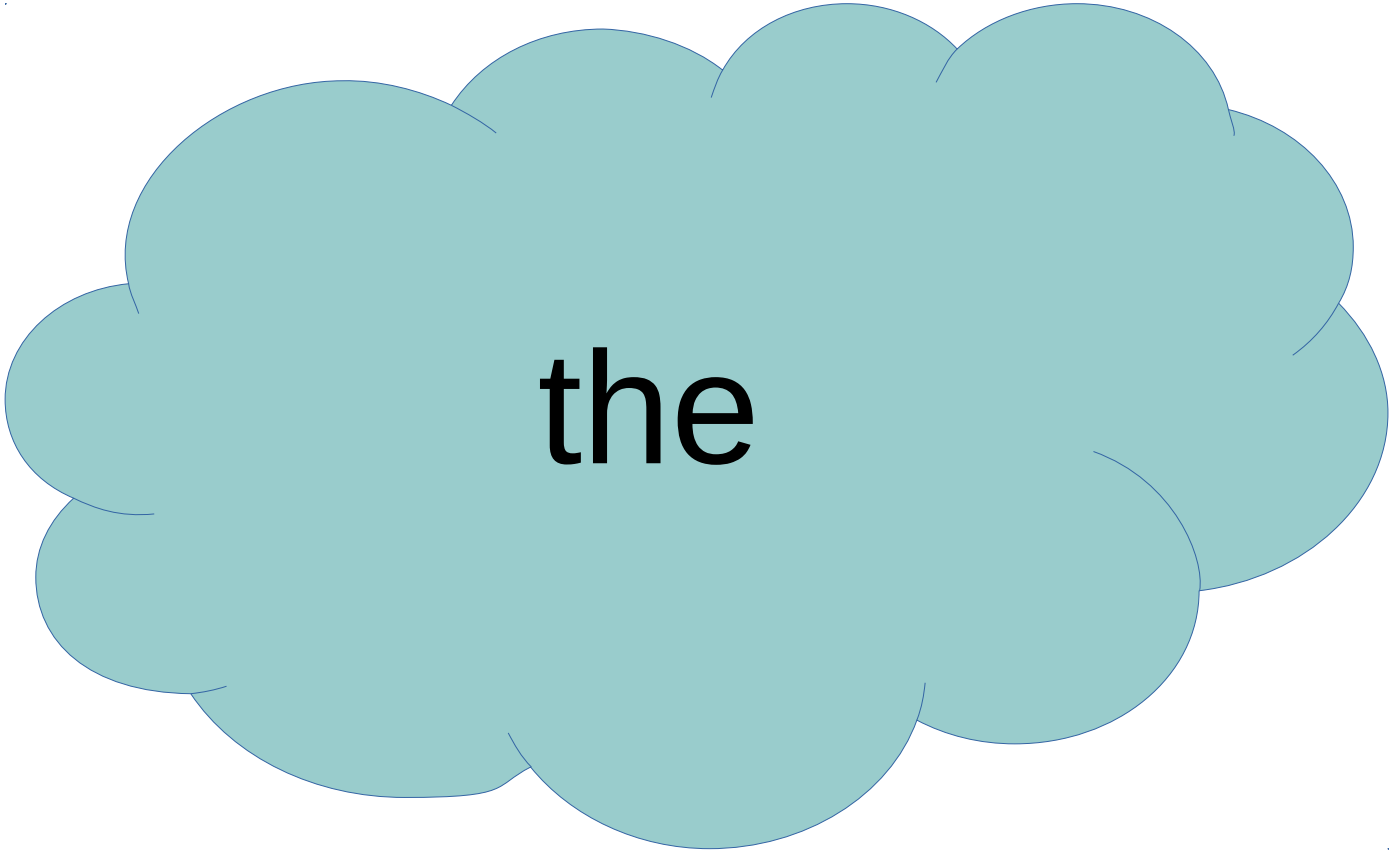


Welcome to the CLOUD!

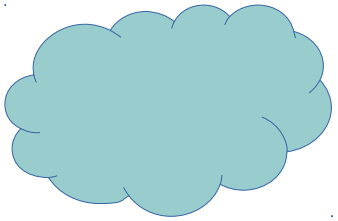
Welcome to the CLOUD!

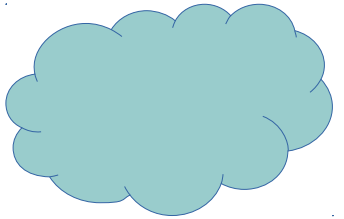


Welcome to the CLOUD!

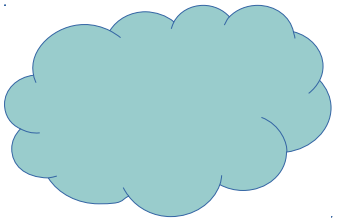
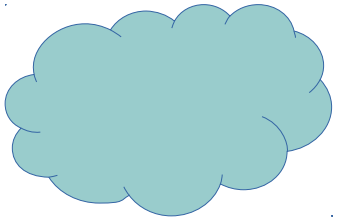


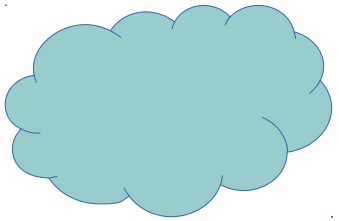
Welcome to the CLOUD!



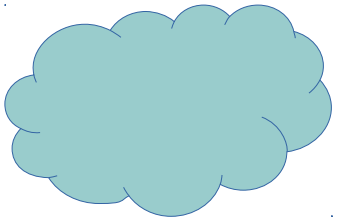


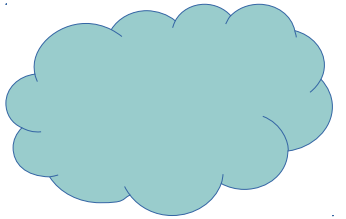

Vaporware!

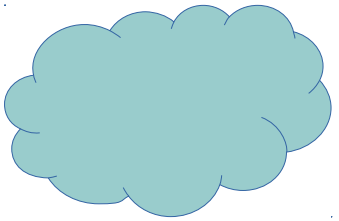
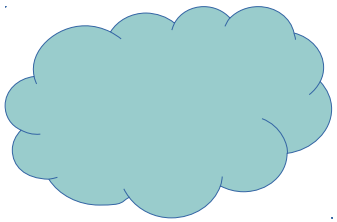




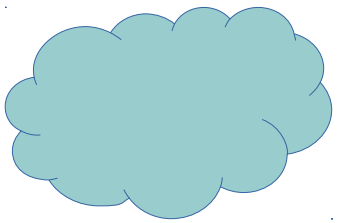


Welcome to the CLOUD!







Well really

Almostware!

Coming soon to trace-cmd

- **Everything up to now has been implemented**
- **But that stuff is boring**
- **The cool stuff is currently being created**
- **It's still under design**
- **YOU can influence the results!!!!**

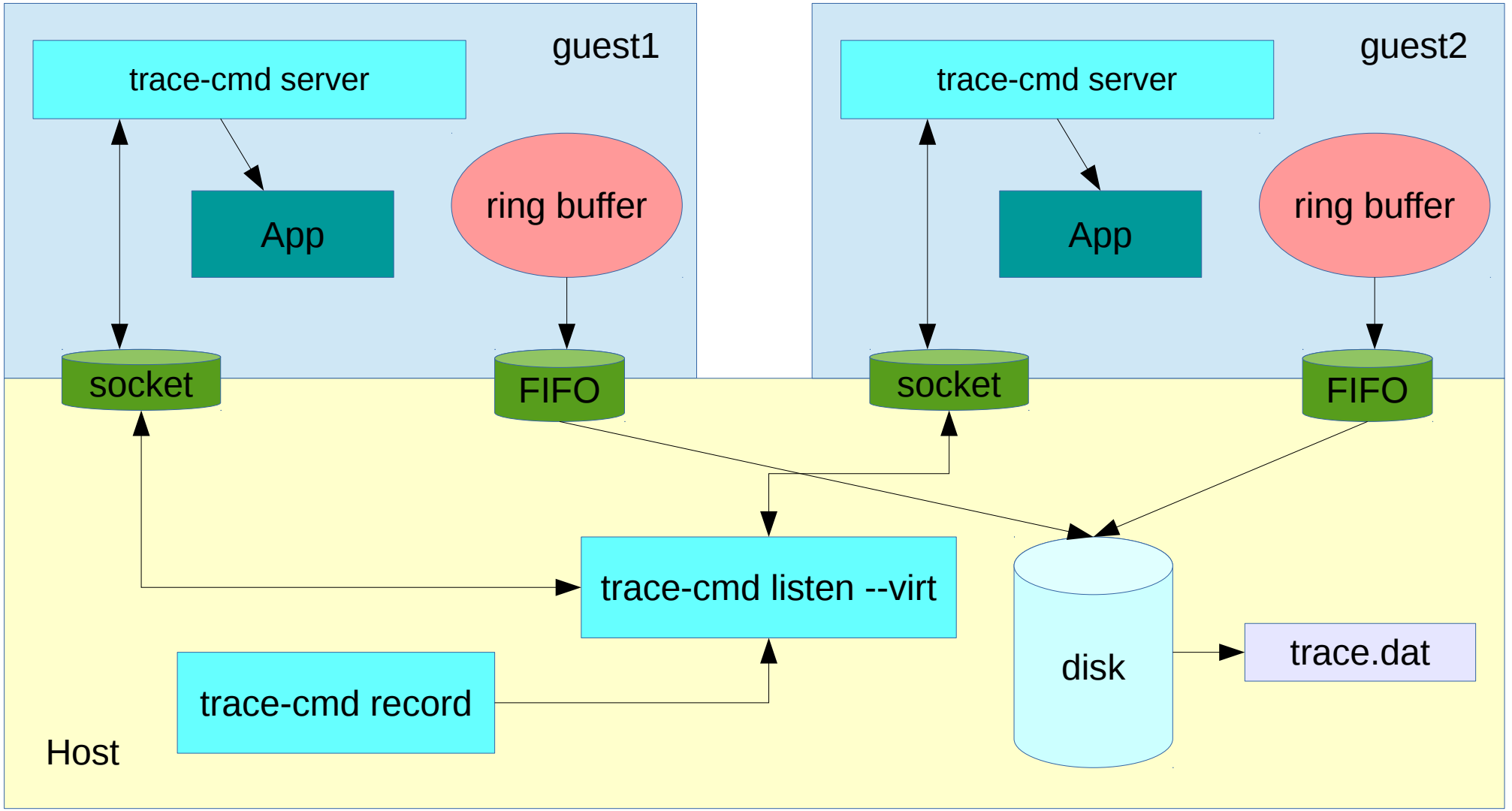
trace-cmd server (what I want)

- **trace-cmd server**
  - **Run from the guest (or a remote host)**
  - **Will let the host start tracing**
  - **Must connect to the host listen --virt**
- **trace-cmd record --dom <guest>**
  - **Run from the host**
  - **Connects to the guest**
    - **through listen --virt running on the host**
  - **Can connect to more than one guest**
  - **Can start tracing on host too!**

trace-cmd server (what I want)

```
trace-cmd record -e kvm \
   --dom guest1 -e all  \
   --dom guest2 -p function \
  hostapp
```

trace-cmd server / ftrace flow



trace-cmd server (what I want)

- **trace-cmd server --exec**
  - **Allows the host to also execute commands**
  - **Synchronize host tracing with guest applications**
  - **Must trust the host (or remote server)**
    - **Basically allowing the host have root access**

trace-cmd server (what I want)

```
trace-cmd record -e kvm \
   --dom guest1 -e all --exec cyclictest -a -p 80 \; \
  hostapp
```

trace-cmd server Issues!

- #1 is security
  - Must explicitly allow exec (with --exec)
  - Who do we trust?
    - Host files access by anyone with group permissions
    - Guest trusts host
      - host needs to keep permissions of the control files
- trace-cmd listen --virt must always run
  - Is there a better method
  - Can we start listen --virt after guest?
  - Can guest run without listen --virt?

Questions?