# Shell and Virtual Machines

Peter J. Denning

# Purpose of Shell

- User interface to the kernel
- Responds to single-line commands to run programs
- A command line names a program to be executed and specifies the files or devices used for its input and output
  - command = name args input output
  - if args, input, or output is not specified, default applies

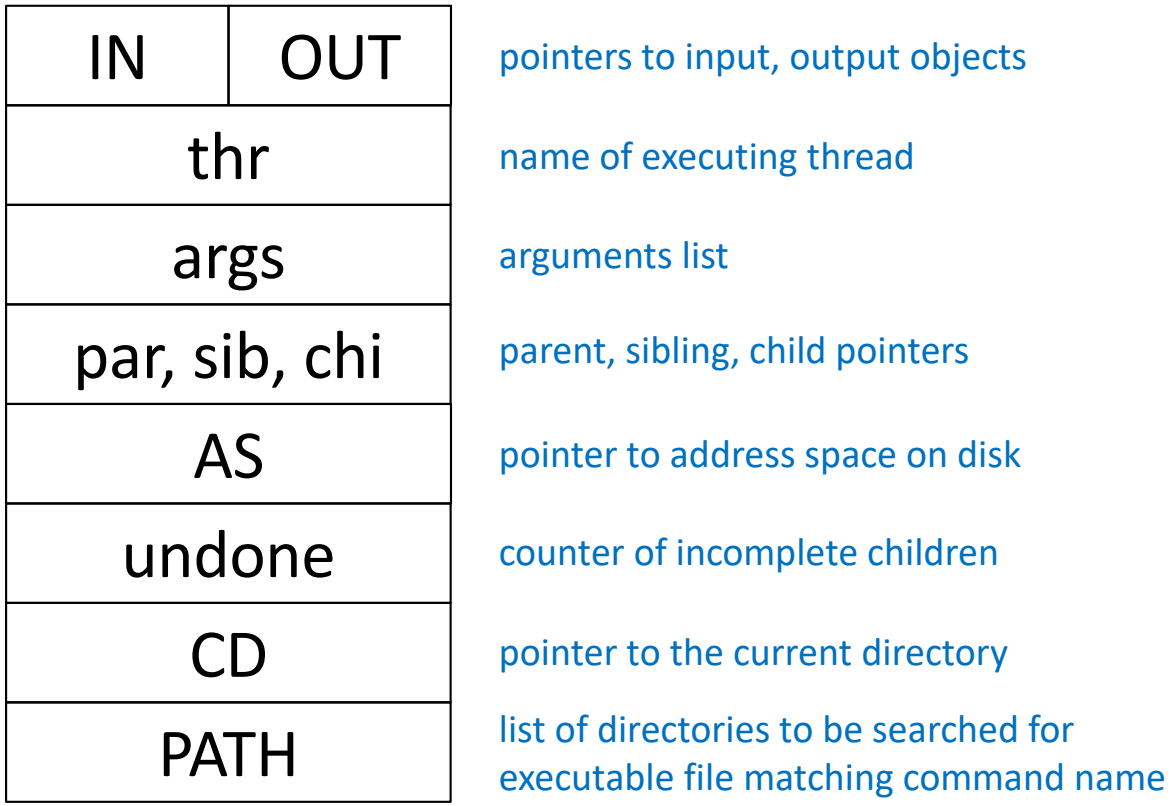
# Execution of Commands

- Shell parses command line into its four components
- Loads named program into a virtual machine
- Sets up its input and output
- Starts the virtual machine and sleeps
- Virtual machine exit awakens the shell
- Cleans up the virtual machine
- Listens for next command

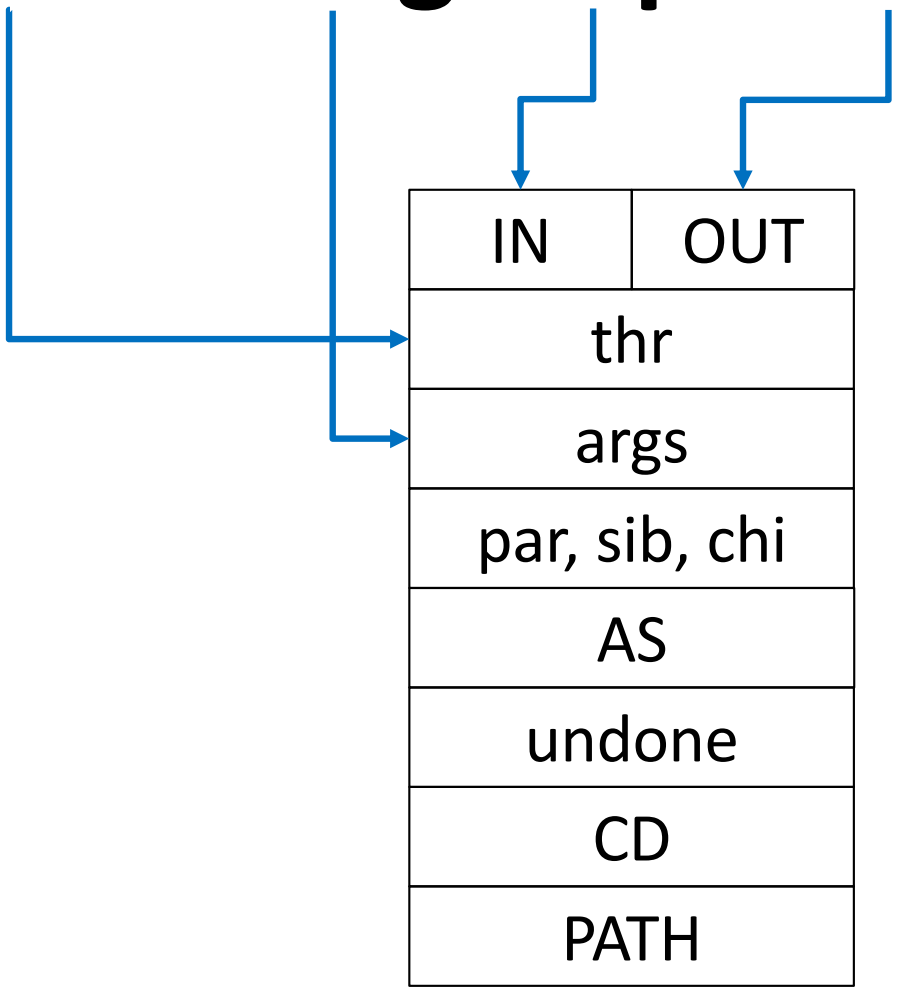
# Virtual Machines

- VMs are simulated computers that run programs
- Created and deleted at Level 8
  - Shell at Level 9 can call down to Level 8
- All Shell does is determine the contents of the VM and lets Level 8 run it while Shell sleeps

# VM template



# name args input output

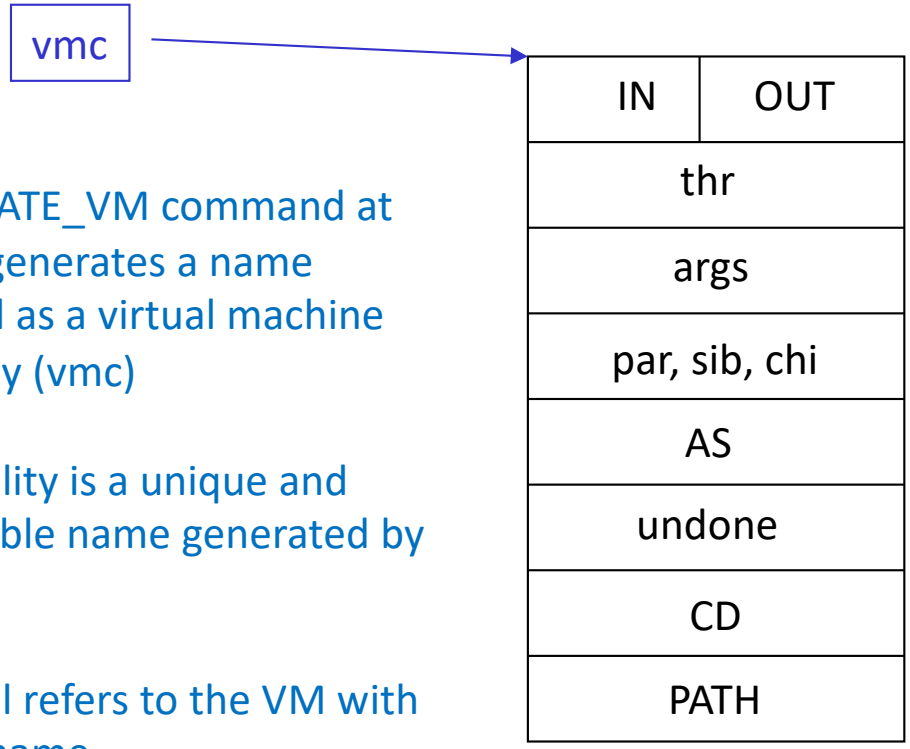



The Shell at Level 9 initializes a VM by giving the template information to Level 8 when it creates the VM

Shell starts VM, which sets undone=1 and Shell sleeps

Level 8 then executes the VM and when it is done EXITs the VM, which sets undone=0 and awakens the Shell

# Naming a VM




The CREATE_VM command at Level 8 generates a name encoded as a virtual machine capability (vmc)

A capability is a unique and unalterable name generated by the OS

The Shell refers to the VM with its vmc name

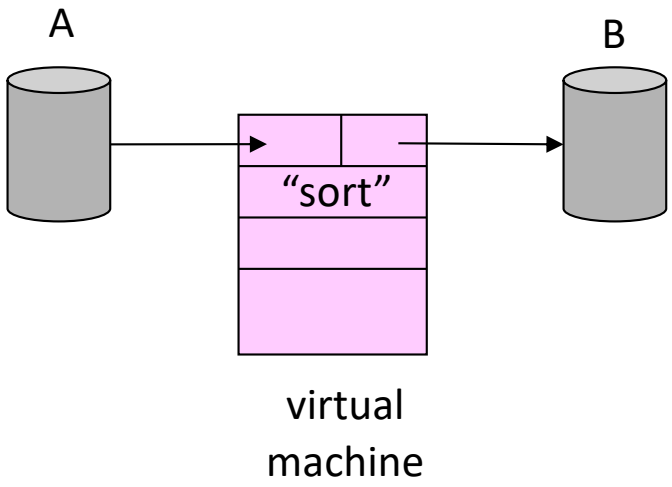
In Level 8, the virtual machine is represented internally by an VM control block formatted according to VM template

# Notation for an executing VM

EXAMPLE – USER TYPES SORT COMMAND:

sort A into B

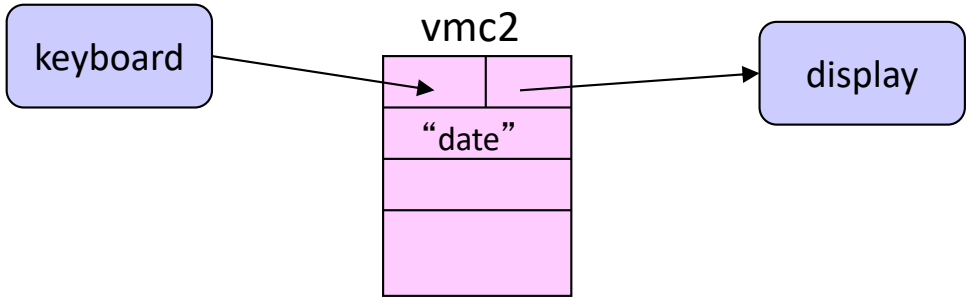
Shell creates VM running “sort” code, with file A as input and B as output



# Examples

The following slides consider examples of commands a user can type on the shell's command lines, and the corresponding virtual machines (VM) the shell asks the kernel to set up and execute.

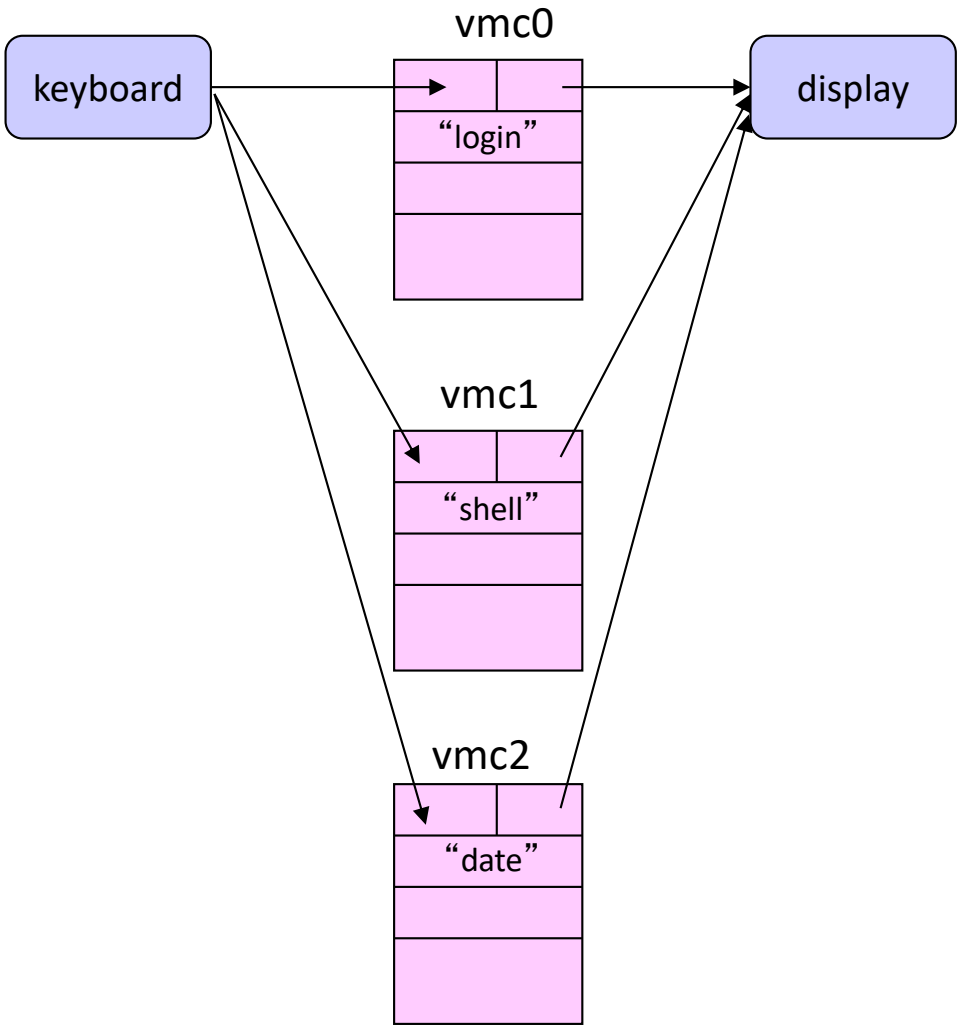
# date




Each arrow represents a capability. Here the IN port of vmc2 contains a capability for a keyboard. The OUT port contains a capability for the display.

The "date" code can read bits from its IN object by the statement READ(IN). It can send bits to its OUT object by the statement WRITE(OUT).

# date



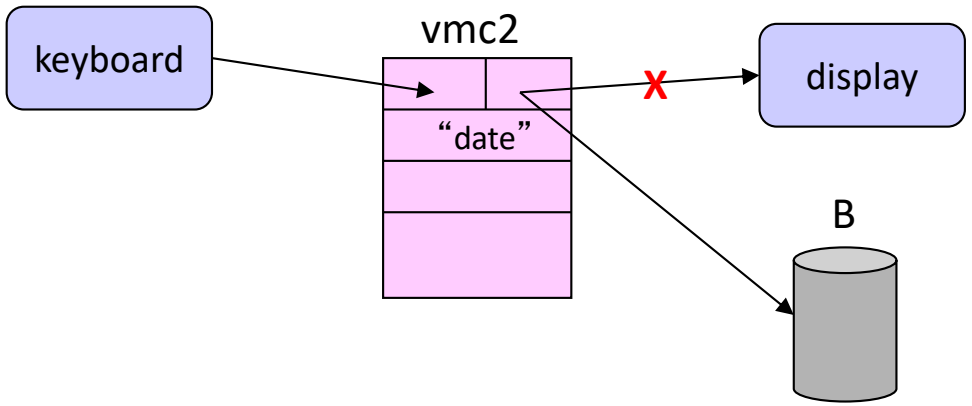

When the OS is started it creates a "login" VM (vmc0) attached to keyboard and display. When a user successfully types a user name and password, login creates a "shell" VM (vmc1) and goes to sleep.

The shell listens to the keyboard. When the user types the command "date", the shell creates a VM (vmc2) running "date" and goes to sleep.

Thus vmc0 is parent of vmc1, which in turn is parent of vmc2. The "par" and "chi" pointers in the VM designate the parent and child VMs, respectively.

Each VM inherits the IN and OUT objects of its parent. There is no conflict because parents sleep until all their children complete. When "date" runs, the "shell" and "login" sleep.

# date > B




By default, a new VM inherits the standard IN and OUT of its parent.

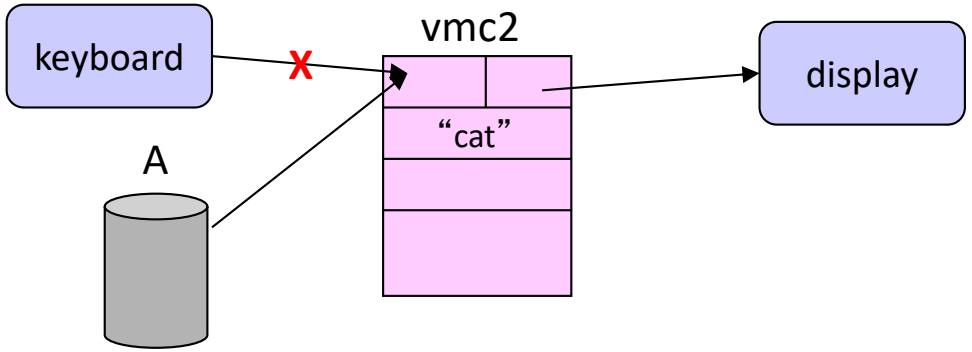
Typically IN = keyboard and OUT = display.

If different IN or OUT is needed, shell provides capabilities to the alternate input and output objects.

Here, notation ">B" tells the shell to place in OUT a capability for file B, replacing the display capability. The output of "date" will be go into file B.

The notation ">B" is called an output redirection.

# cat < A

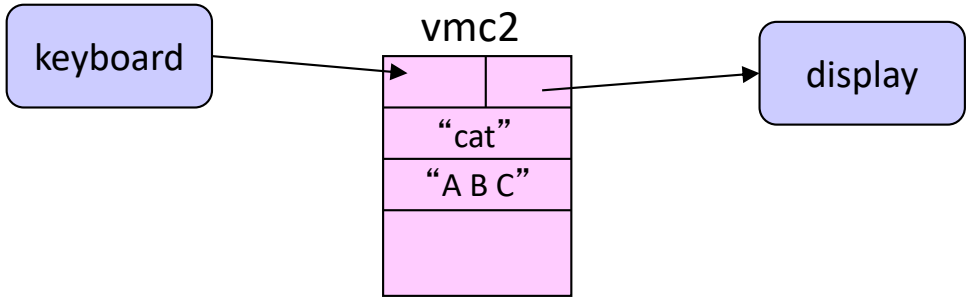



"cat" is a program that copies its input directly to its output. Here the user, with the notation "<A", has directed that a capability for file A be placed in IN, replacing the keyboard capability.

"cat" will copy the entire contents of A to the display.

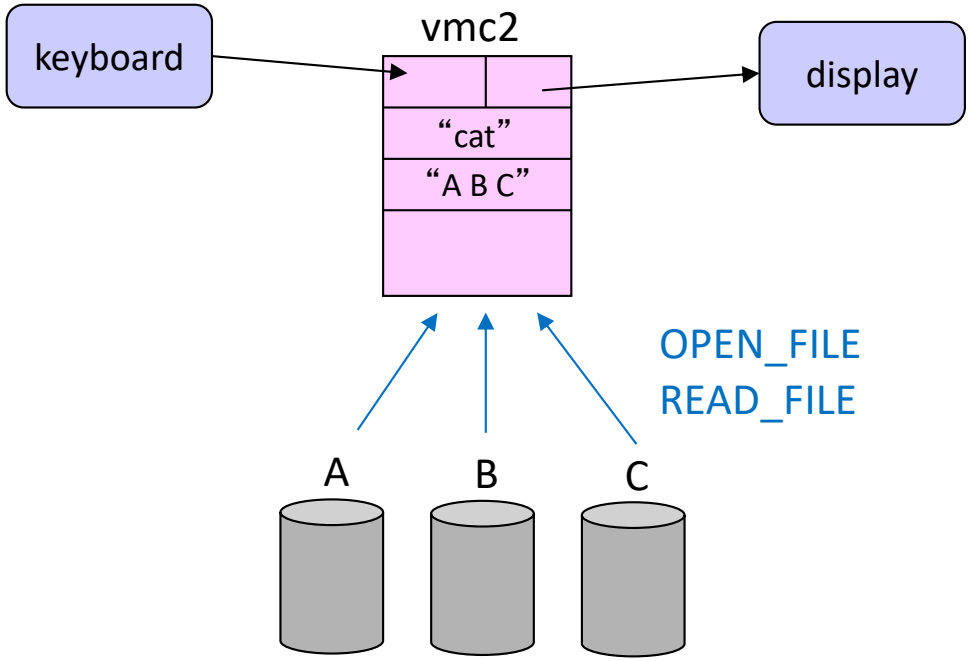
The notation "<A" is called an input redirection.

# cat A B C




“cat” can also concatenate (string together) a series of files specified as arguments. In this case the code of “cat” ignores its IN and instead opens the files A, B, C by calling the file manager. It then reads them in turn, copying their contents to OUT, which is the display.

# cat A B C

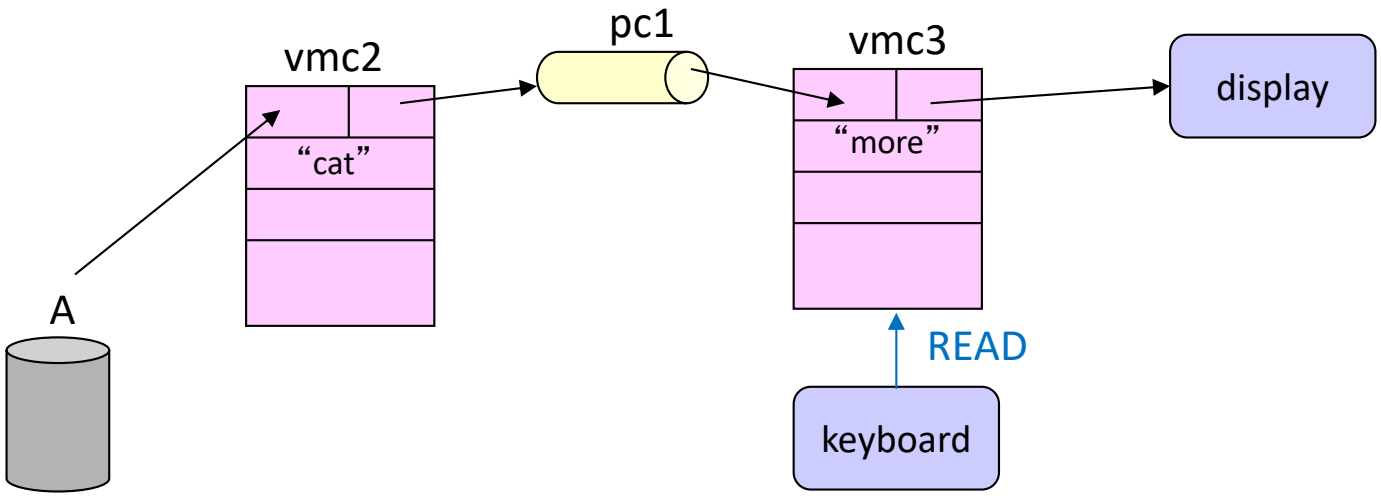



When executing, “cat” opens the files A, B, C by calling the file manager. It then reads them in turn, copying their contents to OUT. This picture shows the executing “cat” having opened and read the three argument files.

Note that the OPEN and READ commands are downward calls to the file manager. They are set up by the “cat” program, not the shell.

# `cat < A | more`

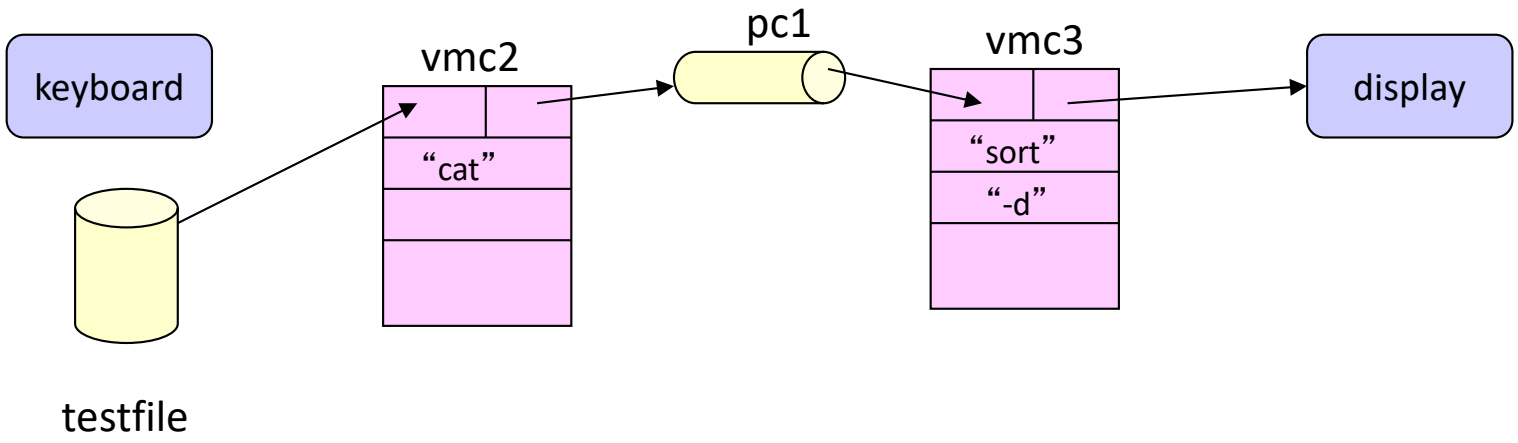
The pipe "|" symbol tells the shell that a second VM will process the output of the first. The pipe is an object that transmits a stream of bits from one VM's OUT to the next VM's IN.




If file A is larger than the display, the user will see many lines scroll by rapidly until "cat" finishes.

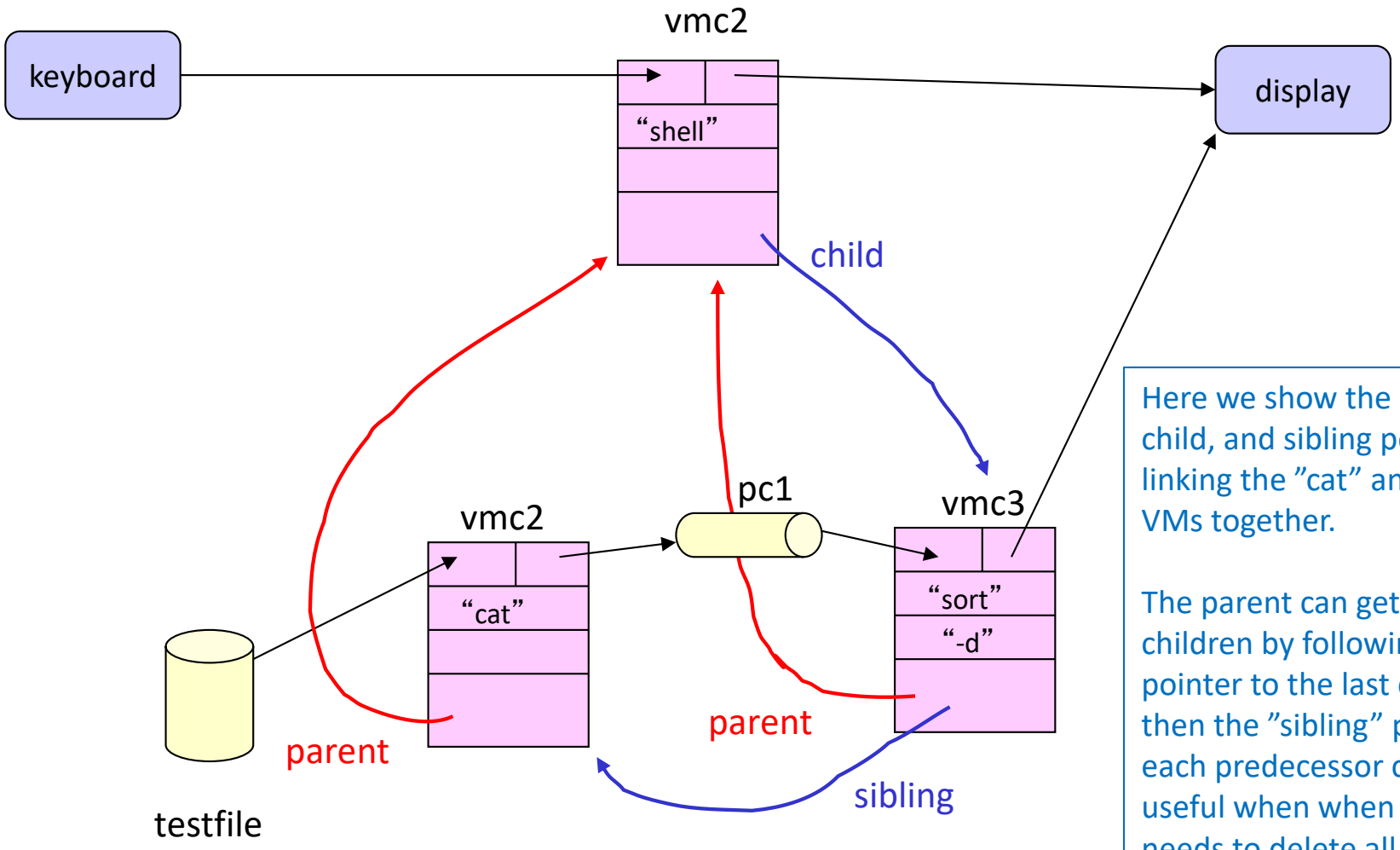
The program "more" chunks the output stream into pieces that fit display screens and sends one piece at a time; it pauses until the user types any key. "more" opens the keyboard and reads it using downward calls to the keyboard device driver. Note that the READ call on the keyboard is set up by the "more" programming, not the shell.

# cat < testfile | sort -d



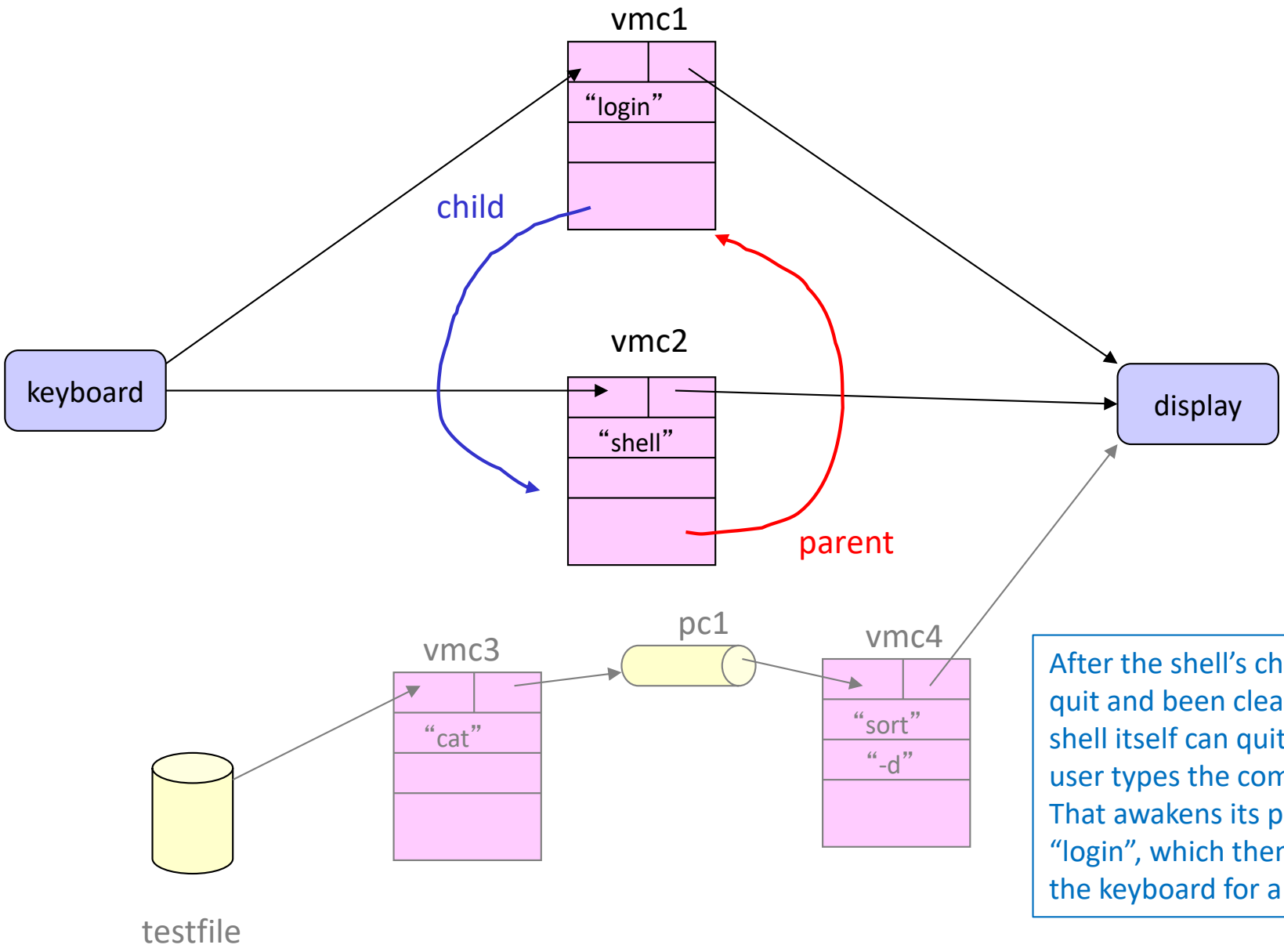

This command line copies the contents of "testfile" to its OUT. That stream is read by "sort" via its IN. "sort" rearranges the lines of the input into descending alphabetic order ("-d") of their first words. "sort" sends that result to the display via its OUT.

# `cat < testfile | sort -d`




Here we show the parent, child, and sibling pointers linking the "cat" and "sort" VMs together.

The parent can get a list of its children by following its "child" pointer to the last child and then the "sibling" pointers to each predecessor child. This is useful when when the parent needs to delete all the children VMs after they have completed their tasks.




After the shell's children have quit and been cleaned up, the shell itself can quit when the user types the command "exit". That awakens its parent, "login", which then listens to the keyboard for a new login.

# Summary

- Commands to shell specify a series of one or more virtual machines in a pipeline
- The parent process (shell) sleeps until all its children are done
- When a VM pipeline is done, its parent cleans up by deleting all its component VMs and pipes, and by closing its open files