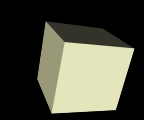
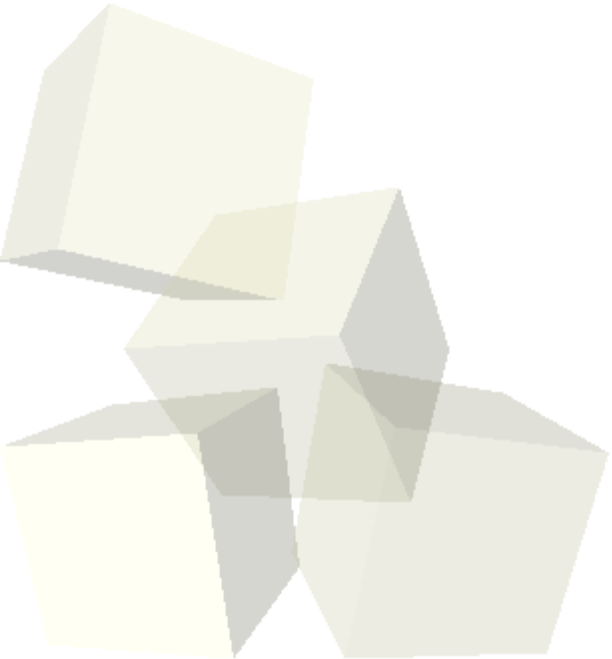


Operating Systems - Advanced

User-mode Linux

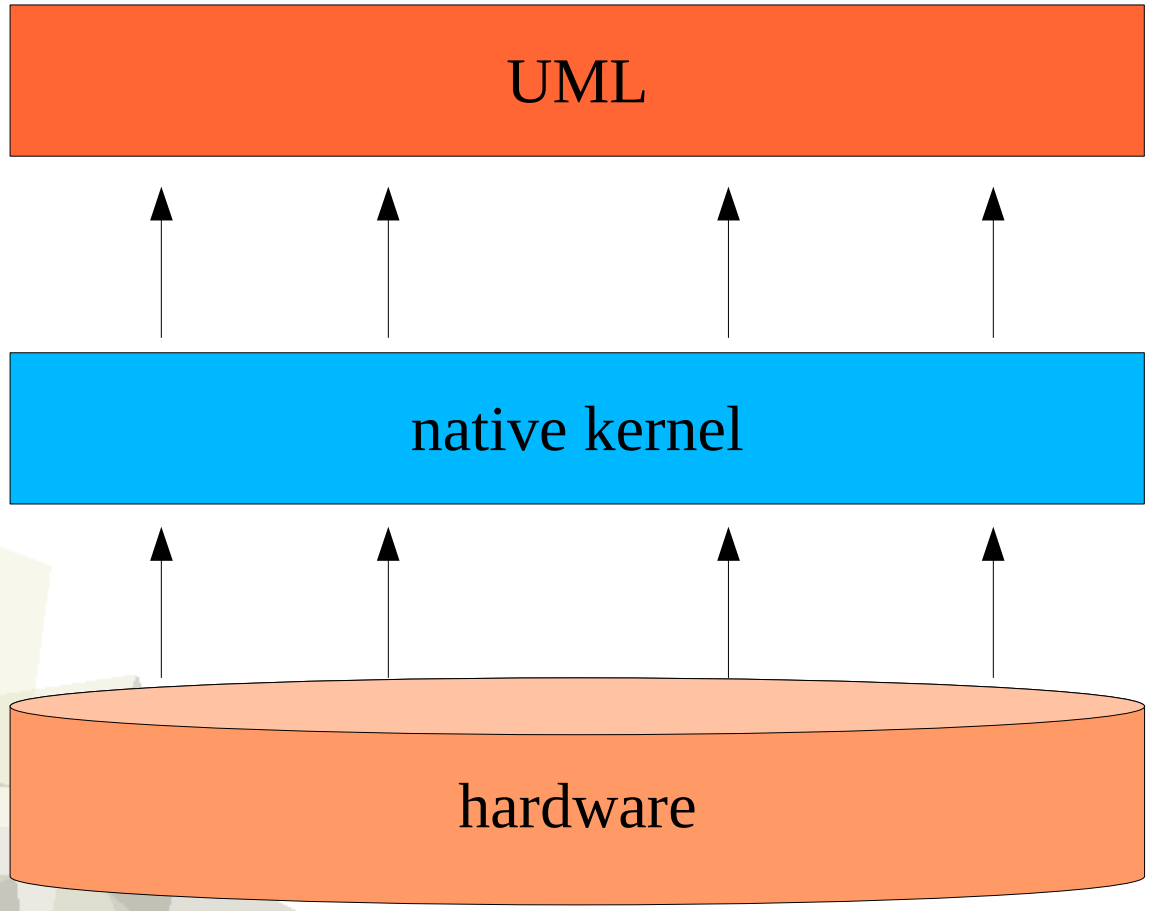
Jeff Dike
jdike@karaya.com



UML

- Linux kernel port on Linux
- User-space virtual machine
- Simulated hardware built on top of the native kernel interface
- UML kernel ported on top of native kernel system calls
- $linux_src/arch/um
- No drivers
- Isolated environment for processes

UML (2)



system calls

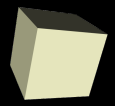
hardware interface

Devices

- console
  - the main console is the one in which the UML kernel was started
  - subsequent consoles run inside an xterm
- block devices
  - emulated through

Devices (2)

- serial links
  - emulated through pseudo-terminal slaves (/dev/pts/0)
  - serial connection through terminals
- network devices
  - specialized daemon sends Ethernet frames between virtual machines
  - can link virtual device to the real one



Processor

- Implemented using arch interface
- The code is a separate architecture named “um”

```
# ls /usr/src/linux/arch/um/
Kconfig         Kconfig.x86_64     Makefile-skas     [...]
Kconfig.char    Makefile           Makefile-tt       [...]
Kconfig.debug   Makefile-i386      Makefile-x86_64   os-Linux
Kconfig.i386    Makefile-ia64      config.release    scripts
Kconfig.net     Makefile-os-Linux  defconfig         sys-i386
Kconfig.scsi    Makefile-ppc       drivers           sys-ia64
```

- User-space code must run unmodified inside the virtual machine
- System calls are intercepted and run on the virtual machine
- An UML instance runs as a single process on the native kernel

System call interception

- ptrace – controlled execution
  - gdb uses ptrace
- A specialized thread uses ptrace to control other threads and processes
- Thread is notified when another thread issues a system call
  - gathers system call id and arguments
  - redirects to virtual machine kernel code (kernel code running in user-space)

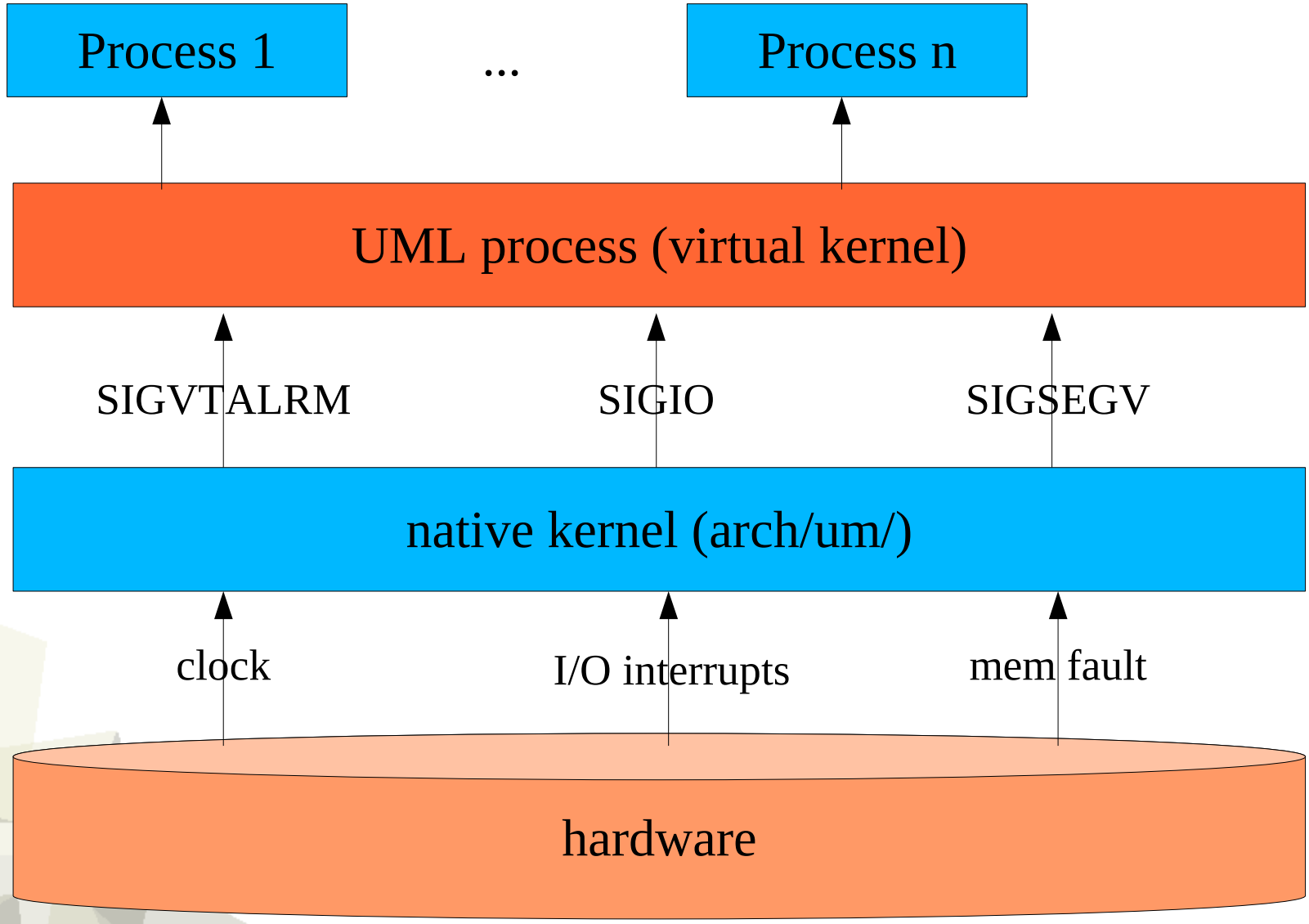
Trap

- A trap may cause the switch from user-mode to kernel-mode
- Issued by hardware (on physical systems)
  - The processor is forced to jump to a certain address in kernel-space
- Implemented in UML through Unix signals
  - SIGALRM/SIGVTALRM – clock
  - SIGIO – hardware interrupts
  - SIGSEGV - memory faults
  - UML kernel defines signal handlers (similar to trap/interrupt/exception handler)

Trap (2)

- Traps run in kernel-mode
- Signal handlers need to run in UML kernel-mode
  - 1. need to use a kernel stack
  - 2. system call interception must be disabled

Trap (3)


Process 1
...
Process n
UML process (virtual kernel)
SIGVTALRM
SIGIO
SIGSEGV
native kernel (arch/um/)
clock
I/O interrupts
mem fault
hardware

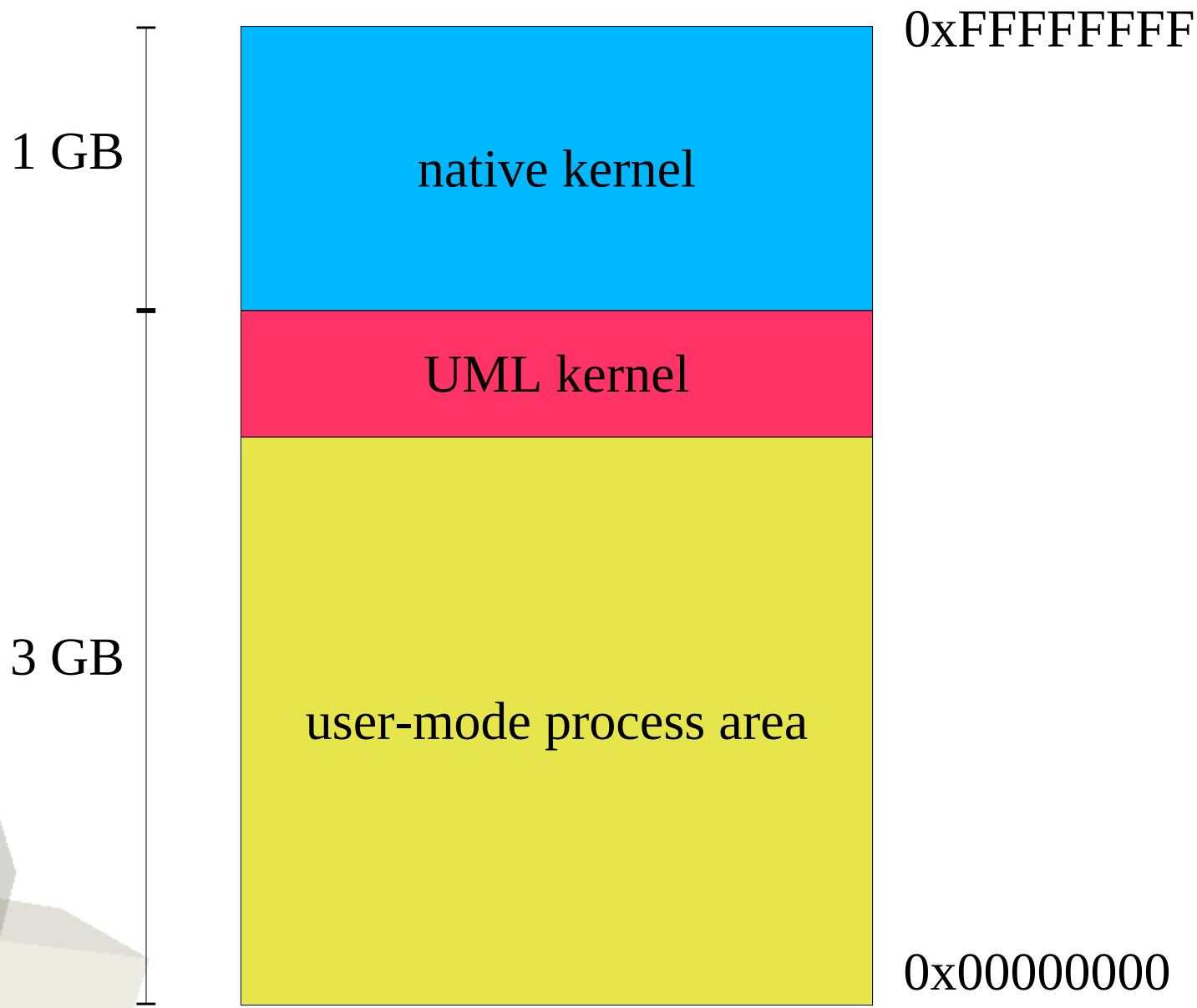
Kernel address space

- Separate memory address space for user-mode and kernel-mode
- UML problem
  - process and kernel co-exist in the same address space (the UML process address space)
- Solution
  - place UML kernel code in an unlikely-to-be-accessed memory area (0xa0000000 – 0xbfffffff)
  - mmap a file in each UML process address space space

Process address space

- A real process is created for each virtual process
- All processes share virtual kernel data
  - map a file containing kernel data in the address space of each process (shared segment)
- Virtual context switches use real context switches on native kernel
- When does a context switch occur?
  - time interrupt in the real kernel
  - SIGVTALRM in the UML kernel

Address space

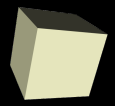


Initialization and shutdown

- Sample run

```
./linux-2.6.19-rc5 ubda=FedoraCore5-x86-root_fs mem=128M
```

- Arguments are sent through a specialized buffer
- Initialize memory, start idle thread
  - start_kernel, mem_init, paging_init
  - register and initialize drivers
- Idle thread becomes monitoring thread (MT)
  - MT starts system call interception
- On shutdown, all processes and threads are killed



Processes

- Process creation is handled by arch/um code
  - generic kernel code calls architecture-specific code
- Architecture-specific code (arch/um) creates a process on the host system
- MT is acting!
  - the new process does initialization (signal handlers, etc.)
  - sends itself a SIGSTOP
  - MT detects the process stop
  - MT ends the system call and returns the fork return value
- Process end
  - process is killed

Processes (2)

- Virtual system

```
uml:~# ps -eH
PID TTY          TIME CMD
    1 ?        00:00:00 init
    2 ?        00:00:00   ksoftirqd/0
    3 ?        00:00:00   watchdog/0
    4 ?        00:00:00   events/0
    5 ?        00:00:00   khelper
    6 ?        00:00:00   kthread
    7 ?        00:00:00     kblockd/0
   31 ?        00:00:00     pdflush
   32 ?        00:00:00     pdflush
   33 ?        00:00:00     kswapd0
   34 ?        00:00:00     aio/0
  592 ?        00:00:00     kcryptd/0
  679 ?        00:00:00     kjournald
  975 ?        00:00:00   syslogd
  981 ?        00:00:00   klogd
 1000 ?        00:00:00   cron
 1022 tty0     00:00:00   login
 1023 tty0     00:00:00     bash
 1032 tty0     00:00:00       vim
 1034 tty0     00:00:00         ps
```

- Physical system

```
ragnarok:~$ ps -C linux -H
  PID TTY          TIME CMD
21084 pts/1    00:00:10 linux
21928 pts/1    00:00:00   linux
21929 pts/1    00:00:00   linux
21930 pts/1    00:00:00   linux
21931 pts/1    00:00:00   linux
22469 pts/1    00:00:00   linux
22479 pts/1    00:00:00   linux
22511 pts/1    00:00:00   linux
22553 pts/1    00:00:00   linux
22555 pts/1    00:00:00   linux
22576 pts/1    00:00:00   linux
```

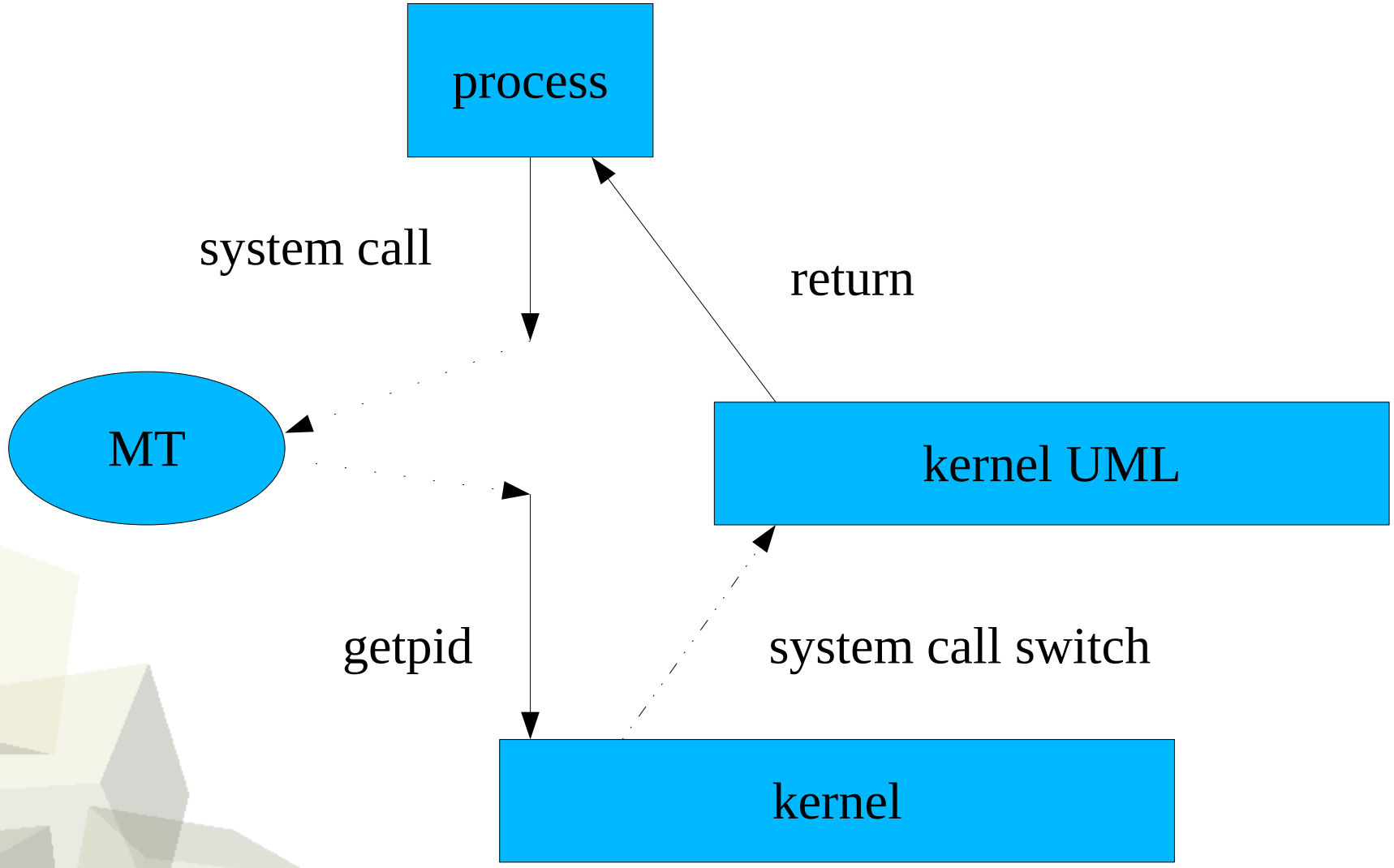
System calls

- Virtualized through MT
  - system calls are redirected to virtual kernel
  - system call is mapped to a “getpid” on the host system

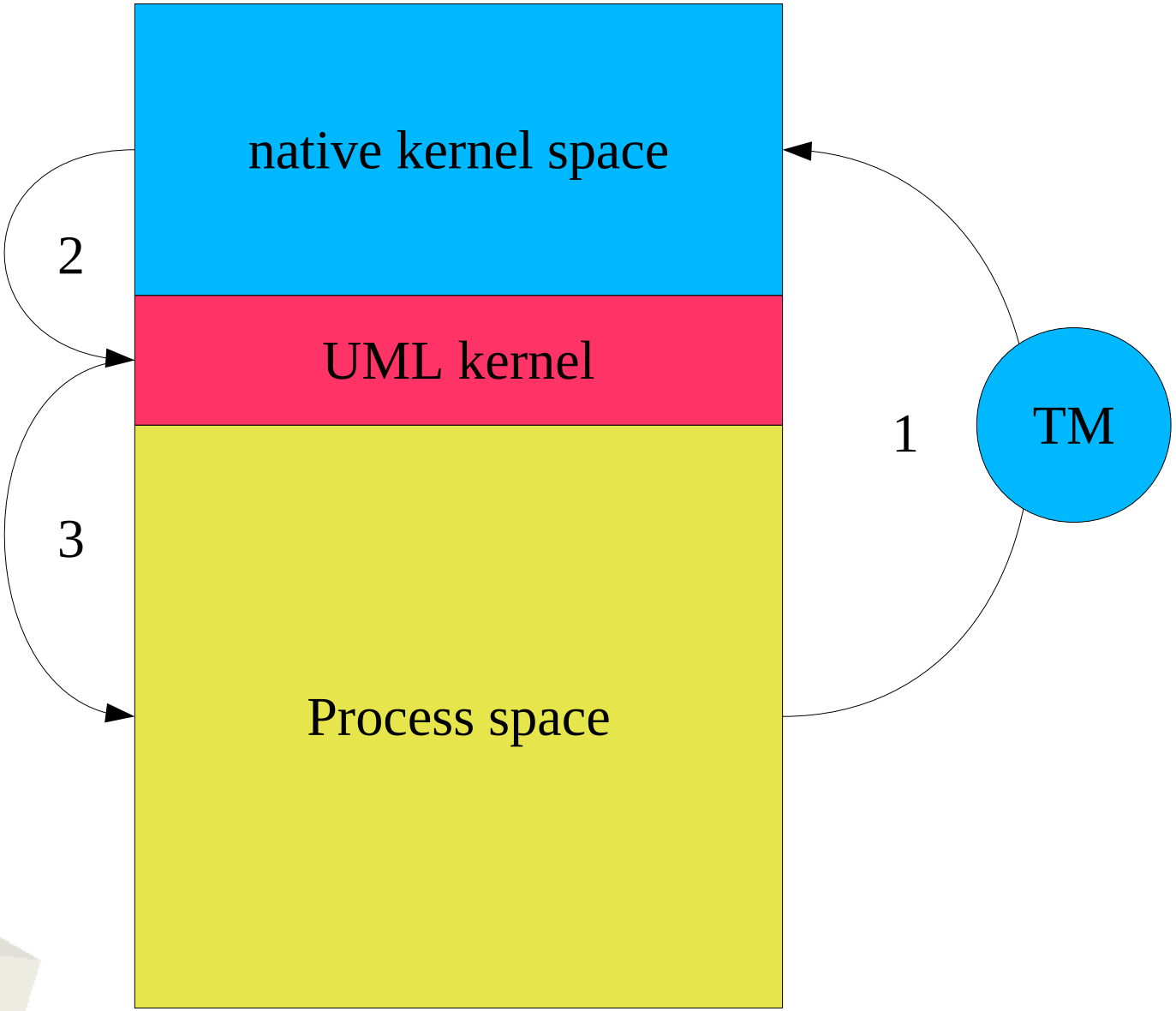
System calls (2)

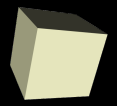
- How to call system call switch on the kernel stack?
  - 1. create an execution context positioning the process at the beginning of the switch statement
  - 2. use a signal when returning from kernel space; the signal handler is the execution context for the system call switch
- MT is notified at the end of the system call handler (in the virtual kernel)
- MT stores the return value in a specialized register
- Process continues the execution of user-space code

System calls (3)



System calls (4)





Context switch

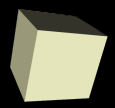
- Linux approach
  - call schedule()
  - choose a new process
  - call architecture-dependent code
- UML (arch/um)
  - notify MT
  - TM stops process and starts the new one

Context switch (2)

- Some pages may be swapped (mapping is still there)
- Swapped pages are stored in a circular buffer
- After scheduling buffer is checked and address space is updated

Signal delivery

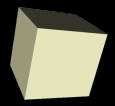
- Linux signals are stored in a queue in the process' task_struct
- The queue is checked on every kernel-mode exit
- Signal is delivered to the host process through SIGUSR2
- The SIGUSR2 handler runs the actual signal handler



Memory faults

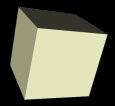
- What is demand paging?
- Memory fault
  - page fault (physical system)
  - SIGSEGV (virtual system)
- Fault handler checks for user-mode fault or kernel-mode fault
- Valid page → page mapping
- Invalid page → SIGSEGV to user process, or kernel oops to kernel
- An exception: passing an invalid point argument from user-space to kernel-space
  - check the address of the instruction generating the fault (exception tables)

- cp -r arch/i386 arch/um
- i386
  - handle through do_irq
- um
  - use SIGIO
  - choose appropriate handler through the device's simulated file descriptor



What have we got?

- A Linux virtual machine running on top of a Linux host OS
- Native applications run unmodified on UML
- Uses the most recent kernel (when compared to other virtualization solutions)
- Starting from 2.6, the um architecture is included in kernel code
- Starting from 2.6, skas (separate kernel address space) replaced MT



Use

- Kernel development/debugging
- Isolation
- prototyping (test on virtual system before deploying on physical system)
- Multiple development environments on the same system

Useful links

- http://www.usenix.org/publications/library/proceedings/als00/2000papers/papers/full_papers/dike/
- https://www.usenix.org/publications/library/proceedings/als01/full_papers/dike/
- http://user-mode-linux.sourceforge.net/
- http://user-mode-linux.sourceforge.net/old/UserModeLinux-HOWTO.html
- http://www.coherenthosting.com/prj/uml/henrique/pool_h01/