Introduction to Operating Systems
Lecture 15: OS examples on I/O system

QIAN Weining
wnqian@sei.ecnu.edu.cn

Institute of Massive Computing
East China Normal University

Outline

- Linux file I/O system calls
- Linux I/O subsystem internals
- Windows XP

Linux file I/O

```
#include <sys/type.h>
#include <sys/stat.h>
#include <fcntl.h>

int open (const char *name, int flags);
int open (const char *name, int flags, mode_t
  mode);
int creat (const char *name, mode_t mode)
```

Reading a file

```
#include <unistd.h>
ssize_t read (int fd, void *buf, size_t len);

ssize_t ret;
while (len != 0 && (ret = read (fd, buf, len)) != 0) {
  if (ret == -1) {
    if (errno == EINTR) continue; /*syscall interrupt*/
    perror (“read”);
    break;
  }
  len -= ret;
  buf += ret;
}
```

Writing a file

```
#include <unistd.h>
ssize_t write (int fd, const void *buf, size_t count);

unsigned long word = 1720;
size_t count;
size_t nr;

count = sizeof (word);
nr = write (fd, &word, count);
if (nr == -1)
  /* error, check errno */
else if (nr != count)
  /* possible error, but 'errno' not set */
```

Synchronized I/O

```
#include <unistd.h>

int fsync (int fd);
int fdatasync (int fd);
int sync (void);

int close (int fd);
```

Seeking in a file

```
#include <sys/types.h>
#include <unistd.h>

off_t lseek (int fd, off_t pos, int origin);

off_t ret;
ret = lseek (fd, (off_t) 1825, SEEK_SET);
if (ret == (off_t) -1)
  /* error */
```

Positional reads/writes

```
#include <unistd.h>

ssize_t pread (int fd, void *buf, size_t
  count, off_t pos);
ssize_t pwrite (int fd, voind *buf, size_t
  count, off_t pos);
```

Multiplexed I/O

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int select (int n,
            fd_set *readfds,
            fd_set *writefds,
            fd_set *exceptfds,
            struct timeval *timeout);
```

Multiplexed I/O cont'd

```
#include <sys/poll.h>
struct pollfd {
  int fd;
  short events;
  short revents;
}

int poll (struct pollfd *fds, unsigned int
  nfds, int timeout);
```

Buffered I/O

```
#include <stdio.h>

FILE *fopen (const char *path, const char
  *mode);

FILE *fdopen (int fd, const char *mode);

int fclose (FILE *stream);

int fcloseall (void);

int fgetc (FILE *stream);

char * fgets (char *str, int size, FILE
  *stream);

size_t fread (void *buf, size_t size, size_t
  nr, FILE *stream);
```

Buffered I/O

```
int fputc (int c, FILE* stream);

int fputs (const char *str, FILE *stream);

size_t fwrite (void *buf, size_t size, size_t
  nr, FILE *stream);

int fseek (FILE *stream, long offset, int
  whence);

int fsetpos (FILE *stream, fpos_t *pos);

void rewind (FILE *stream);

long ftell (FILE *stream);

int fgetpos (FILE *stream, fpos_t *pos);
```

Buffered I/O

```
int fflush (FILE *stream);
int ferror (FILE *stream);
int feof (FILE *stream);
void clearerr (FILE *stream);
int fileno (FILE *stream);
```

Buffered I/O

```
#include <stdio.h>

int setvbuf (FILE *stream, char *buf, int
  mode, size_t size);
```

- Mode:
  - _IONBF: unbuffered
  - _IOLBF: line-buffered
  - _IOFBF: block-buffered

Thread safety

```
#include <stdio.h>

void flockfile (FILE *stream);
void funlockfile (FILE *stream);
int ftrylockfile (FILE *stream);
```

Scatter/gather I/O

```
#include <sys/uid.h>
struct iovec {
  void *iov_base;
  size_t iov_len;
}

ssize_t readv (int fd, const struct iovec
  *iov, int count);
ssize_t writev (int fd, const struct iovec
  *iov, int count);
```

Mapping files into memory

```
#include <sys/mman.h>

void * mmap (void *addr,
             size_t len,
             int prot,
             int flags,
             int fd,
             off_t offset);
int munmap (void *addr, size_t len);
```

Linux 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

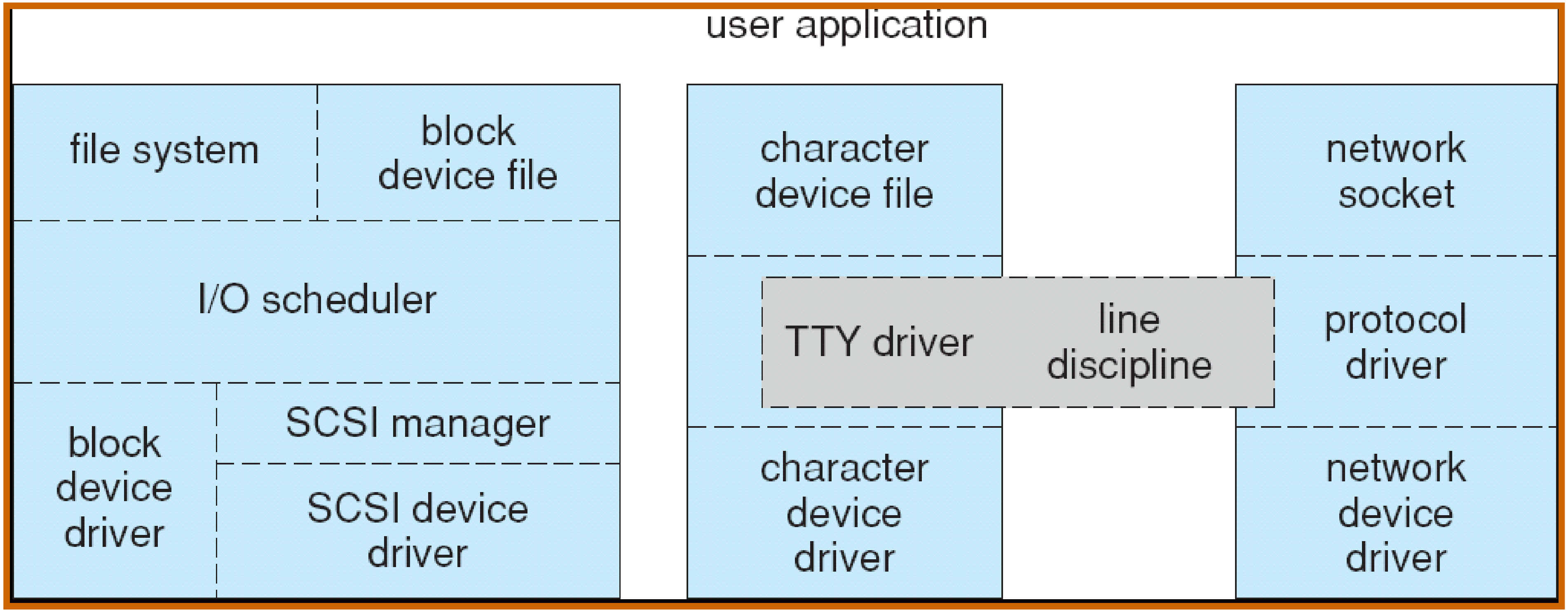
Linux conflict resolution

- 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

Linux I/O

- 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

Linux device-driver block



Linux network structure

- Networking is a key area of functionality for Linux.
  - It supports the standard Internet protocols for UNIX to UNIX communications
  - It also implements protocols native to nonUNIX operating systems, in particular, protocols used on PC networks, such as Appletalk and IPX
- Internally, networking in the Linux kernel is implemented by three layers of software:
  - The socket interface
  - Protocol drivers
  - Network device drivers

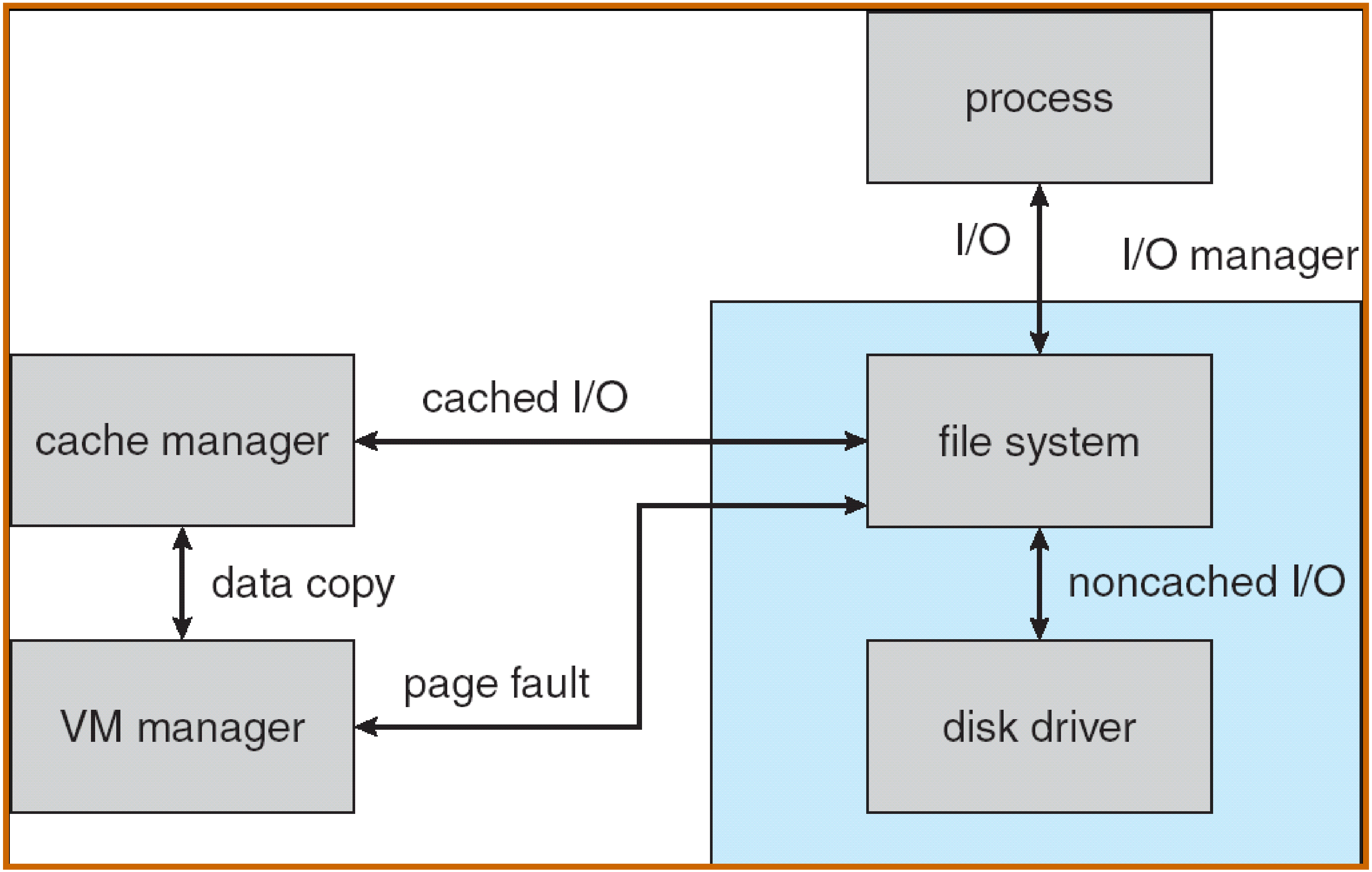
Network structure

- The most important set of protocols in the Linux networking system is the internet protocol suite
  - It implements routing between different hosts anywhere on the network
  - On top of the routing protocol are built the UDP, TCP and ICMP protocols

Windows XP executive: I/O manager

- The I/O manager is responsible for
  - file systems
  - cache management
  - device drivers
  - network drivers
- Keeps track of which installable file systems are loaded, and manages buffers for I/O requests
- Works with VM Manager to provide memory-mapped file I/O
- Controls the XP cache manager, which handles caching for the entire I/O system
- Supports both synchronous and asynchronous operations, provides time outs for drivers, and has mechanisms for one driver to call another

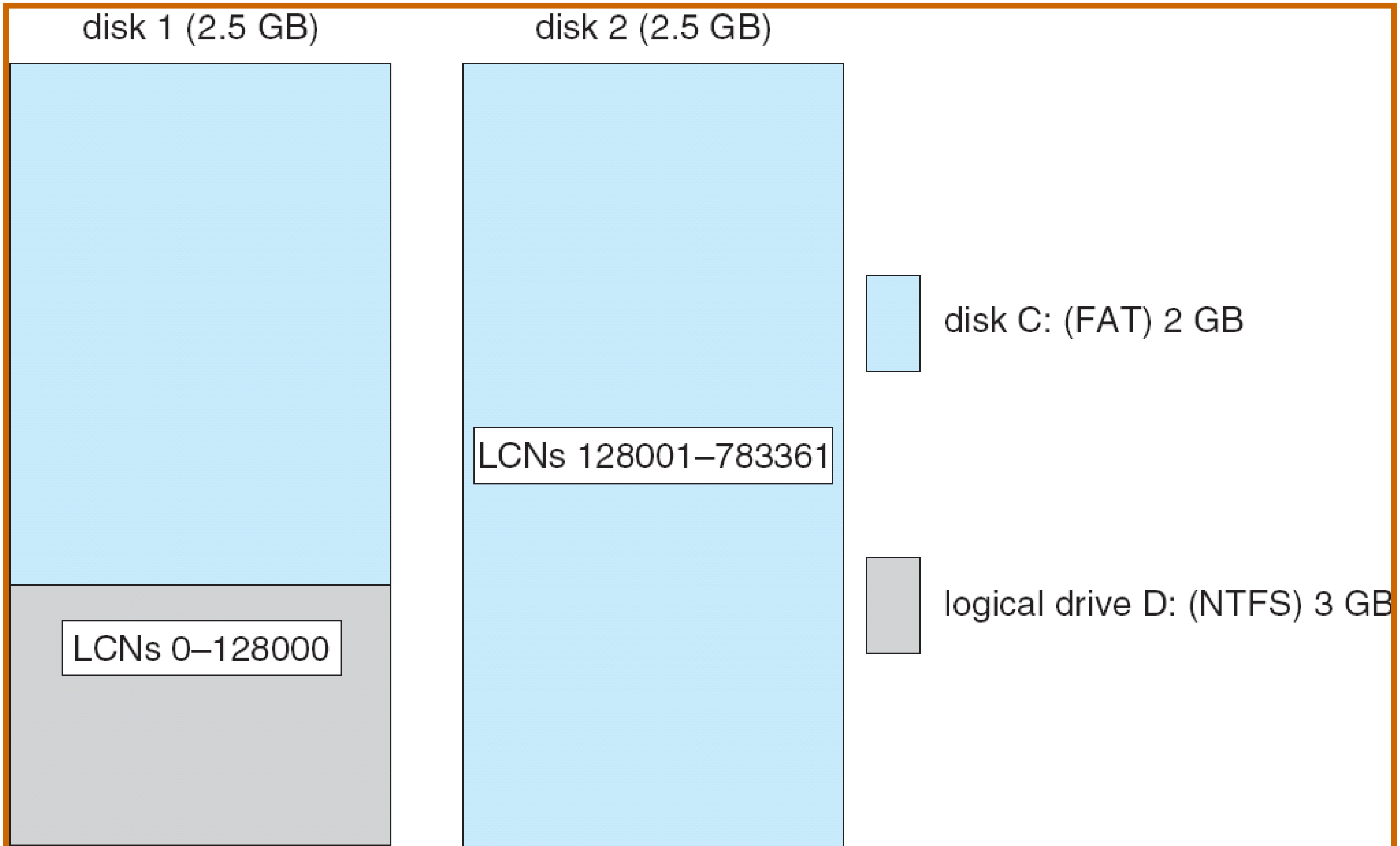
Windows XP file I/O



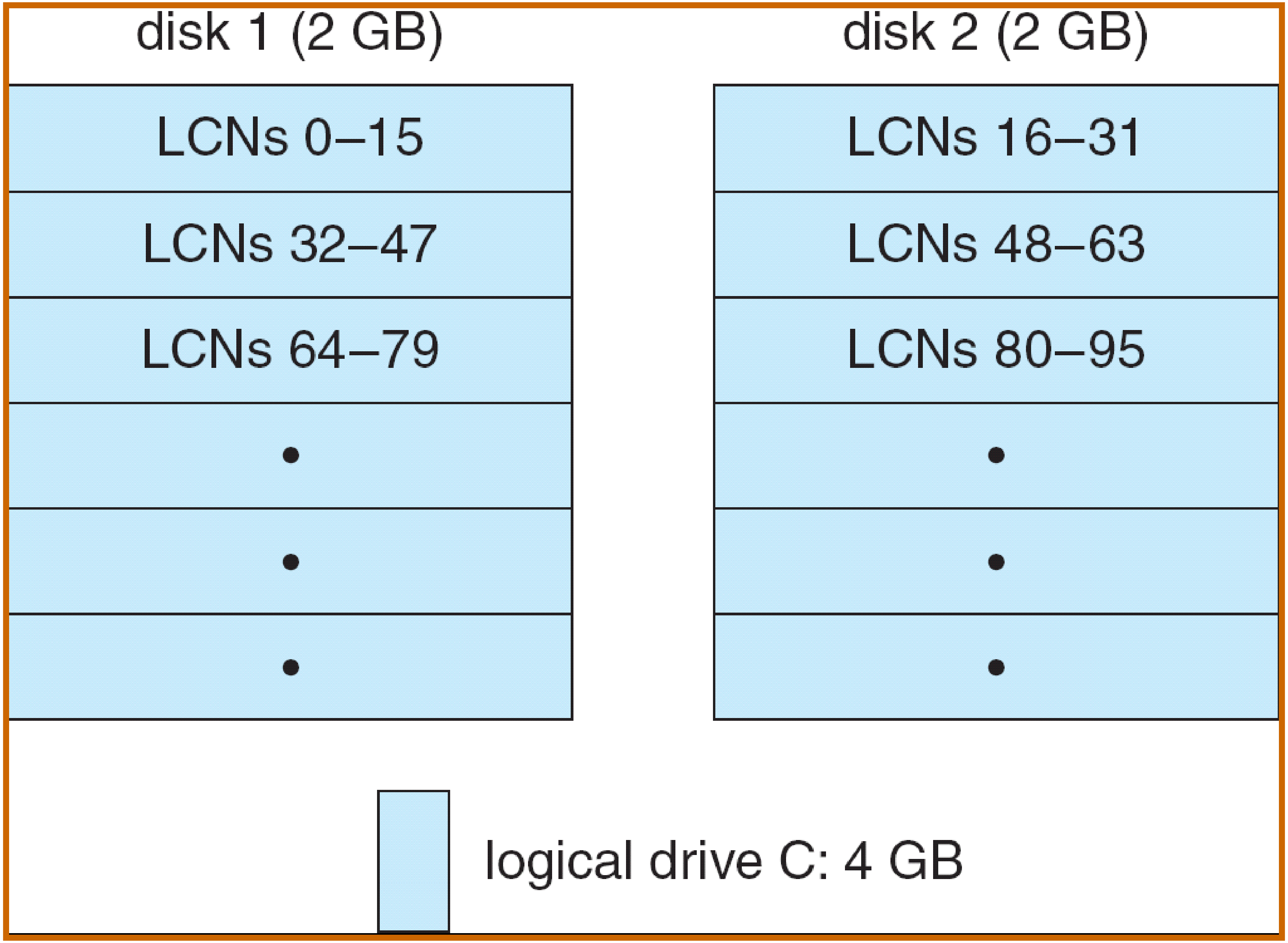
Windows XP volume manager and fault tolerance

- FtDisk, the fault tolerant disk driver for XP, provides several ways to combine multiple SCSI disk drives into one logical volume
- Logically concatenate multiple disks to form a large logical volume, a volume set
- Interleave multiple physical partitions in round-robin fashion to form a stripe set (also called RAID level 0, or "disk striping")
  - Variation: stripe set with parity, or RAID level 5
- Disk mirroring, or RAID level 1, is a robust scheme that uses a mirror set — two equally sized partitions on tow disks with identical data contents
- To deal with disk sectors that go bad, FtDisk, uses a hardware technique called sector sparing and NTFS uses a software technique called cluster remapping

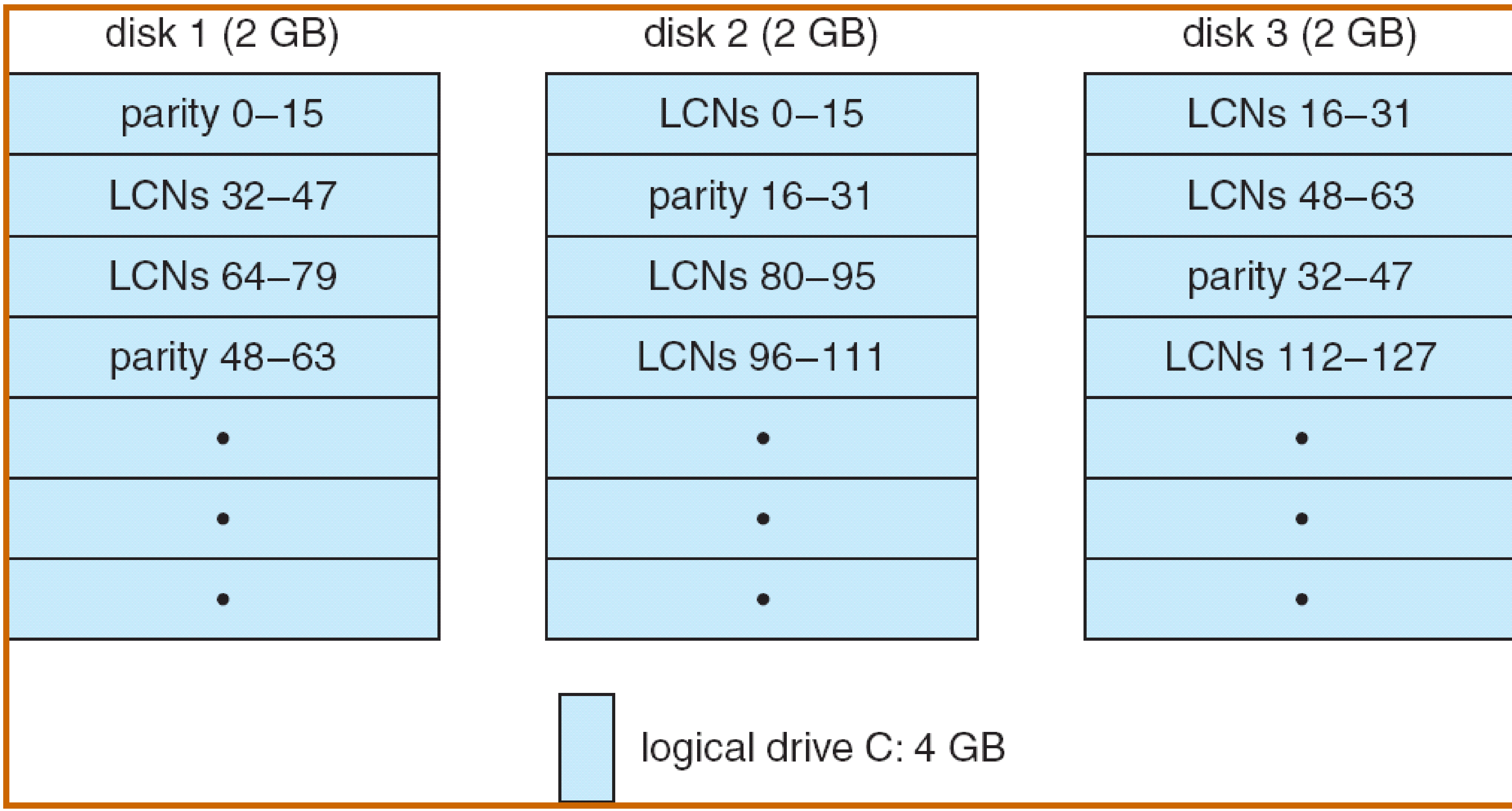
XP volume set on two drives



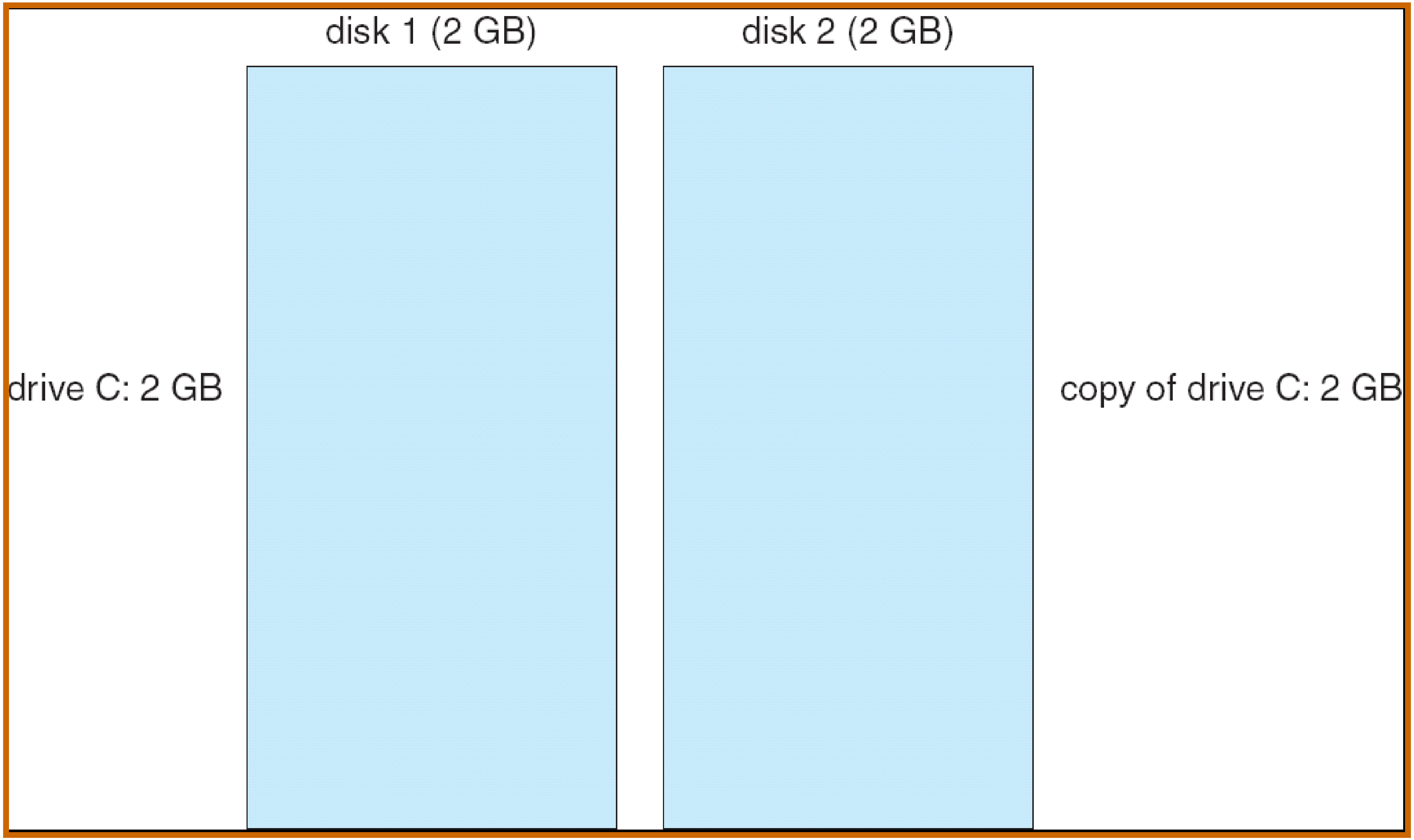
XP stripe set on two drives



XP stripe set with parity on 3 drives



XP mirror set on two drives



- Many slides are copied or adapted from:
  - Slides provided by authors of the textbook (http://codex.cs.yale.edu/avi/os-book/os7/)
  - Robert Love: Linux System Programming. O'Reilly 2007.