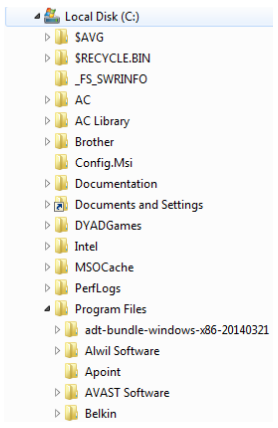
Files



Information saved on the computer is organized into files. There are files for both programs and data.

Since there may be thousands of files on a hard drive, they are organized into a hierarchy – space on the hard drive is divided into *directories* (also known as folders) which can be subdivided again and again. For example, on most Windows machines, there is a directory for Program Files, inside of which there are many directories for different programs. This structure of directories is called a directory tree.

Each drive on a computer is assigned a drive letter[1]. The hard drive is traditionally called C:\. The top-level directory of a drive is called the *root directory*.

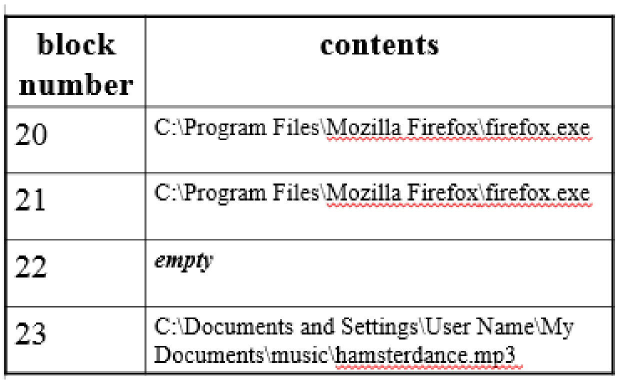
Two files in the same directory cannot have the same name. The real name of a file includes the list of directories you would have to go through to reach the file, starting at the root directory. This is called the *path*. For example C:\Documents\Downloads\Pictures\flowr.jpg says that the file flowr.jpg is in the directory Pictures, which is in the directory Download, which is in the directory Documents, which is on the C:\ drive.

[1] In some cases, a single physical drive may be divided up and treated like more than one, in which case each *logical drive* is assigned its own drive letter.

Index File and Hard Drive

Physically the hard drive is divided into sectors; a sector is the amount of data that can be read at once by a magnetic read-write head, so sectors are very small, too small to be useful for organizing the files we typically store now. Instead, we divide the hard drive into larger areas called *blocks* (also known as clusters). Each block has a number, the hard drive address.

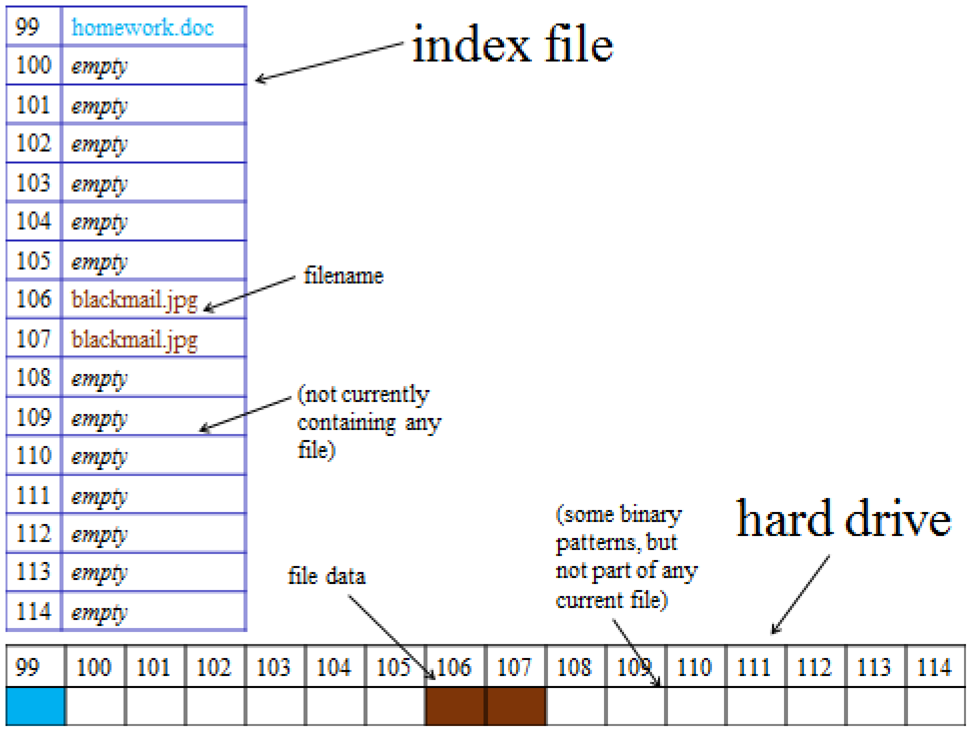
The *index file* is how the file system (the part of the operating system that maintains storage of files) keeps track of where the data for each file is located on a drive. The index file is a table listing all the blocks on the drive and their contents: either a filename (remember that a filename really includes the path, which is how the directory organization is maintained) or a special symbol to mark it as empty (not containing data from any current file).



block number	**contents**
20	C:\Program Files\Mozilla Firefox\firefox.exe
21	C:\Program Files\Mozilla Firefox\firefox.exe
22	***empty***
23	C:\Documents and Settings\User Name\My Documents\music\hamsterdance.mp3

Most files take up many blocks; an mp3 of a song generally takes up a few thousand. The index file entry for each block *only* contains the file name, not the data, and not how much of the block is filled with that data.

For the purposes of examples, however, we'll leave off the path and pretend that files are only a few blocks in size.



In this example, the index file (shown vertically) says that in block 99 there is the data for the file homework.doc and in blocks 106 and 107 is data from the file blackmail.jpg. The drive itself (shown horizontally) is shaded to indicate the actual data stored in those blocks. Since all blocks on the hard drive contain magnetic particles, the blocks marked empty do contain *some* patterns, just not patterns that are part of any current file. We will assume all blocks before 99 are full of file data.

Opening Files

Remember that to read from the HD, the HD controller needs to know the addresses to read. So, to open a file, the file system needs need to tell the HD controller which block numbers the file is in.

To find these block numbers, the file system will need to search through the index file to find all blocks marked with the chosen file name.

So, once we have the file name, search for that name in the index file, and find all blocks that match, then send the matching block numbers to the HD to be read.

Saving Files

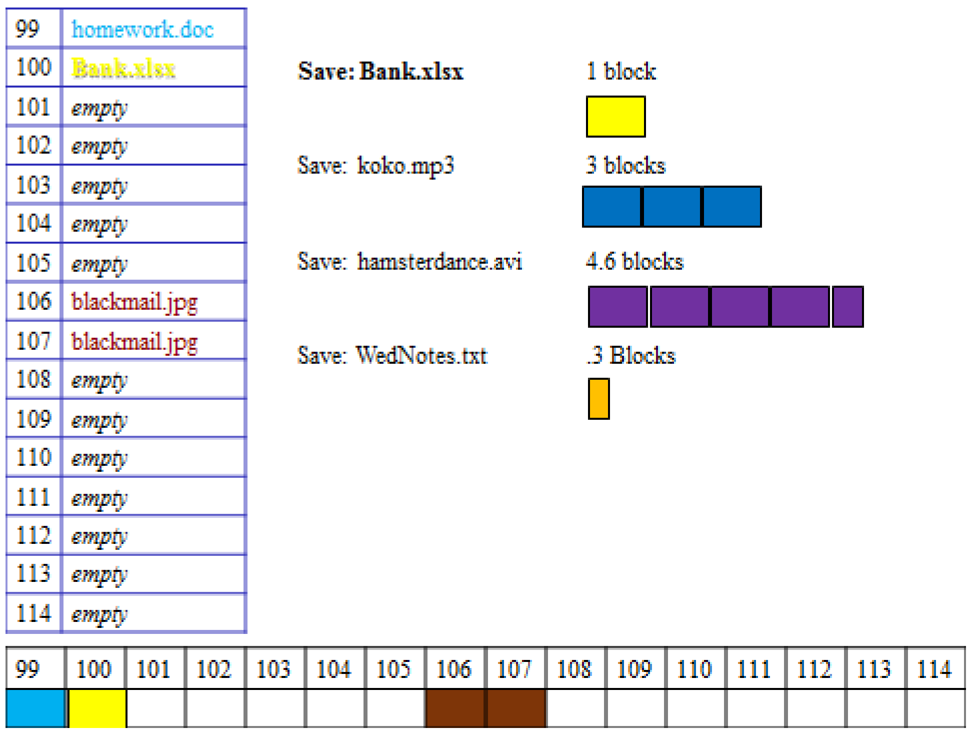
The index file tells us which blocks are not currently in use – the ones marked *empty*.

To save a file, the file system always chooses the first empty block to put it in.

So, suppose in the same situation as our previous example, we have the file Bank.xlsx which is 1 block in size; to save it we choose the first empty block, 100.

In the index file, we need to record that this block is where Bank.xlsx is saved, so we can find it again later, and so we don’t save something else over it, so we change that line in the index file from *empty* to the filename Bank.xlsx.

Now that we know what block to put it in, we send the chosen block number and the data for Bank.xlsx down the bus to the HD to be written.

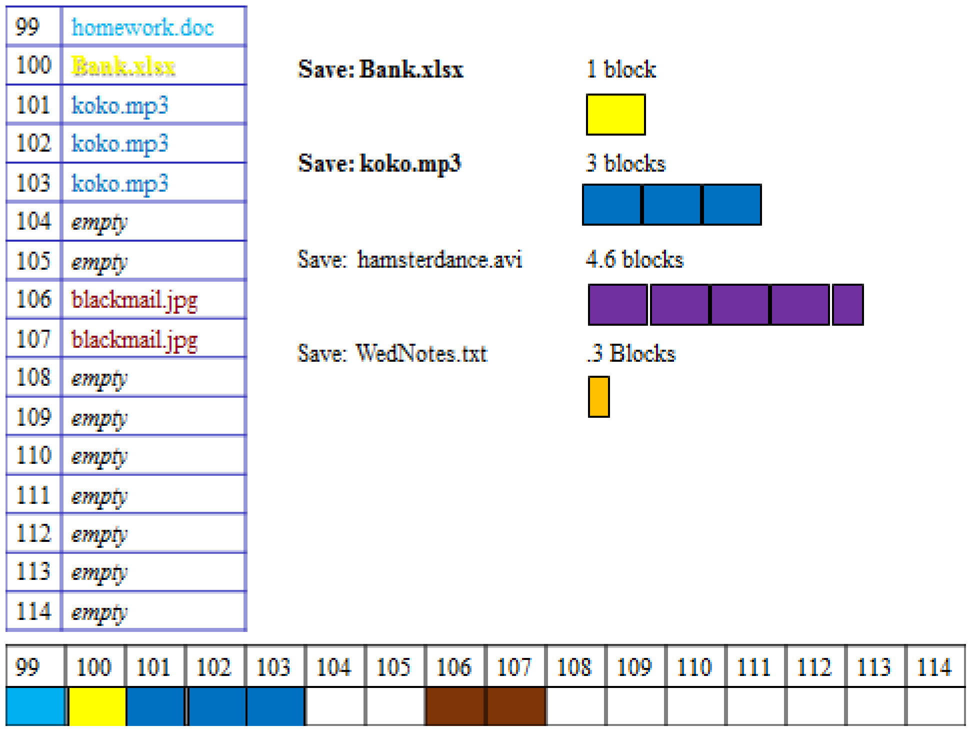


99	homework.doc
100	Bank.xlsx
101	*empty*
102	*empty*
103	*empty*
104	*empty*
105	*empty*
106	blackmail.jpg
107	blackmail.jpg
108	*empty*
109	*empty*
110	*empty*
111	*empty*
112	*empty*
113	*empty*
114	*empty*

Save: Bank.xlsx	1 block 🟨
Save: koko.mp3	3 blocks 🟦🟦🟦
Save: hamsterdance.avi	4.6 blocks 🟪🟪🟪🟪🟪
Save: WedNotes.txt	.3 Blocks 🟧

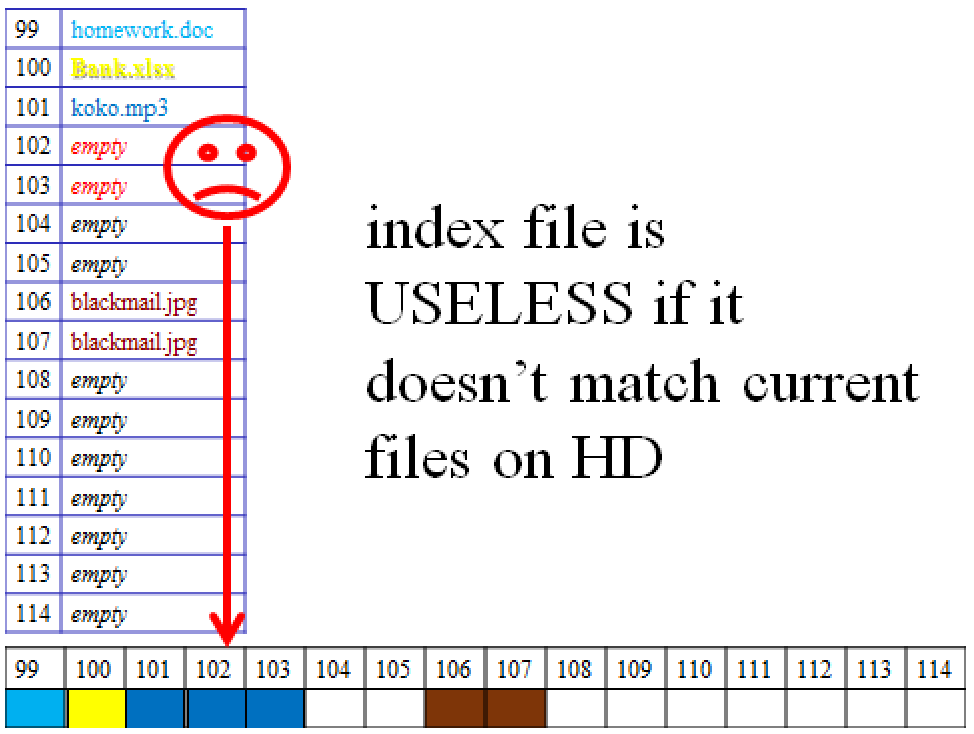
99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114
🟦	🟨						🟫	🟫							

Almost all files will be more than 1 block in size. So, we will repeat the process of choosing the first empty block and recording the file name in the index file, then sending the block numbers and data to the HD to be written, for all blocks of the file. So after saving koko.mp3, which is 3 blocks, we would have.



99	homework.doc
100	Bank.xlsx
101	koko.mp3
102	koko.mp3
103	koko.mp3
104	*empty*
105	*empty*
106	blackmail.jpg
107	blackmail.jpg
108	*empty*
109	*empty*
110	*empty*
111	*empty*
112	*empty*
113	*empty*
114	*empty*

If we only marked block 101 with “koko.mp3” in the index file, then the next time we tried to open it, we would only get 1/3 of the file, and the next file we save would overwrite the rest of the file! The file system is not magically all-knowing; if the block still says *empty* then the computer will believe the block is empty. The computer only knows what the index file tells it about the HD, so the index file must record *all* the blocks of all current files.



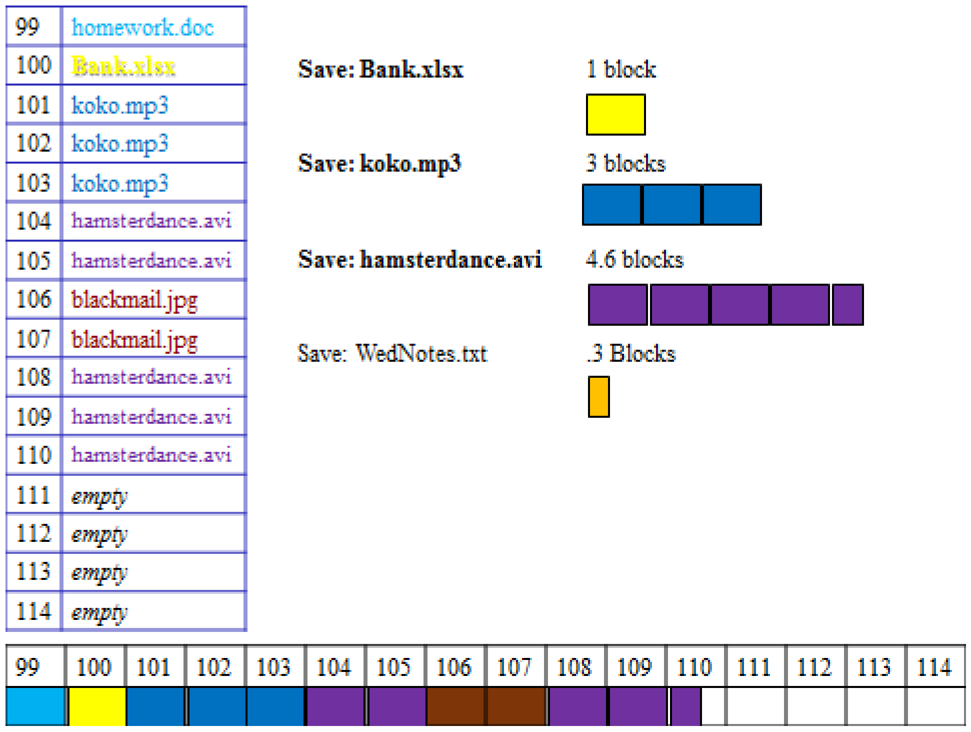
99	homework.doc
100	Bank.xlsx
101	koko.mp3
102	*empty*
103	*empty*
104	*empty*
105	*empty*
106	blackmail.jpg
107	blackmail.jpg
108	*empty*
109	*empty*
110	*empty*
111	*empty*
112	*empty*
113	*empty*
114	*empty*

index file is USELESS if it doesn't match current files on HD

99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114
🟦	🟨	🟦	🟦	🟦			🟫	🟫							

Remember that the computer always chooses the *first* empty block. This means that sometimes a file will occupy blocks that are not contiguous (next to each other). This is called *fragmentation*. For instance when we save hamsterdance.avi, which is 4.6 blocks, it ends up in 104, 105, 108, 109, 110.

Note that we do this even though there would have been enough space to put all the blocks of hamsterdance.avi together. Either way, it would be divided into blocks, but if the blocks are not next to each other, it is fragmented.

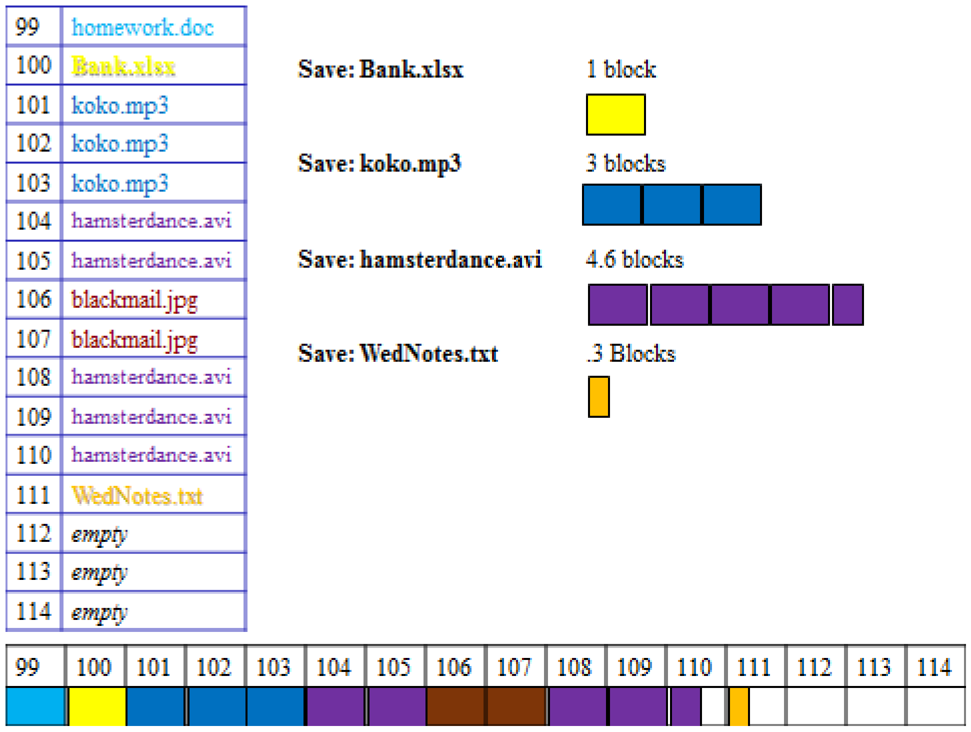


99	homework.doc
100	Bank.xlsx
101	koko.mp3
102	koko.mp3
103	koko.mp3
104	hamsterdance.avi
105	hamsterdance.avi
106	blackmail.jpg
107	blackmail.jpg
108	hamsterdance.avi
109	hamsterdance.avi
110	hamsterdance.avi
111	*empty*
112	*empty*
113	*empty*
114	*empty*

Note that there is nothing in 110 in the index file to indicate that only .6 of that block is filled with data. Each block in the index file is either marked *empty* or contains a file name.

The computer will not scan the data on all blocks of the HD looking for empty space. If it did, saving each file could take several minutes. The computer also does not magically know that some blocks are only partly full. It depends *only* on what the index file says.

So, when we try to save WedNotes.txt, which is only .3 of a block, the computer finds the first block marked empty at 111, and uses this, even though technically the data would fit at the end of block 110.



99	homework.doc
100	Bank.xlsx
101	koko.mp3
102	koko.mp3
103	koko.mp3
104	hamsterdance.avi
105	hamsterdance.avi
106	blackmail.jpg
107	blackmail.jpg
108	hamsterdance.avi
109	hamsterdance.avi
110	hamsterdance.avi
111	WedNotes.txt
112	*empty*
113	*empty*
114	*empty*

The wasted space at the end of a block (as in 110 and 111 here) is called slack space.

Deleting Files

Since accidentally deleting the wrong file is a common problem, most operating systems put an intermediate step in the deletion process. When we first tell the computer to delete a file, the computer instead just moves it to a special directory called something like "Recycling Bin." This does not remove the data or remove the file from the index file or even change which block the file is in on the hard drive. It simply changes the path. Files in the recycle bin can be moved back to their original directory until we "empty" the recycle bin and the files are actually deleted. When we talk about deletion in this course, we always mean this actual deletion, not this intermediate move to the recycle bin.

When we delete a file, our goal is to be sure that it can no longer be found when looking at the directory tree structure, and that we can use that space for other files. So we need to make sure that the blocks for that file are marked *empty* in the index file.

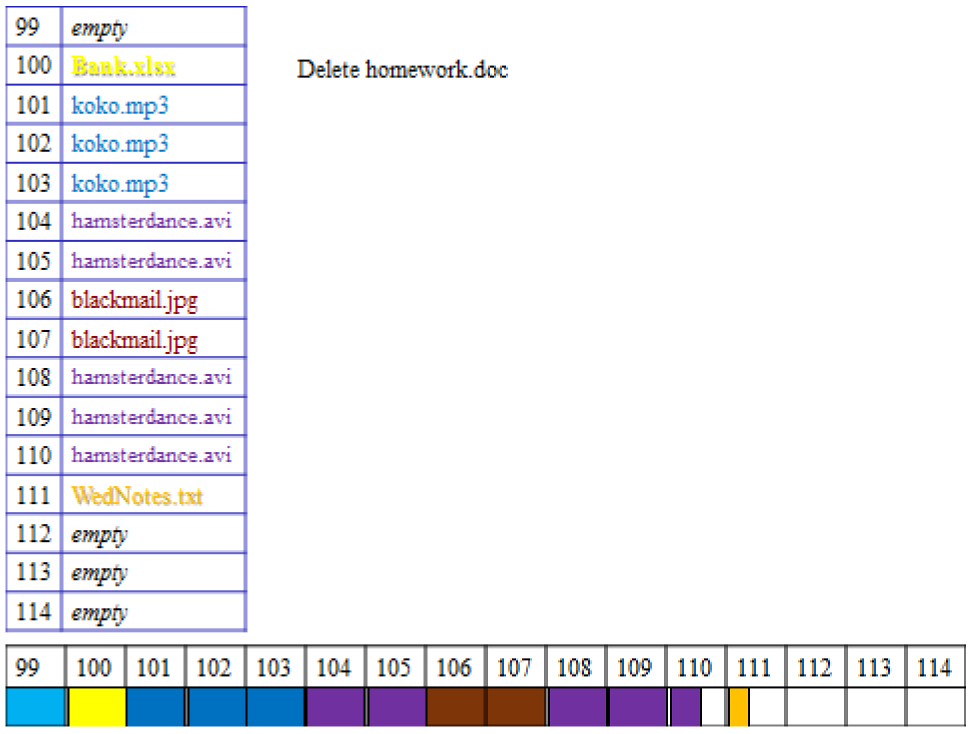
So, if we are deleting the file homework.doc, we need to find all blocks with that filename in the index file, and make sure they are changed to be marked *empty*. In this case there was only one block.

Since the file's former blocks on the hard drive are going to contain *some* patterns until we write a new file there, it costs us nothing to leave the file's patterns in place until then, and it would cost time to change them, so updating the index file is the only thing that happens when we delete. The file's data is still there, but no longer accessible. (Remember, this is totally separate from the file previously being sent to the recycle bin, which only changes the path to the file, but leaves it in the index file!)

A *recovery program* can be used to get back deleted files, if the data is still there and the blocks have not been written over by new files[2]. The recovery program bypasses the index file and looks at the contents of all the blocks on the hard drive and makes up its own list of what is in them. Any data still there from deleted files can then be recovered. Usually, these programs are used to get back important data that was deleted accidentally, but they may be used by agents of the law to investigate files that criminals have tried to get rid of, or by criminals to try to extract data (e.g. passwords) from computers that have been discarded.

[2] Although, even if the blocks are overwritten, suppose one is overwritten by the last block of a new file, which has some slack space. Then a tiny bit of the data from the file could still be there at the end of the block even though the block is "overwritten."

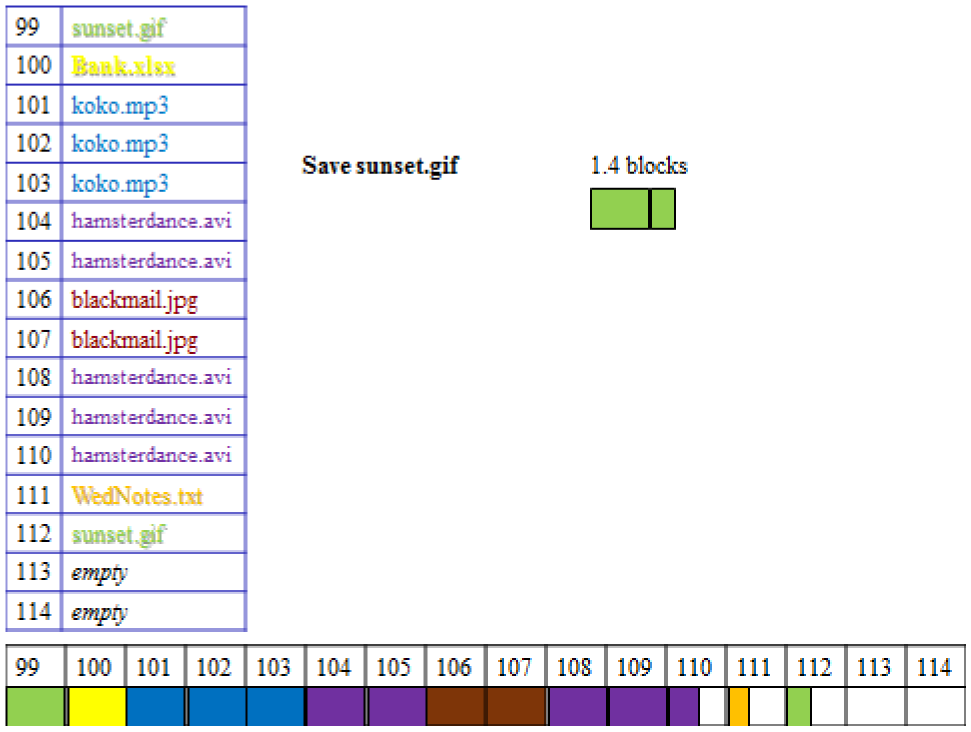
To avoid deleted data being recovered, instead of deleting normally, we should use a *secure delete* or “shredder” program, which first overwrites all the blocks of a file, and then does a normal delete.



99	*empty*
100	Bank.xlsx
101	koko.mp3
102	koko.mp3
103	koko.mp3
104	hamsterdance.avi
105	hamsterdance.avi
106	blackmail.jpg
107	blackmail.jpg
108	hamsterdance.avi
109	hamsterdance.avi
110	hamsterdance.avi
111	WedNotes.txt
112	*empty*
113	*empty*
114	*empty*

99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114
🟦	🟨	🟦	🟦	🟦	🟪	🟪	🟫	🟫	🟪	🟪	🟪	🟧			

When we next save a file, block 99 is the first empty block, so it will get used, and at that point the data from homework.doc is replaced, and can no longer be recovered.



99	sunset.gif
100	Bank.xlsx
101	koko.mp3
102	koko.mp3
103	koko.mp3
104	hamsterdance.avi
105	hamsterdance.avi
106	blackmail.jpg
107	blackmail.jpg
108	hamsterdance.avi
109	hamsterdance.avi
110	hamsterdance.avi
111	WedNotes.txt
112	sunset.gif
113	*empty*
114	*empty*

Deleting leaves empty areas on the HD, which can result in later saved files being fragmented, as has happened to sunset.gif here.

Slack Space

Since few files' size are an exact multiple of the size of a block, most files have a little bit of extra space at the end of the last block they occupy. This is called *slack space*. The slack space is space that cannot be used to store data for any other file, so lots of slack space means we are not making the best use of our space.

Note that the amount of slack space is determined by the size of the blocks: we can never have more than one block worth of slack space per file. So, if we make the block size smaller, that limits the amount of slack space. However, if we divide the hard drive into smaller blocks, then there will be more blocks to keep track of – the index file will be longer – so every time we open or save or delete a file, it will take longer because we have more blocks to deal with.

By contrast, if we use larger blocks, the index file will be shorter, which makes dealing with files faster, but this results in more slack space.

For home computers, hard drives are set up with fairly small block sizes, since we tend to have many small files. This means that slack space really wastes very little space.

Fragmentation

Because we always choose the first empty block when saving, files can end up fragmented when we save a larger file after deleting several smaller ones. A file is said to be fragmented when its blocks are not contiguous (side-by-side).

If the blocks of a file are widely separated on the drive, then there will be a lot of latency in reading and writing the file – lots of time spent waiting for the read-write heads to get from spot to spot on the hard drive. This means that opening and saving the file takes longer.

As we delete and save files on our computers, the files tend to become more and more fragmented, and over time the computer gets slower and slower.

The only way to avoid fragmentation entirely would be to change the way we save files: instead of saving in the first empty spot, we could search until we find enough empty blocks in a row. But for a file that needs several thousand blocks, this may take a very long time, and enough contiguous blocks in a row might not exist. If we did this, saving files would simply take too long.

So, instead of avoiding fragmentation, we periodically defragment our drives. Defragmentation swaps around blocks of files until each file's block are contiguous[3]. To do this to all of a large drive full of fragmented files may take a long time. Most current computers do a little bit of defragmentation automatically on a regular basis so that it never gets too bad, while other types of computer require users to periodically choose to run a defragmentation process.

[3] Note that this makes security and privacy a bit more difficult. If the blocks of a file storing sensitive data are moved around during defragmentation, we could end up with multiple copies of that data on the drive. If we later securely delete the file from its new position, that won't get rid of the copies left behind in other blocks. For this reason, it is better to wipe or destroy the entire drive if it has sensitive information.