Based on slides by Harsha V. Madhyastha

EECS 482 Introduction to Operating Systems

Spring/Summer 2020

Lecture 17: File systems

Nicole Hamilton
https://web.eecs.umich.edu/~nham/
nham@umich.edu

Agenda

1. Grader2 is up.
2. Projects 3 and 4 extended 2 days.
3. Storage devices.
4. File systems.

Agenda

1. Grader2 is up.
2. Projects 3 and 4 extended 2 days.
3. Storage devices.
4. File systems.

Grader2 outage

What I've been told: Problem was a change to the Cosign authentication service used on grader2 webserver that wasn't compatible with the domain wildcarding it uses.

ITS installed an exception and it came back online around 6:00 pm yesterday.

What I've done: I've moved the due dates for P3 and P4 out by 2 days. It's all the slack I have.

P3 July 27 → July 29
P4 August 15 → August 17

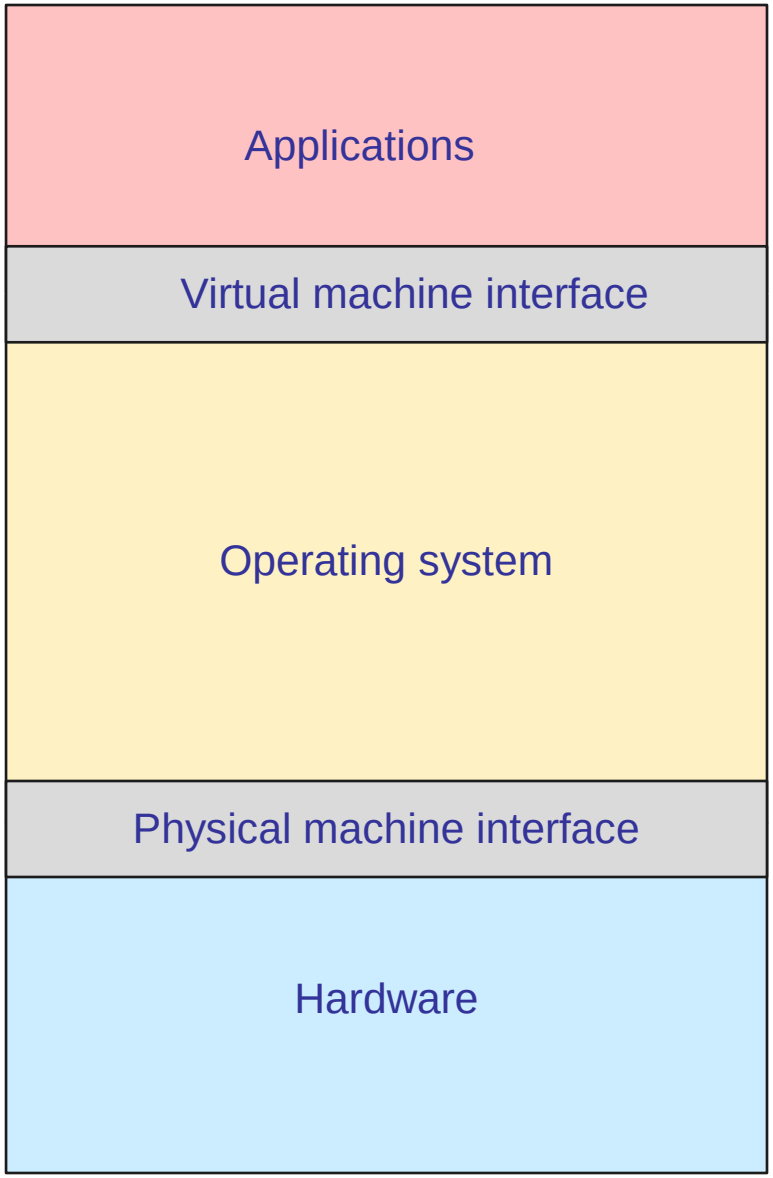
Agenda

1. Grader2 is up.
2. Projects 3 and 4 extended 2 days.
3. Storage devices.
4. File systems.

Dealing with heterogeneity

Many different types of disks and other devices and lots of different interfaces, e.g., ESDI, USB, SCSI, SATA, Fiber channel, m.2.

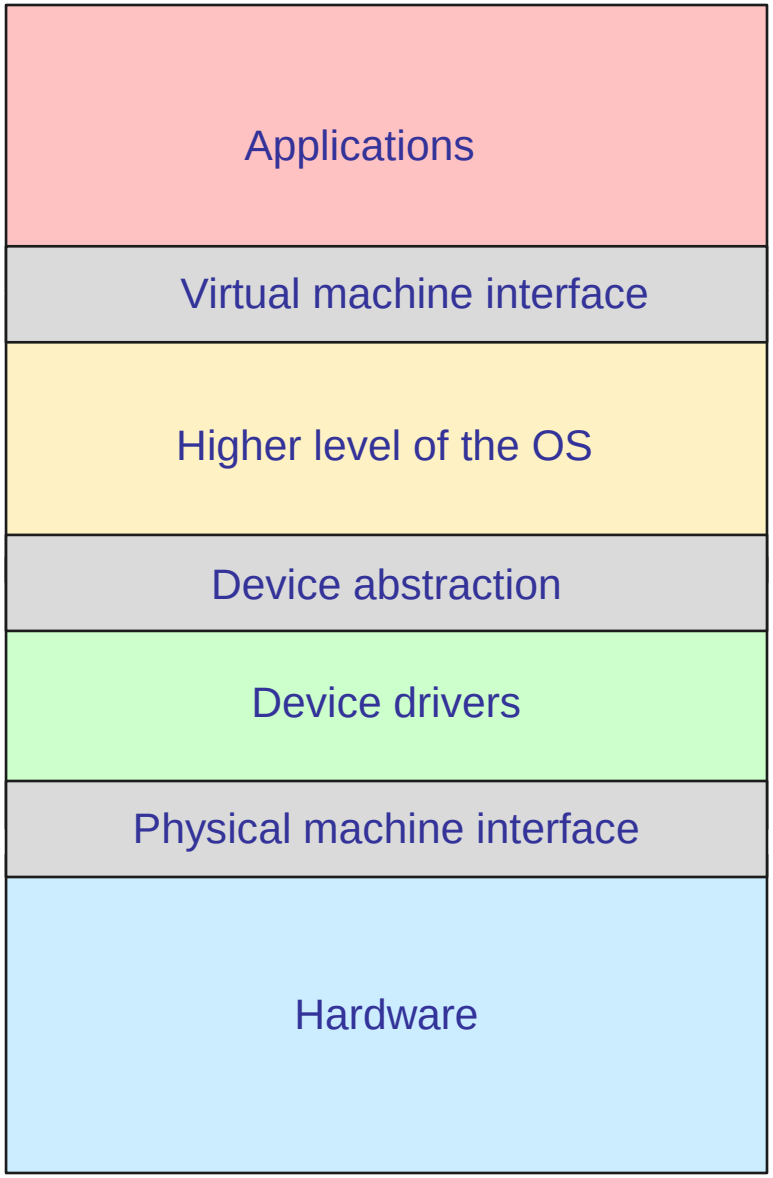
Need a way of managing this diversity.



Dealing with heterogeneity

Solution is to add a device driver abstraction inside the operating system to hide the differences between similar classes of devices.

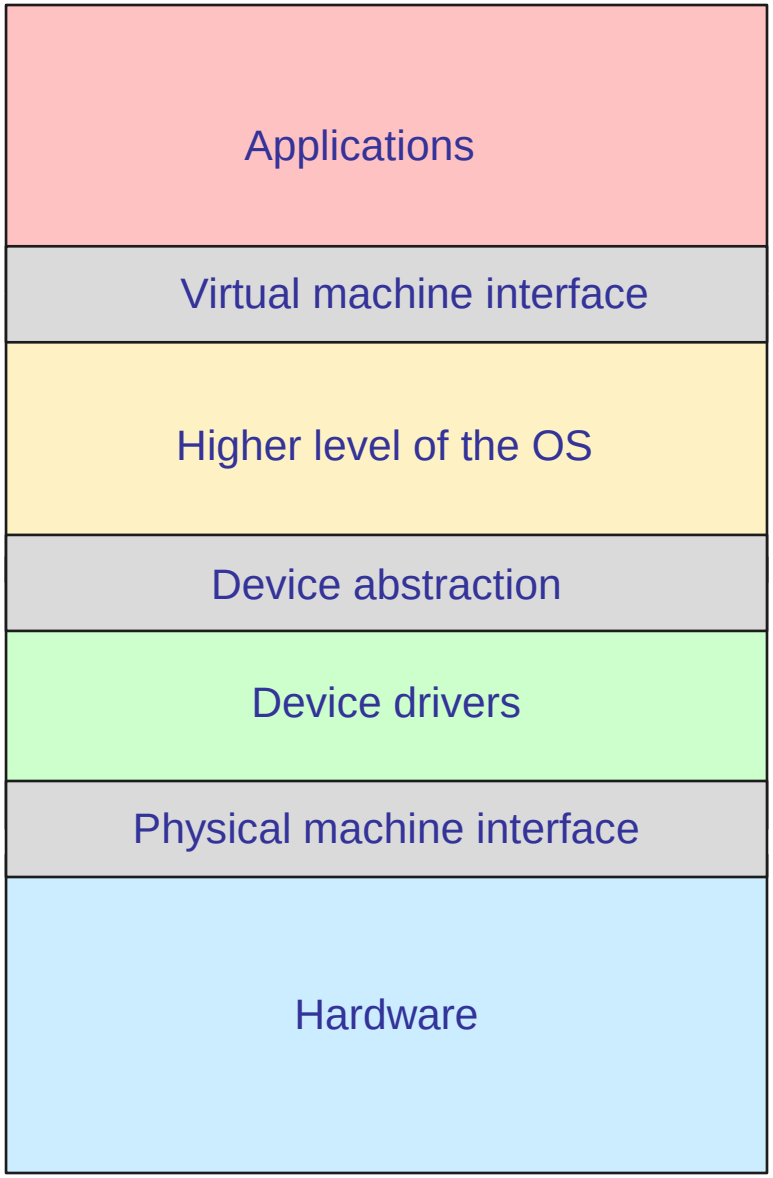
Device drivers create an abstraction of a disk as array of disk blocks.

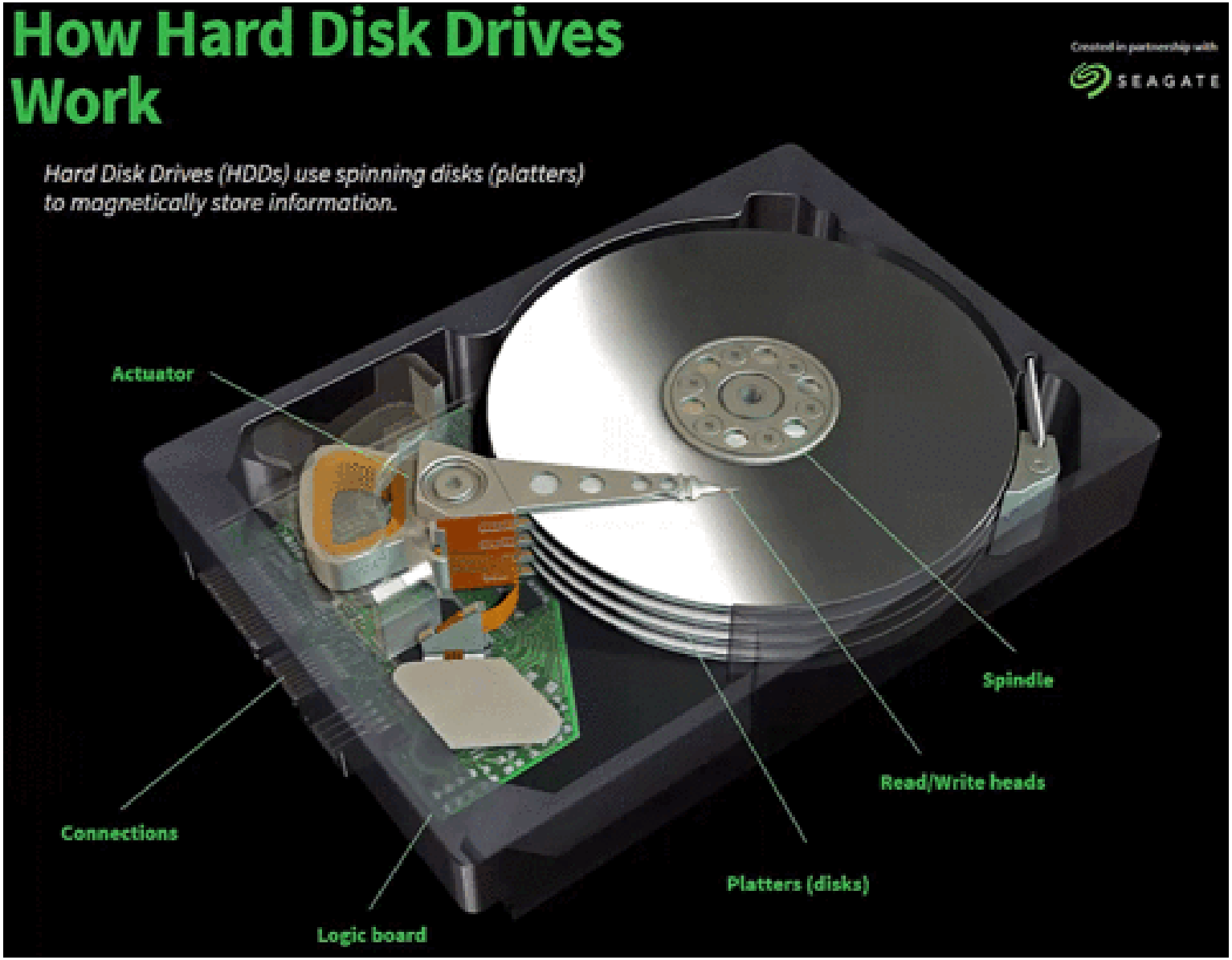


Dealing with heterogeneity

Device drivers are usually supplied by the device manufacturers.

Because they run as a trusted part of the kernel, in the past, they've been a major reason for Windows crashes.




How Hard Disk Drives Work
Created in partnership with
SEAGATE
Hard Disk Drives (HDDs) use spinning disks (platters) to magnetically store information.
Actuator
Spindle
Read/Write heads
Connections
Platters (disks)
Logic board

Storages devices are slow

How to account for slow seek times?

1. Caching.
2. Scheduling of I/O requests, SSTF, SCAN, C-SCAN.
3. Store related items together on disk, e.g., blocks within a file, files within a directory, files with the directory.

Agenda

1. Grader2 is up.
2. Projects 3 and 4 extended 2 days.
3. Storage devices.
4. File systems.

File systems

A *file system* is a data structure on disk which ensures that data persists across:

1. Power outages.
2. Machine crashes/reboots.
3. Process creation/exit.

How to enable persistence across these events?

1. Use persistent storage medium.
2. Write data carefully, ensuring the sequence of writes will not result in unusably corrupted data if the system crashes.
3. Avoid use of addresses that change across processes, meaning any addresses have to refer to locations on the disk.

The file system API

The application programming interface (API) to a file system typically provides these basic functions (plus many more.)

1. Create a file.
2. Delete file.
3. Read a file beginning at an offset or current position.
4. Write a file beginning at an offset or current position.
5. List a directory.
6. Create directory.
7. Move and rename files and directories.

Alternate interface: SQL → Database

File system workloads

Optimize data structure for the common case.

Some general rules of thumb:

1. Most file accesses are reads. The OS can spend more time writing if means reads will be faster.
2. Most programs access files sequentially and entirely.
3. Most files are small, 1KB to 10KB, but most of the space is taken up by large files.

File abstraction

Reality: One (or a few) disks to store data.

- Each is an array of (logical) blocks.

Abstraction: Numerous storage objects (files).

- Each is an array of bytes.

Challenges:

- How to name files and relate them to disk blocks?
- How to find and organize files?

How to store a file?

- Need to store metadata.
  - File name and size.
  - Owner and permissions.
  - Time of creation and last access.
- Need pointers to data.
  - Pointer must be independent of process virtual address.
  - Use logical block number to point to data on disk.
- Store in a *file header*.
  - inode in Unix, Master File Table record in NTFS.
  - Structure that describes file and allows you to find data.

Contiguous allocation

File = array of blocks (an “extent”).

Similar to base-and-bounds memory allocation.

Reserve space when the file is created.

If the file gets too big, move it to a larger free area.

File header contains starting location of file and size.

Pros and cons?

+ Fast sequential access.

+ Easy random access.

- Wastes space in external fragmentation.

- Difficult to grow file.

We will solve the fragmentation problem the same way we did with memory allocation.

Indexed files

File = fixed size array of block pointers.

Just like a page table.

File block #	Disk block #
0	18
1	50
2	-1
3	-1

Pros and cons?

+ Easy to grow a file.

+ Easy random access.

- But potentially slow for sequential access.

How could sequential access be improved?

When growing a file, allocate new blocks close to previous blocks on the same track or cylinder.

Increase the allocation unit with a larger block size or by allocating clusters of blocks together.

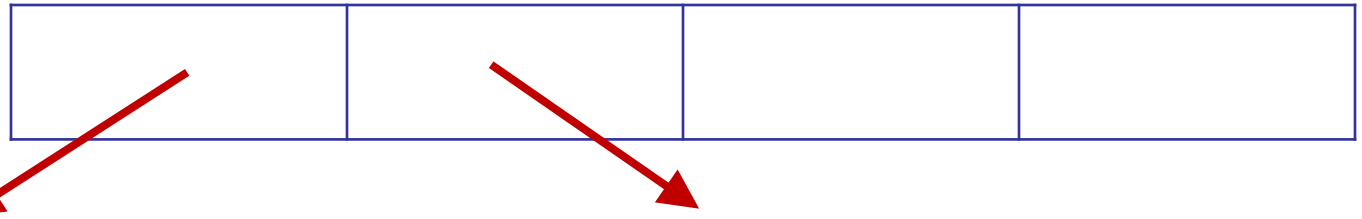
What happens with very large files or files with “holes”, places with no data?

You need an enormous table.

Multi-level indexed files

File = tree of block pointers

level 1 node



level 2 node

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

File block #	Disk block #
4	20
5	11
6	3
7	43

Pros?

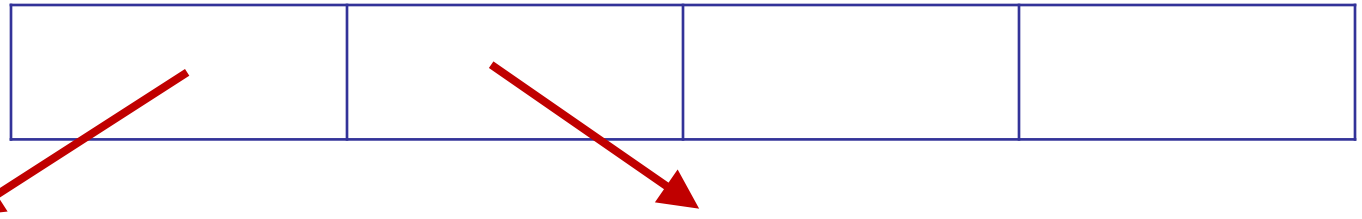
Files can easily grow and have holes.

Allows large files, but small files don't waste header space.

Multi-level indexed files

File = tree of block pointers

level 1 node



level 2 node

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

File block #	Disk block #
4	20
5	11
6	3
7	43

Cons?

Could have lots of seeks for sequential access.

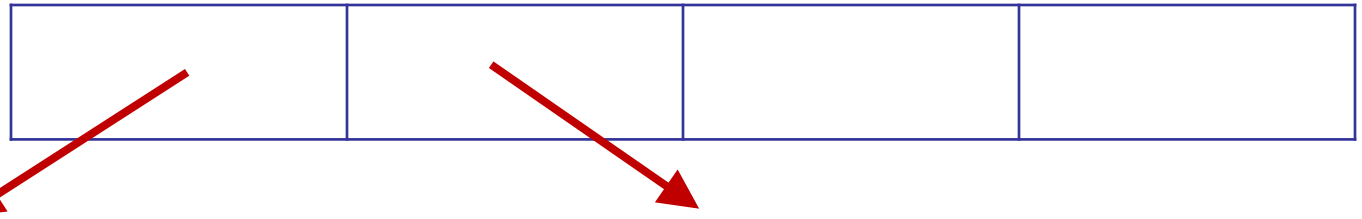
Performance hit due to the indirection.

Still, a limit on filesize.

Multi-level indexed files

File = tree of block pointers

level 1 node



level 2 node

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

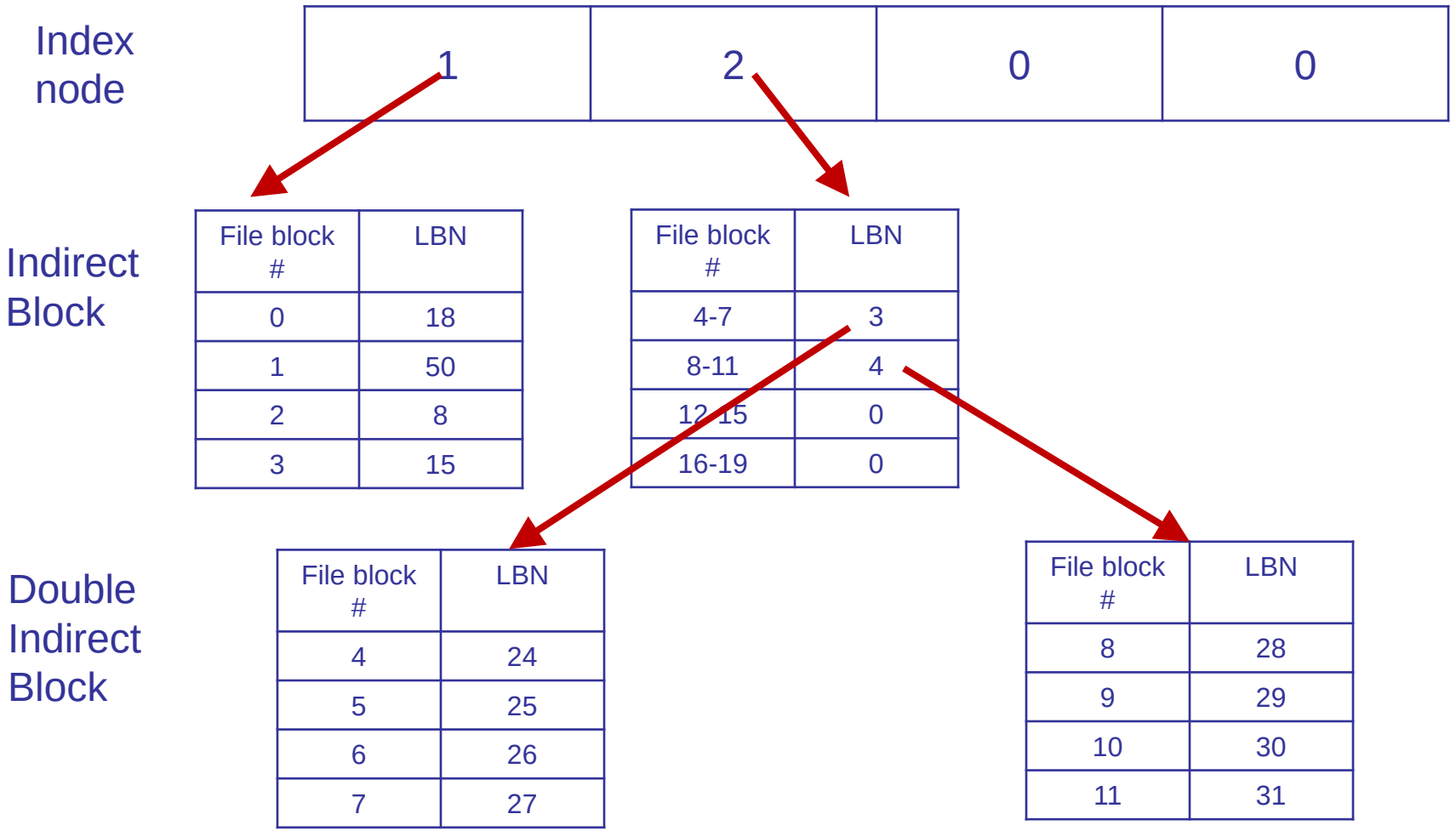
File block #	Disk block #
4	20
5	11
6	3
7	43

Solution

Performance improved with caching, which works well because the block pointers are small.

File size limits are increased with non-uniform depth indices.

Non-Uniform depth for extremely large files



Representing files

Can have other dynamic ways of allocating file.

Must ensure that location of file header does not change as the file grows.

Example: Header is head of linked list.

+ Easy to append or insert.

- Slow sequential access.

- Really slow random access.

Naming and directories

How to specify file to be accessed?

- File name, click on icon, or by attributes or contents.

File name is usually a hierarchical path.

- E.g., `/users/nham/482/notes`
- Allows users to group related files into one folder and assign permissions.
- Allows easy searching, e.g., "`ls /users/nham/482`"

Must translate file name to disk block # of header.

- What data structure to use to store mapping?
- Tree of directories.
- Why not a hash table?
- Hard to traverse single directory.

Directories

Directory: mapping information for a set of files

- Name of file → file header's disk block # for that file.
- Once, array of (name, file header's disk block #) entries.
- Modern file systems: hash table or B-tree.

Directories and files are largely equivalent.

- Same storage structure.
- Directory entry points to inode for file or directory.

Directory Example

/ directory

Name	Block #
“bin”	100
“users”	35
“tmp”	43
“foo.txt”	254

/users directory

Name	Block #
“harshavm”	23
“pmchen”	99
“nham”	0
	0

/users/nham directory

Name	Block #
“482.txt”	44
	0
“src”	55
“foo.txt”	33

Any differences in allowing application to update file versus directory?

Users can put arbitrary data in a file. But a user can’t be allowed to corrupt the file system by writing junk to a directory, solved with limited set of system calls for updating directories.

Example: /users/nham/482/notes

1. Read the file header for / (root directory), which contains pointers to data blocks of the / directory.
2. Read data blocks of /, contains list of the files and directories in /. Each entry contains amapping from name → header's disk block #. One of those entries is "users".
3. Read file header for /users.
4. Read data blocks for /users.
5. Read file header for /users/nham.
6. Read data blocks for /users/nham.
7. Read file header for /users/nham/482.
8. Read data blocks for /users/nham/482.
9. Read file header for /users/nham/482/notes.
10. Read first data block for /users/nham/482/notes.

May be helped by caching the file header for the current working directory.

Unified view of multiple storage devices

Combine multiple storage devices into a file system

- Each device contains own file system (starting with its root)
- A directory entry can point to the root of a different device (often a mount point for a new file system).

Example:

- / (root)
  - bin (same device as /)
  - etc (same device as /)
  - tmp (separate storage device)
  - afs (network storage “device”)

Directory entry: 1) file, 2) directory, or 3) device