File System Implementation

2023 Fall COMP3230A

Contents

Related Learning Outcome

- ILO 2d - describe the principles and techniques used by OS to support persistent data storage

Readings & References

- Required Readings
  - Chapter 40 – **File System Implementation**
    - http://pages.cs.wisc.edu/~remzi/OSTEP/file-implementation.pdf

- Reference
  - Chapter 42 – **Crash Consistency: FSCK and Journaling**
    - http://pages.cs.wisc.edu/~remzi/OSTEP/file-journaling.pdf

Management of Files and Storages

- A file needs space to store its contents
- The file system needs space to store its management data related to files and storage space – metadata
- Where are these metadata being stored? Of course, on the disk
- What kinds of on-disk structures are used by the file system to organize files' data and its metadata?
  - Superblock (for Windows, similar set of info can be located via Master File Table)
  - Inode table (for Windows – Master File Table)
  - Data block bitmap (for Windows – Cluster Bitmap)
  - Inode bitmap

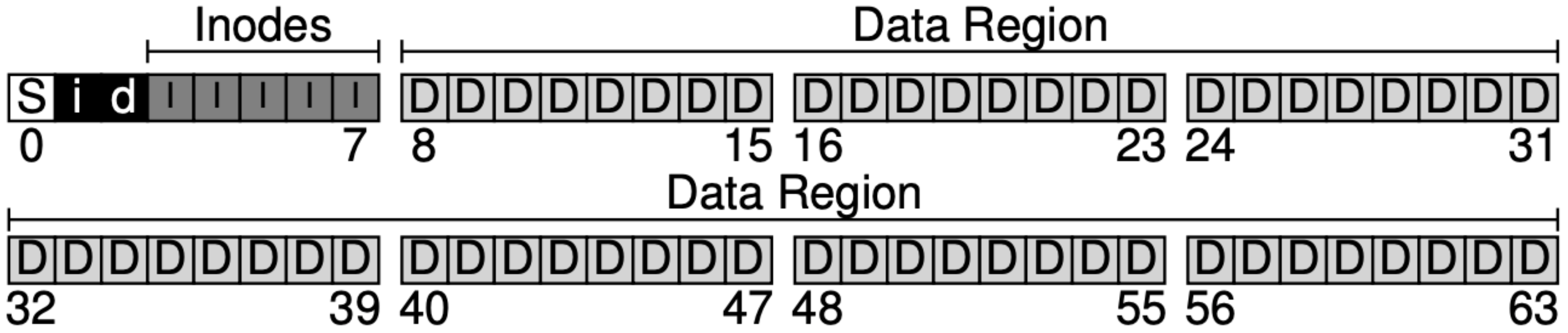
Superblock

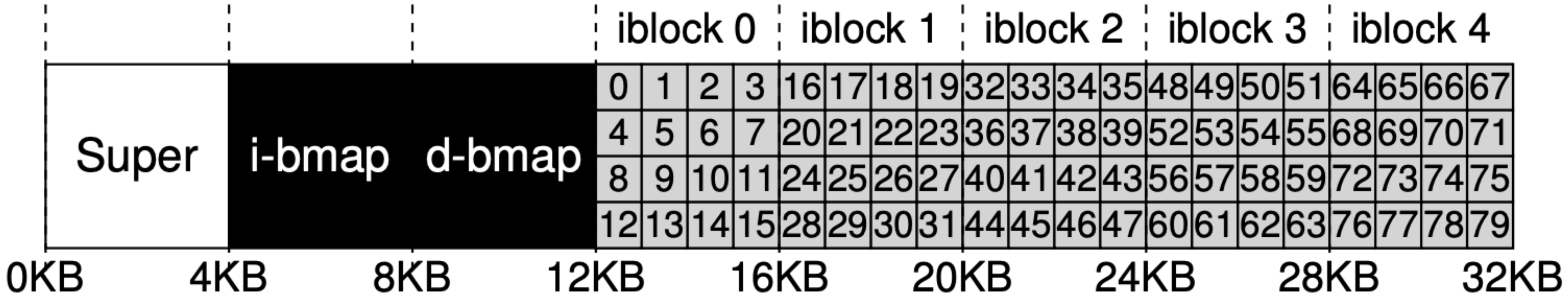
- Superblock is a data structure contains information about the **file system as a whole**
- A superblock might contain
  - The file system identifier
  - Tell us the number of inodes (file control blocks) and data blocks available in this file system
  - Where is the inode table? Inode bitmap? Data block bitmap?
  - Information of the inode of the root directory
- When mounting a file system, OS reads the superblock to locate other on-disk data structures
  - Usually the superblock is found at the first disk block of this file system
- To reduce the risk of data loss, most file systems distribute redundant copies of the superblock throughout the storage device

Inodes Table & Inode Bitmap

- Each file has an inode (**file control block**) that stores the metadata of a file
- Where are the inodes being stored?
  - Some disk blocks are reserved for holding these file control blocks – in the form of an array of inodes (**inode table**)
  - We can view the inode number (low-level id) as an index to the inode table
- Inode bitmap
  - When a file is created, one inode will be allocated from the inode table for this new file
  - When a file is deleted, the inode will be released back to the inode table
  - Thus, the system needs a mechanism to easily check which inodes in the inode table are free or are in used
  - A **inode bitmap** is a simple structure with each bit is used to indicate whether the corresponding inode is free or in-use in the inode table

On-Disk Organization





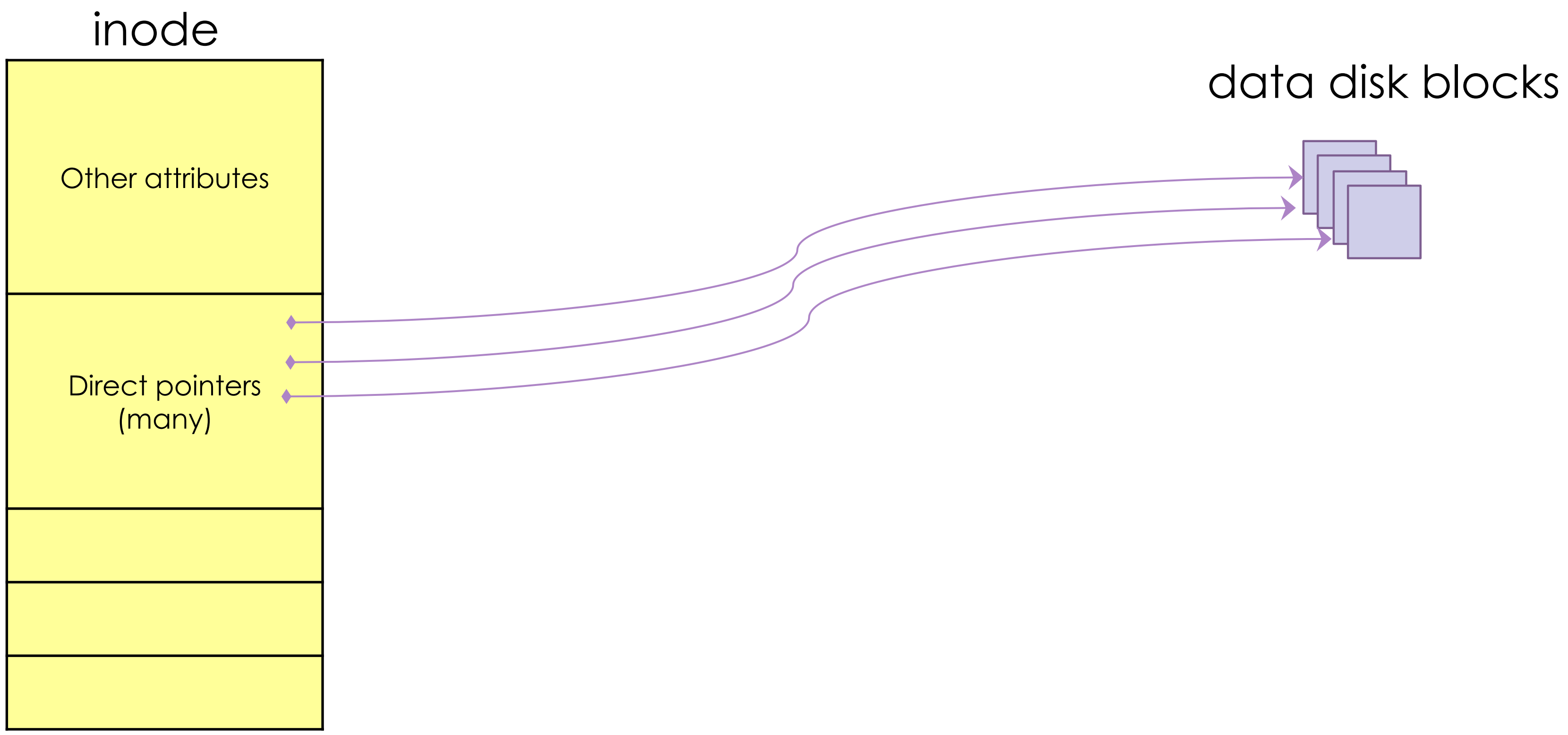
Inode (File Control Block)

- The most important on-disk structure of a file system
  - In Unix, this is called inode
  - In NTFS, this is called file record

- It stores virtually all of the information about a file, including its type, its size, ownerships, permissions, . . ., and location of file contents

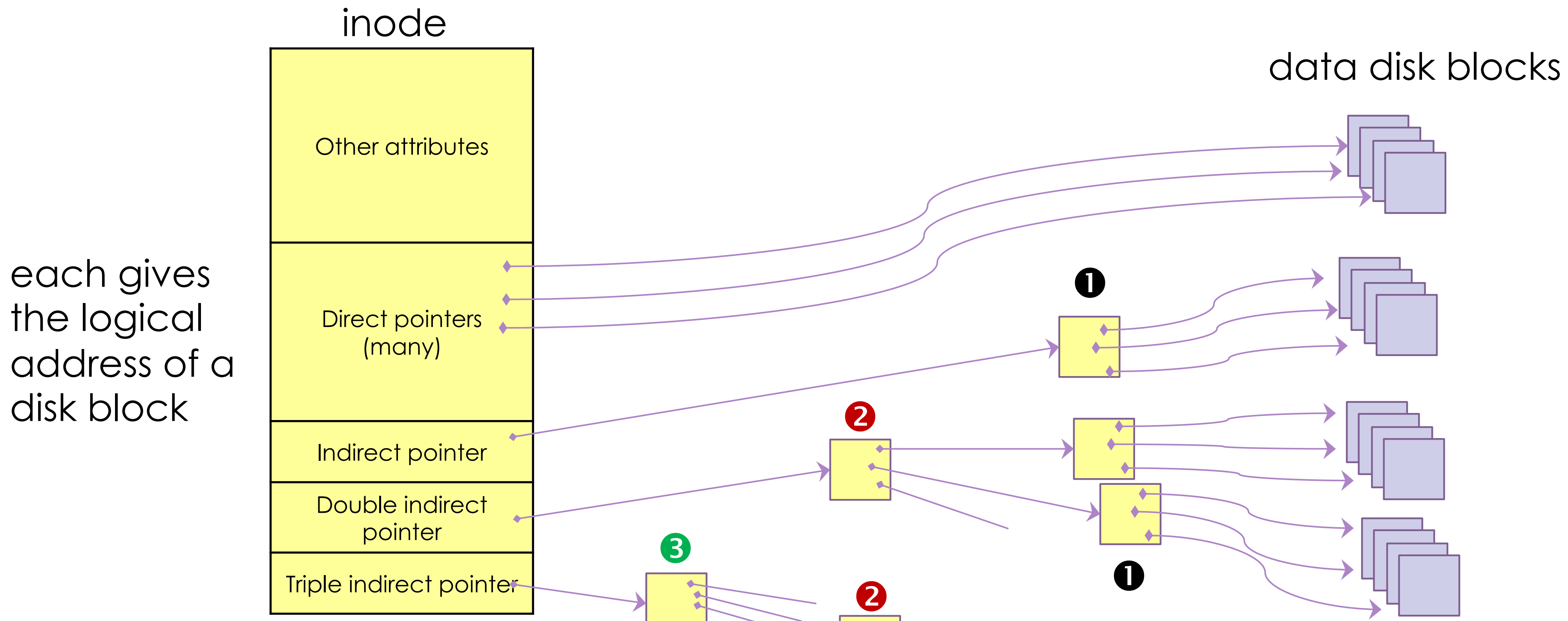
- One important design decision of the inode is that how can the inode keep track on the data blocks of a file
  - The data content of the file is stored in data disk block(s)
  - Thus, inode needs to have some way to **tell us** which **data blocks** are associated to this file in the disk
  - The design decision may have impact on the max size a single file can be in the system

Inode – Multi-level Index



each gives the logical address of a disk block

Inode – Multi-level Index



❶ Indirect block - a disk block full of direct pointers

❷ Double indirect block - a disk block full of indirect pointers

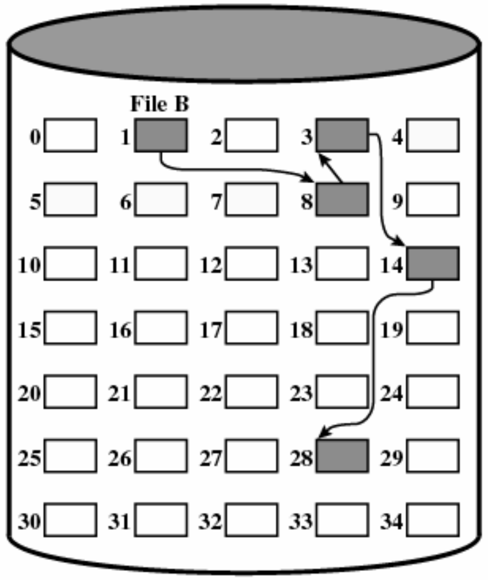
❸ Triple indirect block - a disk block full of double indirect pointers

Inode – Multi-level Index

- Disadvantage
  - Used up some space in the inode to store the pointers
  - Disk blocks used by a file may be scattered all over the disk
- Advantages
  - Can support a very large file size
  - Brings all pointers together (centralized) into the inode; facilitate searching for a particular data block
    - Most files are small; with a small number of direct pointers (typically 12), all data blocks are located by the direct pointers
- Examples: ext2fs, ext3fs, ext4fs, and NTFS

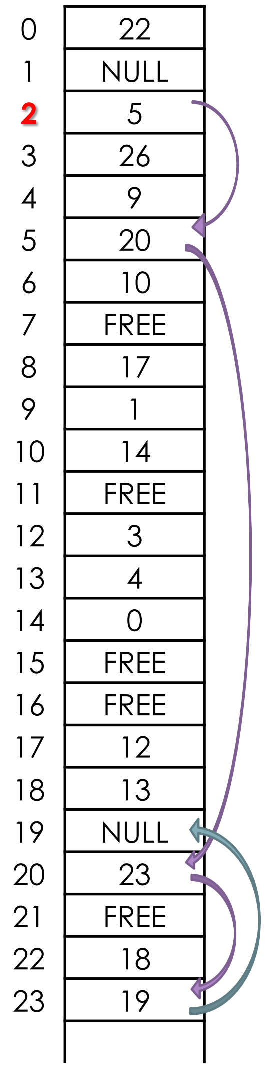
Other Approaches

- Linked-list approach
  - Inside the inode, instead of having multiple direct pointers, just only one pointer which points to the first data block of the file in the disk
  - For a larger file, each data block contains a pointer to the next block in the chain
  - Disadvantages
    - To locate a data item in the file
      - The chain must be **searched from the beginning** to locate the data; search process can be slow as **block-to-block seeks** occur
    - Reliability issue – if the linked list is broken



Other Approaches

- Tabular approach
  - Uses a table to keep track of the allocation of data blocks in the file system
  - Directory entry or inode records where is the first data block of a file
    - This block number is used as an index into the **block allocation table** to find the location of the next block.
    - If current block is the file's last block, then its table entry is null
  - Advantage
    - Pointers that locate file data blocks are centralized
      - The table can be cached so that searching can be traversed quickly; this improves access times
  - Disadvantage
    - For large disks, the block allocation table can become quite large
  - Example: Microsoft's FAT file system



0	22
1	NULL
2	5
3	26
4	9
5	20
6	10
7	FREE
8	17
9	1
10	14
11	FREE
12	3
13	4
14	0
15	FREE
16	FREE
17	12
18	13
19	NULL
20	23
21	FREE
22	18
23	19

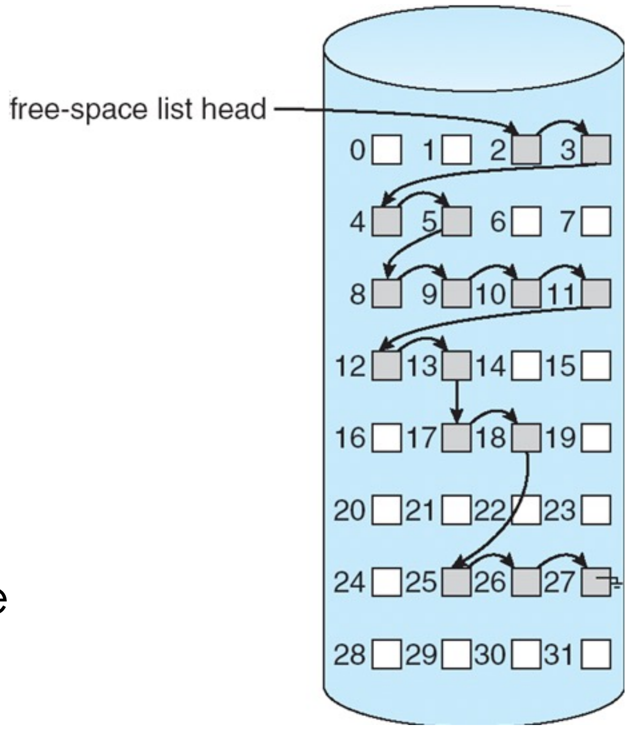
Free Space Management

- Data Block Bitmap is
  - A bitmap that tells us which data disk blocks are free or in-use
  - Each bit represents a data block in disk
    - If the file system has x data blocks, it needs at least x bits = x/8 bytes
      - Example: block size = 2^{12} bytes and disk size = 2^{30} bytes (1 GiB); x = 2^{18} bits (or 32 KiB)

  - Advantage
    - Can determine if contiguous blocks are available at certain locations on disk
  - Disadvantage
    - May need to search the entire bitmap to find a free block

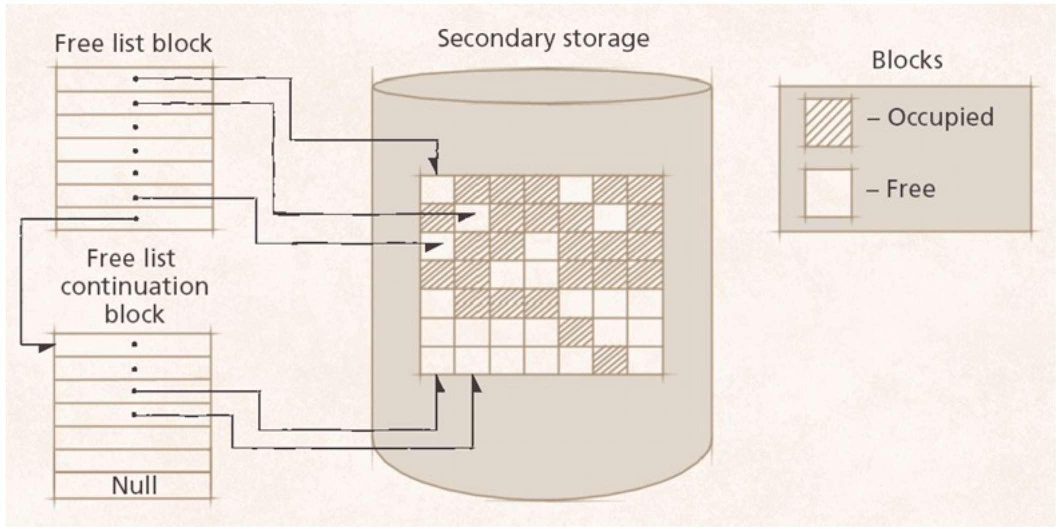
Free Space Management

- Free linked-list
  - Keep a pointer points to the first free block
  - Each block contains a pointer to the next free block, and so on
  - Blocks are allocated from the beginning of the free list
  - Newly freed blocks are appended to the end of the list
  - Disadvantage
    - Every disk block allocation results in a disk read to locate the next free block for updating the pointer



Free Space Management

- Free block-list
  - A disk block stores addresses of n-1 free blocks; the last entry stores a pointer to the next block in the free block-list
  - Faster to find a large number of free blocks
  - Advantage
    - Both free block list and free linked list have low overhead to perform free list maintenance operations
  - Disadvantage
    - Files are likely to be allocated in noncontiguous blocks



Caching and Buffering

- Imagine a file open operation to open an existing file:
  /home/c3230a/overview.txt,
  without touching on the data content
  - Assume the file system has been mounted with the superblock being loaded in only; everything else is still on the disk

- Every file open would require **at least two** reads (disk I/O) for every level in the directory hierarchy
  - One to read the inode of the directory in question, and at least one to read the data of the directory
    - 2 reads for "/"
    - 2 reads for "home"
    - 2 reads for "c3230a"
    - 1 read for the inode of "overview.txt"

Caching and Buffering

- Most file systems use physical memory to cache important blocks
  - If caching is used, the first open may generate a lot of I/O traffic
  - subsequent file opens of files in the directories along the path will mostly result in a cache hit, thus no I/O is needed
    - Both inode and data block of the directory are in cache

Caching and Buffering

- Cache does not benefit Writes as much as Reads
- (Write) Buffering
  - Batch writes by delaying writes: batch some updates into a smaller set of I/Os
    - E.g., A file created and then updated
  - Schedule the I/Os of buffered writes
  - Avoid some writes
    - E.g., a file created and then deleted

- Modern FS buffer writes in memory for 5~30 seconds

Access Control and Protection

- Through the user access control procedure, user is assigned a unique user ID (and group ID)

- The most common approach to file protection is to make access dependent on the identity of the user
  - Each file has an associated **access (-control) list**
    - Mode of access: Read, Write, Execute
  - Many systems recognize three classes of users: Owner, Group, Public
  - For a particular file or subdirectory, define an appropriate access control
    - e.g.,

		RWX
owner access	7	1 1 1
group access	6	1 1 0
public access	1	0 0 1

  - Can be stored in the inode

Access Control and Protection

- Upon receiving a request from a process to open the file, access permission is checked and all associated actions are performed
- If no access right, operation will be denied
- All information is recorded in a new entry in the open file table; returns the index (indirectly via the file descriptor) of this file in the open file table to user
- The process could only access the file via this index
  - The right to access must still **be checked** on every access
  - The open-file-table entry has a capability only for the allowed operations
    - For example, if the file is opened for read only, an attempt to have write operation on the file will be denied even you have the write access right

Integrity Protection

- System crash can result in data inconsistencies (files and file system)
  - A typical file creation may involve following actions
    - Update the directory's inode and data block, inode bitmap, data block bitmap, free counts, new inode, new data block, and other control structures
      - If a system failure occurs during the file creation operation, file data (and metadata) may be left in an inconsistent state

- Routinely backup is the most effective protection scheme
  - e.g. superblocks are duplicated and stored as backup
  - e.g. physical backup of the whole disk, using dd
  - e.g., logical backups
    - Store file system data and its logical structure
    - Inspect the directory structure to determine which files need to be backed up, then write these files to a backup device in a common, often compressed, archival format
    - Example: Unix command - tar
    - Incremental backups are logical backups that store only file system data that has changed since the previous backup

Integrity Protection

- Consistency checking
  - Compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
  - Example: Unix command: *fsck* or DOS command: *chkdsk*
    - FSCK
      - checking superblock to find suspect corruption
      - scan all inodes' state
      - scan all inodes' direct and indirect block pointers to build a correct Free block bitmap
      - scan all inodes to build the Free inode bitmap

Journaling File Systems

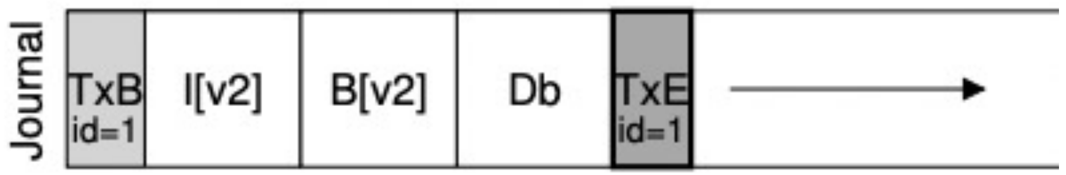
- Examples: NTFS and ext3fs (and ext4fs)
- Perform all file system operations as logged transactions to ensure that they do not leave the system in an inconsistent state
  - Each update to the file system is considered as a transaction and is labeled by a sequence #
  - All transactions are first written to a log – a pre-allocated space in the disk
    - Physical logging - putting the exact physical contents of the update in the log
  - A transaction is considered committed once it is written to the log
  - However, the file system may not yet be updated
  - Only when the update has been written to disk, we called this transaction has completed, and record that in the log so that the space can be reused

Journaling File Systems

- If an error occurs, when the system reboots, the file system recovery process will scan the log and look for not completed transactions; these transactions are thus replayed (in order).
- Two modes of operation
  - Data Journaling: Log all updates, including data blocks are recorded
  - Metadata Journaling: Only metadata updates are logged

Journaling File Systems

Data Journaling



Journal TxB	Journal Contents (metadata)	Journal Contents (data)	TxE	File System Metadata	File System Data
issue	issue	issue			
complete					
	complete				
		complete			
			issue		
			complete		
				issue	issue
					complete
				complete	

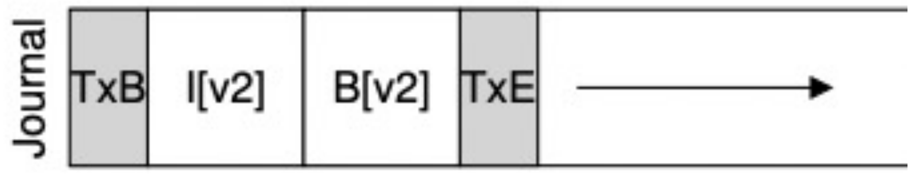
Journal write

Journal commit

Checkpoint

Free

Metadata Journaling



Journal TxB	Journal Contents (metadata)	TxE	File System Metadata	File System Data
issue	issue			issue
				complete
complete				
	complete			
		issue		
		complete		
			issue	
			complete	

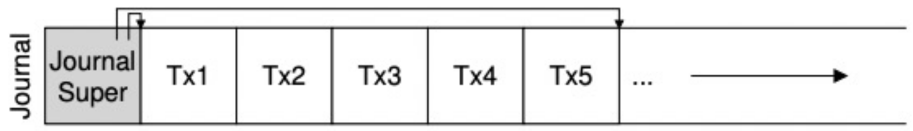
Data write

Journal write

Journal commit

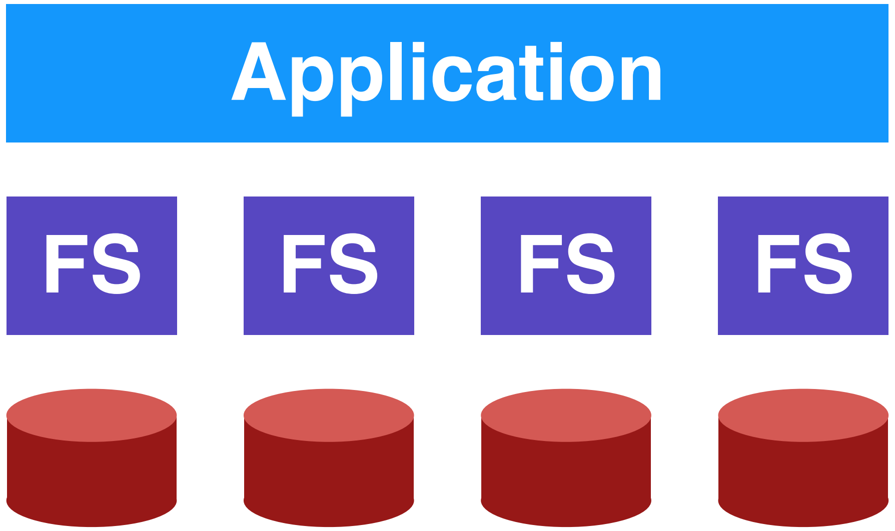
Checkpoint

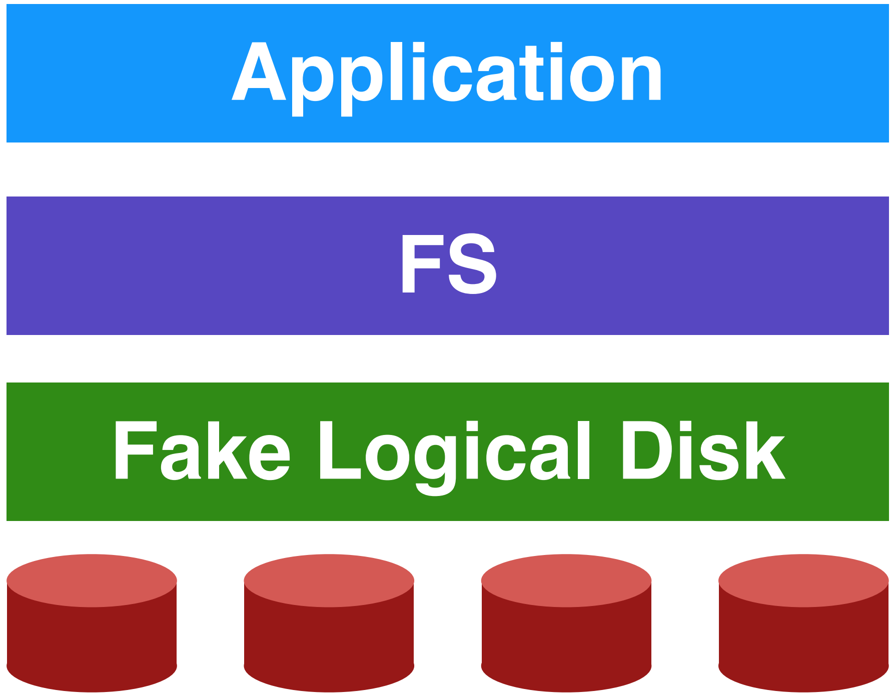
Free



When Having Many Disks

- Challenge: most file systems work on only one disk



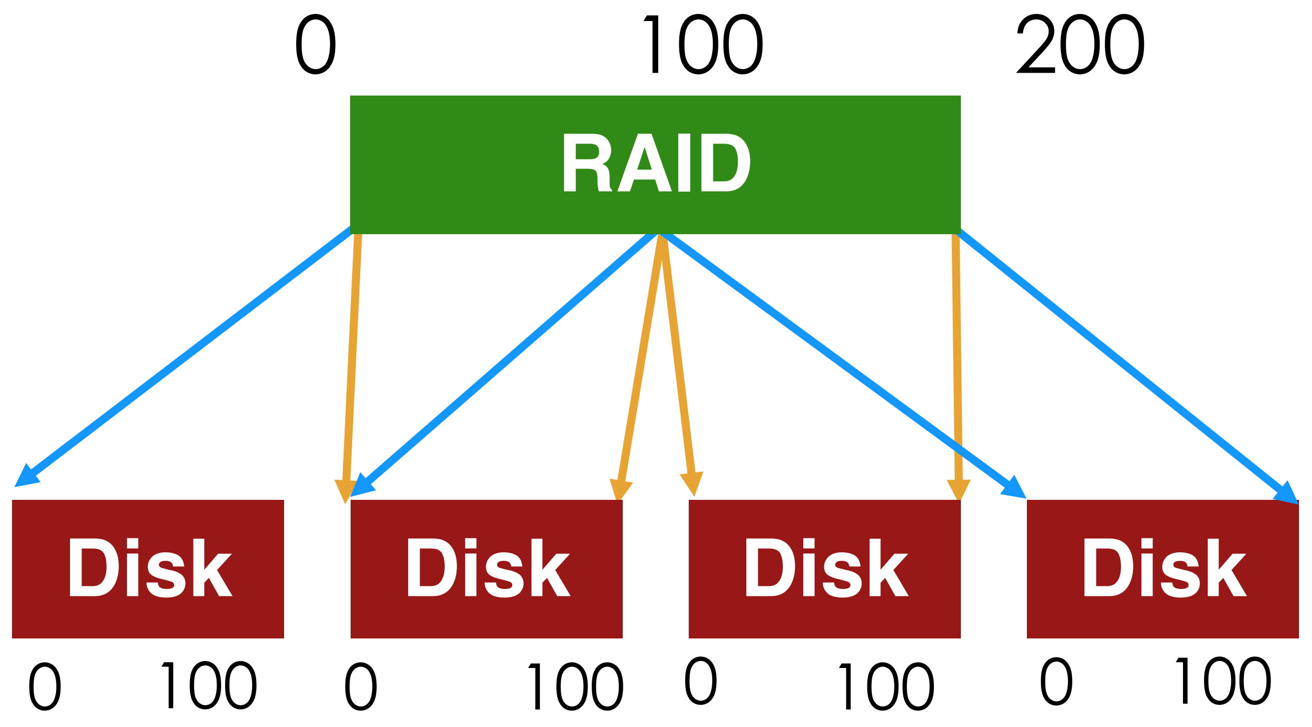


JBOD: **J**ust a **B**unch **O**f **D**isks

RAID: **R**edundant **A**rray of **I**nexpensive **D**isks

RAID: General Strategy

- Mapping & Redundancy: Build reliable, fast, large disk from (many) smaller ones.



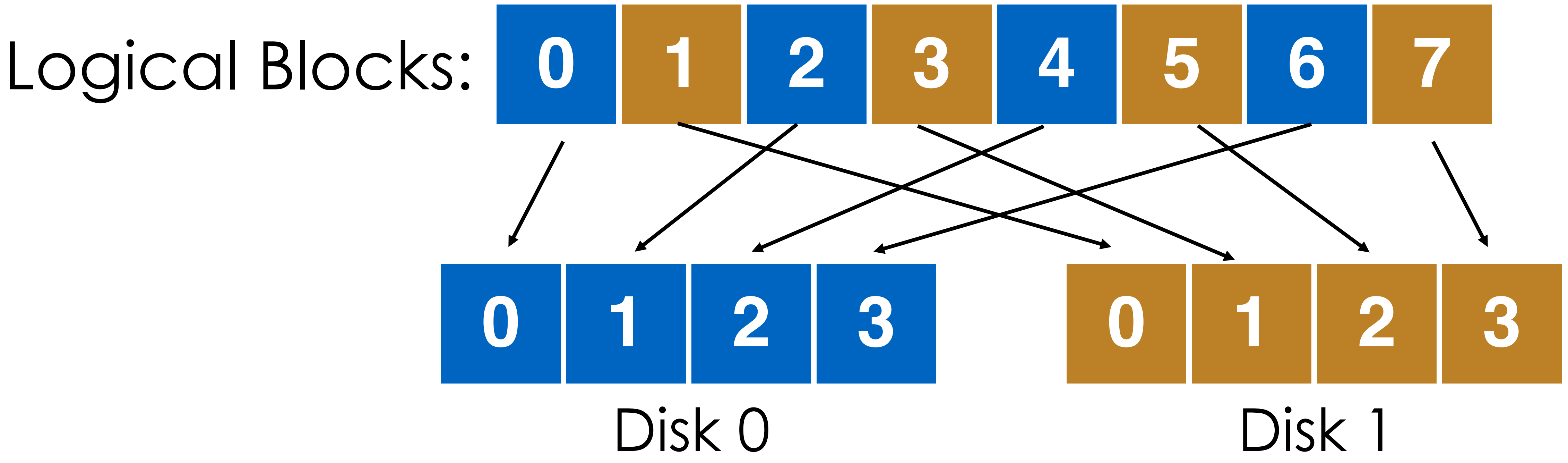
Capacity: how much space can apps use?

Reliability: how many disks can we safely lose? (assume fail stop!)

Performance: how long does each workload take?

N := number of disks
C := capacity of 1 disk
S := sequential throughput of 1 disk
R := random throughput of 1 disk
D := latency of one small I/O operation

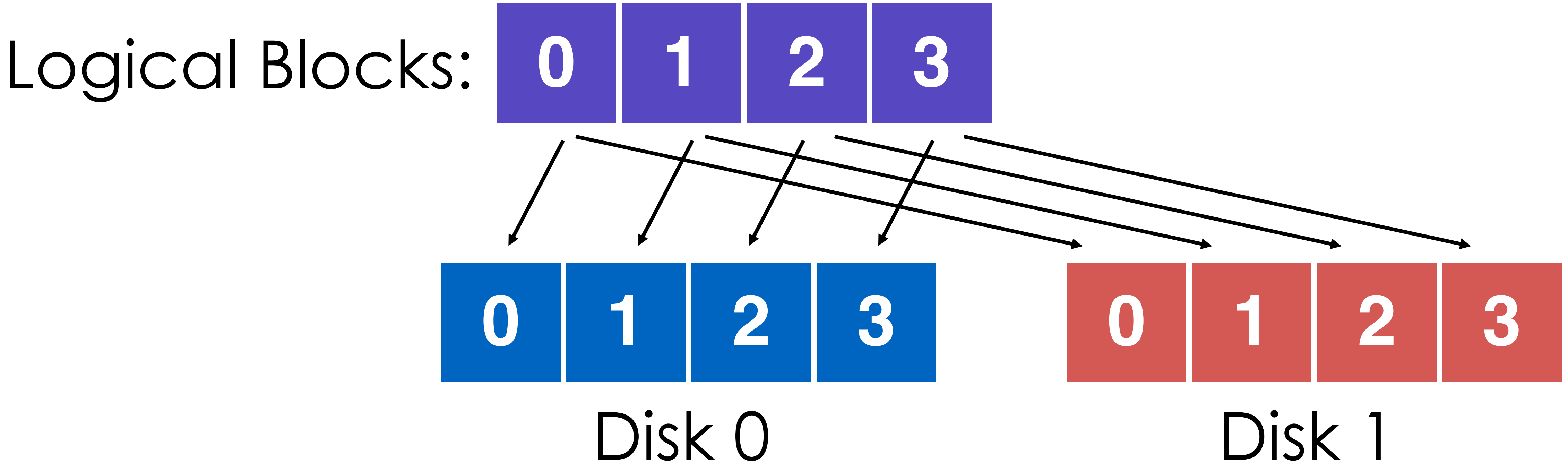
RAID-0: Stripping



Disk 0	Disk 1	Disk 2	Disk 3
0	1	2	3
4	5	6	7
8	9	10	11

Disk 0	Disk 1	Disk 2	Disk 3
0	3	6	9
1	4	7	10
2	5	8	11

RAID-1: Mirroring



Disk 0	Disk 1	Disk 2	Disk 3
0	0	1	1
2	2	3	3
4	4	5	5

RAID-4: Parity

- Using parity disk
  - small-write problem

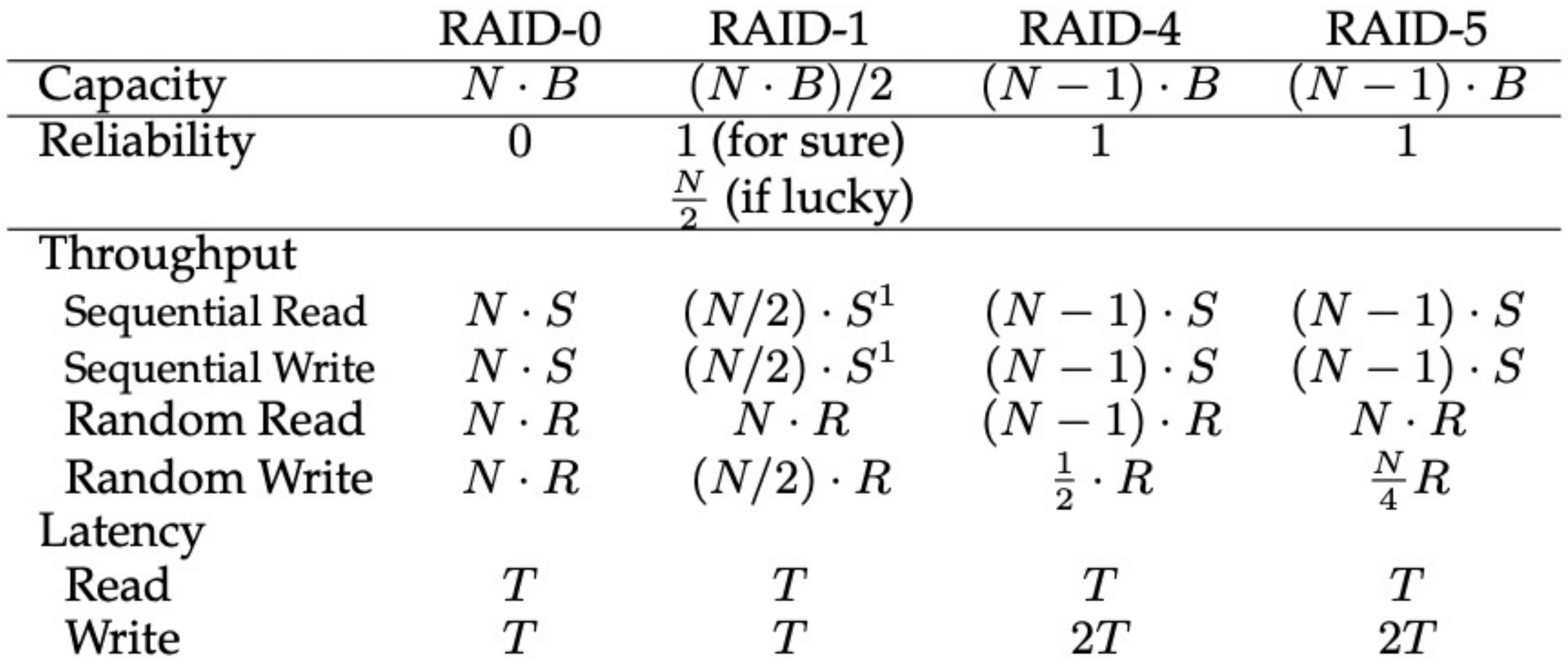
Disk 0	Disk 1	Disk 2	Disk 3	Disk 3
0	1	2	3	P0
4	5	6	7	P1
8	9	10	11	P2
12	13	14	15	P3

RAID-5: Rotate parity

- Identical to RAID-4, but rotate parity disk
  - Overcome small-write problem

Disk 0	Disk 1	Disk 2	Disk 3	Disk 3
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14

RAID Level Comparison



	RAID-0	RAID-1	RAID-4	RAID-5
Capacity	$N \cdot B$	$(N \cdot B)/2$	$(N-1) \cdot B$	$(N-1) \cdot B$
Reliability	0	1 (for sure) $\frac{N}{2}$ (if lucky)	1	1
Throughput				
Sequential Read	$N \cdot S$	$(N/2) \cdot S^1$	$(N-1) \cdot S$	$(N-1) \cdot S$
Sequential Write	$N \cdot S$	$(N/2) \cdot S^1$	$(N-1) \cdot S$	$(N-1) \cdot S$
Random Read	$N \cdot R$	$N \cdot R$	$(N-1) \cdot R$	$N \cdot R$
Random Write	$N \cdot R$	$(N/2) \cdot R$	$\frac{1}{2} \cdot R$	$\frac{N}{4} R$
Latency				
Read	T	T	T	T
Write	T	T	$2T$	$2T$

Figure 38.8: **RAID Capacity, Reliability, and Performance**

N := number of disks
B := capacity (#blocks) of 1 disk
S := sequential throughput of 1 disk
R := random throughput of 1 disk

- RAID-0 is always fastest and has best capacity (but at cost of reliability)
- RAID-5 better than RAID-1 for sequential workloads
- RAID-1 better than RAID-5 for random workloads

AGAIN, TRADE-OFF!

Summary

- Explain what is metadata of the file system
- Discuss and compare various file storage allocation schemes
  - Linked-list allocation
  - Tabular allocation
  - Indexed allocation
- Discuss and compare various free space management schemes
  - Free linked list
  - Free block list
  - Bitmap
- Describe how modern OSs provide access control and data integrity

Operating Systems

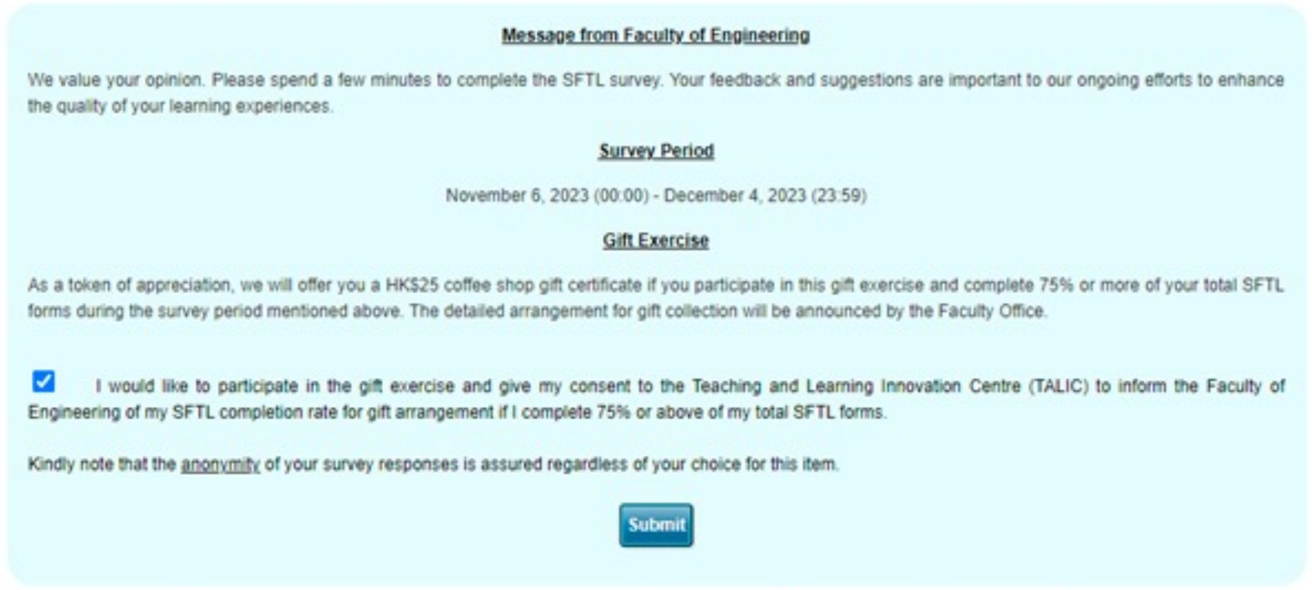
- **Virtualization**
  - CPU Virtualization
    - Process Abstract
      - Address space
      - Process states
      - Process control block
      - Process operations API
      - Signals
    - Limited Direct Execution
      - System calls
      - Context switch
      - Interrupts
    - Scheduling
      - Scheduling metrics
      - FIFO, SJF, HRRN, STCF, RR, MLFQ
      - Multi-core scheduling, Linux CFS
  - Memory Virtualization
    - Address space
    - Address translation: dynamic relocation
    - Segmentation
    - Paging
    - TLB
    - Multi-level paging
    - Inverted page table
    - Swap space
    - Page replacement policy: FIFO, LFR, LRU, Clock
    - Thrashing

- **Concurrency**
  - Thread
    - POSIX threads (pthreads)
    - Race conditions, critical sections, mutual exclusion, atomic operations, synchronization
  - Locks
    - Atomic instructions: test-and-set, compare-and-swap
    - Mutex locks
  - Condition Variables
    - Pthread CVs
    - Producer-Consumer problem
  - Semaphores
    - Binary Semaphores
    - Counting Semaphores
    - Ordering
    - Readers-Writers problem
  - Deadlock
    - Dining philosophers' problem
    - Four necessary conditions
    - Deadlock prevention, avoidance, detection&recovery

- **Persistence**
  - I/O devices (HDD, SSD)
  - Files and Directories
    - Inode
    - File descriptor
    - Hard/Symbolic links
  - File System Implementation
    - On-disk data structure
      - Superblock, Bitmap, Inodes, Data blocks
    - Free space management
      - Bitmap, linked-list, block-list
    - Caching and buffering
    - Access control and protection
    - Journaling file system
      - Data journaling
      - Metadata journaling
- ***Advanced Topics***

Student Feedback on Teaching and Learning (SFTL) Exercise

http://sftl.hku.hk/



Message from Faculty of Engineering

We value your opinion. Please spend a few minutes to complete the SFTL survey. Your feedback and suggestions are important to our ongoing efforts to enhance the quality of your learning experiences.

Survey Period

November 6, 2023 (00:00) - December 4, 2023 (23:59)

Gift Exercise

As a token of appreciation, we will offer you a HK$25 coffee shop gift certificate if you participate in this gift exercise and complete 75% or more of your total SFTL forms during the survey period mentioned above. The detailed arrangement for gift collection will be announced by the Faculty Office.

✓ I would like to participate in the gift exercise and give my consent to the Teaching and Learning Innovation Centre (TALIC) to inform the Faculty of Engineering of my SFTL completion rate for gift arrangement if I complete 75% or above of my total SFTL forms.

Kindly note that the anonymity of your survey responses is assured regardless of your choice for this item.

The Faculty of Engineering will give out a $25 Starbucks coupon to each UG student who has completed 75% or above of the total SFTL form.