General Hardware and C++ Intricacies (Recap)

Hardware Aspects:

- **Storage Hierarchy** : Computer systems employ a hierarchy of storage devices based on speed, cost, and volatility5.
- **Volatile Storage** : Cache (fastest, smallest, volatile), Main Memory (DRAM, fast, larger, volatile)5. Data here is lost without power3.
- **Persistent (Non-Volatile) Storage** : Flash Memory (SSDs, faster than HDDs, durable), Magnetic Disks (HDDs, slower, very large capacity, durable), Optical Disks, Magnetic Tapes (slowest, highest capacity, durable for archival)5136. Data here is retained even without power3.
- **Disk Interfaces** : Standards like SATA (6 gigabits/sec), SAS (12 gigabits/sec), and NVMe (up to 24 gigabits/sec via PCIe) define how disks connect to systems, affecting data transfer rates137.
- **Magnetic Disks** : Composed of platters, read-write heads on arms. Operations involve seek time (moving arm) and rotational latency (waiting for sector)6138. Disk controllers manage low-level operations, checksums, and bad sector remapping139.
- **Flash Storage (SSDs)** : Based on NAND flash. Pages are read (20-100 microseconds) and written once before needing an erase (2-5 milliseconds for erase block). Logical-to-physical page remapping (flash translation table) hides erase latency140141. SSDs have high random read/write IOPS and high sequential transfer rates142. Wear leveling prolongs lifespan by distributing erases142.
- **Storage Class Memory (SCM)** : New technologies like Intel Optane, aiming for speeds comparable to main memory while being non-volatile, potentially blurring the lines between DRAM and persistent storage143.
- **RAID (Redundant Arrays of Independent Disks)** :
- Combines multiple disks to appear as a single, highly reliable, high-capacity, and high-speed unit144.
- **Reliability** : Achieved through redundancy, such as mirroring (RAID 1) or parity blocks (RAID 5, 6)145146. This protects against data loss when individual disks fail144.
- **Performance** : Achieved through parallelism (load balancing small accesses, parallelizing large accesses) and striping data across multiple disks (block-level striping)147.
- **RAID Levels** : Vary in cost, performance characteristics, and fault tolerance (e.g., RAID 0 for speed, RAID 1 for writes and mirroring, RAID 5 for distributed parity, RAID 6 for dual parity against two failures)146....
- **Hardware RAID** : Dedicated hardware controllers often use non-volatile RAM (NV-RAM) to record writes, which is critical for quick recovery from power failures and detecting corrupted blocks151152.

- **Disk Block Access Optimization** : Database systems minimize disk-block transfers by buffering frequently used blocks in memory, employing read-ahead strategies, and using disk-arm-scheduling algorithms (e.g., elevator algorithm)153154.
- **Buffer Manager and Disk Writes** : Buffer managers can optimize disk writes (e.g., reordering) but must ensure data consistency, often by forcing output of blocks for recovery or using journaling file systems that write data in-order to a log disk155....

C++ Code Intricacies:

- **Classes and Objects** : Used to encapsulate data (`attributes`) and behavior (`methods`) (e.g., `Tuple` class, `Book` class)158.
- **Encapsulation** : Using `public` , `private` , `protected` access specifiers to control attribute visibility and protect internal state159.
- **C++ Standard Library Containers** :
- `std::vector` : Dynamic arrays for storing collections of objects (e.g., `table` in `BuzzDB` or `fields` in `Tuple`)160161.
- `std::map` (ordered) and `std::unordered_map` (unordered, hash-based): Associative containers that map keys to values, used for indexing (e.g., `index` in `BuzzDB`)80....
- **Memory Management** :
- **Raw Pointers (** `**new**` / `**delete**` **)** : Require manual allocation and deallocation. Prone to memory leaks (forgetting `delete`), dangling pointers (accessing freed memory), and double-free errors (`delete` twice)163....
- **Smart Pointers (** `**std::unique_ptr**` **, ** `**std::shared_ptr**` **)** : RAII-compliant solutions that automate memory management, preventing common raw pointer issues167.... `std::unique_ptr` provides exclusive ownership, `std::shared_ptr` enables shared ownership via reference counting116....
- **Heap vs. Stack** : The heap is for dynamic memory allocation (longer lifetime, flexible size), managed by `new` / `delete` or smart pointers170171. The stack is for local variables (automatic allocation/deallocation, limited to scope)170172.
- **Constructors and Destructors** : Special member functions. Constructors initialize objects (`Field(int i)`), and destructors clean up resources (e.g., `Field::~Field()` to `delete[] char*` for strings)163....
- **Copy Semantics** : How objects are copied, assigned, and passed by value. "Deep copying" (copying the underlying data, not just pointers) is crucial for objects with dynamically allocated memory to avoid shallow copy issues174175. `std::unique_ptr` cannot be copied but can be moved (`std::move`) to transfer ownership170175.
- **File I/O (** `**fstream**` **, ** `**ifstream**` **, ** `**ofstream**` **)** : Standard C++ libraries for reading from and writing to files176.... `file.flush()` explicitly writes buffered data to disk, important for durability179.... Error checking (`if (!inputFile)`) is vital182.
- **RAII (Resource Acquisition Is Initialization)** : A programming idiom where resource

management (acquisition and release) is tied to the lifetime of objects. Examples include `std::lock_guard` for mutexes, `std::unique_ptr` for memory, and the `StorageManager` handling file streams5152.

- **Templates** : Enable generic programming by allowing functions and classes to operate on different data types without rewriting code, essential for components like `BPlusTree<Key, Value>` 131132.
- **STL Algorithms (** `**std::lower_bound**` **, ** `**std::find_if**` **, ** `**std::move**` **, etc.)** : Provide efficient and robust implementations for common operations on containers, heavily used in data structure implementations118119.
- **Lambda Functions** : Anonymous functions often used for concise custom comparators with STL algorithms118120.