View-OS:
Change your View on Virtualization

Renzo Davoli
University of Bologna
Bologna, Italy

Michael Goldweber
Xavier University
Cincinnati, OH USA

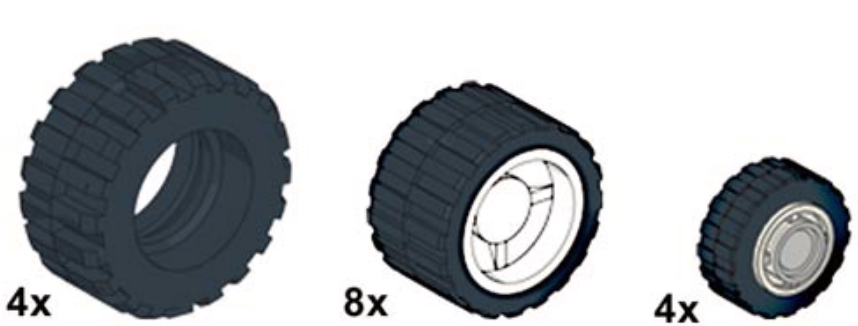
VIRTUAL SQUARE LAB

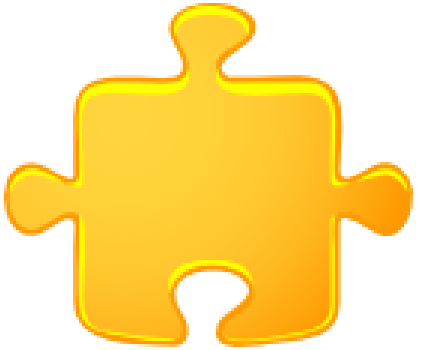
Linux Kongress
Dresden, Germany
October, 29 2009

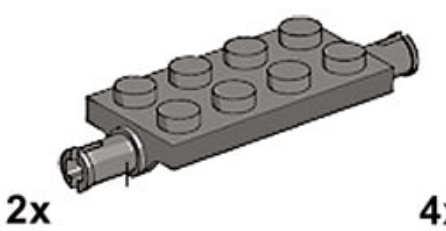
VIRTUAL SQUARE LAB

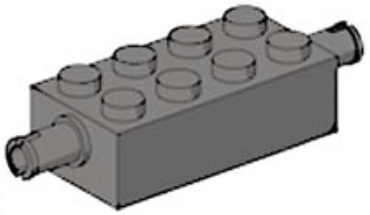
- An international lab on virtuality
- Project repository:
  - Virtual Distributed Ethernet (VDE)
  - LWIPv6
  - PureLibC
  - View-OS
    - umview/kmview
    - modules

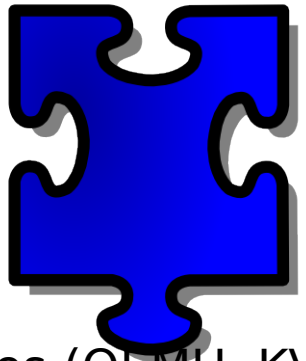
Virtual Square











Virtual Machines (QEMU, KVM, User-Mode Linux),
Virtual Networks (e.g. Overlay networks, VPNs),
Virtual Execution Environments,
Virtual File systems,
Virtual root (chroot),
Virtual users(e.g. fakeroot),
Virtual Time,
Virtual Honey Pots,
Virtual executable intepreters (e.g. binfmt), ...

Virtual Square Goals

Communication

Different *virtualities* must be interconnected.

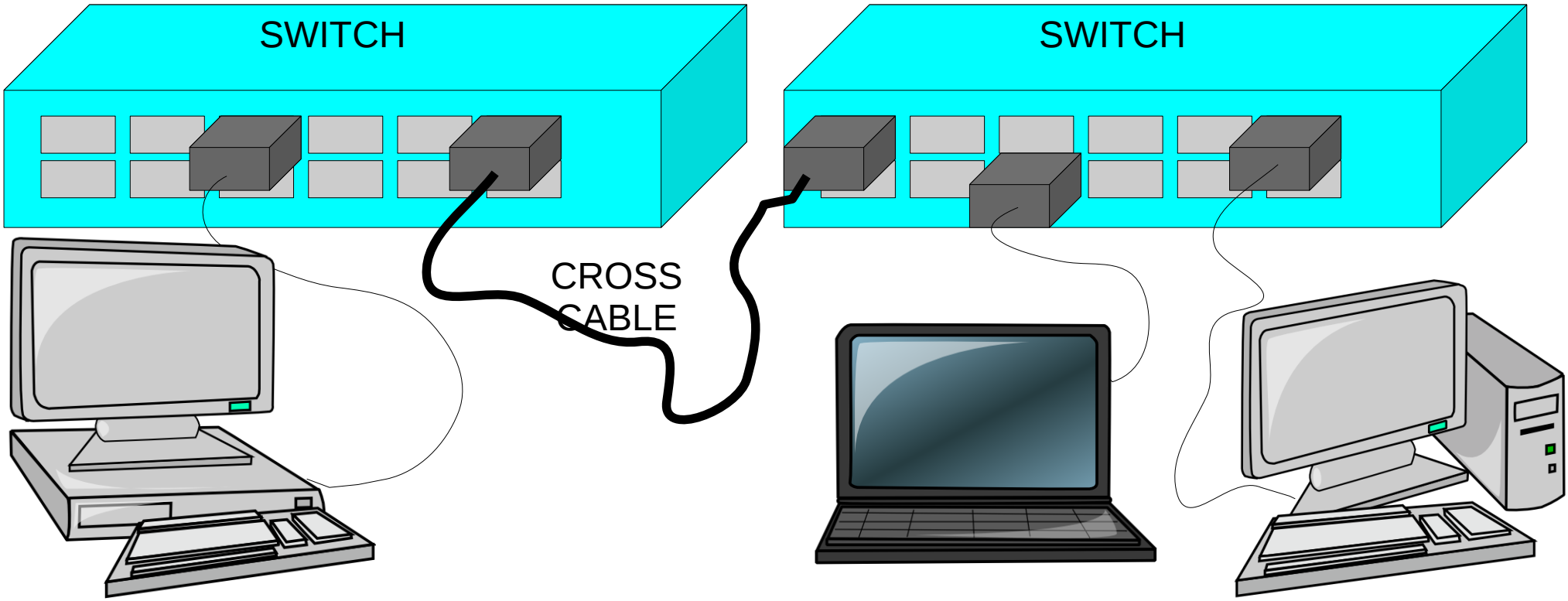
Integration

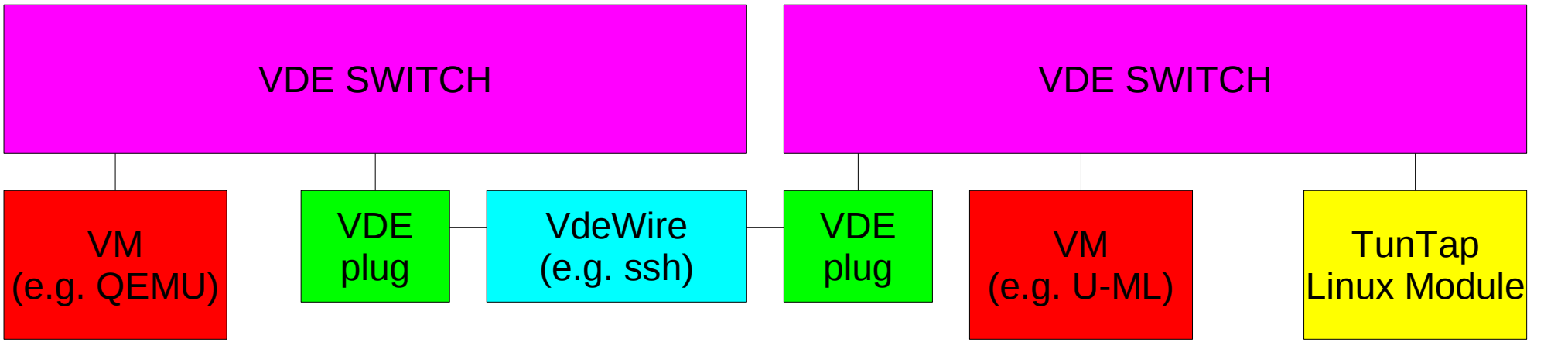
Seeing how specific *virtualities* can be seen as special cases of broader ideas of *virtuality.*

Extension

Several needs could be captured by some model of *virtuality;* V^2 seeks a unifying idea of *virtuality* able to provide a consistent and extensible view.

Virtual Distributed Ethernet (VDE)





LWIPv6

An Ipv4/v6 stack implementation as a library.

Fork project from LWIP project (Adam Dunkels <adam@sics.se>)

Can be connected to any number of VDE, TUN, TAP interfaces.

It's a hybrid stack (not a dual-stack). A single Ipv6 “engine” that is able also to manage Ipv4 packets in compatibility mode

(e.g. 130.136.1.110 is managed as 0::ffff:130.136.1.110).

View-OS

... a process with a view

Each process should be permitted to have its own ***view*** of the execution environment

View Components

filesystem namespace, including the related ownership and permission information,

networking configuration,

system name,

current time,

devices, etc.

...

Global View Assumption

In general processes running on the same computer share the same view.

- A given pathname refers to the same file for all processes.
- All processes use one shared TCP-IP stack for networking hence all processes share the same set of IP addresses and routing policies.
- All processes share the same notion as to which users/processes have special priviledges.

View-OS vs. VMs and Containers

	View-OS	VM	Container
Memory Impact	LOW	HIGH	LOW
Running State	User	User or Kernel	Kernel
Administered by:	user	user	root
Partial Virtualization	Yes	No	Yes (sharing)

Partial Virtualization

- Virtualize just what you need:
  - Virtual and real file systems, devices, networks, etc. co-exist in the process' view
- Support for nested virtualization:
  - e.g. virtual file system defined on virtual devices.

HANDS ON!
How to start a View-OS monitor:

```
user@host:~$ umview bash
This kernel supports: PTRACE_MULTI PTRACE_SYSVM ppoll
View-OS will use: PTRACE_MULTI PTRACE_SYSVM ppoll

pure_libc library found: syscall tracing allowed

rd235 2.6.29-utrace GNU/Linux/View-OS 10585 0
user@host[10585:0]:~$
```

Umview runs on vanilla Linux kernels, Kmview requires a kernel module loaded (and utrace).

Instead of bash one may run his/her favorite executable (e.g. xterm, script....)

View-OS modules

View-OS monitor loads only the virtualities requested by the user:

- Umfuse: file system virtualization
- Umnet: networking virtualization
- Umdev: device virtualization
- Umbinfmt: executable interpreter virtualization
- Viewfs: file system patchworking
- Ummisc: time, system id...

Example #1: Virtual Installation of Software

```
$ um_add_service viewfs
$ mkdir /tmp/newroot
$ viewsu
# mount -t viewfs -o mincow,except=/tmp,vstat /tmp/newroot /
# aptitude install mynewsoftware
```

Create an empty dir

Mount it in “minimal copy on write” mode:

- File mod's are on the real file system when allowed.
- Mod's stored in the mounted dir otherwise.
- A single consistent view.
- Vstat: virtualize stat (support for virtual chown, chmod/setuid, special files)

Example 2: Virtual Networking

```
$ um_add_service umnet
$ mount -t umnetlwipv6 none /dev/net/default
$ ip link set vd0 up
$ ip addr add 10.1.2.3/24 dev vd0
$ ip addr
1: lo0: <LOOPBACK,UP> mtu 0
    link/loopback
    inet6 ::1/128 scope host
    inet 127.0.0.1/8 scope host
2: vd0: <BROADCAST,UP> mtu 1500
    link/ether 02:02:5a:44:e2:06 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::2:5aff:fe44:e206/64 scope link
    inet 10.1.2.3/24 scope global
```

A network stack can be “mounted.”

/dev/net/default is the default stack, but View-OS supports multiple stacks.

Example 2: Virtual Networking

```
$ um_add_service umnet
$ mount -t umnetlwipv6 -o tn0=tunx none /dev/lwip0
$ mount -t umnetlwipv6 -o tp0=tapx,vd0=/tmp/switch none /dev/lwip1
$ mstack /dev/lwip0 ip addr
1: lo0: <LOOPBACK,UP> mtu 0
    link/loopback
    inet6 ::1/128 scope host
    inet 127.0.0.1/8 scope host
2: tn0: <> mtu 0
    link/generic
$ mstack /dev/lwip1 ip addr
1: lo0: <LOOPBACK,UP> mtu 0
    link/loopback
    inet6 ::1/128 scope host
    inet 127.0.0.1/8 scope host
2: vd0: <BROADCAST> mtu 1500
    link/ether 02:02:47:98:ad:06 brd ff:ff:ff:ff:ff:ff
3: tp0: <BROADCAST> mtu 1500
    link/ether 02:02:03:04:05:06 brd ff:ff:ff:ff:ff:ff
$
```

Example 3: Mount a Filesystem

```
$ um_add_service umfuse
$ mount -t umfuseext2 -o ro ext2filesystemimage /mnt
$ mount -t umfusestrangefilesystem strangeimage /mnt2
```

Source compatible with Fuse.

Mount file systems unsupported by the kernel.

Safe mount, limited to this View.

Example 4: Partition a Filesystem Image and Mount its Partition(s)

Step 1: Load the umdev (virtual device) module and mount an empty file as a disk image.

```
$ um_add_service umdev
$ viewsu
# dd of=/tmp/diskimage bs=1024 count=0 seek=1024000
# mount -t umdevmbr /tmp/diskimage /dev/hda
```

Example 4 (con't):

Step 2: partition the file system image:

```
# fdisk /dev/hda
Device contains a valid partition table
Building a new DOS disklabel with disk identifier 0xd403417d.
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-127, default 1): 1
Last cylinder, +cylinders or +size{K,M,G} (1-127, default 127): 127
Command (m for help): p
Disk /dev/hda: 1048 MB, 1048576000 bytes
255 heads, 63 sectors/track, 127 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0xd403417d
Device Boot      Start         End      Blocks   Id System
/dev/hda1               1         127     1020096   83 Linux
Command (m for help): w
The partition table has been altered!
Calling ioctl() to re-read partition table.
Syncing disks.
```

Example 4 (con't)

Step 3: Create the filesystem

```
# mkfs.ext2 /dev/hda1
mke2fs 1.41.8 (20-Jul-2009)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
63872 inodes, 255024 blocks
12751 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=264241152
8 block groups
32768 blocks per group, 32768 fragments per group
7984 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
This filesystem will be automatically checked every 38 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
```

Example 4 (con't):

Step 4: mount the new partition

```
# um_add_service umfuse
# mount -t umfuseext2 -o rw+ /dev/hda1 /mnt
# ls -l /mnt
total 16
drwx------ 2 root root 16384 2009-09-16 11:57 lost+found
```

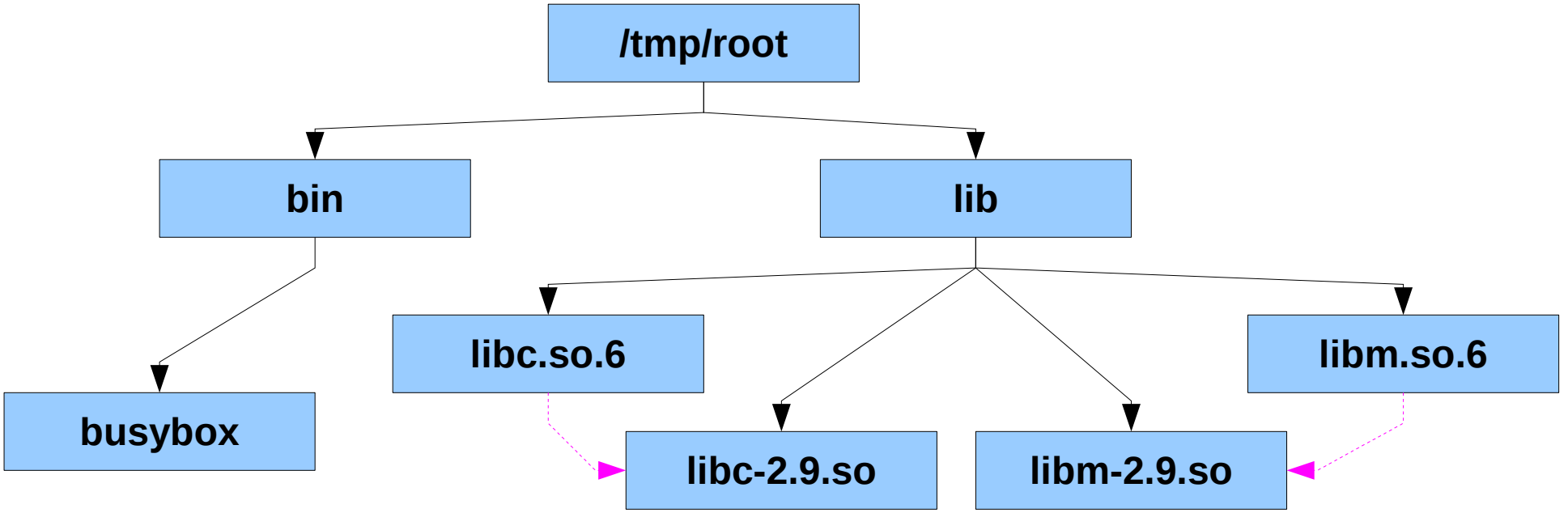
Example of nested virtualization.

Compatible with standard sys-admin commands.

Example 5: User Mode `chroot`

Step 1: create the *cage* filesystem

```
$ mkdir /tmp/root /tmp/root/bin /tmp/root/lib
$ cp /bin/busybox /tmp/root/bin
$ cp /lib/libm-2.9.so /lib/libc-2.9.so /tmp/root/lib
$ cd /tmp/root/lib
$ ln -s libm-2.9.so libm.so.6
$ ln -s libc-2.9.so libc.so.6
$ cd /
```



Example 5: User Mode `chroot`

Step 2: change the file system root:

Core mode: by the virtual chroot system call

```
$ exec /usr/sbin/chroot /tmp/root /bin/busybox sh
BusyBox v1.13.3 (Debian 1:1.13.3-1) built-in shell (ash)
Enter 'help' for a list of built-in commands.
/ $
```

By Viewfs:

```
$ um_add_service viewfs
$ exec busybox sh
BusyBox v1.13.3 (Debian 1:1.13.3-1) built-in shell (ash)
Enter 'help' for a list of built-in commands.
/ $ mount -t viewfs -o move,permanent /tmp/root /
/ $
```

Example 5: User Mode `chroot`

Step 3: the process is in the cage:

```
/ $ ls -lR /
/:
drwxr-xr-x 2 1000   1000     4096 Sep 17 13:37 bin
drwxr-xr-x 2 1000   1000     4096 Sep 17 13:37 lib
/bin:
-rwxr-xr-x   1 1000 1000   401216 Sep 17 13:37 busybox
/lib:
-rwxr-xr-x   1 1000 1000  1302732 Sep 17 13:37 libc-2.9.so
lrwxrwxrwx   1 1000 1000       11 Sep 17 13:37 libc.so.6 -> libc-2.9.so
-rw-r--r--   1 1000 1000   149328 Sep 17 13:37 libm-2.9.so
lrwxrwxrwx   1 1000 1000       11 Sep 17 13:37 libm.so.6 -> libm-2.9.so
/ $
```

Example 6: Create a Ramdisk and use it:

```
$ um_add_service umdev
$ um_add_service umfuse
$ um_add_service umproc
$ mount -t umdevramdisk -o size=100M none /dev/hdx
$ /sbin/mkfs.vfat /dev/hdx
mkfs.vfat 3.0.3 (18 May 2009)
$ mount -t umfusefat -o rw+ /dev/hdx /mnt
$ mount
rootfs on / type rootfs (rw)
/dev/root on / type ext3 (rw,errors=remount-ro,data=ordered)
tmpfs on /lib/init/rw type tmpfs (rw,nosuid,mode=755)
... ...
none on /proc/mounts type proc (ro)
none on /dev/hdx type umdevramdisk (size=100M)
/dev/hdx on /mnt type umfusefat (rw+)
$
```

Another example of nested virtualization

Umproc virtualizes /proc/mounts

Example 7: Virtualization of running processes

Shell #1 (pid 12345, an ordinary shell)

```
sh1 $ mkdir /tmp/mnt
sh1 $ ls /tmp/mnt
sh1 $
```

Shell #2 (running under ViewOS)

```
sh2 $ um_add_service umfuse
sh2 $ mount -t ext2 /tmp/linux.img /tmp
sh2 $ ls /tmp/mnt
bin  boot dev etc lib lost+found mnt    proc sbin tmp usr
sh2 $ um_attach 12345
```

Shell #1 has been “attached” to ViewOS

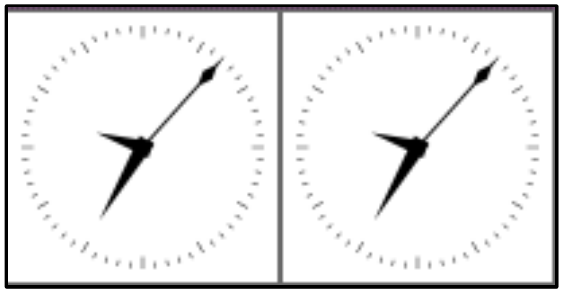
```
sh1 $ ls /tmp/mnt
bin  boot dev etc lib lost+found mnt    proc sbin tmp usr
```

Example 8: Process *proper time*

Start two `xclocks` one from a standard shell, the other from a shell running ViewOS.

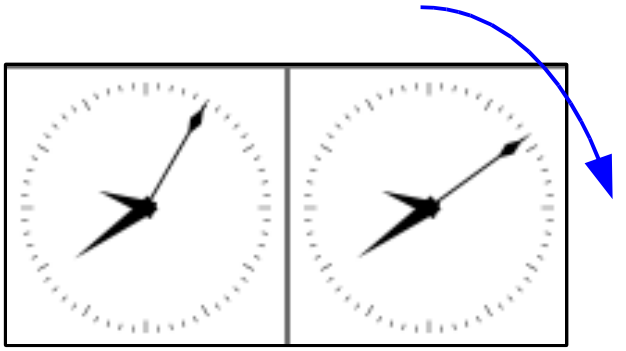
```
sh 1 $ xclock -update 1 &

sh2 $ xclock -update 1 &
sh2 $ um_add_service ummisc
sh2 $ mount -t ummisctime none /tmp/mnt
```

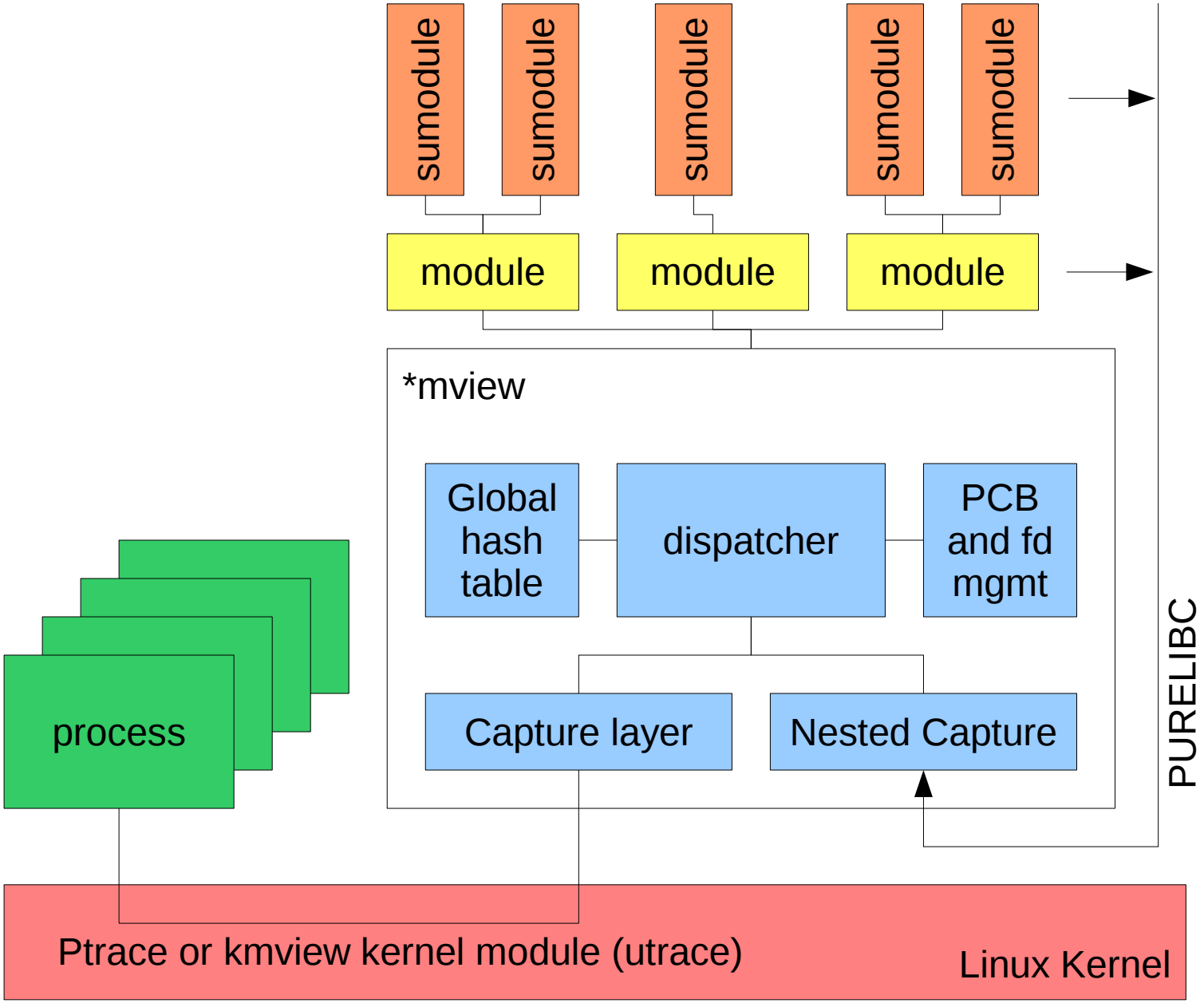


Now change the frequency of the virtual time for ViewOS:

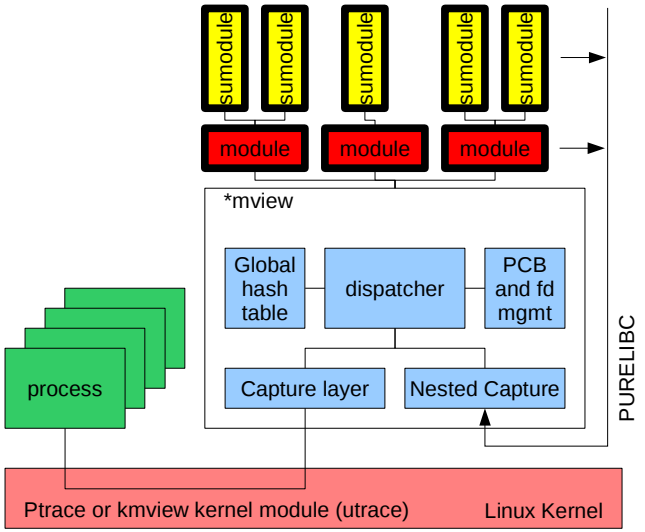
```
sh2 $ echo 2 > /tmp/mnt/frequency
```



Behind the Scenes



Modules & submodules

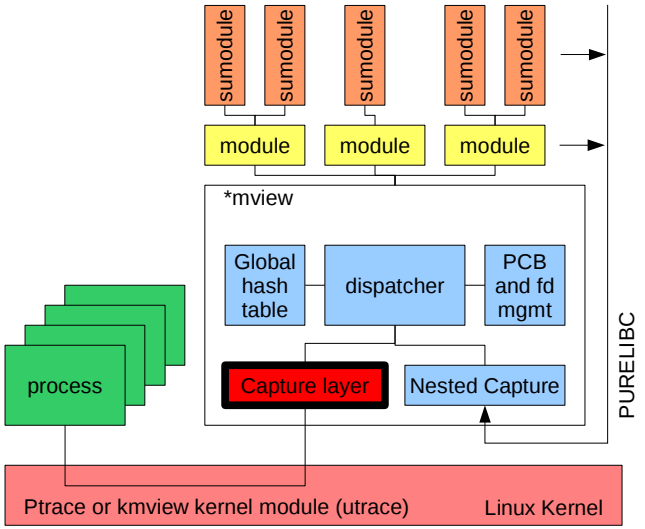


- Modules provide support for classes of virtualizations, e.g.:
  - Umfuse: file systems
  - Umnet: networking
  - Umdev: devices
- Submodules are for specific cases, e.g.:
  - Umfuseext2, Umfusefat
  - Umnetlwipv6, umnetnull
  - Umdevmbr, umdevramdisk

Modules & submodules

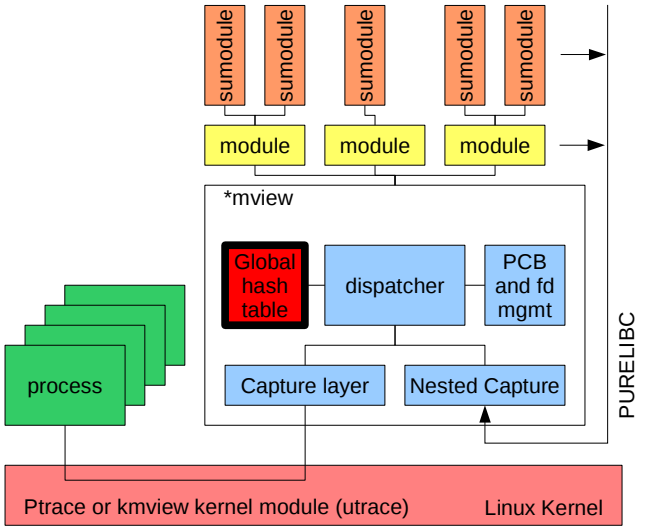
module	description	submodule	description
umproc	/proc/mounts virtualization		
umfuse	User-mode fuse	umfuseext2	ext2 implementation
		umfuseiso9660	iso9660
		umfusefat	vat/vfat
		umfusessh (sshfs)	remote file system via ssh
		umfuseencfs (encfs)	encrypted file system
umnet	network multi stack support	umnetnull	null stack
		umnetlwipv6	Ipv4/v6 hybrid stack
		umnetlink	move/merge stacks
		umnetnative	bypass to the kernel stack
umdev	device virtualization	umdevmbr	DOS master boot record
		umdevnull	null device
		umdevramdisk	ramdisk
		umdevtab	virtual tuntap
ummisc	system call based virtualization	ummisctime	time virtualization
		ummiscuname	uname id virtualization
viewfs	file system patchworking		
umbinfmt	interpreter selection		

Capture a process' system calls



- umview: based on ptrace
  - Vanilla Linux kernel
  - (patches proposed for performance)
- kmview needs a specific kernel module based on utrace.
  - Security enhancement
  - More complete virtualization support (nested View-OS, strace/gdb, SIGSTOP).

Global Hash Table

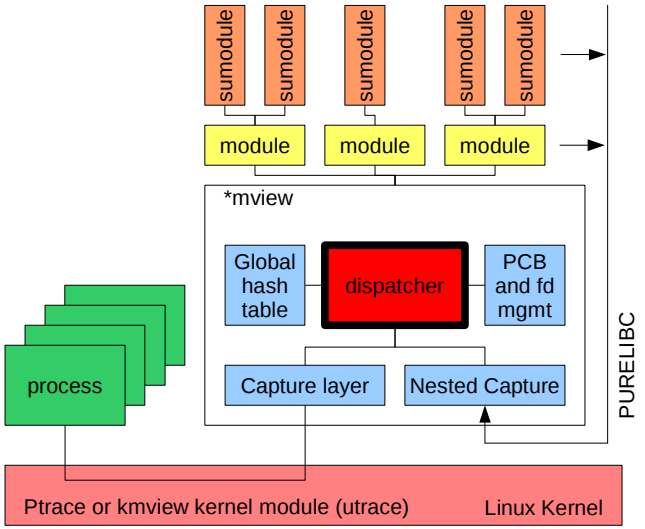


Keeps track of active virtualizations:

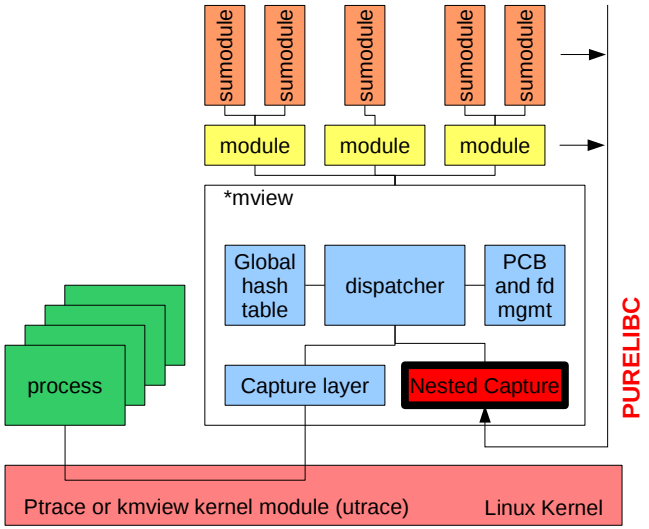
- Pathname objects
- File System Types
- Protocol families
- Device Major/Minor ranges
- System call numbers

Dispatcher

The Dispatcher uses the global hash table to route each system call to the right module or to the kernel.



Nested Capture



- View-OS captures (and can virtualize) the system calls generated by modules and submodules
- Purelibc is a C library providing process self virtualization

Desiderata: 1: Linux Kernel

New Ptrace tags for virtualization support:

- PTRACE_VM: support for partial virtualization. It is possible to skip the current system call and/or the second upcall after the system call. (User-Mode Linux can use this instead of PTRACE_SYSEMU. VM has a simpler implementation than SYSEMU.
- PTRACE_MULTI: process a sequence of ptrace requests + PEEK/POKE of large chunks as a single call. (ptrace exchanges one memory word per call and /proc/{pid}/mem is not writable!)

Desiderata
2. Open Group/POSIX

```
#include <msocket.h>

int msocket(char *path, int domain, int type, int
protocol);
```

Path is the pathname of the stack

domain/type/protocol are the same defined in socket(2).

A stack is a special file (new type of special file, see stat(2)):

```
#define S_IFSTACK 0160000
```

Each process has a default stack for each protocol family (domain).

If **`path==NULL`**, msocket uses the default stack.

It is backwards compatible:

```
#define socket(d,t,p) msocket(NULL,(d),(t),(p))
```

Desiderata: 2 Open Group/POSIX

```
int msocket(char *path, int domain, int type, int protocol);
```

if **type==SOCK_DEFAULT** msocket sets the default stack. e.g.

```
msocket("/dev/net/ipstack2",PF_INET,SOCK_DEFAULT,0);
```

defines **/dev/net/ipstack2** as the default stack for Ipv4

if **type==SOCK_DEFAULT && domain==PF_UNSPEC** msocket sets the default stack for all the protocol families.

Mstack uses msocket: it defines the default stack so that existing applications can use different stacks.

```
$ ip addr
..... ip addr on default net
$ mstack /dev/net/newstack firefox
.... firefox works on newstack
$ mstack /dev/net/otherstack bash
$ ...this new bash works on otherstack
```

Desiderata: 3- C library (glibc, eglibc)

C libraries are *impure,* they are pure C libraries and interface to system calls at the same time.

It is not possible to do self virtualization of system calls for processes using {e}glibc, library calls are internally linked to the system calls (e.g. printf calls write).

Purelibc is a (ld preloaded) layer on {e}glibc which convert the C library in a pure library

The support for self virtualization should be a feature of mainstream {e}glibc.

Desiderata: 4: utrace

UTRACE_STOP

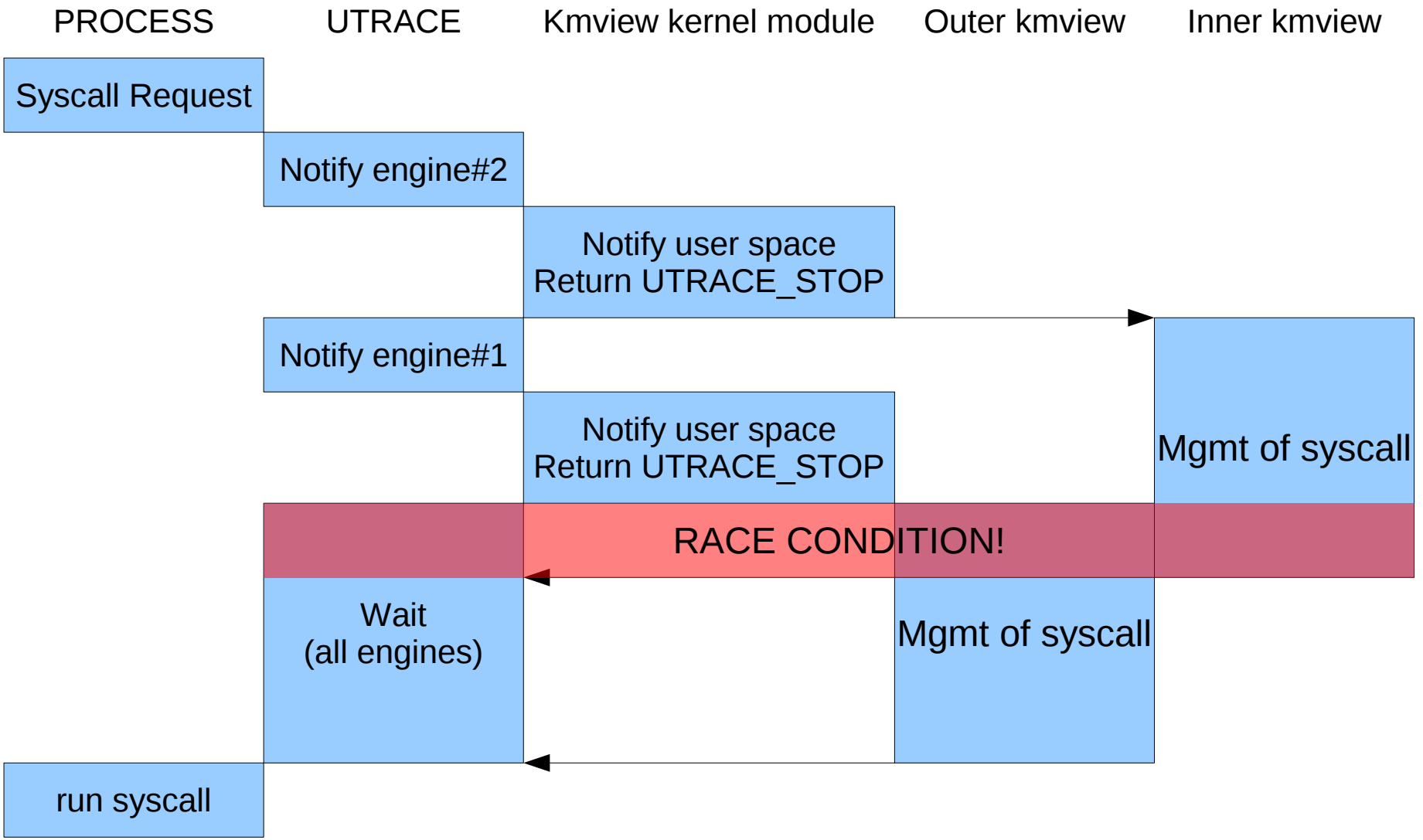
Utrace supports more tracers (engines) on the same process.

Utrace sends the notification to all the tracers and *then* waits for utrace_control(..., UTRACE_RESUME) from each tracer which returned UTRACE_STOP.

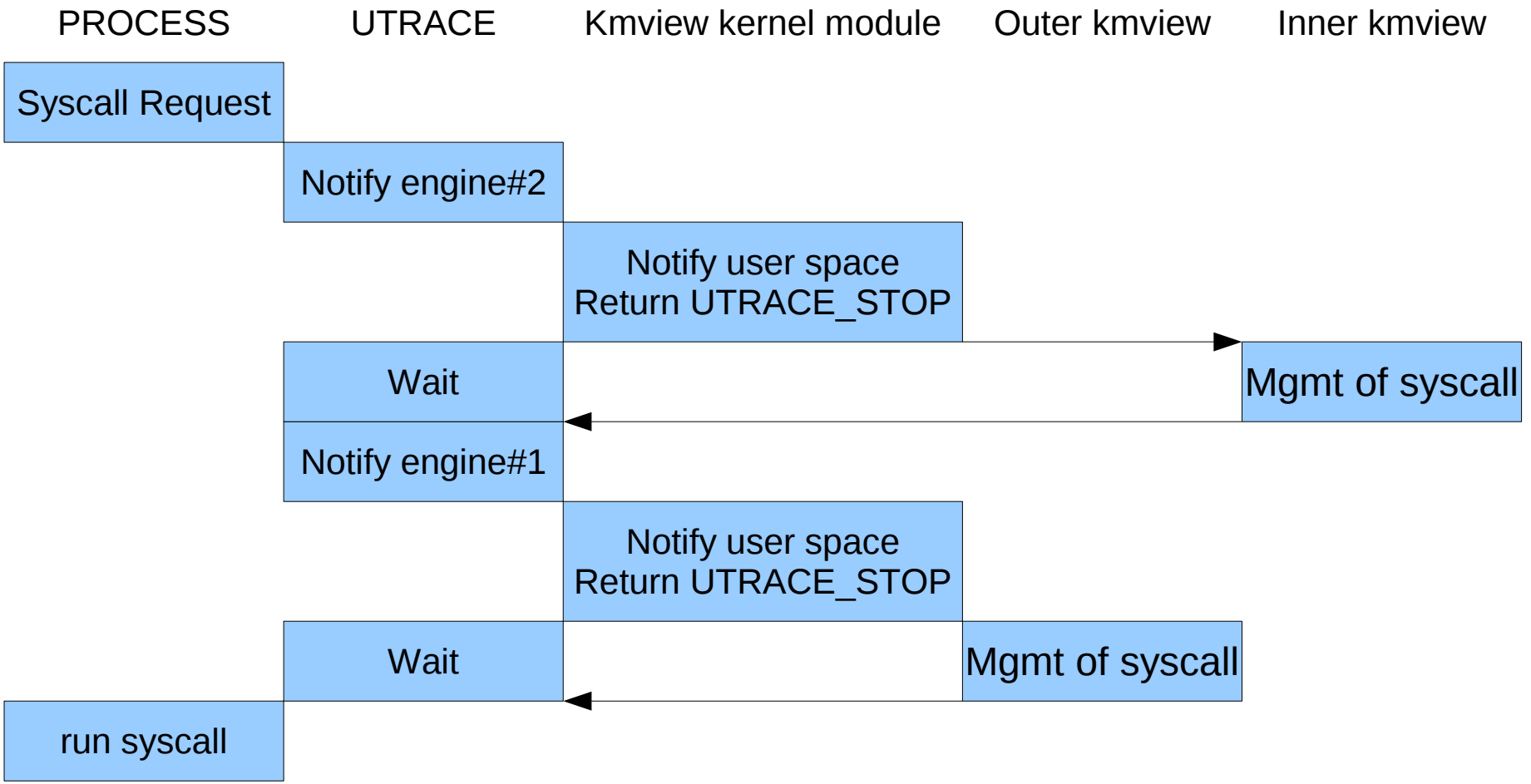
This specification is bad suited for nested virtualization support: a notification functions inspects the state (e.g. System call parameters) and maybe it changes the state. Next tracer must read the state as changed from the previous tracer.

Kmview uses a semaphore in its system call notification function to stop a process because this UTRACE_STOP specification is useless.

current UTRACE_STOP implementation

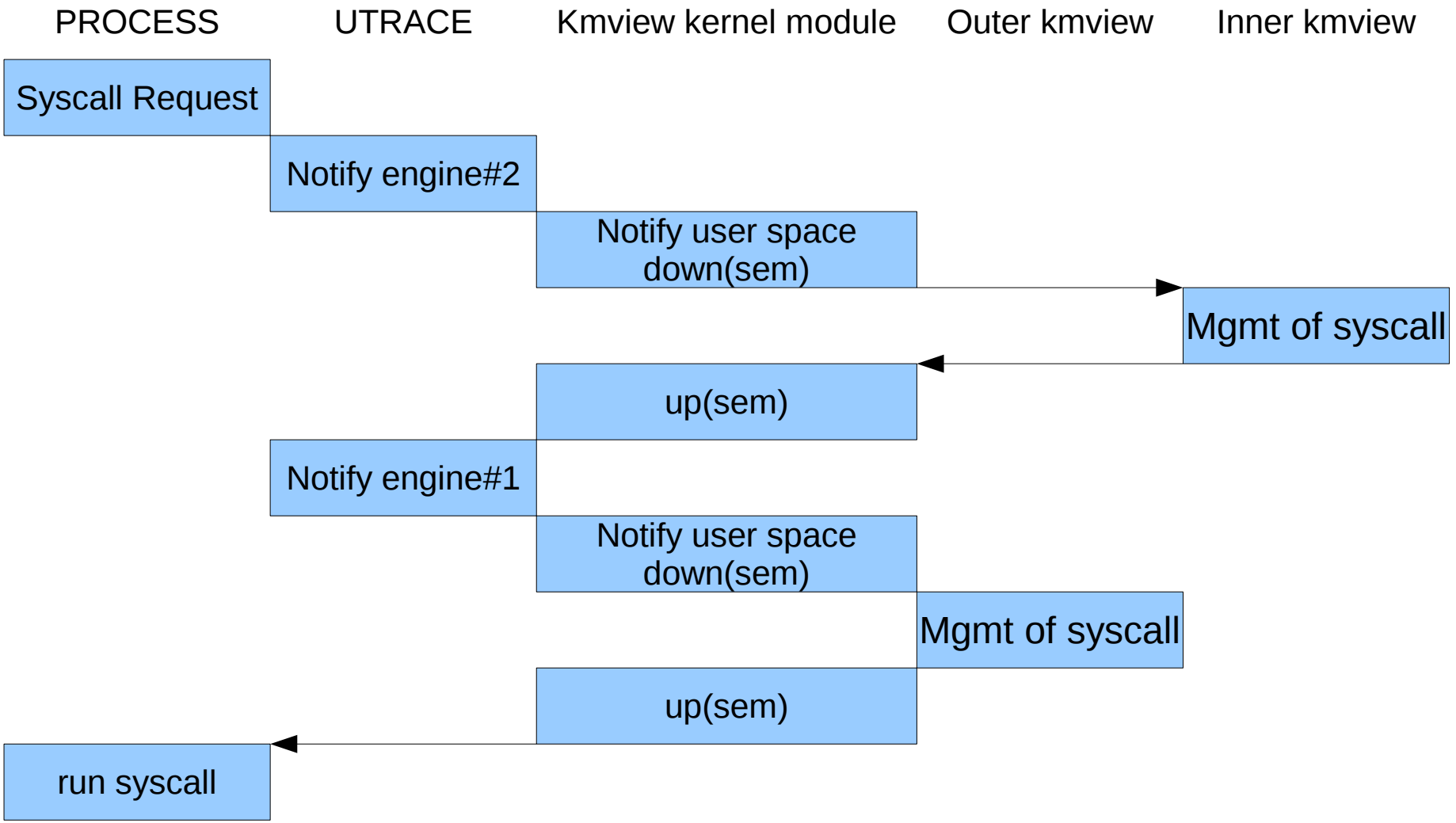


Desiderata: proposal for UTRACE_STOP



Kmview workaround

PTRACE_SYSCALL_{RUN,ABORT} instead of PTRACE_STOP



Desiderata 5: ext2fsprogs/lib/ext2fs

- Add the support for truncate in fileio.c
  - `ext2fs_file_set_size`
- Concurrent access to files for fileio.c
  - Access function to the inode buffer (?)
  - Externally provided inode pointer (?)

Multiple meaning of safety...

Availability, bug effects confinement:

- ViewOS runs outside the kernel, errors in modules may lead to a crash of the View (not a kernel panic!)

Self protection (from mistaken commands):

- Global View Assumption often force to use root access (or powerful capabilities), this is dangerous.

Sandbox non-circumvention:

- At the first sight it seems that Kernel based sandboxes are safer (e.g. seccomp).
  - Kernel based sandboxes are not flexible
  - On/Off security: a bug may compromise the whole system
  - A good support for VM can preserve safety

The more code, the worse security. Is the kernel “too fat?”

- Maintenance problems, side effects, etc.

The missing ring...

- View-OS modules are similar to microkernel servers.
- View-OS captures some of the benefit of microkernels (separation mechanism and policy, flexibility, reliability).
- View-OS allow microkernel services to be implemented (at user level) on monolithic kernels.

View-OS:
Change your View on Virtualization

Renzo Davoli – Michael Goldweber

Questions?