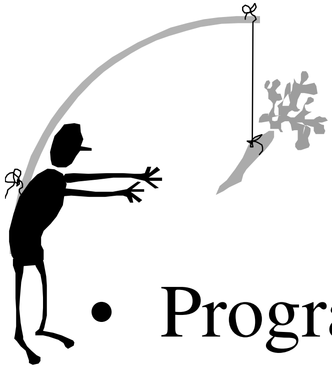
Unix Case Study

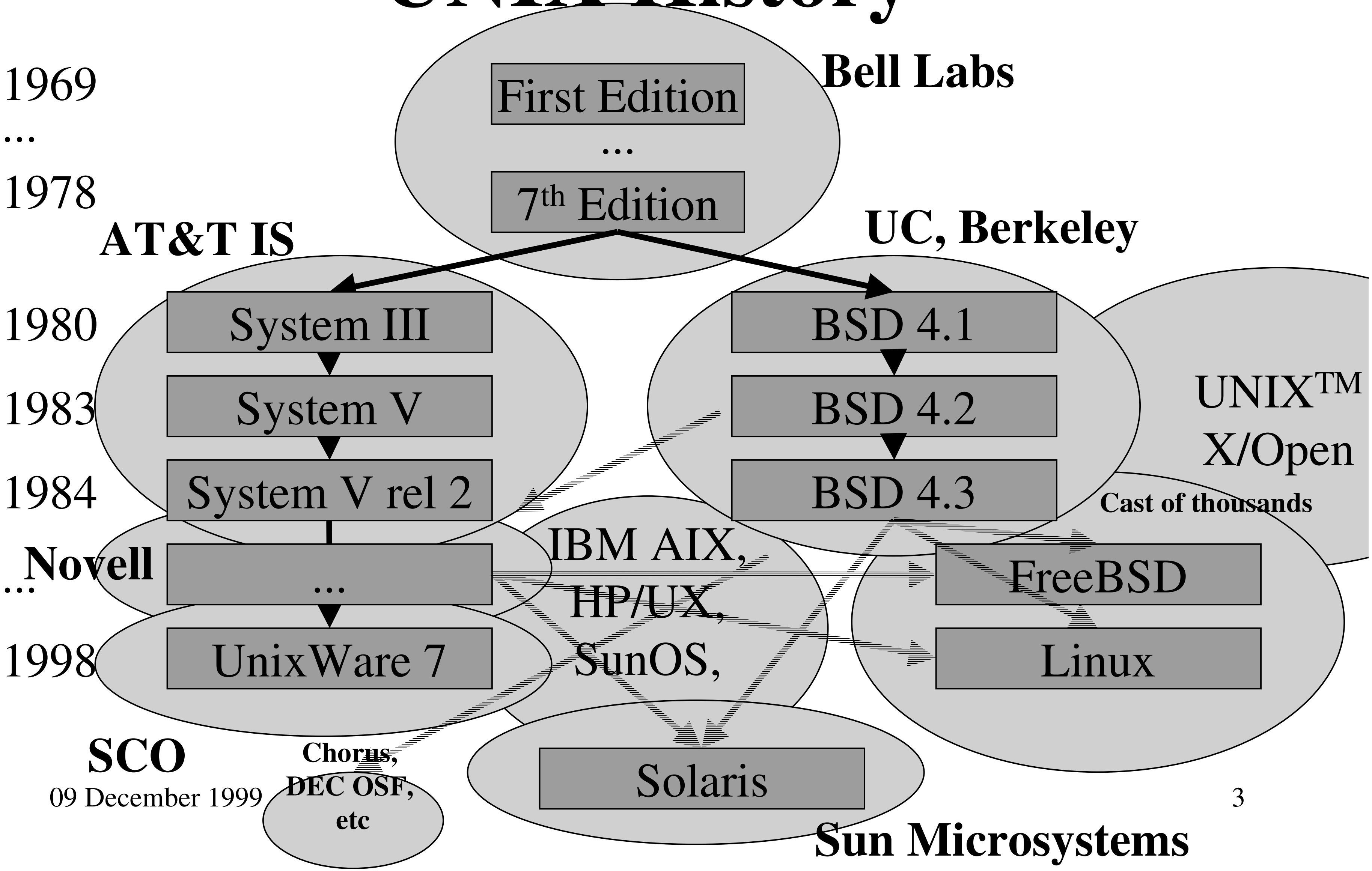
"The UNIX programming environment is unusually rich and productive." - Kernighan and Pike



UNIX Motivation

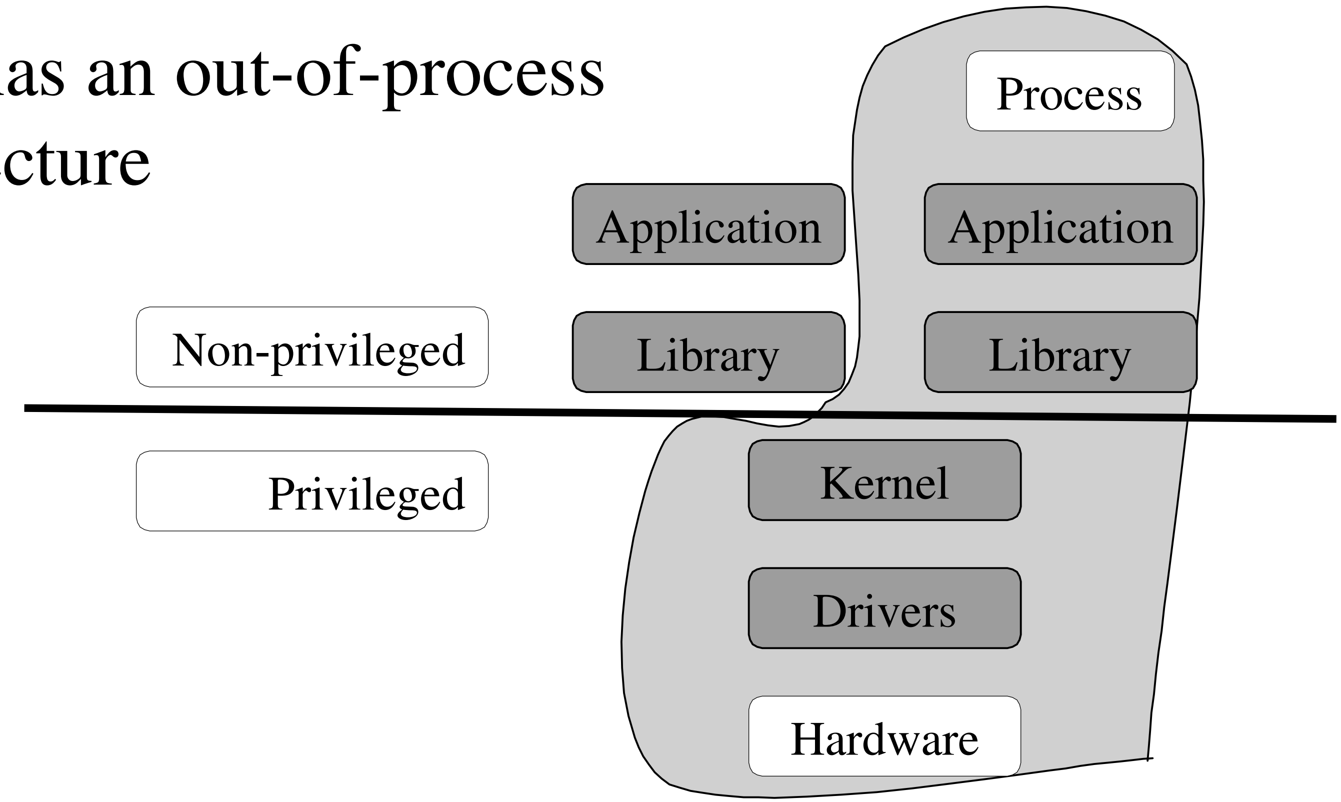
- Program Development Environment
- UNIX - weak pun on “MULTICS”
- Initially (1969) single-user, soon (1973) multi-user timesharing system
- Written in ‘C’
- Fundamental Architectural Design finished in 1978
 (7th Edition)
- Kernel, Environment + Software Tools

UNIX History



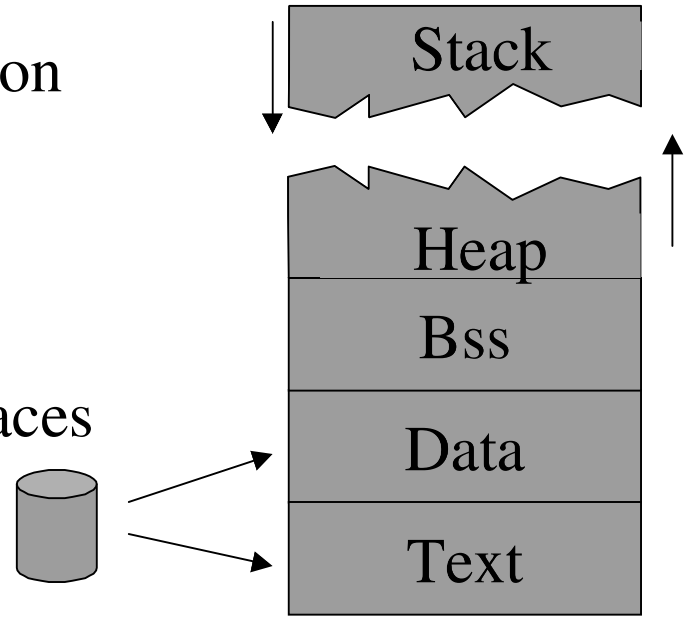
Machine Architecture - Overview

- Process Model:
  - Unix has an out-of-process architecture



Virtual Machine

- Memory <= 4Gb space, virtual (paged), software segments
  - Kernel (shared by all processes)
  - Text, Data, Heap, Stack
- User File ID's - handles for interaction
  - Files
  - Shared Memory Region
  - Network Connections
- System Calls - procedure-like interfaces
- Signals, Process ID's, etc.



Process Model

- Process Creation
  - result = fork()
- Executing a Program
  - result = exec(name, parameters)
- Wait for Child Process
  - result = sleep()
- Terminate
  - exit(n)

```
while (1) {
  read_user_command();
  if (fork()) {
    exec(command, params);
  } else {
    sleep();
}
```

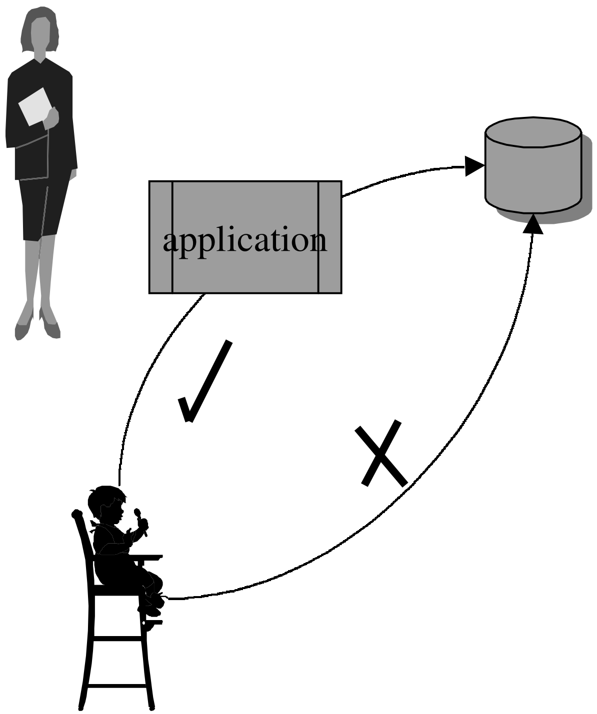
The Process

- Process has its own
  - address space
  - UFID's for interacting with files, networks, etc.
  - signal handling vector
  - UID and GIDs - user ID and group Ids
    - govern permission checks
  - A Flow of Control
    - Threads allow multiple flows of control in shared address space:
    - Limited Kernel support

- All inherited from parent:
  - memory is shared, but copy-on-write
  - shared files allow cooperation (e.g. redirection, pipes)

Protection Model

- Process Concepts:
    - UID
    - GID
- File Attributes: **`RWXRWXRWX`**
    - Owner
    - Group
    - Other
    - Read
    - Write
    - Execute
    - Set User ID
- Primarily files, directories
- Same mechanisms used by devices, network connections, shared libraries



APIs and Standards

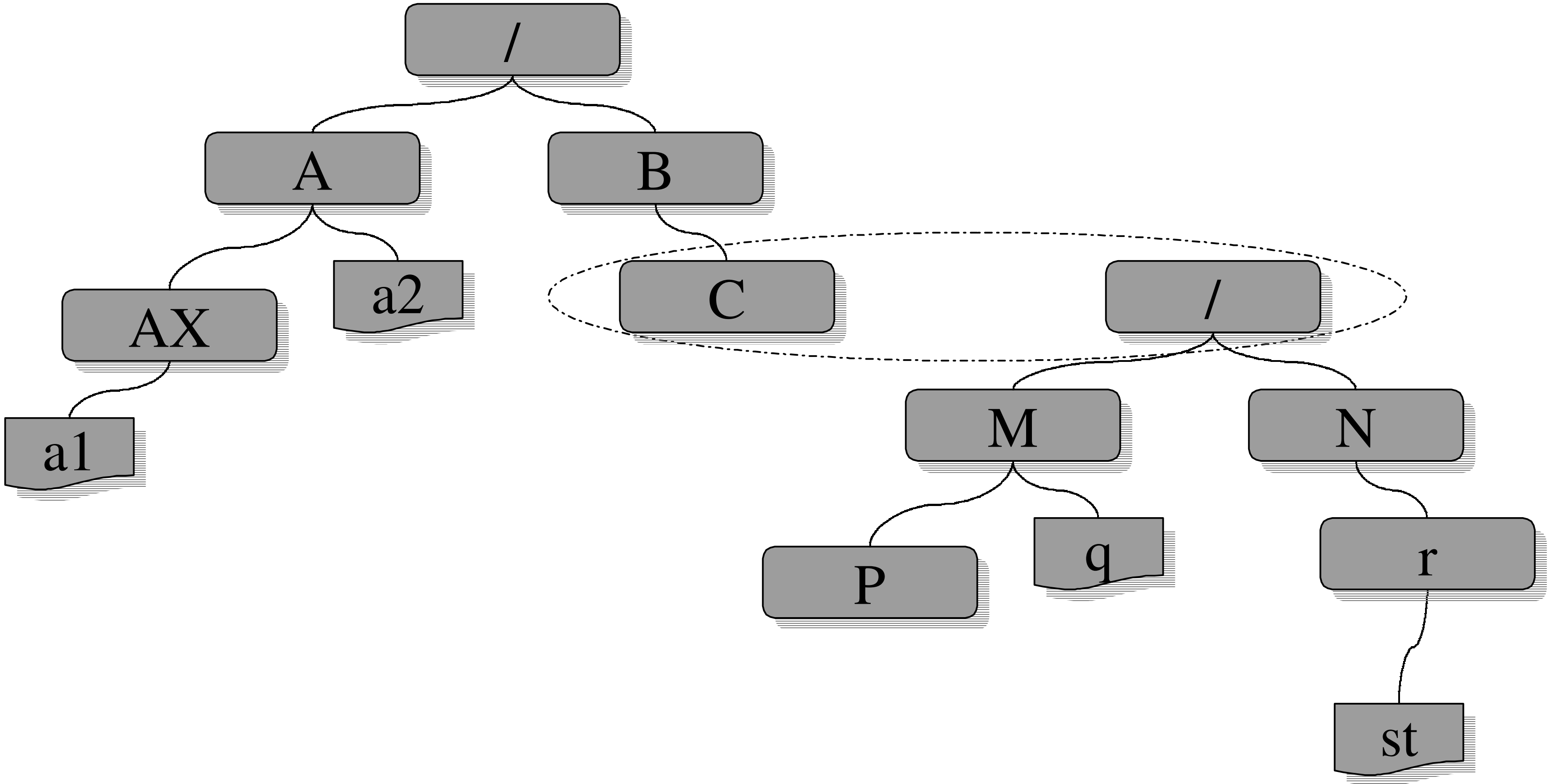
- POSIX (IEEE 1003)
  - Essentially 7th Edition system calls and library functions
  - BSD signals
  - New terminal handling
  - No GUI - character-based I/O only

- X Windows
  - Low-level protocol
  - Between Display Server and Computing Client
  - Look and Feel provided by library (e.g. Motif, Openwindows)
  - Many, replaceable clients, servers, window managers

- APIs available at Kernel (privileged) or Library (untrusted) levels

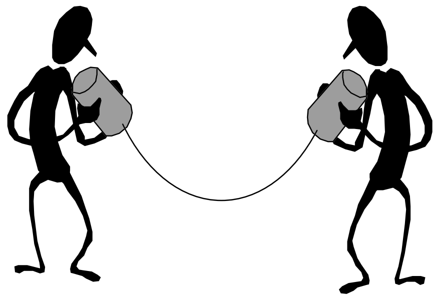
- WABI, SoftWindows, Wine emulation

File System: the *mount* concept



Inter Process Communication

- Pipes
  - composition of character-based programs
  - `cat $* | tr -cs [^A-Za-z] '\012' | \`
    `sort | uniq | comm -23 - /usr/dict/words`

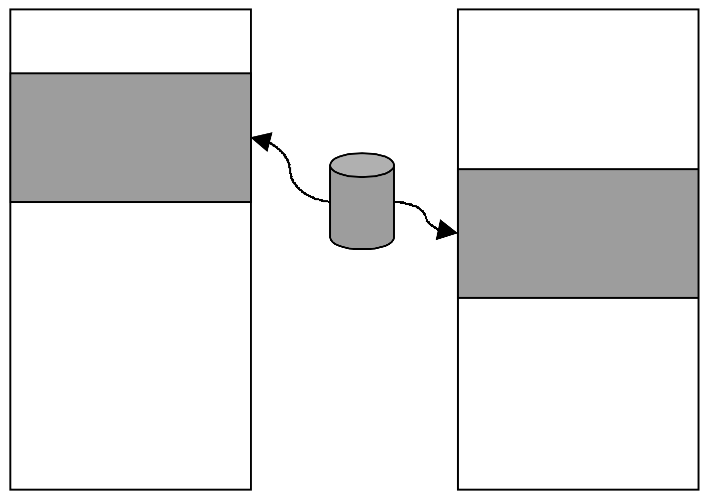


- Events
  - (Signals) - visible in non-privileged code
    - originated in program or kernel
    - individually ignored, caught or handled (recursively) per process

- Interrupts
  - Device-driver specific - visible only in privileged code

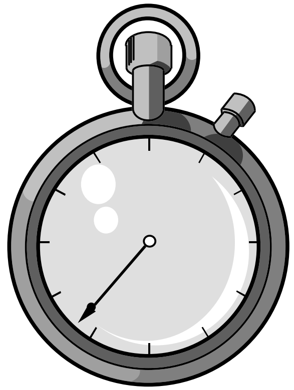
Inter Process Communication (II)

- Shared Memory
  - Generalization of mapping file into memory space
  - Naming, binding and protection via filestore permissions
  - Update semantics:
    - shared or private
    - persistent or transient



Synchronization

- Scheduling
  - "fair", timesharing dynamic priority scheme
  - Limited user assignment of priorities
  - Fair share scheduler for complex workloads
- Semaphores
  - Kernel and RPC facilities for locking
  - Applications may use filestore locks - e.g. mail
- Timers
  - One timer per process
  - Signal on timer elapse
  - plus profiling timer for performance measurement
  - plus system timers for accounting, job scheduling, etc.



Object Naming Conventions

- Files:
  - project/source/main.c
  - /usr/openwin3/bin/olwm

- Devices, printers, pipes, etc:
  - **`/dev/dsk/c0t0d0s`**
  - **`/dev/printers/lp3`**
  - **`lpr -PHP-3-duplex`**

Naming Other objects

- Comms lines: www.cs.man.ac.uk:80
- Signals (events) 1..31
- Processes: 1..64000
- Process groups: PID of lead process
- Sockets: (…)

Object Access Mechanisms

- For filestore-named objects

  ```
  – ufid = open(name, access);
  – nbytes = read(ufid, &buffer, bufsize);
  – nbytes = write(ufid, &buffer, nbytes);
  – close(ufid)
  ```

  – Protection as for files

- Object-specific methods for others
  – Protection is ad-hoc

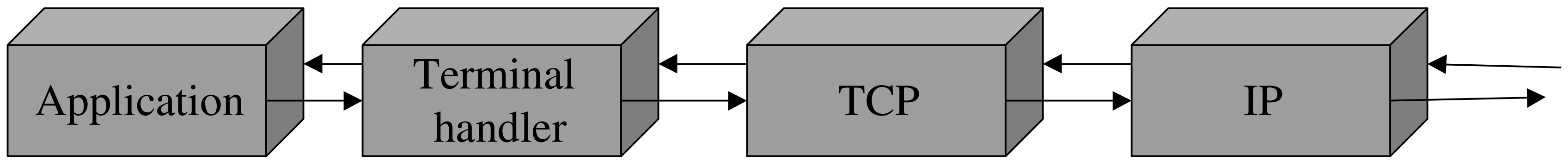
Identifying and binding to objects

- External binding
  - /dev/tty is a synonym in every process
  - Redirection:
    - `ls -l >../FILELIST`
    - `tr [A-Z] [a-z] < capitals > readable.txt`
  - Internal binding
    - `open("/usr/dict/words", 0);`
    - `signal(SIGHUP, SIG_IGN, NULL);`

- Types of file access
  - Default file access is sequential
  - Random access (to byte boundary) is supported
  - Non-sequential file access supported by libraries

I/O and Device Drivers

- Drivers are privileged code (not user-supplied)
- Driver *cannot* use application API
- Application::driver API is driver-specific
- "Streams" offer some rationality:



Device Drivers

- Filesystem Drivers: (e.g. disks, CDROM)
  - block I/O
  - cached in kernel: read ahead/write behind
  - generally synchronous transfers
  - (still byte addressable)

- Character I/O (e.g. modems, tapes, comms)
  - synchronous or asynchronous transfers
  - Unbuffered
  - 1 Application I/O :: 1 Device transfer

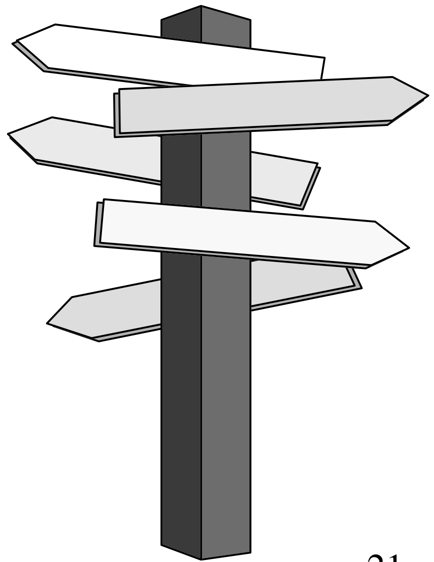
Networking:

- Interconnect technologies:
  - ISDN, Ethernet, Fast Ethernet, FDDI, etc.
  - PPP, DHCP

- Protocols:
  - TCP/IP, IPX, NetBIOS
  - NFS
  - SMB, lpr printing
  - etc.

Interaction Models

- User Interface Model
  - replaceable *shells* (1 per user?)
- Command Line Interface
  - ls, cd, cat, rm
- Scripting capabilities
  - for, while, if, trap
  - pipes, redirection
  - sed, awk, perl
  - invoking shell from C, C from shell
- Many different, alternative Explorer-type applications/environments



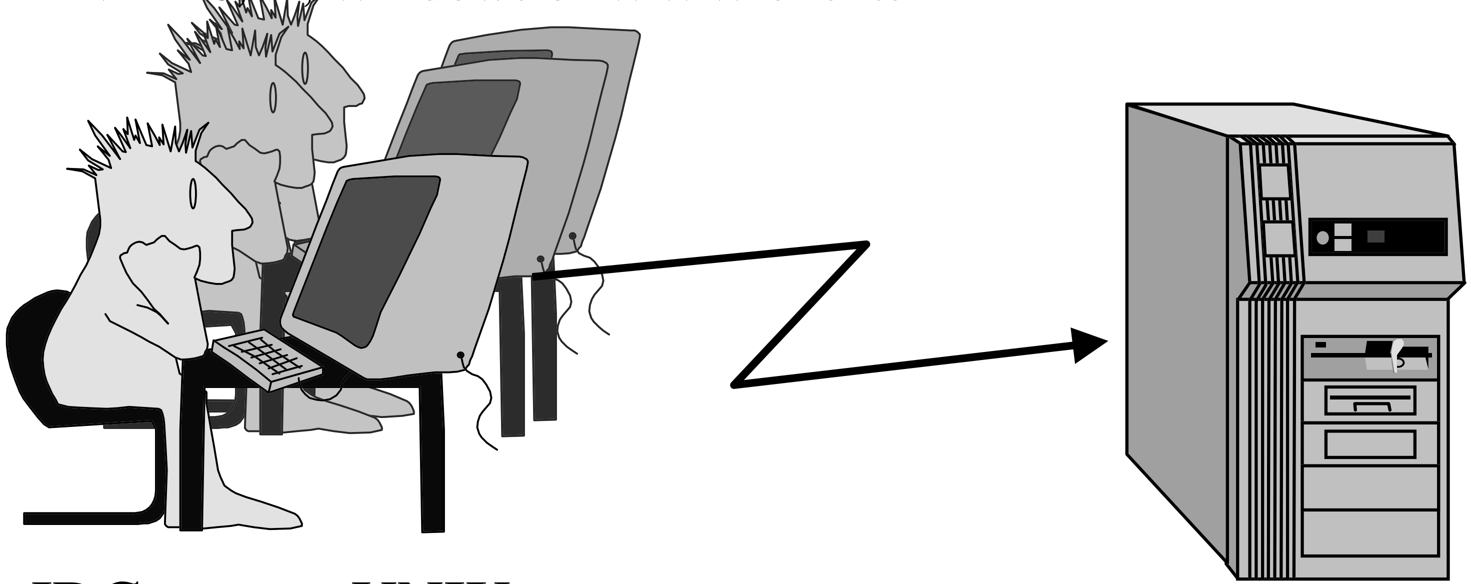
Process Scheduling

- System Initialization through multiple “run-levels”
  - e.g.
    - single user
    - initializing
    - daytime
    - nighttime
  - each run-level has own startup and shutdown scripts

- Delayed jobs:
  - at 0630 Monday week
    - for one-off deferral
  - cron
    - for regular jobs (hourly, daily, weekly, monthly, etc)

Typical Application Deployment Environment

- Typically deployed in “client/server” configuration
  - VT220, X, Windows or WWW clients



- IDC reports UNIX serves:
  - 83% of business databases
  - 55% of internet servers

Summary:

- UNIX is widespread in server marketplace

- Ideally suited to:
  - Software development
  - Rapid prototyping
  - Web servers
  - Team-level servers and larger

- Question marks:
  - scalability
  - GUI support
  - Real-time responses