Hard Disk Geometry and Low-Level Data Structures

- leading-edge hard disks now pack a whopping 20 GB of storage per platter in the same amount of space. Pretty amazing.

- Of course, this trend is only going to continue, with new drives having more and more data in the same space. In order to use all this real estate to best advantage, special methods have evolved for dividing the disk up into usable pieces.
- The goals, as usual, are two-fold: **increasing capacity** and **increasing performance.**

- This section takes a detailed look at how information is
  - encoded,
  - stored, retrieved
  - and managed on a modern hard disk.

- Many of the descriptions in this section in fact form the basis for how data is stored on other media as well.

Hard Disk Data Encoding and Decoding

- Digital information is a **stream of ones and zeros**.
- **Hard disks** store **information** in the form of **magnetic pulses**.
- In order for the PC's data to be stored on the hard disk, therefore, it must be converted to magnetic information.
- When it is read from the disk, it must be converted back to digital information.
- This work is done by the integrated controller built into the hard drive, in combination with sense and amplification circuits that are used to interpret the weak signals read from the platters themselves.

Hard Disk Data Encoding and Decoding

- Magnetic information on the disk consists of a stream of (**very, very small**) magnetic fields.
- As you know, a magnet has two poles, north and south, and magnetic energy (called flux) flows from the north pole to the south pole.
- Information is stored on the hard disk by encoding information into a series of magnetic fields.
- This is done by placing the magnetic fields in one of two polarities:
  - either so
  - the north pole arrives before the south pole as the disk spins (**N-S**),
  - or
  - so the south pole arrives before the north (**S-N**).

Hard Disk Data Encoding and Decoding

- Although it is conceptually simple
  - to match "0 and 1" digital information
  - to **"N-S** and **S-N"** magnetic fields

- the reality is much more complex:
  - a 1-to-1 correspondence is not possible
  - and special techniques must be employed
  - to ensure that the data is written and read correctly.

- This section discusses the technical issues involved in encoding and decoding hard disk data.

Technical Requirements for Encoding and Decoding

- You might think that since there are two magnetic polarities,
  - N-S and S-N,
  - they could be used nicely to represent a
  - "one" and a "zero" respectively,
  - to allow easy encoding of digital information.

- There are **3 key reasons** why
- it is not possible to do this simple 1-to-1 encoding:
- **1. Fields vs. Reversals (suprotnost, promena)**

- **2. Synchronization**

- **3. Field Separation**

Technical Requirements for Encoding and Decoding

- **1. Fields vs. Reversals:** Read/write heads are designed not to measure the actual polarity of the magnetic fields, **but rather *flux reversals***, which occur when the head moves from an area that has north-south polarity to one that has south-north polarity, or vice-versa.
- The reason the heads are designed based on flux reversals instead of absolute magnetic field, is that reversals are easier to measure. When the hard disk head passes from over a reversal a small voltage spike is produced that can be picked up by the detection circuitry.
- As disk density increases, the strength of each individual magnetic field continues to decrease, which makes detection sensitivity critical. What this all means is that the encoding of data must be done based on flux reversals, and not the contents of the individual fields.

Technical Requirements for Encoding and Decoding

- **2. Synchronization:** Another consideration in the encoding of data is the necessity of using some sort of **method** of **indicating** where **one bit ends and another begins.**

- Even if we could use one polarity to represent a "one" and another to represent a "zero", what would happen if we needed to encode on the disk a stream of 1,000 consecutive zeros? It would be very difficult to tell where, say, bit 787 ended and bit 788 began.

- Imagine driving down a highway with no odometer or highway markings and being asked to stop exactly at mile #787 on the highway. It would be pretty hard to do, even if you knew where you started from and your exact speed.

Technical Requirements for Encoding and Decoding

- **3. Field Separation:**
- Although we can conceptually think of putting 1000 tiny N-S pole magnets in a row one after the other, in reality magnetic fields don't work this way.

- **They are additive.**
- Aligning 1000 small magnetic fields near each other would create **one *large* magnetic field**, 1000 times the size and strength of the individual components.

Technical Requirements for Encoding and Decoding

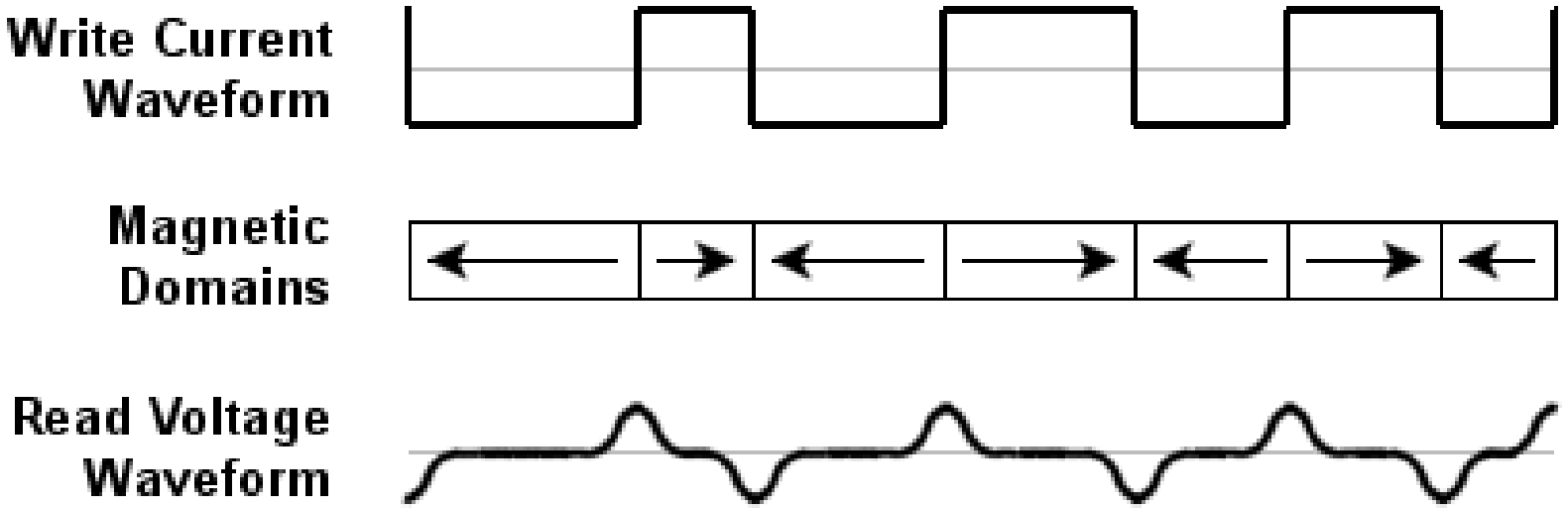
- Therefore, in order to encode data on the hard disk so that we'll be able to read it back reliably, we need to take the issues above into account.

- We must encode using flux reversals, not absolute fields. We must keep the number of consecutive fields of same polarity to a minimum.

- And to keep track of which bit is where, some sort of clock synchronization must be added to the encoding sequence.

Technical Requirements for Encoding and Decoding

- Idealized depiction of the way hard disk data is written and then read. The top waveform shows how patterns are written to the disk. In the middle, a representation is shown of the way the media on the disk is magnetized into domains of opposite direction based on the polarity of the write current. The waveform on the bottom shows how the flux transitions on the disk translate into positive and negative voltage pulses when the disk is read. Note that the pattern above is made up and doesn't follow any particular pattern or encoding method.



Frequency Modulation (FM)

- The first common encoding system for recording digital data on magnetic media was *frequency modulation*, of course abbreviated *FM*.
- This is a simple scheme, where:
  - a **1** is recorded as **two** consecutive flux reversals 1->**RR**
  - a **0** is recorded as a flux reversal followed by no flux reversal 0->**RN**

- This can also be thought of as follows: a flux reversal is made at the start of each bit to represent the clock, and then an additional reversal is added in the middle of each bit for a one, while the additional reversal is omitted for a zero.

- This table shows the encoding pattern for FM (where I have designated "**R**" to represent a flux reversal and "**N**" to represent no flux reversal).
- The average number of flux reversals per bit on a random bit stream pattern is 1.5. The best case (all zeroes) would be 1, the worst case (all ones) would be 2:

Frequency Modulation (FM)

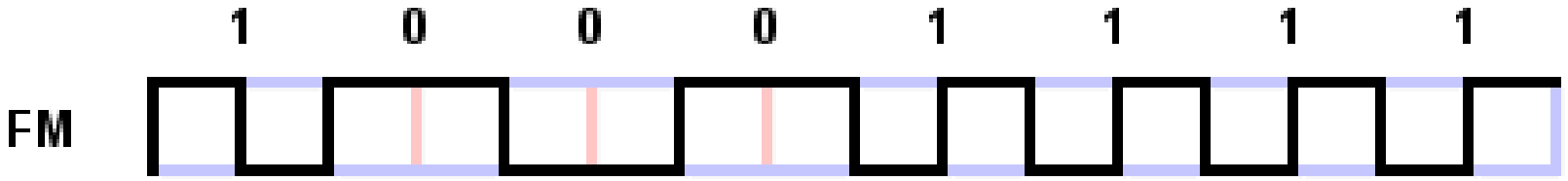
1x0,5 + 2*0.5=1.5

Bit Pattern	Encoding Pattern	Flux Reversals Per Bit	Bit Pattern Commonality In Random Bit Stream
0	RN	1	50%
1	RR	2	50%
Weighted Average		1.5	100%

- The name "frequency modulation" comes from the fact that the number of reversals is doubled for ones compared to that for zeros. This can be seen in the patterns that are created if you look at the encoding pattern of a stream of ones or zeros.
- A byte of all 0 would be encoded as "**RNRNRNRNRNRNRNRN**", while
- a byte of all 1 would be "**RRRRRRRRRRRRRRRR**".
- As you can see, the ones have double the frequency of reversals compared to the zeros; hence frequency modulation (meaning, changing frequency based on data value).

FM

- FM encoding write waveform for the byte "10001111".
- Each bit cell is depicted as a blue rectangle with a pink line representing the position where a reversal is placed, if necessary, in the middle of the cell



FM

- The problem with FM is that it is very wasteful: **each bit requires 2 flux reversal positions**, with a flux reversal being added for clocking every bit.

- Compared to more advanced encoding methods that **try to reduce** the number of clocking reversals, FM requires double (or more) the number of reversals for the same amount of data. This method was used on the earliest floppy disk drives, the immediate ancestors of those used in PCs. If you remember using "single density" floppy disks in the late 1970s or early 1980s, that designation commonly refers to magnetic storage using FM encoding.

- FM was actually made obsolete by MFM before the IBM PC was introduced, but it provides the basis for understanding MFM.

Modified Frequency Modulation (MFM)

- A refinement of the FM encoding method is modified frequency modulation, or MFM. MFM improves on FM by reducing the number of flux reversals inserted just for the clock.

- Instead of inserting a clock reversal at the start of every bit, one is inserted only between consecutive zeros.

- When a 1 is involved there is already a reversal (in the middle of the bit) so additional clocking reversals are not needed.

- When a zero is preceded by a 1, we similarly know there was recently a reversal and another is not needed.

- Only long strings of zeros have to be "broken up" by adding clocking reversals.

MFM

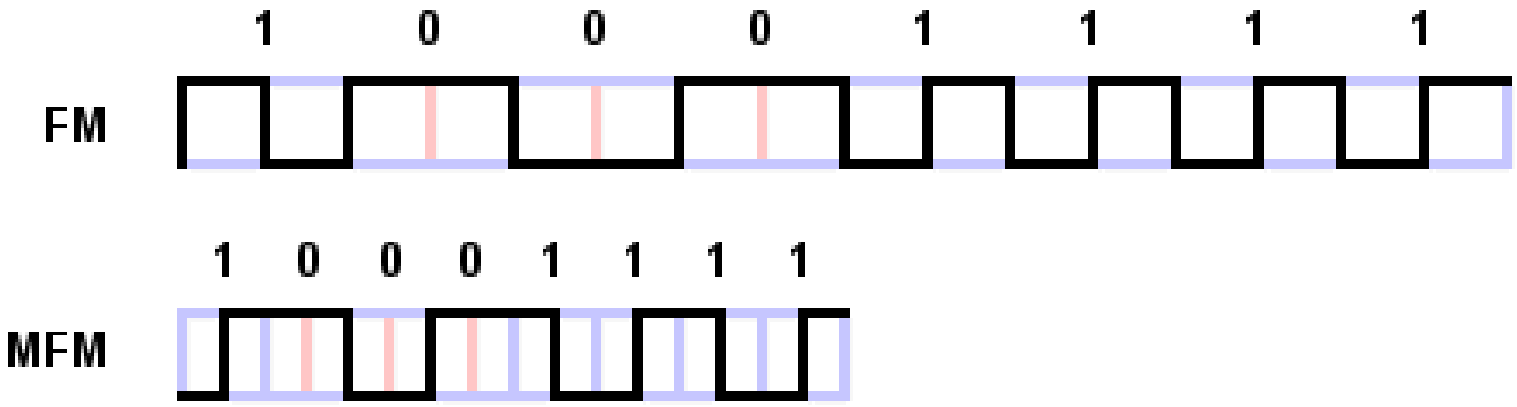
- This table shows the encoding pattern for MFM (where I have designated "R" to represent a flux reversal and "N" to represent no flux reversal). The average number of flux reversals per bit on a random bit stream pattern is 0.75. The best case (a repeating pattern of ones and zeros, "101010...") would be 0.25, the worst case (all ones or all zeros) would be 1:

Bit Pattern	Encoding Pattern	Flux Reversals Per Bit	Bit Pattern Commonality In Random Bit Stream
0 (preceded by 0)	RN	1	25%
0 (preceded by 1)	NN	0	25%
1	NR	1	50%
Weighted Average		0.75	100%

1x0,25 + 0x 0.25 +1*0.5=0.75

MFM

- FM and MFM encoding write waveform for the byte "10001111".
- As you can see, MFM encodes the same data in half as much space, by using half as many flux reversals per bit of data.



MFM

- MFM encoding was used on the earliest hard disks, and also on floppy disks.
- Since the MFM method about doubles the capacity of floppy disks compared to earlier FM ones, these disks were called "double density". In fact, MFM is still the standard that is used for floppy disks today.
- For hard disks it was replaced by the more efficient RLL methods. This did not happen for floppy disks, presumably because the need for more efficiency was not nearly so great, compared to the need for backward compatibility with existing

Run Length Limited (RLL)

- An improvement on the MFM encoding technique used in earlier hard disks and used on all floppies is ***run length limited* or *RLL***.

- This is a more sophisticated coding technique, or more correctly stated, "family" of techniques.
- I say that RLL is a family of techniques because
- there are 2 primary parameters that define how RLL works,
- and therefore, there are several different variations.

Run Length Limited (RLL)

- FM encoding has a simple one-to-one correspondence between the bit to be encoded and the flux reversal pattern.
- You only need to know the value of the current bit.
- MFM improves encoding efficiency over FM by more intelligently controlling where clock transitions are added into the data stream; this is enabled by considering not just the current bit but also the one before it. That's why there are is a different flux reversal pattern for a 0 preceded by another 0, and for a 0 preceded by a 1.
- This "looking backwards" allows improved efficiency by letting the controller consider more data in deciding when to add clock reversals.

Run Length Limited (RLL)

- RLL takes this technique one step further. It **considers groups of several bits** instead of encoding one bit at a time.
- The idea is to mix clock and data flux reversals to allow for even denser packing of encoded data, to improve efficiency.

- The **two parameters that define RLL** are the *run length* and the *run limit* (and hence the name).

- The word "**run**" here refers to a sequence of spaces in the output data stream without flux reversals.
- **run length** is the ***minimum*** spacing between flux reversals
- run limit is the ***maximum*** spacing between them.

- As mentioned before, the amount of time between reversals cannot be too large or the read head can get out of sync and lose track of which bit is where.

RLL

- The particular variety of RLL used on a drive is expressed as "RLL (X,Y)" or "X,Y RLL" where X is the run length and Y is the run limit.

- The most commonly used types of RLL in hard drives are "RLL (1,7)", also seen as **"1,7 RLL**"; and "RLL (2,7)" ("**2,7 RLL**").

- Alright, now consider the spacing of potential flux reversals in the encoded magnetic stream. In the case of "2,7", this means that the the smallest number of "spaces" between flux reversals is 2, and the largest number is 7.
- To create this encoding, a set of patterns is used to represent various bit sequences, as shown in the table below ("R" is a reversal, "N" no reversal, just as with the other data encoding examples):

RLL

- The controller these patterns by parsing the bit stream to be encoded, and matching the stream based on the bit patterns it encounters. If we were writing the byte "10001111" (8Fh), this would be matched as "10-0011-11" and encoded as "NRNN-NNNNRNNN-RNNN". Note that the since every pattern above ends in "NN", the minimum distance between reversals is indeed two. The maximum distance would be achieved with consecutive "0011" patterns, resulting in "**NNNNRNNN-NNNNRNNN**" or seven non-reversals between reversals. Thus, RLL (2,7).

Bit Pattern	Encoding Pattern	Flux Reversals Per Bit	Bit Pattern Commonality In Random Bit Stream
11	RNNN	1/2	25%
10	NRNN	1/2	25%
011	NNRNNN	1/3	12.5%
010	RNNRNN	2/3	12.5%
000	NNNRNN	1/3	12.5%
0010	NNRNNRNN	2/4	6.25%
0011	NNNNRNNN	1/4	6.25%
Weighted Average		0.4635	100%

RLL

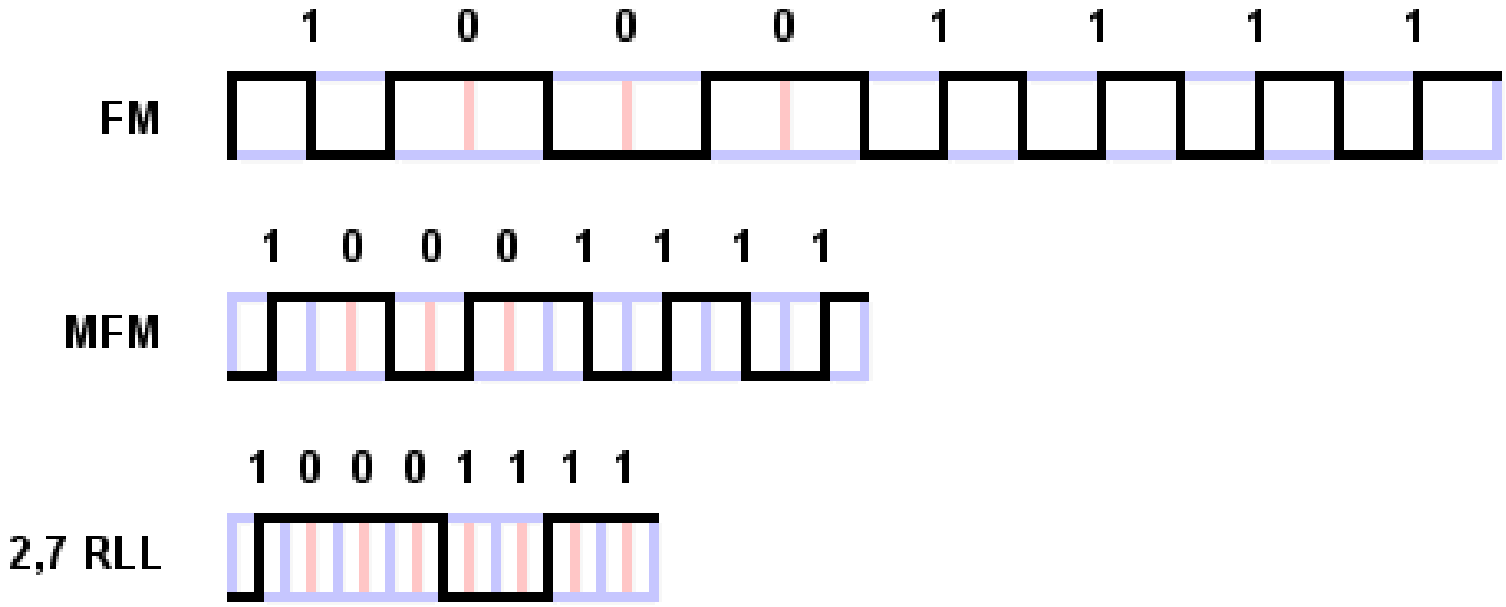
- Comparing the table above to the ones for FM and MFM, a few things become apparent. The most obvious is the increased complexity: seven different patterns are used, and up to four bits are considered a time for encoding.

- The average number of flux reversals per bit on a random bit stream pattern is 0.4635, or about 0.50. This is about a third of the requirement for FM (and about two thirds that of MFM).

- So relative to FM, data can be packed into one third the space. (For the example byte "10001111" we have been using, RLL requires 3 "R"s; MFM would require 7, and FM would need 13.)

RLL

- 2,7 RLL, FM and MFM encoding write waveform for the byte "10001111".
- RLL improves further on MFM by reducing the amount of space required for the same data bits to one third that required for regular FM encoding.



- Due to its greater efficiency, RLL encoding has replaced MFM everywhere but on floppy disks, where MFM continues to be used for historical compatibility reasons.

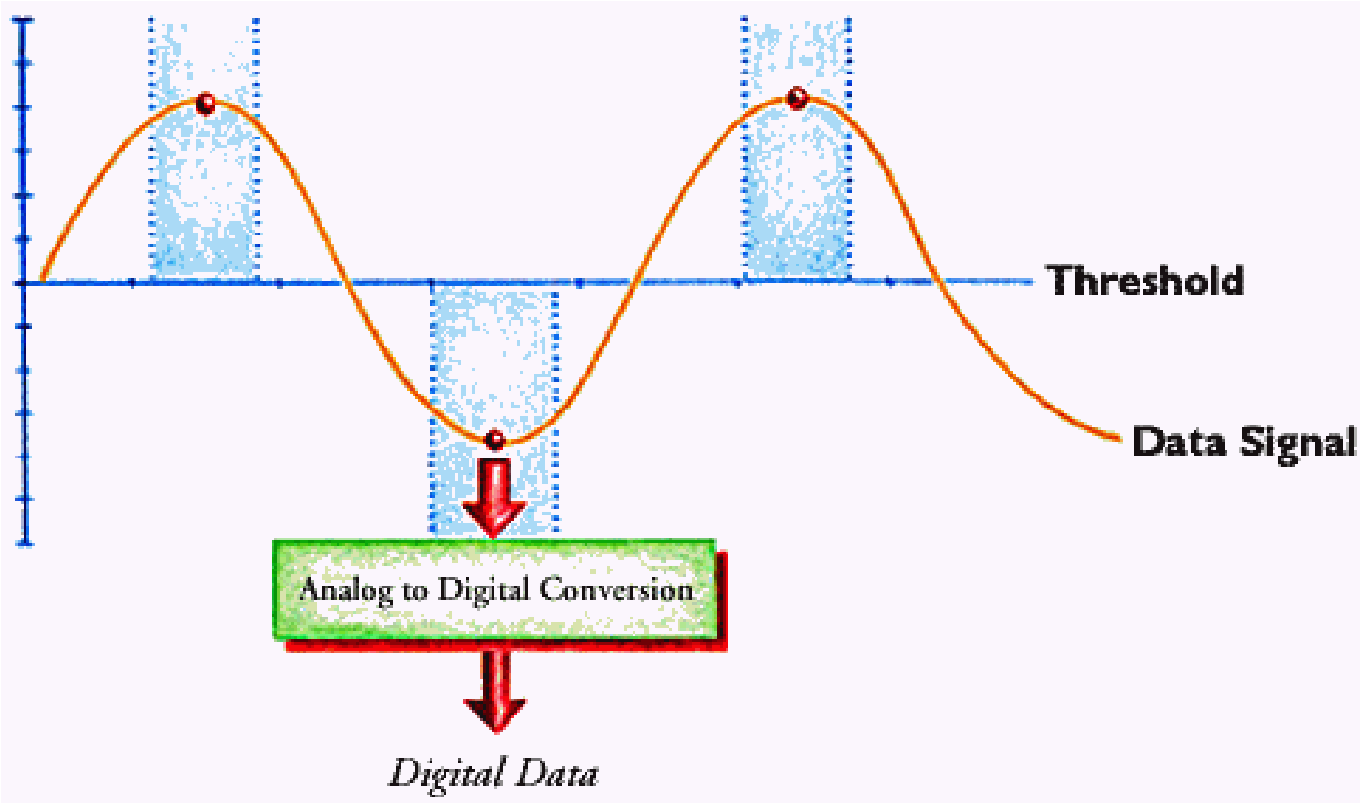
Partial Response, Maximum Likelihood (PRML)

- Standard read circuits work by detecting flux reversals and interpreting them based on the encoding method that the controller knows has been used on the platters to record bits.
- The data signal is read from the disk using the head, amplified, and delivered to the controller.
- The controller converts the signal to digital information by analyzing it continuously, synchronized to its internal clock, and looking for small voltage spikes in the signal that represent flux reversals.

- This traditional method of reading and interpreting hard disk data is called *peak detection*.

PRML

- Conceptual drawing demonstrating the principles behind analog peak detection.
- The circuitry scans the data read from the disk **looking for positive or negative "spikes"** that represent flux reversals on the surface of the hard disk platters

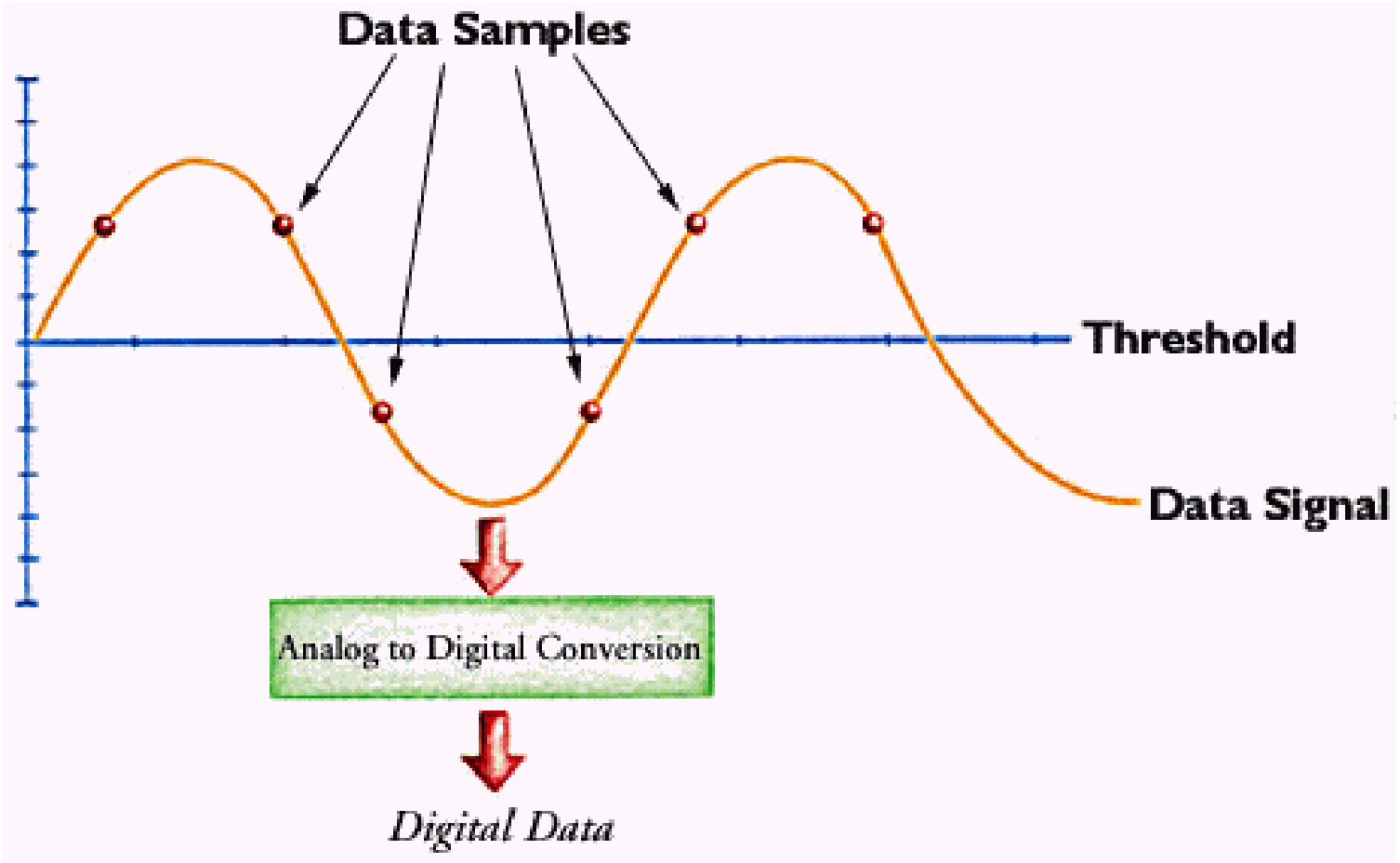


PRML

- This method works fine as long as the peaks are large enough to be picked out from the background noise of the signal. As data density increases, the flux reversals are packed more tightly and the signal becomes much more difficult to analyze, because the peaks get very close together and start to interfere with each other.
- This can potentially cause bits to be misread from the disk. Since this is something that must be avoided, in practical terms what happens instead is that the maximum areal density on the disk is limited to ensure that interference does not occur.
- To take the next step up in density, the magnetic fields must be made weaker. This reduces interference, but causes peak detection to be much more difficult.
- At some point it becomes very hard for the circuitry to actually tell where the flux reversals are.

PRML

- Conceptual drawing demonstrating the principles behind PRML.
- The data stream is sampled and analyzed using digital signal processing techniques



PRML

- While this may seem like an odd (and unreliable) way to read data from a hard disk, it is in fact reliable enough that PRML, and its successor, EPRML, have become the standard for data decoding on modern hard disks.
- PRML allows areal densities to be increased by a full 30-40% compared to standard peak detection, resulting in much greater capacities in the same number of platters.

Extended PRML (EPRML)

- An evolutionary improvement on the PRML design has been developed over the last few years. Called extended partial response, maximum likelihood, extended PRML or just EPRML, this advance was the result of engineers tweaking the basic PRML design to improve its performance.
- EPRML devices work in a similar way to PRML ones: they are still based on analyzing the analog data stream coming form the read/write head to determine the correct data sequence.
- They just use better algorithms and signal-processing circuits to enable them to more effectively and accurately interpret the information coming from the disk.

EPRML

- The chief benefit of using EPRML is that due to its higher performance, areal density (or more correctly, the linear component of areal density) can be increased without increasing the error rate. Claims regarding this increase range from around 20% to as much as 70%, compared to "regular" PRML. Those numbers represent a fairly significant improvement.
- EPRML has now been widely adopted in the hard disk industry and is replacing PRML on new drives.

Hard Disk Tracks, Cylinders and Sectors

- All information stored on a hard disk is recorded in **tracks,** which are concentric circles placed on the surface of each platter, much like the annual rings of a tree. The tracks are numbered, starting from zero, starting at the outside of the platter and increasing as you go in. A modern hard disk has tens of thousands of tracks on each platter.

- A platter from a 5.25" hard disk, with 20 concentric tracks drawn over the surface. Each track is divided into 16 imaginary sectors.

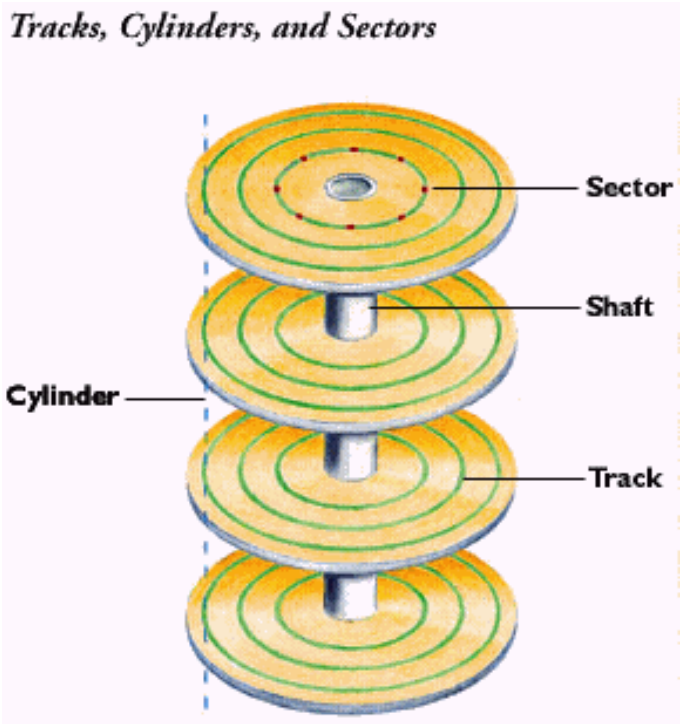
Hard Disk Tracks, Cylinders and Sectors

- Data is accessed by moving the heads from the inner to the outer part of the disk, driven by the head actuator. This organization of data allows for easy access to any part of the disk, which is why disks are called *random access* storage devices.

- Each track can hold many thousands of bytes of data. It would be wasteful to make a track the smallest unit of storage on the disk, since this would mean small files wasted a large amount of space. Therefore, each track is broken into smaller units called *sectors*.

- Each sector holds 512 bytes of user data, plus as many as a few dozen additional bytes used for internal drive control and for error detection and correction.

The Difference Between Tracks and Cylinders



- This diagram illustrates what "cylinder" means on on a hard disk. This conceptual hard disk spindle has four platters, and each platter has three tracks shown on it. The cylinder indicated would be made up of the 8 tracks (2 per surface) intersected by the dotted vertical line shown.*Image* © Quantum Corporation
- For most practical purposes, there really isn't much difference between tracks and cylinders--its basically a different way of thinking about the same thing. The addressing of individual sectors of the disk is traditionally done by referring to cylinders, heads and sectors (CHS). Since a cylinder is the collection of track numbers located at all of the heads of the disk, the specification "track number plus head number" is equal to "(cylinder number plus head number) plus head number", which is thus the same as "track number plus head number".

Track Density and Areal Density

- The ***track density*** of a hard disk refers, unsurprisingly, to how tightly packed the tracks are on the surface of each platter. Every platter has the same track density. The greater the track density of a disk, the more information that can be placed on the hard disk. Track density is one component of areal density, which refers to the number of bits that can be packed into each unit of area on the surface of the disk. More is better--both in terms of capacity and performance.

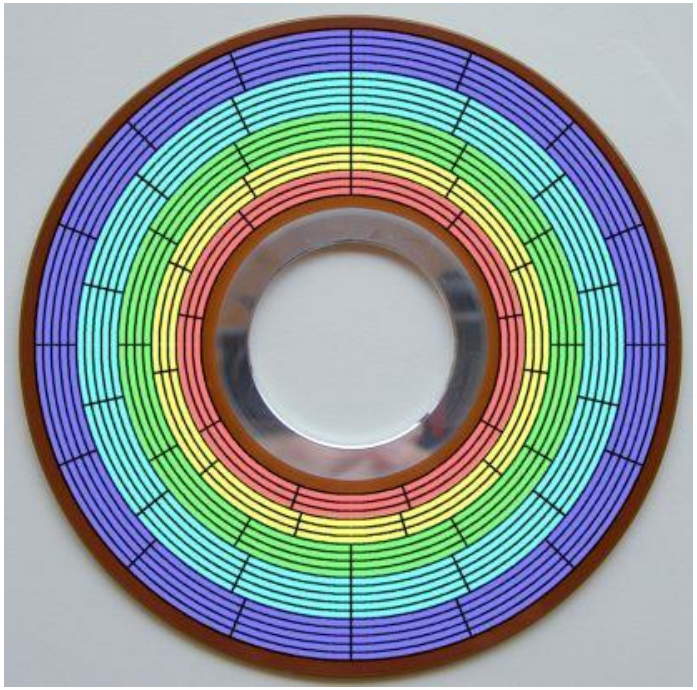
- The earliest PC hard disks had only a few hundred tracks on them, and used larger 5.25" form factor platters, resulting in a track density of only a few hundred tracks per inch. **Modern hard** disks have *tens of thousands* of tracks and can have a density of **30,000 tracks per inch** or more.

- The chief obstacle to increasing track density is making sure that the tracks don't get close enough together that reading one track causes the heads to pick up data from adjacent tracks. To avoid this problem, magnetic fields are made weaker to prevent interference, which leads to other design impacts, such as the requirement for better read/write head technologies and/or the use of PRML methods to improve signal detection and processing.

Zoned Bit Recording

- One way that capacity and speed have been improved on hard disks over time is by improving the utilization of the larger, outer tracks of the disk. The first hard disks were rather primitive affairs and their controllers couldn't handle complicated arrangements that changed between tracks. As a result, every track had the same number of sectors. The standard for the first hard disks was 17 sectors per track.
- Of course, the tracks are concentric circles, and the ones on the outside of the platter are much larger than the ones on the inside--typically double the circumference or more. Since there is a constraint on how tight the inner circles can be packed with bits, they were packed as tight as was practically possible given the state of technology, and then the outer circles were set to use the same number of sectors by reducing their bit density. This means that the outer tracks were greatly underutilized, because in theory they could hold many more sectors given the same linear bit density limitations.
- To eliminate this wasted space, modern hard disks employ a technique called **zoned bit recording (ZBR),** also sometimes called multiple zone recording or even just zone recording. With this technique, tracks are grouped into zones based on their distance from the center of the disk, and each zone is assigned a number of sectors per track. As you move from the innermost part of the disk to the outer edge, you move through different zones, each containing more sectors per track than the one before. This allows for more efficient use of the larger tracks on the outside of the disk.

Zoned Bit Recording



- A graphical illustration of zoned bit recording. This model hard disk has 20 tracks. They have been divided into five zones, each of which is shown as a different color. The **blue zone has 5 tracks, each with 16 sectors**; the **cyan zone 5 tracks of 14 sectors each**; the green zone 4 tracks of 12 sectors; the yellow 3 tracks of 11 sectors, and the red 3 tracks of 9 sectors. You can see that the size (length) of a sector remains fairly constant over the entire surface of the disk (contrast to the non-ZBR diagram on this page.) If not for ZBR, if the inner-most zone had its data packed as densely as possible, every track on this hard disk would be limited to only 9 sectors, greatly reducing capacity.

Zoned Bit Recording

- One interesting side effect of this design is that the raw data transfer rate (sometimes called the media transfer rate) of the disk when reading the outside cylinders is much higher than when reading the inside ones. This is because the outer cylinders contain more data, but the angular velocity of the platters is constant regardless of which track is being read (note that this constant angular velocity is *not* the case for some technologies, like older CD-ROM drives!) Since hard disks are filled from the outside in, the fastest data transfer occurs when the drive is first used.
- Sometimes, people benchmark their disks when new, and then many months later, and are surprised to find that the disk is getting slower! In fact, the disk most likely has not changed at all, but the second benchmark may have been run on tracks closer to the middle of the disk. (Fragmentation of the file system can have an impact as well in some cases.)

Zoned Bit Recording

- As an example, the table below shows the zones used by a 3.8 GB Quantum Fireball TM hard disk, which has a total of 6,810 user data tracks on each platter surface. Also included is the raw data transfer rate for each zone; notice how it decreases as you move from the outer edge of the disk (zone 0) to the hub of the disk (zone 14)--the data transfer rate at the edge is almost double what it is in the middle:

Zone	Tracks in Zone	Sectors Per Track	Data Transfer Rate (Mbits/s)
0	454	232	92.9
1	454	229	91.7
2	454	225	90.4
3	454	225	89.2
4	454	214	85.8
5	454	205	82.1
6	454	195	77.9
7	454	185	74.4
8	454	180	71.4
9	454	170	68.2
10	454	162	65.2
11	454	153	61.7
12	454	142	57.4
13	454	135	53.7
14	454	122	49.5

(From Quantum Fireball TM Product Manual, © 1996 Quantum Corporation.)

Zoned Bit Recording

- A couple of additional thoughts on this data. First, having the same number of tracks per zone is not a requirement; that is just how Quantum set up this disk family. (Compare to the newer IBM drive below.) Second, notice how much larger the sector per track numbers are, compared to the 17 of the earliest disks! Modern drives can pack a lot of storage into a track. Also, this is a 1996-era drive; modern units have even higher numbers of sectors per track in all zones, and much higher data transfer rates. Here's the same chart for the 20 GB/platter, 5400 RPM IBM 40GV drive:

Zone	Tracks in Zone	Sectors Per Track	Data Transfer Rate (Mbits/s)
0	624	792	372.0
1	1,424	780	366.4
2	1,680	760	357.0
3	1,616	740	347.6
4	2,752	720	338.2
5	2,880	680	319.4
6	1,904	660	310.0
7	2,384	630	295.9
8	3,328	600	281.8
9	4,432	540	253.6
10	4,528	480	225.5
11	2,192	440	206.7
12	1,600	420	197.3
13	1,168	400	187.9
14	18,15	370	173.8

(From Deskstar 40GV and 75GXP Product Manual, © 2000 IBM Corporation.)

Zoned Bit Recording

- The standard BIOS settings for IDE/ATA hard disks only allow the specification of a single number for "sectors per track". Since all modern hard disks use ZBR and don't *have* a single number of sectors per track across the disk, they use **logical geometry** for the BIOS setup. IDE hard disks up to 8.4 GB usually tell the BIOS 63 sectors per track and then translate to the real geometry internally; no modern drive uses 63 sectors on *any* track, much less all of them. Hard drives over 8.4 GB can't have their parameters expressed using the IDE BIOS geometry parameters anyway (because the regular BIOS limit is 8.4 GB) so these drives always have 63 sectors per track as "dummy" geometry parameters, and are accessed using logical block addressing. (See here for further discussion.)
- All of the above is one reason why modern drives are low-level formatted at the factory. The hard disk controller has to know the intricate details of the various recording zones, how many sectors are in each track for each zone, and how everything is organized.

Write Precompensation

- As discussed in the section on zoned bit recording, **older hard** disks used the **same number of sectors per track**. This meant that older disks had a **varying bit density as you moved** from the outside edge to the inner part of the platter.
- Many of these older **disks required** that an adjustment be made when **writing the inside tracks**, and a setting was placed in the BIOS to allow the user to specify at what track number this compensation was to begin.

- This entire matter is **no longer relevant to modern hard disks**, but the BIOS setting remains for compatibility reasons. Write precompensation is not done with today's drives; even if it were, the function would be implemented within the integrated controller and would be transparent to the user.

Interleaving

- A common operation when working with a hard disk is reading or writing a number of sectors of information in sequence. After all, a sector only contains 512 bytes of user data, and most files are much larger than that. Let's assume that the sectors on each track are numbered consecutively, and say that we want to read the first 10 sectors of a given track on the hard disk. Under ideal conditions, the controller would read the first sector, then immediately read the second, and so on, until all 10 sectors had been read. Just like reading 10 words in a row in this sentence.
- However, the **physical sectors on a track are adjacent to each other** and not separated by very much space. Reading sectors consecutively requires a certain amount of speed from the hard disk controller. The platters never stop spinning, and as soon as the controller is done reading all of sector #1, it has little time before the start of sector #2 is under the head. Many older controllers used with early hard disks did not have sufficient processing capacity to be able to do this. They would not be ready to read the second sector of the track until after the start of the second physical sector had already spun past the head, at which point it would be too late.
- If the controller *is* slow in this manner, and no compensation is made in the controller, the controller must wait for almost an entire revolution of the platters before the start of sector #2 comes around and it can read it. Then, of course, when it tried to read sector #3, the same thing would happen, and another complete rotation would be required. All this waiting around would kill performance: if a disk had 17 sectors per track, it would take 17 times as long to read those 10 sectors as it should have in the ideal case!

Interleaving

- To address this problem, older controllers employed a function called *interleaving*, allowing the setting of a disk parameter called the *interleave factor*. When interleaving is used, the sectors on a track are logically re-numbered so that they do not correspond to the physical sequence on the disk. The goal of this technique is to arrange the sectors so that their position on the track matches the speed of the controller, to avoid the need for extra "rotations". Interleave is expressed as a ratio, "N:1", where "N" represents how far away the second logical sector is from the first, how far the third is from the second, and so on.
- An example is the easiest way to demonstrate this method. The standard for older hard disks was 17 sectors per track. Using an interleave factor of **1:1**, the sectors would be numbered **1, 2, 3, .. , 17**, and the problem described above with the controller not being ready in time to read sector #2 would often occur for sequential reads. Instead, an interleave factor of **2:1** could be used. With this arrangement, the sectors on a 17-sector track would be numbered as follows: **1, 10, 2, 11, 3, 12, 4, 13, 5, 14, 6, 15, 7, 16, 8, 17, 9**. Using this interleave factor means that while sector 1 is being processed, sector *10* is passing under the read head, and so when the controller is ready, sector 2 is just arriving at the head. To read the entire track, two revolutions of the platters are required. This is twice as long as the ideal case (1:1 interleaving with a controller fast enough to handle it) but it is almost 90% better than what would result from using 1:1 interleaving with a controller that is too slow (which would mean 17 rotations were required).
- What if the controller was too slow for a 2:1 interleave? It might only be fast enough to read every third physical sector in sequence. If so, an interleave of **3:1** could be used, with the sectors numbered as follows: **1, 7, 13, 2, 8, 14, 3, 9, 15, 4, 10, 16, 5, 11, 17, 6, 12**. Again here, this would reduce performance compared to 2:1, if the controller was fast enough for 2:1, but it would greatly *improve* performance if the controller couldn't handle 2:1.

Interleaving

- So this begs the question then: how do you know what interleave factor to use? Well, on older hard disks, the interleave factor was one parameter that had to be tinkered with to maximize performance. Setting it too conservatively caused the drive to not live up to its maximum potential, but setting it too aggressively could result in severe performance hits due to extra revolutions being needed. The perfect interleave setting depended on the speeds of the hard disk, the controller, and the system. Special utilities were written to allow the analysis of the hard disk and controller, and would help determine the optimal interleave setting. The interleave setting would be used when the drive was low-level formatted, to set up the sector locations for each track.
- On modern disk drives, the interleave setting is always 1:1. Controller too slow? Ha! Today's controllers are so fast, much of the time they sit around waiting for the platters, tapping their virtual fingers. How did this situation come to change so drastically in 15 years? Well, it's pretty simple. The spindle speed of a hard disk has increased from 3,600 RPM on the first hard disks, to today's standards of 5,400 to 10,000 RPM. An increase in speed of 50% to 177%. The faster spindle speed means that much less time for the controller to be ready before the next physical sector comes under the head. However, look at what processing power has done in the same time frame: CPUs have gone from 4.77 MHz speeds to the environs of 1 GHz; an increase of over 20,000%! The speed of other chips in the PC and its peripherals have similarly gotten faster by many multiples.

Cylinder and Head Skew

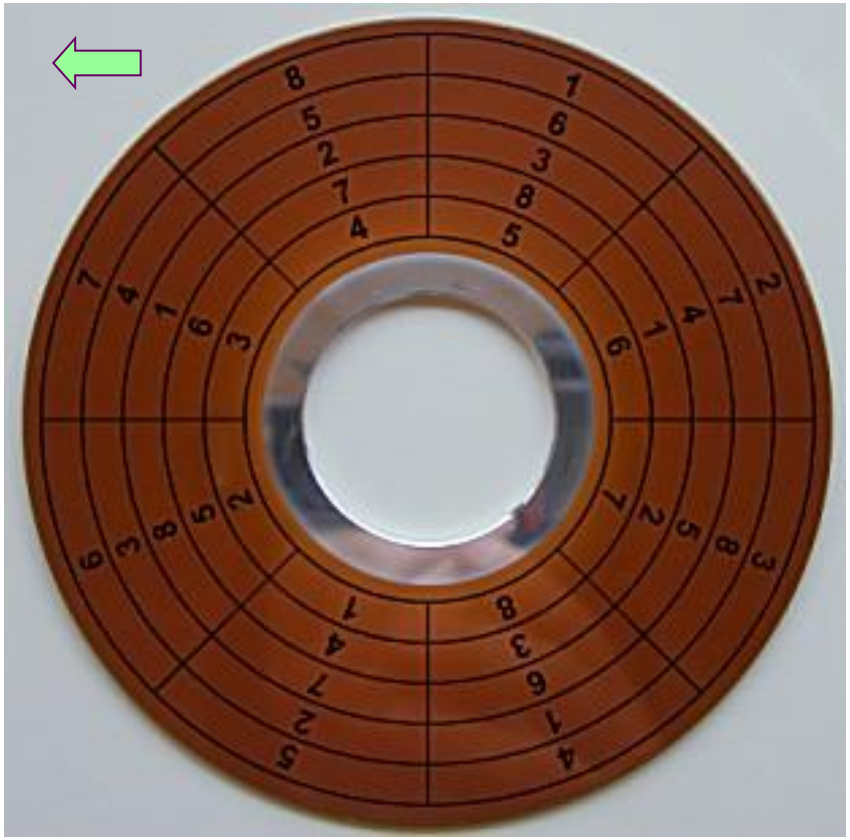
- Sector interleaving was once used on older hard disks to ensure that the sectors were efficiently spaced on the track. This was needed to ensure that sector #2 didn't rotate past the head while sector #1 was being processed. The high-speed disk controllers on modern drives are now fast enough that they no longer are a performance-limiting factor in how the sectors on the disk are arranged.
- However, there are other delay issues within the drive that require spacing to be optimized in even the fastest drives, to maximize performance. And unlike the interleaving situation, these delays are caused by electromechanical concerns and are therefore likely to be with us for as long as hard drives use their current general design.
- The first issue is the delay in time incurred when **switching between cylinders** on the **hard disk**, called appropriately enough, **cylinder switch time**. Let's imagine that we "lined up" all of the tracks on a platter so that the first sector on each track started at the same position on the disk. Now let's say that we want to read the entire contents of two consecutive tracks, a fairly common thing to need to do. We read all the sectors of track #1 (in sequence, since we can use a 1:1 interleave) and then switch to track #2 to start reading it at its first sector.

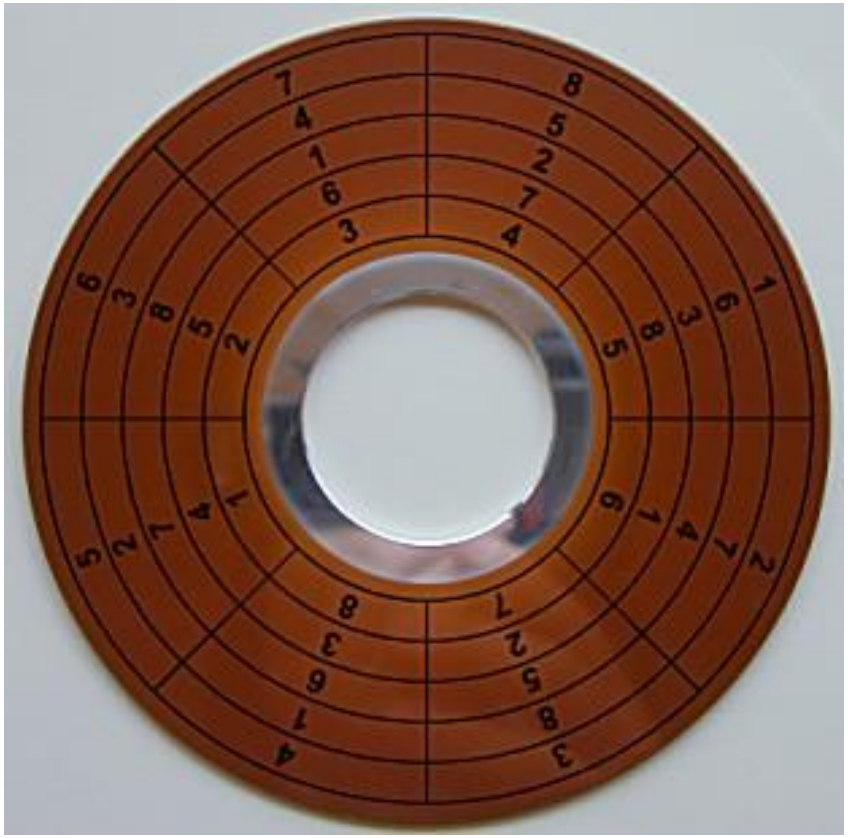
Cylinder and Head Skew

- The problem here is that it takes time to physically move the heads (or more actually, the actuator assembly) to track #2. In fact, it often takes a millisecond or more. Let's consider a modern 10,000 RPM drive. The IBM Ultrastar 72ZX has a specification of only 0.6 milliseconds for seeking from one track to an adjacent one. That's actually quite fast by today's standards. But consider that in that amount of time, a 10,000 RPM drive will perform approximately 10% of a complete revolution of the platters! If sector #1 on track #2 is lined up with sector #1 on track #1, it will be long gone by the time we switch from track #1 to track #2. We'd have to wait for the remaining 90% of a revolution of the platters to do the next read, a big performance penalty. This problem isn't as bad as the interleave one was, because it occurs only when changing tracks, and not every sector. But it's still bad, and it's avoidable.
- The issue is avoided by offsetting the start sector of adjacent tracks to minimize the likely wait time (rotational latency) when switching tracks. This is called *cylinder skew*. Let's say that in the particular zone where tracks #1 and #2 are, there are 450 sectors per track. If 10% of the disk spins by on a track-to-track seek, 45 sectors go past. Allowing some room for error and controller overhead, perhaps the design engineers would shift each track so that sector #1 of track #2 was adjacent to sector #51 of track #1. Similarly, sector #1 of track #3 would be adjacent to sector #51 of track #2 (and hence, adjacent to sector #101 of track #1). And so on. By doing this, we can read multiple adjacent tracks virtually seamlessly, and with no performance hit due to unnecessary platter rotations.
- The same problem, only to a lesser degree, occurs when we change heads within a cylinder. Here there is no physical movement, but it **still takes time for the switch to be made from reading one head to reading another**, so it makes sense to offset the start sector of tracks within the same cylinder so that after reading from the first head/track in the cylinder, we can switch to the next one without losing our "pace". This is called ***head skew***. Since switching heads takes much less time than switching cylinders, head skew usually means a smaller number of sectors being offset than cylinder skew does.

Cylinder and Head Skew

- These two diagrams illustrate the concept of cylinder and head skew. Assume that these platters spin counter-clockwise (as seen from your vantage point) and that they are adjacent to each other (they might be the two surfaces of the same platter.) They each have a cylinder skew of 3, meaning that adjacent tracks are offset by three sectors. In addition, the platter on the right has a head skew of one relative to the one on the left.





Sector Format and Structure

- In the PC world, each sector of a hard disk can store 512 bytes of user data. (There are some disks where this number can be modified, but 512 is the standard, and found on virtually all hard drives by default.) Each sector, however, actually holds much more than 512 bytes of information. Additional bytes are needed for control structures and other information necessary to manage the drive, locate data and perform other "support functions". The exact details of how a sector is structured depends on the drive model and manufacturer. However, the contents of a sector usually include the following general elements:
- **ID Information:** Conventionally, space is left in each sector to identify the **sector's number and location**. This is used for locating the sector on the disk. Also included in this area is status information about the sector. For example, a bit is commonly used to indicate if the sector has been marked defective and remapped.
- **Synchronization Fields:** These are used internally by the drive controller to guide the read process.
- **Data:** The actual data in the sector.
- **ECC:** Error correcting code used to ensure data integrity.
- **Gaps:** One or more "spacers" added as necessary to separate other areas of the sector, or provide time for the controller to process what it has read before reading more bits.
- **Note:** In addition to the sectors, each containing the items above, space on each track is also used for **servo information** (on embedded servo drives, which is the design used by all modern units).

Sector Format and Structure

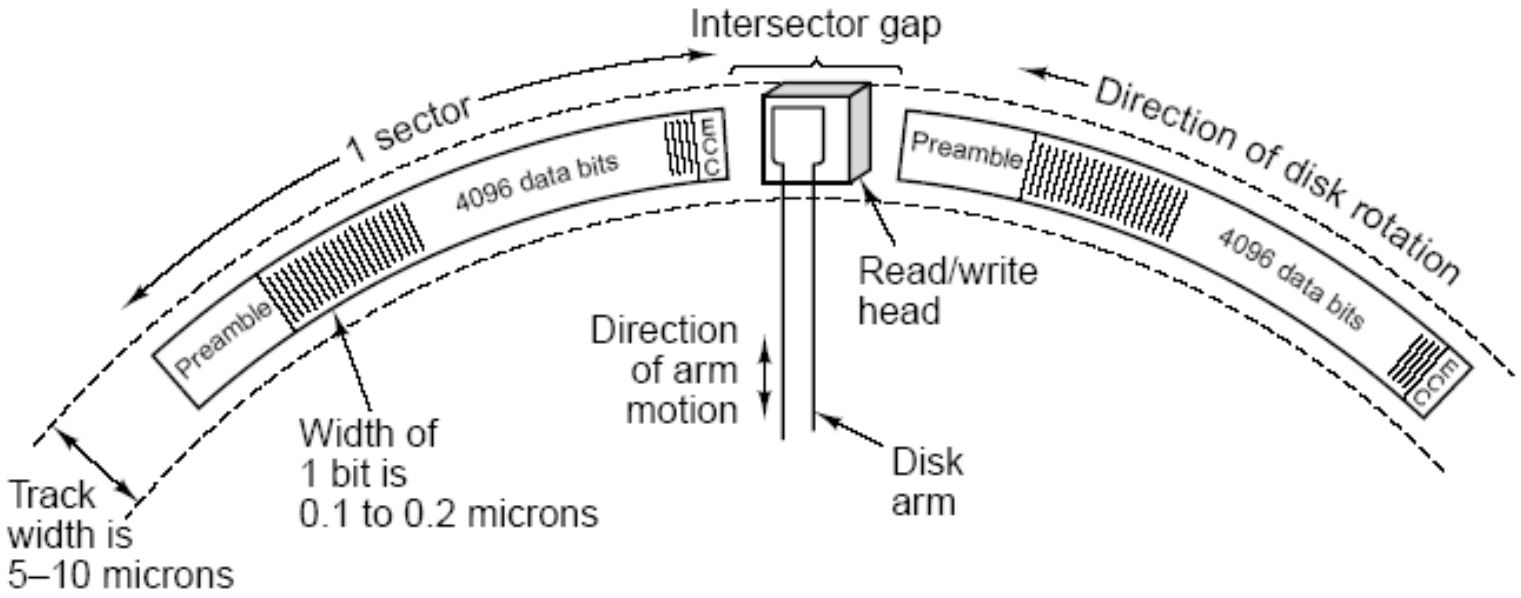


Figure 2-19. A portion of a disk track. Two sectors are illustrated.

Sector Structure

P = Preamble
D = Data Sync Mark
ECC = Error Correcting Code
G = Inter Sector Gap

Hard Disk Formatting and Capacity

- Most PC users are familiar with the concept that a hard disk--in fact, all storage media--must be formatted before it can be used. There is usually some confusion, however, regarding exactly what formatting means and what it does. This is exacerbated by the fact that modern hard disks are not formatted in the same way that older ones were, and also the fact that the utilities used for formatting behave differently when acting on hard disks than when used for floppy disks.
- This section takes a look at issues surrounding disk formatting and capacity, discusses unformatted and formatted hard disk capacity, and looks briefly at formatting utilities.

Two Formatting Steps

- Many PC users don't realize that formatting a hard disk isn't done in a single step. In fact, three steps are involved:
- **Low-Level Formatting:** This is the "true" formatting process for the disk. It **creates the physical structures** (**tracks, sectors, control information**) on the hard disk. Normally, this step begins with the hard disk platters "clean", containing no information. It is discussed in more detail here.
- **Partitioning:** This process divides the disk into logical "pieces" that become different hard disk volumes (drive letters). This is an operating system function and is discussed in detail in its own section.
- **High-Level Formatting:** This final step is also an operating-system-level command. It defines the logical structures on the partition and places at the start of the disk any necessary operating system files.
- As you can see, two of the three steps are "formatting", and this dual use of the word is a big part of what leads to a lot of confusion when the term "formatting" is used. Another strange artifact of history is that the DOS "FORMAT" command behaves differently when it is used on a hard disk than when it is used on a floppy disk. Floppy disks have simple, standard geometry and cannot be partitioned, so the FORMAT command is programmed to automatically both low-level and high-level format a floppy disk, if necessary. For hard disks, however, FORMAT will only do a high-level format. Low-level formatting is performed by the controller for older drives, and at the factory for newer drives.

Low-Level Formatting

- **Low-Level Formatting**
- *Low-level formatting* is the process of outlining the positions of the tracks and sectors on the hard disk, and writing the control structures that define where the tracks and sectors are. This is often called a "true" formatting operation, because it really creates the physical format that defines where the data is stored on the disk. The first time that a low-level format ("LLF") is performed on a hard disk, the disk's platters start out empty. That's the last time the platters will be empty for the life of the drive. If an LLF is done on a disk with data on it already, the data is permanently erased (save heroic data recovery measures which are sometimes possible).
- If you've explored other areas of this material describing hard disks, you have learned that modern hard disks are much more precisely designed and built, and much more complicated than older disks. Older disks had the same number of sectors per track, and did not use dedicated controllers. It was necessary for the external controller to do the low-level format, and quite easy to describe the geometry of the drive to the controller so it could do the LLF. Newer disks use many complex internal structures, including zoned bit recording to put more sectors on the outer tracks than the inner ones, and embedded servo data to control the head actuator. They also transparently map out bad sectors. Due to this complexity, all modern hard disks are low-level formatted at the factory for the life of the drive. There's no way for the PC to do an LLF on a modern IDE/ATA or SCSI hard disk, and there's no reason to try to do so.
- Older drives needed to be re-low-level-formatted occasionally because of the thermal expansion problems associated with using stepper motor actuators. Over time, the tracks on the platters would move relative to where the heads expected them to be, and errors would result. These could be corrected by doing a low-level format, rewriting the tracks in the new positions that the stepper motor moved the heads to. This is totally unnecessary with modern voice-coil-actuated hard disks.
- **Warning:** You should never attempt to do a low-level format on an IDE/ATA or SCSI hard disk. Do not try to use BIOS-based low-level formatting tools on these newer drives. It's unlikely that you will damage anything if you try to do this (since the drive controller is programmed to ignore any such LLF attempts), but at best you will be wasting your time. A modern disk can usually be restored to "like-new" condition by using a zero-fill utility

High-Level Formatting

- After low-level formatting is complete, we have a disk with tracks and sectors--but nothing written on them. *High-level formatting* is the process of writing the file system structures on the disk that let the disk be used for storing programs and data. If you are using DOS, for example, the DOS FORMAT command performs this work, writing such structures as the master boot record and file allocation tables to the disk. High-level formatting is done after the hard disk has been partitioned, even if only one partition is to be used. See here for a full description of DOS structures, also used for Windows 3.x and Windows 9x systems.
- The distinction between high-level formatting and low-level formatting is important. It is not necessary to low-level format a disk to erase it: a high-level format will suffice for most purposes; by wiping out the control structures and writing new ones, the old information is lost and the disk appears as new. (Much of the old data is still on the disk, but the access paths to it have been wiped out.) Under some circumstances a high-level format won't fix problems with the hard disk and a zero-fill utility may be necessary.
- Different operating systems use different high-level format programs, because they use different file systems. However, the low-level format, which is the real place where tracks and sectors are recorded, is the same.

Defect Mapping and Spare Sectoring

- Despite the precision manufacturing processes used to create hard disks, it is virtually impossible to create a disk with tens of millions of sectors and not have some errors show up. Imperfections in the media coating on the platter or other problems can make a sector inoperable. A problem with a sector, if uncorrected, would normally manifest as an error when attempting to read or write the sector, but can appear in other ways as well. Most of us have experienced these errors on occasion when using floppy disk drives.
- Modern disks use ECC to help identify when errors occur and in some cases correct them, however, there will still be physical flaws that ECC cannot overcome, and that therefore prevent parts of a disk from being used. Usually these are individual sectors that don't work, and they are appropriately enough called *bad sectors*. Tracks where there are bad sectors are sometimes called *bad tracks*.
- If you've ever used a disk information utility on a floppy disk (or on a very old hard disk), you've likely at some point seen a report showing a few kilobytes worth of bad sectors. However, if you run such a utility on a modern hard disk, you will normally never see any reports of bad sectors on the disk. Why is this?
- Sectors that are bad cannot be used to store data, for obvious reasons: they are bad because they cannot be trusted to reliably write and/or reproduce the data at a later time. It is therefore necessary for *some* part of the system to keep track of where they are, and not use them. The best way for this to be done is for the drive to detect and avoid them. If the drive does not do this, the operating system must do it. If any bad sectors are not detected until after they have been used, data loss will probably result.
- To allow for maximum reliability then, each disk drive is thoroughly tested for any areas that might have errors at the time it is manufactured. All the sectors that have problems or are thought to be unreliable, are recorded in a special table. This is called *defect mapping*. Some drives go even further than this, mapping out not only the sectors that are questionable, but the ones surrounding them as well. Some drives will map out entire tracks as a safety precaution.

Defect Mapping and Spare Sectoring

- On older hard disks, these problem areas were actually recorded right on the top cover of the disk, usually in hand-writing by the technician testing the drive! This process was necessary because low-level formatting was done by the company assembling the PC--or even the end-user--and this information was used to tell the controller which areas of the disk to avoid when formatting the disk. Part of the low-level format process was to have the person doing the LLF tell the controller which sectors were bad, so it would avoid them, and also tell any high-level format program not to try to use that part of the disk. These markings are what cause "bad sectors" reports to show up when examining older hard disks: these are the areas the disk has been told not to use. Since floppy disks are low-level formatted and high-level formatted at the same time, the same situation applies, even today. If any sectors cannot be reliably formatted, they are marked as "bad" and the operating system will stay away from them.
- While early PC users accepted that a few bad sectors on a drive was normal, there was something distasteful about plopping down $1,000 for a new hard disk and having it report "bad sectors" as soon as you turned it on. There is no way to produce 100% perfect hard disks without them costing a small fortune, so hard disk manufacturers devised an interesting compromise.
- On modern hard disks, a small number of sectors are *reserved* as substitutes for any bad sectors discovered in the main data storage area. During testing, any bad sectors that are found on the disk are programmed into the controller. When the controller receives a read or write for one of these sectors, it uses its designated substitute instead, taken from the pool of extra reserves. This is called *spare sectoring*. In fact, some drives have entire spare tracks available, if they are needed. This is all done completely transparently to the user, and the net effect is that all of the drives of a given model have the exact same capacity and there are no visible errors. This means that the operating system never sees the bad areas, and therefore never reports "bad sectors". They are still there though, just cleverly hidden.

Defect Mapping and Spare Sectoring

- Really, when you think about it, the hard disk companies are sacrificing a small amount of storage for "good looks". It would be more efficient to use all of the sectors on the disk and just map out the few bad ones. However, sometimes marketing wins out over engineering, and it seems that more people want the warm feeling of thinking they have a perfect drive, even if it costs them theoretical storage in the process. Today's drives are so enormous that few people would even care much anyway about a few extra megabytes, but that wasn't always the case!
- Due to spare sectoring, a brand new disk should not have any bad sectors. It is possible, however, for a modern IDE/ATA or SCSI hard disk to develop new bad sectors over time. These will normally be detected either during a routine scan of the hard disk for errors (the easy way) or when a read error is encountered trying access a program or data file (the hard way). When this happens, it is possible to tell the system to avoid using that bad area of the disk. Again, this can be done two ways. At the high level, the operating system can be told to mark the area as bad and avoid it (creating "bad sector" reports at the operating system level.). Alternately, the disk itself can be told at a low level to *remap* the bad area and use one of its spares instead. This is normally done by using a zero-fill or diagnostic utility, which will scan the entire disk surface for errors and tell the controller to map out any problem areas.
- **Warning:** Bad sectors on a modern hard disk are almost always an indication of a greater problem with the disk. A new hard disk should *never* have bad sectors on it; if you buy one that does have bad sectors, immediately return it to the vendor for exchange (and don't let them tell you "it's normal", because it isn't.) For existing hard disks, the vast majority of time, a single bad sector that appears will soon be accompanied by friends. While you can map out and ignore bad sectors, you should make sure to contact the vendor if you see bad sectors appearing during scans, and make sure the data is backed up as well. Personally, I will not use any hard disk that is developing bad sectors. The risk of data loss is too high, and hard drives today are inexpensive compared to the cost of even an hour or two of recovering lost data (which takes a lot more than an hour or two!) See here for more on troubleshooting hard disk errors.

- On some disks, remapped sectors cause a performance penalty. The drive first seeks and reads the sector header of the data, thinking it will be there; then, it sees that the sector has been remapped, and has to do another seek for the new sector. Newer drives using the No-ID sector format eliminate this problem by storing a format map, including sector remaps, in the memory of the drive's controller. See the discussion of sector format for more

Low-Level Format, Zero-Fill and Diagnostic Utilities

- Hard drive manufacturers have created for modern drives replacements for the old LLF utilities. They cause some confusion, because they are often still *called* "low-level format" utilities. The name is incorrect because, again, no utility that a user can run on a PC can LLF a modern drive. A more **proper name for this sort of program is a *zero-fill and diagnostic utility*. This software *does* work on the drive at a low level, usually including** the **following functions (and perhaps others**):
- **Drive Recognition Test:** Lets you test to see if the software can "see" the drive. This is the first step in ensuring that the drive is properly installed and connected.
- **Display Drive Details:** Tells you detailed information about the drive, such as its exact model number, firmware revision level, date of manufacture, etc.
- **Test For Errors:** Analyzes the entire surface of the hard disk, looking for problem areas (bad sectors) and instructing the integrated drive controller to remap them.
- **Zero-Fill:** Wipes off all data on the drive by filling every sector with zeroes. Normally a test for errors (as above) is done at the same time.

Unformatted and Formatted Capacity

- Some portion of the space on a hard disk is taken up by the formatting information that marks the **start and end of sectors**, ECC, and other "overhead". For this reason, a hard disk's storage total depends on if you are looking at the formatted or unformatted capacity. The difference can be quite significant: 20% or even more.
- Older drives that were typically low-level formatted by the user, often had their size listed in terms of unformatted capacity. For example, take the Seagate ST-412, the first drive used on the original IBM PC/XT in the early 1980s. The "12" in this model number refers to the drive's ***unformatted* capacity of 12.76 MB. Formatted, it is actually a 10.65 MB drive**.
- Now, let's be honest: stating the capacity of the hard disk in unformatted terms is lame. Since nobody can use a drive that is unformatted, the only thing that matters is the formatted capacity. Stating the drive in terms of unformatted capacity is not quite as bad as how tape drive manufacturers always report the size of their drives assuming 2:1 compression, of course. But it's still *lame*. :^)
- Fortunately, this is no longer an issue today. Since modern drives are always low-level formatted at the factory, it would be *extremely* weird to state their sizes in terms of unformatted capacity, and manufacturers have stopped doing this. In fact, there usually isn't any easy way to find *out* the unformatted capacity of new drives! So to take another example from our friends at Seagate, the ST-315330A, the "15330" refers to the drive's approximate *formatted* capacity, 15,364 MB (15.4 GB).

Binary vs. Decimal Capacity Measurements

- Computer measurements are expressed in both binary *and* decimal terms, often using the same notation. Due to a mathematical coincidence, the fact that 2^10 (1024) is almost the same number as 10^3 (1000), there are two similar but different ways to express a megabyte or a gigabyte. This phenomenon, and the general problems it causes, are discussed in detail in this fundamentals section. I also discuss there how and why I have begun using alternative measurement notations for binary numbers.
- The problems with binary and decimal are probably more noticed in the area of hard disk capacity than anywhere else. Hard disk manufacturers always use decimal figures for their products' capacity: a 72 GB hard disk has about 72,000,000,000 bytes of storage. However, hard disk makers also use binary numbers where they are normally used--for example, buffer capacities are expressed in binary kilobytes or megabytes--but the same notation ("kB" or "MB") is used as for decimal figures. Hard disks are large, and larger numbers cause the discrepancy between decimal and binary terms to be exaggerated. For example, a 72 GB hard disk, expressed in binary terms, is "only" 67 GB. Since most software uses binary terms, this difference in numbers is the source of frequent confusion regarding "where the rest of the gigabytes went". In fact, they didn't go anywhere. It's just a different way of expressing the same thing.
- This is also the source of much confusion surrounding 2.1 GB hard disks (or 2.1 GB hard disk volumes) and the 2 GB DOS limit on partition size. Since DOS uses binary gigabytes, and 2.1 GB hard disks are expressed in decimal terms, a 2.1 GB hard disk can in fact be entirely placed within a single DOS partition. 2.1 decimal gigabytes is actually 1.96 binary gigabytes. Another example is the BIOS limit on regular IDE/ATA hard disks, which is either 504 MB or 528 MB, depending on which "MB" you are talking about.

Hard Disk Geometry Specifications and Translation

- The generic term used to refer to the way the disk structures its data into platters, tracks and sectors, is its *geometry*. In the early days this was a relatively simple concept: the disk had a certain number of heads, tracks per surface, and sectors per track. These were entered into the BIOS set up so the PC knew how to access the drive, and that was basically that.
- With newer drives the situation is more complicated. The simplistic limits placed in the older BIOSes have persisted to this day, but the disks themselves have moved on to more complicated ways of storing data, and much larger capacities. The result is that tricks must be employed to ensure compatibility between old BIOS standards and newer hard disks.
- **Note:** These issues relate to IDE/ATA hard disks, not SCSI drives, which use a different addressing methodology.

Physical Geometry

- The physical geometry of a hard disk is the actual physical number of heads, cylinders and sectors used by the disk. On older disks this is the only type of geometry that is ever used--the physical geometry and the geometry used by the PC are one and the same. The original setup parameters in the system BIOS are designed to support the geometries of these older drives. Classically, there are three figures that describe the geometry of a drive: the number of cylinders on the drive ("C"), the number of heads on the drive ("H") and the number of sectors per track ("S"). Together they comprise the "CHS" method of addressing the hard disk. This method of description is described in more detail in this description of CHS mode addressing.
- At the time the PC BIOS interfaces to the hard disk were designed, hard disks were simple. They had only a few hundred cylinders, a few heads and all had the same number of sectors in each track. Today's drives do not have simple geometries; they use zoned bit recording and therefore do not have the same number of sectors for each track, and they use defect mapping to remove bad sectors from use. As a result, their geometry can no longer be described using simple "CHS" terms. These drives must be accessed using logical geometry figures, with the physical geometry hidden behind routines inside the drive controller. For a comparison of physical and logical geometry, see this page on logical geometry.
- Often, you have to request detailed specifications for a modern drive to find out the true physical geometry. Even then you might have problems--I called one major drive manufacturer when first writing the site, and the technician had no idea what I was talking about. He kept giving me the logical parameters and insisting they were the physical ones. Finally, I asked him how his drive could have 16 heads when it had only 3 platters, and he got very confused. :^)
- **Tip:** It's easy to tell if you are looking at physical or logical hard disk geometry numbers. Since no current hard drive has the same number of sectors on each track, if you are given a single number for "sectors per track", that *must* be a logical parameter. Also, I am aware of no current hard disk product that uses 8 platters and either 15 or 16 heads. However, all modern, larger IDE/ATA hard disks have a nominal logical geometry specification of 15 or 16 heads, so either of those numbers is a dead giveaway.

Logical Geometry

- When you perform a drive parameter autodetection in your system BIOS setup or look in your new IDE/ATA hard disk's setup manual to see what the drive parameters are, you are seeing the *logical geometry* values that the hard disk manufacturer has specified for the drive. Since newer drives use zoned bit recording and hence have ten or more values for sectors per track depending on which region of the disk is being examined, it is not possible to set up the disk in the BIOS using the physical geometry. Also, the BIOS has a limit of 63 sectors per track, and all newer hard disks *average* more than 100 sectors per track, so even without zoned bit recording, there would be a problem.
- Older hard disks that had simple structures and low capacity did not need special logical geometry. Their physical and logical geometry was the same. Take for example the Seagate ST-251, a 42.8 MB drive that was one of the most popular drives of its day. This drive's "CHS" physical geometry numbers are 820 cylinders, 6 heads, and 17 sectors, and those numbers are what is used by a system that has this drive.
- Newer drives cannot have their true geometries expressed using three simple numbers. To get around this issue, for disks 8.4 GB or smaller, the BIOS is given bogus parameters that give the approximate capacity of the disk, and the **hard disk controller is given intelligence so that it can do automatic** translation between the logical and physical geometry. The actual physical geometry is totally different, but the BIOS (and your system) need know nothing about this. Here's an example showing the difference between the physical and logical geometry for a sample drive, a 3.8 GB Quantum Fireball TM:

Logical Geometry

Specification	Physical Geometry	Logical Geometry
Read/Write Heads	6	16
Cylinders (Tracks per Surface)	6,810	7,480
Sectors Per Track	122 to 232	63
Total Sectors	7,539,840	7,539,840

- If you install this drive, as far as the system is concerned, the disk has 16 heads and 63 sectors on every track, and the hard disk itself takes care of all the "dirty work" of translating requests to their real internal locations. The physical geometry is totally hidden from view. The fact that both geometries equate to the same number of total sectors is *not* a coincidence. The purpose of the logical geometry is to enable access to the entire disk using terms that the BIOS can handle. The logical geometry could theoretically end up with a smaller number of sectors than the physical, but this would mean wasted space on the disk. It can never specify *more* sectors than physically exist, of course.

Logical Geometry

- Another way to get around the problem of complex internal geometry is to change the way the drive is addressed completely. Instead of using the logical geometry numbers directly, most modern drives can be accessed using ***logical block addressing* (*LBA*).** With this method a totally different form of logical "geometry" is used: the sectors are just given a numerical sequence starting with 0. Again, the drive just internally translates these sequential numbers into physical sector locations. So the drive above would have sectors numbered from 0 to 7,539,839. This is just yet another way of providing access to the same sectors. You can read more about LBA here.
- Today's drives are over 8.4 GB in size and have therefore run into an important hard disk capacity barrier: the 8.4 GB (7.8 GiB) capacity barrier. The largest logical parameters that can be used for accessing a standard IDE/ATA drive using normal Int 13h BIOS routines are 1,024 cylinders, 256 heads, and 63 sectors. Since the ATA standard only allows a maximum of 16 for the number of heads, BIOS translation is used to reduce the number of heads and increase the number of cylinders in the specification (see here for details on this). The practical result of all of this, is that the largest logical geometry numbers for IDE/ATA drives are 16,383 cylinders, 16 heads and 63 sectors. This yields a maximum capacity of 8.4 GB.

Logical Geometry

- Drives larger than 8.4 GB can no longer be accessed using regular BIOS routines, and require **extended Int 13h capabilities**. There is no way to even *represent* their full capacity using regular IDE/ATA geometry numbers. Therefore, these drives just specify 16,383 cylinders, 16 heads and 63 sectors to the BIOS for compatibility. Then, access to the drive is performed directly by the Int 13h extension routines, and the logical parameters are completely ignored. Here's how a modern drive, the 34.2 GB IBM Deskstar 34GXP (model DPTA-373420), looks:

Specification	Physical Geometry	Logical Geometry
Read/Write Heads	10	16
Cylinders (Tracks per Surface)	17,494	16,383
Sectors Per Track	272 to 452	63
Total Sectors	66,835,440	16,514,064

Error Correcting Code (ECC)

- The basis of all error detection and correction in hard disks is the inclusion of redundant information and special hardware or software to use it. Each sector of data on the hard disk contains 512 bytes, or 4,096 bits, of user data. In addition to these bits, an additional number of bits are added to each sector for the implementation of *error correcting code* or *ECC* (sometimes also called *error correction code* or *error correcting circuits*). These bits do not contain data; rather, they contain information about the data that can be used to correct any problems encountered trying to access the real data bits.
- There are several different types of error correcting codes that have been invented over the years, but the type commonly used on PCs is the ***Reed-Solomon* algorithm, named for researchers Irving Reed and Gustave** Solomon, who first discovered the general technique that the algorithm employs. Reed-Solomon codes are widely used for error detection and correction in various computing and communications media, including magnetic storage, optical storage, high-speed modems, and data transmission channels. They have been chosen because they are easier to decode than most other similar codes, can detect (and correct) large numbers of missing bits of data, and require the least number of extra ECC bits for a given number of data bits.

Read Error Severities and Error Management Logic

- The hard disk's controller employs a sequence of sophisticated techniques to manage errors that occur when reading data from the disk. In a way, the system is kind of like a troubleshooting flowchart. When a problem occurs, the simplest techniques are tried first, and if they don't work, the problem is escalated to a higher level. Every manufacturer uses different techniques, so this is just a rough example guideline of how a hard disk will approach error management:
- **ECC Error Detection:** The sector is read, and error detection is applied to check for any read errors. If there are no errors, the sector is passed on to the interface and the read is concluded successfully.
- **ECC Error Correction:** The controller will attempt to correct the error using the ECC codes read for the sector. The data can be corrected very quickly using these codes, normally "on the fly" with no delay. If this is the case, the data is fixed and the read considered successful. Most drive manufacturers consider this occurrence common enough that it is not even considered a "real" read error. An error corrected at this level can be considered "automatically corrected".
- **Automatic Retry:** The next step is usually to wait for the disk to spin around again, and retry the read. Sometimes the first error can be caused by a stray magnetic field, physical shock or other non-repeating problem, and the retry will work. If it doesn't, more retries may be done. Most controllers are programmed to retry the sector a certain number of times before giving up. An error corrected after a straight retry is often considered "recovered" or "corrected after retry".
- **Advanced Error Correction:** Many drives will, on subsequent retries after the first, invoke more advanced error correction algorithms that are slower and more complex than the regular correction protocols, but have an increased chance of success. These errors are "recovered after multiple reads" or "recovered after advanced correction".
- **Failure:** If the sector still cannot be read, the drive will signal a read error to the system. These are "real", unrecoverable read errors, the kind that result in a dreaded error message on the screen.

Read Error Severities and Error Management Logic

- Even before the matter of actually reading the data comes up, drives can have problems with locating the track where the data is. Such a problem is called a *seek error*. In the event of a seek error, a similar management program is instituted as that used for read errors. Normally a series of retries is performed, and if the seek still cannot be performed, an unrecoverable seek error is generated. This is considered a drive failure, since the data may still be present, but it is inaccessible.
- Every hard disk model has analysis done on it to determine the likelihood of these various errors. This is based on actual tests on the drive, on statistical analysis, and on the error history of prior models. Each drive is given a rating in terms of how often each error is likely to occur. Looking again at the Quantum Fireball TM, we see the following error rate specifications:

Error Severity	Worst-Case Frequency of Error (Number of Bits Read Between Occurrences)
Automatically Corrected	Not Specified
Recovered Read Errors	1 billion (1 Gb)
Recovered After Multiple Reads (Full Error Correction)	1 trillion (1,000 Gb)
Unrecoverable Read Errors	100 trillion (100,000 Gb)

Error Notification and Defect Mapping

- Many drives are smart enough to realize that if a sector can only be read after retries, the chances are good that something bad may be happening to that sector, and the next time it is read it might not be recoverable. For this reason, the drive will usually do something when it has to use retries to read a sector (but usually not when ECC will correct the problem on the fly). What the drive does depends on how it is designed.
- Modern drives support **SMART**, a **reliability feature that tries to predict drive failure based on technological "leading indicators**". Read errors, excessive numbers of retries, or problems seeking are very commonly included in the set of parameters used to signal impending hard drive doom. Activity of this sort that exceeds a safety threshold determined by the drive's designers may trigger a SMART warning, telling the user that the drive may be failing.
- Today's hard disks will also often take corrective action on their own if they detect that errors are occurring. The occasional difficulty reading a sector would typically be ignored as a random occurrence, but if multiple retries or other advanced error correction procedures were needed to read a sector, many drives would automatically mark the sector bad and relocate its contents to one of the drive's spare sectors. In doing so, the drive would avoid the possibility of whatever problem caused the trouble worsening, and thereby not allow the data to be read at all on the next attempt.