Database Applications (15-415)

DBMS Internals- Part I

Lecture 09, February 12, 2014

Mohammad Hammoud

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

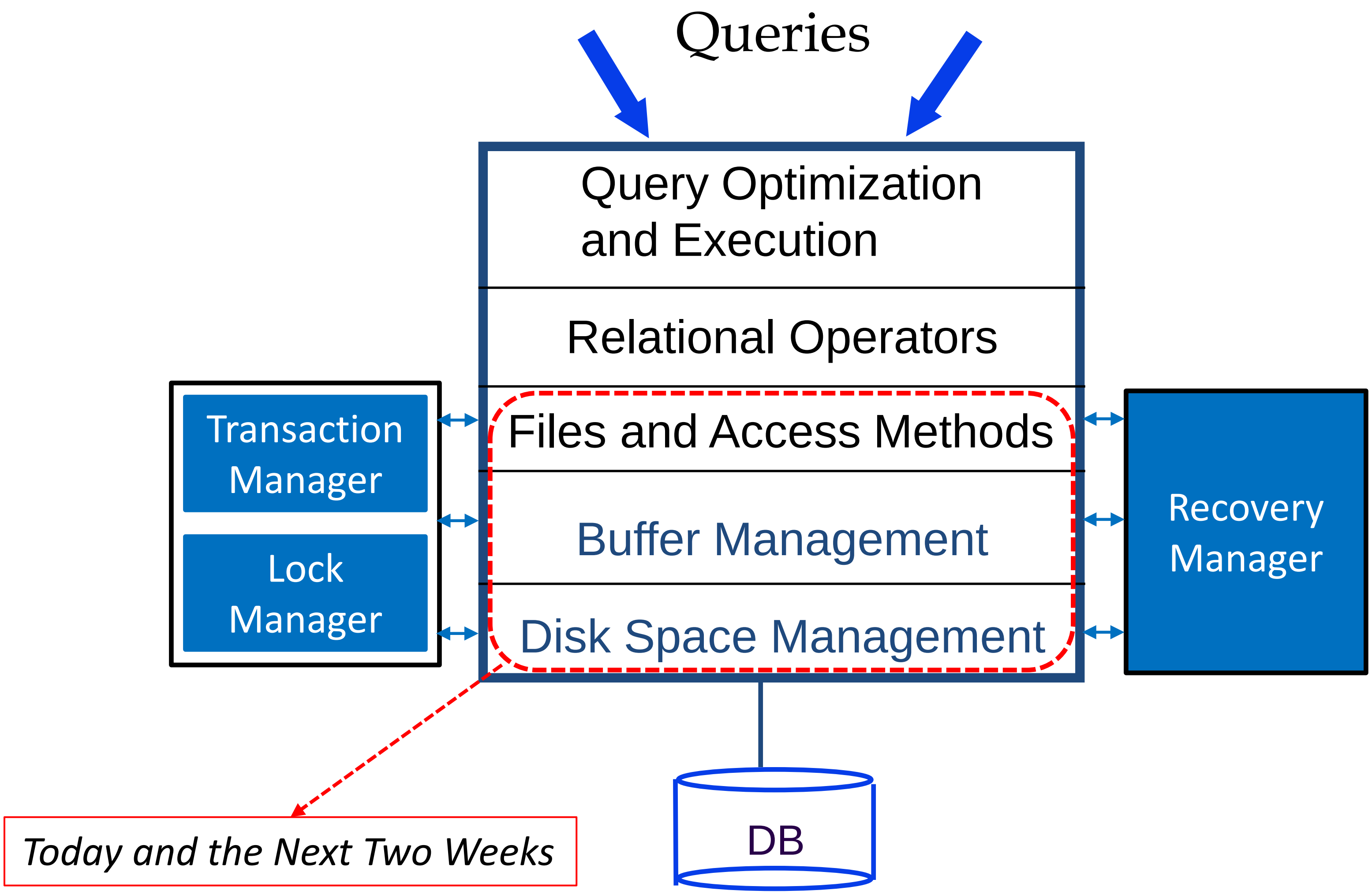
Today…

- Last Session:
  - Quiz I & a Brief Introduction on Disks
- Today's Session:
  - DBMS Internals- Part I
    - Disk Space Management
    - Buffer Management
- Announcements:
  - Quiz I grades are out
  - Project 1 is due on Feb 18 by midnight

Outline

- Where Do DBMSs Store Data? ✓
- Various Disk Organizations and Reliability and Performance Implications on DBMSs
- Disk Space Management
- Buffer Management

DBMS Layers

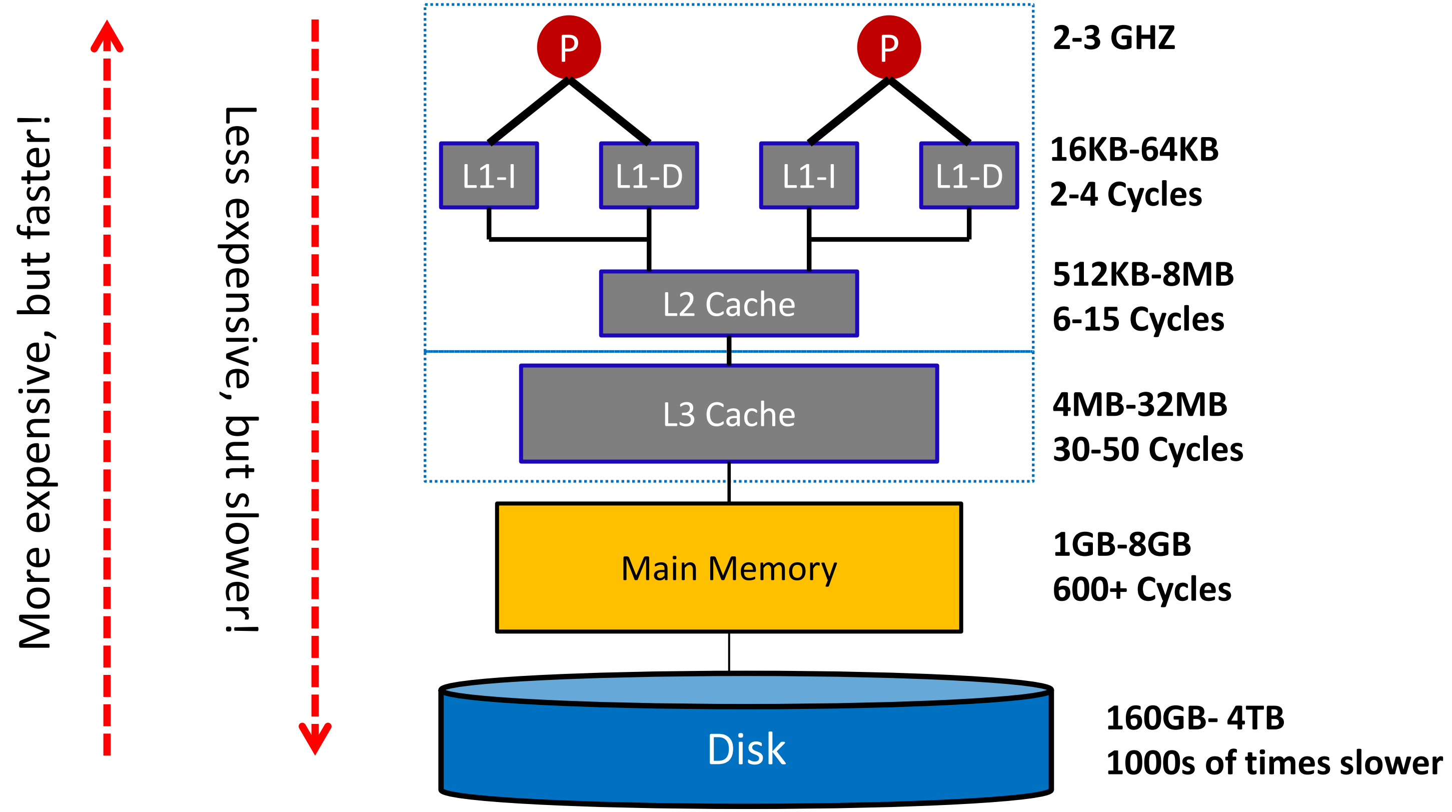


Queries
Query Optimization and Execution
Relational Operators
Files and Access Methods
Buffer Management
Disk Space Management
Transaction Manager
Lock Manager
Recovery Manager
DB
Today and the Next Two Weeks

The Memory Hierarchy

- Storage devices play an important role in database systems

- How systems arrange storage?



Where to Store Data?

- Where do DBMSs store information?
  - DBMSs store large amount of data (e.g., Big Data!)
  - Buying enough memory to store all data is prohibitively expensive (let alone that memories are *volatile*)
  - Thus, databases are usually stored on disks (or tapes for backups)

But, What Will Do With Memory?

- Data must be brought into memory to be processed!
  - READ: transfer data from disk to main memory (RAM)

I/O Time

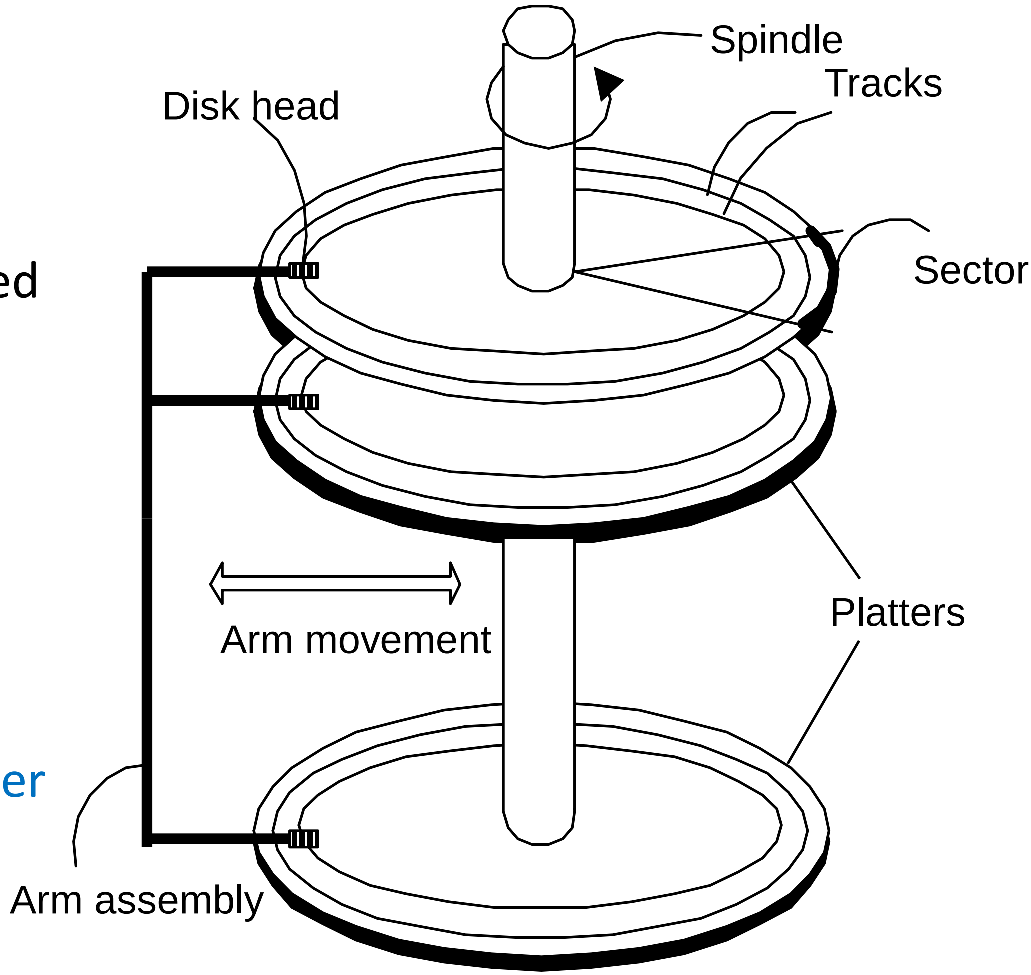
  - WRITE: transfer data from RAM to disk

- I/O time dominates the time taken for database operations!

- To minimize I/O time, it is necessary to store and locate data *strategically*

Magnetic Disks

- Data is stored in disk blocks
- Blocks are arranged in concentric rings called tracks
- Each track is divided into arcs called sectors (whose size is fixed)
- The block size is a multiple of sector size
- The set of all tracks with the same diameter is called cylinder
- To read/write data, the arm assembly is moved in or out to position a head on a desired track



Accessing a Disk Block

- What is I/O time?
  - The time to move the disk heads to the track on which a desired block is located
  - The waiting time for the desired block to rotate under the disk head
  - The time to actually read or write the data in the block once the head is positioned

Accessing a Disk Block

- What is I/O time?
  - Seek Time
  - Rotational Time
  - Transfer Time
- I/O time = seek time + rotational time + transfer time

Implications on DBMSs

- Seek time and rotational delay dominate!
- Key to lower I/O cost: reduce seek/rotation delays!
- How to minimize seek and rotational delays?
  - Blocks on same track, followed by
  - Blocks on same cylinder, followed by
  - Blocks on adjacent cylinder
  - Hence, ***sequential*** arrangement of blocks in a file is a big win!

More on that later…

Outline

- Where Do DBMSs Store Data?
- Various Disk Organizations and Reliability and Performance Implications on DBMSs ✓
- Disk Space Management
- Buffer Management

Many Disks vs. One Disk

- Although disks provide cheap, non-volatile storage for DBMSs, they are usually bottlenecks for DBMSs
  - Reliability
  - Performance

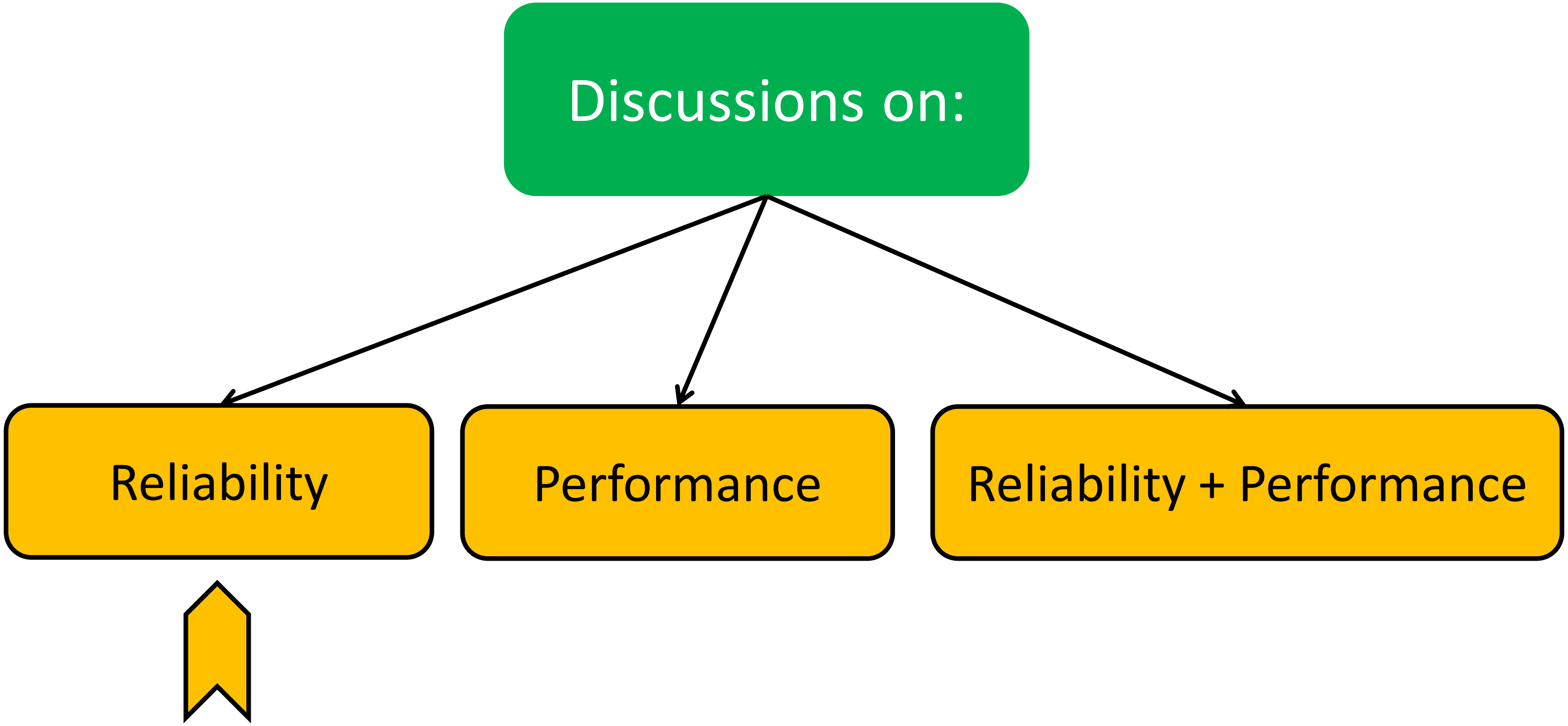
- How about adopting multiple disks?
  1. More data can be held as opposed to one disk
  2. Data can be stored redundantly; hence, if one disk fails, data can be found on another
  3. Data can be accessed concurrently

Many Disks vs. One Disk

- Although disks provide cheap, non-volatile storage for DBMSs, they are usually bottlenecks for DBMSs
  - Reliability
  - Performance

- How about adopting multiple disks?
  1. More data can be held as opposed to one disk — Capacity!
  2. Data can be stored redundantly; hence, if one disk fails, data can be found on another — Reliability!
  3. Data can be accessed concurrently — Performance!

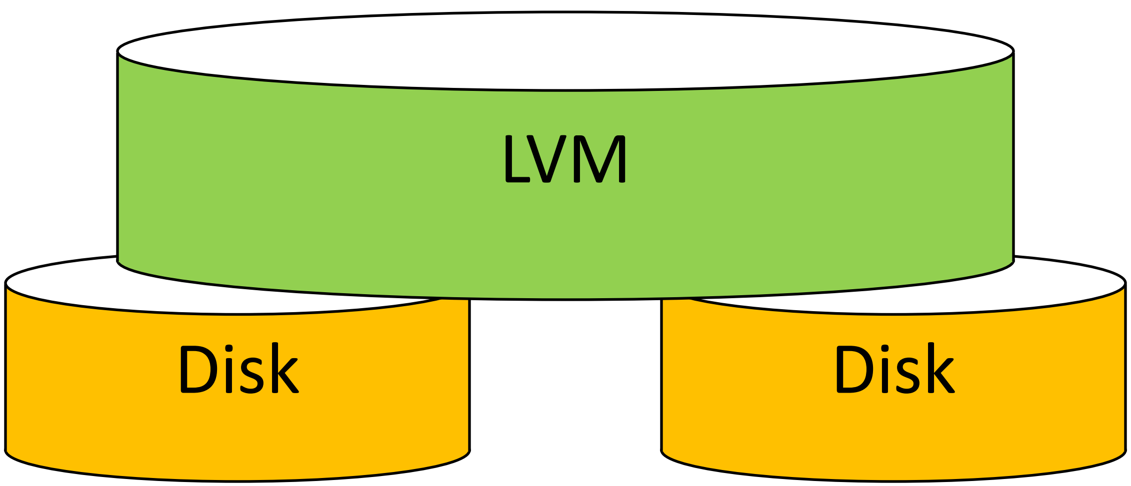
Multiple Disks



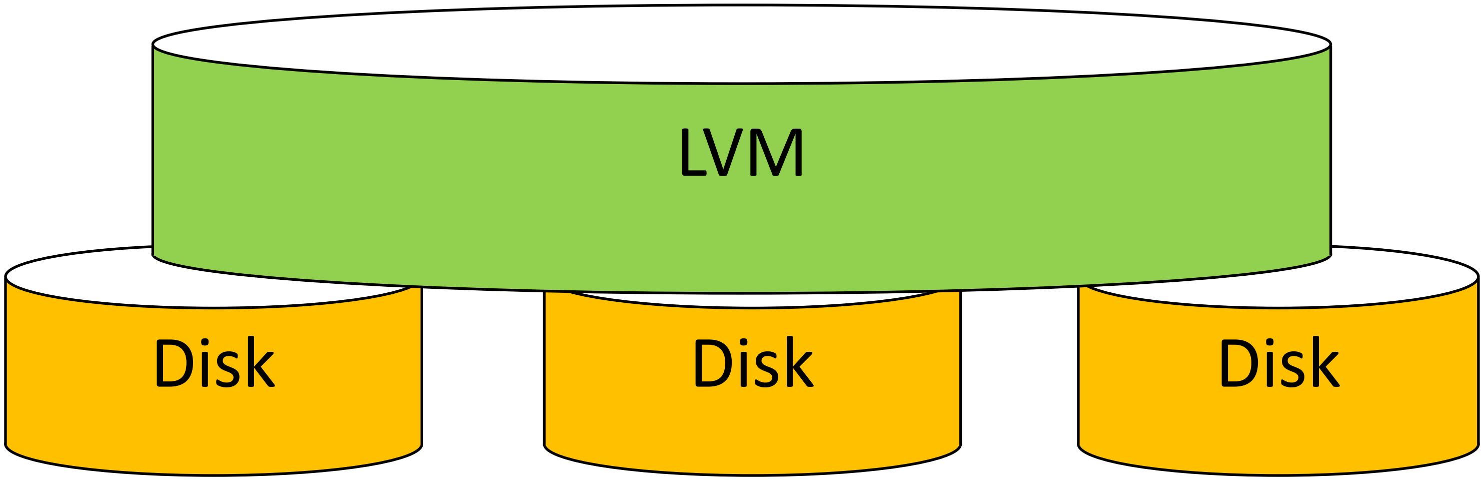
Logical Volume Managers (LVMs)

- But, disk addresses used within a file system are assumed to refer to one particular disk (or sub-disk)

- What about providing an abstraction that makes a number of disks *appear* as one disk?

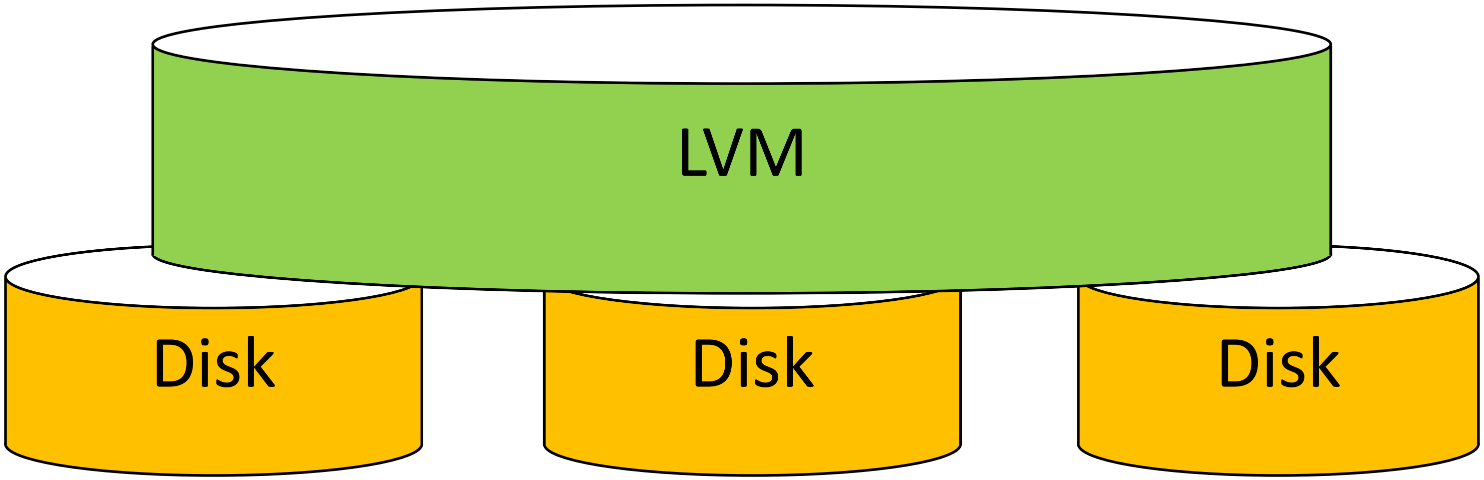


Logical Volume Managers (LVMs)



- What can LVMs do?
  - Spanning:
    - LVM transparently maps a *larger* address space to <u>different</u> disks
  - Mirroring:
    - Each disk can hold a separate, identical copy of data
    - LVM directs writes to the same block address on each disk
    - LVM directs a read to *any* disk (e.g., to the less busy one)

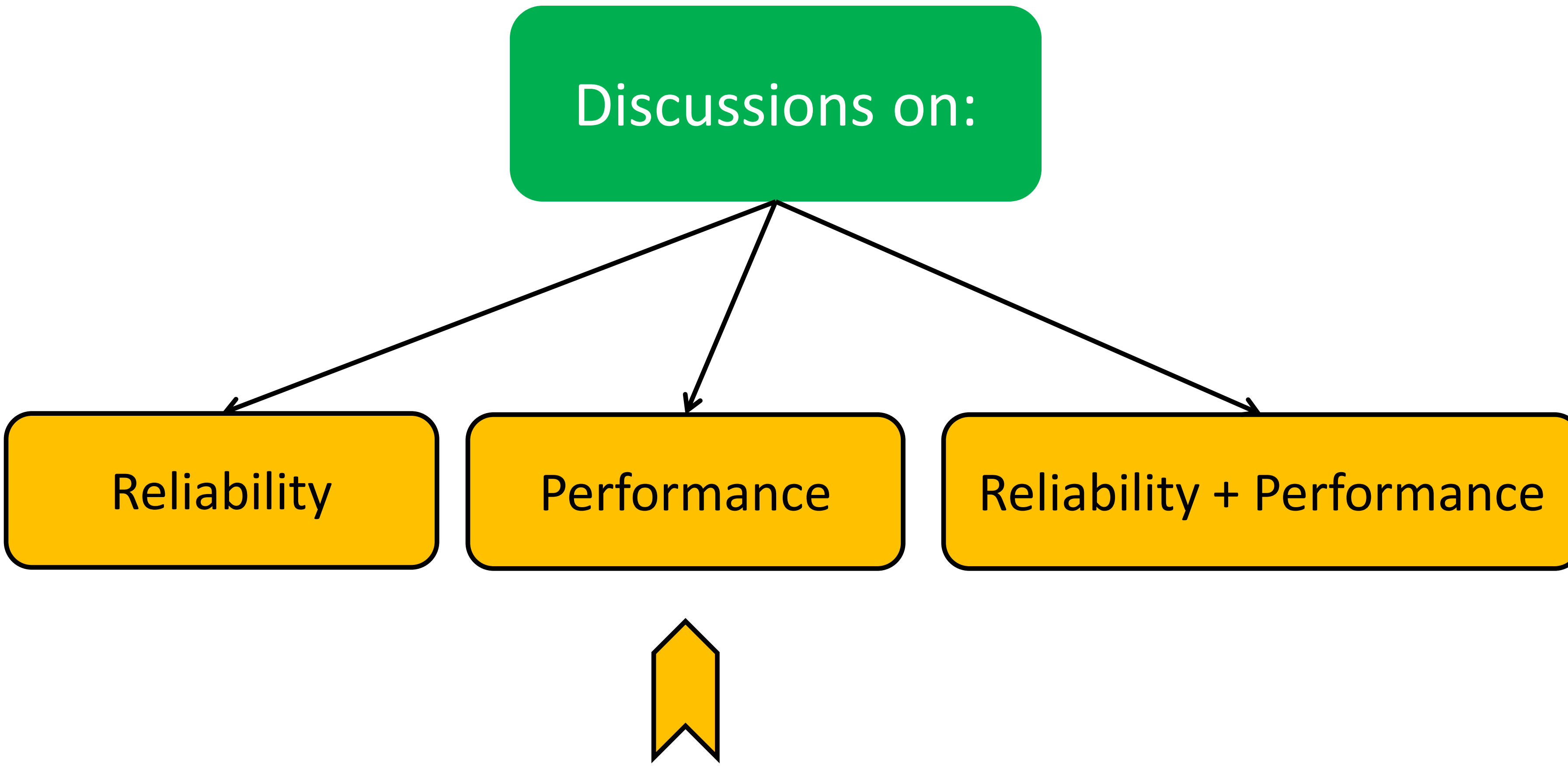
Logical Volume Managers (LVMs)



- What can LVMs do?
  - Spanning:
    - LVM transparently maps a *larger* address space to <u>different</u> disks
  - Mirroring:
    - Each disk can hold a separate, identical copy of data
    - LVM directs writes to the same block address on each disk
    - LVM directs a read to *any* disk (e.g., to the less busy one)

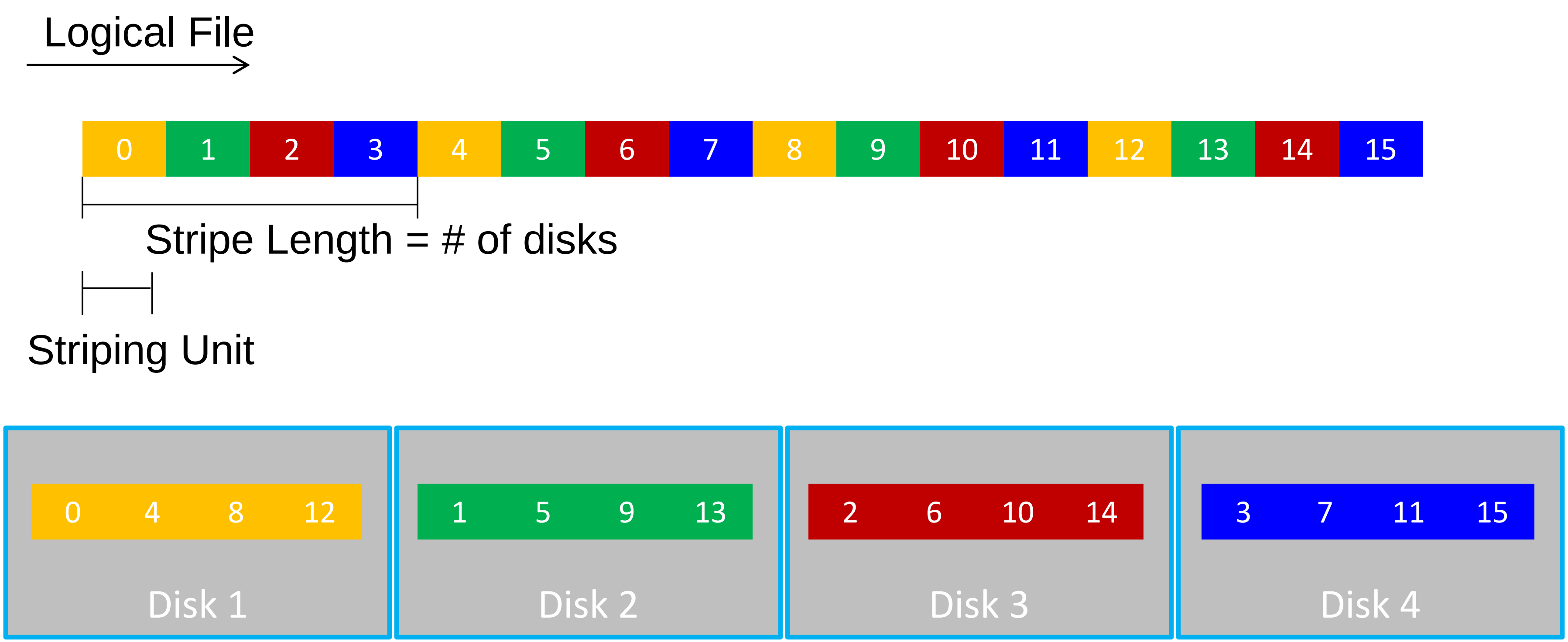
Mainly Provides Redundancy!

Multiple Disks



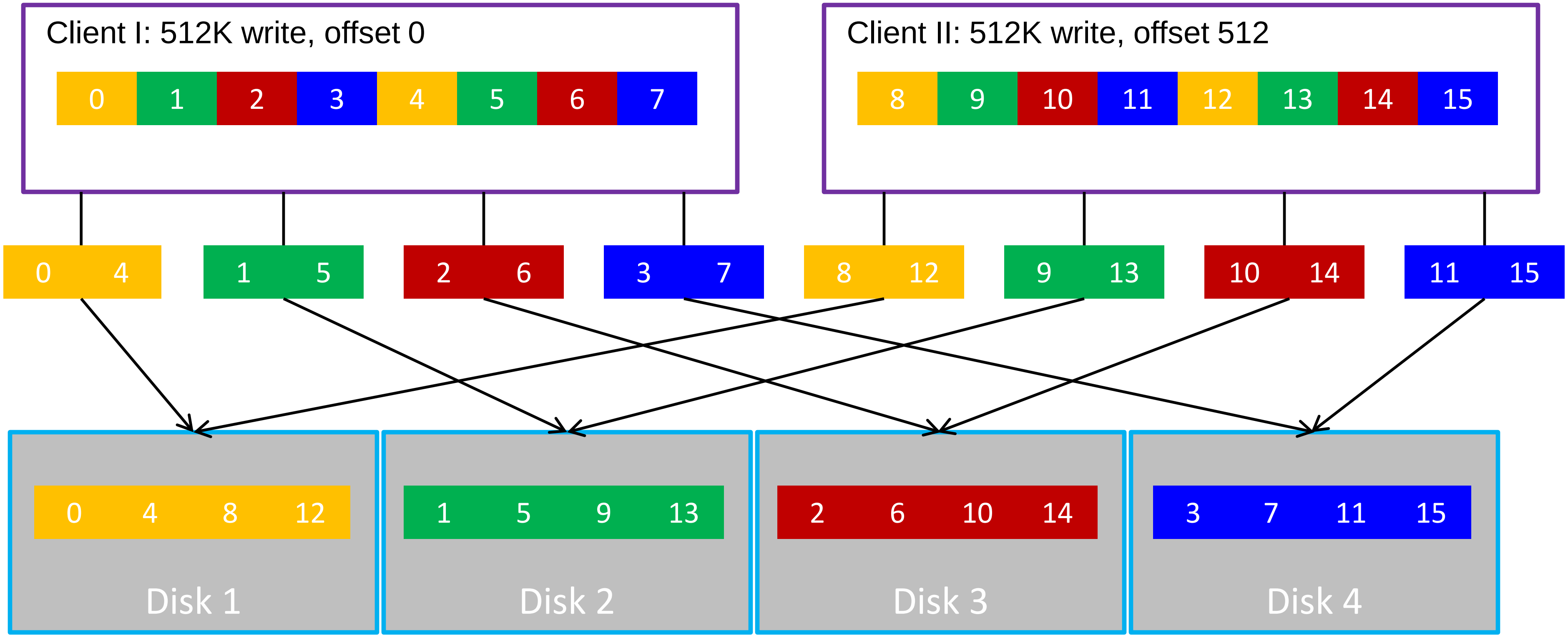
Data Striping

- To achieve parallel accesses, we can use a technique called data striping



Data Striping

- To achieve parallel accesses, we can use a technique called data striping



Data Striping

	Disk 1	Disk 2	Disk 3	Disk 4
Stripe 1	Unit 1	Unit 2	Unit 3	Unit 4
Stripe 2	Unit 5	Unit 6	Unit 7	Unit 8
Stripe 3	Unit 9	Unit 10	Unit 11	Unit 12
Stripe 4	Unit 13	Unit 14	Unit 15	Unit 16
Stripe 5	Unit 17	Unit 18	Unit 19	Unit 20

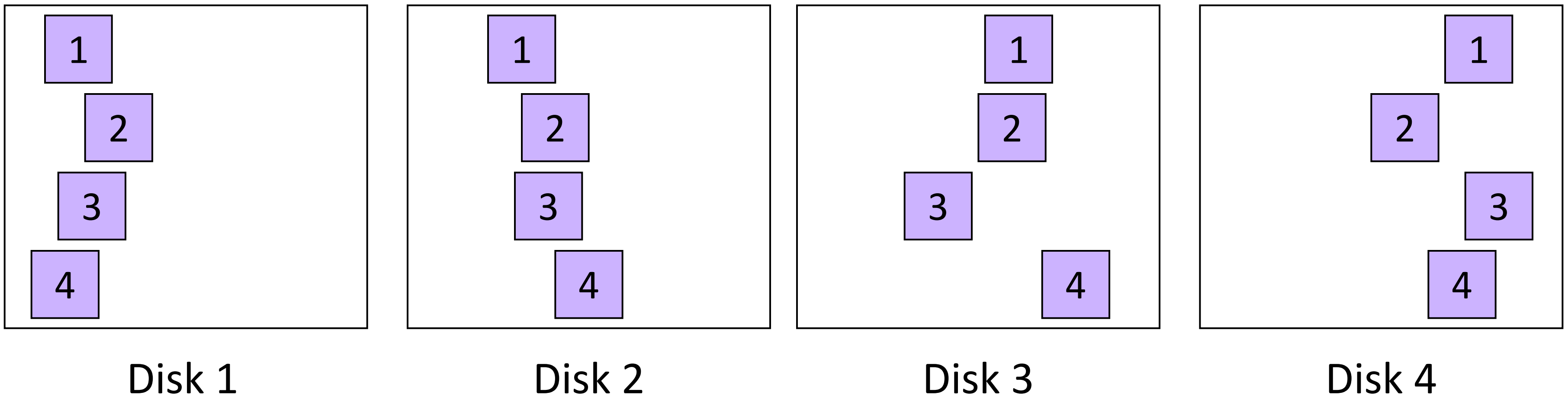
Each stripe is written across all disks *at once*

Typically, a unit is either:

- A bit ➔ **Bit Interleaving**
- A byte ➔ **Byte Interleaving**
- A block ➔ **Block Interleaving**

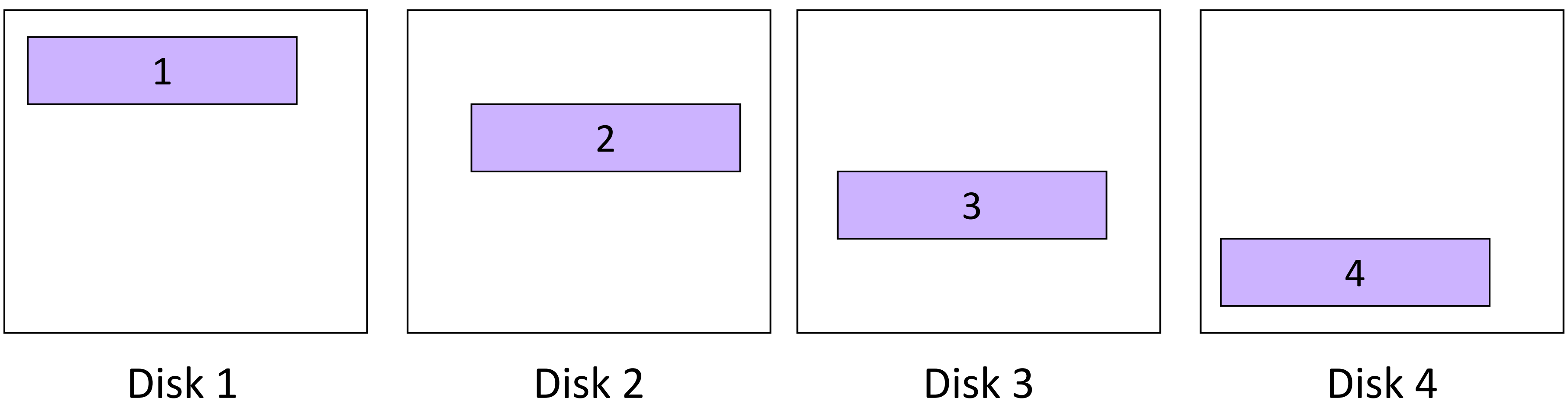
Striping Unit Values: Tradeoffs

- Small striping unit values
  - Higher parallelism (+)
  - Smaller amount of data to transfer (+)
  - Increased seek and rotational delays (-)



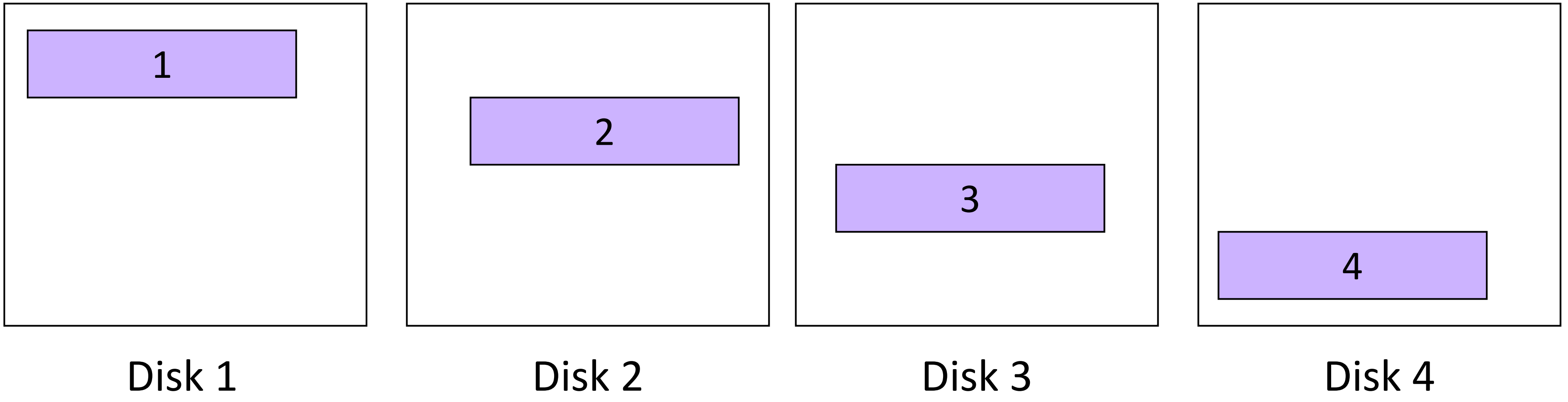
Striping Unit Values: Tradeoffs

- Large striping unit values
  - Lower parallelism (-)
  - Larger amount of data to transfer (-)
  - Decreased seek and rotational delays (+)
  - A request can be handled completely on a separate disk! (- or +)
  - But, multiple requests could be satisfied at once! (+)

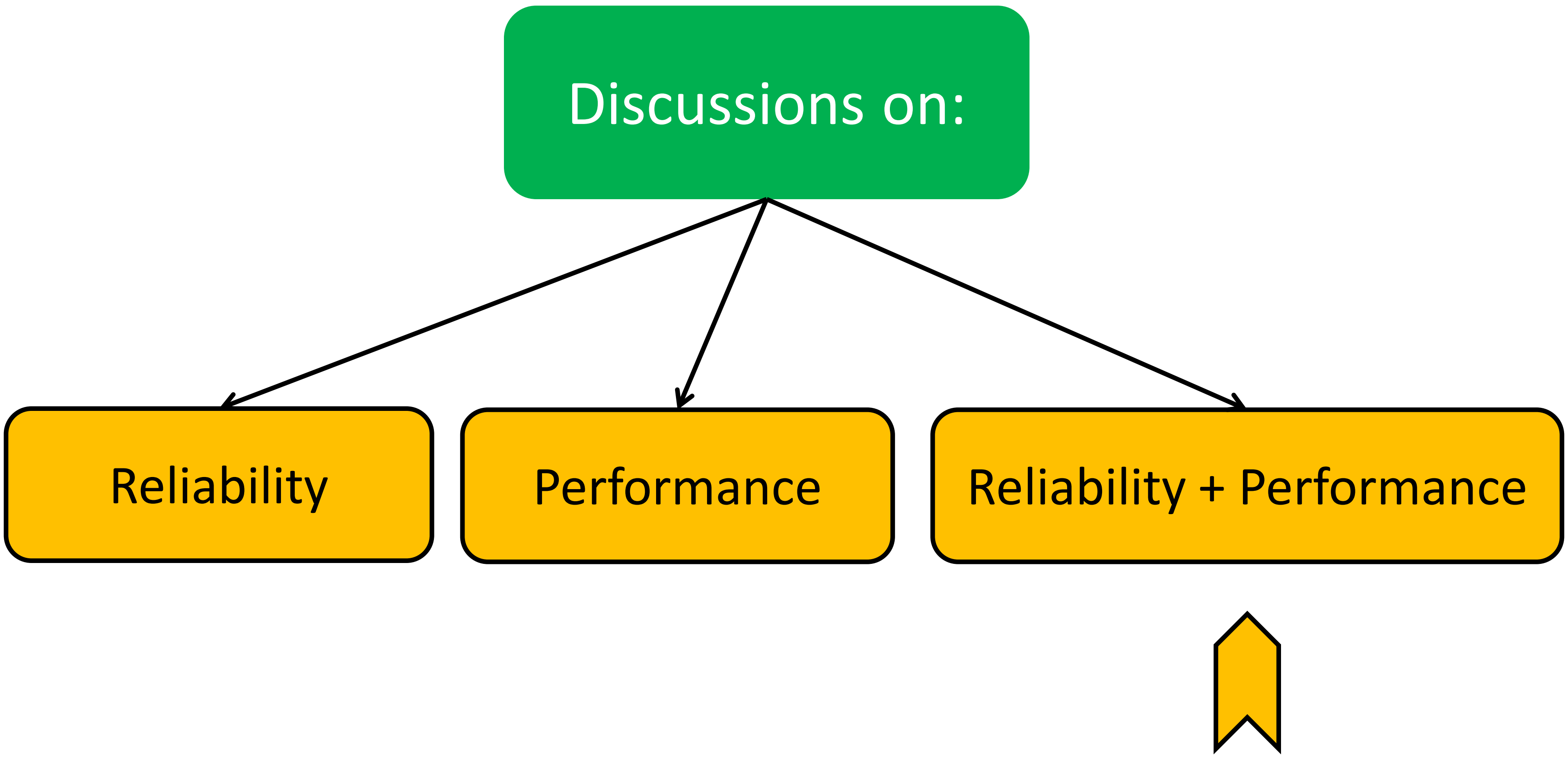


Striping Unit Values: Tradeoffs

- Large striping unit values
  - Lower parallelism
  - Larger amount of data to transfer
  - Decreased seek and rotational delays
  - A request can be handled completely on a separate disk!
  - Number of requests = *Concurrency Factor*



Multiple Disks



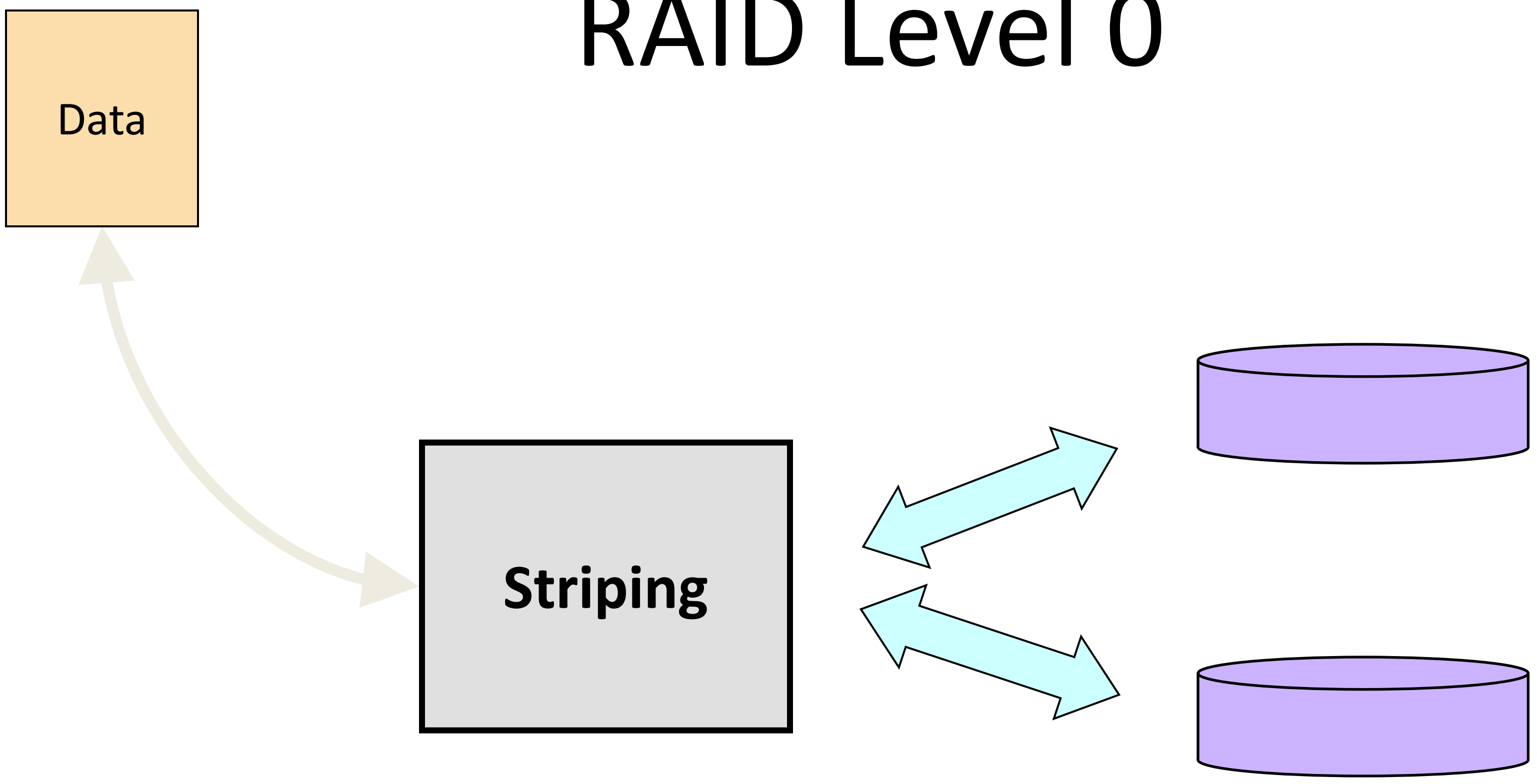
Redundant Arrays of Independent Disks

- A system depending on ***N*** disks is much more likely to fail than one depending on one disk
  - If the probability of one disk to fail is ***f***
  - Then, the probability of N disks to fail is $(1-(1-f)^N)$

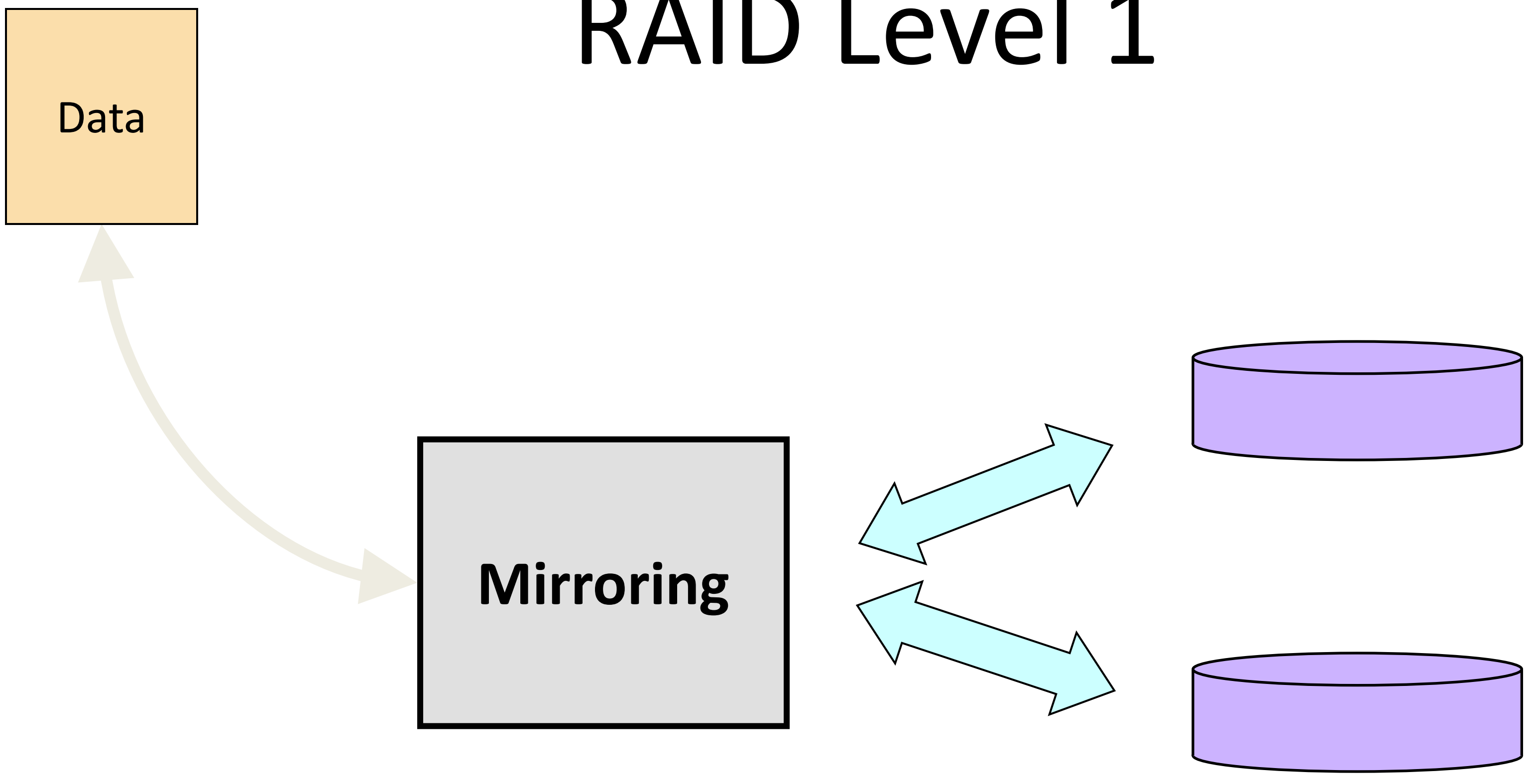
- How would we combine reliability with performance?
  - Redundant Arrays of Inexpensive Disks (**RAID**) *combines* mirroring and striping

Nowadays, Independent!

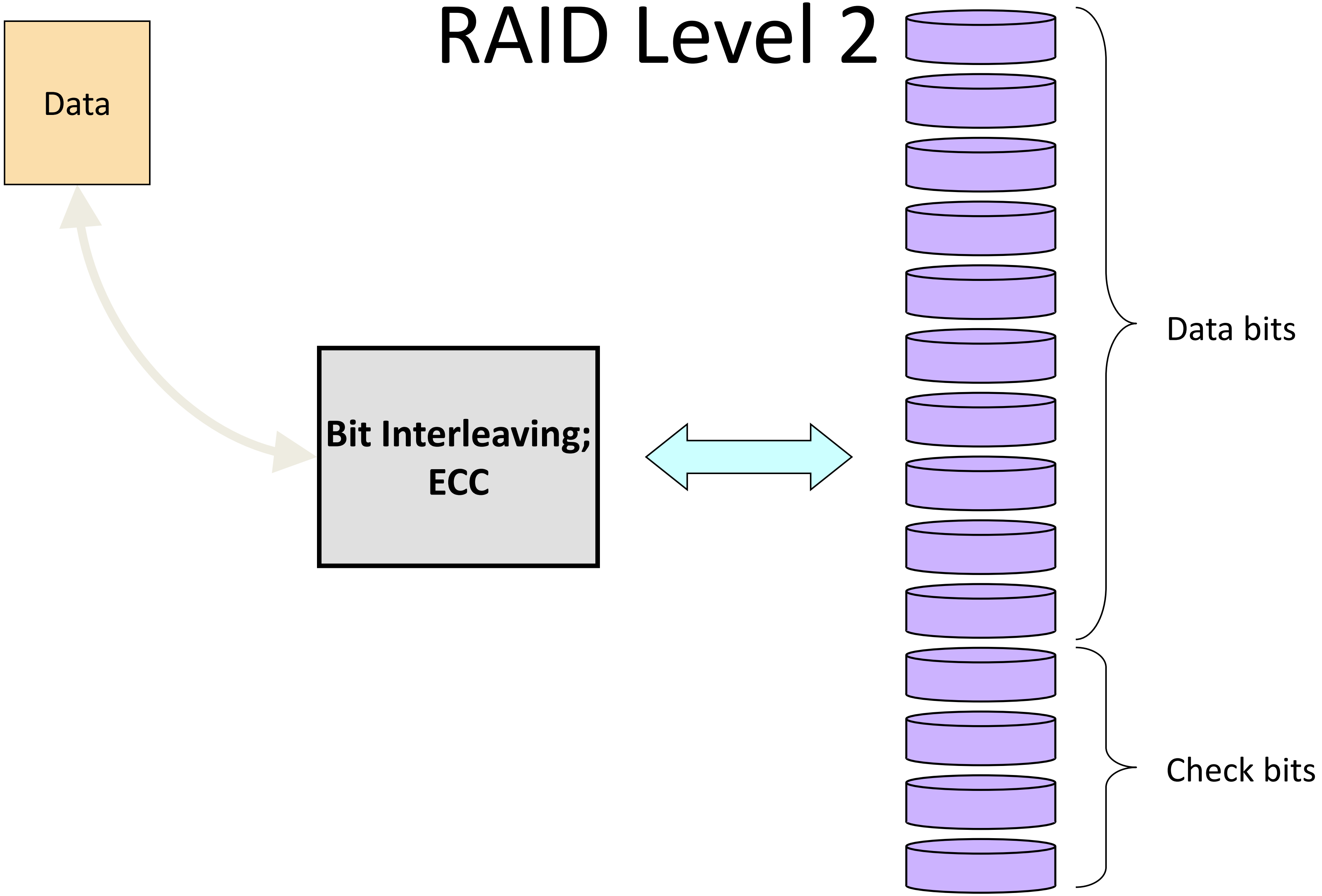
RAID Level 0



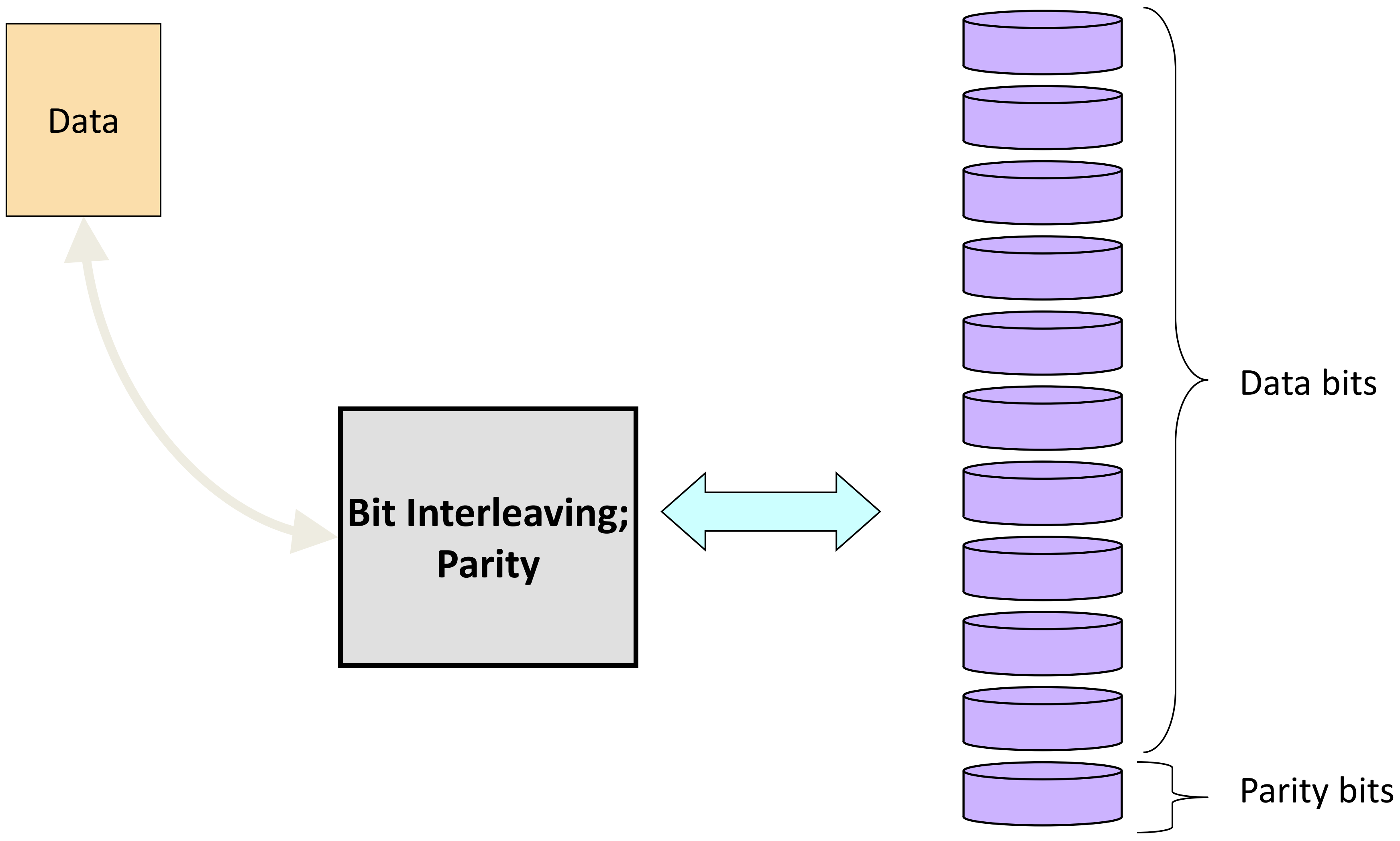
RAID Level 1



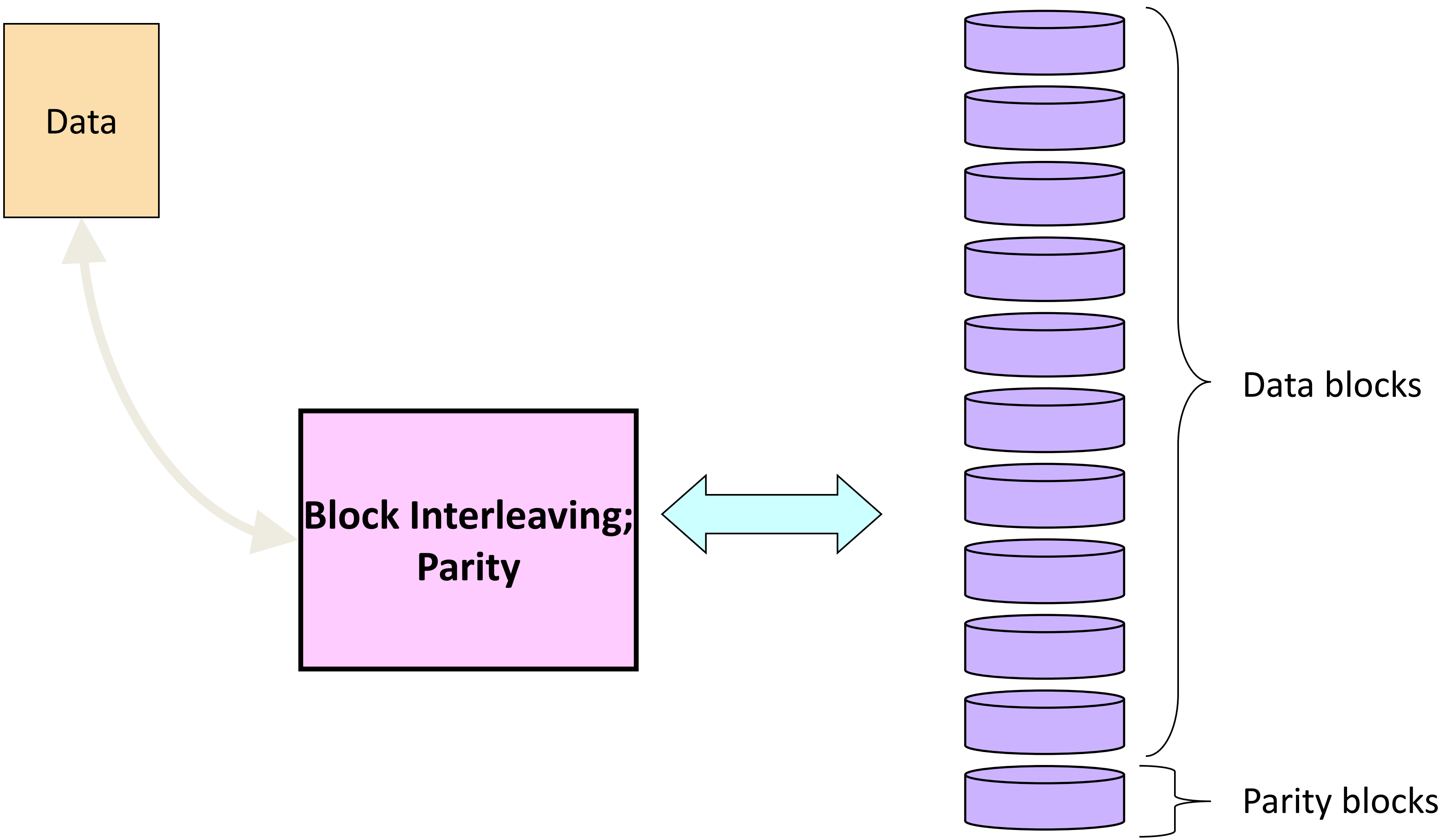
RAID Level 2



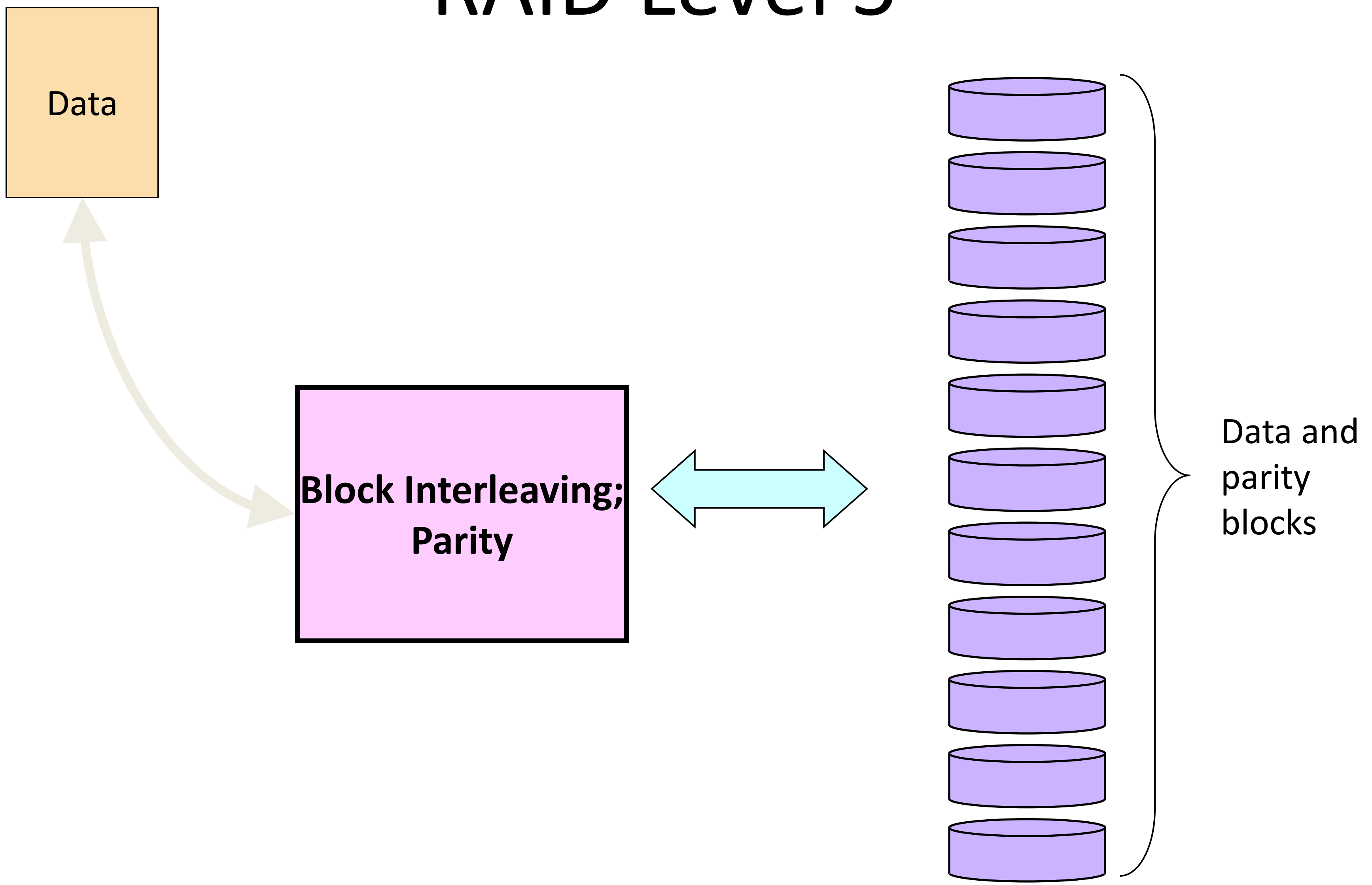
RAID Level 3



RAID Level 4



RAID Level 5



RAID 4 vs. RAID 5

- What if we have a lot of small writes?
  - RAID 5 is the best
- What if we have mostly large writes?
  - Multiples of stripes
  - Either is fine
- What if we want to expand the number of disks?
  - RAID 4: add a disk and re-compute parity
  - RAID 5: add a disk, re-compute parity, and shuffle data blocks among all disks to reestablish the check-block pattern (*expensive!*)

Beyond RAID 5

- RAID 6
  - Like RAID 5, but additional parity
  - Handles two failures
- Cascaded RAID
  - RAID 1+0 (RAID 10)
    - Striping across mirrored drives
  - RAID 0+1
    - Two striped sets, mirroring each other

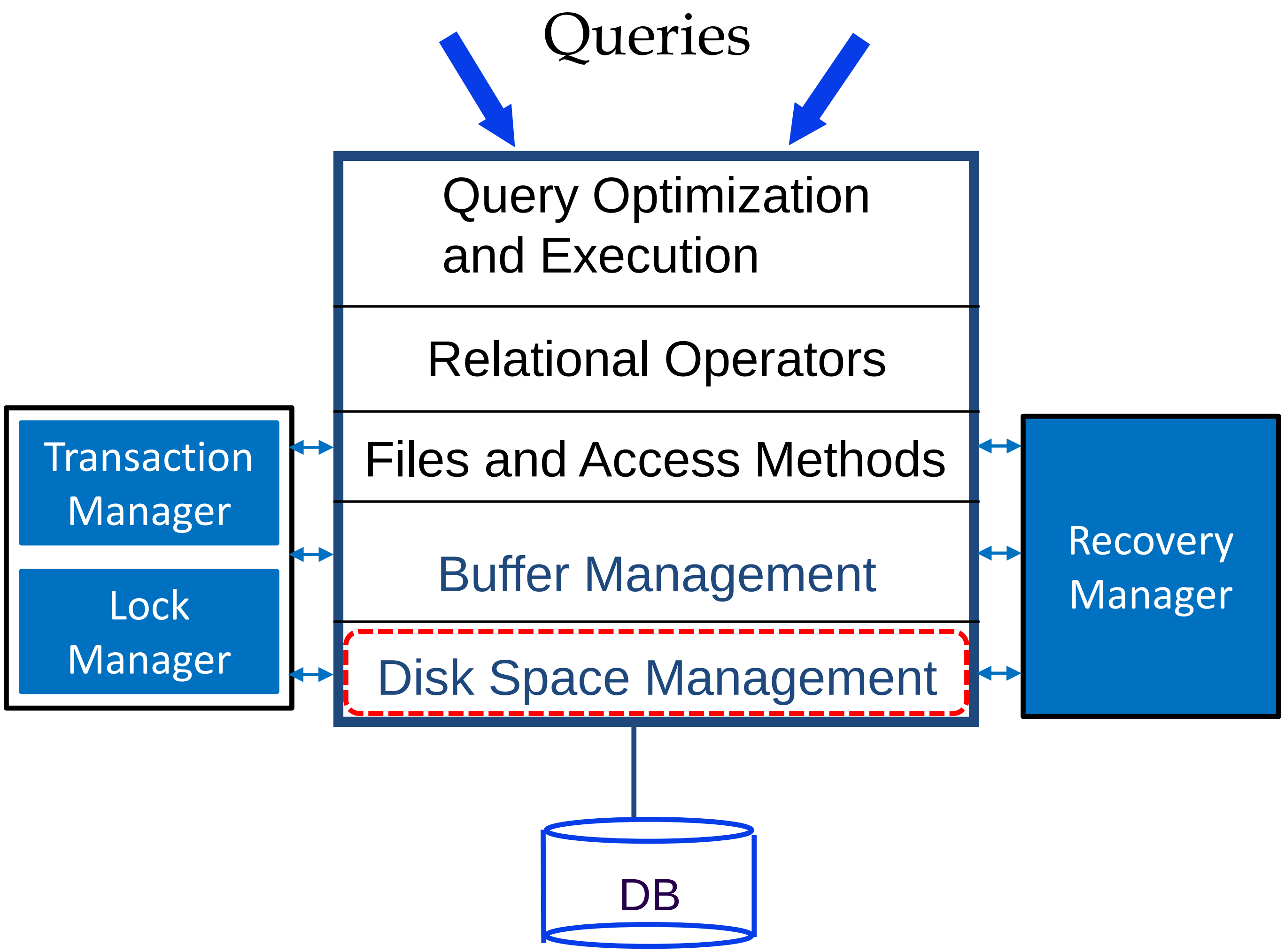
Beyond Disks: Flash

- Flash memory is a relatively new technology providing the functionality needed to hold file systems and DBMSs
  - It is writable
  - It is readable
  - Writing is slower than reading
  - It is non-volatile
  - Faster than disks, but slower than DRAMs
  - Unlike disks, it provides random access
  - Limited lifetime
  - More expensive than disks

Outline

- Where Do DBMSs Store Data?
- Various Disk Organizations and Reliability and Performance Implications on DBMSs
- Disk Space Management ✓
- Buffer Management

DBMS Layers


Queries
Query Optimization
and Execution
Relational Operators
Files and Access Methods
Buffer Management
Disk Space Management
DB
Transaction
Manager
Lock
Manager
Recovery
Manager

Disk Space Management

- DBMSs disk space managers:
  - Support the concept of a page as a unit of data
    - Page size is usually chosen to be equal to the block size
  - Allocate/de-allocate pages as a *contiguous* sequence of blocks on disks
  - Abstracts hardware (and possibly OS) details from higher DBMS levels

Data and Metadata Maintenance

- The DBMS disk space manager keeps track of:
  - Which disk blocks are in use
  - Which pages are on which disk blocks

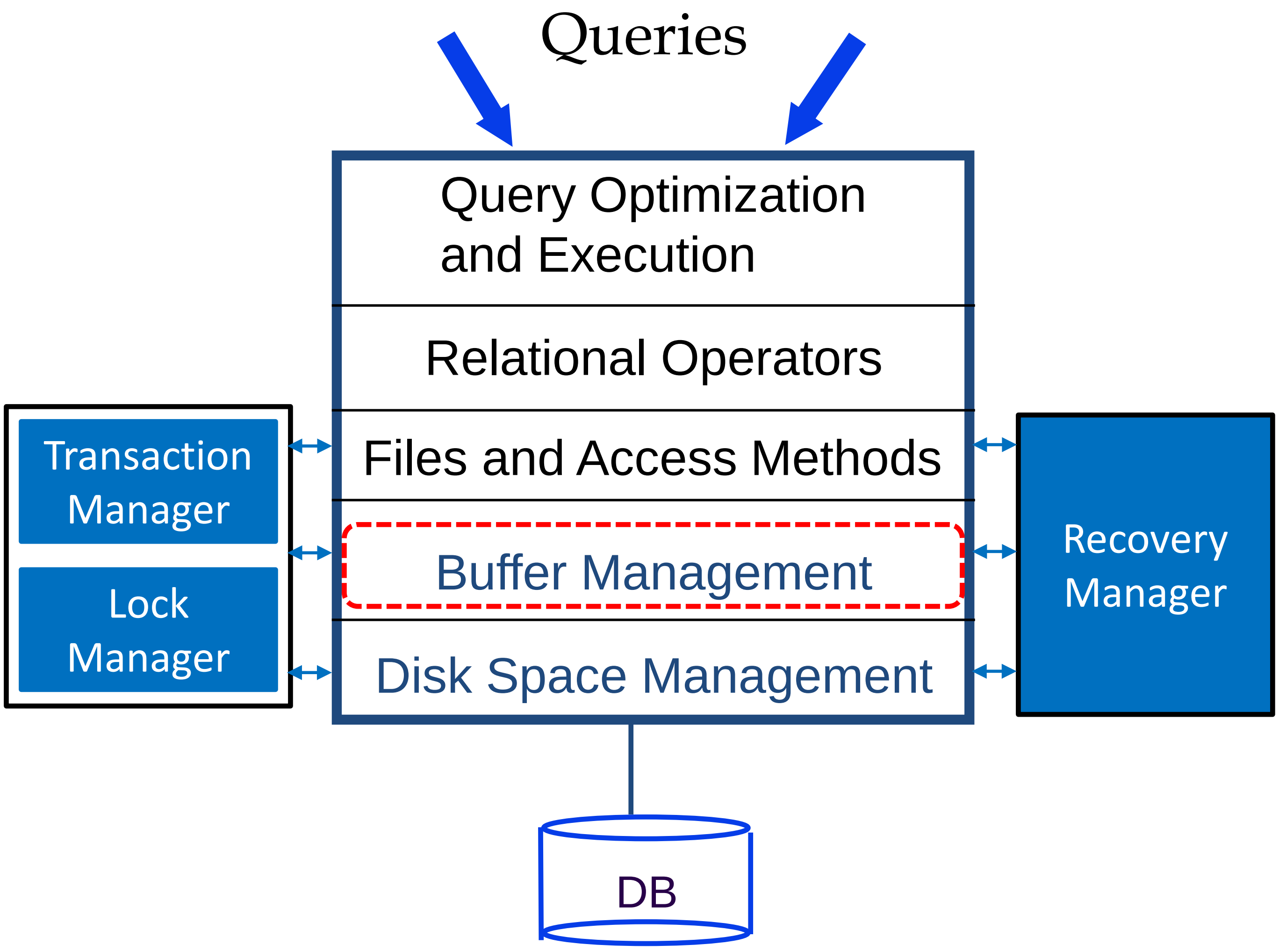
- Blocks can be initially allocated *contiguously*, but allocating and de-allocating blocks usually create *"holes"*

- Hence, a mechanism to keep track of *free blocks* is needed
  - A list of free blocks can be maintained (*storage could be an issue*)
  - Alternatively, a bitmap with one bit per each disk block can be maintained (*more storage efficient and faster in identifying contiguous free areas!*)

OS File Systems vs. DBMS Disk Space Managers

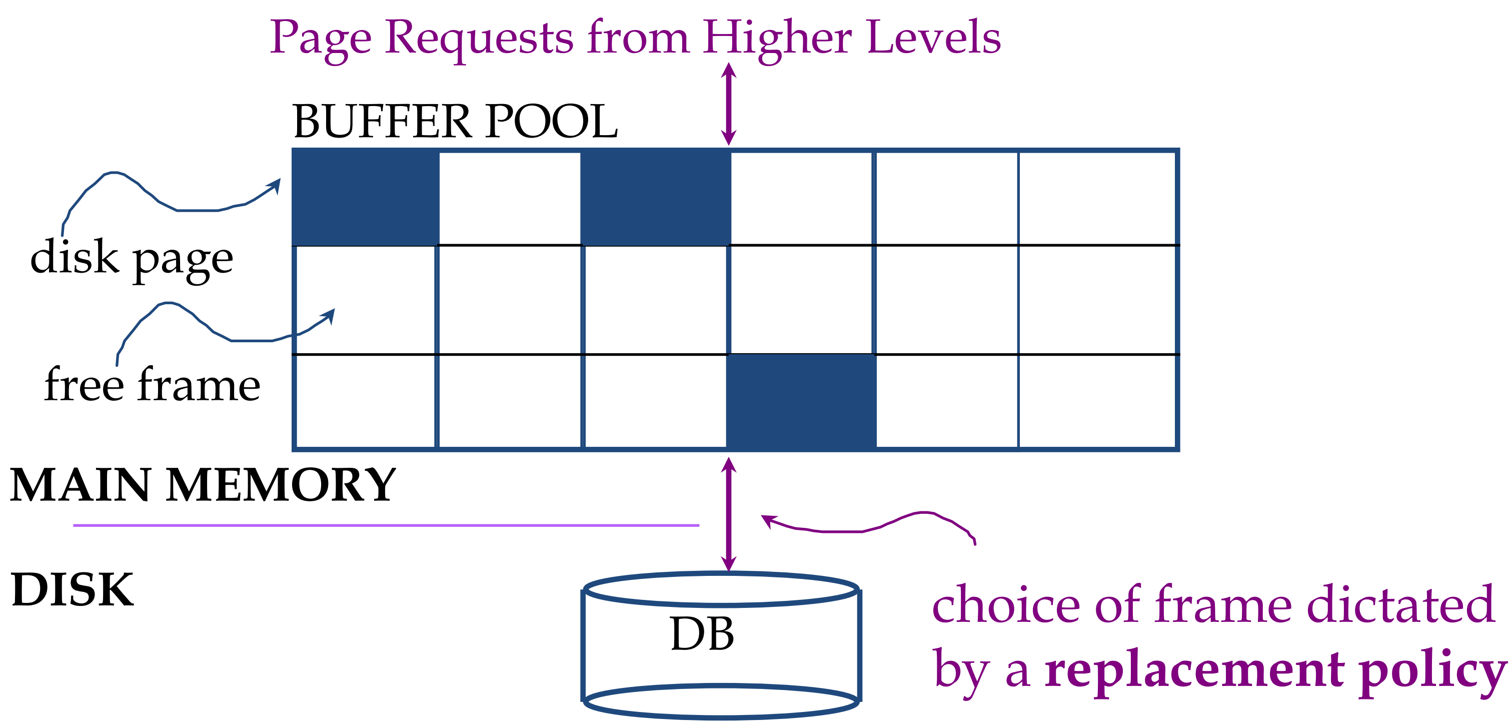
- Operating Systems already employ disk space managers using *their* “file” abstraction
  - “Read byte ***i*** of file ***f***” ➔ “read block ***m*** of track ***t*** of cylinder ***c*** of disk ***d***”
- DBMSs disk space managers usually pursue their own disk management without relying on OS file systems
  - Enables portability
  - Can address larger amounts of data
  - Allows *spanning* and *mirroring*

DBMS Layers


Queries
Query Optimization and Execution
Relational Operators
Files and Access Methods
Buffer Management
Disk Space Management
Transaction Manager
Lock Manager
Recovery Manager
DB

Buffer Management

- What is a DBMS buffer manager?
  - It is the software responsible for bringing pages from disk(s) to RAM as needed
  - It hides the fact that not all data are in the RAM



Satisfying Page Requests

- For each frame in the pool, the DBMS buffer manager maintains two variables:
  - *pin_count*: # of users of a page
  - *dirty*: whether a page has been modified or not

- Upon a *page fault*, the DBMS buffer manager
  - Chooses a frame for *replacement* and increments its pin_count (*a process known as pinning*)
  - If the frame is dirty, writes it back to disk
  - Reads the requested page into the chosen frame

Satisfying Page Requests (Cont'd)

- A frame is not used to store a *new* page until its pin_count becomes 0
  - I.e., until all requestors of the *old* page have unpinned it (*a process known as unpinning*)

- When *many* frames with pin_count = 0 are available, a replacement mechanism is triggered

- If no frame in the pool has pin_count = 0 and a page fault occurs, the buffer manager must *wait* until some page is released!

Replacement Policies

- When a new page is to be placed in the pool, a resident page should be evicted first

- Criterion for an optimum replacement [*Belady, 1966*]:
  - The page that will be accessed the farthest in the future should be the one that is evicted

- Unfortunately, optimum replacement is not implementable!

- Hence, most buffer managers implement a different criterion
  - E.g., the page that was accessed the farthest back in the past is the one that is evicted

Replacement Policies

- When a new page is to be placed in the pool, a resident page should be evicted first

- Criterion for an optimum replacement [*Belady, 1966*]:
  - The page that will be accessed the farthest in the future should be the one that is evicted

- Unfortunately, optimum replacement is not implementable!

- Hence, most buffer managers implement a different criterion
  - *This policy is known as the **Least Recently Used** (**LRU**) policy!*
  - Or: MRU, Clock, FIFO, and Random, among others

The LRU Replacement Policy

- Least Recently Used (LRU):
  - For each page in the buffer pool, keep track of the last time it was *unpinned*
  - Evict the page at the frame which has the *oldest* time

- But, what if a user requires *iterative sequential scans* of data which do not fit in the pool?

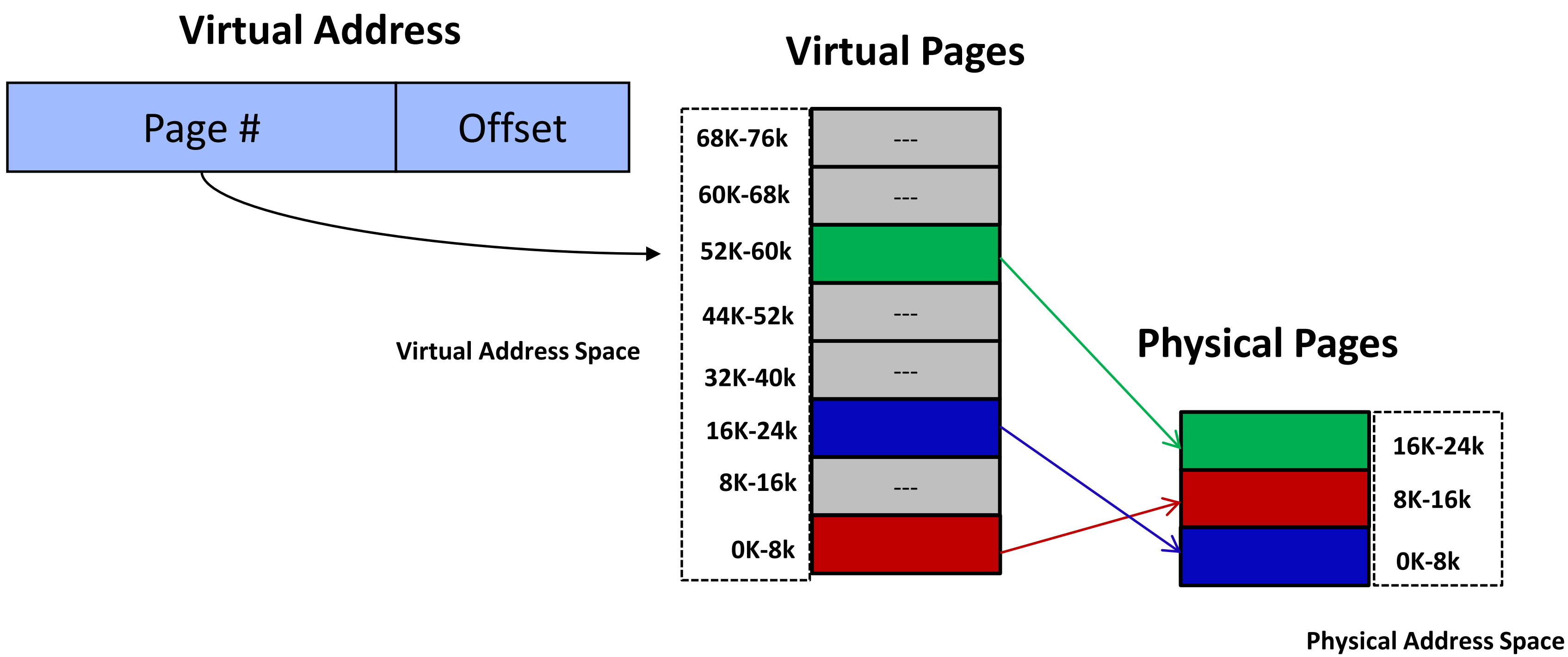
Assume an access pattern of ***A***, ***B***, ***C***, ***A***, ***B***, ***C***, etc.

Access A: ***Page Fault*** | ***Access B:*** ***Page Fault*** | ***Access C:*** ***Page Fault*** | ***Access A:*** ***Page Fault*** | ***Access B:*** ***Page Fault*** | ***Access C:*** ***Page Fault*** | ***Access A:*** ***Page Fault***

This phenomenon is known as "sequential flooding" (*for this, MRU works better!*)

Virtual Memory vs. DBMS Buffer Managers

- Operating Systems already employ a buffer management technique known as virtual memory



Virtual Memory vs. DBMS Buffer Managers

- Nonetheless, DBMSs pursue their own buffer management so that they can:
  - Predict page reference patterns more accurately and applying effective strategies (e.g., *page prefetching* for improving performance)
  - F*orce* pages to disks (needed for the WAL protocol)
    - The OS cannot guarantee this

Concluding Remarks

- DBMSs store data in disks
  - Disks provide large, cheap and non-volatile storage

- I/O time dominates!

- The cost depends on the locations of pages on disk (*among others*)

- It is important to arrange data *sequentially* to minimize *seek* and *rotation* delays

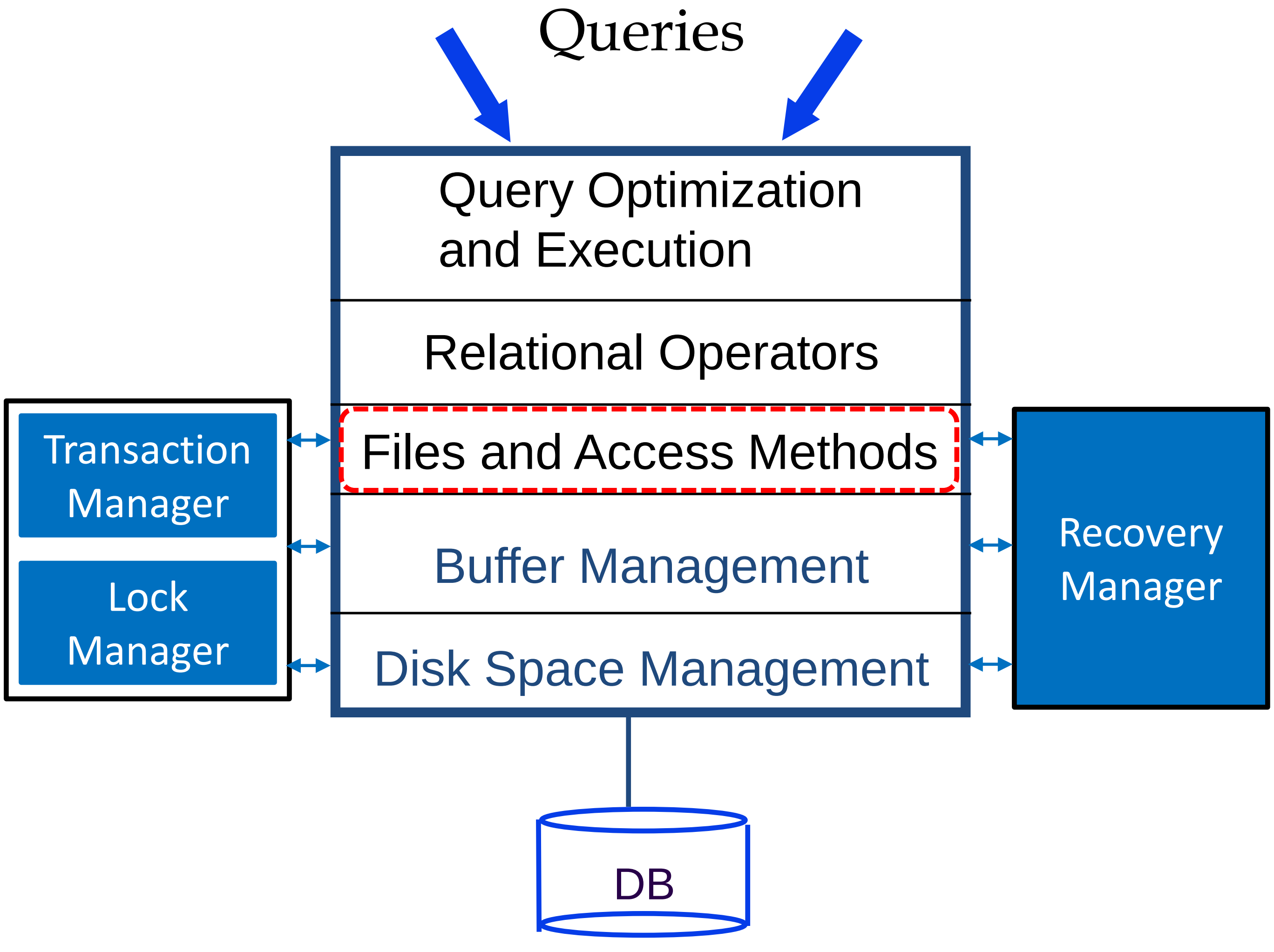
Concluding Remarks

- The lowest layer of the DBMS software which deals with management of space on disk is called disk space manager
  - Higher layers allocate, de-allocate, read and write pages through (routines provided by) this layer

- However, data must be in memory for DBMSs to operate on

- The buffer manager sits on top of the disk space manager and brings pages in from disks to RAM as needed in response to read/write requests

Next Class



Queries
Query Optimization
and Execution
Relational Operators
Files and Access Methods
Buffer Management
Disk Space Management
Transaction
Manager
Lock
Manager
Recovery
Manager
DB