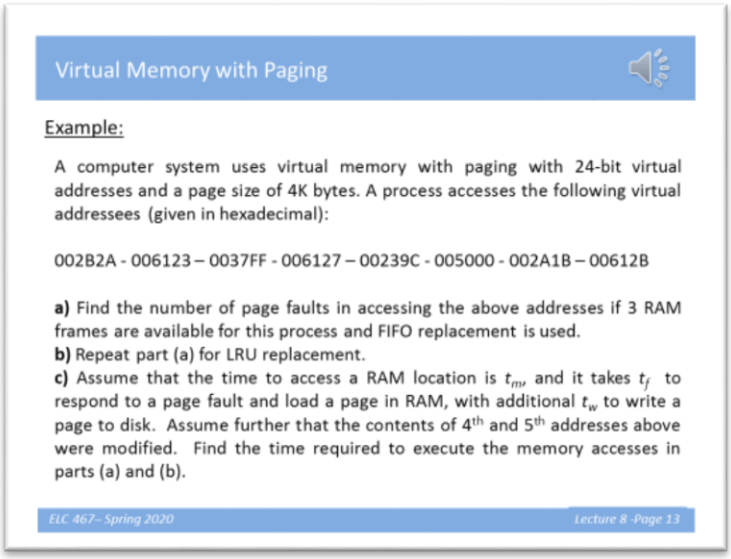
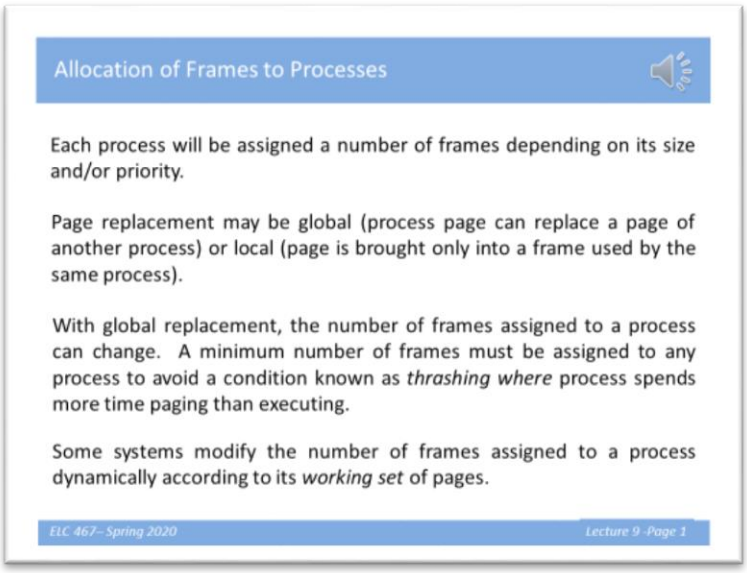


Slide 1:

Note that if a page is modified while in memory, it has to be rewritten to disk when swapped out. This is not necessary if data in page was read only and not modified. System keeps for each page in the page table a bit which is set when its contents are modified, to indicate that it should be written back to disk when removed from RAM.



Slide 3:

One final point in virtual memory is how it will be combined with multitasking. Note that each process will access memory in the whole virtual address space. Thus for example, process A will have data in its virtual page 0, while process B will have different data in its virtual page 0. Processes then share physical memory, with each process owning a number of frames into which its virtual pages will be swapped.

Slide 4:

Thus with LRU in the global replacement case, a new page will replace the least recently used page of any process. In the local replacement case it will replace the least recently used page belonging to the

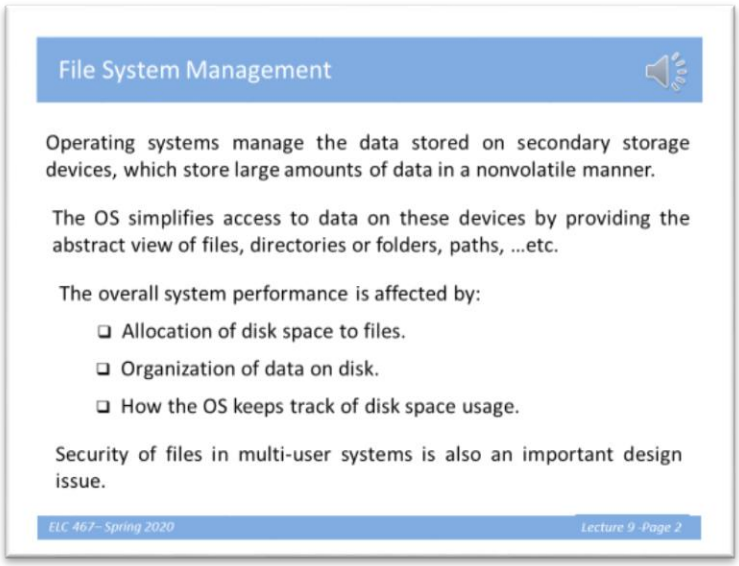
same process only. It is obvious that in the local case the number of frames owned by the process will not change, but this is not guaranteed in the global case.

Slide 5:

The system will specify a minimum number of frames that any active process must keep. With too small number of frames, process may swap out a page, and then access it again quickly. This results in too many page faults. Since each page fault is associated with a relatively long delay, most of process time will be spent in handling page faults.

Slide 6:

For example, if process experiences a high rate of page faults, system increases the number of frames assigned to it. On the other hand, if process has many pages in RAM which are not accessed, system reduces its number of frames.



Slide 7:

We now study a new class of functions related to file management. Secondary storage devices include magnetic disks, semiconductor media, and optical devices.

Slide 8:

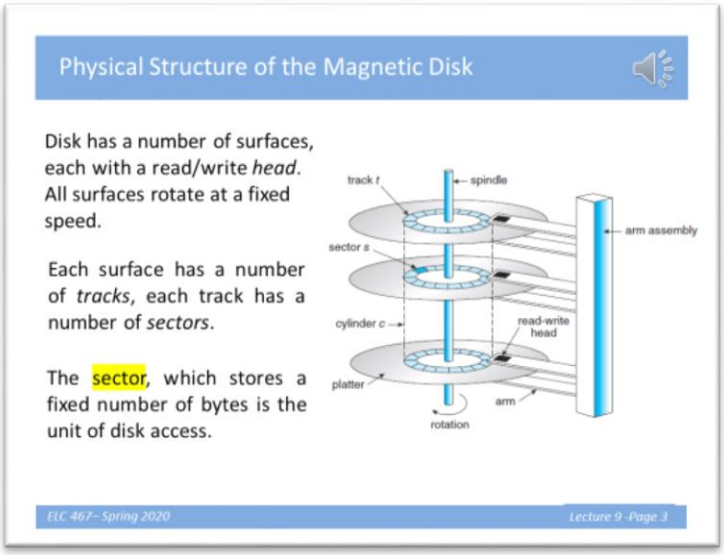
As we said before, what exists physically is a huge number of bits stored on a device. The view that these bits are in files that have names, paths, and access rights is given to the user by the operating system.

Slide 9:

Mass storage devices can store huge amounts of data at relatively low cost and non-volatile manner (when power is switched off, data is not erased). Their main disadvantage is the low access speed compared to main memory. The main design factor in file management functions is to prevent this low speed from slowing the system as a whole.

Slide 10:

Security here concerns allowing the user to define how his data can be accessed by other users.



Slide 11:

Magnetic disks are still an economical choice for large storage capacities. We review here the internal structure of a magnetic disk to understand the sources of access delays.

Slide 12:

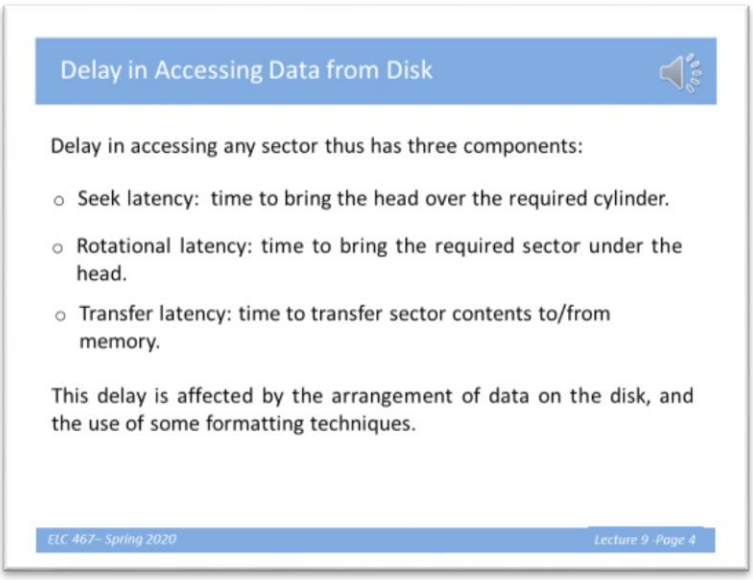
Arm assembly moves all heads together. A position of arm allows heads to read/write from tracks having the same radius, called a cylinder. Moving arm from cylinder to cylinder involves a mechanical movement, which is always slow compared to other delays in system.

Slide 13:

In current technology, outer cylinders can have more sectors and hold more data than inner tracks.

Slide 14:

Only sectors can have addresses. Unlike RAM, individual bytes inside sector cannot be addressed. We always read or write an entire sector (storing for example 512 bytes. Address of sector include three coordinates: its surface, track, and its number within track.

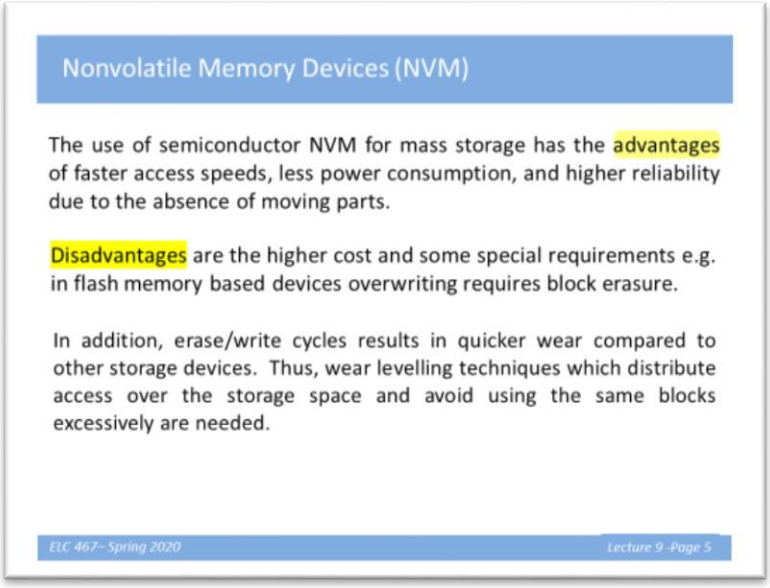


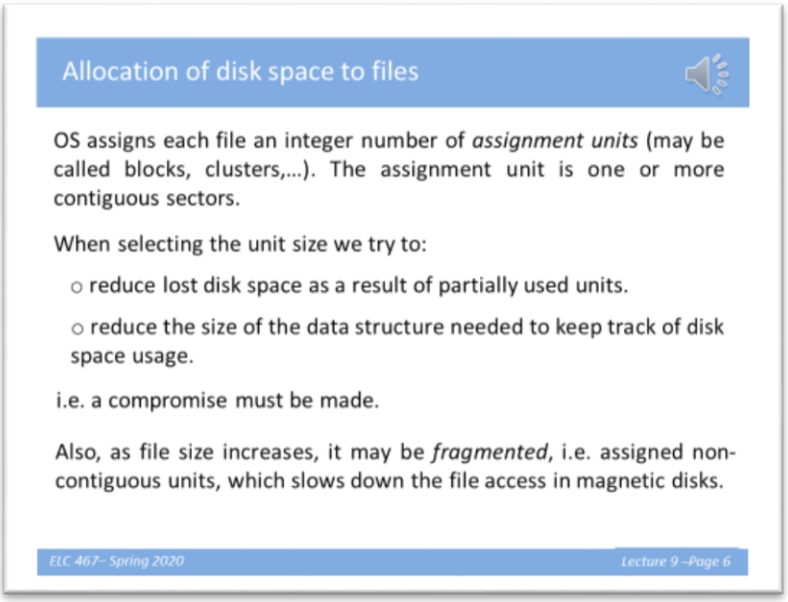
Slide 15:

Seek and rotational latencies are long mechanical delays. They are random, depending on head location when access starts. Data to/from head are stored in buffer, then transferred to/from RAM using DMA. This last component of delay should be smaller, unless bus has high data traffic that slows down DMA.

Slide 16:

For example, if the data of a file is stored on sectors in the same track, seek and rotational latency will occur when moving to first sector in file. Then, rest of file is accessed without further seek or rotational movements. However, if file parts are scattered over different tracks, access will be slow as a result of repeated seeks and rotations.





Slide 18:

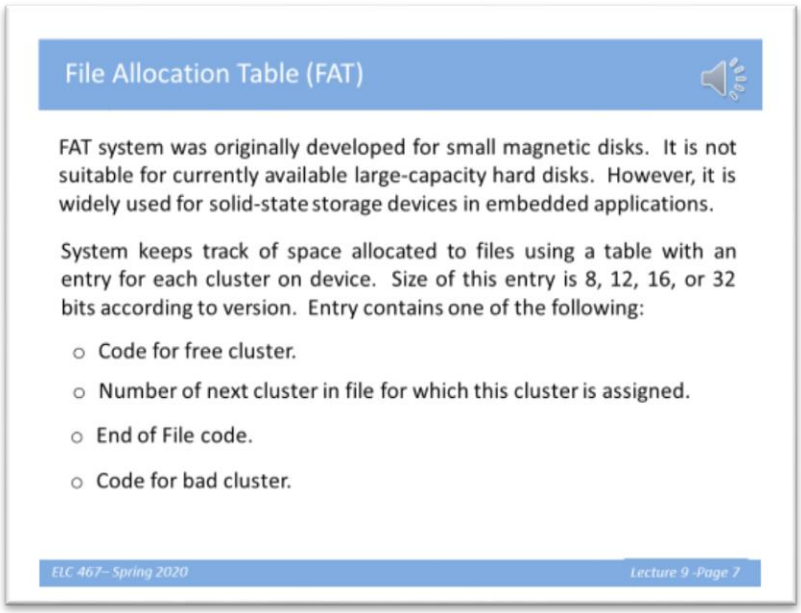
The number of sectors in large capacity devices will be huge. Keeping track of these sectors individually will require large data structures, requiring large storage area and long processing times. Thus, system groups sectors into larger units to reduce the size of these data structures. UNIX based system call these units blocks, while DOS and Windows call them clusters. File will be assigned an integer number of units, as system will not handle fractions of units.

Slide 19:

Last unit assigned to file will typically be not full (unless file size is exactly an integer multiple of unit size). However, space not used by file cannot be used by other files and is hence wasted. Note that unit size need not be the same in all system parts (e.g. different disks or different partitions of the same disk).

Slide 20:

As we said before, fragmenting file into parts on different cylinders will cause multiple seek and rotational delays. However, this may be unavoidable. For example, if user opens an existing file and add data to it, new data would ideally be stored on the same track or cylinder of older file parts. System may find that this cylinder is already filled by other files and will thus be forced to fragment the file.



Slide 21:

We consider different examples of how system keeps track of data stored on disks. Again, these will be variations of the map and linked list ideas mentioned before in memory management. We start by the FAT as an example of a technique suitable for small storage volumes. FAT was used in DOS and Windows for small magnetic disks.

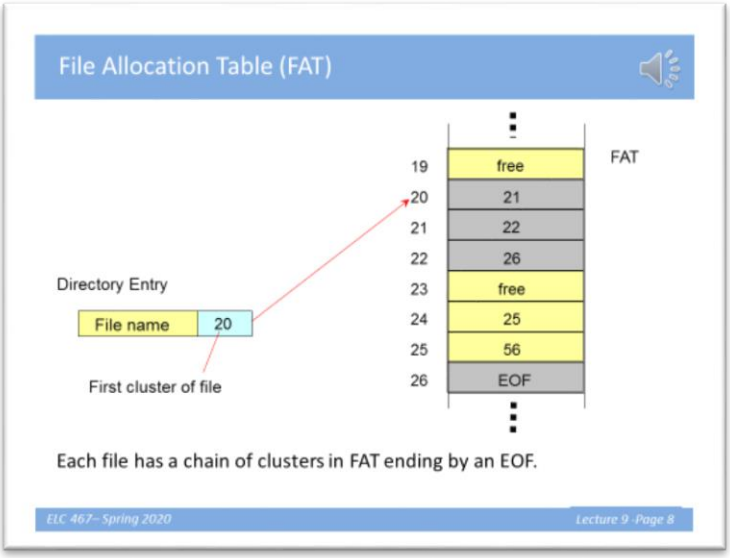
Slide 22:

The table itself is stored in the start of the storage volume. For faster processing it is read in RAM when disk is accessed. If it is modified while in RAM, it should be updated on disk. Note that assignment unit here is called a cluster. FAT 32 for example is the version where each entry (cell) in

table has a size of 32 bits. Since this will hold other cluster numbers as we will see, increasing the number of bits allows larger volume sizes.

Slide 23:

Free cluster is indicated by 0. A nonzero entry for cluster indicates that this cluster is not free. Similar to the idea of a linked list, entry points to the next cluster of file or indicates that this cluster is the end of file. During formatting, system tests clusters. If it detects that a cluster is defective, it marks it with a special code to avoid using it for files.

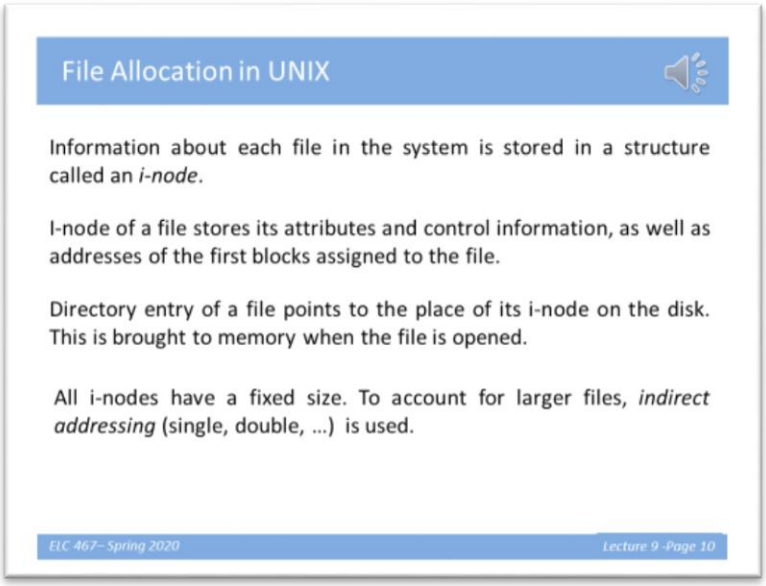


Slide 24:

Thus, there will be a chain for each file in FAT. For example, 20-21-22-26 in figure are clusters of some file (this file is fragmented). What is clearly missing is the start of file, is it 20 or maybe some other cluster is pointing to 20.

Slide 25:

Start of file is known from the directory information. This is a listing of files in current directory (folder) with information about each file (name, size, time of last access,…etc.) and the number of its first cluster.



Slide 28:

FAT is intended for small volumes. For example, system will bring the information of the whole disk to access any one file. This is not suitable for large disks. As another example technique for relatively larger disks we consider the method used by UNIX. Note that what we describe here is the original UNIX file system, which was modified for example in new versions of LINUX to handle still larger storage volumes.

Slide 29:

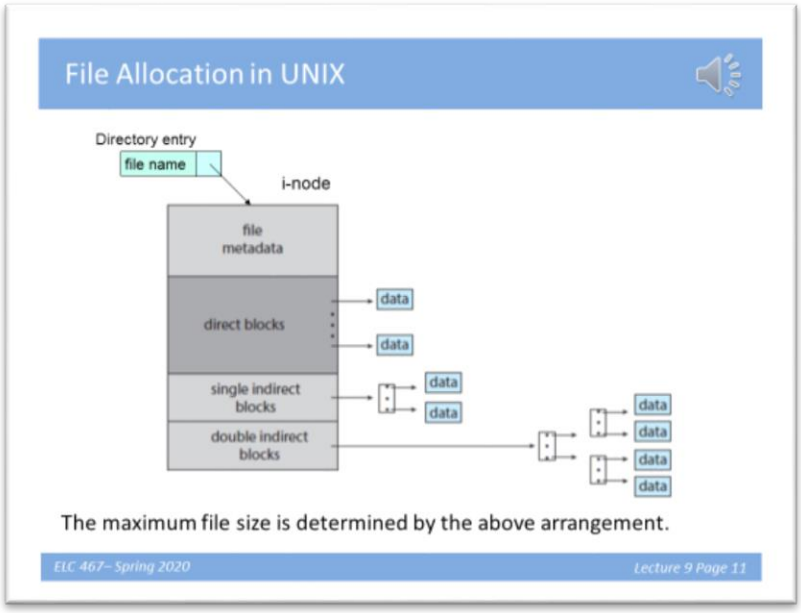
Information of each file is stored separately. "I" stands for index, and it is called a node as it will be part of a tree.

Slide 30:

Assignment units in UNIX are called blocks. Attributes and control information include size, time of last modification, …. etc., as well as access rights as UNIX is a multiuser system.

Slide 31:

As files vary in size, their contents will have more or less blocks. The numbers (addresses) of these blocks will be listed in the i-node, and this will thus need to be of varying size. Instead, UNIX developers chose to give the i-node a fixed size that can hold a specific number of block addresses. If file contains more data blocks, i-node points to extension blocks that holds the addresses of these additional data blocks.

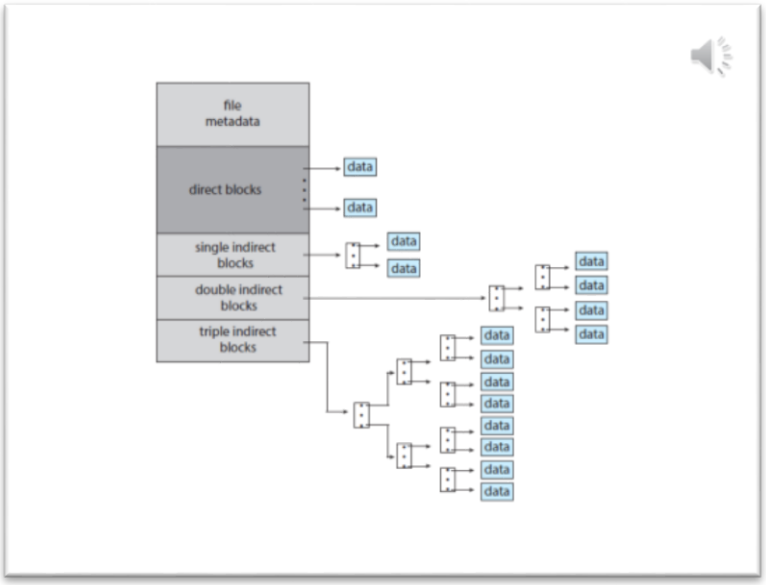


Slide 32:

A number of blocks is listed directly in i-node. If this is not enough, i-node points to a single indirect block that holds more addresses. If file is still much larger, i-node points to a block which points to blocks, each holding more addresses (this is called double indirect).

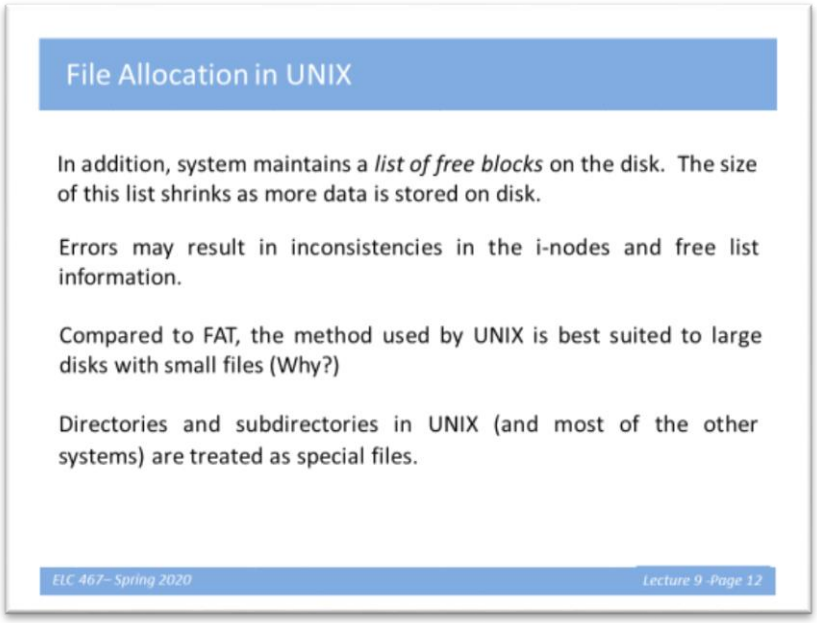
Slide 33:

FAT itself does not put a limit on the file size (all the disk space can be used to store one file). However, the above arrangement of UNIX will limit the file size by the maximum number of block addresses that can be stored in the above tree.



Slide 37:

With more large disks and files, newer versions may also use triple indirect addressing. Also, lists of block addresses need not be explicitly enumerated but can be compressed. For example, instead of listing 100 addresses, system stores that file has 100 blocks starting from block 2000.



Slide 38:

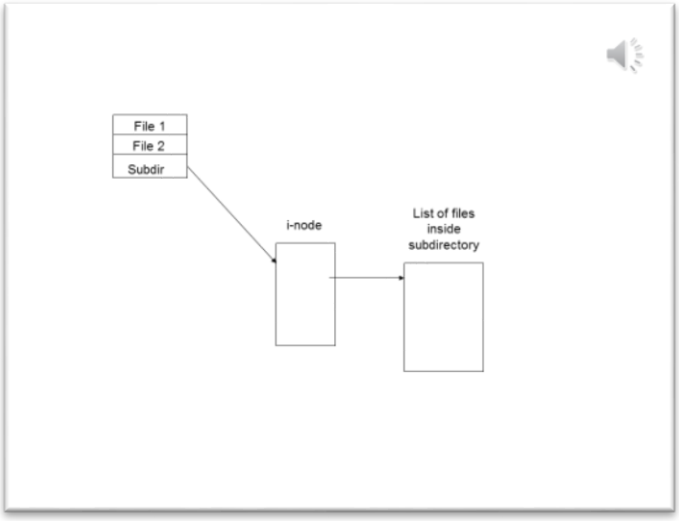
Free clusters can be found directly in FAT. But with the method of UNIX described here, it is not easy to find free blocks in disk, e.g. when a new file need to be created. System keeps this information in a separate list. This list is initially large when disk is empty. But gets smaller as more data is stored on disk.

Slide 39:

The number of each block should exist once, either in free list or in the i-node tree of some file. Otherwise, file system will have an error. Error occurs if system make modifications in i-node while in memory but fails to update them on disk.. This will not occur if files are closed and system is shut down properly, but may occur for example with sudden power failures.

Slide 40:

It is suited to large disks as it does not store the information of the whole volume in one table as FAT does. It is not suited for large files as these will require multiple disk accesses to read indirect addressing blocks resulting in slower access. Remember that the basic system we described was developed at an earlier stage of technology.



Slide 42:

Subdirectories have i-nodes similar to files. Attributes in i-node indicate that it is not a normal file, but a subdirectory. Its i-node will also point to data blocks. Whereas for normal files these contain the file data, for subdirectory these contain list of files inside the subdirectory.