

CSE 120

Operating Systems Principles

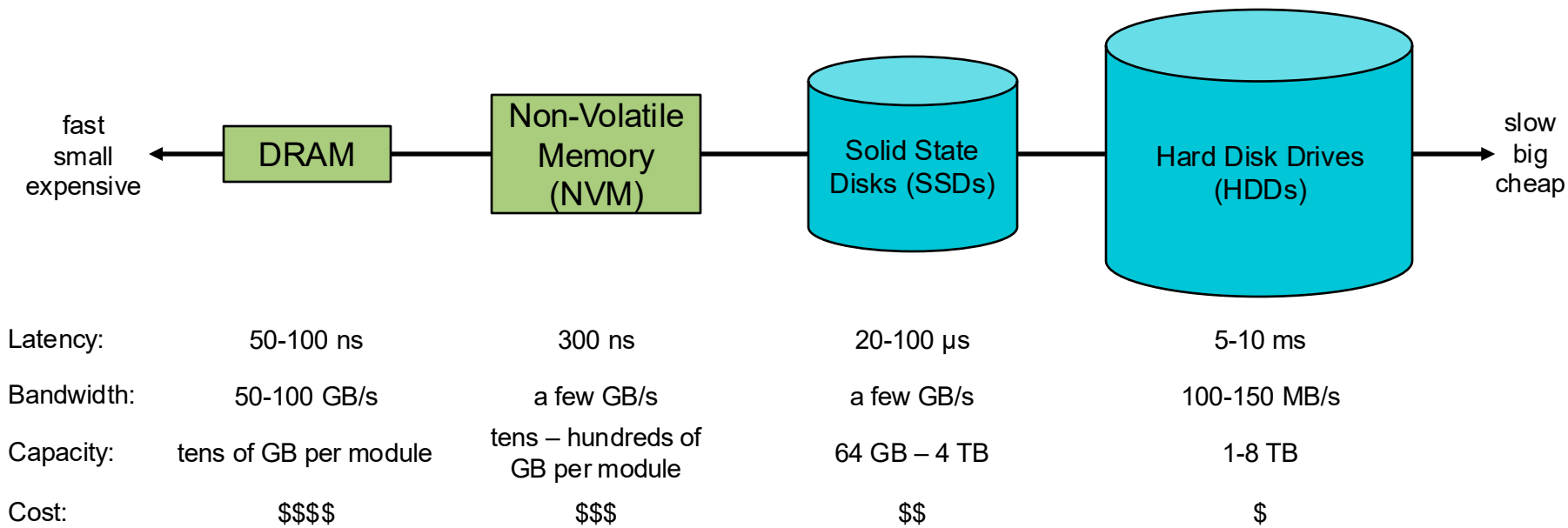
Spring 2025

Lecture 15: File System Disk Layout

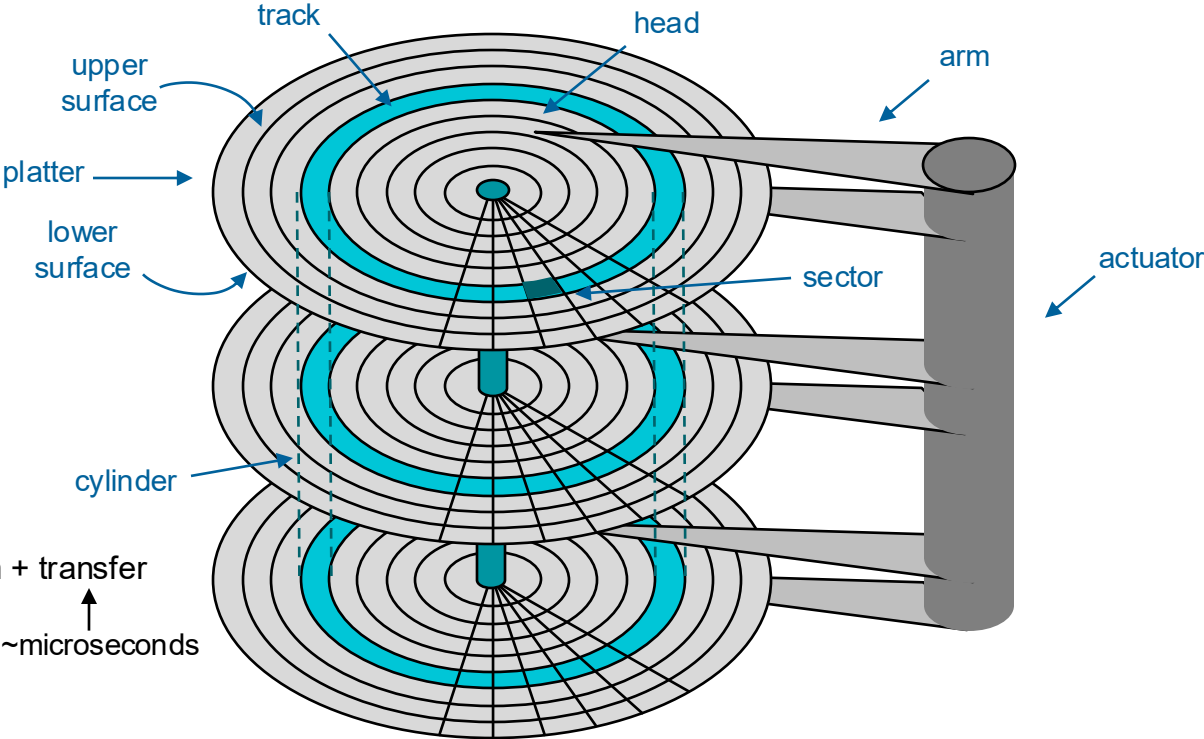
Amy Ousterhout

Memory and Storage Technologies

- Tradeoffs between latency, bandwidth, capacity, and cost



Hard Disks



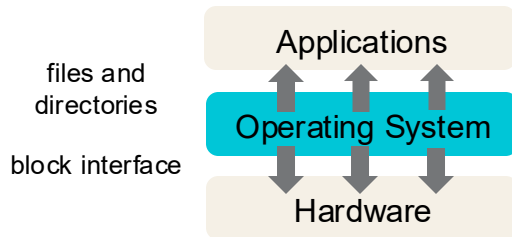
Disk latency = seek + rotation + transfer

 ↑ ↑ ↑

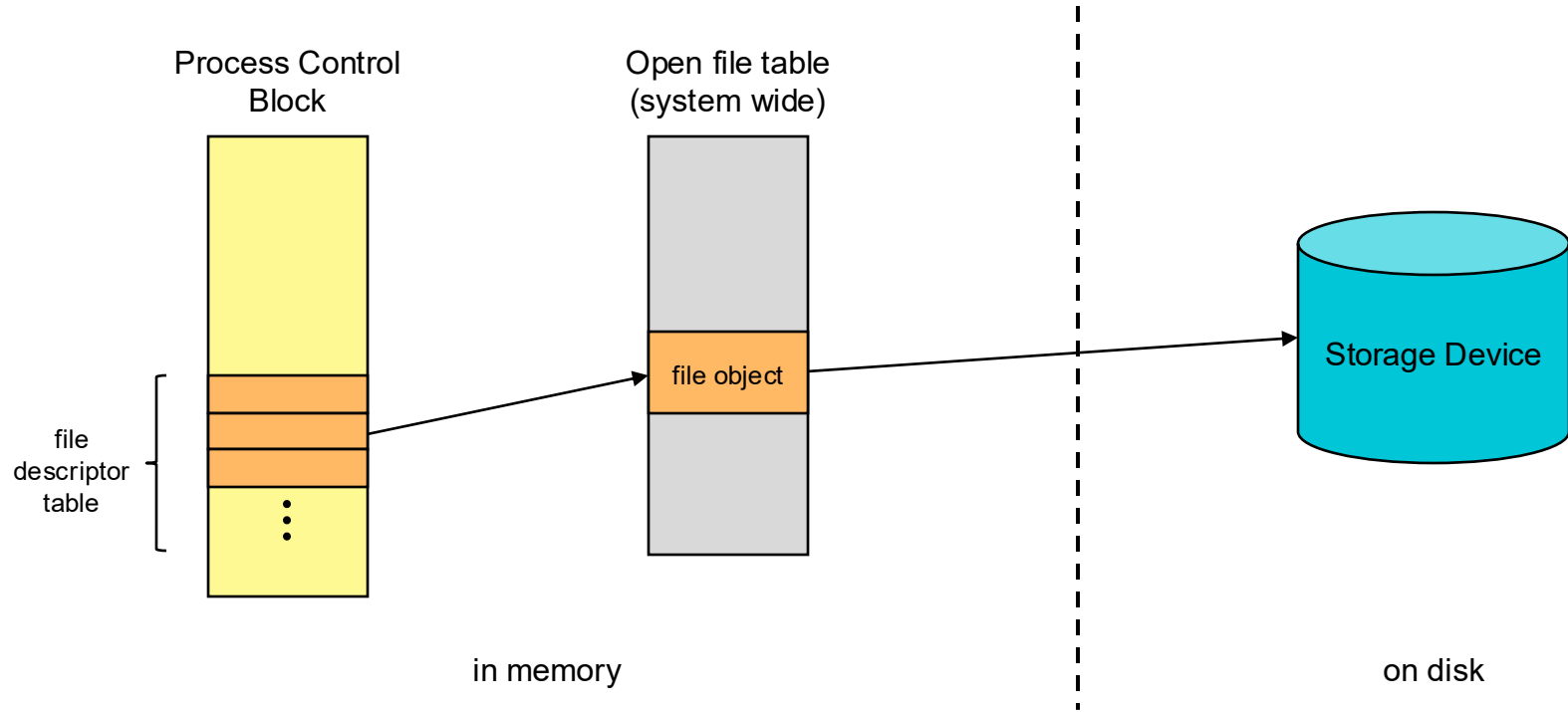
 ~milliseconds ~microseconds

File System APIs

- API between the OS and storage devices
 - **Block interface:** disk maps logical blocks to surface/track/sector
 - OS refers to blocks by their number
- API between the OS and applications
 - **File:** a named collection of bytes
 - **Directory:** a way to organize files logically



Typical File System Data Structures

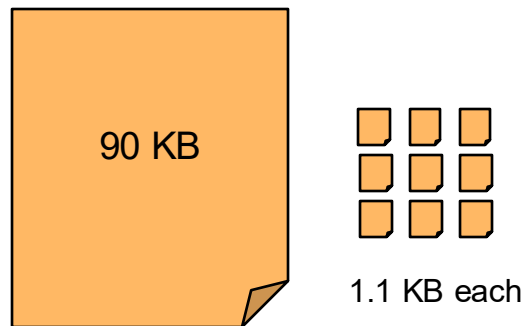


File System Challenges

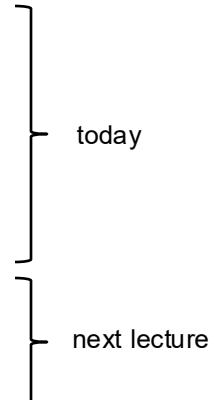
- Files grow and shrink
 - Little a priori knowledge
 - 6-8 orders of magnitude in file sizes
- Disk performance behavior
 - Highly nonuniform access
 - Efficiency can be challenging
- Failures

File System Workloads Motivate Design

- Workloads can influence the design of a file system
- File characteristics (measurements of UNIX and NT)
 - Most files are small (about 8KB)
 - Most of the disk is allocated to large files
 - » 90% of data is in 10% of files
- Access patterns
 - Sequential vs. random
 - Access files in the same directory together
 - » Spatial locality
 - Access metadata at the same time as the file
 - » Need metadata to find data



Key Questions

- How do we keep track of blocks used by a file?
 - Where do we store metadata information?
 - How do we (really) do path name translation?
 - How do we implement common file operations?
 - How can we cache data to improve performance?
 - How can we handle disk crashes properly?
- 
- today
- next lecture

Today's Outline

- File system layout
 - Managing disk space
 - Metadata
 - Path name translation
- Common file operations
 - Hard links
 - Symbolic (soft) links
 - Create
 - Rename
 - Delete

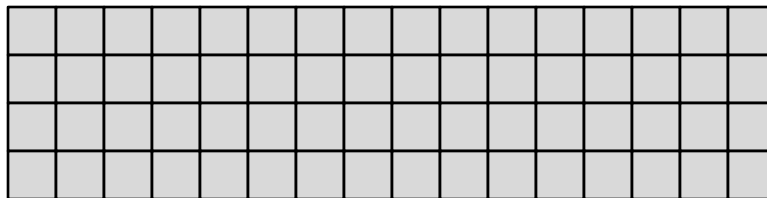
File System Layout


- We start with an empty disk

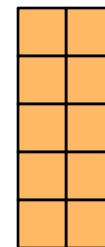


File System Layout – Blocks

- Partition the disk into fixed-size file system blocks

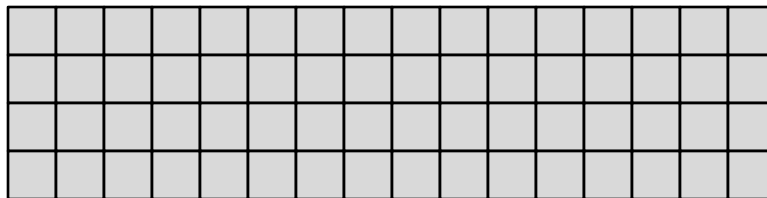


- Typically 4 KB in size
 - Block size is set when the file system is formatted
- Independent of disk physical sector size
 - If a sector is 512 bytes, file system will use 8 sectors/block
- One file can span multiple disk blocks (e.g., 40 KB file uses 10 blocks)
- A small file still uses an entire block 



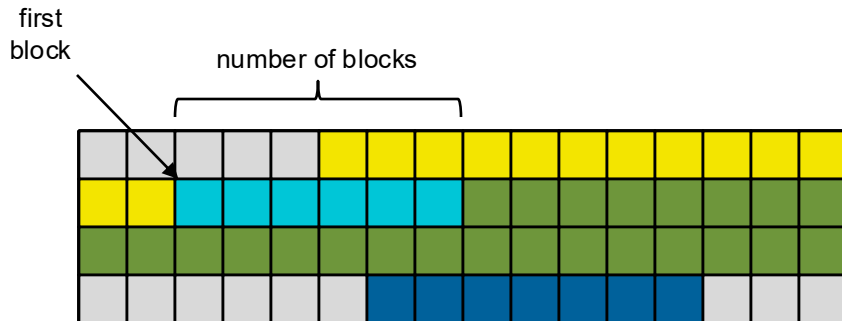
File System Layout – Block Management

- How should we allocate blocks to files?
- How should we keep track of which blocks are for which file?
- We will consider several different approaches



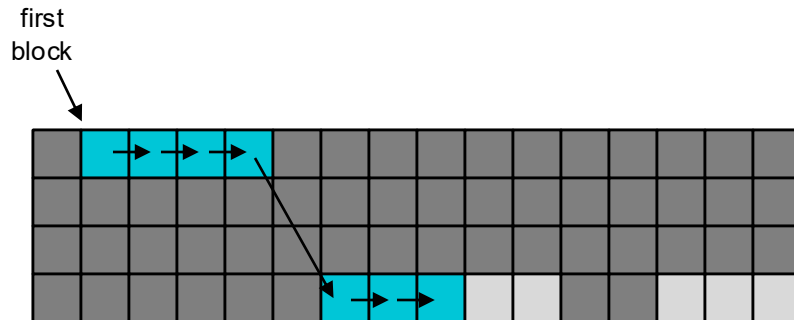
Contiguous Layout (or “extent-based”)

- Allocate a contiguous set of blocks to each file (e.g., IBM OS/360)
- File metadata:
 - Location of first block on disk
 - Number of blocks
- Pros:
 - Simple
 - Easy access, both sequential and random
 - Few seeks for I/O
- Cons:
 - Difficult to grow files
 - As files are created and deleted, fragmentation can occur





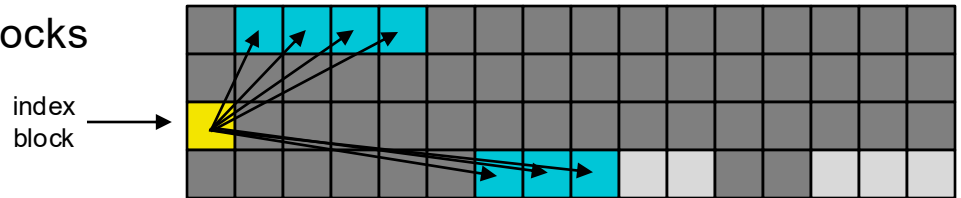
Linked Layout

- Allocate a linked list of blocks (e.g., Xerox Alto)
 - Essentially a linked list on disk per file
- File metadata:
 - Location of first block on disk
 - Each block contains a pointer to the next
- Pros:
 - Can grow files dynamically
 - No fragmentation
- Cons:
 - Random access is now slow
 - Sequential bandwidth may not be good
 - Unreliable: losing one block means losing the rest



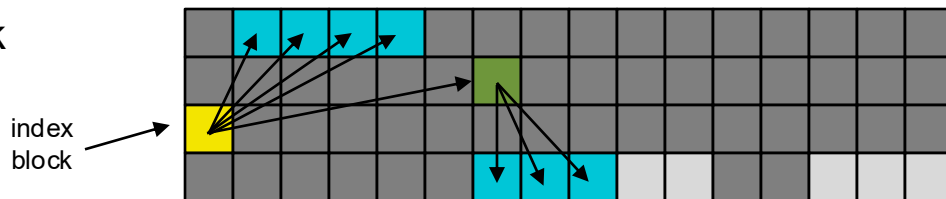
Indexed Layout

- Use a special index block  to store pointers to the data blocks 
- File metadata:
 - Location of the index block on disk
 - Index block contains pointers to blocks
- Pros:
 - Can grow files dynamically
 - No fragmentation
 - Easy random access (after reading the index block)
- Cons:
 - What if one index block is not big enough?
 - Sequential bandwidth may still not be good



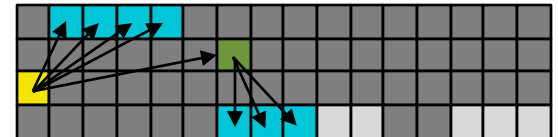
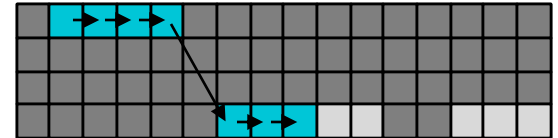
Multi-Level Indexed Layout

- Use a special index block (yellow) to store pointers to the data blocks (cyan) and indirect blocks (green) to store more pointers to data blocks
- File metadata:
 - Location of the index block on disk
 - Index block contains pointers to blocks and indirect blocks
- Pros:
 - Can support much larger files
- Cons:
 - There is still a limit on the maximum file size
 - Sequential bandwidth may still not be good



Disk Layout Summary

- Files span multiple disk blocks
- Where are the blocks for a file located?
 - **Contiguous allocation**
 - » Fast, simplifies directory access
 - » Inflexible, causes fragmentation, needs compaction
 - **Linked structure**
 - » Each block points to the next, directory points to the first
 - » Random access is slow
 - **Indexed structure** (indirection, hierarchy)
 - » An “index block” contains pointers to many other blocks
 - » Handles random better
 - » May need multiple index blocks



Unix Inodes

- Each file is associated with an **inode** (index node) on disk
 - Each inode has a unique inode number
- An inode stores all the **metadata for a file**:
 - All the data blocks of the file can be found from its inode
 - File size (in bytes and blocks)
 - User and group of file owner
 - Protection bits
 - » User/group/other, read/write/execute
 - Link count
 - » How many directory entries point to this inode
 - Timestamps
 - » Created, modified, last accessed, any change

Viewing Metadata

- `ls -l` and `stat` read from the directory and inode
 - Use the `stat` system call

```
[cs120sp25@ieng6-201]:nachos:556$ ls -alU
```

```
total 88
drwxr-x--- 16 cs120sp25 ieng6_inc24 4096 Mar 28 00:29 .
drwxr-x---  3 cs120sp25 ieng6_inc24 4096 May 19 20:49 ..
drwxr-x---  3 cs120sp25 ieng6_inc24 4096 May 18 13:12 proj3
drwxr-x---  3 cs120sp25 ieng6_inc24 4096 May 11 17:17 proj2
drwxr-x---  2 cs120sp25 ieng6_inc24 12288 May 18 15:25 test
drwxr-x---  2 cs120sp25 ieng6_inc24 4096 Mar 28 00:26 bin
drwxr-x---  2 cs120sp25 ieng6_inc24 4096 Mar 28 00:26 security
-rw-r----- 1 cs120sp25 ieng6_inc24 1823 Mar 15 11:05 Makefile
drwxr-x---  2 cs120sp25 ieng6_inc24 4096 Mar 28 00:26 network
-rw-r----- 1 cs120sp25 ieng6_inc24 8495 Mar 15 11:05 README
drwxr-x---  2 cs120sp25 ieng6_inc24 4096 Mar 28 00:26 proj0
drwxr-x---  2 cs120sp25 ieng6_inc24 4096 Mar 28 00:26 vm
drwxr-x---  2 cs120sp25 ieng6_inc24 4096 May  8 23:06 threads
drwxr-x---  3 cs120sp25 ieng6_inc24 4096 May  2 14:29 proj1
drwxr-x---  2 cs120sp25 ieng6_inc24 4096 Mar 28 00:26 ag
drwxr-x---  2 cs120sp25 ieng6_inc24 4096 May  8 22:38 machine
drwxr-x---  2 cs120sp25 ieng6_inc24 4096 May 18 14:36 userprog
drwxr-x---  8 cs120sp25 ieng6_inc24 4096 May 18 15:53 .git
```

name and inode
from the
directory

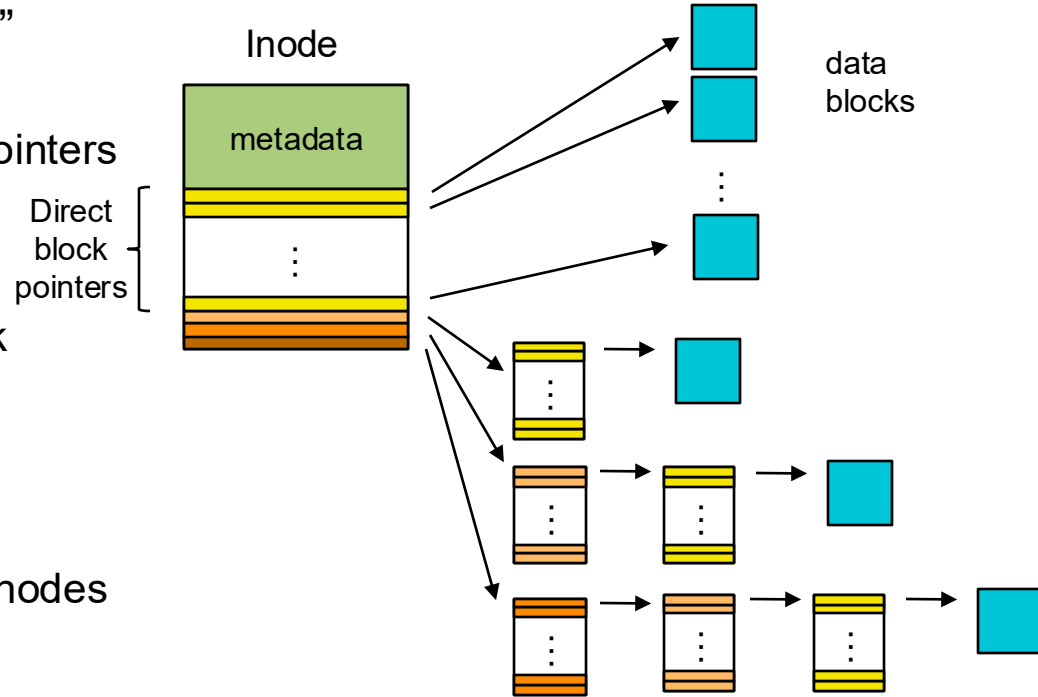
metadata from
the inode

```
[cs120sp25@ieng6-201]:nachos:551$ stat Makefile
```

```
File: Makefile
Size: 1823          Blocks: 8          IO Block: 65536  regular file
Device: 2ah/42d Inode: 238417665  Links: 1
Access: (0640/-rw-r-----)  Uid: (32304/cs120sp25)   Gid: (32304/ieng6_inc24)
Access: 2025-05-18 12:48:14.863529000 -0700
Modify: 2025-03-15 11:05:42.000000000 -0700
Change: 2025-03-28 00:26:56.055954000 -0700
Birth: -
```

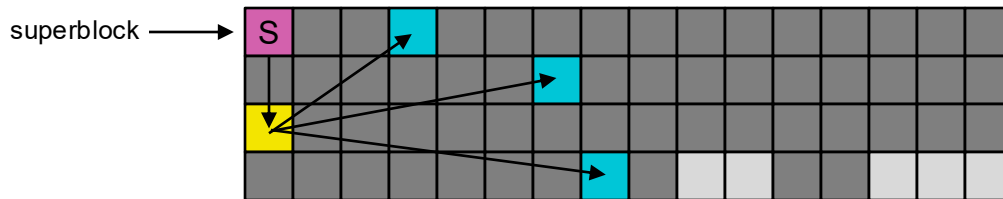
Unix Inodes and Indirect Blocks

- Unix inodes use an “unbalanced” indexed structure
 - Each inode contains ~15 block pointers
 - First 12 point to direct blocks
 - 1 points to a single indirect block
 - 1 points to a double indirect block
 - 1 points to a triple indirect block
- Inodes are small
 - 256 bytes each
 - One inode block contains many inodes



Superblock

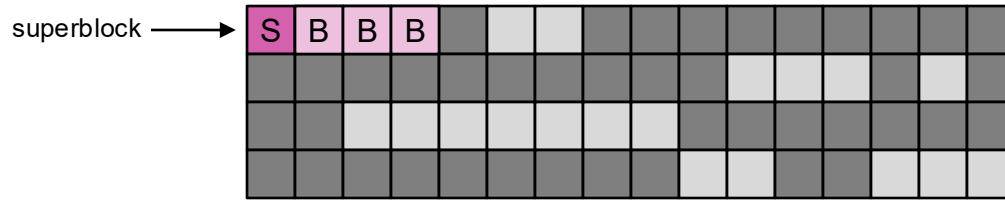
- “/” is the directory that is the root of the file system
- How do we find the inode for “/”?



- The superblock stores a pointer to the inode of “/”
 - The inode for “/” has pointers to all of the blocks storing the directory entries for “/”
- It is the basis for translating all path names
- It is at a fixed, pre-determined location on disk

Block Allocation

- File system needs to keep track of which blocks have been allocated and which are free



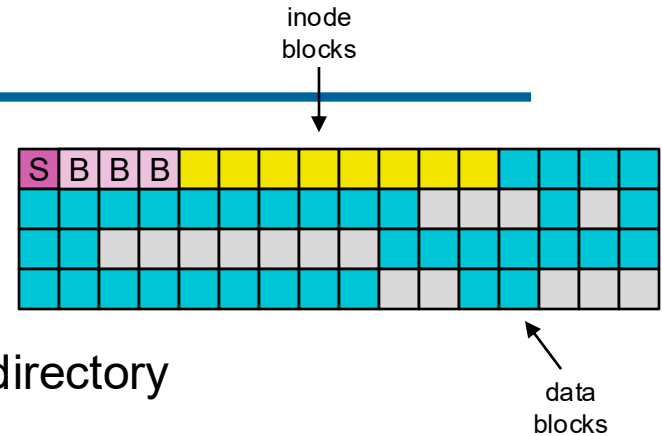
- Free map blocks **B** store a bitmap, one bit per block
 - A set bit indicates that the block is allocated
 - One bitmap for data blocks
 - Another for inode blocks

Block Allocation - Tradeoffs

- **Bitmap**
 - Pro: easy to find contiguous blocks
 - Con: need extra space to store the bitmap
- **Linked list**
 - Pro: no wasted space for bitmap
 - » Use free blocks themselves to store free block list
 - Con: harder to find contiguous blocks

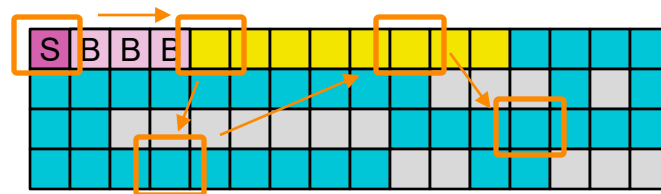
File System Layout

- How do file systems use the disk to store files?
- File systems define a **block size** (e.g., 4KB)
 - Allocate disk space in the granularity of blocks
- A **superblock** determines the location of the root directory
 - Always at a well-known disk location
- A **free map** determines which blocks are free and allocated
 - Usually a bitmap, one bit per block on the disk
- **Inodes** store metadata about files and the locations of their blocks
- Remaining disk blocks used to store **files** and **directories**
 - There are many ways to do this



Revisiting Path Name Translation

- Suppose you want to open “/one”
- What does the file system do?
 - To open “/one”, use superblock to find inode for “/”
 - Read inode for “/” into memory
 - Open and read “/” directory, look for entry “one”
 - This entry specifies the inode number
 - Read the inode for “one” into memory
 - The inode says where the first data block is on disk
 - Read that block into memory to access the data in the file



Poll – Memory Mappings

- Which of the following statements is **incorrect**?
 - A: a virtual page can have no physical page associated with it
true – the page could be swapped out or not mapped
 - B: two virtual pages in different processes can map to the same physical page
true – shared memory or copy on write
 - C: it is possible to translate from a virtual to a physical address without accessing any page tables
true – if the page table entry is cached in the TLB
 - D: one virtual page in one process can map to two different physical pages
false – a virtual page has a single page table entry and maps to at most one physical page
 - E: for some virtual addresses, there might not exist corresponding page table entries
true – unmapped pages with multi-level page tables

Poll – System Calls and Memory Management

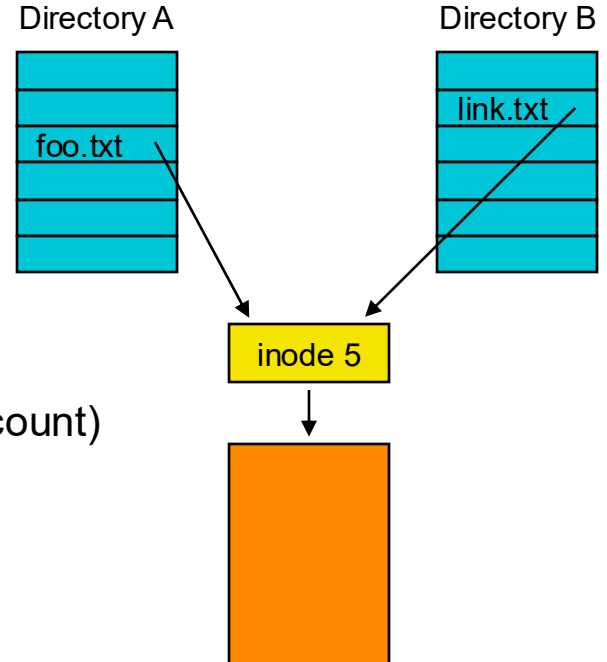
- Which of the following tasks involves a system call?
 - A: growing the heap *brk or mmap*
 - B: growing the stack *triggered by a page fault*
 - C: handling a TLB miss *handled by hardware or else by a fault to the OS (software)*
 - D: page eviction *part of page fault handling*
 - E: writing to shared memory *just a regular load or store instruction (e.g., MOV)*

Today's Outline

- File system layout
 - Managing disk space
 - Metadata
 - Path name translation
- Common file operations
 - Hard links
 - Symbolic (soft) links
 - Create
 - Rename
 - Delete

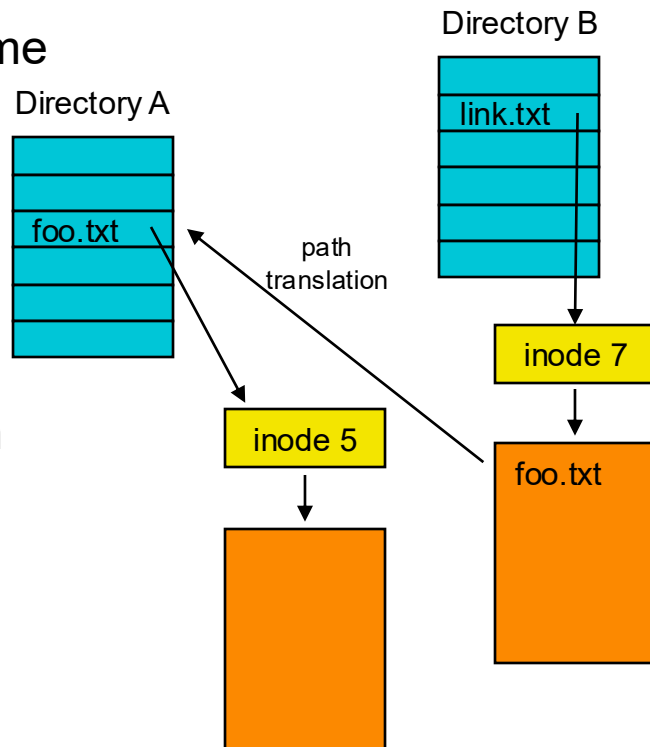
Hard Links

- **Hard link**: a directory entry that associates a name with a file
- Enables a form of **aliasing**:
 - `$ ln file alias`
 - Uses **link** syscall
 - Alias points to the same inode
- “.” and “..” names are hard links to directories
- Deleting a link may or may not remove the file
 - Depends on whether it's the last link (use reference count)
- Limitations:
 - Users cannot create hard links to directories
 - Cannot create hard links to files in other file systems



Symbolic (Soft) Links

- **Soft link**: a file whose contents are another file name
 - File contains the file name
 - Inode is flagged as a soft link
- Enables another form of aliasing:
 - `$ ln -s file alias`
 - Uses **symlink** syscall
- Soft link creates an alias via path name translation
 - File system reads the path alias from the file
 - File system restarts translation
- Limitations:
 - Slower than hard links



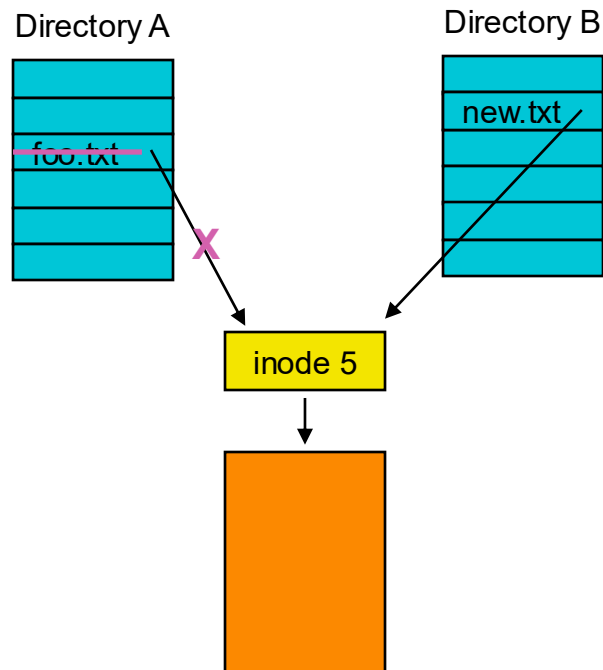
(Live Demo of Links)

Create

- Creating a file “new.txt” is pretty straightforward
- Allocate an **inode**
 - Initialize the metadata (owner, protection, timestamp, etc.)
 - Update inode bitmap
- Allocate a **directory entry**
 - Entry maps “new.txt” to the allocated inode
- When process starts writing, allocate data blocks
 - Update inode to point to allocated data blocks
 - Update data block bitmap
 - Continue to allocate blocks on demand

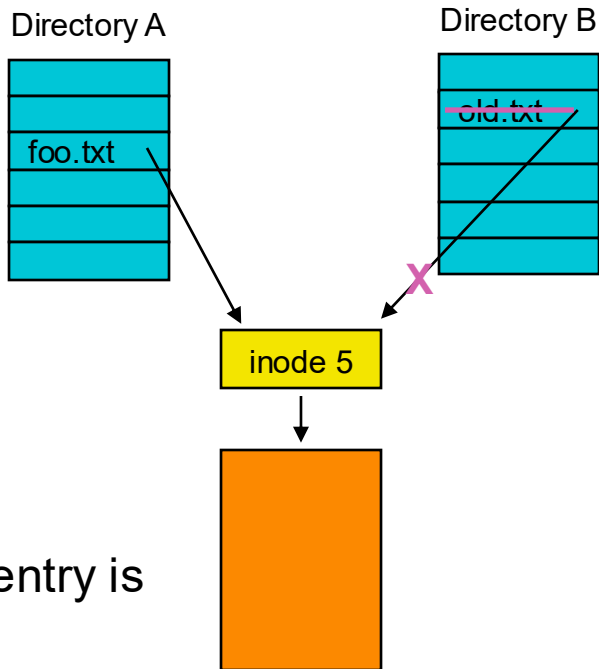
Rename

- One approach: create a new file with the new name, copy contents, delete old file
- More efficient approach:
 - Create a new directory entry with the new name
 - Point to the existing inode
 - Delete the directory entry with the old name
- Execute with mv:
 - `$ mv old new`
 - Uses `rename` syscall



Delete

- Steps for deleting a file “old.txt”:
 - Remove the directory entry for “old.txt”
 - » Hence the syscall name unlink
 - Decrement the reference count in the inode
 - If the file still has links to it, that’s all
 - If there are no remaining links:
 - » Free up the data blocks (update the data block bitmap)
 - » Free up the inode (update the inode bitmap)
 - » Block data is not erased
- If the file is still open in any process, the directory entry is removed but the file blocks are not



Upcoming Tasks

- Discussion tomorrow: project 3
- Read chapters 41-42
- Project 2 code reviews (task 4)
 - Must be completed **by the end of Friday (tomorrow)** – see Piazza for details
- Homework 4
 - **Due Tuesday 6/3** at 11:59 pm
- Project 3
 - **Due Friday 6/6** at 11:59 pm, no option to submit late

