File system structure and naming

Sam King

Administrative

- MP8 out – due 12/8
- HW2 out today – due 12/6

- No class on 12/6
- Final review on 12/8
- I will not have office hours 12/6 – 12/10

Device driver abstractions

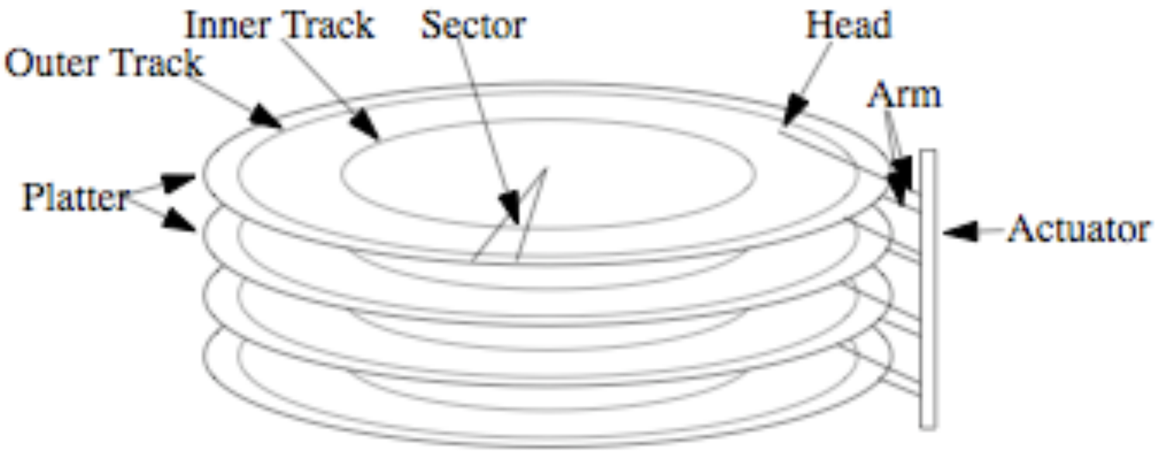
- Driver abstraction makes things "better"
  - Threads: don't worry about sharing CPU
  - Address spaces: don't worry about sharing phys mem
  - Device driver: don't worry about differences and limitations of devices

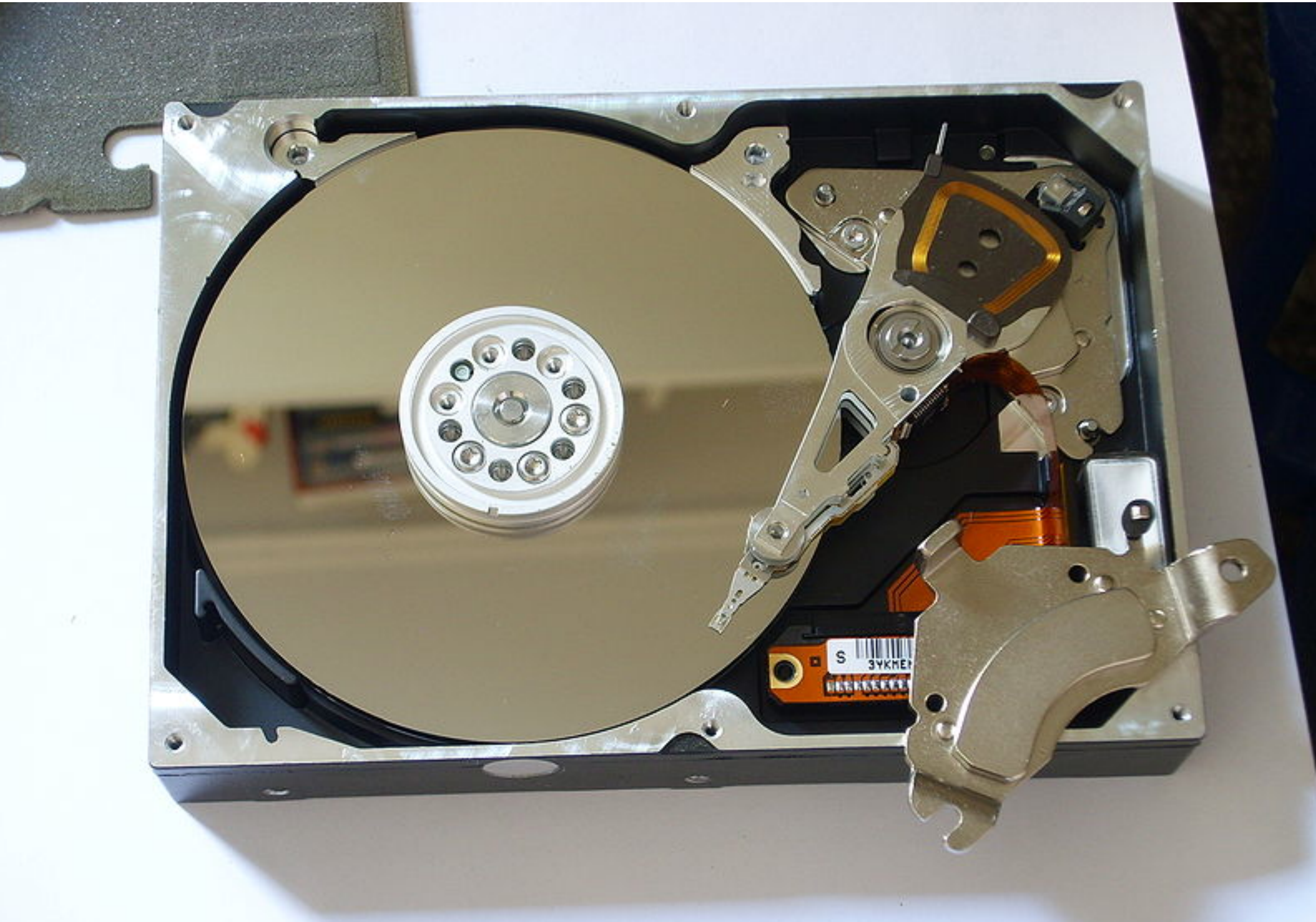
Leopard move bug

- Example of why file systems are important…

Disk geometry and access

- Disk make of a stack of spinning **platters**
- Top and bottom, concentric circles of data (**tracks**), tracks at same radial distance are called **cylinders**
- Each track has a number of **sectors**





From Wikipedia: http://en.wikipedia.org/wiki/File:Apertura_hard_disk_05.jpg

Accessing a disk

- Queuing time (wait for it to be free) 0-inf
- Position disk arm and head (seek and rotate) 0-12 ms
- Access disk data: 50-70 MB/s
  - Increased disk rotations speeds generally faster
  - Faster access times on outside tracks

Optimizing disk performance

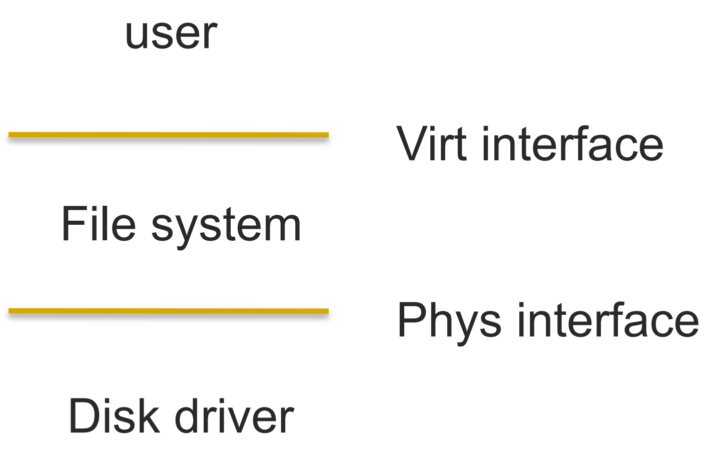
- Disk is slow!
- Best option: caching to eliminate I/O
- When you go do disk, keep positioning time low
- If you do have to re-position, try to amortize

Solid state disks

- Remove mechanical components
- Advantages
  - Fast! Apparently can saturate SATA (1.5 Gb / s)
  - No seek time, waiting for spinning
  - More reliable (maybe)
- Disadvantages
  - More expensive
  - Limited write cycles
    - Write leveling helps
- MacBook Pro with 256 GB SSD
- Windows 7– fast boot using USB flash drive
- Hybrid SSD / traditional disk

File systems

- A file system is an OS abstraction to make the disk easier to use
- Physical reality
  - Slow access to disk blocks
- Illusion provided
  - Fast access to byte oriented files, indexed using symbolic (English) names



user

Virt interface

File system

Phys interface

Disk driver

The file illusion

- How to map file space onto disk space?
    - File system structure on disk; disk allocation
    - Very similar to memory management

- How to use symbolic names instead of disk sectors?
    - Naming; directories
    - Not similar to memory management since virtual and physical both use same name (i.e. address)
    - Not going to discuss much in this class

File system structure

- Overall question: how to organize files on disk
  - What data structure is the right one to use?
  - Side note: many things in OS (and CS in general) boil down to data structures and algorithms
    - E.g., VM was about choosing a data structure/algorithm for translation
    - Algorithms and OS important classes

File system structure

- Need an internal structure that describes the object
  - Called a “file header” in this class
    - Inode in Unix
  - File header also contains miscellaneous information about the file, e.g., file size, modification date, permissions
    - Also called file meta-data
- Many ways to organize data on disk

File system usage patterns

- 80% of file accesses are reads
- Most programs that access a file sequentially access the entire file
  - Alternative is random access
    - Examples?
- Most files are small; most bytes on disk are from large files

Contiguous allocation

- Store a file in one contiguous segment on disk (sometimes called an extent)
- User must declare the size of the file in advance
  - File system will pre-allocate this memory on disk
  - What do you do if the file grows larger?
- File header is simple: starting block num & size
- Similar to base & bounds for mem mngt

Contiguous allocation

- Pros
  - Fast sequential access
    - No seeks between blocks
  - Easy random access
    - Easy and fast to calculate any block in file
- Cons
  - External fragmentation
  - Hard to grow files
  - Wastes space

Linked list

- Each block contains a pointer to the next block of file (along with data)
  - Used by Alto (first personal computer)
- File header contains pointer to first disk block
- Pros
  - Grow easily (i.e. append) files
  - No external fragmentation (pick any free block)
- Cons
  - Sequential access quite slow
    - Lots of seeks between blocks
  - Random access is really slow

Indexed files

- User (or system) declares max # of blocks in a file; system allocates a file header with an array of pointers big enough to point to that number of blocks
- Extra level of indirection, like a page table

File block #	Disk block #
0	18
1	50
2	8
3	15

Indexed files

```
#define FS_BLOCKSIZE 1024
#define FS_MAXFILEBLOCKS 253
#define FS_MAXUSERNAME 7
typedef struct {
    char owner[FS_MAXUSERNAME + 1];
    int size; // size of the file in bytes
    int blocks[FS_MAXFILEBLOCKS]; // array of file blocks
} fs_inode; (note sizeof(fs_inode) == FS_BLOCKSIZE)

Disk_readblock(int diskBlockNo, void *buf);
Disk_lookupinode(char *fileName, fs_inode *inode);

Write code for reading a file block for a given file name
Fs_readblock(char *fileName, int fileBlockNo, void *buf)
```

Indexed files

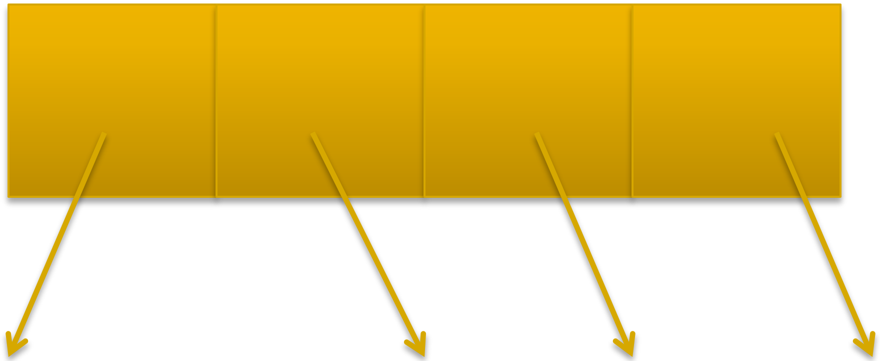
- Pros
  - Can easily grow (up to # of blocks allocated in header)
  - Easy random access loc. Calculation
- Cons
  - Lots of seeks for sequential access
    - How can you make this faster without pre-allocation?
  - Can’t easily grow beyond # block allocation

Large files

- How to deal with large files?
  - Could you assume file might get really large, allocate lots of space in the file header?
  - Could you use a larger block size, eg 4MB?
- Solution: more sophisticated data structure for the file header

Multi-level indexed files

Indexed files are like a shallow tree



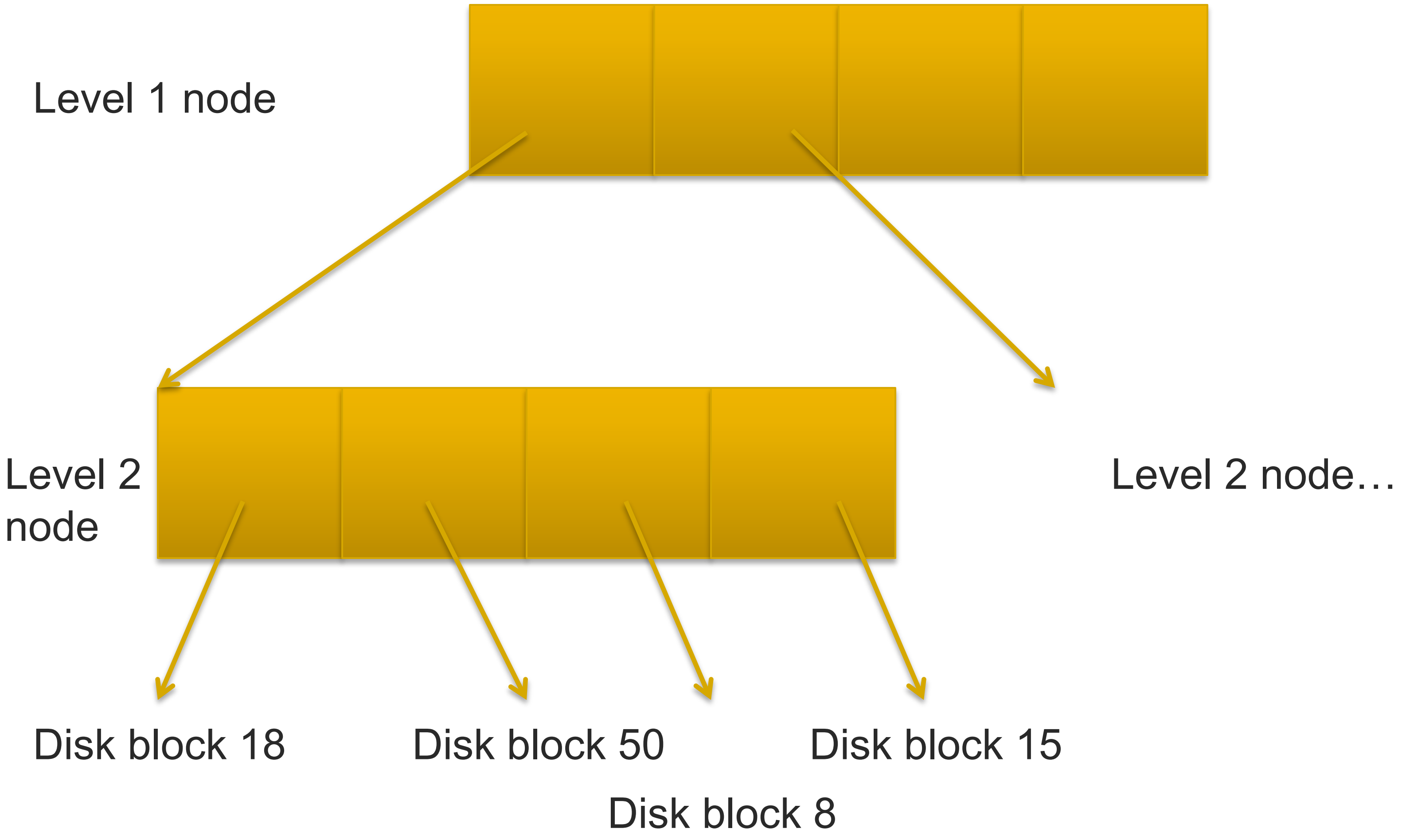
Disk block 18

Disk block 50

Disk block 15

Disk block 8

Multi-level indexed files



Level 1 node

Level 2 node

Level 2 node…

Disk block 18

Disk block 50

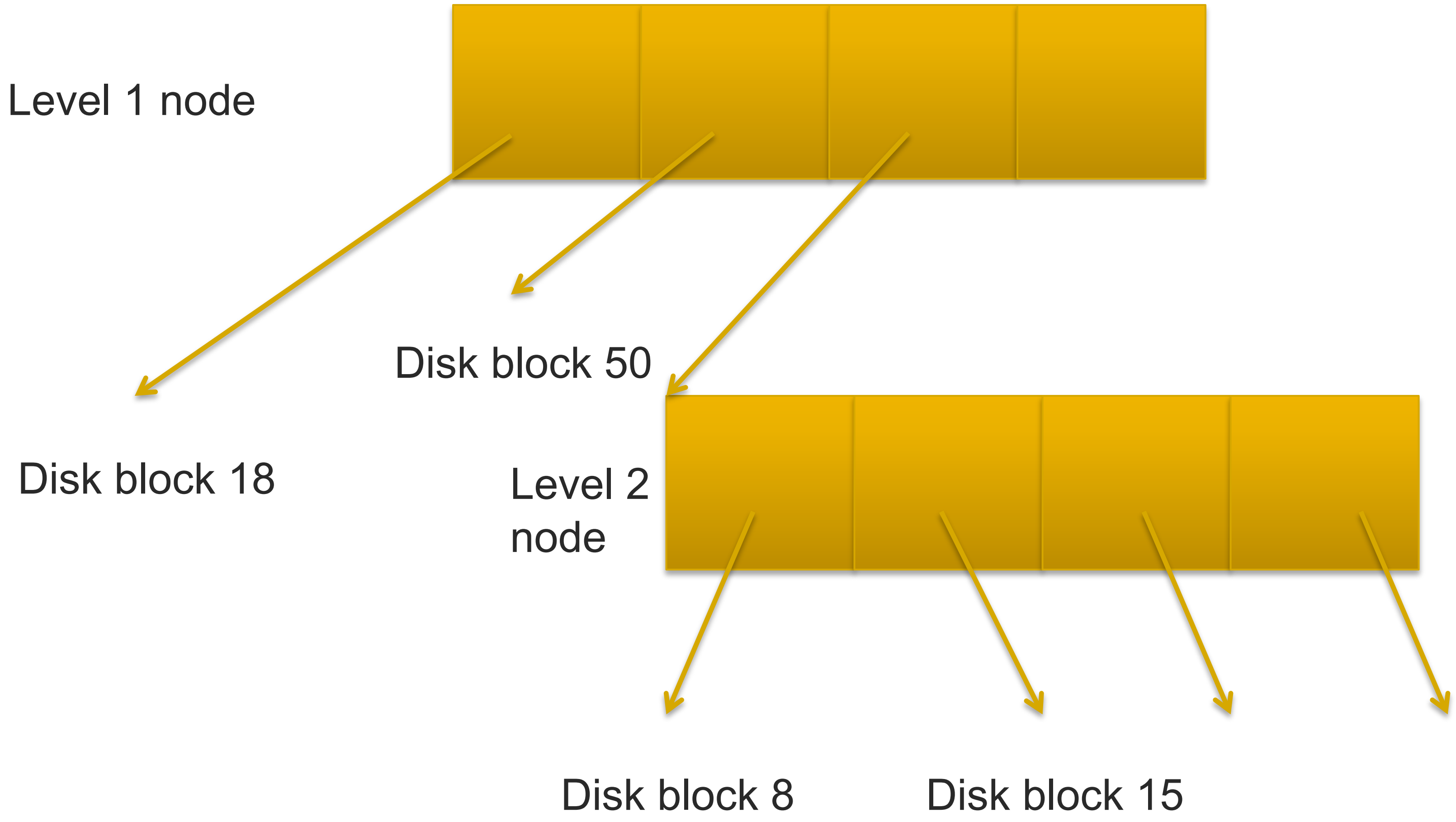
Disk block 15

Disk block 8

Multi-level indexed files

- How many disk accesses to get 1 block of data?
- How do you solve this?

Non-uniform multi-level indexed files



Non-uniform multi-level indexed files

- Pros
  - Files can expand easily
  - Small files don’t pay full overhead of deep trees
- Cons
  - Lots of indirect blocks for big files
  - Lots of seeks for sequential access

On disk file structures

- Could have other dynamically allocated data structures for file header
  - Key feature is to have the location of the file header on disk NOT change when the file grows.
    - Why?

Naming

- How do you specify which file you want to access?
  - Eventually OS must find the file header you want on disk
- Typically user uses symbolic name
  - OS translates name to numeric file header
  - Alternative is to describe to contents of the file

Locating file header disk block

- Could use hash table, expandable array
  - Key is to figure out the file number for the inode, then getting file contents is easy

- Data structure for mapping file name to inode block number is called a **Directory**

Directories

- A directory contains a mapping for a set of files
  - Name -> file header’s disk block # for that file
  - Often a simple array of (name, file header’s disk block #) entries
  - This table is stored in a normal file as normal data. E.g., “ls” can be implemented by reading this file and parsing its contents.

Directories

- We can often treat directories and files in the same way
  - Can use same storage structure to store data
  - Directory entries can point to either a file or another directory

- Can we allow the user to read/write directories arbitrarily?

Directory organization

- Directories typically have hierarchical struct.
  - Directory A has mapping to a bunch of files and **directories** in directory A
- E.g., /home/kingst/cs241-grades.txt
- / is root directory
  - Contains list of files and other directories
  - For each file/directory in /, has a mapping from name to a file header's disk block #
  - One of these entries is "home"

Directory organization

- Home is directory entry within the / dir.
  - Contains a list of files and directories
  - One of the directories in /home is kingst
- /home/kingst is a directory within the / home dir
  - Contains a list of files and other directories
  - One of the files it lists is “cs241-grades.txt”
- How many disk I/Os to access the first bytes of /home/kingst/cs241-grades.txt?