Disks and Clocks

EECE6029

Yizong Cheng

2/17/2016

Magnetic Disks

- A stack of aluminum, alloy, or glass platters typically 3.5 in in diameter.
- On each platter is deposited a thin magnetizable metal oxide.
- Some disks have microcontrollers which do bad block re-mapping, track caching
- Some are capable of doing more then one seek at a time, i.e. they can read on one disk while writing on another
- Real disk geometry is different from geometry used by driver => controller has to re-map request for (cylinder, head,sector) onto actual disk
- Disks are divided into zones, with fewer tracks on the inside, gradually progressing to more on the outside

Parameter	IBM 360-KB floppy disk	WD 3000 HLFS hard disk
Number of cylinders	40	36,481
Tracks per cylinder	2	255
Sectors per track	9	63 (avg)
Sectors per disk	720	586,072,368
Bytes per sector	512	512
Disk capacity	360 KB	300 GB
Seek time (adjacent cylinders)	6 msec	0.7 msec
Seek time (average case)	77 msec	4.2 msec
Rotation time	200 msec	6 msec
Time to transfer 1 sector	22 msec	1.4 μsec

Figure 5-18. Disk parameters for the original IBM PC 360-KB floppy disk and a Western Digital WD 3000 HLFS ("Velociraptor") hard disk.

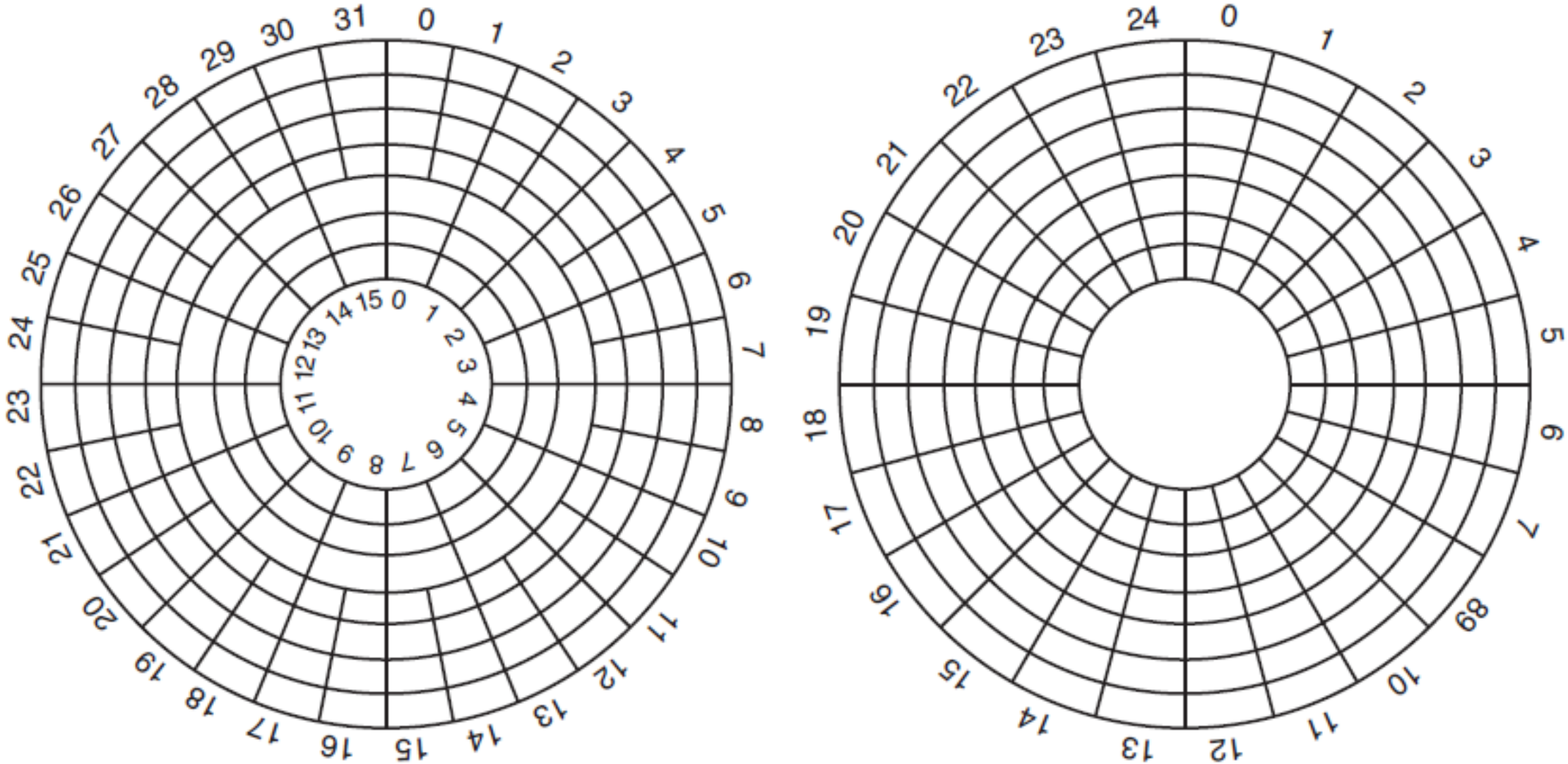

Figure 5-19. (a) Physical geometry of a disk with two zones. (b) A possible virtual geometry for this disk.

RAID

- Parallel I/O to improve performance and reliability
- vs SLED, Single Large Expensive Disk
- Bunch of disks which appear like a single disk to the OS
- SCSI disks often used-cheap, 7 disks per controller
- SCSI is set of standards to connect CPU to peripherals
- Different architectures-level 0 through level 7

RAID Levels

- Raid level 0 uses strips of k sectors per strip.
  - Consecutive strips are on different disks
  - Write/read on consecutive strips in parallel
  - Good for big enough requests
- Raid level 1 duplicates the disks
  - Writes are done twice, reads can use either disk
  - Improves reliability
- Level 2 works with individual words, spreading word + ecc over disks.
  - Need to synchronize arms to get parallelism
- Raid level 3 works like level 2, except all parity bits go on a single drive
- Raid 4,5 work with strips. Parity bits for strips go on separate drive (level 4) or several drives (level 5)

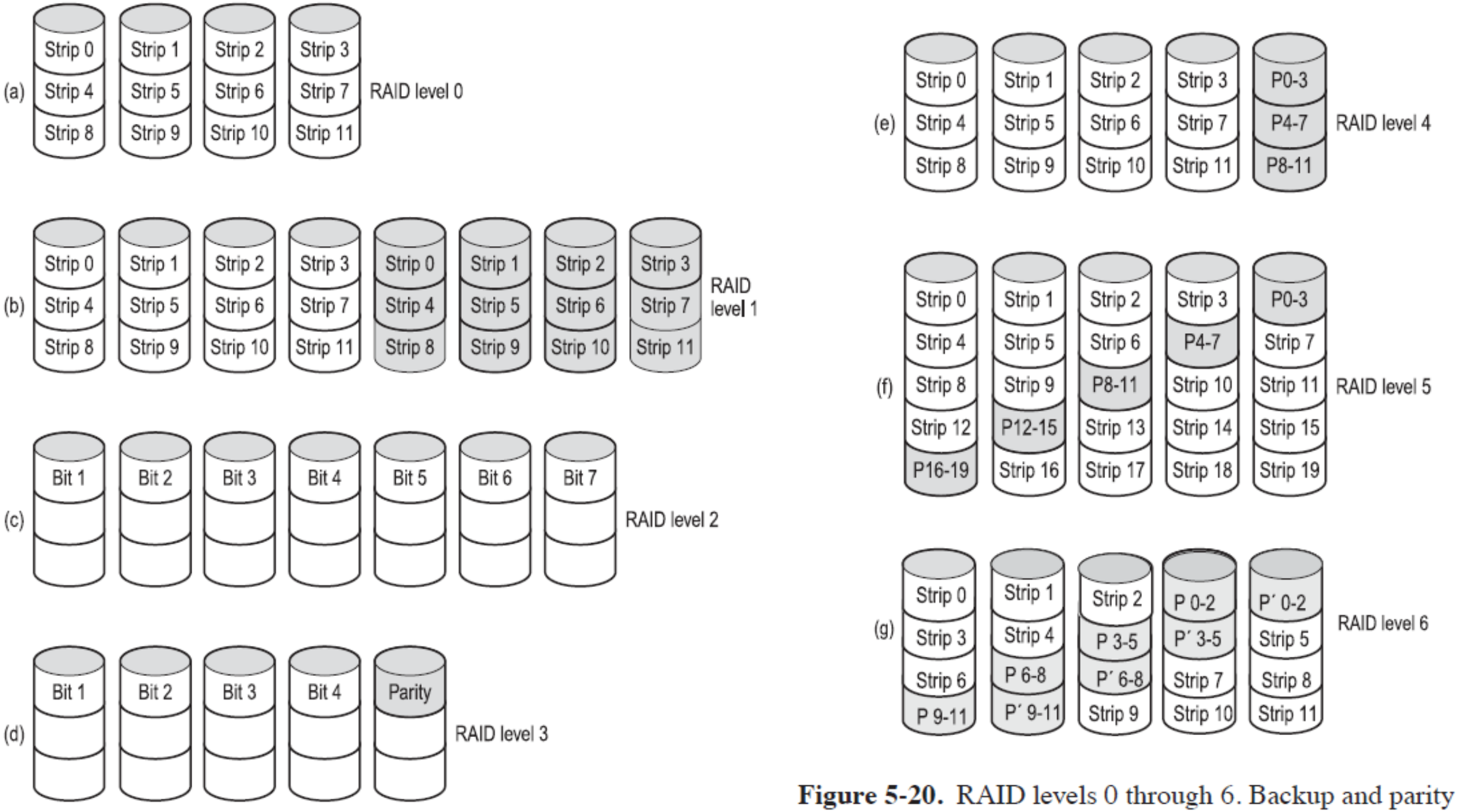

Figure 5-20. RAID levels 0 through 6. Backup and parity drives are s

Low-Level Formatting

- Done by software.
- Concentric tracks, each containing some number of sectors with gaps in between.
  - Most disks use 512-byte sectors
- Preamble: a certain bit pattern for hardware to recognize.
- Preamble: cylinder and sector numbers
- ECC: 16 bytes often for error detection
- cylinder skew and head skew
- interleaving while the controller checks ECC
- 20% capacity may be used for preambles, ECCs and gaps

Master Boot Record

- MBR in sector 0 contains boot code and partition table.
- Partition entries in most systems are limited to 32 bits.
  - So the address of a sector is at most 32 bits and sectors have 512 bytes and the maximum disk size is $2^{32+9} = 2^{41}$ bytes = 2TB
  - GUID Partition Table supports 9.4ZB
- x86 has room for four partitions (C:, D:, E:, and F: if all for Windows)
- One partition must be marked as active in the partition table to be able to boot from the hard disk.
- When the power is turned on, the BIOS runs and reads in MBR and then jumps to the active partition to load the OS.

Disk Sector

Figure 5-21. A disk sector.

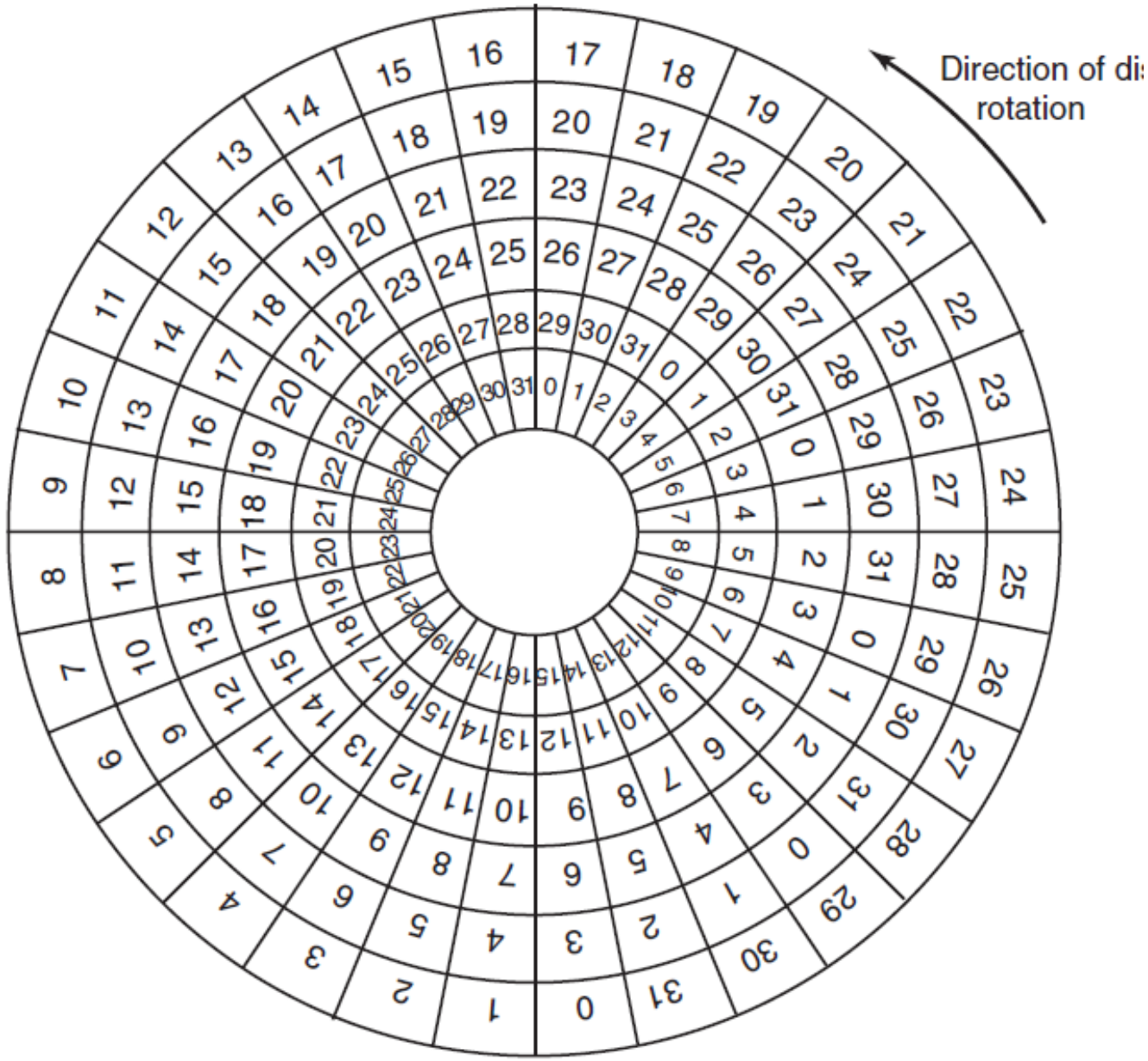

Figure 5-22. An illustration of cylinder skew.

Sector Interleaving

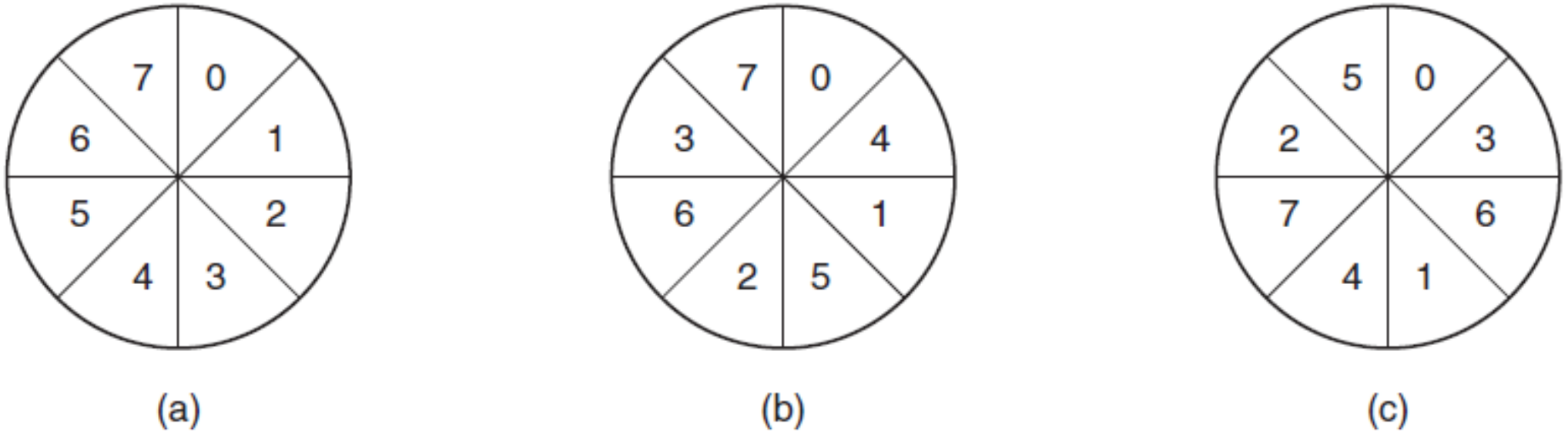

Figure 5-23. (a) No interleaving. (b) Single interleaving. (c) Double interleaving.

Disk Arm Scheduling

- Read/write time factors
  - Seek time (the time to move the arm to the proper cylinder).
  - Rotational delay (the time for the proper sector to rotate under the head).
  - Actual data transfer time.
- Driver keeps list of requests (cylinder number, time of request)
- Try to optimize the seek time
- FCFS is easy to implement, but optimizes nothing

Shortest Seek First Disk Scheduling Algorithm

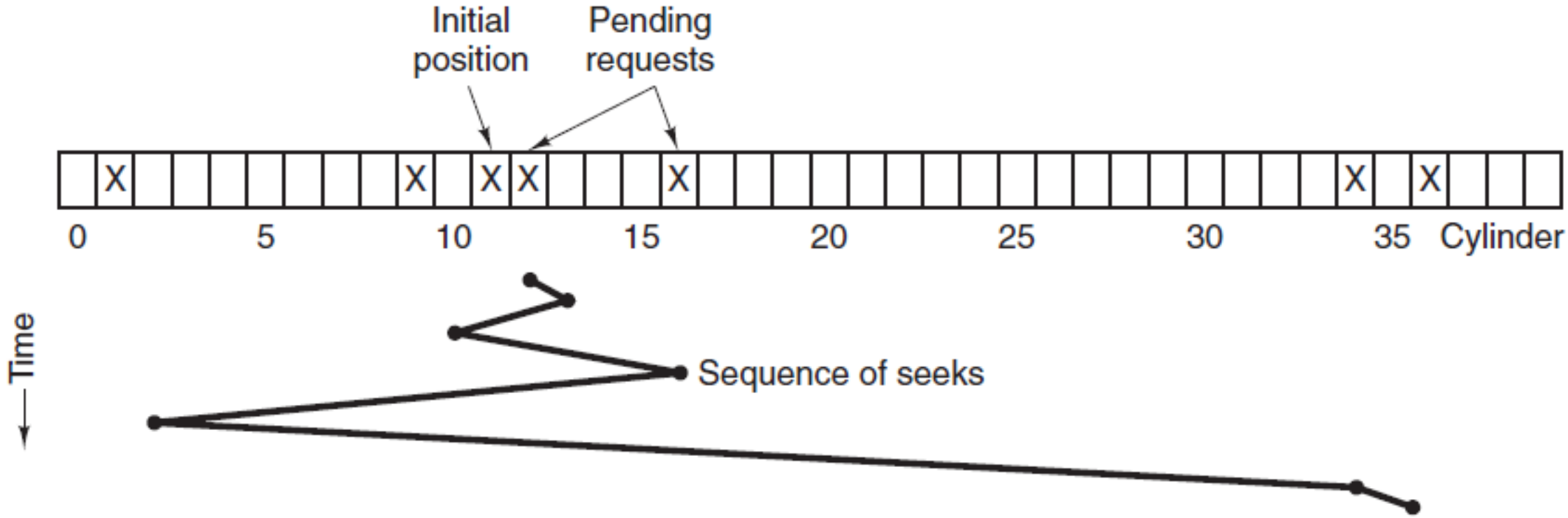

Figure 5-24. Shortest Seek First (SSF) disk scheduling algorithm.

The Elevator Algorithm

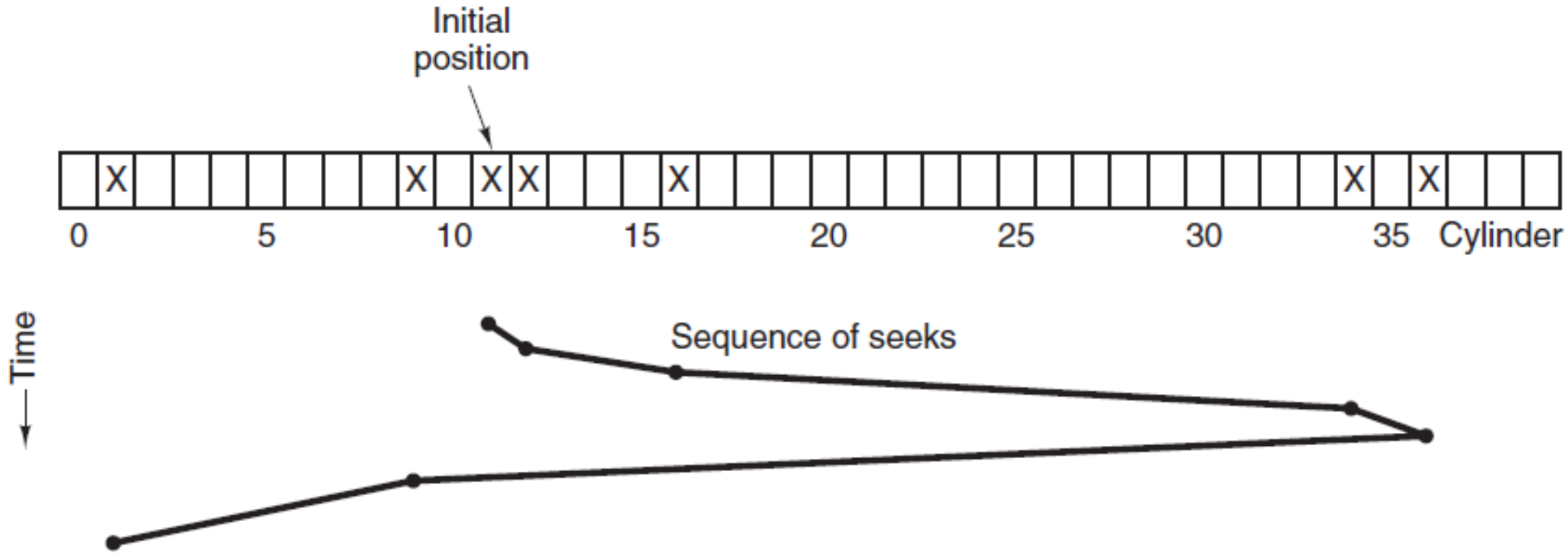

Figure 5-25. The elevator algorithm for scheduling disk requests.

Disk Controller’s Cache

- Disk controllers have their own cache
- Cache is separate from the OS cache
- OS caches blocks independently of where they are located on the disk
- Controller caches blocks which were easy to read but which were not necessarily requested

Bad Sectors

- Manufacturing defect-that which was written does not correspond to that which is read (back)
- Controller or OS deals with bad sectors
- If controller deals with them the factory provides a list of bad blocks and controller remaps good spares in place of bad blocks
- Substitution can be done when the disk is in use-controller “notices” that block is bad and substitutes

Bad Sectors

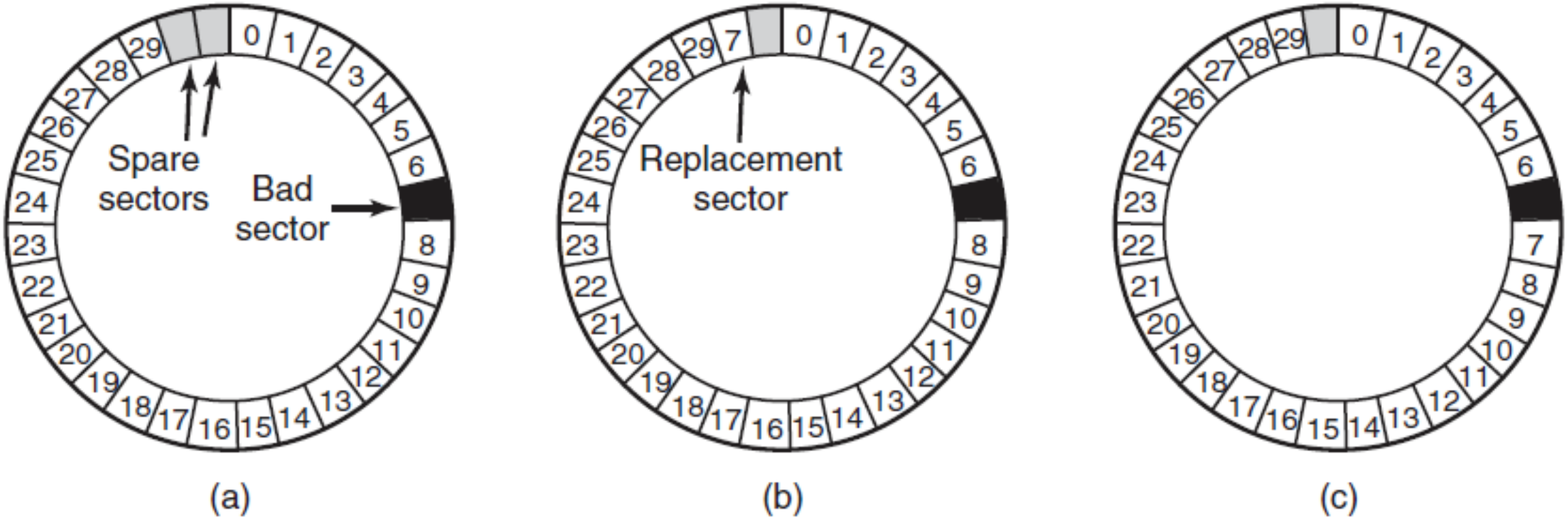

Figure 5-26. (a) A disk track with a bad sector. (b) Substituting a spare for the bad sector. (c) Shifting all the sectors to bypass the bad one.

Stable Storage

- Use 2 identical disks-do the same thing to both disks and 3 operations
- Stable writes. first write, then read back and compare. If they are the same write to second disk. If write fails, try up to n times to get it to succeed. After n failures keep using spare sectors until it succeeds. Then go to disk 2.
- Stable reads. read from disk 1 n times until get a good ECC, otherwise read from disk 2 (assumption that probability of both sectors being bad is negligible)
- Crash recovery. read both copies of blocks and compare them. If one block has an ECC error, overwrite it with the good block. If both pass the ECC test, then pick either

Stable Writes

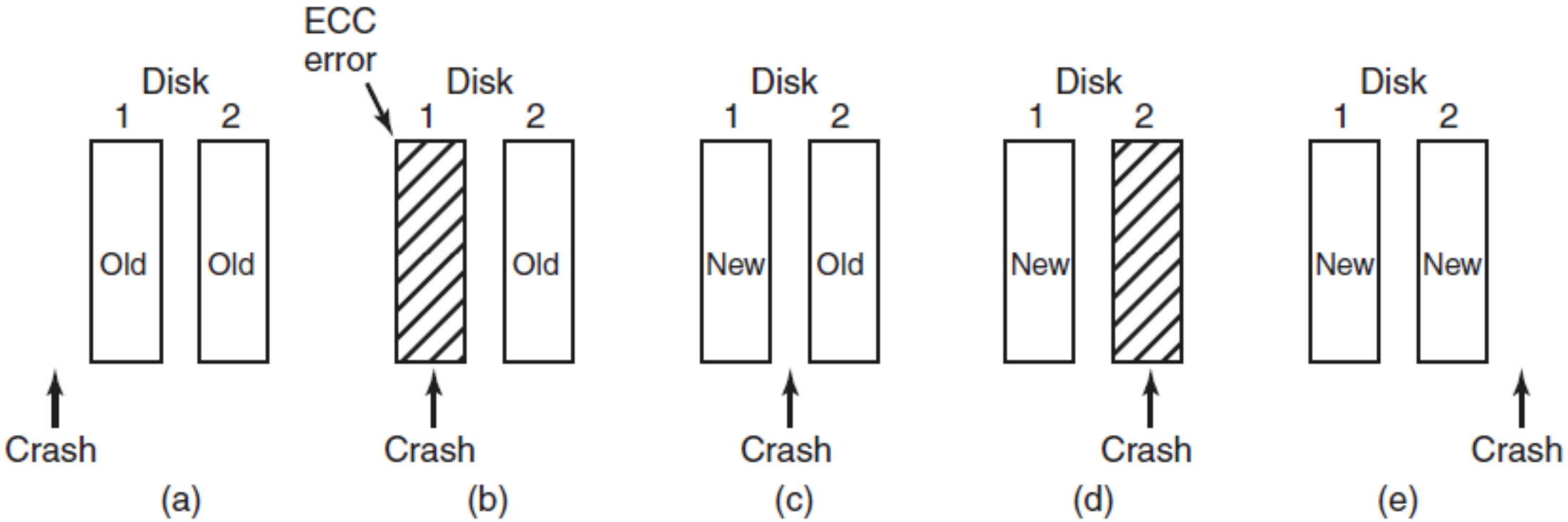

Figure 5-27. Analysis of the influence of crashes on stable writes.

Programmable Clock

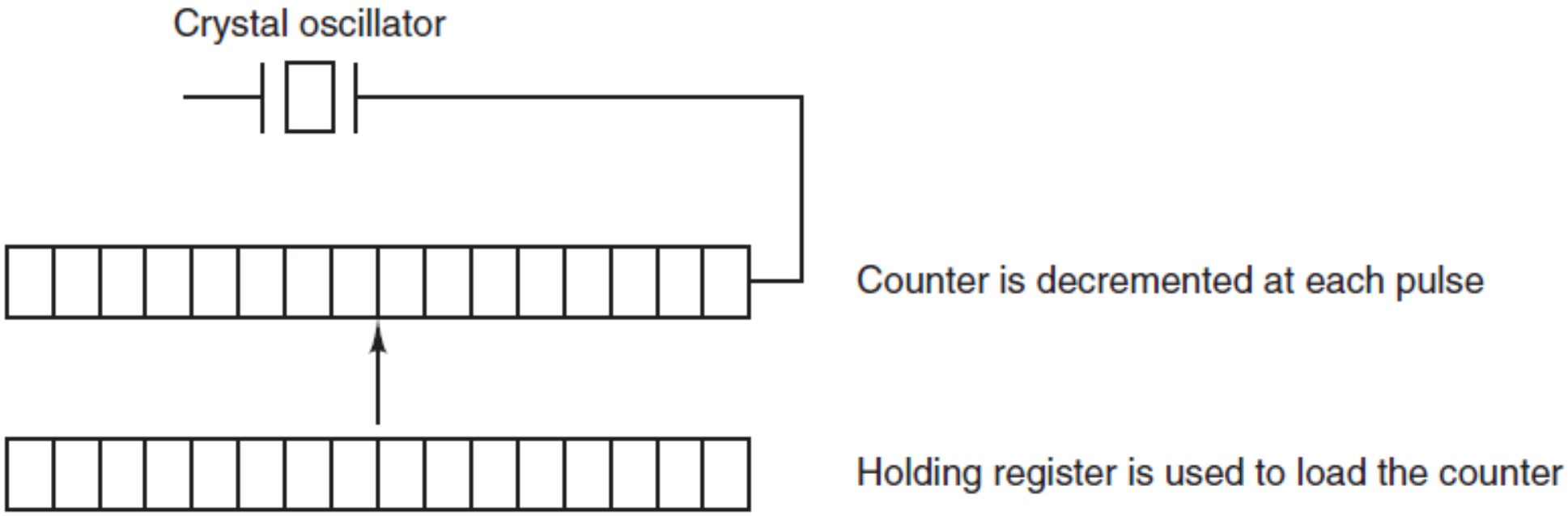

Figure 5-28. A programmable clock.

Duties of a Clock Driver

- Maintaining the time of day.
- Preventing processes from running longer than they are allowed to.
- Accounting for CPU usage.
- Handling alarm system call made by user processes.
- Providing watchdog timers for parts of the system itself.
- Doing profiling, monitoring, statistics gathering.

Time of Day Maintenance

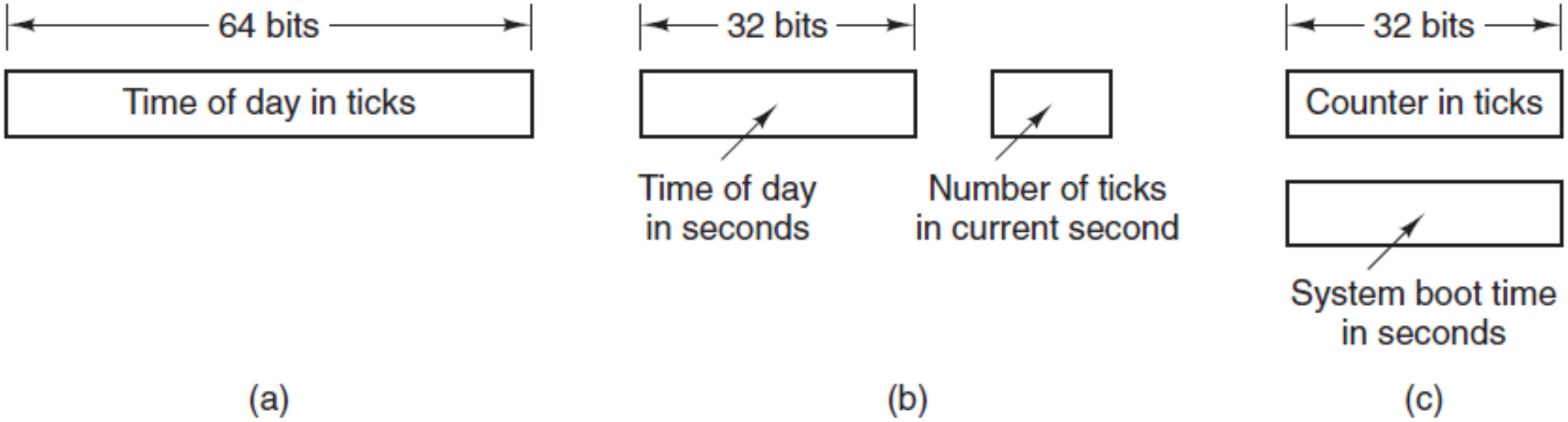

Figure 5-29. Three ways to maintain the time of day.

Multiple Timers, Single Clock

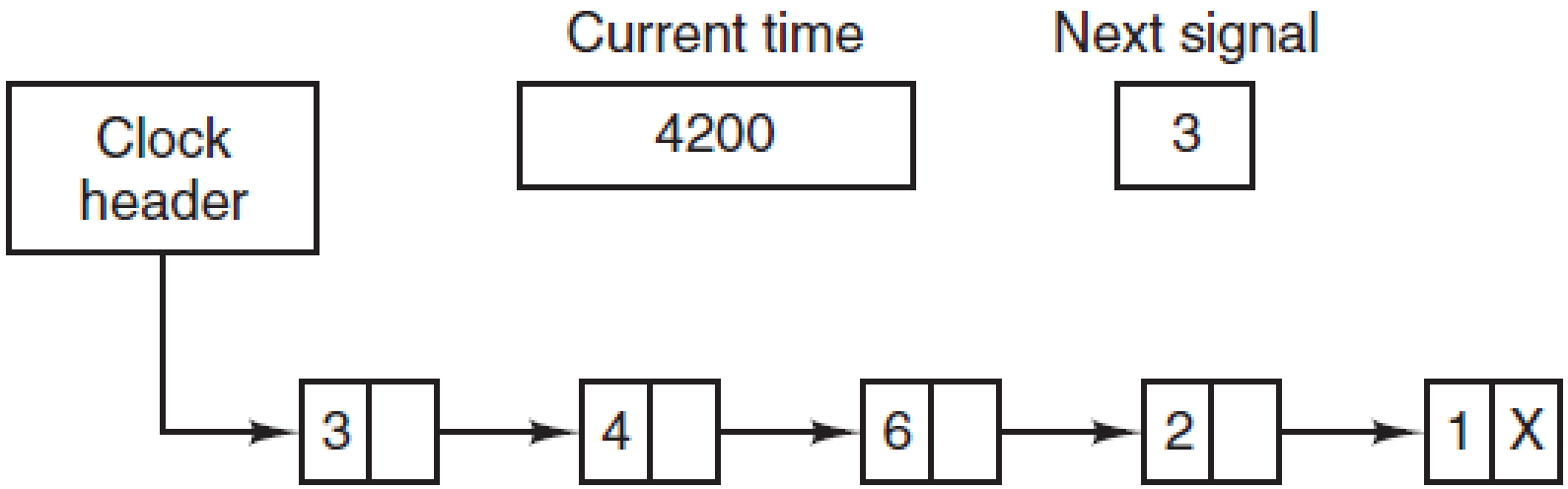

Figure 5-30. Simulating multiple timers with a single clock.

Example of Multiple Timers

- TCP manages four different timers for each connection.
  - retransmission timer for waiting an acknowledgment
  - persist timer for sending window size information
  - keepalive timer for idle connection
  - 2MSL timer for TIME_WAIT state

time.h Functions

```
#include <time.h>

time_t time(time_t *calptr);
```

Returns: value of time if OK, –1 on error

```
#include <time.h>

char *asctime(const struct tm *tmptr);

char *ctime(const time_t *calptr);
```

Both return: pointer to null-terminated string

```
#include <time.h>

struct tm *gmtime(const time_t *calptr);

struct tm *localtime(const time_t *calptr);
```

Both return: pointer to broken-down time

Broken-Down Time in struct tm

```
struct tm {          /* a broken-down time */
  int  tm_sec;       /* seconds after the minute: [0 - 60] */
  int  tm_min;       /* minutes after the hour: [0 - 59] */
  int  tm_hour;      /* hours after midnight: [0 - 23] */
  int  tm_mday;      /* day of the month: [1 - 31] */
  int  tm_mon;       /* months since January: [0 - 11] */
  int  tm_year;      /* years since 1900 */
  int  tm_wday;      /* days since Sunday: [0 - 6] */
  int  tm_yday;      /* days since January 1: [0 - 365] */
  int  tm_isdst;     /* daylight saving time flag: <0, 0, >0 */
};
```

time1.c

```
include <stdio.h>
#include <time.h>

int main(int argc, char *argv[])
{
    time_t now;
    struct tm *tptr;
    time(&now);
    printf("%s\n", ctime(&now));
    tptr = localtime(&now);
    printf("%d\n", tptr->tm_wday);
}
```

alarm, SIGALRM, pause, signal

```
#include <signal.h>

void (*signal(int signo, void (*func)(int)))(int);
```

Returns: previous disposition of signal (see following) if OK, `SIG_ERR` on error

```
#include <signal.h>

int kill(pid_t pid, int signo);

int raise(int signo);
```

Both return: 0 if OK, –1 on error

```
#include <unistd.h>

unsigned int alarm(unsigned int seconds);
```

Returns: 0 or number of seconds until previously set alarm

```
#include <unistd.h>

int pause(void);
```

Returns: –1 with `errno` set to `EINTR`

time2.c

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>

static void sig_alrm(int signo){
    printf("receive SIGALRM\n");
}

int main(int argc, char *argv[])
{
    if (signal(SIGALRM, sig_alrm) == SIG_ERR) exit(1);
    printf("before alarm\n");
    alarm(atoi(argv[1]));
    pause();
    printf("after pause\n");
    return 0;
}
```

times and struct tms

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

clock_t times(struct tms *buf);
```

Returns: elapsed wall clock time in clock ticks if OK, –1 on error

```
struct tms {
  clock_t  tms_utime;  /* user CPU time */
  clock_t  tms_stime;  /* system CPU time */
  clock_t  tms_cutime; /* user CPU time, terminated children */
  clock_t  tms_cstime; /* system CPU time, terminated children */
};
```

sigaction() and struct sigaction

```
#include <signal.h>

int sigaction(int signo, const struct sigaction *restrict act,
              struct sigaction *restrict oact);
```

Returns: 0 if OK, –1 on error

```
 struct sigaction {
   void      (*sa_handler)(int);    /* addr of signal handler, */
                                    /* or SIG_IGN, or SIG_DFL */
   sigset_t sa_mask;                /* additional signals to block */
   int      sa_flags;               /* signal options, Figure 10.16 */

   /* alternate handler */
   void     (*sa_sigaction)(int, siginfo_t *, void *);
};
```

time3.c

```
static void
handler(int sig, siginfo_t *si, void *uc){
   printf("\nYou are too slow.\n");
}

int main(int argc, char *argv[])
{
   timer_t timerid;
   struct sigevent sev;
   struct itimerspec its;
   struct sigaction sa;
   char line[100];
   int n;

   sa.sa_flags = SA_SIGINFO;
   sa.sa_sigaction = handler;
   sigemptyset(&sa.sa_mask);
   if (sigaction(SIGRTMIN, &sa, NULL) == -1){
     perror("sigaction"); exit(1);
   }
```

timer_create, timer_t, struct sigevent

```
sev.sigev_notify = SIGEV_SIGNAL;
sev.sigev_signo = SIGRTMIN;
sev.sigev_value.sival_ptr = &timerid;
if (timer_create(CLOCK_REALTIME, &sev, &timerid) == -1){
  perror("timer_create"); exit(1);
}
printf("timer ID is 0x%lx\n", (long) timerid);
```

timer_settime and struct itimerspec

```
its.it_value.tv_sec = atoi(argv[1]);
its.it_value.tv_nsec = 0;
its.it_interval.tv_sec = atoi(argv[1]);
its.it_interval.tv_nsec = 0;
if (timer_settime(timerid, 0, &its, NULL) == -1){
  perror("timer_settime"); exit(1);
}
```

Timer for Blocking Time

```
    printf("enter a line in %s seconds\n", argv[1]);

    n = read(0, line, 100);
    if (n > 0) write(1, line, n);

    exit(0);
}
```