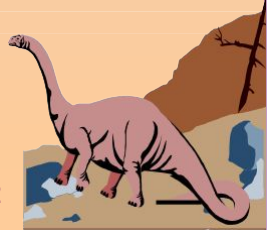




# Module 20: The Linux System

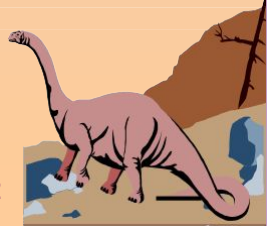
- History
- Design Principles
- Kernel Modules
- Process Management
- Scheduling
- Memory Management
- File Systems
- Input and Output
- Interprocess Communication
- Network Structure
- Security





# History

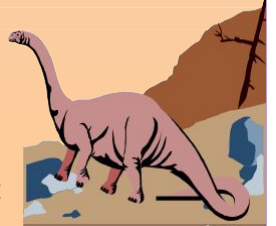
- Linux is a modern, free operating system based on UNIX standards.
- First developed as a small but self-contained kernel in 1991 by Linus Torvalds, with the major design goal of UNIX compatibility.
- Its history has been one of collaboration by many users from all around the world, corresponding almost exclusively over the Internet.
- It has been designed to run efficiently and reliably on common PC hardware, but also runs on a variety of other platforms.
- The core Linux operating system kernel is entirely original, but it can run much existing free UNIX software, resulting in an entire UNIX-compatible operating system free from proprietary code.





# The Linux Kernel

- Version 0.01 (May 1991) had no networking, ran only on 80386-compatible Intel processors and on PC hardware, had extremely limited device-driver support, and supported only the Minix file system.
- Linux 1.0 (March 1994) included these new features:
  - ◆ Support for UNIX's standard TCP/IP networking protocols
  - ◆ BSD-Berkeley Software Distribution-compatible socket interface for networking programming
  - ◆ Device-driver support for running IP over an Ethernet
  - ◆ Enhanced file system
  - ◆ Support for a range of SCSI(Small **C**omputerSystem Interface,) controllers for high-performance disk access
  - ◆ Extra hardware support
- Version 1.2 (March 1995) was the final PC-only Linux kernel.

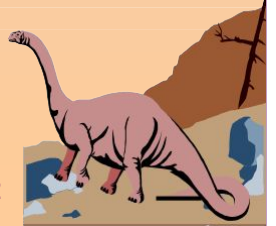





# Linux

## 2.0

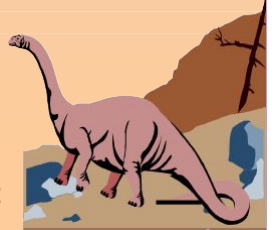
- Released in June 1996, 2.0 added two major new capabilities:
  - ◆ Support for multiple architectures, including a fully 64-bit native Alpha port.
  - ◆ Support for multiprocessor architectures
- Other new features included:
  - ◆ Improved memory-management code
  - ◆ Improved TCP/IP performance
  - ◆ Support for internal kernel threads, for handling dependencies between loadable modules, and for automatic loading of modules on demand.
  - ◆ Standardized configuration interface
- Available for Motorola 68000-series processors, Sun Sparc systems, and for PC and PowerMac systems.






# The Linux System

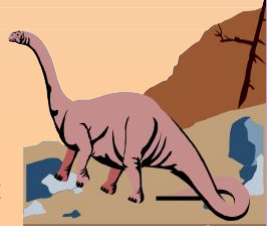
- Linux uses many tools developed as part of Berkeley's BSD operating system, MIT's X Window System, (supports graphic system) and the Free Software Foundation's GNU project.
- The main system libraries were started by the GNU project, with improvements provided by the Linux community.
- Linux networking-administration tools were derived from 4.3BSD code; recent BSD derivatives such as Free BSD have borrowed code from Linux in return.
- The Linux system is maintained by a loose network of developers collaborating over the Internet, with a small number of public ftp sites acting as de facto standard repositories.






# Linux Distributions

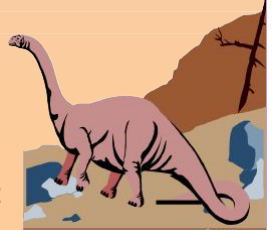
- Standard, precompiled sets of packages, or *distributions*, include the basic Linux system, system installation and management utilities, and ready-to-install packages of common UNIX tools.
- The first distributions managed these packages by simply providing a means of unpacking all the files into the appropriate places; modern distributions include advanced package management.
- Early distributions included SLS(softlanding **Linux** System (**SLS**) was one of the first **Linux** distributions.) and Slackware.*Red Hat* and *Debian* are popular distributions from commercial and noncommercial sources, respectively.
- The RPM Package file format permits compatibility among the various Linux distributions.






# Linux Licensing

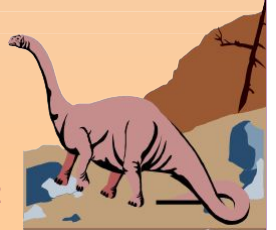
- The Linux kernel is distributed under the GNU General Public License (GPL), the terms of which are set out by the Free Software Foundation.
- Anyone using Linux, or creating their own derivative of Linux, may not make the derived product proprietary; software released under the GPL may not be redistributed as a binary-only product.





# Design Principles

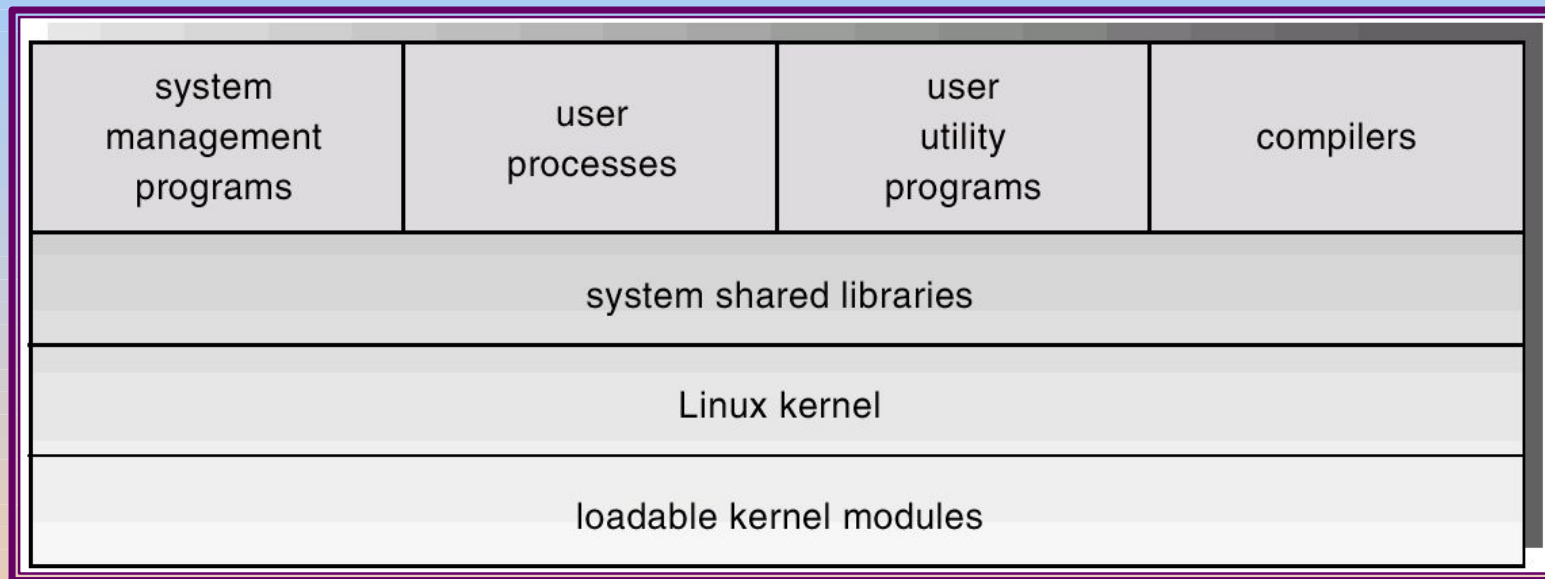
- Linux is a multiuser, multitasking system with a full set of UNIX-compatible tools..
- Its file system adheres to traditional UNIX semantics, and it fully implements the standard UNIX networking model.
- Main design goals are speed, efficiency, and standardization.
- Linux is designed to be compliant with the relevant POSIX (The Portable Operating System Interface) documents; at least two Linux distributions have achieved official POSIX certification.
- The Linux programming interface adheres to the SVR4 (**System V Release 4**, )UNIX semantics, rather than to BSD behavior.







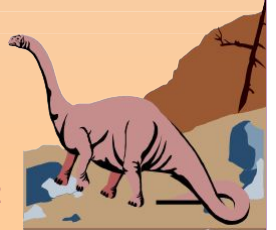
# Components of a Linux System

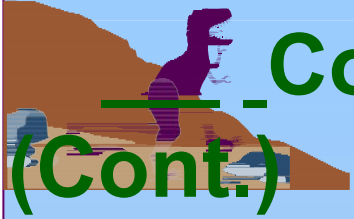




# Components of a Linux System (Cont.)

- Like most UNIX implementations, Linux is composed of three main bodies of code; the most important distinction between the kernel and all other components.
- The **kernel** is responsible for maintaining the important abstractions of the operating system.
  - ◆ Kernel code executes in *kernel mode* with full access to all the physical resources of the computer.
  - ◆ All kernel code and data structures are kept in the same single address space.

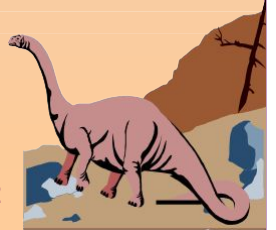





# Components of a Linux System

(Cont.)

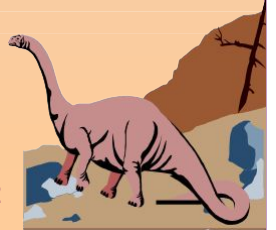
- The **system libraries** define a standard set of functions through which applications interact with the kernel, and which implement much of the operating-system functionality that does not need the full privileges of kernel code.
- The **system utilities** perform individual specialized management tasks.






# Kernel Modules

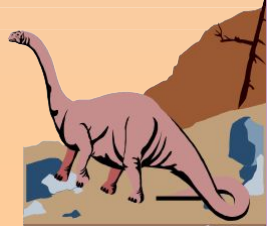
- Sections of kernel code that can be compiled, loaded, and unloaded independent of the rest of the kernel.
- A kernel module may typically implement a device driver, a file system, or a networking protocol.
- The module interface allows third parties to write and distribute, on their own terms, device drivers or file systems that could not be distributed under the GPL. **General Public License**
- Kernel modules allow a Linux system to be set up with a standard, minimal kernel, without any extra device drivers built in.
- Three components to Linux module support:
  - ◆ module management
  - ◆ driver registration
  - ◆ conflict resolution





# Module Management

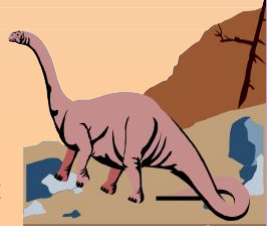
- Supports loading modules into memory and letting them talk to the rest of the kernel.
- Module loading is split into two separate sections:
  - ◆ Managing sections of module code in kernel memory
  - ◆ Handling symbols that modules are allowed to reference
- The module requestor manages loading requested, but currently unloaded, modules; it also regularly queries the kernel to see whether a dynamically loaded module is still in use, and will unload it when it is no longer actively needed.





# Driver Registration

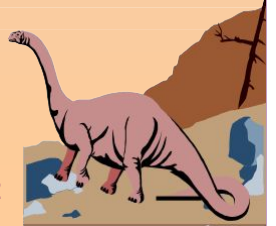
- Allows modules to tell the rest of the kernel that a new driver has become available.
- The kernel maintains dynamic tables of all known drivers, and provides a set of routines to allow drivers to be added to or removed from these tables at any time.
- Registration tables include the following items:
  - ◆ Device drivers
  - ◆ File systems
  - ◆ Network protocols
  - ◆ Binary format






# Conflict Resolution

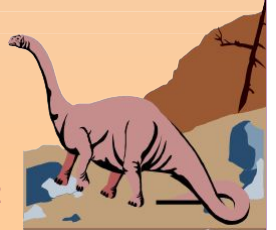
- A mechanism that allows different device drivers to reserve hardware resources and to protect those resources from accidental use by another driver
- The conflict resolution module aims to:
  - ◆ Prevent modules from clashing over access to hardware resources
  - ◆ Prevent *autoprobes* from interfering with existing device drivers
  - ◆ Resolve conflicts with multiple drivers trying to access the same hardware





# Process Management

- UNIX process management separates the creation of processes and the running of a new program into two distinct operations.
  - ◆ The **fork** system call creates a new process.
  - ◆ A new program is run after a call to **execve**. (for execute)
- Under UNIX, a process encompasses all the information that the operating system must maintain to track the context of a single execution of a single program.
- Under Linux, process properties fall into three groups: the process's identity, environment, and context.

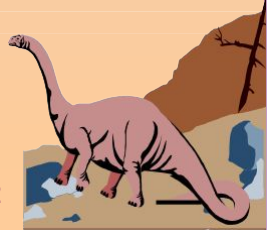







# Process Identity

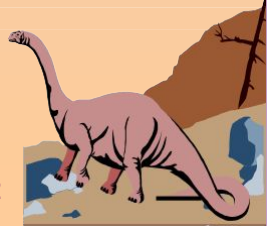
- **Process ID (PID).** The unique identifier for the process; used to specify processes to the operating system when an application makes a system call to signal, modify, or wait for another process.
- **Credentials.** Each process must have an associated user ID and one or more group IDs that determine the process's rights to access system resources and files.
- **Personality.** Not traditionally found on UNIX systems, but under Linux each process has an associated personality identifier that can slightly modify the semantics of certain system calls.  
Used primarily by emulation libraries to request that system calls be compatible with certain specific flavors of UNIX.





# Process Environment

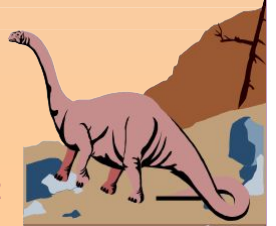
- The process's environment is inherited from its parent, and is composed of two null-terminated vectors:
  - ◆ The argument vector lists the command-line arguments used to invoke the running program; conventionally starts with the name of the program itself
  - ◆ The environment vector is a list of "NAME=VALUE" pairs that associates named environment variables with arbitrary textual values.
- Passing environment variables among processes and inheriting variables by a process's children are flexible means of passing information to components of the user-mode system software.
- The environment-variable mechanism provides a customization of the operating system that can be set on a per-process basis, rather than being configured for the system as a whole.





# Processes and Threads

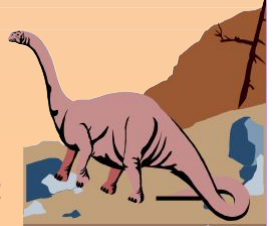
- Linux uses the same internal representation for processes and threads; a thread is simply a new process that happens to share the same address space as its parent.
- A distinction is only made when a new thread is created by the **clone** system call.
  - ◆ **fork** creates a new process with its own entirely new process context
  - ◆ **clone** creates a new process with its own identity, but that is allowed to share the data structures of its parent
- Using **clone** gives an application fine-grained control over exactly what is shared between two threads.





# Scheduling

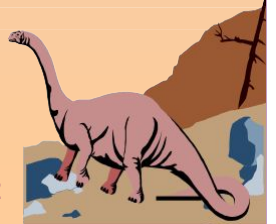
- The job of allocating CPU time to different tasks within an operating system.
- While scheduling is normally thought of as the running and interrupting of processes, in Linux, scheduling also includes the running of the various kernel tasks.
- Running kernel tasks encompasses both tasks that are requested by a running process and tasks that execute internally on behalf of a device driver.





# Process Scheduling

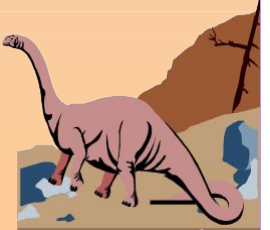
- Linux implements the FIFO and round-robin real-time scheduling classes; in both cases, each process has a priority in addition to its scheduling class.
  - ◆ The scheduler runs the process with the highest priority; for equal-priority processes, it runs the process waiting the longest
  - ◆ FIFO processes continue to run until they either exit or block
  - ◆ A round-robin process will be preempted after a while and moved to the end of the scheduling queue, so that round-robin processes of equal priority automatically time-share between themselves.





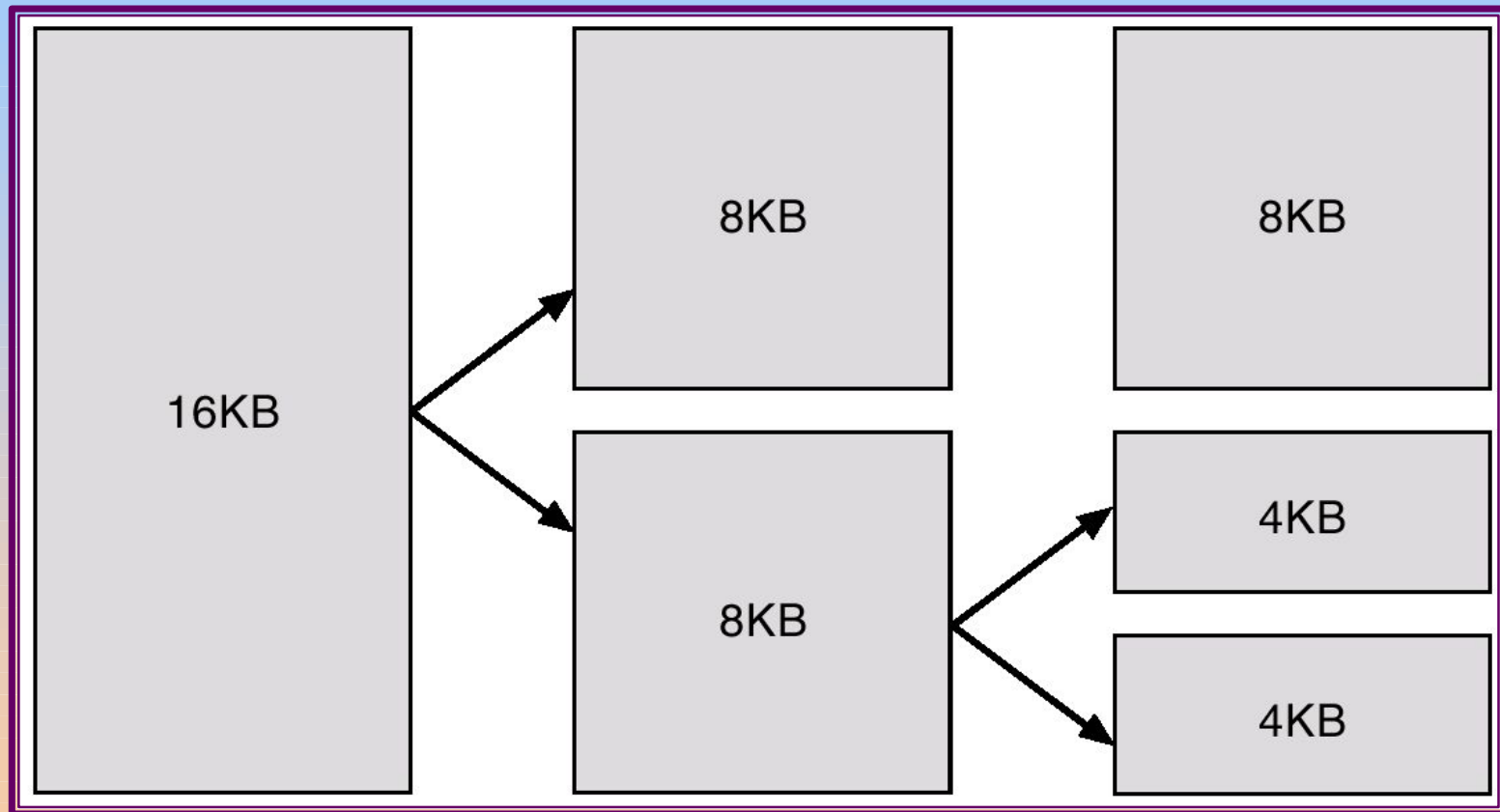
# Memory Management


- Linux's physical memory-management system deals with allocating and freeing pages, groups of pages, and small blocks of memory.
- It has additional mechanisms for handling virtual memory, memory mapped into the address space of running processes.



# Splitting of Memory in a Buddy

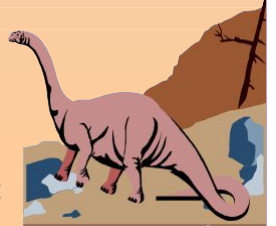
## Heap





# Managing Physical Memory

- The page allocator allocates and frees all physical pages; it can allocate ranges of physically-contiguous pages on request.
- The allocator uses a *buddy-heap* algorithm to keep track of available physical pages.
  - ◆ Each allocatable memory region is paired with an adjacent partner.
  - ◆ Whenever two allocated partner regions are both freed up they are combined to form a larger region.
  - ◆ If a small memory request cannot be satisfied by allocating an existing small free region, then a larger free region will be subdivided into two partners to satisfy the request.
- Memory allocations in the Linux kernel occur either statically (drivers reserve a contiguous area of memory during system boot time) or dynamically (via the page allocator).

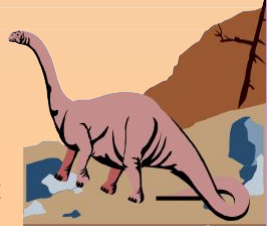






# File Systems

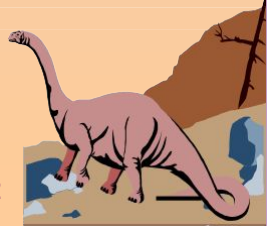
- To the user, Linux's file system appears as a hierarchical directory tree obeying UNIX semantics.
- Internally, the kernel hides implementation details and manages the multiple different file systems via an abstraction layer, that is, the *virtual file system (VFS)*.
- The Linux VFS is designed around object-oriented principles and is composed of two components:
  - ◆ A set of definitions that define what a file object is allowed to look like
    - ✓ The *inode-object* and the *file-object* structures represent individual files
    - ✓ the *file system object* represents an entire file system
  - ◆ A layer of software to manipulate those objects.



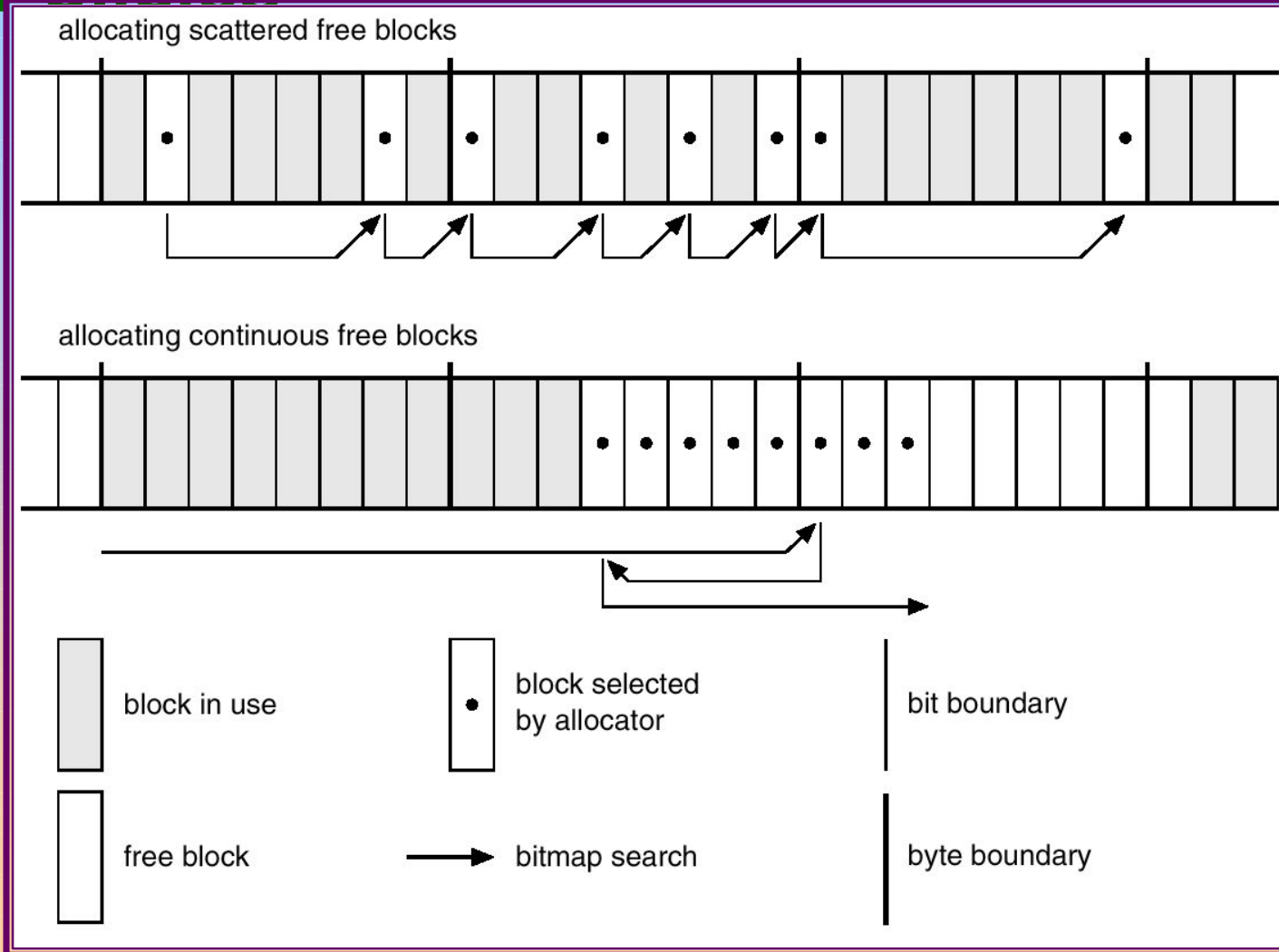



# The Linux Ext2fs File System

- Ext2fs(second extended file system ) uses a mechanism similar to that of BSD Berkeley Software Distribution Fast File System (ffs) for locating data blocks belonging to a specific file.
- The main differences between ext2fs and ffs concern their disk allocation policies.
  - ◆ In ffs, the disk is allocated to files in blocks of 8Kb, with blocks being subdivided into fragments of 1Kb to store small files or partially filled blocks at the end of a file.
  - ◆ Ext2fs does not use fragments; it performs its allocations in smaller units. The default block size on ext2fs is 1Kb, although 2Kb and 4Kb blocks are also supported.
  - ◆ Ext2fs uses allocation policies designed to place logically adjacent blocks of a file into physically adjacent blocks on disk, so that it can submit an I/O request for several disk blocks as a single operation.



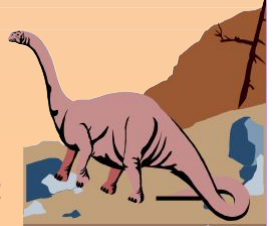
# Ext2fs Block-Allocation Policies





# Input and Output

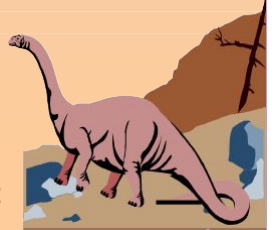
- The Linux device-oriented file system accesses disk storage through two caches:
  - ◆ Data is cached in the page cache, which is unified with the virtual memory system
  - ◆ Metadata is cached in the buffer cache, a separate cache indexed by the physical disk block.
- Linux splits all devices into three classes:
  - ◆ *block devices* allow random access to completely independent, fixed size blocks of data
  - ◆ *character devices* include most other devices; they don't need to support the functionality of regular files.
  - ◆ *network devices* are interfaced via the kernel's networking subsystem





# Security

- The *pluggable authentication modules (PAM)* system is available under Linux.
- PAM is based on a shared library that can be used by any system component that needs to authenticate users.
- Access control under UNIX systems, including Linux, is performed through the use of unique numeric identifiers (**uid** and **gid**).
- Access control is performed by assigning objects a *protections mask*, which specifies which access modes—read, write, or execute—are to be granted to processes with owner, group, or world access.





# Security (Cont.)

- Linux augments the standard UNIX **setuid** mechanism in two ways:
  - ◆ It implements the POSIX specification's saved *user-id* mechanism, which allows a process to repeatedly drop and reacquire its effective uid.
  - ◆ It has added a process characteristic that grants just a subset of the rights of the effective uid.
- Linux provides another mechanism that allows a client to selectively pass access to a single file to some server process without granting it any other privileges.

